EECS-140/141 Introduction to Digital Logic Design
Lecture 7: Sequential Logic Basics

I. OVERVIEW

I.A Combinational vs. Sequential Logic

Combinational Logic (everything so far): Outputs depend entirely on _____ of inputs at any instant of time. If you apply the same input combination at different times, you get:

Sequential Logic: Outputs depend on the _____ in which inputs are applied, not just on the current input combination. So, the *same* input combination at *different* times may produce:

Such circuits "remember" past inputs, so we say they have:

I.B Examples

I.B.1 Alarm System (Assignment 1)

This system has a major shortcoming! If a burglar opens a window, the alarm will sound (good), but if s/he then crawls through the window and then shuts it:

Better to have alarm sound and *keep sounding* until someone intervenes (homeowner, police), even if the "tripping" conditions revert to normal. This is *memory*.

I.B.2 Sequence Locks on Eaton Lab Doors

These require user to enter code on keypad in the proper *sequence* (e.g. 8512). This requires *sequential* logic.

II. SET-RESET (SR) LATCHES

II.A Introduction

Key to creating memory in logic devices is *feedback*: output signals must be connected ("fed back") to inputs.

Simplest sequential logic circuit is a *latch*: output is held ("latched") by the circuit.

II.B Basic (Asynchronous) SR Latch

II.B.1 Circuit

Can make a basic SR latch with NOR gates:

II.B.2 Characteristic Table (Similar to T.T.)

S	R	Q_a	Q_b
0	0		
0	1		
1	0		
1	1		

II.B.3 Timing Diagram

Figure 7.5 illustrates event *sequences*.

Note: from $S = R = 1$ to $S = R = 0$, *one* of Q_a or Q_b goes high, but:

Circuit might also oscillate (outputs change rapidly).

Note: Q_a and Q_b can change at *any* time in response to S and R. When output changes can happen at any time, operation is said to be:

Alarm System: Output of combinational logic circuit can be connected to S input, $Q_a = 1$ sounds alarm, and a manual reset signal would be connected to the R input.

II.C Gated (Clocked) SR Latch

II.C.1 Clock Signal

It is often useful to control *when* outputs can change value. Many digital systems allow these *state changes* at regular intervals controlled by a *Clock* signal.

The % of one period that *Clk* = 1 is the *duty cycle* of the clock signal (here, duty cycle <50%).

II.C.2 NOR Version

We can modify the basic SR latch by ANDing each input (*S* and *R*) with the *Clk* signal.

Note:

Note: Must *avoid* $S = R = 1$, since when *Clk* goes to 0, $S' = R' = 0$ and *Q* output uncertain.

Clk	*S*	*R*	Q_{new}
0	X	X	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Note: While *Clk* = 1 (high *level*), *Q* (and $\bar{Q}$) can change if *S* or *R* change. Any device whose output can change whenever *Clk* = 1 is called:

Could also call it "semi-synchronous".

II.C.3 NAND Version

You will verify equivalence in HW.

III. D (DATA) LATCHES AND FLIP-FLOPS

III.A Introduction

SR latches have 2 separate inputs (*S* and *R*), which force output *Q* to 1 and 0, respectively. Useful in various applications, such as control circuits.

But sometimes we need something slightly different: a way to hold or *store* a binary *data* value until it is needed at some later time.

III.B Gated D Latch

This is a simple modification of a gated SR latch with *single* input *D*:

Clk	*D*	Q_{new}
0	X	
1	0	
1	1	

*Note*1: While *Clk* = 0, *Q stores* value of *D* input at instant *Clk* goes from 1 to 0 (negative clock edge).

*Note*2: In gated D latch, *impossible* to have $S = R = 1$.

III.C D Flip-Flops

III.C.1 Latch vs. Flip-Flop

Have seen that outputs of SR and D latches can change any time *Clk* = 1 (level-sensitive, or semi-synchronous), or at *any* time (asynchronous) if not gated.

For precise circuits, we need to restrict the output (state) changes to *specific* instants when *Clk* transitions from 0 to 1 or from 1 to 0. Such devices are called *edge-triggered*, and simple 1-bit edge-triggered devices are *flip-flops* (FFs).

Circuits whose state (outputs) can change only *once* per clock cycle are called:

III.C.2 Master-Slave D FF

One way (not the best way) to achieve edge-triggering is to *cascade* 2 gated D latches as follows:

Note: While *Clk* = 1, Q_m can change, but Q_s cannot. When *Clk* transitions 1 to 0, Q_m latches the value of *D at that instant* and then holds while *Clk* = 0. Q_s follows Q_m while *Clk* = 0, but Q_m can't change then.

When *Clk* goes 0 to 1, Q_s latches value of Q_m while *Clk* = 1

Result: $Q_s = Q$ can change *only* when *Clk* transitions from 1 to 0, and takes the value of *D* at that instant. The 1 to 0 transition instant is known as:

Hence, this is a *negative edge-triggered D FF.*

Timing Diagram

III.C.3 Edge-Triggered D FF

Master-Slave D FF requires 8 NAND gates (4 for each D latch). We can get the *same* functionality with only 6 NAND gates. Fig 7.11 is the *positive*-edge-triggered version.

III.C.4 Clear and Preset Capability

D FFs are often used in larger circuits where you need to be able to clear all the FFs (set $Q = 0$) with one signal or to preset them all (set $Q = 1$ with another signal.

III.C.4.a Synchronous Clear and Preset

In this case, clear or preset take effect at the *next* triggering clock edge:

III.C.4.b Asynchronous Clear and Preset

In this case, clear or preset take effect *immediately*, regardless of *Clk*. This can be incorporated into FF circuits (see Fig. 7.13 and Fig. 7.14).

IV. OTHER FFs

IV.A Toggle (T) FF

This is a single-input FF that "toggles" (inverts) the output if $T = 1$ and leaves it unchanged if $T = 0$.

Can synthesize with D FF by feeding back Q and $\bar{Q}$ outputs, according to:

IV.B JK FF

The JK FF combines SR function with T function.

J	K	Q_{new}
0	0	
0	1	
1	0	
1	1	

V. REGISTERS

These are sets of *n* FFs used to store/manipulate *n* bits of info.

V.A Basic Shift Register

Can make a shifting circuit with muxes (previous HW), but they have no storage capability. Shift register incorporates *both* shifting and storage.

Example: "Move over" sign

Lighted arrow moves from (3) to (2) to (1) to (0), back to (3), etc. Can implement with 4-bit shift register that *starts* with values 1 0 0 0. Each *Q* bit controls one arrow (1=on, 0=off).

V.B Serial vs. Parallel Data Transfer

Can transfer an *n*-bit data value from one place to another in 2 different ways.

Parallel: use *n* wires, 1 per bit.

Transfer all *n* bits in one clodk time.

Serial: use 1 wire, transfer bit values in successive clock times. Requires *n* clock times to transfer all bits.

Example: USB stands for Universal *Serial* Bus -- a specific standard for *serial* data transfer.

V.C Parallel-Load Shift Register

$\overline{Shift}/Load$ input: Load when 1, Shift when 0.

See Fig 7.19: Note: AND/OR set is 2:1 mux.

Timing Diagram for serial data transfer (Handout).

VI. Counters

VI.A Overview

Counter output value increments (up-counter) or decrements (down-counter) by 1 with each clock "tic".

Note the pattern in binary counting (3 bit word here).

Count	a_2	a_1	a_0
0			
1			
2			
3			
4			
5			
6			
7			
0			

Now look at waveforms (timing diagram):

VI.B Asynchronous (Ripple) Counters

VI.B.1 Asynch Up Counter

Note that a T FF with $T = 1$ (always) and *Clk* inputs produces a_0!

Same for a_1 if $\bar{a}_0$ is clock input to 2^{nd} T FF with $T = 1$.

Same for a_2 if $\bar{a}_1$ is clock input to 3^{rd} T FF with $T = 1$.

Seems to work OK *except* for timing details at trigger instants.

Note that *only* Stage 0 changes state (output) *exactly* at positive edge of *Clk*. Stage 1 changes after delay of Stage 0 and Stage 2 after delays of both Stage 0 and Stage 1. For example, expand time axis around instant t_4.

Similar to ripple effect in Ripple-Carry adder, so called *ripple counter.* Also, *only* Stage 0 is synched to *Clk*, so also called *asynchronous counter.*

This lack of synchronism can cause problems. Will fix later.

VI.B.2 Asynch Down Counter

Verify for yourself that an asynch (ripple) down-counter will have a_0 (instead of $\bar{a}_0$) as Stage 1 clock input, a_1 (instead of $\bar{a}_1$) as Stage 2 clock input. *Same* ripple behavior.

VI.C Synchronous Counters

VI.C.1 Synch Up Counter

Would like all stages to be *directly* triggered by *same Clk* signal so that all FFs change state (output) at the *same* time: synchronous.

Look more closely at counting sequence. Specifically, what is condition for a_1 to *change* (toggle)?

So, T_i (toggle) input for Stage *i* can be:

$T_0 = 1$
$T_1 = a_0$
$T_2 = a_0 a_1$
etc.
$T_n = a_0 a_1 \cdots a_{n-1}$

Let's also add *Enable*(E) and *Clear*($\overline{Clr}$) controls, where $E = 0$ indicates stop (pause) counting and $E = 1$ indicates start/continue counting.

Then:

Use T FFs with asynch clear capability (active low) -- resets count to 0.

VI.C.2 Parallel Load

Often useful to *start* counting with some particular value. Use parallel load feature.

See Section 7.9.3 and Fig. 7.25.

VI.C.3 Counting to an Arbitrary Number

So far, always count mod-2^n. What about mod-k counting, where k is arbitrary (e.g. $k = 10$)?

See Section 7.10 and homework problem.