DIGITAL HARDWARE DESIGN LAB (ECP 313)

MINI PROJECT REPORT

Topic: Morse Code Transmitter

Date of submission: 23/04/2024

Group Members: BT21ECE011 Devika Anerao

BT21ECE012 Aishani Prabhu

BT21ECE087 Anjuli Mothe

BT21ECE120 Drishti Diwani

AIM:

To design and implement a Morse Code transmitter on FPGA board.

WORKING:

1. Input Handling: The module takes inputs SW (switches), KEY (buttons), and CLOCK_50 (50MHz clock) and an output LEDR (LEDs).
2. Letter Selection: Based on the input switches SW, the module selects a specific Morse code pattern M for a particular letter whose length is represented by N. This is done using a case statement that maps switch inputs to Morse code patterns.
3. State Transition Conditions:
    - State s1 (Idle State): In this state, the FSM waits for the start button (KEY[1]) to be pressed. If the start button is pressed (!KEY[1]), the FSM transitions to state s2 (State Selection State). If the start button is not pressed, the FSM remains in state s1.
    - State s2 (State Selection State): In this state, the FSM decides whether to output a dot or a dash based on the next symbol in the Morse code pattern. If the next symbol is 0 (indicating a dot), the FSM transitions to state s5 (Output Dot State). If the next symbol is 1 (indicating a dash), the FSM transitions to state s3 (Output Dash State).
    - State s3 (Output Dash State): In this state, the FSM outputs a dash for Morse code. The FSM remains in this state for a predefined duration (0.5 seconds) before transitioning to state s4.
    - State s4 (Intermediate State): This state acts as an intermediate state between generating a dash and transitioning to the next symbol. The FSM remains in this state for a short duration (0.5 seconds) before transitioning to state s5.
    - State s5 (Output Dot State): In this state, the FSM outputs a dot for Morse code. The FSM remains in this state for a predefined duration (0.5 seconds) before transitioning back to state s2 to process the next symbol.
4. Reset Behavior: If the reset button (KEY[0]) is pressed at any state (s2, s3, s4, or s5), the FSM transitions back to the idle state s1, resetting the Morse code generation process.
5. Clock Counter and Timing: The transitions between states are controlled by a clock counter (count) that counts clock cycles. When the counter reaches a specific threshold corresponding to the desired timing (e.g., 0.5 seconds), the FSM transitions to the next state.
6. LED Output Control: LED output (LEDR) is controlled based on the current state of the FSM. LEDs are turned on or off according to the Morse code pattern being generated and the specific states.

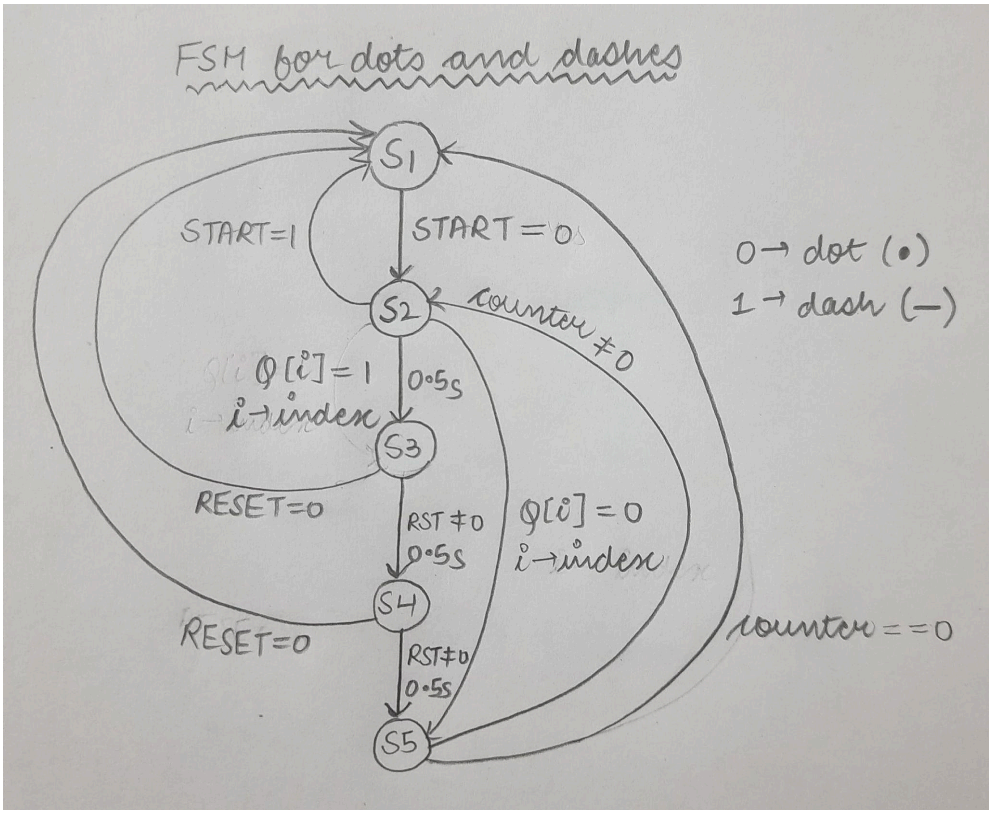
MORSE CODE:

(The symbols along with length of each symbol and the decimal equivalent)

International Morse Code

Decimal eq.	Variable	Morse code	Morse length	Decimal eq.	Variable	Morse Code	Morse length
0	N_0	— — — — —	5	19	J	• — — —	4
1	N_1	• — — — —	5	20	K	— • —	3
2	N_2	• • — — —	5	21	L	• — • •	4
3	N_3	• • • — —	5	22	M	— —	2
4	N_4	• • • • —	5	23	N	— •	2
5	N_5	• • • • •	5	24	O	— — —	3
6	N_6	— • • • •	5	25	P	• — — •	4
7	N_7	— — • • •	5	26	Q	— — • —	4
8	N_8	— — — • •	5	27	R	• — •	3
9	N_9	— — — — •	5	28	S	• • •	3
				21	T	—	1
10	A	• —	2	30	U	• • —	3
11	B	— • • •	4	31	V	• • • —	4
12	C	— • — •	4	32	W	• — —	3
13	D	— • •	3	33	X	— • • —	4
14	E	•	1	34	Y	— • — —	4
15	F	• • — •	4	35	Z	— — • •	4
16	G	— — •	4				
17	H	• • • •	4				
18	I	• •	2				

STATE TRANSITION DIAGRAM:



CODE :

```
//morse code, 1 = dash, 0 = dot
// 1.5 sec led on for dash
// 0.5 sec led on for dot
// 0.5 sec led off between dots and dashes
module morsefinal(SW,KEY,CLOCK_50,LEDR);
input [5:0] SW; // binary input for letter selection
input [1:0] KEY; // Start & Reset button
input CLOCK_50; // 50MHz internal clock
output [0:0] LEDR; // Outputs Morse Code
reg [25:0] count;//counts 50 MHz clock signals to 0.5 seconds
reg [2:0] counter;//length of morse code of a symbol
reg [4:0] M;// to store the morse code of a symbol in binary
reg[2:0]N;//to store the length of a morse code for a symbol
reg [4:0] Q;//shift register outputs, Q[4] is the input to the FSM
reg z;//output to ledr
reg[2:0] ptstate, ntstate;//ptstate represents current state, ntstate represents next state
// Letter representation of SW[2:0] input
```

```
parameter N_0=6'b000000, N_1=6'b000001, N_2=6'b000010,
N_3=6'b000011, N_4=6'b000100, N_5=6'b000101, N_6=6'b0000110,
N_7=6'b000111, N_8=6'b001000, N_9=6'b001001,
A=6'b001010, B=6'b001011, C=6'b001100, D=6'b001101,
E=6'b001110, F=6'b001111, G=6'b010000, H=6'b010001,
I=6'b010010, J=6'b010011, K=6'b010100, L=6'b010101,
M1=6'b010110, N1=6'b010111, O=6'b011000, P=6'b011001,
Q1=6'b011010, R=6'b011011, S=6'b011100, T=6'b011101,
U=6'b011110, V=6'b011111, W=6'b100000, X=6'b100001,
Y=6'b100010, Z=6'b100011;


// 5 States
parameter s1 = 3'b000, s2 = 3'b001, s3 = 3'b010, s4 = 3'b011, s5 = 3'b100;
assign LEDR = z;


always @(SW,N) // anytime symbol selection changes,this block resets N and M
begin
case(SW[5:0])
N_0: begin
N=3'b101;
M = 5'b11111;
end
N_1: begin
N=3'b101;
M = 5'b01111;
end
N_2: begin
N=3'b101;
M = 5'b00111;
end
N_3: begin
N=3'b101;
M = 5'b00011;
end
N_4: begin
N=3'b101;
M = 5'b00001;
end
N_5: begin
N=3'b101;
M = 5'b00000;
end
N_6: begin
N=3'b101;
M = 5'b10000;
end
N_7: begin
N=3'b101;
M = 5'b11000;
end
N_8: begin
N=3'b101;
M = 5'b11100;
end
N_9: begin
N=3'b101;
M = 5'b11110;
end
A: begin
N=3'b010;
M=5'b01xxx;
end
```

```
B:begin
N=3'b100;
M=5'b1000x;
end
C:begin
N=3'b100;
M=5'b1010x;
end
D:begin
N=3'b011;
M=5'b100xx;
end
E:begin
N=3'b001;
M=5'b0xxxx;
end
F:begin
N=3'b100;
M=5'b0010x;
end
G:begin
N=3'b100;
M=5'b110xx;
end
H:begin
N=3'b100;
M=5'b0000x;
end
I:begin
N=3'b010;
M=5'b00xxx;
end
J:begin
N=3'b100;
M=5'b0111x;
end
K:begin
N=3'b011;
M=5'b101xx;
end
L:begin
N=3'b100;
M=5'b0100x;
end
M1:begin
N=3'b010;
M=5'b11xxx;
end
N1:begin
N=3'b010;
M=5'b10xxx;
end
O:begin
N=3'b011;
M=5'b111xx;
end
P:begin
N=3'b100;
M=5'b0110x;
end
Q1:begin
N=3'b100;
M=5'b1101x;
end
R:begin
N=3'b011;
M=5'b010xx;
end
```

```
S:begin
N=3'b011;
M=5'b000xx;
end
T:begin
N=3'b001;
M=5'b1xxxx;
end
U:begin
N=3'b011;
M=5'b001xx;
end
V:begin
N=3'b100;
M=5'b0001x;
end
W:begin
N=3'b011;
M=5'b011xx;
end
X:begin
N=3'b100;
M=5'b1001x;
end
Y:begin
N=3'b100;
M=5'b1011x;
end
Z:begin
N=3'b100;
M=5'b1100x;
end
endcase
end
```

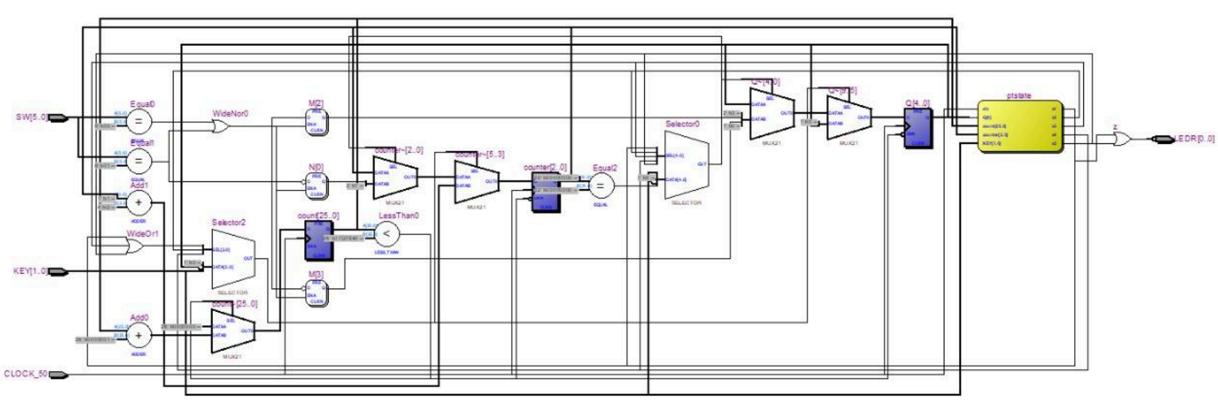
```
// FSM with 5 states :
//State Table - anytime register changes shift output, reset/start is pressed
// State s1 = Idle State
// State s2 = State Selection State
// State s3,s4 = Dash , s5 = Dot
always @(Q[4], KEY[1:0], counter, ptstate)
begin
case (ptstate)
s1: if (!KEY[1]) ntstate = s2; // if start is pressed, goto state s2
        else ntstate = s1; // else remain at state s1
s2: if (!Q[4]) ntstate = s5; // if next Symbol is 0, go to state s5 (outputs 0.5sec)
        else ntstate = s3; // if next Symbol is 1, go tostate s3 (outputs 0.5sec)
// s2 -> s3 -> s4 -> => 1.5 seconds => dash
s3: if (!KEY[0]) ntstate = s1; // as long as reset is pressed, go to state s1
        else ntstate = s4;// else, go to state s4
s4: if (!KEY[0]) ntstate = s1; // as long as reset is pressed, go to state s1
        else ntstate = s5;// else, go to state s5
// s1 -> s5 => 0.5 seconds => dot // the transition turns on LED for 0.5 seconds
s5: if (counter == 0) ntstate = s1; // if counter is 0, no more symbols, go to state s1
        else ntstate = s2;
// else, go to state s2
default: ntstate = 4'bxxxx; // In case of weird behaviour
endcase
end

//clock counter
always @(posedge CLOCK_50)
begin
```

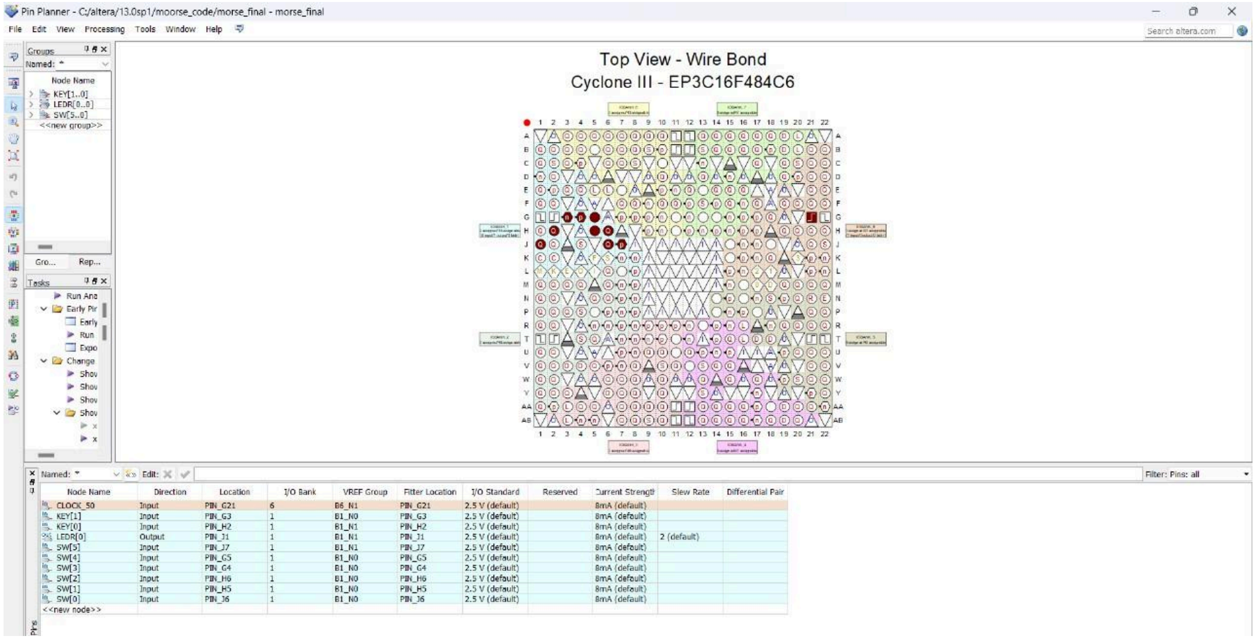
```
if (count < 50000000/2) // at every 0.5 seconds, activate
count <= count + 1;
else
begin
count <= 0;
ptstate <= ntstate; // go to next state
if (ntstate == s1) begin // if next state is s1, update counter to N and pattern to M
counter <=N;
Q <= M;
end
if (ntstate == s5) begin // if state s5
counter <= counter - 1; // deduct counter
// Shift pattern
Q[4] <= Q[3];
Q[3] <= Q[2];
Q[2] <= Q[1];
Q[1] <= Q[0];
Q[0] <= 1'b0;
end
end
end

// LED output based on current state
always @(ptstate)
begin
case (ptstate)
s2: z = 1; // turn on led
s3: z = 1; // turn on led
s4: z = 1; // turn on led
s1: z = 0; // turn off led
s5: z = 0; // turn off led
endcase
end
endmodule
```

RTL VIEW:



PIN PLANNER:



CONCLUSION:

In conclusion, the provided Verilog module effectively implements Morse code generation functionality, utilizing a Finite State Machine for precise control and LED output for visual representation. With its structured approach and clear logic, the module serves as a reliable Morse code generator suitable for various applications