One-player Blackjack

A project write-up for ECE 383

by

C2C Dananga K. Agalakotuwa

May 5, 2024

1.1 Objective Statement

This will be a single-player game of Blackjack against the computer. The computer will act as the dealer.

1.2 Requirements

Basic functionality

- A player must be able to play a complete game of Blackjack against the computer.
- Use keyboard inputs through a UART terminal to implement game inputs.
- The HDMI Monitor must be used to draw the "cards" and simulate gameplay. Grid memory will be implemented to achieve this with each cell being 32x32 pixels.
- For this level, the deck of cards can be fixed instead of being random.
- *The UART feedback may be used for debugging and echoing gameplay.*

B-functionality

- Complete Basic functionality.
- An IR controller must be used to implement game inputs
- For this level, the deck of cards needs to be random, and cards cannot be repeated (2 Jack of Clubs for example).

A-functionality

- Complete both basic functionality and B-functionality.
- Include a slot-machine winning sound for beating the dealer with a screen depicting that the player has won the game

1.3 Level-0 Description & Top-Level Design

Overall Inputs:

Keyboard via USB
IR Controller via GPIO Pins

Overall Outputs:

Display on Monitor via HDMI output
Speaker via Audio output

Overall Behavior:

An IR controller button (PWR button) will be used to start the game. The game will initially be played using the keyboard (“H” key to hit and “S” key to stand). A new input device will be introduced later (IR controller) to control game inputs (One button to hit, Two button to Stand). The display will show cards as they are dealt (dealer’s initial card will not be revealed till end of game). Once the end of game has been reached (dealer is done hitting once a total of 18 or higher

is achieved with or without more cards for the dealer) the result of the game will be shown on the display (a screen with an image of a bag of money for player win, a screen with an image of a thumbs down for player loss).

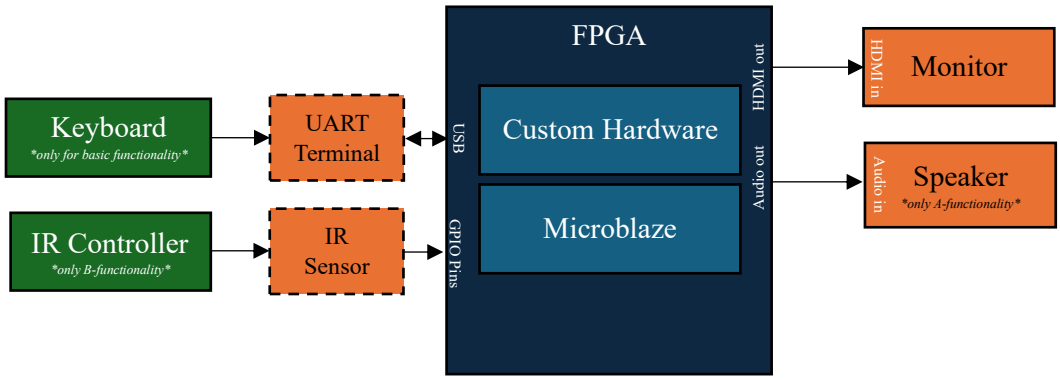


Figure 1: Level-0 Design

2 PLAN

2.1 PROPOSAL

I have corrected the mistakes pointed out to me by Dr York for Section 1, and included changes to functionality.

2.2 DETAILED ARCHITECTURE AND SUB-SYSTEM DESIGN

You need to provide the detailed design of your system. A detailed design should be split into level-1 subsystems, such as Datapath and control.

2.2.1 LEVEL-1 DESIGN

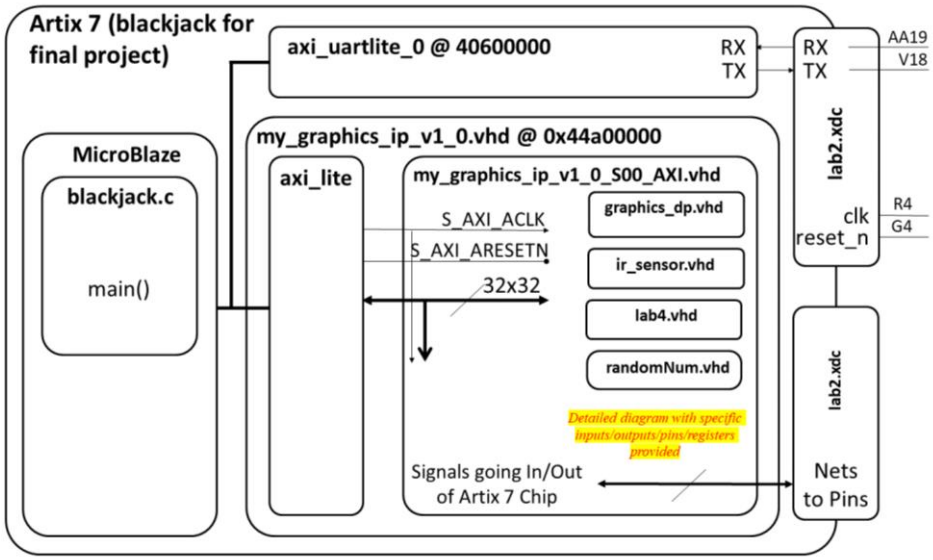


Figure 2: Microblaze Level-1 Design

Figure 1 shows the overall microblaze design which will be used for my final project. The specific registers written and read, inputs, outputs, and, pins for the .xdc file will be provided in Figure 2.

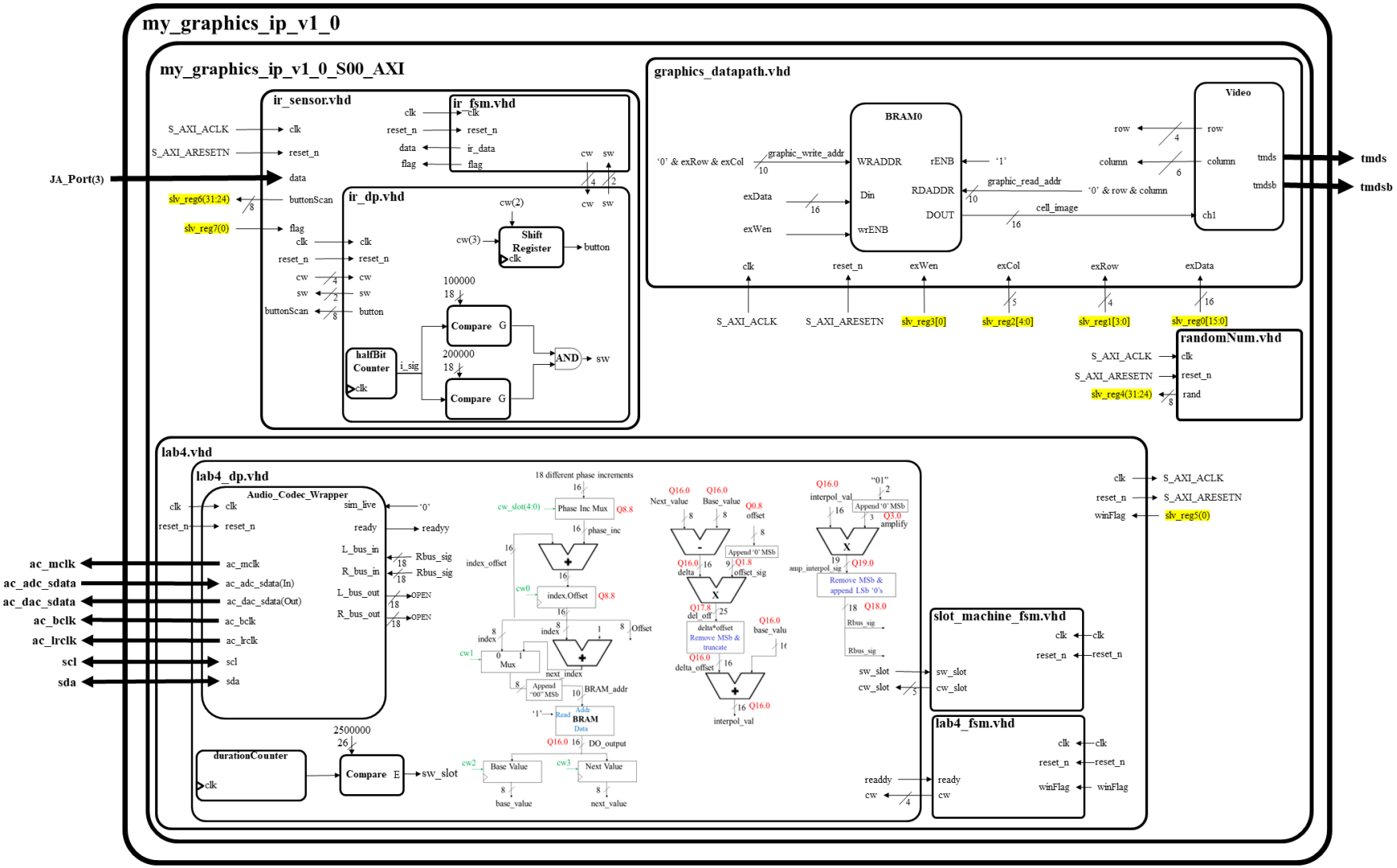


Figure 3: Level-1 Design

Figure 2 shows the four main components that I will be using to achieve the final project. I rely on the microblaze registers to run all these components, therefore, most of my "game code" will be in C-code. For better viewing of the image please see Github repo [ECE383_Dan → Final Project → README → Images].

MicroBlaze Registers [in = read; out = write]		
Signal	**Direction**	**Register/Bits**
exData	read	slv_reg0[15:0]
exRow	read	slv_reg1[3:0]
exCol	read	slv_reg2[4:0]
exWen	read	slv_reg3[0]
random	write	slv_reg4[31:24]
winFlag	read	slv_reg5[0]
buttonScan	write	slv_reg6[31:24]
IRflag	read	slv_reg7[0]
h	(internal: keyboard inputs)	
s	(internal: keyboard inputs)	

Outside Artix 7 (lab2.xdc)		
Signal	**Type**	**Package Pin**
clk	clock	R4
reset	button	G4
ac_mclk	Audio Codec	U6
ac_adc_sdata	Audio Codec	T4
ac_dac_sdata	Audio Codec	W6
ac_bclk	Audio Codec	T5
ac_lrclk	Audio Codec	U5
sda	QSPI	V5
scl	QSPI	W5
tmds[3:0]	HDMI out	T1 AB3 AA1 W1
tmdsb[3:0]	HDMI out	U1 AB2 AB1 Y1
data	Pmod header JA	AB18

Figure 4: Datapath signals to Microblaze registers

Grid Description	Index	Binary Value	BRAM Hex Value
heart	1	00001	0001
diamonds	2	00010	0002
clubs	3	00011	0003
spades	4	00100	0004
ace	5	00101	0005
two	6	00110	0006
three	7	00111	0007
four	8	01000	0008
five	9	01001	0009
six	10	01010	000A
seven	11	01011	000B
eight	12	01100	000C
nine	13	01101	000D
ten	14	01110	000E
queen	15	01111	000F
jack	16	10000	0010
king	17	10001	0011

back	18	10010	0012
one	19	10011	0013
dealerBox	20	10100	0014
playerBox	21	10101	0015
topBorder	23	10111	0017
bottomBorder	24	11000	0018
winBox	25	11001	0019
greenBox	26	11010	001A
loseBox	27	11011	001B
zero	28	11100	001C

Figure 5: Grid Memory

2.3 CALCULATIONS/ANALYSIS/DRAWINGS

Grid Memory Calculations

- I am implementing a 20x15 grid with each cell containing 32x32 pixels.

$$\frac{640pixels}{32pixels} = 20\ cells\ across$$

$$\frac{480pixels}{32pixels} = 15\ cells\ down$$

- My cell rows would be represented with 4-bits and cell columns would be represented with 5-bits. From this,

$$2^4 \times 2^5 = 300\ cells$$

- This means that my BRAM would have 300 entries.

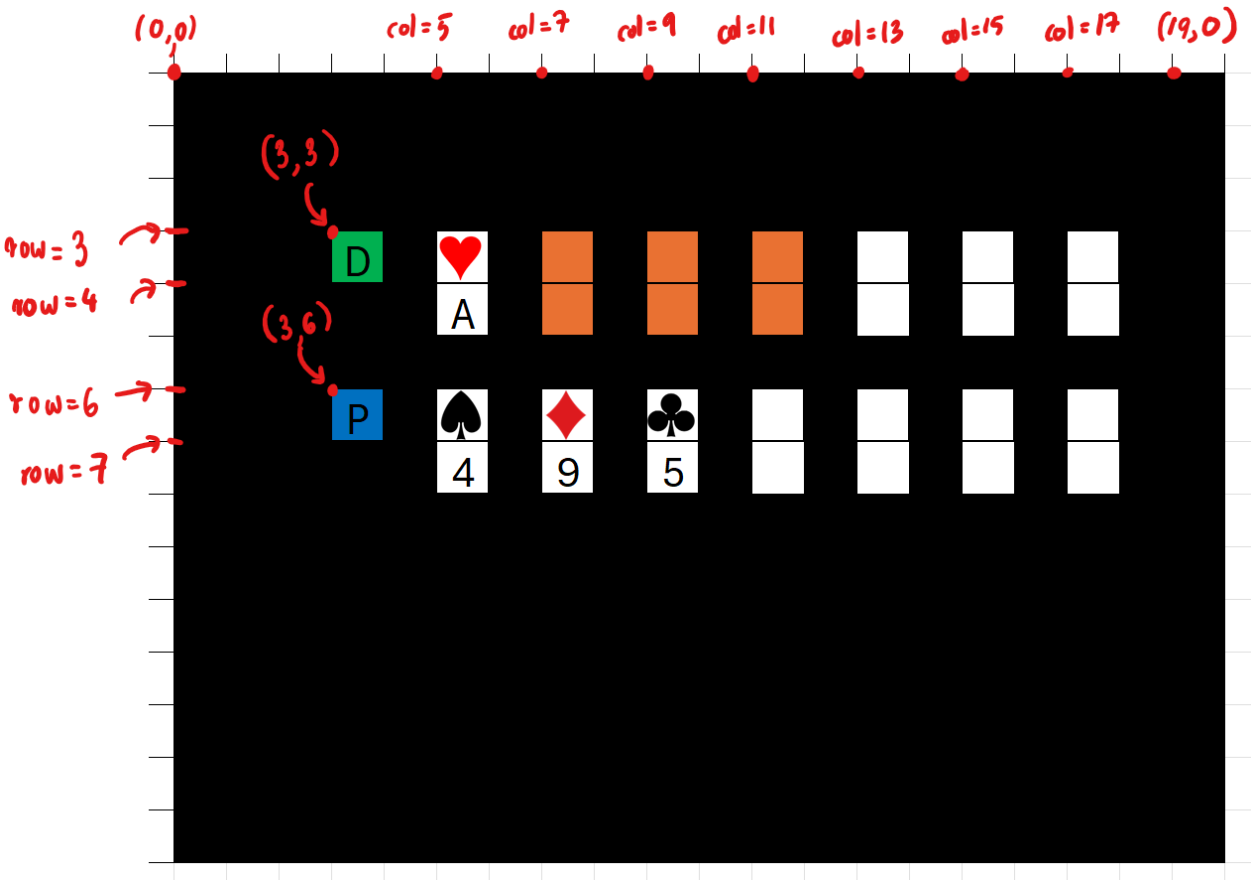


Figure 6: Gameplay Screenshot

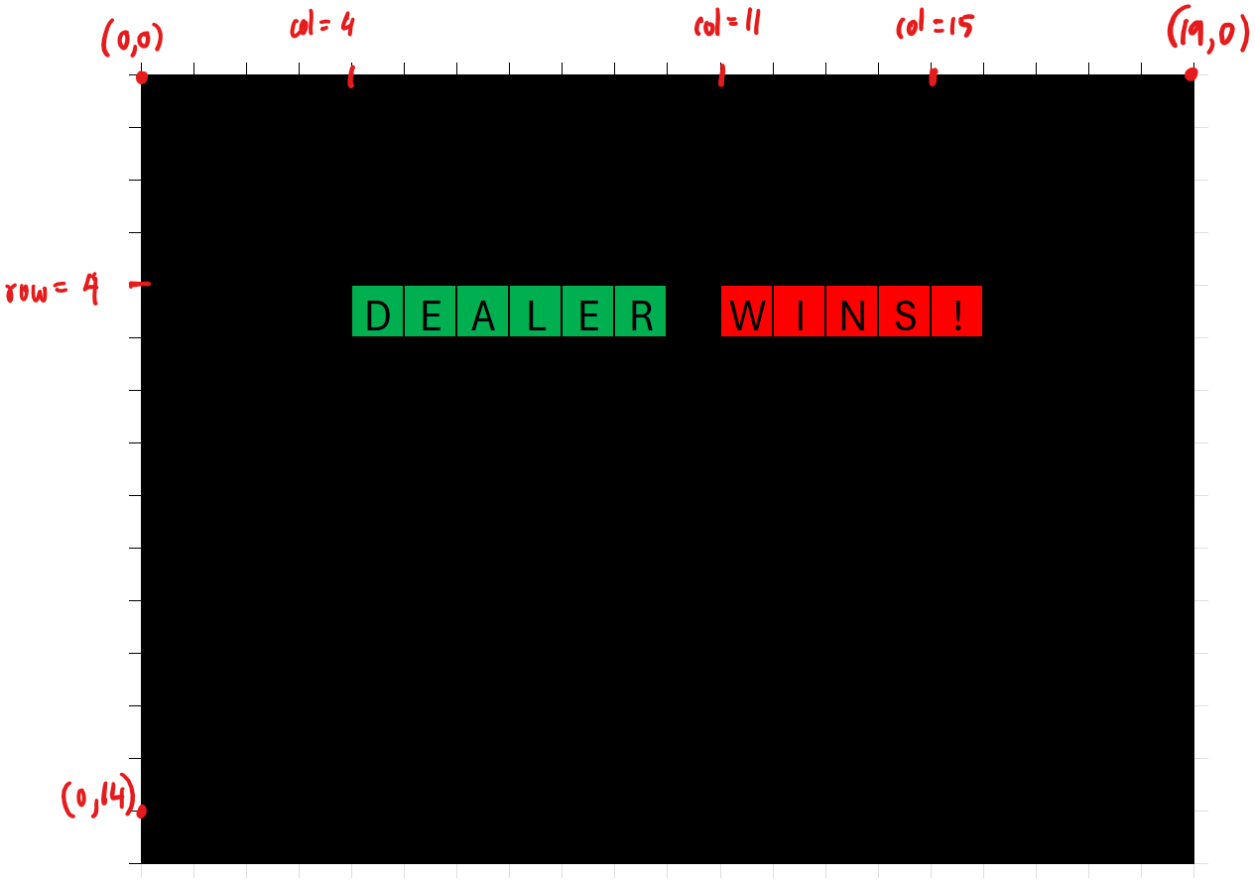


Figure 7: End result screenshot

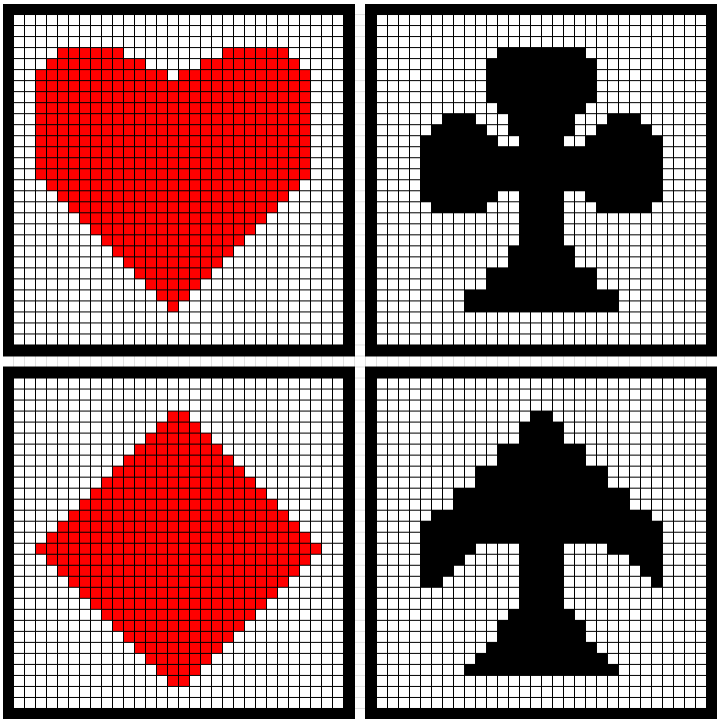


Figure 8: 32x32 pixel cells containing suits

Figure 5 shows the expected output of the display when the game is being played. The orange cards shown on the dealers hand are cards that are in the dealers hand (but the player cannot see them).

Figure 6 shows one of the result possibilities, this case where the Dealer wins.

Figure 7 shows the 32x32 pixel cells that would contain the suits of the deck of cards.

On figures 5, 6, and 7, I have only noted the most important coordinates that would be needed when interfacing this.

2.4 Milestone I

This milestone will mostly consist of putting all the necessary files together and setting up microblaze. After setting up microblaze and other files, as seen in the level-1 design, set up an array of a deck of cards and use SDK to test different combinations of cards being printed to the screen. You do not have to implement the game play yet, although encouraged.

2.5 Milestone II

Finish the gameplay code and ensure the game runs properly by testing with an unrandomized deck of cards and see if behavior is as expected. After this, set up gameplay to work with a random deck of cards, and ensure no cards will be repeated. You can use print statements on your .c file to see which card is used every time a “hit” is made by the player.

Have the IR sensor datapath and finite state machine tested in a separate Vivado project. This ensures that you only need to add these files to your AXI now.

2.6 Updated Functionality and Requirements

Major corrections included in Dr York’s feedback were updated in the Section 1.

3 Milestone 1

I was not able to achieve everything I wanted for this milestone. I got all my images to print on the display using a fsm. This was a test to ensure that the datapath behaves the way I want it to. Finally I got the bitstream file to work without errors to open SDK. However, I was not able to print what I wanted using the commands in sdk. All items uploaded to Github, image of screen also uploaded.

4 Milestone 2

I finished the gameplay code and ensured the game ran properly by testing with an unrandomized deck of cards. I could not yet implement random cards yet. I tried working on the PS2 mouse but ran into a bunch of issues like, the mouse being broken/not fully functional/not getting power. So, I gave up on that dream and set my mind to working on an IR controller. After talking to Dr York about it I think I have a pretty

good understanding of it and will be able to implement it quickly. The only thing I missed for this milestone was testing a random deck of cards.

5 Final Demonstration and Test Results

5.1 final demonstration

The final demonstration met all the requirements (Basic to A-Level). I was able to implement the IR Controller and play the game using button inputs of the IR controller. I was also able to add sound, playing a winning sound (slot machine sound). A full demonstration of my project was given to Dr. York in person, and I have also uploaded a video demo to the Teams channel, under the Put_Demos_Here folder.

5.2 test results

I did my testing separately and then integrated then all in microblaze for each component used; graphics, sounds, IR controller, and random number generator.

For the IR controller I hooked up the flag I was going to use in microblaze to a switch, and the IR data I received to the LEDs. This enabled me to compare what I was seeing in the LEDs to the logic analyzer and determine if my datapath and finite state machine was working accurately.

For the graphics I modified the finite state machine provided to us in the hints and used it to print each of my grid memory objects to the screen repeatedly. This saved me a lot of time in ensuring that I was drawing the right object.

For the sounds I created a separate Vivado project like the other testing. This enabled me to see if the sound I was intending to play, played.

For the random number generator, I did the same thing in creating a new Vivado project, and then hooked up the flag to a switch. I also hooked up the 8-bit random number to the LEDs. This enabled me to see if the random number generated was changing and if the number changed.

6 Presentation

I have uploaded the presentation to my Github.

7 Write up

Appendix A is attached below.

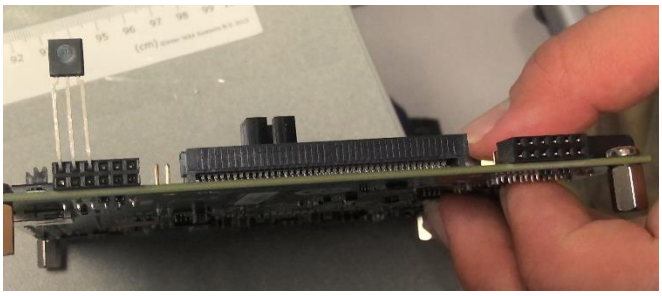
8 Git Repo

All folders are uploaded to Github.

Appendix A: Running the Project

Hardware Setup

1. Plug in the IR sensor to the GPIO pins PMOD header JA(3), VCC, and GND as seen below



2. Hardware is setup now, you are now ready to use the following IR controller

Software Setup

1. Go to the following directory in the Github repo

   ECE383_Dan → Final_Project → SOURCE → Code → v1 → open Vivado project

2. Run Bitstream
3. Export to hardware
4. Launch SDK
5. Once SDK launches, run the program