Over the Air Updates for an IoT Security Device

By:

Sohail Iqbal

Gerson Roger Dalton Cardozo

Ryan Thomas

12/03/18

Table of Contents

1) Executive Summary

IoT devices are becoming increasingly popular for commercial use due to the ever-increasing size of the smart home industry. With the proliferation of more complex smart devices throughout the households; the question of security becomes an issue. If a security flaw in the smart home network device is discovered and the functionality of the device could be compromised, the need arises for a simple way for users to push software security updates while protecting the integrity of the software on the device. As of now if a user to needed update the software of one of their smart home devices they would have to disconnect their smart home device from the network and connect the device to their PC via a USB cable. Then the user would use the device manufactures software to push said update to device through the USB cable. The user must then reconnect their smart home device to the smart home network.

Therefore, there is a need for a simple, fast, user friendly system that is currently not available in the smart home market. The specifications for our solution is to develop an “Over the Air” software update system for a security IoT device, by writing and using in device software to receive update files and program update files to itself and other devices via a proprietary wireless protocol through simple user input via an application. The primary objective is to develop a security device that can receive updates wirelessly while providing the status of update progress with the secondary objective of detecting motion then taking and sending photos to the user in a cost-effective manner.

The solution developed to address this problem is a microcontroller programmed to interface with a FPGA, motion sensor, camera, and Xbee wireless module. The microcontroller will receive updates from a Raspberry Pi wirelessly via Xbee modules and program updates to itself and update the FPGA through bit banging. The microcontroller when not updating will function as a security camera being triggered by motion detection then sending the photo to the Raspberry Pi wirelessly via Xbee modules.

2) Approach

IoT devices are becoming increasingly popular for commercial use due to the rise of the smart home industry. Home appliances ranging from refrigerators and washing machines to ceiling lights and trash cans; every device in the house are becoming to be connected to the internet. As a result, more, devices are connected to the internet in one house than there were on a entire neighborhood street ten years ago. A point has reached where household appliances have the same amount of hardware complexity as a smartphone. With the rise of this complexity, the question of security becomes an issue. If a security flaw in the smart home network is discovered and the functionality of the devices has been removed or modified. The need arises for a simple way for users to push software security updates while protecting the integrity of the software on the device.

- Problem Statement

An Over the Air (OTA) Update system is needed to provide security patches to the wireless sensor devices that can be easily integrated into current smart home networks. The system would need to be cost effective, use existing smart home technology, and use a low amount of power for long use durations. The solution to this problem was to develop a software implementation for packaging, sending, and programing binary files while using existing hardware found in smart home networks.

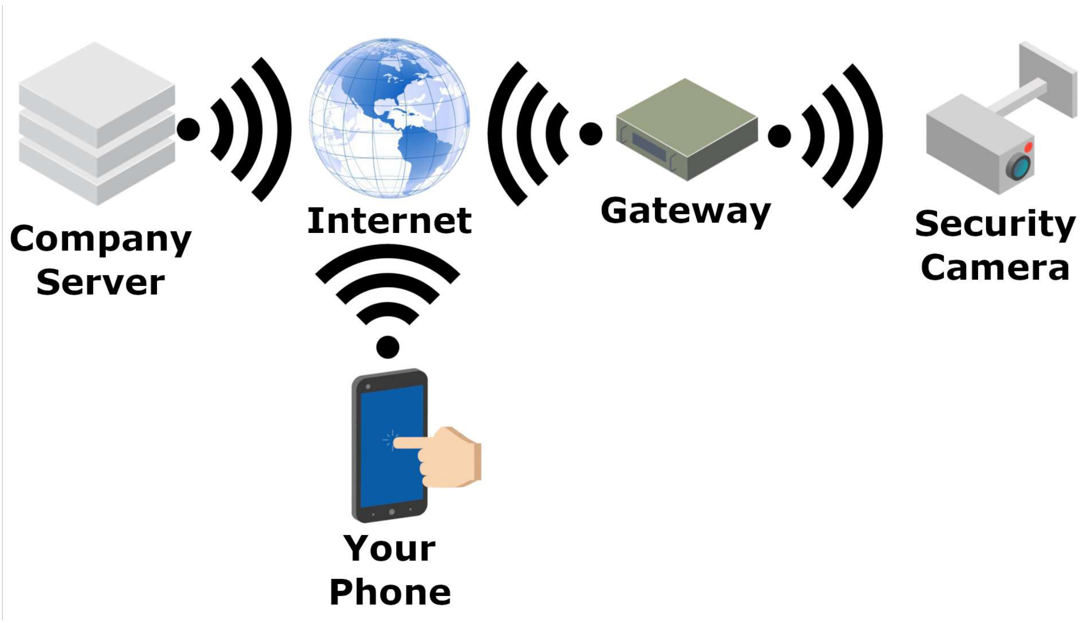


Figure 1.0: User level explanation of what the overall system should look like.

Figure 1.0

- Requirements
  - Secure Communication
    - Data sent to the node is sent through an exclusive private channel.
  - Long battery life
    - Low power device selection and efficient programming methods.
  - Reprogrammable Wirelessly
    - Microcontroller and FPGA can be updated from our gateway without wires
  - Secure Update Delivery
    - Node software Updates must have a verification system
  - Provide Update progress
    - User must be notified of software update progress

- Solutions and Alternatives

The user would have an application installed on their phone that would communicate to a company server in order to communicate with the gateway to check the software version of nodes on a smart home network. The Server would compare version numbers and push a software update if the version hosted on the server is newer than version located on the nodes of the smart home network. Software updates pushed by the server would be received by the gateway then unpacked, parsed and repackaged in a manner than is compatible for a wireless node on the network. In the event that the update becomes corrupt or there is packet loss on the network, the update process must restart from the beginning or resume progress depending on what is causing information loss. On the device end, software must be written for a microcontroller to be able to receive updates even upon losing connection, program itself using its bootloader program and program any other devices connected to it such as a field programmable gate array while being power efficient and maintaining the core functionality of the device.

An alternative costlier solution to this problem is to connect a wifi enabled module to the existing wireless smart home device and program the microcontroller to directly pull software updates from dropbox or github using the protocols provided in the wifi module. All packet loss is handled via the built in protocols of the wifi module. Additional software would need to be written for the module to handle any other data outside the microcontroller's software update file. An app to push software updates is provided through the manufacturer of the wifi module and microcontrollers. This version of the system is costly and not easily integrable into existing smart home networks but provides the utmost security and reliability when needed.

- Contribution of each Team Member

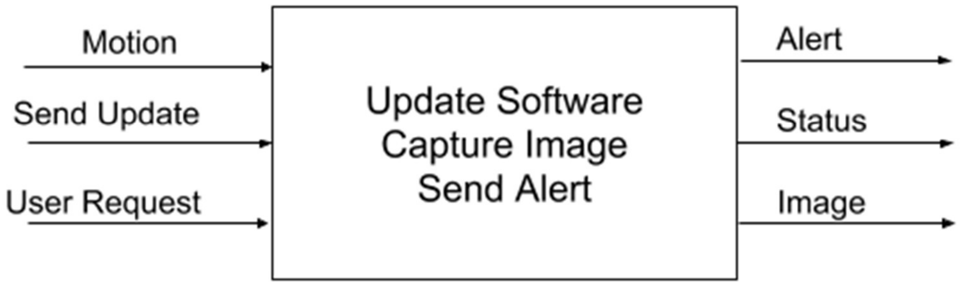
Ryan Thomas is responsible for leading the project by providing schedules, meeting and deadlines that need to meet throughout the course of the semester and following up with team members to make sure that they meet their deadlines. He also provided contingency plans in case one aspect of the project did not work. He also researched and prototyped the feasibility of updating the MSP432 via the bootloader. Ryan developed python code to parse, package send software updates from a Raspberry Pi to MSP432 microcontroller via the UART data protocol. Additionally, he developed code to integrate the Xbee module, PIR motion sensor and camera to MSP432 microcontroller. Ryan also configured hardware and did hardware testing for the PIR, camera, and Xbee modules. He provided assistance in designing the circuit for PCB as it integrates all components. Ryan finally ported FPGA programming source code MSP432.

Sohail Iqbal completed all the required research about the programming of FPGA. He optimized the elliptic curve security algorithm first in Libero SOC environment. Inputs and outputs are assigned using a separate tool. (Multiview navigator). He also generated the required files to program the FPGA. He also optimized the nonfunctional DirectC code first in IAR environment and then in code composer to be successfully built and compiled. Additionally, he is responsible for keeping up with the required group documents such as in progress report, in progress presentation and final draft and report.

Gerson Dalton oversees the PCB layout and the wireless communication. He researched and analyzed the different tools to build the PCB and chose the appropriate modules and protocols for the wireless communication depending on the requirements specified. He created the schematics and layout boards in KiCad. He collaborated with Sohail and Ryan to keep up with the changes inflicted to the schematics throughout the progress of configuring the circuit connection between the different components of the device. He had to analyze the different datasheets and footprints of the components to build the circuit considering the current limitations. He collaborated with Sohail in the creation of the final slides and report of the project.

3) Technical Section

- Top Level Views



Level 0 of the overall system of inputs and outputs are displayed in Figure 1.1

Figure 1.1

Level 1 view of the update process of the system are displayed in Figure 1.2

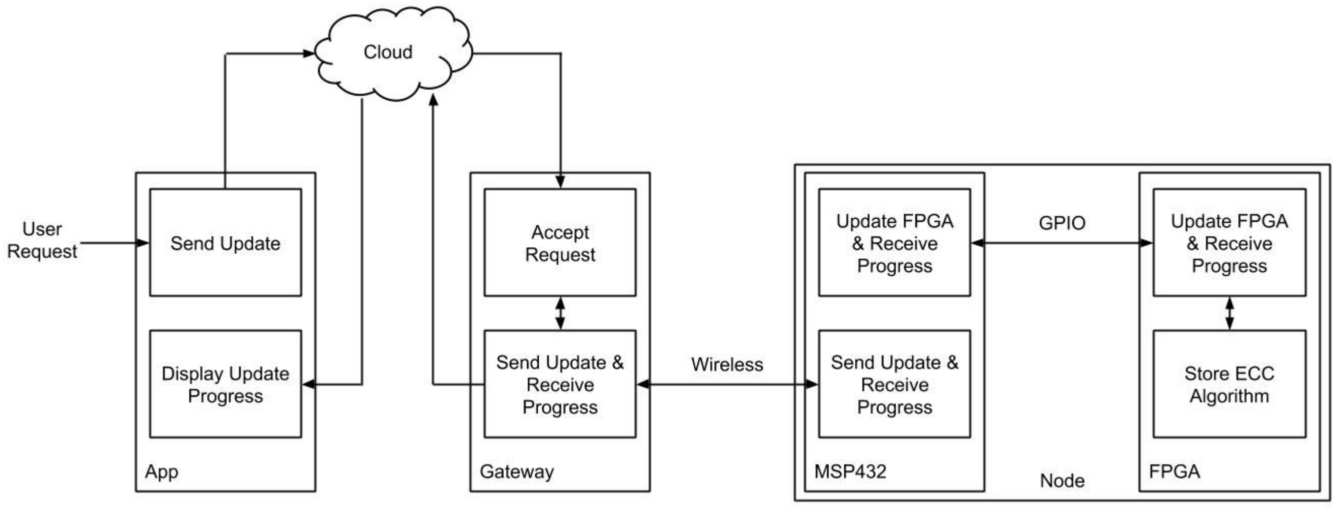

Cloud
User Request
Send Update
Display Update Progress
App
Accept Request
Send Update & Receive Progress
Gateway
Wireless
Update FPGA & Receive Progress
Send Update & Receive Progress
MSP432
GPIO
Update FPGA & Receive Progress
Store ECC Algorithm
FPGA
Node

Figure 1.2

Level 1 view of the photo taking process of the system are displayed in Figure 1.3

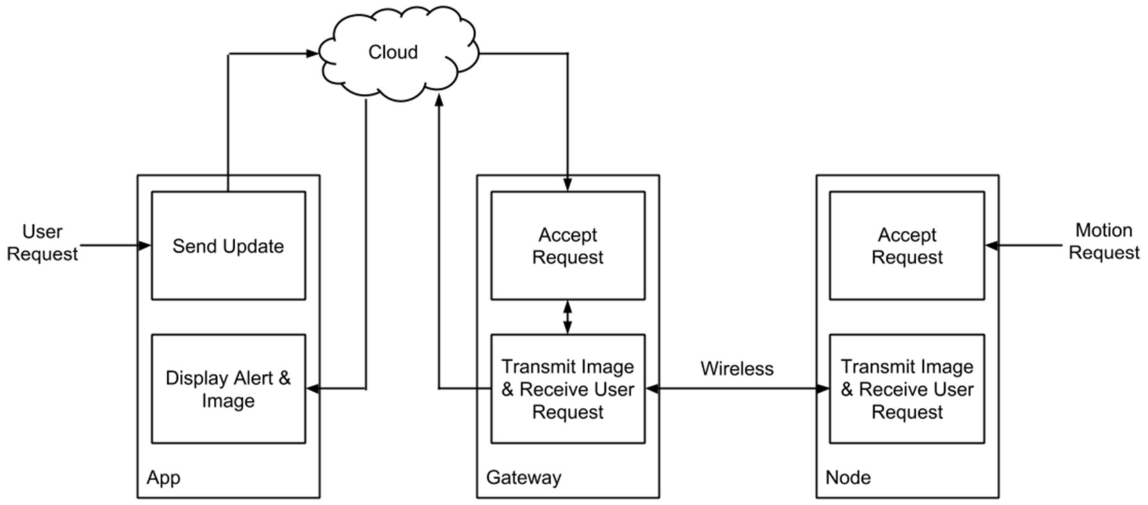


Figure 1.3

Level 1 view of the system architecture is displayed in Figure 1.4

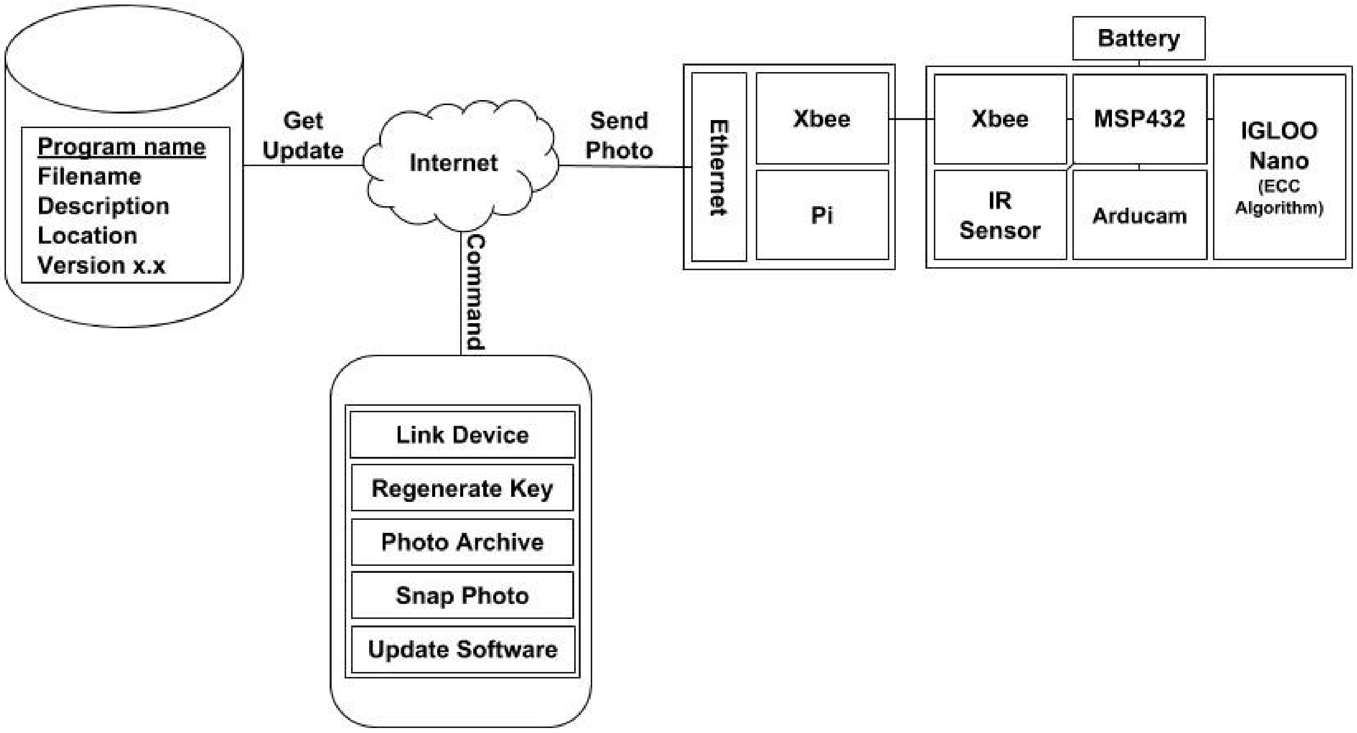


Figure 1.4

- MSP432 to FPGA

Level 2 view of JTAG programing connections between microcontroller and FPGA that are needed for the MSP432 to update IGLOO nano security algorithms in Figure 1.5

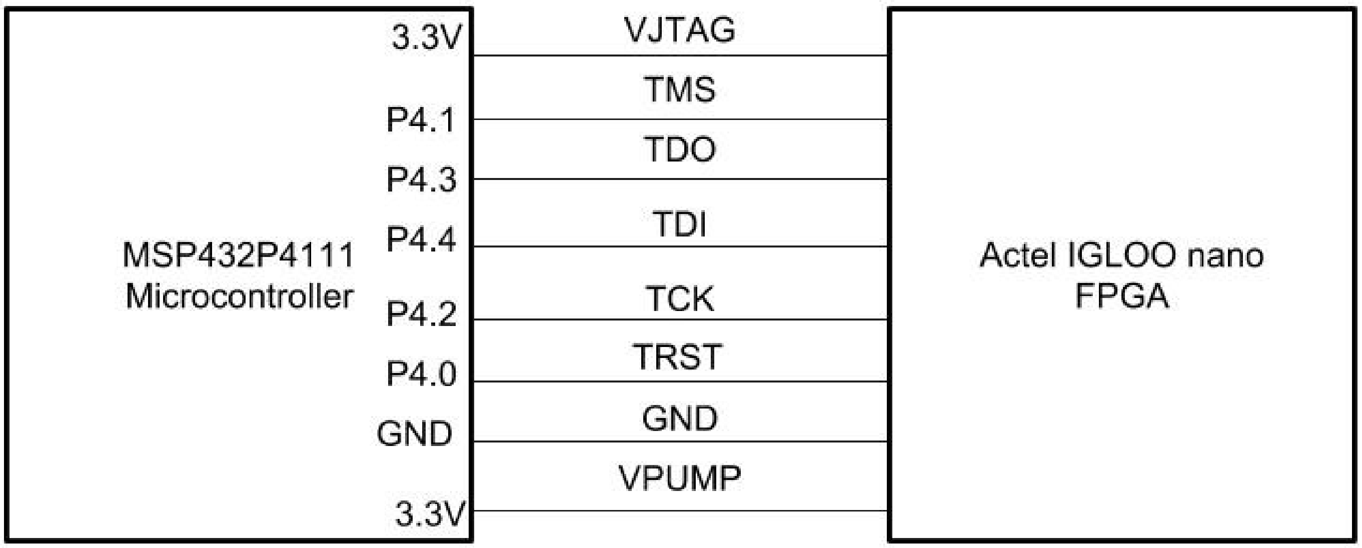


Figure 1.5

- MSP432 to Arducam

Level 2 view of connections between the microcontroller and Arducam that are needed for the MSP432 to configure the camera via I2C and take pictures via SPI in Figure 1.6

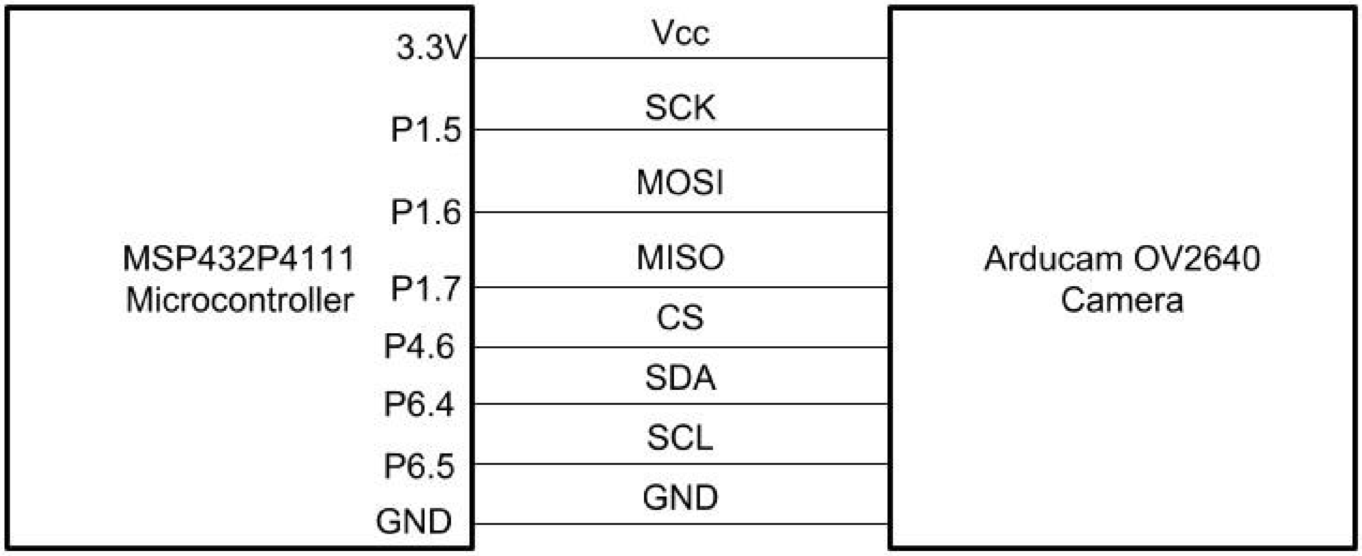


Figure 1.6

- MSP432 to PIR Motion Sensor

Level 2 view of connections between the microcontroller and PIR motion sensor that are needed for the MSP432 to take detect motion via GPIO in Figure 1.7

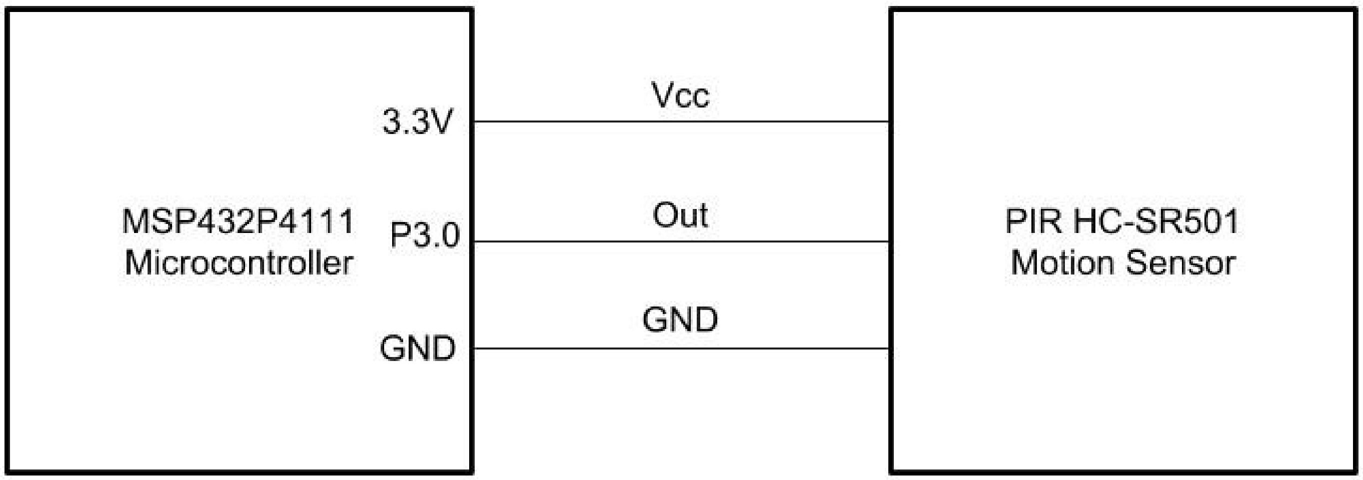


Figure 1.7

- MSP432 to Xbee S2C

Level 2 view of connections between the microcontroller and Xbee S2C that are needed for the MSP432 to communicate with the gateway via UART in Figure 1.8

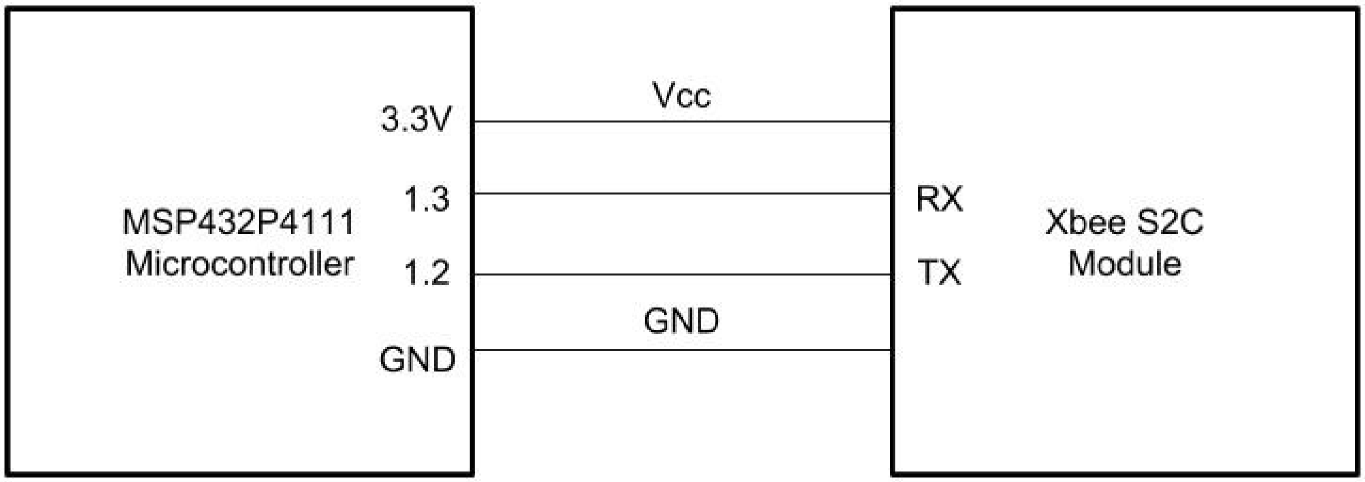


Figure 1.8

- MSP432 to Xbee S2C

Level 2 view of connections between the Raspberry and Xbee S2C that are needed for the Pi to communicate with the node via UART in Figure 1.9

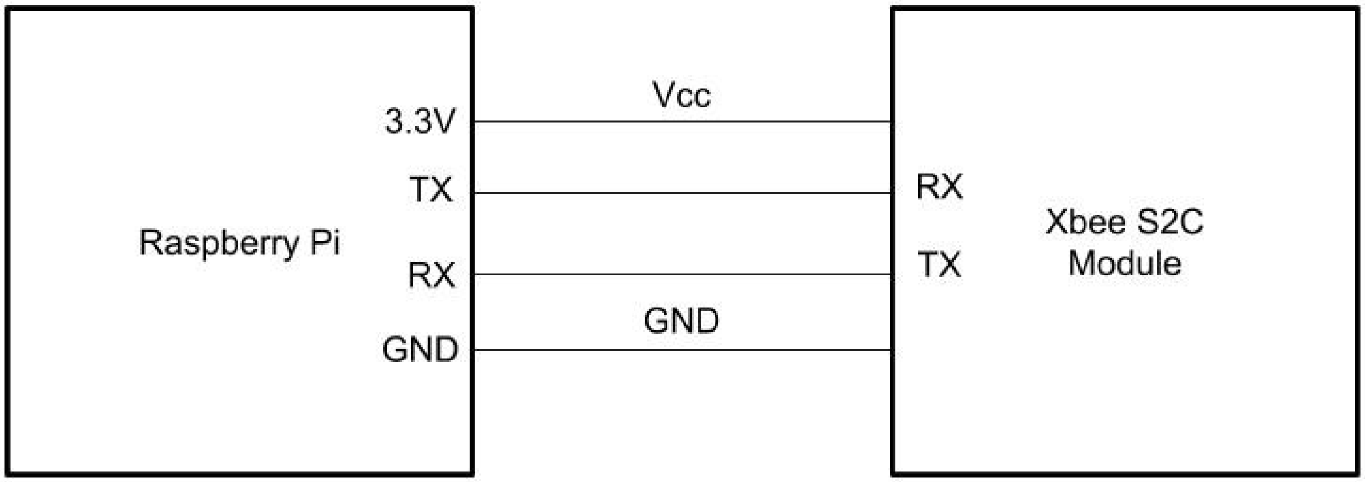


Figure 1.9

- FPGA (IGLOO nano)

The Actel IGLOO nano is one of the low costs, low power FPGAs on the market. The main purpose device is to hold encryption algorithms for encrypting keys of the device. Even after the device turns on and off. It can also be programmed by JTAG through GPIO connections on a microcontroller. A picture of the device is in Figure 2.0

Figure 2.0

- Microcontroller (MSP-432P4111)

The MSP432P411 is the newest MSP432 on the market with 2MB of flash memory and 256 KB of RAM. It is one of low cost, low power microcontrollers on the market that can support what the need of holding an entire FPGA Update which is 243 KB. It is one of the primary components of the system and is used as the central hub for receiving updates, detecting motion, sending photos and programming the FPGA.

The MSP432P4111 is pictured in Figure 2.1

Figure 2.1

- Camera (Arducam OV2640)

The Arducam OV2640 is a 2-megapixel camera that supports high power and lower power modes with configuration via I2C and data transfer via SPI is also supports jpeg compression to send photos efficiently.

The Arducam OV2640 is pictured in Figure 2.2

Figure 2.2

- Motion Sensor (PIR HC-SR501)

The HC-SR501 is the sensor we chose to detect motion as it commonly used in house security systems. Upon an intruder walking within 21 feet in front of the sensor the sensor will output a high signal to the MSP432. The Picture in Figure 2.3 is the PIR HC-SR501 Motion Sensor

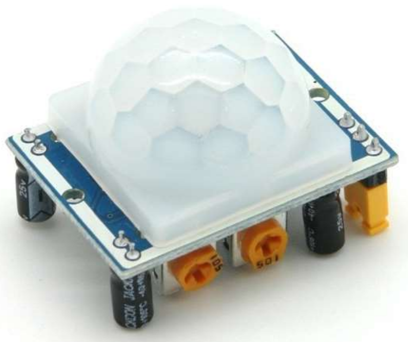


Figure 2.3

- Wireless Communication (Xbee S2C)

The component that we chose for this wireless communication is the Xbee S2C module. It's ideal for the purpose of our project because it's commonly used in the smart home industry by companies like NEST. This makes it easily integrable into existing smart home networks. The picture on Figure 2.4 is the Xbee S2C module

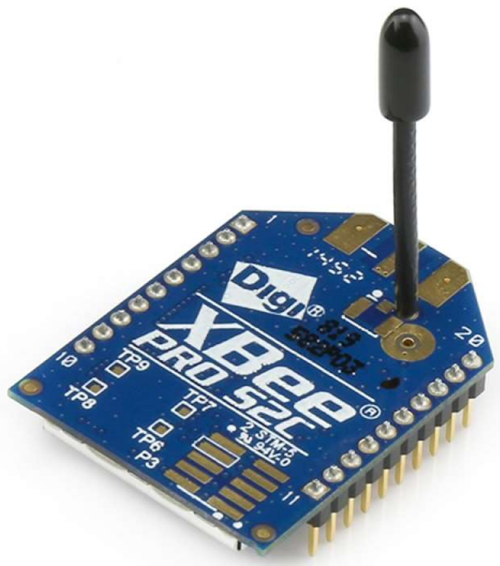


Figure 2.4

- Gateway (Raspberry Pi 3)

The component that we chose for the gateway was the Raspberry Pi 3. It is a low cost, low power 4 core ARM computer. It has excellent software support and has TX and RX pins to interface with the Xbee easily to communicate to the node support for python allows easy communication to servers and phones via APIs.

The Raspberry Pi 3 is pictured in Figure 2.5

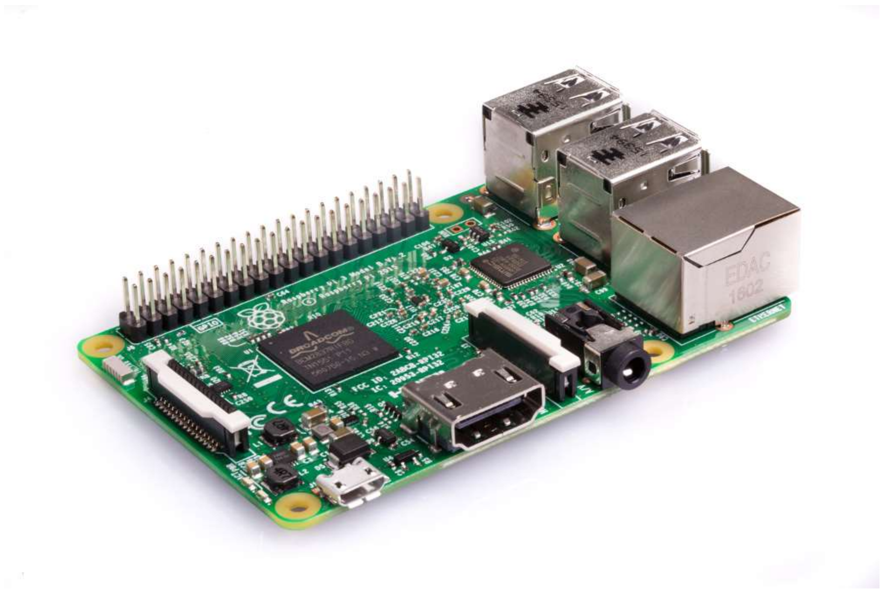
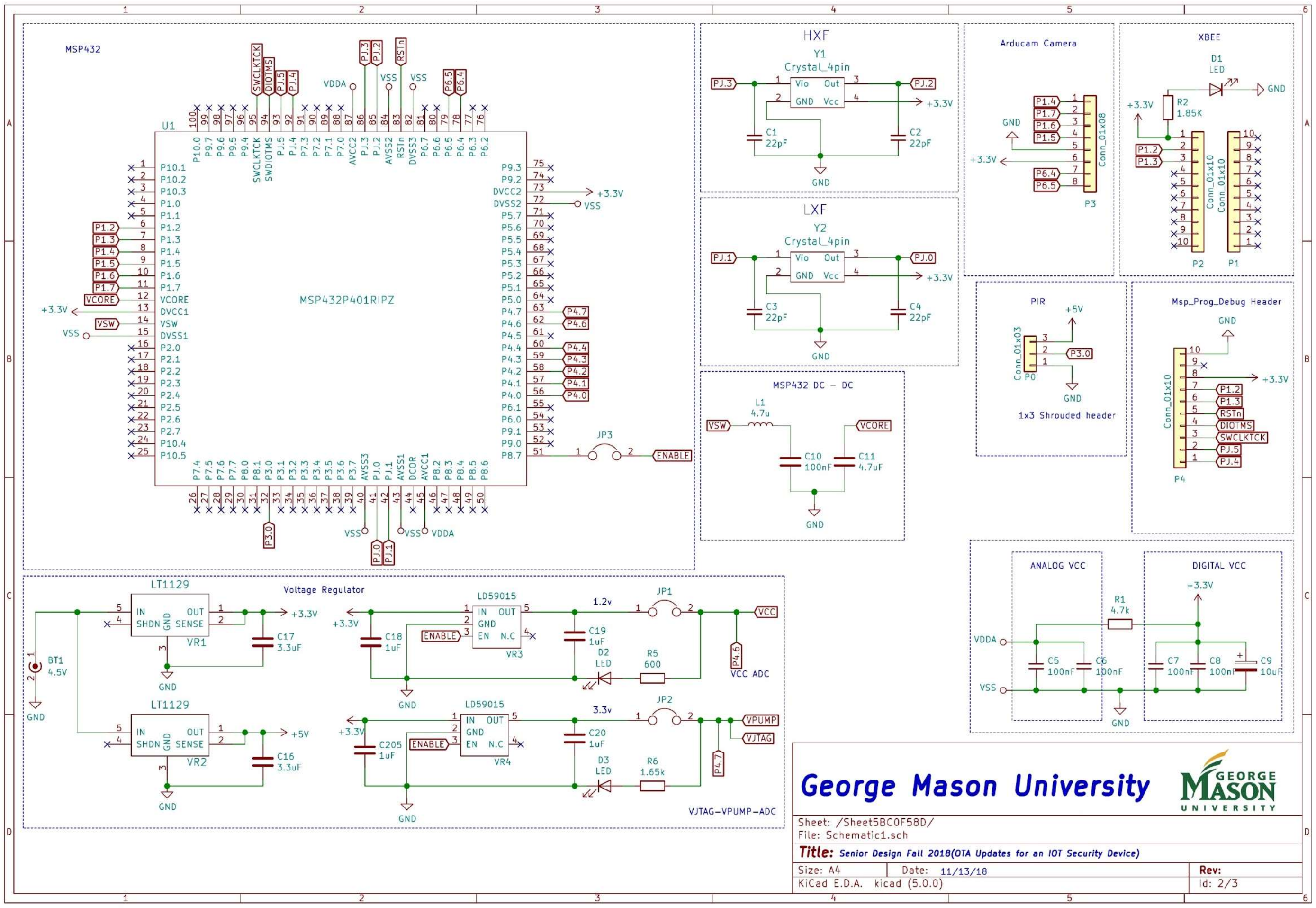


Figure 2.5

- PCB Design
    - Schematics



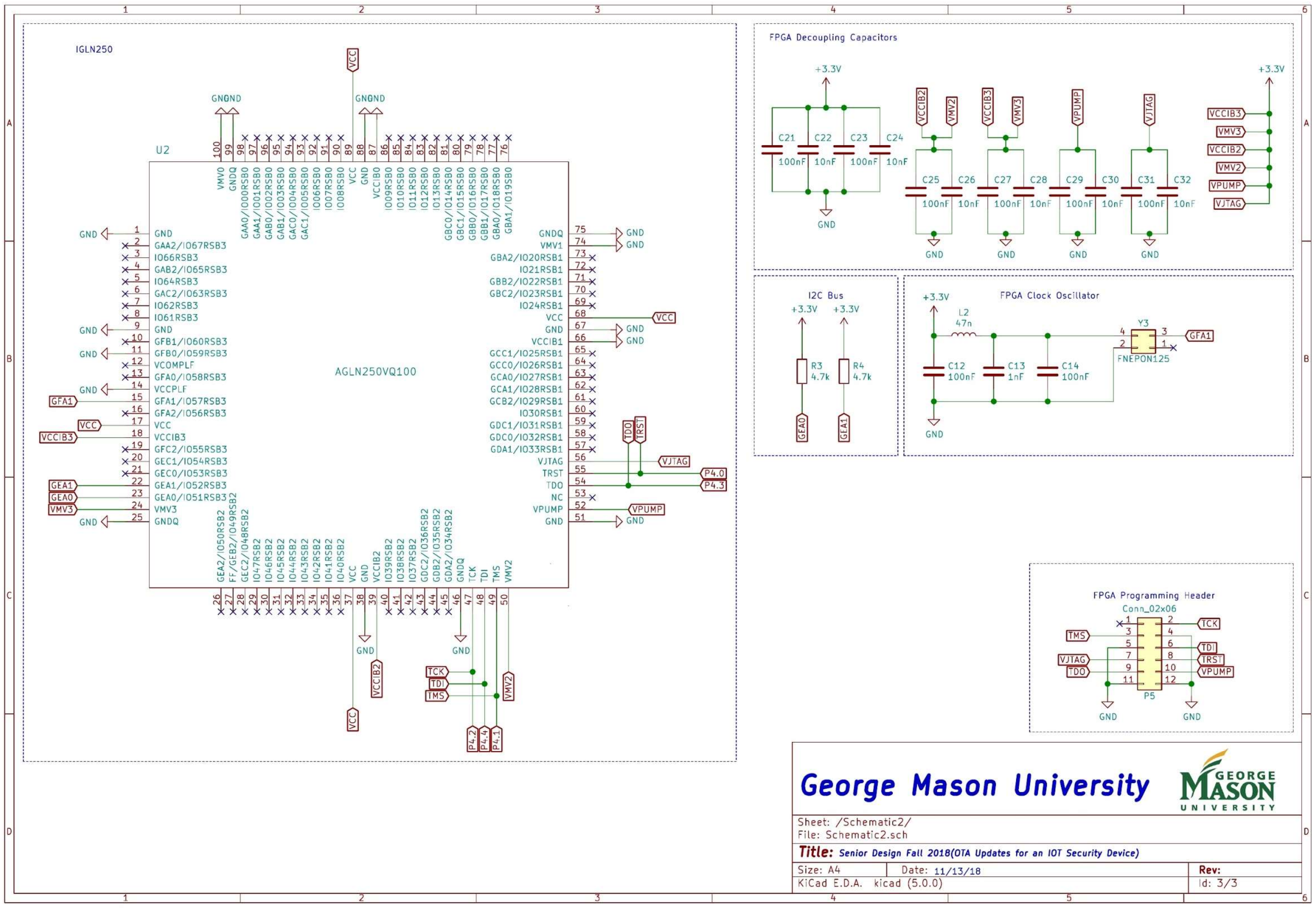


Figure 3.0: Front PCB Layout

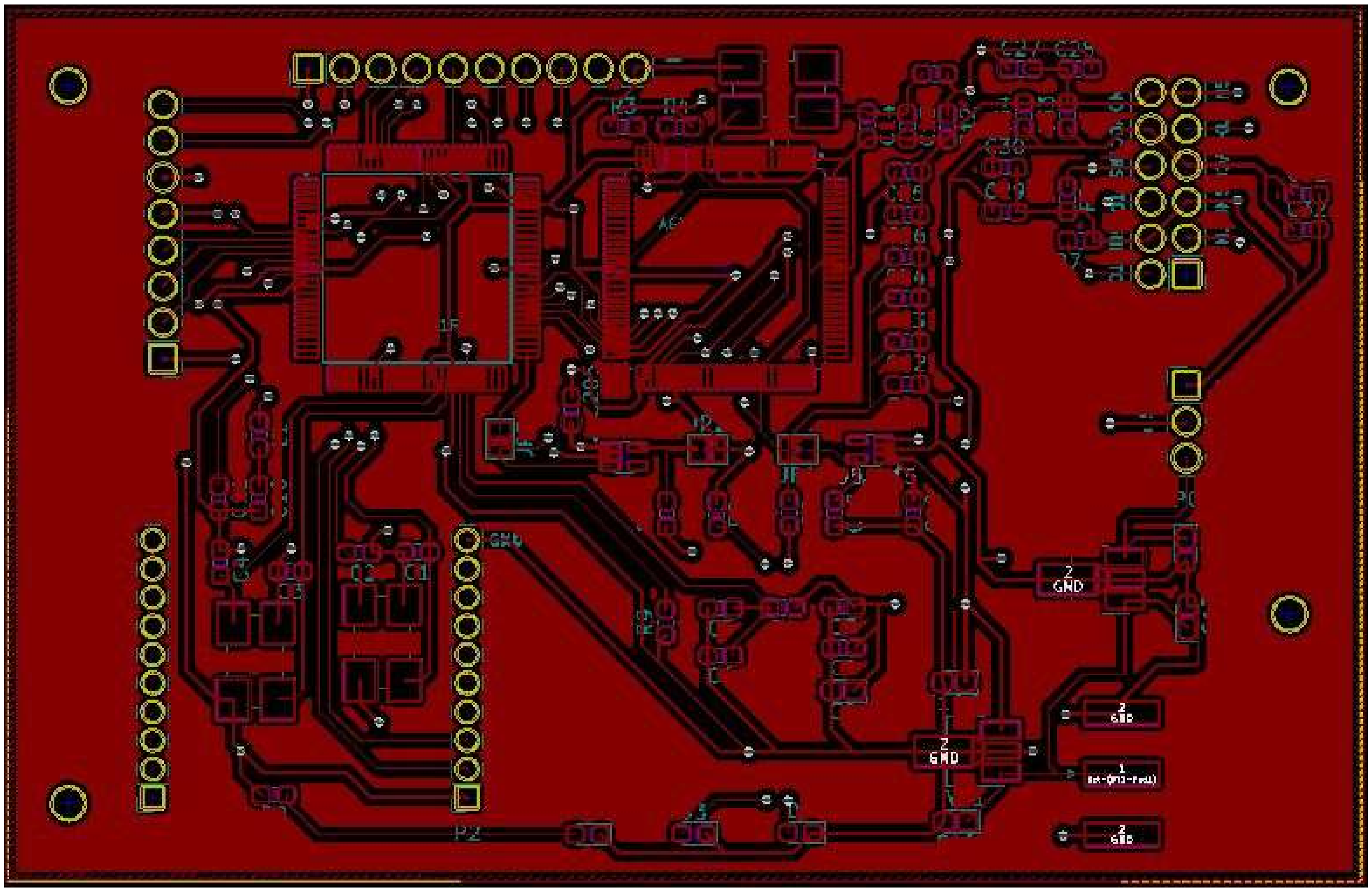

GND
GND
GND
GND
GND
P2
C1
C2
C3

Figure 3.1:Back PCB Layout

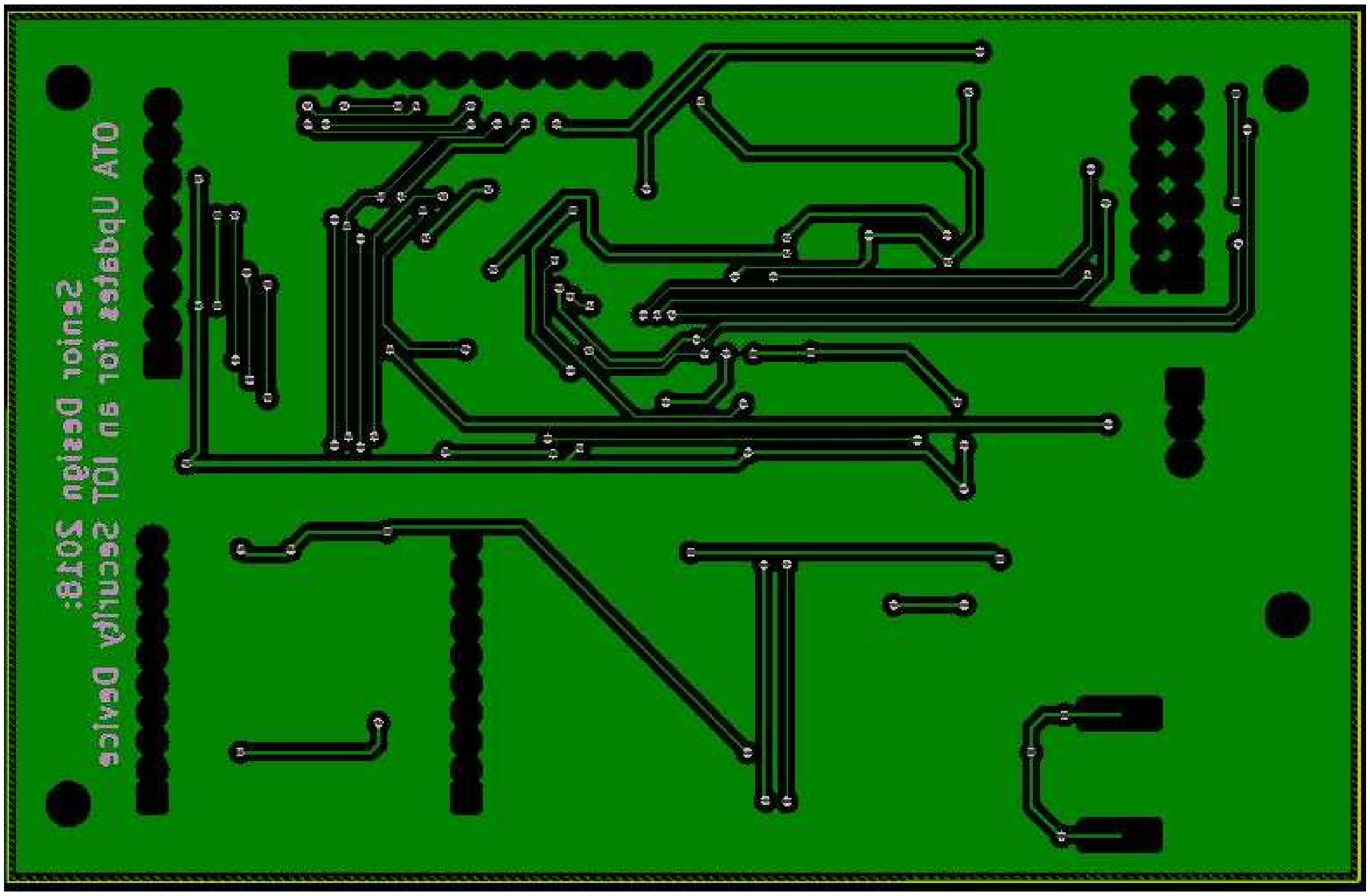


Figure 3.2 and 3.3: 3D Generated Model of PCB

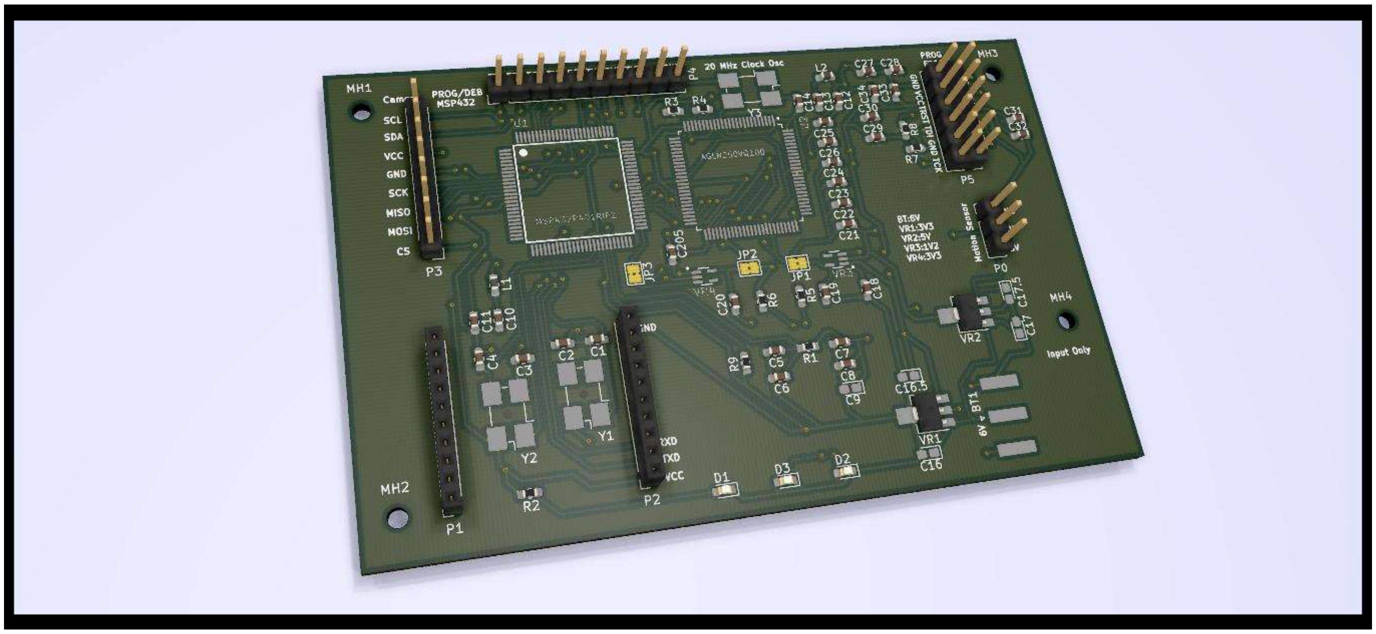
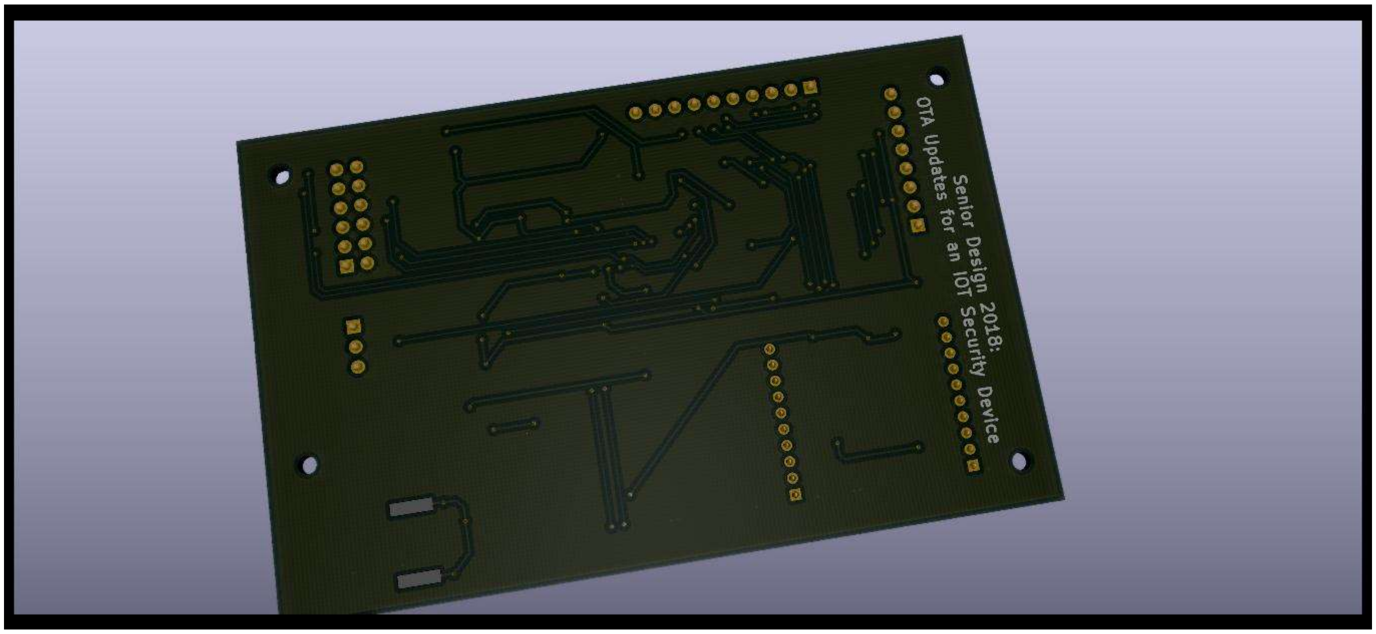
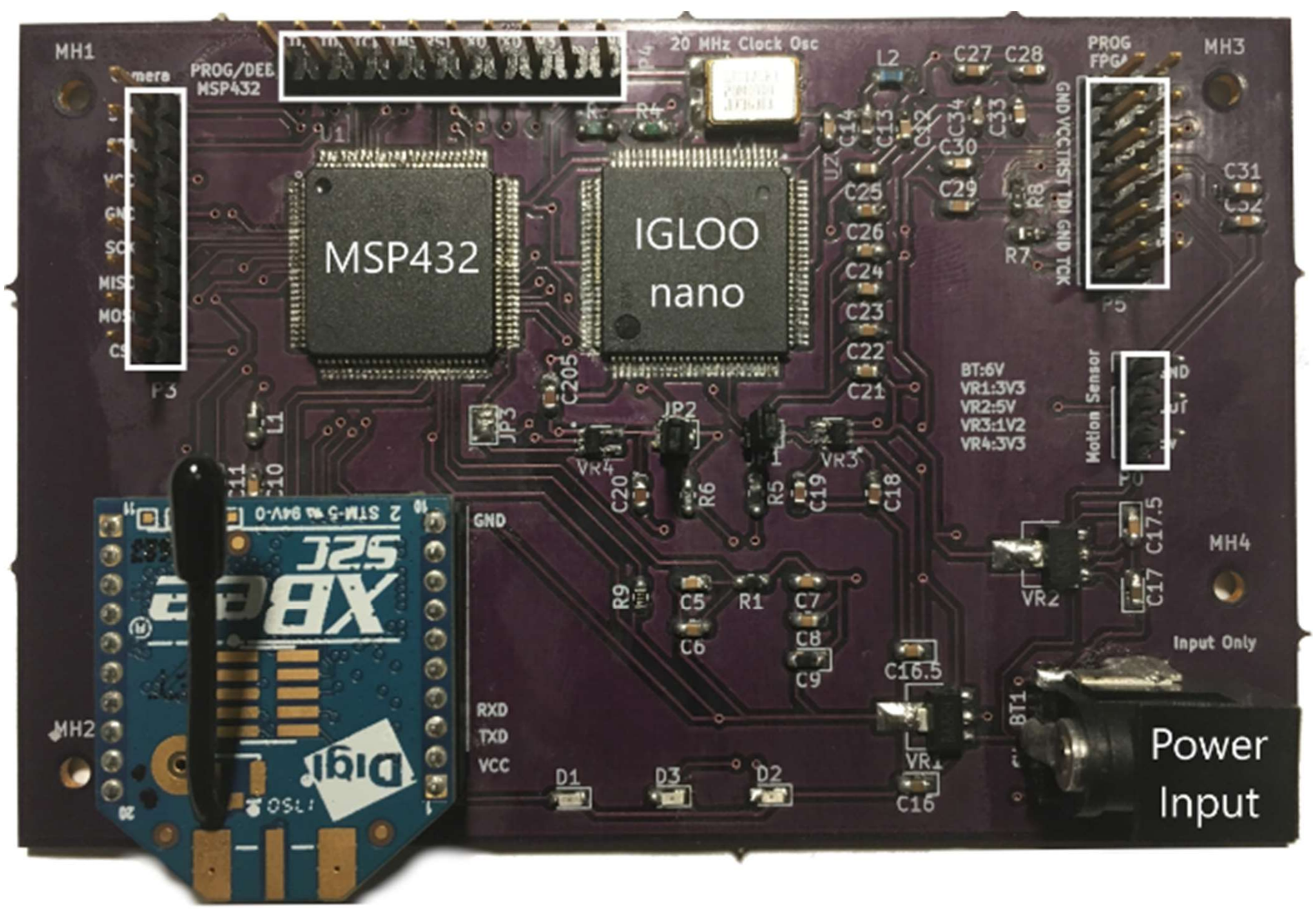



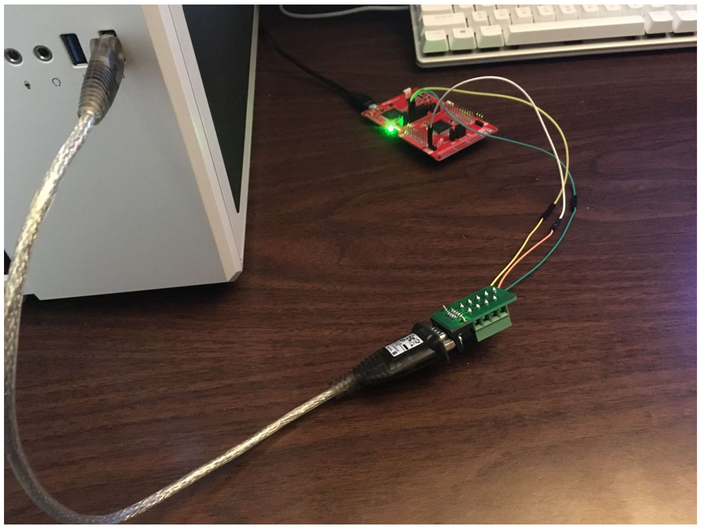
Figure 3.4: Actual Printed PCB



4) Experimentation

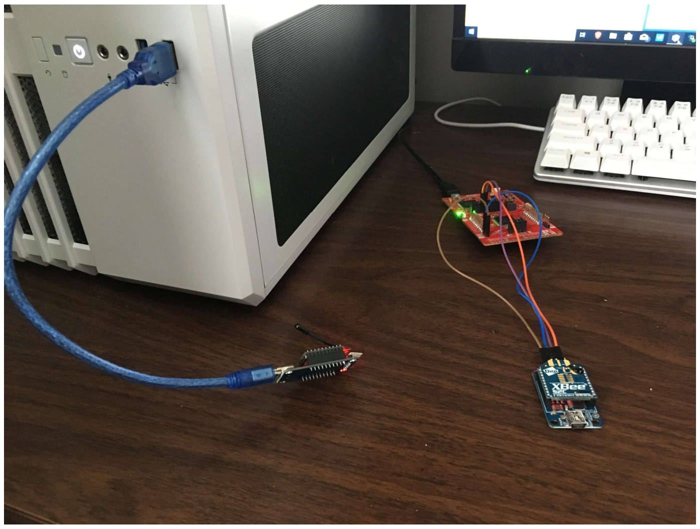
Much of the experimentation is applied to hardware interfacing. At first, we used in house software to communicate through hardware. Each piece of hardware had to be manually configured in order to interact with another piece of hardware within the system.

For the MSP432 we used their BSL scripter to test sending bootloader command to the MSP432



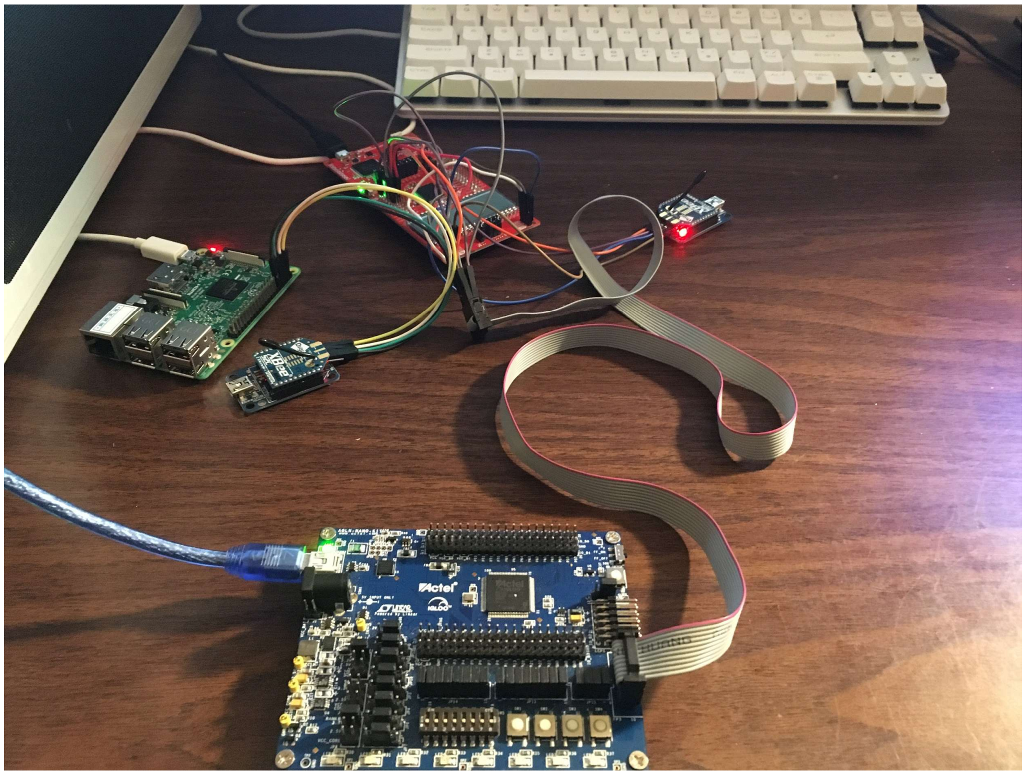
before even attempting to write software that unpacked updates and sent them to the MSP432.

Upon successfully sending update packages to the MSP432 we worked configuring the Xbee modules to send data wirelessly on the its own exclusive channel to test the system and verify that data was sent uncorrupted. We would stress test the wireless communication by sending large packets of data as the FPGA needs at least 230 KB of data to be sent.

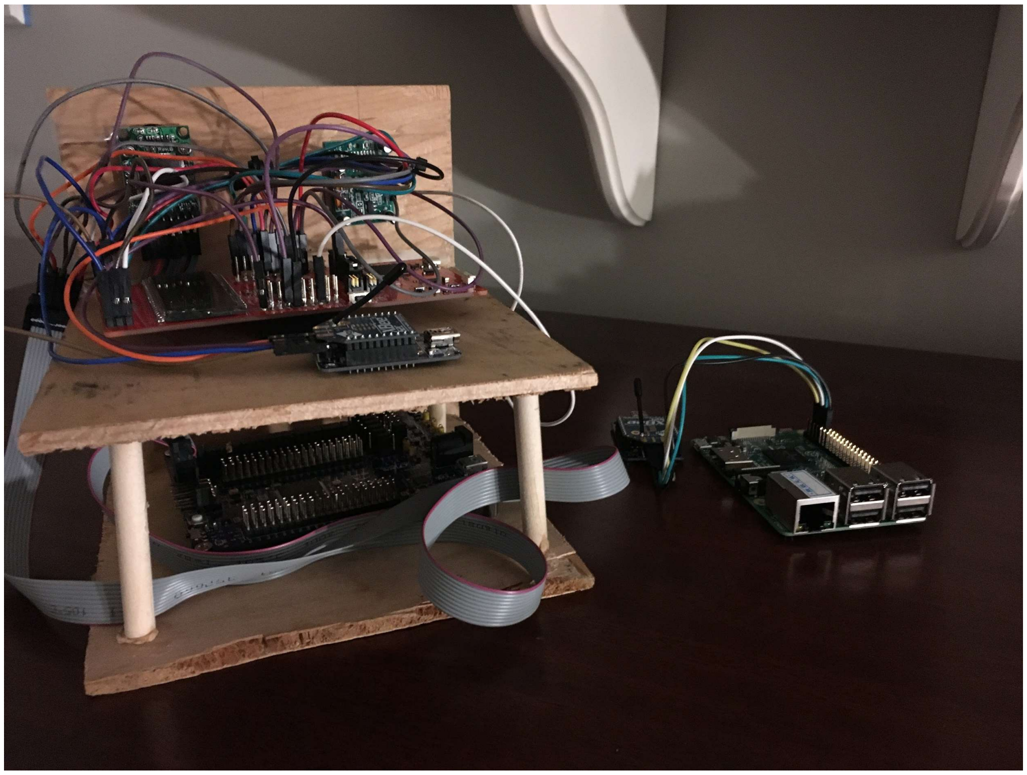


Once the Xbee were done testing we moved onto creating an update package handler in python based on the specification BSL scripter program that the MSP432 manufacturer provided. Then we setup and configured the raspberry pi to the interface with the MSP432 via Xbee with the python update program.

Once updating the MSP432 via raspberry pi was complete. We verified that the FPGA can be programed from the PC through a low cost JTAG programmer for a *.DAT file. We proceeded to start porting the DirectC programming source code to MSP432 in order to give it the capability to program the FPGA like this JTAG programmer



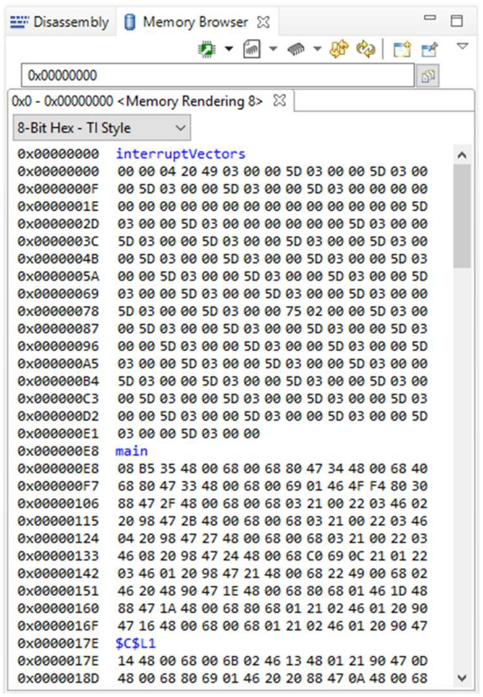
While JTAG programming is still worked on. Configuration and integration of the motion sensor and Arducam were applied to the MSP432 so that it could take and send picture upon the proper triggers. Once all these systems we fully tested in isolation They were all integrated together into one unit where any bugs could be worked out and any system noise could be resolved.



5) Experimental validation using evaluation criteria

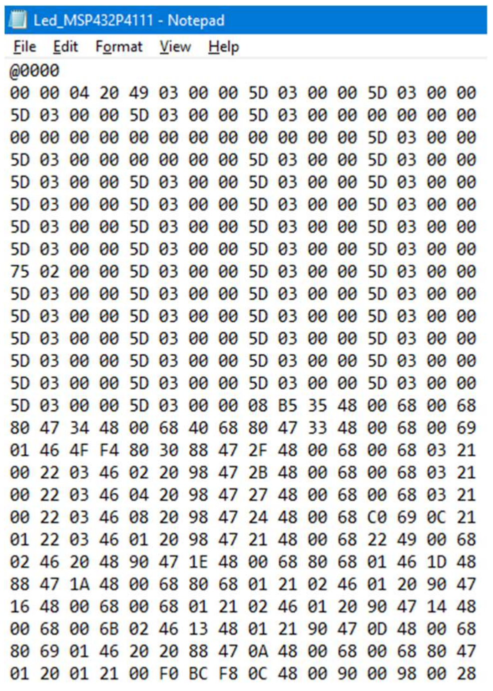
The primary evaluation criteria is based on multiple aspects. The First evaluation is visual hardware feedback. The goal is to develop software that provides easy indications that the program has been flashed to the device. The simplest way to verify this is to have two programs with different functionalities. For the MSP432 one program may have a slow blinking led while the second program would have a fast blinking led. For the FPGA there would be two configurations where the FPGA runs a program that would turn on an LED based on an AND configuration while the other program would turn on an LED based on an OR configuration. The second piece of evaluation is software level data verification. The data being sent from the Raspberry Pi to the MSP432 microcontroller and FPGA needs to verified on a byte to byte basis. That means that all data sent from the Pi must be successfully sent wirelessly and flashed to the MSP432. Using the MSP432's IDE with proper configuration we examine the machine code stored in flash memory and compare it to the original binary file machine code to see if the MSP432 has been successfully updated. The same must be done for the FPGA's binary file. The FPGA programming the source code provides error codes to indicate that the FPGA has been programmed or an error had occurred during programming. In order to verify a successful program on a software level the error code returned on a software level must be zero through examination on the MSP432's IDE in a debug session. The final evaluation was the evaluation of our secondary goals which was to have the MSP432 be able to send photos to the Raspberry Pi upon a motion detection or a user request. When the user walks in front of the node or receives a command the MSP432 takes a photo and sends it to the Pi. The Pi receives the photo and stores the photo into the specified the folder and sends the file to Dropbox and send a notification to the user's phone.

The flash memory of the MSP432 when the blink_led program when programmed via the PC is shown in Figure 5.0 The binary file of blink_led program exported in txt format is in Figure 5.1 The flash memory of the MSP432 when the blink_led is programmed via the gateway is shown in Figure 5.2



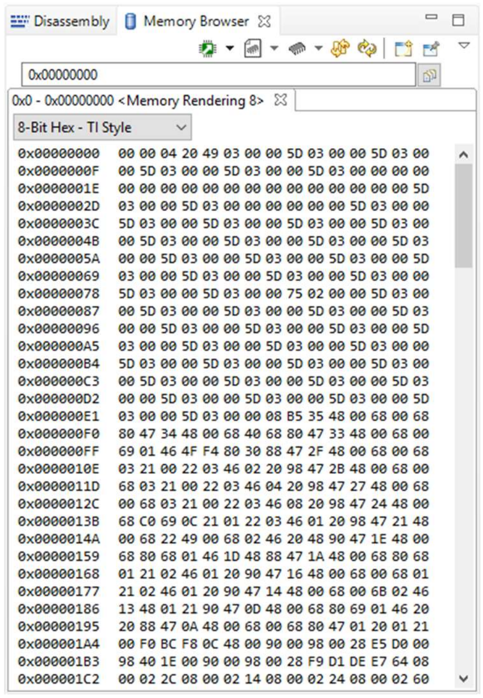
```
0x00000000  interruptVectors
0x00000000  00 00 04 20 49 03 00 00 5D 03 00 00 5D 03 00
0x0000000F  00 5D 03 00 00 5D 03 00 00 5D 03 00 00 00 00
0x0000001E  00 00 00 00 00 00 00 00 00 00 00 00 00 00 5D
0x0000002D  03 00 00 5D 03 00 00 00 00 00 00 5D 03 00 00
0x0000003C  5D 03 00 00 5D 03 00 00 5D 03 00 00 5D 03 00
0x0000004B  00 5D 03 00 00 5D 03 00 00 5D 03 00 00 5D 03
0x0000005A  00 00 5D 03 00 00 5D 03 00 00 5D 03 00 00 5D
0x00000069  03 00 00 5D 03 00 00 5D 03 00 00 5D 03 00 00
0x00000078  5D 03 00 00 5D 03 00 00 75 02 00 00 5D 03 00
0x00000087  00 5D 03 00 00 5D 03 00 00 5D 03 00 00 5D 03
0x00000096  00 00 5D 03 00 00 5D 03 00 00 5D 03 00 00 5D
0x000000A5  03 00 00 5D 03 00 00 5D 03 00 00 5D 03 00 00
0x000000B4  5D 03 00 00 5D 03 00 00 5D 03 00 00 5D 03 00
0x000000C3  00 5D 03 00 00 5D 03 00 00 5D 03 00 00 5D 03
0x000000D2  00 00 5D 03 00 00 5D 03 00 00 5D 03 00 00 5D
0x000000E1  03 00 00 5D 03 00 00
0x000000E8  main
0x000000E8  08 B5 35 48 00 68 00 68 80 47 34 48 00 68 40
0x000000F7  68 80 47 33 48 00 68 00 69 01 46 4F F4 80 30
0x00000106  88 47 2F 48 00 68 00 68 03 21 00 22 03 46 02
0x00000115  20 98 47 2B 48 00 68 00 68 03 21 00 22 03 46
0x00000124  04 20 98 47 27 48 00 68 00 68 03 21 00 22 03
0x00000133  46 08 20 98 47 24 48 00 68 C0 69 0C 21 01 22
0x00000142  03 46 01 20 98 47 21 48 00 68 22 49 00 68 02
0x00000151  46 20 48 90 47 1E 48 00 68 80 68 01 46 1D 48
0x00000160  88 47 1A 48 00 68 80 68 01 21 02 46 01 20 90
0x0000016F  47 16 48 00 68 00 68 01 21 02 46 01 20 90 47
0x0000017E  $C$L1
0x0000017E  14 48 00 68 00 6B 02 46 13 48 01 21 90 47 0D
0x0000018D  48 00 68 80 69 01 46 20 20 88 47 0A 48 00 68
```

Figure 5.0



```
@0000
00 00 04 20 49 03 00 00 5D 03 00 00 5D 03 00 00
5D 03 00 00 5D 03 00 00 5D 03 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 5D 03 00 00
5D 03 00 00 00 00 00 00 5D 03 00 00 5D 03 00 00
5D 03 00 00 5D 03 00 00 5D 03 00 00 5D 03 00 00
5D 03 00 00 5D 03 00 00 5D 03 00 00 5D 03 00 00
5D 03 00 00 5D 03 00 00 5D 03 00 00 5D 03 00 00
5D 03 00 00 5D 03 00 00 5D 03 00 00 5D 03 00 00
75 02 00 00 5D 03 00 00 5D 03 00 00 5D 03 00 00
5D 03 00 00 5D 03 00 00 5D 03 00 00 5D 03 00 00
5D 03 00 00 5D 03 00 00 5D 03 00 00 5D 03 00 00
5D 03 00 00 5D 03 00 00 5D 03 00 00 5D 03 00 00
5D 03 00 00 5D 03 00 00 5D 03 00 00 5D 03 00 00
5D 03 00 00 5D 03 00 00 5D 03 00 00 5D 03 00 00
5D 03 00 00 5D 03 00 00 08 B5 35 48 00 68 00 68
80 47 34 48 00 68 40 68 80 47 33 48 00 68 00 69
01 46 4F F4 80 30 88 47 2F 48 00 68 00 68 03 21
00 22 03 46 02 20 98 47 2B 48 00 68 00 68 03 21
00 22 03 46 04 20 98 47 27 48 00 68 00 68 03 21
00 22 03 46 08 20 98 47 24 48 00 68 C0 69 0C 21
01 22 03 46 01 20 98 47 21 48 00 68 22 49 00 68
02 46 20 48 90 47 1E 48 00 68 80 68 01 46 1D 48
88 47 1A 48 00 68 80 68 01 21 02 46 01 20 90 47
16 48 00 68 00 68 01 21 02 46 01 20 90 47 14 48
00 68 00 6B 02 46 13 48 01 21 90 47 0D 48 00 68
80 69 01 46 20 20 88 47 0A 48 00 68 00 68 80 47
01 20 01 21 00 F0 BC F8 0C 48 00 90 00 98 00 28
```

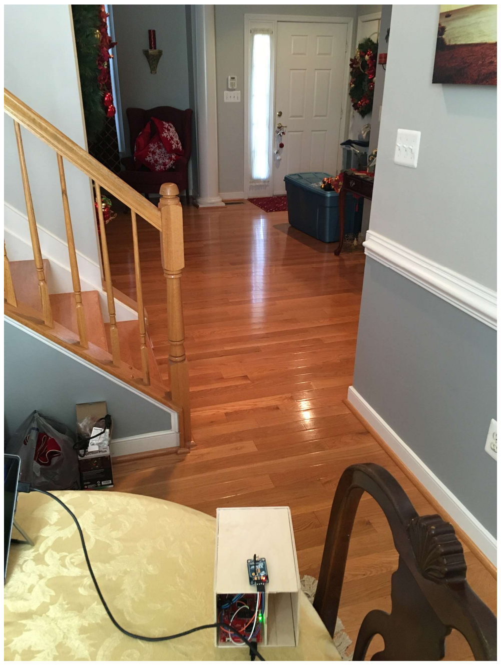
Figure 5.1

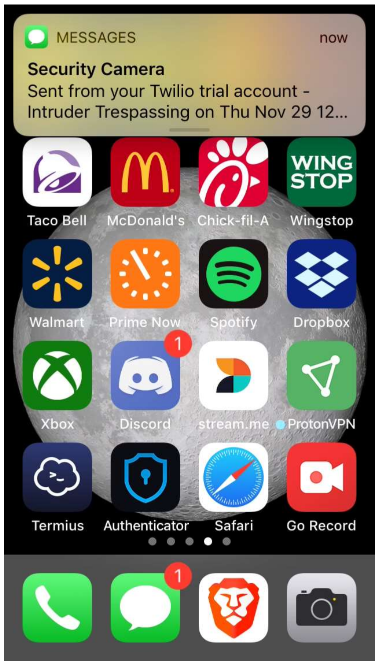


```
0x00000000  00 00 04 20 49 03 00 00 5D 03 00 00 5D 03 00
0x0000000F  00 5D 03 00 00 5D 03 00 00 5D 03 00 00 00 00
0x0000001E  00 00 00 00 00 00 00 00 00 00 00 00 00 00 5D
0x0000002D  03 00 00 5D 03 00 00 00 00 00 00 5D 03 00 00
0x0000003C  5D 03 00 00 5D 03 00 00 5D 03 00 00 5D 03 00
0x0000004B  00 5D 03 00 00 5D 03 00 00 5D 03 00 00 5D 03
0x0000005A  00 00 5D 03 00 00 5D 03 00 00 5D 03 00 00 5D
0x00000069  03 00 00 5D 03 00 00 5D 03 00 00 5D 03 00 00
0x00000078  5D 03 00 00 5D 03 00 00 75 02 00 00 5D 03 00
0x00000087  00 5D 03 00 00 5D 03 00 00 5D 03 00 00 5D 03
0x00000096  00 00 5D 03 00 00 5D 03 00 00 5D 03 00 00 5D
0x000000A5  03 00 00 5D 03 00 00 5D 03 00 00 5D 03 00 00
0x000000B4  5D 03 00 00 5D 03 00 00 5D 03 00 00 5D 03 00
0x000000C3  00 5D 03 00 00 5D 03 00 00 5D 03 00 00 5D 03
0x000000D2  00 00 5D 03 00 00 5D 03 00 00 5D 03 00 00 5D
0x000000E1  03 00 00 5D 03 00 00 08 B5 35 48 00 68 00 68
0x000000F0  80 47 34 48 00 68 40 68 80 47 33 48 00 68 00
0x000000FF  69 01 46 4F F4 80 30 88 47 2F 48 00 68 00 68
0x0000010E  03 21 00 22 03 46 02 20 98 47 2B 48 00 68 00
0x0000011D  68 03 21 00 22 03 46 04 20 98 47 27 48 00 68
0x0000012C  00 68 03 21 00 22 03 46 08 20 98 47 24 48 00
0x0000013B  68 C0 69 0C 21 01 22 03 46 01 20 98 47 21 48
0x0000014A  00 68 22 49 00 68 02 46 20 48 90 47 1E 48 00
0x00000159  68 80 68 01 46 1D 48 88 47 1A 48 00 68 80 68
0x00000168  01 21 02 46 01 20 90 47 16 48 00 68 00 68 01
0x00000177  21 02 46 01 20 90 47 14 48 00 68 00 6B 02 46
0x00000186  13 48 01 21 90 47 0D 48 00 68 80 69 01 46 20
0x00000195  20 88 47 0A 48 00 68 00 68 80 47 01 20 01 21
0x000001A4  00 F0 BC F8 0C 48 00 90 00 98 00 28 E5 D0 00
0x000001B3  98 40 1E 00 90 00 98 00 28 F9 D1 DE E7 64 08
0x000001C2  00 02 2C 08 00 02 14 08 00 02 24 08 00 02 60
```

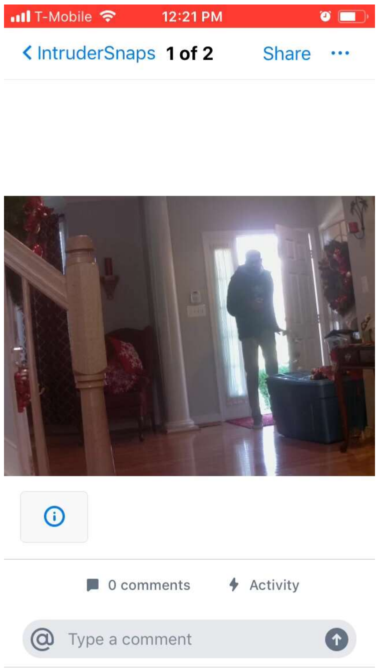
Figure 5.2

The photo below from left to right demonstrate the functionality of taking photos of intruders





Security Camera
Sent from your Twilio trial account - Intruder Trespassing on Thu Nov 29 12...



6) Other Issues

- FPGA

Progress has not been conclusive as the FPGA can be programmed via LCP programmer and programming of FPGA via only microcontroller is not complete.

Obstacles were that Microsemi does not provide enough documentation about the FPGA. The company urges to buy the LCP programmer or flash pro 4 or 5 and program the FPGA using one of those devices. The sample DirectC project provided by Microsemi is not functional (Does not even build). The provide sample project is in IAR environment. The project at first does not even built without modifications or optimizations. Also, we only have access to the student version of IAR tool which is code limited. Once more research about IAR tool and DirectC was done and possible optimizations were made to the provided sample DirectC project, IAR tool returns the error message saying that subscription needs to be upgraded to the full version. Full version of IAR tool is very expensive and does not justify being bought since the required project is then ported to the Code Composer environment where the FPGA will be programmed. IAR tool is only in the picture to fundamentally understand the DirectC code. There was an advice to buy the microcontroller used in the sample project. The microcontroller that is being used in the sample project is the Microsemi microcontroller which is not too expensive (<$48.00) however step by step process to program the FPGA can't be observed since IAR code limited error comes up which can't be taken care of unless complete (not code limited) IAR version is bought. Once all the required steps to program the FPGA are complete and FPGA was not successfully being programmed, the focus shifted from debugging software to debugging hardware. Oscilloscope was connected to the VJTAG and VPUMP pins of the FPGA when programmed via LCP programmer. It was observed that FPGA was pulling current from 50mA to 70mA on VPUMP pin and a fluctuating current of less than 14mA on VJTAG pin. When connected via microcontroller, no flow of current was observed on any of the pins(VPUMP,

VJTAG). Datasheets clearly say there must be a current flow. To further investigate that, Microsemi engineer was called and they refused to help saying the FPGA being used is possibly obsolete. The LCP programmer header on FPGA was also tried to program to FPGA and JTAG-enable pin was hard coded to high, low, and clock (up and down) but success was not achieved.
The *.DAT file is big (234 KB) and could not be hosted on our microcontroller. And another microcontroller (MSP432P411) was bought to resolve the memory issue. Possible alternative is to use LCP programmer for the programming of FPGA via microcontroller and is recommended by Microsemi. A GPIO to USB serial cable is required and this functionality can be achieved.

- MSP432

Issues that we ran into during of the development of this project is that setting up the development environment for different components took a lot more time than expected because Texas Instruments had recently did a massive overall of how their software development kit dependencies operate with MSP432P401R microcontrollers. Before any code writing began it took several weeks satisfying all dependencies of the MSP432 as creating a project in the IDE simply didn't allow the full feature set usage of the microcontroller. Once the MSP432 environment was set up the update code, motion sensor code, and camera code were built and tested. All interfaced devices like the Arducam, Xbee, PIR motion sensor and the Raspberry PI, even though they are well supported and documented didn’t work out the box and had to be configured and tested before interfaced with the MSP432. It was not a difficult task overall, but it did take time out of the originally planned schedule. In the future there will need to be time accounted for device configuration in the development process.

Then work began on developing the FPGA update system. Due to the Size of the FPGA updates a newer MSP432 was acquired with larger flash memory and RAM. This led to use acquiring the

MSP432P4111. The newer MSP432P4111 lead to some more issues even though it was the exact same device as the MSP432P401R with better specifications. The MSP432P4111 is only a year old so their was little documentation and the code once again had to be migrated to a new SDK environment with manual configuration at every level as the IDE didn't support a example project code for non-Real Time Operating systems.

- PCB

PSPICE was chosen at the beginning to build the design. However, it turned out to be difficult and time consuming to learn and use that software. Therefore, KiCad was chosen later on the progress of the project to handle the design. Once the schematics were completed, we built the circuit design in a 4-layer setup. However, this setup wasn't efficient enough since it could be reduced to a 2-layer setup .Datasheets and footprint were really important in this portion of the design since we could have had problems if some components wouldn't match their footprints or pins allocation .The main issue of the 2 layer setup was that we needed more time to minimize the layout of the board considering the many different connections that were to be made ,and also taking care of the keep out areas of the different components such as the Xbee. Another issue is that the PCB design is always evolving with the changes to the circuit design throughout the semester. Once everything was ready, we finally manufactured the PCB and soldered the components onto the board to test the hardware.

7) Administrative Parts

At the end for the semester we were able to complete two out of the three goals we originally set out to do but on a smaller scope. We were able to develop and update MSP432 microcontroller portion of the node wirelessly, develop an update verification system for the node, as well as take and send photos from the node to the gateway and then to the cloud. What we were not able to accomplish is update the FPGA through the microcontroller.

One of the most unexpected issues that we ran into while developing the project is the loss of two team members during the development process..We lost one computer engineer after ECE 492 ended and we lost one electrical engineer during ECE 493. It required a massive shift on how roles were delegated. The project was comprised of three electrical engineers and two computer engineers. The original roles were Ryan and Sohial would work on updating the FPGA via the MSP432; Aneesh would work on updating the MSP432 and leading the team; Mohamed would work on the updating delivery system through the App, server, and gateway; and Roger would design, manufacture and solder the PCB. This would have allowed for a concurrent development for 4 weeks, then upon development completion we would work for 2 weeks on system integration, and the remainder 4 weeks would be left for testing the whole system.

After losing a member in 492 the new roles were delegated as follows. Ryan would lead the team, work on Updating the MSP432 and work closely with Sohial on updating FPGA; Roger worked on designing manufacturing and soldering PCB, and Mohamed would develop the Update system on the gateway, server and app. As development progressed through the semester Mohamed who was responsible for the Update system through the App, Server, and Gateway consistently did not show up to our weekly meetings, we as a team were patient with him as he stated that he was going through some health issues but as time progress he would stop responding to communication through text or even phone calls. We kept Mohamed in all emails and group messages with the hope that he would respond but he did not.

After a team evaluation the decision was made in late October reduce the scope of the project to not have the application or homegrown server backend, however a GUI was still needed for the user to update the node and receive photos from the node. During 493 the new roles were delegated as follows. Ryan would lead the team, work on Updating the MSP432, write a program for the gateway to update the node and work closely with Sohial on updating FPGA; Roger would work on designing, manufacturing, and soldering the PCB. requires at least two Computer engineers as there is a heavy amount of embedded system programing in this project. With the loss of two team members, tasks move from being completed concurrently to being completed linearly. Once work on the MSP432 update and photo taking systems were completed and tested focus of all team members was shifted to primarily get the FPGA updated or programmed by the MSP432. This in turn pushed back the development of the PCB design as Roger started to assist us with hardware debugging.

Without a dedicated member to work on programming the FPGA via a microcontroller, most of the development and testing did not occur until the final 4 weeks of the project. We did our best to get out of our comfort zones and meet the deadlines that we originally set out for but the inherent lack of manpower held us back drastically in fully realizing the original scope of the project. On the bright side we were still able to complete some of the major aspects of the project we set out to do.

- Funds Spent

ITEM	QUANTITY	COST
Actel Igloo Nano Starter Kit	**1**	**Donated**
MSP432P401R Launchpad	**2**	**$25.98**
MSP432P4111 Launchpad	**1**	**$17.99**
Arducam OV2640	**2**	**$51.98**
HC-SR501 Motion Sensor	**5**	**$08.89**
XBee S2C	**2**	**$53.90**
XBee breakout board	**2**	**$13.98**
PCB	**3**	**$89.70**
Raspberry Pi	**1**	**$35.00**
	Total:	**$297.42**

- Man Hours
    - **Ryan (Project Manager): 375 Man Hours**
        - Creating update verification systems and update packages.
        - Responsible for writing C code to have the MSP432 take photos, reprogram itself.
        - Assisting with FPGA- MSP communication interface.
        - Programming the MSP432 boot-loader.
        - Write python code to update the node via Raspberry Pi.
        - Test hardware and debug software.
    - **Sohail: 325 Man Hours**
        - Working on implementing DirectC code for JTAG interface.
        - Optimizing elliptic curve security algorithm to generate the required FPGA files for programming.
        - Writing documentation and creating slides.
        - Test hardware.
    - **Gerson: 325 Man Hours**
        - Create final PCB design for the Node.
        - Handle any issues interfacing via ZigBee to any of the components.
        - Wireless communication between the Xbee modules
        - Test hardware.
        - Collaborating with Sohail to write the slides.
    - **Total Hours: 1025 Hours**

8) Lessons Learned

- Pay attention to device documentation

When doing the research about FPGA, the exact model shown in the datasheet provided by Microsemi was not matching with the one in hand. It was assumed that the FPGA that has somewhat similar naming convention on the datasheet to the FPGA board that is in hand are the same. Also if the *.DAT file was factored in when decision was being made about the microcontroller then some problems could have been resolved earlier. There certainly was some ambiguity about be able to program the FPGA by downloading the bit file to the microcontroller in smaller blocks and do the programming block by block. But even with that, going with MSP432P411 at first instead of going with MSP432P401 would have been a better decision since the cost difference is minimal.

- Better re-planning for primary expectations when undermanned

From the beginning of this project, only 2 out 5 members are working on the things that are the major requirements. (Be able to remotely update the device). Working with devices and performing research on the devices that are used at graduate level can be very time consuming. Assigning more members there to work first on the major expectations would have been a wise and smart thing to do. In the last 4 weeks there was only 1 FPGA and 1 MSP432P4111 that one person would have at time. A better approach would be to put a procedure in place and make a schedule of when and who will have the device for how long. We needed at least one more member proficient in microcontroller, but we would have had a better chance of being successful.

- Read software Documentation

A substantial amount of time was devoted to setting up the IDE of the FPGA and the microcontroller as we were seeking to have the most up to date SDK and device feature set. The syntax for programming setting up project for the MSP432P4111 were drastically different from MSP430 and lead to lose of a few weeks. Also interacting the MSP432 bootloader was very difficult as the MSP432P4111 is a year-old device and has numerous security protocols that needs to meet for the device to be modified in substantial ways.

- Datasheets and time are crucial to build PCBs

While building the PCB layout, we learned that having to place and connect the different components can be challenging because there can be many connections and a 4-layer setup was considered ideal as a solution for this problem. However, it isn't optimal since we can use a 2-layer setup with more difficulty making the connection and taking care of the keep out areas. We had to create certain footprints when it wasn't available to us. Datasheets were very important into the creation of this footprints because one single error in the pin layout of a component could give us many problems.

9) References

- http://www.ti.com/lit/ug/slau356h/slau356h.pdf MSP432 Programmers Guide
- http://www.ti.com/lit/ds/symlink/msp432p4111.pdf MSP432 Datasheet

- https://www.microsemi.com/document-portal/doc_view/130695-ds0110-igloo-nano-low-power-flash-fpgas-datasheet IGLOO Datasheet

- http://www.uctronics.com/download/cam_module/OV2640DS.pdf Arducam OV2640 Datasheet

- https://www.raspberrypi.org/documentation/hardware/computemodule/datasheets/rpi_DATA_CM_1p0.pdf Raspberry Pi Datasheet

- https://www.digi.com/resources/documentation/digidocs/pdfs/90002002.pdf Xbee Datasheet

- https://www.mpja.com/download/31227sc.pdf PIR Motion Sensor Datasheet

- http://kicad-pcb.org/ Kicad PCB Software

10) Appendix A: Proposal (ECE-492)

George Mason University

Implementing OTA Updates for an IoT Security Device

Proposal Document

Authors:
Gerson Dalton Cardozo
M. Sohail Iqbal
Aneesh Malhotra
Mohamed Nur
Ryan Thomas

Supervisor:
Dr. Jens-Peter Kaps

May 20, 2018

Contents

1 Executive Summary

This project will build upon a previous ECE 492 project FPGA Enhanced Wireless Sensor Node for IoT Applications. The project sought to use an FPGA to enhance the security of a wireless sensor node network while maintaining low power consumption. The network that this was implemented on was an in-home security system, which detects motion using an IR sensor on a node, and sends a picture of the intruder to the user via the gateway. Security can be an issue in wireless sensor networks and it may be necessary to update the private key of compromised node or update the firmware of a node to enhance its performance and functionality. The most efficient and inexpensive way to distribute such an update to many users without having to dismantle the system is over-the-air (OTA). In this process a manufacturer will distribute a secure update to its users and the system will update its own firmware. Our goal in this project is to provide OTA update capability to the in-home security system developed by the previous group.

2 Approach

2.1 Motivation

Many IoT devices now use wireless-sensor networks in which a user is able to control several devices called ”nodes” through a single device called a ”gateway”. These kinds of networks are used in many popular smart home devices such as Google’s Nest and Phillips Hue, most of which already provide OTA update capability. Additionally, security for these networks is becoming a concern. Wireless sensor networks are often deployed to monitor and respond to events occurring in the environment and are meant to be left unattended, making them susceptible to a variety of attacks [Sen, 2013]. With the versatile capabilities of FPGA’s, we see many researchers using FPGA’s to enhance the capabilities of wireless sensor networks, including security [G et al., 2016]. As FPGA’s make their way into more wireless sensor networks, we will need to be able to be able to equip them with OTA update capability. Our goal is to provide this capability to our network that uses an FPGA for enhanced security.

2.2 Problem Analysis

The problem we face is to be able to provide OTA updates to the node of the previous system (Figure 1). Since the node is not directly connected to a computer, we must be able to reprogram the node using only the existing hardware on the node. Additionally, the node consists of an MSP432 microcontroller and an Actel IGLOO FPGA. Updating the security features such as the private key and the algorithm on the FPGA will require us to be able to reprogram the FPGA remotely. Likewise, updating the firmware and capabilities of the microcontroller will require us to be able to reprogram the MSP432 remotely.

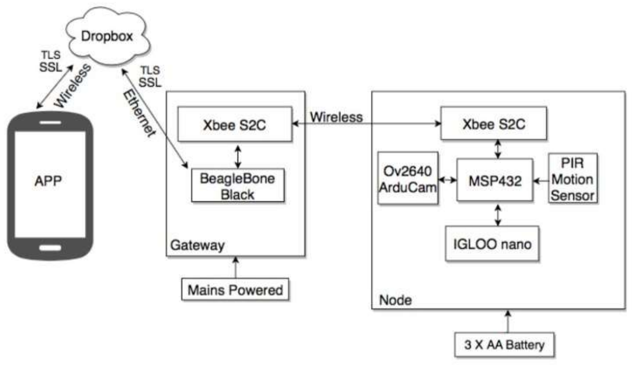


Figure 1: Top Level Diagram of Camera Security System

2.3 Solution

Our solution to this problem is to redesign the system to be able to request updates from an external sever, download the update to the Gateway, push the update to the MSP432 via the XBee, and allow the MSP432 to reprogram both itself and the FPGA through a bootloader. The update will simply consist of a .zip file that consists of a YAML file, the updated code, and some header information. Our goal is to be able to parse the update on the Gateway, and send bootloader commands to the node. The MSP432 on the node will then use these commands to reprogram its own program memory, as well as interface to and reprogram the FGPA.

2.4 Requirements Specifications

- External Server
  - Contains secure signature and contains update ?les.
  - Stores user data collected from the node such as images.
- Update Contains a .zip file with a YAML ?le as well as the source code. The YAML file will have an MSP432 and FPGA section, and will be converted to bootloader commands.
- Phone Provides interface to user Checks for and initiates updates Allows for rekeying and initiating capture from the node camera.
- Gateway
  - Main system component
  - Contains a BeagleBone Black running Linux and an XBee to communicate with the node
  - Parses YAML ?le, and maps it to bootloader commands for the MSP432.
  - Takes in data from the node and makes it accessible to the user.
- Node
  - Secondary system component
  - Contains an MSP432, Actel IGLOO FPGA, an XBee, an ArduCam v5 5MP camera, and an IR sensor.
  - The MSP432 controls all other components and will have the capability to re-program itself and the FPGA.
  - The node will capture an image and send it back to the Gateway.

2.5 Top-Level Design

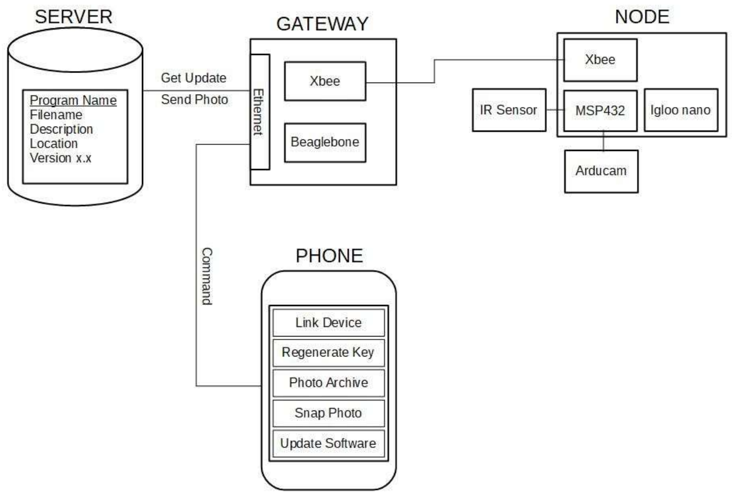


Figure 2: The camera security system no longer uses Dropbox, but rather a server to host the updates.

1. The server contains a file with the version number and location of an update.
2. The user compares the version number of the update to that of the system.
3. If the version numbers do not agree, the user will have the option to begin an update.
4. The Gateway will request the complete update, process it, push the the update and appropriate MSP432 commands to the node.
5. MSP432 will receive the update from the Gateway as well as instructions for reprogramming itself and the FPGA. This will be done via Bootloader commands.

The following diagram depicts this workflow.

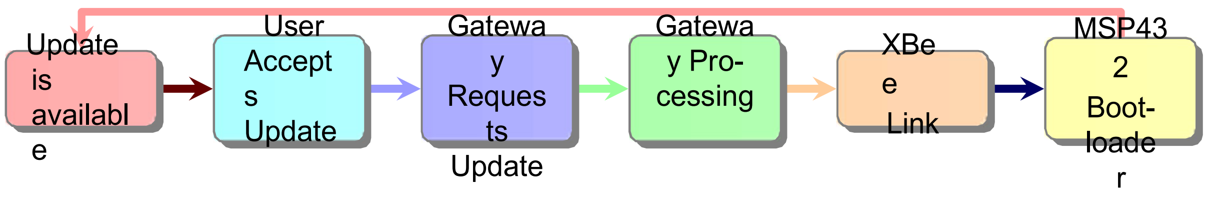


Figure 3: Update Workflow

2.6 Alternative Designs

2.6.1 Dropbox

We had initially decided to use Dropbox as a cloud storage system. Since the gateway is not a member of the local area network, this would have made communication between the Android device and the gateway to be much simpler. This communication, however, is much slower. Additionally a company pushing a firmware update to a user will not typically have access to a user's Dropbox account. A user can also accidentally delete important configuration files such as the ones used for rekeying, making it less reliable and less realistic than a dedicated server. For these reasons, we decided to use a dedicated web server for distributing updates and storing data.

2.7 FPGA Programming

There are many ways about which we can program the ACTEL IGLOO Nano using a microcontroller. One method was rather simple, and involved using a power FET to control the different operating modes, but this component is rather

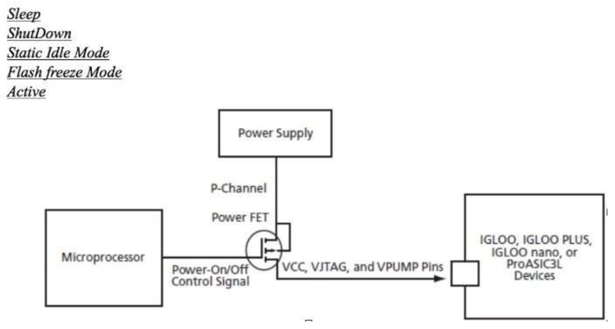
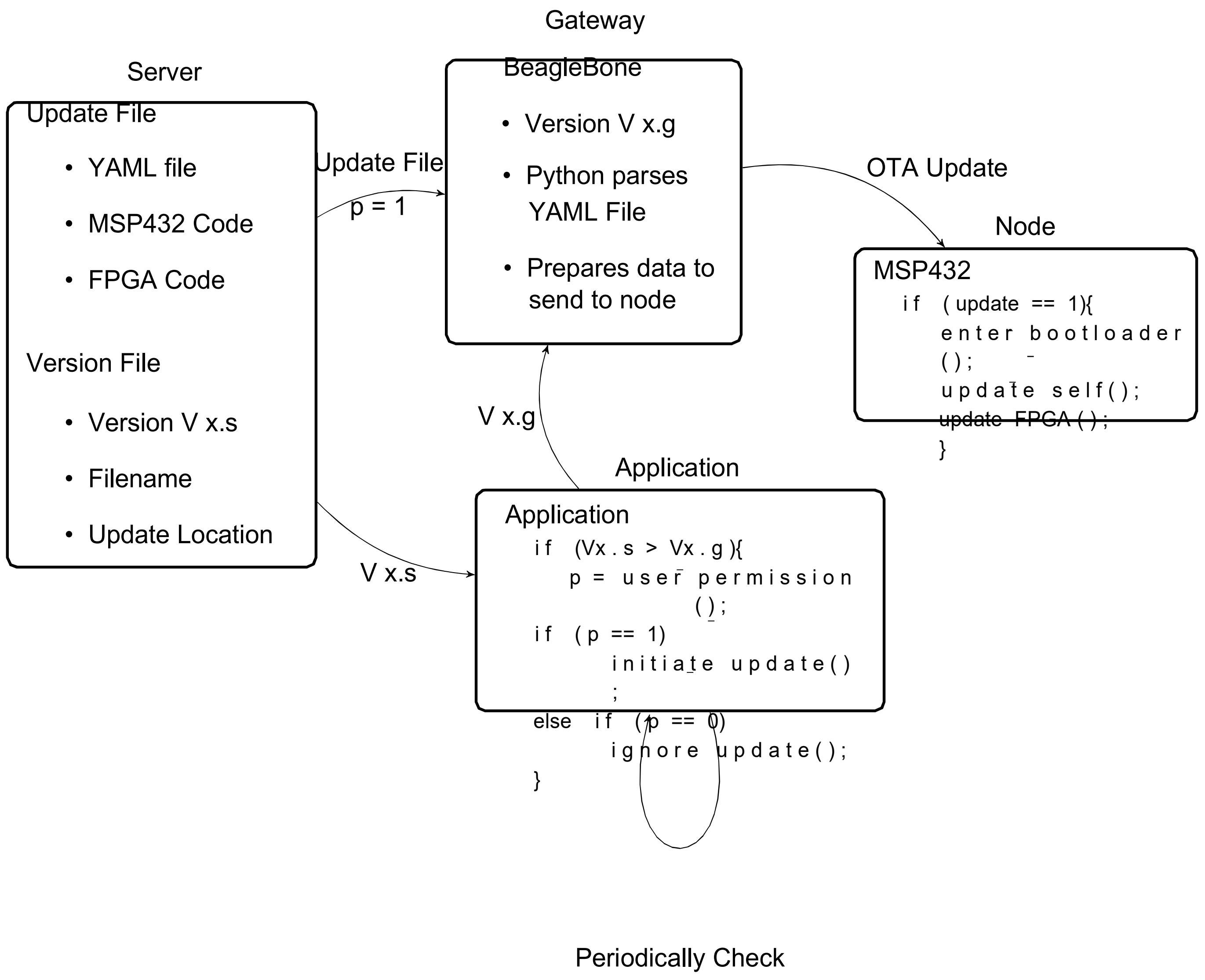

Sleep
ShutDown
Static Idle Mode
Flash freeze Mode
Active
Power Supply
P-Channel
Power FET
Microprocessor
Power-On/Off Control Signal
VCC, VJTAG, and VPUMP Pins
IGLOO, IGLOO PLUS, IGLOO nano, or ProASIC3L Devices

Figure 4: Different Operating Modes of the Actel IGLOO Nano

3 System Architecture

3.1 Architecture with Pseudo-Code



```
if (update == 1){
    enter_bootloader();
    update_self();
    update_FPGA();
}
```

```
if (Vx.s > Vx.g){
    p = user_permission();
if (p == 1)
        initiate_update();
else if (p == 0)
        ignore_update();
}
```

3.2 Architecture 2

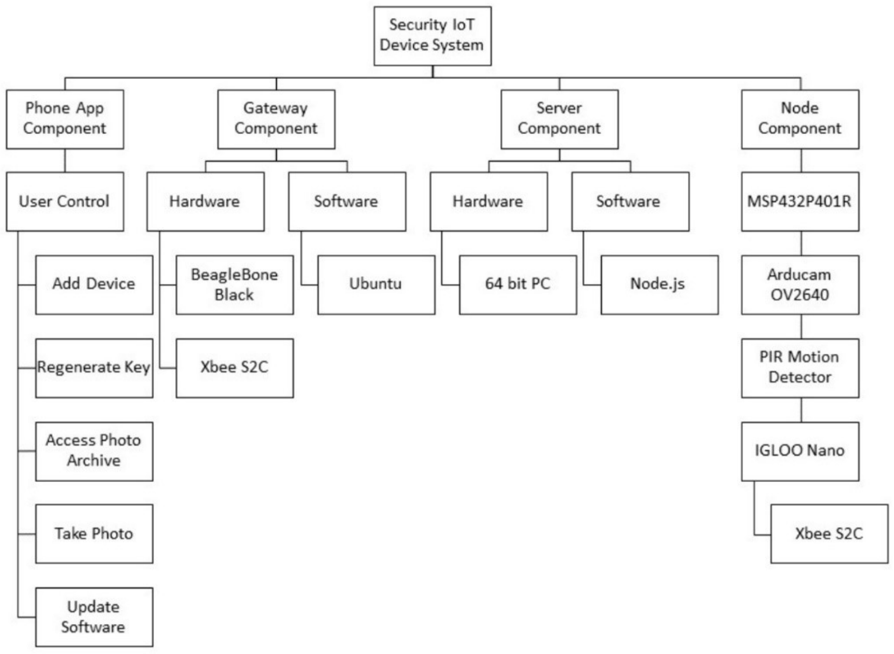


Figure 5: The system architecture with all components and functionality

3.3 Node State Diagram

The following is the state diagram of the node in the IoT Device.

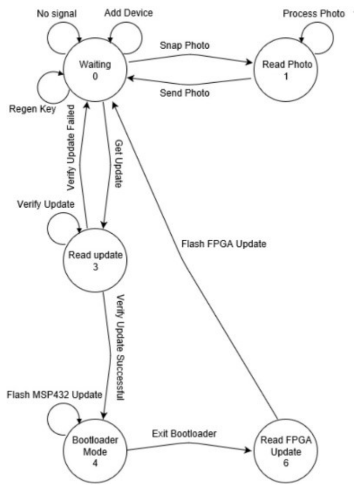


Figure 6: State diagram for the node

3.4 Gateway state diagram

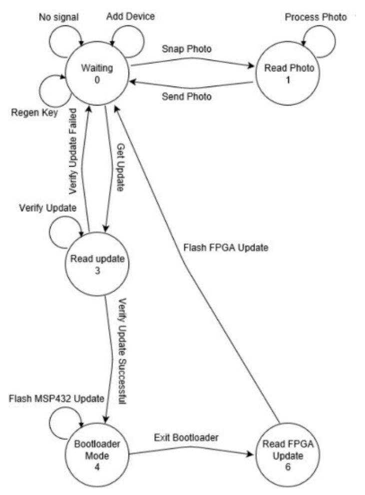


Figure 7: State diagram for the node

3.5 Gateway State Diagram

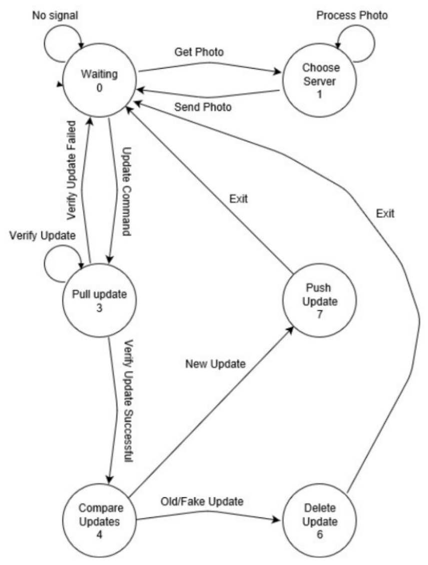


Figure 8: State diagram for the gateway

4 Background Information

The project we are building is an addition to a previous Senior Design Project the background knowledge required can be broken up into the following sections:

4.1 Current System

- Memory
  - Current Flash Main Memory usage is 7.5KB based on current program
  - Current SRAM usage is 2KB for the program and 60KB for the photo
  - Current total SRAM Usage is 62 KB with 8KB of SRAM remaining
- Serial Communication
  - eUSCI_A Modules
  - UART is utilized by the Xbee S2C
  - eUSCI_B Modules
  - GPIO is used be Arducam OV2640, IGLOO nano, and HC-SR501 PIR MOTION DETECTOR
  - SPI is used be Arducam OV2640

4.2 PCB Design

For the PCB we will need to optimize the design for testing. We will need to make each component accessible for testing the power consumption of each component (using a multimeter). We will use KiCad for the PCB design, and will need to learn to use KiCad.

4.3 Server

We will be using Node.js to implement an HTTPS sever that can store the update and some user data. Since this will be used instead of Dropbox, we will have to use the server to store pictures sent from the node to the gateway, since the phone is unable to connect directly to the BeagleBone. We can implement this as a website where images are stored.

4.4 FPGA/MSP432

We will also require a profound knowledge of coding C for programming the MCU that controls the node. Additionally, Linux by adding scripting commands in order for the gateway to push and pull data from the node. On a hardware level there will need to be able to set the clock speeds on the MCU as well configuring pins for serial communication via SPI, I2C, UART. Data on the MCU is handled in 1 byte chunks. And the data transfer speeds are determined by the MCUs clock speeds. We will require knowledge of the STAPL player for programming the IGLOO.

4.5 Security

Keeping the system secure is one of our main goals. In the system that we have so far, we are using the following security protocols

- Security Protocols Used:
  - Transport Layer Security (TLS)
  - 128 bit AES Encryption
  - Produced by the elliptic curve scheme on BeagleBone and FPGA
  - The FPGA has a 128 bit AES encryption within itself to protect data transfer

4.5.1 Elliptic Curve Cryptography

The previous project implemented a security verification scheme based on elliptic curve cryptography. This is implemented as asymmetric cryptography where we generation a session key. This session key is used for securing the communication between the gateway and the node via ZigBee. An elliptic curve is a function of the form under a finite field, such as the Z (mod p)

$$y^2 = x^3 + ax + b$$

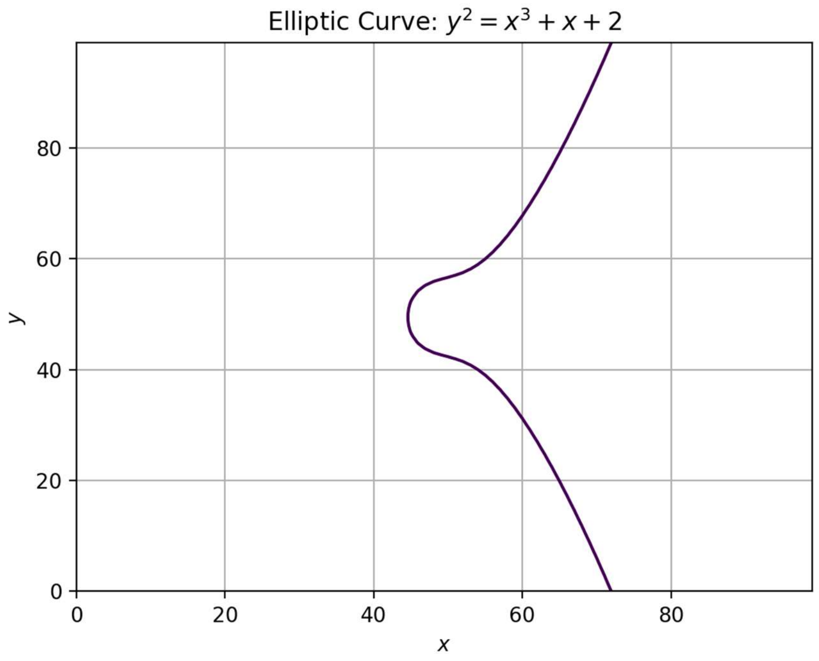


Figure 9: Elliptic Curve for a = 1 and b = 2

We can use these curves to implement a security algorithm in which we use elliptic curve arithmetic to generate a session key. We define the a group operation g on elliptic curves. The BeagleBone's private key will be some number a, and the private key of the Node will be some number M_b The BeagleBone will establish a session key by performing the group operation a times on the public key of the node resulting in $S = M^a_b$, where S is the session key. The Node verifies the session key in the FPGA, which is already programmed to handle heavy arithmetic, and forwards it to the ZigBee as the key used for symmetric AES encryption.

4.5.2 Securing the Update

The update will be secured on the server using TLS (Transport Layer Security), by having the BeagleBone request the update from only the server's certificate and public key. After having this implemented, the update will be secured through the session key that was already established.

5 Prototyping Progress

5.1 List of Acquired Components

1. Actel IGLOO Nano
2. MSP432 Trainer Board

We still need to buy the camera, IR sensor, BeagleBone, and XBee devices.

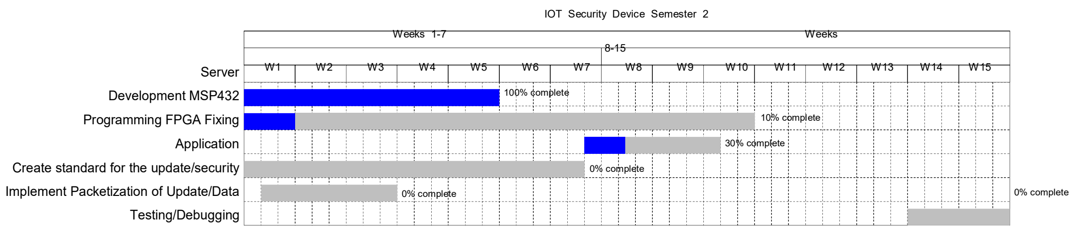
5.2 Current Working Components

We currently have an HTTP server built in Python, but we will rebuild it using Node.js to provide a better user interface. Additionally, we have the FPGA side working. We have a version of the app running as well.

5.3 Testing Plans

We plan on testing the system by adding a noticeable change to the MSP432 code such as toggling the state of an LED. Testing the programmability of the FPGA will be done in a similar way. These experiments are simple, don't involve as much writing as much code, and will allow us to focus on testing programmability.

5.4 Task for Next Semester



Gantt diagram for ECE 493

5.5 Task List

1. FPGA Programming

(a) Simulate a JTAG interface to program the FPGA.

(b) Once MSP432 works, test the model and see if we can change the FPGA code to toggle an LED

2. MSP432 Bootloader
    (a) Setup MSP432 Bootloader
    (b) Use an external signal to activate bootloader mode
    (c) Test bootloader commands by rewriting a block of program memory.
3. Server
    (a) Build HTTP server using Node.js
    (b) Build a website to display images on
    (c) (Theoretically) get a certificate and implement an HTTPS server.
4. Update
    (a) Finalize update design, and make sure everybody knows what the update will looks like
    (b) Build a parser using Python on the BeagleBone that can read the update, and send information to the node.

References

[G et al., 2016] G, L., K, S., and R, V. K. (2016). Elliptic Curve Cryptography implementation on FPGA using Montgomery multiplication for equal key and data size over GF(2m) for Wireless Sensor Networks. In 2016 IEEE Region 10 Conference (TENCON), pages 468–471.

[Sen, 2013] Sen, J. (2013). Security in Wireless Sensor Networks. arXiv:1301.5065 [cs]. arXiv: 1301.5065.

11) Appendix B: Design Document (ECE-492)

Over the Air Updates for an IoT Security Device

By: Ryan Thomas, Sohail Iqbal,
Gerson Dalton, Mohamed Nur

8/8/18

Problem:

- IoT devices are becoming increasingly popular for commercial use due to the else of the smart home industry.
- With the proliferation of more complex smart devices throughout households, the question of security becomes an issue
- In the event that a security flaw in the node network is discovered, there needs to be a way to implement security patches.
- The security patches themselves also need a level of security in order to prevent hackers from delivering a security patch with a loophole.

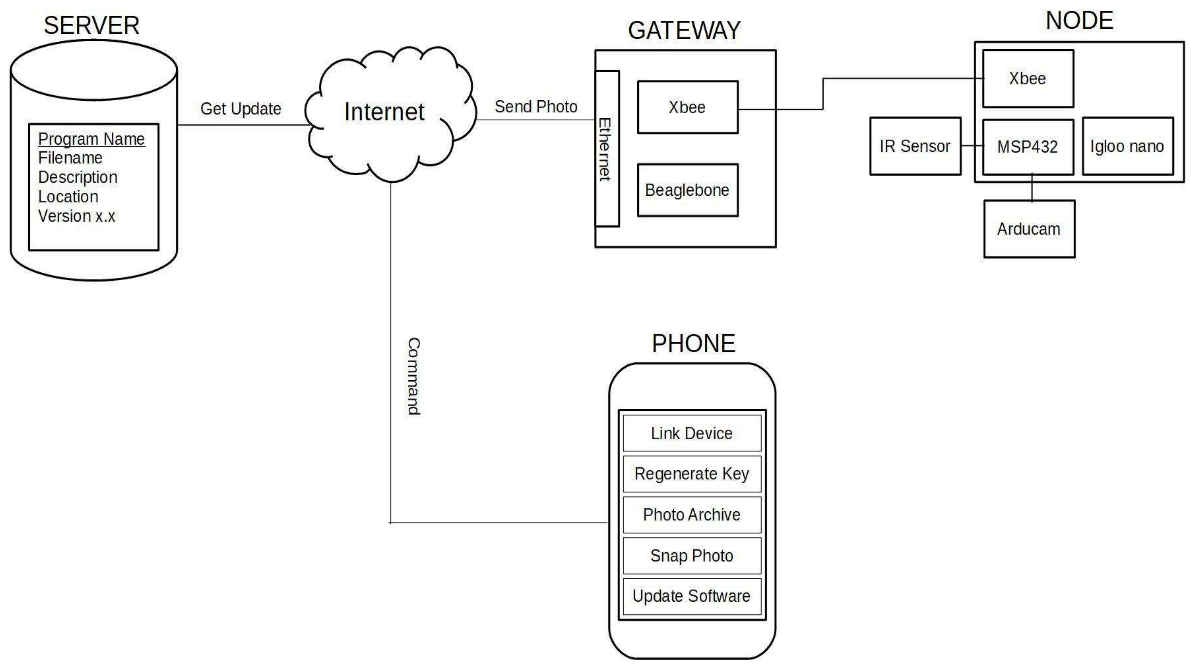
Solution:

- An Over the Air (OTA) Update delivery system is needed to provide security patches to the wireless sensor nodes.
- The node receives a security patch and enters a boot-loader mode to write the security patch to itself.
- Security patches need to be held on a private server that only the nodes have access to via a private gateway.
- In the event that an older security patch is uploaded to the server the gateway must compare the OS version of the node with the OS version on the server.
- In the event that the server is compromised there is a checksum or private key that the gateway must verify in order push a security patch to the node.

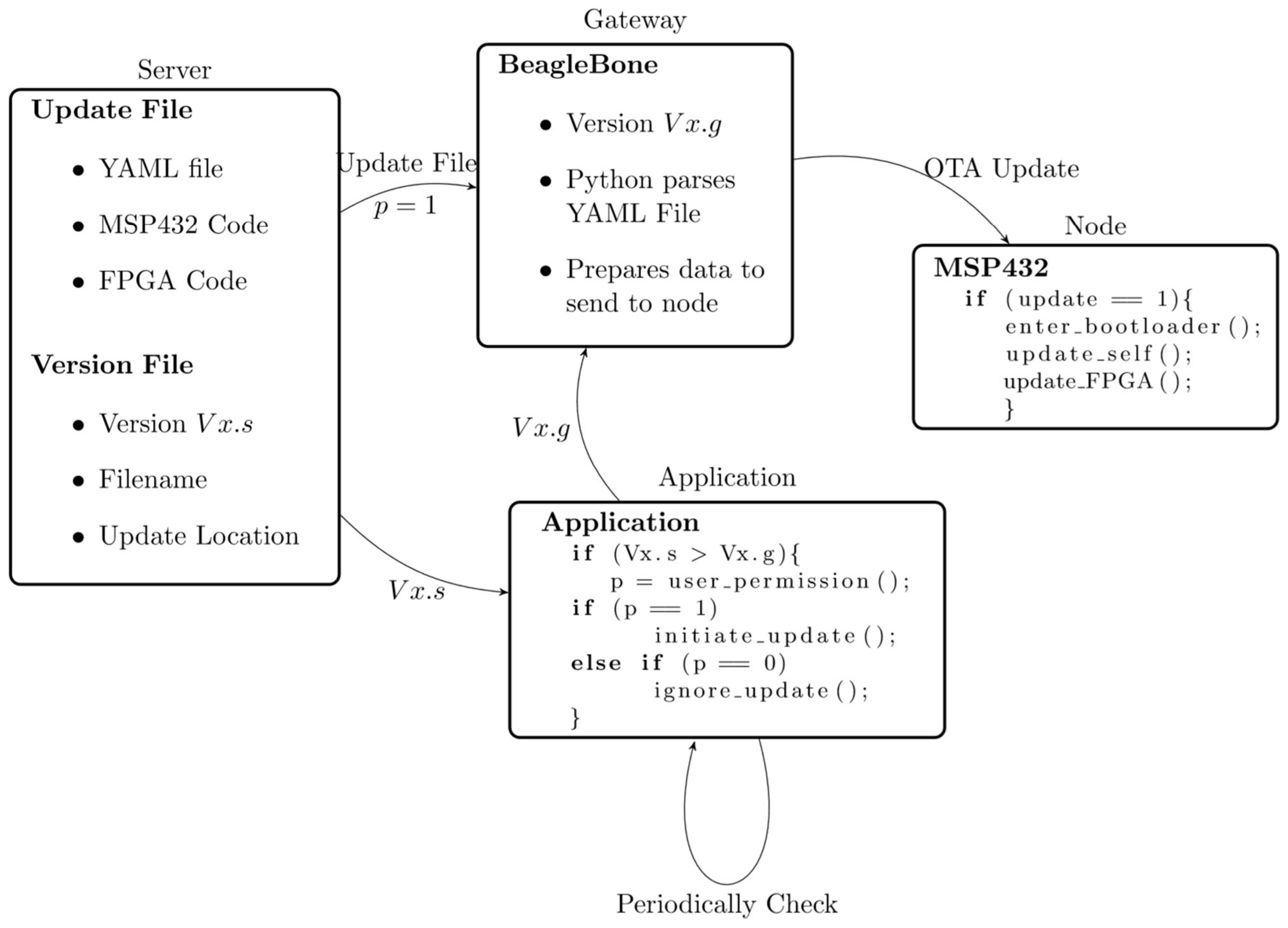
Requirements Specification

- External Server
  - Contains secure signature and contains update files.
  - Stores user data collected from the node such as images.
  - Verifies if nodes needs an update via version verification
- Update
  - Contains a .zip file with a text file as well as the source code.
  - The text file will have an MSP432 and FPGA section, and will be converted to bootloader commands.
- Phone
  - Provides interface to user
  - Checks for and initiates updates
  - Allows for rekeying and initiating capture from the node camera.
- Gateway
  - Contains a Raspberry Pi running Linux and an XBee to communicate with the node
  - Parses text file, and maps it to bootloader commands for the MSP432.
  - Takes in data from the node and makes it accessible to the user.
- Node
  - Contains an MSP432, Actel IGLOO FPGA, an XBee, an ArduCam v5 5MP camera, and an IR sensor.
  - The MSP432 controls all other components and will have the capability to reprogram itself and the FPGA.
  - The node will capture an image and send it back to the Gateway.

Conceptual Sketch



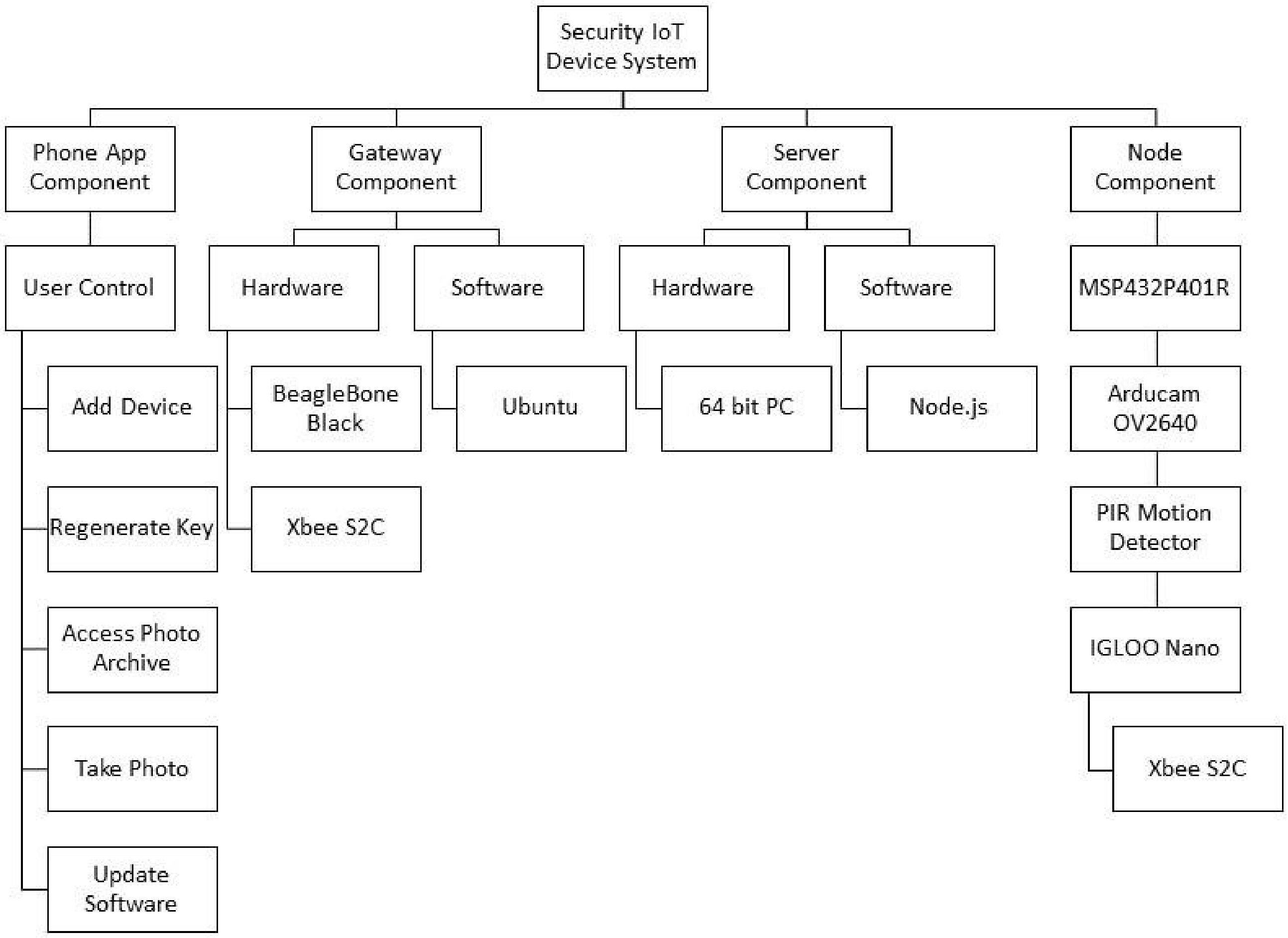
Functional Architecture



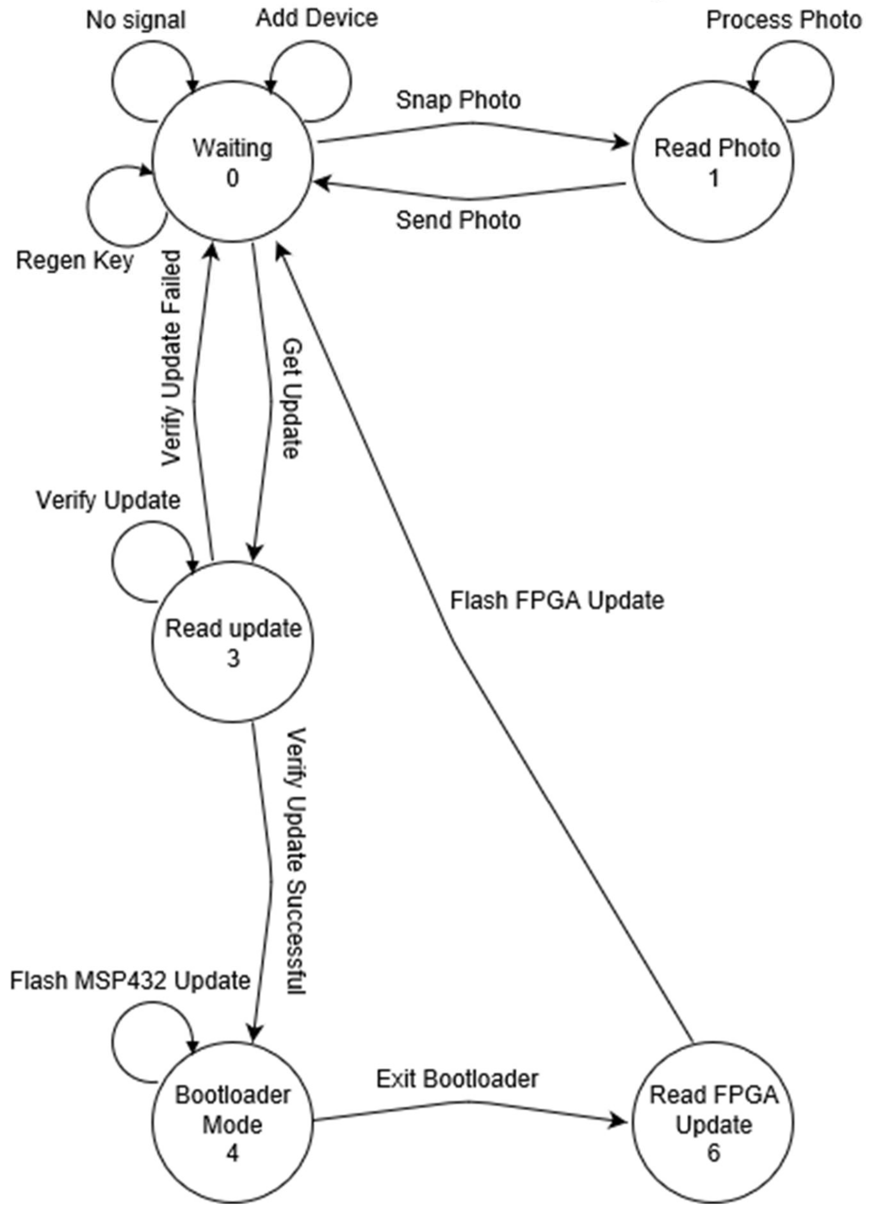
```
if (update == 1){
    enter_bootloader();
    update_self();
    update_FPGA();
    }
```

```
if (Vx.s > Vx.g){
    p = user_permission();
if (p == 1)
        initiate_update();
else if (p == 0)
        ignore_update();
}
```

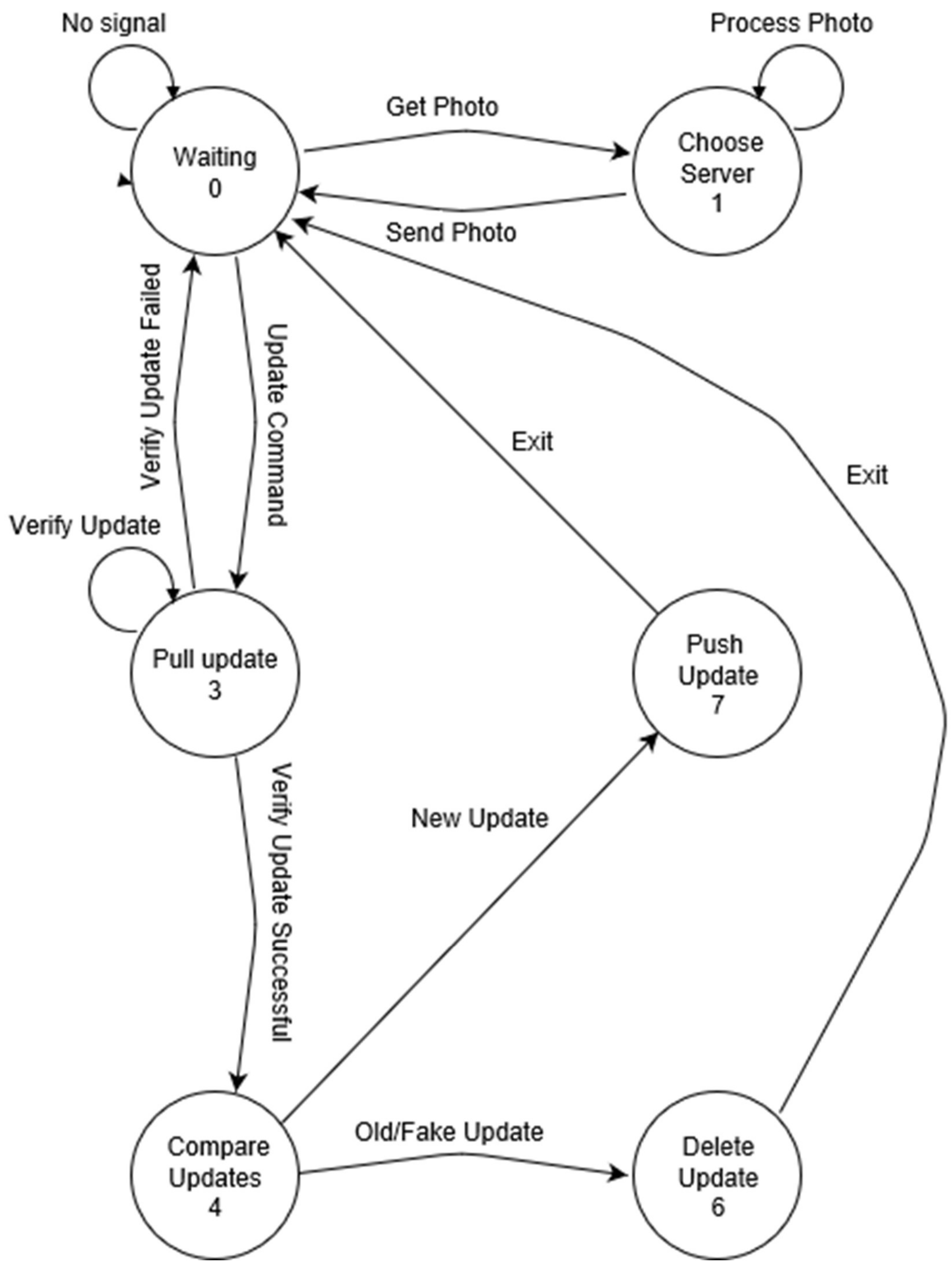
System Architecture



Node State Diagram



Gateway State Diagram



Background Knowledge

Data Interfacing

- SPI Data Interface
    - Shift Registers store eight or sixteen bits
    - Devices transfer one frame at a time
    - A word is transferred, by the processor, to the TX buffer before it is transmitted over the serial link
    - This occurs on both the master and the slave devices.
    - One bit is transmitted at a time (serial communication)
    - When a bit is received the register shifts and transmits the next bit

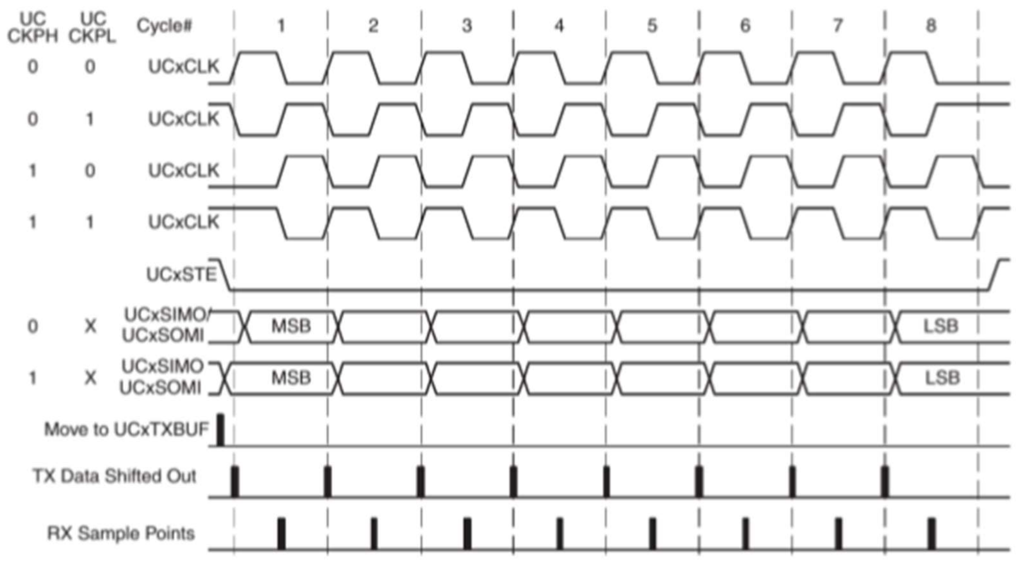
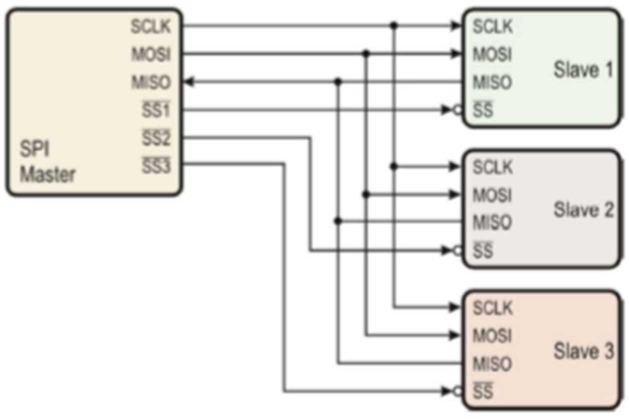


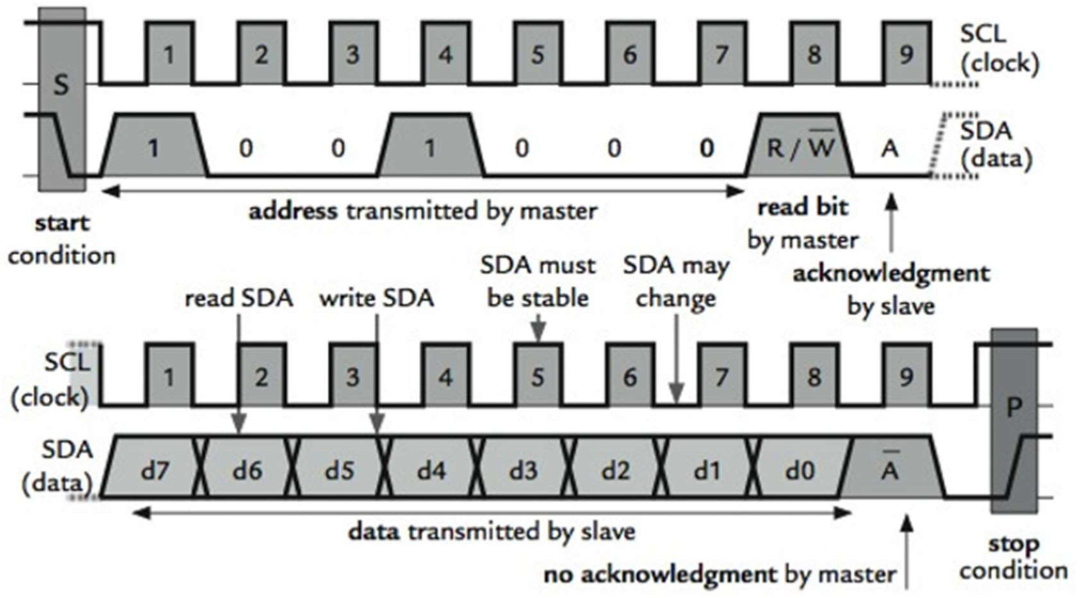
Figure 22-4. eUSCI SPI Timing With UCMSB = 1

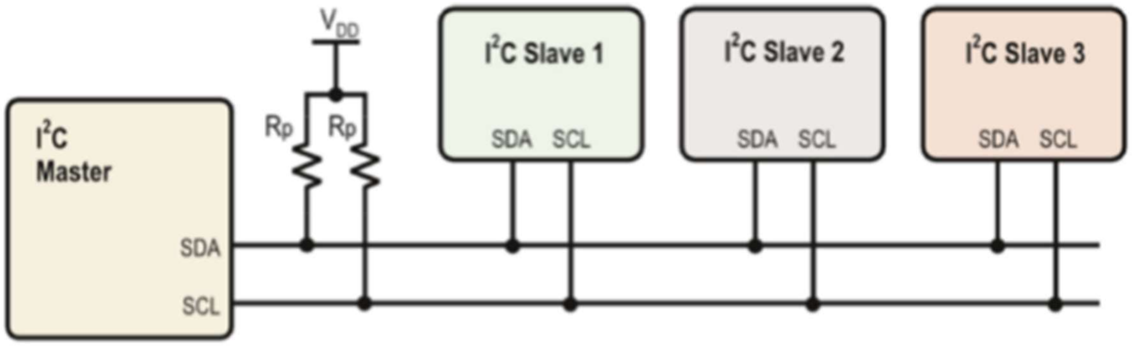
- I2C Data Interface

Fig. 9.22 SPI single-master, multi-slave connection

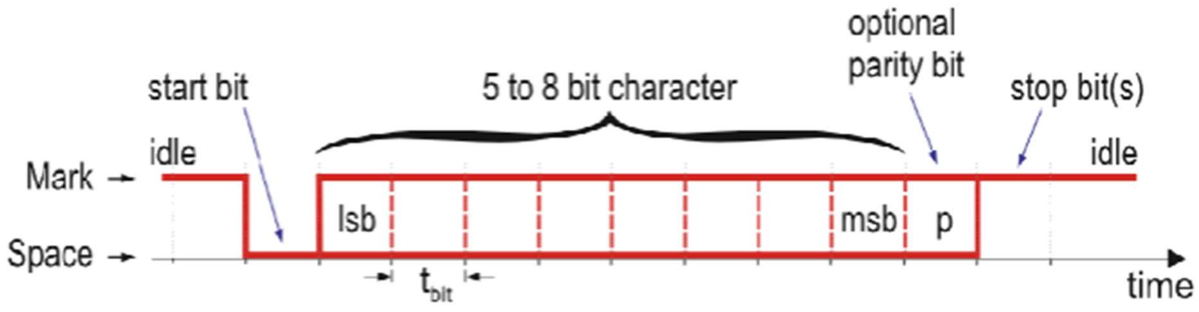


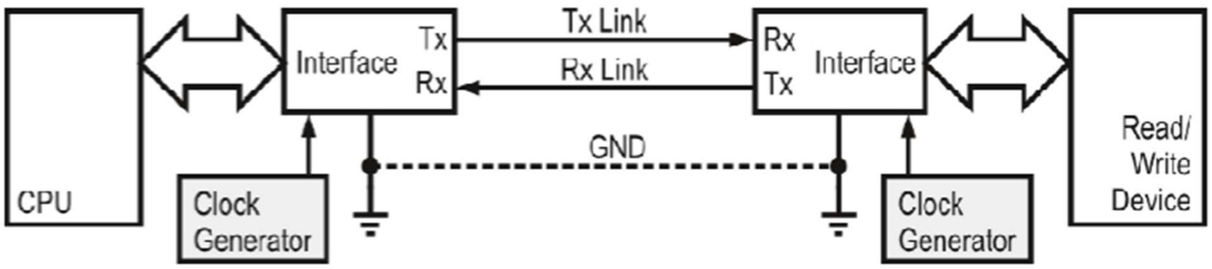
- The master sends a start condition by pulling SDA low while SCL is high
- The master starts the clock and puts the first bit of the address on SDA after SCL has gone low
- The value on SDA is valid after SCL has gone high and is read by all slaves on the bus
- The next 8 clock cycles are used to transmit 1 byte of data from the slave to the master
- There is a final cycle of the clock to set up the stop signal



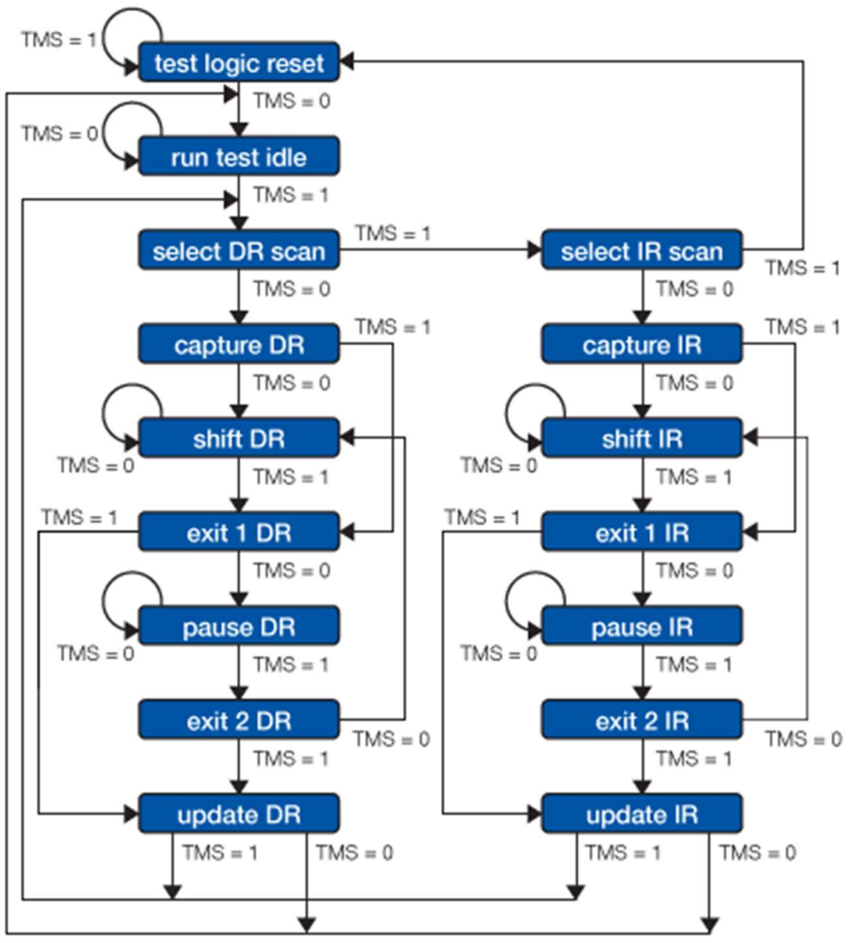


- UART Data Interface
    - Baud rate is the freq. at which bits (not data) are transmitted
    - A baud rate for embedded systems is 9600 bits/sec.
    - Data is sent in frames, each containing a one start bit ,eight data bits, and one stop bit





- JTAG Data Interface (Achieved through bit banging on MSP432)
    - The master sends a start condition resetting the JTAG interface
    - Then the master starts the clock and proceeds to send series of bits that shift the state machine to read instructions (Capture IR, Shift IR, Exit IR)
    - The next 8 clock cycles are used to transmit 1 byte from the master to the slave with the proper instruction
    - The master will proceed to send series of bits that shift the state machine to read data(Capture DR, Shift DR, Exit DR)
    - The next 8 clock cycles are used to transmit 1 byte from the master to the slave with the proper data
    - There is a final cycle of the clock to reset the JTAG interface for future use



Detailed Design

Communications

This section describes the communications protocol for each action initiated by the Application or the Node. Each action has one flow diagram and a description of all messages sent and responses received.

Gateway to Node Update system

Upon Pressing the Update Software button the update the Server will request the software version number of the node and compare it with the latest software version stored on the compute through the gateway.If there is newer software a signal from the app will be sent to the server to push the update. He gateway will receive the update and send a character to the MSP432 will exit low power mode read the character and if the character is correct it will send back an acknowledge character to the gateway and enter bootloader mode. Upon entering bootloader mode the MSP432 will receive the command, update size, the address to write the update to, and the actual update to write to flash. Once the the MSP42 exits bootloader mode it will send an acknowledge character that the update is complete.

Upon verifying the MSP432 software is up to date, the gateway will send a character to receive the FPGA update. The MSP432 will exit low power mode upon receiving the character, allocate ram for the FPGA Update and wait for the gateway to send the FPGA update. The MSP432 will receive the address to store FPGA update chunk in a ram and the FPGA update chunk itself. The MSP432 will store on the FPGA update chunk in the specified address ram then send the chunk to FPGA via JTAG while configuring the JTAG interface accordingly. The ram buffer will repeatedly cleared for string each chunk of the update until the whole update is sent, After sending the update the MSP432 will send a signal saying the FPGA Update is complete and enter low power mode to await further instructions.

App/Server/Gateway Communication

Across the millions of apps, one thing is common; That is, the strive to make a user-friendly and efficient app. This goal should be apparent as every goal of any app is to increase the number of users, regardless of the category the app falls under. Thus, making an app that the user enjoys using. Furthermore, in this system, the role of the app is very crucial, as it is the starting point of the relay of information required in the operation of our device. In the rest of this section, the process of the app development will be discussed along with the overall functionality of the app. This app will initially be available only on android only, which entails development in Android Studio using Java.

App-Server Connection

Prior to discussing the functionality of the App, we must first discuss the underlying connection between the Node.js server and the app. This connection is very important, as it is the connection that lays the foundation for all the other connections. Through this app-server connection, we will able to transmit to and receive data from the server. Regarding the protocol used in this connection, the two main protocols that were considered are the HTTP and WebSocket.

For this connection, the WebSocket protocol is used for a few reasons. First, since this device is dealing with real time data, HTTP would not be ideal to use. On the contrary, WebSocket is bi-directional, meaning that the server and client can each send and receive messages from the other party. With HTTP, the client always has to initiate connections. In addition to being bi-directional, WebSocket is fully Duplex and both the client and server can talk independently, while with HTTP, either the server is talking to the client or the client is talking to the server at any given time. Lastly, while WebSocket allows for communication via a single TCP connection, a new TCP connection needs to be established for every HTTP request/response.

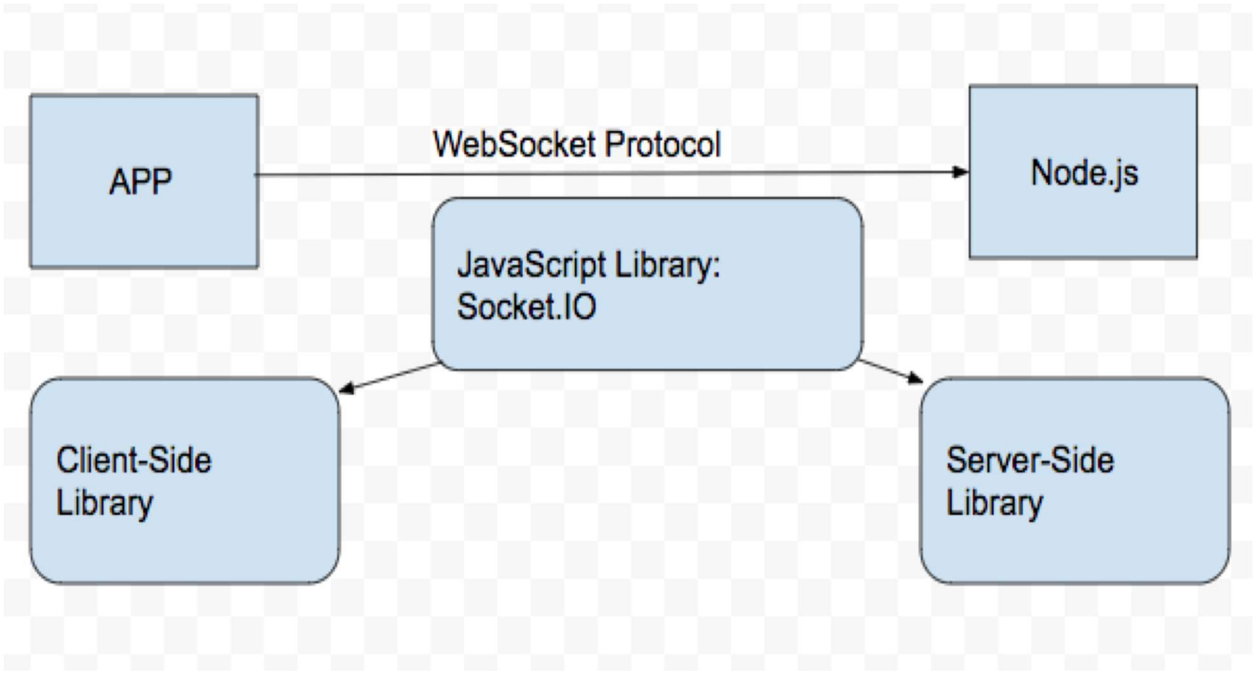
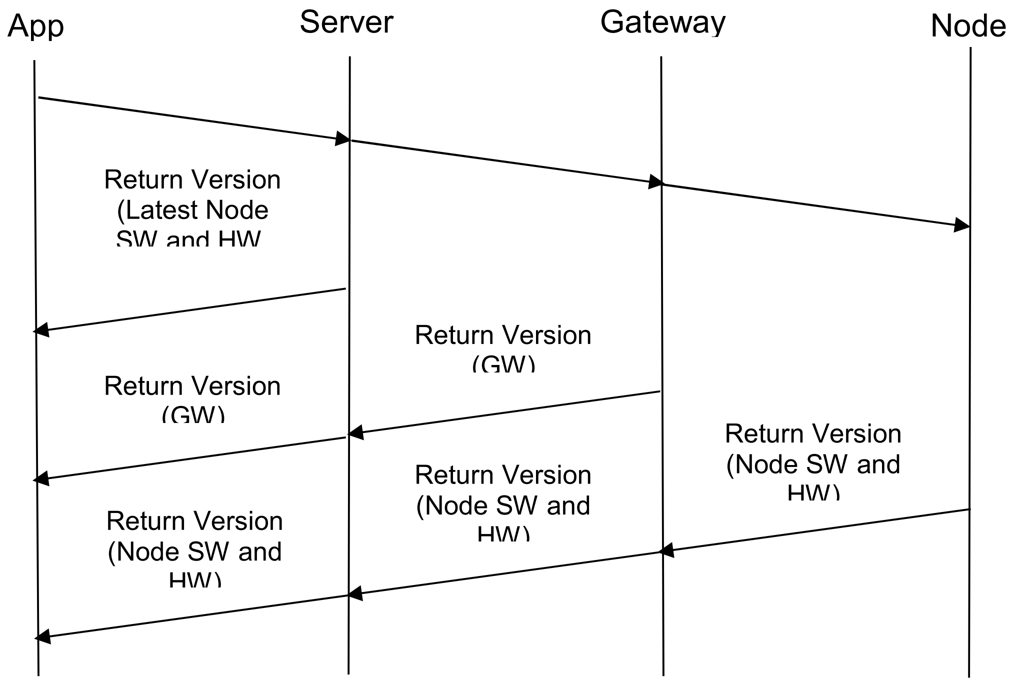


Figure 1: App-Server Connection

Functionality/Operation

Upon opening the app, the user’s dashboard will consist of five buttons: Update Software, Snap Photo, Photo Archive, Regenerate Key, and Add New Device (Link Device). For each button press, the interaction is explained between each component of our system.

Check for Updates


App
Server
Gateway
Node
Return Version (Latest Node SW and HW
Return Version (GW)
Return Version (GW)
Return Version (Node SW and HW)
Return Version (Node SW and HW)
Return Version (Node SW and HW)

Check Update	From: App	To: Server
Initiated by:	User presses “Check for Updates” button on screen	
Message Contents:	● Check Update Command ● Gateway Name ● Node Name	
Expected Response:	3 Messages: ● Gateway Version Number ● Node Software Version Number and Hardware Version Number ● Latest Gateway Version Number, Latest Node Software Version Number and Hardware Version Number	
Response to Initiator:	When any latest version number is greater than the installed version number inform the user. Also inform the user when no updates are available.	
Receiving Damaged Messages:	Inform the user and give the option to try again	
No Response Received	Wait for 20 seconds for responses. Inform the user about each missing response: “*Device* can	

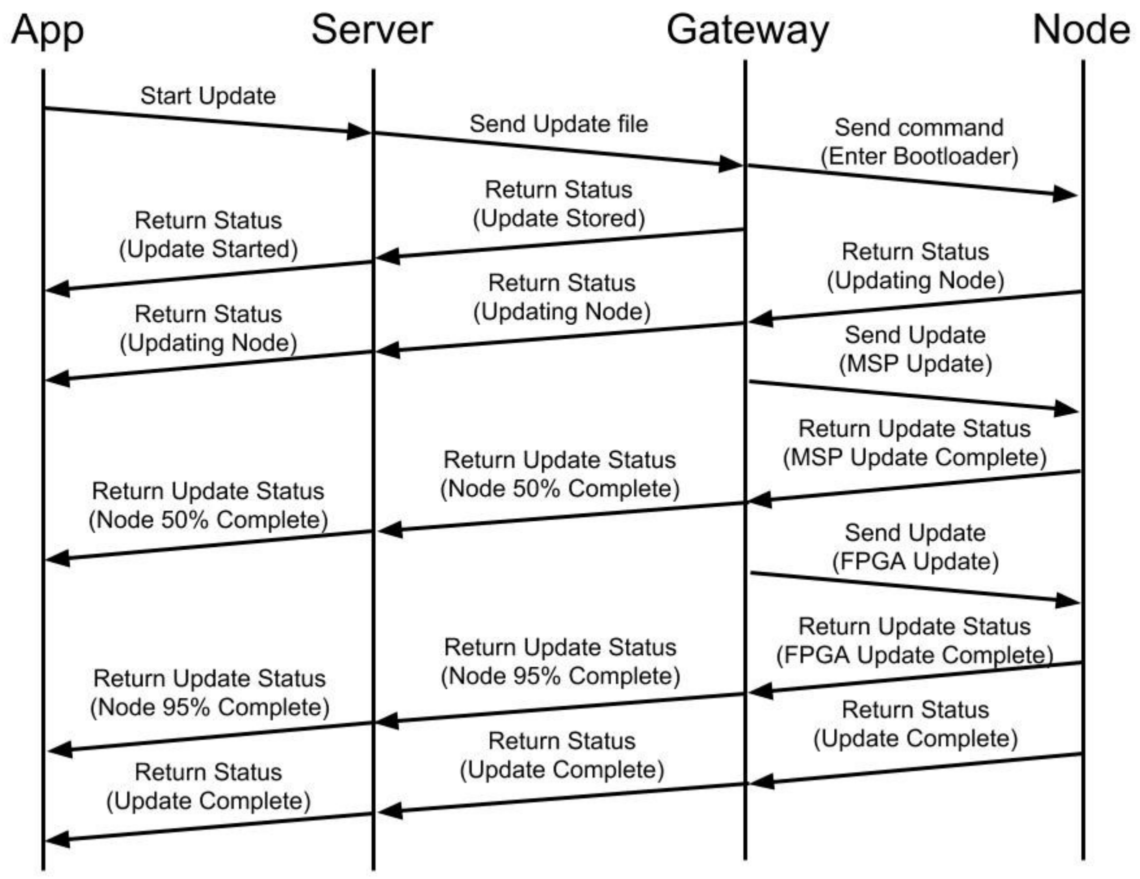
	not be contacted, please try again later.”

Check Version	From: Server	To: Gateway
Initiated by:	Check Update Message from Application	
Message Contents	● Check Version Command ● Gateway Name ● Node Name	
Expected Response:	2 Messages: ● Gateway Version Number ● Node Software Version Number and Hardware Version Number	
Response to Initiator:	● Gateway Software Version Number ● Node Software Version Number and Hardware Version Number ● Latest Gateway Version Number, Latest Node Software Version Number and Hardware Version Number	
Receiving Damaged Messages:	Send Error Message to Initiator.	
No Response Received	Wait for 18 seconds for responses. Send Error Message to Initiator.	

Check Version	From: Gateway	To: Node
Initiated by:	Check Update Message from Server	
Message Contents	● Check Version Command ● Node Name	
Expected Response:	1 Message: ● Node Software Version Number and Hardware Version Number	
Response to Initiator:	● Gateway Software Version Number ● Node Software Version Number and Hardware Version Number	
Receiving Damaged Messages:	Send Error Message to Initiator.	

No Response Received	Wait for 16 seconds for response. Send Error Message to Initiator.

Send Updates


App
Server
Gateway
Node
Start Update
Send Update file
Send command (Enter Bootloader)
Return Status (Update Stored)
Return Status (Update Started)
Return Status (Updating Node)
Return Status (Updating Node)
Return Status (Updating Node)
Send Update (MSP Update)
Return Update Status (MSP Update Complete)
Return Update Status (Node 50% Complete)
Return Update Status (Node 50% Complete)
Send Update (FPGA Update)
Return Update Status (FPGA Update Complete)
Return Update Status (Node 95% Complete)
Return Update Status (Node 95% Complete)
Return Status (Update Complete)
Return Status (Update Complete)
Return Status (Update Complete)

Send Update	From: App	To: Server
Initiated by:	After the app verifies Node SW version is out of date	
Message Contents:	• Send Update Command • Gateway Name • Node Name	
Expected Response:	3 Messages: • Update Progress Started or percentage completed • Node Software Version Number and Hardware Version Number • Latest Gateway Version Number, Latest Node Software Version Number and Hardware Version Number	
Response to Initiator:	When the server starts sending the update file to the gateway send update progress.	

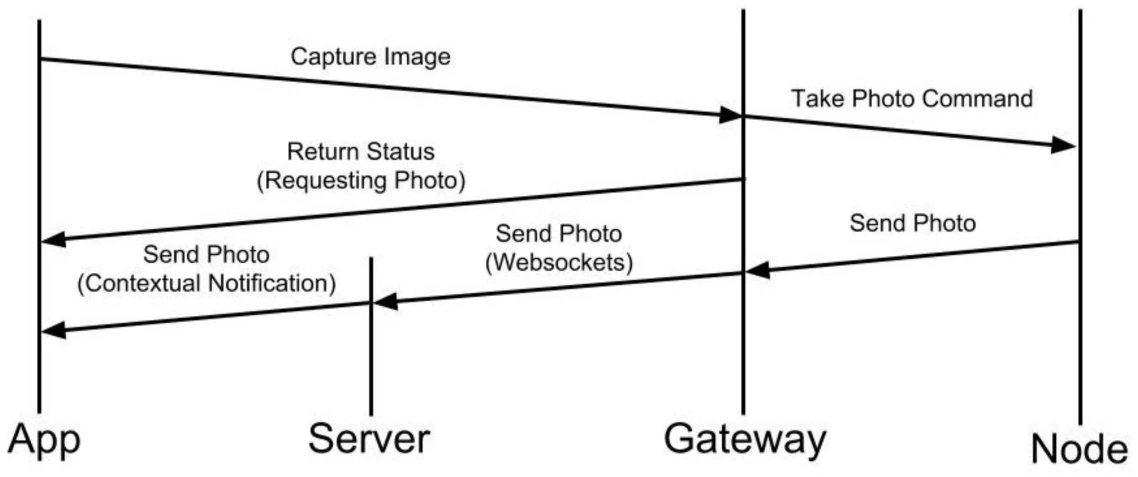
	Inform the user if the Server isn't able to send the update file to the gateway
Receiving Damaged Messages:	Inform the user and give the option to try updating again
No Response Received	Wait for 10 seconds for a response from the server Inform the user about each missing response: "*Update cannot be pushed. Error Code: XX*"

Send Update File	From: Server	To: Gateway
Initiated by:	Receiving Update Command from App	
Message Contents	● Latest Update file ● Gateway Name ● Node Name	
Expected Response:	2 Messages: ● Node update progress (Percentage Complete) ● Node Software Version Number and Hardware Version Number	
Response to Initiator:	● Node update progress (Percentage Complete) ● Recommended Instructions on interacting with Node (Do not unplug power from node or turn off app)	
Receiving Damaged Messages:	Send Error Message to Initiator.	
No Response Received	Wait every 15 seconds for progress update. After 60 seconds send Error Message to Initiator.	

Send Command	From: Gateway	To: Node
Initiated by:	Completion of downloading the update to the gateway	
Message Contents	● Enter bootloader command Command ● Node Name	
Expected Response:	4 Messages: ● Node Software Update Started ● MSP Software Updated ● FPGA Software Updated ● Node update completed	

Response to Initiator:	● Node Update Status (Percentage Completed) ● Node Software Version Number and Node Name
Receiving Damaged Messages:	Send Error Message to Initiator.
No Response Received	Wait for 15 seconds for response. After 30 seconds Send Error Message to Initiator.

Snap Photo



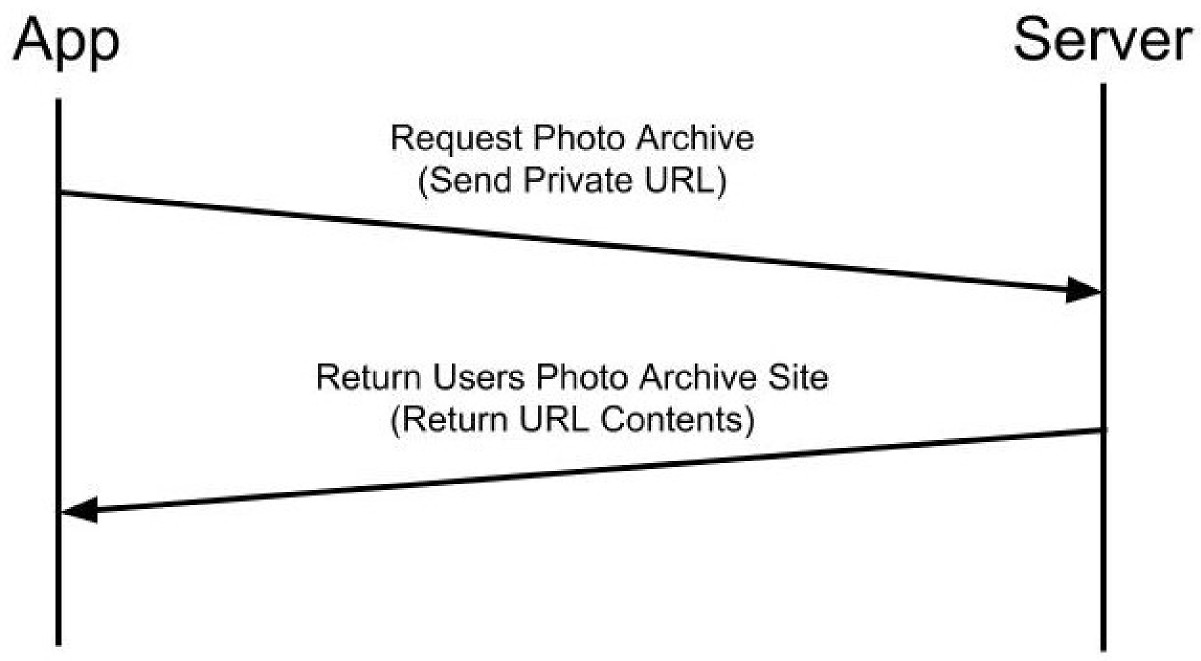
Capture Image	From: App	To: Gateway
Initiated by:	User presses “Snap Photo” button on screen	
Message Contents:	● Capture Image Command ● Gateway Name ● Node Name	
Expected Response:	3 Messages: ● Snap Photo Progress ● Photo received from camera ● Notification with date and time of photo	
Response to Initiator:	Upon receive the command a message will be sent stating the photo is being captured. When the photo is finished being captures it is sent.	
Receiving Damaged Messages:	Inform the user and give the option to try taking a photo again	
No Response Received	Wait for 10 seconds for a response from the server Inform the user about each missing response: “*Photo cannot be requested. Error Code: XX*”	

Take Photo Command	From: Gateway	To: Node
Initiated by:	Receiving Capture Image Command from App	

Message Contents	• Take Photo Command • Node Name
Expected Response:	1 Message: • Captured Photo
Response to Initiator:	• Photo contents • Photo checksum
Receiving Damaged photo:	Send Error Message to Initiator. Reinstate photo capture.
No Response Received	Send message to Initiator the Node is offline.

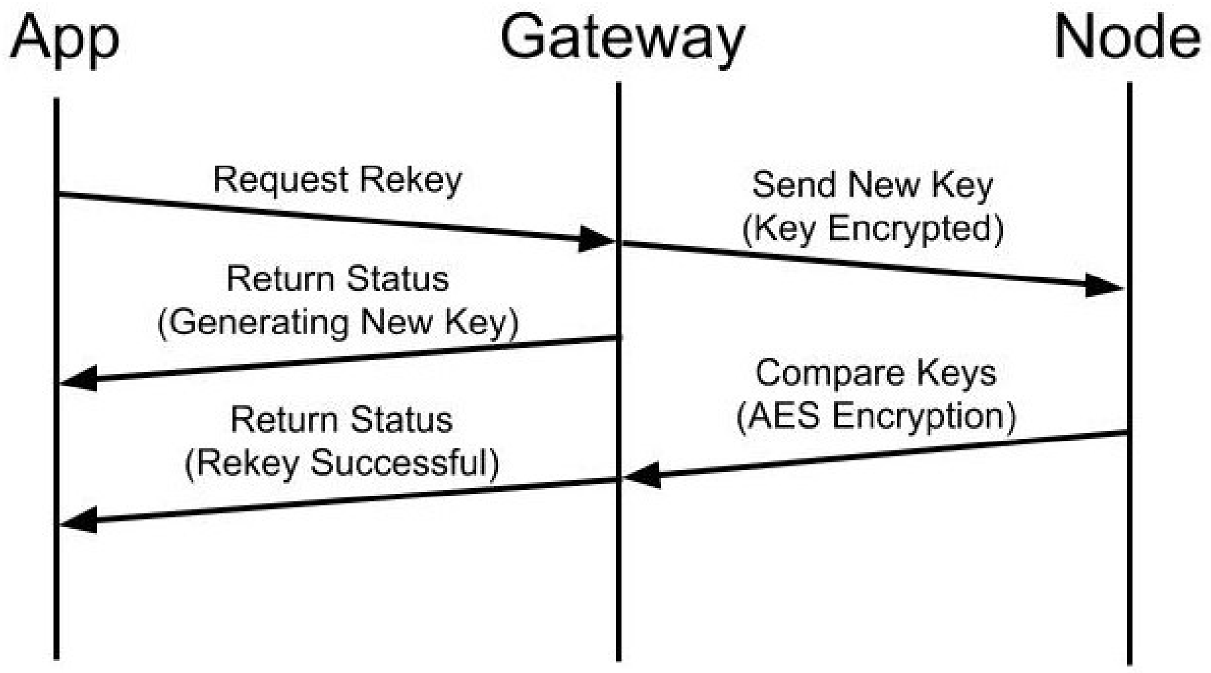
Send Photo	From: Gateway	To: Server
Initiated by:	Completion of sending the photo to the gateway	
Message Contents	• Photo Contents • Latest Pinged Server Name	
Expected Response:	1 Message: • Acknowledgement that Photo has been received	
Response to Initiator:	• Captured Photo • Data and time of photo taken	
Receiving Damaged Messages:	Send Error Message to Initiator.Send photo again to Server.	
No Response Received	Wait for 5 seconds for response. After 15 seconds Send Error Message to Initiator.	

Photo Archive



Request Photo Archive	From: App	To: Server
Initiated by:	User presses “Photo Archive” button on screen	
Message Contents:	● Send Private URL ● Latest Pinged Server Name	
Expected Response:	1 Message: ● HTML Website containing the library of photos	
Receiving Damaged Messages:	Inform the user and automatically try accessing the server again	
No Response Received	Wait for 5 seconds for a response from the server Inform the user about each missing response: “*Server if not available at this time, Trying Again*” After 30 seconds the App will stop requesting to access the server.	

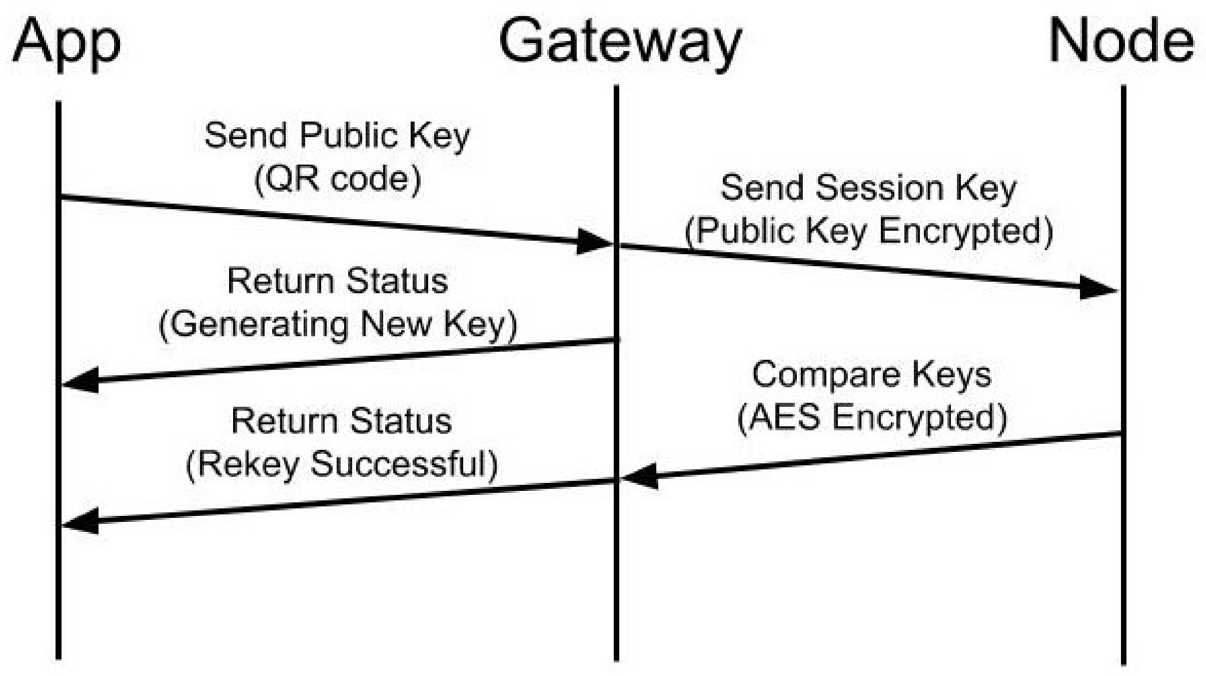
Regenerate Key



Request Rekey	From: App	To: Gateway
Initiated by:	User presses “Regenerate Key” button on screen	
Message Contents:	● Request Rekey Command ● Gateway Name ● Node Name	
Expected Response:	2 Messages: ● Rekey Progress ● Rekey Completion	
Response to Initiator:	When the gateway is generating a new key it will send a progress report to the app	
Receiving Damaged Messages:	Inform the user and give the option to try rekeying again	
No Response Received	Wait for 5 seconds for a response from the gateway Inform the user about each missing response: “*Rekey is not responsive, please try again*”	

Send New Key	From: Gateway	To: Node
Initiated by:	Receiving Regenerate Key from App	
Message Contents	• Encrypted Key session • Node Name	
Expected Response:	1 Message: • Key Session encrypted with AES encryption	
Response to Initiator:	• Node Rekey progress (Percentage Complete) • Recommended Instructions on interacting with Node (Do not unplug power from node or turn off app)	
Receiving Damaged Messages:	Send Error Message to Initiator.	
No Response Received	Wait every 2 seconds for progress update. After 10 seconds send Error Message to Initiator.	

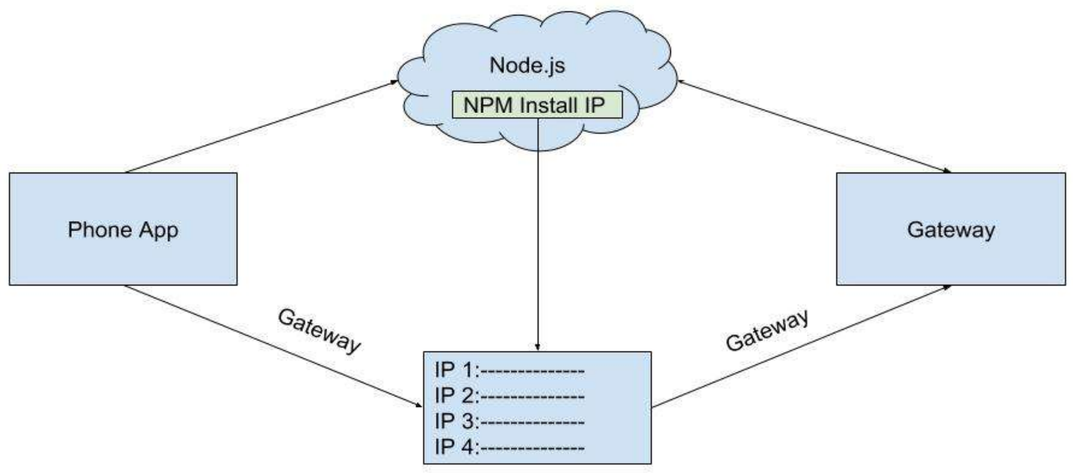
Add New Device


App
Gateway
Node
Send Public Key (QR code)
Send Session Key (Public Key Encrypted)
Return Status (Generating New Key)
Compare Keys (AES Encrypted)
Return Status (Rekey Successful)

Send Public Key	From: App	To: Gateway
Initiated by:	User presses “Link Device” button on screen	
Message Contents:	● Send Public Key Command ● Public Key ● Node Name	
Expected Response:	2 Messages: ● Key Progress ● Key Completion	
Response to Initiator:	While the gateway is encrypts the key session it will send a progress report to the app	
Receiving Damaged Messages:	Inform the user and give the option to try keying again	
No Response Received	Wait for 5 seconds for a response from the gateway Inform the user about each missing response: “*Key generation is not responsive, please try again*”	

Message Contents	● Encrypted Key session ● Node Name
Expected Response:	1 Message: ● Key Session encrypted with AES encryption
Response to Initiator:	● Node Keying progress (Percentage Complete) ● Recommended Instructions on interacting with Node (Do not unplug power from node or turn off app)
Receiving Damaged Messages:	Send Error Message to Initiator.
No Response Received	Wait every 2 seconds for progress update. After 10 seconds send Error Message to Initiator.

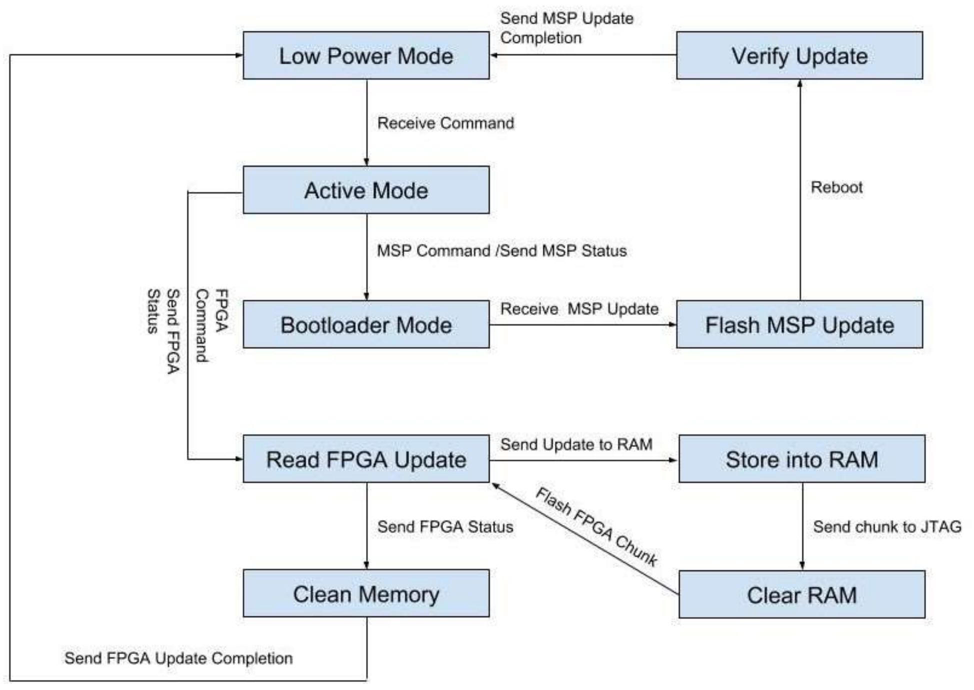
Get Gateway IP address



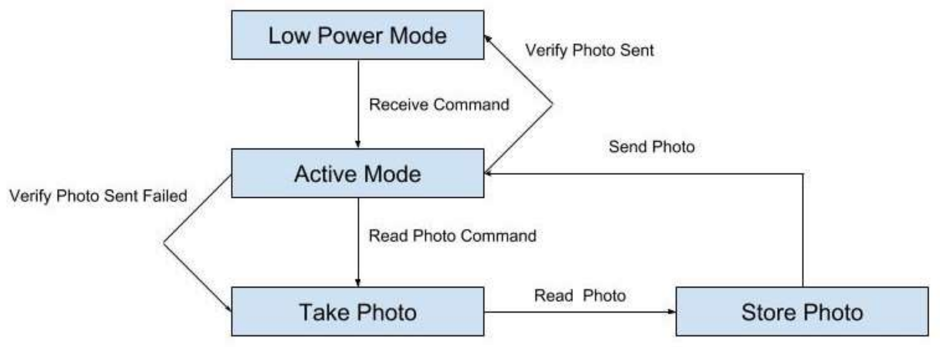
Request IP Address	From: App	To: Server
Initiated by:	Pre-Assigned time to ping without User input	
Message Contents:	● Request IP Address Command ● Router IP address ● Gateway Name	
Expected Response:	2 Messages: ● Gateway IP Address ● Router IP Address	
Response to Initiator:	The system sends no response	
Receiving Damaged Messages:	Retrying obtaining the IP Address	
No Response Received	Retrying obtaining the IP Address. Use the locally store ip to ping the gateway.	

Node State Diagram

MSP Update Process

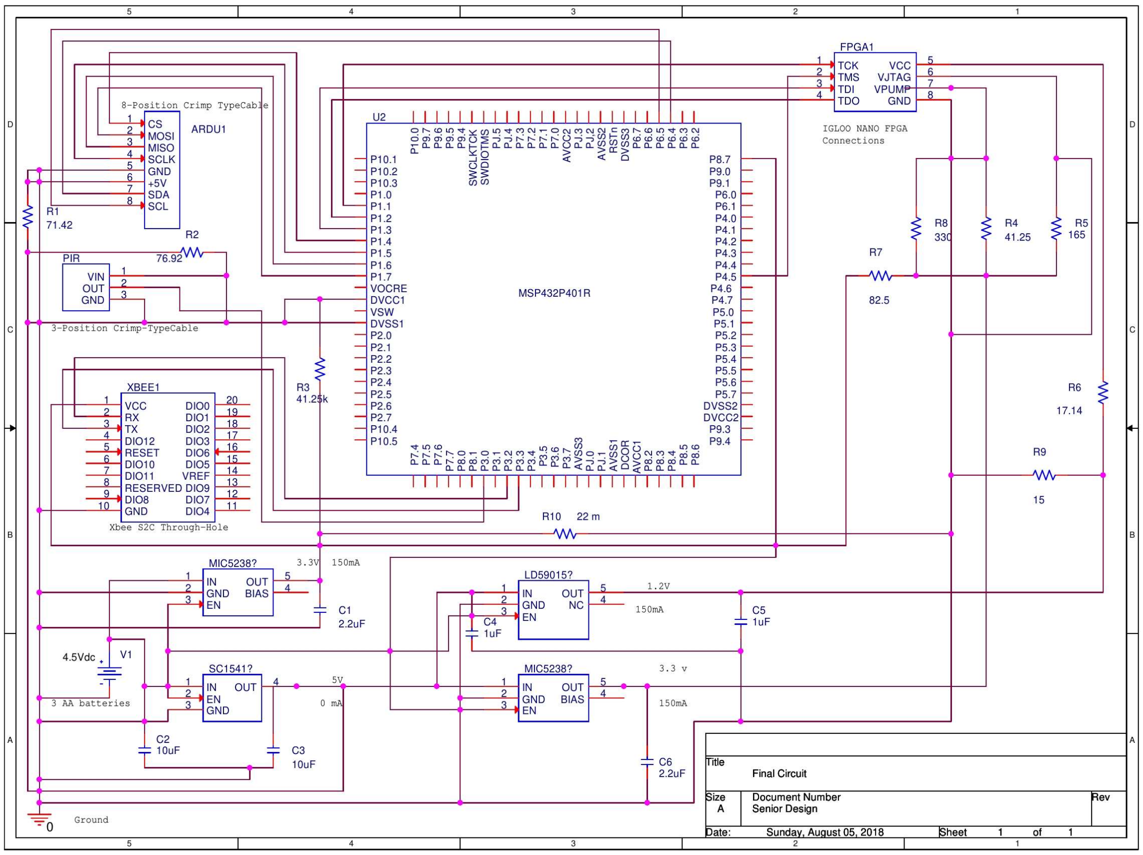


MSP Photo Process (Motion Detected/Request Photo)



Circuit Design

The following is a diagram of the circuit with Description of each connection



Xbee to MSP432 Connection:

Xbee S2C

Protocol:	UART
Speed:	9600 Hz
Operating Voltage:	3.3 V
ON/Off or Sleep:	Pin 1(VCC)
MSP432 Pins:	Pin 3.2 (RX) and 3.3 (TX)
Connector:	Shrouded Pin Header,20 Pins, 4 used Pins: GND, RX, TX, VCC
Process:	A character array of the photo data sent one byte at a time via the TX pin. The Xbee has built RTC protocols to check data transfer. It will keep sending a packet if a packet is lost while transferring.

Motion Sensor to MSP432 Connection:

HC-SR501 PIR Motion Detector

Operating Voltage:	3.3 V
ON/Off or Sleep:	Pin 1(VIN)
MSP432 Pins:	Pin 3.0 (GPIO)
Connector:	Shrouded Pin Header, 3 Pin: GND, VIN,OUT Connection: 3 Position Crimp cable between Motion Sensor and Shrouded Pin Header with Pin 2(OUT) connection to the MSP432
Process:	Upon detecting motion the device will send a high voltage signal to wake the MSP from low power mode and capture a photo.

ArduCamera to MSP432 Connection:

Arducam OV2640

Operating Voltage:	5.0 V
ON/Off or Sleep:	Pin 6.0(+5V)
MSP432 Pins:	PIN 6.5(Clock speed) for I2C (SCL) connects to the SCK pin of the camera PIN 6.4(Data transfer) for I2C(SDA) connects to the SDI pin of the camera PIN 1.4(Slave select) connects to the CS pin of the camera PIN 1.5(Clock) connects to the SCLK pin of the Camera PIN 1.6(Data pin) connects to the MISO pin of the Camera PIN 1.7(Data pin) connects to the MOSI pin of the Camera
Connector:	Shrouded Pin Header, 8 Pin: CS,MOSI,MISO,SCLK,GND,+5V,SDA,SCL Connection: 4-2-2(8) Position Crimp cable between ArduCam and Shrouded Pin Header
Process:	After taking a photo the MSP432 will receive the photo via the MISO pin and then construct character array of the photo one byte at a time while simultaneously calculating the size of the photo.
Software:	SPI with a clock rate of 1 MHz for photo data transfer I2C with a clock rate of 100 kHz for camera initialization and verification

FPGA to MSP432 Connection:

Microsemi IGLOO nano

Operating Voltage:	3.3 V
ON/Off or Sleep:	Pin 6.0(+5V)
MSP432 Pins:	PIN 6.5 is the Clock speed for I2C (SCL) it connects to the SCK pin of the FGPA PIN 6.4 is data transfer for I2C(SDA) and connects to the SDI pin of the FGPA PIN 1.4 is the Slave select pin (STE) and connects to the TMS pin of the FGPA PIN 1.1 is the CLK pin and connects to the TCK pin of the FGPA PIN 1.2 this a MISO data pin and connects to the TDO pin of the FGPA PIN 1.3 this a MOSI data pin and connects to the TDI pin of the FGPA
Connector:	Shrouded Pin Header, 8 Pin: TMS, TDO, TDI, TCLK, GND,+3.3V,SDA,SCL

	Connection: Onboard copper wire connection
Process:	After taking a photo the MSP432 will receive the photo via the MISO pin and then construct character array of the photo one byte at a time while simultaneously calculating the size of the photo.
Software:	SJTAG with a clock rate of 1 MHz for data transfer. Max frequency is 4 MHz

FPGA Electrical Specifications

Voltage	Current
Vcc = 1.2-1.5V	<70mA
Vjtag = 3.3V	<20mA
Vpump = 3.3V	<80mA

Vpump and Vjtag voltages are required for programming the FPGA.

Vjtag and Vcc voltages are required to read the output of the FPGA.

MSP432

The TI MSP432P401R microcontroller is used for currently is reading input from the motion sensor and telling the Arducam OV2640 to take a photo. The MSP432 proceeds to read the photo then send it the XBee module to be sent to the gateway which in turn will be sent to the server, it also reads decrypted keys from the FPGA. The MSP432 will also be used for updating the FPGA by asynchronously reading data received by the XBee module via UART, storing updates into a ram buffer and pushing the update data to FPGA via it's JTAG interface.
The MSP432 will also update itself by receiving a command to enter bootloader mode and then receive bootloader commands to read incoming data from the XBee module via UART and write to the MSP432's flash memory.

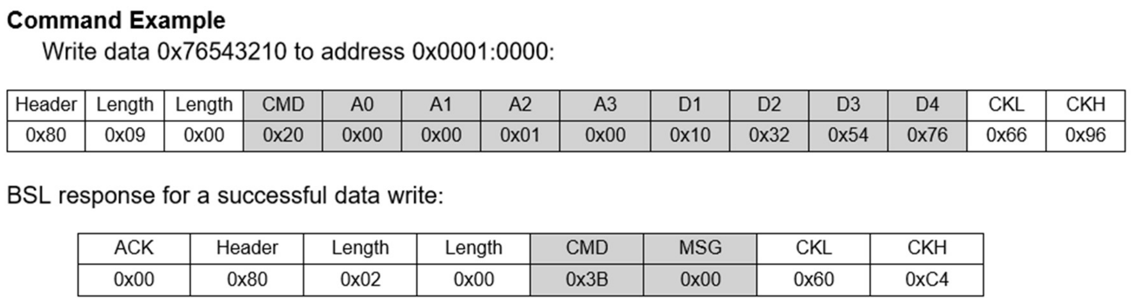
	Architecture	**Size**
1)	SRAM	64 KBytes
2)	Flash	256 Kbytes

Ram Usage: Photo Buffer: 30 KB Update Buffer: 30 KB Program SIze: 4KB
MSP432 Functions: There will be a total of 4 main functions for the device

- Key Device
    - It will receive a session key from the gateway and send it to the FPGA via I2C for decryption. It will the read the session key and encrypt with AES to be sent back.
- Snap Photo
    - It will initialize the camera via I2C and then take, calculate the size and send the compressed to the Xbee to be sent to the server.
- Update MSP432
    - The device will receive a signal to exit low power mode and enter bootloader mode. Bootloader will receive commands and the update file to write to flash
- Update FPGA
    - The device will receive a signal to exit low power mode and be on standby to receive update data to be placed into the ram buffer then sent to the FPGA

MSP432 Update package

The following is a breakdown of the what the MSP432 will receive once it enters bootloader mode. Each value is in hexadecimal format representing each byte. THe first byte is the header followed by the size of the update in bytes. The next byte is the command the specifies write or erase. This is followed by the address bytes that specifies the flash memory's address for where the update will be written. The next few bytes will be the binary file for the MSP432 followed by a checksum.



Command Example

Write data 0x76543210 to address 0x0001:0000:

Header	Length	Length	CMD	A0	A1	A2	A3	D1	D2	D3	D4	CKL	CKH
0x80	0x09	0x00	0x20	0x00	0x00	0x01	0x00	0x10	0x32	0x54	0x76	0x66	0x96

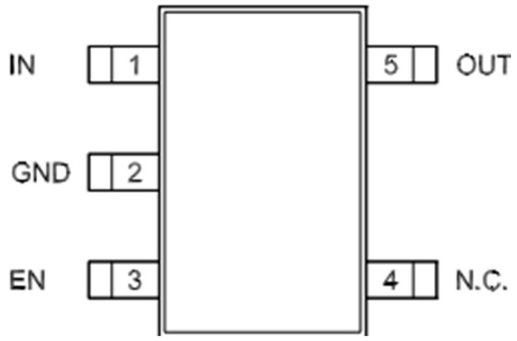
BSL response for a successful data write:

ACK	Header	Length	Length	CMD	MSG	CKL	CKH
0x00	0x80	0x02	0x00	0x3B	0x00	0x60	0xC4

FPGA
Igloo nano FPGA is used to implement the elliptic curve cryptography algorithm. There are 2 main reasons Igloo nano is being chosen. First is that the configuration memory is flash based and once programed, the FPGA holds its data and does not require to be programmed over and over again. The FPGA can be turned off after been programed and used for key verification or rekeying and can be used again without reconfiguring it. The second main reason is that it has a lower power consumption.

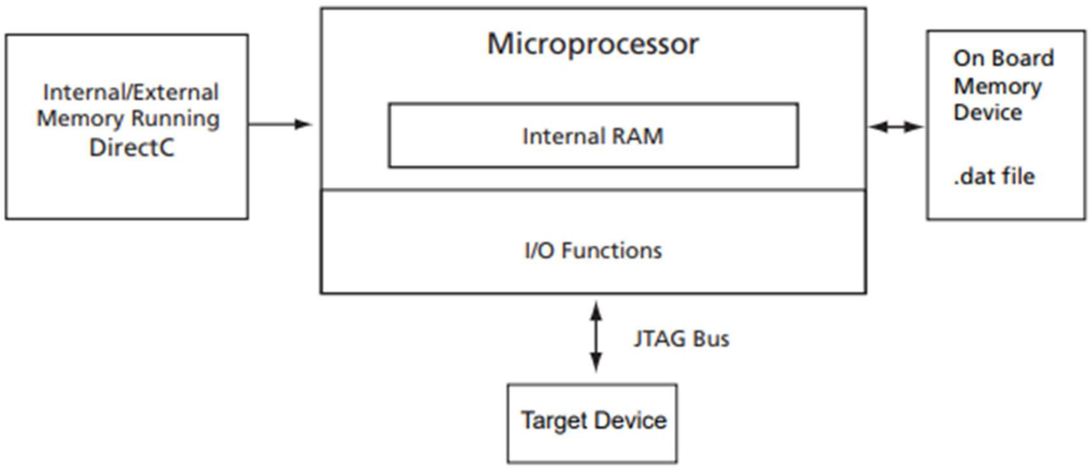
Architecture	**Size**
1. Flash Array or configuration memory	250000 System Gates 6144 Flip Flops
2. SRAM	1,024 bits
3. Flash	504 kbits

FPGA has different modes it can be operated however for this project purposes, it will only be in either ON mode implementing cryptography algorithm or being reconfigured or it will be in SHUTDOWN mode when it will be totally off. The operating modes of the FPGA are controlled using the voltage regulator.(LD59015).

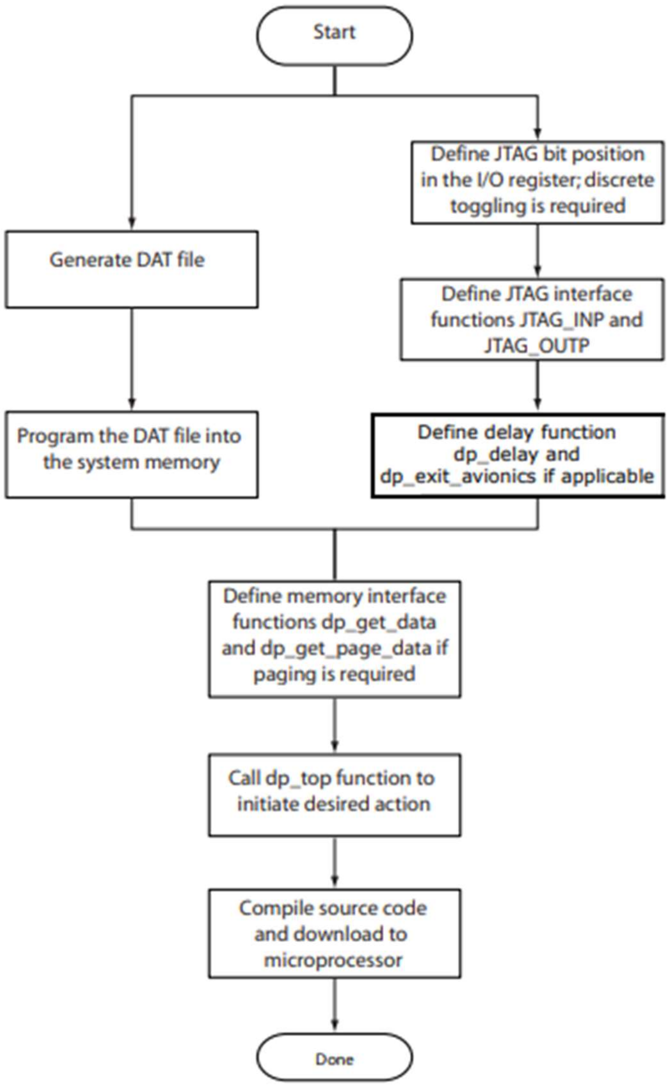


From the figure right above, it is seen that voltage regulator has 5 pins. Pin 4 is not used and not connected to anything. Pin EN is connected to microcontroller. Pin IN is connected to the batter and pin OUT is connected to FPGA. ON and OFF functionality (mode) of the FPGA will controlled by the microcontroller by sending an enable signal to the EN pin of the voltage regulator.

Programming or updating the FPGA
 Cryptography algorithm is given and will be implemented on the FPGA. The objective is to programme the FPGA using microcontroller MSP-432. The reason is a microcontroller is being used that it gives the capability to program and reprogram the FPGA over the air and the user is not required to bring the device to the factory. In system programming ISP technique is used to achieve this objective.
To perform In-System Programming (ISP) for the FPGA, JTAG interface is required which MSP-432 supports. Also access to the data file containing the programming data is also needed.
Diagram from the ISP Microsemi programming guide is shown below

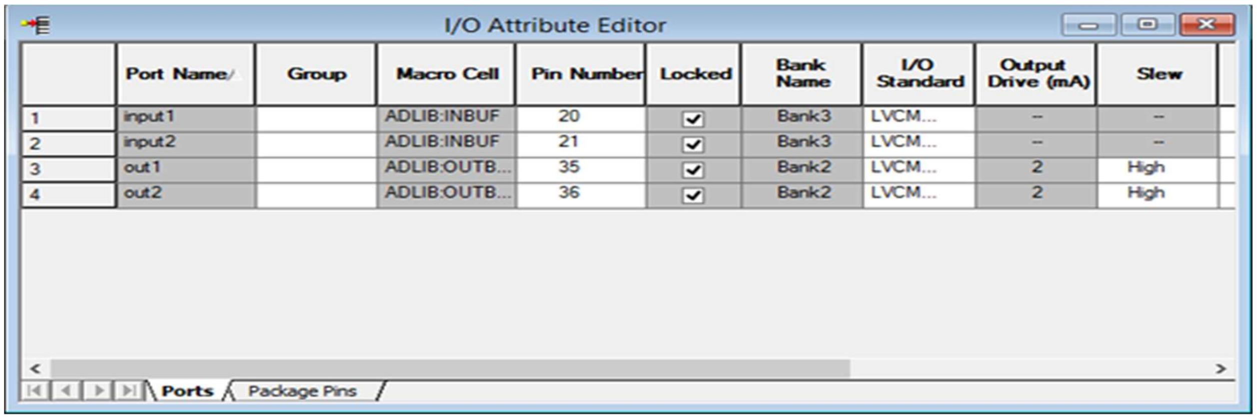


For ISP programming different files are either coded, edited or generated and guid of the tasks are shown in the following figure.


Start
Generate DAT file
Program the DAT file into the system memory
Define JTAG bit position in the I/O register; discrete toggling is required
Define JTAG interface functions JTAG_INP and JTAG_OUTP
Define delay function dp_delay and dp_exit_avionics if applicable
Define memory interface functions dp_get_data and dp_get_page_data if paging is required
Call dp_top function to initiate desired action
Compile source code and download to microprocessor
Done

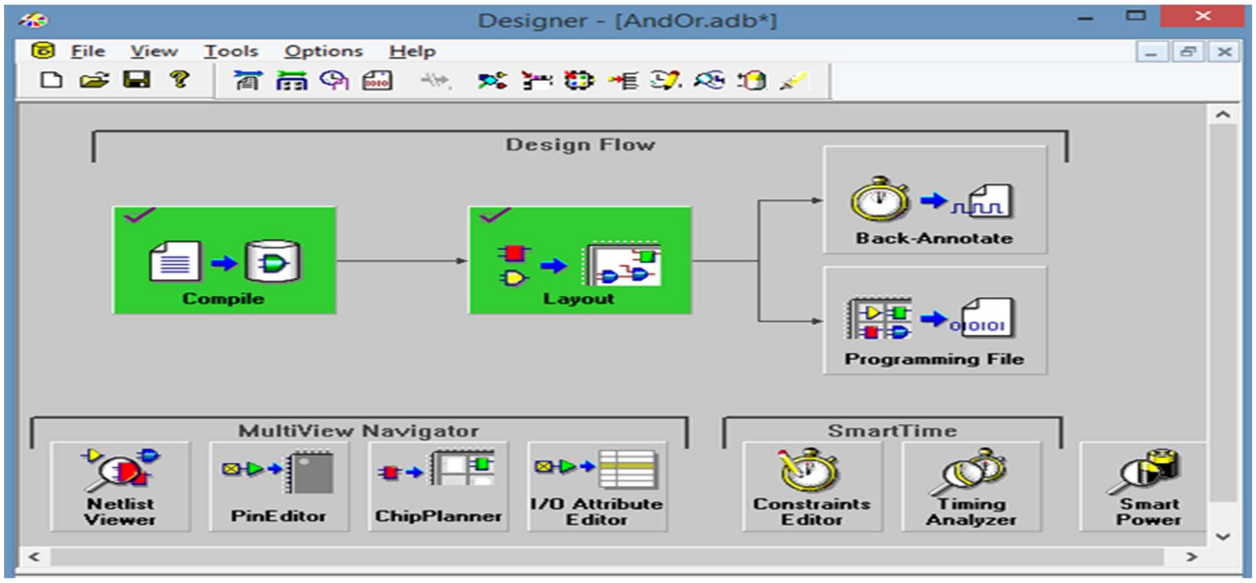
First step in ISP programming is to write the source code in VHDL using Libero SOC software provided by Microsemi. For this project, since the source code is given, cryptography algorithm will simply be uploaded. Once loaded the DAT file is generated. In order to do so first the source code is synthesized and compiled. That will generate a .PDC file. Using that in Designer software provided by Microsemi, following are the steps to create the DAT files.

1) First the inputs and outputs are assigned are assigned to the desired pins. There is no constraint file that needs to be edited. This is achieved by using another program called I/O Attribute editor.

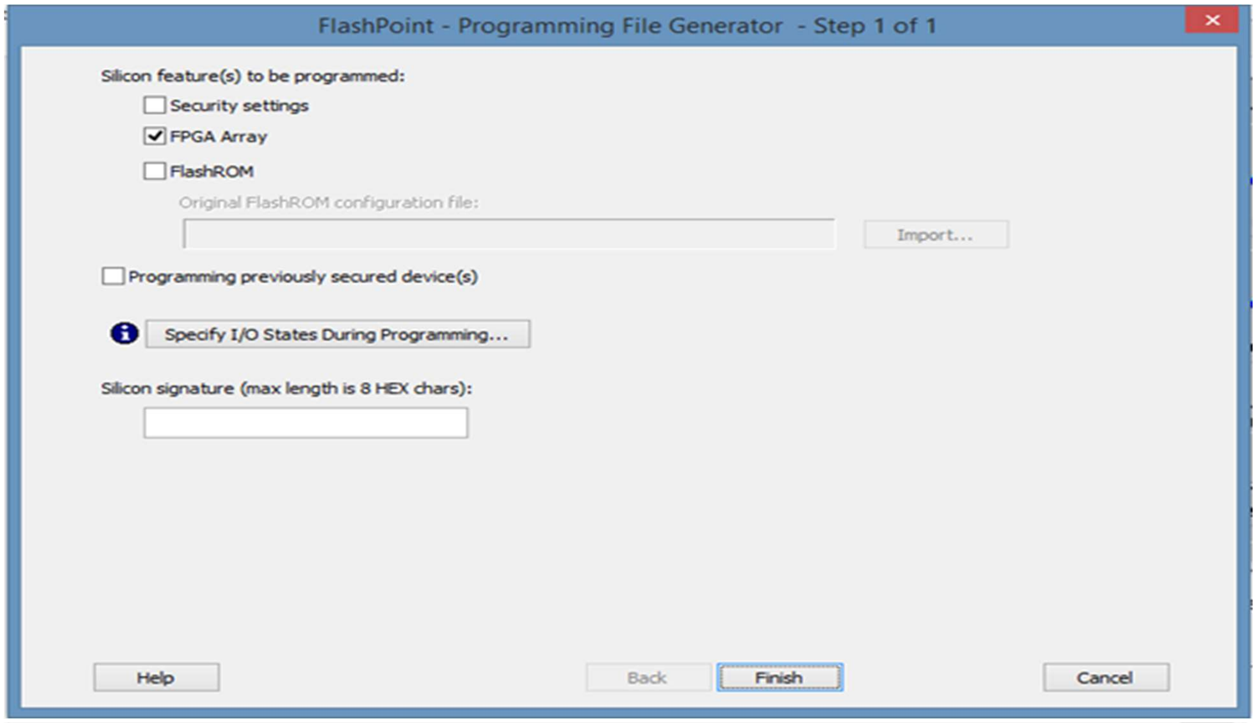


	Port Name	Group	Macro Cell	Pin Number	Locked	Bank Name	I/O Standard	Output Drive (mA)	Slew
1	input1		ADLIB:INBUF	20	✓	Bank3	LVCM...	--	--
2	input2		ADLIB:INBUF	21	✓	Bank3	LVCM...	--	--
3	out1		ADLIB:OUTB...	35	✓	Bank2	LVCM...	2	High
4	out2		ADLIB:OUTB...	36	✓	Bank2	LVCM...	2	High

Pin details are give in the user guide and following the pin details, Pin numbers are assigned. Once pins are assigned, layout is completed using designer tool.



Back-Annotate is the next step and that makes the output format to SDF. After that Programming files are generated. Once clicking Programming File, It gives the options of what needs to be programmed in the FPGA since Igloo Nano FPGA has flash memory also. It also gives the option to implement the AES security algorithm provided by Microsemi. That decrypts the data and keys are generated. Configuration memory is called FPGA Array in Designer tool.



FlashPoint - Programming File Generator - Step 1 of 1

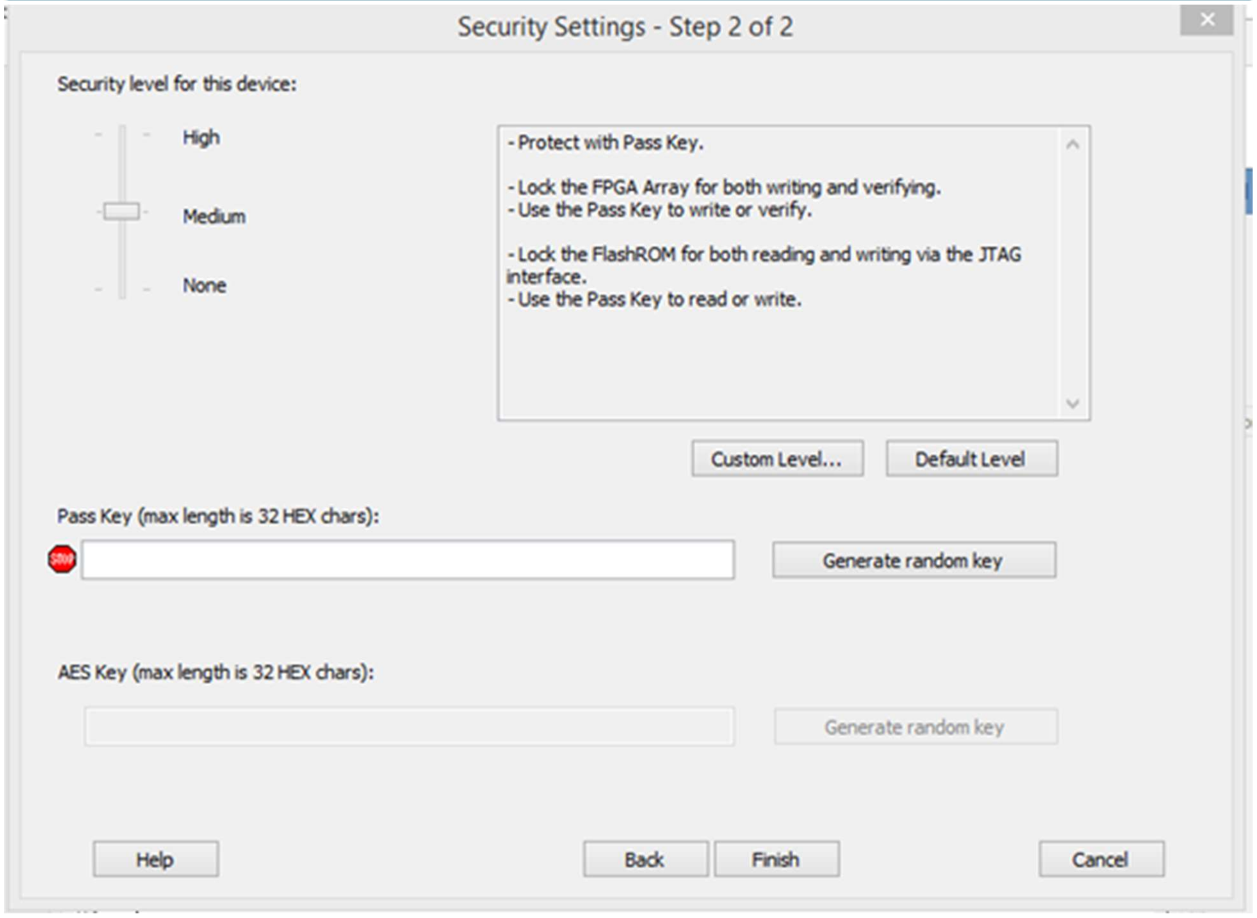
Silicon feature(s) to be programmed:

- [ ] Security settings
- [✓] FPGA Array
- [ ] FlashROM

Original FlashROM configuration file:

- [ ] Programming previously secured device(s)

Silicon signature (max length is 8 HEX chars):



Security Settings - Step 2 of 2

Security level for this device:

High
[✓ Medium]
None

- Protect with Pass Key.

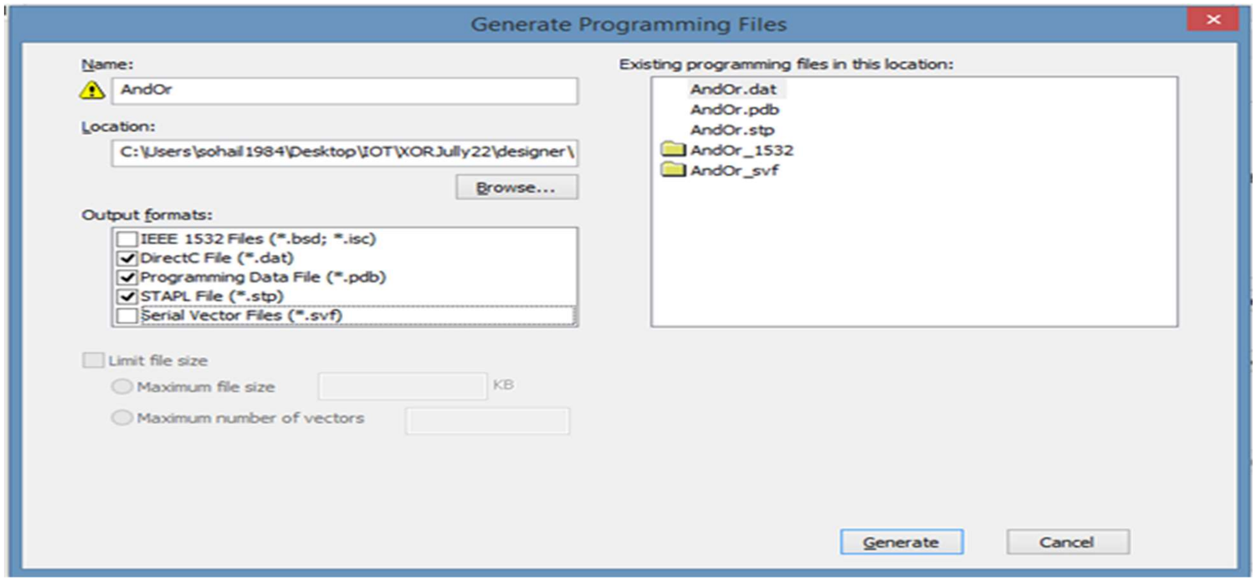
- Lock the FPGA Array for both writing and verifying.
- Use the Pass Key to write or verify.

- Lock the FlashROM for both reading and writing via the JTAG interface.
- Use the Pass Key to read or write.

Pass Key (max length is 32 HEX chars):

AES Key (max length is 32 HEX chars):

After the security settings ,the designer tool gives the option to choose the required files that need to be generated. For our ISP purposes, .DAT file and .PDB files are needed. STPL file is generated to program the FPGA without a microcontroller.


Generate Programming Files

Name: AndOr

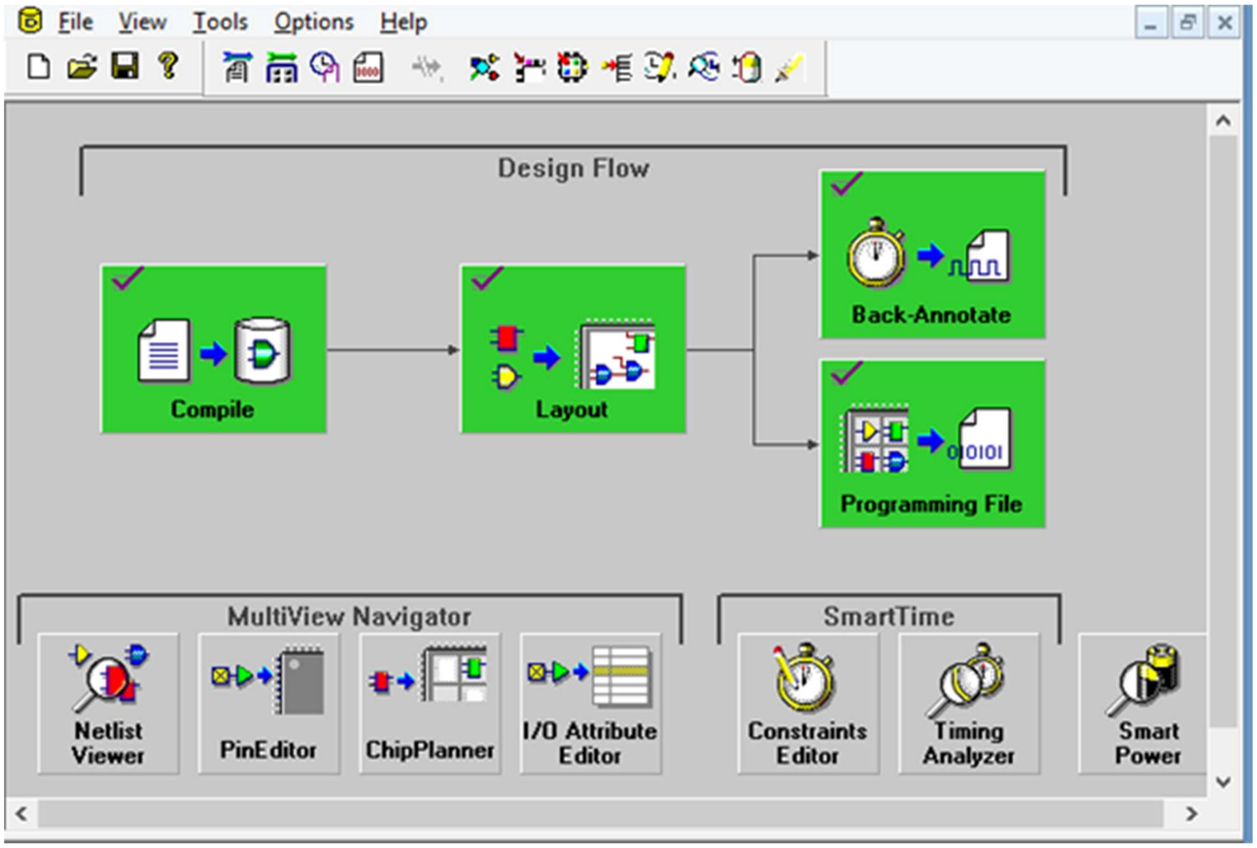
Location: C:\Users\sohail1984\Desktop\IOT\XORJully22\designer\

Output formats:
- [ ] IEEE 1532 Files (*.bsd; *.isc)
- [x] DirectC File (*.dat)
- [x] Programming Data File (*.pdb)
- [x] STAPL File (*.stp)
- [ ] Serial Vector Files (*.svf)

- [ ] Limit file size
  - Maximum file size KB
  - Maximum number of vectors

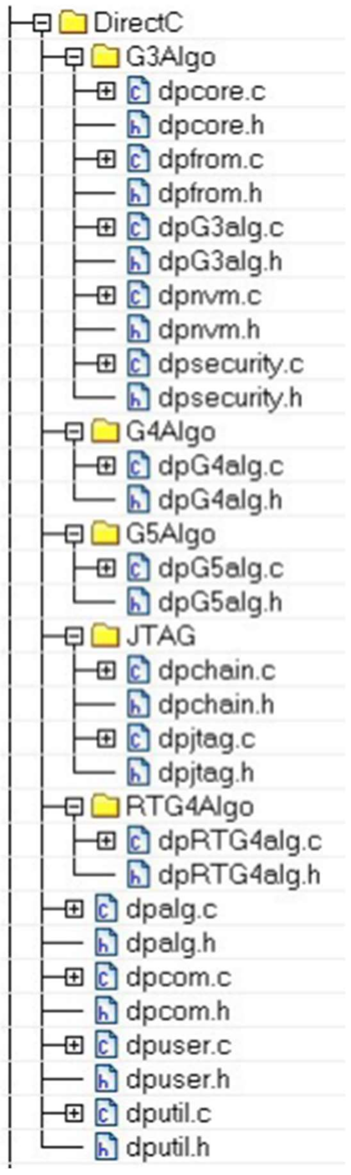
Existing programming files in this location:
- AndOr.dat
- AndOr.pdb
- AndOr.stp
- AndOr_1532
- AndOr_svf

Once the files are generated successfully the Programming File icon becomes green.



The generated files are in the impl1 folder in the designer folder of the project. The DAT file generated is stored in the storage memory of the microcontroller.

Once the .DAT file is generated the next step is to work on the DirectC files. These are C based functions. DirectC files are imported to code composer studio(MSP 432 programming software).



```
DirectC
├─ G3Algo
│  ├─ dpcore.c
│  ├─ dpcore.h
│  ├─ dpfrom.c
│  ├─ dpfrom.h
│  ├─ dpG3alg.c
│  ├─ dpG3alg.h
│  ├─ dpnvm.c
│  ├─ dpnvm.h
│  ├─ dpsecurity.c
│  └─ dpsecurity.h
├─ G4Algo
│  ├─ dpG4alg.c
│  └─ dpG4alg.h
├─ G5Algo
│  ├─ dpG5alg.c
│  └─ dpG5alg.h
├─ JTAG
│  ├─ dpchain.c
│  ├─ dpchain.h
│  ├─ dpjtag.c
│  └─ dpjtag.h
├─ RTG4Algo
│  ├─ dpRTG4alg.c
│  └─ dpRTG4alg.h
├─ dpalg.c
├─ dpalg.h
├─ dpcom.c
├─ dpcom.h
├─ dpuser.c
├─ dpuser.h
├─ dputil.c
└─ dputil.h
```

DirectC code is needed to be modified . First JTAG pin bit locations are defined in the I/O register. API is added to support discrete toggling of the individual JTAG pins. The hardware interface functions (jtag_inp and jtag_outp) are modified to use the hardware API functions designed to control the JTAG port. The delay function (dp_delay) is modified. Memory access functions are edited to access the data blocks. The dp_top function with the action code desired is created. The source code is compiled. This creates a binary executable that is downloaded to the microcontroller for execution.

Functions jtag_inp and jtag_outp are as follows.

```
DPUCHAR jtag_inp(void)
```

```
 {
    DPUCHAR tdo = 0u;
    DPUCHAR ret = 0x80u;


    GPIO_SetDir(JTAG_TDOPORT, JTAG_TDO, 0);


    tdo = (GPIO_ReadValue(JTAG_TDOPORT) >> JTAG_TDOPIN) & 1;


    if (tdo)
       ret = 0x80;
    else
       ret = 0;


    return ret;
 }


void jtag_outp(DPUCHAR outdata)
 {
    GPIO_SetDir(JTAG_TDOPORT, JTAG_TDO, 1);


    if(outdata & TCK)
       GPIO_SetValue(JTAG_TCKPORT, JTAG_TCK);
    else
       GPIO_ClearValue(JTAG_TCKPORT, JTAG_TCK);


    if(outdata & TDI)
       GPIO_SetValue(JTAG_TDIPORT, JTAG_TDI);
    else
       GPIO_ClearValue(JTAG_TDIPORT, JTAG_TDI);


    if(outdata & TMS)
       GPIO_SetValue(JTAG_TMSPORT, JTAG_TMS);
    else
       GPIO_ClearValue(JTAG_TMSPORT, JTAG_TMS);


    if(outdata & TRST)
       GPIO_SetValue(JTAG_TRSTPORT, JTAG_TRST);
    else
       GPIO_ClearValue(JTAG_TRSTPORT, JTAG_TRST);


    if(outdata & TDO)
       GPIO_SetValue(JTAG_TDOPORT, JTAG_TDO);
    else
       GPIO_ClearValue(JTAG_TDOPORT, JTAG_TDO);
```

```
                    return;
                }
```

Delay function is not complete but following is the template

```
void dp_delay(DPULONG microseconds)
{
    volatile DPULONG i;
    volatile DPULONG j;

    #error "still working on the modification."

    for(i=0;i<microseconds;i++) {
            for (j=0;j<50;j++) ;
        }
}
```

```
void dp_get_page_data(DPULONG image_requested_address)
{
    #ifdef ENABLE_EMBEDDED_SUPPORT
    return_bytes = PAGE_BUFFER_SIZE;
       if (image_requested_address + return_bytes > image_size)
    {
        return_bytes = image_size - image_requested_address;
    }


    #endif
    return;
}
```

Prototyping progress report:

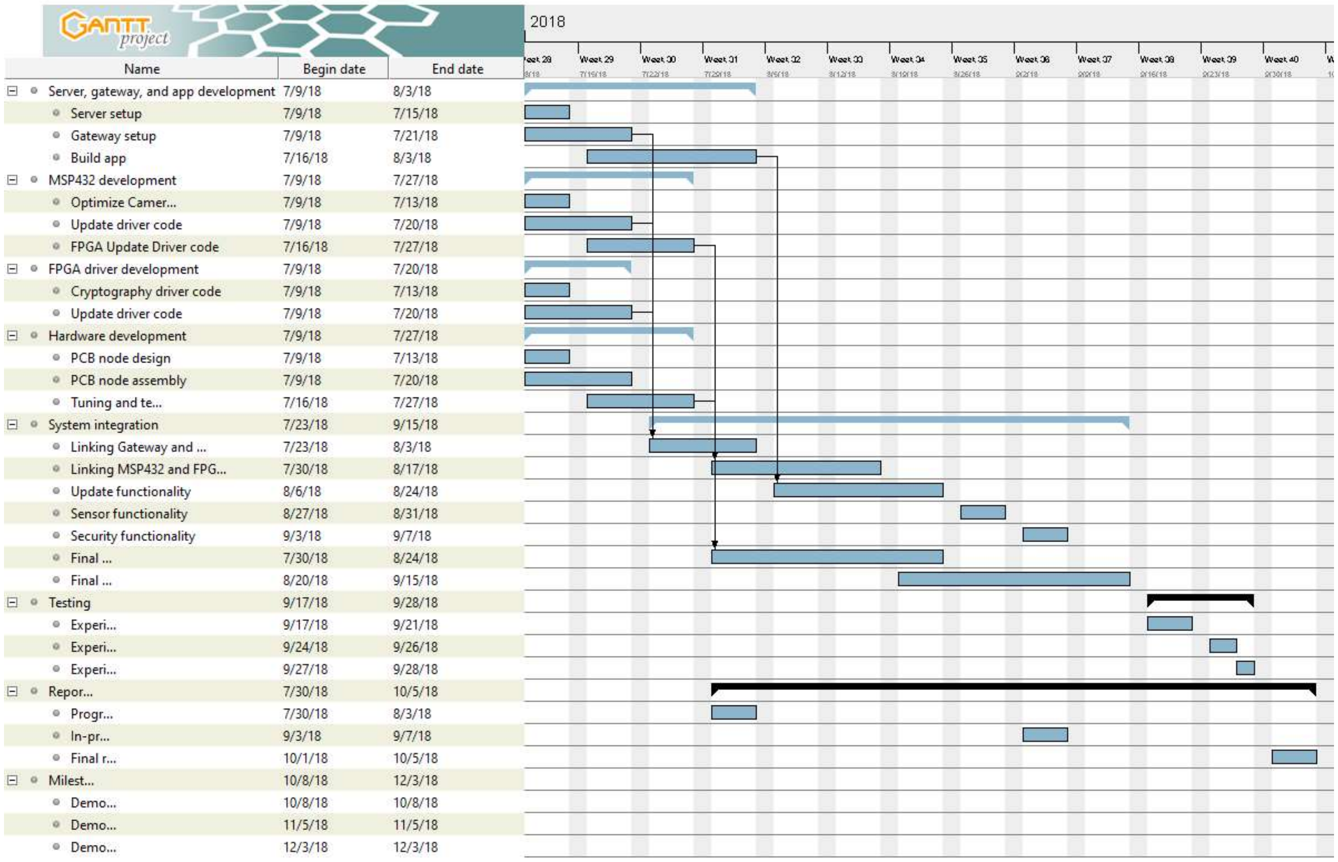
- Acquired Components
  - MSP432P401R Microcontroller
  - USB to RS-232 Interface device
  - Microsemi IGLOO nano FPGA
  - Raspberry Pi 3 rev. B
- Experiments
  - Attempt to program the MSP432 to enter bootloader mode via UART and then send bootloader commands and an the update file for flashing.
  - Learn to program the IGLOO nano
  - Send a signal from a phone application to a Raspberry Pi to turn on an led.
- Lessons learned
  - Even with the mastery of each piece of software on the device the biggest challenges are debugging the hardware interfaces of each device

Testing plan :

- Data verification
  - Most of the experiments done were primarily to validate whether we could successfully send bytes of the data from various types of devices that all had different hardware interfaces.
  - Success was measured mainly on if we were able to successfully send a character to a different device without the character changing when reading on a different device. Upon verification we could implement sending data and commands to write programs or enable devices.

Complete List of Tasks

1. Server, gateway, and app development (3 weeks)
    1.1) Server setup
    1.2) Gateway setup
    1.3) Build app
2. MSP432 development (3 weeks)
    2.1) Optimize Camera driver code
    2.2) Update driver code
    2.3) FPGA Update Driver code
3. FPGA driver development (3 weeks)
    3.1) Cryptography driver code
    3.2) Update driver code
4. Hardware development (3 weeks)
    4.1) PCB node design
    4.2) PCB node assembly
    4.3) Tuning and testing
5. System integration (8 weeks)
    5.1) Linking Gateway and MSP432 via Xbee
    5.2) Linking MSP432 and FPGA systems
    5.3) Update functionality
    5.4) Sensor functionality
    5.5) Security functionality
    5.6) Final PCB tuning
    5.7) Node PCB test
6. Testing (3 weeks)
    6.1) Experiment #1
    6.2) Experiment #2
    6.3) Experiment #3
7. Reporting
    7.1) Progress report
    7.2) In-progress presentation
    7.3) Final report
8. Milestones/Demos
    8.1) Demo #1
    8.2) Demo #2
    8.3) Demo #3



Name	Begin date	End date
Server, gateway, and app development	7/9/18	8/3/18
Server setup	7/9/18	7/15/18
Gateway setup	7/9/18	7/21/18
Build app	7/16/18	8/3/18
MSP432 development	7/9/18	7/27/18
Optimize Camer...	7/9/18	7/13/18
Update driver code	7/9/18	7/20/18
FPGA Update Driver code	7/16/18	7/27/18
FPGA driver development	7/9/18	7/20/18
Cryptography driver code	7/9/18	7/13/18
Update driver code	7/9/18	7/20/18
Hardware development	7/9/18	7/27/18
PCB node design	7/9/18	7/13/18
PCB node assembly	7/9/18	7/20/18
Tuning and te...	7/16/18	7/27/18
System integration	7/23/18	9/15/18
Linking Gateway and ...	7/23/18	8/3/18
Linking MSP432 and FPG...	7/30/18	8/17/18
Update functionality	8/6/18	8/24/18
Sensor functionality	8/27/18	8/31/18
Security functionality	9/3/18	9/7/18
Final ...	7/30/18	8/24/18
Final ...	8/20/18	9/15/18
Testing	9/17/18	9/28/18
Experi...	9/17/18	9/21/18
Experi...	9/24/18	9/26/18
Experi...	9/27/18	9/28/18
Repor...	7/30/18	10/5/18
Progr...	7/30/18	8/3/18
In-pr...	9/3/18	9/7/18
Final r...	10/1/18	10/5/18
Milest...	10/8/18	12/3/18
Demo...	10/8/18	10/8/18
Demo...	11/5/18	11/5/18
Demo...	12/3/18	12/3/18

12) Appendix C: Software printout

- MSP Related Files (in BSL Folder)
  - Flashmailbox.c contains functions to configure the bootloader for UART
  - Main.c contains functions to handle updates via bootloader, handle FPGA update, well as take and send photos through Xbee via UART
  - I2c_driver.c contains functions for configure the camera for JPEG compression
  - Spi_driver.c contains the functions to receive pictures from the camera
  - Msp432_flashmailbox.c contains the bootloader configuration
- FPGA Related Files (in BSL Folder)
  - dpuser.c contains the JTAG pin interfacing to MSP432
  - dcom.c contains the code to interface with MSP432 memory (RAM)
  - dpalg.c Contains the function dp_top() that programs the FPGA
  - Any other files beginning with dp are for FPGA device support
- PCB Related Files
  - OTA Updates for an IOT Security Device.pdf contains the pdf file of the schematics
  - OTA Updates for an IOT Security Device.pro contains the project
  - Any files with .sch contain the hierarchical sheets schematic
  - Any files with .lib contain libraries to different footprints
  - OTA Updates for an IOT Security Device.net contains the netlist
  - PCB folder contains files for the manufacturing of the PCB
  - Footprints folder contains additional footprints for the PCB
  - Any files with .bak , .bck , .dmc , .cmp are files used by Kicad to support the PCB