Digital Design Group Project
EENG 28010

Sequential Logic: Latch vs Flip-flop

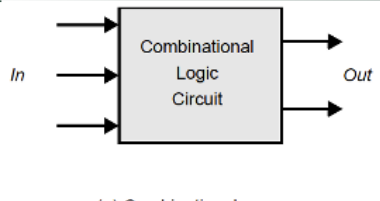
Dr. Faezeh Arab Hassani
Department of Electrical and Electronic Engineering

University of BRISTOL

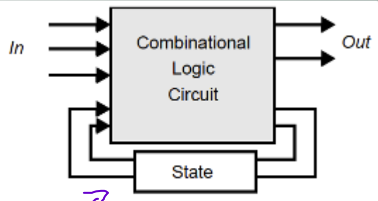
Outline

- ❑ Combinational versus Sequential Logic
- ❑ D-Latch vs D-Flip-flop
- ❑ Inferring Combinational or Sequential Logic

Combinational versus Sequential Logic



(a) Combinational



(b) Sequential

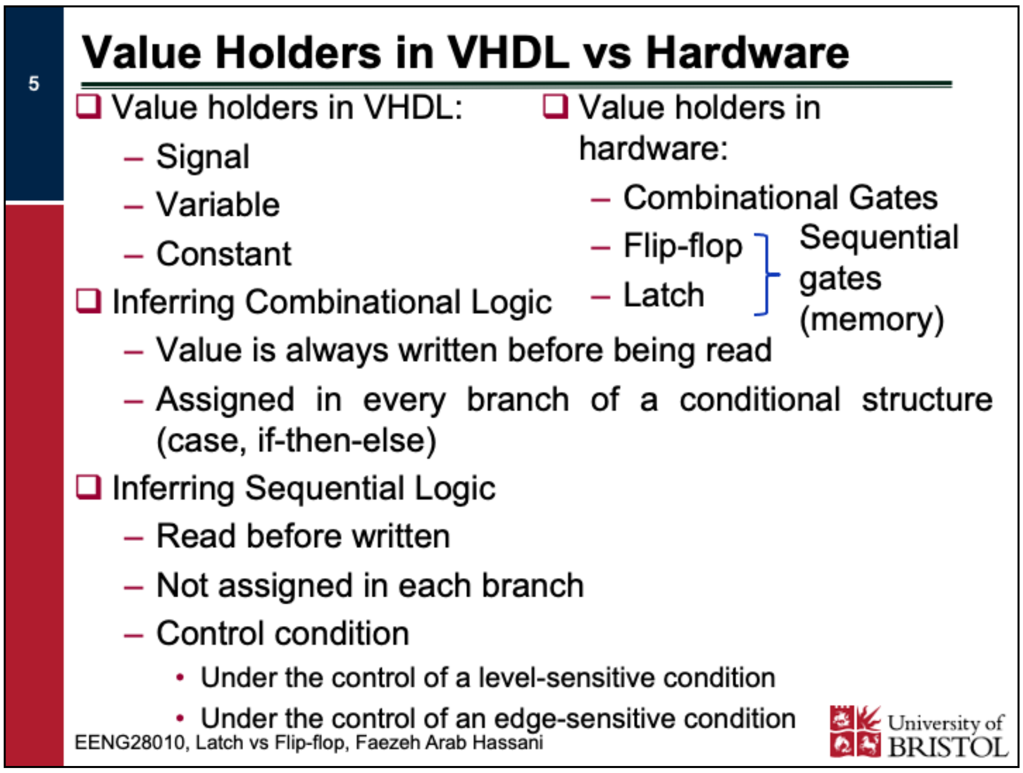
- At any point in time, output is related to the current input signals
- No intentional connection between outputs and inputs

- Output is not only a function of the current input data, but also of previous values
- Includes a combinational logic portion and a module that holds the state
- Example circuits are registers, counters, oscillators, and memory

University of BRISTOL

Outline

- ❑ Combinational versus Sequential Logic
- ❑ D-Latch vs D-Flip-flop
- ❑ Inferring Combinational or Sequential Logic



Value Holders in VHDL vs Hardware

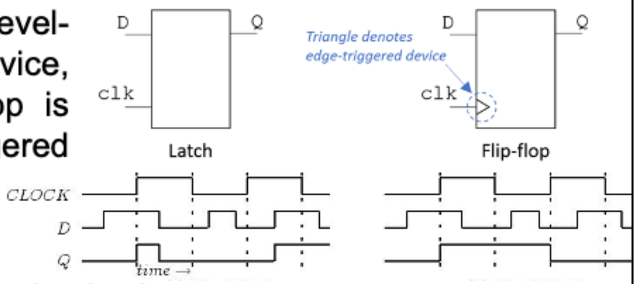
- Value holders in VHDL:
  - Signal
  - Variable
  - Constant
- Value holders in hardware:
  - Combinational Gates
  - Flip-flop
  - Latch

  Sequential gates (memory)
- Inferring Combinational Logic
  - Value is always written before being read
  - Assigned in every branch of a conditional structure (case, if-then-else)
- Inferring Sequential Logic
  - Read before written
  - Not assigned in each branch
  - Control condition
    - Under the control of a level-sensitive condition
    - Under the control of an edge-sensitive condition

University of BRISTOL

D-Latch vs D-Flip-flop

- A latch is a level-sensitive device, whereas a flip-flop is an edge-triggered device.



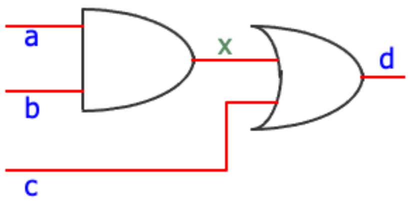
- Disadvantages of using latches:
  - Timing problems
  - Most of the time they are inferred unintentionally
  - It's possible for a latch-based circuit to simulate correctly, but not work in real hardware, because the timing delays on the real hardware don't match those predicted in synthesis

Outline

- ❑ Combinational versus Sequential Logic
- ❑ D-Latch vs D-Flip-flop
- ❑ Inferring Combinational or Sequential Logic

Inferring Combinational Logic – Eg:

```
SIGNAL a, b, c, d: BIT;
---
PROCESS(a, b, c)
   VARIABLE x: BIT;
BEGIN
   x := a AND b;
   d <= x OR c;
END PROCESS;
```

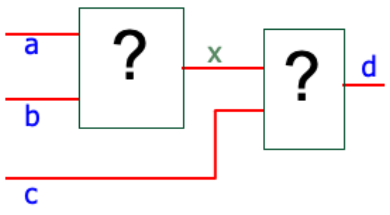


- VHDL semantics require that x retains its value over the entire simulation run
- However x is assigned before reading and used immediately
- Not necessary to store the value of x in a memory element
- d is always assigned and never read; memory not needed

University of BRISTOL

Ambiguous Statements

```
SIGNAL a, b, c, d: BIT;
---
PROCESS(a, b, c)
   VARIABLE x: BIT;
BEGIN
    d <= x OR c;
    x := a AND b;
END PROCESS;
```



- VHDL semantics require that x retains its value over the entire simulation run
- X is read before assigned
- Hence x has to hold its value during repeated executions of process
- Infers memory
- However not clear how to build a latch/flip-flop since x is not assigned under control of any condition
- Will result in error or warning
- Simulation results may differ between pre- and post-synthesis

Inferring a Latch

```
ENTITY what IS
PORT (clk, d: IN std_logic;
   q: out std_logic);
END what;

ARCHITECTURE behav OF what IS
BEGIN
   PROCESS(clk, d)
   BEGIN
      IF clk = '1' THEN
          q <= d;
      END IF;
   END PROCESS;
END behav;
```

- VHDL semantics require that q retains its value over the entire simulation run
- Not assigned in each branch
- Latch inferred as assignment is under the control of a level-sensitive condition

Do not use latches!

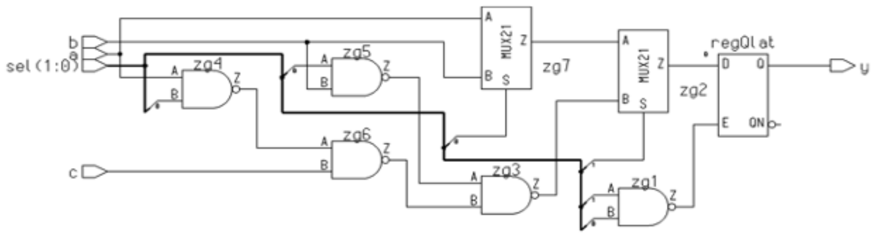
Mistakenly Inferring Latches with CASE

- Not assigning all conditional expressions in a CASE structure can lead to latches being inferred (similar to IF situation)

```
library IEEE;
use IEEE.std_logic_1164.all;

ENTITY mux3_seq is
  PORT(a    : IN  std_logic;
       b    : IN  std_logic;
       c    : IN  std_logic;
       sel : IN std_logic_vector(1 DOWNTO 0);
       y    : OUT std_logic);
END mux3_seq;
```

```
ARCHITECTURE behavior OF mux3_seq IS
  BEGIN
    comb : PROCESS(a,b,c,sel)
      BEGIN
        CASE sel IS
          WHEN "00" => y <= a;
          WHEN "01" => y <= b;
          WHEN "10" => y <= c;
          WHEN OTHERS => --empty
        END CASE;
    END PROCESS comb;
END behavior;
```



University of BRISTOL

Avoiding Unwanted Latches

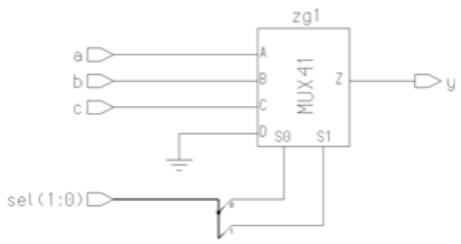
- Assigning values of "don't care" ('-') in these cases can avoid unwanted latch inference

```
library IEEE;
use IEEE.std_logic_1164.all;

ENTITY mux3 is
  PORT(a   : IN  std_logic;
       b   : IN  std_logic;
       c   : IN  std_logic;
       sel : IN std_logic_vector(1 DOWNTO 0);
       y   : OUT std_logic);
END mux3;

ARCHITECTURE behavior OF mux3 IS

  BEGIN
    comb : PROCESS(a,b,c,sel)
      BEGIN
        CASE sel IS
          WHEN "00" => y <= a;
          WHEN "01" => y <= b;
          WHEN "10" => y <= c;
          WHEN OTHERS => y <= '-';
        END CASE;
    END PROCESS comb;
END behavior;
```



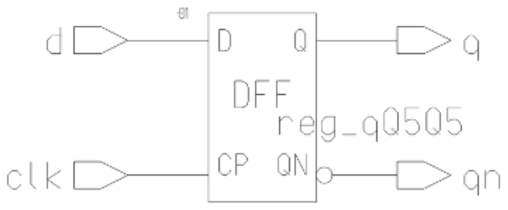
Edge Sensitive D Flip-Flop

- Assignment on rising or falling edge of control signal (Clock)
  - **rising_edge()** and **falling_edge()** functions can be used to specify clock edge
- Memory inferred because “d” is not assigned for all cases

```
library IEEE;
use IEEE.std_logic_1164.all;

ENTITY d_ff is
  PORT(d   : IN  std_logic;
       clk : IN  std_logic;
       q   : OUT std_logic;
       qn  : OUT std_logic);
END d_ff;

ARCHITECTURE behavior OF d_ff IS
  BEGIN
    seq : PROCESS(clk)
      BEGIN
        IF(rising_edge(clk)) THEN
          q <= d;
        END IF;
    END PROCESS seq;
    qn <= not q;
END behavior;
```



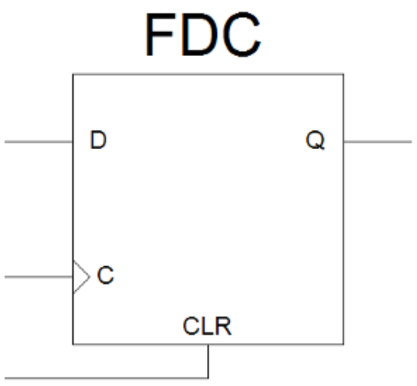
Edge Sensitive D Flip-Flop with Synchronous Reset

- Synchronous reset, so only clk in sensitivity list
- Memory inferred because “d” is not assigned for all cases

```
library IEEE;
use IEEE.std_logic_1164.all;

ENTITY d_ff is
  PORT(d    : IN  std_logic;
       clk : IN  std_logic;
       q    : OUT std_logic);
END d_ff;

ARCHITECTURE behav OF FlipFlop IS
BEGIN
    PROCESS(clk)
    BEGIN
        IF clk'event AND clk= '1' THEN
           IF reset = '0' THEN
                q <= '0';
           ELSE
                q <= d;
           END IF;
        END IF;
    END PROCESS;
END behav;
```



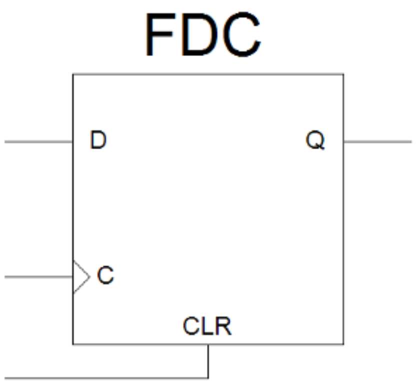
Edge Sensitive D Flip-Flop with Asynchronous Reset

- The only signals in the process sensitivity list other than CLOCK should be asynchronous signals
- Memory inferred because “d” is not assigned for all cases

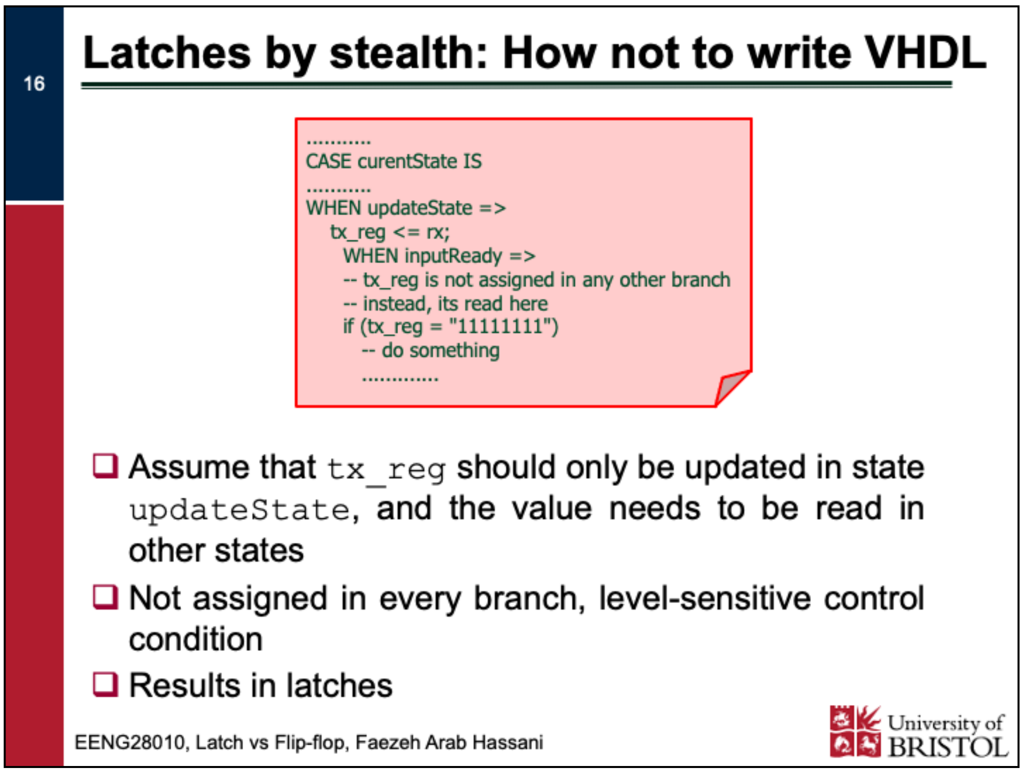
```
library IEEE;
use IEEE.std_logic_1164.all;

ENTITY d_ff is
  PORT(d   : IN  std_logic;
       clk : IN  std_logic;
       q   : OUT std_logic);
END d_ff;

ARCHITECTURE behav OF FlipFlop IS
BEGIN
    PROCESS(clk, reset)
    BEGIN
        IF reset = ‘0’ THEN
           q <= ‘0’;
        ELSIF clk‘event AND clk= ‘1’ THEN
           q <= d;
        END IF;
    END PROCESS;
END behav;
```



University of BRISTOL



Latches by stealth: How not to write VHDL

```
...........
CASE curentState IS
...........
WHEN updateState =>
   tx_reg <= rx;
     WHEN inputReady =>
     -- tx_reg is not assigned in any other branch
     -- instead, its read here
     if (tx_reg = "11111111")
       -- do something
       .............
```

- Assume that `tx_reg` should only be updated in state `updateState`, and the value needs to be read in other states
- Not assigned in every branch, level-sensitive control condition
- Results in latches

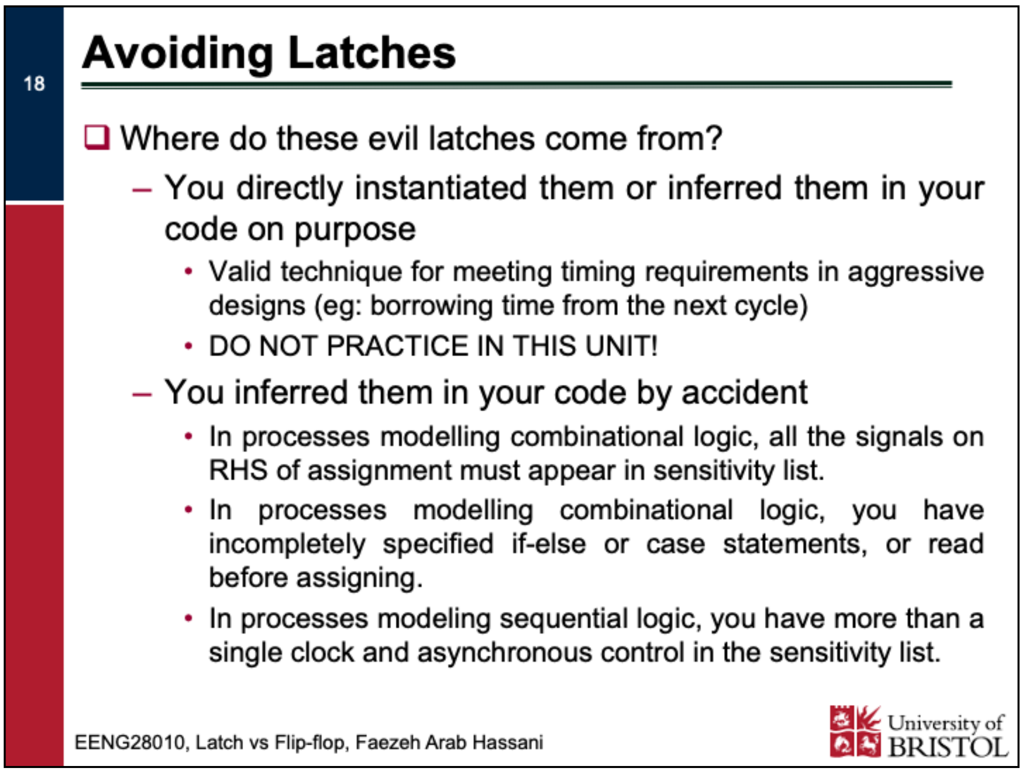
University of BRISTOL

Let's assume that for this design to work, the value of tx_reg should only be updated in state updateState, and the value needs to be read in other states, i.e. memory is needed. Written in this form, a latch is implied, as the assignment is under the control of a level-sensitive condition, and it is not assigned in every branch. A latch being implied is bad enough, but this is implication by stealth, it does not follow a latch template. This code is unlikely to work even in simulation. It is guaranteed not to work when synthesised.

Avoiding Latches by Stealth

```
regSync : PROCESS (clk)
BEGIN
    IF clk'EVENT AND clk='1' THEN
      IF reset='1' THEN
        tx_reg <= X"FF"; -- assign a HEX value to std_logic_vector
      ELSIF currentState = updateState THEN
        tx_reg <= rx;
      END IF;
    END IF;
END PROCESS; -- regSync
```

- Separate Process for the memory element
- Everything that happens inside the process body is predicated upon a clock edge (including the synchronous reset; could also be asynchronous reset)



Avoiding Latches

- Where do these evil latches come from?
  - You directly instantiated them or inferred them in your code on purpose
    - Valid technique for meeting timing requirements in aggressive designs (eg: borrowing time from the next cycle)
    - DO NOT PRACTICE IN THIS UNIT!
  - You inferred them in your code by accident
    - In processes modelling combinational logic, all the signals on RHS of assignment must appear in sensitivity list.
    - In processes modelling combinational logic, you have incompletely specified if-else or case statements, or read before assigning.
    - In processes modeling sequential logic, you have more than a single clock and asynchronous control in the sensitivity list.

University of BRISTOL