# Lab5

# 1 Introductions

## 1.1 Goals

- Have ability in using behavioral model.
- Design and implement simple counter circuit.
- Design and implement frequency divider circuit.

## 1.2 Preparations

- Revise chapter 4 about FlipFlop and chapter 6 about counter circuit.
- Revise how to use behavioral model to describe a circuit.

## 1.3 Targets

- Using behavioral model to implement frequency divider circuit.
- Design and implement counter circuits.

## 2 Contents

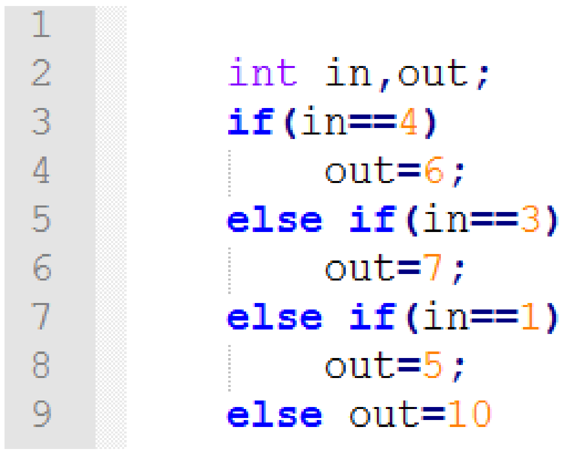
**Question 1 (0.5):** ditinguish the two following principles:

- Combination circuit: ...

- Sequential circuit: ...

## 2.1 Control statements using RTL model

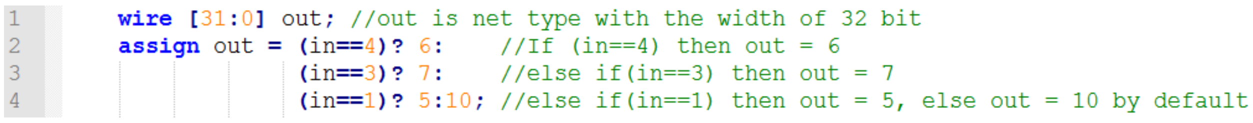
If-else, switch-case is a very effective statement to declare a control architecture. In this part, we will learn how to use if-else, switch-case in Verilog . Figure 1 is a simple example of using if-else to describe a control architecture in C.



```c

    int in,out;
    if(in==4)
        out=6;
    else if(in==3)
        out=7;
    else if(in==1)
        out=5;
    else out=10
```

Figure 1: if-else in C.

Ww can use RTL model with continuous assign to describe a branch declaration as figure 2.



```verilog
    wire [31:0] out; //out is net type with the width of 32 bit
    assign out = (in==4)? 6:    //If (in==4) then out = 6
                 (in==3)? 7:    //else if(in==3) then out = 7
                 (in==1)? 5:10; //else if(in==1) then out = 5, else out = 10 by default
```

Figure 2: Branch declaration by using RTL model.

The syntax is **a = (condition)? b:c** with:

- **a** is the variable that stores result.
- **condition**.
- **b** result of the statment if **condition** is true.
- **c** result of the statment if **condition** is false.

In verilog, all declarations always generate a circuit, that means the result's variable always connect with a signal and it value change whenever its input signals change.

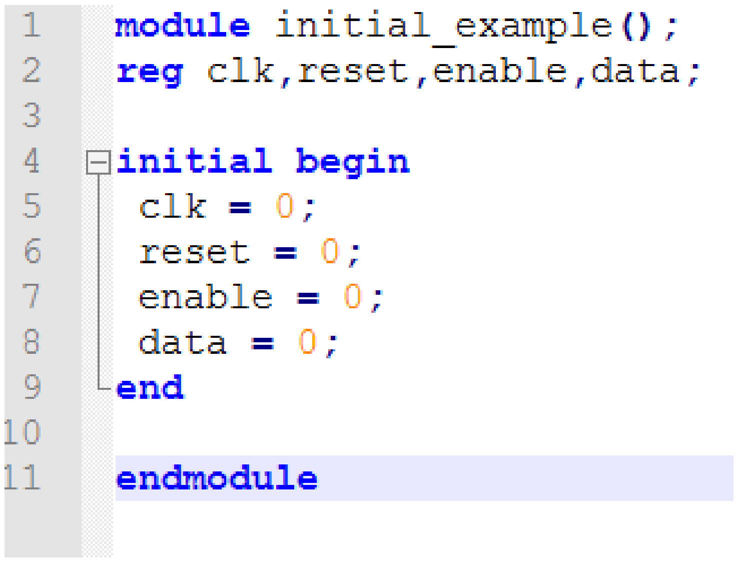
## 2.2 Branch architecture using behavioral model

There are 2 types of block statements can be used in Verilog to describe the circuits' behaviors in behavioral model.

- **initial**: An initial block, as the name suggests, is executed only once when the circuit starts. It's useful to init the very first value for **registers**
- **always**: As the name suggests, an always block executes always, unlike initial blocks which execute only once (at the beginning). A second difference is that an always block should have a sensitive list.

  The sensitive list is the one which tells the always block when to execute the block of code. The symbol after reserved word ' always', indicates that the block will be triggered "at" the condition in parenthesis after symbol @.
  One important note about always block: it can not drive wire data type, but can drive reg types.

Example of initial block:

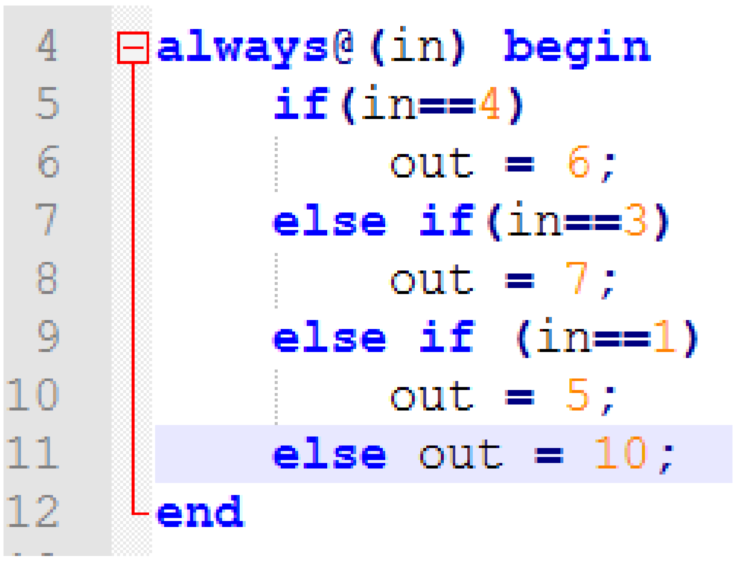


```
module initial_example();
reg clk,reset,enable,data;

initial begin
 clk = 0;
 reset = 0;
 enable = 0;
 data = 0;
end

endmodule
```

Figure 3: Initial using in init the first value for registers.

Example of always block and control architecture:

- Using " always" to generate combination circuit:

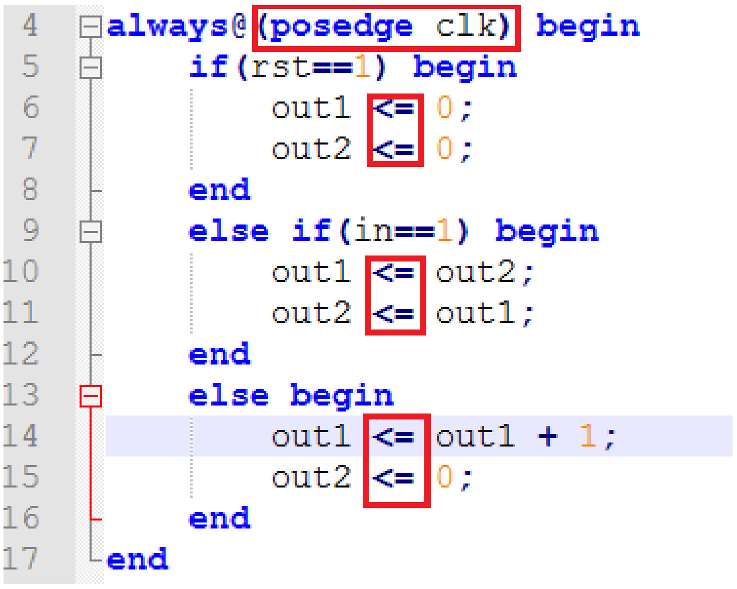


```
always@(in) begin
    if(in==4)
        out = 6;
    else if(in==3)
        out = 7;
    else if (in==1)
        out = 5;
    else out = 10;
end
```

Figure 4: Branch architecture in combination circuit.

In this example, the declarations in block will generate a circuit that execute your declarations **in order**.

- Using " always" to generate sequential circuit (used to be triggered by clock):



```
always@(posedge clk) begin
    if(rst==1) begin
        out1 <= 0;
        out2 <= 0;
    end
    else if(in==1) begin
        out1 <= out2;
        out2 <= out1;
    end
    else begin
        out1 <= out1 + 1;
        out2 <= 0;
    end
end
```

Figure 5: Branch architecture in sequential circuit.

In this example, whenever the circuit detects a clock's edge (positive or negative edge) that was described in sensitive list, all statements in the block are executed **parallel**. In this type of block, we use ¡= for assign statement, this kind of assignment is called as non-blocking assignment.

**Question 2 (1):**

– How can you declare a negative triggered edge in sensitive list?

– In the example 6, assume that out1 = 5 and out2 = 0, in = 1. What is the value of out1 and out2 in the clock trigger? Explain your results?

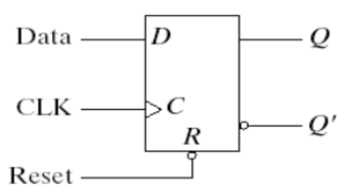
- In the above example, we are using rst to reset the output value. This kind of reset is called as synchronous reset. You have to change the code to describe an asynchronous reset. (Tip: you should distinguish between synchronous and asynchronous reset by looking on http://vlsi.pro/synchronous-asynchronous-reset/)

## 3 Exercices

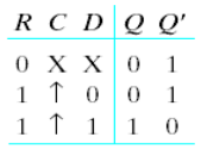
**Exercices 1,2 and 3 will be graded in class**
In this lab, you must to have a hard copy of your report and submit it to lab room. The deadline is on Tuesday 8 May. **The report must to have**:

- Name and ID of every member on the report's cover.
- Answer all questions in Contents part.
- Complete all exercices and attach the following requests on your report:
  - Describe in detail all steps and methods to implement your circuit (for exercices 2, 4, 5, 6, 7, 8).
  - One picture with all scenarios about your circuit's test (waveform) (for exercices 1, 2, 4, 5, 6, 7 , 8).
  - One picture of the circuit generated by your verilog code.

**Exercice 1**: (1pt) Implement D FlipFlop(FF) and JK FF by the following features:



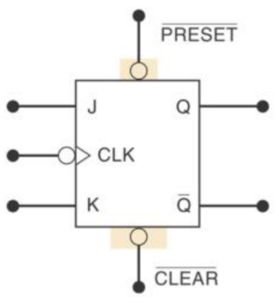
(b) Graphic symbol

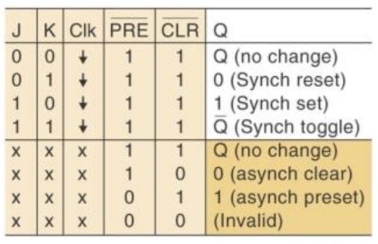


| $R$ | $C$ | $D$ | $Q$ | $Q'$ |
|---|---|---|---|---|
| 0 | X | X | 0 | 1 |
| 1 | ↑ | 0 | 0 | 1 |
| 1 | ↑ | 1 | 1 | 0 |

(b) Function table

Figure 6: D FF





| J | K | Clk | $\overline{\text{PRE}}$ | $\overline{\text{CLR}}$ | Q |
|---|---|---|---|---|---|
| 0 | 0 | ↓ | 1 | 1 | Q (no change) |
| 0 | 1 | ↓ | 1 | 1 | 0 (Synch reset) |
| 1 | 0 | ↓ | 1 | 1 | 1 (Synch set) |
| 1 | 1 | ↓ | 1 | 1 | $\bar{Q}$ (Synch toggle) |
| x | x | x | 1 | 1 | Q (no change) |
| x | x | x | 1 | 0 | 0 (asynch clear) |
| x | x | x | 0 | 1 | 1 (asynch preset) |
| x | x | x | 0 | 0 | (Invalid) |

Figure 7: JK FF

**Exercice 2**: (1pt) Implement a shift 4 bits circuit by using 4 D FF.
Requirements:

- Input: SW[0] as data input.
- Button KEY[0] as clock.

- Output: assign to [3:0]LEDG on board.

**Exercice 3**: (1pt) Design clock divider circuit by using DFF: DE2i-150 provide a 50Mhz clock frequency (read the manual to know which pin you need to use).
**only use DFF** Using DFF to divide the input frequency until it can be observed, assign the output to LEDR and you can see the blinking of LED when the input freq are divided into an appropriate level.
Hint: The frequency will be divided by 2 for each DFF it passes through.

## Counter circuit

Using the output of clock divider in Exercice 3 as clock input of counter circuit:

## Synchronous counter

**Exercice 4**: (1pt) Implement a synchronous counter mod 8 which can count **Up/Down** by using JK FF and show the result on 7seg-led.

**Exercice 5**: (1pt) Implement a synchronous counter mod 11 by using D FF with the rule: 0 1 3 5 2 4 6 7 14 10 11 0... show the result on 7seg-led Bonus (0.5) if you use JK FF instead of D FF.

**Exercice 6**: (0.5pt) Implement 4 bit Synchronous counter with asynchronous parallel load:

- The signal Parallel Load (PL) is active in 0, when PL is active, the data from SW[3:0] is loaded into the counter and is showed by LEDR[3:0].
- When PL is unactive, the counter begins to count up with mod 16.
- You can use either D FF or JK FF.

## Asynchronous counter

**Exercice 7**: (0.5pt) Implement an asynchronous counter mod 8 by using D FF and show the result on 7seg-led.

**Exercice 8**: (1pt) Implement an asynchronous counter by using JK FF and show the result on 7seg-led, select one the two following request:
Counter up from 0 to 8 and reset to 0.
Counter up from 0 n 14 v reset to 0 (+0.5 pt bonus

---

**Exercice 9**: Write your report with a clear structure(1.5pt).