

ECE 271: Chapter 3 Reading Report

Phi Luu

October 15th, 2018

1 Chapter Outline

This chapter covers the basics of sequential logic design and focuses on the functional and timing relationships between inputs and outputs of a sequential circuit. In this chapter, the authors will show how to build memory for sequential circuits and how to design synchronous logic as well as finite state machines. Finally, like chapter 2, this chapter will finish off with the timing concept and parallelism of sequential logic.

1.1 Introduction

Chapter 2 demonstrates how to analyze and design combinational logic. Recall that the outputs of combinational logic depend *only* on the current value of inputs. *Sequential* logic, on the other hand, has its outputs depending on *both* **current** and **previous** input values.

Due to this nature, sequential logic has memory. The memory may explicitly remember the previous inputs or may distill the values into *states* of the system. The state of a digital sequential circuit contains all the necessary past information to explain the future behavior of the circuit. Since memory is an essential component of sequential logic design, latches and flip-flops will be discussed in the following subsection.

1.2 Latches and Flip-Flops

The fundamental building block of memory is a *bistable* element, an element with two stable states. To further understand what a bistable element looks like, consider Figure 1

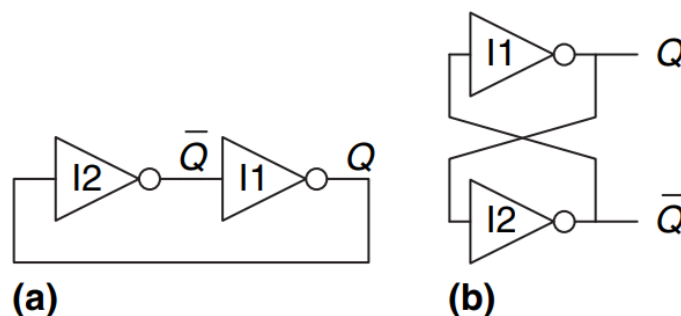


Figure 1: Cross-coupled inverter pair circuits

The circuits in Figure 1 do not have any input, instead the output of an inverter is the input of the other inverter. Consider two possible cases for the initial value of Q : TRUE and FALSE. Since the circuits are cyclic, one can start from anywhere on the circuits and still go back to the initial position. After going around the circuit, Q still holds the value TRUE if it was initially set to TRUE and still holds the value FALSE if it was initially set to FALSE. Therefore, Q is a bistable element.

When the power is first applied to the circuit, the initial state is unknown and thus becomes unpredictable. The cross-doubled inverters are not practical since the user has no inputs to control the state. There are alternative bistable elements that provide input control: *latches* and *flip-flops*.

1. **SR Latch** *SR latch* is composed of two cross-doubled NOR gates, as shown in Figure 2. The latch has two inputs, S and R , and has two outputs, Q and \bar{Q} .

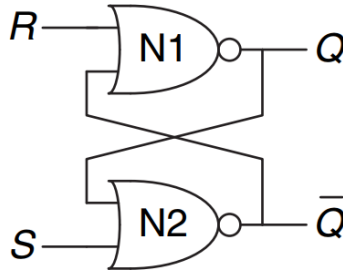


Figure 2: Schematic of an SR latch

The truth table of the SR latch can be solved by considering the inputs case by case.

- (a) $R = 0, S = 1$

When $S = 1$, N2 is always FALSE ($\bar{Q} = \text{FALSE}$). Thus, N1 can be expressed as $R \text{ NOR } \text{FALSE} = \text{FALSE NOR } \text{FALSE} = \text{TRUE}$, and so $Q = \text{TRUE}$.

- (b) $R = 1, S = 0$

Similarly to the first case, $Q = \text{FALSE}$ and $\bar{Q} = \text{TRUE}$

- (c) $R = 1, S = 1$

$Q = \bar{Q} = \text{FALSE}$, using the same deduction method as the first case.

- (d) $R = 0, S = 0$

When $R = S = 0$, the output of N1 depends on the output of N2 and vice versa. Thus, the value of Q and \bar{Q} cannot be solved unless the previous value of Q is known.

- i. $Q_{prev} = 0$

If $Q_{prev} = 0$, the output of N1 must be FALSE, which causes the second input of N2 to be 0. Thus, N2 will produce $\bar{Q} = \text{TRUE}$ as the output. This output matches with N1, since $\text{TRUE NOR } \text{FALSE} = \text{FALSE}$.

- ii. $Q_{prev} = 1$

Using the same deduction method by going around the circuit, $\bar{Q} = 0$.

Combined these two cases, if $R = 0$ and $S = 0$ then $Q = Q_{prev}$ and $\bar{Q} = \bar{Q}_{prev}$.

R	S	Q	\bar{Q}
0	0	Q_{prev}	\bar{Q}_{prev}
0	1	1	0
1	0	0	1
1	1	0	0

Table 1: Truth table of the SR latch

The truth table for the SR latch is shown in Table 1.

For the sake of abstraction and modularity, the SR latch is represented by the symbol in Figure 3

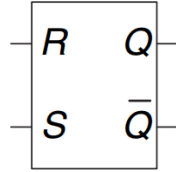


Figure 3: SR latch symbol

The truth table of the SR latch can be interpreted as follows: When R is asserted, the state is set to 0. When S is asserted, the state is set to 1. When both S and R are asserted, the state is set to 0. When neither R nor S is asserted, then the state retains its previous value.

2. D Latch

The D latch has two inputs: the *data* input D controlling what the next state should be and the *clock* input CLK controlling when the state should change. D and CLK then interact with each other via two two-input AND gates, and then connect in series with an SR latch, forming the D latch (see Figure 4).

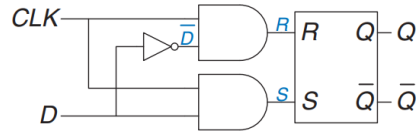


Figure 4: D latch schematic

After using the same case-by-case method as the SR latch, the truth table for the D latch is constructed per Table 2 below.

The CLK input (or the clock) controls when the data flows through the latch.

- When $CLK = 1$, the latch is *transparent*.
Data can flow through the latch. $Q = D$.
- When $CLK = 0$, the latch is *opaque*.
Data is blocked from flowing through the latch, thus the output retains its previous value.
 $Q = Q_{prev}$.

CLK	D	\bar{D}	R	S	Q	\bar{Q}
0	X	\bar{X}	0	0	Q_{prev}	\bar{Q}_{prev}
1	0	1	0	1	0	1
1	1	0	1	0	1	0

Table 2: D latch truth table

3. D Flip-Flop

A *D flip-flop* (also known as *master-slave flip-flop*, *edge-triggered flip-flop*, *positive edge-triggered flip-flop*) can be built from two back-to-back D latches controlled by complementary clocks, as shown in Figure 5. The first latch, *L1*, is called the *master*. The second latch, *L2*, is called the *slave*. The node between *L1* and *L2* is named *N1*.

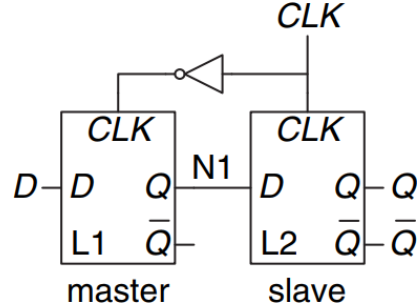


Figure 5: D flip-flop schematic

- When $CLK = 0$, the master is transparent and the slave is opaque. The value at D propagates to the node $N1$ (or $N1 = D$).
- When $CLK = 1$, the master is opaque and the slave is transparent. The (new) value at D is immediately blocked, and the value at $N1$ propagates to the output Q (or $Q = N1$).

In short, **D flip-flop copies D to Q on the rising edge of the clock, and remember its state at all other times.** This is the most important take-out of the D flip-flop.

Figure 6 shows the abstract symbol of the D flip-flop. When the \bar{Q} output is not needed, use the minimized symbol on the right.

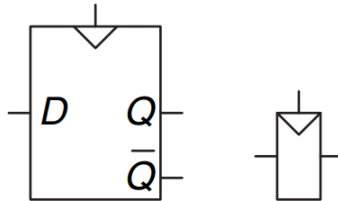


Figure 6: D flip-flop full symbol (left) and minimal symbol (right)

4. Register

An N -bit register is a bank of N flip-flops that share a common CLK input, so that all bits of the register are updated at the same time. Registers are the key building block of most sequential circuits. Figure 7 shows a 4-bit register and its symbol.

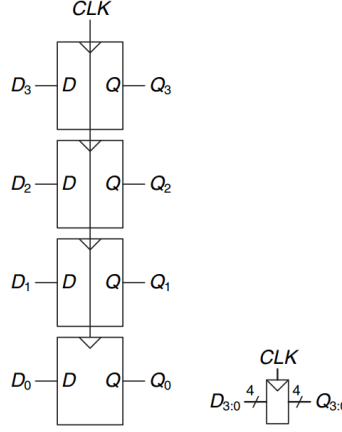


Figure 7: A 4-bit register's schematic (left) and symbol (right)

When the register's clock is on ($CLK = \text{TRUE}$), then its Q outputs will change according to their corresponding D inputs. Conversely, the register's Q outputs will retain their previous values if the clock is off ($CLK = \text{FALSE}$).

5. Enabled Flip-Flop

An *enabled flip-flop* is just a flip-flop that has an extra input EN or $ENABLE$ to determine whether data can be loaded to the flip-flop.

- When $EN = 1 = \text{TRUE}$, an enabled flip-flop behaves just like an ordinary flip-flop, i.e. $Q = Q_{prev}$ if $CLK = 0$ and $Q = D$ if $CLK = 1$.
- When $EN = 0 = \text{FALSE}$, an enabled flip-flop blocks the incoming data regardless of the CLK value.

The enabled flip-flop is useful when the user wants to load a new input value only some of the time rather than on every clock edge. Figure 8 shows the schematics and symbol of the enabled flip-flop.

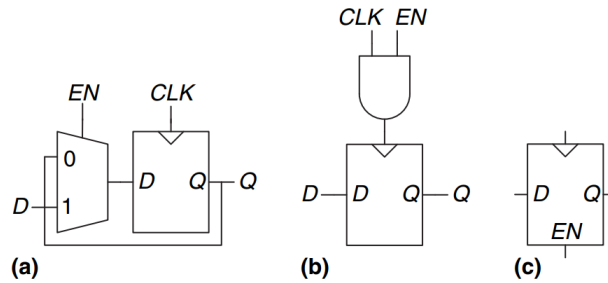


Figure 8: The enabled flip-flop schematics (a and b) and symbol (c)

6. Resettable Flip-Flop

A *resettable flip-flop* is just a flip-flop that has an extra input *RESET*.

- When $RESET = 1 = \text{TRUE}$, the resettable flip-flop ignores the input D and resets the output Q to 0.
- When $RESET = 0 = \text{FALSE}$, the resettable flip-flop behaves just like an ordinary flip-flop, i.e. $Q = Q_{prev}$ if $CLK = 0$ and $Q = D$ if $CLK = 1$.

Resettable flip-flops are useful when the user wants to force a known state (i.e. 0) into all the flip-flops in a system when we first turn it on. Figure 9 shows the schematics and symbol of the resettable flip-flop.

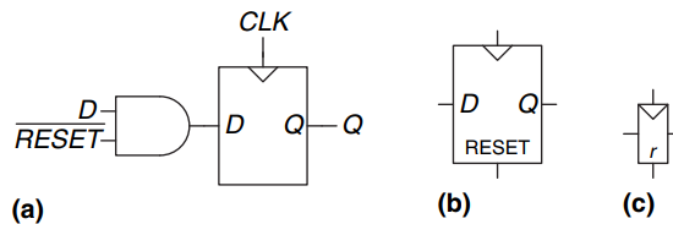


Figure 9: The resettable flip-flop schematics (a and b) and symbol (c)

7. Transistor-Level Latch and Flip-Flop Designs

The fundamental role of a latch is to be transparent or opaque, like a switch. Since latches and flip-flops require a large amount of transistors to implement when built from logic gates, transmission gates can be used to reduce the hardware needed.

8. Putting It All Together

Latches and flip-flops are the fundamental building blocks of sequential circuits. D-latch is level-sensitive, and D flip-flop is edge-triggered.

D-latch is transparent when $CLK = 1$, allowing D to flow through to Q . D flip-flop copies D to Q on the rising edge of CLK . At all other times, the outputs Q of both latch and flip-flop retain their previous value (old state).

A register, which is used in various sequential circuits, is a series of flip-flops, all connected to a single CLK bus.

1.3 Synchronous Logic Design

Sequential circuits are not the same as combinational circuits. The outputs in sequential circuits cannot be determined just by looking at the current values of the inputs. This subsection introduces the notion of synchronous sequential circuits and the dynamic discipline. These concepts will be useful in analyzing and designing sequential circuits.

1. Some Problematic Circuits

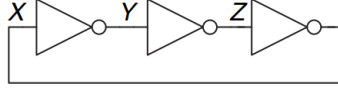


Figure 10: Three-inverter loop

The three-inverter loop (Figure 10) is an *astable* circuit because the value of X contradicts with its initial value after looping back from Z to X .

An example circuit below (Figure 11) is a sequential circuit with a *race condition*. A race condition causes the circuit to fail when some gates are slower than the others. In this case, if the inverter from CLK to \overline{CLK} has a longer delay than the AND and the OR gates, then Q will become stuck at 0. This is an example of *asynchronous* circuit design in which outputs are directly fed back to inputs.

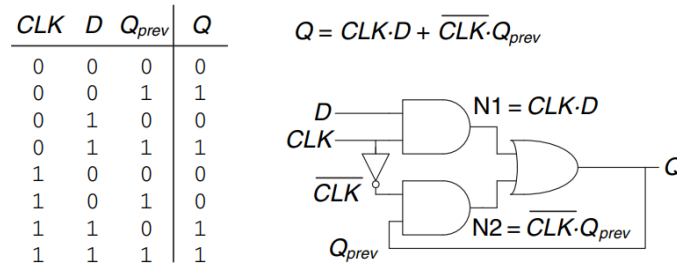


Figure 11: An asynchronous circuit with a race condition

2. Synchronous Sequential Circuits

To avoid the problems mentioned above, it is recommended to add registers somewhere on the cyclic path in the circuit. Since the registers force the state of the system to change only on clock edge, the state is *synchronized* to the clock. This method eliminates all races.

The rule of *synchronous sequential circuit composition* states that a circuit is a synchronous sequential circuit if it consists of the interconnected elements such that

- Every circuit element is either a register or a combinational circuit.
- At least one circuit element is a register.
- All registers receive the same clock signal.
- Every cyclic path contains at least one register.

From these criteria, the flip-flop is the simplest synchronous sequential circuit. Two other common types of synchronous sequential circuit are *finite state machines* and *pipelines*, which will be introduced in the next subsections.

3. Synchronous and Asynchronous Circuits

Different from synchronous design, *asynchronous design* is more general since the timing of the system is not blocked by registers (see the example sequential circuits in Figure 11). Asynchronous is necessary when communicating between systems with different clock or when receiving inputs at arbitrary times.

1.4 Finite State Machines

1. FSM Design Example

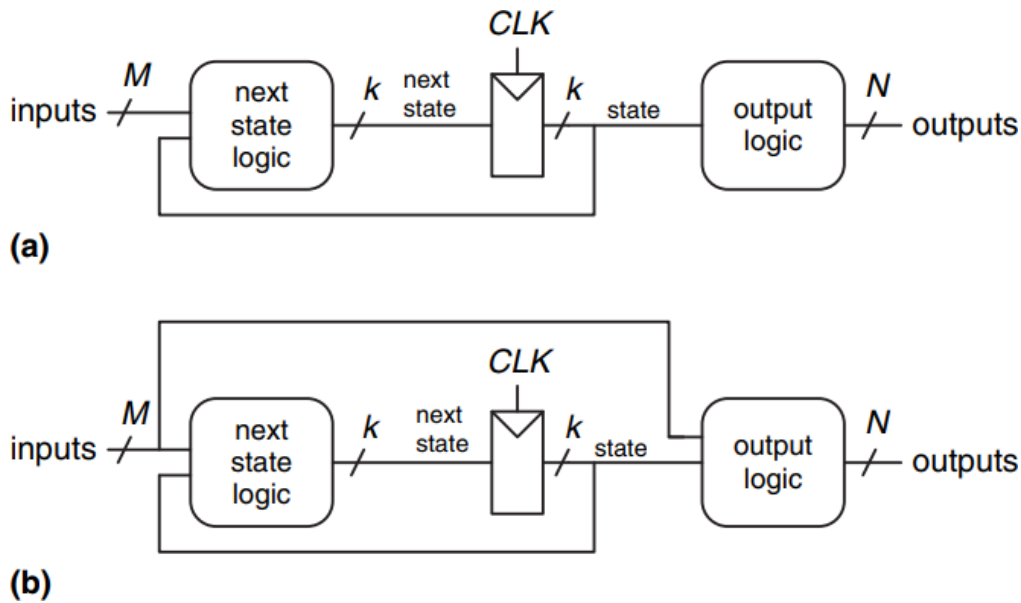


Figure 12: Finite state machines: (a) Moore machine, (b) Mealy machine

Synchronous sequential circuits can be drawn like in Figure 12. These forms are called *finite state machines (FSMs)*. An FSM has M inputs, N outputs, and k bits of state and receives a clock with an optional reset signal.

According to the example problem from the textbook, the steps to design an FSM are as follow:

- Draw a block diagram or a black box of the FSM
- Sketch the *state transition diagram* and complete it based on the problem statements
- Rewrite the transition diagram as a *state transition table*
- Express the states and the outputs in Boolean equations from the table using *binary encodings*. Note that everything has to be encoded in binary numbers
- Simplify the Booleans equations using Boolean algebra or Karnaugh maps
- Sketch and complete the circuit for the FSM

2. State Encodings

State and output encodings are selected arbitrary. Minimizing logic gates in a circuit requires the "best" encoding method. However, there is no simple way to find this "best" method as the number of states is large. In the end, the best encoding choice depends on the specific FSM.

3. Moore and Mealy Machines

On one hand, Moore machines are FSMs whose outputs depends only on the state of the system (outputs labeled inside the circles). On the other hand, Mealy machines are FSMs whose outputs depends on inputs as well as the current state (outputs labeled on the arc and outside of the circles).

4. Factoring State Machines

When designing complex FSMs, it is useful to break down a large FSM into multiple simpler state machines such that the output of some machines is the inputs of others. This method of hierarchy and modularity is called *factoring* of state machines.

5. Deriving an FSM from a Schematic

This process if necessary, for example, when taking on an incomplete documented project or reverse-engineering a system. Follow these guidelines to derive an FSM from a schematic:

- (a) Examine circuit, stating inputs, outputs, and state bits
- (b) Write next state and output equations
- (c) Create next state and output tables
- (d) Reduce the next state table to eliminate unreachable states
- (e) Assign each valid state bit combination a name
- (f) Rewrite next state and output tables with state names
- (g) Draw state transition diagram
- (h) State in words what the FSM does

6. FSM Review

FSMs provide a powerful way to systematically design a sequential circuit from written specifications. Use the following procedure to design an FSM:

- (a) Identify inputs and outputs
- (b) Sketch a state transition diagram
- (c) For a Moore machine, write a state transition table and then write an output table
- (d) For a Mealy state machine, write a combined state transition and output table
- (e) Select state encodings—the selection affects the hardware design
- (f) Write Boolean equations for the next state and output logic
- (g) Sketch the circuit schematic

1.5 Timing of Sequential Logic

[Opening paragraph, if any]

1. The Dynamic Discipline

2. System Timing
3. Clock Skew
4. Metastability
5. Synchronizers
6. Derivation of Resolution Time

1.6 Parallelism

1.7 Summary

2 Grey Box Exploration

1. The first blurb is on page ...
2. The second blurb is on page ...

3 Figures

Two figures were selected from this chapter for special recognition. Figure[...] was selected ...
Figure[...] was selected ...

4 Example Problems

See the attached images on the next pages.

5 Glossary

All definitions were found from the Google search engine, typing "define state" for the first item.

1. State
noun:
 - (a) the particular condition that someone or something is in at a specific time.
 - (b) a nation or territory considered as an organized political community under one government.
adjective:
 - (a) of, provided by, or concerned with the civil government of a country.

- (b) used or done on ceremonial occasions; involving the ceremony associated with a head of state.

verb:

- (a) express something definitely or clearly in speech or writing.
- (b) [music] present or introduce (a theme or melody) in a composition.

2. Sequential

adjective:

- (a) forming or following in a logical order or sequence.
 - [computing] performed or used in sequence.

3. Latch

noun:

- (a) a metal bar with a catch and lever used for fastening a door or gate.

verb:

- (a) fasten (a door or gate) with a latch.

4. Register

noun:

- (a) an official list or record, for example of births, marriages, and deaths, of shipping, or of historic places.
- (b) a particular part of the range of a voice or instrument.

verb:

- (a) enter or record on an official list or directory.
- (b) (of an instrument) detect and show (a reading) automatically.

5. Flip-Flop

noun:

- (a) a light sandal, typically of plastic or rubber, with a thong between the big and second toe.
- (b) [North American] a backward handspring.

verb:

- (a) move with a flapping sound or motion.
- (b) [informal · North American] make an abrupt reversal of policy.

6 Interview Question

See the attached image on the next page.

7 Reflection

8 Questions for Lecture

1. QUESTION 1
2. QUESTION 2
3. QUESTION 3

References