ECE532 Final Report

A Wirelessly Controlled Home Automation System

Team 5: Peter Li Wenxuan Qiu Jing Bo Yang

April 14, 2018

Contents

1 Overview

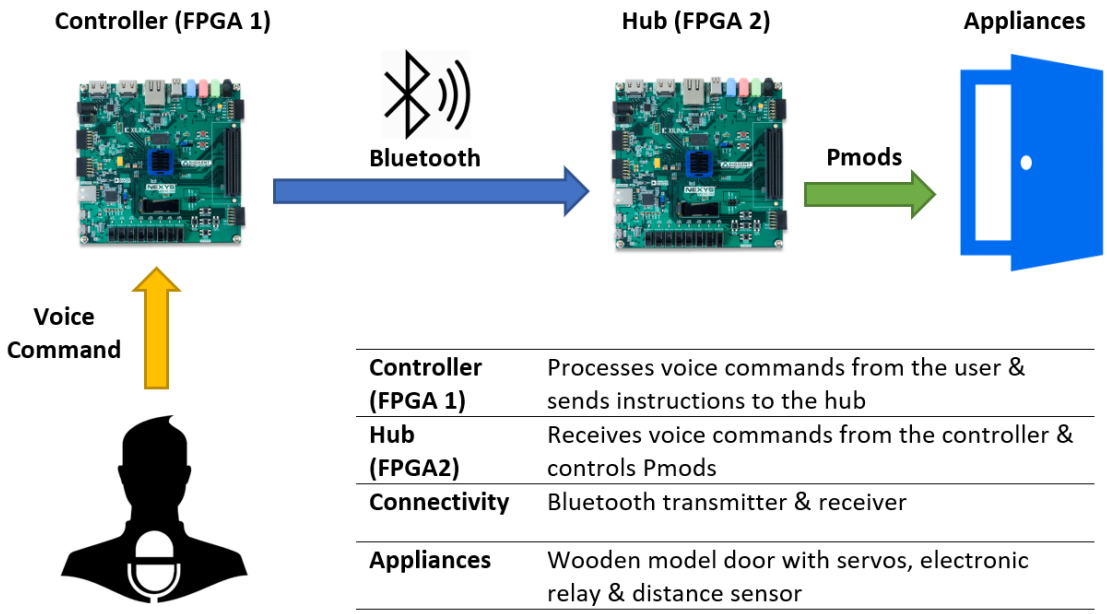
1.1 Motivation

The ever-shrinking size of computer components and the proliferation of high speed Internet have enabled us to connect more devices to the global network. Internet of Things (IoT) is a direct result of this explosion in the number of connected devices because we want not just computers and smartphones, but also home appliances like TVs and refrigerators to be connected. As technology continue to develop, we will no doubt increase our capability to monitor and control these appliances via personal mobile devices. The adoption of IPv6 protocol is one excellent example of how our society is accommodating ballooning number of network-capable devices. Numerous companies have already started rolling out gadgets like smart switches and smart LED lights that enable primitive levels of control. However, these applications are designed to target people who are familiar with technology; few implementations of these smart devices solve daily problems. In order to bridge this apparent gap, we designed a prototype voice activated home control system that goes extends the capabilities of current voice assistants like Amazon Alexa and Google Home. We envision that home assistants like ours will become prevalent in the near future.

1.2 Project Description

For this project, we built a home management system consisting of a controller and a hub (shown in Fig 1). The controller securely and wirelessly communicates with the hub unit, which connects and controls numerous appliances using sensors and actuators. In our project, the controller consists of a FPGA connected to a number-pad for entering control options and a microphone for receiving voice commands. This controller communicates to a hub through wireless network, which then controls various PMOD modules connected to door actuators and lights via another FPGA. A person using the controller can activate these devices using pre-defined audio commands. Additionally, as the person is about to enter the room, the door unlocks / opens, and lights turn on automatically, thus fully automates several desired functions of the home appliances in a sequence. With our design, this system not only allows people to have complete remote control of their appliances, but also enables hand-free interaction between people and appliances. Our voice controlled system is a peek into the future, in which voice assistants will wield more control of their physical surroundings.

For the next three subsections, block diagrams for the controller FPGA board, hub FPGA board and the MFCC audio recognition system will be briefly discussed.



Controller (FPGA 1)	Processes voice commands from the user & sends instructions to the hub
Hub (FPGA2)	Receives voice commands from the controller & controls Pmods
Connectivity	Bluetooth transmitter & receiver
Appliances	Wooden model door with servos, electronic relay & distance sensor

Figure 1: Block diagram for the proposed IoT home automation system [1][2][3][4]

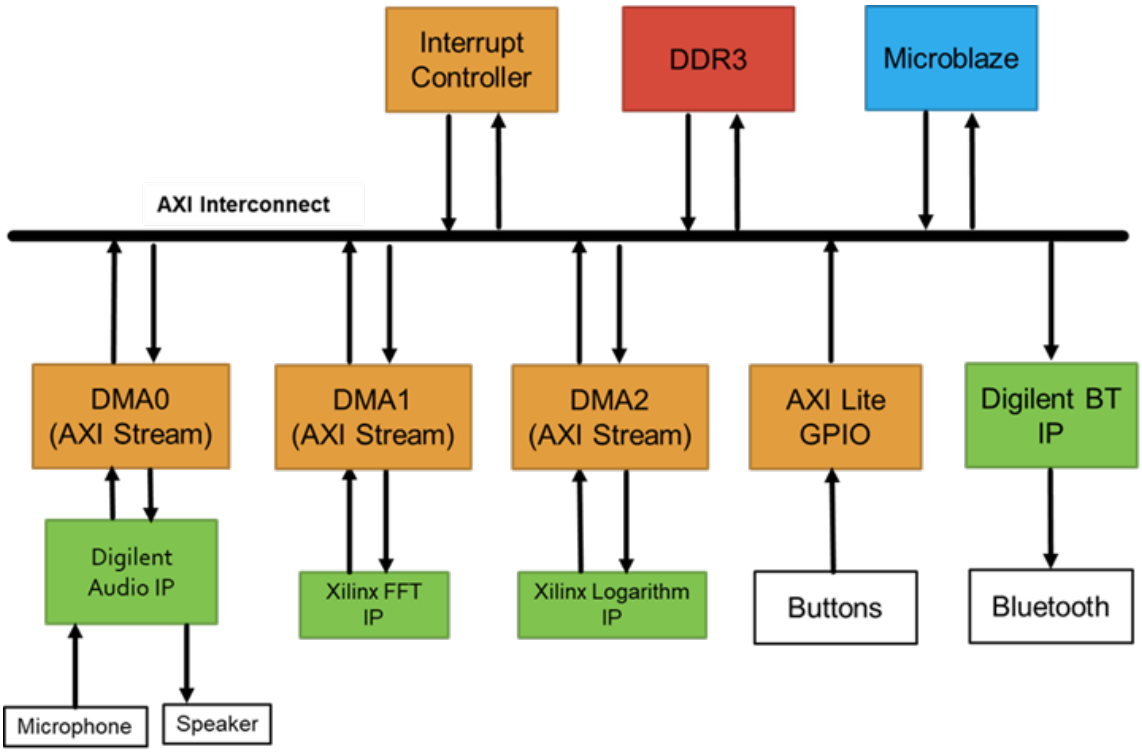


Figure 2: IP block diagram for the controller board

1.3 System Summaries

1.3.1 Controller Board Configuration

The job of our controller board is to provide a processor-based computing system (Microblaze, DDR3) that can record audio (Digilent audio IP) and perform audio recognition with help from various hardware-accelerators (FFT IP, Logarithm IP). In addition, the controller board contains Bluetooth transmitter with its associated Digilent Bluetooth IP so that the decoded audio commands can be sent to the hub FPGA. The entire controller board system is constructed from IPs coming from Xilinx and Digilent IP catalog. The controller board block diagram is shown in Fig 2.

1.3.2 Hub Board Configuration

The purpose of having a hub board is to provide a simple control system that receives commands from the controller FPGA and reacts to those commands appropriately. With help from FPGA, all functions could eventually be designed to execute in parallel. The hub contains two servos that control the door and the lock, respectively. A solid state relay switch is used to turn on and off LEDs. Finally, an ultrasonic sensor is used to trigger a series of events when the appropriate command is issued and relevant conditions are met. The hub system contains a mixture of custom IPs and IPs from the Digilent IP catalog. The controller board block diagram is shown in Fig 3.

1.3.3 MFCC Audio Recognition System

The audio recognition algorithm implemented on our system is commonly used for academic research. After preprocessing of recorded audio and audio segmentation, MFCC features are extracted to match against a library of stored audio samples. For this algorithm, we have first implemented processing routines like voice activity detection, FFT (Fast Fourier Transform), DCT (Discrete Cosine Transform) and DTW (Dynamic Time Wrap) in software, and designed simple interface for replacing them with hardware accelerators. A detailed diagram demonstrating the structure of our audio recognition system, along with hardware components, is shown in Fig 4. Our system had been tested in Python prior to implementation in C to for easier data representation and faster debugging.

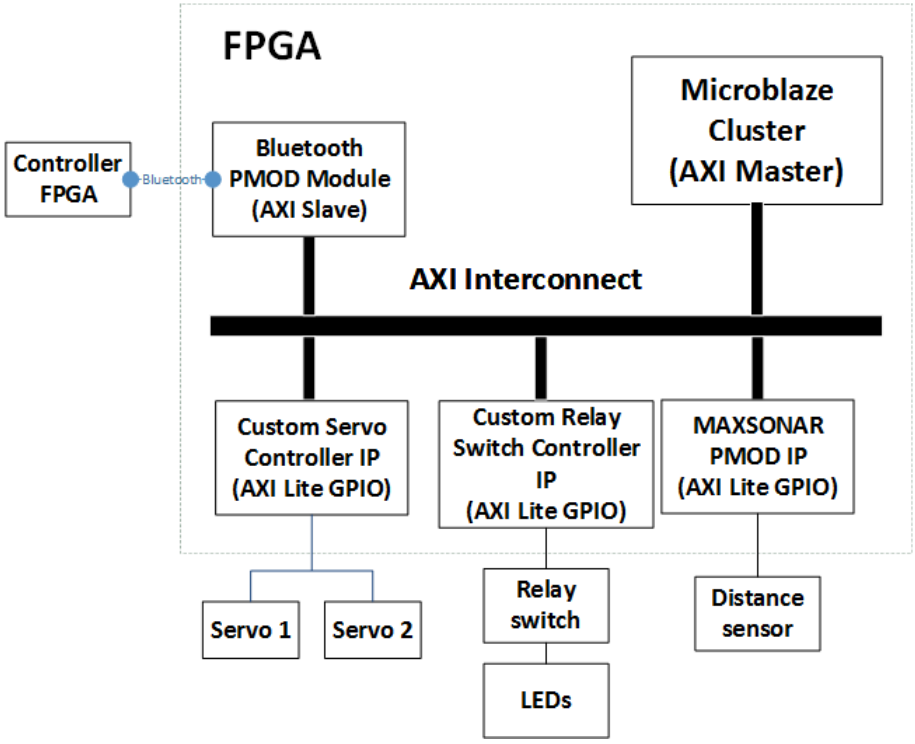


Figure 3: IP block diagram for the hub board

2 Outcome

For project demonstration, we built a prototype hinged door that is powered by a servo and a relay-swtich controlled LED light. We also used a condensor microphone to represent input device for the voice assistant. Our demo was within our expectation. We were able to correctly recognize all support voice commands, transmit them wirelessly from the controller FPGA to the hub FPGA, and use the decoded commands to control various sequences of servo and ultrasonic actions on the physical door model.

Since we spent too much time testing and verifying the reliability for the completed implementation of all of our design components, we did not have sufficient time to further improve the efficiency of our audio recognition algorithm. As shown in Table 1, our audio processing accelerator can be expanded to include more routines that *Microblaze* is not well-suited for.

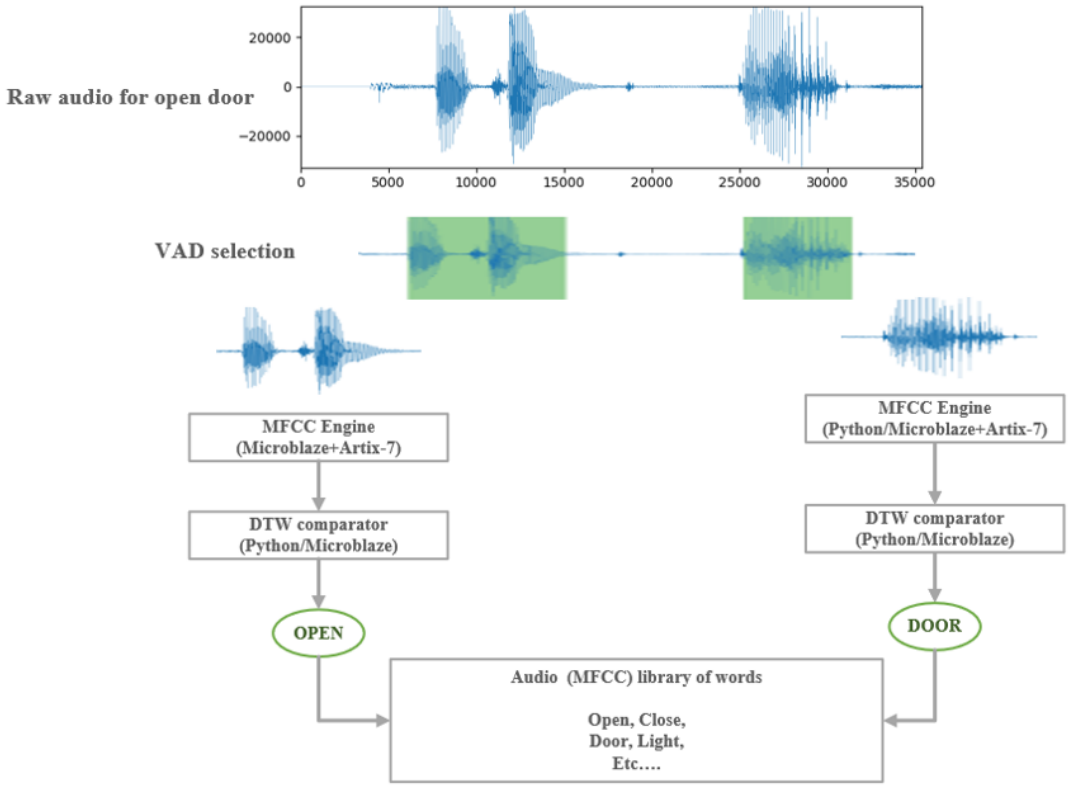


Figure 4: Block diagram for audio recognition algorithm

2.1 Design Components

In this section, we list the design components that we planned to implement and their completion status. Since we had been conservative for project proposal, we were able to complete most tasks listed in the project proposal. Tables outlining status for design components and proposed functional requirements are presented as Table 1 and Table 2. Implementation details are presented in Section 4.

Target	Status
Controller FPGA Board	
Bluetooth communication	*COMPLETED*
Microphone recording	*COMPLETED*
Audio processing accelerator	Can be improved FFT and logarithm modules
Audio processing software *More details below*	*COMPLETED*
Audio Recognition Software	
Python prototype for the algorithm	COMPLETED With helper to generate C code
C (Microblaze) port of Python prototype	COMPLETED Can be compiled on Visual Studio with simple switch
Hub FPGA Board	
PMOD control for servo motors	*COMPLETED*
PMOD control for relay swtich	*COMPLETED*
PMOD control ultrasonic sensor	*COMPLETED*
Communication and control logic	*COMPLETED*

Table 1: Status of critical design components

Target	Status
Controller FPGA Board	
All appliances can be controlled the user	*SATISFIED*
User can activate the controls of each appliances using audio commands	*SATISFIED*
Only authorized users can activate the appliances	*NOT SATISFIED*
Connection between the controller and the hub is wireless and stable	*SATISFIED*
Door must open and close in less than 10 seconds	*SATISFIED* Excluding audio processing time
Door opening and light turning on must be one sequence that can be enabled via the controller	*SATISFIED*

Table 2: Compliance with proposed functional requirements

2.2 Possible Improvements

As mentioned before, we would like to improve the processing speed of our audio processing algorithm. We have fully studied the capabilities of *Microblaze*, from which we learned that *Microblaze's* floating-point processing capability cannot be relied upon. The best way to speed up our audio processign algorithm is to implement a variable-sized matrix multiplier in hardware. Optimization has to be considered to balance usage of *Nexys Video Board*'s limited number of DSP units and performance. Of course, we could migrate the entire C-based algorithm to hardware to extract maximum performance from FPGA. This transfer could be completed with help from HLS (High Level Synthesis), but will require learning more about HLS design.

In addition to performance improvements, the audio recognition system could also be connected to an online database of words. Similarly, the hub board can also be designed to retrieve from a database of "drivers" for additional actuators/sensors. This would enable our system to have *Amazon Alexa*'s skill-like capability. We could also enable live training of audio segments to recognize user-defined words. Of course, this function requires a more complex control system along with Internet communication capability.

Another potential area of improvement is reliability of the ultrasonic sensor. During testing, we realized that the ultrasonic sensor frequently return erroneous readings. The inaccuracy could be attributed to the lack of filter circuit on *Diligent's* ultrasonic PMOD. To address this concern, we could attach an *Arduino* to the hub FPGA. Using an Arduino not only increases the total number of actuators/sensors we can attach, but also adds filter circuits for ultrasonic sensors. From past experience, ultrasonic readings on *Arduino* are mostly reliable and stable. We could also design custom filter circuits to avoid the additional software complexity (I2C communication protocol) associated with attaching an *Arduino* slave.

3 Project Schedule

3.1 Initial Project Schedule

Table 3 shows a time-line that we proposed at the beginning of the term. Notice that we have included several time-slots for testing and debugging to account for potential issues and restrict the scale of our design.

Week Start	Deliverable	JY	PL	WQ
(1) 01-22	Lab Test	Build software to support 7-seg display (part1)	Build "keep largest" Verilog module (part2)	Investigate board manual and build hardware support for 7-seg display
(2) 01-29	Proposal/Demo	Connect Nexys Video board to host computer	Build custom IP for buttons and switches. Verify with simulation	Integrate custom IP, ethernet connection and Microblaze C code
(3) 02-05	Milestone-1	Simulate the proposed audio-processing algorithm on Python with basic mathematic operations (ready to "transcribe" to Verilog)	Build bluetooth communication module and test it with plain message (no protocol)	Design and build an aparatus for hosting the proposed servo+sensor combination. Electrically connect servos and sensors to the FPGA boards.
(4) 02-12	Milestone-2	Establish a communication protocol for communication between controller and hub	Enable microphone recording on FPGA so that the waveform is ready to be processed	Build Verilog modules to enable control of servo and sensors
(5) 02-19	Reading week	Join PL to start building custom IP block (algorithm)	Continue to enable microphone recording. Start building custom IP block to process the waveform (interface)	Complete physical construction of the door+servo+sensors aparatus. Join PL to start building custom IP block (integration)
(6) 02-26	Milestone-3	Same as reading week	Same as reading week	Same as reading week
(7) 03-05	Milestone-4 (mid-project presentation)	Build necessary Microblaze support algorithms.	Complete implementing basic data IO for custom block	Help JY with transfering algorithm. Integrate custom IP block
(8) 03-12	Milestone-5 (in-lab demonstration)	Debugging - Ensure custom audio recognition/verification module is receving correct information	Debugging - Ensure data transferred from microphone to custom IP block is correct	Debugging - Ensure servos and sensors can be fully controlled by the FPGA. Build necessary Microblaze/Verilog drivers.
(9) 03-19	Milestone-6	Integrate audio processing algorithm with WQ's PMOD drivers.	Help with WQ to build PMOD drivers	Continue building PMOD drivers
(10) 03-26	Milestone-7	Debugging and integration	Debugging and integration	Debugging and integration

Table 3: Initial project timeline. JY=Jing Bo Yang. WQ=Wenxuan Qiu. PL=Peter Li.

3.2 Actual Project Schedule

Table 4 shows the actual milestone schedule throughout the semester. Since our initial project schedule allocated extra weeks for debugging and testing components (Week 6,8 and 10), we were able to complete all aspects of initially planned design. Thus, our actual overall accomplishments is very similar to the ones presented in the initial project schedule, minus some setback in terms of computational efficiency. Some order of finishing the proposed tasks were changed compared to the initially planned ones.

Week Start	Deliverable	JY	PL	WQ
(1) 01-22	Lab Test	Built software to support 7-seg display (part1)	Built "keep largest" Verilog module (part2)	Investigated board manual and build hardware support for 7-seg display
(2) 01-29	Proposal/Demo	Connected Nexys Video board to host computer via Ethernet	Build custom IP for buttons and switches for the warm up demo.	Integrate custom IP, Ethernet connection and Microblaze C code
(3) 02-05	Milestone-1	Simulated the proposed audio-processing algorithm on Python with basic mathematic operations (ready to "transcribe" to Verilog)	Investigated Bluetooth transmission and auto-connect protocols based on Digilent IP and tutorial	Designed an aparatus ("door") for hosting the proposed servo+sensor combination.
(4) 02-12	Milestone-2	Finished writing DTW and a custom VAD; started investigating Xilinx FFT IP	Successfully enabled 2 Bluetooth PMODs to auto-connect and communicate with each other via AXI-uartlite drivers	Connected servos to the FPGA and tested their functionalities. Revised the door design.
(5) 02-19	Reading week	Started porting Python code to C	Enabled audio recording and playback triggered with button interrupts. BT PMOD reliability testing.	Built Verilog modules to enable control of servo motors and the relay switch.
(6) 02-26	Milestone-3	Finished code migration to C. Tested read/write with DMA IP	Same as reading week	Same as reading week
(7) 03-05	Milestone-4 (mid-project presentation)	Prepared for mid-project presentation	Investigated different configurations of the Xilinx FFT IP and the DMA IP for AXI-Stream	Worked on physical construction of the door
(8) 03-12	In-lab Demonstration	Tuned features for mid-project demo (performance and accuracy improvements)	Fixed the sampling-rate inconsistency issue in the audio codec. Prepared hardware for the demo	Complete physical construction of the door+servo+sensors aparatus.
(9) 03-19	Milestone-6	Integrated hardware accelerators. Tweaked Microblaze settings for performance	Finalized FFT and Logarithm IP integration using DMAs	Built custom PMOD drivers for controlling the servo motors, relay switch, and sensor
(10) 03-26	Final Demo	Massively improved accuracy through debugging. Added normalization.	Debugging and integration	Debugging and integration

Table 4: Actual project timeline. JY=Jing Bo Yang. WQ=Wenxuan Qiu. PL=Peter Li.

4 Description of Design Components

Detailed description for each of the IP blocks used in the controller FPGA, hub FPGA and the MFCC audio algorithm are discussed in each subsections below.

4.1 IPs in the Controller FPGA

All IPs on the controller FPGA board are listed in Table 5 and come from Xilinx's IP catalog or Diligent's resource center. The major interface connections among all IPs are shown in Fig 5.

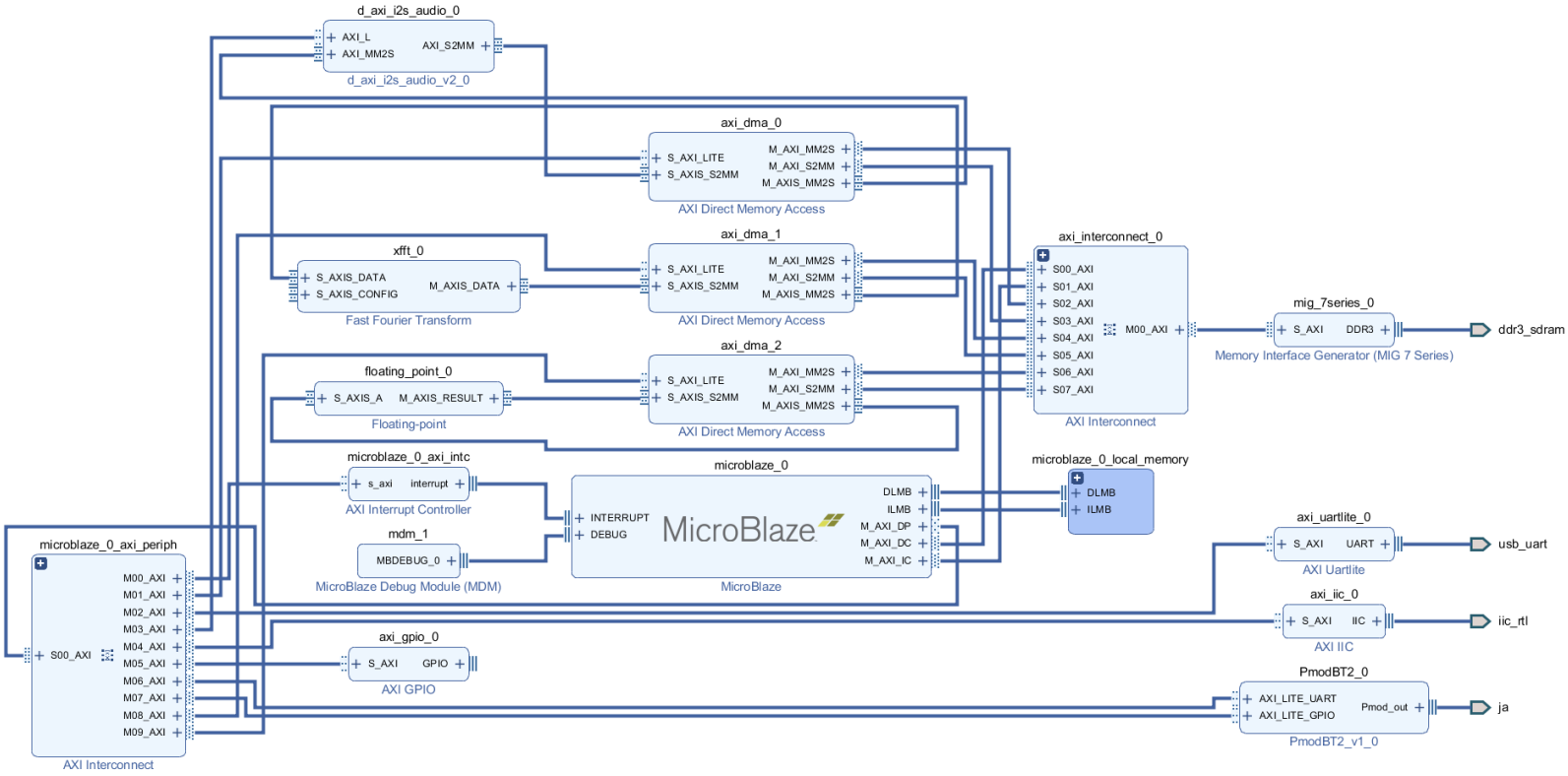


Figure 5: Vivado block design diagram (interface connections only) for the controller FPGA board

IP Name	Function	Source	Changes Made
axi_dma	AXI-Stream operation between IPs and DDR3	Xilinx IP Catalog	Re-configured with different input and output data width
d_axi_i2s_audio	Audio codec data transfer controller	Diligent Tutorial (https://github.com/Digilent/Nexys-Video-DMA)	No change
xfft	Fast Fourier Transform for the audio signal processing	Xilinx IP Catalog	Re-configured to 512-point, full precision fixed-point operation mode
floating_point	Hardware acceleration of floating-point arithmetics	Xilinx IP Catalog	Re-configured for logarithm operation at maximum performance setting
PmodBT2	Bluetooth PMOD controller	Digilent IP Library (https://github.com/Digilent/vivado-library/releases)	No change
axi_iic	I^2C controller for the audio codec	Xilinx IP Catalog	Re-configured to operate at 100MHz
axi_gpio	Read the state of the 4 physical buttons	Xilinx IP Catalog	No change
axi_uartlite	Enable console debug printing using the uart	Xilinx IP Catalog	No change
microblaze	Softcore processor for controlling the operations of all IPs	Xilinx IP Catalog	Enabled floating-point instruction set and high performance mode
microblaze _axi_periph	Connect all peripheral IPs to microblaze	Xilinx IP Catalog	No change
microblaze_axi_in	Interrupt controller for the buttons	Xilinx IP Catalog	No change
axi_interconnect	Connect all AXI-Stream DMAs and Microblaze to DDR3	Xilinx IP Catalog	No change
mig_7series	DDR3 memory for storing audio samples	Xilinx IP Catalog	Configured to 512MB

Table 5: Description of IPs on the controller FPGA board

4.1.1 Controller Board Setup and IP Configuration

Configuration settings, testbenches and APIs for all IPs on the controller board is documented in a 17-page step-by-step tutorial found at this link: Floating Point DMA, FFT and Bluetooth Tutorial

4.2 IPs in the Hub FPGA

IPs on the hub FPGA board are listed in Table 6. The server controller and relay switch IPs are custom while other IPs come from the IP catalogs provided by Xilinx and Digilent. IP connections are shown in Fig 6.

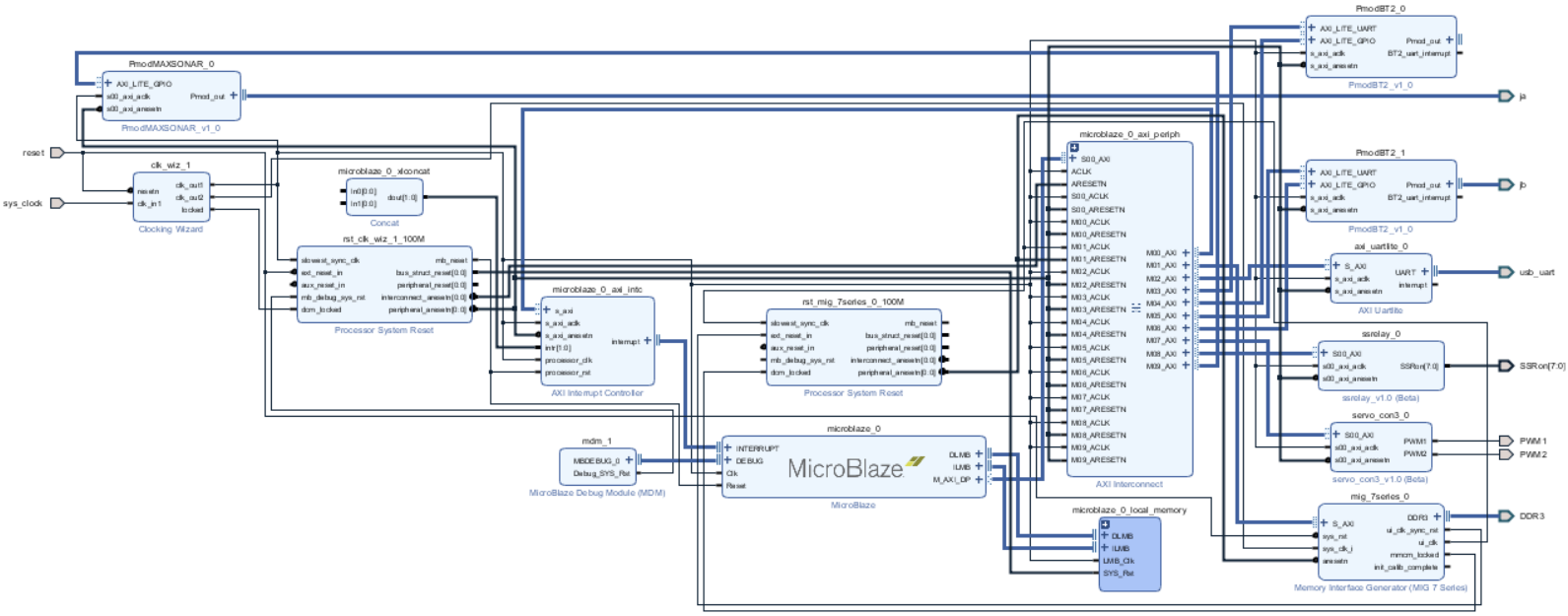


Figure 6: Vivado block design diagram (interface connections only) for the hub FPGA board

4.2.1 Hub Board Setup and IP Configuration

Configuration settings, testing script and APIs for all IPs on the hub board is documented in a tutorial found here: Hub Board PMOD Tutorial.

IP Name	Function	Source	Changes Made
PmodBT2	Bluetooth PMOD controller	Digilent IP Library (https://github.com/Digilent/vivado-library/releases)	No change
axi_uartlite	Enable console debug printing using the uart	Xilinx IP Catalog	No change
microblaze	Softcore processor for controlling the operations of all IPs	Xilinx IP Catalog	No change
microblaze _axi_periph	Connect all peripheral IPs to microblaze	Xilinx IP Catalog	No change
mig_7series	DDR3 memory for storing command data from the controller when necessary	Xilinx IP Catalog	No change
Pmod_MAXSONAR	Ultrasonic sensor PMOD controller	Xilinx IP Catalog	No change
ssrelay	AXI slave for controlling the relay switch	Custom IP	N/A
servo_con3	AXI slave for controlling the relay switch	Custom IP built from example Verilog project (http://www.instructables.com/id/Controlling-Servos-on-FPGA/)	Changed input/output and internal parameters to match the servo motors used and connection interfaces

Table 6: IP descriptions on the hub FPGA board

4.3 Software System for MFCC Audio Recognition

System overview diagram shown in Fig 4 summarizes all critical components of our audio recognition system. More technical details of the system is presented below. This system was created with help from [5]. Source code and a tutorial on using this project can be found at our GitHub repository.

Initial Processing To reduce dependency on volume (loudness) of speech, we run the collected audio through all other processing steps. It is also worth nothing that our DMA sometimes have error handling the first few data points, so it is important to filter those out, especially for the VAD step.

Voice Activity Detection We use VAD to segment audio into individual words. In order to determine whether a 32ms window contains speech, we compare the RMS power of this window against a threshold. Since the DAC on-board *Nexys Video* has significant noise, we have to set the window to a much higher level than the one used on PC. This setting leads to difficulties identifying consonants, or quiet syllables, a problem not experienced on cleaner samples used for PC.

FFT At the beginning, we decided to use a 512 sample (equivalent to 32ms) window for future replacement by a hardware FFT engine. Since our audio is essentially a discrete real input, we can use a simpler FFT formula (Eqn 1). Prior to integration with hardware FFT module, we computed it by doing a matrix multiplication. Coefficients of this matrix are hard-coded because *Microblaze* has poor performance for scientific calculations.

$$X_k = \sum_{n=0}^{N-1} x_n(cos(2\pi kn/N) + isin(2\pi kn/N)) \tag{1}$$

Mel Power We have to apply the mel-scale mapping to each one of the FFT power terms. Equation for mel-scale is shown in Eqn 2.

$$m = 2595log_{10}(1 + \frac{f}{700}) \tag{2}$$

Discrete Cosine Transform DCT can also be performed in a matrix-multiplication manner. Formula for DCT (DCT-II) is presented in Eqn 3. Here, we are treating each one of the power coefficients from FFT as another signal.

$$X_k = \sum_{n=0}^{N-1} x_n \dot{c}os(\frac{\pi}{N}(n+\frac{1}{2})k) \tag{3}$$

Dynamic Time Wrap DTW is a popular algorithm used to compare sequences of different length. In addition to traditional DTW, we introduced penalty terms of length difference, making insertion and deletion having a cost multiplier of $abs(row_{\#} - col_{\#})$. This penalty term is suitable because we only have a few terms in our library, most of which have different lengths. To be more specific, we define the cost to be

```
i = row_number
j = col_number
cost = cart_dist(a1[i - 1], a2[j - 1]) // Cartesian distance between two points
len_diff = max(abs(i - j), 1)
d1 = cost * len_diff + dtw_array[i - 1][j]
d2 = cost * len_diff + dtw_array[i][j - 1]
d3 = cost * max(len_diff / 2, 1) + dtw_array[i-1][j-1]

actual_cost = min(d1, d2, d3)
```

Audio Library As described in previous sections, our system is capable of recognizing open/close, on/off, light, door as well as Mandarin commands kai(open)/guan(close) and deng(light). There is a total of 55 audio samples, approximately 6 samples per term, in our library. We have computed distances between each pair of audio samples. In this test, we found that samples that have the lowest distance to the one in question are always other samples of the same word, thus verifying the correctness of our algorithm. At least 20% difference is observed for the next incorrect word, leaving us enough tolerance for real-world testing.

Python Prototype The entire algorithm had been implemented in Python first. We have added additional plotting tools for debugging and compared our results against values found by popular 3rd party providers. Fig 7 is an example showing that our MFCC computation produces result very similar to that found by *librosa*. In addition, we have created tools to automatically generate coefficients and audio data library in `.c` and `.h` format for painless migration to C.

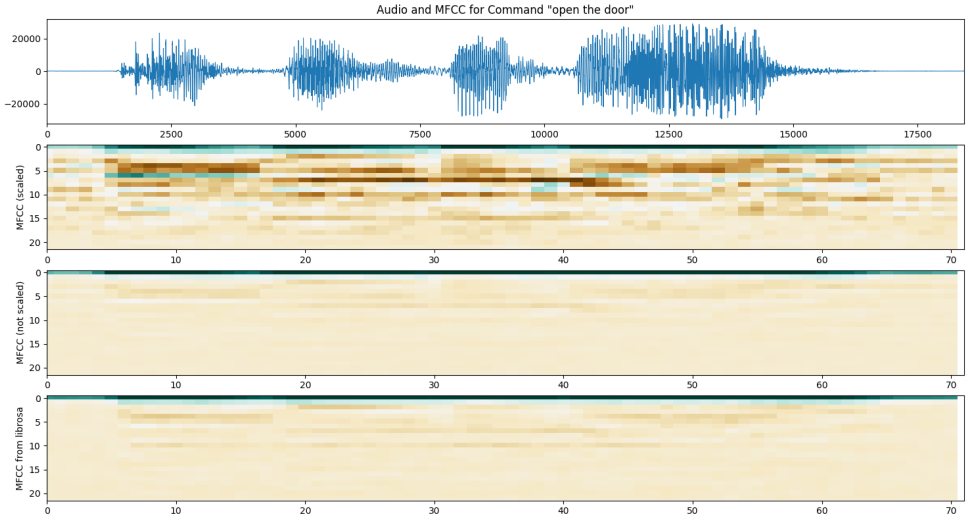


Figure 7: Scaled, unscaled and *librosa* results

5 Design Tree

A detailed list of design components is presented below.

- **Controller Board** (*Nexys Video Board*)
  - **Hardware IPs** Design components for audio acquisition, communication and algorithm accelerator
    - **axi_dma**: Enables AXI-Stream operations among DDR3 and AXI-Stream enabled IP
    - **d_axi_i2s_audio**: Audio codec data transfer controller
    - **xfft**: Fast Fourier Transform for audio signal processing
    - **floating_point**: Hardware acceleration for floating-point logarithm operation
    - **PmodBT2**: Bluetooth PMOD IP for data transmission
  - **Software** *Microblaze* routines for the audio recognition algorithm. All files have `.h` header and `.c` implementations.
    - **coefficients** Matrix multiplication coefficients for prep/post-processing, FFT and DCT

        * **stored_sounds** Pre-computed MFCC library of commands
        * **dma_functions** Interface with hardware accelerators. Deals with DMA memory content handover
        * **audio_processor_entry** Preprocessing and wrapper function for audio recognition routine
        * **audio_divider** VAD and audio segmentation
        * **MFCC** Wrapper for MFCC computation routine, calls each component of the MFCC routine
        * **do_matching** Calls DTW to find the command to which it has the minimum distance
        * **microblaze_stub** A dummy implementation of *Microblaze*-specific functions for the C program to be compatible with Visual Studio

- **Hub Board** (*Nexys Video Board*)
    - **Hardware IPs** Design components for communication and controlling various PMODs (ip_repo)
        * **PmodBT2**: Bluetooth PMOD IP for data transmission
        * **PmodMAXSONAR**: Ultrasonic PMOD IP for distance detection
        * **servo_con3**: Servo controller PMOd IP for controlling two servo motors
        * **ssrelay**: Relay switch PMOD IP
    - **Software** *Microblaze* routines for the hub controller board. All files have `.h` header and `.c` implementations.
        * **hub_control**: Functions and routines for responding to the commands from the controller board

6 Tips and Tricks

6.1 Software Hints

Prototyping and Language Porting Prototyping any algorithm in Python or in Matlab is essential prior to migration to C for Microblaze. Although software designers have great flexibility with these high-level languages, they must consider what is actually achievable using C. For example, Python users frequently use dictionaries, but this is rather difficult to implement in an efficient manner in C without advanced 3rd-party libraries (often not available on *Microblaze*).

Performance Performance optimization on PC is usually unnecessary, especially if the program runs "fast-enough" with high level languages. However, *Microblaze* runs at 10% of PC frequency, only has 1 ALU (no double precision capability), has little cache, and is RISC-based. These factors together significantly lower the performance of *Microblaze*; slowdown in the range of 10x$\tilde{1}$00x is expected. Software designers must also keep in mind that *Microblaze* does not natively support double-precision calculations, nor have support for any of the *math.h* functionalities. Those operations are replaced by "microcode", which are often equivalent to hundreds of lines of assembly code.

6.2 Hardware Hints

Documentations for IP Configurations One of the most challenging tasks for IP usage is to find out how to configure IPs and interface them with Microblaze. There are several resources that help with the the configuration process. The first resource is the technical Xilinx IP documentation. However, since each documentation is around 20-50 pages, it might be challenging to find out the exact configuration setting that one desires for his or her project under time-constraint. Thus, one should look for more practical resources (often contains sample configuration settings) from the Digilent tutorials, Xilinx discussion forum, and past projects.

Drivers for Controlling IPs After generating the bitstream and re-export the hardware, the SDK project BSP files should automatically update with Xilinx driver files that controls each Xilinx IPs. One should read through the driver files and use the appropriate APIs there. One should not attempt to write IP driver APIs from scratch.

7 Video

A video of the control sequence *ENTER* has been recorded and posted on Youtube. This video can accessed from Jingbo's channel.

8 Conclusion

Our home management system is designed to make people's life easier. It relieves users from physically accessing various parts of the home, thus enabling people to have better control of the room. With carefully selected PMOD components and well-designed communication protocol between two FPGAs, our system will be secure, reliable and easy to use. The audio-activated design also fits well into modern homes, fur-

ther reducing physical interactions between human and computer systems. Together, our system forms a generic infrastructure that can be extended for appliances not incorporated into our current design. Systems that implement similar concepts will no doubt become a central part of modern home IoT control center.

References

[1] Freepik: Press release symbol of a man with a microphone
https://www.freepik.com/free-icon/press-release-symbol-of-a-man-with-a-microphone_716892.htm

[2] Flaticon: Open exit door free icon
https://www.flaticon.com/free-icon/open-exit-door_59801

[3] Iconfinder: Bluetooth, connect, sync, wave icon
https://www.iconfinder.com/icons/299354/bluetooth_connect_sync_wave_icon#size=256

[4] Digilent: Nexys Video Artix-7 FPGA: Trainer Board for Multimedia Applications
https://store.digilentinc.com/nexys-video-artix-7-fpga-trainer-board-for-multimedia-applications/

[5] Haythem Fayek: Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between
http://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html