COMP ENG 2DI4
Digital Logic

Lab 5: Design and Implementation of Digital System

Due date: Friday, December 3rd, 2021

Submitted By:
Jil Shah
Shiv Thakar

Pre- Lab Questions

Q1 (5.5)

The differences between a truth table, state table, a characteristic table and an excitation table are stated below:

- Truth table: shows how the logic circuits outputs responds to various circuits input value combinations. Typically explains the functionality of the given logic circuit. It is the table which explains all possible combinations of inputs and the behaviour of the circuit.
- Characteristic Table: determines the next state of a flip flop in terms of the present state. This tells us how the control bit will affect the current state to produce the next state.
- Excitation Table: defines the flip flop variables as a function of the present and next state. Simply, it allows us to identify the known state, the expected future state, and the control bit. This will give us information regarding the excitations and how states change from present to future.
- State table: applicable to the sequential circuits that have different states. This will explain how the transitions occur in a sequential circuit, represents the relationship between present state, inputs, and future states.

The difference between a boolean equation, state equation, characteristic equation and a flip-flop input equation.

- Boolean Equation: expressions that define the logic circuits. An equation that would essentially generate a logical boolean value with specific inputs.
- State equations: expressions that will explain the state of the given circuit as a function of a different states or inputs.
- Characteristic equation: describes the next state of the circuit as a function if inputs and the present value
- Flip Flop Equations: describes how the next state can be obtained from the function table of the different flip flops, the equation will give the next state in terms of the present state outputs.

Q2 (5.19b)

State Table

Present State			Input	Next State			Outputs	FF					
A	B	C	x	A+	B+	C+	y	JA	KA	JB	KB	JC	KC
0	0	0	0	0	1	1	0	0	X	1	X	1	X
0	0	0	1	1	0	0	1	1	X	0	X	0	X
0	0	1	0	0	0	1	0	0	X	0	X	X	0
0	0	1	1	1	0	0	1	1	X	0	X	X	1
0	1	0	0	0	1	0	0	0	X	X	0	0	X
0	1	0	1	0	0	0	1	1	X	X	1	0	X
0	1	1	0	0	0	1	0	0	X	X	1	X	0
0	1	1	1	0	1	0	1	1	X	X	0	X	1
1	0	0	0	0	1	0	0	X	1	1	X	0	X
1	0	0	1	0	1	1	0	X	1	1	X	1	X

K-map

JA

AB/Cx	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	0	0	0
10	X	X	0	0

JA = A' · x

JB

AB/Cx	00	01	11	10
00	1	0	0	0
01	X	X	X	X
11	0	0	0	0
10	1	1	0	0

JB = A' C' x' + A B' C'

JC

AB/Cx	00	01	11	10
00	1	0	X	X
01	0	0	X	X
11	0	0	0	0
10	0	1	0	0

JC = A' B' x' + A B' C' x

KA

AB/Cx	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	0	0	0	0
10	1	1	0	0

KA = B' · C'

KB

AB/Cx	00	01	11	10
00	X	X	X	X
01	0	1	0	1
11	0	0	0	0
10	X	X	0	0

KB = A' C' x + A' C x'

KC

AB/Cx	00	01	11	10
00	X	X	1	0
01	X	X	1	0
11	0	0	0	0
10	X	X	0	0

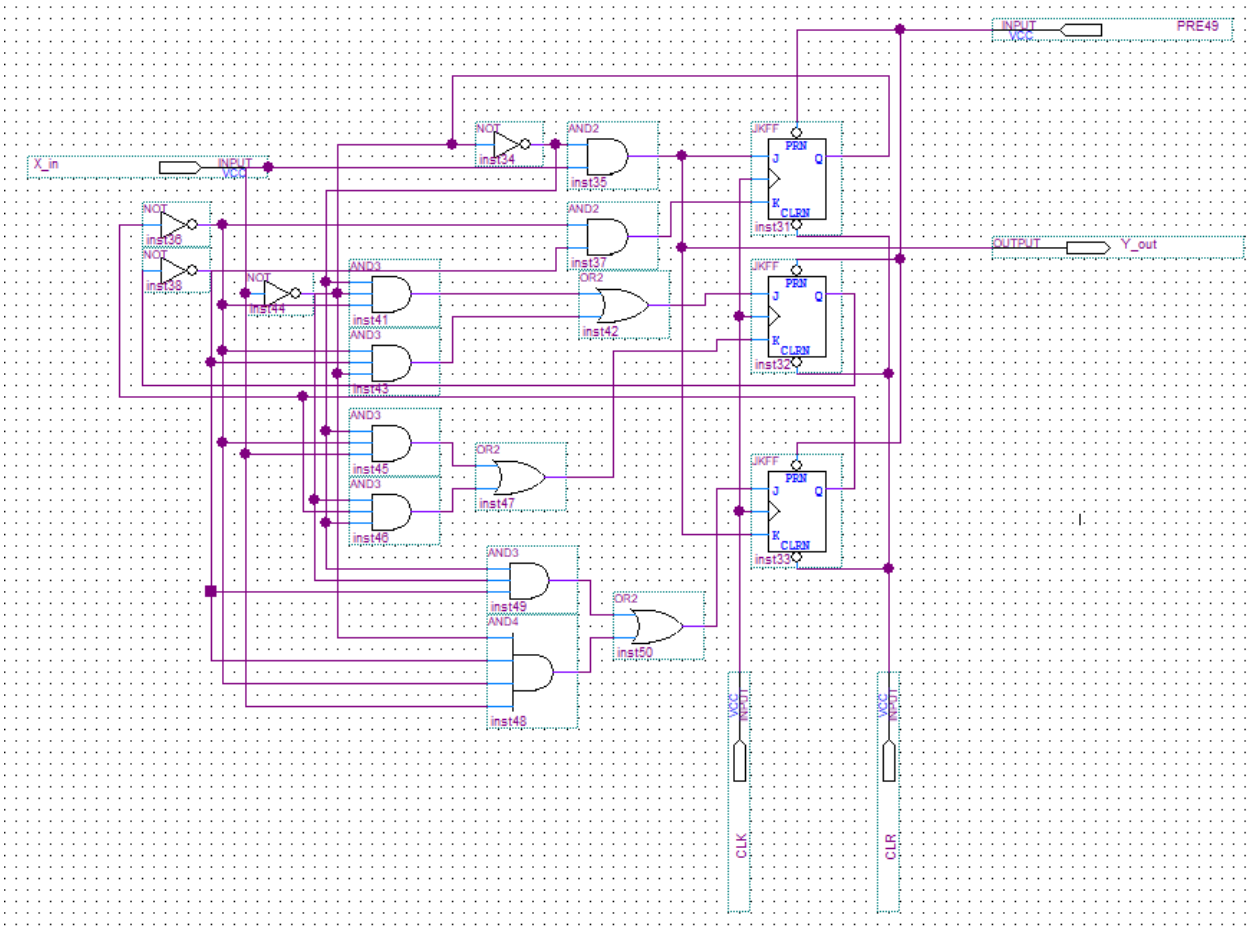
KC = JA

Y

AB/Cx	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	0	0	0
10	0	0	0	0

Y= JA = A' · x

Circuit



Q3 (6.3)

Serial and parallel transfer are both used generally for data transferring between technologies (i.e computers, laptops, phones, and hardware). Serial transmits data through a single channel one bit at a time and is bi-directional, whereas parallel transfer will transmit multiple bits(typically sends a byte) across different channels. Transmitting data can occur asynchronously or synchronously and is identified by an extra bit. To convert serial data to parallel data or parallel to serial data, we can use a shift register. For serial data, the SR will accept all bits into the register at the MSB and then it will take parallel data from the register output. For parallel to serial data conversion, the shift register will accept the parallel data through all channels, and then will transfer each bit across one single channel at the LSB.

Q4 (7.1)

a) 8 K x 32

a) $8K \times 32$ address lines : 13

$= 8 \cdot 2^{10} \cdot 32$ I/O lines : 32

$= 2^3 \cdot 2^{10} \cdot 32$

$= 2^{13} \cdot 32$

b) 2 G x 8

b) $2G \times 8$ address lines = 31

$= 2 \cdot 2^{30} \cdot 8$ I/O lines = 8

$= 2^{31} \cdot 8$

c) 16 M x 32

c) $16M \times 32$ address lines = 24

$= 16 \cdot 2^{20} \cdot 32$ I/O lines = 32

$= 2^4 \cdot 2^{20} \cdot 32$

$= 2^{24} \cdot 32$

d) 256 K x 64

d) $256K \times 64$ address lines = 18

$= 256 \cdot 2^{10} \cdot 64$ I/O lines = 64

$= 2^8 \cdot 2^{10} \cdot 64$

$= 2^{18} \cdot 64$

Q5 (7.8)

a)

a) $\frac{256\,K}{32\,K} = 8$ RAM chips

b)

b) $256\,K = 256 \cdot 2^{10}$

$= 2^8 \cdot 2^{10}$

$= 2^{18}$ → 18 address lines needed to access 256 K bytes

$32\,K = 32 \cdot 2^{10}$

$= 2^5 \cdot 2^{10}$

$= 2^{15}$ → 15 lines are connected

c) Three lines must be decoded for the chip to select inputs. (18 lines - 15 lines)

Q6 (8.1B)

```
//b) R3 <- R3 - 1
R3 = R3 - 1;
R3 <= R3 - 1;
```

The first statement is a procedural assignment called blocking. This is distinguishable with the = symbol. A blocking statement is executed in the order it is listed in. The second statement is a non-blocking statement (<=) which means the statement is executed simultaneously by evaluating the expression on the right side. For the first statement, decreases the value of R3 and assigns this value back to R3. For the second statement R3 - 1 gets stored in a temporary location, and once the expression is evaluated it will be stored within the original assignment that was written (R3).

Q7 (8.1C)

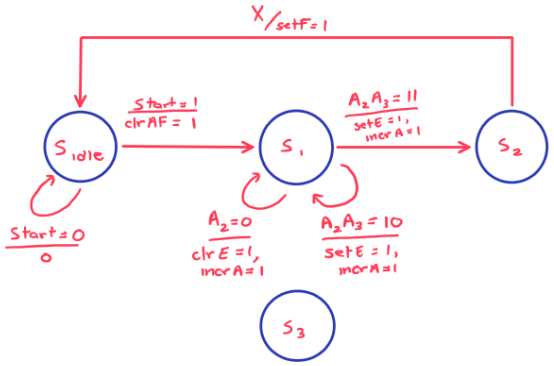
```
//c) if (S1 = 1) then (R0 <- R1) else if (S2 = 1) then (R0 <- R2)
if (S1 == 1)
    (R0 <= R1);
else if (S2 == 1):
    (R0 <= R2);
```

In this HDL code, the assignments occur within a decision statement. If the value of S1 is equal to 1, the program will store the value of R0. If the value of S2 is equal to 1, the contents of

R2 will be stored within R0. Both these statements use non-blocking procedural assignments. This means that the value on the right side (R1 and R2) will be stored within a temporary location and once the expression has been evaluated, then it will be stored in the specified location (R0 in this case).

Lab Experiment

State Diagram



State Table

	Present State		Inputs			Next State		Ouputs					FF	
	G1	G0	Start	A2	A3	G1	G0	set_E	clr_E	set_F	clr_A_F	incre_A	TG1	TG0
S_IDLE	0	0	0	X	X	0	0	0	0	0	0	0	0	0
S_IDLE	0	0	1	X	X	0	1	0	0	0	1	0	0	1
S1	0	1	X	0	X	0	1	0	1	0	0	1	0	0
S1	0	1	X	1	0	0	1	1	0	0	0	1	0	0
S1	0	1	X	1	1	1	1	1	0	0	0	1	1	0
S2	1	1	X	1	1	0	0	0	0	1	0	0	1	1
S3	1	0	X	X	X	X	X	X	X	X	X	X	X	X

We accounted for the unused states by giving them don't care inputs and outputs.
The design entry approach that we choose was the synthesis approach from Week 7. We determined the state table by using the textbook example and the T flip flop excitation table. We reduced the states to simplify the don't care values, we derived the input/output equations using K-maps. Finally, we implemented the logical circuit using Quartus and tested all possible inputs.

5-Variable K-Map

K-MAP - 5 Variable

Set_E

G1=0					G1=1				
G0S/A2A3	00	01	11	10	G0S/A2A3	00	01	11	10
00	0	0	0	0	00	X	X	X	X
01	0	0	0	0	01	X	X	X	X
11	0	0	1	1	11	0	0	0	0
10	0	0	1	1	10	0	0	0	0

Set_E = G1' ·G0·A2

ctlr_E

G1=0					G1=1				
G0S/A2A3	00	01	11	10	G0S/A2A3	00	01	11	10
00	0	0	0	0	00	X	X	X	X
01	0	0	0	0	01	X	X	X	X
11	1	1	0	0	11	0	0	0	0
10	1	1	0	0	10	0	0	0	0

ctrl_E = G1'·G0·A2'

set_F

G1=0					G1=1				
G0S/A2A3	00	01	11	10	G0S/A2A3	00	01	11	10
00	0	0	0	0	00	X	X	X	X
01	0	0	0	0	01	X	X	X	X
11	0	0	0	0	11	0	0	1	0
10	0	0	0	0	10	0	0	1	0

set_F = G1 · A2

clr_A_F

G1=0					G1=1				
G0S/A2A3	00	01	11	10	G0S/A2A3	00	01	11	10
00	0	0	0	0	00	X	X	X	X
01	1	1	1	1	01	X	X	X	X
11	0	0	0	0	11	0	0	0	0
10	0	0	0	0	10	0	0	0	0

ctr_A_F = G1' · G0' · S

incre_A

G1=0					G1=1				
G0S/A2A3	00	01	11	10	G0S/A2A3	00	01	11	10
00	0	0	0	0	00	X	X	X	X
01	0	0	0	0	01	X	X	X	X
11	1	1	1	1	11	0	0	0	0
10	1	1	1	1	10	0	0	0	0

incre_A = G0 · G1'

TG1

G1=0					G1=1				
G0S/A2A3	00	01	11	10	G0S/A2A3	00	01	11	10
00	0	0	0	0	00	X	X	X	X
01	0	0	0	0	01	X	X	X	X
11	0	0	1	0	11	0	0	1	0
10	0	0	1	0	10	0	0	1	0

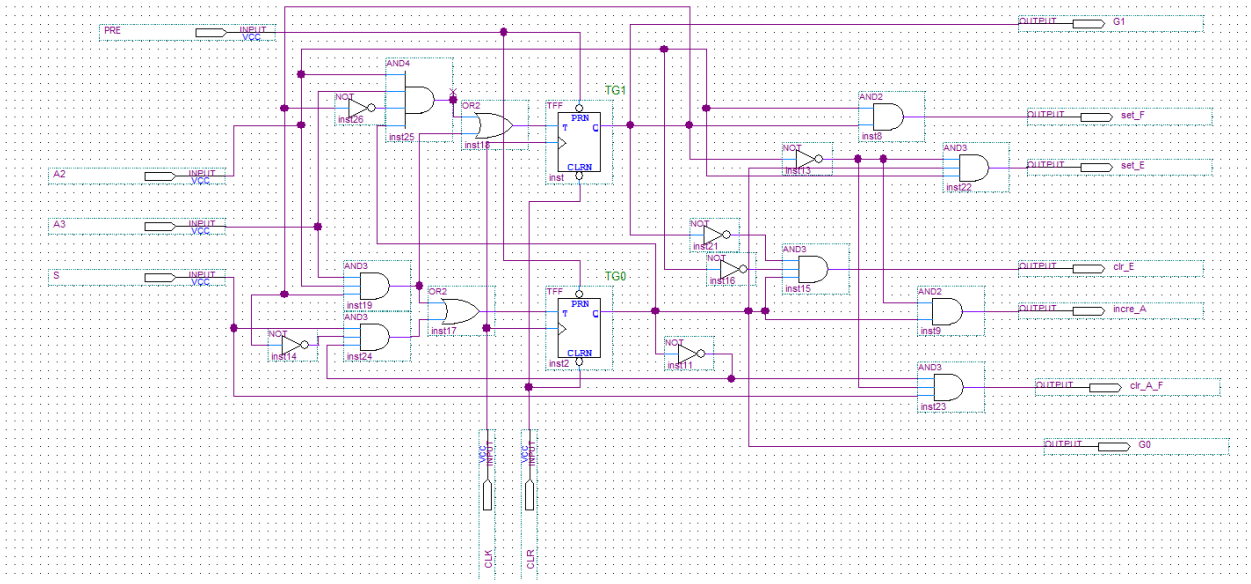
TG1 = G1' · G0 · A2 · A3 + G1 · A2 · A3

TG0

G1=0					G1=1				
G0S/A2A3	00	01	11	10	G0S/A2A3	00	01	11	10
00	0	0	0	0	00	X	X	X	X
01	1	1	1	1	01	X	X	X	X
11	0	0	0	0	11	0	0	1	0
10	0	0	0	0	10	0	0	1	0

TG0 = G1' · G0' · S + G1 · A2 · A3

Circuit



Description

All test cases that we used were based off the state table above, we tested for all possible inputs and changes in binary value. The three variables that we set as counters were S, A2 and A3. The value A2 changed every 10ns, A3 every 20ns and S every 40ns.

Simulation Results