Individual Report
Brickout

Arlene Fu
Jfa49@sfu.ca
Team 14

Table of Contents

1.0 Introduction

The goal of this ENSC 452 course is to implement a program on ZedBoard with both Xilinx's system software and hardware. We decide our project to be a video game similar to the brick breaker. In our game, the user can move the bar at the bottom of the screen left and right in order to bounce the movable ball up and try to hit the bricks. Once the ball hit any brick, the brick will be downgraded and eventually wiped out. We will split our job into 6 milestones and use Agile methodology to accept greater flexibility and to get higher efficiency. Each part will be tested by a specific test plan to ensure the completeness. [1]

2.0 Technical Literature used in the Project

In order to accomplish the goal of our final project, I consulted many available open sources, here I just listing three of them.

2.1. Web Page:

The web page [2] is an online tutorial called ‘Create a VGA controller for the ZedBoard’. It helps me to fully understand how the VGA reads signal and deals with timing.

This web introduces the video displaying and also provides a brief introduction of the VGA on the ZedBoard. I learned the relationship between the video frame and synchronization signals. It also provides the timing information for some specified resolutions.

The page is provided by Dev Flow, which is a specialist in FPGA area. They provide a series of very good tutorials for users to developing skills in Xilinx tools.

This tutorial not only teaches you how to deal with VGA step by step but provide a package of the relevant code. The attached code helps me to successfully display a single color on the monitor. All my later work associated with VGA is based from this.

2.2. User Manual:

The user manual provided by Xilinx is very useful when developing hardware. There is every reason to believe that this source is credible since It is the official documentation provided by Xilinx.

The first time I used the ZedBoard hardware user’s guide [3] is when I did the Xilinx tutorial UG1165. One part of that tutorial teaches how to connect the five AXI GPIO pin. However, the pin number provided in the tutorial is not for ZedBoard, I need to search that from [3]. I also used this user manual when I implemented DDR, SD card, PS button, and VGA.

2.3. Technical Journal:

When I tried to build a pseudo-random number generator, I found that if I randomly select the pattern of the LFSR, then the result will not be random enough. Therefore, I searched on the internet to see whether there is any research about that. One of the international journal “FPGA Based N-Bit LFSR to generate random sequence number” [4] summarize the method of selecting feedback polynomial for the LFSR. I trust the result reporting by this paper because it has been cited by many articles. In fact, by implementing the rules to my circuit, the output sequence looks more like ‘random’.

3.0 Contribution

In our project, my partner Sam focuses on the audio part and game logic, while I am responsible to develop the following part:

Milestone 1, 3, 4, Display image on the monitor through VGA

Phase1:

I spent almost four weeks in total to implement and improve the performance of VGA controller.

In the first milestone, my goal is displaying three side walls, a ball, and paddle on the screen. All the shapes are static, there is no need for the dynamic image.

Initially, I built the VGA controller based on one online tutorial [2]. I modified the source code from [2] and placed it in one RLT structure. At that stage all the pixel frames are hardcoded, (in PL), and I did not connect to any processing system. I decided to set the resolution at 640 x 480, which requires 25.175MHz pixel clock. The refresh rate is 60hz. The RTL is driven by the system clock, and all the computed signal are fed into a FIFO, as shown in Figure 1. I make the system clock to control write clock of the FIFO, and pixel clock(25.175MHz) to guide read operation. In this case, generating pixel is purely based on the system clock, rather than pixel clock. This step provides me the prerequisite to implementing the double buffer to ensure the refresh rate.

Problem encountered:

During this phase, I was not able to get anything on the screen. I double checked the PL connection and cannot figure out why the monitor did not display any signal. For debugging, I created test bench to run the behavioral simulation with the Vivado Simulator (XSIM). After simulating, I noticed the problem is that the signal fed into FIFO overflows, even I set the ‘full' flag for FIFO. I consulted with Dan and find the easiest solution to this problem is to set ‘almost_full’ flag instead of ‘full’ flag.

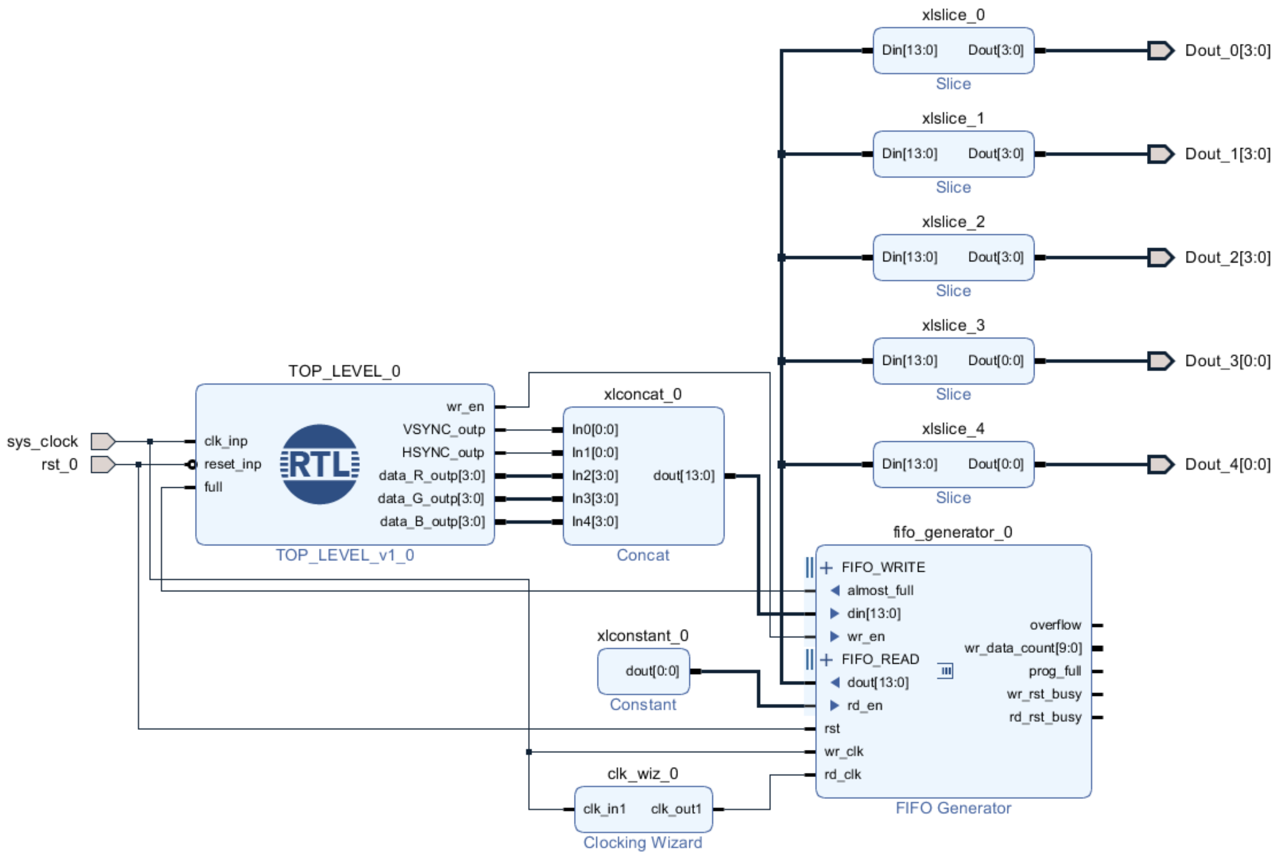


Figure 1 *Block diagram for initial VGA structure*

Phase2:

I struggled with VGA for a while. After accomplishing my milestone 1 and defined a function that allows the ball start moving from an initial location along pre-defined function, I switched to another VGA IP core from GitHub [5]. The VGA controller used memory mapping method rather than streaming. In general, I wrote every pixel per frame (640x480 pixels in total) into a single line array, and then map them to DDR start from 0x00200000. [6]. The way that VGA controller reads the data is simply using memcopy function. I cleared the whole frame buffer when 640x480 pixels finished writing to the buffer. In the meanwhile, I changed the location of the moving ball. But this way made the screen flicking intensively. Therefore, instead of clear the whole buffer every frame, I just overwrote the old ball with black color while updating the location of the ball. However, this is not a long-term solution. I am looking for a way to implement double buffer.

In my code, a class object called 'vga_obj' includes the configuration address for PL (VGA controller) and a 'vga_frame_obj'. 'vga_frame_obj' is a 640x480 array, which each element has 12 bits to represent RGB. I initialized two vga_frame_obj as my buffers, and I defined a swap function to swap with buffers. These two buffers are mapped to DDR address: 0x00200000 and 0x00400000 [6]. The gap between those two addresses is sufficient for store one frame data. A parameter called ‘swap’ is used to toggle between two buffers. For example, when the swap variable is equal to 0, then read buffer 1 and write buffer2.

But when I dealt with the double buffer, I needed to define two addresses and one more variable that assigned by two addresses in turn. Initially, I made that variable as (void*) type, but I found the type is wrong, I needed to use type int. Unfortunately, there were still some bugs in my program and I cannot make double buffer work.

Phase3:

Therefore, I used the IP core shared by Wesley Kendall [7], which using VDMA instead of DMA. For VDMA, Vivado provides an IP core called Video Timing Controller, which makes life a lot easier by having the default setting for different resolutions. After using this design block, I can easily implement frames using double buffer and make the program have the ability to respond smoothly to the button interrupt. In this case, the speed of the ball is 5 times as the refresh rate, which is 60Hz for current resolution.

Phase4:

Since I wish to implement my own VGA controller designed in Phase 1, I added a FIFO instead of Video Timing Controller IP. The ports on VDMA are using memory-mapped to streaming AXI4-streaming bus, which source and sink a continuous stream of data without addresses.

By learning the tutorial [8] and the block diagram from [9], I changed the regular FIFO in Phase 1 to a streaming FIFO, and add my own VGA controller. The block design shows in Figure 2. In this case, I need to output an extra interrupt signal to VDMA when the VGA finishes drawing one frame.

I borrowed part of Wesley’s VDMA driver codes to implement swapping between 2 declared buffers. According to [6], the address range for CPUs’ DDR is from 0x00100000 to 0x3FFFFFFF, I put two buffers in DDR at the address starts from 0x01000000. For every pixel, I directly wrote the data inside the corresponding buffer.

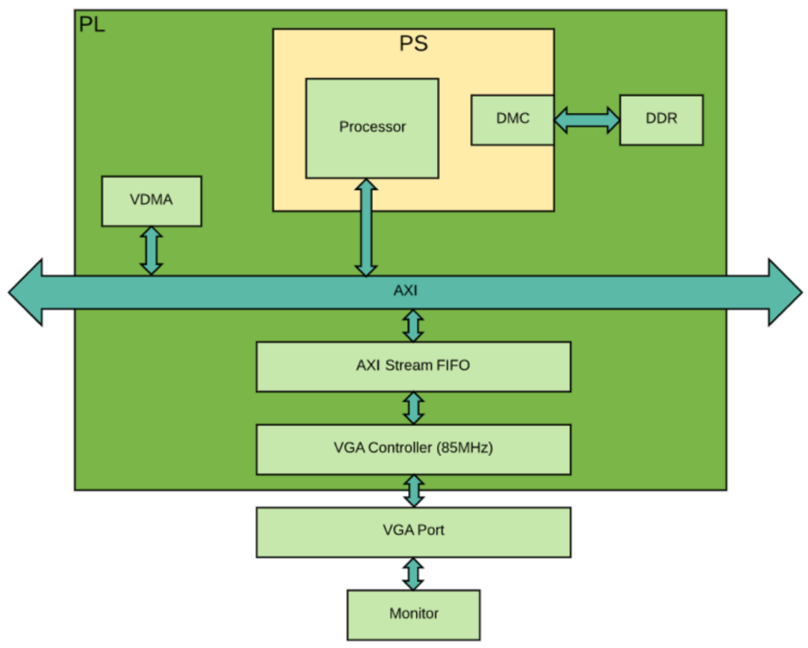


Figure 2. Block design for final VGA structure

Test:

For milestones with VGA controller, I just manually saw the declared images/shapes on the screen through VGA. After implementing the double buffer, the monitor should display without flicking.

MILESTONE 2, ENABLE AXI BUTTON AND PS BUTTON FOR TWO-PLAYER MODE

For our game, it is essential to have buttons work. The goal of my second milestone is to use left and right button on the ZedBoard to control the paddle on the screen. To accomplish this result, I first used the code from Xilinx tutorial UG1165 [10]. The tutorial provided a general rule to apply interrupt with LED, and I changed it to button interrupt. From that, my program was able to detect every interrupt and display the correct message to show which button had been pressed. Besides, I also debounced the buttons to avoid the unreasonable behavior.

From this tutorial, I learned that each on-board peripheral has its unique pin number. I needed to assign this in I/O planning section and notify the processing system to use which peripheral.

For enable the button interrupt, I need to do the following steps:

1. Lookup configuration data about the specific button in the device configuration table, using XGpio_LookupConfig(DeviceID)
2. Initialize a configuration structure with function XGpio_CfgInitialize()
3. Initialize the GPIO driver with XGpio_Initialize()
4. Look for the device configuration based on the unique device id of interrupt controller device by XScuGic_LookupConfig()
5. XScuGic_CfgInitialize() initialize a specific interrupt controller driver

6. Make a connection between the interrupt source and the associated handler using XScuGic_Connect()
7. XScuGic_Enable() enables the interrupt source
8. Enable interrupt by XGpio_InterruptEnable()
9. Have to set the global enable bit and enable the interrupt output source by XGpio_InterruptGlobalEnable(), otherwise, XGpio_InterruptEnable() will not be passed through
10. Set up the interrupt vector table and register an exception handler for a specific exception by function Xil_ExceptionInit() and Xil_ExceptionRegisterHandler()

I find these functions from Xilinx documentation [11] [12] [13]. Figure 3 is the block diagram of my five AXI buttons' connection. As shown in the diagram, before feeding the button values into GPIO IP, I put a debounce block between them. GPIO and ZYNQ processing system are connected using AXI interconnect IP core.

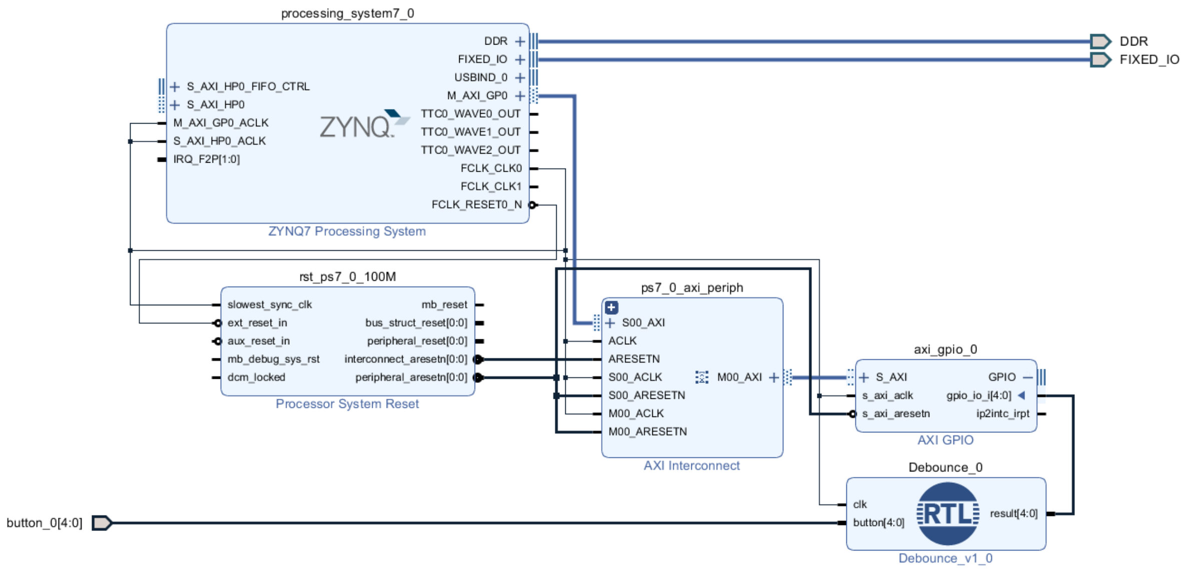
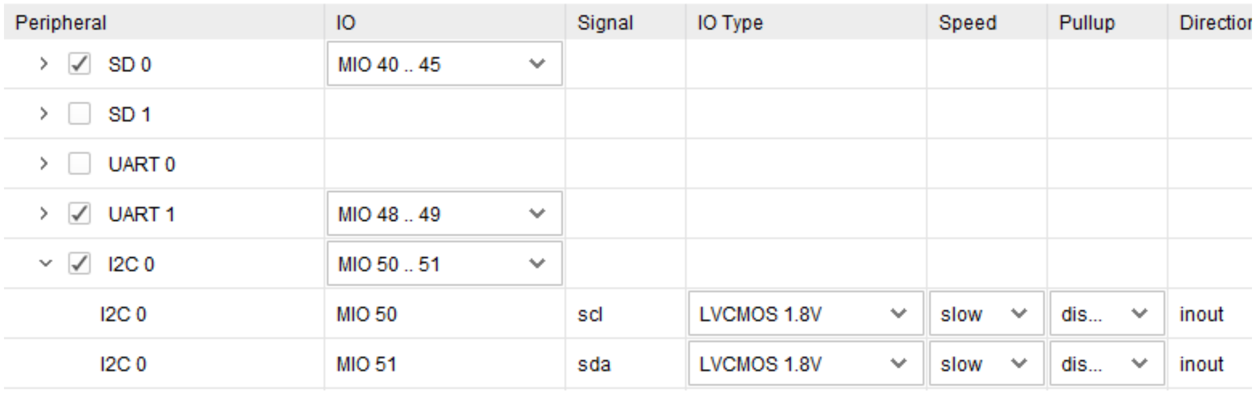


Figure 3. Block diagram for AXI GPIO

We decide to enable two-player mode in our game. This requires two more buttons for the second player. By observing the ZedBoard, there are 4 more buttons (button 6 to button 9). Button 6 ties the PROGRAM_B_0 pin to the ground and button 7 is a reset button. Button 8 and button 9 are the two PS GPIO buttons, which are my target.

Those two buttons can be easily set up by enabling MIO 50 and MIO 51 under I2C 0 section, as shown in Figure 4.



Peripheral	IO	Signal	IO Type	Speed	Pullup	Directio
› ✓ SD 0	MIO 40 .. 45					
› ☐ SD 1						
› ☐ UART 0						
› ✓ UART 1	MIO 48 .. 49					
⌄ ✓ I2C 0	MIO 50 .. 51					
I2C 0	MIO 50	scl	LVCMOS 1.8V	slow	dis...	inout
I2C 0	MIO 51	sda	LVCMOS 1.8V	slow	dis...	inout

Figure 4. Zynq MIO configuration setting for PS GPIO

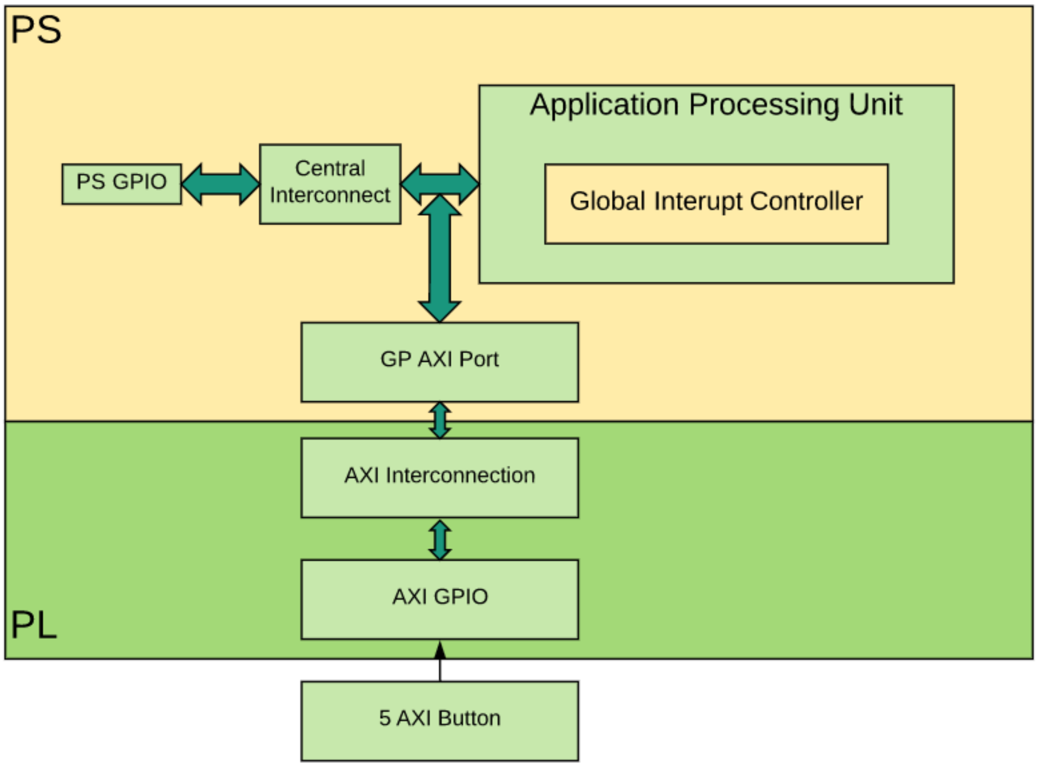


Figure 5. Block diagram for AXI button and PS button

The block diagram of this system is as shown in Figure 5. Those two PS buttons need another interrupt handler.

When I first implemented the interrupt handler for PS GPIO, I did the same thing as I mentioned above, except I used XGpioPs instead of XGpio. I also set the direction of the specified pin, since I had pin MIO 50 for button 8 and MIO 51 for button 9. According to the function definition, 0 represents input and 1 represents output. I set both two pins as input ports and set the function to detect interrupt at rising edge.

After setting up the interrupt service routine, I got the status number 0x40000 every time I pressed button 8 and pressing button 9 returned 0x80000 as a status number. Therefore, I converted these two numbers into integer and used them to determine which button is pressed.

Test:

Switch the game mode to double-user mode, let one player press the buttons (only left, right, button 8 and button 9) to move the paddle. Press up/down/center button will receive the corresponding message on SDK terminal.

Problem encountered:

However, I found that if I use both AXI GPIO interrupt function and PS GPIO interrupt function together, the second interrupt will always overwrite the first one. After debugging for almost one day, I noticed the problem was that I initialized both general interrupt controller and exception table twice.

The general interrupt controller (Xscugic) is a global controller. In the case of having two interrupt instances, I only need to initialize GIC once and connect the two interrupt sources and their corresponding handler separately. A similar issue happens with exception table.

As for debouncing the two PS button, I cannot come up with a way to implement it in hardware. Therefore, I chose a software method. For software debouncing, I just recorded the button pin which I first detect and iterated for 50000 times. If the detected pin is different than the first one, then exit the loop, otherwise count for 50000 times.

In general, I update the frame once I detect one interrupt, (paddle move left if the left button is pressed, and move right while right button interrupt is triggered.) Every time the system detects a correct interrupt, it will clear current frame and draw the new updated pattern. There is also a small process to ensure the paddle will not run off, it will stop if it almost touches the side walls.

MILESTONE 5, PSEUDO-RANDOM NUMBER GENERATOR

In order to generate random brick location and brick level, the project needs a pseudo-random number generator. Although there is a function called ‘rand()' in C code, which also returns a pseudo-random number in the range of 0 to 32767, I decide to generate random numbers by hardware. One of the most common methods in hardware is by means of a linear-feedback shift register (LFSR). LFSR is a shift register whose input bit is a linear function of its previous state. Each register is interpreted as a polynomial and an n-bit LFSR can cycle through $2^n - 1$ states. [14].

I read some papers about LFSR in VHDL, and I find that 8-bit, 16-bit and 32-bit pseudo-random number generator are more common than others. I also try to randomly construct an LFSR, but the output distribution is not random enough. For example, I got the sequence like 4, 23, 8, 36, 7, 7, 58...from 6-bit LFSR. The rules for selecting feedback polynomial are given in [15]. Therefore, I use the following design in Figure 6.

For our early-stage game, our game resolution is 640x480, the brick region is within 580x200 and we only had 250 bricks. An 8-bit pseudo-random number generator, which has 255 possible patterns in total, can satisfy our need. However, the final version has 760 bricks, we need at least 10-bit LFSR. We changed the presentation of the bricks: use an array to represent the location of each brick, and each brick has its corresponding level. We used the random number generator to decide each brick's level. One brick can have at most level 4, and level 0 represents that location does not contain a brick or the brick at that place has been destroyed.

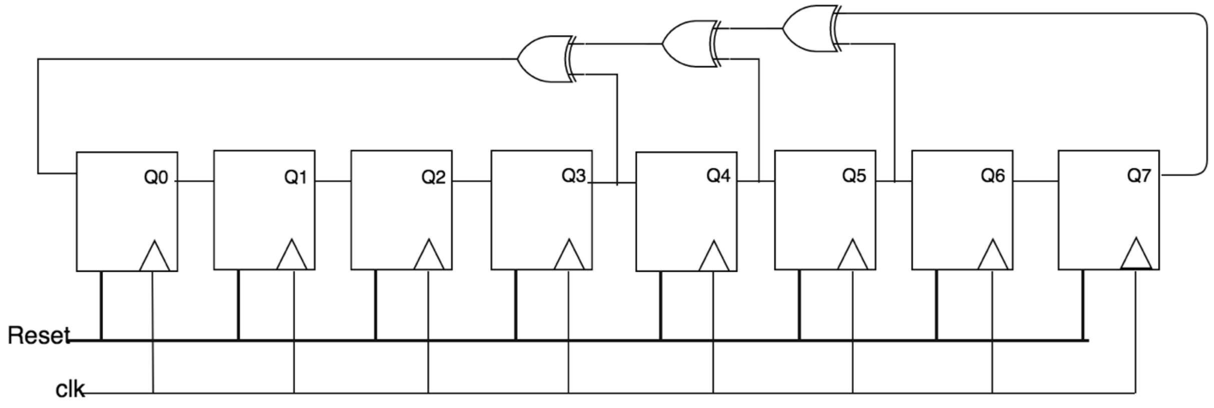


Figure 6.Circuit diagram of 8-bit LFSR

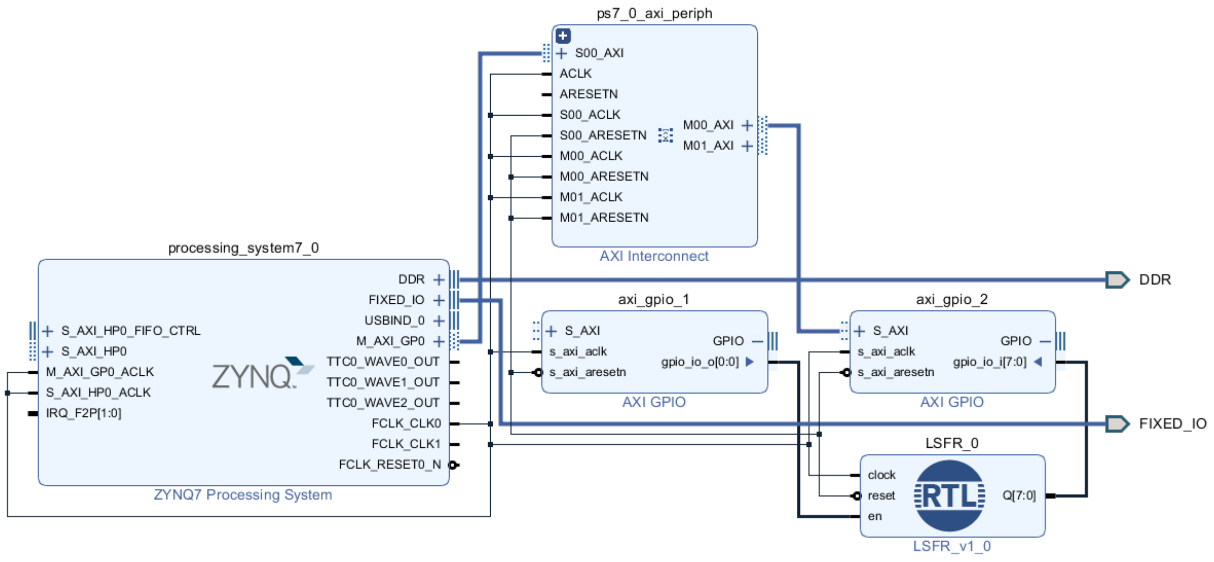


Figure 7. Vivado Block design for LFSR

I use one AXI GPIO (axi_gpio_1 in Figure 7) to feed in enable bit of my LFSR circuit. And there is also another GPIO (axi_gpio_0 in Figure 7) to read the output of the random number. I tried to use only one AXI GPIO with dual ports, this also satisfied my design. But I still choose to use two GPIOs for clearness. Figure 7 is my block design for this pseudo-random number generator. Although this circuit design is not that heavy, I still had one issue when I implemented it. When I tried to read the output from the circuit, I always got value ‘1’ rather than a random number. While debugging, I noticed that I used a processor system reset IP to provide clock and reset, and it is an active-low reset. So, after changing my reset to low, everything worked fine.

To use this LFSR block, after initializing it in my C code, I put it in my main while loop and make it keep generating output. I will fetch the random number whenever I need it.

Test:

Wrote a function to keep fetching the number generated from LFSR. Observe whether the output sequence is random enough.

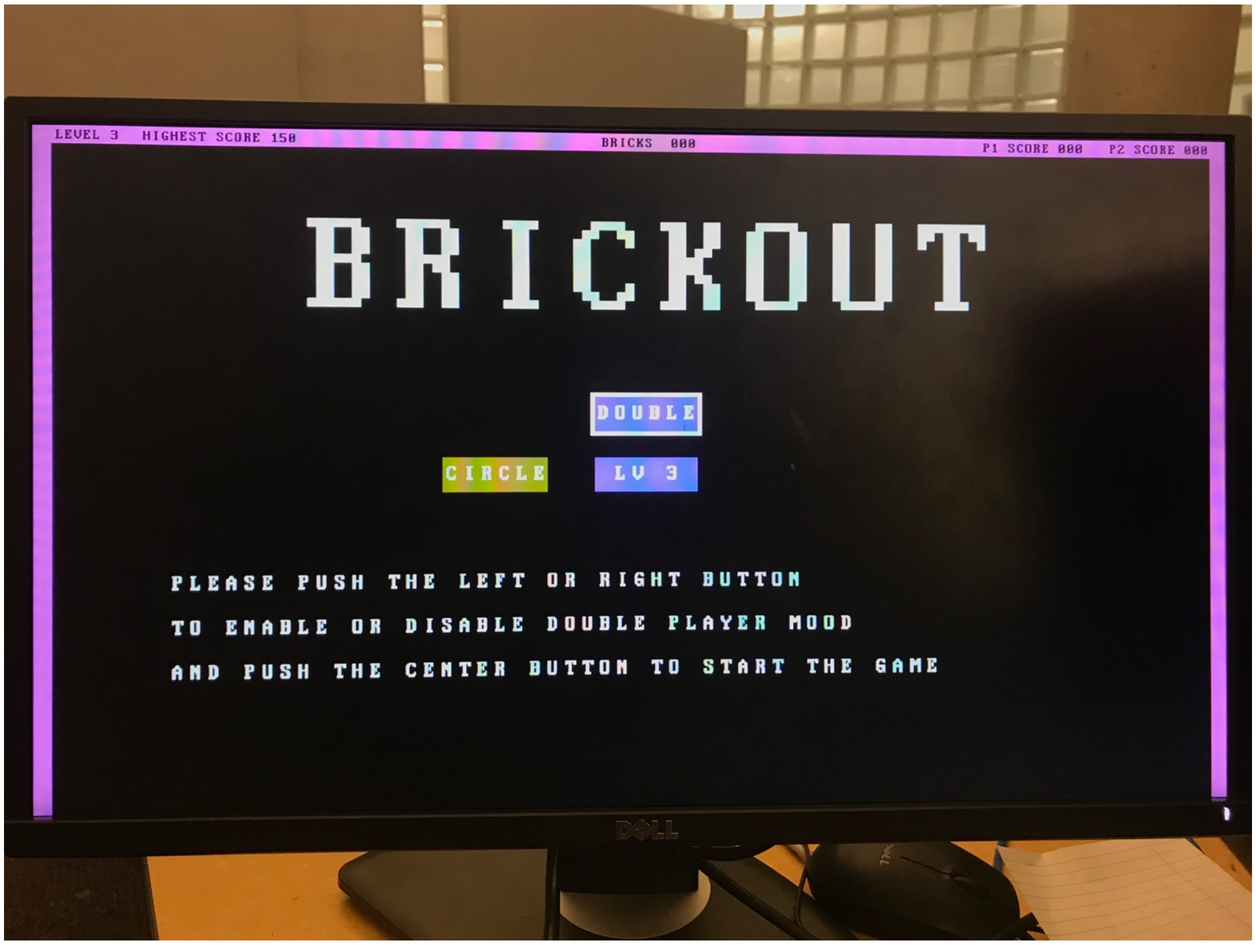
MILESTONE 6, DISPLAY ASCII CHARACTERS ON THE SCREEN

Our game requires displaying some letters on the screen through VGA as a guidance for the user. In the beginning, I planned to implement this in hardware. But I found that it requires a MUX to decide whether to pass the pixel data from C code or from the text block to my VGA controller. I preferred to implement it in C code. I found a VHDL file which defines an 8x16 font ROM for ASCII characters 0 to 127 [16]. I parsed these characters from VHDL style to array in C code and put them into a single .h file.

To print text on the screen, I defined a nested loop to traverse the corresponding array element and write to a contiguous memory block pixel by pixel. For different font of the text, I borrowed magnification logic from Jason Liu. The text result is shown in Figure 8.

MILESTONE 6, IMPLEMENT GAME USER INTERFACE

The goal of my milestone 6 is to display both current score and highest score and enable the menu page. The result shows in Figure 8.



LEVEL 3 HIGHEST SCORE 150 BRICKS 000 P1 SCORE 000 P2 SCORE 000

PLEASE PUSH THE LEFT OR RIGHT BUTTON
TO ENABLE OR DISABLE DOUBLE PLAYER MOOD
AND PUSH THE CENTER BUTTON TO START THE GAME

Figure 8. Home screen

I changed the game logic to keep tracking the score. Every time the ball hits a brick, the brick level will be downgraded by 1 and earn an extra point for the score. For example, If the brick level is 1, after downgrading the brick level will change from 1 to 0, which means the brick will disappear. The brick level varied from 0 to 4, which means destroying one brick can earn at most 4 points. What's more, I also keep tracking the historical highest score and display it on the top of the screen, see Figure 8.

The menu page is enabled, and there are three buttons on the menu. At the beginning of the game, the user will only see the two purple buttons. The user can press left or right button on the board to choose between single user mode and double user mode, and then press the center button to confirm. The second button is the place for the user to change the degree of difficulty of the game. Players can push up and down button on the ZedBoard to change between two buttons, the highlight around the button is also changed correspondingly. Figure 9 is the flowchart of the home menu.

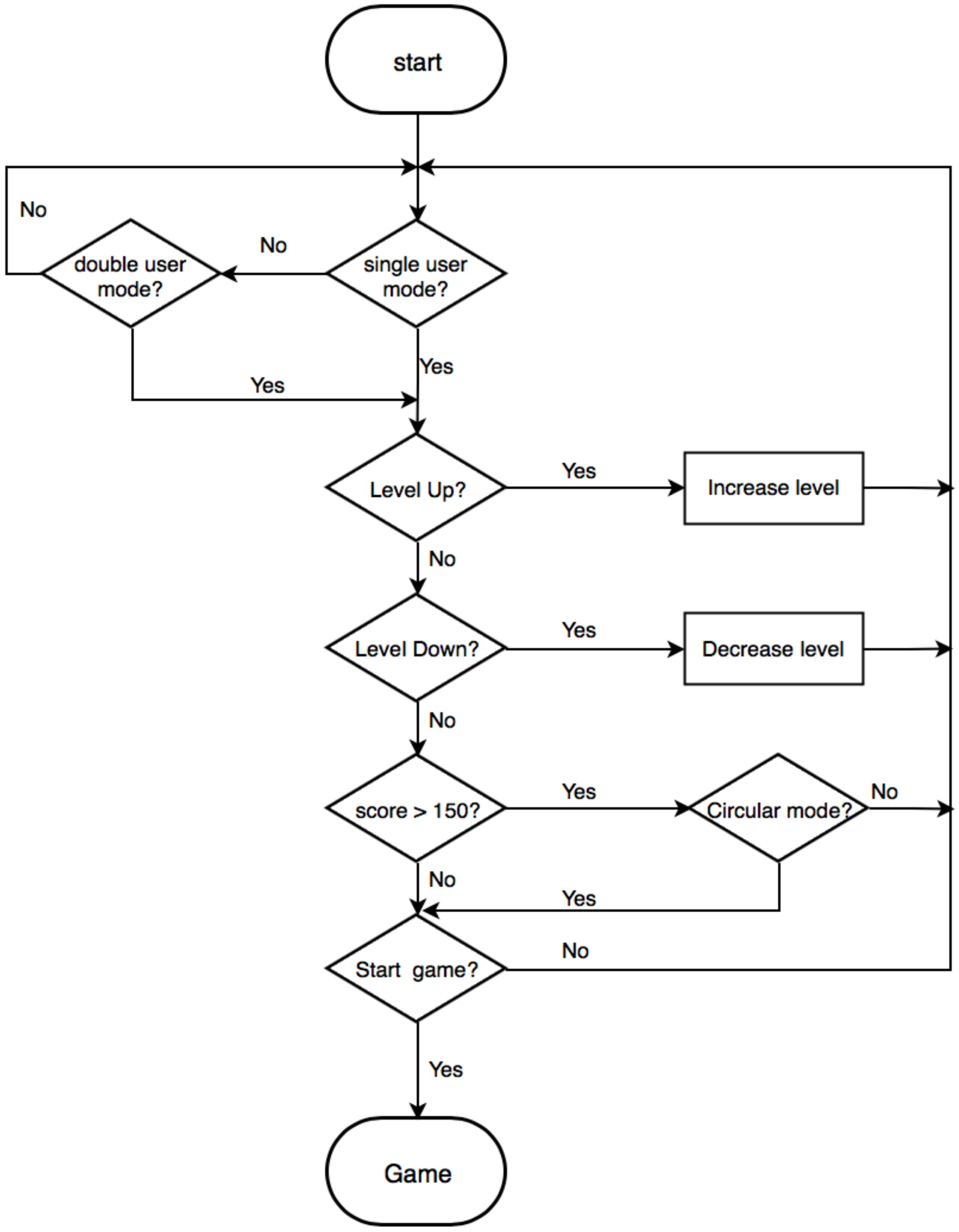


Figure 9. Flowchart of the home page

Test:

To test whether the function is working or not, I just manually check the declared shape on the screen. By pressing the buttons, the highlight should display on the corresponding rectangular. To checking the scoring logic, I just play the game and to see if the score got updated every time one brick has been downgraded.

BONUS: INCREASE GAME RESOLUTION

The initial game resolution is 640x480 with pixel clock at 25.175 MHz. After implementing VDMA and double buffer, our application has the ability to reach a higher resolution.

In our challenge stage, we decided to build a circular mode game. If we implemented the game in initial resolution, the ball and the circular bricks looked more like an oval. The resolution of our lab monitor is 1920x1080. According to [17], the pixel clock of 1080p is 148.50MHz, which is higher than the system clock (100MHz) we are using. We wish to use a resolution, which not only the pixel clock is lower than system clock but has a similar ratio to 1080p. Therefore, we picked 1368x768. The pixel clock is 85.86MHz and the refresh rate is 60Hz.

To change the resolution, there are only a few things to do:

1. Change the second PL fabric clock in Zynq to 85.86MHz
2. Change the parameters in VGA controller, such as front porch, sync pulse, back porch, h_sync polarity and v_sync polarity for both horizontal timing and vertical timing

All the parameters are found from [18].

BONUS: STORE AND READ INFORMATION TO/FROM SD CARD

Since the ZedBoard does not come with an SD card, all the memory we modified is the volatile memory. From [19], data logging in the embedded system is the activity of saving data in a non-volatile memory. The two available non-volatile memories on the ZedBoard are QSPI (quad-SPI) serial NOR flask with a capacity of 256MB and SD card.

We wish to store the historical highest score, which can be achieved by using SD card. The ZedBoard has SD card connector and built-in SD/SDIO host controller, as shown in Figure 10. In order to enable SD0 host controller on PS, I changed the setting as shown in Figure 11. I learned the setting from [19].

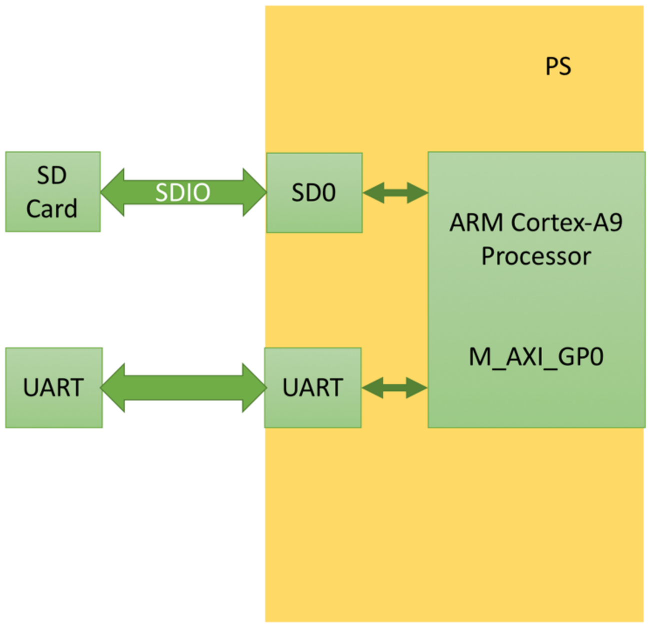
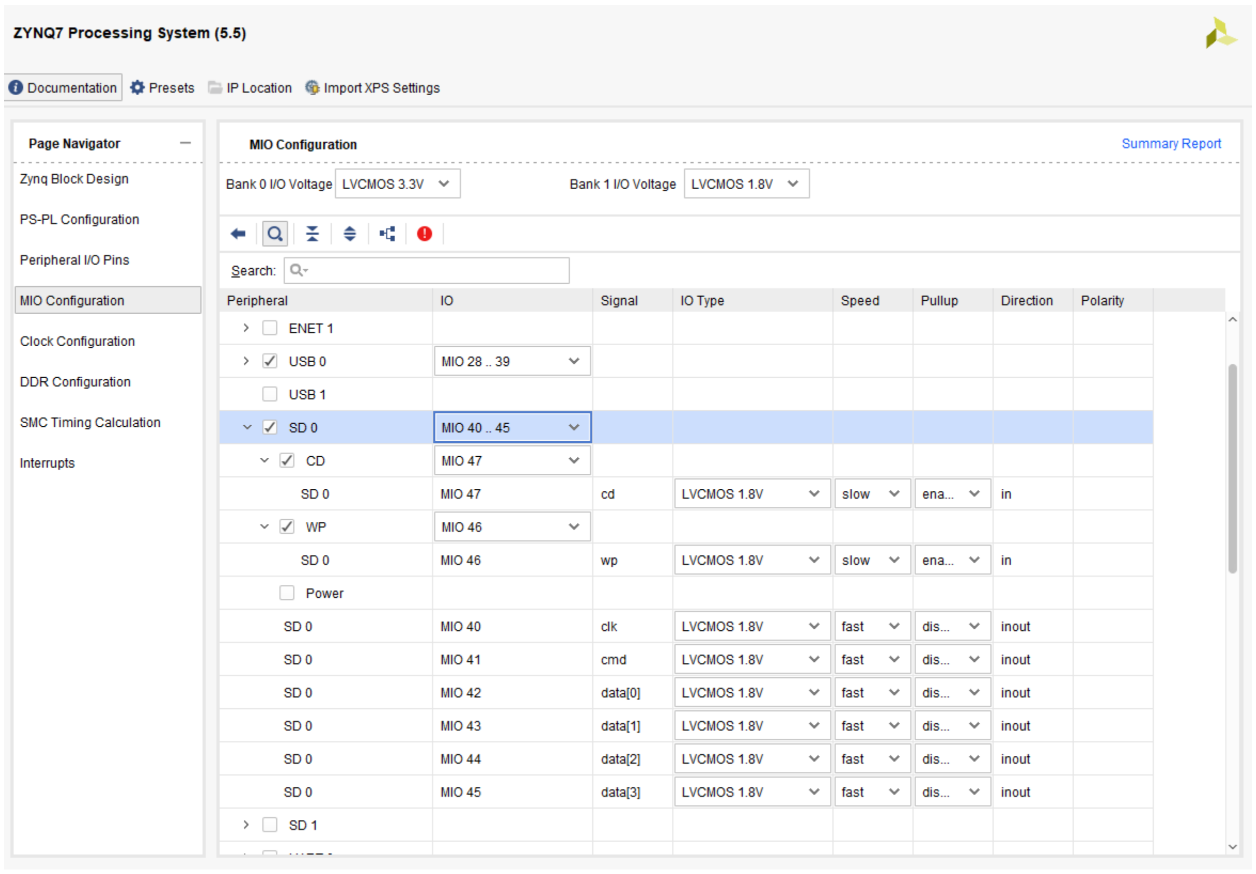


Figure 10. Block diagram for SD card


ZYNQ7 Processing System (5.5)

Documentation | Presets | IP Location | Import XPS Settings

Page Navigator
- Zynq Block Design
- PS-PL Configuration
- Peripheral I/O Pins
- MIO Configuration
- Clock Configuration
- DDR Configuration
- SMC Timing Calculation
- Interrupts

MIO Configuration — Summary Report

Bank 0 I/O Voltage: LVCMOS 3.3V — Bank 1 I/O Voltage: LVCMOS 1.8V

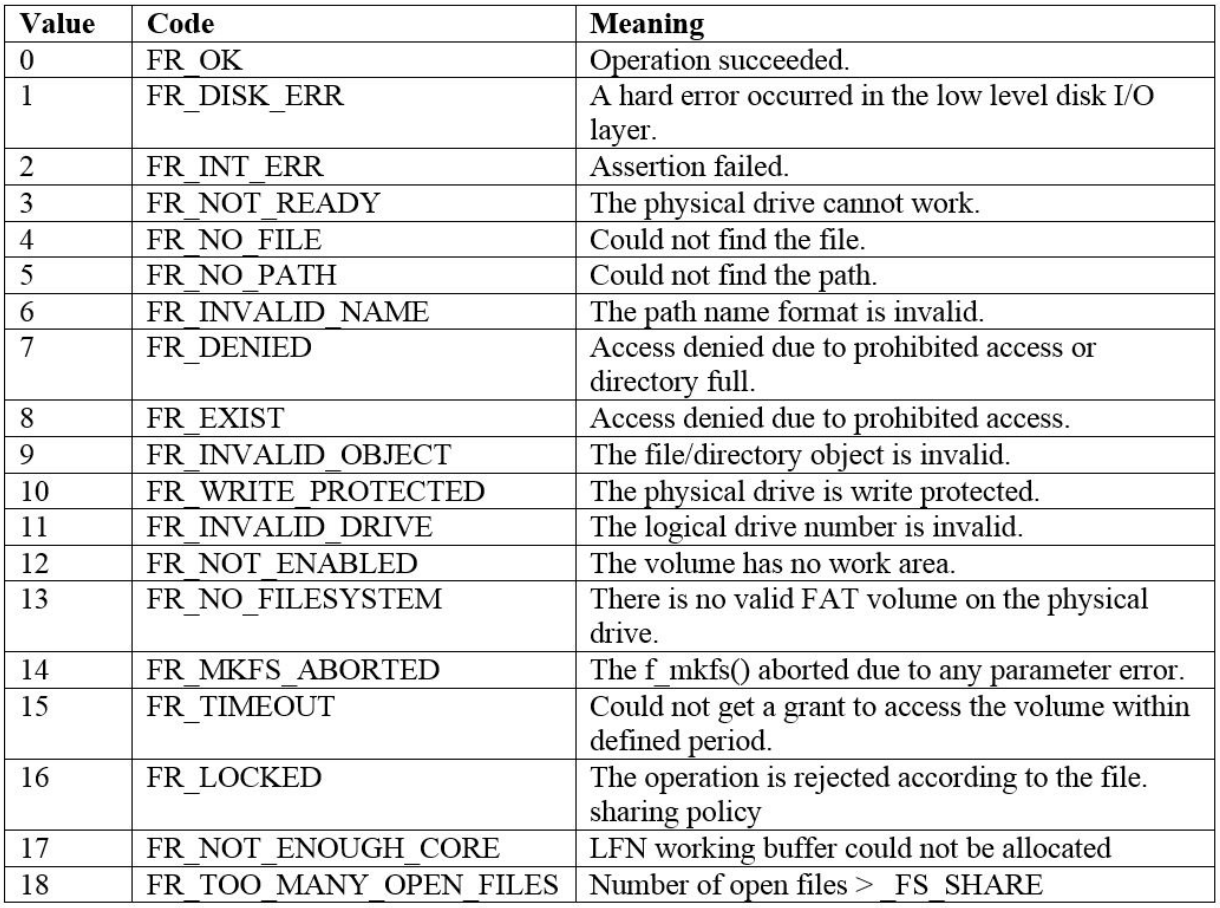
Search:

Peripheral	IO	Signal	IO Type	Speed	Pullup	Direction	Polarity
☐ ENET 1							
✓ USB 0	MIO 28 .. 39						
☐ USB 1							
✓ SD 0	MIO 40 .. 45						
✓ CD	MIO 47						
SD 0	MIO 47	cd	LVCMOS 1.8V	slow	ena...	in	
✓ WP	MIO 46						
SD 0	MIO 46	wp	LVCMOS 1.8V	slow	ena...	in	
☐ Power							
SD 0	MIO 40	clk	LVCMOS 1.8V	fast	dis...	inout	
SD 0	MIO 41	cmd	LVCMOS 1.8V	fast	dis...	inout	
SD 0	MIO 42	data[0]	LVCMOS 1.8V	fast	dis...	inout	
SD 0	MIO 43	data[1]	LVCMOS 1.8V	fast	dis...	inout	
SD 0	MIO 44	data[2]	LVCMOS 1.8V	fast	dis...	inout	
SD 0	MIO 45	data[3]	LVCMOS 1.8V	fast	dis...	inout	
☐ SD 1							

Figure 11. Zynq MIO configuration setting for SD card

After modifying the hardware part, I included generic FAT file system (FatFs) library in board support package setting, which provides functions to eliminate the complexity of dealing with low-level addresses of the files' blocks.

To use the FatFs library, I declare a global instance of file system (FATFS) type to be associated with the SD card. It is also necessary to create an instance of type file (FIL). Almost all the return values of functions provided by the FatFs library are of type FRESULT. Checking this variable can tell whether the FatFs API succeeded or failed. Using Figure 12 below, if some error occurs, I can tell exactly what goes wrong.



Value	Code	Meaning
0	FR_OK	Operation succeeded.
1	FR_DISK_ERR	A hard error occurred in the low level disk I/O layer.
2	FR_INT_ERR	Assertion failed.
3	FR_NOT_READY	The physical drive cannot work.
4	FR_NO_FILE	Could not find the file.
5	FR_NO_PATH	Could not find the path.
6	FR_INVALID_NAME	The path name format is invalid.
7	FR_DENIED	Access denied due to prohibited access or directory full.
8	FR_EXIST	Access denied due to prohibited access.
9	FR_INVALID_OBJECT	The file/directory object is invalid.
10	FR_WRITE_PROTECTED	The physical drive is write protected.
11	FR_INVALID_DRIVE	The logical drive number is invalid.
12	FR_NOT_ENABLED	The volume has no work area.
13	FR_NO_FILESYSTEM	There is no valid FAT volume on the physical drive.
14	FR_MKFS_ABORTED	The f_mkfs() aborted due to any parameter error.
15	FR_TIMEOUT	Could not get a grant to access the volume within defined period.
16	FR_LOCKED	The operation is rejected according to the file. sharing policy
17	FR_NOT_ENOUGH_CORE	LFN working buffer could not be allocated
18	FR_TOO_MANY_OPEN_FILES	Number of open files > _FS_SHARE

Figure 12. FatFs return values (FRESULT) [19]

The most important thing need to be done after declaring the above variables is initializing the SD card and mounting it with the FATFS instance I defined before. The next step is creating/opening a file with both read and write permission. Then I can read and write the number. During the whole program, I only created a single text file to store the string of the highest score. Besides, it is necessary to close the file when the function finishes reading or writing.

The functions I mainly used are:

f_mount(), f_open(), f_lseek(), f_read(), f_write(), f_close().

I also created a simple detection logic to deal with the situation when the SD card is not available.

Test:

By using the related functions, which I defined for SD card, I opened a file and wrote some text on an empty SD card. If the SD card is not inserted in the board, then the program will return a message on the terminal. After that, I shut down the board and turned on it again. If the application detects an SD card and successfully searches the file with the specific filename, then read the content and print it on the terminal. If the return string is the same as what I wrote in, it can prove the function works properly.

Problem encountered:

There was one problem I met when I tried to instantiate an object in FATFS type. The error is that the program complains of undefined reference for all the flash library functions even I included FatFs library already (the BSP has all the files). To fix this issue, there are two things need to do. The first one is including "ff.h" file manually in C code. Another step [20] is right clicking on the app in SDK and selecting C/C++ build settings. Then under ARM v7 GCC linker -> Inferred options -> Software Platform, add an extra flag like the following:

'--start-group,-lxilisf,-lxil,-lgcc,-lc,--end-group'.

4.0 General discussion

4.1. Integration and Problem

After the third milestone, my partner Sam and I spent one day integrating his audio core and my VGA core. Since then, we did integration every week and applied the unit test to ensure both functionalities are working. Since Sam was focusing on game logic and did not need to change hardware part in the later milestones, it is easy for us to perform integration by copying Sam's source code into my SDK project.

When we first tried to add Sam's audio IP block to my VGA project, we could not detect the imported audio block. Initially, we tried to debug by ourselves, we noticed that the project even cannot detect a new GPIO IP. We consulted with Dan and realized the problem is our wrapper cannot update automatically. The way for us to fix this was either manually add the new IP in the wrapper file or deleting the old wrapper and creating a new one.

Theoretically, the wrapper should update automatically, but we cannot figure why our project behaves like that.

4.2. Things I learned about the various tools

During this semester, I learned the following things about various tools:

- I learned how to create test bench for behavioral simulation in Vivado.

- When I searched online to find the solution to my problem, there were many codes are in Verilog. I used Google to understand and convert the code from Verilog to VHDL.
- The I/O planning section gives me a more concrete idea of the connection of the pin on the board.

4.3. Design Methodology

We divided the whole project into six milestones. Sam and I design each individual parts and components which the game will need. By using this method, we will be able to break down the whole task to several different smaller tasks and each of us can work individually and putting our work together will be convenient as long as we agree on the connection agreement. Another advantage of dividing the task using bottom-up design methodology is the ease of testing. Modular design allows us testing each small task's result and debugging each small task will be rational and well-founded.

I worked on VGA controller first, since the ability to display image on the monitor is the baseline of our game. After I successfully printing pixels on the monitor, I worked on implementing interrupts from push buttons. Defining deflected path of the ball, varying brick level, and ball speed had been done accordingly.

The game logic definitely could be improved by adding more specific rules. Since the program is written modularized, it is easy for us to add more new features if we have extra time. For example, I added logic to keep historic playing record, which allows viewing the highest score. Besides, I also increased the resolution of the game by using DMA to access the DDR3 through AXI bus.

4.4. Source code control

At the beginning of the semester, I backed up the whole folder every time I finished a single milestone. As time went on, the complexity of my task is increasing. I backed up the folder every time I finished a sub-goal.

4.5. Other used tools

When I tried to create text ROM as I mentioned above, (in section 3.4), I used sublime editor to process the VHDL file, using align cursor to add commas at once and converting them to array structure.

4.6. Methods used to ensure success

Every time I encountered a problem, I tend to use the search engine to find out the available solution. If I still cannot solve the problem, I talked to my partner first and then consulted Dan. Some other students in the lab are willing to help.

I always tried to start the milestone one week earlier, so I did not need to stay up to finish milestone before its deadline. This also provides me extra time to implement bonus part at the end of the semester.

My work is closely associated with hardware design, and the compiling time for a new bitstream requires a long time (sometimes half an hour for me). Therefore, time management is very important.

At the beginning of the semester, I tried to modify the block design of a working version in Vivado and it generated some errors about time analysis. After I redo everything, that block design just cannot work again. Since then, I will back up the whole folder whenever I achieve a sub-goal.

4.7. What did I learn

Previously I had experience showing images on a monitor through VGA while running UART driver on PetaLinux. This time I figured out the way implementing similar stuff on ZedBoard. However, this project helps me to better understand how the VGA transmits the data. It is the first time I noticed that I can process data in the blank area in order to save computing resource.

Initially, it is not quite clear to me how to receive the interrupt from hardware and handle it in software. Now I fully understood the way to implement the interrupt service routine after spending two days on that.

5.0 Community Contribution

The following list indicates my community contributions.

- I borrowed the code of VDMA driver from Wesley Kendall, I use his code to initialize and set up the VDMA block [7].
- Wesley and Steve helped me to debug the code about button interrupt overwritten issue.
- Isaac Qiao helped me to understand how the VGA's frame synchronous works.
- I helped Isaac Qiao to understand the interface for VDMA block and Stream FIFO to let VGA controller communicate with VDMA.
- I helped Isaac Qiao and Jason Liu to set up the GPIO in both PL and PS.
- I passed the IP block of LFSR to Jason Liu.

6.0 Feedback to Xilinx

The compiling time for a new bitstream requires a long time (sometimes half an hour for me), I hope there is a way to improve this.

7.0 Course feedback

- I would like to have 2 hours lecture and 1 tutorial per week.
- The 2 make up weeks works perfectly for me.

- I prefer to have lab exam to ensure the understanding of the tutorial since the tutorial is really useful.

8.0 Reference

[1] A. Fu, S. Wang, Group report
[2] Florent, “Create a VGA controller for the ZedBoard,” [Online].
Available: http://blog.dev-flow.com/en/11-create-a-vga-controller-for-the-ZedBoard/ [Accessed 15 January 2018].
[3] ZedBoard Hardware User’s Guide, Version 2.2.
http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf
[4] P.K. Babitha, T. Thushara, M.P.Dechakka, “FPGA Based N-Bit LFSR to generate random sequence number,” International Journal of Engineering Research and General Science Volume 3, Issue 3, Part-2 , May-June, 2015.
[5] VGA_mem_mapped. [Online]. Available:
https://github.com/delhatch/VGA_mem_mapped. [Accessed 15 January 2018].
[6] Zynq-7000 All Programmable Soc Technical Reference Manual,
https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf, p112.
[7] Wesley Kendall, VDMA https://piazza.com/class/jbenuzkh1151xw?cid=9. [Accessed 31 January 2018].
[8] J. Johnson, “Using the AXI DMA in Vivado,” [Online]. Available:
http://www.fpgadeveloper.com/2014/08/using-the-axi-dma-in-vivado.html.[Accessed 26 January 2018].
[9] “ov7670_VDMA_vga,” [Online]. https://github.com/dhytxz/ov7670_VDMA_VGA. [Accessed 03 February 2018].
[10] Zynq-7000 All Programmable SoC: Embedded Design Tutorial, Page 44. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug1165-zynq-embedded-design-tutorial.pdf
[11] Xilinx SDK Drivers API Documentation, scugic, [Online]. Available:
https://xilinx.github.io/embeddedsw.github.io/scugic/doc/html/api/.[Accessed 03 February 2018].
[12] GitHub. (2018). Xilinx/embeddedsw. [online] Available at:
https://github.com/Xilinx/embeddedsw/blob/master/XilinxProcessorIPLib/drivers/gpio/src/xgpio_intr.c [Accessed 10 Apr. 2018].
[13] Anon, (2018). OS and Libraries Document Collection. [Online]. Available at:
https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_3/oslib_rm.pdf. [Accessed 10 Apr. 2018].
[14] wikipedia. (2018). [online] Available at: https://en.wikipedia.org/wiki/Linear-feedback_shift_register [Accessed 10 Feb. 2018].
[15] Panda Amit K, Rajput P, Shukla B, “Design of Multi Bit LFSR PNRG and Performance comparison on FPGA using VHDL”, International Journal of Advances in Engineering & Technology (IJAET), Mar 2012, Vol. 3, Issue 1, pp. 566-571.
[16] GitHub. (2018). thelonious/vga_generator. [online] Available at:
https://github.com/thelonious/vga_generator/tree/master/vga_text [Accessed 3 Mar. 2018].

[17] Hamsterworks.co.nz. (2018). VGA timings - Hamsterworks Wiki!. [online] Available at: http://hamsterworks.co.nz/mediawiki/index.php/VGA_timings [Accessed 15 Feb. 2018].
[18] Eewiki.net. (2018). VGA Controller (VHDL) - Logic - eewiki. [online] Available at: http://www.eewiki.net/pages/viewpage.action?pageId=15925278#VGAController(VHDL)-Appendix:VGATimingSpecifications [Accessed 1 Apr. 2018].
[19] Embedded Centric. (2018). Data Logging using SD Cards. [online] Available at: https://embeddedcentric.com/data-logging-using-sd-cards/ [Accessed 30 Mar. 2018].
[20] Forums.xilinx.com. (2018). SDK undefined reference. [online] Available at: https://forums.xilinx.com/t5/Embedded-Development-Tools/SDK-undefined-reference/td-p/818300 [Accessed 15 Feb. 2018].