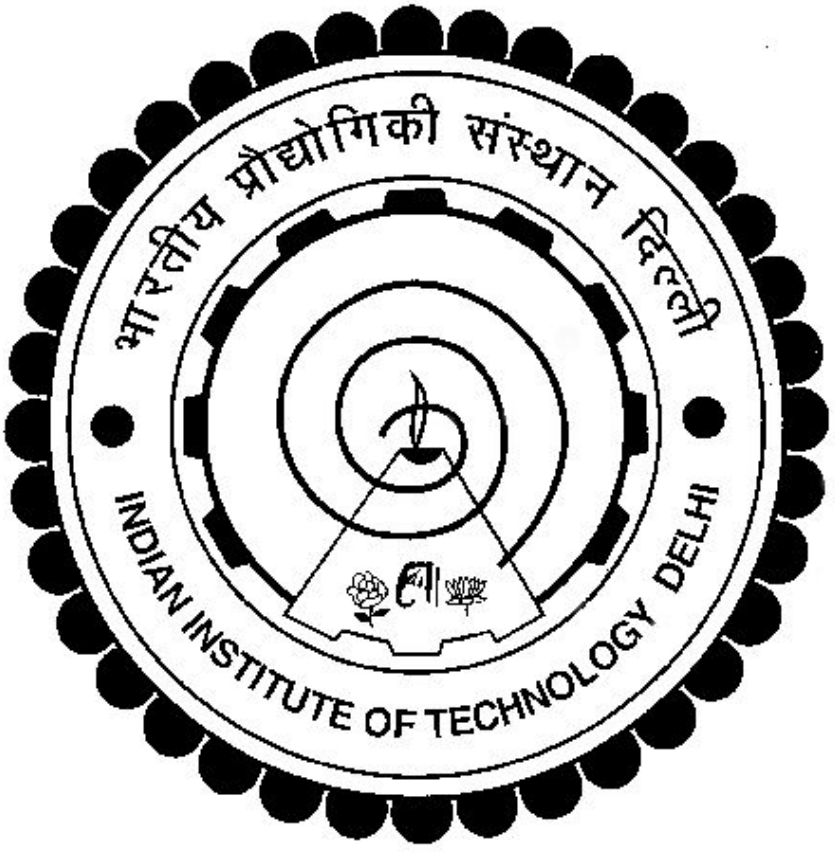
COL-215

MINI PROJECT

PING PONG GAME



S.No	Name	Entry No.
1	Samarth Aggarwal	2016CS10395
2	Ayush Patel	2016CS10396

Abstract

As the name suggests, this project involves the making of a two player “Ping - Pong Game” which is to be played on a ‘Field Programmable Gate Array’. A ping pong game is widely known as table tennis and is popularly played as an indoor sport. The implementation of this game on an FPGA board not only provides a space-efficient way to play the game requiring almost no equipment other than the FPGA board but also provides an intriguing application of VHDL designing to solve real world problems.

Specification

The project involves designing and implementation of “Ping – Pong Game” which is a two player game to be played on the FPGA basys board. The game involves a ball denoted by a lit LED that is returned by each player as it nears their side. The aim of this game is to defend one’s own goal post and score goals against the opponent by returning the ball as it approaches one’s goal post. The game ends as soon as one of thee player scores 9 goals against the other.

The 16 LEDs available on the basys board are used to display the position of the ball. Each player gets two push buttons to enter his requests. One of the pushbutton is used to return the ball when it has reached the end LED on his/her side. This button must be pressed only when the ball has has reached the last LED else a goal is scored. The other pushbutton is for telling the speed with which the ball is to be returned. The press of this button will toggle the speed with which the ball will be returned that is it dictates whether the ball is to be returned with a higher speed or with slower speed. Another button called reset has been introduced to set the initial values whenever a new game is to be started.

Overall Approach

Since the press of a pushbutton is not recorded instantly and the circuit may bounce a few times before settling to the final value, hence we will use debouncing logic for the push buttons. For the debouncing logic, we will use another entity since many inputs have to be debounced so by creating another entity we can reuse the same code. The debouncing logic will involve sampling the input of the pushbuttons by a slow clock so that the bouncing of the input does not reflect in the output. The debouncing logic has to applied to four buttons - two return buttons and two speed selectors.

Reset need not be debounced since a bouncing reset button will indeed lead to the same output as a debounced one. Moreover adding a component to debounce the reset button will simply increase the hardware requirements of the circuit without any additional utility.

The reset button will work asynchronously so as to override the game at any stage and take it to the initial state of a new game.

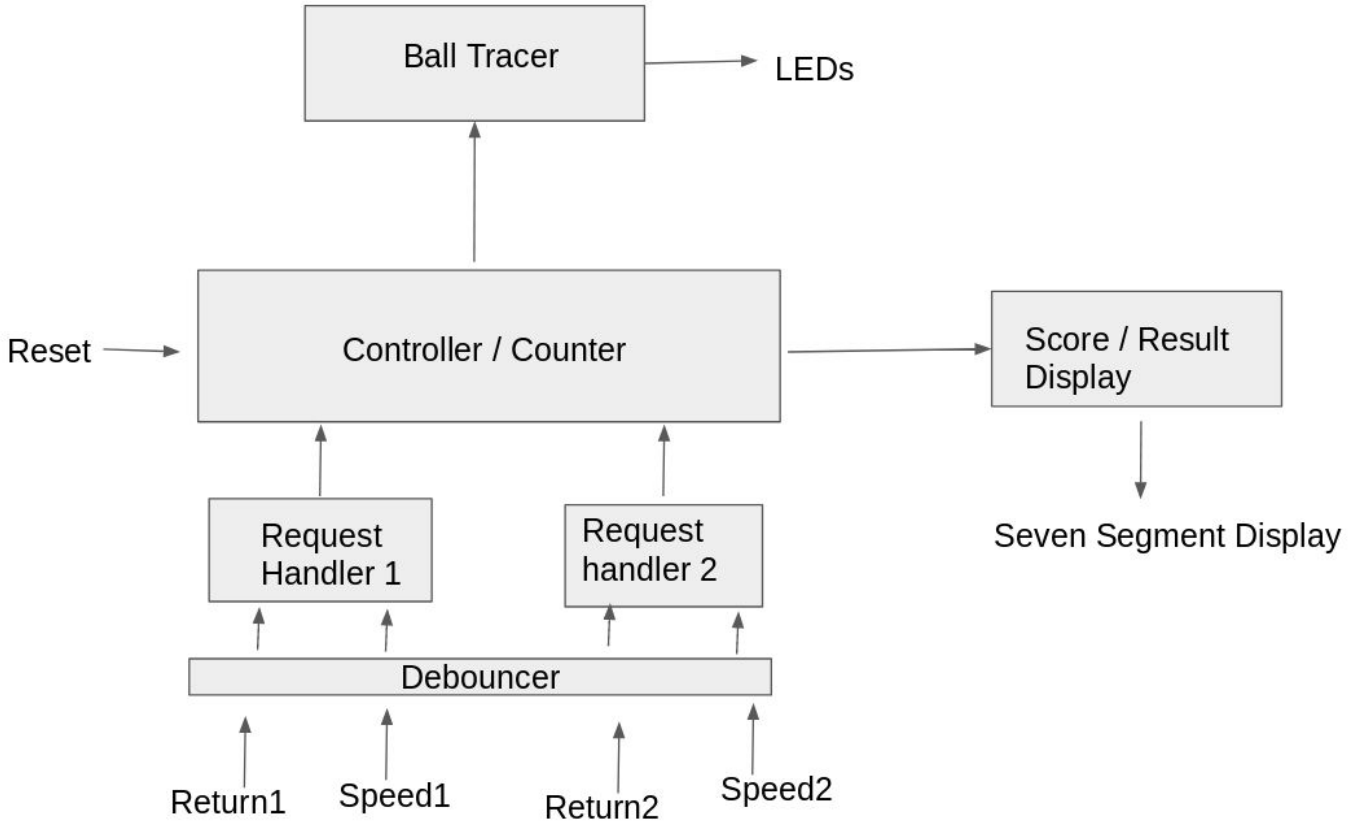
The time for which the ball remains at each of the 16 LED will be decided at the time of testing of the game since it has to be calibrated keeping in mind the ease of play, physical restrictions, level of difficulty etc. Similarly, the speeds for

low and high speed modes will also be decided at the time of testing.

Since two digits corresponding to the respective scores of the players as well as 2 letters corresponding to the return speed of the players have to be displayed on the seven segment display, hence the whole display will be required.

At the end of the game, 2 digits will depict the final score of the players while the rest of the 2 digits will denote which player has won the game.

Block Diagram Containing Major Subsystems



The above block diagram represents the major subsystems, their respective inputs and output fields and their interaction with other subsystems. Following is the description of the major subsystems:

1) Debouncer - It takes the input from the pushbuttons. The input to the request handler is not given directly from the buttons rather the output of the debouncing logic is fed into the request handler. This is done to prevent the recurring

making and breaking of the circuit which is the root cause of why debouncing is needed. Note that reset has not been debounced and is directly added used as an input to the controller since the output is same for the bounced and debounced reset.

2) Request Handlers – They take the requests from the debouncer and pass them on to the controller. It basically consists of that part of the system which is responsible for registering the inputs.

3) Controller – This is the main component which processes the game. At first, it is overridden asynchronously by the reset button which initialises the game with zero scores of the two players and the ball in the middle going towards the left by default.

Then it processes the pushbutton requests and acts accordingly. Hence it checks whether the players have pressed the buttons at the right time or not. If yes, it turns the ball in the other direction and if no, it updates the score of the opponent player and continues the game.

Also, the speed with which the player has decided to return the ball is taken into consideration and accordingly the time in which the ball crosses each LED is decided. At the same time, it gives the output to the LEDs in order to display the position of the ball. Also, it maintains a track of the scoreboard and gives the output to the seven segment display accordingly.

Finally, it is the controller that decides that when should the game be stopped. By default, when either of the player reaches a score of 9, they is declared as the winner. However, this can be modified to suit different applications.

4) Ball Tracer – It involves the LEDs that display the current position of the ball. It takes input from the controller and lights up one of the LED at a time.

4) Score/Result Display – It refers to the seven segment display which displays the current score and also tell the player that won at the end of the game. Also, it denotes the speed with which the ball is to be returned by each player by the letters - ‘L’ and ‘H’ denoting low and high speeds respectively. At the end of the game, the final scores remain on the display while the player speeds are replaced by the number representing the winner.

Coding the Game

Week 1:
The primary focus for this week was to come up with an the design and functioning of the game. But before that, clarity over what was to be done was of utmost importance. Hence, we jotted down the various assumptions that we required to make this project. We also thought of how should the system respond in different situations. Based on this we came up with a preliminary block diagram depicting the different subsystems that we expected to appear in the final project. At the same time we were open to changes in the plan as the need arises. We also thought that we would require only two entities. One for the display using the seven segment display and the other for the rest of the game.

Week 2:
This week was meant to start make the basic structure of the game by populating the architecture of the two entities with VHDL code and getting the game to work. At first, we wrote the code for traversal of the ball between the two extremes and returning the ball at the two ends. Independently, we developed the display entity so that it could later be integrated into the main entity. The display was successfully completed and was tested on the basys board as well. At the end of this week, we had the completed and tested entity for display as well as major portions of the main entity were ready. This included the code for reset and initialisation of the game, traversal of the ball, etc.

Week 3:
The priority for this week was to complete the coding aspect of the project from end to end so that the only thing left over would be testing and debugging. We realised that since we are dealing with pushbuttons hence their input had to be debounced. So another entity has been added corresponding to the debouncing logic so as to facilitate reuse of the same code. Hence, debounce entity was made. Also, all the essential part of the controller and request handler have also been coded. Moreover, the debouncing logic has been integrated with the main entity. However, the display entity was yet to be integrated with the rest of the code.

Week 4:
This goal for this week was to integrate whatever subsystems were left apart until then. Hence, integration of the display with the ping pong entity marked the start of this week. This was followed by extensive testing of the game. Also, several aspects of the game such as propagation speed of the ball had to be decided at the time of testing since they cannot be assigned arbitrarily and one has to consider the level of difficulty, physical restrictions and several other factors while fixing them. The final testing marked the end of this week.

Following is the description of the entity and architecture(containing declaration of signals) that we have used as a part of our VHDL code.

Entities

```
entity ping_pong is
Port ( return1 : in STD_LOGIC;
return2 : in STD_LOGIC;
speed1 : in STD_LOGIC;
speed2 : in STD_LOGIC;
reset : in STD_LOGIC;
clk : in STD_LOGIC;
anode : out STD_LOGIC_VECTOR (3 downto 0);
cathode : out STD_LOGIC_VECTOR (6 downto 0);
led : out STD_LOGIC_VECTOR (15 downto 0));
end ping_pong;

entity debounce is
Port (
CLK : in STD_LOGIC;
x : in STD_LOGIC;
DBx : out STD_LOGIC
);
end debounce;
entity display is
        Port ( b : in STD_LOGIC_VECTOR (15 downto 0) :=(others
=> '0');
        clk : in STD_LOGIC :='0';
         cathode : out STD_LOGIC_VECTOR (6 downto 0)
:=(others => '0');
        anode : out STD_LOGIC_VECTOR (3 downto 0) :=(others
=> '0'));

end display;
```

Architectures

```
architecture Behavioral of ping_pong is
signal count : INTEGER :=0;
signal led_temp : STD_LOGIC_VECTOR (15 downto 0) :=(others
=> '0');
signal dir : STD_LOGIC :='0';
signal movel : STD_LOGIC :='0';
signal mover : STD_LOGIC :='0';
signal speed : STD_LOGIC :='0';
```

```
architecture Behavioral of debounce is
type State_Type is (S0, S1);
signal State : State_Type := S0;
signal DPB, SPB : STD_LOGIC;
signal DReg : STD_LOGIC_VECTOR (7 downto 0);
```

```
architecture Behavioral of display is
signal count : INTEGER :=0;
signal count2 : INTEGER :=0;
signal cat0 : STD_LOGIC_VECTOR (6 downto 0) :=(others => '1');
signal cat : STD_LOGIC_VECTOR (6 downto 0) :=(others => '1');
signal cat1 : STD_LOGIC_VECTOR (6 downto 0) :=(others => '1');
signal cat2 : STD_LOGIC_VECTOR (6 downto 0) :=(others => '1');
signal cat3 : STD_LOGIC_VECTOR (6 downto 0) :=(others => '1');
signal an : STD_LOGIC_VECTOR (3 downto 0) :=(others => '1');
```

Testing and Validation Mechanism

The display has been tested independent of the main system on the basys board as well as using a test bench. We have specially made a test bench for testing the seven segment display. The glitches found were repaired and now the display seems to work perfectly. Moreover, the display was tested for perfection before and even after integration with the main entity - ping pong.

Next, we tested the debounce entity independently on the basys board. After it was devoid of any imperfection, we went ahead with testing of the game as a whole.

We tested our game by creating another test bench for testing of the whole game. This even tested the display again. After testing it for a variety of cases and repairing the glitches found, we tested the game on the basys board. We believe that the best way to test a game is by playing it hence we repeatedly played the game until we were sure of its perfection.

The completion of the testing phase of the game as a whole marked the successful completion of our project.