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

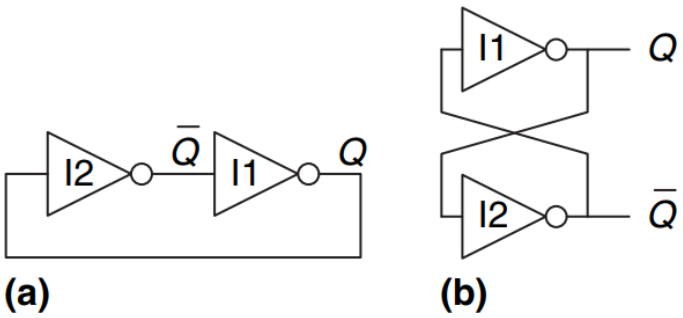


Figure 1: Cross-coupled inverter pair circuits

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 $\bar{Q}$.

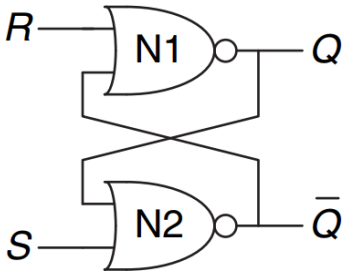


Figure 2: Schematic of an SR latch

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 ($\bar{Q}$ = FALSE). Thus, N1 can be expressed as RNOR FALSE = FALSE NOR FALSE = TRUE, and so Q = TRUE.

(b) $R = 1$, $S = 0$
Similarly to the first case, Q = FALSE and $\bar{Q}$ = TRUE

(c) $R = 1$, $S = 1$
$Q = \bar{Q}$ = 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 $\bar{Q}$ = TRUE as the output. This output matches with N1, since TRUE NOR FALSE = 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

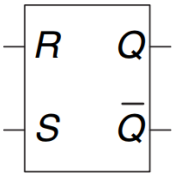


Figure 3: SR latch symbol

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).

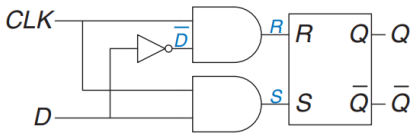


Figure 4: D latch schematic

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 $N1$.

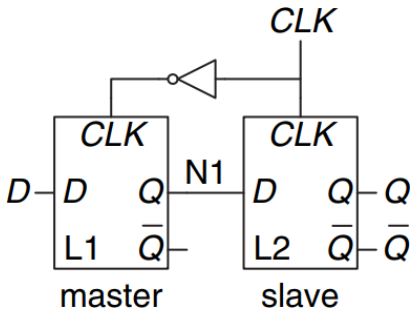


Figure 5: D flip-flop schematic

- 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.

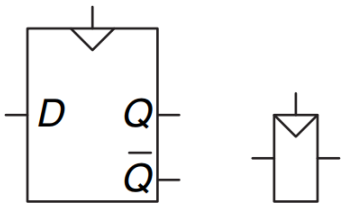


Figure 6: D flip-flop full symbol (left) and minimal symbol (right)

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.

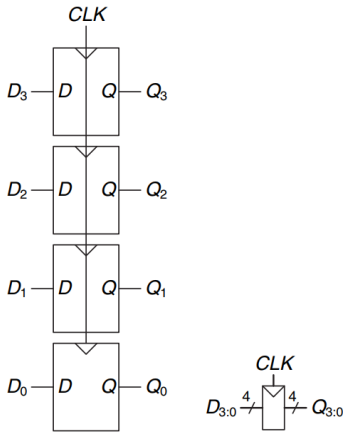


Figure 7: A 4-bit register's schematic (left) and symbol (right)

When the register's clock is on (CLK = 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 = 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 $EN = 1$ = 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 $EN = 0$ = 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.

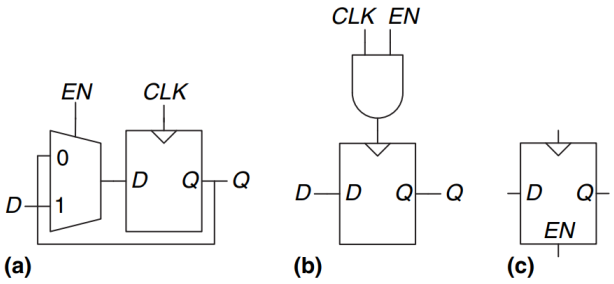


Figure 8: The enabled flip-flop schematics (a and b) and symbol (c)

6. **Resettable Flip-Flop**

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

   - When $RESET = 1$ = TRUE, the resettable flip-flop ignores the input D and resets the output Q to 0.
   - When $RESET = 0$ = 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.

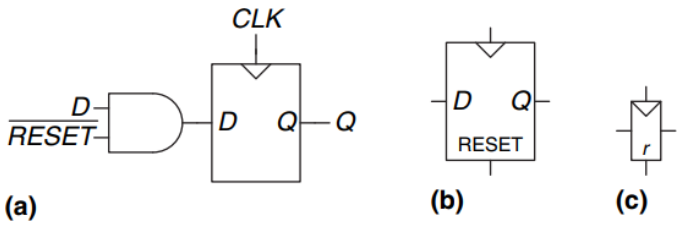


Figure 9: The resettable flip-flop schematics (a and b) and symbol (c)

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 hardward 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**

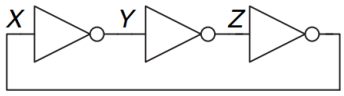


Figure 10: Three-inverter loop

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.

CLK	D	Q_{prev}	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$$Q = CLK \cdot D + \overline{CLK} \cdot Q_{prev}$$

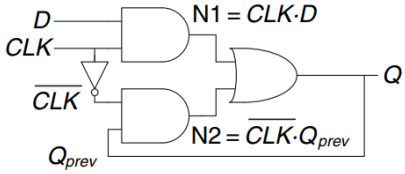


Figure 11: An asynchronous circuit with a race condition

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**

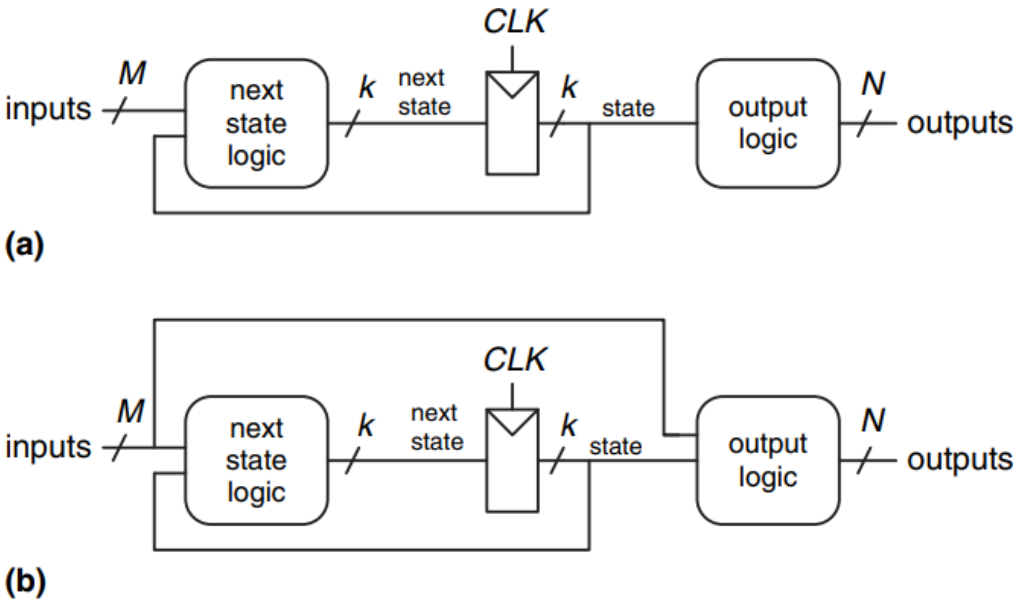


Figure 12: Finite state machines: (a) Moore machine, (b) Mealy machine

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:

(a) Draw a block diagram or a black box of the FSM

(b) Sketch the *state transition diagram* and complete it based on the problem statements

(c) Rewrite the transition diagram as a *state transition table*

(d) 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

(e) Simplify the Booleans equations using Boolean algebra or Karnaugh maps

(f) 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

The aperture of a sequential element is defined by a *setup* time and a *hold* time, before and after the clock edge, respectively. Similar to static discipline which limits the user to using logic levels outside of the forbidden zone, the *dynamic discipline* limits the usage of signals that change outside of the aperture time.

1. **The Dynamic Discipline**

   A synchronous sequential circuit not only has a functional specification but also a timing specification. For the circuit to sample its inputs correctly, the inputs must have stabilized at least some *setup time*, t_{setup}, before the rising edge of the clock and must remain stable for at least some *hold time*, t_{hold}, after the rising edge of the clock.

   $$\text{aperture time} = \text{setup time} + \text{hold time} = t_{\text{setup}} + t_{\text{hold}} \tag{1}$$

   The *dynamic discipline* states that the inputs of a synchronous sequential circuit must be stable during the setup and hold aperture time around the clock edge.

2. **System Timing**

   The *clock period or cycle time*, T_c is the time between rising edges of a repetitive clock signal. Its reciprocal, $f_c = \frac{1}{T_c}$, is the *clock frequency*—measured in units of Hertz (Hz). The higher the frequency is, the more work the system can accomplish.

3. **Clock Skew**

   Clock skew happens when the clock does not reach all registers at exactly the same time. A common reason for this variation is that the wires from the clock source to different registers can be of different lengths, resulting in different delays.

   When doing timing analysis, it is recommended to consider the worst-case scenario to make sure the circuit will time correctly under all circumstances.

4. **Metastability**

   It is not always possible to guarantee that the input to a sequential circuit is stable during the aperture time, especially when the input arrives from the external world.

   When a flip-flop samples an input that is changing during its aperture, the output Q may momentarily take on a voltage between 0 and V_{DD} that is in the forbidden zone. This is called a *metastable* state. Eventually, the flip-flop will resolve the output to a *stable state* of either 0 or 1.

   The *resolution time*, denoted t_{res} is the time required to resolve a stable state during the clock cycle. t_{res} is a random variable.

5. **Synchronizers**

   Asynchronous inputs to digital systems from the real world is inevitable (for example, human input). These asynchronous inputs can lead to metastability in the system if they are not handled correctly. To guarantee good logic levels, all asynchronous inputs should be passed through *synchronizers*—a device that receives an asynchronous input D and a clock CLK and produces an output Q within a bounded amount of time.

1.6 Parallelism

The speed of the system is characterized by the latency and throughout of information moving through it.

A *token* is a group of inputs that are processed to reduce a group of outputs.

The *latency* of a system is the time required for one token to pass through the system from start to end.

The *Throughput* is the number of tokens that can be produced per unit time. The throughput can be improved by processing several tokens at the same time. This is called *parallelism*. There are two forms of parallelism:

- *Spatial parallelism*: Multiple copies of hardware are provided so that multiple tasks can be done at the same time.
- *Temporal parallelism*: Commonly call *pipelining*. A task is broken down into stages, like an assembly line, allowing multiple tasks to spread across the stages.

1.7 Summary

A sequential circuit is a circuit whose output depends on the current value *and* the previous value of the inputs. Due to this nature, sequential circuits have memory which may explicitly remember the previous inputs or may distill the values into states of the system.

Two fundamental bulding blocks of sequential logic are latches and flip-flops. SR latch and D latch are two common types of latches. A D flip-flop consists of two D latches connecting to each other through a node and a clock. 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. There are also many other types of flip-flops, including enabled flip-flop (with an extra ENABLE input) and resettable flip-flop (with an extra RESET input).

Since sequential circuits are not the same as combinational circuits, their outputs cannot be determined just by looking at the current state of the inputs. Synchronous logic design and dynamic discipline allow analyzing and designing sequential logic more easily. Two common types of synchronous sequential circuit are finite state machine and pipeline.

Finite state machines provide a powerful way to systematically design a sequential circuit from written specifications.

Timing is an essential aspect of sequential circuit design. This chapter covers the dynamic discipline and a few issues in the timing of circuits, such as clock skew and metastability.

2 Grey Box Exploration

1. The first blurb is on page 113, which states *Some people call a latch open or closed rather than transparent or opaque. However, we think those terms are ambiguous—does* open *mean transparent like an open door, or opaque, like an open circuit?*.

   This is an interesting question. People interpret words differently, and so a solution to the ambiguity is to have a guideline where terminologies that are easy to confuse are explained in details.

Some people find transparent-opaque make more sense people they relate the latch with a physical door—being transparent means open door, allowing data to flow through and vice versa.

Other people, however, find open-closed make more sense because they relate the latch with the open-closed concept of a circuit—open circuit means no signal can flow from one end to the other and vice versa.

2. The second blurb is on page 121, which states t_{pcq} *stands for the time of propagation from clock to Q, where Q indicates the output of a synchronous sequential circuit.* t_{ccq} *stands for the time of contamination from clock to Q. These are analogous to* t_{pd} *and* t_{cd} *in combinational logic.*

   Recall that t_{pd} in combinational logic is the propagation delay of the system and that t_{cd} is the contamination delay of the system. Similarly, in sequential circuit, t_{pcq} is the propagation delay and t_{ccq} is the contamination delay.

   In combinational circuits, propagation delay is the maximum time from when an input changes until the output(s) reach their final value, and contamination delay is the minimum time from when an input changes until any of the outputs starts to change its value. The propagation and contamination delay in sequential logic design are exactly the same—they just have different notations.

3 Figures

Two figures were selected from this chapter for special recognition. Figure 13 was selected because it concisely shows the key difference between Moore FSM and Mealy FSM. In Moore FSM, the output depends only on the current state of the system. On the other hand, Mealy FSM's output depends not only on the current state of the system but also the raw inputs (notice the wire connecting the inputs with the output logic).

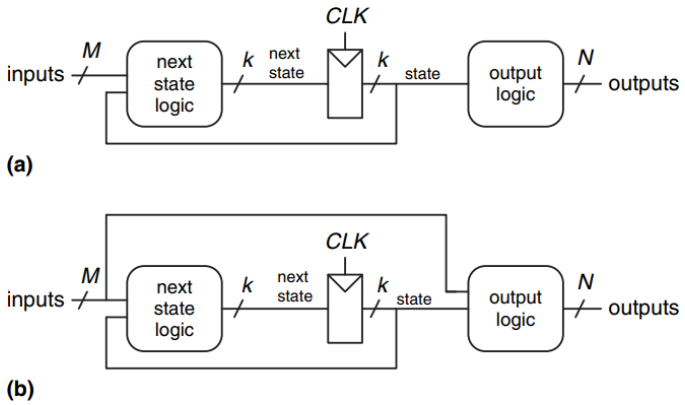


Figure 13: Finite state machines: (a) Moore machine, (b) Mealy machine

Figure 14 was selected because it shows how a register was constructed. An N-bit register is just a series of N flip-flops that share a common CLK input. When the clock raises its edge, all output bits of the register are changed simultaneously to the values of their corresponding input bits. In

other occasions, the output bits retain their previous values. It is quite interesting and surprising to know that registers are constructed from flip-flops.

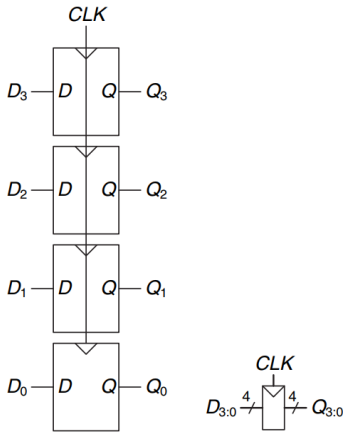


Figure 14: Inside of a 4-bit register

4 Example Problems

See the attached images on the next four pages.

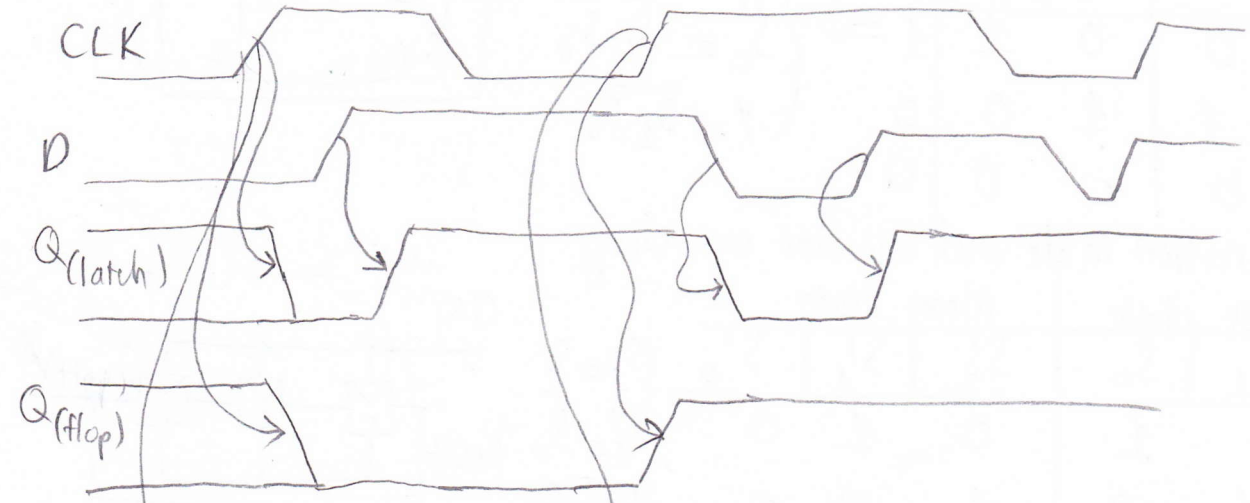
EXAMBLE PROBLEMS

Example 3.1: How many transistors are needed to build the D flip-flop described in this section?

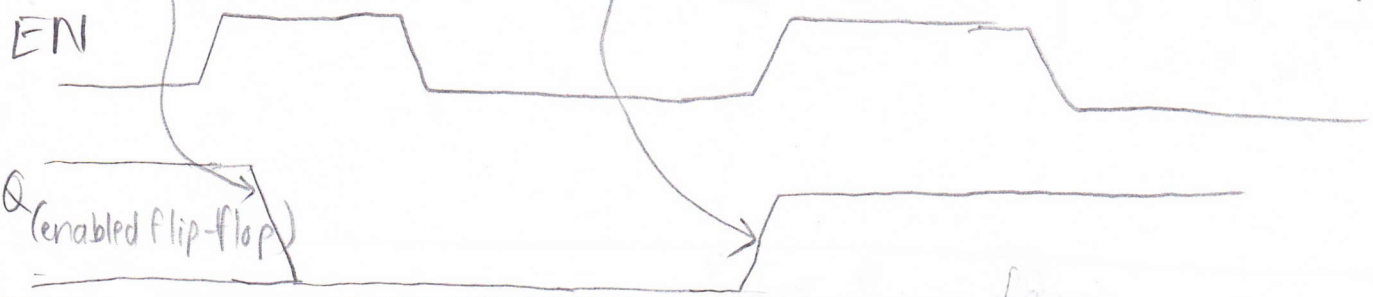
Device	Number of transistors
2-input NAND/NOR	4
NOT	2
2-input AND/OR	6 (NAND/NOR + NOT)
SR latch	8 (2×NOR)
D latch	22 (SR latch + 2× AND + NOT)
D flip-flop	46 (2× D latch + NOT)

Variation: How many transistors are needed to build an enabled flip-flop?
An enabled flip-flop uses a D flip-flop and a 2-input AND gate.
Thus, an enabled flip-flop requires 52 transistors

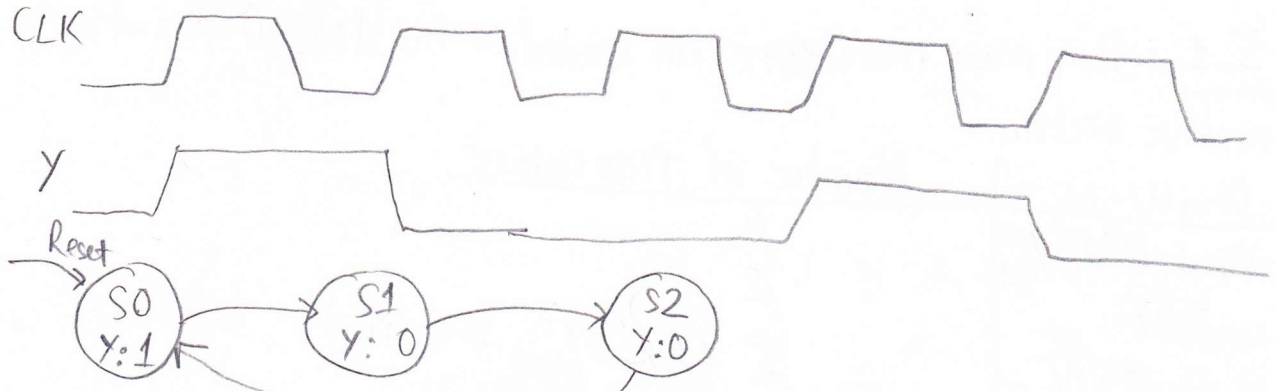
Example 3.2: Determine output Q from the waveforms of CLK and D



Variation: Same graph as above but with an enabled flip-flop



Example 3.6 : FSM state encoding



State transition table

current state	next state
S0	S1
S1	S2
S2	S0

State Encoding

State	$S_{1:0}$
S0	00
S1	01
S2	10

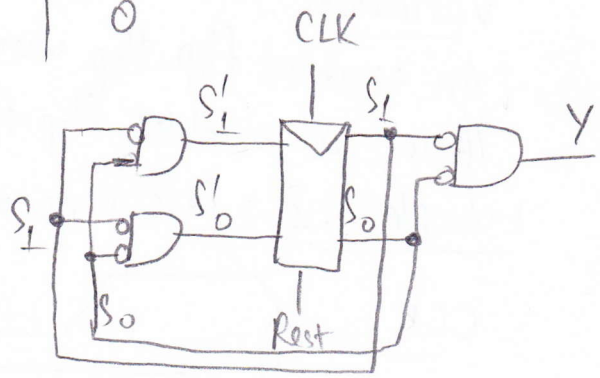
Divide-by-3 output table

current state	output
S0	1
S1	0
S2	0

State transition table with binary encoding

current state S_1	current state S_0	next state S_1'	next state S_0'	Output Y
0	0	0	1	1
0	1	1	0	0
1	0	0	0	0

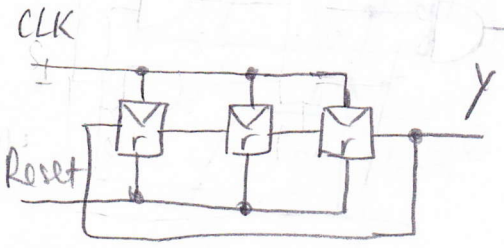
$$\begin{cases} S_1' = \overline{S_1} S_0 \\ S_0' = \overline{S_1}\,\overline{S_0} \\ Y = \overline{S_1}\,\overline{S_0} \end{cases}$$



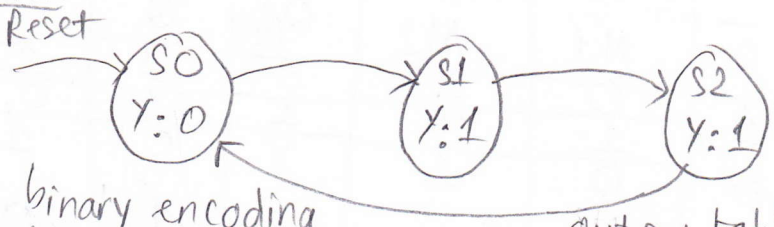
State transition table with one-hot encoding

Current state S_2	Current state S_1	Current state S_0	Next state S_2'	Next state S_1'	Next state S_0'
0	0	1	0	1	0
0	1	0	1	0	0
1	0	0	0	0	1

$$\begin{cases} S_2' = S_1 \\ S_1' = S_0 \\ S_0' = S_2 \\ Y = S_0 \end{cases}$$



Variation:



State binary encoding

State	$S_{1:0}$
S0	00
S1	01
S2	10

output table:

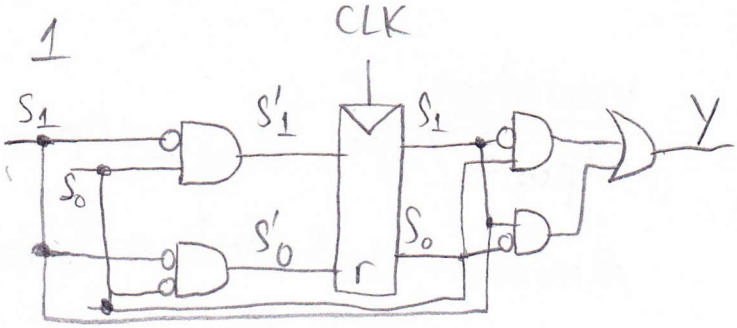
current state	output
S0	0
S1	1
S2	1

State transition table with binary encoding

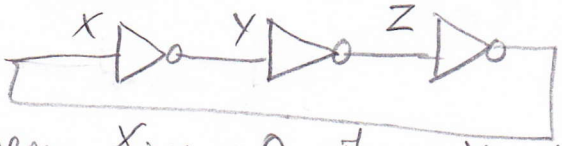
current state		next state		output
S_1	S_0	S'_1	S'_0	Y
0	0	0	1	0
0	1	1	0	1
1	0	0	0	1

$$\Rightarrow \begin{cases} S'_1 = \bar{S}_1 S_0 \\ S'_0 = \bar{S}_1 \bar{S}_0 \\ Y = \bar{S}_1 S_0 + S_1 \bar{S}_0 \end{cases}$$

Sketch:

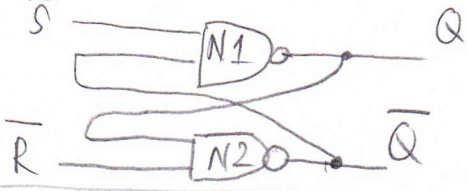


Example 3.3: Astable circuits



Suppose $X_{init} = 0$. Then, $Y = 1$ and $Z = 0$, but then $X = 1$.
Suppose $X_{init} = 1$. Then, $Y = 0$ and $Z = 1$, but then $X = 0$.
The circuits has no stable state. (astable circuit)
X, Y, and Z will oscillate between 0 and 1

Variation:



is this circuit stable or astable?

Truth table:

S	R	$\bar{S}$	$\bar{R}$	N1	N2	Q_{prev}	Q	$\bar{Q}$
0	0	1	1	0	1	0	0	1
				1	0	1	1	0
0	1	1	0	0	1	X	0	1
1	0	0	1	1	0	x	1	0
1	1	0	0	1	1	x	1	1

=> Stable sequential circuit

Example 3.10 : Timing Analysis

$t_{pcq} = 80ps$, $t_{setup} = 50ps$, $t_{pd} = 40ps$, 3 logic gates. $f_{c\,max} = ?$

Minimum clock cycle = $T_{c\,min} = t_{pcq} + 3t_{pd} + t_{setup} = 250\,ps$

Maximum clock frequecy = $f_{c\,max} = \frac{1}{T_{c\,min}} = 4\ GHz$

Variation:

$t_{pcq} = 75ps$, $t_{setup} = 10ps$, $t_{pd} = 100ps$, 7 logic gates. $f_{c\,max} = ?$

Minimum clock cycle = $T_{c\,min} = t_{pcq} + 7t_{pd} + t_{setup} = 785\,ps$

Maximum clock frequency = $f_{c\,max} = \frac{1}{T_{c\,min}} = 1.274\ GHz$

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.

INTERVIEW QUESTION

Question 3.3: What is the difference between a latch and a flip-flop? Under what circumstances is each one preferable?

A latch allows data to flow through when its CLK rises, which updates the output to the current input. When its CLK is low, it blocks the incoming data and thus let the output retain its previous value.

A flip-flop consists of two latches connect to each other via a node. One of the latches connects to CLK, and the other one connects to $\overline{CLK}$. The flip-flop differs from the latch in that it guarantees to update the output after a low-high cycle of the clock. When the CLK rises, the flip-flop allows the incoming data to flow to the intermediate node (the output remains unchanged). When the CLK is low, the flip-flop blocks the incoming data but allows the data to flow from the intermediate node to the output, which effectively change the output to the value of the input when the CLK previously rises.

In short, latch is level-triggered (output changes as soon as the input changes) and flip-flop is edge-triggered (output changes when CLK goes from high to low or from low to high).

It is generally recommended to use flip-flops rather than latches because of the edge-triggered nature of the flip-flops. However, latches can be used for simple circuits (as flip-flops double the number of latches).

7 Reflection

This chapter is more challenging than chapter 2 because sequential circuits have more time aspect attached to it—the previous value of the inputs must be known in order to determine the output. Now that I've learned combinational and sequential logic design, I've become more confident in my ability to build circuits from simple logic gates. Before I began to take this class, I thought only about the functional specifications of a circuit. Nevertheless, I have come to realize that timing is also extremely important, especially in sequential logic.

To me, the Finite State Machine section was the most fun to read. Although I had a little background of embedded systems, being able to systematically design a sequential circuit from written specifications still sounds fascinating to me. Finally, my least favorite section was the Timing of Sequential Logic (even though I learned a lot from the section).

8 Questions for Lecture

1. Do the registers in microcontrollers work the same way as ones mentioned in this chapter? if not, how are they different from each other?

2. Can you provide two examples of a finite state machine, preferrably a Moore machine and a Mealy machine?

3. If SR latch is inconsistent when both S and R are asserted, what purpose could it serve besides making D latches?