Low Power Smart Outlet

Thread based communication

Project report for

Low Power Embedded System Design Techniques

By

Surjith Bhagavath Singh

Chinmay Hitesh Shah

Table of Contents

Introduction

Low Power Smart Outlet is a power outlet with few sensors and a switch/relay. Aim of the project is to make use of Thread protocol to talk to the end devices and be connected to the internet at the same time. This is a small puck like device which can be attached in all the sockets/appliances would be a better solution.

The goal of the project is to design and develop the hardware (Power Supply, AC Actuation system, Sensor interfaces) , firmware (to talk to all these basic peripherals) and software (Mobile/Linux Application)

The product has been designed to operate in low power condition. The parts are meticulously chosen to have a low power solution.

End- devices were developed during fall-2016. For prototyping purpose, a raspberry pi has been used as a server with CEL USB thread module.

Hardware – Overview

Our primary hardware has three parts

- Power Supply (LTC3588)
- Relay Circuitry (MOSFET Actuation)
- Sensor Interface
- Reset Circuitry
- Wireless EFR32MG Module Interface

Power Supply

This project needed a power supply that needs high efficient rectifier, buck converter and capabilities to have an energy backup when the AC Power goes down. LTC3588(Linear Tech) had a PMIC which matches the criteria that has been mentioned above.

A supercapacitor has been designed to make the device stay powered for a day even when there is no AC source available. The reason for this feature is for future implementation of voice control even when there is a power outage.

It has interface with energy harvesting unit as well a bridge rectifier, to connect directly into the AC supply. It has an inbuilt buck converter to boost up the current (To transfer energy from capacitor / battery). It has a continuous current specification of 100mA and Peak current specification of 300mA.

Main load is EFR32MG Thread SoC (127mA – Max according to datasheet on transmitting) and Relay (40mA – Max at 3.3V according to datasheet) and Si7021 sensor.

Transformer(115v/10v) is used to facilitate the maximum input rating of 10V AC. Refer Schematics for details.[1]

[1] Add Schematics Link

Relay Circuitry

Normal relays have just two states ON and OFF, it does not have memory to store the previous state of the relay. In real world scenario, it is needed to have some sort of mechanism to store the previous state of the switch and stay at that state until the user changes its state. (i.e. Users are used to have a manual switch and it has to be turned off whenever they needed it to be turned off or when there is an abnormal case). The reason we did not use a normal relay is because it needs constant continuous current to stay at ON state, In order to operate in a low power condition we had to choose an optimal solution, A latched relay.

A latching relay is a two-position electrically-actuated switch. It is controlled by two momentary-acting switches or sensors, one that 'sets' the relay, and the other 'resets' the relay. The latching relay maintains its position after the actuating switch has been released, so it performs a basic memory function.[2]

This relay just needs a pulse to turn the state to on and off (Set and Reset). This makes the device operate under low power.

MOSFET Actuation circuit has been used to actuate the relay. Refer Schematics for details. MOSFET's have very low leakage current compared to BJTs.

The relay control signals are given from EFR32MG. PF4 and PF5 are connected to Relay Control pins.

Sensor Interface

Si7021 is an I2C based sensor. It has both temperature and Humidity. The reason we chose this sensor is because it just needs 7-8mA for sensing the data.

This sensor has been attached to PC10(SCL) and PC11(SDA). These sensors are low power consuming and it has both data humidity and temperature readings which would be useful for us in the next iteration of the product to control the entire home environment.

It can operate at a range of -10 to 85 degree Celsius, and at 0-100% Relative humidity. It can operate at a voltage range of 1.9 to 3.6 V, which is very optimal for our use case.

Reset Circuitry

Reset circuitry has been interfaced directly with the PGOOD Signal from LTC3588. This will reset the MCU when there is a power outage and super capacitor discharges. A push button has been interfaced to reset pin, in case for debugging purposes.

EFR32MG (Mighty Gecko)

Mighty Gecko is the brain of the entire product. It has Radio inbuilt on it. It supports thread stack. It has 802.15.4 radio and an MCU. It supports Bluetooth, RAIL. The sensor has been interfaced through I2C of EFR32MG. Relay Control signals are also interfaced with GPIO Pins.

This chip has integrated ZigBee, Bluetooth Smart, Thread and proprietary radio on single SoC allowing us multiple choice of radio . We were planning to use ZigBee for proof of concept (Eventually we implemented thread) and later venture out using other radios for future implementation.

[2] https://www.azatrax.com/latching-relay-circuits.html

Silicon labs tools and their chip's low power operation is one more reason for choosing this chip. Support from silicon labs forum was of tremendous help throughout the project. Hardware Abstraction Layer(HAL) was of good help making our lives easier in integrating the peripherals.

Power partitioning is available. Energy Management unit and ultra-low power timer are also available which can be experimented for lower power. Since the project needs I2C for communication with external sensors and I2C is available in all energy modes other than EM4. Linked DMA is available till EM3 helping communication from radio.

Silicon Labs sample codes helped a lot in the start of the project. It was not having a proper documentation though. It helped us to a level to kickstart the project.

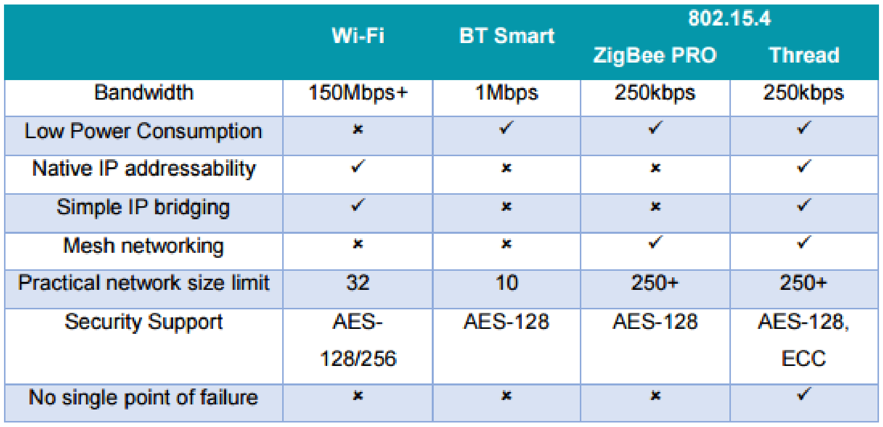
Why Thread?[3]

The Thread stack is an open standard for reliable, cost-effective, low-power, wireless D2D (device-to-device) communication. It is designed specifically for Connected Home applications where IP-based networking is desired and a variety of application layers can be used on the stack. These are the general characteristics of the Thread stack and network:

- Simple network installation, start up and operation: The simple protocols for forming, joining, and maintaining Thread Networks allow systems to self-configure and fix routing problems as they occur.
- Secure: Devices do not join the Thread Network unless authorized and all communications are encrypted and secure.
- Small and large networks: Home networks vary from several devices to hundreds of devices communicating seamlessly. The network layer is designed to optimize the network operation based on the expected use.
- Range: Typical devices in conjunction with mesh networking provide sufficient range to cover a normal home. Spread spectrum technology is used at the physical layer to provide good immunity to interference.
- No single point of failure: The stack is designed to provide secure and reliable operations even with the failure or loss of individual devices.
- Low power: Host devices can typically operate for several years on AA type batteries using suitable duty cycles.

Thread protocol has been widely accepted by the industry because of the properties mentioned above. Google Nest home automation devices, Samsung Artik IoT Home Automation devices are the early users of this protocol and they are one of the founding partners of this protocol.

[3] http://threadgroup.org/Portals/0/documents/whitepapers/Thread%20Stack%20Fundamentals_v2_public.pdf



	Wi-Fi	BT Smart	802.15.4	
			ZigBee PRO	Thread
Bandwidth	150Mbps+	1Mbps	250kbps	250kbps
Low Power Consumption	×	✓	✓	✓
Native IP addressability	✓	×	×	✓
Simple IP bridging	✓	×	×	✓
Mesh networking	×	×	✓	✓
Practical network size limit	32	10	250+	250+
Security Support	AES-128/256	AES-128	AES-128	AES-128, ECC
No single point of failure	×	×	×	✓

Figure 1: Thread comparison with other wireless technologies

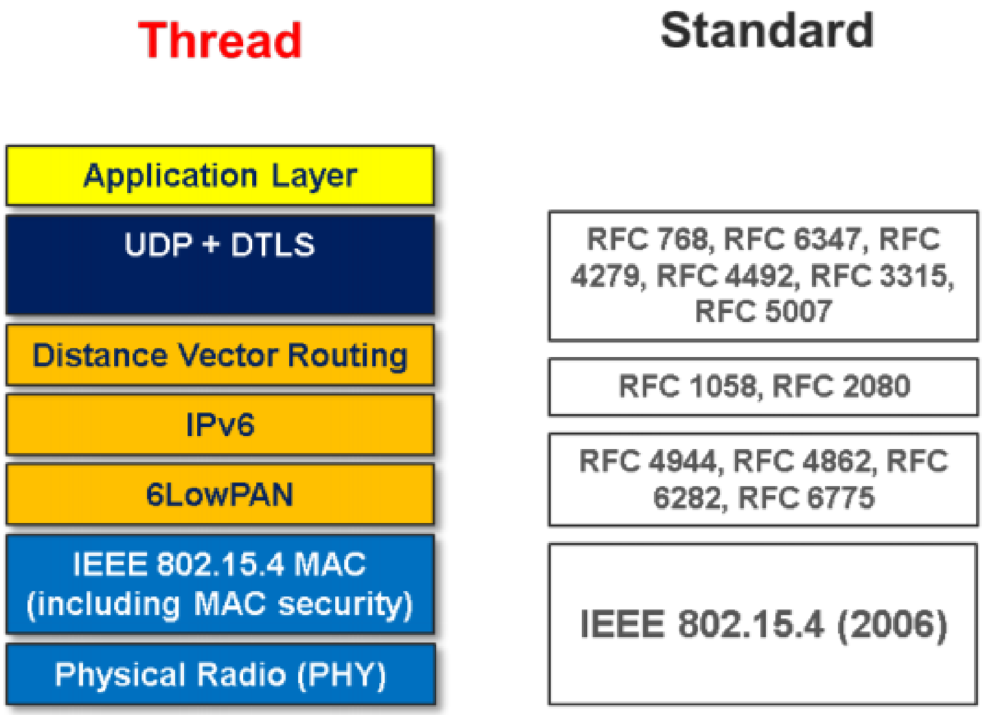

Thread
Standard
Application Layer
UDP + DTLS
Distance Vector Routing
IPv6
6LowPAN
IEEE 802.15.4 MAC (including MAC security)
Physical Radio (PHY)
RFC 768, RFC 6347, RFC 4279, RFC 4492, RFC 3315, RFC 5007
RFC 1058, RFC 2080
RFC 4944, RFC 4862, RFC 6282, RFC 6775
IEEE 802.15.4 (2006)

Figure 2: Thread Stack[4]

[4] http://threadgroup.org/Portals/0/documents/whitepapers/Thread%20Stack%20Fundamentals_v2_public.pdf

Schematics and Layout

Altium has been used for designing the board. AC part has been isolated using transformer for safety purposes. Altium tool was helpful throughout the process. Few packages for the devices are not available, we had to make it on our own. Altium libraries are of great help though. The process of layout is a bit of a work, iterating over and over to have an efficient use of board space has been achieved.

For prototyping purpose debug ports, LED Indicators , Test points were included in the layout. AC Power part is occupying most of the space in the board (This is for safety reasons in the first prototype). In the next iteration, it will be designed to match with the UL Standards.

Block Diagram

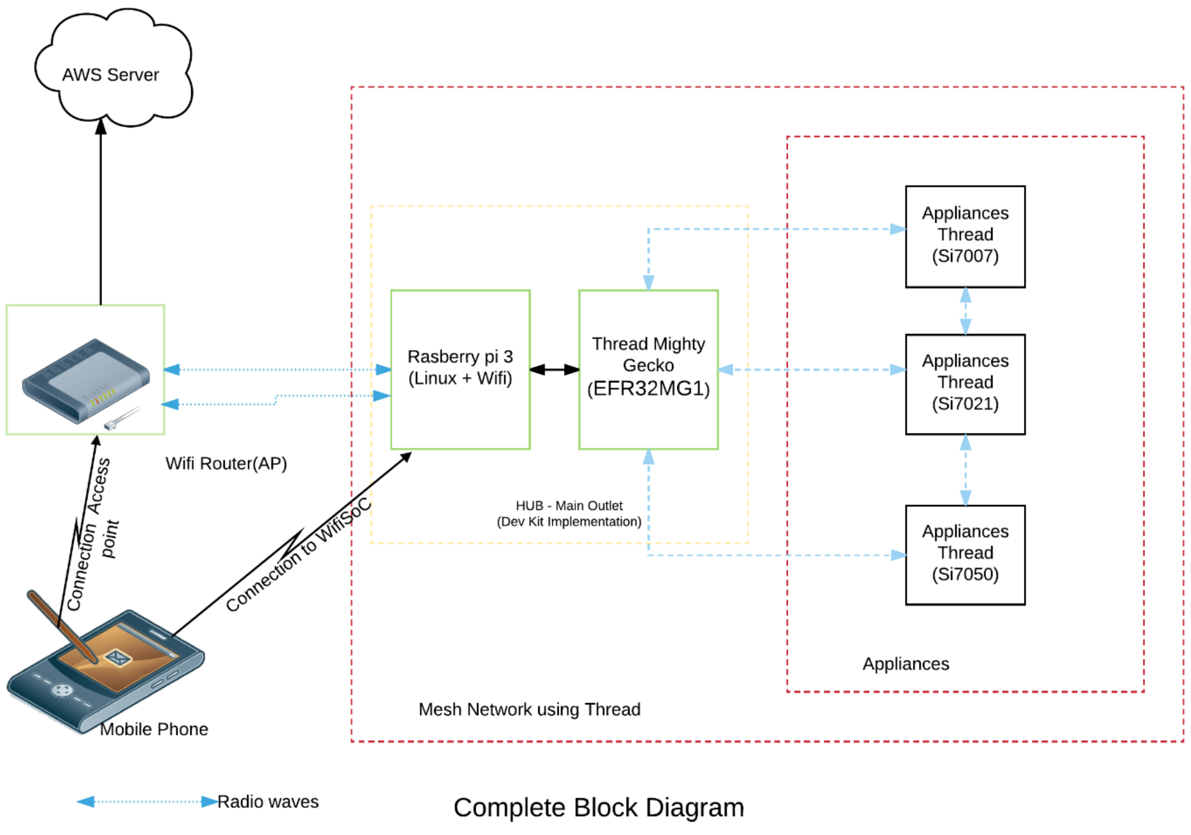

AWS Server
Wifi Router(AP)
Connection Access point
Connection to WifiSoC
Mobile Phone
Rasberry pi 3 (Linux + Wifi)
Thread Mighty Gecko (EFR32MG1)
HUB - Main Outlet (Dev Kit Implementation)
Appliances Thread (Si7007)
Appliances Thread (Si7021)
Appliances Thread (Si7050)
Appliances
Mesh Network using Thread
Radio waves
Complete Block Diagram

Figure 3: Block Diagram

This product is trying to control the electrical appliances remotely using your mobile phone app/Desktop app. How is the goal going to be achieved, we don't have 802.15.4 in our mobile phones yet? Well, it has been solved using a hub based approach. The hub acts as a central server in this prototype. Hub talks to the server to get/put the request back and forth and converts data from the internet and send it to it's corresponding end devices.

A raspberry Pi and a Mighty gecko has been used as a server in this prototype. It has been described in detail in the sections below.

End devices were custom designed for our application and its rough block diagram is given below.

End devices have the hardware that has been mentioned above in hardware section. It is powered using AC Supply in the socket and a backup super capacitor. All the power involved blocks are colored in Red.

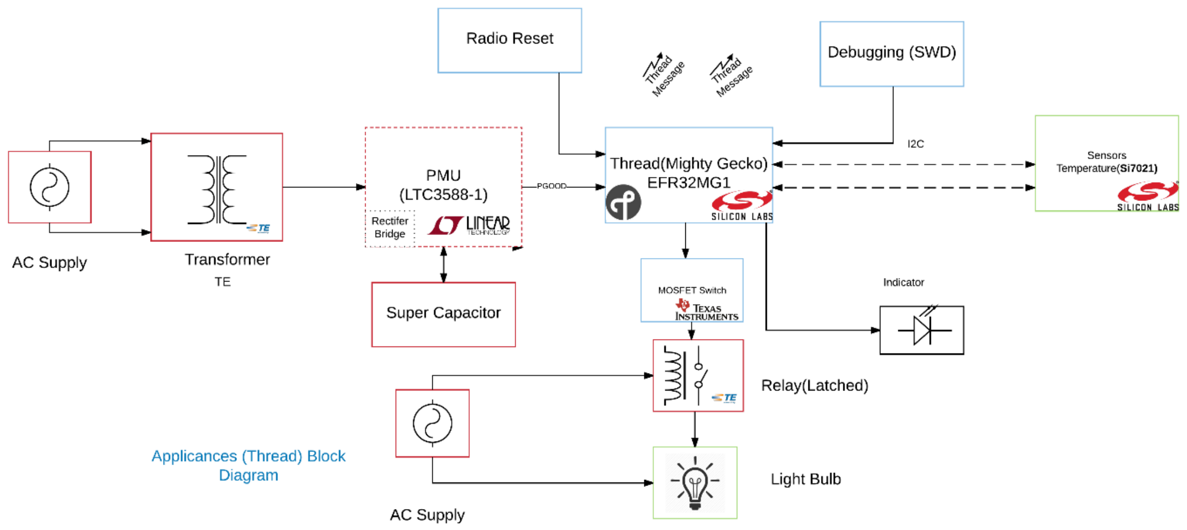


Figure 4: End Device Block Diagram

Hardware Test Plan

Function	Feature to Be tested	Plan Test	Definition of Pass/Fail	Pass/Fail
AC/DC	Transformer	Give 115 VAC on the primary side and check the output on the secondary side, before soldering it to the board	Make sure it gives out 10VAC, If it is above 10VAC it fails the test	Pass
Super Cap	Charging	Connect the AC, and check the vltage across the supercap using an oscilloscope	It should increase and reach 12V	Pass
PGOOD	Functionality of PMIC	Use a power resistor across the VPMIC and Gnd, to dissipate the energy stored in Super Cap, and check for the super cap voltage to go below 3.3V and check the Vout Simultaneously. It should go low to pass this test		Pass
PMIC	Rectification	This test has to be done after transformer test, Check the output at Vout pin of PMIC/ VPMIC testpoints in the board	It should give 3.3V after the capacitor charges for a minute. If it is	Pass

			above 3.3, it fails the test	
Relay	Testing the No Load functionality	Give 3.3V at the control pin of the TI switcch and check the state of the relay for changes	If it does change from Open to Close or Close to open, it passes the test	Pass
Relay	Load Condition	Check the No load test with a AC load plugged into the outlet plug	If it turns on/off the load, it works.	Pass
Sensor	I2C	Connect the MCU with I2C test program to check whether it is reading the data		Pass
JTAG	Board detection	Check the board is getting detected in simplicity studio		Not tested
JTAG	DEBUG	Use the debug option in simplicity studio to check the functionality		Not tested
LED	On/OFF	Program the LED code onto the board and check whether LED is glowing or not		Pass
S/W	Push Button	Add a test case in the code to glow an LED when pushing a button	If the LED glows it passed the test	Pass
	I2C Communication			Pass
	Thread Communication Check	Add a test case to send the sensor readings backa and forth among these two slave devices to complement the LED port every 10 seconds.	If the LED Toggles every 10 seconds, then it passes the test	Pass
	UART on Rasperry Pi and Thread USB Module	Create an UART interface to send a message when there is a reception in the module from other device. The other device sends a message when the push button is pressed	If it pops up the messages on Pi's Terminal it passes the test	Pass through USB
	Sensor Data on Pi	Now send the sensor data on a press of a push button	If it shows the same message as virtual terminal of the other device, this passes the test	Pass
	Relay control from Pi	Send a contol message on every push message reception, to turn on/off the relay.	If this toggles the Relay output, then it passes the test	Pass

	App	Create an interface to read the sensor data and Relay control in app, to send a message to Pi.	If the app shows the readings, then it passes the test	Pass

Test Execution

Software/Hardware Module	**Sub Module**	**Planned Completion Date**	**When Completed**
H/W	Continuity	13th Nov	15th Nov
	AC/DC	**21st Nov**	22nd Nov
	SuperCap	21st Nov	22nd Nov
	Loading microcontroller on AC	23rd Nov	23rd Nov
	Reset /PGOOD Check	22nd Nov	22nd Nov
	Sensor Interface	16th Nov	18th Nov
	Relay Actuation check	18th Nov	18th Nov
	Programming the device through JTAG	25th Nov	Not Tested
	Final check on demo	9th Dec	9th Dec
S/W	LED	18th Nov	18th Nov
	Push Button	18th Nov	20th Nov
	I2C Communication on Board	21st Nov	21st Nov
	Thread Communication Check	23rd Nov	23rd Nov
	Raspberry Pi Interface	25th Nov	27th Nov
	Get the sensor data into Pi's terminal	27th Nov	28th Nov
	Control a relay from Pi's UI	30th Nov	8th Dec
	Talk to the APP from Pi (WiFi)	2nd Dec	4th Dec
	Sensor Data on App	4th Dec	8th Dec
	Control the relay from App	6th Dec	9th Dec

Firmware

Server Implementation

Simplicity IDE has great features to use their thread stack and ember stack. This code has been taken and modified from Silabs Sample Server implementation code.

Inside while loop of main.c it changes state everytime by having a tick function. Once it gets the message from raspberry pi. It processes the message using B2B Protocol functions (Frame analysis to find out whom the message is intended for and how to forward it). Then it does a CoAP post from server to client on CoAP URI at (server/report). There will be a handler on client with this URI address(server/report) to serve what is needed from the client side. This code will be explained in later sections.

CoAP (Constrained Application Protocol)[5]

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained networks in the Internet of Things.The protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation.

Like HTTP, CoAP is based on the wildly successful REST model: Servers make resources available under a URL, and clients access these resources using methods such as GET, PUT, POST, and DELETE. From a developer point of view, CoAP feels very much like HTTP.

Obtaining a value from a sensor is not much different from obtaining a value from a Web API. Since HTTP and CoAP share the REST model, they can easily be connected using application-agnostic cross-protocol proxies. A Web client may not even notice that it just accessed a sensor resource.

The Internet of Things will need billions of nodes, many of which will need to be inexpensive. CoAP has been designed to work on microcontrollers with as low as 10 KiB of RAM and 100 KiB of code space (RFC 7228).

CoAP is designed to use minimal resources, both on the device and on the network. Instead of a complex transport stack, it gets by with UDP on IP. A 4-byte fixed header and a compact encoding of options enables small messages that cause no or little fragmentation on the link layer. Many servers can operate in a completely stateless fashion. The CoAP resource directory provides a way to discover the properties of the nodes on your network.

CoAP was developed as an Internet Standards Document, RFC 7252. The protocol has been designed to last for decades. Difficult issues such as congestion control have not been swept under the rug, but have been addressed using the state of the art.

The Internet of Things cannot spread as long as it can be exploited by hackers nilly-willy. CoAP does not just pay lip service to security, it actually provides strong security. CoAP's default choice of DTLS parameters is equivalent to 3072-bit RSA keys, yet still runs fine on the smallest nodes.

Configuration of CoAP URI's can be done using Simplicity Studio. Under Others/CoAP Dispatch we can define the URI and corresponding callbacks. (This is helpful for handling the commands from clients and server).

[5] http://coap.technology/

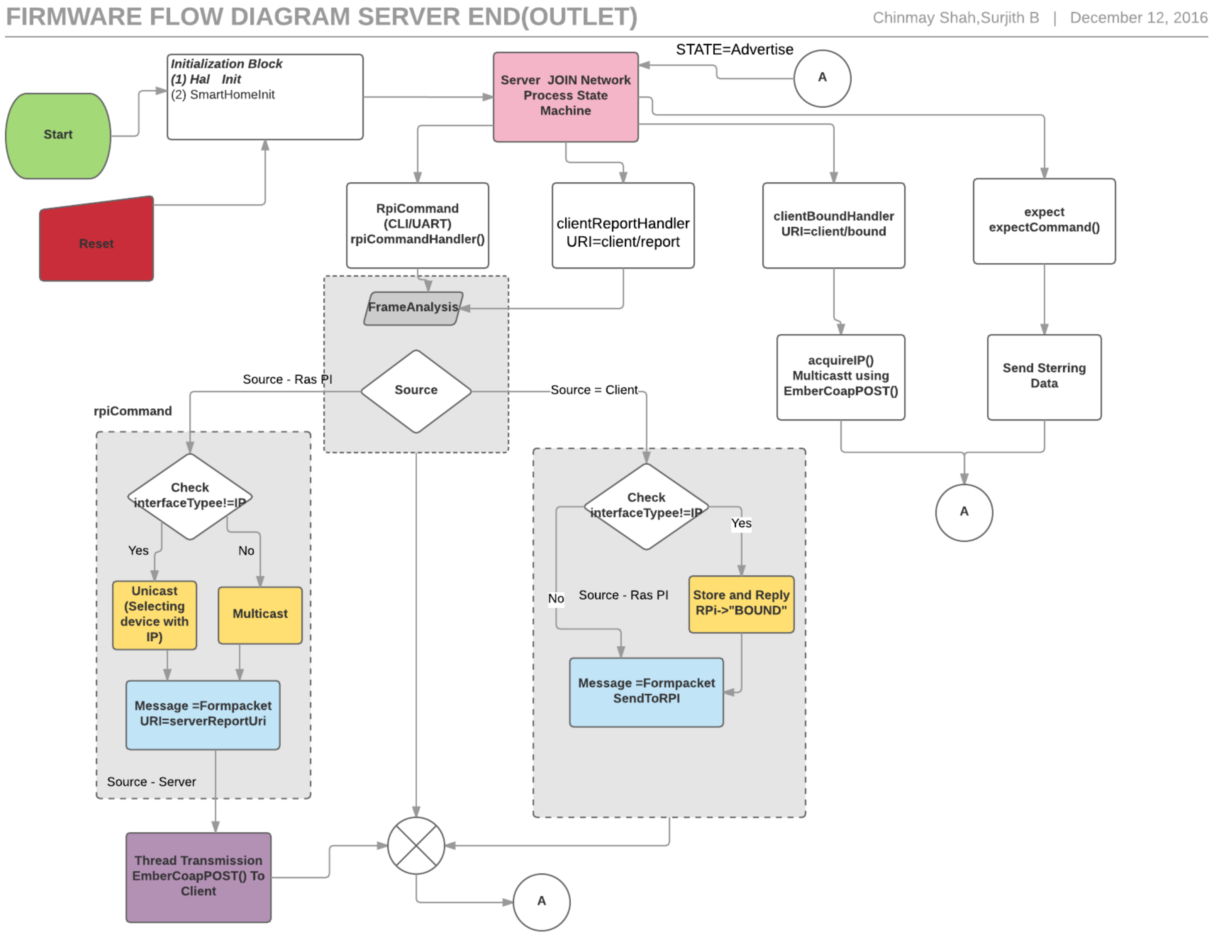

FIRMWARE FLOW DIAGRAM SERVER END(OUTLET)
Chinmay Shah,Surjith B | December 12, 2016
Start
Initialization Block
(1) Hal Init
(2) SmartHomeInit
Reset
Server JOIN Network Process State Machine
STATE=Advertise
A
RpiCommand (CLI/UART) rpiCommandHandler()
clientReportHandler URI=client/report
clientBoundHandler URI=client/bound
expect expectCommand()
FrameAnalysis
acquireIP() Multicastt using EmberCoapPOST()
Send Sterring Data
Source - Ras PI
Source
Source = Client
rpiCommand
Check interfaceTypee!=IP
Yes
No
Unicast (Selecting device with IP)
Multicast
Message =Formpacket URI=serverReportUri
Source - Server
Check interfaceTypee!=IP
Yes
No
Source - Ras PI
Store and Reply RPi->"BOUND"
Message =Formpacket SendToRPI
Thread Transmission EmberCoapPOST() To Client
A
A

Figure 5: Server Firmware Flow

Main

Main function has all the hardware initializations. Hardware Initializations such as clock initialization, chip initialization, watchdog Initialization has been done using halInit() function call in ember stack.

Additional initializations for Si7021, Relay control (GPIO) Initialization has been done in initializationSmartHome() function.

Main function will move forward only If halInit successfully initiates all the hardware and clocks properly. Once the initializations are done, it goes into the loop where it serves watchdog and has a tick for the state machine which has been explained below.

Server state machine is implemented in server-implementation.c. Before going into the server implementation code, there are few configurations that has to be explained in simplicity studio's isc file of the project.

Plugins and Command Line Callbacks

Plugins – These are the plugin callbacks that are needed for server implementations. Few important ones are CoAP, DHCP, ICMP, Network Management. This project is developed on Thread 2.0.0. Stack.

Server is connected to the Raspberry Pi for Hub Operation. So Raspberry will give commands to the server using UART(Command Line Interface). In order to respond to the messages from raspberry pi, the callbacks can be automatically generated in Simplicity Studio. To add a new callback, just add new callback under Printing section in isc file under Command Line Interface. We have to mention the arguments that function calls are going to take and it will be initialized in thread-callbacks.c. It can be modified once it creates the callbacks in those files.

Server State flow

Server State Flow diagram starts with INITIAL, and it checks whether the network is already there or it has to form a new network. It goes to RESUME_NETWORK state. It checks whether random master key flag is set or not.

If the random master key is defined then it sets the security parameters (SET_SECURITY_PARAMETERS) and goes to FORM_NETWORK state. Where it checks whether it is attached to the network or not (EMBER_ JOINED_NETWORK_ATTACHED, This is a variable from the network stored in the server(check old network status)). It goes to GET_COMMISIONER state. In this state it actually checks whether it can become a commissioner, if it can then it goes to BECOME_COMMISIONER State.

Then it transitions to MAKE_ALL_THREAD_NODES_ADDRESS. In this state it gets the multicast address from the nodes and starts advertising. Advertising state repeats after ADVERTISE_PERIOD_MS (60 Seconds). And this returns the to client handler when it receives some message from client.

For detailed understanding refer the Thread server state flow diagram. For detailed explanation refer the code at Github Repo.

Client Implementation

Client code is similar to server code but with less states and the flow is little bit different. Message Handling is same as the server. It handles CoAP Posts and gets with its corresponding callbacks.

Si7021 Interface

I2C Drivers are imported from ZigBee Embernet Stack. humidity-temperature-si7021.c and sensor-control.c has been incorporated to read, measure and initiate sensor. The functions were explained below (SI_7021_ADDR =(0x40<<1))

halHumidityInit()

Initiates the sensor and hardware pins in the MCU. It uses halHumidityStartRead() function to initiate the sensor. Set of HEX Values are loaded onto the I2C Bus to configure the sensor to start reading. Refer the datasheet of Si7021 for more details.[6]

halHumidityStartRead()

Initiates the humidity and temperature sensor Si7021 using si7021Init().Sends a reset command, SI_7021_RESET (0xFE) to sensor to default values.

si7021StartRead ()

This sends a hex code to sensor through I2C Bus, SI_7021_MEASURE_RELATIVEHUMIDITY (0xF5) .

[6] https://www.silabs.com/Support%20Documents%2FTechnicalDocs%2FSi7021-A20.pdf

Humidity needs temperature value to do the temperature compensation for relative humidity. If we need both temperature and Humidity readings. It is better to measure Humidity alone and it can read both Temperature and Humidity.

readTemperature()

It is used for reading higher and lower byte values from sensor and convert it to degree Celsius. It has an option of type which allows measurement before reading if required . This returns the value of temperature when the humidity was measured.

readHumidity()

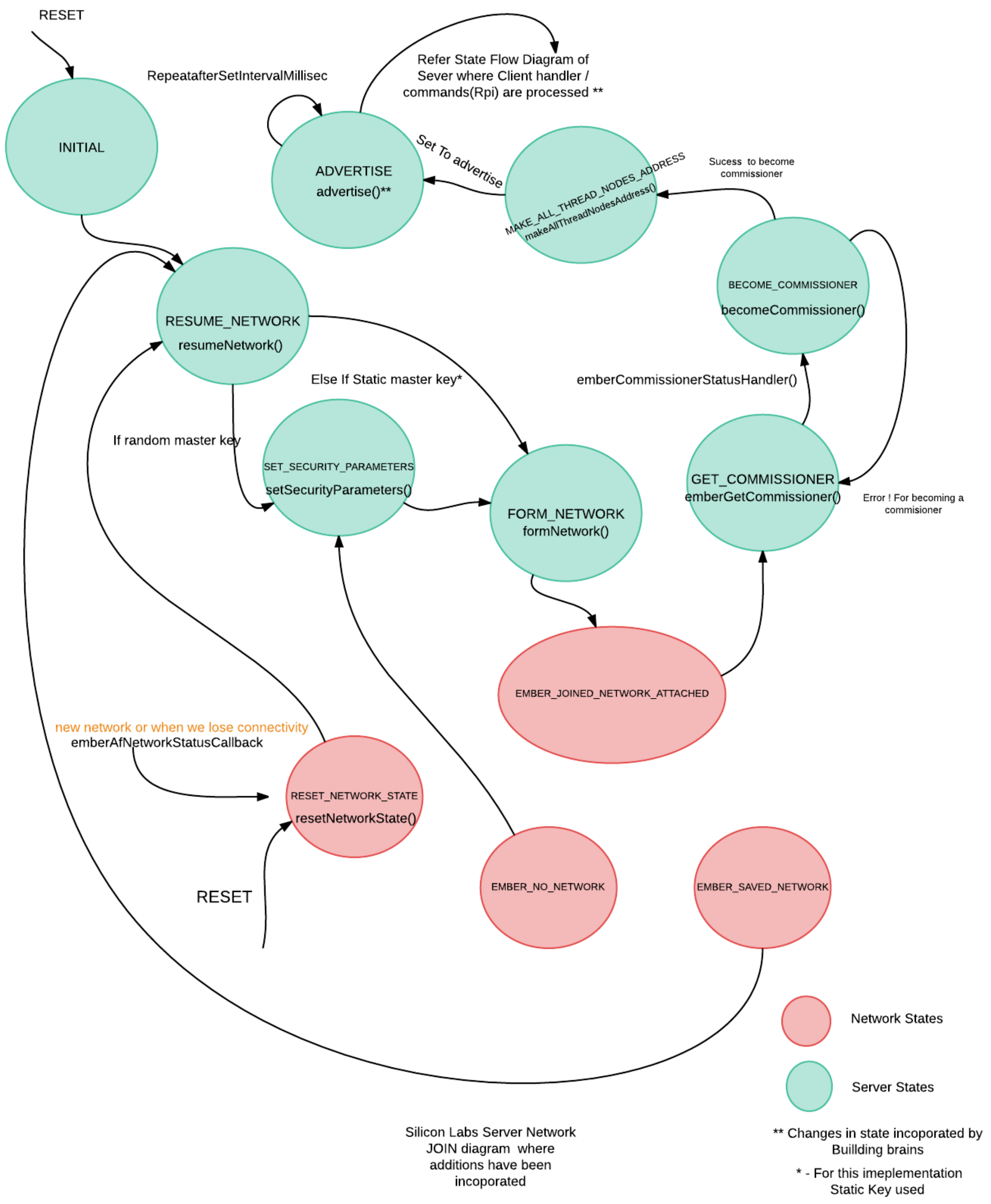
It is used for reading higher and lower byte values from sensor and convert it to Relative humidity (%) after sending measurement command to Si7021.

State Flow

The flow starts at INITIAL State as server. It checks whether the network already existed or not. If it exists then it goes to RESUME_NETWORK state to resume the already existing network. Else it waits for advertisement from server and goes to JOIN_NETWORK State.

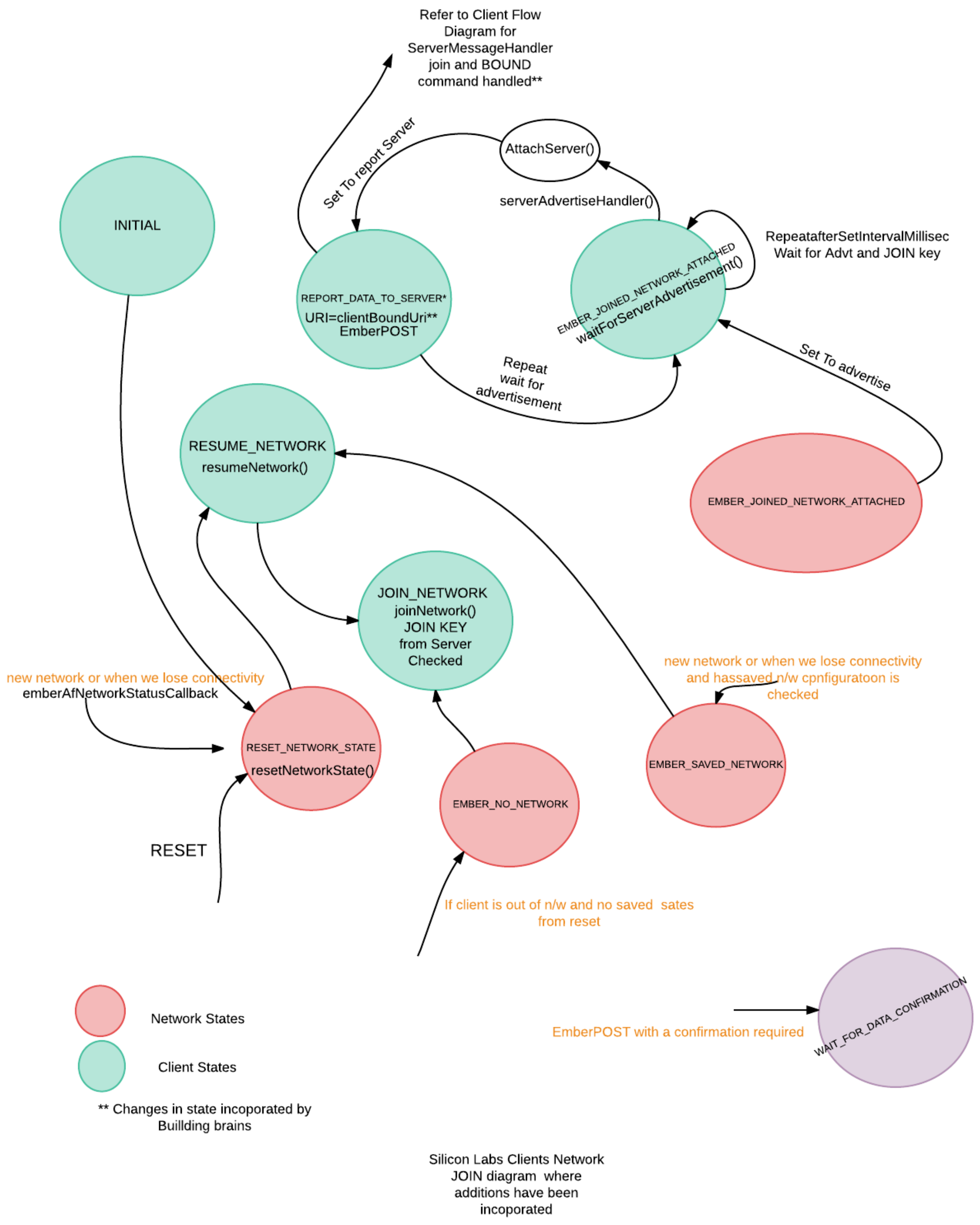
Then it checks whether it is attached to the network (EMBER_JOINED_NETWORK_ATTCHED), if it is attached to the network it just reports data to Server. Else it waits for server advertisement and Bounds to the server when it receives a join request. Then it attaches the server.

After this it just continuously transmits the data to the server on arrival of request.



RESET
RepeatafterSetIntervalMillisec
Refer State Flow Diagram of Sever where Client handler / commands(Rpi) are processed **
INITIAL
ADVERTISE advertise()**
Set To advertise
MAKE_ALL_THREAD_NODES_ADDRESS makeAllThreadNodesAddress()
Sucess to become commissioner
BECOME_COMMISSIONER becomeCommissioner()
RESUME_NETWORK resumeNetwork()
Else If Static master key*
emberCommissionerStatusHandler()
If random master key
SET_SECURITY_PARAMETERS setSecurityParameters()
FORM_NETWORK formNetwork()
GET_COMMISSIONER emberGetCommissioner()
Error ! For becoming a commisioner
EMBER_JOINED_NETWORK_ATTACHED
new network or when we lose connectivity
emberAfNetworkStatusCallback
RESET_NETWORK_STATE resetNetworkState()
RESET
EMBER_NO_NETWORK
EMBER_SAVED_NETWORK
Network States
Server States
Silicon Labs Server Network JOIN diagram where additions have been incoporated
** Changes in state incoporated by Buillding brains
* - For this imeplementation Static Key used

Figure 6: Thread Server Stateflow diagram



Refer to Client Flow Diagram for ServerMessageHandler join and BOUND command handled**
Set To report Server
AttachServer()
serverAdvertiseHandler()
INITIAL
REPORT_DATA_TO_SERVER*
URI=clientBoundUri**
EmberPOST
EMBER_JOINED_NETWORK_ATTACHED
waitForServerAdvertisement()
RepeatafterSetIntervalMillisec
Wait for Advt and JOIN key
Repeat wait for advertisement
Set To advertise
RESUME_NETWORK
resumeNetwork()
EMBER_JOINED_NETWORK_ATTACHED
JOIN_NETWORK
joinNetwork()
JOIN KEY from Server Checked
new network or when we lose connectivity and hassaved n/w cpnfiguratoon is checked
new network or when we lose connectivity
emberAfNetworkStatusCallback
RESET_NETWORK_STATE
resetNetworkState()
EMBER_SAVED_NETWORK
EMBER_NO_NETWORK
RESET
If client is out of n/w and no saved sates from reset
EmberPOST with a confirmation required
WAIT_FOR_DATA_CONFIRMATION
Network States
Client States
** Changes in state incoporated by Buillding brains
Silicon Labs Clients Network JOIN diagram where additions have been incoporated

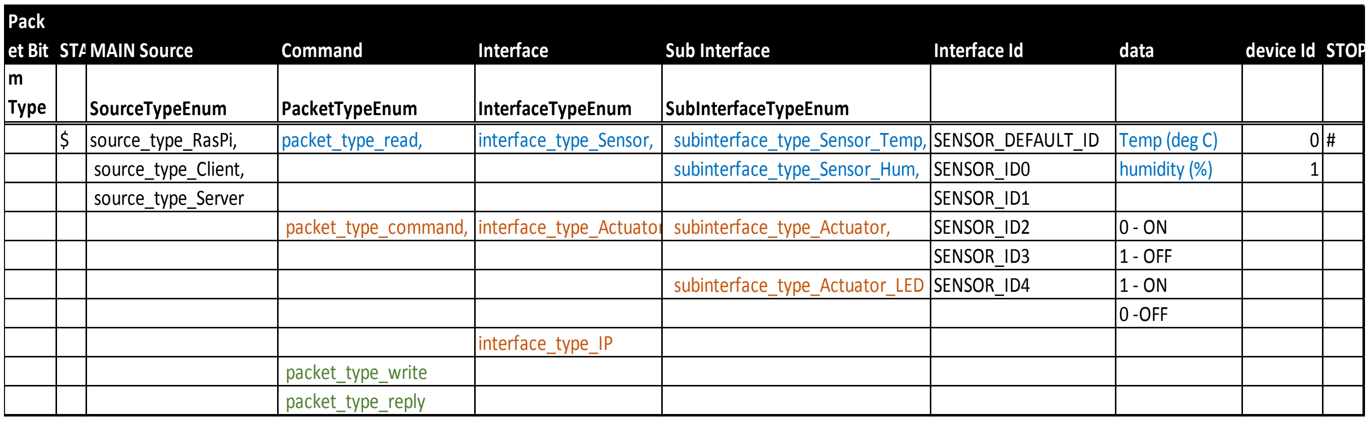
Figure 7: Thread Client State FLow

B2B Protocol

To handle different communication between client and server we have added a separate file unique for our application called B2BProtocol.c and B2Bprotocol.h that forms and analyzes frame which is transferred between server and client.

This protocol is used for perform the tasks such as reading sensor data and controlling actuator etc described later in this section.This also support functions for splitting strings to packet and vice versa for transmission b/w border router/Server to Client.

We have used the following format for the packet with packet contents



Pack et Bit	STA	MAIN Source	Command	Interface	Sub Interface	Interface Id	data	device Id	STOP
m Type		SourceTypeEnum	PacketTypeEnum	InterfaceTypeEnum	SubInterfaceTypeEnum				
	$	source_type_RasPi,	packet_type_read,	interface_type_Sensor,	subinterface_type_Sensor_Temp,	SENSOR_DEFAULT_ID	Temp (deg C)	0	#
		source_type_Client,			subinterface_type_Sensor_Hum,	SENSOR_ID0	humidity (%)	1	
		source_type_Server				SENSOR_ID1			
			packet_type_command,	interface_type_Actuato	subinterface_type_Actuator,	SENSOR_ID2	0 - ON		
						SENSOR_ID3	1 - OFF		
					subinterface_type_Actuator_LED	SENSOR_ID4	1 - ON		
							0 -OFF		
				interface_type_IP					
			packet_type_write						
			packet_type_reply						

Each of the packet contents has an enumerated type and the packet format is described in the above table. This Excel Sheet is uploaded in the github repository. Sametype of commands are represented in similar color. Blue – Read, Orange – Command, Green – Write/Reply.

In order to understand the structure, have a look at B2BProtocol.c in github repo.

Functions supported

*char * formpacket (packet sendPack)*

It forms the contents of structure packet into a string ready for transmission using emberCoapPOST

packet splitPacket ()

It splits the input string into contents of structure packet which can be used for analysis and performing the tasks

void printPacket(packet printPacket)

It prints the contents of packet, Parses the structure and prints it in a readable format.

uint32_t ReadCommand(packet RxPacket, PacketStatus_TypeDef error_type)

Used to call sensor read or actuate depending on interface.

uint32_t ReadSensor(SubInterface_TypeDef subinterface)

Invokes fetchSensorData and returns sensor reading according to sensor type.

uint8_t ExecuteCommand(packet RxPacket, PacketStatus_TypeDef error_type)

Invokes command of ON or OFF(packet.data) for actuator or LED depending on interface and sub interface.

packet FrameAnalysis(char RxMessage[])

It splits the message received either from any source to analyse and execute commands depending on main source type

- source_type_RasPi – Forwards to client and changes source type to source_type_Server
- source_type_Server – Checks for type of command
    - Read type
        - interface_type_IP – reply with data as zero
        - interface_type_sensor – readSensor used
        - interface_type_actuator – readActuator used
    - Execute_cmd
        - ExecuteCommand() – It actuates(on or off) the relay or Led according to command received from the server
    -
- source_type_Client
    - Receives data from Client on server side and changes the source type to server
    - It sends Rpi with a tag "MFPI:" for the raspberry PI to know the reply is meant for PI
    - According to type of interface –
        - IP – Server stores new IP and sends "MFPI:BOUND:<device id>"
        - Sensor – Server sends "MFPI:<sensor data>:<device id>"
        - Others - Server sends "MFPI:<packet>"

Power Energy Analysis

Energy Analysis has been done using Dev Kit. Not on the actual board.

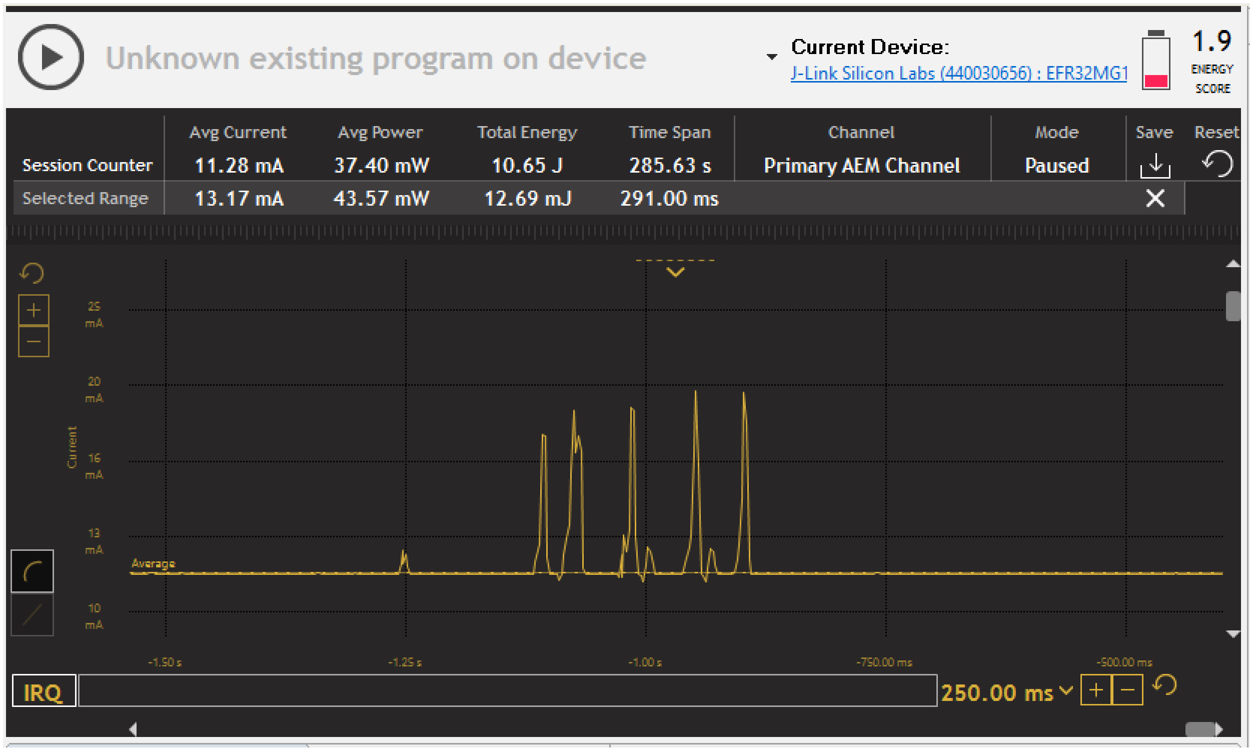
Forming of Network

Advertise and IP capture Command

- For the initial State of receiving of advertise signals from server and replying message for IP interface it uses EmberCoapPOST() resulting in spikes of current
- Transmission – leads to taller peak consuming about 20 mA of current
- Reception – a shorter peak of approximately 12 mA

BOUND and IP capture Command reply

- Once n/w is established during Client replying message for expect command and performing Key matching
- It also sends BOUND message and IP Command interface reply using EmberCoapPOST() resulting in spikes of current
- Reception – leads to taller peak consuming about 25 mA of current
- Transmission – a shorter peak of appox 13 mA

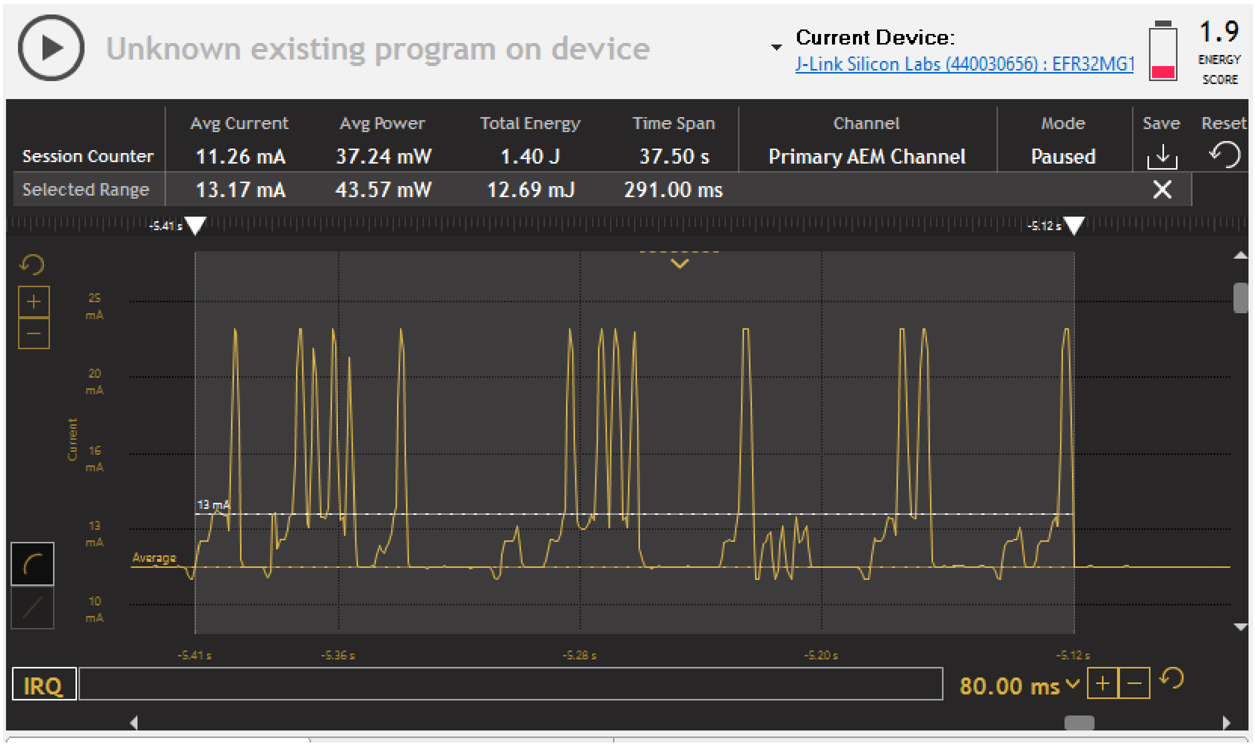

Unknown existing program on device

Current Device: J-Link Silicon Labs (440030656) : EFR32MG1

1.9 ENERGY SCORE

	Avg Current	Avg Power	Total Energy	Time Span	Channel	Mode
Session Counter	11.28 mA	37.40 mW	10.65 J	285.63 s	Primary AEM Channel	Paused
Selected Range	13.17 mA	43.57 mW	12.69 mJ	291.00 ms		

Figure 8: Energy Diagram for advertise state and IP capcture on Client Side


Unknown existing program on device

Current Device: J-Link Silicon Labs (440030656) : EFR32MG1

1.9 ENERGY SCORE

	Avg Current	Avg Power	Total Energy	Time Span	Channel	Mode
Session Counter	11.26 mA	37.24 mW	1.40 J	37.50 s	Primary AEM Channel	Paused
Selected Range	13.17 mA	43.57 mW	12.69 mJ	291.00 ms		

Figure 9 Energy Profiler during BOUND and IP Command Reply

Functionality Commands

LED Control

On State

- Reception – leads to taller peak consuming about 19 mA of current when command received and handled
- Transmission – a shorter peak of appox 13 mA returning status of command
- It transitions from < 11 mA to approx 12 mA when it is in ON

OFF State

- Reception – leads to taller peak consuming approx 19 mA of current when command received and handled
- Transmission – a shorter peak of appox 13 mA returning status of command
- It transitions from > 12 mA to approx 10.5 mA when it is in OFF

RELAY Control

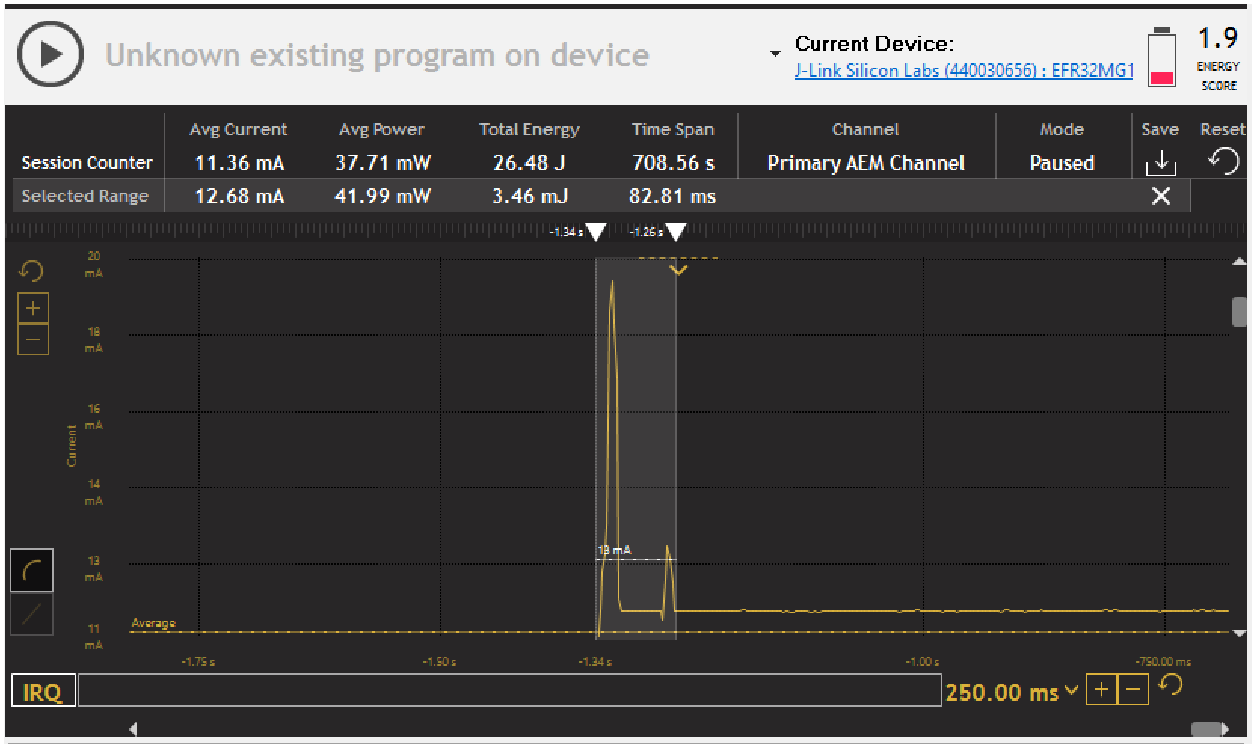
On State

- Tested on the dev kit instead of board
- Reception – leads to taller peak consuming approx 19 mA of current when command received and handled using packet analysis
- Transmission – a shorter peak of appox 13 mA returning status of command

It transitions from > 13.5 mA when LED to be ON

Temperature /Humidity Measurement

- I2C reading and measurement commands about 2mA of current consumption in start of reception
- Reception – leads to taller peak consuming approx 19 mA of current when command received and handled using packet analysis
- Transmission – a shorter peak of appox 13 mA returning status of command


Unknown existing program on device

Current Device:
J-Link Silicon Labs (440030656) : EFR32MG1

1.9 ENERGY SCORE

	Avg Current	Avg Power	Total Energy	Time Span	Channel	Mode	Save	Reset
Session Counter	11.36 mA	37.71 mW	26.48 J	708.56 s	Primary AEM Channel	Paused		
Selected Range	12.68 mA	41.99 mW	3.46 mJ	82.81 ms				

Figure 10 Switching LED state form OFF to ON

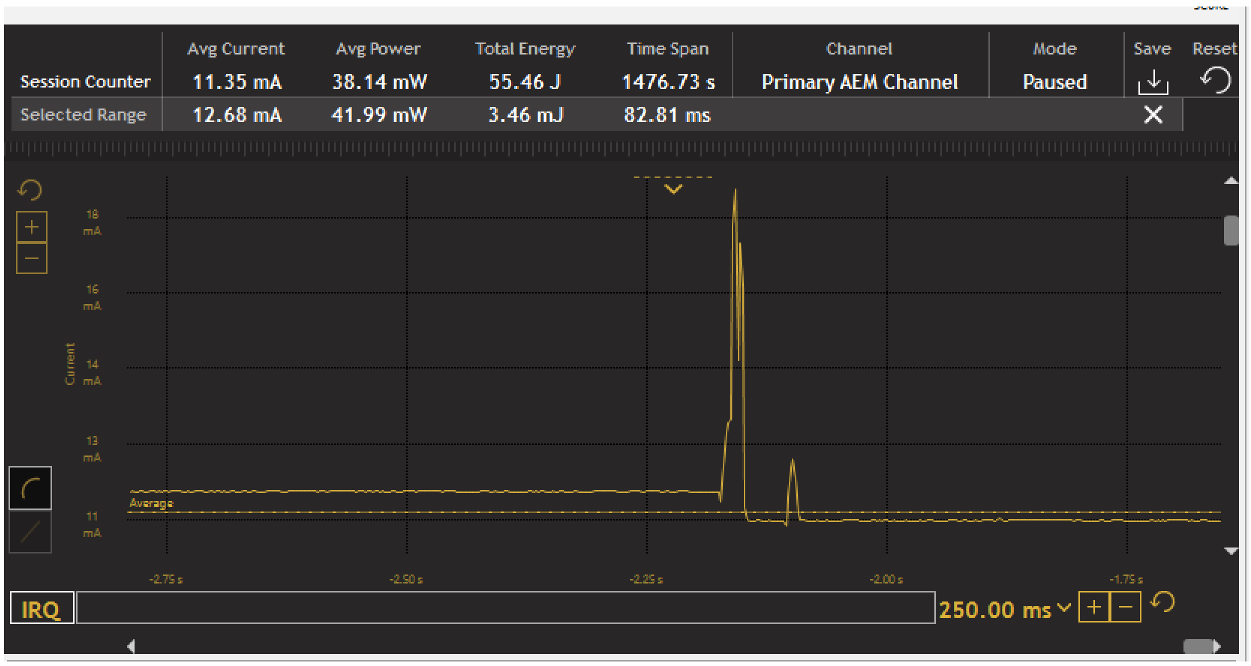
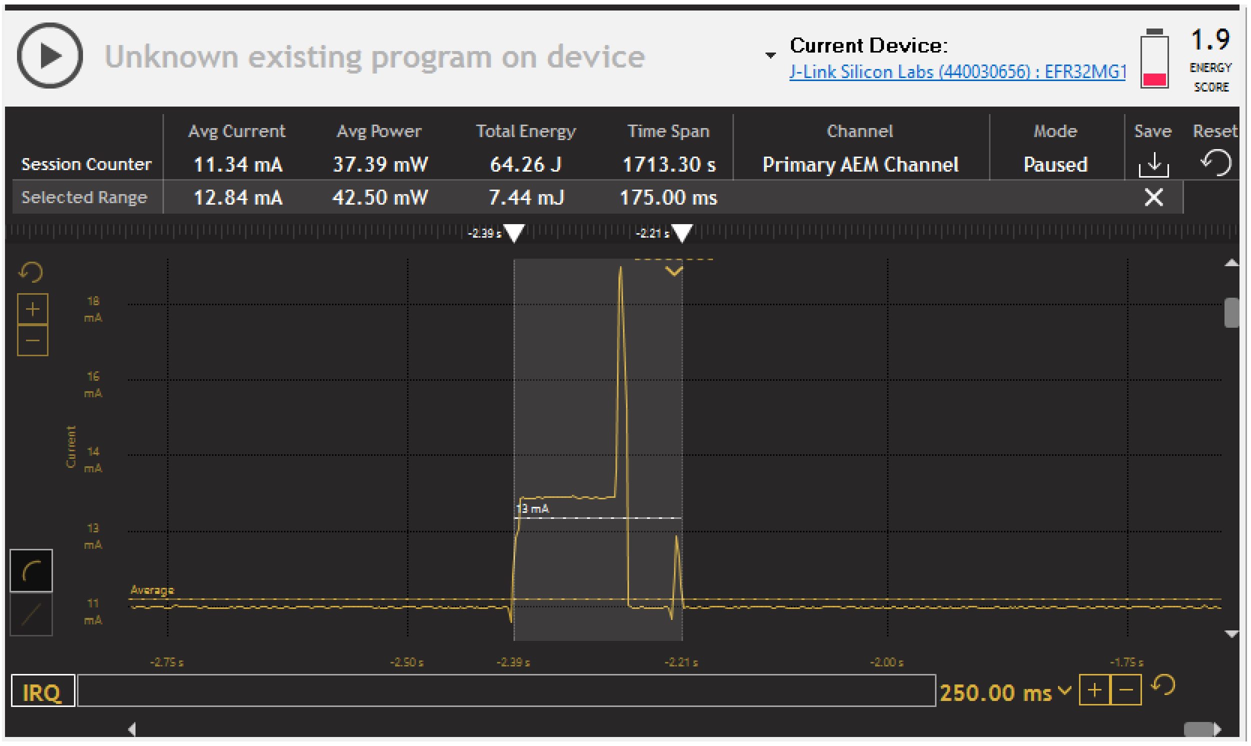
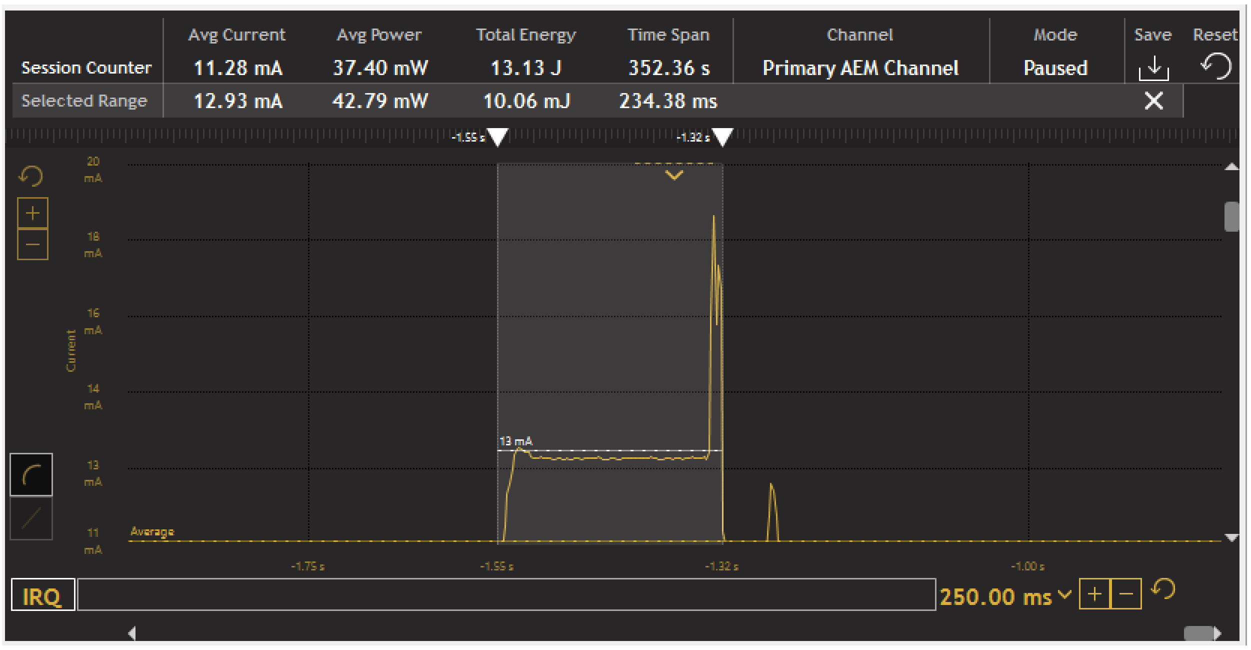

	Avg Current	Avg Power	Total Energy	Time Span	Channel	Mode	Save	Reset
Session Counter	11.35 mA	38.14 mW	55.46 J	1476.73 s	Primary AEM Channel	Paused		
Selected Range	12.68 mA	41.99 mW	3.46 mJ	82.81 ms				

Figure 11 Switching LED state form ON to OFF


Unknown existing program on device
Current Device:
J-Link Silicon Labs (440030656) : EFR32MG1
1.9
ENERGY SCORE
IRQ
250.00 ms

	Avg Current	Avg Power	Total Energy	Time Span	Channel	Mode
Session Counter	11.34 mA	37.39 mW	64.26 J	1713.30 s	Primary AEM Channel	Paused
Selected Range	12.84 mA	42.50 mW	7.44 mJ	175.00 ms		

Figure 12 Switching RELAYstate to ON for 10 ms and OFF state



	Avg Current	Avg Power	Total Energy	Time Span	Channel	Mode
Session Counter	11.28 mA	37.40 mW	13.13 J	352.36 s	Primary AEM Channel	Paused
Selected Range	12.93 mA	42.79 mW	10.06 mJ	234.38 ms		

Figure 13: Temperature and Humidy sensor read

Software

Desktop Application has been developed using Cross platform python package Kivy. This gives the basic UI needed for the mobile application. Twisted python package is used for tcp communication with the server.

Kivy[7]

Kivy is an open source Python library for developing mobile apps and other multitouch application software with a natural user interface (NUI). It can run on Android, iOS, Linux, OS X, and Windows. Distributed under the terms of the MIT license, Kivy is free and open source software.Kivy is the main framework developed by the Kivy organization, alongside Python for Android Kivy iOS and several other libraries meant to be used on all platforms.

The software flow of mobile app has been attached below.

Twisted[8]

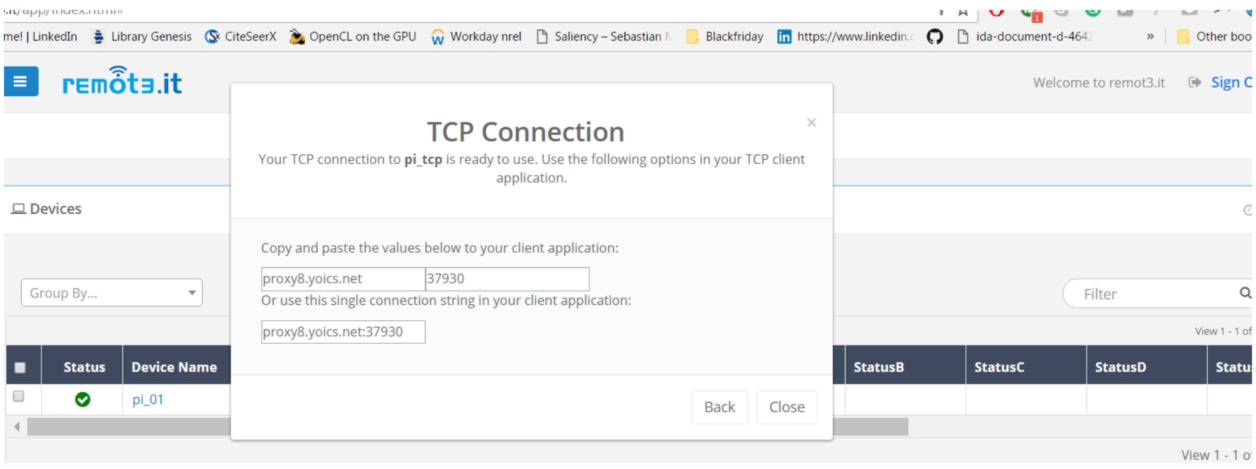
Twisted is an event-driven network programming framework written in Python and licensed under the MIT License.Twisted projects variously support TCP, UDP, SSL/TLS, IP multicast, Unix domain sockets, a large number of protocols (including HTTP, XMPP, NNTP, IMAP, SSH, IRC, FTP, and others), and much more. Twisted is based on the event-driven programming paradigm, which means that users of Twisted write short callbacks which are called by the framework.

In this project twisted has been used to talk to the server and listen to the server. TCP Protocol is used to make a reliable transmission of messages over internet.

Remot3[9]

Remot3 has been used to talk to Pi using internet. Port forwarding has been done using this tool. For detailed instruction on how to do port forwarding on Raspberry Pi, Refer to [remot3.it](https://www.remot3.it/web/) site.

Pi has been configured with remot3 packages. This can be used by following the steps given below



TCP Connection

Your TCP connection to **pi_tcp** is ready to use. Use the following options in your TCP client application.

Copy and paste the values below to your client application:

proxy8.yoics.net 37930

Or use this single connection string in your client application:

proxy8.yoics.net:37930

View 1 - 1 of

	Status	Device Name	StatusB	StatusC	StatusD	Statu
	✓	pi_01				

View 1 - 1 o

Figure 14: Remot3 Configuration

[7] https://en.wikipedia.org/wiki/Kivy
[8] https://en.wikipedia.org/wiki/Twisted_(software)
[9] https://www.remot3.it/web/

Raspberry Pi Server [10]

A python script which acts like a middle man to both Remot3 Cloud server and Thread Server(802.15.4) attached to all the end devices in the thread network.

This receives the message from Mobile Application through Remot3 server to give commands to the end devices.

Raspberry Pi is talking to WSTK via USB (UART through USB). Initially we worked with CEL USB Modules. CEL Modules are EM3588 based hardware. Our End Devices are EFR32MG based devices. We successfully configured EM3588 to send commands through 802.15.4. But we cannot be able to achieve the pairing of EM3588 and EFR32MG. According to thread protocol this should not be an issue. According to Silicon Labs Documentation, this can be paired.

There was an issue with CEL Modules, It needs some linux driver modifications in cp210x driver file. We uploaded the modified linux driver in github repository.[11] CEL Application Engineer have been notified with the issue and solution.

Raspberry Pi's UART Ports (Rx and Tx Pins) needs to be initialized. Detailed documentation on the process is given in this link[12].

The server listens on port 8000 and it is configured to talk to remot3 server. For more details on Pi Server refer to github link of the project.

[10] https://github.com/surjithbs17/thread_application/blob/master/pi/

[11] https://github.com/surjithbs17/thread_application/tree/master/Linux_3.x.x_VCP_Driver_Source

[12] http://spellfoundry.com/2016/05/29/configuring-gpio-serial-port-raspbian-jessie-including-pi-3/

Android Application[13]

Mobile phone app has a main screen with controls like Configure Pi, Add Device and Use Device.

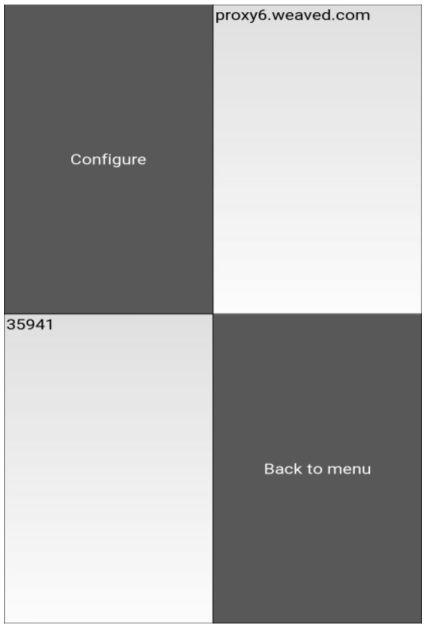


Figure 15: Configure Pi

Configure Pi

After getting the Web address and IP from remot3 server. Open the mobile application and type in the details as shown in figure 4 ("proxy6.weaved.com" and "35941"). Press configure button. This will give a pop up notification saying that the connection has been established with the server address and port number.

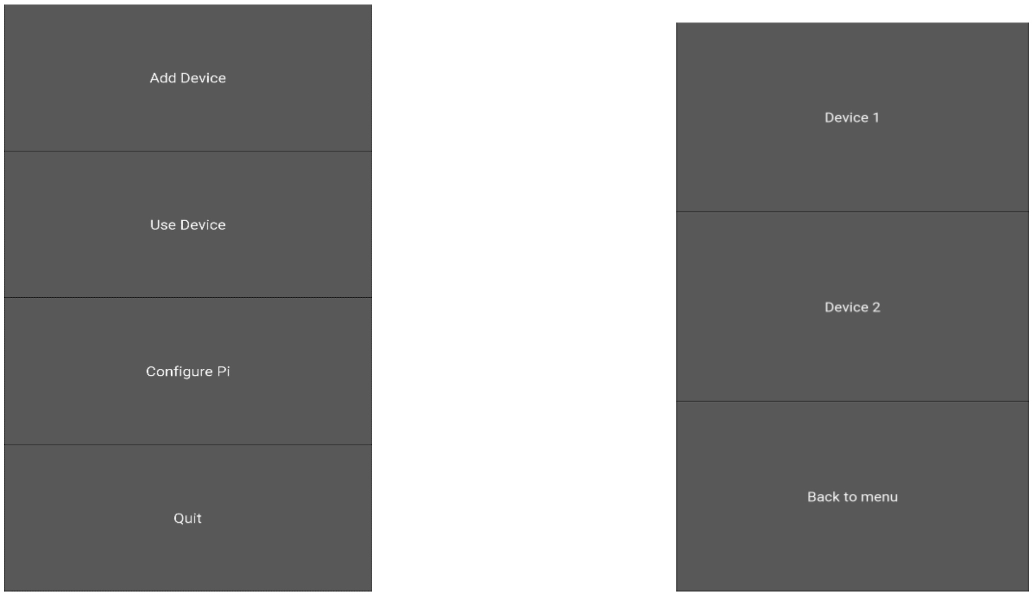
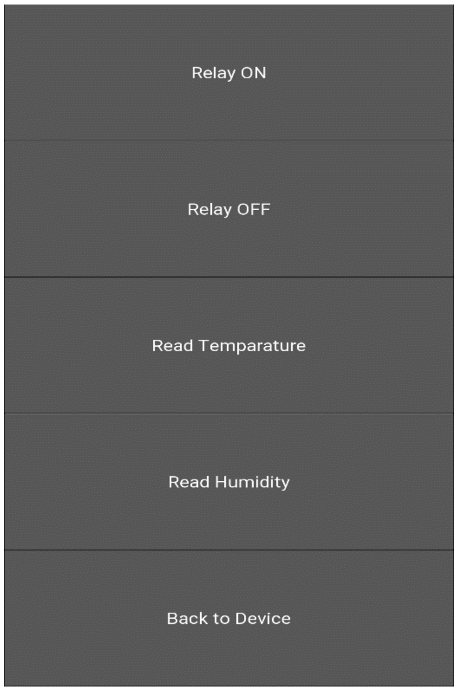


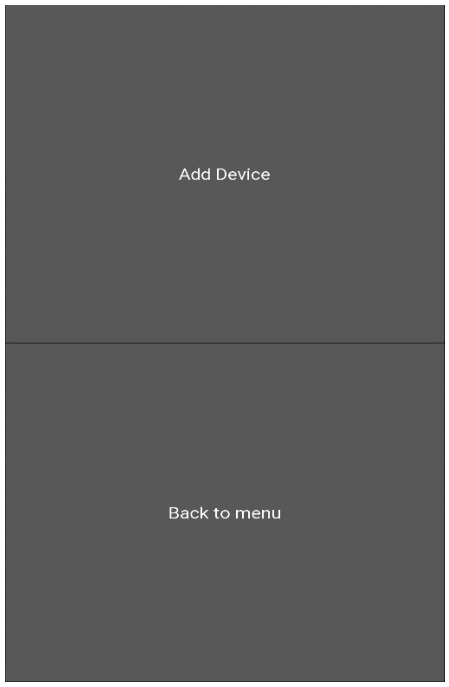
Figure 16: Add device and Use Device

[13] https://github.com/surjithbs17/thread_application/tree/master/mobile_app

Add Device

This button sends “Add device” command to the pi through the server. Pi processes the command to issue the commands needed for EFR32MG and Pi send back a confirmation message saying that the device has been added to the server.





Use Device

Once the device has been added to the thread network, it can be used using this User interface. It will throw an error if the device is not already added into the network.

This button takes you to the next screen where it shows the list of devices. Once you select a device from this screen, it gives the options that device has like (turn on the appliance, turn off the appliance, read temperature, read humidity).

Pi replies back to the Mobile app after doing the task that has been issued by the command. This Pops up as a message saying that device is turned on/off. And returns with the value of the sensor reading if it is a read sensor command.

Quit

This is the exit control for the mobile app.

For detailed understanding have a look at the Flow chart below. Code is available in Github Repository[14].

[14] https://github.com/surjithbs17/thread_application/tree/master/mobile_app

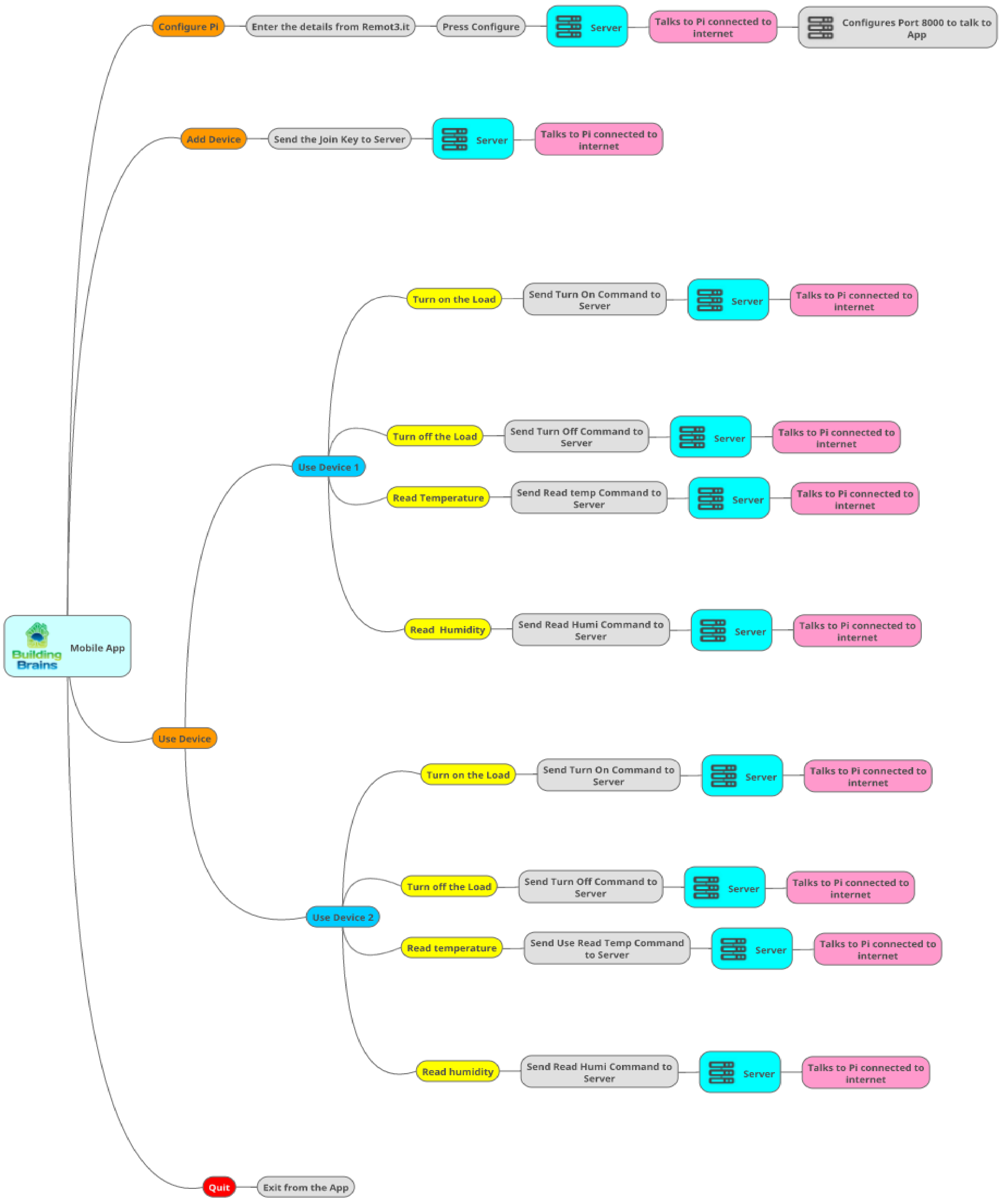

Mobile App
Building Brains
Configure Pi
Enter the details from Remot3.it
Press Configure
Server
Talks to Pi connected to internet
Configures Port 8000 to talk to App
Add Device
Send the Join Key to Server
Server
Talks to Pi connected to internet
Use Device
Use Device 1
Turn on the Load
Send Turn On Command to Server
Server
Talks to Pi connected to internet
Turn off the Load
Send Turn Off Command to Server
Server
Talks to Pi connected to internet
Read Temperature
Send Read temp Command to Server
Server
Talks to Pi connected to internet
Read Humidity
Send Read Humi Command to Server
Server
Talks to Pi connected to internet
Use Device 2
Turn on the Load
Send Turn On Command to Server
Server
Talks to Pi connected to internet
Turn off the Load
Send Turn Off Command to Server
Server
Talks to Pi connected to internet
Read temperature
Send Use Read Temp Command to Server
Server
Talks to Pi connected to internet
Read humidity
Send Read Humi Command to Server
Server
Talks to Pi connected to internet
Quit
Exit from the App

Figure 17: Mobile App Flow Chart

Difficulties faced in the project

- Initially we had issues in getting thread samples to work due to lack of documentation
- Transformer data was incorrect in TE's Site
- MOSFET Switch on the board did not work. Circuit configuration makes it floating instead of making it zero. It could have been solved by testing the circuit before hand.
- CEL Module (EM3588) based USB Module not getting paired with EFR32MG
- CEL Module linux driver needed few modification (Updated Driver is in our project's git repository)
- GPIO configuration in EFR32 has some issues. We somehow made it to work. We still did not understand it fully.
- Power Resistor added to solve inrush current of capacitor and to limit the input current, turns out that limits the current consumption of the load and current limit has been done internally in LTC3588
- EFR32MG UART through pins were not working, we accessed UART using USB.
- Low Power Energy Modes have some interference with thread stack, and we are working on it to understand it better. Forum posts says that this could be solved in future thread stacks.
- Capacitor Discharging faster than expected, understood the issue. We cannot be able to make it go to Energy Modes
- When we soldered the Board, we misplaced the headers and was a big pain removing it and resoldering it.
- We did a mistake in soldering an LTC Soldering and it is good that we soldered two boards to figure out the issue.
- We chose the wrong power plug for the load

Lessons Learned

- It started as a ZigBee Home Automation Project, then we shifted to Thread to have more features and to learn a new protocol
- Shifting to Thread was a good move, it had some cool new features like 6LowPAN and IPV6 addressing. Which made our life easier in doing a high level application (Mobile APP)
- Designing a Power Circuit was pretty difficult, understanding the PMIC to use it to its fullest capability took some time
- Ordering the right part is a tedious task then selecting the part
- Selection of parts needed some careful considerations like the toolchains that are available and support available for the product.
- There was a good learning with the CEL Module. We modified it's Linux driver to work with raspberry pi. But we cannot be able to pair it with our EFR32MG Devices
- Mobile App Development was a new platform for both of us. It gave us a new perspective about the IoT Mobile Apps
- Making it work on individual platforms is easier, integrating it with multiple platforms into a product is a bit of a task
- Altium was a great tool and that was a great getaway from the course

- Good Firmware practices helped a lot in writing the firmware for the project
- Gantt Chart helped us realize how much work is left for a period of time. It helped us organize our schedule. We usually see Gantt chart only during those weekly report submissions. That helped us pushing the project in correct timeline
- Overall Learning thread protocol was interesting. State level programming is one more great learning , state level programming actually makes the thread reliable and self healing.
- Its good to understand the concepts by actually doing it rather than just reading about it.
- After working with thread stacks, we now know what an abstraction layer is (There is a plenty of Abstraction in the stack!).
- CoAP Protocol was a new thing we learnt in this project. It is a very new implementation for handling separate requests (Works similar to HTTP)
- When we started this project we were planning to use WGM110 WiFi Module. Its good we went ahead with Pi.

Summary

The project is working as promised with little bit of modification in the hardware. It has all the features that we mentioned in the proposal. It would have been better if we have had more kits to demonstrate mesh topology. Mobile Application development was not that difficult and it is an interesting area to work on. We will be working on this project next semester as well just to iterate over the prototype and rectify the issues that we faced this semester.

It would have been better if we had some IAR License at our Labs. We had a hard time installing trial licenses in all the new computers. Hopefully Silabs releases their GCC Compiler for thread stack soon.

Acknowledgement

We want to thank Prof.Keith Graham for his extra ordinary support throughout the project, Teaching Assistants Vivek and Sairam for helping us through issues in Altium and Mobile App development, Tim May and Lauren for helping us soldering the parts. Without their help it would have been difficult for us to complete the project in time.