Internal Assessment Test 3 Solution

Sub: Digital System Design using Verilog Sub **Code: BEC302**

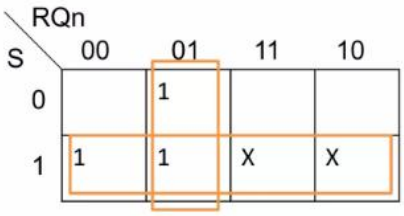
1 (a) Write the characteristics equation for SR flip-flop & JK flip –flop, and draw the excitation table for the SR flip-flop and JK flip-flop.

- **SR flip-flop**

Step 1: Characteristic Table of SR FF

S	R	Qn	Qn+1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

Step 2: K-Map



Step 3: Characteristic Equation

$$Q_{n+1} = S + R' Q_n$$

- Excitation table for the SR flip-flop and JK flip-flop.

Previous State	Next State	Required Inputs	
Qn	Qn+1	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

- **JK Flip flop**:

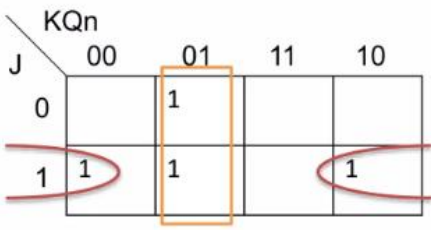
Characteristic Equation of JK Flip-Flop

CHITKARA UNIVERSITY

Step 1: Characteristic Table

J	K	Qn	Qn+1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Step 2: K-Map



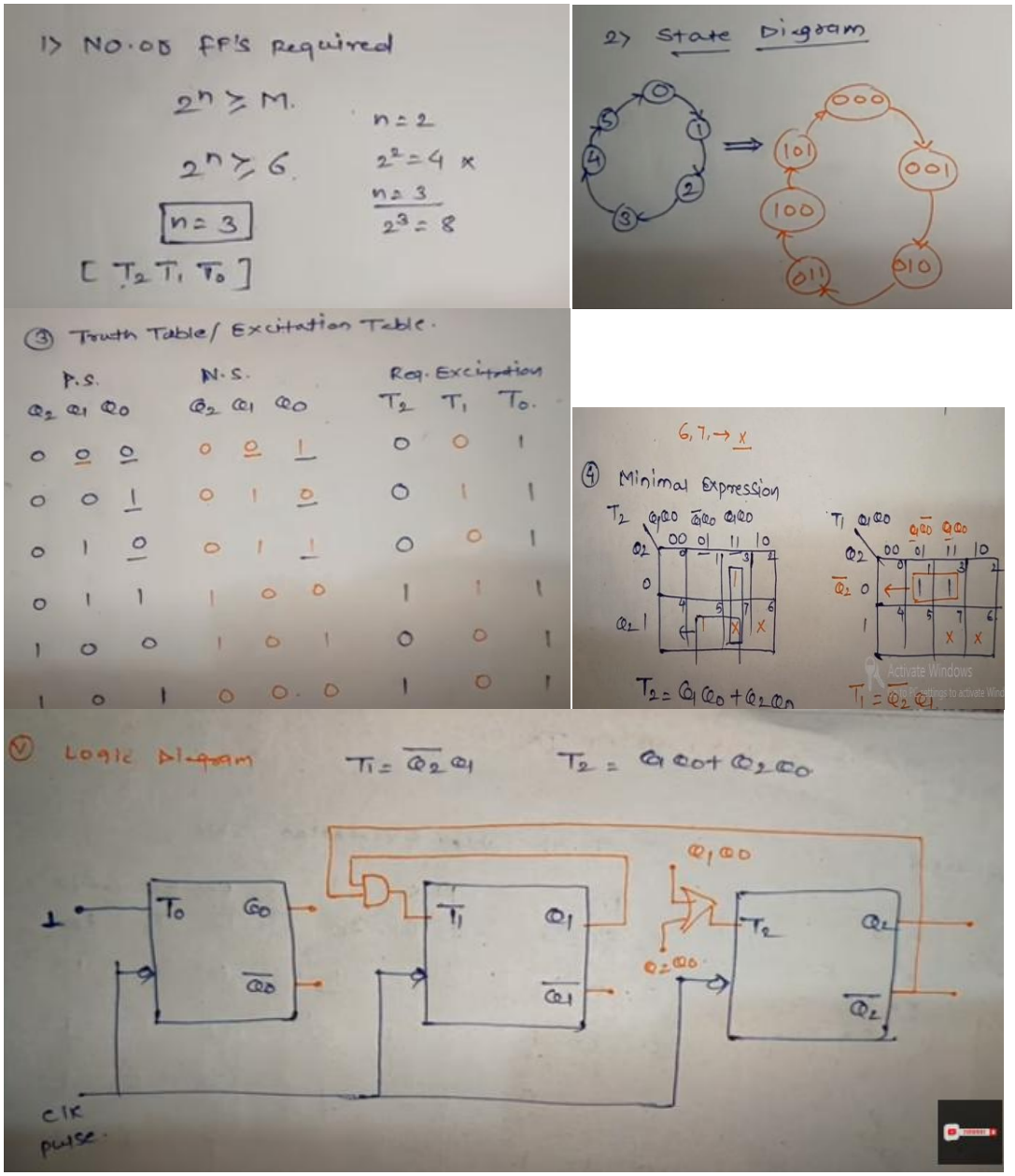
Step 3: Characteristic Equation

$$Q_{n+1} = Q_n' J + Q_n K'$$

Excitation Table of JK FF

Previous State	Next State	Required Inputs	
Qn	Qn+1	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

1 (b) Design a Mod-6 synchronous counter using T Flip-flop.

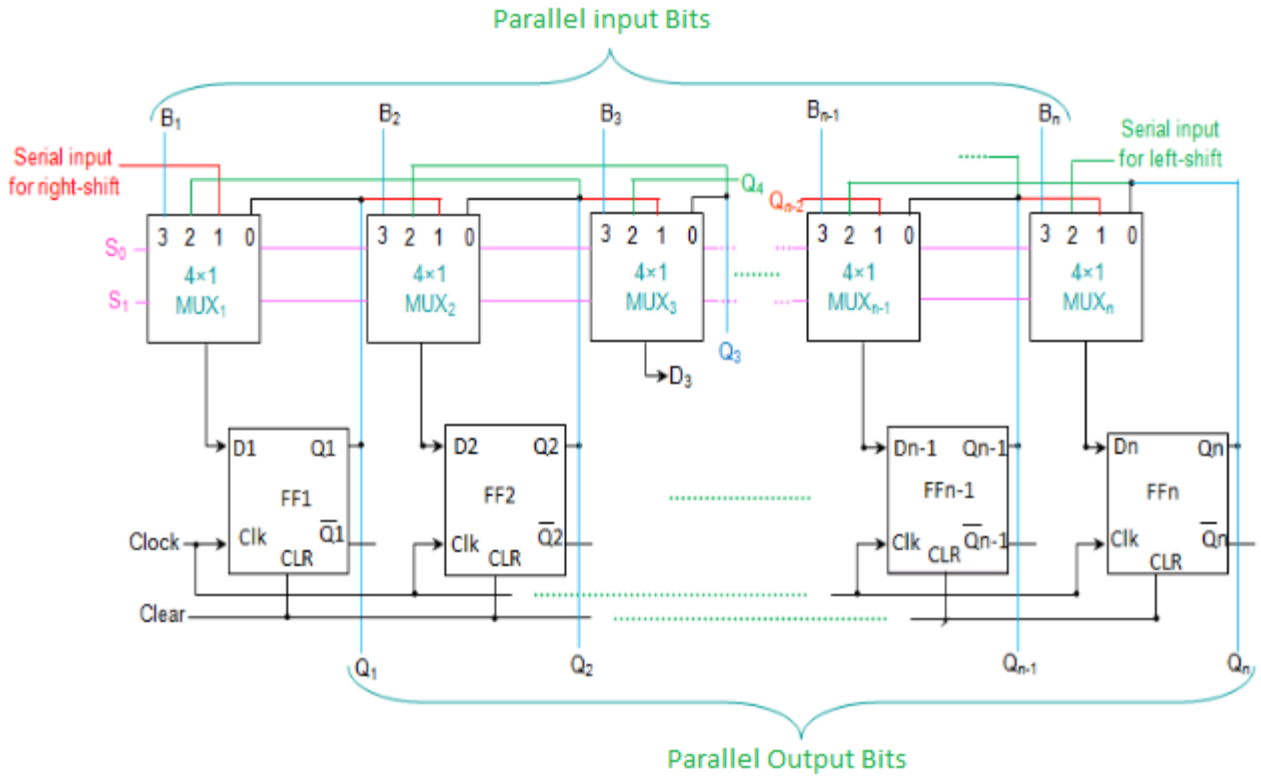


https://www.youtube.com/watch?v=_a8F4zLLSEI

2 Design a 4-bit universal shift register using positive edge triggered D-flip-flop and multiplexers to operate as indicated below:

Mode select Operation
00 Hold
01 Right Shift
10 Left Shift
11 Parallel Load

A Universal shift register is a register which has both the right shift and left shift with parallel load capabilities. Universal shift registers are used as memory elements in computers. A Unidirectional shift register is capable of shifting in only one direction. A bidirectional shift register is capable of shifting in both the directions. The Universal shift register is a combination design of **bidirectional** shift register and a **unidirectional** shift register with parallel load provision. **n-bit universal shift register –** A n-bit universal shift register consists of n flip-flops and n 4×1 multiplexers. All the n multiplexers share the same select lines(S1 and S0)to select the mode in which the shift register operates. The select inputs select the suitable input for the flip-flops.



Basic connections –

1. The first input (zeroth pin of multiplexer) is connected to the output pin of the corresponding flip-flop.
2. The second input (first pin of multiplexer) is connected to the output of the very-previous flip flop which facilitates the right shift.
3. The third input (second pin of multiplexer) is connected to the output of the very-next flip-flop which facilitates the left shift.
4. The fourth input (third pin of multiplexer) is connected to the individual bits of the input data which facilitates parallel loading.

The working of the Universal shift register depends on the inputs given to the select lines. The register operations performed for the various inputs of select lines are as follows:

S1	s0	Register operation
0	0	No changes
0	1	Shift right
1	0	Shift left
1	1	Parallel load

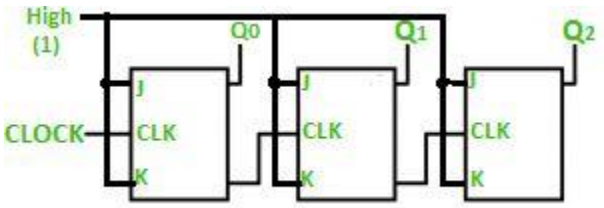
3 Design a 4-bit ripple counter using JK flip-flop along with the waveforms. Compare the counters based shift register.

Ripple counter is a cascaded arrangement of flip-flops where the output of one flip-flop drives the clock input of the following flip-flop. The number of flip flops in the cascaded arrangement depends upon the number of different logic states that it goes through before it repeats the sequence a parameter known as the modulus of the counter. A n-bit ripple counter can count up to 2n states. It is also known as MOD n counter. It is known as ripple counter because of the way the clock pulse ripples its way through the flip-flops. Some of the features of ripple counter are:

- It is an asynchronous counter.
- Different flip-flops are used with a different clock pulse.
- All the flip-flops are used in toggle mode.
- Only one flip-flop is applied with an external clock pulse and another flip-flop clock is obtained from the output of the previous flip-flop.
- The flip-flop applied with an external clock pulse act as LSB (Least Significant Bit) in the counting sequence.

A counter may be an up counter that counts upwards or can be a down counter that counts downwards or can do both i.e.count up as well as count downwards depending on the input control. The sequence of counting usually gets repeated after a limit. When counting up, for the n-bit counter the count sequence goes from 000, 001, 010, … 110, 111, 000, 001, … etc. When counting down the count sequence goes in the opposite manner: 111, 110, … 010, 001, 000, 111, 110, … etc.

A 3-bit Ripple counter using a JK flip-flop is as follows:



In the circuit shown in the above figure, Q0(LSB) will toggle for every clock pulse because JK flip-flop works in toggle mode when both J and K are applied 1, 1, or high input. The following counter will toggle when the previous one changes from 1 to 0.

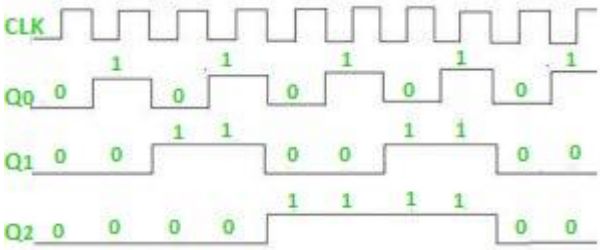
Truth Table is as follows:

Counter State	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

The 3-bit ripple counter used in the circuit above has eight different states, each one of which represents a count value. Similarly, a counter having n flip-flops can have a maximum of 2 to the power n states. The number of states that a counter owns is known as its mod (modulo) number. Hence a 3-bit counter is a mod-8 counter. A mod-n counter may also be described as a divide-by-n counter. This is because the most significant flip-flop (the furthest flip-flop from the original clock pulse) produces one pulse for every n pulses at the clock input of the least significant flip-flop (the one triggers by the clock pulse). Thus, the above counter is an example of a divide-by-4 counter.

Timing diagram

Let us assume that the clock is negative edge triggered so the above the counter will act as an up counter because the clock is negative edge triggered and output is taken from Q.



Counters are used very frequently to divide clock frequencies and their uses mainly involve

digital clocks and in multiplexing. The widely known example of the counter is parallel to serial data conversion logic.

Difference between Ring Counter & Johnson Counter?

Ring Counter:

This counter is developed by modifying a shift register. The true output of the last flip-flop is fed back directly to the data input of the first flip-flop, thus generating a sequence of pulses. For example, for a D Flip-Flop shift register, the Q output of the last flip-flop is connected to the D input of the first flip-flop. These counters are used in digital system to generate control pulses.

Johnson Counter

Johnson counter is a reverse of Ring Counter. In other words, feedback from the last flip-flop is fed inversely to the data input of the first flip-flop. For example, for a D Flip-Flop shift register, the ~Q output of the last flip-flop is fed to the D input of the first flip-flop. These can be used as Divide by n counters as well.

4 Illustrate with examples the data types used to define nets, registers, vectors and arrays.

Verilog supports several data types including nets, registers, vectors, integer, real, parameters, and arrays. More details on these types can be found in almost all subsequent chapters.

1.6.2.1 Nets

Nets are declared by the predefined word wire. Nets have values that change continuously by the circuits that are driving them. Verilog supports four values for nets, as shown in Table 1.13.

TABLE 1.13 Verilog Net Values

Value	Definition
0	Logic 0 (false)
1	Logic 1 (true)
X	Unknown
Z	High impedance

Examples of net types are as follows:

```
wire sum;
wire S1 = 1'b0;
```

The first statement declares a net by the name sum. The second statement declares a net by the name of S1; its initial value is 1'b0, which represents 1 bit with value 0.

1.6.2.2 Register

Register, in contrast to nets, stores values until they are updated. Register, as its name suggests, represents data-storage elements. Register is declared by the predefined word reg. Verilog supports four values for register, as shown in Table 1.14.

TABLE 1.14 Verilog Register Values

Value Definition

0 Logic 0 (false)
1 Logic 1 (true)
X Unknown
Z High impedance

An example of register is:

```
reg Sum_total;
```

The above statement declares a register by the name Sum_total.

1.6.2.3 Vectors

Vectors are multiple bits. A register or a net can be declared as a vector. Vectors are declared by brackets [ ]. Examples of vectors are:

```
wire [3:0] a = 4'b1010;
reg [7:0] total = 8'd12;
```

The first statement declares a net a. It has four bits, and its initial value is 1010 (b stands for bit). The second statement declares a register total. Its size is eight bits, and its value is decimal 12 (d stands for decimal). Vectors are implemented in almost all subsequent chapters.

1.6.2.4 Integers

Integers are declared by the predefined word integer. An example of integer declaration is:

```
integer no_bits;
```

The above statement declares no_bits as an integer.

1.6.2.4 Real

Real (floating-point) numbers are declared with the predefined word real. Examples of real values are 2.4, 56.3, and 5e12. The value 5e12 is equal to 5×10^{12}. The following statement declares the register weight as real:

```
real weight;
```

1.6.2.5 Parameter

Parameter represents a global constant. It is declared by the predefined word parameter. The following is an example of implementing parameters:

```
module compr_genr (X, Y, xgty, xlty, xeqy);
parameter N = 3;
input [N:0] X, Y;
output xgty, xlty, xeqy;
wire [N:0] sum, Yb;
```

To change the size of the inputs x and y, the size of the nets sum, and the size of net Yb to eight bits, the value of N is changed to seven as:

```
parameter N = 7
```

1.6.2.6 Arrays

Verilog, in contrast to VHDL, does not have a predefined word for array. Registers and integers can be written as arrays. Consider the following statements:

```
parameter N = 4;
parameter M = 3;
reg signed
```

5 Write the Verilog code for 8 x 1 MUX using CASE Statement.

```
module mux_case(out,in,s);
output reg out;
input [1:0]in;
input s;
always @ (*)
casex(s)
1'b0:  out=in[0];
1'b1:  out=in[1];
default :  out=1'bx;
endcase
endmodule
```

```
initial begin
      in = 2'b00;
   s = 0;
   end
   always #2 s=s+1;
   always #1 in=in+1;
   initial #20 $finish;
```

6 Illustrate the IF statement, IF as ELSE-IF, signal and variable assignment with an example.

3.4.1 IF Statement

IF is a sequential statement that appears inside process in VHDL or inside always or initial in Verilog. It has several formats, some of which are as follows:

Verilog IF-Else Formats

if (Boolean Expression)
begin
.
statement 1; /
if only one statement, begin and end
.
can be omitted
/
statement 2;
statement 3;
.....
end
else
begin
.
statement a; /
if only one statement, begin and end
.
can be omitted
/
statement b;
statement c;
.....
End

Example:

```
if (clk == 1'b1)
// 1'b1 means 1-bit binary number of value 1.
temp = s1;
else
temp = s2;
```

EXECUTION OF IF AS A LATCH

```
if (clk == 1)
begin
temp = s1;
end
```

```
Verilog
if (Boolean Expression1)
begin
statement1; statement 2;.....
end
else if (Boolean expression2)
begin
statementi; statementii;.....
end
else
begin
statementa; statement b;....
end
```

Example

```
if (signal1 == 1'b1)
temp = s1;
else if (signal2 == 1'b1
temp = s2;
else
temp = s3;
```

7 (a) Write a note on structure of Behavioral Description with example

all Verilog statements inside always
are treated as concurrent, the same as in the data-flow description

Verilog Description

```
module half_add (I1, I2, O1, O2);
input I1, I2;
output O1, O2;
reg O1, O2;
/☐ Since O1 and O2 are outputs and they are
written inside "always," they should be
declared as reg ☐/
always @(I1, I2)
begin
#10 O1 = I1 ^ I2; // statement 1.
#10 O2 = I1 & I2; // statement 2.
/☐The above two statements are
signal-assignment statements with 10 simulation screen units
delay☐/
```

/☐Other behavioral (sequential) statements can be added here☐/
end
endmodule

7 (b) Write a note on Signal Assignment and Variable Assignment with example.

Signal and Variable assignment With the help of Behavioral description of a D-latch, here we study the difference between the signal- and Variable assignment statements. A process is written based on signal-assignment statements, and then another program is written based on Variable assignment statements. A comparison of the simulation waveforms highlights the difference between the two methods.

Examples2: Behavioral description of the D-Latch. module D_latch (d, E, Q, Qb); input d, E; output Q, Qb; reg Q, Qb; always @ (d, E) begin end if (E == 1) begin end Q = d; Qb = ~Q; endmodule

8 Explain the structural description of 3-bit Ripple Carry Adder.

Verilog Description

```
module three_bit_adder (x, y, cin, sum, cout);
input [2:0] x, y;
input cin;
output [2:0] sum;
output cout;
wire [1:0] carry;
FULL_ADDER M0 (x[0], y[0], cin, sum[0], carry[0]);
FULL_ADDER M1 (x[1], y[1], carry[0], sum[1], carry[1]);
FULL_ADDER M2 (x[2], y[2], carry[1], sum[2], cout);
/☐ It is assumed that the module FULL_ADDER
endmodule
```