Modern ASIC Design

Dian Zhou (周电)
E. E. Department
The University of Texas at Dallas
USA
Chapter 4 Logic and Circuit Design

• Objectives
  – Combinational logics
  – Sequential logics
  – Finite state machine
  – Arithmetic circuits
  – Datapath
Introduction

• In the previous chapters we have studied how to develop a specification from a given application, and also discussed how to develop the architecture of the chip to be designed.

• Once the architecture is ready, a designer can move forward to design the logics and circuits as shown in the design flow in Figure 5-1.
Figure 5-1 Logic/circuit design phase in the design flow

- Combinational logic
- Sequential logic
- Finite State Machine
- Arithmetic components
- Datapath
• As concerning modern ASIC design, most logic components, if not all, are available in a cell library, and such predesigned logic cells are optimized for both the manufacturing process and circuit performance.

• Only a few logic components need to be designed to fit the special needs in a real ASIC design process.

• In most of the cases, a logical/circuit level design can be obtained automatically by using a commercial EDA synthesis tool.
  
  – The input to the synthesis tools is the RTL netlist and the output is the gate level netlist.

• The optimization algorithms inside the synthesis tools are very powerful and the resulting gate level netlist is in general very satisfactory.
The focus of this chapter is to provide basic knowledge of combinational and sequential logic circuits and to offer a reference to the commonly used logic blocks for the purpose of self-contained discussion.

- The material to be discussed should have already appeared in most entry-level digital circuitry courses.
- Readers familiar with the subject can skip over this chapter or just use it as a reference.
• Digital logics can be divided into two main classes: combinational and sequential logics.

• Combinational logic usually implements a Boolean expression, where the output is purely a function of the present input.

• In contrast to combinational logic, the typical feature of sequential logic is its memory mechanism, which can store previous logic values, also known as states.
  
  — The storage elements are commonly implemented by flip-flop or latch.
  
  — Thus the output of a sequential circuit is a combined function of the present input and the state of the circuit.
Combinational Logics

- Combinational logic circuits are composed of gates or inverters to execute a particular function and may have one or more outputs.

- They play a significant role as function blocks in the digital circuit, especially, arithmetic logic circuit (ALU) where mathematical calculations are performed like addition, subtraction and multiplication.

- The basic combinational logic circuits include, gate, decoder, encoder and multiplexor.

- Some other complex combinational building blocks like adder and multiplier are used in ALU unit.
Decoder

In an $n$-input decoder, one of $2^n$ possible output lines will be selected at a time. Each output is mapped to one possible logic combination of input value. Figure 5-2 shows a 2 to 4 decoder consisting of four AND gates and two inverters. $A$ and $B$ are the inputs and $Y_0$ to $Y_3$ are the outputs, respectively. In a decoder, only one output will be set to be true according to the specific input combination at the time. For example, when $AB = 00$, $Y_0$ will become 1 and will be selected as the current output. This decoder is formally represented in the following figure.
Figure 5-2 A 2 to 4 decoder: (a) graphic symbol, (b) truth table, and (c) circuit implementation.
A decoder with enable control capability

Figure 5-3 A 2 to 4 decoder with an enable input signal: (a) graphic symbol, (b) truth table, and (c) circuit implementation
• Structure of large decoder
- Decoder is used as address selection unit in memory design

Figure 5-5 Application of decoder in SRAM
Encoder

A binary encoder is designed to reduce the number of bits representing the transmitted information. For an encoder of $2^n$ inputs, it sends out $n$ outputs. It performs an operation opposite to that of a decoder. Figure 5-6 shows a 4 to 2 encoder, where $X_0$ to $X_3$ are inputs and $Y_0$, $Y_1$ are outputs. Each code at output indicates one input pattern where exactly one of the four inputs is 1 as shown in the truth table.
An encoder circuit

Figure 5-6 A 4 to 2 binary encoder: (a) graphic symbol, (b) truth table, and (c) circuit implementation
• Application: priority encoder

One application of this priority encoder is to manage the resource allocation. Suppose the binary code $Y_0Y_1$ represents the priority of users $X_0$, $X_1$, $X_2$ and $X_3$, respectively. The larger the number $Y_0Y_1$ is, the higher priority the corresponding user has. From the truth table, it can be seen that user $X_3$ has the highest priority “11” and $X_0$ has the lowest priority “00”. The system manager can grant the resource to the user’s request according to their priority if only one user can access the resource at any given time.
<table>
<thead>
<tr>
<th>$X_0$</th>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$Y_0$</th>
<th>$Y_1'$</th>
<th>$Z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(a)

(b)

Figure 5-7 A priority encoder: (a) truth table, and (b) circuit implementation
A multiplexer is a combinational digital block used for signal selection. Basically, a multiplexer has multiple inputs and only one output. Only one of the input signals will be selected out each time. One or more control signals are required to indicate which input is selected. Figure 5-8 shows the graphic symbol and truth table of a 2 to 1 multiplexer. $S$ is the selection signal. $A, B$ are the data input and $Z$ is the output. Whenever a selection signal is inserted, either $A$ or $B$ will be chosen and sent out as the output.

![Figure 5-8](image)

Figure 5-8 A 2 to 1 multiplexer: (a) truth table, and (b) logic symbol
- Multiplexer circuit

Figure 5-9 Transistor level structure of a multiplexer
• Gate level structure

Figure 5-10 Gate level structure of a multiplexer
• A 4 to 1 multiplexer

<table>
<thead>
<tr>
<th>$S_0$</th>
<th>$S_1$</th>
<th>$Z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$A$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$C$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$B$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$D$</td>
</tr>
</tbody>
</table>

(a)

Figure 5-11 A 4 to 1 multiplexer: (a) truth table, and (b) circuit structure
Computation in a digital system is carried out by arithmetic logic blocks.

There are basically four arithmetic operations: addition, subtraction, multiplication and division.

- All of them are executed by using the fundamental gates.

Their performance efficiency is critical in a digital system because they are used “heavily” for almost for all applications.

In the following, we will discuss the basic operation principles and different topologies of two most significant arithmetic combinational blocks:

- Adder and Multiplier.
Adder

- Circuit implementation of a full adder

Figure 5-16 Circuit implementation of a full adder
• Ripple Carry Adder

Figure 5-18 Circuit structure of a ripple carry adder
• Circuit implementation

Figure 5-19 Circuit implementation of a 4-bit ripple carry adder
• Carry look-ahead adder

Figure 5-20 The circuit implementation of a 4-bit carry look-ahead adder
Addition/Subtraction

In digital circuit design, subtraction can be performed by adding two operands with one of their value inverted. Thus, it is easy to combine these two operations, addition and subtraction, together in one particular block by adding some peripheral signal and circuits. One implementation is shown in Figure 5-24. A ripple carry adder is reconfigured to perform either addition or subtraction by adding a multiplexer and an inverter. If $C_0 = (\overline{Add}) / Sub$ equals 0, the whole circuit works as a normal adder. If $C_0 = (\overline{Add}) / Sub$ equals 1, one of the operands will be inverted and sent into the adder which then essentially operates subtraction. At the same time, the value of $C_0$ which may be either 1 or 0 will be added into the final result to perform the 2’s complement.
• Circuit structure

Figure 5-24 Circuit structure of addition/subtraction block
An alternative circuit structure

Figure 5-25 An alternative circuit structure for addition/subtraction block
Multiplier

• Multiplication is a complex operation in ALU which usually consumes large hardware resources and has a large propagation delay.

• The multiplier, in nature, is an arrangement of adder array (suppose we only use combinational circuit).

• Before digging into the details of the multiplier, let us look at the binary multiplication procedure.

  – Suppose we have two four-bits binary number $X$ and $Y$. A simple multiplication is performed by generating a set of partial products and summing them to get the final result. Each partial product is produced by multiplying $X$ with each bit of $Y$ and shifting the set of partial products with regard to the bit position of $Y$. 
• Figure 5-26 illustrates the multiplication procedure of two 4-bits binary numbers.

\[\begin{array}{c}
1 & 0 & 1 & 0 \\
1 & 0 & 1 & 1 \\
\hline
1 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 \\
\hline
1 & 1 & 0 & 1 & 1 & 1 & 0 \\
\end{array}\]

Figure 5-26 Multiplication
From the above graph, the multiplier should be designed to efficiently realize the function of generating and summing the partial products. Generally, the partial product generation can be achieved by simply applying a 2-input AND gate. Figure 5-27 is an example of how a line of partial products is generated.

Figure 5-27 Partial product generation
Sequential Logics

• In this section, we will focus on another class of circuit: sequential logic circuit.

• Why do we need the sequential circuit?
  – One intuitive way to answer this question is shown in Figure 5-32.

  • Suppose we have several combinational logic blocks with different functions, and the output of one block is the input of another one. Then how do we manage the data flow between the blocks, if each block needs different time to complete its computation?

  – A solution is to employ the memory circuit to store the instant logic outputs from the combinational blocks, and latch them to the other blocks in a synchronized manner controlled by clocks.
Clearly, the clock period should be longer than that needed for all combinational logic blocks to complete their computation, which is usually the slowest block in the circuit. In this way, large scale digital system can be practically built.

Figure 5-32 Structure of the sequential circuit
Latch and Flip-flop

• Latch and flip-flop are two fundamental sequential logic blocks which conduct data flow and operation synchronization.

• Both latch and flip-flop are based on a so-called bi-stable circuit. The bi-stable circuit is a type of circuit that has two stable states, each representing logic 0 or 1, respectively.

• It is widely used as the storage element in digital logic design. Figure 5-34 shows an example of such bi-stable circuit.
• A bi-stable circuit

Figure 5-34 Bi-stable circuit
Latch

- An SR Latch

![SR Latch Diagram](image)

**Figure 5-36** An SR latch with NOR gate implementation: (a) graphic symbol, (b) truth table, and (c) circuit implementation
Timing diagram

Figure 5-37 An SR latch timing diagram
• **Gated latch**

  – A gated latch usually refers to a latch with a controlled signal. The output of a latch will follow the immediate change of the input when the control signal is active.

  – In real practice, we want to know when the latch changes its state, and controls the transparent period for latch.

    • For example, in some applications, we just want to set the latch to be transparent during the period when the clock is high.

  – A clock can be used as a control signal.

    • Only when the clock is high, will the latch work in the transparent mode during which the change at input will be sent to the output. As the clock goes low, the output will remain the previous state no matter how the input changes.
• A gated SR Latch

Figure 5-39 A gated SR latch with NOR implementation: (a) graphic symbol, (b) truth table, and (c) circuit implementation
Figure 5-40 shows the timing diagram of the gated SR latch. The output changes its state only when the enable signal is active. For all other periods, the output will maintain its previous values.
Flip-flop

• The main problem with using latch is that it has a transparent period when the control signal is active. During this period the output will change with the input.
  – Consequently such a circuit is prone to noise and uncertainty of the input signals.

• To overcome this “flaw”, edge triggered flip-flop was designed.
• As we have discussed, latch is a level-sensitive storage element.
  
  — As long as latch operates on transparent period, the output will follow any immediate change of the input.

• In many practical implementations, we need our circuit to respond at the trigger point of clock, either at the rising edge or at the falling edge.

• Another important storage element, flip-flop is required in this situation.
One widely used flip-flop is the Master-slave D flip-flop. Figure 5-44 illustrates how the master-slave D flip-flop is realized.

- Two D-latches are connected in series and the clock of master latch is inverted and fed into the slave latch.
- During the period when the clock is high, the master latch is transparent and any change at input data will be sent and stored in the output of the master latch.
- The slave latch is disabled by the inverted clock signal during this time. After the clock transits from high to low, the slave latch goes into a transparent period. The previous data sampled in the output of the master latch will pass through the slave latch to the output.
• A DFF circuit

Figure 5-44 A Master-Slave D Flip-Flop
• DFF timing diagram

Figure 5-45 A master-slave D flip-flop timing diagram
Circuit implementation

Figure 5-46 A circuit implementation of Master-Slave D Flip-Flop
Timing issue

• In real practical circuit design, controlling propagation delay is an important factor in guaranteeing a correct logic value.
  
  — To ensure an expected stable output at a latch or a flip-flop, several timing parameters should be carefully designed.
• Figure 5-49 shows three critical timing regions for a D flip-flop.

  – The minimal time interval for the input data must be stable before the event of the clock so that the signal can be reliably sampled and is called the setup time.

  – The minimal time interval for the data to be stable after the event of the clock so that the signal can be reliably sampled is called the hold time.

  – Another important timing parameter is the clock-to-output delay or the propagation delay which is the delay from the event of clock to the change of the output.
Timing parameters of a D flip-flop

Figure 5-49 Timing parameters of a D flip-flop
Figure 5-50 shows how to design a proper clock cycle for a D flip-flop. To ensure the correct function for the circuit below, the clock cycle should meet the requirement:

\[ T \geq t_p + t_{comb} + t_{su} \]

where \( t_{su} \) is the setup time of the flip-flop, \( t_{comb} \) is the propagation delay of the combinational block and \( t_p \) is the propagation delay of the flip-flop. The minimum clock cycle should be larger than the sum of these three time parameters. This is shown in Figure 5-50.

For the hold time, it also needs to satisfy the following condition:

\[ t_{hold} \leq t_{comb} + t_p. \]

Thus, the data can be kept for a short time period for correct sampling before it is flushed by the new data.
- Timing design

Figure 5-50 Timing design
Shift Register

- Register usually refers to a group of coherent flip-flops used to store a set of \( n \)-bit wide data.

- It is an extremely significant block in sequential logic design since the stored data might be updated at every clock cycle and be used to keep a correct function of the whole circuit.

- In this section, we will mainly discuss the shift register and its application in digital logic design.
- A basic shift register

**Figure 5-51** A basic structure for shift register
• **SIPO**
  
  — The shift register can be modified into a serial in parallel out shift register (SIPO) by changing the method of accessing the output.
  
  — Figure 5-52 shows a circuit implementation and operation principle for the SIPO.
Figure 5-52 Serial in parallel out: (a) circuit implementation, (b) timing diagram, and (c) timing table
• **PISO**
  
  – In contrast to the two previous shift registers, the configuration of parallel in serial out (PISO) is more complicated since the PISO circuit should be able to switch between two operation modes: parallel loading and serial shifting.

  • Because both the serial and parallel inputs are required, another mode called switching input should also be included.
Figure 5-53 Parallel in Serial out (PISO)
Counter

• A counter is usually composed of a series of cascading flip-flops.

• As an indispensable element in digital logic circuitry, it serves a variety of different purposes.
  
    – It can be used to either record the time pulses for a particular event or to control the timing by setting up an intentional time interval.
• Synchronous Counter

  – A simple circuit implementation of a synchronous counter is shown in Figure 5-56.

  – All individual flip-flops in this specific structure change their state concurrently according to an instant clock signal.

  – Figure 5-57 gives the timing diagram of this synchronous counter.
Figure 5-56 A synchronous Counter
Figure 5-57 Timing diagram for synchronous counter
• With an enable control

Figure 5-58 Synchronous counter with enable input
Finite State Machine

• Earlier in this chapter as we introduced the sequential circuit, a security system was shown to illustrate an Finite State Machine (FSM).
  – In the flowing we discuss in detail how to design such an FSM step by step.
  – We will also show how to use FSM in the digital system design.
Block diagram of a basic sequential circuit

Figure 5-63 Block diagram of a basic sequential circuit
• Mealy State Machine
  – Output depends on the current state and input
• Moore State Machine
  – Output depends only on the current state

Figure 5-65 Moore State Machine
An FSM example

To introduce the design procedure for a FSM, we start by considering an example of a sequence detector. This circuit is designed to detect a particular binary sequence 101. Whenever there is a complete sequence 101, the output is equal to 1; otherwise, the output is equal to 0. In this design, each sequence 101 is considered as a separated one, i.e., for the input 10101 only the first 101 is considered the detected sequence. However, for the input 101101, there are two detected sequences.

The circuit has one input from which the binary stream is fed in and only one output. The state of the circuit changes at the leading edge of the clock signal. For example, consider the random binary sequence at the input as shown in Figure 5-66. The output becomes 1 only at $t_7$ when an exact 101 sequence is received, and at $t_{10}$ when another 101 sequence is received. In this situation, it is necessary to keep track of the state change of the circuit since the output is not entirely dependent on the current input.
\textbf{Clock} \hspace{0.5em} t_0 \hspace{0.5em} t_1 \hspace{0.5em} t_2 \hspace{0.5em} t_3 \hspace{0.5em} t_4 \hspace{0.5em} t_5 \hspace{0.5em} t_6 \hspace{0.5em} t_7 \hspace{0.5em} t_8 \hspace{0.5em} t_9 \hspace{0.5em} t_{10} \hspace{0.5em} t_{11} \\
\textbf{Input} \hspace{2em} 1 \hspace{0.5em} 1 \hspace{0.5em} 0 \hspace{0.5em} 0 \hspace{0.5em} 0 \hspace{0.5em} 1 \hspace{0.5em} 0 \hspace{0.5em} 1 \hspace{0.5em} 1 \hspace{0.5em} 0 \hspace{0.5em} 0 \hspace{0.5em} 1 \\
\textbf{Output} \hspace{2em} 0 \hspace{0.5em} 0 \hspace{0.5em} 0 \hspace{0.5em} 0 \hspace{0.5em} 0 \hspace{0.5em} 0 \hspace{0.5em} 0 \hspace{0.5em} 0 \hspace{0.5em} 1 \hspace{0.5em} 0 \hspace{0.5em} 0 \hspace{0.5em} 1 \hspace{0.5em} 0

Figure 5-66 Sequence at input and corresponding output

The best way to start designing an FSM is to draw a state diagram, which transforms the conceptual specification to a pictorial representation. It can help the designer to be clearer about the state transition under different inputs. In this particular case, it requires four states to represent all states in the circuit. We call them S0 – S4, respectively. The complete state diagram is shown in Figure 5-67 in which the transitions in different states are labeled by directional arcs.
• State diagram

Figure 5-67 State diagram
- **State transition table**

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State</th>
<th></th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$X = 0$</td>
<td>$X = 1$</td>
<td>$Z$</td>
</tr>
<tr>
<td>S0</td>
<td>S0</td>
<td>S1</td>
<td>0</td>
</tr>
<tr>
<td>S1</td>
<td>S2</td>
<td>S1</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>S0</td>
<td>S3</td>
<td>0</td>
</tr>
<tr>
<td>S3</td>
<td>S0</td>
<td>S1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 5-68 State table
- State coding

<table>
<thead>
<tr>
<th>Current State $Y_0Y_1$</th>
<th>Next State $X = 0$</th>
<th>Next State $X = 1$</th>
<th>Output $Z$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$Y_0Y_1$</td>
<td>$Y_0Y_1$</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>10</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>01</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 5-69 State assigned table
• Logic expression

\[
Y_0 = X \bar{y}_0 y_1 + X y_0 \bar{y}_1
\]

\[
Y_1 = X
\]

\[
Z = y_0 \bar{y}_1
\]

Figure 5-70 Logic expression
• Circuit implementation

Figure 5-71 Circuit implementation of sequence detector
**FSM Design Steps**

**Step 1:** Receive a design specification for the FSM and understand its requirement.

**Step 2:** Create a state diagram according to the given specification and identify the required inputs, outputs, states and all possible state transitions.

**Step 3:** Construct a state table and check for the redundancy states.

**Step 4:** Determine the number of variables to represent all the needed states and make a state assignment.

**Step 5:** Select the proper type of flip-flop and derive the logic expression.

**Step 6:** Implement the logic circuit.
• FSM as the control unit for a datapath circuit

Figure 5-79 An architecture in a microprocessor
Datapath structure

• A digital processor usually consists of three parts: controller, ALU and interface.
  
  – As is shown in Figure 5-79, the control block is the brain of the processor and it supervises and instructs each unit working cooperatively.
  
  – The interface is responsible for the communication between the processor and external devices.
  
  – The ALU block is the core part of the processor where most of the operations are executed, and it takes large hardware resources and the most part of the area.
• Datapath of an FIR processor
Summary

• In this chapter we summarized the basic combinational and sequential logics.

• In the discussion we have reviewed the most commonly used circuit blocks in terms of their function and structure.

• These circuits can be easily modified to fit most application needs.

• The materials discussed here usually appear in undergraduate digital circuit courses.

  — The intent of putting them here is for the self-completion of the discussion, and as a reference.
Homework

1. Explain the difference between combinational and sequential logics. Give an example.
2. Show that a JK flip-flop can be constructed using a T flip-flop and other logic gates.
3. Write a VHDL code that represents an eight-bit Johnson counter. Synthesize the code with CAD tools and give a timing simulation that shows the counting sequence.
4. Determine the number of gates needed to implement an eight-bit carry-look-ahead adder assuming that the maximum fan-in for the gates is four.
5. Design an ALU to be used in the implementation of the MSDAP.
6. Design a shifter register to be used as an implementation of the input functional block of the MSDAP.