Lab 4: Final Project - An Application

Ky To, Phongsakorn Liewsrisuk, Rebecca Chu

Hours
Design: hrs || Test / Debug : hrs || Documentation: 5 hrs

Individual Contributions:
Ky To: experimental performance, report write-up
Phongsakorn Liewsrisuk: experimental performance, report write-up
Rebecca Chu: experimental performance, report write-up

Tables of Contents

ABSTRACT

This project requires expertise and deep understanding in digital logic design such as finite state machine as well as combinational logic. Although we encountered a major difficulty of synchronizing the timing, it was overcame by taking a simpler steps of resetting the D flip- flops. Also, many real-world engineering problems were encountered in this project as such the connection problems and the logic designing problems. We tried to simplify our logic as simple as possible to minimize the error.

INTRODUCTION

The purpose of this lab is to learn to combine the process of designing and developing digital systems to solve the real world engineering problems. The project requires familiarity with Verilog design language, the GAl 22V10 gate array , the Quartus IDE, the DE1 board and most important of all the creativity. The advantage of Quartus is the schematic tools that we can create the schematic diagram and converts them to Verilog code.

The game uses some secret phasor weapons to shoot down the attacking of hundreds of hostile birds randomly coming down from the top. This is played on a 8x8 grid LED matrix. The phasor weapon will be prepared to move left or right at the bottom of the matrix to avoid the birds from the top. If one of the bird hit the phasor, the player loses and the game is reset. If the player is successfully survive through the first 15 seconds of the game, he or she is taken to the next level of super-speed birds attacking. Fortunately, the player can get help from the most right or most left edge of the matrix where he or she can pull out multiple of phasors into the fight.

DESIGN SPECIFICATION

This is the general design specification

- The design of this game is implemented on the DE1 board.
- The implementation of the design is needed to be at least one finite state machine utilize on the Gal22V10 and one on Cyclone II device.
- This system can have both structural coding parts and schematic - converted coding parts.

Game specifications

- The enemy, red dots, will be randomly dropping down from the top of the grid.
- The phaser, green dot, can only shift left and right and will be able to shoot a bullet at a faster speed than the red dots.
- Once the bullet hit the enemy, both of them disappear from the display.
- After you lose, the game will reset itself.

Extra features

- Multi-level play. After 15 seconds, the the enemy will be coming at faster speed.
- Multiple phaser weapons can be used.

DESIGN PROCEDURE

The Shifting

For the shifting of the enemy and the bullet, the random generator feed in the Y inputs to the shift registers , while the shooting button feeds in the X input to the X-shift registers. The outputs of each D flip flop will be compared, which will go through the combinational logic CMB and And gate. They will be displayed through each LEDs in the columns. If the compared values are 1 and 1. Both of the D flipflop that sent in the signal will be reset, thus the LEDs light

will turn off. If the values are not 1 and 1, it will be shifted to the next D- flip flop. Then it will be display on the LED on the next clock cycle.

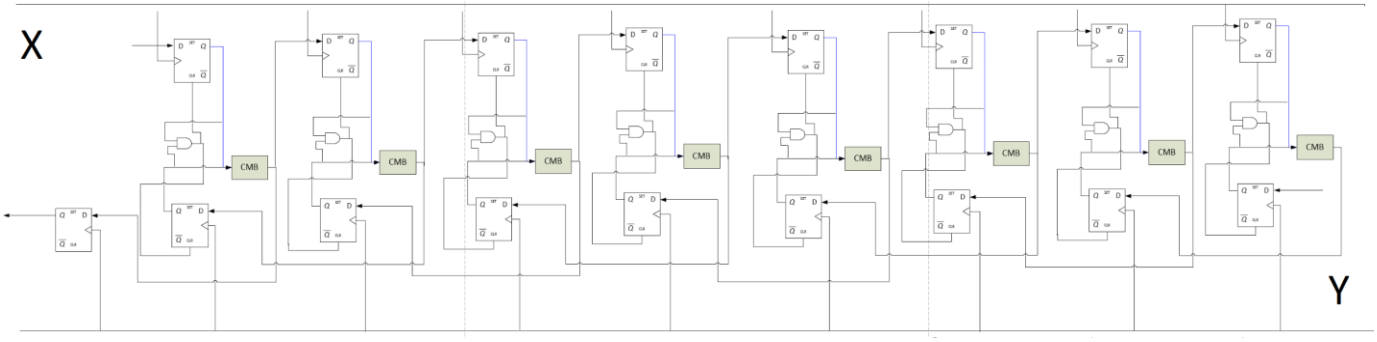


Fig 1.1shows the display for the layout of the shift registers and the pin assignment of the LEDs connection. The green boxes are showing the flow of the X- input, bullet, while the red boxes show the flow of the Y-input, the enemy.

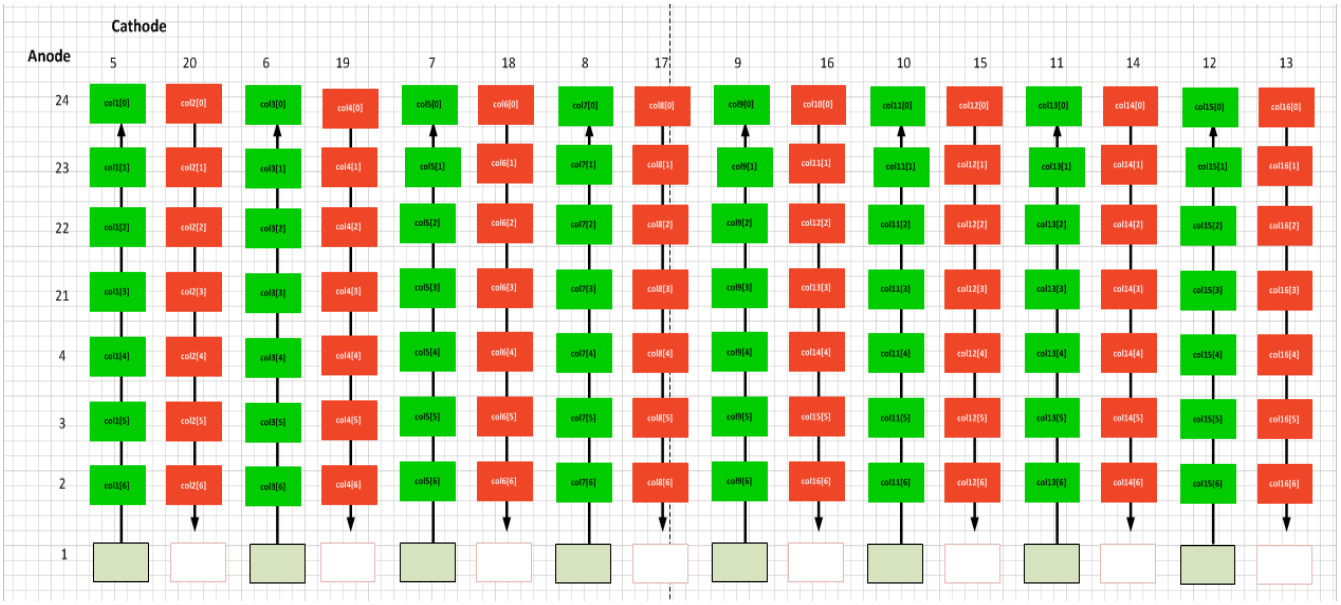


fig 1.2 This is the design of the shift registers for the phasor and on one LED column

X and Y are the input to the CMB while X' and Y' are the output of the D-flip flop.

X	Y	X'	Y'
0	0	0	0
1	0	1	0
0	1	0	1

1	**1**	**0**	**0**

fig 1.9 the truth table for CMB

Y' = ~XY
X' = X~y

Linear Feedback Shift Register

An 8 bits LFSR is used. Specifically, this is an internal XOR LFSR. Between D-flip flop 1 and 2, 3 and 4, 4 and 5, and 5 and 6 are tap with the XOR gate whose has second input from Q of D - flip flop 7. When this system is clocked, the bits that are not tapped shift from left to right while the ones that are tapped are XORed with Q of D flip flop 7.
Detail of the Verilog module for this LFSR can be found in the Appendix.

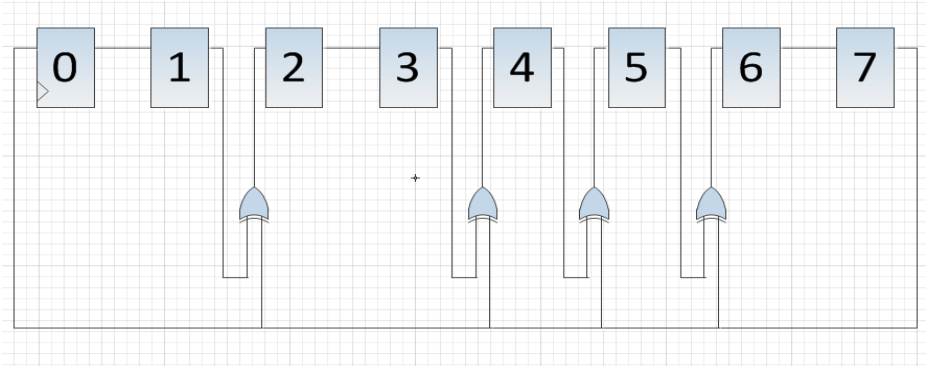


fig 1.3 The basic logic diagram of a LFSR, which is utilized as a random generator of the downcoming baron

Phasor Shift Register
This Phasor shift register is responsible for the phasor weapon moving left or right and shooting out the bullet in the last row of the LED display. This design is implemented with 8 D-flip flops in the shift register. Each flip flop represent its' position on the last row of the 8x8 matrix LED.

There are four inputs to this register: phasor weapon, shoot, shift right, and shift left. As a default, the phasor always starts at the first flip flop, ie, the leftmost side. Each flip flop has a 2 to 1 Mux between the phasor input from previous Flip flop or the output of its own flip flop back into the flip flop. The control selector for the mux is whether the user input left or right. In other words, the phasor will stay at its own position if the user does not put in any left or right signal.

To decide whether the input is going left or right, the phasor combined with the direction signal with the logic gate AND. When the output of the AND gate is 1, it goes through to the or gate before the Mux. The output signal is ANDed with both the right signal and left signal. If there is a right signal, the new output signal will move on as the input signal to the D-flip flop on the

right. If there is a left input signal, the flip flop output will go back to the previous D-flip flop on the left. The phasor weapon is capped by the left and right most boundaries of the phasor.

Additionally, each flip flop sends out the output and ANDed with the shoot signal. As a result, it will only shoot out bullet from where the phasor weapon is currently located at. The output of the D-flip flop is also inverted and set as the output to the DE1 Board for the display on the LED Matrix.

For the detailed schematic design for the Phasor weapon, please see the Appendix.

Gal Chips

Last of all, we used the Gal chip as the counter. Once the Gal chip counts to about 15 seconds, it will send the signal to DE1 board to change the shifting clock to a faster speed. The gal chip is simply designed by using the finite state machine to count to 9 and it will stop at 9 and keep shooting the enable signal to the DE1 board.

	Present State					Next State			
	A	B	C	D		A	B	C	D
0	0	0	0	0		0	0	0	1
1	0	0	0	1		0	0	1	0
2	0	0	1	0		0	0	1	1
3	0	0	1	1		0	1	0	0
4	0	1	0	0		0	1	0	1
5	0	1	0	1		0	1	1	0
6	0	1	1	0		0	1	1	1
7	0	1	1	1		1	0	0	0
8	1	0	0	0		1	0	0	1
9	1	0	0	1		1	0	0	1

	CD			
AB	00	01	11	10
00	1			1
01	1			1
11	d	d	d	d
10	1	1	d	d

$$D_d = (A + \bar{D})En$$

	CD			
AB	00	01	11	10
00		1		1
s01		1		1
11	d	d	d	d
10			d	d

$$D_c = \bar{A}\bar{C}DEn + C\bar{D}En$$

	CD			
AB	00	01	11	10
00			1	
01	1	1		1
11	d	d	d	d
10			d	d

$$D_b = En(B\bar{D} + B\bar{C} + \bar{B}CD)$$

	CD			
AB	00	01	11	10
00				
01			1	
11	d	d	d	d
10	1	1	d	d

$$D_a = En(A + BCD)$$

fig 1.4 displays the K-map and truth table for the Gal chip

Displaying outputs on LED matrix

The LED matrix in use is an 8x8 LEDs with Anode in common. There are 16 columns and 8 rows in the matrix (8 columns of Red LEDs and 8 columns of Green LEDs, one from each group is placed next to the one from other group alternatively). As only one column can be activated at a time, a '16to1' module is used to select out 1 bit out from each row of 16 bits. The selection is cycled using an other 'counter' module that counts from 0 binary to 16 binary. Since there are 8 rows, 8 instantiations of the '16 to 1' module are generated. The inputs of these instances are delivered from the control modules that attached to 8 pairs of 'Red' and 'Green' registers while the outputs, in a cycling manner, control the display of LED lights.

HARDWARE IMPLEMENTATION

Shooting Game

The Verilog code for the shooting game was implemented in the Quartus. With a successful compilation, the code was programmed into the DE1 board through Quartus' USB Blaster with the following inputs and outputs pin assigned:

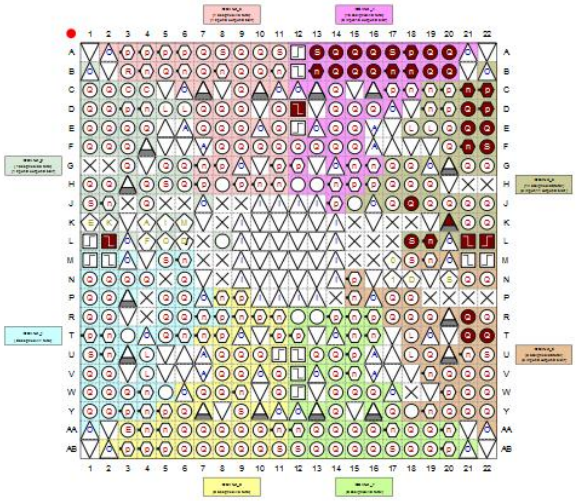


fig 1.5 Location of the Pins on DE1 Board

Node Name	Direction	Location
L	Input	PIN_L22
Out[15]	Output	PIN_B20
Out[14]	Output	PIN_A20
Out[13]	Output	PIN_B19
Out[12]	Output	PIN_A19
Out[11]	Output	PIN_B18
Out[10]	Output	PIN_A18
Out[9]	Output	PIN_B17
Out[8]	Output	PIN_A17
Out[7]	Output	PIN_B16
Out[6]	Output	PIN_A16
Out[5]	Output	PIN_B15
Out[4]	Output	PIN_A15
Out[3]	Output	PIN_B14
Out[2]	Output	PIN_A14
Out[1]	Output	PIN_B13
Out[0]	Output	PIN_A13
Phasor2	Input	PIN_L21
R0	Output	PIN_F21
R1	Output	PIN_E22
R2	Output	PIN_E21
R3	Output	PIN_D22
R4	Output	PIN_D21
R5	Output	PIN_C22
R6	Output	PIN_C21
Shift_Left3	Input	PIN_T21
Shift_R1	Input	PIN_R21
Shoot	Input	PIN_T22
en	Output	PIN_L19
iClk	Input	PIN_D12
nfive	Input	PIN_L18
outPhar	Output	PIN_F22
outclk	Output	PIN_J18
outrst	Output	PIN_K20
switchR	Input	PIN_L2

fig 1.6 Pin Assignment chart for Shooting Games's inputs and outputs on DE1 Board

- iClk is assigned to DE1's 27MHz clock to input the signal
- Out[0] to Out[15] are assigned to the columns of 8x8 LEDs' pins.
- R0, R1, R2, R3, R4, R5, R6 are assigned to the rows of 8x8 LEDs' pins.
- nfive_b, en, and outclk, outrst are assigned to I/O pins for the counter
- Shift_Left3, Shift_R1, and Shoot are assigned KEY buttons that controls the behavior of this weapon function. Shift_left is Key3 (the left one) , Shoot is Key2 (the Middle one), and Shift_right is Key 2 (the right one)
- outPhasor is the output to the I/O pin that lights up the last row of LED for the phasor weapon.
    - Input L is assgined to SW[0] where it is controlled to initialize the Birds to come down.
    - Phasor2 is connected to SW[1] where it can generate the phasor weapon. This is also the switch that allows user to customize the size of the phasor weapon.

The code for the state machine counter is also programed into the 24 pin P22V10G GalChip through ispLever that inputs the clock, rest, and enable the outputs the four bits that counts from 0 to 9. After reaching state 9, it is going to shoot out the signal !Five_b that will make the game into faster mode.

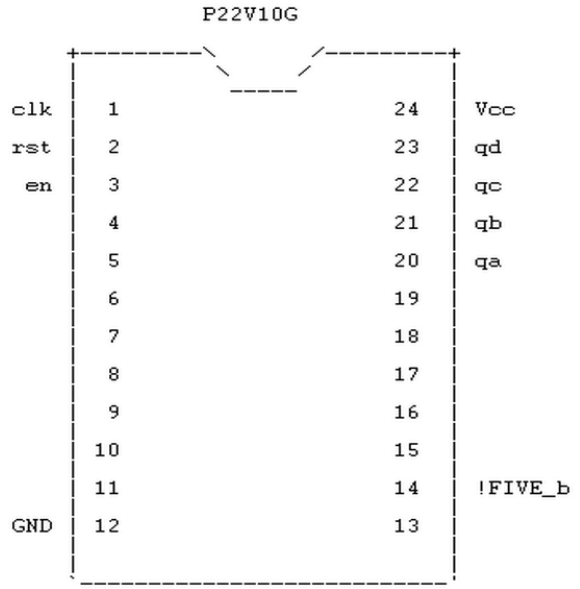


fig 1.7 Pin Assignment for Counter on GAL22V10 Chip

After having all the codes programmed into the DE1 board, the outputs are connected to the LED board to get the display working. In order for the LED to light up, there should be a high voltage in the Row and a ground to the column. For a specific row, each column has two LEDs, one red and one green. To choose which color to turn on, just set ground of the corresponding color pin to the ground. For example, if I want to light up the Red LED in second row, third column, pin

23 needs input voltage and pin 7 will need to be set to ground. Since to light up the pin, the row has to set to high voltage and the column needs to set to ground, there might be unwanted lit up LED where the high voltage row and grounded column intersects. Therefore, the LED is lit up column by column so that no LED will light up accidently. This was achieved by scanning through all the columns 16 columns of the 8x8 LED matrix at a very high frequency. Therefore, although the LEDs are not light all up at the same time, it is actually scanned through at a much higher frequency than our eyes can detect. As a result, when the LED is displayed, it seems like all the columns of the LEDs are light up at the same time.

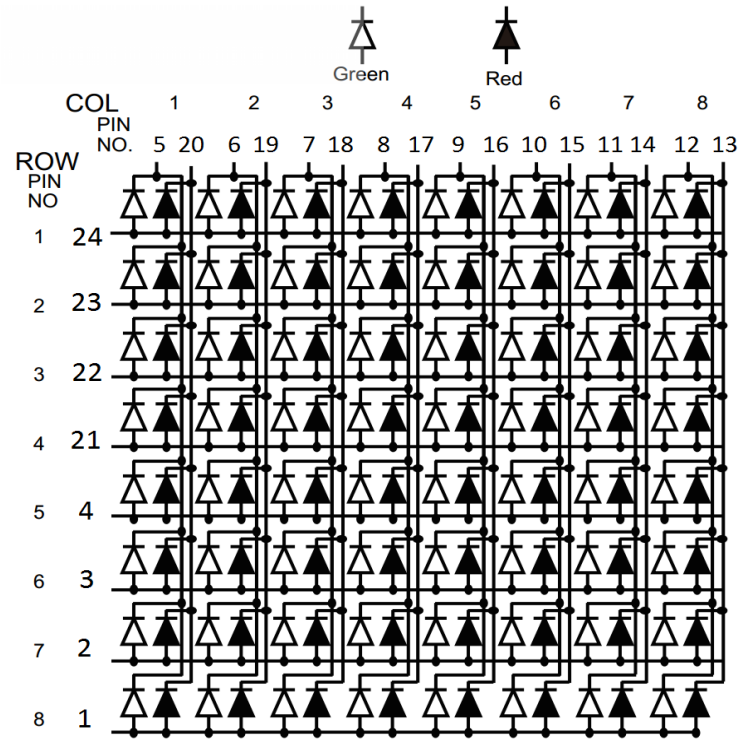


fig 1.8 Pin Assignment for 8x8 LED Matrix

TEST PLAN

First of all, our logics and codes are tested on Bughunter Verilog IDE. There are three main modules are considered as those crucial ones, matrix_shift, phasor and Eight_Bit_LFSR. Except for the phasor module (because it is created using Schematic function in Quartus), each has its own test bench, where test vectors and values are assigned to sets of inputs and waveform for monitored signals are generated. The results are then compared to the expected outcomes that reflect the functionalities of each part of this system. If results are not satisfying or unexplainable, debugging is in process that the codes and logics are scrutinized and undergone changes to ready for new rounds of testing until results are satisfying and reflect the expected and desired functionalities.

For the phasor module, it is tested alone and directly on the DE1 board using signals generated in real time by buttons which are assigned to ‘phasor’’s inputs. Results are illustrated when an active Signal Analyzer probe is in contact with the expected result-outputting ports on the DE1 board.

RESULTS

Gal chip test simulation

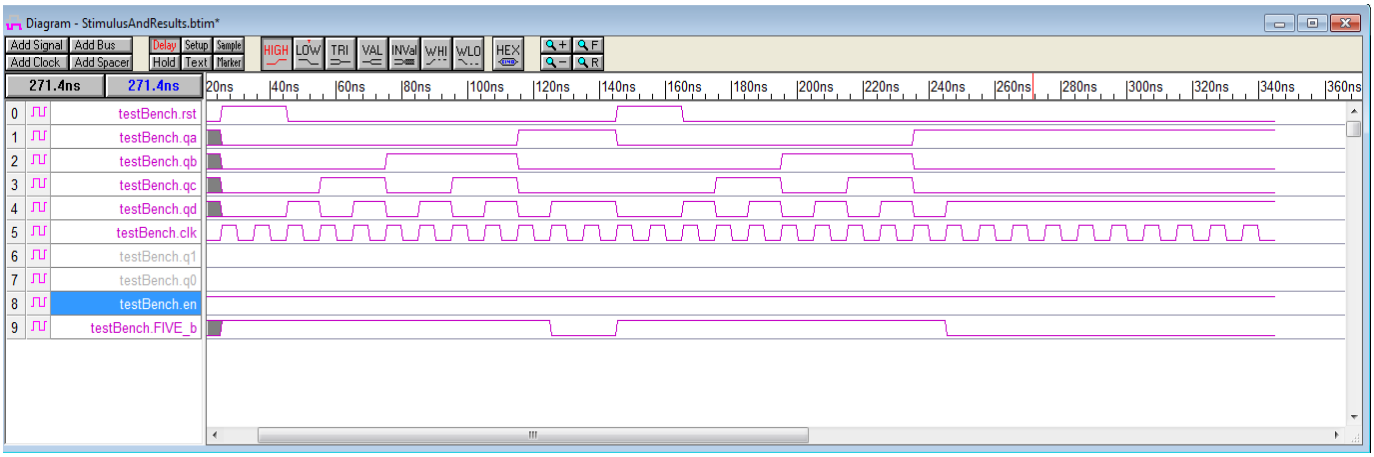


fig 3.3 the gal chip counts to 9 then nFive_b keeps sending signal.

‘matrix_shift’ test simulation

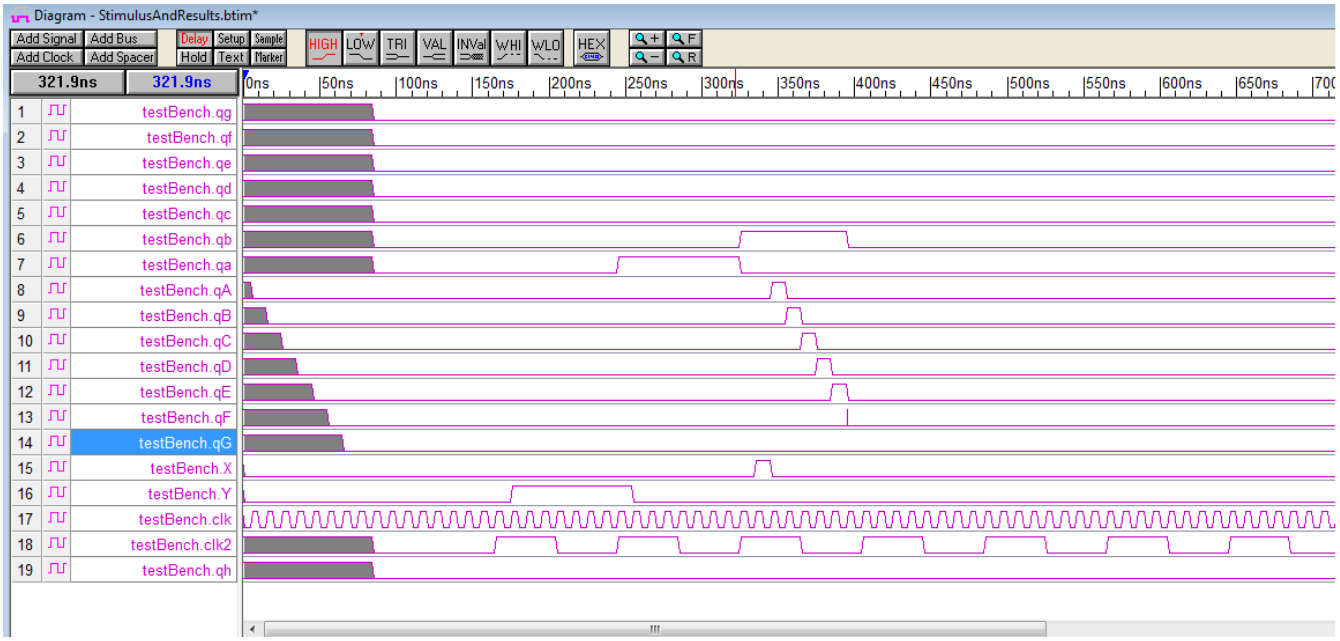


fig 3.4 displays the simulation of the shifting of the bullet (green) and the enemy (red)

Eight_Bit_LFSR test simulation

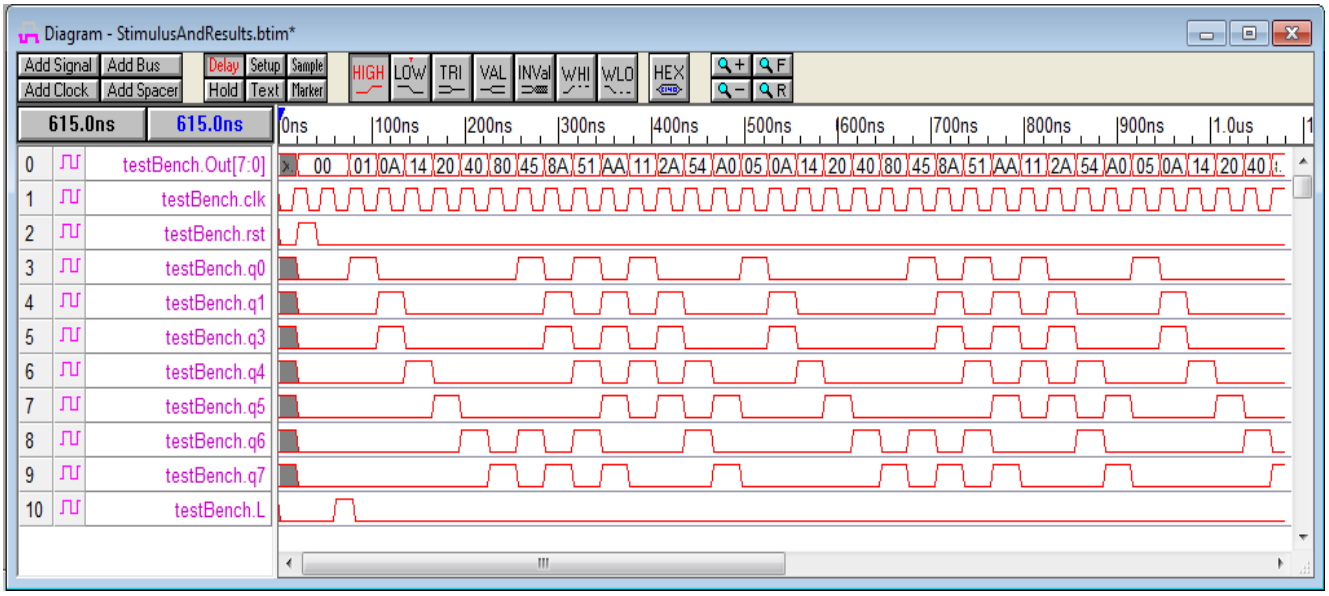


fig 3.5 random generator simulation

Actual Demo

Through the demo in class, our actual implementation on hardware are verified to work and function as designed.

ERROR ANALYSIS

One major difficulty that we faced in this design is the different timing of the clocks of the bullets and the enemies. The timing is hard to match as we are using two different clocks for the D flip-flops. We have spent many hours pondering the possible solutions for this problem. However, we overcame this problems by resetting each pair of the D-flip flops.

SUMMARY

This project is constructed dividedly on GalChip, and on DE1 board (both structural code and Quartus's schematic drawing tool are utilized to programmed the board). The two parts are connected through the assignment of I/O ports to mutually exchanging signals to each part when specific signals are timed to deliver. This creates a closed loop of executions that perform information exchanges between the two components. This 'Shooting Red Barons' game fully utilizes the ability of storing and shifting bits in registers and combines those registers into pipelines of information exchanging and logical calculations. Specifically, data are analyzed and

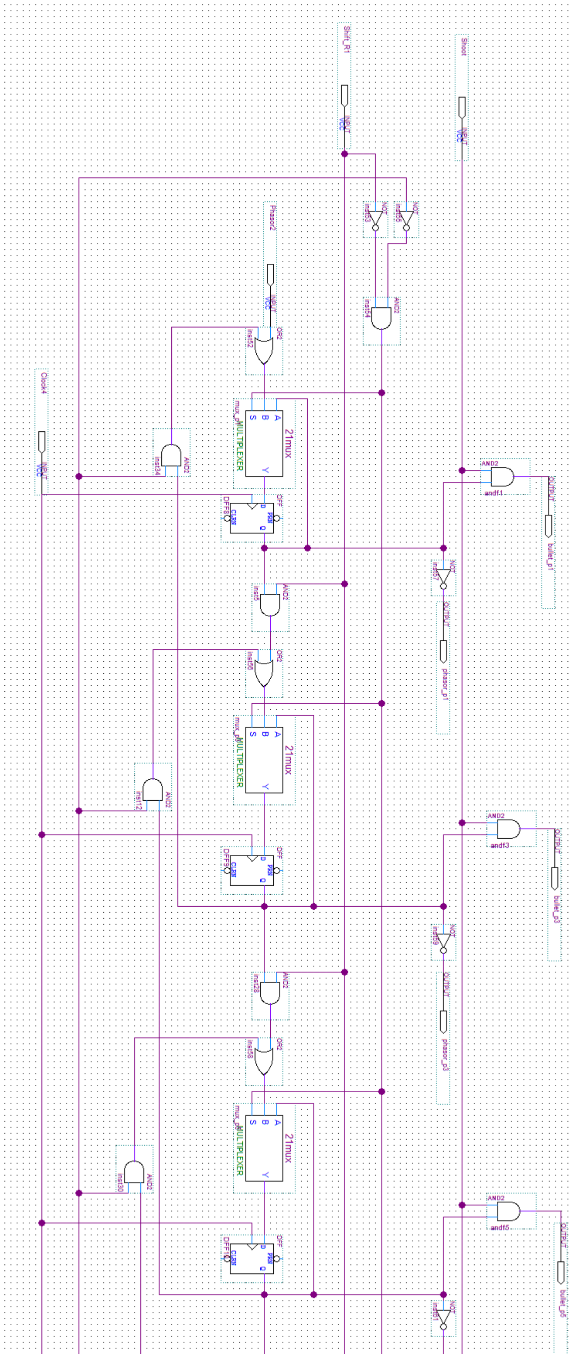
modified in logical controlling modules before shifting to next bit storage and outputting to LED light. Debugging process takes up a good amount of time from starting to finishing. During this process, different tools are used to analyze, test vectors and outputs' generated waveform, 'break points' and individual-statement stepping in, and using physical Signal analyzer probe for detecting real time signals.

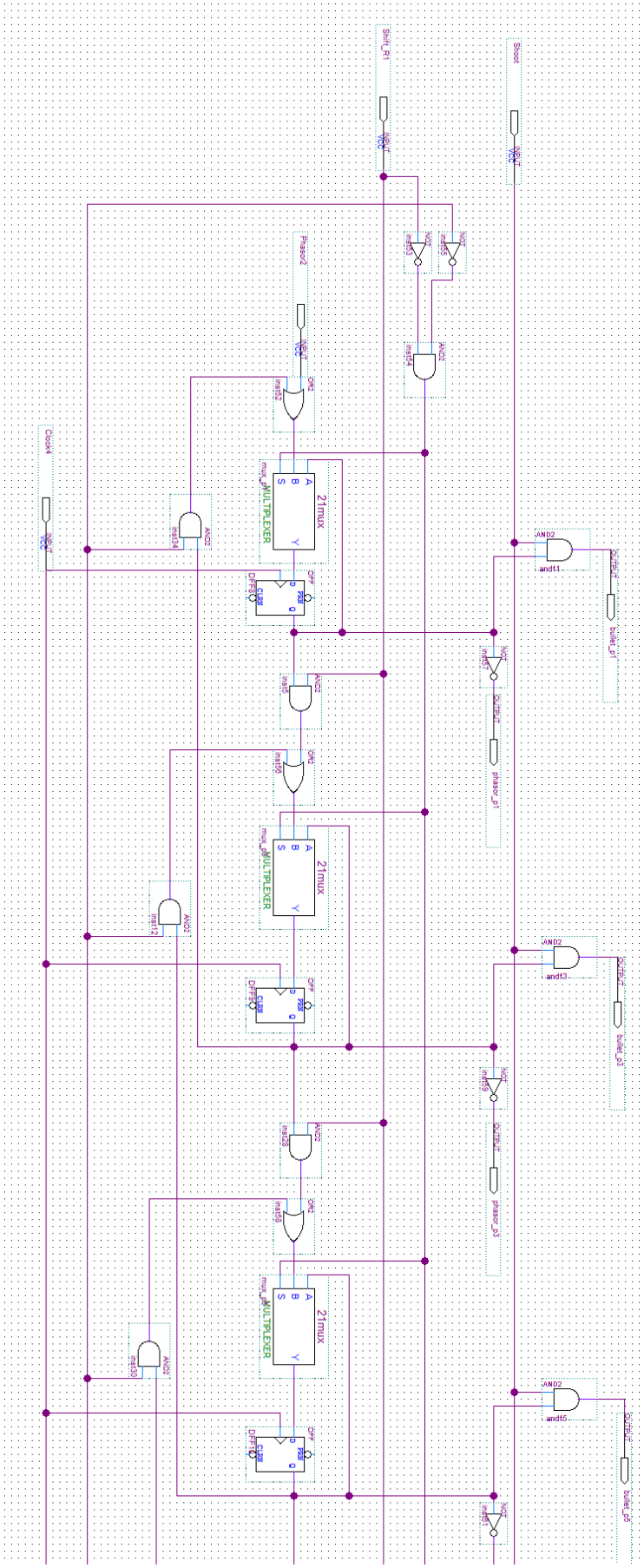
CONCLUSION

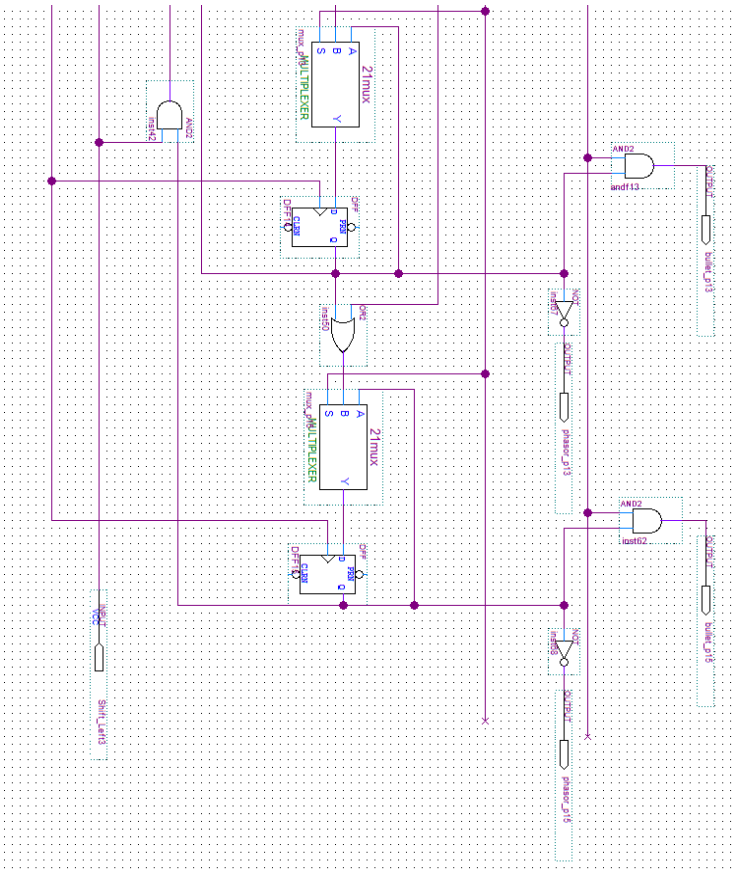
In conclusion, this project requires different digital logic knowledge and skills such as finite state machine, combination logic and connecting signal between the DE1 board and the Gal chip. The projects also requires expertise in simulation of the verilog code and solving the hardware and software problems. We have encountered difficulty in the different timing of the clocks signal, however; this was overcome by taking a simpler steps of designing by resetting the D- flip flip. Overall the project was a success as we have learnt many steps in solving the real world engineering problem and process and gained a valuable team work experience.

APPENDIX

Schematic Design of Phasor Weapon's Shift Register




Shift_R1
Shoot
Phase2
Clock4
2:1mux
MULTIPLEXER
DFFR
AND2
OR2
NOT
andf1
andf3
andf5
bullet_p1
bullet_p3
bullet_p5
phaser_p1
phaser_p3



Verilog Code of 'Shooting Red Barons' system

```
module final7(outrst, en, nfive, outclk, outPhar, Out, R0, R1,R2,R3,R4,R5,R6, L, iClk, Shift_R1,
Shift_Left3, Shoot, Phasor2, switchR);

  input Shift_R1, Shift_Left3, Shoot, Phasor2;
  input nfive, switchR;
  output R0, R1,R2,R3,R4,R5,R6;
  //reg R0,R1,R2,R3,R4,R5,R6;

  output outrst;

  output[15 :0] Out;

  output outclk;
  output en;

  assign en = 1;

  buf buf1 (outclk, tBase[25]);
  buf(outrst, rst);

  output outPhar;// high or low input for row

  wire [6:0] col1;
```

```
  wire [7:0] col2;
  wire [6:0] col3;
  wire [7:0] col4;
  wire [6:0] col5;
  wire [7:0] col6;
  wire [6:0] col7;
  wire [7:0] col8;
  wire [6:0] col9;
  wire [7:0] col10;
  wire [6:0] col11;
  wire [7:0] col12;
  wire [6:0] col13;
  wire [7:0] col14;
  wire [6:0] col15;
  wire [7:0] col16;

  wire  [7:0] X;
  wire fclk, sclk;
  //input Shift_Left3, Shift_R1;

  input L, iClk;

  wire q0, q1, q2, q3, q4, q5, q6, q7, clk2;

  wire rst,qd,qc,qb,qa;

  reg [25:0]  tBase;
  always@(posedge iClk) tBase <= tBase + 1'b1;

  not notfive(timeDone, nfive);

  //or or1(fclk, timeDone,

  buf buf2 (sclk, tBase[21]);
  buf buf3(fclk, tBase[19]);
  //buf timechange(fclk, tBase[20]);

  schem21 timer(timeDone, fclk, sclk, ClkSpeed);

   wire phasor_p1, phasor_p3, phasor_p5, phasor_p7, phasor_p9, phasor_p11, phasor_p13, phasor_p15;

  Eight_Bit_LFSR  gen1(q0,q1,q2,q3,q4,q5,q6,q7,L,clk2,rst);

  shifter myShifter(col2[0], col2[1], col2[2], col2[3], col2[4], col2[5], col2[6], col2[7], col1[0], col1[1], col1[2],
col1[3], col1[4], col1[5], col1[6], clk2, X[0], q0, ClkSpeed, rst);
  shifter myShifter1(col4[0], col4[1], col4[2], col4[3], col4[4], col4[5], col4[6] , col4[7], col3[0], col3[1],
col3[2], col3[3], col3[4], col3[5], col3[6], clk3, X[1], q1, ClkSpeed, rst);
  shifter myShifter2(col6[0], col6[1], col6[2], col6[3], col6[4], col6[5], col6[6] , col6[7], col5[0], col5[1],
col5[2], col5[3], col5[4], col5[5], col5[6], clk4, X[2], q2, ClkSpeed, rst);
  shifter myShifter3(col8[0], col8[1], col8[2], col8[3], col8[4], col8[5], col8[6] , col8[7], col7[0], col7[1],
col7[2], col7[3], col7[4], col7[5], col7[6], clk5, X[3], q3, ClkSpeed, rst);
  shifter myShifter4(col10[0], col10[1], col10[2], col10[3], col10[4], col10[5], col10[6], col10[7], col9[0],
col9[1], col9[2], col9[3], col9[4], col9[5], col9[6], clk6, X[4], q4, ClkSpeed, rst);
  shifter myShifter5(col12[0], col12[1], col12[2], col12[3], col12[4], col12[5], col12[6], col12[7], col11[0],
col11[1], col11[2], col11[3], col11[4], col11[5], col11[6], clk7, X[5], q5, ClkSpeed, rst);
```

```
  shifter myShifter6(col14[0], col14[1], col14[2], col14[3], col14[4], col14[5], col14[6], col14[7], col13[0],
col13[1], col13[2], col13[3], col13[4], col13[5], col13[6], clk8, X[6], q6, ClkSpeed, rst);
  shifter myShifter7(col16[0], col16[1], col16[2], col16[3], col16[4], col16[5], col16[6], col16[7], col15[0],
col15[1], col15[2], col15[3], col15[4], col15[5], col15[6], clk9, X[7], q7, ClkSpeed, rst);

  Count_To_Sixteen counter(Out,qd,qc,qb,qa,tBase[14],rst);

  schemtest3 Row0( qd, qc, qb, qa, col1[6], col2[0],col3[6], col4[0], col5[6], col6[0], col7[6], col8[0], col9[6],
col10[0], col11[6], col12[0], col13[6], col14[0], col15[6], col16[0],R0);
  schemtest3 Row1( qd, qc, qb, qa, col1[5], col2[1],col3[5], col4[1], col5[5], col6[1], col7[5], col8[1], col9[5],
col10[1], col11[5], col12[1], col13[5], col14[1], col15[5], col16[1],R1);
  schemtest3 Row2( qd, qc, qb, qa, col1[4], col2[2],col3[4], col4[2], col5[4], col6[2], col7[4], col8[2], col9[4],
col10[2], col11[4], col12[2], col13[4], col14[2], col15[4], col16[2],R2);
  schemtest3 Row3( qd, qc, qb, qa, col1[3], col2[3],col3[3], col4[3], col5[3], col6[3], col7[3], col8[3], col9[3],
col10[3], col11[3], col12[3], col13[3], col14[3], col15[3], col16[3],R3);
  schemtest3 Row4( qd, qc, qb, qa, col1[2], col2[4],col3[2], col4[4], col5[2], col6[4], col7[2], col8[4], col9[2],
col10[4], col11[2], col12[4], col13[2], col14[4], col15[2], col16[4],R4);
  schemtest3 Row5( qd, qc, qb, qa, col1[1], col2[5],col3[1], col4[5], col5[1], col6[5], col7[1], col8[5], col9[1],
col10[5], col11[1], col12[5], col13[1], col14[5], col15[1], col16[5],R5);
  schemtest3 Row6( qd, qc, qb, qa, col1[0], col2[6],col3[0], col4[6], col5[0], col6[6], col7[0], col8[6], col9[0],
col10[6], col11[0], col12[6], col13[0], col14[6], col15[0], col16[6],R6);

  schemtest3 phasorR( qd, qc, qb, qa, phasor_p1, col2[7],  phasor_p3, col4[7],  phasor_p5, col6[7],
phasor_p7, col8[7],  phasor_p9, col10[7],  phasor_p11, col12[7],  phasor_p13, col14[7],  phasor_p15,
col16[7], outPhar);

  and andDead1(dead1, col2[7], phasor_p1),
                    andDead2(dead2, col4[7], phasor_p3),
                    andDead3(dead3, col6[7], phasor_p5),
                    andDead4(dead4, col8[7], phasor_p7),
                    andDead5(dead5, col10[7], phasor_p9),
                    andDead6(dead6, col12[7],  phasor_p11),
                    andDead7(dead7, col14[7],  phasor_p13),
                    andDead8(dead8, col16[7],  phasor_p15);

          or or1(dead, dead1,dead2,dead3,dead4,dead5,dead6,dead7,dead8);

          //turning on all the light


          // reset when dead or time-up and display for 2 seconds
          or #2000000000 orreset(rst, dead, switchR);


 phasor(Shift_R1, Phasor2 , Shift_Left3, sclk, Shoot, phasor_p1, phasor_p3, X[2], X[1], X[0], X[3], X[4],
X[5], X[6], X[7], phasor_p5, phasor_p7, phasor_p9, phasor_p11, phasor_p13, phasor_p15);

 endmodule


 module schem21(
          selector,
          A0,
          B1,
          pin_name4
```

```
);


input wire	selector;
input wire	A0;
input wire	B1;
output wire	pin_name4;


\21mux b2v_inst(
	.S(selector),
	.B(B1),
	.A(A0),
	.Y(pin_name4));

endmodule

module phasor(
	Shift_R1,
	Phasor2,
	Shift_Left3,
	Clock4,
	Shoot,
	phasor_p1,
	phasor_p3,
	bullet_p5,
	bullet_p3,
	bullet_p1,
	bullet_p7,
	bullet_p9,
	bullet_p11,
	bullet_p13,
	bullet_p15,
	phasor_p5,
	phasor_p7,
	phasor_p9,
	phasor_p11,
	phasor_p13,
	phasor_p15
);


input wire	Shift_R1;
input wire	Phasor2;
input wire	Shift_Left3;
input wire	Clock4;
input wire	Shoot;
output wire	phasor_p1;
output wire	phasor_p3;
output wire	bullet_p5;
output wire	bullet_p3;
output wire	bullet_p1;
output wire	bullet_p7;
output wire	bullet_p9;
output wire	bullet_p11;
```

```
output wire     bullet_p13;
output wire     bullet_p15;
output wire     phasor_p5;
output wire     phasor_p7;
output wire     phasor_p9;
output wire     phasor_p11;
output wire     phasor_p13;
output wire     phasor_p15;

wire    SYNTHESIZED_WIRE_69;
reg     SYNTHESIZED_WIRE_70;
reg     SYNTHESIZED_WIRE_71;
reg     SYNTHESIZED_WIRE_72;
reg     SYNTHESIZED_WIRE_73;
reg     SYNTHESIZED_WIRE_74;
reg     SYNTHESIZED_WIRE_75;
reg     SYNTHESIZED_WIRE_76;
wire    SYNTHESIZED_WIRE_7;
wire    SYNTHESIZED_WIRE_8;
wire    SYNTHESIZED_WIRE_9;
wire    SYNTHESIZED_WIRE_10;
wire    SYNTHESIZED_WIRE_11;
wire    SYNTHESIZED_WIRE_12;
wire    SYNTHESIZED_WIRE_13;
wire    SYNTHESIZED_WIRE_14;
wire    SYNTHESIZED_WIRE_15;
wire    SYNTHESIZED_WIRE_16;
wire    SYNTHESIZED_WIRE_17;
wire    SYNTHESIZED_WIRE_77;
wire    SYNTHESIZED_WIRE_78;
reg     SYNTHESIZED_WIRE_79;
wire    SYNTHESIZED_WIRE_35;
wire    SYNTHESIZED_WIRE_36;
wire    SYNTHESIZED_WIRE_38;
wire    SYNTHESIZED_WIRE_39;
wire    SYNTHESIZED_WIRE_40;
wire    SYNTHESIZED_WIRE_41;
wire    SYNTHESIZED_WIRE_42;
wire    SYNTHESIZED_WIRE_43;
wire    SYNTHESIZED_WIRE_45;
wire    SYNTHESIZED_WIRE_46;
wire    SYNTHESIZED_WIRE_47;
wire    SYNTHESIZED_WIRE_48;
wire    SYNTHESIZED_WIRE_80;
wire    SYNTHESIZED_WIRE_50;
wire    SYNTHESIZED_WIRE_52;
wire    SYNTHESIZED_WIRE_54;
wire    SYNTHESIZED_WIRE_56;
wire    SYNTHESIZED_WIRE_58;
wire    SYNTHESIZED_WIRE_60;
wire    SYNTHESIZED_WIRE_62;
wire    SYNTHESIZED_WIRE_64;
wire    SYNTHESIZED_WIRE_65;
wire    SYNTHESIZED_WIRE_66;
wire    SYNTHESIZED_WIRE_67;
wire    SYNTHESIZED_WIRE_68;
```

```
assign	phasor_p1 = SYNTHESIZED_WIRE_70;
assign	phasor_p3 = SYNTHESIZED_WIRE_73;
assign	phasor_p5 = SYNTHESIZED_WIRE_74;
assign	phasor_p7 = SYNTHESIZED_WIRE_75;
assign	phasor_p9 = SYNTHESIZED_WIRE_76;
assign	phasor_p11 = SYNTHESIZED_WIRE_71;
assign	phasor_p13 = SYNTHESIZED_WIRE_72;
assign	phasor_p15 = SYNTHESIZED_WIRE_79;


assign	bullet_p1 = SYNTHESIZED_WIRE_69 & SYNTHESIZED_WIRE_70;

assign	bullet_p11 = SYNTHESIZED_WIRE_69 & SYNTHESIZED_WIRE_71;

assign	bullet_p13 = SYNTHESIZED_WIRE_69 & SYNTHESIZED_WIRE_72;

assign	bullet_p3 = SYNTHESIZED_WIRE_69 & SYNTHESIZED_WIRE_73;

assign	bullet_p5 = SYNTHESIZED_WIRE_69 & SYNTHESIZED_WIRE_74;

assign	bullet_p7 = SYNTHESIZED_WIRE_69 & SYNTHESIZED_WIRE_75;

assign	bullet_p9 = SYNTHESIZED_WIRE_69 & SYNTHESIZED_WIRE_76;


always@(posedge Clock4)
begin
	begin
	SYNTHESIZED_WIRE_74 <= SYNTHESIZED_WIRE_7;
	end
end


always@(posedge Clock4)
begin
	begin
	SYNTHESIZED_WIRE_75 <= SYNTHESIZED_WIRE_8;
	end
end


always@(posedge Clock4)
begin
	begin
	SYNTHESIZED_WIRE_76 <= SYNTHESIZED_WIRE_9;
	end
end


always@(posedge Clock4)
begin
	begin
	SYNTHESIZED_WIRE_71 <= SYNTHESIZED_WIRE_10;
```

```
	end
end


always@(posedge Clock4)
begin
	begin
	SYNTHESIZED_WIRE_72 <= SYNTHESIZED_WIRE_11;
	end
end


always@(posedge Clock4)
begin
	begin
	SYNTHESIZED_WIRE_79 <= SYNTHESIZED_WIRE_12;
	end
end


always@(posedge Clock4)
begin
	begin
	SYNTHESIZED_WIRE_70 <= SYNTHESIZED_WIRE_13;
	end
end


always@(posedge Clock4)
begin
	begin
	SYNTHESIZED_WIRE_73 <= SYNTHESIZED_WIRE_14;
	end
end

assign	SYNTHESIZED_WIRE_50 = SYNTHESIZED_WIRE_15 | SYNTHESIZED_WIRE_16 |
SYNTHESIZED_WIRE_17;

assign	SYNTHESIZED_WIRE_38 = SYNTHESIZED_WIRE_74 & SYNTHESIZED_WIRE_77;

assign	SYNTHESIZED_WIRE_41 = SYNTHESIZED_WIRE_78 & SYNTHESIZED_WIRE_73;

assign	SYNTHESIZED_WIRE_40 = SYNTHESIZED_WIRE_75 & SYNTHESIZED_WIRE_77;

assign	SYNTHESIZED_WIRE_16 = SYNTHESIZED_WIRE_73 & SYNTHESIZED_WIRE_77;

assign	SYNTHESIZED_WIRE_43 = SYNTHESIZED_WIRE_78 & SYNTHESIZED_WIRE_74;

assign	SYNTHESIZED_WIRE_42 = SYNTHESIZED_WIRE_76 & SYNTHESIZED_WIRE_77;

assign	SYNTHESIZED_WIRE_68 = SYNTHESIZED_WIRE_78 & SYNTHESIZED_WIRE_75;

assign	SYNTHESIZED_WIRE_67 = SYNTHESIZED_WIRE_71 & SYNTHESIZED_WIRE_77;
```

```
assign  SYNTHESIZED_WIRE_66 = SYNTHESIZED_WIRE_78 & SYNTHESIZED_WIRE_76;

assign  SYNTHESIZED_WIRE_65 = SYNTHESIZED_WIRE_72 & SYNTHESIZED_WIRE_77;

assign  SYNTHESIZED_WIRE_46 = SYNTHESIZED_WIRE_78 & SYNTHESIZED_WIRE_71;

assign  SYNTHESIZED_WIRE_45 = SYNTHESIZED_WIRE_79 & SYNTHESIZED_WIRE_77;

assign  SYNTHESIZED_WIRE_48 = SYNTHESIZED_WIRE_78 & SYNTHESIZED_WIRE_72;

assign  SYNTHESIZED_WIRE_47 = SYNTHESIZED_WIRE_78 & SYNTHESIZED_WIRE_79;

assign  SYNTHESIZED_WIRE_15 = SYNTHESIZED_WIRE_70 & SYNTHESIZED_WIRE_77;

assign  SYNTHESIZED_WIRE_39 = SYNTHESIZED_WIRE_78 & SYNTHESIZED_WIRE_70;

assign  SYNTHESIZED_WIRE_36 =  ~SYNTHESIZED_WIRE_78;

assign  SYNTHESIZED_WIRE_80 = SYNTHESIZED_WIRE_35 & SYNTHESIZED_WIRE_36;

assign  SYNTHESIZED_WIRE_35 =  ~SYNTHESIZED_WIRE_77;

assign  SYNTHESIZED_WIRE_58 = SYNTHESIZED_WIRE_38 | SYNTHESIZED_WIRE_39;

assign  SYNTHESIZED_WIRE_60 = SYNTHESIZED_WIRE_40 | SYNTHESIZED_WIRE_41;

assign  SYNTHESIZED_WIRE_62 = SYNTHESIZED_WIRE_42 | SYNTHESIZED_WIRE_43;

assign  bullet_p15 = SYNTHESIZED_WIRE_69 & SYNTHESIZED_WIRE_79;

assign  SYNTHESIZED_WIRE_54 = SYNTHESIZED_WIRE_45 | SYNTHESIZED_WIRE_46;

assign  SYNTHESIZED_WIRE_56 = SYNTHESIZED_WIRE_47 | SYNTHESIZED_WIRE_48;

assign  SYNTHESIZED_WIRE_69 =  ~Shoot;

assign  SYNTHESIZED_WIRE_78 =  ~Shift_R1;

assign  SYNTHESIZED_WIRE_17 =  ~Phasor2;

assign  SYNTHESIZED_WIRE_77 =  ~Shift_Left3;


\21mux b2v_mux_p1(
        .S(SYNTHESIZED_WIRE_80),
        .B(SYNTHESIZED_WIRE_50),
        .A(SYNTHESIZED_WIRE_70),
        .Y(SYNTHESIZED_WIRE_13));


\21mux b2v_mux_p11(
        .S(SYNTHESIZED_WIRE_80),
        .B(SYNTHESIZED_WIRE_52),
```

```
        .A(SYNTHESIZED_WIRE_71),
        .Y(SYNTHESIZED_WIRE_10));


\21mux b2v_mux_p13(
        .S(SYNTHESIZED_WIRE_80),
        .B(SYNTHESIZED_WIRE_54),
        .A(SYNTHESIZED_WIRE_72),
        .Y(SYNTHESIZED_WIRE_11));


\21mux b2v_mux_p15(
        .S(SYNTHESIZED_WIRE_80),
        .B(SYNTHESIZED_WIRE_56),
        .A(SYNTHESIZED_WIRE_79),
        .Y(SYNTHESIZED_WIRE_12));


\21mux b2v_mux_p3(
        .S(SYNTHESIZED_WIRE_80),
        .B(SYNTHESIZED_WIRE_58),
        .A(SYNTHESIZED_WIRE_73),
        .Y(SYNTHESIZED_WIRE_14));


\21mux b2v_mux_p5(
        .S(SYNTHESIZED_WIRE_80),
        .B(SYNTHESIZED_WIRE_60),
        .A(SYNTHESIZED_WIRE_74),
        .Y(SYNTHESIZED_WIRE_7));


\21mux b2v_mux_p7(
        .S(SYNTHESIZED_WIRE_80),
        .B(SYNTHESIZED_WIRE_62),
        .A(SYNTHESIZED_WIRE_75),
        .Y(SYNTHESIZED_WIRE_8));


\21mux b2v_mux_p9(
        .S(SYNTHESIZED_WIRE_80),
        .B(SYNTHESIZED_WIRE_64),
        .A(SYNTHESIZED_WIRE_76),
        .Y(SYNTHESIZED_WIRE_9));

assign  SYNTHESIZED_WIRE_52 = SYNTHESIZED_WIRE_65 | SYNTHESIZED_WIRE_66;

assign  SYNTHESIZED_WIRE_64 = SYNTHESIZED_WIRE_67 | SYNTHESIZED_WIRE_68;


endmodule


module schemtest3
```

```
(
	pin_name1,//sel3
	pin_name2,//sel2
	pin_name3,//sel1
	pin_name4,//sel0
	pin_name5,//in0
	pin_name6,
	pin_name7,
	pin_name8,
	pin_name9,
	pin_name10,
	pin_name11,
	pin_name12,
	pin_name13,
	pin_name14,
	pin_name15,
	pin_name16,
	pin_name17,
	pin_name18,
	pin_name19,
	pin_name20,//in15
	pin_name21//output
);


input wire	pin_name1;
input wire	pin_name2;
input wire	pin_name3;
input wire	pin_name4;
input wire	pin_name5;
input wire	pin_name6;
input wire	pin_name7;
input wire	pin_name8;
input wire	pin_name9;
input wire	pin_name10;
input wire	pin_name11;
input wire	pin_name12;
input wire	pin_name13;
input wire	pin_name14;
input wire	pin_name15;
input wire	pin_name16;
input wire	pin_name17;
input wire	pin_name18;
input wire	pin_name19;
input wire	pin_name20;
output wire	pin_name21;

wire	SYNTHESIZED_WIRE_0;

assign	SYNTHESIZED_WIRE_0 = 0;

\161mux	b2v_inst(
	.SEL3(pin_name1),
	.GN(SYNTHESIZED_WIRE_0),
	.IN0(pin_name5),
```

```
            .SEL0(pin_name4),
            .SEL1(pin_name3),
            .SEL2(pin_name2),
            .IN3(pin_name8),
            .IN2(pin_name7),
            .IN1(pin_name6),
            .IN6(pin_name11),
            .IN5(pin_name10),
            .IN4(pin_name9),
            .IN9(pin_name14),
            .IN8(pin_name13),
            .IN7(pin_name12),
            .IN11(pin_name16),
            .IN12(pin_name17),
            .IN10(pin_name15),
            .IN15(pin_name20),
            .IN14(pin_name19),
            .IN13(pin_name18),
            .OUT(pin_name21));


endmodule

module CMB(X_out, Y_out, X, Y);

  output X_out, Y_out;
  input X, Y;

  not notX(nX, X),
     notY(nY, Y);

  and (X_out, X, nY),
     (Y_out, Y, nX);

endmodule

module shifter(qa, qb, qc, qd, qe, qf, qg, qh, qA, qB, qC, qD, qE, qF, qG, clk2, X, Y, clk, rst);

  output qa, qb, qc, qd, qe, qf, qg, qh, qA, qB, qC, qD, qE, qF, qG, clk2 ;

  input X,Y, clk, rst;

  wire qa, qb, qc, qd, qe, qf, qg, qh, qA, qB, qC, qD, qE, qF, qG;
  wire da, db, dc, dd, de, df, dg, dh, dA, dB, dC, dD, dE, dF, dG;

  wire D7reset,D6reset,D5reset,D4reset,D3reset,D2reset,D1reset;
  wire d1_rst,d2_rst,d3_rst,d4_rst,d5_rst,d6_rst,d7_rst;

  DF0 FFclk(qaC, nqaC, daC, clk ,rst);
  not notA(daC, qaC);

  DF0 FFclk1(qbC, nqbC, dbC, clk ,rst);

  xor xor1(outX, qaC, qbC);
```

```
not not1(dbC, outX);

DF0 FFclk2(qcC, nqcC, dcC, clk ,rst);

and and1(out1, qcC, qbC),
and2(out2, qaC, qcC),
and3(out3, nqcC, nqbC, nqaC);

or or1(dcC, out1, out2, out3);

assign clk2 = qcC;

// Y input
 DF0 FFa(qa, nqa, da, clk2, D7reset);

        DF0 FFb(qb, nqb, db, clk2, D6reset);

        DF0 FFc(qc, nqc, dc, clk2, D5reset);

        DF0 FFd(qd, nqd, dd, clk2, D4reset);

 DF0 FFe(qe, nqe, de, clk2, D3reset);

        DF0 FFf(qf, nqf, df, clk2, D2reset);

        DF0 FFg(qg, nqg, dg, clk2, D1reset);

        DF0 FFh(qh, nqh, dh, clk2, rst);


//X input

 DF0 FFA(qA, nqA, dA, clk, D1reset);

        DF0 FFB(qB, nqB, dB, clk, D2reset);

        DF0 FFC(qC, nqC, dC, clk, D3reset);

        DF0 FFD(qD, nqD, dD, clk, D4reset);

 DF0 FFE(qE, nqE, dE, clk, D5reset);

        DF0 FFF(qF, nqF, dF, clk, D6reset);

        DF0 FFG(qG, nqG, dG, clk, D7reset);

//input to X
buf buf0( dA , X);

//input to Y
buf buf1 (da , Y);
```

```
  // X input > A B C D E F G
  //          g f e d c b a < input Y

  or  ord1(D1reset, d1_rst, rst);
  or  ord2(D2reset, d2_rst, rst);
  or  ord3(D3reset, d3_rst, rst);
  or  ord4(D4reset, d4_rst, rst);
  or  ord5(D5reset, d5_rst, rst);
  or  ord6(D6reset, d6_rst, rst);
  or  ord7(D7reset, d7_rst, rst);


  CMB Comb1(outAg_X, outAg_Y , qA ,qg);
  and andrst1(d1_rst, qA,qg);
  buf buf2( dB , outAg_X); // output from comb logic shift right to dB
  buf buf3 (dh , outAg_Y);

  CMB Comb2(outBf_X, outBf_Y , qB ,qf);
  and andrst2(d2_rst, qB,qf);
  buf buf5 (dg , outBf_Y); // output from comb logic shift left to dg
  buf buf4 (dC , outBf_X); // output from comb logic shift right to dC

  CMB Comb3(outCe_X, outCe_Y , qC ,qe);
  and andrst3(d3_rst, qC,qe);
  buf buf6( df , outCe_Y);
  buf buf7 (dD , outCe_X);

  CMB Comb4(outDd_X, outDd_Y , qD ,qd);
  and andrst4(d4_rst, qD,qd);
  buf buf9 (de , outDd_Y);
  buf buf8 (dE , outDd_X);

  CMB Comb5(outEc_X, outEc_Y , qE ,qc);
  and andrst5(d5_rst, qE, qc);
  buf buf11 (dd , outEc_Y);
  buf buf10 (dF , outEc_X);

  CMB Comb6(outFb_X, outFb_Y , qF ,qb);
  and andrst6(d6_rst, qF,qb);
  buf buf12 (dc , outFb_Y);
  buf buf13 (dG , outFb_X);

  //test Y show up right away
  CMB Comb7(outGa_X, outGa_Y , qG ,qa);
  and andrst7(d7_rst, qG,qa);

  buf buf14 (db , outGa_Y);

endmodule


module Count_To_Sixteen (Out,q3,q2,q1,q0,clk,rst);

  input clk,rst;
```

```
  output[15 :0] Out;
  //output[6  :0] R
  output q3,q2,q1,q0;

  and a30(d30,q3,nq2);
  and a31(d31,q3,q2,nq1);
  and a32(d32,q3,q2,nq0);
  and a33(d33,nq3,q2,q1,q0);
  or o3(d3,d30,d31,d32,d33);

  and a20(d20,q2,nq1);
  and a21(d21,q2,nq0);
  and a22(d22,nq2,q1,q0);
  or o2(d2,d20,d21,d22);


  xor x0(d1,q1,q0);

  nand aOut0 (Out[0], nq3,nq2,nq1,nq0);
  nand aOut1 (Out[1], nq3,nq2,nq1,q0);
  nand aOut2 (Out[2], nq3,nq2,q1,nq0);
  nand aOut3 (Out[3], nq3,nq2,q1,q0);
  nand aOut4 (Out[4], nq3,q2,nq1,nq0);
  nand aOut5 (Out[5], nq3,q2,nq1,q0);
  nand aOut6 (Out[6], nq3,q2,q1,nq0);
  nand aOut7 (Out[7], nq3,q2,q1,q0);
  nand aOut8 (Out[8], q3,nq2,nq1,nq0);
  nand aOut9 (Out[9], q3,nq2,nq1,q0);
  nand aOut10 (Out[10], q3,nq2,q1,nq0);
  nand aOut11 (Out[11], q3,nq2,q1,q0);
  nand aOut12 (Out[12], q3,q2,nq1,nq0);
  nand aOut13 (Out[13], q3,q2,nq1,q0);
  nand aOut14 (Out[14], q3,q2,q1,nq0);
  nand aOut15 (Out[15], q3,q2,q1,q0);

  DF0 FF0(q0, nq0, nq0, clk, rst);
          DF0 FF1(q1, nq1, d1, clk, rst);
          DF0 FF2(q2, nq2, d2, clk, rst);
  DF0 FF3(q3, nq3, d3, clk, rst);

endmodule


module Eight_Bit_LFSR(q0,q1,q2,q3,q4,q5,q6,q7,L,clk,rst);

  //output [7:0] Out;
  output q0,q1,q2,q3,q4,q5,q6,q7;

  input L,clk, rst;

  wire d0,d1,d2,d3,d4,d5,d6,d7,q0,q1,q2,q3,q4,q5,q6,q7,clk,rst;
  //wire [7:0] Out;

  reg [7:0] In = 00000001;
```

```
  DF0 FF0(q0, nq0, d0, clk, rst);
  DF0 FF1(q1, nq1, d1, clk, rst);
  DF0 FF2(q2, nq2, d2, clk, rst);
  DF0 FF3(q3, nq3, d3, clk, rst);
  DF0 FF4(q4, nq4, d4, clk, rst);
  DF0 FF5(q5, nq5, d5, clk, rst);
  DF0 FF6(q6, nq6, d6, clk, rst);
  DF0 FF7(q7, nq7, d7, clk, rst);

  not not0(nL,L);

  and and0(sel1,L,In[0]);
  and and1(sel1_2,nL,q7);
  or or0(d0,sel1,sel1_2);

  and and2(sel2,L,In[1]);
  and and3(sel2_2,nL,q0);
  or or1(d1,sel2,sel2_2);

  xor xorA(xorInA,q1,q7);
  and andA(selA,L,In[2]);
  and andA0(selA_1,nL,xorInA);
  or orA(d2,selA,selA_1);

  and and6(sel4,L,In[3]);
  and and7(sel4_2,nL,q2);
  or or3(d3,sel2,sel2_2);

  xor xor0(xorIn0,q3,q7);
  and and8(sel5,L,In[4]);
  and and9(sel5_1,nL,xorIn0);
  or or4(d4,sel5,sel5_1);

  xor xor1(xorIn1,q4,q7);
  and and10(sel6,L,In[5]);
  and and11(sel6_1,nL,xorIn1);
  or or5(d5,sel6,sel6_1);

  xor xor2(xorIn2,q5,q7);
  and and12(sel7,L,In[6]);
  and and13(sel7_1,nL,xorIn2);
  or or6(d6,sel7,sel7_1);


  and and14(sel8,L,In[7]);
  and and15(sel8_2,nL,q6);
  or or7(d7,sel8,sel8_2);

endmodule

module DF0(q, qBar, D, clk, rst);
  input D, clk, rst;
  output q, qBar;
  reg q;
```

```
  not n1 (qBar, q);

  always@ (posedge rst or posedge clk)
  begin
    if(rst)
      q = 0;

    else
      q = D;

  end

endmodule
```

Gal chip code

```
module timer (FIVE_b, qa, qb, qc, qd, clk, rst, en);

        output  qd /* synthesis LOC= "P23" */ ;
        output  qc /* synthesis LOC= "P22" */ ;
        output  qb /* synthesis LOC= "P21" */  ;
        output  qa /* synthesis LOC= "P20" */  ;
        output  FIVE_b /* synthesis LOC= "P4" */ ;
        input clk, rst,en;

        DF0 FFa(qa, nqa, da, clk, rst);

        DF0 FFb(qb, nqb, db, clk, rst);

        DF0 FFc(qc, nqc, dc, clk, rst);

        DF0 FFd(qd, nqd, dd, clk, rst);


  and and0 (da1, qb, qc, qd, en);
  and and2(da2, qa, en);
  or or0 (da, da1, da2);

  and and1 (db1, qb, nqc, en);
  and and3 (db2, qb, nqd, en);
  and and4 (db3, nqb, qc, qd, en);
  or or1 (db, db1, db2, db3);

  and and5(dc1,nqa,nqc,qd, en);
```

```
  and and6(dc2,qc,nqd, en);
  or or2(dc,dc1,dc2);

  and andA(dd1, qa, en);
  and andC(dd2, nqd, en);
  or or3(dd, dd2, dd1);

  and and9(nFIVE_b, qa,nqb,nqc,qd,en);

  not notF(FIVE_b, nFIVE_b);

endmodule

module DF0(q, qBar, D, clk, rst);
  input D, clk, rst;
  output q, qBar;

  reg q;

  not n1 (qBar, q);

  always@ (posedge rst or posedge clk)
  begin
    if(rst)
      q = 0;

    else
      q = D;

  end

endmodule
```