Zhang Yinuo

SOPC DESIGN BASED ON FPGA AND TOUCH SCREEN

Bachelor's Thesis

Bachelor of Information Technology Program

May 2010

!Unexpected End of Formula

MIKKELIN AMMATTIKORKEAKOULU

Mikkeli University of Applied Sciences

DESCRIPTION

MIKKELIN AMMATTIKORKEAKOULU Mikkeli University of Applied Sciences	**Date of the bachelor's thesis** May 10, 2010
Author(s) Zhang Yinuo	**Degree programme and option** Bachelor of Information Technology

Name of the bachelor's thesis
SoPC design based on FPGA and touch screen

Abstract

The System on a programmable chip (SOPC) is a hot technique nowadays, widely used in control, monitor and assist systems. As a combined technique of programmable logic device (PLD) and application-specific integrated circuit (ASIC), SOPC often integrates hard-core or soft-core CPU, digital signal processor (DSP), memory, I/O and Field Programmable Gate Array (FPGA). Since its high flexibility and cheap price, SOPC is considered as the future of semiconductor industry.

To grasp the idea of SOPC, I chose it as my final project – to see how the SOPC works via making a small game. So the whole task can be divided into two parts, studying the SOPC as well as the related knowledge and completing my design. As the task is based on Nios II Embedded Evaluation Kit, I decided to implement my game involving the FPGA and the LCD touch screen, the two important parts of the system. My design is a game named Hunt Mice.

Subject headings, (keywords)

SoPC, FPGA, Nios II embedded evaluation kit, LCD touch screen

Pages 57	**Language** English	**URN**

Remarks, notes on appendices

Tutor Osmo Ojamies	**Employer of the bachelor's thesis**

ACKNOWLEDGEMENT

I would like to express my gratitude to all those who helped me during the final thesis. First of all, I gratefully acknowledge the help of my supervisor Mr. Osmo Ojamies, who has offered me valuable suggestions in both preparing for programming and writing the paper. Furthermore, high tribute should be paid to Ms. Liu Yifei, with whom I discussed the solutions of the problems during the final thesis, and who has inspired me a lot. I also want to thank Ms. Elina Halonen, who checked the language errors for me.

GLOSSARY

ASIC	Application-Specific Integrated Circuit
CPLD	Complex Programmable Logic Device
DSP	Digital Signal Processor
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
HSMC	High Speed Mezzanine Card
IDE	Integrated Development Environment
JTAG	Joint Test Action Group
LCD	Liquid Crystal Display
LTPS	Low Temperature Poly Silicon
PLD	Programmable Logic Device
SOC	System On a Chip
SOPC	System On a Programmable Chip
VHDL	Very-high-speed-integrated Hardware Description Language

FIGURES AND TABLES

Figures

CONTENT

1 INTRODUCTION

The System on a programmable chip (SOPC) is a hot technique nowadays, widely used in control, monitor and assist systems. As a combined technique of programmable logic device (PLD) and application-specific integrated circuit (ASIC), SOPC often integrates hard-core or soft-core CPU, digital signal processor (DSP), memory, I/O and Field Programmable Gate Array (FPGA). Since its high flexibility and cheap price, SOPC is considered as the future of semiconductor industry. [1]

To grasp the idea of SOPC, I chose it as my final project, to see how the SOPC works via making a small game. So the whole task can be divided into two parts, studying the SOPC as well as the related knowledge, and completing my design. As the task is based on Nios II Embedded Evaluation Kit, I decided to implement my game involving the FPGA and the LCD touch screen, the two important parts of the system. My design is a game named Hunt Mice.

There are 5 chapters in the paper. In the first chapter, I describe the devices and software of my working environment for the task. The second chapter shows the idea of my design – the Hunt Mice game, as well as the structure of the program, the solutions of common problems, and every module in my program in detail. Then, I list the working flow of the task in the third chapter and indicate the performance of the game in the fourth chapter. Finally, the conclusion including my experiences and summarization of the whole project is in the fifth chapter.

2 DESCRIPTION OF WORKING ENVIRONMET

2.1 Devices

I completed the task – the Hunting Mice Game – based on Nios II Embedded Evaluation Kit. It contains an FPGA development broad as well as a LCD screen-touching system, which are its main features related to my task.

2.1.1 Nios II Embedded Evaluation Kit

The Altera® Nios II Embedded Evaluation Kit, Cyclone III Edition is the hardware device of the task, which contains a field-programmable gate array (FPGA) starter board, an LCD Multimedia Daughtercard, hardware and software development tools and their documentation, and so on. The whole system is a system-on-a-chip (SOPC) designing using FPGAs. [2]

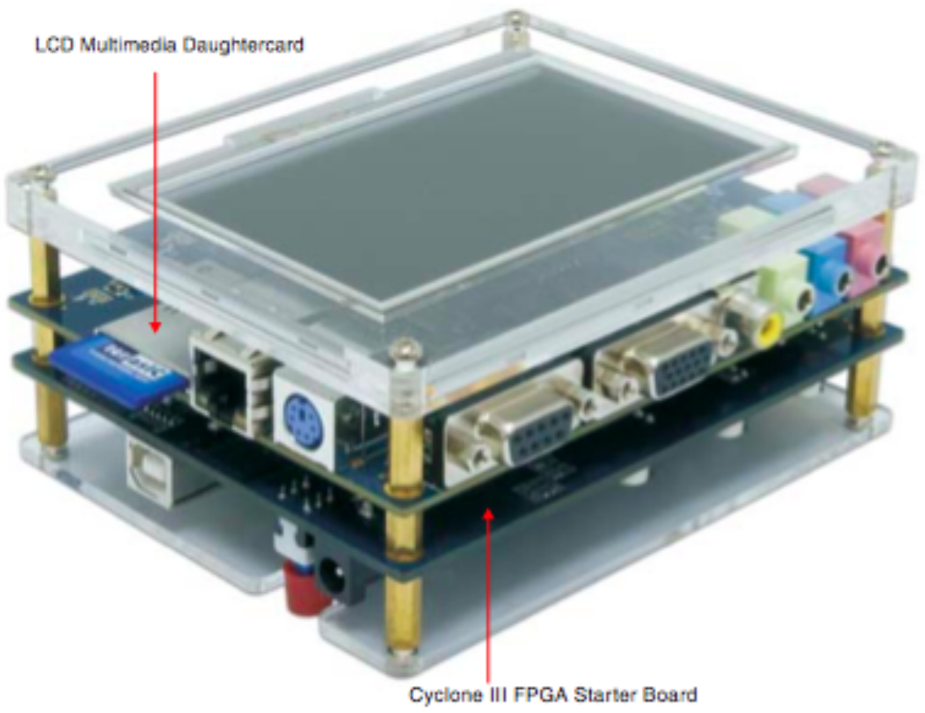


Figure 2-1 the Nios II Embedded Evaluation Kit [7]

The Altera Cyclone III FPGA is in the Starter board. Furthermore, there is an FPGA hardware reference design in its flash memory, meanwhile several sample applications on the SD-Card Flash provided. The main features of the Cyclone III development board are: [2][3]

- Low-power consumption Altera Cyclone III EP3C25 chip in a 324-pin FineLine BGA (FBGA) package
- Expandable through HSMC connector
- 32-Mbyte DDR SDRAM
- 16-Mbyte parallel flash device for configuration and storage
- 1 Mbyte high-speed SSRAM memory
- Four user push-button switches
- Four user LEDs

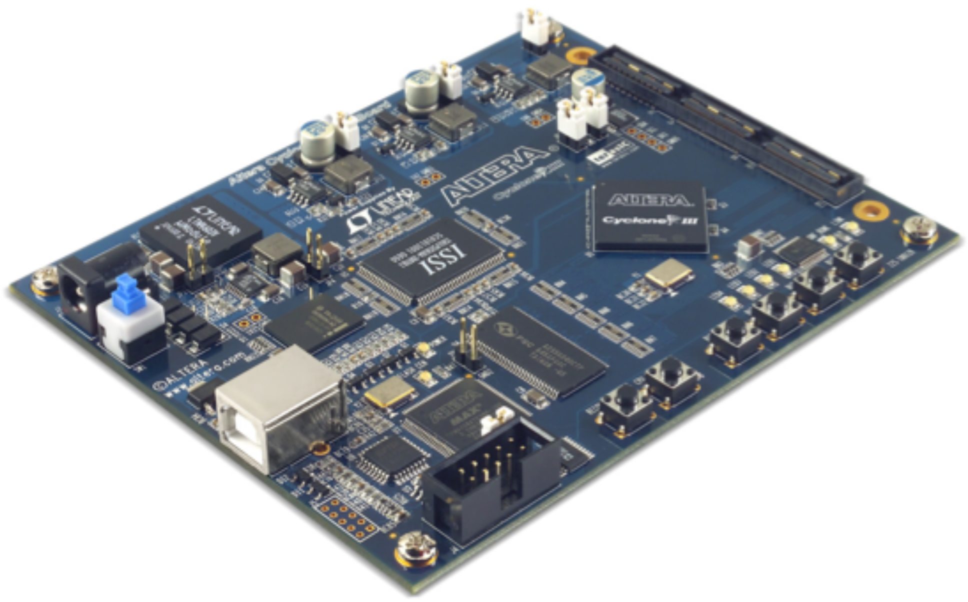


Figure 2-2 the Diagonal View of the Cyclone III FPGA Starter Board [3]

Nios II is designed easily to be configured and customized through the Quartus II, by the C/C++ software application written within the NIOS II IDE.

2.1.2 Embedded System

Nios II is an embedded system. An embedded system is a system used for control, monitor and assistance. Embedded as a part of a complete device, it often contains hardware, or say mechanical parts along with software. With high flexibility, both the software and hardware of an embedded system are programmable. A kind of special computing system as it is, it is widely used in devices with digital I/O. Most embedded systems are implemented by a single program, while some can contain an operating system. [4]

2.1.3 SOPC

The system on a programmable chip (SOPC) is a special embedded system. It is a system on chip (SOC), integrating all components of an electronic system into a single integrated circuit (chip). [5] Meanwhile, it is a programmable system, flexible in designing, improving and updating. In other words, SOPC combines the advantages from both SOC and FPGA. [1]

2.1.4 FPGA

A Field Programmable Gate Array (FPGA) is an integrated semiconductor device, which can be configured by customers after manufacturing. Nowadays, the FPGA is the mainstream of integrated circuit design and verification technology.

FPGAs consist of programmable logic components – the so called "logic blocks", which are "wired together" under the supervision of a hierarchy of reconfigurable interconnects. Logical blocks can be programmed to perform from basic logic gates, such as AND, OR, NOT and XOR, to complex combinational functions. In most FPGAs, there are logical blocks appearing as memory elements, which can be simple flip-flops, or more complicated and complete memory blocks.

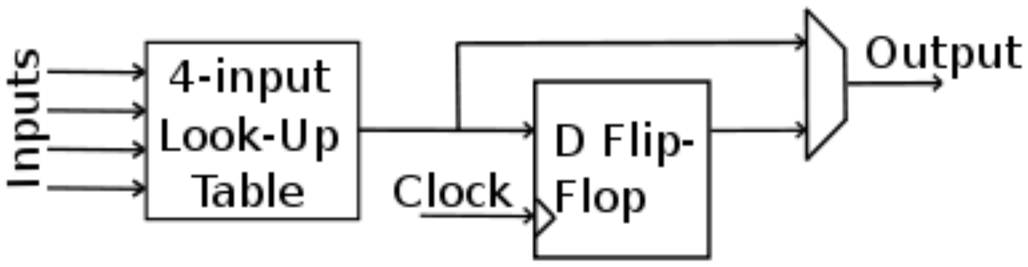


Figure 2-3 a Typical FPGA Logic Block [6]

To program an FPGA, the user can use a schematic design or a hardware description language (HDL), the most common ones of which are VHDL and Verilog. Then, the program is generated into a technology-mapped netlist by an electronic design automation tool, fitted to the actual FPGA architecture. Once verified by the user via simulation tools, it will be downloaded into the FPGA and run to perform the designed functions. [6]

2.1.5 LCD Multimedia Daughtercard

The Daughtercard is a full-featured multimedia card, providing the applications of video, audio and Ethernet, combining with the FPGA Starter board which supports the High Speed Mezzanine Card (HSMC) connectors.

The LCD Multimedia Daughtercard is created to provide a set of interfaces for the applications mentioned above, concretely speaking, which are LCD touch screen, VGA out, composite video in, audio in/out, microphone in, plus Ethernet, SD-Card, PS/2, and RS-232 interfaces. The LCD Multimedia Daughtercard contains the major components as listed: [7]

- MAX II CPLD EPM2210F324
- LCD Touch Panel
- Interfaces
  - HSMC expansion interface
  - Audio codec interface
  - Video decoder interface
  - VGA interface
  - Serial interface
  - PS/2 interface
  - Ethernet
- Clocking circuitry
- Memory
- Power supply

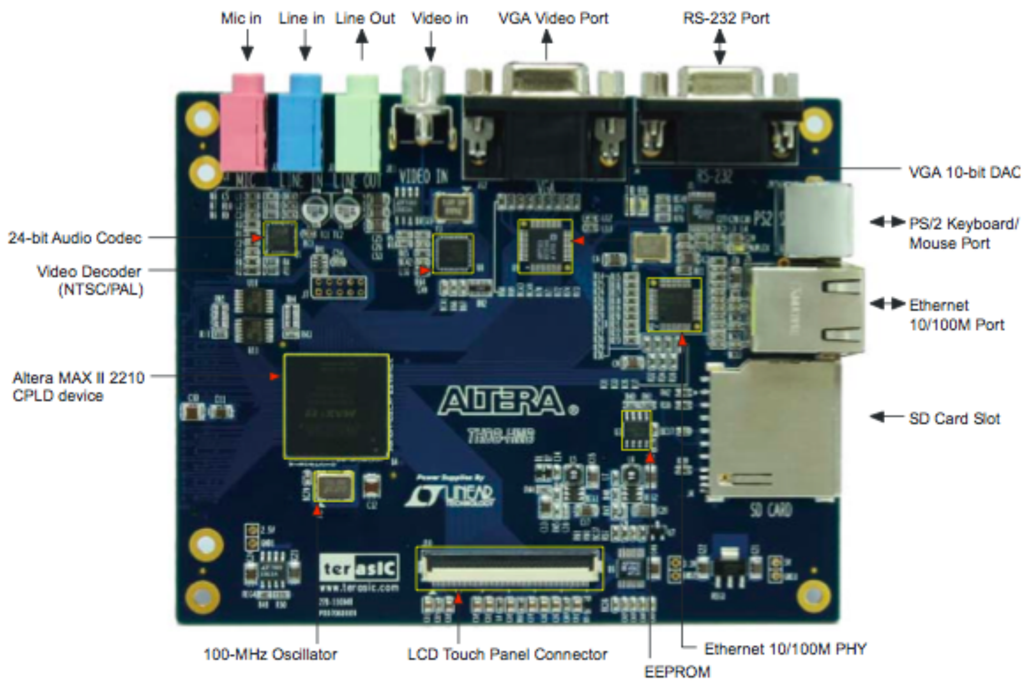


Figure 2-4 Top View of the LCD Multimedia Daughtercard [7]

2.1.5.1 MAX II CPLD

MAX II CPLD (EPM2210F324) plays an essential role in time-division multiplexing of signals for LCD and VGA color data bus, as well as voltage level shifting for FPGA and interfaces chips. To allow for more functionality given the constraint of a pin-limited HSMC connector interface, MAX II CPLD provides the bus controller. [7]

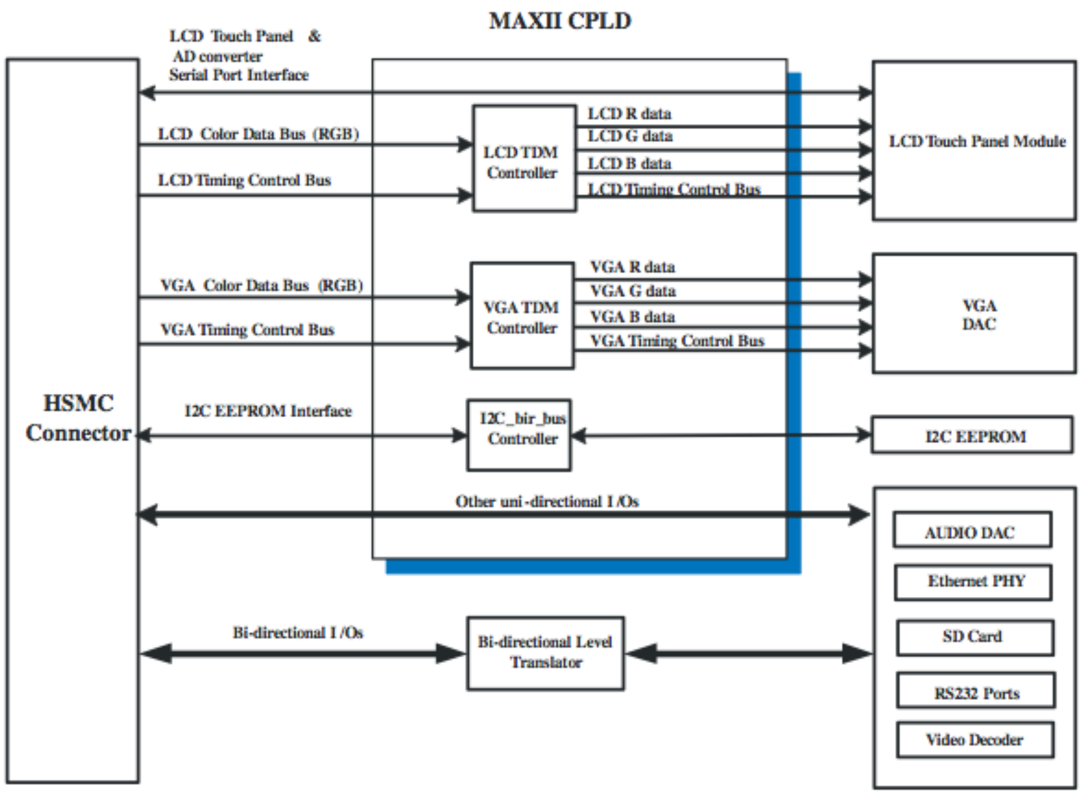


Figure 2-5 the Block Diagram of MAX II Bus Controller [7]

2.1.5.2 LCD Touch Panel

The 4.3” LCD Toppoly TD043MTEA1 module is the active matrix color TFT LCD module, with 800x480 pixel resolution. [7] LTPS (Low Temperature Poly Silicon) TFT technology is applied with vertical and horizontal drivers built on the panel, controlled by the serial interface commands. [8]

The timing protocol of the LCD TDM controller is described in the following figures. Figure 2-6 shows the input timing waveform information of the LCD TDM Controller implemented in the MAX II CPLD. Color pixel data (in the order “BGR”) are carried by an 8-bit wide HC_LCD_DATA, with each pixel described by three successive clock-cycles. A HC_HD pulse is used to sign the position of the BLUE color sample, considered as the start of each three-clock pixel-period. Furthermore, Figure 2-7 shows the timing information on the output side. 24-bit RGB data are generated by the LCD TDM block to the LCD panel, and so is the NCLK clock, with 1/3 the frequency of the incoming clock HC_NCLK. [7]

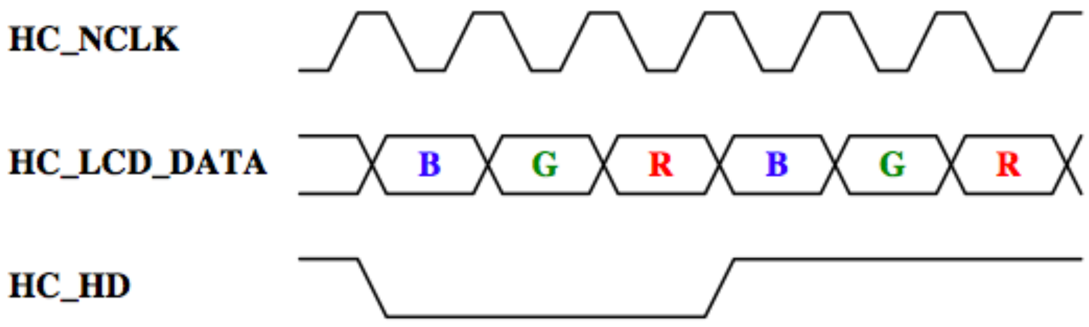


Figure 2-6 the Timing Diagram On the Input Side of VGA TDM Controller [7]

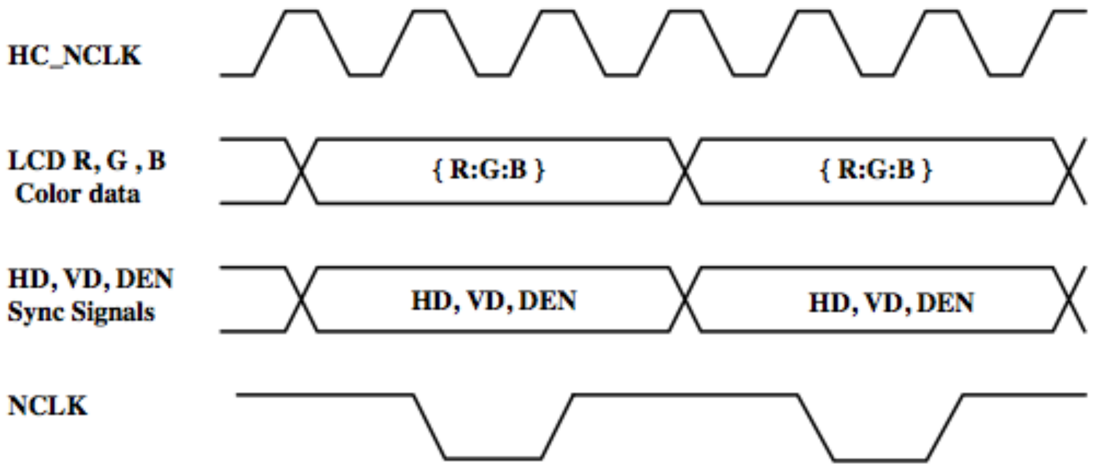


Figure 2-7 the Timing Diagram On the Output Side of VGA TDM Controller [7]

2.2 Software

Development for Nios II consists of two separate steps: hardware generation, and software creation. [9] In the task, I created the software of my project by the NIOS II IDE in C language, and then generated the hardware by the Quartus II.

2.2.1 Nios II IDE

The Nios® II Integrated Development Environment (IDE) is the primary software development tool for the Nios II family of embedded processors. Nios II IDE key features are the following: [10]

- New project wizards and software templates
- Compiler for C and C++ (GNU)
- Source navigator, editor, and debugger
- Eclipse project-based tools

- Software build tools
- Nios II C-to-Hardware (C2H) Acceleration Compiler (licensed separately)
- Complete documentation and training

2.2.2 Quartus II Web Edition

Quartus II is an Altera software intent for HDL designs. Not only the user can program and compile the designs, but they can also simulate and do the timing analysis. After the program is done, Quartus II can also synthesize it. The Web Version is a free version of Quartus II. [11]

3 THE DESIGN OF THE GAME

3.1 Description of the game

This is a mice-hunting game. Welcome message and the rules of the game are shown in the very beginning. When the game starts, on the touch screen will appear the mice - the black squares, which can be hunted by clicking them. Once clicked, the black square will turn green.

There are 3 levels for the game: junior, senior and super, which can be chosen before playing. Measured by the percentage of hunted mice, and valued by four levels (poor, fair, good and perfect), the performance of the player is shown after playing. Then, the player can go back to the welcome page.

3.2 The structure of the program of the game

3.2.1 The flow of the program

As the flow chart (Figure 3-1) shows, the program of the game can be divided into four modules – Main (Welcome and Back), Choose Level, Play Game and Show Result.

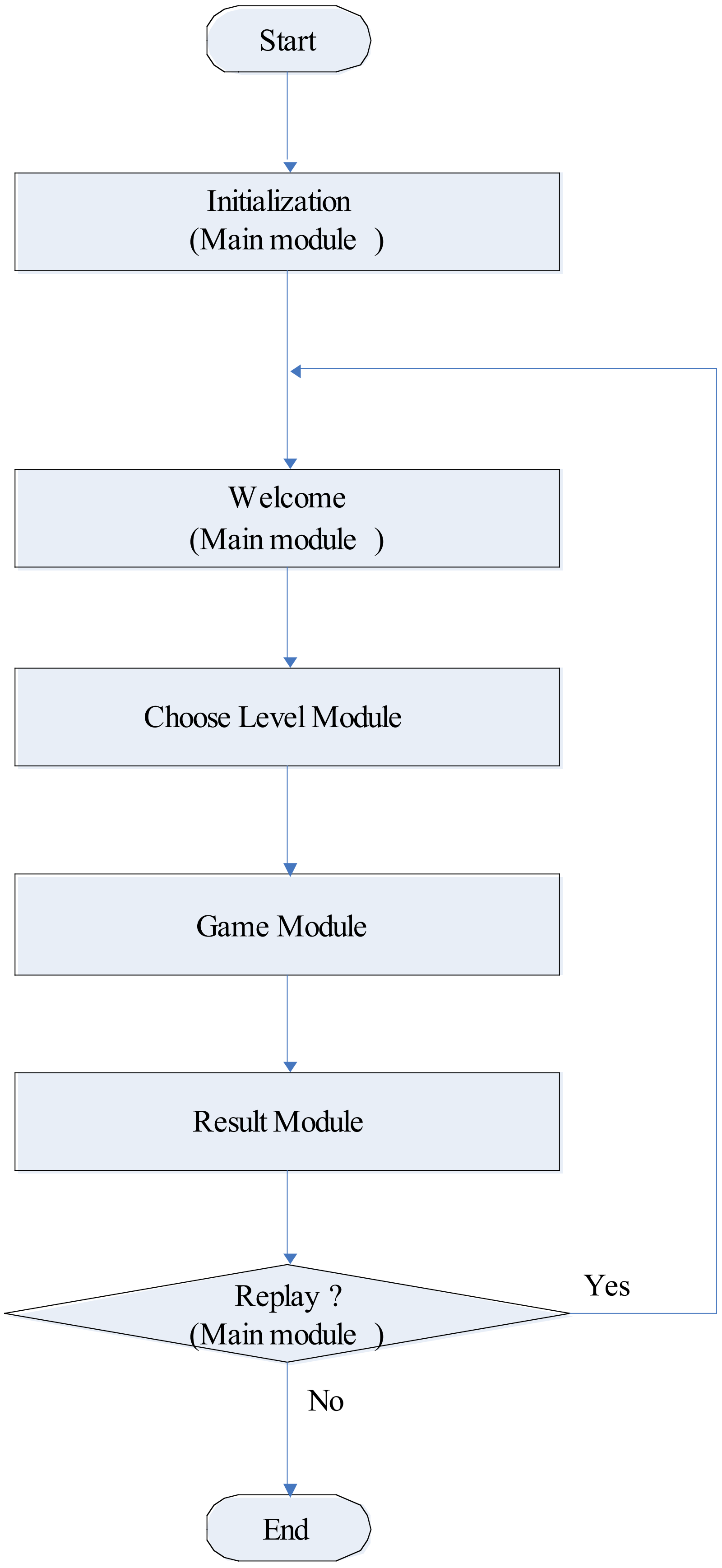


Figure 3-1 the Flow Chart of the Whole Program

3.2.2 Head files

In the program, I used a lot of functions of the library of the C language as well as of the NIOS II IDE system, which are defined in the head files below:

```
#include <unistd.h>

#include <stdio.h>

#include <string.h>

#include <io.h>

#include <stdlib.h>

#include <malloc.h>

#include <altera_avalon_sgdma_regs.h>

#include "alt_touchscreen.h"

#include "altera_avalon_spi_regs.h"

#include "alt_irq.h"

#include "altera_avalon_pio_regs.h"

#include "alt_tpo_lcd.h"

#include "alt_video_display.h"

#include "simple_graphics.h"

#include "alt_cache.h"
```

Among these head files, alt_tpo_lcd.h and alt_video_display.h takes charge of the display of the touch screen, while alt_touchscreen.h translates the touch movement on the screen into the input of the program, and simple_graphices.h contains the functions of display the basic graphs in the screen.

3.2.3 Functions made by myself

I wrote several functions to complete the game, which are listed below:

```
//the main module

int main(void);

//the module of choosing the level

int TouchLevel(alt_touchscreen_scaled_pen_data pen_data,

alt_video_display* display, alt_touchscreen* screen);

//the module of playing the game – hunt the mice

int Game(int MouseUp, alt_touchscreen_scaled_pen_data pen_data,

alt_video_display* display, alt_touchscreen* screen);

//the module of display the result

void Result(int MouseUp, int counter, alt_video_display* display);

//Display the welcome message and the rules of the game

void DisplayWelcome(alt_video_display* display);

//Display the button of choosing the level

void DisplayLevel(alt_video_display* display);

//Used for time control

void delay(int time);
```

3.3 Solutions of common problems and explanation of each module

3.3.1 Solutions of common problems

There are basic several problems that repeatedly appear in different parts of the program. I will describe the solutions of them in front of the explanation of each module. Some of the functions of the solutions are copied from the examples of NIOS II IDE system, others are made by myself.

3.3.1.1 Delay

In the game, as there are 3 speed levels of the mice appearing, I must control the time relatively accurately, however, not according to the International System units of time, such as seconds and milliseconds. Therefore, a delay function is necessary in my program. The delay function is used not only to control the appearing time of each mice during the Game module, but also to insert a little break between pressing the buttons or clicking the mice in the screen and implementing the steps behind them, in order to make the player to see the color changing in the buttons and mice, and aware that where he/she has just touched the screen.

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. The code of the delay function is shown below:

```
void delay(int time)

{

    int i, j, k;

    for(i=0; i<=time; i++)

        for(j=0; j<=2000; j++)  k = 1;

}
```

3.3.1.2 Input from the touch screen

As the interaction between the player and the game is completed by an input via touching the screen, it is necessary to get the position of the touched point – the "pen". There is a function in the library of the examples of the NIOS II IDE system, alt_touchscreen_get_pen, which can get the X and Y coordinates of the location of the pen, as well as return a Boolean value indicating whether the pen is up or down. When pen_down = 0, the X and Y are meaningless. In my program, I used it when choosing the level, hunting the mice, and going back to the welcome page. The code of the function is shown below:

```
unsigned int alt_touchscreen_get_pen (alt_touchscreen* screen,

                      int* pen_down,

                      int* x,

                      int* y)

{
  unsigned int x_adc_sample  = 0;
  unsigned int y_adc_sample  = 0;
  unsigned int sample_number = 0;

  sample_number = get_coherent_pen_data
    (&(screen->pen_state),
     &(screen->pen_adc_data),
     screen->swap_xy,
     pen_down,
     &x_adc_sample,
     &y_adc_sample );

  // Note that we have to scale even if the pen isn't down.
  // The value you get is the LAST LOCATION the pen was seen.
  surface_scaler_compute_output (&(screen->surface_scaler),
                 x_adc_sample,  y_adc_sample,
                 x,             y             );
  return sample_number;
}
```

3.3.1.3 Mimic the pressing feeling of real buttons when clicked

Unlike the buttons on a cell phone, the touch screen cannot provide a special feeling of press. In other words, without seeing the behind steps, the player cannot know whether he/she has touched the screen and where he/she has touched. To solve the problem, I made the buttons and the mice to change their color once touched, which was to redraw a same-size box of another color in the same place as clicked before.

Since the embedded system implemented each statement so fast, I added a delay function after each set of color changing statements, to let the player see the different colors of the object he/she touches. Therefore, I mimicked the feeling of pressing a real button on the touch screen.

Here is an example of that:

```
//get the position of the touched point
alt_touchscreen_get_pen(screen,
                        (&pen_data.pen_down),
                        (&pen_data.x),
                        (&pen_data.y));
//if touch the area of the button “BACK”
if(pen_data.x>=100&&pen_data.x<=200&&pen_data.y>=365&&pen_data.y<=430
)
{
        //redraw a gray button in the former place in former size
        vid_draw_box (100, 365, 200, 430, 0xCCCCCC, 1, display);
        //rewrite the red word “BACK” on the button in the former place
in former font
        vid_print_string(135, 393, 0x9C0036, cour10_font, display,
"BACK");

        //delay for a while to let the player see the changed color

        delay(2000);

        (Omit the following statements…)

}
```

3.3.1.4 Draw in the screen

In the library of the examples of NIOS II IDE system, simple_graphices.h and simple_graphices.c contain the functions of display the basic graphs and strings in the screen. I will introduce those functions I have used in my program.

· To print string on the screen: vid_print_string

The function is to print a string, whose position, color and font are given when calling the function, in the screen. The position is defined by the variables “horit_offset” and “vert_offset”. The color is declared by the variable “color”, whose format is hexadecimal RGB, e.g. 0x000000, while the font by “font”. The function will call the function vid_print_char, which is used to print single characters on the screen.

```
int vid_print_string(int horiz_offset, int vert_offset, int color, char
*font, alt_video_display* display, char string[])
{
  int i = 0;

  int original_horiz_offset;


  original_horiz_offset = horiz_offset;


  // Print until we hit the '\0' char.

  while (string[i]) {

    //Handle newline char here.

    if (string[i] == '\n') {

      horiz_offset = original_horiz_offset;

      vert_offset += 12;

      i++;

      continue;

    }

    // Lay down that character and increment our offsets.

    vid_print_char (horiz_offset, vert_offset, color, string[i], font,
display);

    i++;

    horiz_offset += 8;

  }

  return (0);

}
```

· To draw a line on the screen: vid_draw_line

The function vid_draw_line is to draw line between the two end points. The two end points' position, the color and the width of the line are set when calling the function. To make it faster, the function first checks whether the line is horizontal. If so, it calls vid_draw_horiz_line to draw the horizontal line directly; if not, it calls vid_draw_sloped_line. The function vid_draw_sloped_line draws a line according to the Bresenham algorithm.

```
__inline__ void vid_draw_line(int horiz_start, int vert_start, int
horiz_end, int vert_end, int width, int color, alt_video_display*
display)
{
  if( vert_start == vert_end )
  {
        vid_draw_horiz_line( (unsigned short)horiz_start,
                             (unsigned short)horiz_end,
                             (unsigned short)vert_start,
                             color,
                             display );
  }
  else
  {
        vid_draw_sloped_line( (unsigned short)horiz_start,
                              (unsigned short)vert_start,
                              (unsigned short)horiz_end,
                              (unsigned short)vert_end,
                              (unsigned short)width,
                              color,
                              display );
  }
}
```

・To draw a box on the screen: vid_draw_box

The function is to draw a box in a given place in the screen. Besides the position of top left corner and right bottom corner of the box, the color and the filing signal (the variable “fill”) are defined in the calling. If fill = 0, the box is empty, while if fill = 1, the box is filled by its color.

```
int vid_draw_box (int horiz_start, int vert_start, int horiz_end, int
vert_end, int color, int fill, alt_video_display* display)
{
  // If we want to fill in our box
  if (fill) {
     vid_paint_block (horiz_start, vert_start, horiz_end, vert_end,
color, display);
  // If we're not filling in the box, just draw four lines.
  } else {
    vid_draw_line(horiz_start, vert_start, horiz_start, vert_end-1, 1,
color, display);
    vid_draw_line(horiz_end-1, vert_start, horiz_end-1, vert_end-1, 1,
color, display);
    vid_draw_line(horiz_start, vert_start, horiz_end-1, vert_start, 1,
color, display);
    vid_draw_line(horiz_start, vert_end-1, horiz_end-1, vert_end-1, 1,
color, display);
  }
  return (0);
}
```

3.3.1.5 Clear the screen

I drew a filled box covering the whole screen to clear the screen. The code is shown below:

```
vid_draw_box (0, 0, 799, 479, 0xFFFFF0, 1, display);
```

3.3.2 Main

The Main module contains the initialization, displays the welcome message and the rules, entrance of Choose Level module, and going back to replay. The flow chart of Main module is shown as Figure 3-2.

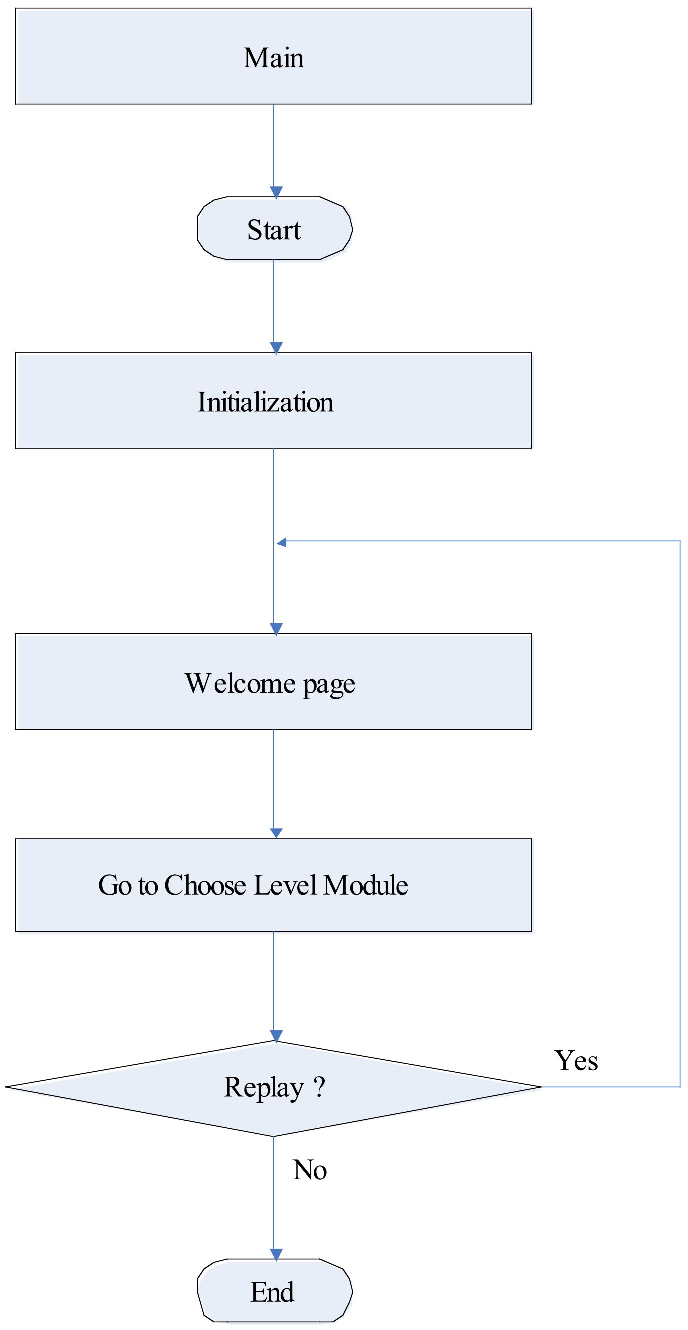


Figure 3-2 Flow Chart of Main Module

The initialization is related to structures of alt_touchscreen and alt_video_display. The variables initialized here are very important to the program, because they are used in every module to get data from the touch on the screen as well as to display there. The code of the initialization is shown below:

```
alt_touchscreen_scaled_pen_data pen_data;
```

```
alt_video_display* display;

alt_touchscreen* screen;

display = alt_video_display_init("/dev/lcd_sgdma", 800, 480, 32,

                                ALT_VIDEO_DISPLAY_USE_HEAP,

                                ALT_VIDEO_DISPLAY_USE_HEAP,

                                1);
```

After the initialization, the statements of displaying the welcome page will be implemented:

```
DisplayWelcome(display);

DisplayLevel(display);
```

I wrote two functions to display the welcome page as above: DisplayWelcome and DisplayLevel. The function DisplayWelcome is to print the strings of welcome message and the rules of the game, while the function DisplayLevel is to draw the background – a light gray ground and a dark red cross, and the cue words of the buttons for choosing the level. The code of the two functions is shown below:

```
void DisplayWelcome(alt_video_display* display)

{

    vid_draw_box (0, 0, 799, 479, 0xFFFFF0, 1, display);

    vid_print_string(270, 135, 0x722E40, cour10_font, display,

                    "^^*!Hunt Mice!*^^");

    vid_print_string(270, 225, 0x722E40, cour10_font, display,

                    "Hunt the mice by clicking the black boxes (the
mice) :p");

    vid_print_string(270, 245, 0x722E40, cour10_font, display,

                    "HAVE FUN~~");

    vid_print_string(270, 315, 0x722E40, cour10_font, display,

                    "Please choose the level:");

}


void DisplayLevel(alt_video_display* display)

{

    vid_draw_box (100, 0, 200, 479, 0x9C0036, 1, display);//dark red

    vid_draw_box (0, 365, 799, 430, 0x9C0036, 1, display);//dark red

    vid_print_string(295, 393, 0xFFFFF0, cour10_font, display,

"JUNIOR");

    vid_print_string(445, 393, 0xFFFFF0, cour10_font, display,

"SENIOR");

    vid_print_string(600, 393, 0xFFFFF0, cour10_font, display,

"SUPER");

}
```

Once any button of the level is touched, the program would enter the Choose Level module. When the game ends and the result is shown, there will be a button “BACK” making the player go back to the welcome page. As mentioned before, once the button “BACK” is clicked, it and its cue word will change the color to gray and red respectively, which will remain the new color for a while by the delay function.

3.3.3 Choose level

The Choose level module deals with the choice of the level. As the main part of the module, the function TouchLevel matches the number of mice for different levels and passes them down to the next steps. The flow chart of Choose Level module is shown as Figure 3-3.

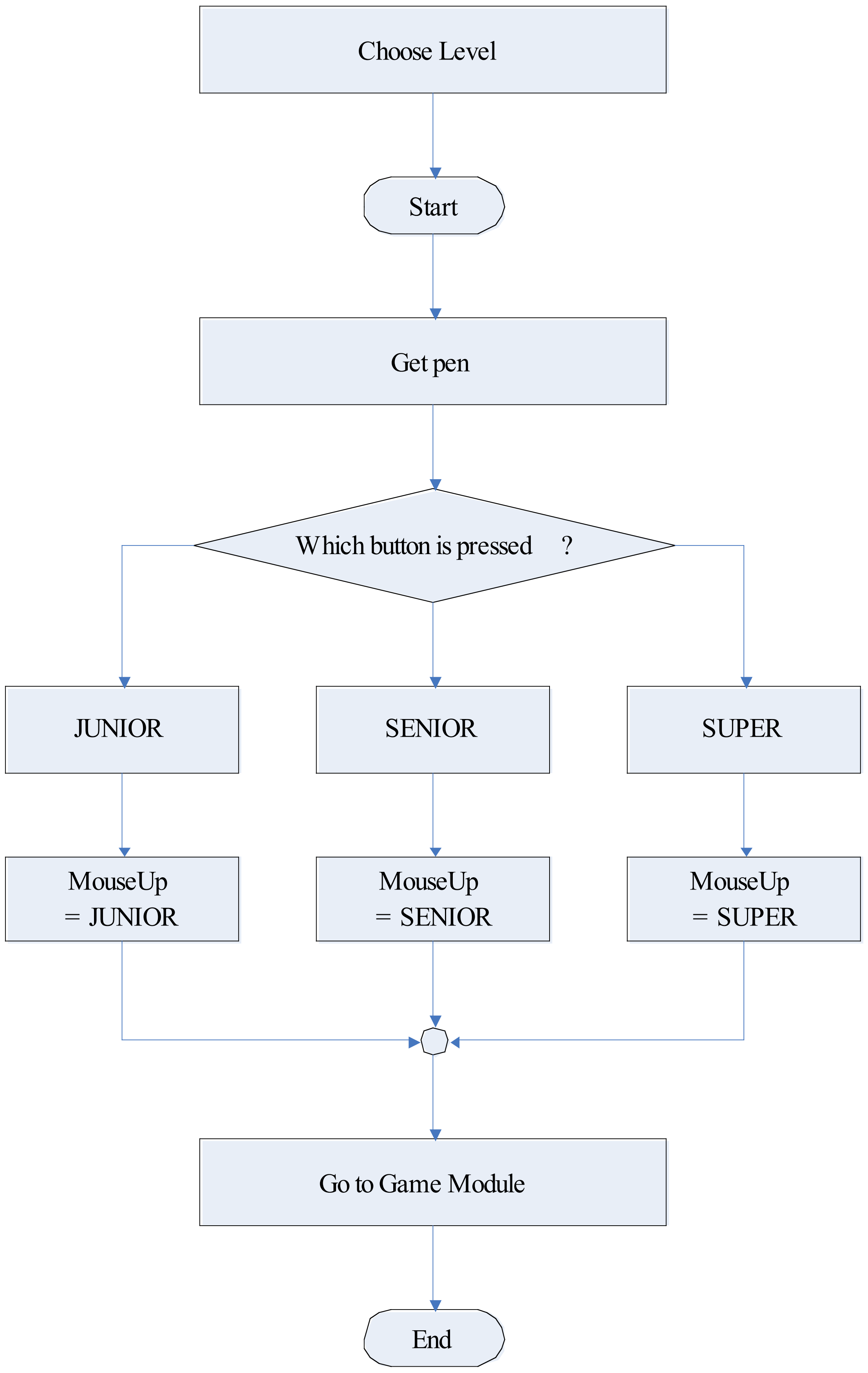


Figure 3-3 the Flow Chart of Choose Level Module

To make the program easy to test, I defined a macro for the number of mice of each level:

```
#define JUNIOR 10

#define SENIOR 20

#define SUPER  30
```

The button “JUNIOR” is located at the area 270 ≤ x ≤ 370 && 365 ≤ y ≤ 430. If the is pen is in the area, the button and the cue word “JUNIOR” will turn to the color of each other. Then the total number of mice – the variable “MouseUp” will be set as “JUNIOR”, and the program will enter the Game module.

```
if(pen_data.x>=270&&pen_data.x<=370&&pen_data.y>=365&&pen_data.y<=430
)

{

        vid_draw_box (270, 365, 370, 430, 0xCCCCCC, 1, display);

        vid_print_string(295, 393, 0x9C0036, cour10_font, display,
"JUNIOR");

        MouseUp = JUNIOR;

        counter = Game(MouseUp, pen_data, display, screen);

}
```

The button “SENIOR” is located at the area 420 ≤ x ≤ 520 && 365 ≤ y ≤ 430. If the pen is in the area, the button and the cue word “SENIOR” will turn the color of each other. Then the total number of mice, the variable “MouseUp” will be set as “SENIOR”, and the program will enter the Game module.

```
else
if(pen_data.x>=420&&pen_data.x<=520&&pen_data.y>=365&&pen_data.y<=430
)
{
        vid_draw_box (420, 365, 520, 430, 0xCCCCCC, 1, display);
        vid_print_string(445, 393, 0x9C0036, cour10_font, display,
"SENIOR");
        MouseUp = SENIOR;
        counter = Game(MouseUp, pen_data, display, screen);
}
```

The button “SUPER” is located at the area $570 \leq x \leq 670$ && $365 \leq y \leq 430$. If the pen is in the area, the button and the cue word “SUPER” will turn the color of each other. Then the total number of mice, the variable “MouseUp” will be set as “SUPER”, and the program will enter the Game module.

```
else
if(pen_data.x>=570&&pen_data.x<=670&&pen_data.y>=365&&pen_data.y<=430
)
{
        vid_draw_box (570, 365, 670, 430, 0xCCCCCC, 1, display);
        vid_print_string(600, 393, 0x9C0036, cour10_font, display,
"SUPER");
        MouseUp = SUPER;
        counter = Game(MouseUp, pen_data, display, screen);
}
```

3.3.4 Game

The Game module is the soul of the whole program. The main part of the module is the function Game. The function 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 (clicked), the black mice will turn green. 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 Game module is shown as Figure 3-4.

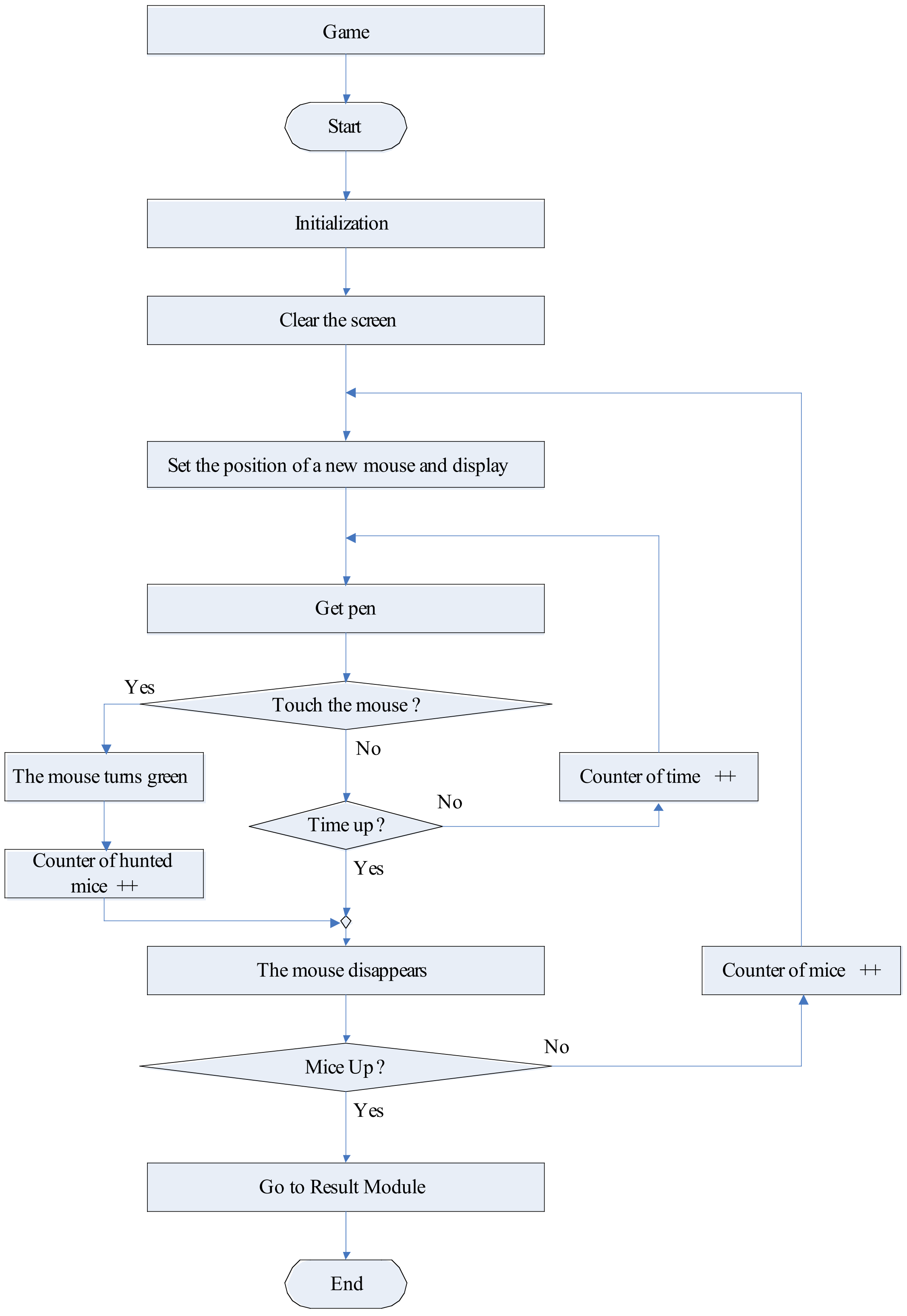


Figure 3-4 the Flow Chart of the Game Module

In the very beginning of the module sets the time “TimeUp” that a mouse will appear in the current game according to the “MouseUp”:

```
switch(MouseUp)
{
        case JUNIOR : {TimeUp = 50; break;}
        case SENIOR : {TimeUp = 25; break;}
        case SUPER  : {TimeUp = 10; break;}
        default     : {TimeUp = 0;  break;}
}
```

The mice of the current game then come out in random places and disappear one by one. In the program, I used black squares with a side length of 50 to stand for the mice. The place of each mouse is set by the function rand():

```
x0 = rand()%(739-210+1)+210;
x1 = x0 + 50;
y0 = rand()%(305-10+1)+10;
y1 = y0 + 50;
```

Then I used a set of nested “for” statements as the timer to count the “lifetime” of each mouse, in order to wait for the hunting movement in the same time. In different levels, there are different “lifetimes” of the mice, defined by the variable “TimeUp”. Once hunted, the mouse will turn green, remaining green 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.

```
for(l=1; l<=MouseUp; l++)
    {
        //SET THE POSITION OF THE MICE
        x0 = rand()%(739-210+1)+210;
        x1 = x0 + 50;
        y0 = rand()%(305-10+1)+10;
        y1 = y0 + 50;

        vid_draw_box (x0, y0, x1, y1, 0x000000, 1, display);

        for(m=1; m<=TimeUp; m++)
            for(n=1; n<=2000; n++)
            {
                alt_touchscreen_get_pen(screen, (&pen_data.pen_down),
                        (&pen_data.x), (&pen_data.y));

                //once hunted, the mouse turns green
if((pen_data.x>=x0)&&(pen_data.x<=x1)&&(pen_data.y>=y0)&&(pen_data.y<
=y1))
                {
                    vid_draw_box (x0, y0, x1, y1, 0x00FF00, 1, display);
                    counter ++;
                    delay(1500);
                    goto DISAPPEAR;
                }
            }
DISAPPEAR:  vid_draw_box (x0, y0, x1, y1, 0xFFFFF0, 1, display);
    }
  vid_draw_box (100, 0, 200, 479, 0x9C0036, 1, display);//dark red
    vid_draw_box (0, 365, 799, 430, 0x9C0036, 1, display);//dark red
    Result(MouseUp, counter, display);
```

3.3.5 Result

The Result module is used for displaying the score of the game, measured by the percentage of the hunted mice among the all the mice of the last game. Furthermore, a comment of the player's performance will be given according to the score. The flow chart of the Result module is shown as Figure 3-5.

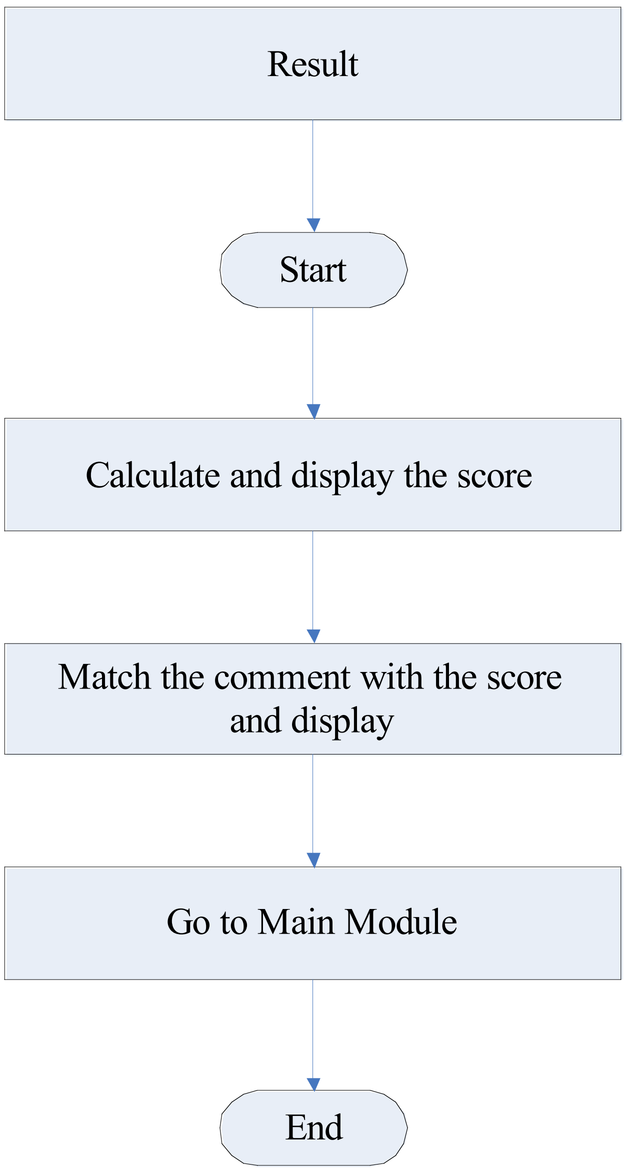

Result
Start
Calculate and display the score
Match the comment with the score and display
Go to Main Module
End

Figure 3-5 Flow Chart of the Result Module

Since the function vid_print_string cannot print things of any data type but string/char, the score needs to be translated into the "string" type, which can be implemented by the function sprintf. The code of calculating and displaying the score is shown below:

```
char ScoreString[10];

score = (int)counter*100 / MouseUp;

sprintf(ScoreString, "%d / 100", score);

vid_print_string(300, 195, 0xFF0000, cour10_font, display,
ScoreString);
```

The comment of performance is given out according to the score. The relationship between the scores and the comments is shown in Table 3-1. The code of display the comment is shown below:

```
if(score < 25)

        vid_print_string(300, 235, 0x722E40, cour10_font, display,

                        "Poor You need more practice.");

else if((score >= 25)&&(score < 50))

        vid_print_string(300, 235, 0x722E40, cour10_font, display,

                        "Fair.");

else if((score >= 50)&&(score < 75)

        vid_print_string(300, 235, 0x722E40, cour10_font, display,

                        "Good!");

else

        vid_print_string(300, 235, 0x722E40, cour10_font, display,

                        "Perfect Congratulations^.^");
```

Table 3-1 Relationship between the scores and the comments

Score	Comment
Score < 25%	Poor　You need more practice.
25% ≤ Score < 50%	Fair.
50% ≤ Score < 75%	Good!
Score ≥ 75%	Perfect　Congratulations^.^

4 WORKING FLOW

In this part, I will describe the steps with which I completed the whole program, from installing the software of NIOS II IDE system to making the final version into a flash file. There were seven steps in all:

· Install the software of the NIOS II IDE

· Create a new C/C++ project

· Program

· Build project and download

· Run the project (test online)

· Modify the project

· Build the final version of the project into a flash file

My working flow is shown as Figure 4-1.

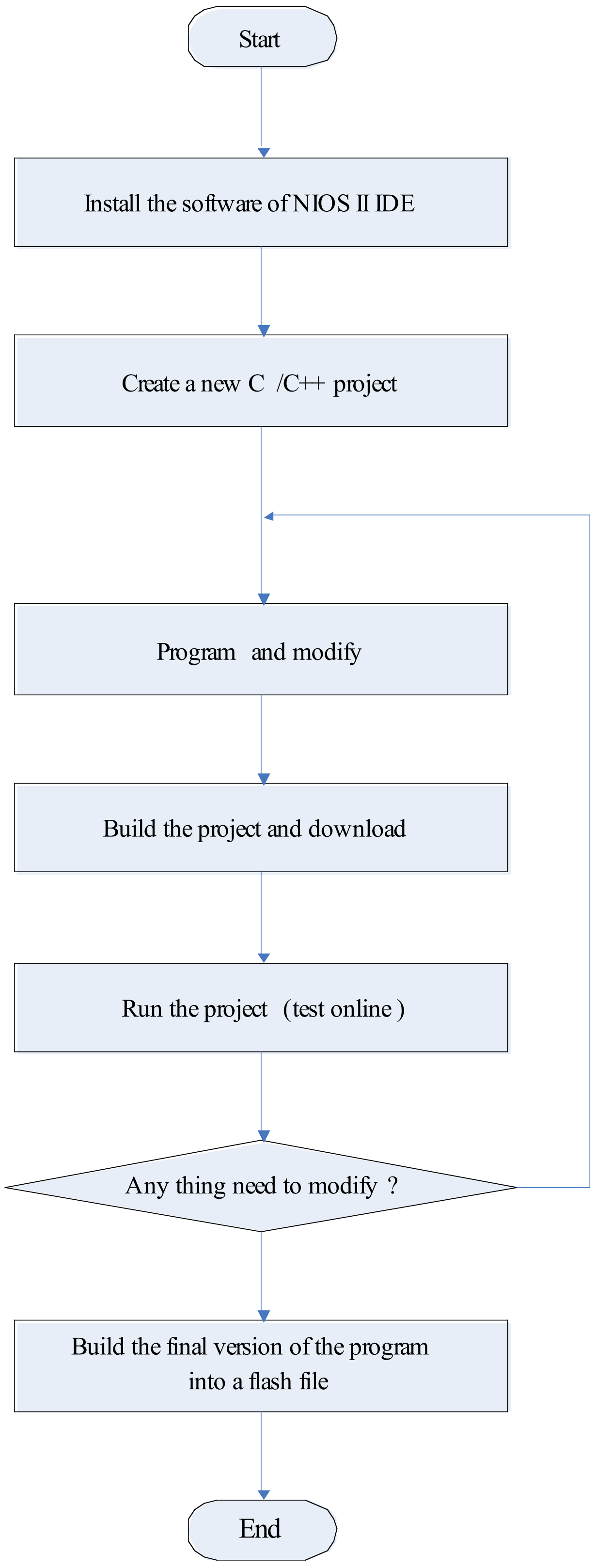

Start
Install the software of NIOS II IDE
Create a new C /C++ project
Program and modify
Build the project and download
Run the project (test online)
Any thing need to modify ?
Build the final version of the program into a flash file
End

Figure 4-1 My Working Flow

4.1 Install the software of NIOS II IDE

To program the Nios II Embedded Evaluation Kit, I installed three pieces of software as the requirements of the “User Guide”[2] :

· The Altera® Complete Design Suite software on the host computer

· The Nios II Embedded Evaluation Kit CD ROM

· The USB-Blaster™ driver software on the host computer. The Cyclone III FPGA starter development board includes integrated USB-Blaster circuitry for FPGA programming.

Among the software of Altera® Complete Design Suite software, Nios II 7.2 IDE is used for programming as well as building the flash file, while Quartus II 7.2 Web Edition for downloading the program to the Cyclone III FPGA of the Nios II Embedded Evaluation Kit. On the other hand, the Nios II Embedded Evaluation Kit is important for completing the whole task, because it contains the guiders and standard design examples offing the library of basic functions to drive the hardware. Besides, the USB-Blaster driver makes the programmer able to download the .sof file of the program via JTAG to test it online.

4.2 Create a new C/C++ project

Open the Nios II 7.2 IDE software, and create a new C/C++ project by choosing File > New > Nios II C/C++ Application to open the New Project wizard. The path is shown as Figure 4-2.

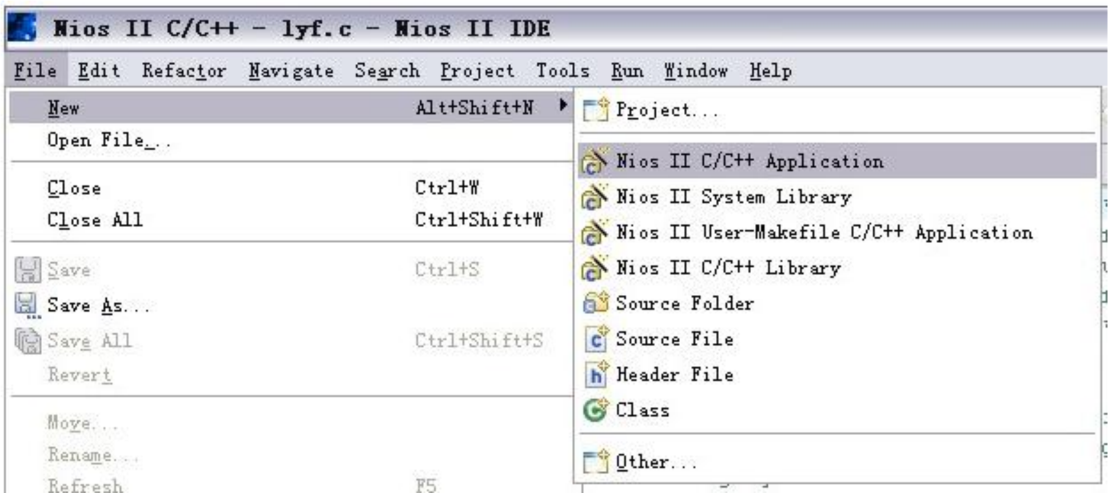
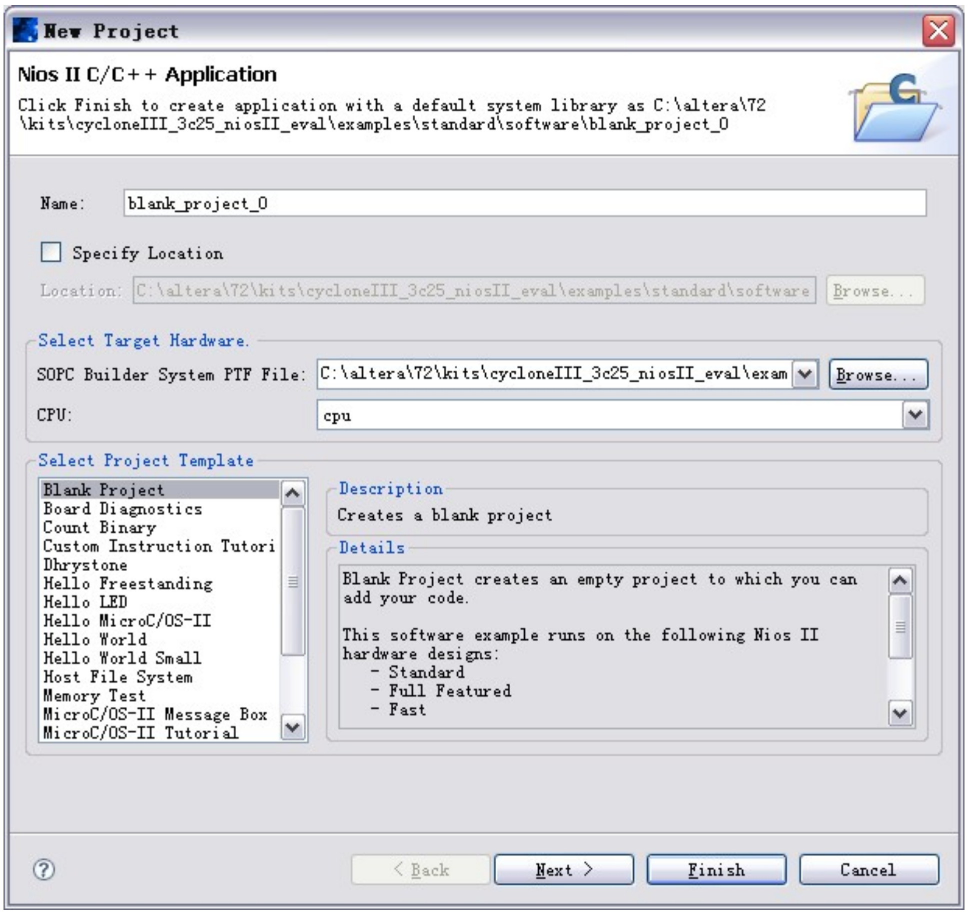


Figure 4-2 the Path to Create a New C/C++ Project

In the New Project dialogue, I set the SOPC Builder System PTF File and selected "Blank project" in the Select Project Template list, as the Figure 4-3.



Click Finish to create application with a default system library as C:\altera\72\kits\cycloneIII_3c25_niosII_eval\examples\standard\software\blank_project_0

Name: blank_project_0

Specify Location

Location: C:\altera\72\kits\cycloneIII_3c25_niosII_eval\examples\standard\software

SOPC Builder System PTF File: C:\altera\72\kits\cycloneIII_3c25_niosII_eval\exam

CPU: cpu

Blank Project
Board Diagnostics
Count Binary
Custom Instruction Tutori
Dhrystone
Hello Freestanding
Hello LED
Hello MicroC/OS-II
Hello World
Hello World Small
Host File System
Memory Test
MicroC/OS-II Message Box
MicroC/OS-II Tutorial

Creates a blank project

Blank Project creates an empty project to which you can add your code.

This software example runs on the following Nios II hardware designs:
- Standard
- Full Featured
- Fast

Figure 4-3 Setting in the New Project dialogue

4.3 Build the project and download

Preparing for downloading the program to the kit, I first should compile the project by Build Project, which is to right click my project and choose "Build Project", shown as Figure 4-4.

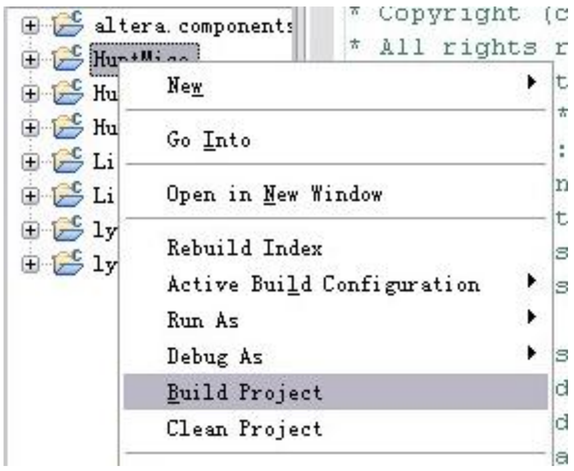


Figure 4-4 Build Project

Then I needed to download the .sof file of my project to the kit. Opened the Quartus II software via Tool > Quartus II Programmer in the Nios II IDE dialogue, and added the .sof file of my project by clicking “Add File…” in the Quartus II Programmer dialogue, followed by clicking “Start” to download the file.

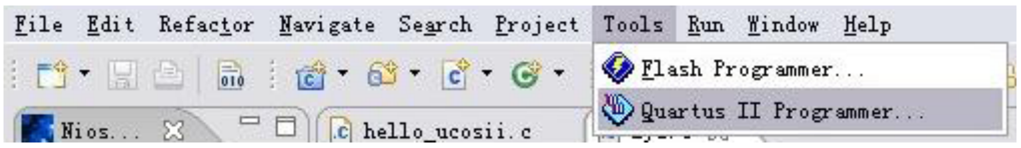
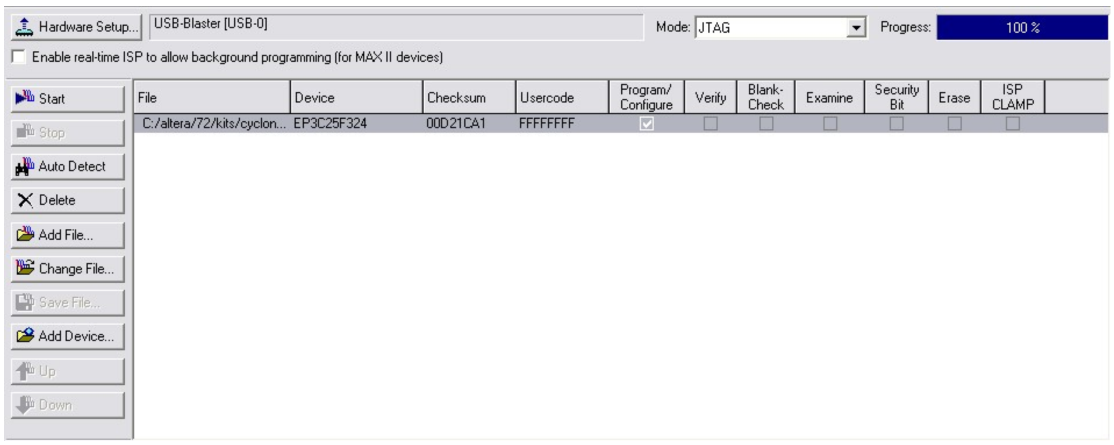

File Edit Refactor Navigate Search Project Tools Run Window Help
Flash Programmer...
Quartus II Programmer...

Figure 4-5 Open the Quartus II Programmer



USB-Blaster [USB-0] Mode: JTAG Progress: 100 %

Enable real-time ISP to allow background programming (for MAX II devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine	Security Bit	Erase	ISP CLAMP
C:/altera/72/kits/cyclon...	EP3C25F324	00D21CA1	FFFFFFFF	✓	☐	☐	☐	☐	☐	☐

Figure 4-6 Add File and Download

4.4 Run the project (test online)

Downloaded to the kit, my project could run in the kit, or be tested in the hardware. The way is to right click my project and choose “Run As > Nios II Hardware”, shown as Figure 4-7.

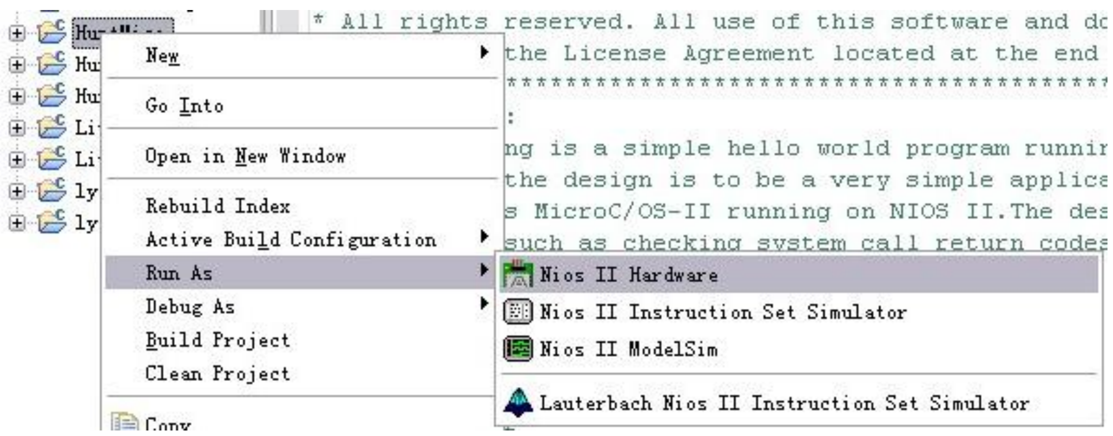

```
* All rights reserved. All use of this software and dc
the License Agreement located at the end
*****************************************
:
ng is a simple hello world program runnir
the design is to be a very simple applice
s MicroC/OS-II running on NIOS II.The des
such as checking system call return codes
```

Figure 4-7 Run the Project

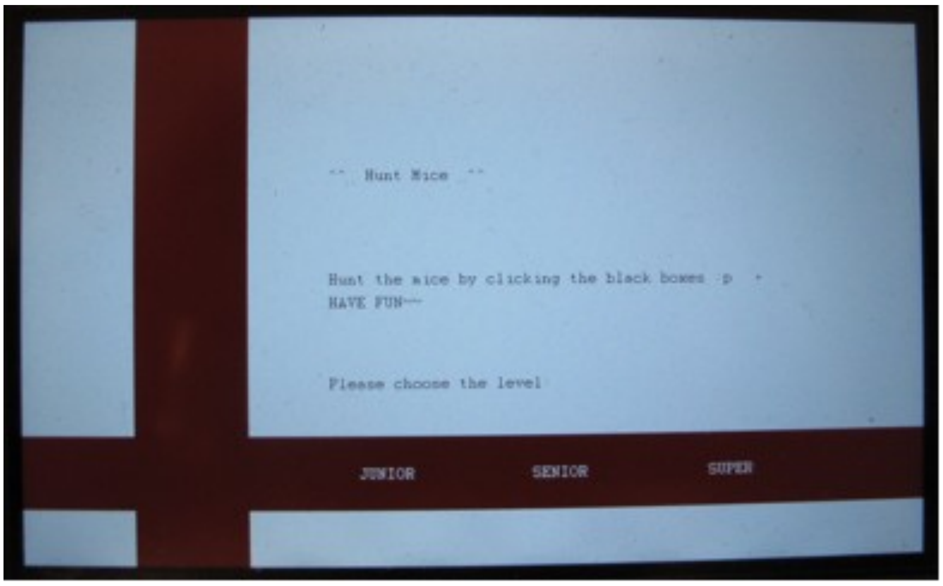
Since the downloading is via JTAG (via the USB-Blaster driver), during the downloading and running time, the kit should be connected to the host computer. In other words, I tested and modified the program by running as Nios II hardware online.

4.5 Build the final version of the program into a flash file

When I finished the program, I needed to store it somewhere, instead of always running in the hardware online. As the embedded system can load applications from a SD card, I built my final program into a flash file to store in the SD card. There are detailed instructions in the “User Guide”, telling how to build the project into a flash version.

5 THE PERFORMANCE OF THE GAME

This part is to indicate the performance of the flash file of the project of the game. As I loaded my game “Hunt Mice by ZYN” from the SD card, first came out the welcome page telling the welcome message and the rules of the game, shown as Figure 5-1.



^^ Hunt Mice ^^

Hunt the mice by clicking the black boxes :p ~
HAVE FUN~~

Please choose the level

Figure 5-1 the Welcome Page of the Game

Then I chose the level “JUNIOR” by clicking the button “JUNIOR”. The button “JUNIOR” turned to light gray, while the cue word “JUNIOR” to dark red; the game of junior level therewith began, shown as Figure 5-2.

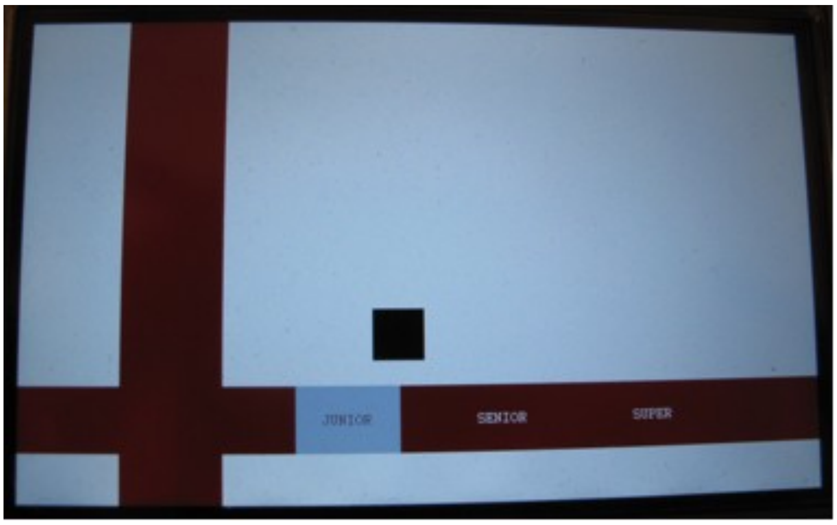
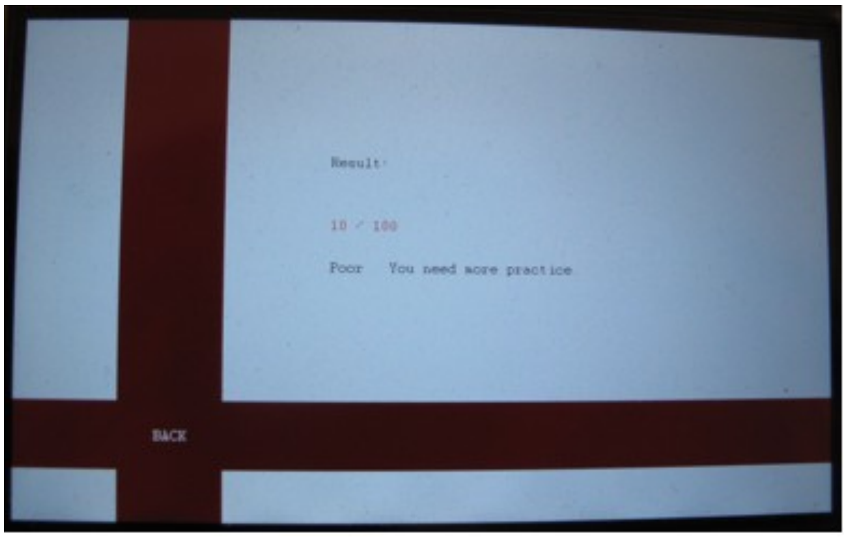


Figure 5-2 During the Game

Once I clicked a mouse, the black square, it turned to green immediately. If I was not able to hunt a mouse, it remained black until “dead”, or say “disappeared”. As soon as all mice of the level had come out, the result was displayed as Figure 5-3. At that time, the button “BACK” appeared, which I clicked to go back to the welcome page.



```
Result:

10 / 100

Poor  You need more practice
```

Figure 5-3 the Result of a Game

6 CONCLUSIONS

SOPC is the product of the new generation. Aiming at learning the theory and the operating principles of SOPC, I programmed a game named Hunt Mice, which was to hunt the mice by clicking their position on the touch screen. The hardware environment is Nios II Embedded Evaluation Kit, and I mainly focused on the FPGA and the LCD touch screen. I wrote the game by C language in Nios II IDE, and downloaded it into the FPGA by Quartus II.

During my task, I came across with many challenges. The largest challenge appeared in the very beginning of my programming: I could not download my own program into the FPGA following the steps in the user guide. After several days of study, I finally found out the reasons. One was that I had not added enough head files related to the hardware into my program. Another was I chose to use a Blank Project to build my program, instead of the Hello World project which was recommended by the user guide, so I could not go through the following steps of downloading. In that situation, I changed the order of the steps of downloading, first to download the program via Quartus II, and then to run my own program as Nios II hardware in Nios II IDE, which was contrary to the steps in the user guide. I finally succeeded to run my own project in the FPGA online.

Another impressing challenge was to mimic the touch feeling of buttons on the screen. The purpose of that was to make the player have enough time to aware where he/she has just clicked. Since the FPGA implements the commands very fast, much faster than the response time of human beings, the player could not realize the FPGA's implementation command by command. So I inserted a delay time after each point that needed to be paid attention to. Meanwhile, to make the player to see what he/she has clicked better, I mimicked the touch feeling of buttons on the screen by changing the color of buttons and the mice and keeping the changing displaying on the screen for a while. In the end, the game reached a very good performance level, thank to the solution.

Although I have solved many problems during the task, there are still some points needing future development. The Nios II Embedded Evaluation Kit is too "wasteful" for such a simple game. The sample applications restored in the SD card of the system shows that the device works perfectly in a colorful picture and video display, the sound playing and Internet accession. To go further, I can use pictures of the mouse standing for the mice instead of black squares, and add some sound when hunting a mouse while some video to tell the result of a game. Even I can connect the Internet with the game, to make people compete via the Internet.

To sum up, I have really learned a lot while done the task. Not only did I grasp the knowledge about the structure and the principles of SOPC and related issues, but I also experienced a whole process of solving a problem from the very beginning, planning for the whole task, searching for a solution when facing challenges and finally, succeeding in the end.

REFERENCES:

[1] SOPC_百度百科

<URL: http://baike.baidu.com/view/525684.htm?fr=ala0_1>

[2] Nios II Embedded Evaluation Kit, Cyclone III Edition, User Guide

[3] Cyclone III FPGA Starter Board, Reference Manual

[4] Embedded system – From Wikipedia, the free encyclopedia

<URL: http://en.wikipedia.org/wiki/Embedded_system>

[5] System-on-a-chip – From Wikipedia, the free encyclopedia

<URL: http://en.wikipedia.org/wiki/System-on-a-chip>

[6] Field-programmable gate array – From Wikipedia, the free encyclopedia

<URL: http://en.wikipedia.org/wiki/FPGA>

[7] LCD Multimedia Daughtercard, Reference Manual

[8] LTPS LCD Specification, Model Name: TD043MTEA1

[9] Nios II – From Wikipedia, the free encyclopedia

<URL: http://en.wikipedia.org/wiki/Nios_II>

[10] Software Development Tools for the Nios II Processor

<URL: http://www.altera.com/products/ip/processors/nios2/tools/ide/ni2-ide.html>

[11] Quartus II – From Wikipedia, the free encyclopedia

<URL: http://en.wikipedia.org/wiki/Quartus_II>