University of British Columbia
Electrical and Computer Engineering
Digital Design and Microcomputers CPEN312

Lecture 9: Flip-Flops and Registers.

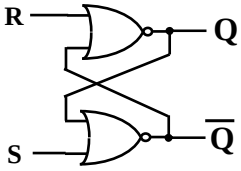
Dr. Jesús Calviño-Fraga. P.Eng.
Department of Electrical and Computer Engineering, UBC
Office: KAIS 3024
E-mail: jesusc@ece.ubc.ca
Phone: (604)-827-5387

February 2, 2017

Objectives

- Latches
- Flip-Flops
- Registers
- Asynchronous/Ripple Counters
- Cascade Ripple Counters

SR Latch With NOR Gates

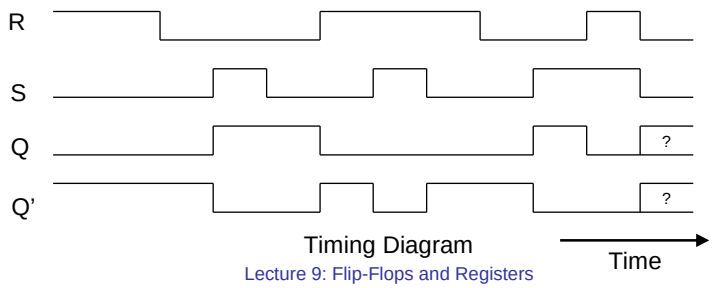


Circuit

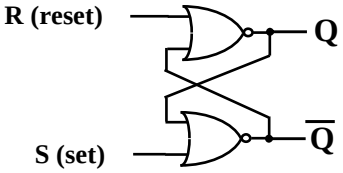
S	R	Q	Q'
0	0	0/1	0/1
0	1	0	1
1	0	1	0
1	1	0	0

(no change)

Characteristic Table



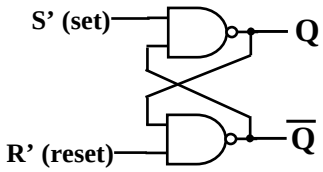
SR Latch With NOR Gates



Time

R	S	Q	$\overline{Q}$	Comment
0	0	?	?	Stored state unknown
0	1	1	0	"Set" Q to 1
0	0	1	0	Now Q "remembers" 1
1	0	0	1	"Reset" Q to 0
0	0	0	1	Now Q "remembers" 0
1	1	0	0	Both go low (forbidden)
0	0	?	?	Random!

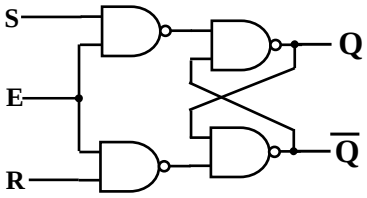
SR Latch With NAND Gates



Time

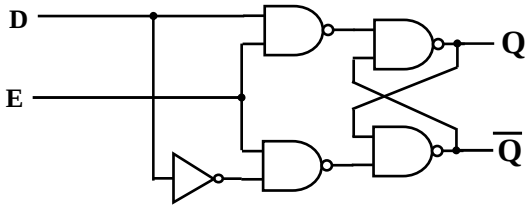
R	S	Q	$\overline{Q}$	Comment
1	1	?	?	Stored state unknown
1	0	1	0	"Set" Q to 1
1	1	1	0	Now Q "remembers" 1
0	1	0	1	"Reset" Q to 0
1	1	0	1	Now Q "remembers" 0
0	0	1	1	Both go high
1	1	?	?	Random!

Gated SR Latch



E	S	R	Q	Q'
1	1	0	1	0
1	0	1	0	1
1	0	0	Q(t-1)	Q'(t-1)
1	1	1	?	?
0	x	x	Q(t-1)	Q'(t-1)

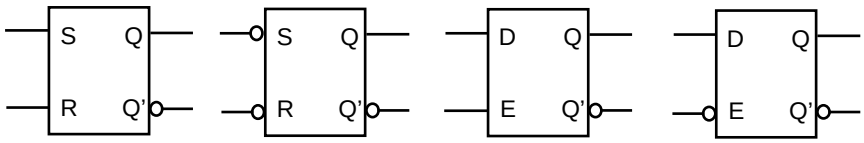
Gated D Latch



E	S	R	Q	Q’
1	1	0	1	0
1	0	1	0	1
0	x	x	Q(t-1)	Q’(t-1)

The humble D Latch is the basic unit of memory! It can store just one bit.

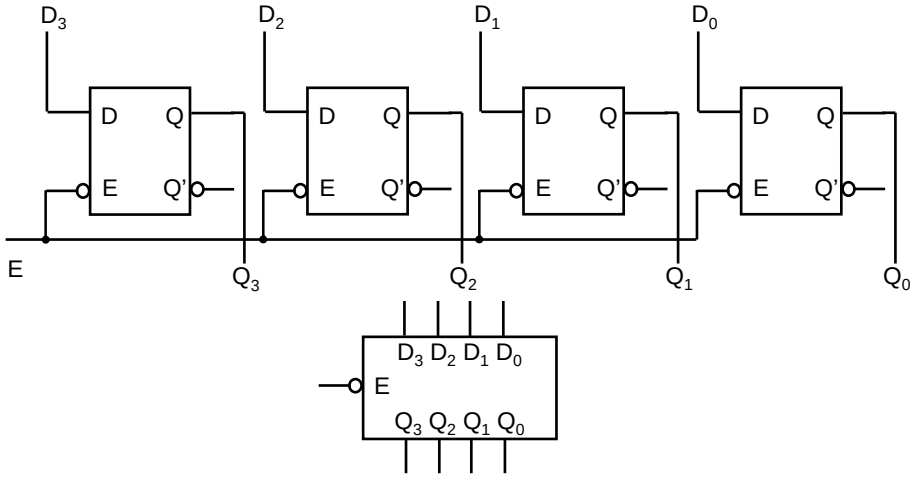
Latch Symbols



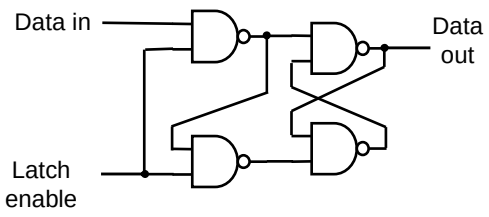
SR Latch | S’R’ Latch | D Latch with Active High Enable | D Latch with Active Low Enable

Multi-bit D latch

- Used to store n-bits.

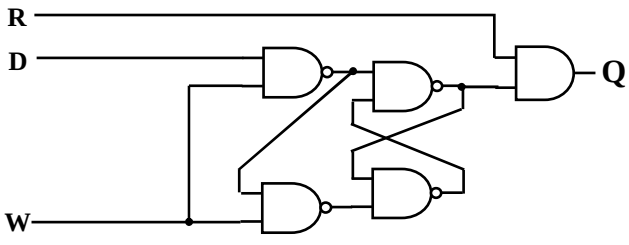


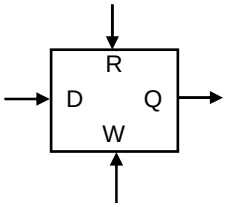
Improved D Latch



I mentioned before that the D latch is a unit of memory. Since we want to have as much memory as possible, we optimize its design as much as possible. By implementing the D latch as in the figure above we save one gate.

Gated D Latch with Output enable

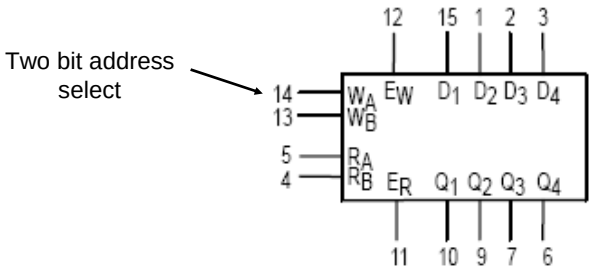




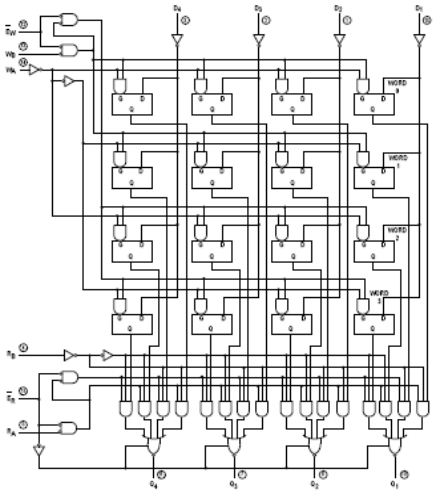
With this simple D Latch we can also build a register. If we pile-up a bunch of registers we get a 'register file'. It is quite difficult to draw, so I'll show you the diagram of a commercially available register file…

74LS670 register file

- Provides one input and one output port only:



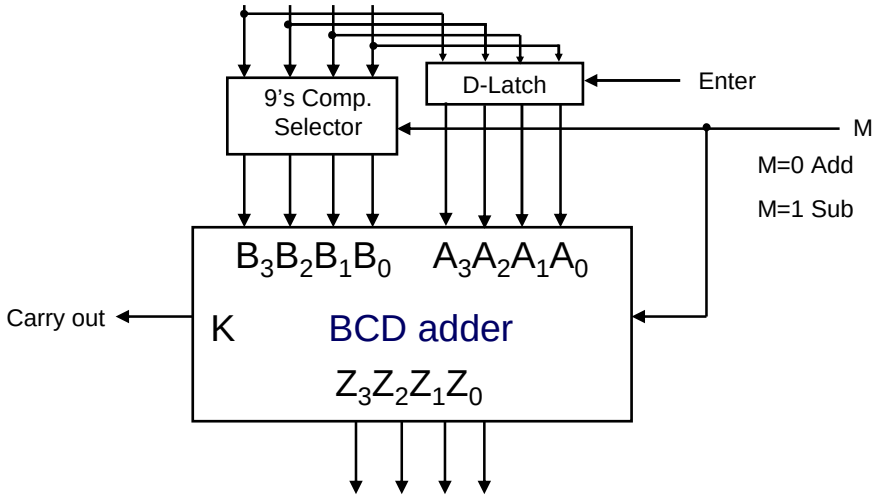
74LS670 register file



Example 1

- BCD Adder/Subtractor with latched input:
  - We can connect the same switches to both A and B.
  - Uses one 4-bit latch.
  - We need an “Enter” key to provide the value in port A throughout the latch.
  - Lab #2 can be done now with 4 BCD digits!

BCD Adder/Subtractor

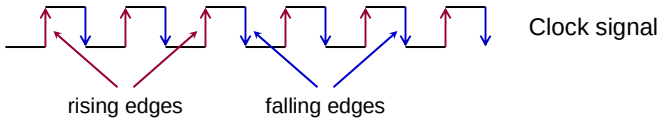


Limitations of Latch Circuits

- When the enable signal is active, the excitation inputs are gated directly to the output Q. Therefore, any change in the inputs causes an immediate change in the latch output.
- We can solve the problem above by using a special timing control signal: the *clock.* It restricts the times at which the states of the latch may change.
- The addition of the clock signal leads to the edge-triggered latch, also called *flip-flop*.

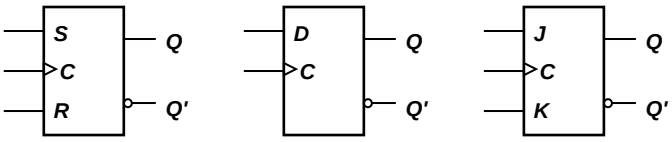
Flip-flops

- In Flip-flops, the outputs change state at a specified point on an input called the *clock*.
- They can change state either at the *positive edge* (rising edge) or at the *negative edge* (*falling edge*) of the clock signal.

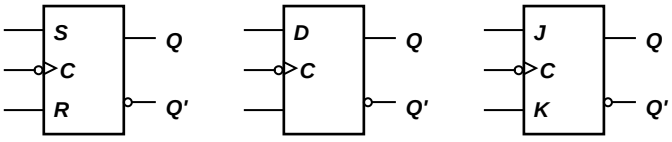


Flip-flops

- Symbols for the SR, D and JK edge-triggered flip-flops. Note the “>” symbol at the clock input.

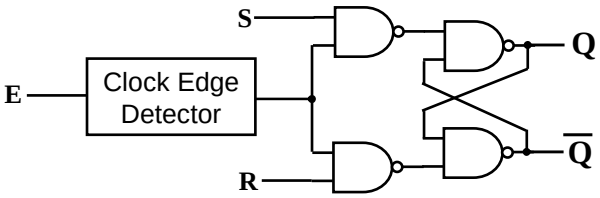


Positive edge-triggered flip-flops



Negative edge-triggered flip-flops

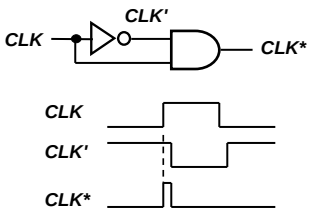
SR Flip-Flop



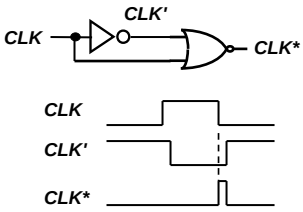
CLK	S	R	Q	Q'	Comment
?	0	0	Q(t-1)	Q'(t-1)	No Change
↑	0	1	0	1	Reset
↑	1	0	1	0	Set
↑	1	1	?	?	Invalid

? = irrelevant ("don't care")
↑ = clock transition LOW to HIGH

Clock Edge Detectors



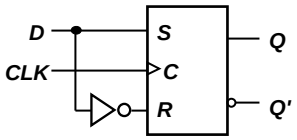
Positive-going transition
(rising edge)



Negative-going transition
(falling edge)

In both detectors above we use the delay introduced by the NOT gate to generate a very narrow output pulse!

D Flip-flop



A positive edge-triggered D flip-flop formed with an S-R flip-flop.

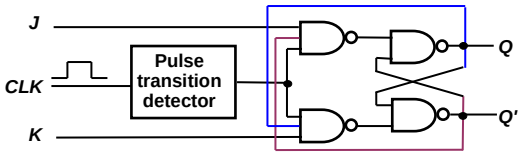
D	CLK	Q(T+1)	Comment
1	↑	1	Set
0	↑	0	Reset

↑ = clock transition LOW to HIGH

JK Flip-Flop

- Q and Q' are fed back to the pulse-steering NAND gates.
- No invalid state.
- The JK Flip-Flop includes a *toggle* state.
  - *J*=HIGH (and *K*=LOW) ⇨ SET state
  - *K*=HIGH (and *J*=LOW) ⇨ RESET state
  - both inputs LOW ⇨ no change
  - both inputs HIGH ⇨ toggle

JK Flip-Flop

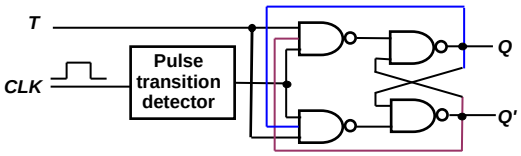


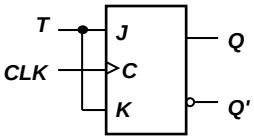
J	K	CLK	Q(t+1)	Comments
0	0	↑	Q(t)	No change
0	1	↑	0	Reset
1	0	↑	1	Set
1	1	↑	Q(t)'	Toggle

Q	J	K	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

$Q(t+1) = J.Q' + K'.Q$

T Flip-Flop

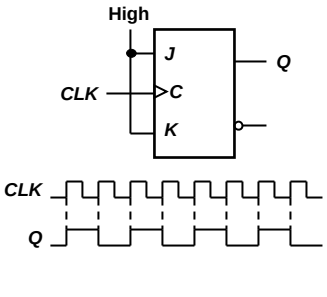




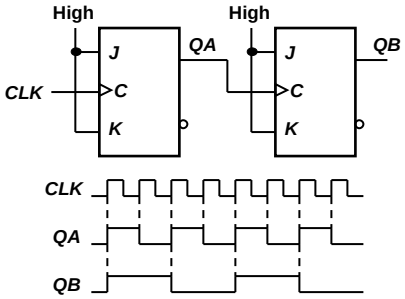
T	CLK	Q(t+1)	Comments
0	↑	Q(t)	No change
1	↑	Q(t)'	Toggle

Q	T	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

Example: Frequency Division



Divide clock frequency by 2.



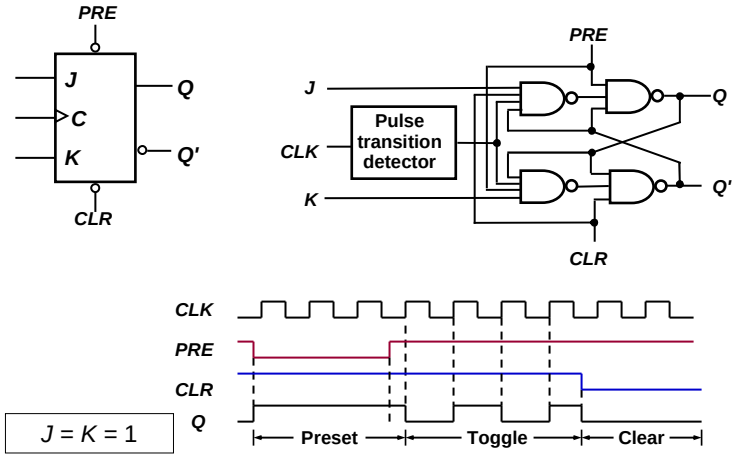
Divide clock frequency by 4.

This is also a counter!

Asynchronous Inputs: Preset & Clear

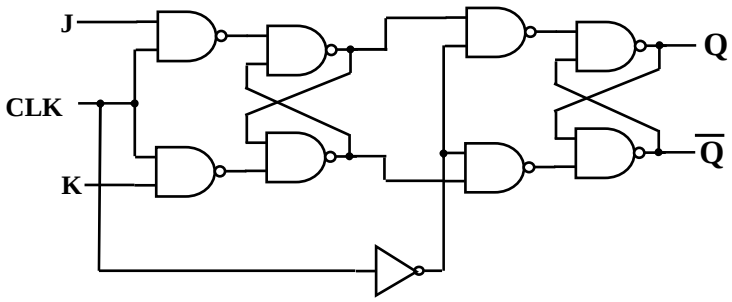
- SR, D and JK inputs are synchronous inputs, as data on these inputs are transferred to the flip-flop's output only on the sensitive edge of the clock pulse.
- Asynchronous inputs affect the state of the flip-flop independently of the clock; example: *preset* (*PRE*) and *clear* (*CLR*). Assuming active high preset and clear inputs:
  - When *PRE*=HIGH, *Q* is immediately set to HIGH.
  - When *CLR*=HIGH, *Q* is immediately cleared to LOW.
- The flip-flop is in normal operation mode when both *PRE* and *CLR* inactive.

Asynchronous Inputs: Preset & Clear



Master-Slave JK Flip-Flop

- Another way of implementing a Flip-Flop is by using a Master-Slave configuration. For example, this is the JK flip-flop:



Counters

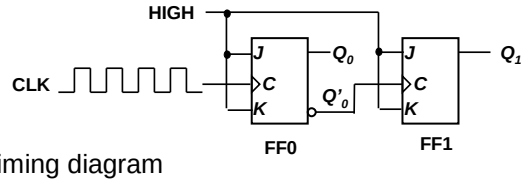
- Counters are circuits that cycle through a specified number of states.
- Two types of counters:
  - synchronous (parallel) counters
  - asynchronous (ripple) counters
- Ripple counters allow some flip-flop outputs to be used as a source of clock for other flip-flops.
- Synchronous counters apply the same clock to all flip-flops.

Ripple Counters

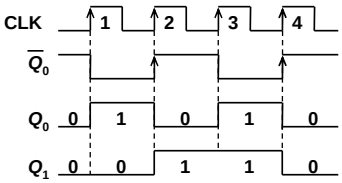
- Ripple counters: the flip-flops do not change their outputs at exactly the same time because they do not have a common clock source.
- Also known as asynchronous counters, as the input clock pulse propagates asynchronously through the counter. As a consequence accumulative delay is present and an important drawback.
- For n flip-flops $\rightarrow$ a MOD (modulus) 2^n counter. (For example for n=3, the counter has up to 8 possible 'count' values).
- Output of the last flip-flop (MSB) divides the input clock frequency by the MOD number of the counter, for this reason a counter is used also as a *frequency divider*.

Binary Ripple Counter

- Output of one flip-flop is connected to the clock input of the next flip-flop. If we use two F-Fs:



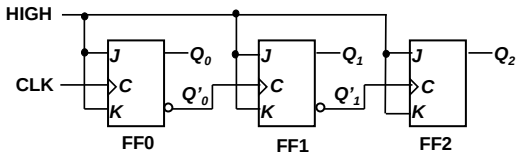
Timing diagram

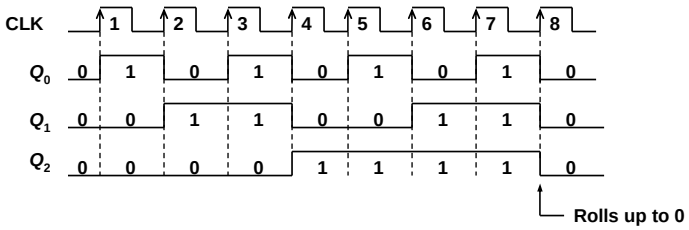


$00 \rightarrow 01 \rightarrow 10 \rightarrow 11 \rightarrow 00$ ...

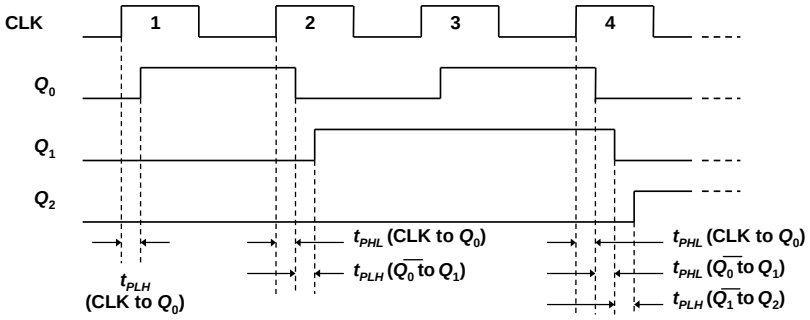
Binary Ripple Counter

- If we use three F-Fs:





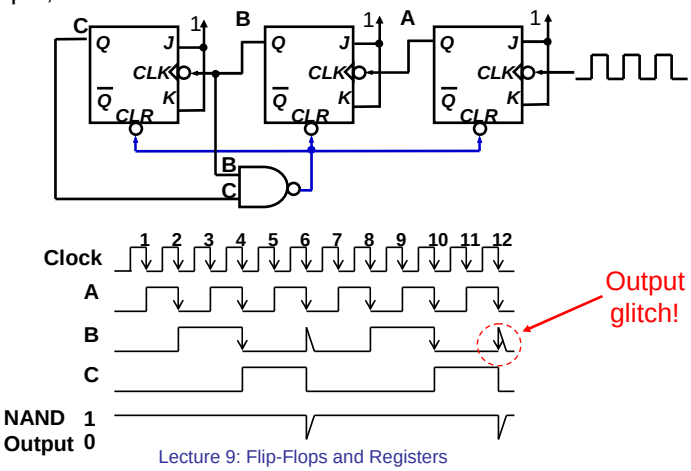
Propagation Delay in Ripple Counters



- If the aggregated delay is greater than the clock pulse, the output of the counter will be incorrect!

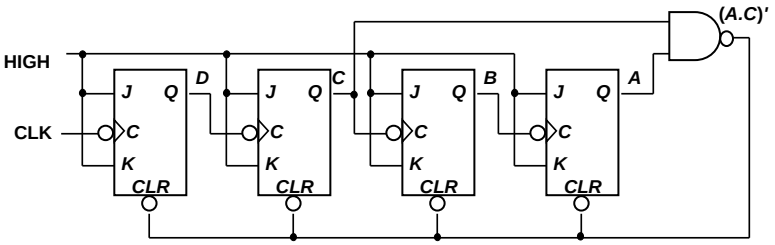
Ripple Counters With Less than 2^n States

- The idea is to reset the counter when it reaches a particular count. For example, consider this circuit:

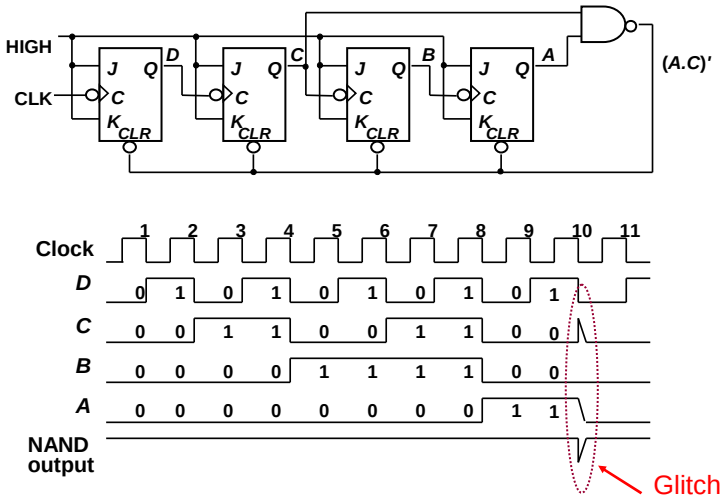


BCD Ripple Counter

- BCD Counters are counters with 10 states. In a BCD ripple counter we need to detect $(1010)_2$ in order to reset the counter

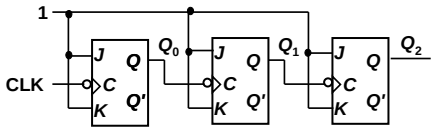


BCD Ripple Counter

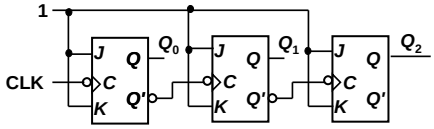


Ripple Down Counters

- A ripple counter can be made to count down by changing the source for the next FF from Q to Q':



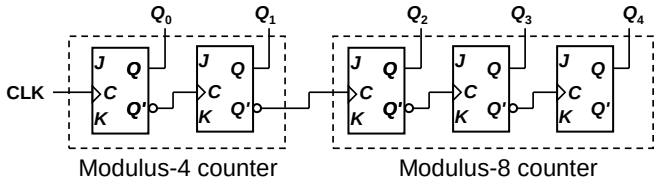
3-bit binary up counter



3-bit binary down counter

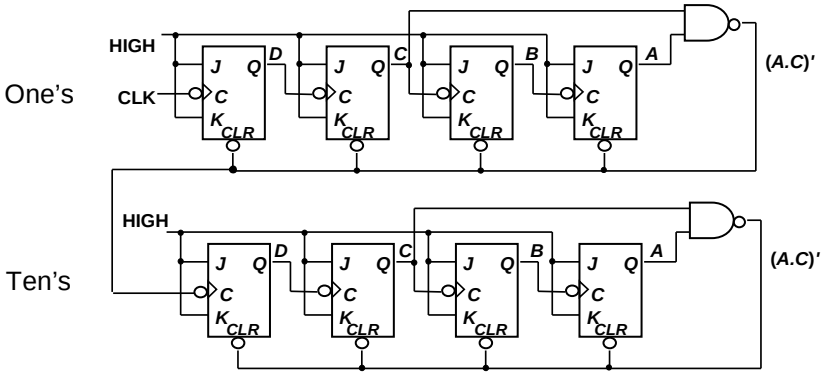
Cascading Ripple Counters

- It possible to implement large ripple counters by cascading smaller ripple counters.
- Example: A 5-bit ripple counter constructed from a 2-bit counter and a 3-bit counter:



Cascading Ripple Counters

- A two digit BCD counter:



Example: Ripple Counters in VHDL.

- For this example I'll use the 50 MHz oscillator in the DE0-CV board to make a BCD seconds counter (0 to 59) and display the current count using the 7-segment displays.
- The 50 MHz clock signal is attached to pin_N2.

VHDL BCD Counter

```
LIBRARY ieee;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_ARITH.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY BCDCOUNT IS
        port(
        KEY0, CLK_50     :IN      STD_LOGIC;
        MSD, LSD :OUT STD_LOGIC_VECTOR (0 to 6)
        );
END BCDCOUNT;

ARCHITECTURE a of BCDCOUNT is
        SIGNAL ClkFlag:  STD_LOGIC;
        SIGNAL Internal_Count:    STD_LOGIC_VECTOR(28 downto 0);
        SIGNAL HighDigit, LowDigit: STD_LOGIC_VECTOR(3 downto 0);
        SIGNAL MSD_7SEG, LSD_7SEG: STD_LOGIC_VECTOR(0 to 6);
BEGIN

        LSD<=LSD_7SEG;
        MSD<=MSD_7SEG;
```


Divide the 50MHz to get 0.5Hz

```
PROCESS(CLK_50)
BEGIN
        if(CLK_50'event and CLK_50='1') then
                if Internal_Count<25000000 then
                        Internal_Count<=Internal_Count+1;
                else
                        Internal_Count<=(others => '0');
                        ClkFlag<=not ClkFlag;
                end if;
        end if;
END PROCESS;
```

Two BCD Counters. The output on one is the clock of the other…

```
PROCESS(ClkFlag, KEY0)
BEGIN
        if(KEY0='0') then -- reset
                LowDigit<="0000";
                HighDigit<="0000";
        elsif(ClkFlag'event and ClkFlag='1') then
                if (LowDigit=9) then
                        LowDigit<="0000";
                        if (HighDigit=5) then
                                HighDigit<="0000";
                        else HighDigit<=HighDigit+'1';
                        end if;
                else
                        LowDigit<=LowDigit+'1';
                end if;
        end if;
END PROCESS;
```

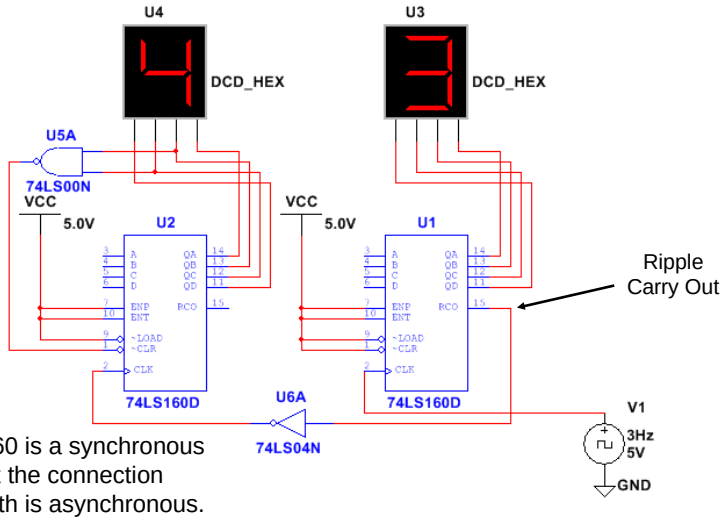

BCD to 7-segment

```
PROCESS(LowDigit, HighDigit)
BEGIN
        case LowDigit is
                when "0000" => LSD_7SEG <= "0000001";
                when "0001" => LSD_7SEG <= "1001111";
                when "0010" => LSD_7SEG <= "0010010";
                when "0011" => LSD_7SEG <= "0000110";
                when "0100" => LSD_7SEG <= "1001100";
                when "0101" => LSD_7SEG <= "0100100";
                when "0110" => LSD_7SEG <= "0100000";
                when "0111" => LSD_7SEG <= "0001111";
                when "1000" => LSD_7SEG <= "0000000";
                when "1001" => LSD_7SEG <= "0000100";
                when others => LSD_7SEG <= "1111111";
        end case;
```

BCD to 7-segment

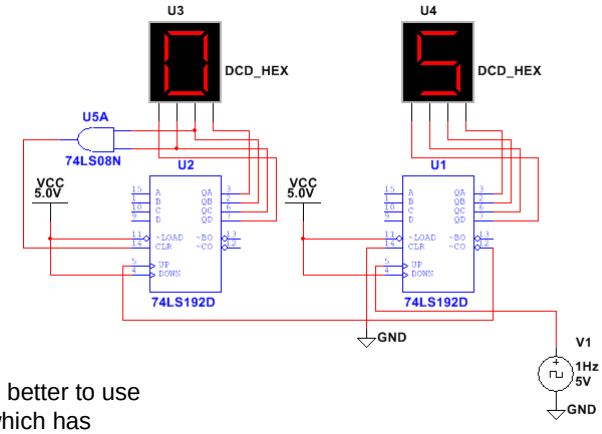
```
            case HighDigit is
                    when "0000" => MSD_7SEG <= "0000001";
                    when "0001" => MSD_7SEG <= "1001111";
                    when "0010" => MSD_7SEG <= "0010010";
                    when "0011" => MSD_7SEG <= "0000110";
                    when "0100" => MSD_7SEG <= "1001100";
                    when "0101" => MSD_7SEG <= "0100100";
                    when "0110" => MSD_7SEG <= "0100000";
                    when "0111" => MSD_7SEG <= "0001111";
                    when "1000" => MSD_7SEG <= "0000000";
                    when "1001" => MSD_7SEG <= "0000100";
                    when others => MSD_7SEG <= "1111111";
            end case;
        END PROCESS;

end a;
```


BCD Ripple Connection Using Multisim



The 74LS160 is a synchronous counter, but the connection between both is asynchronous.

BCD Ripple Connection Using Multisim



For the lab is better to use a 74LS192 which has asynchronous load and reset inputs.

Exercises

- Design a BCD ripple counter using Flip-Flops that follows the sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1, 2, 3, etc. (You can use it for lab 3)
- Design a 2-digit BCD down counter.