Just Dab

ECE532 Final Report
University of Toronto

April 13, 2017

<u>By Group 17:</u>
Maxim Antipin
Jonathan Chan
(Ted) Mingyi Jia

Table of Contents

1. Overview

“Just Dance” is a musical rhythm game released in 2009, a major commercial success with over 40 million units sold. Our project, “Just Dab”, is an adaptation of the game using image recognition techniques to detect and classify dance moves performed by the user.

The motivation behind the project was combining several aspects of hardware and software including a neural network, accelerometers and VGA output into a single functional system.

The game is played in the following way:

1. The user is instructed to make a particular dance move with on-screen instructions
2. The user performs the requested dance move
3. The game evaluates if and how well the dance move is performed, and assigns a score
4. The user continues to perform dance moves instructed by the game

Illustrated in Figure 1 is our implementation of the dance game:

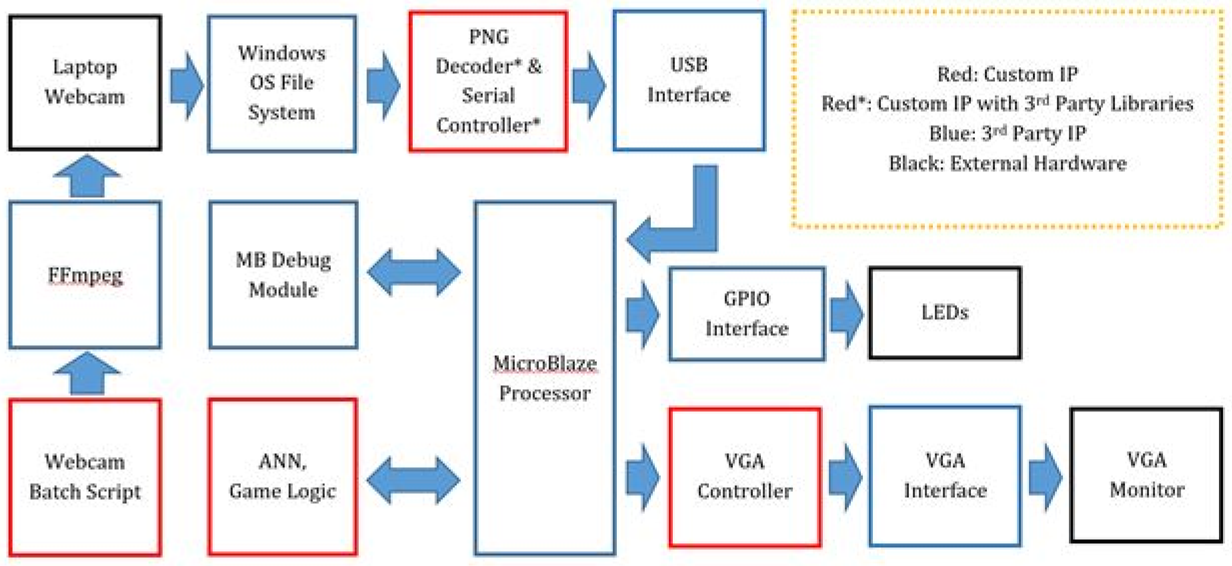


Figure 1: Block Diagram of “Just Dab”

The system consists of several custom IP blocks (red border) and 3rd party IP blocks that were adopted and integrated to suit the requirements of the project.

The following is a description of the flow of the system:

1. Webcam batch script executes FFmpeg and captures a 640x480 picture of the user, and scales it down to 8x6 resolution.
2. PNG decoder transforms the image to BMP format, extracting the red intensity values and normalizing it using the background behind the user captured during calibration.
3. Serial controller sends the pixel data to the FPGA through USB.
4. The neural network classifies the input as a dance move.
5. The game logic assigns a score, displays the score on the LEDs, and passes the next dance move to the VGA controller.
6. The VGA controller selects and displays the appropriate image.

2. Outcome

The final features of the project were similar to the original proposed features. There were several changes made to some of the modules, but the only major omission resulting from a lack of time to complete the module was sound output. The following is a comparison of the original features, and the resulting project:

Feature	Original Proposal	Final Result
Capture User Input	Use accelerometers to collect data about the dance move.	Use a webcam to take frequent images to collect data about the dance move.
Normalize Input Data	N/A – No normalization planned initially.	Convert PNG to bitmap, extract red channel values only, scale original image down to 8x6 pixel resolution, compress intensity from 0-255 range to 0-127 range. Subtract intensity values from background values acquired during calibration to help detect movement.
Classify Dance Move (No change)	Use a neural network to make a decision about which dance move (and how well) is being performed by the user.	
Game Logic (No change)	Have a sequence of dance moves presented by the game which the user must emulate. Every move is evaluated by the neural network and the game logic assigns a score.	
VGA Output (No change)	The game displays an image of the next move required to be performed by the user.	
Sound Output	Have a song playing to dance along to, along with “correct” and “wrong” dance move sound effects to give the user additional feedback.	N/A – Ran out of time to finish this module.

Table 1: Comparison of Proposed and Resulting Features

We didn't expect to require normalization of the input data, but that part of the project turned out to be one of the most significant adjustments made to improve the speed and accuracy of the system. By scaling the image down to 8x6 the transfer of the image to the FPGA takes less than a second, and subtracting the background values acquired during calibration allows the game to better detect motion and operate independently of the static environment behind the user.

The project works very well given a well calibrated environment. The neural network is not complex enough to accommodate various lighting conditions and variable distance between the user and the webcam. Therefore, the project was calibrated to work optimally at a distance of 6 feet between the user and the webcam in an environment with artificial light. In addition, there should be sufficient contrast between the background and the user's clothing and body to help the neural network perform accurately.

The following is a matrix of how well the system performs after collecting data for 200 dance moves, with the top row showing the move performed by the user, versus the leftmost column showing the move requested by the game. The value in each cell represents how often the game believes the dance move performed matches the requested move.

Accuracy Matrix	**No Move**	**Left Dab**	**Right Dab**	**Both Hands Up**
No Move	98%	0%	1%	1%
Left Dab	4%	93%	1%	2%
Right Dab	5%	4%	87%	4%

Table 2: Summary of true positives, true negatives, false positives and false negatives for dance move detection.

As it can be seen, when the correct move is performed the game is accurate on average about 92.6% of the time. When the wrong move is performed, the game correctly classifies it as the wrong move about 97.5% of the time. Therefore, false positives and false negatives both occur rarely.

The communication between the host computer and the FPGA is reliable, and has only failed a few times throughout the semester. We are not aware of the exact cause of the failure, but it is related to the Python script sometimes not being able to read the image file generated by the webcam.

The VGA and LED output also works well without any issues.

The system could be improved in several ways, including:

1. Using a different neural network topology more suited for the use case. The neural network implemented is a standard fully connected feedforward network. Other topologies such as locally connected networks are better suited for image recognition and processing which we did not have the time to experiment with.
2. The user experience can be improved by displaying the image captured by the webcam on the VGA to help the user understand why or why not their dance was correct.
3. Depending on the time available, it may be feasible to transfer the project to an HDMI board. This would require rewriting the VGA module to support HDMI output, but would allow a much higher bit rate in the input. This would help avoid compressing the captured data to a low resolution and reduce the latency associated with PNG decoding and normalization.

If we could start over, we would make better use of the available resources including Piazza. At the beginning of the project, our team used a significant amount of time trying to debug issues on our own instead of reaching out to the University of Toronto community through Piazza. When we finally started utilizing Piazza, we managed to solve bugs and problems faster allowing us to complete the project on time.

In addition, we would have begun integration much earlier instead of continuing to work on our own modules in parallel. Integration of different parts always introduces problems. In our case, a significant system level design change had to be made to accommodate the limitations of the system that were not apparent before integration. This cost us a considerable amount of time and put the the entire project at risk.

If another team of 3 would take over this project, we would recommend splitting up the work as follows:

1. Experiment with different neural network topologies, especially locally connected networks that are better suited for image processing.
2. Port the project to a HDMI board, allowing for higher bit rates transporting the information, which in turn reduces the latency, avoids compression of the data, and allows the display output to show the data captured by the HDMI camera.
3. Make the sound module functional, significantly improving the user experience.

All three recommended “next steps” can be performed in parallel and substantially improve the project over a span of 3-4 weeks.

3. Project Schedule

Milestone	Original Milestone Description	Actual Accomplishment
1 (Feb 10)	Setup MicroBlaze system along with basic components: - Run “Hello World” type program in C. - Basic VGA output, cycling the color of the entire screen between Red, Green and Blue. - Demonstrate basic test bench.	Setup MicroBlaze system along with basic components: - Run “Hello World” program in C. - Demo VGA output using VHDL. - Functional but buggy neural network in C.
2 (Feb 17)	Testbench demo: Demonstrate complete test bench with all ports defined.	- Created Vivado Debug Core to verify the VGA. - Partially completed test bench for neural network.
3 (Mar 3)	- Get accelerometers communicating to FPGA. - Design a Verilog IP that returns a binary value depending if a threshold for acceleration is reached within 1 second of request. Display basic output on VGA to give feedback to user when to move accelerometers. Also provide basic output through VGA for confirmation if threshold in acceleration is reached in time. - Build basic ANN in C, using a 6D dataset to confirm basic functionality (since we will have 6 dimensions of data from two accelerometers): Y = 1 + 5x + 3x^2 + 2x^3 - 2x^4 + 7x^5 - 3x^6	Testbench demo: - Accelerometers unfit for project due to short cables and high signal degradation with longer cables. -The display controller is written in VGA. - VGA can display simple colors. - Added “classification” to neural network. - Completed test bench for neural network. Using 4x4 BMP images for testing instead of 6D dataset.
4 (Mar 10)	Beginning of module integration: - Design a protocol for storing a “dance sequence” in memory: containing at least 3 different types of dance moves, starting at time = 0 and request to perform them at a specific time > 0. Add functionality to Verilog IP to read the “dance sequence” from memory. - Add functionality to VGA to show upcoming dance move. Refine graphics to look more	- Able to get input data from external WebCam in readable format for ANN. - Host system outputs data to USB-UART bridge. - Reads input data from USB-UART bridge and outputs on LEDs. - Used Python Script to generate verilog ROM from

	pleasant for the end user. - Collect data and fine tune ANN topology to perform binary classification: the dance move “running man” and other movements/no movement at all. Replace the basic acceleration threshold function from previous milestone with ANN output.	images. - Implemented those images into design. - Using C library to convert PNG images from webcam to BMP that can be fed to neural network.
5 (Mar 17)	Mid-project Demo: - Playback song along with “successful dance move” / “failed dance move” sounds during game. - Add functionality to Verilog IP to calculate the score of the user depending on number of successful moves. - Continue collecting data and fine tuning ANN topology to perform classification of two dance moves: “running man” and the “dab”.	Mid-project Demo: - Host system decoded WebCam PNG input data to BMP, neural network processes it, sends result to FPGA. Result displayed on LEDs.
6 (Mar 24)	Improving user experience: - Add functionality to select type of song and dance sequence at beginning of the game. - Continue collecting data and fine tuning ANN topology to perform classification of three dance moves: “running man”, “dab”, and the “whip”.	- Able to set the expected dance move to be displayed on VGA. - Able to detect left and right “dab” dance moves with improved accuracy. - Debug sending and reading bytes through the USB Uart port with UartLite. - Change system level design to perform PNG decoding on host computer, downscale neural network to fit in SDK.
7 (Mar 31)	Final Demo: - Complete system with all components integrated working reliably. - Polish presentation of the project for the end user. - Take care of low priority bugs or issues that have accumulated throughout the project.	Final Demo: - Game logic finished, score represented on LEDs. - Expected dance move shown on VGA. - Reworked game logic to have random dance moves. - ANN tuned to be more accurate.

Table 3: Changes in project schedule.

Given the major changes in the system level design of the project, it is expected that the proposed project schedule changed significantly. Setbacks at the beginning of the project significantly delayed overall progress, and although the end result is a functional system, it is missing the sound component and some of the planned user experience improvements including more than 2 dance moves along with selection of a particular song.

4. Description of Blocks

4.1 FPGA IP

FPGA IP includes all the IPs that are implemented on the FPGA, including the VGA Controller, UartLite Serial Data receiver, and the MicroBlaze System. This section does not contain the C code that is embedded into the MicroBlaze Processor.

4.1.1 VGA Controller

The VGA Controller gets an input from the Neural Network embedded MicroBlaze and then uses that input to select a stored image to display onto the VGA port. This VGA Controller System can be described in detail with Figure 2.

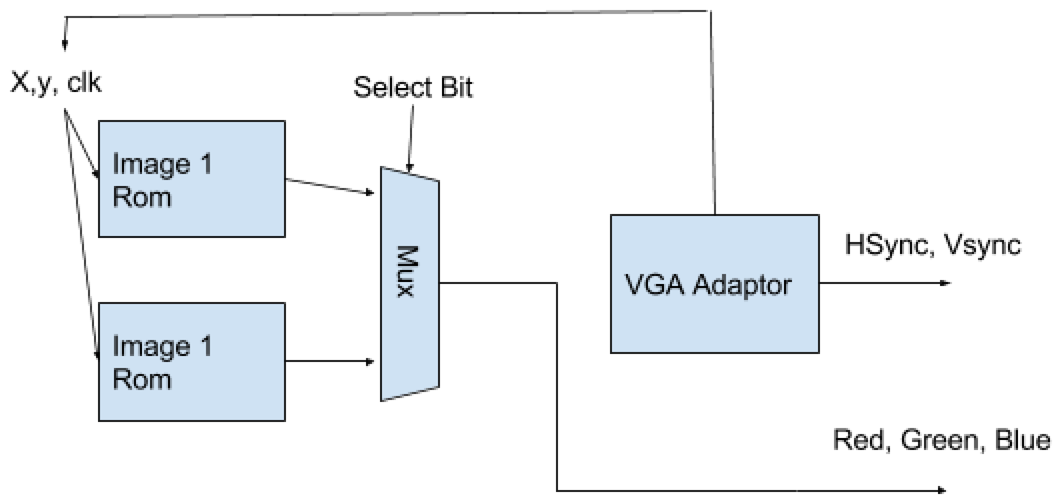


Figure 2: VGA Controller

The Neural Network will output a select bit that selects the color to be displayed on the VGA. The VGA Adaptor will generate the HSync and VSync signals for the VGA port based on the resolution selected. The Adaptor also generate a x and y coordinate for the two ROMs to select the correct pixel in those images. Then a mux is used to select the color to be outputted onto the VGA.

The design process of this part of the project started with testing the various parameters in the VGA Adaptor to display different resolutions. I learned through testing that not all resolutions will display on a monitor, it requires specific aspect ratios. I've also refreshed myself on how a VGA port should be driven with which pixel rate.

Due to the large amount of time it requires to synthesize large images that's stored on the ROM, I've decided to decrease the resolution of the image and the VGA display to as low as possible, in order to speed up the synthesis time.

To test the VGA Controller, I wrote tests in the SDK that can toggle the select bit to display different pictures on the VGA. The outcome of this test could be observed on the monitor. After this was completed, I added this IP into the MicroBlaze system to be integrated with the rest of the design.

4.1.2 UartLite
The UartLite is utilized to act as a receiver from the USB Uart Serial Port to receive the raw data coming into the FPGA from the external Webcam. The UartLite itself contains functions that can be called to distinguish each byte and can also send data as well. I have tested this receiver by sending in test text data into the USB port and printing out the data onto the console on the FPGA side. After I have ensured this worked, I integrated this with the other parts of the design to feed the data into the onboard C code.

4.1.3 MicroBlaze System
The MicroBlaze System was setup to contain the maximum RAM size possible to have enough room for both data and instructions in the system. The MicroBlaze processor links to several GPIOs to control the different components of our design. These GPIOs includes VGA, LED, and UartLite.

4.1.4 Vivado Debug Core
The Vivado Debug Core was an extra block created to verify the design during real time runs by probing into different links in between blocks. The Debug Core captures a few milliseconds of those signals and displays them for us to verify if the design is doing what is expected in actual runs.

4.2 External Blocks
This section contains the IPs that we used outside of the FPGA.

4.2.1 FFmpeg
The FFmpeg program is used for accessing a system's WebCam and can be used to record video or take pictures from the WebCam. FFmpeg also has the capability to scale the videos or pictures down to smaller sizes. FFmpeg on its own is able to capture pictures. For our project, FFmpeg is executed from a WebCam Batch Script and used for taking a picture and scaling the PNG image size down so that it can be decoded by the PNG decoder.

4.2.2 WebCam Batch Script
The WebCam Batch Script is a batch script created that executes the FFmpeg program to work for the purpose of our game. Here, the batch script is in an infinite loop that takes a picture

every 1 second, then scales the image down to an 8x6 resolution and saves it to disk on the host computer. It is necessary to scale the images down since the code for the neural network had to be optimized and reducing the image size also solved the problem of the slow 1 KB/s transfer rate between the host and FPGA.

4.2.3 PNG Decoder & Serial Controller (Host Side)
The PNG decoder makes use of a Python library to decode the PNG format into a bitmap format for easier processing. The IP block reads from disk the image captured by the webcam, converts from PNG to BMP and extracts the red intensity values, linearly scaling them from a range of 0-255 to 0-127. The last step is finding the absolute value of the subtraction between the current image and the background intensity values acquired during calibration. This normalization helps detect movement and makes the static environment behind the user independent of the game.

The same Python script also includes the serial controller, which casts the intensity values to the appropriate format and sends them to the FPGA through one of the COM ports.

4.2.5 Image to Verilog ROM converter
This was a third party Python script made to convert images to verilog ROM files. The ROM takes in a x and y coordinate, and outputs the color of the pixel at that coordinate.This converter was used to convert the images of the left and right dabs into verilog. Then they were instantiated in the VGA Controller to display them when the Neural Network calls them.

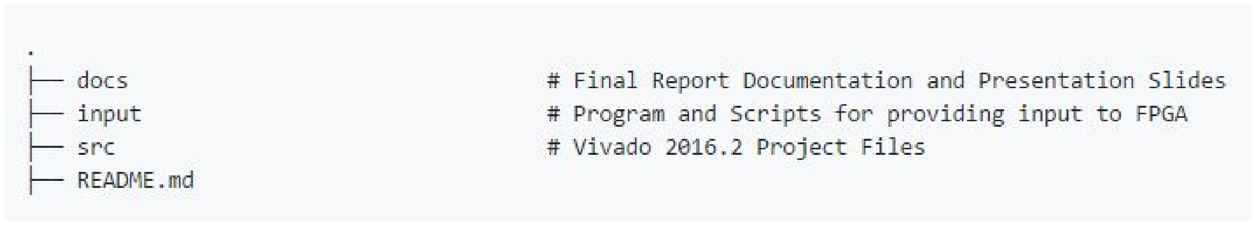
4.3 MicroBlaze Processor C Code

4.3.1 Neural Network
The neural network was written in C. The topology is a fully connected feedforward network, with 48 input layer units (representing the 8X6 pixel input image resolution), a single hidden layer of 3 hidden units, and an output layer with 3 hidden units representing a one-hot encoding of the probability the input is classified as a “left dab”, “right dab” or “no move”.

4.3.2 Game Logic
Game logic was integrated within the neural network code that would run on the MicroBlaze processor. Initially, the game logic was added to the python script and would send the next dance move to the FPGA to be displayed. Afterwards, it was decided to implement the game logic directly with the neural network code to reduce the uncertainty of sending the data through USB. The game logic serves as the main method of showing the player if a dance move was performed correctly. In its final state, the game logic is integrated within the neural network and randomly selects what the next dance move should be, selects the image ROM to be used, sets the VGA to output the next dance move, increments the score if the ANN detects the correct dance move, and selects the next unique dance move to be displayed.

5. Description of Your Design Tree

Included in the GitHub repository are the necessary Vivado Project files, IP blocks, 3rd party programs, and documentation for this final report. Our repo directory tree is as below:



```
.
├── docs                          # Final Report Documentation and Presentation Slides
├── input                         # Program and Scripts for providing input to FPGA
├── src                           # Vivado 2016.2 Project Files
├── README.md
```

Inside the “docs” directory is where the final presentation slides can be found, and where this document will also be found. The “src” directory contains the entire Vivado Design Suite project that is already synthesized, implemented, and bitstream already generated. All the IP blocks used for this project as described above are also found in the “src” directory. Lastly, the “input” directory contains a 3rd party program “ffmpeg” used for automating the process of taking pictures and also has the capability for scaling images down. Additional scripts created for the project can be found in the “input” directory as well for automating image capture and decoding the png files into a bitstream that can be sent to the FPGA.

Specific IP blocks that were created are found under “**src/project_4.srcs/sources_1/new**” which include:

1. **left_dab_12_bit_rom.v** - the “left dab” image used by the vga module.
2. **right_dab_12_bit_rom.v** - the “right dab” image used by the vga module
3. **vga.v** - main output module and controller for the image to display on VGA output

6. Tips and Tricks

1. Do research on the many devices that can be used for your project to make sure it fits the needs of your project
2. Team communication is key in order to complete the project since each team is limited to a single board and working in parallel without the ability to test can lead to problems
3. Ask questions on Piazza earlier than later, you will often waste a lot of time trying to figure something out that could be realized earlier.
4. Check resource limitations of the hardware such as SRAM size and program with those limitations in mind
5. Start early in connecting various parts of the project, so ensure the skeleton of the code works. After that, build on the various parts to improve functionality. A lot of the work is in interfacing the various parts.