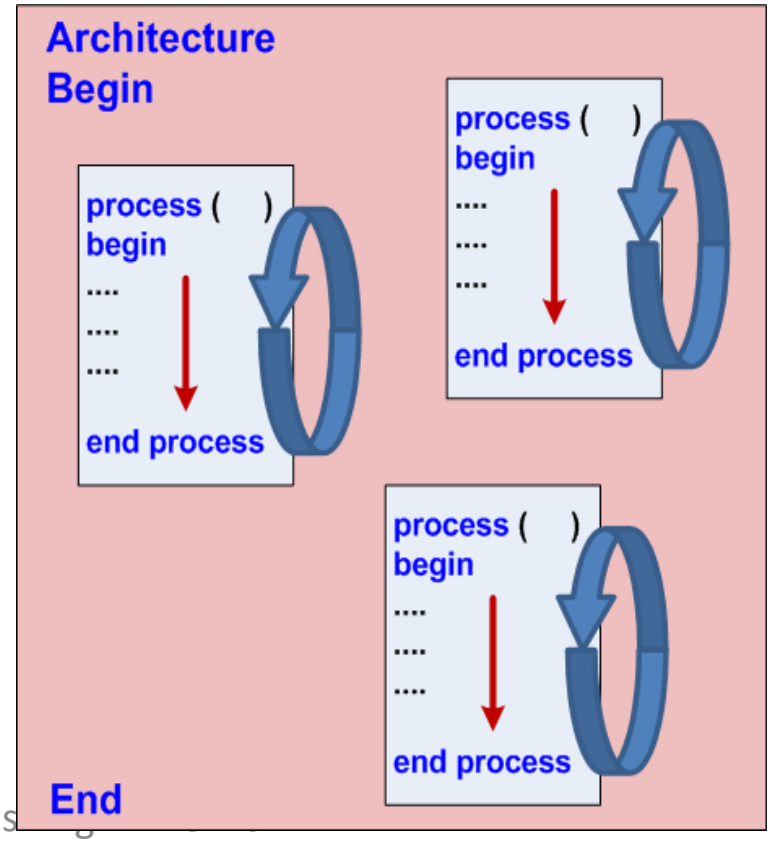
Programmable Electronics using FPGAs

This week

- Recap: A process block
- Procedural statements: wait, if-else, case, while, for
- Blocking and non-blocking statements
- How to avoid latches
- Different styles of VHDL coding
- Lab work: Identify, design, simulate and test a circuit

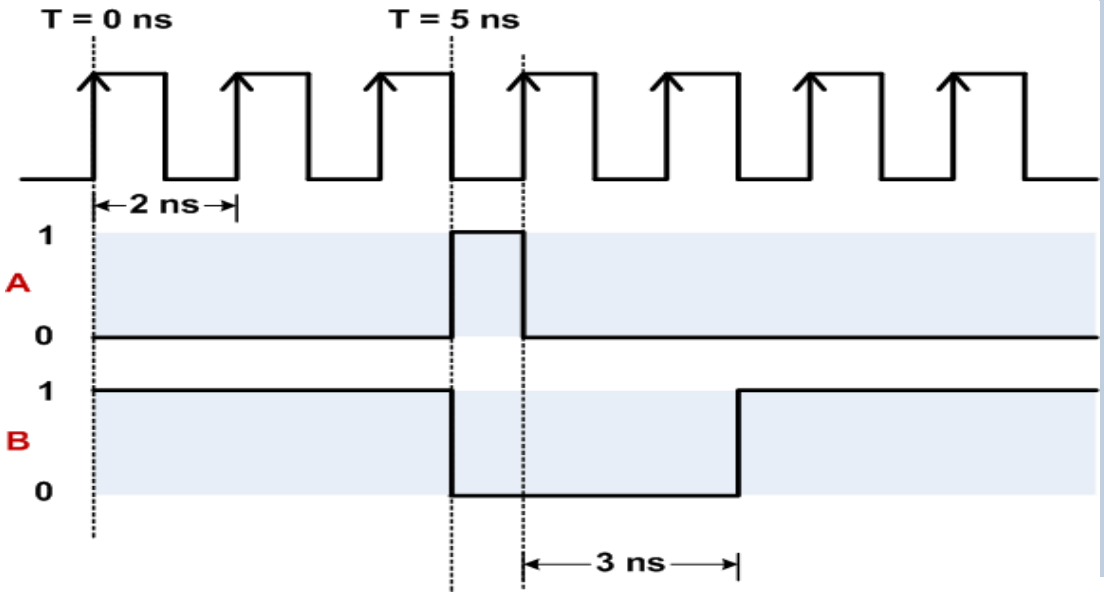
Recap: Process blocks

- A process is the sequential composition of a number of (related) combinational logic statements.
- The statements inside a process are called procedural (sequential) statements.
- Combinatorial logic: No edge triggered logic in a process
- Sequential logic: Edge triggered logic in a process



Sequential statements: wait

- wait statement is used:
  - To suspend the execution of a process;
  - To specify a condition when a process will be resumed;
- Not allowed in a process having sensitivity list.
- Synthesis tools ignore wait statement
- Only used in a test bench!

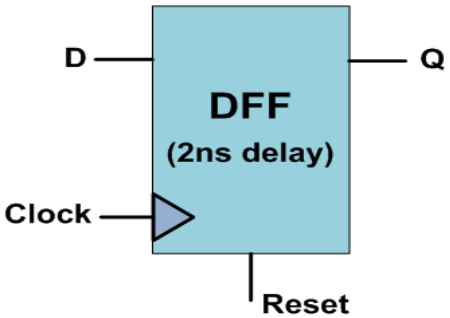


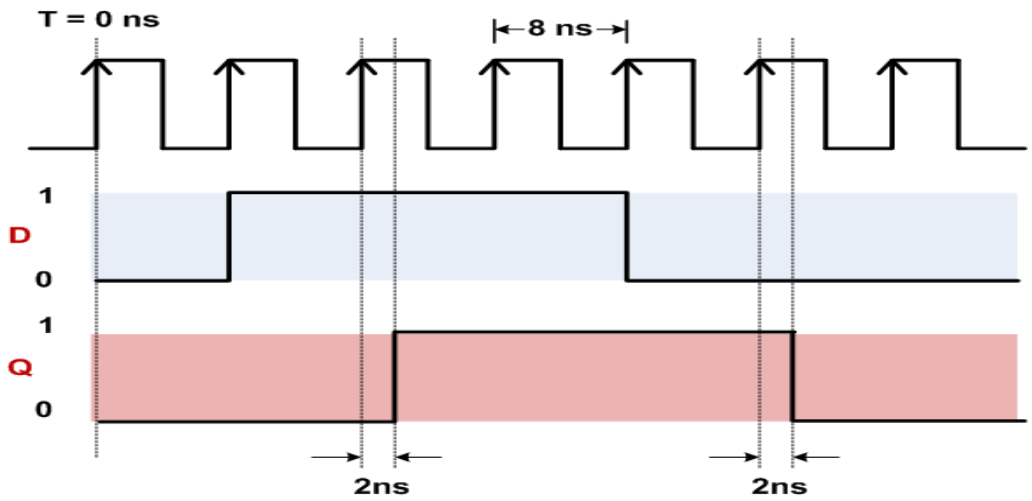

```
PROCESS BEGIN
    A <= '0';
    B <= '1';
    wait for 5 ns;

    A <= '1';
    B <= '0';
    wait until rising_edge(Clock);
    A <= '0';
    wait for 3 ns;
    B <= '1';
END PROCESS;
```

Sequential statements: after <delay>

- after clause can be used in sequential statements to model delays in signals.
  - Expression on right-hand side will be assigned to the signal on left-hand side after the delay specified from the current time.
- Synthesis tools ignore after clauses.
- Can be observed in simulation

```
PROCESS ( Clock, Reset )
BEGIN
    IF Reset = '1' THEN
        Q <= '0';
    ELSIF rising_edge(Clock) THEN
        Q <= D after 2 ns;
    END IF;
END PROCESS;
```

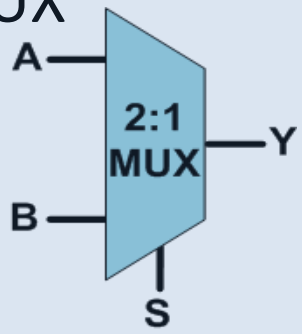




Sequential statements: if-else

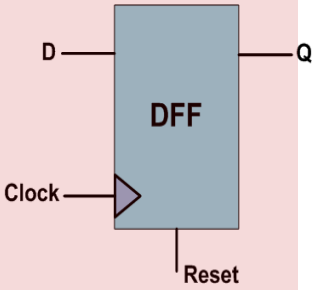
- if statement is a conditional branch.
- Based on an expression, one of two possible execution branches are chosen.

Example: 2-to-1 MUX




```
PROCESS ( A, B, S )
BEGIN
        IF   S = '1'  THEN
             Y <= A;
        ELSE
             Y <= B;
        END IF;
END PROCESS;
```

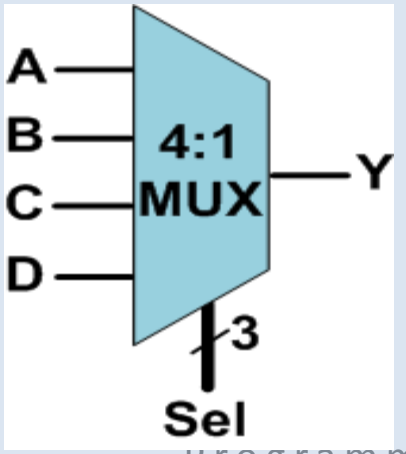
Example: DFF with reset




```
PROCESS ( Clock, Reset )
BEGIN
      IF Reset = '1' THEN
            Q <= '0';
      ELSIF rising_edge(Clock) THEN
            Q <= D;
       END IF;
END PROCESS;
```

Sequential statements: case

- Based on a expression, the case statement selects one of several alternative statements for execution.
- Unlike if statement, case statement does not require a boolean expression.
- others is the default if no others are triggered.



Example: 4-to-1 MUX

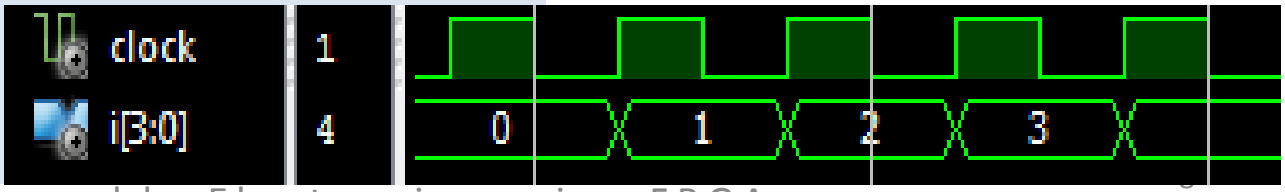
```
PROCESS ( A, B, C, D, Sel )
BEGIN
    CASE Sel IS
        when "000" => Y <= A;
        when "001" => Y <= B;
        when "010" => Y <= C;
        when "011" => Y <= D;
        when others => Y <= '0';
    END CASE;
END PROCESS;
```

Sequential statements: while *loop*

- A while loop statement is a conditional loop statement. It executes until the condition becomes false.
- Beware of using while loops in RTL design, as there may be a hidden impact on clock speed.

Example: A while loop in a process

```
PROCESS
BEGIN
    while i <= 3  loop
        wait until rising_edge(Clock);
        i <= i + 1;
    end loop;
END PROCESS;
```

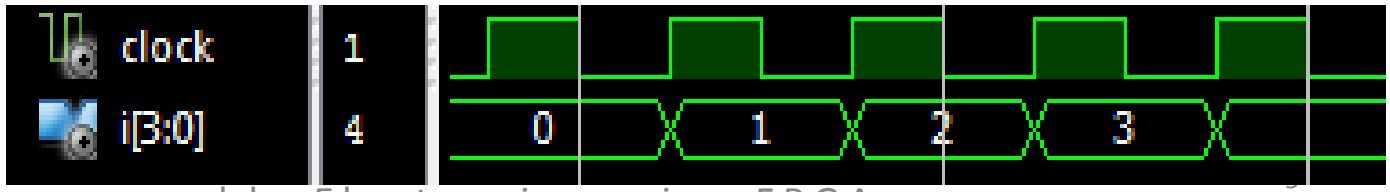


Sequential statements: for loop

- A for loop statement executes one or more statements repeatedly a fixed number of times.

Example: A for loop in a process

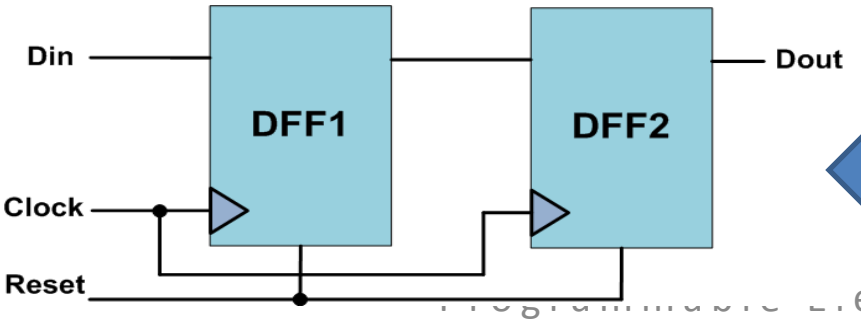
```
PROCESS
BEGIN
    for i in 0 to 3 loop
        wait until rising_edge(Clock);
    end loop;
END PROCESS;
```



Non-blocking sequential statements

- Signal assignment statements are executed sequentially, but are also non-blocking!
  - *signal <= expression;*
  - Signals have no intermediate values.
- We could have written RTL as two concurrent processes.

Question – how can you swap the value of two variables?




```
Architecture  RTL of ShfReg is
 signal dff1, dff2 : STD_LOGIC;
begin
    process(Clock, Reset)
    begin
        if Reset = '1' then
            dff1 <= '0';
            dff2 <= '0';
        elsif rising_edge(Clock) then
            dff1 <= Din;
            dff2 <= dff1;
        end if;
        Dout <= dff2;
    end process;
End RTL;
```

Blocking sequential statements

- The keyword variable can be used to store intermediary results in a process.
  - Declared and used within a process block only – not globally in an architecture
- Variable assignment statements are blocking
  - *variable := expression;*

```
PROCESS  (A)
variable X, Y, Z : std_logic := "00";
BEGIN
        X := A ;
        Y := X + 1;
        Z := X + Y ;
END PROCESS;
```

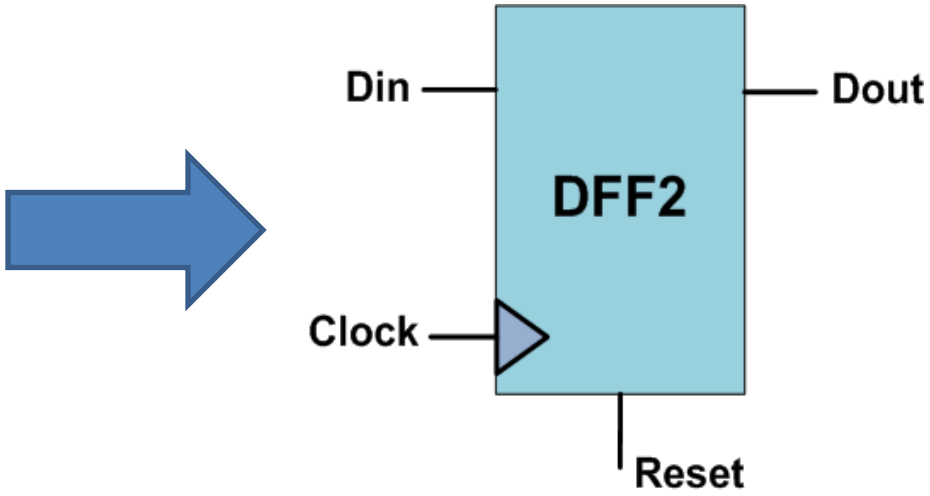
Assume *A* is 1 before entering in this process block

At the end of process block,

X will be 1
Y will be 2
Z will be 3

Blocking sequential statements

```
Architecture  RTL of ShfReg is
begin
    process(Clock, Reset)
    variable dff1, dff2 : STD_LOGIC;
    begin
        if Reset = '1' then
            dff1 := '0';
            dff2 := '0';
        elsif rising_edge(Clock) then
            dff1 := Din;
            dff2 := dff1;
        end if;
        Dout <= dff2;
    end process;
End RTL;
```



How to avoid latches?

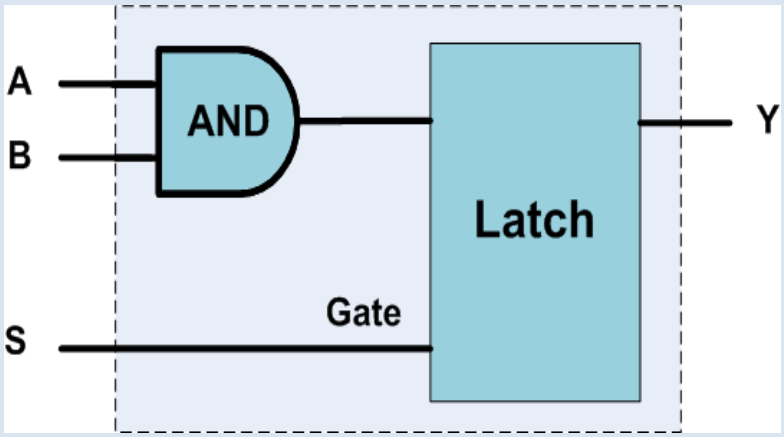
- All input signals of a process must be mentioned in its sensitivity list.
- if statement must be completely specified – there must be an else clause present.
- Incomplete if statements or sensitivity lists may create a latch.
- Unlike DFF a latch is a level-trigger storage.
- We should avoid latches in FPGA designs, as they may lead to timing problems.

Example:

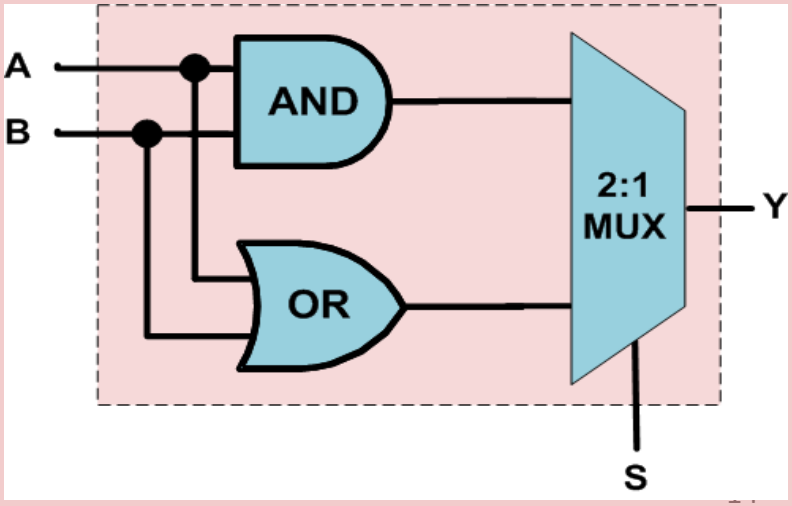
```
PROCESS ( A )  -- Here B is missing in sensitivity list.
BEGIN
    Y <= A AND B;
END PROCESS;
```

How to avoid latches?

```
PROCESS ( A, B, S )
BEGIN
        if S = '1'  then
              Y <= A AND B;
        end if;
END PROCESS;
```

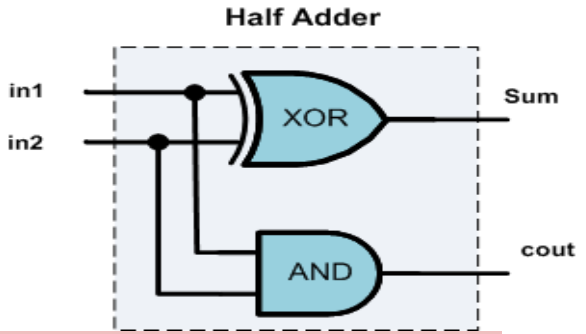



```
PROCESS ( A, B, S )
BEGIN
        if S = '1'  then
              Y <= A AND B;
        else
              Y <= A OR B;
        end if;
END PROCESS;
```



Different styles of VHDL coding

- A circuit can be designed using different styles in VHDL without compromising the resources.



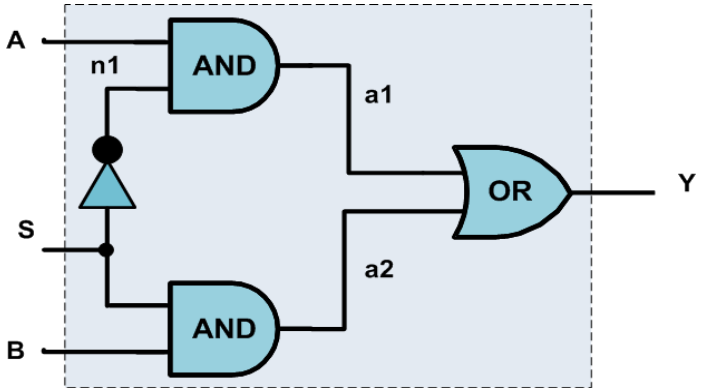
Example: Half-adder using dataflow modelling

```
Architecture RTL of HalfAdd IS
BEGIN
    Sum <= in1 XOR in2;
    Cout <= in1 AND in2;
END RTL;
```

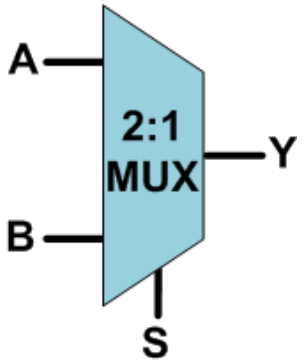
Example: Half-adder using behavioural modelling

```
Architecture RTL of HalfAdd IS
BEGIN
    PROCESS ( in1, in2 )
    BEGIN
        Sum <= in1 XOR in2;
        Cout <= in1 AND in2;
    END PROCESS;
END RTL;
```

Different styles of VHDL coding



OR



Example: MUX using dataflow modelling

```
Architecture RTL of Mux IS
BEGIN
    n1 <= NOT S;
    a1 <= A AND n1;
    a2 <= B AND S;
    Y <= a1 OR a2;
END RTL;
```

Example: MUX using behavioural modelling

```
Architecture RTL of Mux IS
BEGIN
    PROCESS ( A, B, S )
    BEGIN
        if S = '0' then
            Y <= A;
        else
            Y <= B;
        end if;
    END PROCESS;
END RTL;
```

Lab work: Identify, design, simulate, and test a circuit

- **Step 1:** Represent this circuit using Boolean expressions and a truth table.
- **Step 2:** Implement this circuit in VHDL firstly using dataflow modelling, and secondly using behavioural modelling.
- **Step 3:** Create a test bench and compare the simulation with your truth table.
- **Step 4:** Implement your design on the FPGA board.