VHDL Registers and Counters

Announcements

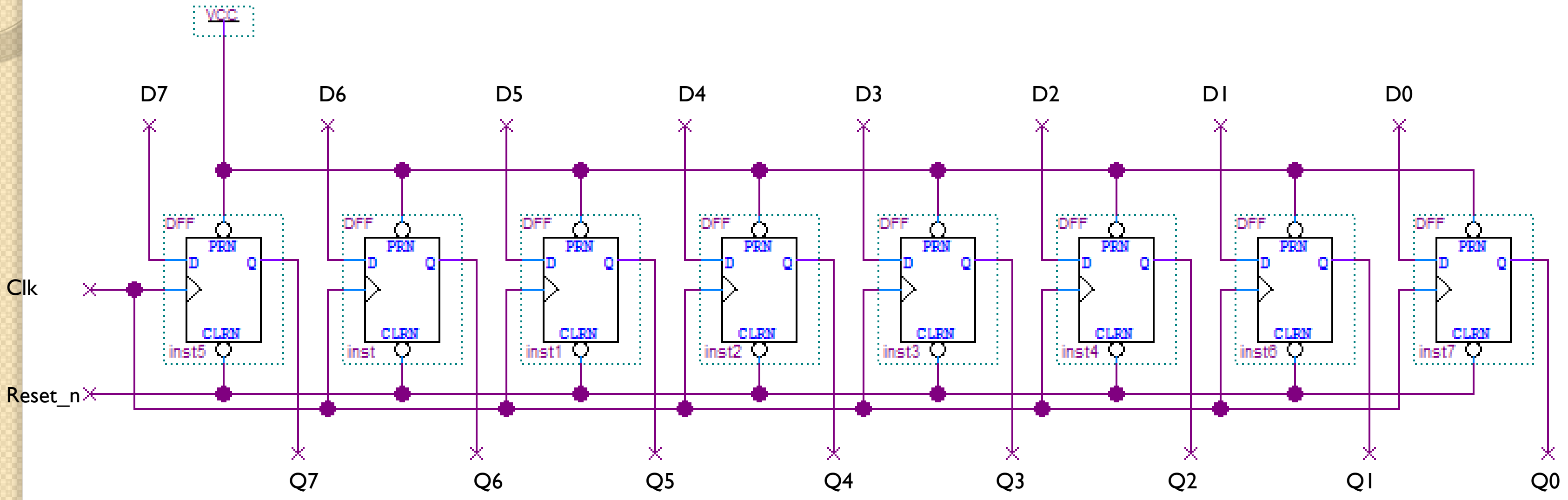
- Homework #8 posted – due Wednesday
- Quiz Wednesday on homework
- Reading assignment:
    - Chapter 6 – especially the examples

Multibit Registers

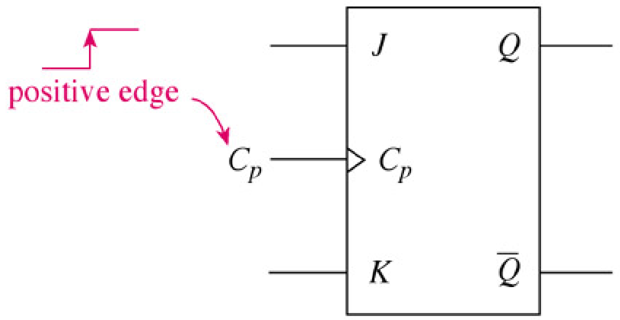
- A D flip-flop is a 1-bit register
- Two or more D flip-flops with their clocks inputs connected together form a multibit register (aka a register)
- The width of the register is determined by the number of memory elements it contains
- Widths of 8, 16, 32 and 64 are common in computing systems

Register VHDL

- Write the entity and architecture for an 8-bit register with asynchronous reset

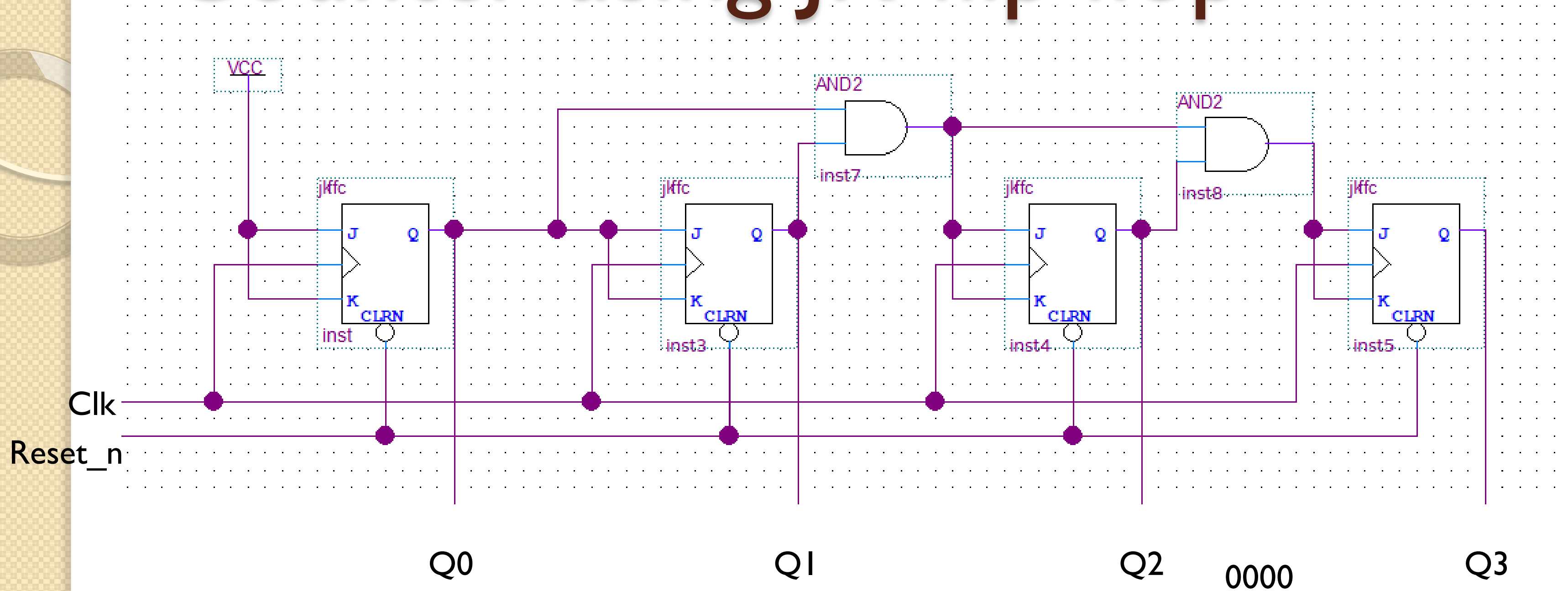


JK Flip-Flop



J	K	Clk	Q	QN	
0	0	↑	Q	QN	Stay the same
0	1	↑	0	1	Clear
1	0	↑	1	0	Set
1	1	↑	$\bar{Q}$	$\overline{QN}$	toggle

Counter using JK-flip flop



- Q0 (Least significant bit) will toggle on every clock cycle because its J and K inputs are tied to VCC
- Subsequent bits will only toggle when **all** previous Qs are 1. This is the purpose of the AND gates

0000

0001

0010

0011

0100

0101

0110

0111

1000

:

Counters

- Simple

```
ENTITY Count4 IS
    PORT(clk, reset_n  : IN STD_LOGIC;
              count          : OUT STD_LOGIC_VECTOR(3 downto 0));
END Count 4;

ARCHITECTURE model of Count4 IS
    SIGNAL int_count : STD_LOGIC_VECTOR(3 downto 0);
BEGIN
    Simple : PROCESS(clk, reset_n)
            BEGIN
                        IF (reset_n = ‘0’) THEN
                                    int_count <= “0000”;
                        ELSIF (rising_edge(clk)) THEN
                                    int_count <= int_count +1;
                        END IF;
            END PROCESS;
    count <= int_count;
END model;
```

Counters

- Counters are commonly used for timers and clock dividers
- Always use a synchronous counter
- Ripple counters (where one stage clocks the next stage) use less gates, but produce too much delay for today's applications and clock speeds

Counters

- Stop before rolling over

```
ARCHITECTURE model of Count4 IS
    SIGNAL int_count : STD_LOGIC_VECTOR(3 downto 0);
    CONSTANT MAX_VAL : STD_LOGIC_VECTOR(3 downto 0) := “1100”;
BEGIN
    Simple : PROCESS(clk, reset_n)
            BEGIN
                    IF (reset_n = ‘0’) THEN
                            int_count <= “0000”;
                    ELSIF (rising_edge(clk)) THEN
                            IF (int_count = MAX_VAL) THEN
                                    int_count <= “0000”;
                            ELSE
                                    int_count <= int_count + 1;
                            END IF;
                    END IF;
            END PROCESS;
    count <= int_count;
END model;
```

Using a counter to make a timer

- The clock on the DE0 board is 50 MHz
    - How do you use that to make a ½ second timer?
    - How many bits would you need?

½ second delay = 500 ms

We need to count the number of 50 MHz clock cycles in 500 ms

50MHz clock has a period of 20 ns

# counts = delay / clock period

# counts = $500 \times 10^{-3} / 20 \times 10^{-9} = 25 \times 10^{6}$

$25 \times 10^{6} = 17D7840_{H}$

*Don't forget to subtract 1 since we start at 0

```
ENTITY delay_unit IS
   PORT(clk, reset_n          : IN STD_LOGIC;
        flag                  : OUT STD_LOGIC);  --this flag is raised every 500ms
END delay_unit;

ARCHITECTURE behave of delay_unit IS
   signal count   : std_logic_vector(27 downto 0);
   constant half_sec_count : std_logic_vector(27 downto 0) := X"17D783F";
BEGIN
   PROCESS(clk, reset_n) IS
      BEGIN
         if (reset_n = '0') then           --asynchronous reset
            count <= (others => '0');
         elsif (clk'event and clk = '1') then
            if (count = half_sec_count) then  --stop when you reach 1/2 sec
               count <= (others => '0');
            else
               count <= count + 1;
            end if;
         end if;
      end process;
   PROCESS(clk, reset_n) IS  --this process produces a 1 clk cycle flag every
                             --half second.
      BEGIN
         if (reset_n = '0') then
            flag <= '0';
         elsif (clk'event and clk = '1') then
            if (count = half_sec_count) then
               flag <= '1';
            else
               flag <= '0';
            end if;
         end if;
      end process;

   end behave;
```

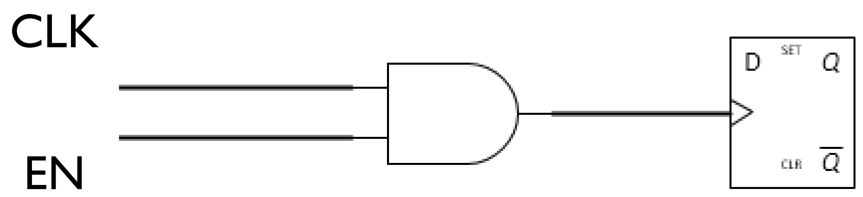
Using an enable signal

- Within a clocked process, if you only want something to happen when a certain condition exists
  - Do not gate with clock signal
  - Use an enable

```
IF (rising_edge(clk)) THEN
   IF enable = '1' THEN
      do something;
    ELSE
      stay the same;
```

Using an enable signal

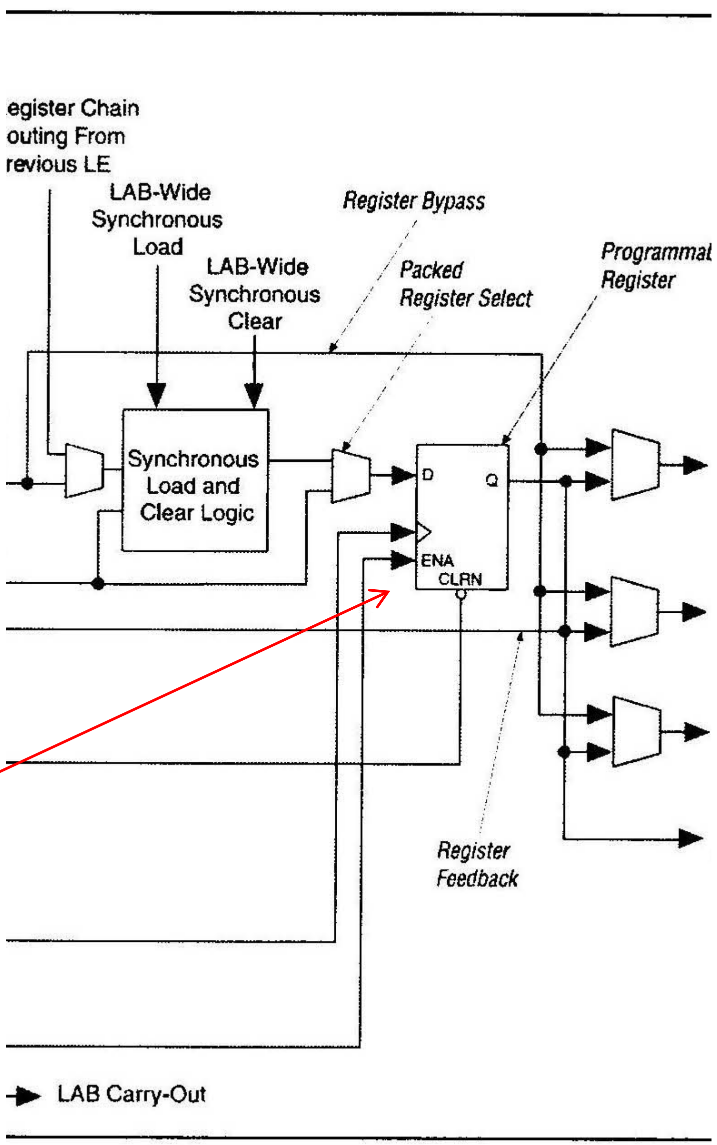
- The only thing that goes in the clock line is the clock
  - **If (rising_edge(clk)) then** or
  - **if (clk'event and clk='1') then**
- Including the enable creates a gate
  - **If (rising_edge(clk) and enable = '1') then**



  - Why is this bad?

Using an enable signal

```
Begin
    If reset_n = '0' then
        Q <= '0';
    Elsif rising_edge(clk) then
        If enable = '1' then
            Q <= D;
        End if;
    End if;
End process;
```



Flip-flop in the cyclone II
Is designed with enable

BCD Counter

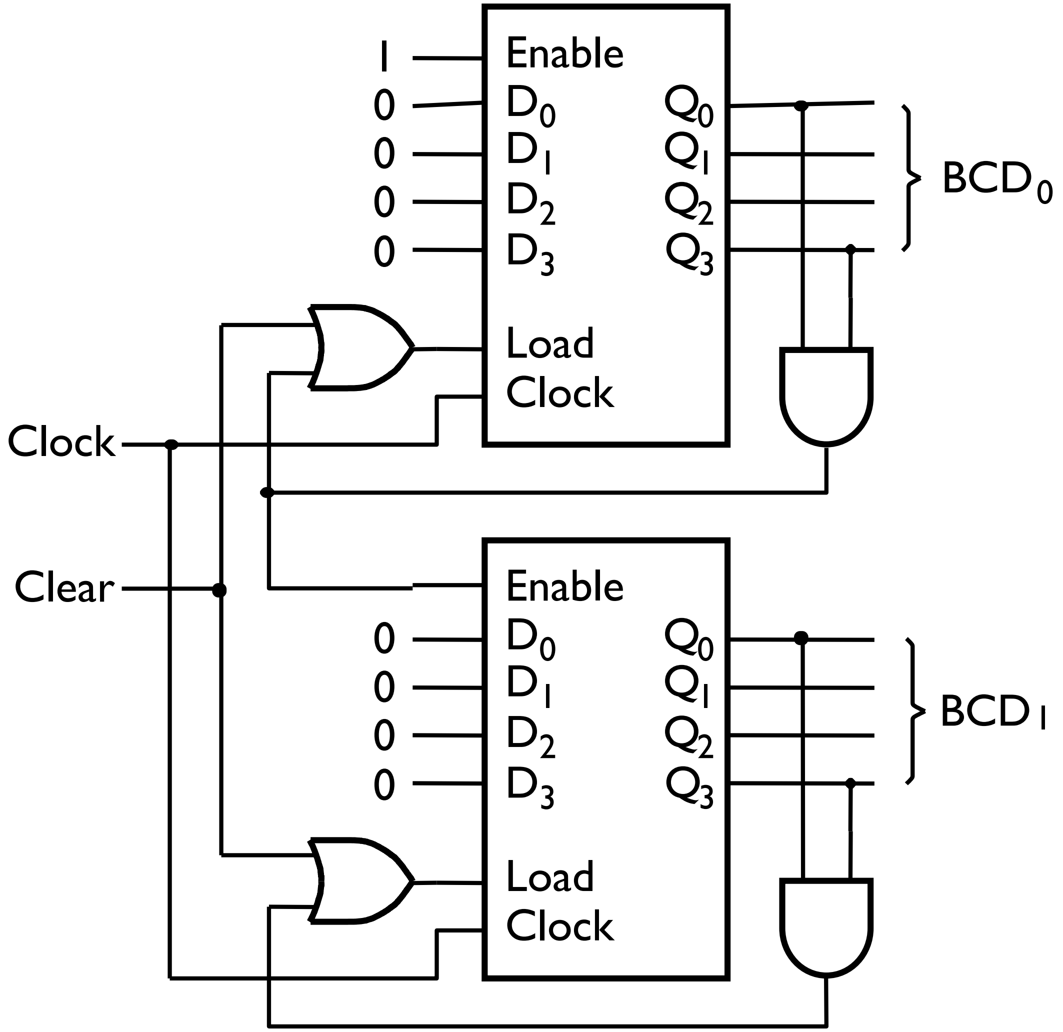
- Binary Coded Decimal is a 4 bit code.
  - Used to display decimal digits
  - Counting order
    - One digit: 0,1,2,3,4,5,6,7,8,9,0…..
    - Two digits: ….20,21,22,23,24,25,26,27,28,29,30…
  - Follows regular decimal counting order
    - However, each digit is represented by 4 bits

0001 1001 1001 = 199

0010 0000 0000 = 200

- A BCD counter needs to know to roll over at 1001, not 1111

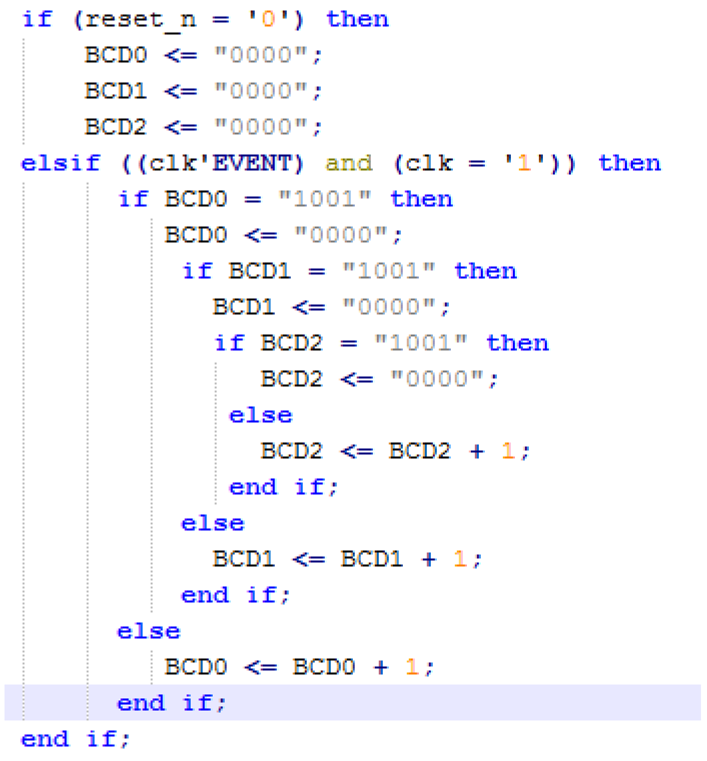
BCD Counter



1
0
0
0
0
Enable
D_0
D_1
D_2
D_3
Q_0
Q_1
Q_2
Q_3
BCD_0
Load
Clock
Clock
Clear
Enable
0
0
0
0
D_0
D_1
D_2
D_3
Q_0
Q_1
Q_2
Q_3
BCD_1
Load
Clock

BCD Counter

- Starting with LSD (the least significant digit), check if it will roll over. If it rolls over, check if it causes next bit to roll over, etc…
- If no roll over, just increment



```
if (reset_n = '0') then
    BCD0 <= "0000";
    BCD1 <= "0000";
    BCD2 <= "0000";
elsif ((clk'EVENT) and (clk = '1')) then
      if BCD0 = "1001" then
         BCD0 <= "0000";
          if BCD1 = "1001" then
            BCD1 <= "0000";
            if BCD2 = "1001" then
               BCD2 <= "0000";
             else
               BCD2 <= BCD2 + 1;
             end if;
          else
            BCD1 <= BCD1 + 1;
          end if;
      else
         BCD0 <= BCD0 + 1;
      end if;
end if;
```