CECS 461/561: Hardware/Software Design

Spring 2021

Thomas Pridy | Monil Shah

Final System Report

(Tic Tac Toe Game)

Contents

I - Introduction **3**

II - Operation **5**

Figure 1: Example Project Blended Color Pattern 7

Figure 2: Example Project Serial Output 7

Figure 3: TicTacToe HDMI Output 8

Figure 4: TicTacToe Serial Output 8

III - Theory **10**

IV - Hardware Source Code **11**

Figure 5: Hardware Block Diagram 11

V - Software Source Code **12**

tictactoe.c 12

VI - Conclusion **24**

VII - Appendix **25**

References 25

I - Introduction

The Final Project of the CECS 561 Hardware/Software co-design course consisted of two phases. For this project, we are using the Zybo Z7-10 Zynq-7000 and some external peripherals such as a monitor. The goal of this project was to create a tic-tac-toe game using the HDMI as the video output to the monitor screen while providing the user with the ability to use switches and buttons as the input to navigate and select the boxes on the Tic Tac Toe game. Instead of displaying the 'O' and 'X', we are using the solid color to display each players' marker for the game. (we used Red and Green Marker).

We are going to add hardware components which are HDMI, UART, GPIO Buttons, SD card reader and the GPIO dip switches. Moreover, the output GUI will be displayed on the monitor screen connected through HDMI to the Zybo. Additionally, our base system project goal is to interact through the keyboard to move the position of the grid in the Tic Tac Toe game on the monitor screen. Whereas, the Advanced system project goal is to design Tic Tac Toe game using the input from switches and push buttons. The direction of moving left, right, up or down in 3 X 3 grid will be controlled by the push button and the movement for the tile in grid will be shown by the red border tile. Once the red border tile is decided to place on the selected position, we can confirm it using a dip switch as it puts the blue maker on the tile. On the other hand, the computer chooses the tile using a random algorithm that is a random generator to select the place of tile on a game by a bluemarker. Once the game is done, it clears the grid to resume the game. Furthermore, a message will be displayed on the computer screen regarding the game result. This has been done by the serial communication using UART and zybo board as they are connected through the micro-usb cable. Section II will cover more detail about the different

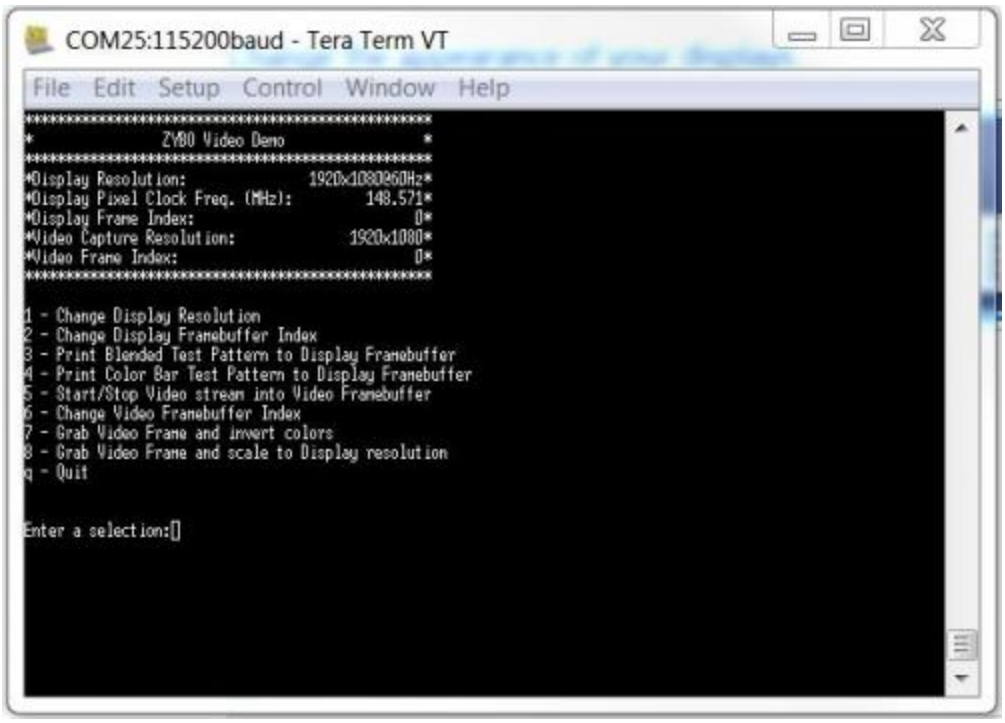
operations that we have performed with these hardware components. Furthermore, section III will discuss the theory of each component. Section IV and section V will give us an idea about the hardware design that is the block diagram of the whole project and software design that needs to be run on the application project. Source code of the software is mentioned under this section V. Finally, section VI gives the brief conclusion about our project and the key problems that we have faced during the implementation of the project.

II - Operation

Before getting started it was important to verify we could get HDMI output on the hardware in the example project and build our system from there. When creating and launching the example project, we found that it was created in 2018.2 so when we opened it with 2018.3 we had to upgrade some IP's that the system was using which meant opening and running Synthesis on the project. These IP's will automatically get flagged as being out-of-date and give you the option to upgrade them. Once you select upgrade, Vivado will automatically upgrade the IP's to the newer version and you will have a successful 2018.3 version of the HDMI demo project. Next, we generated the bitstream and exported it to the SDK so that we could run the example software project. We were able to run the software project and get demo patterns on the display verifying that this project could be the baseline for our hardware for our final project. Figure 1 and 2 show this example project output:

Figure 1: Example Project Blended Color Pattern

Figure 2: Example Project Serial Output



After we had the example project working, all we would have to do is trim the hardware design to only what we needed (because video capture was included in the example project), remove these hardware components from the xdc file, and then modify the software design to run tic tac toe instead of the example project. In addition, we also have to add the GPIO instances for the buttons and switches for inputs to the software application. This included running Synthesis and Implementation on the new hardware design and exporting a bitstream over to the SDK. Once we had this bit stream we would create a new board support package based on the new hardware definition file and create a new application project to use this board support package. We then imported the source code and modified it to draw a tic tac toe board as well as a highlight box and markers based on the input from buttons and switches from the user. We added into the software that the value of the switches changes the functionality of the push buttons so that the user would be able to achieve multiple things with only four push buttons. Including navigation, marker selection, and exiting the game. Figure 3 and 4 show the current state of this software:

Figure 3: TicTacToe HDMI Output

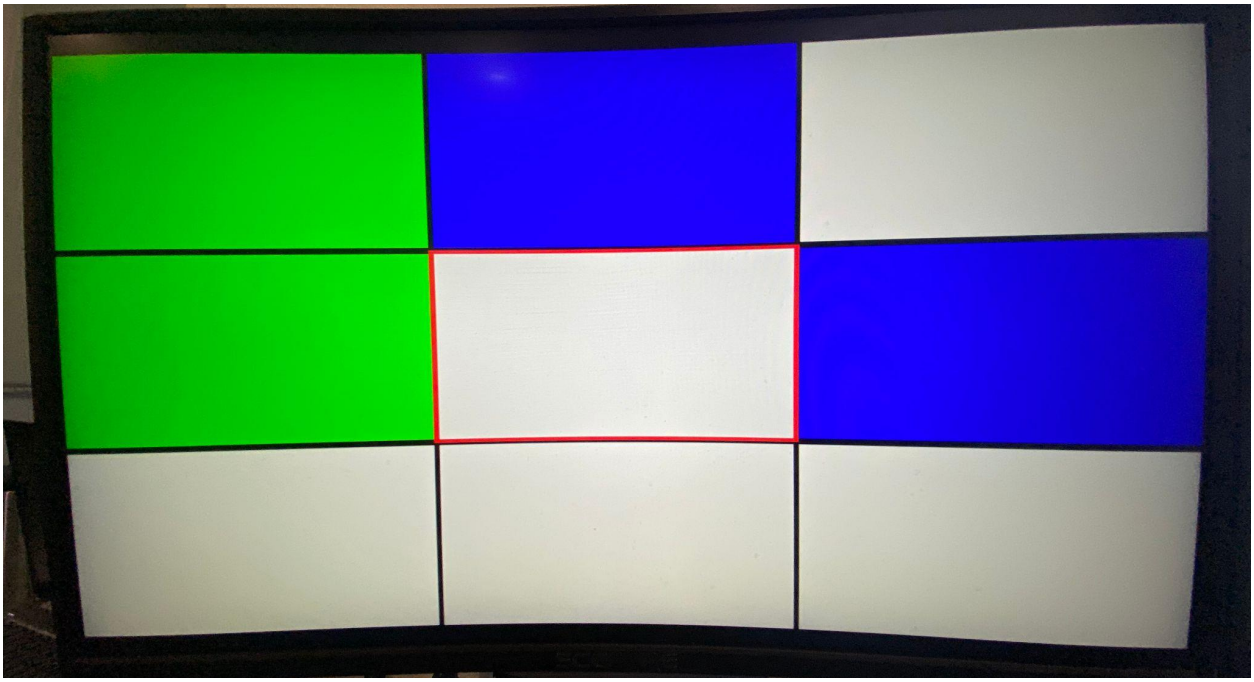
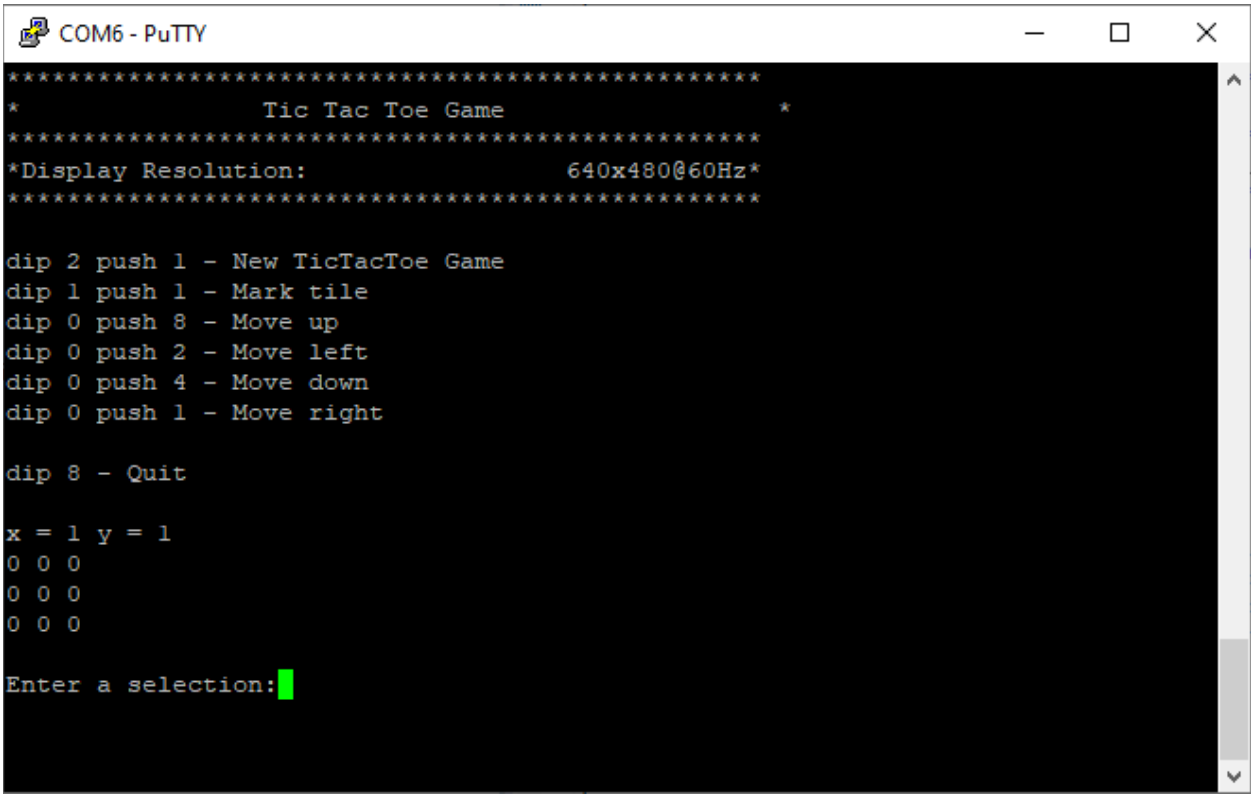


Figure 4: TicTacToe Serial Output



We then created a bootable image of our software application so that we can load the software to an SD Card. When the Zybo board is set to SD Card boot, the processor will grab the boot.bin from the SD Card, program the FPGA with the bit file, and then launch our software application. This way we don't need to use the SDK to program the FPGA and run our application but instead just flip on the switch. You can see a demonstration of our final project https://drive.google.com/file/d/1pq51e9kGKxrXqIf1N4qNy0fwq8xq0T_m/view?usp=sharing.

III - Theory

In this project, we are using many new IP's that were not used during class or during labs this semester. The most important of which are the AXI Video Direct Memory Access, the Video Timing Controller, and the AXI4-Stream to Video Out. The AXI Video direct Memory Access is the most important because it allows us to store the data of the pixel to memory, and then memory to output with minimal processor intervention. The AXI-lite bus allows the processor to communicate with the AXI VDMA for the data transfers. The AXI_MM2S and AXI_S2MM are memory-mapped buses that provide VMDA access to the DDR. These buses are also AXI4-streaming buses, which allow for a continuous stream of data without addresses.

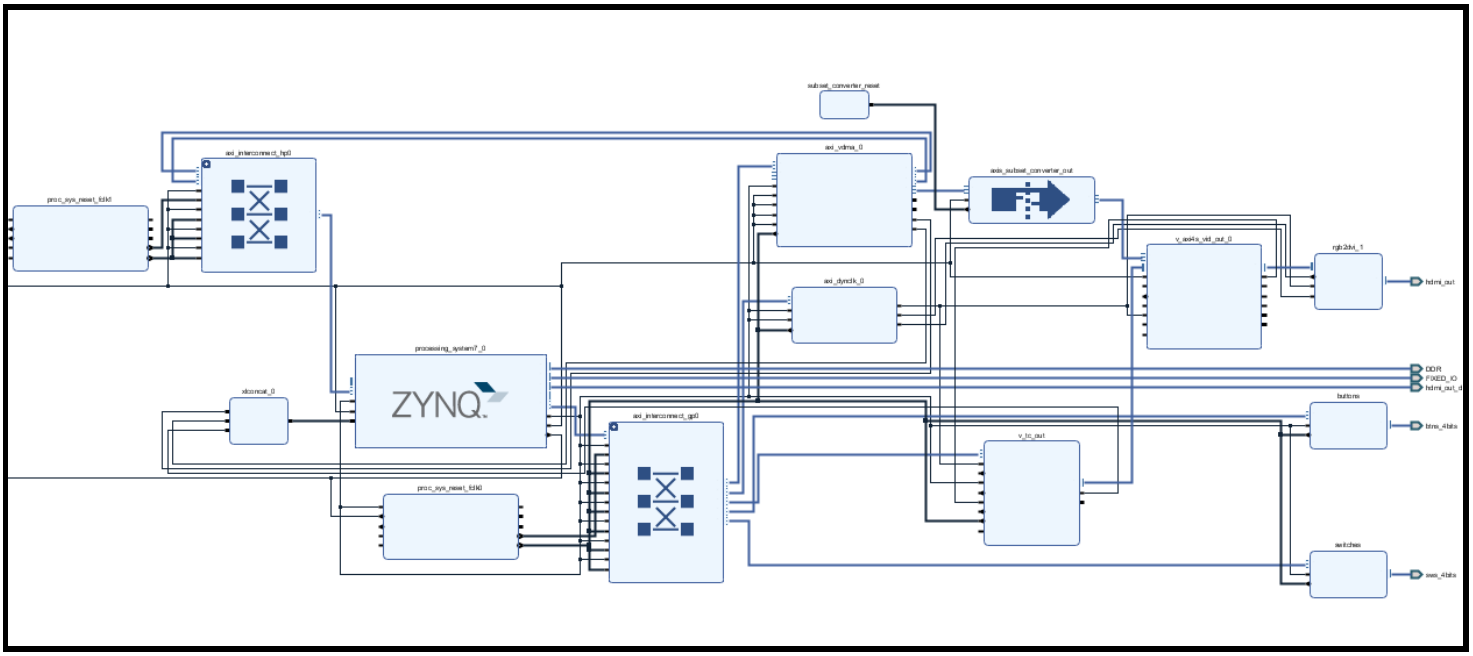
The next IP I would like to mention is the Video Timing Controller that controls the timing of the AXI4-Stream to Video Output IP. This module is controlled through the AXI Interconnect and generates the video timing signals and allows for adjustment of timing within a video design. This in conjunction with the Dynamic Clock Generator generates and controls all clocks required for outputting the video signals from the AXI VDMA to the HDMI output.

The Last IP to discuss is the AXI4-Stream to Video Out. The AXI4-Stream to Video Out core converts an AXI4-Stream Slave interface that conforms with the AXI4-Stream Video protocol to a video output, consisting of parallel video data, video syncs, blanks, and data valid. This module provides the RGB data from the VDMA and then passes the signals to the RGB to DVI encoder that will output the HDMI signals to the TX HDMI peripheral in our design.

On the software side, and mentioned in the Operation section of this document, we use the DDR memory for stack and heap for the variables and code running for tic-tac-toe. This isn't a huge program so DDR is the most efficient interface.

IV - Hardware Source Code

Figure 5: Hardware Block Diagram



https://drive.google.com/file/d/1z_aPHZeh3WJvFqRrwuFbyv5iVKOF4vO9/view?usp=sharing

The block diagram is a little hard to read so I have provided a link to the pdf output. This block diagram includes the HDMI output and serial output that is used in our current design. In addition, there are two instances of the GPIO IP so that we can read input from the buttons and switches as we use this as user input to play the tic tac toe game. Other than that, this hardware is booted and programmed by the SD Card through a boot.bin that we generated so that we wouldn't need to use the SDK to launch our project.

V - Software Source Code

The software project is similar to the basic system project, but now the input from the user is coming from the buttons and push buttons rather than the UART and keyboard input. We are using similar functions to draw the tic-tac-toe board, marker highlighter, and markers onto the HDMI screen. For now this is the source code we have been using: (*Note: this is incomplete and is subject to change as we near the final version of our projec*t)

tictactoe.c

```
/*****************************************************************************/
/*                                                                           */
/*      tictactoe.c    --      Tic Tac Toe    on Zybo over HDMI
*/
/*                                                                           */
/*****************************************************************************/
/*      Author: Thomas Pridy
              */
/*****************************************************************************/
/*  Module Description:
              */
/*                                                                           */
/*             This file contains code for running the game tictactoe on the   */
/*      Zybo board over HDMI.
              */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/

/* ------------------------------------------------------------ */
/*                          Include File Definitions
*/
/* ------------------------------------------------------------ */

#include "display_ctrl/display_ctrl.h"
#include "intc/intc.h"
#include <stdio.h>
#include "xuartps.h"
#include "math.h"
#include <ctype.h>
#include <stdlib.h>
#include "xil_types.h"
#include "xil_cache.h"
#include "timer_ps/timer_ps.h"
#include "xparameters.h"
#include "tictactoe.h"

/*
 * XPAR redefines
```

```
 */
#define DYNCLK_BASEADDR             XPAR_AXI_DYNCLK_0_BASEADDR
#define VDMA_ID                             XPAR_AXIVDMA_0_DEVICE_ID
#define HDMI_OUT_VTC_ID             XPAR_V_TC_OUT_DEVICE_ID
#define SCU_TIMER_ID                XPAR_SCUTIMER_DEVICE_ID
#define UART_BASEADDR                       XPAR_PS7_UART_1_BASEADDR

/* ------------------------------------------------------------ */
/*                          Global Variables
       */
/* ------------------------------------------------------------ */
/*
 * Display and Video Driver structs
 */
DisplayCtrl dispCtrl;
XAxiVdma vdma;
INTC intc;
char fRefresh; //flag used to trigger a refresh of the Menu on video detect
int game_x = 1;
int game_y = 1;
int marker = GREEN;
int win = 0;
int turns = 0;
int board[3][3];

/*
 * Framebuffers for video data
 */
u8 frameBuf[DISPLAY_NUM_FRAMES][DEMO_MAX_FRAME] __attribute__((aligned(0x20)));
u8 *pFrames[DISPLAY_NUM_FRAMES]; //array of pointers to the frame buffers


XGpio dip, push;

/* ------------------------------------------------------------ */
/*                          Procedure Definitions
*/
/* ------------------------------------------------------------ */

int main(void)
{
       Display_Initialize();
       resetBoard();

       xil_printf("-- Start of the Program --\r\n");

       XGpio_Initialize(&dip, XPAR_SWITCHES_DEVICE_ID);
       XGpio_SetDataDirection(&dip, 1, 0xffffffff);

       XGpio_Initialize(&push, XPAR_BUTTONS_DEVICE_ID);
       XGpio_SetDataDirection(&push, 1, 0xffffffff);

       GameRun();

       return 0;
}


void Display_Initialize()
{
       int Status;
       XAxiVdma_Config *vdmaConfig;
       int i;

       /*
        * Initialize an array of pointers to the 3 frame buffers
```

```
	 */
	for (i = 0; i < DISPLAY_NUM_FRAMES; i++)
	{
		pFrames[i] = frameBuf[i];
	}

	/*
	 * Initialize a timer used for a simple delay
	 */
	TimerInitialize(SCU_TIMER_ID);

	/*
	 * Initialize VDMA driver
	 */
	vdmaConfig = XAxiVdma_LookupConfig(VDMA_ID);
	if (!vdmaConfig)
	{
		xil_printf("No video DMA found for ID %d\r\n", VDMA_ID);
		return;
	}
	Status = XAxiVdma_CfgInitialize(&vdma, vdmaConfig, vdmaConfig->BaseAddress);
	if (Status != XST_SUCCESS)
	{
		xil_printf("VDMA Configuration Initialization failed %d\r\n", Status);
		return;
	}

	/*
	 * Initialize the Display controller and start it
	 */
	Status = DisplayInitialize(&dispCtrl, &vdma, HDMI_OUT_VTC_ID, DYNCLK_BASEADDR, pFrames,
DEMO_STRIDE);
	if (Status != XST_SUCCESS)
	{
		xil_printf("Display Ctrl initialization failed during demo
initialization%d\r\n", Status);
		return;
	}
	Status = DisplayStart(&dispCtrl);
	if (Status != XST_SUCCESS)
	{
		xil_printf("Couldn't start display during demo initialization%d\r\n", Status);
		return;
	}

	PrintPattern(dispCtrl.framePtr[dispCtrl.curFrame], dispCtrl.vMode.width,
dispCtrl.vMode.height, dispCtrl.stride, TIC_TAC_TOE);
	PrintPattern(dispCtrl.framePtr[dispCtrl.curFrame], dispCtrl.vMode.width,
dispCtrl.vMode.height, dispCtrl.stride, BOX);

	return;
}

void GameRun()
{
	int psb_check = 0;
	int dip_check = 0;
	PrintMenu();

	while (dip_check != 8)
	{
		fRefresh = 0;

		psb_check = XGpio_DiscreteRead(&push, 1);
		dip_check = XGpio_DiscreteRead(&dip, 1);
```

```
		if (marker == BLUE)
		{
			cpuTurn();
		}
		if (dip_check == 0)
		{
			switch(psb_check)
			{
			case 8: //Up
				if ((game_y != 1))
				{
					game_y -= 1;
					PrintPattern(pFrames[dispCtrl.curFrame],
dispCtrl.vMode.width, dispCtrl.vMode.height, DEMO_STRIDE, TIC_TAC_TOE);
					PrintPattern(pFrames[dispCtrl.curFrame],
dispCtrl.vMode.width, dispCtrl.vMode.height, DEMO_STRIDE, BOX);
				}
				break;
			case 2:
				if (game_x != 1)
				{
					game_x -= 1;
					PrintPattern(pFrames[dispCtrl.curFrame],
dispCtrl.vMode.width, dispCtrl.vMode.height, DEMO_STRIDE, TIC_TAC_TOE);
					PrintPattern(pFrames[dispCtrl.curFrame],
dispCtrl.vMode.width, dispCtrl.vMode.height, DEMO_STRIDE, BOX);
				}
				break;
			case 4:
				if ((game_y != 3))
				{
					game_y += 1;
					PrintPattern(pFrames[dispCtrl.curFrame],
dispCtrl.vMode.width, dispCtrl.vMode.height, DEMO_STRIDE, TIC_TAC_TOE);
					PrintPattern(pFrames[dispCtrl.curFrame],
dispCtrl.vMode.width, dispCtrl.vMode.height, DEMO_STRIDE, BOX);
				}
				break;
			case 1:
				if (game_x != 3)
				{
					game_x += 1;
					PrintPattern(pFrames[dispCtrl.curFrame],
dispCtrl.vMode.width, dispCtrl.vMode.height, DEMO_STRIDE, TIC_TAC_TOE);
					PrintPattern(pFrames[dispCtrl.curFrame],
dispCtrl.vMode.width, dispCtrl.vMode.height, DEMO_STRIDE, BOX);
				}
				break;
			}
		}
		else if (dip_check == 1)
		{
			switch(psb_check)
			{
			case 1:
				if ((board[game_x-1][game_y-1] == NONE))
				{
					board[game_x-1][game_y-1] = marker;
					PrintPattern(pFrames[dispCtrl.curFrame],
dispCtrl.vMode.width, dispCtrl.vMode.height, DEMO_STRIDE, TIC_TAC_TOE);
					PrintPattern(pFrames[dispCtrl.curFrame],
dispCtrl.vMode.width, dispCtrl.vMode.height, DEMO_STRIDE, BOX);
					turns++;
					checkForWin(marker);
					//If win, reset game. Otherwise change turns
					if (win == 1)
```

```
                                        {
                                                game_x = 1;
                                                game_y = 1;
                                                resetBoard();
                                                xil_printf("\x1B[H"); //Set cursor to top left of
terminal
                                                xil_printf("\x1B[2J"); //Clear terminal
                                                switch(marker)
                                                {
                                                case GREEN:
                                                        xil_printf("Congratulations! GREEN
won!\n\r");
                                                        xil_printf("Resetting...\n\r");
                                                        break;
                                                case BLUE:
                                                        xil_printf("Congratulations! BLUE
won!\n\r");
                                                        xil_printf("Resetting...\n\r");
                                                        break;
                                                }
                                                marker = GREEN;
                                                win = 0;
                                                turns = 0;
                                                TimerDelay(1000000);
                                                PrintPattern(pFrames[dispCtrl.curFrame],
dispCtrl.vMode.width, dispCtrl.vMode.height, DEMO_STRIDE, TIC_TAC_TOE);
                                                PrintPattern(pFrames[dispCtrl.curFrame],
dispCtrl.vMode.width, dispCtrl.vMode.height, DEMO_STRIDE, BOX);
                                        }
                                        else if (turns >= 9)
                                        {
                                                turns = 0;
                                                game_x = 1;
                                                game_y = 1;
                                                marker = GREEN;
                                                resetBoard();
                                                xil_printf("\x1B[H"); //Set cursor to top left of
terminal
                                                xil_printf("\x1B[2J"); //Clear terminal
                                                xil_printf("Stalemate!\n\r");
                                                xil_printf("Resetting...\n\r");
                                                TimerDelay(1000000);
                                                PrintPattern(pFrames[dispCtrl.curFrame],
dispCtrl.vMode.width, dispCtrl.vMode.height, DEMO_STRIDE, TIC_TAC_TOE);
                                                PrintPattern(pFrames[dispCtrl.curFrame],
dispCtrl.vMode.width, dispCtrl.vMode.height, DEMO_STRIDE, BOX);
                                        }
                                        else
                                        {
                                                if (marker == GREEN) marker = BLUE;
                                                else marker = GREEN;
                                        }
                                        PrintMenu();
                                }
                                break;
                        }
                }
                else if (dip_check == 2)
                {
                        switch(psb_check)
                        {
                        case 1:
                                game_x = 1;
                                game_y = 1;
                                resetBoard();
                                marker = GREEN;
```

```
                                PrintPattern(pFrames[dispCtrl.curFrame], dispCtrl.vMode.width,
dispCtrl.vMode.height, DEMO_STRIDE, TIC_TAC_TOE);
                                PrintPattern(pFrames[dispCtrl.curFrame], dispCtrl.vMode.width,
dispCtrl.vMode.height, DEMO_STRIDE, BOX);
                                break;
                        }
                }

        }

        return;
}

void PrintMenu()
{
        xil_printf("\x1B[H"); //Set cursor to top left of terminal
        xil_printf("\x1B[2J"); //Clear terminal
        xil_printf("**************************************************\n\r");
        xil_printf("*                Tic Tac Toe Game                *\n\r");
        xil_printf("**************************************************\n\r");
        xil_printf("*Display Resolution: %28s*\n\r", dispCtrl.vMode.label);
        xil_printf("**************************************************\n\r");
        xil_printf("\n\r");
        xil_printf("dip 2 push 1 - New TicTacToe Game\n\r");
        xil_printf("dip 1 push 1 - Mark tile\n\r");
        xil_printf("dip 0 push 8 - Move up\n\r");
        xil_printf("dip 0 push 2 - Move left\n\r");
        xil_printf("dip 0 push 4 - Move down\n\r");
        xil_printf("dip 0 push 1 - Move right\n\r");
        xil_printf("\n\r");
        xil_printf("dip 8 - Quit\n\r");
        xil_printf("\n\r");
        xil_printf("x = %d ",game_x);
        xil_printf("y = %d \n\r",game_y);
        xil_printf("%d %d %d\n\r",board[0][0],board[1][0],board[2][0]);
        xil_printf("%d %d %d\n\r",board[0][1],board[1][1],board[2][1]);
        xil_printf("%d %d %d\n\r",board[0][2],board[1][2],board[2][2]);
        xil_printf("\n\r");
        xil_printf("Enter a selection:");
}

void ChangeRes()
{
        int fResSet = 0;
        int status;
        char userInput = 0;

        /* Flush UART FIFO */
        while (XUartPs_IsReceiveData(UART_BASEADDR))
        {
                XUartPs_ReadReg(UART_BASEADDR, XUARTPS_FIFO_OFFSET);
        }

        while (!fResSet)
        {
                CRMenu();

                /* Wait for data on UART */
                while (!XUartPs_IsReceiveData(UART_BASEADDR))
                {}

                /* Store the first character in the UART recieve FIFO and echo it */
                userInput = XUartPs_ReadReg(UART_BASEADDR, XUARTPS_FIFO_OFFSET);
                xil_printf("%c", userInput);
                status = XST_SUCCESS;
                switch (userInput)
```

```
		{
		case '1':
			status = DisplayStop(&dispCtrl);
			DisplaySetMode(&dispCtrl, &VMODE_640x480);
			DisplayStart(&dispCtrl);
			fResSet = 1;
			break;
		case '2':
			status = DisplayStop(&dispCtrl);
			DisplaySetMode(&dispCtrl, &VMODE_800x600);
			DisplayStart(&dispCtrl);
			fResSet = 1;
			break;
		case '3':
			status = DisplayStop(&dispCtrl);
			DisplaySetMode(&dispCtrl, &VMODE_1280x720);
			DisplayStart(&dispCtrl);
			fResSet = 1;
			break;
		case '4':
			status = DisplayStop(&dispCtrl);
			DisplaySetMode(&dispCtrl, &VMODE_1280x1024);
			DisplayStart(&dispCtrl);
			fResSet = 1;
			break;
		case '5':
			status = DisplayStop(&dispCtrl);
			DisplaySetMode(&dispCtrl, &VMODE_1600x900);
			DisplayStart(&dispCtrl);
			fResSet = 1;
			break;
		case '6':
			status = DisplayStop(&dispCtrl);
			DisplaySetMode(&dispCtrl, &VMODE_1920x1080);
			DisplayStart(&dispCtrl);
			fResSet = 1;
			break;
		case 'q':
			fResSet = 1;
			break;
		default :
			xil_printf("\n\rInvalid Selection");
			TimerDelay(500000);
		}
		if (status == XST_DMA_ERROR)
		{
			xil_printf("\n\rWARNING: AXI VDMA Error detected and cleared\n\r");
		}
	}
}

void CRMenu()
{
	xil_printf("\x1B[H"); //Set cursor to top left of terminal
	xil_printf("\x1B[2J"); //Clear terminal
	xil_printf("**************************************************\n\r");
	xil_printf("*               Tic Tac Toe Game                 *\n\r");
	xil_printf("**************************************************\n\r");
	xil_printf("*Current Resolution: %28s*\n\r", dispCtrl.vMode.label);
	printf("*Pixel Clock Freq. (MHz): %23.3f*\n\r", dispCtrl.pxlFreq);
	xil_printf("**************************************************\n\r");
	xil_printf("\n\r");
	xil_printf("1 - %s\n\r", VMODE_640x480.label);
	xil_printf("2 - %s\n\r", VMODE_800x600.label);
	xil_printf("3 - %s\n\r", VMODE_1280x720.label);
	xil_printf("4 - %s\n\r", VMODE_1280x1024.label);
```

```
	xil_printf("5 - %s\n\r", VMODE_1600x900.label);
	xil_printf("6 - %s\n\r", VMODE_1920x1080.label);
	xil_printf("q - Quit (don't change resolution)\n\r");
	xil_printf("\n\r");
	xil_printf("Select a new resolution:");
}


/*
 * Bilinear interpolation algorithm. Assumes both frames have the same stride.
 */
void ScaleFrame(u8 *srcFrame, u8 *destFrame, u32 srcWidth, u32 srcHeight, u32 destWidth, u32
destHeight, u32 stride)
{
	float xInc, yInc; // Width/height of a destination frame pixel in the source frame
coordinate system
	float xcoSrc, ycoSrc; // Location of the destination pixel being operated on in the
source frame coordinate system
	float x1y1, x2y1, x1y2, x2y2; //Used to store the color data of the four nearest source
pixels to the destination pixel
	int ix1y1, ix2y1, ix1y2, ix2y2; //indexes into the source frame for the four nearest
source pixels to the destination pixel
	float xDist, yDist; //distances between destination pixel and x1y1 source pixels in
source frame coordinate system

	int xcoDest, ycoDest; // Location of the destination pixel being operated on in the
destination coordinate system
	int iy1; //Used to store the index of the first source pixel in the line with y1
	int iDest; //index of the pixel data in the destination frame being operated on

	int i;

	xInc = ((float) srcWidth - 1.0) / ((float) destWidth);
	yInc = ((float) srcHeight - 1.0) / ((float) destHeight);

	ycoSrc = 0.0;
	for (ycoDest = 0; ycoDest < destHeight; ycoDest++)
	{
		iy1 = ((int) ycoSrc) * stride;
		yDist = ycoSrc - ((float) ((int) ycoSrc));

		/*
		 * Save some cycles in the loop below by presetting the destination
		 * index to the first pixel in the current line
		 */
		iDest = ycoDest * stride;

		xcoSrc = 0.0;
		for (xcoDest = 0; xcoDest < destWidth; xcoDest++)
		{
			ix1y1 = iy1 + ((int) xcoSrc) * 3;
			ix2y1 = ix1y1 + 3;
			ix1y2 = ix1y1 + stride;
			ix2y2 = ix1y1 + stride + 3;

			xDist = xcoSrc - ((float) ((int) xcoSrc));

			/*
			 * For loop handles all three colors
			 */
			for (i = 0; i < 3; i++)
			{
				x1y1 = (float) srcFrame[ix1y1 + i];
				x2y1 = (float) srcFrame[ix2y1 + i];
				x1y2 = (float) srcFrame[ix1y2 + i];
				x2y2 = (float) srcFrame[ix2y2 + i];
```

```
                                /*
                                 * Bilinear interpolation function
                                 */
                                destFrame[iDest] = (u8)
((1.0-yDist)*((1.0-xDist)*x1y1+xDist*x2y1) + yDist*((1.0-xDist)*x1y2+xDist*x2y2));
                                iDest++;
                        }
                        xcoSrc += xInc;
                }
                ycoSrc += yInc;
        }

        /*
         * Flush the framebuffer memory range to ensure changes are written to the
         * actual memory, and therefore accessible by the VDMA.
         */
        Xil_DCacheFlushRange((unsigned int) destFrame, DEMO_MAX_FRAME);

        return;
}

void PrintPattern(u8 *frame, u32 width, u32 height, u32 stride, int pattern)
{
        u32 xcoi, ycoi;
        u32 iPixelAddr;

        switch (pattern)
        {
        case TIC_TAC_TOE:
                for(xcoi = 0; xcoi < (width*3); xcoi+=3)
                {
                        iPixelAddr = xcoi;

                        for(ycoi = 0; ycoi < height; ycoi++)
                        {
                                if ((xcoi % (width) == width - 4) || (xcoi % (width) == width -
3) || (xcoi % (width) == width - 2) || (xcoi % (width) == width - 1) || (xcoi % (width) == 0)
|| (xcoi % (width) == 1) || (xcoi % (width) == 2) || (xcoi % (width) == 3) || (xcoi % (width)
== 4))
                                {
                                        frame[iPixelAddr] = 0;
                                        frame[iPixelAddr + 1] = 0;
                                        frame[iPixelAddr + 2] = 0;
                                        iPixelAddr += stride;
                                }
                                else if ((ycoi % (height/3) == height/3 - 2) || (ycoi %
(height/3) == height/3 - 1)|| (ycoi % (height/3) == 0)|| (ycoi % (height/3) == 1)|| (ycoi %
(height/3) == 2))
                                {
                                        frame[iPixelAddr] = 0;
                                        frame[iPixelAddr + 1] = 0;
                                        frame[iPixelAddr + 2] = 0;
                                        iPixelAddr += stride;
                                }
                                else
                                {
                                        int red = 0;
                                        int green = 0;
                                        int blue = 0;
                                        for (int c = 0; c < 3; c++)
                                        {
                                                for (int b = 0; b < 3; b++)
                                                {
```

```
                                                if ((board[c][b] == GREEN) && (xcoi <=
(width)*(c+1)) && (xcoi >= (width)*(c)) && (ycoi <= (height/3)*(b+1)) && (ycoi >=
(height/3)*(b)))
                                                {
                                                        red = 0;
                                                        green = 255;
                                                        blue = 0;
                                                }
                                                else if ((board[c][b] == BLUE) && (xcoi <=
(width)*(c+1)) && (xcoi >= (width)*(c)) && (ycoi <= (height/3)*(b+1)) && (ycoi >=
(height/3)*(b)))
                                                {
                                                        red = 0;
                                                        green = 0;
                                                        blue = 255;
                                                }
                                                else if ((board[c][b] == NONE) && (xcoi <=
(width)*(c+1)) && (xcoi >= (width)*(c)) && (ycoi <= (height/3)*(b+1)) && (ycoi >=
(height/3)*(b)))
                                                {
                                                        red = 255;
                                                        green = 255;
                                                        blue = 255;
                                                }
                                        }
                                }
                                frame[iPixelAddr] = blue;
                                frame[iPixelAddr + 1] = green;
                                frame[iPixelAddr + 2] = red;
                                iPixelAddr += stride;

                        }

                }
        }
        Xil_DCacheFlushRange((unsigned int) frame, DEMO_MAX_FRAME);
        break;
case BOX:
        for(xcoi = 0; xcoi < (width*3); xcoi+=3)
        {
                iPixelAddr = xcoi;

                for(ycoi = 0; ycoi < height; ycoi++)
                {
                        if ((xcoi % (width) == width - 4) || (xcoi % (width) == width -
3) || (xcoi % (width) == width - 2) || (xcoi % (width) == width - 1) || (xcoi % (width) == 0)
|| (xcoi % (width) == 1) || (xcoi % (width) == 2) || (xcoi % (width) == 3) || (xcoi % (width)
== 4))
                        {
                                if ((xcoi <= width*game_x + 8) && (xcoi >=
((width*3)/4)*(game_x - 1)) && (ycoi <= (height/3)*(game_y) ) && (ycoi >=
(height/3)*(game_y-1)))
                                {
                                        frame[iPixelAddr + 0] = 0;
                                        frame[iPixelAddr + 1] = 0;
                                        frame[iPixelAddr + 2] = 255;
                                }

                        }
                        else if ((ycoi % (height/3) == height/3 - 2) || (ycoi %
(height/3) == height/3 - 1)|| (ycoi % (height/3) == 0)|| (ycoi % (height/3) == 1)|| (ycoi %
(height/3) == 2))
                        {
                                if ((xcoi <= width*game_x ) && (xcoi >= ((width))*(game_x
- 1)) && (ycoi <= (height/3)*(game_y) ) && (ycoi >= (height/3)*(game_y-1)))
                                {
```

```
                                        frame[iPixelAddr + 0] = 0;
                                        frame[iPixelAddr + 1] = 0;
                                        frame[iPixelAddr + 2] = 255;
                                }

                        }
                        iPixelAddr += stride;
                }
        }
        Xil_DCacheFlushRange((unsigned int) frame, DEMO_MAX_FRAME);
        break;
    default :
        xil_printf("Error: invalid pattern passed to DemoPrintTest");
    }
}

void ISR(void *callBackRef, void *pVideo)
{
    char *data = (char *) callBackRef;
    *data = 1; //set fRefresh to 1
}

void resetBoard()
{
    int x,y;
    for (x = 0; x < 3; x++)
    {
        for (y = 0; y < 3; y++)
        {
            board[x][y] = NONE;
        }
    }
}

void checkForWin(int color)
{
    if ((board[0][0] == color) &&  (board[1][0] == color) && (board[2][0] == color))
    {
        win = 1;
    }
    else if ((board[0][1] == color) &&  (board[1][1] == color) && (board[2][1] == color))
    {
        win = 1;
    }
    else if ((board[0][2] == color) &&  (board[1][2] == color) && (board[2][2] == color))
    {
        win = 1;
    }
    else if ((board[0][0] == color) &&  (board[0][1] == color) && (board[0][2] == color))
    {
        win = 1;
    }
    else if ((board[1][0] == color) &&  (board[1][1] == color) && (board[1][2] == color))
    {
        win = 1;
    }
    else if ((board[2][0] == color) &&  (board[2][1] == color) && (board[2][2] == color))
    {
        win = 1;
    }
    else if ((board[0][0] == color) &&  (board[1][1] == color) && (board[2][2] == color))
    {
        win = 1;
    }
    else if ((board[0][2] == color) &&  (board[1][1] == color) && (board[2][0] == color))
    {
```

```
                win = 1;
        }

}

void cpuTurn()
{
        int success = 0;

        while (success != 1)
        {
                int num_x = (rand() % 3) + 1;
                int num_y = (rand() % 3) + 1;

                if (board[num_x-1][num_y-1] == NONE)
                {
                        success = 1;
                        board[num_x-1][num_y-1] = marker;
                        PrintPattern(pFrames[dispCtrl.curFrame], dispCtrl.vMode.width,
dispCtrl.vMode.height, DEMO_STRIDE, TIC_TAC_TOE);
                        PrintPattern(pFrames[dispCtrl.curFrame], dispCtrl.vMode.width,
dispCtrl.vMode.height, DEMO_STRIDE, BOX);
                        turns++;
                        checkForWin(marker);
                }
        }
        if (win == 1)
        {
                game_x = 1;
                game_y = 1;
                resetBoard();
                xil_printf("\x1B[H"); //Set cursor to top left of terminal
                xil_printf("\x1B[2J"); //Clear terminal
                switch(marker)
                {
                case GREEN:
                        xil_printf("Congratulations! GREEN won!\n\r");
                        xil_printf("Resetting...\n\r");
                        break;
                case BLUE:
                        xil_printf("Uh Oh! BLUE won!\n\r");
                        xil_printf("Resetting...\n\r");
                        break;
                }
                marker = GREEN;
                win = 0;
                turns = 0;
                TimerDelay(1000000);
                PrintPattern(pFrames[dispCtrl.curFrame], dispCtrl.vMode.width,
dispCtrl.vMode.height, DEMO_STRIDE, TIC_TAC_TOE);
                PrintPattern(pFrames[dispCtrl.curFrame], dispCtrl.vMode.width,
dispCtrl.vMode.height, DEMO_STRIDE, BOX);
        }
        marker = GREEN;
}
```

VI - Conclusion

To summarize the final system of the Tic Tac Toe game, we configured the HDMI and the zybo board through the micro usb cable as it was a challenging part for us to do. Till now, we displayed the 3 x 3 grid on the monitor screen. We were also able to control the user interface using the dip switches and push buttons and move the grid on the monitor video screen. Basically, the keyboard is connected to the zybo board and the video screen is plugged through the HDMI. Furthermore, we used UART for the serial communication terminal.

While designing the basic system of the project, we designed the skeleton program of the Tic Tac Toe game and configured HDMI with a board. Furthermore, we worked on the complete software source code for the advanced system of the Tic Tac Toe game where we used dip switches to select the tile from 3 X 3 grid of game and filled it with a green marker. Moreover, we can have access to move the tile using 4 push buttons for move left, move up, move right, move down from left to right order of push buttons respectively. The main challenging part of the project was to configure HDMI and Zybo board. Whoever wins the game, a message will be transferred using serial communication and will be shown on the screen regarding the status of the game. For ex. “Computer won” or “You won”. Once the game is ended, the screen gets blanked to resume the game again if the user wants.

To summarize it all, we had worked on the configuration and design of the system at basic level. Working towards the advanced level, we executed the application using the GPIO buttons and switches, UART, HDMI as one of the main components to achieve the goal of game You VS Computer with the random algorithm of Tic Tac Toe game.

VII - Appendix

References

Source Code Repo

https://github.com/TPridy/TicTacToe

Example Project

https://github.com/Digilent/Zybo-Z7-10-HDMI

AXI4-Stream Infrastructure

https://www.xilinx.com/support/documentation/ip_documentation/axis_infrastructure_ip_suite/v1_1/pg085-axi4stream-infrastructure.pdf

AXI4-Stream to Video Out

https://www.xilinx.com/support/documentation/ip_documentation/v_axi4s_vid_out/v4_0/pg044_v_axis_vid_out.pdf

Demonstration

https://drive.google.com/file/d/1pq51e9kGKxrXqIf1N4qNy0fwq8xq0T_m/view?usp=sharing