ECE 2214

Digital Logic Design

L-24 Registers and counters with CAD tools

Fall 2022

Outline

- Use of registers and counters with schematics in QUARTUS II
- Registers and counters in VHDL code
- Registers and counters module instantiation
- Use of sequential statement to implement registers and counters
- Application example

Summary of previous lecture

- Synchronous counters with D flip-flops
  - Not the best choice
  - Needs Enable to controls external logic
  - More complex design
- N-bit implementation is achieved

  $D_0 = Q_0 \oplus E_N$

  $D_1 = Q_1 \oplus Q_0E_N$

  $D_2 = Q_2 \oplus Q_1Q_0E_N$

  $D_3 = Q_3 \oplus Q_2Q_1Q_0E_N$

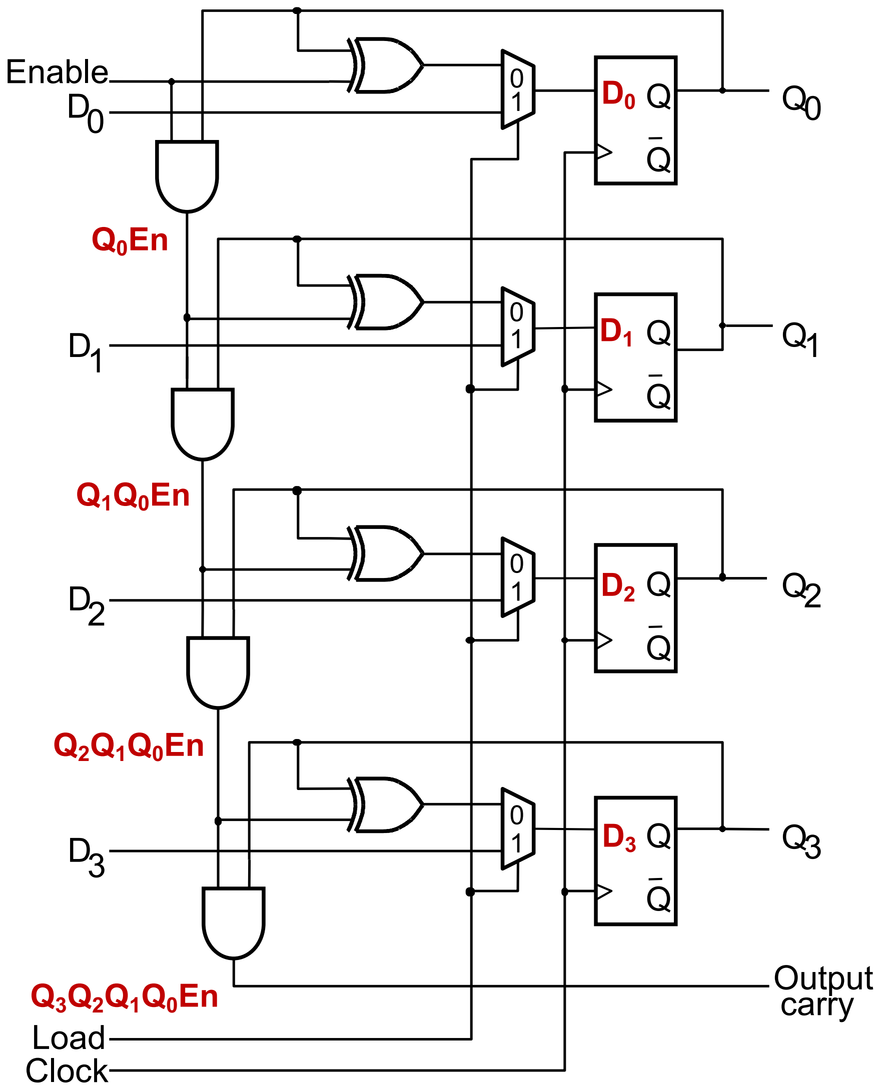
  …

  …

  …

  $D_n = Q_n \oplus Q_{n-1}. . .Q_1Q_0E_N$

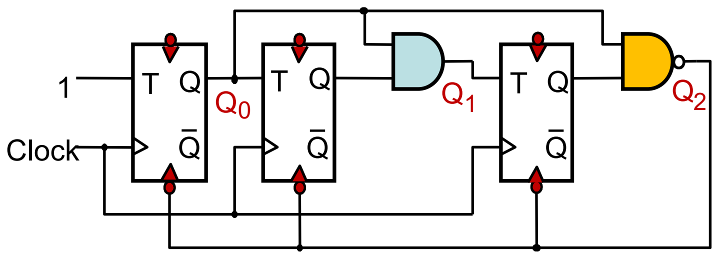
- Parallel loads are achieved with muxes at the D inputs
  - Select from parallel load of Q_x



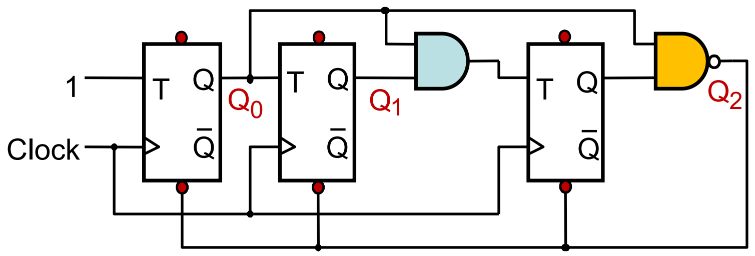
A counter with parallel-load capability.

Summary of previous lecture

- Synchronous reset of counters
  - All changes in the circuit are clock synchronized
  - Any N-bit binary counter resets automatically after reaching the number 2^N-1
  - Preset and reset of F-F can be used to set the counter at any counting sequence
  - Designer should select unique features of the counting sequence desired and implement a logic to activate the load/reset pin



- Asynchronous reset of counters
  - Any change in the counting sequence is active immediately
  - External logic is also used to reset sequence
  - Last counting sequence is of short duration
  - Designer should count one state more to achieve proper counting
- Valid for down or up counters



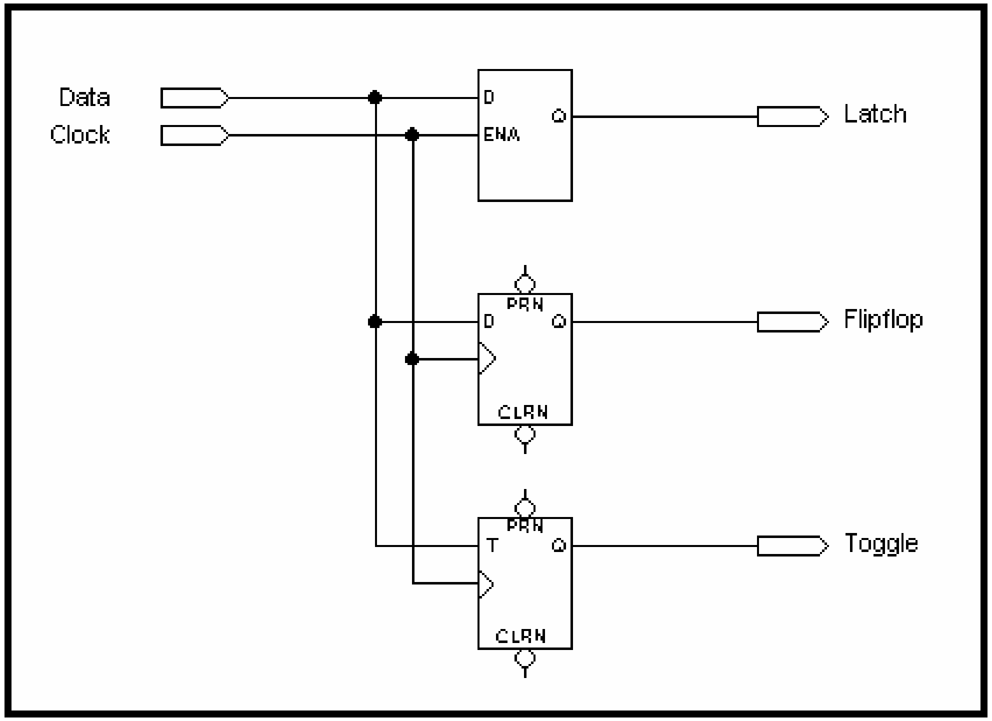
Summary of previous lecture

- Other types of counters
  - BCD counters
    - A binary-coded counter base 10
    - Each digit count from 0-9
    - Any excess from the least significant digit (LCD) is added to the next digit
    - **Similar procedure applies to counter in any other radix**
      - **i.e. Radix 13: need 4 bits to count from 0-C (0000-1100)**
        any excess the LCD is added to the next digit
  - Ring counters
    - Special type of counters that only one of the outputs is ON at a time
    - Shift the output circularly along all outputs
  - Johnson counters
    - A modification to the ring counter
    - Q complemented of the last bit is fed back to the first input
    - Circuit outputs two patterns 4-bit counter as:
      (1000, 1100, 1110, 1111) & (0111, 0011, 0001, 0000)

Registers and counters with schematics in Quartus II

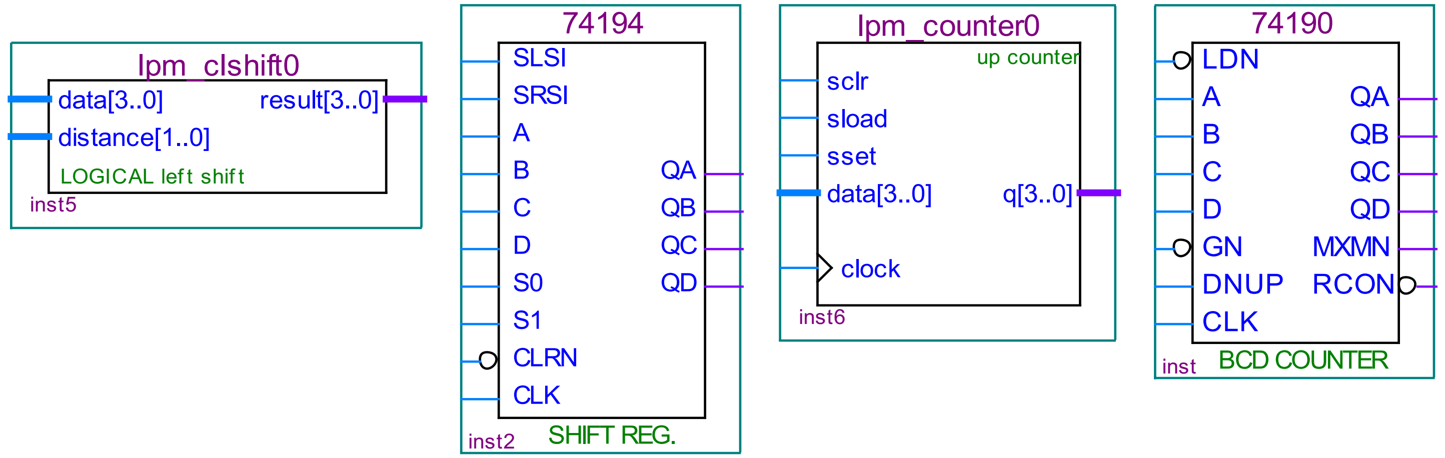
- Flip-flops, registers and counters can be accomplished in many different ways in QUARTUS II
- Implementation depends on design requirements
- QUARTUS provides default modules for flip-flops and counters

- T, D and JK flip-flops are available with clear and preset
- Located in the "primitive" library



Registers and counters with schematics in Quartus II

- Registers and counters have configurable pre-built modules



- LPM (library of parameterized modules) also includes flip-flops, registers, counters and other useful circuits
    - Are configurable
- Other libraries provide modules for commercially available circuits

Registers and counters in VHDL

- Registers and counters are performed in several ways in QUARTUS II using VHDL
- There are two main ways for the implementation:
  - Instantiate the pre-defined module from the LPM library
  - Implement the circuits with sequential statements

- Register and counter module instantiation:
  - Open wizard from “\Tools\IP Catalog” in QUARTUS
  - Select a configurable pre-defined module that fit the design requirements
  - Declare the module as a component in your root level design
  - Use GENERIC MAP construction to assign values to the parameters of the instantiated sub-circuit
    - Similar to PORT MAP construction
  - Use PORT MAP construction to connect variables form the top-level design to the low-level design

Assigning constants and variables to a pre-built module in Quartus II

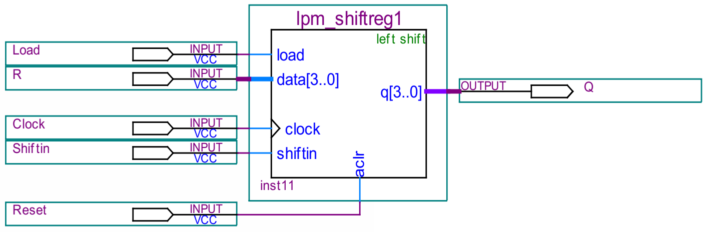
- Values can be assigned when the component is instantiated as:

```
…
Instance: module_name
        GENERIC MAP ( Const_assign1,{ Const_assign_x})
        PORT MAP ( Var_assign1,{ Var_assign_x});
```

- Note that values can be assigned to more than one variable
- GENERIC MAP assigns **constant values** to variables defined in the pre-built module
- PORT MAP assigns **external variables** to variables defined in pre-built module
- Some variables are optional (will take the default values if not explicitly defined)
- Careful revision of the module information is required in order to achieve the desired results

Example of pre-built module use in Quartus II

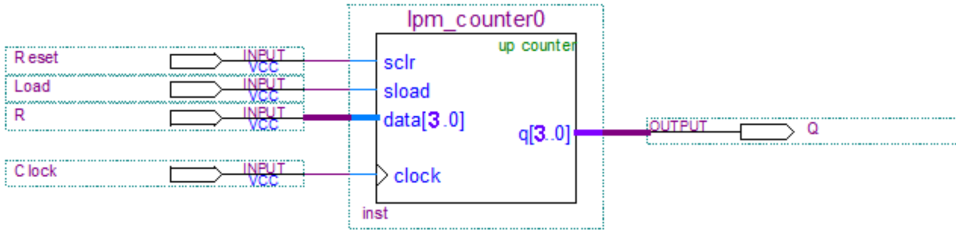
Example 1: Use of VHDL code to instantiate a LPM shift register module



```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
LIBRARY lpm ;
USE lpm.lpm_components.all ;

ENTITY shift IS
    PORT ( Clock            : IN    STD_LOGIC ;
           Reset            : IN    STD_LOGIC ;
           Shiftin, Load    : IN    STD_LOGIC ;
           R                : IN    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           Q                : OUT   STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END shift ;

ARCHITECTURE Structure OF shift IS
BEGIN
    instance: lpm_shiftreg
        GENERIC MAP (LPM_WIDTH => 4, LPM_DIRECTION => "RIGHT")
        PORT MAP (data => R, clock => Clock, aclr => Reset,
            load => Load, shiftin => Shiftin, q => Q ) ;
END Structure ;
```

Example of pre-built module use in Quartus II

Example 2: Use of VHDL code to instantiate a LPM counter module



```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
LIBRARY lpm ;
USE lpm.lpm_components.all ;

ENTITY Count4 IS
    PORT ( Clock        : IN     STD_LOGIC ;
           Load         : IN     STD_LOGIC ;
           Reset        : IN     STD_LOGIC ;
           R            : IN     STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           Q            : OUT    STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END Count4 ;

ARCHITECTURE Behavior OF Count4 IS
BEGIN
    Instance: lpm_counter
        GENERIC MAP (LPM_WIDTH => 4, LPM_DIRECTION => “UP”)
        PORT MAP (data => R, clock => Clock, sclr => Reset,
                        sload => Load, q => Q);

END Behavior;
```

Sequential statements to implement registers and counters

- There are several forms to implement registers and counters
- **Example 3.** Implementation of an N-bit register with asynchronous clear

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY regn IS
    GENERIC ( N : INTEGER := 16 ) ;
    PORT ( D              : IN   STD_LOGIC_VECTOR(N−1 DOWNTO 0) ;
           Resetn, Clock  : IN   STD_LOGIC ;
           Q              : OUT  STD_LOGIC_VECTOR(N−1 DOWNTO 0) ) ;
END regn ;

ARCHITECTURE Behavior OF regn IS
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            Q <= (OTHERS => '0') ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            Q <= D ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

- Code performs a register of any number of digits (variable N)
- Assignment operator ":=" is used to set the default value of N
- The GENERIC assignment (N=16) is equivalent to Q="0000000000000000"
- The statement (OTHERS => '0') result in a '0' assigned to each bit regardless of the total number selected

Use of sequential statement to implement registers and counters

- **Example 4.** Implementation of a 4-bit up-counter

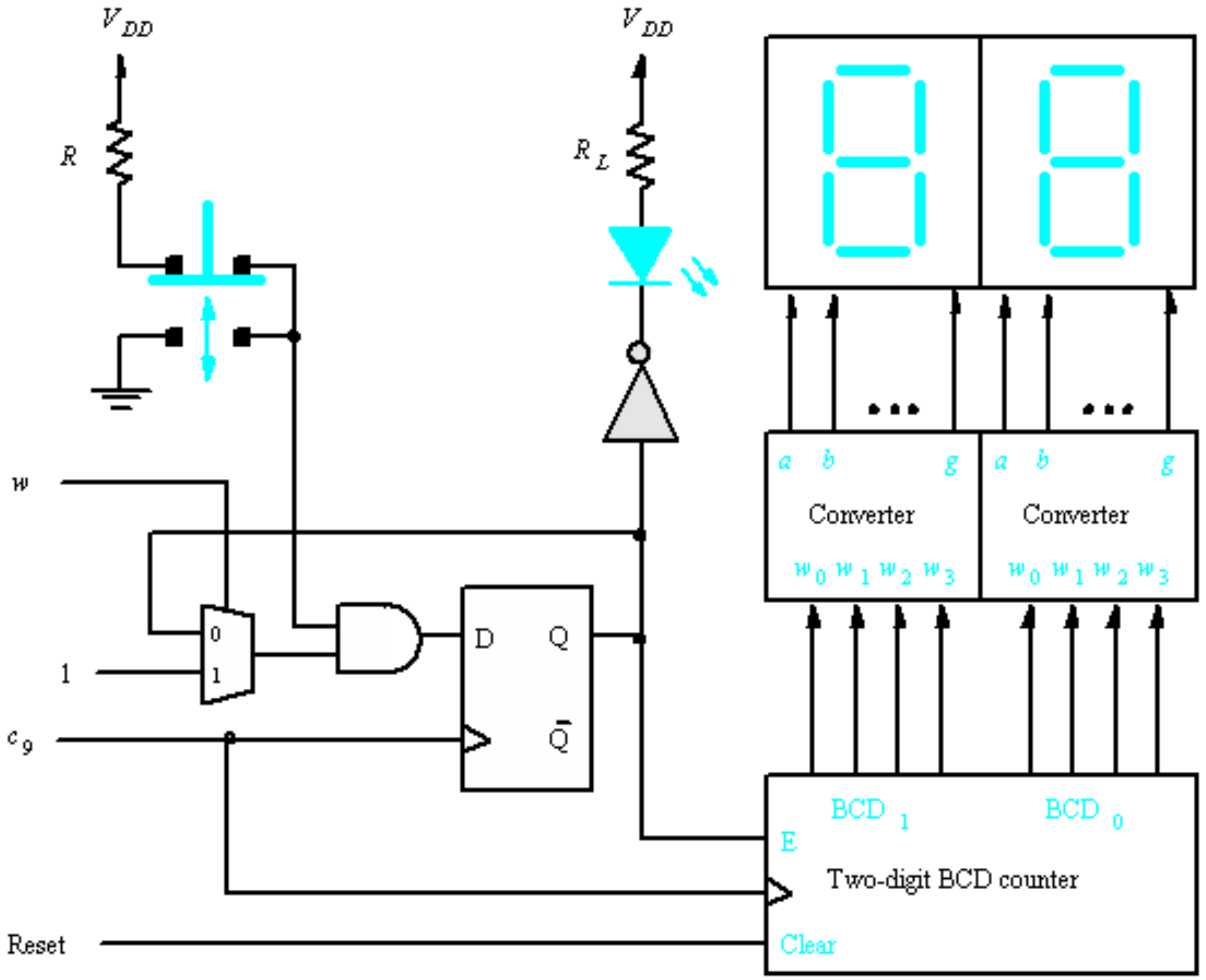
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

ENTITY upcount IS
    PORT ( Clock, Resetn, E   : IN    STD_LOGIC ;
           Q                  : OUT   STD_LOGIC_VECTOR (3 DOWNTO 0)) ;
END upcount ;

ARCHITECTURE Behavior OF upcount IS
    SIGNAL Count : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
BEGIN
    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN
            Count <= "0000" ;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
            IF E = '1' THEN
                Count <= Count + 1 ;
            ELSE
                Count <= Count ;
            END IF ;
        END IF ;
    END PROCESS ;
    Q <= Count ;
END Behavior ;
```

- External variables can not be assigned more than twice inside the sequential statement
  - Error produced due to assignment of multiple constants to a variable
- Required a local variable (Count)
- You can assign the local variable before or after the sequential assignment
- Compiler generates a multiplexer

Application example of counters

- Design of a circuit to measure the reaction time of a person
- **Motivation:**
  - Digital circuits function at a remarkable speed: average delay of logic gates can be in the order of few nanoseconds
  - Human's reaction is very slow compared to the speed of digital circuits
  - Let's construct a circuit that can measure the reaction time of a person for a specific event
  - The event is the time response of a person detecting a small light on (Light emitting diode, LED)
  - The reaction time is measured when the person press a switch and turn the light off
- **System description:**
  - 2-digit system to display up to a hundred fractions of a second
  - The clock available changes at a rate of 102.4kHz

Application example

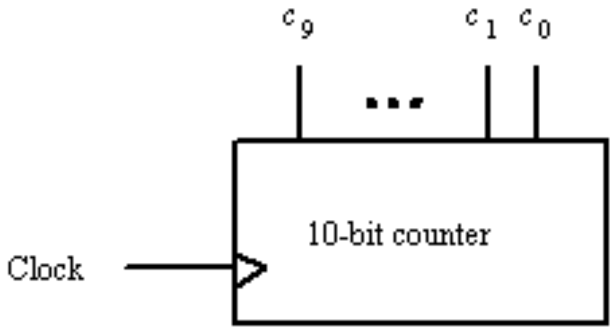


Application example

- Implementation:
  - **Clock signal:** Usually of high frequency to control many synchronic processes in the design and also to ensure stability of the clock signal
  - Clock should be reduced to 100Hz to measure the time at a resolution of 1/100 seconds (from 0/100 to 99/100)
- A binary counter can be used to divide the original clock signal and obtain the 100Hz signal needed. Calculation of the number of bit or flip-flops needed:

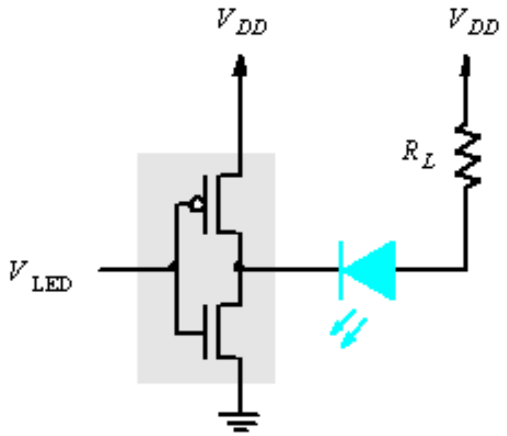
  102400 Hz/100Hz= 1024

  - We know from previous lectures that binary counter divide numbers by 2^N, with N being the number of bits. Therefore 2^N=1024
    - Need a counter with 10 bits



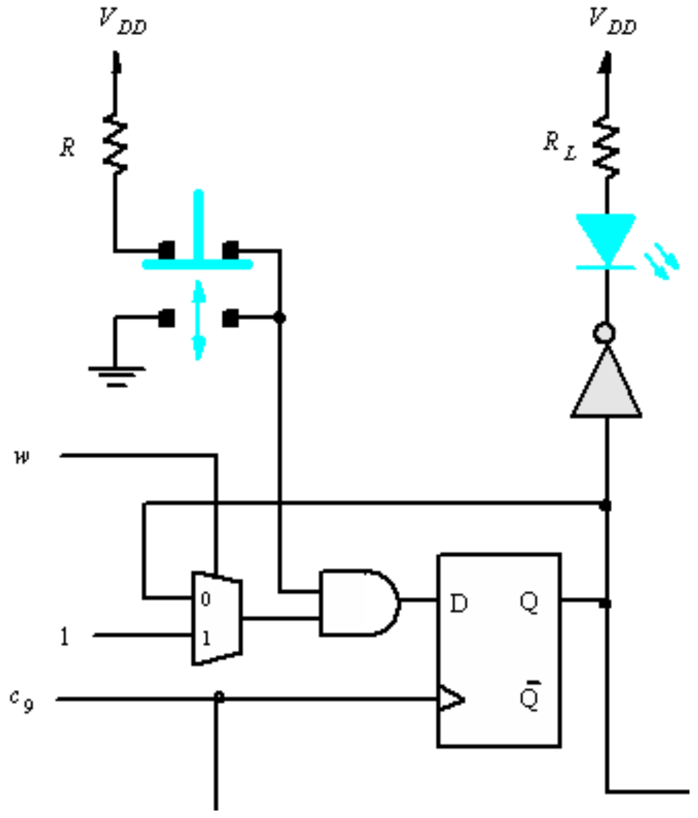
Application example

- **LED circuit:**
  - The LED will emit light when current flows through the device
  - Current controlled by the NOT gate
- If the output of the NOT gate is ‘0’, V_{LED} is ON (a current flows to ground through the lower transistor)
- If the output of the NOT gate is ‘1’, V_{LED} is OFF (same potential is applied in both terminals of the LED).
- Note that V_{LED} is controlled by the output Q of the D flip-flop



Application example

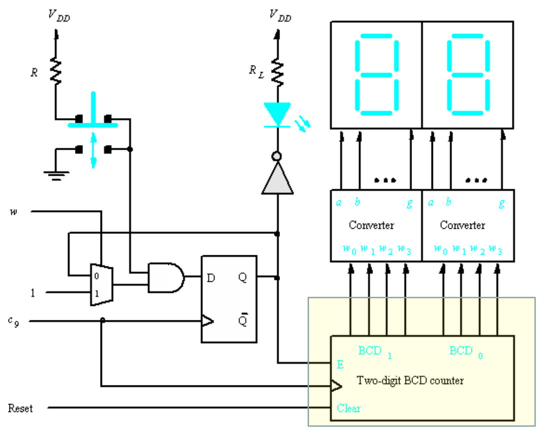
- **Controlling the switch to reset the LED**:
  - LED has to be “ON” when the event “light” occurs
  - LED should be “OFF” when switch is pressed
  - The clock signal and the switch input have to be gated to stop the clock signal when the switch is pressed
  - The input **w** is needed to start the reaction measurement
  - w =1 output a 1 from the multiplexer
  - It is combined with the ‘1’ from the switch in default state
  - This makes the output of the F-F to 1 and …
  - Switch the LED “ON”



Application example

- **Counter:**
- Counting the time signal
- Display counting in BCD (0-99)/100 seconds
- Two digit are needed
- Clock provides the time base
- Reset enable to re-start the measurement
- Two-digit BCD counter can be implemented in VHDL as:

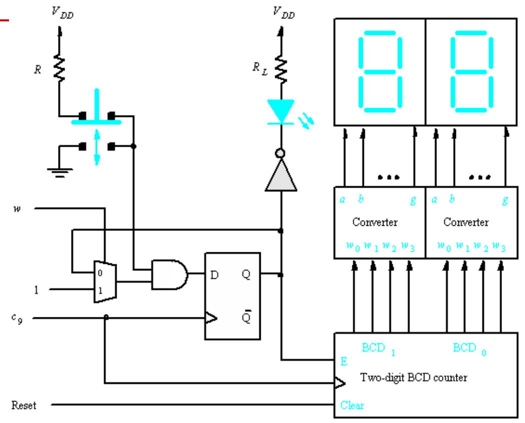
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

ENTITY BCDcount IS
    PORT (     Clock          : IN         STD_LOGIC ;
               Clear, E           : IN         STD_LOGIC ;
               BCD1, BCD0     : BUFFER    STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END BCDcount ;

ARCHITECTURE Behavior OF BCDcount IS
BEGIN
    PROCESS ( Clock )
    BEGIN
        IF Clock'EVENT AND Clock = '1' THEN
            IF Clear = '1' THEN
                    BCD1 <= "0000" ; BCD0 <= "0000" ;
            ELSIF E = '1' THEN
                IF BCD0 = "1001" THEN
                    BCD0 <= "0000" ;
                    IF BCD1 = "1001" THEN
                        BCD1 <= "0000";
                    ELSE
                        BCD1 <= BCD1 + '1' ;
                    END IF ;
                ELSE
                    BCD0 <= BCD0 + '1' ;
                END IF ;
            END IF ;
        END IF;
```



Application example

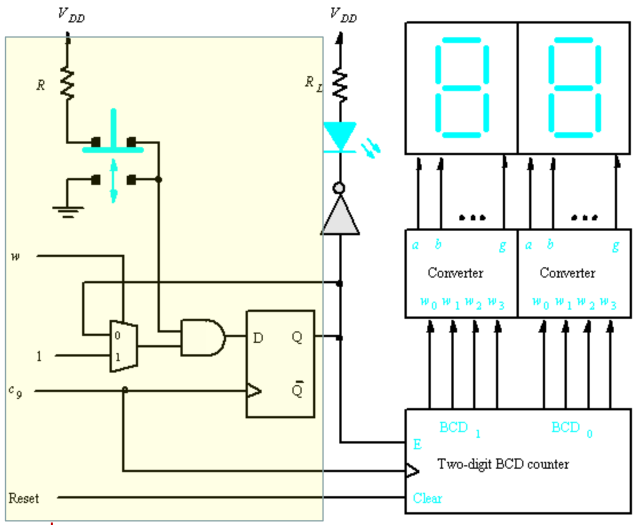
- The whole reaction timer application can be implemented in VHDL

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY reaction IS
	PORT ( c9, Reset		: IN		STD_LOGIC ;
		w, Pushn		: IN		STD_LOGIC ;
		LEDn			: OUT	STD_LOGIC ;
		Digit1, Digit0	: BUFFER	STD_LOGIC_VECTOR(1 TO 7) ) ;
END reaction ;

ARCHITECTURE Behavior OF reaction IS
	COMPONENT BCDcount
		PORT ( Clock		: IN		STD_LOGIC ;
			Clear, E		: IN		STD_LOGIC ;
			BCD1, BCD0	: BUFFER	STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
	END COMPONENT ;
	COMPONENT seg7
		PORT ( bcd	: IN	STD_LOGIC_VECTOR(3 DOWNTO 0) ;
			leds	: OUT	STD_LOGIC_VECTOR(1 TO 7) ) ;
	END COMPONENT ;
	SIGNAL LED : STD_LOGIC ;
	SIGNAL BCD1, BCD0 : STD_LOGIC_VECTOR(3 DOWNTO 0) ;
```



Application example

```
BEGIN
    flipflop: PROCESS (c9,Pushn,w)
    BEGIN
        WAIT UNTIL c9'EVENT AND c9 = '1' ;
        IF Pushn = '0' THEN
            LED <= '0' ;
        ELSIF w = '1' THEN
            LED <= '1' ;
        END IF ;
    END PROCESS ;

    LEDn <= NOT LED ;
    counter: BCDcount PORT MAP ( c9, Reset, LED, BCD1, BCD0 ) ;
    seg1 : seg7 PORT MAP ( BCD1, Digit1 ) ;
    seg0 : seg7 PORT MAP ( BCD0, Digit0 ) ;
END Behavior ;
```



- Notice that c9 is the output of bit 9 of the counter used to reduce the frequency of the original clock to 100Hz
- It is not shown in this design

Summary

- Introduced
  - Implementation of registers and counters in QUARTUS II using schematics
  - Registers and counters in VHDL code
    - Register and counter module instantiation
    - Use of sequential statement to implement registers and counters
  - Application examples
- Reference
  - Fundamentals of Digital Logic with VHDL Design 3/e, Stephen Brown, Zvonko Vranesic; McGraw-Hill, 2009. Chapter 7, pp. 423-471
  - Digital Design; Principles and Practices. Fourth Edition. John F. Wakerly, Prentice Hall, 2006. ISBN 0-13-186389-4. Chapter 7, pp. 710-758
- Next lecture
  - Summary of topics to be evaluated in midterm 2 next Lecture