Lab 2: A SIMPLE MICE-HUNTING GAME

1. Game Description

This is a simple mice-hunting game from SOPC DESIGN BASED ON FPGA AND TOUCH SCREEN.
Because of the limited hardware components on board DE2 (No Touch screen), we simplified this game and used an LCD on board for display.

When the game starts, a welcome message and the rules will appear on the screen.

After that, the game begins with three possible positions shown on the LCD screen: _____0, ___0__, and _0____ representing **right**, **middle**, and **left** positions of the letter 0.

The letter 0 will randomly appear in one of these three positions for a limited time.
Your goal is to **"hunt" the 0** by pressing the correct switch that matches its position:

- Press the **left switch** when 0 is on the **left**
- Press the **middle switch** when 0 is in the **middle**
- Press the **right switch** when 0 is on the **right**

If you press the correct switch in time:

- The screen will show **"Correct."**
- Your **score increases by 1**

If you press the wrong switch or do nothing:

- The 0 will disappear
- A new 0 will appear randomly at a different position

There are **three difficulty levels** to choose from before playing:

- **Junior** (more hunting time)
- **Senior** (medium hunting time)
- **Super** (short hunting time)

After the game ends, your performance is evaluated based on the percentage of successful hits (hunted 0s):

- **Poor**
- **Fair**
- **Good**
- **Perfect**

Finally, the game returns to the welcome screen for another round.

2. Hardware Components.

We built a Nios II processor using SDRAM (… Kbyte) instead of Memory on chip to provide larger memory for storing a C program, but at a lower speed. We add an **SDRAM Controller that** provides an interface between an FPGA system and external SDRAM memory. It manages the complex timing and control signals required for reading from and writing to SDRAM chips, allowing components like the Nios II processor to access large amounts of external memory efficiently. This controller enables the system to store and retrieve code, data, and other runtime information beyond the limited on-chip memory, making it essential for applications that require more memory resources

We use **Interval Timer** to generate precise timing events in an FPGA-based system. It functions as a countdown timer that can trigger periodic interrupts when it reaches zero. This makes it useful for creating time delays, measuring elapsed time, or generating regular system ticks for real-time operations. The timer can be configured to automatically reload and continue generating interrupts at fixed intervals, allowing the processor (such as Nios II) to perform time-based tasks like blinking LEDs, polling sensors, or maintaining software timers.

We using **PIO** for LEDs, switches, and 7-segment LEDs, and **Altera Avalon LCD 16207** to control the LCD (16 ✕2) on board.

a. PIO (Parallel Input/Output) is an interface component that allows the FPGA to communicate directly with peripheral devices through parallel signal lines (port lines). PIO provides a simple data channel between the internal processor or FPGA logic and external devices such as buttons, LEDs, sensors, or displays. Through PIO, data can be efficiently read or written via internal registers, while

the physical port lines connect to FPGA pins to control or receive signals from peripherals. PIO supports flexible configurations such as data bus width, input, output, or bidirectional modes, and can enable interrupts to quickly respond to events from peripheral devices. Thus, PIO plays a crucial role as a bridge, enabling FPGA designs to easily integrate and control peripherals in embedded systems.

b. The Altera Avalon LCD 16207 is an IP core designed to interface an FPGA system, typically featuring a Nios II processor, with a 16x2 character LCD display module. This IP core acts as a controller that manages communication between the FPGA's Avalon bus and the LCD's parallel interface signals, including data lines and control signals (such as RS, RW, and Enable). It abstracts the low-level timing and control details required by the LCD module, allowing the processor or FPGA logic to easily write characters, send commands, and control cursor movement through simple read/write operations on the Avalon bus. By using this IP, designers can efficiently integrate a standard 16x2 LCD display into embedded FPGA systems, facilitating clear visual output without the need to manually handle the LCD protocol.

We also use a **System ID Peripheral** to ensure compatibility between the hardware system and the software running on it. It stores a unique identifier and timestamp generated during hardware compilation. When downloading a software application (such as a Nios II program), the System ID is checked to verify that the software matches the correct hardware version. This prevents errors caused by mismatched versions and helps improve system reliability and debugging efficiency.

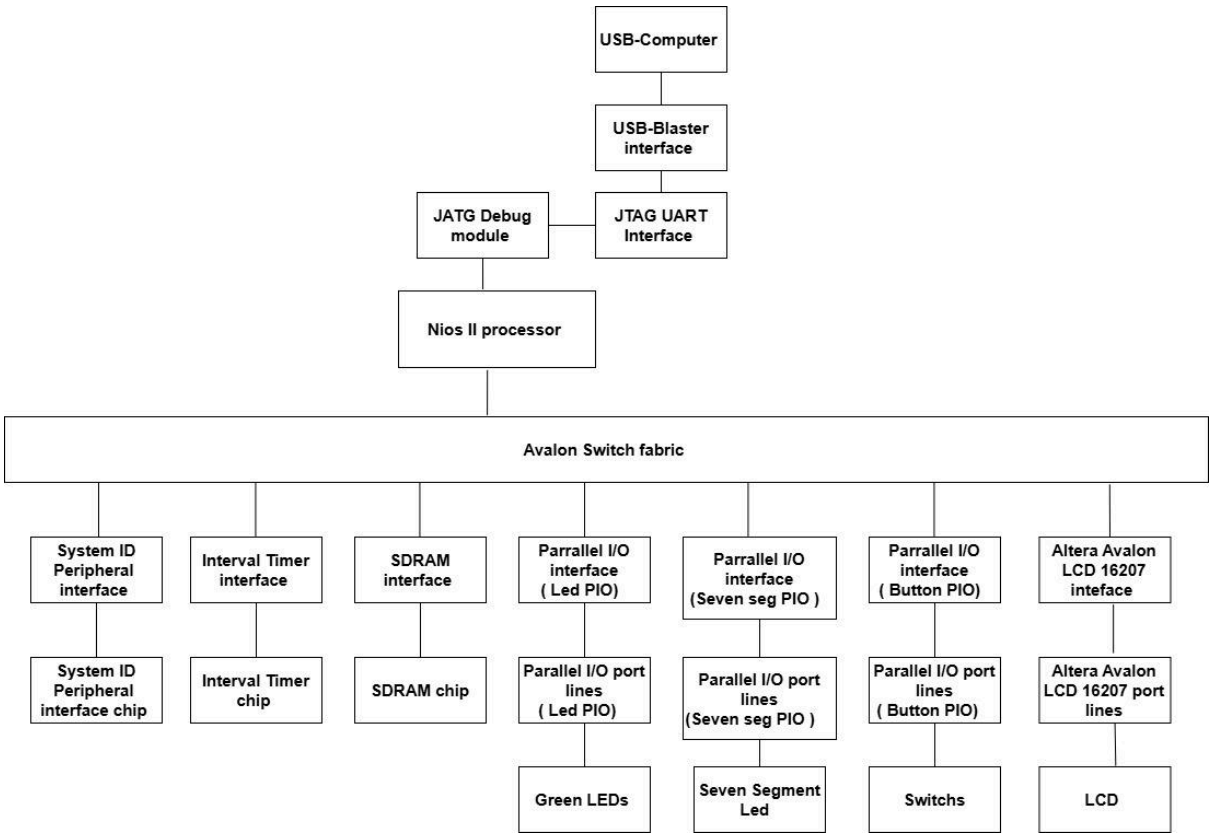
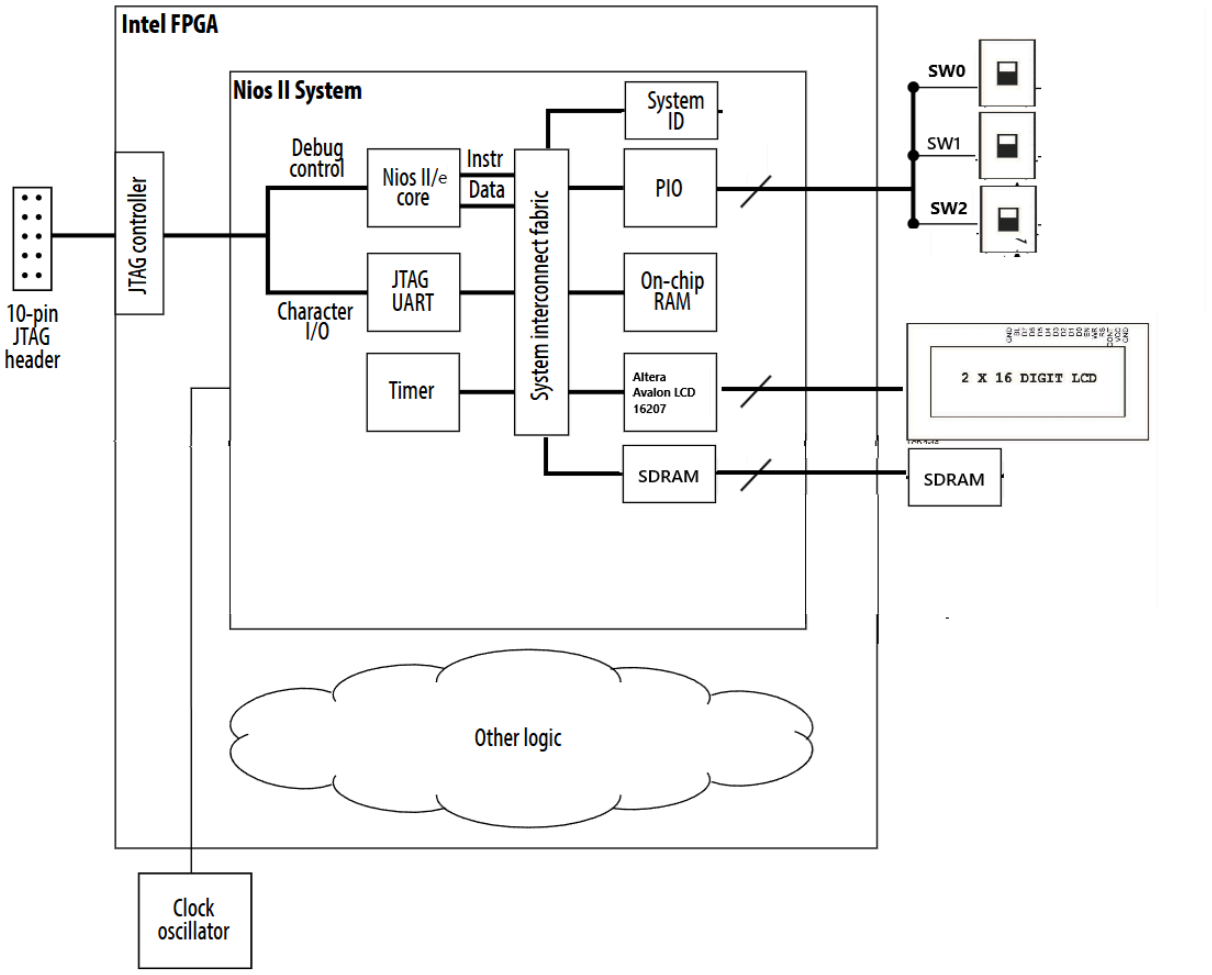
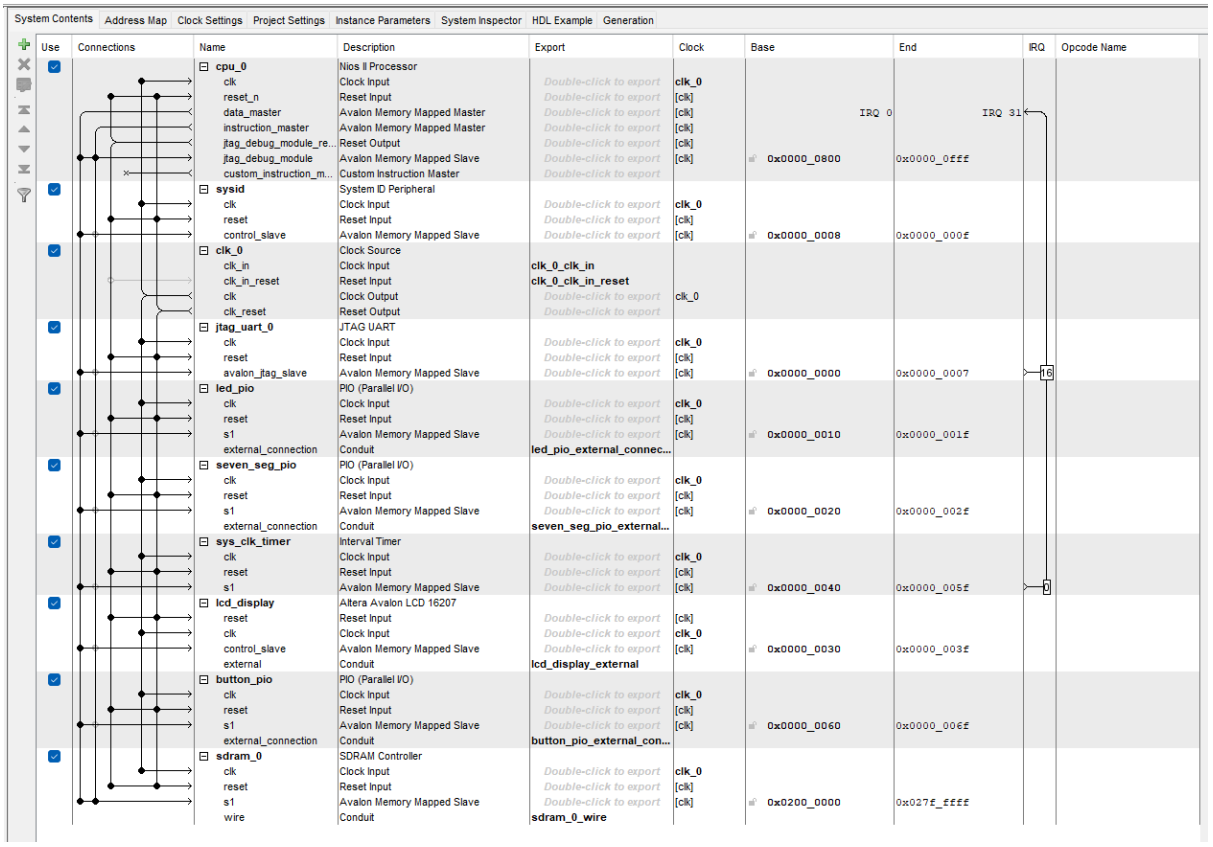
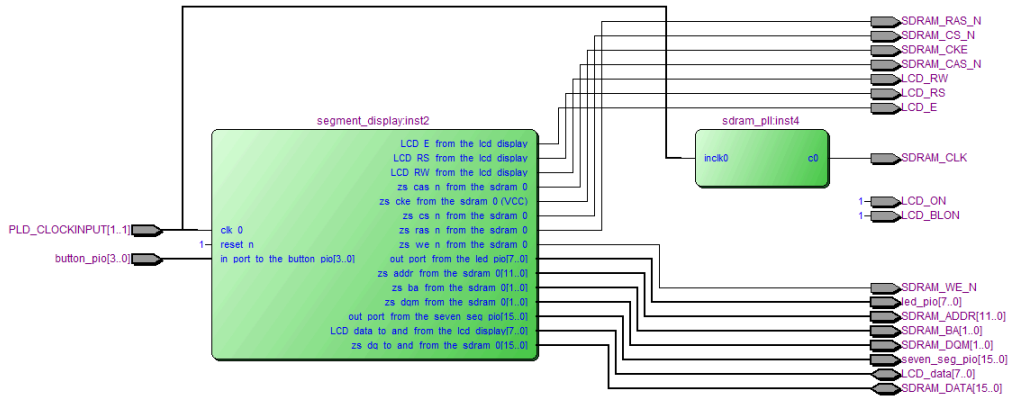


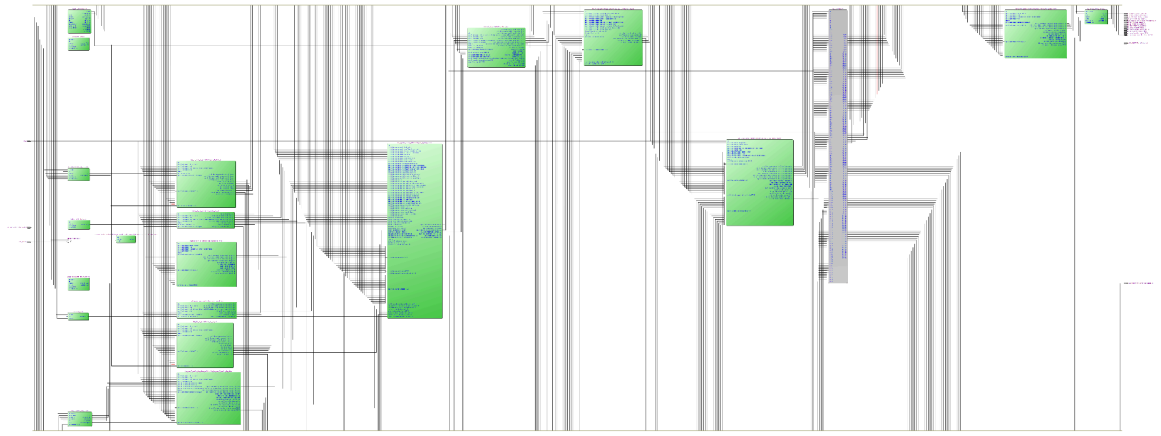
Figure 1. A Nios II System implemented on a DE-series board.

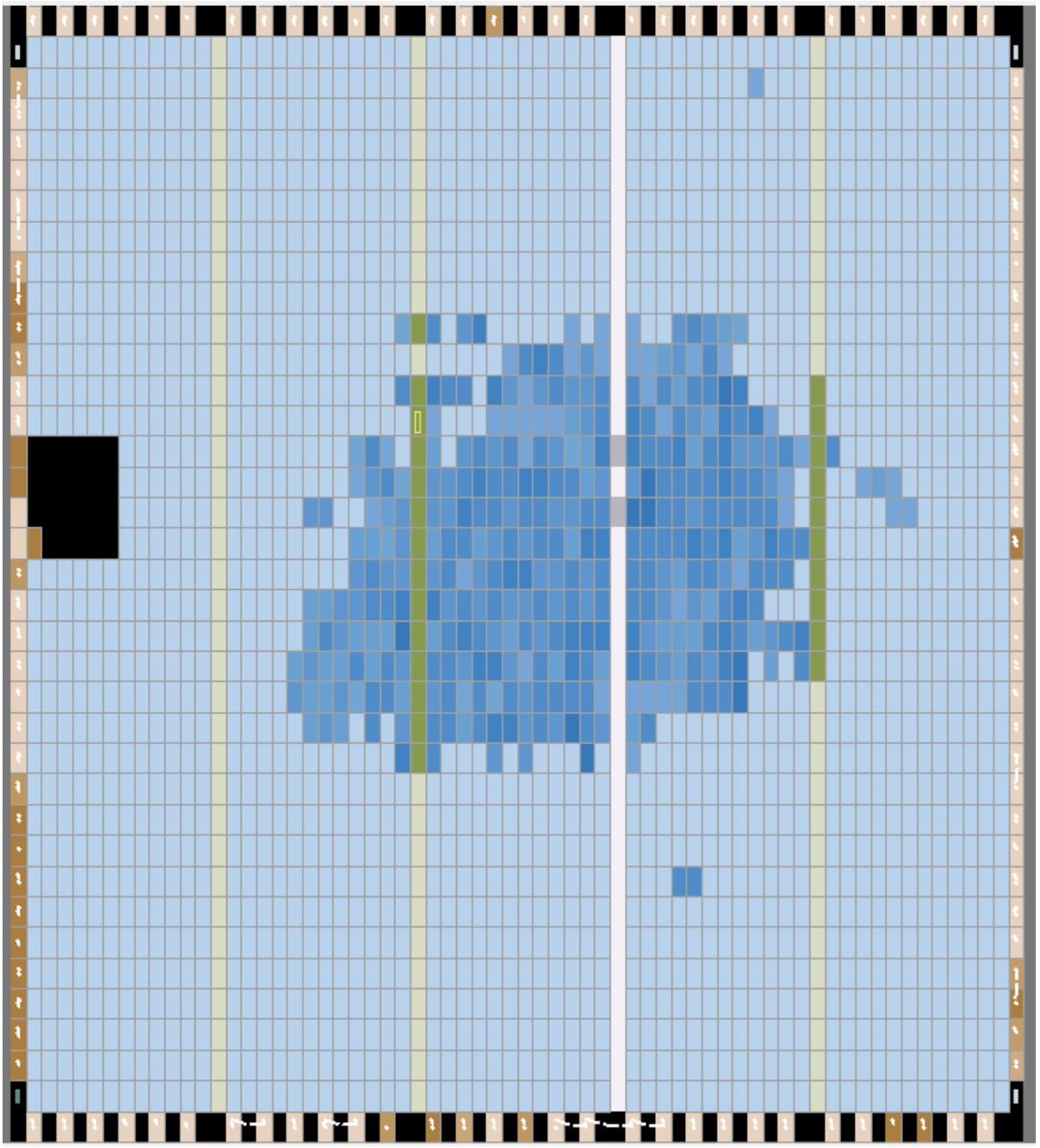
Target Board




System Contents
Address Map
Clock Settings
Project Settings
Instance Parameters
System Inspector
HDL Example
Generation
Use
Connections
Name
Description
Export
Clock
Base
End
IRQ
Opcode Name
cpu_0
Nios II Processor
clk
Clock Input
clk_0
reset_n
Reset Input
data_master
Avalon Memory Mapped Master
IRQ 0
IRQ 31
instruction_master
jtag_debug_module_re...
Reset Output
jtag_debug_module
Avalon Memory Mapped Slave
0x0000_0800
0x0000_0fff
custom_instruction_m...
Custom Instruction Master
sysid
System ID Peripheral
control_slave
0x0000_0008
0x0000_000f
clk_0
Clock Source
clk_in
clk_0_clk_in
clk_in_reset
clk_0_clk_in_reset
clk_reset
jtag_uart_0
JTAG UART
avalon_jtag_slave
0x0000_0000
0x0000_0007
led_pio
PIO (Parallel I/O)
0x0000_0010
0x0000_001f
external_connection
Conduit
led_pio_external_connec...
seven_seg_pio
0x0000_0020
0x0000_002f
seven_seg_pio_external...
sys_clk_timer
Interval Timer
0x0000_0040
0x0000_005f
lcd_display
Altera Avalon LCD 16207
reset
0x0000_0030
0x0000_003f
lcd_display_external
button_pio
0x0000_0060
0x0000_006f
button_pio_external_con...
sdram_0
SDRAM Controller
0x0200_0000
0x027f_ffff
wire
sdram_0_wire


segment_display:inst2
sdram_pll:inst4
PLD_CLOCKINPUT[1..1]
button_pio[3..0]
clk 0
reset n
in port to the button pio[3..0]
LCD E from the lcd display
LCD RS from the lcd display
LCD RW from the lcd display
zs cas n from the sdram 0
zs cke from the sdram 0 (VCC)
zs cs n from the sdram 0
zs ras n from the sdram 0
zs we n from the sdram 0
out port from the led pio[7..0]
zs addr from the sdram 0[11..0]
zs ba from the sdram 0[1..0]
zs dqm from the sdram 0[1..0]
out port from the seven seg pio[15..0]
LCD data to and from the lcd display[7..0]
zs dq to and from the sdram 0[15..0]
inclk0
c0
SDRAM_RAS_N
SDRAM_CS_N
SDRAM_CKE
SDRAM_CAS_N
LCD_RW
LCD_RS
LCD_E
SDRAM_CLK
LCD_ON
LCD_BLON
SDRAM_WE_N
led_pio[7..0]
SDRAM_ADDR[11..0]
SDRAM_BA[1..0]
SDRAM_DQM[1..0]
seven_seg_pio[15..0]
LCD_data[7..0]
SDRAM_DATA[15..0]





3. Program Flow

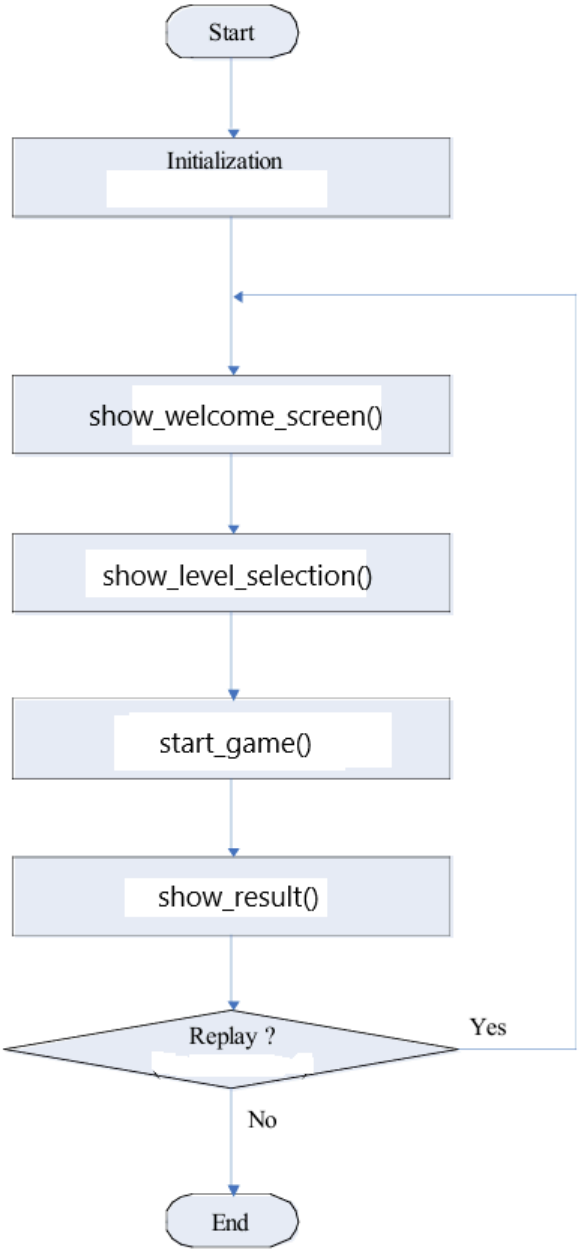


Figure: General flow for the mice hunting game.

4. C program for the game.

a. Display the welcome message on the LCD

We use the function LCD_PRINTF from the library to display two messages: Welcome to Mouse Hunt! and Catch mice with switches! on the LCD to welcome the user to our game.

```
void show_welcome_screen(FILE *lcd)
{
   LCD_PRINTF(lcd, "%c%s %c%s %c%s Welcome to Mouse Hunt!\n",
 ESC, ESC_TOP_LEFT, ESC, ESC_CLEAR, ESC, ESC_COL1_INDENT5);

 LCD_PRINTF(lcd, "%c%s Catch mice with switches!\n", ESC,
ESC_COL2_INDENT5);
}
```

The LCD_PRINTF function in the code is used to display text and control commands on an LCD screen. Although LCD_PRINTF is not a standard C function but rather a custom macro or function, it operates similarly to fprintf, enabling data output to the LCD with specific formatting.

The function takes the following parameters:

(1) First parameter: FILE *lcd - This is a pointer to the LCD device or file, specifying the output destination (similar to stdout in printf)

(2) Second parameter: format string - This string, such as "%c%s %c%s %c%s Welcome to Mouse Hunt!\n", defines how data is displayed, including control characters (%c), strings (%s), and fixed text; characters like \n may move the LCD cursor to the next line depending on the device.

(3) Subsequent parameters: variable arguments (varargs) - These are values corresponding to the format specifiers, such as ESC, ESC_TOP_LEFT, ESC_CLEAR, ESC_COL1_INDENT5, or text strings, used to send control commands (e.g., clear screen, move cursor) or text data to the LCD.

The LCD_PRINTF function combines these parameters to format and display content accurately on the LCD screen, ensuring commands and text appear in the correct position and format.

Result of the function on the LCD is:

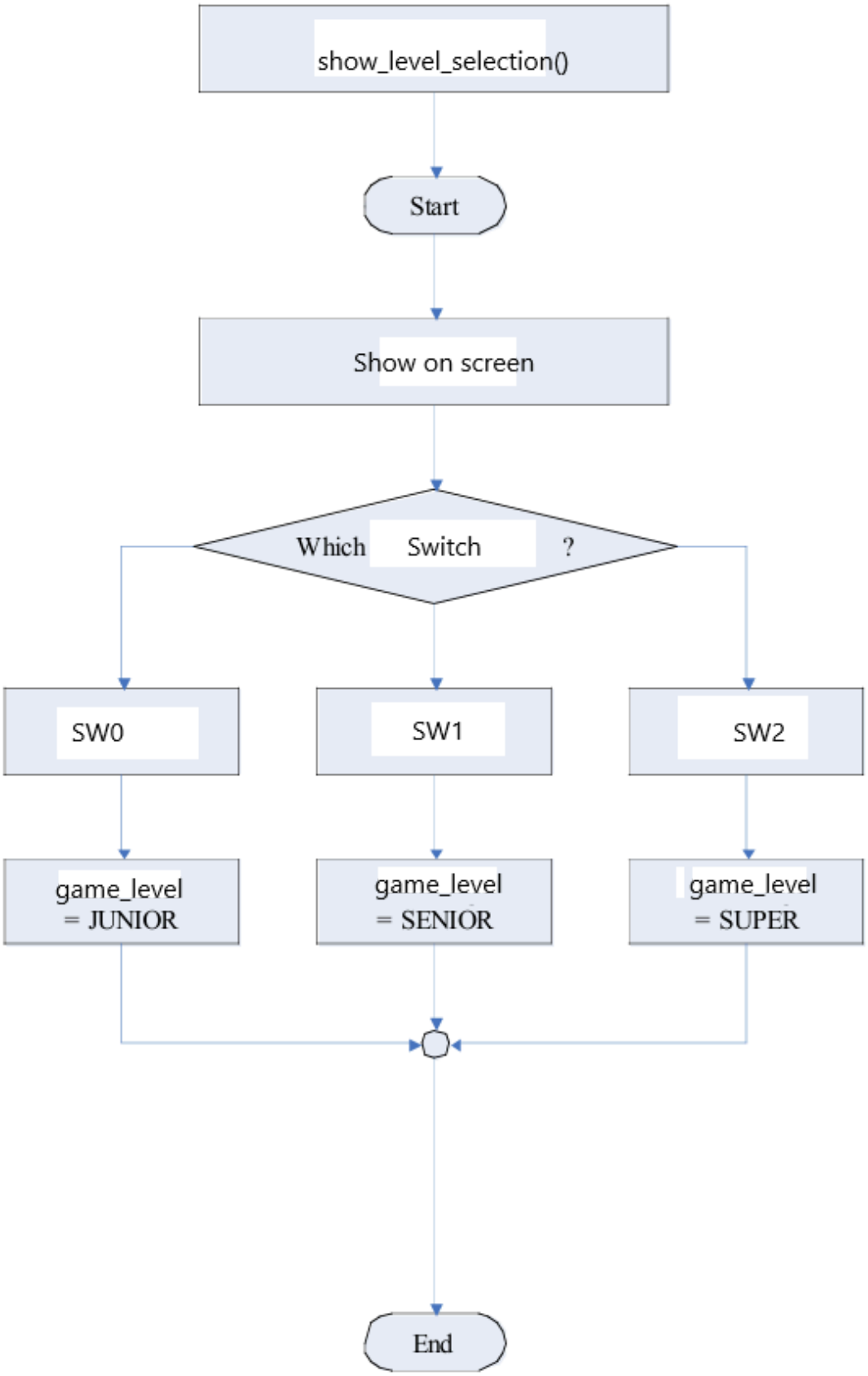
```
Welcome to Mouse Hunt!
    Catch mice with switches!
```

b. Level Selection

void show_level_selection(FILE *lcd)

```
{
   LCD_PRINTF(lcd, "%c%s %c%s %c%s Select level:\n", ESC,
ESC_TOP_LEFT,
            ESC, ESC_CLEAR, ESC, ESC_COL1_INDENT5);
   LCD_PRINTF(lcd, "%c%s SW0-Junior SW1-Senior \n", ESC,
ESC_COL2_INDENT5);
   LCD_PRINTF(lcd, "%c%s SW2-Super \n", ESC, ESC_COL2_INDENT5);

   // Wait for level selection
   while(1)
   {
      alt_u8 buttons =
IORD_ALTERA_AVALON_PIO_DATA(BUTTON_PIO_BASE) & 0x7;
      if(buttons & 0x1) {
        game_level = JUNIOR;
        printf("Level: %d \n", game_level);
        break;
      }
      if(buttons & 0x2) {
        game_level = SENIOR;
        printf("Level: %d \n", game_level);
        break; }
      if(buttons & 0x4) {
        game_level = SUPER;
        printf("Level: %d \n", game_level);
        break; }

      usleep(10000);
   }
}
```

The show_level_selection() module deals with the choice of the level. As the main part of the module, the function matches the number of mice for different levels and passes them down to the next steps. The flow chart of the Choose Level module is shown in Figure.



The button “JUNIOR” is located SW0, “SENIOR” is SW1, and “SUPER” is SW2. If the user switches, then the total number of mice – the variable “game_level” will be

set as “JUNIOR”, “SENIOR”, “SUPER” and the program will enter the start_game() module.

c. Delay in milliseconds function

Since I need not to delay according to the International System units, a set of nested “for” statements can reach the purpose of the function

```
void delay_ms(int time)
{

  int i, j, k;
  for(i=0; i<=time; i++)
     for(j=0; j<=2000; j++) k = 1;
}
```

d. Create a random position

We generate a position of mice randomly on the LCD by using the rand() function, and we denote that 0, 1, and 2 for respectively left, middle, and right positions.

```
int get_random_position()
{
   return rand() % 3; // Returns 0, 1, or 2
}
```

e. Display target in the LCD

```
void display_target(FILE *lcd, int position)
{
   const char *patterns[3] = {"____0", "__0__", "_0___"};
   LCD_PRINTF(lcd, "%c%s %c%s %c%s %s\n", ESC, ESC_TOP_LEFT,
         ESC, ESC_CLEAR, ESC, ESC_COL1_INDENT5,
patterns[position]);
}
```

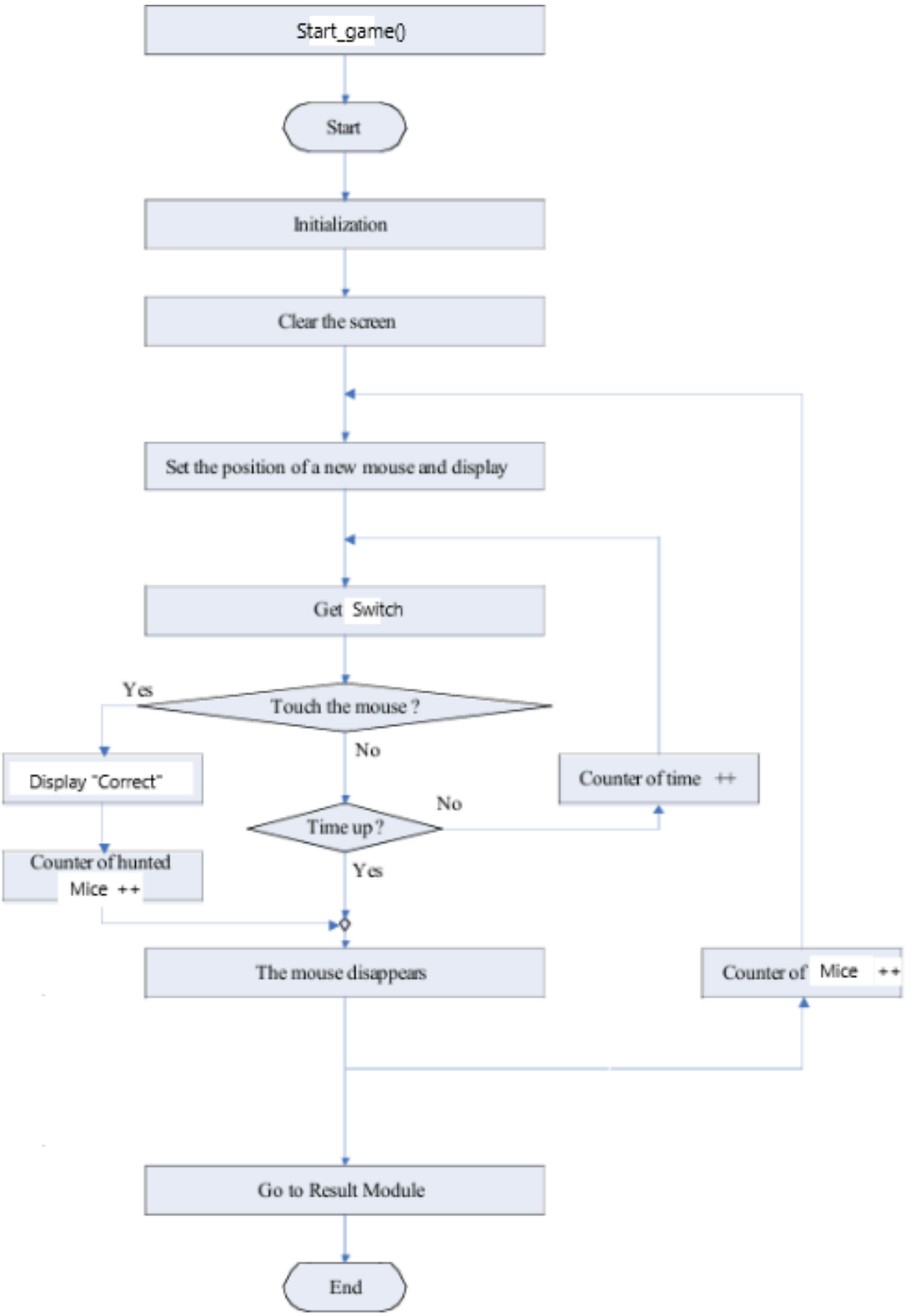
f. Start the game

```
void start_game(FILE *lcd)
{
  HuntedMice = 0;
  mice = 0;
  int game_time = 0; // 30 seconds for the game
  if (game_level == JUNIOR){
```

```
        game_time = 30;
    }
    else if (game_time == SENIOR){
        game_time = 20;
    }
    else {
        game_time = 10;
    }


    LCD_PRINTF(lcd, "%c%s %c%s %c%s Game started!\n", ESC, ESC_TOP_LEFT,
            ESC, ESC_CLEAR, ESC, ESC_COL1_INDENT5);

    time_t start_time = time(NULL);


    while(difftime(time(NULL), start_time) < game_time)
    {
        current_position = get_random_position();
        display_target(lcd, current_position);

        // Wait for player input or timeout (1 second)
        time_t target_start = time(NULL);
        int caught = 0;

        while(difftime(time(NULL), target_start) < 1.0)
        {
            alt_u8 buttons =
IORD_ALTERA_AVALON_PIO_DATA(BUTTON_PIO_BASE) & 0x7;

            if((buttons & 0x1) && current_position == 0) caught = 1; // SW0 for right
            if((buttons & 0x2) && current_position == 1) caught = 1; // SW1 for middle
            if((buttons & 0x4) && current_position == 2) caught = 1; // SW2 for left

            if(caught)
            {
                LCD_PRINTF(lcd, "%c%s %c%s %c%s Correct!\n", ESC,
ESC_TOP_LEFT,
                        ESC, ESC_CLEAR, ESC, ESC_COL1_INDENT5);
                usleep(500000);
```

```
            HuntedMice++;
            mice++;
            printf("\nHuntedMice: %d", HuntedMice);
            usleep(500000); // Show "Correct" for 0.5s
            break;
        }

        usleep(10000);
    }

    if(!caught)
    {
        mice++;
    }
  }
}
```

The start_game() module is the soul of the whole program. The main part of the module is the function start_game(). The function start_game()receives the total number of mice that will appear during one game, and that will determine the speed of the mice changing their positions. Once hunted (switch), the screen will display “Correct”. No matter whether a mouse has been hunted, it will disappear after a certain time. When all the mice of the game have appeared, the number of the hunted mice will be returned. The flow chart of the start_game() module

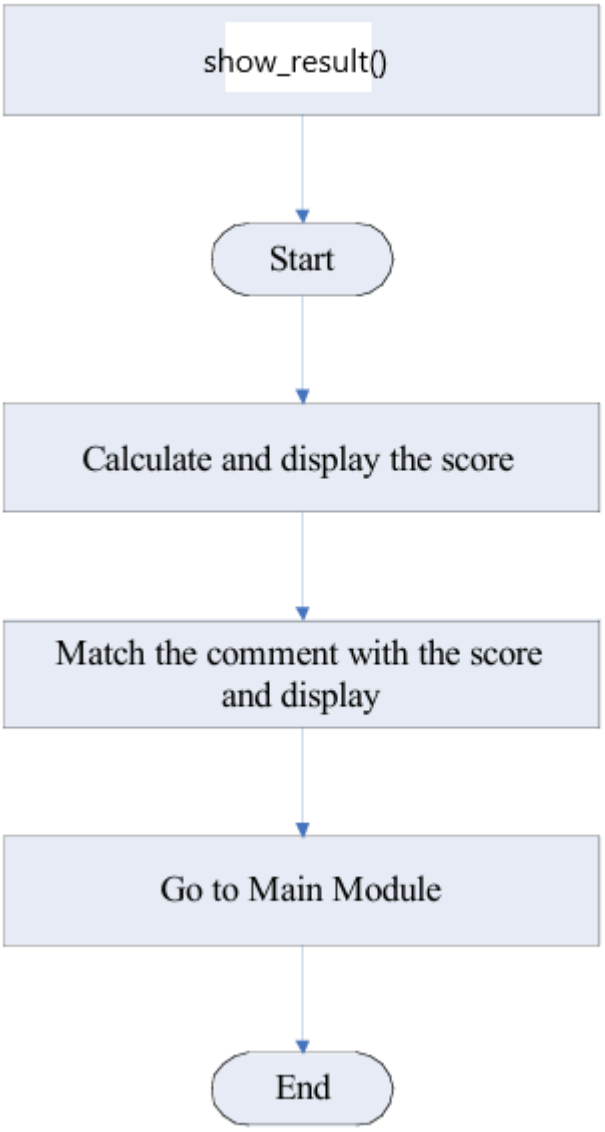

Start_game()
Start
Initialization
Clear the screen
Set the position of a new mouse and display
Get Switch
Yes
Touch the mouse ?
No
Display "Correct"
Counter of time ++
No
Time up ?
Counter of hunted Mice ++
Yes
The mouse disappears
Counter of Mice ++
Go to Result Module
End

In the very beginning of the module sets the time “game_time” that time for a game
The mice of the current game then come out in random places and disappear one by one. In the program, I used 0____, ___0____, _____0 to stand for the mice. The place of each mouse is set by the function rand()
Then I used a set of nested “while” statements as the timer to count the “game_time” , in order to wait for the hunting movement in the same time. In different levels, there are different “game_time”. Once hunted, the screen will display “Correct”, remaining for a certain while, which is measured by the delay function, and then disappear. If not hunted during the lifetime, the mouse will disappear, too. A new mouse will come out after the former one disappears. Meanwhile, a counter counts the hunted mice. In the end, when all mice of the game have appeared, the number of the hunted mice will be returned, and the program will enter the Result module.

g. Show the result on the LCD
We calculate the ratio of the number of hunted mice to the total mice that appeared on the LCD and compare it with the score range to display a comment (POOR/FAIR/ GOOD) on the LCD. We also display on result on the console window to manage.

```
void show_result(FILE *lcd)
{
   float percentage = (mice > 0) ? (float)HuntedMice / mice * 100 : 0;
   int rating;

   if(percentage < 25) rating = POOR;
   else if(percentage < 50) rating = FAIR;
   else if(percentage < 75) rating = GOOD;
   else rating = PERFECT;

   const char *ratings[] = {"Poor", "Fair", "Good", "Perfect"};

   LCD_PRINTF(lcd, "%c%s %c%s %c%s Game Over!\n", ESC,
ESC_TOP_LEFT,
         ESC, ESC_CLEAR, ESC, ESC_COL1_INDENT5);
   LCD_PRINTF(lcd, "%c%s HuntedMice: %d/%d\n", ESC,
ESC_COL2_INDENT5, HuntedMice, mice);
   LCD_PRINTF(lcd, "%c%s Rating: %s\n", ESC, ESC_COL2_INDENT5,
ratings[rating]);
```

```
    printf("HuntedMice: %d\n", HuntedMice);
    printf("Mice: %d\n", mice);
    printf("Rating: %s\n", ratings[rating]);
}
```



Score < 25%	Poor
25% ≤ Score < 50%.	Fair
50% ≤ Score < 75%!	Good
Score ≥ 75%	Perfect

h. Main program

```
int main(void)
```

```
{
   FILE *lcd = LCD_OPEN();
   if(lcd == NULL) return 1;

   while(1)
   {
      show_welcome_screen(lcd);
      printf("\nShow welcome screen\n");
      usleep(2000000); // Wait 2 seconds


      show_level_selection(lcd);
      printf("\nShow level selection\n");
      usleep(2000000); // Wait for level selection

      printf("\nStart_game\n");
      start_game(lcd);
      usleep(1000000); // Game duration

      show_result(lcd);
      printf("\nShow result\n");
      usleep(5000000); // Show result for 5 seconds


   }

   LCD_CLOSE(lcd);
   return 0;
}
```

The main function of the program follows the overall flow of the game. First, the LCD is initialized, and the show_welcome_screen function is called to display a welcome message, followed by a 2-second delay. Next, the show_level_selection function is called to prompt the user to select a game level. The program waits until the user makes a selection by pressing a switch. After a level is chosen, the program pauses for another 2 seconds before starting the game by calling start_game(lcd). Once the game duration ends, the show_result(lcd) function is called to display the result on the LCD. Finally, after 5 seconds, the game automatically returns to the first step.

i. Full version code

- Source file: file game.c

```
#include "game.h"
#include <time.h>

// Global variables
static int game_level = JUNIOR;
static int HuntedMice = 0;
static int mice = 0;
static int current_position = 0;

// Function prototypes
void show_welcome_screen(FILE *lcd);
void show_level_selection(FILE *lcd);
void start_game(FILE *lcd);
void show_result(FILE *lcd);
void display_target(FILE *lcd, int position);
int get_random_position();
void delay_ms(int milliseconds);

int main(void)
{
   FILE *lcd = LCD_OPEN();
   if(lcd == NULL) return 1;

   while(1)
   {
      show_welcome_screen(lcd);
      printf("\nshow welcome screen\n");
      usleep(2000000); // Wait 2 seconds


      show_level_selection(lcd);
      printf("\nshow level selection\n");
      usleep(2000000); // Wait for level selection

      printf("\nstart_game\n");
      start_game(lcd);
      usleep(1000000); // Game duration
```

```
        show_result(lcd);
        printf("\nshow result\n");
        usleep(5000000); // Show result for 5 seconds


    }

    LCD_CLOSE(lcd);
    return 0;
}

void show_welcome_screen(FILE *lcd)
{
    LCD_PRINTF(lcd, "%c%s %c%s %c%s Welcome to Mouse Hunt!\n", ESC,
ESC_TOP_LEFT,
            ESC, ESC_CLEAR, ESC, ESC_COL1_INDENT5);
    LCD_PRINTF(lcd, "%c%s Catch mice with switches!\n", ESC,
ESC_COL2_INDENT5);
}

void show_level_selection(FILE *lcd)
{
    LCD_PRINTF(lcd, "%c%s %c%s %c%s Select level:\n", ESC,
ESC_TOP_LEFT,
            ESC, ESC_CLEAR, ESC, ESC_COL1_INDENT5);
    LCD_PRINTF(lcd, "%c%s SW0-Junior SW1-Senior \n", ESC,
ESC_COL2_INDENT5);
    LCD_PRINTF(lcd, "%c%s SW2-Super \n", ESC, ESC_COL2_INDENT5);

    // Wait for level selection
    while(1)
    {
        alt_u8 buttons =
IORD_ALTERA_AVALON_PIO_DATA(BUTTON_PIO_BASE) & 0x7;
        if(buttons & 0x1) {
            game_level = JUNIOR;
            printf("Level: %d \n", game_level);
            break;
        }
```

```
        if(buttons & 0x2) {
          game_level = SENIOR;
          printf("Level: %d \n", game_level);
          break; }
        if(buttons & 0x4) {
          game_level = SUPER;
          printf("Level: %d \n", game_level);
          break; }

        usleep(10000);
    }
}

void start_game(FILE *lcd)
{
    HuntedMice = 0;
    mice = 0;
    int game_time = 0; // 30 seconds for the game
    if (game_level == JUNIOR){
          game_time = 30;
    }
    else if (game_time == SENIOR){
          game_time = 20;
    }
    else {
          game_time = 10;
    }


    LCD_PRINTF(lcd, "%c%s %c%s %c%s Game started!\n", ESC,
ESC_TOP_LEFT,
            ESC, ESC_CLEAR, ESC, ESC_COL1_INDENT5);

    time_t start_time = time(NULL);


    while(difftime(time(NULL), start_time) < game_time)
    {
        current_position = get_random_position();
        display_target(lcd, current_position);
```

```
        // Wait for player input or timeout (1 second)
        time_t target_start = time(NULL);
        int caught = 0;

        while(difftime(time(NULL), target_start) < 1.0)
        {
            alt_u8 buttons =
IORD_ALTERA_AVALON_PIO_DATA(BUTTON_PIO_BASE) & 0x7;

            if((buttons & 0x1) && current_position == 0) caught = 1; // SW0 for
right
            if((buttons & 0x2) && current_position == 1) caught = 1; // SW1 for
middle
            if((buttons & 0x4) && current_position == 2) caught = 1; // SW2 for
left

            if(caught)
            {
                LCD_PRINTF(lcd, "%c%s %c%s %c%s Correct!\n", ESC,
ESC_TOP_LEFT,
                        ESC, ESC_CLEAR, ESC, ESC_COL1_INDENT5);
                usleep(500000);
                HuntedMice++;
                mice++;
                printf("\nHuntedMice: %d", HuntedMice);
                usleep(500000); // Show "Correct" for 0.5s
                break;
            }

            usleep(10000);
        }

        if(!caught)
        {
            mice++;
        }
    }
}
```

```
void display_target(FILE *lcd, int position)
{
   const char *patterns[3] = {"____0", "__0__", "_0___"};
   LCD_PRINTF(lcd, "%c%s %c%s %c%s %s\n", ESC, ESC_TOP_LEFT,
         ESC, ESC_CLEAR, ESC, ESC_COL1_INDENT5,
patterns[position]);
}

int get_random_position()
{
   return rand() % 3; // Returns 0, 1, or 2
}

void show_result(FILE *lcd)
{
   float percentage = (mice > 0) ? (float)HuntedMice / mice * 100 : 0;
   int rating;

   if(percentage < 25) rating = POOR;
   else if(percentage < 50) rating = FAIR;
   else if(percentage < 75) rating = GOOD;
   else rating = PERFECT;

   const char *ratings[] = {"Poor", "Fair", "Good", "Perfect"};

   LCD_PRINTF(lcd, "%c%s %c%s %c%s Game Over!\n", ESC,
ESC_TOP_LEFT,
         ESC, ESC_CLEAR, ESC, ESC_COL1_INDENT5);
   LCD_PRINTF(lcd, "%c%s HuntedMice: %d/%d\n", ESC,
ESC_COL2_INDENT5, HuntedMice, mice);
   LCD_PRINTF(lcd, "%c%s Rating: %s\n", ESC, ESC_COL2_INDENT5,
ratings[rating]);
   printf("HuntedMice: %d\n", HuntedMice);
   printf("Mice: %d\n", mice);
   printf("Rating: %s\n", ratings[rating]);
}

void delay_ms(int time)
{
```

```
    int i, j, k;
    for(i=0; i<=time; i++)
        for(j=0; j<=2000; j++) k = 1;
}
```

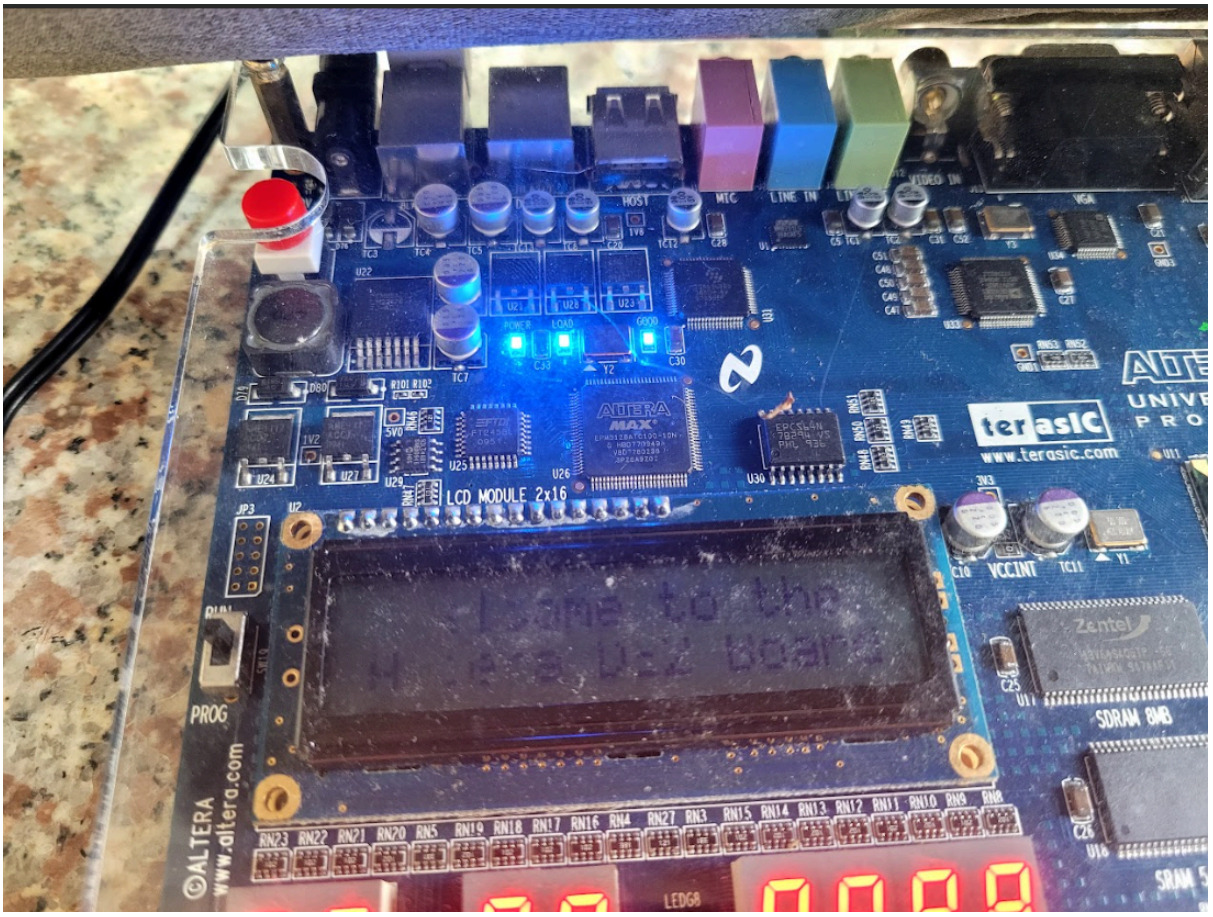
- Header file: file game.h

```
#ifndef GAME_H
#define GAME_H

#include "alt_types.h"
#include "altera_avalon_pio_regs.h"
#include "sys/alt_irq.h"
#include "system.h"
#include <stdio.h>
#include <unistd.h>

// LCD Definitions
#ifndef LCD_DISPLAY_NAME
#   define LCD_CLOSE(x)
#   define LCD_OPEN() NULL
#   define LCD_PRINTF(lcd, ...)
#else
#   define LCD_CLOSE(x) fclose((x))
#   define LCD_OPEN() fopen("/dev/lcd_display", "w")
#   define LCD_PRINTF fprintf
#endif

#define ESC 27
#define ESC_CLEAR "K"
#define ESC_TOP_LEFT "[1;0H"
#define ESC_COL1_INDENT5 "[1;5H"
#define ESC_COL2_INDENT5 "[2;5H"

// Game constants
#define JUNIOR 0
#define SENIOR 1
#define SUPER 2
```

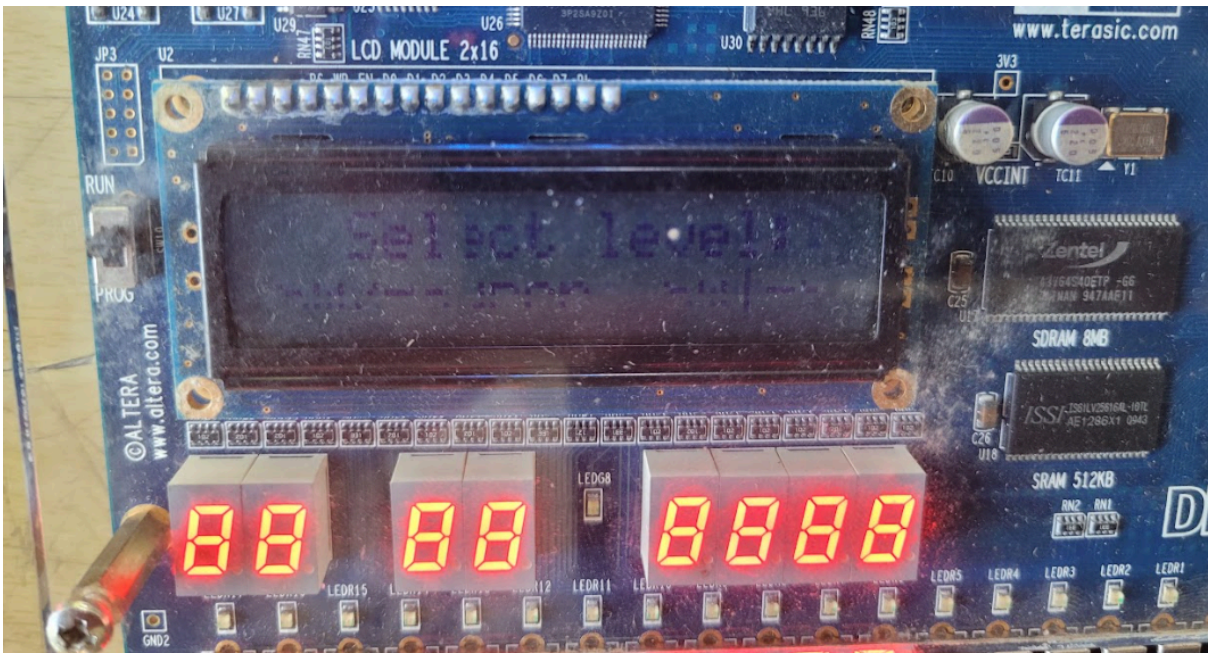
```
#define POOR 0
#define FAIR 1
#define GOOD 2
#define PERFECT 3

#endif
```

5. Result: Game hunt mice on FPGA SoC Hunt mice on FPGA

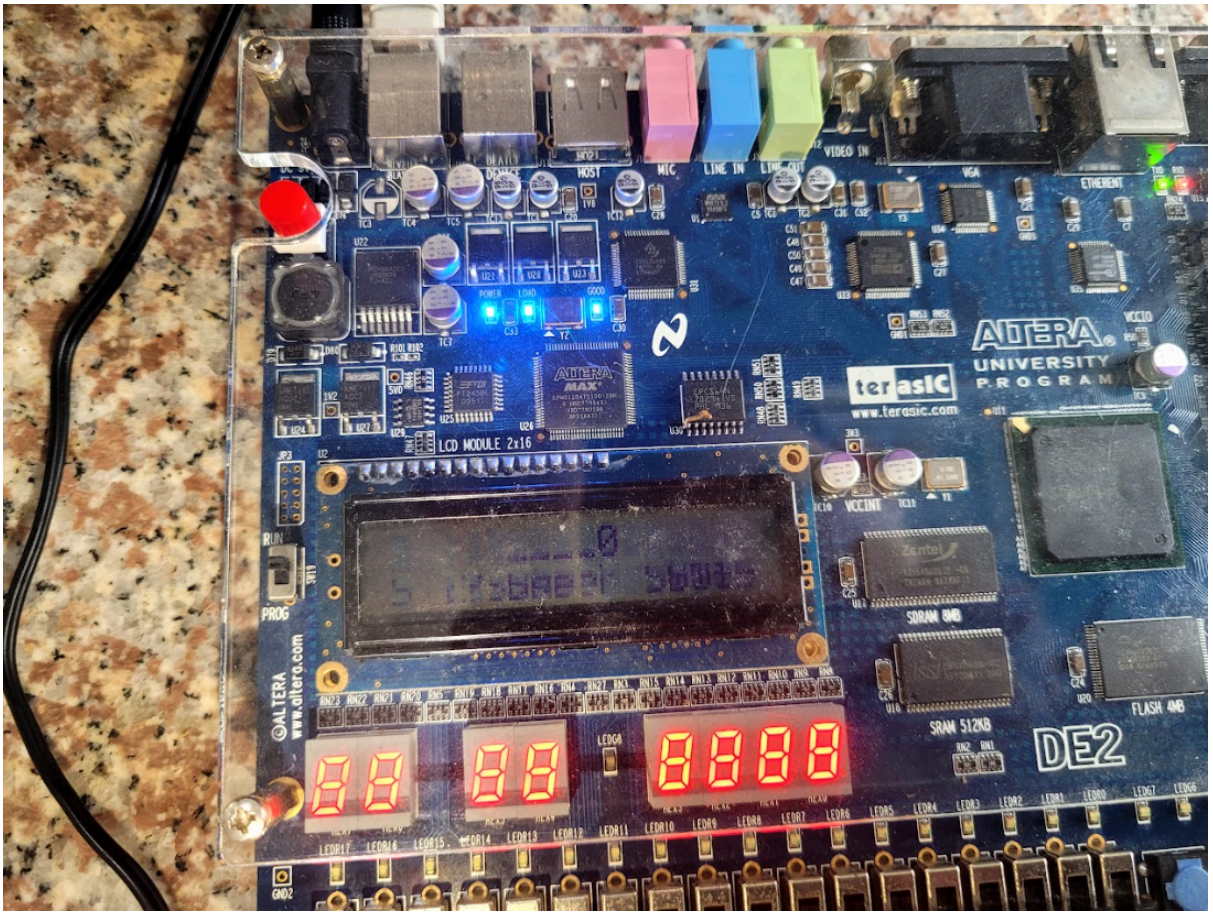
Unfortunately, my kit DE2 didn't perform well due to a poor LCD screen

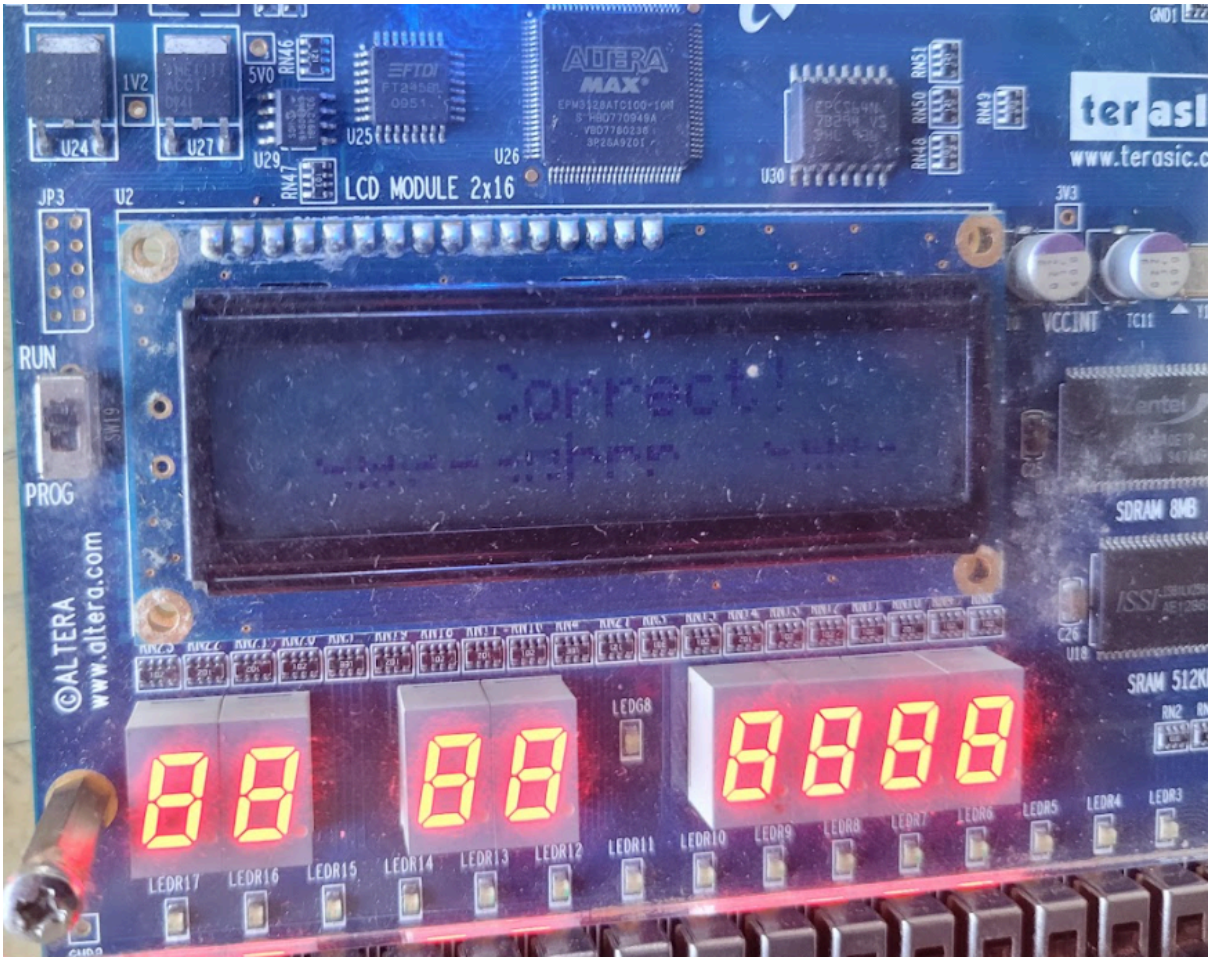


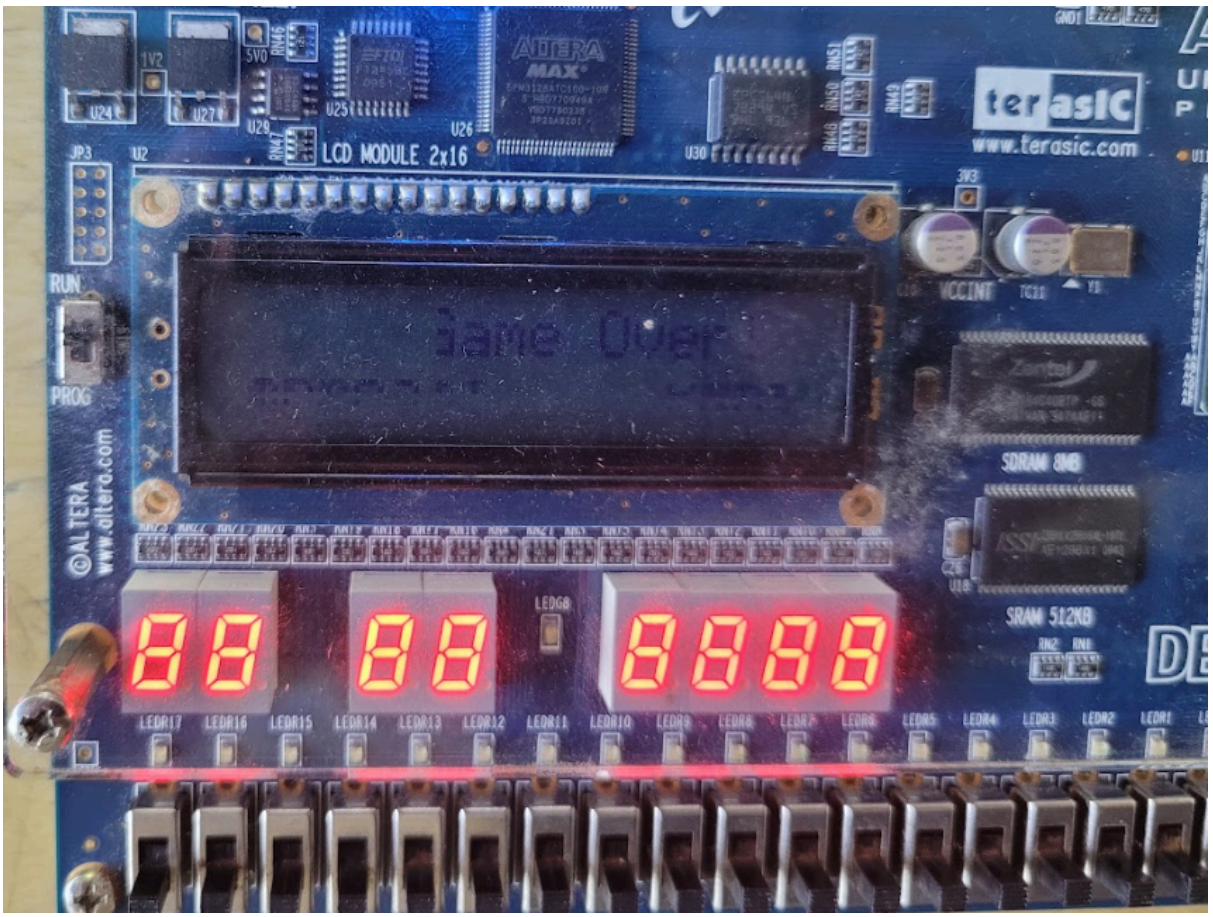
I loaded my game "Hunt Mice", and first came out the welcome page telling the welcome message and the rules of the game. After a few second we chose the level "SUPER" by switching SW2.



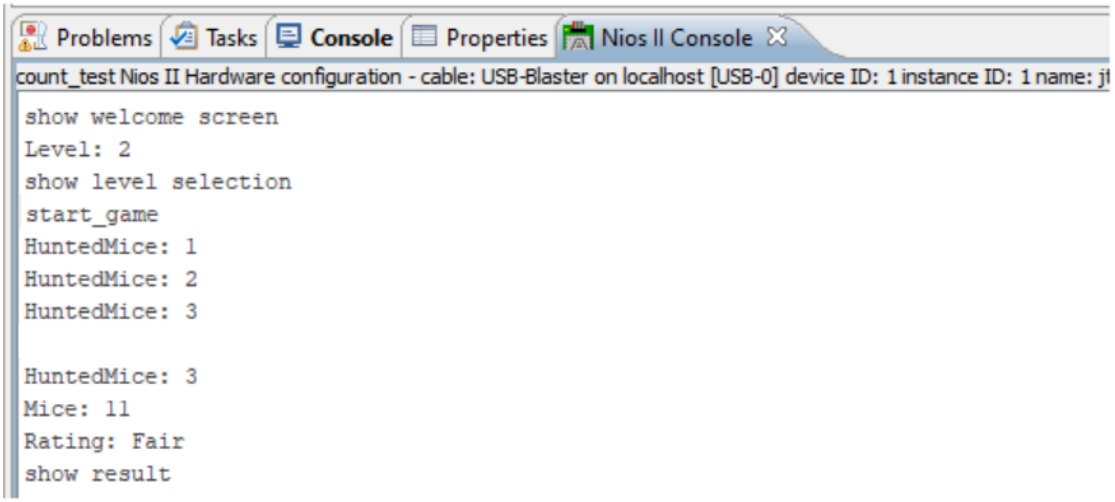
Once I switched the right mouse, it will display “Correct” and hold for 0.5 second. If I was not able to hunt a mouse, it would continue to display the next mouse. As soon as all mice of the level had come out, the result was displayed as Figure.




LCD MODULE 2x16
JP3
U2
RUN
PROG
SW19
©ALTERA
www.altera.com
LEDG8
LEDR17
LEDR16
LEDR15
LEDR14
LEDR13
LEDR12
LEDR11
LEDR10
LEDR9
LEDR8
LEDR7
LEDR6
LEDR5
LEDR4
LEDR3
SDRAM 8MB
SRAM 512K
VCCINT
3V3
U24
U27
U25
U26
U29
U30
5V0
1V2



We also write printf() to print all the steps on Nios II console



6. Reference

SOPC DESIGN BASED ON FPGA AND TOUCH SCREEN

Altera v12.1 SOPC and NIOS LCD Display Counter Lab 6