FPGA Remote Laboratory Using IoT Approaches

by
Alexander Michael Magyari

A thesis submitted to the Department of Electrical and Computer Engineering,
Cullen College of Engineering
in partial fulfillment of the requirements for the degree of

Master of Science

in Computer and Systems Engineering

Chair of Committee: Dr. Yuhua Chen

Committee Member: Dr. Jinghong Chen

Committee Member: Dr. Yi-Lung Mo

University of Houston
December 2021

DEDICATION

In memory of my mother, whom I hope to have made proud with the presentation of this thesis and the completion of my degree.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Yuhua Chen. I attribute both my academic and current career successes to her continued guidance and encouragement throughout my undergraduate and graduate studies.

I would also like to thank my father, whose consistent interest in my work has led to many long talks about the technological frontier. Lastly, I would like to thank my fiancée, Nicole, for keeping me healthy and sane throughout the pandemic that inspired this work.

ABSTRACT

Field-Programmable Gate Arrays (FPGAs) are high-end devices that are not easily shared between multiple users. In this work, a remotely accessible FPGA framework using accessible Internet of Things (IoT) approaches was developed. This was created to provide a method for students to receive the same level of educational quality in a remote environment that they would receive in a typical, in-person course structure for a university-level digital design course. Keeping cost in mind, the functionality of an entry-level FPGA and a Raspberry Pi Zero was combined to provide IoT access for laboratory work. Previous works in this field allow only one user to access an FPGA at a time, which requires students to schedule time slots. This design is unique in that it gives multiple users the ability to simultaneously interact with one individual top-level design on an FPGA. This novel design has the benefit for classroom presentations, collaboration and debugging, and eliminates the need for restricting student access to a time slot for FPGA access. Further, the hardware wrapper is lightweight, utilizing less than 1% of tested FPGA chips, allowing it to be integrated with resource-heavy designs. The application is meant to scale with large user bases; there is no difference between how many users can interact with the remote design, regardless of the complexity of the design. Further, the number of users who can interact with a single project is limited only by the bandwidth restrictions imposed by Google Firebase, which is far beyond any practical number of users for simultaneous access.

TABLE OF CONTENTS

APPENDICES

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

With the presence of the Covid-19 Virus, declared a pandemic by the World Health Organization on March 11th, 2020, the professional and educational world saw a sudden shift to an online presence. Sixty percent of higher education institutes denoted that the current outbreak has increased their presence of online learning [1]. This shift called for the immediate development of tools that could better sustain online working environments than those that were already present. The removal of the classroom in a teaching environment also reduces the amount of interaction between teaching assistants, classroom peers, and professors. While this improves safety by reducing the transmission of Covid, this has a significant impact on the quality of learning [1]. Specific to courses that require expensive laboratory equipment, remote learning disenfranchises students from access to high end equipment, such as Field-Programmable Gate Arrays (FPGAs). These two key factors are especially prominent in the realm of digital design, as student interaction with teaching assistants on FPGA devices is pivotal when debugging hardware designs.

This work seeks to increase the viability of a remote classroom specific for higher education in a digital-design curriculum. Specifically, this proposed method seeks to replicate the availability for direct teaching–assistant interactions with students on FPGA development boards in a collaborative, remote environment. Further, this was accomplished using tools that would be affordable to an average student. By providing a tool that is both affordable and accessible to students, the proposed method will mitigate the negative effects of remote learning.

Development efforts were focused on integrating a variety of systems to provide a multi-platform, multi-user interactivity tool referred to as the Remote FPGA. The Remote FPGA platform implements the necessary hardware modules to allow for remote access for a single, top-level design at a time. The work develops a method in which a design can be accessed via the Internet of Things (IoT) approach. This was accomplished by integrating an FPGA with a Raspberry Pi Zero. The user utilizes a

tool, developed in Python, to embed the top module design into a set of hardware modules that allows the FPGA to communicate with the Raspberry Pi via a universal asynchronous receiver transmitter (UART). This tool has an easy to follow graphical user interface (GUI) that will output a remotely accessible FPGA project that can be readily uploaded to the FPGA/Raspberry Pi system.

Once the user has incorporated the remote FPGA hardware wrapper into their design, it can be accessed via a cross-platform application that has the potential to be deployed on both Android, iOS, and Windows based devices. The application displays the remote development board interface and provides both input and output functionality. This increases remote lab mobility and accessibility; users no longer need access to a desktop computer or the physical development board to interact with their remote laboratory. The physical inputs on the development board are disabled to prevent interference with the remote access application. The Remote FPGA System also adds the ability to add various peripheral devices that are not already on the development board, such as seven segment displays, a variety of light emitting diodes (LEDs), push buttons, and slide switches.

Further, multiple users can access the same FPGA via the application at the same time. User inputs and outputs are reflected on all applications that are connected to the FPGA in real-time. This allows for multiple users to collaborate simultaneously on the same project, and is useful in laboratory grading and debugging. The ability to collaborate and have a number of users working on the same FPGA is unique to this work; other developments in online FPGA platforms allow only one user to connect to any given FPGA at a time [2]–[7]. Only allowing one user to access an FPGA remotely at a time is an issue, and it leads to further difficulties: students must schedule a time slot to work on a remote FPGA which in turn limits the amount of time that students can have FPGA access, and students do not have the ability to collaborate on projects with their teammates or teaching assistants. This work remediates these issues by removing the limit on the number of users that can access a single FPGA.

This proposed method was successfully implemented in a digital design course at the University of Houston. The design methodology was tested in the Spring of 2021, where students could access a completed lab with means to clarify instructions for expected inputs and outputs. Future plans include having students purchase their own development kit so as to allow for the upload of their own labs. This work was launched on multiple development boards from Intel in conjunction with a Raspberry Pi Zero, bringing the total of this technology to approximately $120 per user.

The following sections of this paper are organized as follows. Chapter 2 describes the related research in the field. Chapter 3 discusses the implementation of the proposed method. Chapter 4 discusses the outcome of this work. Finally, Chapter 5 concludes the work and describes the value and future works of the proposed method.

CHAPTER 2: RELATED RESEARCH

This work involves a novel application for FPGAs onto the IoT, and seeks to unite these two concepts and further the availability of remote learning applications. A background on both the IoT and the various methods for integrating an FPGA onto the IoT are provided.

Internet of Things

The Internet of Things (IoT) comes with modern developments in smart homes, cars, and wearable devices as the availability for communication between various sensors and processors emerges with widely available internet access. The IoT is defined as a system that contains a variety of smart devices that can communicate with each other without the need for human interaction [8]. While IoT, at its core, operates independently of human interaction, it provides a communication for a variety of systems which may involve human input. These system pathways include Human-to-Machine (H2M), Human-to-Human(H2H), and Machine in Humans (MiH) [9]. These systems, along with examples, are shown in Figure 2.1.

Within each of the aforementioned systems, a variety of components can communicate between themselves. A component on the IoT has the ability to transmit data, receive data, or use a combination of both. A component can be or contain any one, or more, of the following [8]:

- **Internet Access:** Access to the internet, which is provided via a processing unit, is an essential feature of any IoT device. Internet access allows the device to communicate to other devices within an IoT system without the need for human interaction [8].

- **Platforms:** A platform is any wired or wireless communication method that allows for a device to communicate with an Internet Service Provider (ISP) to

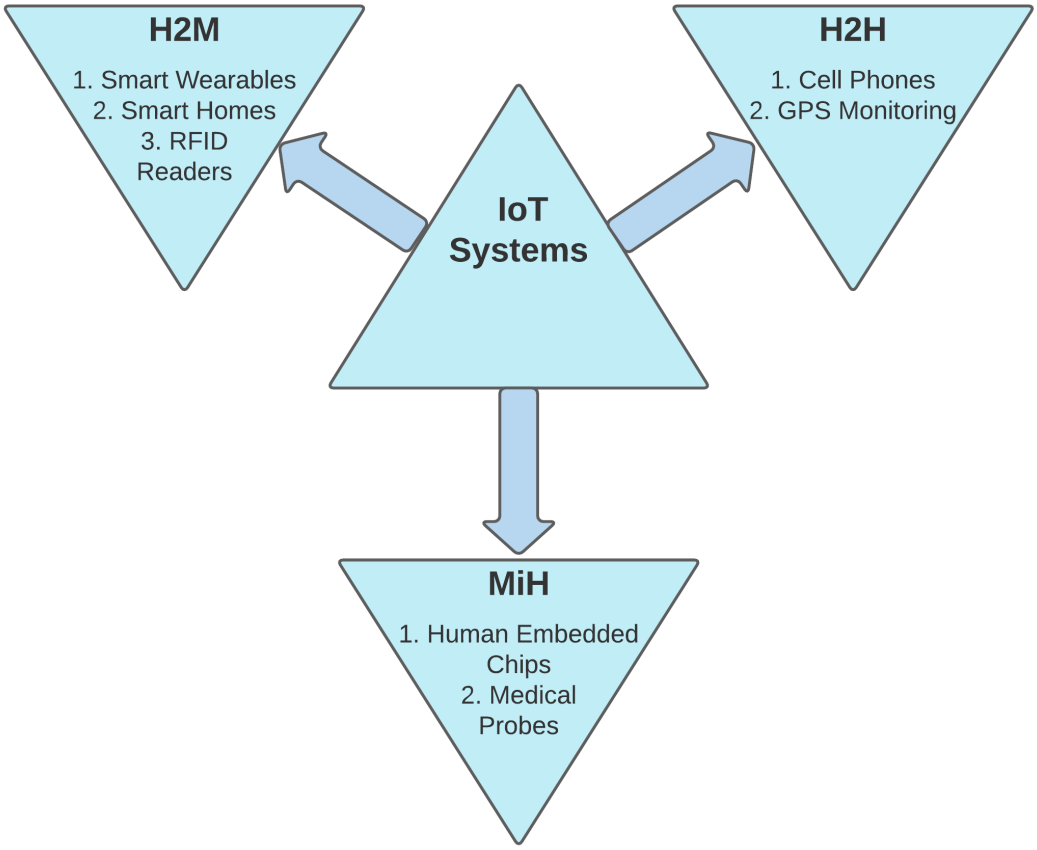


Figure 2.1: Systems within the Internet of Things.

provide internet functionality. Possible platforms include Wi-Fi, Ethernet, 5G, Bluetooth, and Zigbee [9].

- **Devices:** This includes smart devices which integrate sensing and processing capabilities. In the following examples, the term ”smart” refers to the device’s ability to collect data via peripheral sensors and process that data before making it available to the user or another IoT device. Examples include smart watches, smart phones, smart cars, and smart homes [9].

- **Sensors:** Sensors are often peripheral devices such as barometers, speedometers, proximity sensors, audio and video sensors, and thermometers. Sensors are utilized for data acquisition. This data can then be either stored, processed, or sent to another IoT Device [10].

- **Data Storage:** Depending on the particular use-case of the device, it may be optimal to store data that is acquired via either via internet access or on-board processing. Options for data storage include either local storage or uploading the data to the cloud [10].

FPGAs can play a vital role as any one of the components listed above. In many cases, FPGAs can provide IoT functionality that would not be possible with a classical microprocessor in an IoT system.

Field-Programmable Gate Arrays

FPGAs fall into the semiconductor category between microprocessors and Application-Specific Integrated Circuits (ASICs). FPGAs balance the programability of a microprocessor with the parallel processing power and speed of an ASIC [11]. FPGAs rely on a hardware description language (HDL), such as Verilog or Very High Speed Integrated Circuit Language (VHDL), to describe a set of complex logic functions [11], [12].

FPGAs integrate some powerful features from both a conventional ASIC and a microprocessor[11], [12]. Similar to an ASIC, the FPGA design is able to often rely on a clock operating in the mega- to gigahertz range. This means that basic digital instructions can be processed within anywhere from a few nanoseconds to a few hundred picoseconds [13]. Similar to a microprocessor, on the other hand, FPGAs can be reprogrammed, whereas ASICs cannot. The ability to reprogram an FPGA allows engineers to expand and modify digital designs without being required to reconstruct a digital circuit manually [11]. FPGAs constitute a wide range of modern applications, from digital signal processing [13], [14] to machine learning [14], [15], and from image processing [15] to data mining [16], and other functions not mentioned here.

FPGA Network Bridging Methods

An FPGA die itself does not have the physical means for network connections. However, designs for interpreting networking information can be implemented. From

connecting an FPGA to various network interfaces such as wireless antennas or Ethernet ports, to communicating with System on Chips (SoCs) that have the means for network capabilities, to instantiating soft core processors to handle the heavy lifting of network communication, there exists a multitude of opportunities for integrating an FPGA design with the IoT. These possibilities are described in the following subsections.

Network Interfacing via Soft Core Processors

FPGAs are often resource constrained, whether it be a limitation of power, logic units, or timing. To combat this, soft core processors are often used in lieu of complex, resource-heavy Register Transfer Level (RTL) designs. Soft core processors are microcontrollers, but rather than being physical devices implemented in silicon, they are instantiated in HDL. As processors, they can process compiled C code similar to a traditional processor, and as an HDL instantiation, they can be reconfigured to fit the designer's needs [17]. Engineers can configure the soft core processor to work with any hardware module such as a SPI or AXI bus, and they can customize features such as memory size or the instruction set architecture of the device [18]. This flexibility opens up a plethora of opportunities for integrating an FPGA with the IoT network.

While the soft core is not a means for connecting directly to a network, it does provide the means to efficiently interpret the data from a network, such as an Ethernet or WiFi connection. With a soft core working as a piece of middle-ware between an FPGA and an active network, complex functionality such as integration, repeated division, and register mapping can be offloaded from HDL. This has the potential to save engineers both time and resources [19].

Further, many soft core IPs are freely available for use. Examples of readily available soft core processors include the Z-scale from Berkley [20] and the ORCA from VectorBlox, both of which utilize RISC-V processing [19]. These open-source soft-core processors can serve as free alternatives to their hardware and private IP counterparts, opening the door for research where funding is an issue.

Soft Core Network Implementations

With the wide configurability of the soft core processor comes a wide range of potential applications for integration with the IoT on an FPGA. Huang et al. developed an IoT sensor hub that collected data from various sensors before uploading the data to the network [21]. Following the acquisition of data on a sensor, a hardware module within the FPGA processes the data and stores it on the instantiated soft core processor. The processor manages all of the incoming data before uploading it to the network via an Ethernet interface [21].

Further on regarding the topic of IoT sensor integration, Myint et al. developed a water quality monitoring system by combining the functionality of an FPGA and an instantiated NIOS-II soft core processor [22]. Myint utilized a UART serial connection to interface the instantiated processor with a Zigbee hardware communication module. This design method allows for wireless monitoring of water quality by measuring features such as water temperature and water level [22]. By relying on the functionality of the soft core processor, Myint did not have to develop the complex HDL required to communicate with the Zigbee module, and could instead rely on a simpler implementation in C.

System on Chip Network Bridge

With the increase in network access methods such as WiFi, Bluetooth, and 5G networks, comes the ever-increasing difficulty of utilizing this wide array for network communication tools in IoT. To remedy this, there exists a wide variety of SoCs that can provide a simple method for integrating a device, such as an FPGA, with the internet, and thereby forming a bridge to the IoT network. An SoC includes the various hardware components necessary for computational processing and data storage within a single chip [23].

Typically, ready-made SoCs are designed to support operating systems such as Linux or Microsoft Windows. Further, in the following examples of SoCs, methods for Internet access are integrated into the chip design as well. Examples of modern SoCs

that utilize components for IoT access such as Ethernet and WiFi include the Raspberry Pi, Arduino, Beaglebone, Intel Galileo, and the Adafruit Feather [24]. These systems are shown in Figure 2.2.

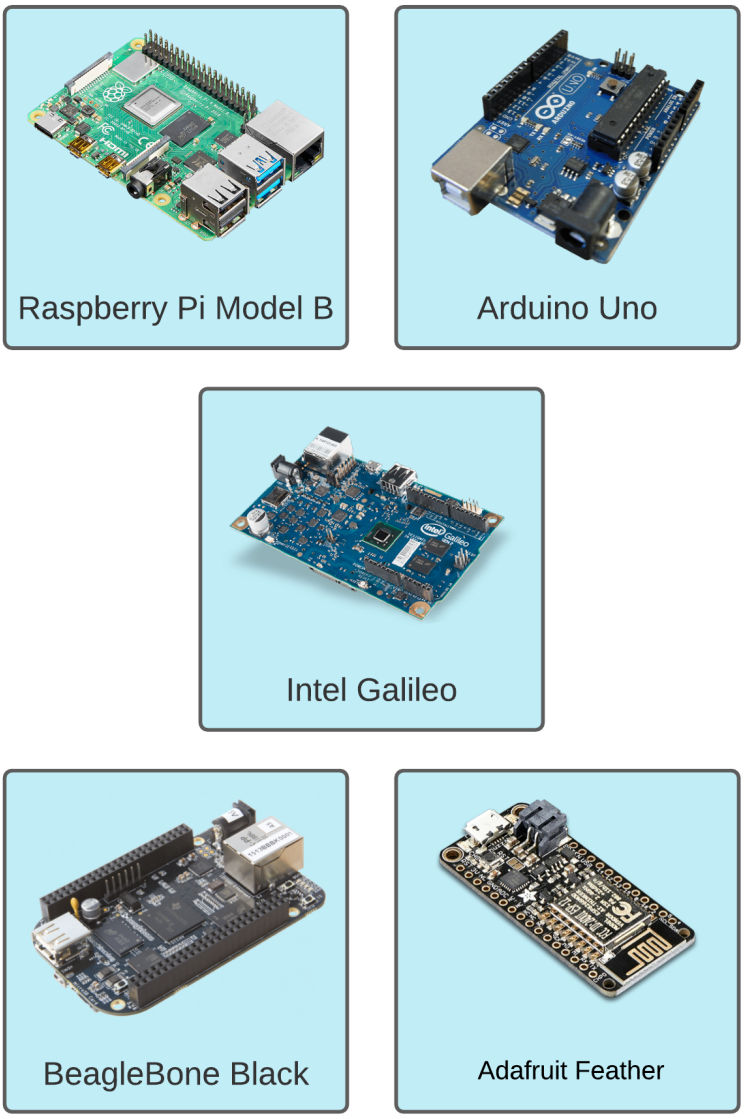


Figure 2.2: Full system on chip products.

The aforementioned SoCs, outside of providing easy access to the IoT for an FPGA, offer multiple methods for interfacing with the FPGA, such as USB, GPIO, and Ethernet. The following subsections discuss some of these examples in more depth.

Raspberry Pi Assisted FPGA Network

While the Raspberry Pi offers much more functionality than a network connection, the Linux system coupled with various network interfaces makes for a simple opportunity to integrate network access with an FPGA. With models spanning various sizes and functionalities, the Raspberry Pi has found its way into multiple networking designs for FPGAs, such as image processing for the IoT [25], smart buildings [26], and health monitoring systems [27].

Most Raspberry Pi models have various serial communication protocols such as I^2C, UART, and SPI, which allow them to quickly integrate with FPGAs using packaged Intellectual Property (IP). This benefit can make for rapid prototyping and deliverables. The networking capabilities of the various Raspberry Pi models are summarized in Figure 2.3. In this figure, models rated for 1000BaseT Wired Ethernet surpass the rating for 10/100 Wired Ethernet, however, the 10/100 Wired Ethernet bandwidth is still technically supported. This is indicated by a shaded teal box in Figure 2.3.

	Ethernet		*Local Area Network*		*Other*
	10/100 Wired	1000BaseT Wired	802.11ac	802.11n	Bluetooth
RPi 1 B	✓	×	×	×	×
RPi 1 B+	✓	×	×	×	×
RPI 2	✓	×	×	×	×
RPi 3 B	✓	×	×	✓	✓
RPi 3 B+		✓	✓	✓	✓
RPi 4B		✓	✓	✓	✓
RPi Zero W	×	×	×	✓	✓

Figure 2.3: Comparison of Raspberry Pi Models [28].

BeagleBone Assisted FPGA Network

The BeagleBone line of SOCs operates similarly to a Raspberry Pi. They have a variety of microprocessors, depending on the particular board, and have the capacity to

run a variety of Linux operating systems. Further, the BeagleBone products implement various network functionalities, ranging from gigabit Ethernet speeds to Bluetooth to WiFi functionality. The various network capabilites of the BeagleBone SoCs are summarized in Figure 2.4. Similar to Figure 2.3, the shaded teal boxes in the 10/100 Wired Ethernet column indicated that the 10/100 Wired bandwidth is supported, but the official rating of the BeagleBone model exceeds 10/100 Wired Ethernet.

	Ethernet		*Local Area Network*		*Other*
	10/100 Wired	1000BaseT Wired	802.11ac	802.11n	Bluetooth
BB Blue	×	×	×	✓	×
BB Black	✓	×	×	×	×
BB Black Wireless	×	×	✓	✓	✓
BB Black Enhanced		✓	×	×	×
BB AI		✓	✓	✓	✓
BB Green	✓	×	×	×	×
BB X15		✓	×	×	×
BB xM	✓	×	×	×	×

Figure 2.4: Comparison of BeagleBone Models [29].

Various FPGA networking projects have implemented the BeagleBone, as it is a versatile chip with quick set up times for network functionality as its processing power can be coupled with its onboard wired and/or wireless connections. Examples of works relying on a BeagleBone variant for networking include remote laboratory interfaces [30] which rely on the BeagleBone's network capabilities for interfacing an FPGA with the IoT, and remote DC motor control [31] which utilizes the onboard Ethernet controller of a BeagleBone to allow IoT access for an HDL design.

Modern Works Integrating FPGAs onto the IoT

While integrating FPGAs onto the IoT is far from a trivial task, FPGAs have a variety of uses and implementations as an IoT device [32]–[37]. IoT ready FPGAs are currently being utilized for online methods, including digital image processing, smart grid energy management, managing complex timing systems, broadcasting video recording solutions, data encryption, medical diagnoses and monitoring, and storage system solutions [32]. These topics open discussion not only for IoT FPGA applications, but also for the various methods of interfacing an FPGA with the Internet.

A popular method for providing an IoT interface with an FPGA is the Raspberry Pi [32], [38]. A wireless monitoring system, proposed by Gophane et al., utilizes a combination of sensors alongside a Spartan 6 FPGA working in series with a Raspberry Pi 3 [38]. Gophane et al. further utilizes the functionality of the IoT by integrating their FPGA system with an Internet storage cloud for allowing access to the data acquired by the FPGA. While their work did not utilize the onboard WiFi of the Raspberry Pi 3, they were able to implement a ZigBee module to provide the IoT connectivity for the Raspberry Pi [38].

One further FPGA IoT integration method utilizes SoC FPGAs [39]. SoC FPGAs are available from multiple manufacturers such as Xilinx and Intel. These models of FPGAs include on-board processors, such as ARM Cortex-M3s or ARM Cortex-A9s [39]. Further, FPGA devices that do not have an on-board central processing unit (CPU) can utilize soft core processors, which essentially convert the FPGA into a SoC model [39]. By utilizing an SoC FPGA, Basilino et al. successfully implemented an IoT message processing system in which they increased processing performance by 308% [39].

Once an FPGA has been integrated with the IoT, there are a seemingly endless amount of applications. With the combined power of parallel processing provided by the FPGA along with Internet access, researchers are able to tackle an incredible amount of issues. For example, Kang et al. [33] utilize an IoT-capable FPGA for transmitting and mining data from an external server. The FPGA gives the ability to process the data

extracted from the server with an improved collection rate of over 200% and an increase in energy efficiency of 15% [33].

One further application of an FPGA in the realm of IoT is edge computing, which was researched by Ferdian et al. [34]. By relying on an FPGA with IoT access rather than a conventional microprocessor, Ferdian et al. utilized an FPGA as an IoT node to process and encrypt data before transmitting it. The edge computing provided by the FPGA is able to reduce the needed data bandwidth by 66% by encrypting and compressing the data as compared to raw, unprocessed data [34]. Further, as the FPGA can process data in parallel, Ferdian demonstrated that the processing speed of the FPGA remained constant even when presented with an increasing amount of sensors [34].

An interesting application of an IoT FPGA is presented by Sung et al., in which a multitude of sensor data is processed by an FPGA and then hosted on a web page with means for a home care system [35]. The FPGA sends data to a server via a wired RS-485 connection where users can view the various sensor data. Further, by harnessing the power of IoT, the FPGA can trigger an alert email to the user in the case that a sensor is detecting foreboding data, such as a temperature above 28 °C [35].

One common FPGA application is facial recognition; Peng et al. takes one step further and integrates this service with IoT [36]. By building a deep neural network interface on a Zynq FPGA combined with Internet connectivity provided over Ethernet, Peng is able to accurately detect front faces [36]. The data, while processed in the deep neural network on the FPGA, is transmitted over Ethernet where it is checked against a database. The stored database data is then compared to the transmitted data from the FPGA, where information can then be loaded about the registered face [36]. This method allows a remote camera to detect and recognize an individual via an external FPGA node.

One additional application of IoT-capable FPGA devices is a vehicle monitoring system, developed by Wang et al. [37]. A system is realized in a Xilinx Zynq-7000 with dual-core Arm Cortex A9s that is integrated with a 4G module, providing

wireless Internet access from within a moving vehicle [37]. The FPGA, with the power of parallel processing, is able to monitor up to six digital cameras and global positioning system (GPS) tracking while transferring data to the client [37].

FPGAs in Remote Classrooms

The topic of remote access FPGAs for student laboratory access is documented in various discussions [2]–[7], primarily with a focus of either viewing an FPGA via a webcam [2], [3], [7], or emulating the development board via a web application [4], [5]. The increase in online lab settings may be attributed to the slow movement of educational courses from in-person to online prior the quick shift from the pandemic. Current works rely on a point to point transmission, that is, only one user can access a remote FPGA lab at a time [2]–[7]. This prevents users from collaborating on team projects and makes it difficult for students to work with teaching assistants.

For example, Hashemian et al. utilizes LabView and the Xilinx Spartan3 platform to create a remote FPGA server in which students can directly interact with an FPGA to view lab assignments [2]. Hashemian et al. rely on a Windows XP remote access terminal to provide direct access to the FPGA as if the user were directly interacting with it. The PC accessed via the remote terminal shows a webcam focusing on the FPGA alongside the LabView GUI that provides functionality for interacting with the FPGA inputs [2].

Similarly, Morgan et al. realize a remote FPGA application with a webcam and GUI pair on a Xilinx Nexys 2 [3]. Morgan also developed a method in which users are given the ability to view finite state machines on the Nexys, and interaction with the FPGA server is done via a web application as opposed to a remote desktop link. Further, students have the ability to upload their own modules to the FPGA server.

In another approach [4], Mohsen et al. developed a system in which multiple concurrent users can utilize a system of FPGA devices simultaneously. A user logs into a web server and is assigned a Universal Serial Bus (USB) port corresponding to an unused FPGA device. Once logged in, the user has the capability to upload their

design module and interact with the FPGA via a serial console. Outputs from the FPGA are viewable via a webcam interface. With this method, the amount of users is directly limited by the amount of available FPGA devices.

Another idea exists in which an FPGA lab is built around the remote access capability of the device, rather than a series of labs. Schwandt et al. prove the ability of an FPGA in a single remote lab setting [5]. This remote FPGA implementation allows students to upload an image which is then processed via a pre-loaded FPGA lab, and the output is shown directly to the user. The primary function of this lab is not to provide remote functionality, but to prove the ability of an FPGA in a remote setting.

One of the developments in the realm of remote FPGAs utilizes a Nexys 3 for remote access. Peinado et al. developed an admin panel in which user access to remote FPGA servers can be tracked and limited by various amounts of time [6]. This is to ensure that remote FPGA servers are available for use and that a few students cannot consume all the available resources for an indefinite amount of time. Peinado et al. do not use a webcam to show the board, but instead, they utilize a GUI and a serial monitor for input and output from the FPGA. Further, they utilize an Arduino to simulate various peripheral devices on the Nexys [6].

One final implementation of an online digital design laboratory, the Cyber Lab, combines multiple FPGA devices implemented as a data server and as an experimental platform for students. The Cyber Lab expands functionality beyond one FPGA, giving students the ability to process large sets of data in parallel on multiple devices. The Cyber Lab again utilizes the popular set up of a webcam for showing the FPGA interface. Further, the Cyber Lab implements a scheduler via a web server to ensure that all students who wish to access the available FPGAs have the ability to do so [7].

Significance of Proposed Work

This work defines a new technique for integrating IoT access to a variety of development boards. A method is provided for simple IoT peripheral access for any FPGA design. This is completed through the development of a tool for embedding IoT access into any Verilog hardware design. Further, this work allows multiple users to collaborate on one FPGA device, unlike any previous works. By loading the user application GUI, all users can directly interact with the design while seeing the interactions from other users in real time. Further, the application reports low latency results, which are further summarized in Chapter 4. This collection of benefits allows users to interact with the FPGA development board remotely while providing an experience similar to interacting with the FPGA development board in person.

CHAPTER 3: MATERIALS AND METHODS

Remote FPGA Framework Implementation

The remote access FPGA package is produced with all of the utilities required to quickly begin hosting FPGA interactivity on the IoT. The user must begin with a hardware design that is compatible with one of the supported development boards, including an existing pinout. Next, the user would utilize the included Python Wizard, which implements the necessary hardware modules needed for remote access. The Python Wizard will output a copy of the user project that is now remote access ready. This new project is then synthesized, implemented, and flashed to the FPGA via the Intel Quartus Application. For the final step, the user would copy a configuration file entitled "wpa_supplicant.conf" to the SD card with the necessary information to connect to the local WiFi network. The contents for the configuration file must be exactly as shown in Listing 1, with *network_name* being replaced with the name of the network, and *network_password* being replaced with the password of the network [40].

Listing 1: Rasperry Pi network configuration file.

```
network={
    ssid="network_name";
    psk="network_password";
}
```

The design will then be ready for remote access from the included cross-platform application. The design flow for implementing the remote access FPGA system can be seen in Figure 3.1.

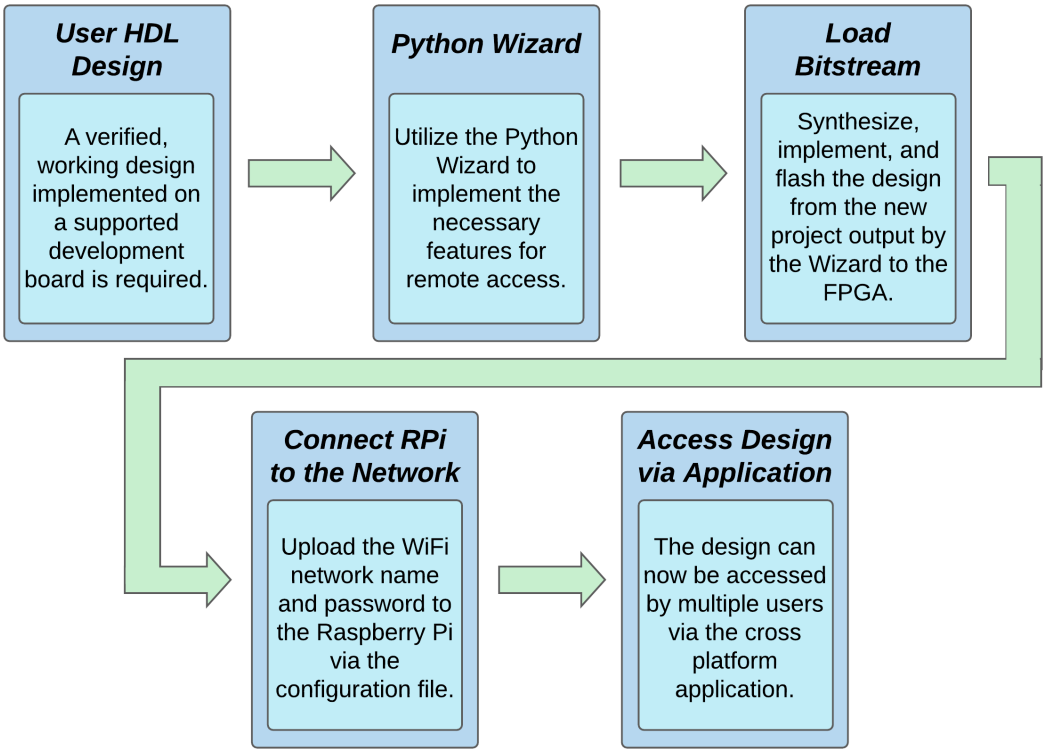


Figure 3.1: Remote access implementation flow.

System Architecture Design

The remote access FPGA system requires multiple hardware and software modules to function, including developments in Python, Typescript, Google Firebase [41], and Verilog HDL. With means for communicating user input and output values with the Raspberry Pi, the student module must work in conjunction with an interactivity hardware module that is integrated into the user design via the Python Wizard. The interactivity module arranges for outputs from the student module to be packed into a serial bitstream to be sent to the Raspberry Pi, and simultaneously interprets incoming bitstreams as inputs for the student module. An interactivity module template, also referred to as the hardware wrapper, is designed, and the wrapper is modified by the Python Wizard so as to fit the design of the user's module and copied into a new Quartus project along with a copy of the user project. The wizard further implements the necessary wires to drive the user defined signals from the interactivity module according to the serial inputs received from the Raspberry Pi. Similarly, the wires are

also implemented from the user module to the interactivity module to drive the serial outputs to the FPGA. The user module remains unmodified by the Python Wizard; it is only moved from being a top-level module to an embedded module in the wrapper, with the inputs and outputs now being driven by the interactivity module. The entire system architecture is shown in Figure 3.2.

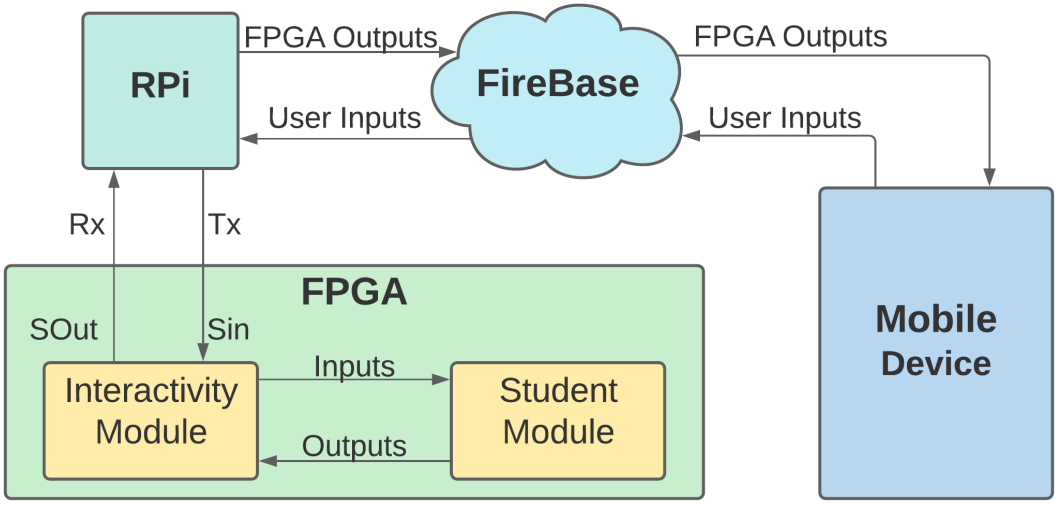


Figure 3.2: Remote access system architecture.

The Raspberry Pi grants the FPGA the ability to communicate with Google Firebase, which is a cloud-based system that manages the two-way communication between the Raspberry Pi and the mobile device. The mobile device—the remote user endpoint—allows for the interactivity between remote users and the physical FPGA. Google Firebase is the only portion of this work to have a one-to-many relationship with the user base. Each user application is communicating directly with Google Firebase, and any changes within the database are interpreted directly by the Raspberry Pi. There is only one data stream between a Raspberry Pi and Google Firebase at any time, regardless of the number of active users. This allows for the framework to scale directly with the user base, as the number of users who can access an FPGA remotely is limited only by the bandwidth restrictions from Google Firebase. Similarly, the amount of Remote FPGA projects that can be hosted is limited only by data constraints imposed by Google Firebase [41]. Each individual module is further described in later sections.

Mobile Application

The mobile application was developed via the Ionic development platform. Ionic increases application functionality by providing a cross-platform development environment that utilizes web-based programming languages, such as TypeScript. Once an application is developed through Ionic, the Ionic command line interface allows the designer to export the application to iOS and Android platforms. Further, Ionic formats all visuals, such as buttons, fonts, and text alignment to be consistent with each respective platform.

The mobile application implements various objects to display the binary FPGA pin values in realtime. The objects and their associated properties are modeled after the physical peripheral devices and are shown in Table 3.1. Each object is assigned a ”name,” which is used for identification within the application. Further, each object is also given a negative logic variable, which determines if it is using a pull up or pull down resistor network. Each peripheral is assigned a pin object which is an active listener to the Firebase Realtime Database, and is responsible for updating its parent peripherals depending on the values received from the database.

Table 3.1: Application class implementations.

Class	Properties
SevenSegment	Name: String NegativeLogic: Bool SegmentImage: String Pins: Array<Pin>
LED	Name: String NegativeLogic: Bool LEDImage: String PinConnect: Pin
Slide	Name: String NegativeLogic: Bool PinConnect: Pin Flipped: Bool
PushButton	Name: String NegativeLogic: Bool PinConnect: Pin Depressed: Bool
Pin	Assignment: String Direction: Int Value: Int Parent: Array<Peripheral>

Further, the mobile application actively monitors the Google Firebase Realtime Database, specifically listening for any changes to an FPGA table. Upon the reception of a pin modification, the application updates the corresponding peripheral in the GUI.

Google Firebase Real-Time Database

Google Firebase is a real-time database, meaning that all changes are automatically updated to any program that has an active listener attached to that database. Google Firebase is an ideal candidate for the remote access FPGA application, as all changes from both the FPGA and the mobile application must be immediately reflected on the opposite end of the data line. Further, Google Firebase relies on NoSQL, storing JavaScript Object Notation (JSON) objects rather than the typical Structured Query Language (SQL) table hierarchy. The default JSON object that is passed between the Raspberry Pi and mobile application through Google Firebase is shown in Figure 3.3.

Each individual FPGA is identified within the database by the unique hardware ID located on the Raspberry Pi. This provides a uniquely identafiable key that the mobile application can use to request updates on pin values. Within each individual FPGA table is the users self-denoted ID and the peripheral values of the FPGA. The included pins in this table are all pins from the FPGA's General Purpose Input/Outputs (GPIOs), LEDs, seven segment displays, push buttons, and slide switches. All outputs are mutable from the Raspberry Pi, and all inputs are mutable from the user application.

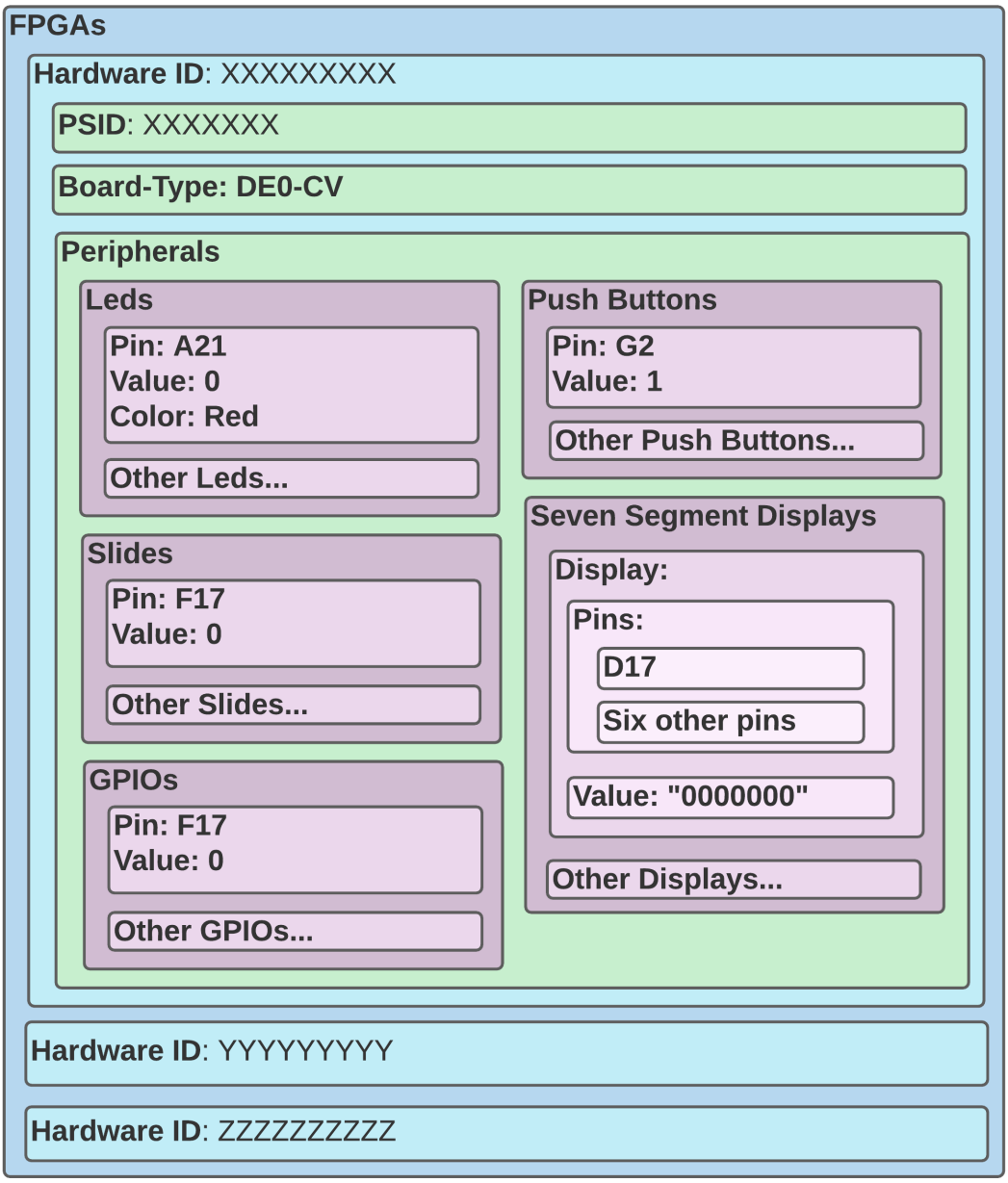


Figure 3.3: Database architecture.

Raspberry Pi to FPGA Interface

The Raspberry Pi model was selected with two key parameters in mind: affordability and on-board WiFi. Very few pins are required for connection with the FPGA, so GPIO availability is not an issue with any Raspberry Pi model. With these two requirements, the ideal candidate is the Raspberry Pi Zero-W, a small version of the original Pi with a manufacturer's suggested retail price (MSRP) of $10. The pin connections and signal names for the DE0-CV and DE2-115 development boards from Intel are described in Table 3.2. The DE0-CV and DE2-115 development boards utilize Cyclone V and Cyclone IV-E FPGAs from Intel, respectively. Further, the development boards have peripherals that allow the user to interact with the FPGA without the need for any additional circuit building, such as LEDs, slide switches, and push buttons. These peripherals are deactivated when the remote FPGA wrapper is in use, as the peripherals are accessed via the mobile application instead. Both of the aforementioned development boards were utilized for testing purposes with the remote FPGA wrapper.

The benefit of serializing the data from the FPGA is that there are only three necessary data connections from the Raspberry Pi, despite the management of all of the input and output signals from the FPGA. This is accomplished by encoding the pin values for the user-selected board and transmitting them over UART to the Raspberry Pi.

Table 3.2: Raspberry Pi Pin connections.

Raspberry Pi Pin	DE0-CV Pin	DE2-115 Pin	Signal Name	Signal Description
2	5V	5V	5[V] Power	Five-volt power supply.
6	Gnd	Gnd	Ground	Common ground.
7	J17	AH23	Reset	Software controlled FPGA reset.
8	G12	AH26	R-Pi UART TX	Serial transmit line.
10	K16	AG26	R-Pi UART RX	Serial receive line.
11	G15	AG23	UART Error	Active high upon transmission error.

With means to transmit pin changes for both inputs and outputs, data including the modified pin along with the new pin value must be sent back and forth between the Raspberry Pi and FPGA. Pin values for various development boards are extracted from a local JSON file loaded on the Raspberry Pi, and the correct file is loaded based on configuration data that is loaded to the FPGA via the compilation wizard. Once the pins have been extracted, they are sorted alphabetically and enumerated in binary values. Binary pin values for the DE0-CV board are shown in Table 3.3.

Remote FPGA Hardware Wrapper

The Python Wizard embeds the user design into the Remote FPGA Hardware Wrapper. The abstract FPGA system diagram is shown in Figure 3.4. The hardware wrapper is composed of four modules that drive the student module: a UART for communication with the Raspberry Pi, a First In First Out Buffer (FIFO) for storage of data as it as received by the wrapper, an input manager for driving signals to the student module, and an output manager for receiving output signals from the student module.

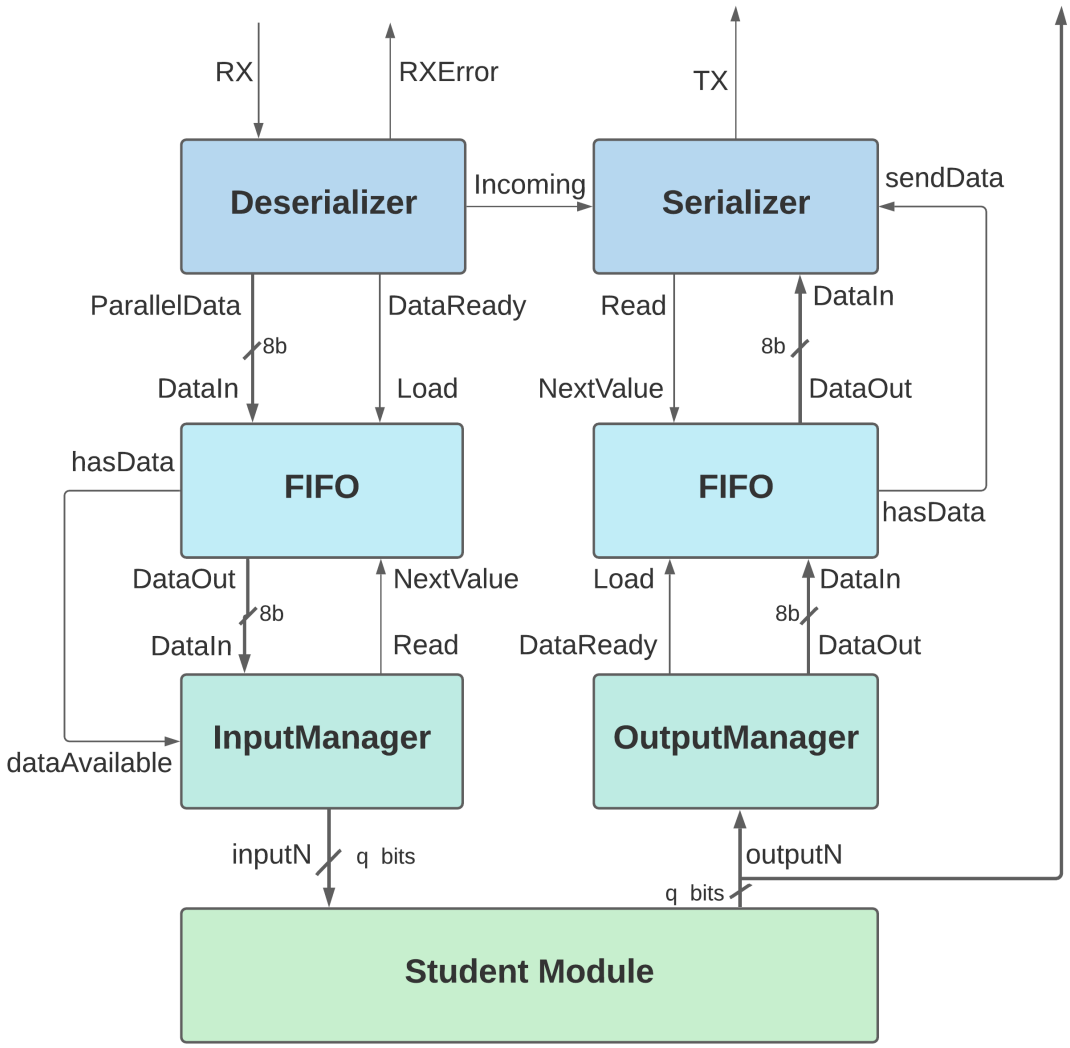


Figure 3.4: Hardware wrapper architecture.

Table 3.3: DE0-CV pin reference values.

Pin #	Pin	Pin #	Pin	Pin #	Pin	Pin #	Pin
0000000	A12	0100000	D13	1000000	M18	1100000	U1
0000001	A13	0100001	D17	1000001	M20	1100001	U13
0000010	A14	0100010	E14	1000010	M21	1100010	U15
0000011	A15	0100011	E15	1000011	M6	1100011	U16
0000100	AA1	0100100	E16	1000100	M7	1100100	U17
0000101	AA10	0100101	F12	1000101	M8	1100101	U2
0000110	AA13	0100110	F13	1000110	N1	1100110	U20
0000111	AA14	0100111	F14	1000111	N19	1100111	U21
0001000	AA15	0101000	F15	1001000	N2	1101000	U22
0001001	AA17	0101001	G11	1001001	N20	1101001	U7
0001010	AA18	0101010	G13	1001010	N21	1101010	V13
0001011	AA19	0101011	G16	1001011	N9	1101011	V14
0001100	AA2	0101100	G17	1001100	P14	1101100	V16
0001101	AA20	0101101	G18	1001101	P16	1101101	V18
0001110	AA22	0101110	H10	1001110	P17	1101110	V19
0001111	AB12	0101111	H14	1001111	P18	1101111	V20
0010000	AB13	0110000	H18	1010000	P19	1110000	V21
0010001	AB15	0110001	J11	1010001	P9	1110001	W16
0010010	AB17	0110010	J13	1010010	R15	1110010	W19
0010011	AB18	0110011	J18	1010011	R16	1110011	W2
0010100	AB20	0110100	J19	1010100	R17	1110100	W21
0010101	AB21	0110101	K17	1010101	R21	1110101	W22
0010110	AB22	0110110	K19	1010110	R22	1110110	W9
0010111	B12	0110111	K20	1010111	T12	1110111	Y14
0011000	B13	0111000	K21	1011000	T13	1111000	Y15
0011001	B15	0111001	K22	1011001	T14	1111001	Y16
0011010	B16	0111010	L1	1011010	T15	1111010	Y17
0011011	C1	0111011	L17	1011011	T17	1111011	Y19
0011100	C13	0111100	L18	1011100	T18	1111100	Y20
0011101	C15	0111101	L19	1011101	T19	1111101	Y21
0011110	C16	0111110	L2	1011110	T20	1111110	Y22
0011111	C2	0111111	L8	1011111	T22	1111111	Y3

FIFO Buffer

To account for the relatively slow read and write times of the UART module, a FIFO is put in place to hold data during transmit and receive procedures. The FIFO is a configurable buffer that will hold data until it can be used. Data is loaded into the buffer such as a queue: new data is loaded at the back of the queue, and when new data is requested from the buffer, the oldest data is retrieved. Both FIFOs are configured with an 8-bit width. The FIFO on the input side has a depth of 16 words and the FIFO on the output side has a depth of 64 words.

Table 3.4: Signals of the custom-logic FIFO module.

Name	Direction	Width	Description
Load	In	1b	When asserted, the FIFO loads the value DataIn into the buffer.
DataIn	In	FIFO_WIDTH	The data to store into the buffer.
hasData	Out	1b	Asserted when the FIFO contains data.
dataOut	Out	FIFO_WIDTH	The oldest data in the buffer.
nextValue	In	1b	When asserted, the FIFO sends the next data value stored in the buffer to dataOut.

UART Digital Serializer/Deserializer

The UART is comprised of two main components: a serializer and a deserializer. The serializer allows for parallel data to be transmitted to the Raspberry Pi. By receiving a parallel input vector of some n bits, the serializer will sequentially shift the bits out to the Raspberry Pi one at a time. The deserializer is the contrary to the serializer. The deserializer will receive a sequential series of bits from the Raspberry Pi and shift them into a parallel vector of n bits. These two components make up the ”receiver/transmitter” of the UART module. The length of the bit vectors, also known as the amount of data frames, must be known at the time of design for both devices on either end of the UART. Further, both devices must know the order of bits that are to be transmitted through the UART. Otherwise, even if a bit vector is serialized, it may be serialized backwards.

Further, the UART is configured as half-duplex, meaning that it will attempt to not transmit and receive at the same time. This is to allow for use with a wide variety of devices. While the UART will not transmit data when receiving data, it does have the ability to receive data while transmitting. While this breaks the property of a traditional half-duplex UART, this allows for the incorporation into system with a full duplex UART. Signal descriptions for the UART module are described in Table 3.5.

Table 3.5: UART Configuration Properties.

UART Property	Value
Configuration	Half- Duplex
Baud Rate	256,000
Data Frames	8
Parity	One Bit Even
Bit Order	Most Significant Bit First

The other half of the UART acronym, ”Universal Asynchronous,” refers to the communication method's lack of a shared clock. Both devices on either end of the UART transmission lines can operate on their own individual clock, however, they must know how many bits are to be transmitted per second, referred to as the baud rate. With the baud rate in mind, the deserializer can sample the bit line according to its own clock and determine the state of the incoming bit. The UART signal descriptions are listed in Table 3.6.

Generated Input Manager

The Input Manager module is one of the modules that must be compiled via the Python Wizard. Depending on the amount of user inputs detected from the top module to be embedded into the system for remote access, the Input Manager will have the same exact corresponding outputs. For example, if the embedded module has five input signals with a width of four bits, the Input Manager will have a generated five outputs with width of four bits. In this manner, as the Input Manager receives an updated user input from the FIFO, it will decode the pin value from the most significant seven bits, and then set the corresponding signal to the value in the least significant bit. It will continue this cycle as long as the FIFO contains data.

Table 3.6: Signals of the custom-logic UART module.

Name	Direction	Width	Description
RX	In	1b	Incoming bit via UART.
RXError	Out	1b	Asserted on a timing or parity error.
TX	Out	1b	Bit being transmitted via UART.
Incoming	Deserializer->Serializer	1b	Asserted if there is an incoming UART transmission.
sendData	In	1b	Asserted when the FIFO has data to send.
DataIn	In	8b	Data to transmit.
Read	Out	1b	Asserted when the serialized has finished sending data and is ready for new data to transmit.
DataReady	Out	1b	Asserted for one clock cycle when the deserializer has new data to load into the FIFO.
ParallelData	Out	8b	The data to be loaded into the FIFO.

Table 3.7: Signals of the generated Input Manager.

Name	Direction	Width	Description
dataIn	In	8b	Data meant to be decoded from the FIFO.
Read	In	1b	Asserted when data has been sent to the embedded module so that new data may be decoded.
dataAvailable	Out	1b	Asserted when the FIFO is not empty.
Input[N]	Out	q bits	Corresponding signals going to the embedded student module with a width of q bits. Generated via the Python Wizard.

Generated Output Manager

Similar to the Input Manager, the Output Manager handles all of the embedded FPGA module outputs. The Output Manager has a corresponding input wire for each output of the embedded student module. The Output Manager then encodes the signal based on the pin assignment of the updated output from the student module. The encoding of the signal is determined by the Python Wizard via a JSON file. For example, if the FPGA toggles an output on a wire that has an assigned pin value of AA15 to a logic high, from Table 3.3 it can determine that the encoded value will be 8'b00010001. This is done by taking the pin value from the table, 7'b0001000, and

concatenating it with a single bit value of 1, which represents the logical value that should be assigned to the pin. This data is then output to the FIFO, so that it can be transmitted to the Raspberry Pi.

Table 3.8: Signals of the generated Output Manager.

Name	Direction	Width	Description
dataOut	Out	8b	The packed and encoded pin and pin value.
dataReady	Out	1b	Asserted when new data has been encoded.
output[N]	In	1 bits	Embedded module outputs to be encoded with width of q bits. This signal is also routed to the physical peripherals on the FPGA development board.

Embedded Student Module

The embedded student module is a user defined module, which is embedded into the Remote FPGA framework when the Python Wizard is run. This is the module that users will view on the mobile application. Both inputs and outputs are determined by the user design. All inputs are wired to the Input Manager, with the exceptions of system clocks. The system clocks are wired through the top-level module through to the embedded module. Outputs are both wired to the Output Manager and exposed on the top-level module so that they may be viewed on the mobile application as well as the physical FPGA device.

Python Compiler Wizard

The embedded module signals must be adapted to fit the remote access hardware wrapper. One could do this manually, but it is a long and meticulous procedure that should be automated. For this reason, the Python Wizard was developed. The information required by the Python Wizard includes the project folder destination, the destination for the new, remote-enabled project, the user .qsf constraint file, the user source code location, the name of the top module, and the development board that should be displayed within the user application. The Python Wizard then creates the inputs and outputs for the interactivity module from a template file as needed relative

to the top-module from the user. Next, the tool embeds the original top module into the generated interactivity module. Instead of overwriting the user project, the Python Wizard creates a new folder and project with the remote-access implementation.

The Python Wizard utilizes the basic user interface features of the *tkinter* library from Python. There are some basic input checks built into the wizard , such as confirming the folder locations and the correct file extensions. Once the build has been completed, the user can open the new Quartus project, recompile, and flash the FPGA with their new remote access-enabled project. The tool is limited in that it only works with HDL top modules that are implemented via Verilog. The Python Wizard is robust enough to detect the signal width and direction regardless of where it was defined in the code. As long as the original design is robust enough to synthesize prior to the implementation of the hardware wrapper, the Python Wizard will succeed in integrating remote functionality.

CHAPTER 4: RESULTS

Experimental Results

By utilizing the Ionic platform, a GUI for interacting with the FPGA remotely via a variety of mobile and PC devices was successfully implemented. The devices include PCs, Macs, devices utilizing iOS, and devices utilizing Android OS. Further, the application interface allows for users to interact with one another via the same FPGA lab simultaneously, regardless of the user platform. For example, a user with iOS devices, a user with a PC, and a user with an Android device can all interact with the same lab simultaneously.

When the user first opens the remote access FPGA application, they are greeted by a server browser page that lists all of the available FPGA devices for connectivity. Any FPGA that is connected to Google Firebase via the Raspberry Pi will be displayed here by the ID number. Upon selecting a device, the user will be taken to the display screen for their selected FPGA laboratory.

The layout is loaded from Firebase, with the default values being stored on the Raspberry Pi in a JSON file. All input and output peripherals that are by default on the physical board are listed in the device view. The DE2-115 development board from Intel is shown in Figure 4.1, and the DE0-CV development board from Intel is shown in Figure 4.2.

As can be seen in both views, all development board peripherals are displayed. The seven segment displays are shown at the top, labeled as SS[x], where x is the ID of the display device. The LEDs are shown underneath, labeled by their default pin connection, followed by the slide switches, again labeled by their pin connects. The last set of on-board peripherals, the push buttons, are shown at the very bottom. All connected users can interact with the design. This is to encourage collaboration between students and teaching assistants during the debugging/presentation of a project. All user inputs are sequenced, and the input that is last received by the Raspberry Pi server will

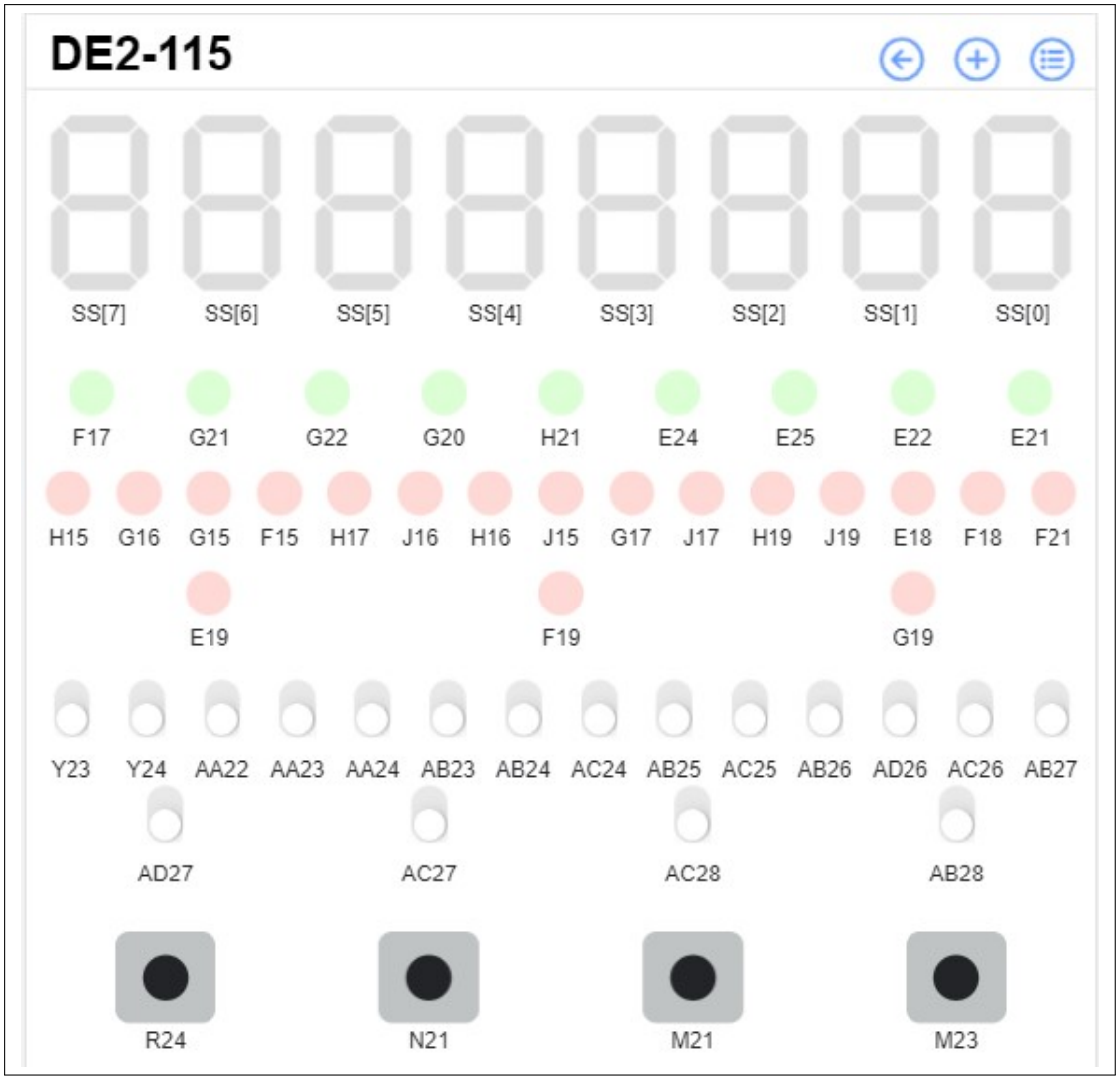


Figure 4.1: DE2-115 on an iPad.

be the one reflected on the FPGA as all inputs are sequentially loaded into a FIFO. Further, three buttons can be seen in the menu toolbar. They are, in order from left to right: the back button, the add peripheral button, and the list peripheral button.

The back button simply allows the user to exit the view for the current FPGA and return to the FPGA server selection screen. The add peripheral button will allow the user to add a new input or output to a GPIO pin located on the development board. This functionality can be seen in Figure 4.3a. For this new peripheral to work, the GPIO pin must be assigned to a signal in the original Quartus project before the remote connectivity wrapper is added to the project. All LEDs are grouped according to their

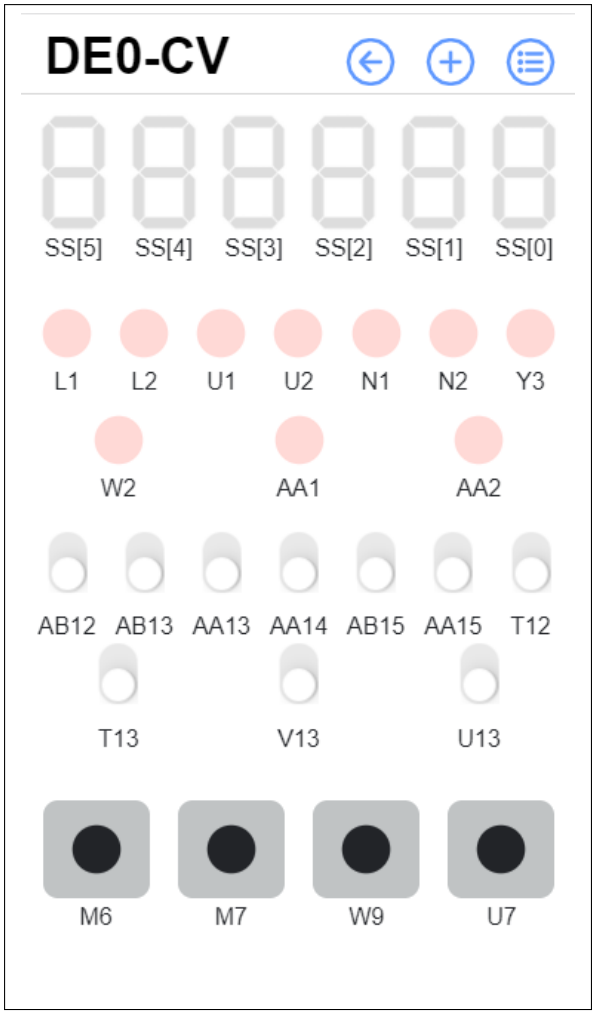


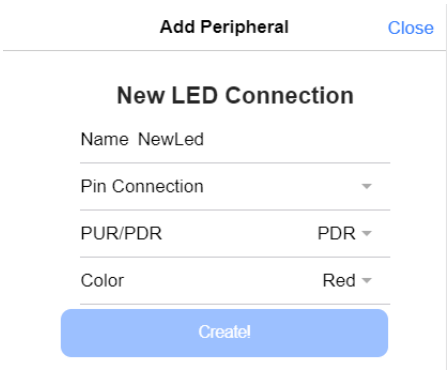
Figure 4.2: DE0-CV on an iPhone.

color: red, green, and miscellaneous. The rows will wrap to the next row if their length exceeds with width of the view port.

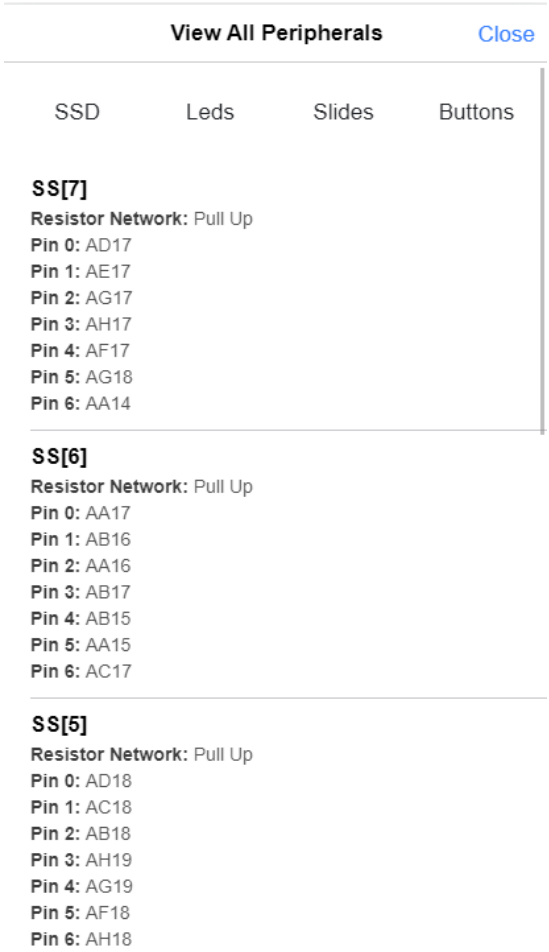
One additional feature of the view FPGA screen is that all interactions from other users on an application connected to the FPGA are seen in real-time. For example, if a user holds down the button at pin U7, the button will be shown as pressed in the live view for all other users connected to the same FPGA server. This is to further emulate an environment where all users are working on the same project in the same room.

The view peripheral button, shown as the list icon in the menu toolbar, is particularly useful as it lists the properties, including the assigned pin, for each displayed peripheral device. Further, this is where a user can remove a device from the view. Within the view peripheral screen, users can select which devices they would like to examine by type: LED, slide switch, seven segment display, or push button. This functionality is shown in

Figure 4.3b. By swiping left on a peripheral, a delete button is exposed, which can then be used to remove that peripheral from the display. All modifications to peripherals, including additions and deletions, are reflected on all connected users' screens.



(**a**) Adding a peripheral to the view.



(**b**) View peripheral properties.

Figure 4.3: Multiple functionalities of the user application.

The lowest-cost experimental result consists of a DE0-CV development board, which implements a Cyclone V FPGA, and a Raspberry Pi Zero. The total academic cost for the system is $120 at the time of the writing. An image of the Raspberry Pi/DE0-CV combination can be seen in Figure 4.4.

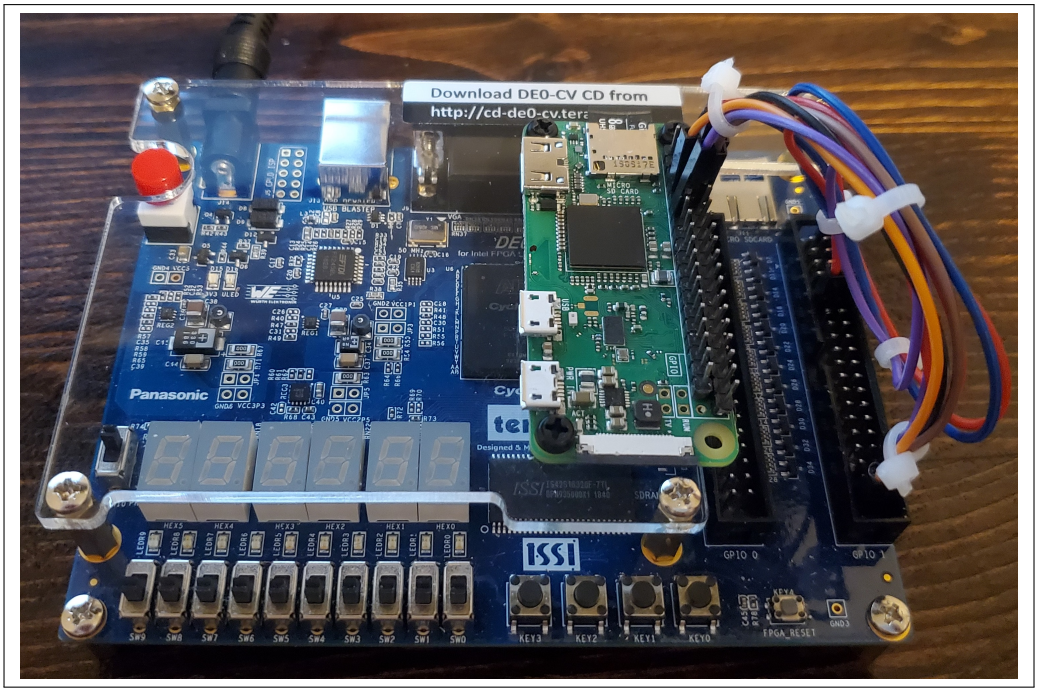


Figure 4.4: Complete remote access FPGA system.

The Python Wizard is essential for incorporating a student lab with the remote IoT capabilities. The Python wizard requires seven inputs: the user ID, the Quartus project folder, the Verilog source code folder, the top module for the project, the folder to put the newly built Quartus project, the target development board, and an option to synchronize the seven segment display updates. If the *Sync SSD* box is left unchecked, the displays will update on the GUI one bit at a time, as opposed to smooth transitions between numbers. Any errors along the way will be displayed under "Build Result"; otherwise, "Build Completed!" will be displayed. The GUI for the Python Wizard can be seen in Figure 4.5.

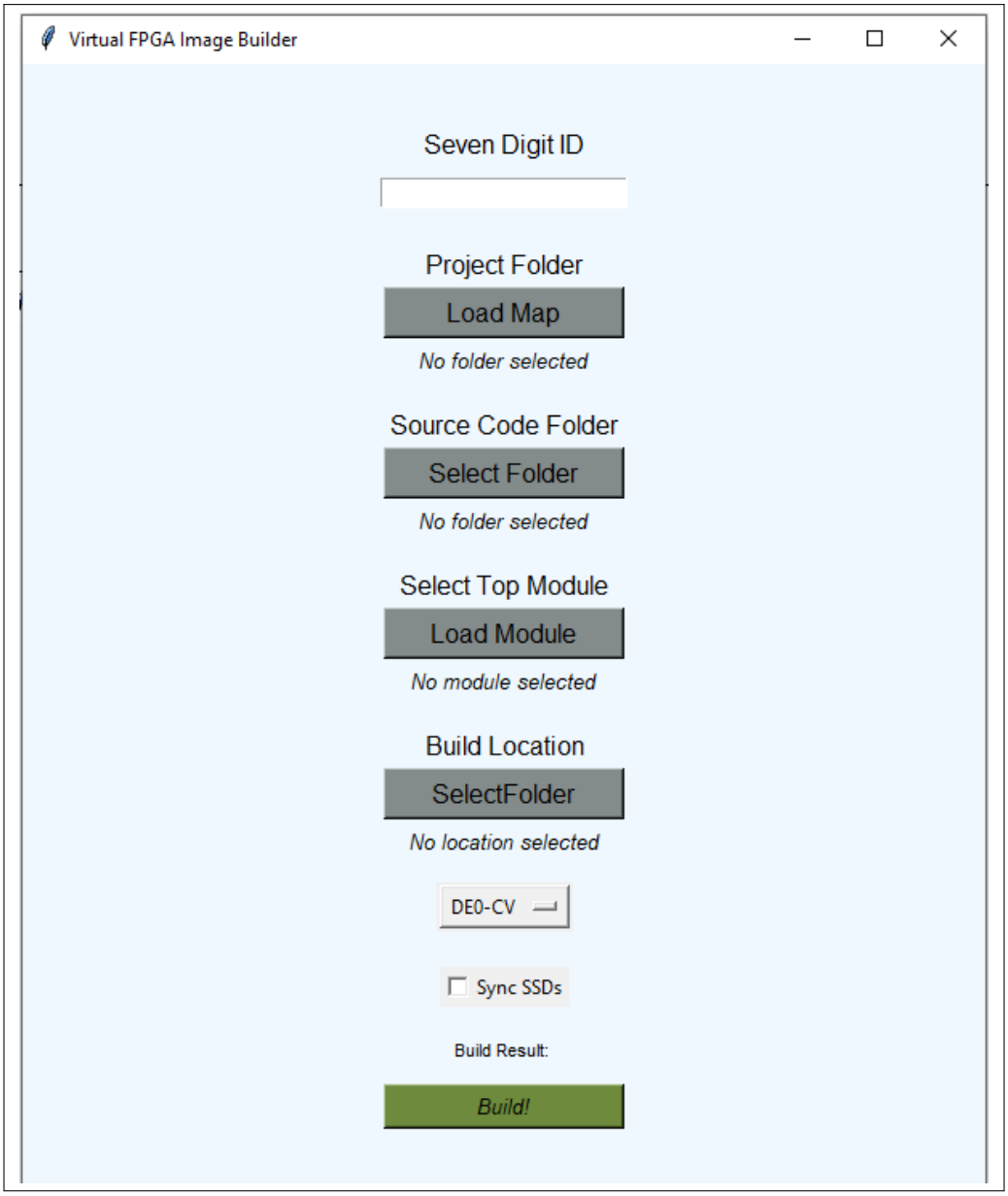


Figure 4.5: Python Wizard user interface.

Resource Usage

The remote FPGA hardware wrapper was designed to be as lightweight as possible, so as not to utilize a large amount of resources on the physical FPGA. This is pertinent as the hardware wrapper for remote applications must utilize as little hardware as possible to allow the user to develop a complex hardware design. Resource utilization for various FPGA chip sets is shown in Table 4.1. Adaptive Logic Modules (ALMs) are the fundamental building block on Intel FPGAs, as they are

composed of two or more look up tables (LUTs) that are used to implement the user-designed digital logic [42]. Note that in the remote FPGA application, the hardware wrapper utilizes less than 1% of available ALMs on the tested platforms. This is ideal for a light-weight development platform such as the remote-access tool. The final synthesis report is shown in Figure 4.6.

Table 4.1: Resource usage for various FPGA chip sets.

FPGA Device	ALM Utilization
DE-0 Dev Board	141 (<1%)
DE2-115 Dev Board	342 (<1%)
Cyclone VE Dev Board	271 (<1%)
Cyclone V SoC Dev Kit	265 (<1%)
Arrow SoCKit	267 (<1%)

Flow Status	Successful - Tue ... 30 18:36:33 2021
Quartus Prime Version	18.1.0 Build 625 0...18 SJ Lite Edition
Revision Name	Lab1_MAGYARI_A
Top-level Entity Name	remoteFPGATOP
Family	Cyclone V
Device	5CEBA4F23C7
Timing Models	Final
Logic utilization (in ALMs)	141 / 18,480 (< 1 %)
Total registers	145
Total pins	30 / 224 (13 %)
Total virtual pins	0
Total block memory bits	0 / 3,153,920 (0 %)
Total DSP Blocks	0 / 66 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 4 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 4.6: FPGA hardware wrapper synthesis report.

Latency

Due to the interactive nature of the remote FPGA Lab, latency between a remote FPGA and a user application is a critical component for correctly interpreting inputs and outputs from the FPGA. With means to measure round-trip data latency between

the user application and FPGA, data was timed as it traveled from the user application to the FPGA for processing, and back again to the application. A lightweight hardware module was loaded to the DE0-CV board in which an output LED was assigned to an input switch, so that the data was immediately reflected in the user application upon reception of an input.

The latency test began timing data transmission upon the flip of a slide switch via the user interface, and ended when the application realized the corresponding updated LED pin. This test was run 10,000 times. The results are displayed in a histogram which is shown in Figure 4.7, and the statistics are summarized in Table 4.2.

Table 4.2: Server response times.

Statistic	**Value**
Total Tests n	10,000
Maximum	6050 [ms]
Minimum	2.75 [ms]
Results greater than 1 [s]	.38%
Results less than .2 [s]	74%
Population Mean σ	196 [ms]
Standard Deviation μ	112 [ms]

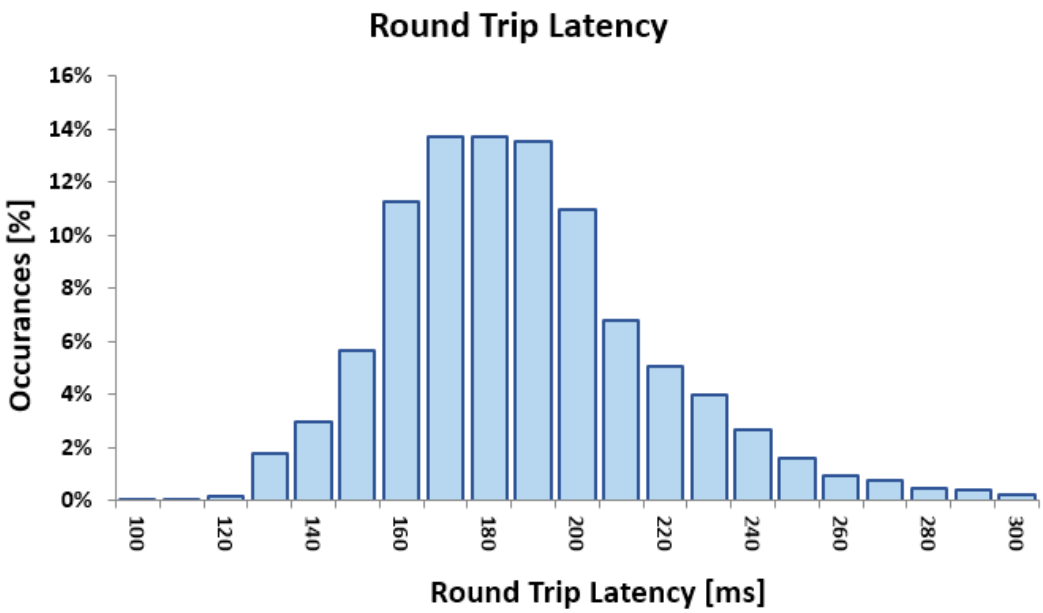


Figure 4.7: Signal round-trip latency histogram.

Discussion

The remote FPGA platform that was developed provides a modern approach when compared to other remote lab methods [2], [3], [7]. By utilizing the Ionic development platform, convenient, mobile access to student laboratories is provided. Students can harness the power of an FPGA by using a personal mobile device to access their remote FPGA design as opposed to having to be in the presence of the physical development board. Further, the approach utilizes cloud applications for communicating changes on the FPGA. By introducing communication via the cloud, the need for a webcam has been eliminated, in turn reducing the overall cost, network usage, and setup time of the system. This is beneficial for both university engineering departments and their respective students, as reduced costs increases the ability to scale the system, and in turn, increases accessibility for students.

This novel design also opens up the possibility for collaboration on student projects by allowing multiple users to access the same remote lab simultaneously. As opposed to single user access [2]–[7], simultaneous access capability enables collaboration, project presentation for online classes, and remote grading capabilities for teaching assistants. As each user application accesses the cloud instead of directly accessing the FPGA, there can be an unlimited amount of users for a single FPGA without reducing quality.

Further, the remote access FPGA allows users to expand upon the development possibilities of an FPGA that this methodology utilizes by giving users the ability to add and remove peripheral devices. For example, if a student were to decide on the classical red–yellow–green stoplight lab, the remote FPGA application gives that student the ability to add additional multicolor LEDs and push buttons necessary to complete their project, despite the fact that the physical FPGA development board may only have red LEDs. This can all be done without incurring the financial cost of buying the physical peripheral or going through the process of setting up a physical resistor network, allowing the students to focus on the digital design aspect of the lab as opposed to the set-up of electronic components.

Future directions for this work are still being explored as well. The remote FPGA system was designed to scale, as the hardware wrapper is portable between FPGA manufacturers and the peripherals displayed in the applications are loaded via a modifiable JSON-file. By adding other JSON files that include peripheral information and pin connections for other development boards, the mobile application can be expanded to include other FPGA board types.

CHAPTER 5: CONCLUSIONS

In this work, an IoT solution for remote digital design laboratories was proposed. A system architecture which includes a cross-platform web application, Google Firebase access, Raspberry Pi, and a host FPGA device was presented. This novel design is unique from other works in that it allows multiple users to collaborate remotely on one FPGA, which is useful for debugging, classroom-style presentations, and group projects. Further, this design does not rely on the use of webcams for displaying FPGA output to the user and can be accessed via a variety of devices such as computers, mobile phones, and tablets. All outputs are still reflected on the physical FPGA development board as well as within a cross-platform application.

The Remote FPGA hardware wrapper can be added to any FPGA project with the Python Wizard, significantly reducing setup times for remote access. The Python Wizard is designed to require minimal user input for simplicity: after loading the user design, various input parameters can be selected via the user interface to customize the remote accessibility of the design. For example, users can choose the development board layout they wish to display within the remote access application and input their user ID for server identification purposes.

Once the user design is being accessed via the remote access application, the user has further ability to customize the FPGA interface layout. While the default user interface layout replicates that of the development board selected in the Python Wizard, the user can add, remove, or modify existing peripheral devices such as LEDs, seven segment displays, slide switches, and push buttons. Further, these additional peripheral devices can be connected to a virtual pull-up or pull-down resistor network, allowing for either positive or negative logic implementations.

All user inputs within the application, such as button presses and toggle switches, are reflected within all other users' applications that are accessing the same design. For example, if one user pushes and holds down a button on their screen, and another user

is accessing the same design via the remote access application, then that pressed button will be reflected on both of their screens.

The hardware wrapper utilizes a UART for communicating signal inputs and outputs, a FIFO for managing signal data packets, and an input and output manager for driving signals to the user's hardware module. Both the input and output manager are generated via the Python Wizard. The input manager drives the input signals on the user module according to data received by the UART, and the output manager reads the user module output signals and loads them into the UART to be transmitted to the Raspberry Pi.

The hardware wrapper design requires less than 1% of hardware resources on tested FPGA chips, and data latency from the application to the FPGA and back has an average round trip time of 196 ms. This delay is an acceptable rate, as it is a near instantaneous response to the user input. This design was developed using the DE0-CV and DE2-115 boards from Intel, but can easily replicate the peripheral devices of other development board with the use of an updated JSON file.

Future plans for this work include adding more JSON board support files for Intel based development boards. Further, new developments are planned that aim to add a password protection feature to the user application to prevent unauthorized users from accessing student projects.

REFERENCES

[1] G. Marinoni, H. Van't Land, and T. Jensen, "The impact of covid-19 on higher education around the world," *IAU Global Survey Report*, 2020.

[2] R. Hashemian and J. Riddley, "Fpga e-lab, a technique to remote access a laboratory to design and test," in *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*, IEEE, 2007, pp. 139–140.

[3] F. Morgan, S. Cawley, F. Callaly, S. Agnew, P. Rocke, M. O'Halloran, N. Drozd, K. Kepa, and B. McGinley, "Remote fpga lab with interactive control and visualisation interface," in *2011 21st International Conference on Field Programmable Logic and Applications*, IEEE, 2011, pp. 496–499.

[4] A. E.-R. Mohsen, M. Y. GadAlrab, Z. elhaya Mahmoud, G. Alshaer, M. Asy, and H. Mostafa, "Remote fpga lab for zynq and virtex-7 kits," in *2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS)*, IEEE, 2019, pp. 185–188.

[5] A. Schwandt and M. Winzker, "Make it open-improving usability and availability of an fpga remote lab," in *2019 IEEE Global Engineering Education Conference (EDUCON)*, IEEE, 2019, pp. 232–236.

[6] Ó. Oballe-Peinado, J. Castellanos-Ramos, J. A. Sánchez-Durán, R. Navas-González, A. Daza-Márquez, and J. A. Botın-Córdoba, "Fpga-based remote laboratory for digital electronics," in *2020 XIV Technologies Applied to Electronics Teaching Conference (TAEE)*, IEEE, 2020, pp. 1–5.

[7] N. Fujii and N. Koike, "Iot remote group experiments in the cyber laboratory: A fpga-based remote laboratory in the hybrid cloud," in *2017 International Conference on Cyberworlds (CW)*, IEEE, 2017, pp. 162–165.

[8] N. K. Jumaa, O. A. Abdulhameed, and R. H. Abbas, "A theoretical background of iot platforms based on fpgas," *Communications on Applied Electronics*, vol. 7, pp. 6–10, 2018.

[9] A. Nauman, Y. A. Qadri, M. Amjad, Y. B. Zikria, M. K. Afzal, and S. W. Kim, "Multimedia internet of things: A comprehensive survey," *IEEE Access*, vol. 8, pp. 8202–8250, 2020.

[10] S. A. Dehkordi, K. Farajzadeh, J. Rezazadeh, R. Farahbakhsh, K. Sandrasegaran, and M. A. Dehkordi, "A survey on data aggregation techniques in iot sensor networks," *Wireless Networks*, vol. 26, no. 2, pp. 1243–1263, 2020.

[11] Z. A. O. Nasri Sulaiman, M. Marhaban, and M. Hamidon, "Design and implementation of fpga-based systems-a review," *Australian Journal of Basic and Applied Sciences*, vol. 3, no. 4, pp. 3575–3596, 2009.

[12] A. Upegui and E. Sanchez, "Evolving hardware by dynamically reconfiguring xilinx fpgas," in *International Conference on Evolvable Systems*, Springer, 2005, pp. 56–65.

[13] A. Al-Safi, A. Al-Khayyat, A. M. Manati, and L. Alhafadhi, "Advances in fpga based pwm generation for power electronics applications: Literature review," in *2020 11th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, IEEE, 2020, pp. 0252–0259.

[14] S. Rasoulinezhad, D. Boland, and P. H. Leong, "Mlblocks: Fpga blocks for machine learning applications," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021, pp. 228–228.

[15] X. Wang, C. Li, and J. Song, "Motion image processing system based on multi core fpga processor and convolutional neural network," *Microprocessors and Microsystems*, vol. 82, p. 103 923, 2021.

[16] P. Liu, W. Qingqing, and W. Liu, "Enterprise human resource management platform based on fpga and data mining," *Microprocessors and Microsystems*, vol. 80, p. 103 330, 2021.

[17] V. Jayakrishnan and C. Parikh, "Embedded processors on fpga: Soft vs hard," in *Proceedings of the 2019 ASEE North Central Section Conference*, 2019.

[18] R. Höller, D. Haselberger, D. Ballek, P. Rössler, M. Krapfenbauer, and M. Linauer, "Open-source risc-v processor ip cores for fpgas—overview and evaluation," in *2019 8th Mediterranean Conference on Embedded Computing (MECO)*, IEEE, 2019, pp. 1–6.

[19] Z. Zang, Y. Liu, and R. C. Cheung, "Reconfigurable risc-v secure processor and soc integration," in *2019 IEEE International Conference on Industrial Technology (ICIT)*, IEEE, 2019, pp. 827–832.

[20] J. M. Szefer, W. Zhang, Y.-Y. Chen, D. C. 3, K. Chan, W. X. Li, R. C. Cheung, and R. B. Lee, "
rapid single-chip secure processor prototyping on the opensparc fpga platform," in *22nd IEEE International Symposium on Rapid System prototyping*, IEEE, 2011.

[21] S.-Z. Huang and R.-Q. Chen, "Fpga-based iot sensor hub," in *2018 International Conference on Sensor Networks and Signal Processing (SNSP)*, IEEE, 2018, pp. 139–144.

[22] C. Z. Myint, L. Gopal, and Y. L. Aung, "Reconfigurable smart water quality monitoring system in iot environment," in *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*, 2017, pp. 435–440. DOI: 10.1109/ICIS.2017.7960032.

[23] M. S. BenSaleh, S. M. Qasim, A. A. AlJuffri, and A. M. Oheid, "Design of an advanced system-on-chip architecture for internet-enabled smart mobile devices," in *2018 30th International Conference on Microelectronics (ICM)*, IEEE, 2018, pp. 323–326.

[24] R. A. Ghate, S. K. Tilekar, and S. V. Chavan, "Comparative study of intelligent and smart development platforms employed for internet of thing's applications," *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, vol. 12, no. 12, pp. 810–821, 2021.

[25] A. Rupani, P. Whig, G. Sujediya, and P. Vyas, "A robust technique for image processing based on interfacing of raspberry-pi and fpga using iot," in *2017 International Conference on Computer, Communications and Electronics (Comptelix)*, IEEE, 2017, pp. 350–353.

[26] Q. Huang, K. Rodriguez, N. Whetstone, and S. Habel, "Rapid internet of things (iot) prototype for accurate people counting towards energy efficient buildings.," *J. Inf. Technol. Constr.*, vol. 24, pp. 1–13, 2019.

[27] Y. Wang and S. Jang, "A pulse sensor interface design for fpga based multisensor health monitoring platform," 2019.

[28] R. P. Foundation, *Raspberry pi documentation*. [Online]. Available: `https://www.raspberrypi.org/documentation`.

[29] B. Foundation, *Beaglebone documentation*. [Online]. Available: `https://beagleboard.org/boards`.

[30] I. Jaziri, L. Charaabi, and K. Jelassi, "Remote web-based control laboratories using embedded linux and field-programmable gate array," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 232, no. 9, pp. 1146–1154, 2018.

[31] I. Jaziri, L. Chaarabi, and K. Jelassi, "A remote dc motor control using embedded linux and fpga," in *2015 7th International Conference on Modelling, Identification and Control (ICMIC)*, 2015, pp. 1–5. DOI: `10.1109/ICMIC.2015.7409332`.

[32] M. Elnawawy, A. Farhan, A. Al Nabulsi, A. Al-Ali, and A. Sagahyroon, "Role of fpga in internet of things applications ," in *2019 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, IEEE, 2019, pp. 1–6.

[33] S. Kang, J. Moon, and S. Jun, "Fpga-accelerated time series mining on low-power iot devices," in *2020 IEEE 31st International Conference on Application-*

specific Systems, Architectures and Processors (ASAP), 2020, pp. 33–36. DOI: 10.1109/ASAP49362.2020.00015.

[34] R. Ferdian, R. Aisuwarya, and T. Erlina, "Edge computing for internet of things based on fpga," in *2020 International Conference on Information Technology Systems and Innovation (ICITSI)*, IEEE, 2020, pp. 20–23.

[35] G.-M. Sung, C.-T. Lee, and C.-R. Chen, "Iot-based home care system with a fpga development board by using rs-485 interface and verilog hdl," in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, IEEE, 2020, pp. 3370–3374.

[36] L. Peng, Z. Xin, and G. Ping, "Design and implementation of remote deepface model face recognition system based on sbrio fpga platform and nb-iot module," in *2019 2nd International Conference on Safety Produce Informatization (IICSPI)*, IEEE, 2019, pp. 505–509.

[37] S. Wang, Y. Hou, F. Gao, and X. Ji, "A novel iot access architecture for vehicle monitoring system," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, IEEE, 2016, pp. 639–642.

[38] K. C. Gophane and P. Bhaskar, "Fpga based adaptive iot framework for distinct applications," in *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, IEEE, 2018, pp. 1–6.

[39] L. R. Brasilino and M. Swany, "Low-latency coap processing in fpga for the internet of things," in *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, IEEE, 2019, pp. 1057–1064.

[40] *Raspberry pi documentation*. [Online]. Available: https://www.raspberrypi.org/documentation/computers/configuration.html.

[41] "Realtime database limits - firebase realtime database ," *Google* , 2021. [Online]. Available: https://firebase.google.com/docs/database/usage/limits.

[42] "Fpga architecture," *Intel Altera*, Jul. 2006. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01003.pdf.

PYTHON WIZARD

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Dec 3 18:12:57 2020

@author: Alexander Magyari
"""
import re
import json
import collections
import shutil
import errno
from pathlib import Path
import os
import glob
from tkinter import Tk, StringVar, IntVar, filedialog, messagebox,
    Label, Entry, Button, N, CENTER, Checkbutton, OptionMenu

RX_DICT = {
    'comment': re.compile(r'//(?P<comment>.*)\n'),
    'commentBlockStart': re.compile(r'/\*(?P<commentStart>.*)\n'),
    'commentBlockEnd': re.compile(r'(?P<commentEnd>.*)\n')
}


##########################################################
############## Initialize tkinter window ################
##########################################################

def main():
```

```
29    locs = {
30        "buiild_loc_dest": "",
31        "source_code_folder_dest": "",
32        "top_mod_dest": "",
33        "project_fold_dest": "",
34        "pin_map_dest": ""
35    }
36
37    # If board are added here, the board pinouts must be
38    # uploaded to the fpga "boards" folder as a .json, and the
39    # board index in this list be updated in the RPi file as well.
40    # It is important that new boards are appended to the end of the
41    # list!
42
43    boards = [
44        "DE0-CV",
45        "DE2-115"
46    ]
47
48    default_prgrm_output = "Build Result: "
49    prgrm_output = default_prgrm_output + ""
50    prgrm_output_wrapper = [default_prgrm_output, prgrm_output]
51    root = Tk()
52    PSID = StringVar()
53    sync_ssds = IntVar()
54
55    # # This is the section of code which creates the main window
56    root.geometry('600x700')
57    root.configure(background='#F0F8FF')
58    root.title('Virtual FPGA Image Builder')
59
```

```
Label(root, text='Seven Digit ID', bg='#F0F8FF', font=('arial', 12, 'normal')).place(relx=0.5, y=50, anchor=CENTER)
Entry(root, width=25, textvariable=PSID).place(relx=0.5, y=80, anchor=CENTER)

#
# # This is the section of code which creates the a label
Label(root, text='Source Code Folder', bg='#F0F8FF', font=('arial', 12, 'normal')).place(relx=0.5, y=225, anchor=CENTER)
# # This is the section of code which creates the a label
source_code_folder_label = Label(root, text='No folder selected', bg='#F0F8FF', font=('arial', 10, 'italic'), width=100, anchor=CENTER)
source_code_folder_label.place(relx=0.5, y=285, anchor=CENTER)
# # This is the section of code which creates a button
Button(root, text='Select Folder', bg='#838B8B', font=('arial', 12, 'normal'), command=lambda: browse_for_sc_folder(locs, root, source_code_folder_label)).place(relx=0.5, y=255, width=150, anchor=CENTER)


#
# # This is the section of code which creates the a label
Label(root, text='Select Top Module', bg='#F0F8FF', font=('arial', 12, 'normal')).place(relx=0.5, y=325, anchor=CENTER)
# # This is the section of code which creates the a label
top_module_label = Label(root, text='No module selected', bg='#F0F8FF', font=('arial', 10, 'italic'), width=100, anchor=CENTER)
top_module_label.place(relx=0.5, y=385, anchor=CENTER)
# # This is the section of code which creates a button
```

```
Button(root, text='Load Module', bg='#838B8B', font=('arial', 12, 'normal'), command=lambda: browse_for_top_mod(locs, top_module_label)).place(relx=0.5, y=355, width=150, anchor=CENTER)

#
# # This is the section of code which creates the a label
Label(root, text='Project Folder', bg='#F0F8FF', font=('arial', 12, 'normal')).place(relx=0.5, y=125, anchor=CENTER)
# # This is the section of code which creates the a label
pinmap_label = Label(root, text='No folder selected', bg='#F0F8FF', font=('arial', 10, 'italic'), width=100, anchor=CENTER)
pinmap_label.place(relx=0.5, y=185, anchor=CENTER)
# # This is the section of code which creates a button
Button(root, text='Load Map', bg='#838B8B', font=('arial', 12, 'normal'), command=lambda: browse_for_project_folder(locs, root, pinmap_label)).place(relx=0.5, y=155, width=150, anchor=CENTER)

#
# # This is the section of code which creates the a label
Label(root, text='Build Location', bg='#F0F8FF', font=('arial', 12, 'normal')).place(relx=0.5, y=425, anchor=CENTER)
# # This is the section of code which creates the a label
build_location_label = Label(root, text='No location selected', bg='#F0F8FF', font=('arial', 10, 'italic'), width=100, anchor=CENTER)
build_location_label.place(relx=0.5, y=485, anchor=CENTER)
# # This is the section of code which creates a button
Button(root, text='SelectFolder', bg='#838B8B', font=('arial', 12, 'normal'), command=lambda: browse_for_build_loc(locs, root,
```

```
    build_location_label)).place(relx=0.5, y=455, width=150,
    anchor=CENTER)

# y = 525
user_board = StringVar(root)
user_board.set(boards[0])
board_dropdown = OptionMenu(root, user_board, *boards)
board_dropdown.place(relx=0.5, y=525, anchor=CENTER)

c1 = Checkbutton(root, text='Sync SSDs', variable=sync_ssds,
    onvalue=1, offvalue=0)
c1.place(relx=0.5, y=575, anchor=CENTER)

# # This is the section of code which creates a button
Button(root, text='Build!', bg='#6E8B3D', font=('arial', 10,
    'italic'), command=lambda: build_program(prgrm_output_wrapper,
    locs["source_code_folder_dest"], locs["top_mod_dest"],
    locs["project_fold_dest"], locs["buiild_loc_dest"],
    locs["pin_map_dest"], PSID.get(), program_output_label,
    build_config_variable(sync_ssds.get()), build_board(boards,
    user_board.get()),
    boards.index(user_board.get()))).place(relx=0.5, y=650,
    width=150, anchor=CENTER)


# # This is the section of code which creates the a label
program_output_label = Label(root, text=prgrm_output, bg='#F0F8FF',
    font=('arial', 8, 'normal'))
program_output_label.place(relx=0.5, y=605, width=500, anchor=N)
root.mainloop()
```

```
    root.quit()




class Port:

    def __init__(self, n="", ic=False, d="input", w=0, p=None):
        self.name = n
        self.is_clock = ic
        self.direction = d
        self.width = w
        self.ports = p

    def set_port_width(self):
        for _ in range(self.width):
            self.ports.append("emptyPort")

    def set_port(self, port_number, value):
        self.ports[port_number] = value

def build_board(board_array, board):
    loc = board_array.index(board)
    return binary_for_verilog(loc, 8)

def build_config_variable(sync_ssds):
    """


    Parameters
```

```
    ----------
    sync_ssds : INT
        Determines if the FPGA should sync seven segment display output.
            1= y, 0 = n

    Returns
    -------
    8 bit int as string formatted for verilog, defined as follows:
        {empty, empty, empty, empty, empty, empty, empty, sync_ssds}

    """
    config_var = 0
    config_var += sync_ssds

    return binary_for_verilog(config_var, 8)



def copy_folder(src, dst):
    """
    Copys a folder

    Parameters
    ----------
    src : STRING
        folder to copy.
    dst : STRING
        location to copy folder to.

    Returns
    -------
```

```
    None.

    """
    try:
        if Path(dst).is_dir():
            if not any(file.endswith(".qsf") for file in os.listdir(dst)):
                print("Trying to copy to non-empty and non-project based directory!")
                return
            else:
                shutil.rmtree(dst)
        shutil.copytree(src, dst)
    except OSError as exc: # python >2.5
        if exc.errno == errno.ENOTDIR:
            shutil.copy(src, dst)
        else: raise


def first(iterable, default=None):
    """
    Returns the first match in an iterable

    Parameters
    ----------
    iterable : LIST
        list to check for a match in .
    default : ANY, optional
        Item to return if no match is found. The default is None.

    Returns
```

```
    -------
    TYPE
        DESCRIPTION.

    """
    for item in iterable:
        return item
    return default


def _parse_line(line):
    """
    Do a regex search against all defined regexes and
    return the key and match result of the first matching regex

    """

    for key, rx in RX_DICT.items():
        match = rx.search(line)
        if match:
            return key, match
    # if there are no matches
    return None, None


def remove_comments(string):
    """
    Removes comments from verilog file.


    Parameters
```

```
    ----------
    string : TYPE
        Code to remove comments from

    Returns
    -------
    TYPE
        Input code sans comments

    """
    pattern = r"(\".*?(?<!\\)\"|\'.*?(?<!\\)\')|(/\*.*?\*/|//[^\r\n]*$)"
    # first group captures quoted strings (double or single)
    # second group captures comments (//single-line or /* multi-line */)
    regex = re.compile(pattern, re.MULTILINE|re.DOTALL)
    def _replacer(match):
        if match.group(2) is not None:
            return ""
        else:
            return match.group(1) # captured quoted-string
    return regex.sub(_replacer, string)


def binary_for_verilog(digit, length):
    bin_basic = bin(int(digit)).replace("0b", "")
    while len(bin_basic) < length:
        bin_basic = "0" + bin_basic
    bin_basic = str(length) + "'b" + bin_basic
    return bin_basic


def convert_embedded_digits_to_binary(string, max_digit):
```

```
    start_loc = string.find("{{CONVERT_TO_BINARY:")
    min_length = len(bin(max_digit).replace("0b", ""))

    while start_loc > 0:
        end_loc = string[start_loc:].find("}}") + start_loc
        digit_to_convert = string[start_loc +
            len("{{CONVERT_TO_BINARY:"):end_loc]
        digit_converted = binary_for_verilog(digit_to_convert,
            min_length)
        string = string[:start_loc] + digit_converted + string[end_loc +
            len("}}"):]
        start_loc = string.find("{{CONVERT_TO_BINARY:")
    print("Req bits ", max_digit, min_length)
    return string

# this is the function called when the button is clicked
def browse_for_sc_folder(dic, root, source_code_folder_label):
    def_location = "../."
    if dic["project_fold_dest"] != "":
        def_location = dic["project_fold_dest"]
    dic["source_code_folder_dest"] = filedialog.askdirectory(
        parent=root,
        initialdir=def_location,
        title='Select your source code folder')
    if not any(fname.endswith('.v') for fname in
        os.listdir(dic["source_code_folder_dest"])):
        messagebox.showinfo(
            "Warning!",
            "The selected folder has no .v files! This surely is not
                your source code folder.")
        dic["source_code_folder_dest"] = ""
```

```
296     else:
297         source_code_folder_label.configure(text=
298                                             ["source_code_folder_dest"])
299
300
301 # this is the function called when the button is clicked
302 def browse_for_top_mod(dic, top_module_label):
303     def_location = "../."
304     if dic["source_code_folder_dest"] != "":
305         def_location = dic["source_code_folder_dest"]
306     elif dic["project_fold_dest"] != "":
307         def_location = dic["project_fold_dest"]
308     dic["top_mod_dest"] = filedialog.askopenfilename(
309         initialdir=def_location, title="Select top module",
310         filetypes=[("Verilog files", "*.v")])
311     if dic["top_mod_dest"]:
312         top_module_label.configure(text=dic["top_mod_dest"])
313
314
315 # this is the function called when the button is clicked
316 def browse_for_project_folder(dic, root, pinmap_label):
317     dic["project_fold_dest"] = filedialog.askdirectory(
318         parent=root,
319         initialdir="./..",
320         title='Select your main project folder')
321     file_count = 0
322     for fname in os.listdir(dic["project_fold_dest"]):
323         if fname.endswith('.qsf'):
324             dic["pin_map_dest"] = dic["project_fold_dest"] + "/" + fname
325             file_count += 1
326
```

```
    if file_count == 0:
        dic["project_fold_dest"] = ""
        messagebox.showinfo("Warning!",
                            "Your main project folder must be the location of a precompiled "+
                            "Quartus Project. There should be a .qsf file in this folder.")
    elif file_count > 1:
        dic["project_fold_dest"] = ""
        messagebox.showinfo("Warning!",
                            "There are multiple .qsf files located in this folder. Please ensure"+
                            " only one .qsf file exists.")
    elif dic["buiild_loc_dest"] == dic["project_fold_dest"]:
        dic["project_fold_dest"] = ""
        messagebox.showinfo("Warning!",
                            "The selected folder is the same as your build folder. You must select"+
                            " a different folder.")
    else:
        pinmap_label.configure(text=dic["project_fold_dest"])


# this is the function called when the button is clicked
def browse_for_build_loc(dic, root, build_location_label):
    dic["buiild_loc_dest"] = filedialog.askdirectory(parent=root,
                                                     initialdir="./../Build",
                                                     title='Select folder to build new '+
                                                     'project')
    if dic["buiild_loc_dest"] == dic["project_fold_dest"]:
```

```
        dic["buiild_loc_dest"] = ""
        messagebox.showinfo("Warning!",
                            "The selected folder is the same as your main
                                project folder. You"+
                            " must select a different folder.")
    else:
        build_location_label.configure(text=dic["buiild_loc_dest"])



def set_build_output(string, program_output_label):
    print(string)
    program_output_label.configure(text=string)

def build_program(program_output_text, source_code_folder_dest,
    top_mod_dest,
                  project_fold_dest, buiild_loc_dest, pin_map_dest,
                      people_soft_id,
                  program_output_label, configuration_variable,
                      board_to_use, board_name):
    try:
        print(people_soft_id)
        error_message = ""
        if len(people_soft_id) != 7:
            error_message += "\nPSID must be exactly 7 digits long!"
        if not people_soft_id.isdigit():
            error_message += "\nPSID must only be numbers!"
        if not source_code_folder_dest:
            error_message += "\nNo source code folder selected!"
            print(source_code_folder_dest)
        if not top_mod_dest:
```

```
380             error_message += "\nNo top module verilog file selected!"
381         if not project_fold_dest:
382             error_message += "\nNo project folder selected!"
383         if not buiild_loc_dest:
384             error_message += "\nNo build location selected!"
385         if error_message:
386             set_build_output(program_output_text[0] + error_message +
                    "\nBuild not completed.", program_output_label)
387             return
388 
389         ##########################################################
390         ################## Create Build Folder ##################
391         ##########################################################
392         build_folder = buiild_loc_dest + "/build"
393         copy_folder(project_fold_dest, build_folder)
394         shutil.copy("inc/basefiles/FIFO.v", build_folder + "/src")
395         shutil.copy("inc/basefiles/deserializer.v", build_folder +
                "/src")
396         shutil.copy("inc/basefiles/serializer.v", build_folder + "/src")
397 
398         port_list = []
399         ##########################################################
400         #### Extract port size and direction from top module ####
401         ##########################################################
402         student_module_instance = ""
403         with open(top_mod_dest, 'r') as file:
404             no_comments = remove_comments(file.read())
405             no_tabs = re.sub(r'(^[ \t]+)', '', no_comments, flags=re.M)
406             no_new_lines = re.sub(r"\n*", "", no_tabs)
407             lines = no_new_lines.split(';')
408             for line in lines:
```

```
409                 if line.startswith('module'):
410                     student_module_instance = line
411                     ports =
                            line[line.find("(")+1:line.find(")")].split(',')
412                     for port in ports:
413                         p = Port("newPort", False, "undef", 1, [])
414                         port_string = port.split()
415                         actual_name = port_string[len(port_string) - 1]
416                         p.name = actual_name.strip()
417                         if len(port_string) > 1:
418                             if port_string[0] == "output":
419                                 p.direction = "output"
420                             if port_string[0] == "input":
421                                 p.direction = "input"
422                             width_str = port_string[len(port_string) - 2]
423                             b_one = width_str.find("[")+1
424                             b_two = width_str.find("]")
425                             if not (b_one == 0 and b_two == -1):
426                                 width = width_str[b_one:b_two]
427                                 width = width.replace(" ", "")
428                                 width = width.replace("[", "")
429                                 width = width.replace("]", "")
430                                 width = width.split(":")
431                                 width = int(width[0]) - int(width[1]) + 1
432                                 p.width = width
433                         port_list.append(p)
434
435                 if line.startswith('input'):
436                     line = line[5:len(line) + 1]
437                     no_extra_space = re.sub(r"(\s+)", " ", line)
438                     b_one = no_extra_space.find("[")+1
```

```
439                 b_two = no_extra_space.find("]")
440                 width = no_extra_space[b_one:b_two]
441                 if b_one == 0 and b_two == -1:
442                     width = 1
443                     port_names = line.split(',')
444                 else:
445                     width = width.replace(" ", "")
446                     width = width.replace("[", "")
447                     width = width.replace("]", "")
448                     width = width.split(":")
449                     width = int(width[0]) - int(width[1]) + 1
450                     port_names = no_extra_space[b_two +
                            1:len(line)].split(',')
451                 for p in port_names:
452                     actual_name = p.split()
453                     actual_name = actual_name[len(actual_name) - 1]
454                     po = first(x for x in port_list if x.name ==
                            actual_name.strip())
455                     if po is not None:
456                         po.direction = "input"
457                         po.width = width
458
459
460             if line.startswith('output'):
461                 line = line[6:len(line) + 1]
462                 no_extra_space = re.sub(r"(\s+)", " ", line)
463                 b_one = no_extra_space.find("[")+1
464                 b_two = no_extra_space.find("]")
465                 width = no_extra_space[b_one:b_two]
466                 if b_one == 0 and b_two == -1:
467                     width = 1
```

```
468                     port_names = line.split(',')
469                 else:
470                     width = width.replace(" ", "")
471                     width = width.replace("[", "")
472                     width = width.replace("]", "")
473                     width = width.split(":")
474                     width = int(width[0]) - int(width[1]) + 1
475                     port_names = no_extra_space[b_two +
                            1:len(line)].split(',')
476                 for p in port_names:
477                     actual_name = p.split()
478                     actual_name = actual_name[len(actual_name) - 1]
479                     po = first(x for x in port_list if x.name ==
                            actual_name.strip())
480                     if po is not None:
481                         po.direction = "output"
482                         po.width = width
483
484         for port in port_list:
485             port.set_port_width()
486
487         ##########################################################
488         ########## Extract port mapping from .qsf file ##########
489         ##########################################################
490         with open(pin_map_dest, 'r') as file:
491             lines = file.readlines()
492             for line in lines:
493                 if line.startswith("set_location_assignment"):
494                     info =
                            line[len("set_location_assignment"):len(line)-1]
495                     pin = info.split()[0]
```

```
496                 location = info.split()[2]
497                 b_one = location.find("[")+1
498                 b_two = location.find("]")
499                 if b_one == 0 and b_two == -1:
500                     loc_number = 0
501                 else:
502                     loc_number = location[b_one:b_two]
503                     location = location[0:b_one - 1]
504
505                 pf = first(x for x in port_list if x.name ==
                        location.strip())
506                 if pf is not None:
507                     pf.set_port(int(loc_number), pin[4:len(pin)])
508
509
510         ##########################################################
511         ######## Load pin list for binary enumeration ###########
512         ##########################################################
513         pin_dict = {}
514         pin_reset_vals = {}
515         print("BN: " + str(board_name))
516         if board_name == 0:
517             json_data = json.load(open('inc/de0-cv.json', 'r'))
518         elif board_name == 1:
519             json_data = json.load(open('inc/de2115.json', 'r'))
520         else:
521             print("File not found. Board code: " + str(board_name))
522         counter = 0
523         for key, value in json_data['Pins'].items():
524             pin_dict[key] = 0
525             counter += 1
```

```
            pin_reset_vals[key] = value
        pin_dict = collections.OrderedDict(sorted((pin_dict.items())))
        counter = 0
        for key, value in pin_dict.items():
            pin_dict[key] = str(format(counter, '015b'))
            counter += 1
            print(key, ":", pin_dict[key])


        ##########################################################
        ###### Build Replacement Strings for Build files ########
        ##########################################################
        input_list_string = ""
        input_reset_string = ""
        input_process_string = ""

        output_list_string = ""
        output_pin_parameters = ""
        output_reg_init_string = ""
        output_reset_string = ""
        output_logic_string =
            "\t\t\t\t\t{{CONVERT_TO_BINARY:2}}:\n\t\t\t\t\tbegin\n"
        max_output_state = 3
        nested_ifs = 0
        for p in port_list:
            reset_value = ""
            for i in range(p.width):
                if p.ports[i] in pin_dict:
                    reset_value += str(pin_reset_vals[p.ports[i]])
                else:
                    p.is_clock = True
```

```
556
557             ### Build input strings
558             if p.direction == "input" and not p.is_clock:
559                 ### {{INPUT_LIST_HERE}} Modifier
560
561                 if len(input_list_string) > 0:
562                     input_list_string += ", "
563                 input_list_string += "output reg "
564                 if p.width - 1 > 0:
565                     input_list_string += "[" + str(p.width - 1) + ":0] "
566                 input_list_string += "stu_" + p.name
567                 ### {{RESET_INPUTS_HERE}} Modifier
568                 input_reset_string += "\t\t\t" + "stu_" + p.name + " <= "
                        + str(p.width) + "'b"
569                 # for b in range(p.width):
570                 #     input_reset_string += "0"
571                 input_reset_string += reset_value
572                 input_reset_string += ";\n"
573                 ### {{PROCESSED_INPUTS_HERE}} Modifier
574                 for b in range(p.width):
575                     input_process_string += ("\t\t\t\t\t\t15'b" +
                            pin_dict[p.ports[b]]
576                                              + ": " + "stu_" + p.name)
577                     if p.width - 1 > 0:
578                         input_process_string += "[" + str(b) + "]"
579                     input_process_string += " <=
                            ___DATA_TO_PROCESS___[0];\n"
580
581             ### Build Output strings
582             if p.direction == "output" and not p.is_clock:
583                 ### {{OUTPUT_LIST_HERE}} Modifier
```

```
584                 if len(output_list_string) > 0:
585                     output_list_string += ", "
586                 output_list_string += "input "
587                 if p.width - 1 > 0:
588                     output_list_string += "[" + str(p.width - 1) + ":0] "
589                 output_list_string += "stu_" + p.name
590                 ### {{OUTPUT_PARAMETERS_HERE}}
591                 param_string = "parameter "
592                 if p.width - 1 > 0:
593                     param_string += "[" + str(p.width * 15 - 1) + ":0] "
594                 param_string += "PIN" + "stu_" + p.name + " = "
595                 if p.width - 1 > 0:
596                     param_string += "{"
597                 for b in reversed(range(p.width)):
598                     param_string += "15'b" + pin_dict[p.ports[b]]
599                     if b != 0:
600                         param_string += ", "
601                 if p.width - 1 > 0:
602                     param_string += "}"
603                 param_string += ";\n"
604                 output_pin_parameters += param_string
605                 ### {{OUTPUT_CACHE_HERE}}
606                 register_string = "reg "
607                 if p.width - 1 > 0:
608                     register_string += "[" + str(p.width - 1) + ":0] "
609                 register_string += "CACHEstu_" + p.name + ";\n"
610                 output_reg_init_string += register_string
611                 ### {{RESET_CACHE_HERE}}
612                 output_reset_string += "\t\t\t\tCACHEstu_" + p.name + "
                    <= 0;\n"
613                 ### {{OUTPUT_LOGIC_HERE}}
```

```
614                 if nested_ifs > 5:
615                     output_logic_string += ("\t\t\t\t\t" +
                            "{{CONVERT_TO_BINARY:" + str(max_output_state) +
616                                                "}}:" + "\n\t\t\t\t\tbegin\n")
617                     nested_ifs = 0
618                     max_output_state += 1
619                 if nested_ifs != 0:
620                     output_logic_string += "\n\t\t\t\t\t\telse\n"
621                 if_statement = "\t\t\t\t\t\tif (" + "CACHEstu_" + p.name
                        + " != " + "stu_" + p.name
622                 if_statement +=
                        ")\n\t\t\t\t\t\tbegin\n\t\t\t\t\t\t\tdataReady <=
                        1'b1;\n"
623                 output_logic_string += if_statement
624                 if p.width - 1 == 0:
625                     nested_if_body = "\n\t\t\t\t\t\t\tdataOut <= " +
                            "PIN" + "stu_" + p.name + "[14:7];"
626                     nested_if_body +=
                            "\n\t\t\t\t\t\t\tSECOND_HALF_OF_SIGNAL <= {" +
                            "PIN" + "stu_" + p.name + "[6:0], stu_" + p.name +
                            "};"
627                     nested_if_body += "\n\t\t\t\t\t\t\tCACHEstu_" +
                            p.name + " <= stu_" + p.name + ";\n"
628                     output_logic_string += nested_if_body
629                     nested_ifs += 1
630                 else:
631                     nested_if_body = ""
632                     for b in range(p.width):
633                         if nested_if_body != "":
634                             nested_if_body = "\n\t\t\t\t\t\t\telse\n"
635                         """
```

```
636                 nested_if_body += "\t\t\t\t\t\t\tif " +
                        "(CACHEstu_" + p.name + "[" + str(b) + "] ^ "
                        + "stu_" + p.name + "[" + str(b) + "] == 1'b1)"
637                 nested_if_body += "\n\t\t\t\t\t\t\tbegin"
638                 nested_if_body += "\n\t\t\t\t\t\t\t\tdataOut <=
                        {" + "PIN" + "stu_" + p.name + "[" + str((b +
                        1) * p.width - 1) + ":" + str(b * p.width) +
                        "], stu_" + p.name + "[" + str(b) + "]};"
639                 nested_if_body += "\n\t\t\t\t\t\t\t\tCACHEstu_" +
                        p.name + "[" + str(b) + "] <= stu_" + p.name +
                        "[" + str(b) + "];"
640                 nested_if_body += "\n\t\t\t\t\t\t\tend"
641                 """
642                 nested_if_body += "\t\t\t\t\t\t\tif " +
                        "(CACHEstu_" + p.name + "[" + str(b) + "] ^ "
                        + "stu_" + p.name + "[" + str(b) + "] == 1'b1)"
643                 nested_if_body += "\n\t\t\t\t\t\t\tbegin"
644                 nested_if_body += "\n\t\t\t\t\t\t\t\tdataOut <=
                        {" + "PIN" + "stu_" + p.name + "[" + str(b *
                        15 + 14) + ":" + str(b * 15 + 7) + "]};"
645                 nested_if_body +=
                        "\n\t\t\t\t\t\t\t\tSECOND_HALF_OF_SIGNAL <= {"
                        + "PIN" + "stu_" + p.name + "[" + str(b * 15 +
                        6) + ":" + str(b * 15) + "], stu_" + p.name +
                        "[" + str(b) + "]};"
646                 nested_if_body += "\n\t\t\t\t\t\t\t\tCACHEstu_" +
                        p.name + "[" + str(b) + "] <= stu_" + p.name +
                        "[" + str(b) + "];"
647                 nested_if_body += "\n\t\t\t\t\t\t\tend"
648                 output_logic_string += nested_if_body
649                 nested_ifs += 1
```

```
650                 output_logic_string += "\n\t\t\t\t\t\t\t\tCURRENT_STATE <=
                        " + "{{CONVERT_TO_BINARY:" + str(1) + "}};\n"
651                 output_logic_string += "\t\t\t\t\t\t\tNEXT_STATE_BUFFER
                        <= " + "{{CONVERT_TO_BINARY:" + str(max_output_state
                        - 1) + "}};\n\t\t\t\t\t\tend"
652                 if p == port_list[-1] or nested_ifs > 5:
653                     output_logic_string +=
                            "\n\t\t\t\t\t\telse\n\t\t\t\t\t\tbegin\n\t\t\t\t\t\t\t
                        CURRENT_STATE <= "
654                     if p == port_list[-1]:
655                         output_logic_string +=
656                             "{{CONVERT_TO_BINARY:2}};\n"
657                     else:
658                         output_logic_string += "{{CONVERT_TO_BINARY:" +
                                str(max_output_state) + "}};\n"
659                     output_logic_string +=
                            "\t\t\t\t\t\tend\n\t\t\t\t\tend"
660         # t = output_logic_string.rsplit(str(max_output_state), 1)
661         # output_logic_string = "2".join(t)
662         output_logic_string =
                convert_embedded_digits_to_binary(output_logic_string,
                max_output_state - 1)
663         if max_output_state <= 2:
664             current_state_size_string = ""
665         else:
666             current_state_size_string = "[" +
                    str(len(bin(max_output_state).replace("0b", "")) - 1) +
                    ":0] "
667
668
669         ### Build strings for generated top module
```

```
670         to_student_wire_list = ""
671         input_module_input_list = ""
672         from_student_wire_list = ""
673         output_module_input_list = ""
674         student_clock_list = ""
675         # format student module
676         student_module_instance = student_module_instance[len("module ")
                + student_module_instance.find("module "):]
677         student_module_instance = re.sub(r"(\s|,|\()input\s", r"\1",
                student_module_instance)
678         student_module_instance = re.sub(r"(\s|,|\()output\s", r"\1",
                student_module_instance)
679         student_module_instance = re.sub(r"(\s|,|\()wire\s", r"\1",
                student_module_instance)
680         student_module_instance = re.sub(r"(\s|,|\()reg\s", r"\1",
                student_module_instance)
681         student_module_instance = re.sub(r"(\s|,|\()\[.*?\]\s", r"\1",
                student_module_instance)
682         student_module_instance += ";"
683         student_module_instance =
                student_module_instance[:student_module_instance.find("(")]
                + " student_module_instance " +
                student_module_instance[student_module_instance.find("("):]
684         for p in port_list:
685             if p.direction == "input" and not p.is_clock:
686                 to_student_wire_list += "\t wire "
687                 if p.width - 1 > 0:
688                     to_student_wire_list += "[" + str(p.width - 1) + ":0]
                        "
689                 to_student_wire_list += "toStudentModule_" + p.name +
                    ";\n"
```

```
690                 input_module_input_list += ", " + "toStudentModule_" +
                        p.name
691                 student_module_instance = re.sub(r"(\s|,|\()" + p.name +
                        r"(\s|,|\))", r"\1" + "toStudentModule_" + p.name +
                        r"\2", student_module_instance)
692             elif p.is_clock and "M9" not in p.ports:
693                 student_clock_list += ", " + "input studentClock_" +
                        p.name
694                 student_module_instance = re.sub(r"(\s|,|\()" + p.name +
                        r"(\s|,|\))", r"\1" + "studentClock_" + p.name +
                        r"\2", student_module_instance)
695             elif p.is_clock:
696                 student_module_instance = re.sub(r"(\s|,|\()" + p.name +
                        r"(\s|,|\))", r"\1" + "clk" + r"\2",
                        student_module_instance)
697             if p.direction == "output" and not p.is_clock:
698                 from_student_wire_list += ", output wire "
699                 if p.width - 1 > 0:
700                     from_student_wire_list += "[" + str(p.width - 1) +
                            ":0] "
701                 from_student_wire_list += "fromStudentModule_" + p.name
702                 output_module_input_list += ", " + "fromStudentModule_" +
                        p.name
703                 student_module_instance = re.sub(r"(\s|,|\()" + p.name +
                        r"(\s|,|\))", r"\1" + "fromStudentModule_" + p.name +
                        r"\2", student_module_instance)
704
705         ##########################################################
706         ############# Build User Inp/Out Manager ################
707         ##########################################################
708         # print(input_list_string)
```

```
709         # print(input_reset_string)
710         # print(input_process_string)
711         # print(output_list_string)
712         # print(output_pin_parameters)
713         # print(output_reg_init_string)
714         # print(output_reset_string)
715         # print(output_logic_string)
716         # print(to_student_wire_list)
717         # print(from_student_wire_list)
718         # print(output_module_input_list)
719         # print(input_module_input_list)
720         # print(student_module_instance)
721         # print(student_clock_list)
722         with open('inc/basefiles/inputManager.v', 'r') as file:
723             data = file.read()
724             data = data.replace('{{INPUT_LIST_HERE}}', input_list_string)
725             data = data.replace('{{RESET_INPUTS_HERE}}',
                    input_reset_string)
726             data = data.replace('{{PROCESSED_INPUTS_HERE}}',
                    input_process_string)
727
728         with open(build_folder + '/src/inputManager.v', 'w') as
                filetowrite:
729             filetowrite.write(data)
730
731
732         with open('inc/basefiles/outputManager.v', 'r') as file:
733             data = file.read().replace('{{MAX_STATE}}',
                    str(max_output_state - 1))
734             data = convert_embedded_digits_to_binary(data,
                    max_output_state - 1)
```

```
735             data = data.replace('{{BOARD_DATA_HERE}}', board_to_use)
736             data = data.replace('{{CONFIG_DATA_HERE}}',
                    configuration_variable)
737             data = data.replace('{{OUTPUT_LIST_HERE}}',
                    output_list_string)
738             data = data.replace('{{OUTPUT_PARAMETERS_HERE}}',
                    output_pin_parameters)
739             data = data.replace('{{OUTPUT_CACHE_HERE}}',
                    output_reg_init_string)
740             data = data.replace('{{RESET_CACHE_HERE}}',
                    output_reset_string)
741             data = data.replace('{{STATE_LOGIC}}', output_logic_string)
742             data = data.replace('{{CURRENT_STATE_SIZE}}',
                    current_state_size_string)
743             data = data.replace('{{PSID_HERE}}',
                    binary_for_verilog(people_soft_id, 24))
744
745         with open(build_folder + '/src/outputManager.v', 'w') as
                filetowrite:
746             filetowrite.write(data)
747
748         with open('inc/basefiles/remoteFPGATOP.v', 'r') as file:
749             data = file.read()
750             data = data.replace('{{EXPOSED_WIRES}}',
                    from_student_wire_list)
751             data = data.replace('{{INPUT_TO_STUDENT}}',
                    to_student_wire_list)
752             data = data.replace('{{INPUT_WIRES}}',
                    input_module_input_list)
753             data = data.replace('{{STUDENT_MODULE}}',
                    student_module_instance)
```

```
754             data = data.replace('{{OUTPUT_WIRES}}',
                    output_module_input_list)
755             data = data.replace('{{STUDENT_CLOCKS}}', student_clock_list)
756
757         with open(build_folder + '/src/remoteFPGATOP.v', 'w') as
                filetowrite:
758             filetowrite.write(data)
759
760
761     ##########################################################
762     ################## Modify .qsf file #####################
763     ##########################################################
764         files = glob.glob(build_folder + '/*.qsf')
765         if len(files) > 1:
766             print("More than one .qsf file found! Argghh :(")
767         with open(files[0], 'r') as file:
768             lines = file.readlines()
769             final_data = ""
770             for line in lines:
771                 if line.strip().startswith('set_global_assignment -name
                        TOP_LEVEL_ENTITY'):
772                     line = 'set_global_assignment -name TOP_LEVEL_ENTITY'
                            + ' remoteFPGATOP'
773                 elif line.strip().startswith('set_location_assignment
                        PIN_'):
774                     port = line.strip()[line.strip().rfind(" "):]
775                     p = line.strip().find('set_location_assignment PIN_')
                            + len('set_location_assignment PIN_')
776                     q = line.strip()[p:].find(" ")
777                     pinName = line.strip()[p:q + p]
778                     p = port.rfind("[") + 1
```

```
779                 if p != 0:
780                     q = port.rfind("]")
781                     port = port[:p - 1]
782                 if (pinName == "G12" or pinName == "K16" or pinName
                        == "G15" or pinName == "J17" or pinName == "M9")
                        and (board_name == "DE0-CV"):
783                     print("Warning: Pins required for VFPGA detected
                            in student module. Skipping pin: " + pinName)
784                     continue
785                 if (pinName == "AH26" or pinName == "AG26" or pinName
                        == "AG23" or pinName == "AH23" or pinName == "Y2")
                        and (board_name == "DE0-CV"):
786                     print("Warning: Pins required for VFPGA detected
                            in student module. Skipping pin: " + pinName)
787                     continue
788                 matchedPort = first(x for x in port_list if x.name ==
                        port.strip())
789                 if matchedPort is None:
790                     print("Warning: unknown port match.")
791                 else:
792                     if matchedPort.direction == "input" and not
                            matchedPort.is_clock:
793                         continue #Dont add this line.
794                     elif matchedPort.is_clock:
795                         line = line.replace(port, " studentClock_" +
                                matchedPort.name)
796                     else:
797                         line = line.replace(port, "
                                fromStudentModule_" + matchedPort.name)
798
799             final_data += "\n" + line.strip()
```

```
800
801         final_data += "\nset_global_assignment -name VERILOG_FILE
                src/FIFO.v\n"
802         final_data += "set_global_assignment -name VERILOG_FILE
                src/outputManager.v\n"
803         final_data += "set_global_assignment -name VERILOG_FILE
                src/inputManager.v\n"
804         final_data += "set_global_assignment -name VERILOG_FILE
                src/remoteFPGATOP.v\n"
805         final_data += "set_global_assignment -name VERILOG_FILE
                src/serializer.v\n"
806         final_data += "set_global_assignment -name VERILOG_FILE
                src/deserializer.v\n"
807         #DE0 CV
808         if board_name == 0:
809             final_data += "set_location_assignment PIN_G12 -to RX\n"
810             final_data += "set_location_assignment PIN_K16 -to TX\n"
811             final_data += "set_location_assignment PIN_G15 -to RXError\n"
812             final_data += "set_location_assignment PIN_J17 -to rst\n"
813             final_data += "set_location_assignment PIN_M9 -to clk\n"
814             final_data += "set_global_assignment -name IOBANK_VCCIO 3.3V
                    -section_id 7A\n"
815         #DE2 115
816         if board_name == 1:
817             final_data += "set_location_assignment PIN_AH26 -to RX\n"
818             final_data += "set_location_assignment PIN_AG26 -to TX\n"
819             final_data += "set_location_assignment PIN_AG23 -to
                    RXError\n"
820             final_data += "set_location_assignment PIN_AH23 -to rst\n"
821             final_data += "set_location_assignment PIN_Y2 -to clk\n"
```

```
822             # final_data += "set_global_assignment -name IOBANK_VCCIO
                    3.3V -section_id 1\n"
823
824
825         with open(files[0], 'w') as file:
826             file.write(final_data)
827
828         set_build_output("Build Complete!", program_output_label)
829     except Exception as e:
830         program_output_text[1] = str(e) + "\nBuild not completed."
831         set_build_output(program_output_text[1], program_output_label)
832
833
834
835 if __name__ == '__main__':
836     main()
```

INPUT MANAGER TEMPLATE

```
module inputManager(input [7:0] ___DATA_IN___,
                    input ___DATA_AVAILABLE___,
                    output reg ___READ___,
                    input ___INPUT_MANAGER_CLOCK___,
                    input ___INPUT_MANAGER_RESET___,
                    {{INPUT_LIST_HERE}});
    parameter ___START_READ_STATE___ = 1'b00,
              ___END_READ_STATE___ = 2'b01,
              ___WRITE_STATE___ = 2'b10,
              ___WAIT_STATE___ = 2'b11;
    reg [1:0] ___INPUT_MANAGER_STATE___;
    reg [15:0] ___DATA_TO_PROCESS___;
    reg __DATA_READ_COUNTER__;
    always @(posedge ___INPUT_MANAGER_CLOCK___)
    begin
        if (___INPUT_MANAGER_RESET___ == 1'b0)
        begin
{{RESET_INPUTS_HERE}}
            __DATA_READ_COUNTER__ <= 1'b0;
            ___READ___       <= 1'b0;
            ___DATA_TO_PROCESS___ <= 0;
            ___INPUT_MANAGER_STATE___  <=
            ___START_READ_STATE___;
        end
        else
        begin
            case (___INPUT_MANAGER_STATE___)
                ___START_READ_STATE___:
                begin
```

```
30                      if (___DATA_AVAILABLE___ == 1'b1)
31                      begin
32                          ___READ___ <= 1'b1;
33                          if (__DATA_READ_COUNTER__ == 1'b0)
34                          begin
35                              ___DATA_TO_PROCESS___[15:8] <=
36                              ___DATA_IN___;
37                              __DATA_READ_COUNTER__ = 1'b1;
38                          end
39                          else
40                          begin
41                              ___DATA_TO_PROCESS___[7:0] <=
42                              ___DATA_IN___;
43                              __DATA_READ_COUNTER__ = 1'b0;
44                          end
45                          ___INPUT_MANAGER_STATE___ <=
46                          ___END_READ_STATE___;
47                      end
48                  end
49                  ___END_READ_STATE___:
50                  begin
51                      ___READ___ <= 1'b0;
52                      if (__DATA_READ_COUNTER__ == 1'b1)
53                      begin
54                          ___INPUT_MANAGER_STATE___ <=
55                          ___WAIT_STATE___;
56                      end
57                      else
58                      begin
59                          ___INPUT_MANAGER_STATE___ <=
60                          ___WRITE_STATE___;
```

```
61                      end
62                  end
63                  ___WAIT_STATE___:
64                  begin
65                      ___INPUT_MANAGER_STATE___ <=
66                      ___START_READ_STATE___;
67                  end
68                  ___WRITE_STATE___:
69                  begin
70                      case (___DATA_TO_PROCESS___[15:1])
71  {{PROCESSED_INPUTS_HERE}}
72                      endcase
73                      ___INPUT_MANAGER_STATE___ <=
74                      ___START_READ_STATE___;
75                  end
76              endcase
77          end
78      end
79  endmodule
```

OUTPUT MANAGER TEMPLATE

```
module outputManager(output reg [7:0] dataOut,
                     output reg dataReady,
                     input clk,
                     input rst,
                     {{OUTPUT_LIST_HERE}});

//PSID reg
parameter [23:0] PSID = {{PSID_HERE}};
reg [3:0] ConfigCounter;

//Pin List
{{OUTPUT_PARAMETERS_HERE}}
parameter STATE_RUN = 1'b0, STATE_WAIT = 1'b1;
reg {{CURRENT_STATE_SIZE}}CURRENT_STATE;
reg {{CURRENT_STATE_SIZE}}NEXT_STATE_BUFFER;
reg [1:0] SECOND_HALF_COUNTER;
reg [7:0] SECOND_HALF_OF_SIGNAL;
// Cache List
{{OUTPUT_CACHE_HERE}}


always @(posedge clk)
begin : CATCH_CHANGED_VALUES
    begin
        if (rst == 1'b0)
        begin
        {{RESET_CACHE_HERE}}
            dataReady <= 1'b0;
            dataOut   <= 8'b00000000;
```

```
30              CURRENT_STATE <= 0;
31              NEXT_STATE_BUFFER <= 0;
32              SECOND_HALF_OF_SIGNAL <= 0;
33              SECOND_HALF_COUNTER <= 0;
34              ConfigCounter <= 4'b0000;
35          end
36          else
37          begin
38              case (CURRENT_STATE)
39              {{CONVERT_TO_BINARY:0}}:
40              begin
41                  case (ConfigCounter)
42                  4'b0000:
43                      begin
44                          dataOut <= {{CONFIG_DATA_HERE}};
45                          dataReady <= 1'b1;
46                          ConfigCounter <= 4'b0001;
47                      end
48                      4'b0001:
49                      begin
50                          dataReady <= 1'b0;
51                          ConfigCounter <= 4'b0010;
52                      end
53                      4'b0010:
54                      begin
55                          dataOut <= {{BOARD_DATA_HERE}};
56                          dataReady <= 1'b1;
57                          ConfigCounter <= 4'b0100;
58                      end
59                      // Skipped a case whoops
60                      4'b0100:
```

```
61                          begin
62                              dataReady <= 1'b0;
63                              ConfigCounter <= 4'b0101;
64                          end
65                          4'b0101:
66                          begin
67                              dataOut <= PSID[23:16];
68                              dataReady <= 1'b1;
69                              ConfigCounter <= 4'b0110;
70                          end
71                          4'b0110:
72                          begin
73                              dataReady <= 1'b0;
74                              ConfigCounter <= 4'b0111;
75                          end
76                          4'b0111:
77                          begin
78                              dataOut <= PSID[15:8];
79                              dataReady <= 1'b1;
80                              ConfigCounter <= 4'b1000;
81                          end
82                          4'b1000:
83                          begin
84                              dataReady <= 1'b0;
85                              ConfigCounter <= 4'b1001;
86                          end
87                          4'b1001:
88                          begin
89                              dataOut <= PSID[7:0];
90                              dataReady <= 1'b1;
91                              ConfigCounter <= 4'b1010;
```

```
92                      end
93                      4'b1010:
94                      begin
95                          dataReady <= 1'b0;
96                          ConfigCounter <= 4'b1011;
97                      end
98                      4'b1011:
99                      begin
100                     CURRENT_STATE <= {{CONVERT_TO_BINARY:2}};
101                     end
102                 endcase
103             end
104             // Send second half of data
105             {{CONVERT_TO_BINARY:1}}:
106             begin
107             case (SECOND_HALF_COUNTER)
108             2'b00:
109             begin
110                 dataReady <= 1'b0;
111                 SECOND_HALF_COUNTER <= 2'b01;
112             end
113                 2'b01:
114             begin
115                 dataReady <= 1'b1;
116                 dataOut <= SECOND_HALF_OF_SIGNAL;
117                 SECOND_HALF_COUNTER <= 2'b10;
118             end
119                 2'b10:
120             begin
121             dataReady <= 1'b0;
122             if (dataReady == 1'b1)
```

```
123                 begin
124                     CURRENT_STATE <= NEXT_STATE_BUFFER;
125                 end
126                 else
127                 begin
128                 if (NEXT_STATE_BUFFER ==
129                 {CONVERT_TO_BINARY:{{MAX_STATE}}}})
130                 begin
131                     CURRENT_STATE <= {{CONVERT_TO_BINARY:2}};
132                 end
133                 else
134                 begin
135                     CURRENT_STATE <= NEXT_STATE_BUFFER + 1'b1;
136                 end
137                 end
138                 SECOND_HALF_COUNTER <= 0;
139                 end
140                 endcase
141                 end
142                 {{STATE_LOGIC}}
143                 endcase
144             end
145         end
146     end
147 endmodule
```

REMOTE TOP MODULE TEMPLATE

```
module remoteFPGATOP(
                input RX,
                output wire RXError,
                output wire incomingMessage,
                output wire TX,
                output wire transmitting,
                input clk,
                input rst{{EXPOSED_WIRES}}{{STUDENT_CLOCKS}}
                );

    // Wires between deserializer and FIFO_In
    wire [7:0] parallelDataIn;
    wire deserialDataReady;

    // Wires between FIFO_In and Input manager
    wire FIFOInHasData;
    wire [7:0] FIFOInDataOut;
    wire inputManagerReadData;

    // Generated Wires between Input Manager and Student Module
{{INPUT_TO_STUDENT}}

    // Wires between output manager and FIFO_Out
    wire [7:0] FIFOOutDataIn;
    wire FIFOOutLoadData;

    // Wires between FIFO_out and Serializer
    wire FIFOOutHasData, serializerReadData;
    wire [7:0] FIFOOutDataOut;
```

```
30
31      deserializer Deserializer(RX,
32                                deserialDataReady,
33                                incomingMessage,
34                                parallelDataIn,
35                                RXError,
36                                clk,
37                                rst);
38
39      FIFO FIFOToStudent (parallelDataIn,
40                          deserialDataReady,
41                          inputManagerReadData,
42                          FIFOInHasData,
43                          FIFOInDataOut,
44                          clk,
45                          rst);
46
47      inputManager InputManager (FIFOInDataOut,
48                                 FIFOInHasData,
49                                 inputManagerReadData,
50                                 clk,
51                                 rst{{INPUT_WIRES}});
52
53      {{STUDENT_MODULE}}
54
55      outputManager OutputManager (FIFOOutDataIn,
56                                   FIFOOutLoadData,
57                                   clk,
58                                   rst{{OUTPUT_WIRES}});
59
60      FIFO #(.WIDTH(8), .DEPTH(64))
```

```
61      FIFOFromStudent(FIFOOutDataIn,
62                      FIFOOutLoadData,
63                      serializerReadData,
64                      FIFOOutHasData,
65                      FIFOOutDataOut,
66                      clk,
67                      rst);
68
69      serializer Serializer (FIFOOutDataOut,
70                             FIFOOutHasData,
71                             TX,
72                             serializerReadData,
73                             transmitting,
74                             clk,
75                             rst);
76
77
78  endmodule
```