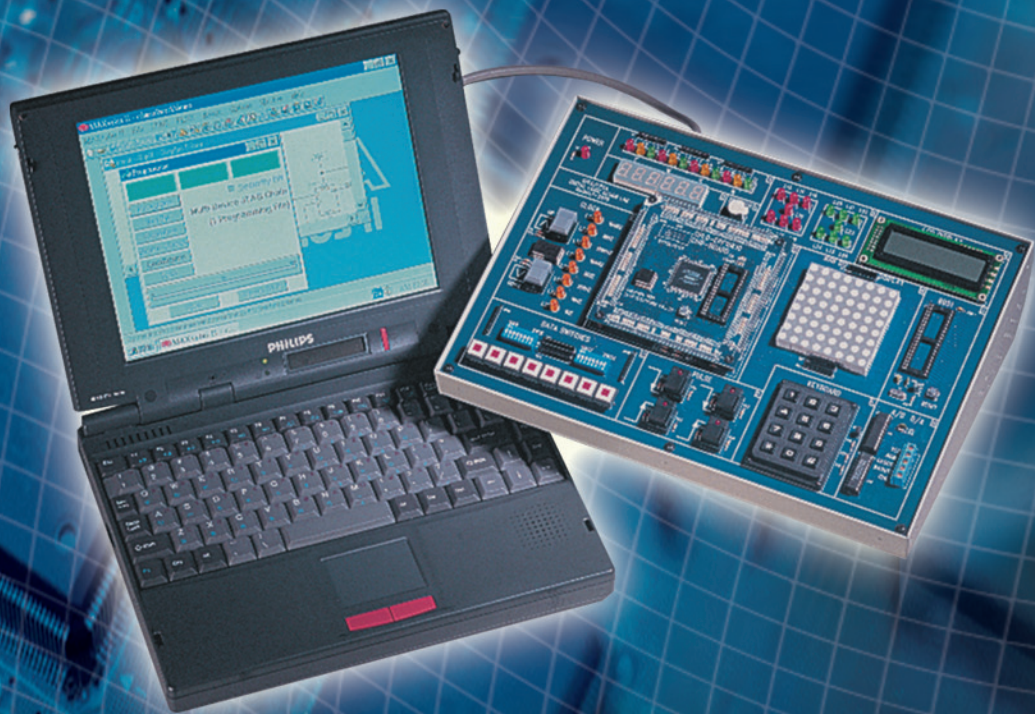


CPLD Logic Design and Practices



LEAP ELECTRONIC CO., LTD.



The information in this document is subject to change without notice

LEAP software products are copyrighted by and shall remain the property of LEAP ELECTRONIC CO., LTD. Use duplication, or disclosure is subject to restrictions as stated in LEAP's applicable software license.

LEAP ELECTRONIC makes no warranty of any kind with regards to this material, including, but not limited to, implied warranties of merchantability and fitness for a particular purpose, except as stated in Leap's applicable software license.

No part of this document may be copied or reproduced in any form without the prior written consent of LEAP ELECTRONIC CO., LTD.

Copyright © 2000 LEAP ELECTRONIC CO., LTD.

All Rights Reserved.

First Edition, November 2000.

By LEAP ELECTRONIC CO., LTD.

CPLD Logic Design and Practices is a trademark of LEAP ELECTRONIC CO., LTD. Other brands and product names are trademarks or registered trademarks of their respective holders.



Contents

Chapter 1. Introduction to Digital Logic

- 1.1 Introduction to Digital Logic
- 1.2 Integrated Digital Logic Design Environment
- 1.3 Programmable Logic Device – PLD
- 1.4 PC Aided Digital Logic Design
- 1.5 Experimental Platform
- 1.6 Evaluation and Test

Chapter 2. Numerical System

- 2.1 Numeric Expressions
- 2.2 Numerical System Conversion
- 2.3 Numerical Complement
- 2.4 Negative Binary Number Expression
- 2.5 Binary Arithmetic Operations
- 2.6 Binary-coded Decimal (BCD) Code
- 2.7 Review

Chapter 3. Basic Logic Theories

- 3.1 Boolean Algebra
- 3.2 Boolean Algebra Simplification
- 3.3 Logic Gate
- 3.4 Applications of Logic Gate
- 3.5 Practices
- 3.6 Review

Chapter 4. A New Design Methodology

- 4.1 MAX+PLUS II Baseline Setup and Start
- 4.2 How to Use Mouse
- 4.3 Graphic Entry
- 4.4 Functional Simulation



- 4.5 Floorplan and Design Compilation
- 4.6 Device Programming and Circuit Testing
- 4.7 Use Graphic Entry to Complete LEDTEST Example
- 4.8 Review

Chapter 5. Combinational Logic Circuit

- 5.1 The Design, Simulation and Test of General Combinational Logic Circuit
- 5.2 The Design, Simulation and Test of Adder
- 5.3 The Design, Simulation and Test of Subtractor
- 5.4 The Design, Simulation and Test of Comparator
- 5.5 The Design, Simulation and Test of Encoder
- 5.6 The Design, Simulation and Test of Decoder
- 5.7 The Design, Simulation and Test of MUX
- 5.8 The Design, Simulation and Test of DMUX
- 5.9 The Question of Hazards
- 5.10 Evaluations

Chapter 6. Sequential Logic Circuit

- 6.1 Basic Concept of Sequential Logic Circuit
- 6.2 The Design, Simulation and Test of Synchronous Counter
- 6.3 The Design, Simulation and Test of Synchronous Shift Register
- 6.4 The Design, Simulation and Test of Synchronous Shift Count Register
- 6.5 The Design, Simulation and Test of Asynchronous Counter
- 6.6 Evaluation

Chapter 7. SIMPLE DESIGN EXAMPLES

- 7.1 Frequency Generator
- 7.2 Simple Electronic Dice
- 7.3 Timer
- 7.4 Simple Traffic Light Controller



- 7.5 Dot Matrix Displayer Test Circuit
- 7.6 Keyboard Scan and Display Scan Circuit
- 7.7 LCD Interface Circuit
- 7.8 Evaluations

Chapter 8. Connecting with Analog Circuit

- 8.1 A/D Converter – ADC0804
- 8.2 D/A Converter – AD7528
- 8.3 Single Chip – 8051
- 8.4 Design Example – Connecting with ADC0804, AD7528, and 8951
- 8.5 Evaluation

Chapter 9. CPLD LOGIC DESIGN LAB PLATFORM LP-2900

- 9.1 Function Description to LP-2900
- 9.2 Setting up LP-2900
- 9.3 The Architecture and Circuits of LP-2900
- 9.4 Pin arrangement of LP-2900
- 9.5 Evaluations

Appendix A. PLD Suppliers and Main Products

- A.1 PLD Suppliers and Main Products
- A.2 ALTERA's CPLD Devices

Appendix B. The Built-in Resources of MAX+PLUS II

- B.1 Primitives
- B.2 Macrofunctions

CHAPTER 1

Introduction to Digital Logic





1.1 Introduction to Digital Logic

In general, all physical quantities in nature such as temperature, humidity, length, speed, and time are continuously changing. We call these continuously changing signals as “analog” signals. In contrast to analog, there is a discrete signal called “digital” signals. A signal processing system with use of analog signals is called “analog system”. Signals in this kind of system continuously fluctuate over time between high and low voltages. For example, signals are changing from -10 voltage to $+10$ voltage. Similarly, a signal processing system with use of digital signals is called “digital system”, and signals in this system can only be considered either on or off, high or low values. For example, signals can change either 0 voltage or $+5$ voltage.

From above, we could know there are two systems to process signals. One is analog system and the other is digital system. Generally, a digital system comes with more benefits than analog. It is programmable, faster, precise, and flexible. Besides, as the signals are discrete but not continuous, a digital system can be less affected by the changes of elements’ natures such as the problem of a worn-out transistor. Those are the reasons why digital systems are greatly adopted in the world.

For sure, all physical quantities in nature are shown in the form of analog signals. They are measurable, visible and controllable. If we would like to take the advantages of digital systems, we have to transform signals from analog to digital and process by digital systems. The transforming process is as below:



Sensors can transform the natural physical quantities into electronic signals such as voltage or current signals that are still analog signals. Electronic signals will then been transformed into digital signals by Analog/Digital Converter. Actually, Analog/Digital Converter is responsible for sampling and quantization. Sampling is to take the value of a physical quantity at one point and then mark it as a digital number, and this is what we called “Quantization”. Processed digital signals will be

transformed into analog signals in some circumstances to have better control on some objectives such as flow control gates, fans, and heaters etc. The transforming process is as below:

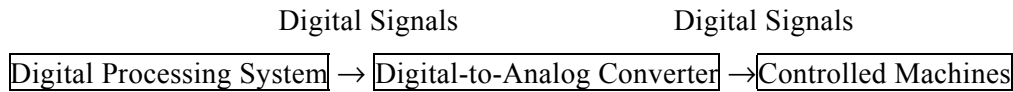


Figure 1.1, it is the flow control diagram for a chemical factory. A/D Converter, Digital Processing System, and D/A Converter are the key points in Figure 1.1 as well as in this book. To introduce digital processing systems, we will discuss the topic from the basic theory first, including Numeric System (Chapter 2), Boolean algebra (Chapter 3), and then Basic Gate (Chapter 3), combinational logic (Chapter 5) as well as Sequential Circuit (Chapter 6). More over, we will talk about A/D Converter and D/A Converter in Chapter 8. In the rest of the Chapter 2, we will introduce a new logic design environment called “Integrated Digital Logic Design Environment”. **This integrated environment will greatly give us logic design and simulation.** In Chapter 4, we will further discuss how to set up and use the EDA tool, MAX+PLUS II.

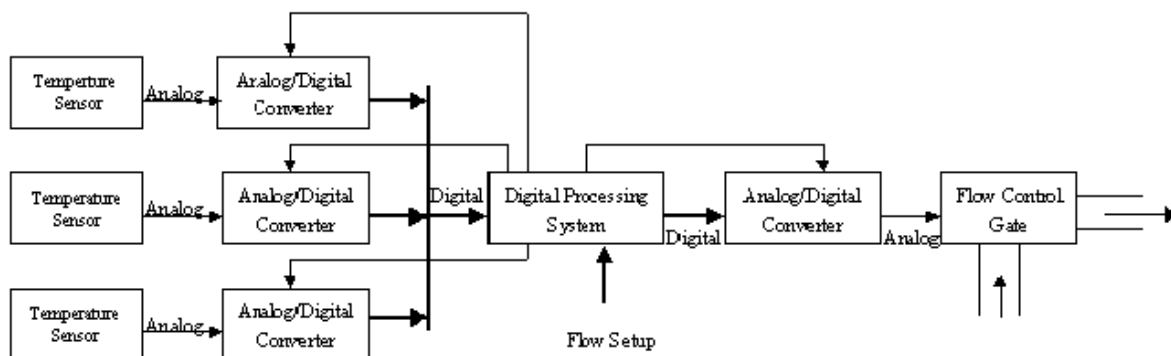


Figure 1.1 A Flow Control Diagram for a Chemical Factory

1.2 Integrated Digital Logic Design

Environment

Currently, engineers can use two different chips to make PCB (Printed Circuit Board) when they design circuits. One is standard/discrete logic such as TTL/COMS 74/54 family; the other is ASIC (Application-Specific Integrated Circuits). ASIC shown in Figure 1.2 can be divided into 4 different catalogs: PLD (Programmable Logic Devices), Gate Array, Cell-Based IC, and Full-Custom IC.

Standard logic is TTL/CMOS (e.g.: 74/54 family) that comes with a specific function. This is the first device to make digital logic circuits and digital systems in history. As the needs for digital circuits are more and more complicated and the new technologies are developed faster than before, standard circuits gradually cannot satisfy customers, and, as a result, a new market is there for Gate Array, Cell-Based IC, and Full-Custom IC. All the devices use different ways to design circuits and different process to produce circuits. For a programmable logic device (PLD), it is an integrated circuit that has user-configuration functions, including Boolean expression or registered function etc, to customize the circuits and meet customer's needs. Those functions make PLDs quite different than TTL/CMOS standard devices. As long as the elements have the user-configuration functions, they are PLD elements, including 3 different catalogs: 1.) PAL/GAL with simple functions, low capacity, and low pin counts, 2.) FPGA (Field-Programmable Gate Array) with high capacity, higher pin counts, and 3.) CPLD (Complex PLD) with high capacity and high pin counts. Those different PLDs have their own different structures and internal-memory-type design technologies. Depended on requirements, we could choose the right devices. PAL is a simple PLD, which has gate counts from 100 to 1000 and IC pin counts within 28-pin. It is an old process with bipolar procedure, can only write data once, and cannot delete old data. Its advantage is faster speed, but its disadvantage is more electrical consumption. GAL, another simple PLD, uses CMOS as its process. It can rewrite data and has less electrical consumption. However, its operating speed is slower than PAL relatively.

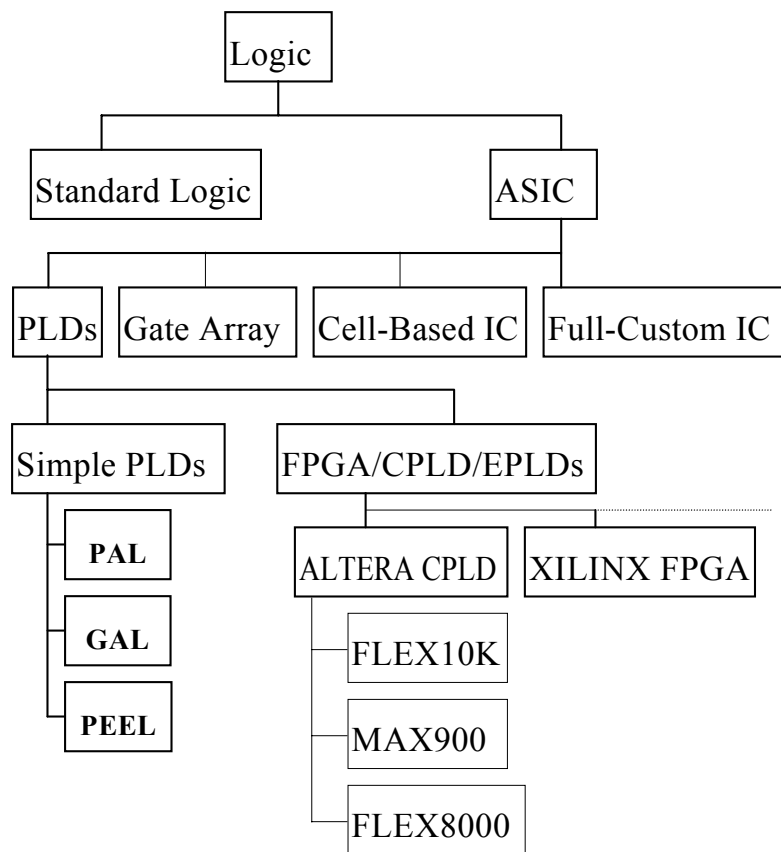


Figure1.2 Catalogs of logic devices

FPGA/CPLD process is CMOS. Currently there are five techniques to produce FPGA/CPLD: EPROM, EEPROM, FLASH, SRAM and Anti-Fuse. FPGA has higher density. Compared with CPLD, it uses less logic gates and focuses on registers. However, its routing is complicated, causing the problems of varying and longer timing delay. For new comers or students, they definitely think FPGA is more difficult and have to study harder and longer to understand it. Currently, XILINX, Actel, Atmel and AT&T are the key suppliers, and, among them, XILINX has greater market share and is the biggest FPGA supplier in the market. For CPLD, it can multi-erase data, program data, and fixed delay. It also allows users to apply and design easily. The main suppliers of CPLD include AMD, ALTERA, Lattice, Cypress, and ICT etc. Among them, AMD and ALTERA are the two key players in the market.

As the adoption of higher density and new process of PLD, the prices are gradually decreasing, and, as a result, PLD has given consumers higher-density, more effective,



and cheaper devices than standard logic. In another words, we could put all TTL/COMS standard circuits from a big board to a small piece of CPLD. It reduces room for a big board and time for welding process, and speed the circuit up significantly. Therefore, PLD has greatly been seen as a “super star”.

Table 1.1 is the comparison for PLD, Standard/Discrete Logic, and Full-Custom IC. Clearly, PLD has the great advances in speed, density, price, developing time, modifiability, prototyping, development tools, integrity, and time to market.

Table1.1 Comparison of logic devices

Requirements	PLD	Discrete Device	ASIC (Full-Custom IC)
Speed	Very Fast	Slow	Very Fast
Density	High	Low	Very High
Price	Cheap	Expensive	Very Cheap
Developing	Short	Proper	Long
Modifiabili	Very Flexible	Proper	Very Inflexible
Prototyping	Very Easy	Very	Very Difficult
Tools	Easy Access and	Not	Not Expensive
Integrity	Very Small	Very Big	Very Small
Time to	Short	Proper	Long

1.2.1 Techniques of Traditional Digital Circuit Design

As we mentioned before, Standard/Discrete Logic or ASIC can be used to make circuit board. The function of Full-Custom IC is fully user-defined, but Gate Array is a semi-product and therefore its function is defined based on this. For Cell-Based IC,



its function is to use well-defined cells in a cell base to complete circuit design. Usually, when engineers use these kinds of ASIC chips or even add with some of standard logic to design circuit boards, they are making PCB (Printed Circuit Board) samples directly. In another words, after completing circuit designs by computers, we will assign a third party to produce and weld all elements, or only ask a third party to weld SMD (Surface Mounted Device) with high pin counts and take the rest of the production procedures by ourselves. We might add/decrease elements to/from sample version 1, and have line jumping or line cutting and testing to have more correct sample boards. After completing the testing for sample version 1, we then assign a third party to produce sample version 2, and back to test it again. To have finalized circuits, we might have sample version 3 or version 4 if needed. If all correction has been done and all the requirements are met, the circuit board development is completed. We could start mass production in next. These kinds of circuits are mainly for digital system circuit designs. Currently, most of R&D in high tech industries uses the ways to design and produce new circuits.

The traditional ways of the design of digital logic circuit for standard logic, which is greatly accepted by the public, is as follow:

- Design circuits on the papers.
- Weld or wire all elements by practice board.
- Test circuit board by the tools such as multifunctional meter, logic probe, oscillator display, and functional generator.
- Correct circuit designs and circuit boards by adding or decreasing elements and by line jumping and line cutting testing.
- Assign a third party to produce circuit sample boards; after testing and correcting sample boards, start mass production.

Designing circuits on papers is very ineffective and inefficient. It is not easy to edit and make a change on papers. It is also unattractive, time consuming, and not easy to store. Except engineers having great experience in circuit designs, without scientific verification, there will definitely have a great of mistakes in designs. Moreover, workers use practice board to weld or wire all elements. It is going to make the job more complicated, time consuming, and defective. In order to weld all the elements, the size of a board used for standard logic very big. It is hard to decrease the board size, and, thereafter, against the market trends requiring light and small circuit boards.



As Standard logic is not going to be on production lines, we could predict that the way to design circuits will not be accepted by the market soon. It is a fact. It is the trend in the market.

1.2.2 Integrated Digital Logic Design Environment

Because PCs functions are more and more, memory size is larger and the prices are lower than before, the world PC users are increasing significantly. As a result, electronic design automatic (EDA) is greatly available in the market. The EDA software can be used in PC platform. EDA software offers the functions including graphics, texts, and waveform entry. It offers a great user-friendly environment to modify designs and manage files and we called this is the scientific ways to manage files. EDA software also gives users the environment to scientifically verify circuits: functional simulation and timing simulation. With the two simulations, new comers or less experiencing engineers almost can precisely complete the circuit designs.

With the increase of the high-density requirement and the adoption of new process, PLD prices are decreasing gradually. Compared with standard logic, PLD, consequently, can give users better devices with higher density, more efficiency, and cheaper prices. This allows us to use programmable CPLD devices with SRAM technology to design an experimental platform. Except SRAM CPLD devices, this platform also has power, PC downloading interface, and I/O elements such as LED, seven segments display, buzzer, clocks, switches, pulse switches, 4×3 keyboard, 8×8 dot matrix display, LCD display, and A/D & D/A circuit modules. The main purpose of the platform is to offer a simple and accessible environment to verify circuits and to reduce the time of circuit design. To perform the process, we have to download the circuits to the SRAM CPLD devices in the platform first, and then give required clock or input signals, inspect the results to ensure if the circuits meet specifications.

To integrate PCs, EDA software, and experimental platform, we will briefly introduce the general flow of integrated digital logic design in Figure 1.3. We use EDA software on PC to make design entry and to simulate circuits. To verify the designed

circuits, we need to download “Configuration Bits” to SRAM CPLD devices in platform by the cables connecting between PC and the platform. If there are still defects after testing, we go back to the previous steps: modifying the designs, re-doing the simulation, downloading the circuits and re-testing the circuits until correct. After circuits are finalized, we then could produce the circuits.

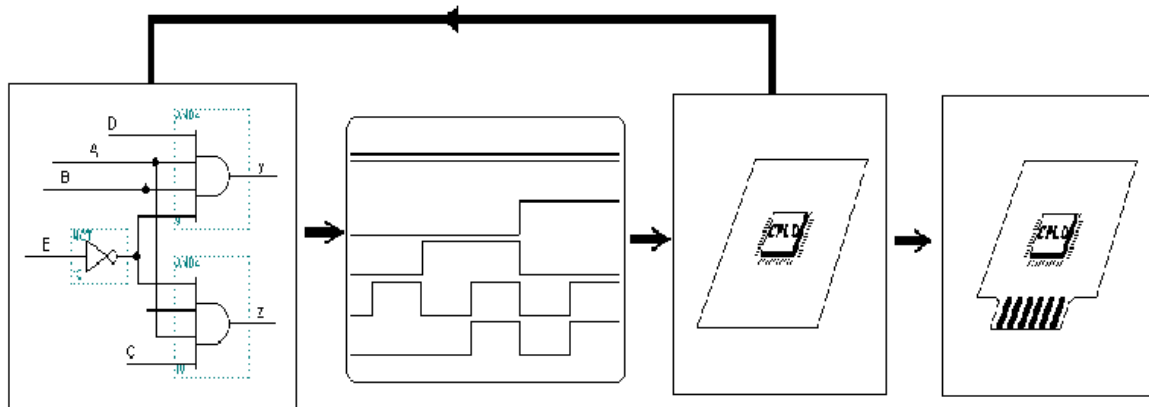


Figure 1.3 General flow of integrated digital logic design

Figure 1.4 details the flow of integrated digital logic circuit design. For this flow, we will briefly discuss in Section 1.4, and have further discussion with real examples in Chapter 4. Table 1.2 to 1.5, we will compare the differences between traditional and integrated logic circuit designs in terms of tools, elements, circuit design flow, and study requirements.

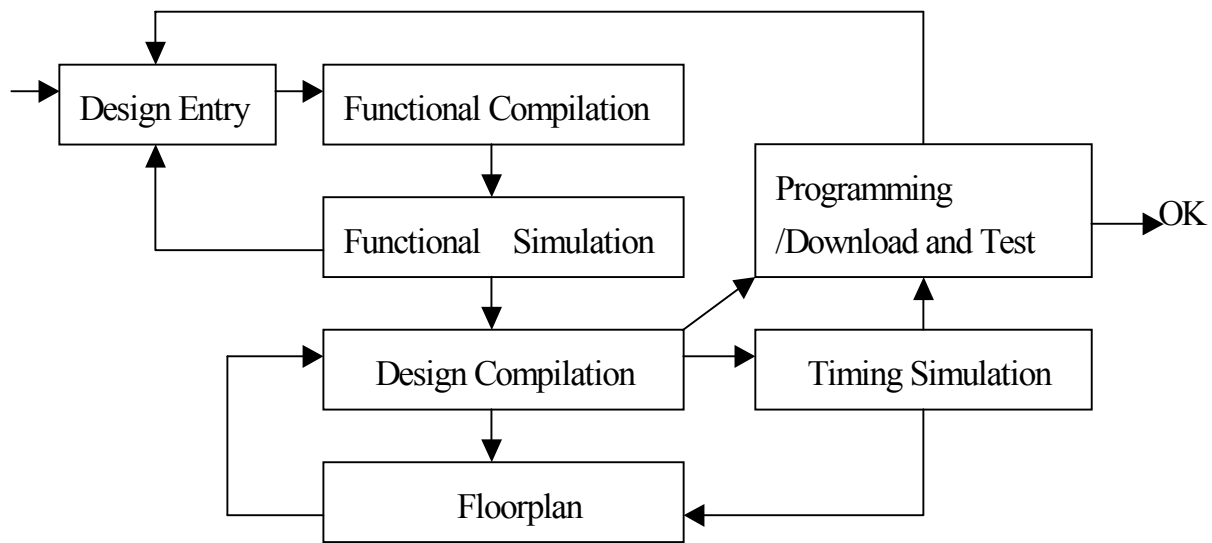


Figure 1.4 Flow of integrated logic circuit design

Table1.2 Comparison of traditional and integrated digital logic design (by tools)

Types	Tools
Traditional Digital Logic Design	<ul style="list-style-type: none">❑ Multifunctional Meter❑ Oscillator Display❑ Logic Probe❑ Logic Analyzer❑ Experimental board and Wiring Tool or Welding Iron❑ ...
Integrated Digital Logic Design	<ul style="list-style-type: none">❑ Multifunctional Meter❑ Oscillator Display❑ Logic Probe❑ PCs and EDA Design Tools❑ Experimental Platform



Table 1.3 Comparison of traditional and integrated digital logic design
(By elements)

Types	Elements
Traditional Digital Logic Design	<ul style="list-style-type: none">❑ Resister❑ Capacitor❑ LED, Seven Segment Display, Dot Matrix Display, LCD❑ SSI, MSI, LSI❑ A/D, D/A❑ Transistor, Amplifier❑ VLSI Chips❑ Wild Spread PCB❑ ...
Integrated Digital Logic Design	<ul style="list-style-type: none">❑ Resister❑ Capacitor❑ LED, Seven Segment Display, Dot Matrix Display, LCD❑ PLD components❑ A/D, D/A❑ Transistor, Amplifier❑ VLSI Chips❑ Compacted PCB❑ ...



Table 1.4 Comparison of traditional and integrated digital logic design
(By circuit design flow)

Types	Circuit Design Flow
Traditional Digital Logic Design	<ul style="list-style-type: none">❑ Specification Definition❑ Circuits Design on Papers (Only Schematic Entry)❑ Chips Selection❑ PCB Welding or Wiring❑ Circuit Testing and Modifying
Integrated Digital Logic Design	<ul style="list-style-type: none">❑ Specification Definition❑ Data Input (Including Schematic, Text, and Waveform)❑ Design Compilation❑ Design Simulation

Table 1.5 Comparison of traditional and integrated digital logic design
(By study requirements)

Types	Study Requirements
Traditional Digital Logic Design	<ul style="list-style-type: none">❑ Great Experience and Knowledge Needed❑ Use of Various Equipment❑ Time Consuming and Great Experience Requirement for PCB Welding or Wiring❑ Problems with Keeping Design Information



Integrated Digital Logic Design	<ul style="list-style-type: none">❑ Less Experience and Knowledge Needed❑ Decrease of Using Various Equipment❑ Time Saving, Ease of Understanding the Knowledge, and Good Information Maintenance, Reuse of Information, and Convenience of Modifying Design Data by Computer Aided Design❑ Decrease of PCB Welding or Wiring
---------------------------------	--

To the integrated digital logic design flow, we have following conclusions:

1. EDA computer aided design can effectively help us to learn new knowledge and save time. It can also keep design information well; reuse and modify the information easily.
2. It decreases the work of welding or wiring circuit boards. It also reduces defects from human factors.
3. It has “Design Entry → Circuit Simulation → Downloading Testing” streamline process, which is just right to today faster circuit development.
4. By performing the above process, PCB samples are almost completed and finalized, and have fewer defects. We then could decrease the possibilities to reproduce another samples and therefore shorten the developing time.
5. Because CPLD has great programmability, we could make samples and perform “Design Entry → Circuit Simulation → Downloading Testing” process at the same time. It helps us to shorten or avoid the time to wait for samples. (Note: LP-2900 is completed by this development mode.)
6. Because CPLD has great higher gate counts which can integrate many logic chips in a small board.



The integrated digital logic design flow can also be lectured in school. Instructors can introduce theories first, talk design and simulation next, and then prove the theories by downloading and testing circuits. Students will well understand the theories after taking the course. Each of theories or design examples can be proved by testing in class immediately. Students will have great impression and understanding, and so feel confident, willing to study further knowledge about digital logic. Unlike the flow of integrated digital logic design, the flow of traditional digital logic design will not have IC simulation after the introducing theories and design examples. It requires users to weld circuits or put components on breadboards by her/himself to test circuits. Welding process is very time consuming, plus bad connection in breadboards and human factors can cause faulty problems and prolong the processing time. Student might feel discouraged and only understand one or two simple logic designs and practices. If so, there will have no hope and improvement for our digital circuit design. Isn't it?

1.3 Programmable Logic Device - PLD

In previous section, we have mentioned that engineers could use Standard/Discrete Logic (e.g. TTL/COMS 74/54 family) and ASIC to make circuit boards. ASIC shown in Figure 1.2 can be divided into 4 different catalogers: Programmable Logic Devices, Gate Array, Cell-Based IC, and Full-Custom IC. However, higher performance, high-density logic integration, greater cost-effectiveness, and short development cycle are the four key factors considered by customers to buy chips. “Higher performance” is clock speed rate or coherent signal propagation, and is closely related to circuit process and architecture. For “high density logic integration”, it means that the device can integrate more logic gate counts in same area. This is one of the goals when engineers design circuits in the second run. They try to put more circuits in a smaller area to reduce PCB space and cost. “Greater cost-effectiveness” is of performing the same performance of circuit with less expense. “Short development cycle”, it obviously means to shorten development period, including the stages: design entry, compiling process, simulation, and programming as well as PCB testing. Definitely, the shorter developed time the better to catch up time to market. From the comparison shown in Table 1.1, PLD has greater advantages than Standard/Discrete Logic for all comparison items, but is inferior to Full-Custom IC in density and cost.



Programmable logic device (PLD) is a kind of IC. It has a user-configuration function, which allows users to customize their logic functions including Boolean expression and/or registered function. It is quite different from TTL/COMS standard logic that only provides fixed functions in early stage. As PLD density is increasing and the new process is adopted, prices are gradually decreasing, and therefore PLD has been able to offer higher density, higher performance, and lower prices than standard logic. It is becoming very valuable in the market.

All devices that come with user-configuration functions can be seen as PLD devices. They include PAL/GAL devices with the simple, low-density, low-pin-count features, field-programmable gate array (FPGA) with high-density and high-pin-count features, and complex PLD with high-density and high-pin-count feature. Those PLDs have their own different internal structures and internal memory design models. It is depended on the needs and the sizes of the circuits to select devices properly. To PAL, it is a simple programmable logic device, which has gate counts from 100 to 1000 and IC pin counts within 28 pins. Bipolar process is used for PAL, which is an old product and can only record once and cannot delete old data. PAL has the potential of faster speed, but it can consume more electric power. For GAL, it is also a simple PLD, but is mad CMOS. GAL can rewrite and delete data many times. Its speed rate is slower but power consumption is less than PAL.

FPGA and CPLD are made by CMOS. Currently there are five techniques used in FPGA/CPLD, and those are EPROM, EEPROM, FLASH, SRAM, and Anti-Fuse. FPGA has higher density. The difference from CPLD is of that FPGA has less logic gates and focuses on registers. However, FPGA routing is complicated, causing the unfixed and longer timing delay. For students, it definitely is not easy to understand, and has to take longer to study the knowledge. Currently, there are some key suppliers in the market such as XILINX, Actel, Atmel, and AT&T. Among them, XILINX is the biggest supplier having the greatest market share in the market. CPLD, on the other hand, can rewrite and delete date unlimitedly and fix timing delay. It is also easy to use and make designs. The key CPLD suppliers include AMD, ALTERA, Lattice, Cypress, and ICT, etc. Among them, AMD and ALTERA are the two biggest suppliers in the market. (See Appendix A - PLD Suppliers and Main Products.)

Presently, PLD is applied in telecommunications such as mobile phones and radio bases, etc. It is also used in data communication network such as LAN, ATM, and

printers as well as scanners. For sure, it will be promoted into TV game and educational markets in the future.

PLD is constructed by configuration cell, logic cell, and interconnect. Configuration cell is a kind of memory. Engineers can program the cell to remember the connection processes between I/O pins and logic cells, between logic cells and logic cells, and between logic cells and interconnect. The process of those configuration cells is called “technology”, and all current PLD application technologies are listed in Table 1.6.

Table 1.6 Technologies of PLD configuration cell

Technologies	Re-configurable	Erase Methods	Types	Security
EPROM	Yes	Ultraviolet Ray	Nonvolatile	Yes
EEPROM	Yes	Electrical Erase	Nonvolatile	Yes
FLASH	Yes	Electrical Erase	Nonvolatile	Yes
Anti-Fuse	No	---	Nonvolatile	Yes
SRAM	Yes	Power Off	Volatile	No

Logic cell has two main design trends. One is “Macrocell” with the structure of “sum of production term”; the other is “logic element” with the structure of “Look-up Table” (LUT). In Macrocell, all productions are summed up for exclusive-OR operation and then connected to a programmable flip-flop. Thus, Macrocell has greater logic capacity, and is only constrained by the number of productions (It also called “P-term”.) A Marcocell usually has 20 to 40 logic gates. Logic element is usually made with 4 to 8 input LUT circuits, 1 or 2 programmable flip-flop, one faster carry circuit, and one sequentially-connected circuit to increase fan in (speed will not be affected a lot after sequentially connecting.). A logic element usually has 10 to 20 logic gates, and therefore a logic element is basically smaller than Marocell. PLD with sum of production term has more logic gates, and so is very suitable for the circuits requiring more combination logic designs. On the other hand, PLD with logic elements is useful for the sequential logic circuits requiring more registers.



Interconnect structures give channels to deliver information from I/O ports to logic elements. Currently “FPGA segmented interconnect” and “CPLD continuous interconnect” are two mainly connection types, and this is the key difference between FPLD and CPLD interconnects structures. FPGA segmented interconnect uses varying length lines connected by pass transistor or anti-fuses to connect logic cells. Each connecting point has an on/off element to control connecting direction. To build up required interconnection, all signals will have to pass through cells by several long or short channels. Thus, whenever we modify a design, the routed path will be different and delay changes, and consequently, this segment connection cannot predict the time delay by interconnection. For continuous interconnection, it uses metal wires to round elements in horizontal and vertical directions. Each metal wire only can transfer one signal, and this is what we called “global” interconnection. Several (8 to 16) logic elements are collected in one Logic Area Block, LAB. Elements are connected to each other by local interconnection first and become LAB blocks, and then, LAB blocks are linked together to complete the whole connection by global interconnection. As a result of this interconnection, the delay is predictable.

To sum above, we have following conclusions:

- Because there is a complicated wiring job to decide, the time to translate FPGA is longer than CPLD.
- Because CPLD interconnection is simple, it quite reduces wiring spaces, and the number of gates can also be increased a lot by use of three-metal process.
- FPGA is helpful for data path application such as pipeline design. CPLD, comparably, is useful for logic applications.

ALTERA is a US company. It is specialized in PLD and sells all different PLD products, including PLD (Programmable Logic Device), PAL (Programmable Array Logic), PLA (Programmable Logic Array), GAL (General Array Logic), and FPGA (Field-Programmable Gate Arrays), etc., popular in the current market. In early age, an electronic engineer had to use breadboard and many logic elements to verify a new circuit design. Welding devices and correcting mistakes also made an engineer worn out. Contrarily, today an engineer can just use PLD and remodel the circuit designs in computer. It is just like making your own circuit devices on your own tables. It cannot just save welding time but also avoid any mistakes caused in welding process.

Moreover, it can reduce the PCB space to meet the market trends requiring smaller and lighter electronic products.

Because digital logic can be calculated by mathematics, we can use different circuits to achieve the same circuit functions shown as in Figure 1.5. Initially, an AND-OR can replace AND-OR-AND integrated circuit. We then assume we would like to make a simple structure which connections can be changed, as we need.

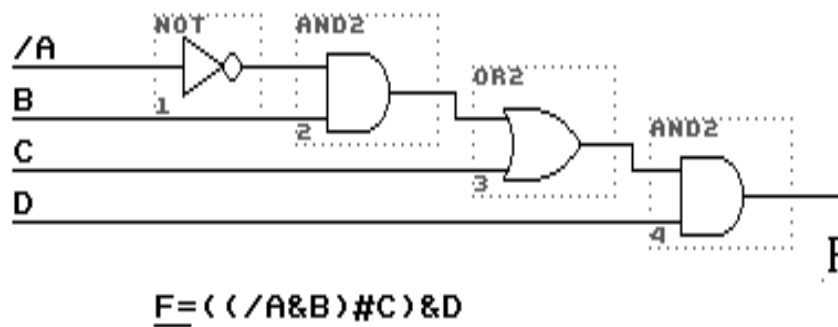


Figure 1.5 AND-OR-AND logic circuit

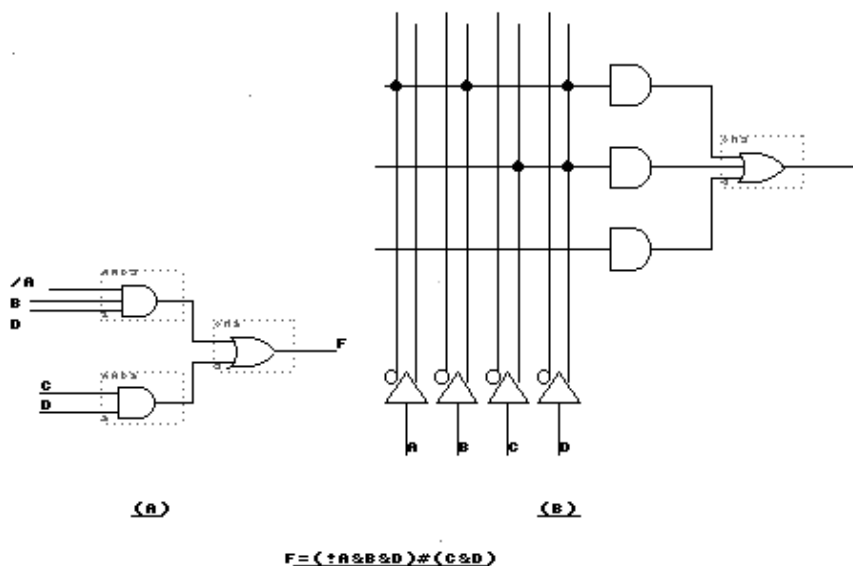


Figure 1.6 PLD evolution

Similarly, the structure in Figure 1.6b is a simple single structure. It is made by several multi-input AND gates and OR gates. Each AND gate input uses the same input signals that can be connected to AND gate directly or inversely. However, not all input signals will be connected to AND gates. Users can decide if input signals

would be connected to AND gate. For example, if we have the logic function $F = \{A \& B \& D\} + \{C \& D\}$, we can design a circuit as Figure 1.6b. If users have different circuit designs, they would also get different functions, and that is the first idea for programmable logic. For further explanation, we only have to extend the structure in Figure 1.6b, and logic functions will become more complicated consequently. This multi-input AND gates are called as P-term. For old PLD integrated circuit, it only has multiple P-terms like the structure in Figure 1.6b. Today, this kind of structure cannot meet most of engineers' demands. This is because of less gate counts and lack of Flip-flop. Therefore, in Figure 1.7, we add Latch and channel selection functions to the basic structure in Figure 1.6.

Until now, the general logic we have talked about is only a general structure. It requires users to decide which connection points they would like to link together and would not. Of course, referring to element structure, users can decide whether to make an interconnection for all circuit functions. For elements requiring less interconnection points, it might be acceptable. However, if elements have more than thousand interconnection points, this job would become very tedious and users would easily make mistakes. Thus, we need computer aids to help us to design circuits easily, and then use the software to make interconnection plan. Thereafter, we know a completed programmable logic world have to be done by programmable hardware and software to help design interconnection. For the two parts, we will discuss in the following section.

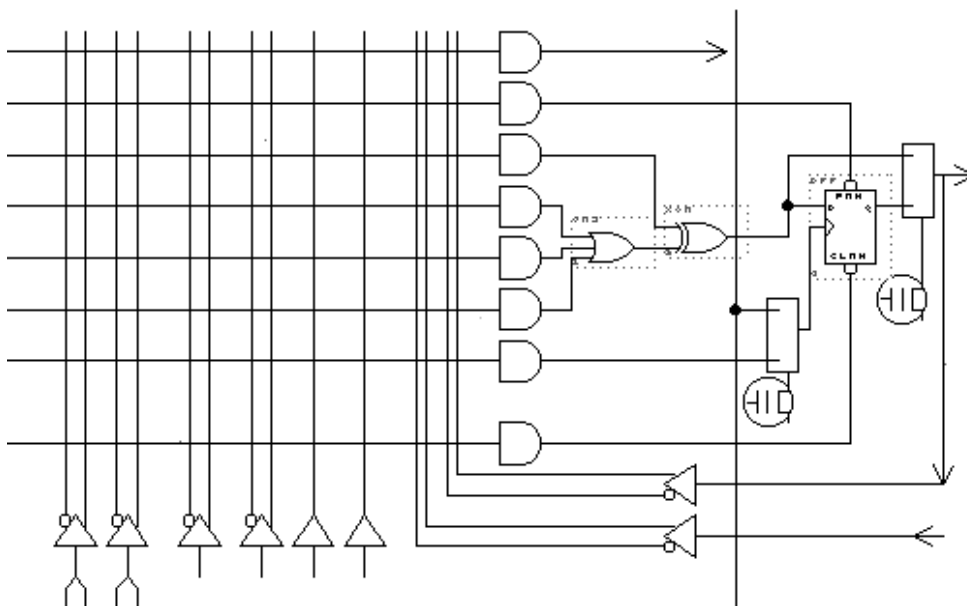


Figure 1.7 Macrocell structure



First, we would like to introduce ALTERA hardware. Usually PLD application is restricted by the hardware including inner gates and I/O pins. ALTERA produces gates from 150 to 1 million and pins from 20 pins to 560 pins for its devices. Definitely, those numbers are still being updated, and we know ALTERA can give users broad and various selections. Beside that, interconnection is also one of ALTERA's unique features. Because ALTERA uses metal wires for its interconnections, and it adds extra metal wire connection between any two Logic Blocks. We call this connections as "continuous interconnect", different from "segmented interconnect". Because of this difference, its interconnection time delay is predictable and would not be affected by interconnection path.

1.3.1 ALTERA CPLD:

Until 2000, ALTERA devices can be divided into 10 different families:

1. CLASSIC
2. MAX3000
3. MAX5000
4. MAX7000
5. FLASH LOGIC
6. FLEX6000
7. FLEX8000
8. MAX9000
9. FLEX10K
10. FLEX20K

MAX7000 family has members: MAX7000, MAX7000E, MAX7000S, MAX7000A and MAX7000AE, etc. FLEX10K family has members: FLEX10K, FLEX10KA, FLEX10KB, and FLEX10KE, etc.

In the programmable logic world, the unique feature is a logic cell can be reproduced continuously in a chip. The difference between higher capacity and lower capacity of



devices in the same family is the amount of the reproduced logic cells. Therefore, we could understand other devices from the smallest device in the same family.

Other relevant information, please see Appendix A and B.

1.4 PC Aided Digital Logic Design

In previous section, we have briefly talked about ALTERA devices, and know ALTERA has all family devices. However, we still have many questions need for solutions. As a device has a lot of interconnection points, if users have to make connected-or-disconnected decisions for all points, it will obviously increase the difficulties to the job. Thus, a good PLD must have great design software to help users complete the work effectively and efficiently, and so ALTERA is currently more focusing on software development while designing great family devices.

PLD Computer Aid Digital Circuit Design means engineers could use computers to complete PLD digital circuit design. This is a new design environment. It offers an integrated data management, hierarchical design, and multi-window environment. The design process includes four steps: design entry, compilation process, verification simulation, and PLD programming. The first step is “design entry”. It includes graphic editor, text editor, and waveform editor. A hierarchical mix entry, which is a combination with graph, text, and waveform, is the best way of design.

Compilation is the process including the testing of electric feature of entry circuits (e.g. short cut and source less input, etc.), circuit synthesize and netlist extract for functional simulations, circuit floorplan (that is to fit the circuit into PLD), and the netlist extract for timing simulation as well as the creation of PLD programming files. Verification simulation can be divided into functional simulation and timing simulation. Functional simulation is actually timing simulation when we assume propagation delay and setup time is zero. Timing simulation, on the other hand, is the simulation performing based on the value of the propagation delay and setup time from actual circuit floorplan into real PLD. Functional simulation could roughly verify circuits first to ensure circuit functions meet specifications. Timing simulations then further ensure the circuits work well in PLD.



Then after verification, programming PLD is a must process. There are two technologies: (1) download with SRAM technologies; (2) programming with EPROM, EEPROM, and FLASH are adopted in PLD. The download technology is useful during R&D and learning periods; programming, however, is very helpful when circuits are well developed already or circuits would not be modified in a short time. No matter which technologies we would use, it needs to have development tools to complete the work effectively and efficiently.

From a design diagram, the PLD Computer Aid Digital Circuit Design System is the great tool for engineers or students to complete their work efficiently. Because the systems can offer great circuit entries, verification simulation, devices programming environments as well as integrated data management, they shorten the time to design new circuits and catch up the time to market efficiently. Besides the above basic functions, PLD Computer Aid Digital Circuit Design System also has the following features:

1. Hierarchical design with mixed entries: Graphic editor is quite useful for a small logic design. Because the transformation can easily completed from Truth table or state diagram to logic gates by workforce. Logic gates are then entered into the form of graphic editor. However, it will become very time consuming and easy to make mistakes to complete the transformation by workforce when trying to make a big logic design. Therefore, like writing any programs in C language, we use texts to describe circuits and have computers, which specialize in computing and mapping algorithms to handle the transformation, making circuit modification easy. Moreover, a circuit has hundred thousand of gates, which have great duplication, and graphic entry becomes a hard and complicated task. No matter in terms of design or fault detection, graphic entry is more difficult than text. Beside graphic and text entries, we could also use waveform entry to describe circuits. By wave entry, we could know the output that input data would be related to, if we assume the circuits as a dark box. The computers are asked for the generation of Truth table or state diagram. Thus, the generated circuits might not be the simplest ones. The three entries have their own benefits



and drawbacks. They are all useful in different areas. If we use the three together, we could further improve efficiency in design and affectivity in teamwork. To combine the three together, hierarchical design function, therefore, is essential.

2. Structure independent: During the early stages of design entry, functional compilation (including logic synthesis and minimization as well as netlist extract for simulation), and functional simulation, engineers do not have to concern which PLD device will be used and what the internal structure is. This is what we called “Structure Independent”. Until the end of design compilation, by technology mapping algorithms, we then place synthesized circuit on the selected PLD. If the circuits are too big to be fitted in a piece of device, we might partition the circuits before placement or change a bigger device. This unique feature gives a great flexibility for future use of PLD.
3. Providing industry-standard LPM: LPM element is a macro-function, which allows users to change its sizes based on parameters. For example, the length of LPM counters and LPM adders is a parameter that can be changeable; the type and the number of bits of LPM multiplexer and LPM register are another changeable parameters. The LPM provides a simplified design entry and has better circuit integration.
4. Providing time driven compilation: To increase performance of the designed circuits is the goal for engineers in the second stage, or to meet the propagation delay (t_{pd}) and the speed (f_{max}) are defined on the design specification. If a development system could provide time driven compilation, it could reduce much complicated work based on the user specification to synthesize circuit integrate and plan as well as allocate circuits.



5. Providing multi-device simulation function: When designing big circuits, we usually partition circuit into several smaller PLD devices. At this moment, a singular-device simulation might have some problems and cannot fully and efficiently assist engineers to complete their whole design projects. Therefore, it would be better to have multi-device simulation functions to make design process more smoothly, easily, and correctly.
6. Great Design-rule checking ability: In design process, if development systems could early find some unreliable logic problems such as static hazards, race condition, multi-level clocks and asynchronous input, etc., we could greatly decrease circuit failures. Therefore, this feature can much shorten the leading time to launch new circuits in the market.
7. Providing standard CAE interface: To connect with other EDA design systems, we should systems that have interfaces able to read and write VHDL and EDIF. Because of the interface to connect with other systems, we could share design resources and conveniently and efficiently complete the design projects with our team members.

In the next section, we will detail the EDA tool of ALTERA-- MAX+PLUS II.

1.4.1 ALTERA EDA Tool-MAX+PLUS II

ALTERA EDA tool is a kind of software called “MAX+PLUS II”. Until 2000, it already has version 9.x. This software can support designers from design entry to the creation of programming file that programs interconnection as well as simulation process. In another words, designers can use the same software to complete all design process. There is no need to get another software from third party to support ALTERA hardware devices. Once designers familiar with MAX+PLUS II, all logic design can be totally completed by this same software. In this case, designers or students do not have to

continuously learn different software and therefore save more time. When designers need same-function logic circuit, they do not have to repeat the design process again and only have to save the circuit as an element in MAX+PLUS II firmware library. By calling the element, designers can add the circuit into their designs, and interconnect each of element inputs and outputs to reduce lots of work for logic design entry.

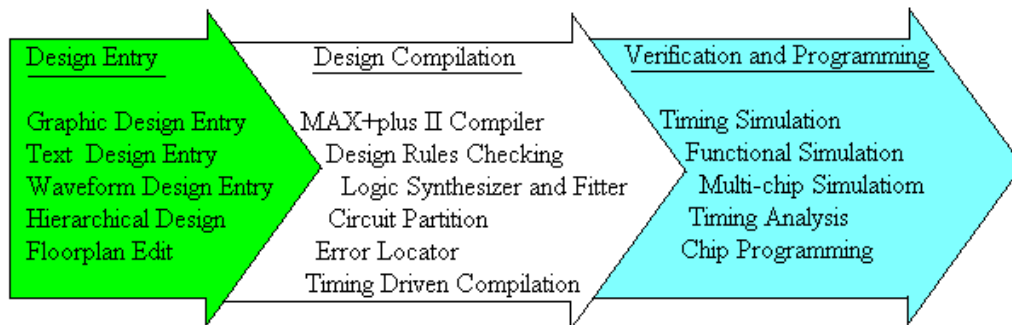


Figure 1.8 MAX + PLUS II functional diagram and design flow

MAX+PLUS II is a window application tool. It can be used with Windows 95/98 and Windows NT. At the same time, ALTERA also offers a workstation edition. Therefore, the use of MAX+PLUS II is the same as the use of general window applications. Mostly it operates with graphs. Figure 1.8 indicates MAX+ PLUS II functional diagram and design flow. Fundamentally, a programmable logic design has 3 major steps:

Design entry: In previous section, we already know workforce cannot program all interconnection points efficiently, and it will become more difficult for a bigger circuit design. Thus, we need a user-friendly tool to describe the logic circuit that we want. It is just like programming. Designers can use C language to write programs. They can also use assembly or even machine codes to write programs. It is easy to understand the advantages and disadvantages of design entry by using C language. Figure 1.9 indicates the graphic entry in MAX+PLUS II.

Design compilation: After design entry, incorrect description or electric faults can be detected by compilation. Once all circuits are correct, compilers will follow designers' direction to synthesize circuits into selected devices (that is logic synthesis and floorplan.), and generate programming files for programming devices.

Verification and programming: After completing above two steps, we already have a programmed device. However, there might still have some problems. For example, design description is accurate but circuit function is incorrect. In another words, it is because design is incorrect (logic error) or devices cannot meet the real requirements (timing error). Therefore, it is still necessary to use real circuits to verify the devices and ensure they are applicable in the real world. Simulation, as a result, could identify circuit operation status without physical circuits. It greatly helps designers to detect faulty problems in early design process.

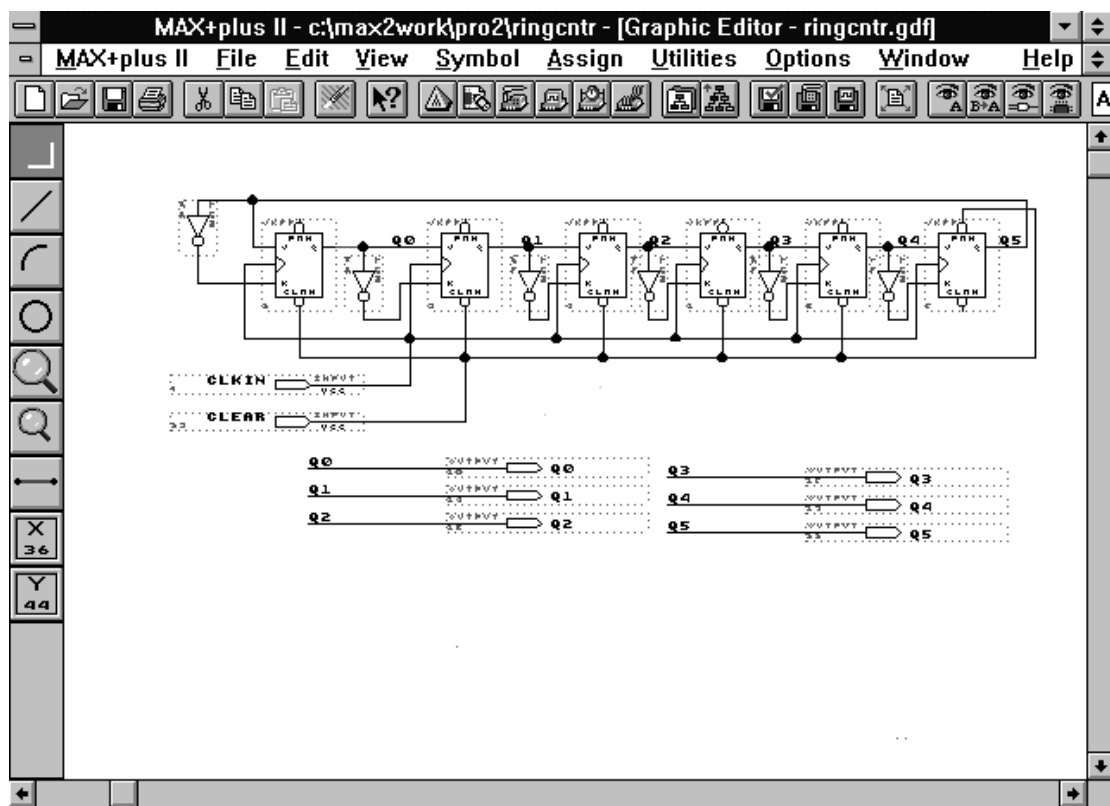


Figure 1.9 Graphic entry in MAX+PLUS II

The following is MAX + PLUS II functional description:

Design entry : Fundamentally, MAX+PLUS II has three ways to input designs. Those three can be used together in one circuit design; that is, they can mutually use together. Designers can choose the best ways to make each partial circuit as “sub-circuit” separately, and combine all of them together finally. In MAX+PLUS II, it also has had standardized 74 family functional elements. Designers can call the elements out and use them directly. MAX+PLUS II software also provides the features of Library of Parameterized Modules (LPM). Designers only need to set up LPM parameters in advance. Various functional circuits will be generated automatically with different bits and functions. For example, memory, adder, and multiplier are parameterized.

Graphic entry: Graphic entry is the most acceptable technology by designers. To complete design entry, it only requires interconnection between functional blocks, like drawing circuit diagrams. Figure 1.9 is Graphic entry in MAX+PLUS II.

Text entry: Though it is easy to design circuits by graphic entry, it pre-requires all designs completely first, including Truth-value table or state diagram needed to be reformed into logic circuits by workforce, and then entered in graphs. However, once a design fault is found and needs to be corrected, we have to redo all calculation and design entry. Contrarily, if by text editors, we only have to correct the design fault, not to redo the whole process, and then have computers to handle all the calculation, and that is computers’ expertise. For sure, this is just one of text entry benefits. Not like graphic entry, text entry does not have hundreds or thousands of graphs needed to enter. It makes the jobs easier on circuit design and error detection for devices with thousand gates. MAX+PLUS II totally have three description languages: ALTERA Hardware Description Language, AHDL, Very high speed integrated circuit Hardware Description Language, VHDL, and Verilog HDL. Figure 1.10 is Text entry in MAX+PLUS II.

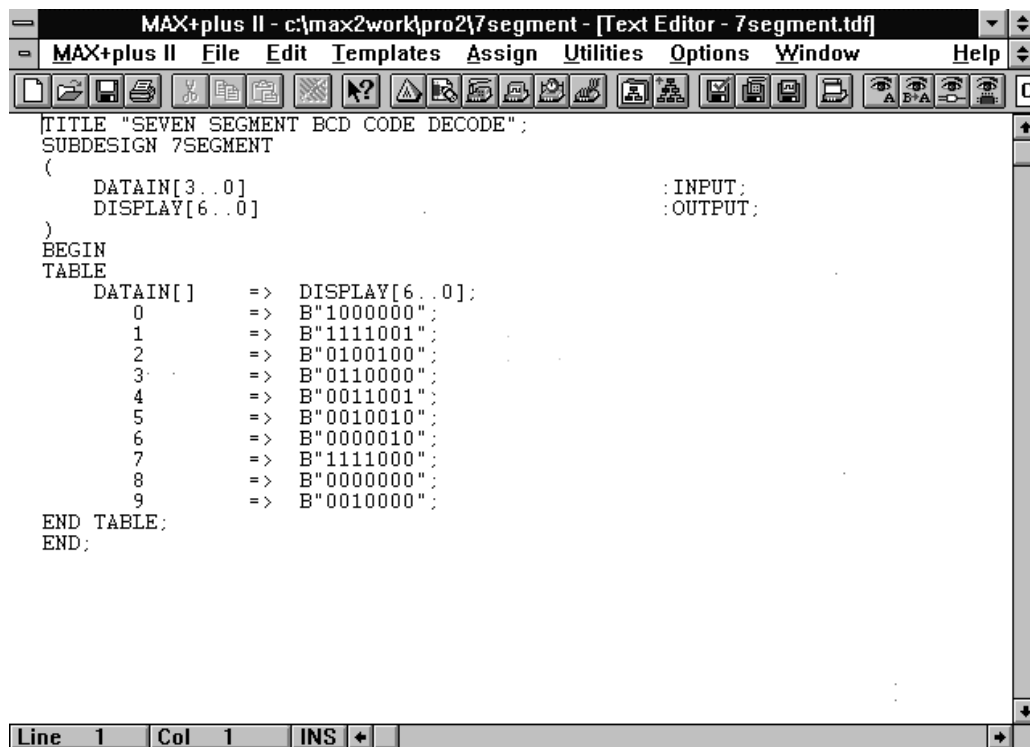


Figure 1.10 Text entry in MAX+PLUS II

Waveform entry: Waveform is the third way of MAX+PLUS II design entries. It allows users to describe circuit functions by drawing waveform directly. Figure 1.11 is Waveform entry in MAX+PLUS II.

Industry-standard CAE entry: EDIF is an industry-standard CAE netlist file. MAX+PLUS II uses this standard file format to communicate with other CAD software such as Synopsis, Viewlogic, Cadence, and Mentor Graphics, etc. It also offers functional library for them. Currently, MAX+PLUS II has two standards. One is EDIF200 and the other is EDIF300.

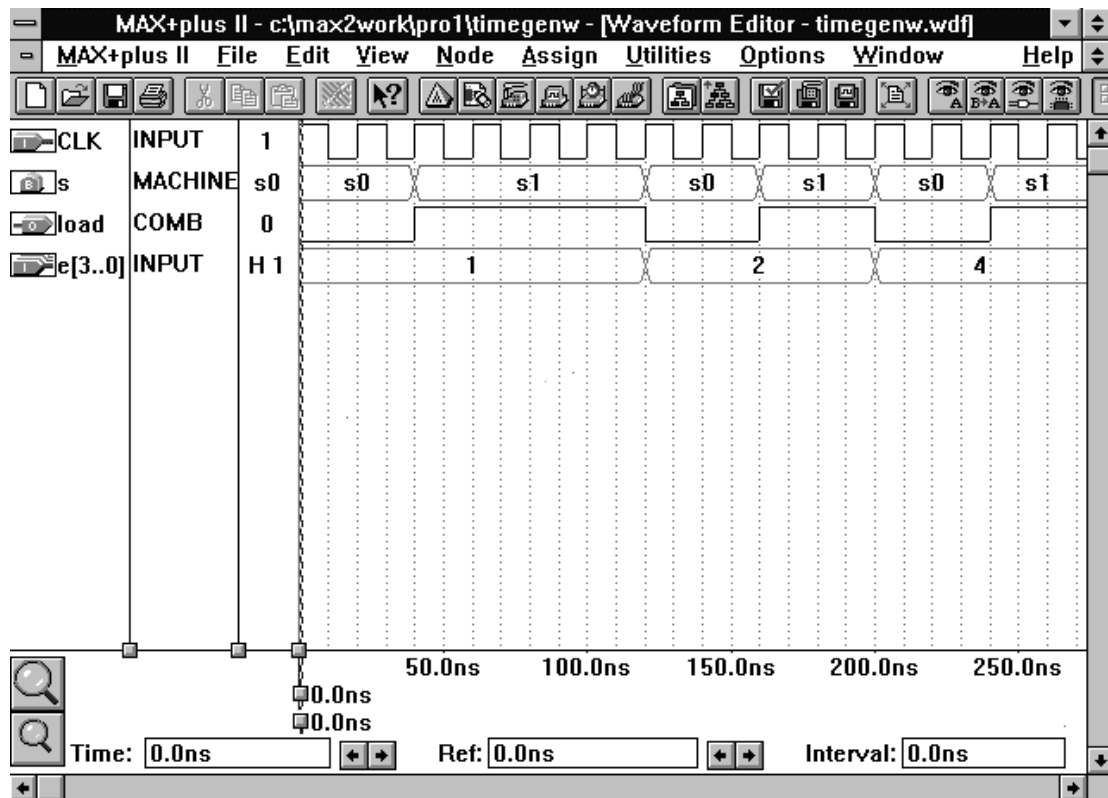


Figure 1.11 Waveform entry in MAX+PLUS II

Design compilation: No matter what design technologies and ALTERA devices are adopted, to complete logic synthesis after verifying a circuit's electric features, it is necessary to pass compilation process shown in Figure 1.12. Once compilation is finished, we will get programming files and some information such as the reports of delay status and pin count arrangement. During this process, it also provides some convenient tools to help designers find faults and increase efficiency. The functions of compilation are described as below:

- (1) Design-rules checking: In the early of compilation, it can detect potential problems from design files, such as oscillation and pointing out location as a reference for designers.
- (2) Logic synthesis and fitting: This is the core part of the whole software. It can give you logic synthesis and circuit fitting based on the PLD structures you choose. After giving synthesis and fitting, it then decides all interconnection for the whole circuit design.
- (3) Multi-device partitioning: If a design file cannot be fitted in a certain selected device, software will automatically divide the design file into two more devices. This process could be fully or partially done by workforce. It

allows us not really separate design files, and only has to reset data for future use.

- (4) Timing-driven compilation: This function allows users to set up some timing parameters such as delayed time and highest frequency, etc. A compiler will follow the parameter setting and try to come out desired solutions on its best. However, because of the structures of logic elements, designers can take advantages of time-driven compilation only when adopting the devices of FLEX8000 and FLEX10K families. If adopting other family devices, it is not necessary to have this kind of function.
- (5) Automatic error location: During compilation process, whenever a design fault is found, systems will show an error message. By using automatic error location features, the systems will automatically open the files that have errors occurred, and then indicate the error locations clearly.

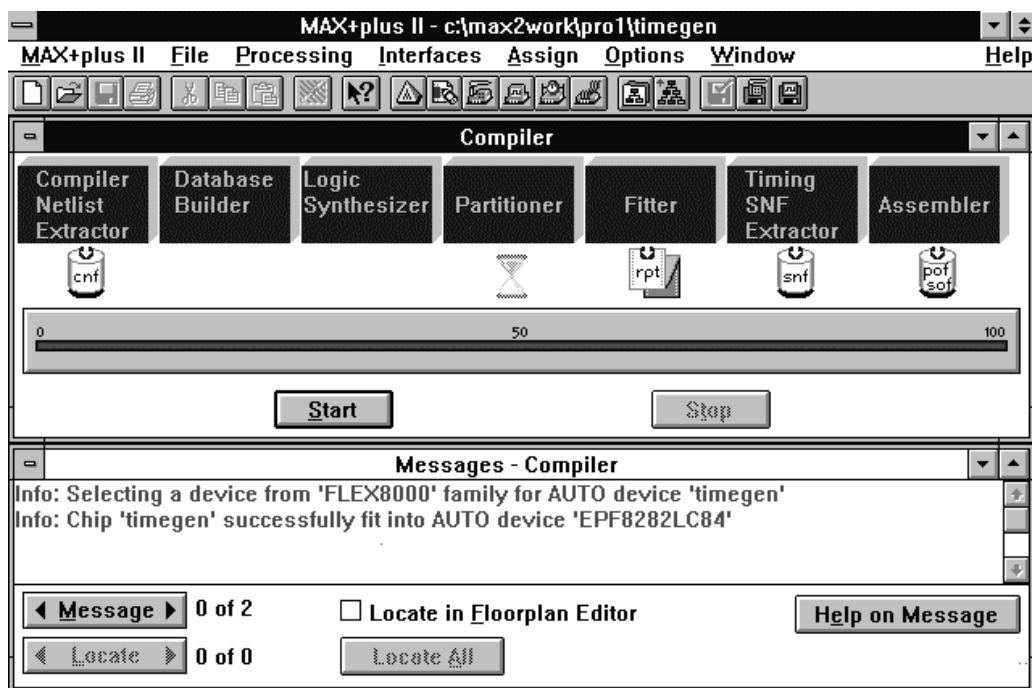


Figure 1.12 Design compilation in MAX+PLUS II

Verification and programming:

- (1) Simulation: Designers can use waveform editor to define the input waveform of the designed circuits. After entering waveforms, the software can automatically simulate and display output waveforms for inspecting design. In MAX+PLUS II, there are three simulations: 1.) Timing simulation including delay time and setup time of devices selected by designers, 2.) Functional simulation excluding delay time and setup time of devices

- selected by designers, and 3.) Multi-device simulation allowing users interconnect several ALTERA devices and then simulate all together.
- (2) Timing analysis: This function helps designers to understand timing performance by numbers at the beginning of design projects. The timing performance, for example, is as delay time, setup time, and registered performance analysis shown in Figure 1.13 to 1.17.

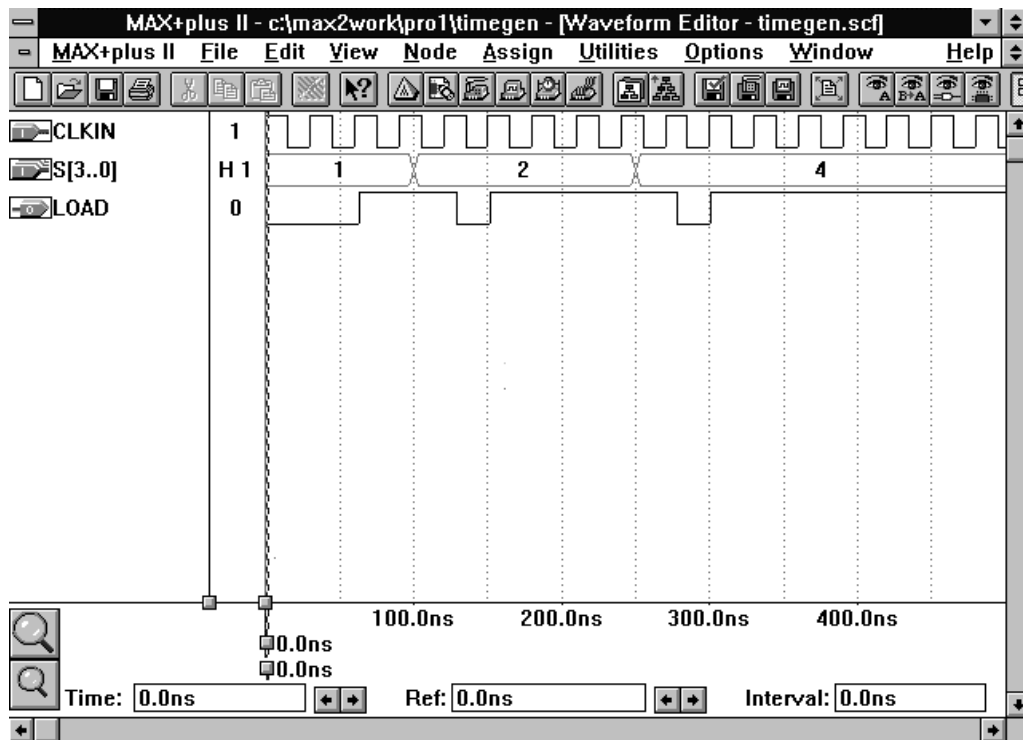


Figure 1.13 Timing simulation in MAX+PLUS II

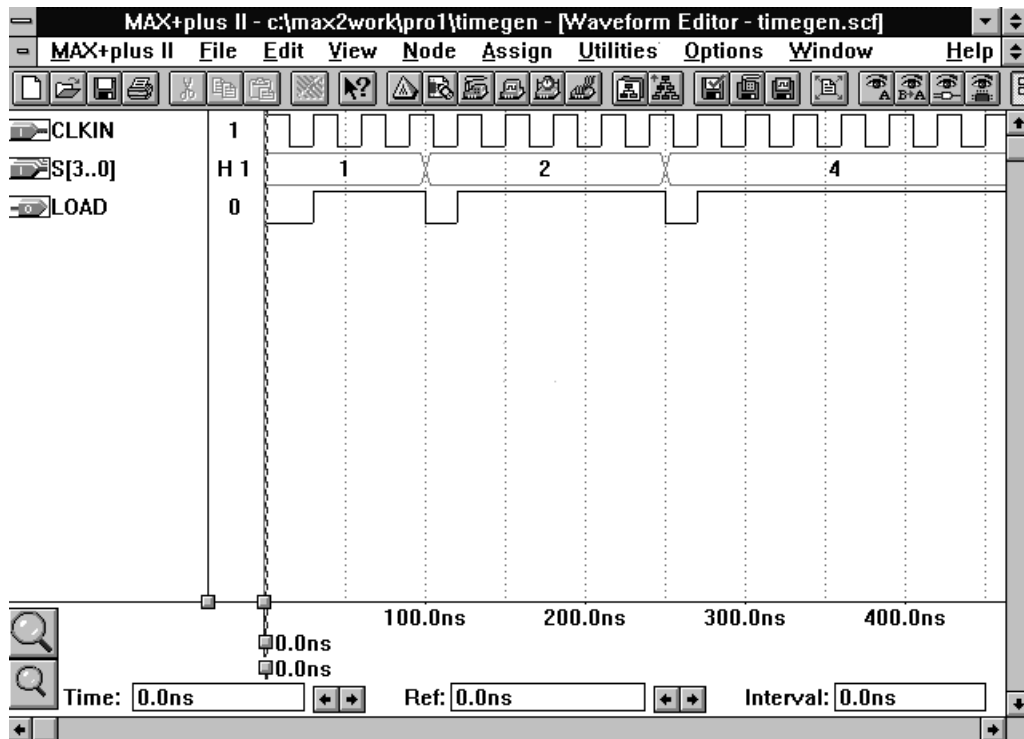


Figure 1.14 Functional simulation in MAX+PLUS II

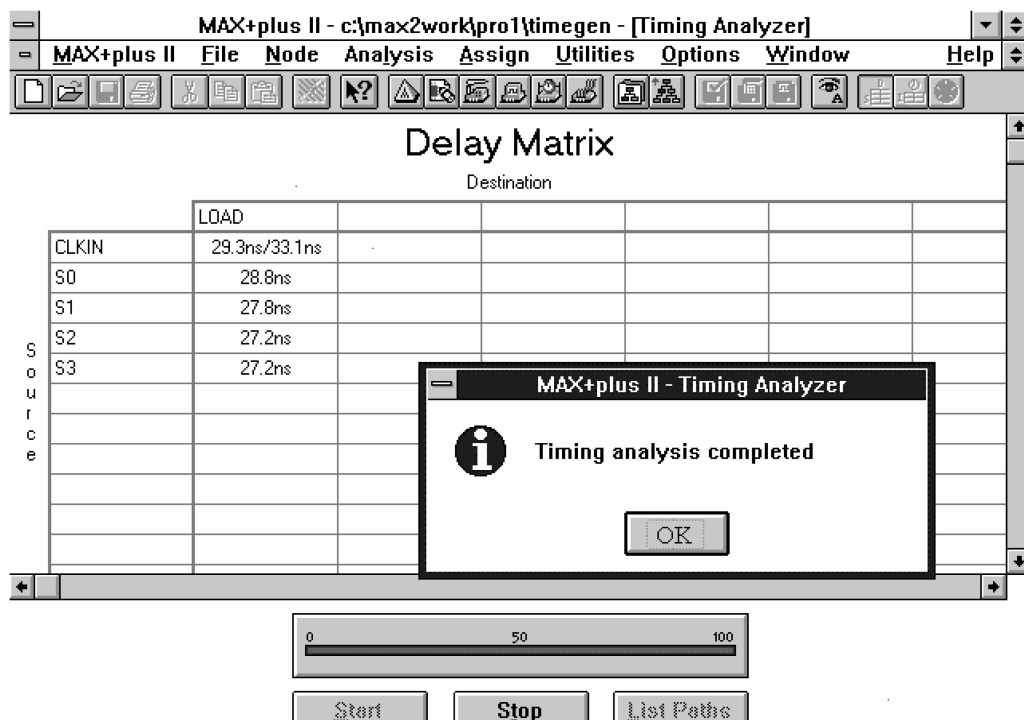


Figure 1.15 Time analysis in MAX+PLUS II—delay time analysis

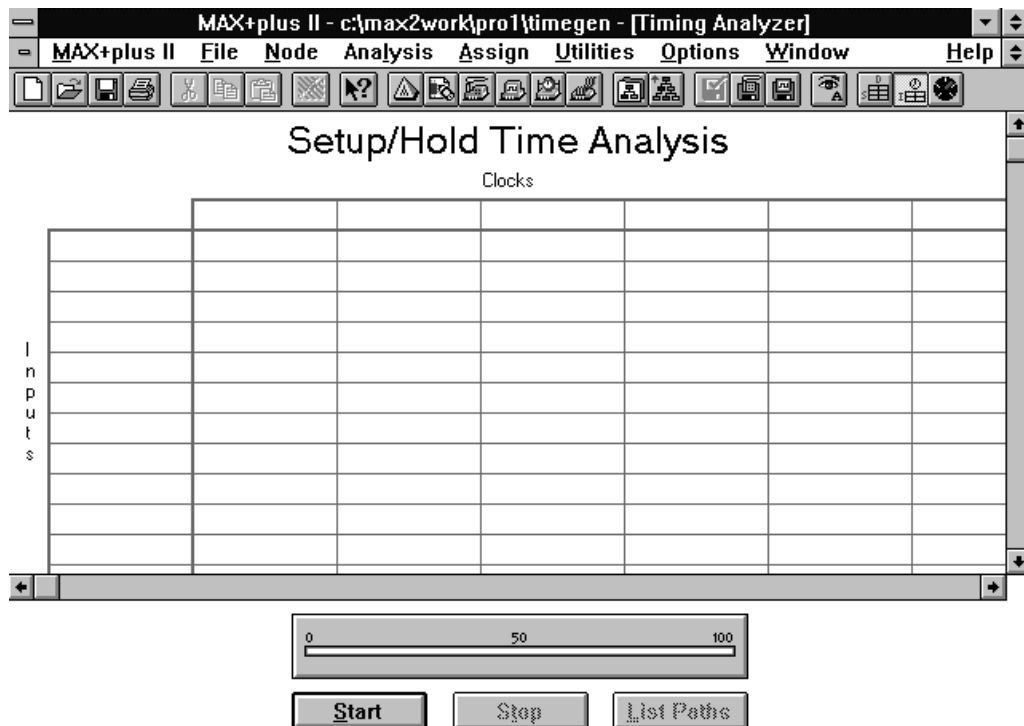


Figure 1.16 Time analysis in MAX+PLUS II—setup time analysis

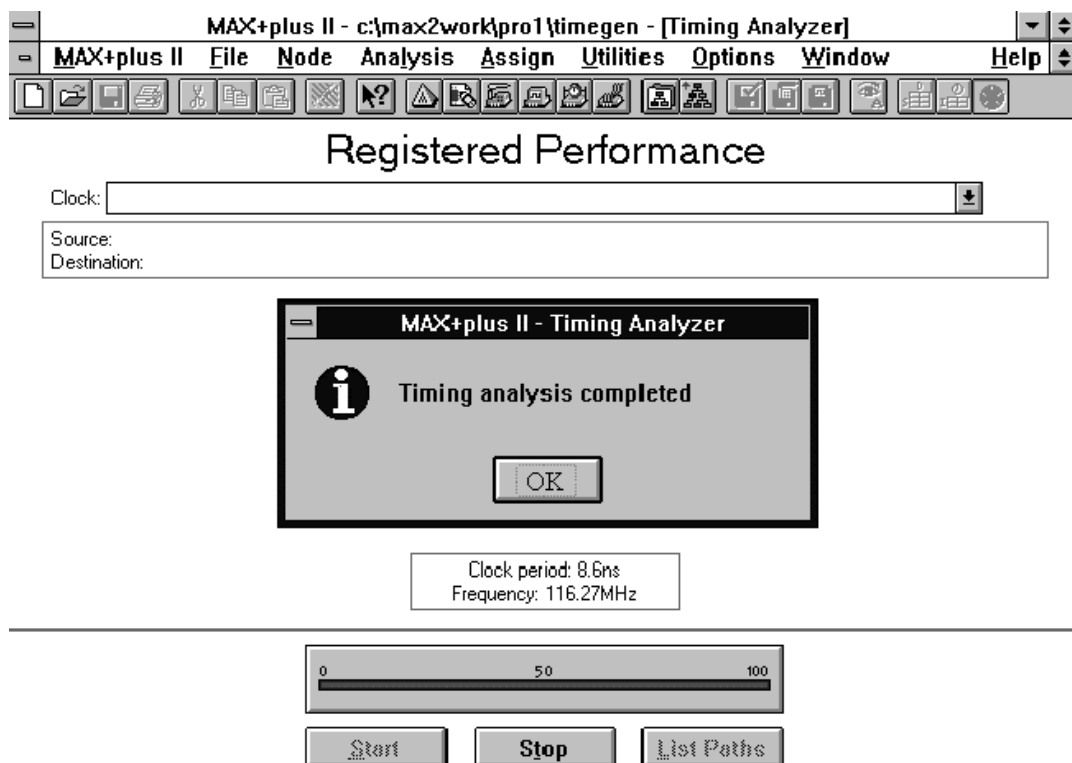


Figure 1.17 Time analysis in MAX+PLUS II—registered performance analysis



- (3) Device programming: When software completes all process, configuration data will be programmed or written into relative devices for testing hardware.

The above is the introduction of ALTERA devices and software. Currently, ALTERA provides free software of MAX+PLUS II for users with registration requirement. The operation process of the free software is the same as commercial version but only available for the devices of the families of CLASSIC, MAX5000, EPM7032, EPM7064, and EPM7096 as well as EPF8282. Meanwhile, it does not support waveform entry, and multi-device partitioning and simulation, etc. Other than that, the rest of the functions are exactly the same as MAX+PLUS II commercial version. About the user guide, MAX+PLUS II has great on-line help. We will further discuss how to set up and use the software of MAX+PLUS II Baseline 9.23 in Chapter 4.

1.5 Experimental Platform

An experimental platform is essential in the integrated digital logic design environment. It not just requires SRAM CPLD device, but also power, downloading interface, and I/O elements which include LED, seven segments display, buzzer, clocks, switches, pulse switches, 43 keyboard, 8×8 dot matrix display, liquid display, and A/D & D/A circuit modules. The main purpose of the platform is to offer a simple and accessible environment to test circuits and to reduce the time needed for circuit design.

1.5.1 LP-2900 CPLD Logic Design

Experimental Platform

In Figure 1.18, LP-2900 CPLD logic design experimental platform was the new product of Leap Company in 1999. The company currently focuses on the development of the logic experimental platform, which has the learning environment that integrates design, simulation, and verification. The development also has the educational features such as easy setup, great access, quick response, and progressive



learning. It is based on ALTERA EPF10K10TC144-4 CPLD to develop a multi-function logic design experimental platform. The platform has CPLD device board, I/O element experimental board, PC download interface, and power.

❖ CPLD

On the CPLD device board, there are an ALTERA 10K device, an EPROM device socket, a reset switch, and a pin status display LED which is a surface mounted device (SMD). ALTERA EPF10K10TC144-4 CPLD devices provide flexibility and convenience for continuously downloading and programming new circuits. The sockets of EPROM devices can be plugged EPROM devices programmed with “configuration data”, providing another way to program EPF10K10TC144-4 CPLD devices. A reset switch changes 10K devices from user mode to command mode. After circuit configuration and circuit reset, 10K devices would back to original user mode. A pin status display LED is a SMD device that displays the status of all pin counts, making circuit defects easy to find out.

❖ I/O element experimental board

The big board under CPLD devices is I/O element experimental board. This big board totally has 12 different I/O elements including: 1.) 4 sets of red, yellow, and green LED; 2.) 6 common cathode 7-segment displays; 3.) One buzzer; 4.) Two electronic dices; 5.) One clock circuit; 6.) 3 sets of 8-bit data switches; 7.) 4 pulse switches; 8.) One 4×3 keyboard; 9.) One 8×8 dot matrix display; 10.) One LCD display; 11.) A/D & D/A circuit modules, and 12.) 8051 module. The experimental board almost includes all I/O elements generally used in digital logic circuits. It provides a whole completed learning environment or fast prototyping circuit design environment.

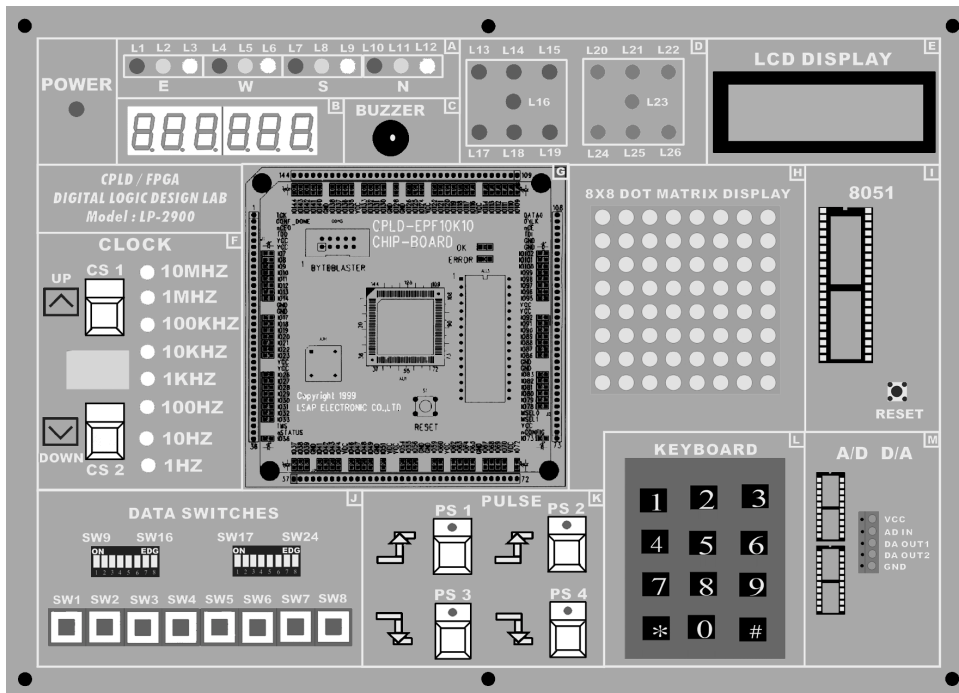


Figure 1.18 Overview of LP-2900 CPLD logic design experimental platform

❖ PC Printer Download Interface

To program 10K devices, a download interface provides a great convenient parallel channel to download “configuration data” from a PC printer port. It is not necessary to install or remove interface, but plug in printer cables.

❖ Power

AC 90V~260V 50/60Hz, 2A input provides power for all circuits and has short circuit protection.

Other reference, please see Chapter 9 in this book.



1.6 Evaluation and Test

Please answer the following questions to review this chapter.

- ❑ Do you know what benefits of digital systems are better than analog systems?
- ❑ Do you know what characteristics A/D converters and D/A converters play respectively?
- ❑ Do you know what design environments are introduced in this chapter?
- ❑ Could you indicate what types of devices are been used by engineers today?
- ❑ Do you know why standard logic devices would be gradually disappeared?
- ❑ Do you know the factors making “Integrated Digital Logic Design Environment”?
- ❑ Could you name some of PLD suppliers?
- ❑ Could you explain why ALTERA 8K and 10K devices are able to continuously download and program new circuits?

CHAPTER 2

Numerical System



LEAP



In the digital world, it is unavoidable to describe something by numbers. How many numerical systems expressing numbers are in the digital world? How do they express? In this chapter, we will focus on conversion and expression of those numerical systems. We also will introduce binary arithmetic and BCD codes in this chapter.

2.1 Numeric Expressions

Because humans have ten fingers and toes, naturally we would like to count numbers by our fingers, and that is what we called “decimal system”. In another words, each number is from 0 to 9 (unit, decimal, hundred) , and then carry to next digits after counting to 10. Decimal system is the most popular numerical system in our lives. Beside decimal, there also have binary, octal, and hexadecimal commonly used in numerical systems. Table 2.1 indicates the common expressions of the numerical systems.

A numerical system’s base is the number of the symbols included in this system. Decimal system, for example, has 10 symbols of 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. Therefore its base is 10. Each weight in numerical systems is the multiple of the base of its previous weight. For instance, 2725D is a four-digit number.

5	—————	its weight is 10^0
2	—————	its weight is $10^1 = 10^0$ (the weight of 5)×10
7	—————	its weight is $10^2 = 10^1$ (the weight of 2)×10
2	—————	its weight is $10^3 = 10^2$ (the weight of 7)×10

Table 2.1 Numerical system common expression

Numerical System	Expression	Example
Decimal	Add D (decimal) after the last digit (omissible)	1245D or 1245
Binary	Add B (binary) after the last digit	01010101B
Octal	Add O (octal) after the last digit	4767O
Hexadecimal	Add H (hexadecimal) after the last digit	1A2FH

Therefore, 2725D is $2 \times 10^3 + 7 \times 10^2 + 2 \times 10^1 + 5 \times 10^0$.

In general, an r-base numerical system uses the numbers from 0 to r-1. Value N could be express by an r-base system as below:

$$N = a_n \times r^n + a_{n-1} \times r^{n-1} + a_{n-2} \times r^{n-2} + a_{n-3} \times r^{n-3} + \dots + a_1 \times r^1 + a_0 \times r^0$$

In the equation, “n” represents 0, 1, 2, 3, ... The symbol “r” is the base of the numerical system, and “a” is the number from 0 to r – 1.

For value N from 0 to 1 could be expressed:

$$N = a_{-1} \times r^{-1} + a_{-2} \times r^{-2} + a_{-3} \times r^{-3} + a_{-4} \times r^{-4} + \dots + a_{-n+1} \times r^{-n+1} + a_{-n} \times r^{-n}$$

Thus, a decimal fraction 0.8125 is:

$$\begin{aligned} 0.8125 &= 0.8000 + 0.0100 + 0.0020 + 0.0005 \\ &= 8 \times 10^{-1} + 1 \times 10^{-2} + 2 \times 10^{-3} + 5 \times 10^{-4} \\ &= a_{-1} \times 10^{-1} + a_{-2} \times 10^{-2} + a_{-3} \times 10^{-3} + a_{-4} \times 10^{-4} \end{aligned}$$

Where $a_{-1} = 8$, $a_{-2} = 1$, $a_{-3} = 2$ and $a_{-4} = 5$

2.1.1 Binary

As mentioned before, a numerical system base is the number of the symbols used in the system. For a binary system, it has 2 symbols: 0 and 1. Therefore its base is 2. In numerical systems, each weight is the multiple of the base of the previous weight; for example, 01001100B:

0	—————	its weight is 2^0
0	—————	its weight is $2^1 = 2^0$ (the weight of 0) \times 2
1	—————	its weight is $2^2 = 2^1$ (the weight of 0) \times 2
1	—————	its weight is $2^3 = 2^2$ (the weight of 1) \times 2
0	—————	its weight is $2^4 = 2^3$ (the weight of 1) \times 2
0	—————	its weight is $2^5 = 2^4$ (the weight of 0) \times 2
1	—————	its weight is $2^6 = 2^5$ (the weight of 0) \times 2
0	—————	its weight is $2^7 = 2^6$ (the weight of 1) \times 2

Therefore, $01001100B = 0 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$. For value N from 0 to 1, its binary expression is:

$$N = a_{-1} \times 2^{-1} + a_{-2} \times 2^{-2} + a_{-3} \times 2^{-3} + a_{-4} \times 2^{-4} + \dots + a_{-n+1} \times 2^{-n+1} + a_{-n} \times 2^{-n}$$

The binary expression of 0.1101B is:

$$\begin{aligned} 0.1101B &= 0.1000 + 0.0100 + 0.0000 + 0.0001 \\ &= 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} \end{aligned}$$

2.1.2 Octal

Because a base is the number of symbols used in a numerical system, an octal

numerical system therefore has 8 symbols including 0, 1, 2, 3, 4, 5, 6, and 7, and its base is 8. In a numerical system, each weight is the multiple of the base of the previous weight. For further understanding, we make an example, 37014O:

4	—————	its weight is 8^0
1	—————	its weight is $8^1 = 8^0$ (the weight of 4) $\times 8$
0	—————	its weight is $8^2 = 8^1$ (the weight of 1) $\times 8$
7	—————	its weight is $8^3 = 8^2$ (the weight of 0) $\times 8$
3	—————	its weight is $8^4 = 8^3$ (the weight of 7) $\times 8$

Therefore, $37014O = 3 \times 8^4 + 7 \times 8^3 + 0 \times 8^2 + 1 \times 8^1 + 4 \times 8^0$

For value N from 0 to 1, its octal expression is:

$$N = a_{-1} \times 8^{-1} + a_{-2} \times 8^{-2} + a_{-3} \times 8^{-3} + a_{-4} \times 8^{-4} + \dots + a_{-n+1} \times 8^{-n+1} + a_{-n} \times 8^{-n}$$

Thus, the octal of 0.2154O is:

$$\begin{aligned} 0.2154O &= 0.2000 + 0.0100 + 0.0050 + 0.0004 \\ &= 2 \times 8^{-1} + 1 \times 8^{-2} + 5 \times 8^{-3} + 4 \times 8^{-4} \end{aligned}$$

2.1.3 Hexadecimal

A base of a numerical system is the number of symbols used in the system. For a hexadecimal system, its base is 16, having 16 symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. The letters A, B, C, D, E, and F represent the values of 10, 11, 12, 13, 14, and 15. Each weight is the multiple of the base of the previous weight. 702a4cH, for example, is a six-digit hexadecimal number.

c	—————	its weight is 16^0
4	—————	its weight is $16^1 = 16^0$ (the weight of c) $\times 16$
a	—————	its weight is $16^2 = 16^1$ (the weight of 4) $\times 16$
2	—————	its weight is $16^3 = 16^2$ (the weight of a) $\times 16$
0	—————	its weight is $16^4 = 16^3$ (the weight of 2) $\times 16$
7	—————	its weight is $16^5 = 16^4$ (the weight of 0) $\times 16$

Thus, 702a4cH is $7 \times 16^5 + 0 \times 16^4 + 2 \times 16^3 + a \times 16^2 + 4 \times 16^1 + c \times 16^0$.

For value N from 0 to 1, its hexadecimal expression is as below:

$$N = a_{-1} \times 16^{-1} + a_{-2} \times 16^{-2} + a_{-3} \times 16^{-3} + a_{-4} \times 16^{-4} + \dots + a_{-n+1} \times 16^{-n+1} + a_{-n} \times 16^{-n}$$

The hexadecimal expression of 0.2c09H is:

$$\begin{aligned} 0.2c09H &= 0.2000 + 0.0c00 + 0.0000 + 0.0009 \\ &= 2 \times 16^{-1} + c \times 16^{-2} + 0 \times 16^{-3} + 9 \times 16^{-4} \end{aligned}$$

2.2 Numerical System Conversion

Usually people use decimal systems for daily bases, but computers use binary or hexadecimal systems. It is unavoidable to make a conversion between various numerical systems. Normally we could find binary mutually exchanging with decimal, binary mutually converting with hexadecimal, and hexadecimal mutually converting with decimal.



2.2.1 Binary vs. Decimal Conversion

❖ Binary-to-Decimal Conversion

It is very easy to convert an r-base number to a decimal number. It only needs to sum up the products of all digits multiplied by weight values. For example:

$$\begin{aligned}
 101011.011\text{B} &= 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} \\
 &= 1 \times 32 + 1 \times 8 + 1 \times 2 + 1 + 1 \times 0.25 + 1 \times 0.125 \\
 &= 43.375\text{D}
 \end{aligned}$$

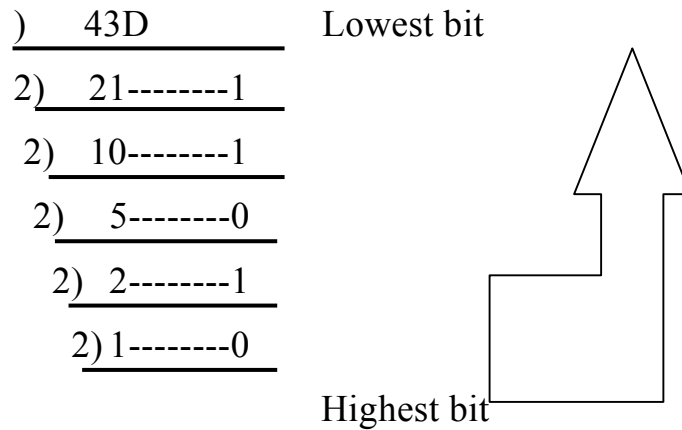
❖ Decimal-to-Binary Conversion

In general, a value N that we usually called is a decimal. To make a conversion from decimal to binary, we could use binary expression to find out each “a” factor. The conversion process is shown as below:

$$\begin{aligned}
 N &= a_n \times 2^n + a_{n-1} \times 2^{n-1} + a_{n-2} \times 2^{n-2} + a_{n-3} \times 2^{n-3} + \dots + a_1 \times 2^1 + a_0 \times 2^0 \\
 &= (a_n \times 2^{n-1} + a_{n-1} \times 2^{n-2} + a_{n-2} \times 2^{n-3} + a_{n-3} \times 2^{n-4} + \dots + a_1) \times 2 + a_0 \\
 &\quad \text{(After N divided by 2, and the remainder is } a_0) \\
 &= ((a_n \times 2^{n-2} + a_{n-1} \times 2^{n-3} + a_{n-2} \times 2^{n-4} + a_{n-3} \times 2^{n-5} + \dots + a_2) \times 2 + a_1) \times 2 + a_0 \\
 &\quad \text{(After the quotient divided by 2, and the remainder is } a_{-1}) \\
 &= (((a_n \times 2^{n-3} + a_{n-1} \times 2^{n-4} + a_{n-2} \times 2^{n-5} + \dots + a_3) \times 2 + a_{-2}) \times 2 + a_{-1}) \times 2 + a_0
 \end{aligned}$$

After the new quotient divided by 2, and the remainder is a_{-2} . We then proceed the same process until the quotient is smaller than 2.

For further explanation, we make an example, 43D converting to binary, as below:



$$43D = 101011B$$

2.2.2 Octal-to-Decimal Conversion

❖ Octal-to-Decimal Conversion

We already know it is easy to convert an r-base number to a decimal number and it only needs to sum up products of all digits multiplied by weight values. Thus,

$$\begin{aligned}
 370.14O &= 3 \times 8^2 + 7 \times 8^1 + 0 \times 8^0 + 1 \times 8^{-1} + 4 \times 8^{-2} \\
 &= 3 \times 64 + 7 \times 8 + 1 \times 8^{-1} + 4 \times 8^{-2} \\
 &= 192 + 56 + 0.125 + 0.0625 \\
 &= 248.1875D
 \end{aligned}$$

❖ Decimal-to-Octal Conversion

In general, a value N that we usually called is a decimal. To make a conversion from decimal to octal, we could use octal expression to find out each “a” factor. The conversion process is shown as below:

$$\begin{aligned}
 N &= a_n \times 8^n + a_{n-1} \times 8^{n-1} + a_{n-2} \times 8^{n-2} + a_{n-3} \times 8^{n-3} + \dots + a_1 \times 8^1 + a_0 \times 8^0 \\
 &= (a_n \times 8^{n-1} + a_{n-1} \times 8^{n-2} + a_{n-2} \times 8^{n-3} + a_{n-3} \times 8^{n-4} + \dots + a_1) \times 8 + a_0
 \end{aligned}$$



$$\begin{aligned}
& \text{(After } N \text{ divided by } 8, \text{ and the remainder is } a_0) \\
& = ((a_{-n} \times 8^{n-2} + a_{-n-1} \times 8^{n-3} + a_{-n-2} \times 8^{n-4} + a_{-n-3} \times 8^{n-5} + \dots + a_{-2}) \times 8 + a_1) \times 8 + \\
& \quad a_0 \quad \text{(After quotient is divided by } 8, \text{ and the remainder is } a_1) \\
& = (((a_{-n} \times 8^{n-3} + a_{-n-1} \times 8^{n-4} + a_{-n-2} \times 8^{n-5} + \dots + a_{-3}) \times 8 + a_{-2}) \times 8 + a_1) \times 8 + a_0
\end{aligned}$$

The remainder of the new quotient divided by 8 is a_{-2} ; follow the same process until quotient is smaller than 8.

For example, 243D converting to an octal number is as below:

$$\begin{array}{r}
8) \quad 243D \\
\hline
8) \quad 30\text{-----}3 \\
\hline
\quad 3\text{-----}6 \\
\hline
243D = 363O
\end{array}$$

2.2.3 Hexadecimal-to-Decimal Conversion

❖ Hexadecimal-to-Decimal Conversion

We already know it is easy to convert an r-base number to a decimal number and it only needs to sum up products of all digits multiplied by weight values. Thus,

$$\begin{aligned}
370.14 \text{ H} &= 3 \times 16^2 + 7 \times 16^1 + 0 \times 16^0 + 1 \times 16^{-1} + 4 \times 16^{-2} \\
&= 3 \times 256 + 7 \times 16 + 1 \times 16^{-1} + 4 \times 16^{-2} \\
&= 768 + 112 + 0.0625 + 0.015625 \\
&= 880.078125D
\end{aligned}$$

❖ Decimal-to-Hexadecimal Conversion

Similarly, to make a conversion from decimal to hexadecimal, we could use hexadecimal expression to find out each “a” factor. The conversion process is shown as below:



$$\begin{aligned}
N &= a_{-n} \times 16^n + a_{-n-1} \times 16^{n-1} + a_{-n-2} \times 16^{n-2} + a_{-n-3} \times 16^{n-3} + \dots + a_1 \times 16^1 + a_0 \times 16^0 \\
&= (a_{-n} \times 16^{n-1} + a_{-n-1} \times 16^{n-2} + a_{-n-2} \times 16^{n-3} + a_{-n-3} \times 16^{n-4} + \dots + a_1) \times 16 + a_0 \\
&\quad \text{(After } N \text{ divided by } 16, \text{ the remainder is } a_0) \\
&= ((a_{-n} \times 16^{n-2} + a_{-n-1} \times 16^{n-3} + a_{-n-2} \times 16^{n-4} + a_{-n-3} \times 16^{n-5} + \dots + a_2) \times 16 + a_1) \times 16 + a_0 \\
&\quad \text{(After the quotient is divided by } 16, \text{ and the remainder is } a_1) \\
&= (((a_{-n} \times 16^{n-3} + a_{-n-1} \times 16^{n-4} + a_{-n-2} \times 16^{n-5} + \dots + a_3) \times 16 + a_2) \times 16 + a_1) \times 16 + a_0 \\
&\quad \text{(After the new quotient is divided by } 16, \text{ the remainder is } a_2)
\end{aligned}$$

Follow the procedure until the quotient is smaller than 16.

For example, 542D converts to a hexadecimal number as follows:

$$\begin{array}{r} 16) \quad 542D \\ \hline 16) \quad 33\text{-----}E \\ \hline \quad \quad 2\text{-----}1 \\ \quad \quad 542D = 21eH \end{array}$$

2.2.4 Binary-to-Octal Conversion

❖ Binary-to-Octal Conversion

Because an octal number could be referred to three binary numbers, shown in table 2.2. Therefore, when converting a binary number to an octal number, it only needs to partition into groups with 3 digits in a group from the right to the left of the binary number. If there are not enough 3 digits in the last group, add “0” on the left side of the last digit. The divided groups could be orderly changed to an octal digit by looking up a cross-reference list. For example:



1100110001B
↓
001 100 110 001 B
↓ ↓ ↓ ↓
1 4 6 1 O

Table 2.2 Octal-to-Binary cross-reference list

Octal	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

❖ Octal-to-Binary Conversion

Conversely, for an octal-to-binary conversion, it only needs to follow the cross-reference list shown as above to convert an octal digit to three binary digits.

For example:

3 5 7 6 3 O
↓ ↓ ↓ ↓ ↓
011 101 111 110 011 B



2.2.5 Binary-to-Hexadecimal Conversion

❖ Binary-to-Hexadecimal Conversion

Because a hexadecimal number could be referred to four binary numbers, shown in table 2.3. Therefore, when converting a binary number to an hexadecimal number, it only needs to partition into groups with 4 digits from the right to the left of the binary number. If there are not enough 4 digits in the last group, add “0” on the left side of the last digit. The divided groups could be orderly referred to a hexadecimal digit by looking up a cross-reference list. For example:

$$\begin{array}{ccccccc} & & & & & & 1101110001\text{B} \\ & & & & & & \downarrow \\ & 0011 & & 0111 & & 0001 & \text{B} \\ & \downarrow & & \downarrow & & \downarrow & \\ & 3 & & 7 & & 1 & \text{H} \end{array}$$

Table 2.3 Hexadecimal-to-Binary cross-reference list

Hexadecimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000



9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

❖ Hexadecimal-to-Binary Conversion

Conversely, for a hexadecimal-to-binary conversion, it only needs to follow the cross-reference list shown as above to convert a hexadecimal number to four binary digits. For example:

3	C	7	E	3	H
↓	↓	↓	↓	↓	
0011	1100	0111	1110	0011	B

2.3 Numerical Complement

2.3.1 9's Complement

9's complement is a diminished complement of a decimal number. 9's complement of a number is determined by subtracting the number from 9.



(1) 9's complement of 4

$$\begin{array}{r} 9 \\ - 4 \\ \hline 5 \end{array}$$

(2) 9's complement of 39

$$\begin{array}{r} 99 \\ - 39 \\ \hline 60 \end{array}$$

(3) 9's complement of 517

$$\begin{array}{r} 999 \\ - 517 \\ \hline 482 \end{array}$$

(4) 9's complement of 5408

$$\begin{array}{r} 9999 \\ - 5408 \\ \hline 4591 \end{array}$$

2.3.2 10's Complement

10's complement is another complement of a decimal complement number. 10's complement of a number is the difference between 10 and the number. The other definition of 10's complement of a number is to subtract the number from 9 and plus 1. For example:

(1) 10's complement of 4 (2) 10's complement of 39 (3) 10's complement of 517

9	99	999
- 4	- 39	- 517
<hr/>	<hr/>	<hr/>
5	60	482
+ 1	+ 1	+ 1
<hr/>	<hr/>	<hr/>
6	61	483

2.3.3 1's Complement

1's complement is a diminished complement of a binary number. 1's complement of a number is determined by subtracting the number from 1. For example:



(1) 1's complement of 010B

$$\begin{array}{r} 111 \\ - 010 \\ \hline 101 \end{array}$$

(2) 1's complement of 1001B

$$\begin{array}{r} 1111 \\ - 1001 \\ \hline 0110 \end{array}$$

(3) 1's complement of 10101110B

$$\begin{array}{r} 11111111 \\ - 10101110 \\ \hline 01010001 \end{array}$$

2.3.4 2's Complement

2's complement is another binary complement number. It is defined as “1's complement of a number plus 1”. For example:

(1) 1's complement of 010B

$$\begin{array}{r} 111 \\ - 010 \\ \hline 101 \\ + 1 \\ \hline 110 \end{array}$$

(2) 1's complement of 1001B

$$\begin{array}{r} 1111 \\ - 1001 \\ \hline 0110 \\ + 1 \\ \hline 0111 \end{array}$$

(3) 1's complement of 10101110B

$$\begin{array}{r} 11111111 \\ - 10101110 \\ \hline 01010001 \\ + 1 \\ \hline 01010010 \end{array}$$

2.4 Negative Binary Number Expression

In digital systems such as digital calculators and digital computers, processing positive and negative numbers is unavoidable, and, therefore, it is necessary to have proper expression on binary positive/negative numbers. Totally there are 3 ways to express positive/negative binary numbers described as below:

❖ Signed-Magnitude Expression :

The highest bit of a binary number is sign bit, and the rest are magnitude bits to

indicate the number. For sign bits, “0” is a positive number and “1” is a negative number. For example:

011101 is 29
111101 is -29

Table 2.4 is a numerical cross-reference list of 8-bit signed-magnitude expression. Its numerical expression is from +127 to 0 and from -0 to -127. “0” in this table could be expressed as +0 (00000000) and -0 (10000000).

Table 2.4 Numerical cross-reference list of 8-bit signed-magnitude expression

+/- Number	8-bit Signed-magnitude Expression
+127	01111111
+126	01111110
+125	01111101
...	...
+4	00000100
+3	00000011
+2	00000010
+1	00000001
+0	00000000
-0	10000000
-1	10000001
-2	10000010
-3	10000011
-4	10000100
...	...
-125	11111101
-126	11111110
-127	11111111



❖ 1's Complement Expression:

1's complement is the way to express a negative number. For example:

011101 is 29

100010 is -29

100010 is 1's complement of 011101.

Table 2.5 is a numerical cross-reference list of 8-bit 1's complement expression. Its numerical expression is from +127 to 0 and from -0 to -127. "0" is expressed as +0 (00000000) and -0 (11111111).

Table 2.5 Numerical cross-reference list of 8-bit 1's complement expression

+/- Number	8-bit 1's Complement Expression
+127	01111111
+126	01111110
+125	01111101
...	...
+4	00000100
+3	00000011
+2	00000010
+1	00000001
0	00000000
-0	11111111
-1	11111110
-2	11111101
-3	11111100
-4	11111011



...	...
-125	10000010
-126	10000001
-127	10000000

❖ 2's Complement Expression

2's complement is another way to express a negative number. For example:

011101 is 29

100011 is -29

100011 is 2's complement of 011101.

Table 2.6 is a numerical cross-reference list of 8-bit 2's complement expression. Its numerical expression is from +128 to 0 and from 0 to -127. "0" in this cross-reference list can be only expressed as 0 (00000000). Obviously, 2's complement expression can have one more number for its numerical expression than the other two described in section 2.4 and 2.5.

Table 2.6 Numerical cross-reference list of 8-bit 2's complement expression

+/- Number	8-bit 2's Complement Expression
+128	10000000
+127	01111111
+126	01111110
+125	01111101
...	...
+4	00000100
+3	00000011



+2	00000010
+1	00000001
0	00000000
-1	11111111
-2	11111110
-3	11111101
-4	11111100
...	...
-125	10000011
-126	10000010
-127	10000001

2.5 Binary Arithmetic Operations

Like decimal, in 2's complement system, binary can have arithmetic operations: addition, subtraction, multiplication, and division. Their arithmetic operations are introduced in the following sections.

2.5.1 Binary Addition

Binary add operation has four basic rules as below:

$$\begin{array}{r}
 0B \\
 + 0B \\
 \hline
 0B
 \end{array}
 \qquad
 \begin{array}{r}
 0B \\
 + 1B \\
 \hline
 1B
 \end{array}
 \qquad
 \begin{array}{r}
 1B \\
 + 0B \\
 \hline
 1B
 \end{array}
 \qquad
 \begin{array}{r}
 1B \\
 + 1B \\
 \hline
 10B \\
 \uparrow \text{Carry}
 \end{array}$$

Referring to the addition of multi-digit binary, we have to take “carry bit” into account in the operation. For example:



$$\begin{array}{r}
 1\ 0\ 0\ 1\ 1\ 1\ 0\ 1\text{B} \\
 +\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\text{B} \\
 \hline
 1\ 1 \quad \leftarrow \text{Carry Bit} \\
 1\ 1\ 0\ 1\ 0\ 1\ 1\ 1\text{B}
 \end{array}$$

2.5.2 Binary Subtraction

As add operation, binary subtraction also has four basic rules:

$$\begin{array}{r}
 0\text{B} \\
 -\ 0\text{B} \\
 \hline
 0\text{B}
 \end{array}
 \quad
 \begin{array}{r}
 0\text{B} \\
 -\ 1\text{B} \\
 \hline
 11\text{B} \\
 \uparrow \\
 \text{Borrow}
 \end{array}
 \quad
 \begin{array}{r}
 1\text{B} \\
 -\ 0\text{B} \\
 \hline
 1\text{B}
 \end{array}
 \quad
 \begin{array}{r}
 1\text{B} \\
 -\ 1\text{B} \\
 \hline
 0\text{B}
 \end{array}$$

While subtracting multi-digit binary numbers, we have to consider “borrow bit”.

For example:

$$\begin{array}{r}
 1\ 0\ 0\ 1\ 1\ 1\ 0\ 1\text{B} \\
 -\ 0\ 1\ 1\ 1\ 1\ 0\ 1\ 0\text{B} \\
 \hline
 1\ 1 \quad 1 \quad \leftarrow \text{Borrow Bit} \\
 0\ 0\ 1\ 0\ 0\ 0\ 1\ 1\text{B}
 \end{array}$$

❖ Subtraction of 2's Complement

In binary numerical arithmetic, we could use 2's complement addition to finish subtraction by adding minuend and 2's complement of subtrahend together and then ignoring end-round carry. For example:



$$\begin{array}{r}
 1\ 0\ 0\ 1\ 1\ 1\ 0\ 1\text{B} \\
 -\ 0\ 1\ 1\ 1\ 1\ 0\ 1\ 0\text{B} \\
 \hline
 1\ 1 \qquad 1 \qquad \leftarrow \text{Borrow} \\
 0\ 0\ 1\ 0\ 0\ 0\ 1\ 1\text{B}
 \end{array}
 \rightarrow
 \begin{array}{r}
 1\ 0\ 0\ 1\ 1\ 1\ 0\ 1\text{B} \\
 +\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 0\text{B} \\
 \hline
 1 \quad 1\ 1\ 1 \qquad \leftarrow \text{Carry} \\
 1\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 1\text{B} \\
 \uparrow \\
 \text{Ignoring end-around carry}
 \end{array}$$

2.5.3 Binary Multiplication

As previous arithmetic operations, binary multiplication has four basic rules:

$$\begin{array}{r}
 0\text{B} \\
 \times 0\text{B} \\
 \hline
 0\text{B}
 \end{array}
 \qquad
 \begin{array}{r}
 0\text{B} \\
 \times 1\text{B} \\
 \hline
 0\text{B}
 \end{array}
 \qquad
 \begin{array}{r}
 1\text{B} \\
 \times 0\text{B} \\
 \hline
 0\text{B}
 \end{array}
 \qquad
 \begin{array}{r}
 1\text{B} \\
 \times 1\text{B} \\
 \hline
 1\text{B}
 \end{array}$$

To multiple multi-digit binary numbers, the calculation is the same as decimal multiplication. For example:

$$\begin{array}{r}
 \qquad \qquad \qquad 1\ 0\ 0\ 1\ 1\ 1\text{B} \quad \text{Multiplicand} \\
 \times \qquad \qquad 0\ 1\ 0\ 0\ 1\ 0\text{B} \quad \text{Multiplier} \\
 \hline
 \qquad \qquad \qquad 0\ 0\ 0\ 0\ 0\ 0 \quad 1^{\text{st}} \text{ Partial Product} \\
 \qquad \qquad 1\ 0\ 0\ 1\ 1\ 1 \quad 2^{\text{nd}} \text{ Partial Product} \\
 \qquad \qquad 0\ 0\ 0\ 0\ 0\ 0 \quad 3^{\text{rd}} \text{ Partial Product} \\
 \qquad 0\ 0\ 0\ 0\ 0\ 0 \quad 4^{\text{th}} \text{ Partial Product} \\
 \qquad 1\ 0\ 0\ 1\ 1\ 1 \quad 5^{\text{th}} \text{ Partial Product} \\
 +\ 0\ 0\ 0\ 0\ 0\ 0 \quad 6^{\text{th}} \text{ Partial Product} \\
 \hline
 0\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 0\text{B} \quad \text{Final Product}
 \end{array}$$



2.5.4 Binary Division

Like decimal division, binary division can use long division. For example:

(1)

Divisor	11 B	$ \begin{array}{r} 10B \\ 11B \overline{) 110B} \\ \underline{11} \\ 00B \end{array} $	Quotient Dividend Remainder
---------	------	---	---

(2)

Divisor	11 B	$ \begin{array}{r} 11B \\ 11B \overline{) 1001B} \\ \underline{11} \\ 11 \\ \underline{11} \\ 00B \end{array} $	
---------	------	--	--

(3)

Divisor	100 B	$ \begin{array}{r} 10.1B \\ 100B \overline{) 1010.0B} \\ \underline{100} \\ 100 \\ \underline{100} \\ 000B \end{array} $	
---------	-------	---	--

2.6 Binary - coded Decimal (BCD) Code

In digit logic circuit, all arithmetic processes are completed by binary. However, we are used to decimal systems in our daily lives. In section 2.2, we have already

known binary-decimal conversion, and it is truly uneasy. Therefore, a combination between binary and decimal codes is made, called “BCD Code” (Binary Coded Decimal). For coding, it only uses 4-bit binary numbers from 0 to 9 shown as in table 2.7.

Table 2.7 BCD Code-to-Decimal cross-reference list

BCD Code	Decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9

In another words, each digit of a decimal number could be expressed as a 4-bit binary number in BCD code. For example: a decimal number of 5168D could be referred to a 16-bit binary number, 0101000101101000B, in BCD code. The way to convert numbers is as below:

3	5	7	6	3D
↓	↓	↓	↓	↓
0011	0101	0111	0110	0011B



2.7 Review

Please answer the following questions to review this chapter:

- ❑ Do you know what numerical systems are? Which numerical system is mostly used in our daily lives?
- ❑ Do you know what numbers or digits can be found in an octal number?
- ❑ Do you know how to get 10's complement of a decimal number?
- ❑ Do you know how to get 2's complement of a binary number?
- ❑ Do you know how to express a negative number of a binary number?
- ❑ Do you know the four basic rules of all binary arithmetic operations?
- ❑ Do you know what BCD code is?

CHAPTER 3

Basic Logic Theories



LEAP



In this chapter, we will introduce some basic logic theories such as Boolean algebra and theorem, Boolean algebra simplification, and basic logic gates, etc. Two major digit logic circuits, combinational and sequential logic circuits, also will be introduced in Chapter 5 and 6 in this book separately.

3.1 Boolean Algebra

Boolean algebra is different from general algebra. “0” and “1” are its only algebra numbers. “NOT”, “AND”, and “OR” are its basic algebra operations shown in Table 3.1. Therefore, “Boolean algebra” is easy. A great English mathematician George Boolean publishes it in 1854. “1” and “0” can be seen as numbers or logic status. Like general algebra, Boolean algebra has variables that are usually expressed in letters such as A, B, C and D in Table 3.1. Those variables can only represent 0/ 1 or false/true, respectively.

From basic operations, Boolean algebra shown in Table 3.2 can also define some other compound operations.

❖ Basic Operations

Table 3.1 Basic operations in Boolean algebra

Abbreviation	Symbol of Operation	Example	Explanation
NOT Operation	'	$X = A'$	X is opposite to A.
AND Operation	•	$X = A \cdot B \cdot C$	While all of A, B, and C are equal to “1”, X is “1”; otherwise, X is “0”.



OR Operation	+	$X = A + B + C + D$	At least one of A, B, C, and D is “1”, X is “1”; otherwise, X is “0”.
--------------	---	---------------------	---

Note : In Boolean Algebra, operational symbol “ \cdot ” is usually omitted for convenience.

❖ Further Developed Operations

Table 3.2 Other Boolean algebra's operations

Abbreviation	Symbol of Operation	Example	Explanation
NAND Operation		$X = (A \cdot B \cdot C)'$	After AND operation, perform NOT operation.
NOR Operation		$X = (A + B + C)'$	After OR operation, perform NOT operation.
XOR Operation (Exclusive OR)	\oplus	$X = A \oplus B$ $= A'B + AB'$	1. NOT operation on A/B first. 2. AND operation on A'B (AB') later. 3. And then OR operation on A'B and AB'
NXOR Operation (Inclusive OR)		$X = (A \oplus B)'$	After XOR operation, complete NOT operation.

In daily bases, we could also find some Boolean algebra examples. For example, If it is “raining” (expressed by variable A) outside and I am “going out for business” (expressed by variable B), I have to “bring my umbrella” (expressed by variable Y). Therefore, “bring my umbrella” becomes valid only if the assumptions of “raining” and “going out for business” are all occurred, and this is AND operation; that is, $Y = AB$. For another example, if going to Dr. Sun Yat-Sen Memorial Hall, we could “drive our own car” (expressed by variable A), “ride a motorcycle” (expressed



variable B), or “take a bus” (expressed by variable C). All transportation can take us to our destination (expressed by variable Z). This is what we called “OR operation”; that is, $Z = A + B + C$. We could take one of the three ways to go to Dr. Sun Yat-Sen Memorial Hall.

3.1.1 Truth Table and Boolean Algebra Expression

Assume there is an issue of “if tomorrow is Sunday (A) and John asks me to see a movie (B), I will go to see a movie with him (X).” We could use a table called “Truth table” to list all the possible situations about the issue.

Table 3.3 Examples by Truth table

Input		Output X	Explanation
A	B		
0	0	0	Since tomorrow is not Sunday and John does not ask me out for a movie, I will not go to see a movie with him.
0	1	0	Though John asks me out, I will not go to see a movie with him since tomorrow is not Sunday.
1	0	0	Though tomorrow is Sunday, I will not go to see a movie with John since he does not ask me out.
1	1	1	Since tomorrow is Sunday and John asks me out, I will go to see a movie with him.



In Table 3.3, “0” in column A means “tomorrow is not Sunday” and “1” means “tomorrow is Sunday”. “0” in column B means “John does not ask me out to see a movie” and “1” means “John asks me out to see a movie”. “0” in column X means “I will not go to see a movie with John” and “1” means “I will go to see a movie with John”. From this example, we could understand that Truth table could give us a clear idea. In the above table, A and B have two numbers (0 and 1) separately. Therefore, totally there are $2 \times 2 = 4$ different possible situations.

The content of Truth table can be expressed by algebra expression. However, how could we express an algebra expression of X? The answer is simple! It only needs to find out all combinations causing output X equal to “1” and then perform “OR” operation to link all combinations together. Therefore, $X = AB$ (omitted the operational symbol of “ \cdot ”). To verify if X is what we describe in Truth table, we could use 0 or 1 to substitute A and B.

Let’s make another example for further understanding of how to use a Truth table. Assume there is a committee that has 5 members going to vote. To pass the voting issue, the committee requires reaching an agreement from a half of the members without waiving any right to vote. Therefore, how many different situations are there after voting the issue? Because each member has two choices: agree (expressed by “1”) or disagree (expressed by “0”), totally there are $2 \times 2 \times 2 \times 2 \times 2 = 2^5 = 32$ different results from voting, and the Truth table is shown as Table 3.4:



Table 3.4 Truth table for 5-member committee voting an issue

Committee					Voting Result
A	B	C	D	E	R
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	0
0	0	1	0	0	0
0	0	1	0	1	0
0	0	1	1	0	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	0	1	0
1	0	0	1	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	1	1
1	0	1	1	0	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	0	1	1
1	1	0	1	0	1
1	1	0	1	1	1
1	1	1	0	0	1
1	1	1	0	1	1
1	1	1	1	0	1
1	1	1	1	1	1



Then we come out a question “How to express algebra expression R?” and the answer is quite simple. It only needs to find out all combinations making output R equal to “1” and then adopt “OR” operations to include all combinations together. Therefore:

$$R = A'B'CDE + A'BC'DE + A'BCD'E + A'BCDE' + A'BCDE + AB'C'DE + AB'CDE' + AB'CD'E + AB'CDE + ABC'D'E + ABC'DE' + ABC'DE + ABCD'E' + ABCD'E + ABCDE' + ABCDE$$

3.1.2 Boolean Theorems and Boolean Algebra Laws

❖ Boolean Theorems

Boolean theorems are the identities such as Equations 3.1~3.9. They are developed from Boolean algebra operation. We could use those identities also called “Boolean theorems” to simplify the complicated Boolean algebra expressions. For example, in section 3.1.1, algebra expression R is not the simplest expression since it dose not come with the least variables and operations.

1. With 0/1 Operations

$$\text{OR Operation: } A+0 = A \quad \dots\dots\dots (3.1)$$

$$\text{AND Operation: } A \cdot 1 = A \quad \dots\dots\dots (3.2)$$

$$\text{OR Operation: } A+1 = 1 \quad \dots\dots\dots (3.3)$$

$$\text{AND Operation: } A \cdot 0 = 0 \quad \dots\dots\dots (3.4)$$

2. Equal theorem

$$\text{OR Operation: } A + A = A \quad \dots\dots\dots (3.5)$$

$$\text{AND Operation: } A \cdot A = A \quad \dots\dots\dots (3.6)$$

3. Complementary theorem

$$\text{OR Operation: } A + A' = 1 \quad \dots\dots\dots (3.7)$$

$$\text{AND Operation: } A \cdot A' = 0 \quad \dots\dots\dots (3.8)$$

4. Involution theorems

$$(A')' = A \quad \dots\dots\dots (3.9)$$

The further explanation of those theorems are described as below:

1. With 0/1 Operation

- (1) $A + 0 = A$: The OR operation of algebra A with “0” is A. When $A = 1$, OR operation of algebra A with “0” is 1; similarly, when $A = 0$, OR operation of algebra A with “0” is 0.
- (2) $A \cdot 1 = A$: The AND operation of algebra A with “1” is A. When $A = 1$, AND operation of algebra A with “1” is 1; similarly, when $A = 0$, AND operation of algebra A with “1” is 0.
- (3) $A + 1 = 1$: The OR operation of algebra A with “1” is 1. When $A = 1$, OR operation of algebra A with “1” is 1; similarly, when $A = 0$, OR operation of algebra A with “1” is still 1.
- (4) $A \cdot 0 = 0$: The AND operation of algebra A with “0” is 0. When $A = 1$, AND operation of algebra A with “0” is 0; similarly, when $A = 0$, AND operation of algebra A with “0” is still 0.

2. Equal theorem: Equal theorem can be divided into to categories: “OR equal theorem” and “AND equal theorem”.

- (1) OR equal theorem: OR operation of algebra A with A is A, and OR equal theorem is the result of the OR operation. Therefore, OR equal theorem is algebra A. When $A = 1$ and $A = 1$, OR operation is 1; similarly $A = 0$ and $A = 0$, OR operation is 0.
- (2) AND equal theorem: AND operation of algebra A with A is A. AND equal theorem is the result of the AND operation. Therefore, AND equal theorem is algebra A. When $A = 1$ and $A = 1$, AND operation is 1; similarly, when $A = 0$ and $A = 0$, AND operation is 0.

3. Complementary theorem: Complementary theorem can also be divided into two categories: “OR complementary theorem” and “AND complementary theorem”.
 - (1) OR complementary theorem: OR operation of algebra A with A' is 1.
OR complementary theorem is the result of the OR operation. Therefore, OR complementary theorem is 1. When $A = 1$ and $A' = 0$, OR operation of A with A' is 1; similarly, when $A = 0$ and $A' = 1$, the OR operation is still 1.
 - (2) AND complementary theorem: AND operation of algebra A with A' is 0.
AND complementary theorem is the result of AND operation of algebra A with A'. Therefore, AND complementary theorem is 0. When $A = 1$ and $A' = 0$, the AND operation is 0; similarly, when $A = 0$ and $A' = 1$, the AND operation is still 0.
4. Involution theorems : In the theorems, algebra A will back to its original value after running NOT operation twice.

❖ Boolean Algebra Law

In Boolean algebra operation, we still need to follow some laws described as bellow:

1. Commutative law

$$\text{OR operation: } A+B = B+A \quad \dots\dots\dots (3.10)$$

$$\text{AND operation: } A \cdot B = B \cdot A \quad \dots\dots\dots (3.11)$$

2. Associative law

$$\text{OR operation: } A + (B + C) = (A + B) + C \quad \dots\dots\dots (3.12)$$

$$\text{AND operation: } (A \cdot B) \cdot C = A \cdot (B \cdot C) \quad \dots\dots\dots (3.13)$$

3. Distributive law

$$\text{AND operation: } A \cdot (B + C) = (A \cdot B) + (A \cdot C) \quad \dots\dots\dots (3.14)$$



$$\text{OR operation: } A + (B \cdot C) = (A + B) \cdot (A + C) \dots\dots\dots (3.15)$$

The verification of Equation 3.14 is shown in Table 3.5 as below:

Table 3.5 Verification of Equation 3.14 by Truth table

Input			Output	
A	B	C	$A \cdot (B + C)$	$(A \cdot B) + (A \cdot C)$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

The following is the verification of Equation 3.15:

$$\begin{aligned}
 (A + B) \cdot (A + C) &= A (A + C) + B (A + C) \dots\dots\dots \text{After apply AND} \\
 &\hspace{15em} \text{distributive law} \\
 &= AA + AC + AB + BC \dots\dots\dots \text{After reapply AND} \\
 &\hspace{15em} \text{distributive law} \\
 &= A + AC + AB + BC \dots\dots\dots \text{After apply equal theorem} \\
 &= A \cdot 1 + AC + AB + BC \dots\dots\dots \text{After apply Equation 3.2} \\
 &= A (1 + C + B) + BC \dots\dots\dots \text{After reapply AND} \\
 &\hspace{15em} \text{distributive law} \\
 &= A + BC \dots\dots\dots \text{After apply Equation 3.3}
 \end{aligned}$$



4. Elimination law

$$\text{OR operation: } A + (A \cdot B) = A \quad \dots\dots\dots (3.16)$$

$$\text{AND operation: } A \cdot (A + B) = A \quad \dots\dots\dots (3.17)$$

5. Demorgan's theorem

$$\text{OR operation: } (A + B)' = A' \cdot B' \quad \dots\dots\dots (3.18)$$

$$\text{AND operation: } (A \cdot B)' = A' + B' \quad \dots\dots\dots (3.19)$$

Note 1: In Boolean algebra, we usually omit the operation symbol “ \cdot ” for convenience.

Note 2: We could verify Commutative law, Associative law, Elimination law, and Demorgan's theorem by Truth table.

In Boolean algebra, commutative law and associative law tell us that the sequence of written variables is not related to the consequence of any operations.

6. Boolean Simplification Theorem

$$XY + XY' = X \quad \dots\dots\dots (3.20)$$

$$(X + Y)(X + Y') = X \quad \dots\dots\dots (3.21)$$

$$X + XY = X \quad \dots\dots\dots (3.22)$$

$$X(X + Y) = X \quad \dots\dots\dots (3.23)$$

$$(X + Y')Y = XY \quad \dots\dots\dots (3.24)$$

$$(XY') + Y = X + Y \quad \dots\dots\dots (3.25)$$



The verification of Equation 3.20 is shown as below:

$$\begin{aligned}
 XY + XY' &= X(Y + Y') \quad \text{After find the common factors} \\
 &= X \quad \text{After apply complementary theorem: } X + X' = 1
 \end{aligned}$$

The verification of Equation 3.21 is as below:

$$\begin{aligned}
 (X + Y)(X + Y') &= XX + XY + XY' + YY' \quad \text{after apply AND operation, adopt complementary theorem} \\
 &\quad (XX' = 0) \text{ and equal theorem } (XX = X), \text{ and then find the common factors.} \\
 &= X + X(Y + Y') \quad \text{Apply complementary theorem: } X + X' = 1 \\
 &= X + X \quad \text{Apply equal theorem: } X + X = X \\
 &= X
 \end{aligned}$$

The verification of Equation 3.22 is as below:

$$\begin{aligned}
 X + XY &= X(1 + Y) \quad \text{After find the common factors} \\
 &= X \cdot 1 \quad \text{After apply Equation 3.3} \\
 &= X \quad \text{After apply Equation 3.2}
 \end{aligned}$$

The verification of Equation 3.23 is as below:

$$X(X + Y') = X + XY' \quad \text{After apply AND}$$



$$\begin{aligned}
 & \text{distributive law} \\
 & = X \cdot 1 + XY' \dots\dots\dots \text{After apply Equation 3.2} \\
 & = X(1 + Y) + XY' \dots\dots\dots \text{After apply Equation 3.3} \\
 & = X + X(Y + Y') \dots\dots\dots \text{Apply Equation 3.2 and} \\
 & \quad \text{AND distributive law} \\
 & = X + X \dots\dots\dots \text{Apply complementary theorem:} \\
 & \quad X + X' = 1 \\
 & = X \dots\dots\dots \text{Apply equal theorem: } X + X = X
 \end{aligned}$$

The verification of Equation (3.24):

$$\begin{aligned}
 (X + Y')Y &= XY + YY' \dots\dots\dots \text{After apply AND} \\
 & \quad \text{distributive law} \\
 & = XY + 0 \dots\dots\dots \text{Apply complementary theorem:} \\
 & \quad XX' = 0 \\
 & = XY \dots\dots\dots \text{After apply Equation 3-1}
 \end{aligned}$$

The verification of Equation 3.25:

$$\begin{aligned}
 (XY') + Y &= XY' + Y(1 + X) \dots\dots\dots \text{After apply } Y \cdot 1 = Y; X + 1 = 1 \\
 & = XY' + Y + XY \dots\dots\dots \text{After apply AND} \\
 & \quad \text{distributive law} \\
 & = X(Y + Y') + Y \dots\dots\dots \text{After apply commutative law and} \\
 & \quad \text{multiplication distributive law} \\
 & = X + Y \dots\dots\dots \text{After apply complementary} \\
 & \quad \text{theorem: } Y + Y' = 1
 \end{aligned}$$

7. Consensus theorem

$$XY + YZ + X'Z = XY + X'Z \quad \text{..... (3.26)}$$

$$(X+Y)(Y+Z)(X'+Z) = (X+Y)(X'+Z) \quad \text{..... (3.27)}$$

The verification of Equation 3.26 is shown as below:

$$\begin{aligned} XY + YZ + X'Z &= XY + 1 \cdot YZ + X'Z \quad \text{..... After apply Equation 3.2} \\ &= XY + (X + X')YZ + X'Z \quad \text{..... After apply} \\ &\quad \text{complementary theorem} \\ &\quad Y + Y' = 1 \\ &= XY + XYZ + X'YZ + X'Z \quad \text{..... After apply AND} \\ &\quad \text{distributive law} \\ &= XY(1 + Z) + X'Z(Y + 1) \quad \text{..... After apply AND} \\ &\quad \text{distributive law} \\ &= XY \cdot 1 + X'Z \cdot 1 \quad \text{..... After apply Equation 3.3} \\ &= XY + X'Z \quad \text{..... After apply Equation 3.2} \end{aligned}$$

The term of “YZ “eliminated in Equation 3.26 is called “consensus”. Actually, “consensus” is a term found in both of XY and X'Z, and consequently we could eliminate “consensus” for less production terms, but how to find out “consensus” in an Equation? First, we could check if it has a variable expressed in one production term and its complement in other production term. If it has, we then eliminate the variable and its complement after multiplying both terms. For example: (XY) (X'Z) = ~~XX'~~YZ = YZ.

The following is the verification of Equation 3.27:

$$\begin{aligned} (X + Y)(Y + Z)(X' + Z) &= (Y + Z) (X + Y) (X' + Z) \\ &= (Y + Z) (X'Y + XZ) \end{aligned}$$



$$\begin{aligned}
 &= X'Y + X'YZ + XYZ + XZ \\
 &= X'Y + XZ \\
 &= (X + Y)(X' + Z)
 \end{aligned}$$

The term “Y+Z” eliminated in Equation 3.27 is called “consensus”. It actually can be found in both of terms “X + Y” and “X’ + Z”, causing a repeating status. Therefore, to reduce sum, we could eliminate “consensus” by checking if it has a variable expressed in one sum term and its complement in other term. If it does have, we could eliminate the variable and its complement after adding (OR) both terms. For example:

$$(X + Y) + (X' + Z) = \cancel{X + X'} + Y + Z = Y + Z$$

8. Multiplication and factor theorem

$$(X + Y)(X' + Z) = XZ + X'Y \quad \dots\dots\dots (3.28)$$

$$XY + X'Z = (X + Z)(X' + Y) \quad \dots\dots\dots (3.29)$$

The following is the verification of Equation 3.28:

$$\begin{aligned}
 (X + Y)(X' + Z) &= (X + Y)X' + (X + Y)Z \quad \dots\dots\dots \text{After apply AND} \\
 &\hspace{15em} \text{distributive law} \\
 &= X'Y + YZ + XZ \quad \dots\dots\dots \text{After apply} \\
 &\hspace{15em} \text{complementary theorem:} \\
 &\hspace{15em} XX' = 0 \\
 &= X'Y + XZ \quad \dots\dots\dots \text{After apply Equation 3.26} \\
 &\hspace{15em} \text{of Consensus theorem}
 \end{aligned}$$

Equation 3.29 will not be discussed in this book. We would like to invite readers to verify by themselves.



9. Duality theorem

$$(x + y + z + \cdots)^D = xyz \cdots \quad (3.30)$$

$$(xyz \cdots)^D = x + y + z + \cdots \quad (3.31)$$

Duality theorem is defined as:

If $x + y + z = x \cdot z$ is true,
 $\downarrow \quad \downarrow \quad \downarrow \quad + \rightarrow \cdot, \cdot \rightarrow +, 1 \rightarrow 0, 0 \rightarrow 1$
 $x \cdot y \cdot z = x + z$ is existing.

In equal theorem, we could find out this kind of duality such as:

$$\text{OR operation: } A + A = A \quad (3.5)$$

\downarrow

$$\text{AND operation: } A \cdot A = A \quad (3.6)$$

In complementary theorem, we also could find out duality. For example:

$$\text{OR operation: } A + A' = 1 \quad (3.7)$$

\downarrow

$$\text{AND operation: } A \cdot A' = 0 \quad (3.8)$$

Duality exists in 0/1 operation, communicative law, associative law, distributive law, simplification law, multiplication and factor law, and Demorgan's theorem. Therefore, knowing duality can help readers to understand other theorems and laws more easily.

3.1.3 Types of Boolean Algebraic Expression

Boolean algebraic expression usually can be divided into two classes: SOP (Sum of Production) and POS (Production of Sum). The ways to express SOP are:

$$X = ABC + AB'$$

$$Y = AB + A'BC + BCD' + BC'$$

$$Z = AB' + A'B + C'D + CD'$$

In another words, SOP is an algebra expression that sums up several production terms together.

POS, another type of Boolean algebra expression, can be expressed as below:

$$X = (C + A) (B' + A')$$

$$Y = (B + A') (C + D' + B)$$

$$Z = (B' + A') (B + C') (D + C)$$

Like SOP, POS is an algebra expression that multiplies several sum terms together.

In general, SOP is more popular than POS because it is easier to get from Truth tables by describing questions. For example: if there is a Boolean algebra Truth table shown in Table 3.6, first we could write down all productions as AB' and $A'B$ in Table 3.6 when the output Y is 1. We then sum up all productions (OR operation) to get the SOP expression, $Y = AB' + A'B$.

Table 3.6 SOP Truth table

Input		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

$$\rightarrow AB' \quad (1)$$

$$\rightarrow A'B \quad (2)$$

❖ Minterm and Maxterm

For algebra with n variables, minterm is the production of n variables. Each of the variables can only be shown out once by its original or complementary expression. For example: all of the following productions in the algebra expression are minterms.

$$f = A'BC + AB'C' + AB'C + ABC' + ABC$$

For algebra with n variables, maxterm is the sum of n variables. Each of the variables can only be shown out once by its original or complementary expression. For example: all of the following sums in the algebra expression are maxterms.

$$f = (A + B + C)(A + B + C')(A + B' + C)$$

For Boolean algebra expression expressed by minterms, we could use the symbol “ m ” to represent each of its minterms. For example, $m_1 = A'B'C$ and $m_5 = AB'C$. Thus:

$$f = A'BC + AB'C' + AB'C + ABC' + ABC$$

↓ expressed by symbol “m”

$$f(A, B, C) = m_3 + m_4 + m_5 + m_6 + m_7$$

↓ further abbreviated

$$f(A, B, C) = \sum m(3, 4, 5, 6, 7)$$

For Boolean algebra expression expressed by maxterms, we could use the symbol “M” to represent each of its maxterms. For example: $M_1 = A' + B' + C$ and $M_5 = A + B' + C$. Thus:

$$f = (a + b + c)(a + b + c')(a + b' + c)$$

↓ expressed by symbol “M”

$$f(A, B, C) = M_5 M_6 M_7$$

↓ further abbreviated

$$f(A, B, C) = \prod M(5, 6, 7)$$

3.2 Boolean Algebra Simplification

$f(x_1, x_2, x_3, \dots, x_n)$ is a Boolean algebra which is combined with some Boolean variables $x_1, x_2, x_3, \dots, x_n$ and some algebra operations of AND, OR and NOT operations. For example:

$$f(x, y, z) = xy + x'z + y'z$$

This is a three-Boolean-variable algebra. In this algebra equation, if $x = 0$, $y = 0$, and $z = 1$, f is then equal to 1. To verify the above, we use 0 to substitute x and y and 1 to substitute z in $xy + x'z + y'z$. The verification process is shown as below:



$$\begin{aligned}
 f(0, 0, 1) &= 0 \cdot 0 + 0' \cdot 1 + 0' \cdot 1 \\
 &= 0 \cdot 0 + 1 \cdot 1 + 1 \cdot 1 \\
 &= 0 + 1 + 1 \\
 &= 1
 \end{aligned}$$

For an n -Boolean-variable Boolean algebra, we could also use Truth table to describe it. Because each Boolean variable has the numbers of “0” (false) and “1” (true), there should be 2^n combinations of variables. Table 3.7 is the Truth table of Boolean algebra equation $f(x, y, z) = xy + x'z + y'z$.

Table 3.7 Truth table of the Boolean algebra expression

$$f(x, y, z) = xy + x'z + y'z$$

X	Y	Z	$f(x, y, z)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

To here, we have to understand that there is not only one Boolean equation having the Truth table described as above (in this case, the equation that we talked before is $f(x, y, z) = xy + x'z + y'z$). In deed, $f(x, y, z) = xy + z$ also has the Truth table shown in Table 3.7. In another words, $f(x, y, z) = xy + x'z + y'z$ is not the simplest Boolean algebra equation because it has 3 productions and 6 variables, more than

the equation $f(x, y, z) = xy + z$ that has 2 productions and 3 variables. Therefore, it is necessary to simplify Boolean algebra equation to “reduce productions and variables”. In the following sections, we will discuss Boolean theorem simplification in Section 3.2.1, Karnaugh Map simplification in Section 3.2.2, and Quine-McCluskey Method in Section 3.2.3.

3.2.1 Boolean Theorem Simplification

Obviously, Boolean theorem simplification is to simplify Boolean algebra expression by Boolean theorem. This simplification requires users understand the theorem very well and have great experience on it, and so we could maximize the theorem’s efficiency and effectiveness. In general, the simplification has some rules as below:

1. Term combination: Use $XY + X'Y = X$ to combine terms together. For example:

$$ABCD' + ABC'D' = ABD'$$

and

$$\begin{aligned} AB'C + ABC + A'BC &= AB'C + ABC + ABC + A'BC \\ &= AC + BC \end{aligned}$$

2. Term elimination: Use $X + XY = X$ to eliminate redundant terms or use Consensus theorem to eliminate consensus. For example:

$$AB' + AB'C = AB'$$

and

$$A'BC' + BCD + A'BD = A'BC' + BCD \quad A'BD \text{ is consensus}$$

3. Variable eliminate: Use $X + X'Y = X + Y$ to eliminate redundant variables. For example:

$$\begin{aligned}
 A'B + A'B'C'D' + ABCD' &= A'(B + B'C'D') + ABCD' \\
 &= A'(B + C'D') + ABCD' \\
 &= A'C'D' + ABCD' + A'B \\
 &= A'C'D' + B(A' + ACD') \\
 &= A'C'D' + A'B + BCD'
 \end{aligned}$$

4. Adding extra terms for combining or eliminating other terms: The ways to add terms includes: (1) add XX' ; (2) multiply $(X + X')$; (3) multiply YZ with $(XY + X'Z)$; (4) Add repeating terms, or (5) add XY to X . For example: if there is a 5-member committee going to vote an issue, in Boolean algebra expression, we could add many extra terms (repeating terms) to increase chances of combining terms.

$$\begin{aligned}
 R &= A'B'CDE + A'BC'DE + A'BCD'E + A'BCDE' + A'BCDE + AB'C'DE + \\
 &\quad AB'CDE' + AB'CDE + AB'CD'E + ABC'D'E + ABC'DE' + ABC'DE + \\
 &\quad ABCD'E' + ABCD'E + ABCDE' + ABCDE \\
 &= \boxed{A'B'CDE + AB'CDE + A'BCDE + ABCDE} + \\
 &\quad \boxed{ABC'DE' + ABCDE'} + \\
 &\quad \boxed{ABCDE' + ABCD'E'} + \\
 &\quad \boxed{A'BCDE + A'BCD'E + ABCD'E + ABCDE} + \\
 &\quad \boxed{ABCD'E + ABCDE + AB'CDE + AB'CD'E} + \\
 &\quad \boxed{ABC'DE + ABCDE + AB'C'DE + AB'CDE} + \\
 &\quad \boxed{ABCDE + ABC'DE + A'BCDE + A'BC'DE} + \\
 &\quad \boxed{ABCDE + ABCDE' + A'BCDE + A'BCDE'} + \\
 &\quad \boxed{AB'CDE + AB'CDE' + ABCDE + ABCDE'} + \\
 &\quad \boxed{ABC'D'E + ABC'DE + ABCD'E + ABCDE} \\
 &= CDE + ABDE' + ABCE' + BCE + ACE + ADE + BDE + BCD + ACD + \\
 &\quad ABE
 \end{aligned}$$

After some practices, we could come out some conclusions:

1. Boolean theorem simplification is not a systematic and easy simplification. It requires users understand the theorem and have great experience on it to maximize efficiency and effectiveness.
2. Boolean theorem simplification cannot clearly identify if a Boolean algebra expression has been simplified.
3. Boolean theorem simplification can only be properly used in Boolean algebra simplification with a few of variables.

3.2.2 Karnaugh Map of Simplification

Karnaugh Map is another 2-dimension Truth table. It can be used to express Boolean expression. It is also a great tool to simplify Boolean algebra. Karnaugh Map can be classified as dual-variable, 3-variable, 4-variable, 5-variable, and even 6-variable Karnaugh Maps based on the numbers of variables in the algebra expression. While variables more than six, it is too difficult to express as Karnaugh Map. The different kinds of Karnaugh Map simplification will be introduced in the following sections.

❖ Dual Variables

If there is a Boolean equation $Z = X' + XY$, we could express it by Karnaugh Map in Table 3.8a. X-axis represents variable X and y-axis represents variable Y.

Step 1: Draw a dual-variable Karnaugh Map and fill in input/output variables and variable's values shown as Table 3.8b. The vertical columns show the input value of Y and the horizontal rows show the input value of X.

Table 3.8a Application of Dual-variable Karnaugh Map

$$Z = X' + XY$$

Output		Input Y	
Z		0	1
Input	0	1	1
X	1	0	1

← The possible value
of Y

Table 3.8b Dual-variable Karnaugh Map- unfilled table

Ouput		InputY	
Z		0	1
Input	0		
X	1		

Step 2: Fill in output variable values in the fields where the output value is equal to 1 first and then in the fields where the output value is equal to 0. X' represents $X'Y'$ and $X'Y$. Then mark “1” in the fields where input X is “0” and input Y is “0” (“00” field) and where input X is “0” and input Y is “1” (“01” field). Beside the two situations described above, output Z is equal to “1” while input X is “1” and input Y is “1”. Mark “1” in the “11” field also. For the rest of unfilled fields (or empty entries), fill with “0” shown as in Table 3.8e.

Table 3.8c Application of Dual-variable Karnaugh Map, $Z = X' + XY$

Output		Input Y	
Z		0	1
Input	0	1	1
X	1	0	1

Step 3: Find out the ways to combine most of terms and eliminate most of variables. For example: in Table 3.8d, xy and $x'y$ can combine together to be Y (the field bordered with double lines). In Table 3.8e, $x'y'$ and $x'y$ can combine together to be X' (the field bordered with double lines). To combine terms together, we come out two rules: 1.) From left to right or from up to down, select two, four, or eight of “1” which are next to each other combine together to be a production; 2.) Choose the biggest combination that can combine most of terms together.

Table 3.8d Application of Dual-variable Karnaugh Map, $Z = Y + \boxed{?}$

Output Z		Input Y	
		0	1
Input X	0	1	1
	1		1

Table 3.8e Application of dual variable Karnaugh Map, $Z = X' + \boxed{?}$

Output Z		Input Y	
		0	1
Input X	0	1	1
	1	0	1

Step 4: Sum up (OR) each independent terms and combinational term. Therefore, we have $Z = X' + Y$ as below:

$$Z = X' + XY \rightarrow Z = X' + Y$$

2 terms 3 literals 2 terms 2 literals



❖ 3 Variables

If there is a Boolean equation $Z = AB + A'BC' + BC$, we could express it by Karnaugh Map in Table 3.9a.

Table 3.9a Application of 3-variable Karnaugh Map, $z = ab + a'bc' + bc$

Output Z		Input AB			
		00	01	11	10
Input	0		1	1	
C	1		1	1	

←The possible
combinational values of
AB

Step 1: Draw a 3-variable Karnaugh Map and fill in input/output variables and variable's values shown as Table 3.9b. The vertical columns show the input values of AB and the horizontal rows show the input values of C.

Table 3.9b 3-variable Karnaugh Map- unfilled table

Output Z		Input AB			
		00	01	11	10
Input	0				
C	1				

Step 2: Fill in output variable values in the fields where the output value is 1 first and then in the fields where the output value is 0. AB represents ABC and ABC' . Then mark "1" in the "111" and "110" fields. Beside the above two fields, output Z will be equal to "1" by $A'BC'$, and "010" field will be marked with "1" also. Similarly, BC represents ABC and $A'BC$. Mark "1" in the "111" and "011" fields. For the rest of unfilled fields, mark with "0" (or leave empty boxes) shown as in Table 3.9e.

Table 3.9c Application of 3-variable Karnaugh Map, $Z = AB + A'BC' + BC$

Output Z		Input AB			
		00	01	11	10
Input	0		1	1	
C	1		1	1	

Step 3: Find out the ways of combining most of terms to eliminate most of variables. For example: in Table 3.9d, ABC , ABC' , $A'BC$, and $A'BC'$ can be combined together to be B (the field bordered with double lines). In order to combine terms together, we come out two rules: 1.) Select two, four, or eight of "1" next to each other from left to right or from up to down and combine them together to be a production; 2.) Choose the biggest combination that can combine most of terms together.

Table 3.9d Application of 3-variable Karnaugh Map, $Z = AB + A'BC' + BC$

Output Z		Input AB			
		00	01	11	10
Input	0		1	1	
C	1		1	1	

Step 4 : Sum up (OR) each independent terms and combinational terms. Therefore, we could simplify $Z = B$ as below:

$$\begin{array}{ccc}
 Z = AB + A'BC' + BC & \rightarrow & Z = B \\
 3 \text{ terms } 7 \text{ literals} & & 1 \text{ terms } 1 \text{ literal}
 \end{array}$$

❖ 4 Variables

If there is a Boolean algebra expression $Z = A'B' + B'C' + CD + A'D'$, we could express it by Karnaugh Map in Table 3.10a.

Table 3.10a Application of 4-variable Karnaugh Map,

$$z = A'B' + B'C' + CD + A'D'$$

Output		Input AB			
Z		00	01	11	10
Input CD	00	1	1		1
	01	1			1
	11	1	1	1	1
	10	1	1		

←the possible value
of AB

Step 1: Draw a 4-variable Karnaugh Map and fill in input/output variables and variable's values shown in Table 3.10b. The vertical columns show the input values of AB and the horizontal rows show the input values of CD.

Table 3.10b 4-variable Karnaugh Map- unfilled table

Output		Input AB			
Z		00	01	11	10
Input CD	00				
	01				
	11				
	10				

Step 2: Fill in output variable values in the fields where the output value is 1 first and then in the fields where the output value is 0. $A'B'$ represents

$A'B'C'D'$, $A'B'C'D$, $A'B'CD'$ and $A'B'CD$. Then mark “1” in the “0000”, “0001”, “0010” and “0011” fields shown in Table 3.10c. Similarly, BC represents $A'B'C'D'$, $A'B'C'D$, $AB'C'D'$ and $AB'C'D$. Mark “1” in the “0000”, “0001”, “1000” and “1001” fields shown in Table 3.10d. CD represents $A'B'CD$, $A'BCD$, $AB'CD$ and $ABCD$. Mark “1” in the “0011”, “0111”, “1011” and “1111” fields shown in Table 3.10e. $A'D'$ represents $A'B'C'D'$, $A'B'CD'$, $A'BC'D'$ and $A'BCD'$. Mark “1” in the “0000”, “0010”, “0100” and “0110” fields shown in Table 3.10f. For the rest of unfilled fields, fill with “0” (or leave empty boxes).

Table 3.10c Application of 4-variable Karnaugh Map,

$$Z = \underline{A'B'} + B'C' + CD + A'D'$$

Output		Input AB			
Z		00	01	11	10
Input CD	00	1			
	01	1			
	11	1			
	10	1			

Table 3.10d Application of 4-variable Karnaugh Map,

$$Z = A'B' + \underline{B'C'} + CD + A'D'$$

Output		Input AB			
Z		00	01	11	10
Input CD	00	1			1
	01	1			1
	11	1			
	10	1			

Table 3.10e Application of 4-variable Karnaugh Map,

$$Z = A'B' + B'C' + \underline{CD} + A'D'$$

Output		Input AB			
		00	01	11	10
Input CD	00	1			1
	01	1			1
	11	1	1	1	1
	10	1			

Table 3.10f Application of 4-variable Karnaugh Map,

$$Z = A'B' + B'C' + CD + \underline{A'D'}$$

Output Z		Input AB			
		00	01	11	10
Input CD	00	1	1		1
	01	1			1
	11	1	1	1	1
	10	1	1		

Step 3: Find out the ways of combining most of terms to eliminate most of variables. For example: in Table 3.10g, $A'B'C'D'$, $A'B'CD'$, $A'BC'D'$ and $A'BCD'$ can be combined together to be $A'D'$ (the field bordered with double lines). Like Table 3.10g, Table 3.10h indicates that $A'B'CD$, $A'BCD$, $AB'CD$ and $ABCD$ can be combined together to be CD (the field bordered with double lines). In Table 3.10i, $A'B'C'D'$, $A'BC'D$, $AB'C'D'$ and $AB'C'D$ can be combined together to be $B'C'$ (the field bordered with double lines). In order to combine terms together, we come out two rules: 1.) Select two, four, or eight of “1” next to each other from

left to right or from up to down and combine them together to be a production; 2.) Choose the biggest combination that can combine most of terms together.

Table 3.10g Application of 4-variable Karnaugh Map,

$$Z = \boxed{?} + A'D'$$

Output Z		Input AB			
		00	01	11	10
Input CD	00	1	1		1
	01	1			1
	11	1	1	1	1
	10	1	1		

Table 3.10h Application of 4-variable Karnaugh Map, $Z = \boxed{?} + cd + A'D'$

Output Z		Input AB			
		00	01	11	10
Input CD	00	1	1		1
	01	1			1
	11	1	1	1	1
	10	1	1		

Table 3.10i Application of 4-variable Karnaugh Map, $Z = b'c' + cd + A'D'$

Output Z		Input AB			
		00	01	11	10
Input CD	00	1	1		1
	01	1			1
	11	1	1	1	1
	10	1	1		

Step 4: Sum up (OR) each independent terms and combinational terms. Therefore, we could simplify $Z = B'C' + CD + A'D'$ $Z = B'C' + CD + A'D'$ as below:

$$Z = A'B' + B'C' + CD + A'D' \rightarrow Z = B'C' + CD + A'D'$$

4 terms 8 literals 3 terms 6 literals

❖ 5-variable Karnaugh Map

If there is a Boolean algebra expression for over 5 committee members voting an issue:

$$R = A'B'CDE + A'BC'DE + A'BCD'E + A'BCDE' + A'BCDE + AB'C'DE + AB'CDE' + AB'CDE + AB'CD'E + ABC'D'E + ABC'DE' + ABC'DE + ABCD'E' + ABCD'E + ABCDE' + ABCDE$$

We could express it by Karnaugh Map as Table 3.11a.

Table 3.11a Karnaugh Map of Boolean algebra expression for over 5 committee members voting an issue

R			Input CD			
			00	01	11	10
Output AB	00	E = 0				
		E = 1			1	
	01	E = 0			1	
		E = 1		1	1	1
	11	E = 0		1	1	1
		E = 1	1	1	1	1



	10	E = 0			1	
		E = 1	0	1	1	1

Step 1: Draw a 5-variable Karnaugh Map and fill in input/output variables and variable's values shown in Table 3.11b. The vertical columns show the input values of CD and the horizontal rows show the input values of AB and E.

Table 3.11b Karnaugh Map of Boolean algebra expression for over 5 committee members voting an issue- unfilled table

R			Input CD			
			00	01	11	10
Output AB	00	E=0				
		E=1				
	01	E=0				
		E=1				
	11	E=0				
		E=1				
	10	E=0				
		E=1				

Step 2: Fill in output variable values in the fields where the output value is 1 first and then in the fields where the output value is 0. Because this is a vote passed by over half of 5 committee members, it only needs to fill in “1” at the fields shown in Table 3.11a. For example: ABCD'E' has three “1” (A, B, and C).



Step 3: Find out the ways of combining most of terms to eliminate most of variables such as Table 3.11c~3.11L.

Table 3.11c Karnaugh Map of Boolean algebra expression for over 5 committee members voting an issue, $R = \boxed{?} + \underline{ABE}$

R			Input CD			
			00	01	11	10
Input AB	00	E=0				
		E=1				
	01	E=0				
		E=1				
	11	E=0				
		E=1	1	1	1	1
	10	E=0				
		E=1				

Table 3.10d Karnaugh Map of Boolean algebra expression for over 5 committee members voting an issue, $R = \boxed{?} + \underline{ACD} + \underline{ABE}$

R			Input CD			
			00	01	11	10
Input AB	00	E=0				
		E=1				
	01	E=0				
		E=1				
	11	E=0			1	
		E=1			1	
	10	E=0			1	
		E=1				



		E=1			1	
--	--	-----	--	--	---	--

Table 3.11e Karnaugh Map of Boolean algebra expression for over 5 committee members voting an issue, $R = \boxed{?} + \underline{BCD} + ACD + ABE$

R			Input CD			
			00	01	11	10
Input AB	00	E=0				
		E=1				
	01	E=0			1	
		E=1			1	
	11	E=0			1	
		E=1			1	
	10	E=0				
		E=1				

Table 3.11f Karnaugh Map of Boolean algebra expression for over 5 committee members voting an issue, $R = \boxed{?} + \underline{BDE} + BCD + ACD + ABE$

R			Input CD			
			00	01	11	10
Input AB	00	E=0				
		E=1				
	01	E=0				
		E=1		1	1	
	11	E=0				
		E=1		1	1	
	10	E=0				
		E=1				

Table 3.11g Karnaugh Map of Boolean algebra expression for over 5 committee members voting an issue, $R + \boxed{?} + \underline{ADE} + BDE + BCD + ACD + ABE$

R			Input CD			
			00	01	11	10
Input AB	00	E = 0				
		E = 1				
	01	E = 0				
		E = 1				
	11	E = 0				
		E = 1		1	1	
	10	E = 0				
		E = 1		1	1	

Table 3.11h Karnaugh Map of Boolean algebra expression for over 5 committee members voting an issue, $R = \boxed{?} + \underline{BCE} + ADE + BDE + BCD + ACD + ABE$

R			Input CD			
			00	01	11	10
Input AB	00	E = 0				
		E = 1				
	01	E = 0				
		E = 1			1	1
	11	E = 0				
		E = 1			1	1
	10	E = 0				
		E = 1				

Table 3.11i Karnaugh Map of Boolean algebra expression for over 5 committee



members voting an issue, $R = \boxed{?} + \underline{ACE} + BCE + ADE + BDE + BCD + ACD + ABE$

R			Input CD			
			00	01	11	10
Input AB	00	E = 0				
		E = 1				
	01	E = 0				
		E = 1				
	11	E = 0				
		E = 1			1	1
	10	E = 0				
		E = 1			1	1

Table3.11j Karnaugh Map of Boolean algebra expression for over 5 committee members voting an issue, $R = \boxed{?} + \underline{ABDE'} + ACE + BCE + ADE + BDE + BCD + ACD + ABE$

R			Input CD			
			00	01	11	10
Input AB	00	E = 0				
		E = 1				
	01	E = 0				
		E = 1				
	11	E = 0		1	1	
		E = 1				
	10	E = 0				
		E = 1				



Table 3.11k Karnaugh Map of Boolean algebra expression for over 5 committee members voting an issue, $R = \boxed{?} + \underline{ABCE}' + ABDE' + ACE + BCE + ADE + BDE + BCD + ACD + ABE$

R			Input CD			
			00	01	11	10
Input AB	00	E = 0				
		E = 1				
	01	E = 0				
		E = 1				
	11	E = 0			1	1
		E = 1				
	10	E = 0				
		E = 1				

Table 3.11l Karnaugh Map of Boolean algebra expression for over 5 committee members voting an issue, $R = \underline{CDE} + ABCE' + ABDE' + ACE + BCE + ADE + BDE + BCD + ACD + ABE$

R			Input CD			
			00	01	11	10
Input AB	00	E = 0				
		E = 1			1	
	01	E = 0				
		E = 1			1	
	11	E = 0				
		E = 1			1	
	10	E = 0				
		E = 1			1	



Step 4: Sum up (OR) each independent terms and combinational terms. Therefore,

$$R = CDE + ABDE' + ABCE' + BCE + ACE + ADE + BDE + BCD + ACD + ABE$$

The result from simplification:

$$\begin{aligned} R = & A'B'CDE + A'BC'DE + A'BCD'E + A'BCDE' + A'BCDE + AB'C'DE + \\ & AB'CDE' + AB'CDE + AB'CD'E + ABC'D'E + ABC'DE' + ABC'DE + \\ & ABCD'E' + ABCD'E + ABCDE' + ABCDE \end{aligned}$$

16 terms	80 literals
----------	-------------



$$R = CDE + ABDE' + ABCE' + BCE + ACE + ADE + BDE + BCD + ACD + ABE$$

10 terms	32 literals
----------	-------------

❖ 6-variable Karnaugh Map

For 6-variable Karnaugh Map, we only introduce the ways to draw its unfilled table shown as in Table 3.12a. Because there are too many variables, it is hard to find out any variables that really express next to each other. (In reality, it has already happened to 5-variable Karnaugh Map.) Therefore, except the real next-to-next relationship, we try to identify any possible areas that seemly have next-to-next relationship as in Table 3.12b and Table 3.12c. As long as the areas have the same numbers, they then have next-o-next relationship, and just try to express their relative locations. Therefore, it is very important that readers have to understand what is the real meaning of next-to-next relationship. Don't misunderstand it!

Table 3.12a Unfilled table and next-to-next area identification of 6-variable Karnaugh Map

R			Input AB							
			00		01		11		10	
			C = 0	C = 1	C = 0	C = 1	C = 0	C = 1	C = 0	C = 1
Input DE	00	F = 0								
		F = 1								
	01	F = 0								
		F = 1								
	11	F = 0								
		F = 1								
	10	F = 0								
		F = 1								

Table 3.12b Next-to-next area identification of 6-variable Karnaugh Map

R			Input AB							
			00		01		11		10	
			C = 0	C = 1	C = 0	C = 1	C = 0	C = 1	C = 0	C = 1
Input DE	00	F = 0	2	8		5	6	6		8
		F = 1		8	4		6	6		8
	01	F = 0	2			5				
		F = 1			4				7	7
	11	F = 0	3	3		5				
		F = 1			4				7	7
	10	F = 0	3	3		5	6	6		
		F = 1			4		6	6		

Table 3.12c Next-to-next area identification of 6-variable Karnaugh Map
(Continuous)

R			Input AB							
			00		01		11		10	
			C = 0	C = 1	C = 0	C = 1	C = 0	C = 1	C = 0	C = 1
Input DE	00	F = 0	14	14	12				14	14
		F = 1	14	14	11	11			14	14
	01	F = 0	9						9	
		F = 1								
	11	F = 0	9		13		13		9	
		F = 1		10						10
	10	F = 0			12					
		F = 1		10	11	11				10

To here, we could understand:

1. Karnaugh Map can systematically and easily simplify Boolean equation expression.
2. Karnaugh Map can clearly identify Boolean equation expression has been minimization.
3. Karnaugh Map is better used in Boolean minimization with less than 6 variables. If algebra expressions have more than 6 variables, Quine-McCluskey Method is then needed, and in the next section, we will discuss the new minimization, “Quine-McCluskey Method”.

3.2.3 Quine-McCluskey Method

Quine-McCluskey Method can simplify minterm algebra expression to the simplest SOP. (Note: it must be a minterm algebra expression.) The simplification can systematically minimize a multi-variable algebra expression, great for paper work and easy to write down calculator programs. This simplification can cover the Karnaugh Map's drawbacks. Before introducing "Quine-McCluskey Method", there are three terms need to define first: implicant, prime implicant, and essential prime implicant.

1. Implicant: In Boolean expression, the production term that can be combined with other production term(s) are called as "Implicant of Boolean Expression".
2. Prime implicant: In Boolean expression, the production term that cannot be combined with other production term(s) to eliminate variables are called as "Prime Implicant of Boolean Expression".
3. Essential prime implicant: If an algebra minterm only belongs to a prime implicant, the implicant is then called as "Essential Prime Implicant".

The main ideas of Quine-McCluskey Method is:

1. Express Boolean algebra by minterm tables.
2. Systematically apply the theorem $xy + xy' = x$ to eliminate variables and derive all prime implicants.
3. Use prim implicant table, choose one group, which has the simplest prime implicants, and then sum up (OR operation) all those terms to represent original Boolean expression and have the smallest numbers of variables.



For example, use the Boolean expression $f(a, b, c, d) = \sum m(0, 1, 2, 5, 6, 7, 8, 9, 10, 14)$ to further explaining Quine-McCluskey Method.

Step 1: Group Boolean algebra minterms as Table 3.13 minterm table. There is no “1” in the group 0; one “1” in the group 1; two “1” in the group 2, and so on and so forth. Please notify that Quine-McCluskey Method must be started from minterm.

Table 3.13a Minterm table of $f(a,b,c,d) = \sum m(0, 1, 2, 5, 6, 7, 8, 9, 10, 14)$

Group	m	Implicant	Non-prime Implicant
Group 0	0	0000	✓
Group 1	1	0001	✓
	2	0010	✓
	8	1000	✓
Group 2	5	0101	✓
	6	0110	✓
	9	1001	✓
	10	1010	✓
Group 3	7	0111	✓
	14	1110	✓

Step 2: Find out prime implicant by theorem $xy + xy' = x$:

1. Try to simplify all implicants in the groups 0 and 1 by the theorem $xy + xy' = x$. If the implicants can be simplified, new implicants (or prime implicants) will be derived and then filled in the group 0 in new table (shown as Table 3.13). Symbol “—” means the variables have been simplified. Meanwhile, in

Table 3.13a, the symbol “ \checkmark ” in the column of non-prime implicant indicates that the implicant is a non-prime implicant. Then, try again to simplify all the implicants in the groups 1 and 2, and fill in the group 1 in the new Table 3.13b. For the groups 2 and 3, perform the same simplification process again and fill in the group 2 in the new Table 3.13b, and so on and so forth.

2. In the same group of the new table, erase repeating combinational terms and mark with double cross-out lines. (Note: there is no repeating terms in Table 3.13b)

Table 3.13b First simplified table

Group	Simplifying Resources	New Implicants	Non-prime Implicant
Group 0	0, 1	000–	\checkmark
	0, 2	00–0	\checkmark
	0, 8	–000	\checkmark
Group 1	1, 5	0–01	
	1, 9	–001	\checkmark
	2, 6	0–10	\checkmark
	2, 10	–010	\checkmark
	8, 9	100–	\checkmark
	8, 10	10–0	\checkmark
Group 2	5, 7	01–1	
	6, 7	011–	
	6, 14	–110	\checkmark
	10, 14	1–10	\checkmark

3. If new tables such as Table 3.13b still could be found, repeat a and b processes again. Otherwise, it means the process of finding prime implicants has been completed.

Table 3.13c Second simplified table

Group	Simplifying Resources	New Implicants	Non-prime Implicant
Group 0	0, 1, 8, 9	–00–	
	0, 2, 8, 10	–0–0	
	0, 8, 1, 9	–00–	
	0, 8, 2, 10	–0–0	
Group 1	2, 6, 10, 14	—10	
	2, 10, 6, 14	—10	

Step 3: Create a prime implicant table, and identify essential prime implicants. Use the prime implicants from step 2, create a prime implicant table such as Table 3.13d shown as below. Minterms are shown on the top of the vertical part of the table, prime implicants are on the left side of the horizontal part of the table, and the minterms covered by each prime implicant are in the mid part of the table. According to essential prime implicant definition, both (0, 1, 8, 9) and (0, 2, 8, 10) are essential prime implicants since m_9 and m_{14} only belongs to (0, 1, 8, 9) and (0, 2, 8, 10), respectively.

Table 3.13d Prime implicant table

Simplifying Resources	Prime Implicant	Minterm of Boolean Expression										Type of Prime Implicant
		0	1	2	5	6	7	8	9	10	14	
(0, 1, 8, 9)	–00–	x	x					x	x			Essential Prime Implicant
(0, 2, 8, 10)	–0–0	x		x				x		x		
(2, 6, 10, 14)	—10			x		x				x	x	Essential Prime Implicant
(1, 5)	0–01		x		x							
(5, 7)	01–1				x		x					
(6, 7)	011–					x	x					

Step 4: Select the solution having the fewest prime implicants and Boolean variables to completely cover all minterms to represent the Boolean expression. The details of the selecting process is:

1. Choose essential prime implicants first, and eliminate the covered minterms. For example, in Table 3.13e, select the two essential prime implicants, (0, 1, 8, 9) and (2, 6, 10, 14).
2. Choose the prime implicants covering most of the minterms. For example, in Table 3.13f, m_5 and m_7 are the only two uncovered minterms, and just can be covered by (5, 7).

Table 3.13e Prime implicant table

Simplifying Resources	Prime Implicant	Minterm of Boolean Expression										Type of Prime Implicant
		0	1	2	5	6	7	8	9	10	14	
(0, 1, 8, 9)	–00–	x	x					x	x			Essential Prime Implicant
(0, 2, 8, 10)	–0–0	x		x				x		x		
(2, 6, 10, 14)	—10			x		x				x	x	Essential Prime Implicant
(1, 5)	0–01		x		x							
(5, 7)	01–1				x		x					
(6, 7)	011–					x	x					

Step 5: Sum up (OR operation) all the selected prime implicants to find out the simplest Boolean expression.

$$\begin{aligned}
 f(a, b, c, d) &= \sum m(0, 1, 2, 5, 6, 7, 8, 9, 10, 14) \\
 &= B'C' + CD' + A'BD
 \end{aligned}$$

Note: The simplest expression might not be the only one; that is, there might be another simplest expressions.

Table 3.13f Prime implicant table

Simplifying Resources	Prime Implicant	Minterm of Boolean Expression										Type of Prime Implicant
		0	1	2	5	6	7	8	9	10	14	
(0, 1, 8, 9)	–00–	x	x					x	x			Essential Prime Implicant
(0, 2, 8, 10)	–0–0	x		x				x		x		
(2, 6, 10, 14)	—10			x		X				x	x	Essential Prime Implicant
(1, 5)	0–01		x		x							
(5, 7)	01–1				x		x					
(6, 7)	011–					X	x					

For “don’t care” terms of Boolean expression, how could we use Quine-McCluskey Method to simplify the expression?

The following equation is the example to help us understand how Quine-McCluskey Method can do with “don’t care” terms of Boolean expression.

$$f(a, b, c, d) = \sum m(2, 3, 7, 9, 11, 13) + \sum d(1, 10, 15)$$

Step 1: Group minterms and “don’t care” terms of Boolean equation are shown in Table 3.14a. We could not find “1” in the group 0, but a “1” in the group 1 and two “1” in the group 2, and so on and so forth.

Table 3.14a $f(a, b, c, d) = \sum m(2, 3, 7, 9, 11, 13) + \sum d(1, 10, 15)$

Minterm and “Don’t Care” term table

Group	m	d	Implicant	Non-prime Implicant
Group 1		1	0001	✓
	2		0010	✓
Group 2	3		0011	✓
	9		1001	✓
		10	1010	✓
Group 3	7		0111	✓
	11		1011	✓
	13		1101	✓
Group 4		15	1111	✓

Step 2: Find prime implicants by the theorem, $xy + xy' = x$.

1. Try to use the theorem $xy + xy' = x$ to simplify all implicants in the groups 1 and 2. If new implicants (or prime implicants) are found after simplification, fill them in the group 1 of new table (Table 3.14b), and symbolize “—” to indicate the variables are simplified already. Meanwhile, in Table 3.14a, mark “✓” in the column of non-prime implicant to indicate the implicant is a non-prime implicant. Then, try to simplify all the implicants in the groups 2 and 3, and fill them in the group 2 in the new table (Table 3.14b). Repeat the same process for the groups 3 and 4 shown in Table 3.14b, and so on and so forth.
2. In the same group of the new table, erase the repeating combinational terms and mark with double cross-out lines. (Note: there are no repeating terms in Table 3.14b.)

3. If there is still a new table, repeat the processes 1 and 2 for the new table (Table 3.14b). Otherwise, the process of finding prime implicants is completed. Table 3.14c is the result from repeating the processes 1 and 2.

Table 3.14b First simplified table

Group	Simplifying Resources	New Implicant	Non-prime Implicant
Group 1	(1, 3)	00–1	✓
	(1, 9)	–001	✓
	(2, 3)	001–	✓
	(2, 10)	–010	✓
Group 2	(3, 7)	0–11	✓
	(3, 11)	–011	✓
	(9, 11)	10–1	✓
	(9, 13)	1–01	✓
	(10, 11)	101–	✓
Group 3	(7, 15)	–111	✓
	(11, 15)	1–11	✓
	(13, 15)	11–1	✓

Step 3: Create a prime implicant table and indicate the essential prime implicants. Use the prime implicants resulted from step 2 to establish a prime implicant table as Table 3.14d. Please be aware of that only minterm can be listed in the table, not “don’t care” terms. According to the prime implicant definition, (2, 3, 10, 11), (3, 7, 11, 15) and (9, 11, 13, 15) are essential prime implicants since m_2 , m_7 , and m_{13} only belong to (2, 3, 10, 11), (3, 7, 11, 15) and (9, 11, 13, 15), separately.

Table 3.14c Second simplified table

Group	Simplifying Resources		Non-prime Implicant
Group 1	(1, 3, 9, 11)	-0-1	
	(1, 9, 3, 11)	-0-1	
	(2, 3, 10, 11)	-01-	
	(2, 10, 3, 11)	-01-	
Group 2	(3, 7, 11, 15)	--11	
	(3, 7, 11, 15)	--11	
	(9, 11, 13, 15)	1--1	
	(9, 11, 13, 15)	1--1	

Table 3.14d Prime implicant table

Simplifying Resources	Prime Implicant	Minterm of Boolean Expression						Type of Prime Implicant
		2	3	7	9	11	13	
(1, 3, 9, 11)	-0-1		x		x	x		
(2, 3, 10, 11)	-01-	x	x			x		Essential Prime Implicant
(3, 7, 11, 15)	--11		x	x		x		Essential Prime Implicant
(9, 11, 13, 15)	1--1				x	x	x	Essential Prime Implicant

Step 4: Select the solution, which comes out the fewest prime implicants and Boolean variables to completely cover all the minterms to represent the

Boolean expression. To start the selection, first choose essential prime implicants and delete covered minterms. For example, in Table 3.14e, select the three essential prime implicants: (2, 3, 10, 11), (3, 7, 11, 15), and (9, 11, 13, 15), and luckily they already cover all the minterms.

Step 5: Sum up (OR operation) all the selected prime implicants and the summation is the simplest Boolean expression.

$$f(a, b, c, d) = \sum m(2, 3, 7, 9, 11, 13) + \sum d(1, 10, 15) \\ = B'C + CD + AD$$

Table 3.14e Prime implicant table

Simplifying Resources	Prime Implicant	Minterm of Boolean Expression						Type of Prime Implicant
		2	3	7	9	11	13	
(1,3,9,11)	-0-1		x		x	x		
(2,3,10,11)	-01-	X	x			x		Essential Prime Implicant
(3,7,11,15)	--11		x	x		x		Essential Prime Implicant
(9,11,13,15)	1—1				x	x	x	Essential Prime Implicant

3.3 Logic Gate

Logic gate is a digit circuit to make Boolean operation become real. Totally, there are three basic Boolean operations: AND, OR, and NOT operations, and therefore logic gates can be divided into AND gate, OR gate, and NOT gate. In another words, we could complete all digit logic circuits by the three basic logic gates. In the next sections, we will not only introduce basic logic gates but also other derivative logic gates such as NAND, NOR, and XOR gates.

3.3.1 AND Gate

AND gate is a digital logic gate performing Boolean AND operation (or also called as “production operation”). Compared to dual-variable production, three-variable production, and four-variable production, etc., AND gate also can be classified as dual-input AND gate, three-input AND gate, and four-input AND gate, etc. An AND gate output can be “1” only when all AND gate inputs are equal to “1”; otherwise, the output must be “0”. A and B switches in Figure 3.1, for example, must be “ON” at the same time to connect the circuits and lighten the bulbs. From previous chapter, we already know that Truth table is a great tool to describe the relationship between inputs and outputs of Boolean equation. Actually, it is also good to describe logic gates. The AND gate functions described by Truth table are shown as below:

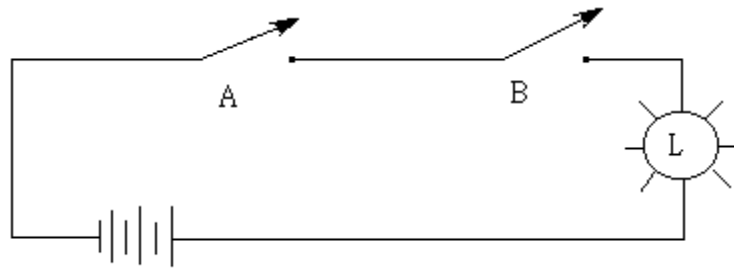


Figure 3.1 AND gate physical definition

❖ Dual-input AND Gate

For a dual-input AND gate, we could use Truth table (Table 3.15) to describe its functions. Obviously, output Y will be “1” only when A and B are equal to “1” at the same time; otherwise, output Y will be “0”. This kind of dual-input AND gate is usually represented by the circuit symbol in Figure 3.2.

Table 3.15 Truth table of dual-input AND gate

Input		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

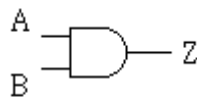


Figure 3.2 Circuit symbol of dual-input AND gate

❖ Three-input AND Gate

Similarly, for a three-input AND gate, we could also use Truth table to describe its functions as Table 3.16. Clearly, output Y will be equal to “1” only when A, B, and C are equal to “1”; otherwise, output Y must be equal to “0”. This kind of three-input AND gate is usually represented by the circuit symbol shown in Figure 3.3.

Table 3.16 Truth table of three-input AND gate

Input			Output
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

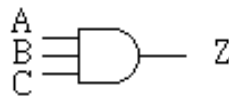


Figure 3.3 Circuit symbol of three-input AND gate

❖ Four-input AND Gate

For a four-input AND gate, we could use Truth table (Table 3.17) to describe the gate functions. Obviously, output Y can be equal to “1” only when A, B, C, and D are equal to “1”; otherwise, output Y must be equal to “0”. We usually use the

Circuit symbol shown in Figure 3.4 to symbolize this kind of four-input AND gate.

Table 3.17 Truth table of four-input AND gate

Input				Output
A	B	C	D	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

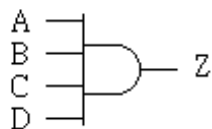


Figure 3.4 Circuit symbol of four-input AND gate

3.3.2 OR Gate

An OR gate is a digital logic gate performing Boolean OR operation (or called as “sum operation”). Like dual-variable sum, three-variable sum, and four-variable sum, etc., an OR gate also can be divided into dual-input, three-input, and four-input OR gates, etc. To have output equal to “1”, one of OR gate inputs has to be equal to “1”. Otherwise, output will be “0”. For example, either A or B switch in Figure 3.5 is “ON”. The circuits are still connected and the bulbs are still lightened. Like AND gate functions, all OR gate functions can be described by Truth tables, and we will discuss further in the following sections.

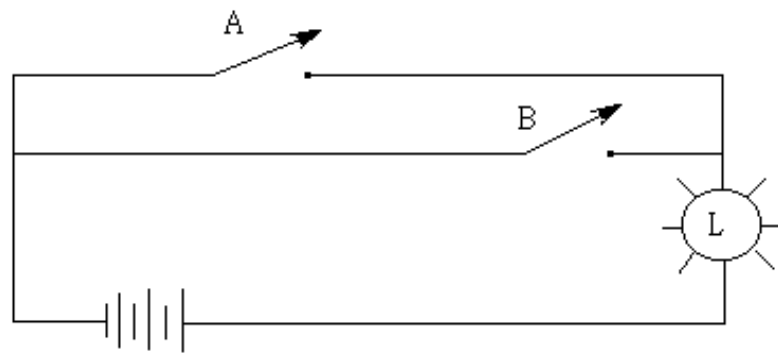


Figure 3.5 OR gate physical definition

❖ Dual-input OR Gate

For a dual-input OR gate, we could use Truth table (Table 3.18) to describe its functions. Obviously, output Y is equal to “1” when there is at least one of A and B is equal to “1”; otherwise, output Y is equal to “0”. This kind of dual-input OR gate is usually symbolized by Circuit symbol shown in Figure 3.6.

Table 3.18 Truth table of dual input OR gate

Input		Output
A	B	Y

0	0	0
0	1	1
1	0	1
1	1	1



Figure 3.6 Circuit symbol of dual-input OR gate

❖ Three-input OR Gate

Similarly, for a three-input OR gate, we could also use Truth table (Table 3.19) to describe the gate functions. Obviously, output Y is equal to “1” when there is at least one of A, B and C equal to “1”; otherwise, output Y must be equal to “0”. This three-input OR gate can be symbolized by the Circuit symbol in Figure 3.7.

Table 3.19 Truth table of three-input OR gate

Input			Output
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

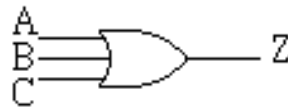


Figure 3.7 Circuit symbol of three-input OR gate

❖ Four-input OR Gate

The functions of a four-input OR gate can also be described by Truth table (Table 3.20). Like previous gates, the output Y of a four-input OR gate can be equal to “1” when there is at least one of A, B, C, and D equal to “1”; otherwise, output Y must be equal to “0”. The Circuit symbol in Figure 3.8 can be used to symbolize the four-input OR gate usually.

Table 3.20 Truth table of four-input OR gate

Input				Output
A	B	C	D	Y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1

1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

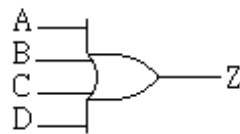


Figure 3.8 Circuit symbol of four-input OR gate

3.3.3 NOT Gate

A NOT gate is a mono-input logic gate. It is also a digital logic gate performing Boolean NOT operation (or called as “complementary operation”). Like AND gate, we use Truth table as Table 3.21 to describe the NOT gate functions. Clearly, output Y is equal to “0” when A is equal to “1”; otherwise, output Y is equal to “1”. Usually, we use the circuit symbol in Figure 3.9 to symbolize NOT gate.

Table 3.21 Truth table of NOT gate

Input	Output
A	Z
0	1
1	0

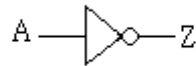


Figure 3.9 Circuit symbol of NOT gate

3.3.4 XOR Gate

A XOR (Exclusive OR) gate is a logic gate but not a basic logic gate. It is made with several basic logic gates, and performs Boolean XOR operation. Table 3.22 shows Truth table of XOR gate. For “Exclusive OR” gate with dual inputs, output Z is equal to “1” only when A and B are not simultaneously equal to the same value; otherwise, output Z is equal to “0”. Figure 3.10a is XOR circuit symbol. From Table 3.22, Figure 3.10b shows XOR equal-effect circuits made with basic logic gates.

Table 3.22 Truth table of XOR gate

Input		Output
A	B	Z
0	0	0
1	0	1
0	1	1
1	1	0



Figure 3.10a Circuit symbol of XOR gate

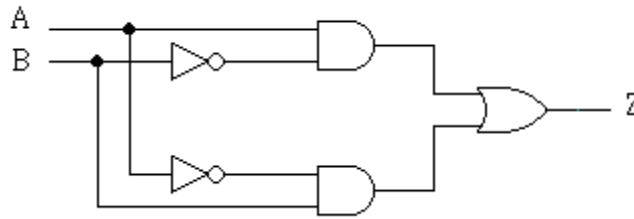


Figure 3.10b Equal-effect circuit of XOR gate

3.3.5 NAND Gate

A NAND gate is a logic gate but not a basic logic gate. It is made with AND and NOT basic logic gates, and performs Boolean AND and NOT operations, which are production operation and complementary operation. Compared to production of dual variables, three variables, and four variables... etc., a NAND gate can also be divided into dual input, three-input, and four-input NAND gates. Identically, we will use Truth table to describe NAND gate functions.

❖ Dual-input NAND Gate

For a dual-input NAND gate, we could use Truth table in Table 3.23 to describe NAND gate functions. Its output Y is equal to “0” only when both A and B are equal to “1”; otherwise, output Y must be equal to “1”. The dual-input NAND gate can be usually shown by the circuit symbol in Figure 3.11a. The small circle at the output pin means inverse directions or complements. According to the information in Table 3.23, Figure 3.11b identifies the equal-effect NAND circuits made with basic logic gates. From the equal-effect circuits, the small circle could be defined more clearly at the output pin.

Table 3.23 Truth table of dual input NAND gate

Input		Output
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

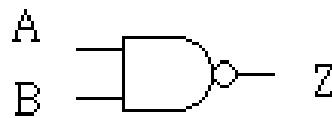


Figure 3.11a Circuit symbol of dual-input NAND gate

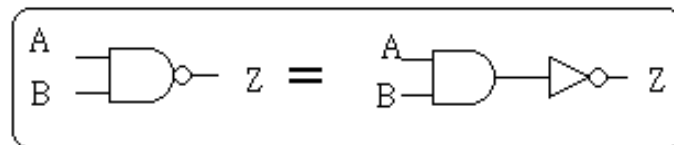


Figure 3.11b Equal-effect circuit of NAND gate

❖ Three-input NAND Gate

Likewise, for a three-input NAND gate, we could use Truth table (Table 3.24) to describe the gate functions. Output Y will be equal to “0” only when all of A, B, and C are equal to “1”; otherwise, output Y must be equal to “1”. Usually we use the Circuit symbol in Figure 3.12a to represent a three-input NAND gate. Figure 3.12b is based on Table 3.24 to show equal-effect NAND circuits made with basic logic gates.

Table 3.24 Truth table of three-input NAND gate

Input			Output
A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

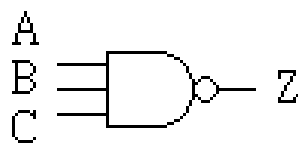


Figure 3.12a Circuit symbol of three-input NAND gate

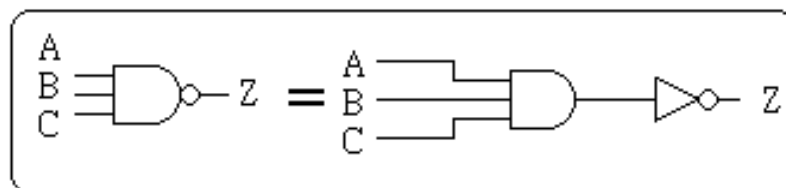


Figure 3.12b Equal-effect circuit of NAND gate

❖ Four-input NAND Gate

For a four-input NAND gate, we could use Truth table (Table 3.25) to describe the

gate functions. Output Y will be equal to “0” only when all of A, B, C, and D are equal to “1”; otherwise, output Y must be equal to “1”. Usually we use the Circuit symbol in Figure 3.13a to represent a four-input NAND gate. Figure 3.13b is based on Table 3.25 to show equal-effect NAND circuits made with basic logic gates.

Table 3.25 Truth table of four-input NAND gate

Input				Output
A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

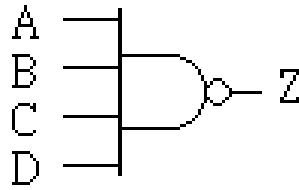


Figure 3.13a Circuit symbol of four-input NAND gate

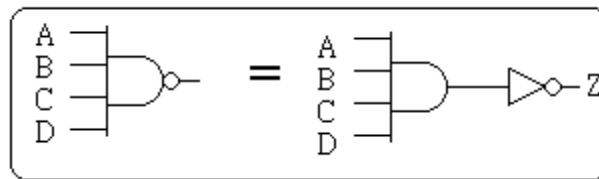


Figure 3.13b Equal-effect circuit of four-input NAND gate

3.3.6 NOR Gate

A NOR gate is another logic gate but not a basic logic gate. It is made with OR and NOT basic logic gates, and performs Boolean OR and NOT operations, which are OR operation and complementary operation. Similarly, a NAND gate has dual-input, three-input, and four-input NOR gates.

❖ Dual-input NOR Gate

For a dual-input NOR gate, we could use Truth table (Table 3.26) to describe the gate functions. Output Y will be equal to “0” when there is at least one of A and B equal to “1”; otherwise, output Y must be equal to “1”. Usually we use the Circuit symbol in Figure 3.14a to represent the NOR gate. Figure 3.14b is based on Table 3.26 to show equal-effect NOR circuits made with basic logic gates.

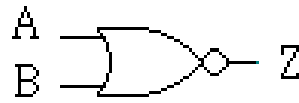


Figure 3.14a Circuit symbol of dual-input NOR gate

Table 3.26 Truth table of dual input NOR gate

Input		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

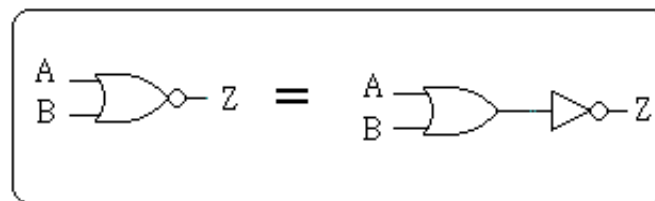


Figure 3.14b Equal-effect circuit of dual-input NOR gate

❖ Three-input NOR Gate

Likewise, we could use Table 3.27 for describing the function of three-input NOR gate functions. Output Y will be equal to “0” when there is at least one of A, B, and C equal to “1”; otherwise, output Y must be equal to “1”. Usually, the circuit symbol, in Figure 3.15a, be used to represent three-input NOR gates. Figure 3.15b, based on Table 3.27, shows that the equal-effect circuit of three-input NOR gates made with basic logic gates.

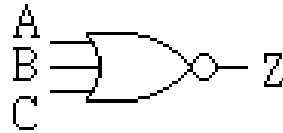


Figure 3.15a Circuit symbol of three-input NOR gate

Table 3.27 Truth table of three-input NOR gate

Input			Output
A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

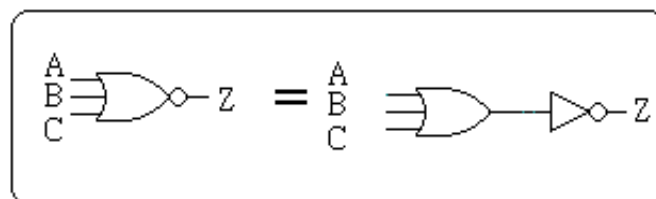


Figure 3.15b Equal-effect circuit of three-input NOR gate

❖ Four-input NOR Gate

The functions of a four-input NOR gate can also be described by Truth table (Table 3.28). Obviously, output Y will be equal to “1” when there is at least one of A, B, C, and D equal to “1”; otherwise, output Y must be equal to “0”. Usually the circuit symbol, in Figure 3.16a, be used to represent a three-input NOR gates. Figure 3.16b is based on Table 3.28 to show equal-effect circuits of four-input NOR gates made with basic logic gates.

Table 3.28 Truth table of four-input NOR gate

Input				Output
A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

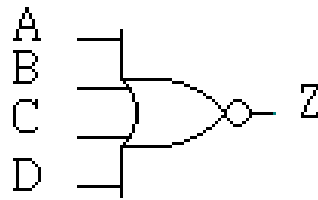


Figure 3.16a Circuit symbol of four-input NOR gate

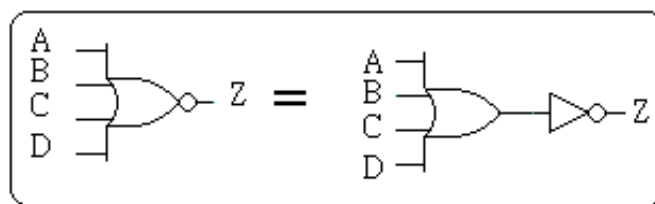


Figure 3.16b Equal-effect circuit of four-input NOR gate

3.3.7 NXOR Gate

A XNOR (Inclusive OR) gate is a digital logic gate but not a basic logic gate. It is combined with XOR and NOT basic logic gates, and performs Boolean XNOR operation. Its Truth table is shown as Table 3.29. For an Inclusive OR gate with dual inputs, its output Z will be equal to “1” only when A and B are simultaneously equal to the same value; otherwise, output Z will be equal to “0”. Figure 3.17a shows XNOR Circuit symbol. Figure 3.17b is based on Table 3.29 to show XNOR equal-effect circuits made with logic gates.

Table 3.29 Truth table of XNOR gate

Input		Output
A	B	Z
0	0	1

1	0	0
0	1	0
1	1	1



Figure 3.17a Circuit symbol of XNOR gate

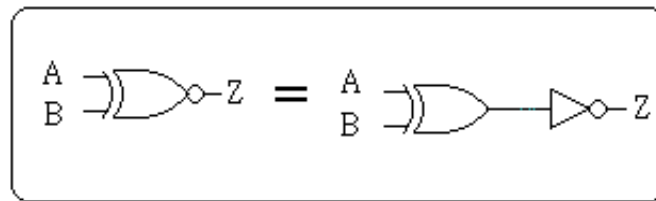


Figure 3.17b Equal-effect circuit of XNOR gate

3.3.8 Demorgan's Equal-effect Circuit

In section 3.12, we have discussed Demorgan's theorem. For illustration, we list the equations as below:

$$\text{OR operation: } (A + B)' = A' \cdot B' \quad \text{..... (3.18)}$$

$$\text{AND operation: } (A \cdot B)' = A' + B' \quad \text{..... (3.19)}$$

According to Demorgan's theorem, we could derive equal-effect circuits. The following is a dual-input example. For multi-inputs, its derivative process is the same as dual inputs.

❖ Equal-effect Circuit of Demorgan's NAND Gate (Figure 3.18)

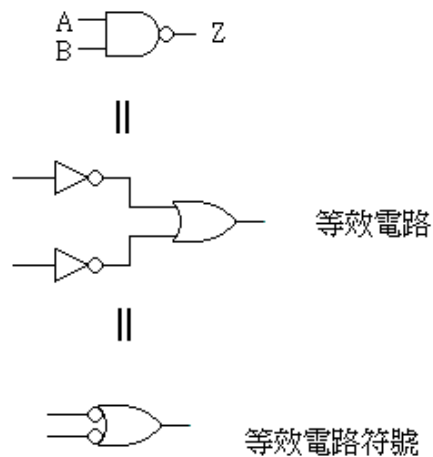


Figure 3.18 Circuit symbol and equal-effect circuit of Demorgan's NAND gate

❖ Equal-effect Circuit of Demorgan's NOR Gate (Figure 3.19)

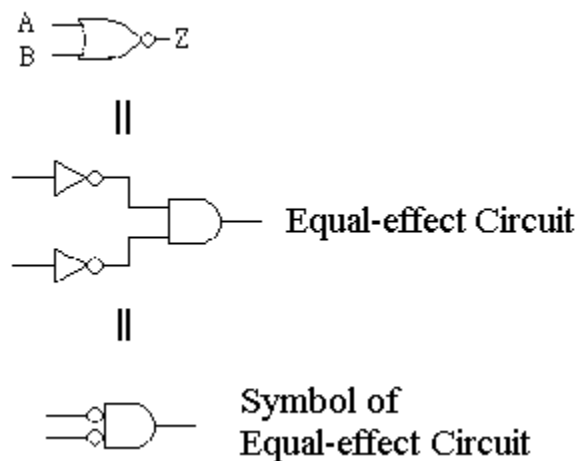


Figure 3.19 Equal-effect circuit and circuit symbol of Demorgan's NOR gate

3.4 Applications of Logic Gate

In Section 3.1, we have introduced Boolean algebra, including the basic concepts of Boolean algebra definition, theorems, Truth tables and expressions. In Section 3.2, we introduce minimization methodologies usually used in Boolean algebra, including Boolean minimization, Karnaugh Map minimization, and Quine-McCluskey method. Once familiar with those minimization methodologies, it is easy to simplify Boolean algebra to have the fewest productions and variables. In section 3.3, we discussed logic gates. We know that logic gates are the equal digital circuits performing Boolean operations. Any Boolean algebra expression can operate equally by logic gates. Therefore, the goal in Section 3.2 is “use the fewest logic gates and the fewest connections to perform Boolean algebra operations.” The followings are some examples of logic gate applications to complete Boolean algebra operations.

❖ Logic Gate Application 1

The following is the simplified Boolean algebra expression in the Section 3.2.3:

$$\begin{aligned}f(a, b, c, d) &= \sum m(2, 3, 7, 9, 11, 13) + \sum d(1, 10, 15) \\&= B'C + CD + AD\end{aligned}$$

Use logic gates to perform the simplified Boolean algebra expression.

Figure 3.20 is the logic circuit diagram performing $f(a, b, c, d) = B'C + CD + AD$

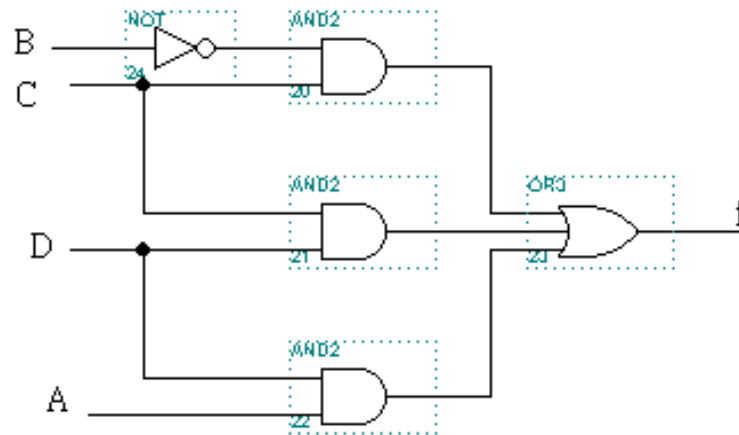


Figure 3.20 Logic circuit diagram of $f(a, b, c, d) = B'C + CD + AD$

❖ Logic Gate Application 2

The following is the simplified Boolean algebra expression in the section 3.2.3:

$$\begin{aligned} f(a, b, c, d) &= \sum m(0, 1, 2, 5, 6, 7, 8, 9, 10, 14) \\ &= B'C' + CD' + A'BD \end{aligned}$$

Use logic gates to perform the simplified Boolean algebra expression.

Figure 3.21 is the logic circuit diagram performing $f(a, b, c, d) = B'C' + CD' + A'BD$.

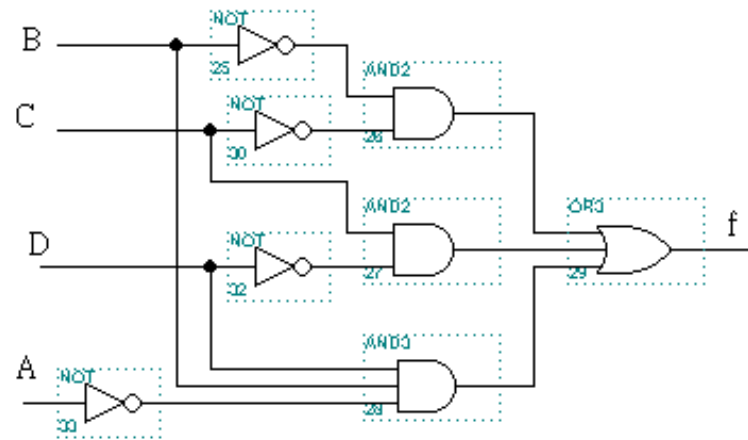


Figure 3.21 Logic gate circuit diagram of $f(a, b, c, d) = B'C' + CD' + A'BD$

❖ Logic Gate Application 3

Use logic gates to perform the simplified Boolean algebra expression, $R = CDE + ABDE' + ABCE' + BCE + ACE + ADE + BDE + BCD + ACD + ABE$, in the Section 3.2.1. Figure 3.22 is the logic circuit diagram for performing $R = CDE + ABDE' + ABCE' + BCE + ACE + ADE + BDE + BCD + ACD + ABE$.

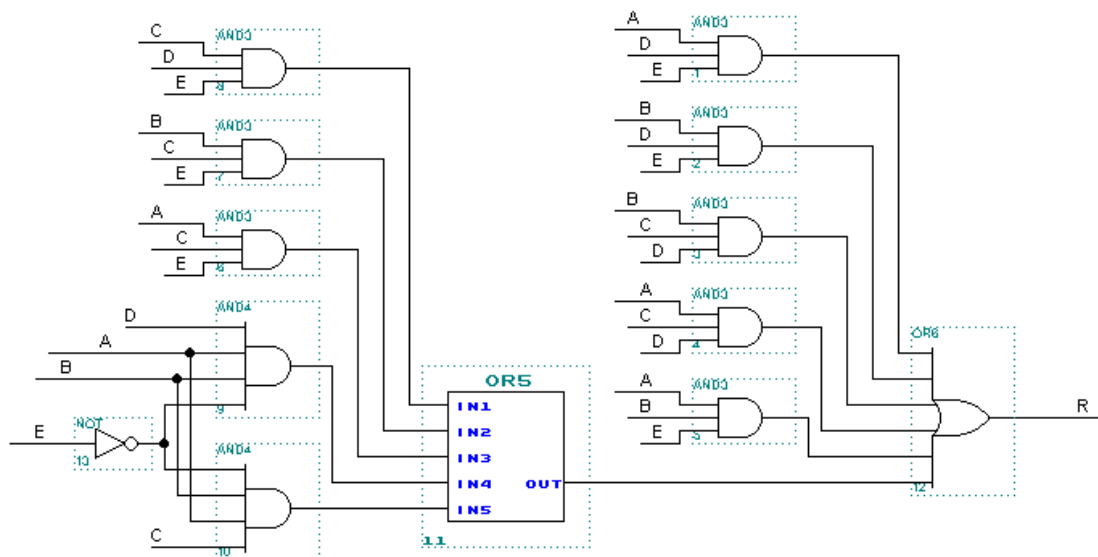


Figure 3.22 Logic Gate Circuit Diagram of $R = CDE + ABDE' + ABCE' + BCE + ACE + ADE + BDE + BCD + ACD + ABE$



3.5 Practices

1. Please find out and delete the consensus terms of the following expressions:

$$ABC'D + A'BE + BC'DE ;$$

$$(A' + B + C)(A + D)(B + C + D) ;$$

$$AB'C + A'BD + BCD' + A'BC ;$$

$$A'B'C + BC'D' + A'CD + AB'D' + BCD + AC'D' \circ$$

2. Please try to verify the Equation 3-29.

3. Please multiply the following expressions and express in POS:

$$(A + B)(A + C')(A + D)(BC'D + E) ;$$

$$(A + B' + C)(B' + C + D)(A + C) ;$$

$$(A' + BE')(BE' + C + D)(E + C') \circ$$

4. Express the next expressions in SOP:

$$AB'C + D ;$$

$$BC'D + A'BE + BEF ;$$

$$W + X'Y + VZ \circ$$

5. Use Truth tables to prove the next algebra expressions are true:

$$(A + CD)(A' + B) = A'CD + AB \circ$$

6. Please use Boolean theorems to simplify the following expressions:

$$XY + X'YZ' + YZ ;$$

$$(XY' + Z)(X + Y')Z ;$$

$$XY' + Z + (X' + Y)Z ;$$



$$A'D(B' + C) + A'D'(B + C') + (B' + C)(B + C') ;$$

$$W'X' + X'Y' + YZ + W'Z' \circ$$

7. Use Karnaugh Map to simplify $Z = Y' + X'Y'$.
8. Use Karnaugh Map to minimize $Z = (AB' + C)(A + B')C$.
9. Use Karnaugh Map to minimize $Z = AB' + C + (A' + B)C'$.
10. Use Karnaugh Map to minimize the following expressions:
 $XY + X'YZ' + YZ ;$
 $(XY' + Z)(X + Y')Z ;$
 $XY' + Z + (X' + Y)Z ;$
 $A'D(B' + C) + A'D'(B + C') + (B' + C)(B + C') ;$
 $W'X' + X'Y' + YZ + W'Z' \circ$
11. Please use Quine-McCluskey Method to minimize Boolean algebra expression as below:
 $f(a, b, c) = \sum m(0, 1, 2, 5, 6, 7) \circ$



3.6 Review

Please answer the following questions to review this chapter.

- ❑ Do you know what Boolean algebra is?
- ❑ Could you describe and verify 10 Boolean theorems?
- ❑ Could you describe all Boolean algebra expressions?
- ❑ Could you point out three Boolean algebra minimization methods introduced in this chapter?
- ❑ Do you know the key points to adopt the simplifications of Boolean theorems?
- ❑ Do you know which simplification is the most systematic and good for minimize multi-variables?
- ❑ Do you know the basic logic gates?
- ❑ Could you plot a Demorgan's equal-effect circuit of an NAND gate?

CHAPTER 4

A New Design
Methodology —
PC Aided Digital
Logic Design Using
MAX+PLUS II
Baseline (Version 9.23)





In this chapter, we will focus on MAX+PLUS II Baseline version 9.23. Graphic-edited circuit entry technology will be introduced in Section 4.3, circuit functional simulation in Section 4.4, floorplan and design compilation in Section 4.5, and device programming and circuit verification in Section 4.6. To further understand LP-2900 CPLD logic design experimental platform, we will give examples with graphic edit technologies in Section 4.7 and review the following topic:

1. Design (circuit) entry
2. Compilation and error location
3. Functional simulation
4. Floorplan
5. Design compilation, which is the generation and conversion of “Configuration Data”, and then
6. Download to FLEX10K experimental platform to complete circuit verification

4.1 MAX+PLUS II Baseline Setup and Start

Before we set up the program, please make sure your PC has at least 64 MB DRAM and 200 MB hard disk space after installing Windows 95/98 or Windows NT. For sure, the faster CPU speed the better. A MAX+PLUS II Baseline version 9.23 CD can download from <http://www.altera.com/>. The following is the short description of Baseline setup process (version 9.23):

1. First, please check your PC has at least 200 MB HD space and 64 MB DRAM after install Windows 95/98.

2. Create a sub-directory in C drive as C:\baseline.
3. Download “baseline923.exe” from <http://www.altera.com/> and save in C:\baseline.
4. Select “Start”, click “Run”, and type “C:\BaseLine\baseline923.exe” to run the command as Figure 4.1a and Figure 4.1b.
5. Please enter your name and your company name (or school name) as Figure 4.1c.
6. To set up directory path, type “C:\BaseLine\MAXPLUS2” and “C:\BaseLine\MAX2WORK” in dialog and then follow the directions shown from Figure 4.1d to Figure 4.1f.
7. After start running MAX+PLUS II Baseline, read the license agreement as Figure 4.2a and 4.2b. Copy protection information will be shown out as in Figure 4.2c since we haven’t had authority and software guard key yet. Click “Yes” and find the message as Figure 4.2d. You will know the Baseline restrictions and ways of getting the license.

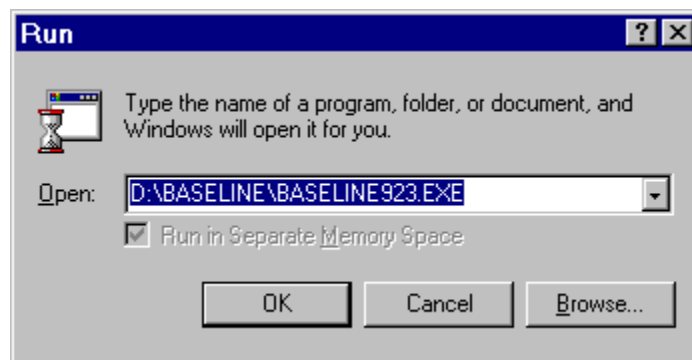


Figure 4.1a Baseline 9.23 setup

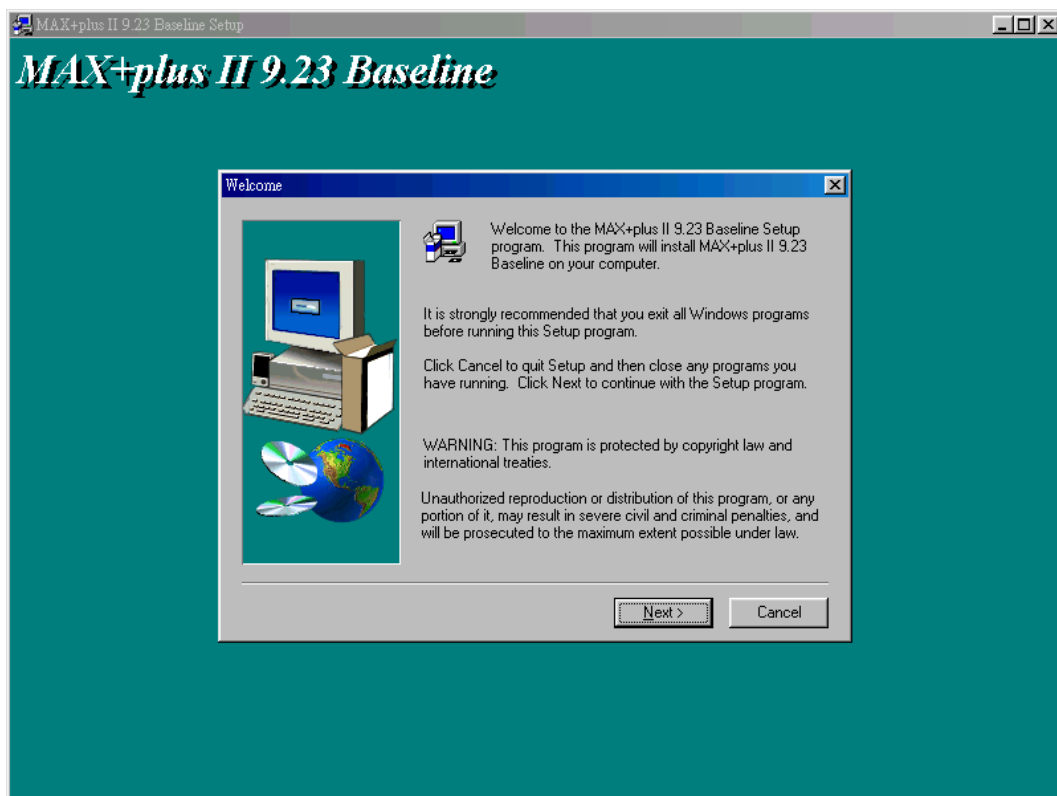


Figure 4.1b Baseline 9.23 setup (continue)

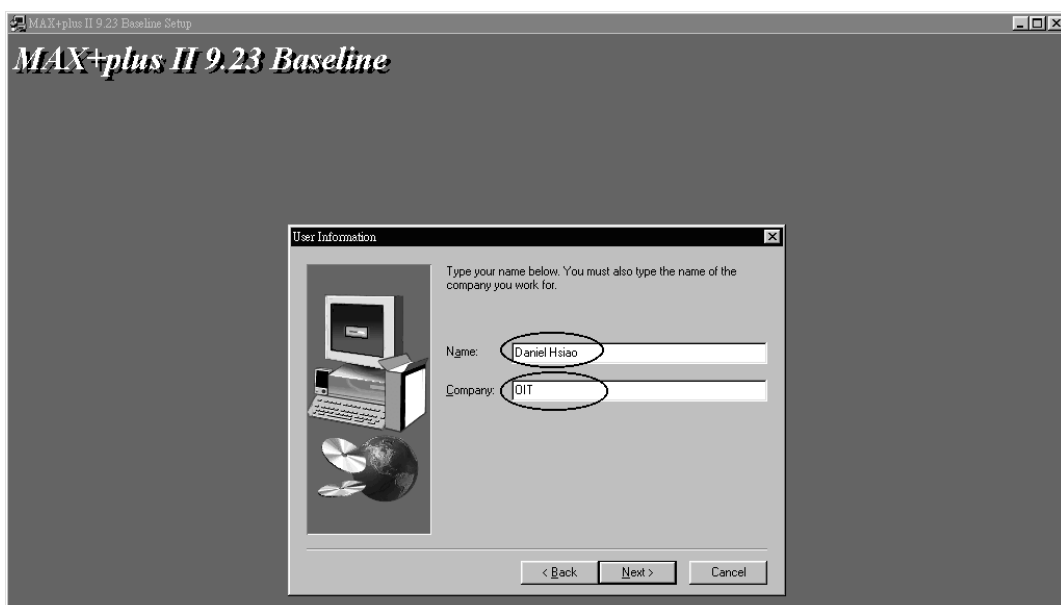


Figure 4.1c Baseline 9.23 setup (continue)

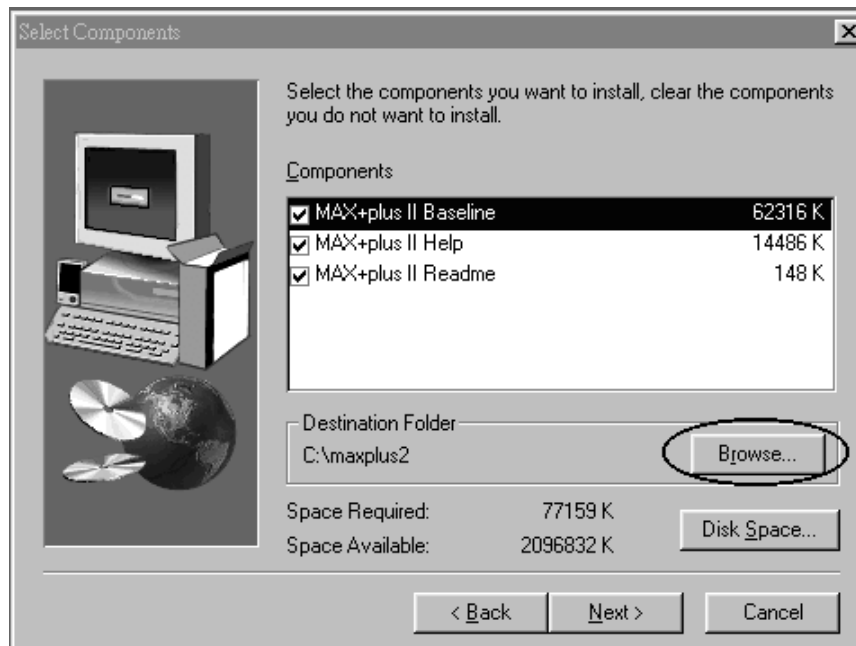


Figure 4.1d Baseline 9.23 setup (continue)

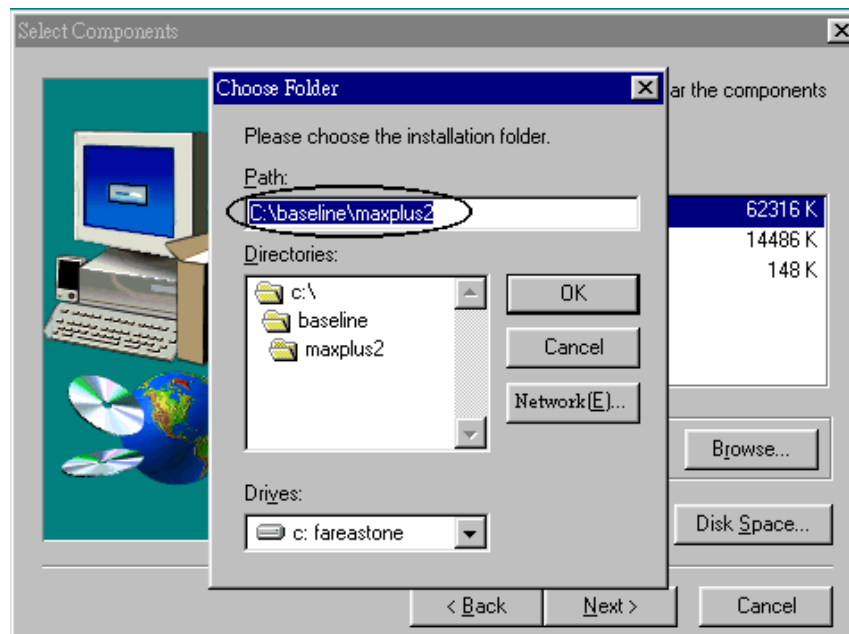


Figure 4.1e Baseline 9.23 setup (continue)

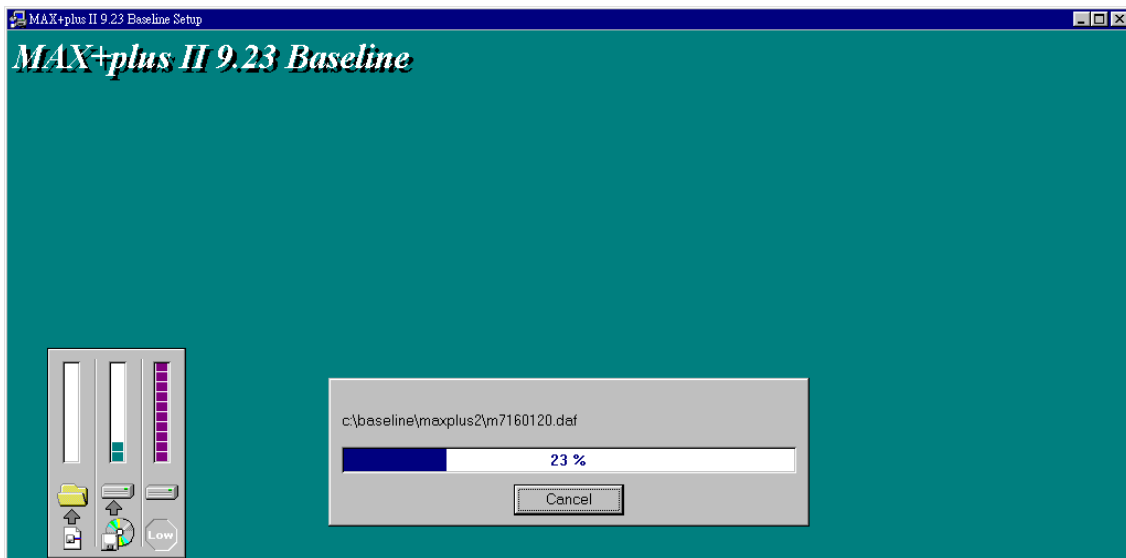


Figure 4.1f Baseline 9.23 setup (continue)

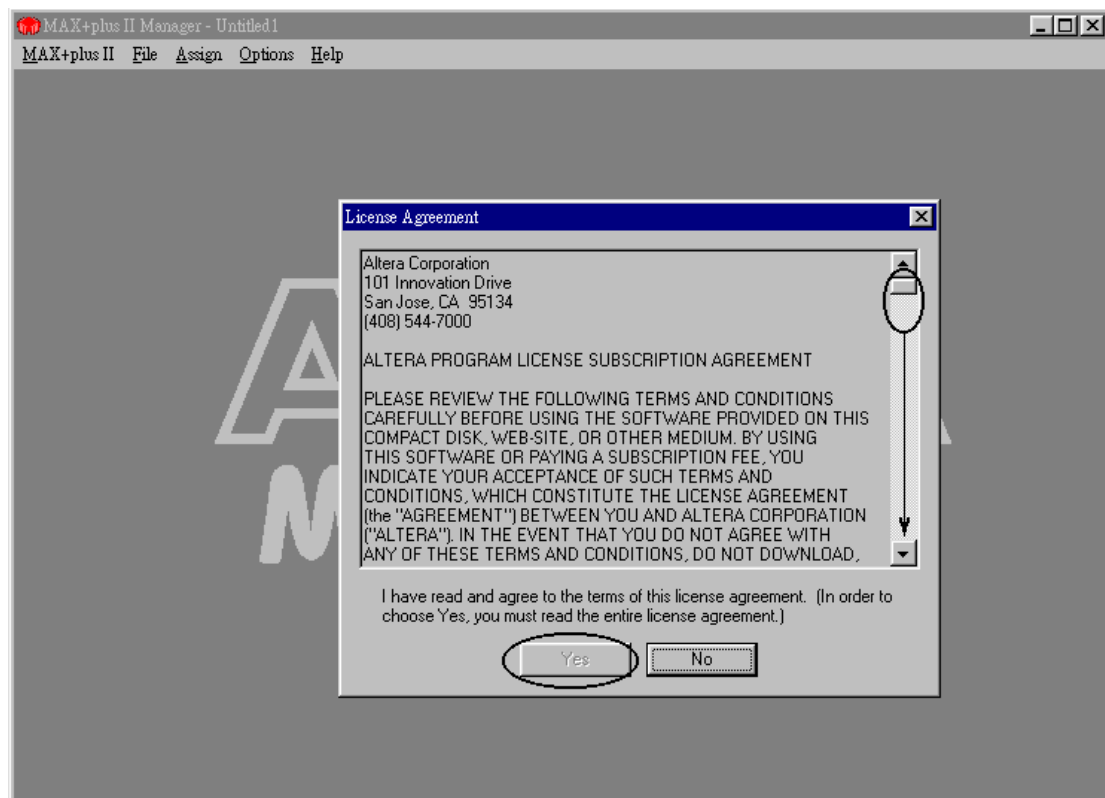


Figure 4.2a Baseline 9.23 license obtainment and setup

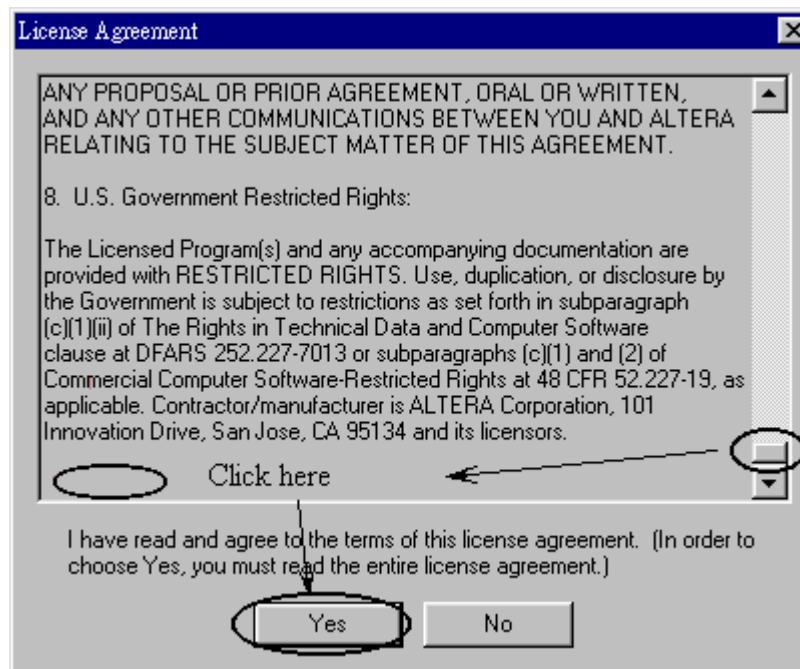


Figure 4.2b Baseline 9.23 license obtainment and setup (continue)

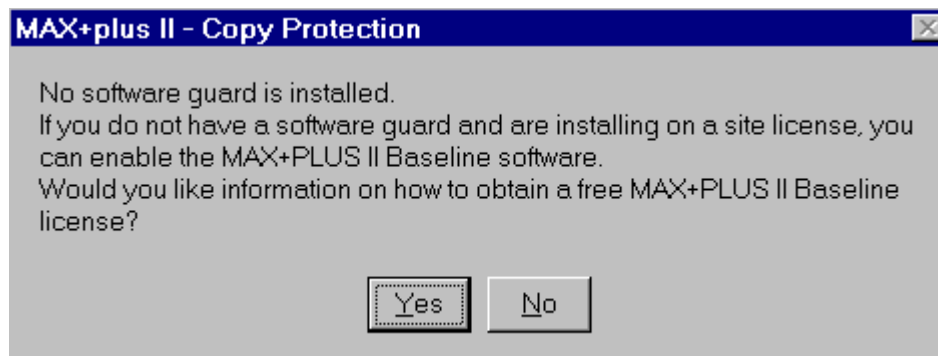


Figure 4.2c Baseline 9.23 license obtainment and setup (continue)

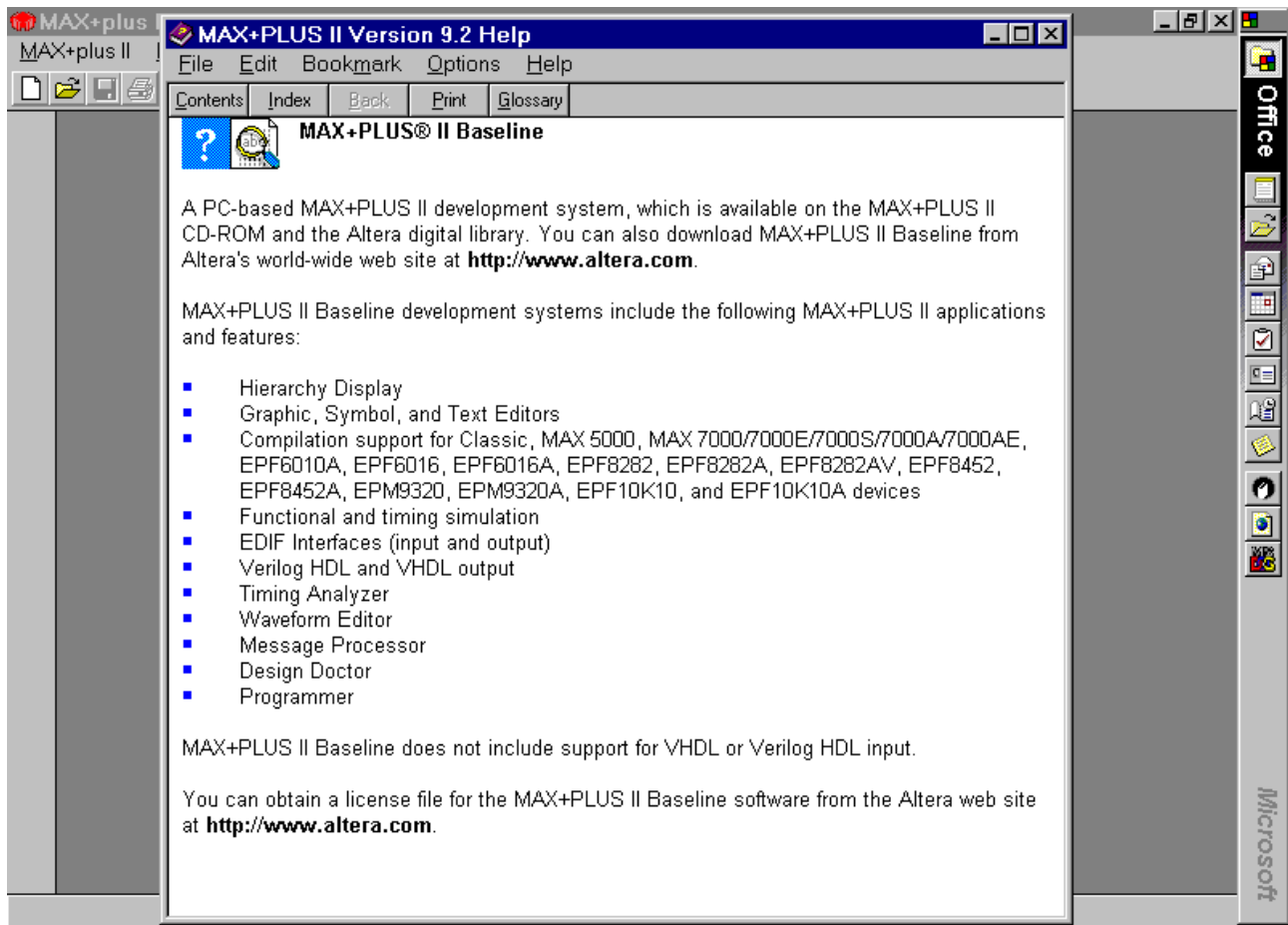


Figure 4.2d Baseline 9.23 license obtainment and setup (continue)

8. To have Baseline 9.23 license, please choose “Option” and then “License Setup” as in Figure 4.2e. Click “System Info...” later as in Figure 4.2f. You could find your hard drive serial number and write it down as in Figure 4.2g. We need the number to apply the license on ALTERA website.
9. Go to the <http://www.altera.com/> website as Figure 4.2h. Select “MAX+PLUS II Licenses & Authorization Codes” on the web site indicated as in Figure 4.2h. Click the first item “Free MAX+PLUS II Baseline software” on MAX+PLUS II Licensing web page as in Figure 4.2i. Carefully enter your hard drive serial number in the blank area circled as in

Figure 4.2j, and fill out your information as in Figure 4.2k. Make sure you give correct email account before send your application. You will get confirmation from ALTERA soon after ALTERA receives your application.



Figure 4.2e Baseline 9.23 license obtainment and setup (continue)

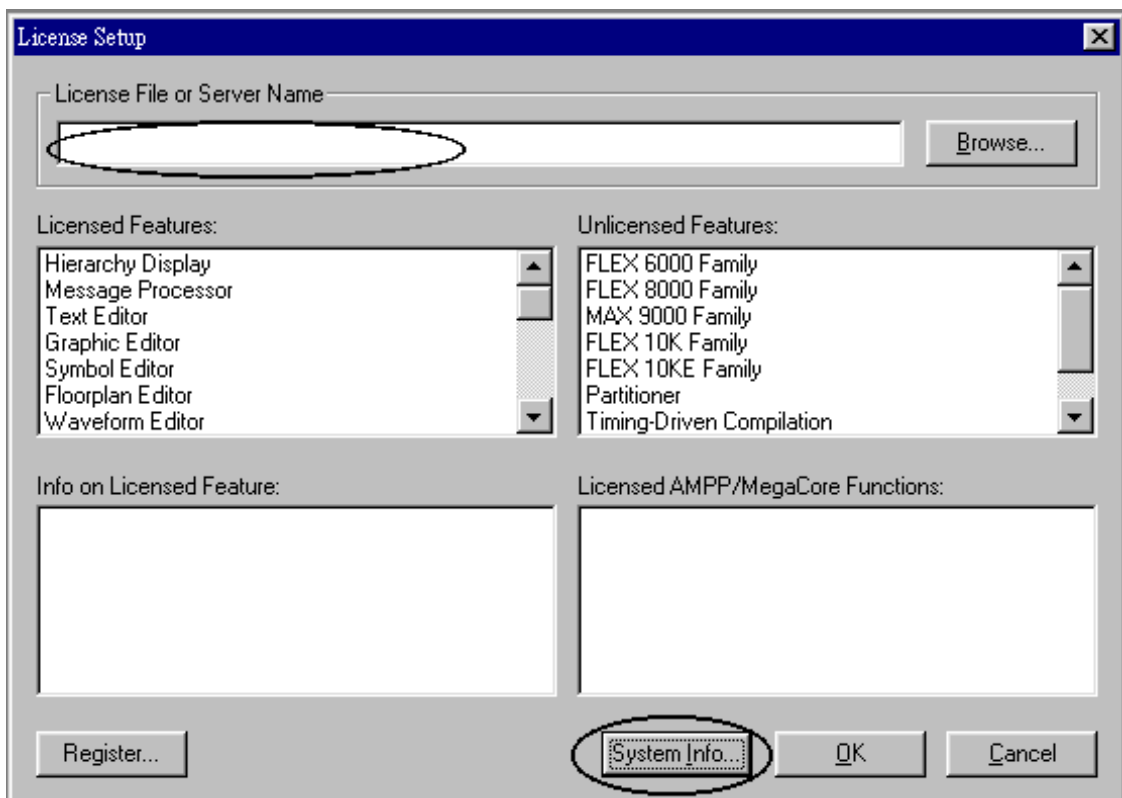


Figure 4.2f Baseline 9.23 license obtainment and setup (continue)

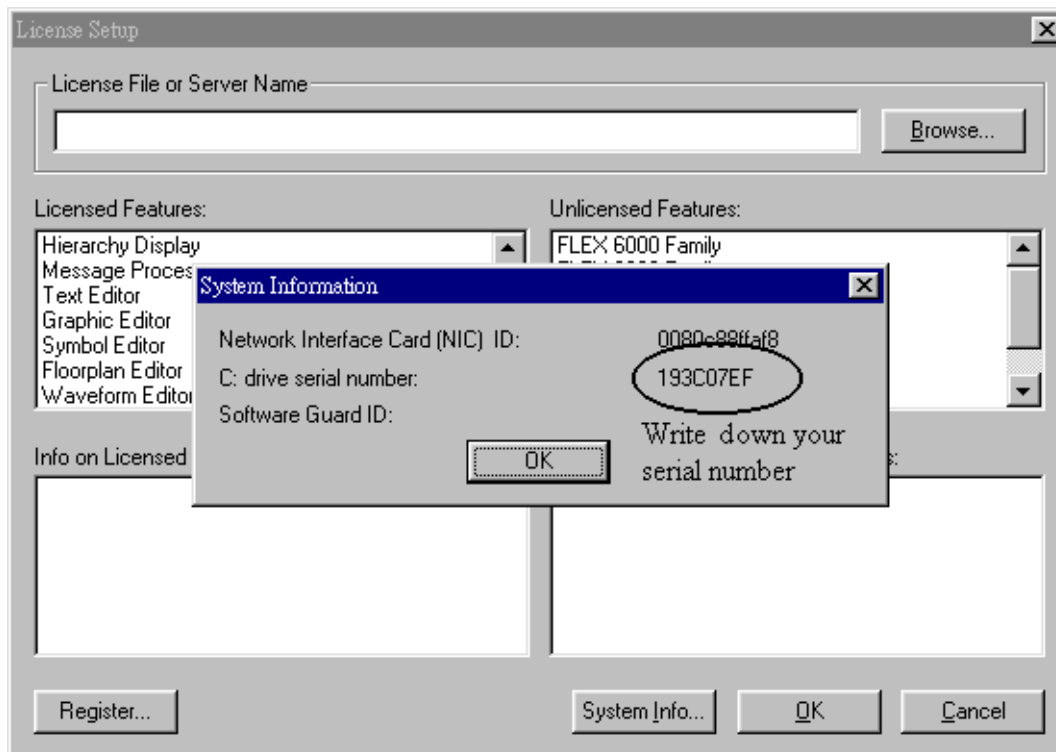


Figure 4.2g Hard drive serial number obtainment



Figure 4.2h Baseline 9.23 license obtainment and setup (continue)

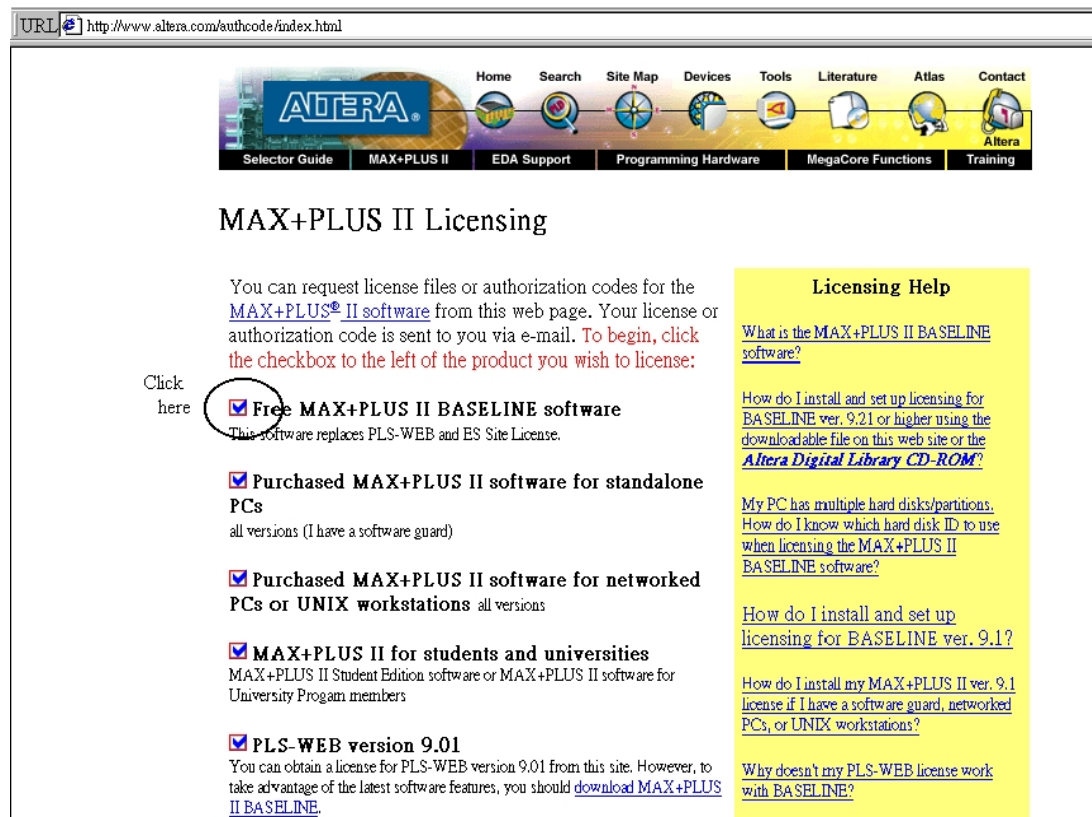


Figure 4.2i Baseline 9.23 license obtainment and setup (continue)

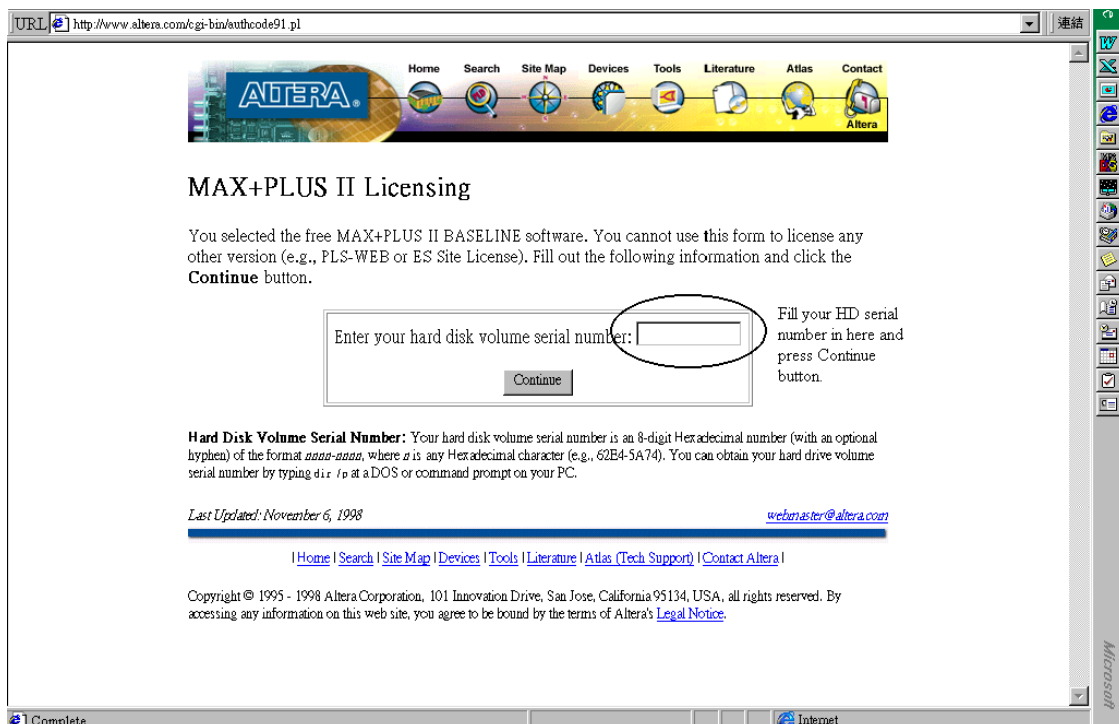
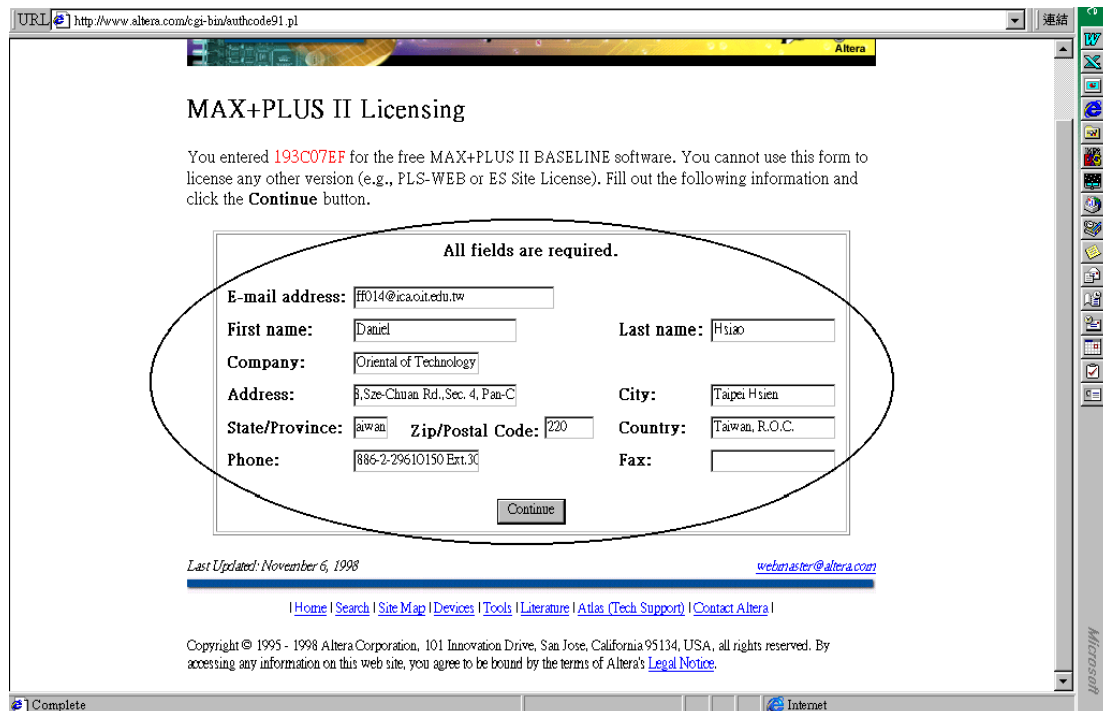


Figure 4.2j Baseline 9.23 license obtainment and setup (continue)



URL: <http://www.altera.com/cgi-bin/authcode91.pl>

MAX+PLUS II Licensing

You entered **193C07EF** for the free MAX+PLUS II BASELINE software. You cannot use this form to license any other version (e.g., PLS-WEB or ES Site License). Fill out the following information and click the **Continue** button.

All fields are required.

E-mail address:	ff014@ica.ut.edu.tw		
First name:	Daniel	Last name:	Hsiao
Company:	Oriental of Technology		
Address:	p. Sze-Chuan Rd., Sec. 4, Pan-C	City:	Taipei Hsien
State/Province:	Taiwan	Zip/Postal Code:	220
Country:	Taiwan, R.O.C.		
Phone:	886-2-29610150 Ext.30	Fax:	

Continue

Last Updated: November 6, 1998 webmaster@altera.com

[Home](#) | [Search](#) | [Site Map](#) | [Devices](#) | [Tools](#) | [Literature](#) | [Atlas \(Tech Support\)](#) | [Contact Altera](#)

Copyright © 1995 - 1998 Altera Corporation, 101 Innovation Drive, San Jose, California 95134, USA, all rights reserved. By accessing any information on this web site, you agree to be bound by the terms of Altera's [Legal Notice](#).

Figure 4.2k Baseline 9.23 license obtainment and setup (continue)

10. When you receive the mail from ALTERA, save "license.dat" in C:\baseline.
11. Please re-start MAX+PLUS 9.23 BaseLine, click the buttons "Option" and "License Setup". You will see Figure 4.2f appearing on your screen. Enter "C:\baseline\license.dat". It will show as Figure 4.2L, and click OK.

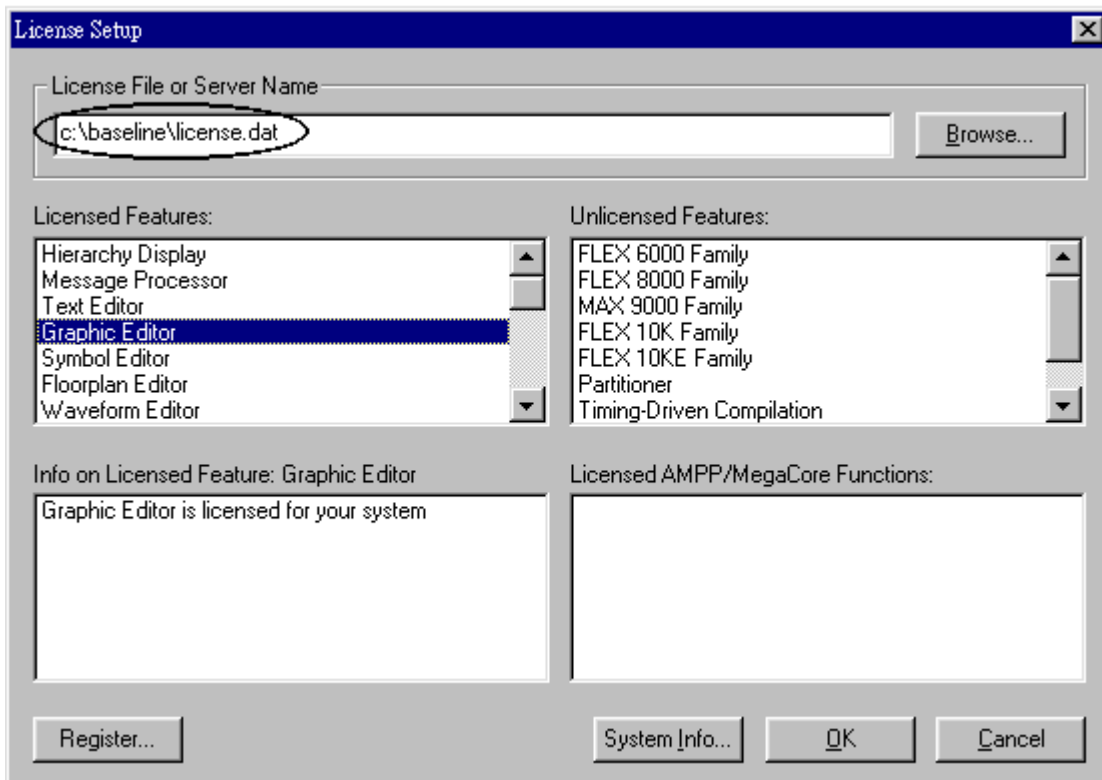


Figure 4.2L Baseline 9.23 installation procedures and licensing acquisition
(continue)

*Congratulation, you are now already download MAX+PLUS 9.23 BaseLine, and
are authorized to use this program.*

4.2 How to Use Mouse

Mouse is widely used in the PC world. The advantage of using mouse as one kind of computer appliance is it is able to create mutual communication for both users and computer. Nowadays, mouse is acknowledged as most efficient way to scan desktop and to deliver messages. Mouse is especially more effective in graphic design environment. In this case, a regular two-button mouse could be adopted in MAX+PLUS II environment.



❖ Mouse Operations

Cursor: Indicate the place where mouse located on the screen; in different context mouse has shown on different shapes and different meanings.

Click: Fast press once and release button.

Double click: Fast press twice and release the left button.

Drag: Hold down the bottom and move.

Drag-drop: Position the mouse pointer over an object on your screen then press and hold down the button, move the mouse to where you want to place the object then release the button.

❖ Shapes of Cursor and Useful Functions

Arrow: This is an arrow in NNW direction. The main function of arrow is to click selection, the objects, or items.

Hourglass: When hourglass appears on the screen, you might have to wait a while, because it shows MAX+PLUS II is currently processing.

Insertion-Point: This is an I-shape arrow shows you where the next characters you type will appear on the screen.

Finger: This is appears in standard toolbar to provide assistant options.

Question mark: Press F1 to show the mouse.

Square-icon: In Graphic Editor, press this button and draw a square shape.

Cross-icon: This shows the connection point.



4.3 Graphic Entry

Graphic entry is easy to learn and apprehend, the standard application procedure as follows:

1. Identify project name
2. Create new file name
3. Set up and display guideline
4. Enter primitives and macro functions
5. Remove, delete, recover and duplicate on circuit symbol
6. Connect pins
7. Identify the names of I/O pin and netlist
8. Save and check basic errors
9. Create a default symbol
10. Compile for functional simulation (“Functional compilation” will be used in the following discussion.)
11. Close the design file

We take the following case as an example of preventing keypad bouncing.

1. Identify project name

First of all, click the file button, press the project button, you will see Figure 4.3a. At this point, select the “name” icon once again, and Figure 4.3b will be shown. You can type “disbounce” under the project name or select “disbounce” if the project name has existed. Then click OK. You are now complete the process of selecting the project name. We always take “File > Project > Name” to represent the above procedures, and always remember to click “File” first; then click “Project”, and finally hit the “Name” button.

2. Create new file name (two methods)

(1) Method one:

- In the function bar, select “File” and click “New” as shown in Figure 4.4;
- In “File Type” dialog, click “Graphic Editor file” and “OK”, as shown in Figure 4.4b;
- Under “File”, click “Save As...” button as this procedure shows in Figure 4.4c;
- Type “disbounce.gdf” into the file name column, as it shown in Figure 4.4d.

(2) Method two:

- MAX+PLUS II > Graphic Editor to start graphic editor as Figure 4.4e;
- File > SaveAs...;
- Type “disbounce.gdf” into File Name field.

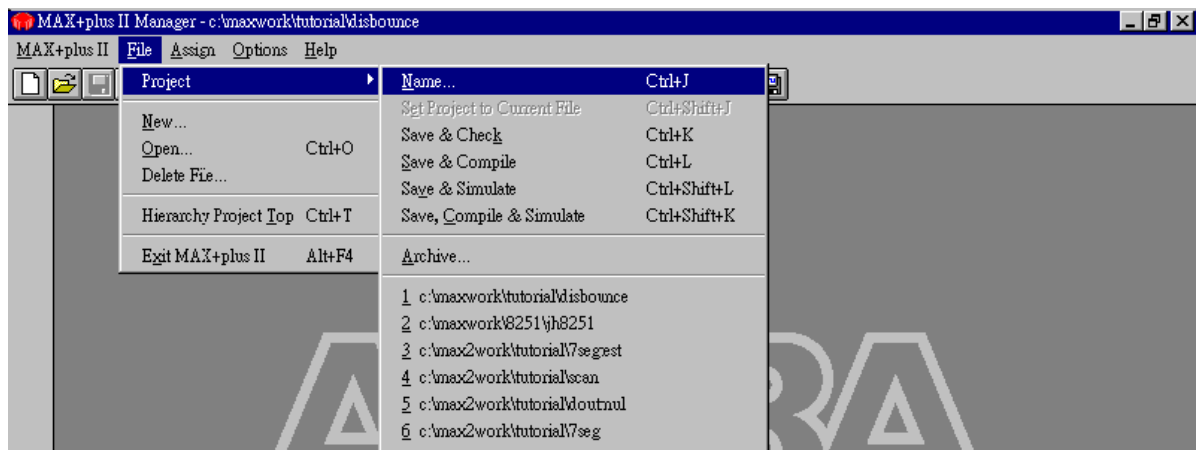


Figure 4.3a Identify project name

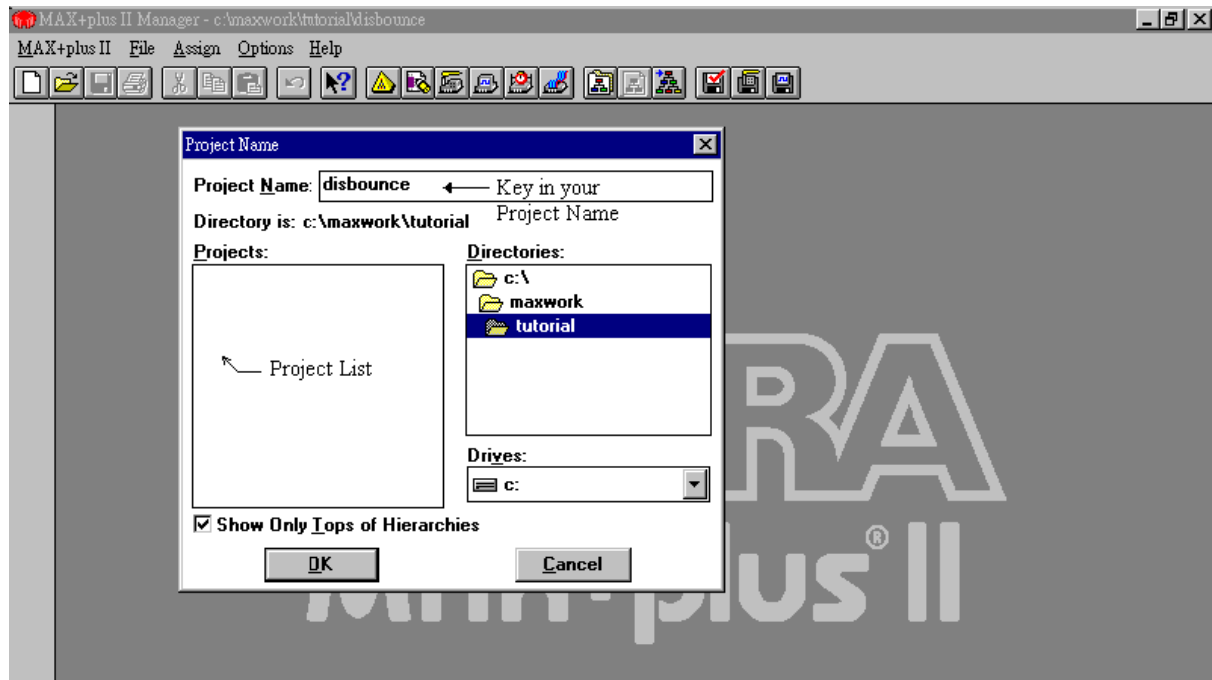


Figure 4.3b Identify project name (continue)

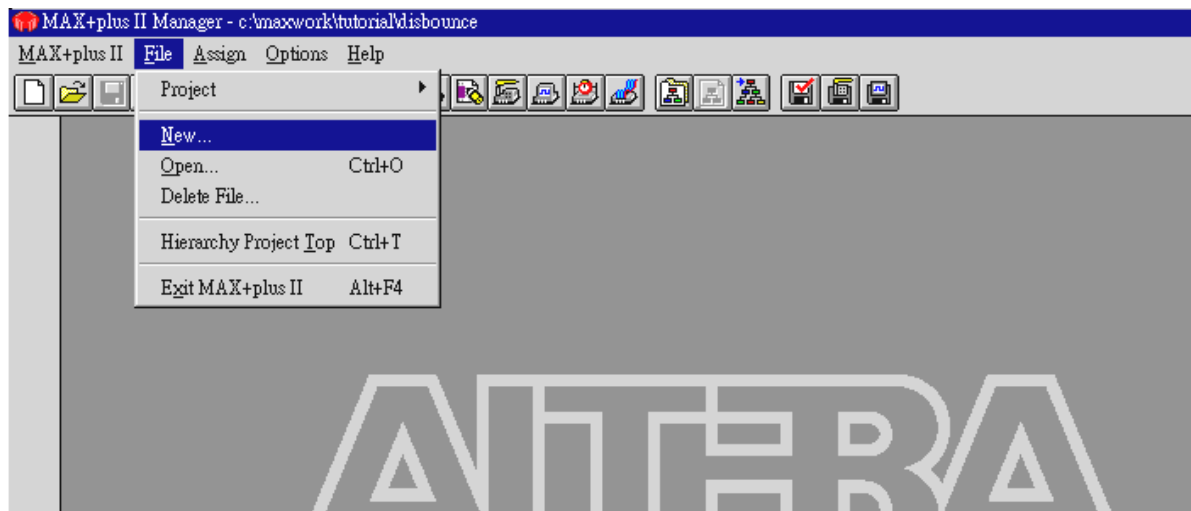


Figure 4.4a Create a new file (1)—select “New” under File function

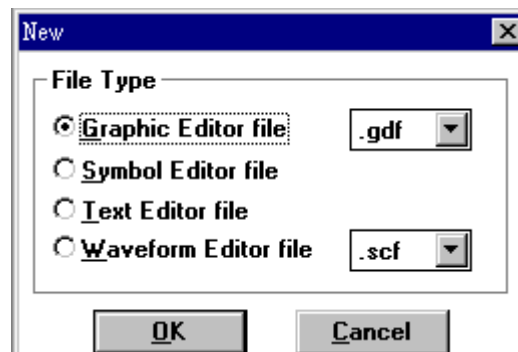


Figure 4.4b Create new file (2)—select “Graphic Editor file” and click OK in File Type dialog

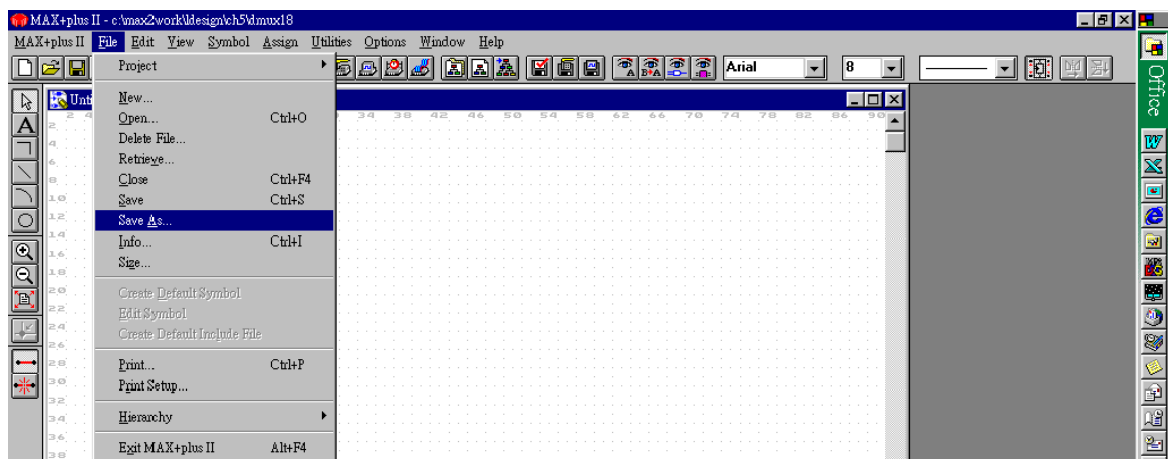


Figure 4.4c Create new file (3)—select “Save As...” in file function

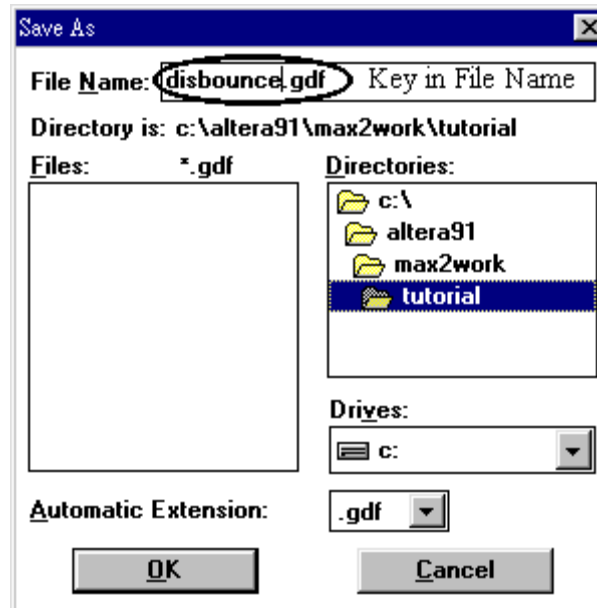


Figure 4.4d Create new file (4)—type “disbounce.gdf” in File Name dialog



Figure 4.4e Create new file (5)—select “Graphic Editor” in MAX+PLUS II

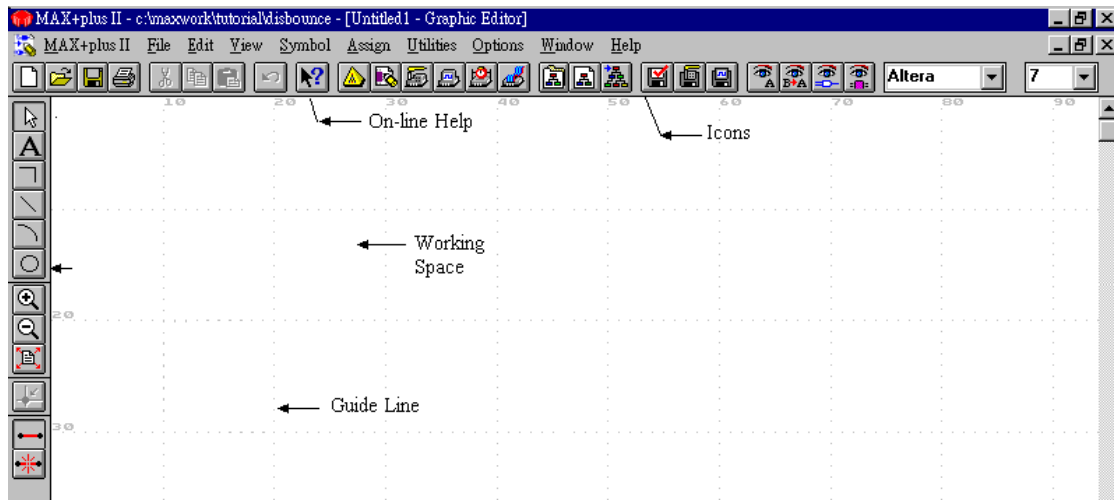


Figure 4.5 Graphic Editor environment illustration

3. Set up and display guideline

As Figure 4.5 shown, middle section is the main working area, where we construct circuit diagram. Over the top of working area, menu bar will appear and standard toolbar will be on left edge of the screen. They both provide various functions to assist constructing circuit diagram. You will see toolbar on the left side of the window, which displays a list of functional buttons to assist you drawing circuit diagram. When you point one of the buttons on the toolbar, it will describe functions of this particular icon on the left bottom of the window. You might click help on-line icon and select objects you want to have more information. Figure 4.6 shows how to use on-line help. Another way for help is to press F1 on your keyboard shown as in Figure 4.7.

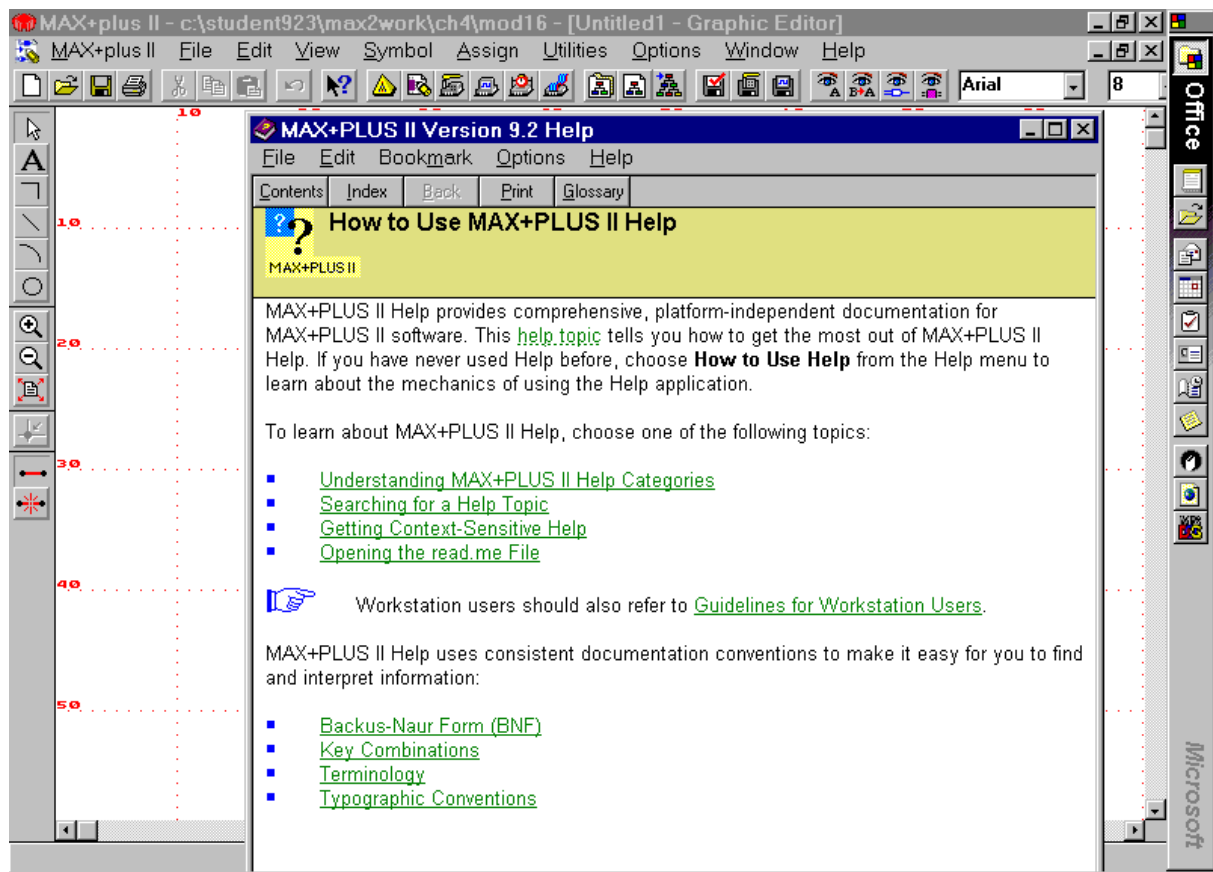


Figure 4.6 On-line help

On the “Options” bar, click “Show Guidelines” button, to display grid and guide lines. Spacing between XY-axis in grid and guidelines is able to set up by selecting “Guideline Spacing” under the “Options”. As Figures 4.8a and 4.8b, click “Color Palette” under Options, and you will see the information as in Figure 4.9 to select the colors of grid and guidelines. Under “Options”, you will see different functional options like Font 、Text Size 、Line Style and Rubber band

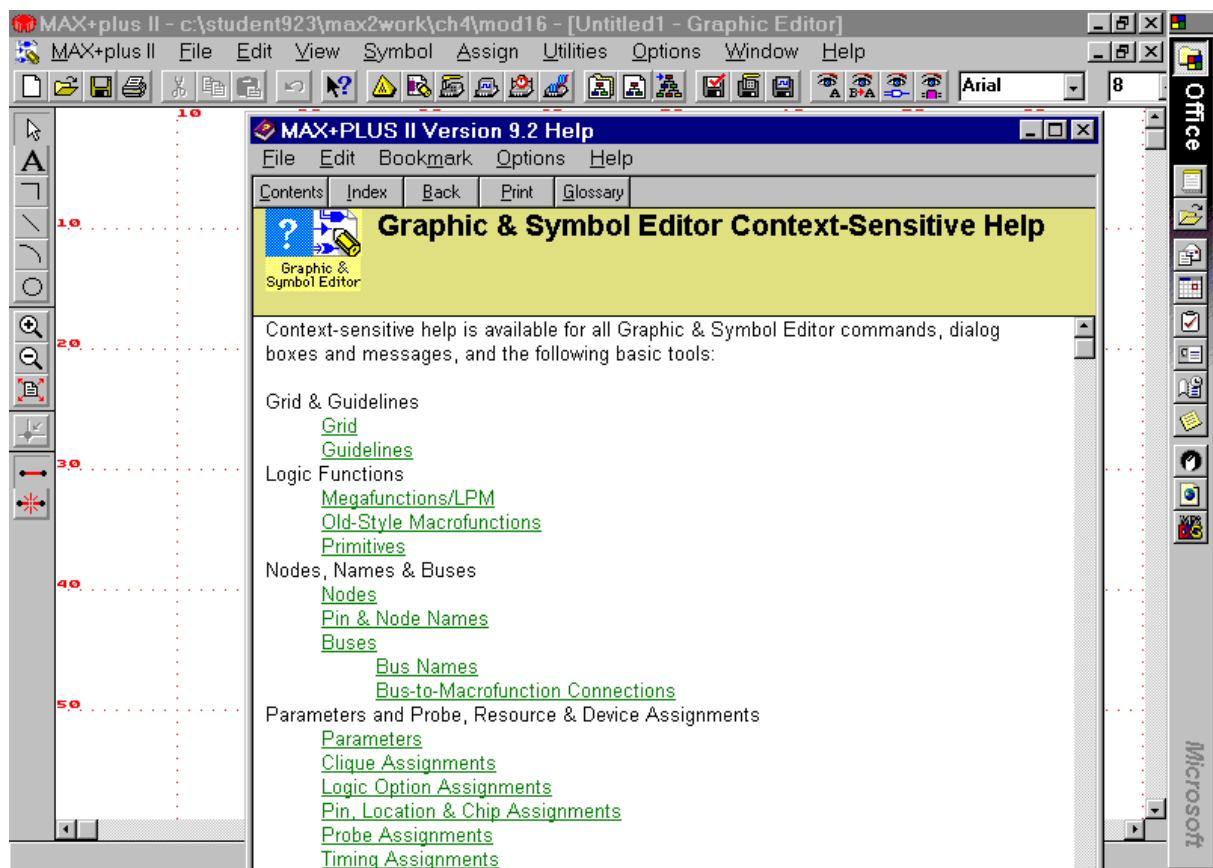


Figure 4.7 F1 function in Graphic Editor

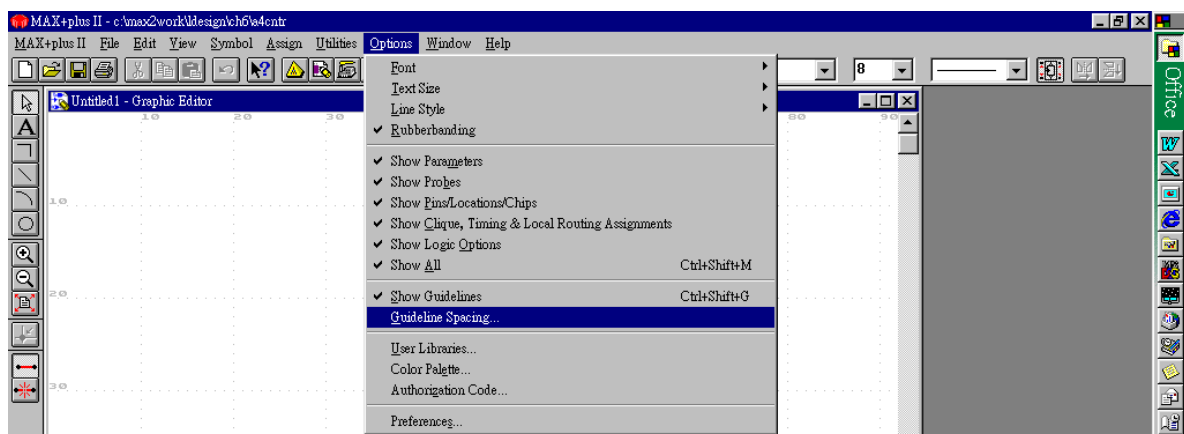


Figure 4.8a Set up spacing between XY-axis in grid and guide lines

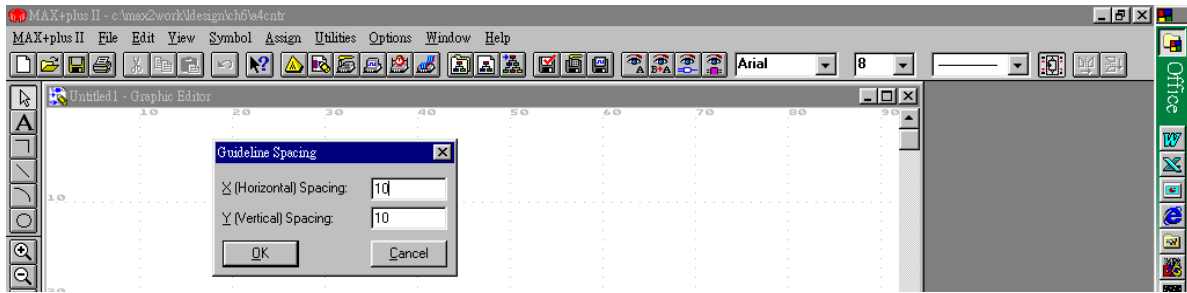


Figure 4.8b Set up spacing between XY-axis in grid and guide lines (continue)

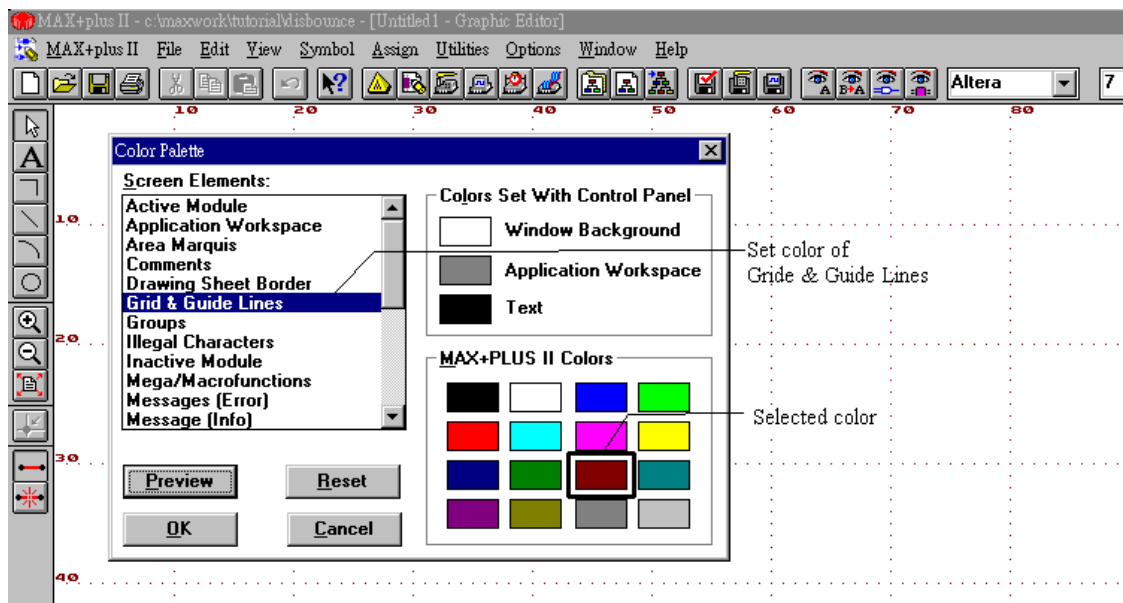


Figure 4.9 Set up the color of grid and guide lines

4. Enter primitives and macro functions

When you are in blank spaces in Graphic Editor window, click the left button of the mouse twice. You will see the window shown exactly the same as Figure 4.10, and you can type primitive names. Figure 4.11 shows that the DFF primitives have been entered. Please redo the same procedure for INPUT, VCC, NOT, and OUTPUT primitive entries. Those primitives are shown as Figure 4.12.

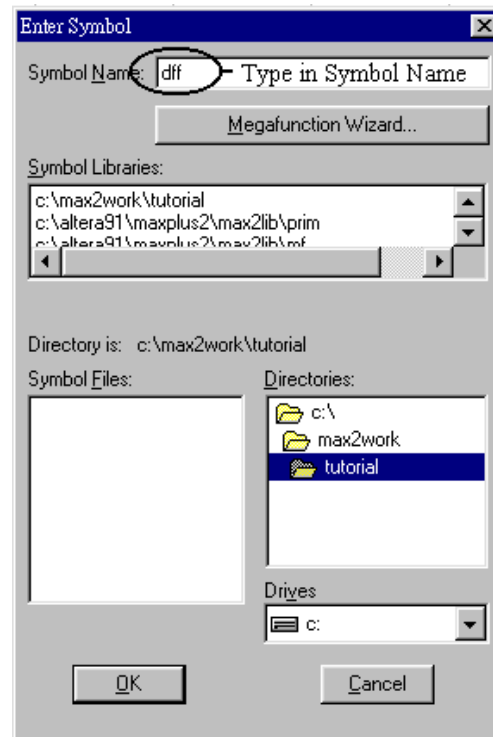


Figure 4.10 Primitive entry

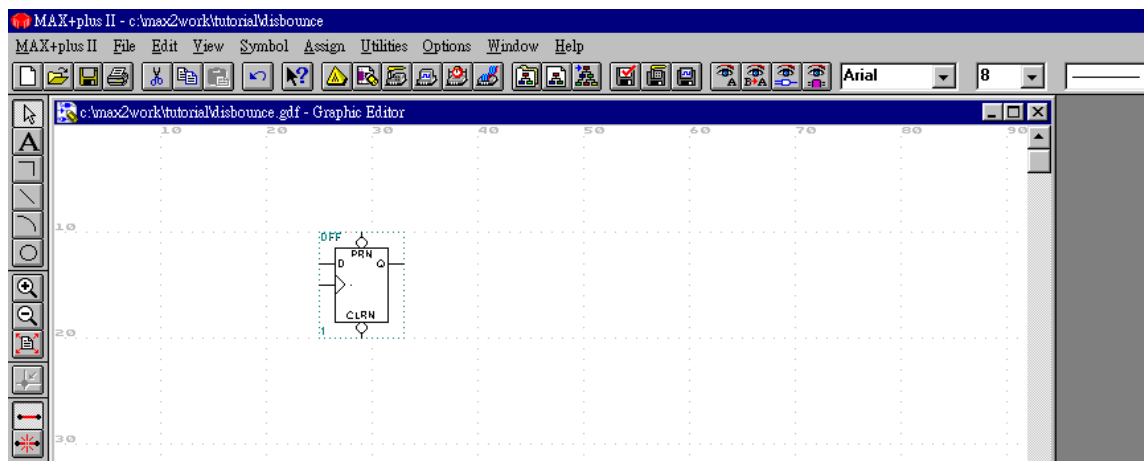


Figure 4.11 Primitive entry—DFF primitive

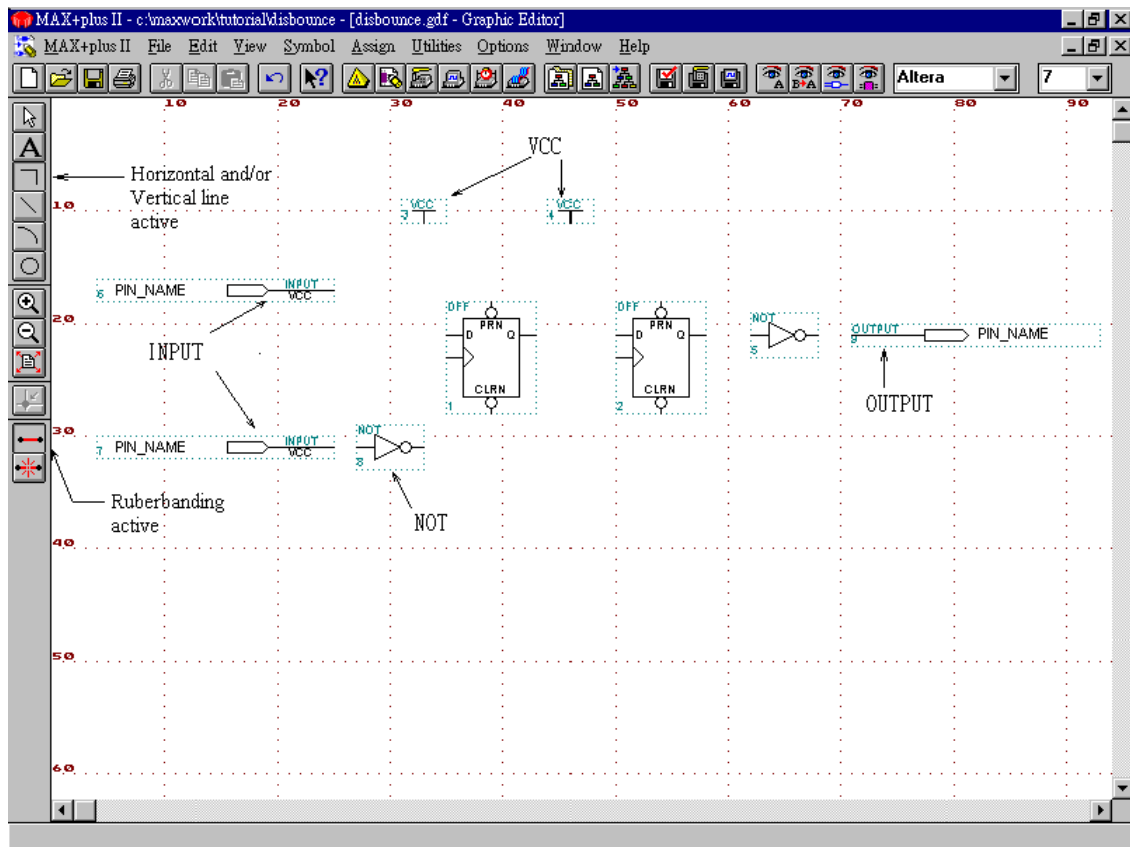


Figure 4.12 Completion of primitive entry

5. Delete, Remove, Retrieve, Copy Primitives and Macro Functions

- (1) Remove: If you wish to remove the primitive, use mouse to select an object, (the object will be framed by red lines) and drag the object to where you want to place.
- (2) Delete & recover: In order to delete a primitive, click the primitive first and press Del, (or select scissors icon) then you can delete this primitive. If you want to recover the object, press Ctrl + Z on your keyboard to get back.
- (3) Copy: Copying circuits are also frequently used in MAX+PLUS II environment. First, select the primitive, press Alt and Ctrl, then point at the object and drag it to where you want to place then release the button.

You could also pick the object first, and press Ctrl + C to copy the object, and move mouse to the place where you want and press Ctrl + V to paste the copied object.

6.Pin connection

In order to construct circuit connection, select “Line Style” under the “Options”, as shown in Figure 4.13. A thin line stands for singular connection, a bolder line represents for Bus connection. (Please be caution to click the bar appearing after selecting Line Style). Move the mouse to connection point (a cross cursor will show up). Click the left side button of the mouse and drag the mouse to where you want to route and connect. As the same method, draw the lines until a circuit is constructed. Figure 4.14 shows the connecting process and Figure 4.15 shows the final result of the connection.

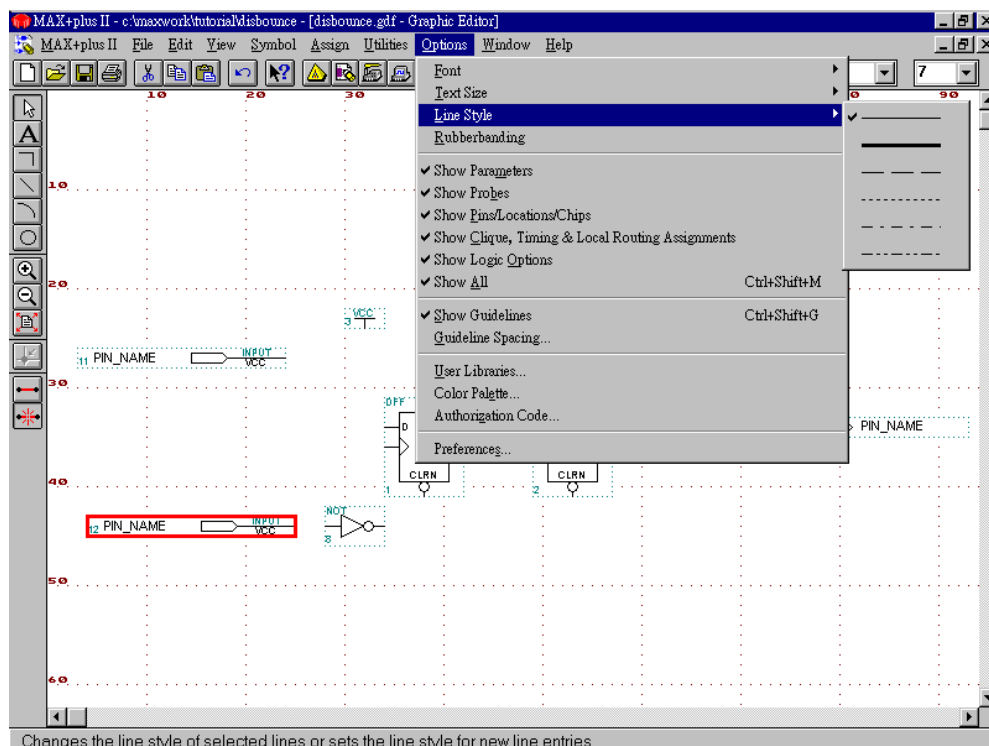
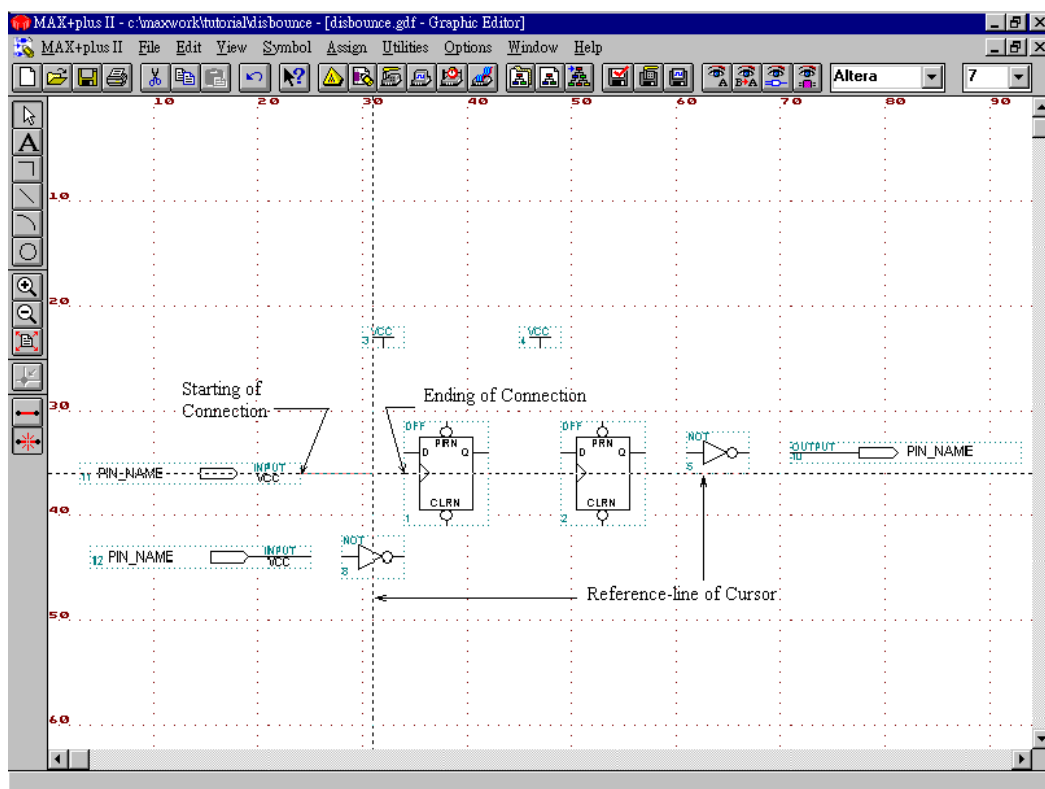


Figure 4.13 Selection of line style

7.Name I/O pin and netlist

As shown in Figure 4.5, to change or name the connection, click the left button on the mouse, click “PIN_NAME” to change the names of I/O pins. For naming netlist, move the cursor to the netlist and click it. The netlist will become red and a small square will show up. Start to enter a new name. Once naming is finished, if any changes are needed, double click the name



again (the area will inverse). Then, you can now enter the new name.

Figure 4.14 Pin connection

8.Save and check basic errors

Once you completed all the editing processes, select File > Project > Save & Check (or press Ctrl + K) to save the document and check all the basic errors. Figure 4.16a indicates this operation. After MAX+PLUS II checks all of the circuit, there might be

signals showing error messages in the diagram. You might follow the instruction to make correction. Then save and check again until no more errors exist (Figure 4.16b) before next steps. Please be aware of that File > Save (or Ctrl + S) can only save your document, but not check any basic errors.

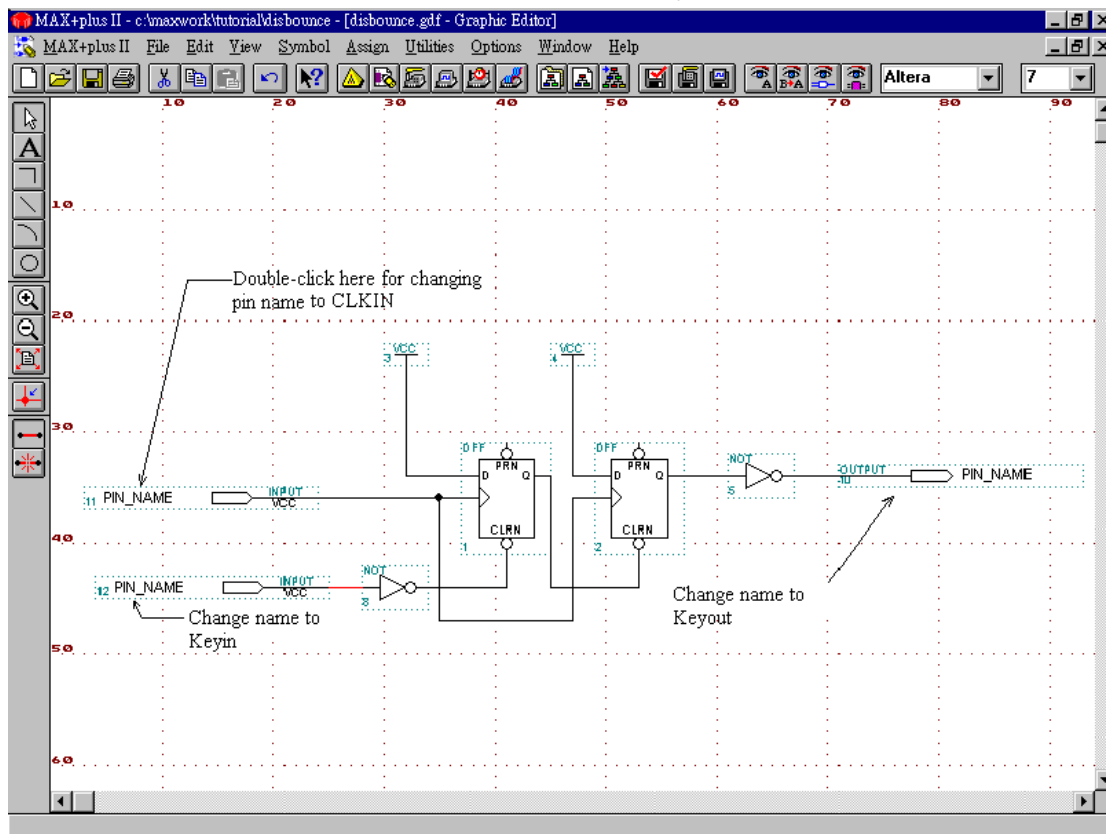


Figure 4.15 Final result of the connection

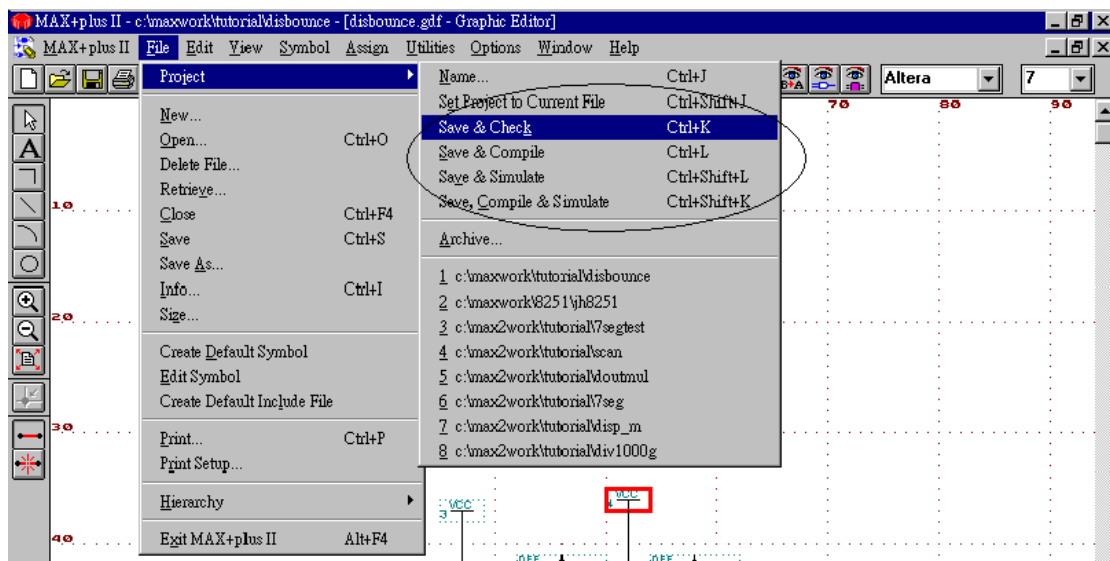


Figure 4.16a Save and check basic errors

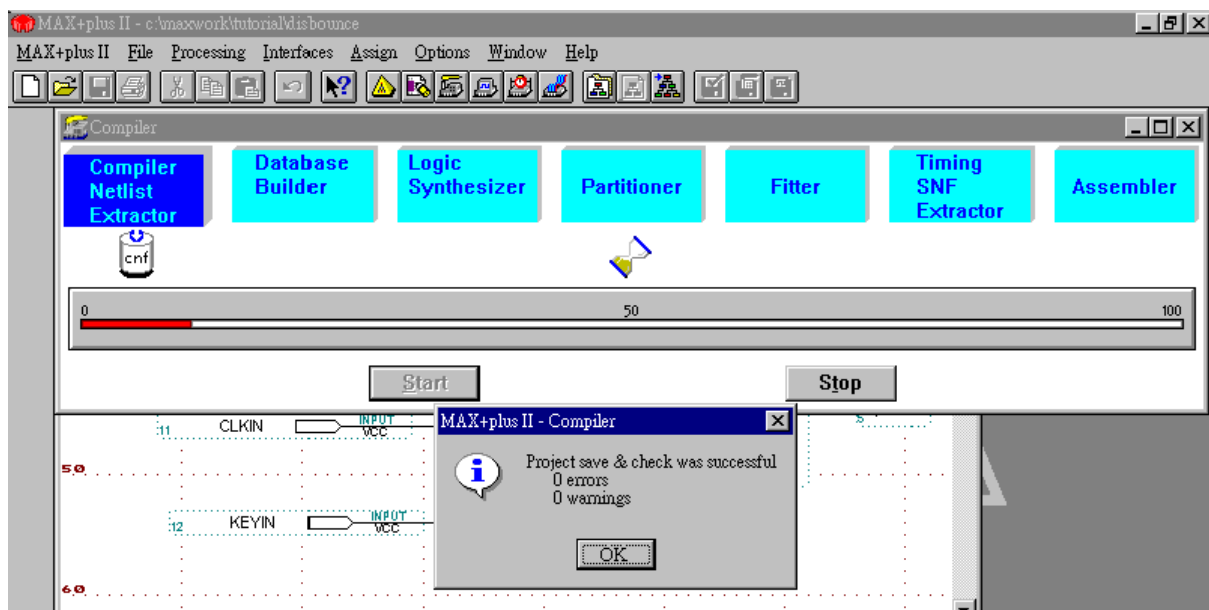


Figure 4.16b Completion of save and check

9. Create a default symbol

When you complete a new circuit or there are any changes on the names or counts of I/O pins, click File > Create Default Symbol to create a new symbol representing the newest circuit diagram for the upper circuit layer calling. Figure 4.16c shows the operation of creating a new symbol of the



circuit. Figure 4.16d shows the symbol for disbounce.gdf. Click File > Edit Symbol to make necessary steps of editing symbols.

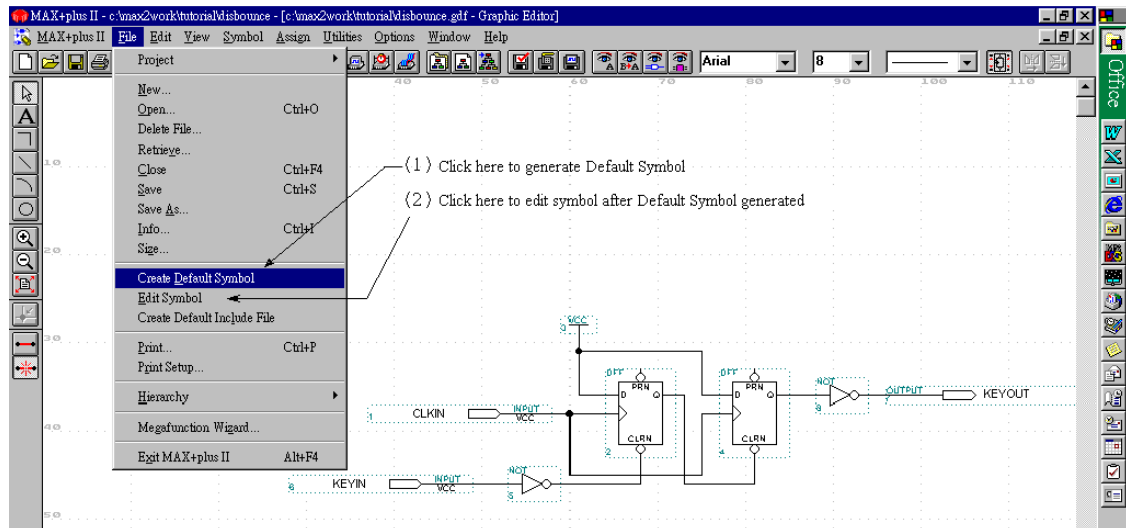


Figure 4.16c Creating a new circuit symbol after completing save and check basic errors

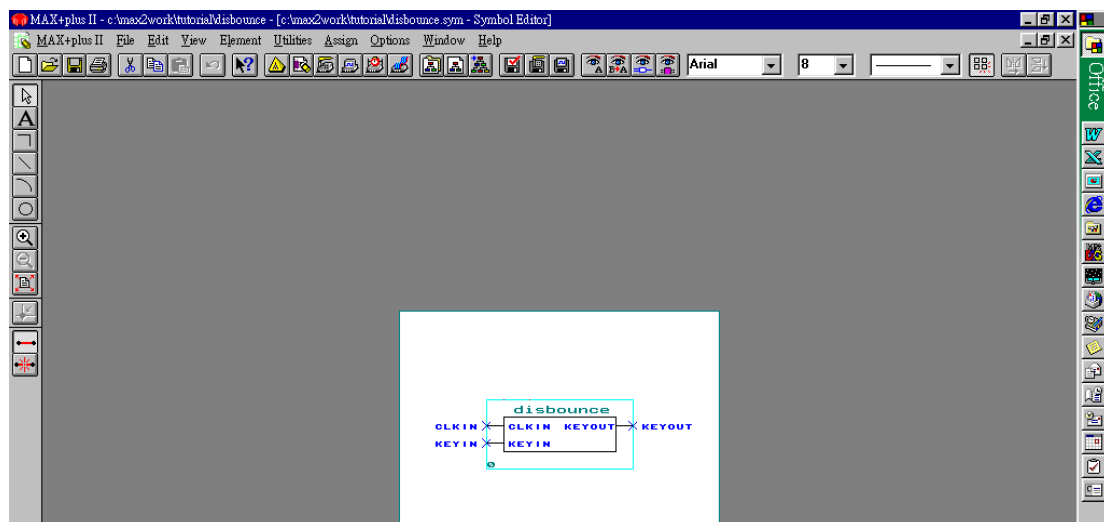


Figure 4.16d Disbounce.gdf symbol and symbol edit window

10.Functional Compilation

Congratulations! You have successfully reached to this step. The following are prepare steps for functional simulation:

- (1) Open compiler window by selecting MAX+PLUS II > Compiler as
- (2) Select Processing > Functional SNF Extractor to choose the compiler needed as Figure 4.18.
- (3) Click the start button to run compilation as Figure 4.19.

Make any error correction as needed. If there are no errors, congratulation, you have successfully completed the circuit diagram entry. Now, you can start the next functional simulation directly, or you might have a break from your desktop!

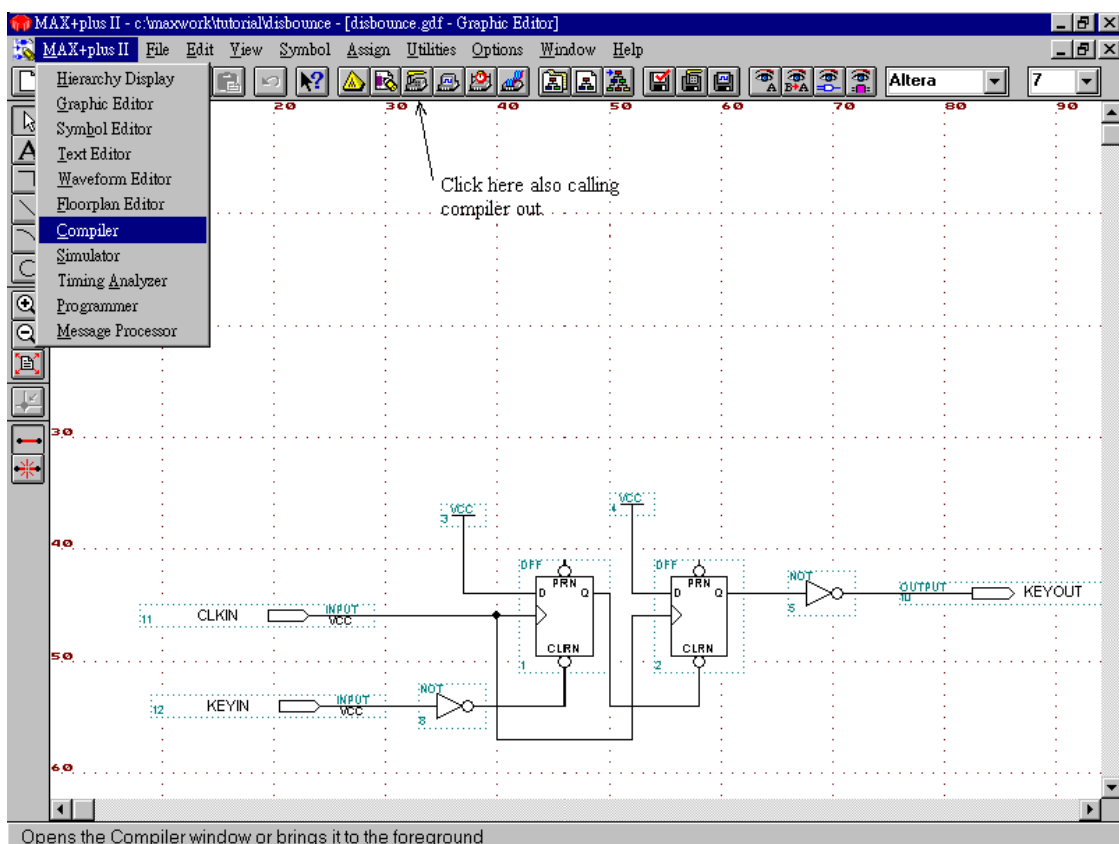


Figure 4.17 Compiler calling

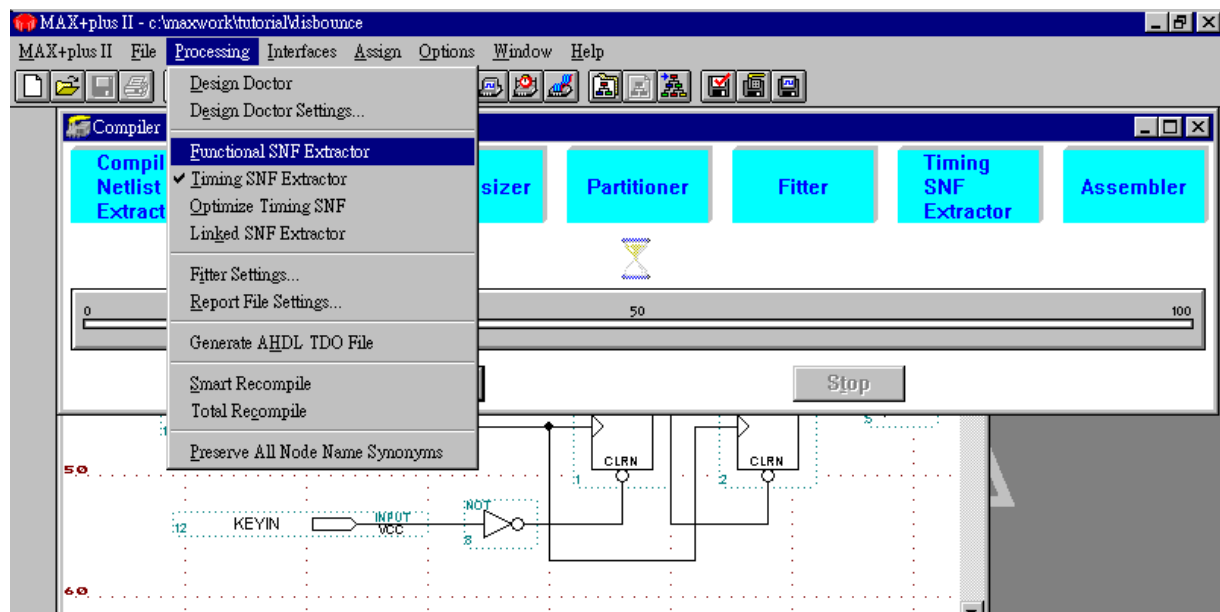


Figure 4.18 Select functions of compiler

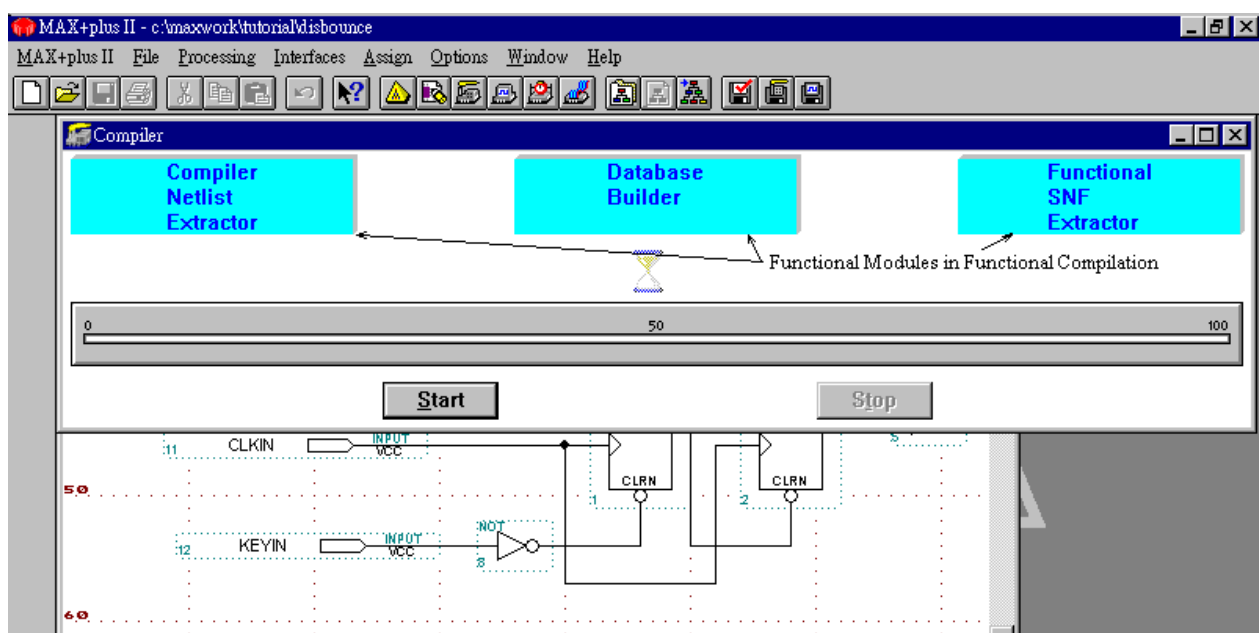


Figure 4.19 Start the functional compilation for functional simulation

11. Close the file

Click “File > Close “ to close the document, or click the close icon on the window.

4.4 Functional Simulation

As we introduced the utilities of compilation in Section 4.3, the functional module of Functional SNF Extractor have prepared data for the functional simulation of the circuit. However, it required further definition and identification of input signals and which output or internal nodes should be put extra notification. In general, it can be illustrated as textual files or waveform files. ALTERA provides both of these illustrations and it is easier to learn and understand by waveform illustration. Therefore, the functional simulation procedures using waveform illustration are as follows:

Open new file;

1. Select inputs, internal nodes, and outputs;
2. Define the waveforms of inputs;
3. Activate functional simulator to implement functional simulation;
4. Manage the errors and inspect simulation results;
5. Close file.

The following is an example of simulating a disbounce circuit. The whole procedures are illustrated by figures.

1. Open new file

Please open a new file by File > New, and click Waveform Editor File (Please note the extension file name is .SCF), as shown in Figure 4.20, to open a Waveform Editor file. Graphic 4.21 is an untitled waveform editor file. Please save this file and use the name of the project by “File > Save As...” as shown in Figure 4.22. Figure 4.23 illustrates another way to open an untitled waveform editor file by MAX+PLUS II > Waveform Editor.

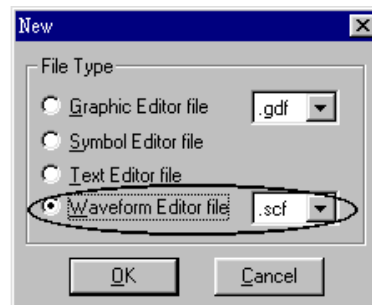


Figure 4.20 Open a new waveform editor file

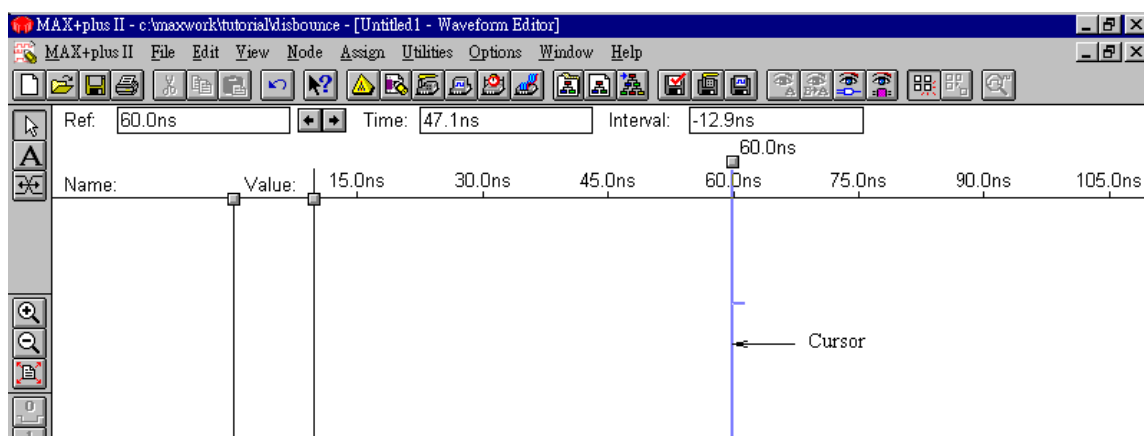


Figure 4.21 An untitled waveform editor file

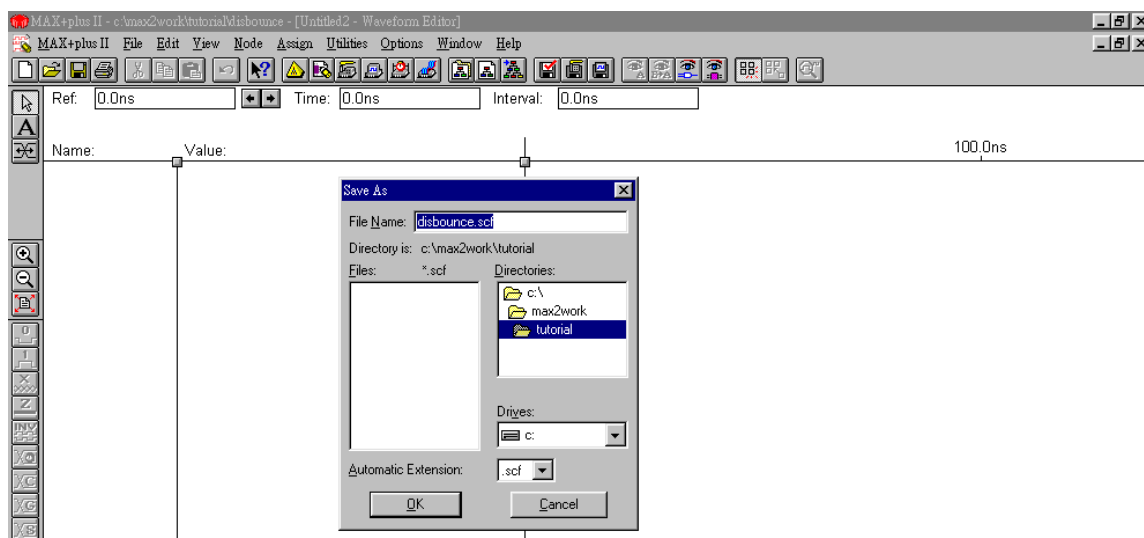


Figure 4.22 Use File > Save As...to name a waveform editor file

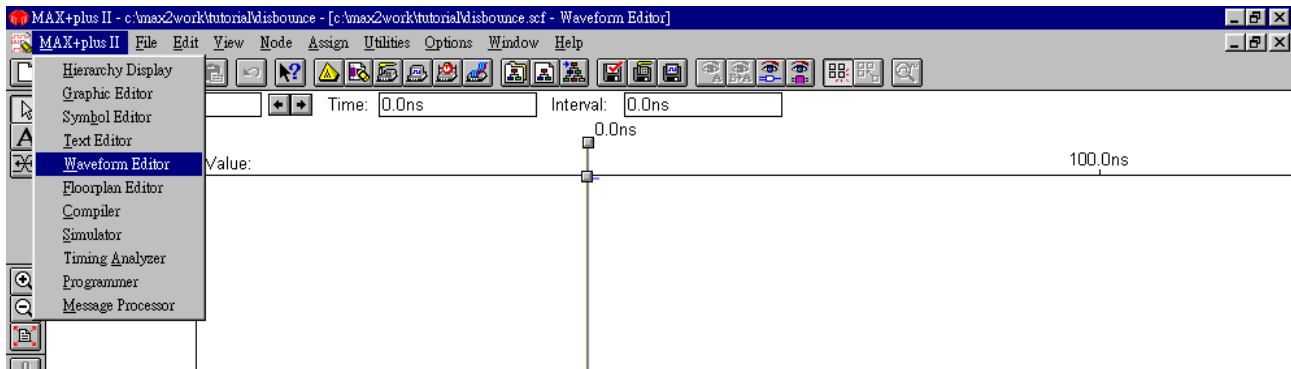


Figure 4.23 Another way to open an untitled waveform editor file

2. Select inputs, internal nodes, and outputs

The most convenient way to select the inputs, nodes, and outputs is to select these nodes from a SNF file, which is generated from the above section. Figure 4.24 shows the demonstration. From the menu of “Node” to select Enter Nodes from SNF...the windows of Figure 4.25a, Figure 4.25b would appear. Please note that the order of Node > Enter Nodes from SNF can work only after implementing the order of File > Save As.

In Figure 4.25a, click “(1)”, List, the SNF file would show proper node type (Type) and name (“*” demonstrate all the file names) and would list the available nodes and groups for you to choose. In this example, please press the left button on the mouse at “(2)” in the block of “Available Nodes and Groups” and drag to “(3)”. This action means you are ready to select the nodes of KEYIN [I], CLKIN [I] and KEYOUT [O]. At this time, please click “ \geq ”, the result would come up, as shown in Figure 4.25b. Click OK to complete the selection process. Figure 4.26 is the result of the selected inputs, nodes, and outputs. Figures 4.27a and 4.27b are for setting up grid size by “Options > Grid Size...”.

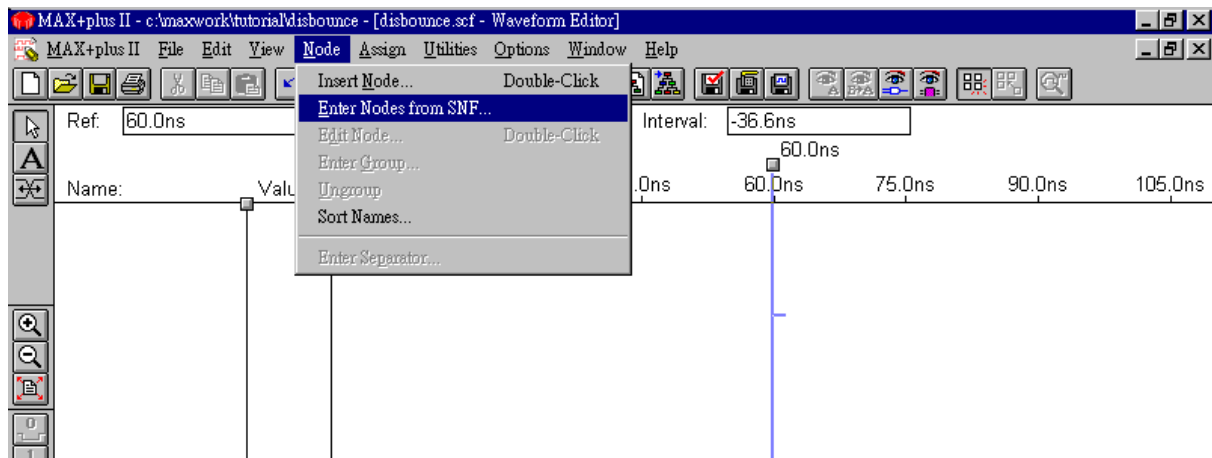


Figure 4.24 Select input, internal, and output nodes

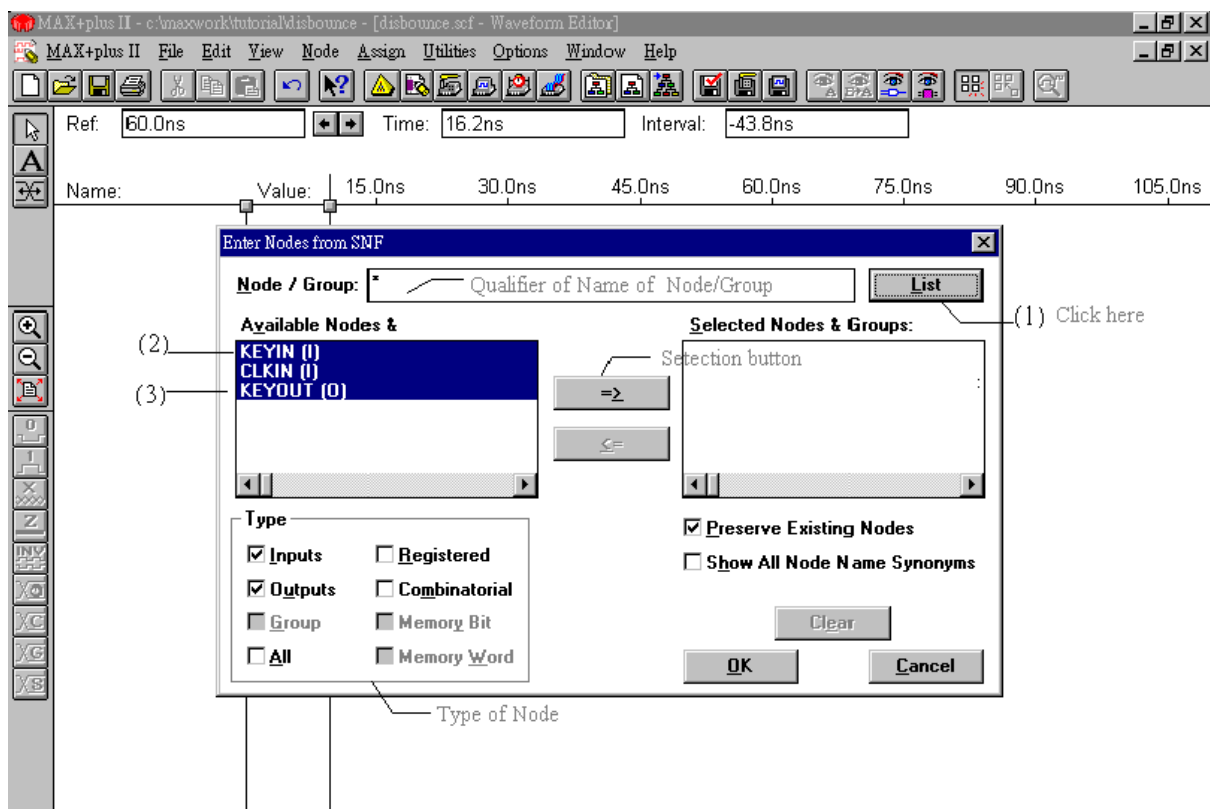


Figure 4.25a Nodes selection window

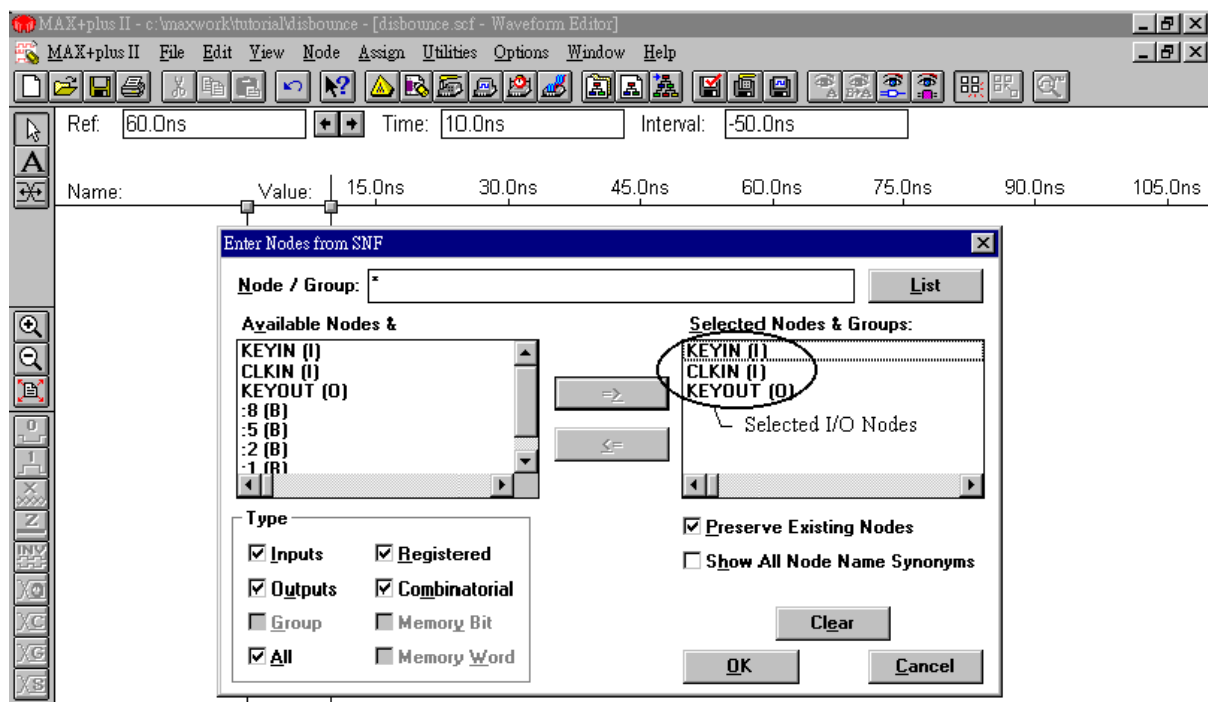


Figure 4.25b Node selection window

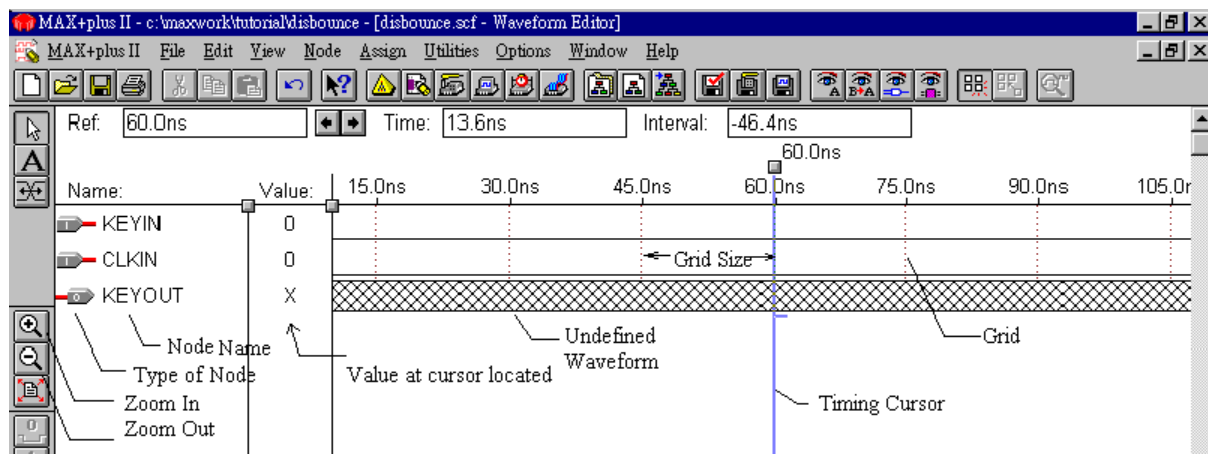


Figure 4.26 The result of the selected inputs, nodes, and outputs and the names of the columns

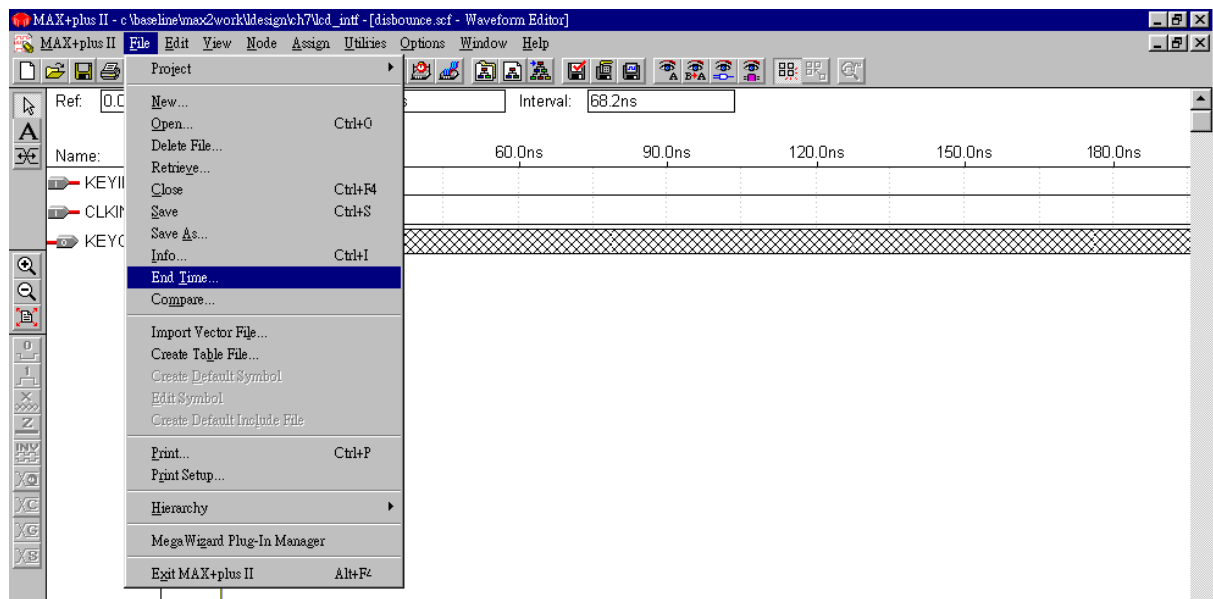


Figure 4.27a Options > Grid Size...Command to set grid size

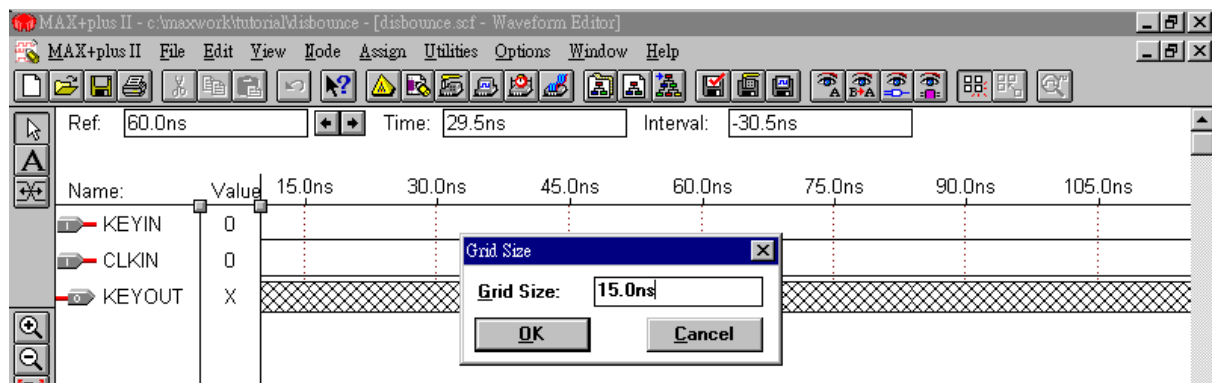


Figure 4.27b Setting End Time

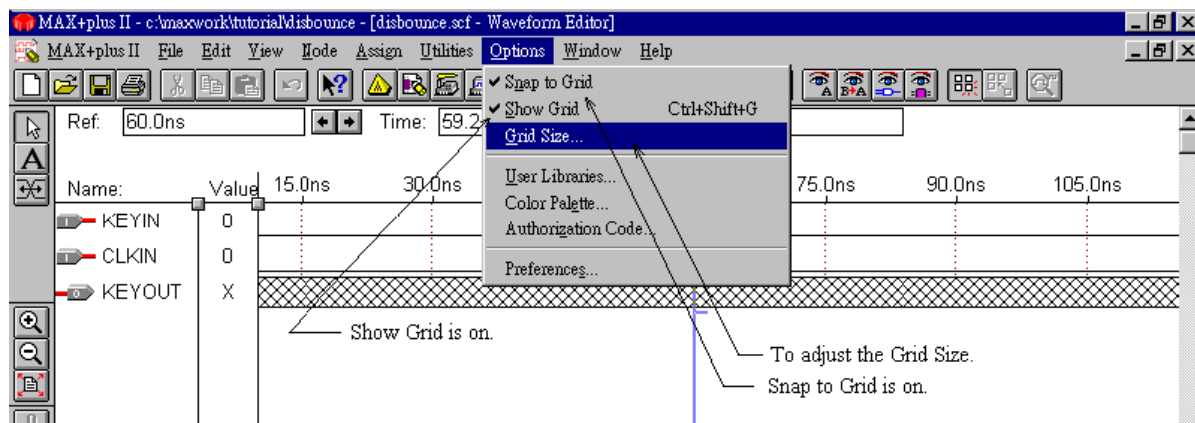


Figure 4.27c Set grid size by using Option> Grid Size... command.

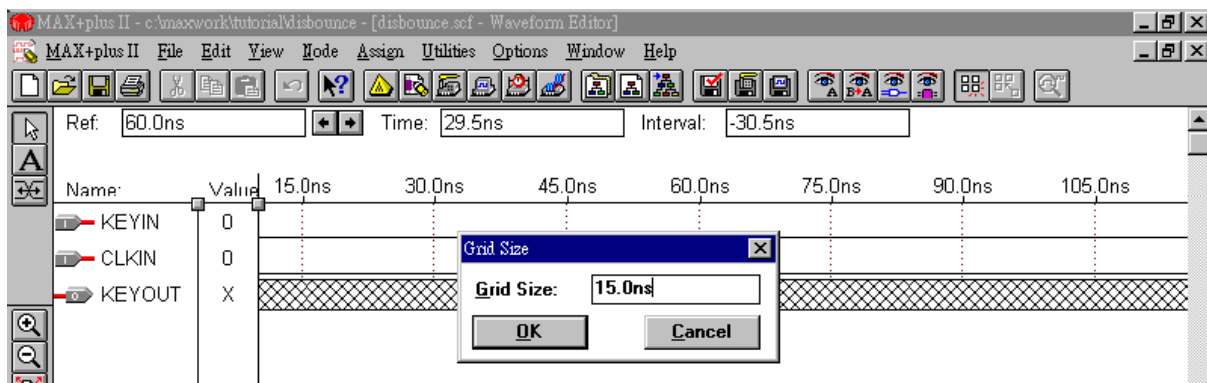


Figure 4.27d Set Grid Size equal to 15 ns

3. Input waveform definition

After completing the above steps, some original waveform data would be saved in the waveform file. Therefore, we have to utilize the following tools to define the waveform input for the later functional simulation. Please double click “Edit” in the Menu for the following operations, as shown in Figure 4.28.

- (1) Undo: Undo the last action. Can be done by Edit > Undo or Ctrl+Z.
- (2) Cut: Cut the selected waveform data block to clip board. Can be done by Edit > Cut or Ctrl+X.

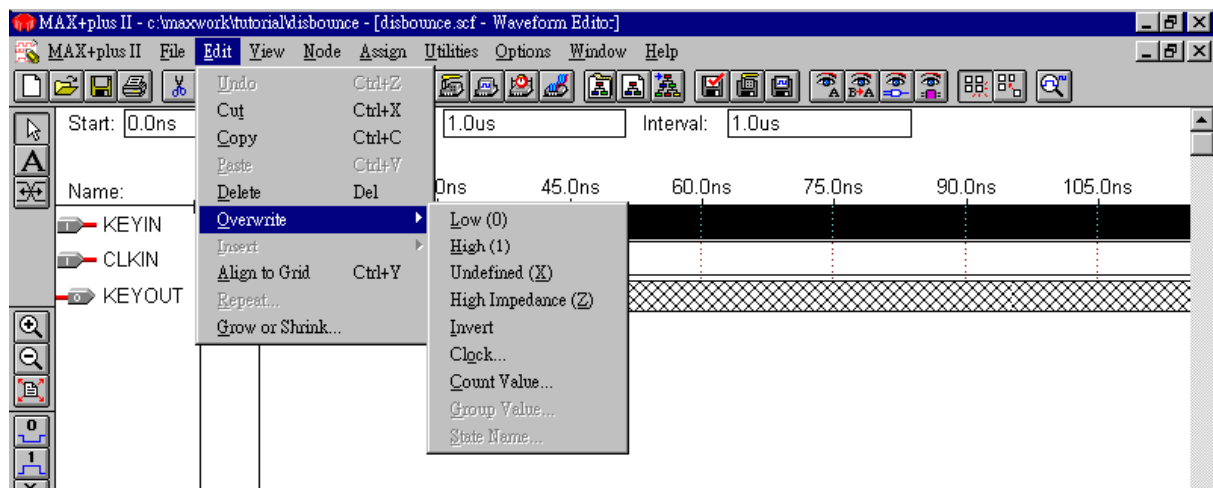


Figure 4.28 Waveform “Edit” menu

- (3) Copy: Copy the selected waveform data block. Can be done by Edit > Copy or Ctrl+C.
- (4) Paste: Paste the copied data once. Can be done by Edit > Paste or Ctrl+V.
- (5) Delete: Delete the selected data. Can be done by Edit > Delete or Del.
- (6) Overwrite: Overwrite the waveform data onto the selected waveform data block. Can be done by Edit > Overwrite. There are nine overwrite functions: Logic “1” signal, Logic “0” signal, unknown signal (x), High impedance signal (Z), invert signal, clock signal, counter value, group and state name overwrite. Please refer to Figure 4.28 and Figure 4.29a.
- (7) Insert: Insert the waveform data to the selected block. Can be done by Edit > Insert.
- (8) Align to Grid: Align the selected data to the grid. Can be done by Edit > Align to Grid or Ctrl+Y.
- (9) Repeat: Repeat the selected waveform block. Can be done by Edit > Repeat....

(10) Grow or Shrink: Horizontally grow or shrink the selected waveform data. It can be done by Edit > Grow or Shrink.

To the selected waveform data, there are three types also:

- (1) A block: As shown in Figure 4.29a. Press the left button of the mouse at the start point “(3)” and drag to “(2)”.
- (2) A Node: As shown in Figure 4.29b. Press the left button of the mouse at the start point “(1)” and the selected area will be blocked.
- (3) Nodes: As shown in Figure 4.29b. Press the left button of the mouse at “(1)” and drag to “(2)”.

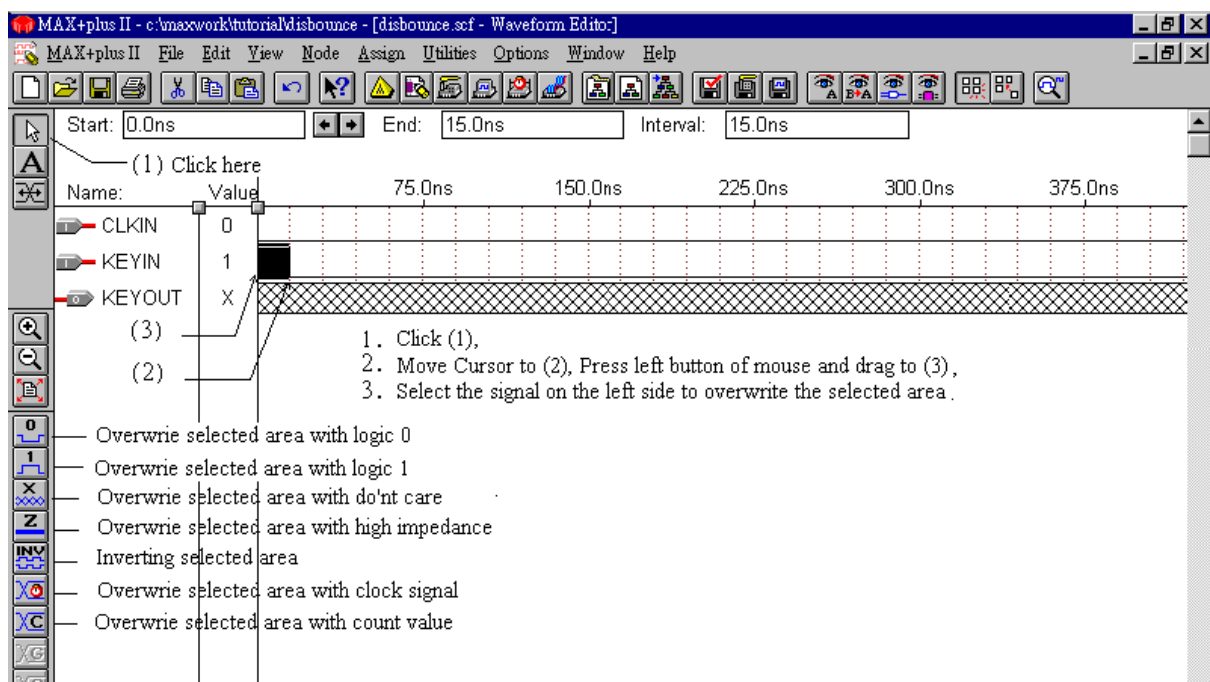


Figure 4.29a Select a waveform data block

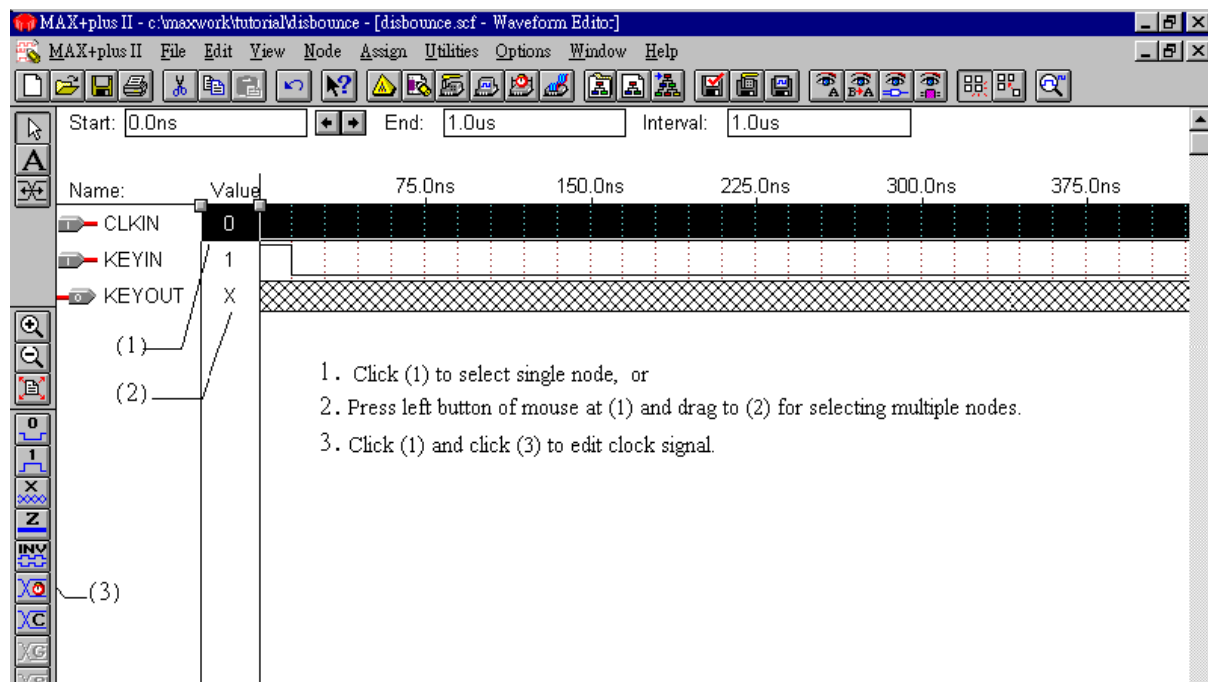


Figure 4.29b Select a node or several nodes data block

In Figure 4.29a, in order to overwrite the signal data, please click the left button of the mouse at “(1)” and drag to “(2)” to block the area. Then click the left button of the mouse at the left banner of the window to choose the overwritten signal type. This function can also apply to the data block in Figure 4.29b. Figure 4.29c and 4.29d illustrate the fast generation of clock waves. Another waveform editor is waveform edit cursor. Figure 4.30a and 4.30b demonstrate the situation of edit KEYIN signal. Please form the KEYIN waveform and save it by “File > Save As...”.

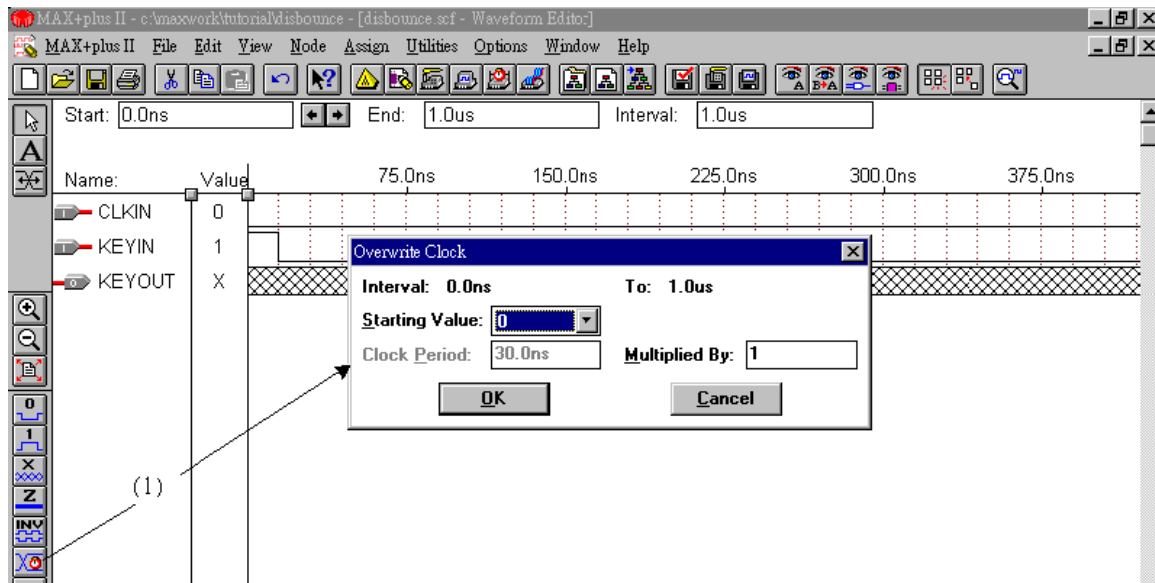


Figure 4.29c Clock waveform generation

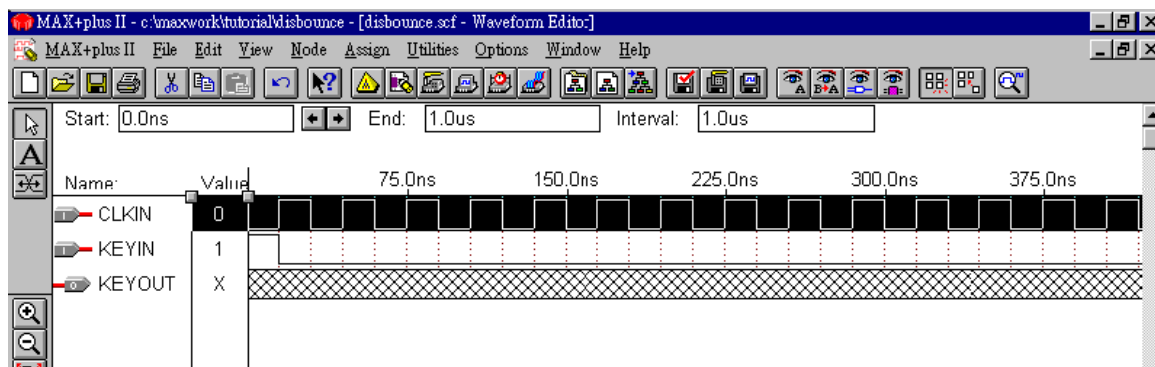


Figure 4.29d Clock waveform generation(continue)

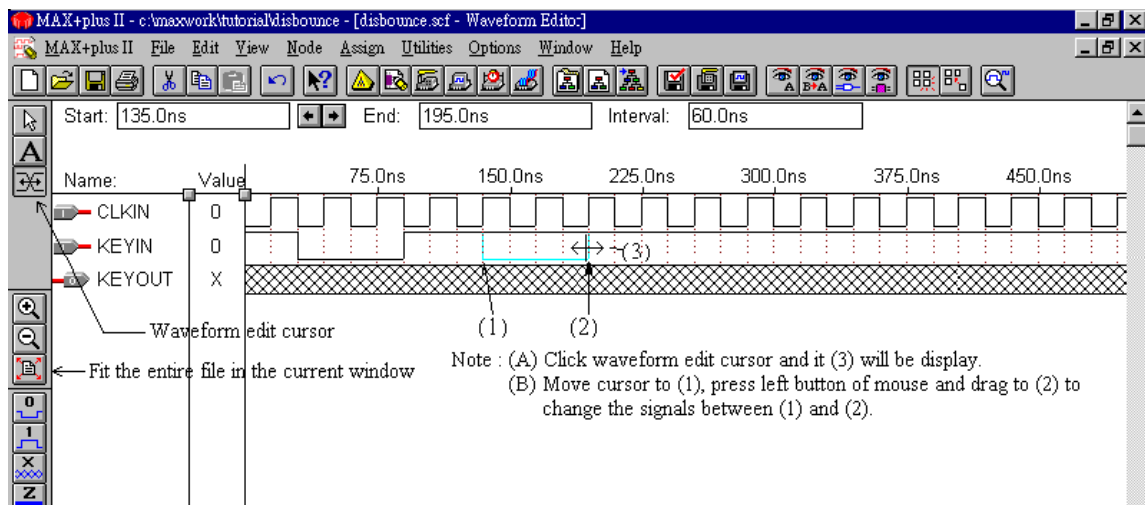


Figure 4.30a Edit KEYIN wave by waveform cursor

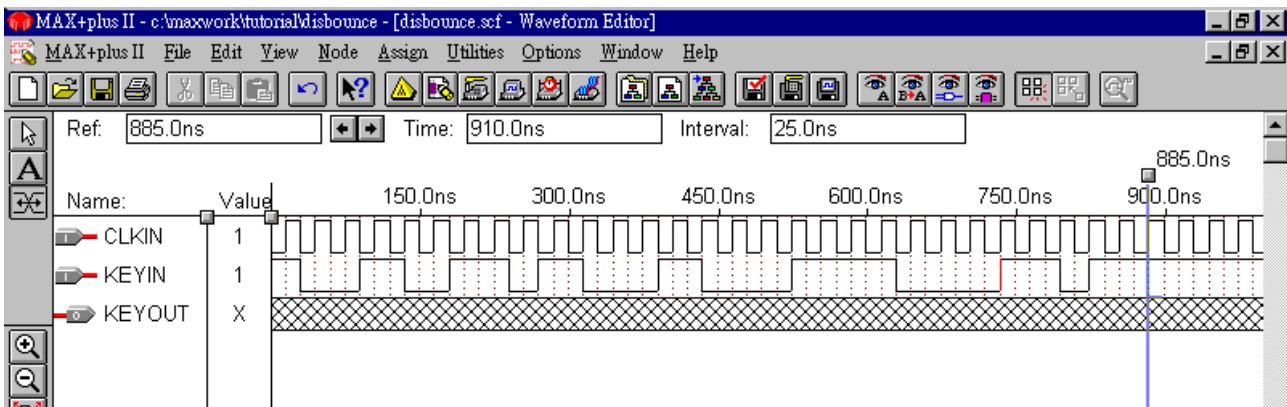


Figure 4.30b Edit KEYIN wave by waveform cursor (continue)

4. Activate functional simulator to implement functional simulation

After completing the definition of simulation waveform, we can activate the simulator by MAX+PLUS II > Simulator, as shown in Figure 4.31. At this time, click Start button to activate ALTERA functional simulation. Please note the file name of simulation input.

5. Manage the errors and inspect simulation results

If there are errors of the simulation results, please read the error signals, and modify the errors. If there are no errors, click the waveform window to inspect the output waveform of simulation result, as shown in Figure 4.33.

If the simulation result does not meet the functions, please choose the file and go back to Section 4.3 to modify and compile the circuit. Then, implement the functional simulation instructed in this section until the result meets the functions. In Figure 4.33, the disbounce signal has been eliminated.

6. Close file: Please close current file by File > Close.

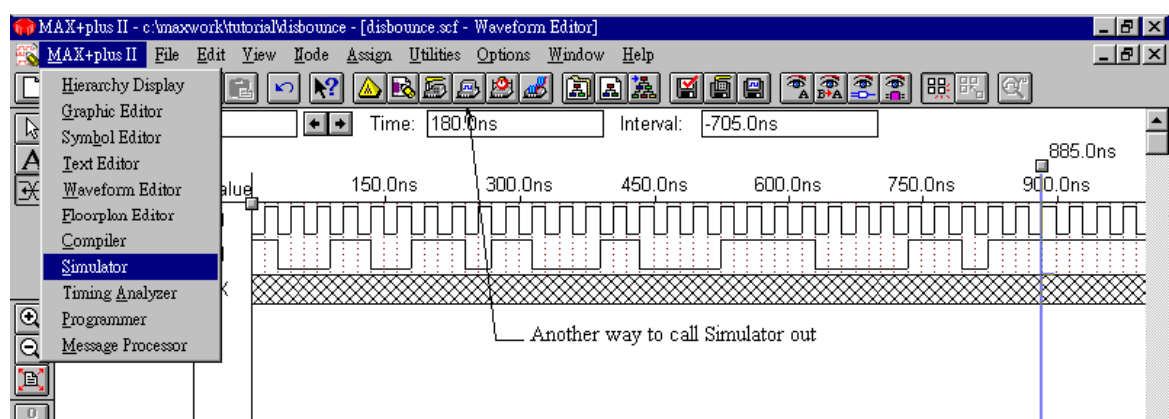


Figure 4.31 Activate functional simulation of MAX+PLUS II

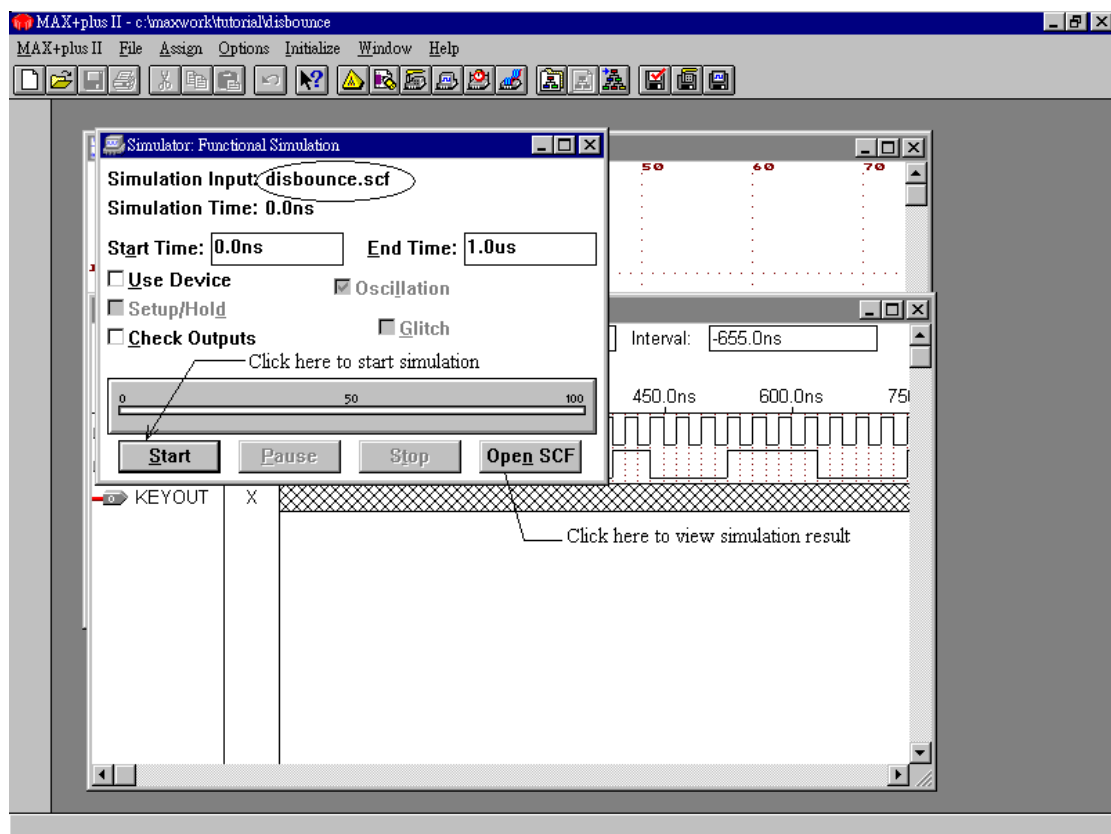


Figure 4.32 Functional simulation window

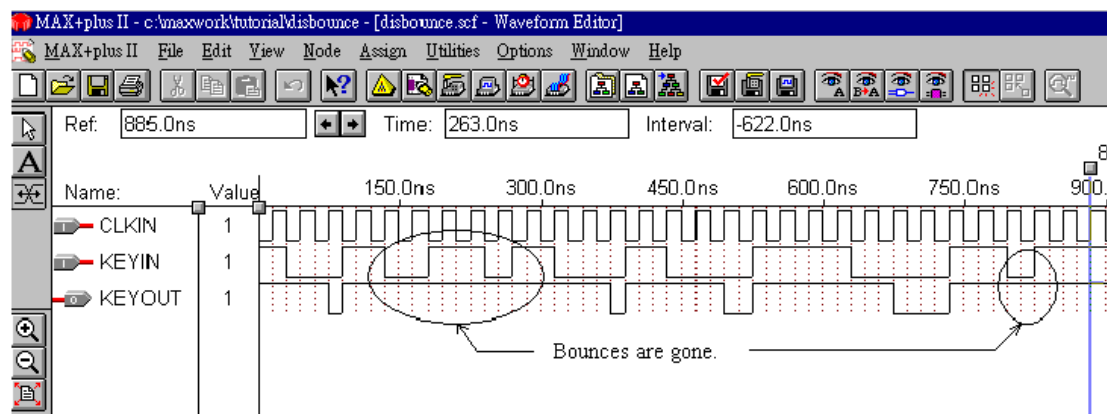


Figure 4.33 Functional simulation result

4.5 Floorplan and Design Compilation

Floorplan is the arrangement of circuit into FPGA chip, which includes chip assignment, the arrangement of input and output pins, and the arrangement of LAB (Logic Array Block) of the circuits...etc. However, not until implementing functional simulation of the circuits can these functions work. For a beginner, the priority is to familiarize with chip assignment, the arrangement of input and output pins. The critical issue is that all the floorplans are managed by Floorplan Editor which can be done by MAX+PLUS II > Floorplan Editor. The procedures are as follows:

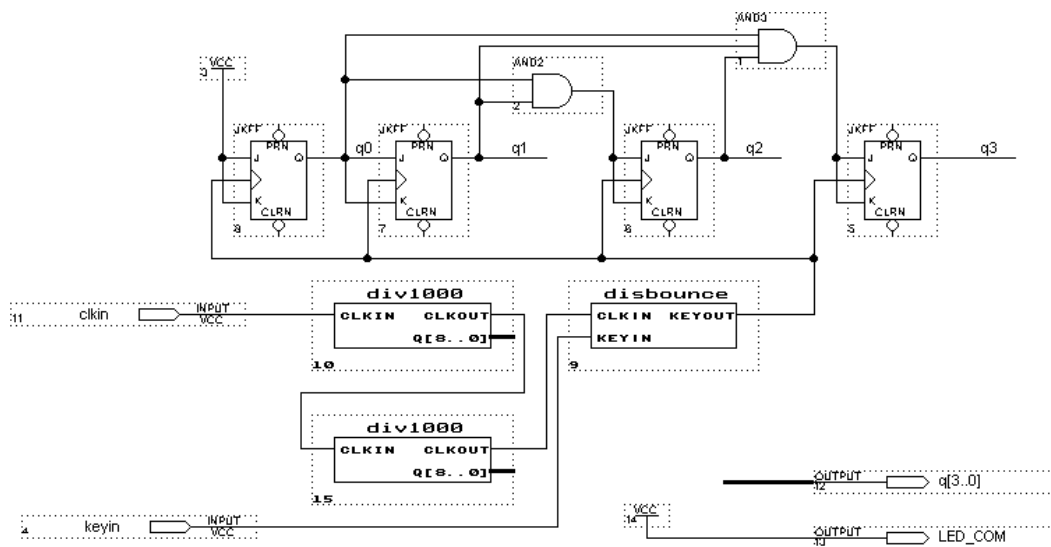


Figure 4.34a mod16.gdf

1. Setting up the processing of the compiler
2. Selecting FPGA chip
3. Floorplan
4. Design compilation after floorplan.

Let's take the file of mod16.gdf, Figure 4.34a and 4.34b, as an example to illustrate the whole procedure.

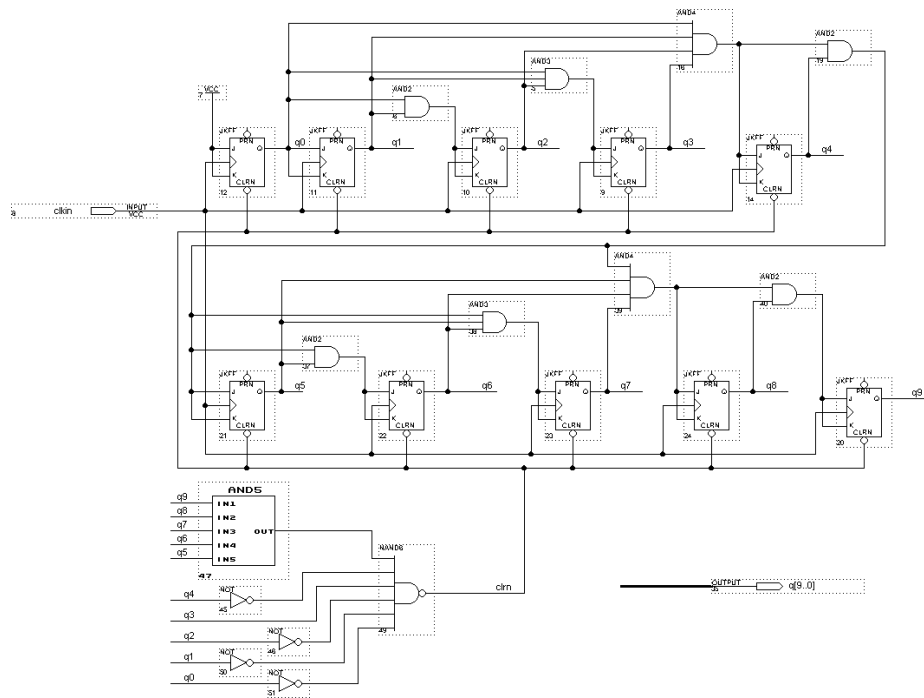


Figure 4.34b div1000.gdf sub-circuit in mod16.gdf

First of all, please edit the circuit of div1000.gdf, Figure 4.34b, by the graphic editor, and complete the functional simulation, Figure 4.34c and Figure 4.34d, by the functional simulator. At the mean time, generate an internal circuit symbol for using in mod16.gdf. Figure 4.34e is the relevant passive circuit. We can directly download the configuration data.

1. Setting up compiler process

If the window of Figure 4.35 shows up when you are using the compiler, please click Function SNF Extractor of “Process” to make it turns to Figure

4.35, which has more compilation functional modules. From left to right are Compiler Netlist Extractor, Database Builder, Logic Synthesizer, Partitioner, Fitter, Timing SNF Extractor and Assembler. From here, we can understand that the Processes of ALTERA compiler can be set to meet any kind of requirement. The Compiler's Processes will be described in detail in the following chapter.

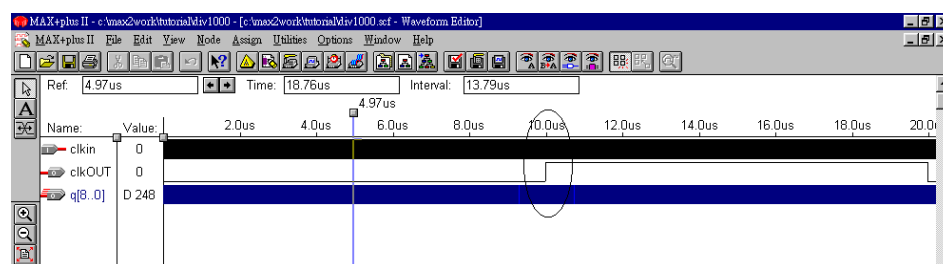


Figure 4.34c Simulation result of div1000.gdf sub-circuit

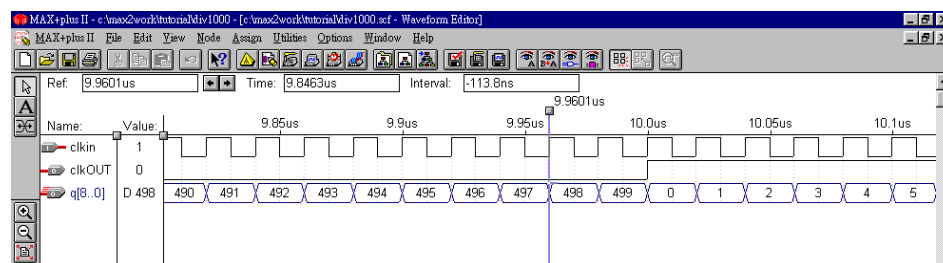


Figure 4.34d Simulation result of div1000.gdf sub-circuit
(Magnify the circle in Figure 4.34c)

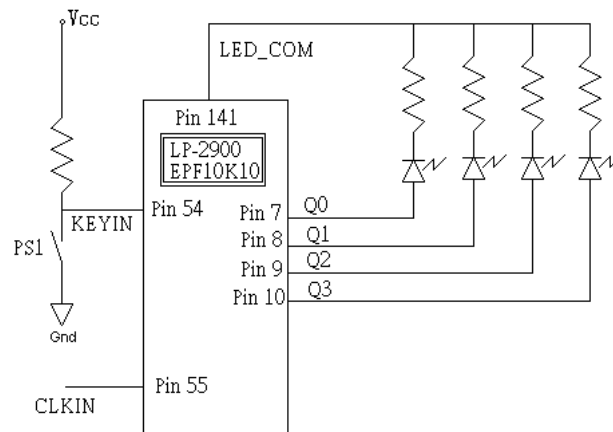


Figure 4.34e Peripheral passive circuit of mod16.gdf

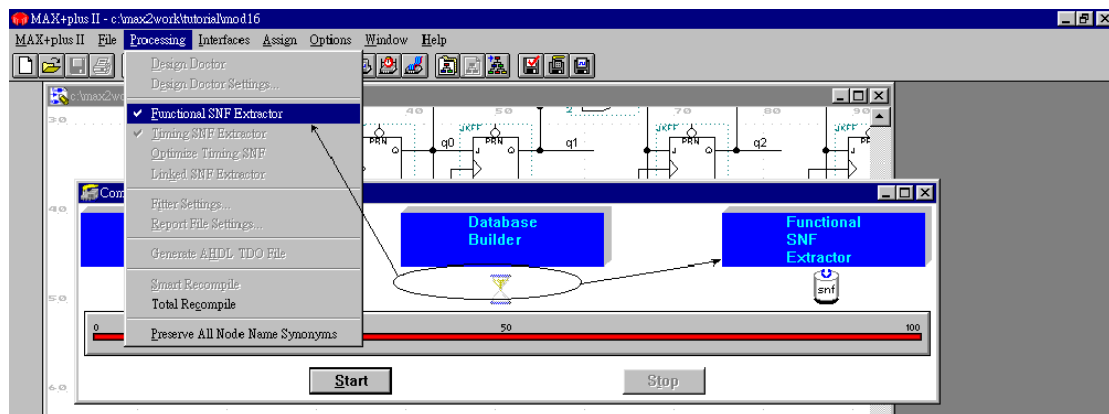


Figure 4.35 Simple compiler window of functional simulation

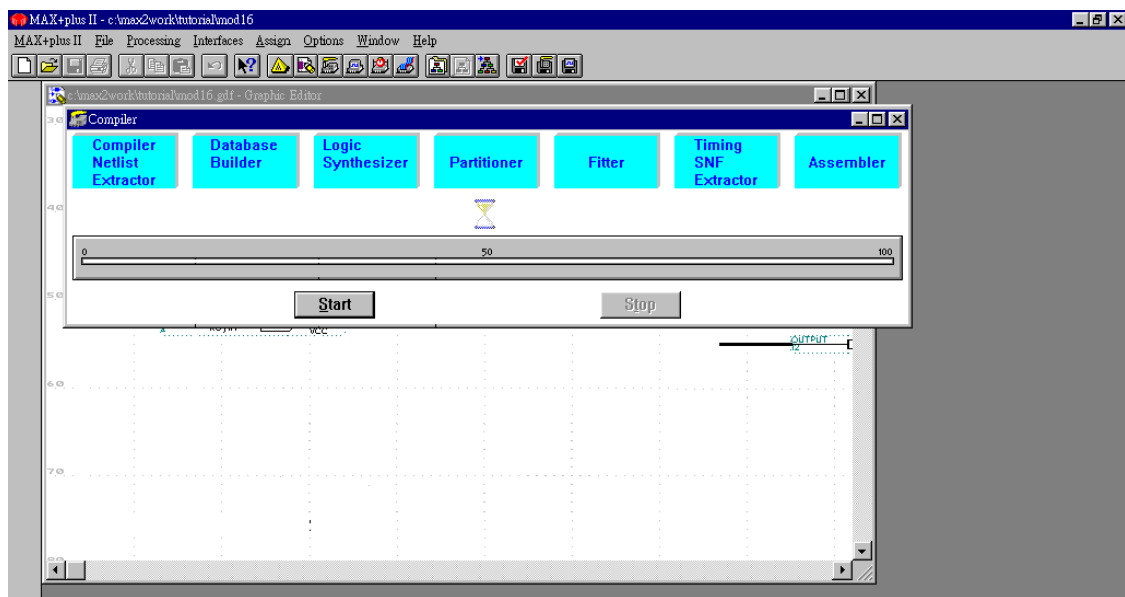


Figure 4.36 Compiler window

2. Selecting FPGA device

Please follow the instructions of Figure 4.37a and Figure 4.37b to choose EPF10K10TC144-4 of FLEX 10K family, which is a SRAM 144-pin chip.

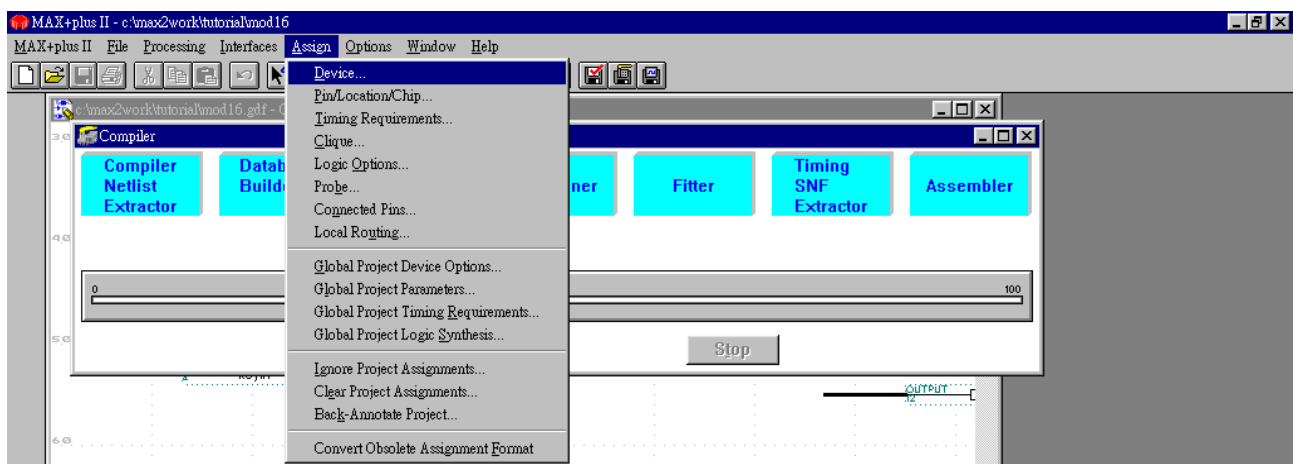


Figure 4.37a Assigning FPGA device

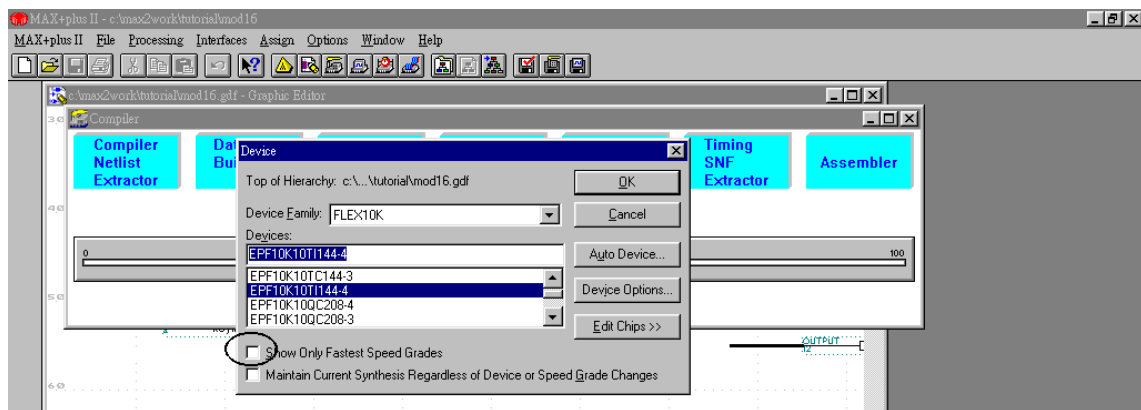


Figure 4.37b Assigning FPGA device

3. Floorplan

Figure 4.38 is the last compilation, the below half figure is the device overview. Another way to view the chip is through a LAR view, which allows checking the arrangement in the selected chip and the arrangement of the pins. It would spontaneously offer the relevant information as long as the cursor moves to the circuit block that you intend to check. On the other hand, the device view can only offer the information of pin arrangement. Both of these views can switch the functions of each view as long as double click the left bottom of the mouse at the location of the cursor, as shown in Figure 4.38

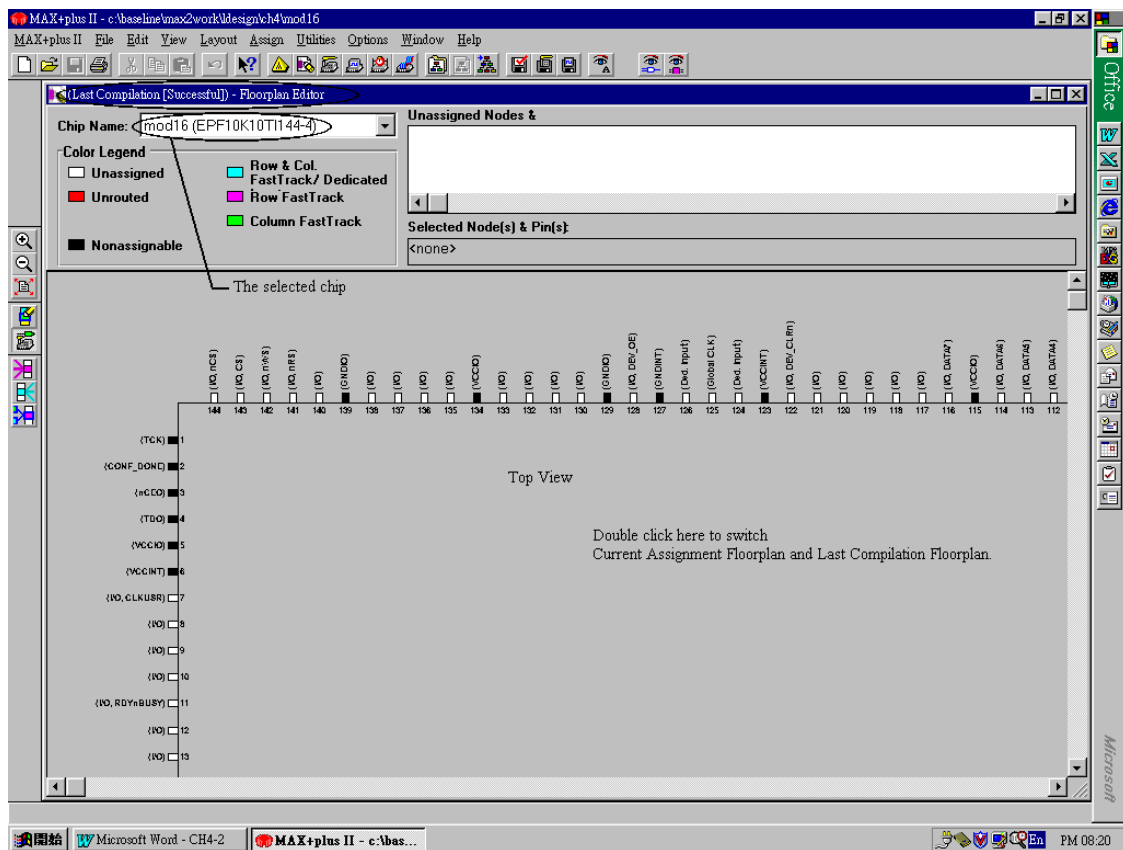


Figure 4.38 Device view of the last compilation

If you are not really satisfied with your pin arrangement or even would like to make any changes, you could re-plan the pin assignment by Layout > Current Assignments as Figure 4.39. Once the window is there, you could use the left bottom of your mouse, select the pin from the block at the upper-right corner, and drag it to the desired pin in the lower part of the window. Release the button to reassign I/O pin. We also could select any pin from lower of the window and drag it to another desired pin and release it to change pin assignment. As Figure 4.34b, please move CLKIN to Pin 55, KEYIN to the Pin 47, Q1 to Pin 7, Q2 to Pin 8, Q3 to Pin 9, Q4 to Pin 10, and LED_COM output to Pin 141, which is Common anode. The EPF10K10TC144-4 pin assignments are listed as Table 4.1.



Table 4.1 EPF10K10TC144-4 pin assignments

Name of Signal Line	Pins of EPF10K10TC144-4
CLKIN	Pin 55
KEYIN	Pin 47
Q1	Pin 7
Q2	Pin 8
Q3	Pin 9
Q4	Pin 10
LED_COM	Pin 141

4. Design Compilation After Floorplanning

By floorplanning, please open Compiler window as Figure 4.36 after complete the latest pin assignment. Select “Processing > Total Recompile” to start Compiler. If there is no error message displayed, you will get a file named “mod16.sof”, which we will use in the next section to program EPF10K10TC144-4 chip. However, if an error message is shown out, please back to the step 3 and modify the pin assignment as well as the compilation until all correct.

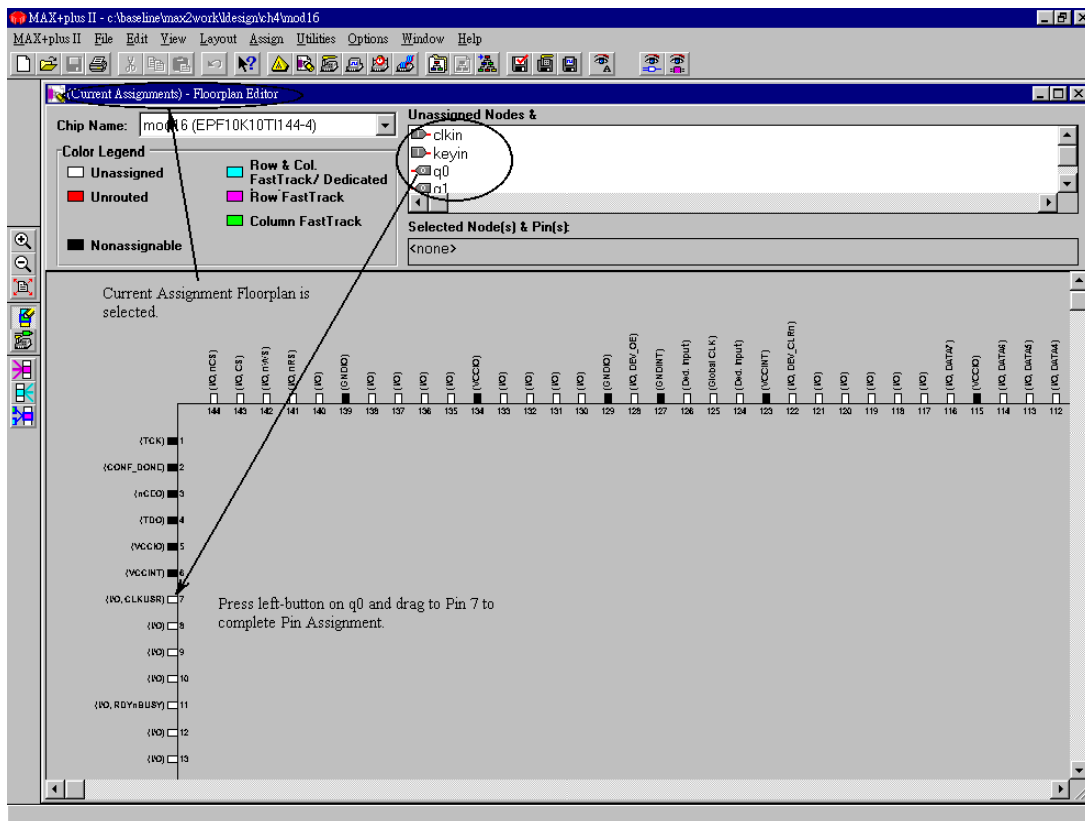


Figure 4.39 Compiler —current assignments

4.6 Device Programming and Circuit Testing

For ALTERA device programming, it requires users check what type of the reconfigurable element is used in the selected device. The type of the reconfigurable element could be EPROM, EEPROM, FLASH, or SRAM. Different types will use different programming approaches. Since we already talked about EPF10K-10TC144-4 CPLD in the last section, we will use it as our programming example. EPF10K-10TC144-4 CPLD is an ALTERA SRAM device. All of the configuration data has to be installed into the circuit completely after the system powers on. In another words, FLEX10K family offers users a great flexibility to reconfigure different circuits with different re-configuration data. To complete the



reconfiguration when the system powers on, ALTERA offers ALTERA's serial configuration EPROM which can save configuration data for FLEX10K family and install configuration data to complete circuit configuration when the system is starting process. In Table 4.2a, we list the types and specifications of the typical examples applying ALTERA's serial configuration EPROM.

Table 4.2a Typical application examples of ALTERA's serial configuration EPROM

FLEX10K Device	Serial Configuration EPROM
EPF10K10, EPF10K10A	EPC1 or EPC1441
EPF10K20	EPC1 or EPC1441
EPF10K30, EPF10K30A, EPF10K30B	EPC1 or EPC1441
EPF10K40	EPC1
EPF10K50, EPF10K50V, EPF10K50B	EPC1
EPF10K70	EPC1
EPF10K100, EPF10K100A, EPF10K100B	EPC1 $\times 2$
EPF10K130V, EPF10K130B	EPC1 $\times 2$
EPF10K180B	EPC1 $\times 3$
EPF10K250A, EPF10K250B	EPC1 $\times 4$
EPF8282A	EPFC1, EPC1441, EPC1213 or EPC1064
EPF8282AV	EPFC1, EPC1441 or EPC1064V
EPF8452A	EPFC1, EPC1441, EPC1213 or EPC1064
EPF8636A	EPFC1, EPC1441 or EPC1213
EPF8820A	EPFC1, EPC1441 or EPC1213
EPF81188A	EPFC1, EPC1441 or EPC1213
EPF81500A	EPFC1, EPC1441 or EPC1213 $\times 2$
EPF6016, EPF6016A	EPFC1 or EPC1441
EPF6024A	EPFC1 or EPC1441



Table 4.2b Types and characteristics of ALTERA's serial configuration EPROM

Serial Configuration	Description
EPFC1	1,046,496 bits, Voltage: 5.0V or 3.3V
EPC1441	440,800 bits, Voltage: 5.0V or 3.3V
EPC1213	212,942 bits, Voltage: 5.0V
EPC1064	65,536 bits, Voltage: 5.0V
EPC1064V	65,536 bits, Voltage: 3.3V

To make it easy to use ALTERA MAX+PLUS II and help new users adopt FLEX10K devices, Leap Company gives a great LP-2900 CPLD logic design experimental platform, which has two major hardware components as below:

❖ CPLD Device Board

This module has the circuits including:

1. Programming circuit: The circuit made by a target chip EPF10K10TC144. Its function is to receive the configuration data from programmer or EPROM, to program EPF10K10TC144, and to drive the outside circuit.
2. Download circuit: Its function is to send the “configuration data” to a “programming circuit” to easily program EPF10K10TC144. Totally there are many various programming technologies, and here we only introduce three as below:
 - (1) It can be done via PC printer parallel port to receive “configuration data”, and forward to a “programming circuit”. Definitely, it requires users to run MAX+PLUS II Programmer by PC to send “configuration data”.
 - (2) It can be done by EPROM, which has an EPROM 2764 socket on the board, to receive “configuration data” and forward the data to a “programming circuit”.



- (3) Without “download circuits”, it can also be done by ByteBlaster connection bus to download the configuration data. ByteBlaster connection bus can connect between PC printer parallel ports and ByteBlaster plugs on the CPLD-EPF10K10 device board. It also requires MAX+PLUS II Programmer on PC to send “configuration data”.

❖ I/O Element Experimental Platform

The module has some input buttons or selection switches and some output display circuits such as LED, Dot Matrix, and LCD. The experimental platform also has A/D converter, D/A converter, and an 8051 microprocessor. For the details of the information, please refer to the explanation in Chapter 9.

By Leap Company’s LP-2900 CPLD logic design experimental platform, new users can use the following technologies to program FLEX10K10 devices:

1. Download via printer parallel port:
 - (1) Use connection bus to connect between PC printer parallel port and LP-2900 experimental platforms.
 - (2) After power on the experimental platform, LED D1 at the upper-left corner is lightened up, and then press RESET button.
 - (3) In MAX+PLUS II, start programmer window by selecting MAX+PLUS II > Programmer as Figure 4.40. Because we haven’t finished the “programming hardware setting”, “Programming hardware is not installed.” will be shown up, and please click “Yes”.
 - (4) Double click for Hardware Setup window by selecting Option > Hardware Setup as Figure 4.41. After starting Hardware Setup window, please find “Hardware Type” and drag down to select “ByteBlaster” as

Figure 4.42a and Figure 4.42b. Once complete the hardware setting, it is unnecessary to redo the setting process except we make some changes to the downloading technologies.

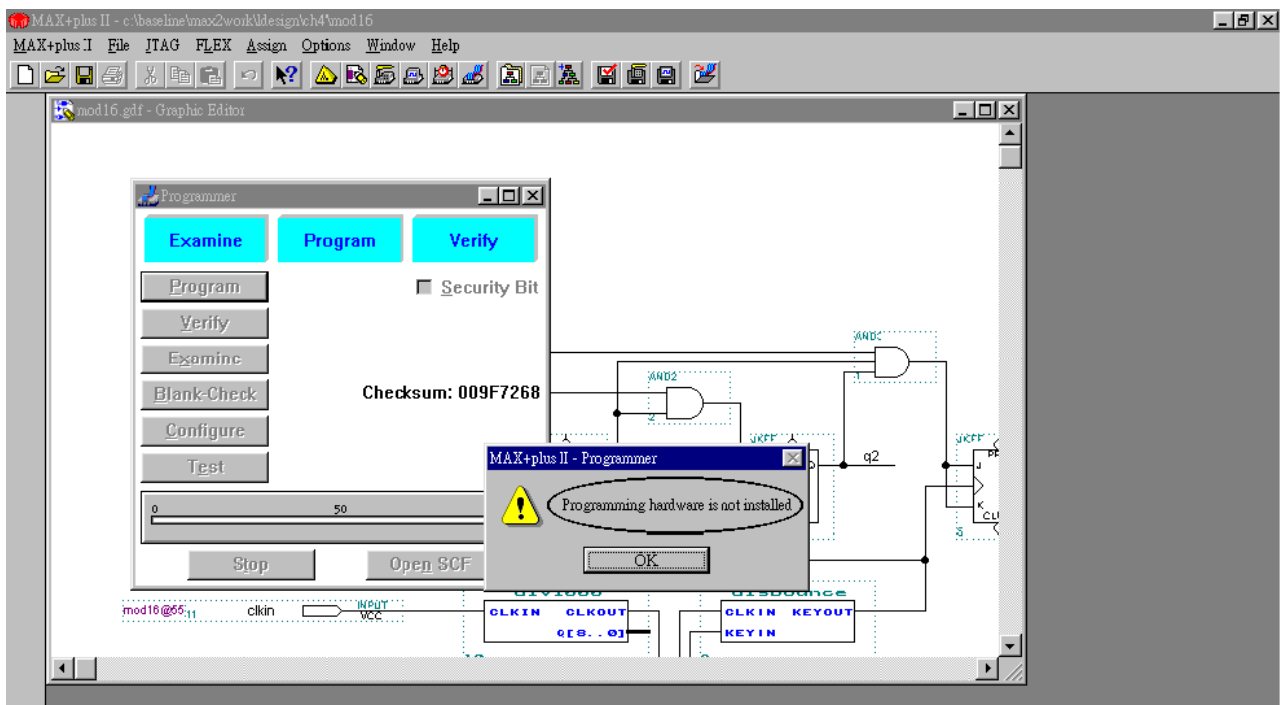


Figure 4.40 Starting Programmer window at first time by MAX+PLUS II

> Programmer

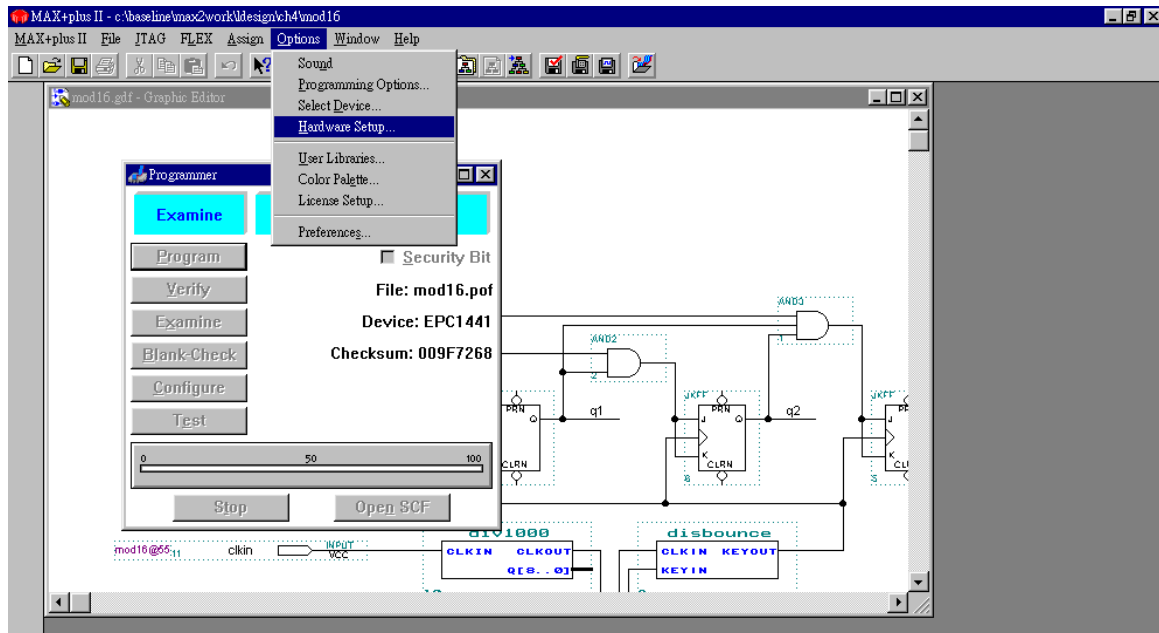


Figure 4.41 Starting Hardware Setup window

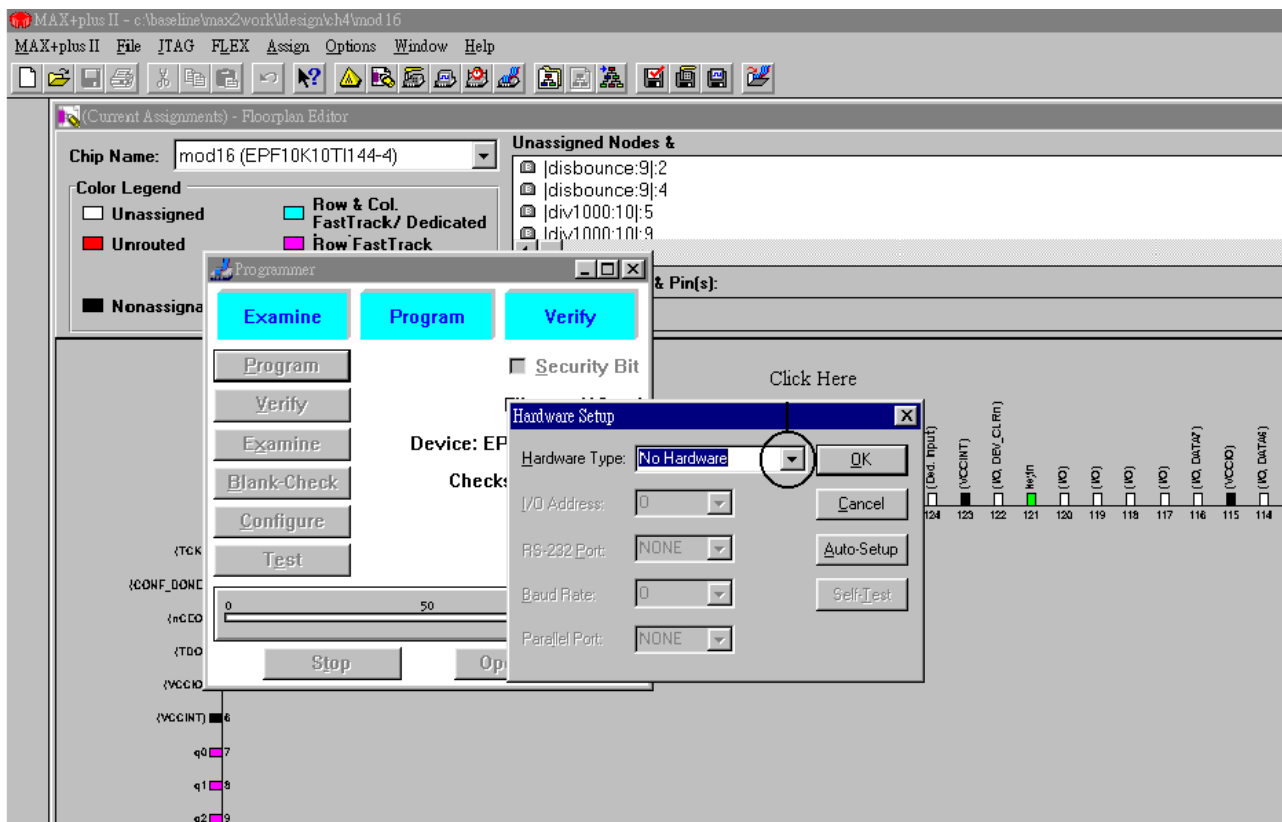


Figure 4.42a Hardware Setup window

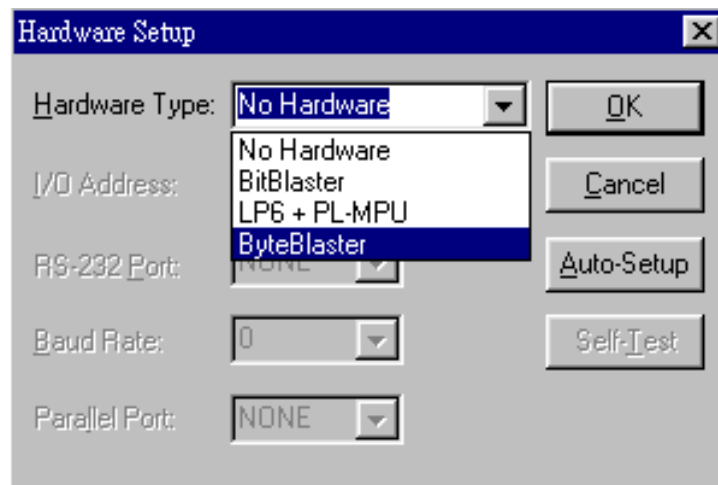


Figure 4.42b Hardware Setup window (continue)

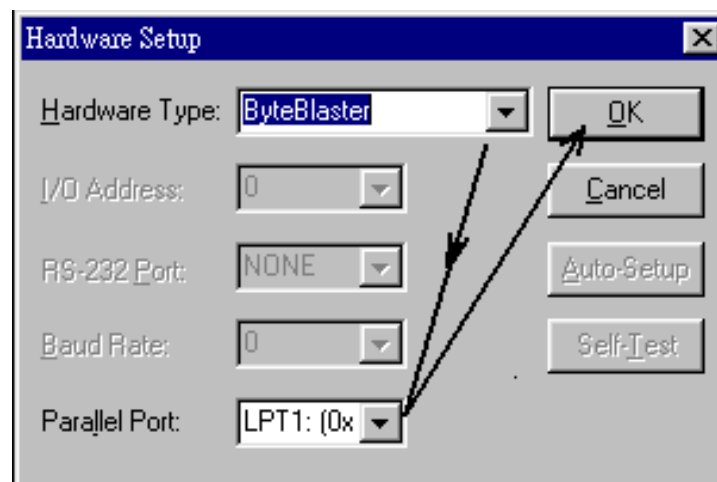


Figure 4.42c Hardware Setup (continue)

- (5) Start Multi-device JTAG Chain Setup by selecting JTAG > Multi-Device JTAG Chain Setup as Figure 4.43. In Multi-device JTAG Chain Setup window, please drop down the “Device Name” column and select “EPF10K10” as Figure 4.44a. Then click “Select Programming File”, choose the “mod16.sof” file as Figure 4.44b, and then click “Add” and



“Ok” as Figure 4.44c to close Multi-device JTAG Chain Setup window. If the message as Figure 4.44d is found, Multi-device JTAG Chain Setup mode is then closed completely, and so please click “Yes”.

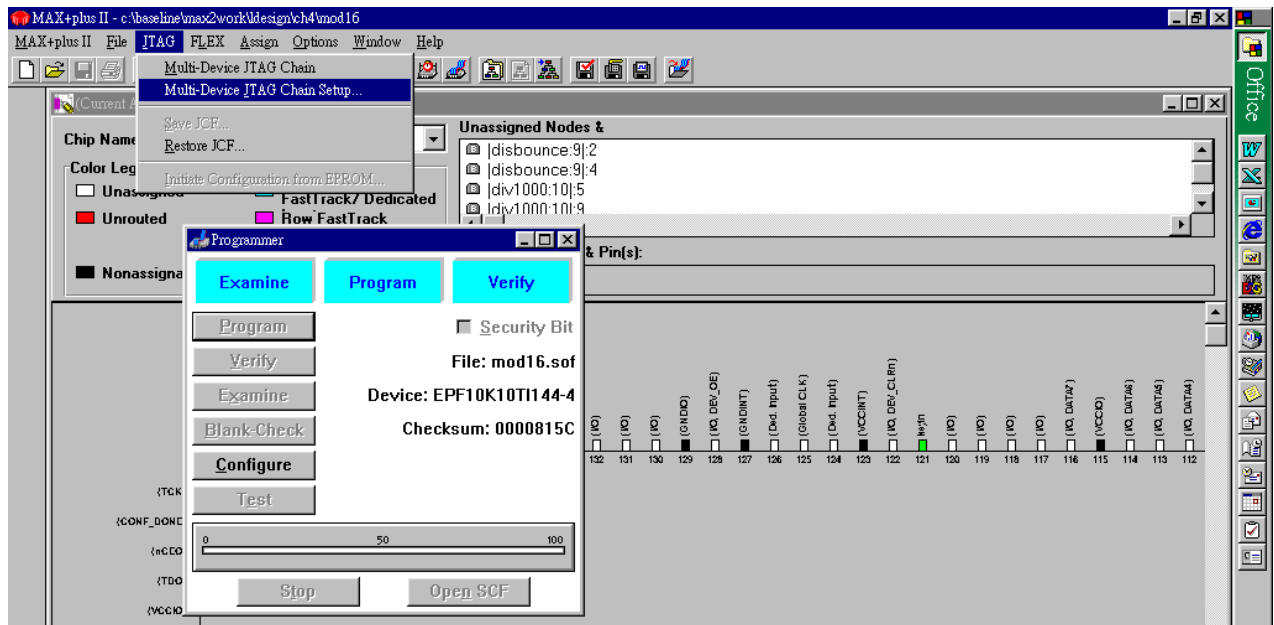


Figure 4.43 Starting Multi-device JTAG Chain Setup window

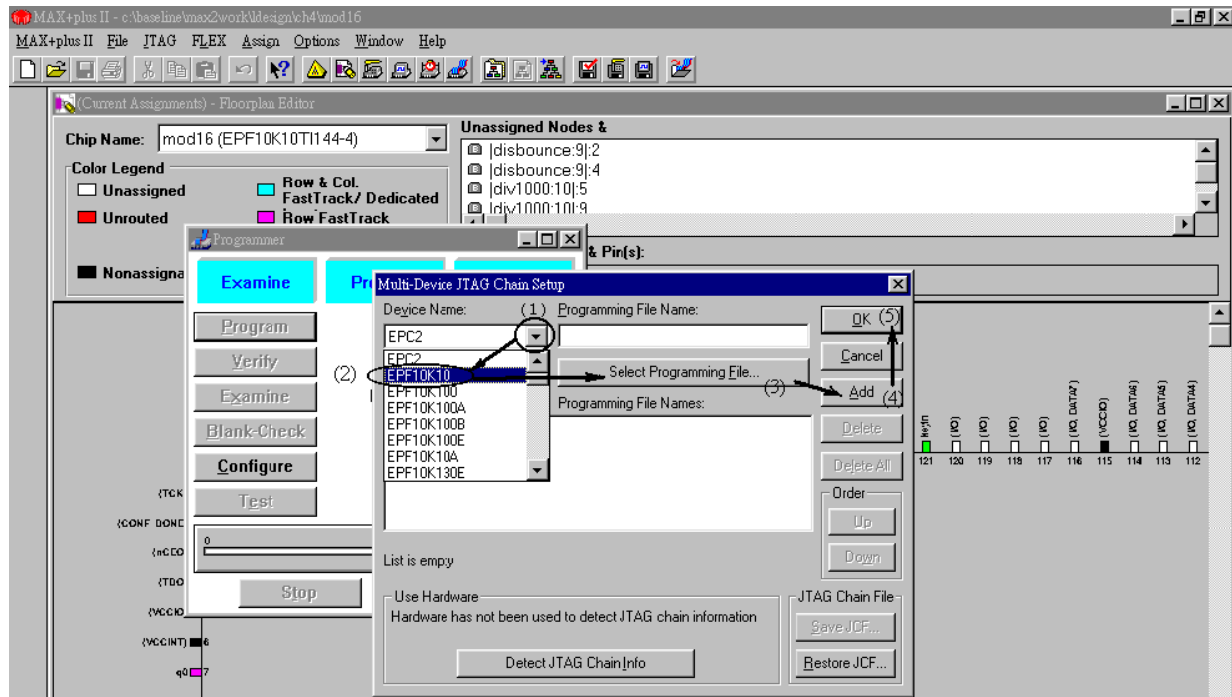


Figure 4.44a Multi-device JTAG Chain Setup

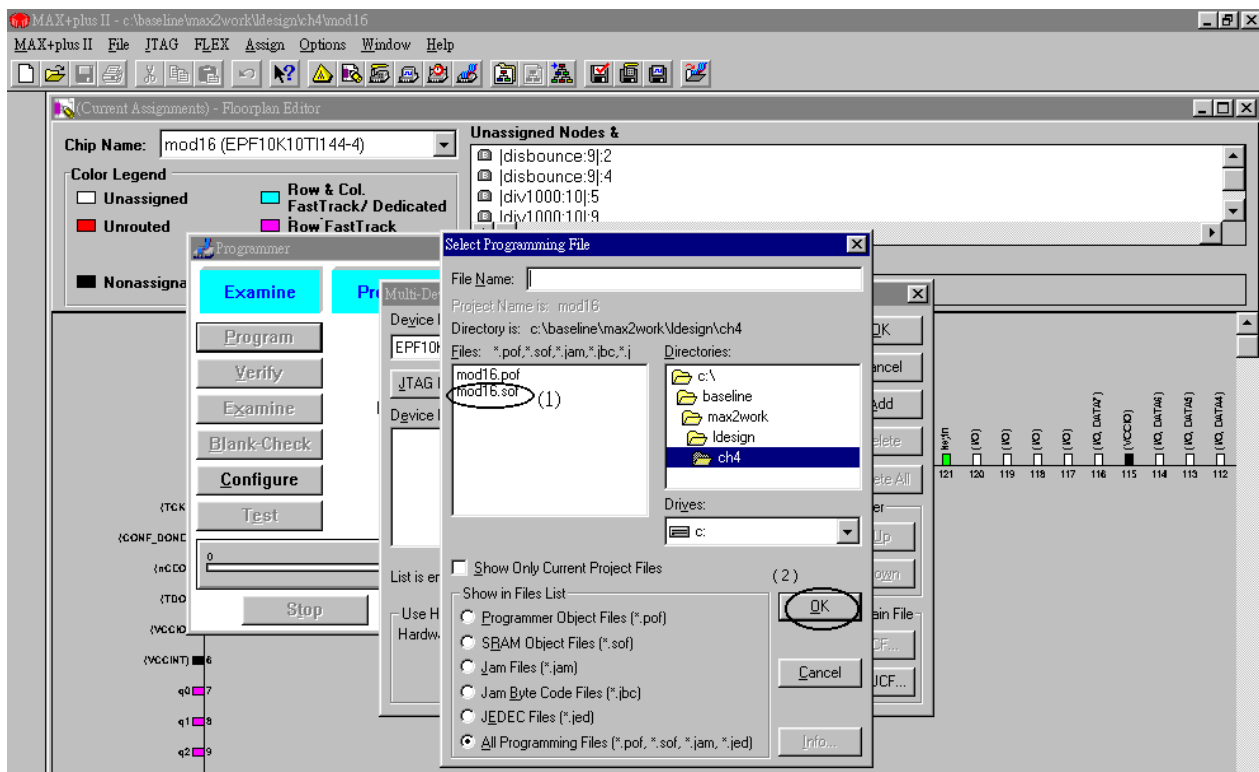


Figure 4.44b Multi-device JTAG Chain Setup (continue)

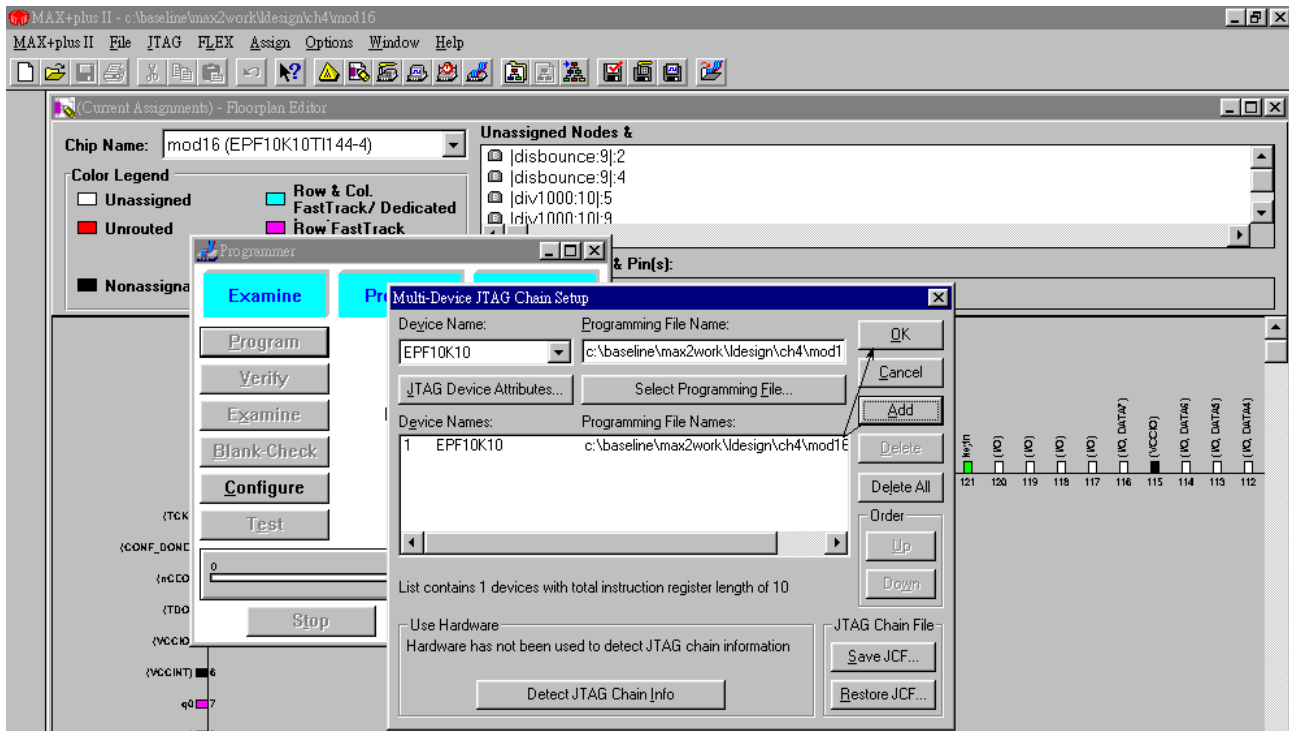


Figure 4.44c Multi-device JTAG Chain Setup (continue)

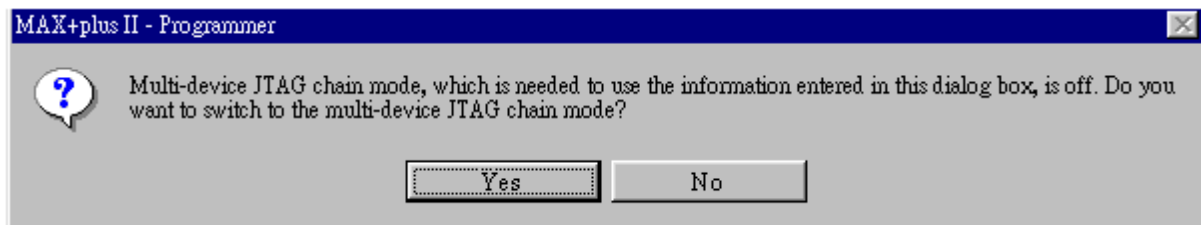


Figure 4.44d Multi-device JTAG Chain Setup (continue)

(6) Back to Programmer window, click “Configure” as Figure 4.45 to download circuit.

Once the programming is completed, the OK light on the CPLD device board will be on; otherwise ERROR light will be. Click PS1 at lower-bottom corner in LP-2900 CPLD logic circuit experimental platform as Figure 4.46. Meanwhile, check L1 to L4 LED at the upper-right corner on



the experimental platform to know if it has changes from binary 0, 1, 2, 3, ..., 15 to 0, 1, 2, 3,

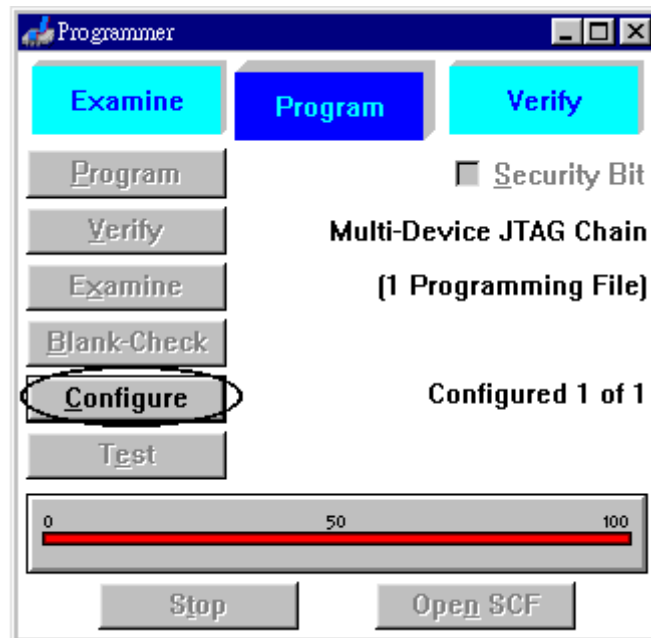


Figure 4.45 Download circuit to LP-2900

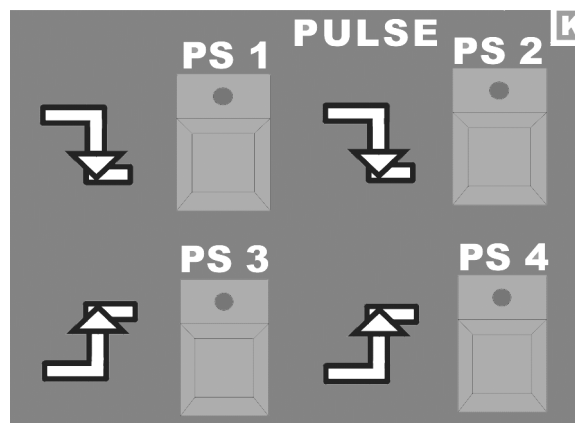


Figure 4.46 PS1 location in LP-2900

Congratulation! First Time is Always Unforgettable...

2. Download by parallel EPROM

After the design file is totally compiled, please select File > Convert SRAM Object Files as Figure 4.47 to convert programming files into Single-Device.HEX format. In another words, in Convert SRAM Object Files window, click “Add” in Input File column as Figure 4.48, and then click “Ok” after select “.hex (Single-Device)” in Output File column. The data will be programmed into EPROM (in this case, it is 2764.) and then insert it onto LP-2900 experimental platform.

At that point, it is not necessary to connect between the experimental platform and a PC printer parallel port. It only requires to power on the experimental platform, and LED D1 will be on. Click “Reset” button. The OK light on the successfully downloaded CPLD-EPF10K10 will be on; otherwise, ERROR light will be on and we have to repeat the steps 1 and 2.

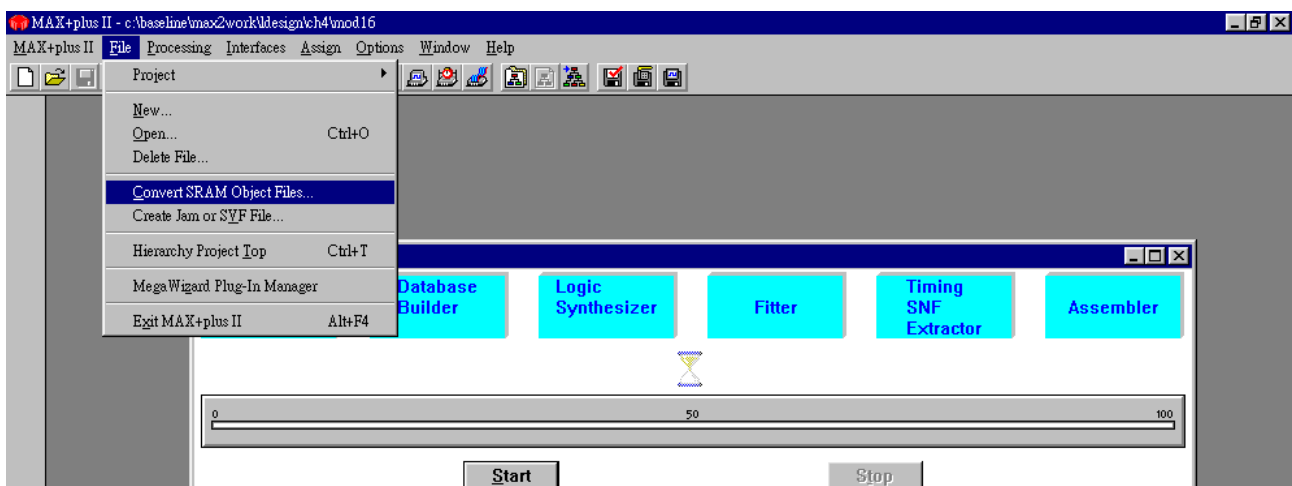


Figure 4.47 Starting Convert SRAM Object Files window by using

File > Convert SRAM Object Files

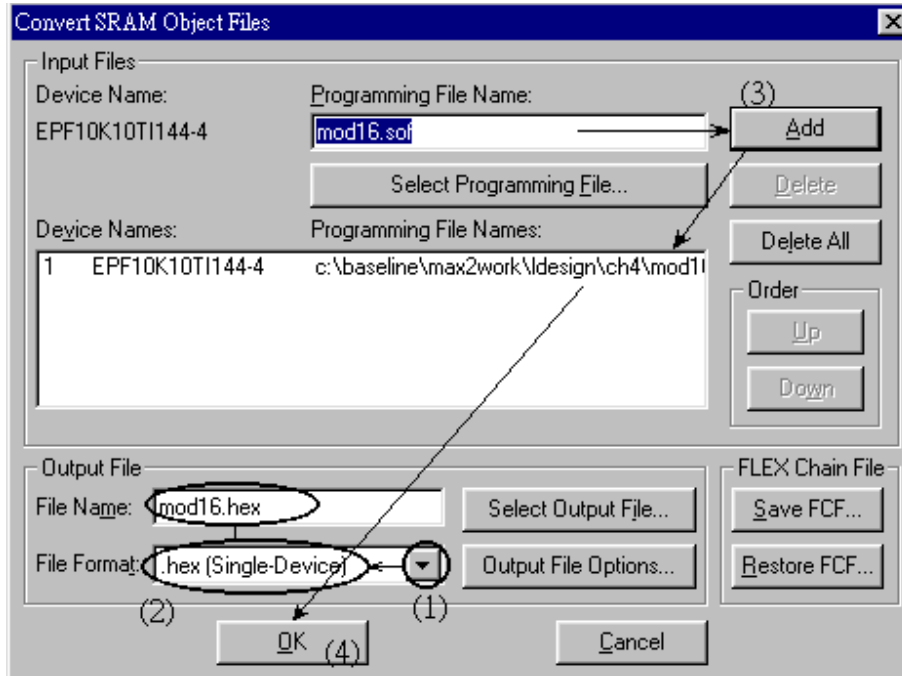


Figure 4.48 Converting programming file into Single-Device format

4.7 Use Graphic Entry to Complete LEDTEST Example

In this section, we will give you an example to review (1) graphic-entry circuit design; (2) functional simulation; (3) floorplan and design compilation; (4) the practice of device programming and circuit verification via LP-2900 CPLD logic circuit experimental platform. Because the clock used in LP-2900 CPLD logic circuit experimental platform is 10 MHz, we need one or two Frequency Division circuits and 12-bit Ripple Counters. First, use graphic-entry circuit design to complete the three sub-circuit designs, LEDTEST.GDF design, and some required simulations.



To start the practice, please complete the following direction and refer to previous sections when needed.

- Step 1: As Figure 4.49a, please make graphic entry for the sub-circuit DIV10.GDF. Create a default symbol as Figure 4.49b. For the related operational process, please see Section 4.3.
- Step 2: Please complete the functional simulation of the sub-circuit DIV10.GDF as Figure 4.49c. For the related operational process, please see Section 4.4.
- Step 3: Please use the two elements of div1000 and div10, enter the sub-circuit CLKGEN.GDF as Figure 4.50a, and create a default symbol as Figure 4.50b. For the related operational information, please see Section 4.3.

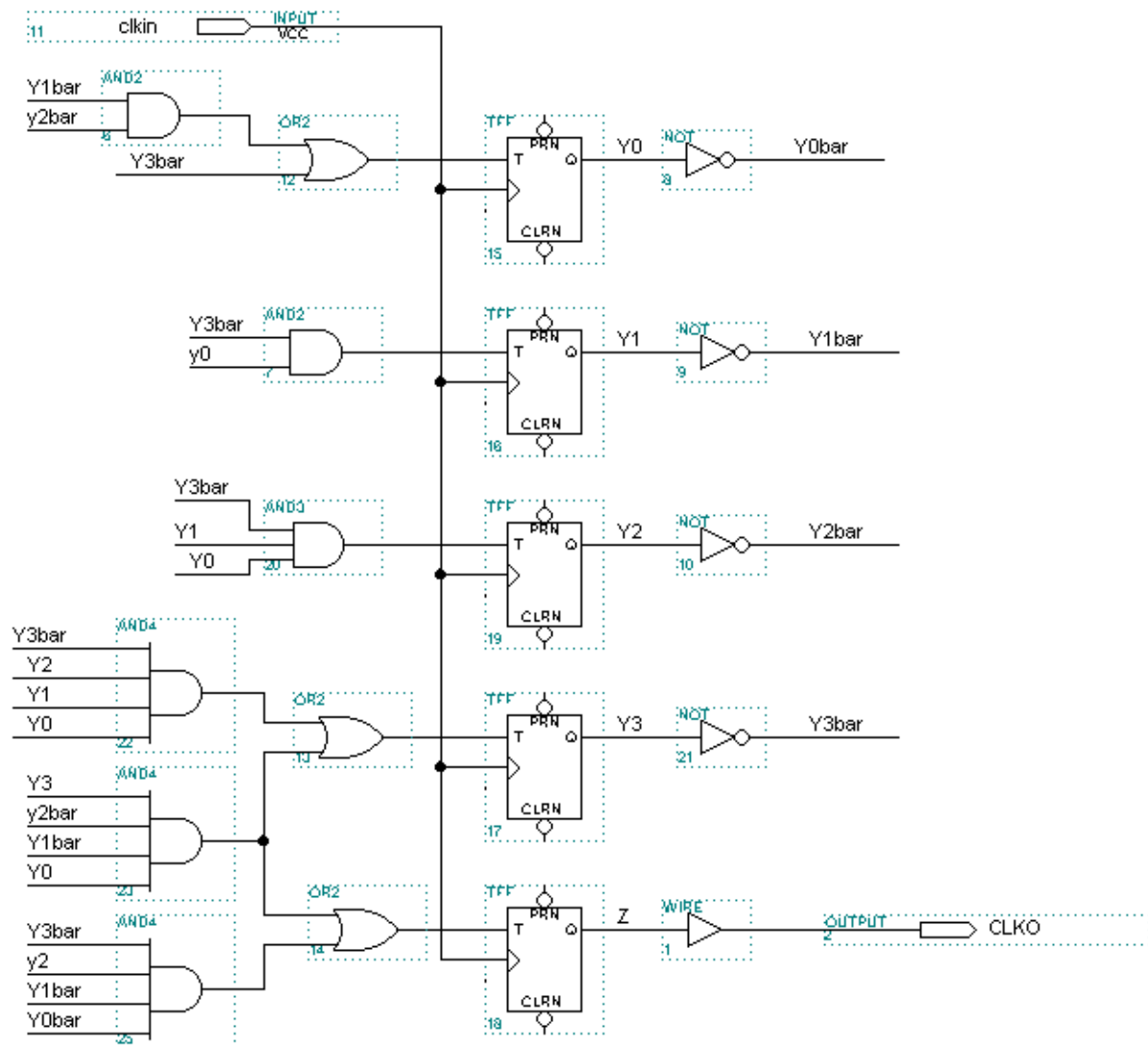


Figure 4.49a Sub-circuit DIV10.GDF (File: DIV10.GDF)

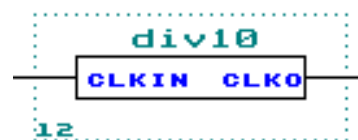


Figure 4.49b Default symbol of sub-circuit DIV10.GDF (File: DIV10.SYM)

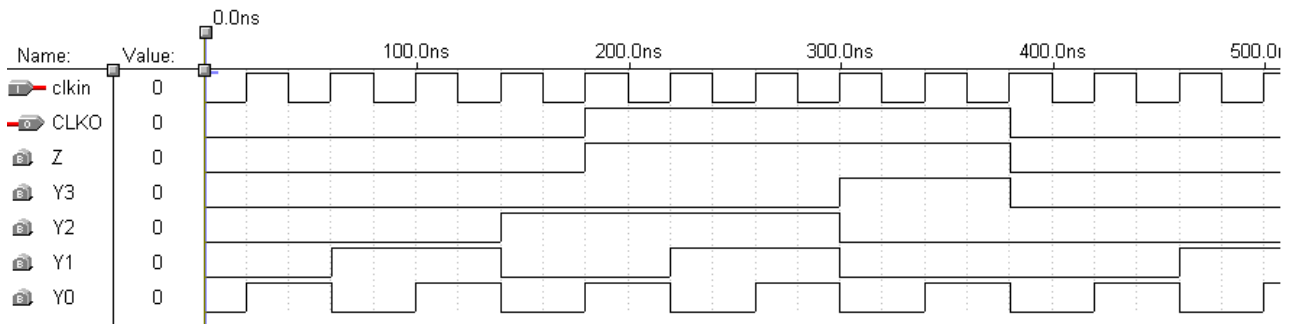


Figure 4.49c Functional simulation results of sub-circuit DIV10.GDF (File: DIV10.SCF)

Step 4: Please enter the sub-circuit RING12.GDF as Figure 4.51a, and create a default symbol as Figure 4.51b. For the related operation information, please see Section 4.3.

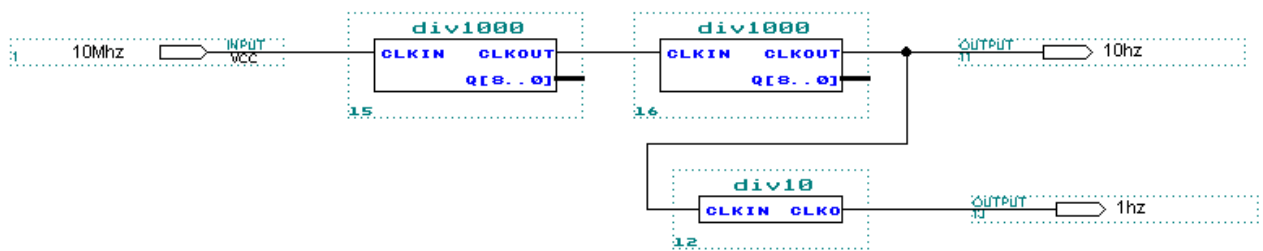


Figure 4.50a Sub-circuit CLKGEN.GDF (File: CLKGEN.GDF)

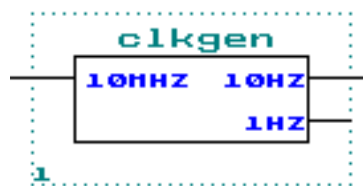


Figure 4.50b Default symbol of sub-circuit CLKGEN.GDF (File: CLKGEN.SYM)

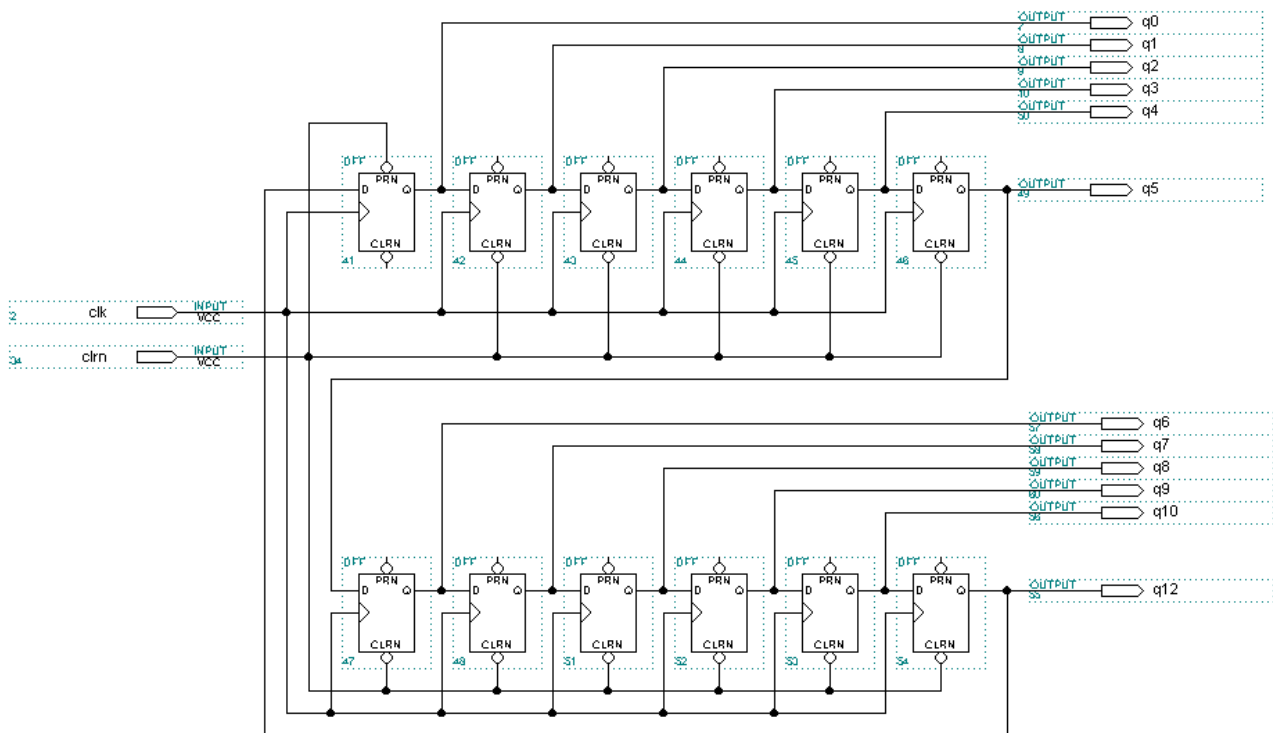


Figure 4.51a Sub-circuit RING12.GDF (File: RING12.GDF)

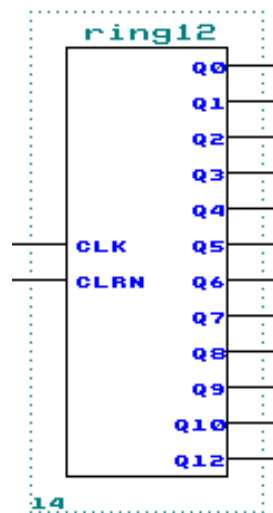


Figure 4.51b Default symbol of RING12.GDF sub-circuit (File: RING12.SYM)

Step 5: Please complete the functional simulation of the sub-circuit RING12.GDF

as Figure 4.51c. For the related operational information, please see Section 4.4.

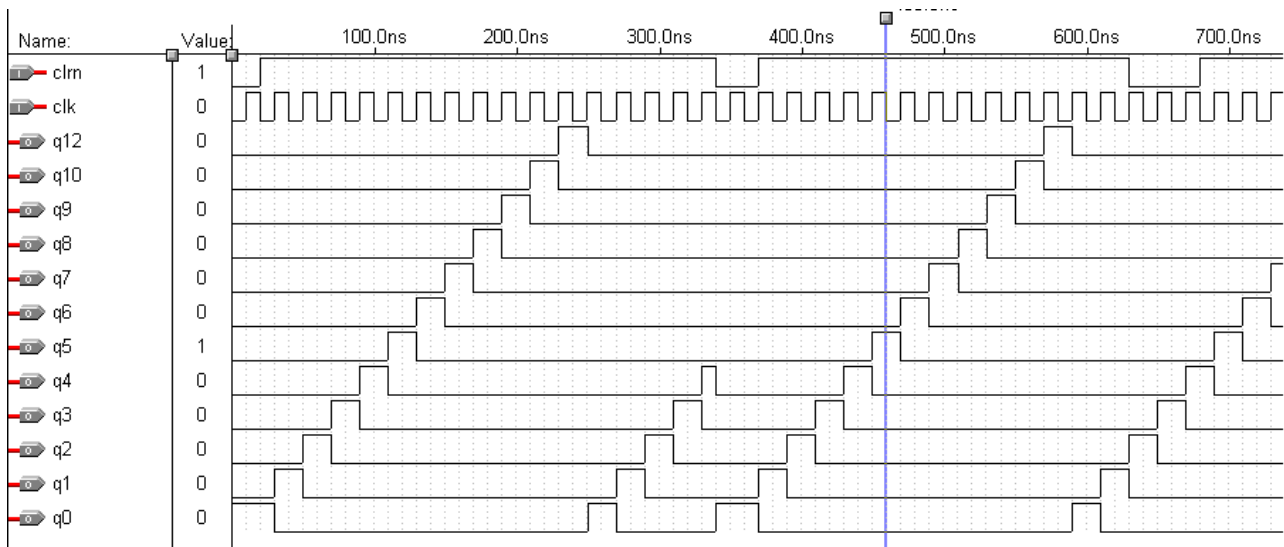


Figure 4.51c Functional simulation results of sub-circuit Ring12.GDF (File: ring12.SCF)

Step 6: Please use the two elements, CLKGEN and RING12 to complete LEDTEST design entry. For the related operational information, please see Section 4.3.

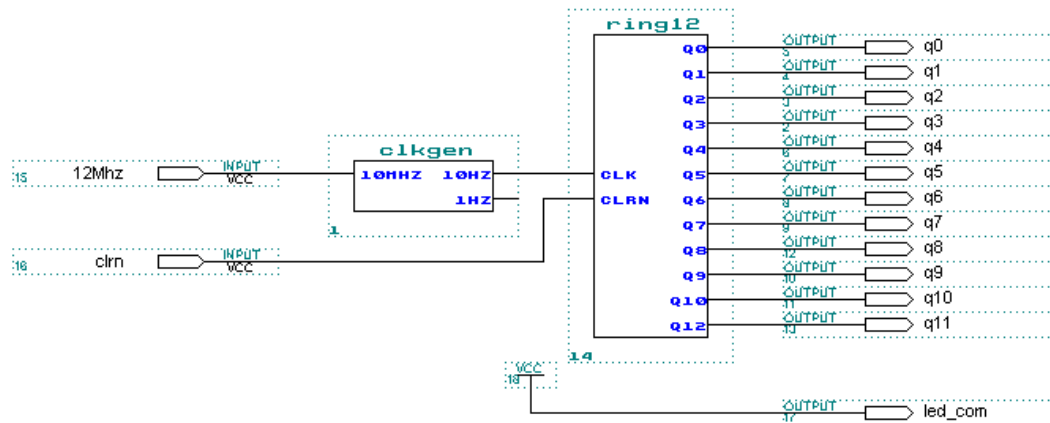


Figure 4.52 Main circuit LEDTEST.GDF (File: LEDTEST.GDF)

Step 7: Please refer to Section 4.5.

1. Call out the compiler by MAX+PLUS II > Compiler.
2. Ensure the design compilation by Processing > Timing SNF Extractor.
3. Select the device family (FLEX10K) and the device (EPF 10K10TC144-4) by Assign > Device.
4. Start compiler for free compilation.
5. Start floorplan by MAX+PLUS II > Floorplan, and then go to pin current assignment window by Layout > Current Assignment. Use Table 4.3 to do pin assignment.
6. After pins are planned already, redo the compilation. Select Layout > Last Compilation for Floorplan Editor to ensure the last compilation is what we expected.

Step 8: Please refer to Section 4.6.

1. Assemble LP-2900 CPLD logic design platform, and connect well with a printer bus.
2. Power on, and click “Reset” bottom. LED D1 will be lightened up.



Table 4.3 Pin assignment of EPF10K10TC144-4

Name of Signal	EPF10K10TC144-4 Pin Count	Name of Signal	EPF10K10TC144-4 Pin Count
CLKIN	Pin 55	Q6	Pin 13
CLRN	Pin 47	Q7	Pin 14
Q0	Pin 7	Q8	Pin 17
Q1	Pin 8	Q9	Pin 18
Q2	Pin 9	Q10	Pin 19
Q3	Pin 10	Q11	Pin 20
Q4	Pin 11		
Q5	Pin 12	LED_COM	Pin 141

3. In MAX+PLUS II, start Programmer window by MAX+PLUS II > Programmer as Figure 4.40. Because the “hardware setting” is totally completed, the information “Programming hardware is not installed.” will not be found. However, if the information is still there, please refer to “programming hardware setting” in Section 4.6.
4. Start Multi-Device JTAG Chain Setup window by JTAG > Multi-Device JTAG Chain Setup as Figure 4.43. After go into the window, if there are still some old programming files (in this case, it should be MOD16.SOF), please delete them as Figure 4.53. Select the device “EPF10K10” from “Device Name” as Figure 4.44a. Click “Select Programming File” and choose the file “LEDTEST.sof” as Figure 4.44b. Click “Add” and then “Ok” to exit the Multi-device JTAG Chain Setup window as Figure 4.44c. If the Figure 4.44d is found, it tells us that Multi-device JTAG Chain Setup mode is closed, and, therefore, click “Yes” please.

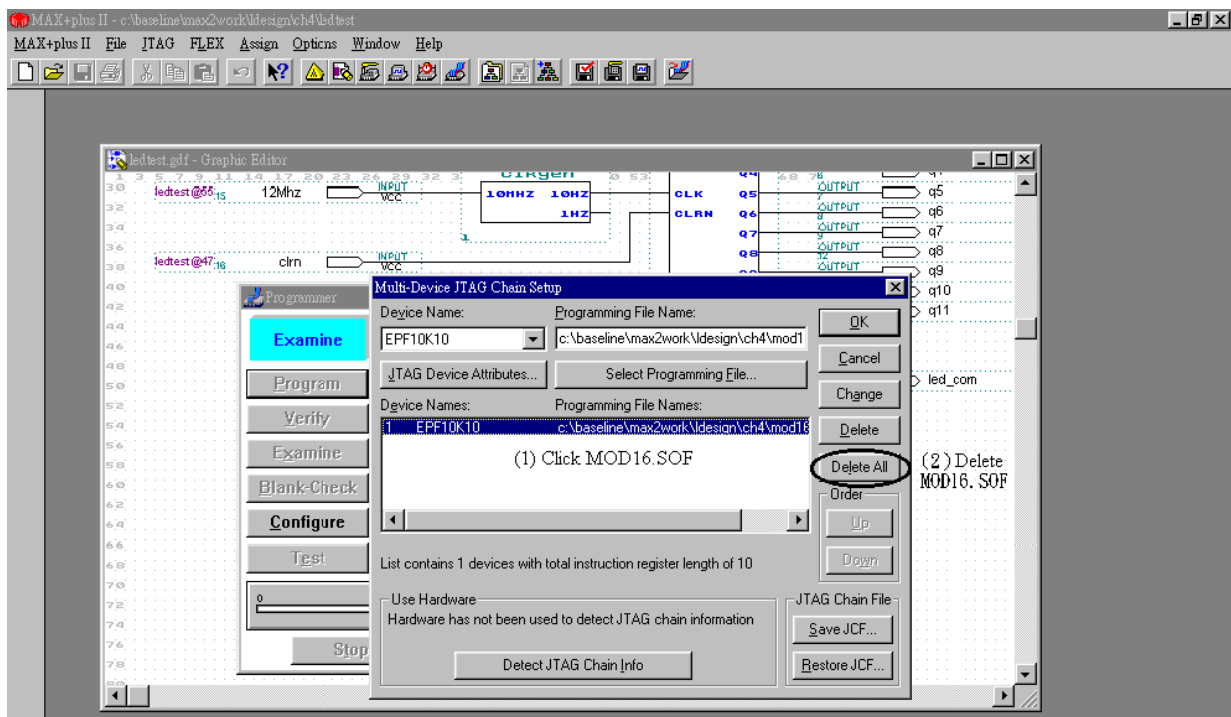


Figure 4.53 Delete old programming file “MOD16.SOF”

5. Use your mouse to click “Configure” in Programmer window, and start to download programming. If the download is not successful, please repeat the steps 1 to 4.
6. If the download is successfully completed, please click SW1 (CLRN) and check if LED is running well. If not, (1) please check if the pin assignment is correct; (2) please download LEDTEST.SOF from CD to check if the clock is correct.

~ *Congratulation! You have Completed Another Practice!* ~

Similarly, we would like to suggest you repeat the steps from 1 to 8. For sure, you will improve a lot next time.



4.8 Review

Please answer the following questions to review this chapter.

- ❑ Do you know the name of EDA software introduced in this chapter? Do you know how to set it up?
- ❑ Do you know what functional simulation is?
- ❑ Do you know what floorplan is?
- ❑ Do you know what device programming is? Which programmable device is discussed in this chapter?
- ❑ Do you know any other design entries except graphic entry?

CHAPTER 5

Combinational Logic Circuit



Combinational logic circuit is main category of logic circuit. This circuit main character is that output relates to now input and no relate to past input. If output relates to now input and past input circuit, it is called sequential logic. We main topic is combinational logic circuit in this chapter.

5.1 The Design, Simulation and Test of General Combinational Logic Circuit

The design of combinational logic circuit main goal is constructing circuit of conformable circuit behave specification by using the less logic gate and the less input. The design step is as follows,

1. Establish Truth table, Karnaugh map or Boolean expression by circuit specification
2. From Truth table or Karnaugh map drives Boolean expression of sum-of-product or product-of-sum
3. Minimize Boolean expression possibly
4. According to Boolean expression, in EDA tool (this book main use MAX+PLUS II) use appropriate logic gate to complete design entry by graphic editor (design entries also includes text and waveform, but this book main uses Figure editor)
5. Then simulation this circuit and check whether the functions meet the specification

After floorplan, downloads this circuit into selected device and performs the circuit test

Now, we use reality example to explain all process. If we have following circuit “the circuit includes four inputs and one output and when input is two or beyond two “1”, the output is “1”, or output is “0”. ”



Step 1 : Establish Truth table, like Table 5.1 (hypothetically, four input variable are A.B.C.D, and output variable is Z) or Karnaugh map of Table5.2.

Table 5.1 Truth table of circuit

Input				Output
A	B	C	D	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Table 5.2 Karnaugh map of circuit

Z		CD			
		00	01	11	10
AB	00	0	0	1	0
	01	0	1	1	1
	11	1	1	1	1
	10	0	1	1	1

Step 2 : To drive Boolean expression of sum-of-product or product-of-sum from Truth table, Table 5.2

$$Z = A'B'CD + A'BC'D + A'BCD' + A'BCD + AB'CD' + AB'C'D + AB'CD + ABCD' + ABC'D' + ABCD + ABC'D \dots\dots\dots(5-1)$$

Step 3: Minimize Boolean expression possibly (or getting minimum Boolean expression by Karnaugh map). For minimum, we will lead some repeat terms into expression (5-1) (the terms of underline), and rewrite expression (5-3).

$$Z = A'B'CD + A'BCD + AB'CD + ABCD + ABC'D' + ABC'D + \underline{ABCD} + ABCD' + A'BC'D + \underline{ABC'D} + \underline{A'BCD} + \underline{ABCD} + AB'C'D + \underline{AB'CD} + \underline{ABC'D} + \underline{ABCD} + A'BCD' + \underline{A'BCD} + \underline{ABCD'} + \underline{ABCD} + AB'CD' + \underline{AB'CD} + \underline{ABCD'} + \underline{ABCD} \dots\dots\dots(5-2)$$

$$\begin{aligned} Z &= (A'B' + A'B + AB' + AB) CD + AB (C'D' + C'D + \underline{CD} + CD') + BD \\ &\quad (A'C' + \underline{AC'} + \underline{A'C} + \underline{AC}) + AD (B'C' + \underline{B'C} + \underline{BC'} + \underline{BC}) + BC (A'D' + \underline{A'D} + \underline{AD'} + \underline{AD}) + AC (B'D' + \underline{B'D} + \underline{BD'} + \underline{BD}) \\ &= AB + AC + AD + BC + BD + CD \dots\dots\dots(5-3) \end{aligned}$$

or like Karnaugh map minimization of Table 5.3a~Table 5.3f (note: please choice in same table when drawing figure) :

Table 5.3a $Z = \underline{AB} + \boxed{?}$

Z		CD			
		00	01	11	10
AB	00	0	0	1	0
	01	0	1	1	1
	11	1	1	1	1
	10	0	1	1	1

Table 5.3b $Z = AB + \underline{CD} + \boxed{?}$

Z		CD			
		00	01	11	10
AB	00	0	0	1	0
	01	0	1	1	1
	11	1	1	1	1
	10	0	1	1	1

Table 5.3c $Z = AB + CD + \underline{AD} + \boxed{?}$

Z		CD			
		00	01	11	10
AB	00	0	0	1	0
	01	0	1	1	1
	11	1	1	1	1
	10	0	1	1	1

Table 5.3d $Z = AB + CD + AD + \underline{BD} + \boxed{?}$

Z		CD			
		00	01	11	10
AB	00	0	0	1	0
	01	0	1	1	1
	11	1	1	1	1
	10	0	1	1	1

Table 5.3e $Z = AB + CD + AD + BD + \underline{BC} + \boxed{?}$

Z		CD			
		00	01	11	10
AB	00	0	0	1	0
	01	0	1	1	1
	11	1	1	1	1
	10	0	1	1	1

Table 5.3f $Z = AB + CD + AD + BD + BC + \underline{AC}$

Z		CD			
		00	01	11	10
AB	00	0	0	1	0
	01	0	1	1	1
	11	1	1	1	1
	10	0	1	1	1

Step 4 : According to Boolean expression, in MAX+PLUS II, uses appropriate logic gate to complete circuit entries of Figure 5.1 by graphic editor.

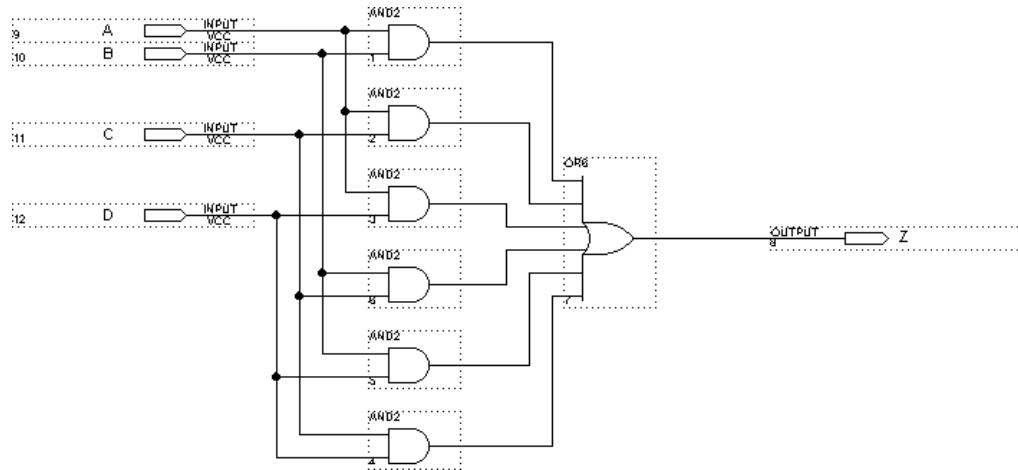


Figure 5.1 Circuit diagrams of ex1.gdf by using graphic entries in MAX+PLUS II
(document : ex1.gdf)

Step 5 : Then simulation this circuit and check whether the functions meet the specification. The Table 5.2 is simulation result of Table 5.1 circuit, we can know requirement of circuit specifications from result of simulation “ the output is ”1” when inputs have two or more two “1”, otherwise output is “0”.”

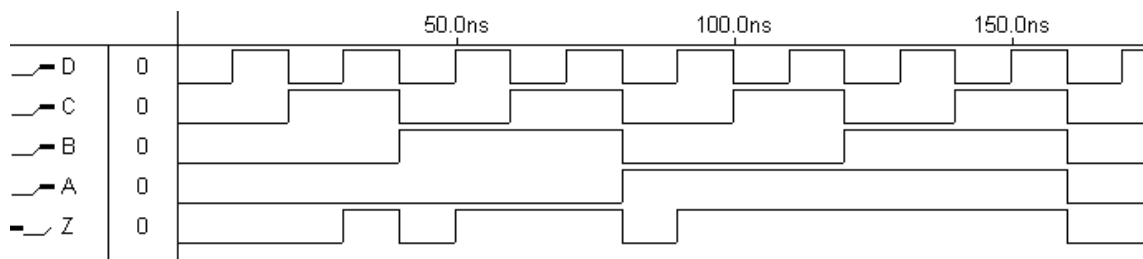


Figure 5.2 Simulation result of ex1.gdf circuit (document : ex1.scf)

Step 6 : After floorplan, download the circuit into selected device and performance the circuit test are needs. For downloading test, we use LEDs on panel of LP-2900 are grouped into common anode then connect to pin 141 of EPF10K10TC144. Therefore, the pin 141 must be connected to Vcc for LED common anode. In other words, the circuit of Figure 5.1 need to be modified to become Figure 5.3 then compiles it again. (in the future, if needs to use LED downloading test, this modified is necessary, the reader must note it) After modified, we can use floorplan technique described in Section 4.6, then select chip EPF10K10TC144-4 and use Table 5.4 pin assignment reference ◦

After assemble platform LP-2900, we download this circuit to chip EPF10K10TC144-4, then try to press SW1, SW2, SW3 and SW4 (left-bottom of LP-2900) and note changes of L1 (Z).

Table 5.4 Pin assignment of EPF10K10TC144-4 on LP-2900

Name of Signal	Pin of EPF10K10TC144-4
A	Pin 47
B	Pin 48
C	Pin 49
D	Pin 51
Z	Pin 7
LED_COM	Pin 141

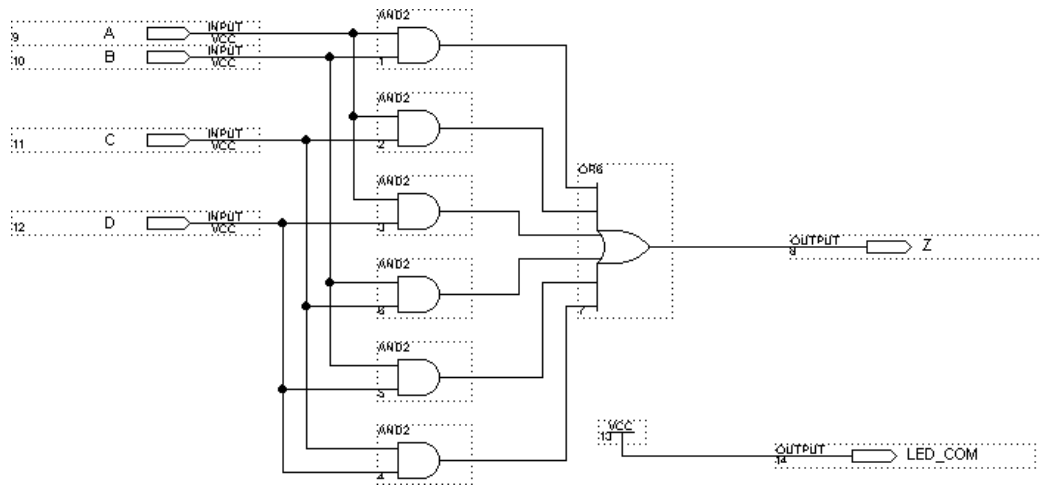


Figure 5.3 The ex1.gdf circuit of after modified

5.2 The Design, Simulation and Test of Adder

Generally, the adder divides into Half Adder and Full Adder. Half Adder includes two inputs (add and addend), the output sum and carry. The Full Adder also includes carry as input, the output sum and carry. We proceed to the design, simulation and test of adder.

5.2.1 The Design, Simulation and Test of Half Adder

The circuit specification of Half Adder is “ circuit includes two inputs A and B, and one Sum output and Cout carry output. The Sum outputs “0” and the Cout outputs “0” when input is “00” ; the Sum outputs “1” and the Cout outputs “0” when input is “01” ; the Sum outputs “1” and the Cout outputs “0” when input is “10” ; the Sum outputs “0” and Cout outputs “1” when input is “11” .”. According to the procedure illustrated in above section, we design as follows :

Step 1 : Establish Truth table, like Table 5.5 (hypothetically, two inputs are A and B, and outputs are Sum and Cout).

Table 5.5 Truth table of Half Adder

Input		Output	
A	B	Sum	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Step 2 : To drive Boolean expression of sum-of-product or product-of-sum from Truth table, Table 5.5

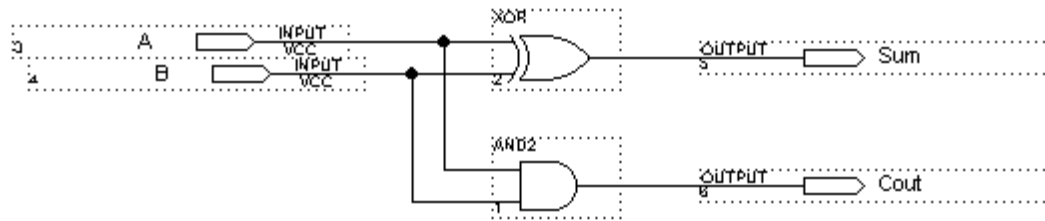
$$\text{Sum} = AB' + A'B = A \oplus B \dots\dots\dots (5-4)$$

$$\text{Cout} = AB \dots\dots\dots (5-5)$$

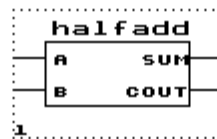
Step 3 : Minimize Boolean expression possibly. Because the expression (5-4) and expression (5-5) had been minimized, they will not be minimized.

Step 4 : According to Boolean expression, in MAX+PLUS II, uses appropriate logic gate to complete circuit entries of Figure 5.4 by graphic editor.

Step 5 : Then simulation this circuit and check whether the functions meet the specification. The Table 5.5 is simulation results of Half Adder, and makes sure function specification of conformable.



(a) Half-adder



(b) Circuit Symbol of Half-adder Generated by MAX+plus II

Figure 5.4 Circuit diagrams and symbol by using graphic entries in MAX+PLUS II (document : halfadd.gdf)

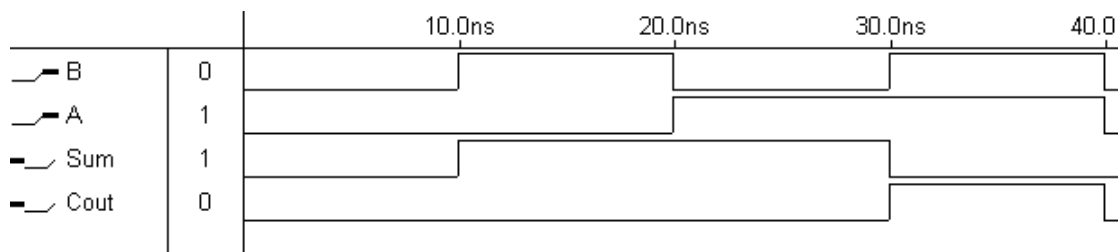


Figure 5.5 Simulation result of halfadd.gdf circuit (document : halfadd.scf)

Step 6 : After floorplan, download the circuit into selected device and performance the circuit test are needs. For downloading test, we use LEDs on panel of LP-2900 to group into common anode then connect to pin 141 of EPF10K10TC144. Therefore, the pin 141 must be connected to Vcc for LED common a node. In other words, the circuit of Figure 5.5 need to be modified to become Figure 5.6 then compiles it again. After modified, we

can use floorplan technique of Section 4.6, then select chip EPF10K10TC144-4 and use Table 5.6 pin assignment reference. After assemble platform LP-2900, we download Half Adder to chip EPF10K10TC144-4. Please try to push SW1 and SW2 on left-bottom of LP-2900, and please note the changes of L1 (Sum) and L2 (Cout).

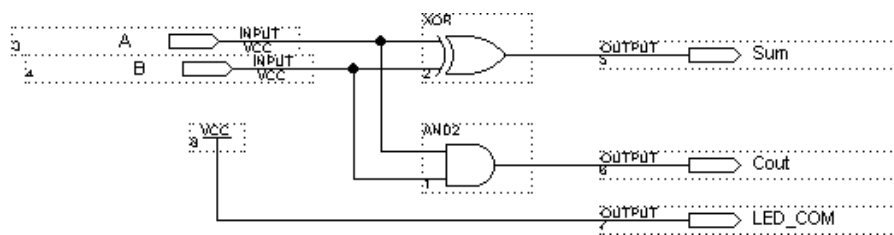


Figure 5.6 The halfadd.gdf circuit of after modified

Table 5.6 Pin assignment of chip EPF10K10TC144-4

Name of Signal	Pin of EPF10K10TC144-4
A	Pin 47
B	Pin 48
Sum	Pin 7
Cout	Pin 8
LED_COM	Pin 141

5.2.2 The Design, Simulation and Test of Full Adder

Now, we proceed to the design, simulation and test of Full Adder. The circuit specification of Full adder is “ circuit includes three inputs (A, B, and Cin), one

Sum output and Cout output. The Sum outputs “0” and the Cout outputs “0” when input is “000” ; the Sum outputs “1” and the Cout outputs “0” when input is “010” ; the Sum outputs “1” and the Cout outputs “0” when input is “100” ; the Sum outputs “0” and Cout outputs “1” when input is “101” ; the Sum outputs “0” and the Cout outputs “1” when input is “110” ; the Sum outputs “1” and the Cout outputs “1” when input is “111””. As the Half Adder, we design Full Adder as follows :

Step 1 : Establish Truth table like Table 5.7 (hypothetically, three inputs are A, B and Cin, and outputs are Sum and Cout).

Table 5.7 Truth table of Full Adder

Input			Output	
A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Step 2 : To drive Boolean expression of sum-of-product or product-of-sum from Truth table, Table 5.7

$$\text{Sum} = A'B'Cin + A'BCin' + AB' Cin' + ABCin \dots\dots\dots (5-6)$$

$$Cout = A'Bcin + AB'Cin + ABCin' + ABCin \dots\dots\dots(5-7)$$

Step 3 : Minimize Boolean expression possibly.

$$\begin{aligned} Sum &= A'B'Cin + A'BCin' + AB'Cin' + ABCin \\ &= (A'B' + AB) Cin + (A'B + AB') Cin' \\ &= (A \oplus B)' Cin + (A \oplus B) Cin' \\ &= A \oplus B \oplus Cin \dots\dots\dots(5-8) \end{aligned}$$

$$\begin{aligned} Cout &= A'Bcin + AB'Cin + ABCin' + ABCin \\ &= (A \oplus B) Cin + (Cin + Cin') AB \\ &= (A \oplus B) Cin + AB \dots\dots\dots(5-9) \end{aligned}$$

Step 4 : According to Boolean expression, in MAX+PLUS II, uses appropriate logic gate to complete circuit entries of Figure 5.7 by graphic editor.

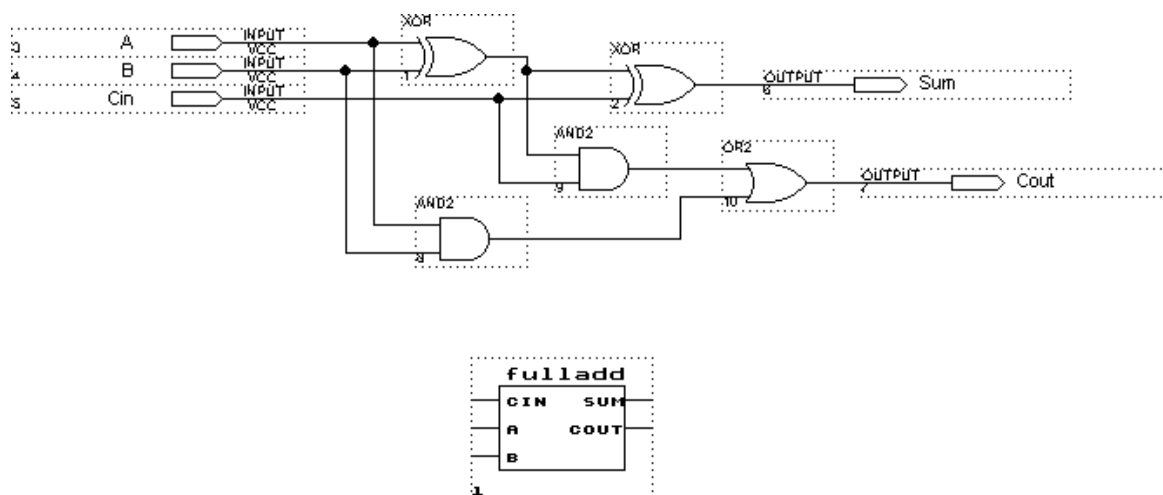


Figure 5.7 Circuit diagrams and symbol by using graphic entries in MAX+PLUS II (document : fulladd.gdf)

Step 5 : Then simulation the circuit and check whether the functions meet the specification. Figure 5.8 is simulation result of Full adder.

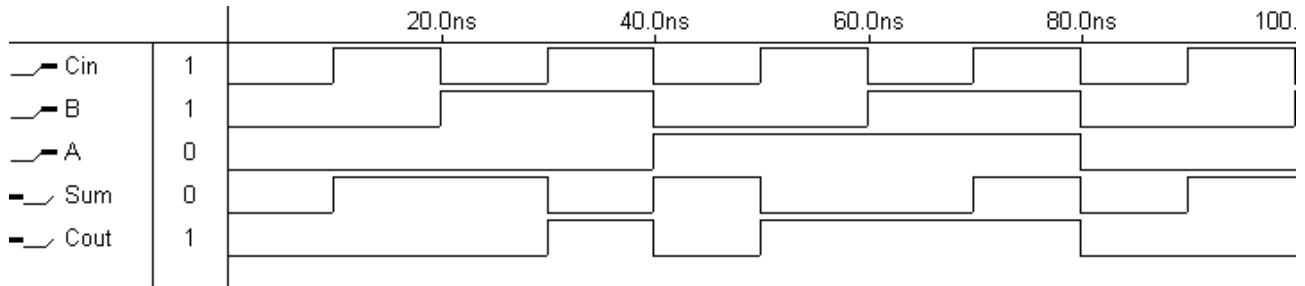


Figure 5.8 Simulation result of fulladd.gdf circuit (document : fulladd.scf)

Step 6 : After floorplan, download Full Adder and perform the circuit test are needs.

As circuit modified that showed in Figure 5.6 of Section 5.2.1, please modify Full adder circuit of Figure 5.8. Please re-compile it after modifying, and adapt the floorplan techniques in Section 4.6, select chip EPF10K10TC144-4 and use Table 5.8 pin assignment reference. After assemble Lab platform LP-2900, download Full Adder to chip EPF10K10TC144-4. Please try to push SW1 (A), SW2 (B) and SW3 (Cin) on left-bottom of LP-2900, and please note the changes of L1 (Sum) and L2 (Cout)

Table 5.8 Pin assignment of EPF10K10TC144-4

Name of Signal	Pin of EPF10K10TC144-4
A	Pin 47
B	Pin 48
Cin	Pin 49
Sum	Pin 7
Cout	Pin 8
LED_COM	Pin 141

5.2.3 The Design, Simulation and Test of Ripple Carry Adder

In primary two sections, the adder only deals with add of one-bit, but actual circuit usual completes addition of four-bit and beyond four-bit. In multiple bits addition, if carry bit of each addition state uses carry out of previous addition stage. This is called Ripple Carry Adder as Figure 5.9. The simulation result is as Figure 5.10 .

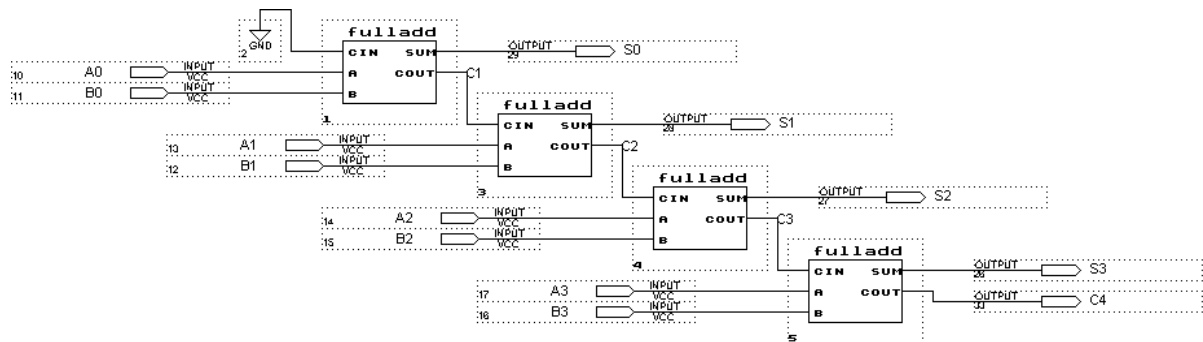


Figure 5.9 Circuit diagrams of Ripple Carry Adder by using graphic entries in MAX+PLUS II (document : rip_add.gdf)

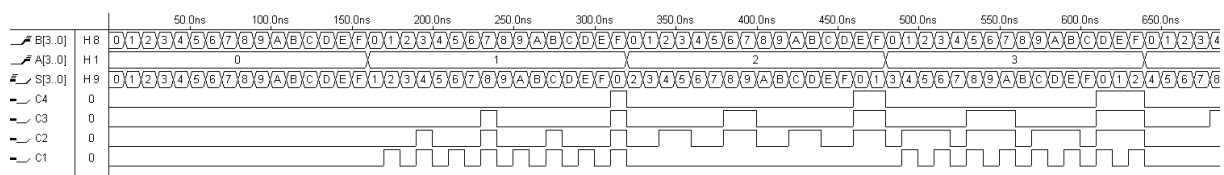


Figure 5.10 Simulation result of circuit rip_add.gdf (document : rip_add.scf)

Step 1: Establish Boolean expression. According to expression (5-8) and expression (5-9), we can drives Boolean expression of $S_0 \sim S_3$ and $C_1 \sim C_4$.

$$S_0 = A_0 \oplus B_0 \oplus C_0 \dots\dots\dots (5-10)$$

$$C_1 = (A_0 \oplus B_0) C_0 + A_0 B_0 \dots\dots\dots (5-11)$$

$$S_1 = A_1 \oplus B_1 \oplus C_1 \dots\dots\dots (5-12)$$

$$C_2 = (A_1 \oplus B_1) C_1 + A_1 B_1 \dots\dots\dots (5-13)$$

$$S_2 = A_2 \oplus B_2 \oplus C_2 \dots\dots\dots (5-14)$$

$$C_3 = (A_2 \oplus B_2) C_2 + A_2 B_2 \dots\dots\dots (5-15)$$

$$S_3 = A_3 \oplus B_3 \oplus C_3 \dots\dots\dots (5-16)$$

$$C_4 = (A_3 \oplus B_3) C_3 + A_3 B_3 \dots\dots\dots (5-17)$$

Step 2 : Minimize Boolean expression possibly. The expression (5-10) and (5-17) had been minimized, it will not be minimized.

Step 3 : According to Boolean expression, in MAX+PLUSII, uses appropriate logic gate to complete circuit entries by graphic editor. We use one Full adder to complete expression 5-10 and expression 5-11; use another Full adder to complete expression 5-12 and expression 5-13 ; use the other Full adder to complete expression 5-14 and expression 5-15 ; use last Full adder to complete expression 5-16 and expression 5-17 ; The circuit be illustrated as Figure 5.9.

Step 4 : Then simulation the circuit and check whether the functions meet the specification. Figure 5.10 is simulation result of Ripple Carry Adder.

Step 5 : After floorplan, download this circuit into selected device and performance the circuit test are needs. As circuit modified that showed in Figure 5.6 of Section 5.2.1, please modify Full Adder circuit of Figure 5.9. Please re-

compile it after modifying, and adapt the floorplan techniques in Section 4.6, select chip EPF10K10TC144-4 and use Table 5.9 pin assignment reference. After assemble Lab platform LP-2900, download Ripple Carry Adder to chip EPF10K10TC144-4. Please try to push SW12~SW9 (MSB) and SW20~SW17 (LSB) on left-bottom of LP-2900, and please note the changes of L5~L1, for example, you push 5+2 and see if it becomes “00111”

Table 5.9 Pin assignment of EPF10K10TC144-4

Name of Signal	Pin of EPF10K10TC144-4	Name of Signal	Pin of EPF10K10TC144-4
A0	Pin 68	S0	Pin 11
A1	Pin 67	S1	Pin 10
A2	Pin 65	S2	Pin 9
A3	Pin 64	S3	Pin 8
B0	Pin 81	C4	Pin7
B1	Pin 80	B3	Pin78
B2	Pin 79	LED_COM	Pin 141

Note : There are no pin assignment for C1, C2 and C3.

Discussion : If the propagation delay of AND gate, XOR gate (exclusion) and OR gate are t_{pd} , then we can get accurate time required of $S_0 \sim S_3$ and $C_1 \sim C_4$ as

Table 5.10,

Table 5.10 Table of time required getting accurate $S_0 \sim S_3$ and $C_1 \sim C_4$

Output Signal	Time Required	Description
S_0	$2 t_{pd}$	Propagation delay of two XOR gates
C_1	$3 t_{pd}$	Propagation delay of one XOR, one AND and one OR
S_1	$4 t_{pd}$	Propagation delay of add one XOR after showed C_1
C_2	$5 t_{pd}$	Propagation delay of add one AND and one OR after
S_2	$6 t_{pd}$	Propagation delay of add one XOR after showed C_2
C_3	$7 t_{pd}$	Propagation delay of add one AND and one OR after
S_3	$8 t_{pd}$	Propagation delay of add one XOR after showed C_3
C_4	$9 t_{pd}$	Propagation delay of add one AND and one OR after

For a N-bit adder, we can drive propagation delay of C_n

$$2n + 1, n = 1 \sim N \quad \text{.....} \quad (5-18)$$

And propagation delay of S_m

$$(m + 1) * 2, m = 0 \sim (N-1) \quad \text{.....} \quad (5-19)$$

We get truth from expression (5-18) and expression (5-19), it is when N is large and propagation delay of S_m and C_n is more. In other words, the Ripple Carry Adder needs more time to finish addition, because serial connected cause this result.

5.2.4 The Design, Simulation and Test of Carry Look-ahead Adder

In Section 5.2.3, we understand that Ripple Carry Adder has the more serial bit connection the more finish time required. For reduced time of add operation, the

parallel connected is good way, which carry input can't use previous stage carry output and causes propagation wait. In other word, we need use special design to get carry input, this approach is called Look-ahead Carry.

Step 1 : Establish Boolean expression, if we define two functions as follows,

$$\text{generation function : } G_i = A_i B_i \quad \text{..... (5-20)}$$

$$\text{propagation function : } P_i = A_i \oplus B_i \quad \text{..... (5-21)}$$

and take expression (5-20) and (5-21) into (5-11), we can drive Boolean algebra expression C_1 as follows :

$$C_1 = P_0 C_0 + G_0 \quad \text{..... (5-22)}$$

Taking expression (5-22) into expression (5-13), we can drive Boolean algebra expression C_2 as follows :

$$C_2 = P_1 P_0 C_0 + P_1 G_0 + G_1 \quad \text{..... (5-23)}$$

Taking expression (5-23) into expression (5-15), we can drive Boolean algebra expression C_3 as follows :

$$C_3 = P_2 P_1 P_0 C_0 + P_2 P_1 G_0 + P_1 G_1 + G_2 \quad \text{..... (5-24)}$$

Taking expression (5-24) into expression (5-17), we can drive Boolean algebra expression C_4 as follows :

$$C_4 = P_3 P_2 P_1 P_0 C_0 + P_3 P_2 P_1 G_0 + P_3 P_2 G_1 + P_3 G_2 + G_3 \quad \text{..... (5-25)}$$

then we can get accurate time required of $S_0 \sim S_3$ and $C_1 \sim C_4$ as Table 5.11

Table 5.11 Time required for getting accurate $S_0 \sim S_3$ and $C_1 \sim C_4$ in

Carry Look-ahead

Output Signal	Time Required	Description
S_0	$2 t_{pd}$	Propagation delay of two XOR gates
C_1	$3 t_{pd}$	Propagation delay of one XOR, one AND and one OR
S_1	$4 t_{pd}$	Propagation delay of add one XOR after showed C_1
C_2	$3 t_{pd}$	Propagation delay of add one XOR, one AND and one OR
S_2	$4 t_{pd}$	Propagation delay of add one XOR after showed C_2
C_3	$3 t_{pd}$	Propagation delay of add one XOR, one AND and one OR
S_3	$4 t_{pd}$	Propagation delay of add one XOR after showed C_3
C_4	$3 t_{pd}$	Propagation delay of add one XOR, one AND and one OR

From Table 5.11, we can understand that the finish time required for addition is fix-value, not relative to bit length of Carry Look-ahead Adder. The time required of Carry Look-ahead Adder is less then Ripple Carry Adder.

Step 2 : Minimize Boolean expression possibly. The expression (5-22) and (5-25) had been minimized, they will not be minimized.

Step 3 : According to Boolean expression, in MAX+PLUS II , uses appropriate logic gate to complete circuit entries by graphic editor. For achievement Carry Look-ahead Adder, we design new adder of Figure 5.11. It includes Sum, Count output, propagation function output and generation function output. Design circuit of Figure 5.12 Carry Look-ahead Adder by using this new Full Adder and basic of expression of (5-22)~(5-25)

Step 4 : Then simulation this circuit and check whether the functions meet the specification. Figure 5.13 is simulation result of Carry Look-ahead Adder.

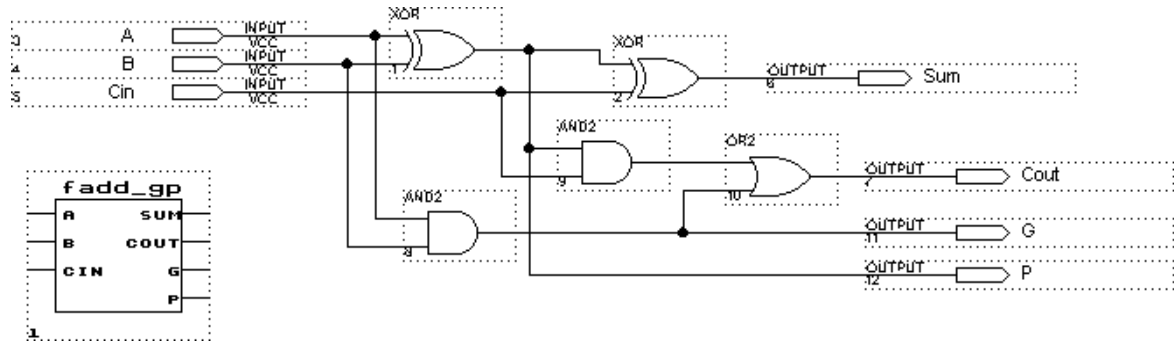


Figure 5.11 New Full Adder and symbol(document : fadd_gp.gdf)

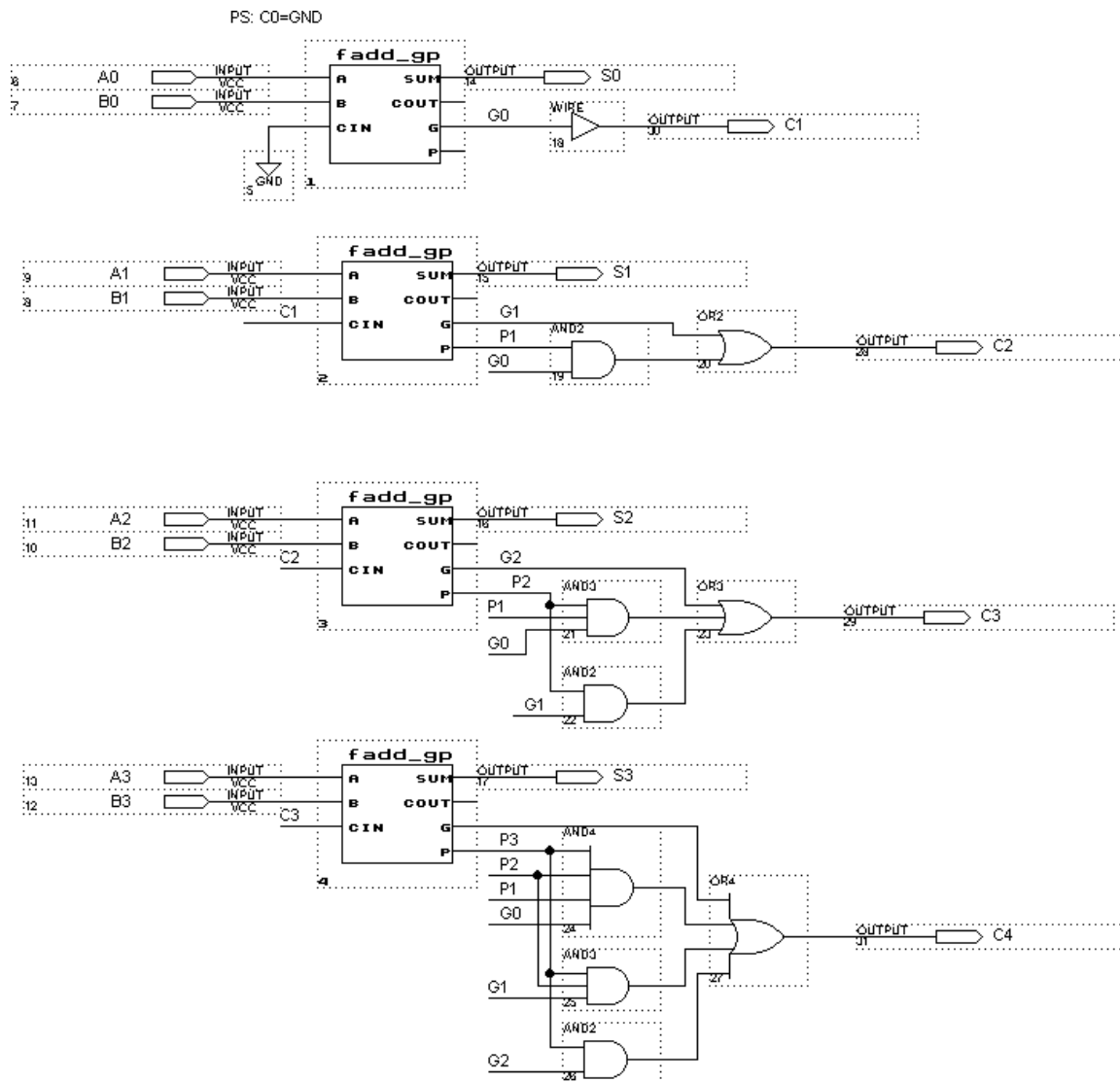


Figure 5.12 Carry of Carry Look-ahead Adder by using graphic entries in MAX+PLUS II (document : lah_add.gdf)

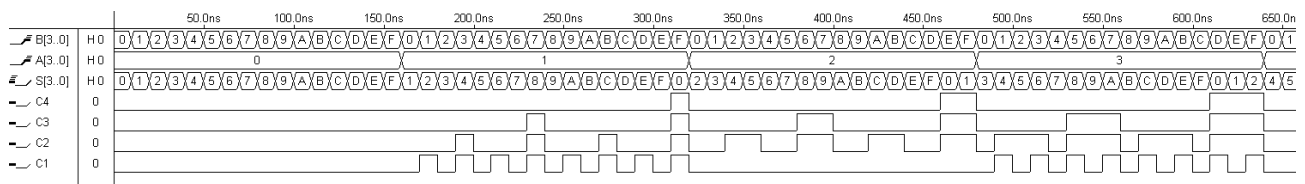


Figure 5.13 Simulation result of circuit lah_add.gdf (document : lah_add.scf)

Step 5 : After floorplan, download this circuit into selected device and perform the circuit test are needs. As circuit modified that showed in Figure 5.6 of Section 5.2.1, please modify circuit of Figure 5.12. Please re-compile it after modifying, and adapt the floorplan techniques in Section 4.6, select chip EPF10K10TC144-4 and use Table 5.12 pin assignment reference. After assemble Lab platform LP-2900, download Carry Look-ahead Adder to chip EPF10K10TC144-4. Please try to push SW12~SW9 (MSB) and SW20~SW17 (LSB) on left-bottom of LP-2900, and please note the changes of L5~L1, for example, you push 5+4 and see if it becomes “00111”

Table 5.12 Pin assignment of EPF10K10TC144-4

Name of Signal	Pin of EPF10K10TC144-4	Name of Signal	Pin of EPF10K10TC144-4
A0	Pin 68	S ₀	Pin 11
A1	Pin 67	S ₁	Pin 10
A2	Pin 65	S ₂	Pin 9
A3	Pin 64	S ₃	Pin 8
B0	Pin 81	C ₄	Pin7
B1	Pin 80	B ₃	Pin78
B2	Pin 79	LED_COM	Pin 141

Note: There are no pin assignments for C₁, C₂ and C₃.

5.3 The Design, Simulation and Test of Subtractor

5.3.1 The Design, Simulation and Test of Half Subtractor

The circuit specification of Half Subtractor is "the circuit includes two inputs (X, Y), one different output and borrow output. When input is "00", the different output is "0", the borrow output is "0"; when input is "01", the different is "1", the borrow is "1"; when input is "10", the different is "1", the borrow is "0"; when input is "11", the different is "0", the borrow output is "0"." According to the procedure described Section 5.1, we design as follows,

Step 1 : Establish Truth table as Table 5.13 (hypothetically, the two inputs are X and Y, and output is different and borrow)

Table 5.13 Truth table of Half Subtractor circuit

Input		Output	
X	Y	Dif	Bo
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Step 2 : To drive Boolean expression of sum-of-product or product-of-sum from Truth table.

$$\text{Dif} = XY' + X'Y = X \oplus Y \dots\dots\dots (5-26)$$

$$B_o = X'Y \quad \text{..... (5-27)}$$

Step 3 : Minimize Boolean expression possibly. Since the expression (5-26) and expression (5-27) had been minimized, they will not be minimized.

Step 4 : According to Boolean expression, in MAX+PLUS II, uses appropriate logic gate to complete circuit entries of Figure 5.14 by graphic editor.

Step 5 : Then simulation this circuit and check whether the functions meet the specification. Figure 5.15 is simulation result of Half Subtractor

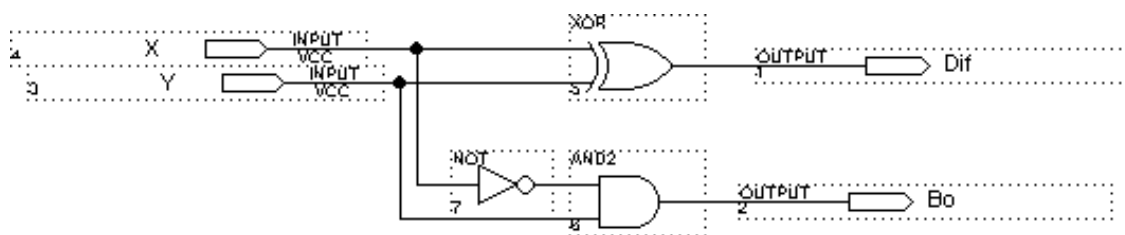


Figure 5.14 Circuit of Half Subtractor by using graphic entries in MAX + PLUS II
(document : halfsub.gdf)

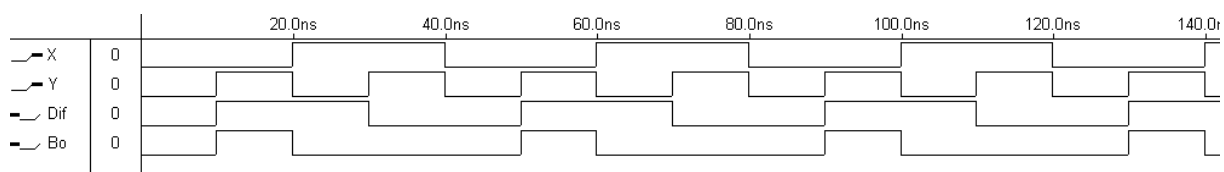


Figure 5.15 Simulation result of halfsub.gdf circuit (document : halfsub.scf)

Step 6 : After floorplan, download this circuit into selected device and perform the circuit test are needs. As circuit modified that showed in Figure 5.6 of Section 5.2.1, please modify Half Subtractor circuit of Figure 5.14. Please re-compile it after modifying, and adapt the floorplan techniques in

Section 4.6, select chip EPF10K10TC144-4 and use Table 5.14 pin assignment reference. After assemble Lab platform LP-2900, download Half Subtractor to chip EPF10K10TC144-4. Please try to push SW1 and SW2 on left-bottom of LP-2900, and please note the changes of L1 (Dif) and L2 (Bo).

Table5.14 Pin assignment of EPF10K10TC144-4

Name of Signal	Pin of EPF10K10TC144-4
X	Pin 47
Y	Pin 48
Dif	Pin 7
Bo	Pin 8
LED_COM	Pin 141

5.3.2 The Design, Simulation and Test of Full Subtractor

Now, we progress the design, simulation and test of Full Subtractor. The circuit specification of Full Subtractor is "the circuit includes three inputs (X, Y and Bin), one different output and borrow output. When input is "000", the different output is "0", the borrow output is "0" ; when input is "001", the different output is "1", the borrow output is "1" ; when input is "010", the different output is "1", the borrow output is "1" ; when input is "011", the different output is "0", the borrow output is "1" ; when input is "100", the different output is "1", the borrow output is "0" ; when input is "101", the different output is "0", the borrow output is "0" ;when input is "110", the different output is "0",the borrow output is "0" ; when input is



“111” , the different output is “1”,the borrow output is “1”. Like Half Subtractor, we design of Full Subtractor as follows:

Step 1 : Establish Truth table as Table 5.15 (hypothetically, the three inputs are X a Y and Bin and output is Dif and Bo)

Table 5.15 Truth table of circuit

Input			Output	
X	Y	Bin	Dif	Bo
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Step 2 : To drive Boolean expression of sum-of-product or product-of-sum from Truth table.

$$\text{Dif} = X'Y'\text{Bin} + X'Y\text{Bin}' + XY'\text{Bin}' + XY\text{Bin} \dots\dots\dots (5-28)$$

$$\text{Bo} = X'Y'\text{Bin} + X'Y\text{Bin}' + X'Y\text{bin} + XY\text{Bin} \dots\dots\dots (5-29)$$

Step 3 : Minimize Boolean expression possibly.

$$\begin{aligned}
 \text{Dif} &= X'Y'\text{Bin} + X'Y\text{Bin}' + XY'\text{Bin}' + XY\text{Bin} \\
 &= (X'Y' + XY) \text{Bin} + (X'Y + XY') \text{Bin}' \\
 &= (X \oplus Y)'\text{Bin} + (X \oplus Y) \text{Bin}' \\
 &= X \oplus Y \oplus \text{Bin} \dots\dots\dots (5-30)
 \end{aligned}$$

$$\begin{aligned}
 \text{Bo} &= X'Y'\text{Bin} + X'Y\text{Bin}' + X'Y\text{bin} + XY\text{Bin} \\
 &= X' (Y'\text{Bin} + Y\text{Bin}') + Y\text{Bin} \\
 &= X' (Y \oplus \text{Bin}) + Y\text{Bin} \dots\dots\dots (5-31)
 \end{aligned}$$

Step 4 : According to Boolean expression, in MAX+PLUS II, uses appropriate logic gate to complete circuit entry of Figure 5.16 by graphic editor.

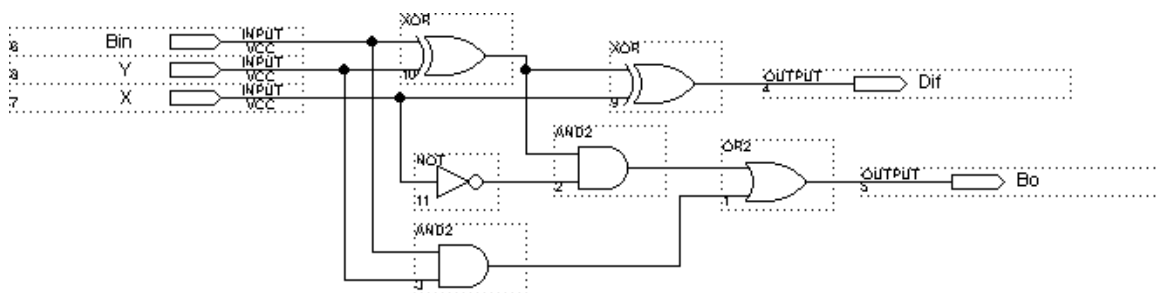


Figure 5.16 Circuit of Full Subtractor by using graphic entries in MAX+PLUS II
(document : fullsub.gdf)

Step 5 : Then simulation this circuit and check whether the functions meet the specification. The Figure 5.17 is simulation result of Full Subtractor.

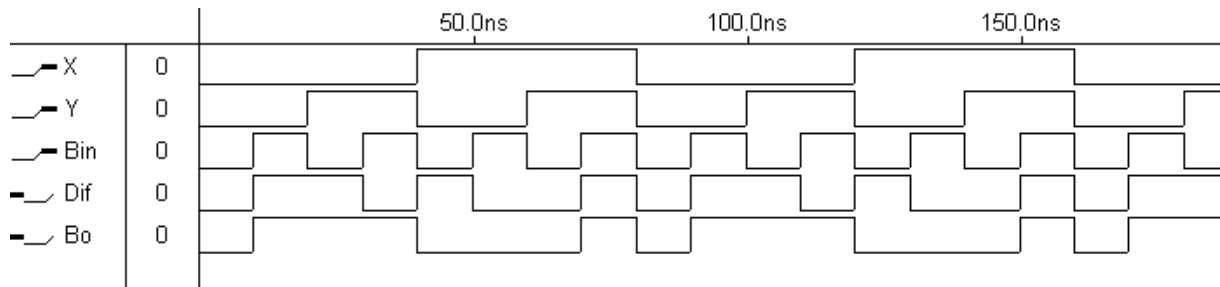


Figure 5.17 Simulation result of circuit fullsub.gdf (document : fullsub.scf)

Step 6 : After floorplan, download this circuit into selected device and perform the circuit test are needs. As circuit modified that showed in Figure 5.6 of Section 5.2.1, please modify Full Subtractor circuit of Figure 5.16. Please re-compile it after modifying, and adapt the floorplan techniques in Section 4.6, select chip EPF10K10TC144-4 and use Table 5.16 pin assignment reference. After assemble Lab platform LP-2900, download Full Subtractor to chip EPF10K10TC144-4. Please try to push SW1 (X), SW2 (Y) and SW3 (Bin) on left-bottom of LP-2900, and please note the changes of L1(Dif) and L2(Bo).

Table 5.16 Pin assignment of EPF10K10TC144-4

Name of Signal	Pin of EPF10K10TC144-4
X	Pin 47
Y	Pin 48
Bin	Pin 49
Dif	Pin 7
Bo	Pin 8
LED_COM	Pin 141

5.3.3 The Design, Simulation and Test of 2's Complement Subtractor

In Chapter two, we understand it all new digital system use 2's complement system. In other words, we can use 2's complement addition to complete subtraction. In Chapter two, the 2's complement can be completed by 1's complement plus one as Figure 5.18(a). So 2's complement of B can be completed by taking NOT from $B_0 \sim B_3$ and setting C_0 to 1. Figure 5.18(b) is circuit of 2's Complement Subtractor, and Figure 5.19 is simulation result.

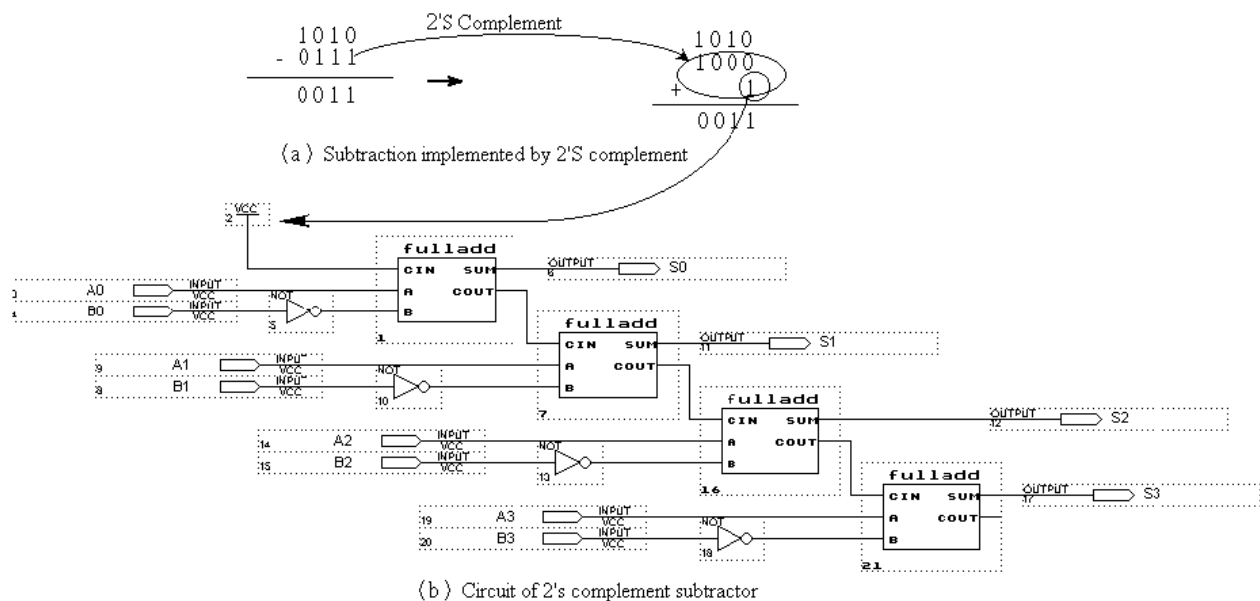


Figure 5.18 Circuit of 2's Complement Subtractor (document : compsub.gdf)

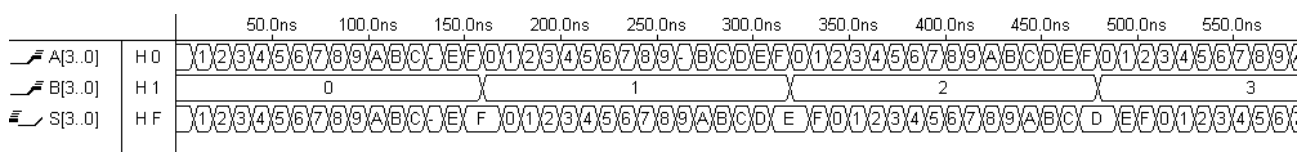


Figure 5.19 Simulation result of circuit 2's Complement Subtractor (document : compsub.scf)

5.4. The Design, Simulation and Test of Comparator

The comparator decides the relationship between input A and input B. It compares input of two n-bit binary number and produces three possible relation outputs (G, E and L). The specification of comparator is "when 2 bits X value is greater than 2 bits Y value, the G output is "1" otherwise G output is "0" ; when 2 bits X value is equal to 2 bits Y value, the E output "1", otherwise E output is "0" ; when 2 bits X value is less 2 bits Y value , the L output is "1", otherwise L output is "0".". After having those specifications, we can start to design the circuit.

Step 1 : Establish Karnaugh map, as Table 5.17a~5.17c .

Table 5.17a Karnaugh map of output G

G		Y_1Y_0			
		00	01	11	10
X_1X_0	00	0	0	0	0
	01	1	0	0	0
	11	1	1	0	1
	10	1	1	0	0

Table 5.17b Karnaugh map of output E

E		Y_1Y_0			
		00	01	11	10
X_1X_0	00	1	0	0	0
	01	0	1	0	0
	11	0	0	1	0
	10	0	0	0	1

Table 5.17c Karnaugh map of output L

L		Y ₁ Y ₀			
		00	01	11	10
X ₁ X ₀	00	0	1	1	1
	01	0	0	1	1
	11	0	0	0	0
	10	0	0	1	0

Step 2 : To drive minimum Boolean expression from Truth table

$$G = X_1 Y_1' + X_0 Y_1' Y_0' + X_1 X_0 Y_0' \dots\dots\dots (5-32)$$

$$E = X_1' X_0' Y_1' Y_0' + X_1' X_0 Y_1' Y_0 + X_1 X_0 Y_1 Y_0 + X_1' X_0 Y_1' Y_0 \dots\dots\dots (5-33)$$

$$L = X_1' Y_1 + X_1' X_0' Y_0 + X_0' Y_1 Y_0 \dots\dots\dots (5-34)$$

Step 3 : Minimize Boolean expression possibly. Because the expression (5-32) and (5-32) and expression (5-33) had been minimized, they will not be minimized.

Step 4 : According to Boolean expression, in MAX+PLUS II, uses appropriate logic gate to complete circuit entries of Figure 5.20 by graphic editor.

Step 5 : Then simulation this circuit and check whether the functions meet the specification. Figure 5.21 is simulation result of Comparator.

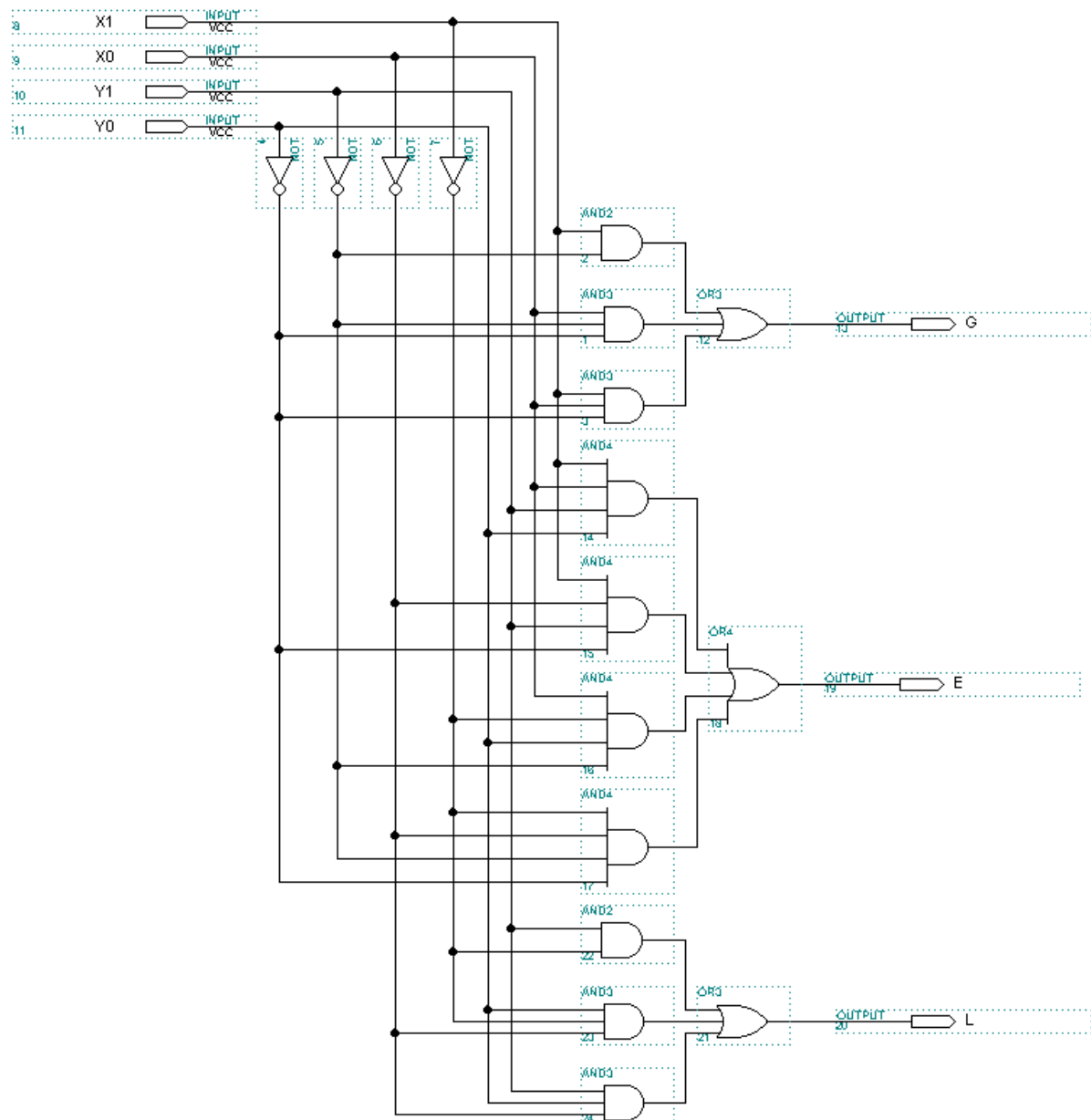


Figure 5.20 Circuit of Comparator by using graphic entries in MAX+PLUS II
(document : compr.gdf)

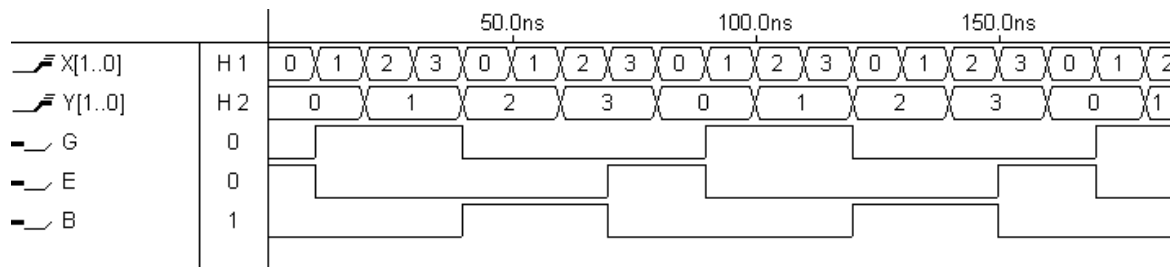


Figure 5.21 Simulation result of circuit compr.gdf (document : compr.scf)

Step 5 : After floorplan, download this circuit into selected device and performance the circuit test are needs. As circuit modified that showed in Figure 5.6 of Section 5.2.1, please modify Comparator circuit of Figure 5.20. Please re-compile it after modifying, and adapt the floorplan techniques in Section 4.6, select chip EPF10K10TC144-4 and use Table 5.18 pin assignment reference. After assemble Lab platform LP-2900, download Comparator circuit to chip EPF10K10TC144-4. Please try to push SW1 (X1), SW2 (X2), SW7 (Y1) and SW8 (Y0) on left-bottom of LP-2900, and please note the changes of L1 (G), L2 (E) and L3 (L).

Table5.18 Pin assignment of EPF10K10TC144-4

Name of Signal	Pin of EPF10K10TC144-4
X0	Pin 48
X1	Pin 47
Y0	Pin 63
Y0	Pin 62
G	Pin 7
E	Pin 8
L	Pin 9
LED_COM	Pin 141

5.5 The Design, Simulation and Test of Encoder

Because the memory cell of calculator is 2-state memory way, the calculator external character set (alphabet, numeric and symbol) need to encode by binary, then save in memory. This encoding movement usual needs encoding circuit to finish. On the other hand, the encoded code which get from memory need to be decoded for output in its original form. Figure 5.22 illustrates the procedure from input, save in binary, process and output data. Generally, the character set not only express in binary, but also binary encoding. The reason is reduced binary bit length. We see this situation from Table 5.19, the binary of alphabetic character set“0”~“9” need ten-bit and binary encoding only need four-bit.

Figure 5.23 shows that key encoded in binary after expressed in binary. After understanding Encoder, we will complete the design, simulation and test of Encoder as Figure 5.23.

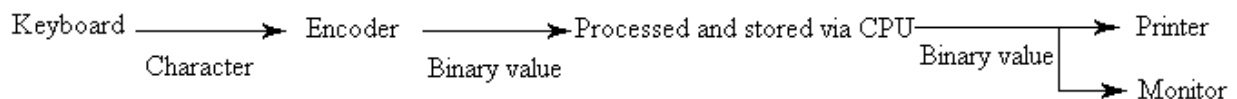


Figure5.22 Input, save in binary, process and output data process flows.

Table 5.19 Compare binary expression of alphabet character set “0”~“9” with binary encoding of alphabet character set “0”~“9”

Character	Binary Expression	Binary
0	0000000001	0000
1	0000000010	0001
2	0000000100	0010
3	0000001000	0011
4	0000010000	0100
5	0000100000	0101
6	0001000000	0110
7	0010000000	0111
8	0100000000	1000
9	1000000000	1001

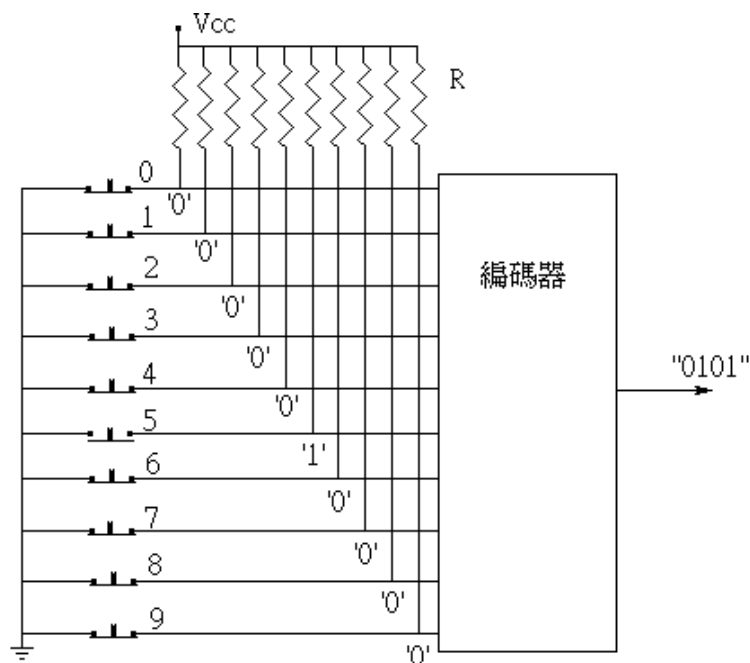


Figure 5.23 The application of key binary encoding after binary expression

Step 1 : Establish Truth table as Table 5.20 °

Table 5.20 Truth table of Encoder

Input	Output
$D_9 \sim D_0$	$O_3 \sim O_0$
0000000001	0000
0000000010	0001
0000000100	0010
0000001000	0011
0000010000	0100
0000100000	0101
0001000000	0110
0010000000	0111
0100000000	1000
1000000000	1001

Step 2 : To drive minimum Boolean expression from Truth table

$$\begin{aligned}
 O_0 = & D_9'D_8'D_7'D_6'D_5'D_4'D_3'D_2'D_1D_0' + D_9'D_8'D_7'D_6'D_5'D_4'D_3D_2'D_1'D_0' + \\
 & D_9'D_8'D_7'D_6'D_5D_4'D_3'D_2'D_1'D_0' + D_9'D_8'D_7D_6'D_5'D_4'D_3'D_2'D_1'D_0' + \\
 & D_9D_8'D_7'D_6'D_5'D_4'D_3'D_2'D_1'D_0' \dots\dots\dots (5-35)
 \end{aligned}$$

$$\begin{aligned}
 O_1 = & D_9'D_8'D_7'D_6'D_5'D_4'D_3'D_2D_1'D_0' + D_9'D_8'D_7'D_6'D_5'D_4'D_3D_2'D_1'D_0' + \\
 & D_9'D_8'D_7'D_6D_5'D_4'D_3'D_2'D_1'D_0' + D_9'D_8'D_7D_6'D_5'D_4'D_3'D_2'D_1'D_0' \\
 & \dots\dots\dots (5-36)
 \end{aligned}$$

$$\begin{aligned}
 O_2 = & D_9'D_8'D_7'D_6'D_5'D_4D_3'D_2'D_1'D_0' + D_9'D_8'D_7'D_6'D_5D_4'D_3'D_2'D_1'D_0' + \\
 & D_9'D_8'D_7'D_6D_5'D_4'D_3'D_2'D_1'D_0' + D_9'D_8'D_7D_6'D_5'D_4'D_3'D_2'D_1'D_0' \\
 & \dots\dots\dots (5-37)
 \end{aligned}$$

$$O_3 = D_9'D_8D_7'D_6'D_5'D_4'D_3'D_2'D_1'D_0' + D_9D_8'D_7'D_6'D_5'D_4'D_3'D_2'D_1'D_0' + \dots \quad (5-38)$$

Step 3 : Minimize Boolean expression possibly.

$$O_0 = (D_9'D_7'D_5'D_3'D_1 + D_9'D_7'D_5'D_3D_1' + D_9'D_7'D_5D_3'D_1' + D_9'D_7D_5'D_3'D_1' + D_9D_7'D_5'D_3'D_1') D_8'D_6'D_4'D_2'D_0' \quad (5-39)$$

$$O_1 = (D_7'D_6'D_3'D_2 + D_7'D_6'D_3D_2' + D_7'D_6D_3'D_2' + D_7D_6'D_3'D_2') D_9'D_8'D_5'D_4'D_1'D_0' \quad (5-40)$$

$$O_2 = (D_7'D_6'D_5'D_4 + D_7'D_6'D_5D_4' + D_7'D_6D_5'D_4' + D_7D_6'D_5'D_4') D_9'D_8'D_3'D_2'D_1'D_0' \quad (5-41)$$

$$O_3 = (D_9'D_8 + D_9D_8') D_7'D_6'D_5'D_4'D_3'D_2'D_1'D_0' \quad (5-42)$$

Step 4 : According to Boolean expression, in MAX + PLUS II, uses appropriate logic gate to complete circuit entries of Figure 5.24 by graphic editor.

Step 5 : Then simulation this circuit and check whether the functions meet the specification. Figure 5.25 is simulation result of Encoder.

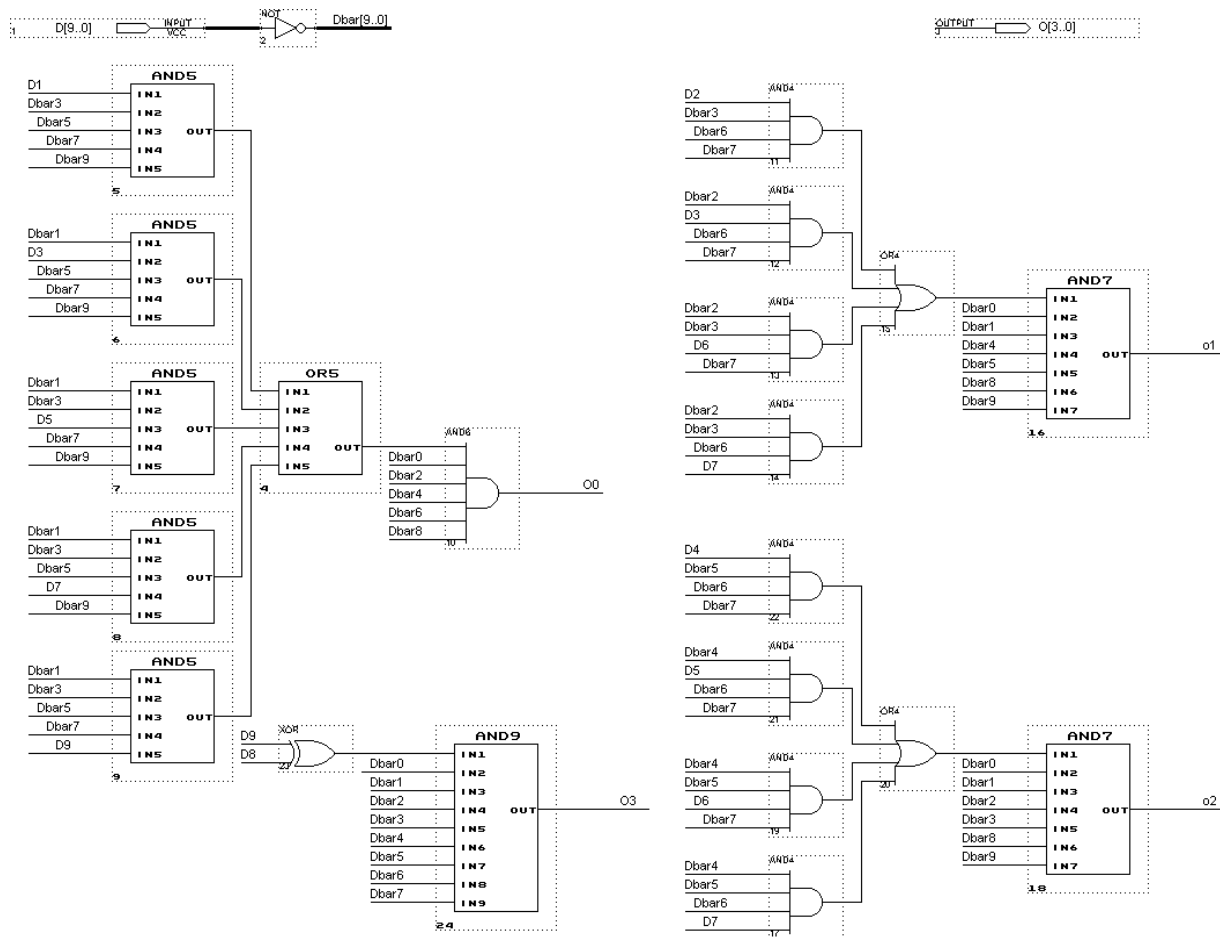


Figure5.24 Circuit of Encoder by using graphic entries in MAX + PLUS II
(document : encod10.gdf)




		100.0ns		200.0ns		300.0ns		400.0ns				
 D[9..0]	H 000	001	002	004	008	010	020	040	080	100	200	000
 Q[3..0]	H 0	0	1	2	3	4	5	6	7	8	9	0
 Dbar[9..0]	H 3FF	3FE	3FD	3FB	3F7	3EF	3DF	3BF	37F	2FF	1FF	3FF

Figure5.25 Simulation result of circuit encod10.gdf (document : encod10.scf)

Step 6 : After floorplan, download this circuit into selected device and performance the circuit test are needs. As circuit modified that showed in Figure 5.6 of

Section 5.2.1, please modify Encoder circuit of Figure 5.24. Please re-compile it after modifying, and adapt the floorplan techniques in Section 4.6, select chip EPF10K10TC144-4 and use Table 5.21 pin assignment reference. After assemble Lab platform LP-2900, download Encoder to chip EPF10K10TC144-4. Please try to push SW1 (D0), SW2 (D1)~SW10 (D9) on left-bottom of LP-2900, and please note the changes of L1 (O3) , L2 (O2), L3 (O1) and L4 (O0).

Table 5.21 Pin assignment of EPF10K10TC144-4

Name of Signal	Pin of EPF10K10TC144-4	Name of Signal	Pin of EPF10K10TC144-4
D0	Pin 47	D8	Pin 64
D1	Pin 48	D9	Pin 5
D2	Pin 49	O0	Pin 10
D3	Pin 51	O1	Pin 9
D4	Pin 59	O2	Pin 8
D5	Pin 60	O3	Pin 7
D6	Pin 62	LED_COM	Pin 141
D7	Pin 63		

5.6 The Design, Simulation and Test of Decoder

An n bit to 2^n Decoder can decode n bits to 2^n data. Figure 5.26 is an application decoder drives 16 LEDs. We see each LED connecting current-limit resistor and

showing common anode connected in Figure 5.26. So the Decoder output “0” can cause LED on and bright. Because it is input “0101”, the sixth LED is bright.

5.6.1 The Design, Simulation and Test of 4 to 16 Decoder

Figure 5.26 is a 4 to 16 Decoder. Please complete the design, simulation and test of this Decoder.

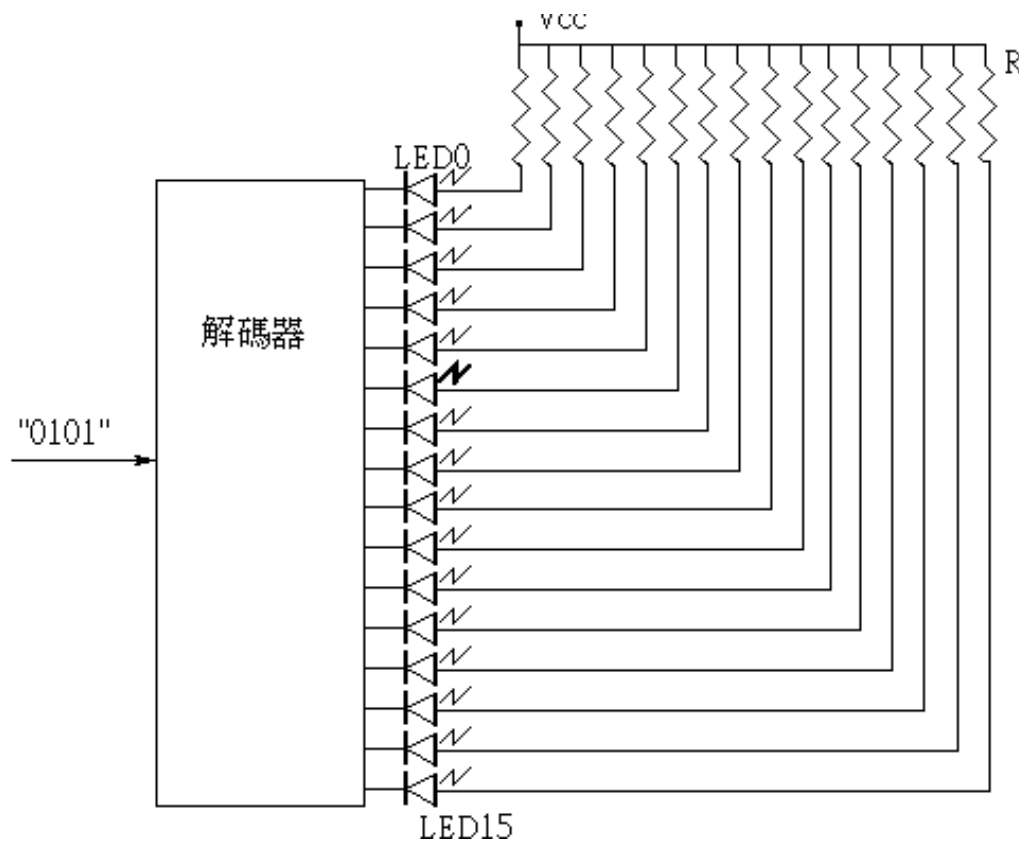


Figure 5.26 Four-bit encoder drives 16 LEDs

Step 1: Establish Truth table as Table 5.22

Step 2 : From Truth table of Table 5.22To drives Karnaugh map of Table 5.23a~5.23p.

Table 5.22 Truth table of Decoder

Input	Output
$D_3 \sim D_0$	$O_{15} \sim O_0$
0000	1111111111111110
0001	1111111111111101
0010	1111111111111011
0011	1111111111110111
0100	1111111111011111
0101	1111111111011111
0110	1111111110111111
0111	1111111101111111
1000	1111111011111111
1001	1111110111111111
1010	1111101111111111
1011	1111011111111111
1100	1110111111111111
1101	1101111111111111
1110	1011111111111111
1111	0111111111111111

Table 5.23a Karnaugh map of output O_0

O_0		$D_1 D_0$			
		00	01	11	10
$D_3 D_2$	00	0	1	1	1
	01	1	1	1	1
	11	1	1	1	1
	10	1	1	1	1

Table 5.23b Karnaugh map of output O_1

O_1		D_1D_0			
		00	01	11	10
D_3D_2	00	1	0	1	1
	01	1	1	1	1
	11	1	1	1	1
	10	1	1	1	1

Table 5.23c Karnaugh map of output O_2

O_2		D_1D_0			
		00	01	11	10
D_3D_2	00	1	1	1	0
	01	1	1	1	1
	11	1	1	1	1
	10	1	1	1	1

Table 5.23d Karnaugh map of output O_3

O_3		D_1D_0			
		00	01	11	10
D_3D_2	00	1	1	0	1
	01	1	1	1	1
	11	1	1	1	1
	10	1	1	1	1

Table 5.23e Karnaugh map of O_4

O_4		D_1D_0			
		00	01	11	10
D_3D_2	00	1	1	1	1
	01	0	1	1	1
	11	1	1	1	1
	10	1	1	1	1

Table 5.23f Karnaugh map of output O_5

O_5		D_1D_0			
		00	01	11	10
D_3D_2	00	1	1	1	1
	01	1	0	1	1
	11	1	1	1	1
	10	1	1	1	1

Table 5.23g Karnaugh map of output O_6

O_6		D_1D_0			
		00	01	11	10
D_3D_2	00	1	1	1	1
	01	1	1	1	0
	11	1	1	1	1
	10	1	1	1	1

Table 5.23h Karnaugh map of output O_7

O_7		D_1D_0			
		00	01	11	10
D_3D_2	00	1	1	1	1
	01	1	1	0	1
	11	1	1	1	1
	10	1	1	1	1

Table 5.23i Karnaugh map of output O_8

O_8		D_1D_0			
		00	01	11	10
D_3D_2	00	1	1	1	1
	01	1	1	1	1
	11	1	1	1	1
	10	0	1	1	1

Table 5.23j Karnaugh map of output O_9

O_9		D_1D_0			
		00	01	11	10
D_3D_2	00	1	1	1	1
	01	1	1	1	1
	11	1	1	1	1
	10	1	0	1	1

Table 5.23k Karnaugh map of output O_{10}

O_{10}		D_1D_0			
		00	01	11	10
D_3D_2	00	1	1	1	1
	01	1	1	1	1
	11	1	1	1	1
	10	1	1	1	0

Table 5.23L Karnaugh map of output O_{11}

O_{11}		D_1D_0			
		00	01	11	10
D_3D_2	00	1	1	1	1
	01	1	1	1	1
	11	1	1	1	1
	10	1	1	0	1

Table 5.23m Karnaugh map of output O_{12}

O_{12}		D_1D_0			
		00	01	11	10
D_3D_2	00	1	1	1	1
	01	1	1	1	1
	11	0	1	1	1
	10	1	1	1	1

Table 5.23n Karnaugh map of output O_{13}

O_{13}		D_1D_0			
		00	01	11	10
D_3D_2	00	1	1	1	1
	01	1	1	1	1
	11	1	0	1	1
	10	1	1	1	1

Table 5.23o Karnaugh map of output O_{14}

O_{14}		D_1D_0			
		00	01	11	10
D_3D_2	00	1	1	1	1
	01	1	1	1	1
	11	1	1	1	0
	10	1	1	1	1

Table 5.23p Karnaugh map of output O_{15}

O_{15}		D_1D_0			
		00	01	11	10
D_3D_2	00	1	1	1	1
	01	1	1	1	1
	11	1	1	0	1
	10	1	1	1	1

$$O_0 = D_1 + D_0 + D_3 + D_2 \dots\dots\dots (5-43)$$

$$O_1 = D_1 + D_0' + D_3 + D_2 \dots\dots\dots (5-44)$$

$$O_2 = D_1' + D_0 + D_3 + D_2 \dots\dots\dots (5-45)$$

$$O_3 = D_1' + D_0' + D_3 + D_2 \dots\dots\dots (5-46)$$

$$O_4 = D_1 + D_0 + D_3 + D_2' \dots\dots\dots (5-47)$$

$$O_5 = D_1 + D_0' + D_3 + D_2' \dots\dots\dots (5-48)$$

$$O_6 = D_1' + D_0 + D_3 + D_2' \dots\dots\dots (5-49)$$

$$O_7 = D_1' + D_0' + D_3 + D_2' \dots\dots\dots (5-50)$$

$$O_8 = D_0 + D_1 + D_2 + D_3' \dots\dots\dots (5-51)$$

$$O_9 = D_0' + D_1 + D_2 + D_3' \dots\dots\dots (5-52)$$

$$O_{10} = D_0' + D_1' + D_2 + D_3' \dots\dots\dots (5-53)$$

$$O_{11} = D_0' + D_1' + D_2 + D_3' \dots\dots\dots (5-54)$$

$$O_{12} = D_0 + D_1 + D_2' + D_3' \dots\dots\dots (5-55)$$

$$O_{13} = D_0' + D_1 + D_2' + D_3' \dots\dots\dots (5-56)$$

$$O_{14} = D_0' + D_1' + D_2' + D_3' \dots\dots\dots (5-57)$$

$$O_{15} = D_0' + D_1' + D_2' + D_3' \dots\dots\dots (5-58)$$

Step 3 : Minimize Boolean expression possibly. Because expression (5-43) and (5-58) had been minimized, they will not be minimized.

Step 4 : According to Boolean expression, in MAX+PLUS II, uses appropriate logic gate to complete circuit entries of Figure 5.27 by graphic editor.

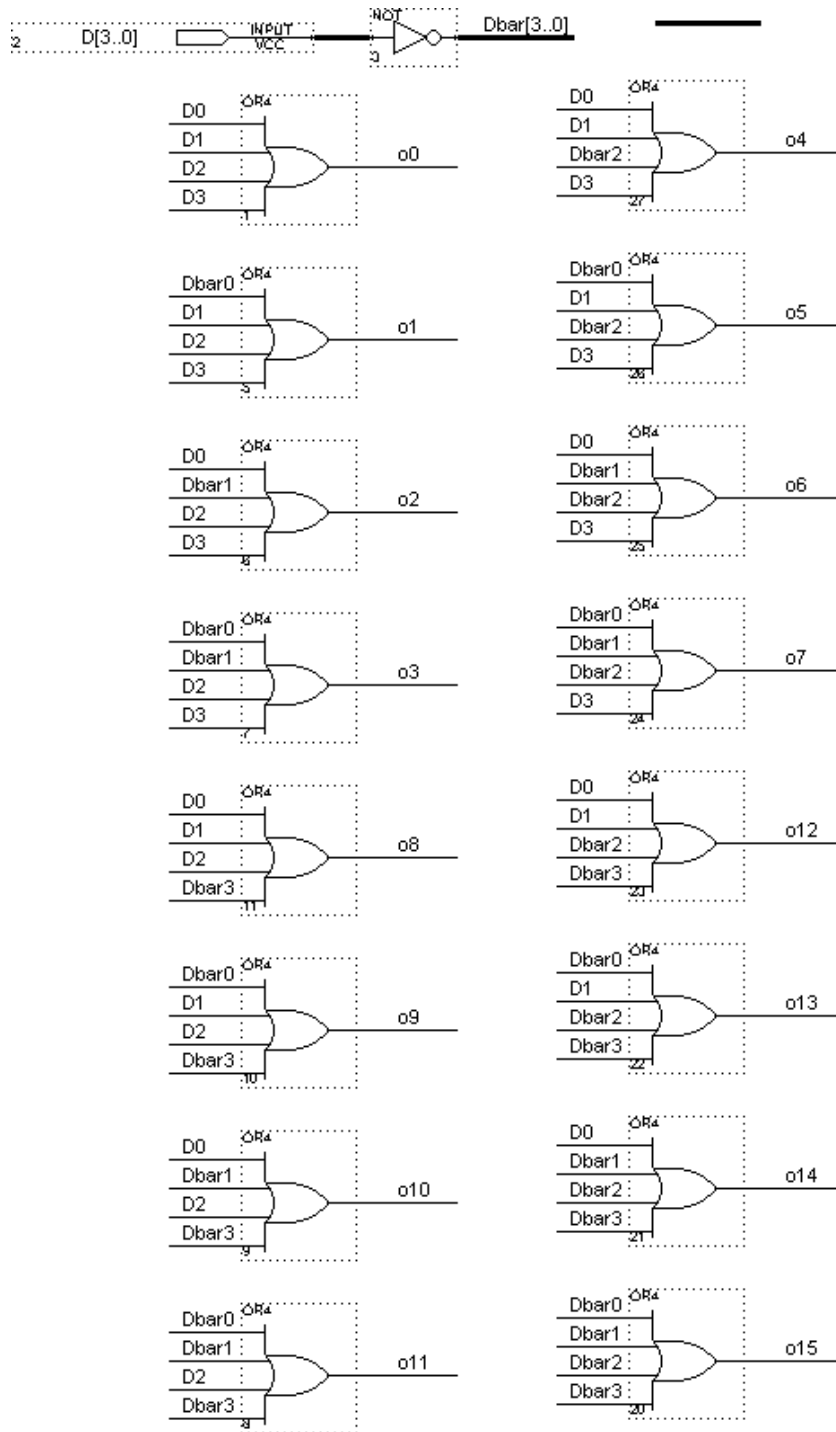


Figure 5.27 Circuit of Decoder by using graphic entries in MAX+PLUS II
(document : dec4x16.gdf)

Step 5 : Then simulation this circuit and check whether the functions meet the specification. Figure 5.28 is simulation result of Decoder.

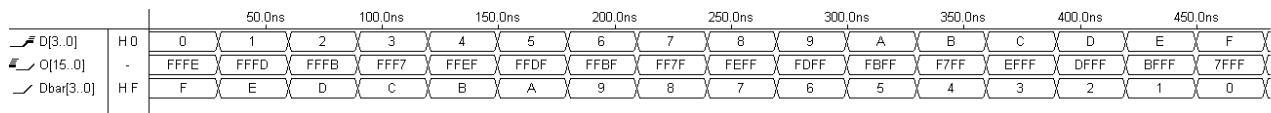


Figure 5.28 Simulation result of circuit dec4x16.gdf (document : dec4x16.scf)

Step 6 : After floorplan, download this circuit into selected device and performance the circuit test are needs. As circuit modified that showed in Figure 5.6 of Section 5.2.1, please modify Decoder circuit of Figure 5.27. Please re-compile it after modifying, and adapt the floorplan techniques in Section 4.6, select chip EPF10K10TC144-4 and use Table 5.24 pin assignment reference. After assemble Lab platform LP-2900, download this 4 to 16 Decoder to chip EPF10K10TC144-4. Please try to push SW1 (D_0), SW2 (D_1), SW3 (D_2) and SW4 (D_3) on left-bottom of LP-2900, and please note the change of L1 (O_{11})、L2 (O_{10})、L3 (O_9)、 \cdots L16 (O_0). For example, you can input “0110” at $D_3 \sim D_0$ and see if O_6 is bright.

Table5.24 Pin assignment of EPF10K10TC144-4

Name of Signal	Pin of EPF10K10TC144-4	Name of Signal	Pin of EPF10K10TC144-4
D ₀	Pin 47	O ₇	Pin 11
D ₁	Pin 48	O ₈	Pin 10
D ₂	Pin 49	O ₉	Pin 9
D ₃	Pin 51	O ₁₀	Pin 8
O ₀	Pin 20	O ₁₁	Pin 7
O ₁	Pin 19		
O ₂	Pin 18	LED_COM	Pin 141
O ₃	Pin 17		
O ₄	Pin 14		
O ₅	Pin 13		
O ₆	Pin 12		

Note: In O₁₅~O₁₂, there are no output pin assignment

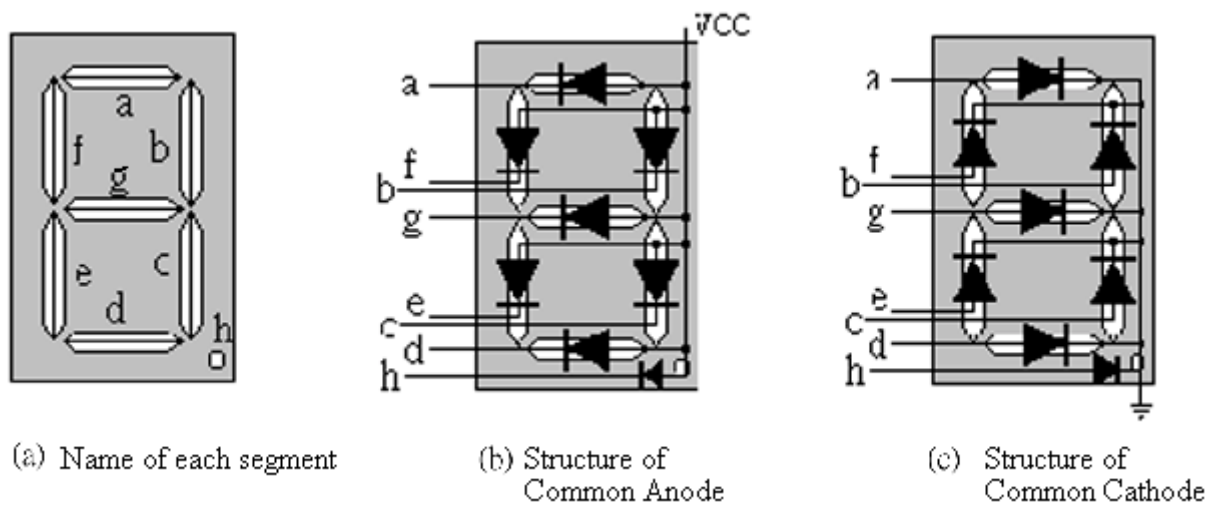
5.6.2 The Design, Simulation and Test of BCD to 7 Segment Display

● The Principle of 7-segment Display

7-segment display constitutes by seven rectangle LED as Figure 5.29(a), but some right-bottom of 7-segment displays have one circle LED. In digital circuit, 7-segment display is passive component and use frequently. 7-segment display can be divided into common anode structure (Figure 5.29(b)) and common cathode structure (Figure 5.29(c)). If we use common anode 7-segment display, we to

connect common anode to VCC and need input “0”, then corresponding rectangle LED will on and bright. The bright electric current is supplied by Vcc. Contrary, If we use common cathode 7-segment display, we need to connect common cathode to ground and input “1”, then corresponding rectangle LED will be on and bright. The bright electric current is supplied by input signal. Base on this bright principle, in common anode 7-segment display, it will show “0” character if inputs “11000000B” ; it will show “1” character if inputs “11111001B” ; it will show “2” character if inputs “10100100B” ... so on and so forth.

Figure 5.29 Exterior and category of 7-segment display



Similarly, in common cathode 7-segment display, it will show “0” character if inputs “00111111B” ; it will show “1” character if inputs “00000110B” ; it will show “2” character if inputs “01011011”... so on and so forth.

After understanding bright principle of 7-segment display, we want to design the Decoder circuit of common anode 7-segment display with value 0 to 9.

Step 1: Establish Truth table of 7-segment display Decoder with value 0 to 9 as

Table 5.25

Common Anode : 11000000B	11111001B	10100100B	10110000B	10011001B
Common Cathode : 00111111B	00000110B	01011011B	01001111B	01100110B
Common Anode : 10010010B	10000010B	11111000B	10000000B	10010000B
Common Cathode : 01101101B	01111101B	00000111B	01111111B	01101111B

Figure 5.30 Decoder codes of “0~9” for 7-segment display

Table 5.25 Truth table of 7-segment display Decoder with value 0 to 9

Input	Output
$D_3 \sim D_0$	hgfedcba
0000	11000000
0001	11111001
0010	10100100
0011	10110000
0100	10011001
0101	10010010
0110	10000010
0111	11111000
1000	10000000
1001	10010000

Step 2 : From Truth table of Table 5.25 drives Karnaugh map of Table 5.26a~5.26h.

Table 5.26a Karnaugh map of output a

a		$D_1 D_0$			
		00	01	11	10
$D_3 D_2$	00	0	1	0	0
	01	1	0	0	0
	11	0	0	0	0
	10	0	0	0	0

Table 5.26b Karnaugh map of output b

b		D_1D_0			
		00	01	11	10
D_3D_2	00	0	0	0	0
	01	0	1	0	1
	11	0	0	0	0
	10	0	0	0	0

Table 5.26c Karnaugh map of output c

c		D_1D_0			
		00	01	11	10
D_3D_2	00	0	0	0	1
	01	0	0	0	0
	11	0	0	0	0
	10	0	0	0	0

Table 5.26d Karnaugh map of output d

d		D_1D_0			
		00	01	11	10
D_3D_2	00	0	1	0	0
	01	1	0	1	0
	11	0	0	0	0
	10	0	0	0	0

Table 5.26e Karnaugh map of output e

e		D_1D_0			
		00	01	11	10
D_3D_2	00	0	1	1	0
	01	1	1	1	0
	11	0	0	0	0
	10	0	1	0	0

Table 5.26f Karnaugh map of output f

f		D_1D_0			
		00	01	11	10
D_3D_2	00	0	1	1	1
	01	0	0	1	0
	11	0	0	0	0
	10	0	0	0	0

Table 5.26g Karnaugh map of output g

g		D_1D_0			
		00	01	11	10
D_3D_2	00	1	0	0	0
	01	0	0	0	0
	11	0	0	0	0
	10	0	0	0	0

Table 5.26h Karnaugh map of output h

h		D ₁ D ₀			
		00	01	11	10
D ₃ D ₂	00	1	1	1	1
	01	1	1	1	1
	11	1	1	1	1
	10	1	1	1	1

$$a = D_3'D_2D_1'D_0' + D_3'D_2'D_1'D_0 \text{ -----(5-59)}$$

$$b = D_3'D_2D_1'D_0 + D_3'D_2D_1D_0' \text{ ----- (5-60)}$$

$$c = D_3'D_2'D_1D_0' \text{ -----(5-61)}$$

$$d = D_3'D_2'D_1'D_0 + D_3'D_2D_1'D_0' + D_3'D_2D_1D_0 \text{ -----(5-62)}$$

$$e = D_3'D_0 + D_2'D_1'D_0 + D_3'D_2D_1' \text{ -----(5-63)}$$

$$f = D_3'D_2'D_0 + D_3'D_2'D_1 + D_3'D_1D_0 \text{ -----(5-64)}$$

$$g = D_3'D_2'D_1' + D_3'D_2D_1D_0 \text{ -----(5-65)}$$

$$h = V_{cc} \text{ -----(5-66)}$$

Step 3 : Minimize Boolean expression possibly. Because the expression (5-59) and (5-66) had been minimized, they will not be minimized.

Step 4 : According to Boolean expression, in MAX+PLUS II, uses appropriate logic gate to complete circuit entries of Figure 5.31 by graphic editor.

Step 5 : Then simulation this circuit and check whether the functions meet the specification. Figure 5.32 is simulation result of BCD to 7-segment display.

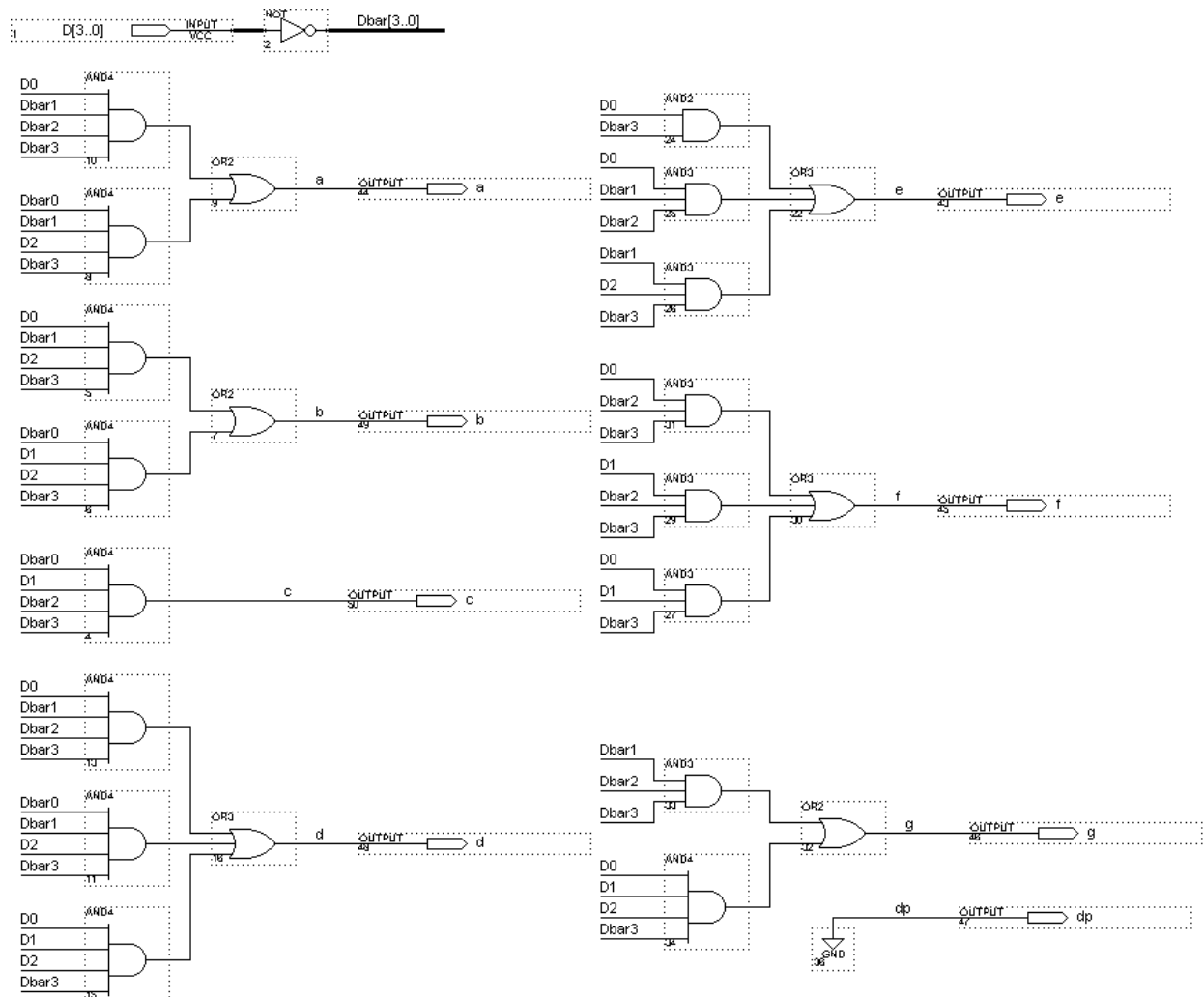


Figure 5.31 Circuit of Decoder by using graphic entries in MAX + PLUS II
(document : seg7dec.gdf)

		50.0ns	100.0ns	150.0ns	200.0ns	250.0ns	300.0ns	350.0ns	
D[3..0]	H 6	0	1	2	3	4	5	6	7
seg[7..0]	-	11000000	11111100	10100100	10110000	10011001	10010010	10000010	11111000
Dbar[3..0]	H 9	F	E	D	C	B	A	9	8

Figure 5.32 Simulation result of circuit seg7dec.gdf (document : seg7dec.scf)

Step 6: After floorplan, download this circuit into selected device and performance the circuit test are needs. Because 7-segment display on LP-2900 is common cathode (but the decoder is designed for common anode 7-segment display), we need to modify this circuit as Figure 5.33. Besides, in figure be showed excess three output signal lines 74138_DE1, 74138_DE2 and 74138_DE3, they are signal of driving 74138(3 to 8 Decoder), and select one Y of eight output “0”. Y0~Y5 connect to common cathode of six 7-segment display(C1~C6). Therefore, if Y1 outputs “0” (74138_DE [1..3] output is “001”), the data of a, b, c, ...and dp will be lead to second 7-segment display. If Y3 outputs “0” (74138_DE [1..3] output is “0011”), the data of a, b, c, ...and dp will be lead to fourth 7-segment display...so on and so forth. Please re-compile it after modifying, and adapt the floorplan techniques in Section 4.6, select chip EPF10K10TC144-4 and use Table 5.27 pin assignment reference. After assemble Lab platform LP-2900, download 7-segment display decoder to chip EPF10K10TC144-4. Please try to push SW1 (D₀), SW2 (D₁), SW3 (D₂) and SW4 (D₃) on left-bottom of LP-2900, and please note the changes of 7-segment display and note its showing location. Like, when D₃~D₀ input is “0110” and if “6” is bright.

Table 5.27 Pin assignment of EPF10K10TC144-4

Name of Signal	Pin of EPF10K10TC144-4	Name of Signal	Pin of EPF10K10TC144-4
D0	Pin 47	A	Pin 23
D1	Pin 48	B	Pin 26
D2	Pin 49	C	Pin 27
D3	Pin 51	D	Pin 28
		E	Pin 29
		F	Pin 30
		G	Pin 31
		H	Pin 32
		74138_DE1	Pin 33
		74138_DE2	Pin 36
		74138_DE3	Pin 37

5.7 The Design, Simulation and Test of MUX

The multiplexer is called data selector, following left of Figure 5.34. The figure illustrated that select one of 2^n lines to Y output by n lines. The data lines which not be selected is not be output. Usually, MUX is defined by 2^n to 1.

● The design, simulation and test of 8 to 1 MUX

In this section, we perform the design of 8 to 1 MUX. The selection line includes three due to 8 equal to 2^3 . The specification of 8 to 1 MUX is: “we choose D_0 output to Y when $S_2S_1S_0 = “000”$; we choose D_1 output to Y when $S_2S_1S_0 =$

“001” ; we choose D_2 output to Y when $S_2S_1S_0 = “010”$; we choose D_3 output to Y when $S_2S_1S_0 = “011”$; we choose D_4 output to Y when $S_2S_1S_0 = “100”$; we choose D_5 output to Y when $S_2S_1S_0 = “101”$; we choose D_6 output to Y when $S_2S_1S_0 = “110”$; we choose D_7 output to Y when $S_2S_1S_0 = “111”$.”

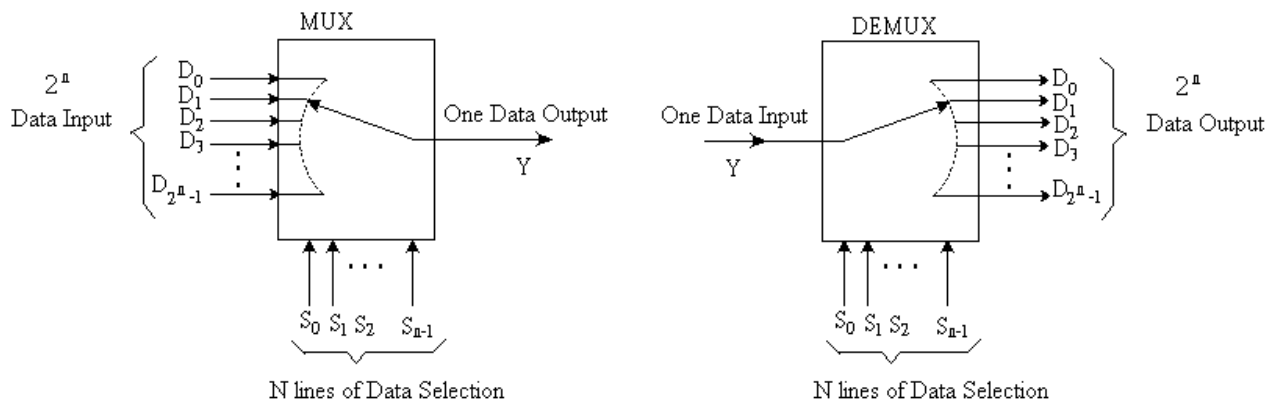


Figure 5.34 Functional diagrams of MUX and DMUX

Step 1: Establish Truth table of MUX as Table 5.28.

Step 2 : From Truth table of Table 5.28 drives Boolean expression of Y .

$$Y = S_2'S_1'S_0'D_0 + S_2'S_1'S_0'D_1 + S_2'S_1S_0'D_2 + S_2'S_1S_0'D_3 + S_2S_1'S_0'D_4 + S_2S_1'S_0'D_5 + S_2S_1S_0'D_6 + S_2S_1S_0'D_7 \dots\dots\dots (5-67)$$

Step 3 : Minimize Boolean expression possibly. Because the expression (5-67) had been minimized, they will not be minimized.

Step 4 : According to Boolean expression, in MAX+PLUS II, uses appropriate logic gate to complete circuit entries of Figure 5.35 by graphic editor.

Step 5 : Then simulation this circuit and check whether the functions meet the specification. Figure5.36 is simulation result of 8 to 1 MUX.

Table 5.28 Truth table of MUX

Input	Output
$S_2S_1S_0$	Y
000	D_0
001	D_1
010	D_2
011	D_3
100	D_4
101	D_5
110	D_6
111	D_7

Step 6 : After floorplan, download this circuit into selected device and performance the circuit test are needs. As circuit modified that showed in Figure 5.6 of Section 5.2.1, please modify MUX circuit of Figure 5.35. Please re-compile it after modifying, and adapt the floorplan techniques in Section 4.6, select chip EPF10K10TC144-4 and use Table 5.29 pin assignment reference. After assemble Lab platform LP-2900, download 8 to 1 MUX to chip EPF10K10TC144-4. Please try to push SW1 (D_0), SW2 (D_1), SW3 (D_2),... and SW8 (D_7) on left-bottom of LP-2900, and push switch SW9~SW11 then observe how L1 (Y) react $D_0 \sim D_7$.

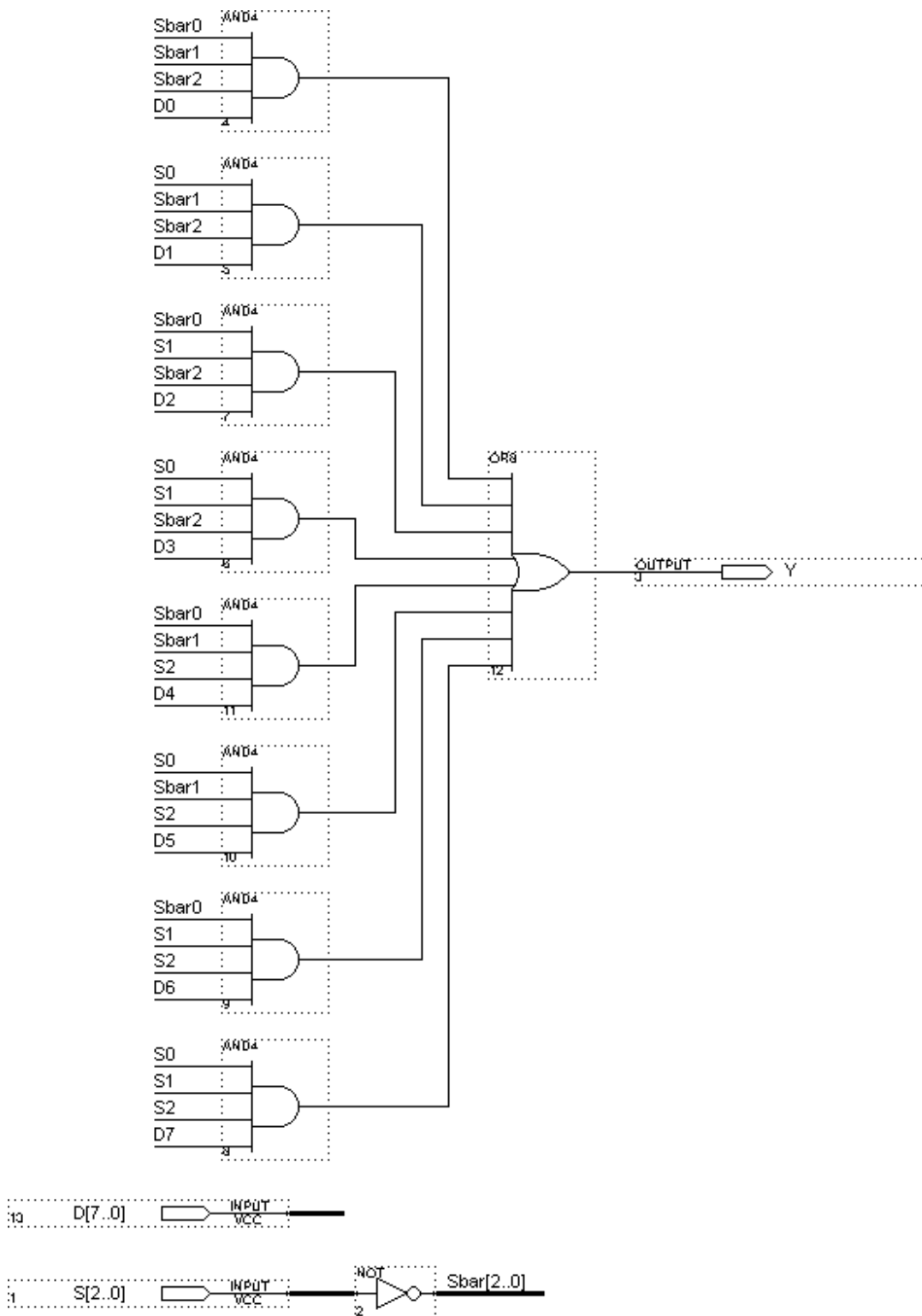


Figure 5.35 Circuit of MUX by using graphic entries in MAX+PLUS II
(document : mux81.gdf)

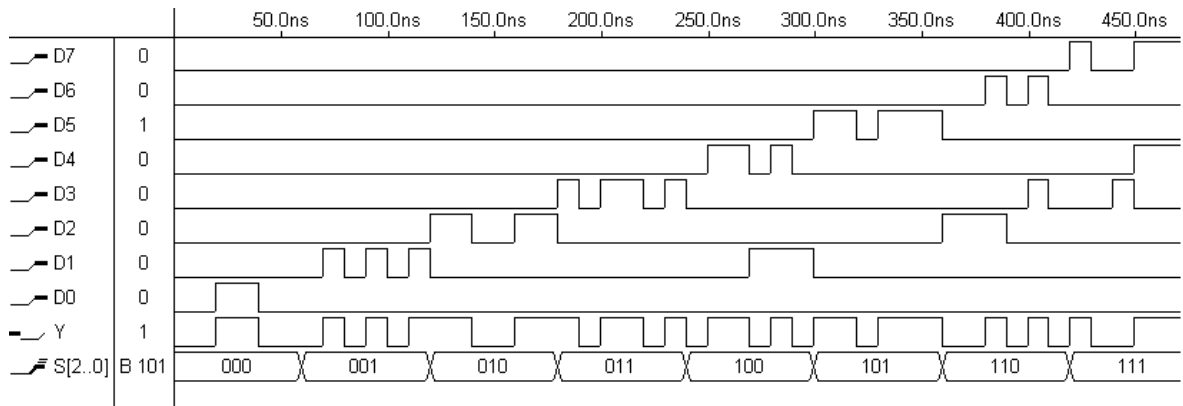


Figure 5.36 Simulation result of circuit mux81.gdf (document : mux81.scf)

Table 5.29 Pin assignment of EPF10K10TC144-4

Name of Signal	Pin of EPF10K10TC144-4	Name of Signal	Pin of EPF10K10TC144-4
S0	Pin 67	D0	Pin 47
S1	Pin 65	D1	Pin 48
S2	Pin 64	D2	Pin 49
		D3	Pin 51
		D4	Pin 59
		D5	Pin 60
		D6	Pin 62
		D7	Pin 63
		Y	Pin 7
		LED_COM	Pin 141

5.8 The Design, Simulation and Test of DMUX

The demultiplexer is called data distributor, right of Figure 5.34. The figure illustrated that Y will be distributed to one of 2^n line by n lines. The data lines which not be selected is not be output. Usually, DMUX defined by 1 to 2^n .

❖ The design, simulation and test of 1 to 8 DMUX

In this section, we perform the design of 1 to 8 DMUX. The choice line includes three due to 8 equals to 2^3 . The specification of 1 to 8 DMUX is: “ D output to Y_0 when $S_2S_1S_0 = “000”$; we choose D output to Y_1 when $S_2S_1S_0 = “001”$; we choose D output to Y_2 when $S_2S_1S_0 = “010”$; we choose D output to Y_3 when $S_2S_1S_0 = “011”$; we choose D output to Y_4 when $S_2S_1S_0 = “100”$; we choose D output to Y_5 when $S_2S_1S_0 = “101”$; we choose D output to Y_6 when $S_2S_1S_0 = “110”$; we choose D output to Y_7 when $S_2S_1S_0 = “111”$.”

Step 1: Establish Truth table of DMUX as Table 5.30.

Step 2 : From Truth table of Table 5.34 drives Boolean expression.

$$Y_0 = DS_2'S_1'S_0' \dots\dots\dots (5-68)$$

$$Y_1 = DS_2'S_1'S_0 \dots\dots\dots (5-69)$$

$$Y_2 = DS_2'S_1S_0' \dots\dots\dots (5-70)$$

$$Y_3 = DS_2'S_1S_0 \dots\dots\dots (5-71)$$

$$Y_4 = DS_2S_1'S_0' \dots\dots\dots (5-72)$$

$$Y_5 = DS_2S_1'S_0 \dots\dots\dots (5-73)$$

$$Y_6 = DS_2S_1S_0' \dots\dots\dots (5-74)$$

$$Y_7 = DS_2S_1S_0 \dots\dots\dots (5-75)$$

Table 5.30 Truth table of DMUX (D is data of input end)

Input	Output
$S_2 S_1 S_0$	$Y_7 Y_6 Y_5 Y_4 Y_3 Y_2 Y_1 Y_0$
000	0000000D
001	000000D0
010	00000D00
011	0000D000
100	000D0000
101	00D00000
110	0D000000
111	D0000000

Step 3 : Minimize Boolean expression possibly. Because the expression (5-68) and (5-75) had been minimized, they will not be minimized.

Step 4 : According to Boolean expression, in MAX+PLUS II, uses appropriate logic gate to complete circuit entries of Figure 5.37 by graphic editor.

Step 5 : Then simulation this circuit and check whether the functions meet the specification. Figure5.38 is simulation result of DMUX (please note each cycle)

Step 6 : After floorplan, download this circuit into selected device and performance the circuit test are needs. As circuit modified that showed in Figure 5.6 of Section 5.2.1, please modify DMUX circuit of Figure 5.37. Please re-compile it after modifying, and adapt the floorplan techniques in Section 4.6, select chip EPF10K10TC144-4 and use Table 5.31 pin assignment reference. After assemble Lab platform LP-2900, download 1 to 8 DMUX

to chip EPF10K10TC144-4. Please try to push SW1 (D0) on left-bottom of LP-2900, and push switch SW9~SW11 then observe change of L1 (Y0), L2 (Y1), L3 (Y2)····and L8 (Y7)

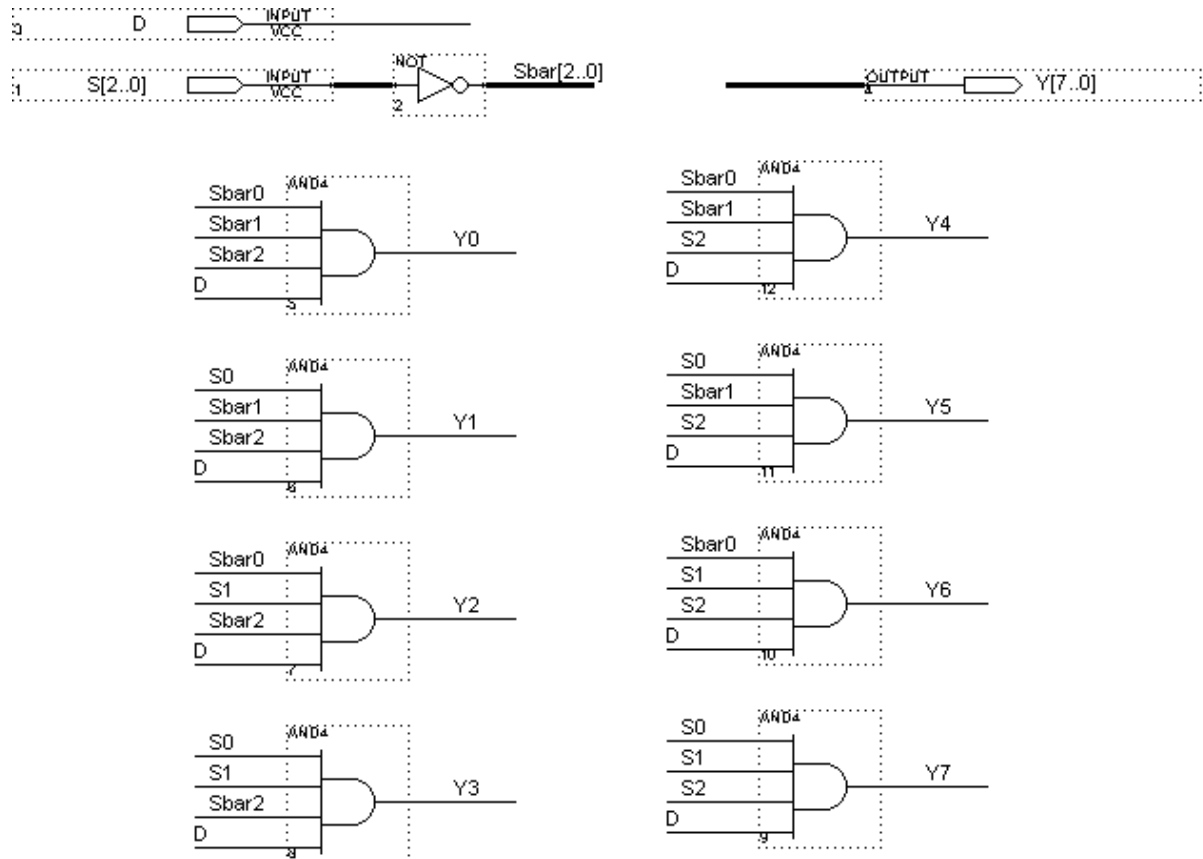


Figure 5.37 Circuit of DMUX by using graphic entries in MAX+PLUS II
(document : dmux18.gdf)

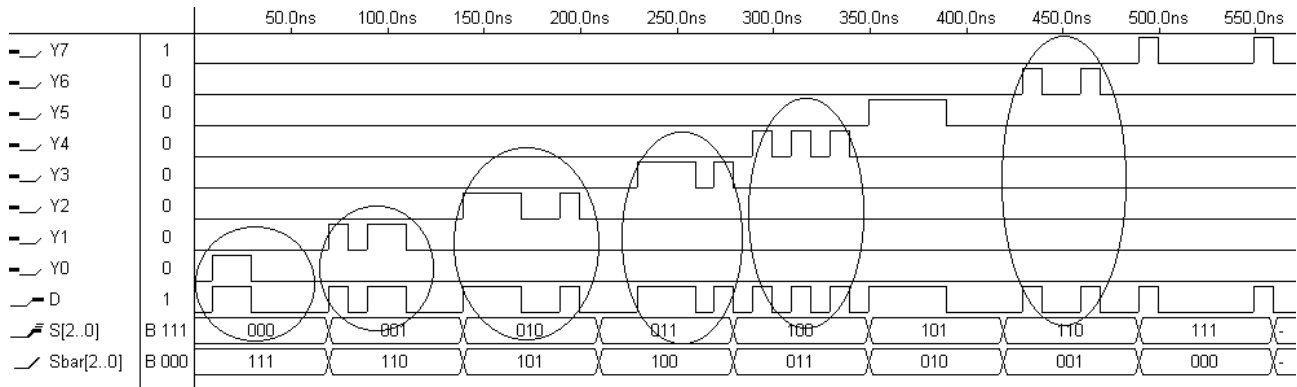


Figure 5.38 Simulation result of circuit dmux18.gdf (document : dmux18.scf)

Table 5.31 Pin assignment of EPF10K10TC144-4

Name of Signal	Pin of EPF10K10TC144-4	Name of Signal	Pin of EPF10K10TC144-4
S0	Pin 67	D	Pin 47
S1	Pin 65	Y0	Pin 7
S2	Pin 64	Y1	Pin 8
		Y2	Pin 9
		Y3	Pin 10
		Y4	Pin 11
		Y5	Pin 12
		Y6	Pin 13
		Y7	Pin 14
		LED_COM	Pin 141

5.9 The Question of Hazards

The logic signal elapses one circuit that depends on the propagation delay, and there is high unascertained in propagation delay. The gate delay is different following

different logic system. Even there is different logic and different propagation delay in same logic system. It is possible that gate delay along different propagation path and there is different delay when signal changing. Because of different path delay possible causes temporary (or quick) pulsation in logic circuit. This temporary pulsation is called hazards, like in Figure 5.39(a). In Figure 5.39(a), for example, one input signal of OR gate becomes low from high and the other one input signal becomes high from low. Since the propagation delay is not equal, it will possible hide Hazards occurrence (gray area). In Figure 5.39(b), if above OR gate input signal is early below input signal, the Hazards will not show under this situation. On the other hand, in the Figure 5.39(c), if above OR gate input signal is late below input signal, the Hazards will be showed under this situation. The output should keep static 1, but shows fast low hazards which we call static 1-hazards. On the other hand, for an AND gate, it possible will show static 0-hazards.

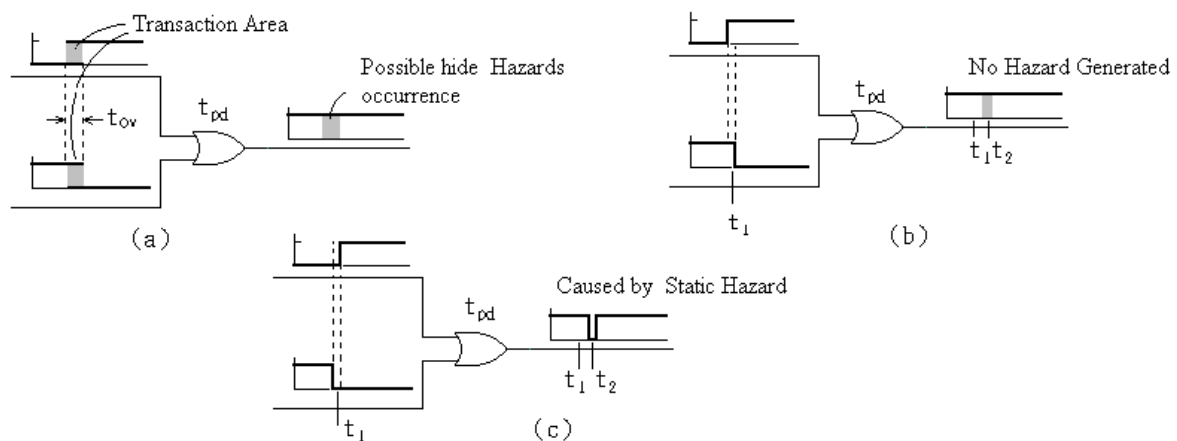


Figure 5.39 Static1-hazards

The hazards include static hazards and dynamic hazards. Figure 5.40 is illustrated the means and form reason of dynamic. The $X_2(t)$ changes from high to low and pass different path (causing different propagation delay), than the G5 output forms static1-hazards. The G5 inputs which one pass two gates delay and another pass one

gate delay. Basically, the signal of two gates delay pass more belatedly than signal of one gate delay. For G7 inputs which one pass three gates delay and another pass two gates delay. It is obviously that the G7 below input signal is early than above signal, than it will cause dynamic hazards showing on G7 output. Why it be called dynamic hazards? That's when G7 output is becoming low to high, happens the hazards. Similarly, when it is becoming high to low, happens the hazards. It also is called dynamic hazards.

Hazards, main form reason is by different signal propagation delay path, so there is high unascertained. If the width of Hazards is smaller than inertial delay of logic gate, it is not occurrence. But it is not good way to obstruct hazard by inertial delay. Controlled and prevention of hazards is an important concern. We will not introduce in this book, please reader consult reference.

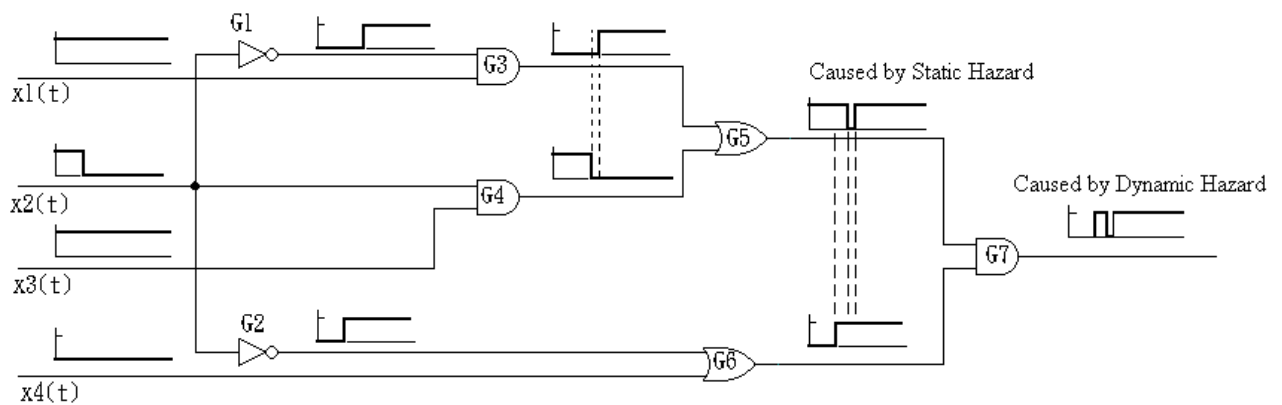


Figure 5.40 Dynamic hazards



5.10 Evaluations

Please do the following evaluation according to the questions listed below

- ☐ Do you know what does combinational logic circuit define?
- ☐ Do you know step of design, simulation and test of general combination logic circuit?
- ☐ Do you know design principle of Look-ahead Carry Adder?
- ☐ Do you know we can design addition and subtractor together by using principle of 2's complement?
- ☐ Can you design, simulate and test a 3 to 8 Decoder?
- ☐ Can you design, simulate and test 1 to 4 DMUX?
- ☐ Can you explain meaning of Dynamic Hazard and form cause?

CHAPTER 6

Sequential Logic Circuit



LEAP

In this chapter, we will introduce sequential logic circuit. Except basic concept of sequential logic, we also introduce the design, simulation and test of synchronous counter, synchronous shift-register, synchronous shift count register, and asynchronous counter.

6.1 Basic Concept of Sequential Logic Circuit

In previous chapter, we already mentioned that combinational logic circuit output only related to input signal. Once the input signal changes, the combinational logic circuit output changes at once. It will react the output of previous item input, and disappears at once. In other words, there is not memory ability of combinational logic. Therefore, if circuit output not only relate with immediately input but also previous output, which we call sequential logic circuit. The Figure 6.1 is the model of general sequential logic circuit. Basically, the combinational logic circuit and memory cell constructs the sequential logic circuit. The circuit combinational logic receives two input signals, which come from input of periphery circuit and memory cell. “The input from memory cell” recodes present state. The combinational logic circuit includes two output signals, which outputs to periphery circuit and memory cell. “Output to memory cell” is next state.

6.1.1 The Synchronous and Asynchronous Operation

The sequential logic can divide into category of synchronous sequential logic and category of asynchronous sequential logic. In synchronous sequential logic, the change of internal state is controlled by synchronous clock. This clock usual is pulse (as Figure 6.2). The On-time stands for logic “1”, and Off-time stands for logic “0”. The On-time and Off-time are just clock period. The clock duty can be defined as follows

Duty cyclic= on time/clock period

In the clock, we call positive edge or rising edge when signal changes from "0" to "1". Relatively, we call negative edge or falling edge when signal changes from "1" to "0".

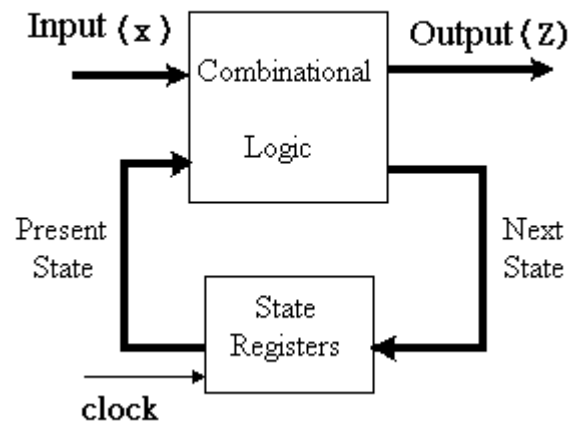


Figure 6.1 Model of general sequential logic

In the synchronous model, the logic circuit must coordinate trigger of clock signal. Only changes output state when clock trigger, then this state will keep until next trigger coming. In the asynchronous model, logic circuit can change output signal anytime. According to category of synchronous and category of asynchronous, we can mention asynchronous sequential logic circuit model of Figure 6.3 and synchronous sequential logic circuit model of Figure 6.4. In the Figure, thick lines stand for most signal lines.

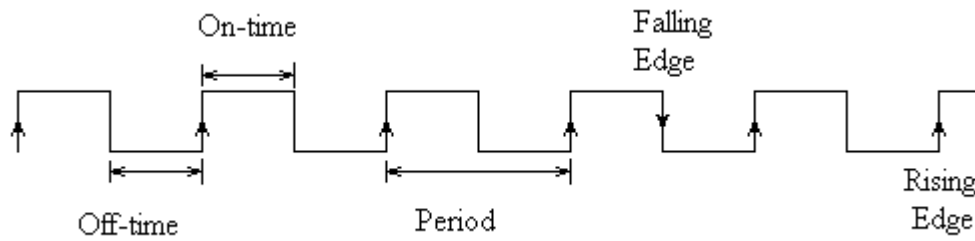


Figure 6.2 Terminologies of clock

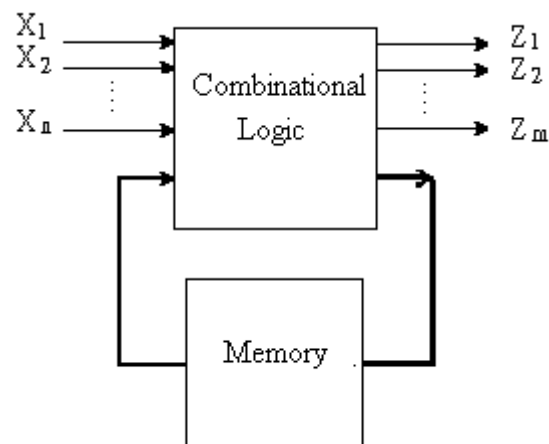
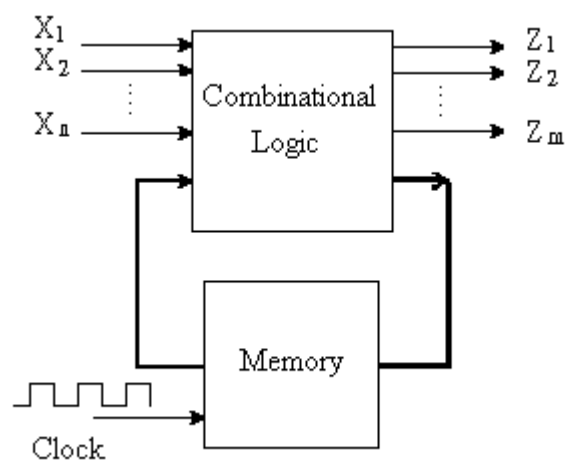


Figure 6.3 Model of asynchronous sequential logic

Figure 6.4 Model of synchronous sequential logic



Because asynchronous sequential circuit is not common synchronous timing signal, it will happen some problems (like, racing and hazards) in design and dependence. Those problems will lead to asynchronous sequential circuit limits in use of a few special circuits (as counter). So sequential circuit almost are synchronous sequential circuit. In this book, the introduced circuit in Section 6.2, 6.3 and 6.4 are synchronous, but circuit in section 6.5 is asynchronous circuit.

6.1.2 The Latch and Flip-flops

The sequential circuit, usual takes Flip-flops as memory cell. SR Latch is basic of other Flip-flops. Flip-flops can be divided into category of edge trigger Flip-flops and category of gate controlled Flip-flops. Relative to edge trigger model, gate controlled Flip-flops uses level trigger. The edge trigger Flip-flops includes D type Flip-flops, JK Flip-flops and T type Flip-flops. The gate controlled Flip-flops includes D type Flip-flops, SR Flip-flops, JK Flip-flops and T type Flip-flops. We will introduce their functions following.

1. SR Latch

Generally, SR Latch can be made by two NOR gates or two NAND gates. In the Figure 6.5, left is logic circuit of SR Latch making up by two NOR gates, the middle is logic circuit of SR Latch making up by two NAND gates and right is the circuit symbol. S is input end of setting signal, R is input end of resetting signal, Q is output and Q' is complement output of Q.

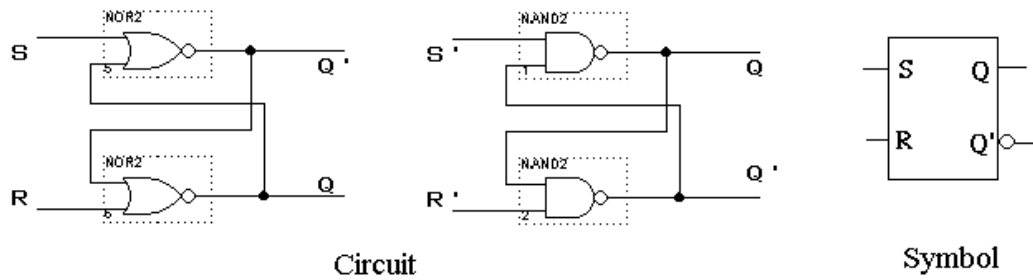


Figure 6.5 Logic circuit diagrams and circuit signal of SR Latch

❖ The operation principle of SR Latch

If time is t , $Q=R=S=0$, and S becomes 1 from 0

$$Q'(t + t_{pd}) = (S(t) + Q(t))' = (1 + 0)' = 0$$

$$Q(t + 2t_{pd}) = (R(t) + Q'(t + t_{pd}))' = (0 + 0)' = 1$$

$$Q'(t + 3t_{pd}) = (S(t) + Q(t + 2t_{pd}))' = (1 + 1)' = 0$$

$$Q(t + 4t_{pd}) = (R(t) + Q'(t + 3t_{pd}))' = (0 + 0)' = 1$$

...

So SR Latch changes state and new state firm to $(S, R) = (1, 0)$. If S again becomes 0 from 1 at time t_a , the outputs are as follows

$$Q'(t_a + t_{pd}) = (S(t_a) + Q(t_a))' = (0 + 1)' = 0$$

$$Q(t_a + 2t_{pd}) = (R(t_a) + Q'(t_a + t_{pd}))' = (0 + 0)' = 1$$

$$Q'(t_a + 3t_{pd}) = (S(t_a) + Q(t_a + 2t_{pd}))' = (0 + 1)' = 0$$

$$Q(t_a + 4t_{pd}) = (R(t_a) + Q'(t_a + 3t_{pd}))' = (0 + 0)' = 1$$

...

We can see SR Latch not changing state and old state still firm to $(S, R) = (0, 0)$.

Similarly, if time is t , $Q = R = S = 0$ and R become 1 from 0.

$$Q(t + t_{pd}) = (R(t) + Q'(t))' = (1 + 1)' = 0$$

$$Q'(t + 2t_{pd}) = (S(t) + Q(t + t_{pd}))' = (0 + 0)' = 1$$

So SR Latch changes state and new state firm to $(S, R) = (0, 1)$. If R again becomes 0 from 1 at time t_a , the outputs are as follows:

$$Q(t_a + t_{pd}) = (R(t_a) + Q'(t_a))' = (0 + 1)' = 0$$

$$Q'(t_a + 2 t_{pd}) = (S(t_a) + Q(t_a + t_{pd}))' = (0 + 0)' = 1$$

We can see SR Latch not changing state and old state still firm to $(S, R) = (0, 0)$

Similarly, if time is t , $Q = R = S = 0$ and R become 1 from 0.

$$Q'(t + t_{pd}) = (S(t) + Q(t))' = (1 + 0)' = 0$$

$$Q(t + t_{pd}) = (R(t) + Q'(t))' = (1 + 1)' = 0$$

$$Q'(t + 2 t_{pd}) = (S(t) + Q(t + t_{pd}))' = (1 + 0)' = 0$$

$$Q(t + 2 t_{pd}) = (R(t) + Q'(t + t_{pd}))' = (1 + 0)' = 0$$

So SR Latch changes state and new state firm to $(Q, Q') = (0, 0)$. If S and R again become 0 from 1 at time t_a , the outputs are as follows

$$Q'(t_a + t_{pd}) = (S(t_a) + Q(t_a))' = (0 + 0)' = 1$$

$$Q(t_a + t_{pd}) = (R(t_a) + Q'(t_a))' = (0 + 0)' = 1$$

$$Q'(t_a + 2 t_{pd}) = (S(t_a) + Q(t_a + t_{pd}))' = (0 + 1)' = 0$$

$$Q(t_a + 2 t_{pd}) = (R(t_a) + Q'(t_a + t_{pd}))' = (0 + 1)' = 0$$

$$Q'(t_a + 3 t_{pd}) = (S(t_a) + Q(t_a + 2 t_{pd}))' = (0 + 0)' = 1$$

$$Q(t_a + 3 t_{pd}) = (R(t_a) + Q'(t_a + 2 t_{pd}))' = (0 + 0)' = 1$$

...

It is happened to uncertain situation of 1 and 0 (if logic gate delay of Q and Q' path and line delay is same). In other words, the SR simultaneously rises to 1 and falls down 0, it will lead to SR Latch showing uncertain situation. So this situation

should be forbidden. Actually, logic gate delay of Q and Q' pad and line delay is not same. So two paths decide next situation by racing, the result also is different.

❖ Truth table of SR Latch

Comprehensive above analysis, SR Latch be illustrated as Truth table of Table 6.1. Among of Q+ stands for new Q value after new SR Latch input.

Table 6.1 Truth table of RS Latch

S	R	Q+
0	0	Q
1	0	1
0	1	0
1	1	Result of after race

2. Gate controlled Flip-flops

❖ Gate controlled D type Flip-flops (D type Latch)

Figure 6.6(a) is logic circuit of trigger D type Flip-flops and Figure 6.6(b) is its' circuit signal. D is input end of signal, ena is input end of enable signal (gate controlled signal), Q is output and Q' is complement output of Q. In Figure 6.6(b) is another way to establish logic circuit of D Latch by using SR Latch (this is reason why SR Latch is basic of Flip-flops)

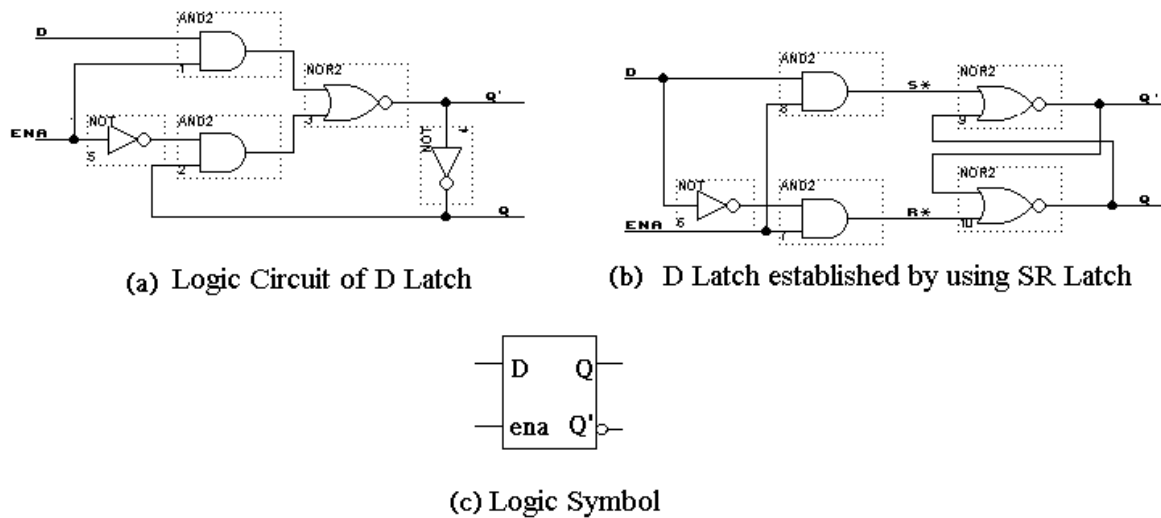


Figure 6.6 D type Latch

❖ The operation principle of D type Latch

- (1) For logic circuit of Figure 6.6(a), because $ena=1$, D input reacts to Q output (please see thick lines in Figure 6.7(a)). When $ena=0$, D signal of Q feed back and let Q still keeping original D input (please see thick lines of Figure 6.7(b))
- (2) For logic circuit of Figure 6.6(b), the (S^*, R^*) is equal input (0,0) if $ena=0$, it will let Q and Q' keeping original. The (S^*, R^*) is equal input (1,0) if $ena=1$ and $D=1$, it will let $Q=1$ and $Q'=0$ output, the (S^*, R^*) is (0,1) if $ena=1$ and $D=0$, it will let $Q=0$ and $Q'=1$ output.

❖ Truth table of D type Flip-flops

Comprehensive above analysis, D type Flip-flops be illustrated as Truth table of Table 6.2.

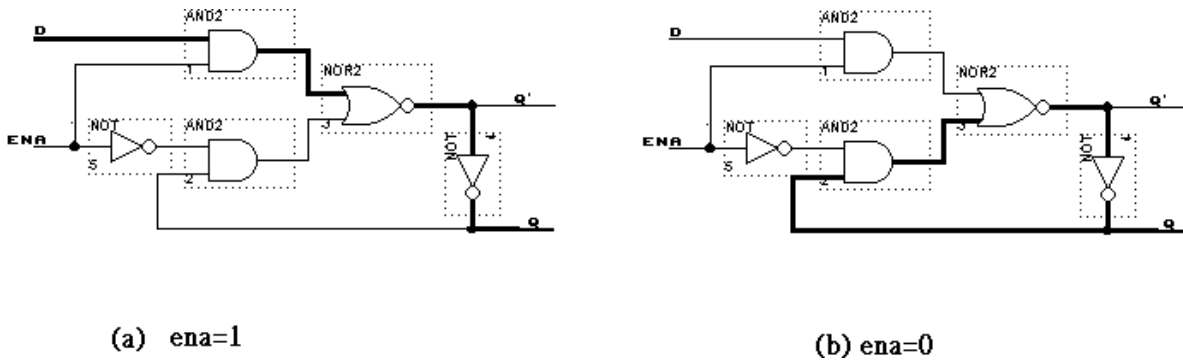


Figure 6.7 Principle of D type Latch

Table 6.2 Truth table of D type Latch

D	ena	Q^+
X	0	Q
1	1	1
0	1	0

❖ Gate controlled SR Flip-flops

In Figure 6.8, the left, establishes logic circuit of SR Flip-flops by using SR Latch, the right is its' circuit symbol, S and R are input end of signal, ena is input end of enable signal (gate controlled signal), Q is output and Q' is complement output of Q. Comprehensive above SR Latch analysis, gate controlled SR Flip-flops be illustrated as Truth table of Table 6.3. Among of Q^+ stands for new Q value after new SR input. Because SR Latch exists useless input situation, this Flip-flop almost is useless.

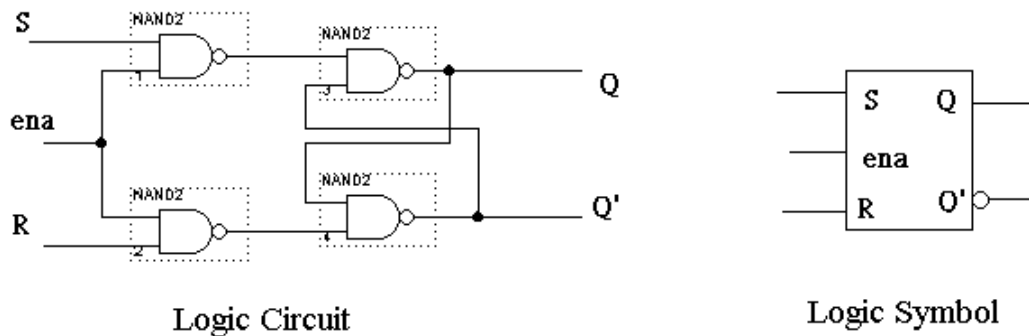


Figure 6.8 Gate controlled SR Flip-flop

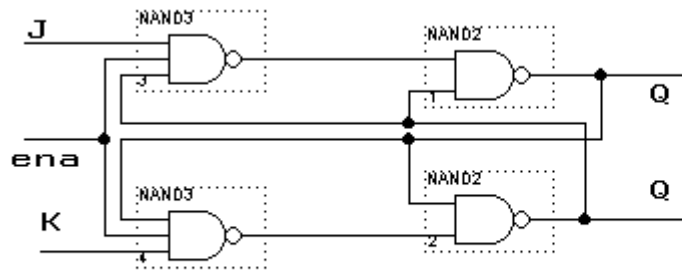
Table 6.3 Truth table of gate controlled SR Flip-flops

S	R	ena	Q ⁺	Description
X	X	0	Q	Memory
0	0	1	Q	Memory
0	1	1	0	Clear
1	0	1	1	Setting
1	1	1	X	Vain

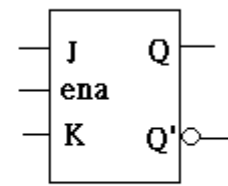
❖ Gate controlled JK Flip-flops

In Figure 6.9, the left, establish logic circuit of JK Flip-flops by using SR Latch, the right is its' circuit symbol, J and K are input end of signal, ena is input end of enable signal (gate controlled signal), Q is output and Q' is mutual output of Q.

For improving useless input of SR Flip-flops ($S=1, R=1$), the JK Flip-flops is designed. In Figure 6.10a and 6.10b, we see ($J=1, K=1$) becoming switching functions. JK Flip-flops be illustrated as Truth table of Table 6.4

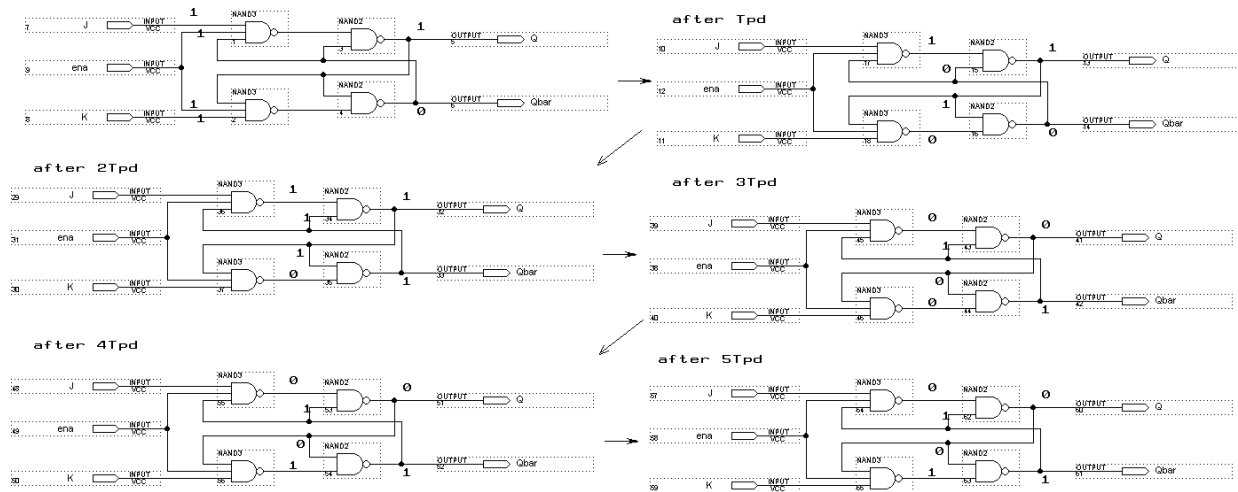


Logic Circuit



Logic Symbol

Figure 6.9 Gate controlled JK Flip-flops


Figure 6.10a Switch function of gate controlled JK Flip-flops when $J = 1$, $K = 1$ and $Q = 1$

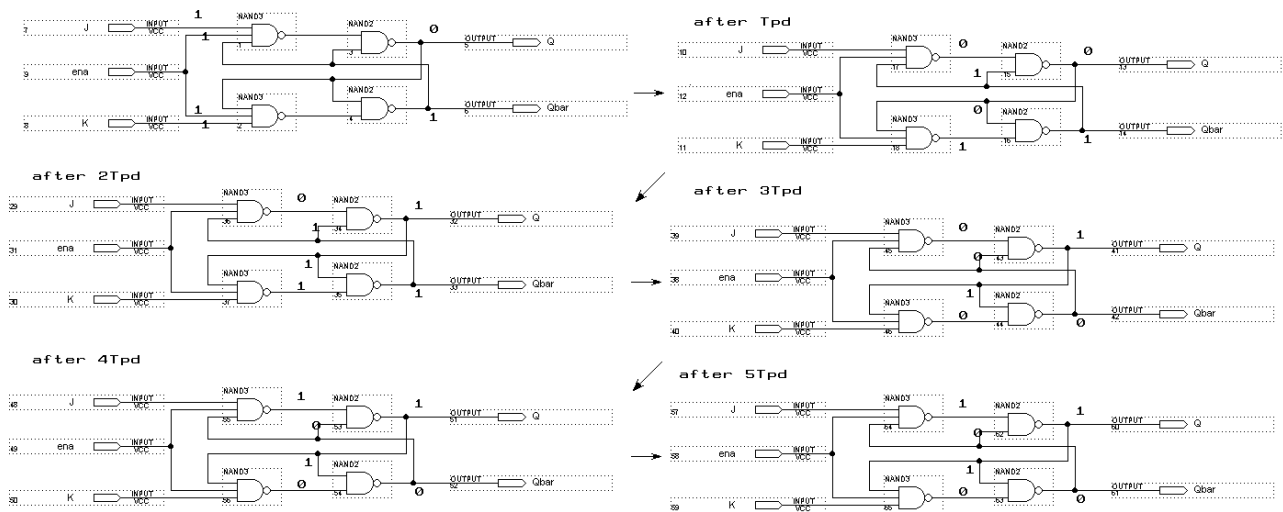


Figure 6.10a Switch function of gate controlled JK Flip-flops when $J = 1$, $K = 1$ and $Q = 0$

Table 6.4 Truth table of gate controlled JK Flip-flops

J	K	ENA	Q^+	Description
X	X	0	Q	Memory
0	0	1	Q	Memory
0	1	1	0	Clean
1	0	1	1	Setting
1	1	1	Q'	Switched

❖ Gate controlled T type Flip-flops

In Figure 6.11, the left, establishes logic circuit of T type Flip-flops by using SR Latch, the right is its' circuit symbol, T is input end of signal, ena is input end of enable signal (gate controlled signal), Q is output and Q' is mutual output of Q. JK Flip-flops be illustrated as Truth table of Table 6.5.

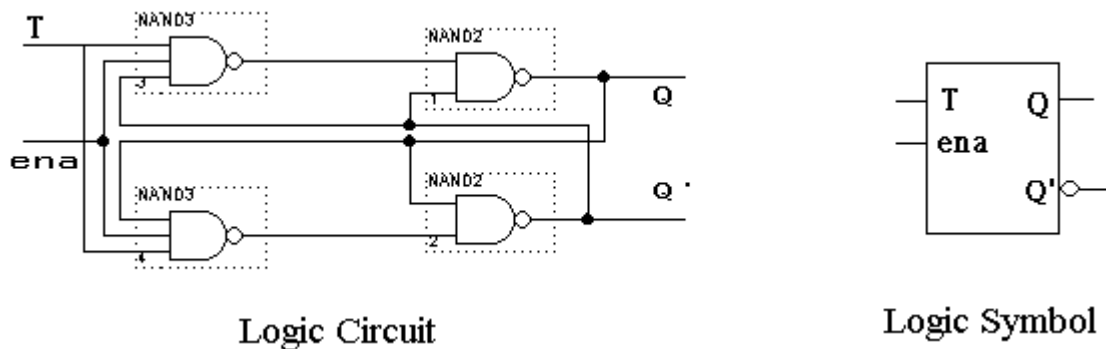


Figure 6.11 Gate controlled T type Flip-flop

Table 6.5 Truth table of gate controlled T type Flip-flop

T	Ena	Q+	Description
X	0	Q	Memory
0	1	Q	Memory
1	1	Q'	Switched

❖ Positive edge detector and negative edge detector

At present we had introduced gate controlled D type Flip-flops, gate controlled SR type Flip-flops, gate controlled JK type Flip-flops and gate controlled T type Flip-flops, which all be controlled by level trigger. Next we will introduce edge trigger Flip-flops. Before gate controlled input signal of gate controlled Flip-flops plus positive (or negative) edge detector, it will become edge trigger Flip-flops. This positive (or negative) edge detector is a clock to pulse detector that detects clock positive edge (or negative) and outputs a pulse. Figure 6.12(a) is block illustration of positive edge detector circuit. After understanding edge detector circuit, we start introducing edge trigger Flip-flops.

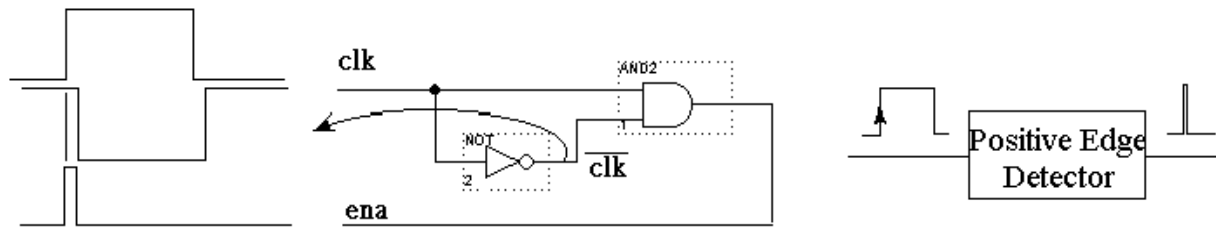


Figure 6.12(a) Positive edge detector circuit and illustration

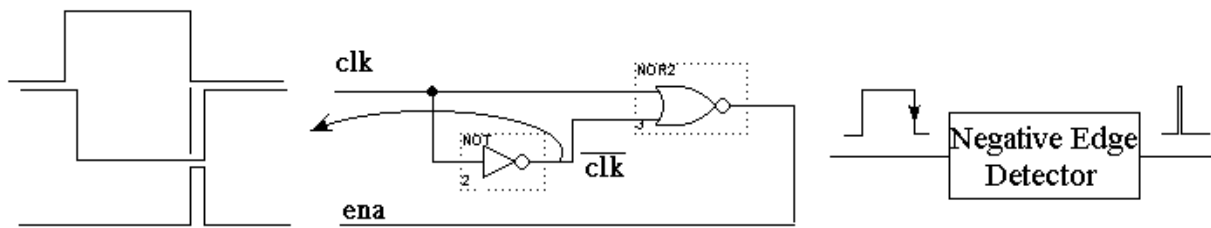


Figure 6.12(b) Negative edge detector circuit and illustration

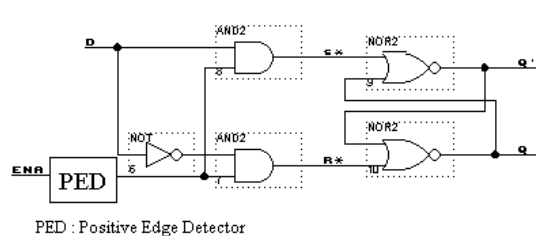
3. Edge trigger Flip-flops

❖ Edge trigger D type Flip-flops

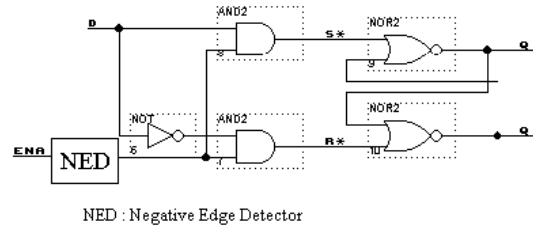
Figure 6.13 is a circuit diagram of D type Flip-flops with edge trigger and its circuit symbol. If compare Figure 6.13 with Figure 6.6, we will know only more positive edge (negative) detector. Compared with above circuit symbol of Figure 6.13, negative edge trigger is more one little circle than positive. From the Truth table (Table 6.6) of edge trigger D type Flip-flops, we can see change of output happening after positive edge or negative edge happened.

Table 6.6 Truth table of edge trigger D type Flip-flop

D	CLK	Q	Q+
X	Not ↓ (↑)	x	Q
0	↓ (↑)	0	0
0	↓ (↑)	1	0
1	↓ (↑)	0	1
1	↓ (↑)	1	1



(a) Circuit and Symbol of D-type Flip-flop with Positive Trigger



(b) Circuit and Symbol of D-type Flip-flop with Negative Trigger

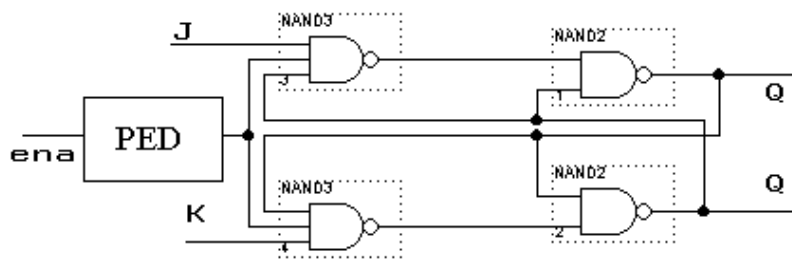
Figure 6.13 Circuit and symbol of edge trigger D Flip-flop

❖ Edge trigger JK Flip-flop

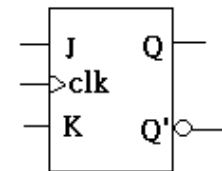
The Figure 6.14 is circuit and symbol of edge trigger JK Flip-flop. If compare Figure 6.14 with Figure 6.9, we will know only more positive edge (negative) detector. From the Truth table (Table 6.7) of edge trigger JK Flip-flops, we can see change of output happening after positive edge or negative edge happened.

Table 6.7 Truth table of edge trigger JK Flip-flops

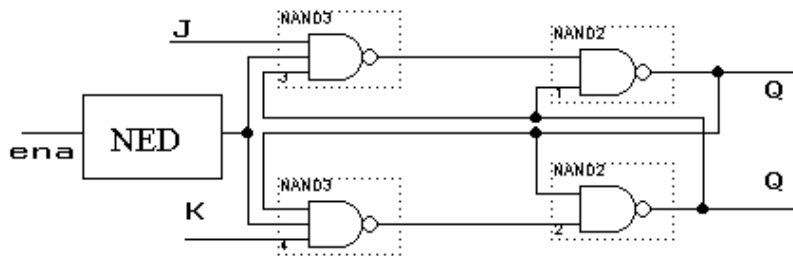
J	K	CLK	Q+	Description
X	X	Not \downarrow (\uparrow)	Q	Memory
0	0	\downarrow (\uparrow)	Q	Memory
0	1	\downarrow (\uparrow)	0	Clean
1	0	\downarrow (\uparrow)	1	Setting
1	1	\downarrow (\uparrow)	Q'	Switched



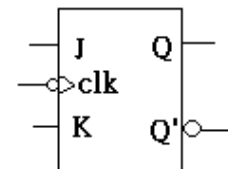
(a) Circuit of JK Flip-flop with Positive Trigger



Symbol



(b) Circuit of JK Flip-flop with Negative Trigger



Symbol

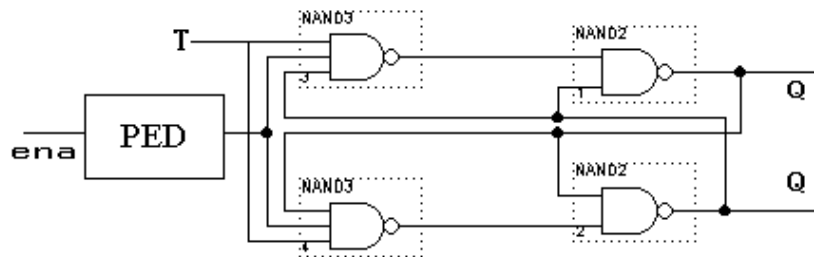
Figure 6.14 Circuit and symbol of edge trigger JK

❖ Edge trigger T type Flip-flop

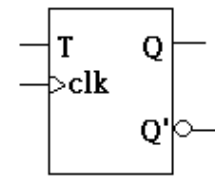
The Figure 6.15 is circuit and symbol of edge trigger T type Flip-flop. If compare Figure 6.15 with Figure 6.11, we will know only more positive (negative) edge detector. From the Truth table (Table 6.8) of edge trigger JK type Flip-flop, we can see change of output happening after positive edge or negative edge happened.

Table 6.8 Truth table of edge trigger T type Flip-flops

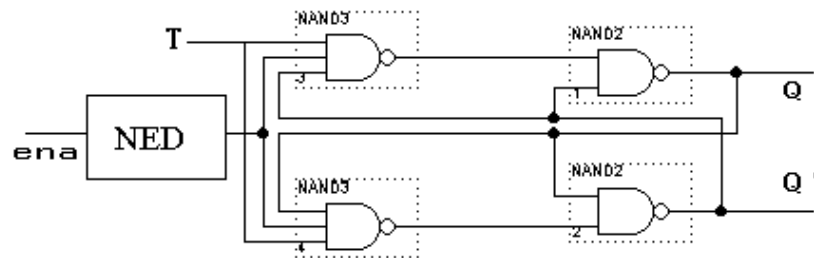
T	CLK	Q+	Description
X	Not \downarrow (\uparrow)	Q	Memory
0	\downarrow (\uparrow)	Q	Memory
1	\downarrow (\uparrow)	Q'	Switched



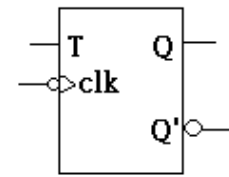
(a) Circuit of T-type Flip-flop with Positive Trigger



Symbol



(b) Circuit of T-type Flip-flop with Negative Trigger



Symbol

Figure 6.15 Circuit and symbol of edge trigger T Flip-flop

❖ The asynchronous preset and clear function of Flip-flops

Some Flip-flops includes two important inputs, which are asynchronous preset function and asynchronous clear function. T type Flip-flops is possessed of preset input and clear input as Figure 6.16. To clear T Flip-flops let Q outputting 0 (Q' outputting 1) when $\text{CLR}' = 0$ and $\text{PRN}' = 0$ (1) input. To preset T type Flip-flops let Q outputting 1 (Q' outputting 0) when $\text{CLR}' = 1$, and $\text{PRN}' = 0$ input.

Truth table of gate controlled T type Flip-flop and edge trigger T type Flip-flops be listed in Table 6.9(a) and Table 6.9(b).

Similarly, Jk Flip-flops is possessed of asynchronous clear and preset function, as Figure 6.17, Truth table of JK Flip-flops and edge trigger JK Flip-flops be listed in Table 6.10(a) and Table 6.10(b).

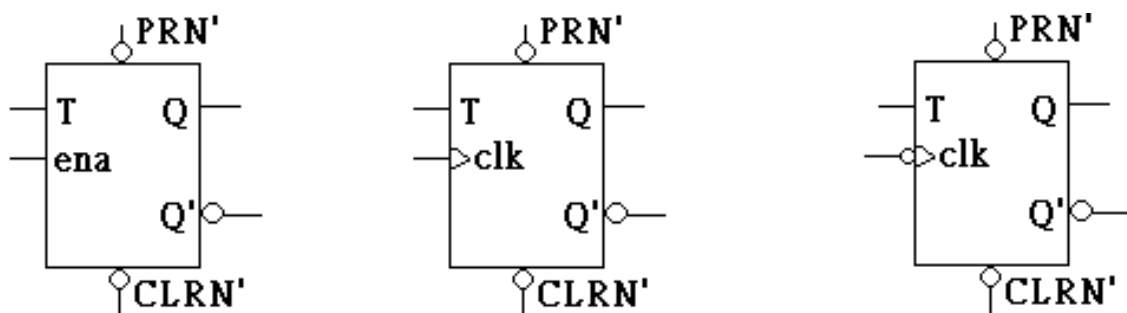


Figure 6.16 T type Flip-flops with asynchronous clear and preset function

Table 6.9a Truth table of gate controlled T type Flip-flops

CLRN'	PRN'	T	ENA	Q+	Description
0	0	X	X	0	Clear
0	1	X	X	0	Clear
1	0	X	X	1	Preset
1	1	X	0	Q	Memory
1	1	0	1	Q	Memory
1	1	1	1	Q'	Switched

Table 6.9b Truth table of edge trigger T type Flip-flops

CLRN'	PRN'	T	CLK	Q+	Description
0	0	X	X	0	Clear
0	1	X	X	0	Clear
1	0	X	X	1	Preset
1	1	X	Not \downarrow (\uparrow)	Q	Memory
1	1	0	\downarrow (\uparrow)	Q	Memory
1	1	1	\downarrow (\uparrow)	Q'	Switched

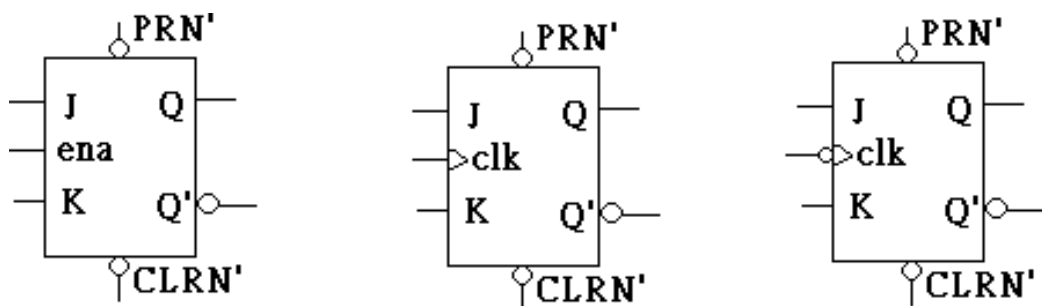


Figure 6.17 JK Flip-flops is possessed of asynchronous clear and present function



Table 6.10a Truth table of gate controlled JK Flip-flops

CLRN'	PRN'	J	K	ENA	Q+	Description
0	0	X	X	X	0	Clear
0	1	X	X	X	0	Clear
1	0	X	X	X	1	Setting
1	1	X	X	0	Q	Memory
1	1	0	0	1	Q	Memory
1	1	0	1	1	0	Clear
1	1	1	0	1	1	Setting
1	1	1	1	1	Q'	Switched

Figure 6.10b Truth table of edge trigger JK Flip-flops

CLRN'	PRN'	J	K	CLK	Q+	Description
0	0	X	X	X	0	Clear
0	1	X	X	X	0	Clear
1	0	X	X	X	1	Setting
1	1	X	X	Not ↓ (↑)	Q	Memory
1	1	0	0	↓ (↑)	Q	Memory
1	1	0	1	↓ (↑)	0	Clear
1	1	1	0	↓ (↑)	1	Setting
1	1	1	1	↓ (↑)	Q'	Switched

6.1.3 The State Tables and State Diagrams

We already mentioned model of sequential logic circuit in Figure 6.1. The all outputs result of previous input will be illustrated as state of circuit in this model. So anytime circuit output relates to present state and input. Next state of circuit is decided in same time. This relation of output, present state, next state and output, which can be illustrated by state tables and state diagrams.

Figure 6.11a is the state of sequential circuit that illustrated by state table. It includes three columns, first column stands for present state, second column stands for input $x = 0$, last column stands for input $x = 1$. For sequential circuit, present state includes four states, so be illustrated by two bits. This table illustrate that “if input is $x = 0$, output is 11 and 0 (next state is 11, output is “0”), if input is $x=1$, output is 01 and 0 (next state is 01, output is “0”) when present state is “00”; if input is $x=0$, output is 11 and 0 (next state is 11, output is ”0”), if input is $x=1$, output is 00 and 0 (next state is 00, output is “0”), when present state is “01”, the other situation is so on and so forth. The state of sequential circuit been illustrated by another state tables as Figure 6.11b.

Table 6.11a Sequential circuit state be illustrated by state tables

Present state y_1y_2	Input	
	$x = 0$	$x = 1$
00	11, 0	01, 0
01	11, 0	00, 0
11	10, 0	10, 1
10	10, 0	11, 1

Note : next state , output

Table 6.11b Sequential circuit state be illustrated by another form

Present state	Input	
y_1y_2	$x = 0$	$x = 1$
A	C, 0	B, 0
B	C, 0	A, 0
C	D, 0	D, 1
D	D, 0	C, 1

The Karnaugh map of next state (Y_1Y_2) and output z separately can be established by state tables and state diagrams, (like, Figure 6.12a, 6.12b and 6.12c), than getting the functions is:

$$z = xy_1$$

$$Y_1 = x' + y_1$$

$$Y_2 = xy_2' + x'y_1'$$

After getting z , Y_1 and Y_2 functions, we can easy draw this sequential logic circuit by logic gate and Flip-flops as Figure 6.18.

Table 6.12a Karnaugh map of Y_2 , $Y_2 = xy_2' + x'y_1'$

Y_2		y_1y_2			
		00	01	11	10
X	0	1	1	0	0
	1	1	0	0	1

Table 6.12b Karnaugh map of Y_1 , $Y_1 = x' + y_1$

Y_1		y_1y_2			
		00	01	11	10
X	0	1	1	1	1
	1	0	0	1	1

Table 6.12c Karnaugh map of z , $z = xy_1$

Z		y_1y_2			
		00	01	11	10
X	0	0	0	0	0
	1	0	0	1	1

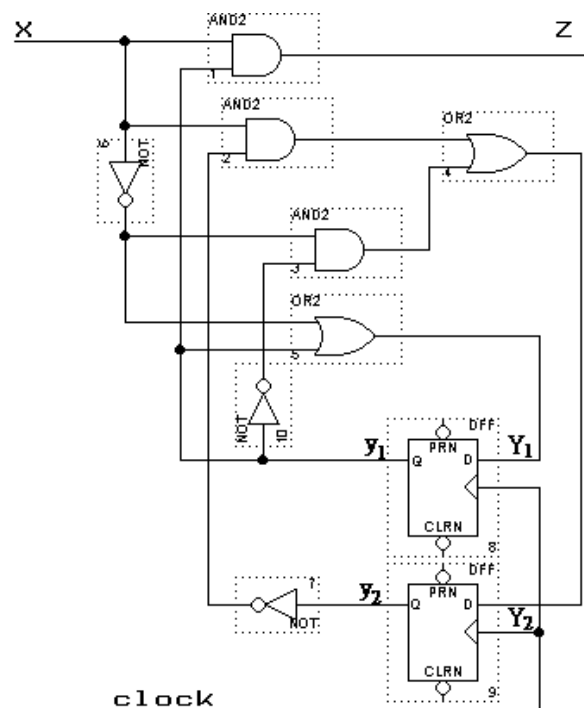
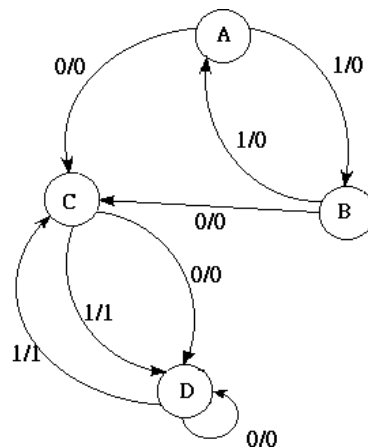


Figure 6.18 Sequential logic circuit of Table 6.11a

Another way to illustrate sequential logic circuit is state diagrams. Figure 6.17 is state diagrams of sequential logic circuit of Table 6.11b. The circles stand for state, arrows stands for state changing direction. The numbers on arrows stand for input condition of transformation and output of after transformation, like, 1/0 on A to B line stands for that transforms state to B and output “0” when state A and “1” input.

Figure 6.19 State diagrams of Table 6.11b



6.1.4 Mealy State Machine and Moore State Machine

In previous section, we had mentioned sequential circuit, which output and next state relates with present state and input. This sequential circuit is called Mealy state machine. Another model is Moore state machine, this model output only relates to present state, and next state relates to present state and input. Figure 6.20 is model of Moore state machine, and state tables of Moore state machine be illustrated as Table 6.13. Please reader compares it with Table 6.11b.

Table 6.13 Sepresentation of state tables of Moore state machine

Present state	Input		Output
	X = 0	X = 1	Z
A	B	D	1
B	C	A	0
C	C	D	0
D	B	D	0

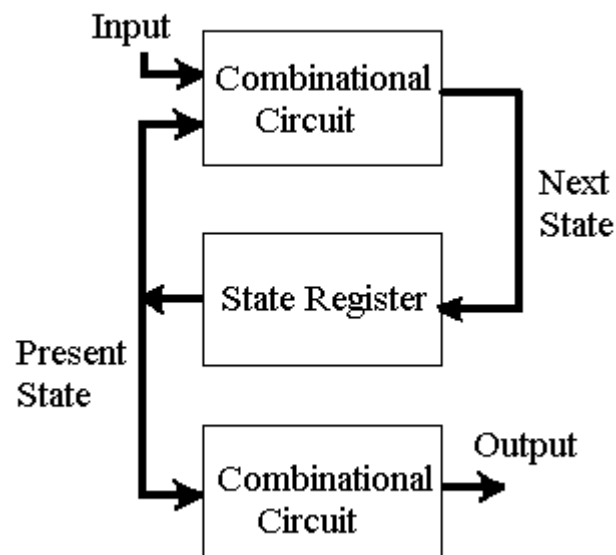


Figure 6.20 Model of Moore state machine

Figure 6.21 is state diagrams of Moore state machine. From Figure 6.21, we can see the different to Figure 6.19. In Figure 6.21, A/1 in circle stands for state A and output is “1”, it also stands for output only relation with present state. The “0” on A/1 to B/0 are line stands for that transform condition is “0”.

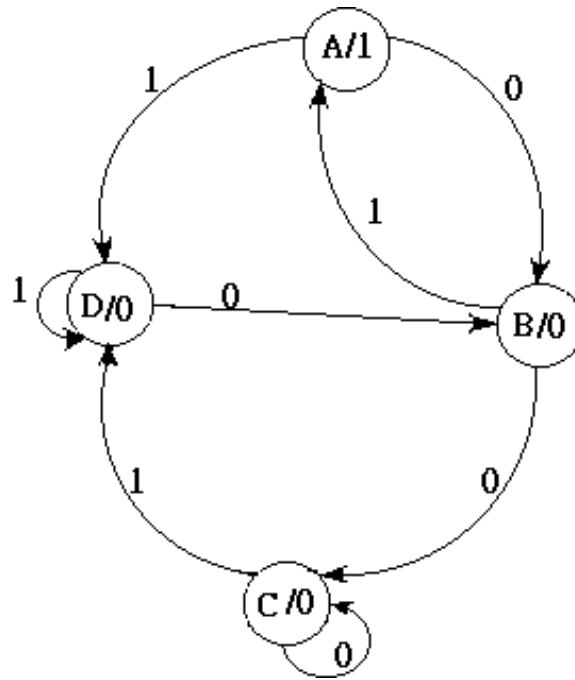


Figure 6.21 Representation of state diagrams of Moore state machine

6.1.5 The Design Progress of Synchronous Sequential Logic

Generally, the design progress of synchronous sequential logic circuit is as follows,

1. Complete the state assignment of the circuit specification and illustrate by state diagrams or state tables;
2. Find each Karnaugh map of Flip-flops inputs and output functions;
3. Find any minimum expression of input and output functions;
4. Complete above circuit entry by using graphic editor in MAX+PLUS II;

5. Complete the circuit functional simulation by using MAX+PLUS II and check weather the functions meet the specification. Go on the next step if meet the specification; otherwise go back to step 1 to check the cause of error sequentially;
6. If the circuit allowed to test by downloading (programming), select download (programming) chip and then floorplan;
7. Download (programming) the circuit to chip and test the circuit, if can't meet specification, go back to step 1 to check the cause of error sequentially.

Now, illustrate all procedures following example.

Example: Please design sequential circuit to fit with state tables of Table 6.14a by using JK Flip-flops.

Table 6.14a State tables of one sequential circuit

Present State	Input x_1x_2			
	00	01	11	10
A	C, 0	D, 0	D, 0	A, 1
B	C, 0	D, 0	D, 1	A, 1
D	A, 0	B, 0	B, 1	A, 1
C	A, 0	B, 0	B, 0	A, 1

We design as follows:

Step 1 : Complete the state assignment of the circuit specification and illustrated by state diagrams or state tables.

Due to only four states, it can cover those states only by two bits. The state assignment is as follows:

$$A = 00, B = 01, C = 10, D = 11$$

The new state table is like Table 6.14b.

Table 6.14b New state tables

Present State	Input x_1x_2			
	00	01	11	10
00	10, 0	11, 0	11, 0	00, 1
01	10, 0	11, 0	11, 1	00, 1
11	00, 0	01, 0	01, 1	00, 1
10	00, 0	01, 0	01, 0	00, 1

Step 2 : To establish Karnaugh map of J_1 (Table 6.14d), K_1 (Table 6.14e), J_2 (Table 6.14f), K_2 (Table 6.14g) and Z (Table 6.14h) by using excitation table (Table 6.14c).

Table 6.14c Excitation table of Flip-flops

Present	Next State	DFF	JK FF		TFF
Q	Q^+	D	J	K	T
0	0	0	0	–	0
0	1	1	1	–	1
1	0	0	–	1	1
1	1	1	–	0	0

Note : “–” (don’t care), it can be set 0 or 1.

Analysis of Table 6.14c is as follows:

1. For D type Flip-flop

(1) If present state Q is 0, and next state Q wants to be 0, D input must be 0.

(2) If present state Q is 0, and next state Q wants to be 1, D input must be 1.

(3) If present state Q is 1 and next state Q wants to be 0, D input must be 0.

(4) If present state Q is 1 and next state Q wants to be 1, D input must be 1.

2. For JK Flip-flops

(1) If present state Q is 0, and next state Q wants to be 0, J input must be 0 and K input must be –.

(2) If present state Q is 0, and next state Q wants to be 1, J input must be 1 and K input must be –.

(3) If present state Q is 1, and next state Q wants to be 0, J input must be – and K input must be 1.

(4) If present state Q is 1, and next state Q wants to be 1, J input must be – and K input must be 0.

3. For T type Flip-flops

(1) If present state Q is 0, and next state Q wants to be 0, T input must be 0.

(2) If present state Q is 0, and next state Q wants to be 1, T input must be 1.

(3) If present state Q is 1, and next state Q wants to be 0, T input must be 1.

(4) If present state Q is 1, and next state Q wants to be 1, T input must be 0.

Table 6.14d Karnaugh map of J_1 , $J_1 = x_1' + x_2$

J_1		Input x_1x_2			
		00	01	11	10
Present State Input	00	1	1	1	0
	01	1	1	1	0
	11	–	–	–	–
	10	–	–	–	–

The procedure of Table 6.14d is as follows:

1. When Present State Input is $y_1y_2 = 00$ and input is $x_1x_2 = 00$, next state is $y_1y_2 = 10$. The y_1 becomes 1 from 0, so needs $J_1 = 1$;
2. When Present State Input is $y_1y_2 = 00$ and input is $x_1x_2 = 01$, next state is $y_1y_2 = 11$. The y_1 becomes 1 from 0, so needs $J_1 = 1$;
3. When Present State Input is $y_1y_2 = 00$ and input is $x_1x_2 = 11$, next state is $y_1y_2 = 11$. The y_1 becomes 1 from 0, so needs $J_1 = 1$;
4. When Present State Input is $y_1y_2 = 00$ and input is $x_1x_2 = 10$, next state is $y_1y_2 = 00$. The y_1 becomes 0 from 0, so needs $J_1 = 0$;
5. When Present State Input is $y_1y_2 = 01$ and input is $x_1x_2 = 00$, next state is $y_1y_2 = 10$. The y_1 becomes 1 from 0, so needs $J_1 = 1$;
6. When Present State Input is $y_1y_2 = 01$ and input is $x_1x_2 = 01$, next state is $y_1y_2 = 11$. The y_1 becomes 1 from 0, so needs $J_1 = 1$;
7. When Present State Input is $y_1y_2 = 01$ and input is $x_1x_2 = 11$, next state is $y_1y_2 = 11$. The y_1 becomes 1 from 0, so needs $J_1 = 1$;
8. When Present State Input is $y_1y_2 = 01$ and input is $x_1x_2 = 10$, next state is $y_1y_2 = 00$. The y_1 becomes 0 from 0, so needs $J_1 = 0$;
9. When Present State Input is $y_1y_2 = 11$ and input is $x_1x_2 = 00$, next state is $y_1y_2 = 00$. The y_1 becomes 0 from 1, so needs $J_1 = \text{—}$;
10. When Present State Input is $y_1y_2 = 11$ and input is $x_1x_2 = 01$, next state is $y_1y_2 = 01$. The y_1 becomes 0 from 1 , so needs $J_1 = \text{—}$;
11. When Present State Input is $y_1y_2 = 11$ and input is $x_1x_2 = 11$, next state is $y_1y_2 = 01$. The y_1 becomes 0 from 1 , so needs $J_1 = \text{—}$;
12. When Present State Input is $y_1y_2 = 11$ and input is $x_1x_2 = 10$, next state is $y_1y_2 = 00$. The y_1 becomes 0 from 1 , so needs $J_1 = \text{—}$;
13. When Present State Input is $y_1y_2 = 10$ and input is $x_1x_2 = 00$, next state is $y_1y_2 = 00$. The y_1 becomes 0 from 1 , so needs $J_1 = \text{—}$;

14. When Present State Input is $y_1y_2 = 10$ and input is $x_1x_2 = 01$, next state is $y_1y_2 = 01$. The y_1 becomes 0 from 1, so needs $J_1 = -$;
15. When Present State Input is $y_1y_2 = 10$ and input is $x_1x_2 = 11$, next state is $y_1y_2 = 01$. The y_1 becomes 0 from 1, so needs $J_1 = -$;
16. When Present State Input is $y_1y_2 = 10$ and input is $x_1x_2 = 10$, next state is $y_1y_2 = 00$. The y_1 becomes 0 from 1, so needs $J_1 = -$;

After Karnaugh map minimization, we get $J_1 = x_1' + x_2$.

Table 6.14e Karnaugh map of K_1 , $K_1 = 1$

K_1		Input x_1x_2			
		00	01	11	10
Present State Input y_1y_2	00	—	—	—	—
	01	—	—	—	—
	11	1	1	1	1
	10	1	1	1	1

The procedure of Table 6.14e is as follows:

1. When Present State Input is $y_1y_2 = 00$ and input is $x_1x_2 = 00$, next state is $y_1y_2 = 10$. The y_1 becomes 1 from 0, so needs $K_1 = -$;
2. When Present State Input is $y_1y_2 = 00$ and input is $x_1x_2 = 01$, next state is $y_1y_2 = 10$. The y_1 becomes 1 from 0, so needs $K_1 = -$;
3. When Present State Input is $y_1y_2 = 00$ and input is $x_1x_2 = 11$, next state is $y_1y_2 = 11$. The y_1 becomes 1 from 0, so needs $K_1 = -$;
4. When Present State Input is $y_1y_2 = 00$ and input is $x_1x_2 = 10$, next state is $y_1y_2 = 00$. The y_1 becomes 0 from 0, so needs $K_1 = -$;
5. When Present State Input is $y_1y_2 = 01$ and input is $x_1x_2 = 00$, next state is

- $y_1y_2 = 10$. The y_1 becomes 1 from 0, so needs $K_1 = -$;
6. When Present State Input is $y_1y_2 = 01$ and input is $x_1x_2 = 01$, next state is $y_1y_2 = 11$. The y_1 becomes 1 from 0, so needs $K_1 = -$;
 7. When Present State Input is $y_1y_2 = 01$ and input is $x_1x_2 = 11$, next state is $y_1y_2 = 11$. The y_1 becomes 1 from 0, so needs $K_1 = -$;
 8. When Present State Input is $y_1y_2 = 01$ and input is $x_1x_2 = 10$, next state is $y_1y_2 = 00$. The y_1 becomes 0 from 0, so needs $K_1 = -$;
 9. When Present State Input is $y_1y_2 = 11$ and input is $x_1x_2 = 00$, next state is $y_1y_2 = 00$. The y_1 becomes 0 from 1, so needs $K_1 = 1$;
 10. When Present State Input is $y_1y_2 = 11$ and input is $x_1x_2 = 01$, next state is $y_1y_2 = 01$. The y_1 becomes 0 from 1, so needs $K_1 = 1$;
 11. When Present State Input is $y_1y_2 = 11$ and input is $x_1x_2 = 11$, next state is $y_1y_2 = 01$. The y_1 becomes 0 from 1, so needs $K_1 = 1$;
 12. When Present State Input is $y_1y_2 = 11$ and input is $x_1x_2 = 10$, next state is $y_1y_2 = 00$. The y_1 becomes 0 from 1, so needs $K_1 = 1$;
 13. When Present State Input is $y_1y_2 = 10$ and input is $x_1x_2 = 00$, next state is $y_1y_2 = 00$. The y_1 becomes 0 from 1, so needs $K_1 = 1$;
 14. When Present State Input is $y_1y_2 = 10$ and input is $x_1x_2 = 01$, next state is $y_1y_2 = 01$. The y_1 becomes 0 from 1, so needs $K_1 = 1$;
 15. When Present State Input is $y_1y_2 = 10$ and input is $x_1x_2 = 11$, next state is $y_1y_2 = 01$. The y_1 becomes 0 from 1, so needs $K_1 = 1$;
 16. When Present State Input is $y_1y_2 = 10$ and input is $x_1x_2 = 10$, next state is $y_1y_2 = 00$. The y_1 becomes 0 from 1, so needs $K_1 = 1$;

After Karnaugh map minimization, we get $K_1 = 1$.

According to established of above J_1 and K_1 Karnaugh map, we continue establishing Karnaugh map J_2 , K_2 and Z as Table 6.14f~Table 6.14h.

Table 6.14f Karnaugh map of J_2 , $J_2 = x_2$

J_2		Input x_1x_2			
		00	01	11	10
Present State Input	00	0	1	1	0
	01	—	—	—	—
	11	—	—	—	—
	10	0	1	1	0

Table 6.14g Karnaugh map of K_2 , $K_2 = x_2'$

K_2		Input x_1x_2			
		00	01	11	10
Present State Input	00	—	—	—	—
	01	1	0	0	1
	11	1	0	0	1
	10	—	—	—	—

Table 6.14h Karnaugh map of Z , $Z = x_1x_2' + y_2x_1$

Z		Input x_1x_2			
		00	01	11	10
Present State Input	00	0	0	0	1
	01	0	0	1	1
	11	0	0	1	1
	10	0	0	0	1

Step 3 : Find any minimum expression of input and output functions

$$J_1 = x_1' + x_2$$

$$K_1 = 1$$

$$J_2 = x_2$$

$$K_2 = x_2'$$

$$Z = x_1x_2' + y_2x_1$$

Step 4 : Complete circuit entry, Figure 6.22, by using graphic editor in MAX+PLUS II

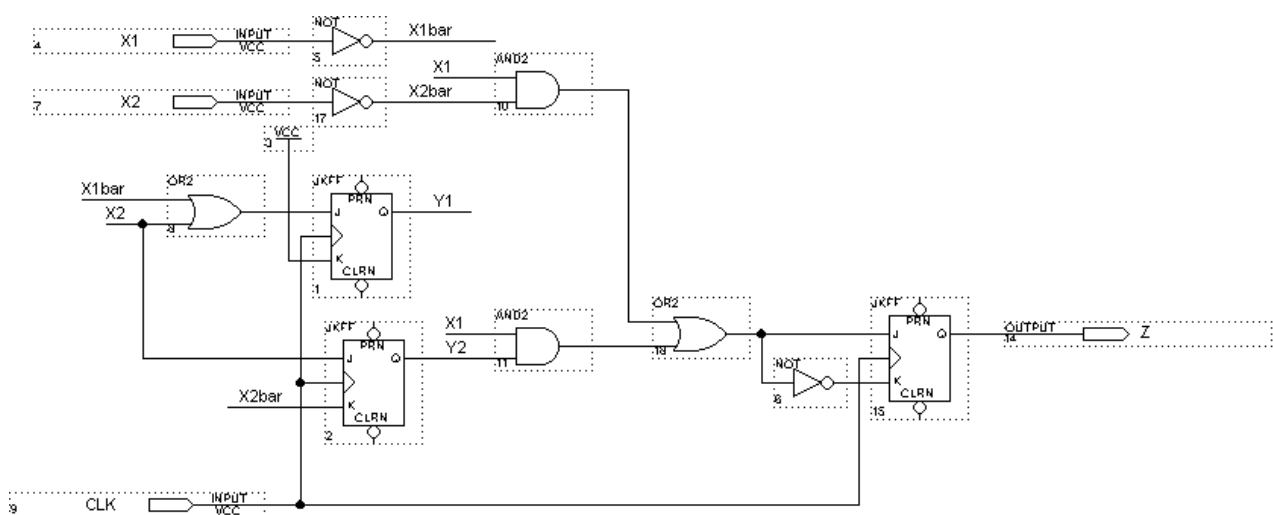


Figure 6.22 Complete the circuit entry of Table 6.14a by using MAX+PLUS II
(document : T614A.GDF)

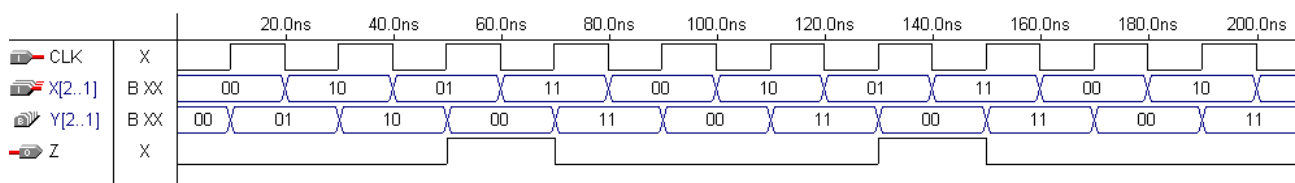


Figure 6.23 The functional simulation result of Table 6.14a by using
MAX+PLUS II simulator (document : T614A.SCF)

Step 5: Complete the circuit functional simulation by using MAX+PLUS II, Figure 6.23 shown the simulation result, and check weather the functions meet the specification. Go on the next step if meet the specification; otherwise go back to step 1 to check the cause of error sequentially

Step 6 : If the circuit allow to test by downloading (programming), select download (programming) chip and then floorplan.

As circuit modified that showed in Figure 5.6 of Section 5.2.1, please modify Figure 6.22. Please re-compile it after modifying, and adapt the ploorplan techniques in Section 4.6, select chip EPF10K10TC144-4 and use Table 6.15 pin assignment reference.

Table 6.15 Pin assignment of EPF10K10TC144-4

Name of Signal	Pin of EPF10K10TC144-4	Name of Signal	Pin of EPF10K10TC144-4
CLKIN	Pin 54 (PS1)	Y1	Pin 8
X1	Pin 48	Y2	Pin 7
X2	Pin 47	LED_COM	Pin 141

After assemble logic circuit design Lab platform LP-2900, download T614A to chip EPF10K10TC144-4. Please regulate input X1 and X2, try to push PS1 on left-bottom of LP-2900, and please note the changes of L1 (Y2) and L2 (Y1).

Exercise 1 : Please design sequential circuit of state tables (Table 6.14a) by T type Flip-flops.



Exercise 2 : Please design sequential circuit of state tables (Table 6.14a) by D type Flip-flops.

6.2 The Design, Simulation and Test of Synchronous Counter

In this section, we will design four bits binary counter and BCD counter by using JK Flip-flops.

6.2.1 Four-bit Binary Counter

There is a counter, which the initial value is 0000 (0), and arithmetic value of counter becomes 0001 (1) when clock positive edge enters. If arithmetic clock keeps flowing in , the counter will sequentially turn to 0010 (2) → 0011 (3) → 0100 (4) → 0101 (5) → ... 1110 (14) → 1111 (15) → 0000 (0) → ...and keeps cycling like this.

From above circuit specification, we know it is a four-bit binary up counter. The progress of the design is as follows:

Step 1 : Complete the state assignment of the circuit specification and illustrated by state diagrams or state tables. Circuit specification of four-bit binary counter been illustrated as state tables of Table 6.16a.

Table 6.16a State tables of four-bit binary counter sequential circuit

Present State	Next State	Output
0000	0001	0000
0001	0010	0001
0010	0011	0010
0011	0100	0011

0100	0101	0100
0101	0110	0101
0110	0111	0110
0111	1000	0111
1000	1001	1000
1001	1010	1001
1010	1011	1010
1011	1100	1011
1100	1101	1100
1101	1110	1101
1110	1111	1110
1111	0000	1111

Step 2 : Find each Karnaugh map of Flip-flops inputs and output function by using excitation table. We have found find each Karnaugh map of Flip-flops inputs and output function as Table 6.16b~Table 6.16i by using excitation table of Table 6.14c.

Table 6.16b Karnaugh map of J_0 , $J_0 = V_{cc}$

J_0		Present State Input Y_1Y_0			
		00	01	11	10
Present State Input	00	1	—	—	1
	01	1	—	—	1
	11	1	—	—	1
	10	1	—	—	1

Table 6.16c Karnaugh map of K_0 , $K_0 = V_{cc}$

K_0		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	–	1	1	–
	01	–	1	1	–
	11	–	1	1	–
	10	–	1	1	–

Table 6.16d Karnaugh map of J_1 , $J_1 = Y_0$

J_1		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	0	1	-	-
	01	0	1	-	-
	11	0	1	-	-
	10	0	1	-	-

Table 6.16e Karnaugh map of K_1 , $K_1 = Y_0$

K_1		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	–	–	1	0
	01	–	–	1	0
	11	–	–	1	0
	10	–	–	1	0

Table 6.16f Karnaugh map of J_2 , $J_2 = Y_1 Y_0$

J_2		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input $Y_3 Y_2$	00	0	0	1	0
	01	—	—	—	—
	11	—	—	—	—
	10	0	0	1	0

Table 6.16g Karnaugh map of K_2 , $K_2 = Y_1 Y_0$

K_2		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input $Y_3 Y_2$	00	—	—	—	—
	01	0	0	1	0
	11	0	0	1	0
	10	—	—	—	—

Table 6.16h Karnaugh map of J_3 , $J_3 = Y_2 Y_1 Y_0$

J_3		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input $Y_3 Y_2$	00	0	0	0	0
	01	0	0	1	0
	11	—	—	—	—
	10	—	—	—	—

Table 6.16i Karnaugh map of K_3 , $K_3 = Y_2Y_1Y_0$

K_3		Present State Input Y_1Y_0			
		00	01	11	10
Present State Input	00	-	-	-	-
	01	-	-	-	-
	11	0	0	1	0
	10	0	0	0	0

Step 3 : Find any minimum expression of input and output functions

$$J_0 = V_{cc}$$

$$K_0 = V_{cc}$$

$$J_1 = Y_0$$

$$K_1 = Y_0$$

$$J_2 = Y_1Y_0$$

$$K_2 = Y_1Y_0$$

$$J_3 = Y_2Y_1Y_0$$

$$K_3 = Y_2Y_1Y_0$$

$$Q_0 = Y_0$$

$$Q_1 = Y_1$$

$$Q_2 = Y_2$$

$$Q_3 = Y_3$$

Step 4 : Complete circuit entry, Figure 6.24, by using graphic editor in MAX+PLUS II

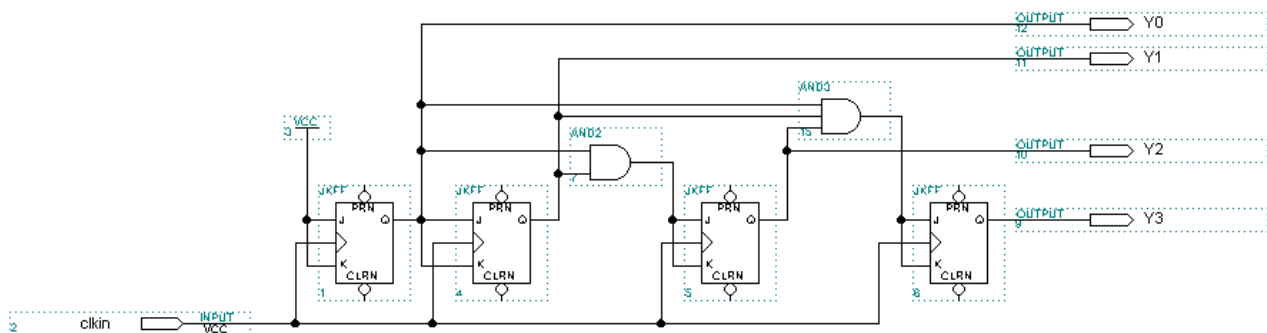


Figure 6.24 Complete the circuit entry of Table 6.16a by using MAX+PLUS II
(document : S4CNTR.GDF)

Step 5: Complete the circuit functional simulation by using MAX+PLUS II, Figure 6.25 shows the result of functional simulation, and check whether the functions meet the specification. Go on the next step if meet the specification; otherwise go back to step 1 to check the cause of error sequentially.

Step 6 : If the circuit allow to test by downloading (programming), select download (programming) chip and then floorplan.

As circuit modified that showed in Figure 5.3 of Section 5.1, please modify Figure 6.24. Please re-compile it after modifying, and adapt the floorplan techniques in Section 4.6, select chip EPF10K10-TC144-4 and use Table 6.17 pin assignment reference. After assemble logic circuit design Lab platform LP-2900, download four-bit binary up counter to chip EPF10K10TC144-4. Please try to push PS1 on left-bottom of LP-2900, and please note the changes of L1 (Y3), L2 (Y2), L3 (Y1) and L4, which if is sequence “0000”、“0001”、“0010”、... “1111”

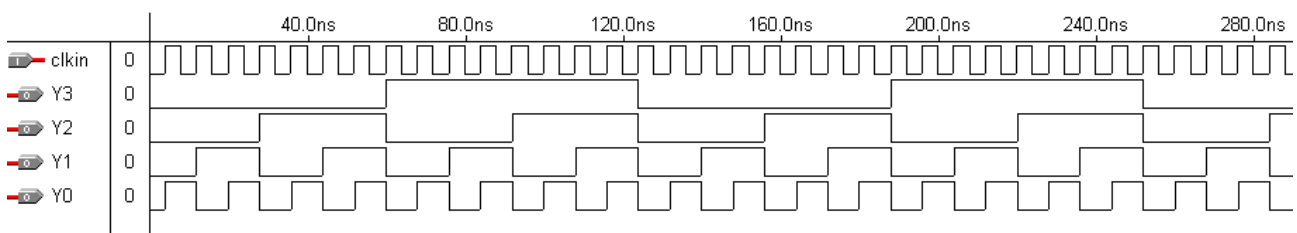


Figure 6.25 Complete simulation of circuit Table 6.16a by using MAX+PLUS II
(document : S4CNTR.SCF)

Table 6.17 Pin assignment of EPF10K10TC144-4

Name of Signal	Pin of EPF10K10TC144-4	Name of Signal	Pin of EPF10K10TC144-4
CLKIN	Pin 54 (PS1)	Y2	Pin 49
Y0	Pin 47	Y3	Pin 50
Y1	Pin 48	LED_COM	Pin 141

Exercise 3 : To design sequential circuit of four-bit binary down counter by JK Flip-flops.

Exercise 4 : To design sequential circuit of four-bit binary down counter by D type Flip-flops.

6.2.2 BCD Counter

There is a counter, which the initial value is 0000(0), and arithmetic value of counter becomes 0001 (1) when clock positive edge enters. If arithmetic clock keeps flowing in , the counter will sequentially turn to 0010 (2) → 0011 (3) → 0100 (4) → 0101 (5) → ... 1001 (9) → 0000 (0) → ...and keeps cycling like this

From above circuit specification, we know it is a BCD up counter. The progress of the design is as follows

Step 1 : Complete the state assignment of the circuit specification and illustrate by state diagrams or state tables. Circuit specification of BCD up counter been illustrated as state tables of Table 6.18a.

Step 2 : Find each Karnaugh map of Flip-flops inputs and output function by using excitation table. We have found each Karnaugh map of Flip-flops inputs and output function as Table 6.16b~Table 6.16i by using excitation table of Table 6.14c.

Table 6.18a State tables of BCD up counter sequential circuit

Present State	Next State	Output
0000	0001	0000
0001	0010	0001
0010	0011	0010
0011	0100	0011
0100	0101	0100
0101	0110	0101
0110	0111	0110
0111	1000	0111
1000	1001	1000
1001	0000	1001

Table 6.18b Karnaugh map of J_0 , $J_0 = V_{cc}$

J_0		Present State Input Y_1Y_0			
		00	01	11	10
Y ₃ Y ₂ Present State Input	00	1	—	—	1
	01	1	—	—	1
	11				
	10	1	—		

Table 6.18c Karnaugh map of K_0 , $K_0 = V_{cc}$

K_0		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	—	1	1	—
	01	—	1	1	—
	11	/	/	/	/
	10	—	1	/	/

Table 6.18d Karnaugh map of J_1 , $J_1 = Y_3' Y_0$

J_1		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	0	1	-	-
	01	0	1	-	-
	11	/	/	/	/
	10	0	0	/	/

Table 6.18e Karnaugh map of K_1 , $K_1 = Y_3' Y_0$

K_1		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	—	—	1	0
	01	—	—	1	0
	11	/	/	/	/
	10	—	—	/	/

Table 6.18f Karnaugh map of J_2 , $J_2 = Y_1 Y_0$

J_2		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input $Y_3 Y_2$	00	0	0	1	0
	01	—	—	—	—
	11				
	10	0	0		

Table 6.18g Karnaugh map of K_2 , $K_2 = Y_1 Y_0$

K_2		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input $Y_3 Y_2$	00	—	—	—	—
	01	0	0	1	0
	11				
	10	—	—		

Table 6.18h Karnaugh map of J_3 , $J_3 = Y_2 Y_1 Y_0 + Y_3 Y_0$

J_3		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input $Y_3 Y_2$	00	0	0	0	0
	01	0	0	1	0
	11				
	10	—	—		

Table 6.18i Karnaugh map of K_3 , $K_3 = Y_2Y_1Y_0 + Y_3Y_0$

K_3		Present State Input Y_1Y_0			
		00	01	11	10
	00	—	—	—	—
Y_3Y_2	01	—	—	—	—
Present State	11				
	10	0	1		

Step 3 : Find any minimum expression of input and output functions.

$$\begin{array}{ll}
 J_0 = V_{cc} & , \quad K_0 = V_{cc} \\
 J_1 = Y_3'Y_0 & , \quad K_1 = Y_3'Y_0 \\
 J_2 = Y_1Y_0 & , \quad K_2 = Y_1Y_0 \\
 J_3 = Y_2Y_1Y_0 + Y_3Y_0 & , \quad K_3 = Y_2Y_1Y_0 + Y_3Y_0 \\
 Q_0 = Y_0 & , \quad Q_1 = Y_1 \\
 Q_2 = Y_2 & , \quad Q_3 = Y_3
 \end{array}$$

Step 4 : Complete the circuit entry, Figure 6.26, by using graphic editor in MAX+PLUS II.

Step 5: Complete the circuit functional simulation by using MAX+PLUS II, Figure 6.27 shows the result of functional simulation, and check weather the functions meet the specification. Go on the next step if meet the specification; otherwise go back to step 1 to check the cause of error sequentially.

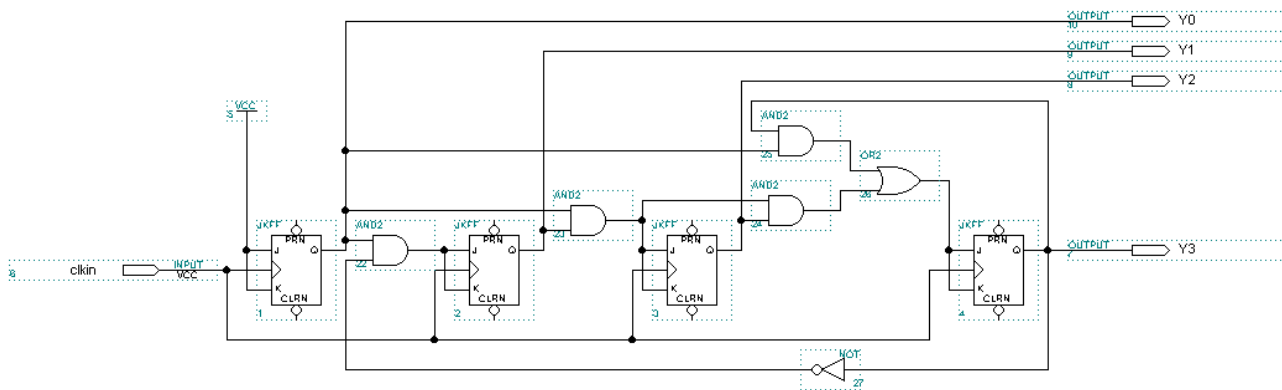


Figure 6.26 Complete the circuit entry of Table 6.18a by using MAX+PLUS II(document : SBCDCNTR.GDF)

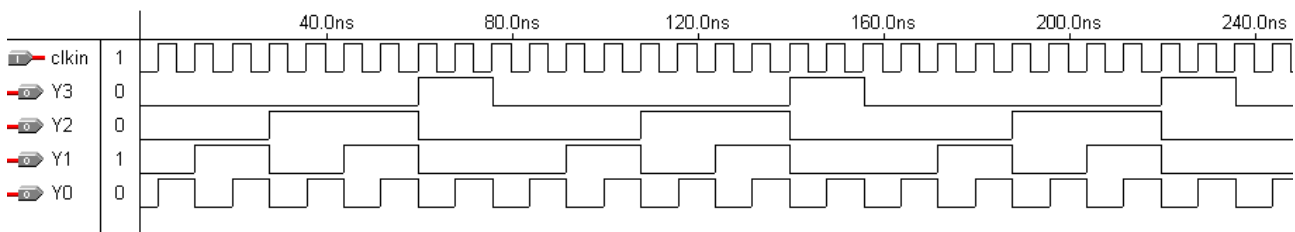


Figure 6.27 The functional simulation result of Table6.18a by using MAX+PLUS II simulator (document: SBCDCNTR.SCF)

Step 6 : If the circuit allow to test by downloading (programming), select download (programming) chip and then floorplan.

As circuit modified that showed in Figure 5.3 of Section 5.1, please modify Figure 6.26. Please re-compile it after modifying, and adapt the ploorplan techniques in Section 4.6, select chip EPF10K10TC144-4 and use Table 6.19 pin assignment reference. After assemble logic circuit design Lab platform LP-2900, download BCD up counter to chip EPF10K10TC144-4. Please try to push PS1 on left-bottom of LP-2900, and please note the changes of L1 (Y3), L2 (Y2), L3 (Y1) and L4, which if is sequence “ 0000”,“0001”,“0010”,... “1001”,“0000”.

Table 6.19 Pin assignment of EPF10K10TC144-4

Name of Signal	Pin of EPF10K10TC144-4	Name of Signal	Pin of EPF10K10TC144-4
CLKIN	Pin 54 (PS1)	Y2	Pin 49
Y0	Pin 47	Y3	Pin 50
Y1	Pin 48	LED_COM	Pin 141

Exercise 5 : Please design sequential circuit of BCD down counter by JK Flip-flops

Exercise 6 : Please design sequential circuit of BCD down counter by D type Flip-flops

6.3 The Design, Simulation and Test of Synchronous Shift Register

In this section, we will use D type Flip-flops to design four shift registers: four-bit serial input and serial output (SISO), four-bit serial input and parallel output (SIPO), Four-bit parallel input and serial output (PISO), Four-bit parallel input and parallel output (PIPO)

6.3.1 SISO Shift Register

There a circuit, which has four registers with input $D_3D_2D_1D_0$ and output $Y_3Y_2Y_1Y_0$, has serial input Sin and serial output So. Original data Y_1 showed in Y_0 , original data Y_2 showed in Y_1 , and original data Y_3 showed in Y_2 and original data Sin showed in Y_3 when clock positive edge enters. Besides, So data is Y_0 data.

To achieve above circuit specification design, the design progress as follows

Step 1 : Complete the state assignment of the circuit specification and illustrate by state diagrams or state tables. Circuit specification of four-bit SISO shift register been illustrated as state tables of Table 6.20a.

Table 6.20a State tables of SISO shift register sequential circuit

Present State	Input X		Output
	0	1	So
0000	0000	1000	0
0001	0000	1000	1
0010	0001	1001	0
0011	0001	1001	1
0100	0010	1010	0
0101	0010	1010	1
0110	0011	1011	0
0111	0011	1011	1
1000	0100	0100	0
1001	0100	1100	1
1010	0101	1101	0
1011	0101	1101	1
1100	0110	1110	0
1101	0110	1110	1
1110	0111	1111	0
1111	0111	1111	1

Note : Next state shows in double-line grid

Step 2 : Find each Karnaugh map of Flip-flops inputs and output function by using excitation table. We have found each Karnaugh map of D type Flip-flops inputs and output function as Table 6.20b~Table 6.20f by using excitation table of Table 6.14c. (This example uses Karnaugh map of 5 variables)

Table 6.20b Karnaugh map of D_0 , $D_0 = Y_1$

Input $x = 0$

D_0		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	0	0	1	1
	01	0	0	1	1
	11	0	0	1	1
	10	0	0	1	1

Input $x = 1$

D_0		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	0	0	1	1
	01	0	0	1	1
	11	0	0	1	1
	10	0	0	1	1

Table 6.20c Karnaugh map of D_1 , $D_1 = Y_2$

Input $x = 0$

D_1		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	0	0	0	0
	01	1	1	1	1
	11	1	1	1	1
	10	0	0	0	0

Input X = 1

D ₁		Present State Input Y ₁ Y ₀			
		00	01	11	10
Y ₃ Y ₂ Present State Input	00	0	0	0	0
	01	1	1	1	1
	11	1	1	1	1
	10	0	0	0	0

Table 6.20d Karnaugh map of D₂, D₂ = Y₃

Input x = 0

D ₂		Present State Input Y ₁ Y ₀			
		00	01	11	10
Y ₃ Y ₂ Present State Input	00	0	0	0	0
	01	0	0	0	0
	11	1	1	1	1
	10	1	1	1	1

Input X = 1

D ₂		Present State Input Y ₁ Y ₀			
		00	01	11	10
Y ₃ Y ₂ Present State Input	00	0	0	0	0
	01	0	0	0	0
	11	1	1	1	1
	10	1	1	1	1

Table 6.20e Karnaugh map of D_3 , $D_3 = x$

Input $x = 0$

D_3		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	0	0	0	0
	01	0	0	0	0
	11	0	0	0	0
	10	0	0	0	0

Input $X = 1$

D_3		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	1	1	1	1
	01	1	1	1	1
	11	1	1	1	1
	10	1	1	1	1

Table 6.20f Karnaugh map of S_0 , $S_0 = Y_0$

Input $x = 0$

S_0		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	0	1	1	0
	01	0	1	1	0
	11	0	1	1	0
	10	0	1	1	0

Input X = 1

So		Present State Input Y ₁ Y ₀			
		00	01	11	10
Y ₃ Y ₂ Present State Input	00	0	1	1	0
	01	0	1	1	0
	11	0	1	1	0
	10	0	1	1	0

Step 3 : Find any minimum expression of input and output functions

$$\begin{aligned} D_0 &= Y_1 & ; & & D_1 &= Y_2 \\ D_2 &= Y_3 & ; & & D_3 &= X \\ S_0 &= Y_0 \end{aligned}$$

Step 4 : Complete the circuit entry, shown in Figure 6.28, by using graphic editor in MAX+PLUS II.

Step 5: Complete the circuit functional simulation by using MAX+PLUS II, Figure 6.29 illustrates the result of functional simulation, and check whether the functions meet the specification. Go on the next step if meet the specification; otherwise go back to step 1 to check the cause of error sequentially.

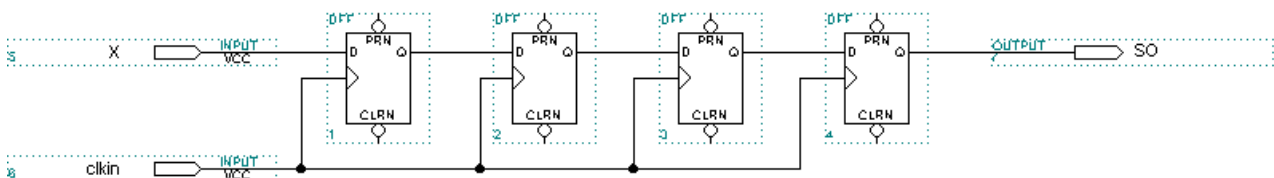


Figure 6.28 Complete the circuit entry Table 6.20e by using MAX+PLUS II
(document : S4SISO.GDF)

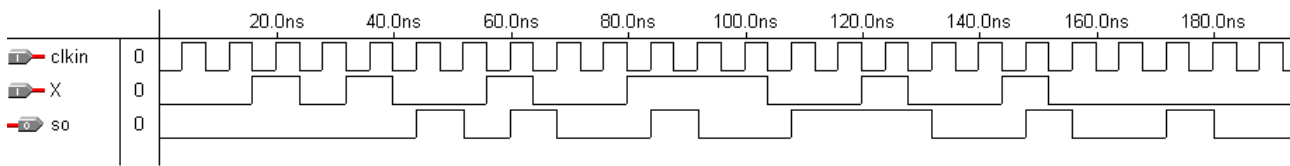


Figure 6.29 The functional simulation result of Table 6.20e by using MAX+PLUS II simulator (document : S4SISO.SCF)

Step 6 : If the circuit allow to test by downloading(programming), select download (programming) chip and then floorplan.

As circuit modified that showed in Figure 5.3 of Section 5.1, please modify Figure 6.28. Please re-compile it after modifying, and adapt the ploorplan techniques in Section 4.6, select chip EPF10K10TC144-4 and use Table 6.21 pin assignment reference. After assemble logic circuit design Lab platform LP-2900, download SISO shift register to chip EPF-10K10TC144-4. Please regulate X input and try to push PS1 on left-bottom of LP-2900. Please note the changes of L1 (So).

Table 6.21 Pin assignment of EPF10K10TC144-4

Name of Signal	EPF10K10TC144-4 Chip pin	Name of Signal	EPF10K10TC144-4 Chip pin
CLKIN	Pin 54 (PS1)	So	Pin 7
X	Pin 47	LED_COM	Pin 141

Exercise 7 : Please design sequential circuit of Four-bit binary SISO shift register by JK Flip-flops.

Exercise 8 : Please design sequential circuit of Four-bit binary SISO shift register by T type Flip-flops.

6.3.2 SIPO Shift Register

There a circuit, which has four registers with input $D_3D_2D_1D_0$ and output $Y_3Y_2Y_1Y_0$, has a serial input S_{in} and parallel output $P_3P_2P_1P_0$. Original data Y_1 showed in Y_0 , original data Y_2 showed in Y_1 , and original data Y_3 showed in Y_2 and original data S_{in} showed in Y_3 when clock positive edge enters. Besides, $P_3P_2P_1P_0$ also show $Y_3Y_2Y_1Y_0$ data.

To approach above circuit specification design, the design progress as follows

Step 1 : Complete the state assignment of the circuit specification and illustrated by state diagrams or state tables. Circuit specification of Four-bit SIPO shift register is illustrated as state tables of Table 6.22a.

Table 6.22a State table of SIPO shift register sequential circuit

Present	Input X		Output $P_3P_2P_1P_0$
	0	1	
0000	0000	1000	0000
0001	0000	1000	0001
0010	0001	1001	0010
0011	0001	1001	0011
0100	0010	1010	0100
0101	0010	1010	0101
0110	0011	1011	0110
0111	0011	1011	0111
1000	0100	0100	1000
1001	0100	1100	1001
1010	0101	1101	1010

1011	0101	1101	1011
1100	0110	1110	1100
1101	0110	1110	1101
1110	0111	1111	1110
1111	0111	1111	1111

Step 2 : Find each Karnaugh map of Flip-flops inputs and output function by using excitation table. We have found each Karnaugh map of D type Flip-flops inputs and output function as Table 6.22b~Table 6.22i by using excitation table of Table 6.14c. (The example uses Karnaugh map of 5 variables)

Table 6.22b Karnaugh map of D_0 , $D_0 = Y_1$

Input $x = 0$

D_0		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	0	0	1	1
	01	0	0	1	1
	11	0	0	1	1
	10	0	0	1	1

Input $X = 1$

D_0		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	0	0	1	1
	01	0	0	1	1
	11	0	0	1	1
	10	0	0	1	1

Table 6.22c Karnaugh map of D_1 , $D_1 = Y_2$

Input $x = 0$

D_1		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	0	0	0	0
	01	1	1	1	1
	11	1	1	1	1
	10	0	0	0	0

Input $X = 1$

D_1		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	0	0	0	0
	01	1	1	1	1
	11	1	1	1	1
	10	0	0	0	0

Table 6.22d Karnaugh map of D_2 , $D_2 = Y_3$

Input $x = 0$

D_2		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	0	0	0	0
	01	0	0	0	0
	11	1	1	1	1
	10	1	1	1	1

Input $X = 1$

D_2		Present State Input $Y_1 Y_0$			
		00	01	11	10
$Y_3 Y_2$ Present State Input	00	0	0	0	0
	01	0	0	0	0
	11	1	1	1	1
	10	1	1	1	1

Table 6.22e Karanaugh map of D_3 , $D_3 = x$ Input $x = 0$

D_3		Present State Input $Y_1 Y_0$			
		00	01	11	10
$Y_3 Y_2$ Present State Input	00	0	0	0	0
	01	0	0	0	0
	11	0	0	0	0
	10	0	0	0	0

Input $X = 1$

D_3		Present State Input $Y_1 Y_0$			
		00	01	11	10
$Y_3 Y_2$ Present State Input	00	1	1	1	1
	01	1	1	1	1
	11	1	1	1	1
	10	1	1	1	1

Table 6.22f Karnaugh map of P_0 , $P_0 = Y_0$

P_0		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	0	1	1	0
	01	0	1	1	0
	11	0	1	1	0
	10	0	1	1	0

Table 6.22g Karnaugh map of P_1 , $P_1 = Y_1$

P_0		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	0	0	1	1
	01	0	0	1	1
	11	0	0	1	1
	10	0	0	1	1

Table 6.22h Karnaugh map of P_2 , $P_2 = Y_2$

P_2		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	0	0	0	0
	01	1	1	1	1
	11	1	1	1	1
	10	0	0	0	0

Table 6.22i Karnaugh map of P_3 , $P_3 = Y_3$

P_3		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input $Y_3 Y_2$	00	0	0	0	0
	01	0	0	0	0
	11	1	1	1	1
	10	1	1	1	1

Step 3 : Find any minimum expression of input and output functions

$$\begin{array}{ll}
 D_0 = Y_1 & ; \quad D_1 = Y_2 \\
 D_2 = Y_3 & ; \quad D_3 = X \\
 P_0 = Y_0 & ; \quad P_1 = Y_1 \\
 P_2 = Y_2 & ; \quad P_3 = Y_3
 \end{array}$$

Step 4 : Complete the circuit entry, Figure 6.30, by using graphic editor in MAX+PLUS II

Step 5: Complete the circuit functional simulation by using MAX+PLUS II, Figure 6.3 illustrates the result of functional simulation, and check weather the functions meet the specification. Go on the next step if meet the specification; otherwise go back to step 1 to check the cause of error sequentially.

Step 6 : If the circuit allow to test by downloading (programming), select download (programming) chip and then floorplan.

As circuit modified that showed in Figure 5.3 of Section 5.1, please modify Figure 6.30. Please re-compile it after modifying, and adapt the ploorplan techniques in Section 4.6, select chip EPF10K10-TC144-4 and use Table 6.23 pin assignment reference. After assemble logic circuit design Lab platform LP-2900, download SISO shift register to chip EPF10K10TC144-4. Please regulate X input and try to push PS1 on left-bottom of LP-2900, and please note the changes of L1 (P3), L2 (P2), L3 (P1) and L4 (Po).

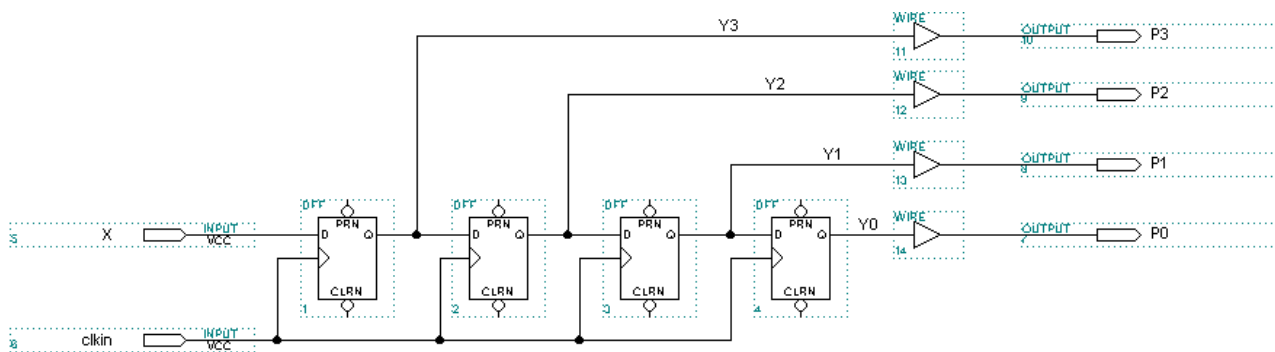


Figure 6.30 Complete the circuit entry of Table 6.22a by using MAX+PLUS II
(document : S4SIPO.GDF)

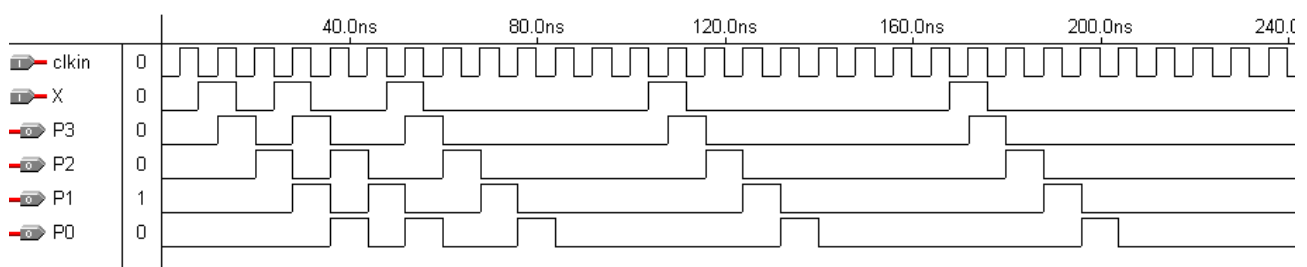


Figure 6.31 The functional simulation result of Table6.22a by using MAX +
PLUS II simulator (document : S4SIPO.SCF)

Table 6.23 Pin assignment of EPF10K10TC144-4

Name of Signal	Pin of EPF10K10TC144-4	Name of Signal	Pin of EPF10K10TC144-4
CLKIN	Pin 54 (PS1)	P2	Pin 48
X	Pin 47	P3	Pin 47
P0	Pin 50		
P1	Pin 49	LED_COM	Pin 141

Exercise 9 : Please design sequential circuit of Four-bit binary SIPO shift register by JK Flip-flops.

Exercise 10 : Please design sequential circuit of Four-bit binary SIPO shift register by T type Flip-flops.

6.3.3 PISO Shift Register

There a circuit, which has four registers with input $D_3D_2D_1D_0$ and output $Y_3Y_2Y_1Y_0$, has parallel input dcba and serial output So and input L is load signal. Original data Y_1 showed in Y_0 , original data Y_2 showed in Y_1 , original data Y_3 showed in Y_2 and data “0” showed in Y_3 when $L=0$ and clock positive edge enters. Besides, So also shows Y_0 data.

To achieve above circuit specification design, the design progress as follows

Step 1 : Complete the state assignment of the circuit specification and illustrated by state diagrams or state tables. Circuit specification of four-bit PISO shift register be illustrated as state tables of Table 6.24a

Step 2 : Find each Karnaugh map of Flip-flops inputs and output function by using excitation table. We have found Karnaugh map of D type Flip-flops inputs and output function as Table 6.24b~Table 6.24f by using excitation table of Table 6.14c. (The example uses Karnaugh map of 6 variables).

Table 6.24a State tables of PISO shift register sequential circuit

Present State	Input		Output So
	0	1	
0000	0000	dcba	0
0001	0000	dcba	1
0010	0001	dcba	0
0011	0001	dcba	1
0100	0010	dcba	0
0101	0010	dcba	1
0110	0011	dcba	0
0111	0011	dcba	1
1000	0100	dcba	0
1001	0100	dcba	1
1010	0101	dcba	0
1011	0101	dcba	1
1100	0110	dcba	0
1101	0110	dcba	1
1110	0111	dcba	0
1111	0111	dcba	1

Note : dcba is parallel input data

Table 6.24b Karnaugh map of D_0 , $D_0 = La + L'Y_1$

Input $L = 0$, a

D_0		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input $Y_3 Y_1$	00	0	0	1	1
	01	0	0	1	1
	11	0	0	1	1
	10	0	0	1	1

Input $L = 1$, a

D_0		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input $Y_3 Y_2$	00	A	a	a	a
	01	A	a	a	a
	11	A	a	a	a
	10	A	a	a	a

Table 6.24c Karnaugh map of D_1 , $D_1 = Lb + L'Y_2$

Input $L = 0$, b

D_1		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input $Y_3 Y_2$	00	0	0	0	0
	01	1	1	1	1
	11	1	1	1	1
	10	0	0	0	0

Input L = 1, b

D ₁		Present State Input Y ₁ Y ₀			
		00	01	11	10
Y ₃ Y ₂ Present State Input	00	B	b	b	b
	01	B	b	b	b
	11	B	b	b	b
	10	B	b	b	b

Table 6.24d Karnaugh map of D₂, D₂ = Lc + L'Y₃

Input L = 0, c

D ₂		Present State Input Y ₁ Y ₀			
		00	01	11	10
Y ₃ Y ₂ Present State Input	00	0	0	0	0
	01	0	0	0	0
	11	1	1	1	1
	10	1	1	1	1

Input L = 1, 0 c

D ₂		Present State Input Y ₁ Y ₀			
		00	01	11	10
Y ₃ Y ₂ Present State Input	00	C	c	c	c
	01	C	c	c	c
	11	C	c	c	c
	10	C	c	c	c

Input L = 0, d

D_3		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	0	0	0	0
	01	0	0	0	0
	11	0	0	0	0
	10	0	0	0	0

Table 6.24e Karnaugh map of D_3 , $D_3 = Ld$

Input L = 1, d

D_3		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	d	d	d	d
	01	d	d	d	d
	11	d	d	d	d
	10	d	d	d	d

Table 6.24f Karnaugh map of S_o , $S_o = Y_0$

S_o		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	0	1	1	0
	01	0	1	1	0
	11	0	1	1	0
	10	0	1	1	0

Step 3 : Find any minimum expression of input and output functions

$$\begin{aligned}D_0 &= La + L'Y_1 & ; & & D_1 &= Lb + L'Y_2 \\D_2 &= Lc + L'Y_3 & ; & & D_3 &= Ld \circ \\So &= Y_0 \circ\end{aligned}$$

Step 4 : Complete circuit entry, Figure 6.32, by using graphic editor in MAX+PLUS II

Step 5: Complete the circuit functional simulation by using MAX+PLUS II, Figure 6.33 illustrates the result of functional simulation, and check weather the functions meet the specification. Go on the next step if meet the specification; otherwise go back to step 1 to check the cause of error sequentially.

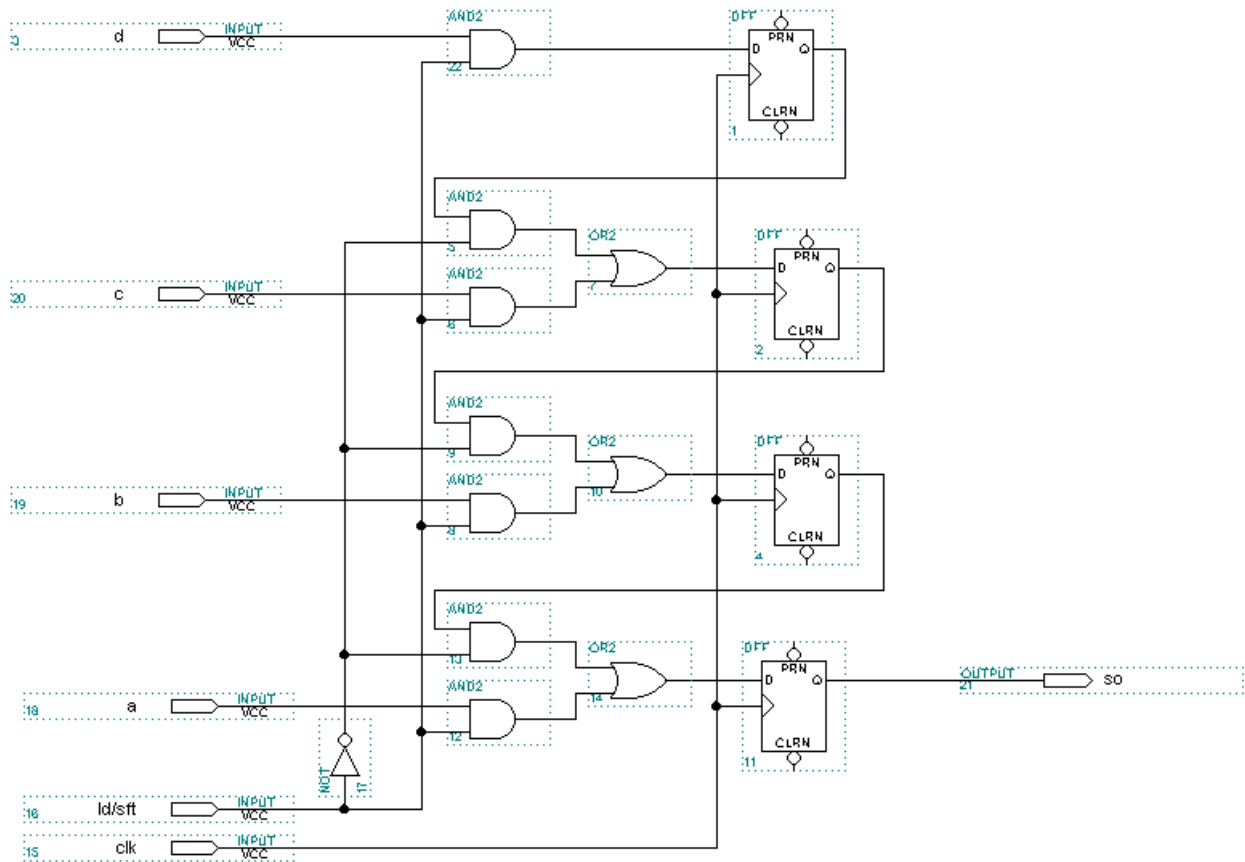


Figure 6.32 Complete the circuit entry of Table 6.24a by using MAX+PLUS II
(document : S4PISO.GDF)

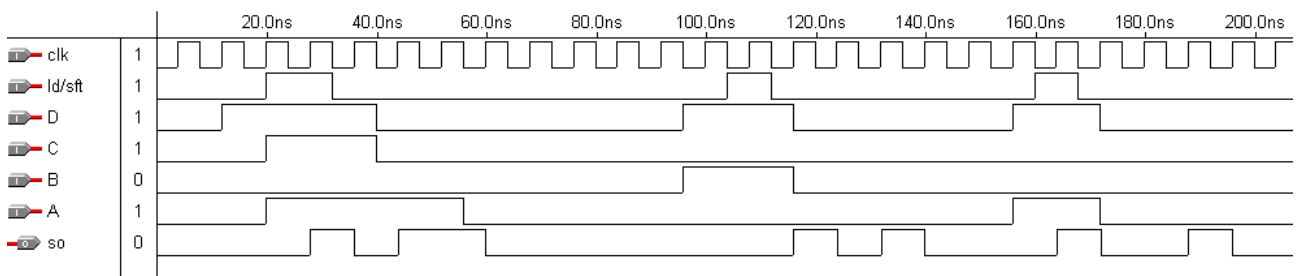


Figure 6.33 The functional simulation results of Table 6.24a by using MAX + PLUS II simulator(document : S4PISO.SCF)

Step 6 : If the circuit allow to test by downloading (programming), select download (programming) chip and then floorplan.

As circuit modified that showed in Figure 5.3 of Section 5.1, please modify Figure 6.32. Please re-compile it after modifying, and adapt the ploorplan techniques in Section 4.6, select chip EPF10K10TC144-4 and use Table 6.25 pin assignment reference. After assemble logic circuit design Lab platform LP-2900, download four-bit PISO shift register to chip EPF-10K10TC144-4. Please regulate input of a (SW3), b (SW2), c (SW1) and d (SW0). Let Ld/sft (SW8) on “1” and try to push PS1 on left-bottom of LP-2900, then let Ld/sft (SW8) off “0” and try to push PS1 on left-bottom of LP-2900. Please note the changes of L1 (So), which if shows input of a (SW3), b (SW2), c (SW1) and d (SW0).

Table 6.25 Pin assignment of EPF10K10TC144-4

Name of Signal	EPF10K10TC144-4 chip pin	Name of Signal	EPF10K10TC144-4 chip pin
D	Pin 47	CLKIN	Pin 54 (PS1)
C	Pin 48	Ld/sft	Pin 63
B	Pin 49	So	Pin 7
A	Pin 51	LED_COM	Pin 141

Exercise 11 : Please design sequential circuit of Four-bit binary PISO shift register by JK Flip-flops.

Exercise 12 : Please design sequential circuit of Four-bit binary PISO shift register by T type Flip-flops.

6.3.4 PIPO Shift Register

There a circuit, which has four registers with input $D_3D_2D_1D_0$ and output $Y_3Y_2Y_1Y_0$, has parallel input dcba and parallel output $P_3P_2P_1P_0$ and input L is load signal. The input dcba show in $Y_3Y_2Y_1Y_0$ when $L=1$ and clock positive edge enters. Original

data Y_1 showed in Y_0 , original data Y_2 showed in Y_1 , original data Y_3 showed in Y_2 and data “0” showed in Y_3 when $L=0$ and clock positive edge enters. Besides, $P_3P_2P_1P_0$ also shows $Y_3Y_2Y_1Y_0$ data.

To approach above circuit specification design, the design progress as follows

Table 6.26a State tables of PIPO shift register sequential circuit

Present State	Input L		Output $P_3P_2P_1P_0$
	0	1	
0000	0000	dcba	0000
0001	0000	dcba	0001
0010	0001	dcba	0010
0011	0001	dcba	0011
0100	0010	dcba	0100
0101	0010	dcba	0101
0110	0011	dcba	0110
0111	0011	dcba	0111
1000	0100	dcba	1000
1001	0100	dcba	1001
1010	0101	dcba	1010
1011	0101	dcba	1011
1100	0110	dcba	1100
1101	0110	dcba	1101
1110	0111	dcba	1110
1111	0111	dcba	1111

Note : dcba is parallel input data

Step 1 : Complete the state assignment of the circuit specification and illustrates by state diagrams or state tables. Circuit specification of four-bit PIPO shift register is illustrated as state tables of Table 6.26a.

Step 2 : Find each Karnaugh map of Flip-flops inputs and output function by using excitation table. We have found Karnaugh map of D type Flip-flops inputs function as Table 6.27b~Table 6.27i by using excitation table of Table 6.14c.

Table 6.27b Karnaugh map of D_0 , $D_0 = La + L'Y_1$

Input $L = 0$, a

D_0		Present State Input Y_1Y_0			
		00	01	11	10
Present State Input Y_3Y_2	00	0	0	1	1
	01	0	0	1	1
	11	0	0	1	1
	10	0	0	1	1

Input $L = 1$, a

D_0		Present State Input Y_1Y_0			
		00	01	11	10
Present State Input Y_3Y_2	00	A	a	a	a
	01	A	a	a	a
	11	A	a	a	a
	10	A	a	a	a

Table 6.27c Karnaugh map of D_1 , $D_1 = Lb + L'Y_2$

Input $L = 0$, b

D_1		Present State Input Y_1Y_0			
		00	01	11	10
Present State Input Y_3Y_2	00	0	0	0	0
	01	1	1	1	1
	11	1	1	1	1
	10	0	0	0	0

Input $L = 1$, b

D_1		Present State Input Y_1Y_0			
		00	01	11	10
Present State Input Y_3Y_2	00	B	b	b	b
	01	B	b	b	b
	11	B	b	b	b
	10	B	b	b	b

Table 6.27d Karnaugh map of D_2 , $D_2 = Lc + L'Y_3$

Input $L = 0$, c

D_2		Present State Input Y_1Y_0			
		00	01	11	10
Present State Input Y_3Y_2	00	0	0	0	0
	01	0	0	0	0
	11	1	1	1	1
	10	1	1	1	1



Input L = 1, c

D ₂		Present State Input Y ₁ Y ₀			
		00	01	11	10
Y ₃ Y ₂ Present State Input	00	C	c	c	c
	01	C	c	c	c
	11	C	c	c	c
	10	C	c	c	c

Table 6.27e Karnaugh map of D₃, D₃ = Ld

Input L = 0, d

D ₃		Present State Input Y ₁ Y ₀			
		00	01	11	10
Y ₃ Y ₂ Present State Input	00	0	0	0	0
	01	0	0	0	0
	11	0	0	0	0
	10	0	0	0	0

Input L = 1, d

D ₃		Present State Input Y ₁ Y ₀			
		00	01	11	10
Y ₃ Y ₂ Present State Input	00	D	d	d	d
	01	D	d	d	d
	11	D	d	d	d
	10	D	d	d	d

Table 6.27f Karnaugh map of P_0 , $P_0 = Y_0$

P_0		Present State Input $Y_1 Y_0$			
		00	01	11	10
$Y_3 Y_2$ Present State Input	00	0	1	1	0
	01	0	1	1	0
	11	0	1	1	0
	10	0	1	1	0

Table 6.27g Karnaugh map of P_1 , $P_1 = Y_1$

P_1		Present State Input $Y_1 Y_0$			
		00	01	11	10
$Y_3 Y_2$ Present State Input	00	0	0	1	1
	01	0	0	1	1
	11	0	0	1	1
	10	0	0	1	1

Table 6.27h Karnaugh map of P_2 , $P_2 = Y_2$

P_1		Present State Input $Y_1 Y_0$			
		00	01	11	10
$Y_3 Y_2$ Present State Input	00	0	0	0	0
	01	1	1	1	1
	11	1	1	1	1
	10	0	0	0	0

Table 6.27i Karnaugh map of $P_3, P_3 = Y_3$

P_1		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input	00	0	0	0	0
	01	0	0	0	0
	11	1	1	1	1
	10	1	1	1	1

Step 3 : Find any minimum expression of input and output functions

$$\begin{aligned}
 D_0 &= L_a + L'Y_1 & ; & & D_1 &= L_b + L'Y_2 \\
 D_2 &= L_c + L'Y_3 & ; & & D_3 &= L_d \\
 P_0 &= Y_0 & ; & & P_1 &= Y_1 \\
 P_2 &= Y_2 & ; & & P_3 &= Y_3
 \end{aligned}$$

Step 4 : Complete circuit entry, Figure 6.34, by using graphic editor in MAX+PLUS II.

Step 5: Complete the circuit functional simulation by using MAX+PLUS II, Figure 6.35 illustrates the result of functional simulation, and check weather the functions meet the specification. Go on the next step if meet the specification; otherwise go back to step 1 to check the cause of error sequentially.

Step 6 : If the circuit allow to test by downloading (programming), select download (programming) chip and then floorplan.

As circuit modified that showed in Figure 5.3 of Section 5.1, please modify Figure 6.34. Please re-compile it after modifying, and adapt the ploorplan techniques in section 4.6, select chip EPF10K10TC144-4 and use Table 6.28 pin assignment reference. After assemble logic circuit design Lab platform LP-2900, download four-bit PIPO shift register to chip EPF10K10TC144-4. Please regulate input of a (SW3), b (SW2), c (SW1) and d (SW0). Let Ld/sft (sw8) on “1” and try to push PS1 on left-bottom of LP-2900. Please note the changes of L1 (P3), L2 (P2), L3 (P1) and L4 (P0). And then let Ld/sft (sw8) off “1” and try to push PS1. Please note the changes of L1, L2, L3 and L4.

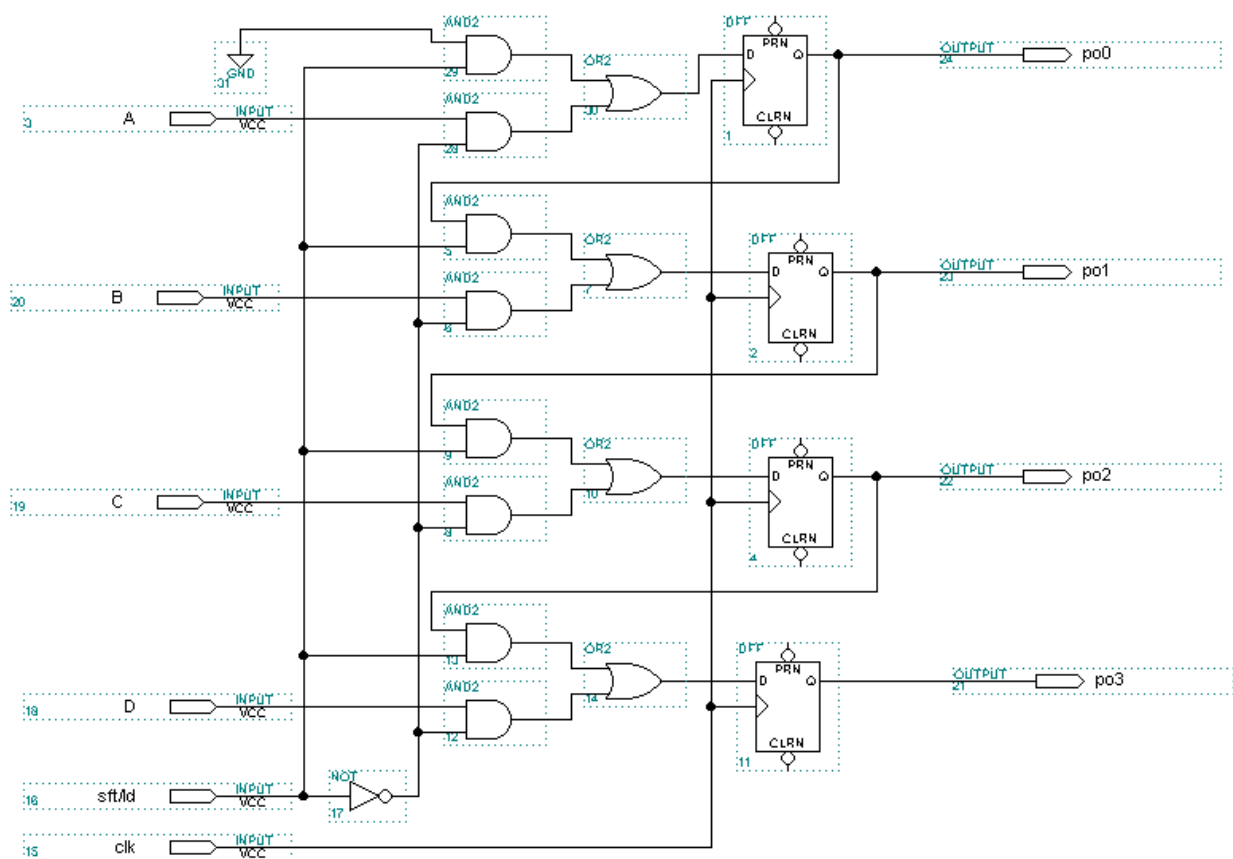


Figure 6.34 Complete the circuit entry of Table 6.24a by using MAX + PLUS II
(document : S4PIPO.GDF)

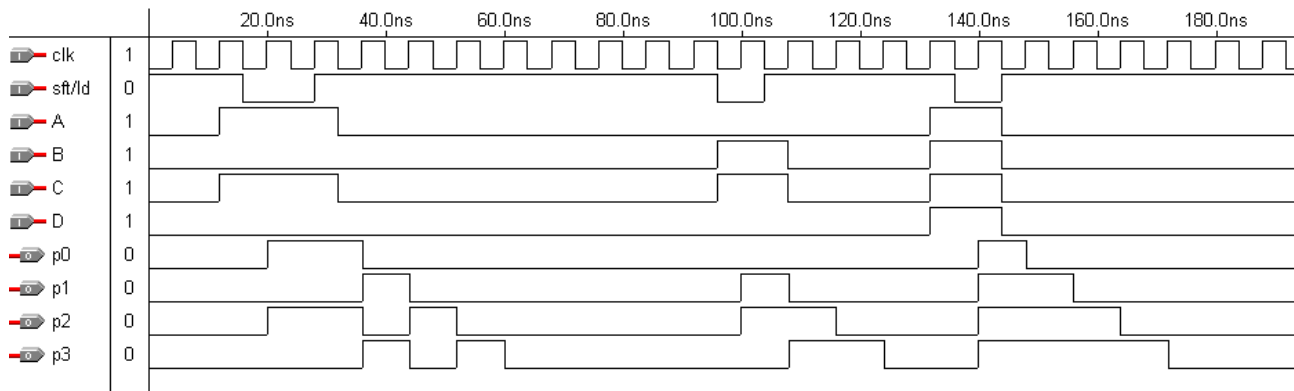


Figure 6.35 The functional simulation results of Table 6.27a by using MAX + PLUS II simulator (document : S4PIPO.SCF)

Table 6.28 Pin assignment EPF10K10TC144-4

Name of Signal	EPF10K10TC144-4 chip pin	Name of Signal	EPF10K10TC144-4 chip pin
CLKIN	Pin 54 PS1)	P0	Pin 10
Ld/sft	Pin 63	P1	Pin 9
D	Pin 47	P2	Pin 8
C	Pin 48	P3	Pin 7
B	Pin 49		
A	Pin 51	LED_COM	Pin 141

Exercise 13 : Please design sequential circuit of Four-bit binary PIPO shift register by JK Flip-flops.

Exercise 14 : Please design sequential circuit of Four-bit binary PIPO shift register by T type Flip-flops.

6.4 The Design, Simulation and Test of Synchronous Shift Count Register

In this section, we will use T type Flip-flops to design four-bit ring counter and four-bit Johnson counter.

6.4.1 Ring Counter

There is a counter, which the state of after clear is 1000 (8); arithmetic value of counter becomes 0100 (4) when clock positive edge enters. If arithmetic clock keeps flowing in , the counter will sequentially turn to 0010 (2) → 0001 (1) → 1000 (8) → 0100 (4) → 0010 (2) → 0001 (1) → 1000 (8) →and keeps cycling like this.

From above circuit specification, we know it is a Four-bit Ring counter with asynchronous clear function, because its “1” bit cyclic progress likes as ripple, the progress of the design is as follows

Step 1 : Complete the state assignment of the circuit specification and illustrated by state diagrams or state tables. Circuit specification of Four-bit ring counter is illustrated as state tables of Table 6.29a.

Table 6.29a Sequential circuit of state tables of Ring counter

Present State	Input clrn		Output $D_3D_2D_1D_0$
	0	1	
1000	1000	0100	1000
0100	1000	0010	0100
0010	1000	0001	0010
0001	1000	1000	0001

Step 2 : Find each Karnaugh map of Flip-flops inputs and output function by using excitation table. We have found each Karnaugh map of T type Flip-flops inputs function as Table 6.29b~Table 6.29e by using excitation table of Table 6.14c.

Table 6.29b Karnaugh map of T_0 , $T_0 = Y_3'Y_2'Y_1'Y_0 + Y_3'Y_2'Y_1Y_0'clrn$

Input clrn = 1

T_0		Present State Input Y_1Y_0			
		00	01	11	10
Y_3Y_2 Present State Input	00		1		1
	01	0			
	11				
	10	0			

Input clrn = 0

T_0		Present State Input Y_1Y_0			
		00	01	11	10
Y_3Y_2 Present State Input	00		1		0
	01	0			
	11				
	10	0			

Table 6.29c Karnaugh map of T_1 , $T_1 = Y_3'Y_2Y_1'Y_0'clrn + Y_3'Y_2'Y_1Y_0'$

Input clrn = 1

T_1		Present State Input Y_1Y_0			
		00	01	11	10
Y_3Y_2 Present State Input	00		0		1
	01	1			
	11				
	10	0			

Input clrn = 0

T_0		Present State Input Y_1Y_0			
		00	01	11	10
Y_3Y_2 Present State Input	00		0		1
	01	0			
	11				
	10	0			

Table 6.29d Karnaugh map of T_2 , $T_2 = Y_3'Y_2Y_1'Y_0' + Y_3Y_2'Y_1'Y_0'clrn$

Input clrn = 1

T_2		Present State Input Y_1Y_0			
		00	01	11	10
Y_3Y_2 Present State Input	00		0		0
	01	1			
	11				
	10	1			

Input clrn = 0

T_0		Present State Input $Y_1 Y_0$			
		00	01	11	10
$Y_3 Y_2$ Present State Input	00		0		0
	01	1			
	11				
	10	0			

Table 6.29e Karnaugh map of T_3 , $T_3 = Y_3'Y_2'Y_1'Y_0 \text{clrn} + Y_3Y_2'Y_1'Y_0' \text{clrn}$

Input clrn = 1

T_3		Present State Input $Y_1 Y_0$			
		00	01	11	10
$Y_3 Y_2$ Present State Input	00		1		0
	01	0			
	11				
	10	1			

Input clrn = 0

T_0		Present State Input $Y_1 Y_0$			
		00	01	11	10
$Y_3 Y_2$ Present State Input	00		0		0
	01	0			
	11				
	10	0			

Step 3 : Find any minimum expression of input and output functions

$$T_0 = Y_3'Y_2'Y_1'Y_0 + Y_3'Y_2'Y_1Y_0' \text{ clrn}$$

$$T_1 = Y_3'Y_2Y_1'Y_0' \text{ clrn} + Y_3'Y_2'Y_1Y_0'$$

$$T_2 = Y_3'Y_2Y_1'Y_0' + Y_3Y_2'Y_1'Y_0' \text{ clrn}$$

$$T_3 = Y_3'Y_2'Y_1'Y_0 \text{ clrn} + Y_3Y_2'Y_1'Y_0' \text{ clrn}$$

$$Q_0 = Y_0 \quad ; \quad Q_1 = Y_1$$

$$Q_2 = Y_2 \quad ; \quad Q_3 = Y_3$$

Step 4 : Complete the circuit entry, Figure 6.36, by using graphic editor in MAX+PLUS II.

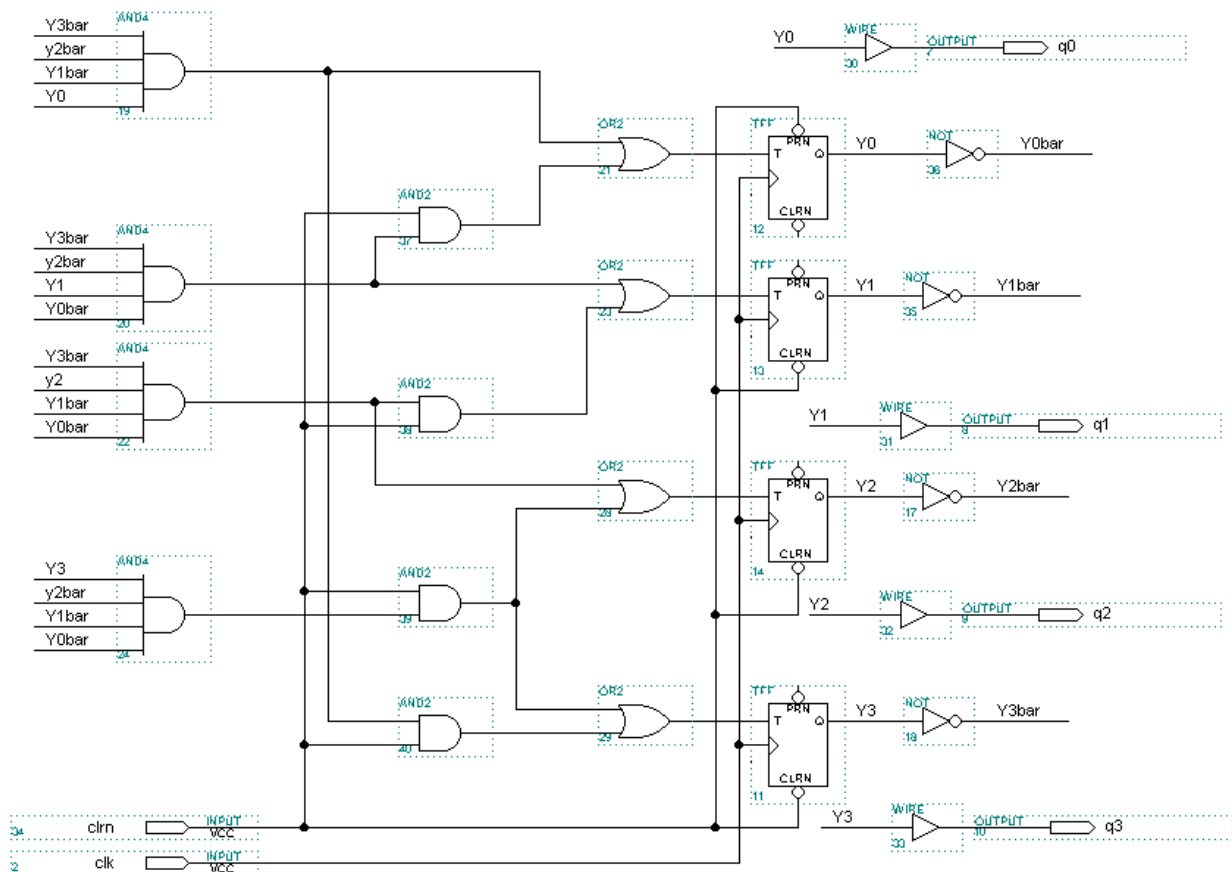


Figure 6.36 Complete the circuit entry of Table 6.29a by using MAX+PLUS II
(document : Ring4.GDF)

Step 5: Complete the circuit functional simulation by using MAX+PLUS II, Figure 6.37 illustrates the result of functional simulation, and check whether the functions meet the specification. Go on the next step if meet the specification; otherwise go back to step 1 to check the cause of error sequentially.

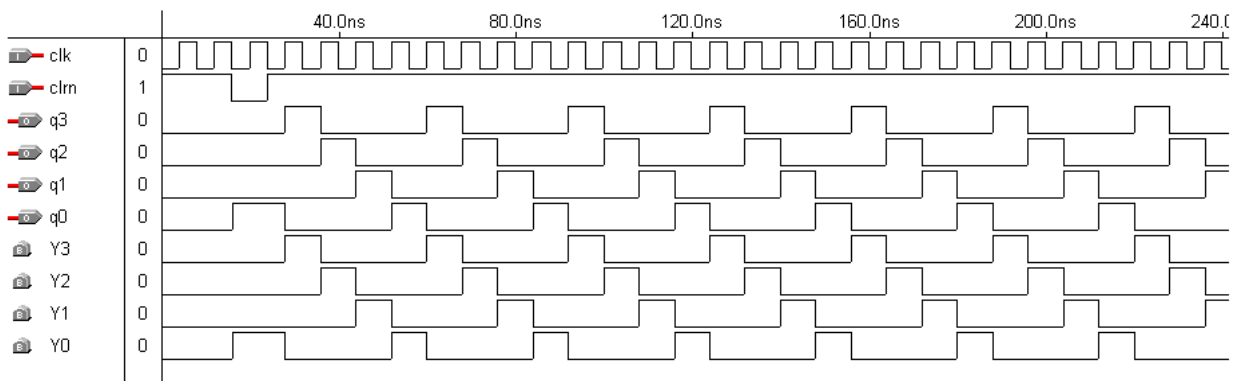


Figure 6.37 The functional simulation results of Table 6.29a by using MAX+PLUS II simulator (document : Ring4.SCF)

Step 6 : If the circuit allow to test by downloading (programming), select download (programming) chip and then floorplan.

As circuit modified that showed in Figure 5.3 of Section 5.1, please modify Figure 6.36. Please re-compile it after modifying, and adapt the floorplan techniques in section 4.6, select chip EPF10K10TC144-4 and use Table 6.30 pin assignment reference. After assemble logic circuit design Lab platform LP-2900, download four-bit ring counter to chip EPF10K10TC144-4. Please try to push SW1 (CLRn) then push PS1 on left-bottom of LP-2900. Please note the changes of L1 (Q3), L2 (Q2), L3 (Q1) and L4 (Q0), and see if cyclic like as 1000 (8) → 0100 (4) → 0010 (2) → 0001 (1) → 1000 (8) → 0100 (4) → 0010 (2) → 0001 (1) → 1000 (8) →

Table 6.30 Pin assignment of EPF10K10TC144-4

Name of Signal	Pin of EPF10K10TC144-4	Name of Signal	Pin of EPF10K10TC144-4
CLKIN	Pin 54(PS1)	Q2	Pin 8
CLRN	Pin 47	Q3	Pin 7
Q0	Pin 10		
Q1	Pin 9	LED_COM	Pin 141

Exercise 15 : Please design sequential circuit of Four-bit bins by using JK Flip-flops.

Exercise 16 : Please design sequential circuit of Four-bit binary PIPO ring counter by using D type Flip-flops.

6.4.2 Johnson Counter

There is a counter, which the state of after clear is 0000 (0), arithmetic value of counter becomes 0000 (1) when clock positive edge enters. If arithmetic clock keeps flowing in , the counter will sequentially turn to 0011 (3) → 0111 (7) → 1111 (F) → 1110 (E) → 1100 (C) → 1000 (8) → 0000 (0) → 0001 (1) →.....and keeps cycling like this

From above circuit specification, we know it is a four-bit johnson counter with asynchronous clear. The progress of the design is as follows

Step 1 : Complete the state assignment of the circuit specification and illustrated by state diagrams or state tables. Circuit specification of Four-bit Johnson counter is illustrated as state tables of Table 6.31a.

Table 6.31a Sequential circuit of state tables of Johnson counter

Present State	Input clrn		Output $D_3D_2D_1D_0$
	0	1	
0000	0000	0001	0000
0001	0000	0011	0001
0011	0000	0111	0011
0111	0000	1111	0111
1111	0000	1110	1111
1110	0000	1100	1110
1100	0000	1000	1100
1000	0000	0000	1000

Step 2 : Find each Karnaugh map of Flip-flops inputs and output function by using excitation table. We have found each Karnaugh map of T type Flip-flops inputs function as Table 6.31b~Table 6.31e by using excitation table of Table 6.14c.

Table 6.31b Karnaugh map of T_0 , $T_0 = Y_3'Y_2'Y_1'Y_0' + Y_3Y_2Y_1Y_0$

Input clrn = 1

T_0		Present State Input Y_1Y_0			
		00	01	11	10
Present State Input	00	1	0	0	
	01			0	
	11	0		1	0
	10	0			

Table 6.31c Karnaugh map of T_1 , $T_1 = Y_3'Y_2'Y_1'Y_0 + Y_3Y_2Y_1Y_0'$

Input clrn = 1

T_1		Present State Input Y_1Y_0			
		00	01	11	10
Present State Input	00	0	1	0	
	01			0	
	11	0		0	1
	10	0			

Table 6.31d Karnaugh map of T_2 , $T_2 = Y_3'Y_2'Y_1Y_0 + Y_3Y_2Y_1'Y_0'$

Input clrn = 1

T_2		Present State Input Y_1Y_0			
		00	01	11	10
Present State Input	00	0	0	1	
	01			0	
	11	1		0	0
	10	0			

Table 6.31e Karnaugh map of T_3 , $T_3 = Y_3'Y_2Y_1Y_0 + Y_3Y_2'Y_1'Y_0'$

Input clrn = 1

T_3		Present State Input Y_1Y_0			
		00	01	11	10
Present State Input	00	0	0	0	
	01			1	
	11	0		0	0
	10	1			

Step 3 : Find any minimum expression of input and output functions

$$T_0 = Y_3'Y_2'Y_1'Y_0' + Y_3Y_2Y_1Y_0$$

$$T_1 = Y_3'Y_2'Y_1'Y_0 + Y_3Y_2Y_1Y_0'$$

$$T_2 = Y_3'Y_2'Y_1Y_0 + Y_3Y_2Y_1'Y_0'$$

$$T_3 = Y_3'Y_2Y_1Y_0 + Y_3Y_2'Y_1'Y_0'$$

$$Q_0 = Y_0 \quad ; \quad Q_1 = Y_1$$

$$Q_2 = Y_2 \quad ; \quad Q_3 = Y_3$$

Step 4 : Complete the circuit entry, Figure 6.38, by using graphic editor in MAX+PLUS II

Step 5: Complete the circuit functional simulation by using MAX+PLUS II, Figure 6.39 illustrates the result of functional simulation, and check whether the functions meet the specification. Go on the next step if meet the specification; otherwise go back to step 1 to check the cause of error sequentially.

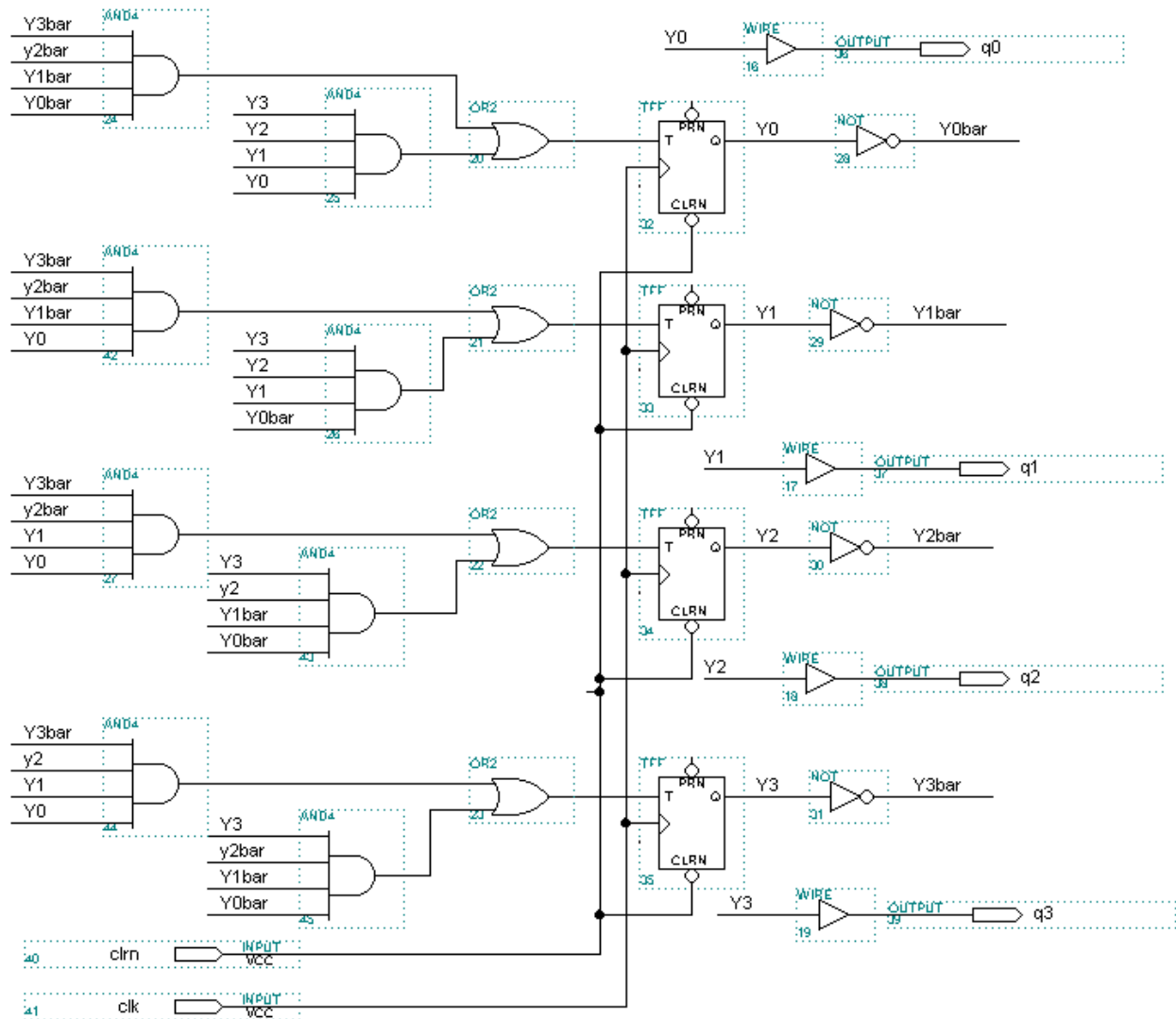


Figure 6.38 Complete the circuit entry of Table 6.31a by using MAX+PLUS II
(document : JSCNTR4.GDF)

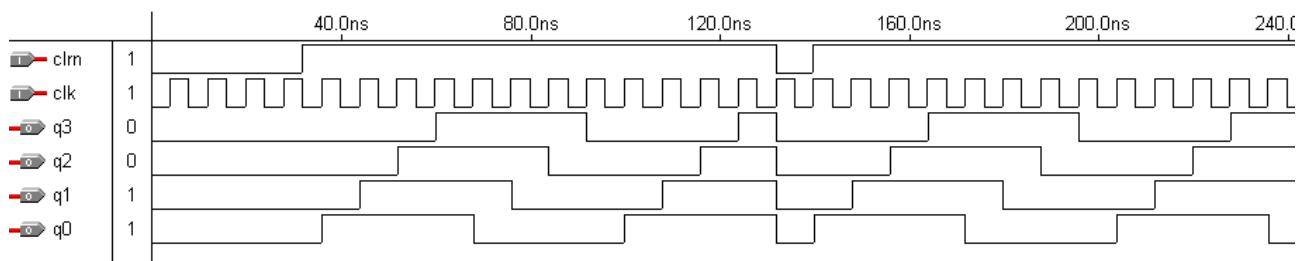


Figure 6.39 The functional simulation results of Table 6.31a by using
MAX+PLUS II simulator (document: JSCNTR4.SCF)

Step 6 : If the circuit allow to test by downloading (programming), select download (programming) chip and then floorplan.

As circuit modified that showed in Figure 5.3 of Section 5.1, please modify Figure 6.38. Please re-compile it after modifying, and adapt the ploorplan techniques in Section 4.6, select chip EPF10K10TC144-4 and use Table 6.32 pin assignment reference. After assemble logic circuit design Lab platform LP-2900, download four-bit Johnson counter to chip EPF10K10TC144-4. Please try to push SW1 (CLRN) then push PS1 on left-bottom of LP-2900. Please note the changes of L1 (Q3), L2 (Q2), L3 (Q1) and L4 (Q0), and see if cyclic like as follows, 0000 (0) → 0001 (1) → 0011 (3) → 0111 (7) → 1111 (F) → 1110 (E) → 1100 (C) → 1000 (8) → 0000 (0) → 0001 (1) → ...

Table 6.32 Pin assignment of EPF10K10TC144-4

Name of Signal	Pin of EPF10K10TC144-4	Name of Signal	Pin of EPF10K10TC144-4
CLKIN	Pin 54 (PS1)	Q2	Pin 8
CLRN	Pin 47	Q3	Pin 7
Q0	Pin 10		
Q1	Pin 9	LED_COM	Pin 141

Exercise 17 : Please design sequential circuit of Four-bit Johnson counter by using JK Flip-flops.

Exercise 18 : Please design sequential circuit of Four-bit Johnson counter by using D type Flip-flops.

6.5 The Design, Simulation and Test of Asynchronous Counter

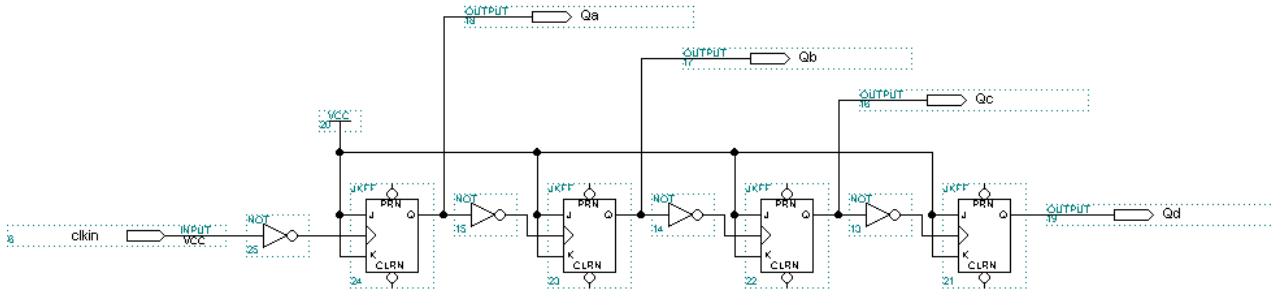
The asynchronous sequential circuit, sequential circuit of clock of Flip-flops doesn't connect together. So Flip-flops doesn't change state in same time. Frequently, primary stage output (positive edge or negative edge) becomes next stage clock input. Therefore, asynchronous sequential circuit doesn't have systematic procedure of design. It leads to asynchronous sequential circuit unusual applied, we will introduce common asynchronous count circuit as follows

6.5.1 Asynchronous Four-bit Binary Counter

Apply JK Flip-flops changing state function, when $J=K=1$ clock enters, illustrated in Table 6.14c. The proceed of asynchronous counter design is as follow

Step 1: design circuit and complete circuit entry by using graphic editor of MAX+PLUS II. We can design Four-bit binary counter as Figure 6.40. The first JK Flip-flops be clocked by clkin negative edge, the second JK Flip-flops be clocked by QA, inverse of first JK Flip-flops output, the third JK Flip-flops be clocked by QB, inverse of second JK Flip-flops output, the fourth JK Flip-flops be clocked by QC, inverse of third JK Flip-flops output. So, QA wants to complete a cyclic, it needs two CLKIN negative edges (two CLKIN clocks), QB wants to complete a cyclic, it needs two QA negative edges (two QA clocks), QC wants to complete a clock, it needs two QB negative edges (two QB clocks), QD wants to complete a clock, it needs two QC negative edges (two QC clocks), in other words, if QD wants to complete a clock it needs $2 \times 2 \times 2 \times 2$ CLKIN negative edge (also 16 CLKIN clocks). It is four-bit binary up counter.

Figure 6.40 Four-bit binary up counter (documents : A4UCNTR.GDF) ◦



Step 2 : Complete the Figure 6.39 circuit functional simulation by using MAX+PLUS II and check whether the functions meet the specification. Go on the next step if meet the specification; otherwise go back to step 1 to check the cause of error sequentially. Figure 6.41 is simulation result of four-bit binary up counter. Propagation delay of asynchronous clock is illustrated as Figure 6.42. Figure 6.43 is four-bit binary down counter. Figure 6.44 is simulation result of four-bit binary down counter. Please compare Figure 6.40 with Figure, you can see that the inverse of JK Flip-flops outputs be considered as down counter.

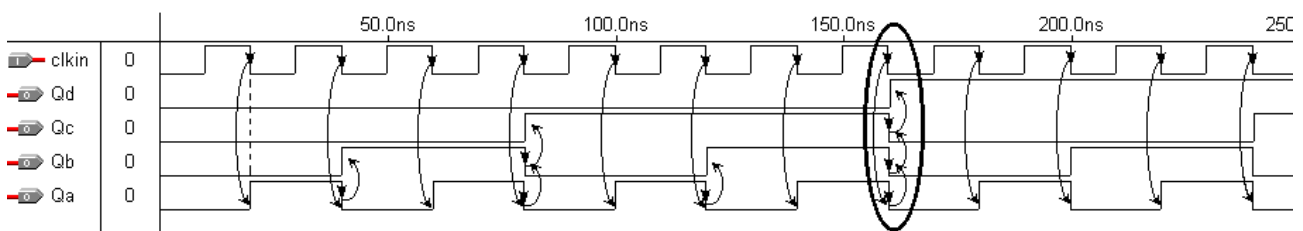


Figure 6.41 The simulation result of four-bit binary up counter (document : A4UCNTR.SCF)

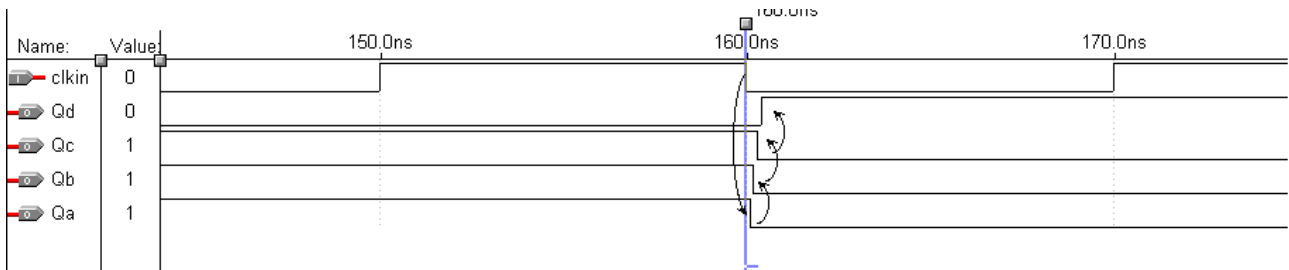


Figure 6.42 The magnified diagrams of cyclic of Figure 6.41

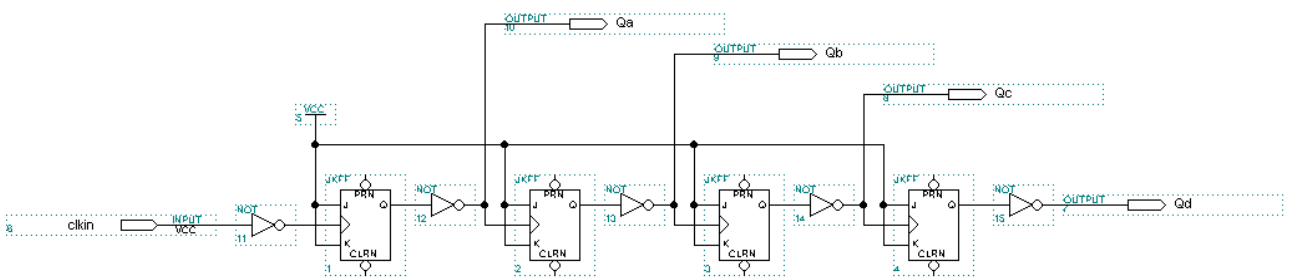


Figure 6.43 Four-bit binary down counter(document : A4DCNTR.GDF)

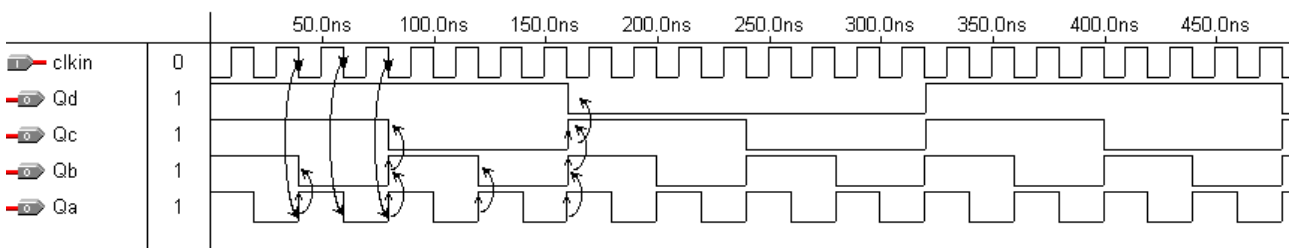


Figure 6.44 simulation result of four-bit binary down counter (document : A4DCNTR.SCF)

Step 3 : If the circuit allow to test by downloading (programming), select download (programming) chip and then floorplan.

As circuit modified that showed in Figure 5.3 of Section 5.1, please modify Figure 6.40. Please re-compile it after modifying, and adapt the ploorplan techniques in

Section 4.6, select chip EPF10K10TC144-4 and use Table 6.33 pin assignment reference. After assemble logic circuit design Lab platform LP-2900, download four-bit binary up (down) counter to chip EPF10K10TC144-4. Please try to push SW1 (CLR_N) then push PS1 on left-bottom of LP-2900. Please note the changes of L1 (Q₃), L2 (Q₂), L3 (Q₁) and L4 (Q₀), and to see if 0000 (0) → 0001 (1) → 0010 (2) → 0011 (3) → 0100 (4) → 0101 (5) → 0110 (6) → 0111 (7) → 1000 (8) → ... keeps cycling by binary count.

Table 6.33 Pin assignment of EPF10K10TC144-4

Name of Signal	Pin of EPF10K10TC144-4	Name of Signal	Pin of EPF10K10TC144-4
CLKIN	Pin 54 (PS1)	Q ₀	Pin 10
LED_COM	Pin 141	Q ₁	Pin 9
		Q ₂	Pin 8
		Q ₃	Pin 7

6.5.2 Asynchronous BCD Counter

The design procedures of asynchronous BCD counter are as follows.

Step 1: design circuit and complete circuit entry by using graphic editor of MAX+PLUS II. We can design BCD counter as Figure 6.45. The first JK Flip-flops is clocked by clkin negative edge, the second JK Flip-flops be clocked by QA, inverse of first JK Flip-flops, the third JK Flip-flops be clocked by QB, inverse of second JK Flip-flops, the fourth JK Flip-flops be clocked by QC, inverse of third JK Flip-flops. So, QA wants to complete a clock, it needs two CLKIN negative edges (two CLKIN clocks), QB wants

to complete a clock, it needs two QA negative edges (two QA clocks), QC wants to complete a clock, it needs two QB negative edges (two QB clocks), QD wants to complete a clock, it needs two QC negative edges (two QC clocks), in other words, if QD wants to complete a clock it needs $2 \times 2 \times 2 \times 2$ CLKIN negative edge (also 16 CLKIN clocks). But, the count comes to (QD, QC, QB, QA) = (1, 0, 1, 0), it will show one asynchronous clear and all come back to “0000”. The “1010” only shows briefly. It is BCD counter.

Step 2 : Complete the Figure 6.46 circuit entry and Figure 6.47 circuit functional simulation by using MAX+PLUS II and check weather the functions meet the specification. Go on the next step if meet the specification; otherwise go back to step 1 to check the cause of error sequentially.

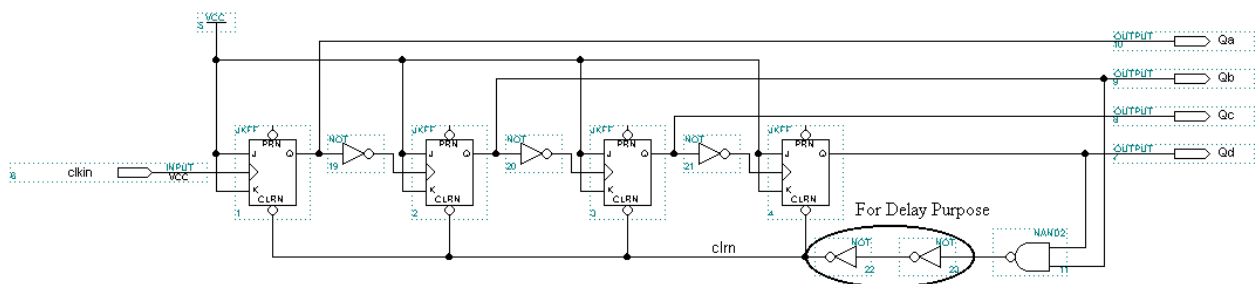


Figure 6.45 BCD counter(document : ABCDCNTR.GDF)

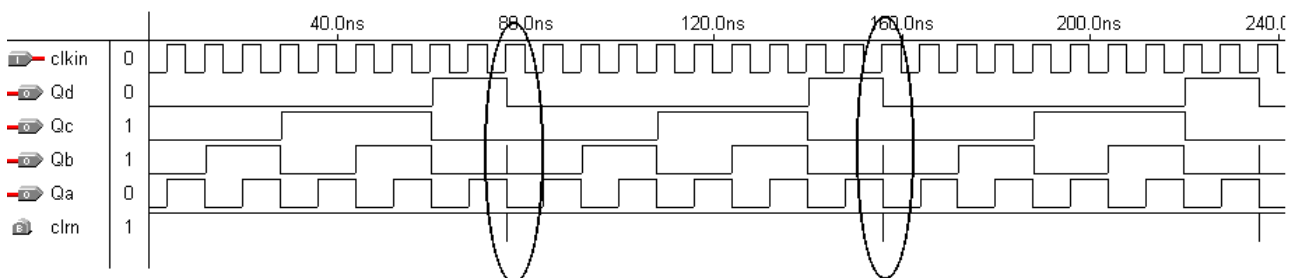


Figure 6.46 Simulation result of BCD counter(document : ABCDCNTR.SCF)

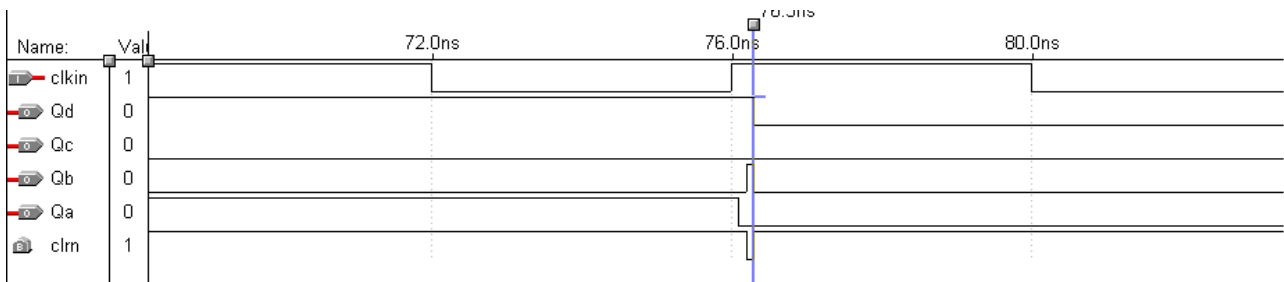


Figure 6.47 The magnified diagrams of cyclic of Figure 6.46 (document : ABCDCNTR.SCF) (“1010” shows briefly)

Step 3 : If the circuit allow to test by downloading (programming), select download (programming) chip and then floorplan.

As circuit modified that showed in Figure 5.3 of Section 5.1, please modify Figure 6.45. Please re-compile it after modifying, and adapt the ploorplan techniques in Section 4.6, select chip EPF10K10TC144-4 and use Table 6.34 pin assignment reference. After assemble logic circuit design Lab platform LP-2900, download SCD counter to chip EPF10K10TC144-4. Please try to push SW1 (CLRN) then push PS1 on left-bottom of LP-2900. Please note the changes of L1 (Q3), L2 (Q2), L3 (Q1) and L4 (Q0), and see if 0000 (0) → 0001 (1) → 0010 (2) → 0011 (3) → 0100 (4) → 0101 (5) → 0110 (6) → 0111 (7) → 1000 (8) → 1001 (9) → 0000 (0) → ... keeps cycling by BCD counter.

Table6.34 Pin assignment of EPF10K10TC144-4

Name of Signal	Pin of EPF10K10TC144-4	Name of Signal	Pin of EPF10K10TC144-4
CLKIN	Pin 54 (PS1)	Q1	Pin 9
LED_COM	Pin 141	Q2	Pin 8
Q0	Pin 10	Q3	Pin 7

6.5.3 Asynchronous Mod 14 Counter

The design procedures of asynchronous mod 14 counter are as follows

Step 1: design circuit and complete circuit entry by using graphic editor of MAX+PLUS II. We can design asynchronous mod 14 counter as Figure 6.48. The first JK Flip-flops is clocked by clkin negative edge, the second JK Flip-flops be clocked by QA, inverse of first JK Flip-flops, the third JK Flip-flops be clocked by QB, inverse of second JK Flip-flops, the fourth JK Flip-flops be clocked by QC, inverse of third JK Flip-flops. So, QA wants to complete a cyclic, it needs two CLKIN negative edges (two CLKIN clocks), QB wants to complete a clock, it needs two QA negative edges (two QA clocks), QC wants to complete a clock, it needs two QB negative edges (two QB clocks), QD wants to complete a clock, it needs two QC negative edges (two QC clocks), in other words, if QD wants to complete a clock it needs $2 \times 2 \times 2 \times 2$ CLKIN negative edge (also 16 CLKIN clocks). But, the count comes to (QD, QC, QB, QA) = (1, 1, 1, 0), it will show one asynchronous clear and all come back to “0000”. The “1110” only shows briefly. It is asynchronous mod 14 counter.

Step 2 : Complete the Figure 6.46 design entry and Figure 6.49 circuit functional simulation by using MAX+PLUS II and check weather the functions meet the specification. Go on the next step if meet the specification; otherwise go back to step 1 to check the cause of error sequentially.

Step 3 : If the circuit allow to test by downloading (programming), select download (programming) chip and then floorplan.

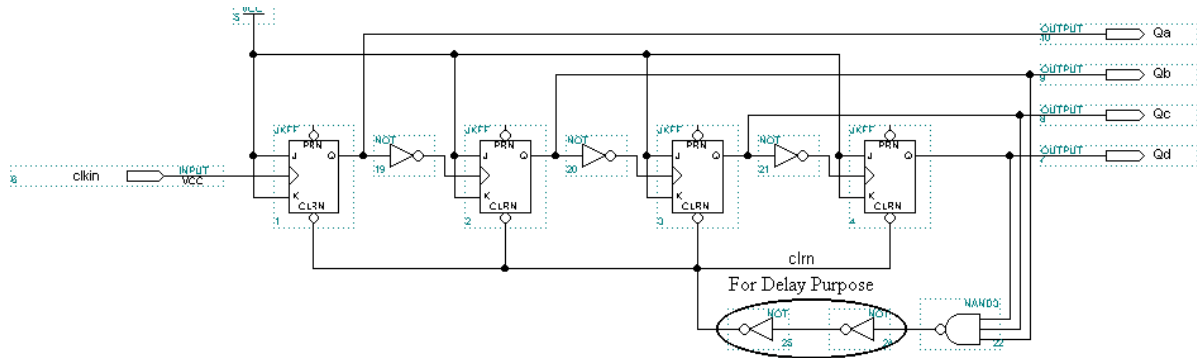


Figure 6.48 Asynchronous mod14 counter (document : Amod14.GDF)

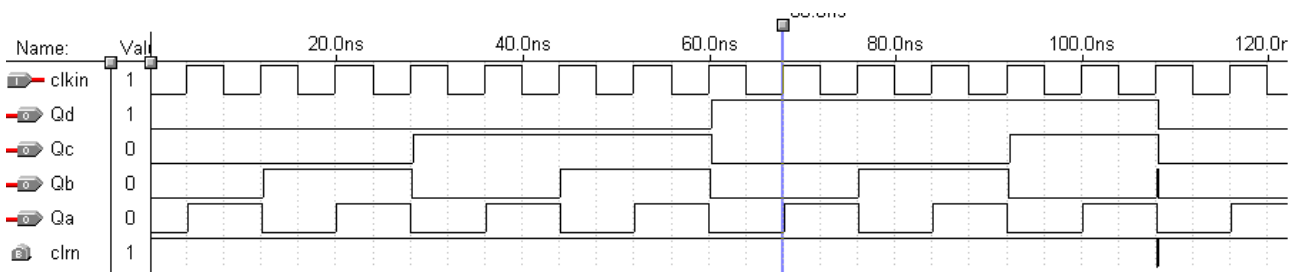


Figure 6.49 Simulation result of asynchronous mod14 counter (document : Amod14.SCF)

As circuit modified that showed in Figure 5.3 of Section 5.1, please modify Figure 6.48. Please re-compile it after modifying, and adapt the ploorplan techniques in Section 4.6, select chip EPF10K10TC144-4 and use Table 6.35 pin assignment reference. After assemble logic circuit design Lab platform LP-2900, download asynchronous mod 14 counter to chip EPF10K10TC144-4. Please try to push SW1 (CLR) then push PS1 on left-bottom of LP-2900. Please note the changes of L1 (Q3), L2 (Q2), L3 (Q1) and L4 (Q0), and see if 0000 (0) → 0001 (1) → 0010 (2) → 0011 (3) → 0100 (4) → 0101 (5) → 0110 (6) → 0111 (7) → 1000 (8) → ... → 1011 (13) → 0000 (0) → ... keeps cycling by MOD/4 count.

Table 6.35 Pin assignment of EPF10K10TC144-4

Name of Signal	Pin of EPF10K10TC144-4	Name of Signal	Pin of EPF10K10TC144-4
CLKIN	Pin 54 (PS1)	Q0	Pin 10
LED_COM	Pin 141	Q1	Pin 9
		Q2	Pin 8
		Q3	Pin 7

6.6 Evaluation

Please do the following evaluation according to the questions listed below,

- ☐ Do you know what does sequential logic circuit define?
- ☐ Do you know those are memory cell of sequential logic circuit?
- ☐ Do you know process of design, simulation and test of sequential logic circuit?
- ☐ Do you know what is Mealy state machine? What is Moore state machine?
- ☐ The sequential logic circuit is almost synchronous. Why?
- ☐ Can you design, simulate and test synchronous mod 13 counter?
- ☐ Can you design, simulate and test five-bit Johnson counter?

CHAPTER 7

SIMPLE DESIGN EXAMPLES



LEAP

In this chapter, some frequent-applied design examples will be illustrated allowing the readers to familiarize with MAX+PLUS II devices and tactfully apply the designing theories stated in Chapter 5 and Chapter 6. The examples of circuit's layout drawn in this chapter include Frequency Generator, Simple Electronic Dice, Counter, Simple Traffic Light Controller, Dot Matrix Display, Keyboard Scan and Display, and LCD Interface.

7.1 Frequency Generator

Various clock frequencies, from Hz, KHz to MHz, are required in the field of digital logic. The generation of diverse frequencies depends on the frequency divider of the main frequency, such as the generation of quartz oscillator. The design of divider, therefore, is quite important. In this section, the divider is designed individually as the following basic divider formats: $\div 2$, $\div 5$, $\div 10$ and $\div 50$. The readers, referring to this section, are invited to design other dividers by your own creation.

7.1.1 $\div 2$ Divider Design

Step 1 : Form a Truth table (Table 7.1a)

Table 7.1a State table of $\div 2$ divider's sequential circuit

Present State	Next State	Output
0	1	0
1	0	1

Step 2 : From the above Truth table (Table 7.1a) leads to the following Karnaugh Map and equation (Table 7.1b).



Table 7.1b Karnaugh Map of T, T = Vcc

T		
Present State Input	0	1
Y	1	1

Step 3 : Figure out the minimized equation of each input and output function.

$$T = VCC$$

Step 4 : Please use the Graphic editor of MAX+PLUS II to complete editing the circuit entry of the above equation (Figure 7.1a).

Step 5 : Complete functional simulation by using Simulator in MAX+PLUS II and test whether the functions meet the circuit specifications (Table 7.1b). Go on to the next step if the functions meet the specifications; otherwise go back to step 1 to check the cause of error sequentially.

Step 6 : Should circuits allow download testing, download (or programming) to test the circuit after selecting the download (or programming) device and the floorplan program;

~Skip this download test~

Stop 7: Please refer to File > Create Default Symbol to generate the internal circuit symbol of ÷2 divider for the upper layer circuit.

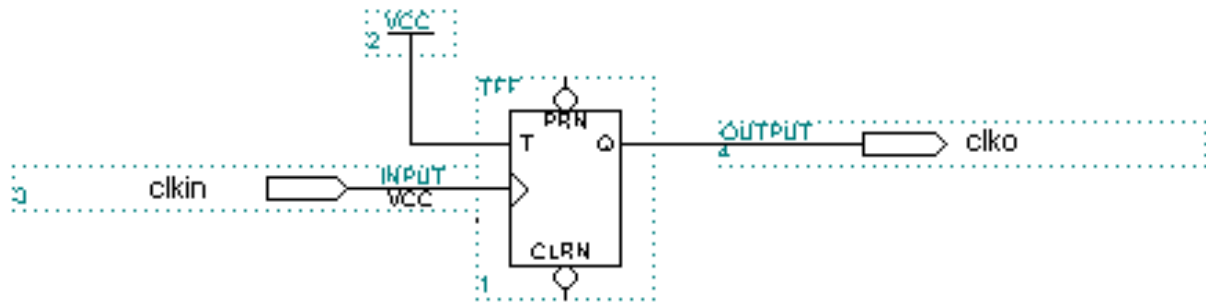


Figure 7.1a $\div 2$ divider circuit entry by using graphic editor in MAX+PLUS II
(File : div2.gdf)

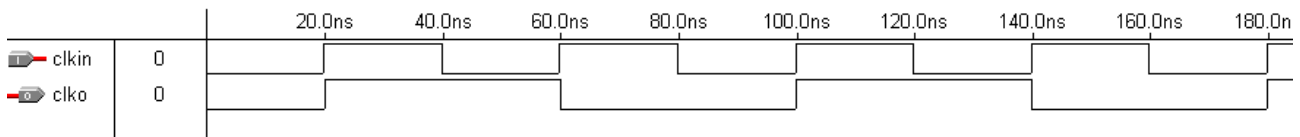


Figure 7.1b Simulation result of div2.gdf (File : div2.scf)

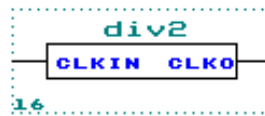


Figure 7.1c The symbol of $\div 2$ divider circuit

7.1.2 $\div 5$ Divider Design

Step 1 : Form a Truth table. There are five states, 0, 1, 2, 3 and 4, requiring 3-bit, $T_2T_1T_0$ to signify the state value (Table 7.2a).

Step 2: From the above Truth table (Table 7.2a) leads to the following Karnaugh Map and equations (Table 7.2b~7.2e).

Table 7.2a State table of ÷5 divider's sequential circuit

Current State	Next State	Output
000	001	0
001	010	0
010	011	1
011	100	1
100	000	1

Table 7.2b Karnaugh Map of T_0 , $T_0 = Y_2' + Y_1'Y_0'$

T_0		Present State Inputs Y_1Y_0			
		00	01	11	10
Present State Inputs Y_2	0	1	1	1	1
	1	1			

Note : “/” represents the state would never occur, so as in the following talbes.

Table 7.2c Karnaugh Map of T_1 , $T_1 = Y_2'Y_0$

T_1		Present State Inputs Y_1Y_0			
		00	01	11	10
Present State Inputs Y_2	0	0	1	1	0
	1	0			

Table 7.2d Karnaugh Map of T_2 , $T_2 = Y_2'Y_1Y_0 + Y_2Y_1'Y_0'$

T_2		Present State Inputs Y_1Y_0			
		00	01	11	10
Present State Inputs Y_2	0	0	0	1	0
	1	1			

Table 7.2e Karnaugh Map of Z, $Z = Y_2'Y_1Y_0' + Y_2Y_1'Y_0'$

Z		Present State Inputs Y_1Y_0			
		00	01	11	10
Present State Inputs Y_2	0	0	0	0	1
	1	1			

Step 3: Figure out each minimized input and output functions.

$$T_0 = Y_2' + Y_1'Y_0'$$

$$T_1 = Y_2'Y_0$$

$$T_2 = Y_2'Y_1Y_0 + Y_2Y_1'Y_0'$$

$$Z = Y_2'Y_1Y_0' + Y_2Y_1'Y_0'$$

Step 4: Please use the Graphic editor of MAX+PLUS II to complete the circuit entry of the above equations (Figure 7.2a) .

Step 5: Complete functional simulations by using Simulator in MAX+PLUS II and test whether the functions meet the circuit specifications (Table 7.2b). Go on to the next step if the functions meet the specifications; otherwise, go back to step 1 to check the cause of error sequentially.

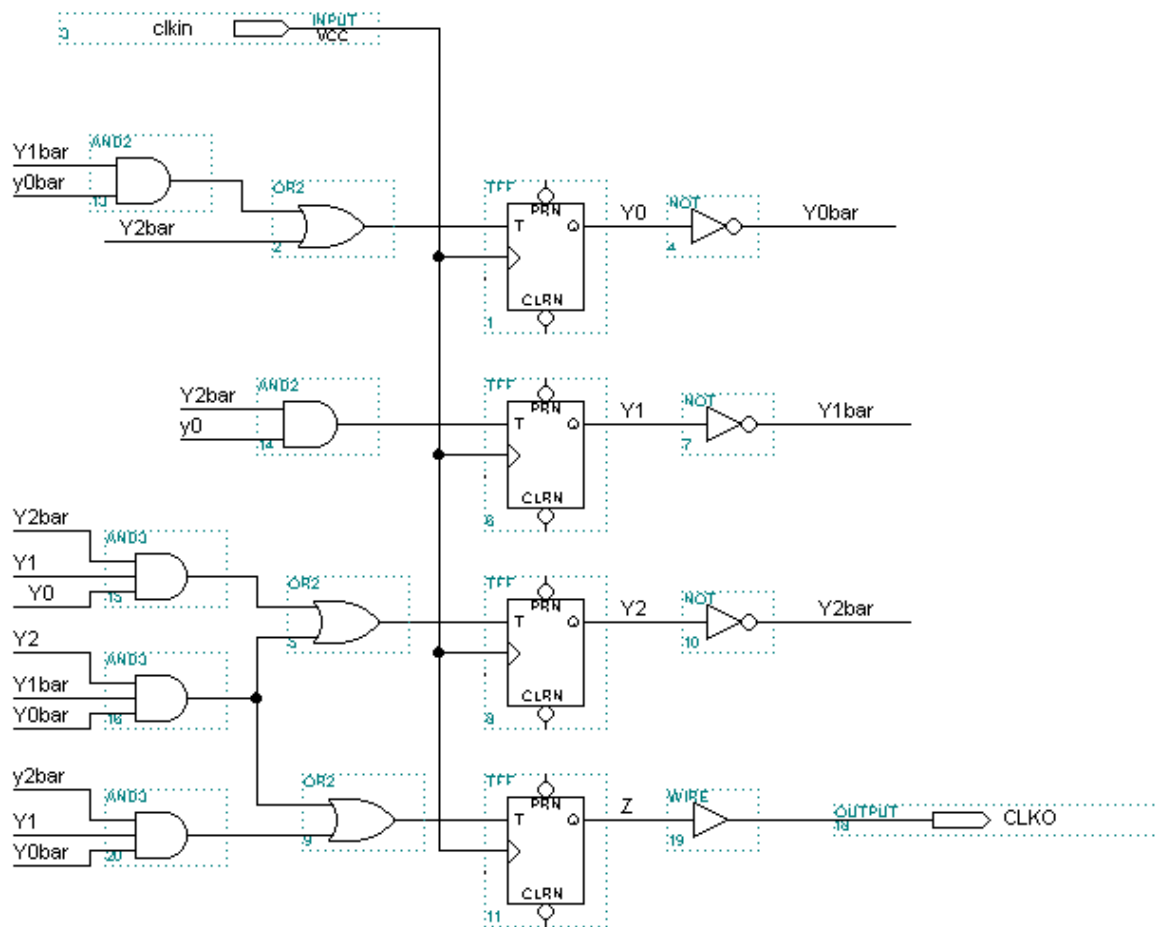


Figure 7.2a Using MAX+PLUS II Graphic editor to create $\div 5$ divide circuit
(File: div5.gdf)

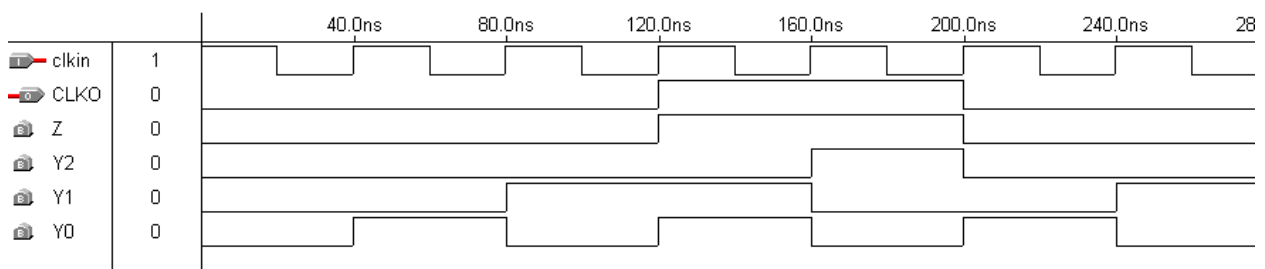


Figure 7.2b div5.gdf simulation result (File: div5.scf)

Step 6: Should circuits allow download testing, download (or programming) to test the circuit after selecting the download (or programming) device and

the floorplan program;

~Skip this download test~

Step 7: Please refer to File>Create Default Symbol to generate the internal circuit symbol of ÷5 divider for the upper layer circuit.

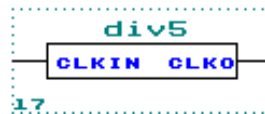


Figure 7.2c The symbol of ÷5 divider

7.1.3 ÷ 10 Divider Design

Step 1: Form a Truth table. There are 10 states, 0, 1, 2, 3, 4,... and 9, requiring a 4-bit, $T_3T_2T_1T_0$ to signify the state value (Table 7.3a) ◦

Step 2: From the above Truth table (Table 7.3a) leads to the following Karnaugh Map and equations. (Table 7.3b~7.3f) ◦

Table 7.3a State table of ÷10 divide's sequential circuit

Present State	Next State	Output
0000	0001	0
0001	0010	0
0010	0011	0
0011	0100	0
0100	0101	0
0101	0110	1
0110	0111	1

0111	1000	1
1000	1001	1
1001	0000	1

Table 7.3b Karnaugh Map of T_0 , $T_0 = Y_3' + Y_2'Y_1'$

T_0		Present State Inputs Y_1Y_0			
		00	01	11	10
Present State Inputs Y_3Y_2	00	1	1	1	1
	01	1	1	1	1
	11				
	10	1	1		

Table 7.3c Karnaugh Map of T_1 , $T_1 = Y_3'Y_0$

T_1		Present State Inputs Y_1Y_0			
		00	01	11	10
Present State Inputs Y_3Y_2	00	0	1	1	0
	01	0	1	1	0
	11				
	10	0	0		

Table 7.3d Karnaugh Map of T_2 , $T_2 = Y_3'Y_1Y_0$

T_2		Present State Inputs Y_1Y_0			
		00	01	11	10
Present State Inputs Y_3Y_2	00	0	0	1	0
	01	0	0	1	0
	11				
	10	0	0		

Table 7.3e Karnaugh Map of T_3 , $T_3 = Y_3'Y_2Y_1Y_0 + Y_3Y_2'Y_1'Y_0$

T_3		Present State Inputs Y_1Y_0			
		00	01	11	10
Present State Inputs Y_3Y_2	00	0	0	0	0
	01	0	0	1	0
	11				
	10	0	1		

Table 7.3f Karnaugh Map of Z , $Z = Y_3'Y_2Y_1'Y_0' + Y_3Y_2'Y_1'Y_0$

Z		Present State Inputs Y_1Y_0			
		00	01	11	10
Present State Inputs Y_3Y_2	00	0	0	0	0
	01	1	0	0	0
	11				
	10	0	1		

Step 3: Figure out each minimized input and output functions.

$$T_0 = Y_3' + Y_2'Y_1'$$

$$T_1 = Y_3'Y_0$$

$$T_2 = Y_3'Y_1Y_0$$

$$T_3 = Y_3'Y_2Y_1Y_0 + Y_3Y_2'Y_1'Y_0$$

$$Z = Y_3'Y_2Y_1'Y_0' + Y_3Y_2'Y_1'Y_0$$

Step 4: Please use the Graphic editor of MAX+PLUS II to complete the circuit entry of the above equations (Figure 7.3a) ◦

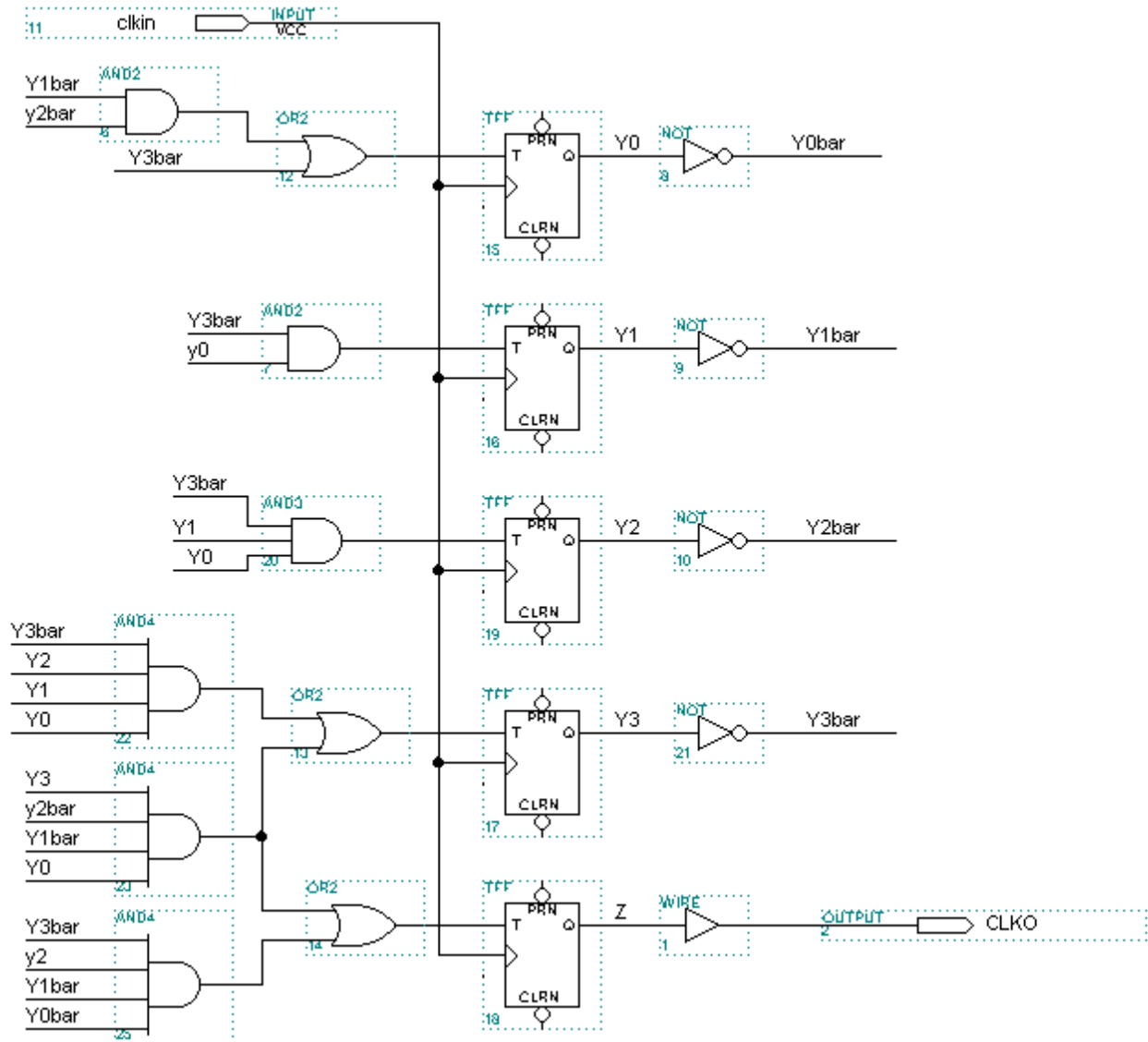


Figure 7.3a Using MAX+PLUS II Graphic editor to create ÷10 divide circuit
(File: div10.gdf)

Step 5: Complete functional simulation by using Simulator in MAX+PLUS II and test whether the functions meet the circuit specifications (Table 7.3b). Go on to the next step the functions meet the specifications; otherwise, go back to step 1 to check the cause of error sequentially.

Step 6: If the circuits allow download testing, download (or programming) to test the circuit after selecting the download (or programming) device and floorplan program.

~Skip this download test~

Step 7: Please refer to File > Create Default Symbol to generate the internal circuit symbol of $\div 10$ circuit for the upper layer circuit. (Figure 7.3c)

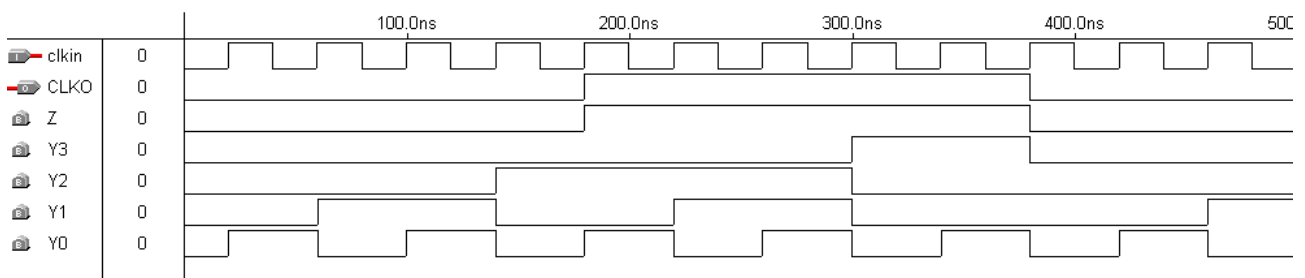


Figure 7.3b Simulation Result of div10.gdf (File: div10.scf)

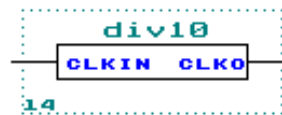


Figure 7.3c Internal Circuit symbol of $\div 10$ divide circuit

7.1.4 $\div 50$ Divider Design

Step 1: Form a Truth table. There are 50 states, 0, 1, 2, 3, 4, ...and 49, requiring a 6-bit, $T_5T_4T_3T_2T_1T_0$ to signify the state value (Table 7.4a) .

Step 2: From the above Truth table (Table 7.4a) leads to the following Karnaugh Map and equations. (Table 7.4b~Table 7.4h) .

Table 7.4a State table of ÷50 divider's sequential circuit

Present State	Next State	Output
000000	000001	0
000001	000010	0
000010	000011	0
000011	000100	0
000100	000101	0
...
011000	011001	0
011001	011010	1
...
110001	000000	1

Table 7.4b Karnaugh Map of T_0 , $T_0 = Y_5' + Y_5Y_4' + Y_5Y_3'Y_2'Y_1'$

$$Y_5 = 0$$

T_0			Present State Inputs Y_1Y_0			
			00	01	11	10
Present State Inputs Y_3Y_2	00	$Y_4 = 0$	1	1	1	1
		$Y_4 = 1$	1	1	1	1
	01	$Y_4 = 0$	1	1	1	1
		$Y_4 = 1$	1	1	1	1
	11	$Y_4 = 0$	1	1	1	1
		$Y_4 = 1$	1	1	1	1
	10	$Y_4 = 0$	1	1	1	1
		$Y_4 = 1$	1	1	1	1



$$Y_5 = 1$$

T_0			Present State Inputs Y_1Y_0			
			00	01	11	10
Present State Inputs Y_3Y_2	00	$Y_4 = 0$	1	1	1	1
		$Y_4 = 1$	1	1	/	/
	01	$Y_4 = 0$	1	1	1	1
		$Y_4 = 1$	/	/	/	/
	11	$Y_4 = 0$	1	1	1	1
		$Y_4 = 1$	/	/	/	/
	10	$Y_4 = 0$	1	1	1	1
		$Y_4 = 1$	/	/	/	/

Table 7.4c Karnaugh Map of T_1 , $T_1 = Y_5'Y_0 + Y_5Y_4'Y_0$

$$Y_5 = 0$$

T_1			Present State Inputs Y_1Y_0			
			00	01	11	10
Present State Inputs Y_3Y_2	00	$Y_4=0$	0	1	1	0
		$Y_4=1$	0	1	1	0
	01	$Y_4=0$	0	1	1	0
		$Y_4=1$	0	1	1	0
	11	$Y_4=0$	0	1	1	0
		$Y_4=1$	0	1	1	0
	10	$Y_4=0$	0	1	1	0
		$Y_4=1$	0	1	1	0



$$Y_5 = 1$$

T_1			Present State Inputs Y_1Y_0			
			00	01	11	10
Present State Inputs Y_3Y_2	00	$Y_4 = 0$	0	1	1	0
		$Y_4 = 1$	0	0	/	/
	01	$Y_4 = 0$	0	1	1	0
		$Y_4 = 1$	/	/	/	/
	11	$Y_4 = 0$	0	1	1	0
		$Y_4 = 1$	/	/	/	/
	10	$Y_4 = 0$	0	1	1	0
		$Y_4 = 1$	/	/	/	/

Table 7.4d Karnaugh Map of T_2 , $T_2 = Y_5'Y_1Y_0 + Y_5Y_4'Y_1Y_0$

$$Y_5 = 0$$

T_2			Present State Inputs Y_1Y_0			
			00	01	11	10
Present State Inputs Y_3Y_2	00	$Y_4 = 0$	0	0	1	0
		$Y_4 = 1$	0	0	1	0
	01	$Y_4 = 0$	0	0	1	0
		$Y_4 = 1$	0	0	1	0
	11	$Y_4 = 0$	0	0	1	0
		$Y_4 = 1$	0	0	1	0
	10	$Y_4 = 0$	0	0	1	0
		$Y_4 = 1$	0	0	1	0

$$Y_5 = 1$$

T_2			Present State Inputs Y_1Y_0			
			00	01	11	10
Present State Inputs Y_3Y_2	00	$Y_4=0$	0	0	1	0
		$Y_4=1$	0	0	/	/
	01	$Y_4=0$	0	0	1	0
		$Y_4=1$	/	/	/	/
	11	$Y_4=0$	0	0	1	0
		$Y_4=1$	/	/	/	/
	10	$Y_4=0$	0	0	1	0
		$Y_4=1$	/	/	/	/

Table 7.4e Karnaugh Map of T_3 , $T_3 = Y_5'Y_2Y_1Y_0 + Y_5Y_4'Y_2Y_1Y_0$

$$Y_5 = 0$$

T_3			Present State Inputs Y_1Y_0			
			00	01	11	10
Present State Inputs Y_3Y_2	00	$Y_4=0$	0	0	0	0
		$Y_4=1$	0	0	0	0
	01	$Y_4=0$	0	0	1	0
		$Y_4=1$	0	0	1	0
	11	$Y_4=0$	0	0	1	0
		$Y_4=1$	0	0	1	0
	10	$Y_4=0$	0	0	0	0
		$Y_4=1$	0	0	0	0



$$Y_5 = 1$$

T_3			Present State Inputs Y_1Y_0			
			00	01	11	10
Present State Inputs Y_3Y_2	00	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$	0	0		
	01	$Y_4 = 0$	0	0	1	
		$Y_4 = 1$				
	11	$Y_4 = 0$	0	0	1	0
		$Y_4 = 1$				
	10	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$				

Table 7.4f Karnaugh Map of T_4 , $T_4 = Y_5'Y_3Y_2Y_1Y_0 + Y_5Y_4'Y_3Y_2Y_1Y_0 + Y_5Y_4Y_3'Y_2'Y_1'Y_0$

$$Y_5 = 0$$

T_4			Present State Inputs Y_1Y_0			
			00	01	11	10
Present State Inputs Y_3Y_2	00	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$	0	0	0	0
	01	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$	0	0	0	0
	11	$Y_4 = 0$	0	0	1	0
		$Y_4 = 1$	0	0	1	0
	10	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$	0	0	0	0

$$Y_5 = 1$$

T_4			Present State Inputs Y_1Y_0			
			00	01	11	10
Present State Inputs Y_3Y_2	00	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$	0	1		
	01	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$				
	11	$Y_4 = 0$	0	0	1	0
		$Y_4 = 1$				
	10	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$				

Table 7.4g Karnaugh Map of T_5 , $T_5 = Y_5'Y_4Y_3Y_2Y_1Y_0 + Y_5Y_4Y_3'Y_2'Y_1'Y_0$

$$Y_5 = 0$$

T_5			Present State Inputs Y_1Y_0			
			00	01	11	10
Present State Inputs Y_3Y_2	00	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$	0	0	0	0
	01	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$	0	0	0	0
	11	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$	0	0	1	0
	10	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$	0	0	0	0

$$Y_5 = 1$$

T_5			Present State Inputs Y_1Y_0			
			00	01	11	10
Present State Inputs Y_3Y_2	00	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$	0	1		
	01	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$				
	11	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$				
	10	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$				

Table 7.4h Karnaugh Map of Z , $Z = Y_5'Y_4Y_3Y_2'Y_1'Y_0' + Y_5Y_4Y_3'Y_2'Y_1'Y_0$

$$Y_5 = 0$$

Z			Present State Inputs Y_1Y_0			
			00	01	11	10
Present State Inputs Y_3Y_2	00	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$	0	0	0	0
	01	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$	0	0	0	0
	11	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$	0	0	0	0
	10	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$	1	0	0	0

$Y_5 = 1$

Z			Present State Inputs Y_1Y_0			
			00	01	11	10
Present State Inputs Y_1Y_2	00	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$	0	1		
	01	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$				
	11	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$				
	10	$Y_4 = 0$	0	0	0	0
		$Y_4 = 1$				

Step 3: Figure out each minimized input and output functions.

$$T_0 = Y_5' + Y_5Y_4' + Y_5Y_3'Y_2'Y_1'$$

$$T_1 = Y_5'Y_0 + Y_5Y_4'Y_0$$

$$T_2 = Y_5'Y_1Y_0 + Y_5Y_4'Y_1Y_0$$

$$T_3 = Y_5'Y_2Y_1Y_0 + Y_5Y_4'Y_2Y_1Y_0$$

$$T_4 = Y_5'Y_3Y_2Y_1Y_0 + Y_5Y_4'Y_3Y_2Y_1Y_0 + Y_5Y_4Y_3'Y_2'Y_1'Y_0$$

$$T_5 = Y_5'Y_4Y_3Y_2Y_1Y_0 + Y_5Y_4Y_3'Y_2'Y_1'Y_0$$

$$Z = Y_5'Y_4Y_3Y_2'Y_1'Y_0' + Y_5Y_4Y_3'Y_2'Y_1'Y_0$$

Step 4: Please use the Graphic editor of MAX+PLUS II to complete the circuit entry of the above equations (Figure 7.4a) ◦

Step 5: Complete functional simulation by using Simulator in MAX+PLUS II and

test whether the functions meet the circuit specifications (Figure 7.4b and Figure 7.4c). Go on to the next step if the functions meet the specifications; otherwise go back to step 1 to check the cause of error sequentially.

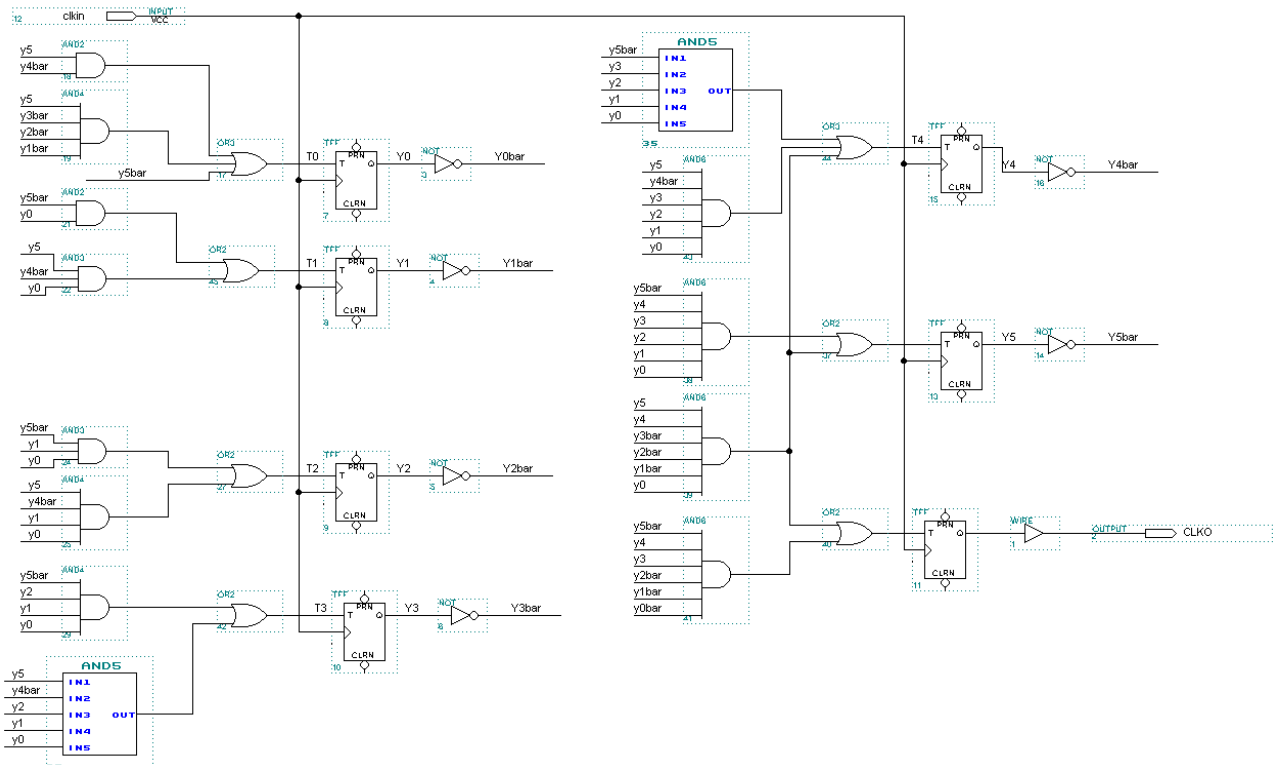


Figure 7.4a Using MAX+PLUS II Graphic editor to create $\div 50$ divider
(File: div50.gdf)

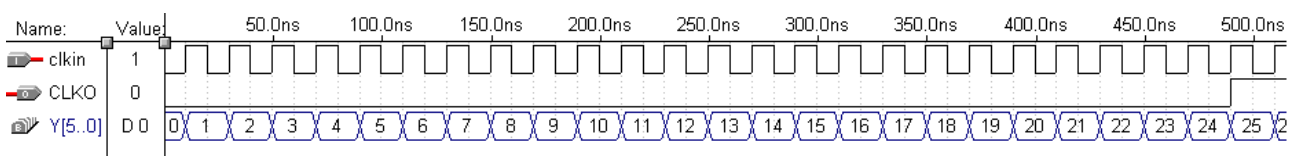


Figure 7.4b Simulation result of div50.gdf (File: div50.scf)

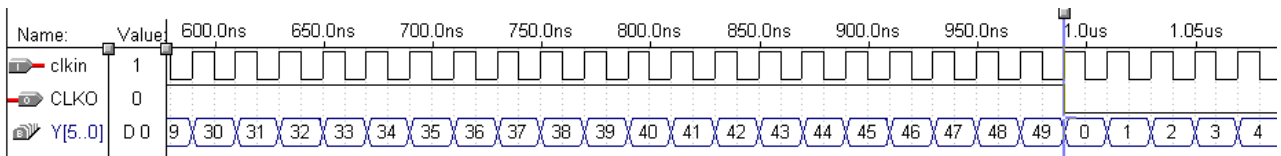


Figure 7.4c Simulation result of div50.gdf (File: div50.scf)

Step 6: If the circuits allow download testing, download (or programming) to test the circuit after selecting the download (or programming) device and floorplan program.

~Skip this download test~

Step 7: Please refer to File > Create Default Symbol to create the internal circuit symbol of ÷50 circuit for the upper layer circuit (Figure 7.4d).

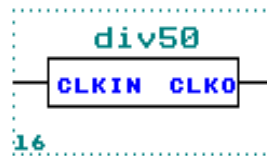


Figure 7.4d Internal circuit symbol of ÷50 divide circuit

7.1.5 Frequency Generator

In this section, we will use the circuits designed in the previous sections, to divide the frequency of 10 MHz to 1 MHz, 100 KHz, 10 KHz, 1 KHz, 100 Hz, 10 Hz and 1 Hz, as illustrated in Figure 7.5a. Figure 7.5b~7.5d are the simulation Results.

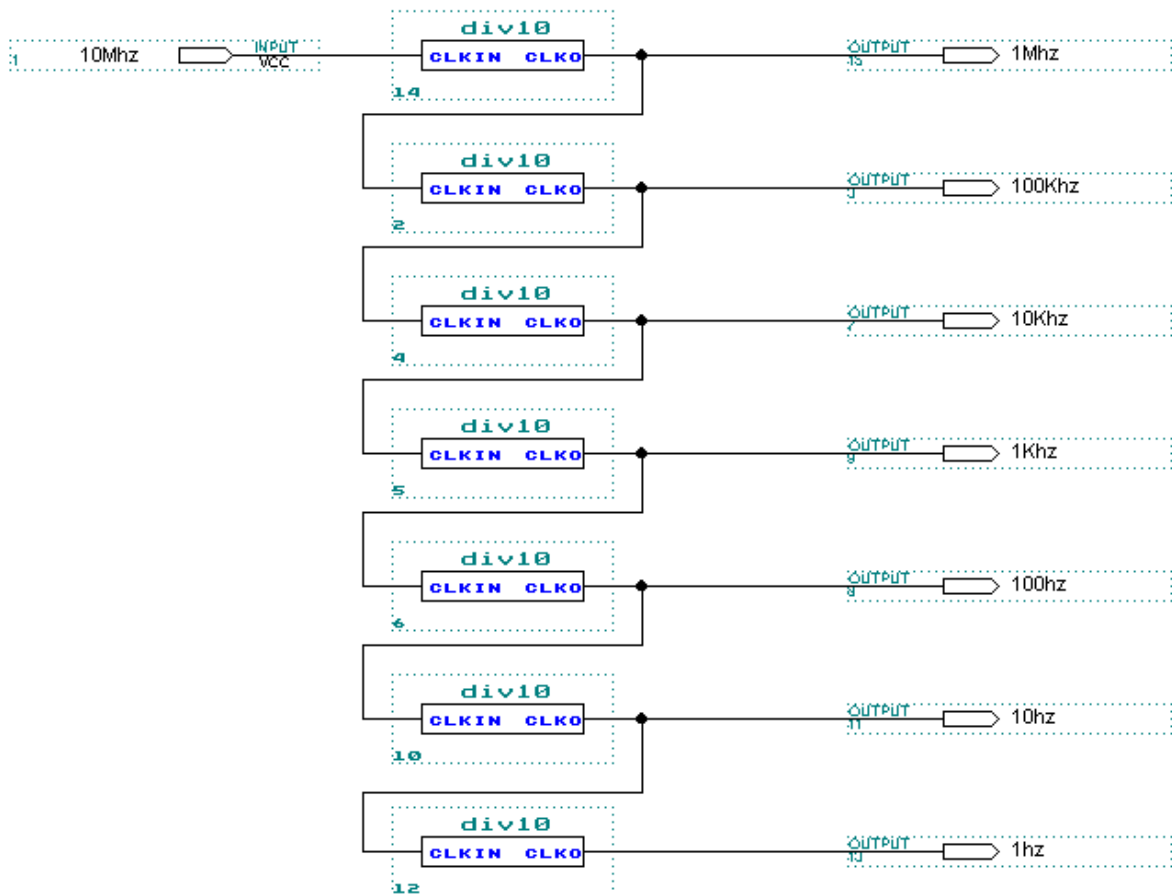


Figure 7.5a Using MAX+PLUS II Graphic editor to create frequency generator circuit (File: clkgen.gdf)

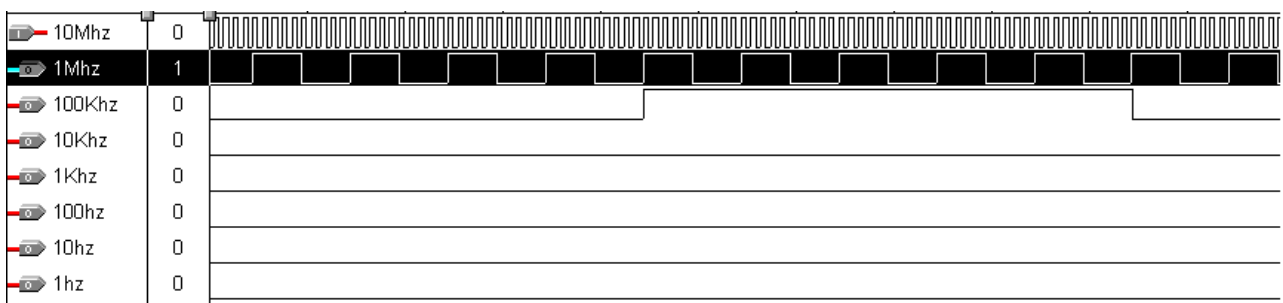


Figure 7.5b Frequency generator circuit simulation result of 1 MHz and 100 KHz (File: clkgen.scf)

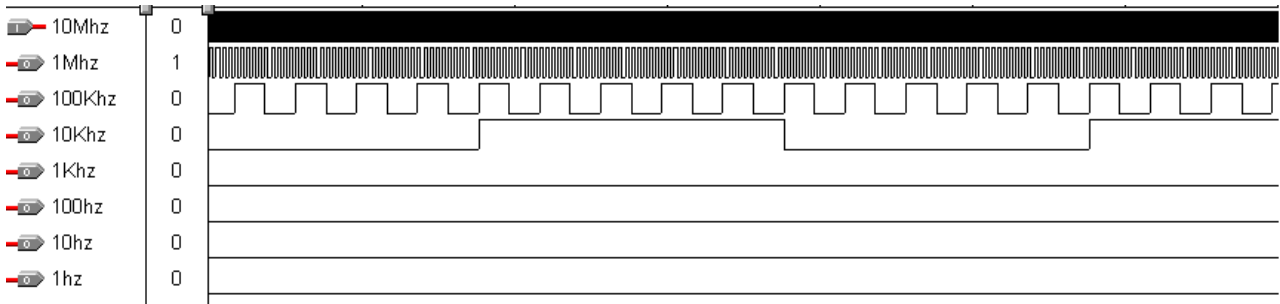


Figure 7.5c Frequency generator circuit simulation result of 10 KHz
(File: clkgen.scf)

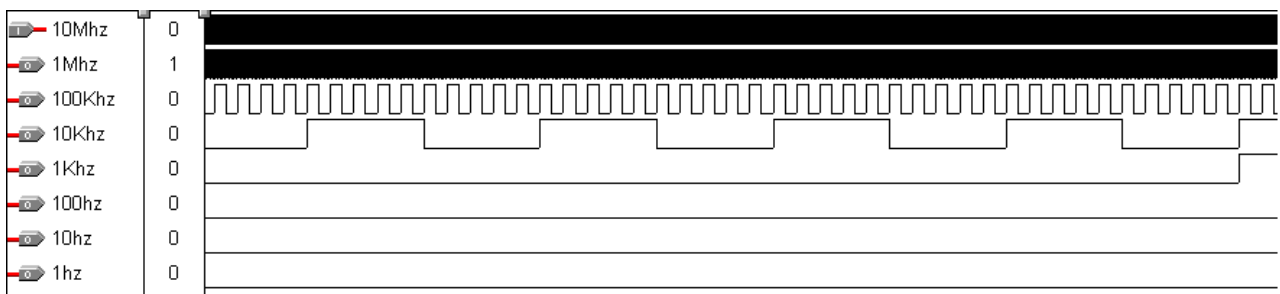


Figure 7.5d Frequency generator circuit simulation result of 1 KHz
(File: clkgen.scf)

7.2 Simple Electronic Dice

To design the circuits of a simple dice game, we need 3 types of sub-circuits (terms of hierarchical design, which means integrate the tiny circuits to compose a large circuit) dice decoder circuit, MOD6 counter and frequency generator. The frequency generator would divide the high frequency into two diverse frequencies, and differentiate to the MOD6 counter. The MOD6 counter would output to the dice decoder to show the points.

7.2.1 Dice Decoder Circuit

The Dice Decoder introduced here is a 6 to 7 decoder. The Dice Decoder Circuits have to apply the functions shown in Figure 7.6. Please complete the design, simulation, and verification of this decoder.

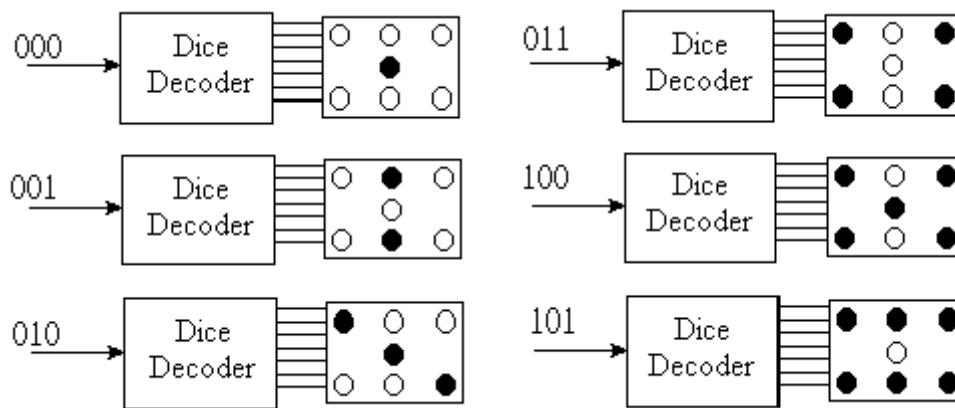


Figure 7.6 Functional code of dice decoder

Step 1: Form a Truth table (Table 7.5a).

Table 7.5a Truth table of dice decoder

Inputs	Outputs
$D_2D_1D_0$	$O_6 \sim O_0$
000	0001000
001	0100010
010	1001001
011	1010101
100	1011101
101	1110111



Step 2: The above Truth table leads to the following Karnaugh Matrix and equations. (Table 7.5b~Table 7.5h).

Table 7.5b Karnaugh Map of output O_0

O_0		D_1D_0			
		00	01	11	10
D_2	0	0	0	1	1
	1	1	1	0	0

Table 7.5c Karnaugh Map of output O_1

O_1		D_1D_0			
		00	01	11	10
D_2	0	0	1	0	0
	1	0	1	0	0

Table 7.5d Karnaugh Map of output O_2

O_2		D_1D_0			
		00	01	11	10
D_2	0	0	0	1	0
	1	1	1	0	0

Table 7.5e Karnaugh Map of output O_3

O_3		D_1D_0			
		00	01	11	10
D_2	0	1	0	0	1
	1	1	0	0	0

Table 7.5f Karnaugh Map of output O_4

O_4		D_1D_0			
		00	01	11	10
D_2	0	0	0	1	0
	1	1	1	0	0

Table 7.5g Karnaugh Map of output O_5

O_6		D_1D_0			
		00	01	11	10
D_2	0	0	1	0	0
	1	0	1	0	0

Table 7.5h Karnaugh Map of output O_6

O_6		D_1D_0			
		00	01	11	10
D_2	0	0	0	1	1
	1	1	1	0	0

Step 3: Figure out each minimized input and output functions.

$$\begin{aligned}
 O_0 &= D_2D_1' + D_2'D_1 & ; & & O_1 &= D_1'D_0 \\
 O_2 &= D_2D_1' + D_2'D_1D_0 & ; & & O_3 &= D_2'D_0' + D_1'D_0' \\
 O_4 &= O_2 & ; & & O_5 &= O_1 \\
 O_6 &= O_0
 \end{aligned}$$

Step 4: According to the above Boolean Equations, please use the proper logic gate by Graphic editor of MAX+PLUS II to create the circuit. (Figure 7.7a) .

Step 5: Complete the functional simulation and check weather the functions meet the specification. If the functions meet the specification, please create the internal circuit symbol for the upper circuit. Figure 7.7b is the simulation result of the dice decoder, and it meets the functional specifications.

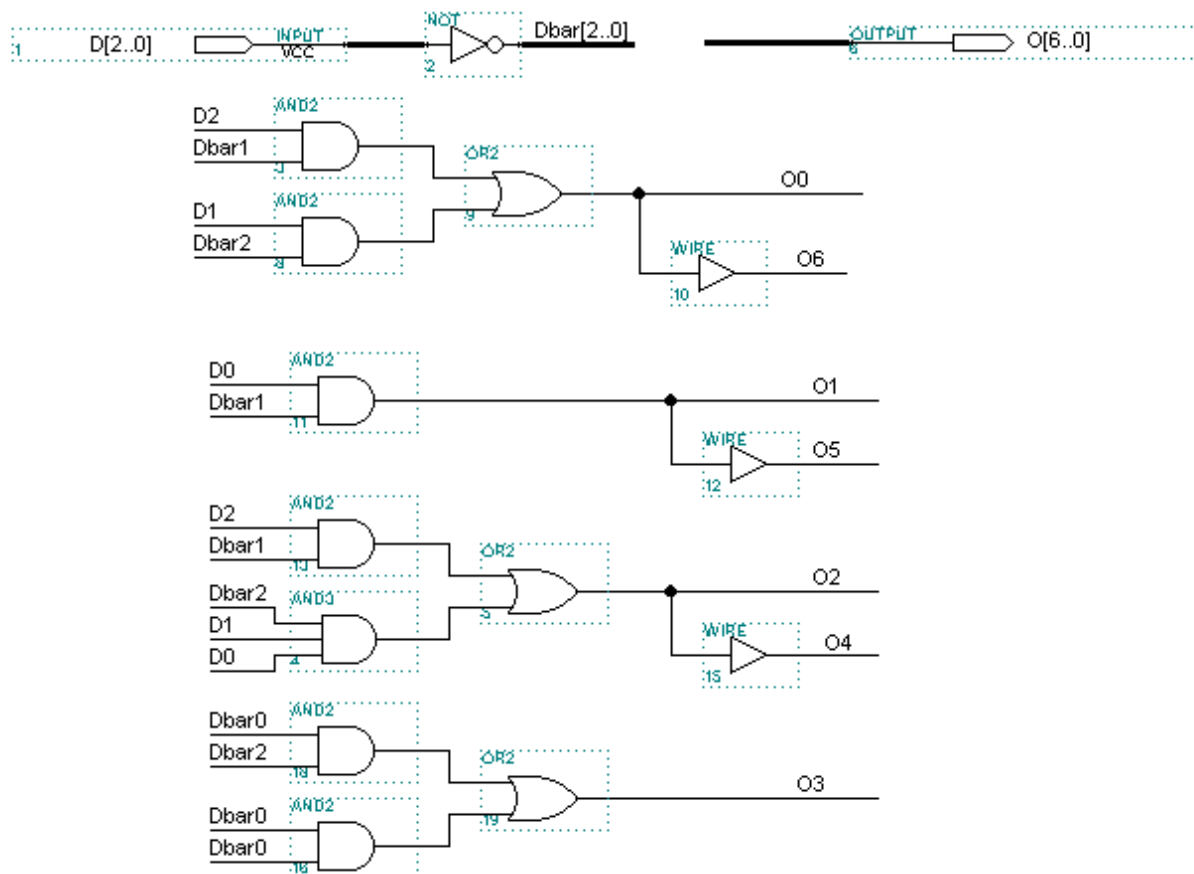


Figure 7.7a Using MAX+PLUS II Graphic editor to create decide decoder circuits
(File: dice_dec.gdf)

		20.0ns	40.0ns	60.0ns	80.0ns	100.0ns
D[2..0]	B 100	000	001	010	011	100
O[6..0]	B 1011101	0001000	0100010	0011001	1010101	1011101
Dbar[2..0]	B 011	111	110	101	100	011

Figure 7.7b Simulation result of dice decoder (File: dice_dec.scf)

Step 6: After floorplan programming, please download the circuits and perform the testing of the circuit. Please modify Figure 7.7a, dice decoder circuit, as shown in Figure 5.3, to Figure 7.7c. Dice_COM is ready for connecting the VCC output to the dice anode. Please re-compile it after modifying and adapt the floorplan programming techniques in Section 4.6. Please select EPF10K10TC144-4 chip and use the pin assignment reference shown in Table 7.6.

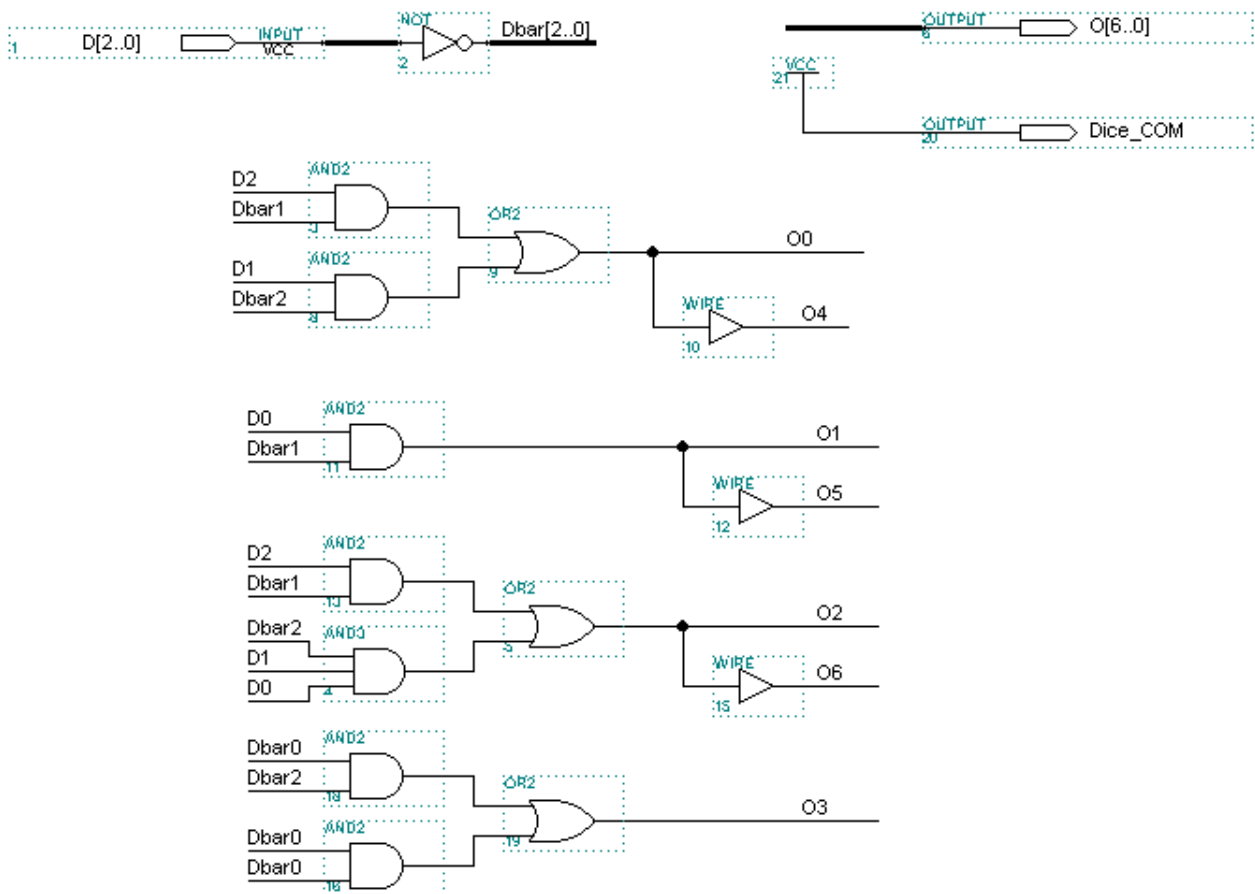


Figure 7.7c Using MAX+PLUS II Graphic editor to create dice decoder circuits

Table 7.6 Pin assignment of EPF10K10TC144-4 chip

Signal	EPF10K10TC144-4 chip pin	Signal	EPF10K10TC144-4 chip pin
D0	Pin 47	O3	Pin 10
D1	Pin 48	O4	Pin 11
D2	Pin 49	O5	Pin 12
O0	Pin 7	O6	Pin 13
O1	Pin 8		
O2	Pin 9	Dice_COM	Pin 142

After setting up LP-2900 Lab Platform, go on to download the dice decoder to EPF10K10TC144-4 chip. Try to push down SW1 (D0), SW2 (D1), and SW3 (D2), on the left-bottom of LP-2900. Please note the changes of L13 (O0), L14 (O10), L15 (O9) ... and L19 (O0) .

Step 7: Please refer to File > Create Default Symbol to generate the internal circuit symbol of “Dice Decoder” for the upper layer circuit (Figure 7.7d).

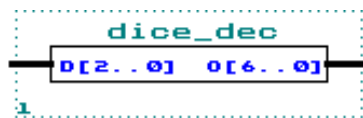


Figure 7.7d The internal circuit symbol of “Dice Decoder”

7.2.2 MOD6 Counter

There a counter, which the initial value is 0000 (0), has the value turns to 0001 (1) when the positive edge of the clock enters. If the clock keeps flowing in, the counter will sequentially turns to 0010 (2) → 0011 (3) → 0100 (4) → 0101 (5) → 0000 (0) → ...and keeps cycling like this.



From the above circuit specifications, we know it is a MOD6 Counter. The design is as follows:

Step 1: Complete the state setup of the circuit specification and illustrate by a State table. The circuit specification of MOD6 counter can be illustrated as the State table 7.7a.

Table 7.7a MOD 6 Counter sequential circuits

Present State	Next State	Outputs
000	001	000
001	010	001
010	011	010
011	100	011
100	101	100
101	000	101

Step 2: Figure out the input and output functions of Karnaugh Matrix or other minimized functions of each Flip-Flop by adapting an Excitation table. From the Flip-Flop Excitation table, Table 6.14c, we can figure out the Karnaugh Matrix of Flip-Flop input function, as shown from Table 7.7b to Table 7.7g.

Table 7.7b Karnaugh Map of J_0 , $J_0 = Y_2'Y_0' + Y_1'Y_0'$

J_0		Present State Input Y_1Y_0			
		00	01	11	10
Present State	0	1	—	—	1
Input Y_2	1	1	—		

Table 7.7c Karnaugh Map of K_0 , $K_0 = Y_1'Y_0 + Y_2'Y_0$

K_0		Present State Input Y_1Y_0			
		00	01	11	10
Present State	0	–	1	1	–
Input Y_2	1	–	1		

Table 7.7d Karnaugh Map of J_1 , $J_1 = Y_2'Y_0$

J_1		Present State Input Y_1Y_0			
		00	01	11	10
Present State	0	0	1	–	–
Input Y_2	1	0	0		

Table 7.7e Karnaugh Map of K_1 , $K_1 = Y_2'Y_0$

K_1		Present State Input Y_1Y_0			
		00	01	11	10
Present State	0	–	–	1	0
Input Y_2	1	–	–		

Table 7.7f Karnaugh Map of J_2 , $J_2 = Y_2'Y_1Y_0$

J_2		Present State Input Y_1Y_0			
		00	01	11	10
Present State	0	0	0	1	0
Input Y_2	1	–	–		

Table 7.7g Karnaugh Map of K_2 , $K_2 = Y_1'Y_0$

K_2		Present State Input Y_1Y_0			
		00	01	11	10
Present State	0	—	—	—	—
Input Y_2	1	0	1		

Step 3: Figure out each minimized input and output functions.

$$\begin{aligned}
 J_0 &= Y_2'Y_0' + Y_1'Y_0' & ; & & K_0 &= Y_1'Y_0 + Y_2'Y_0 \\
 J_1 &= Y_2'Y_0 & ; & & K_1 &= Y_2'Y_0 \\
 J_2 &= Y_2'Y_1Y_0 & ; & & K_2 &= Y_1'Y_0 \\
 Q_0 &= Y_0 & ; & & Q_1 &= Y_1 \\
 Q_2 &= Y_2
 \end{aligned}$$

Step 4: Using the Graphic editor of MAX+PLUS II to create a circuit entry illustrated in Figure 7.8.

Step 5: Complete functional simulation by using MAX+PLUS II (Figure 7.9) and test whether the functions meet the circuit specifications. Go on to the next step if the functions meet the specifications; otherwise, reinitialize step 1 to check the cause of error sequentially.

Step 6: If the circuits allow test by downloading, download (or programming) to test the circuit after selecting the download (or programming) device and floorplan program. Please modify Figure 7.8, MOD6 counter circuit, as the verification of the circuit shown in Figure 5.3. Re-compile it when complete the verification. Then, adapting the floorplan programming techniques instructed in Section 4.6, to select an EPF10K10TC144-4 chip

and use the pin assignment shown in Table 7.8. After assemble LP-2900 Lab Platform, download MOD6 counter to EPF10K10TC144-4 chip. Try to push PS1 down on the left-bottom of LP-2900. Please note the changes of L1 (O2), L2 (O1), and L3 (O0).

Step 7: Please refer to File > Create Default Symbol to generate the internal circuit symbol of MOD 6 (Figure 7.10) for the good of the upper layer circuit.

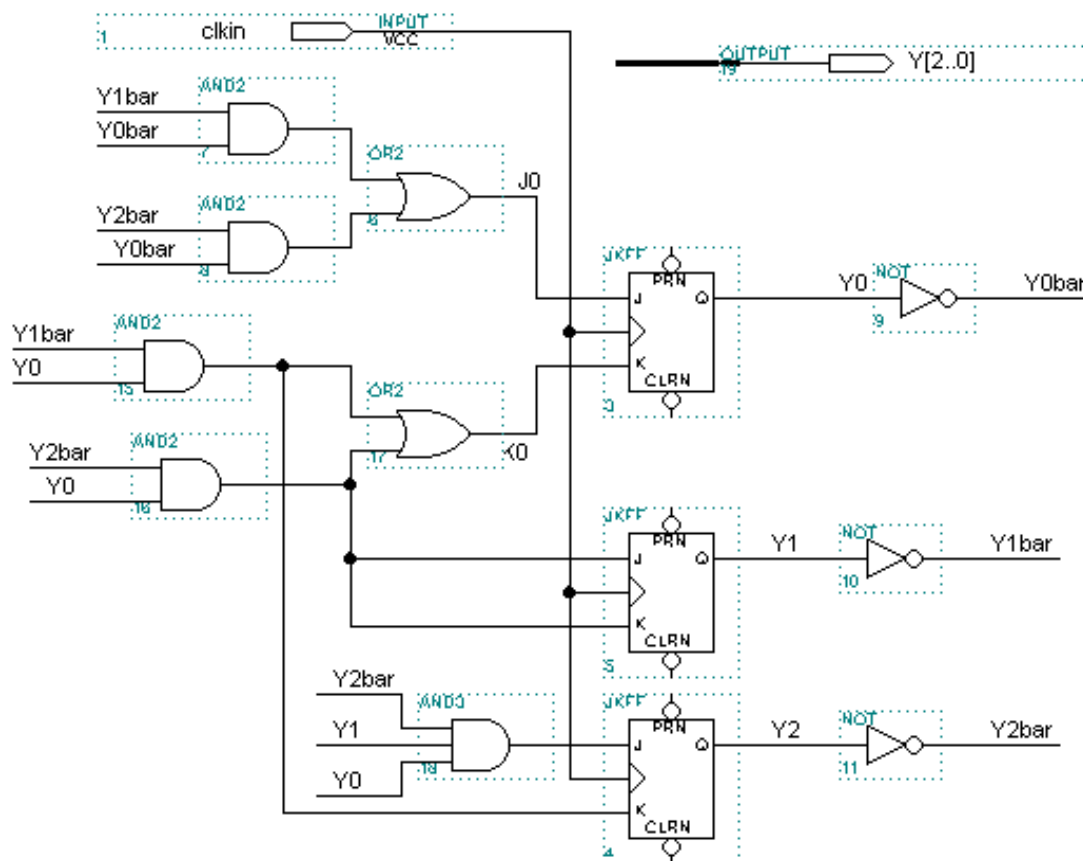


Figure 7.8 Using MAX+PLUS II Graphic editor to create MOD 6 counter circuit
(File : mod6.gdf)

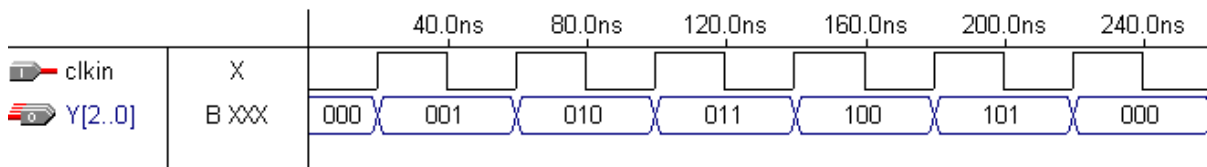


Figure 7.9 Simulation result of Mod 6 counter circuit (File : mod6.scf)

Table 7.8 Pin assignment of EPF10K10TC144-4 chip

Signal	EPF10K10TC144-4 chip pin	Signal	EPF10K10TC144-4 chip pin
CLKIN	Pin 54		
Q0	Pin 9	LED_COM	Pin 141
Q1	Pin 8		
Q2	Pin 7		

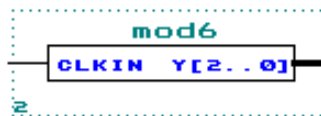


Figure 7.10 Internal circuit symbol of MOD 6

7.2.3 Dice Game Circuit

In this section, we will use “dice decoder circuit”, “MOD6 counter” and the frequency circuit listed in Section 7.1 to compose a simple dice game circuit.

Step 1: Using MAX+PLUS II Graphic editor to create a circuit entry illustrated in Figure 7.11a and modify circuit, Figure 7.11b, for simulation purpose.

Step 2: Complete functional simulation (Figure 7.12) by using MAX+PLUS II, and test whether the functions meet the circuit specifications. For the convenience of the mimic of the circuit shown in Figure 7.11a, we slightly modify the circuit as shown in Figure 7.11b. Go on to the next step if the

functions meet the specifications; otherwise, reinitialize step 1 to check the cause of error sequentially.

Step 3: If the circuit allows download testing, download (or programming) to test the circuit after selecting the download (or programming) device and floorplan.

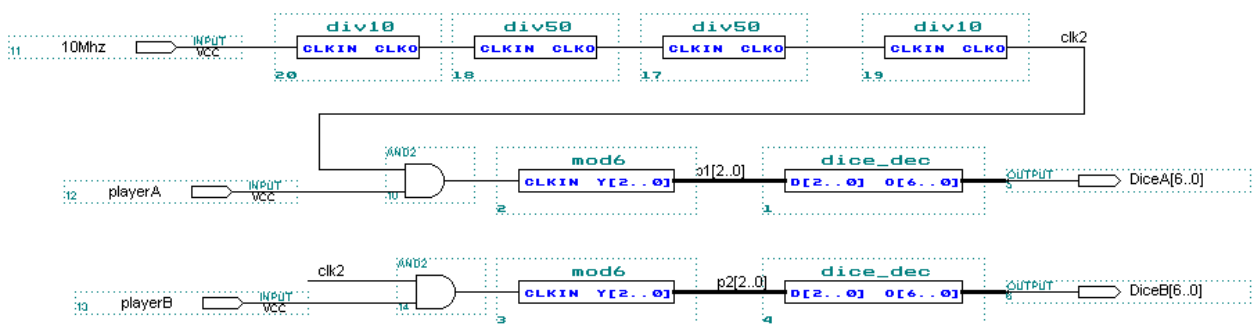


Figure 7.11a Use MAX+PLUS II Graphic editor to create dice game circuit
(File: dice_game.gdf)

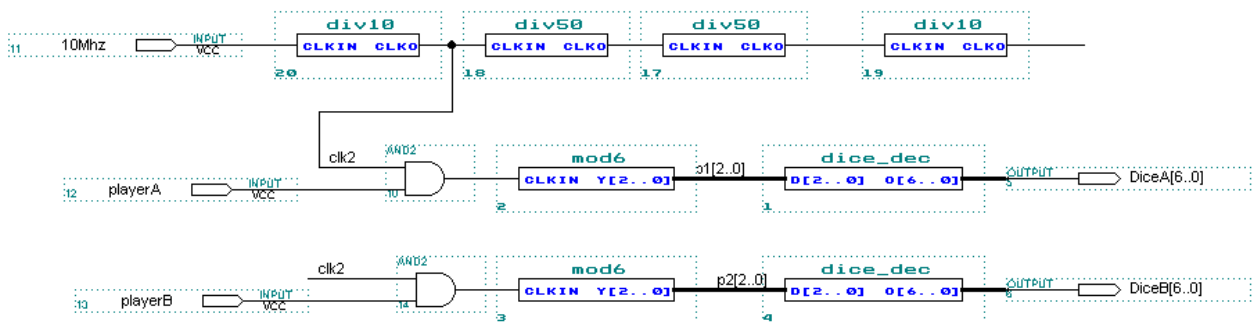


Figure 7.11b Dice game circuit modified for simulation purpose
(File: dice_gamb.gdf)

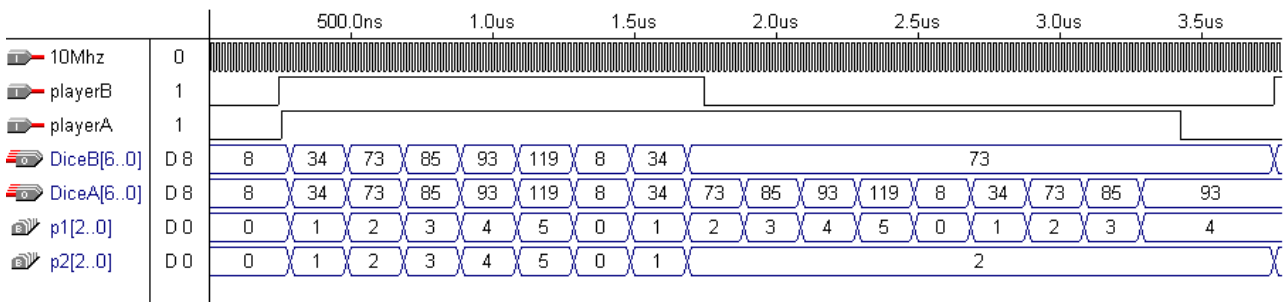


Figure 7.12 Simulation result of “Dice Game Circuit” (File: dice_gamb.scf)

Please restore Figure 7.11a, modify this Dice Game Circuit according to Figure 5.4a shown in Section 5.2.1. Please re-compile it when complete the modification. Then, adapting the floorplan programming techniques instructed in Section 4.6 to choose EPF10K10TC144-4 chip, and use pin assignment in Table 7.9.

After setting up LP-2900 Lab Platform, download Dice Game Circuit to EPF10K10TC144-4 chip. Try to push PS3 (Pin 124) and PS4 (Pin 126) on the left bottom of LP-2900. Please note the changes of the “Dice”.

Table 7.9 Pin assignment of EPF10K10TC144-4 chip

Signal	EPF10K10TC144-4 pin setup	Signal	EPF10K10TC144-4 pin setup
DiceA0	Pin7	DiceB0	Pin14
DiceA1	Pin8	DiceB1	Pin17
DiceA2	Pin9	DiceB2	Pin18
DiceA3	Pin10	DiceB3	Pin19
DiceA4	Pin11	DiceB4	Pin20
DiceA5	Pin12	DiceB5	Pin21
DiceA6	Pin13	DiceB6	Pin22
PlayerA	Pin 124	PlayerB	Pin 126
CLKIN	Pin 55	DICE_COM	Pin 142



7.3 Timer

In this section, we will design a timer that has a start/stop button to control the start and stop of the timer, and a clrn button to clear the figure. The timer is set as follows:

X. XX. XX. X.
Hour Minute Second 10 Hertz

To implement the timer circuit, the circuit of 10 Hz pause generator, decimal counter, and 60-carry counter, 12-carry counter and a scan display are required. Since the circuit of 10Hz pause generator circuit has shown in Section 7.1.5, we will not mention it again in this section. The following sections are the descriptions of each sub-circuit that will be integrated into a timer circuit.

7.3.1 Decimal Counter

Step 1: Form a Truth table (Table 7.10a) ◦

Table 7.10a State table of decimal counter sequential circuit

Present State	Next State	Outputs Clko	Outputs x[3..0]
0000	0001	0	0000
0001	0010	0	0001
0010	0011	0	0010
0011	0100	0	0011
0100	0101	0	0100
0101	0110	1	0101
0110	0111	1	0110

0111	1000	1	0111
1000	1001	1	1000
1001	0000	1	1001

Note: Decimal counter is similar with 10-divider

Step 2: From the above Truth table (Table 7.10a) leads to the following Karnaugh Map and equations. (Table 7.10b to Table 7.10j)

Table 7.10b Karnaugh Map of T_0 , $T_0 = Y_3' + Y_2'Y_1'$

T_0		Present State Input Y_1Y_0			
		00	01	11	10
Present State Input Y_3Y_2	00	1	1	1	1
	01	1	1	1	1
	11				
	10	1	1		

Table 7.10c Karnaugh Map of T_1 , $T_1 = Y_3'Y_0$

T_1		Present State Input Y_1Y_0			
		00	01	11	10
Present State Input Y_3Y_2	00	0	1	1	0
	01	0	1	1	0
	11				
	10	0	0		

Table 7.10d Karnaugh Map of T_2 , $T_2 = Y_3'Y_1Y_0$

T_2		Present State Input Y_1Y_0			
		00	01	11	10
Present State Input Y_3Y_2	00	0	0	1	0
	01	0	0	1	0
	11				
	10	0	0		

Table 7.10e Karnaugh Map of T_3 , $T_3 = Y_3'Y_2Y_1Y_0 + Y_3Y_2'Y_1'Y_0$

T_3		Present State Input Y_1Y_0			
		00	01	11	10
Present State Input Y_3Y_2	00	0	0	0	0
	01	0	0	1	0
	11				
	10	0	1		

Table 7.10f Karnaugh Map of clk_0 , $clk_0 = Y_3'Y_2Y_1'Y_0' + Y_3Y_2'Y_1'Y_0$

clk_0		Present State Input Y_1Y_0			
		00	01	11	10
Present State Input Y_3Y_2	00	0	0	0	0
	01	1	0	0	0
	11				
	10	0	1		

Table 7.10g Karnaugh Map of X_0 , $X_0 = Y_0$

X_0		Present State Input Y_1Y_0			
		00	01	11	10
Present State Input Y_3Y_2	00	0	1	1	0
	01	0	1	1	0
	11				
	10	0	1		

Table 7.10h Karnaugh Map of X_1 , $X_1 = Y_1$

X_1		Present State Input Y_1Y_0			
		00	01	11	10
Present State Input Y_3Y_2	00	0	0	1	1
	01	0	0	1	1
	11				
	10	0	0		

Table 7.10i Karnaugh Map of X_2 , $X_2 = Y_2$

X_2		Present State Input Y_1Y_0			
		00	01	11	10
Present State Input Y_3Y_2	00	0	0	0	0
	01	1	1	1	1
	11				
	10	0	0		

Table 7.10j Karnaugh Map of X_3 , $X_3 = Y_3$

X_3		Present State Input $Y_1 Y_0$			
		00	01	11	10
Present State Input $Y_3 Y_2$	00	0	0	0	0
	01	0	0	0	0
	11				
	10	1	1		

Step 3: Figure out each minimized functions of each Inputs and Outputs.

$$\begin{aligned}
 T_0 &= Y_3' + Y_2'Y_1' & ; & & T_1 &= Y_3'Y_0 \\
 T_2 &= Y_3'Y_1Y_0 & ; & & T_3 &= Y_3'Y_2Y_1Y_0 + Y_3Y_2'Y_1'Y_0 \\
 X_0 &= Y_0 & ; & & X_1 &= Y_1 \\
 X_2 &= Y_2 & ; & & X_3 &= Y_3 \\
 clko &= Y_3'Y_2Y_1'Y_0' + Y_3Y_2'Y_1'Y_0
 \end{aligned}$$

Step 4: Please use the Graphic editor of MAX+PLUS II to complete editing the circuit entry of the above equations (Figure 7.13a) .

Step 5: Complete functional simulation by using MAX+PLUS II (Figure 7.13b) and test weather the functions meet the circuit specifications. Go on to the next step if the functions meet the specifications; otherwise go back to step 1 to check the cause of error sequentially.

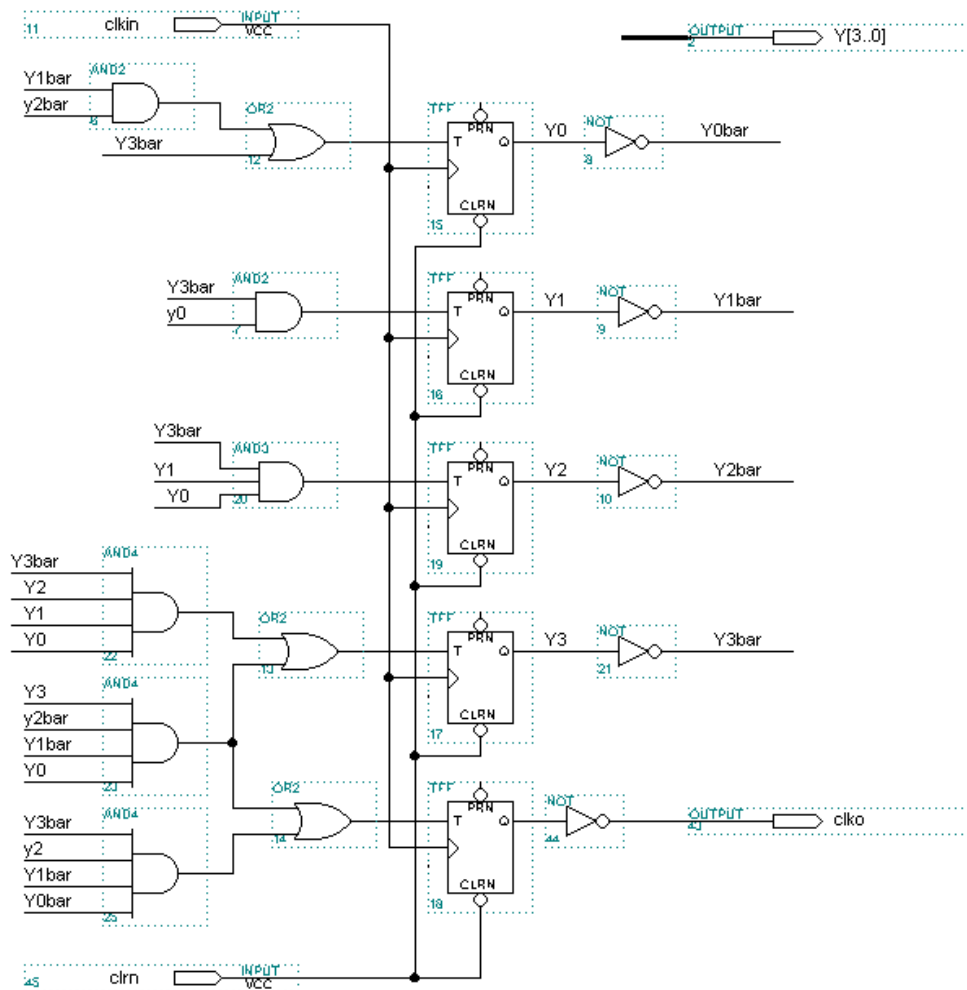


Figure 7.13a Use Graphic editor of MAX+PLUS II to create decimal counter
(File: bcd10.gdf)

Step 6: If the circuits allow download testing, download (or programming) to test the circuit after selecting the download (or programming) device and floorplan program.

~Skip the download testing~

Step 7: Please use File > Create Default Symbol to generate a decimal counter circuit symbol (Figure 7.13c) for the use of upper layer circuits.

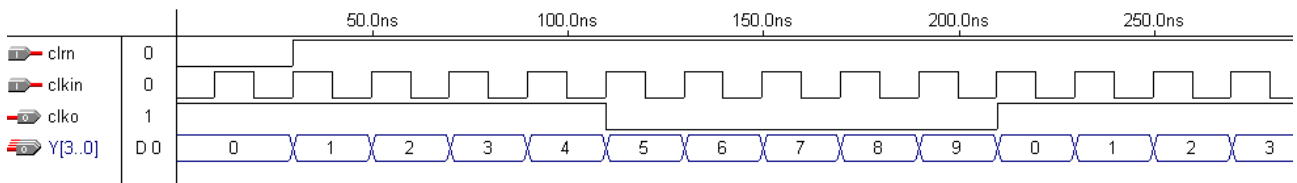


Figure 7.13b Simulation result of decimal counter circuit
(File: bcd10.scf)

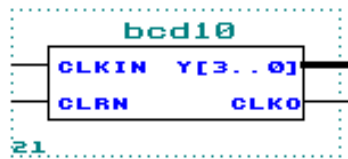


Figure 7.13c Internal circuit symbol of decimal counter

7.3.2 60-carry Counter Circuit

Two elements 6-carry counter and decimal counter compose a 60-carry counter. The design of decimal counter is illustrated in 7.3.1. 6-carry counter, adapting the clock output of decimal counter, is designed as follows:

Step 1: Form a 6-carry counter Truth table (Table 7.11a).

Table 7.11a State table of a 6-carry counter's sequential circuit

Present State	Next State	Outputs clko	Outputs x[2..0]
000	001	0	000
001	010	0	001
010	011	0	010
011	100	1	011
100	101	1	100
101	000	1	101

Step 2: From the above Truth table (Table 7.11a) leads to the following Karnaugh Map and equations (Table 7.11b~Table 7.11e).

Table 7.11b Karnaugh Map of T_0 , $T_0 = V_{cc}$

T_0		Present State Input x_1x_0			
		00	01	11	10
Present State	0	1	1	1	1
Input x_2	1	1	1		

Table 7.11c Karnaugh Map of T_1 , $T_1 = x_2'x_0$

T_1		Present State Input x_1x_0			
		00	01	11	10
Present State	0	0	1	1	0
Input x_2	1	0	0		

Table 7.11d Karnaugh Map of T_2 , $T_2 = x_2'x_1x_0 + x_2x_1'x_0$

T_2		Present State Input x_1x_0			
		00	01	11	10
Present State	0	0	0	1	0
Input x_2	1	0	1		

Table 7.11e Karnaugh Map of Clko, $clko = x_2'x_1'x_0 + x_2x_1'x_0$

clko		Present State Input x_1x_0			
		00	01	11	10
Present State	0	0	0	0	1
Input x_2	1	0	1		

Step 3: Figure out the minimized equations of each input and output function. Decimal parts are shown in 7.3.1. The following is are parts of 6-carry counter:

$$\begin{aligned}
 T_0 &= V_{cc} & ; & & T_1 &= x_2'x_0 \\
 T_2 &= x_2'x_1x_0 + x_2x_1'x_0 & ; & & clko &= x_2'x_1'x_0 + x_2x_1'x_0
 \end{aligned}$$

Step 4: Please use the Graphic editor of MAX+PLUS II to complete editing the circuit entry of the above equations (Figure 7.14a) .

Step 5: Complete functional simulation by using MAX+PLUS II (Figure 7.14b and Figure 7.14c) and test weather the functions meet the circuit specifications. Go on to the next step if the functions meet the specifications; otherwise go back to step 1 to check the cause of error sequentially

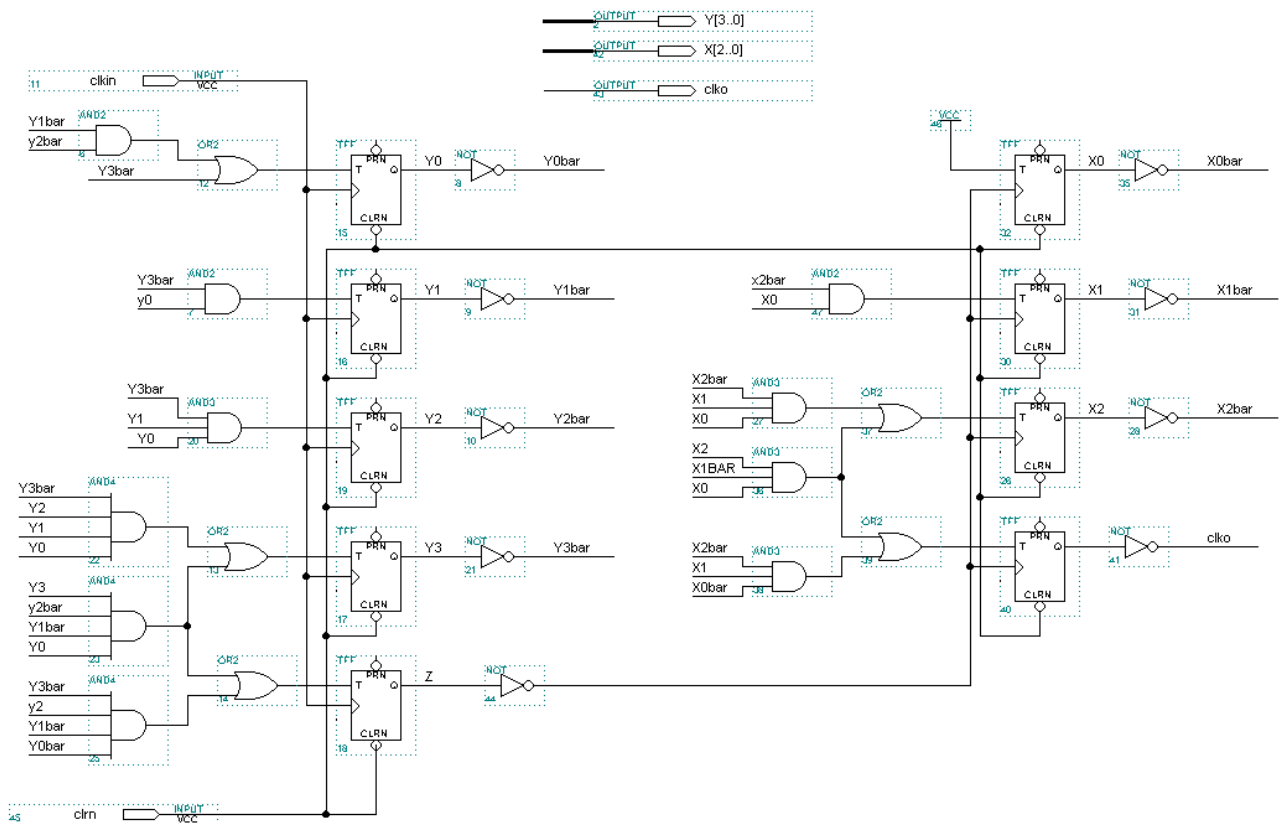


Figure 7.14a Use AX+PLUS II Graphic editor to create a 60-carry counter circuit
(File: bcd60.gdf)

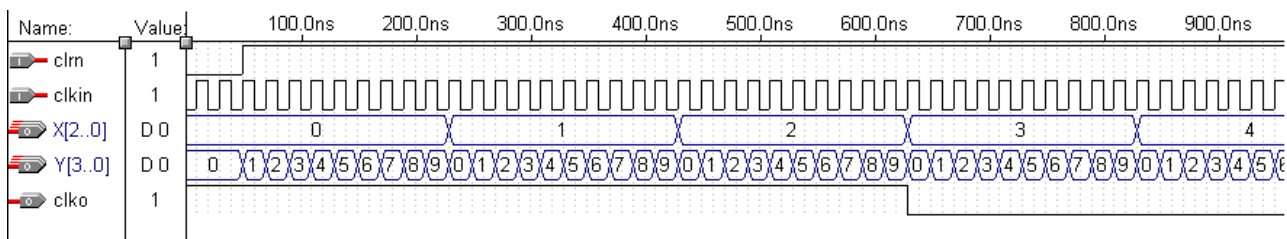


Figure 7.14b Simulation results of a 60-carry counter circuit (File: bcd60.scf)

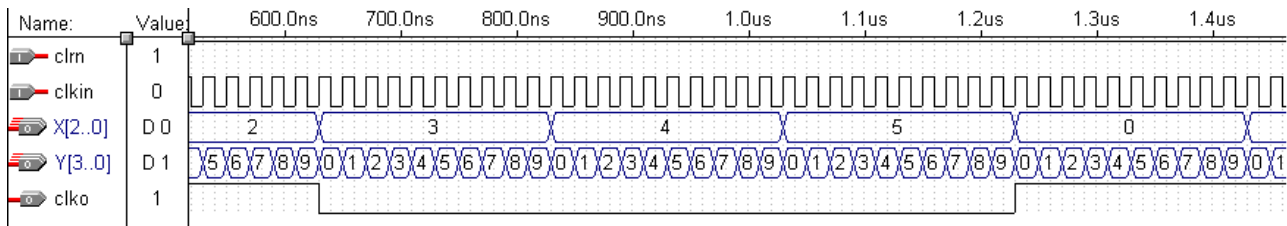


Figure 7.14c Simulation result of a 60-carry counter circuit (File: bcd60.scf)

Step 6: If the circuits allow download testing, download (or programming) to test the circuit after selecting the download (or programming) device and floorplan program.

~Skip this download testing~

Step 7: Please use File > Create Default Symbol to generate an internal circuit symbol of 60-carry counter Circuit (Figure 7.14d) for the upper layer circuit.

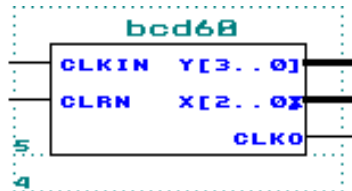


Figure 7.14d The internal circuit symbol of 60-carry counter

7.3.3 12-carry Counter Circuit

Like 60-carry counter, 12-carry counter is designed as the followings:

Step 1: Form a Truth table (Table 7.12a) ◦

Step 2: From the above Truth table (Table 7.12a) leads to the following Karnaugh Map and equations (Table 7.12b to Table 7.12g) ◦

Table 7.12a State table of 12-carry counter's sequential circuit

Present State $X_0Y_3Y_2Y_1Y_0$	Next State $x_0Y_3Y_2Y_1Y_0$	Output clk0	Output $x_0Y_3Y_2Y_1Y_0$
00000	00001	0	00000
00001	00010	0	00001
00010	00011	0	00010
00011	00100	0	00011
00100	00101	0	00100
00101	00110	0	00101
00110	00111	1	00110
00111	01000	1	00111
01000	01001	1	01000
01001	10000	1	01001
10001	10010	1	10001
10010	00000	1	10010

Note: x_0 is decimal

Table 7.12b Karnaugh Map of T_0 , $T_0 = x_0'Y_3' + Y_3'Y_2'Y_1' + x_0'Y_3Y_2'Y_1'$

T_0			Present State Input Y_1Y_0			
			00	01	11	10
Present State Input Y_3Y_2	00	$X_0 = 0$	1	1	1	1
		$X_0 = 1$	1	1		0
	01	$X_0 = 0$	1	1	1	1
		$X_0 = 1$				
	11	$X_0 = 0$				
		$X_0 = 1$				
	10	$X_0 = 0$	1	1		
		$X_0 = 1$				

Table 7.12c Karnaugh Map of T_1 , $T_1 = Y_3'Y_0 + x_0Y_3'Y_2'Y_1$

T_1			Present State Input Y_1Y_0			
			00	01	11	10
Present State Input Y_3Y_2	00	$X_0 = 0$	0	1	1	0
		$X_0 = 1$	0	1	/	1
	01	$X_0 = 0$	0	1	1	0
		$X_0 = 1$	/	/	/	/
	11	$X_0 = 0$	/	/	/	/
		$X_0 = 1$	/	/	/	/
	10	$X_0 = 0$	0	1	/	/
		$X_0 = 1$	/	/	/	/

Table 7.12d Karnaugh Map of T_2 , $T_2 = Y_1Y_0$

T_2			Present State Input Y_1Y_0			
			00	01	11	10
Present State Input Y_3Y_2	00	$X_0 = 0$	0	0	1	0
		$X_0 = 1$	0	0	/	0
	01	$X_0 = 0$	0	0	1	0
		$X_0 = 1$	/	/	/	/
	11	$X_0 = 0$	/	/	/	/
		$X_0 = 1$	/	/	/	/
	10	$X_0 = 0$	0	0	/	/
		$X_0 = 1$	/	/	/	/

Table 7.12e Karnaugh Map of T_3 , $T_3 = x_0'Y_3'Y_2Y_1Y_0 + x_0'Y_3Y_2'Y_1'Y_0$

T_3			Present State Input Y_1Y_0			
			00	01	11	10
Present State Input Y_3Y_2	00	$X_0 = 0$	0	0	0	0
		$X_0 = 1$	0	0	/	0
	01	$X_0 = 0$	0	0	1	0
		$X_0 = 1$	/	/	/	/
	11	$X_0 = 0$	/	/	/	/
		$X_0 = 1$	/	/	/	/
	10	$X_0 = 0$	0	1	/	/
		$X_0 = 1$	/	/	/	/

Table 7.12f Karnaugh Map of T_4 , $T_4 = x_0'Y_3Y_2'Y_1'Y_0 + x_0Y_3'Y_2'Y_1Y_0'$

T_4			Present State Input Y_1Y_0			
			00	01	11	10
Present State Input Y_3Y_2	00	$X_0 = 0$	0	0	0	0
		$X_0 = 1$	0	0	/	1
	01	$X_0 = 0$	0	0	0	0
		$X_0 = 1$	/	/	/	/
	11	$X_0 = 0$	/	/	/	/
		$X_0 = 1$	/	/	/	/
	10	$X_0 = 0$	0	1	/	/
		$X_0 = 1$	/	/	/	/

Table 7.12g Karnaugh Map of clko, $clko = x_0'Y_3'Y_2Y_1Y_0' + x_0Y_3'Y_2'Y_1Y_0'$

T_5			Present State Input Y_1Y_0			
			00	01	11	10
Present State Input Y_3Y_2	00	$X_0 = 0$	0	0	0	0
		$X_0 = 1$	0	0		1
	01	$X_0 = 0$	0	0	0	1
		$X_0 = 1$				
	11	$X_0 = 0$				
		$X_0 = 1$				
	10	$X_0 = 0$	0	1		
		$X_0 = 1$				

Step 3: Figure out the minimized equations of each input and output functions.

$$T_0 = x_0'Y_3' + Y_3'Y_2'Y_1' + x_0'Y_3Y_2'Y_1'$$

$$T_1 = Y_3'Y_0 + x_0Y_3'Y_2'Y_1$$

$$T_2 = Y_1Y_0$$

$$T_3 = x_0'Y_3'Y_2Y_1Y_0 + x_0'Y_3Y_2'Y_1'Y_0$$

$$T_4 = x_0'Y_3Y_2'Y_1'Y_0 + x_0Y_3'Y_2'Y_1Y_0'$$

$$Clko = x_0'Y_3'Y_2Y_1Y_0' + x_0Y_3'Y_2'Y_1Y_0'$$

Step 4: Please use the Graphic editor of MAX+PLUS II to complete editing the circuit entry of the above equations (Figure 7.15a) .

Step 5: Complete functional simulation by using MAX+PLUS II (Figure 7.15b) and test weather the functions meet the circuit specifications. Go on to the next step if meets the specifications; otherwise go back to step 1 to check the cause of error sequentially.

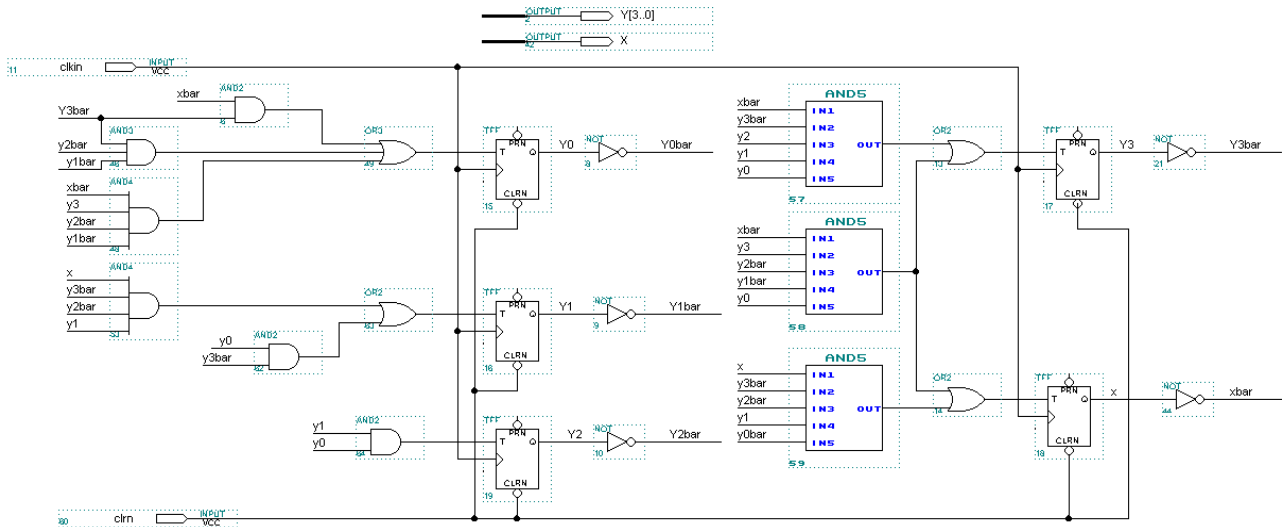


Figure 7.15a Use Graphic editor of MAX+PLUS II to create 12-carry counter circuit (File: bcd12.gdf)

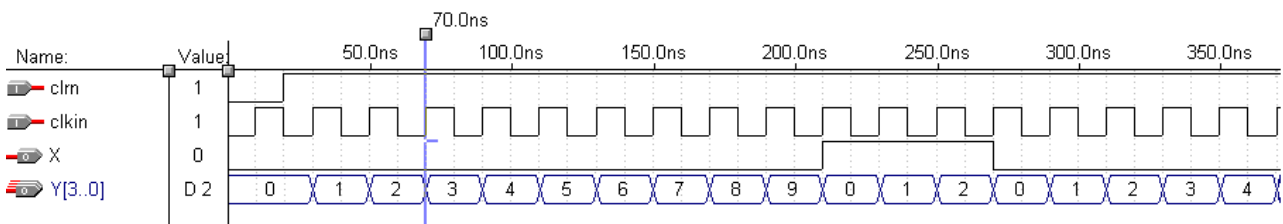


Figure 7.15b Simulation result of 12-carry counter (File: bcd12.scf)

Step 6: If the circuits allow download testing, download (or programming) to test the circuit after selecting the download (or programming) device and floorplan program.

~Skip this download testing~

Step 7: Please use File > Create Default Symbol to generate an internal circuit symbol of 12-carry Counter (Figure 7.15c) for the upper layer circuit.

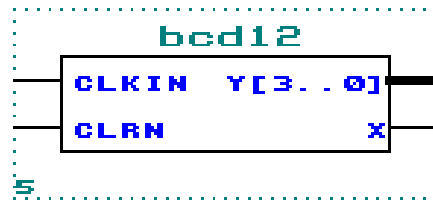


Figure 7.15c The internal circuit symbol of 12-carry counter

7.3.4 Scan Display Circuit

The Scan Display Circuit is in charge of selecting data, and through 7-segment display decoding, the figures will be displayed on the proper 7-segment display. Take Figure 7.16a as an example, in segment T0 ($c_0 = "0"$, others = "1") D0 is selected to Y. After decoding, the result will be transferred from Z to (a, b, c, d, e, f, g, h). In segment T1 ($c_1 = "0"$, others = "1") D1 is selected to Y. After decoding, the result will be transferred from Z to (a, b, c, d, e, f, g, h). In segment T2 ($c_2 = "0"$, others = "1") D2 is selected to Y. After decoding, the result will be transferred from Z to (a, b, c, d, e, f, g, h). In segment T3 ($c_3 = "0"$, others = "1") D3 is selected to Y. After decoding, the result will be transferred from Z to (a, b, c, d, e, f, g, h). In segment T4 ($c_4 = "0"$, others = "1") D4 is selected to Y. After decoding, the result will be transferred from Z to (a, b, c, d, e, f, g, h). In segment T5 ($c_5 = "0"$, others = "1") D5 is selected to Y. After decoding, the result will be transferred from Z to (a, b, c, d, e, f, g, h). In segment T6 ($c_0 = "0"$, others = "1") again, D0 is selected to Y. After decoding, the result will be transferred from Z to (a, b, c, d, e, f, g, h). The transmission will keep on cycling like this. If the speed of scanning is faster than that of visual pause, we can see the steady, non-flashing display of the digits.

Theoretically, we need three types of circuits, scan signal generator, data selecting,

and 7-segment display decoding circuit. There are only 144 pins on EPF10K-10TC144-4 chip on LP-2900 Logic Circuit Design Lab Platform; therefore, except for EPF10K10TC144-4 chip, a 74138 LSI chip, see Figure 7.16b, which provides the scan signal of c1 to c6 (code number on LP-2900) is required to save pins. However, EPF10K10TC144-4 chip has provided a, b, and c signals for activating 74138 to generate scan signals of c1 to c6. So, what kind functions of 74138 are? We can distinguish its functions from the following Truth table (Table 7.13). Under the condition of $G1 = "1"$ and $G2A = G2B = "0"$, make ABC sequentially turns "000" to "111" and the scan signals would be consequentially generated. Therefore, we modify the scan display circuit of Figure 7.16a into a circuit like Figure 7.16c. In other words, scan display circuit turns into a composition circuits (in the frame of dotted lines) from MOD 8 circuit (to generate ABC signals for 74138), data selecting circuit, and 7-segment display decoding circuit.

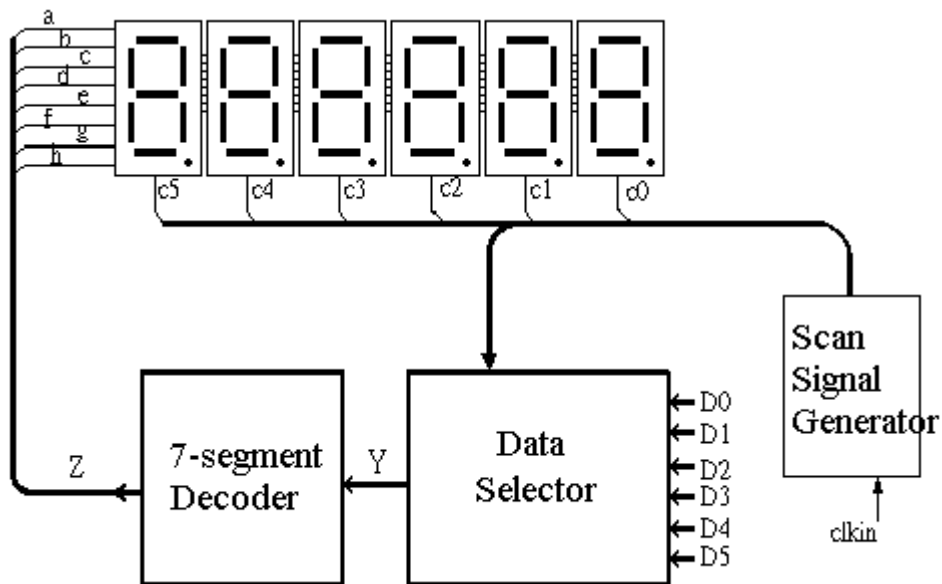


Figure 7.16a Scan display

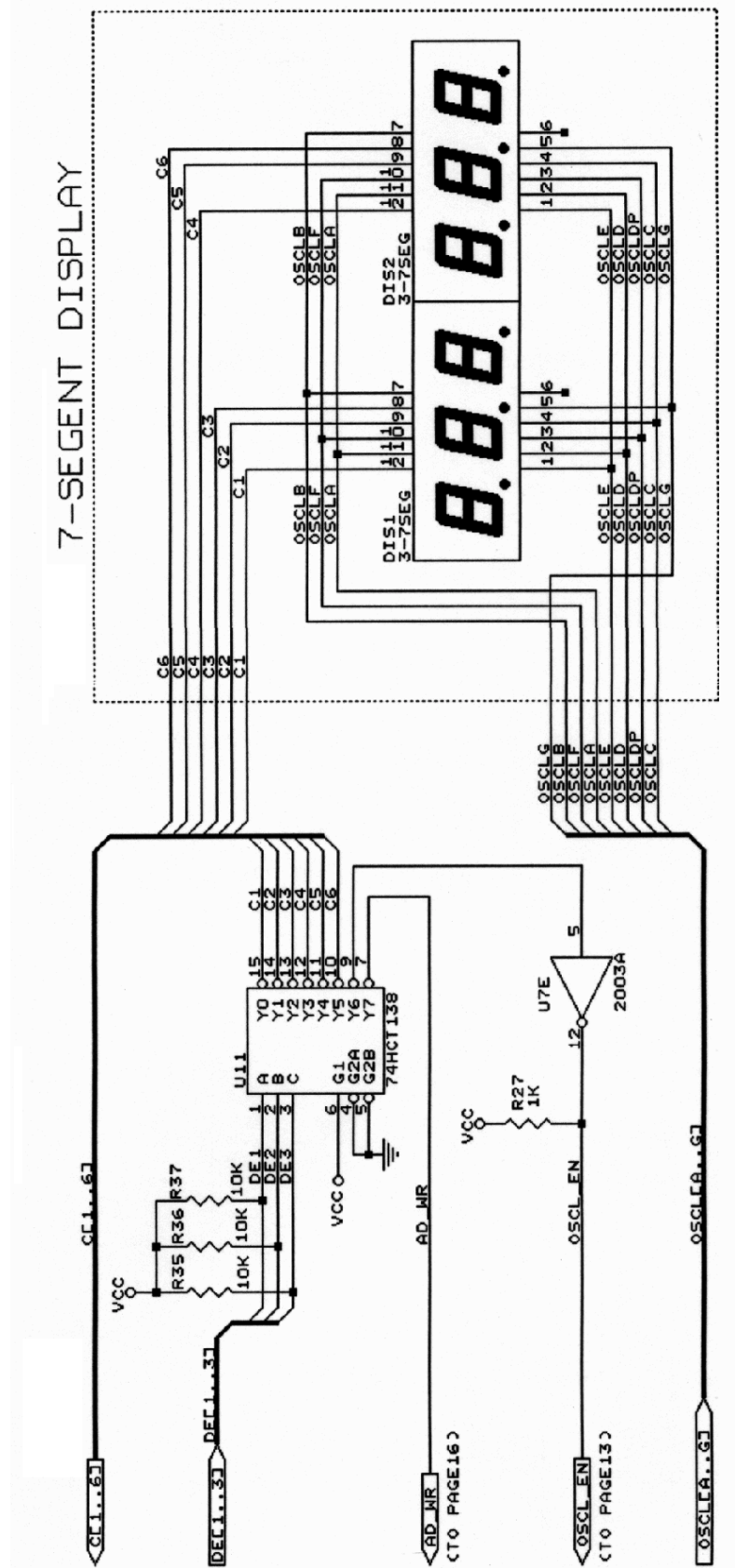


Figure 7.16b Scan Display in LP-2900

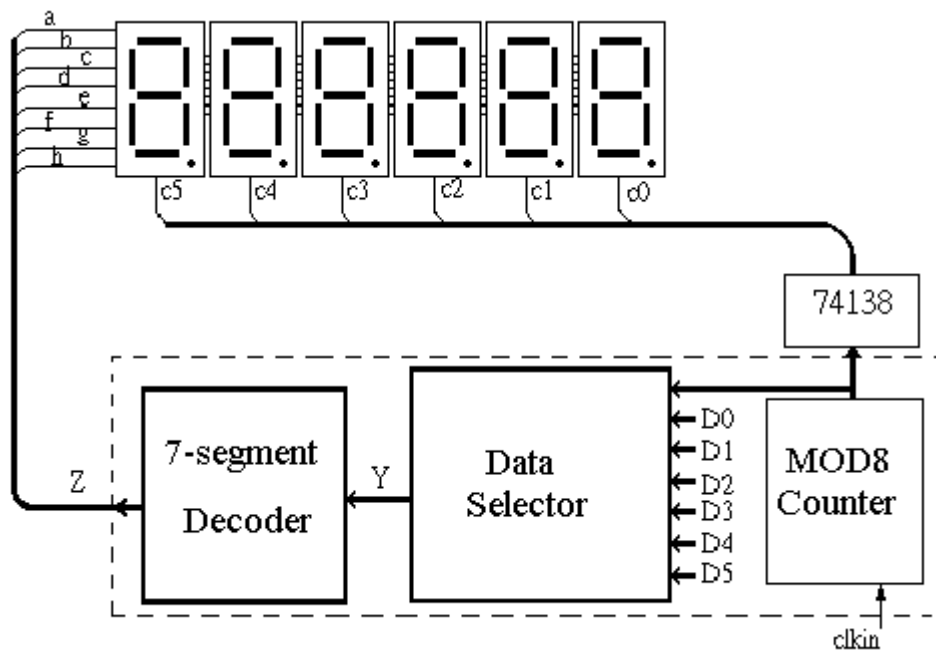


Figure 7.16c Modified Scan Display

Table 7.13 Truth table of 74138

Inputs				Outputs
G1	G2A	G2B	CBA	$Y_7Y_6Y_5Y_4Y_3Y_2Y_1Y_0$
0	—	—	—	11111111
—	1	1	—	11111111
1	0	0	000	11111110
1	0	0	001	11111101
1	0	0	010	11111011
1	0	0	011	11110111
1	0	0	100	11101111
1	0	0	101	11011111
1	0	0	110	10111111
1	0	0	111	01111111

Step 1: The 7-segment display decoding circuit has been introduced in Section 5.6.2 which we will adopt the circuit directly in this section. For the circuit design of MOD 8 counter, please refer to Figure 7.17a in Section 7.2.2. Figure 7.17b illustrates the simulation result of Figure 7.17a, whereas the Figure 7.17c shows the internal circuit symbol.

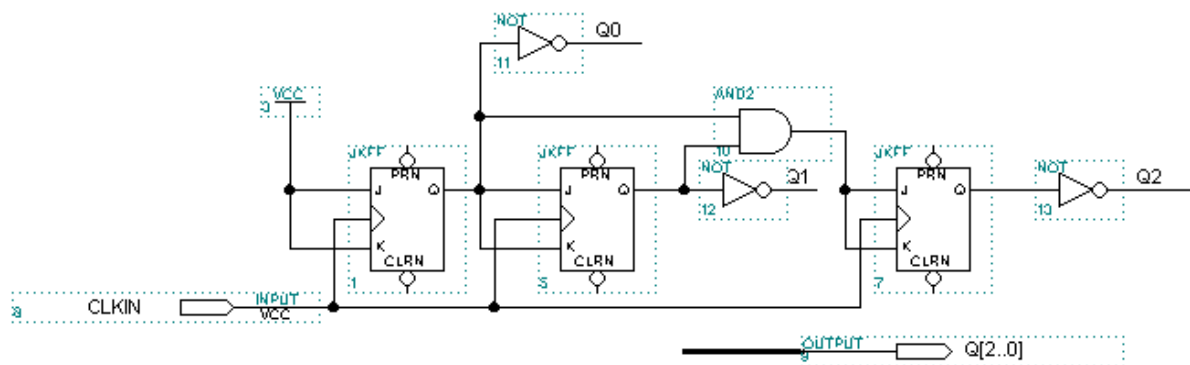


Figure 7.17a MOD 8 Counter

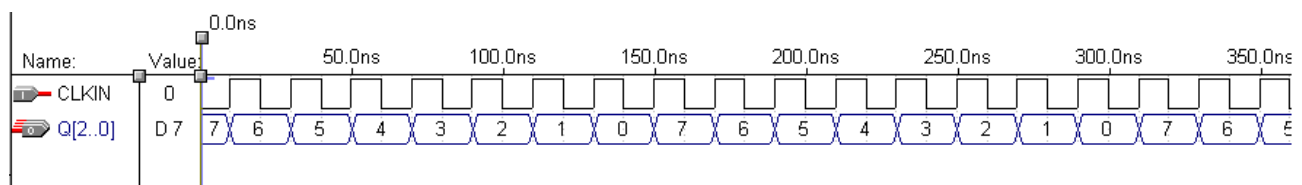


Figure 7.17b Simulation result of MOD 8 Counter

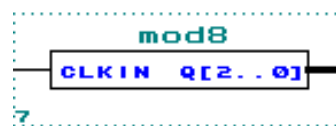


Figure 7.17c The internal circuit symbol of MOD 8 Counter

Step 2: Similarly, please refer to the design, simulation, and verification of 8 to 1 multiplexer illustrated in Section 5.7.1 to design the circuit of data selecting. Figure 7.18c shows the simulation result whereas Figure 7.18b illustrates the internal circuit symbol.

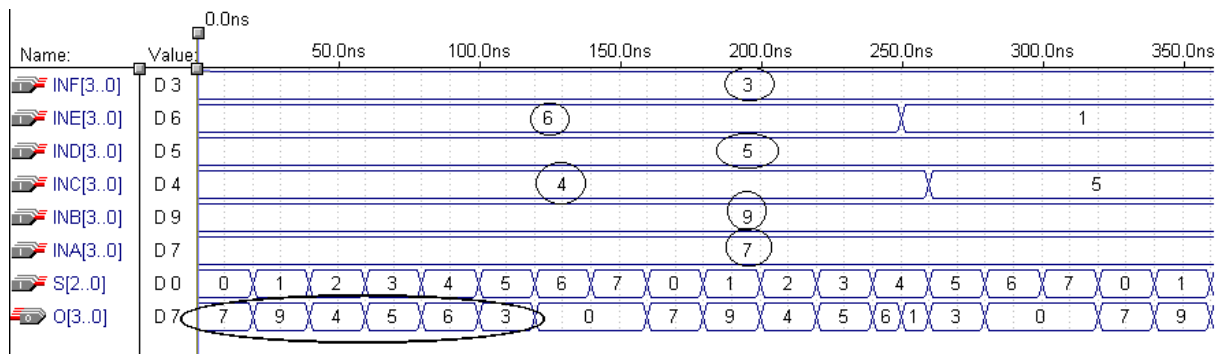


Figure 7.18a Simulation result of data selecting circuit

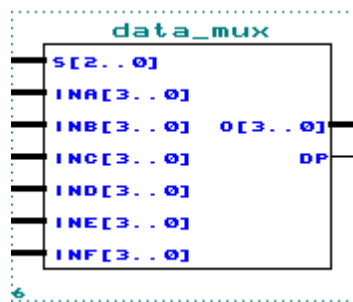


Figure 7.18b The internal circuit symbol of data selecting circuit

Step 3: The scan display circuit, Figure 7.19a, can be composed as the completion of mod8 counter, data selecting circuit, and 7-segment display decoding circuit. Figure 7.19b is the simulation result of Figure 7.19a whereas Figure 7.19c is the internal circuit symbol.

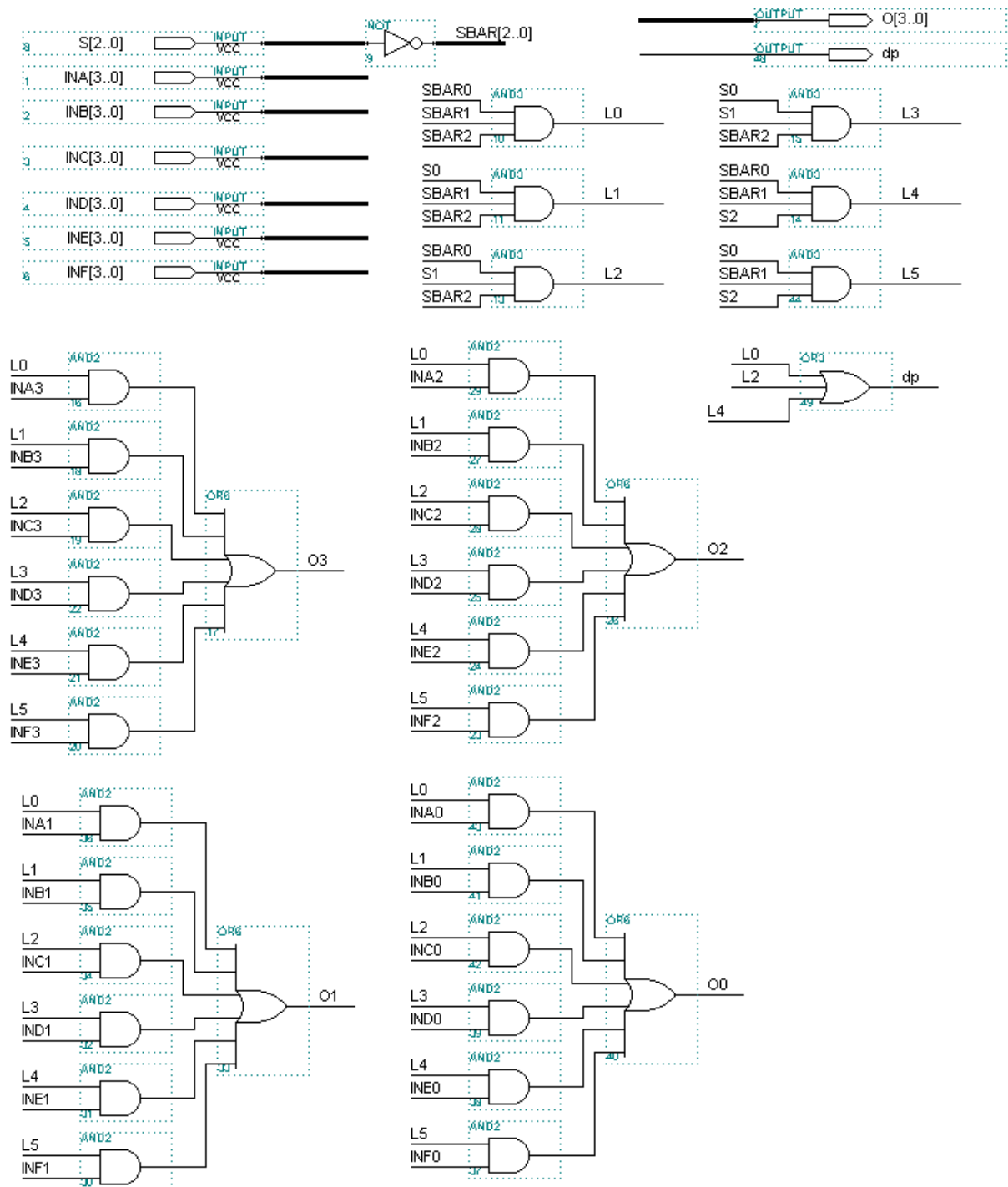


Figure 7.18c Data selecting circuit

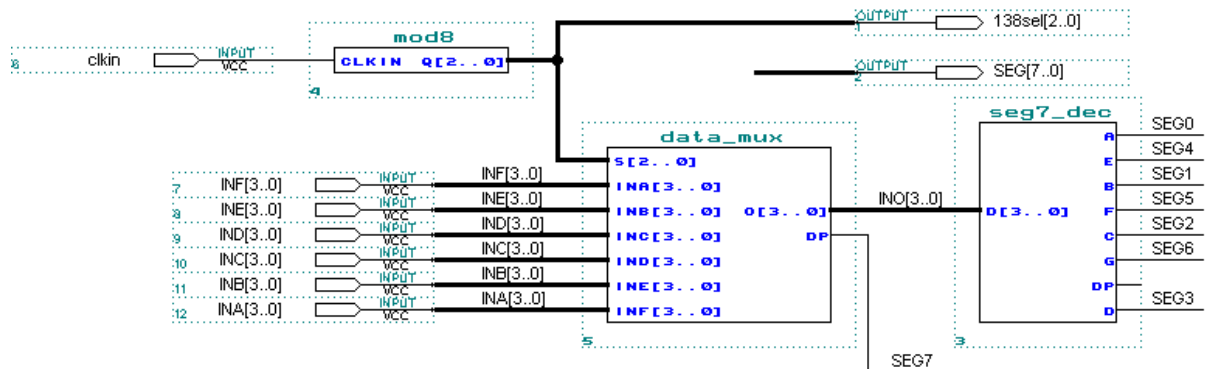


Figure 7.19a Scan display circuit

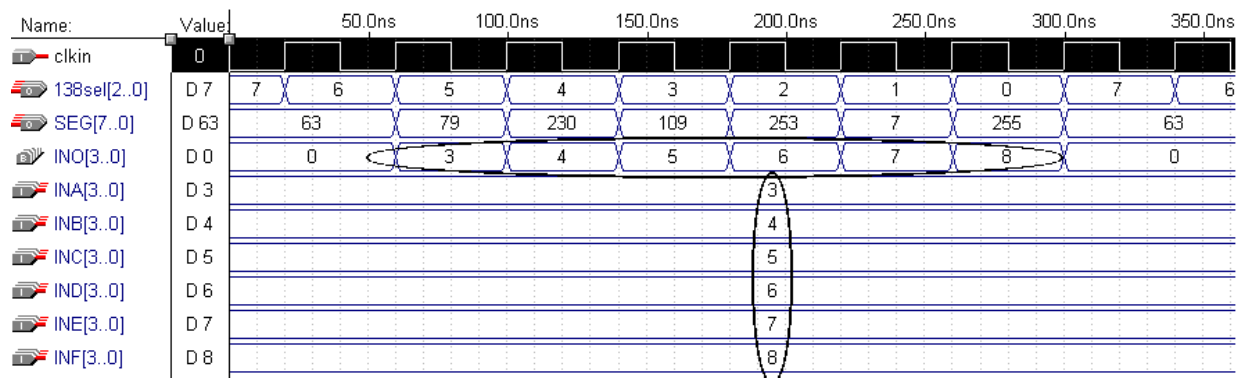


Figure 7.19b Simulation result of scan display circuit



Figure 7.19c The internal circuit symbol of scan display circuit

7.3.5 Timer Circuit

Step 1: Please use the Graphic editor of MAX+PLUS II to create the circuit entry of the following figure (Figure 7.20).

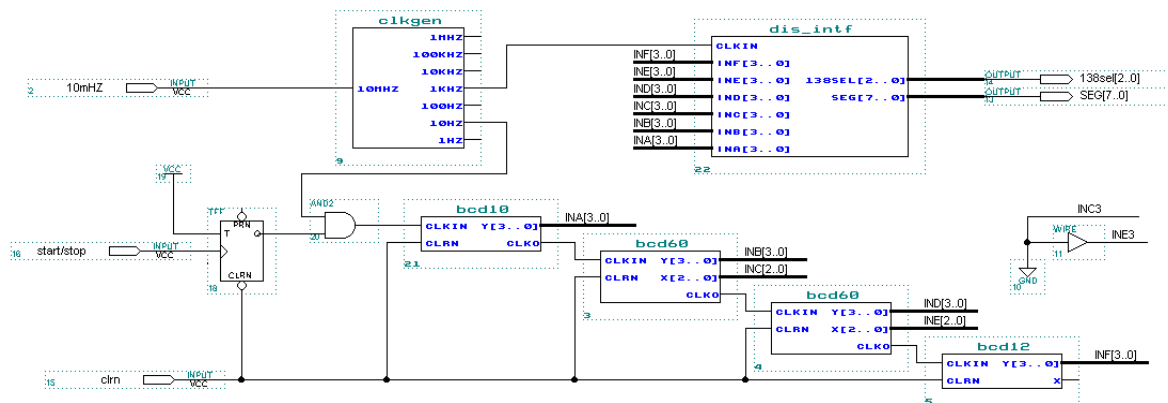


Figure 7.20 Use Graphic editor of MAX+PLUS II to create the Timer Circuit
(File: timer.gdf)

Step 2: Complete functional simulation by using MAX+PLUS II and test whether the functions meet the circuit specification.

~Skip this functional simulation~

Step 3: If the circuits allow download testing, download (or programming) to test the circuit after selecting the download (or programming) device and floorplan program. Please adapt the floorplan programming techniques instructed in Section 4.6. Choose an EPF10K10TC144-4 chip and use the pin assignment shown in Table 7.14. After setting up LP-2900 Lab Platform, download the timer circuit to EPF10K10TC144-4 chip. Please try to push PS1 (START/STOP) on the bottom of LP-2900 and SW4 (clrn) on the left-bottom of LP-2900. At the mean time, please note the changes of 7-segment display and the position of displaying.

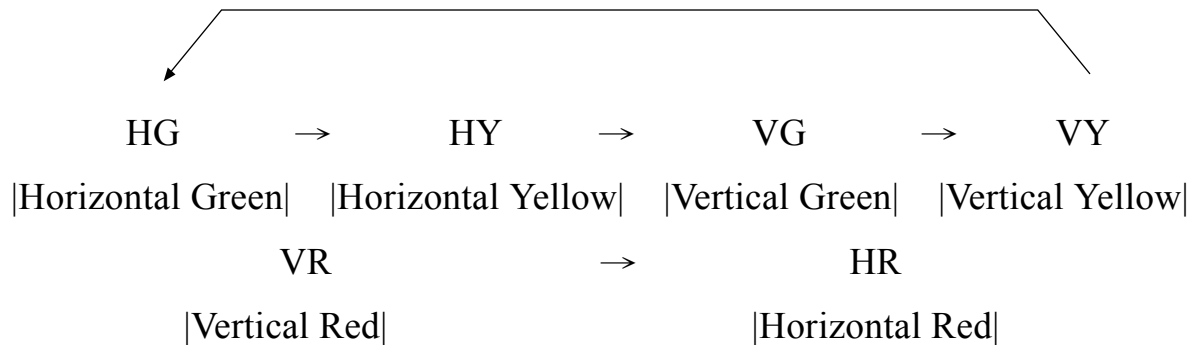


Table 7.14 Pin assignment of EPF10K10TC144-4 chip

Signal	EPF10K10TC144-4 chip pin	Signal	EPF10K10TC144-4 chip pin
Clrn	Pin 51	A	Pin 23
10 MHz	Pin 55	B	Pin 26
Start/stop	Pin 54	C	Pin 27
		D	Pin 28
138 sel0	Pin 33	E	Pin 29
138 sel1	Pin 36	F	Pin 30
138 sel2	Pin 37	G	Pin 31
		H	Pin 32

7.4 Simple Traffic Light Controller

The following is the status changes of the traffic light that we are familiar with:



From the above circuit specifications, the circuits of the state machine, and lightening timer are required for the traffic light controller. The state machine will keep cycling with the states of $HG \rightarrow HY \rightarrow VG \rightarrow VY \rightarrow \dots$ where as the lightening timer is for controlling the flashing time of the light.

7.4.1 State Machine

Step 1: Please complete the state assignment of the circuit specifications and illustrate with a state diagram or a State table; the specifications of the traffic light state machine is shown as the State table of Table 7.15a.

Step 2: Use the Excitation table to figure out each input and output functions' Karnaugh Map or other minimized functions. Please use Table 6.14c, Flip-Flop Excitation table, to figure out the Karnaugh Map of D Type Flip-Flop's Input Functions, as Table 7.15b~Table 7.15g.

Table 7.15a State table of traffic light state machine

Present State	Inputs x_1x_2				Outputs Abcd
	00	01	10	11	
00	00	01	00	00	1000
01	01	01	10	01	0100
10	10	11	10	10	0010
11	11	11	00	11	0001

Note: The double-lined frame shows the next state

Table 7.15b Karnaugh Map of D_0 , $D_0 = x_1'x_0 + Y_0x_1' + Y_0x_0$

D_0		Present State Input x_1x_0			
		00	01	11	10
Y ₁ Y ₀ Present State Input	00	0	1	0	0
	01	1	1	1	0
	11	1	1	1	0
	10	0	1	0	0

Table 7.15c Karnaugh Map of D_1 , $D_1 = Y_1x_1' + Y_1x_0 + Y_1'Y_0x_1x_0'$

D_1		Present State Input x_1x_0			
		00	01	11	10
Y ₁ Y ₀ Present State Input	00	0	0	0	0
	01	0	0	0	1
	11	1	1	1	0
	10	1	1	1	0

Table 7.15d Karnaugh Map of a, $a = Y_1'Y_0'$

a		Present State Input x_1x_0			
		00	01	11	10
Y ₁ Y ₀ Present State Input	00	1	1	1	1
	01	0	0	0	0
	11	0	0	0	0
	10	0	0	0	0

Table 7.15e Karnaugh Map of b, $b = Y_1'Y_0$

b		Present State Input x_1x_0			
		00	01	11	10
Y ₁ Y ₀ Present State Input	00	1	1	1	1
	01	0	0	0	0
	11	0	0	0	0
	10	0	0	0	0

Table 7.15f Karnaugh Map of c, $c = Y_1Y_0'$

c		Present State Input x_1x_0			
		00	01	11	10
Y ₁ Y ₀ Present State Input	00	1	1	1	1
	01	0	0	0	0
	11	0	0	0	0
	10	0	0	0	0

Table 7.15g Karnaugh Map of d, $d = Y_1 Y_0$

d		Present State Input $x_1 x_0$			
		00	01	11	10
$Y_1 Y_0$ Present State Input	00	1	1	1	1
	01	0	0	0	0
	11	0	0	0	0
	10	0	0	0	0

Step 3: Figure out the minimized functions of each Input and output.

$$D_0 = x_1' x_0 + Y_0 x_1' + Y_0 x_0$$

$$D_1 = Y_1 x_1' + Y_1 x_0 + Y_1' Y_0 x_1 x_0'$$

$$a = Y_1' Y_0'$$

$$b = Y_1' Y_0$$

$$c = Y_1 Y_0'$$

$$d = Y_1 Y_0$$

Step 4: Please use the Graphic editor of MAX+PLUS II to complete editing the circuit entry of the above equations (Figure 7.21a) .

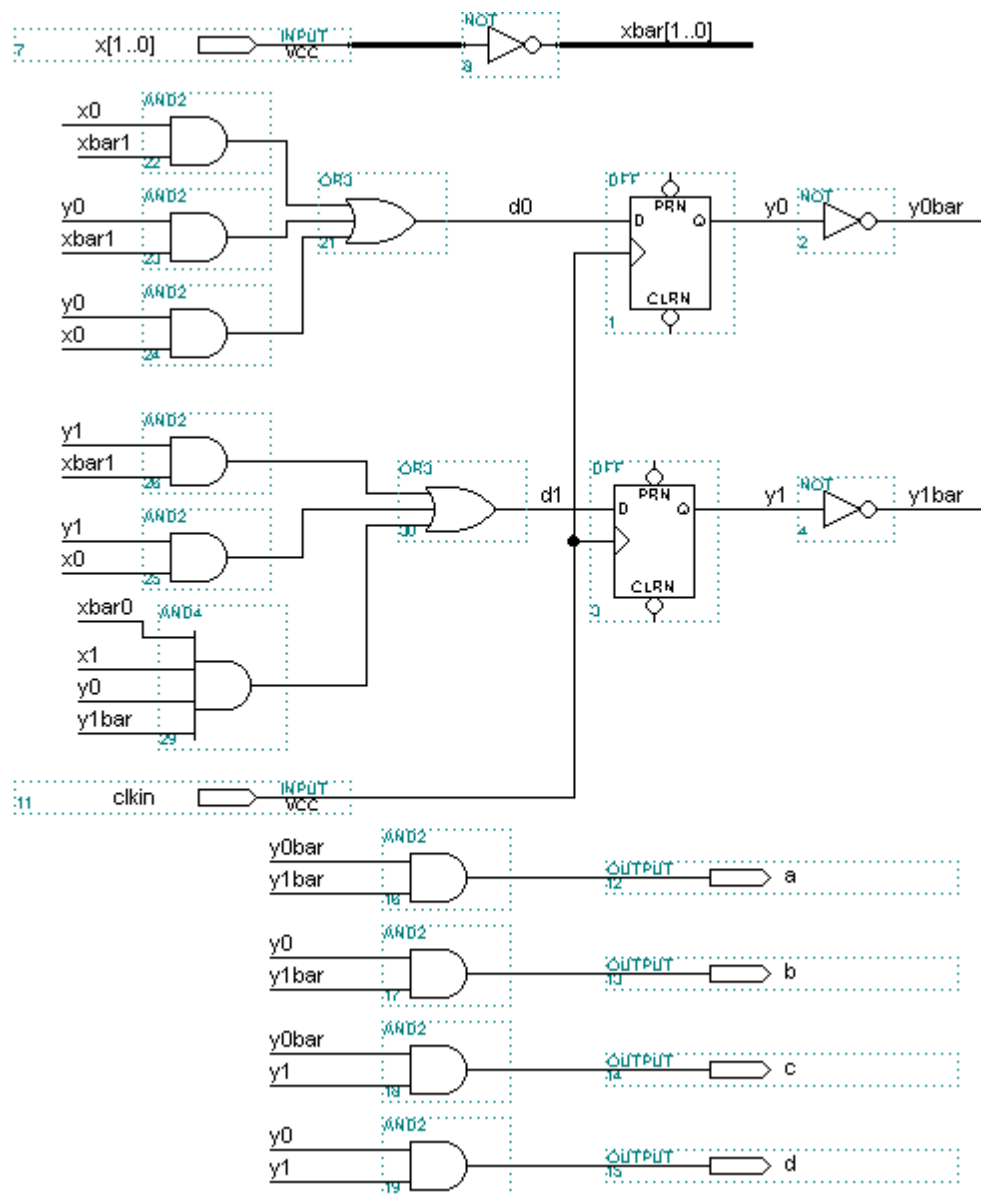


Figure 7.21a Use Graphic editor of MAX+PLUS II to create traffic light state machine circuit (File: traf_stm.gdf)

Step 5: Complete functional simulation by using MAX+PLUS II (Figure 7.21b) and test whether the functions meet the circuit specifications. Go on to the next step if the functions meet the specifications; otherwise go back to step 1 to check the cause of error sequentially.

Step 6: If the circuits allow testing by download, download (or programming) to test the circuit after selecting the download (or programming) device and floorplan program.

~Skip this download testing~

Step 7: Please use File > Create Default Symbol to generate the internal circuit symbol (Figure 7.21c) for the upper layer circuit.

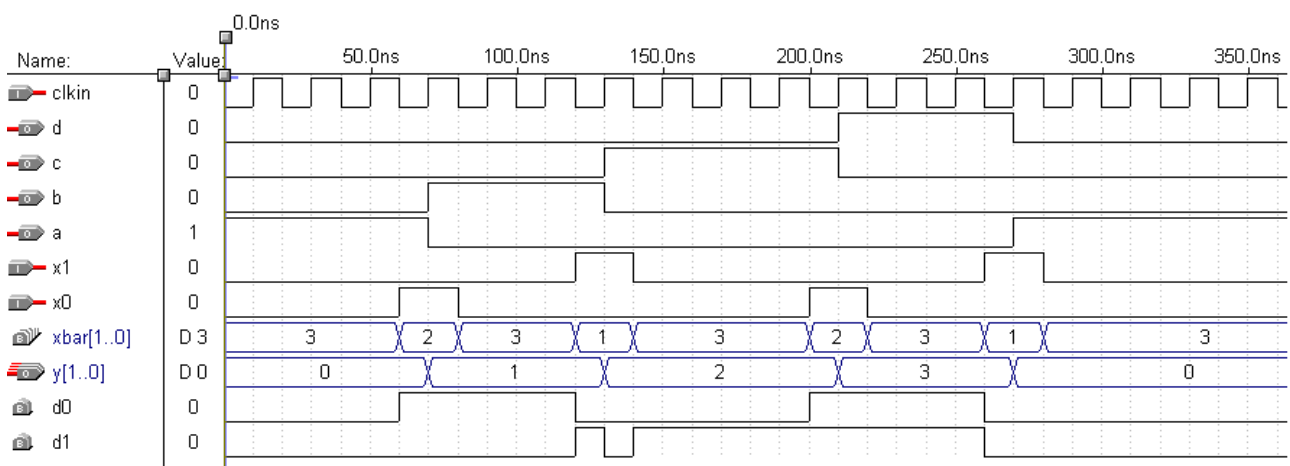


Figure 7.21b Simulation result of traffic light state machine
(File: traf_stm.scf)

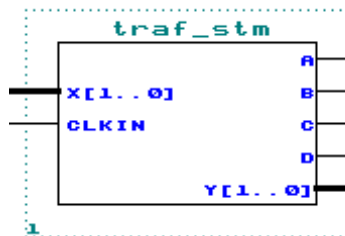


Figure 7.21c The internal circuit symbol of traffic light state machine



7.4.2 Lightening Timer

Refer to Section 6.4.1 of ring counter, each pulse input will make a position movement. In other words, for a 4-step ring counter, a cycle costs four pulses. If each pulse spends one second, then a cycle would spend 4 seconds. This is the reason why we take a ring counter as the timer. Please refer to Section 6.4.1 for detail information of a ring counter. ◦

7.4.3 Simple Traffic Light Controller

Step 1: Please use MAX+PLUS II Graphic editor to complete the circuit entry of Figure 7.22a.

Step 2: Complete functional simulation by using MAX+PLUS II (Figure 7.22b) and test whether the functions meet the circuit specifications.

Step 3: If the circuits allow download testing, download (or programming) to test the circuit after selecting the download (or programming) device and floorplan program.

Please adapt the techniques of floorplan programming instructed in Section 4.6 to choose an EPF 10K10TC144-4 chip and use the pin set up of Table 7.16. After setting up LP-2900 Lab Platform, download the traffic light circuit to the EPF 10K 10TC144-4 chip and try to push SW1 (clrn) on the left bottom of LP-2900. At the same time, please note the changes of L1~L6 LED and the position of lighting.

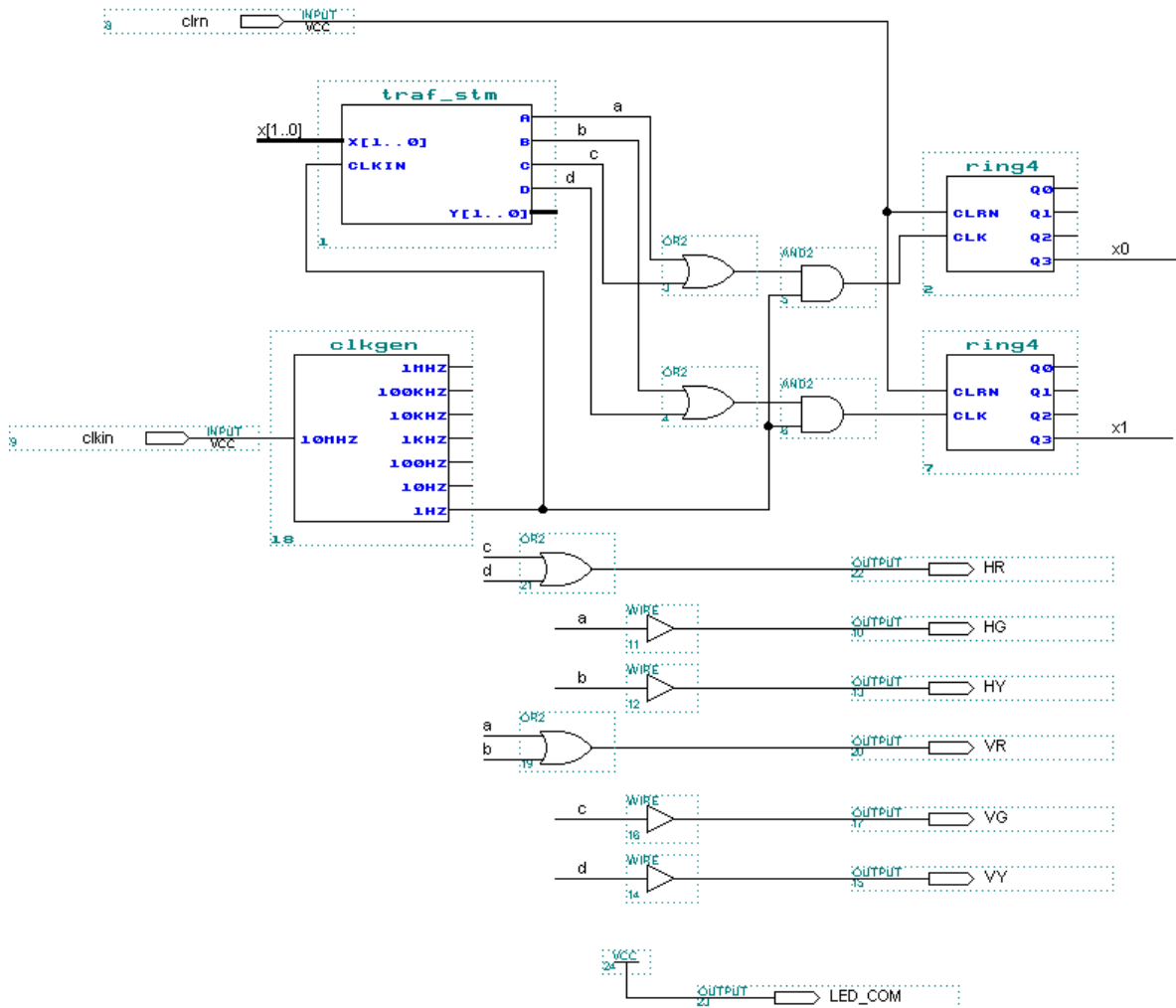


Figure 7.22a Use Graphic editor of MAX+PLUS II to create Traffic Light Controller (File: traf light.gdf)

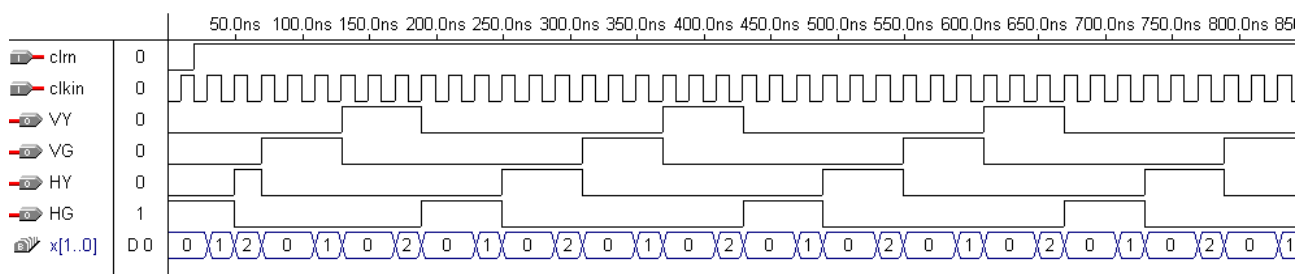


Figure 7.22b Use Graphic editor of MAX+PLUS II to create Traffic Controller Circuit (File: traf light.scf)

Table 7.16 Pin assignment of EPF10K10TC144-4 chip

Signal	EPF10K10TC144-4 chip pin	Signal	EPF10K10TC144-4 chip pin
CLKIN	Pin 55 (OSC)	HG	Pin 12 (L6)
CLRN	Pin 47 (SW1)	HY	Pin 11 (L5)
HG	Pin 9 (L3)	HR	Pin 710 (L4)
HY	Pin 8 (L2)		
HR	Pin 7 (L1)	LED_COM	Pin 141

7.5 Dot Matrix Displayer Test Circuit

Dot Matrix LED is a common display device in digital circuit. The appearance, which may have two types, is shown as Figure 7.23a. One is uni-color dot matrix LED and the other is bi-color dot matrix LED. For a bi-color one, the structure is illustrated as Figure 7.23b. In each circle, there exists two LEDs, which one is red, and the other is green. Therefore, the combination shows 4 conditions: unlighted, red light, green light and yellow light. From the construction layout, signal Row has to be sent to “1” together with sending Col to “0”, the relative LED would consequently flash.

In this section, a simple test circuit of a bi-color dot matrix LED is drawn here as an illustration. The progress of design is as follows:

- Step 1: Use the Graphic editor of MAX+PLUS II to complete the circuit entry as illustrated in Figure 7.24.
- Step 2: Complete functional simulation by using MAX+PLUS II and test whether the functions meet the circuit specifications.

~Skip the functional simulation~

Step 3: If the circuits allow download testing, download (or programming) to test the circuit after selecting the download (or programming) device and floorplan program. Please adapt the techniques of floorplan programming instructed in Section 4.6 to choose an EPF10K10TC144-4 chip, and use the pin assignment listed in Table 7.17. After setting up LP-2900 Lab Platform, please download the test circuit of simple bi-color dot matrix LED to EPF10K10TC144-4 chip. Please notice the changes of bi-color dot matrix LED.

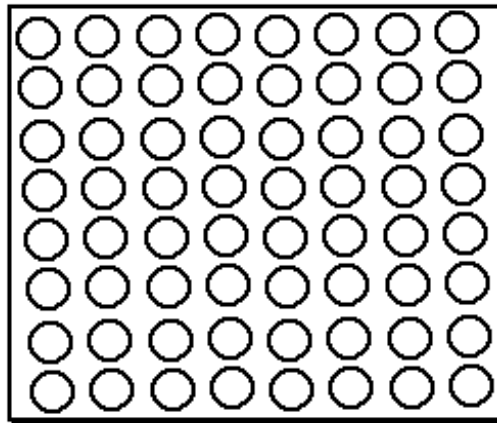


Figure 7.23a The appearance of dot matrix LED

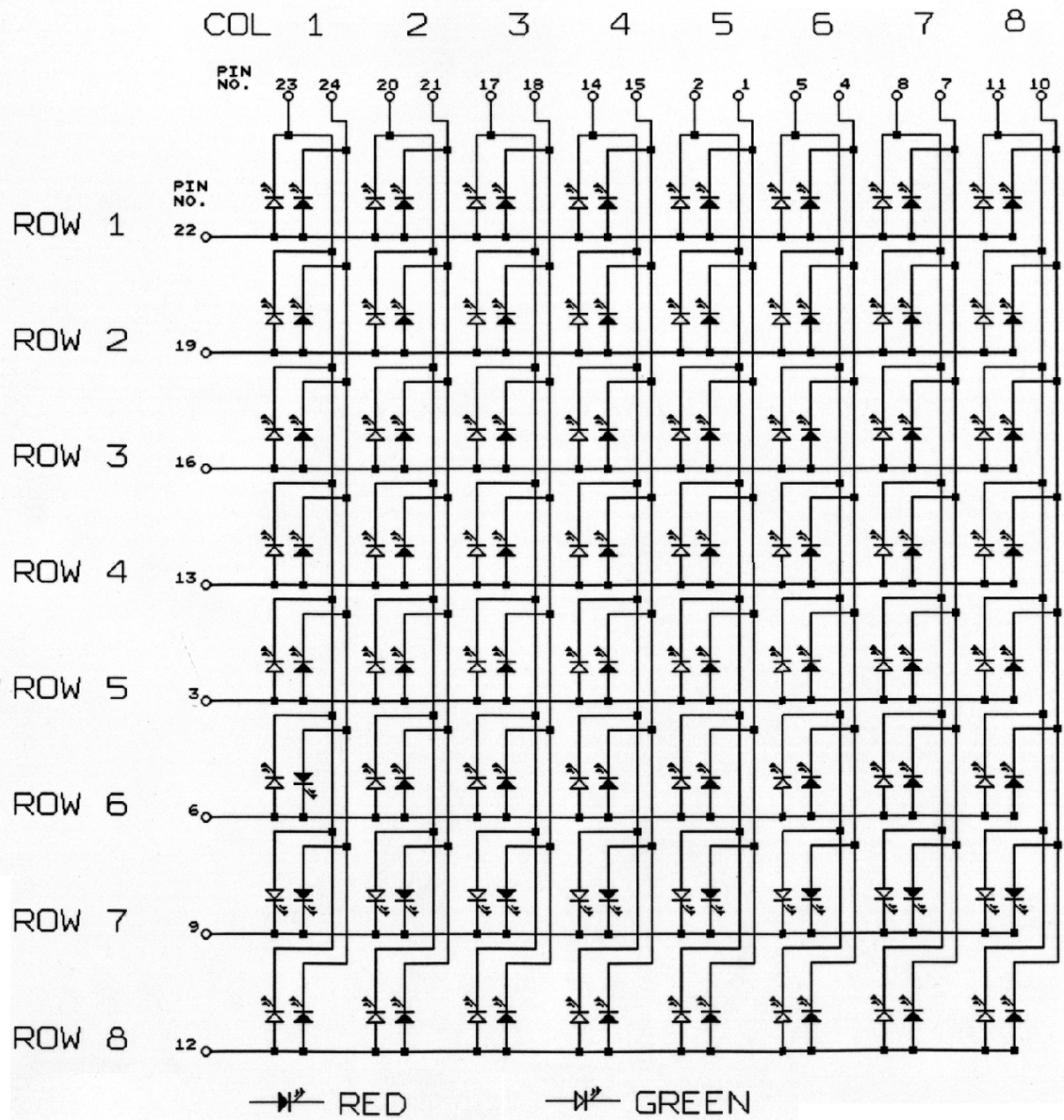


Figure 7.23b The structure of a bi-color dot matrix LED

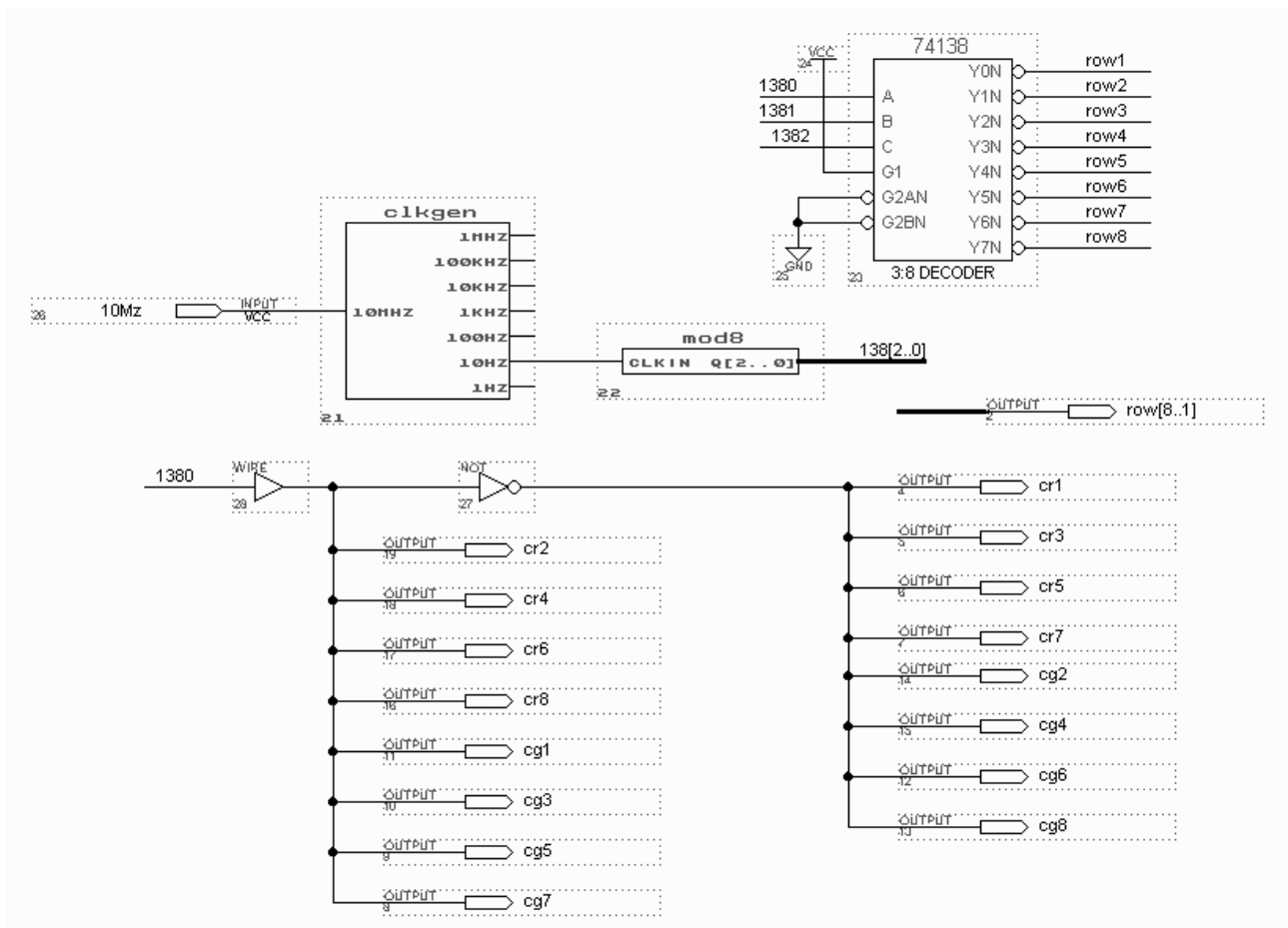


Figure 7.24 Use the Graphic editor of MAX+PLUS II to create a test circuit board of simple bi-color dot matrix LED (File: dot_mtrx.gdf)

Table 7.17 Pin assignment of EPF10K10TC144-4 chip

Signal	EPF10K10TC144-4 chip pin	Signal	EPF10K10TC144-4 chip pin
10 Mhz	Pin 55		
Cr1	Pin 98	Cg1	Pin 112
Cr2	Pin 99	Cg2	Pin 113
Cr3	Pin 100	Cg3	Pin 114
Cr4	Pin 101	Cg4	Pin 116
Cr5	Pin 102	Cg5	Pin 117

Cr6	Pin 109	Cg6	Pin 118
Cr7	Pin 110	Cg7	Pin 119
Cr8	Pin 111	Cg8	Pin 120
Row 1	Pin 88	Row 5	Pin 92
Row 2	Pin 89	Row 6	Pin 95
Row 3	Pin 90	Row 7	Pin 96
Row 4	Pin 91	Row 8	Pin 97

7.6 Keyboard Scan and Display Scan Circuit

Keyboard and 7-segment displayer are the common input and output devices in the digital system. The input of keyboard scan and the display circuit of 7-segment displayer are, therefore, become the required circuits. The scan display circuit of 7-segment displayer has instructed in Section 7.3.4. Hereafter is the design descriptions of the integrated keyboard scan.

♣ Principles of keyboard scan:

The left diagram of Figure 7.25a shows an illustration of a 4×3 keyboard. The black dot depicts the connections when pushing the button, whereas the white dot depicts the non-connection of non-pushed buttons. In the left diagram of Figure 7.25a, scan signals generator circuit creates signals $c_3 \sim c_0$, which are sent to keyboard and keyboard decoder circuit separately. The changing sequences of $c_3 \sim c_0$ are : “1110” \rightarrow “1101” \rightarrow “1011” \rightarrow “0111” \rightarrow “1110” $\rightarrow \dots$, and cycling. Hypothesis that now $c_3 \sim c_0$ are “1101”. $r[2..0] = “111”$ because the button is not

push down. After decoding through keyboard decoder circuit, the result is that $d_3 \sim d_0$ are “1111” and shift latch clock, the control signals for shift and latch data, is “0”. As in the right diagram of Figure 7.25a, hypothesis that $c_3 \sim c_0$ are “1101”. Because the button of 6 is push down, $r[2..0] = “110”$. After decoding through keyboard decoder circuit, the result is that $d_3 \sim d_0$ are “0110”, while the shift latch clock turns to “1”.

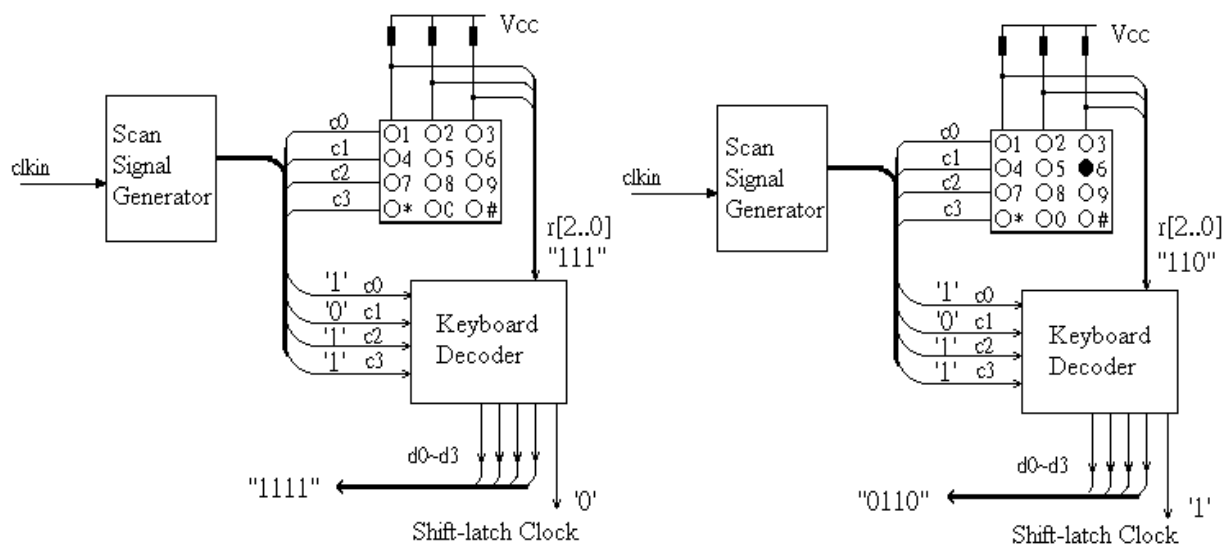


Figure 7.25a Block diagram of keyboard scanner and decoder

In the left diagram of Figure 7.25b, $c_3 \sim c_0$ are “1101” and because the button of 4 is push down, $r[2..0] = “011”$. Through keyboard decoder circuit, $d_3 \sim d_0$ turn to “0100” and the shift latch clock is “1”. In the right diagram of Figure 7.25b, $c_3 \sim c_0$ are “1101” and because the button of 5 is push down, $r[2..0] = “101”$. From this pattern, we know that when $c_1 = “0”$, go on to check weather the button of 4, 5 and 6 is push down individually, then we will get the value of $r[2..0]$ which would be sent with $c_3 \sim c_0$ to the keyboard decode circuit for decoding. Accordingly, when $c_0 = “0”$, check weather the button of 1, 2 and 3 is bush down individually; when $c_2 = “0”$, check weather the button of 7, 8, and 9 is push down; when $c_3 = “0”$, check

weather “*”, “0”, and “#” is push down.

Keyboard decode circuit implements decoding by means of the input of $c3 \sim c0$ and $r[2..0]$.

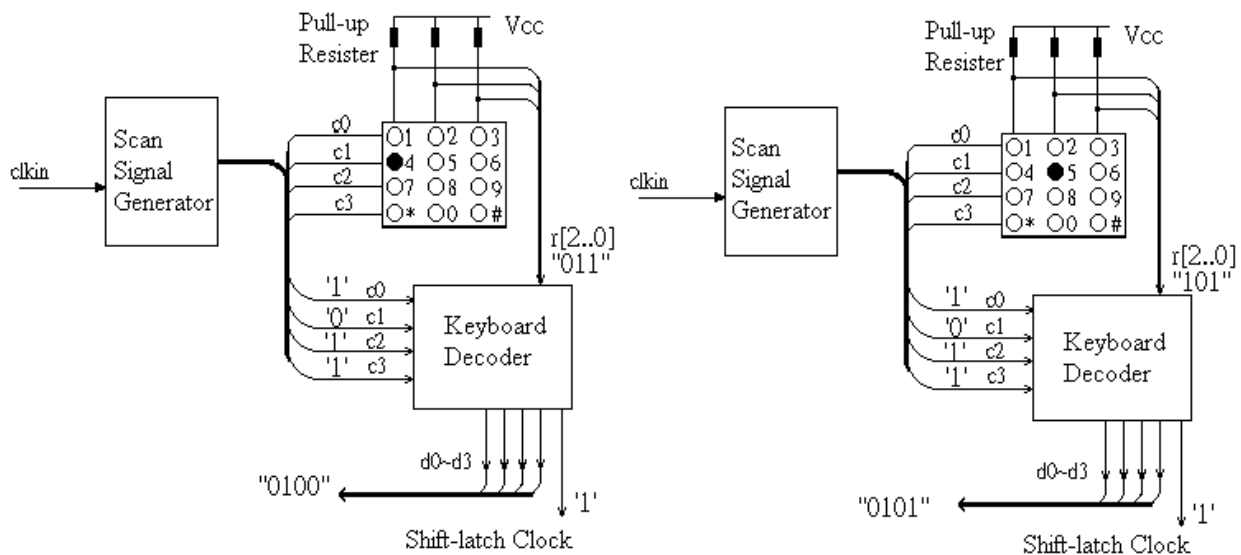


Figure 7.25b Block Diagram of keyboard scanner and decoder (Continue)

To form a keyboard scan and display scan circuit, three types of sub-circuits are required: (1) Scan signal generate circuit, disbounce and (single-key detector circuit); (2) Keyboard decoder circuit; (3) Data buffer circuit, display scan circuit of 7-segment display, as illustrated in Figure 7.26. Scan signal generator circuit creates the signals of scan keyboard, usually scans 7-segment displayer as well. Disbounce and single-key detector eliminate the bounce when pushing the key and prevent multiple keys input. Keyboard decoder can decode the input keys. Data buffer circuit keeps the latest six input keys. The displays scan circuits of the 7-segment displayer in charge of displaying the data in buffer to the six 7-segment displayers. However, in LP-2900, the $c3 \sim c0$ scan wires on the keyboard are same as the wires $C3 \sim C0$ 7-segment displayer. In other words, scan signals is the scan wires of 7-segment displayer. Please refer to Section 7.3.4 for the scan and display circuits of

the 7-segment displayer. The data buffer is composed by 6 sets of 4-bit parallel shift registers for storing the code sent by keyboard decoder. Directions of shift are: new push code→Set 1→Set 2→Set 3→Set 4→Set 5→Set 6→....

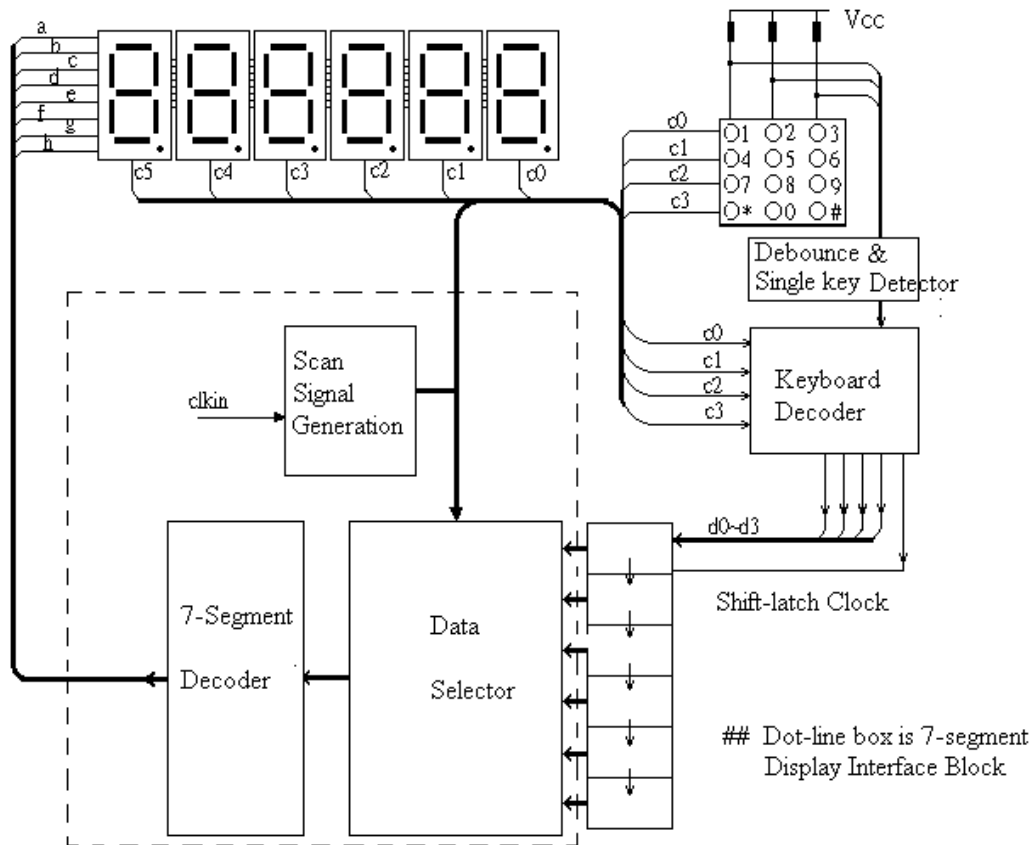


Figure 7.26 Functional Diagram of keyboard scan and display scan circuit

7.6.1 Disbounce and Single-key detector

Step 1: Please use the Graphic editor of MAX+PLUS II to edit a circuit entry as shown in Figure 7.27a. Usually, the buttons would bump and lead to a misunderstanding of multiple inputs. Therefore, it is necessary to create a disbounce mechanism. Please refer to Chapter 4 for designing and functional simulation of the disbounce circuit. In this section, we will adapt it as the blueprint for designing a proper disbounce cell of 4×3 keyboard

as Figure 7.27a. Figure 7.27b is the internal circuit symbol. We can compose a 4 x 3 disbounce circuit array after creating a disbounce cell or according to the disbounce cell to design a horizontal single-key detector and vertical single-key detector circuits, as Figure 7.27d and Figure 7.27e.

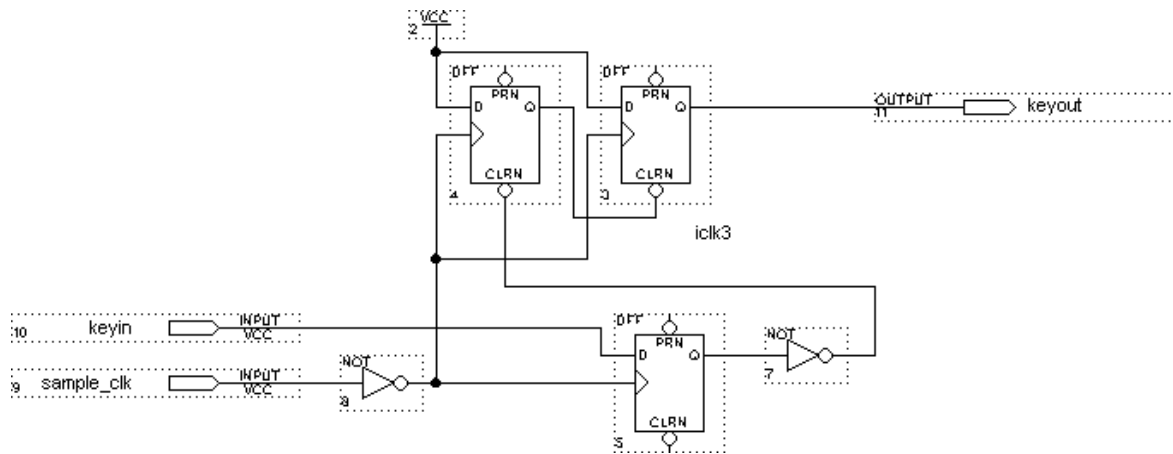


Figure 7.27a Disbounce circuit

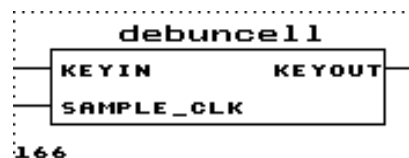


Figure 7.27b Circuit symbol of disbounce

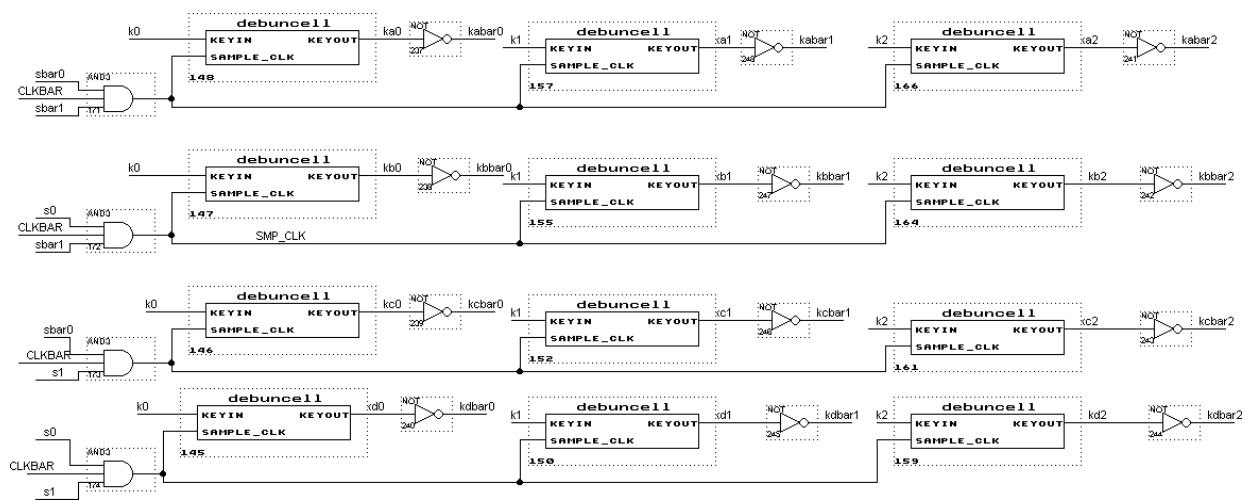


Figure 7.27c 4 × 3 disbounce circuits array (File: key_debun.gdf)

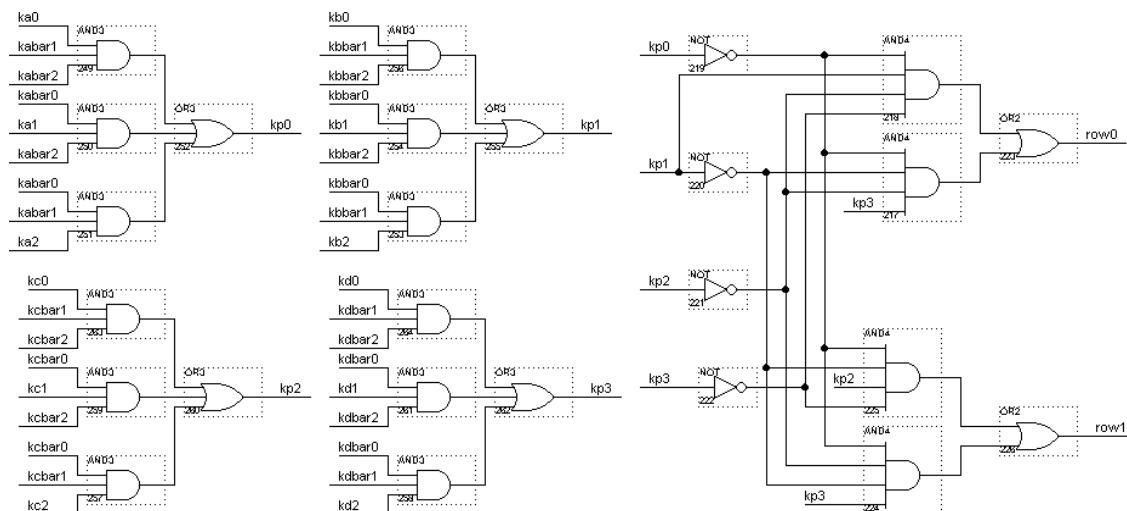


Figure 7.27d 4 × 3 keyout Circuits (File: key_debun.gdf)

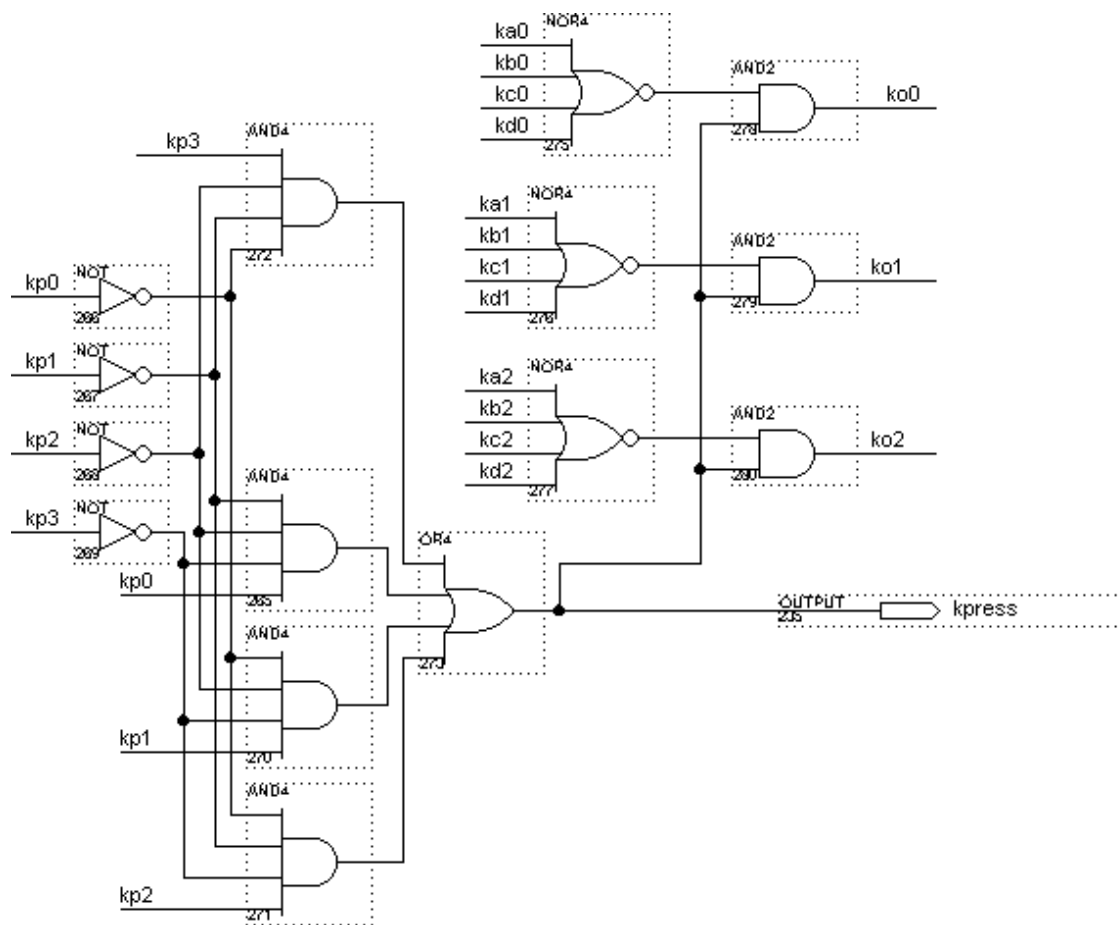


Figure 7.27e 4×3 keyout disbounce cricuits (Continue) (File: key_debun.gdf)

Step 2: Complete functional simulation by using MAX+PLUS II (Figure 7.27f) and test whether the functions meet the circuit specifications. Please notice the relations of the circles on the Figure below.

Step 3: If the circuits allow downloading to test, download (or programming) to test the circuit after selecting the download (or programming) device and floorplan program. For download testing, please create the internal circuit symbol, Figure 7.27, of key_debun and modify the circuits as illustrated in Figure 7.27g. Further, please adapt the techniques of floorplan programming instructed in Section 4.6, and select EPF10K10TC144-4 chip as well as the pin assignment listed in Table 7.18.

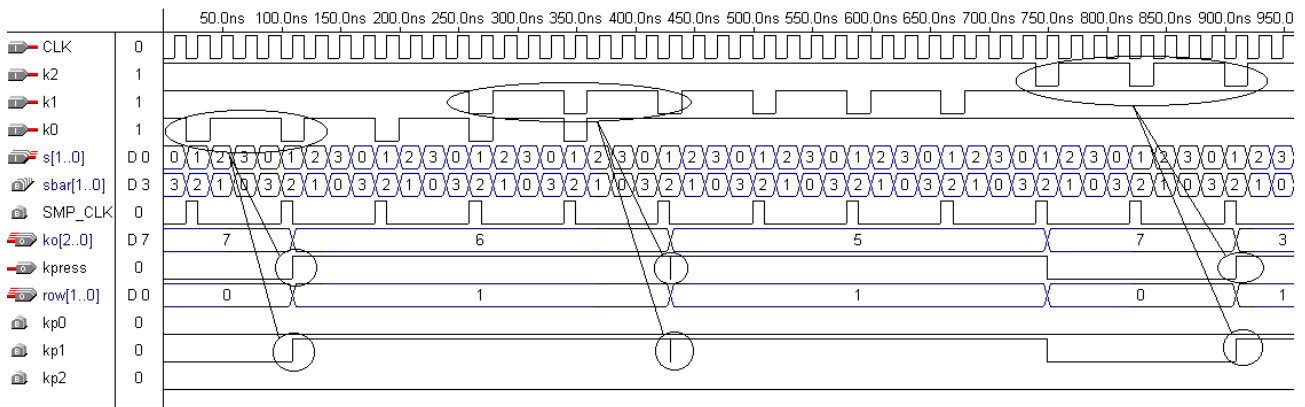


Figure 7.27f Functional simulation results of kbd_debun.gdf circuit
(File: key_debun.scf)

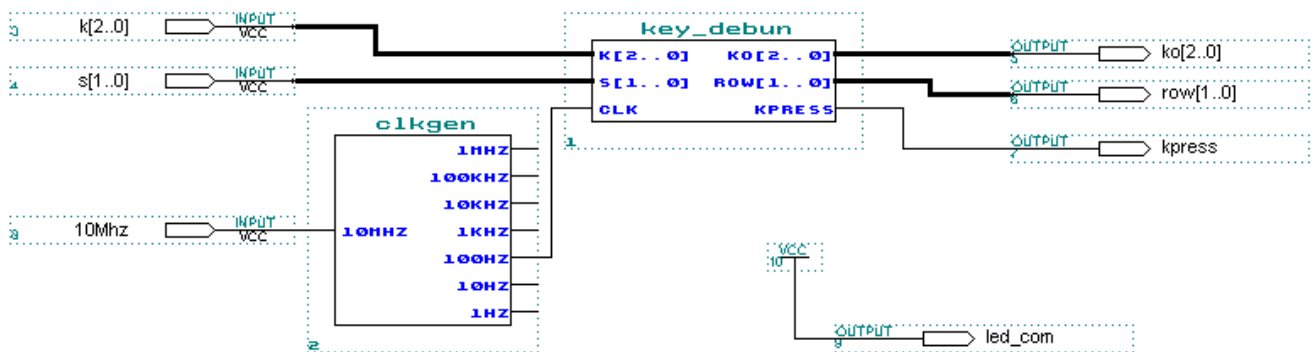


Figure 7.27g debun_test.gdf

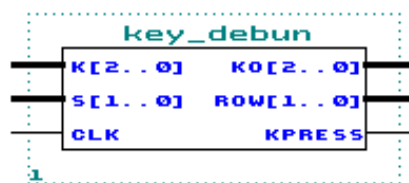


Figure 7.27h The internal circuit symbol of key_debun

Table 7.18 Pin assignment of EPF10K10TC144-4 chip

Signal	EPF10K10TC144-4 chip pin	Signal	EPF10K10TC144-4 chip pin
10Mhz	Pin 55		
Ko0	Pin 7	S0	Pin 47
Ko1	Pin 8	S1	Pin 48
Ko2	Pin 9		
Kpress	Pin 10	K0	Pin 49
Row 0	Pin 11	K1	Pin 51
Row 1	Pin 12	K2	Pin 59

After setting up LP-2900 Lab Platform, please push down the keys of SW3, SW4, and SW5 which means the inputs of K0, K1, and K2 are “1”. Then, please download the file, debun_test.sof, to EPF-10K10TC144-4 chip. Pull up SW3, or SW4, or SW5 to check weather the changes of L1 to L6 meet the requirements. Please note weather there is outputs when pulling up two or three keys simultaneously. Pushing down the keys of SW3, SW4, and SW5 at the same time and adjust SW1 (S0) and SW2 (S1) to another combination and repeat the actions to check.

7.6.2 Keyboard Decoder

Step 1: Form a Truth table (Table 7.19a).

Step 2: From the above Truth table (Table 7.19a) leads to the following Karnaugh Map and equations. (Table 7.19b to Table 7.19e)

Table 7.19a The State table of keyboard decoder

Inputs (From Scan Circuit) s_1s_0	Inputs (From keyboard) $k_2k_1k_0$	Outputs $d_3d_2d_1d_0$	Descriptions
00	111	1111	No action
00	011	0001	Push Key “1”
00	101	0010	Push Key “2”
00	110	0011	Push Key “3”
01	111	1111	No action
01	011	0100	Push Key “4”
01	101	0101	Push Key “5”
01	110	0110	Push Key “6”
10	111	1111	No action
10	011	0111	Push Key “7”
10	101	1000	Push Key “8”
10	110	1001	Push Key “9”
11	111	1111	Nil
11	011	1010	Push Key “*”
11	101	0000	Push Key “0”
11	110	1011	Push Key “#”

Table 7.19b Karnaugh Map of D_0 ,

$$D_0 = k_2k_1k_0 + s_1k_2k_1 + s_0'k_1k_0 + s_1's_0k_2k_0 + s_0'k_2k_1$$

D_0			Present State Input s_1s_0			
			00	01	11	10
Present State Input k_2k_1	00	$K_0 = 0$	0	0	0	0
		$K_0 = 1$	0	0	0	0
	01	$K_0 = 0$	0	0	0	0
		$K_0 = 1$	1	0	0	1
	11	$K_0 = 0$	1	0	1	1
		$K_0 = 1$	1	1	1	1
	10	$K_0 = 0$	0	0	0	0
		$K_0 = 1$	0	1	0	0

Table 7.19c Karnaugh Map of D_1 , $D_1 = k_2k_1k_0 + s_1'k_2k_1 + s_0k_2k_1 + s_1k_1k_0 + s_1's_0'k_2k_0$

D_1			Present State Input s_1s_0			
			00	01	11	10
Present State Input k_2k_1	00	$K_0 = 0$	0	0	0	0
		$K_0 = 1$	0	0	0	0
	01	$K_0 = 0$	0	0	0	0
		$K_0 = 1$	0	0	1	1
	11	$K_0 = 0$	1	1	1	
		$K_0 = 1$	1	1	1	1
	10	$K_0 = 0$	0	0	0	0
		$K_0 = 1$	1	0	0	0



Table 7.19d Karnaugh Map of D_2 , $D_2 = k_2k_1k_0 + s_1's_0k_2k_0 + s_1's_0k_2k_1 + s_1's_0k_1k_0 + s_1s_0'k_1k_0$

D_2			Present State Input s_1s_0			
			00	01	11	10
Present State Input k_2k_1	00	$K_0 = 0$	0	0	0	0
		$K_0 = 1$	0	0	0	0
	01	$K_0 = 0$	0	0	0	0
		$K_0 = 1$	0	1	0	1
	11	$K_0 = 0$	0	1	0	0
		$K_0 = 1$	1	1	1	1
	10	$K_0 = 0$	0	0	0	0
		$K_0 = 1$	0	1	0	0

Table 7.19e Karnaugh Map of D_3 , $D_3 = k_2k_1k_0 + s_1k_2k_1 + s_1s_0'k_2k_0 + s_1s_0k_1k_0$

D_3			Present State Input s_1s_0			
			00	01	11	10
Present State Input k_2k_1	00	$K_0=0$	0	0	0	0
		$K_0=1$	0	0	0	0
	01	$K_0=0$	0	0	0	0
		$K_0=1$	0	0	1	0
	11	$K_0=0$	0	0	1	1
		$K_0=1$	1	1	1	1
	10	$K_0=0$	0	0	0	0
		$K_0=1$	0	0	0	1

Step 3: Figure out the minimized equations of each Input and output functions

$$D0 = k_2k_1k_0 + s_1k_2k_1 + s_0'k_1k_0 + s_1's_0k_2k_0 + s_0'k_2k_1$$

$$D1 = k_2k_1k_0 + s_1'k_2k_1 + s_0k_2k_1 + s_1k_1k_0 + s_1's_0'k_2k_0$$

$$D2 = k_2k_1k_0 + s_1's_0k_2k_0 + s_1's_0k_2k_1 + s_1's_0k_1k_0 + s_1s_0'k_1k_0$$

$$D3 = k_2k_1k_0 + s_1k_2k_1 + s_1s_0'k_2k_0 + s_1s_0k_1k_0$$

Step 4: Please use the Graphic editor of MAX+PLUS II to complete editing the circuit entry of the above equations (Figure 7.28a)

Step 5: Complete functional simulation by using MAX+PLUS II (Figure 7.28b) and test whether the functions meet the circuit specifications. Go on to the next step if the functions meet the specifications; otherwise go back to step 1 to check the cause of error sequentially.

Step 6: Please download the circuits after floorplan programming and test the circuits. Modify Figure 7.28a as Figure 5.3 in Section 5.1 and re-compile. Further, please adapt the techniques of floorplan programming instructed in Section 4.6, and select EPF10K10TC144-4 chip as well as the pin setup listed in Table 7.20.

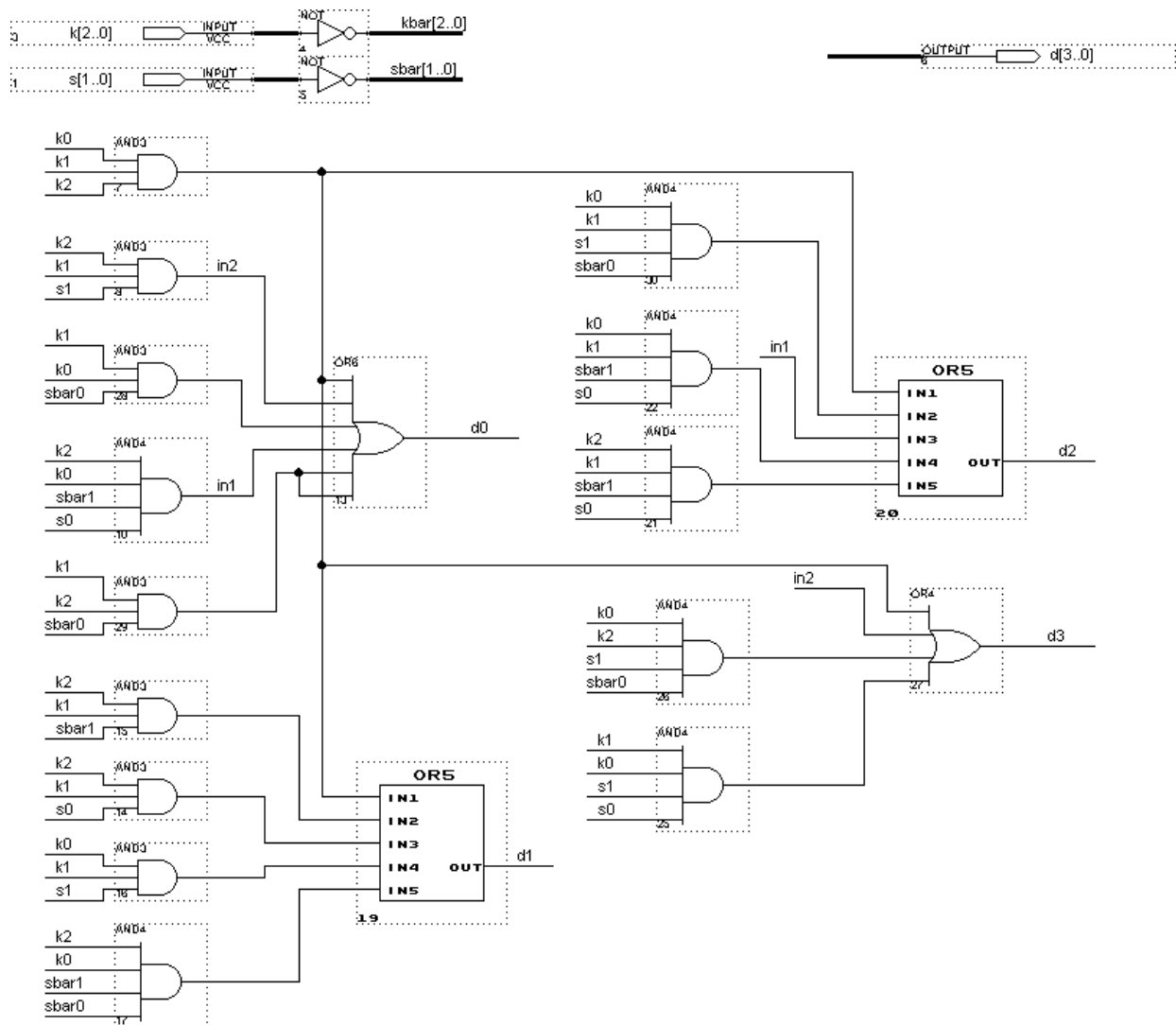


Figure 7.28a Keyboard decoder created by the Graphic editor of MAX+PLUS II
(File: kbd_dec.gdf)

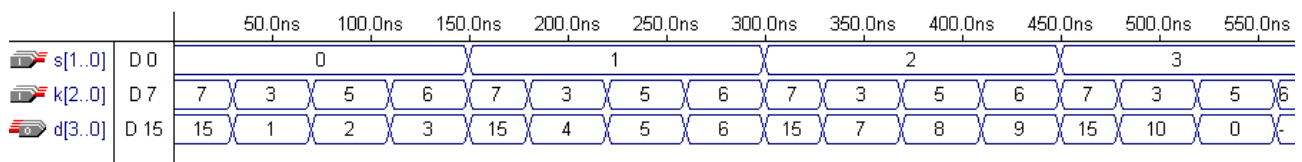


Figure 7.28b Simulation result of Keyboard decoder (Figure 7.28a)
(File: kbd_dec.scf)

Table 7.20 Pin assignment of EPF10K10TC144-4 chip

Signal	EPF10K10TC144-4 chip pin	Signal	EPF10K10TC144-4 chip pin
D0	Pin 10	K0	Pin 47
D1	Pin 9	K1	Pin 48
D2	Pin 8	K2	Pin 49
D3	Pin 7	S0	Pin 51
		S1	Pin 59
LED_COM	Pin 141		

After setting up LP-2900 Lab Platform, please download the keyboard decode circuit to EPF10K10TC144-4 chip. Try to push down SW1 (k0), or SW2 (k1) or SW3 (K2) on the left bottom of LP-2900 and note the changes of L1 (D3), L2 (D2), L3 (D1), and L4 (D0). Adjust the inputs of S0 and S1 and, again, push down SW1 (k0) or SW2 (k1) or SW3 (K2), and note the changes of L1 (D3), L2 (D2), L3 (D1), and L4 (D0).

Step 7: Creating the internal circuit symbol (Figure 7.28) by means of File > Create Default Symbol for the upper circuits.

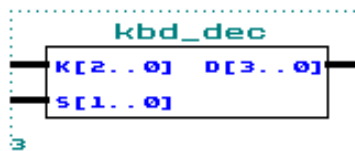
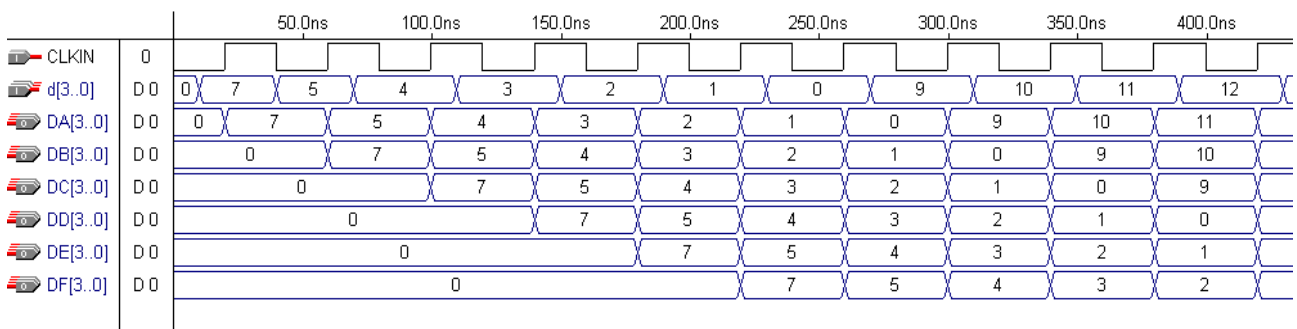


Figure 7.28c The internal circuit symbol of keyboard decoder
(Figure 7.28a)

7.6.3 Data Buffer

Data Buffer (Figure 7.29b) is composed by 6 sets of 4-bit parallel shift register, for storing the key code sent by keyboard decode circuit. The directions of the shift are: new key code→Set 1→Set 2→Set 3→Set 4→Set 5→Set 6→.... As a matter of fact, this is the bit-expansion of one bit SIPO in Section 6.3.2.

Figure 7.29a Simulation result of data buffer circuit (File: data_buf.scf)



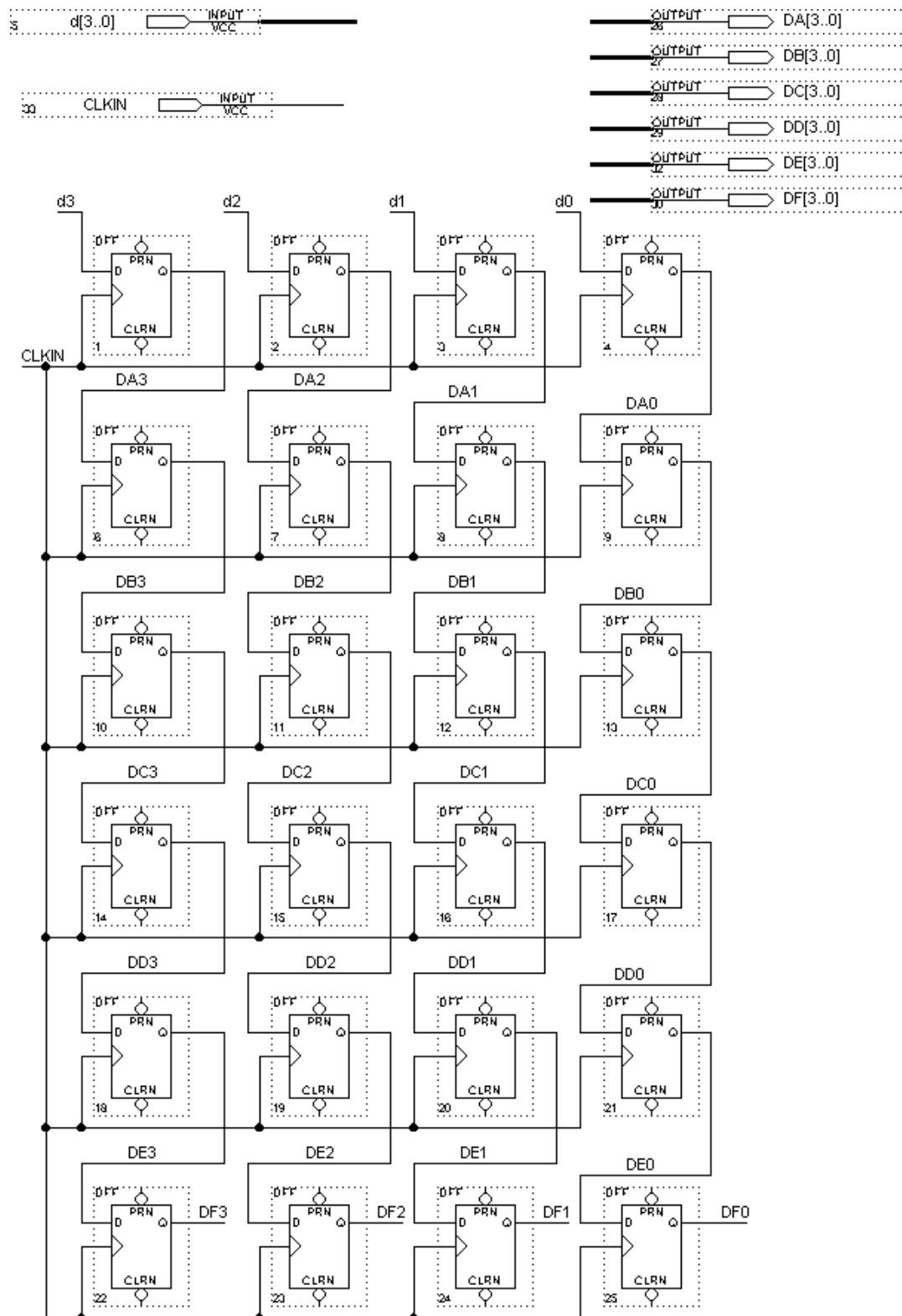


Figure 7.29b Use Graphic editor of MAX+PLUS II to create data buffer circuit

(File: data_buf.gdf)

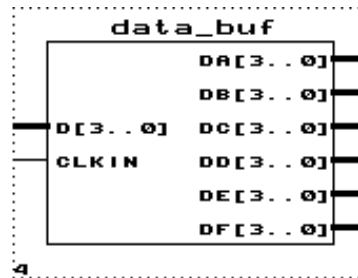


Figure 7.29c The internal circuit symbol of data buffer

7.6.4 Complete Keyboard Scan and Display Scan Circuit

Step 1: Use Graphic editor MAX+PLUS II to create a circuit entry as illustrated in Figure 7.30.

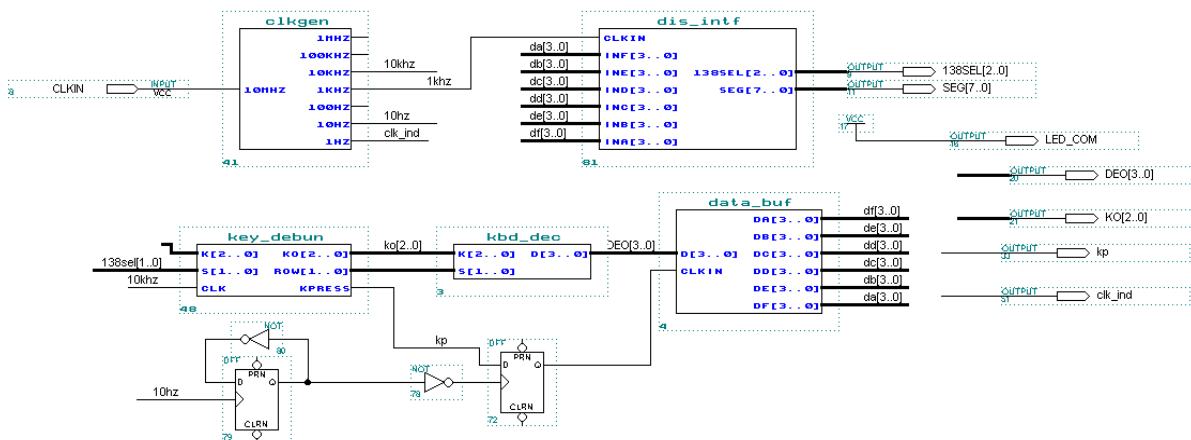


Figure 7.30 Keyboard scan and display scan by using graphic editor of MAX+PLUS II (File: kbd_7seg.gdf)

Step 2: Complete functional simulation by using MAX+PLUS II and test whether the functions meet the circuit specifications.

~Skip the functional simulation~



Step 3: If the circuits allow downloading to test, download (or programming) to test the circuit after selecting the download (or programming) device and floorplan program.

After setting up LP-2900 Lab Platform, please select EPF10K10TC144-4 chip and the pin setup listed in Table 7.21. Try to push down SW4 (clrn) on the middle bottom of LP-2900 and note the changes of 7-segment displayer and the changes of L1 (CLK_IND), L2 (KP), L4 (DE0), L5 (DE1), L6 (DE2), L7 (DE3), L10 (KO2), L11 (KO1) and L12 (KO2).

Table 7.21 Pin assignment of EPF10K10TC144-4 chip

Signal	EPF10K10TC144-4 chip pin	Signal	EPF10K10TC144-4 chip pin
LED_COM	Pin 141	A	Pin 23
10 Mhz	Pin 55	B	Pin 26
CLK_IND	Pin 7	C	Pin 27
KP	Pin 8	D	Pin 28
DE0	Pin 10	E	Pin 29
DE1	Pin 11	F	Pin 30
DE2	Pin 12	G	Pin 31
DE3	Pin 13	H	Pin 32
K2	Pin 18		
K1	Pin 19	138sel0	Pin 33
K0	Pin 20	138sel1	Pin 36
KEY2	Pin 42	138sel2	Pin 37
KEY1	Pin 43		
KEY0	Pin 44		

7.7 LCD Interface Circuit

Nowadays, LCD displayer plays an important role in our daily life. From variety domestic electrics such as alarm clock and microwave, to office suppliers such as fax machine, printer, copy machine, calculator and notebook. Therefore, it applies to many mechanics and is very popular. In this section, we will introduce the applications and interface circuit of a design textual LCD module.

7.7.1 Descriptions of LCD Module

Currently, there are two types of mini LCD module, one is textual LCD module, and the other is Figure LCD module. First, let's get acquainting with textual LCD module, which applies to most of the office machine. Figure 7.31 illustrates the internal structure of textual LCD, which is composed by LCD board, LCD driving chip, and HD44780 control chip. The features are as follows:

1. Compatible with 4-bit or 8-bit CPU;
2. Data Display Ram(DD RAM) has 80 bytes , and can display 80 words;
3. Installed Character Generator ROM (CG RAM), and installed 160 5 ×7 dot matrix characters;
4. Installed Character Generator RAM (CG RAM), and allows to build eight 5 ×7 dot matrix characters;
5. CPU can read both the data of DD RAM and CG RAM;
6. HD44780 provides many functions of display control orders, such as clear displayer, cursor reset, on/off display, on/off cursor, flash display, etc.

Note: Please refer to other references for further information about display control instructions.

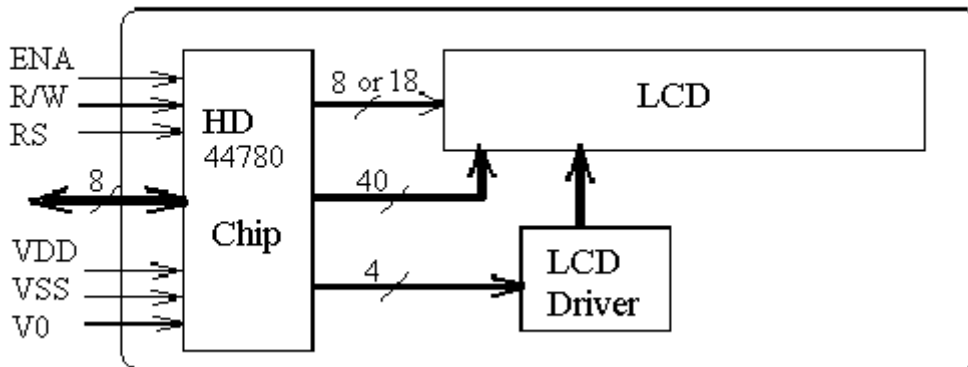


Figure 7.31 The functions blocks of Textual LCD

From Figure 7.31, we know that the connection of LCD interface circuit and textual LCD module is through control wires and data wires, 4 or 8 lines. The control lines are ENA, RS, and R/W that the descriptions are listed in Table 7.22.

Table 7.22 The combination of ENA, RS, and R/W

ENA	RS	R/W	Descriptions
1	0	0	Writing orders to IR Register of HD44780
1	0	1	Reading Busy Flag (DB7) and Dress Counter (DB6~DB0)
1	1	0	Writing data to DR Register (CG RAM or DD RAM)
1	1	1	Reading data from DR Register (CG RAM or DD RAM)

Each time, after sending power, LCD module has to accomplish the initial phase of sending instructions to IR (Instruction Register), which the address is 00H. The processes of the initial phase are described as follows.

1.Setting Functions

Instructions Format:



ENA	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	1	DL	N	F	*	*

DL: setting data width, DL = 0 is 4-bit, DL = 1 is 8-bit.

N: display rows are one or two.

F: setting character types.

N	F	rows	Character types
0	0	1	5 ×7 dot
0	1	1	5 ×10 dot
1	*	2	5 ×7 dot

Therefore, “38H” means 8-bit, double line display, and 5 × 7 dot character.

2.Open displayer

Instructions Format:

ENA	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	0	1	D	C	B

D: D = 0, data not display

D = 1, data display

C: C = 0, cursor not display

C = 1, cursor display

B: B = 0, not flush

B = 1, flush

Therefore, “0EH” means the displayer is open and the cursor display.



3. Enter to Mode setup

Instructions Format

ENA	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	0	0	1	I/D	S

I/D: I/D = 0, when reading/writing data to DD RAM, address counter should minus one unit.

I/D = 1, when reading/ writing data to DD RAM, address counter should plus one unit.

S: S = 1, when writing data to DD RAM, displayed data move left for a column.

S = 0, when writing data to DD RAM, the displayed data should not move left for a column.

Therefore, when “06H” means reading/writing data to DD RAM, the address counter would add one unit; and when writing data to DD RAM, the displayed data would not move left for a column.

4. Clear Displayer

Instruction Format:

ENA	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	0	0	0	0	1

Complete LCD Interface Circuit by “Data Path Circuit” and “Control Circuit.” The control circuit is in charge of generating necessary signals of ENA, RS, and R/W and creating states which start from “000” (the state of sending instruction 38H), and “001” (the state of sending instruction 0EH), then “010” (the state of

sending instruction 06H), and “011” (the state of sending instruction 01H), and the following is “100” which sends out the digits. Except for the clock input, the “Control Circuit” has “START” signals for activating clear and sends out data. According to the states sent by the “Control Circuit,” the “Data Path Circuit” sends out the relative data, including displayed digits and instructions, to LCD module.

7.7.2 Data Path Circuit of LCD Interface

“Data Path Circuit” is in charge of supplying the data for LCD modules, which include the orders of the initial phase, such as 38H, 0EH, 06H and 01H, and the input of the displayed data.

Step 1: Use the Graphic editor of MAX+PLUS II to create the following circuit entry, from Figure 7.23a to Figure 7.32h, and create the internal circuit symbol.

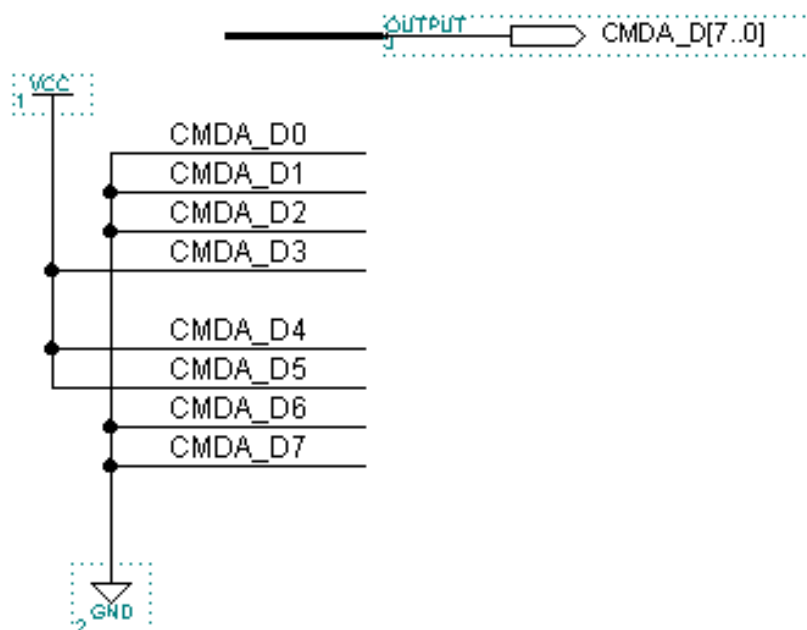


Figure 7.32a Data Circuit Module “38H” (File: 38H.gdf)

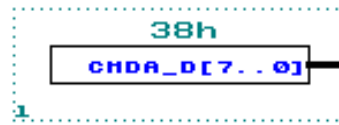


Figure 7.32b The internal circuit symbol of data circuit module “38H”

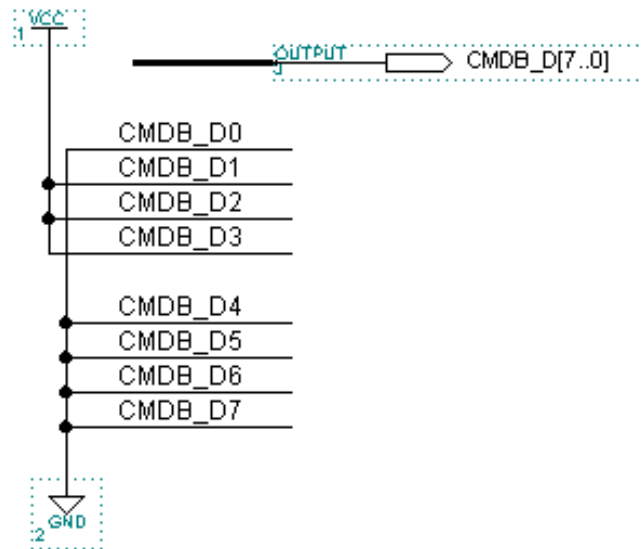


Figure 7.32c Data circuit module “0EH” (File: 0EH.gdf)

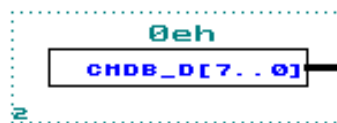


Figure 7.32d The internal circuit symbol of data circuit module “0EH”

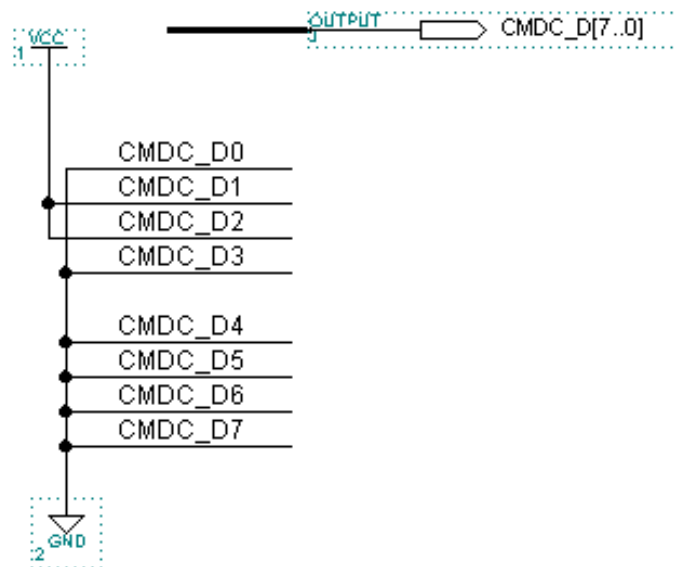


Figure 7.32e Data circuit module “06H” (File: 06H.gdf)

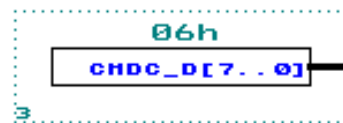


Figure 7.32f the internal circuit symbol of data circuit module “06H”

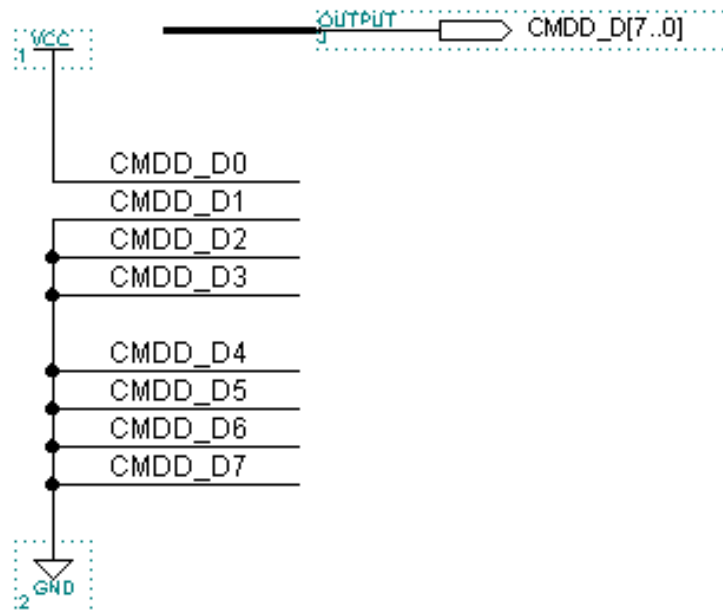


Figure 7.32g Data Circuit Module “01H” (File: 01H. gdf)

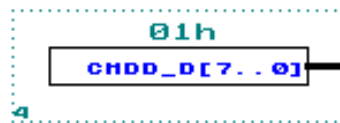


Figure 7.32h The internal circuit symbol of data circuit module “01H”

Step 2: Please use the Graphic editor of MAX+PLUS II to create the following circuit entry of “8-bit 5 × 1 data multiplexer”, Figure 7.33a, and the internal circuit symbol.

Step 3: Complete functional simulation of “8-bit 5 x 1 data multiplexing circuit” by using MAX+PLUS II (Figure 7.33b) and test whether the functions meet the circuit specification. Go on to generate the internal circuit symbol, Figure 7.33c.

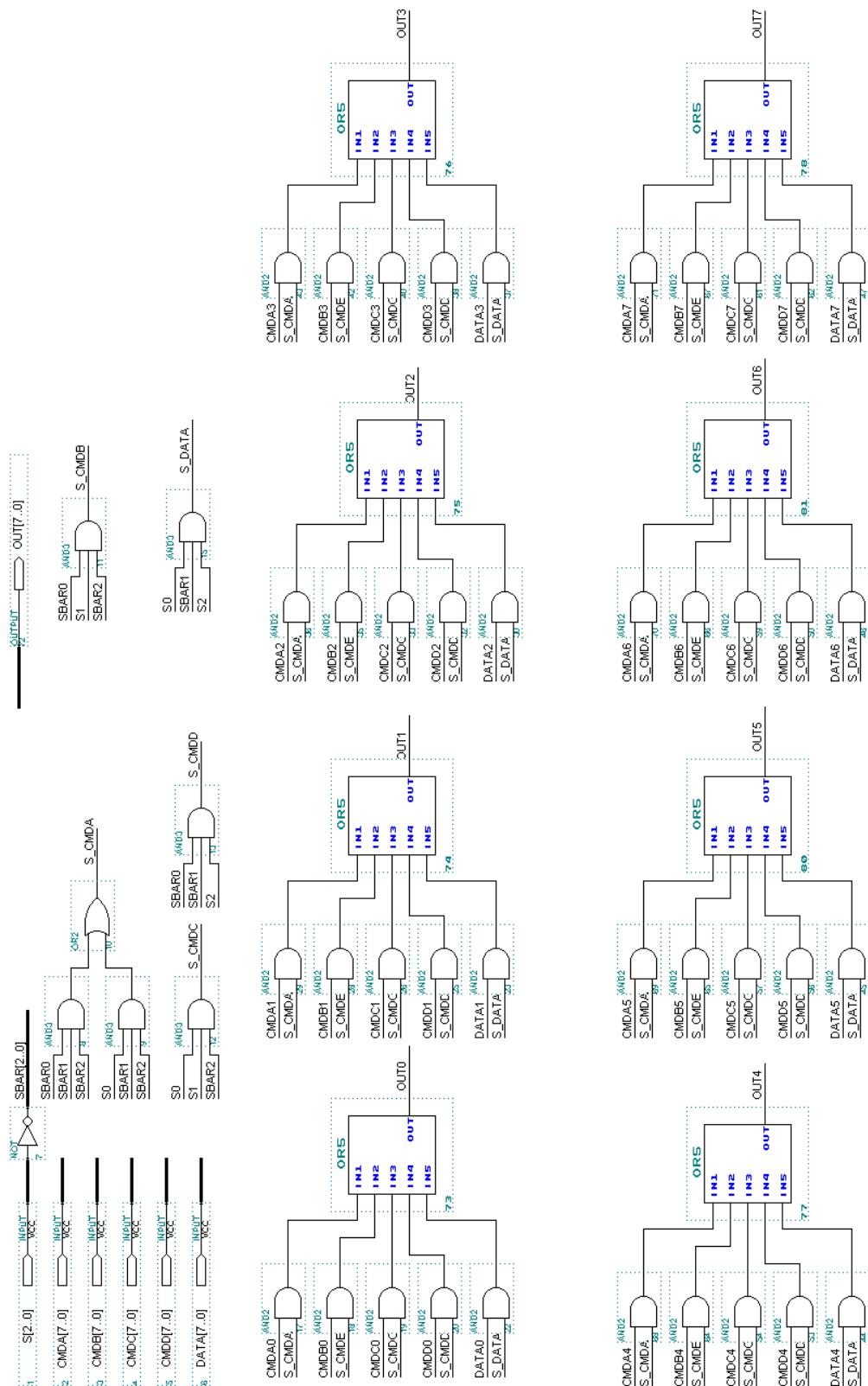


Figure 7.33a 8-bit 5 × 1 data multiplexer circuit (File: LCD_MUX.gdf)

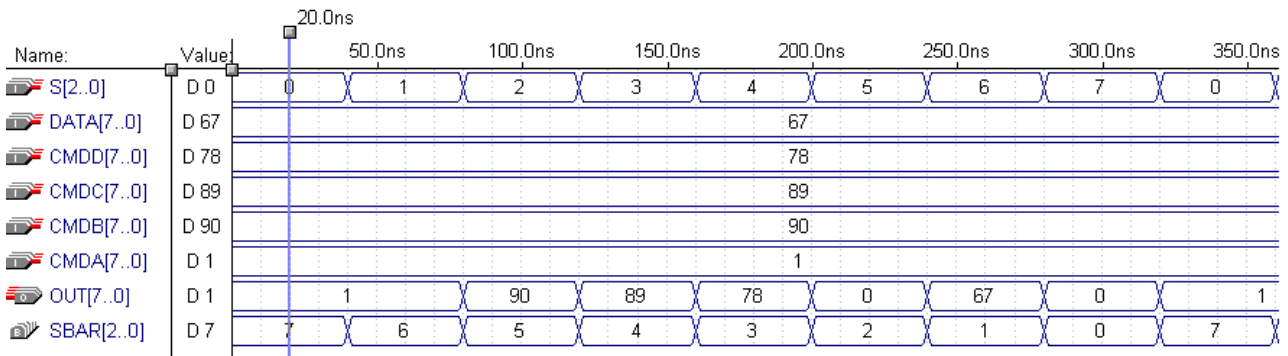


Figure 7.33b Simulation result of 8-bit 5 × 1 data multiplexer circuits
(File: LCD_MUX.scf)

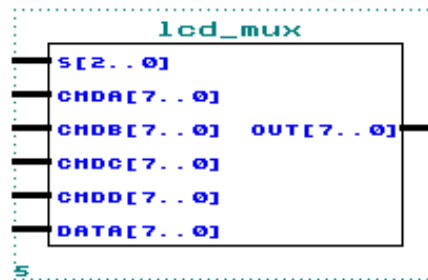


Figure 7.33c Internal circuit symbol of 8-bit 5 × 1 data multiplexer circuit

Step 4 : Use the Graphic editor of MAX+PLUS II to complete the circuit entry of data path circuit (Figure 7.34a).

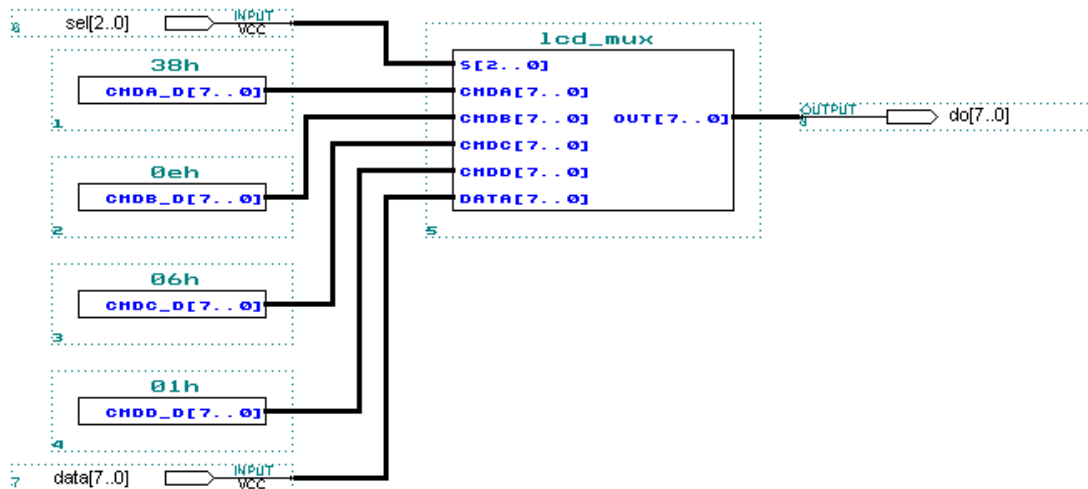


Figure 7.34a Data path circuit (File: DATAPATH.gdf)

Step 5: Complete functional simulation of “Data Path” by using MAX+PLUS II and test whether the functions meet the circuit specification. Figure 7.34b shows the simulation results of data path circuit. Then, go on to generate the internal circuit symbol, Figure 7.34c.

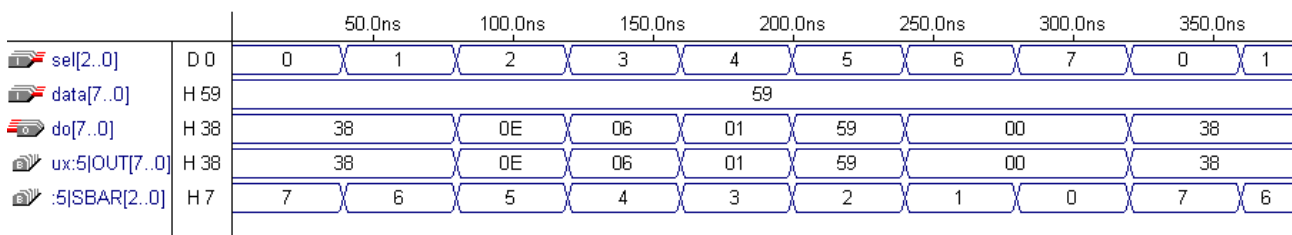


Figure 7.34b Simulation result of data path circuit (File: DATAPATH.scf)

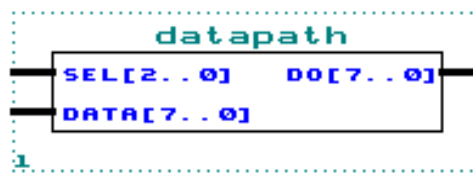


Figure 7.34c Internal circuit symbol of data path circuit

7.7.3 Control Circuit of LCD Interface Circuit

“Control Circuit” in charge of generating necessary signals of EN, RS, and R/W.

Step 1: Use the Graphic editor of MAX+PLUS II to complete the following circuit, Figure 7.35a and Figure 7.33b, and generate the internal circuit symbols.

Step 2: Complete functional simulation by using MAX+PLUS II and test whether the functions meet the circuit specifications. Figure 7.35b shows the

simulation result of CTRL.gdf circuit. Go on to generate the internal circuit symbol if meets the specifications.

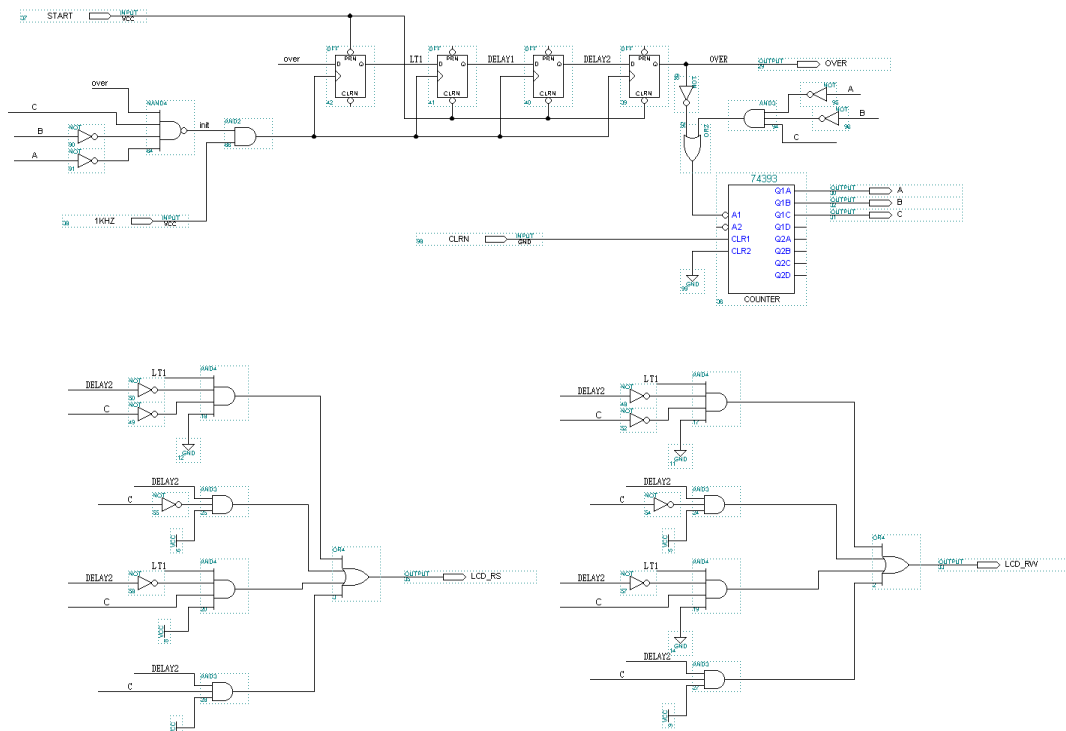


Figure 7.35a Control circuit module (File: CTRL.gdf)

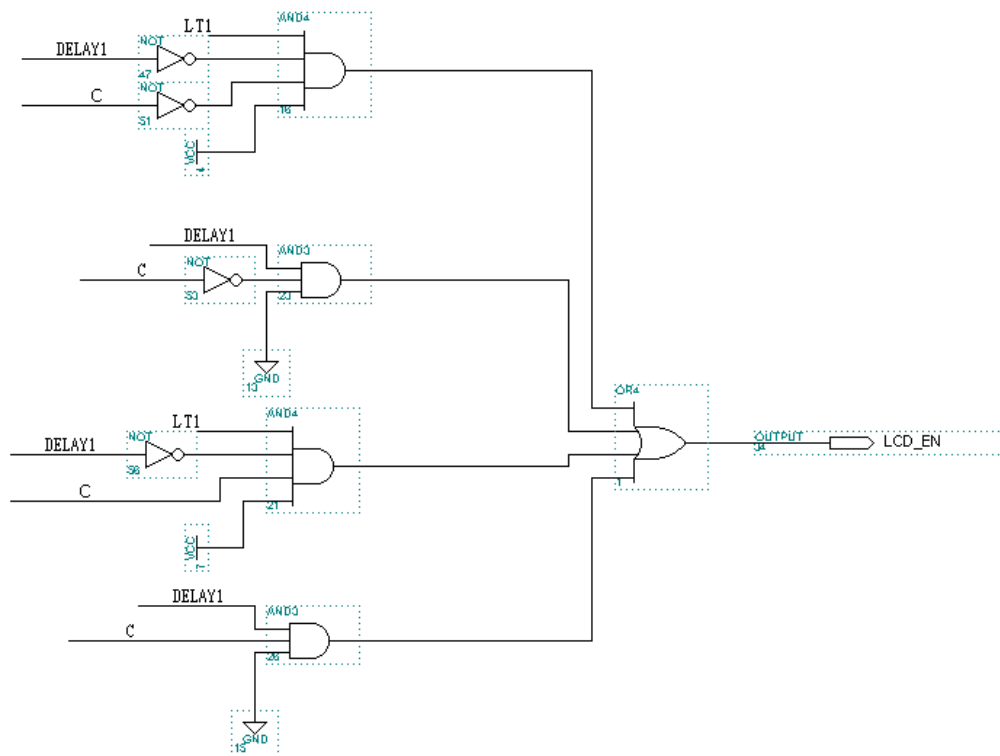


Figure 7.35a Control circuit module (continue)

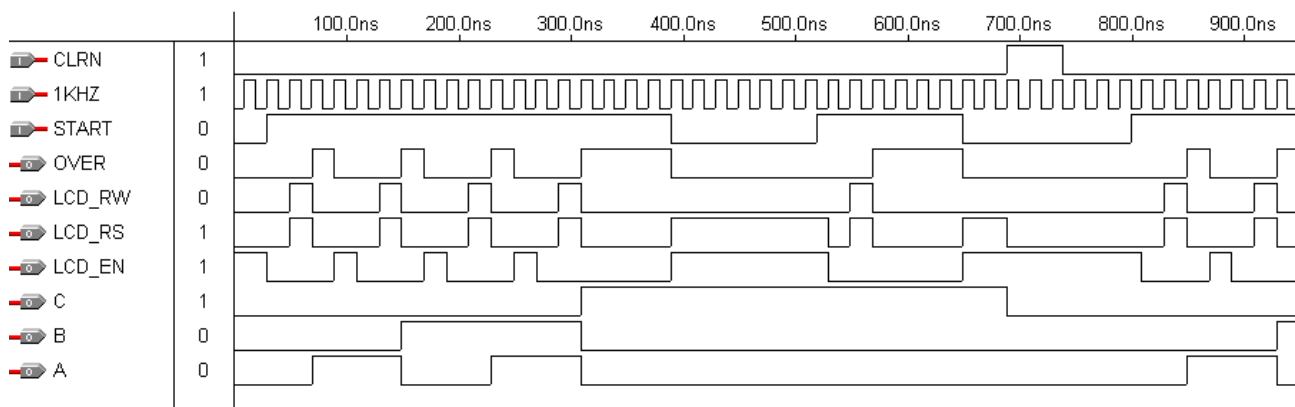


Figure 7.35b Simulation result of control circuit (File: CTRL.scf)

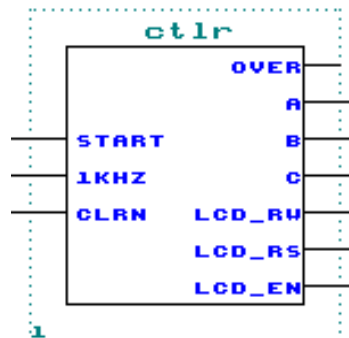


Figure 7.35c The internal circuit symbol of control circuit module

7.7.4 Complete LCD Interface Circuit

Step 1: Please use the Graphic editor of MAX+PLUS II to create the following circuit, Figure 7.36a. Please use the devices of DIV10.GDF, CTRL.GDF, DATAPATH, DISBOUNCE listed in this chapter and DIV1000 listed in Chapter 4 to complete the LCD interface.

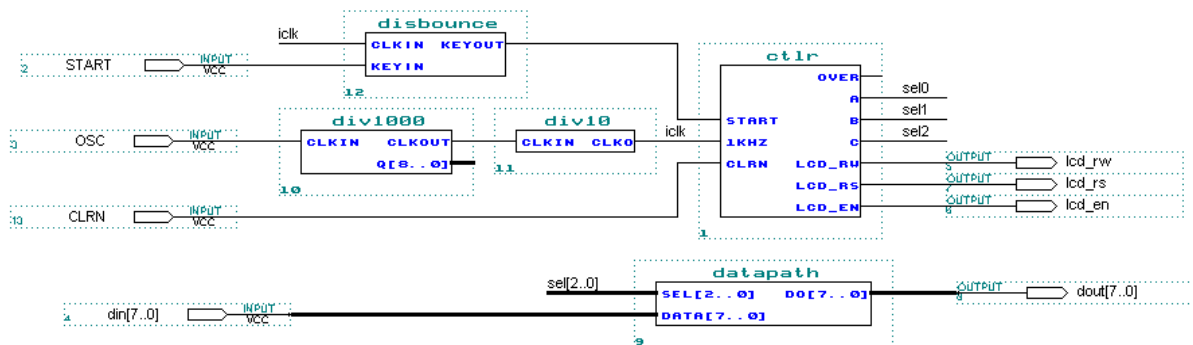


Figure 7.36a Complete LCD interface circuit (File: LCD_INTF.gdf)

Step 2: Complete functional simulation by using MAX+PLUS II and test whether the functions meet the circuit specifications.

~Skip this functional simulation~

Table 7.23 Pin setup of EPF10K10TC144-4 chip

Signal	EPF10K10TC144-4 chip pin	Signal	EPF10K10TC144-4 chip pin
10 MHz	Pin 55	LCD_EN	Pin 130
START	Pin 124	LCD_RS	Pin 122
CLRN	Pin 126	LCD_RW	Pin 128
DIN0	Pin 63	DOUT	Pin 131
DIN1	Pin 62	DOUT	Pin132
DIN2	Pin 60	DOUT	Pin133
DIN3	Pin 59	DOUT	Pin135
DIN4	Pin 51	DOUT	Pin136
DIN5	Pin 49	DOUT	Pin137
DIN6	Pin 48	DOUT	Pin138
DIN7	Pin 47	DOUT	Pin140

Step 3: If the circuits allow download testing, download (or programming) to test the circuit after selecting the download (or programming) device and floorplan program. Please adapt the techniques of floorplan programming instructed in Section 4.6 to choose an EPF10K10TC144-4 chip and use the pin assignment listed in Table 7.23. After setting up LP-2900 Lab Platform, download the keyboard scan and display circuit to EPF10K10TC144-4 chip, and try to:

1. After pressing PS4 (CLRN) at the middle bottom of LP-2900, try to press PS3 (START). The LCD should be cleared.
2. Please press PS3 after inserting the digits of ASCII code, such as 32H



of “2”, from SW1 (DIN7)~SW8 (DIN0). The digit “2” will display in front of the cursor on the LCD. Please try to insert other digits.

This example is for your own reference on designing LCD Module Interface Circuit. Please modify the circuit to satisfy your need.



7.8 Evaluations

Please do the following evaluations according to the questions listed below:

- ❑ Do you know the principles of each frequency generation?
- ❑ Do you know the display scan principles of a 7-segment displayer?
- ❑ Do you know how to detect the keys of a 4 × 3 keyboard?
- ❑ Do you know how to test a 8 × 8 bi-color dot matrix LED?
- ❑ Can you sense whether the key bumps? If yes, how to eliminate it?
- ❑ Can you design, simulate, and verify the display scan interface circuit of a 7-segment displayer?
- ❑ Can you design, simulate, and verify the clock interface?

CHAPTER 8

Connecting with Analog Circuit



LEAP

In Chapter one, it notes that all the physical measuring is analog signal. However, the digital signal processor is more concise, faster, programmable, adjustable and less effective by device features than analog signal. Therefore, A/D converter and D/A converter are acquired in order to converse analog signal into digital signal and, conversely, digital signal to analog signal. There are many kinds of A/D converter and D/A converter in the market. In this chapter, the most frequently used one is introduced.

8.1 A/D Converter—ADC0804

Produced by Harris Semi-conductor Co., ADC0804 is an A/D converter. The function of A/D converter is to quantify the analog signal to digital signal after sampling. ADC0804 is a CMOS successive approximation style A/D converter, which has modified potentiometric ladder and three status outputs and is compatible with the control bus of 8080A. Without any interface circuit, ADC0804 converter can directly connect with microprocessor.

The input of analog differential voltage has a great common-mode-rejection and allows analog zero-voltage offsetting. The input adjustment of referential voltage allows any little voltage span to be coded in a complete 8-bit figure.

Main Features of ADC0804 :

1. Compatible with the bus of microprocessor, 80c48 and 80c80/85, and can be directly connected without any interface circuit.
2. Time cost for transmission is less than 100us.
3. Ease interface with most of microprocessors.
4. Can be operated “individually”

5. Differential analog voltage input.
6. Work under bandgap referential voltage.
7. Provide TTL compatible input/output signal.
8. The chip contains clock generator circuit.
9. 0V to 5V analog input voltage (requiring only a single +5V for operation).
10. No zero-adjustment required.

Figure 8.1 is the pin layout of a 20-pin ADC0804. Once the input of /WR generates varies from high to low, SAR register in ADC0804 will be latched, reset shift register, and /INTR will output high voltage. As long as /CS and /WR input stay in low voltage, ADC0804 will keep in the reset status. If one of these two input signals changes from low to high as the transmission beginning, /INTR will output low-level voltage which is for notifying microprocessor that the transmission has completed. Low-level input voltage of /CS and /RD can reset the output signal of /INTR.

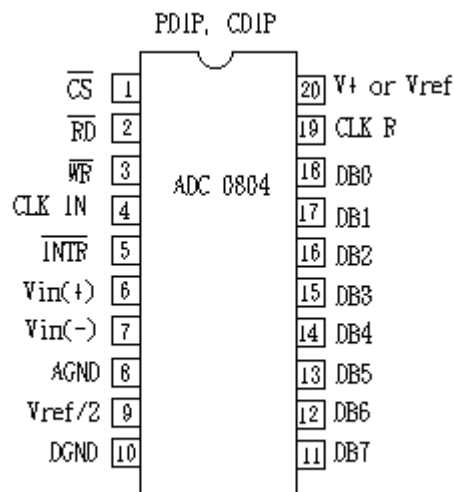


Figure 8.1 Pin configuration of a 20-pin ADC0804

8.2 D/A Converter—AD7528

AD7528 is a CMOS double 8-bit buffer multiply digital/analog converter produced by Analog Devices Co. Figure 8.2 is the Block Diagram of AD7528 which has two sets of separated latch providing a perfect interface of microprocessor. Through the shared 8-bit TTL/COMS compatible input port, data can be sent to one of the latch. /DAC A/DAC B is the control line for controlling the latch that the data sent to and transform the data. Writing data to AD7528 is as simple as that to RAM. Further, AD7528 is compatible with microprocessors—6800, 8080, 8085, and Z80.

Figure 8.3 is the pin configurations of a 20-pin AD7528 which has 2 kinds of package, DIP and SOIC. /CS is the control line of chip selection; /CS = “0” reflects AD7528 is being selected. Table 8.1 is a signal combination functional description of the signal lines- /CS, /WR, and /DAC A/DAC B.

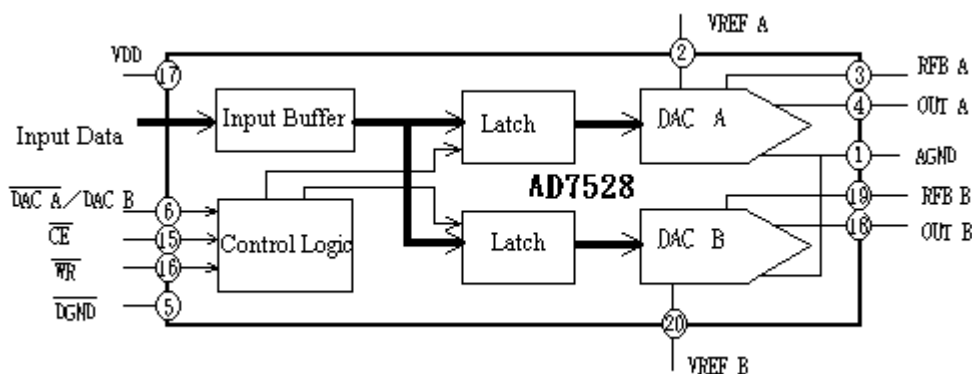


Figure 8.2 Block diagram of AD7528

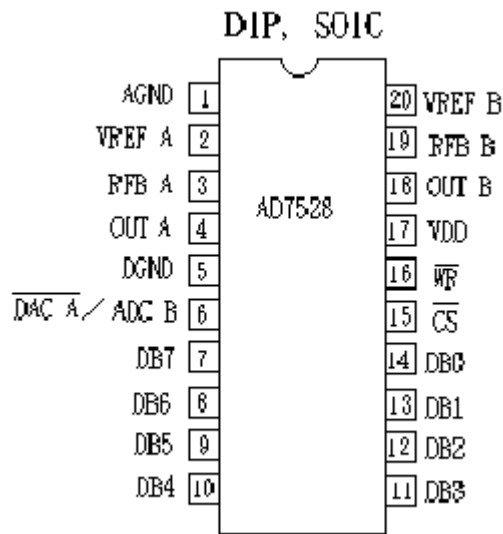


Figure 8.3 Pin configuration of AD7528

Table 8.1 Functional descriptions of /CS, /WR and /DAC A/DAC B signal combination

/CS	/WR	DAC A/DAC B	Description
1	X	X	Output the original transmission value
X	1	X	Output the original transmission value
0	0	0	Operate transmission of D/A in set A
0	0	1	Operate transmission of D/A in set B

8.3 Single Chip -- 8051

MCS-51 is the code of a single chip family. Table 8.2 is the list of MCS-51 family and the main components. Program memory capacity means the program memory of a single chip whereas data memory capacity means the data memory of a single chip.



Table 8.2 MCS-51 family and the main components

Name	Program Memory Capacity	Data Memory Capacity	16-bit Timer	Serial I/O Port	Serial I/O Port	Circuit Type
8051	4K (ROM)	128 bytes	2	4	1	
8751	4K (EPROM)	128 bytes	2	4	1	HMOS
8031	unavailable	128 bytes	2	4	1	HMOS
8052	8K (ROM)	256 bytes	3	4	1	HMOS
8752	8K (EPROM)	256 bytes	3	4	1	HMOS
8032	unavailable	256 bytes	3	4	1	HMOS
80c51	4K (ROM)	128 bytes	2	4	1	CHMOS
87c51	4K (EPROM)	128 bytes	2	4	1	CHMOS
80c31	Unavailable	128 bytes	2	4	1	CHMOS

The following is the general descriptions of 8x51:

1. An 8-bit single chip for controlling
2. Strengthen logic operation instruction of one bit
3. The chip has a 128-bit RAM,
4. The chip contains 2 Timer/Counter,
5. The chip contains 1 Full-duplex UART (Universal Asynchronous Receiver),
6. The ship contains 5 interrupt resources with 2 level priority architecture,
7. The chip has a clock oscillator circuit,
8. The chip can expand externally to 64K program memory,
9. The chip can expand externally to 64K data memory.



❖ Pin configuration of 8051

The pin configuration of 8051 is displayed as Figure 8.4. The descriptions of each connection are listed in Table 8.3.

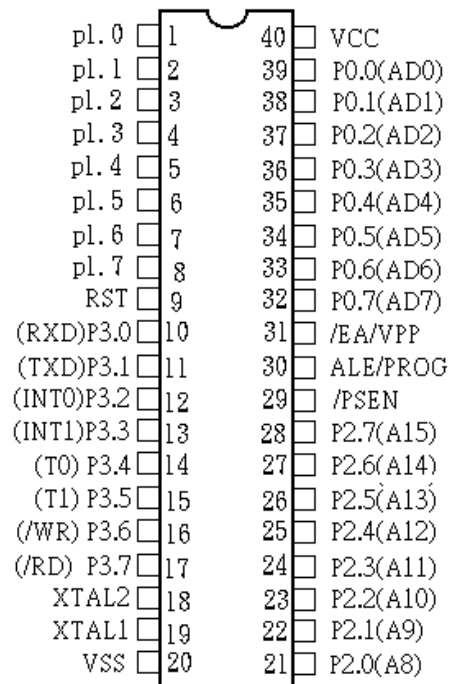


Figure 8.4 Pin Configuration of 8051

Table 8.3 Pin descriptions of 8051

Pin Name	Pin	Descriptions
Vcc	40	Positive terminal of power. Voltage: 5V \pm 10% .
Vss	20	Negative terminal of power.
RST	9	Reset signal input of CPU. Staying in low voltage normally but raise to high voltage as resetting and will keep at least 2 machine cycles while the single chip will progress the diverse jobs of resetting the system. Afterward, the address starts from 0000h when operating.



ALE/PROG	30	This pin has two functions: (1) ALE is the initial of address latch enable. When 8051 is reading the external program or when the data sends out the addressing signals, the signals would be accompanied sending out for the external circuit to latch the low-bit signals of the addressing line; (2) PROG is the special functional input pin when 8751 is programming.
/PSEN	29	/PSEN is the initial of program strobe enable. When 8051 is reading the external program memory, it will at the same time, sending out the signal for reading the program code.
/EA/VPP	31	The pin has double functions: (1) /EA is the initial of external access. Activate with voltage low. Implementing reading the external programs as /EA = "0" while reading the internal programs as /EA = "1"; (2) Vpp is the programming voltage input pin when 8751 is programming.
XTAL1	19	Input terminal of systematic oscillator crystal.
XTAL2	20	Output terminal of systematic oscillator crystal.
PORT0	39~32	Port0 is part of the open drain construction; therefore, it requires an external connection to pull-up resistor. There are three functions of this set: (1) bit-addressable bi-directional port, (2) low byte of output address, (3) bi-direction port.
PORT1	1~8	Port1 is a bit-addressable bi-direction port with internal pull-up resistor.
PORT2	21~28	There are two functions of this set (1) bit-addressable bi-direction port with internal pull-up resistor, (2) high byte of output address.
PORT3	10~17	This set has 2 functions: (1) bit-addressable bi-direction port with internal pull-up resistor; (2) The functions of each pin are as follows: RXD (10): input port of serial communication; TXD (11): output port of serial communication; INT0 (12): input of external interrupt 0, INT1 (13): input of external interrupt1; T0 (14): input of external Timer0; T1 (15): input of external Timer1; /WR (16): writing signal; /RD (17): reading signal.

Figure 8.5 illustrates the extend circuits of 8051 single chip is not complicated. Usually, it only requires reset circuits, oscillator and some I/O devices.

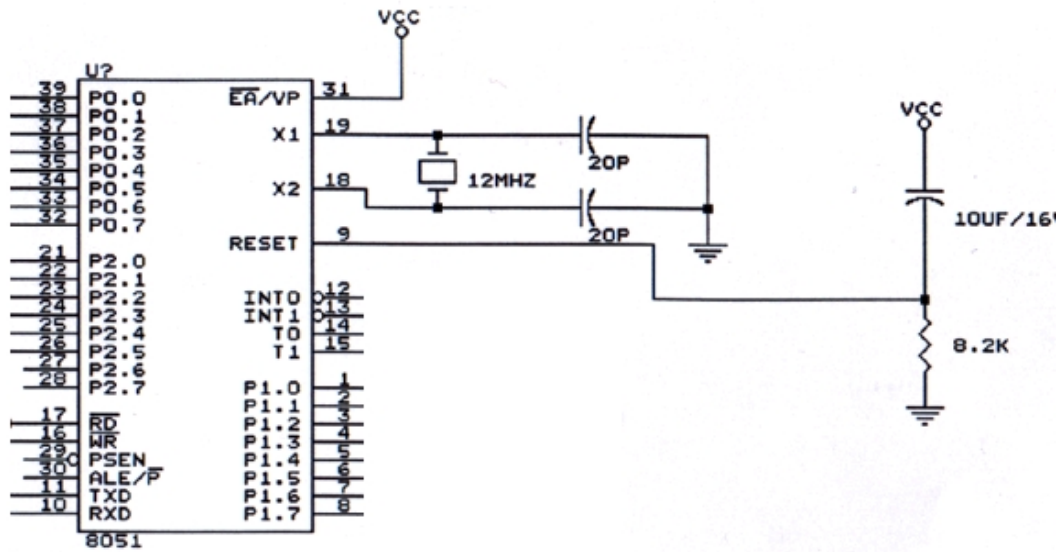


Figure 8.5 8x51 microprocessor and its reset circuit and oscillator circuit

❖ 8051 addressing modes

Please note that there are five addressing modes of 8051:

1. Direct addressing,
2. Indirect addressing,
3. Register addressing,
4. Immediate addressing,
5. Indexed addressing.

❖ 8051 Program Development Flow

Generally, the flow of 8051 program development is illustrated as 8.6:

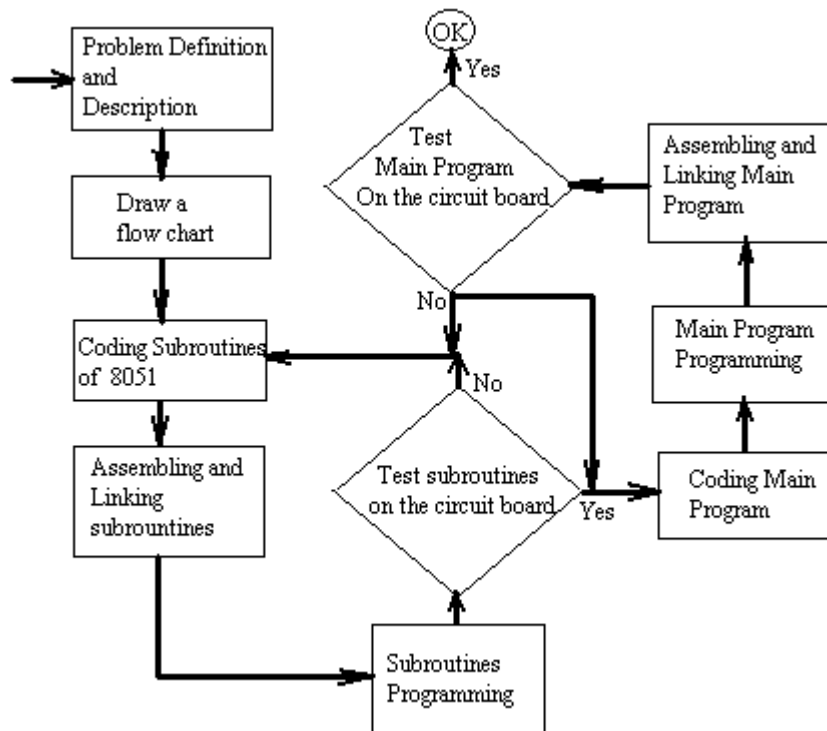


Figure 8.6 program development flow of a 8x51 processor

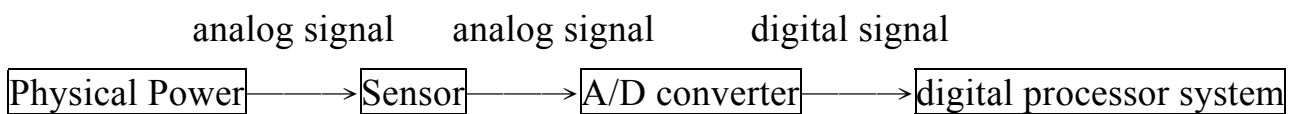
The flow starts with problem definition and explanation, then to draw and develop a flow chart. According to the flow chart, the next step is to editing, encoding, linking, programming and testing the programs of the sub-program. If problem occurs, then modify the sub-program and re-encoding, linking, programming, and testing until the problem are eliminated. After the sub-program or go back to edit the main-program to modify the main program, and re-encoding, linking, programming and testing until the problems are eliminated. After the sub-program has been developed, please edit, encode, link, program and test the main program. Again, if problem occurs, go back to edit the sub-program to modify the sub-program or go



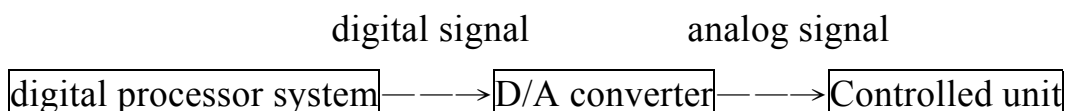
back to edit the main-program to modify the main program, and re-encoding, linking, programming and testing until the problems are eliminated.

8.4 Design Example—Connecting with ADC0804, AD7528, and 8951

We know that all the natural physical power is analog power from the instruction of Chapter 1, and the power is measurable, detectable, and controllable. To implement the advantages of digital system, we have to transform the analog signal into digital signal and process with digital signal. The procedures of transformation are as follows:



The sensor can transform the natural physical power into the signals of electric—voltage signal or electric current signal—that the signal stays in analog signal. The voltage signal becomes digital signal through A/D converter under some circumstances. The digital signal, after processed, can turn to analog signal to control some mechanisms, such as current control valve, cooling fan, heater). The conversion is as follows



For interpreting the role of ADC0804, AD7528 and 8951, Figure 8.7 illustrates the implication of ADC0804, AD7528, and 8951. The 8-bit counter is for replacing the

physical power and sensor while 8951 is the digital processor system which take the responsibilities of controlling D/A converter, accepting A/D notification to restore the transferred data, digits transforms, scanning the 7-segment displayer and sending out the displayed digits to the 7-segment displayer. We will implement the circuits in the dotted frame of Figure 8.7 to CPLD chip.

In LP-2900 Logic Circuit Design Lab Platform, the connecting of 8951, ADC0804 and AD7528 is shown as Figure 8.8. The circuit layout of Figure 8.9 is the 7-segment displayer of LP-2900 Logic Circuit Design Lab Platform. The connection and signal definitions of 8951 and EPF10K10TC144-4 is shown as Table 8.4 and Table 8.5 shows that of ADC0804 and EPF10K10TC144-4 and Table 8.6 listed that of AD7528 and EPF10K10TC144-4.

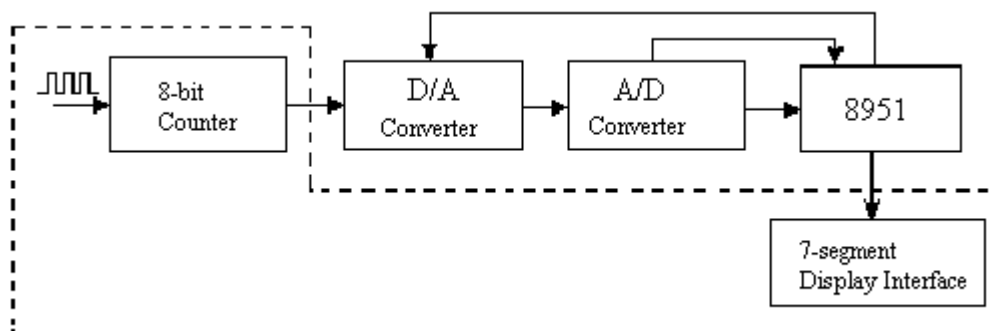
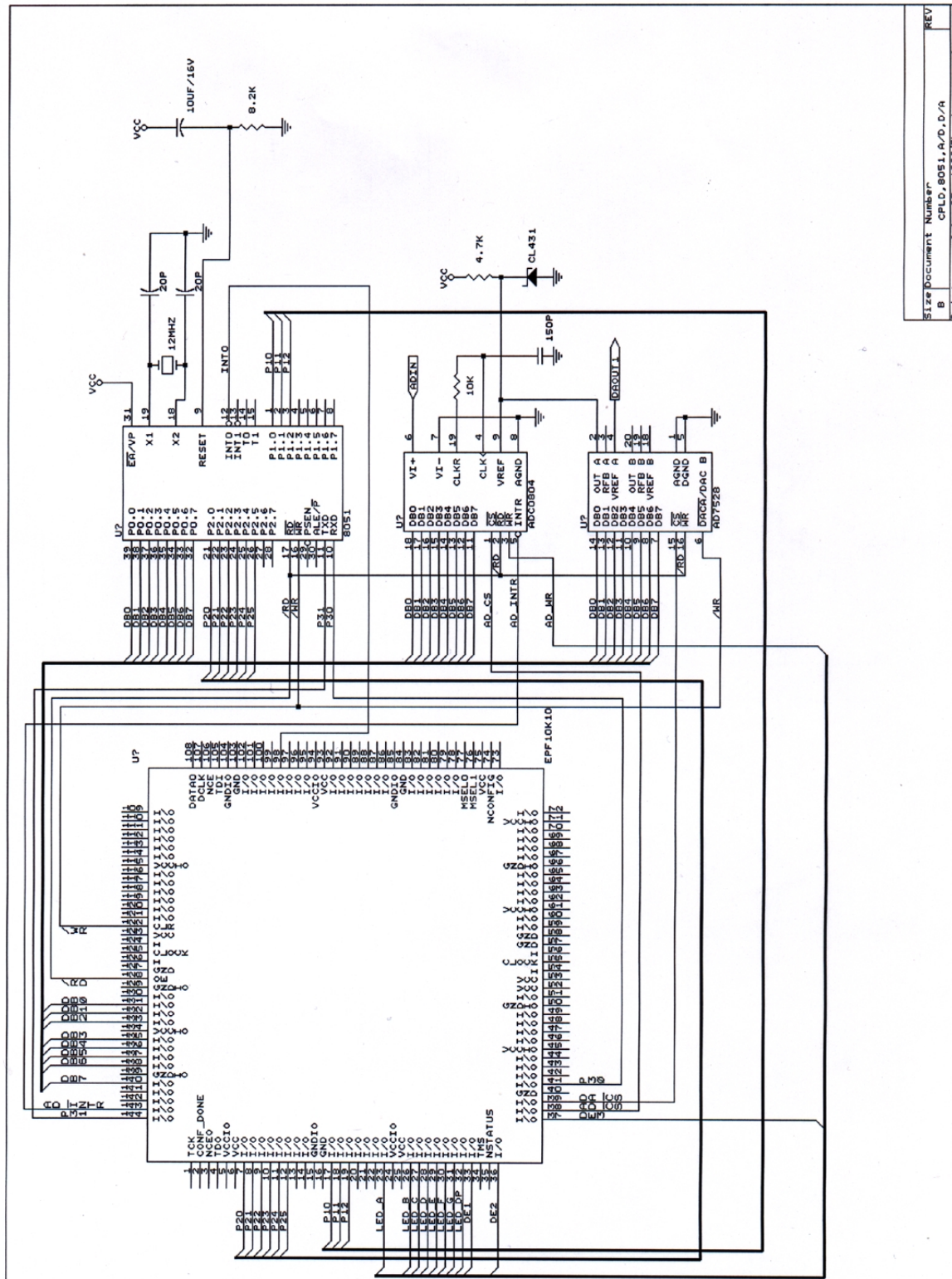


Figure 8.7 Connecting application of ADC0804, AD7528, and 8951



Size	Document Number	REV
B	CPLD.8051.A/D/A	
Date:	JULY 23, 1999	Sheet of

Figure 8.8 Connecting application of 8951 and ACD0804, AD7528 on LP-2900

Lab Platform

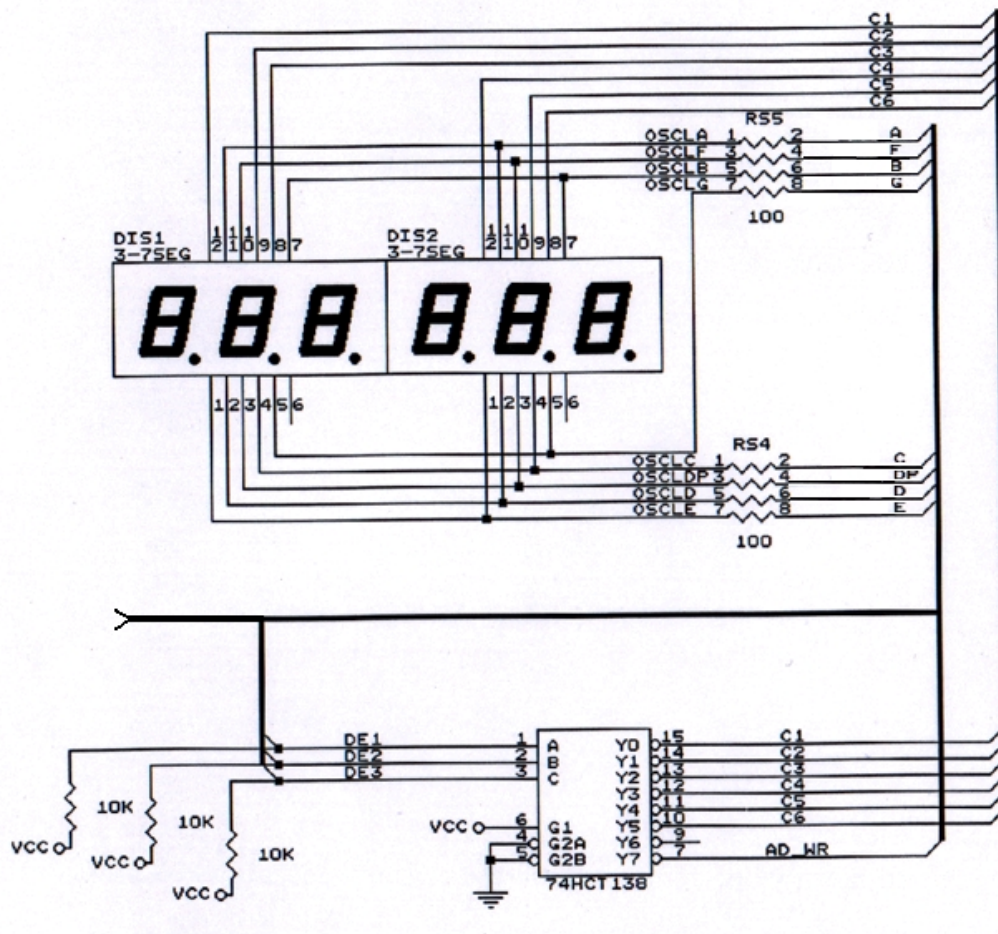


Figure 8.9 Circuit Diagram of a 7-segment displayer on LP-2900 Lab Platform

Table 8.4 Connections of 8951 and EPF10K10TC144-4 chips and signal definition

Code	P0.0	P0.1	P0.2	P0.3	P0.4	P0.5	P0.6	P0.7
Signal	AD0	AD1	AD2	AD3	AD4	AD5	AD6	AD7
Pin	Pin 131	Pin 132	Pin 133	Pin 135	Pin 136	Pin 137	Pin 138	Pin 140

P0.0~P0.7 also connect D0~D7 of LCD

Code	P1.0	P1.1	P1.2	P1.3	P1.4	P1.5	P1.6	P1.7
Signal	DE1	DE2	DE3	Not used in the current example				
Pin	Pin 17	Pin 18	Pin 19	Pin 20	Pin 21	Pin 22	Pin 141	Pin 142

P1.0~P1.7 also connect L21~L26 and RG_EN and BAR_EN



Code	P2.0	P2.1	P2.2	P2.3	P2.4	P2.5	P2.6	P2.7
Signal	D0	D1	D2	D3	Dp	ENA		
Pin	Pin 7	Pin 8	Pin 9	Pin 10	Pin 11	Pin 12	Pin 13	Pin 14

P2.0~P2.7 also connect L1~L8

Code	P3.0	P3.1	P3.2	P3.3	P3.4	P3.5	P3.6	P3.7
Signal	AD_CS	DA_CS	INT0				/WR	/RD
Pin	Pin 41	Pin 144	Pin 98	Pin 99	Pin 100	Pin 101	Pin 122	Pin 128

P3.2~P3.5 also connect CR1~CR4 of 8 × 8 dot matrix

P3.6~P3.7 also RS and RW of LCD

Table 8.5 Connections and signal definitions of ADC0804 and
EPF10K10TC144-4

A/D→ADC0804

Code	DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7
Signal	AD0	AD1	AD2	AD3	AD4	AD5	AD6	AD7
Pin	Pin 131	Pin 132	Pin 133	Pin 135	Pin 136	Pin 137	Pin 138	Pin 140

DB0~DB7 also connect D0~D7 of LCD

Code	/CS	/RD	AD_INTR
Signal	AD_CS	/RD	AD_INTR
Pin	Pin 38	Pin 128	Pin 143

AD_WR is Y6 of 3 to 8 decoder output; /RD also connects RW of LCD.



Table 8.6 Connections and signal definition of AD7528 and EPF10K10TC144-4

D/A→AD7528

Code	DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7
Signal	DD0	DD1	DD2	DD3	DD4	DD5	DD6	DD7
Pin	Pin 131	Pin 132	Pin 133	Pin 135	Pin 136	Pin 137	Pin 138	Pin 140

DB0~DB7 also connect D0~D7of LCD

Code	/CS	/WR	/DACA
Signal	DA_CS	/WR	/DACA
Pin	Pin 39	Pin 128	Pin 122

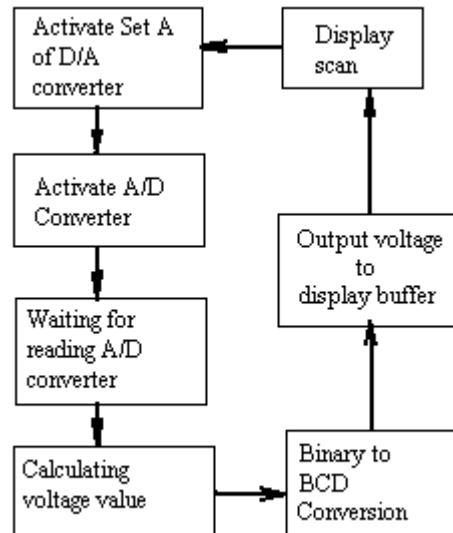
/WR also connect RW of LCD ; /DACA also connect RS of LCD

In order to implement the connection applications of Figure 8.7 and ADC0804, AD7528 as well as 8951 on LP-2900 Logic Circuit Design Lab Platform, we have to

1. Complete 8951 programs writing and programming,
2. Complete the parts of dotted line of Figure 8.7 in CPLD chip.

Step 1: Please complete the following 8951 program assembling and programming.

We know the functions of 8951 are:



According to the procedure displayed above, we can, therefore, write AD2 program of 8951 in Figure 8.10. When assembled the file, AD2.HEX, of Figure 8.11 is acquired. Please program this file onto 8951.

```
.symbols
.linklist
.debug asm

DisplayBuffer    equ        30h        ;4 bytes
ScanCounter      equ        34h
VoltLow          equ        35h
VoltHigh         equ        36h
;-----
                org        0h
                jmp        Reset
Reset:
                mov        sp,#70h        ;Setup stack pointer
                mov        ScanCounter,#0
                mov        p1,#06h        ;Initial scan value
```



```
        mov     p2,#20h           ;Setup blank value
        mov     p3,#ffh           ;AD_CS-->'H' , DA_CS-->'H'
                                   ;/RD-->'H', /WR-->
;-----
ADLoop:
        Push    p1
        push    p2
        anl     p3,#fdh           ;DA_CS-->'L'
        anl     p3,#3fh           ;DA_WR-->'L',DACA-->'L'
        orl     p3,#c0h           ;DA_WR-->'H',DACA-->'H'
        orl     p3,#02h           ;DA_CS-->'H'
; Above four instructions activate set A of D/A Converter
        anl     p3,#feh           ;AD_CS-->'L'
        mov     p2,#20H           ;Setup Blank value
        mov     p1,#07h           ;AD /WR-->'L'
        mov     p1,#06h           ;AD /WR-->'H'
; Above four instructions activate A/D Converter
        jb      p3.2,$            ; Wait for A/D conversion complete
        anl     p3,#7fh           ;/RD-->'L'
        mov     a,p0              ;Read A/D result
        orl     p3,#80h           ;/RD-->'H'
; Above three instructions read data from A/D conversion
        orl     p3,#01h           ;AD_CS-->'H'
; Above nine instructions activate A/D conversion and wait for reading its data
        pop
        pop
        call    Transfer           ; For value translation
        call    DisplayVoltIntoDisplayBuffer
; Call subprogram of output voltage value
        push
        push    psw
        setb    rs0                ;select RB1
        clr     rs1
        call    ScanDisplay        ;Call Subroutine of scan display
```



```
        pop        psw
        pop        a
        jmp        ADLoop
;-----
; Subroutine of Scan Display
;-----
ScanDisplay:
        mov        r0,#ScanCounter
        inc        @r0
        cjne       @r0,#4,NotOver
        mov        @r0,#0
NotOver:
        cjne       @r0,#0,ScanDisplay2
        mov        p2,30h          ;Output contain of 30h to p2
        mov        p1,#05h        ;(DE1,DE2,DE3)—> “101”
        ret
ScanDisplay2:
        cjne       @r0,#1,ScanDisplay3
        mov        p2,31h
        mov        p1,#04h        ;(DE1,DE2,DE3)—> “011”
        ret
ScanDisplay3:
        cjne       @r0,#2,ScanDisplay4
        mov        p2,32h
        orl        p2,#10h
        mov        p1,#03h        ;(DE1,DE2,DE3)—> “010”
        ret
ScanDisplay4:
        mov        p2,33h
        mov        p1,#02h        ;(DE1,DE2,DE3)—> “001”
        ret
;
;-----
; Move voltage value, which in VoltHigh and VoltLow, to DisplayBuffer for display
DisplayVoltIntoDisplayBuffer:
        mov        r1,#DisplayBuffer
```



```
Loop:      mov     a,VoltLow
           mov     b,#10h
           div     ab
           mov     @r1,b
           inc     r1
           mov     @r1,a
           inc     r1
           mov     a,VoltHigh
           mov     b,#10h
           div     ab
           mov     @r1,b
           inc     r1
           mov     @r1,a
           ret
```

;

;

; Subprogram of value translation

; Input: a registr

; Output: VoltHigh and VoltLow

; Note : The read in value times 0.02 is voltage value

;

Transfer:

```
           mov     b,#2
           mul     ab
           mov     VoltHigh,b
           mov     VoltLow,a
           call    Bin2Bcd
           mov     VoltHigh,r4
           mov     VoltLow,r3
           ret
```

;

;

; Subprogram of binary to BCD

; Input: VoltHigh and VoltLow

; Output : r3 and r4 registers

;

;

; Call subprogram of binary to BCD

Bin2Bcd:

```
    mov     r5,#16           ;16 bits
    clr     a
    mov     r3,a
    mov     r4,a
```

TLoop:

```
    mov     a,VoltLow
    rlc     a
    mov     VoltLow,a
    mov     a,VoltHigh
    rlc     a
    mov     VoltHigh,a
    mov     a,r3
    addc    a,r3
    da      a
    mov     r3,a
    mov     a,r4
    addc    a,r4
    da      a
    mov     r4,a
    djnz    r5,Tloop
    ret
    end
```

Figure 8.10 AD2 program of 8951

Step 2: Insert the programmed 8951 onto the 8951 socket on LP-2900.

Step 3: Complete the circuit entry, Figure 8.12, by using proper logic gate in the Graphic Editor of MAX+plus II.

Step 4: Simulating the circuit, and test whether the functions meet the circuit specifications. Please proceed the functional simulation after modifying Figure 8.12 to Figure 8.13. Figure 8.14 is the simulation result of counter and display interface and it meets the specifications.

```
:1000000002000375817075340075900675A0207527
:10001000B0FFC090C0A053B0FD53B03F43B0C04349
:10002000B00253B0FE75A02075900775900620B2FF
:10003000FD53B07FE58043B08043B001D0A0D090A5
:1000400012009F120087C0E0C0D0D2D3C2D41200E9
:1000500057D0D0D0E00112783406B604027600B64C
:1000600000078530A075900522B601078531A0757F
:10007000900422B6020A8532A043A0107590032294
:100080008533A0759002227930E53575F01084A78C
:10009000F009F709E53675F01084A7F009F7227525
:1000A000F002A485F036F5351200B08C368B35227F
:1000B0007D10E4FBFCE53533F535E53633F536EBFD
:1000C0003BD4FBEC3CD4FCDDEC2249
:00000001FF
```

Figure 8.11 AD2 hexadecimal file

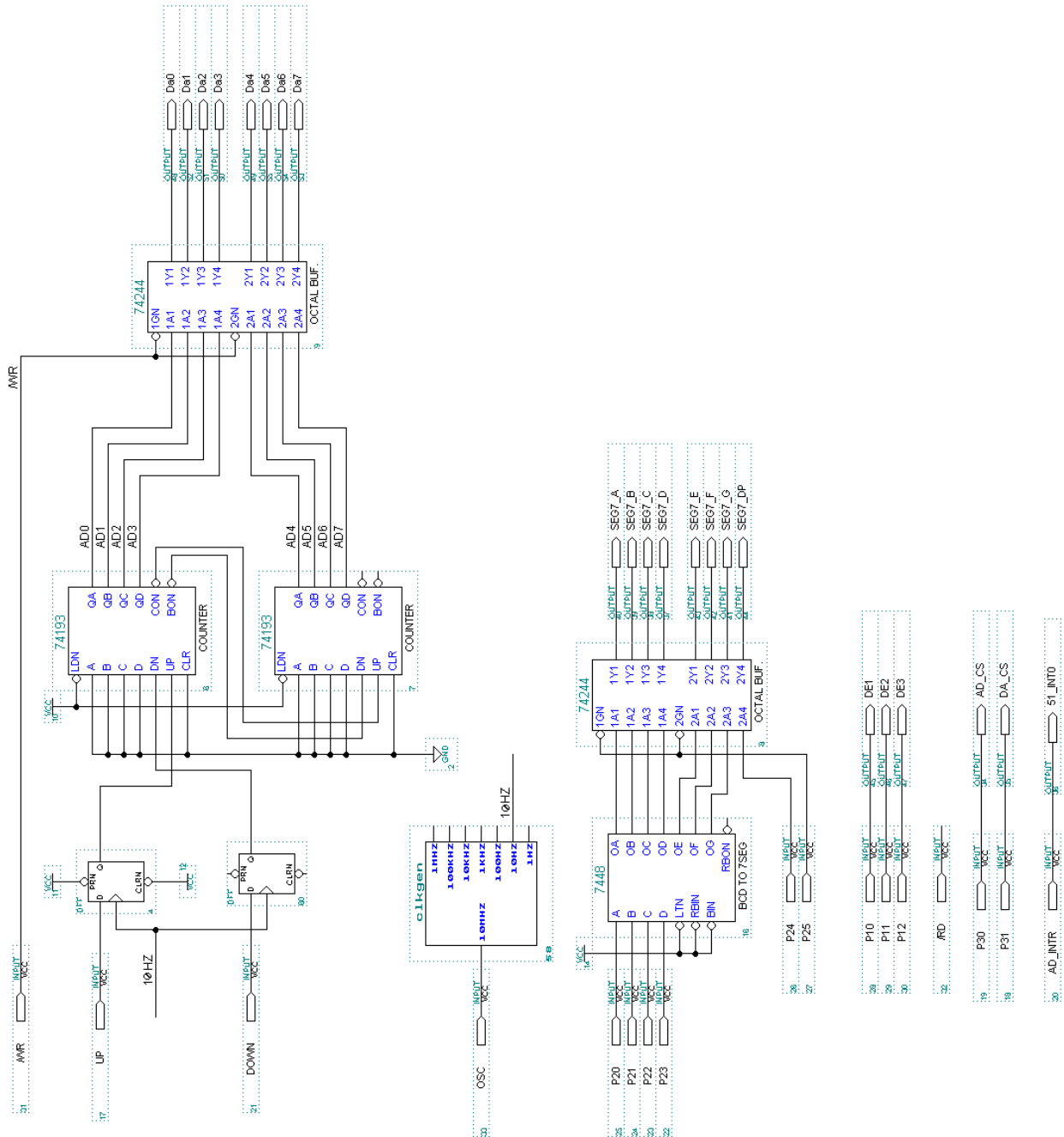


Figure 8.12 Counter and Display interface

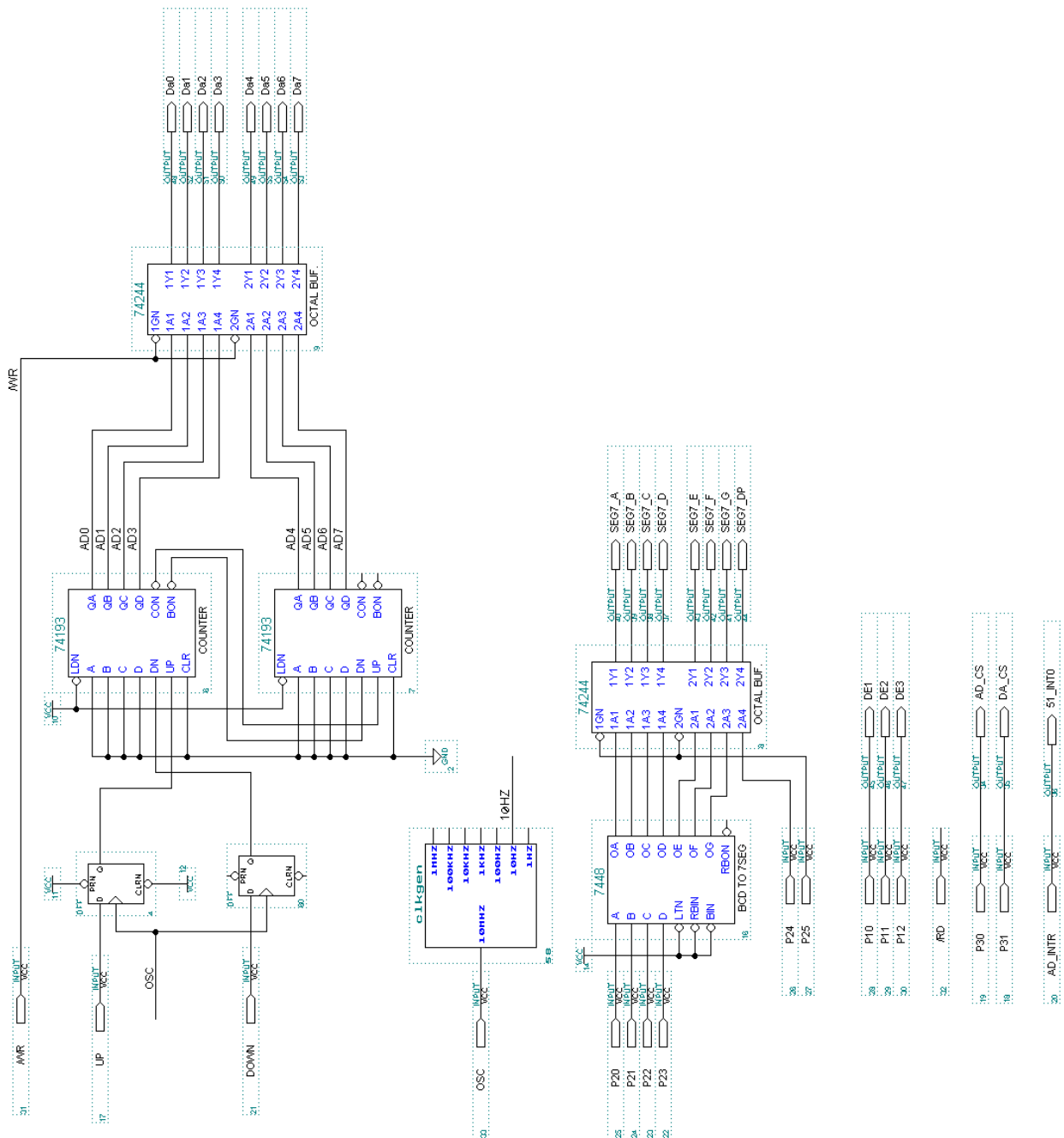


Figure 8.13 Modified counter and display interface for simulation and verification purpose

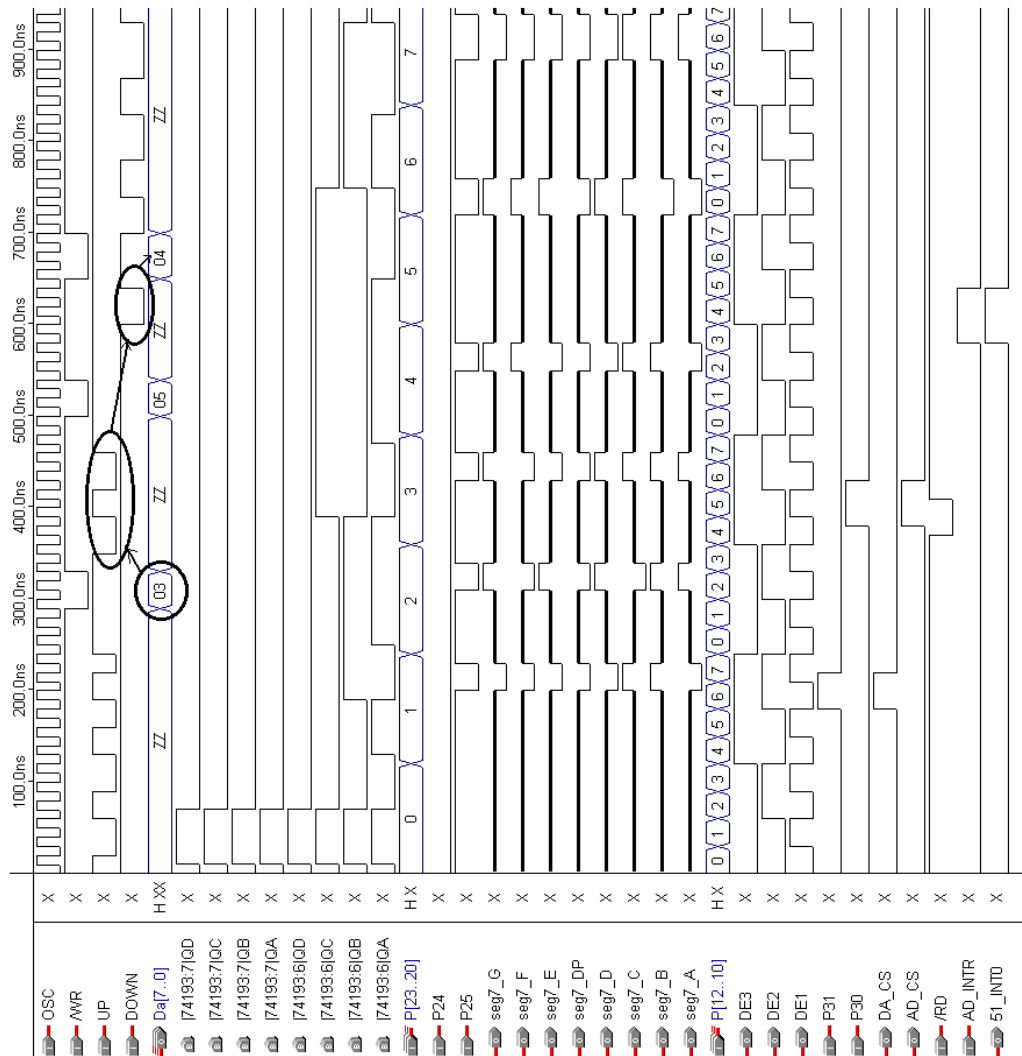


图 8.14 Simulation result of Figure 8.13 circuit.

Step 5: Please download the circuit, Figure 8.13, after floorplan programming for testing circuit. You may use the floorplan techniques illustrated in Section 4.6 and select an EPF10K10TC144-4 chip while referencing the pin assignment listed in Table 8.7. After assemble up LP-2900 Lab platform, please download the designed circuit to the EPF10K10TC144-4 chip and connect ADIN of CON7 with DAOUT1 by wire. Therefore, the signals transmitted by D/A converter can send to A/D converter for conversing. Try to push the button of UP (PS4) on the left middle of LP-2900 and note



weather the digits of 7-segment displayer raised (2.48v is the highest). Then, try to push the button of DOWN (PS2) on the left middle of LP-2900 and note weather the digits of 7-segment displayer decreased.

Table 8.7 Pin assignment EPF10K10TC144-4 chip

Name of Signal	EPF10K10TC144-4 chip pin	Name of Signal	EPF10K10TC144-4 chip pin
DA0	Pin 131	SEG7_A	Pin 23
DA1	Pin 132	SEG7_B	Pin 26
DA2	Pin 133	SEG7_C	Pin 27
DA3	Pin 135	SEG7_D	Pin 28
DA4	Pin 136	SEG7_E	Pin 29
DA5	Pin 137	SEG7_F	Pin 30
DA6	Pin 138	SEG7_G	Pin 31
DA7	Pin 140	SEG7_DP	Pin 32
P2.0	Pin 7	P3.0	Pin 41
P2.1	Pin 8	P3.1	Pin 144
P2.2	Pin 9		
P2.3	Pin 10	AD_CS	Pin 38
P2.4	Pin 11	DA_CS	Pin 39
P2.5	Pin 12	OSC	Pin 55
P1.0	Pin 17	51_INT0	Pin 98
P1.1	Pin 18	AD_INTR	Pin 143
P1.2	Pin 19	DOWN	Pin 56
DE1	Pin 33	UP	Pin 126
DE2	Pin 36	AD_RD	Pin 128
DE3	Pin 37	/WR	Pin 122



8.5 Evaluation

Please evaluate the accomplishment according to the following questions:

- ☐ Do you know which A/D converter is adapted in this chapter?
- ☐ Do you know which D/A converter is adapted in this chapter?
- ☐ Can you describe the steps to activate A/D converter?
- ☐ Can you describe the steps to activate D/A converter?

CHAPTER 9

CPLD LOGIC DESIGN LAB PLATFORM LP-2900



9.1 Function Description to LP-2900

Due to the rapid improvement in electronic technology in these years, our life is becoming more and more convenient and comfortable. The domestic electrics, mobile phones and computer side products all of these indicate the electric products are getting smaller, light-weighted, and powerful. Digital circuits are gradually replacing the analog circuits. At the mean time, with the improvement of production in digital circuits, the standard IC, such as TTL/COMS is being replaced by CPLD/FPGA. With the population of CPLD chip, it is about time to reverse the teaching methodologies in Logic Design. LP-2900 CPLD Logic Design Lab Platform (Figure 9.1) is a product produced by Leap Electronic Co. in 1999. Under the devoted research and development, Leap Electronic Co. integrates the major functions--design, simulation, and verification--to provide a comprehend logic design teaching environment, in which the features are easy to set up and operate, instant response, and the course arrangement set from generous to sophisticated.

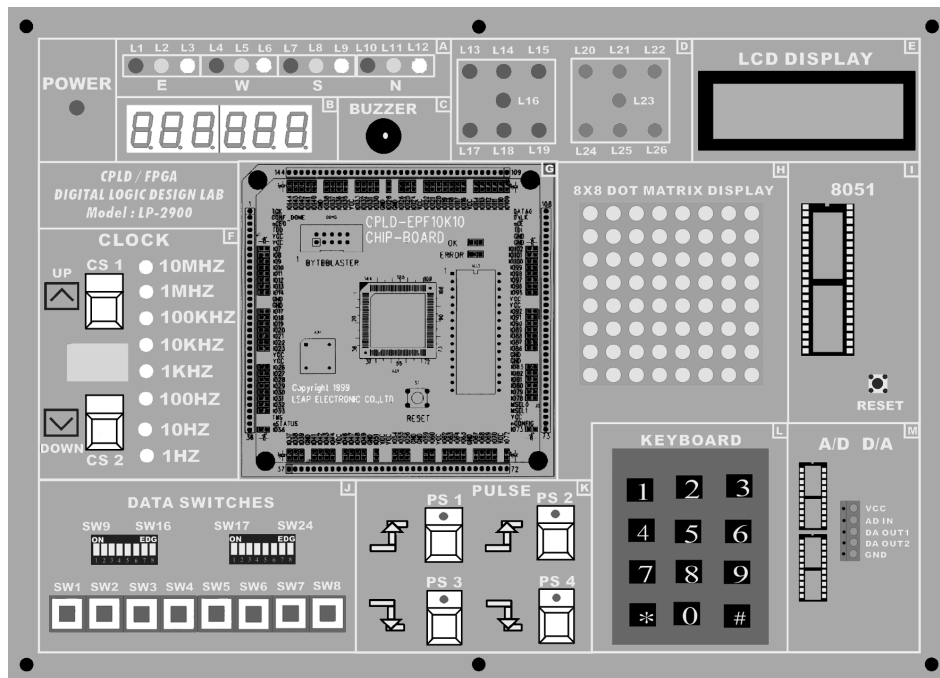


Figure 9.1 LP-2900 Logic Design Lab Platform



Taking ALTERA EPF10K10TC144-4 CPLD chip as the core, LP-2900 is designed as a multi-functional logic design lab platform, which is divided into four sections, CPLD chipboard, I/O device lab board, PC printer download interface, and power.

❖ CPLD chip board

The four parts (Figure 9.2) set on the CPLD chip board include one ALTERA 10K series chip, one EPROM chip socket, one reset bottom, and one pin status display LED (Surface Mounted Device, SMD). ALTERA EPF10K10TC144-4 CPLD chip provides a diversity and convenient rout of constant re-loading to program new circuits. To provide an alternative method to program, users can insert EPROM chip with programmed “configuration data” to EPROM chip socket. The reset bottom is set to allow 10K chip to exit the user mode and enter into the command mode. After configuring the circuits and resetting, it will progress to re-activate the user mode. The programming methods introduced in this book allow 10K chip automatically exit user mode, enter into the command mode to configure and reset and then re-activate the user mode. Thus, it is not necessary to push the reset bottom before downloading. The pin status display LED is a SMD, showing the status of each pin after the power turns on, for detecting the situation of the circuits.

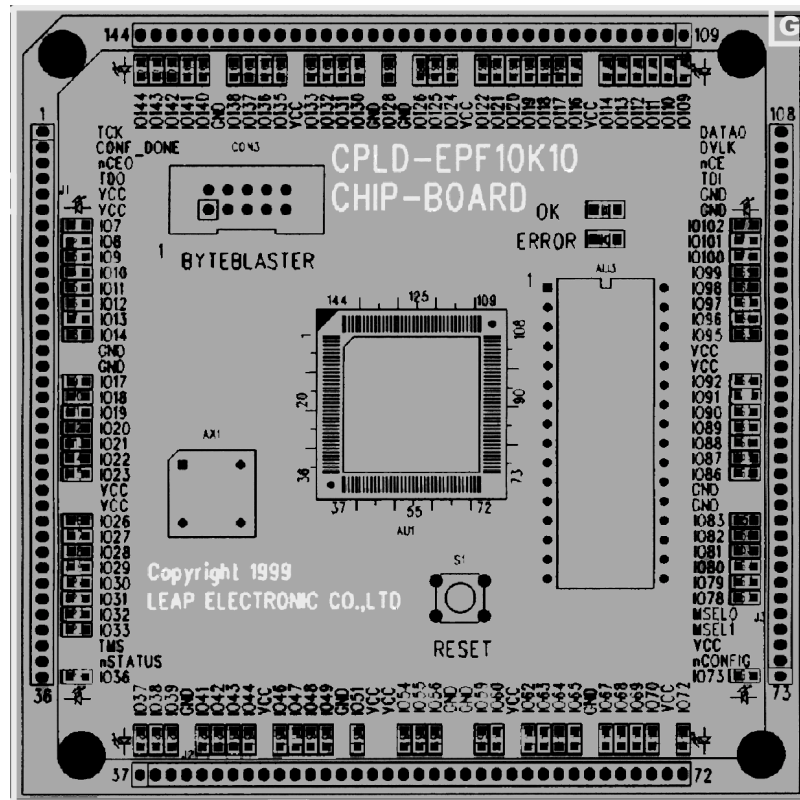


Figure 9.2 CPLD chip-board

❖ I/O device lab board

The big board under the CPLD chipboard is the I/O device lab board. There are 12 I/O devices on the board, which are (1) 4 sets of Red-Yellow-Green LED; (2) 6 common cathode 7 segment display; (3) one buzzer; (4) 2 electronic dices; (5) one clock circuit; (6) 3 sets of 8-bit data switch; (7) 4 pulse bottom; (8) one frequently-used 4 × 3 keyboard; (9) one 8 × 8 dot matrix LED display; (10) one LCD; (11) A/D & D/A circuit module; and (12) 8051 single chip module. The board contains all frequent-used I/O devices of digital logic circuits to provide a learning comprehend environment.

❖ PC printer download interface circuit

The circuit provides downloading “configuration data” from PC printer port that is a



convenient rout for programming 10K chip. It saves the trivial works for disassembling to set the interface. The only job that takes is to connect printer bus line to PC printer port.

❖ Power

AC 90V~260V, 50Hz/60Hz, DC 5V, 2A input, providing all the power needed for all circuits and it has a short-circuit-protected measure.

LP-2900 CPLD Logic Design Lab Platform should be used with ALTERA MAX+PLUS II software. Currently, the top three popular versions in the market of Taiwan are (1) Business Version (9.1 version, February 1999). To make the whole functions work, it requires a connecting key-pro to the printer port. (2) Student Version (7.21SE version, June 1999). Required by the US universities to apply CPLD courses, 7.21SE Version is designed for educating. Although this version does not provide functional simulation (except timing simulation), and only supply 2 types of chip, EPM7128S and EPF10K20, however, it provides VHDL design environment. (3) Baseline Version (9.23 Version, June 1999). Except for VHDL design environment, it provides a design environment as graphic entry, text entry, and waveform entry. Meanwhile, it provides functional and timing simulations. This version can be used with any ALTERA chips. For free download and usage of Student and Baseline Version, please login at <http://www.altera.com>.

The features of LP-2900 CPLD Logic Design Lab Platform are :

1. Easy to setup and collect;
2. Clear description on the board for easier operation;
3. EPF10K10TC144-4 CPLD chip on the CPLD chipboard provides constant reloading for programming new circuits. This is very flexible and



convenient. Meanwhile, EPF10K10TC144-4 chip offers a wide range circuit application from tiny to large circuits.

4. The pin status display of SMD LED shows the current status of each pin. This equipment is quite convenient for circuit detection.
5. I/O device lab-board comprises most of the I/O devices required for digital logic circuit, providing a comprehensive learning environment.
6. Used with MAX+PLUS II, it provides an integrated environment of design, simulation, and verification of digital circuit. This integrated environment can not only ease the universities' courses on digital logic design, digital circuit design, digital system design and VHDL digital circuit design but also provide an excellent environment for the department of R&D to develop circuits.
7. The book "CPLD Logic Circuit Design and Practice" is edited by graphic-oriented, arranged from simple to sophisticated, draw out a lots of illustrations, instruct with detail descriptions. Teaching with that book to present logic design instruction and practices, and unite theories and phenomenon testimony are the brand new teaching methodology.
8. To complete the combined circuit practices using 8051 and CPLD.
9. Compatible with WIN95/98/2000/NT working systems.

9.2 Setting up LP-2900

❖ Setting up LP-2900 CPLD Logic Design Lab Platform

It requires only two steps to setup LP-2900,

1. Connecting the power cord with 110V socket ;
2. Connecting one terminal of the printer bus line with the parallel port of PC,

which has MAX+PLUS II, and the other with the parallel port of LP-2900 CPLD Logic Design Lab Platform.

9.3 The Architecture and Circuits of LP-2900

1. Figure 9.3 illustrates LP-2900 Lab-Platform is composed by four parts, CPLD chipboard, I/O Lab-board, PC printer download interface and power.
2. Figure 9.4 demonstrates the connection status of the three components on the CPLD chipboard. There are module connector, CLPD chip, and ISP download bus connector.
3. Figure 9.5 illustrates the connection status of the eleven parts on the I/O Lab-board that are module connector, buzzer, 7 segment display, clock, pulse keys, 8 × 8 dot matrix display, LCD module, 4 × 3 keyboard module, 8051 single chip, A/D & D/A module, and LED display, including the LED of dice array.
4. Figure 9.6 describes the connection of EPF10K10TC144-4.
5. Figure 9.7 is the socket layout of the chip module.
6. Figure 9.8 is the circuit layout of I/O status LED on chip module.
7. Figure 9.9 is the download interface of EEPROM and printer parallel port.
8. Figure 9.10 illustrates the connection on I/O Lab-board.
9. Figure 9.11 demonstrates the module connector on CPLD chipboard.
10. Figure 9.12 is the LED display driver on I/O Device Lab-board.
11. Figure 9.13 is the driver of 7-segment display and buzzer on I/O Device Lab-board.
12. Figure 9.14 is the buttons of pulse and clock up/down on I/O Device Lab-board.

13. Figure 9.15 is the circuits of 8×8 Dot Matrix Display, LCD module, and 4×3 keyboard.
14. Figure 9.16 is the circuits of data switch on I/O Device Lab-board, one of which is a set of 8-key with LED display.
15. Figure 9.17 illustrates A/D and D/A circuits on I/O Device Lab-board.
16. Figure 9.18 demonstrates 8×8 Dot Matrix Module on I/O Device Lab-board.

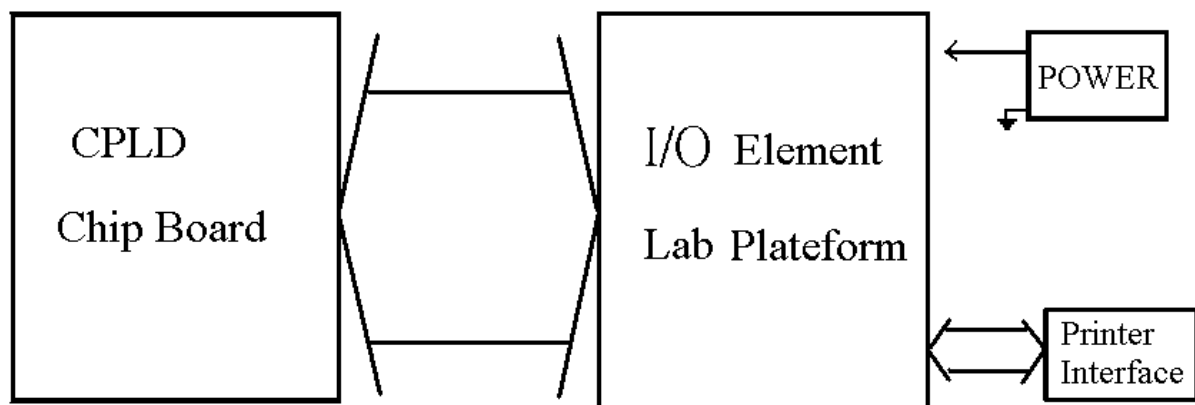
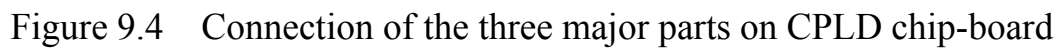


Figure 9.3 Blocks of LP-2900



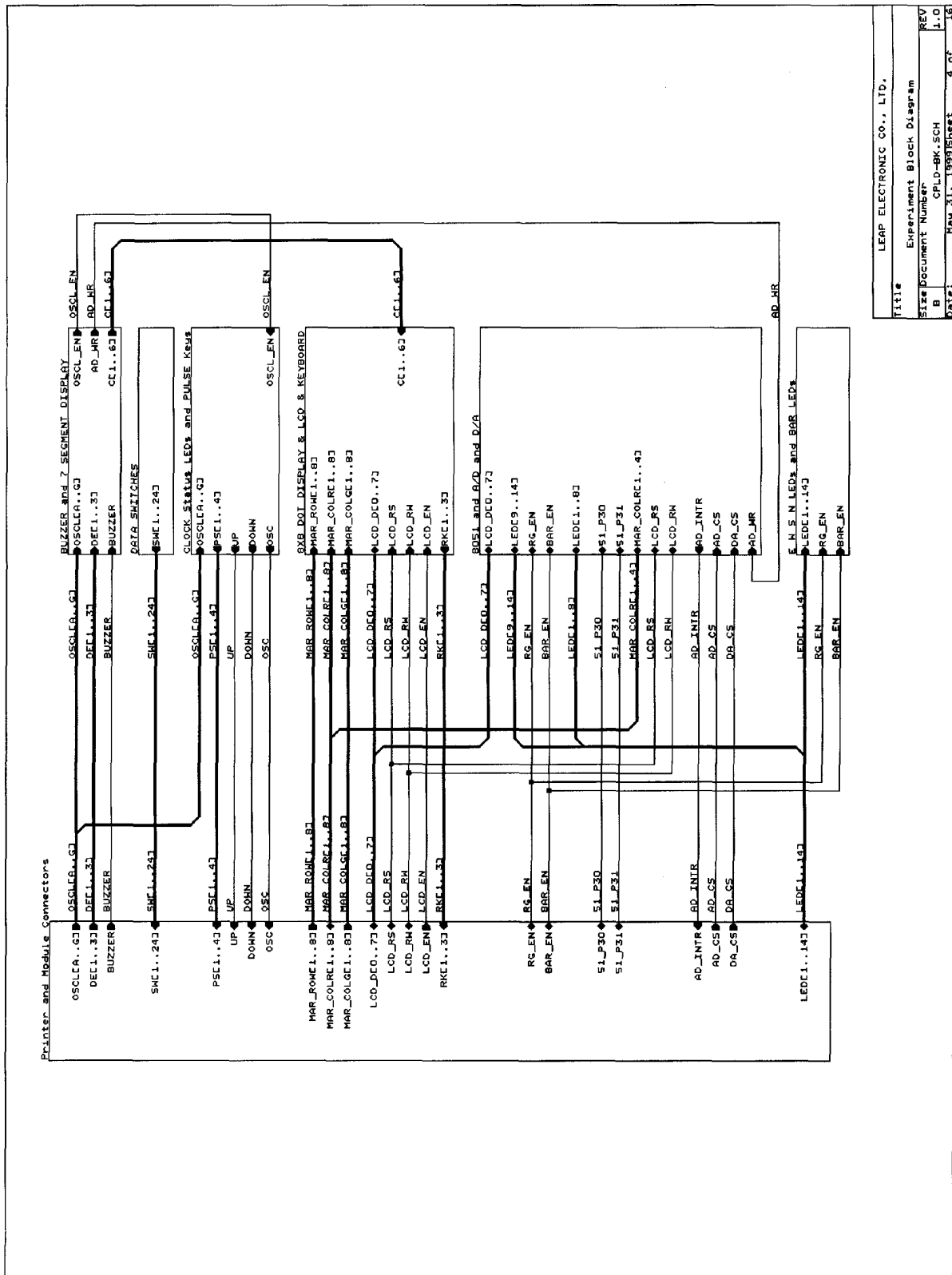


Figure 9.5 Block Diagram of I/O Device Lab-board

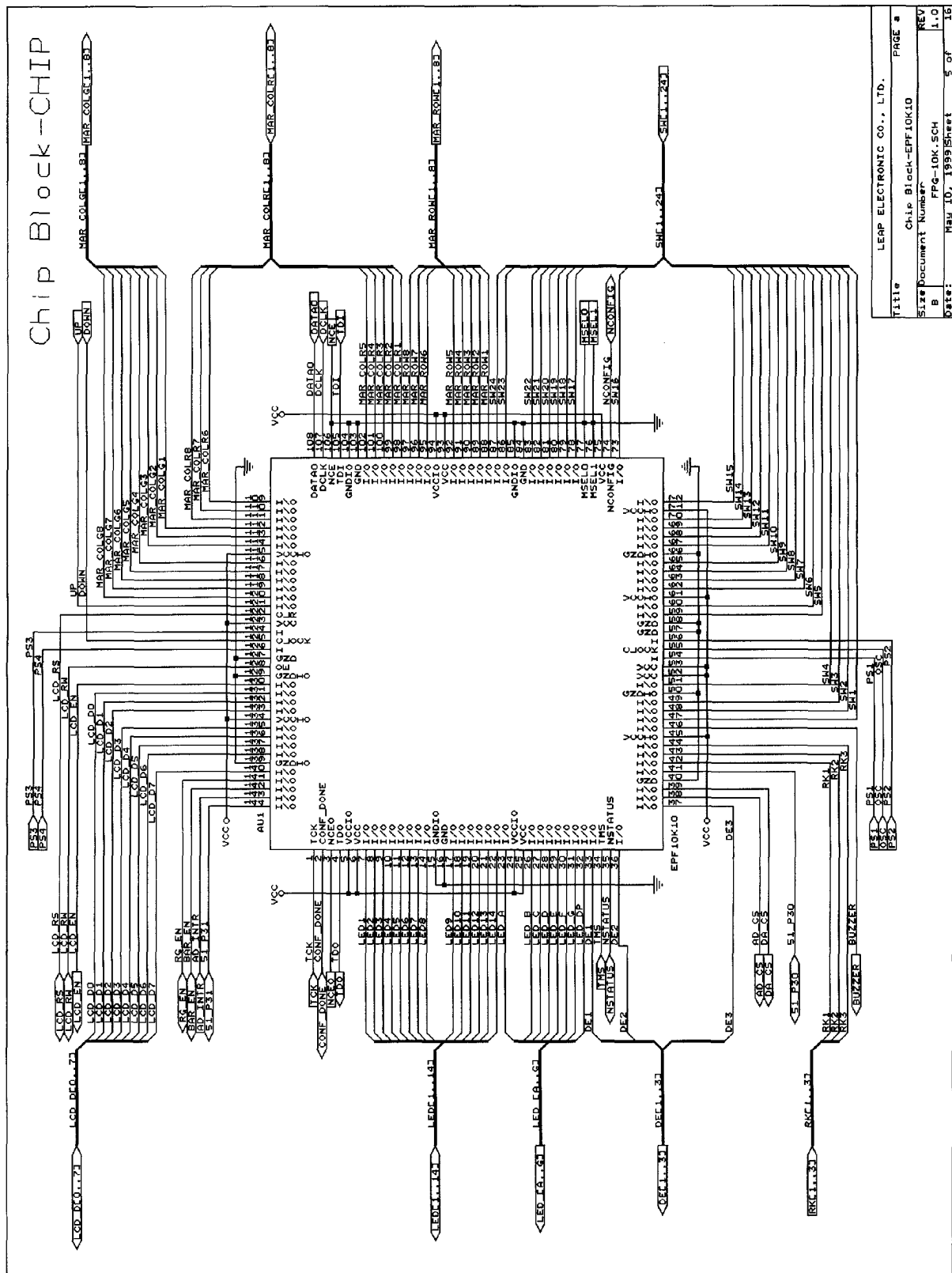


Figure 9.6 Connection of EPF10K10TC144-4

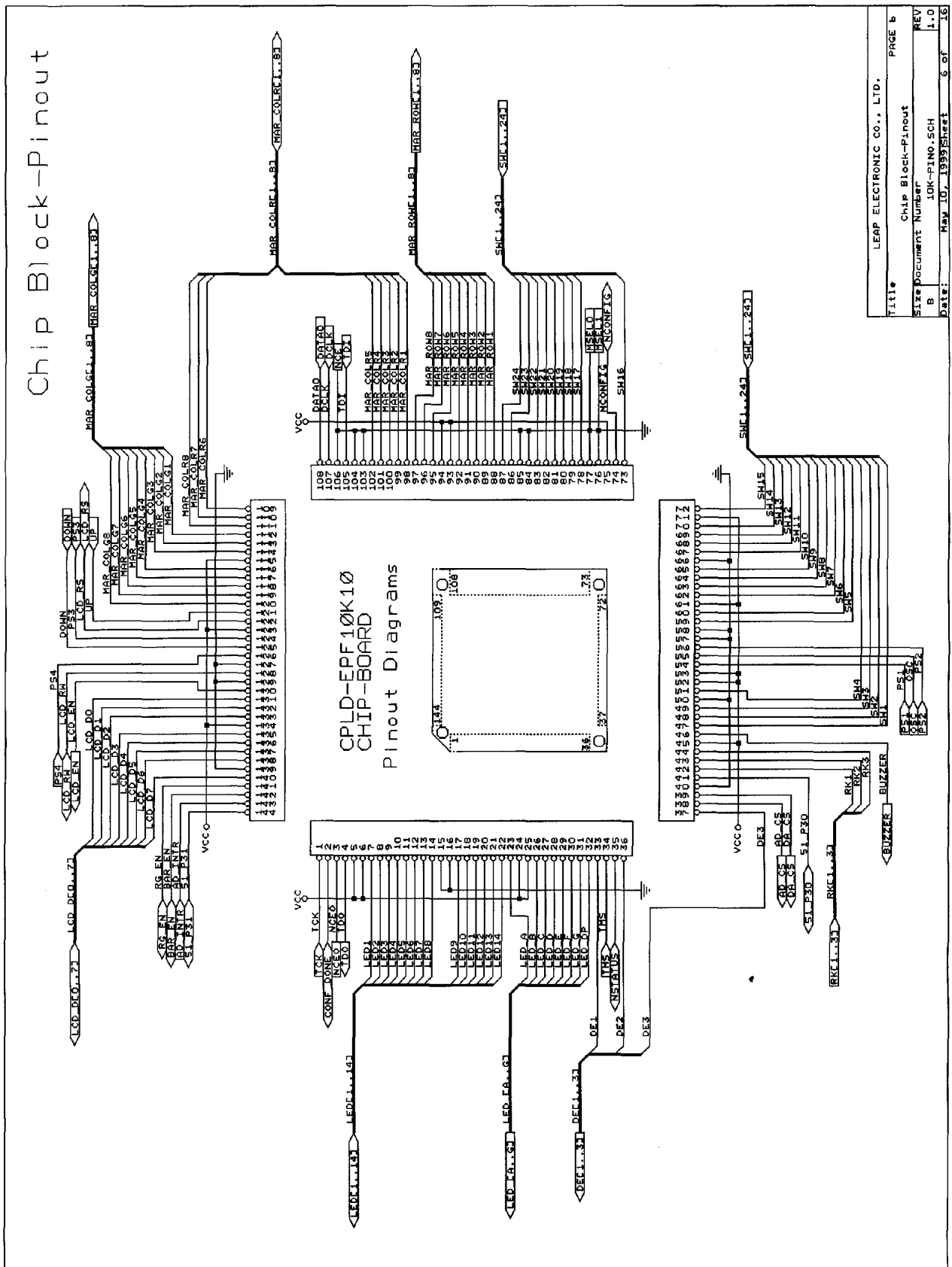
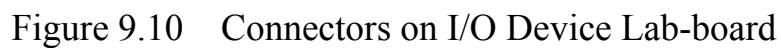


Figure 9.7 Chip Block-Pinout









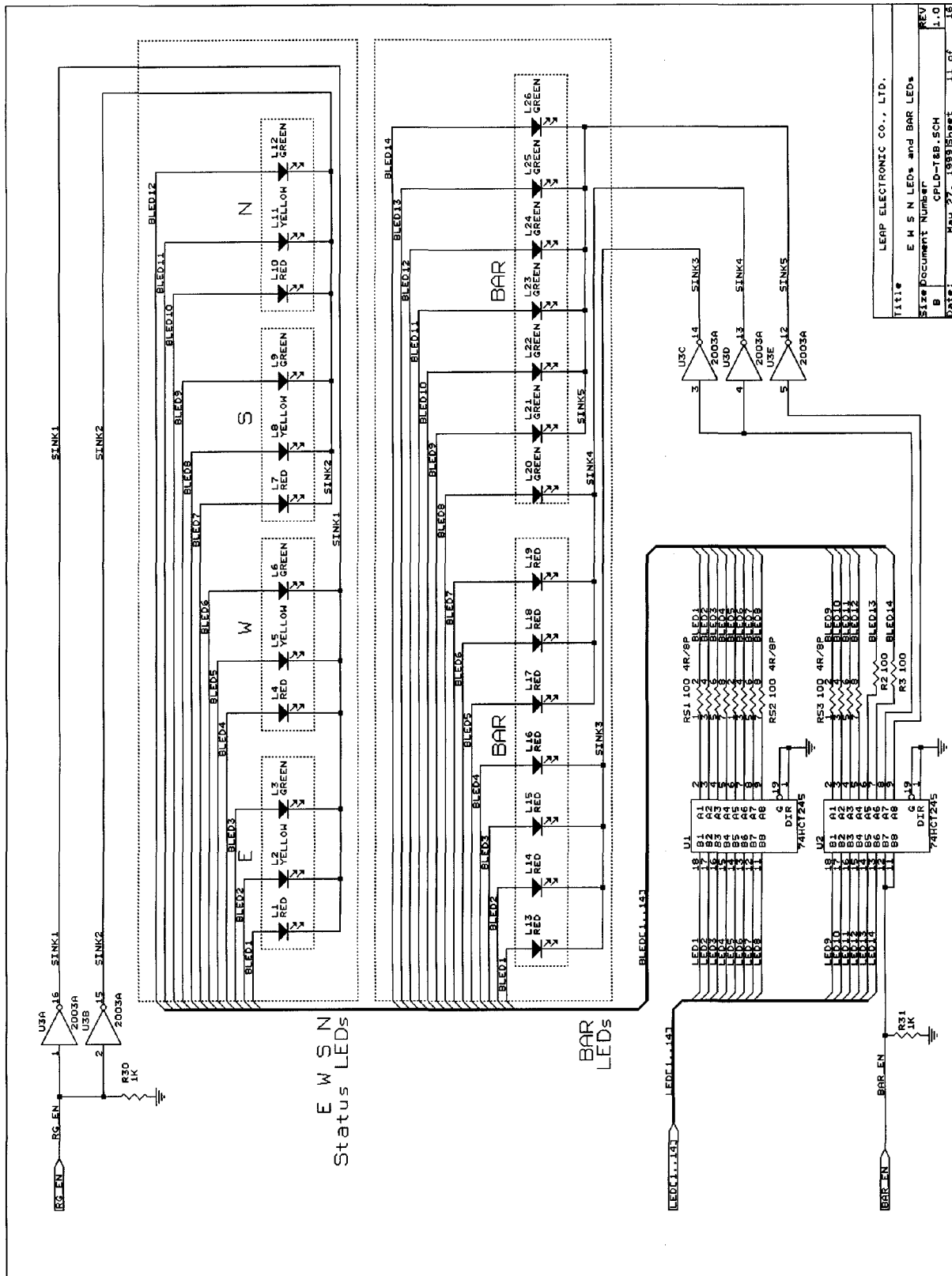


Figure 9.12 LED Display Driver on I/O Device Lab-board

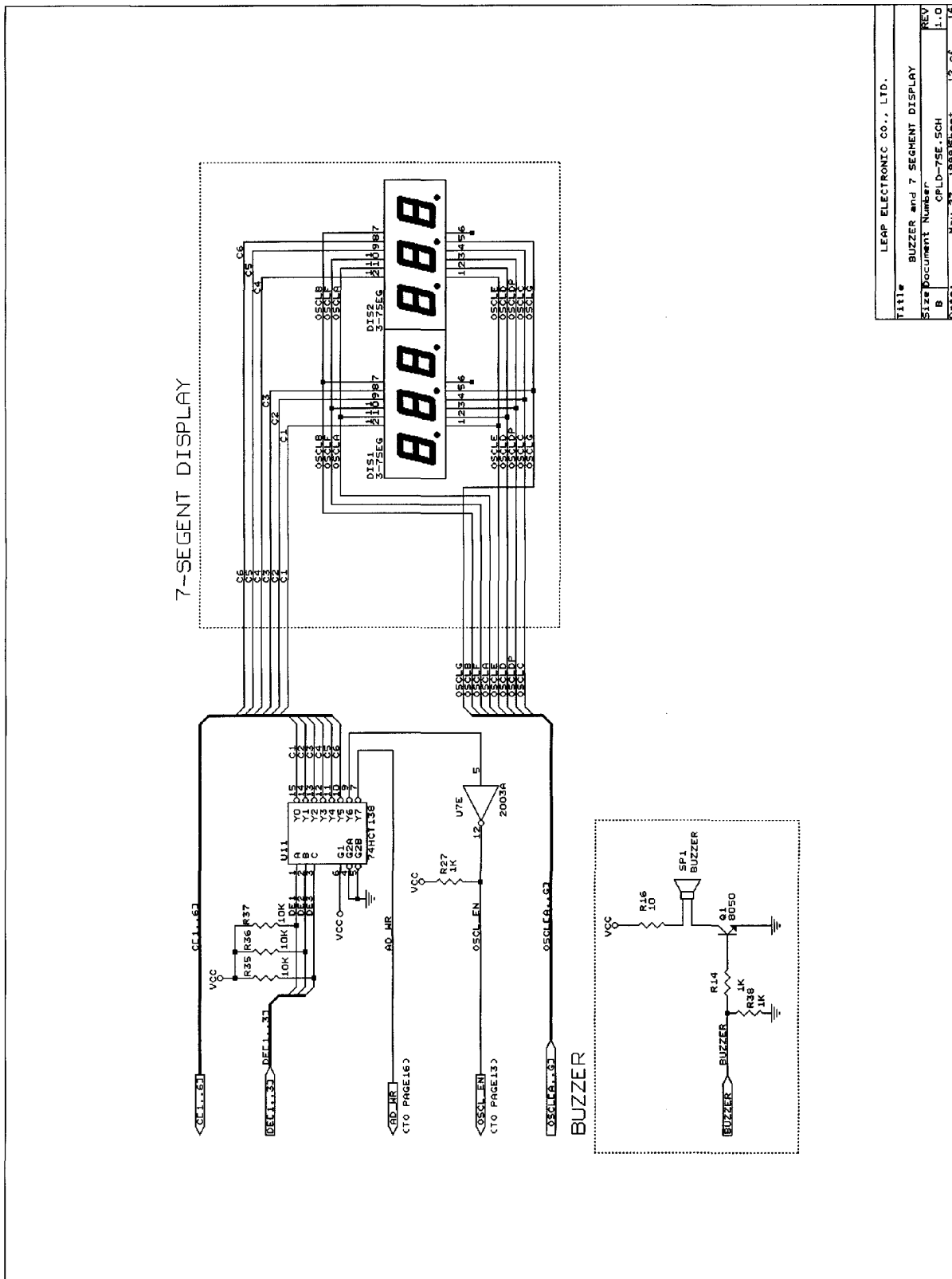


Figure 9.13 7-Segment Display and Buzzer on I/O Device Lab-board

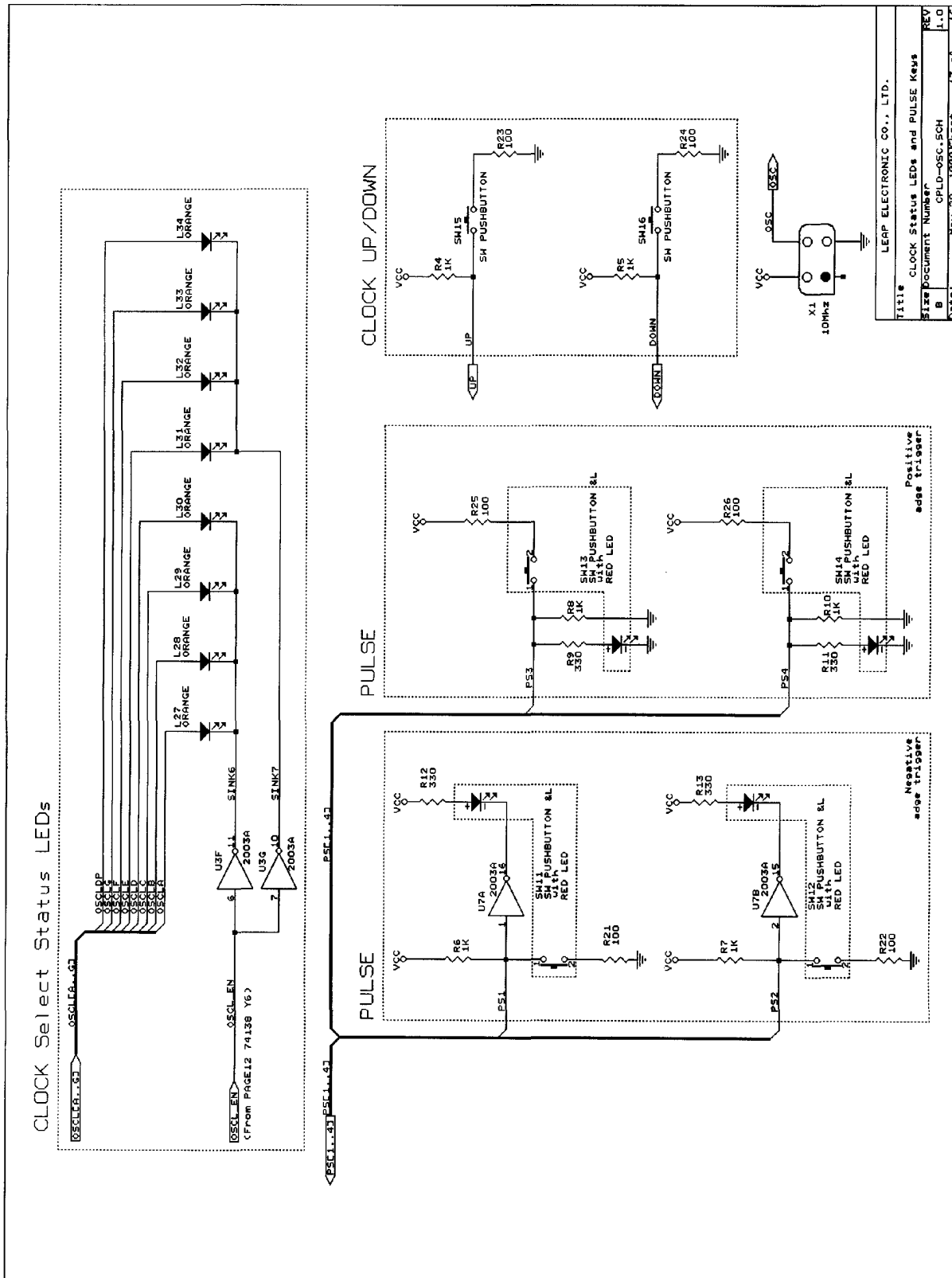


Figure 9.14 Pulse and Clock Up/Down Push Buttons on I/O Device Lab-board

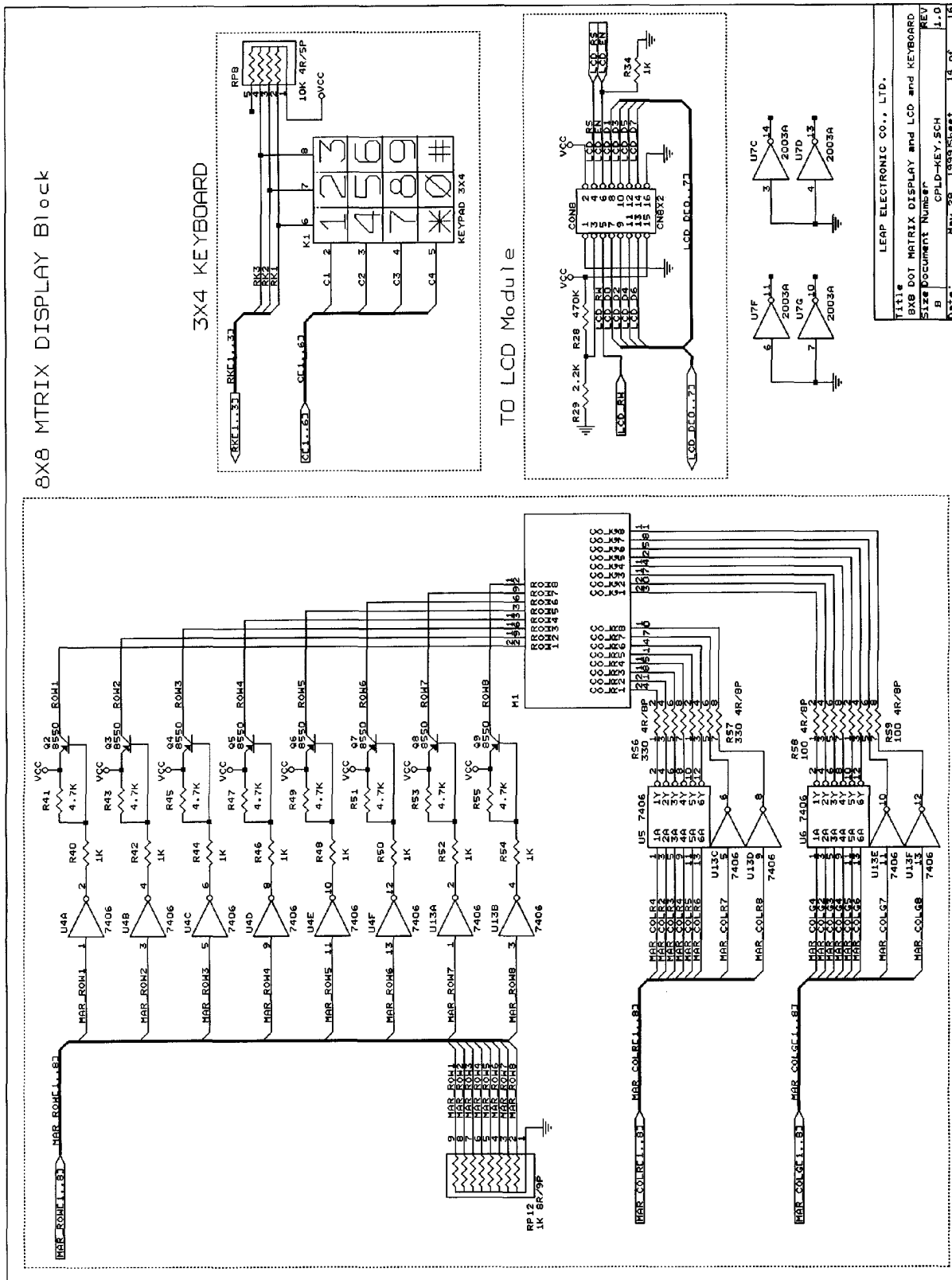


Figure 9.15 Dot Matrix Display, keyboard, and LCD Module

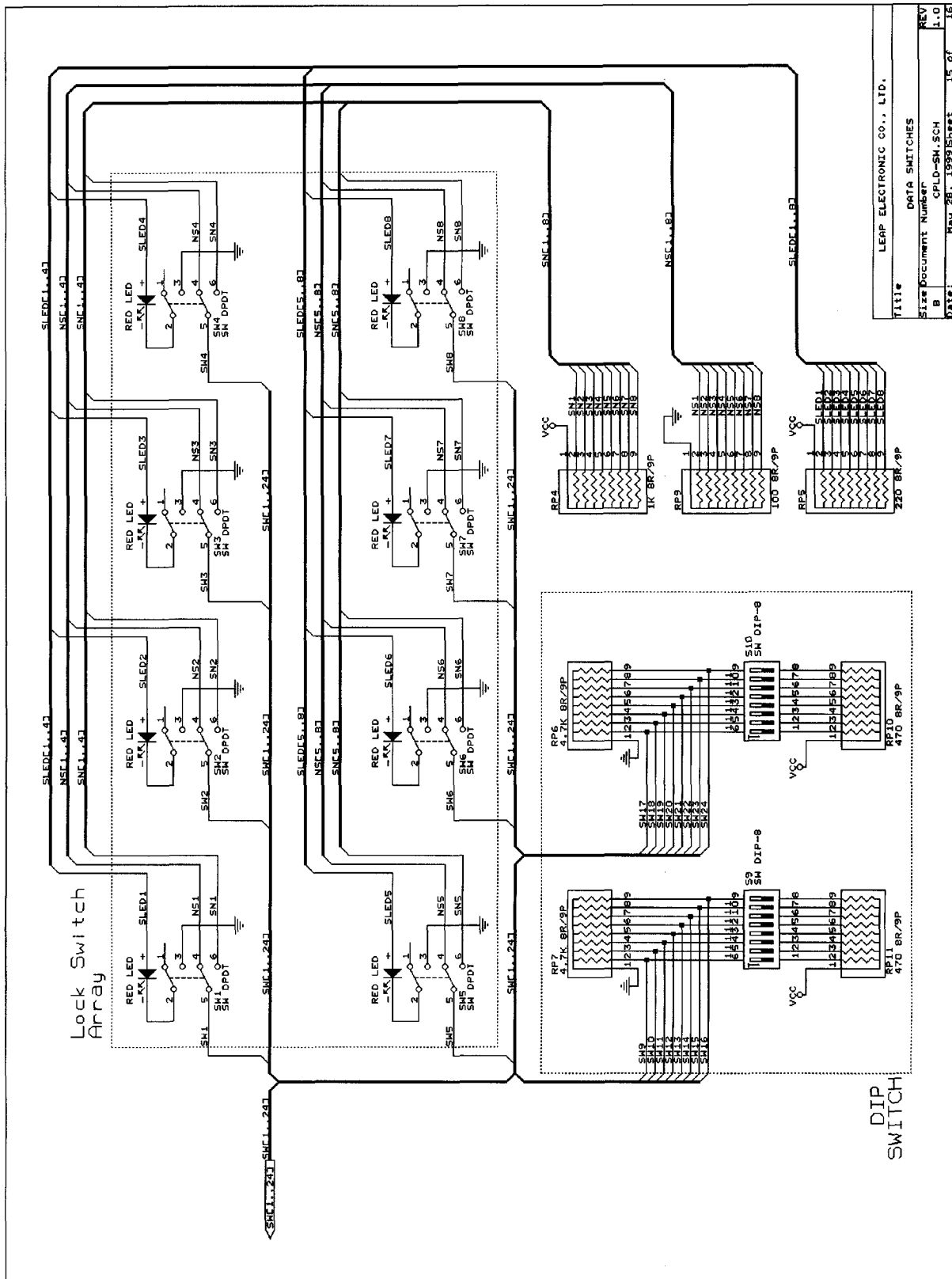
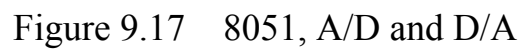


Figure 9.16 Data Switches on I/O Device Lab-board



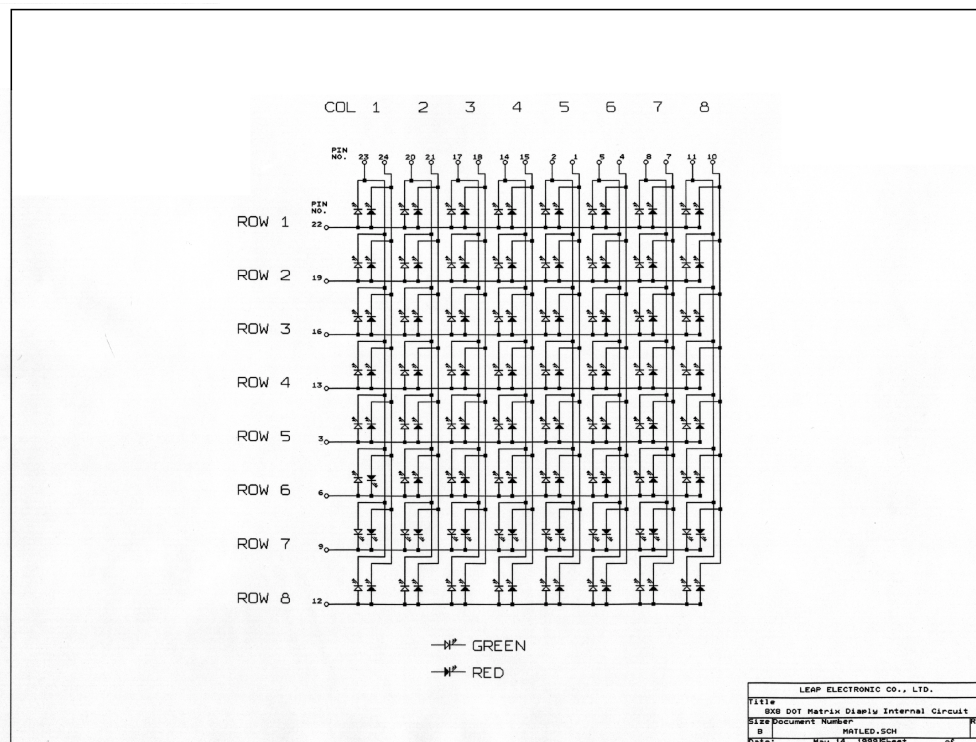
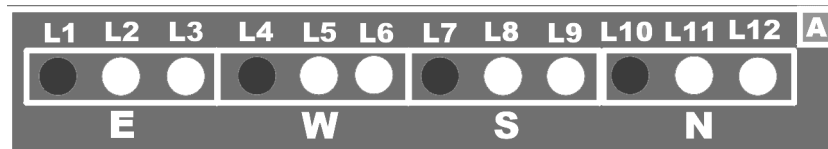


Figure 9.18 8 × 8 dot matrix module on I/O device lab board

9.4 Pin arrangement of LP-2900

Pin arrangement data, an important data that highly used, is an essential data for floor plane setup. The pin arrangement data of ALTERA EPF10K10TC144 on LP-2900 is described in detail as follows.

9.4.1 Red-Yellow-Green LED



Code	L1	L2	L3	L4	L5	L6	L7	L8
Device	Red	Yellow	Green	Red	Yellow	Green	Red	Yellow
Pin	Pin 7	Pin 8	Pin 9	Pin 10	Pin 11	Pin 12	Pin 13	Pin 14

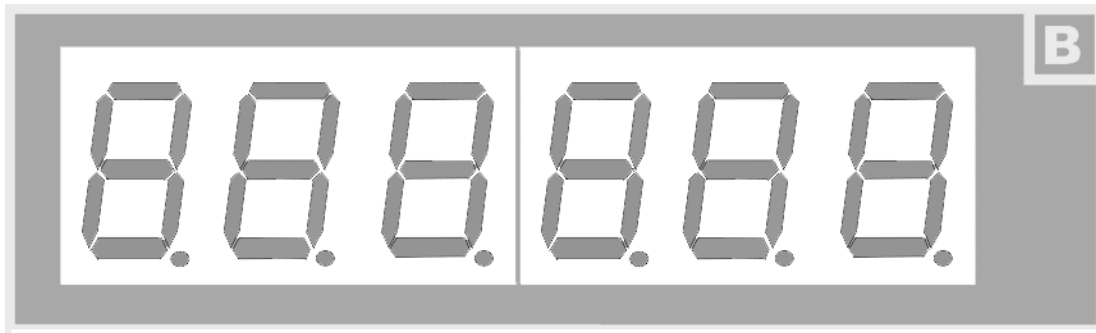
Code	L9	L10	L11	L12	LED_COM
Device	Green LED	Red LED	Yellow LED	Green LED	Common cathode of LED1～ LED12
Pin	Pin 17	Pin 18	Pin 19	Pin 20	Pin 141

* L1~L12 are LED anode inputs for each LED

† LED_COM is the common cathode of all LED



9.4.2 7-Segment Display with Common Cathode



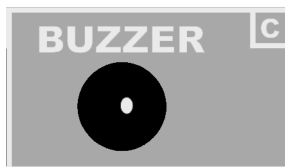
Code	A	B	C	D	E	F	G	DP
Device	7 Segment Display							
Pin	Pin 23	Pin 26	Pin 27	Pin 28	Pin 29	Pin 30	Pin 31	Pin 32

Code	DE1	DE2	DE3	—	—	—	—	—
Device	74138			—	—	—	—	—
Pin	Pin33	Pin36	Pin37	—	—	—	—	—

† DE1, DE2 and DE3 are connected to 74138 which outputs Y0~Y5 as C1~C6.

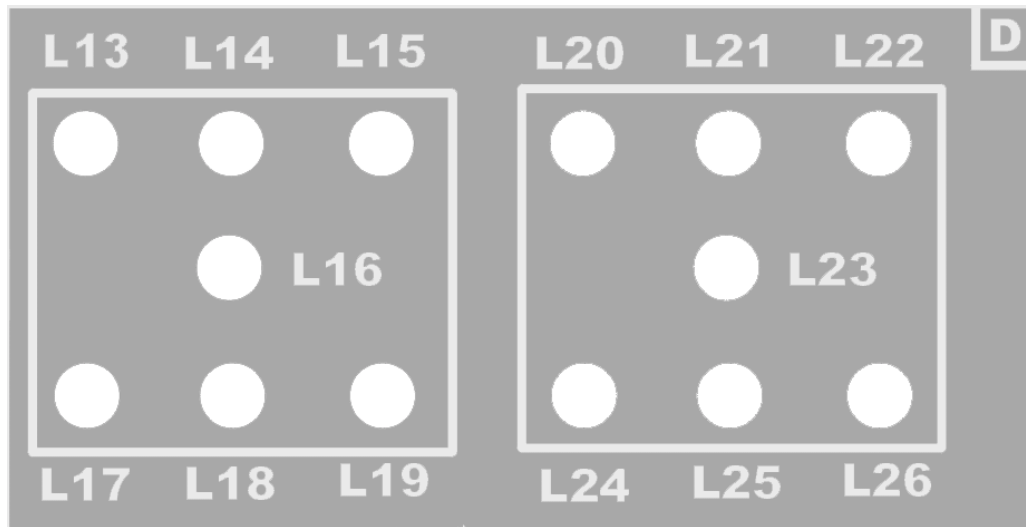
† C1~C6 are the common cathodes of 6 7-segment display.

9.4.3 BUZZER



Code	SP1
Device	Sp1
Pin	Pin 46

9.4.4 Electronic Dice



Code	L13	L14	L15	L16	L17	L18	L19
Device	Red Dice						
Pin	Pin 7	Pin 8	Pin 9	Pin 10	Pin 11	Pin 12	Pin 13

Code	L20	L21	L22	L23	L24	L25	L26
Device	Green Dice						
Pin	Pin 14	Pin 17	Pin 18	Pin 19	Pin 20	Pin 21	Pin 22

Code	Dice_COM	—	—	—	—
Device	Common cathode of L13~L26	—	—	—	—
Pin	Pin 142	—	—	—	—

† L13~L26 are anode inputs for each LED

† Dice_COM is the common cathode of L13~L26

9.4.5 LCD display



Code	EN	RS	RW	D0	D1	D2	D3	D4
Device	LCD							
Pin	Pin 130	Pin 122	Pin 128	Pin 131	Pin 132	Pin 133	Pin 135	Pin 136

Code	D5	D6	D7	—	—	—	—	—
Device	LCD			—	—	—	—	—
Pin	Pin 137	Pin 138	Pin 140	—	—	—	—	—

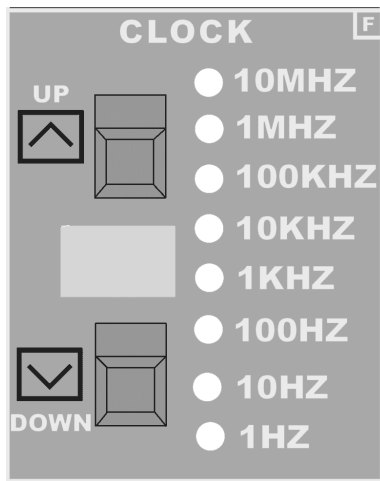
9.4.6 CLOCK

Code	L27	L28	L29	L30	L31	L32	L33	L34
Device	Yellow LED							
Pin	Pin 23	Pin 26	Pin 27	Pin 28	Pin 29	Pin 30	Pin 31	Pin 32

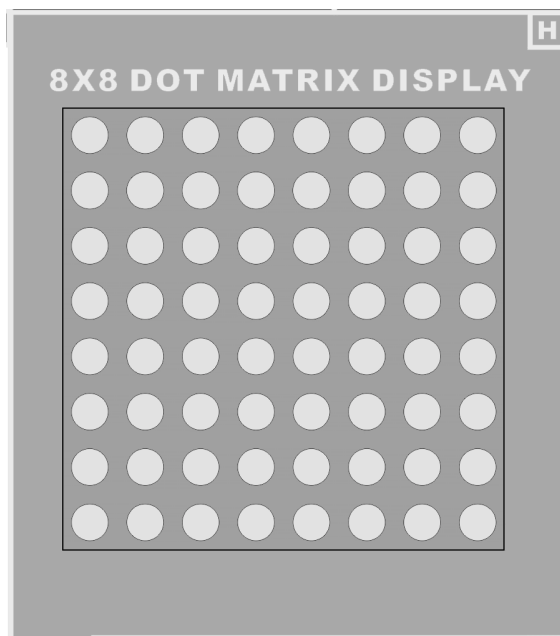
Code	DE1	DE2	DE3
Device	74138		
Pin	Pin 33	Pin 36	Pin 37

Code	OSC	UP	DOWN
Device	OSC	Button	Button
Pin	Pin 55	Pin 121	Pin 125

† DE1, DE2 and DE3 are connected with 74138 which output Y6 as common cathode of L27~L34.



9.4.7 8 x 8 Dot Matrix LED Display



❖ Common Anodes

Code	Row 1	Row 2	Row 3	Row 4	Row 5	Row 6	Row 7	Row 8
Device	8 x8 Dot Matrix							
Pin	Pin 88	Pin 89	Pin 90	Pin 91	Pin 92	Pin 95	Pin 96	Pin 97

❖ **Red Cathodes**

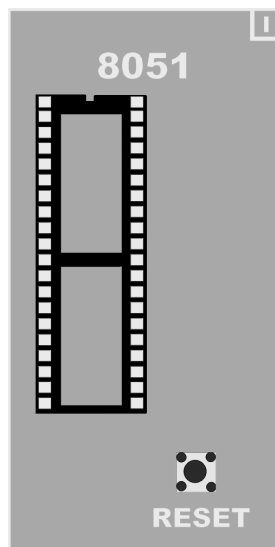
Code	CR1	CR2	CR3	CR4	CR5	CR6	CR7	CR8
Device	8 ×8 Dot Matrix							
Pin	Pin 98	Pin 99	Pin 100	Pin 101	Pin 102	Pin 109	Pin 110	Pin 111

† CR1~CR8 driven by HI

❖ **Green Cathodes**

Code	CG1	CG2	CG3	CG4	CG5	CG6	CG7	CG8
Device	8 ×8 Dot Matrix							
Pin	Pin 112	Pin 113	Pin 114	Pin 116	Pin 117	Pin 118	Pin 119	Pin 120

† CG1~CG8 driven by HI

9.4.8 8051 Single Chip

Code	P0.0	P0.1	P0.2	P0.3	P0.4	P0.5	P0.6	P0.7
Device	8051							
Pin	Pin 131	Pin 132	Pin 133	Pin 135	Pin 136	Pin 137	Pin 138	Pin 140

† P0.0~P0.7 also connect with D0~D7 of LCD



Code	P1.0	P1.1	P1.2	P1.3	P1.4	P1.5	P1.6	P1.7
Device	8051							
Pin	Pin 17	Pin 18	Pin 19	Pin 20	Pin 21	Pin 22	Pin 141	Pin 142

† P1.0~P1.7 also connect L21~L26 and LED_COM and Dice_COM。

Code	P2.0	P2.1	P2.2	P2.3	P2.4	P2.5	P2.6	P2.7
Device	8051							
Pin	Pin 7	Pin 8	Pin 9	Pin 10	Pin 11	Pin 12	Pin 13	Pin 14

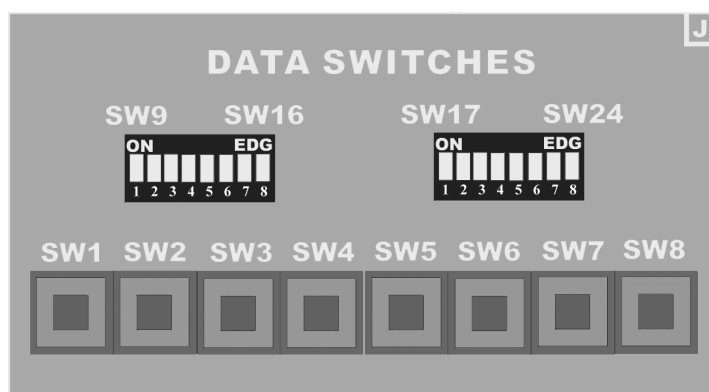
† P2.0~P2.7 also connect L1~L8。

Code	P3.0	P3.1	P3.2	P3.3	P3.4	P3.5	P3.6	P3.7
Device	8051							
Pin	Pin 41	Pin 144	Pin 98	Pin 99	Pin 100	Pin 101	Pin 122	Pin 128

† P3.2~P3.5 also connect CR1~CR4 on 8 × 8 Dot Matrix

† P3.6~P3.7 also connect RS and RW on LCD

9.4.9 DATA SWITCHES



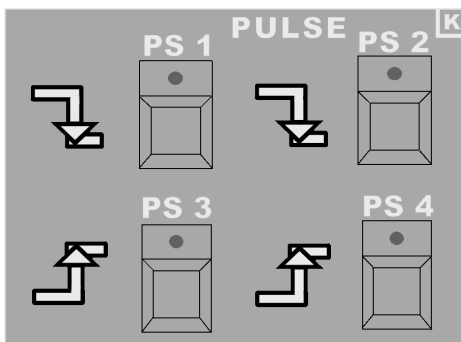
Code	SW1	SW2	SW3	SW4	SW5	SW6	SW7	SW8
Device	Push Button							
Pin	Pin 47	Pin 48	Pin 49	Pin 51	Pin 59	Pin 60	Pin 62	Pin 63



Code	SW9	SW10	SW11	SW12	SW13	SW14	SW15	SW16
Device	Dip Switch							
Pin	Pin 64	Pin 65	Pin 67	Pin 68	Pin 69	Pin 70	Pin 72	Pin 73

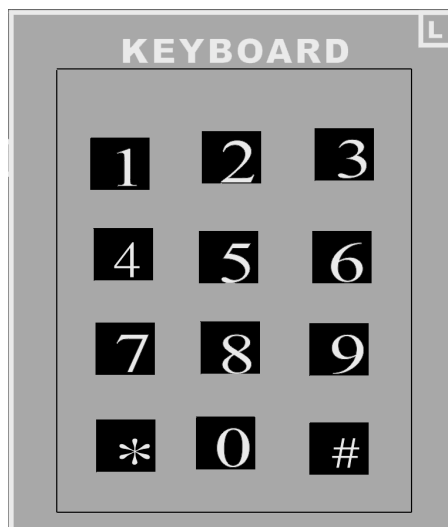
Code	SW17	SW18	SW19	SW20	SW21	SW22	SW23	SW24
Device	Dip Switch							
Pin	Pin 78	Pin 79	Pin 80	Pin 81	Pin 82	Pin 83	Pin 86	Pin 87

9.4.10 PULSE



Code	PS1	PS2	PS3	PS4
Device	Push Button with LED			
Pin	Pin 54	Pin 56	Pin 124	Pin 126

9.4.11 KEYBOARD

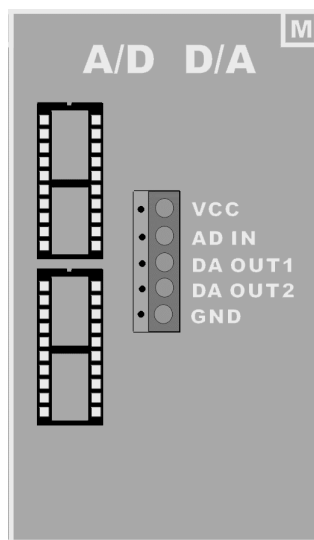




Code	DE1	DE2	DE3	RK1	RK2	RK3
Device	KEYBOARD					
Pin	Pin 33	Pin 36	Pin 37	Pin 42	Pin 43	Pin 44

† DE1, DE2 and DE3 are connected to 74138 which outputs Y0~Y3 connect to C1~C4 on the keyboard.

9.4.12 A/D, D/A



❖ A/D → ADC0804

Code	DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7
Device	ADC0804							
Pin	Pin 131	Pin 132	Pin 133	Pin 135	Pin 136	Pin 137	Pin 138	Pin 140

† DB0~DB7 also connect D0~D7 on LCD

Code	/CS	/RD	DE1	DE2	DE3	AD_INTR
Device	ADC0804					
Pin	Pin 38	Pin 128	Pin 33	Pin 36	Pin 37	Pin 143

† AD_WR connect to the output Y6 of 74138.



† DE1, DE2 and DE3 are connected to 74138 as inputs.

† /RD also connect RW on LCD

❖ D/A-->AD7528

Code	DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7
Device	AD7528							
Pin	Pin 131	Pin 132	Pin 133	Pin 135	Pin 136	Pin 137	Pin 138	Pin 140

† DB0~DB7 also connect D0~D7 on LCD

Code	/CS	/WR	/DACA	—	—	—	—	—
Device	AD7528			—	—	—	—	—
Pin	Pin39	Pin128	Pin122	—	—	—	—	—

† /WR also connect RW on LCD

† /DACA connect RS on LCD

9.5 Evaluations

Please do the following evaluations according to the questions listed below:

- ☐ Do you know which Lab-Platform is introduced in this chapter?
- ☐ Do you know which CPLD chip is adopted in this Lab-Platform?
- ☐ Do you know the main functions of Lab-Platform? Do you know how to setup?
- ☐ Do you know how to check the arrangement of Pins?

APPENDIX A

PLD Suppliers and Main Products





A.1 PLD Suppliers and Main Products

Supplier	ATMEL				
Technology	Flash or EPROM	Erase by UV*	Erase by UV*	Erase by UV*	Flash
Family	ATF1500	ATV5000	ATF750B	ATF2500B	ATF16v8ce
Gates	1500	5000	750	2500	
Speed (Mhz)	125	50	95	100	
T _{pd} (Max)	7.5 ns	25 ns	7.5 ns	7 ns	10 ns
GLB***	32	128	20	24	8
Registers	32	128	20	48	8
I/O Pins	32	52	20		16
Package	PLCC	PLCC	DIP	DIP/PLCC	DIP/PLCC
Other Features	Turbo, flash, Low Power	Low Power	Low Power	Low Power	“Zero” Stand by Power**

Supplier	ATMEL				
Technology	Flash	Flash	SRAM	SRAM	SRAM
Family	ATF220v10ce	AT6010	AT6005	AT6003	AT6002
Gates		10000	5000	3000	2000
Speed (Mhz)		250	250	250	250
T _{pd} (Max)		1.2 ns	1.2 ns	1.2 ns	1.2 ns
GLB***	10	6400	3136	1600	1024
Registers	10	6400	3136	1600	1024
I/O Pins	20	204	108	120	96
Package	DIP/PLCC	PQFP	PLCC/PGA	PLCC/PQFP	PLCC/PQFP
Other Features	“Zero” Stand by Power **	Turbo mode	Turbo mode, Cache Logic	Turbo mode, Cache Logic	Turbo mode, Cache Logic

*UV: Ultraviolet Ray; ** Typical “Zero” standby power is 10 μ A; ***GLB (Generic Logic Block).



Supplier	Cypress				
Technology	Flash	Flash	EPROM	Flash	Anti-Fuse
Family	16V8	22V10	7C340	7C370	PASIC380
Gates					1–8 K
Speed (MHz)	125	125		143	200
T _{pd} (Max)	7.5 ns	7.5 ns	15 ns	8.5 ns	
GLB**	8	10	32-192	32-128	96–768
Registers	8	10	32-192	32-128	96–768
I/O Pins	10/8	12/8	80	134	44–180
Package	DIP/PLCC	DIP/PLCC	PGA/PLCC	TQFR/PGA/ PLCC	PLCC/PGA/ TQFP
Other Features				ISP* and support VHDL	High Speed with 3.3V

*ISP: In System Programming °

Supplier	ICT			
Technology				
Family	PEEL16V8	PEEL20V8	PEEL22V10	PEEL18V8
Gates				
Speed (MHz)				
T _{pd} (Max)	25, 15, 10, 7, 5	25, 15, 10, 7, 5	25, 15, 10, 7, 5	25, 15, 10, 7, 5
GLB**				
Registers	8	8	10	8
I/O Pins	8/8	12/8	12/10	10/8
Package	24 Pin DIP/SOIC/PLCC	24 Pin DIP/SOIC	24 Pin DIP/SOIC 28 Pin PLCC	20 Pin DIP/SOIC/ PLCC/TSSOP
Other Features				



Supplier	ICT				
Technology			EEPROM	EEPROM	EEPROM
Family	PEEL22CG10	PEEL22C08	PA7024	PA7128	PA7140
Gates			40	36	72
Speed (MHz)			71.4	83.3	66
T _{pd} (Max)	25,15, 10, 7, 5	25,15,10, 7, 5	15, 20, 25	15, 20	20, 25
GLB**					
Flipflops	10	8	40	36	60
I/O Pins	10/12	14/8	2/22	14/12	14/24
Package	24 Pins DIP/SOIC	24 Pins DIP/SOIC/TSS OP	24 Pins DIP/SOIC 28 Pins PLCC	28 Pins DIP/SOIC/ PLCC	40 Pins DIP 44 Pins PLCC
Other Features					

Supplier	ALTERA			
Technology	EEPROM	EEPROM	EPROM	Flash/SRAM or EPROM/SRAM
Family	Classic	MAX5000	MAX7000	Flash Logic
Gates	150–900	60–3750	60–5000	800–3200
Speed (MHz)	111.1	125	178.6	83.3
T _{pd} (Max)	7.5 ns	10 ns	5 ns	10 ns
GLB**	8–48	16–192	32–256	40–160
Registers	8–48	16–192	32–256	RAM 20480 Bits or 40–160
I/O Pins	64	84	164	172
Package	DIP/PLCC/PGA	DIP/QFP/ PLCC/PGA	PLCC/PGA/ PQFP	PLCC/QFP
Other Features			EPM7032V uses 3.3V	3.3V and 5V used together



Supplier	ALTERA		
Technology	EEPROM	SRAM	SRAM
Family	MAX9000	FLEX8000	FLEX10K
Gates	6000–12000	2500–16000	10K–100K
Speed (MHz)	125	125	70
T _{pd} (Max)	12		
GLB**	320–560	208–1296	576–4992
Flipflops	484–772	282–1500	720–5392
I/O Pins	168–216	78–208	150–406
Package	PLCC/QFP/PGA	PLCC/QFP/PGA	PLCC/QFP/PGA
Other Features	Use single 5V ISP	EPF8282V uses 3.3V	Included EAB* can implement DSP, CP, MICRO CONTROL, RAM and FIFO.

*EAB: Embedded Array Block.

Supplier	AMD					
Technology	E ² CMOS	E ² CMOS	E ² CMOS	E ² CMOS	E ² CMOS	E ² CMOS
Family	MACH1 MACH110	P+ MACH111	MACH1 MACH120	MACH1 MACH130	P+ MACH131	MACH2 MACH210
Gates	900	900	1200	1800	1800	1800
Speed (MHz)	77	143	77	66	133	133
T _{pd} (Max)	12	5	12	15	7.5	7.5
GLB**	32	32	48	64	64	64
Flipflops	32	32	48	64	64	64
I/O Pins	38	38	56	70	70	38
Package	44 Pins PLCC	44-Pins PLCC /TQFP	68 Pins PLCC	84 Pins PLCC	84 Pins PLCC	44 Pins PLCC
Other Features						

Note: Delay time of AMD element is predictable.



Supplier	AMD					
Technology	E ² CMOS	E ² CMOS	E ² CMOS	E ² CMOS	E ² CMOS	E ² CMOS
Family	MACH2 MACH220	P+ MACH211	P+ MACH221	MACH2 MACH230	P+ MACH231	MACH2 MACH215
Gates	2400	1800	2400	3600	3600	1500
Speed (Mhz)	77	133	133	66	133	77
T _{pd} (Max)	12	7.5	7.5	15	7.5	12
GLB**	96	64	96	128	128	64
Flipflops	96	64	96	128	128	64
I/O Pins	58	38	56	70	70	38
Package	68 Pins PLCC	44-Pins PLCC /TQFP	68 Pins PLCC	84 Pins PLCC	84 Pins PLCC	44 Pins PLCC
Other Features						

Supplier	AMD					
Technology	E ² CMOS	E ² CMOS	E ² CMOS	E ² CMOS	E ² CMOS	E ² CMOS
Family	MACH3 MACH355	MACH4 MACH435	MACH4 MACH445	MACH4 MACH465	MACH5 V+ M5-128	MACH5 V+ M5-192
Gates	3500	5000	5000	10000	5000	7500
Speed (MHz)	47.6	83.3	83.3	83.3	125	125
T _{pd} (Max)	15	12	712	12	7.5	7.5
GLB**	96	128	128	256	128	192
Flipflops	96	192	192	384	128	192
I/O Pins	102	70	102	146	68, 104, 120	Same as left column+160
Package	144 Pins PPQF	84 Pins PLCC	100 Pins PQFP	208 Pins PQFP	PQFP/ TQFP	44 Pins PLCC
Other Features						



Supplier	AMD			
Technology	E ² CMOS	E ² CMOS	E ² CMOS	E ² CMOS
Family	MACH5 V+ M5-256	MACH5 V+ M5-320	MACH5 V+ M5-384	MACH5 V+ M5-512
Gates	10000	12000	15000	20000
Speed (MHz)	125	125	125	125
T _{pd} (Max)	7.5	7.5	7.5	7.5
GLB**	256	320	384	512
Flipflops	256	320	384	512
I/O Pins	68, 104, 120, 160	120, 160, 184, 192	120, 160, 184, 192	120, 160, 184, 192, 256
Package	PQFP/TQFP	PQFP/BGA	PQFP/BGA	PQFP/BGA
Other Features				

Supplier	LATTICE				
Technology	E ² CMOS	E ² CMOS	E ² CMOS	E ² CMOS	E ² CMOS
Family	1000	2000	3000	6000	PLSI & SPLSI
Gates	8000	6000	14000	20000	1000~11000
Speed (MHz)	125	154	100	70	50~150
T _{pd} (Max)	7.5	5.5	10	15	5.5
GLB**	192	128	320	192	32~256
Flipflops	288	128	480	416	32~384
I/O Pins	108	136	108	169	34~128
Package	PLCC, JLCC, PQFP, CPGA, TQFP	PLCC, TQFP, PQFP, MQUAD	MQUAD	MQFP	PLCC, MQPF, TQPF, CPGA, PQFP
Other Features	1.ISP 2.Reconfigurable	1.ISP 2.Reconfigurable	1.ISP 2.Reconfigurable	1.ISP 2.Reconfigurable 3. Built-in SRAM	1.ISP 2.Daisy chain 3.Download



A.2 ALTERA's CPLD Devices

Till now(1996) , Seven family series of CPLD are provided by ALTERA:

1. CLASSIC
2. MAX5000
3. MAX7000
4. FLASH LOGIC
5. FLEX8000
6. MAX9000
7. FLEX10K

In the CLPD logic world, CPLD almost produced by duplicating unitary cell. At the same series, bit's capacity size depends on the duplicated count. Thus once comprehensive of the smallest element, the whole series elements could roughly in control. Nevertheless, it's probably not enough for a designer merely knowing the smallest bit; the following items are required for user to know:

1. Manufacture technique: It refers to how to deal with memorial contacts with either on or off way. Presently, applicable techniques included EPROM, EEPROM, FLASH, SRAM and Anti-Fuse. Different manufacture technique would have different process steps. For example, EEPROM user can clean AMD originally design by electric method; but if configuration bit manufactured by SRAM, which attribute belonged to volatility memory, once turning off power, the content would no more exist. Thus the user should do more steps to take care this feature, that means the user shall re-download configuration while starting up computer.



2. Capacity: It's no doubt that chip capacity is important. Over small chip couldn't layout all digital circuits into one chip. On the other hand, use chip with exceeding capacity would cause too many unused circuits and raise capitalized cost.
3. Fundamental structure: Include basic unit mentioned above and router source for connecting internal element.
4. Specific function: Except structural specification mentioned above, some chip could change timing delay and electric specification by setting, and things like that could be as specific function.

Each series will be discussed with the following four items:

❖ Classic series

1. Manufacture Techniques: Classics family mainly adopt EPROM which divided into two packages: **Weed-out windows** and **no weed-out windows**. It is an One Time Program (OTP) device if it used EPROM but no weed-out windows to clear data by UV radiation.
2. Capacity: Gate count from 150 to 900, but for macrocell it could be 8 to 16. The characteristics of chip of Classic series are listed in Table A.1.
3. Fundamental structure: In this family, different chip leads to different structure. Figure A.1 is structure diagram shown big structure is composed by several smaller compositions.
4. Specific function : This family possess security bit for programming.



Table A.1 The device characteristics of Classic family

Chip Feature	EP22V10 EP22V10E	EP610 EP610T	EP610I	EP910 EP910T	EP910I	EP1810 EP1810T
Available gates	400	600	600	900	1800	1800
Usable gates	200	300	300	450	450	900
Macrocells	10	16	16	24	24	48
Maximum user I/O pins	22	20	20	36	36	64
t_{PD} (ns)	7.5	15(35)*	10	30	12	20 (45)
f_{CNT} (MHz)	111.1	83.3 (28.6)*	100	33.3	100	50 (22.2)

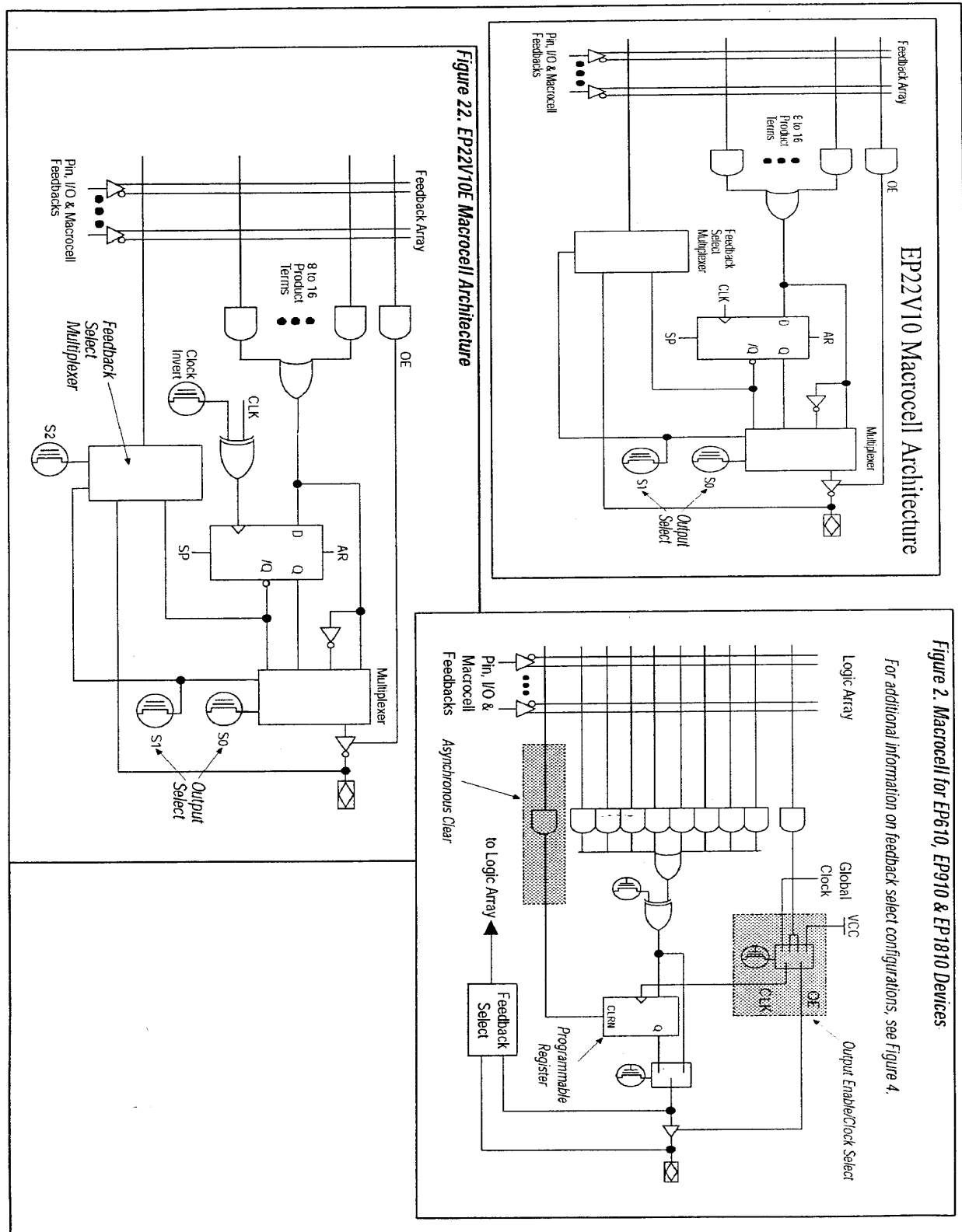


Figure A.1 Architecture of Classic family



❖ MAX5000 family

1. Manufacture techniques : Same as Classic family.
2. Capacity: Gate count is from 600 to 3750, but for macrocell, it could be 16 to 192. The device characteristics of this family are listed in Table A.2.
3. Fundamental structure : Figure A.2 is architecture of MAX5000 family. In this family, gathering 16 basic elements is called Logic Array Block (LAB). All macrocell of same block will share all input lines and a P-Term block.
4. Specific function : Possess security bit for scheming as well.

Table A.2 The device characteristics of MAX5000 family

Feature	CHIP				
	EPM5032	EPM5064	EPM5128	EPM5130	EPM5192
Available gates	1200	2500	5000	5000	7500
Usable gates	600	1250	2500	2500	3750
Macrocells	32	64	128	128	192
LABs	1	4	8	8	12
Expanders	64	128	256	256	384
Routing	GLOBAL	PIA	PIA	PIA	PIA
Maximum user I/O pins	24	36	60	68, 84	72
T_{PD} (ns)	10	15	15	15	15
T_{ASU} (ns)	3	5	5	5	5
T_{CO} (ns)	6	8	8	8	8
f_{CNT} (MHz)	125	83.3	83.3	83.3	83.3

Figure 1. MAX 5000 Architecture

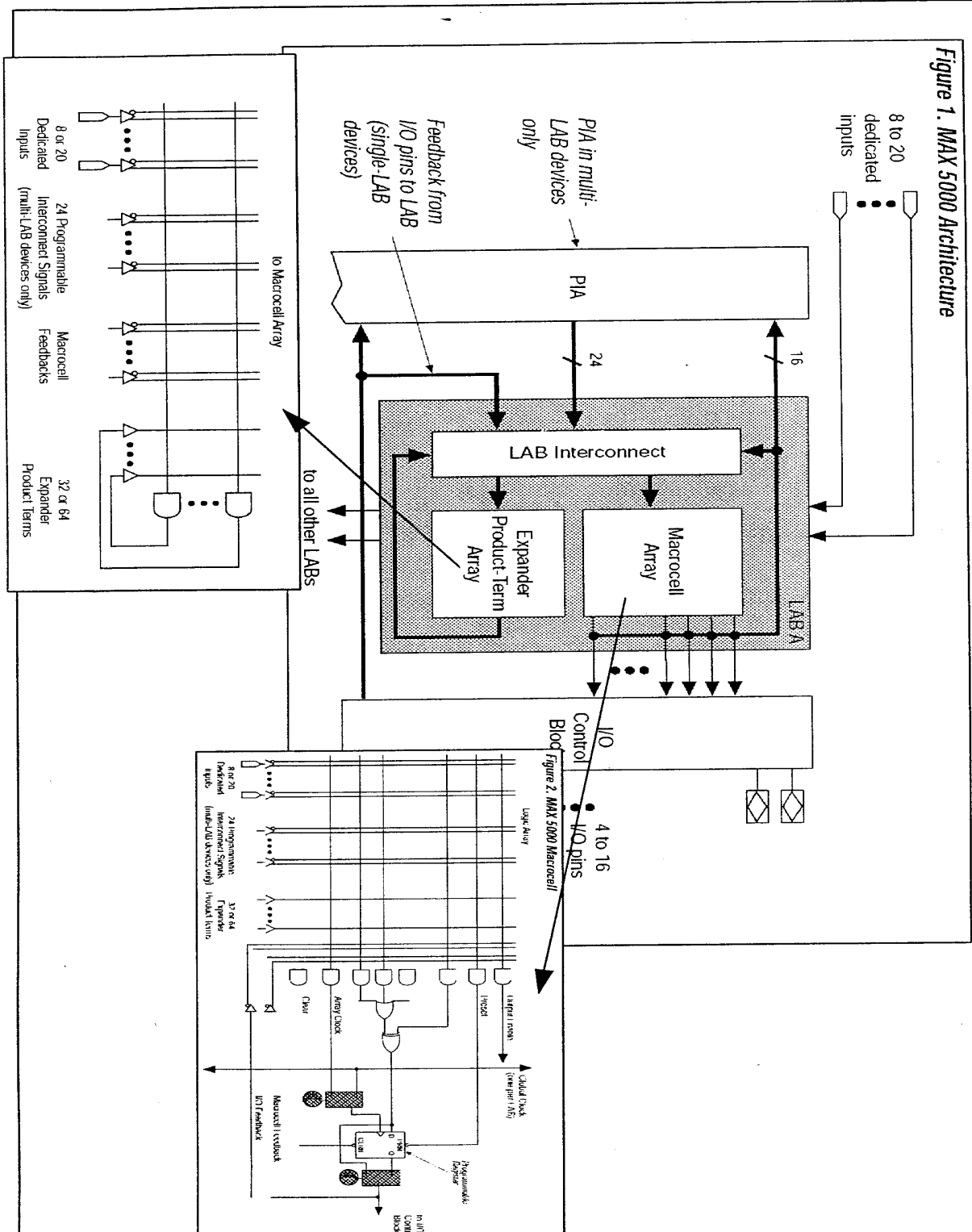


Figure A.2 Architecture of MAX5000 family



❖ MAX7000 family

1. Manufacture techniques: MAX7000 family manufactured in EEPROM manner.
2. Capacity: Gate count is from 600 to 5,000, but imacrocells could be from 32 to 256. Table A.3 lists device characteristics of this family.
3. Fundamental structure: In MAX7000 family, it could subdivide to three subsystems: MAX7000, MAX7000E and MAX7000S. Figure A.3 to Figure A.5 indicate the architecture of them. Basically, their structure is similar, and the mainly difference is MAX7000E owned 5V of ISP (In System Programmability), having more four OE control lines than MAX7000, and capable to be an open collect output.
4. Specific Function: All device of this family possess security bits for programming, and every macrocell could be respectively controlled as turbo mode and non-turbo mode. Besides, of IO structure, MAX7000E and MAX7000S have slew-rate control of signal output.

TableA.3 The characteristics of chips of MAX7000 series

Feature	CHIP							
	EPM703	EPM703	EPM706	EPM709	EPM712	EPM716	EPM719	EPM725
Available gates	1,200	1,200	2,500	3,600	5,000	6,400	7,500	10,000
Usable gates	600	600	1,250	1,800	2,500	3,200	3,750	5,000
Macrocells	32	32	64	96	128	160	192	256
Maximum user I/O Pins	36	36	68	76	100	104	124	164
t_{PD} (ns)	5	12	6	6	7.5	7.5	10	10
t_{SU} (ns)	4	10	5	5	6	6	7	7
t_{FSU} (ns)	—	—	—	—	3	3	3	3
t_{COI} (ns)	3.5	7	4	4	4.5	4.5	5	5
f_{CNT} (MHz)	178.6	90.9	151.5	151.5	125	125	100	100

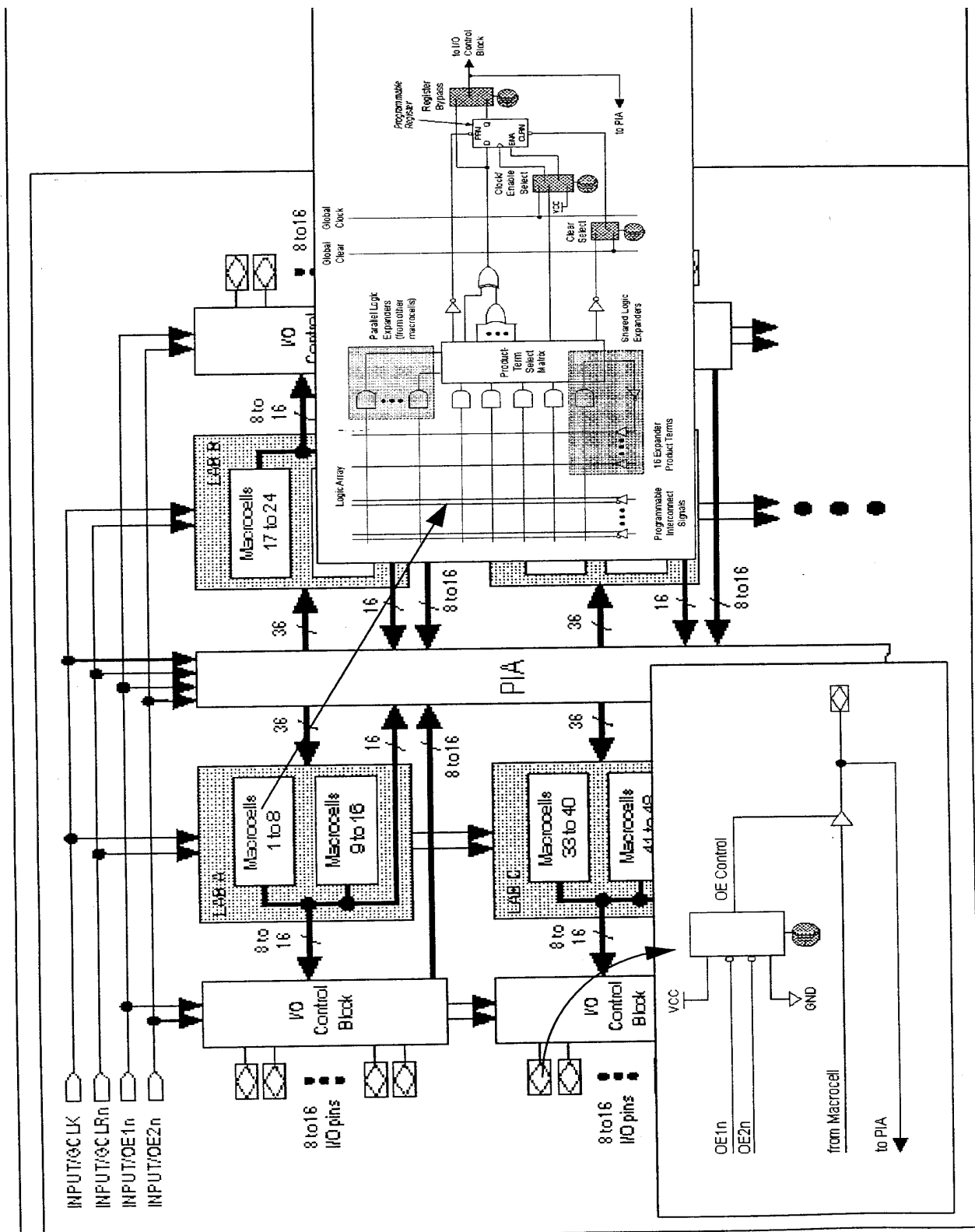


Figure A.3 Architecture of MAX7000 family

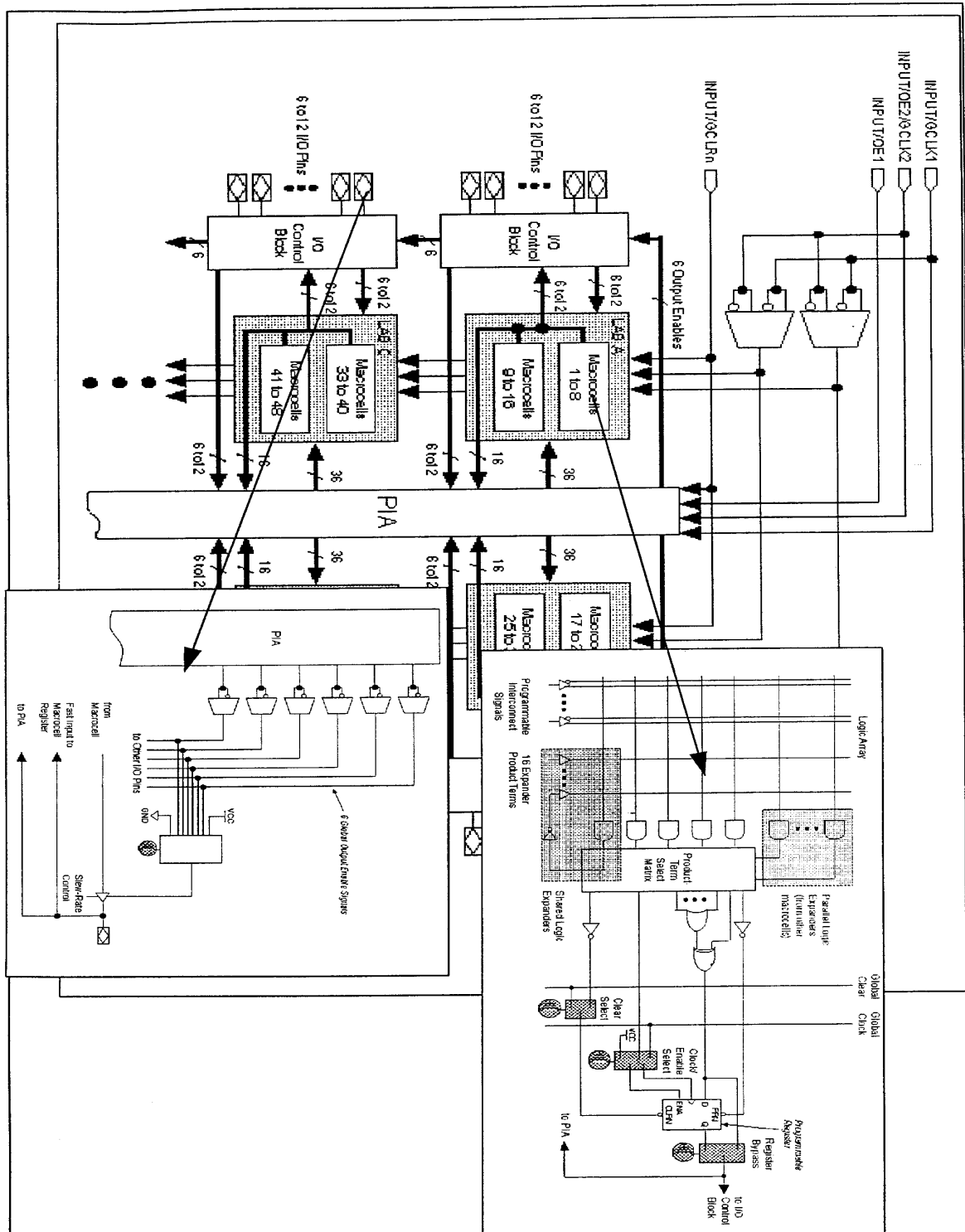


Figure A.4 Architecture of MAX7000E family

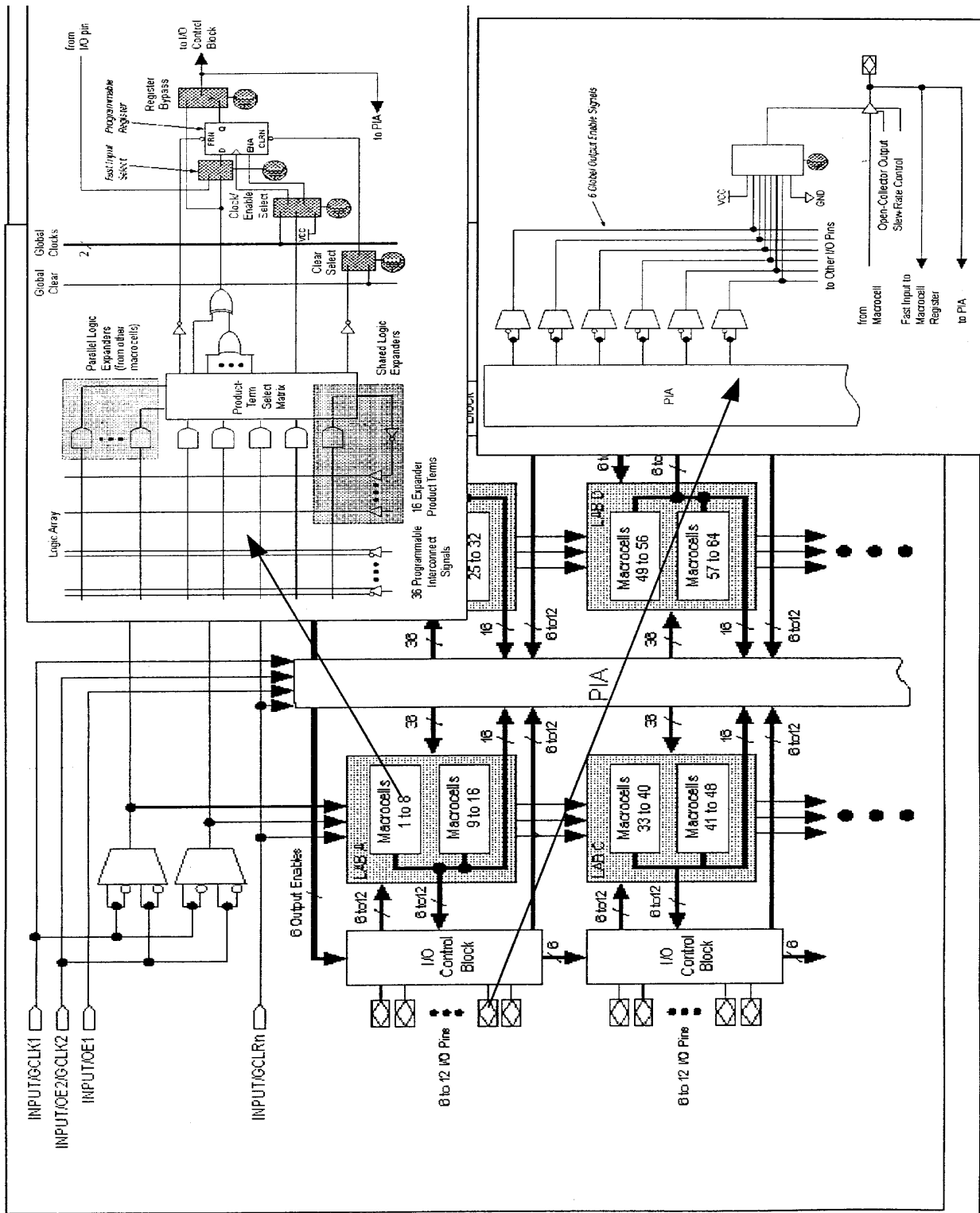


Figure A.5 Architecture of MAX7000S series



❖ Flash Logic family

1. Manufacture techniques: In this family, each device simultaneously adopt two techniques, it could be SRAM and EPROM or SRAM and FLASH to control the same configuration bit. In other words, people download configuration data directly to SRAM while the circuit in developing phase. After developed, write the configuration data into nonvolatile memory EPROM or FLASH. By this way, elasticity of developing phase and practice of developed time could be both considered.
2. Capacity: Gate count from 800 to 3,200, but for macrocells, it could be 40 to 160. Table A.4 lists device characteristics of this family, in which 740 and 780 manufactured by SRAM and EPROM, but 880 and 8160 are made by SRAM and FLASH.
3. Fundamental structure: Figure A.6 is the architecture of Flash Logic family. The most distinctive part of this family is every LAB could evolve into 2 kinds via setting. One is P-Term of MAX7000 family, and another is 128×10 SRAM memory. This alternative option is based on a LAB, and everyone could plan out diverse types.
4. Specific function: Peculiar function of this family mainly is on I/O parts. Every I/O related to LAB, which in control individually, could be 5V I/O or 3.3 V I/O. Beside, every I/O could work on open drain or pull high register independently. This family also includes security bit for programming.



Table A.4 The characteristics of chips of Flash logic family

Feature	CHIP			
	EPX740	EPX780	EPX880	EPX8160
Available gates	1,600	3,200	3,200	6,400
Usable gates	800	1,600	1,600	3,200
Total SRAM bits	5,120	10,240	10,240	20,480
Macrocells	40	80	80	160
Maximum user I/O pins	44-P PLCC (32) 68-P PLCC (52)	84-P PLCC (62) 132-P QFP (104)	84-P PLCC (62) 160-P QFP (104)	208-P QFP (172)
t_{PD} (ns)	10	10	10	10
t_{CO} (ns)	6	6	6	6
f_{CNT} (MHz)	83.3	83.3	80	80

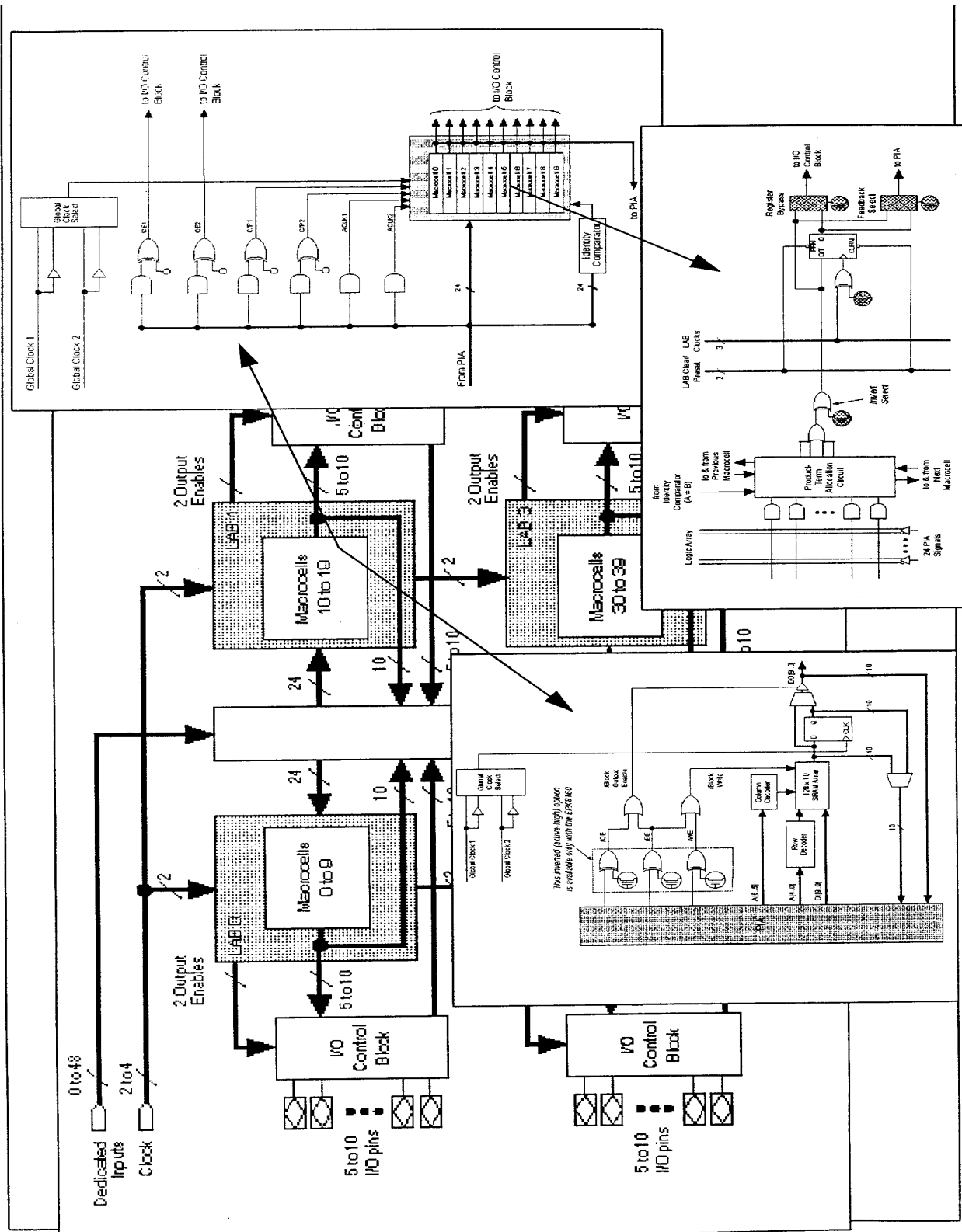


Figure A.6 Architecture of Flash logic family



❖ FLEX8000 family

1. Manufacture techniques: FLEX8000 is manufactured by using SRAM technique. Due to FLEX8000 merely could be produced by volatility memorie that means if power turn off and all bits configuration data would vanish at the same time. Therefore, it's necessary for each device shall be configured again while power turn on. We call this operation as “in circuit configuration” (ICR).
2. Capacity: Gate count is from 2,500 to 16,000 and LE element could be 208 to 1,296. Table A.5 lists every device characteristic in this family.
3. Fundamental structure: Figure A.7 is the fundamental structure of FLEX8000 family. It combined only with several blocks which shown totally different with previous family structure. In fact, this family is no more adopt P-Term for logical configuration, but use memory for storing Luck Up Table (LUT) to implement logic function. By which, all 4 input logical configurations can present with 16-bit memory. Owing to structure difference, this logical unit of this family renames as LE (Logical Element). Structurally, LUT add two blocks, CARRY and CASCADE, on back for concatenating information between LE. In that case, for one thing it could save external signal input resource, and for another it could avoid delaying time. In wholly framework, FLEX8000 adopt connection bus of three-dimension, which includes horizontal and vertical connection among LE blocks as well as the internal logic block connection. All I/O signals are output through bus, and every I/O embedded one register for signal storage.
4. Specific function: Every LE of this family can be controlled in turbo mode and non-turbo mode respectively. Besides, IO structure has slew-rate control.



Table A.5 The characteristics of chips of FLEX8000 family

Feature	CHIP					
	EPF8282 EPF8282V EPF8282A EPF8282AV	EPF8452 EPF8254A	EPF8636A	EPF8820 EPF8820A	EPF81188 EPF81188A	EPF81500 EPF81500A
Available gates	5,000	8,000	12,000	16,000	24,000	32,000
Usable gates	2,500	4,000	6,000	8,000	12,000	16,000
Flipflops	282	452	636	820	1,188	1,500
Logic element	208	336	504	672	1,008	1,296
Maximum user I/O pins	78	120	136	152	184	208
JTAG BST	Yes	No	Yes	Yes	No	Yes
Package	84P PLCC 100P TQFP	84P PLCC 160 PQFP 160 PGA 100 TQFP	84P PLCC 160P PQFP 192P PGA 208P RQFP	160P PQFP 192P PGA 208P RQFP 225P BGA	208P PQFP 232P PGA 240P RQFP	240P RQFP 280P PGA 304P RQFP

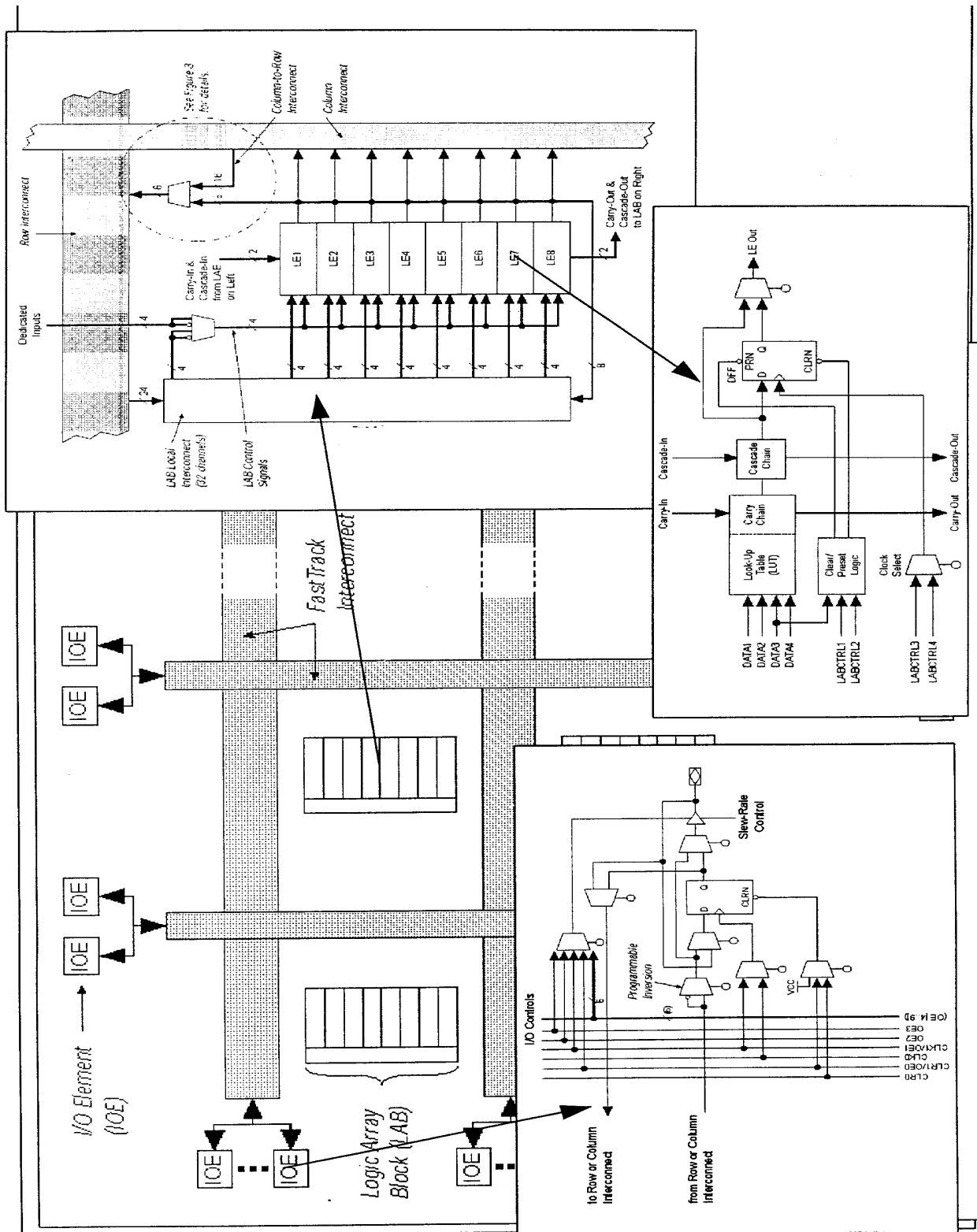


Figure A.7 Architecture of FLEX 8000 family

Since FLEX8000 family is helpful for learning, further introduction to the internal structure is needed. FLEX 8000 configured by Logic Area Blocks (LAB), I/O element and interconnect FastTrack. Every LAB block consists of eight LE elements. Like Figure A.8, LE is the smallest logic element of FLEX8000 family. Figure A.9 shows that each LE composed by one 4-input (LUT), one programmable register, one carry chain and one cascade chain. LUT could produce circuit and then come into quad variable function quickly. The programmable register can program as D-Type, T-Type, JK Type or SR Type, which input signals like clock, clear and preset are driven by exclusive input pin, general use I/O input pin or by any internal logic. This LE provides compose capability of combinational logic and sequential logic circuit. For combinational logic circuit, LUT can skip over directly from programmable register and then output from LE.

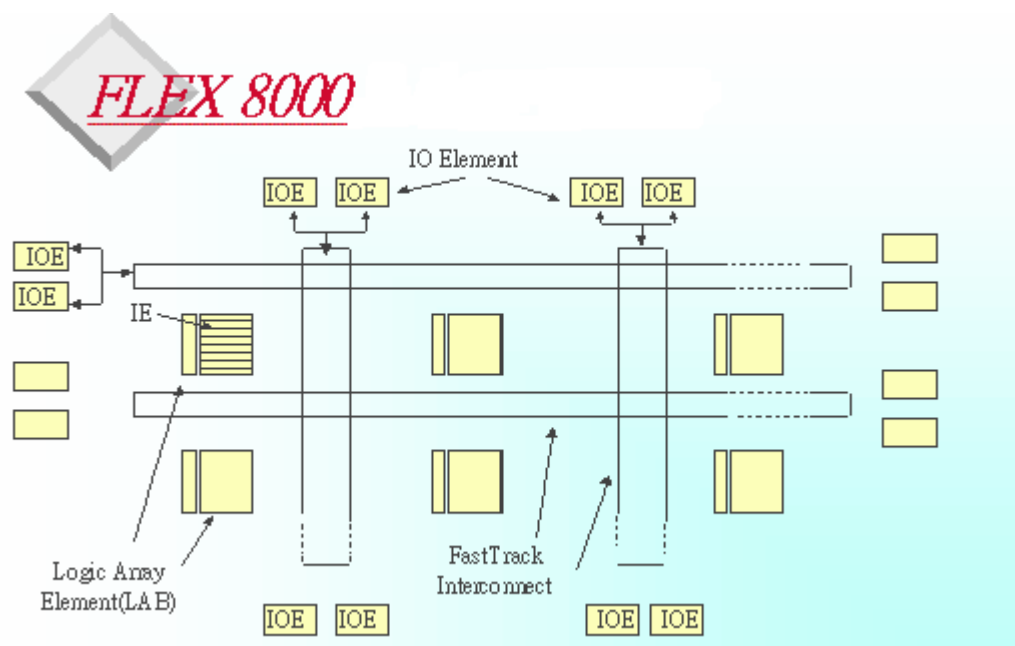


Figure A.8 FLEX8000 Device Block Diagram

FLEX8000 has two specific high-speed data links: carry-in chain and cascade chain for connecting adjacent without via local interconnection. Figure A.10 shows that



high-speed counter and adder can be produced by carry chain. Carry chain can produce high fan-in with lower timing delay circuit, e.g. OR cascade application in Figure A.11 and an AND application in Figure A.12. However, overuse carry chain and cascade chain would decrease applicable wire resource for other LE use. So, we suggest user to apply them when in high-speed design requirement.

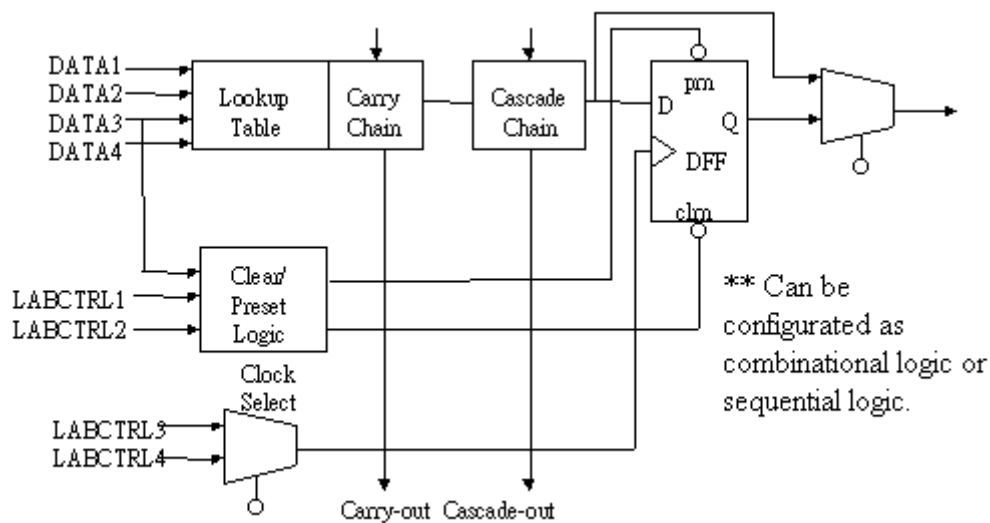


Figure A.9 FLEX8000 Logic Element

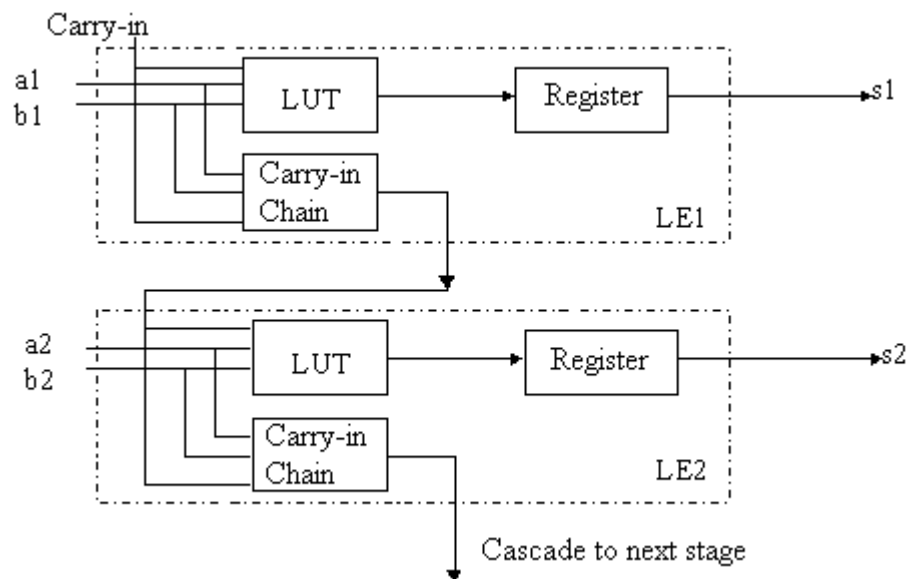


Figure A.10 FLEX8000 Carry Chain Operation

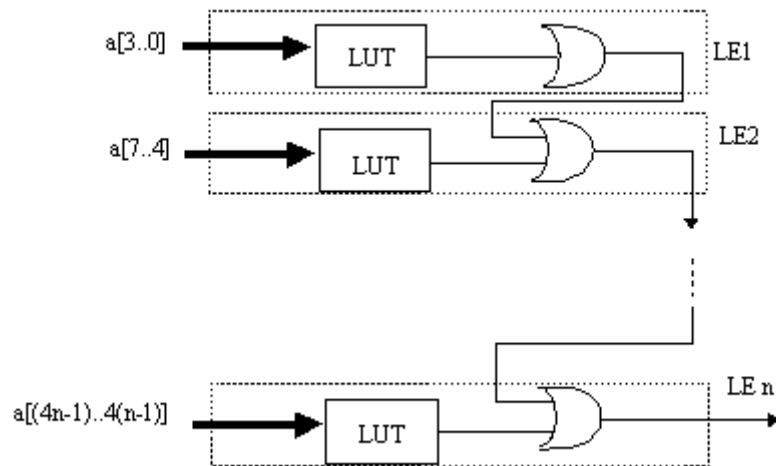


Figure A.11 FLEX8000 OR-cascade Chain Operation

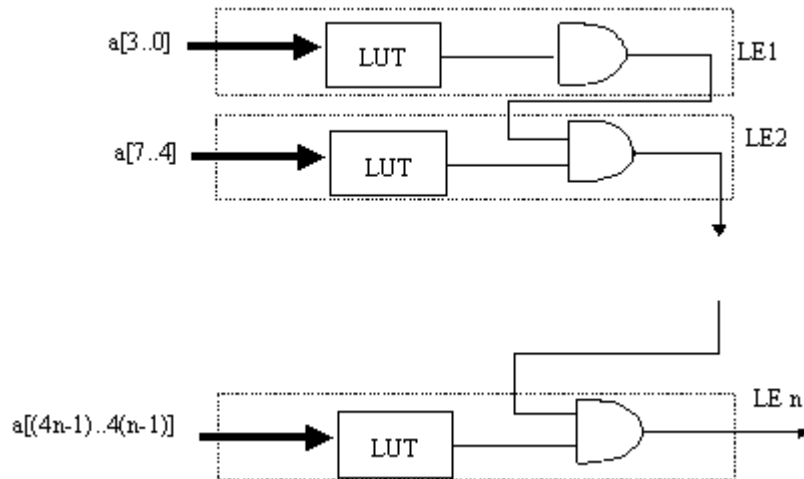


Figure A.12 FLEX8000 AND-cascade Chain Operation

There are four operating modes of LE element in FLEX8000 family. Those are Normal mode, Arithmetic Mode, Up/Down Count Mode and Clearable Count mode, shown in Figure A.13 ~ Figure A.16. Each mode has ten input signals, which are Clock, Clear, Preset, four signals come from local interconnect of LAB, one feedback from programmable register, Cascade-in and Carry-in. Different mode uses different LE resource. Normal mode has Cascade-in chain that suits for general circuit or decoder with mass inputs. D1, D2, D3 and D4 come from local interconnect of LAB block. D3

and Carry-in are input to 1-of-2 multiplexer, which output then send to 4-input LUT. The output of AND logic, output of LUT AND operates with Cascade-in from previous step, can be as 1-of-2 multiplexer input, as D input of programmable register, or as Cascade-out. Therefore, the output of LE can be via Flip-flop or not.

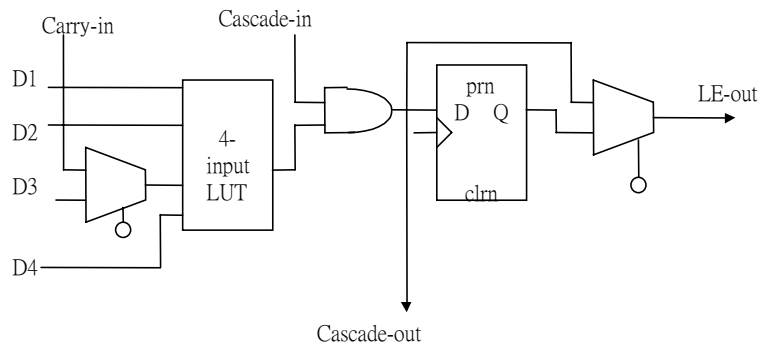


Figure A.13 FLEX8000 LE Normal Mode

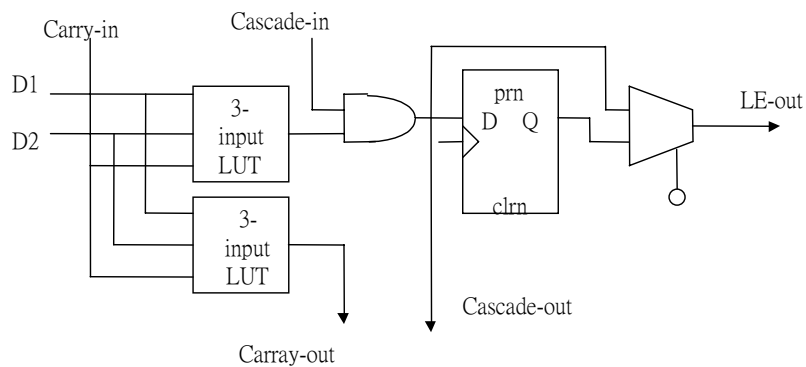


Figure A.14 FLEX8000 LE Arithmetic Mode

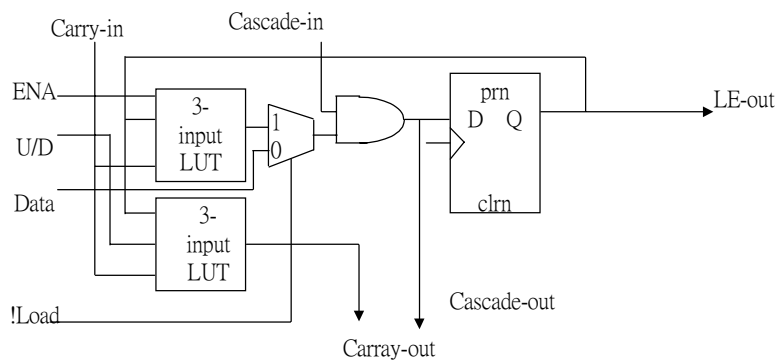


Figure A.15 FLEX8000 LE Up/Down Count Mode

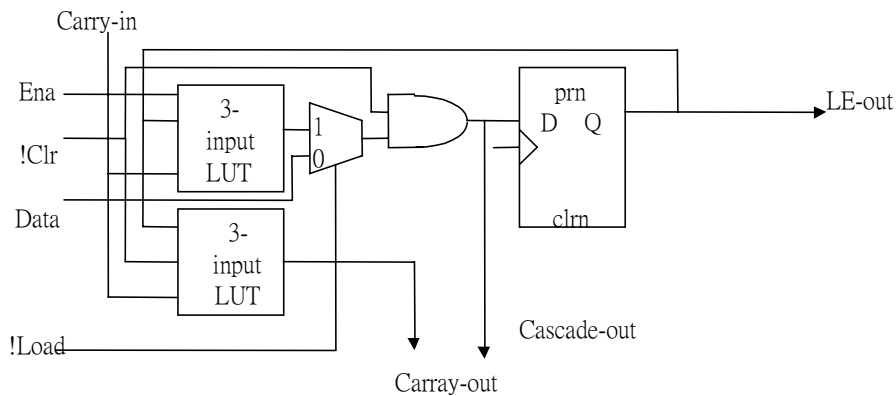


Figure A.16 FLEX8000 LE Clearable Counter Mode

Two 3-bit parallel input LUT of arithmetic mode is special suited to produce high-speed adder, high-speed accumulator and high-speed comparator. Upper/down counter mode provides counting enable, synchronous up/down control and data load. No exception, Up/down counter mode also applies 3-bit parallel input LUT, one result in count value and another produces high-speed carry. One 1-of-2 Multiplexer provides synchronous data load, and asynchronous data load can completed merely with Clear and Preset signals without using LUT resource. Clearable counter mode is a little bit similar to up/down counter mode, in addition to synchronous clear control replace synchronous up/down control, 1-of-2 Multiplexer's output AND with synchronous clear signal then output.

Figure A.17 shows that the Preset of programmable register that is controlled by d3 and LABCTRL1 and Figure A.18 shows that the Clear is controlled by d3, LABCTR1 and LABCTL2.

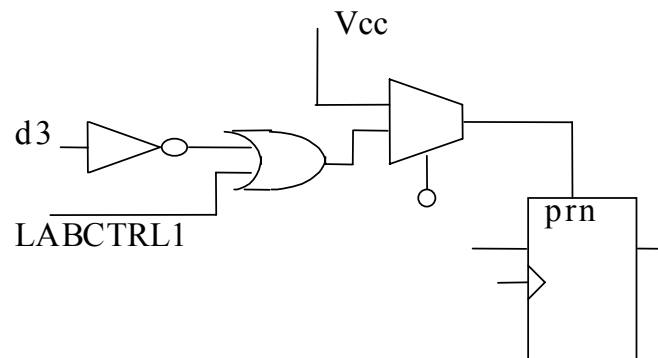


Figure A.17 Asynchronous preset of FLEX8000 LE

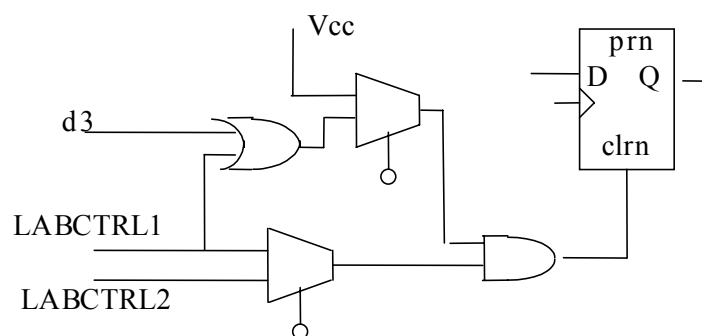


Figure A.18 Asynchronous clear of FLEX8000 LE

Operation mode of FLEX8000: For FLEX8000 adopted SRAM to save configuration data, which needed re-download configuration data when plug-in, and this process called configuration. When configuration completed, FLEX8000 reset registers and enable I/O pin then works like logic circuit. This reset process called initialization. From configuration to initialization, this stage named command mode, and then enter general on-line work called user mode. SRAM allow configuring FLEX8000 by downloading on-lined new configuration data. For this real-time configuration can reinforce device enter configuration and initialization (namely command mode) processes just by dedicated pin, then back to user mode. The whole process only took less than 100ms. When device plug-in, it could be configured either in automatically or controlled by external circuit. FLEX8000 can use embedded oscillator or external clock signal of device to do initialize. Dedicate pins are used to



control when device enter configuration and initializing (namely command mode) processes.

Configuration schemes of FLEX8000: Like Table A.6, FLEX 8000 provided six kinds of configuration schemes. Under active schemes, FLEX8000 dominated all configuration process and internal oscillator (typical value is 2~6 MHz) offer external synchronizing clock and control signal. AS mode uses ALTERA configuration device (serial type) to save configuration data, and APU & APD mode adopts Parallel EPROM, such as 2732 or 2764, to save configuration data. When FLEX8000 in passive mode, all configuration process controlled by external circuit and provided clock from outside. What is called intelligent host referring to microprocessor unit something like controller.

Table A.6 Data Source for Configuration

Configuration Scheme	Acronym	Data Source
Active Serial	AS	ALTERA configuration device
Active parallel up	APU	Parallel configuration device
Active parallel down	APD	Parallel configuration device
Passive serial	PS	Serial data path
Passive parallel synchronous	PPS	Intelligent host
Passive parallel asynchronous	PPA	Intelligent host



❖ MAX9000 family

1. Manufacture techniques: Like MAX7000 family manufactured by EEPROM.
2. Capacity: Gate count is from 6000 to 12000 and macrocell could be 320 to 560.

Table A.7 is device characteristics of MAX9000 family.

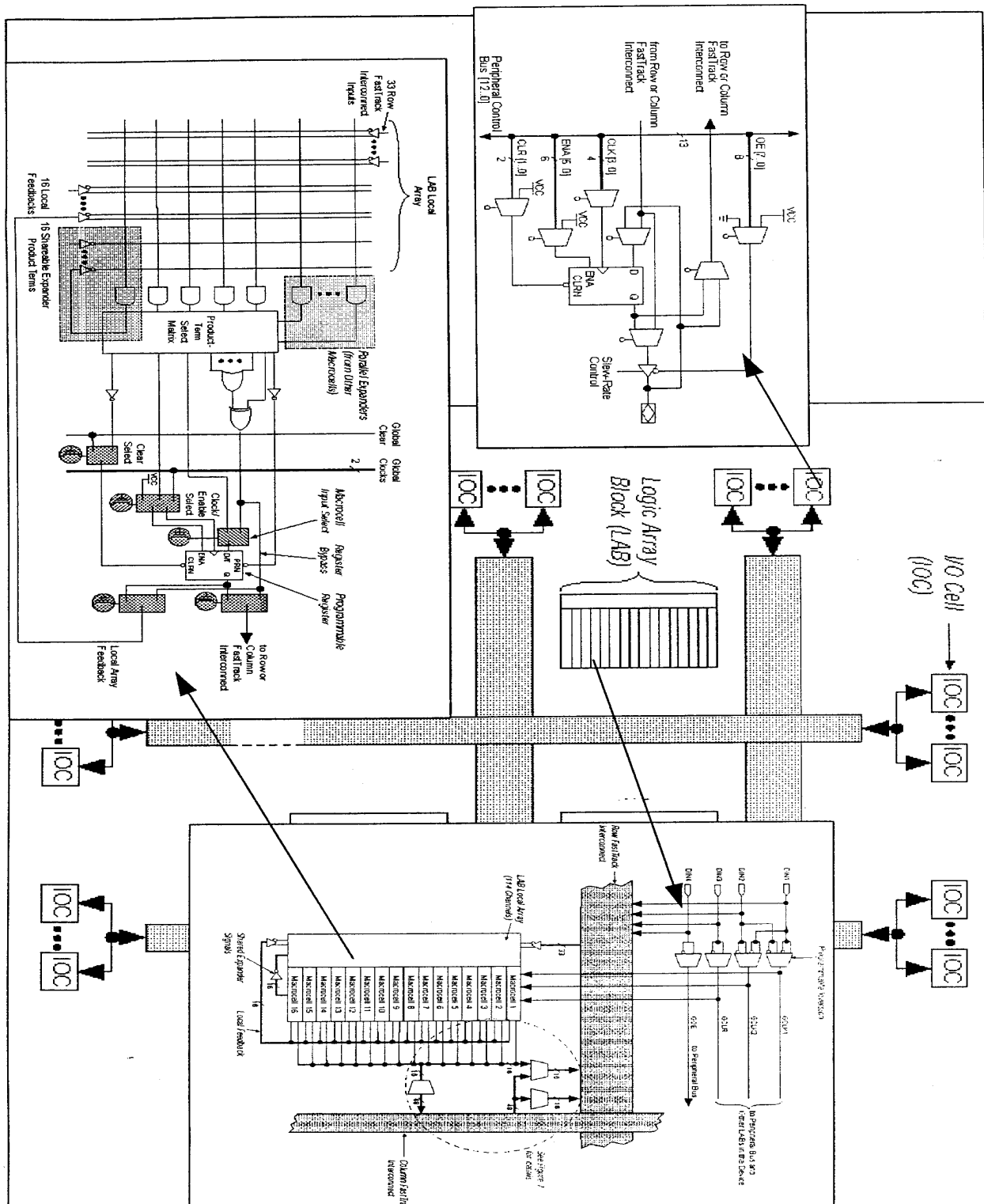
Table A.7 The characteristics of chips of MAX9000

Feature	CHIP			
	EPM9320	EPM9400	EPM9480	EPM9560
Available gates	12000	16000	20000	24000
Usable gates	6000	8000	10000	12000
Flipflops	484	580	676	772
Macrocells	320	400	480	560
Maximum user I/O pins	168	184	200	216
t_{PD1} (ns)	12	12	15	15
t_{FSU} (ns)	3	3	5	5
t_{FCO} (ns)	6	6	7	7
F_{CNT} (Mhz)	125	125	118	118

3. Fundamental structure: We can say MAX9000 family, its structure almost follows MAX7000 family, is the extension of MAX7000 family. Figure A.19 is the architecture of MAX9000 family. Since capacity of MAX9000 family larger than MAX7000 family, the internal bus structure becomes as three dimensions. All I/O have one register and all output from bus (same as FLEX8000). MAX9000 generally parallel to MAX7000 in macrocell, but MAX9000 design each LE with two outputs to promote utility rate of macrocells.



4. Specific function: In this family, security bit embedded in every device. Each macrocell could be controlled into turbo mode and non-turbo mode respectively. Each IO structure also has slew-rate control. Besides, like MAX7000S family, this family provides 5V ISP.





❖ FLEX10K family

1. Manufacture technique: FLEX10K is manufactured by SRAM (same as FLEX8000) and ICR action is needed as well.
2. Capacity: Typical gate count is from 10K to 100K and LE count is 576 to 4,992. The embedded RAM block size of FLEX10K is from 6,144 bits to 24,576 bites. Table A.8 is device characteristics of this family.
3. Fundamental structure: In wholly structure, FLEX10K basically is the extension of FLEX8000. Except for Embedded Array Block (EAB), its structure different from traditional programmable logic. Briefly say, the detached RAM making connection with traditional logic block by using internal bus. By this way, general circuit could be completed in traditional LE. Besides, even a great quantity of memory and registers are demanded, FLEX10K still can finish work in Embedded Array Block (EAB) without wasting lot of LE and signal connection resource those are used in traditional logic block. EAB has more function than memory. At partial of arithmetic circuit, memory mapping is more workable than logic configuration. Multiplier is the good example. However, hundreds of ns are required for logic configuration of multiplier to compute result, but memory mapping manner only requires several tens of ns to access memory. Figure A.20a and Figure A.20b are the architecture of FLEX10K family. Most structure of FLEX10k similar to FLEX8000, but each I/O pin of FLEX10K has individual OE control, open drain and slew-rate control in I/O structure.
4. Specific function: Turbo-mode and non-turbo mode respectively controlled by every LE of this family. Each IO structure also possesses input slew-rate control.



Table A.8 The characteristics of chips of FLEX10K family

Feature	CHIP						
	EPF10K10	EPF10K20	EPF10K30	EPF10K40	EPF10K50	EPF10K70	EPF10K100
Typical gates	10,000	20,000	30,000	40,000	50,000	70,000	100,000
Usable gates	7K~31K	15K~63K	22K~69K	29K~93K	36K~116K	46K~118K	62K~158K
LEs	576	1152	1728	2304	2880	3744	4992
RAM bits	6,144	12,288	12,288	16,384	20,480	18,432	24,576
Flipflops	720	1,344	1,968	2,576	3,184	4,096	5,392
Maximum user I/O pins	150	198	246	278	310	358	406

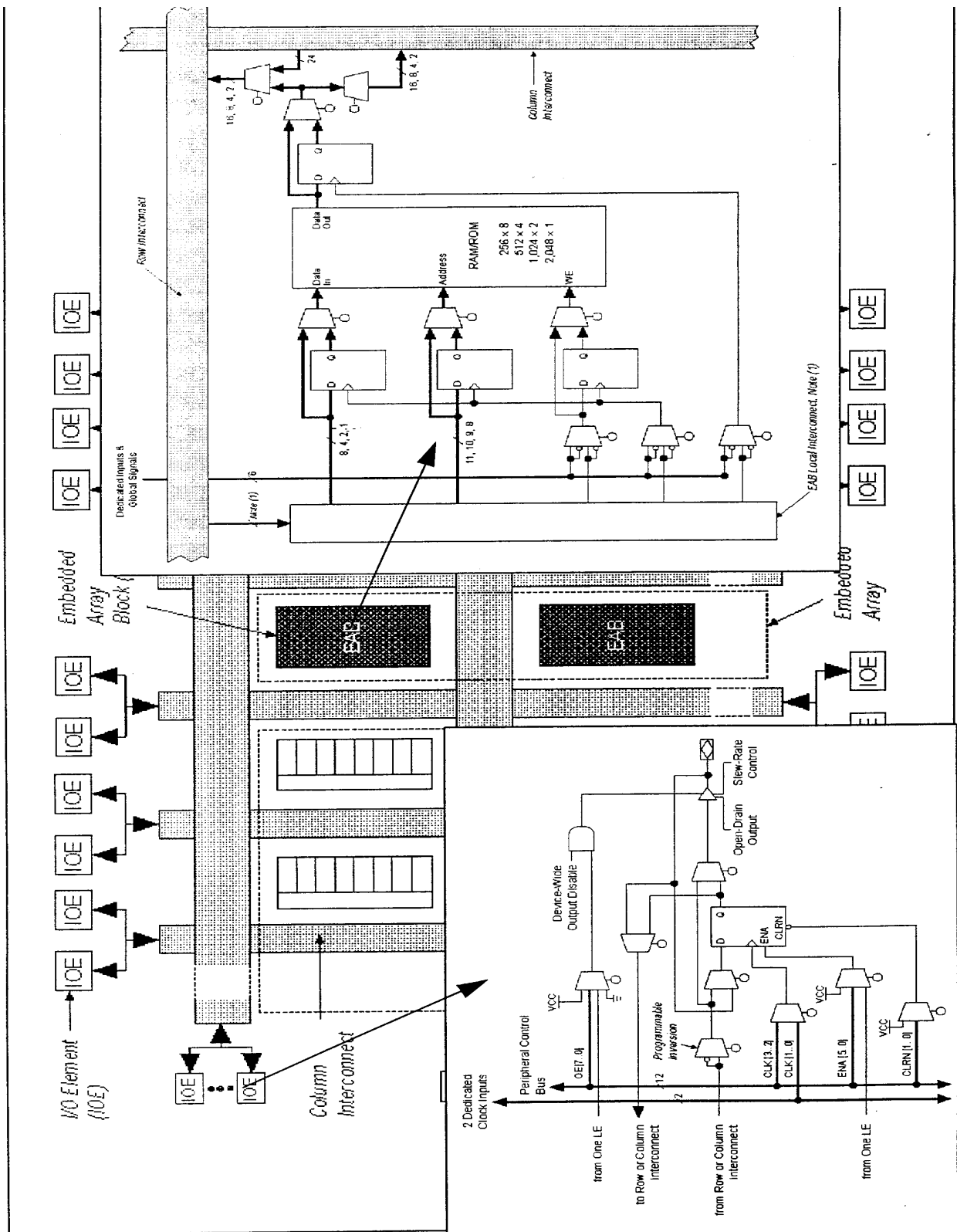


Figure A.20a Architecture of FLEX 10K family (1)

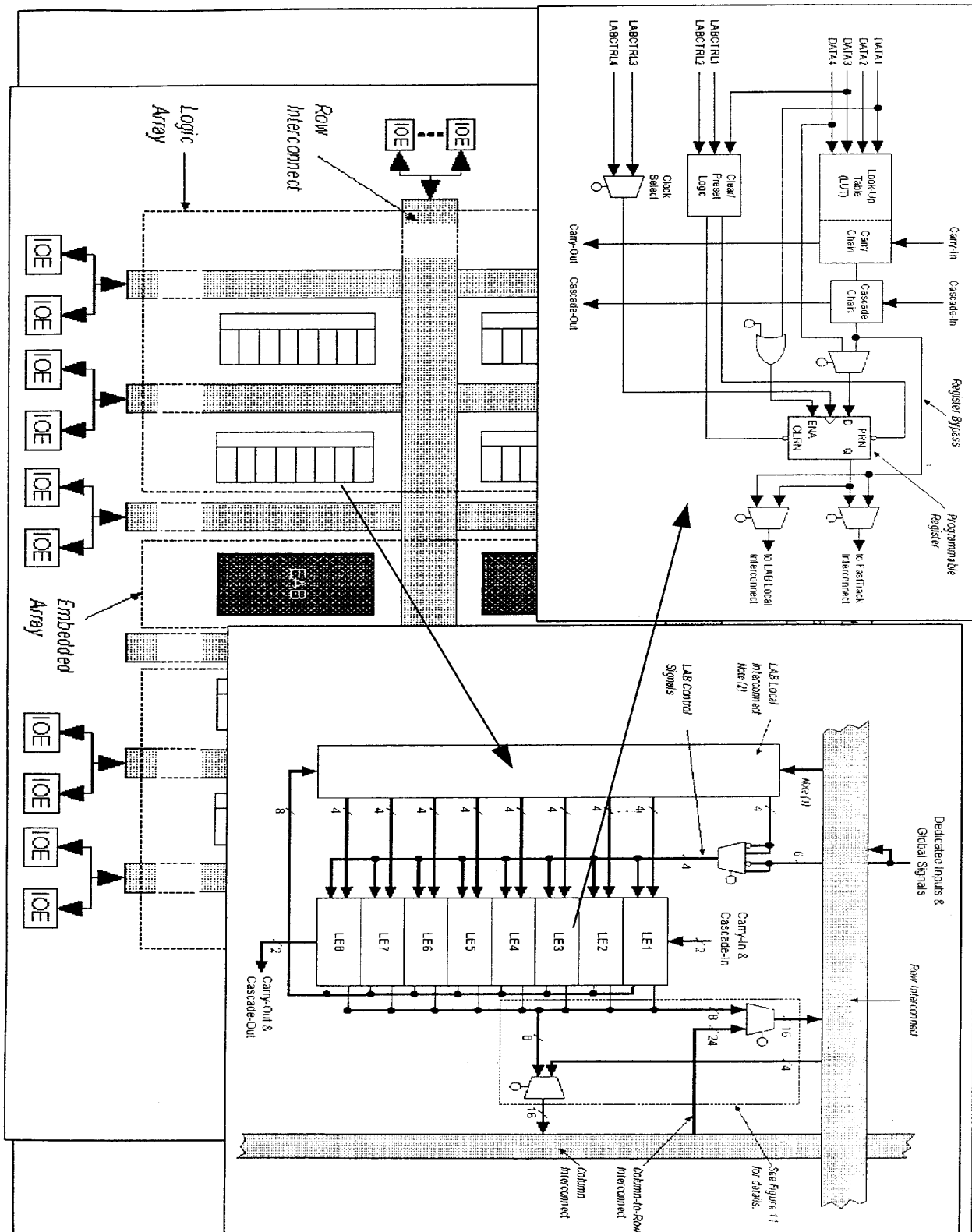


Figure A.20b Architecture of FLEX 10K family (2)

APPENDIX B

The Built-in Resources of MAX+PLUS II





MAX+PLUS II provides more built-in resources to engineers whom are familiar with standard discrete primitives can keep on using. There two categories, Primitives and Macrofunctions, built-in resources are provided. Buffer, Flip-flop/Latch, Input/output and Basic Logic are four resource classes in Primitives. There are 20 categories in Macrofunctions, which include Adder, Frequency Divider, ALU, Latch, Application Specific Function, Multiplier, Buffer, Multiplexer, Comparator, Parity Generator/checker, Rate Multiplier, Counter, Register, Decoder, Shift Register, Digital Filter, Storage Register, EDAC, SSI and Encoder.

MAX+PLUS II has provided furnish built-in resources of standard discrete primitives, for readers of this book (Graphic entry manner) who only need to key in the name of circuit symbol like as AND3, XOR, OR4, 74138, ... Reader can take built-in resources listed below for reference.

B.1 Primitives

B.1.1 Primitive Categories

There are four built-in primitive categories in MAX+PLUS II. Those include buffer, flip-flop and latch, input/output primitives and logic primitives.

B.1.2 Description of Primitives

Primitives categorized as above and listed for description like following:



Buffer Primitives	
Primitive	Function Prototype
LCELL (MCELL)	FUNCTION LCELL (in) RETURNS (out);
SOFT	FUNCTION SOFT (in) RETURNS (out);
CARRY	FUNCTION CARRY (in) RETURNS (out);
CASCADE	FUNCTION CASCADE (in) RETURNS (out);
EXP	FUNCTION EXP (in) RETURNS (out);
GLOBAL (SCLK)	FUNCTION GLOBAL (in) RETURNS (out);
WIRE (GDFs only)	OUT = input IN
TRI	FUNCTION TRI (in, oe) RETURNS (out);

Flip-flop & Latch Primitives	
Primitive	Function Prototype
SRFF	FUNCTION SRFF (S, R, CLK, CLRN, PRN) RETURNS (Q);
SRFFE	FUNCTION SRFFE (S, R, CLK, CLRN, PRN, ENA) RETURNS (Q);
TFF	FUNCTION TFF (T, CLK, CLRN, PRN) RETURNS (Q);
TFFE	FUNCTION TFFE (T, CLK, CLRN, PRN, ENA) RETURNS (Q);



DFF	FUNCTION DFF (D, CLK, CLRN, PRN) RETURNS (Q);
DFFE	FUNCTION DFFE (D, CLK, CLRN, PRN, ENA) RETURNS (Q);
JKFF	FUNCTION JKFF (J, K, CLK, CLRN, PRN) RETURNS (Q);
JKFFE	FUNCTION JKFFE (J, K, CLK, CLRN, PRN, ENA) RETURNS (Q);
LATCH	FUNCTION LATCH (D, ENA) RETURNS (Q);

Input & Output Primitives/Ports	
Primitive	Description
BIDIR or INOUT	AHDL Syntax: io: BIDIR; VHDL Syntax: io: INOUT
INPUT or IN	AHDL Syntax: in1: INPUT; VHDL Syntax: in1: IN
OUTPUT or OUT	AHDL Syntax: out1: OUTPUT; VHDL Syntax: out1: OUT

Logic Primitives (GDFs only)		
Primitive	Description	Name
AND	OUT = logical AND of inputs IN1, IN2,... IN12 = 2, 3, 4, 6, 8, or 12 inputs	AND2, AND3, AND4, AND6, AND8, AND12
NOR	OUT = logical NOR of inputs IN1, IN2,... IN12 = 2, 3, 4, 6, 8, or 12 inputs	NOR2, NOR3, NOR4, NOR6, NOR8, NOR12
NOT	OUT = inverse of input IN = 1 input	NOT



OR	OUT = logical OR of inputs IN1, IN2,... IN12 = 2, 3, 4, 6, 8, or 12 inputs	OR2, OR3, OR4, OR6, OR8, OR12
VCC	Assigns a node or bus to VCC	VCC
GND	Assigns a node or bus to GND	GND
XNOR	OUT = logical exclusive NOR of inputs IN1 and IN2	XNOR
BAND	OUT = logical BAND of inputs IN1, IN2,... IN12 = 2, 3, 4, 6, 8, or 12 inputs	BAND2, BAND3, BAND4, BAND6, BAND8, BAND12
BNAND	OUT = logical BNAND of inputs IN1, IN2,... IN12 = 2, 3, 4, 6, 8, or 12 inputs	BNAND2, BNAND3, BNAND4, BNAND6, BNAND8, BNAND12
BNOR	OUT = logical BNOR of inputs IN1, IN2,... IN12 = 2, 3, 4, 6, 8, or 12 inputs	BNOR2, BNOR3, BNOR4, BNOR6, BNOR8, BNOR12
BOR	OUT = logical BOR of inputs IN1, IN2,... IN12 = 2, 3, 4, 6, 8, or 12 inputs	BOR2, BOR3, BOR4, BOR6, BOR8, BOR12
XOR	OUT = logical exclusive OR of inputs IN1 and IN2	XOR
NAND	OUT = logical NAND of inputs IN1, IN2,... IN12 = 2, 3, 4, 6, 8, or 12 inputs	NAND2, NAND3, NAND4, NAND6, NAND8, NAND12



B.2 Macrofunctions

B.2.1 Macrofunction Categories

Macrofunctions include Adder, Frequency Divider, ALU, Latch, Application_Specific function, Multiplier, Buffer, Multiplexer, Comparator, Parity Generator/checker, Rate Multiplier, Counter, Register, Decoder, Shift Register, Digital Filter, Storage Register, EDAC, SSI Function, Encoder.

B.2.2 Description of Macrofunctions

According to above categories, Description of Macrofunctions are listed as below:

Adders		
Macrofunction	Description	Function Prototype
8FADD	8-Bit Full Adder	FUNCTION 8FADD (cin, a[8..1], b[8..1]) RETURNS (cout, sum[8..1]);
8FADDB	8-Bit Full Adder	FUNCTION 8FADDB (cin, a[8..1], b[8..1]) RETURNS (cout, sum[8..1]);
7480	Gated Full Adder	FUNCTION 7480 (cn0, a1, a2, as, ac, b1, b2, bs, bc) RETURNS (cn1n, sum, sumn);
7482	2-Bit Binary Full Adder	FUNCTION 7482 (a[2..1], b[2..1], c0) RETURNS (sum[2..1], c2);
7483	4-Bit Binary Full Adder with Fast Carry	FUNCTION 7483 (a[4..1], b[4..1], c0) RETURNS (s[4..1], c4);
74183	Dual Carry-Save Full	FUNCTION 74183 (1cn0, 1b, 1a, 2cn0, 2a, 2b)



	Adder	RETURNS (1sum, 1cn1, 2sum, 2cn1);
74283	4-Bit Full Adder with Fast Carry	FUNCTION 74283 (a[4..1], b[4..1], cin) RETURNS (cout, sum[4..1]);
74385	4-Bit Adder/Subtractor with Clear	FUNCTION 74385 (clrn, 1s/an, 1a, 1b, 2s/an, 2a, 2b, 3s/an, 3a, 3b, 4s/an, 4a, 4b, clk) RETURNS (1s, 2s, 3s, 4s);

Truth Table of 8FADD (Full Adder)

Inputs			Outputs	
CIN	A8 .. A1	B8 .. B1	SUM8 .. SUM1	
L	00000000	00000000	00000000	
H	00000000	00000000	00000001	(CIN + A + B = SUM)
	⋮	⋮	⋮	
H	00001001	00011000	00100010	(1 + 9 + 24 = 34)
	⋮	⋮	⋮	

Truth Table of 8FADDB (Full Adder)

Inputs			Outputs	
CIN	A8..A1	B8..B1	SUM8..SUM1	
0	00000000	00000000	00000000	
1	00000000	00000000	00000001	(CIN + A + B = SUM)
	⋮	⋮	⋮	
1	00001001	00011000	00100010	(1 + 9 + 24 = 34)
	⋮	⋮	⋮	

Truth Table of 7480 (Full Adder)

Inputs			Outputs		
CN0	A*	B**	CN1N	SUM	SUMN
L	L	L	H	L	H
L	L	H	H	H	L
L	H	L	H	H	L

L	H	H	L	L	H
H	L	L	H	H	L
H	L	H	L	L	H
H	H	L	L	L	H
H	H	H	L	H	L

* A = /AC + /AS + A1 & A2 (Note : “/” is an inversion symbol).

**** B = /BC + /BS + B1 & B2 ◦**

Truth Table of 7482 (Full Adder)

Input				Output					
				When C0 = L			When C0 = H		
A1	B1	A2	B2	SUM1	SUM2	C2	SUM1	SUM2	C2
L	L	L	L	L	L	L	H	L	L
H	L	L	L	H	L	L	L	H	L
L	H	L	L	H	L	L	L	H	L
H	H	L	L	L	H	L	H	H	L
L	L	H	L	L	H	L	H	H	L
H	L	H	L	H	H	L	L	L	H
L	H	H	L	H	H	L	L	L	H
H	H	H	L	L	L	H	H	L	H
L	L	L	H	L	H	L	H	H	L
H	L	L	H	H	H	L	L	L	H
L	H	L	H	H	H	L	L	L	H
H	H	L	H	L	L	H	H	L	H
L	L	H	H	L	L	H	H	L	H
H	L	H	H	H	L	H	L	H	H
L	H	H	H	H	L	H	L	H	H
H	H	H	H	L	H	H	H	H	H

Truth Table of 7483 (Full Adder)

Input				Output					
				When C0 = L, C2 = L			When C0 = H, C2=H		
A1[A3]	B1[3]	A2[4]	B2[4]	S1[3]	S2[4]	C2[4]	S1[3]	S2[4]	C2[4]
L	L	L	L	L	L	L	H	L	L



H	L	L	L	H	L	L	L	H	L
L	H	L	L	H	L	L	L	H	L
H	H	L	L	L	H	L	H	H	L
L	L	H	L	L	H	L	H	H	L
H	L	H	L	H	H	L	L	L	H
L	H	H	L	H	H	L	L	L	H
H	H	H	L	L	L	H	H	L	H
L	L	L	H	L	H	L	H	H	L
H	L	L	H	H	H	L	L	L	H
L	H	L	H	H	H	L	L	L	H
H	H	L	H	L	L	H	H	L	H
L	L	H	H	L	L	H	H	L	H
H	L	H	H	H	L	H	L	H	H
L	H	H	H	H	L	H	L	H	H
H	H	H	H	L	H	H	H	H	H

Note: The inputs A1, B1, B2 and C0 determined S1 and S2 outputs and internal carry C2. And then, the other inputs C2, A3, B3, A4 and B4 determined S3, S4 and C4 outputs.

Truth Table of 74183 (Full Adder)

Inputs			Outputs	
CN0	B	A	SUM	CN1
L	L	L	L	L
L	L	H	H	L
L	H	L	H	L
L	H	H	L	H
H	L	L	H	L
H	L	H	L	H
H	H	L	L	H
H	H	H	H	H



Truth Table of 74283 (Full Adder)

Inputs				Outputs					
				When CPUT0 = L [When COUT2 =L]			When COUT0 = H [When COUT2 = H]		
A1[A3]	B1[B3]	A2[A4]	B2[B4]	SUM1 [SUM3]	SUM2 [SUM4]	COUT2 [COUT4]	SUM1 [SUM3]	SUM2 [SUM4]	COUT2 [COUT4]
L	L	L	L	L	L	L	H	L	L
H	L	L	L	H	L	L	L	H	L
L	H	L	L	H	L	L	L	H	L
H	H	L	L	L	H	L	H	H	L
L	L	H	L	L	H	L	H	H	L
H	L	H	L	H	H	L	L	L	H
L	H	H	L	H	H	L	L	L	H
H	H	H	L	L	L	H	H	L	H
L	L	L	H	L	H	L	H	H	L
H	L	L	H	H	H	L	L	L	H
L	H	L	H	H	H	L	L	L	H
H	H	L	H	L	L	H	H	L	H
L	L	H	H	L	L	H	H	L	H
H	L	H	H	H	L	H	L	H	H
L	H	H	H	H	L	H	L	H	H
H	H	H	H	L	H	H	H	H	H

Note: The inputs A1, B1, A2, B2 and COUT0 determined SUM1 and SUM2 outputs and internal carry COUT2. And then, the other inputs COUT2, A3, B3, A4 and B4 determined SUM3, SUM4 and COUT4 (COUT) outputs.



Truth Table of 74385 (Adder)

Inputs					Data-in Carry Flip-flop		Outputs S	Function
CLRN	S/AN	A	B	CLK	Before L > H	After L > H		
L	L	X	X	X	L	L	L	Clear
L	H	X	X	X	H	H	L	Clear
H	L	L	L	↑	L	L	L	Add
H	L	L	L	↑	H	L	H	Add
H	L	L	H	↑	L	L	H	Add
H	L	L	H	↑	H	H	L	Add
H	L	H	L	↑	L	L	H	Add
H	L	H	L	↑	H	H	L	Add
H	L	H	H	↑	L	H	L	Add
H	L	H	H	↑	H	H	H	Add
H	H	L	L	↑	L	L	H	Subtract
H	H	L	L	↑	H	H	L	Subtract
H	H	L	H	↑	L	L	L	Subtract
H	H	L	H	↑	H	L	H	Subtract
H	H	H	L	↑	L	H	L	Subtract
H	H	H	L	↑	H	H	H	Subtract
H	H	H	H	↑	L	L	H	Subtract
H	H	H	H	↑	H	H	L	Subtract

Note: ↑ stands for rising edge of signal.



Frequency Dividers		
Macrofunction	Description	Function Prototype
FREQDIV	Frequency Divider	FUNCTION FREQDIV (clr, clk, g) RETURNS (dv2, dv4, dv8, dv16);
7456	Frequency Divider	FUNCTION 7456 (clr, clkb, clka) RETURNS (qc, qb, qa);
7457	Frequency Divider	FUNCTION 7457 (clr, clkb, clka) RETURNS (qc, qb, qa);

Arithmetic Logic Units		
Macrofunction	Description	Function Prototype
74181	Arithmetic Logic Unit	FUNCTION 74181 (s[3..0], m, cn, a3n, a2n, a1n, a0n, b3n, b2n, b1n, b0n) RETURNS (gn, pn, f3n, f2n, f1n, f0n, aeqb, cn4);
74182	Look-Ahead Carry Generator	FUNCTION 74182 (pn3, pn2, pn1, pn0, gn3, gn2, gn1, gn0, ci) RETURNS (pn, gn, cz, cy, cx);
74381	Arithmetic Logic Unit/Function Generator	FUNCTION 74381 (s[2..0], a[3..0], b[3..0], cin) RETURNS (pn, gn, f[3..0]);
74382	Arithmetic Logic Unit/Function Generator	FUNCTION 74382 (s[2..0], a[3..0], b[3..0], cin) RETURNS (ovr, cn4, f[3..0]);



Truth Table of 74181 (Arithmetic Logic Unit) (1)

Selection					Active Low Data	
S3	S2	S1	S0	M = H Logic Functions	M = L; Arithmetic Operations	
					Cn = L (No Carry)	Cn = H (With Carry)
L	L	L	L	$F = /A$	$F = A \text{ minus } 1$	$F = A$
L	L	L	H	$F = /(AB)$	$F = AB \text{ minus } 1$	$F = AB$
L	L	H	L	$F = /A + B$	$F = A(/B) \text{ minus } 1$	$F = A(/B)$
L	L	H	H	$F = 1$	$F = \text{minus } 1 \text{ (2s Comp)}$	$F = \text{ZERO}$
L	H	L	L	$F = /(A+B)$	$F = A \text{ plus } (A + /B)$	$F = A \text{ plus } (A + /B) \text{ plus } 1$
L	H	L	H	$F = /B$	$F = AB \text{ plus } (A + /B)$	$F = AB \text{ plus } (A + /B) \text{ plus } 1$
L	H	H	L	$F = /(A\$B)$	$F = A \text{ minus } B \text{ minus } 1$	$F = A \text{ minus } B$
L	H	H	H	$F = A + /B$	$F = A + /B$	$F = (A + /B) \text{ plus } 1$
H	L	L	L	$F = (/A)B$	$F = A \text{ plus } (A + B)$	$F = A \text{ plus } (A + B) \text{ plus } 1$
H	L	L	H	$F = A \$ B$	$F = A \text{ plus } B$	$F = A \text{ plus } B \text{ plus } 1$
H	L	H	L	$F = B$	$F = A(/B) \text{ plus } (A + B)$	$F = A(/B) \text{ plus } (A + B) \text{ plus } 1$
H	L	H	H	$F = A + B$	$F = (A + B)$	$F = (A + B) \text{ plus } 1$
H	H	L	L	$F = 0$	$F = A \text{ plus } A^*$	$F = A \text{ plus } A \text{ plus } 1$
H	H	L	H	$F = A(/B)$	$F = AB \text{ plus } A$	$F = AB \text{ plus } A \text{ plus } 1$
H	H	H	L	$F = AB$	$F = A(/B) \text{ plus } A$	$F = A(/B) \text{ plus } A \text{ plus } 1$
H	H	H	H	$F = A$	$F = A$	$F = A \text{ plus } 1$

Note: “/” is an inversion symbol.



Truth Table of 74181 (2)

Selection					Active Low Data	
S3	S2	S1	S0	M = H Logic Functions	M = L; Arithmetic Operations	
					/Cn = H (No Carry)	/Cn = L (With Carry)
L	L	L	L	$F = /A$	$F = A$	$F = A$
L	L	L	H	$F = /(A+B)$	$F = A + B$	$F = A \text{ plus } 1$
L	L	H	L	$F = (/A)B$	$F = A + /B$	$F = (A + /B) \text{ plus } 1$
L	L	H	H	$F = 0$	$F = \text{minus } 1 \text{ (2s Comp)}$	$F = \text{ZERO}$
L	H	L	L	$F = /(AB)$	$F = A \text{ plus } A(/B)$	$F = A \text{ plus } A(/B) \text{ plus } 1$
L	H	L	H	$F = /B$	$F = (A + B) \text{ plus } A(/B)$	$F = (A+B) \text{ plus } A(/B) \text{ plus } 1$
L	H	H	L	$F = A \$ B$	$F = A \text{ minus } B \text{ minus } 1$	$F = A \text{ minus } B$
L	H	H	H	$F = A (/B)$	$F = A(/B) \text{ minus } 1$	$F = A(/B)$
H	L	L	L	$F = /A + B$	$F = A \text{ plus } AB$	$F = A \text{ plus } AB \text{ plus } 1$
H	L	L	H	$F = /(A\$B)$	$F = A \text{ plus } B$	$F = A \text{ plus } B \text{ plus } 1$
H	L	H	L	$F = B$	$F = (A + /B) \text{ plus } AB$	$F = (A + /B) \text{ plus } AB \text{ plus } 1$
H	L	H	H	$F = AB$	$F = AB \text{ minus } 1$	$F = AB$
H	H	L	L	$F = 1$	$F = A \text{ plus } A^*$	$F = A \text{ plus } A \text{ plus } 1$
H	H	L	H	$F = A + /B$	$F = (A + B) \text{ plus } A$	$F = (A + B) \text{ plus } A \text{ plus } 1$
H	H	H	L	$F = AB$	$F = (A + /B) \text{ plus } A$	$F = (A + /B) \text{ plus } A \text{ plus } 1$
H	H	H	H	$F = A$	$F = A \text{ minus } 1$	$F = A$

* Each bit shift to next higher significant bit.



Truth Table of 74182 (Look-Ahead Carry Generator)

Function Table of GN

Inputs							Output
GN3	GN2	GN1	GN0	PN3	PN2	PN1	GN
L	X	X	X	X	X	X	L
X	L	X	X	L	X	X	L
X	X	L	X	L	L	X	L
X	X	X	L	L	L	L	L
Other Combinations							H

Function Table of PN

Inputs				Output
PN3	PN2	PN1	PN0	PN
L	L	L	L	L
Other Combinations				H

Function Table of CX

Inputs			Output
GN0	PN0	C1	CX
L	X	X	H
X	L	H	H
Other Combinations			L

Function Table of CY

Inputs					Output
GN1	GN0	PN1	PN0	C1	CY
L	X	X	X	X	H
X	L	L	X	X	H
X	X	L	L	H	H
Other Combinations					L



Function Table of CZ

Inputs							Output
GN2	GN1	GN0	PN2	PN2	PN0	C1	GN
L	X	X	X	X	X	X	H
X	L	X	L	X	X	X	H
X	X	L	L	L	X	X	H
X	X	X	L	L	L	H	H
Other Combinations							L

Truth Table of 74381 (Arithmetic Logic Unit)

Inputs				Outputs F[3..0]
Operation	S2	S1	S0	
Clear	L	L	L	L
$B - A$	L	L	H	$B - A - C_n$
$A - B$	L	H	L	$A - B - C_n$
$A + B$	L	H	H	$A + B + C_n$
$A \oplus B$	H	L	L	$A \oplus B$
$A \# B$	H	L	H	$A \# B$
$A \& B$	H	H	L	$A \& B$
Preset	H	H	H	H

Note: 74182 to cascade multiple 74381 by using GN and PN outputs.

Truth Table of 74382 (Arithmetic Logic Unit)

Inputs				Outputs F[3..0]
Operation	S2	S1	S0	
Clear	L	L	L	L
$B - A$	L	L	H	$B - A - C_n$
$A - B$	L	H	L	$A - B - C_n$
$A + B$	L	H	H	$A + B + C_n$
$A \oplus B$	H	L	L	$A \oplus B$
$A \# B$	H	L	H	$A \# B$
$A \& B$	H	H	L	$A \& B$
Preset	H	H	H	H



For a 16 bits typical application, three 74381 cascades one 74382 and one 74182 generates Look-ahead carry, OVR and CN4 carry.

Latches		
Macrofunction	Description	Function Prototype
EXPLATCH	Latch Implemented with Expanders	FUNCTION EXPLATCH (d, ena) RETURNS (q);
INPLTCH	Input Latch Implemented with Expanders	FUNCTION INPLTCH (d, g) RETURNS (q);
NANDLTCH	/SR NAND Latch with Expanders	FUNCTION NANDLTCH (sn, rn) RETURNS (q, qn);
NORLTCH	SR NOR Latch with Expanders	FUNCTION NORLTCH (s, r) RETURNS (q, qn);
7475	4-Bit Bistable Latch	FUNCTION 7475 (1d, 2d, 3d, 4d, e12, e34) RETURNS (1q, 1qn, 2q, 2qn, 3q, 3qn, 4q, 4qn);
7477	4-Bit Bistable Latch	FUNCTION 7477 (1d, 2d, 3d, 4d, e12, e34) RETURNS (1q, 2q, 3q, 4q);
74116	Dual 4-Bit Latch with Clear	FUNCTION 74116 (1clrn, 2clrn, 1g1n, 1g2n, 2g1n, 2g2n, 1d[4..1], 2d[4..1]) RETURNS(1q[4..1], 2q[4..1]);
74259	8-Bit Addressable Latch with Clear	FUNCTION 74259 (clrn, gn, s[2..0], data) RETURNS (q[7..0]);
74279	Quad /SR Latch	FUNCTION 74279 (s11n, s12n, r1n, s2n, r2n, s31n, s32n, r3n, s4n, r4n) RETURNS (q[4..1]);
74373	Octal Transparent D-	FUNCTION 74373 (oen, g, d[8..1])



	Type Latch with Tri-State Outputs	RETURNS (q[8..1]);
74373B	Octal Transparent D-Type Latch with Tri-State Outputs	FUNCTION 74373B (oen, g, d[8..1]) RETURNS (q[8..1]);
74375	4-Bit Bistable Latch	FUNCTION 74375 (1d, 2d, 3d, 4d, e12, e34) RETURNS (1q, 1qn, 2q, 2qn, 3q, 3qn, 4q, 4qn);
74549	8-Bit 2-Stage Pipelined Latch	FUNCTION 74549 (g, g1n, g2n, insel, d[7..0], outsel, oen) RETURNS (y[7..0]);
74604	Octal 2-Input Multiplexed Latch with Tri-State Outputs	FUNCTION 74604 (clk, sel, a[1..8], b[1..8]) RETURNS (y[1..8]);
74841	10-Bit D-Type Latch with Tri-State Outputs	FUNCTION 74841 (oen, c, d[1..10]) RETURNS (q[1..10]);
74841B	10-Bit D-Type Latch with Tri-State Outputs	FUNCTION 74841B (d[10..1], oen, c) RETURNS (q[10..1]);
74842	10-Bit D-Type Latch with Tri-State Outputs	FUNCTION 74842 (oen, c, d[1..10]) RETURNS (q[1..10]);
74842B	10-Bit D-Type Inverting Latch with Tri-State Outputs	FUNCTION 74824B (dn[10..1], oen, c) RETURNS (q[10..1]);
74843	9-Bit Bus Interface D-Type Latch with Tri-State Outputs	FUNCTION 74843 (oen, clrn, pren, ena, d[1..9]) RETURNS (q[1..9]);
74844	9-Bit Bus Interface D-Type Inverting Latch with Tri-State Outputs	FUNCTION 74844 (oen, clrn, pren, ena, dn[1..9]) RETURNS (q[1..9]);
74845	8-Bit Bus Interface D-Type Latch with Tri-	FUNCTION 74845 (oen1, oen2, oen3, clrn, pren, ena, d[1..8])



	State Outputs	RETURNS (q[1..8]);
74846	8-Bit Bus Interface D-Type Inverting Latch with Tri-State Outputs	FUNCTION 74846 (oen1, oen2, oen3, clrn, pren, ena, d[1..8]) RETURNS (q[1..8]);
74990	8-Bit Transparent Read-Back Latch	FUNCTION 74990 (oerb, c) RETURNS (d[1..8], q[1..8]);

Truth Table of EXPLATCH (Latch)

Inputs ENA D		Outputs Q
L	X	Q ₀
H	L	L
H	H	H

Truth Table of 7475 (Latch)

Inputs		Outputs	
D	E	Q	QN
L	H	H	H
L	H	H	L
X	L	Q ₀ *	/Q ₀

Truth Table of 74116 (Latch)

Inputs			Outputs	
CLR _N	Enable		D	Q
	G1 _N	G2 _N		
L	X	X	X	L
H	L	L	L	L
H	L	L	H	H
H	X	H	X	Q ₀
H	H	X	X	Q ₀

Truth Table of 74279 (Latch)

Inputs		Outputs
SN*	RN	Q
H	H	Q ₀
L	H	H
H	L	L
L	L	H**

* For latches with double S inputs: H = both SN inputs high. L = one or both SN inputs low.



** This configuration is nonstable; that is, it may not persist when the SN and RN inputs return to their inactive (high) level.

Truth Table of 74373 (Latch)

Inputs			Outputs
0EN	G	D	Q
H	X	X	Z
L	X	X	X
L	H	L	L
L	H	H	H
L	L	X	Qo

Application-Specific Functions		
Macrofunction	Description	Function Prototype
NTSC	NTSC Video Control Signal Generator	FUNCTION NTSC (clock, reset) RETURNS (csync, hd, vd, blank, burst, field);
PLL	Rising- and Falling-Edge Detector	FUNCTION PLL (a, b, nset) RETURNS (nup, tri-up, ndown, tri-down);

Multipliers		
Macrofunction	Description	Function Prototype
MULT2	2-Bit Sign Magnitude Multiplier	FUNCTION MULT2 (a[2..0], b[2..0], g) RETURNS (y[4..0]);
MULT24	2-Bit-by-4-Bit Parallel Binary Multiplier	FUNCTION MULT24 (a[5..1], b[3..1], g) RETURNS (y[7..1]);
MULT4	4-Bit Parallel Binary Multiplier	FUNCTION MULT4 (a[5..1], b[5..1], g) RETURNS (y[9..1]);



MULT4B	4-Bit Parallel Binary Multiplier	FUNCTION MULT4B (a[5..1], b[5..1], g) RETURNS (y[9..1]);
TMULT4	4-Bit-by-4-Bit Parallel Binary Multiplier	FUNCTION TMULT4 (gan, gbn, a[5..1], b[5..1]) RETURNS (y[9..1]);
7497	Synchronous 6-Bit Rate Multiplier	FUNCTION 7497 (clk, clr, enn, strbn, b[5..0] , uni/cas) RETURNS (y, zn, tcn);
74261	2-Bit Parallel Binary Multiplier	FUNCTION 74261 (b[4..0], m[2..0], g) RETURNS (q4n, q[3..0]);
74284	4-Bit-by-4-Bit Parallel Binary Multiplier (Upper 4 Bits of Result)	FUNCTION 74284 (gan, gbn, a[4..1], b[4..1]) RETURNS (y[8..5]);
74285	4-Bit-by-4-Bit Parallel Binary Multiplier (Lower 4 Bits of Result)	FUNCTION 74285 (gan, gbn, a[4..1], b[4..1]) RETURNS (y[4..1]);

Truth Table of MULT2 (Multiplier)

Inputs							Outputs				
A2*	A1	A0	B2*	B1	B0	G	Y4*	Y3	Y2	Y1	Y0
X	X	X	X	X	X	L	L	L	L	L	L
L	a1	a0	L	b1	b0	H	L	A multiply by B			
L	a1	a0	H	b1	b0	H	H				
H	a1	a0	L	b1	b0	H	H				
H	a1	a0	H	b1	b0	H	L				

* Can be considered as sign bit of Signed and Magnitude.



Truth Table of MULT24 (Multiplier)

Inputs									Outputs						
A5*	A4	A3	A2	A1	B3*	B2	B1	G	Y7*	Y6	Y5	Y4	Y3	Y2	Y1
X	X	X	X	X	X	X	X	L	L	L	L	L	L	L	L
L	a4	a3	a2	a1	L	b2	b1	H	L	A 乘以 B					
L	a4	a3	a2	a1	H	b2	b1	H	H						
H	a4	a3	a2	a1	L	b2	b1	H	H						
H	a4	a3	a2	a1	H	b2	b1	H	L						

* Can be considered as sign bit of Signed and Magnitude.

Truth Table of 7497 (Multiplier)

Inputs										Outputs		
CLR	ENN	STRBN	B5	B4	B3	B2	B1	B0	UNI/CAS	Y	ZN	TCN
H	X	H	X	X	X	X	X	X	H	L	H	L
L	L	L	B[5..0]						H	*	**	***
L	H	X	X	X	X	X	X	X	H	Yo	Zno	H
L	L	L	X	X	X	X	X	X	L	H		

* Y has B[5..0] low pulses in 64 clock cycles.

** ZN has B[5..0] high pulses in 64 clock cycles.

*** TCN pulses low every 64 clock cycles for cascading.

Truth Table of 74284 (Multiplier)

Inputs				Outputs
GAN	GBN	A[4..1]	B[4..1]	Y[8..5]
1	1	X	X	0
1	0	X	X	0
0	1	X	X	0
0	0	A[4..1]	B[4..1]	$Y[8..5] = A[4..1] \times B[4..1]$



Buffers		
Macrofunction	Description	Function Prototype
BTRI	Active-Low Tri-State Buffer	FUNCTION BTRI (oen, in) RETURNS (out);
74240	Octal Inverting Tri-State Buffer	FUNCTION 74240 (1gn, 1a[1..4], 2gn, 2a[1..4]) RETURNS (1y[1..4], 2y[1..4]);
74240B	Octal Inverting Tri-State Buffer with 2 Sections	FUNCTION 74240B (a[4..1], b[4..1], agn, bgn) RETURNS (ay[4..1], by[4..1]);
74241	Octal Tri-State Buffer	FUNCTION 74241 (1gn, 1a[1..4], 2g, 2a[1..4]) RETURNS (1y[1..4], 2y[1..4]);
74241B	Octal Tri-State Buffer with 2 Sections	FUNCTION 74241B (a[4..1], b[4..1], agn, bg) RETURNS (ay[4..1], by[4..1]);
74244	Octal Tri-State Buffer	FUNCTION 74244 (1gn, 1a[1..4], 2gn, 2a[1..4]) RETURNS (1y[1..4], 2y[1..4]);
74244B	Octal Tri-State Buffer with 2 Sections	FUNCTION 74244B (a[4..1], b[4..1], agn, bgn) RETURNS (ay[4..1], by[4..1]);
74365	Hex Tri-State Buffer	FUNCTION 74365 (gn1, gn2, a[1..6]) RETURNS (y[1..6]);
74366	Hex Inverting Tri-State Buffer	FUNCTION 74366 (gn1, gn2, a[1..6]) RETURNS (yn[1..6]);
74367	Hex Tri-State Buffer	FUNCTION 74367 (1gn, 1a[1..4], 2gn, 2a[1..2]) RETURNS (1y[1..4], 2y[1..2]);
74368	Hex Inverting Tri-State Buffer	FUNCTION 74368 (1gn, 1a[1..4], 2gn, 2a[1..2]) RETURNS (1yn[1..4], 2yn[1..2]);



74465	Octal Tri-State Buffer	FUNCTION 74465 (gn[1..2], a[1..8]) RETURNS (y[1..8]);
74466	Octal Inverting Tri-State Buffer	FUNCTION 74466 (gn[1..2], a[1..8]) RETURNS (yn[1..8]);
74467	Octal Tri-State Buffer	FUNCTION 74467 (1gn, 1a[1..4], 2gn, 2a[1..4]) RETURNS (1y[1..4], 2y[1..4]);
74468	Octal Inverting Tri-State Buffer	FUNCTION 74468 (1gn, 1a[1..4], 2gn, 2a[1..4]) RETURNS (1yn[1..4], 2yn[1..4]);
74540	Octal Inverting Tri-State Buffer	FUNCTION 74540 (gn[1..2], a[1..8]) RETURNS (yn[1..8]);
74541	Octal Tri-State Buffer	FUNCTION 74541 (gn[1..2], a[1..8]) RETURNS (y[1..8]);

Truth Table of 74244 (Buffer)

Inputs		Outputs
GN	A	Y
H	X	Z
L	L	L
L	H	H

Truth Table of 74466 (Buffer)

Inputs			Outputs
GN1	GN2	A	YN
H	X	X	Z
X	H	X	Z
L	L	L	H
L	L	H	L

Multiplexers		
Macrofunction	Description	Function Prototype
21MUX	2-Line-to-1-Line Multiplexer	FUNCTION 21MUX (s, a, b) RETURNS (y);
161MUX	16-Line-to-1-Line Multiplexer	FUNCTION 161MUX (gn, sel[3..0], in[15..0]) RETURNS (out);



2X8MUX	2-Line-to-1-Line Multiplexer for 8-Bit Buses	FUNCTION 2X8MUX (sel, a[7..0], b[7..0]) RETURNS (y[7..0]);
81MUX	8-Line-to-1-Line Multiplexer	FUNCTION 81mux (c, b, a, d[7..0], gn) RETURNS (y, wn);
74151	8-Line-to-1-Line Multiplexer	FUNCTION 74151 (c, b, a, d[7..0], gn) RETURNS (y, wn);
74151B	8-Line-to-1-Line Multiplexer	FUNCTION 74151B (c, b, a, d[7..0], gn) RETURNS (y, wn);
74153	Dual 4-Line-to-1-Line Multiplexer	FUNCTION 74153 (b, a, 1gn, 1c[3..0], 2gn, 2c[3..0]) RETURNS (1y, 2y);
74157	Quad 2-Line-to-1-Line Multiplexer	FUNCTION 74157 (gn, sel, a[4..1], b[4..1]) RETURNS (y[4..1]);
74158	Quad 2-Line-to-1-Line Multiplexer with Inverting Outputs	FUNCTION 74158 (gn, sel, 1a, 2a, 3a, 4a, 1b, 2b, 3b, 4b) RETURNS (1yn, 2yn, 3yn, 4yn);
74251	8-Line-to-1-Line Data Selector with Tri-State Outputs	FUNCTION 74251 (c, b, a, d[7..0], gn) RETURNS (y, wn);
74253	Dual 4-Line-to-1-Line Data Selectors with Tri-State Outputs	FUNCTION 74253 (b, a, 1gn, 1c[0..3], 2gn, 2c[0..3]) RETURNS (1y, 2y);
74257	Quad 2-Line-to-1-Line Multiplexers with Tri-State Outputs	FUNCTION 74257 (gn, sel, a[4..1], b[4..1]) RETURNS (y[4..1]);
74258	Quad 2-Line-to-1-Line Multiplexers with Inverting Tri-State Outputs	FUNCTION 74258 (gn, sel, a[4..1], b[4..1]) RETURNS (yn[4..1]);
74298	Quad 2-Input	FUNCTION 74298 (wrsl, clkn, a1, b1, c1,



	Multiplexer with Storage	d1, a2, b2, c2, d2) RETURNS (qa, qb, qc, qd);
74352	Dual 4-Line-to-1-Line Data Selector /Multiplexer with Inverting Outputs	FUNCTION 74352 (b, a, 1gn, 1c[0..3], 2gn, 2c[0..3]) RETURNS (1yn, 2yn);
74353	Dual 4-Line-to-1-Line Data Selector /Multiplexer with Tri-State Inverting Outputs	FUNCTION 74353 (b, a, 1gn, 1c[0..3], 2gn, 2c[0..3]) RETURNS (1yn, 2yn);
74354	8-Line-to-1-Line Data Selector/Multiplexer/ Register with Tri-State Outputs	FUNCTION 74354 (gn1, gn2, g3, s[2..0], scn, dcn, d[7..0]) RETURNS (y, wn);
74356	8-Line-to-1-Line Data Selector/Multiplexer/ Register with Tri-State Outputs	FUNCTION 74356 (gn1, gn2, g3, s[2..0], scn, clk, d[7..0]) RETURNS (y, wn);
74398	Quad 2-Input Multiplexer with Storage	FUNCTION 74398 (sel, a1, b1, c1, d1, a2, b2, c2, d2, clk) RETURNS (qa, qan, qb, qbn, qc, qcn, qd, qdn);
74399	Quad 2-Input Multiplexer with Storage	FUNCTION 74399 (sel, a1, b1, c1, d1, a2, b2, c2, d2, clk) RETURNS (qa, qb, qc, qd);

Truth Table of 2×8 MUX (Multiplexer)

Inputs			Outputs
SEL	A[7..0]	B[7..0]	Y[7..0]
H	a[7..0]	b[7..0]	a[7..0]
L	a[7..0]	b[7..0]	b[7..0]



Truth Table of 74151 (Multiplexer)

Inputs				Outputs	
Select			Enable		
C	B	A	GN	Y	WN
X	X	X	H	L	H
L	L	L	L	D0	/D0
L	L	H	L	D1	/D1
L	H	L	L	D2	/D2
L	H	H	L	D3	/D3
H	L	L	L	D4	/D4
H	L	H	L	D5	/D5
H	H	L	L	D6	/D6
H	H	H	L	D7	/D7

Truth Table of 74298 (Multiplexer)

Inputs		Outputs			
WRS�	CLKN	QA	QB	QC	QD
L		A1	B1	C1	D1
H		A2	B2	C2	D2
X	H	QAo*	QBo*	QCo*	QDo*

* QAo..QDo equal to the level of QA to QD at the last falling edge of clock.

Comparators		
Macrofunction	Description	Function Prototype
8MCOMP	Note 8-Bit Magnitude Comparator	FUNCTION 8MCOMP (a[7..0], b[7..0]) RETURNS (altb, aeqb, agtb, aeb[7..0]);
8MCOMPB	8-Bit Magnitude Comparator	FUNCTION 8MCOMPB (a[7..0], b[7..0]) RETURNS (altb, aeqb, agtb, aeb[7..0]);
7485	Note 4-Bit Magnitude Comparator	FUNCTION 7485 (a[3..0], b[3..0], agbi, albi, aebi) RETURNS (agbo, albo, aebo);



74518	Note 8-Bit Identity Comparator	FUNCTION 74518 (p[7..0], q[7..0], gn) RETURNS (pq);
74518B	8-Bit Identity Comparator	FUNCTION 74518B (p[7..0], q[7..0], gn) RETURNS (pq);
74684	8-Bit Magnitude/ Identity Comparator	FUNCTION 74684 (p[7..0], q[7..0]) RETURNS (equaln, p_gr_qn);
74686	8-Bit Magnitude/ Identity Comparator	FUNCTION 74686 (g1n, g2n, p[7..0], q[7..0]) RETURNS (equaln, p_gr_qn);
74688	8-Bit Identity Comparator	FUNCTION 74688 (gn, p[7..0], q[7..0]) RETURNS (equaln);

Truth Table of 7485 (Comparator)

Comparing Inputs				Cascading Inputs			Outputs		
A3, B3	A2, B2	A1, B1	A0, B0	AGB1	ALB1	AEB1	AGB0	ALB0	AEB0
$A3 > B3$	X	X	X	X	X	X	H	L	L
$A3 < B3$	X	X	X	X	X	X	L	H	L
$A3 = B3$	$A2 > B2$	X	X	X	X	X	H	L	L
$A3 = B3$	$A2 < B2$	X	X	X	X	X	L	H	L
$A3 = B3$	$A2 = B2$	$A1 > B1$	X	X	X	X	H	L	L
$A3 = B3$	$A2 = B2$	$A1 < B1$	X	X	X	X	L	H	L
$A3 = B3$	$A2 = B2$	$A1 = B1$	$A0 > B0$	X	X	X	H	L	L
$A3 = B3$	$A2 = B2$	$A1 = B1$	$A0 < B0$	X	X	X	L	H	L
$A3 = B3$	$A2 = B2$	$A1 = B1$	$A0 = B0$	H	L	L	H	L	L
$A3 = B3$	$A2 = B2$	$A1 = B1$	$A0 = B0$	L	H	L	L	H	L
$A3 = B3$	$A2 = B2$	$A1 = B1$	$A0 = B0$	L	L	H	L	L	H
$A3 = B3$	$A2 = B2$	$A1 = B1$	$A0 = B0$	X	X	H	L	L	H
$A3 = B3$	$A2 = B2$	$A1 = B1$	$A0 = B0$	H	H	L	L	L	L
$A3 = B3$	$A2 = B2$	$A1 = B1$	$A0 = B0$	L	L	L	H	H	L



Truth Table of 74688 (Comparator)

Inputs		Outputs
GN	DATA	EQUALN
H	X	H
L	$P = Q$	L
L	$P <> Q$	H

Parity Generators/Checkers		
Macrofunction	Description	Function Prototype
74180	9-Bit Odd/Even Parity Generator/Checker	FUNCTION 74180 (a, b, c, d, e, f, g, h, evni, oddi) RETURNS (evns, odds);
74180B	9-Bit Odd/Even Parity Generator/Checker	FUNCTION 74180B (d[7..0], evni, oddi) RETURNS (evns, odds);
74280	9-Bit Odd/Even Parity Generator/Checker	FUNCTION 74280 (a, b, c, d, e, f, g, h, i) RETURNS (even, odd);
74280B	9-Bit Odd/Even Parity Generator/Checker	FUNCTION 74280B (d[8..0]) RETURNS (evns, odds);

Truth Table of 74180 (Parity Generator/Checker)

Inputs			Outputs	
# of Hs at				
A through H	ENVI	ODDI	EVNS	ODDS
Even	H	L	H	L
Odd	H	L	L	H
Even	L	H	L	H



Odd	L	H	H	L
X	H	H	L	L
X	L	L	H	H

Truth Table of 74280 (Parity Generator/Checker)

Number of Inputs A through I that are High					Outputs	
					EVEN	ODD
0	2	4	6	8	H	L
1	3	5	7	9	L	H

Converters		
Macrofunction	Description	Function Prototype
74184	BCD-to-Binary Converter	FUNCTION 74184 (e, d, c, b, a, gn) RETURNS (y[8..1]);
74185	Binary-to-BCD Converter	FUNCTION 74185 (e, d, c, b, a, gn) RETURNS (y[8..1]);

Truth Table of 74184 (Converter) (1)

BCD	Inputs						Outputs				
	E	D	C	B	A	GN	Y5	Y4	Y3	Y2	Y1
0-1	L	L	L	L	L	L	L	L	L	L	L
2-3	L	L	L	L	H	L	L	L	L	L	H
4-5	L	L	L	H	L	L	L	L	L	H	L
6-7	L	L	L	H	H	L	L	L	L	H	H
8-9	L	L	H	L	L	L	L	L	H	L	L
10-11	L	H	L	L	L	L	L	L	H	L	H
12-13	L	H	L	L	H	L	L	L	H	H	L
14-15	L	H	L	H	L	L	L	L	H	H	H



16–17	L	H	L	H	H	L	L	H	L	L	L
18–19	L	H	H	L	L	L	L	H	L	L	H
20–21	H	L	L	L	L	L	L	H	L	H	L
22–23	H	L	L	L	H	L	L	H	L	H	H
24–25	H	L	L	H	L	L	L	H	H	L	L
26–27	H	L	L	H	H	L	L	H	H	L	H
28–29	H	L	H	L	L	L	L	H	H	H	L
30–31	H	H	L	L	L	L	L	H	H	H	H
32–33	H	H	L	L	H	L	H	L	L	L	L
34–35	H	H	L	H	L	L	H	L	L	L	H
36–37	H	H	L	H	H	L	H	L	L	H	L
38–39	H	H	H	L	L	L	H	L	L	H	H
Any	X	X	X	X	X	H	H	H	H	H	H

Input conditions other than those shown produce high levels at outputs Y1 to Y5.

Outputs Y6, Y7, and Y8 are not used for BCD-to-binary conversion.

Truth Table of 74184 (2)

BCD	Inputs					Outputs			
	E*	D	C	B	A	GN	Y8	Y7	Y6
0	L	L	L	L	L	L	H	L	H
1	L	L	L	L	H	L	H	L	L
2	L	L	L	H	L	L	L	H	H
3	L	L	L	H	H	L	L	H	L
4	L	L	H	L	L	L	L	H	H
5	L	L	H	L	H	L	L	H	L
6	L	L	H	H	L	L	L	L	H
7	L	L	H	H	H	L	L	L	L
8	L	H	L	L	L	L	L	L	H
9	L	H	L	L	H	L	L	L	L
0	H	L	L	L	L	L	L	L	L
1	H	L	L	L	H	L	H	L	L
2	H	L	L	H	L	L	H	L	L
3	H	L	L	H	H	L	L	H	H



4	H	L	H	L	L	L	L	H	H
5	H	L	H	L	H	L	L	H	L
6	H	L	H	H	L	L	L	H	L
7	H	L	H	H	H	L	L	L	H
8	H	H	L	L	L	L	L	L	H
9	H	H	L	L	H	L	L	L	L
Any	X	X	X	X	X	H	H	H	H

* When these devices are used as complement converters, input E is used as a mode control. When this input is low, the BCD 9s complement is generated; when it is high, the BCD 10's complement is generated.

Input conditions other than those shown produce high levels at outputs Y6, Y7, and Y8.

Outputs Y1 through Y5 are not used for BCD 9s or 10s complement conversion.

Truth Table of 74185 (Converter)

BCD	Inputs						Outputs							
	E	D	C	B	A	GN	Y8	Y7	Y6	Y5	Y4	Y3	Y2	Y1
0-1	L	L	L	L	L	L	H	H	L	L	L	L	L	L
2-3	L	L	L	L	H	L	H	H	L	L	L	L	L	H
4-5	L	L	L	H	L	L	H	H	L	L	L	L	H	L
6-7	L	L	L	H	H	L	H	H	L	L	L	L	H	H
8-9	L	L	H	L	L	L	H	H	L	L	L	H	L	L
10-11	L	L	H	L	H	L	H	H	L	L	H	L	L	L
12-13	L	L	H	H	L	L	H	H	L	L	H	L	L	H
14-15	L	L	H	H	H	L	H	H	L	L	H	L	H	L
16-17	L	H	L	L	L	L	H	H	L	L	H	L	H	H
18-19	L	H	L	L	H	L	H	H	L	L	H	H	L	L
20-21	L	H	L	H	L	L	H	H	L	H	L	L	L	L
22-23	L	H	L	H	H	L	H	H	L	H	L	L	L	H
24-25	L	H	H	L	L	L	H	H	L	H	L	L	H	L
26-27	L	H	H	L	H	L	H	H	L	H	L	L	H	H
28-29	L	H	H	H	L	L	H	H	L	H	L	H	L	L
30-31	L	H	H	H	H	L	H	H	L	H	H	L	L	L
32-33	H	L	L	L	L	L	H	H	L	H	H	L	L	H
34-35	H	L	L	L	H	L	H	H	L	H	H	L	H	L



36–37	H	L	L	H	L	L	H	H	L	H	H	L	H	H
38–39	H	L	L	H	H	L	H	H	L	H	H	H	L	L
40–41	H	L	H	L	L	L	H	H	H	L	L	L	L	L
42–43	H	L	H	L	H	L	H	H	H	L	L	L	L	H
44–45	H	L	H	H	L	L	H	H	H	L	L	L	H	L
46–47	H	L	H	H	H	L	H	H	H	L	L	L	H	H
48–49	H	H	L	L	L	L	H	H	H	L	L	H	L	L
50–51	H	H	L	L	H	L	H	H	H	L	H	L	L	L
52–53	H	H	L	H	L	L	H	H	H	L	H	L	L	H
54–55	H	H	L	H	H	L	H	H	H	L	H	L	H	L
56–57	H	H	H	L	L	L	H	H	H	L	H	L	H	H
58–59	H	H	H	L	H	L	H	H	H	L	H	H	L	L
60–61	H	H	H	H	L	L	H	H	H	H	L	L	L	L
62–63	H	H	H	H	H	L	H	H	H	H	L	L	L	H
Any	X	X	X	X	X	H	H	H	H	H	H	H	H	H

Rate Multipliers		
Macrofunction	Description	Function Prototype
74167	Synchronous Decade Rate Multiplier	FUNCTION 74167 (clk, clr, enn, strbn, b[3..0], uni/cas, set9) RETURNS (y, zn, eno);

Truth Table of 74167 (Rate Multiplier)

Inputs									Outputs			註解
CLR	ENN	STRBN	B3	B2	B1	B0	#Clock Pulses	UNI/CAS	Y	ZN	ENO	
H	X	H	X	X	X	X	X	H	L	H	H	1
L	L	L	L	L	L	L	10	H	L	H	1	2
L	L	L	L	L	L	H	10	H	1	1	1	2
L	L	L	L	L	H	L	10	H	2	2	1	2
L	L	L	L	L	H	H	10	H	3	3	1	2
L	L	L	L	H	L	L	10	H	4	4	1	2
L	L	L	L	H	L	H	10	H	5	5	1	2



L	L	L	L	H	H	L	10	H	6	6	1	2
L	L	L	L	H	H	H	10	H	7	7	1	2
L	L	L	H	L	L	L	10	H	8	8	1	2
L	L	L	H	L	L	H	10	H	9	9	1	2
L	L	L	H	L	H	L	10	H	8	8	1	2, 3
L	L	L	H	L	H	H	10	H	9	9	1	2, 3
L	L	L	H	H	L	L	10	H	8	8	1	2, 3
L	L	L	H	H	L	H	10	H	9	9	1	2, 3
L	L	L	H	H	H	L	10	H	8	8	1	2, 3
L	L	L	H	H	H	H	10	H	9	9	1	2, 3
L	L	L	X	X	X	X	10	L	H	9	1	4

- Note: 1. This is a simplified illustration of the clear function. The states of Clocks and the strobe can affect the logic level of Y and Z. A low UNI/CAS will cause output Y to remain high.
2. Each rate illustrated assumes a constant value at rate inputs; however, these illustrations in no way prohibit variable-rate inputs.
3. These input conditions exceed the range of the decimal rate inputs.
4. UNI/CAS can be used to inhibit output Y.

Counters		
Macrofunction	Description	Function Prototype
GRAY4	Gray Code Counter	FUNCTION GRAY4 (clk, ena) RETURNS (qd, qc, qb, qa);
UNICNT	Universal 4-Bit Up/ Down Counter Left/ Right Shift Register with Asynchronous Set and Load, Clear, and Cascade	FUNCTION UNICNT (clk, clr, set, load, ctst, dnup, rtlt, cin, data, d, c, b, a) RETURNS (qd, qc, qb, qa, cout);
16CUDSLR	16-Bit Binary Up/ Down Counter Left/ Right Shift Register with Asynchronous Set	FUNCTION 16CUDSLR (clk, clrn, setn, data, stct, dnup, ltrt) RETURNS (q[16..1]);
16CUDSRB	16-Bit Binary Up/	FUNCTION 16CUDSRB (clk, clrn, setn,



	Down Counter with Left/Right Shift Register, Asynchronous Clear, and Asynchronous Set	data, stct, dnup, ltrt) RETURNS (q[16..1]);
4COUNT	4-Bit Binary Up/Down Counter with Synchronous Load (LDN), Asynchronous Clear, and Asynchronous Load (SETN)	FUNCTION 4COUNT (clk, clrn, setn, ldn, cin, dnup, d, c, b, a) RETURNS (qd, qc, qb, qa, cout);
8COUNT	8-Bit Binary Up/Down Counter with Synchronous Load (LDN), Asynchronous Clear, and Asynchronous Load (SETN)	FUNCTION 8COUNT (clk, clrn, setn, ldn, dnup, gn, h, g, f, e, d, c, b, a) RETURNS (qh, qg, qf, qe, qd, qc, qb, qa, cout);
7468	Dual Decade Counter	FUNCTION 7468 (1clk1, 1clk2, 1clrn, 2clk, 2clrn) RETURNS (1qd, 1qc, 1qb, 1qa, 2qd, 2qc, 2qb, 2qa);
7469	Dual Binary Counter	FUNCTION 7469 (1clk1, 1clk2, 1clrn, 2clk, 2clrn) RETURNS (1qd, 1qc, 1qb, 1qa, 2qd, 2qc, 2qb, 2qa);
7490	Decade or Binary Counter with Clear and Set-to-9	FUNCTION 7490 (set9a, set9b, clra, clrb, clka, clkb) RETURNS (qd, qc, qb, qa);
7492	Divide-by-12 Counter	FUNCTION 7492 (clra, clrb, clka, clkb) RETURNS (qd, qc, qb, qa);
7493	4-Bit Binary Counter	FUNCTION 7493 (clka, clkb, ro1, ro2)



		RETURNS (qd, qc, qb, qa);
74143	4-Bit Counter/Latch, 7-Segment Driver	FUNCTION 74143 (clk, clrn, strbn, pcein, scein, bin, rbin, dpi) RETURNS (qd, qc, qb, qa, max, a, b, c, d, e, f, g, dpo, rbon);
74160	4-Bit Decade Counter with Syn- chronous Load and Asynchronous Clear	FUNCTION 74160 (clk, ldn, clrn, enp, ent, d, c, b, a) RETURNS (qd, qc, qb, qa, rco);
74161	4-Bit Binary Up Counter with Syn- chronous Load and Asynchronous Clear	FUNCTION 74161 (clk, ldn, clrn, enp, ent, d, c, b, a) RETURNS (qd, qc, qb, qa, rco);
74162	4-Bit Decade Up Counter with Syn- chronous Load and Synchronous Clear	FUNCTION 74162 (clk, ldn, clrn, enp, ent, d, c, b, a) RETURNS (qd, qc, qb, qa, rco);
74163	4-Bit Binary Up Counter with Syn- chronous Load and Synchronous Clear	FUNCTION 74163 (clk, ldn, clrn, enp, ent, d, c, b, a) RETURNS (qd, qc, qb, qa, rco);
74168	Synchronous 4-Bit Decade Up/Down Counter	FUNCTION 74168 (ldn, entn, enpn, u/dn, clk, d[3..0]) RETURNS (q[3..0], tcn);
74169	Synchronous 4-Bit Binary Up/Down Counter	FUNCTION 74169 (ldn, entn, enpn, u/dn, clk, d[3..0]) RETURNS (q[3..0], tcn);
74176	Presettable Decade Counter	FUNCTION 74176 (clrn, ldn, clk1, clk2, d, c, b, a) RETURNS (qd, qc, qb, qa);
74177	Presettable Binary Counter	FUNCTION 74177 (clrn, ldn, clk1, clk2, d, c, b, a) RETURNS (qd, qc, qb, qa);



74190	4-Bit Decade Up/ Down Counter with Asynchronous Load	FUNCTION 74190 (clk, gn, ldn, dnup, d, c, b, a) RETURNS (qd, qc, qb, qa, mxmn, rcon);
74191	4-Bit Binary Up/ Down Counter with Asynchronous Load	FUNCTION 74191 (clk, gn, ldn, dnup, d, c, b, a) RETURNS (qd, qc, qb, qa, mxmn, rcon);
74192	4-Bit Decade Up/ Down Counter with Asynchronous Clear Registers	FUNCTION 74192 (clr, up, dn, ldn, d, c, b, a) RETURNS (qd, qc, qb, qa, con, bon);
74193	4-Bit Binary Up/ Down Counter with Asynchronous Clear	FUNCTION 74193 (clr, up, dn, ldn, d, c, b, a) RETURNS (qd, qc, qb, qa, con, bon);
74196	Presettable Decade Counter	FUNCTION 74196 (clrn, ldn, clk1, clk2, d, c, b, a) RETURNS (qd, qc, qb, qa);
74197	Presettable Binary Counter	FUNCTION 74197 (clrn, ldn, clk1, clk2, d, c, b, a) RETURNS (qd, qc, qb, qa);
74290	Decade Counter with Clear	FUNCTION 74290 (clka, clk b, clra, clrb, set9a, set9b) RETURNS (qd, qc, qb, qa);
74292	Programmable Frequency Divider/ Digital Timer	FUNCTION 74292 (clk1, clk2, clrn, e, d, c, b, a) RETURNS (q, tp1, tp2, tp3);
74293	Binary Counter with Clear	FUNCTION 74293 (clka, clk b, clra, clrb) RETURNS (qd, qc, qb, qa);
74294	Programmable Frequency Divider/ Digital Timer	FUNCTION 74294 (clk1, clk2, clrn, d, c, b, a) RETURNS (q, tp);
74390	Dual Decade Counter	FUNCTION 74390 (1clr, 1clka, 1clkb, 2clr, 2clka, 2clkb) RETURNS (1qd, 1qc, 1qb, 1qa, 2qd, 2qc,



		2qb, 2qa);
74393	Dual 4-Bit Up Counter with Asynchronous Clear	FUNCTION 74393 (a1, clr1, a2, clr2) RETURNS (q1a, q1b, q1c, q1d, q2a, q2b, q2c, q2d);
74490	Dual 4-Bit Decade Counter	FUNCTION 74490 (1set9, 1clr, 1clk, 2set9, 2clr, 2clk) RETURNS (1qd, 1qc, 1qb, 1qa, 2qd, 2qc, 2qb, 2qa);
74568	Decade Up/Down Counter with Synchronous Load and Clear and Asynchronous Clear	FUNCTION 74569 (clk, entn, enpn, aclrn, sclrn, u/dn, ldn, d[3..0], oen) RETURNS (q[3..0], rcon, ccon);
74569	Binary Up/Down Counter with Synchronous Load and Clear and Asynchronous Clear	FUNCTION 74569 (clk, entn, enpn, aclrn, sclrn, u/dn, ldn, d[3..0], oen) RETURNS (q[3..0], rcon, ccon);
74590	8-Bit Binary Counter with Tri-State Output Registers	FUNCTION 74590 (gn, cclrn, ccken, cclk, rclk) RETURNS (qh, qg, qf, qe, qd, qc, qb, qa, rcon);
74592	8-Bit Binary Counter with Input Registers	FUNCTION 74592 (cclrn, cloadn, rclk, ccken, cclk, h, g, f, e, d, c, b, a) RETURNS (rcon);
74668	Synchronous Decade Up/Down Counter	FUNCTION 74668 (clk, entn, enpn, u/dn, ldn, d[3..0]) RETURNS (q[3..0], tcn);
74669	Synchronous 4-Bit Binary Up/Down Counter	FUNCTION 74669 (clk, entn, enpn, u/dn, ldn, d[3..0]) RETURNS (q[3..0], tcn);
74690	Synchronous Decade	FUNCTION 74690 (gn, cclrn, ldn, enp, ent,



	Counter with Out put Registers, Multiplexed Tri-State Outputs, and Asynchronous Clear	rclrn, rclk, r/cn, cclk, d[3..0]) RETURNS (q[3..0], rco);
74691	Synchronous Binary Counter with Output Registers, Multiplexed Tri-State Outputs, and Asynchronous Clear	FUNCTION 74691 (gn, cclrn, ldn, enp, ent, rclrn, rclk, r/cn, cclk, d[3..0]) RETURNS (q[3..0], rco);
74693	Synchronous Binary Counter with Output Registers, Multiplexed Tri-State Outputs, and Synchronous Clear	FUNCTION 74693 (gn, cclrn, ldn, enp, ent, rclrn, rclk, r/cn, cclk, d[3..0]) RETURNS (q[3..0], rco);
74696	Synchronous Decade Up/Down Counter with Output Registers, Multiplexed Tri-State Outputs, and Asynchronous Clear	FUNCTION 74696 (u/dn, r/cn, rclk, ldn, gn, entn, enpn, d3, d2, d1, d0, cclrn, cclk) RETURNS (tcn, q3, q2, q1, q0);
74697	Synchronous Binary Up/Down Counter with Output Registers, Multiplexed Tri-State Outputs, and Asynchronous Clear	FUNCTION 74697 (u/dn, r/cn, rclk, ldn, gn, entn, enpn, d3, d2, d1, d0, cclrn, cclk) RETURNS (tcn, q3, q2, q1, q0);



74698	Synchronous Decade Up/Down Counter with Output Registers, Multiplexed Tri-State Outputs, and Synchronous Clear	FUNCTION 74698 (u/dn, r/cn, rclk, ldn, gn, entn, enpn, d3, d2, d1, d0, cclrn, cclk) RETURNS (tcn, q3, q2, q1, q0);
74699	Synchronous Binary Up/Down Counter with Output Registers, Multiplexed Tri-State Outputs, and Synchronous Clear	FUNCTION 74699 (u/dn, r/cn, rclk, ldn, gn, entn, enpn, d3, d2, d1, d0, cclrn, cclk) RETURNS (tcn, q3, q2, q1, q0);

Truth Table of 4COUNT (Counter)

Inputs										Outputs				
CLK	CLRN	SETN	LDN	CIN	DNUP	D	C	B	A	QD	QC	QB	QA	COU
X	L	X	X	X	X					L	L	L	L	X
X	H	L	X	X	X	d	c	b	a	D	c	b	a	X
↑	H	H	L	X	X	d	c	B	a	D	c	b	a	X
↑	H	H	H	L	X					Hold				X
↑	H	H	H	H	H					Count Down				L
↑	H	H	H	H	L					Count Up				L
X	X	X	X	H	H					L	L	L	L	H
X	X	X	X	H	L					H	H	H	H	H



Truth Table of 7490 (Counter)

Inputs			Outputs			
CLR*	SET*	CLK	QD	QC	QB	QA
H	L	X	L	L	L	L
L	H	X	H	L	L	H
H	H	X	Illegal			
L	L		Count			

*CLR = CLRA & CLRB.

**SET = SET9A & SET9B.

Possible Counting Configurations:

DECADE: QA connected to CLKB

Count	QD	QC	QB	QA
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	L	H	L	H
6	L	H	H	L
7	L	H	H	H
8	H	L	L	L
9	H	L	L	H

BI-QUINARY: QD connected to CLKA

Count	QD	QC	QB	QA
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	H	L	L	L
6	H	L	L	H



7	H	L	H	L
8	H	L	H	H
9	H	H	L	L

Binary: This mode can be an inefficient implementation of a binary counter and is not recommended by ALTERA. For a four-bit counter with similar features, use the 74161 or 4count macrofunction.

Truth Table of 74161 (Counter)

Inputs									Outputs				
CLK	LDN	CLRn	ENP	ENT	D	C	B	A	QD	QC	QB	QA	RC0
X	X	L	X	X					L	L	L	L	L
↑	L	H	X	X	d	C	b	a	D	c	b	a	*
↑	H	H	X	L					QD	QC	QB	QA	*
↑	H	H	L	X					QD	QC	QB	QA	*
↑	H	H	H	H					L	L	L	L	L
↑	H	H	H	H					L	L	L	H	L
↑	H	H	H	H					L	L	H	L	L
↑	H	H	H	H					L	L	H	H	L
↑	H	H	H	H					L	H	L	L	L
↑	H	H	H	H					L	H	L	H	L
↑	H	H	H	H					L	H	H	L	L
↑	H	H	H	H					L	H	H	H	L
↑	H	H	H	H					H	L	L	L	L
↑	H	H	H	H					H	L	L	H	L
↑	H	H	H	H					H	L	H	L	L
↑	H	H	H	H					H	L	H	H	L
↑	H	H	H	H					H	H	L	L	L
↑	H	H	H	H					H	H	L	H	L
↑	H	H	H	H					H	H	H	L	L
X	H	H	H	H					H	H	H	H	H

* RCO = QD & QC & QB & QA & ENT.



Registers		
Macrofunction	Description	Function Prototype
ENADFF	Enabled D-Type Flip-flop	FUNCTION ENADFF (d, clk, clrn, prn, ena) RETURNS (q);
XPDFFE	D-Type Flip-flop Implemented with Expanders (or with DFF Primitive for FLEX 8000 Projects)	FUNCTION EXPDFF (d, clk, clrn, prn) RETURNS (q, /q);
7470	AND-Gated JK Flip-flop with Preset and Clear	FUNCTION 7470 (prn, clrn, clk, j1, j2, jn, k1, k2, kn) RETURNS (q, qn);
7471	JK Flip-flop with Preset	FUNCTION 7471 (prn, clk, j1a, j1b, j2a, j2b, k1a, k1b, k2a, k2b) RETURNS (q, qn);
7472	AND-Gated JK Flip-flop with Preset and Clear	FUNCTION 7472 (prn, clrn, clk, j1, j2, j3, k1, k2, k3) RETURNS (q, qn);
7473	Dual JK Flip-flop with Clear	FUNCTION 7473 (1clrn, 1clk, 1j, 1k, 2clrn, 2clk, 2j, 2k) RETURNS (1q, 1qn, 2q, 2qn);
7474	Dual D-Type Flip-flop with Asynchronous Preset and Asynchronous Clear	FUNCTION 7474 (1prn, 1clrn, 1clk, 1d, 2prn, 2clrn, 2clk, 2d) RETURNS (1q, 1qn, 2q, 2qn);
7476	Dual JK Flip-flop with Asynchronous Preset and Asynchronous Clear	FUNCTION 7476 (1prn, 1clrn, 1clk, 1j, 1k, 2prn, 2clrn, 2clk, 2j, 2k) RETURNS (1q, 1qn, 2q, 2qn);



7478	Dual JK Flip-flop with Asynchronous Preset, Common Clear, and Common Clock	FUNCTION 7478 (clrn, 1prn, 1j, 1k, 2prn, 2j, 2k, clk) RETURNS (1q, 1qn, 2q, 2qn);
74107	Dual JK Flip-flop with Clear	FUNCTION 74107 (1j, 1k, 1clrn, 1clk, 2j, 2k, 2clrn, 2clk) RETURNS (1q, 1qn, 2q, 2qn);
74109	Dual JK Flip-flop with Preset and Clear	FUNCTION 74109 (1prn, 1j, 1kn, 1clrn, 1clk, 2prn, 2j, 2kn, 2clrn, 2clk) RETURNS (1q, 1qn, 2q, 2qn);
74112	Dual JK Negative-Edge-Triggered Flip-flop with Preset and Clear	FUNCTION 74112 (1prn, 1j, 1k, 1clrn, 1clk, 2prn, 2j, 2k, 2clrn, 2clk) RETURNS (1q, 1qn, 2q, 2qn);
74113	Dual JK Negative-Edge-Triggered Flip-flop with Preset	FUNCTION 74113 (1prn, 1j, 1k, 1clk, 2prn, 2j, 2k, 2clk) RETURNS (1q, 1qn, 2q, 2qn);
74114	Dual JK Negative-Edge-Triggered Flip-flop with Preset, Common Clear, and Common Clock	FUNCTION 74114 (1prn, 1j, 1k, 1clk, 2prn, 2j, 2k, clrn, clk) RETURNS (1q, 1qn, 2q, 2qn);
74171	Quad D-Type Flip-flops with Clear	FUNCTION 74171 (clrn, clk, d1, d2, d3, d4) RETURNS (q1, qn1, q2, qn2, q3, qn3, q4, qn4);
74172	Multi-Port Register File with Tri-State Outputs	FUNCTION 74172 (1grn, 2grn, 1r0, 1r1, 1r2, 2w/r0, 2w/r1, 2w/r2, 1w0, 1w1, 1w2, 1da, 1db, 2da, 2db, 1gwn, 2gwn, clk) RETURNS (1qa, 1qb, 2qa, 2qb);
74173	4-Bit D-Type Re-	FUNCTION 74173 (clr, clk, mn, nn, g1n,



	gister	g2n, 1d, 2d, 3d, 4d) RETURNS (1q, 2q, 3q, 4q);
74174	Hex D-Type Flipflop with Common Clear	FUNCTION 74174 (clrn, clk, 1d, 2d, 3d, 4d, 5d, 6d) RETURNS (1q, 2q, 3q, 4q, 5q, 6q);
74174B	Hex D-Type Flipflop with Common Clear	FUNCTION 74174B (clrn, clk, d[6..1]) RETURNS (q[6..1]);
74175	Quad D-Type Flip flop with Common Clock and Clear	FUNCTION 74175 (clrn, clk, 1d, 2d, 3d, 4d) RETURNS (1q, 1qn, 2q, 2qn, 3q, 3qn, 4q, 4qn);
74273	Octal D-Type Flipflop with Asynchronous Clear	FUNCTION 74273 (clrn, clk, d[8..1]) RETURNS (q[8..1]);
74273B	Octal D-Type Flip- flop with Asyn- chronous Clear	FUNCTION 74273B (clrn, clk, d[8..1]) RETURNS (q[8..1]);
74276	Quad J/K Flip-flop Register with Common Preset and Clear	FUNCTION 74276 (prn, clrn, 1j, 1kn, 1clk, 2j, 2kn, 2clk, 3j, 3kn, 3clk, 4j, 4kn, 4clk) RETURNS (1q, 1qn, 2q, 2qn, 3q, 3qn, 4q, 4qn);
74374	Octal D-Type Flip- flop with Tri-State Outputs and Output Enable	FUNCTION 74374 (clk, oen, d[8..1]) RETURNS (q[8..1]);
74374B	Octal D-Type Flip- flop with Tri-State Outputs and Output Enable	FUNCTION 74374B (clk, oen, d[8..1]) RETURNS (q[8..1]);
74376	Quad JK Flip-flop with Common Clock and Common Clear	FUNCTION 74376 (clk, clrn, 1j, 1kn, 2j, 2kn, 3j, 3kn, 4j, 4kn) RETURNS (1q, 2q, 3q, 4q);



74377	Octal D-Type Flip-flop with Enable	FUNCTION 74377 (en, clk, d[8..1]) RETURNS (q[8..1]);
74377B	Octal D-Type Flip-flop with Enable	FUNCTION 74377B (en, clk, d[8..1]) RETURNS (q[8..1]);
74378	Hex D-Type Flip-flop with Enable	FUNCTION 74378 (en, clk, d[6..1]) RETURNS (q[6..1]);
74379	Quad D-Type Flip-flop with Enable	FUNCTION 74379 (en, clk, d[4..1]) RETURNS (q[4..1], qn[4..1]);
74396	Octal Storage Register	FUNCTION 74396 (strbn, clk, d1, d2, d3, d4) RETURNS (1q1, 1q2, 1q3, 1q4, 2q1, 2q2, 2q3, 2q4);
74548	8-Bit 2-Stage Pipelined Register with Tri-State Outputs	FUNCTION 74548 (clk, clkenn1, clkenn2, insel, d[7..0], outsel, oen) RETURNS (y[7..0]);
74670	4-Bit by 4-Bit Register File with Tri-State Outputs	FUNCTION 74670 (wb, wa, gwn, rb, ra, grn, d[1..4]) RETURNS (q[1..4]);
74821	10-Bit Bus Interface Flip-flop with Tri-State Outputs	FUNCTION 74821 (oen, clk, d[1..10]) RETURNS (q[1..10]);
74821B	10-Bit D-Type Flip-flop with Tri-State Outputs	FUNCTION 74821B (d[10..1], oen, clk) RETURNS (q[10..1]);
74822	10-Bit Bus Interface Flip-flop with Tri-State Inverting Outputs	FUNCTION 74822 (oen, clk, d[1..10]) RETURNS (q[1..10]);
74822B	10-Bit D-Type Inverting Flip-flop with Tri-State Inverting Outputs	FUNCTION 74822B (dn[10..1], oen, clk) RETURNS (q[10..1]);



74823	9-Bit Bus Interface Flip-flop with Tri-State Outputs	FUNCTION 74823 (oen, clrn, clkenn, clk, d[1..9]) RETURNS (q[1..9]);
74823B	9-Bit D-Type Flip-flop with Tri-State Outputs	FUNCTION 74823 (oen, clrn, clkenn, clk, d[1..9]) RETURNS (q[1..9]);
74824	9-Bit Bus Interface Flip-flop with Tri-State Inverting Outputs	FUNCTION 74824 (oen, clrn, clkenn, clk, dn[1..9]) RETURNS (q[1..9]);
74824B	9-Bit D-Type Inverting Flip-flop with Tri-State Inverting Outputs	FUNCTION 74824B (dn[9..1], oen, clk, clrn, clkenn) RETURNS (q[9..1]);
74825	8-Bit Bus Interface Flip-flop with Tri-State Outputs	FUNCTION 74825 (oe1n, oe2n, oe3n, clrn, clkenn, clk, d[1..8]) RETURNS (q[1..8]);
74825B	Octal D-Type Flip-flop with Tri-State Outputs	FUNCTION 74825B (d[8..1], oe1n, oe2n, oe3n, clk, clrn, clkenn) RETURNS (q[8..1]);
74826	9-Bit Bus Interface Flip-flop with Tri-State Inverting Outputs	FUNCTION 74826 (oe1n, oe2n, oe3n, clrn, clkenn, clk, d[1..8]) RETURNS (q[1..8]);
74826B	Octal D-Type Inverting Flip-flop with Tri-State Inverting Outputs	FUNCTION 74826B (dn[8..1], oe1n, oe2n, oe3n, clk, clrn, clkenn) RETURNS (q[8..1]);



Truth Table of ENADFF (Register)

Inputs					Outputs
CLRN	PRN	ENA	D	CLK	Q
L	H	X	X	X	L
H	L	X	X	X	H
L	L	X	X	X	Illegal
H	H	L	X	X	Qo
H	H	H	L	↑	L
H	H	H	H	↑	H
H	H	X	X	L	Qo

Truth Table of 7474 (Register)

Inputs				Outputs	
PRN	CLRN	CLK	D	Q	QN
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	Illegal	
H	H	↑	L	L	H
H	H	↑	H	H	L
H	H	L	X	Qo*	/Qo

*Qo = Level of Q before clock pulse

Truth Table of 74374 (Register)

Inputs			Outputs
OEN	CLK	D	Q
H	X	X	Z
L	X	X	X
L	↑	L	L
L	↑	H	H
L	L	X	Qo



Decoders		
Macrofunction	Description	Function Prototype
16DMUX	4-Bit Binary-to-16-Line Decoder	FUNCTION 16DMUX (d, c, b, a) RETURNS (q[15..0])
16NDMUX	4-Bit Binary-to-16-Line Decoder	FUNCTION 16NDMUX (d, c, b, a) RETURNS (qn[15..0]);
7442	1-Line-to-10-Line BCD-to-Decimal Decoder	FUNCTION 7442 (d, c, b, a) RETURNS (o0n, o1n, o2n, o3n, o4n, o5n, o6n, o7n, o8n, o9n);
7443	Excess-3-to-Decimal Decoder	FUNCTION 7443 (d, c, b, a) RETURNS (o0n, o1n, o2n, o3n, o4n, o5n, o6n, o7n, o8n, o9n);
7444	Excess-3-Gray-to-Decimal Decoder	FUNCTION 7444 (d, c, b, a) RETURNS (o0n, o1n, o2n, o3n, o4n, o5n, o6n, o7n, o8n, o9n);
7445	BCD-to-Decimal Decoder	FUNCTION 7445 (d, c, b, a) RETURNS (o0n, o1n, o2n, o3n, o4n, o5n, o6n, o7n, o8n, o9n);
7446	BCD-to-7-Segment Decoder	FUNCTION 7446 (ltn, rbin, d, c, b, a, bin) RETURNS (oa, ob, oc, od, oe, 'of', og, rbon);
7447	BCD-to-7-Segment Decoder	FUNCTION 7447 (ltn, rbin, d, c, b, a, bin) RETURNS (oa, ob, oc, od, oe, 'of', og, rbon);
7448	BCD-to-7-Segment Decoder	FUNCTION 7448 (ltn, rbin, d, c, b, a, bin) RETURNS (oa, ob, oc, od, oe, 'of', og, rbon);
7449	BCD-to-7-Segment Decoder	FUNCTION 7449 (d, c, b, a, bin) RETURNS (oa, ob, oc, od, oe, 'of', og);
74137	3-Line-to-8-Line Decoder with Address Latches	FUNCTION 74137 (gln, g1, g2n, c, b, a) RETURNS (y[0..7]);
74138	3-Line-to-8-Line Decoder	FUNCTION 74138 (g1, g2an, g2bn, c, b, a) RETURNS (y0n, y1n, y2n, y3n, y4n, y5n, y6n, y7n);



74139	Dual 2-Line-to-4-Line Decoder	FUNCTION 74139 (g1n, b1, a1, g2n, b2, a2) RETURNS (y10n, y11n, y12n, y13n, y20n, y21n, y22n, y23n);
74145	BCD-to-Decimal Decoder	FUNCTION 74145 (d, c, b, a) RETURNS (o0n, o1n, o2n, o3n, o4n, o5n, o6n, o7n, o8n, o9n);
74154	4-Line-to-16-Line Decoder	FUNCTION 74154 (g1n, g2n, d, c, b, a) RETURNS (o0n, o1n, o2n, o3n, o4n, o5n, o6n, o7n, o8n, o9n, o10n, o11n, o12n, o13n, o14n, o15n);
74155	Dual 2-Line-to-4-Line Decoder/Demultiplexer	FUNCTION 74155 (2cn, 1c, selb, sela, 2gn, 1gn) RETURNS (2y0n, 2y1n, 2y2n, 2y3n, 1y0n, 1y1n, 1y2n, 1y3n);
74156	Dual 2-Line-to-4-Line Decoder/Demultiplexer	FUNCTION 74156 (2cn, 1c, selb, sela, 2gn, 1gn) RETURNS (2y0n, 2y1n, 2y2n, 2y3n, 1y0n, 1y1n, 1y2n, 1y3n);
74246	BCD-to-7-Segment Decoder	FUNCTION 74246 (ltn, rbin, bin, d, c, b, a) RETURNS (oa, ob, oc, od, oe, f', og, rbon);
74247	BCD-to-7-Segment Decoder	FUNCTION 74247 (ltn, rbin, bin, d, c, b, a) RETURNS (oa, ob, oc, od, oe, f', og, rbon);
74248	BCD-to-7-Segment Decoder	FUNCTION 74248 (ltn, rbin, bin, d, c, b, a) RETURNS (oa, ob, oc, od, oe, f', og, rbon);
74445	BCD-to-Decimal Decoder	FUNCTION 74445 (d, c, b, a) RETURNS (o0n, o1n, o2n, o3n, o4n, o5n, o6n, o7n, o8n, o9n);



Truth Table of 16DMUX (Decoder)

Inputs				Outputs							
D	C	B	A	Q15	Q14	Q13	...	Q3	Q2	Q1	Q0
L	L	L	L	L	L	L	...	L	L	L	H
L	L	L	H	L	L	L	...	L	L	H	L
L	L	H	L	L	L	L	...	L	H	L	L
L	L	H	H	L	L	L	...	H	L	L	L
L	H	L	L	L	L	L	...	L	L	L	L
L	H	L	H	L	L	L	...	L	L	L	L
L	H	H	L	L	L	L	...	L	L	L	L
L	H	H	H	L	L	L	...	L	L	L	L
H	L	L	L	L	L	L	...	L	L	L	L
H	L	L	H	L	L	L	...	L	L	L	L
H	L	H	L	L	L	L	...	L	L	L	L
H	L	H	H	L	L	L	...	L	L	L	L
H	H	L	L	L	L	L	...	L	L	L	L
H	H	L	H	L	L	H	...	L	L	L	L
H	H	H	L	L	H	L	...	L	L	L	L
H	H	H	H	H	L	L	...	L	L	L	L

Truth table of 74138 (Decoder)

Inputs					Outputs							
Enable		Select										
G1	G2*	C	B	A	Y0N	Y1N	Y2N	Y3N	Y4N	Y5N	Y6N	Y7N
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

* $G2 = G2AN + G2BN$.



Truth Table of 74246 (Decoder)

Inputs							Outputs							
LTN	RBIN	BIN	D	C	B	A	OA	OB	OC	OD	OE	OF	OG	RBON
H	H	H	L	L	L	L	On	On	On	On	On	On	Off	H
H	L	H	L	L	L	L	Off	Off	Off	Off	Off	Off	Off	L
H	X	H	L	L	L	H	On	On	On	Off	Off	Off	Off	H
H	X	H	L	L	H	L	On	On	Off	On	On	Off	On	H
H	X	H	L	L	H	H	Off	On	On	On	Off	Off	On	H
H	X	H	L	H	L	L	On	On	On	Off	Off	On	On	H
H	X	H	L	H	L	H	On	Off	On	On	Off	On	On	H
H	X	H	L	H	H	L	On	Off	On	On	On	On	On	H
H	X	H	L	H	H	H	On	On	On	Off	Off	Off	Off	H
H	X	H	H	L	L	L	On	On	On	On	On	On	On	H
H	X	H	H	L	L	H	Off	On	On	On	Off	On	On	H
H	X	H	H	L	H	L	Off	Off	Off	On	On	Off	On	H
H	X	H	H	L	H	H	Off	Off	On	On	Off	Off	On	H
H	X	H	H	H	L	L	On	On	Off	Off	Off	On	On	H
H	X	H	H	H	L	H	Off	Off	Off	On	Off	On	On	H
H	X	H	H	H	H	L	Off	Off	Off	On	On	On	On	H
H	X	H	H	H	H	H	Off	Off	Off	Off	Off	Off	Off	H
L	X	H	X	X	X	X	On	On	On	On	On	On	On	H
H	X	L	X	X	X	X	Off	Off	Off	Off	Off	Off	Off	H

Shift Registers

Macrofunction	Description	Function Prototype
BARRELST	8-Bit Barrel Shifter	FUNCTION BARRELST (s[2..0], ldst, a, b, c, d, e, f, g, h, clk) RETURNS (qa, qb, qc, qd, qe, qf, qg, qh);
BARRLSTB	8-Bit Barrel Shifter	FUNCTION BARRLSTB (s[2..0], ldst, d[7..0], clk) RETURNS (q[7..0]);
7491	Serial-In Serial-Out Shift Register	FUNCTION 7491 (clk, a, b) RETURNS (qh, qhn);



7494	4-Bit Shift Register with Asynchronous Preset and Asynchronous Clear	FUNCTION 7494 (p1a, p2a, p1b, p2b, p1c, p2c, p1d, p2d, pe1, pe2, clr, clk, ser) RETURNS (out);
7495	4-Bit Parallel-Access Shift Register	FUNCTION 7495 (mode, clk1, clkr, ser, d[0..3]) RETURNS (q[0..3]);
7496	5-Bit Shift Register	FUNCTION 7496 (clrn, pe, a, b, c, d, e, clk, ser) RETURNS (qa, qb, qc, qd, qe);
7499	4-Bit Shift Register with /JK Serial Inputs and Parallel Outputs	FUNCTION 7499 (mode, clk2, clk1, j, kn, a, b, c, d) RETURNS (qa, qb, qc, qd, qdn);
74164	Serial-In Parallel-Out Shift Register	FUNCTION 74164 (clk, clrn, a, b) RETURNS (qa, qb, qc, qd, qe, qf, qg, qh);
74164B	Serial-In Parallel-Out Shift Register	FUNCTION 74164B (clk, clrn, a, b) RETURNS (q[7..0]);
74165	Parallel Load 8-Bit Shift Register	FUNCTION 74165 (clk, clk1h, stld, ser, a, b, c, d, e, f, g, h) RETURNS (qh, qhn);
74165B	Parallel Load 8-Bit Shift Register	FUNCTION 74165B (clk, clk1h, stld, ser, d[7..0]) RETURNS (q7, q7n);
74166	8-Bit Shift Register with Clock Inhibit	FUNCTION 74166 (clrn, stld, clk1h, clk, ser, a, b, c, d, e, f, g, h) RETURNS (qh);
74178	4-Bit Shift Register	FUNCTION 74178 (st, ld, ser, clk, a, b, c, d) RETURNS (qa, qb, qc, qd);
74179	4-Bit Shift Register with Clear	FUNCTION 74179 (clrn, st, ld, clk, ser, a, b, c, d) RETURNS (qa, qb, qc, qd, qdn);



74194	4-Bit Bidirectional Shift Register with Parallel Load	FUNCTION 74194 (clrn, s1, s0, clk, slsi, srsi, d, c, b, a) RETURNS (qd, qc, qb, qa);
74195	4-Bit Parallel-Access Shift Register	FUNCTION 74195 (clrn, st/ldn, clk, j, kn, d[0..3]) RETURNS (q[0..3], q3n);
74198	8-Bit Bidirectional Shift Register	FUNCTION 74198 (clrn, s1, s0, clk, slsi, srsi, a, b, c, d, e, f, g, h) RETURNS (qa, qb, qc, qd, qe, qf, qg, qh);
74199	8-Bit Parallel-Access Shift Register	FUNCTION 74199 (clrn, st/ldn, clk, j, kn, d[0..7]) RETURNS (q[0..7]);
74295	4-Bit Right-Shift Left-Shift Register with Tri-State Outputs	FUNCTION 74295 (oe, ld/shn, clk, ser, d[3..0]) RETURNS (q[3..0]);
74299	8-Bit Universal Shift/Storage Register	FUNCTION 74299 (clrn, s1, s0, g1n, g2n, clk, sr, sl) RETURNS (a/qa, b/qb, c/qc, d/qd, e/qe, f/qf, g/qg, h/qh, qa2, qh2);
74350	4-Bit Shift Register with Tri-State Outputs	FUNCTION 74350 (oen, s0, s1, d-[3..1], d[0..3]) RETURNS (y[0..3]);
74395	4-Bit Cascadable Shift Register with Tri-State Outputs	FUNCTION 74395 (clrn, ld/shn, clk, ser, d[1..4], oen) RETURNS (q[1..4], q4b);
74589	8-Bit Shift Register with Input Latches and Tri-State Output	FUNCTION 74589 (oen, srclk, ser, srl, rclk, d[0..7]) RETURNS (qhn);
74594	8-Bit Shift Register with Output Latches	FUNCTION 74594 (srclrn, rclrn, srclk, rclk, ser) RETURNS (qa, qb, qc, qd, qe, qf, qg, qh, qhn);



74595	8-Bit Shift Register with Output Latches and Tri-State Outputs	FUNCTION 74595 (gn, srclrn, srclk, rclk, ser) RETURNS (qa, qb, qc, qd, qe, qf, qg, qh, qhn);
74597	8-Bit Shift Register with Input Register	FUNCTION 74597 (srclrn, srlrn, rclk, d[7..0], srclk, ser) RETURNS (qhn);
74671	4-Bit Universal Shift Register/Latch with Direct-Overriding Clear and Tri-State Outputs	FUNCTION 74671 (gn, r/sn, srclrn, s1, s0, srclk, serl, serr, a, b, c, d, rclk) RETURNS (qa, qb, qc, qd, casc);
74672	4-Bit Universal Shift Register/Latch with Synchronous Clear and Tri-State Outputs	FUNCTION 74672 (gn, r/sn, srclrn, s1, s0, srclk, serl, serr, a, b, c, d, rclk) RETURNS (qa, qb, qc, qd, casc);
74673	16-Bit Shift Register	FUNCTION 74673 (csn, r/wn, srclk, stclrn, mode, ser) RETURNS (y[0..15], q15);
74674	16-Bit Shift Register	FUNCTION 74674 (csn, r/wn, clk, mode, ser, p[15..0]) RETURNS (q15);



Truth Table of BARRELST (Shift Register)

Inputs						Outputs		
S2	S1	S0	LDST	A..H	CLK	QA	QB..QG	QH
X	X	X	X	X	L	Qao	Qbo..Qgo	QHo
X	X	X	H	a..h	↑	a	b..g	h
L	L	L	L	X	↑	QAn	QBn..QGn	QHn
L	L	H	L	X	↑	QBn	QCn..QHn	QAn
L	H	L	L	X	↑	Qcn	QDn..QAn	QBn
L	H	H	L	X	↑	Qdn	QEn..QBn	QCn
H	L	L	L	X	↑	QEn	QFn..QCn	QDn
H	L	H	L	X	↑	Qfn	QGn..QDn	QEn
H	H	L	L	X	↑	QGn	QHn..QEn	QFn
H	H	H	L	X	↑	Qhn	Qan..QFn	QGn

Truth Table of 74164 (Shift Register)

Inputs				Outputs*	
CLK	CLRn	A	B	QA	QB..QH
X	L	X	X	L	QBo..QHo
L	H	X	X	Qao	QAn..QGn
L > H	H	H	H	H	QAn..QGn
L > H	H	L	X	L	QAn..QGn
L > H	H	X	L	L	QAn..QGn

* QAn, QGn = level of QA or QG before the most recent transition of the Clock; indicates a one-bit shift.



Truth Table of 74165 (Shift Register)

Inputs					Outputs			
CLK	CLK1H	STLD	SER	Parallel	Internal		Output	
				A..H	QA	QB	QH	QHN
X	X	L	X	a..h	a	B	h	/h
L	L	H	X	X	QAo	Qbo	Qho	/QGn
↑	L	H	H	X	H	QAn*	QGn*	
↑	L	H	L	X	L	Qan	QGn	/QGn
X	H	H	X	X	QAo	QBo	QHo	/QHo

* QAn, QGn = level of QA or QG before the most recent ↑ transition of the Clock.

Digital Filters		
Macrofunction	Description	Function Prototype
74297	Digital Phase-Locked Loop Filter	FUNCTION 74297 (kclk, d/upn, enctr, d, c, b, a, idclk, phase_a1, phase_b, phase_a2) RETURNS (idout, ecpd, xorpd);

Truth Table of 74297 (Digital Filter)

K Counter Function Table

Inputs				Outputs
D	C	B	A	Modulo (K)
L	L	L	L	Inhibited
L	L	L	H	2 ³
L	L	H	L	2 ⁴
L	L	H	H	2 ⁵
L	H	L	L	2 ⁶
L	H	L	H	2 ⁷
L	H	H	L	2 ⁸



L	H	H	H	2^9
H	L	L	L	2^{10}
H	L	L	H	2^{11}
H	L	H	L	2^{12}
H	L	H	H	2^{13}
H	H	L	L	2^{14}
H	H	L	H	2^{15}
H	H	H	L	2^{16}
H	H	H	H	2^{17}

XOR Phase Detector

Inputs		Outputs XORPD
Phase_A1	Phase_B	
L	L	L
L	H	H
H	L	H
H	H	L

Edge-Controlled Phase Detector

Inputs		Outputs ECPD
Phase_A2	Phase_B	
H or L	↓	H
↓	H or L	L
H or L	↑	No Change
↑	H or L	No Change



Storage Registers		
Macrofunction	Description	Function Prototype
7498	4-Bit Data Selector/Storage Register	FUNCTION 7498 (clkn, wrdsl, a1, b1, c1, d1, a2, b2, c2, d2) RETURNS (qa, qb, qc, qd);
74278	4-Bit Cascadable Priority Register	FUNCTION 74278 (p0, g, d[4..1]) RETURNS (y[4..1], p1);

Truth Table of 7498 (Storage Register)

Inputs										Outputs			
CLKN	WRDSL	A1	B1	C1	D1	A2	B2	C2	D2	QA	QB	QC	QD
↓	L	a1	b1	c1	d1	X	X	X	X	A1	b1	c1	d1
↓	H	X	X	X	X	a2	b2	c2	d2	A2	b2	c2	d2

Truth Table of 74278 (Storage Register)

Inputs		Internal Latch Nodes				Outputs								
P0	G	D1	D2	D3	D4	/Q1	/Q2	/Q3	/Q4	Y1	Y2	Y3	Y4	P1
L	H	H	X	X	X	L	X	X	X	H	L	L	L	H
L	H	L	H	X	X	H	L	X	X	L	H	L	L	H
L	H	L	L	H	X	H	H	L	X	L	L	H	L	H
L	H	L	L	L	H	H	H	H	L	L	L	L	H	H
L	H	L	L	L	L	H	H	H	H	L	L	L	L	L
L	L	X	X	X	X	Latched when G goes low					Same function of /Q as on first 5 lines			
H	L	X	X	X	X						L	L	L	H
H	H	Internal /Q levels are same function of D inputs as on the first 5 lines.									L	L	L	H



EDAC		
Macrofunction	Description	Function Prototype
74630	16-Bit Parallel Error Detection and Correction Circuit	FUNCTION 74630 (s1, s0, db[15..0], cb[5..0]) RETURNS (dbo[15..0], cbo[5..0], sef, def);
74636	8-Bit Parallel Error Detection and Correction Circuit	FUNCTION 74636 (s1, s0, db[7..0], cb[4..0]) RETURNS (dbo[7..0], cbo[4..0], sef, def);

Truth Table of 74636 (EDAC)

Memory Cycle	Control		EDAC Function	Data I/O	Check Word I/O	Error Flags	
	S1	S2				SEF	DEF
Write	L	L	Generate Check Word	Inp Data	Output Check Word	L	L
Read	L	H	Read Data & Check Word	Inp Data	Input Check Word	L	L
Read	H	H	Flag Errors	Latch D	Latch Check Write	Enabled	
Read	H	L	Correct Data & Synd Bits	Cor Data	Syndrome Bits	Enabled	

Error Function Table

Total Number of Errors		Error Flags		Data Correction
Data Word	Check Word	SEF	DEF	
0	0	L	L	Not Applicable
1	0	H	L	Correction
0	1	H	L	Correction
1	1	H	H	Interrupt
2	0	H	H	Interrupt
0	2	H	H	Interrupt

Check-Word Bits are derived from parity bits as follows

Check Word	8-Bit Data Word*
CB0	DB0, DB1, DB3, DB4



CB1	DB0, DB2, DB3, DB5, DB6
CB2	DB1, DB2, DB4, DB5, DB7
CB3	DB0, DB1, DB2, DB6, DB7
CB4	DB3, DB4, DB5, DB6, DB7

* The five check bits are parity bits derived from the data bits listed.

Error Syndrome Table

Error Location	Syndrome Error Code				
	CB0	CB1	CB2	CB3	CB4
DB0	L	L	H	L	H
DB1	L	H	L	L	H
DB2	H	L	L	L	H
DB3	L	L	H	H	L
DB4	L	H	L	H	L
DB5	H	L	L	H	L
DB6	H	L	H	L	L
DB7	H	H	L	L	L
CB0	L	H	H	H	H
CB1	H	L	H	H	H
CB2	H	H	L	H	H
CB3	H	H	H	L	H
CB4	H	H	H	H	L
No Error	H	H	H	H	H

SSI Functions		
Macrofunction	Description	Function Prototype
CBUF	Complementary Buffer	FUNCTION CBUF (1) RETURNS (2, 3);
INHB	Inhibit Gate	FUNCTION INHB (2, 3) RETURNS (1);
7400	NAND2 Gate	FUNCTION 7400 (2, 3)



		RETURNS (1);
7402	NOR2 Gate	FUNCTION 7402 (2, 3) RETURNS (1);
7404	NOT Gate	FUNCTION 7404 (2) RETURNS (1);
7408	AND2 Gate	FUNCTION 7408 (2, 3) RETURNS (1);
7410	NAND3 Gate	FUNCTION 7410 (2, 3, 4) RETURNS (1);
7411	AND3 Gate	FUNCTION 7411 (2, 3, 4) RETURNS (1);
7420	NAND4 Gate	FUNCTION 7420 (2, 3, 4, 5) RETURNS (1);
7421	AND4 Gate	FUNCTION 7421 (2, 3, 4, 5) RETURNS (1);
7423	Dual 4-Input NOR Gate with Strobe	FUNCTION 7423 (1a, 1b, 1c, 1d, 1g, 2a, 2b, 2c, 2d, 2g) RETURNS (1y, 2y);
7425	Dual 4-Input NOR Gate With Strobe	FUNCTION 7425 (1a, 1b, 1c, 1d, 1g, 2a, 2b, 2c, 2d, 2g) RETURNS (1y, 2y);
7427	NOR3 Gate	FUNCTION 7427 (2, 3, 4) RETURNS (1);
7428	Quad 2-Input Positive NOR Buffer	FUNCTION 7428 (a1, b1, a2, b2, a3, b3, a4, b4) RETURNS (y1, y2, y3, y4);
7430	NAND8 Gate	FUNCTION 7430 (2, 3, 4, 5, 6, 7, 8, 9) RETURNS (1);
7432	OR2 Gate	FUNCTION 7432 (2, 3) RETURNS (1);
7437	Quad 2-Input Positive NAND	FUNCTION 7437 (1a, 1b, 2a, 2b, 3a, 3b, 4a, 4b)



	Buffer	RETURNS (1y, 2y, 3y, 4y);
7440	Dual 4-Input Positive NAND Buffer	FUNCTION 7440 (1a, 1b, 1c, 1d, 2a, 2b, 2c, 2d) RETURNS (1y, 2y);
7450	Dual 2-Wide 2- Input AND-OR- INVERT Gate	FUNCTION 7450 (1x, 1xn, 1a, 1b, 1c, 1d, 2a, 2b, 2c, 2d) RETURNS (1yn, 2yn);
7451	Dual AND-OR- INVERT Gate	FUNCTION 7451 (1a, 1b, 1c, 1d, 1e, 1f, 2a, 2b, 2c, 2d) RETURNS (1yn, 2yn);
7452	AND-OR Gate	FUNCTION 7452 (x, a, b, c, d, e, f, g, h, i) RETURNS (y);
7453	Expandable 4- Wide AND-OR- INVERT Gate	FUNCTION 7453 (xn, x, a, b, c, d, e, f, g, h) RETURNS (yn);
7454	4-Wide AND-OR- INVERT Gate	FUNCTION 7454 (a, b, c, d, e, f, g, h, i, j) RETURNS (yn);
7455	2-Wide, 4-Input AND-OR- INVERT Gate	FUNCTION (a, b, c, d, e, f, g, h) RETURNS (yn);
7464	4-2-3-2 Input AND-OR- INVERT Gate	FUNCTION 7464 (a, b, c, d, e, f, g, h, i, j, k) RETURNS (y);
7486	XOR Gate	FUNCTION 7486 (2, 3) RETURNS (1);
74133	13-Input NAND Gate	FUNCTION 74133 (2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14) RETURNS (1);
74134	12-Input NAND Gate with Tri- State Output	FUNCTION 74134 (2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, oen) RETURNS (1);
74135	Quad XOR/XNOR	FUNCTION 74135 (1a, 2b, 12c, 2a, 2b, 3a, 3b,



Encoders		
Macrofunction	Description	Function Prototype
74147	10-Line-to-4-Line BCD Encoder	FUNCTION 74147 (1n, 2n, 3n, 4n, 5n, 6n, 7n, 8n, 9n) RETURNS (dn, cn, bn, an);
74148	8-Line-to-3-Line Octal Encoder	FUNCTION 74148 (ein, 0n, 1n, 2n, 3n, 4n, 5n, 6n, 7n) RETURNS (a2n, a1n, a0n, gsn, eon);
74348	8-Line-to-3-Line Priority Encoder with Tri-State Outputs	FUNCTION 74348 (ei, d[0..7]) RETURNS (eo, gs, a[2..0]);

Truth Table of 74148 (Encoder)

[illegible]



Truth Table of 74348 (Priority Encoder)

Inputs									Outputs				
E1	D0	D1	D2	D3	D4	D5	D6	D7	E0	GS	A2	A1	A0
H	X	X	X	X	X	X	X	X	H	H	Z	Z	Z
L	H	H	H	H	H	H	H	H	H	L	Z	Z	Z
L	X	X	X	X	X	X	X	L	L	H	L	L	L
L	X	X	X	X	X	X	L	H	L	H	L	L	H
L	X	X	X	X	X	L	H	H	L	H	L	H	L
L	X	X	X	X	L	H	H	H	L	H	L	H	H
L	X	X	X	L	H	H	H	H	L	H	H	L	L
L	X	X	L	H	H	H	H	H	L	H	H	L	H
L	X	L	H	H	H	H	H	H	L	H	H	H	L
L	L	H	H	H	H	H	H	H	L	H	H	H	H

True/Complement I/O Elements		
Macrofunction	Description	Function Prototype
7487	4-Bit True/Complement I/O Element	FUNCTION 7487 (a[4..1], b, c) RETURNS (y[4..1]);
74265	Quad Complementary Output Elements	FUNCTION 74265 (1a, 2a, 2b, 3a, 3b, 4a) RETURNS (1w, 1yn, 2w, 2yn, 3w, 3yn, 4w, 4yn);



Truth Table of 7487 (True/Complement I/O Element)

Inputs		Outputs			
B	C	Y1	Y2	Y3	Y4
L	L	/A1	/A2	/A3	/A4
L	H	A1	A2	A3	A4
H	L	H	H	H	H
H	H	L	L	L	L

Truth Table of 74265 (True/Complement I/O Element)

Inputs	Outputs		Inputs		Outputs	
1A (4A)	1W(4W)	1YN(4YN)	2A(3A)	2B(3B)	2W(3W)	2YN(3YN)
L	L	H	L	X	L	H
H	H	L	X	L	L	H
			H	H	H	L