Eletronic Systems of Computers

Project Report of Memory Game

Group 2
94336 - Guilherme Trindade
97142 - Hélio Júnior

Professor:
José Teixeira de Sousa

January 27, 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 PS / 2 interface, using a keyboard to enter the game keys according to the correct sequence. But first, we will try to implement the game algorithm with the switch interface, because it's easier for us if the project flow goes wrong, we have something to demonstrate in the final project delivery.

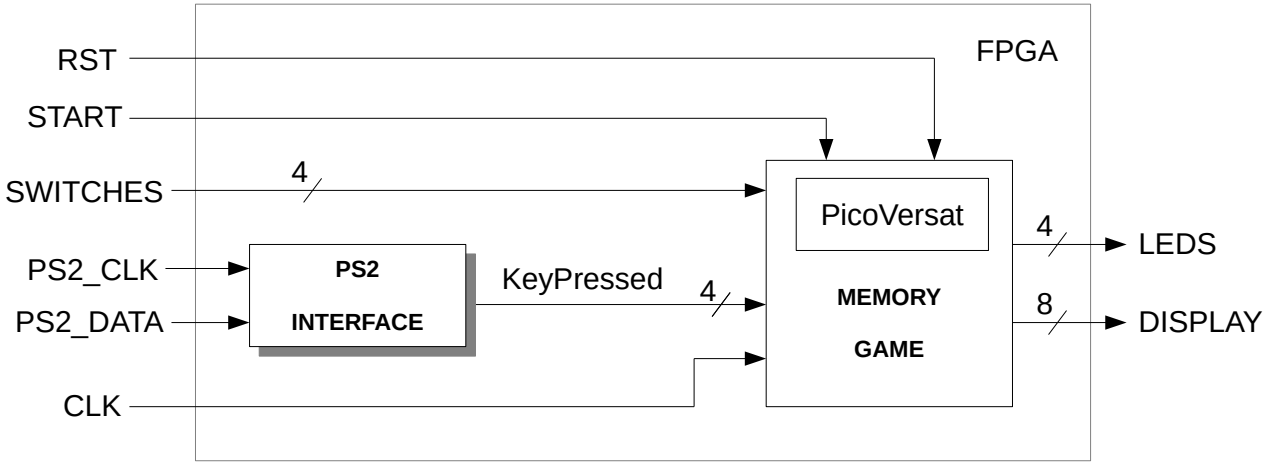

Figure 1: Block Diagram

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.
start	IN	Button driver	Start game signal.
PS/2 clk	IN	PS/2 driver	PS/2 clock signal.
PS/2 data	IN	PS/2 driver	PS/2 data signal.
LEDs	OUT	LED driver	Represent the game board.
Display	OUT	Display driver	Represents the game score and level.
Switches	IN	Display driver	Keys to input instead of using the keyboard.

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.

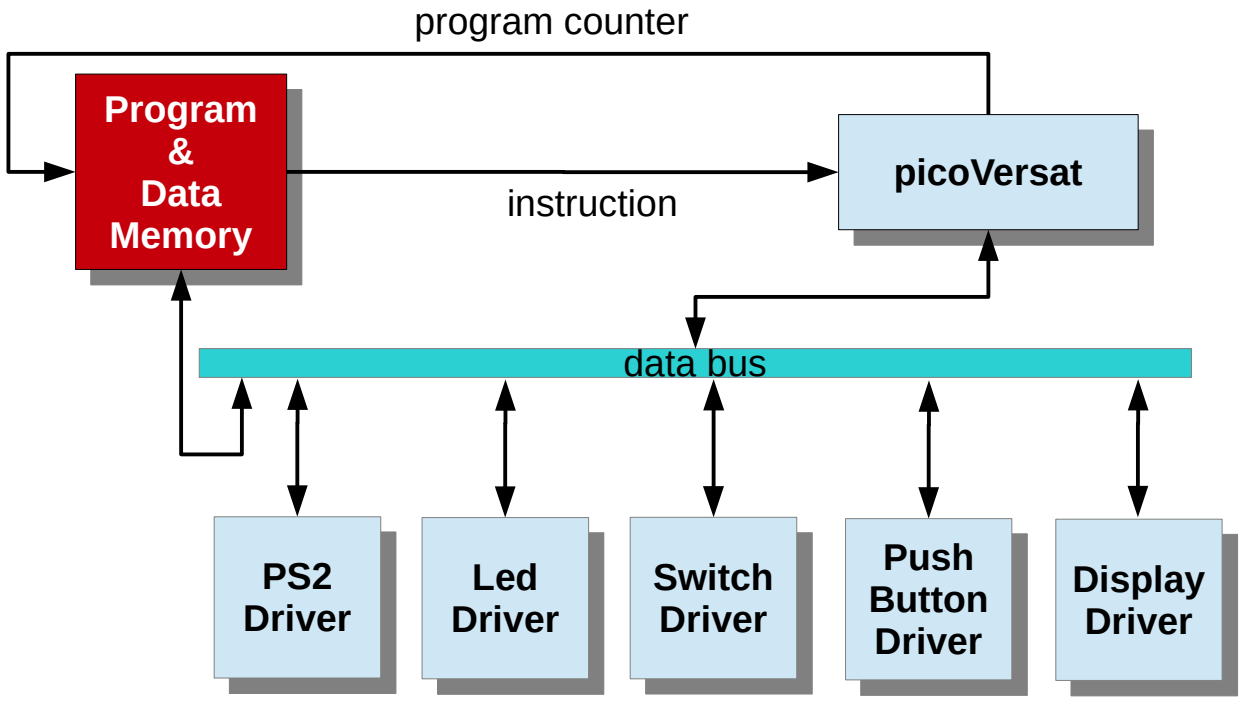

Figure 2: PicoVersat SoC with five peripherals

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

4.1 PS/2 Driver

This peripheral can be used to drive a PS/2 interface and using a keyboard to input the data, this driver makes the management and processing of the incoming data, decoding the pressed key.

Name	Address	Bits	Description
PS2_BASE	610	1-0	PS2 clk and PS2 data.

Table 2: Description of the base address from PS/2 peripheral.

4.2 Led driver

This peripheral is a driver to output the value of the LEDs, depending on the value written from the address in the table 2, the driver decodes which LEDs should be lighted.

Name	Address	Bits	Description
LED_BASE	602	3-0	Each bit corresponds a one LED.

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

4.3 Switch driver

This peripheral 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	604	3-0	Each bit corresponds a one switch.

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

4.4 Push-button driver

This peripheral drives the value of the buttons and deals with the debounce problem.

Name	Address	Bits	Description
BUTTON_BASE	606	1-0	Reset and start button.

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

4.5 Display driver

This peripheral writes to the address from the table 5, 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
DISPLAY_BASE	608	11-0	Data bits [7:0] and anode bits [11:8].

Table 6: Description of the base address from display 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	0x200	Read+Write	0	Register file peripheral
CPRT_BASE	0x258	Write only	NA	Debug printer periheral
LED_BASE	0x25A	Read+Write	0	Led peripheral
SWITCH_BASE	0x25C	Read only	0	Switch peripheral
BUTTON_BASE	0x25E	Read only	0	Button peripheral
DISPLAY_BASE	0x260	Write only	0	Display peripheral
PS2_BASE	0x262	Read only	0	PS2 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 and the score of the player, waits until the start button is pressed to initiate the game and call the state game.

The game state is where the game occurs, this state has the main tasks such as sending the keystrokes to the verification state, if the verification flag is set to zero, the state will generate a new level and a new random sequence and the game will continue.

The check state have the responsibility of checking each move and set the value of the check flag according to the value of received keys. Due the check flag the state will show the value of each level and score, if the flag last level is set to one the check state will return to the idle mode and send a message that the player have won the game.

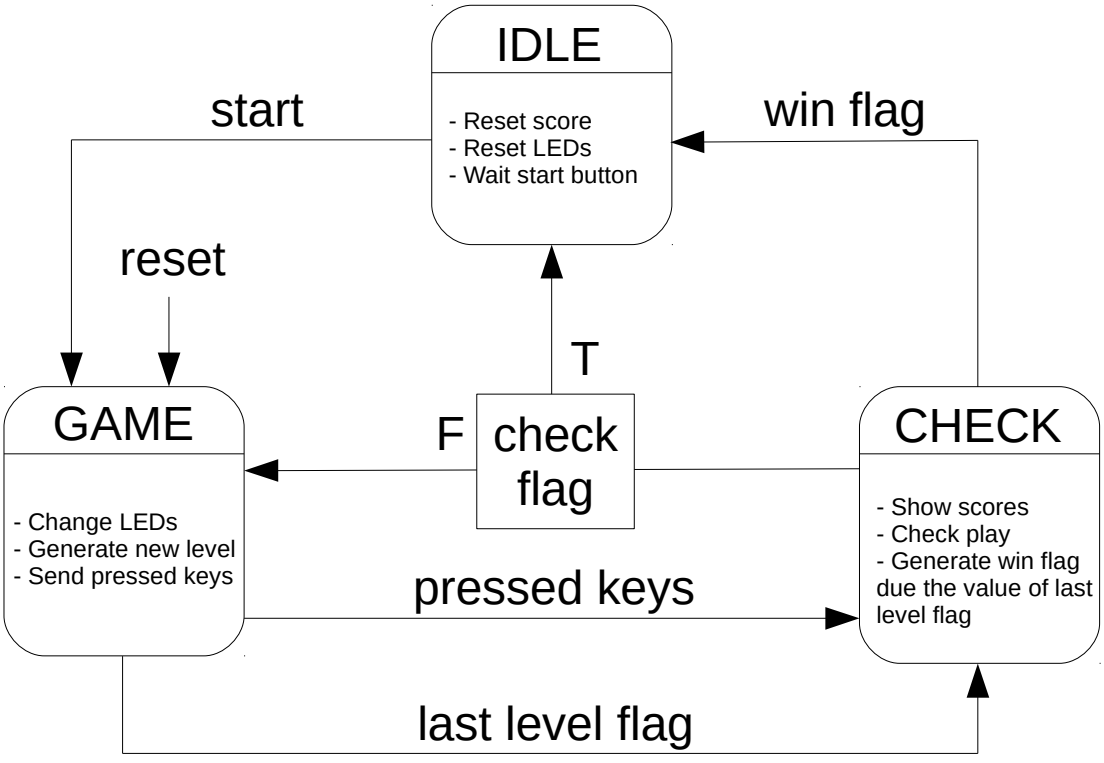

Figure 3: State diagram

7 Implementation

8 Results

9 Conclusions

During the development of this project, many issues were encountered either software and hardware related. This is not ideal due to the fact that it takes too much memory. But since we didn't found a way of getting it to work properly and reliably, the less efficient but working method was used. During the requirement definition of the project, it was discussed if the TIVA had enough space for a full framebuffer. Other hardware related problems were encountered, the GPIO expanders had some anomalous behavior on the MISO line that prevented the GPIO read funcions from working properly. Also it was later discovered that the input impedance of the ADCs was too high and it was preventing it from reading the proper values. The amount of time spent debugging the software and hardware also prevented from testing the I2C interface, although being quite important for the EGSE main purpose, the core of the driver was implemented and since a working version of the driver was already implemented on the non-FreeRTOS version of the EGSE, the rest of the I2C testing and debugging would be not complicated in the future in the scope of the ISTSAT-1 project.

Even after these drawbacks, we considered this project an excellent learning experience. After this the group understands and it's able to design, implement, test and manufacture programmable and completely dedicated electronic systems based on microcontrollers and configurable logic electronic devices (FPGA).

Appendices

A 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 sw = SWITCHES;
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;
}

/*void delay(int time){
```

```
24          int i;
25          for(i=0;i<time;i++); // 5000000
26          return;
27      }*/
28
29      int main (){
30          int* disp0 = (int*)d0;
31          int* disp1 = (int*)d1;
32          int* disp2 = (int*)d2;
33          int* disp3 = (int*)d3;
34          int* t = (int*)t;
35          int* b = (int*)btn;
36          int* s = (int*)sw;
37          int* l = (int*)leds;
38          int* lr = (int*)lfsr;
39          int a[8] = {1,8,4,2,4,2,1,8};
40          int numDisplay[10] = {TODISP0, TODISP1, TODISP2, TODISP3, TODISP4,\
41                                TODISP5, TODISP6, TODISP7, TODISP8, TODISP9};
42          int key[8] = {0,0,0,0,0,0,0,0};
43          int state = 0;
44          int i,j = 0;
45          int level = 1;
46          int hits = 0;
47          int totalHits = 0;
48          int index = 0;
49          int button = 0;
50          int scores = 0;
51          while (1){
52              switch (state) {
53                  case 0:
54                      init(disp0, disp1, disp2, disp3);
55                      //Generate random sequence
56                      for(i = 0;i < 8;i++){
57                          key[i] = a[*lr];
58                      }
59                      while(*b != 9);
60                      assign(TODISP0, disp3);
61                      assign(TODISP0, disp2);
62                      state = 1;
63                      break;
64                  case 1:
65                      if (level == 5){
66                          level = 1;
67                      }
68                      assign(TODISPNILL, disp1);
69                      assign(numDisplay[level], disp0);
70                      for(i=0;i<(level + 3);i++){
71                          assign(0,l);
72                          for(j=0;j<3000000;j++);
73                          assign(key[i],l);
74                          for(j=0;j<3000000;j++);
75                      }
76                      assign(0,l);
77                      state = 2;
78                      break;
79                  case 2:
80                      hits = 0;
81                      while(hits < (level + 3)){
82                          while(button == 0){
83                              button = *b;
84                          }
85                          //assign(hits,l);
86                          for(j=0;j<1000000;j++);
87                          if(button != 1 && button != 2 && button != 4 && button != 8) {
88                              continue;
89                          }
90                          if(button != key[hits]) {
91                              assign(TODISPNILL,disp0);
92                              assign(TODISPNILL,disp3);
```

```
93                              assign(TODISP0,disp1);
94                              assign(TODISP6,disp2);
95                              for(j=0;j<5000000;j++);
96                              level = 1;
97                              scores = 0;
98                              button = 0;
99                              state = 3;
100                             break;
101                         }
102                         hits++;
103                         totalHits++;
104                         button = 0;
105                         if(totalHits < 10){
106                             assign(TODISP0, disp3);
107                             assign(numDisplay[totalHits], disp2);
108                         }
109                         else if(totalHits >= 10 && totalHits < 20){
110                             assign(TODISP1, disp3);
111                             index = totalHits - 10;
112                             assign(numDisplay[index], disp2);
113                         }
114                         else if(totalHits >= 20 && totalHits < 30){
115                             assign(TODISP2, disp3);
116                             index = totalHits - 20;
117                             assign(numDisplay[index], disp2);
118                         }
119                         else if(totalHits >= 30){
120                             assign(TODISP3, disp3);
121                             index = totalHits - 30;
122                             assign(numDisplay[index], disp2);
123                         }
124                         //assign(48,l);
125                         for(j=0;j<1000000;j++);
126                     }
127                     if(hits == (level + 3)){
128                         level++;
129                     }
130                     state = 1;
131                     break;
132                 default:
133                     break;
134             }
135         }
136         return 0;
137     }
138
```