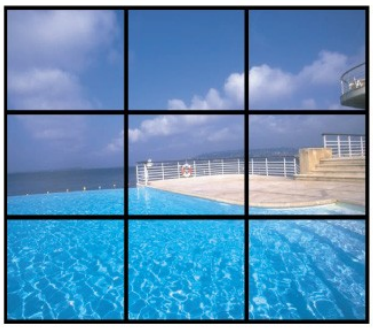
EE318 Electronic Design Lab
Spring 2011
Project Report

Group 3

Sarjan R. Satra - 08d07004
Lee Zhi Long - 10v051003
Lu Youkang - 10v071003

1. Project Aims



Video Wall

This project aims to design a device that can drive a video wall system using FPGA. A video wall is a system that consists of multiple computer monitors, video projectors, or television sets tiled together contiguously or overlapped in order to form one large screen.

For this project, we will be working on developing a video wall driver for a 4 monitor video wall tiled in a 2*2 manner. We will draw the video source (essentially images or frames) from the memory card of the FPGA, split the images, zoom them appropriately and output them to the respective monitor at the same time.

2. Design Approach

2.1 Hardware

The hardware involved in this project:

1. Virtex-5 FPGA kit (Xilinx ML505 Board)
2. Respective connectivity cables for the board (USB to JTAG for programming, USB to RS232 for serial communication)
3. Multi-card reader (For the Compact Flash drive)
4. Video output cables (VGA and DVI)
5. Monitors

Xilinx ML505 kit –

The kit has a Xilinx Virtex-5 FPGA. It also has 1 DVI output port and 1 VGA input port. Also the FPGA has a Microblaze soft processor which we have used in our design. A soft processor is a processor which has been completely implemented in the FPGA itself using logic synthesis.

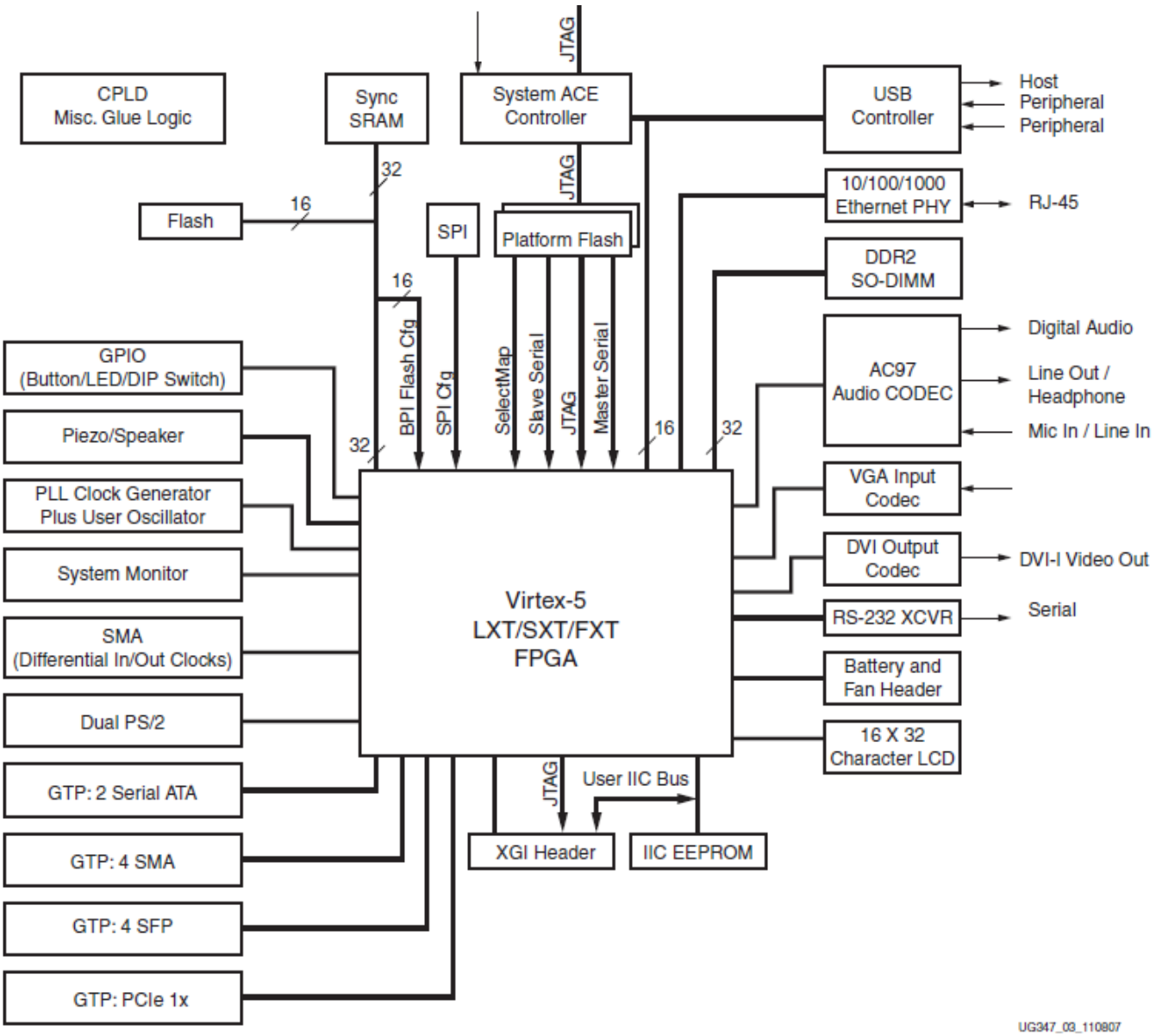


Figure 1-1: **Virtex-5 FPGA ML50x Evaluation Platform Block Diagram**

In order to interface the VGA port to the FPGA, there is a very fast ADC AD9980 which operates at data rates as fast as 95 MSPS (Mega Samples Per Second). To interface the VGA port we essentially have to use this ADC which converts the analog VGA signals to digital format suitable for processing in the FPGA.
Similarly to interface the DVI port we have a Chrontel CH7301C DVI transmitter which is a display controller device which accepts a digital graphics input signal, and encodes and transmits data through a DVI or DFP (Digital flat panel).
It has a pixel rate of upto 165 MHz. Hence to interface the DVI port we must use the Chrontel DVI controller.
The FPGA can be programmed using JTAG. There is a RS232 port which can be used to establish serial communication with a PC for debugging purposes.

2.2 Flow of Project

Since we haven't implemented a structure for a live video data stream, we use static images from the flash disk of the ML505 kit. These images can be thought of as a single frame of a video.
The static images are first read from the flash drive, split into multiple parts (4 here) as required and the then zoomed to match the screen size. Since the kit does not have multiple DVI output ports, we display only one of the parts (One quarter here) of the original image on the monitor.

2.3 Programming

The main part of this project is focused on the programming of the FPGA kit to output the video as intended. We have used some IP cores from the Xilinx website which help us interface the DVI port to the FPGA. The FPGA also has a Microblaze soft processor which is also used in the project for reading the file from the flash.

Hence the programming was done in C for the Microblaze processor and Verilog for the FPGA. We used Xilinx ISE and Xilinx EDK softwares (Version 11) for programming the FPGA kit.

2.4 Flow of Programming

3. Project Management

3.1 Goals

We have decided to break up our project into smaller and more manageable goals to aid in project management.
Also we changed our main goal after the Midsem evaluation. Our original goal was to take a live stream of video and then implement a 2*2 video wall driver for that. That proved to be very difficult given the less time.

Our new goal is to implement a video wall driver for input which is a stream of static images taken from the Compact Flash drive of the kit. Each of these images can be treated as a frame for the video and when run fast enough looks like a video.

3.2 Stage 1:
Our first task is to program the FPGA kit such that it can read a still image from the inbuilt memory card and display on the screen. This is to make sure that the output drivers are working and that a communication link has been established between the board and the memory card.

3.2 Stage 2:
Next, we aim to isolate a portion of the picture (1/4 in this case) and display it on the monitor. Additional programming needs to be done to expand the image by writing duplicates on different addresses to ensure that the still image is blown up and fills the screen.

3.3 Stage 3:
Finally, we will have to adjust the output frame rate such that it is capable of outputting a video onto the monitor. The frames must also be divided appropriately to display on all the screens simultaneously.

4. Design Implementation

4.1 Research
A lot of research was done while selecting the FPGA kit, as none of the FPGA kits in the department matched our requirements. Finally we settled for the ML505 kit through with which we could atleast test our concept.
The bulk of our time was spent researching on the capabilities of the ML505 board and the various protocols and mechanism it uses to output images using its DVI port.
The first thing that we learnt was programming the FPGA for a simple LED ON-OFF function. Later we focused our efforts on programming the VGA interface part. We left that part as it proved to be a mammoth task. This led us to altering our goals and taking still images from flash disk as input rather than live data stream from the VGA port.
We studied about DVI port interfacing and came across some IP cores for the same on the Xilinx website. This helped us to interface the DVI port rather easily.
Further research was also done on how to access the memory card via the board. In addition, research on the addressing system employed in the IP core was also done to ensure that we have the necessary knowledge to code the desired output.

4.2 IP Cores

The IP cores are a set of protocols that are needed to be loaded for the FPGA and the board to output an image to a screen via its DVI port. However, since the IP cores are generic, we will have to modify it and program it to receive our picture stills from the memory card and to crop the images and expand it fully on the screen.
We have used the XUPV5-LX110T std IP cores which contains cores for DVI interfacing.

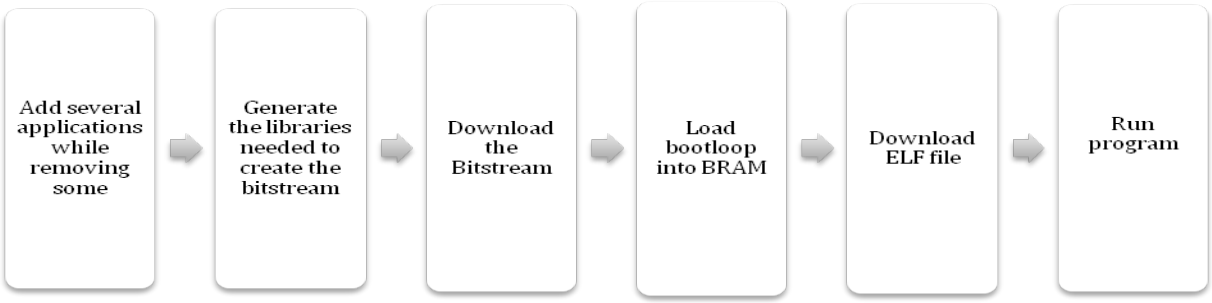
4.3 Programming the FPGA

Main steps for programming an FPGA :

- Write Verilog/VHDL code
- Check Syntax
- Synthesize
- Write a constraints file for mapping the FPGA pins to the appropriate peripherals (.ucf for pin constraints)
- Execute Place and Route
- Implement Design
- Generate a Programming File
- Configure the Target Device and program

For more details on this refer to Xilinx ISE 11 Indepth Tutorial on the Xilinx website.

4.4 Compile Design (Microblaze)



The basic operational steps for programming Microblaze are:

- Build All the user applications.
- Generate Bitstream
- Update the Bitstream
- Download the Bitstream into the FPGA
- Launch XMD Bash Shell
- Download ELF file
- Connect and Run

For more information on the specifics of the above please have a look at the XUPV5-LX110T std IP core's manual on the Xilinx website.

5.Test Procedure

No specific test procedure. Compile the design, execute the design on the FPGA kit and watch the output on the monitor.

6. The Journey/ Problems faced

The first two weeks were spent finding the best FPGA kit to match our requirements/specifications. At the same time we learnt the basics of programming an FPGA. We worked on Spartan-3 kit available in the WEL lab and tried to light an LED using a switch. Thereafter we moved on to the ML505 kit and run the basic LED on-off program in it. It has slightly different method of programing via JTAG and hence it took us a while to find out.

We started to develop an algorithm for interfacing the VGA port of the kit using the AD9980 ADC. We drew the timing diagrams according to the VESA specifications (Video Electronics Standards Association) and designed an algorithm. But implementing the same turned out to be a mammoth task and so we left that part.

We faced a lot of problems finding the correct licensed version of the Xilinx ISE and EDK softwares required for our project. This greatly hampered our progress in the first half of the semester.

Instead we focused our attention on the DVI interfacing. We modified our goals after the Midterm evaluation and instead of taking live video data stream from VGA, decided to use a stream of images from flash disk as our input. This enabled us to test our DVI interfacing part.

While trying to interface the DVI port, we came across some DVI IP cores on the Xilinx website. We implemented the same and this made our DVI interfacing job quite simple.
Finally we implemented our image splitting and zooming algorithms into the IP core to complete the modified goal.
Understanding the addressing used in the program and implementation of our image splitting and zooming algorithm also proved to be quite challenging as the program behaved in weird ways and we did not have any specific method for debugging except from trying out different things, executing them and drawing conclusions.
Many times the image used to be distorted, partial images were displayed, two frames would overlap, the one quarter of the image would turn to one-half or one-eight. Sometimes we even got images with only one/two of the RGB colours.

7. Future work and improvements

- The inherent ability of the development board to buffer data from the compact flash is restrained to about 96MB (which turns out to be 48 images of 2MB each), and if this can be improved, it is possible to output a slideshow or video of a longer period
- Instead of taking input (images and videos) from the compact flash, input can be taken from a DVI source (eg. a CPU with DVI port); appropriate hardware for the same is needed. It is then passed through the FPGA and output accordingly.
- Also right now we are showing the output (one quarter of a screen) on a single monitor due to hardware constraints. This can be improved and multiple monitors can be employed using a FPGA kit with multiple DVI output ports.
- Currently zooming is done just by duplicating pixels i.e. replicating the same pixel in 4 adjacent pixels (vertically and horizontally). This can be improved and interpolation can be used to improve the output picture quality.
- The controller algorithm can be made more general so as to accommodate for a NxM sized video wall while keeping the resolutions of all the monitors different. This is quite a challenging task.

8. Conclusion

Though we could not implement the data input part of our original goal due to various reasons (hardware constraints, software licensing problems and lack of time), we successfully completed the data output part and hence the modified goal.
We have also shown how a stream of images can act as a video while testing for the output (DVI interfacing part).

The data input part can be integrated with our project to implement a 2*2 video wall driver. Also additional improvements will improve the quality of the output data as well as generalize the idea to a NxM video wall.

Our project was quite challenging and none of us/faculty had worked on such a thing before. Hence we were on our own and we had to find our way out whenever we were stuck somewhere.
It was a very nice experience working on something challenging and we had a great time.

9. References

Datasheets : AD9980 ADC, Chrontel 7301C DVI Transmitter, Virtex-5 FPGA.

Guides : Xilinx 11 Tutorial, ML505 kit guide_ug347.

Books : Design Recipe for FPGA (Peter Wilson), FPGA Prototyping by Verilog Examples (Pong P. Chu).

Standards : VESA VGA standards, DDWG's DVI standards.

IP cores :

XUPV5-LX110T User Manual
http://www.xilinx.com/univ/xupv5-lx110T-manual.htm

XUPV5-LX110T Standard IP Design Adding PCores
http://www.xilinx.com/products/boards/xupv5/design_files_122/BSB_STD_IP_PCORES/xupv5-lx110t_std_ip_pcores.pdf

xupv5-lx110t_bsb_std_ip_overlay (ZIP)
http://www.xilinx.com/products/boards/xupv5/base-system-builder.htm

10. Program codes

```
/***************************** Include Files
*********************************/
#include "xio.h"
#include "sleep.h"
#include <sysace_stdio.h>
#include "xuartns550_l.h"
#include "xparameters.h"
#include "lcd.h"
#include "memory_map.h"
#ifdef PPC440CACHE
#include "xcache_l.h"
#endif


// Demo Parameters
#define IMAGE_BASEADDR  (DDR_BASEADDR + 0x0A000000)
#define IMAGE_MAXADDR   (IMAGE_BASEADDR + 0x06000000)
//Defines the Max no of images that can be stored... 48 in our case
/* structs for bitmaps */
typedef struct {
   unsigned short int type;              /* Magic identifier            */
   unsigned int size;                    /* File size in bytes          */
   unsigned short int reserved1, reserved2;
```

```
   unsigned int offset;                 /* Offset to image data, bytes */
} HEADER;

typedef struct {
   unsigned int size;              /* Header size in bytes      */
   int width,height;               /* Width and height of image */
   unsigned short int planes;      /* Number of color planes    */
   unsigned short int bits;        /* Bits per pixel            */
   unsigned int compression;       /* Compression type          */
   unsigned int imagesize;         /* Image size in bytes       */
   int xresolution,yresolution;    /* Pixels per meter          */
   unsigned int ncolors;           /* Number of colors          */
   unsigned int importantcolors;   /* Important colors          */
} INFOHEADER;


/* reads two bytes from file */
static unsigned short ReadUShort(SYSACE_FILE *fptr)

{
  unsigned char b0, b1;
  int numread;
  char readBuffer[2];
  numread = sysace_fread(readBuffer, 1, 2, fptr);
  b0 = readBuffer[0];
  b1 = readBuffer[1];
  return ((b1 << 8) | b0);
}

/* reads four bytes from file in little endian order */
static unsigned int ReadUIntLil(SYSACE_FILE *fptr)

{
  unsigned char b0, b1, b2, b3;
  int numread;
  char readBuffer[4];
  numread = sysace_fread(readBuffer, 1, 4, fptr);
  b0 = readBuffer[0];
  b1 = readBuffer[1];
  b2 = readBuffer[2];
  b3 = readBuffer[3];
  return ((((((b3 << 8) | b2) << 8) | b1) << 8) | b0);
}

/* opens and reads from CF a 640x480 bitmap file, placing it in
memory at baseaddr in a format for the tft */

int read_image(char FileName[], int baseaddr)
{
```

```
 unsigned char readBuffer[1920];
 SYSACE_FILE *infile;
 int i, j, numread, temp, writeaddr;
 HEADER header;
 INFOHEADER infoheader;
 infile = sysace_fopen(FileName, "r");
 if(infile) {
   xil_printf("Reading file : %s\n\r", FileName);
   LCDPrintString ("Loading Slide:  ", &FileName[3]);
   /* Read and check the header */
   header.type = ReadUShort(infile);
   header.size = ReadUIntLil(infile);
   header.reserved1 = ReadUShort(infile);
   header.reserved2 = ReadUShort(infile);
   header.offset = ReadUIntLil(infile);
   infoheader.size = ReadUIntLil(infile);
   infoheader.width = ReadUIntLil(infile);
   infoheader.height = ReadUIntLil(infile);
   infoheader.planes = ReadUShort(infile);
   infoheader.bits = ReadUShort(infile);
   infoheader.compression = ReadUIntLil(infile);
   infoheader.imagesize = ReadUIntLil(infile);
   infoheader.xresolution = ReadUIntLil(infile);
   infoheader.yresolution = ReadUIntLil(infile);
   infoheader.ncolors = ReadUIntLil(infile);
   infoheader.importantcolors = ReadUIntLil(infile);

 /* Process the data */

 //-------------------------------------------------
   //LEFT BOTTOM CORNER

   /*for (j=(((infoheader.height)/2)-1);j>=0;j--) {
     numread = sysace_fread(readBuffer, 1, 1920, infile);
     for (i=0;i<((infoheader.width)/2);i++) {

     temp = ((((readBuffer[(i*3)+2] << 8) | readBuffer[(i*3)+1]) <<
8) | readBuffer[(i*3)]);

      writeaddr = baseaddr+(j*1024+i)*8;
       XIo_Out32(writeaddr, temp);
       XIo_Out32(writeaddr+4, temp);
       XIo_Out32(writeaddr+512*8, temp);
       XIo_Out32(writeaddr+512*8+4, temp);
     }
   }*/
```

```
   //-----------------------------------------------------
   //LEFT TOP CORNER
   /*for (j=((infoheader.height)-1);j>=0;j--) {
     numread = sysace_fread(readBuffer, 1, 1920, infile);
     for (i=0;i<((infoheader.width)/2);i++) {

       temp = ((((readBuffer[(i*3)+2] << 8) | readBuffer[(i*3)+1]) <<
8) | readBuffer[(i*3)]);

      writeaddr = baseaddr+(j*1024+i)*8 - (((infoheader.height)/2)-
1)*1024*8 ;

       XIo_Out32(writeaddr, temp);
       XIo_Out32(writeaddr+4, temp);
       XIo_Out32(writeaddr+512*8, temp);
       XIo_Out32(writeaddr+512*8+4, temp);
     }
   } */

  //-----------------------------------------------------

   //RIGHT BOTTOM CORNER
   /*for (j=(((infoheader.height)/2)-1);j>=0;j--) {
     numread = sysace_fread(readBuffer, 1, 1920, infile);
     for (i=((infoheader.width)/2);i<(infoheader.width);i++) {

      temp = ((((readBuffer[(i*3)+2] << 8) | readBuffer[(i*3)+1]) <<
8) | readBuffer[(i*3)]);

       writeaddr = baseaddr+(j*1024+i)*8 -((infoheader.width)/2)*8;

       XIo_Out32(writeaddr, temp);
       XIo_Out32(writeaddr+4, temp);
       XIo_Out32(writeaddr+512*8, temp);
       XIo_Out32(writeaddr+512*8+4, temp);
     }
   } */

   //-----------------------------------------------------
   //RIGHT TOP CORNER

   for (j=((infoheader.height)-1);j>=0;j--) {
     numread = sysace_fread(readBuffer, 1, 1920, infile);
     for (i=((infoheader.width)/2);i<(infoheader.width);i++) {

       temp = ((((readBuffer[(i*3)+2] << 8) | readBuffer[(i*3)+1]) <<
8) | readBuffer[(i*3)]);
```

```
        writeaddr = baseaddr+(j*1024+i)*8 -((infoheader.width)/2)*8;

        XIo_Out32(writeaddr, temp);
        XIo_Out32(writeaddr+4, temp);
        XIo_Out32(writeaddr+512*8, temp);
        XIo_Out32(writeaddr+512*8+4, temp);
      }
    }

    //-----------------------------------------------------

    //ORIGINAL
    /*for (j=(((infoheader.height)-1));j>=0;j--) {
      numread = sysace_fread(readBuffer, 1, 1920, infile);

    for (i=0;i<((infoheader.width));i++) {

        temp = ((((readBuffer[(i*3)+2] << 8) | readBuffer[(i*3)+1]) <<
8) | readBuffer[(i*3)]);

        writeaddr = baseaddr+(j*1024+i)*4;
        XIo_Out32(writeaddr, temp);
      }
    }*/

    sysace_fclose(infile);
    return 1;
  } else {
    return 0;
  }
}


/* reads a series of images from CF: "image01.bmp",
"image02.bmp", "image03.bmp", ... */
/* stores each in memory on consecutive 2MB boundaries starting at
IMAGE_BASEADDR up to IMAGE_MAXADDR */
int get_images()
{
 char file[] = "a:\\image01.bmp";
  int baseaddr = IMAGE_BASEADDR;
  int count = 0;
  while(baseaddr < IMAGE_MAXADDR && read_image(file,
baseaddr)) {
    if (file[9] == '9') {
      file[8]++;
      file[9] -= 9;
    } else {
```

```
      file[9]++;
    }
    baseaddr += 0x200000;
    count++;
  }
  return count;
}


/* infinite loop that fills fifos and handles button pushing */

void play_video(Xuint32 count) {
  unsigned int j;
  unsigned int temp;
  unsigned int tftptr;
  Xboolean button_n, button_e, button_s, button_w, button_c;
  Xboolean past_button_n, past_button_e, past_button_s,
past_button_w, past_button_c;
  Xboolean paused = 0;
  tftptr = IMAGE_BASEADDR;
  past_button_n = 0;
  past_button_e = 0;
  past_button_s = 0;
  past_button_w = 0;
  past_button_c = 0;
  j = 0;
  while (1) {
    //timing for slideshow change
    j += 1024;
    // auto advance tft
    if ((j % 0x20000000 == 0) && (!paused)) {
      if (tftptr == IMAGE_BASEADDR+0x200000*(count-1))
      tftptr = IMAGE_BASEADDR;
        else tftptr += 0x200000;
        XIo_Out32(TFT_BASEADDR, tftptr);
        //XIo_Out32(TFT_BASEADDR, tftptr+4);
    }


    // check for button pushes
    if (j % 0x10000 == 0) {
      temp = XIo_In32(PUSHB_CWSEN_BASEADDR);
      button_n = (temp >> 0) & (0x00000001);
      button_e = (temp >> 1) & (0x00000001);
      button_s = (temp >> 2) & (0x00000001);
      button_w = (temp >> 3) & (0x00000001);
      button_c = (temp >> 4) & (0x00000001);
      if (button_e && !past_button_e) {
```

```
            if (tftptr == IMAGE_BASEADDR+0x200000*(count-1))
tftptr = IMAGE_BASEADDR;
       else tftptr += 0x200000;
       XIo_Out32(TFT_BASEADDR, tftptr);
      }
      if (button_w && !past_button_w) {
            if (tftptr == IMAGE_BASEADDR) tftptr =
IMAGE_BASEADDR+0x200000*(count-1);
       else tftptr -= 0x200000;
       XIo_Out32(TFT_BASEADDR, tftptr);
      }
      if (button_c && !past_button_c) {
           paused = !paused;
      if (paused) LCDPrintString ("Slideshow Paused", "Resume=
Button C");
      else LCDPrintString ("Slideshow Ready ", "VGA Active      ");
      }
      past_button_n = button_n;
      past_button_e = button_e;
      past_button_s = button_s;
      past_button_w = button_w;
      past_button_c = button_c;
    }
  }
}


int main()
{
int count;
  #ifdef PPC440CACHE
  XCache_EnableICache(PPC440_ICACHE);
  XCache_EnableDCache(PPC440_DCACHE);
  #endif
  XUartNs550_SetBaud(UART_BASEADDR, UART_CLOCK,
UART_BAUDRATE);
  XUartNs550_mSetLineControlReg(UART_BASEADDR,
XUN_LCR_8_DATA_BITS);

  /* initialize GPIO */
  //XIo_Out32(PUSHB_CWSEN_BASEADDR, 0x00000000);
  //XIo_Out32(GPIO_CONTROL, 0xFFFFFE00);

  /* print directions to LCD screen */
  LCDOn();
  LCDInit();

  /* get images */
  print("\nProgram running.\n\r");
```

```
  count = get_images();
  print("\n\rReads done\n\r");
  LCDPrintString ("Slideshow Ready ", "VGA Active      ");

  /* set TFT pointer */
  XIo_Out32(TFT_BASEADDR, IMAGE_BASEADDR);

  /* initialize and play video */
  play_video(count); // infinite loop
  #ifdef PPC440CACHE
  XCache_DisableDCache();
  XCache_DisableICache();
  #endif
  return 0;
}
```

Thank You