Bernburg
Dessau
Köthen

Hochschule Anhalt
Anhalt University of Applied Sciences

emw
Fachbereich
Elektrotechnik, Maschinenbau
und Wirtschaftsingenieurwesen

Final Report

Hardware/Software Co-Design

Ibrahim Hassan , MET 2023, 5152780

Amer Haj Kasem, MET 2023, 4070916

Mehmet Cinar, MET 2023, 5128329

Eyuel Deribe Abera, MBE 2022, 5114414

Topic:

Design of a Picture Processing System

22.04.2024

Date of Submission

Prof. Dr.-Ing. Ingo Chmielewski &

Prof. Dr Michael Brutscheck

Contents

1 Motivation and goal setting

1.1 Introduction

For the Hardware/Software codesign project, we choose to work on picture processing system with the DE2-115 board and 4.3” LCD touch panel. With this project it is intended that we become more fluent with VHDL coding in hardware part and learn more about c programming in software part.

1.2 Objective

In our project we aimed to manage reading jpg images from SD card, decoding jpg images, resizing image according to the screen, applying grey scale on images, applying edge detection, and designing a basic file browser.

Our primary objective was to read the images from a SD card. The SD had to be compatible with the DE2-115 board and in the correct format. Later the images should fit in the screen with suitable proportions.

Moreover, applying the grey scale and edge detection should be in a manner that does not shuffle the proportions of the images. Lastly, our design should allow the user to easily navigate between images and apply grey scale and edge detection.

1.3 Methods

The methodology for this project encompasses the following stages:

1. **LCD Interfacing**: Established connection for both display and touch functionalities.
2. **SD Card Reader Interfacing**: Linked SD card reader and initialized data reading.
3. **Image Reading**: Extracted JPEG images from SD card.
4. **Image Decoding**: Converted JPEG format to usable data.
5. **Image Resizing**: Adjusted image dimensions to fit LCD screen.
6. **Gray Scale Application**: Implemented grayscale transformation.
7. **Edge Detection**: Executed edge detection algorithms.
8. **File Browser**: Developed interface for navigating SD card files.

1.4 Software

The software tools used in this research include:

1. Word for documentation.
2. Quartus II 13.0sp1 (64-bit) Web Edition.
3. Nios II 13.0sp1 Software Build Tools for Eclipse.
4. ChatGPT by OpenAI for troubleshooting and research.
5. JPEG Library for image decoding and scaling.

2 Hardware

2.1 Altera DE2-115 Board

The flexibility of FPGAs gives us the possibility to integrate additional applications and image processing algorithms to the system without any cost in hardware. Its offers advantages in terms of lower power consumption, lower cost, and abundance of logic, memory and digital signal processing capabilities. Responding to increased versatile low-cost spectrum needs driven by the demand for mobile video, voice, data access, and the hunger for high-quality images, the new DE2-115 offers an optimal balance of low cost, low power and a rich supply of logic, memory and DSP capabilities.

The DE2 series has consistently been at the forefront of educational development boards by distinguishing itself with an abundance of interfaces to accommodate various application needs. Extending its leadership and success, Terasic announces the latest DE2-115 that features the Cyclone IV E device. Responding to increased versatile low-cost spectrum needs driven by the demand for mobile video, voice, data access, and the hunger for high-quality images, the new DE2-115 offers an optimal balance of low cost, low power and a rich supply of logic, memory and DSP capabilities.The Cyclone EP4CE115 device equipped on the DE2-115 features 114,480 logic elements (LEs), the largest offered in the Cyclone IV E series, up to 3.9-Mbits of RAM, and 266 multipliers. In addition, it delivers an unprecedented combination of low cost and functionality, and lower power compared to previous generation Cyclone devices.The DE2-115 adopts similar features from the earlier DE2 series primarily the DE2-70, as well as additional interfaces to support mainstream protocols including Gigabit Ethernet (GbE). A High-Speed Mezzanine Card (HSMC) connector is provided to support additional functionality and connectivity via HSMC daughter cards and cables. For large-scale ASIC prototype development, a connection can be made with two or more FPGA-based boards by means of a HSMC cable through the HSMC connector.

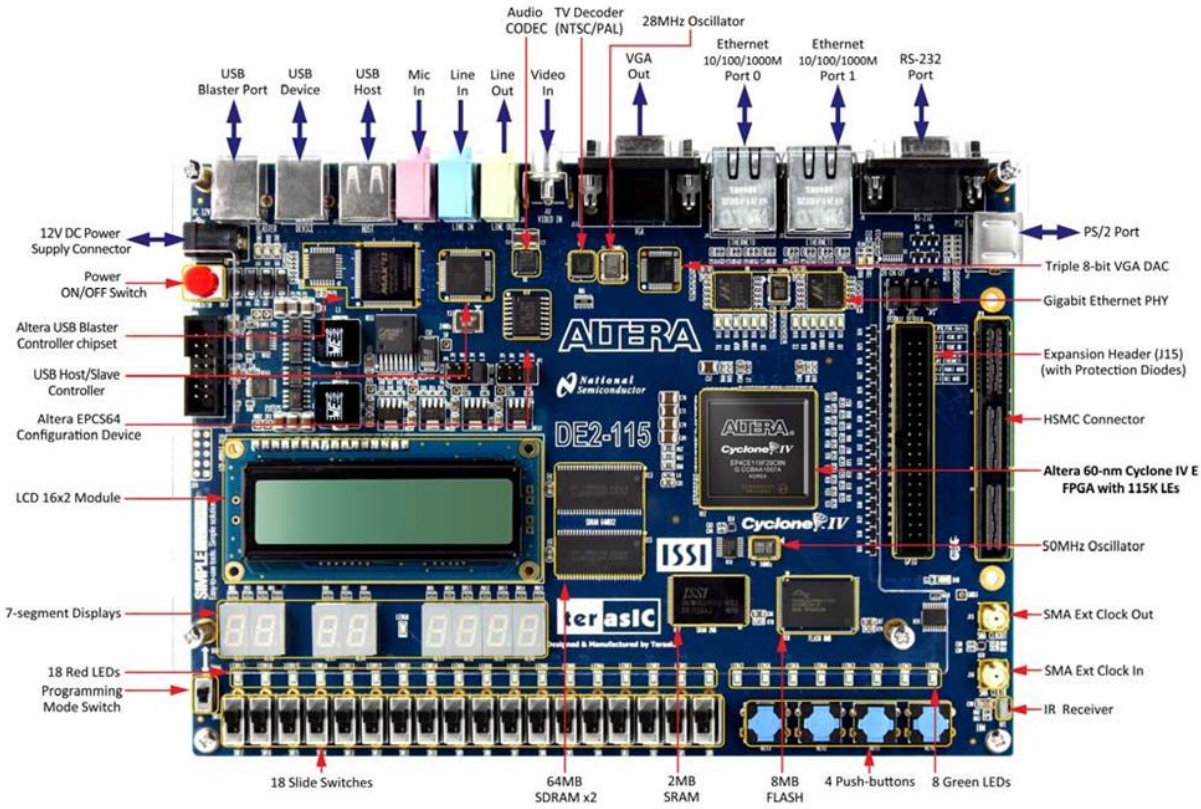


Figure 2.1 DE2-115 Front

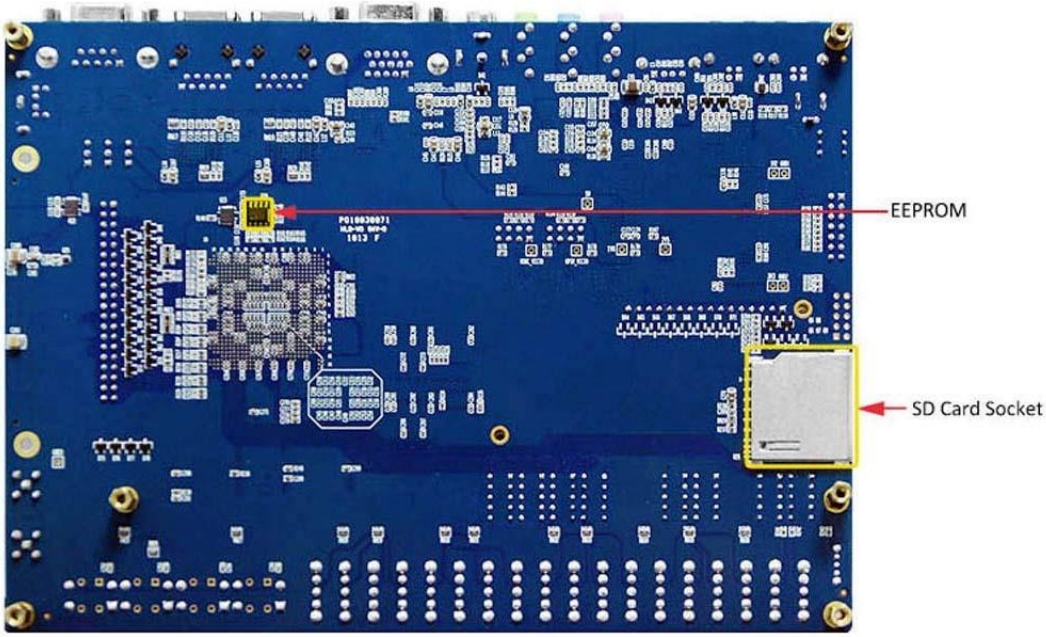


Figure 2.2 DE2-115 Back

The DE2-115 board has many features that allow users to implement a wide range of designed circuits, from simple circuits to various multimedia projects.The following hardware is provided on the DE2-115 board:

- Altera Cyclone IV 4CE115 FPGA device

- Altera Serial Configuration device – EPCS64
- USB Blaster (on board) for programming; both JTAG and Active Serial (AS) programming modes are supported
- 2MB SRAM
- Two 64MB SDRAM
- 8MB Flash memory
- SD Card socket
- 4 Push-buttons
- 18 Slide switches
- 18 Red user LEDs
- 9 Green user LEDs
- 50MHz oscillator for clock sources
- 24-bit CD-quality audio CODEC with line-in, line-out, and microphone-in jacks
- VGA DAC (8-bit high-speed triple DACs) with VGA-out connector
- TV Decoder (NTSC/PAL/SECAM) and TV-in connector
- 2 Gigabit Ethernet PHY with RJ45 connectors
- USB Host/Slave Controller with USB type A and type B connectors
- RS-232 transceiver and 9-pin connector
- PS/2 mouse/keyboard connector
- IR Receiver
- 2 SMA connectors for external clock input/output
- One 40-pin Expansion Header with diode protection
- One High Speed Mezzanine Card (HSMC) connector
- 16x2 LCD module

This figure gives the block diagram of the DE2-115 board. To provide maximum flexibility for the user, all connections are made through the Cyclone IV E FPGA device. Thus, the user can configure the FPGA to implement any system design.

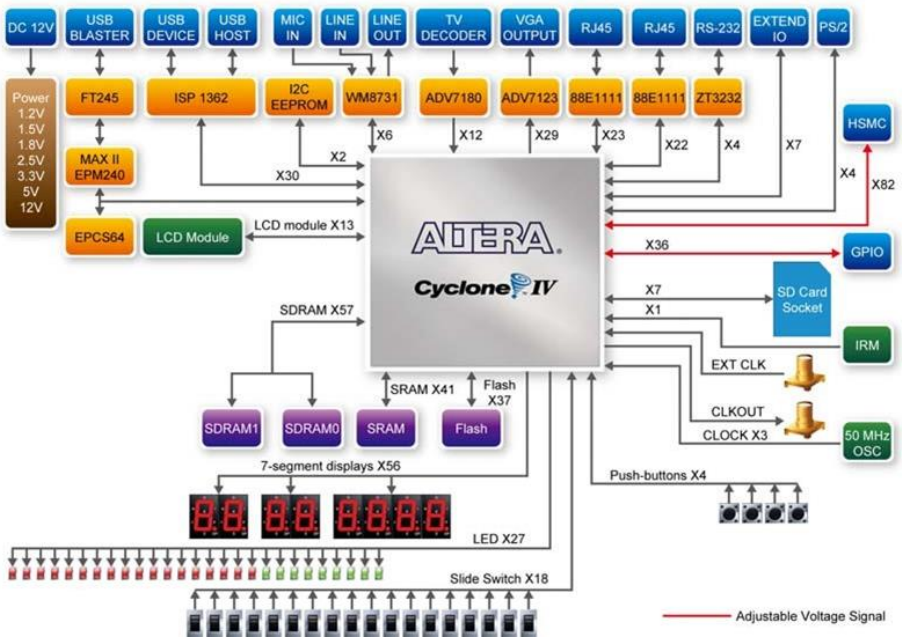


Figure 2.3 Block Diagram of DE2-115

2.2 TRDB_LTM Touch Screen Panel

The TRDB_LTM (LTM) Kit provides everything you need to develop applications using a digital touch panel on an Altera DE4/DE2-115/DE2/DE1 board. The kit contains complete reference designs and source code for implementing a photo viewer demonstration and a color pattern generator using the LTM kit and an Altera DE2-115/DE2/DE1 board. This chapter provides users key information about the kit.

Figure 2.4 LTM

The feature set of the LTM is listed below:

1. Equipped with Toppoly TD043MTEA1 active matrix color TFT LCD module.
2. Support 24-bit parallel RGB interface.
3. 3-wire register control for display and function selection.
4. Built-in contrast, brightness, and gamma modulation.
5. Converting the X/Y coordination of the touch point to its corresponding digital data via the Analog Devices AD7843 AD converter.

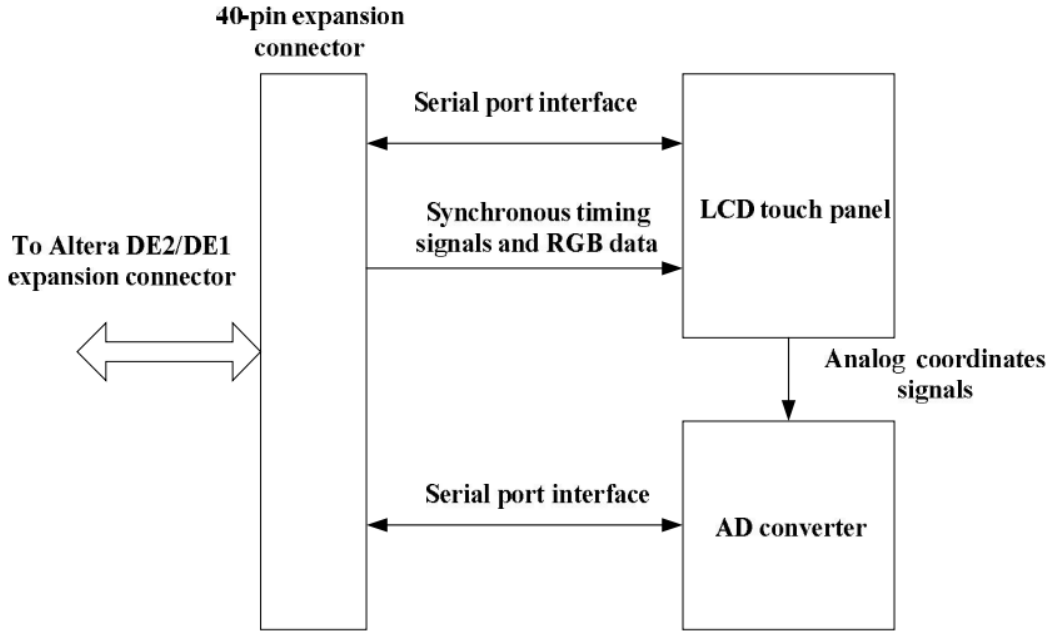


Figure 2.5 Block Diagram of LTM

The LTM consists of three major components: LCD touch panel module, AD converter, and 40-pin expansion header. All of the interfaces on the LTM are connected to Altera DE4/DE2-115/DE2/DE1 board via the 40-pin expansion connector. The LCD and touch panel module will take the control signals provided directly from FPGA as input and display images on the LCD panel. Finally, the AD converter will convert the coordinates of the touch point to its corresponding digital data and output to the FPGA via the expansion header.

The LCD and touch panel module on the LTM is equipped with a LCD driver IC to support three display resolutions and with functions of source driver, serial port interface, timing controller, and power supply circuits. To control these functions, users can use FPGA to configure the registers in the LCD driver IC via serial port interface.

Also, there is an analog to digital converter (ADC) on the LTM to convert the analog X/Y coordinates of the touch point to digital data and output to FPGA through the serial port interface of the ADC. Both LCD driver IC and ADC serial port interfaces are connected to the FPGA via the 40-pin expansion header and IDE cable.

Because of the limited number of I/O on the expansion header, the serial interfaces of the LCD driver IC and ADC need to share the same clock (ADC_DCLK) and chip enable (SCEN) signal I/O on the expansion header. To avoid both the serial port interfaces may interfere with each other when sharing the same clock and chip enable signals, the chip enable signal (CS), which is inputted into the ADC will come up with a logic inverter. Users need to pay attention controlling the shared signals when designing the serial port interface controller.

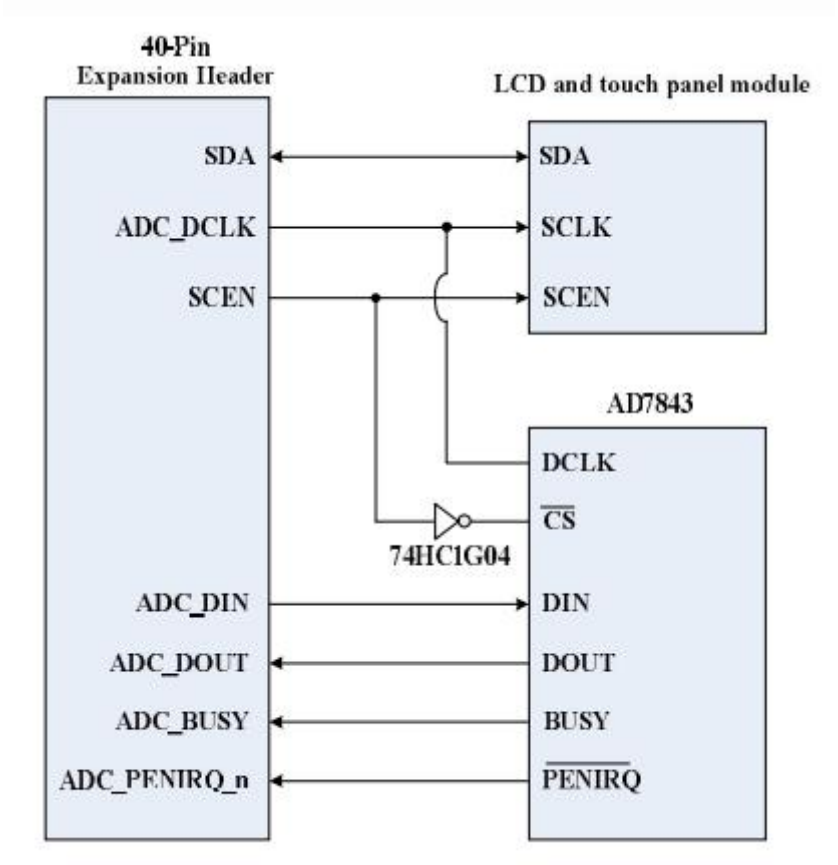


Figure 2.6 The Serial Interface of the LCD

The LCD driver IC supports a clock synchronous serial interface as the interface to a FPGA to enable instruction setting. Please notice that in addition to the serial port interface signals, NCLK input should also be provided while setting the registers. Figure 3.2 and Table 3.1 show the frame format and timing diagram of the serial port interface. The LCD driver IC recognizes the start of data transfer on the falling edge of SCEN input and starts data transfer. When setting instruction, the TPG110 inputs the setting values via SDA on the rising edge of input SCL.

To determine the sequencing and the timing of the image signals displayed on the LCD panel, the corresponding synchronous signals from FPGA to the LCD panel should follow the timing specification.

Figure 5 illustrates the basic timing requirements for each row (horizontal) that is displayed on the LCD panel. An active-low pulse of specific duration (time in the figure) is applied to the horizontal synchronization (HD) input of the LCD panel, which signifies the end of one row of data and the start of the next. The data (RGB) inputs on the LCD panel are not valid for a time period called the hsync back porch ( ) after the hsync pulse occurs, which is followed by the display area ( ). During the data display area the RGB data drives each pixel in turn across the row being displayed. Also, during the period of the data display area, the data enable signal (DEN) must be driven to logic high. Finally, there is a time period called the hsync front porch ( ) where the RGB signals are not valid again before the next hsync pulse can occur.

The timing of the vertical synchronization (VD) is the same as shown in Figure 3.4, except that a vsync pulse signifies the end of one frame and the start of the next, and the data refers to the

set of rows in the frame (horizontal timing). Table 3.2 and 3.3 show for different resolutions, the durations of time periods , , , and for both horizontal and vertical timing. Finally, the timing specification of the synchronous signals is shown in the Table 3.4.

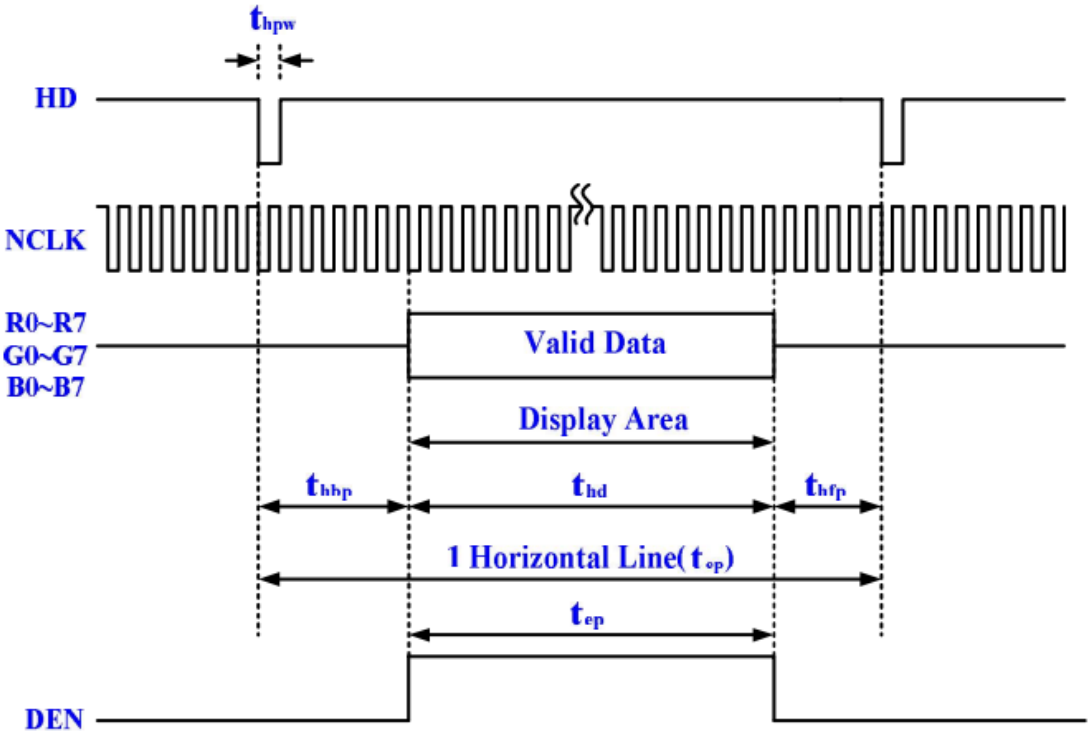


Figure 2.7 LCD horizantal timing specification

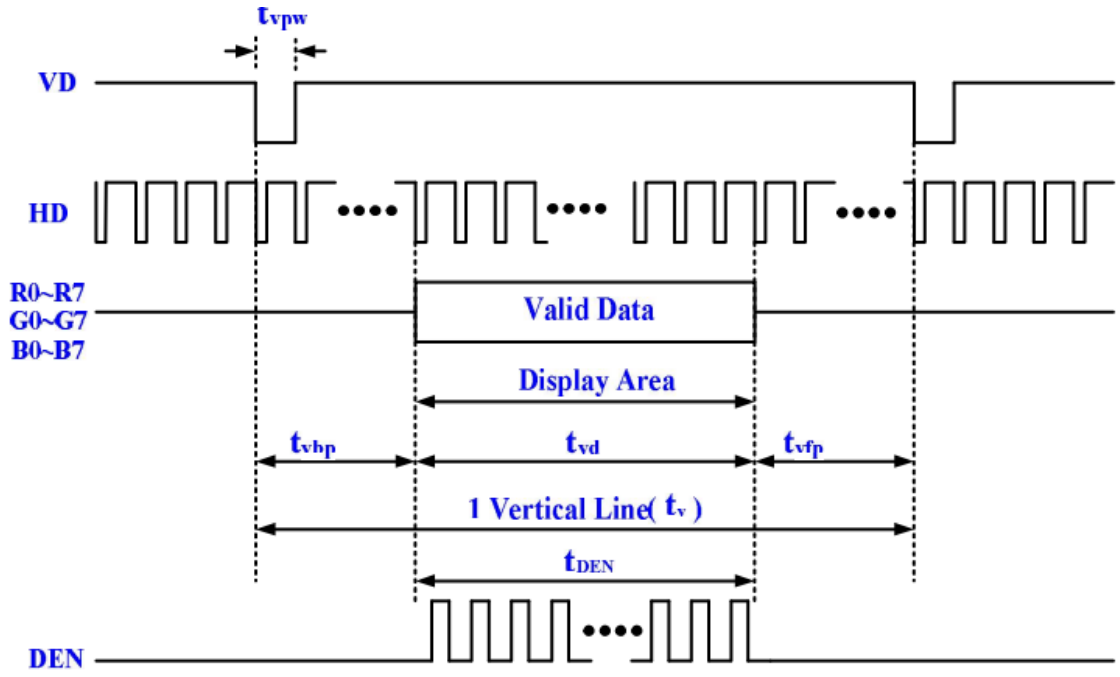


Figure 2.8 LCD Vertical Timing Specification

Parameter	Symbol	Min.	Unit
NCLK period	PW_{CLK*1}	25	ns
NCLK pulse high period	PWH_{*1}	10	ns
NCLK pulse low period	PWL_{*1}	10	ns
HD,VD, DEN, data setup time	t_{ds}	5	ns
HD,VD, DEN, data hold time	t_{dh}	5	ns

Figure 2.9 The Timing parameters of the LCD synchronous signals

The LTM also equipped with an Analog Devices AD7843 touch screen digitizer chip. The AD7843 is a 12-bit analog to digital converter (ADC) for digitizing x and y coordinates of touch points applied to the touch screen. The coordinates of the touch point stored in the AD7843 can be obtained by the serial port interface.

To obtain the coordinate from the ADC, the first thing users need to do is monitor the interrupt signal ADC_PENIRQ_n outputted from the ADC. By connecting a pull high resistor, the ADC_PENIRQ_n output remains high normally. When the touch screen connected to the ADC is touched via a pen or finger, the ADC_PENIRQ_n output goes low, initiating an interrupt to a FPGA that can then instruct a control word to be written to the ADC via the serial port interface.

As soon as the bit stream is downloaded into the FPGA, the register values of the LCD driver IC using to control the LCD display function will be configured by the LCD_SPI_Controller block, which uses the serial port interface to communicate with the LCD driver IC. Meanwhile, the Flash_to_SDRAM_Controller block will read the RGB data of one picture stored in the Flash, and then write the data into SDRAM buffer. Accordingly, both the synchronous control signals and the picture data stored in the SDRAM buffer will be sent to the LTM via the LCD_Timing_Controller block.

When users touch LTM screens, the x and y coordinates of the touch point will be obtained by the ADC_SPI_Controller block through the ADC serial port interface. Then the Touch_Point_Detector block will determine whether these coordinates are in a specific range. If the coordinates fit the range, the Touch_Point_Detector block will control the Flash_to_SDRAM_Controller block to read the next or previous picture's data from the Flash and repeat the steps as mentioned before to command the LTM to display the next or previous picture.

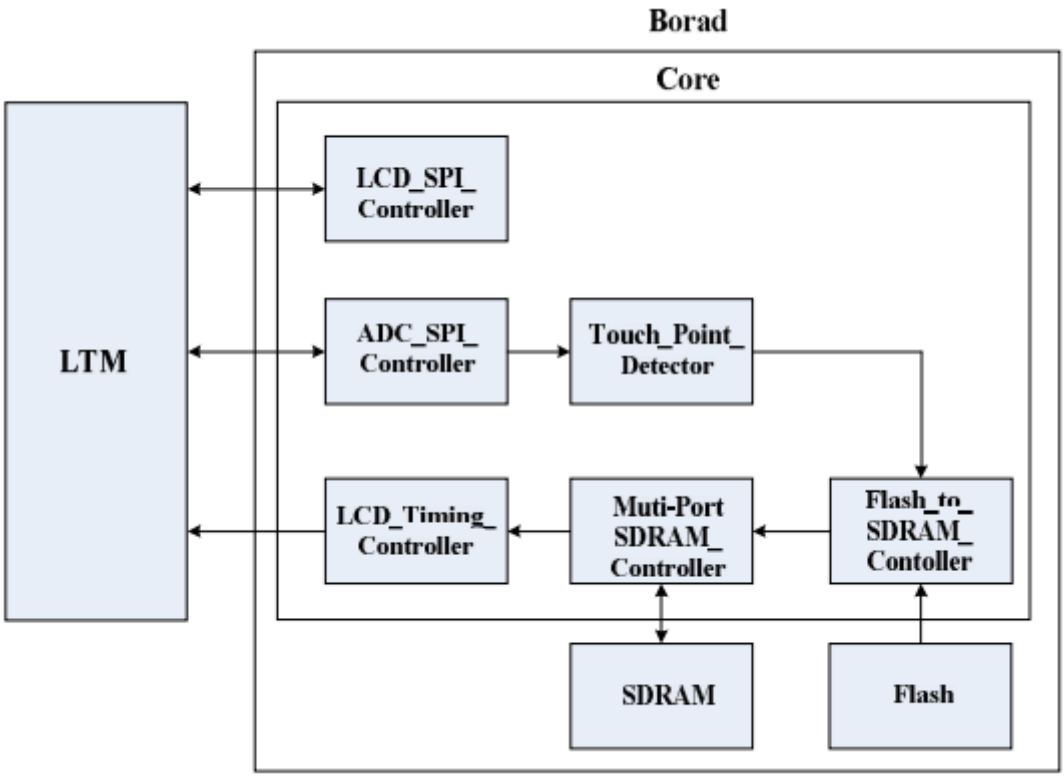


Figure 2.10 Block Diagram of the Ephoto Demonstration

2.3 SD Reader

SD (secure digital) card is a memory card widely used for massive storage. In this design, the SD card is utilized to store the raw image data and acceleration data. A block diagram of the SD card is shown in figure. It consists of a 9-pin interface, a card controller, a memory interface and a memory core. The 9-pin interface allows the exchange between a connected system and the card controller. The controller can read/write data from/to the memory core using the memory core interface. In addition, several internal registers are provided to store the state of the card.

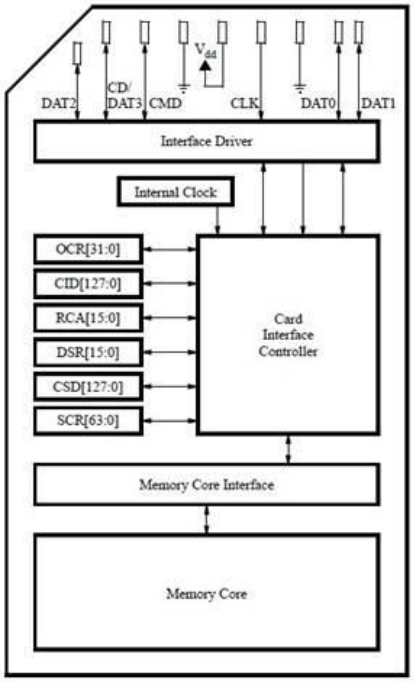


Figure 2.11 SD Card Block Diagram

To interface with an SD card, the VEEK-MT features an SD card socket, on which pins labeled as CLK, CMD, DAT0, and CD/DAT3 are connected to the FPGA. The data exchange between the FPGA and the SD card can adopt one of the two modes: SD mode or SPI (Serial Peripheral Interface) mode. The SD mode is a proprietary format and uses four lines for data transfer. The SPI is an open standard for serial interfaces and is widely used in embedded applications. We selects SPI mode for SD card communication. Therefore the SD Controller in this Nios II system mainly implements the SPI protocol.

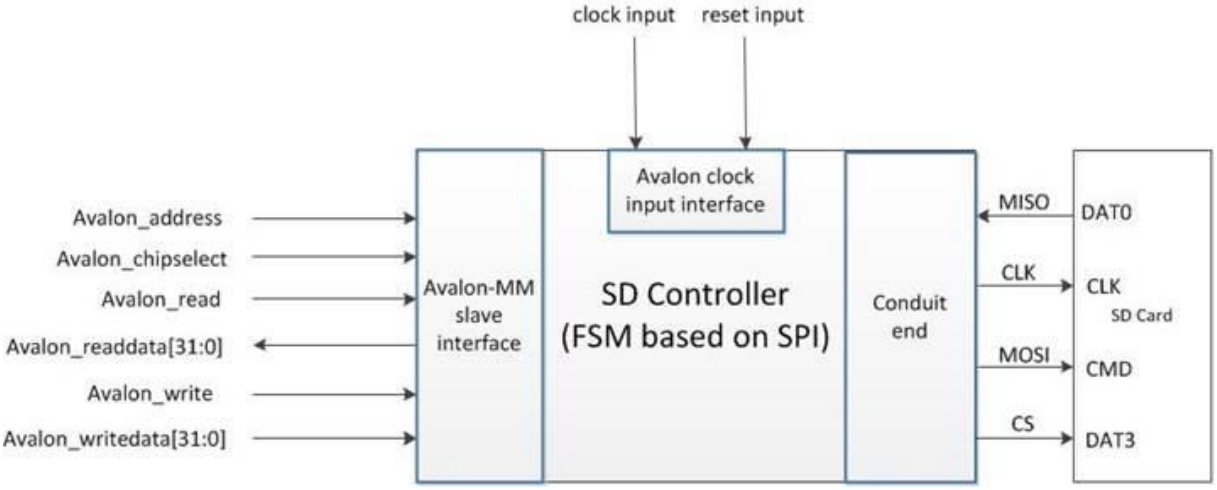


Figure 2.12 SD Controller IP

3 Hardware Architecture

3.1 Devolopment Flow

The embedded SOPC system design consists of hardware development and software development, which are implemented by **Quartus II Design Software** and **Nios II Embedded Design Suite** respectively. We select version 14.1 as our development platforms. The basic development flow (mainly for our design) is shown in figure 3.1.

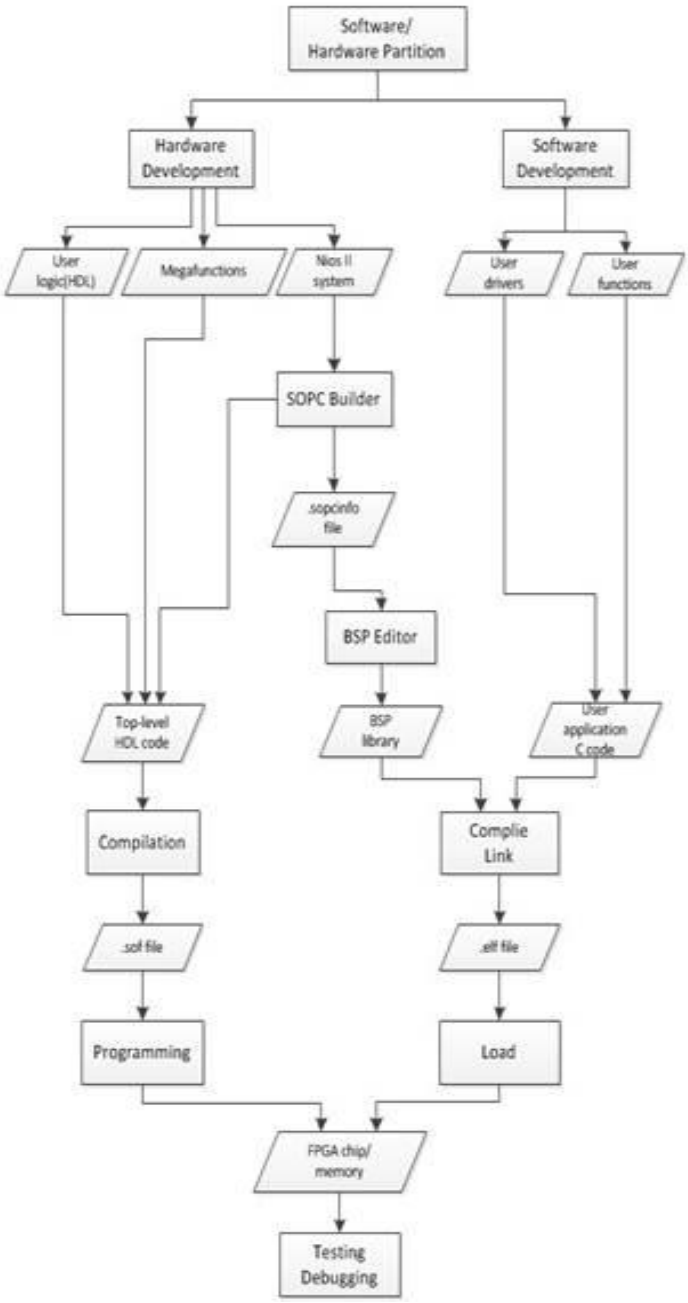


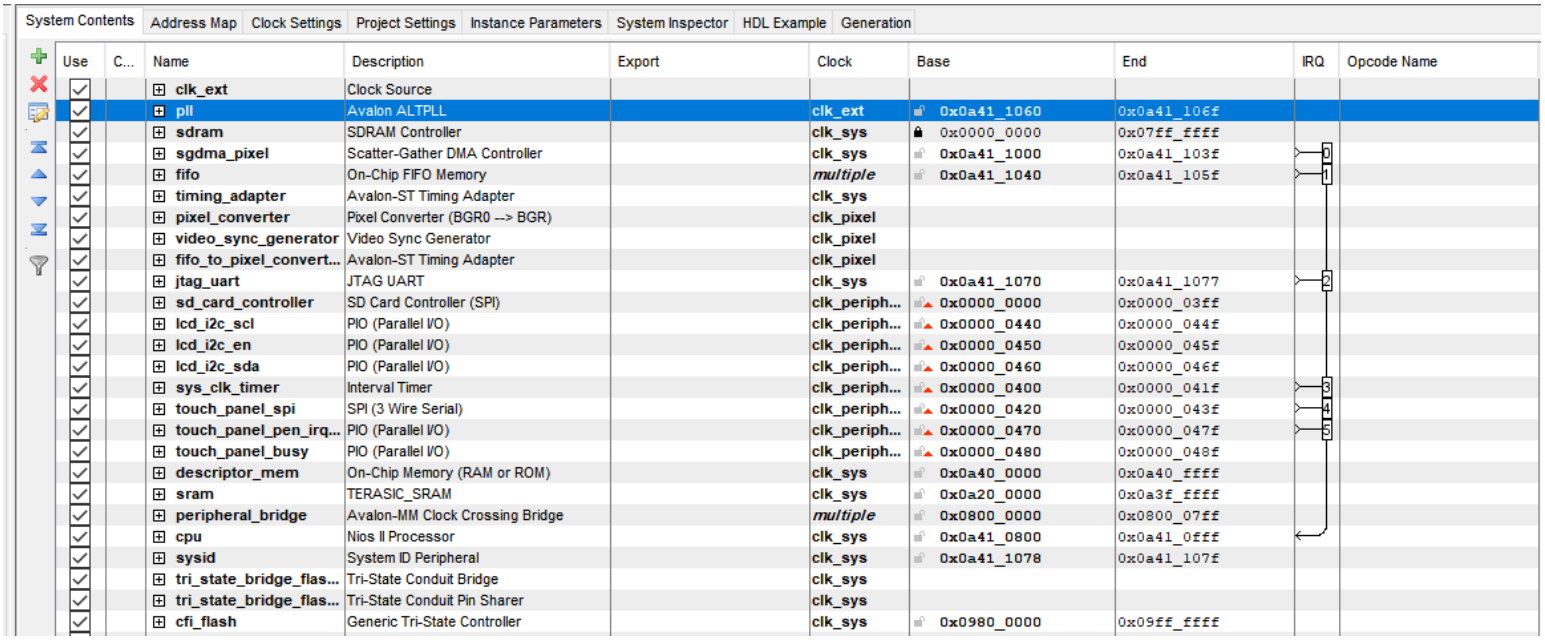
Figure 3.1 Quartus II Design Flow

The Altera Quartus II Design Software provides a complete, multi-platform design environment that easily adapts to specific hardware development. It includes solutions for all phases of FPGA and CPLD design through the easy-to-use graphical user interface. The left branch in Figure represents the Quartus II-based hardware design flow. A detailed description is given below:

1. **Design Entry**: We use Hardware Description Language (Verilog HDL), Mega Wizard Plug-In Manager and SOPC Builder to build system-level design. Mega functions are parameterizable functional blocks. The Mega Wizard Plug-In Manager allows one to create custom mega functions which can be instantiated in design files. The SOPC Builder can help implement customized Nios II system. In this software package, one can configure the Nios II soft processor, select the desired standard I/O cores, and incorporate the user designed I/O peripherals. Then the SOPC Builder generates the HDL code for the customized Nios II system and also generates the .sopcinfo file that contains system configuration information. This code is combined with other HDL code to form the top-level HDL description of the complete hardware. In addition, initial design constraints should be specified through Assignment editor and Settings dialog box.
2. **Compilation**: The compilation process consists of analysis and synthesis, fitting, timing analysis and assembling. This process realizes functions such as checking the syntax of HDL code, transforming HDL constructs to gate-level components, deriving the layout inside FPGA chip, performing timing analysis and finally producing programming files (.sof for JTAG programming, .pof for AS programming).
3. **Programming and Debugging**: In this step, the configuration file is downloaded into the target device. The Signal Tap II Logic Analyzer can be used for debugging. This logic analyzer captures real-time signal behavior and supervises the interactions between hardware and software in the system design. The Quartus II software allows one to select signals to capture, when signal capture starts, and how many data samples to capture

3.2 Verilog and Qsys Builder

The Verilog portion of the program is primarily generated from the **Qsys** configuration. The code is contained in different Verilog files split up into various components with one main file, DE2_115_LTM_Pic.v. This establishes some of the outside connections between the FPGA and DE2-115's hardware and the hardware connections into the NIOS II. The program uses a most of the DE2's features and these create the connections and interfaces in the FPGA necessary to access them.



Use	C...	Name	Description	Export	Clock	Base	End	IRQ	Opcode Name
		clk_ext	Clock Source						
		pll	Avalon ALTPLL		clk_ext	0x0a41_1060	0x0a41_106f		
		sdram	SDRAM Controller		clk_sys	0x0000_0000	0x07ff_ffff		
		sgdma_pixel	Scatter-Gather DMA Controller		clk_sys	0x0a41_1000	0x0a41_103f	0	
		fifo	On-Chip FIFO Memory		*multiple*	0x0a41_1040	0x0a41_105f	1	
		timing_adapter	Avalon-ST Timing Adapter		clk_sys				
		pixel_converter	Pixel Converter (BGR0 --> BGR)		clk_pixel				
		video_sync_generator	Video Sync Generator		clk_pixel				
		fifo_to_pixel_convert...	Avalon-ST Timing Adapter		clk_pixel				
		jtag_uart	JTAG UART		clk_sys	0x0a41_1070	0x0a41_1077	2	
		sd_card_controller	SD Card Controller (SPI)		clk_periph...	0x0000_0000	0x0000_03ff		
		lcd_i2c_scl	PIO (Parallel I/O)		clk_periph...	0x0000_0440	0x0000_044f		
		lcd_i2c_en	PIO (Parallel I/O)		clk_periph...	0x0000_0450	0x0000_045f		
		lcd_i2c_sda	PIO (Parallel I/O)		clk_periph...	0x0000_0460	0x0000_046f		
		sys_clk_timer	Interval Timer		clk_periph...	0x0000_0400	0x0000_041f	3	
		touch_panel_spi	SPI (3 Wire Serial)		clk_periph...	0x0000_0420	0x0000_043f	4	
		touch_panel_pen_irq...	PIO (Parallel I/O)		clk_periph...	0x0000_0470	0x0000_047f	5	
		touch_panel_busy	PIO (Parallel I/O)		clk_periph...	0x0000_0480	0x0000_048f		
		descriptor_mem	On-Chip Memory (RAM or ROM)		clk_sys	0x0a40_0000	0x0a40_ffff		
		sram	TERASIC_SRAM		clk_sys	0x0a20_0000	0x0a3f_ffff		
		peripheral_bridge	Avalon-MM Clock Crossing Bridge		*multiple*	0x0800_0000	0x0800_07ff		
		cpu	Nios II Processor		clk_sys	0x0a41_0800	0x0a41_0fff		
		sysid	System ID Peripheral		clk_sys	0x0a41_1078	0x0a41_107f		
		tri_state_bridge_flas...	Tri-State Conduit Bridge		clk_sys				
		tri_state_bridge_flas...	Tri-State Conduit Pin Sharer		clk_sys				
		cfi_flash	Generic Tri-State Controller		clk_sys	0x0980_0000	0x09ff_ffff		

Figure 3.2 The Project Qsys File

The Qsys file is provided as an example to interface the Touch panel and the SD card controller, beside the various clocks and busses the main components are :

- **cpu** Nios ii processor used to run the software part of the project (C code)
- **sdram** used as memory for the project
- **lcd_i2c** controls the serial port interface of the LTM, transmitting data to the LCD register
- **adc_spi_controller** configures the ADC and returns the pressed coordinates of the LCD panel
- **touch_panel_spi** used as the main control input to control the program
- **sd_card_controller** the main interface to access the sd card
- **pll** used to generate various clocks from the input 50 MHz clock

3.3 Block Diagram

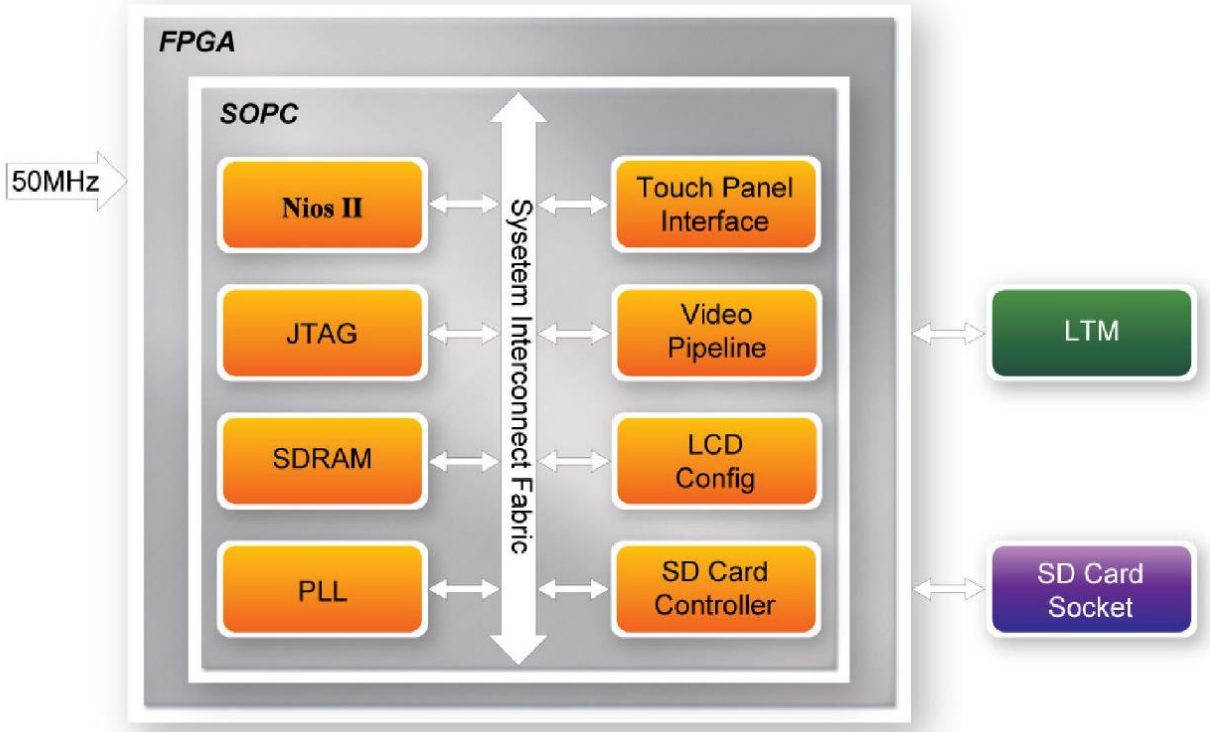


Figure 3.3 Block Diagram

1. **Nios II**: The main software code runs on the Nios II softcore processor, including decoding jpeg file, applying patterns and running the file browser system.
2. **SD Card Controller**: The SD card is initialized and read through the SD card controller.
3. **Data Storage in SDRAM**: JPEG image data is transferred from the SD card to memory.
4. **LCD Display**: Initialize and controls the LCD using SPI.
5. **Video Pipeline**: Is modified to just access one buffer which is currently displayed on screen instead of loading multiple buffers at once.
6. **Touch Panel Interface**: Controls the AD converter used to obtain the X/Y coordinates of the touch point.
7. **PLL**: Generate various clocks using the input 50 MHz clock for various project elements.

4 Software Structure

4.1 Introduction

The Altera Nios II Embedded Design Suite (EDS) is used for software development. It can be accessed by the SBT (Software Build Tools) command-line interface, by the SBT GUI, or by the IDE GUI. The SBT GUI is selected to develop the software in this design. The SBT GUI is based on the Eclipse open development environment and supports creating, modifying, building, running and debugging C programs targeted for a Nios II system.

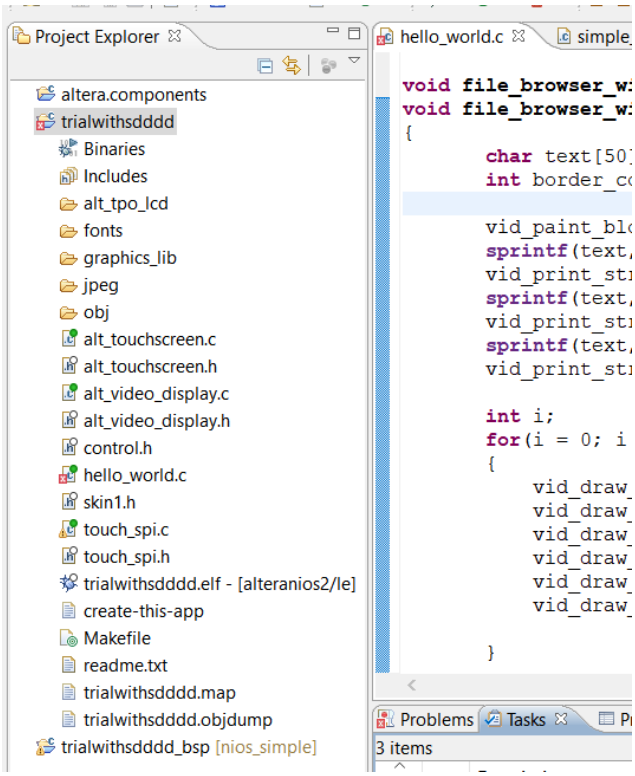


Figure 4.1 Main Project Software and BSP Code

A Nios II software project contains two major parts: user applications and BSP (Board Support Package). The former is the user's programs which include user I/O drivers and user high-level functions, and the latter is the support codes for a specific Nios II configuration. Note that the BSP is based on the information from the .sopcinfo file generated by SOPC Builder. The code from the two parts are compiled and linked into a single software image (.elf) and loaded into as shown in figure 4.1.

4.2 LCD Panel

We primarily rely on the API offered by the Panel manufacturer for initializing and engaging with the LCD touch panel. Shown below, we demonstrate the initialization processes for both the touch controller and the display of the panel.

```
int result;
////Initial Touch panel
TOUCH_HANDLE hTouch;
hTouch = Touch_Init(TOUCH_PANEL_SPI_BASE, TOUCH_PANEL_PEN_IRQ_N_BASE, TOUCH_PANEL_PEN_IRQ_N_IRQ);
if (!hTouch)
{
    printf("Failed to init touch");
}
////Initial LCD Display
alt_video_display* display_global;
printf("Initializing LCD display controller\n  ");
display_global = alt_video_display_init( "/dev/sgdma_pixel",              // Name of video controller
                                         800,                            // Width of display
                                         480,                            // Height of display
                                         32,                             // Color depth (32 or 16)
                                         SDRAM_BASE+SDRAM_SPAN/2,        // Where we want our frame buffers
                                         DESCRIPTOR_MEM_BASE,            // Where we want our descriptors
                                         NUM_FRAME );
if( display_global )
    printf(" - LCD Initialization OK\n");
else
    printf(" - LCD FAILED\n");
alt_video_display_clear_screen(display_global,0x0);
///Setup the LCD parameter
alt_tpo_lcd s_lcd;
alt_tpo_lcd * lcd = &s_lcd;
lcd->scen_pio = LCD_I2C_EN_BASE;
lcd->scl_pio  = LCD_I2C_SCL_BASE;
lcd->sda_pio  = LCD_I2C_SDA_BASE;
result = alt_tpo_lcd_init(lcd, 800, 480);
if(result)
  {
    printf("LCD spi write failed\n"); // failure
  }
else
  {
    printf("LCD spi write OK\n"); // success
  }
```

4.3 SD Card

Using the API provided by the sd_controller IP and the FAT library to interact with the FAT16 file system, we initialize the SD card. We then retrieve the names of the jpg files located within the directory /jpg on the SD card, as demonstrated below.

```
///Initial the SD Card
int volumes_mounted;
sd_set_clock_to_max( 80000000 ); ////set spi clock divider coeff.
usleep (1000);
volumes_mounted = sd_fat_mount_all();////mount fat file system.
if( volumes_mounted <= 0 )
{
   printf( "SD Card Mount FAILED\n" );
}
else
{
   printf("SD Card Mount OK\n");
}
/////Read out the SD card specified subdirectory files
char * filelist_buff = (char *)malloc(32768);
memset(filelist_buff,0,32768);
int num_files = 0;
num_files = sd_list("/jpg/",filelist_buff);
num_files = num_files-2; ////subtract ". .."
alt_u32 i=0,j=0;
```

Following this, we can access each individual image by referencing its index in the filelist_buff. Subsequently, we pass the file pointer pointing to the respective image to the jpeg_decode function for additional processing.

```
int write_buffer(alt_video_display *display_global,char *pic_name,int frame_write_index)
{
     FILE * infile;                                                    //file pointer
     char * cp_pic_name = (char*)malloc(100);                          //allocate memory for image name
     sprintf(cp_pic_name,"/dev/sd_card_controller/jpg/%s",pic_name);   //obtain the image name
     infile = fopen(cp_pic_name,"rb");                                 //open the image file
     if (infile == NULL){
        printf("open file failed!\n");
        return -1;
     }
     unsigned int  *frame = (unsigned int *)display_global->buffer_ptrs[frame_write_index]->buffer;
     jpeg_decode(frame, infile, cp_pic_name, display_global);          //passing the image file pointer
     fclose(infile);
     free(cp_pic_name);
     return 0;
}
```

4.4 JPEG Decoding and Scaling

We'll utilize the API from the JPEG library integrated into our project to decode JPEG data into raw image data and perform scaling. The library supports scaling options of 1/2, 1/4, and 1/8 only. To determine the appropriate scaling for fitting the image onto the display panel, we calculate based on the image dimensions. Our scaling method is simple and less accurate; for instance, with 1/2 scaling, we discard every other row and column. Below is a representation of this approach.

```
(void) jpeg_start_decompress(&cinfo);
row_stride = cinfo.output_width * cinfo.output_components;
buffer = (*cinfo.mem->alloc_sarray)((j_common_ptr) &cinfo, JPOOL_IMAGE, row_stride, 1);


unsigned int my_buffer[480*800];
unsigned int my_buffer_1d[480*800];
unsigned int my_index = 0;
while (cinfo.output_scanline < cinfo.output_height) {
  // Read a scanline every time
  (void) jpeg_read_scanlines(&cinfo, buffer, 1);

  // Ultra-simple non-accurate scaling method: Periodically throw away one
  if(output_is_scaled) {
    if((rownum * output_scaled_height) < (outrow * jpeg_scaled_height)) {
      rownum++;
      continue;
    }
  }

  JDIMENSION num_cols = cinfo.output_width;
  outptr = *buffer;                                    //output row buffer (3 RGB)
```

4.5 Gray Scale

Gray scaling is achieved by computing the average of the RGB values for each pixel and assigning this average value to all three RGB components. The implementation of this process is provided below.

```
//Gray Scale
//Done by obtaining the average of the three colors RGB
//Then assign this average value to the Pixel
outptr[0] = ( outptr[0] + outptr[1] + outptr[2] )/3;                    //taking average of the 3 RGB values
outptr[1] = outptr[0];                                                  //assign the average value to each of the 3
outptr[2] = outptr[0];
//Saving the Gray scale filtered image into a Buffer
my_buffer[my_index] = ( (outptr[0] << 16) + (outptr[1] << 8) + outptr[2] );
my_buffer_1d[my_index] = outptr[0];
```

4.6 Edge Detection

For edge detection, we employed a basic symmetrical horizontal kernel featuring positive values of +1 and negative values of -1. Subsequently, we assess whether the convolution result for each value exceeds our threshold. If it does, we designate a white pixel; otherwise, a black pixel is assigned.

```
for(i = 1; i+1 < 480 ; i++) {
    for(j = 1; j+1 < 800; j++) {
        sum = 0;
                                //The Kernel in use is a simple Horizontal symmetrical Kernel

        sum = tempin[800*(i-1) + (j-1)]*(-1) +   tempin[800*(i-1) + (j)]*(-1) +    tempin[800*(i-1) + (j+1)]*(-1)

             +tempin[800*(i) + (j-1)]*  (0) +   tempin[800*(i) + (j)]*  (0) +    tempin[800*(i) + (j+1)]*  (0)

             +tempin[800*(i+1) + (j-1)]*(1) +   tempin[800*(i+1) + (j)]*(1) +    tempin[800*(i+1) + (j+1)]*(1);

        int temp = 0;
        temp = sum;
        // Thresholding:
        if(temp > 20) temp = 255;
        else temp = 0;

        * (frame+(i)*800+j) =
                    ( (temp << 16) + (temp << 8) + temp );        //Display the result of edge Detection to
                                                                  //the current pixel
```

4.7 Main Function

The main function is simply a file browser window displaying the names of 3 image files read from the SD card and the user can pick which picture to display using the touch screen.

```
void file_browser_window(alt_video_display *display_global)
{
       char text[50];
       int border_color = DODGERBLUE_24;

       vid_paint_block (0, 0, 799, 479, KHAKI_24, display_global);                                          //Paint Background
       sprintf(text, "%s%s", "FILE 1 : ", name_list[0] );
       vid_print_string_alpha(200, 62, YELLOW_24 , MEDIUMVIOLETRED_24, tahomabold_32, display_global,text);     //Display File 1 Name
       sprintf(text, "%s%s", "FILE 2 : ", name_list[1] );
       vid_print_string_alpha(200, 217, YELLOW_24 , MEDIUMVIOLETRED_24, tahomabold_32, display_global,text);    //Display File 2 Name
       sprintf(text, "%s%s", "FILE 3 : ", name_list[2] );
       vid_print_string_alpha(200, 372, YELLOW_24 , MEDIUMVIOLETRED_24, tahomabold_32, display_global,text);    //Display File 3 Name

       int i;
       for(i = 0; i < 10 ; i++)
       {
           vid_draw_line(1    , 140 + i , 800   , 140 + i , 1, border_color, display_global);                //Upper middle line
           vid_draw_line(1    , 295 + i , 800   , 295 + i , 1, border_color, display_global);                //Lower middle line
           vid_draw_line(1    , i       , 800   , i       , 1, border_color, display_global);                //Upper border
           vid_draw_line(1    , 470 + i , 800   , 470 + i , 1, border_color, display_global);                //Lower border
           vid_draw_line(i    , 0       , i     , 479     , 0, border_color, display_global);                //Left  border
           vid_draw_line(790+i, 0       , 790+i , 479     , 0, border_color, display_global);                //Right border
```

After displaying the image, users can activate grayscale mode by touching anywhere on the LCD display. Similarly, they can trigger edge detection by touching the screen. During the display of the edge detection filtered image, touching the screen will return users to the main file browser.

```
while(1)
{
    file_browser_window(display_global);
    Touch_EmptyFifo(hTouch);
    usleep(1000000);
    while(1)
    {
        if (!Touch_GetXY(hTouch, &x,&y))
        continue;
        printf("x=%d,y=%d\r\n",x,y);
        if ((x<=750 && x>=50)&& (y<=199 && y>=25)) //Trigger area upper File 1
          {
            write_buffer(display_global,name_list[0],display_global->buffer_being_written);
            break;
          }

        if ((x<=750 && x>=50)&& (y<=375 && y>=200)) //Trigger area middle File 2
          {
            write_buffer(display_global,name_list[1],display_global->buffer_being_written);
            break;
          }

        if ((x<=750 && x>=50)&& (y<=550 && y>=375)) //Trigger area lower File 3
          {
            write_buffer(display_global,name_list[2],display_global->buffer_being_written);
            break;
          }
    }

}
```

The Flow Chart of software is shown below in figure 4.2.

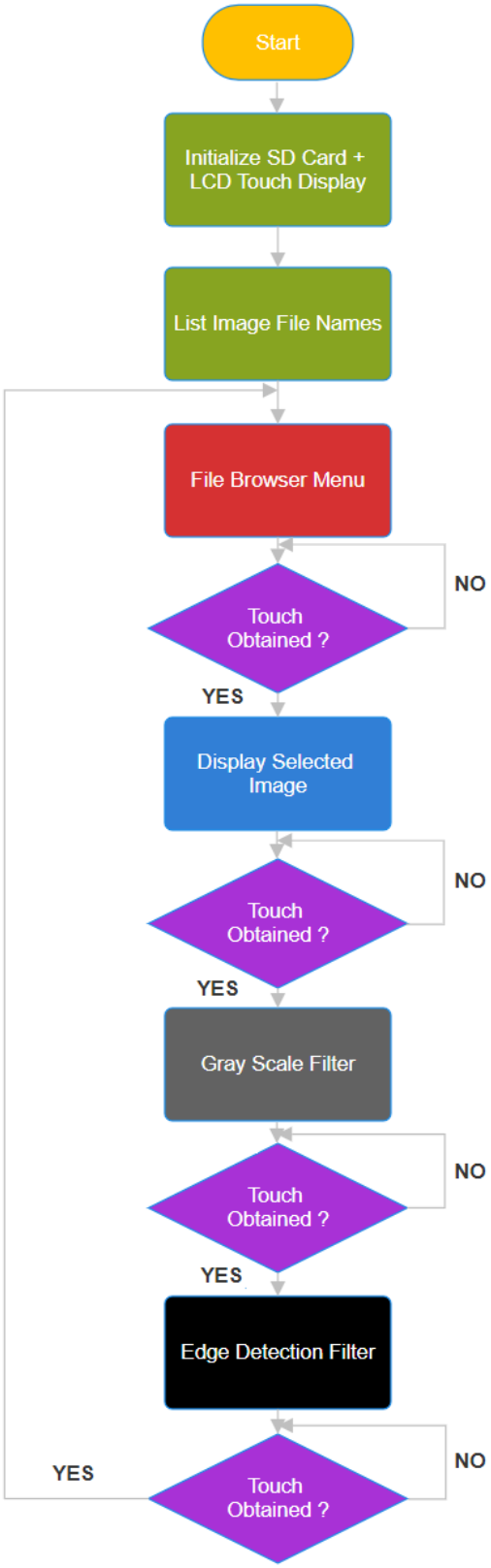


Figure 4.2 Main Function Flow Chart

5 Conclusion and Future Thoughts

In this project, we successfully implemented all required functionalities despite encountering numerous challenges along the way. Although our understanding of Verilog remained limited due to the predominant use of the Qsys platform, we gained valuable insights into the interplay between hardware and software development environments. Working with the LCD touch and display interface in C code proved both enjoyable and challenging.

For future enhancements, we propose expanding the file browser to accommodate a larger number of images by implementing multiple pages. Additionally, improving the quality of the edge detection mechanism could be achieved by incorporating a Gaussian filter beforehand.

Figure 5.1 Image Processor Project in Action

6 Figure directory

7 Refrences

[1] Terasic, ‘‘TRDB_LTM User Manual‘‘, http://www.terasic.com.tw/attachment/archive/237/TRDB_LTM_UserGuide_v1.22.pdf.

[2] Terasic, ‘‘DE2-115 User Manual‘‘, https://www.terasic.com.tw/attachment/archive/502/DE2_115_User_manual.pdf.

[3] Michael Barker, ‘‘IMAGE AND VIDEO PROCESSING ON ALTERA DE2-115 USING NIOS II SOFT-CORE PROCESSOR ‘‘, https://www.secs.oakland.edu/~ganesan/ece576f14project/index.htm.

[4] Xiang Zhou, “ Touch Pong CSEE 4840 Embedded System Design

Final Report“, https://www.cs.columbia.edu/~sedwards/classes/2012/4840/reports/Touch-Pong.pdf.

[5] Rodriego Frriera, “ FCUTC Digital System Design Final Project: Whack a Mole“, https://home.isr.uc.pt/~jfilipe/files/Report_Group7.pdf.

[6] Subramaniam Ganesan, Oakland University, “ Advanced course on Embedded Systems design using FPGA“, https://asee-ncs.org/proceedings/2010/papers/135.pdf.

[7] Zhe Lin, ‘‘Digital Image Processing in C (Chapter 4): Edge Detection, Laplacian, Sobel, Gamma Correction, and Histogram Equalization ‘‘, https://medium.com/@wilson.linzhe/digital-image-processing-in-c-chapter-4-edge-detection-and-grayscale-transformation-laplacian-dfb8de02f213.