香港中文大學
The Chinese University of Hong Kong

CENG3430 Rapid Prototyping of Digital Systems

Lecture 04: Combinational Circuit and Sequential Circuit

Ming-Chang YANG

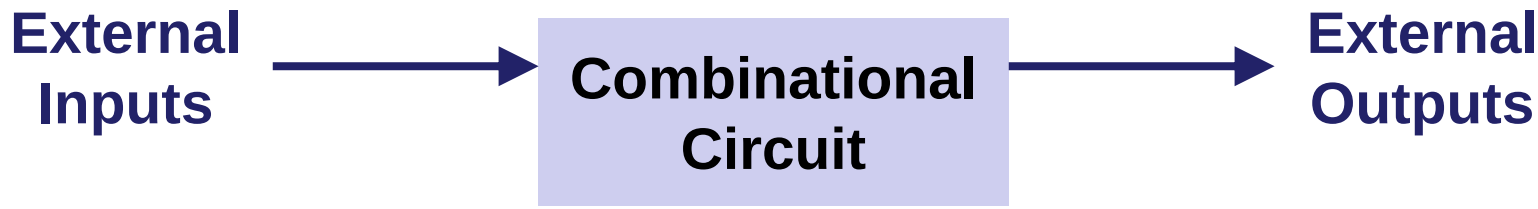
mcyang@cse.cuhk.edu.hk

Outline

- Combinational Circuit and Sequential Circuit
  - Combinational Circuit: No Memory
    - Decoder
    - Multiplexer
    - Bi-directional Bus
  - Sequential Circuit: Has Memory
    - Latch
    - Flip-flop
      - Asynchronous Reset and Synchronous Reset
    - Finite State Machine (FSM)
      - Clock Edge Detection
      - Direct Feedback Path
      - Types and Examples

Combinational Circuit

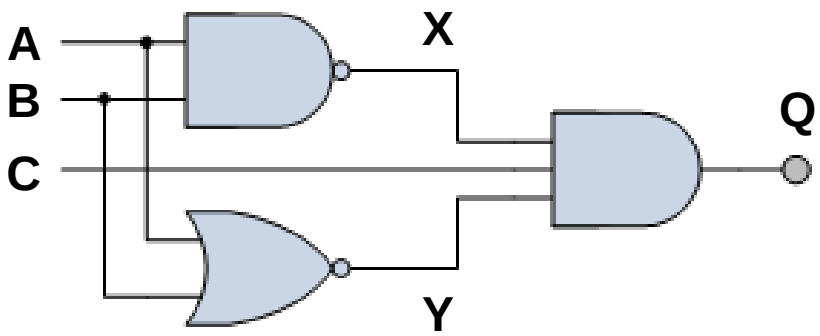
- **Combinational Circuit**: **no memory**
  - Outputs are a function of the *present* inputs only.
    - As soon as inputs change, the values of previous outputs are lost.
    - It has no internal state (i.e., has no memory).
    - Common Examples: *Comparator*, *Encoder/Decoder*, *Full/Half Adder*, *Multiplexer*, *Bi-directional Bus*, etc.
  - **Rule**: You can build a combinational circuit using **either** concurrent statements (i.e., statements outside `process`) **or** sequential statements (i.e., statements inside `process`) .



Modeling Combinational Logic (1/3)

- Three typical ways for modeling a combinational logic:
  1) **Logic/Schematic Diagram** shows the wiring and connections of each individual logic gate.
  2) **Boolean Expression** is an expression in Boolean algebra that represents the logic circuit.
  3) **Truth Table** provides a concise list that shows all the output states for each possible combination of inputs

Logic/Schematic Diagram



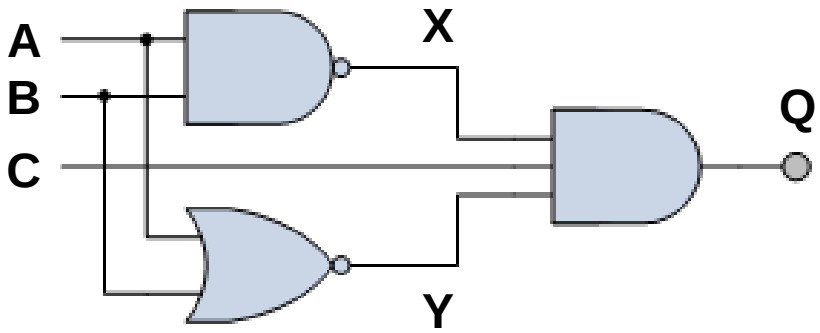

```
architecture com_arch of comb is
signal X, Y: std_logic;
begin
  X <= not (A and B);
  Y <= not (A or B);
  Q <= (X and C) and Y;
end com_arch;
```

https://www.electronics-tutorials.ws/combination/comb_1.html

Modeling Combinational Logic (2/3)

- Three typical ways for modeling a combinational logic:
  1) **Logic/Schematic Diagram** shows the wiring and connections of each individual logic gate.
  2) **Boolean Expression** is an expression in Boolean algebra that represents the logic circuit.
  3) **Truth Table** provides a concise list that shows all the output states for each possible combination of inputs

Logic/Schematic Diagram



How?

Boolean Expression

$$Q = \overline{(A \cdot B)} \cdot \overline{(A + B)} \cdot C$$

```
architecture com_arch of comb is
begin
  Q <= (not (A and B)) and
       (not (A or B)) and C;
end com_arch;
```

https://www.electronics-tutorials.ws/combination/comb_1.html

Modeling Combinational Logic (3/3)

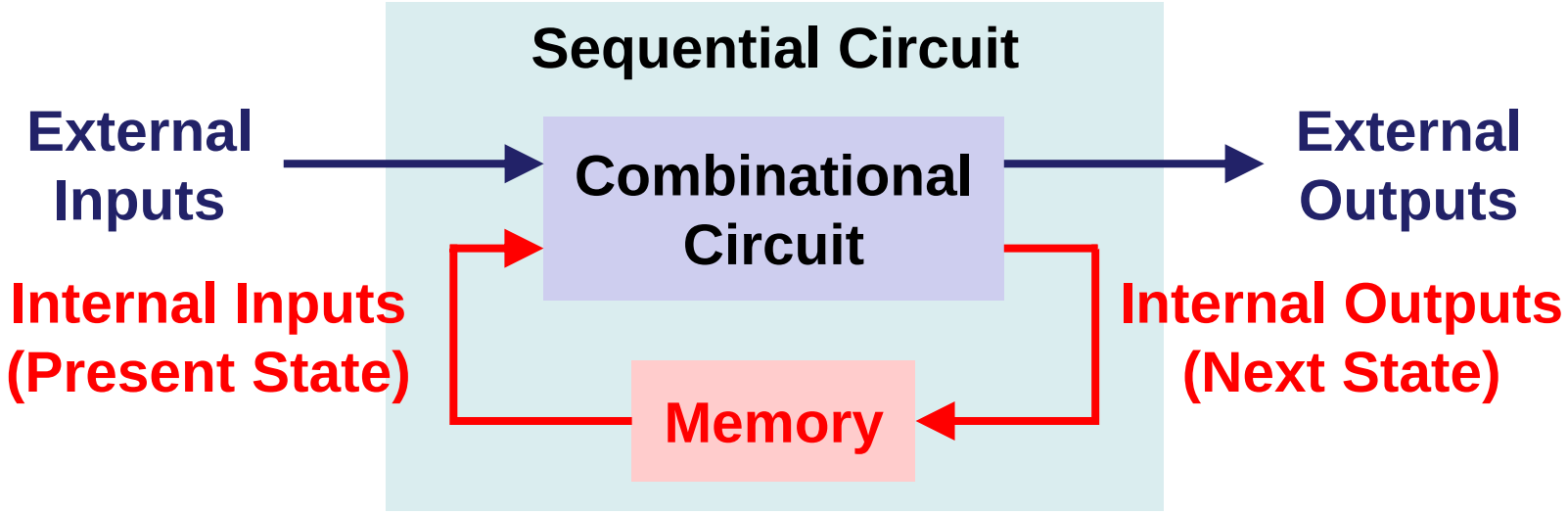
- Three typical ways for modeling a combinational logic:
  1) **Logic/Schematic Diagram** shows the wiring and connections of each individual logic gate.
  2) **Boolean Expression** is an expression in Boolean algebra that represents the logic circuit.
  3) **Truth Table** provides a concise list that shows all the output states for each possible combination of inputs

A	B	C	Q
0	0	0	0
0	**0**	**1**	**1**
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

```
process (A, B, C)
begin
   if( A = '0' and B = '0' and C = '1' ) then
     Q <= '1';
   else
     Q <= '0';
   end if;
end process;
```

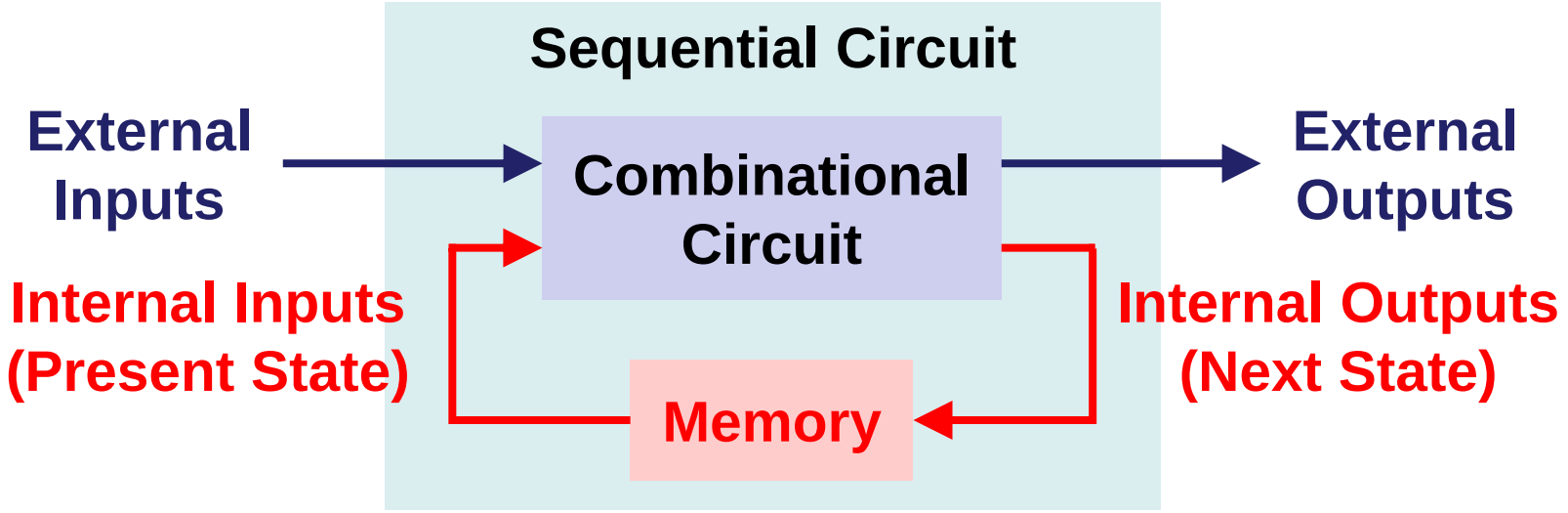
Sequential Circuit

- **Sequential Circuit**: **has memory**
  - Outputs are a function of the *present* inputs and the *previous* outputs (i.e., the **internal state**).
    - It changes outputs based on inputs; but the outputs also depend upon previous outputs (i.e., the **internal state**) (i.e., has memory).
    - Example: *Latch*, *Flip-Flop*, *Finite State Machine*, etc.
  - **Rule**: You **must** build a sequential circuit with **only** sequential statements (i.e., statements inside `process`).



Combinational vs. Sequential Circuit

- **Combinational Circuit**: **no memory**
  - ① Outputs are a function of the *present* inputs only.
  - ② **Rule**: Use **either** concurrent **or** sequential statements.
- **Sequential Circuit**: **has memory**
  - ① Outputs are a function of the *present* inputs and the *previous* outputs (i.e., the **internal state**).
  - ② **Rule**: **Must** use sequential statements (i.e., `process`) .

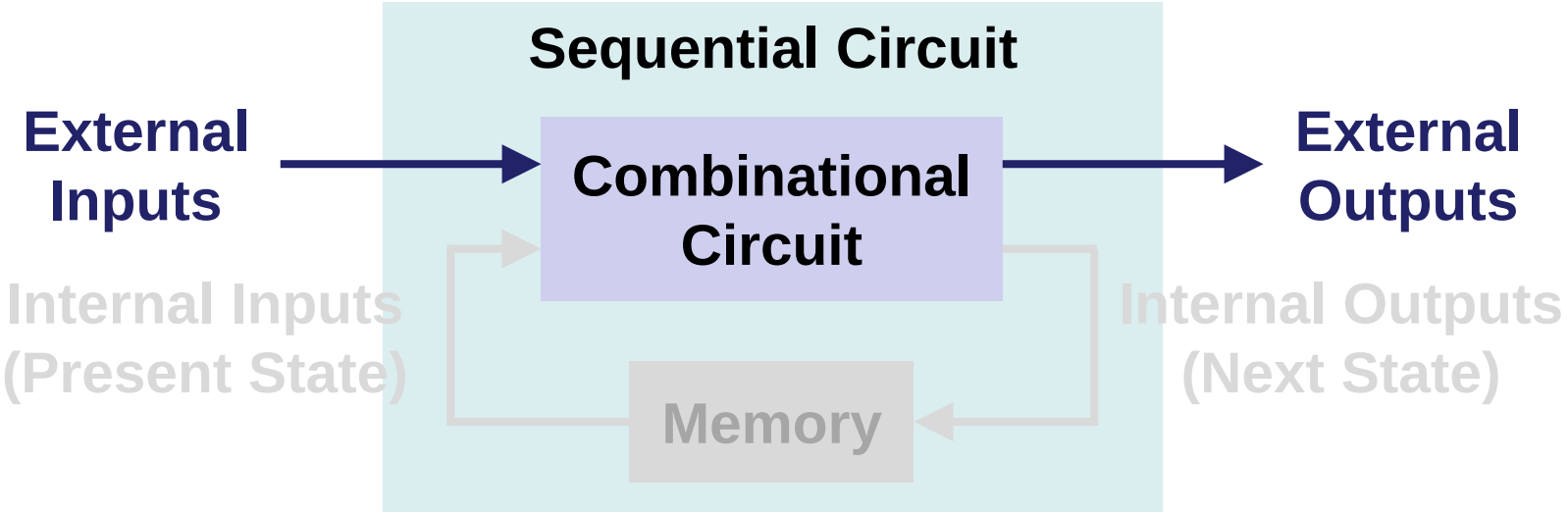


Outline

- Combinational Circuit and Sequential Circuit
  - Combinational Circuit: No Memory
    - Decoder
    - Multiplexer
    - Bi-directional Bus
  - Sequential Circuit: Has Memory
    - Latch
    - Flip-flop
      - Asynchronous Reset and Synchronous Reset
    - Finite State Machine (FSM)
      - Clock Edge Detection
      - Direct Feedback Path
      - Types and Examples

Recall: Combinational Circuit

- **Combinational Circuit**: **no memory**
  - ① Outputs are a function of the *present* inputs only.
  - ② **Rule**: Use **either** concurrent **or** sequential statements.
- **Sequential Circuit**: **has memory**
  - ① Outputs are a function of the *present* inputs and the *previous* outputs (i.e., the **internal state**).
  - ② **Rule**: **Must** use sequential statements (i.e., `process`) .



Combinational Logic as a Process

- Consider a simple combinational logic:

```
c <= a and b;
```

- This logic can be also modeled as a process:
  - All signals referenced in process must be in sensitivity list.

```
entity And_Good is
  port (a, b: in std_logic; c: out std_logic);
end And_Good;
architecture Synthesis_Good of And_Good is
begin
  process (a, b) -- sensitive to signals a and/or b
  begin
    c <= a and b; -- c updated
  end process;
end;
```

Combinational Circuit: Decoder (1/2)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity decoder_ex is
port (in0,in1: in std_logic;
      out00,out01,out10,out11: out std_logic);
end decoder_ex;
architecture decoder_ex_arch of decoder_ex is
begin
  process (in0, in1)
  begin
    if in0 = '0' and in1 = '0' then
      out00 <= '1';
    else
      out00 <= '0';
    end if;
    if in0 = '0' and in1 = '1' then
      out01 <= '1';
    else
      out01 <= '0';
    end if;
```

out00

out01

in 0	in 1	out 00	out 01	out 10	out 11
0	**0**	**1**	0	0	0
0	**1**	0	**1**	0	0
1	0	0	0	1	0
1	1	0	0	0	1

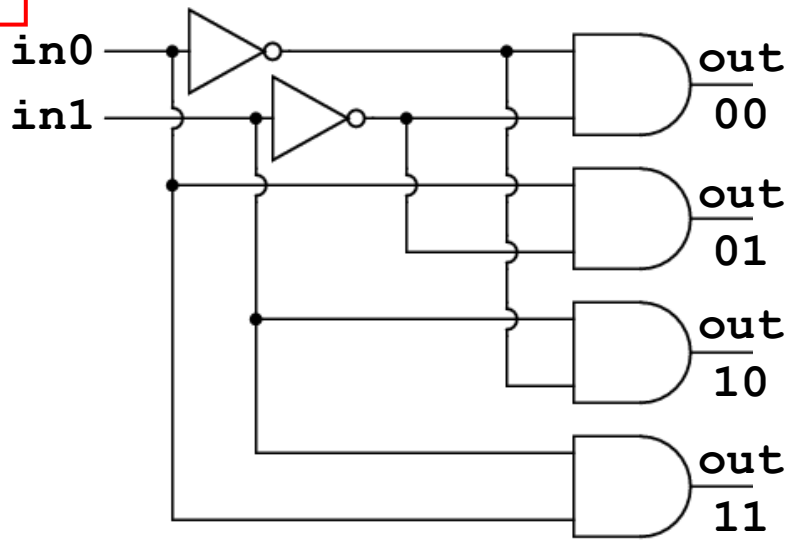
Combinational Circuit: Decoder (2/2)

```
    ...
    if in0 = '1' and in1 = '0' then
      out10 <= '1';
    else
      out10 <= '0';
    end if;
    if in0 = '1' and in1 = '1' then
      out11 <= '1';
    else
      out11 <= '0';
    end if;
  end process;
end decoder_ex_arch;
```

out10

out11

in 0	in 1	out 00	out 01	out 10	out 11
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



https://www.allaboutcircuits.com/textbook/digital/chpt-9/decoder/

Class Exercise 4.1

Student ID: _____ Date:
Name: _____ _____

- Implement the Encoder based on the given table:

```
port(
                                                              );
…
architecture encoder_ex_arch of encoder_ex is
begin
  process (                          )
  begin




  end process;
end encoder_ex_arch;
```

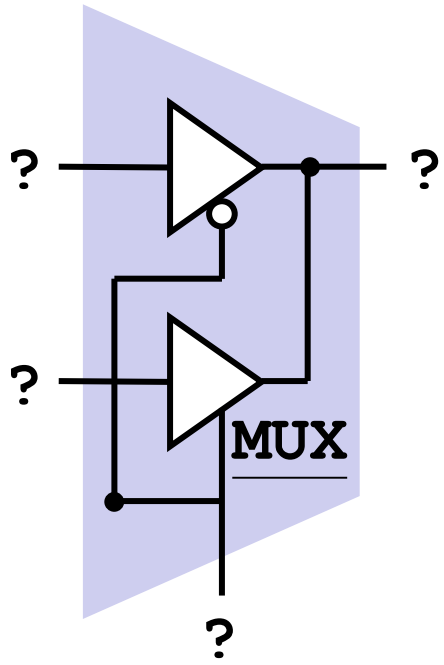
in 00	in 01	in 10	in 11	out 0	out 1
1	0	0	0	**0**	**0**
0	**1**	0	0	**0**	**1**
0	0	**1**	0	**1**	**0**
0	0	0	**1**	**1**	**1**

Outline

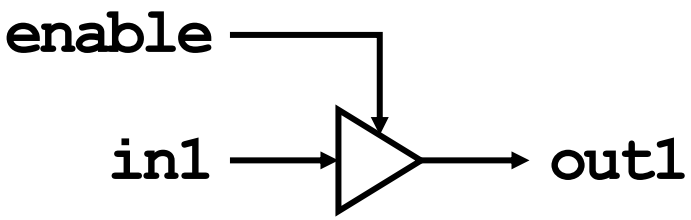
- Combinational Circuit and Sequential Circuit
  - Combinational Circuit: No Memory
    - Decoder
    - Multiplexer
    - Bi-directional Bus
  - Sequential Circuit: Has Memory
    - Latch
    - Flip-flop
      - Asynchronous Reset and Synchronous Reset
    - Finite State Machine (FSM)
      - Clock Edge Detection
      - Direct Feedback Path
      - Types and Examples

Combinational Circuit: Multiplexer

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity mux_ex is
port (in1,in2,sel: in std_logic;
                out1: out std_logic);
end mux_ex;
architecture mux_ex_arch of mux_ex is
begin
  process (in1, in2, sel)
  begin
    if sel = '0' then
      out1 <= in1; -- select in1
    else
      out1 <= in2; -- select in2
    end if;
  end process;
end mux_ex_arch;
```



Recall: Tri-state Buffer



in1	enable	out1
0	0	Z
1	0	Z
0	1	0
1	1	1

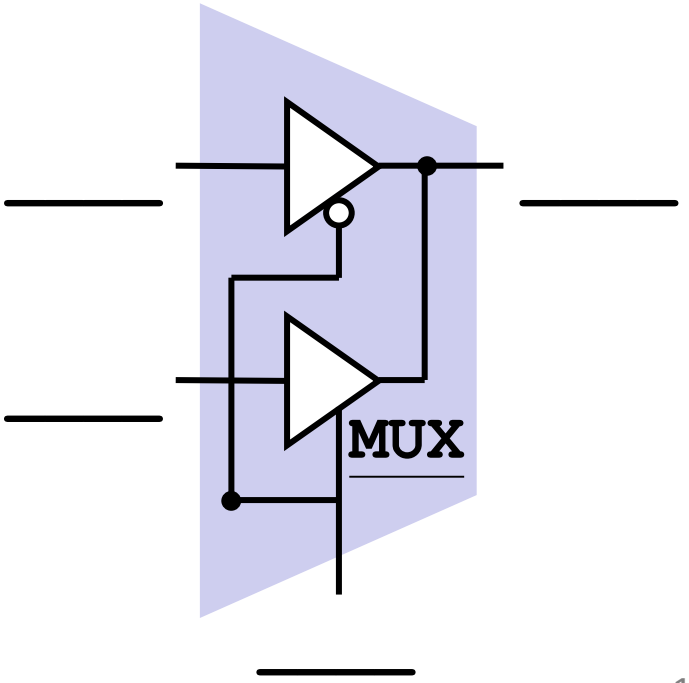
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity tri_ex is
port (in1, enable: in std_logic;
                out1: out std_logic);
end tri_ex;
architecture tri_ex_arch of tri_ex is
begin
    out1 <= in1 when enable = '1' else 'Z';
end tri_ex_arch;
```

Class Exercise 4.2

Student ID: _____ Date:
Name: _____ _____

- Specify the I/O signals in the circuit:

```
entity mux_ex is
port (in1,in2,sel: in std_logic;
                out1: out std_logic);
end mux_ex;
architecture mux_ex_arch of mux_ex is
begin
  process (in1, in2, sel)
  begin
    if sel = '0' then
      out1 <= in1;
    else
      out1 <= in2;
    end if;
  end process;
end mux_ex_arch;
```

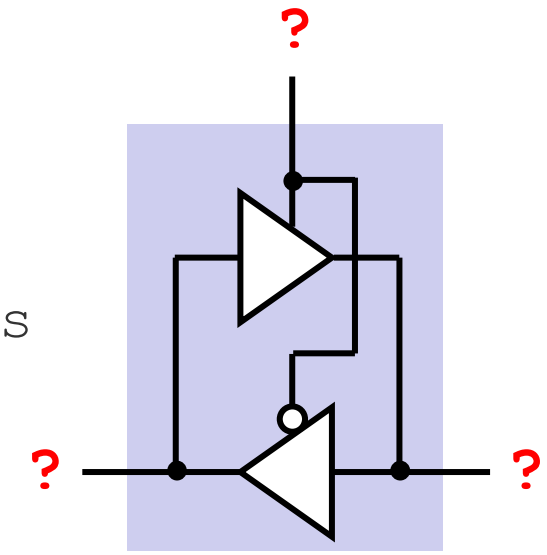


Outline

- Combinational Circuit and Sequential Circuit
  - Combinational Circuit: No Memory
    - Decoder
    - Multiplexer
    - Bi-directional Bus
  - Sequential Circuit: Has Memory
    - Latch
    - Flip-flop
      - Asynchronous Reset and Synchronous Reset
    - Finite State Machine (FSM)
      - Clock Edge Detection
      - Direct Feedback Path
      - Types and Examples

Combinational Circuit: Bi-directional Bus

```
entity inout_ex is
port (io1, io2: inout std_logic;
           ctrl: in std_logic);
end inout_ex;
architecture inout_ex_arch of inout_ex is
begin
  process (io1, io2, ctrl) is begin
    if (ctrl = '1') then io1 <= io2;
    else io1 <= 'Z';
    end if;
  end process;
  process (io1, io2, ctrl) is begin
    if (ctrl = '0') then io2 <= io1;
    else io2 <= 'Z';
    end if;
  end process;
end inout_ex_arch;
```



io1 follows "**io2.in**"

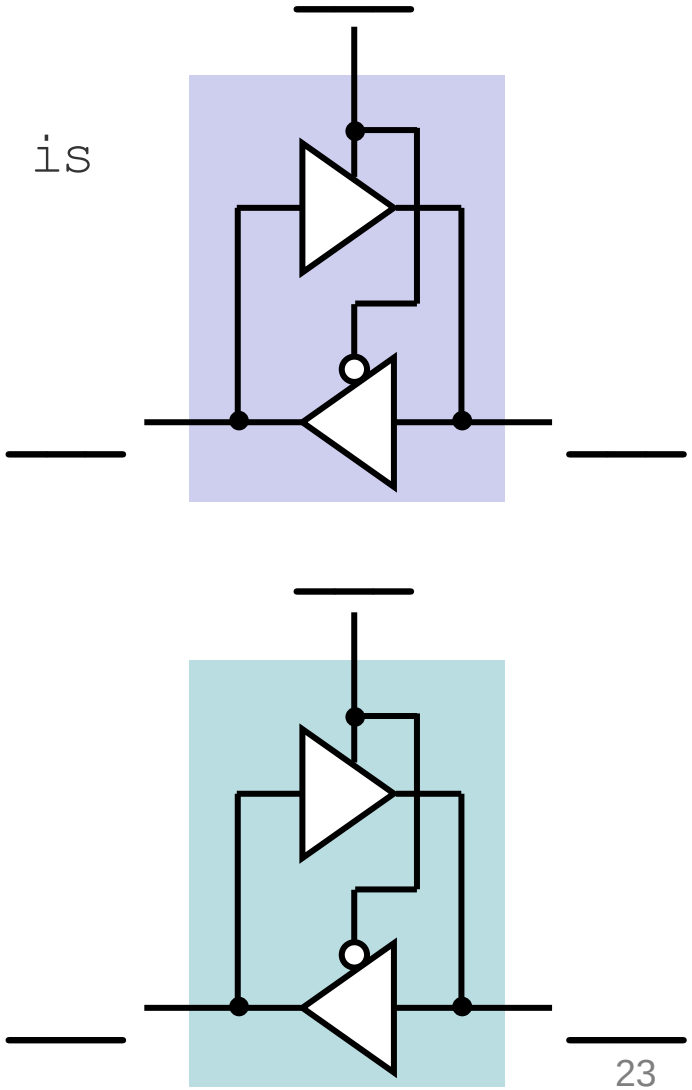
io2 follows "**io1.in**"

Class Exercise 4.3

Student ID: _____ Date:
Name: _____ _____

```
entity inout_ex is
port (io1, io2: inout std_logic;
           ctrl: in std_logic);
end inout_ex;
architecture inout_ex_arch of inout_ex is
begin
  process (io1, io2, ctrl) is begin
    if (ctrl = '1') then io1 <= io2;
    else io1 <= 'Z';
    end if;
  end process;
  process (io1, io2, ctrl) is begin
    if (ctrl = '0') then io2 <= io1;
    else io2 <= 'Z';
    end if;
  end process;
end inout_ex_arch;
```

- Specify I/O signals:

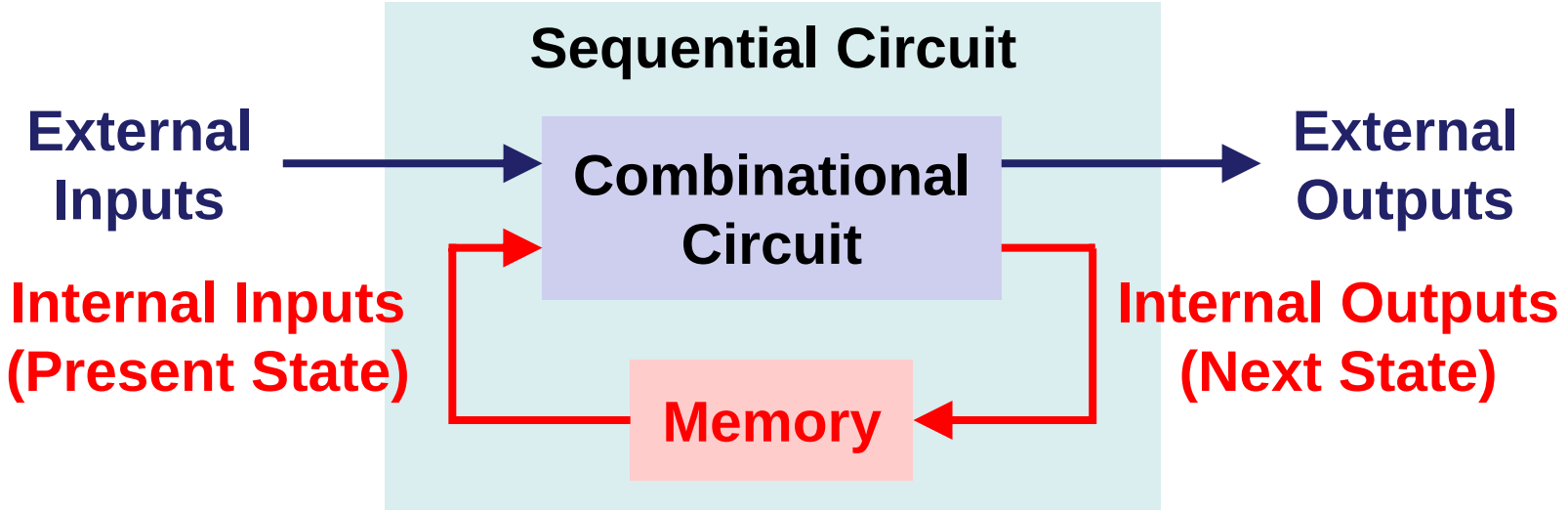


Outline

- Combinational Circuit and Sequential Circuit
  - Combinational Circuit: No Memory
    - Decoder
    - Multiplexer
    - Bi-directional Bus
  - Sequential Circuit: Has Memory
    - Latch
    - Flip-flop
      - Asynchronous Reset and Synchronous Reset
    - Finite State Machine (FSM)
      - Clock Edge Detection
      - Direct Feedback Path
      - Types and Examples

Recall: Sequential Circuit

- **Combinational Circuit: no memory**
  - ① Outputs are a function of the *present* inputs only.
  - ② **Rule**: Use **either** concurrent **or** sequential statements.
- **Sequential Circuit**: **has memory**
  - ① Outputs are a function of the *present* inputs and the *previous* outputs (i.e., the **internal state**).
  - ② **Rule**: **Must** use sequential statements (i.e., `process`) .



Latches and Flip Flops

- Latches and Flip-flops (FF) are the basic elements used to store information.
  - Each latch and flip flop can keep one bit of data.

- The main difference between latch and flip-flop:
  - A latch continuously checks input and changes the output **whenever** there is a change in input.
    - A latch has **no** clock signal.
  - A flip-flop continuously checks input and changes the output **only at times** determined by the clock signal.
    - A flip flop has a clock signal.

Outline

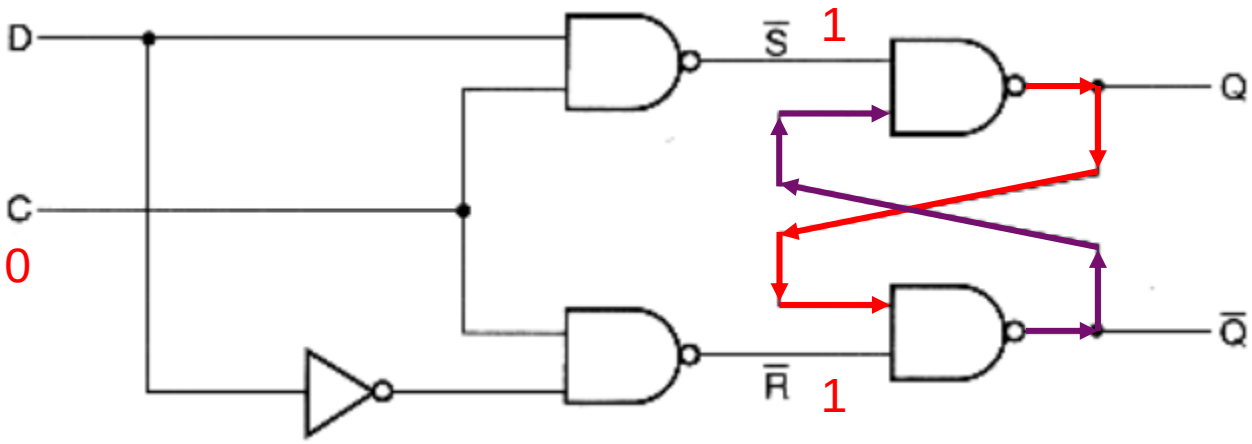
- Combinational Circuit and Sequential Circuit
  - Combinational Circuit: No Memory
    - Decoder
    - Multiplexer
    - Bi-directional Bus
  - Sequential Circuit: Has Memory
    - Latch
    - Flip-flop
      - Asynchronous Reset and Synchronous Reset
    - Finite State Machine (FSM)
      - Clock Edge Detection
      - Direct Feedback Path
      - Types and Examples

Sequential Circuit: Latch (1/2)

- Latches are asynchronous (**no** CLOCK signal).
  - It changes output **only in response** to input.
- **Case Study**: **D Latch**
  - When enable line C is high, the output Q follows input D.

  → That is why D latch is also called as transparent latch.
    - When enable line C is asserted, the latch is said to be transparent.
  - When C falls, the last state of D input is trapped and held.

  → That is why the latch has memory!



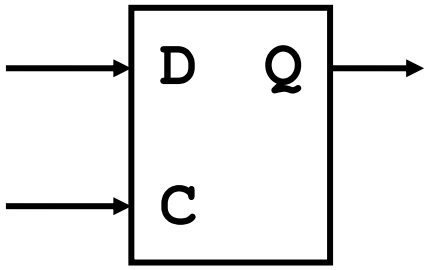
Data need to be held.

C	D	Next state of Q
0	X	No change
1	0	Q = 0; Reset state
1	1	Q = 1; Set state

https://www.edgefx.in/digital-electronics-latches-and-flip-flops/

Sequential Circuit: Latch (2/2)

```
 1 library IEEE;--(ok vivado 2014.4)
 2 use IEEE.STD_LOGIC_1164.ALL;
 3 entity latch_ex is
 4 port (C, D: in std_logic;
 5           Q: out std_logic);
 6 end latch_ex;
 7 architecture latch_ex_arch of latch_ex is
 8 begin
 9   process(C, D) -- sensitivity list
10   begin
11     if (C = '1') then
12       Q <= D;
13     end if;
       -- no change (memory)
14   end process;
15 end latch_ex_arch;
```



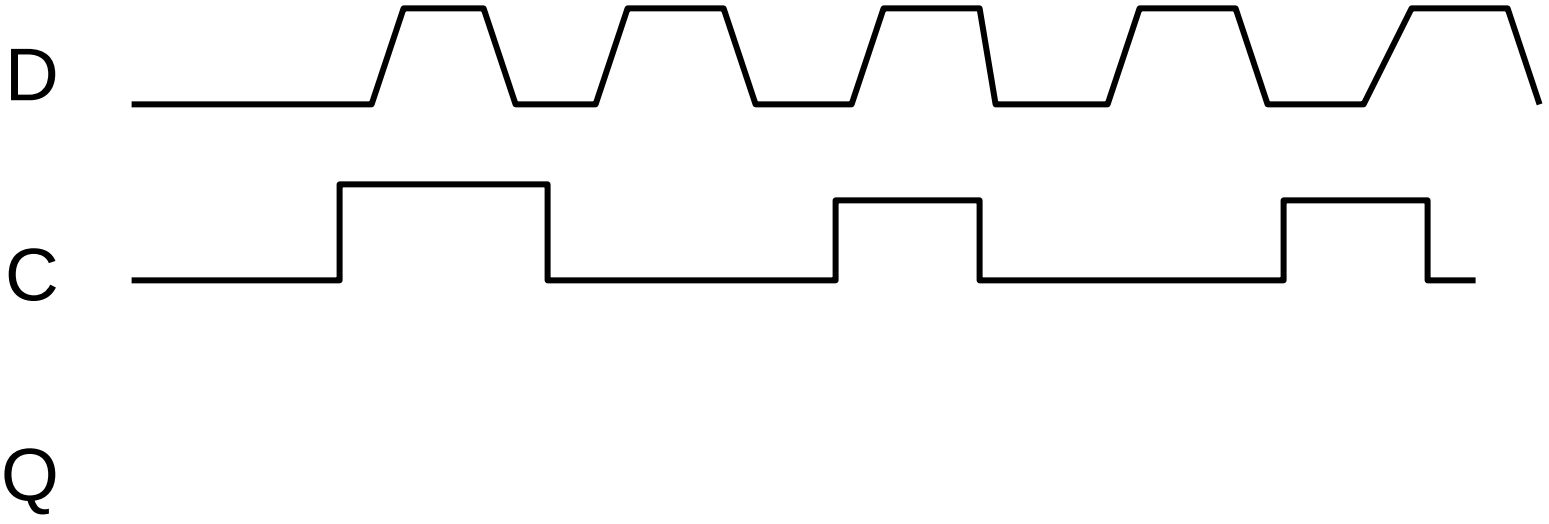
C	D	Next state of Q
0	X	No change
1	0	Q = 0; Reset state
1	1	Q = 1; Set state

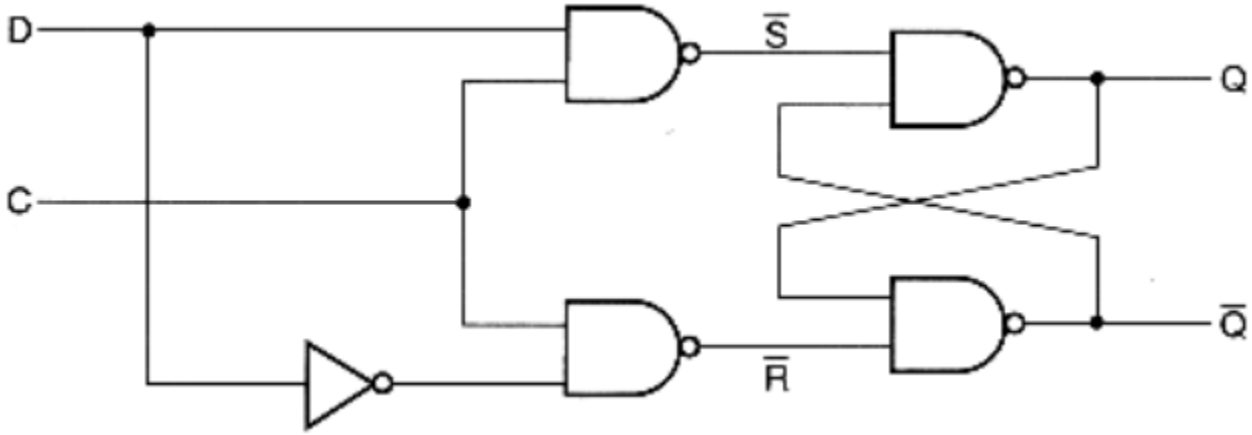
https://www.edgefx.in/digital-electronics-latches-and-flip-flops/

Class Exercise 4.4

Student ID: _____ Date:
Name: _____ _____

- Given a D latch, draw Q in the following figure:





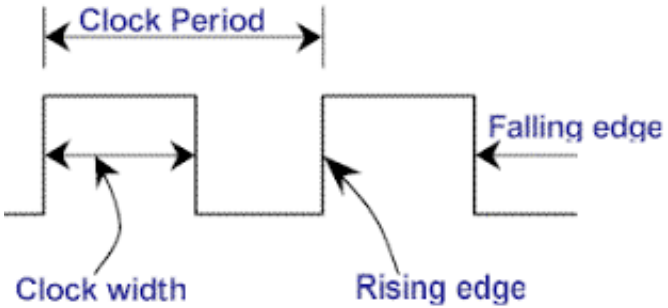
C	D	Next state of Q
0	X	No change
1	0	Q = 0; Reset state
1	1	Q = 1; Set state

Outline

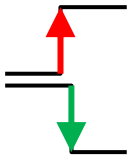
- Combinational Circuit and Sequential Circuit
  - Combinational Circuit: No Memory
    - Decoder
    - Multiplexer
    - Bi-directional Bus
  - Sequential Circuit: Has Memory
    - Latch
    - Flip-flop
      - Asynchronous Reset and Synchronous Reset
    - Finite State Machine (FSM)
      - Clock Edge Detection
      - Direct Feedback Path
      - Types and Examples

Sequential Circuit: Flip-flop

- A **Latch** is a non-clock-controlled memory device.
  ① It has **no** CLOCK signal.
  ② It changes output only in response to data input (i.e., the value is set asynchronously)**.**



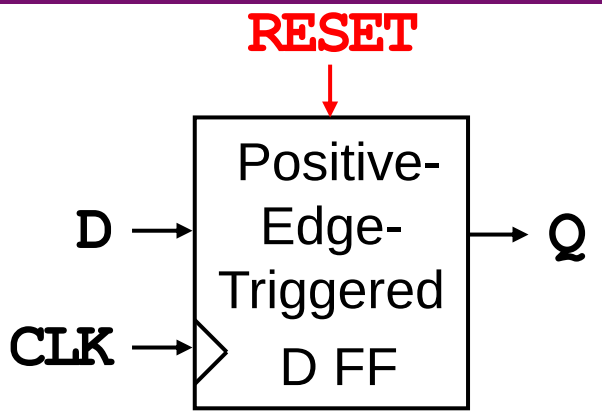
- A **Flip-flop (FF)** is a clock-controlled memory device.
  ① Different from a Latch, it **has** a CLOCK signal as input.
  ② It stores the input value (i.e., low or high) and **outputs** the stored value **only in response to** the CLOCK signal.
    - **Positive-Edge-Triggered**: At every **low to high** of CLOCK.
    - **Negative-Edge-Triggered**: At every **high to low** of CLOCK.



  ③ The value can be **reset** asynchronously or synchronously.
    - **Async. Reset**: Reset the value **anytime**.
    - **Sync. Reset**: Reset the value on **positive** or **negative clock edges**.

Positive-Edge-Triggered FF with Async. Reset

```
 1 library IEEE;
 2 use IEEE.STD_LOGIC_1164.ALL;
 3 entity dff_async is
 4 port(D,CLK,RESET: in std_logic;
 5                   Q: out std_logic);
 6 end dff_async;
 7 architecture dff_async_arch of dff_async is
 8 begin
 9   process(CLK, RESET) -- sensitivity list
10   begin
11      if (RESET = '1') then
12         Q <= '0'; -- Reset Q immediately
13      elsif CLK = '1' and CLK'event then
14         Q <= D; -- Q follows input D
15      end if;
        -- no change (so has memory)
16   end process;
17 end dff_async_arch;
```



← Positive-edge-triggered

Recall: Attributes (Lec01)

- Another important signal attribute is the **'event**.
  - This attribute yields a Boolean value of TRUE if an event has just occurred on the signal.
  - It is used primarily to determine if a **clock** has transitioned.
- Example (*more in Lec04*):

```
…
port(my_in, clock: in std_logic;
             my_out: out std_logic);
…
if clock = '1' and clock'event then
    my_out <= my_in;
```

Class Exercise 4.5

Student ID: _____ Date:
Name: _____ _____

- Consider the following VHDL implementation of a positive-edge-triggered FF with asynchronous reset:

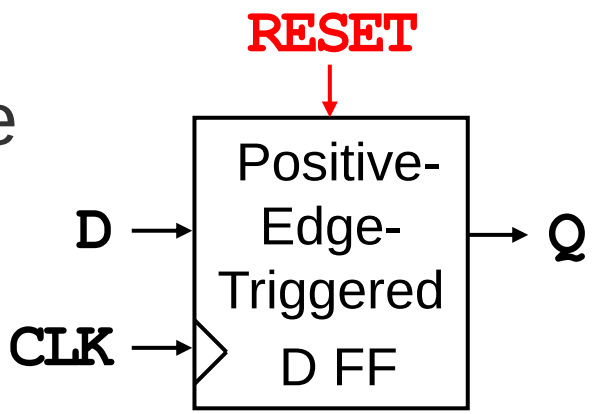
```
...
 9   process(CLK, RESET) -- sensitivity list
10   begin
11      if (RESET = '1') then
12         Q <= '0'; -- Reset Q
13      elsif CLK = '1' and CLK'event then
14         Q <= D; -- Q follows input D
15      end if;
        -- no change (so has memory)
16   end process;
...
```

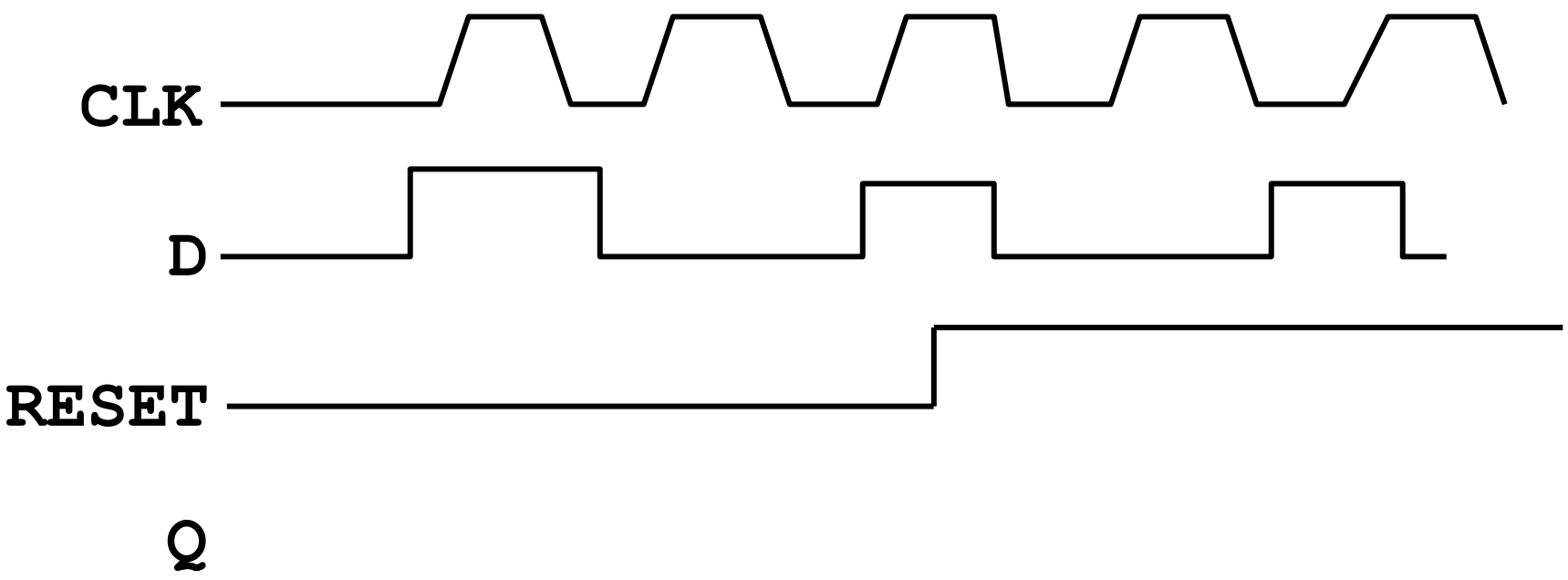
  - When will line 9 be executed?
    Answer: _____
  - Which signal is more "powerful"? **CLK** or **RESET**?
    Answer: _____

Class Exercise 4.6

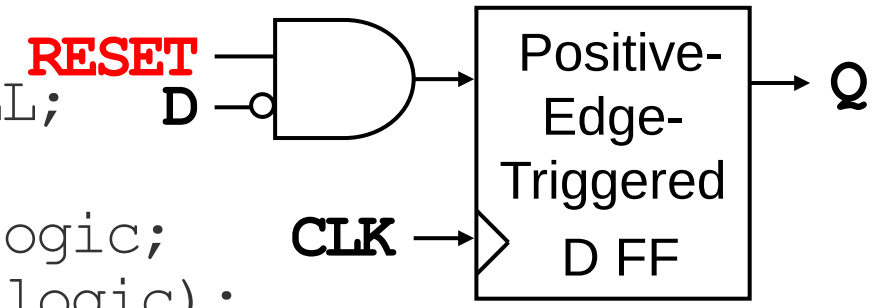
Student ID: _____ Date:
Name: _____ _____

- Given a Positive-edge-triggered D Flip-flop with **async.** reset, draw the output Q.





Positive-Edge-Triggered FF with Sync. Reset




```
 1 library IEEE;
 2 use IEEE.STD_LOGIC_1164.ALL;
 3 entity dff_sync is
 4 port(D,CLK,RESET: in std_logic;
 5                   Q: out std_logic);
 6 end dff_sync;
 7 architecture dff_sync_arch of dff_sync is begin
 8 process(CLK)
 9   begin
10     if CLK = '1' and CLK'event then
11       if (RESET = '1') then
12         Q <= '0'; -- Reset Q
13       else
14         Q <= D; -- Q follows input D
15       end if;
16     end if;
       -- no change (so has memory)
17   end process;
18 end dff_sync_arch;
```

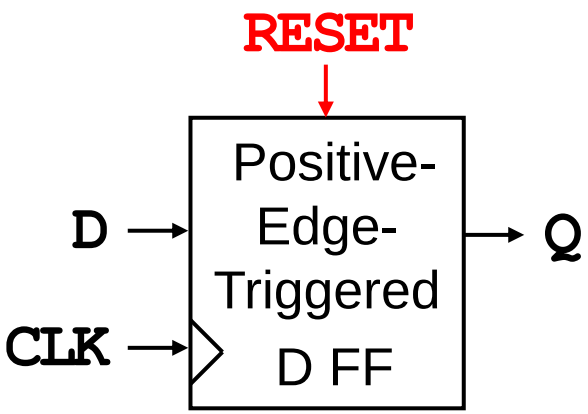
← **RESET** can be removed (why?)

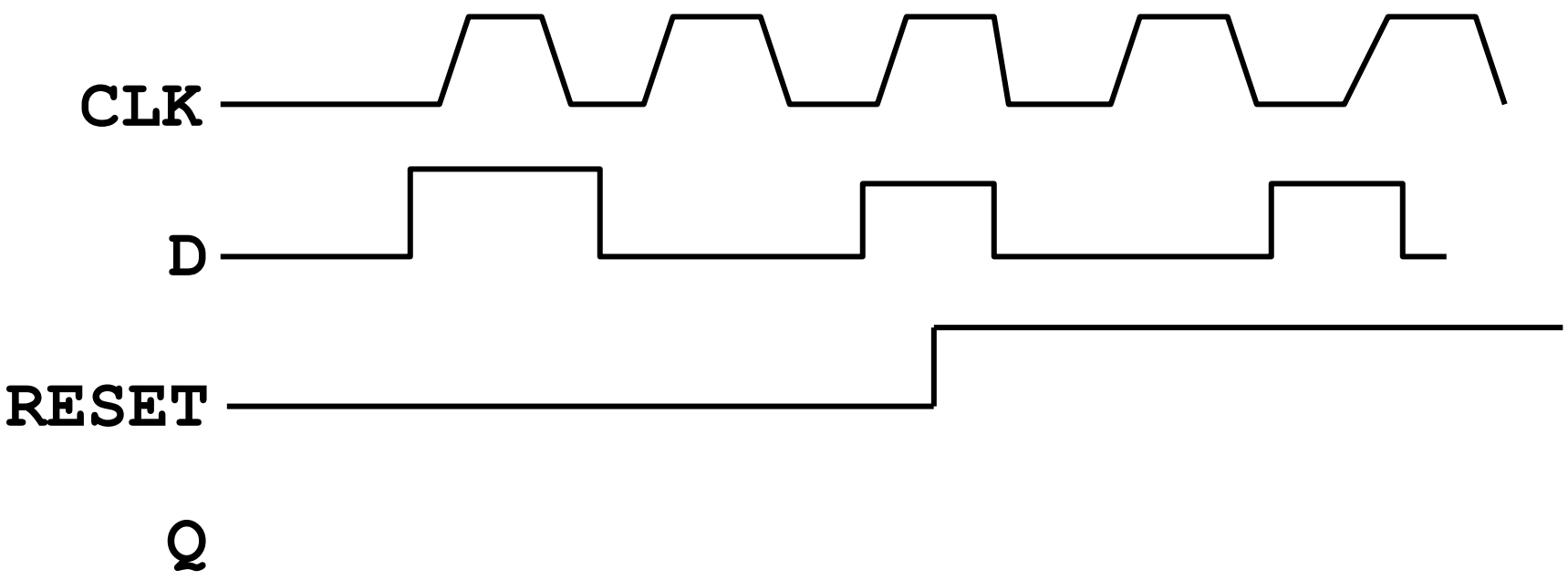
← Positive-edge-triggered

Class Exercise 4.7

Student ID: _____ Date:
Name: _____ _____

- Given a Positive-edge-triggered D Flip-flop with **sync.** reset, draw the output Q.





Async. Reset vs. Sync. Reset (1/2)

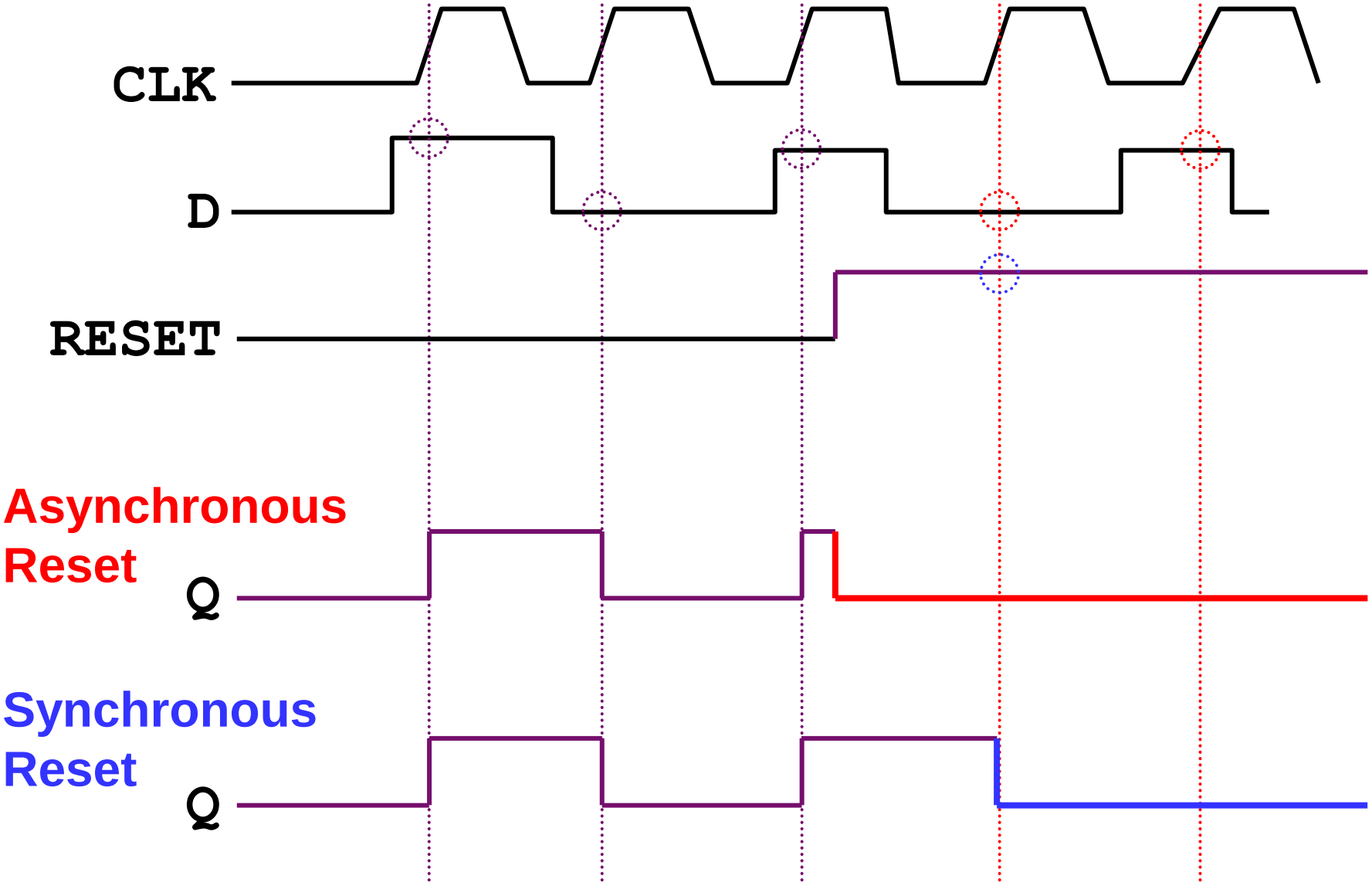
- The order of the statements inside the process determines asynchronous reset or synchronous reset.
  - **Asynchronous Reset** (check **RESET** first!)

```
11      if (RESET = '1') then
12         Q <= '0'; -- Reset Q
13      elsif CLK = '1' and CLK'event then
14         Q <= D; -- Q follows input D
15      end if;
```

  - **Synchronous Reset** (check **CLK** first!)

```
10     if CLK = '1' and CLK'event then
11       if (RESET = '1') then
12         Q <= '0'; -- Reset Q
13       else
14         Q <= D; -- Q follows input D
15       end if;
16     end if;
```

Async. Reset vs. Sync. Reset (2/2)

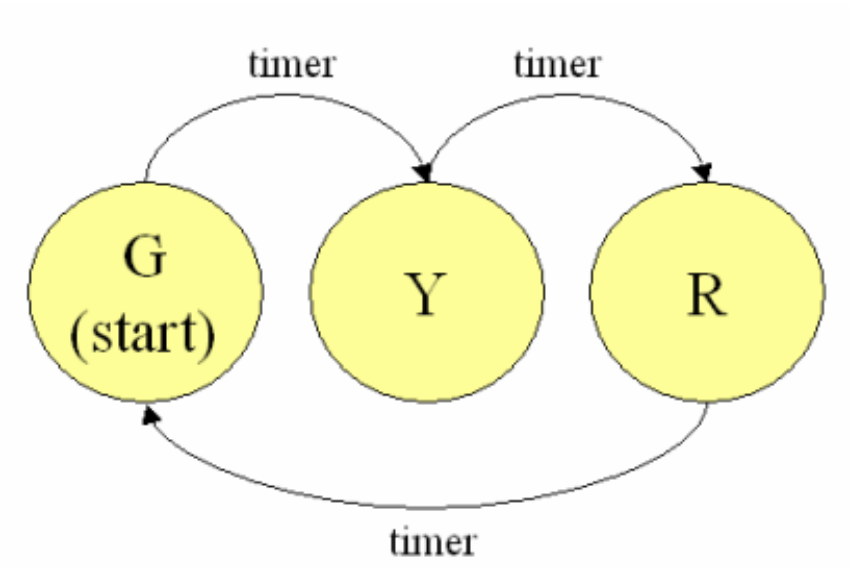


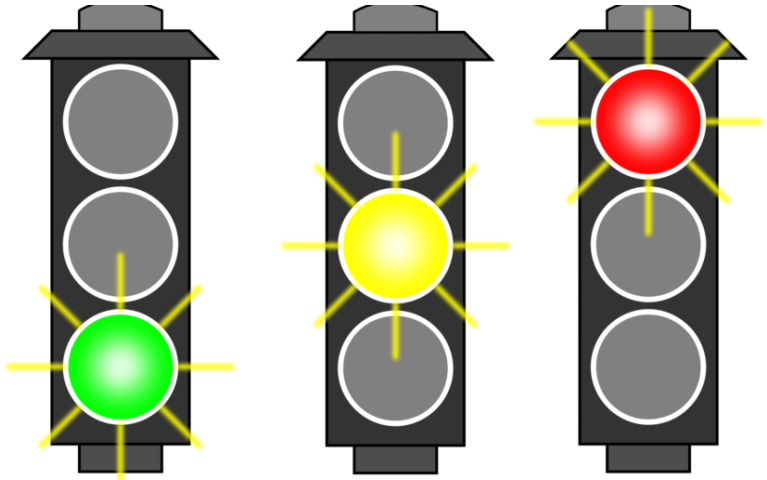
Outline

- Combinational Circuit and Sequential Circuit
  - Combinational Circuit: No Memory
    - Decoder
    - Multiplexer
    - Bi-directional Bus
  - Sequential Circuit: Has Memory
    - Latch
    - Flip-flop
      - Asynchronous Reset and Synchronous Reset
    - Finite State Machine (FSM)
      - Clock Edge Detection
      - Direct Feedback Path
      - Types and Examples

Finite State Machine (FSM)

- **Finite State Machine (FSM)**: A system jumps from one **state** to another:
  - Within a pool of finite states, and
  - Upon clock edges and/or input transitions.
- Example of FSM: traffic light, digital watch, CPU, etc.





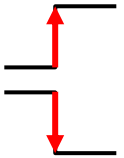
- Two crucial factors: *time (clock edge)* and *state (feedback)*

Outline

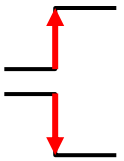
- Combinational Circuit and Sequential Circuit
  - Combinational Circuit: No Memory
    - Decoder
    - Multiplexer
    - Bi-directional Bus
  - Sequential Circuit: Has Memory
    - Latch
    - Flip-flop
      - Asynchronous Reset and Synchronous Reset
    - Finite State Machine (FSM)
      - Clock Edge Detection
      - Direct Feedback Path
      - Types and Examples

Clock Edge Detection

- Both "**wait until**" and "**if**" statements can be used to detect the clock edge (e.g., **CLK**):
- "**wait until**" **statement:**
  - `wait until CLK = '1'; -- rising edge`
  - `wait until CLK = '0'; -- falling edge`

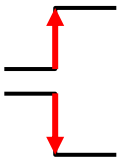


- "**if**" **statement:**
  - `if CLK'event and CLK = '1' -- rising edge`
  - `if CLK'event and CLK = '0' -- falling edge`



OR

  - `if( rising_edge(CLK) ) -- rising edge`
  - `if( falling_edge(CLK) ) -- falling edge`



rising_edge(CLK) vs. CLK'event

- **rising_edge()** function in std_logic_1164 library

```
FUNCTION rising_edge  (SIGNAL s : std_ulogic) RETURN BOOLEAN IS
BEGIN
    RETURN (s'EVENT AND (To_X01(s) = '1') AND
                        (To_X01(s'LAST_VALUE) = '0'));
END;
```

  - It results **TRUE** when there is an edge transition in the signal **s**, the present value is '**1**' and the last value is '**0**'.
  - If the last value is something like '**Z**' or '**U**', it returns a **FALSE**.

- The statement (**clk'event and clk='1'**)
  - It results **TRUE** when the there is an edge transition in the **clk** and the present value is '**1**'.
  - *It does not see whether the last value is '0' or not.*

Use rising_edge() / falling_edge() with "if" statements!

When to use "wait until" or "if"? (1/2)

- **Synchronous Process**: Computes values only on clock edges (i.e., only sensitive/sync. to clock signal).
  - **Rule:** Use "**wait-until**" or "**if**" for **synchronous** process:

Usage of "wait until"

```
process ← NO sensitivity list implies that there is one clock signal.
begin
   wait until clk='1';← The first statement must be wait until.
   …
end process
```

*Note: IEEE VHDL requires that a process with a wait statement must not have a sensitivity list, and the **first statement** must be **wait until**.*

Usage of "if"

```
process (clk)← The clock signal must be in the sensitivity list.
begin
   …
   if( rising_edge(clk) )← NOT necessary to be the first line.
   …
end process
```

When to use "wait until" or "if"? (2/2)

- **Asynchronous Process**: Computes values on clock edges or when asynchronous conditions are TRUE.
  - That is, it must be sensitive to the clock signal (if any), and to all inputs that may affect the asynchronous behavior.
  - **Rule:** Only use "**if**" for **asynchronous** process:

Usage of "if"

```
process (clk, input_a, input_b, …)
begin
   …
   if( rising_edge(clk) )
   …
end process
```

← The sensitivity list should include the clock signal, and all inputs that may affect asynchronous behavior.

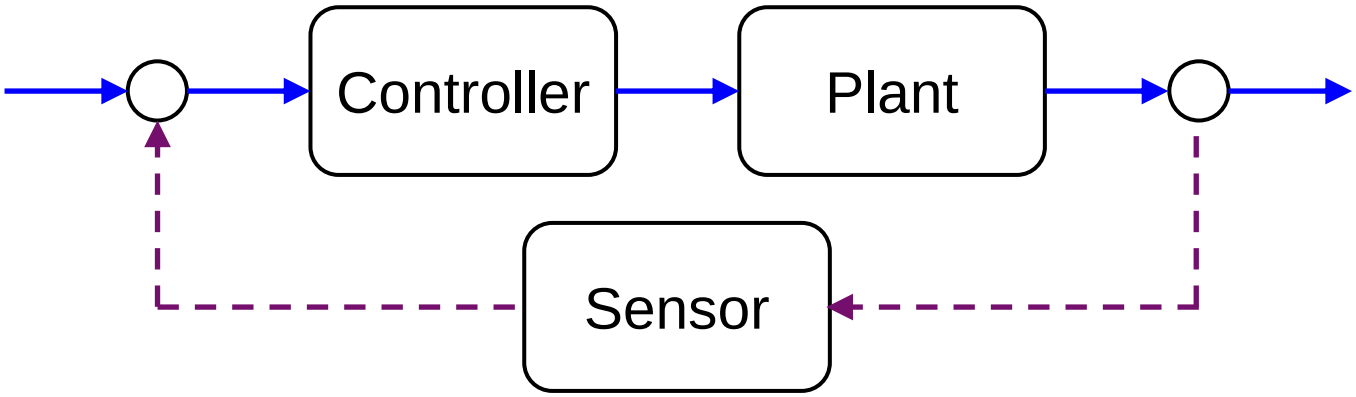
Simply use "if" statements for both sync. and async. processes!

Outline

- Combinational Circuit and Sequential Circuit
  - Combinational Circuit: No Memory
    - Decoder
    - Multiplexer
    - Bi-directional Bus
  - Sequential Circuit: Has Memory
    - Latch
    - Flip-flop
      - Asynchronous Reset and Synchronous Reset
    - Finite State Machine (FSM)
      - Clock Edge Detection
      - Direct Feedback Path
      - Types and Examples

Feed-forward and Feedback Paths

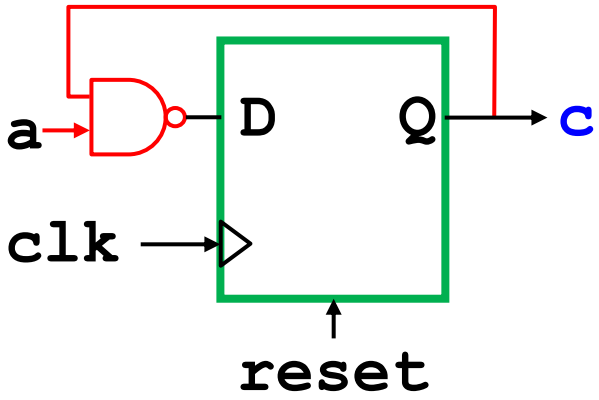
- So far, we mostly focus on logic with feed-forward (or open-loop) paths.



- Now, we are going to learn feedback (or closed-loop) paths—*the key step of making a finite state machine*.

Direct Feedback Path

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity feedback_1 is
port(a,clk,reset: in std_logic;
     c: buffer std_logic);
end feedback_1;
architecture feedback_1_arch of feedback_1 is
begin
  process(clk, reset) -- async.
  begin
    if reset = '1' then c <= '0';
    elsif rising_edge(clk) then
      c <= not(a and c);
    end if;
  end process;
end feedback_1_arch ;
```



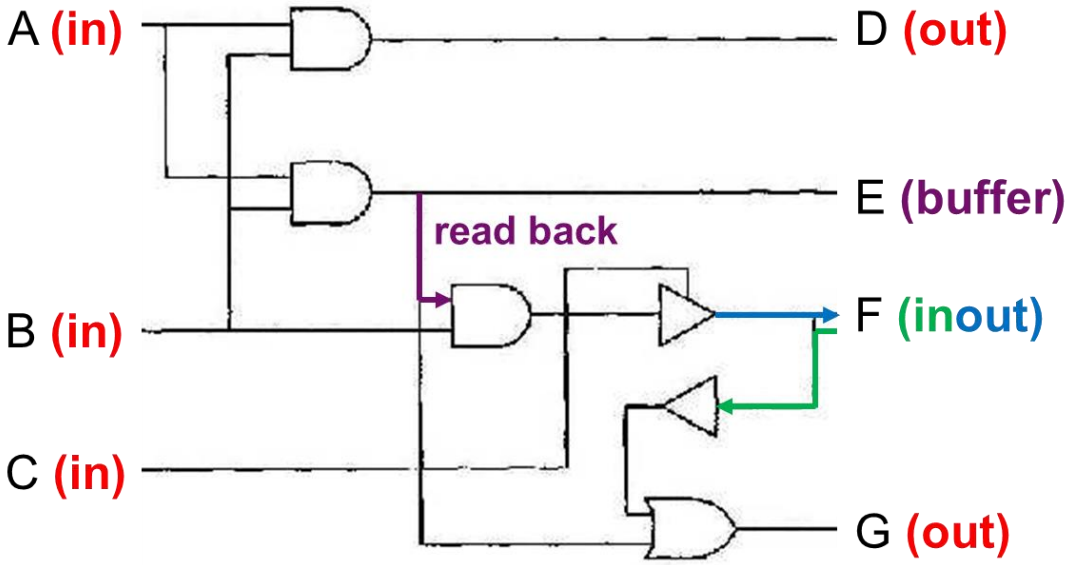
① Signal **c** forms a **closed loop**.

- **not(a and c)** takes effect at the next rising clock edge.
- The current **c** holds for one cycle.

② "**<=**" is like a flip-flop.

Internal Feedback: inout or buffer

- Recall (*Lec01*): There are 4 modes of I/O pins:
  1) **in**: Data flows **in** only
  2) **out**: Data flows **out** only (cannot be read back by the entity)
  3) inout: Data flows **bi-directionally** (i.e., in or out)
  4) buffer: Similar to **out** but it can be **read back** by the entity



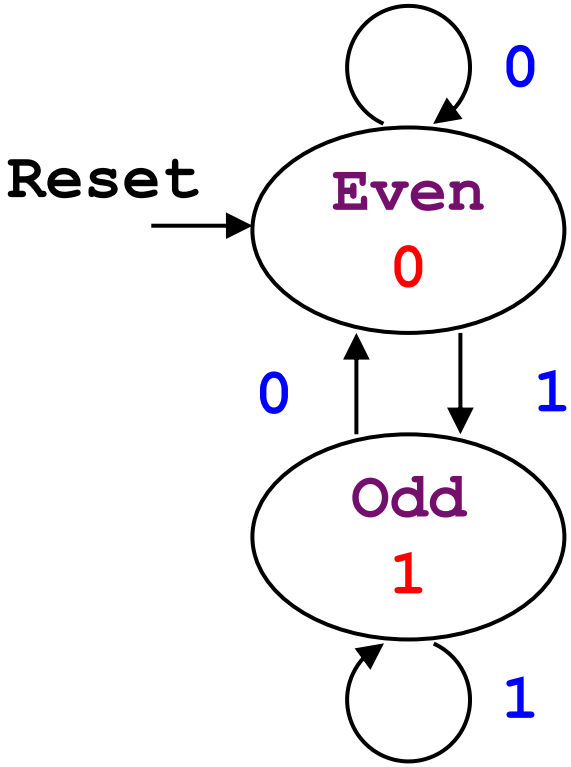
- Both buffer and inout can be read back internally.
  - inout can also read external input signals.

Outline

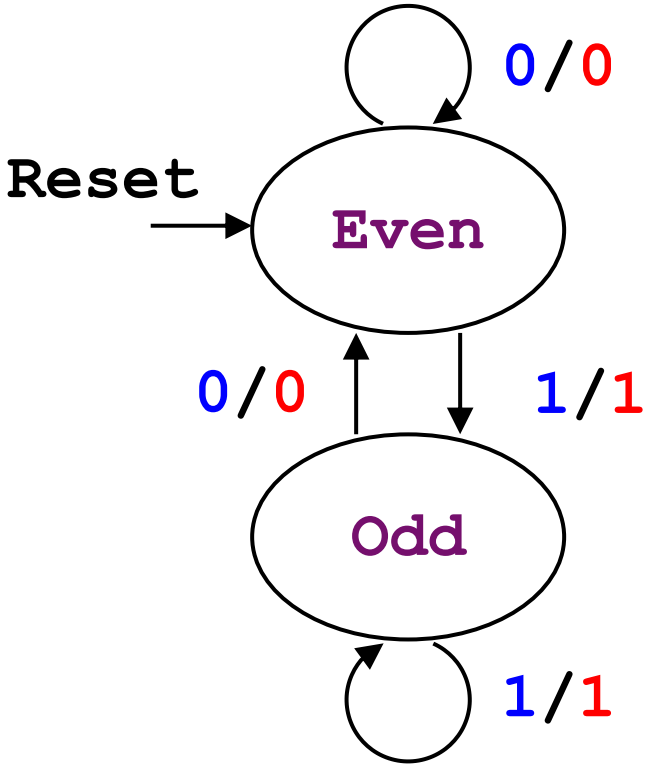
- Combinational Circuit and Sequential Circuit
  - Combinational Circuit: No Memory
    - Decoder
    - Multiplexer
    - Bi-directional Bus
  - Sequential Circuit: Has Memory
    - Latch
    - Flip-flop
      - Asynchronous Reset and Synchronous Reset
    - Finite State Machine (FSM)
      - Clock Edge Detection
      - Direct Feedback Path
      - Types and Examples

Types of Finite State Machines

- **Moore Machine**:
  - Outputs are a function of the present state *only*.
- **Mealy Machine**:
  - Outputs are a function of the present state *and* the present inputs.



State / input / output



Suggestion: Maintain the internal state explicitly!

Moore Machine

- **Moore Machine**: *outputs rely on present state only.*

```
architecture moore_arch of fsm is
signal s: bit; -- internal state
begin
  process (s)                                   Combinational Logic
  begin
    OUTX <= not s; -- output
  end process;
  process (CLOCK, RESET)                        Sequential Logic
  begin
    if RESET = '1' then s <= '0';
    elsif rising_edge(CLOCK) then
      s <= not (INX and s); -- feedback
    end if;
  end process;
end moore_arch;
```

Mealy Machine

- **Mealy Machine**: *outputs depend on state and inputs.*

```
architecture mealy_arch of fsm is
signal s: bit; -- internal state
begin
  process (INX, s)                    Combinational Logic
  begin
    OUTX <= (INX or s); -- output
  end process;
  process (CLOCK, RESET)              Sequential Logic
  begin
    if RESET = '1' then s <= '0';
    elsif rising_edge(CLOCK) then
      s <= not (INX and s); -- feedback
    end if;
  end process;
end mealy_arch;
```

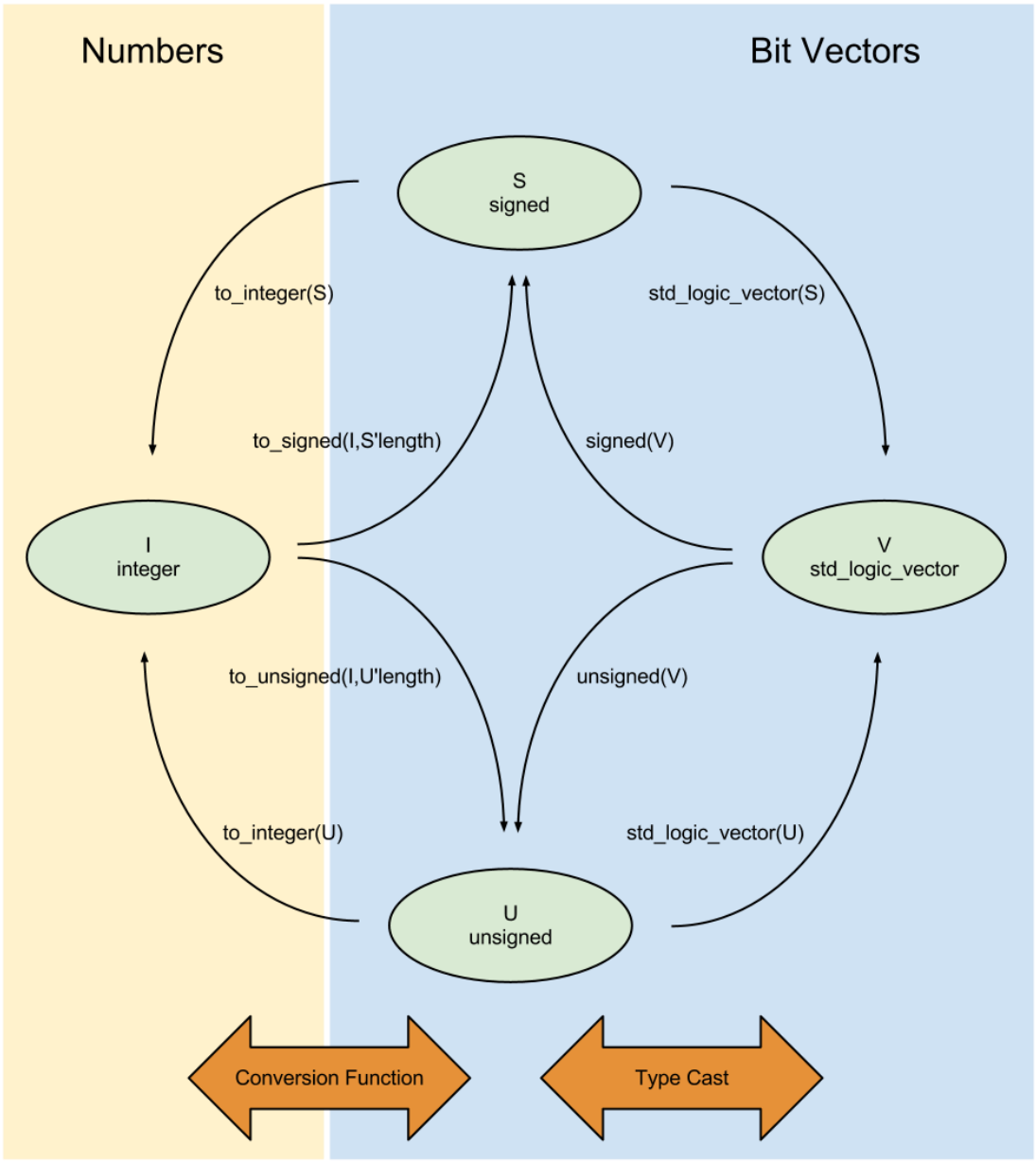
FSM Example 1) Up/Down Counter

- **Up/Down Counters**: Generate a sequence of counting patterns according to the clock and inputs.

```
use IEEE.Numeric_Std.ALL;
entity counter is
  port(CLK: in std_logic;
       RESET: in std_logic;
       COUNT: out std_logic_vector(3 downto 0));
end counter;
architecture counter_arch of counter is
signal s: std_logic_vector(3 downto 0); -- internal state
begin
  COUNT <= s; -- output                          Combinational Logic
  process(CLK, RESET)                            Sequential Logic
  begin
    if(RESET = '1') then s <= "0000";
    else
      if( rising_edge(CLK) ) then
        s <= std_logic_vector(unsigned(s) + 1); -- feedback
      end if;
    end if;
  end process;
end counter_arch;
```

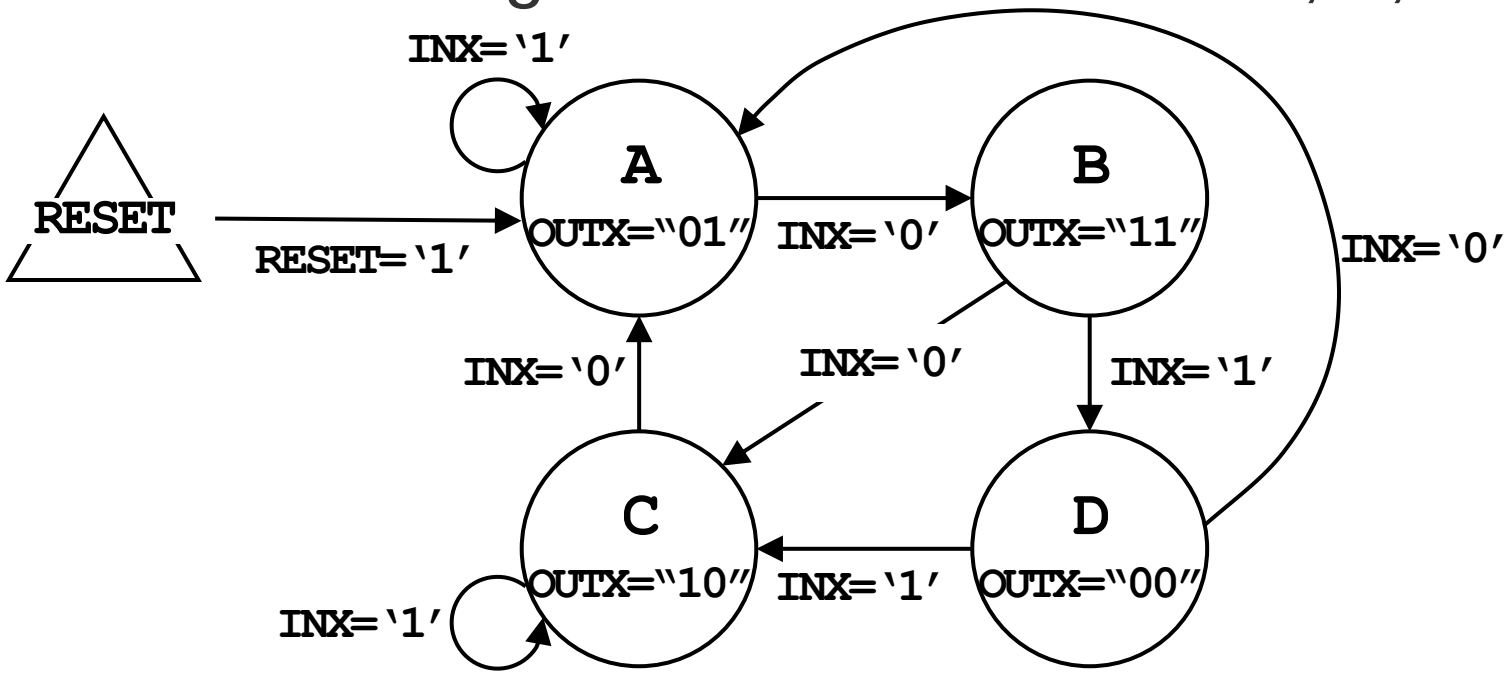
FSM Example 1) Up/Down Counter (2/2)

- VHDL is strongly-typed language.
  - Data objects of different base types CANNOT to assigned to each other without using type-casting or type-conversion.
    - **Type-casting:** Move between `std_logic_vector` and `signed`/`unsigned`.
    - **Type-conversion**: Move between `signed`/`unsigned` and `integer`.



FSM Example 2) Pattern Generator (1/3)

- **Pattern Generator**: Generates any pattern we want.
  - Example: the control unit of a CPU, traffic light, etc.
- Given the following machine of 4 states: **A**, **B**, **C** and **D**.



  - The machine has an asynchronous **RESET**, a clock signal **CLK** and a 1-bit synchronous input signal **INX**.
  - The machine also has a 2-bit output signal **OUTX**.

FSM Example 2) Pattern Generator (2/3)

```
library IEEE;
use IEEE.std_logic_1164.all;
entity pat_gen is port(
RESET,CLOCK,INX: in STD_LOGIC;
OUTX: out STD_LOGIC_VECTOR(1 downto 0));
end pat_gen;
architecture arch of pat_gen is
type state_type is (A,B,C,D);
signal s: state_type; -- state
begin
process(CLOCK, RESET)
begin
  if RESET = '1' then
    s <= A;
  elsif rising_edge(CLOCK) then
    -- feedback
    case s is
    when A =>
      if INX = '1' then s <= A;
      else s <= B; end if;
    when B =>
      if INX = '1' then s <= D;
      else s <= C; end if;
    when C =>
      if INX = '1' then s <= C;
      else s <= A; end if;
    when D =>
      if INX = '1' then s <= C;
      else s <= A; end if;
    end case;
  end if;
end process;
```

Sequential Logic

```
process(s)
begin
  case s is
    when A => OUTX <= "01";
    when B => OUTX <= "11";
    when C => OUTX <= "10";
    when D => OUTX <= "00";
  end case;
end process;
end arch;
```

Combinational Logic

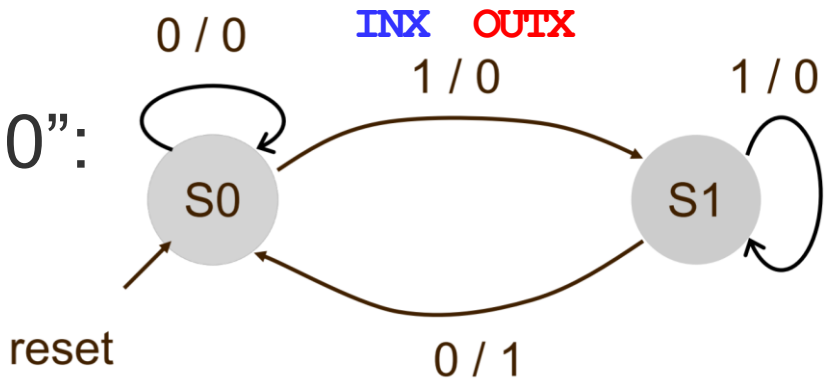
FSM Example 2) Pattern Generator (3/3)

- Encoding methods for representing patterns/states:
  - **Binary Encoding**: Using N flip-flops to represent 2^N states.
    - Less flip-flops but more combinational logics
  - **One-hot Encoding**: Using N flip-flops for **N** states.
    - More flip-flops but less combination logic
  - *Xilinx default seeting is one-hot encoding.*
    - *Change at* ***`synthesis → options`***
    - *http://www.xilinx.com/itp/xilinx4/data/docs/sim/vtex9.html*

Class Exercise 4.8

Student ID: _____ Date:
Name: _____ _____

- Complete a Mealy Machine that recognizes sequence “10”:




```
architecture arch of mealy_fsm is
type state_type is (S0, S1);
signal s: std_logic; -- state
begin
process(CLK, RESET) -- seq
begin
  if(RESET = '1') then s <= __;
  else
    if( rising_edge(CLK) ) then
      case s is
      when S0 =>
        if input = __ then
          s <= __; -- feedback
        else
          s <= __; -- feedback
        end if;
      when S1 =>
        if input = __ then
          s <= __; -- feedback
        else
          s <= __; -- feedback
        end if;
      end case;
    end if;
  end if;
end process;
OUTX <= __ when (s=__ and INX=__)
        else __; -- output
end arch;
```

Rule of Thumb: VHDL Coding Tips

① **Maintain the internal state(s) explicitly**

② **Separate combinational and sequential logics**

- Write **at least two processes**: one for combinational logic, and the other for sequential logic
  - Maintain the internal state(s) using a sequential process
  - Drive the output(s) using a combination process

③ **Keep every process as simple as possible**

- Partition a large process into **multiple small ones**

④ **Put every signal** (that your process must be sensitive to its changes) **in the sensitivity list**.

⑤ **Avoid assigning a signal from multi-processes**

- It may cause the "**multi-driven**" issue.

Summary

- Combinational Circuit and Sequential Circuit
  - Combinational Circuit: No Memory
    - Decoder
    - Multiplexer
    - Bi-directional Bus
  - Sequential Circuit: Has Memory
    - Latch
    - Flip-flop
      - Asynchronous Reset and Synchronous Reset
    - Finite State Machine (FSM)
      - Clock Edge Detection
      - Direct Feedback Path
      - Types and Examples