Student 1: Shravya Suresh Roll Number: 210260046

Student 2: Roll Number:

Final project title: Morse Pulse Detector

Final project Executive summary: 10

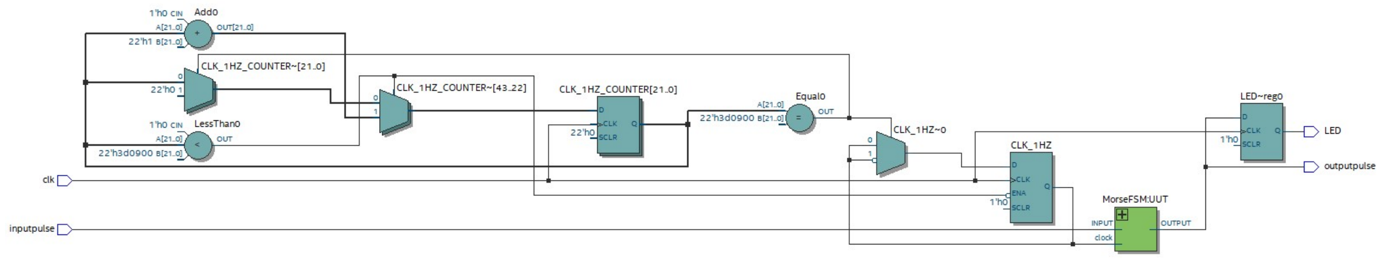
This project simulates the two ‘alphabets’ of Morse Code – a dot and a dash – through a digital pulse displayed using an LED. Input pulses of varying length are sent into the FPGA. A length of 2 two clock cycles represents a ‘dot’ in Morse Code, displayed as a quick blink of the LED. A length of 4 two clock cycles represents a ‘dash in Morse Code, displayed as a longer blink of the LED.

Final project details:

Theoretical design: 50

1. **Block Diagram of the System**

   The Block diagram of the system, as implemented by the FPGA, is:



2. **Analysis of the Design**

   For implementing the above design, I incorporated the following inputs and outputs:

   <u>**Inputs**</u>

   - inputpulse: This is the main input signal to the FPGA system, which helps to determine the resulting output. In the hardware, I implemented this with the help of a push-button switch and the VCC voltage derived from the FPGA itself. I configured the switch such that it allows DC input into the defined Finite State Machine (implemented using the FPGA) only when the switch is pressed. When the switch is not pressed, no current will enter the Finite State Machine.
   - clk: This is the internal clock signal of the FPGA. Hence, this did not have to be externally implemented through the hardware.

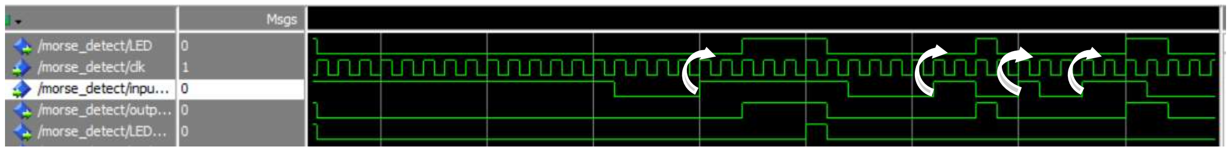
   Signals

   - CLK_1HZ: This helps to redefine the internal clock signal of the FPGA to a signal of 1Hz. This helps in better visibility of the glowing of the LED on the main circuit.
   - CLK_1HZ_COUNTER: This is used to control the redefinition of the internal clock.

   Outputs

   - LED: This is the output signal received from the FPGA after passing through the Finite State Machine. This signal is fed into a grounded LED, which helps to visually perceive the results obtained from the circuit. When the ‘LED’ signal is high, the LED connected to it lights up, while the LED stays off for a low ‘LED’ signal.
   - outputpulse: This performs a similar functionality to the ‘LED’ output signal. It maps onto the ‘LED’ signal in the top-level entity file during program execution.

3. **Simulated Timing Diagram**
   Below is the simulated Timing Diagram of the system, with cause and effect annotations.



4. **Displayed Outputs**
   The displayed output is the glowing of the LED on the main circuit.
   In the FSM state of a 'dot', the LED glows for a span of 1 second, representing the 'dot' of Morse Code.
   In the FSM state of a 'dash', the LED glows for a span of 3 seconds, representing the 'dash' of Morse Code.
   In all other states, the LED remains off.

Experiment Implementation: 40

In the demonstration of this project, I aim to show the following:
1. Initially, the LED remains OFF. The switch remains unpressed since the FPGA does not receive any input signal.
2. When the push-button is pressed, the circuit is completed and an input signal from the 3.3V pin of the FPGA is fed into the input pin of the FPGA. This represents the input into the Finite State Machine.
3. If the push button is pressed for uptil 0.5 milliseconds, the LED will not glow. This is because a buffer 'noise' state was incorporated to account for any faulty or unwanted input signals sent to the FPGA.
4. If the push button is pressed for uptil 1 millisecond, the system reaches the 'dot' state. Here. If the input remains HIGH, then the LED will not glow. But if the input is LOW, then the LED will glow to represent the 'dot' of Morse Code.
5. After completing 1.5 milliseconds with the push button pressed down, the LED will glow for a span of 3 seconds, indicating a 'dash' of Morse Code.
6. After this, the system is reset and the process can start again.

Connection with Computer Science and Physics 50 bonus marks

Morse Code was a revolutionary creation bsack in its time. It signaled the first time in human history that complex thoughts could be communicated at long distances almost instantaneously. The reference to letter frequency makes for extremely efficient communications: Simple words with common letters can be transmitted very quickly. Longer words can still be sent, but they take more time.

Morse Code still finds wide applications today, especially in communication technologies such as ham radio communication. Amateur radio operators still use Morse Code, as do a few other government services, such as aviation beacons, etc.

In Computer Science, Morse Code is highly relevant as a representation of a binary code. It is also widely utilized in cryptography. An important application is signaling for help using SOS, "· · · — — — · · ·". This can be sent many ways: keying a radio on and off, flashing a mirror, toggling a flashlight, and similar other methods. Morse code has been employed as an assistive technology, helping people with a variety of disabilities to communicate. Morse code is very useful to people with severe motion disabilities, as long as they have some minimal motor control. Morse code can also be translated by computer and used in a speaking communication aid.

I conclude with a fun fact: Morse Code day falls on April 27. As we look forward to the day, let's all learn

'_ _   _ _ _   . _ .   . . .   .' Code!

Appendix:

VHDL Code:

MorseFSM.vhd

```
-- include libraries

LIBRARY IEEE;

USE ieee.std_logic_1164.all;


ENTITY MorseFSM IS

PORT (

      clock :     IN STD_LOGIC;

      INPUT :     IN STD_LOGIC;

      OUTPUT:     OUT STD_LOGIC := '0');

END ENTITY;


-- Architecture definition for the MorseFSM entity

ARCHITECTURE RTL OF MorseFSM IS

TYPE State_type IS (O, N, DS, S, DA, L1, L2, L3);  -- Define the states

SIGNAL State : State_Type;    -- Create a signal that uses the different states


BEGIN


      --Input Process

      PROCESS (clock)

      BEGIN

      IF rising_edge(clock) THEN    -- if there is a rising edge of the clock,
then do the following


            -- Based on the value of the state variables and any other control
signals, changes to a new state.

            CASE State IS


                  WHEN O =>

                        OUTPUT <= '0';


                        IF INPUT='1' THEN

                              State <= N;


                        ELSIF INPUT='0' then
                              State <= O;

                        END IF;

                  WHEN N =>
                        OUTPUT <= '0';

                        IF INPUT='1' THEN
                              State <= DS;

                        ELSIF INPUT='0' THEN
                              State <= O;

                        END IF;

                  WHEN DS =>
                        OUTPUT<= '0';
                        IF INPUT='1' THEN
                              State <= DA;

                        ELSIF INPUT='0' THEN
                              State <= S;

                        END IF;
                  WHEN S=>
                        OUTPUT  <= '1';
                        STATE <= O;

                  WHEN DA=>
                        OUTPUT <= '1';
                        STATE <= L1;

                  WHEN L1=>
                        OUTPUT <= '1';
                        STATE <= L2;

                  WHEN L2=>
                        OUTPUT <= '1';
```

```
                        STATE <= L3;

                  WHEN L3=>
                        OUTPUT <= '1';
                        STATE <= O;

                  WHEN OTHERS => -- Invalid state
                        OUTPUT <= '0';
                        State <= O;

            END CASE;
    END IF;
      END PROCESS;

END RTL;
```

Morse_Detect.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Morse_Detect IS
PORT(LED: BUFFER STD_LOGIC;
            clk           :   IN STD_LOGIC;
      inputpulse :      IN STD_LOGIC;
      outputpulse:      BUFFER STD_LOGIC);
END Morse_Detect;

ARCHITECTURE Detect_Pulse OF Morse_Detect IS
      COMPONENT MorseFSM
   PORT (clock :  IN STD_LOGIC;
                  INPUT :     IN STD_LOGIC;
                  OUTPUT:     OUT STD_LOGIC);
      END COMPONENT;

      SIGNAL CLK_1HZ : STD_LOGIC;
      SIGNAL CLK_1HZ_COUNTER : NATURAL RANGE 0 TO 4000000;

      BEGIN
      PROCESS(CLK)
      BEGIN
            IF RISING_EDGE(CLK) THEN

                  LED <= outputpulse;

                  IF CLK_1HZ_COUNTER < 4000000 THEN
                        CLK_1HZ_COUNTER <= CLK_1HZ_COUNTER + 1;
                  ELSIF CLK_1HZ_COUNTER = 4000000 THEN
                        CLK_1HZ <= NOT CLK_1HZ;
                        CLK_1HZ_COUNTER <= 0;
                  END IF;
            END IF;
      END PROCESS;

      UUT:  MorseFSM
      PORT MAP (clock => CLK_1HZ, INPUT => inputpulse, OUTPUT => outputpulse);

END Detect_Pulse;
```