EE310 – LAB #2

Ping Pong Game

Alperen Yaşar / 24834

Engincan Varan / 25050

Introduction

In this lab assignment we were supposed to design and implement a simple ping pong game. At start-up, two players will be able to enter their 3-character names through 4-bit switch inputs. When they are ready, game will start, with the ball starting from the 4th led, moving towards right. When it reaches to the end, corresponding players should press the button. If a player presses too early or late, game will end and a point will be given to the other player. There will be a total of five rounds, and depending on their scores, the winner will be displayed at the end.

A) FSM Approach

We've started designing by building a finite state machine. There will be 5 main stages: initial state, name entrance state(s), round state, pause state and endgame state. As we will take 6 characters (48 bits) at total from users, and we have only 4 switches, there will be 12 states for this purpose. At each state, players will be able to input 4 bit of their next character. Then, machine will move to round state, where the game begins. Round state will return to itself 5 times, and then move to the ending state, where the scores will be compared, and winner will be announced. Between each of these rounds, there will be a pause state, where players can wait as much as they want before starting the next round.

1) Initial State

In this state we'll just update the screen and wait for the user to press the north button. *Ping Pong Game* will be written to the screen. Once the player presses the north button, it will move into the player name entry states. After each reset or endgame, the machine will move into this initial state.

2) Player Name Entry State(s)

These are the 12 states that players will enter their 3-character names by 4 bits each time. At first, the screen will look like this:

XXX : 0
XXX : 0

As players keep on entering their names, X's will update.

ALX : 0
XXX : 0

At each of these states, machine will wait for north button to be pressed. Once it is pressed, it will save the current situation the switches and move into the next state. Until north button is

pressed, switching the switches will not change anything. After the last character input is provided, machine will move into the pause state before starting the round directly.

3) Pause State

There is nothing much happening in this state. Machine just holds every information as it is and waits for the player to press the north button. In this state, we are lighting up all LED's to understand that we are in pause state. Once the player presses the north button, it will move into the round state.

4) Round State

This is the main state where everything happens. We'll cycle through this state 5 times, as we were supposed to play a total of 5 rounds before announcing the winner. Before each round state, we have a round initializer state, where we only set the 4^{th} LED on, and start the timer. There is nothing complex in that state, and the timer will be introduced later. This round initializer state will be called before every round state. In the round state, we are supposed to move the ball to the right or left, whichever is necessary, each 0.25 seconds. We've defined an 8 bit counter and initialized it as 00010000, in the round initial state. Instead of counting bit-wise, we're just right shifting, or left shifting the number, in the sequential part. In combinational part, we're just parsing this data to LED's in order, where the only 1 in the sequence will be the lighting LED. This shift will happen with a periodicity of 0.25 seconds, which will be handled by the timer in timer module. Direction of the movement is stored in a single bit number, where 0 means it should do a left shift and 1 means a right shift.

At each clock, machine will check whether counter is 10000000 or 00000001, to see if any player should press their button. If this is the case, a flag named press_now will be raised to 1 (for player 1) or 2 (for player 2). If this is not the case, press_now flag will remain 0. This flag is done for a specific purpose. If we did not use it, player would have to press in the same clock cycle where machine have checked for 10000000 or 00000001. This way, we are letting the player a time lapse of 0.25 seconds to press the button. If press_now is not 0, and the counter is no more 10000000 or 00000001, this means one of the players had to press, and missed their opportunity, as timer once more raised and ball has moved. In that case, round has ended, and whichever player missed the ball (which is stored in press_now as 1 or 2) has lost the round. Machine will move into the check state as the next state.

There is another way a player can lose the game, and it is much easier to check thanks to the press_now flag implemented before. If a player presses the button too soon, the other player will score a point. We are basically checking this by looking press_now flag. If a button is pressed, but the flag is not set to that player, then that player will lose. I.e. if player 1 presses the button, but press_now is 0 or 2, this means player 1 pressed early and lost the round. In that case, it will again move into the check state.

Meanwhile all of these are happening, the LCD will stay stable, showing the current situation such as:

ALP : 2
ENG : 1

5) Check State

In this state, we are just updating the LCD screen with the new scores. Scores are being updated at the end of the round state. We're checking the sum of the scores to see if fifth round has finished. If it is, once the north button is pressed, it moves into the finish state, where winner is announced. Else, it moves back to the pause state, to start another round from beginning.

6) Finish State

In this state, we are determining the winner and announcing it to the screen. It looks like

ALP : 3 WINNER!!
ENG : 2

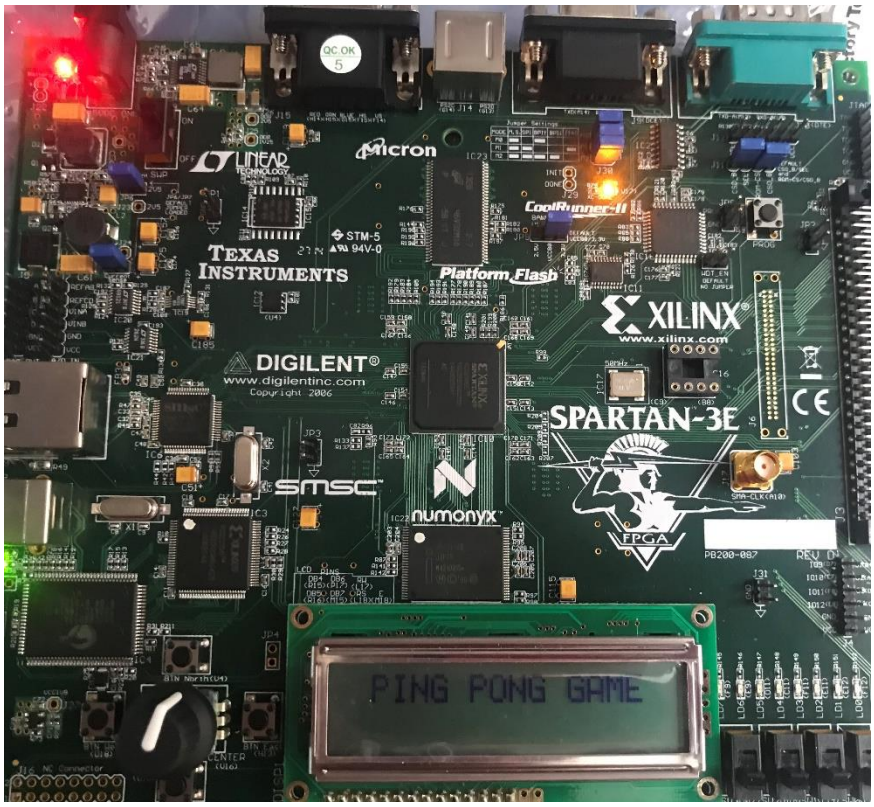
Once north button is pressed, it moves into the idle state, the very beginning of FSM. As a little easter egg, if once of the players reach a score of 5, there is a intermediate state between finish and idle, where we display *WINNER WINNER CHICKEN DINNER!* to the screen. This state also returns to the beginning of the fsm when north button is pressed.

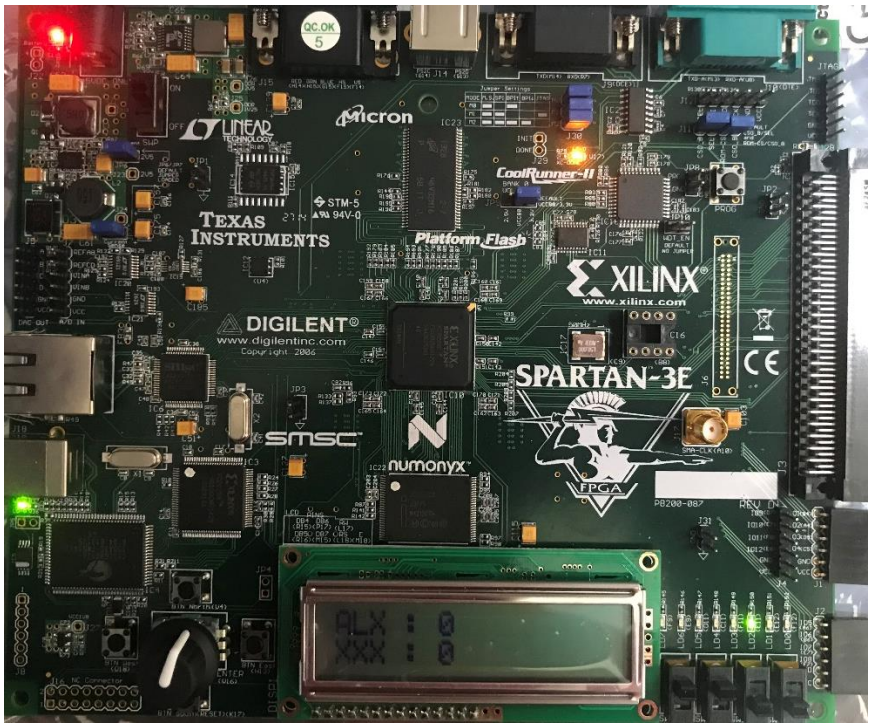
B) Implementation of Timer Module

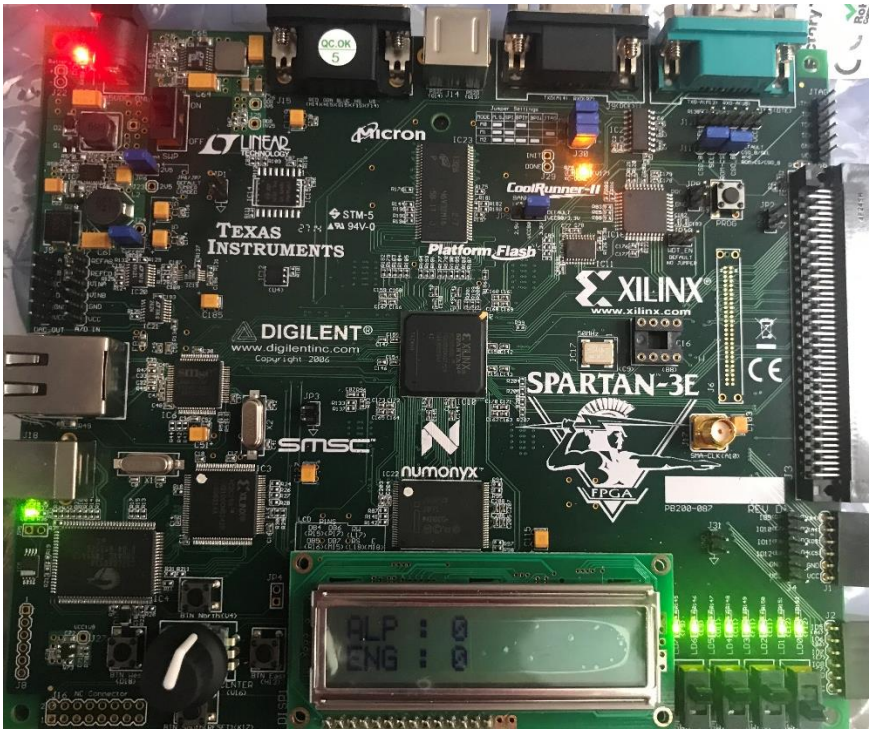
Our FPGA board works with 50 MHZ clock frequency. This means, at 0.25 seconds there will be 12.5 M clock cycles. Keeping this in mind we can create a timer based on a simple counter. We've created a register of 24 bits, where we count one-by-one at each positive edge of clock. When the count is equal to 12.5 million, this means 0.25 seconds have passed. The module has three inputs: reset, run and clock. When necessary, for instance in pause states, we can reset and pause the timer to start it at round initializer state. It has a single done output, which raises when 12.5 million is reached and counter is returned back to zero. It stays as high for one clock cycle, and during this period, top module handles the shift operation of LED counter.

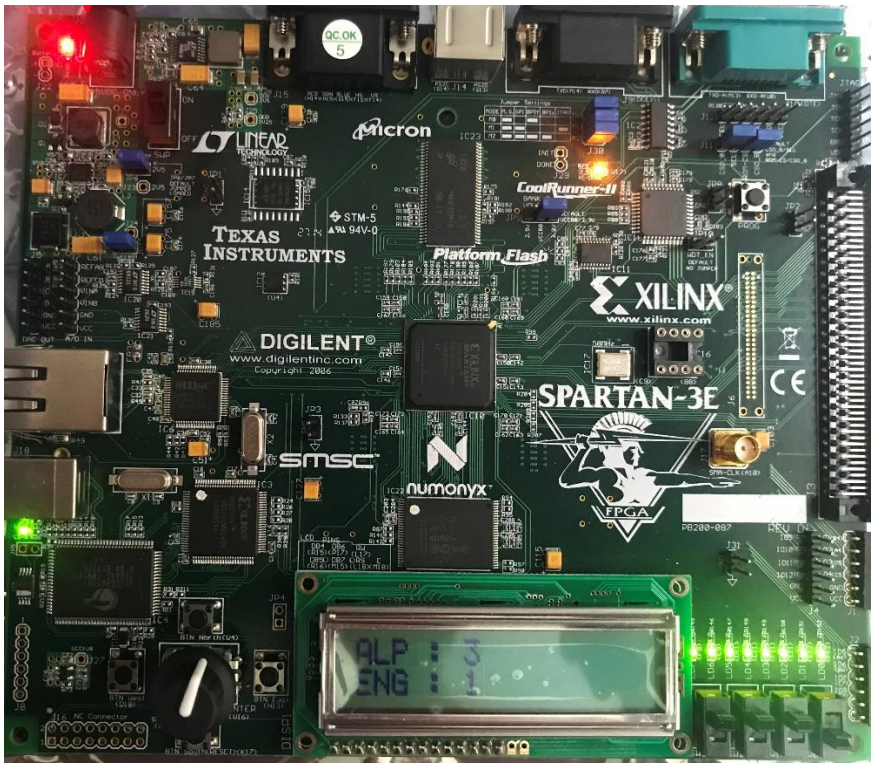
C) Testing the Circuit

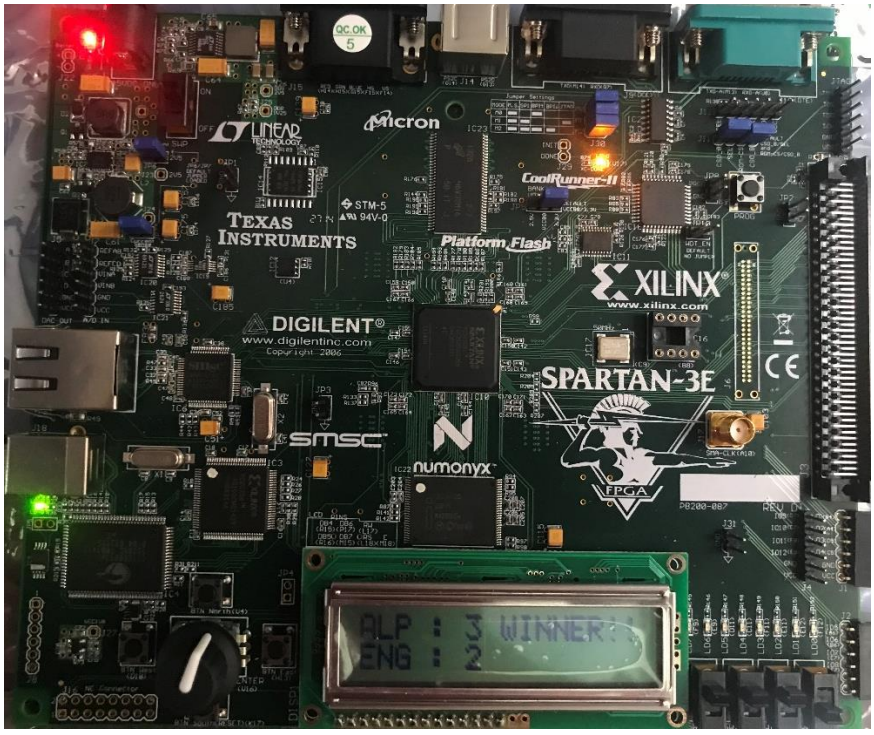
We've divided our testing into two. For state transitions and combinational stuff, we've used testbench to see whether it works correctly. However, implementation of the game was easier to check and debug on the FPGA board directly. So, once we were sure our FSM worked correctly (at least its state transitions), we've moved on by implementing the Verilog code to the FPGA board. It was much easier to debug LCD or LED related problems directly on the FPGA board. We've tried to find some edge cases and try them to see what happens, and if any bug occurs tried to solve it. Results on the board can be seen below. It is working properly.




QC.OK
5
Micron
LINEAR TECHNOLOGY
CoolRunner-II
STM-5
94V-0
TEXAS INSTRUMENTS
Platform Flash
XILINX
www.xilinx.com
DIGILENT®
www.digilentinc.com
Copyright 2006
SPARTAN-3E
FPGA
smsc
numonyx
ALX : 0
XXX : 0


QC.OK
5
Micron
LINEAR TECHNOLOGY
CoolRunner-II
STM-5
94V-0
TEXAS INSTRUMENTS
Platform Flash
XILINX
www.xilinx.com
DIGILENT®
www.digilentinc.com
Copyright 2006
SPARTAN-3E
FPGA
smsc
numonyx
ALP : 0
ENG : 0


QC.OK
5
Micron
LINEAR TECHNOLOGY
CoolRunner-II
TEXAS INSTRUMENTS
STM-5
94V-0
Platform Flash
XILINX
www.xilinx.com
DIGILENT
www.digilentinc.com
Copyright 2006
SPARTAN-3E
FPGA
smsc
numonyx
ALP : 3
ENG : 1


QC.OK
5
Micron
LINEAR TECHNOLOGY
CoolRunner-II
TEXAS INSTRUMENTS
STM-5
94V-0
Platform Flash
XILINX
www.xilinx.com
DIGILENT
www.digilentinc.com
Copyright 2006
SPARTAN-3E
FPGA
smsc
numonyx
ALP : 3 WINNER!
ENG : 2

D) Conclusion

In general, this lab was not very difficult to design with pen and paper. Difficult thing with this lab was debugging some major issues due to some minor mistakes. We had to spend hours to implement the circuit on the board and check every edge case we can imagine, then detect the errors and try to solve it. However, it was much more enjoyable than the last lab as it is something that you can see on the board directly. In next labs, due to the current situation, it is told that we'll focus more on testbench simulation. However, I believe, at least I hope, each group has a FPGA board at home (if they didn't leave it at dormitories) and it would be better if we continue with projects that we can also implement on the board and see the results directly.