ELEC 204

FALL 2020

DIGITAL DESIGN PROJECT

Instructor: Mehmet Cengiz Onbaşlı

Project Name: Even/Odd Game

Mahmut Alpaycı

ID: 68655

Nesim Şişik

ID: 69724

1. How Project Intended to Work

In this project, we wanted to design an even / odd game. In this game, while the counter in the middle of the seven segment controls the game time, random numbers appear on the right and left sides of the seven segment. There are two players on the right and left side of the game. 2 buttons were given to both players. Different numbers are shown to both players and they are asked to know whether the number shown is odd or even by pressing their odd or even buttons. Game time is 60 seconds. Each player gets 1 point for the number he knows correctly. The player who knows the most numbers correctly after 60 seconds wins the game.

We wanted to design a timer for game time. In the counter, we used two 4-bit numbers representing the tens and ones of a second. The number in the tens digit decreases by one step from 5 backwards. The number in the ones place also decreases with each step from 9 backwards.

We wanted to use the Linear Feedback Shift Register methodology to generate numbers in our code. We XOR the 0th and 1st bits of the 4-bit numbers, write the result instead of the Most significant bit and delete the least significant bit. We wanted to generate new random numbers by shifting in this way.

We checked the least significant bit to check whether the resulting random numbers are even or odd. If LSB 1 means odd, 0 means even.

We have specified separate process for all button inputs. If all transactions were happening at the rising edge of the Master Clock, there would be no time between displaying the random number and pressing the button.

We wanted to compare the scores we held after the game time ended. Finally, we assign this comparison result to the output that determines the winner.

1. Functional Units

Since we do not have a board, we did not use the reference units required to implement the Seven Segment (decoder.vhd, drivwe.vhd etc.). We wrote our project on a single VHD code named ProjectCode. Each signal and architecture is specified in a Single VHD unit. Finally, we have another simulation unit called ProjectSimulation that we created for simulation. All components included in this unit come from our ProjectCode main VHD code.

1. Purpose of the Signals

RandomLeft: signal that the new random number for left player is generated

RandomRight: signal that the new random number for left player is generated

RandMsbRight: signal that keeps the new random bit for right number

RandMsbLeft: signal that keeps the new random bit for left number

LeftNumber: signal that keeps left number

RightNumber: signal that keeps right number

ScoreLeft1: signal which keeps record of successful odd choices for left player

ScoreRight1: signal which keeps record of successful odd choices for right player

ScoreLeft2: signal which keeps record of successful even choices for left player

ScoreRight2: signal which keeps record of successful even choices for right player

IsOver: signal which indicates if the countdown is over and when it is '1' it prevents

1. Explanation of How It Works

Firstly, for game we needed a countdown which starts from 59 and when it comes to 00 the game should be ended. To create the countdown, we defined 2 numbers one for left number and the other is right number of the countdown. In other words, they can be represented as units’ digits (right one) and the tens digit (left number). In every clock rising edge I subtract ‘1’ from the right number and when it is come to ‘0’ I make it value to the ‘9’ to maintain countdown. For the left number I defined a variable which has range 0 to 10. Then I incremented that variable in every clock cycle and when it reached 10, I subtract ‘1’ from the left number. Because when the right number changes for 10-time left number should be changed. For example, 40,39,38, 37…31,30,29. And after reaching 10 I assigned 0 to that variable (CounterLeft). To create an even-odd game we needed 2 random numbers for players. By using a Linear-feedback shift register method we created 2 different numbers. Firstly, we stated two numbers which are RandomLeft and RandomRight then we used XOR function to create a random bit then we shifted our number to the right and assigned it MSB to a variable that we created randomly by using the XOR function. But when the countdown is over, I use the variable “over” to stop the alteration of numbers. Then to create a choice mechanism we created 4 inputs which are PRO, PRE, PLO, PLE. They are an odd choice for right player, even choice for right player, odd choice for left player and even choice for left player, respectively. We used process function and rising\_edge function to detect to increment. When the button pressed, we checked the required conditions which are IsOver signal and last bit (LSB) of the random number corresponding to player. If IsOver is equal to 1 than player cannot increment the score because IsOver is equal to 1 when the countdown is over. To increment score when the countdown is not over players should press the correct button according to their number. If the number is odd in other words LSB of the number is 1 they should press the button which are corresponds to odd choice (PRO, PLO) and if the number is even which means LSB of the number is equal to 0 players should press the button corresponds to the even choice (PRE, PLE) to increment score. When the countdown is over, we checked the scores of both players. If the score of the left player is more than right player, we made output W to “10” which indicates player 2 won which represent the player left. If the score of the right player is more than left player, we made output W to “01” which indicates player 1 won which represent the player right. If their score is equal, then we assigned “00” value to the W which indicates draw situation.

In the first 19 lines, there are comments containing project information. The libraries we use from the 20th line are listed. We have defined the inputs and outputs of our project VHDL code with the Line 34 entity section. Line 35 is the line in which the integer N defined for Clock is assigned as 10. The lines between 36th and 47th are the lines where we define the inputs and outputs.

Line 37: 4-bit Time\_Left; output for the left number of the countdown

Line 38: 4- bit Time\_Right; output for the right number of the countdown

Line 39: 4-bit RandNumLeft; output for the random number generated for player left

Line 40: 4-bit RandNumRight; output for the random number generated for player right

Line 41: 4-bit LeftScore; output for the score of the player left

Line 42: 4-bit RightScore; output for the score of the player right

Line 43: 1-bit PRE; button input for even choice for player right

Line 44: 1-bit PRO; button input for odd choice for player right

Line 45: 1-bit PLE; button input for even choice for player left

Line 46: 1-bit PLO; button input for odd choice for player left

Line 47: 2-bit W; output which indicates the results when the countdown is over

Entity node ends at line 48.

From line 51 to line 62, we have defined intermediate signals in the architecture section.

Line 52 -) RandomLeft: signal that the new random number for left player is generated

Line 53 -) RandomRight: signal that the new random number for left player is generated

Line 54 -) RandMsbRight: signal that keeps the new random bit for right number

Line 55 -) RandMsbLeft: signal that keeps the new random bit for left number

Line 56 -) LeftNumber: signal that keeps left number

Line 57 -) RightNumber: signal that keeps right number

Line 58 -) ScoreLeft1: signal which keeps record of successful odd choices for left player

Line 59 -) ScoreRight1: signal which keeps record of successful odd choices for right player

Line 60 -) ScoreLeft2: signal which keeps record of successful even choices for left player

Line 61 -) ScoreRight2: signal which keeps record of successful even choices for right player

Line 62 -) IsOver: signal which indicates if the countdown is over and when it is '1' it prevents

From the 64th line, the section where the main operations of our code will be performed has started. We assign intermediate signals between the 65th and 70th lines to the outputs.

The Master Clock process, which is the main clock, starts from the 72 line. One of the two variables that we have defined on lines 73 and 74 is for the counter. Someone also checks the end of the game.

The 75th line is the line where the Master Clock process starts.

It is the part where we determine the Most Significant Bit of the random numbers that we produce with the Linear Feedback Shift Register method on the 77th and 78th lines.

Line 77: We XOR the 0 and 1 bits of the previous random number to determine the left player's next random number. The result is the Most Significant Bit of the next random number on the left.

Line 78: We XOR the 0 and 1 bits of the previous random number to determine the right player's next random number. The result is the Most Significant Bit of the next random number on the right.

Line 79: If condition is the one that will perform the operations to be performed in each cycle corresponding to the rising edge of the Master Clock.

Operations between the 82nd and 97th lines indicate that the game is over when the game time is over and the actions taken to determine the winner.

If condition in line 82 is created for the operations that will be activated when the time expires. It continues up to 96 lines.

If the Over variable in 83 lines is 1, it means the game is over.

The IsOver variable on line 84 stops the score increase after the end of the game when logic is 1.

On line 86, we change the time manually from 01 to 00 at the end of the game because we want to prevent entering the other cycle.

Between the 87th and 89th lines includes if condition, which evaluates the situation where the sum of the odd and even scores of the left player is greater than the sum of the odd and even scores of the right player. In this condition, the W variable that determines the winner is '10'.

Between the 90th and 92nd lines includes if condition, which evaluates the situation where the sum of the odd and even scores of the right player is greater than the sum of the odd and even scores of the left player. In this condition, the W variable that determines the winner is '01'.

Between the 93rd and 95th lines includes if condition, which evaluates the situation where the sum of the left player's odd and even scores is equal to the sum of the right player's odd and even scores. In this condition, the W variable that determines the winner is '00'.

The 97th line is the beginning of if condition, which includes actions during game time and game play.

At lines 100 and 101 we shift our numbers to the right and we assign new random generated bits to most significant bits of those numbers. By doing these operations we successfully create our new random generated numbers

At lines 104-108 we check if the right number of the clock became zero. If it is zero, then we assign the value 9 to it to maintain the clock process. If it is nonzero, we decrease it by 1 to maintain clock process.

At lines 110-114 according to the COUNTERLEFT variable we check whether it is right time to decrease the left number of the clock. If COUNTERLEFT is equal to then it means 10 seconds passes and we need to decrement the number at the left by 1. After decrementing we assign the value 0 to COUNTERLEFT variable and at every cycle because of 1 second passes, we increment COUNTERLEFT by.

Between lines 121-128 process begins when the PRO changes if we catch a rising edge of PRO it means that player right made an odd choice. To get point the game should not be over and the least significant bit of the random number corresponds to right player must be 1. So, we check these conditions and if these conditions are met, we increment the signal which keeps the record of the correct odd choices for right player by 1

Between lines 130-137 process begins when the PRE changes if we catch a rising edge of PRE it means that player right made an even choice. To get point the game should not be over and the least significant bit of the random number corresponds to right player must be ‘0’. So, we check these conditions and if these conditions are met, we increment the signal which keeps the record of the correct even choices for right player by 1

Between lines 139-146 process begins when the PLE changes if we catch a rising edge of PLE it means that player left made an even choice. To get point the game should not be over and the least significant bit of the random number corresponds to left player must be ‘0’. So, we check these conditions and if these conditions are met, we increment the signal which keeps the record of the correct even choices for left player by 1

Between lines 147-154process begins when the PLO changes if we catch a rising edge of PLO it means that player left made an odd choice. To get point the game should not be over and the least significant bit of the random number corresponds to left player must be ‘1’. So, we check these conditions and if these conditions are met, we increment the signal which keeps the record of the correct odd choices for left player by 1

İn our testbench code, at stimulus part firstly we wait for 7ns to see the first random numbers for both players. Then each player makes only one choice whether odd or even choice by pressing buttons. To represent buttons, we have 4 inputs which are PRO, PRE, PLE, PLO. After the random generated numbers come each player presses a button so, for right player even or odd choice became 1 other one stays as zero. Conditions are same for the left player.

between the lines 112-115 we can see that right and left made an even choice. Their even buttons are ‘1’ and odd buttons are zero.

At lines 116-121 we made the values of all buttons as ‘0’ to demonstrate the real pressed button situation because when you press button you get 1 value but when you take off your hand the value you get turn backs to zero at the project we demonstrated.

After those lines we repeated, process goes in the same way. Again, at lines 122-125 players make their choices then at lines 126-130 we assign the value 0 to all buttons. We repeat these lines with different choices until the game ends and to show that players cannot increment their score after the countdown ends there are some choices to demonstrate that situation.

1. Limitations of Our Solution

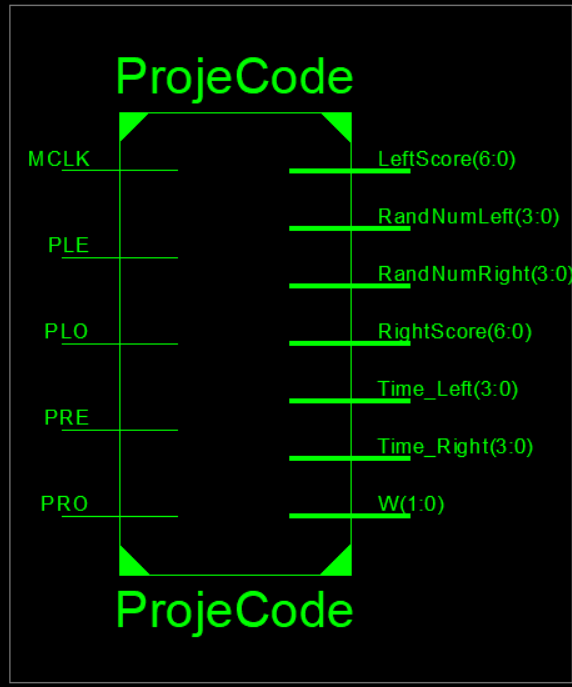
In our code we designed a game where players make choices if the number is odd or even. And in a game logic they must press only one button when the number is shown. However, if they try to press both the even and the odd choice button they will get one point no matter the number is. Or if they press the correct button twice for a number, they will get 2 points. We assumed that players follow the rules of the game which are they can make only one choice for a number and they can only press one time to corresponding button.

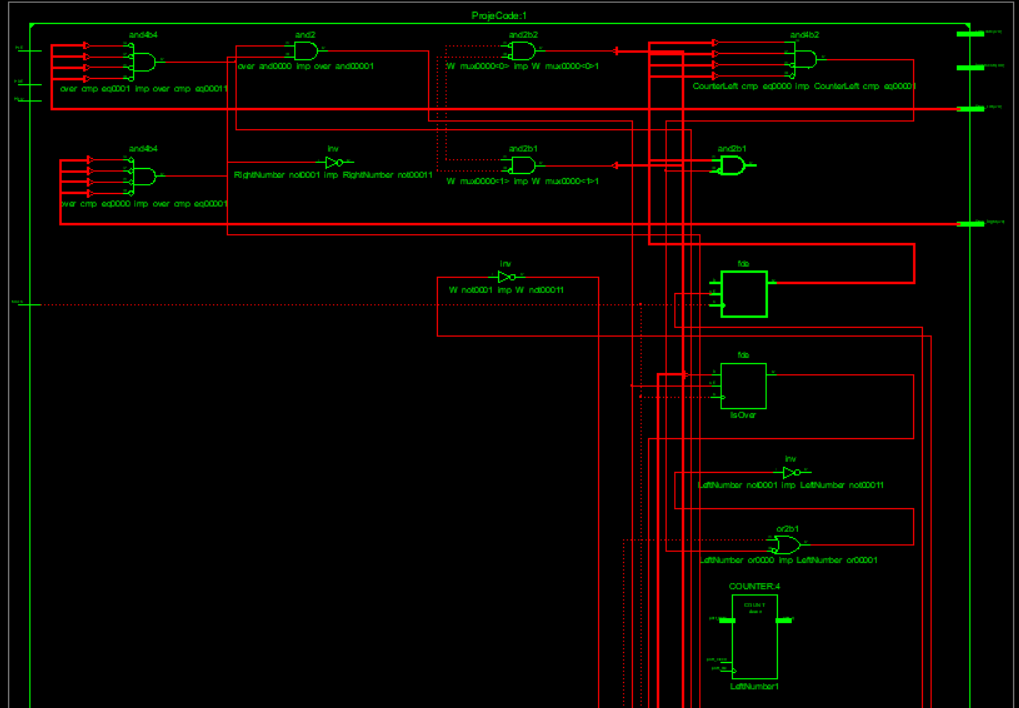
1. References

* <https://en.wikipedia.org/wiki/Linear-feedback_shift_register>
* ELEC 204 Digital System Design Laboratory Manual Experiment #4
* <https://stackoverflow.com/questions/43081067/pseudo-random-number-generator-using-lfsr-in-vhdl>

1. Appendices

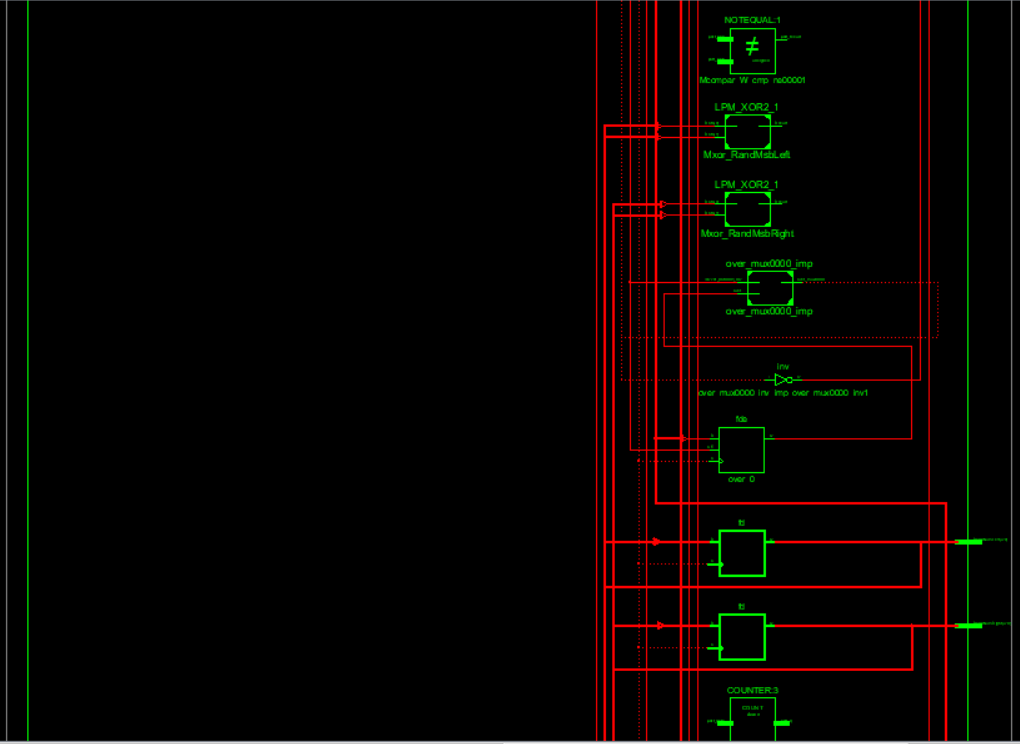
* RTL schematic

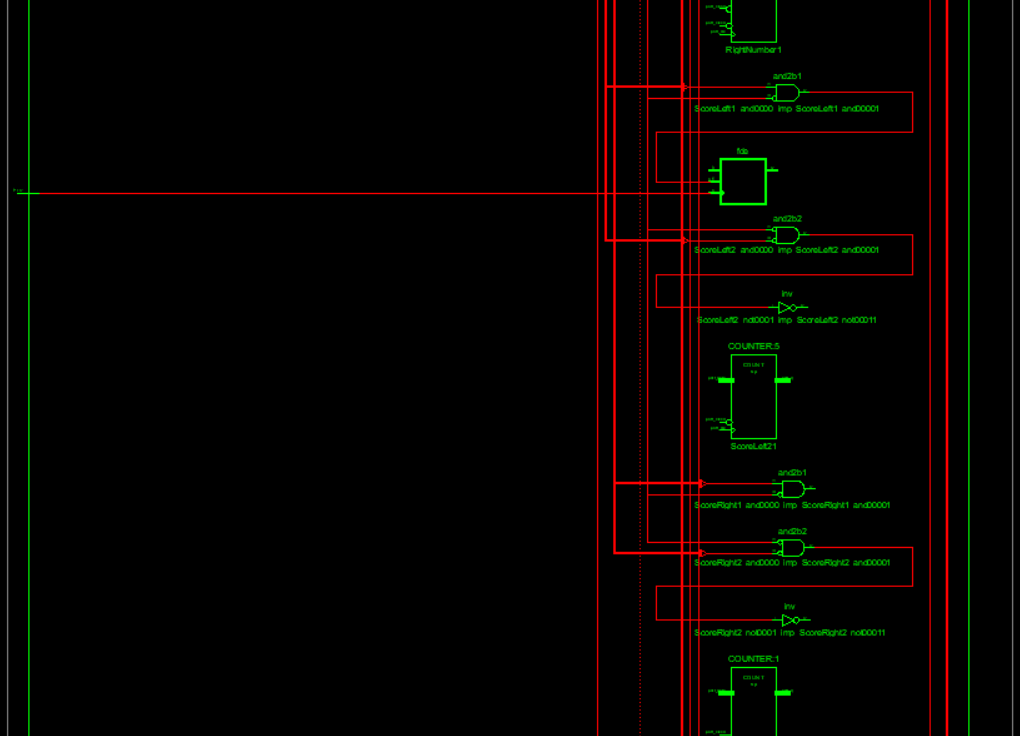




metin, elektronik eşyalar içeren bir resim

Açıklama otomatik olarak oluşturuldu





metin, elektronik eşyalar, vitrin içeren bir resim

Açıklama otomatik olarak oluşturuldu

* Source Codemetin içeren bir resim

  Açıklama otomatik olarak oluşturuldu

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

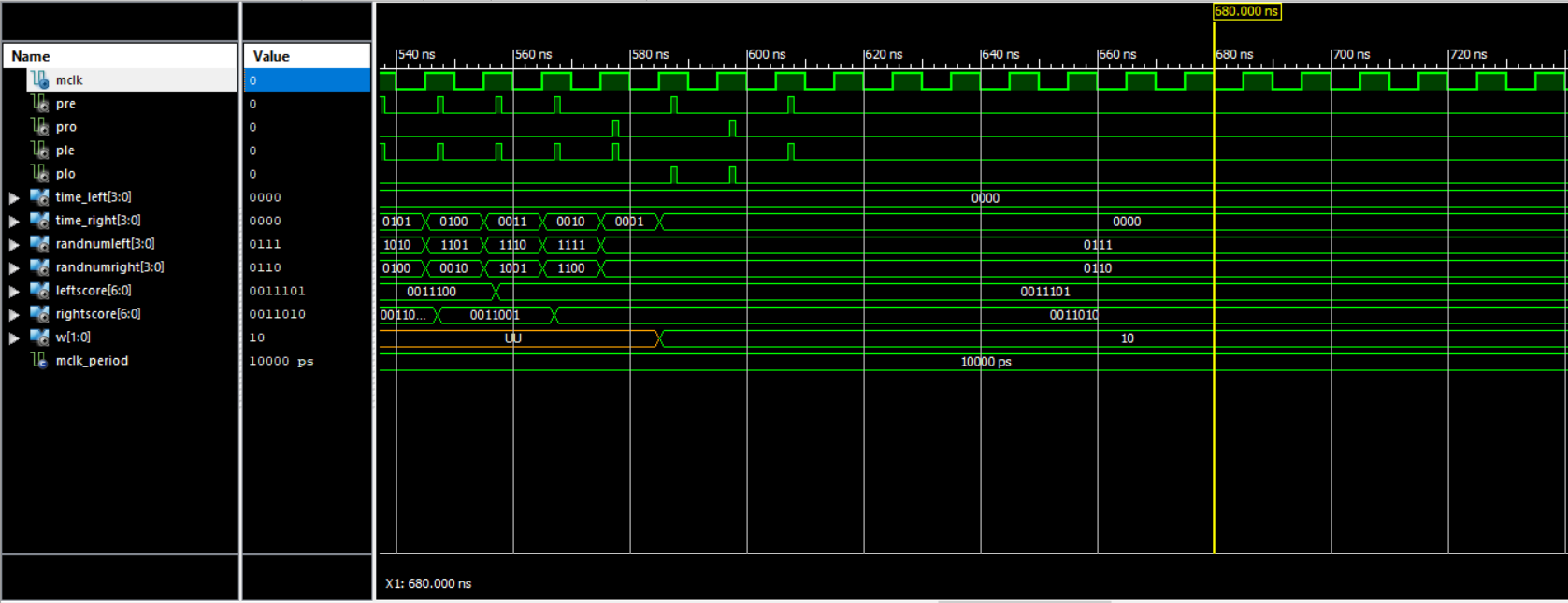
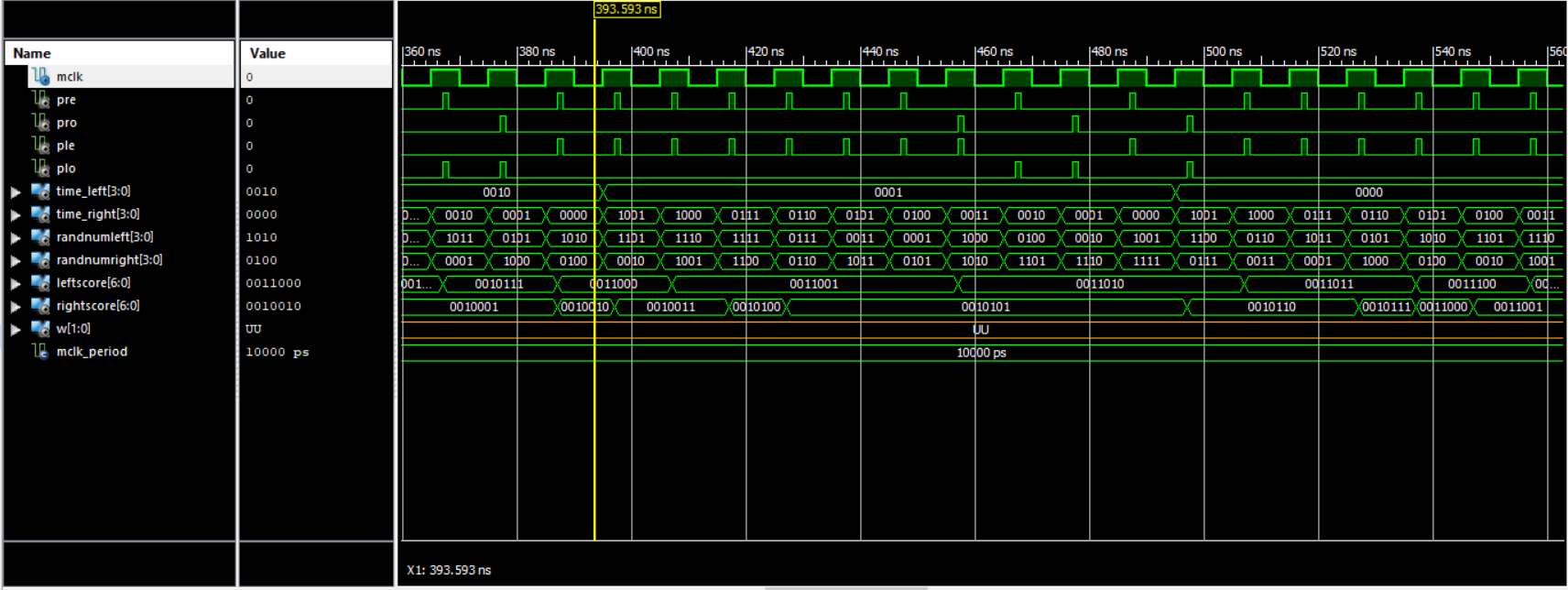
metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

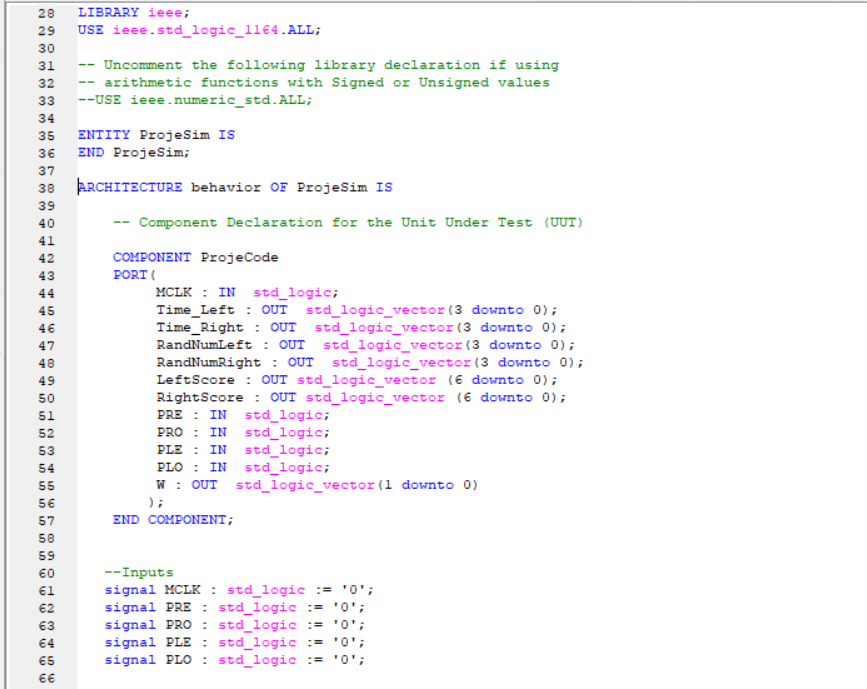
* Timing Diagram



metin, elektronik eşyalar içeren bir resim

Açıklama otomatik olarak oluşturuldu

* VHDL TESTBENCH CODE



metin içeren bir resim

Açıklama otomatik olarak oluşturuldu



tablo içeren bir resim

Açıklama otomatik olarak oluşturuldu