Eletronic Systems of Computers

Project Report of Memory Game

Group 2
94336 - Guilherme Trindade
97142 - Hélio Júnior

Professor:
José Teixeira de Sousa

January 29, 2020

Contents

List of Tables

List of Figures

1 Introduction

This proposal is to present the work to be done at the Computer Electronic Systems (SEC) course of the Master's degree in Electronic Engineering (MEE), the main objective is to development the memory game, to be performed on Field-Programmable Gate Array device (FPGA).

The project will be developed using a hardware description language (verilog), following the structural logic description model, and the control part of the system will be implemented using assembly languague of the PicoVersat processor.

2 Block Diagram

The project block diagram is shown in Fig. 1. The latest version of the project is for the entire system to work with the button interface, to enter the game keys according to the correct sequence.

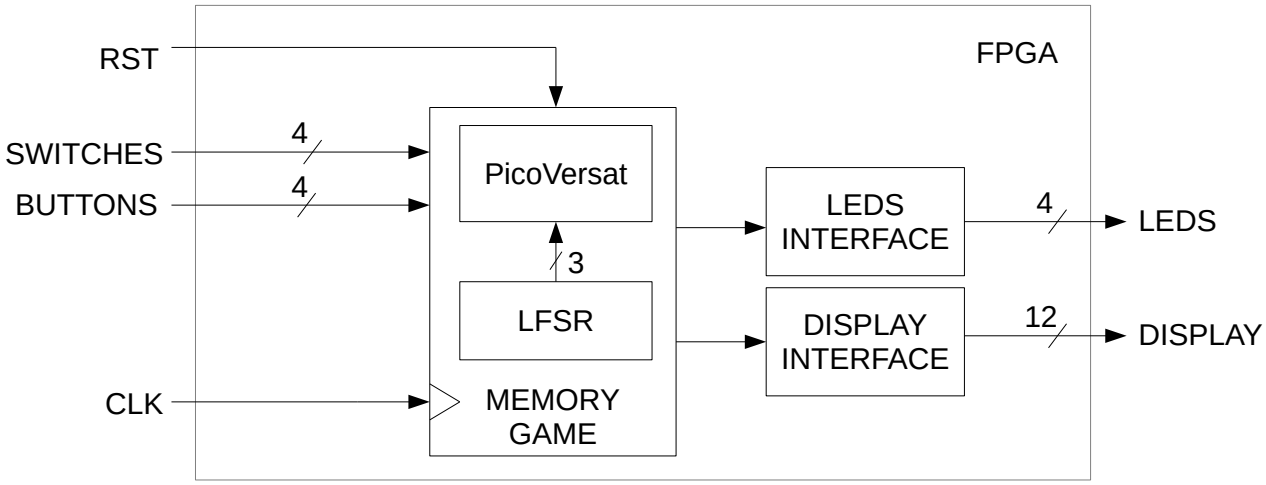

Figure 1: Block Diagram

3 Interface Signals

The interface signals of the project are described in Table 1.

Name	Direction	Peripheral	Description
clk	IN	FPGA clock	Clock signal.
rst	IN	Button driver	Reset game signal.
Buttons	IN	Button driver	Buttons to play the game.
Switches	IN	Switch driver	Implemented but not used.
Leds	OUT	LED driver	Represent the game board.
Displays	OUT	Display driver	Represents the game score and level.

Table 1: Interface signals.

4 Peripherals

A simple System on Chip (SoC) including picoVersat, a program and data memory, and the five peripherals attached to the data bus is shown in Figure 2.

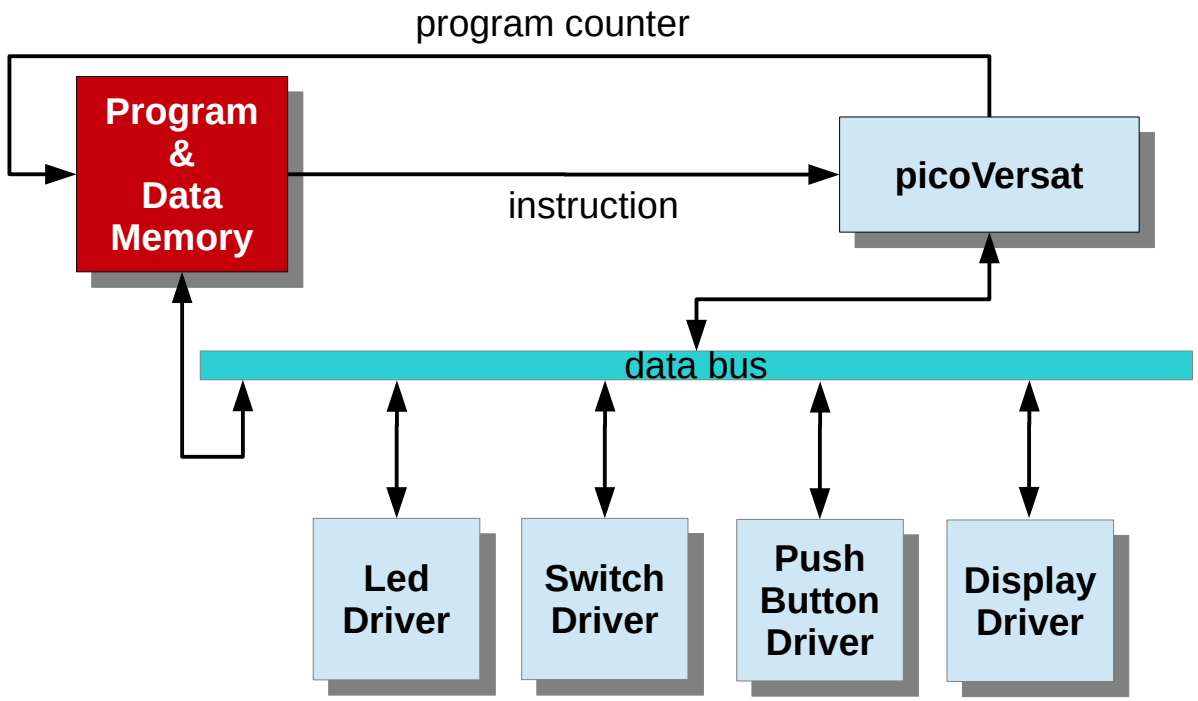

Figure 2: PicoVersat SoC with five peripherals

Refer to the memory map in section 5 to check the base addresses of the peripherals.

4.1 Display driver

This peripheral writes to the address from the Table 2, and the driver decodes which display should be light up, because the data from the 4 displays are connect in parallel and its just necessary to change the value of the anode.

Name	Address	Bits	Description
DISPLAY0	0x820	11-0	Data bits [7:0] and anode bit [8].
DISPLAY1	0x822	11-0	Data bits [7:0] and anode bit [9].
DISPLAY2	0x824	11-0	Data bits [7:0] and anode bit [10].
DISPLAY3	0x826	11-0	Data bits [7:0] and anode bit [11].

Table 2: Description of the base address from display peripheral.

4.2 Led driver

This peripheral shown in Table 3 is a driver to output the value of the LEDs, depending on the value written from the address in the table 3, the driver decodes which LEDs should be lighted.

Name	Address	Bits	Description
LED_BASE	0x828	7-0	Each bit corresponds a one LED.

Table 3: Description of the base address from led peripheral.

4.3 Switch driver

This peripheral shown in Table 4 is a driver for reading the value of the switches, depending on the value read from the address in table 4, the driver saves the status of each switch for PicoVersat use to control the system.

Name	Address	Bits	Description
SWITCH_BASE	0x82A	6-0	Each bit corresponds a one switch.

Table 4: Description of the base address from switch peripheral.

4.4 Button driver

This peripheral shown in Table 5 drives the value of the buttons and deals with the debounce problem.

Name	Address	Bits	Description
BUTTON_BASE	0x82C	3-0	Buttons to play the game.

Table 5: Description of the base address from button peripheral.

4.5 LFSR Driver

This peripheral shown in Table 9 can be used to geneate random sequences with a range of 0 to 7.

Name	Address	Bits	Description
LFSR_BASE	0x82E	2-0	Linear-feedback shift register.

Table 6: Description of the base address from LFSR peripheral.

5 Memory Map

The memory map of the system, as seen by picoVersat programs, is given in Table 7.

Mnemonic	Address	Read/Write	Read Latency	Description
REGF_BASE	0x800	Read+Write	0	Register file peripheral.
CPRT_BASE	0x812	Write only	NA	Debug printer periheral.
EXT_BASE	0x820	Write only	NA	External interface.
DISPLAY0	0x820	Read+Write	0	Display0 peripheral.
DISPLAY1	0x822	Read only	0	Display1 peripheral.
DISPLAY2	0x824	Read only	0	Display2 peripheral.
DISPLAY3	0x826	Write only	0	Display3 peripheral.
LED_BASE	0x828	Write only	0	LED peripheral.
SWITCH_BASE	0x82A	Write only	0	SWITCH peripheral.
BUTTON_BASE	0x82C	Write only	0	BUTTON peripheral.
LFSR_BASE	0x82E	Read only	1	LFSR peripheral.
PROG_BASE	0x0	Read+Write	1	User programs and data.

Table 7: Memory map base addresses

6 Program Description

The state diagram of the Fig. 3, it's the proposal system to implement, this diagram is intend to give an abstract description of the behavior of tthe system that will be running in the PicoVersat processor. This behavior is represented as a series of events that occur in the three states, and by this events the program will flow the schematic doing the value of the events.

The idle state has the main objective of reset all the important variables of the system, after reseting the value of each led, score and level. The system waits until the user pressed the condition of start to initiate the game and call the show state.

The show state is where a new level is generated, a new random sequence is shown in the led's. As the level increases the time between each led blinking increases too.

The game state have the responsibility of checking each move and increment the respective score and level. This state will show the value of each level and score. After the player win the game or loose the system will return to the idle mode showing a message that the player have won the game.

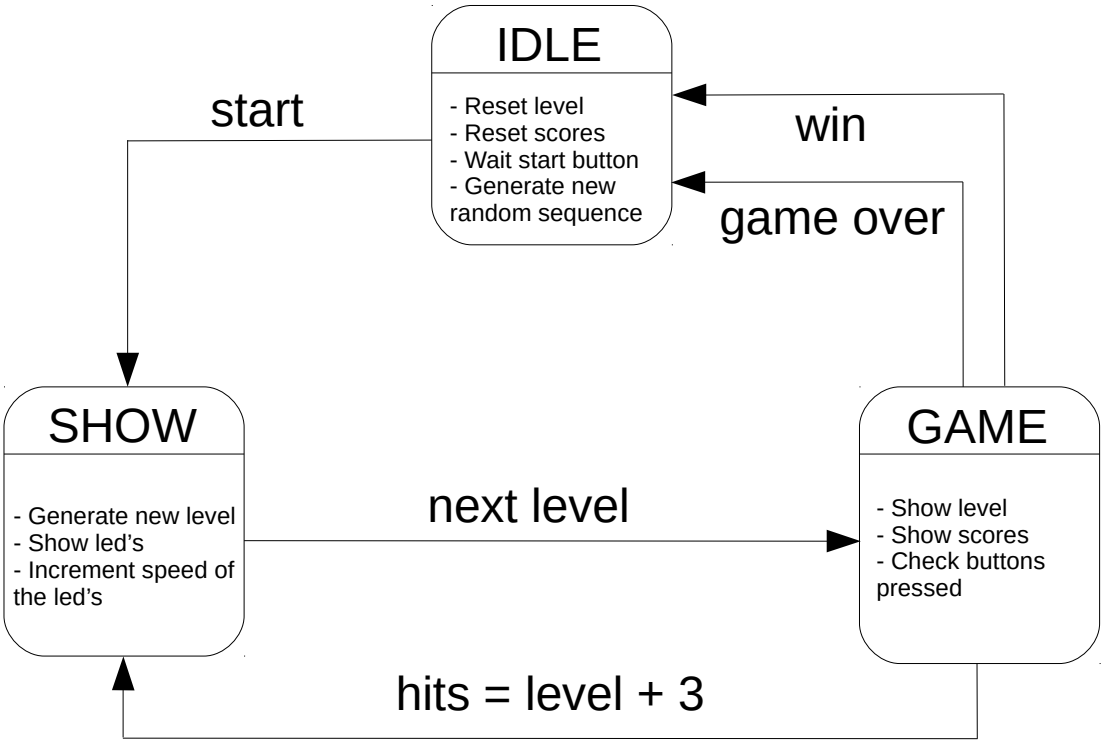

Figure 3: State diagram

7 Implementation

7.1 External interface

Communication with peripherals was build on top of Picoversat external parrallel interface, as shown in Figure 4. PicoVersat coordinates if an exchange of data is to be made with an external periferal, being them read from peripheral or write to. An external decoder takes responsibility to select which peripheral is desired by program, taking in consideration memory mapping from Table 7. From external decoder mapping the data path will be build according to program needs, this aproccach makes Picoversat implemtation inmutable, and allows to peripherals and data path's to be build externally to Picoversat's.

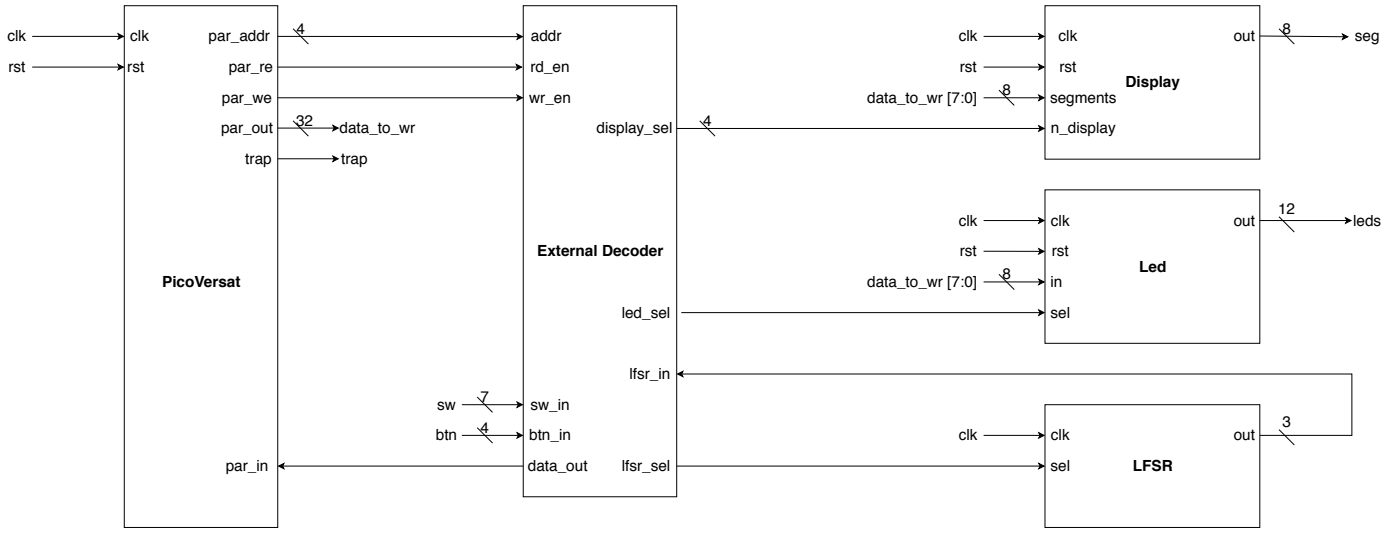

Figure 4: External interface block diagram.

7.2 Description of the display peripheral

For each of the four digits to appear bright and continuously illuminated, all four digits should be driven once every 1 to 16ms. For example, for a 50 KHz clock of the FPGA, the entire display would be refreshed once every 16ms, and each digit would be illuminated for 1/4 of the refresh cycle, or 4ms. Figure 5 shows an example timing diagram for a four-digit seven-segment controller.

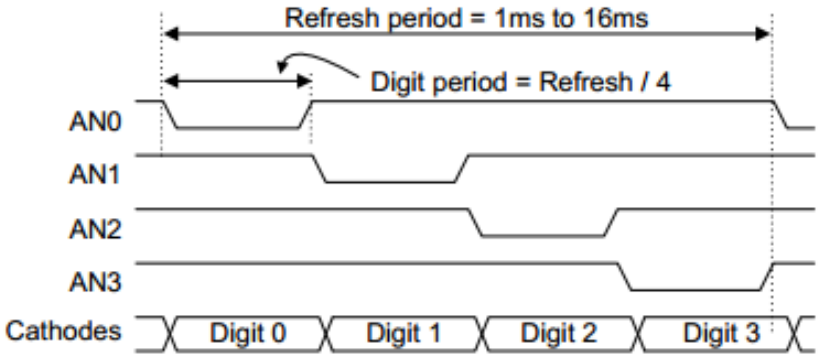

Figure 8. Multiplexed 7seg display timing

Figure 5: 7 segment display timing.

To generate all the configurations possible in the seven segment display, in the Table 8 is the value of each bit. Each of the displays are assembled with a common cathode so to lighting the respective led of each display the value to be wrtitten on the port it is a zero.

Content	Seg [7:0]
0	11000000
1	11111001
2	10100100
3	10110000
4	10011001
5	10010010
6	10000010
7	11111000
8	10000000
9	10010000
E	10011001

Table 8: Content available to output on displays.

Displays driver (Figure 6) has four registers (flip-flop type D) with the ability to store the values arriving from PicoVersat data bus, it is possible to update each display individually by requesting for it's specific address . A seven-segment display is activated once every 1 ms, for which an 18-bit counter has been implemented. This counter is incremented with each clock cycle, when the counter reaches the desired value, a control signal is generated to notify a four-bit shift register. Every time this control signal takes the high value, the shift register is shifted slightly to the left to cover the requirement for all four screens being updated in a 16 ms cycle. The four-bit register can be seen in Figure 6 as a multiplexer, according to word of four bits the shift register will choose one of the inputs.

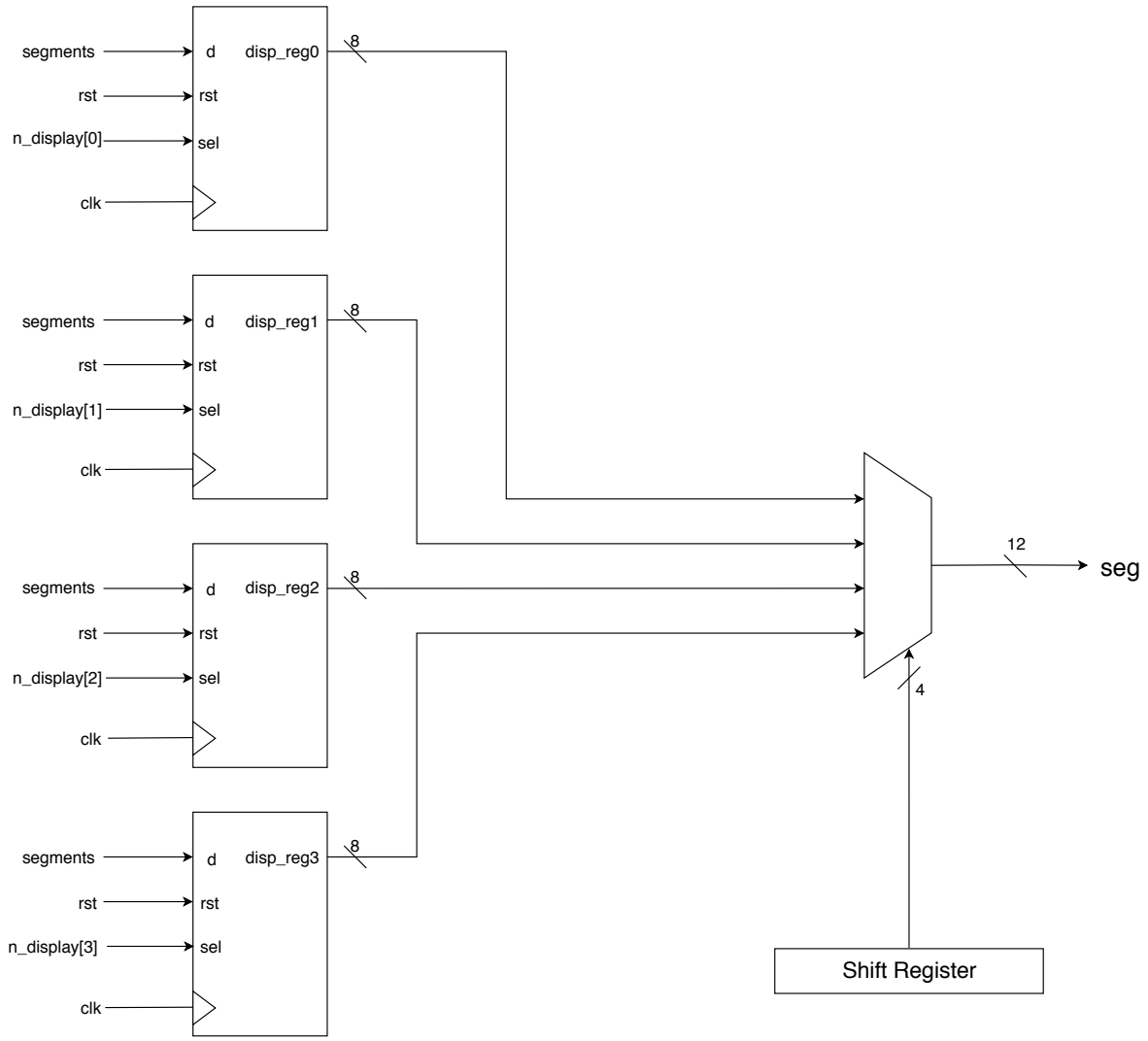

Figure 6: Block diagram of the display implementation.

7.3 Description of the LFSR peripheral

Linear Feedback Shift Register (LFSR) shown in Figure 7, it is a shift register whose input bit is a linear function of its previous state. The output from a standard shift register is fed back into its input in such a way as to cause the function to endlessly cycle through a sequence of patterns. Due the fact these circuit is simple to construct and are useful on our application, it is used to generate the random sequences to the memory game.

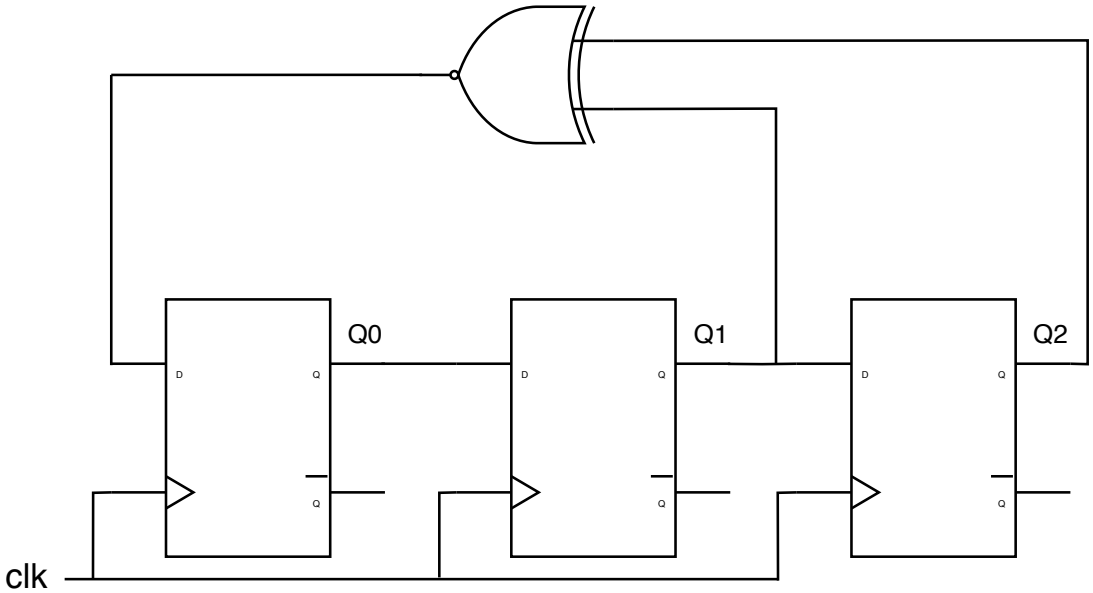

Figure 7: LFSR peripheral.

The LFSR have 3 bits and the Table 9 shows the generation of multiple sequences contained in a range from [0:7]. During the development of this project we find some issues in the generation key of the game. To resolve this problem we create a LUT with a size of 8 containing shuffled values (1,2,4,8) of the game LEDS, finally we use the values produce by the LFSR to index this LUT to generate different sequences for the game.

clk	Q_0	Q_1	Q_2
-	0	0	1
1	1	0	0
2	0	1	0
3	1	0	1
4	1	1	0
5	1	1	1
6	0	1	1
7	0	0	1

Table 9: Truth table of the circuit LFSR.

8 Conlusion and results

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	143	1,920	7 %
Number of 4 input LUTs	901	1,920	46 %
Number of occupied Slices	512	960	53 %
Number of Slices containing only related logic	512	512	100 %
Number of Slices containing unrelated logic	0	512	0 %
Total Number of 4 input LUTs	929	1,920	48 %
Number used as logic	869	-	
Number used as a route-thru	28	-	
Number used as 16x1 RAMs	32	-	
Number of bonded IOBs	34	83	40 %
Number of RAMB 16s	4	4	100 %
Number of BUFGMUXs	1	24	4 %
Average Fanout of Non-Clock Nets	3.55	-	

Table 10: Device Utilization Summary

Appendices

A Header file included on the source code

```
#define D0 2080
#define D1 2082
#define D2 2084
#define D3 2086
#define LED 2088
#define SWITCHES 2090
#define BUTTONS 2092
#define LFSR 2094
#define TRAP 2098


#define TODISP0 0xC0
#define TODISP1 0xF9
#define TODISP2 0xA4
#define TODISP3 0xB0
#define TODISP4 0x99
#define TODISP5 0x92
#define TODISP6 0x82
#define TODISP7 0xF8
#define TODISP8 0x80
#define TODISP9 0x90
#define TODISPE 0x86
#define TODISPNILL 0xFF
```

B Source code running on PicoVersat

```
#include "memorygame.h"
#include "assign.h"

int d0 = D0;
int d1 = D1;
int d2 = D2;
int d3 = D3;
int leds = LED;
int btn = BUTTONS;
int lfsr = LFSR;
int t = TRAP;

void init(int* disp0,int* disp1,int* disp2, int* disp3){
    assign(TODISP5, disp3);
    assign(TODISPNILL, disp2);
    assign(TODISPNILL, disp1);
    assign(TODISPNILL, disp0);

    return;
}

int main (){
    int* disp0 = (int*)d0;
    int* disp1 = (int*)d1;
    int* disp2 = (int*)d2;
    int* disp3 = (int*)d3;
    int* t = (int*)t;
    int* b = (int*)btn;
    int* l = (int*)leds;
    int* lr = (int*)lfsr;
    int a[8] = {1,8,4,2,4,2,1,8};
    int numDisplay[10] = {TODISP0, TODISP1, TODISP2, TODISP3, TODISP4,\
```

```
                        TODISP5, TODISP6, TODISP7, TODISP8, TODISP9};
    int key[8] = {0,0,0,0,0,0,0,0};
    int state = 0;
    int i,j = 0;
    int level = 1;
    int hits = 0;
    int totalHits = 0;
    int index = 0;
    int button = 0;
    int speed = 3000000;
    while (1){
        switch (state) {
            case 0:
                init(disp0, disp1, disp2, disp3);
                for(i = 0;i<8;i++){
                    key[i] = a[*lr];
                }
                while(*b != 9);
                assign(TODISP0, disp3);
                assign(TODISP0, disp2);
                state = 1;
                break;
            case 1:
                assign(TODISPNILL, disp1);
                assign(numDisplay[level], disp0);
                for(i=0;i<(level + 3);i++){
                    assign(0,l);
                    for(j=0;j<speed;j++);
                    assign(key[i],l);
                    for(j=0;j<speed;j++);
                }
                assign(0,l);
                state = 2;
                break;
            case 2:
                hits = 0;
                while(hits < (level + 3)){
                    while(button == 0){
                        button = *b;
                    }
                    for(j=0;j<1000000;j++);
                    if(button != 1 && button != 2 && button != 4 && button != 8) {
                        continue;
                    }
                    if(button != key[hits]) {
                        assign(TODISPNILL,disp0);
                        assign(TODISPNILL,disp3);
                        assign(TODISP0,disp1);
                        assign(TODISP6,disp2);
                        for(j=0;j<10000000;j++);
                        level = 1;
                        speed = 3000000;
                        totalHits = 0;
                        button = 0;
                        state = 0;
                        break;
                    }
                    hits++;
                    totalHits++;
                    button = 0;
                    if(totalHits < 10){
                        assign(TODISP0, disp3);
                        assign(numDisplay[totalHits], disp2);
                    }
                    else if(totalHits >= 10 && totalHits < 20){
                        assign(TODISP1, disp3);
                        index = totalHits - 10;
                        assign(numDisplay[index], disp2);
                    }
```

```
102                     else if(totalHits >= 20 && totalHits < 30){
103                         assign(TODISP2, disp3);
104                         index = totalHits - 20;
105                         assign(numDisplay[index], disp2);
106                     }
107                     else if(totalHits >= 30){
108                         assign(TODISP3, disp3);
109                         index = totalHits - 30;
110                         assign(numDisplay[index], disp2);
111                         for(j=0;j<10000000;j++);
112                         assign(TODISPE,disp1);
113                         assign(TODISP6,disp0);
114                         for(j=0;j<10000000;j++);
115                         level = 1;
116                         speed = 3000000;
117                         totalHits = 0;
118                         button = 0;
119                         state = 0;
120                         break;
121                     }
122                 }
123                 if(hits == (level + 3)){
124                     level++;
125                     speed -= 500000;
126                     state = 1;
127                     break;
128                 }
129             default:
130                 break;
131         }
132     }
133     return 0;
134 }
```