Liu Yifei

CHINESE CALENDAR SYSTEM BASED ON SOPC DESIGN

Bachelor's Thesis
Bachelor of Information Technology Program

May 2010

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

DESCRIPTION

MIKKELIN AMMATTIKORKEAKOULU	**Date of the bachelor's thesis** May 10th, 2010
Author(s) Liu Yifei	**Degree programme and option** Bachelor of Information Technology
Name of the bachelor's thesis Chinese calendar system based on SoPC design	

Abstract

Nowadays, with the development of electronic technology, Programmable Logic Device has become more and more important in the electronic industry. From the earliest ROM, to PAL and GAL, then CPLD and FPGA, PLD have developed for more than forty years. In recent years, FPGA has becomes the leading sector.

Besides, touch screen products, emerged in the second half of the 1940s, has also become more and more popular with their convenience to use. It is an electronic visual output that can detect the presence and determine the location of a touch within the display area. It can indicates the presence of a touchscreen by interacting physically with what is typically shown on a display.

My aim of this final project is just to achieve my own design with the basic knowledge of FPGA in touchscreen system. The idea is to make a system which can convert the year of the Gregorian calendar into the name of that year based on the Chinese calendar. It is a design which could combine traditional culture with modern technology.

Subject headings, (keywords)
FPGA, SoPC, Cyclone III starter board, LCD, Quartus II, Nios II IDE, Chinese calendar system

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

Remarks, notes on appendices

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

GLOSSARY

ROM	Read-Only Memory
PAL	Programmable Array Logic
GAL	Generic Array Logic
CPLD	Complex Programmable Logic Device
FPGA	Field Programmable Gate Array
PLD	Programmable Logic Device
IDE	Integrated Development Environment
SOPC	System-on-a-Programmable-Chip
PROM	Programmable Read-Only Memory
EPROM	Erasable Programmable Read-Only Memory
CLB	Configurable Logic Block
HDL	Hardware Description Language
RTL	Register Transfer Level
SOC	System On Chip
DSP	Digital Signal Processing
VHDL	Very-High-Speed Integrated Circuit HardwareDescription Language

FIGURES

CONTENTS

1. Introduction

Nowadays, with the development of electronic technology, Programmable Logic Device has become more and more important in the electronic industry. From the earliest ROM, to PAL and GAL, then CPLD and FPGA, PLD have developed for more than forty years. In recent years, FPGA has becomes the leading sector.

Besides, touch screen products, emerged in the second half of the 1940s, has also become more and more popular with their convenience to use. It is an electronic visual output that can detect the presence and determine the location of a touch within the display area. It can indicates the presence of a touchscreen by interacting physically with what is typically shown on a display. [1]

My aim of this final project is just to achieve my own design with the basic knowledge of FPGA in touchscreen system. In ancient China, there is another kind of method to count the years, which is much different compared to the calendar we are now using. So in this project, my idea is to make a system which can convert the year of the Gregorian calendar into the name of that year based on the Chinese calendar. It is a design which could combine traditional culture with modern technology.

Altera is a company that is a major manufacturer of high-end PLDs. PLDs can be programmed again during the design cycle as well as in the field to perform multiple functions, and they support a fast design process.[2] So in my project, the main devices are chosen from it. I use Nios II Embedded Evaluation Kit and Cyclone III starter board as the FPGA part, Nios II 7.2 IDE as the software of program design environment, together with Quartus II 9.1 Web Edition as the software to download program into the kit.

This paper can be divided into four parts. The first section is the background of this project, in which I will give a brief introduction of PLD and SOPC. The second section will focus on the hardware and software devices I used to achieve my design. The third section is the main part of my thesis, which elaborates on my design and the program code to complete the whole idea. The fourth section will be the steps to achieve the program and download it into the board using those software. At last, I will give a short conclusion.

2. Background of the project

2.0 Overview

As my project is based on the theory like PLD and SOPC, in this chapter, I will focus on the background knowledge of my project.

2.1 Development of PLD

A programmable logic device or PLD is an electronic component which is used for building digital circuits that could be reprogramed. Unlike a normal logic gate, which has a static function, a PLD has an undefined function when manufactured. Before the usage of PLD in a circuit, it must be programmed.[3]

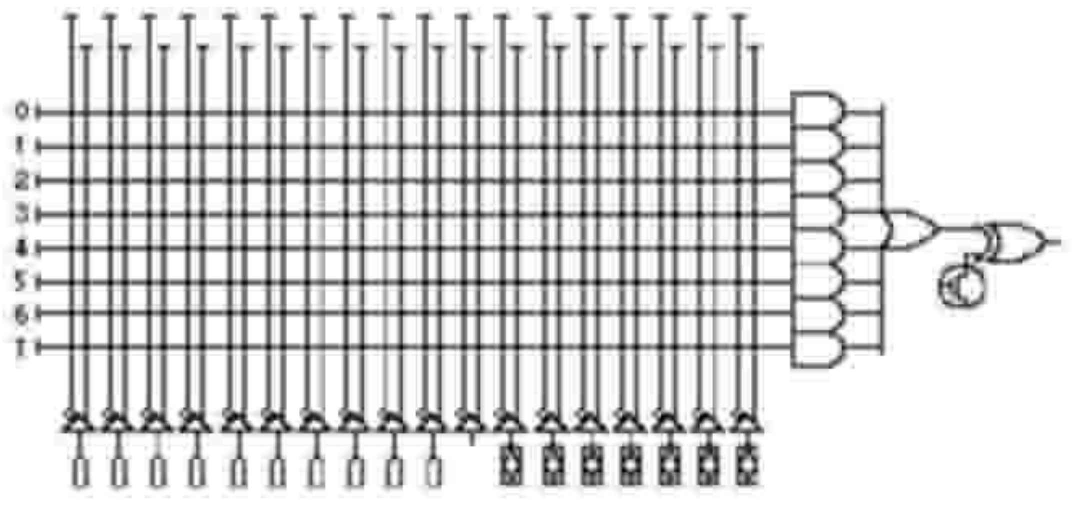


Figure 2-1 Stucture of PLD[6]

Before PLDs were invented, read-only memory (ROM) chips were used to create different combinational logic functions of a number of inputs.

The early programmable devices only included programmable ROMs or PROMs, ultraviolet-erasable PROMs or EPROMs, and electrically erasable PROMs or EEPROMs. They could only accomplish some simple logic functions. There are advantages to use ROM, for example, they could be programmed using a standard programmer instead of more specialised hardware or software. But the disadvantages are also obvious. For example, they are slower than static logic circuits, they consume more power, and they are often more expensive than programmable logic, especially when a higher speed is needed and so on

In the 1970s, the ROM was developed into a real PLD. In 1970, Texas Instruments developed a mask-programmable IC based on the IBM read-only associative memory or ROAM, created the Programmable Logic Array for this device. After that, several products emerged based on PROM, EPROM and EEPROM. In this period, PAL and GAL played the chief roles.[3]

The term Programmable Array Logic (PAL) was introduced by Monolithic Memories, Inc. (MMI) in March 1978. The main composition of PAL devices includes a small PROM (programmable read-only memory) core with output logic, which is used for implementing specialised logic functions with few components. The PAL architecture consists of two main components: a logic plane and output logic macrocells. The logic plane is that programmable read-only memory (PROM) array which allows the signals present on the devices pins routing to an output logic macrocell. [4]

An innovation of the PAL was the generic array logic device, or GAL, which was invented by Lattice Semiconductor in 1985. This device is almost the same as the PAL,especially in logical properties, but it can be reprogrammed. Lattice GALs combine CMOS and electrically erasable (E^2) floating gate technology to reach higher speed and lower power.

PALs and GALs are available only in small sizes, equals to a few hundred logic gates. Because of the request for bigger logic circuits, complex PLDs or CPLDs appeared. These link several PALs by programmable interconnections in one integrated circuit. CPLDs can replace thousands, or even hundreds of thousands, of logic gates.[3]

While PALs were developing into GALs and CPLDs, there happened a separate stream of development which is based on gate array technology. These are the most popular devices used nowadays, called field-programmable gate array (FPGA). FPGAs can be used to implement any logical function an ASIC could perform and it has become the main trend in electronic technology development. It is also the basic theory of my project.

2.2 FPGA

2.2.1 Structure of FPGA

A field-programmable gate array (FPGA) is an integrated circuit which can be configured by the customer themselves after manufacturing. That reflects the so called "field-programmable". The common FPGA structure contains the following components: an array of configurable logic blocks (CLBs), I/O pads, and routing channels. Generally, all the routing channels have the same number of wires, known as width. An Altera Stratix IV GX FPGA is shown below. [5]

Figure 2-2 An Altera Stratix IV GX FPGA [5]

The most important part of FPGA should be logic blocks, which are shown in Figure 2-3, accompanied by a hierarchy of reconfigurable interconnects that wire the blocks together. One can configure logic blocks to perform complex combinational functions and of course some simple logic gates such as AND and XOR. In most FPGAs, the logic blocks also contain memory elements, for example simple flip-flops, or more complete blocks. [5]

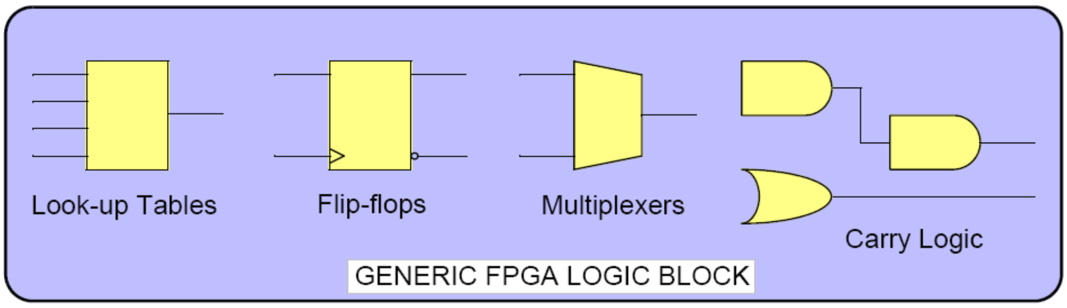


Figure 2-3 Generic FPGA logic blocks[7]

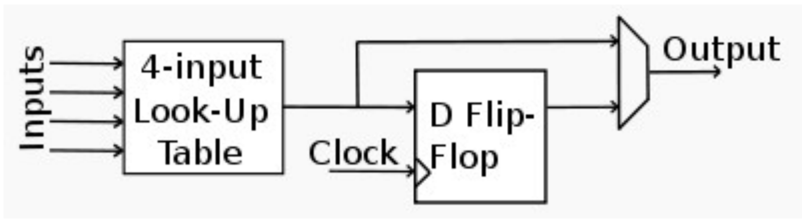


Figure 2-4 Typical logic block [5]

Compared with CPLD, a Field Programmable Gate Array (FPGA) contains more smaller individual blocks and has a larger interconnection structure dominating the entire chip. The basic architecture of FPGA is shown below.

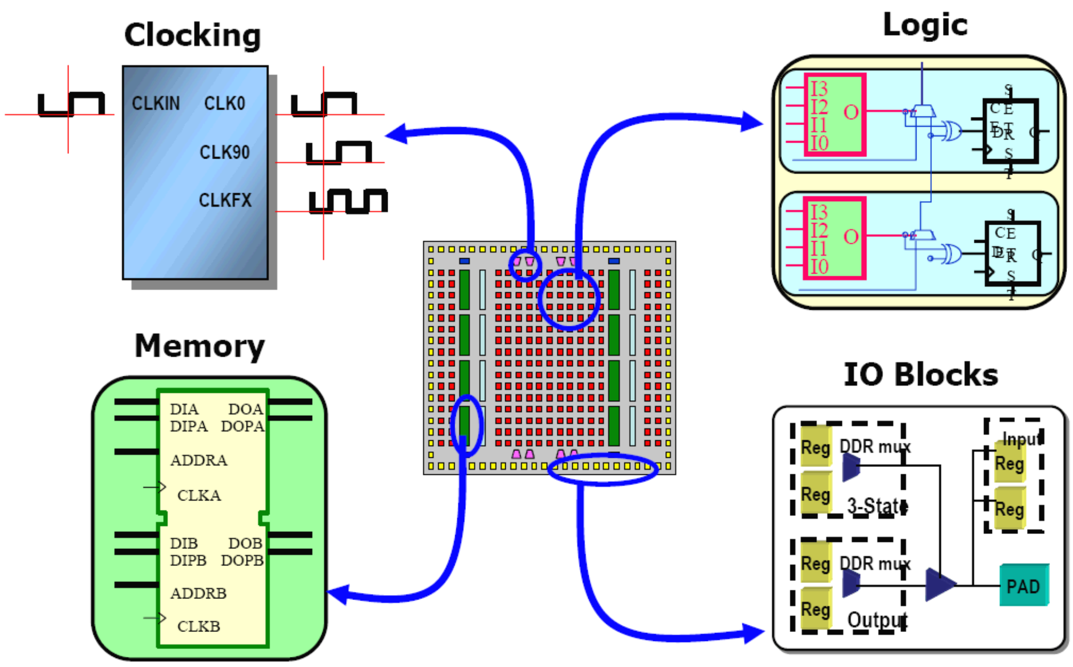


Figure 2-5 Basic FPGA architecture[7]

2.2.2 Design flow

To design a FPGA, the following steps are usually involved in the design flow:

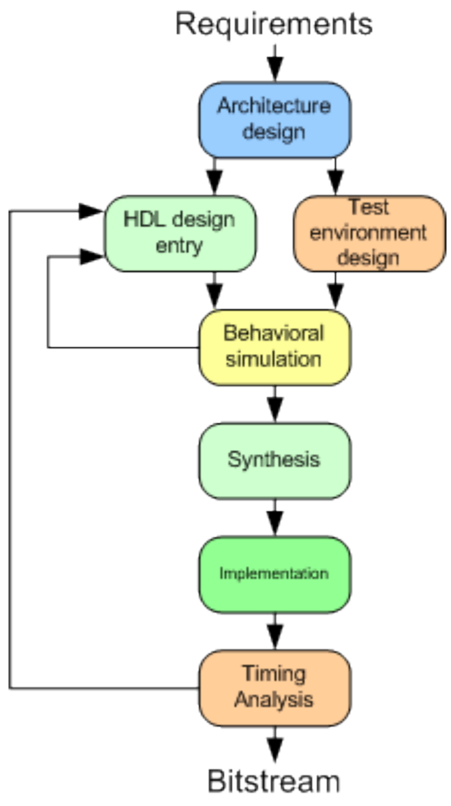


Figure 2-6 FPGA design flow[8]

At the beginning, obviously, is to design the architecture. We need to analyses the project requirements, problem decomposition and some other needs.

Then the real FPGA design begins. There are four main stages: Design entry, simulation, synthesis and implementation.

Design entry describes the device in a formal hardware description language (HDL). The most common HDLs are VHDL and Verilog. Using those languages, hardware can be described at different levels of detail. The most common level used today is Register Transfer Level (RTL). This level describes the functions of the FPGA with logic relations between memory elements.

Then comes the simulation part, which is also an important step. It is to check whether the program is correct by verifying whether the function is operating as intended. If there are differences, the code should be modified until it reaches the requirement.

The third step is synthesis. After the design is correct in simulation, it is time to translate HDL language into FPGA specific building blocks. That is synthesis. Synthesis is performed by a special software called a synthesizer. There may also be some problems and potential errors during the process of synthesis that can't be found in the simulation part and need to be paid more attention.

After the synthesis is successful, the HDL description is converted into netlists. Then the implementation begins. It includes three steps: translate, map and place & route, of which place & route are regarded as the most important ones. After combining all the input netlists and constraints to a logic design file and mapped onto particular device's internal structure, it is time to place and route. It is to allocate FPGA resources, such as logic cells and connection wires. During that process, we also need to check whether the implemented design satisfies timing constraints specified by the user. Then using bitstream generator, these configuration data are written to a bitstream which can be accepted by FPGA.

2.3 SOPC

SOPC is the core theory of my project. It is a combination of SOC and PLD. So I want to give a introduction of SOC.

2.3.1 SOC

System-on-a-chip or system on chip (SoC or SOC) is the technology of integrating all the components of some electronic systems into one integrated circuit. It includes digital, analog, mixed-signal, and radio-frequency functions on a single chip substrate.[9]

The composition of a SoC includes the following components: one microcontroller, microprocessor or DSP core, memory blocks, timing sources, peripherals including counter-timers, real-time timers and power-on reset generators, external and analog interfaces, and Voltage regulators and power management circuits.

In addition, an SOC also includes the software that controls the microcontroller, peripherals and interfaces. The design flow for an SoC, which is shown in Figure 2-7, aims to develop both hardware and software.

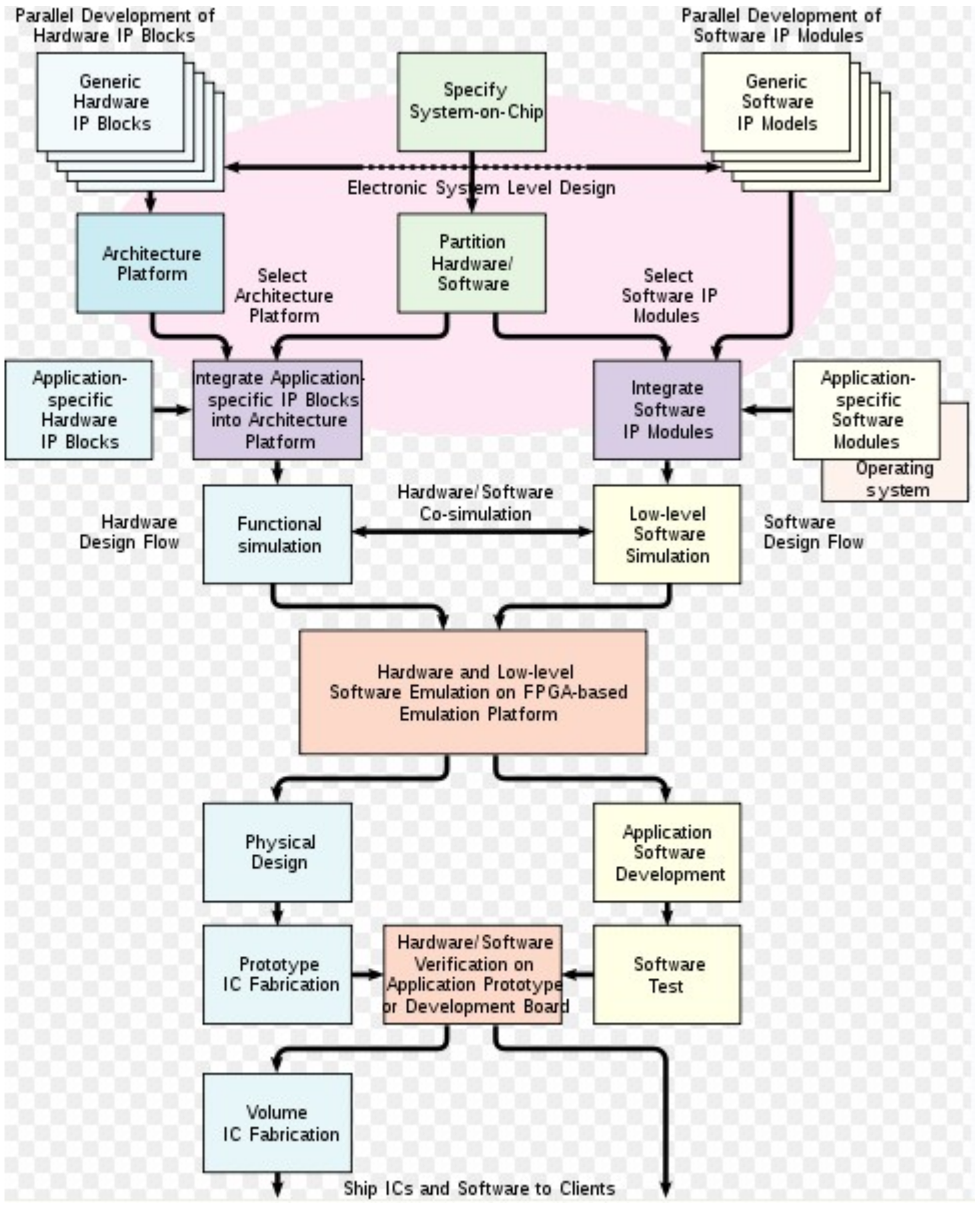

Parallel Development of Hardware IP Blocks
Parallel Development of Software IP Modules
Generic Hardware IP Blocks
Specify System-on-Chip
Generic Software IP Models
Electronic System Level Design
Architecture Platform
Select Architecture Platform
Partition Hardware/ Software
Select Software IP Modules
Application-specific Hardware IP Blocks
Integrate Application-specific IP Blocks into Architecture Platform
Integrate Software IP Modules
Application-specific Software Modules
Operating system
Hardware Design Flow
Functional simulation
Hardware/Software Co-simulation
Low-level Software Simulation
Software Design Flow
Hardware and Low-level Software Emulation on FPGA-based Emulation Platform
Physical Design
Application Software Development
Prototype IC Fabrication
Hardware/Software Verification on Application Prototype or Development Board
Software Test
Volume IC Fabrication
Ship ICs and Software to Clients

Figure 2-7 System-on-a-Chip design flow [9]

2.3.2 SOPC

SOPC, a System-on-a-Programmable-Chip, is a special kind of embedded system: first it is a kind of SOC, which integrates all the components of a computer or other electronic system into a single integrated circuit. Secondly, it is a kind of programmable system, which more flexible and can reach different requests with different design.

SOPC has the following features:

- Contains at least one embedded processor core
- Contains small capacity and high-speed RAM resources
- Plenty of IP Core resources
- Plenty of Programmable resources
- Configuration processor interface and FPGA programming interface
- May contain several programmable circuits
- Single chip and low power consumption[10]

3. Devices

3.0 Overview

Based on the theory declared before, I need some essential devices to complete my own design. In this part I will focus on the hardware and software with which I achieved my project.

3.1 Hardware Part

3.1.1 Cyclone III starter board

The Cyclone III starter board provides a hardware platform that offers a unique opportunity to customize your development environment via expansion connectors and daughter cards as well as evaluate the featurerich, low-power Altera Cyclone III device.

The view of Cyclone III starter board is shown as follows:

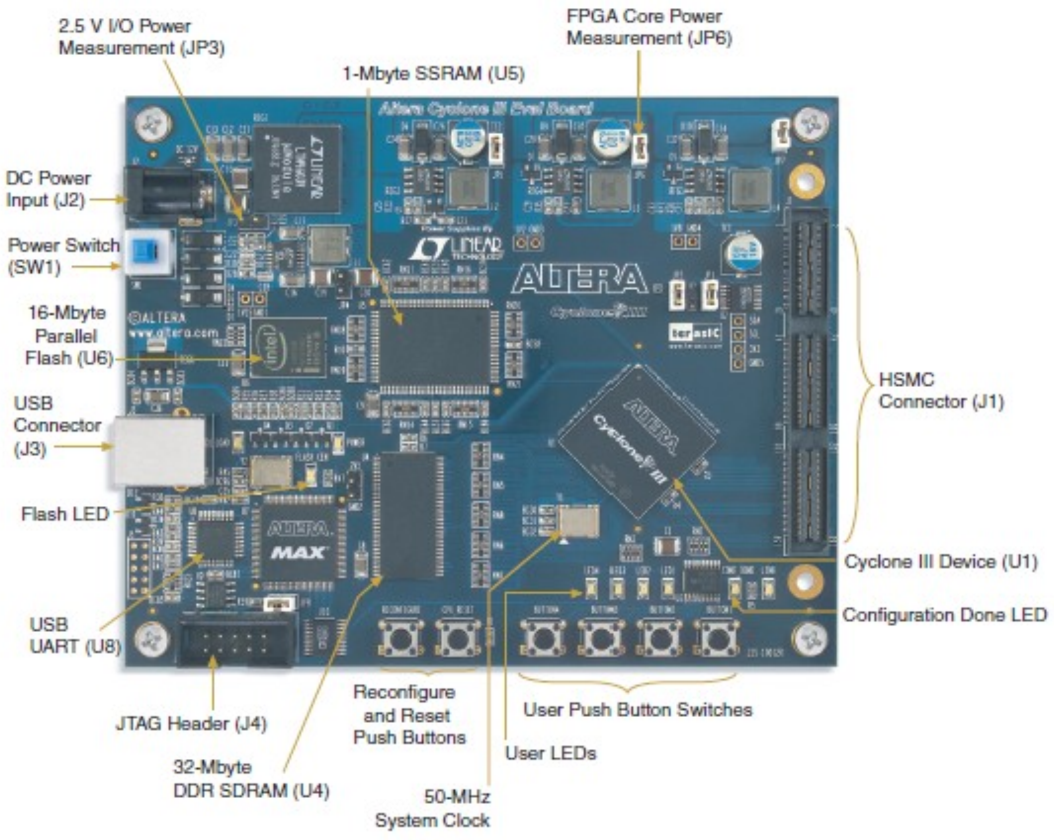


Figure 3-1 Cyclone III starter board [11]

Component Blocks of the board is listed below:

Altera Cyclone III EP3C25F324 FPGA

• 25K logic elements (LEs)

• 66 M9K memory blocks (0.6 Mbits)

• 16 18x18 multiplier blocks

• Four PLLs

• 214 I/Os

Clock management system

• One 50 MHz clock oscillator to support a variety of protocols

• The Cyclone III device distributes the following clocks from its

on-board PLLs:

DDR clock

SSRAM clock

Flash clock

HSMC connector

• Provides 12 V and 3.3 V interface for installed daughter cards

• Provides up to 84 I/O pins for communicating with HSMC

daughter cards

General user-interface

• Four user LEDs

• Two board-specific LEDs

• Push-buttons:

System reset

User reset

Four general user push-buttons

Memory subsystem

- Synchronous SRAM device

  1-Mbyte standard synchronous SRAM
  167-MHz
  Shares bus with parallel flash device

- Parallel flash device

  16-Mbyte device for active parallel configuration and
  storage
  Shares bus with SRAM device

- DDR SDRAM device

  56-pin, 32-Mbyte DDR SDRAM
  167-MHz
  Connected to FPGA via dedicated 16-bit bus

Built-in USB-Blaster interface

- Using the Altera EPM3128A CPLD
- For external configuration of Cyclone III device
- For system debugging using the SignalTap® and Nios® debugging console
- Communications port for Board Diagnostic graphical user interface (GUI) [11]

3.1.2 LCD

The LCD Multimedia Daughtercard was created to provide a set of interfaces including LCD touchscreen, VGA out, composite video in, audio in/out, microphone in, plus Ethernet, SD-Card, PS/2, and RS-232 interfaces.[12]

The following figure shows the top view of the LCD Multimedia Daughtercard

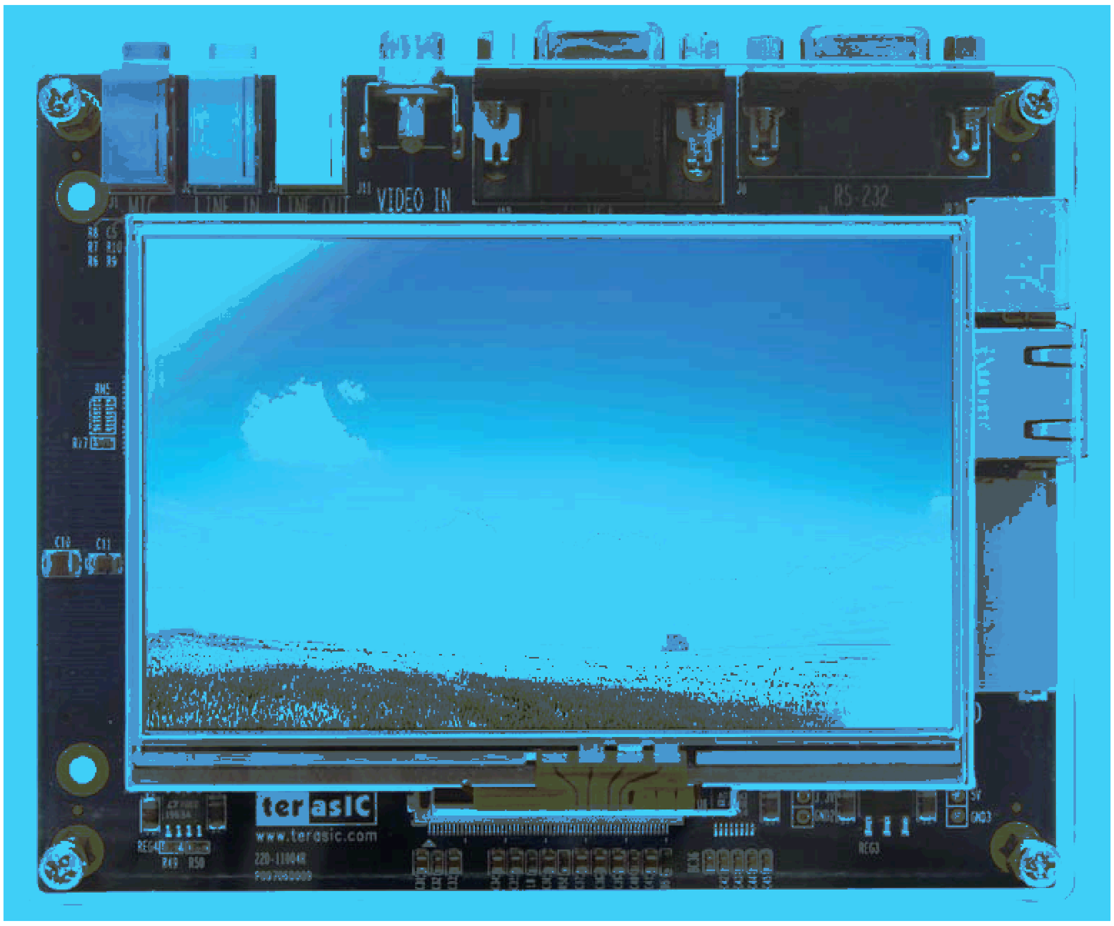


Figure 3-2 Top view of the LCD Multimedia Daughtercard [12]

The LCD Multimedia Daughtercard contains the following components.

MAX II CPLD EPM2210F324

- 2210 Logic elements
- 272 User I/Os
- 324 pin FineLine BGA package

LCD Touch-screen Display

- 800 X 480 pixel 4.3" Display

24-bit Audio Codec

SD Flash Connector

10/100 Ethernet physical layer (PHY)

PS/2 Connector

Other Interfaces

- RS-232 Level-shifters
- RCA Jack (Video In)
- 10-bit VGA Output DAC
- Composite Video ADC [12]

3.2 Software Part

3.2.1 Quartus II Software

The Altera Quartus II design software is a comprehensive environment for system-on-a-programmable-chip (SOPC) design. It provides a complete design environment. By using this, your specific design needs can be easily contented. The Quartus II software includes solutions for all phases of FPGA and CPLD design.[13]

The design flow is shown as follows:

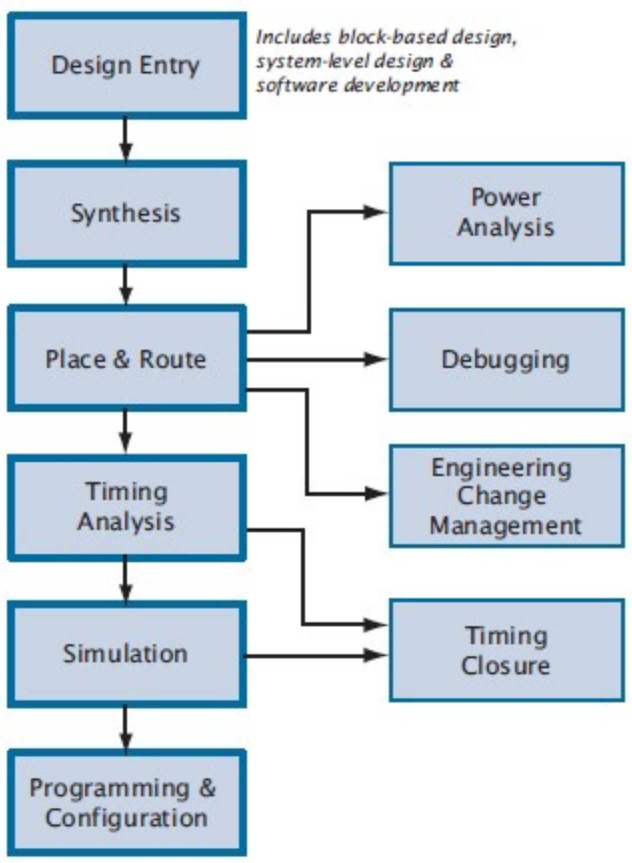

Design Entry
Includes block-based design, system-level design & software development
Synthesis
Place & Route
Timing Analysis
Simulation
Programming & Configuration
Power Analysis
Debugging
Engineering Change Management
Timing Closure

Figure 3-3 Quartus II design flow[13]

3.2.2 NIOS II 7.2 IDE

The Nios II integrated development environment (IDE) is the most important graphical software development tool for the Nios II embedded processors. The Nios II IDE development platform can works for all Nios II processor systems. Nios II IDE can accomplish all the software development tasks, like editing, building, debugging, and profiling programs.

There are two typical design flows involving the Nios II IDE. One is to work entirely within the IDE, the other is to work with the Nios II software build tools first, then import it into the IDE. Depending on your own design, not all steps are required. You can return to a previous step at any point in the process.

The IDE-only design flow includes the following steps:

- Create a project
- Configure the project properties
- Edit the C/C++ application code
- Build the C/C++ application project
- Run and debug the project
- Profile execution performance
- Store the project firmware on a target board

This design flow is mostly used by those who require only a limited control over the building process.

The Nios II software build tools design flow includes the following steps:

- Import the software build tools projects
- These projects are configured prior to import
- Edit the C/C++ application code
- Build the C/C++ application project
- Run and debug the project
- Profile execution performance
- Store the project firmware on a target board[14]

This design flow is mostly used for those who don't want the IDE to manage the makefiles.

4. Program design

4.0 Overview

This part is the main part of my project. Here I will introduce my design ideas and the way to achieve them. It includes the functions that the software provides and my own programs.

4.1 Application Description

My work was to design a Chinese Calendar System. After inputing a year, the output of this system will be the corresponding name of that year according to the traditional Chinese way of counting years. It will also display the sign animal of that year.

In ancient China, people use a different kind of method to count years, which is known as Stems-and-Branches. It is a cyclic numeral system of 60 combinations of the two basic cycles, the ten Heavenly Stems and the twelve Earthly Branches. The ten Heavenly Stems include Jia, Yi, Bing, Ding, Wu, Ji, Geng, Xin, Ren, Gui. The twelve Earthly Branches include Zi, Chou, Yin, Mao, Chen, Si, Wu, Wei, Shen, You, Xu, Hai. Each branch has a sign animal, which are: Rat, Ox, Tiger, Hare, Dragon, Snake, Horse, Sheep, Monkey, Cock, Dog, Boar.

This application consists of two main parts to get it work. The first one is the input function. This part completes the task that after inputing the year, it can be displayed on the screen and stored as a number in the program. The second one is the year count function. This part is to convert the year by the Chinese method of counting years, then display the name and sign animal of that year.

The whole system also consist of two parts, the entry system part and the main system part. The flow of those two parts are shown below:

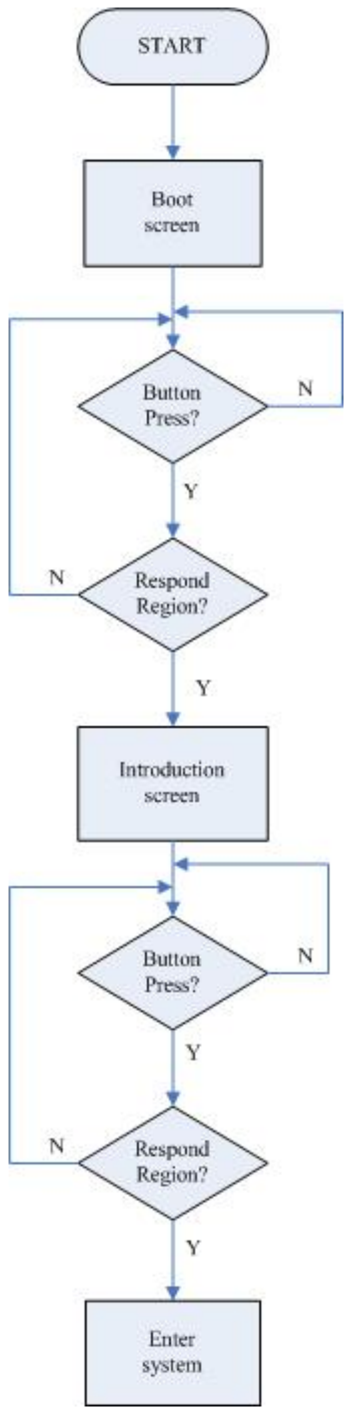


Figure 4-1 Flow of entry part

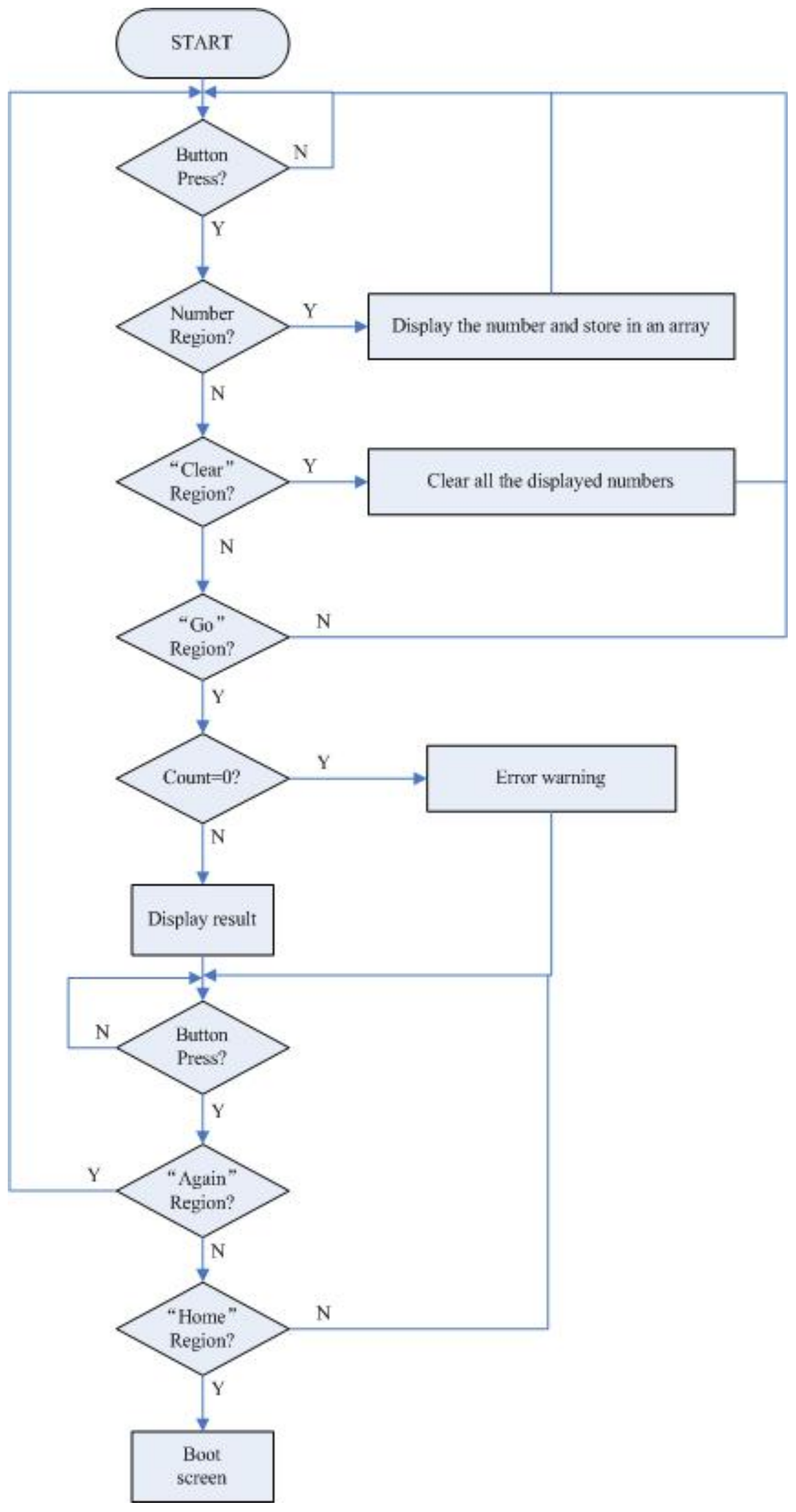


Figure 4-2 Flow of main system

The main program is designed using C language and the NIOS II Embedded Design Suite is used to implement it.

4.2 Header Files

Header Files is an important part of program design. They include different functions that can be called in the program by declaring them before the main program. In the NIOS II IDE system and C language, there are many header files that are useful in my program. The following list shows the header files included in my application:

```
#include <ctype.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <io.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/alt_cache.h>
#include <malloc.h>
#include <priv/alt_file.h>
#include <sys/alt_irq.h>
#include <altera_avalon_sgdma.h>
#include <altera_avalon_sgdma_descriptor.h>
#include <altera_avalon_sgdma_regs.h>
#include "alt_touchscreen.h"
#include "altera_avalon_spi_regs.h"

#include "alt_irq.h"
#include "system.h"
```

```
#include "altera_avalon_pio_regs.h"
#include "alt_tpo_lcd.h"
#include "alt_video_display.h"
#include "simple_graphics.h"
#include "alt_alarm.h"
#include "alt_cache.h"
#include "system.h"
#include "alt_types.h"
#include "control.h"
#include "alteraC3.h"
```

Those header files include system control functions, drivers of LCD screen, touch screen functions and some basic graphics drawing functions. All of these functions are significant for completing my application.

4.3 Useful functions

Some of the functions in the software are frequently employed in my program and I will introduce them briefly here.

1). alt_touchscreen_get_pen

Because this project is based on touch screen hardware, the most essential function is to get pen down information to discern the touched position and execute the corresponding reaction. This function returns three values, pen_down indicates if the pen is down and x,y demonstrate the location of the pen.

```
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;
}
```

2). vid_draw_line

This function is a basic one to draw on the screen. Some other graphics may call this function to be drawn. The horiz_start and vert_start declare the start point position of the line and the horiz_end and vert_end declare the end point position of the line. The width and color obviously show the just mean of the name of these variables.

```
__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 )
   {
//      c2h_draw_horiz_line( (unsigned short)horiz_start,
//                                   (unsigned short)horiz_end,
//                                   (unsigned short)vert_start,
//                                   color,
//
display->buffer_ptrs[display->buffer_being_written]->buffer);

      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 );
   }
}
```

```
void vid_scroll_string_quit(vid_text_scroll_struct* scroll)
{
    free(scroll->string);
    free(scroll);
}

vid_text_scroll_struct* vid_scroll_string_init(int hbegin, int vbegin, int hend, int f_color,
int b_color, char* font, int ms_delay, char *string)
{
    vid_text_scroll_struct* scroll;
    scroll = malloc(sizeof (vid_text_scroll_struct));

    scroll->hbegin = hbegin;
    scroll->vbegin = vbegin;
    scroll->hend = hend;
    scroll->f_color = f_color;
    scroll->b_color = b_color;
    scroll->string = malloc(strlen(string)+2);
    strcpy(scroll->string, string);
    scroll->font = font;
    scroll->ms_delay = ms_delay;
    scroll->ticks_at_last_move = alt_nticks();
    scroll->text_scroll_index = 0;
    scroll->text_scroll_started = 0;
    scroll->window_width = scroll->hend - scroll->hbegin;
    scroll->length_of_string = strlen(string);
    scroll->string_points = scroll->length_of_string * 8;
    scroll->scroll_points = (scroll->window_width + scroll->string_points);
```

```
  return(scroll);
}

int vid_scroll_string(vid_text_scroll_struct* scroll, alt_video_display* display)
{
 int x_start, x_end, x_index, string_x_index, string_char_index, char_row, char_column;
  char character, column_mask;
  char* font_char_ptr;
  char pixels_to_move_by = 1;

  // If it's time to move the scroll..
  if (alt_nticks() >= (scroll->ticks_at_last_move + ((alt_ticks_per_second() *
(scroll->ms_delay)) / 1000))) {
    scroll->ticks_at_last_move = alt_nticks();

    // Track where we are in the scroll.
    if(scroll->text_scroll_started == 0) {
      scroll->text_scroll_index = 0;
      scroll->text_scroll_started = 1;
    } else if(scroll->text_scroll_index >= scroll->scroll_points)  {
      scroll->text_scroll_started = 0;
    } else {
      scroll->text_scroll_index += pixels_to_move_by;
    }

    //Find out where we start
    if (scroll->text_scroll_index < scroll->window_width) {
      x_start = scroll->hbegin + scroll->window_width - scroll->text_scroll_index;
    } else {
```

```
    x_start = scroll->hbegin;
  }
  //Find out where we end
  if (scroll->string_points > scroll->text_scroll_index) {
    x_end = scroll->hend;
  } else {
    x_end = (scroll->hend - scroll->text_scroll_index + scroll->string_points);
  }

  // Write the string segment a column (x) at a time
  for(x_index = x_start; x_index < x_end; x_index++) {
    // Find the x index we're at within the string
    // If first part of string hasnt yet reached left side of scroll window
    if (scroll->text_scroll_index < scroll->window_width) {
      string_x_index = (x_index - x_start);
    } else {
      string_x_index = scroll->text_scroll_index - scroll->window_width + x_index -
x_start;
    }
    //Find the character we're supposed to be writing
    string_char_index = (string_x_index / 8);
    character = scroll->string[string_char_index];
    char_column = (string_x_index % 8);
    column_mask = (((unsigned int)0x80) >> char_column);
    font_char_ptr = (scroll->font + ((character - 0x20) * FONT_10PT_ROW));
    //We have all the data now, so let's write a column
    for(char_row = 0; char_row < 11; char_row++) {
      // If the font table says this pixel is on, then set it to the foreground color
      if (*(font_char_ptr + char_row) & column_mask) {
        vid_set_pixel(x_index, scroll->vbegin + char_row, scroll->f_color, display);
```

```
            // Otherwise, set it to the background color.
            } else {
               vid_set_pixel(x_index, scroll->vbegin + char_row, scroll->b_color, display);
//background color
            }
         }
      }
      // Erase the leftover column (x) of the last string we wrote.
      vid_draw_line(x_end, scroll->vbegin, x_end, scroll->vbegin + 10, 1, scroll->b_color,
display);
      // Log what time we moved the scroll.
   }
   return(0);
}
```

3). vid_draw_box

This function was used the most in my program. It is to draw a box on the screen at the assigned position. The horiz_start and vert_start should be the position of the left-top corner of the box. The horiz_end and vert_end should be the position of the right-bottom corner of the box. The color, of course, is the color of the box. If the value of fill is 1, the box will be filled in that color you assigned before.

```
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);
}
```

4). vid_draw_circle

This function is used for drawing a circle on the screen. Hcenter and Vcenter should be assigned the position of the center of that circle. The value of radius indicates how big the circle will be. The variables color and fill are just like the function of vid_draw_box.

```
int vid_draw_circle(int Hcenter, int Vcenter, int radius, int color, char fill,
alt_video_display* display)
{
  int x = 0;
  int y = radius;
  int p = (5 - radius*4)/4;

  // Start the circle with the top, bottom, left, and right pixels.
  vid_circle_points(Hcenter, Vcenter, x, y, color, fill, display);

  // Now start moving out from those points until the lines meet
  while (x < y) {
    x++;
    if (p < 0) {
        p += 2*x+1;
      } else {
        y--;
        p += 2*(x-y)+1;
      }
      vid_circle_points(Hcenter, Vcenter, x, y, color, fill, display);
  }
  return (0);
}
```

5). vid_print_string

This function is to print string on the screen. Horiz_offset and vert_offset indicate the start position of that string. Color and font are the attributes of the string. At last, the content of the string should be listed in quotation marks.

```
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);
}
```

4.4 My own functions

Based on those functions that were provided by the library, I also established some small functions which will be constantly called in the main program.

1). Delay function

Because the system can implement instructions rapidly, delay function is necessary in a project to control the reaction time. In my program, I use the loop function, completed by "for" statement, to implement the delay function.

```
 void delay(int time)
{
       int i,j,c;
       for(i=0;i<=time;i++)
       {
             for(j=0;j<=2000;j++)
             {
                  c=1 ;
```

```
        }
    }
}
```

2) Reset pen function

After first called the alt_touchscreen_get_pen function, the three values will not be changed. So I need to establish a function to empty the pointer and reset the values if the screen is touched again.

In the library there is a function that can initialize all the members and set it to update the pen down information. The statements are as follows:

```
while(1) {
    OSTimeDlyHMSM(0, 0, 0, (1000 / TOUCHSCREEN_SAMPLE_RATE));
    alt_touchscreen_event_loop_update(screen);
}
```

So I set a downcheck function which completes the task initializing the pointer and resets the new values of pen down information:

```
void downcheck( alt_touchscreen_scaled_pen_data pen_data,
                alt_video_display* display,
                alt_touchscreen* screen)
{
 while(1) {
    OSTimeDlyHMSM(0, 0, 0, (1000 / TOUCHSCREEN_SAMPLE_RATE));
    alt_touchscreen_event_loop_update(screen);
    alt_touchscreen_get_pen(screen,     (&pen_data.pen_down),     (&pen_data.x),
(&pen_data.y));
```

```
        if(pen_data.pen_down == 1)
        break;
                        }
}
```

4.5 The main design of my program

Based on those basic functions, I could begin my own project. The flow of my design has already been demonstrated before, so this part I will focus on the program that achieve them.

4.5.1 The boot screen

When starting the system, the boot screen should come first. Its appearance is shown below:

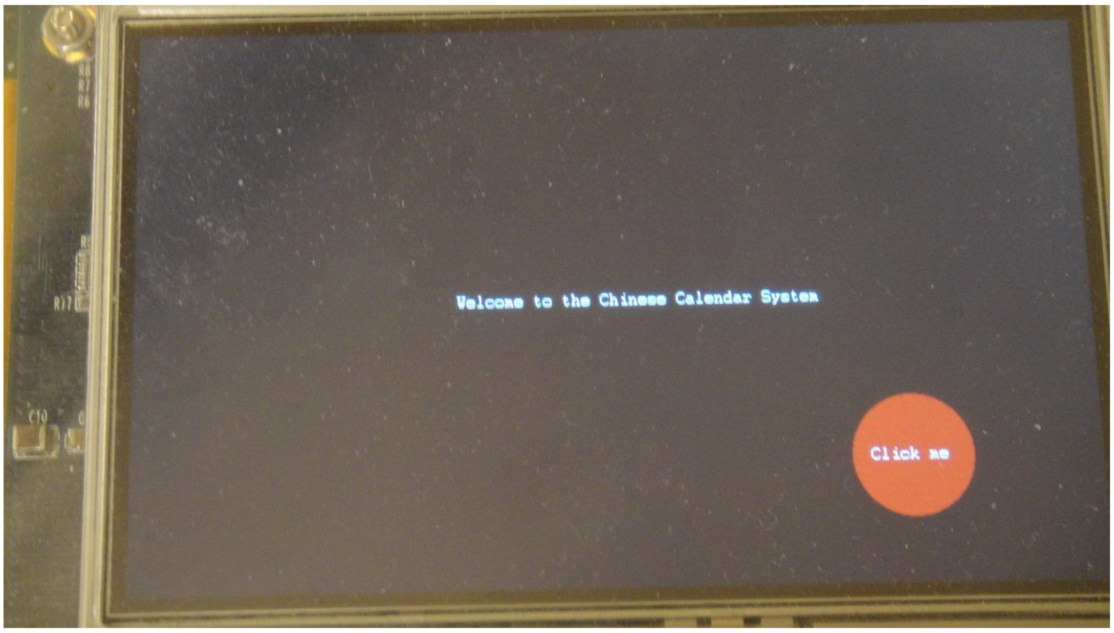


Figure 4-3 boot screen

The following program draws the appearance of the boot screen:

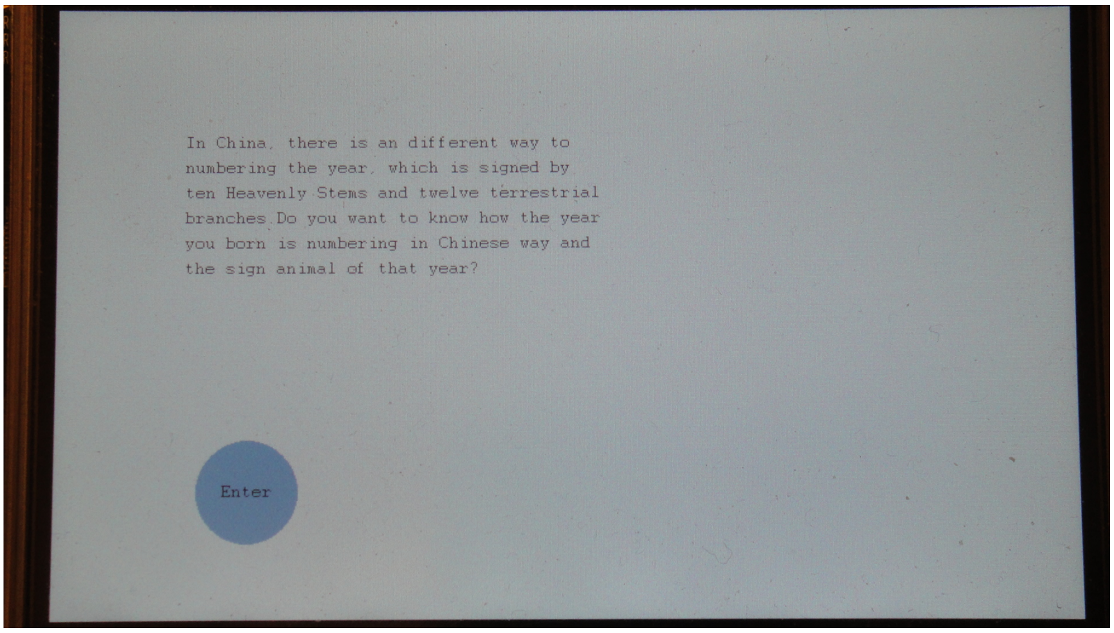
```
vid_draw_box (0, 0, 799, 479, 0x000000, 1, display);
        vid_print_string(270, 235, 0xFFFFFF, cour10_font, display, "Welcome to the Chinese Calendar System");
        vid_draw_circle(645, 375, 50, 0xAE0000, 1, display);
        vid_print_string(610, 370, 0xFFFFF0, cour10_font, display, "Click me");
```

The box drawn at the bottom right-hand corner is designed as the respond region to begin that counting system. After clicking the key, it will enter the Chinese Calendar System and then we can see the introduction interface, which is achieved by the following program:

```
while(1){
            alt_touchscreen_get_pen(screen, (&pen_data.pen_down), (&pen_data.x), (&pen_data.y));

  if(pen_data.x>=600&&pen_data.x<=690&&pen_data.y>=350&&pen_data.y<=400)
            {
             vid_draw_box (0, 0, 799, 479, 0xFFFFF0, 1, display);
             vid_print_string(100, 100, 0x000000, cour10_font, display, "In China, there is an different way to ");
             vid_print_string(100, 120, 0x000000, cour10_font, display, "numbering the year, which is signed by");
             vid_print_string(100, 140, 0x000000, cour10_font, display, "ten Heavenly Stems and twelve terrestrial");
             vid_print_string(100, 160, 0x000000, cour10_font, display, "branches.Do you want to know how the year");
             vid_print_string(100, 180, 0x000000, cour10_font, display, "you born is numbering in Chinese way and");
             vid_print_string(100, 200, 0x000000, cour10_font, display, "the sign animal of
that year?");
             vid_draw_circle(150, 380, 40, 0x9AC0CD, 1, display);
             vid_print_string(130, 375, 0x000000, cour10_font, display, "Enter");
             break;
         }
     }
```

The "if" statement is used to check the touch screen information. If the pen is down and just in the respond region, this interface will be displayed. It is shown below:



In China, there is an different way to
numbering the year, which is signed by
ten Heavenly Stems and twelve terrestrial
branches.Do you want to know how the year
you born is numbering in Chinese way and
the sign animal of that year?

Figure 4-4 Introduction screen

On the screen, there is also a button. After being clicked, it will enter the core part of the system, which is the year input and converse part.

```
  while(1){
        alt_touchscreen_get_pen(screen,      (&pen_data.pen_down),      (&pen_data.x),
(&pen_data.y));



if(pen_data.x>=100&&pen_data.x<=190&&pen_data.y>=350&&pen_data.y<=400)
            {
            systemdisply(pen_data, display, screen);
            break;
            }
             }
```

We can see that the function systemdisplay() is called to enter the core system.

4.5.2 The core system

4.5.2.1 System display interface

The main interface of this system is shown below:

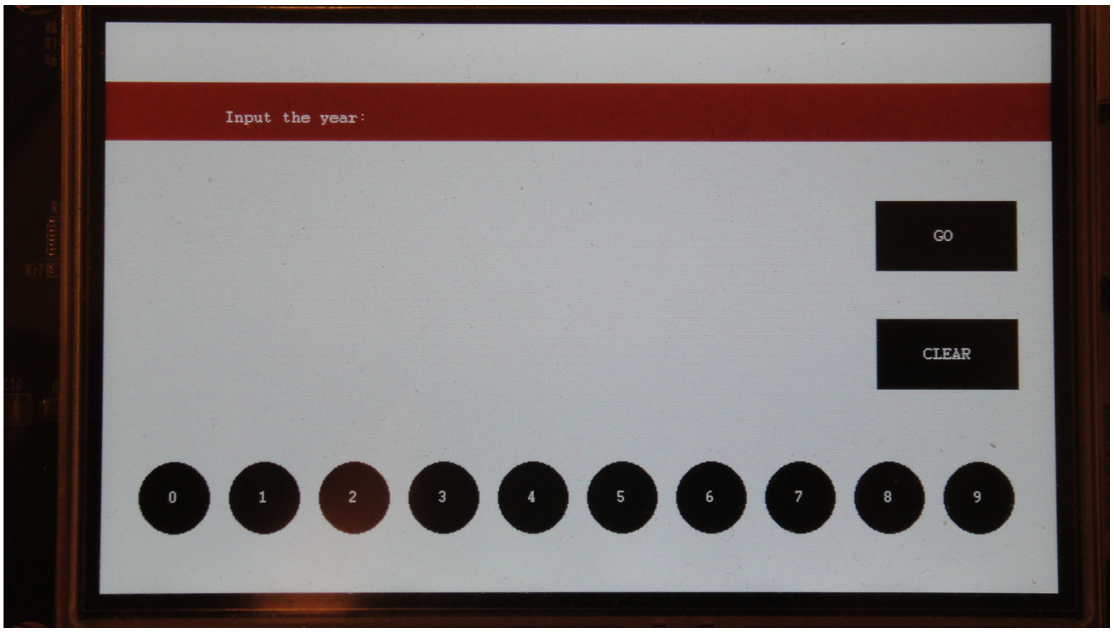


Figure 4-5 System main screen

I set up a specialized function to display the system input screen. In this function, it is easier to display the ten numbers using "for" loop statement. The whole program is as follows:

```
void systemdisply(alt_touchscreen_scaled_pen_data pen_data,
                        alt_video_display* display,
                        alt_touchscreen* screen)
{
  int i,center=60;
  char numbers[10];
  vid_draw_box (0, 0, 799, 479, 0xFFFFF0, 1, display);
  vid_draw_box (0, 50, 799, 100, 0xAE0000, 1, display);
  vid_print_string(100, 75, 0xFFFFF0, cour10_font, display, "Input the year:");
for(i=0;i<10;i++)
{
vid_draw_circle(center, 400, 30, 0x000000, 1, display);
sprintf(numbers, "%d", i);
vid_print_string(center-5, 395, 0xFFFFF0, cour10_font, display, numbers);
center=center+75;
}
vid_draw_box (650, 150, 770, 210, 0x000000, 1, display);
vid_draw_box (650, 250, 770, 310, 0x000000, 1, display);
vid_print_string(700, 175, 0xFFFFF0, cour10_font, display, "GO");
vid_print_string(690, 275, 0xFFFFF0, cour10_font, display, "CLEAR");
while(1) {
      OSTimeDlyHMSM(0, 0, 0, (1000 / TOUCHSCREEN_SAMPLE_RATE));
      alt_touchscreen_event_loop_update(screen);
      alt_touchscreen_get_pen(screen,         (&pen_data.pen_down),         (&pen_data.x),
(&pen_data.y));
```

```
      if(pen_data.pen_down == 1)
      break;
   }
  inputcontrol(pen_data, display, screen);
}
```

On this screen, we can do three kinds of things: input the year, clear the input numbers and see the result. The “if” statement was used for checking which one should be executed. The clear function is to initialize the system. The check result function leads to the final result shown interface. The year input function can display the number one has just input and transmit that year to another function to convert it according to the Chinese traditional calendar.

In this function, I used vid_draw_box and vid_draw_circle to draw the region of those buttons and vid_print_string function is to print string on the button to indicate the functions of those buttons. Sprintf function is used to print the just number on screen, cooperating with the vid_print_string function and an array to store numbers.

4.5.2.2 Year input and control system

When the screen is touched, if it is in the respond region, there will be three conditions that have already been described before. So in this part the function is to discern what region is touched and the reaction it will have.

The program for controlling the reaction of this system is shown below:

```
int i,xposi,displayposi=30,count=0,theyear,a,b;
    int year[4]={0,0,0,0};
    char numbers[10];
```

```
CHECK:
          xposi=30;
while(1)
{
alt_touchscreen_get_pen(screen, (&pen_data.pen_down), (&pen_data.x), (&pen_data.y));
if(pen_data.x>=650&&pen_data.x<=770&&pen_data.y>=250&&pen_data.y<=310)
               {
                 systemdisply(pen_data, display, screen);
                 break;
               }
            else
if(pen_data.x>=650&&pen_data.x<=770&&pen_data.y>=150&&pen_data.y<=210)
               {
                 break;
               }
            if(count>=4)
              goto END;
            else
if(pen_data.x>=0&&pen_data.x<=799&&pen_data.y>=370&&pen_data.y<=430)
            {
                 for(i=0;i<10;i++)
                 {

if(pen_data.x>=xposi&&pen_data.x<=(xposi+60)&&pen_data.y>=370&&pen_data.y<=43
0)
                    {
                    sprintf(numbers, "%d", i);
                    vid_print_string(displayposi, 150, 0x000000, cour10_font, display,
numbers);
                    year[count]=i;
```

```
                count++;
                delay(1000);
                displayposi=displayposi+10;
                }
                xposi=xposi+75;
              }
               downcheck(pen_data, display, screen);
 END:                 goto CHECK;
            }
      }
```

If the pen is located in the “clear” box, it will recall the system display function to initialize the system and clean the numbers that have been already input.

If the pen is located in the “go” box, it will break the **while(1)** loop and can directly lead to the result display part.

If it does not belong to either of the above situations, then check whether the pen is down in the number input regions. If yes, the respond should be the year input part.

In this part, I also used the loop function to display numbers on the screen. At the same time, I defined two variables for the next counting part. One integer “count” to count how many numbers have been input and one array to store the input year.

The “for” loop is used for checking if the pen is down in the number region and which number it is from 1 to 0. As long as the pen_down value is checked to be true, the system displays the number according to the times this cycle has been performed. Besides, the variable “count” will add one and the number will be stored into the array. Then I called the initializing pen function to prepare the next touch of screen and respond for the right performances.

The amount of the input number is restricted to be four or less than four, so that it can represent a year. If the value of variable "count" equals four, the system will stop executing the input function and just wait for the two other situations.

4.5.2.3 Year counting

After input the year, I set up a function to compose the numbers into a whole number so that it can stand for a year and can be converted into the Chinese expression. The program of counting function is shown as follows:

```
int countyear(int count, int year[4])
{
    int sum=0, product=1,i,j;
   for(i=0;i<4;i++)
    {
         for(j=1;j<count;j++)
         {
             product=product*10;
         }
        count--;
        sum=sum+product*year[i];
        product=1;
    }
    return sum;
}
```

After defining this function, I can call it in the result display part so that it can convert scattered numbers into a whole year which could take part in the latter counting.

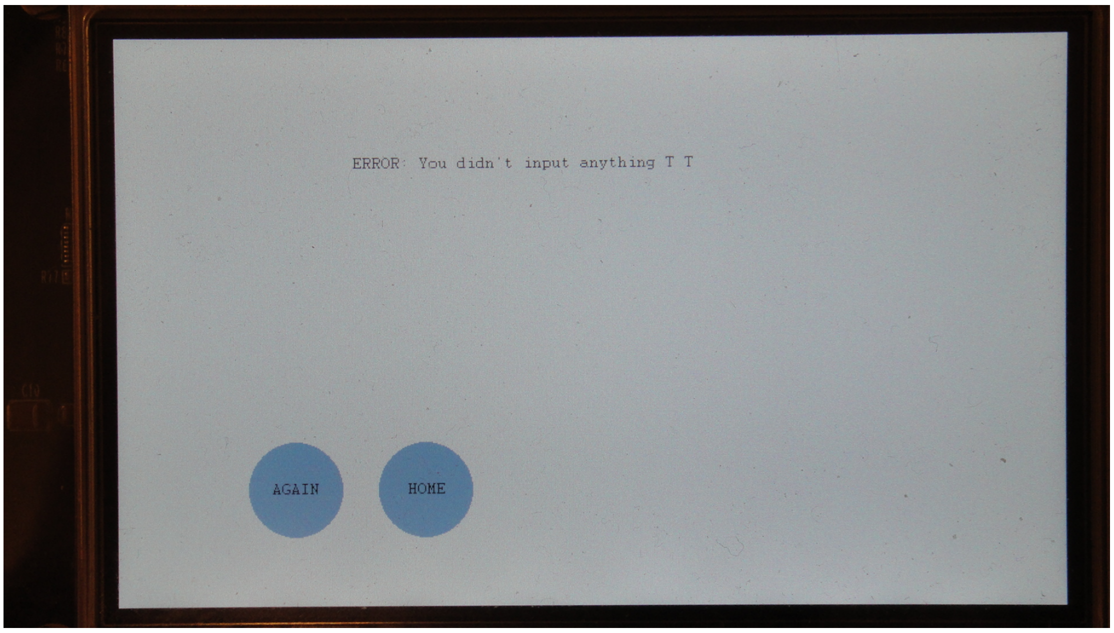
4.5.2.4 Displaying result

As I introduced before, in China, years are represented by two small cycles, the ten Heavenly Stems and the twelve Earthly Branches. So I just used the modulus divided by ten and twelve respectively to judge which Heavenly Stems and Earthly Branches it belongs to. If we know the Earthly Branch of that year, the sign animal will be naturally known. So I just use the “switch” statement to achieve the result display part.

```
vid_draw_box (0, 0, 799, 479, 0xFFFFF0, 1, display);
vid_draw_circle(150, 380, 40, 0x9AC0CD, 1, display);
vid_print_string(130, 375, 0x000000, cour10_font, display, "AGAIN");
vid_draw_circle(260, 380, 40, 0x9AC0CD, 1, display);
vid_print_string(245, 375, 0x000000, cour10_font, display, "HOME");
if(count==0)
{
vid_print_string(200, 100, 0x000000, cour10_font, display, "ERROR: You didn't input anything T T");
}
else
{
vid_print_string(200, 100, 0x000000, cour10_font, display, "In ancient China, this year is called:");
vid_print_string(200, 200, 0x000000, cour10_font, display, "The sign animal of this year is:");
theyear=countyear(count,year);
a=theyear%10;
switch(a)
{
        case 0: vid_print_string(530, 100, 0xf30400, cour10_font, display, "GENG");break;
        case 1: vid_print_string(530, 100, 0xf30400, cour10_font, display, "XIN");break;
        case 2: vid_print_string(530, 100, 0xf30400, cour10_font, display, "REN");break;
        case 3: vid_print_string(530, 100, 0xf30400, cour10_font, display, "GUI");break;
        case 4: vid_print_string(530, 100, 0xf30400, cour10_font, display, "JIA");break;
        case 5: vid_print_string(530, 100, 0xf30400, cour10_font, display, "YI");break;
        case 6: vid_print_string(530, 100, 0xf30400, cour10_font, display, "BING");break;
        case 7: vid_print_string(530, 100, 0xf30400, cour10_font, display, "DING");break;
        case 8: vid_print_string(530, 100, 0xf30400, cour10_font, display, "WU");break;
        case 9: vid_print_string(530, 100, 0xf30400, cour10_font, display, "JI");break;
    }
    b=theyear%12;
    switch(b)
    {
        case 0: vid_print_string(580, 100, 0xf30400, cour10_font, display, "SHEN");
                vid_print_string(530, 200, 0xf30400, cour10_font, display, "Monkey clever");break;
        case 1: vid_print_string(580, 100, 0xf30400, cour10_font, display, "YOU");
                vid_print_string(530, 200, 0xf30400, cour10_font, display, "Rooster deep thinkers");break;
        case 2: vid_print_string(580, 100, 0xf30400, cour10_font, display, "XU");
                vid_print_string(530, 200, 0xf30400, cour10_font, display, "Dog loyalty");break;
        case 3: vid_print_string(580, 100, 0xf30400, cour10_font, display, "HAI");
                vid_print_string(530, 200, 0xf30400, cour10_font, display, "Pig chivalrous");break;
        case 4: vid_print_string(580, 100, 0xf30400, cour10_font, display, "ZI");
                vid_print_string(530, 200, 0xf30400, cour10_font, display, "Rat charm");break;
```

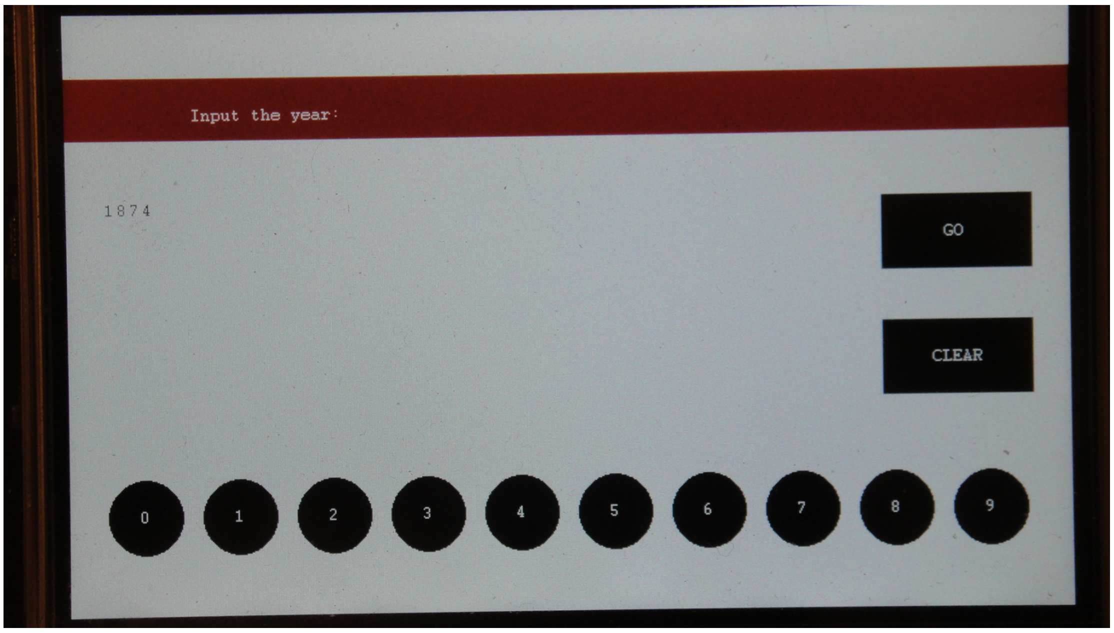
```
          case 5: vid_print_string(580, 100, 0xf30400, cour10_font, display, "CHOU");
                    vid_print_string(530, 200, 0xf30400, cour10_font, display, "Ox
patient");break;
          case 6: vid_print_string(580, 100, 0xf30400, cour10_font, display, "YIN");
                    vid_print_string(530, 200, 0xf30400, cour10_font, display, "Tiger
sensitive");break;
          case 7: vid_print_string(580, 100, 0xf30400, cour10_font, display, "MAO");
                    vid_print_string(530, 200, 0xf30400, cour10_font, display, "Rabbit
articulate");break;
          case 8: vid_print_string(580, 100, 0xf30400, cour10_font, display, "CHEN");
                    vid_print_string(530, 200, 0xf30400, cour10_font, display, "Dragon
healthy");break;
          case 9: vid_print_string(580, 100, 0xf30400, cour10_font, display, "SI");
                    vid_print_string(530, 200, 0xf30400, cour10_font, display, "Snake
deep");break;
          case 10: vid_print_string(580, 100, 0xf30400, cour10_font, display, "WU");
                    vid_print_string(530, 200, 0xf30400, cour10_font, display, "Horse
popular");break;
          case 11: vid_print_string(580, 100, 0xf30400, cour10_font, display, "WEI");
                    vid_print_string(530, 200, 0xf30400, cour10_font, display, "Goat
elegant");break;
      }
      }
```

There are two different situations after clicking the “go” button, distinguished by whether someone has input numbers. So I specified two kinds of results to be displayed. If nothing was input, the system will warn you of the error. If there was a year that has been input, the system will obviously display the result. In the program, those two situations can be judged by the value of “count”. The interfaces shown based on the two situations are respectively like these:



ERROR: You didn't input anything T T

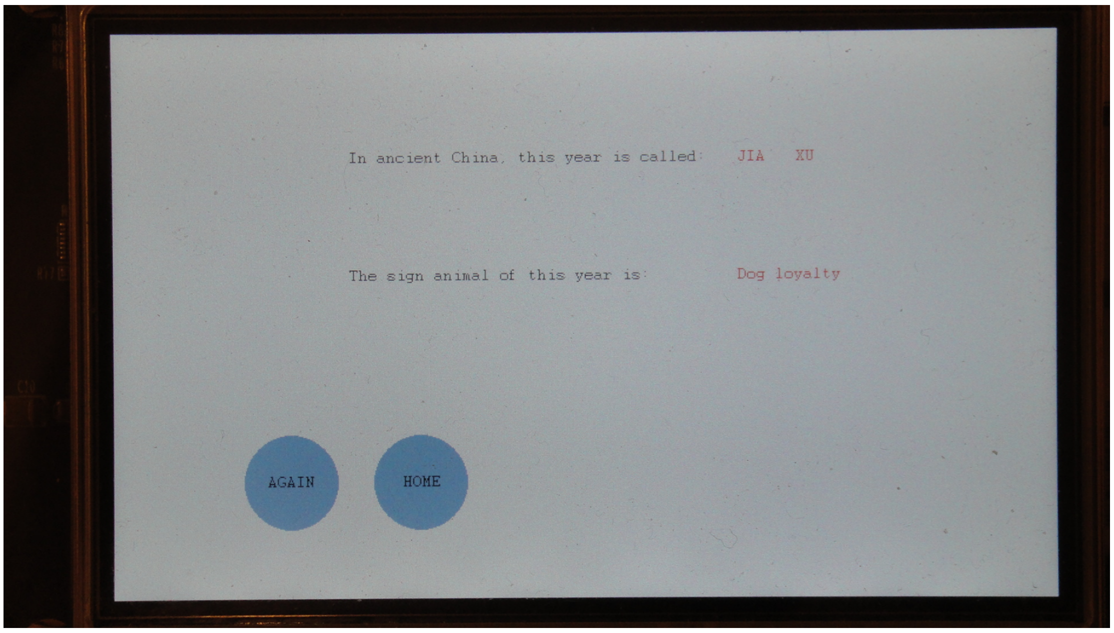
Figure 4-6 Result display when nothing was input



Input the year:

1874

Figure 4-7 Input a year


In ancient China, this year is called: JIA XU

The sign animal of this year is: Dog loyalty

Figure 4-8 Result display

In the result display interface, there are also two buttons, which can lead to the boot screen and the core system. If the button “HOME” is clicked, the function that shows the boot screen will be called in the program. If the button “AGAIN” is clicked, the function to display the core system interface will be called. This kind of function can be achieved by the following statements:

```
while(1)
    {
        alt_touchscreen_get_pen(screen, (&pen_data.pen_down), (&pen_data.x),
(&pen_data.y));

if(pen_data.x>=100&&pen_data.x<=190&&pen_data.y>=350&&pen_data.y<=400)
        {
            systemdisply(pen_data, display, screen);
            break;
        }
        else
if(pen_data.x>=220&&pen_data.x<=300&&pen_data.y>=350&&pen_data.y<=400)
            {
                  home(pen_data, display, screen);
                  break;
            }
      }
```

These are the main parts of the design of my application. To get the work done normally, there are still some other things that need to be paid attention to.

4.5.3 The essential declarations

In my program, I have defined several functions so that they can be called conveniently. In case that they could be in use when called, I need to declare them before the whole program as follows:

```
void downcheck( alt_touchscreen_scaled_pen_data pen_data,
                        alt_video_display* display,
                        alt_touchscreen* screen)

void systemdisply(alt_touchscreen_scaled_pen_data pen_data,
                        alt_video_display* display,
                        alt_touchscreen* screen);

void home(alt_touchscreen_scaled_pen_data pen_data,
                        alt_video_display* display,
                        alt_touchscreen* screen);

void inputcontrol( alt_touchscreen_scaled_pen_data pen_data,
                          alt_video_display* display,
                    alt_touchscreen* screen);

int countyear(int count, int year[4]);
```

In addition, as I wish to get the display and touchscreen functions work well, in the main function, I also need to declare the following things:

```
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);
  int x;

  x = alt_touchscreen_init(
    screen,
    TOUCH_PANEL_SPI_BASE,
    TOUCH_PANEL_SPI_IRQ,
    TOUCH_PANEL_PEN_IRQ_N_BASE,
    50,
    ALT_TOUCHSCREEN_SWAP_XY);

    alt_touchscreen_calibrate_upper_right (screen,
                         3946,    3849,     // ADC readings
```

```
                          799,          0    ); // pixel coords


   alt_touchscreen_calibrate_lower_left    (screen,
                          132,       148,         // ADC readings
                            0,       479    );    // pixel coords


    if(x) {
   printf("Failed.\n");


 } else {
   printf("Success.\n");
 }
```

5. Implementing the program

5.0 overview

The code has to be executed using software. In this project, NIOS II Embedded Design Suite was used to achieve it. As for Download the FPGA configuration file into the Cyclone III board to run the system, Quartus II Web Edition is also needed to compile the project into a .sof file and download it. In this section, I will describe the steps of using the altera software to achieve the project.

5.1 Preparation for the project

To complete the whole program, the first thing should be installing those recommended software. The Altera Complete Design Suite provided all the software needed. After installing the software, another important thing is to install the Nios II Embedded Evaluation Kit, which provided documentation and example applications. It also includes the USB-Blaster Driver.

Then I need to request a license from the Altera website for the Quartus II software. As there was something wrong with getting the license on the website, I downloaded another version, Quartus II version 9.1 Web Edition, which does not need a license

Then I could open the Nios II IDE to start the project.

5.2 Start a new project

After installing the software, I could start my own project. Open the Nios II IDE software and start a new project. Choose File=>New=>NIOS II C/C++ Application as shown below:

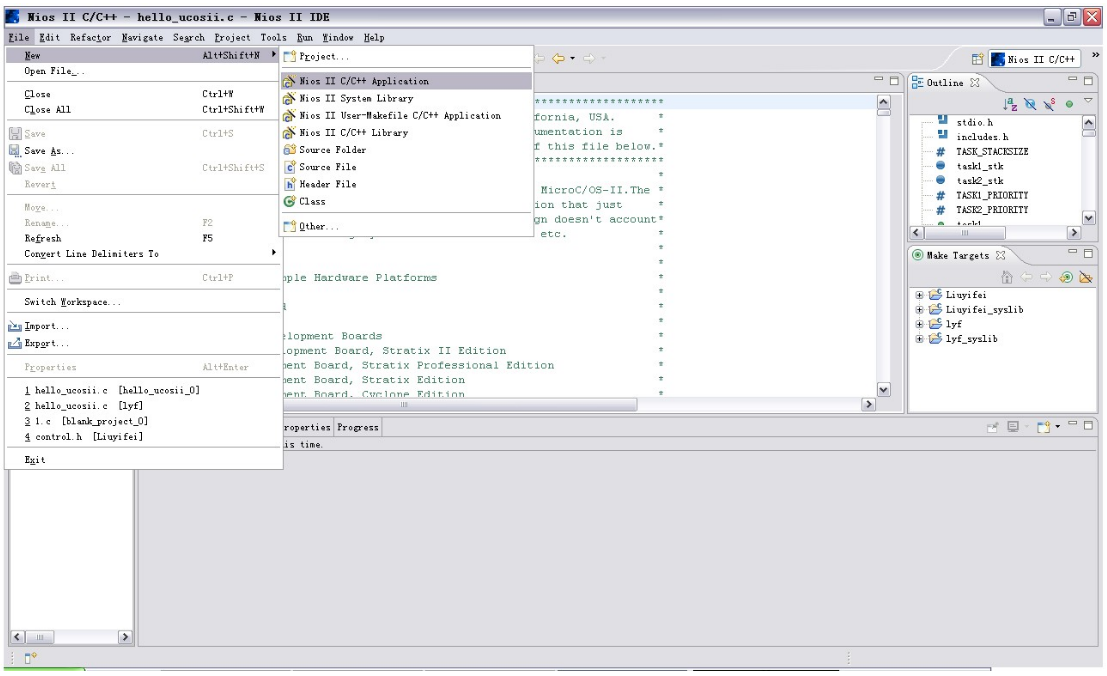
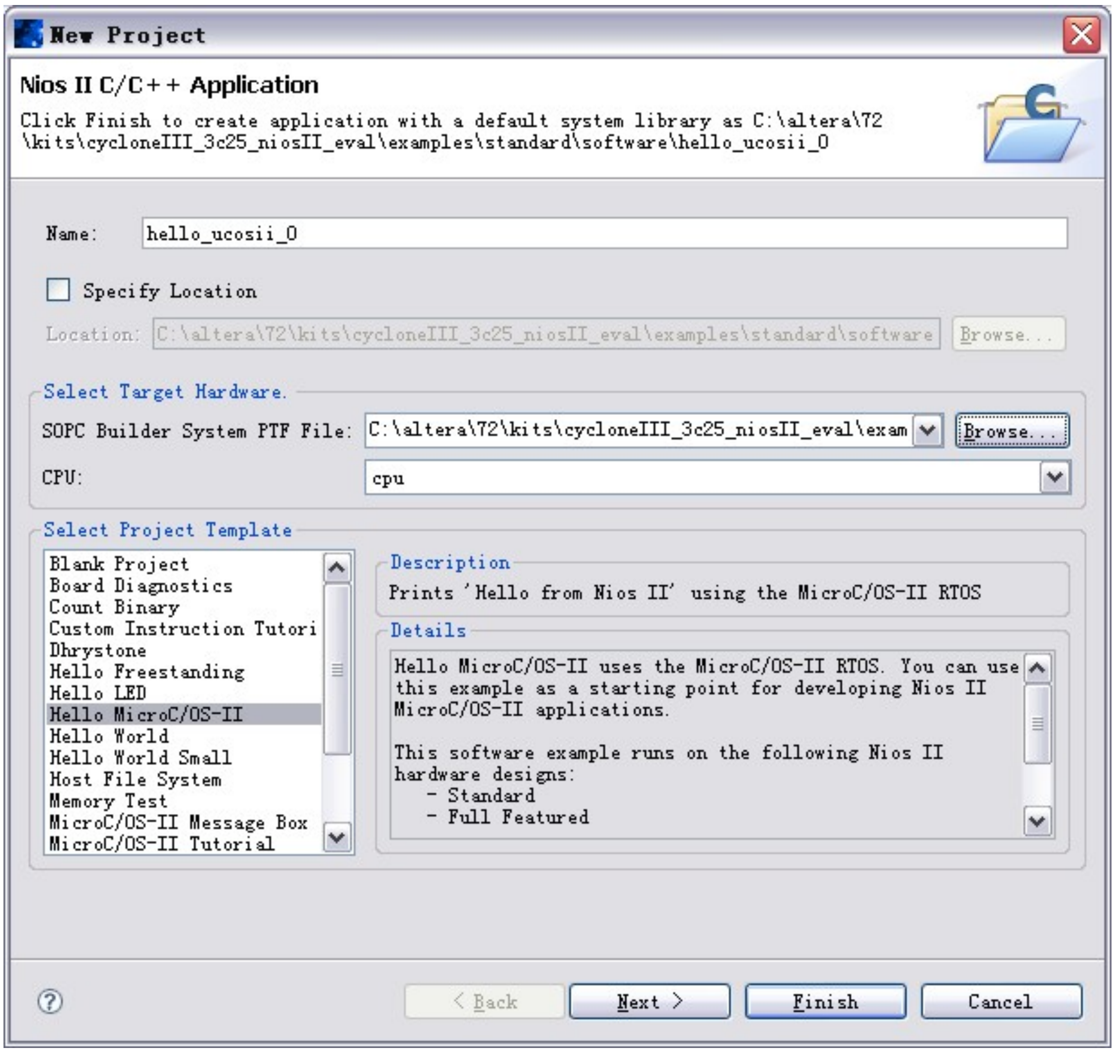


Figure 5-1 Start a new project

Then the new project setting window will open. Type the project name and browse the PTF file in the kit file folder. Then select MicroC/OS-II template for my application.



New Project

Nios II C/C++ Application

Click Finish to create application with a default system library as C:\altera\72\kits\cycloneIII_3c25_niosII_eval\examples\standard\software\hello_ucosii_0

Name: hello_ucosii_0

Specify Location

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

Select Target Hardware.

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

CPU: cpu

Select Project Template

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

Description

Prints 'Hello from Nios II' using the MicroC/OS-II RTOS

Details

Hello MicroC/OS-II uses the MicroC/OS-II RTOS. You can use this example as a starting point for developing Nios II MicroC/OS-II applications.

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

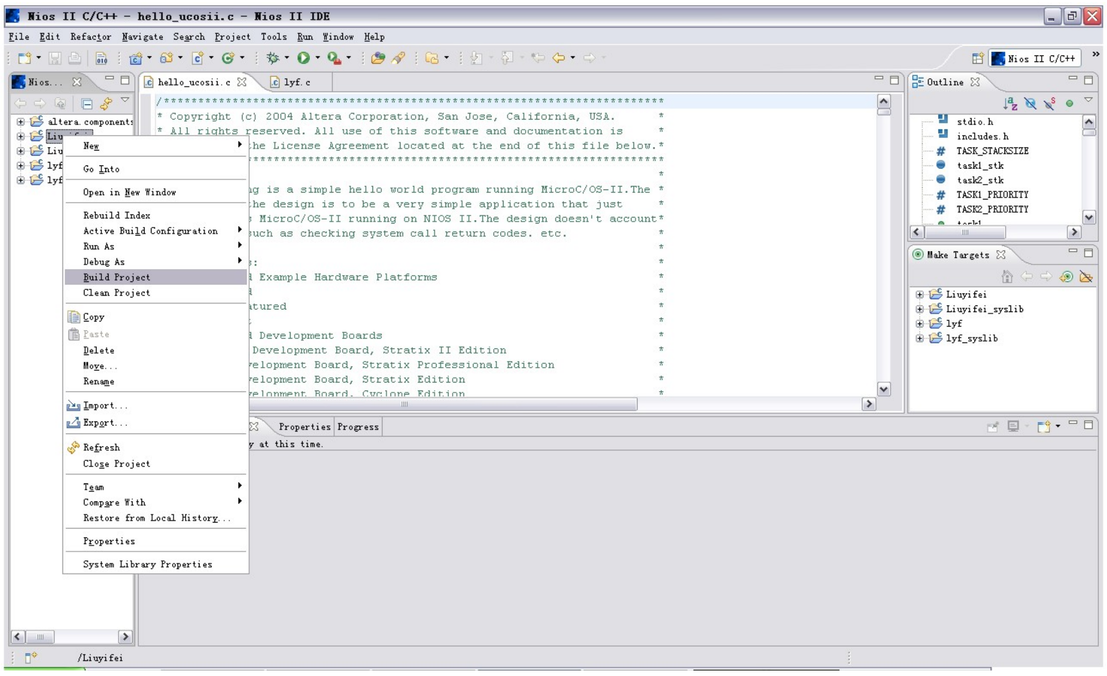
Figure 5-2 Setting window

Then click finish to build a new project from which I can start my own program. The main code is just as I introduced before.

5.3 Download the program

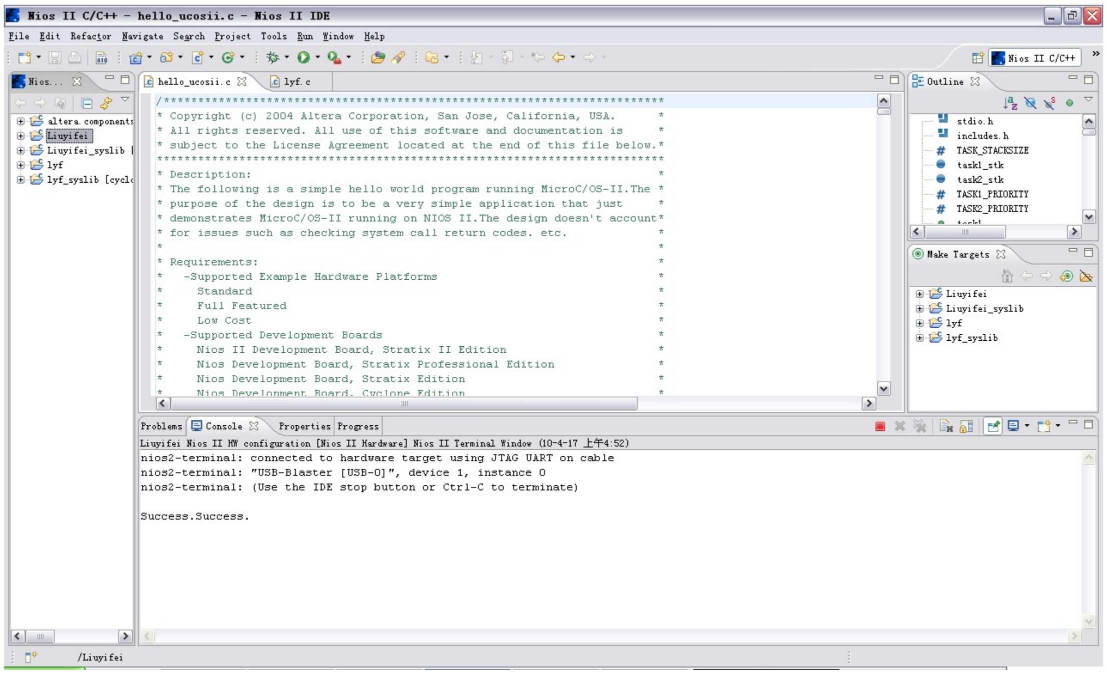
After finishing the program, it needs to be downloaded into the board to execute it.

First is building the program to compile it into a executable program. Right click the project, then choose "Build Project" like the following figure shows. Wait until the building finished. If there are errors in the program, it will inform you and you need to correct them until there is no error notified.



```
/****************************************************************************
* Copyright (c) 2004 Altera Corporation, San Jose, California, USA.        *
* All rights reserved. All use of this software and documentation is       *
```

Figure 5-3 Build the project


Nios II C/C++ - hello_ucosii.c - Nios II IDE
File Edit Refactor Navigate Search Project Tools Run Window Help
hello_ucosii.c
lyf.c
Outline
Make Targets
Problems
Console
Properties
Progress
/Liuyifei

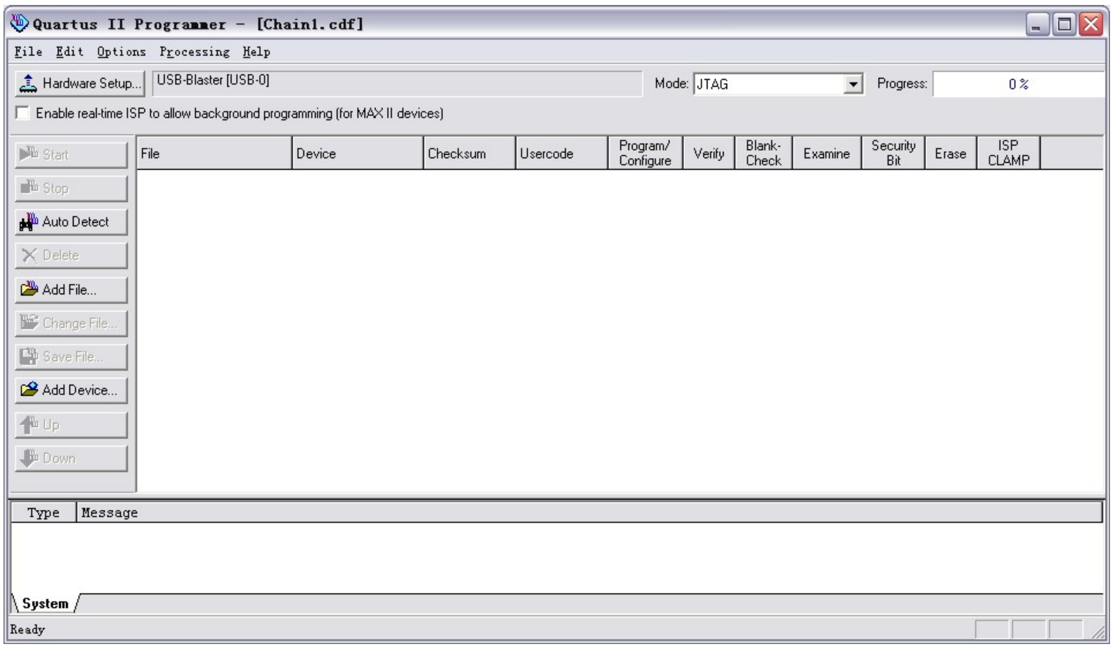
```
/****************************************************************************
* Copyright (c) 2004 Altera Corporation, San Jose, California, USA.        *
* All rights reserved. All use of this software and documentation is       *
* subject to the License Agreement located at the end of this file below.*
****************************************************************************
* Description:                                                             *
* The following is a simple hello world program running MicroC/OS-II.The *
* purpose of the design is to be a very simple application that just      *
* demonstrates MicroC/OS-II running on NIOS II.The design doesn't account*
* for issues such as checking system call return codes. etc.              *
*                                                                          *
* Requirements:                                                            *
*   -Supported Example Hardware Platforms                                  *
*     Standard                                                             *
*     Full Featured                                                        *
*     Low Cost                                                             *
*   -Supported Development Boards                                          *
*     Nios II Development Board, Stratix II Edition                        *
*     Nios Development Board, Stratix Professional Edition                 *
*     Nios Development Board, Stratix Edition                              *
*     Nios Development Board, Cyclone Edition                              *
```

```
Liuyifei Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (10-4-17 上午4:52)
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-0]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

Success.Success.
```

Figure 5-4 Building successful

After that is running the program. To do this, you should right click the project, choose Run As=>NIOS II Hardware. If you haven't added the SOF file to the kit before, it will inform you that you should do that first. At that time Quartus II is requested. A dialog box will automatically emerge as Figure 5-5 shows. Then click "Add File" in the left. Choose the .sof file and then click "start" to download it. When it finishes, you can again run the program as mentioned before, which is shown in Figure 5-7.



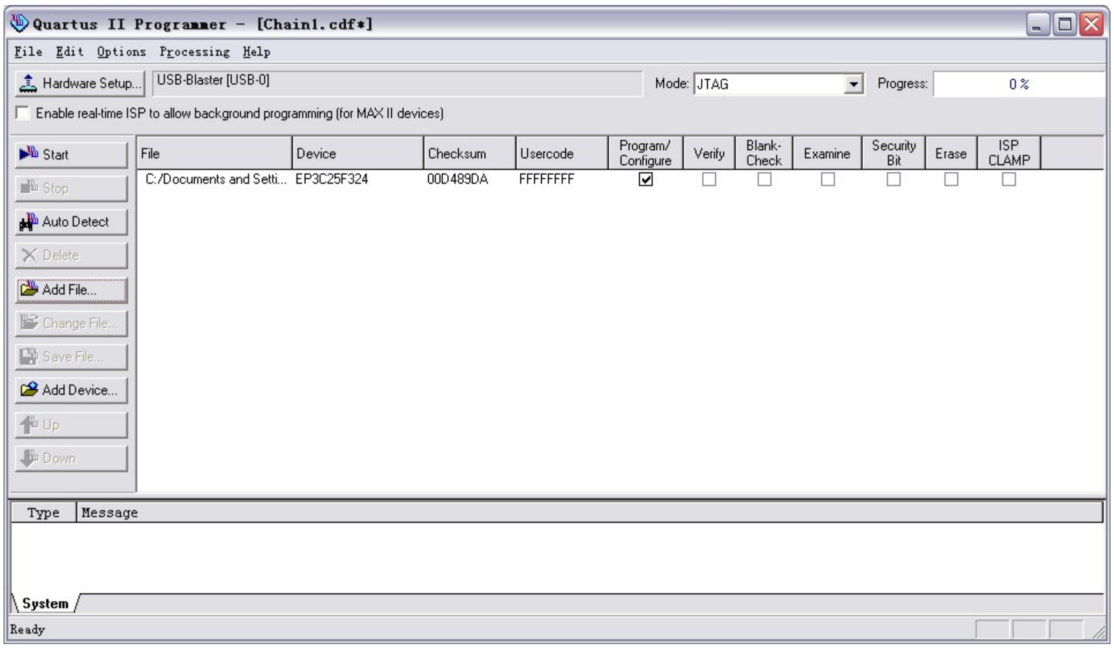
USB-Blaster [USB-0]

Mode: JTAG

Progress: 0 %

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

Figure 5-5 Download SOF file dialog box



Quartus II Programmer - [Chain1.cdf*]

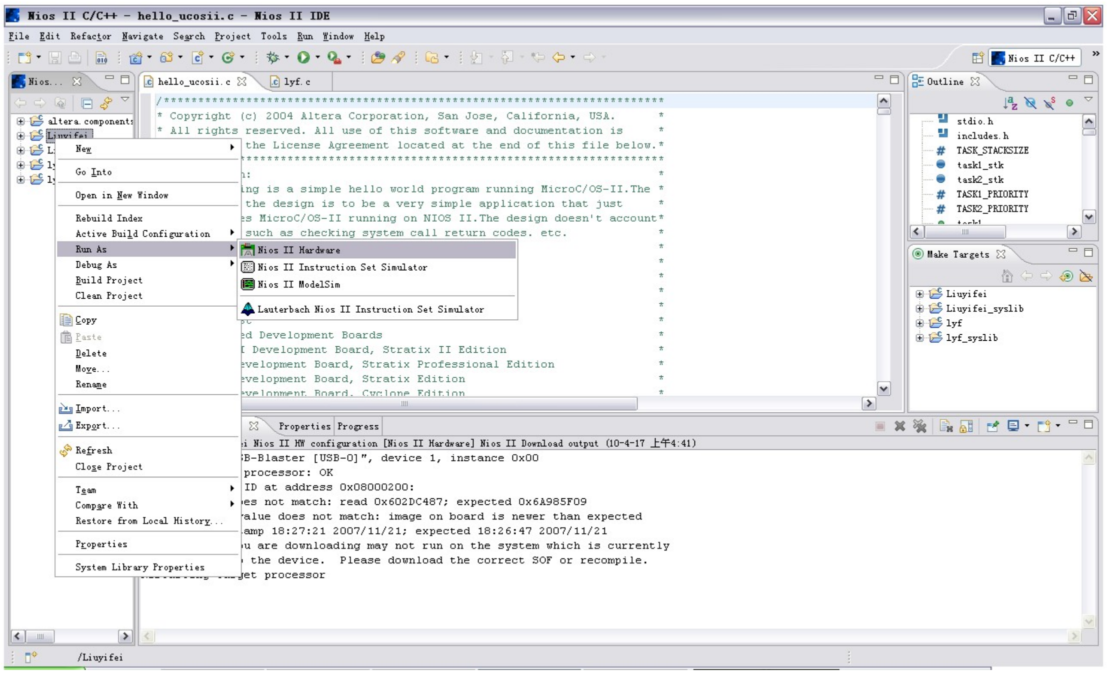
Hardware Setup... USB-Blaster [USB-0] Mode: JTAG Progress: 0 %

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:/Documents and Setti...	EP3C25F324	00D489DA	FFFFFFFF	☑	☐	☐	☐	☐	☐	☐

Ready

Figure 5-6 Add the SOF file



Nios II C/C++ - hello_ucosii.c - Nios II IDE
Run As
Nios II Hardware
Nios II Instruction Set Simulator
Nios II ModelSim
Lauterbach Nios II Instruction Set Simulator

```
/*************************************************************************
* Copyright (c) 2004 Altera Corporation, San Jose, California, USA.      *
* All rights reserved. All use of this software and documentation is     *
  the License Agreement located at the end of this file below.*
 *************************************************************************
 h:                                                                      *
 ing is a simple hello world program running MicroC/OS-II.The *
  the design is to be a very simple application that just      *
 es MicroC/OS-II running on NIOS II.The design doesn't account*
  such as checking system call return codes. etc.              *
 ed Development Boards                                          *
 I Development Board, Stratix II Edition                        *
 evelopment Board, Stratix Professional Edition                 *
 evelopment Board, Stratix Edition                              *
 evelopment Board, Cyclone Edition                              *
```

```
i Nios II HW configuration [Nios II Hardware] Nios II Download output (10-4-17 上午4:41)
B-Blaster [USB-0]", device 1, instance 0x00
 processor: OK
 ID at address 0x08000200:
es not match: read 0x602DC487; expected 0x6A985F09
alue does not match: image on board is newer than expected
amp 18:27:21 2007/11/21; expected 18:26:47 2007/11/21
u are downloading may not run on the system which is currently
 the device.  Please download the correct SOF or recompile.
get processor
```

Figure 5-7 Run the program

Then the downloading is finished and we can see the boot screen as Figure 4-3 showed before.

6. Conclusion

My final thesis is a project based on SOPC theory. It uses the NIOS II system to complete my own design, aiming at getting familiar with the FPGA work flow, and how to achieve it using NIOS II. After doing this project, I really grasped more about the basic idea of developing a project by the NIOS II embedded system. Besides getting information from the tutorial and internet to know how those devices, such as Cyclone III starter board, LCD Multimedia Daughtercard, NIOS II IDE and Quartus II, I obtained more from the process of working and the challenges I ever encountered.

It is obvious that when using new kinds of devices, there must be a lot of problems. The first challenge I came across was just the problem how to handle the hardware and software. After the LCD could work normally, I installed the NIOS II EDS 7.2 and Quartus II 7.2 into my computer and want to run the example program. At first there was something wrong with my computer as it couldn't find the usb blaster drive automatically. With the help of the internet, I used the install guide to find it in the drivers/usb-blasater subdirectory of the folder where Quartus II installed in. Although the board could connect to the computer, the Quartus II still could not download the program. After asking my tutor and searching in the internet, the reason was considered to be a mistake of the licence. Then I changed Quartus II 7.2 software into the version 9.1 Web Edition, which does not need a licence. At that time, it started working.

Then the most problems I encountered was in the process of programming. First was to understand the functions that the system has provided. The biggest challenge for me was how to reset the pen pointer. Because this is the first time I use touch screen devices, most of the functions related to touch pen were unknown for me. It took me a long time to find out the way to empty the pen pointer. Then when I got the tutorial find more functions related to the touchscreen, by experimenting several times, I created a function to especially initialize the touchscreen information.

Another problem in programming was about the delay function. Because the speed of the system to execute the instructions is fairly fast, most times people cannot react. So the delay function is important in the program. During programming, I encountered lots of problems which emerged because of this, so I created a delay function so that it can be called whenever needed.

The challenges I mentioned before were some of main problems. There was also details that cannot be all explicit. During the process of solving those problems, I got an extremely large number of knowledge about FPGA. Now I understand the basic flow of developing SOPC project and have experienced in solving problems. It will be considerably useful in the future.

REFERENCES

[1]touchscreen, wikipedia

http://en.wikipedia.org/wiki/Touchscreen

[2]Altera, wikipedia

http://en.wikipedia.org/wiki/Altera

[3]Programmable logic device, wikipedia

http://en.wikipedia.org/wiki/Programmable_logic_device

[4]Programmable Array Logic,wikipedia

http://en.wikipedia.org/wiki/Programmable_Array_Logic

[5]Field-programmable gate array, wikipedia

http://en.wikipedia.org/wiki/Field-programmable_gate_array

[6]PLD, 百度百科

http://baike.baidu.com/view/111579.htm?fr=ala0_1_1

[7]Introduction to Field Programmable Gate Arrays

http://cas.web.cern.ch/cas/Sweden-2007/Lectures/Web-versions/Serrano-1.pdf

[8]FPGA design flow overview

http://www.fpgacentral.com/docs/fpga-tutorial/fpga-design-flow-overview

[9]System-on-a-chip,wikipedia

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

[10]SOPC,百度百科

http://baike.baidu.com/view/525684.htm

[11]cycloneIII_3c25_start_board_reference_manual.pdf

[12]lcd_multimedia_daughtercard_reference_manual.pdf

[13]Introduction to the Quartus II Software

http://www.altera.com/literature/manual/intro_to_quartus2.pdf

[14]Nios II IDE Help System

http://courses.cit.cornell.edu/ece576/NiosII_doc/ug_nios2_ide_help.pdf