ECE532 - Final Report

Let's Dance

Team ID: 15

Yi Qing Li
Kevin Siu
Yinghui Fan

2017-04-13

Table of Contents

1. Overview

1.1 Motivation

Motion sensing and capture (detection and measurement of movement of objects) is a very interesting and challenging problem. Applications of this technology include security cameras and video games. Most notably, the Nintendo Wii remote controllers, the Microsoft Xbox Kinect and the Playstation Camera all incorporate some form of this technology. This technology has enabled a variety of dance-based video games, where the console detects the movement of the players. Inspired by these games, our team proposes “Let’s Dance”, a dancing game based off of motion detection. The motivation behind this is we want to build something that uses motion detection technology, but at the same time we can have fun doing it.

Unlike the Nintendo Wii (which uses remotes) or Xbox Kinect which uses body tracking, we propose using coloured bracelets and anklets to detect motion. This is very similar to a past project “Twist Up”, which required the player moving their hands and feet to a certain marked area on the screen. However, instead of static object detection, our project will do motion detection requiring the player to perform a certain dance motion. This is a more challenging problem, but we believe this makes the game much more active and fun for the player(s).

1.2 Goals

The goal for our project is to create a dance game based off of motion detection. The camera should successfully detect the movements of the player by tracking the position of the coloured anklets and bracelets and determine if the dance move was successfully completed. The display should indicate to the player the next dance move to perform. We believe this approach will prove advantageous over other dance-based games because there is no setup (just wear the bracelets/anklets), and it is very simple to pick up and play - making it perfect fun for a party.

1.3 Block Diagram

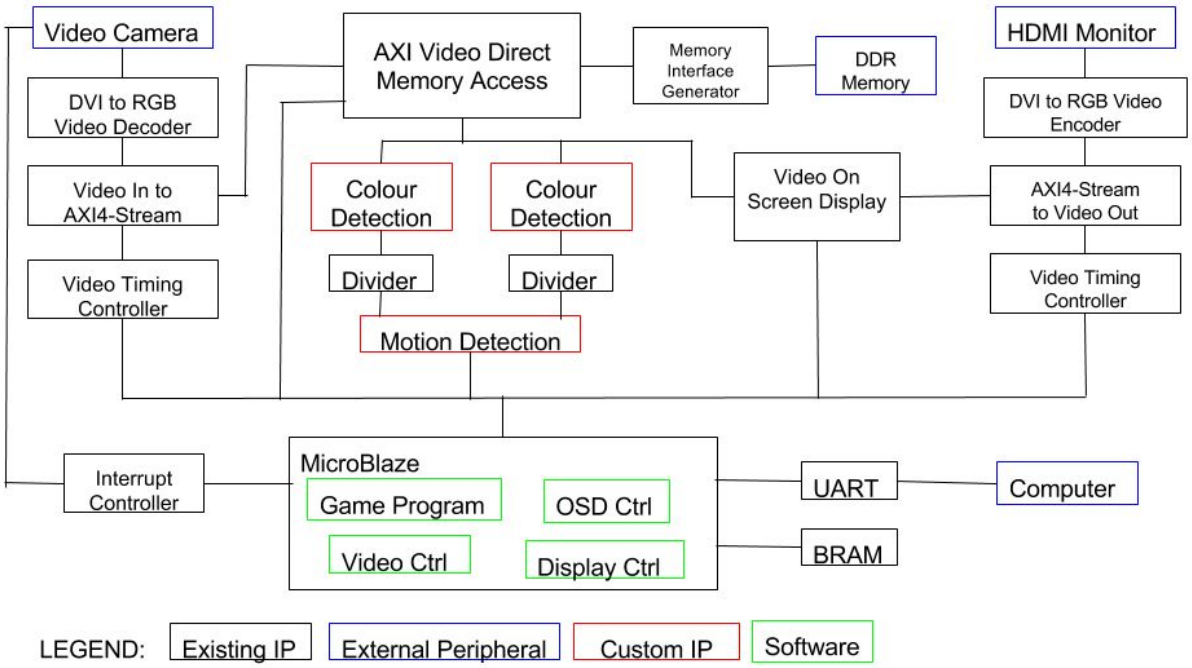

Figure 1 - System overview

1.4 Brief Description of IP

IP	**Description**	
DVI to RGB Video Decoder	A video decoder, which translates video signals into RGB data.	Existing IP
DVI to RGB Video Encoder	A video encoder, which translates RGB data into video signals.	Existing IP
Video in to AXI4-Stream	This IP core converts common parallel video signals to an AXI4-Stream interface. [1]	Existing IP

AXI4-Stream to Video Out	This IP core converts AXI4-Stream interface signals to standard parallel video output interface with timing signals. [2]	Existing IP
Video Timing Controller	A general purpose video timing detector and generator, which detects active data timing, and generate video timing signals. [3]	Existing IP
Interrupt Controller	Concentrates multiple interrupt inputs from peripheral devices to a single interrupt output to the system processor. [4]	Existing IP
AXI Video Direct Memory Access	Provides high-bandwidth direct memory access between memory and AXI4-Stream video type target peripherals including peripherals which support the AXI4-Stream Video protocol. [5]	Existing IP
Memory Interface Generator	Creates memory controller and physical interface for DDR memory.	Existing IP
DDR Memory	For buffering video frame.	External peripherals
Video On Screen Display	Provides for basic video overlay and blending functions for video system. [6]	Existing IP
Color Detection	A custom IP implements an iterative algorithm, which traverses through the entire frame to find the desired color, and then calculate the coordinates of the color block.	Custom IP
Divider	A pipelined division	Existing IP
Motion Detection	A custom IP detects the direction of the movements, given the coordinates of the object during motion.	Custom IP
MicroBlaze	This IP is a Xilinx 32-bit RISC Harvard architecture soft processor which provide high level control logic for the system. Microblaze is used in our project to control data flow of video streaming. [7]	Custom IP
UART	Testing and debugging purposes.	Existing IP
BRAM	Configurable memory module attaches to Microblaze through interface controller. It provides memory space required by the software component.	Existing IP

2. Outcome

2.1 Results

The final project meets the criteria proposed in the original plan with minor modifications, and the project achieves the features and functionalities proposed originally as well. Overall, the project has been successful. The details of the achievements and modifications can be found in the charts below.

List of functional requirements and features and the status of completeness

Function #	**Functional Requirements and Features**	**Status**
F1	A live video should be input through a video camera and displayed on a screen	Completed
F2	Coordinates of the colored bracelets and anklets in each frame should be correctly determined by the FPGA	Completed
F3	Movements of the player's limbs should be determined from a series of stored coordinates	Completed
F4 Original	The next dance instruction should be randomly selected from a pool of instructions and updated periodically	Modified, see below
F4 New	The next dance instruction should be taken out of a *pre-designed* sequence of instructions	Completed
F5	The next instruction and current score should be incorporated into video frames and displayed on the screen	Completed
F6	The match between the player's movements and the corresponding dance instructions should be determined by the FPGA	Completed

Acceptance Criteria and the final results

Function #	**Acceptance Criteria**	**Results**
F1	Video signal from the camera can be shown on the screen in real time and with good quality	Achieved
F2 Original	The FPGA can find the four different color blocks and distinguish them from the background, and the calculated coordinates of the the colored blocks should be within 20 pixels from the actual coordinates	Modified, see below
F2 New	The FPGA can find the *two* different color blocks at one time and distinguish them from the background, and the calculated coordinates of the the colored blocks should be within 20 pixels from the actual coordinates	Achieved
F3	The FPGA can determine the general trend of a player's movements, for example, right hand moving up	Achieved
F4	The next dance instruction changes once every 10-20 seconds, and the instruction pattern has relatively few repetitions of the same instruction	Achieved
F5	On the screen, the next dance instruction and current score are displayed on top of the video frames without changing the quality of the video	Achieved
F6	Within the period of one dance instruction, the FPGA can determine whether the player follows the current instruction based on the calculated movements of the player	Achieved

2.2 Future Steps and Improvements

One thing that might help to improve this project is to do more fine tuning of the parameters, and these tunable parameters include acceptable error range of color pixels and acceptable error range of movements. These parameters depend on the environment (eg. lighting) and the player (eg. distance between the player and the camera, size of the player).

The possible next step for this project is to add functionality to detect more complex movements and design a more complex dance.

3. Project Schedule

Week	Original Plan	Actual Result
1	- Implement and demonstrate the colour detection and motion detection algorithm working in software. - Make the coloured bracelets. - Build the Microblaze system. - Using HDMI (possibly based off of a reference design), stream video from the input to display.	- Implement pyramid algorithm and colour detection, and tests them in software. - Implement motion detection algorithm in software and verify functionality using sample coordinates - Implement the base design to get HDMI video streaming from the HDMI-in port to the HDMI-out port.

2	- Implement the pyramid and colour and motion detection IP in hardware.
	- Add additional blocks to the design including AXI VDMA, BRAM, UART and DDR.
	- **Testbench Demo:**
	  - Demonstrate the block level testbench used for verifying the functionality of the pyramid IP block
	  - Demonstrate the block level testbench used for verifying the functionality of the colour and motion detection IP
	  - Demonstrate the framework to be used for system level testing. Show how UART interface can be used to assist in debugging.
		- Implement pyramid and color detection ip blocks in Verilog, create tests and simulation.
		- Implement motion detection algorithm (for detecting moving up, down, left, right) as a custom IP
		- Add Video On Screen Display block to draw the graphical overlay onto the screen

3	- Integrate the completed colour detection custom IP blocks into the rest of the system. Perform validation, synthesis, and implementation.	- Video processing in software to detect the coordinates of the color in motion. - Create testbench for motion detection IP and test its

	- With the hardware for system complete, perform simple tests to determine whether the system is able to detect simple movements	functionality using sample coordinates - Update system to be capable of drawing text/boxes onto the screen (on top of the video) - **Testbench Demo:** - Demonstrate the block level testbench used for verifying the functionality of the colour and motion detection IP
4	- Fix any bugs or issues discovered from the testing in the previous week - Develop more complex test cases - Write the software for the gameplay	- New functionality added into color detection: for every enable high, 10 frames are processed automatically with certain delay in between frames. - Add new functionality of detecting more complex movements (including arcs) to the algorithm and implement and verify in software - Initial phase of system integration - added Colour IP block and Motion IP block into the rest of the system and synthesize.

5	- **Mid-Project Demo:**
	  - Demonstrate the program can detect a sequence of movements
	  - Demonstrate the overlay and game software working
		- Debug and testing of the entire system:
		  - Integration went rather smoothly
		  - Many bugs were found and fixed
		  - Majority of system working as expected - still some bugs to flush out
		- **Mid-Project Demo:**
		  - Demonstrate the program can detect a sequence of movements
		  - Demonstrate the overlay and game software working

6	- Integrate the software game program with the hardware system. Test and debug any	- Automate movement instructions - Add in all four colours - Update motion detection IP to

	issues that arise in the integration process. - Decide which features will be in and/or out	include more complex movements, integrate it with the entire design - More bug fixes
7	- Debug remaining issues. - Finalize the software for the game - **Final Demo:** - Demonstrate the entire system working - user should be able to play a game of “Let’s Dance” - Make a video	- Make the coloured bracelets - Clean up software code, add a more user-friendly interface - Recorded a video of gameplay

A common theme that appeared in our milestones were issues that we hadn’t originally accounted for in most of our planned work. Hence for the most part, our actual work turned out to be more inclusive and more objective than previously expected. This was mainly to do with our skillset improving from planning to implementation. By going over our table, it is noticeable that our planned work was more abstract and less focused than our actual work.

4. Description of the Blocks

4.1 Colour Detection

The colour detection IP blocks takes in RGB value and the xy-coordinate of all the pixels in a frame, one pixel at a time. The block is enabled by the motion detection ip, and when the enabling signal is high, it processes 10 frames every time. The frame sampling rate is set by the MicroBlaze via registers. The colour detection block then calculates the average xy-coordinate of a desired color set by the motion detection IP. The inferred coordinates are then passed to motion detection IP for further calculation. (See Appendix A for a snip of the core algorithm in software)

4.2 Motion Detection

The motion detection IP is responsible for finding a match between the movements of a player and the current dance instruction. This IP communicates with MicroBlaze processor by using reg0 and reg1. Reg0 is write-only, and it allows MicroBlaze to set the start/stop signal, dance instructions and color codes. Reg1 is read-only, and it signals MicroBlaze when the results are ready to be read, and it contains results indicating whether movements of each color match the instructions. When the motion detection IP receives a start signal from MicroBlaze, it also sends start signals and color codes to two color detection IP. It reads the coordinates from color detection IPs when the ready-to-read signal is high, and it sends stop signals when ten sets of

coordinates are received from each color detection IP. This motion detection IP processes the data from two color detection IP in parallel. The algorithm checks the direction of the movements, the range of the movements and the concavity of the movements (for arcs), and it also accepts a certain range of errors. Besides, the algorithm can detect the start and stop points of the correct movements, therefore, any kind of movements before or after the correct movements (within one dance instruction) will not affect the result. When results for both colors are ready, the results are stored in reg1 and the read-to-read signal in reg1 will be high. The MicroBlaze processor can therefore access the results.

4.3 Existing IP

The following section list the various existing IP blocks that were used in the design and provides a description of each explaining the source, their purpose, and their general functionality.

4.3.1 DVI to RGB Video Decoder and DVI to RGB Video Encoder

The **DVI to RGB Video Decoder** and **DVI to RGB Video Encoder** are IP blocks developed by Digilent. The DVI to RGB Video Decoder is used to convert the video data captured from the HDMI-IN port of the Nexys Video board from DVI (HDMI) to RGB format. The DVI to RGB Video Encoder is used to convert video data from RGB format back into DVI (HDMI) so that it can be output on the HDMI-OUT port of the Nexys Video board. These IP were provided as part of the Nexys Video HDMI demo which can be accessed here.

4.3.2 Video in to AXI4-Stream and AXI4-Stream to Video Out

The **Video in to AXI4-Stream** and **AXI-4 Stream to Video Out** are IP blocks developed by Xilinx [1, 2]. The Video into AXI4-Stream IP block converts the video data from an RGB format to the AXI4-Stream protocol. This is required because almost all of the video processing elements developed by Xilinx use the AXI4-Stream interface. The AXI-4 Stream to Video Out converts data from the AXI4-Stream protocol back to RGB.

4.3.3 Video Timing Controller

The **Video Timing Controller** is an IP block developed by Xilinx. It is used to give timing pulses to synchronize the horizontal and vertical pulses in the frames of the video stream. It is connected to the Video in to AXI-4 Stream and AXI4-Stream to Video Out IP blocks. Further details about this IP block can be found at [3].

4.3.4 AXI Interrupt Controller

The **AXI Interrupt Controller** is an IP block developed by Xilinx. It was used in our design to trigger an interrupt to software when the board detected that a video camera was connected. This way, the software can start feeding the video to the monitor. Further details about this IP block can be found at [4].

4.3.5 AXI Video Direct Memory Access

The **AXI Video Direct Memory Access (VDMA)** is an IP block provided by Xilinx. It is used to provide high bandwidth access to memory through the AXI4-stream interface. In our design, we use it to buffer our frames in memory. It receives the video-in data and writes it via the AXI4-stream interface to the memory. At the same time, it reads the frame data from memory and sends it to our video-out path. More details about this IP block can be found at [5]

4.3.6 Memory Interface Generator

The **Memory Interface Generator** is a configuration tool used to generate a specialized IP block that interfaces with external memory blocks (i.e. DDR). We use it in our design to connect the rest of the system to the DDR memory.

4.3.7 Video On Screen Display

The **Video On Screen Display** is an IP block developed by Xilinx. It can be configured to display multiple video streams, as well as draw text and/or boxes onto the screen (as seen in Figure 1 below). In this project, we configured this block to use two layers, the bottom layer being the video out stream and the top layer being the graphical overlay (the arrows and score in the figure below). The contents of the graphics layer can be controlled by software, and is done so by writing colour, font and string values to ‘banks’ in the block, and writing ‘instructions’ to draw the text. For specifics about this block, we point you to [6]

Figure 2 - Video On Screen Display Example. Video shown in background with arrows and score drawn on top.

4.3.8 Divider Generator

The **Divider Generator** is a configuration tool that Xilinx provides which can be used to generate a pipelined efficient integer divider. It is used in our design to calculate the average X and Y pixels of a colour. We use the Radix-2 mode, with divisor and dividend of size 32 bits. Refer to [8] for further details.

4.3.9 AXI UART Lite

The **AXI UART Lite** is an IP block owned by Xilinx. It provides a UART interface, which we use in our design for mainly debugging (writing values read from registers onto the terminal). For more details on this block, see [9].

5. Description of the Design Tree

The files for this project can be accessed through the following GitHub link:
https://github.com/KevinSiu1/G15_LetsDance

The structure of the GitHub repository is as follows:

src: This directory contains the project files and IP

- **ColorDetect2_1.0**: This contains the files related to the Colour Detection IP
  - **drivers/ColorDetect2_1.0**: This contains software drivers for the IP block

- **hdl**: This contains the top level Verilog wrapper and the AXI interface
- **src**: This contains the colour detection core and stream interface files

- **MotionDetIP_1.0**: This contains the files related to the Motion Detection IP
  - **drivers/MotionDetIP_1.0**: This contains software drivers for the IP block
  - **hdl**: This contains the code for the Verilog module

- **project:** This contains the Vivado project files
  - **hdmi.xpr:** This is the main Vivado project file
  - **hdmi.srcs/sources_1:** This directory contains the source files for all the blocks used in the design
  - **hdmi.sdk:** This folder contains the software files
    - **hdmi_wrapper_hw_platform_3**: This contains the .bit file used to program the FPGA
    - **test:** This directory contains the source code for the Let's Dance game.
    - **test_bsp:** This contains the board support files

docs: This directory contains the documentation related to this project

- **ECE532_G15_Final_Report:** This is the final report for the project (i.e. this file)
- **Final_Presentation.pdf:** These are the slides used in the final demo

video: This directory contains a video of the game in action :)

- **video.mp4:** Video showing the gameplay

6. Tips and Tricks

Below are a couple tips and tricks:

- Keep backups on the cloud. Sometimes your laptop’s disk will crash.
- Ensure individual components work (more or less) before integration - debugging an integrated system is 10x harder than debugging the individual components
- Test early and often

7. References

[1] "Xilinx Inc. Video In to AXI4-Stream," [Online]. Available: https://www.xilinx.com/products/intellectual-property/video_in_to_axi4_stream.html. [Accessed 14 April 2017].

[2] "Xilinx Inc. AXI4-Stream to Video Out," [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi4_stream_to_video_out.html. [Accessed 14 April 2017].

[3] "Xilinx Inc. Video Timing Controller," [Online]. Available: https://www.xilinx.com/products/intellectual-property/ef-di-vid-timing.html. [Accessed 14 April 2017].

[4] "Xilinx Inc. AXI Interrupt Controller," [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi_intc.html. [Accessed 14 April 2017].

[5] "Xilinx Inc. AXI VDMA," [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_vdma/v6_2/pg020_axi_vdma.pdf. [Accessed 2 February 2017].

[6] "Xilinx Inc. Video On-Screen Display," [Online]. Available: https://www.xilinx.com/products/intellectual-property/ef-di-osd.html. [Accessed 2 February 2017].

[7] "Xilinx Inc. Microblaze," [Online]. Available: http://www.xilinx.com/products/intellectual-property/microblazecore.html#overview. [Accessed 2 February 2017].

[8] “Xilinx Inc. Divider,” [Online]. Available: https://www.xilinx.com/products/intellectual-property/divider.html [Accessed 12 April 2017].

[9] “Xilinx Inc. AXI UART Lite,” [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi_uartlite.html [Accessed 12 April 2017].

8. Appendices

8.1 Appendix A

```
def ColorDetect(img, color):
    height, width, channels = img.shape
    sum_h = 0
    sum_w = 0
    counter = 0

    for i in range(0, height):
        for j in range(0, width):
            blue = img[i, j, 0]
            green = img[i, j, 1]
            red = img[i, j, 2]

            if ((color == 'red') and (red >= 170) and (green <= 60) and (blue <= 60)) \
            or ((color == 'green') and (red <= 90) and (green >= 120) and (blue <= 80)) \
            or ((color == 'blue') and (red <= 45) and (green <= 170) and (blue >= 100)) \
            or ((color == 'yellow') and (red >= 220) and (green >= 210) and (blue <= 160)):
                sum_h += i
                sum_w += j
                counter += 1

    if counter != 0 :
        avg_h = int(round(sum_h / counter))
        avg_w = int(round(sum_w / counter))
    else:
        avg_h = -1
        avg_w = -1

    return avg_h, avg_w
```

Figure 3 - a snip of color detection core algorithm in software