# EE-421: Digital System Design

## Synchronous vs Asynchronous Reset

Dr. Rehan Ahmed [rehan.ahmed@seecs.edu.pk]

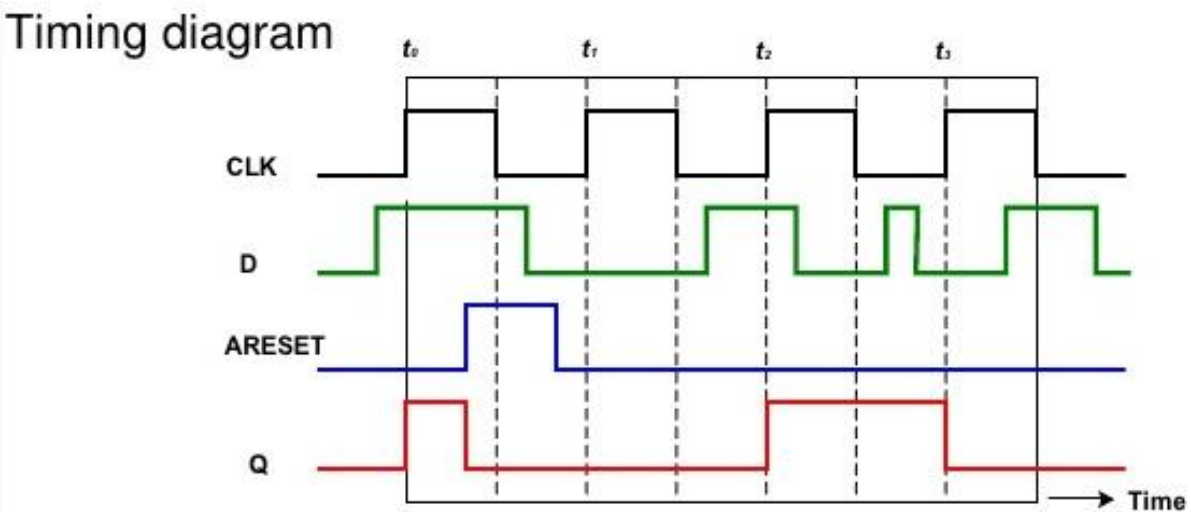
NUST

SCHOOL OF ELECTRICAL ENGINEERING &
COMPUTER SCIENCE (SEECS)

# Asynchronous and Synchronous Reset

- Reset signals are used to initialize the hardware to a known state
  - Usually activated at system start (on power up)

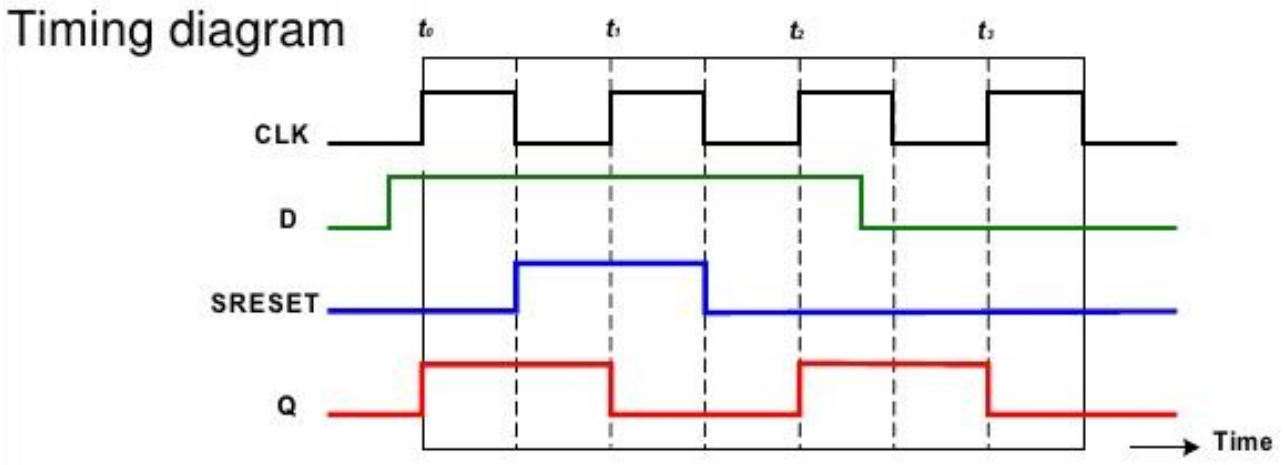
- **Asynchronous Reset**
  - The reset signal is sampled independent of the clock
  - Reset gets the highest priority
  - Sensitive to glitches, may have metastability issues
    - Will be discussed later in the course ….



# Asynchronous and Synchronous Reset

- Reset signals are used to initialize the hardware to a known state
  - Usually activated at system start (on power up)

- **Synchronous Reset**
  - The reset signal is sampled with respect to the clock
  - The reset should be active long enough to get sampled at the clock edge
  - Results in completely synchronous circuit



# D Flip-Flop with Asynchronous Reset

```
module flop_ar (input             clk,
                input             reset,
                input      [3:0] d,
                output reg [3:0] q);

  always @ (posedge clk, negedge reset)
    begin
       if (reset == 0) q <= 0;   // when reset
       else            q <= d;   // when clk
    end
endmodule
```

- In this example: two events can trigger the process:
  - A ***rising edge*** on clk
  - A ***falling edge*** on reset

# D Flip-Flop with Asynchronous Reset

```
module flop_ar (input              clk,
                input              reset,
                input      [3:0] d,
                output reg [3:0] q);

  always @ (posedge clk, negedge reset)
    begin
      if (reset == 0) q <= 0;   // when reset
      else            q <= d;   // when clk
    end
endmodule
```

- For longer statements, a begin-end pair can be used
  - To improve readability
  - In this example, it was not necessary, but it is a good idea

# D Flip-Flop with Asynchronous Reset

```
module flop_ar (input             clk,
                input             reset,
                input      [3:0] d,
                output reg [3:0] q);

  always @ (posedge clk, negedge reset)
    begin
      if (reset == 0) q <= 0;    // when reset
      else            q <= d;    // when clk
    end
endmodule
```

- First reset is checked: if reset is 0, q is set to 0.
  - This is an asynchronous reset as the reset can happen independently of the clock (on the negative edge of reset signal)
- If there is no reset, then regular assignment takes effect

# D Flip-Flop with **Synchronous** Reset

```
module flop_sr (input             clk,
                input             reset,
                input      [3:0] d,
                output reg [3:0] q);

  always @ (posedge clk)
    begin
       if (reset == '0') q <= 0;   // when reset
       else              q <= d;   // when clk
    end
endmodule
```

- The process is sensitive to only clock
  - Reset ***happens only*** when the ***clock rises***. This is a synchronous reset

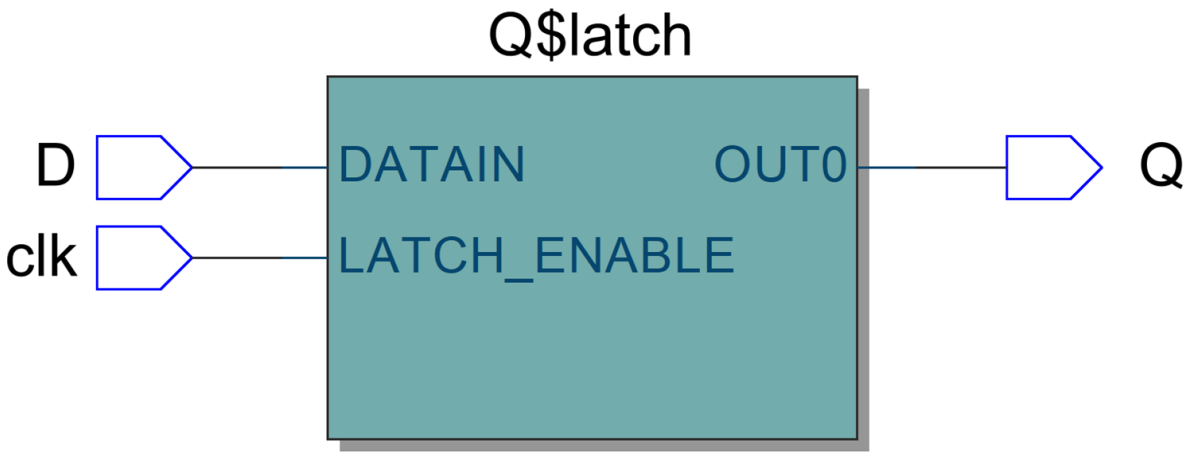
# D Flip-Flop with Enable and Reset

```
module flop_en_ar (input             clk,
                   input             reset,
                   input             en,
                   input      [3:0] d,
                   output reg [3:0] q);

  always @ (posedge clk, negedge reset)
    begin
      if (reset == '0') q <= 0;   // when reset
      else if (en)      q <= d;   // when en AND clk
    end
endmodule
```

- A flip-flop with **enable** and **reset**
  - Note that the en signal is ***not*** in the *sensitivity list*
- q gets d only when clk is rising **and** en is 1

# Example: D Latch

```
module latch (input            clk,
              input      [3:0] d,
              output reg [3:0] q);

  always @ (clk, d)
    if (clk) q <= d;       // latch is transparent when
                           // clock is 1
endmodule
```

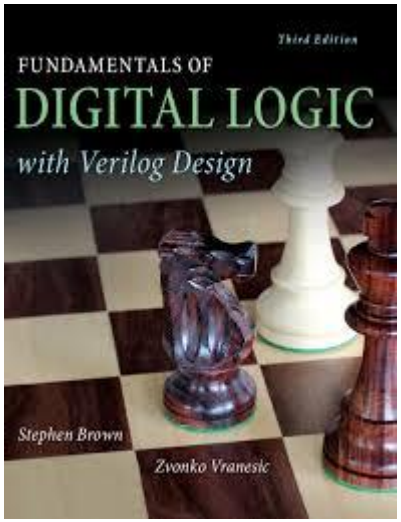


# Summary: Sequential Statements So Far

- Sequential statements are within an `always` block
- The sequential block is triggered with a change in the sensitivity list
- Signals assigned within an **always** must be declared as `reg`
- We use `<=` for (non-blocking) assignments and do not use `assign` within the always block.

# Recommended Reading

- Digital System Design with Verilog HDL, 3/e, b **S**tephen Brown and **Z**vonko Vranesic. [**S&Z**]
  - S&Z,
    - Chapter-5 for review
      - 5.1, 5.3, 5.8
    - 5.13.1 for Verilog



# THANK YOU