ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI) GRADO EN INGENIERÍA ELECTROMECÁNICA

ESTABLISHING WI-FI CONNECTION BETWEEN STATION DEVICES AND AN ENTERPRISE SECURITY-ENABLED NETWORK

Autor: Raimundo Alonso Álvarez

Director: Khashayar Olia

Madrid 2018

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Raimundo Alonso Álvarez

DECLARA ser el titular de los derechos de propiedad intelectual de la obra: ESTABLISHING WI-FI CONNECTION BETWEEN STATION DEVICES AND AN ENTERPRISE SECURITY-ENABLED NETWORK,

que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente).*

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e

intereses a causa de la cesión.

d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

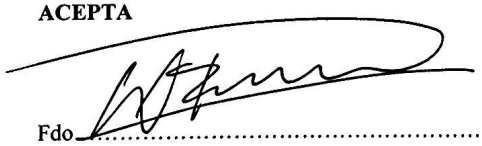
6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 15 de Julio de 2018

ACEPTA



Fdo.....

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

ESTABLISHING WI-FI CONNECTION BETWEEN STATION DEVICES AND AN ENTERPRISE SECURITY-ENABLED NETWORK

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2017/2018 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Raimundo Alonso Álvarez

Fdo.:  Fecha: 24/ 07/ 2018

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Khashayar Olia

Fdo.:  Fecha: 08/01/2018

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI) GRADO EN INGENIERÍA ELECTROMECÁNICA

ESTABLISHING WI-FI CONNECTION BETWEEN STATION DEVICES AND AN ENTERPRISE SECURITY-ENABLED NETWORK

Autor: Raimundo Alonso Álvarez

Director: Khashayar Olia

Madrid 2018

ESTABLECER UNA CONEXIÓN WI-FI ENTRE DISPOSITIVOS Y UNA RED CON SEGURIDAD DE EMPRESA

Autor: Alonso Álvarez, Raimundo

Director: Olia, Khashayar

Entidad Colaboradora: CSULA – Cal State University Los Angeles

RESUMEN DEL PROYECTO

INTRODUCCIÓN

Nuevas tecnologías se han ido desarrollando a lo largo de los últimos años, haciendo la vida más sencilla y reduciendo en gran medida la presencia del ser humano en las acciones cotidianas, dando lugar a una mayor interconexión entre los integrantes de la sociedad y la tecnología que se emplea. Este concepto ha tomado forma con el Internet de las cosas (IoT), revolucionando la relación que existe entre individuo y tecnología.

El Internet de las cosas (IoT) es un concepto que defiende la idea de una interconexión entre distintos sistemas físicos, vehículos, casas y otros sistemas integrados con Internet; de tal modo que dichos sistemas estarían interconectados entre ellos y podrían intercambiar información. Introduce una nueva forma de interaccionar con el mundo tecnológico donde todos los sistemas que componen ese mundo tienen una identidad única a la vez que pueden mostrarse al resto de dispositivos para comunicarse entre sí e intercambiar información.

El Internet de las cosas tiene distintas aplicaciones y está presente en numerosas áreas, tales como: edificios y ciudades inteligentes, control medioambiental, la medicina, negocios inteligentes, seguridad y vigilancia. Un ejemplo que refleja la presencia del Internet de las cosas es el frigorífico inteligente. Este electrodoméstico está conectado a la nube, de tal manera que cuando encuentra escasez de algún producto en su interior, informa a las tiendas correspondientes para que envíen esos productos. Se trata de un sistema físico que intercambio información con otros sistemas físicos (tiendas) a través de Internet.

Este proyecto está basado en el concepto del Internet de las cosas, en la comunicación inalámbrica de dispositivos y en el intercambio de información. A la hora de conectar dos o más dispositivos entre sí e intercambiar información se puede encontrar tecnologías como UART. UART (Universal Asynchronous Receiver-Transmitter) es el dispositivo que controla los puertos en un sistema integrado; un UART transmite la información desde el transmisor y otro UART, situado en el receptor, recibe esa información. Es un sistema que permite una transmisión tanto en paralelo como en serie y requiere de cables que conecten al transmisor y al receptor; por ello, esta tecnología no es válida para este proyecto ya que se busca una conexión inalámbrica entre dispositivos.

OBJETIVOS

El objetivo de este proyecto es conectar múltiples dispositivos Wi-Fi (móviles y estacionarios) a una red Wi-Fi en la que se utiliza un sistema con seguridad de empresa. Este proceso implica la configuración de los módulos de evaluación habilitados Wi-Fi de Texas Instruments, CCS3220SF-LAUNCHXL, para detectar la red Wi-Fi segura del campus y poder comunicarse con el servidor de seguridad para verificar la integridad de la estación y la red, dando como resultado la instalación de los certificados de la red. El objetivo del proyecto es establecer esta conexión segura entre los dispositivos (móviles y estacionarios) y la red segura, y después determinar la calidad de comunicación de los sistemas móviles. La manera de comprobar que la calidad de la comunicación sea correcta es mostrando en el terminal una salida si se obtiene una entrada válida.

RECURSOS

Acerca de los recursos empleados se diferencia entre hardware y software. El hardware empleado son dos CC3220SF-LAUNCHXL de Texas Instruments, estos dispositivos son la última actualización de la familia CC3xxx que además tienen integrados capacidades Wi-Fi. Dicho dispositivo consiste en un microcontrolador inalámbrico de un único chip con 1 MB de memoria Flash, 256 KB de memoria RAM y características para llevar a cabo aplicaciones relacionadas con el Internet de las cosas.

Acerca del software empleado en este proyecto, se utiliza una plataforma llamada Code Composer Studio (CCS). Se trata de un entorno de desarrollo integrado para desarrollar, programar, aplicaciones que son usadas por los dispositivos de Texas Instruments. Además, se utiliza otra plataforma que se llama Uniflash; su uso se debe a

que en una parte en particular del proyecto suponía de gran utilidad usar dicha plataforma (permite "flashear" al mismo tiempo en la memoria del controlador tanto el código del programa como los certificados necesarios para ejecutar la aplicación). Cabe puntualizar que el código empleado se basa en el TI-RTOS que supone un sistema operativo en tiempo real; y destacar también, la ayuda proporcionada por el equipo de técnicos de Texas Instruments la cual ha sido posible a través de su página web (https://e2e.ti.com/).

METODOLOGÍA

Para llevar a cabo el proyecto se ha seguido una estrategia compuesta por 4 puntos o guías fundamentales para lograr los objetivos establecidos. Hay que tener en cuenta que tanto los dispositivos de la familia CC3xxx de Texas Intruments como el software empleado eran de carácter desconocido para los integrantes del proyecto antes del mismo; por ello, las guías establecidas parten de una toma de contacto tanto del hardware como del software para un entendimiento profundo de ambos. Estas 4 directrices establecidas son:

1. Toma de contacto con el hardware (CC3220SF-LAUNCHXL) y el software empleado: para ello se va a llevar a cabo la demostración "Out of box", en la cual se usa el hardware como un punto de acceso al que se va a conectar un ordenador (MacBook Air) que controlará la demostración. Esta demostración cosiste en encender y apagar un LED y en obtener, mediante coordenadas, la posición en el espacio de la placa.

2. Probar y profundizar en las capacidades Wi-Fi del hardware: para ello se va efectuar un "Wi-Fi provisioning" (aprovisionamiento de Wi-Fi) donde la placa opera como estación en búsqueda de puntos de acceso a los que conectarse. Para llevar a cabo el aprovisionamiento, se empleará una aplicación de móvil desarrollada por Texas Intruments llamada “SimpleLink Wi-Fi Starter Pro” mediante la cual se controla a que punto de acceso se desea conectar.

3. Conectar el hardware a una red inalámbrica con seguridad de empresa: para ello se tomará como base el código de la demostración de MQTT que proporciona Texas Instruments y se modificará para lograr el objetivo deseado.

4. Intercambio de información entre dos CC3220SF-LAUNCHXL y comprobar la calidad de dicha comunicación: para ello se usará el "Message Queuing Telemetry Transport" (MQTT) que consiste en un protocolo de comunicación basado en publicar y suscribirse a temas. Al suscribirte a un tema, todo aquello que sea publicado en ese tema será recibido por el dispositivo subscrito a él.

RESULTADOS Y CONCLUSIONES

La primera propuesta planteada a la hora de conectar el hardware a una red con seguridad de empresa fue el "aprovisionamiento de Wi-Fi" (Wi-Fi provisioning) pues funcionaba perfectamente para conectar la placa a redes con seguridad personal. Sin embargo, tuvo que ser descartada debido a que para conectar a una red de seguridad de empresa se requiere una serie de certificados (tres en total, para la autentificación del cliente y del servidor) que tienen que ser proporcionados por el departamento IT de la empresa, en este proyecto la empresa es la universidad Cal State University LA. El éxito del "aprovisionamiento Wi-Fi" para conectar a una red de seguridad personal se debe a la no necesidad de ningún certificado especial, sino unicamente una serie de certificados básicos proporcionados por Texas Instruments.

El departamento IT de Cal State University LA no proporciona los certificados requeridos citados anteriormente, sino que proponen una conexión basada en un determinado método de autentificación, EAP TTLS MSCHAPv2. Este método de autentificación en vez de requerir certificados únicamente requiere los datos de la cuenta personal de la universidad. Para implementar dicho método, se necesita realizar la conexión hardware-red del campus de manera manual, por lo que el "aprovisionamiento Wi-Fi" que realiza la conexión de manera automática se descartó.

A la hora de implementar el método de autentificación propuesto por el departamento IT, EAP TTLS MSCHAPv2, ha sido necesario deshabilitar la autentificación de servidor. En principio, esto no era necesario ya que al utilizar el método EAP TTLS MSCHAPv2 ningún certificado es requerido; sin embargo, es la única manera de poder realizar la conexión a la red y supone un problema: el hardware se conectará y enviará los datos requeridos para la conexión a cualquier punto de acceso con el mismo nombre que el

punto de acceso de Cal State University LA ya que no va a verificar la verdadera identidad del punto de acceso.

El mensaje que las placas mandan y reciben consta de 12 variables: un ID para identificar el mensaje, el año en el que se emite, un ID de la región, un contador y 8 variables de data. La forma de establecer si la calidad comunicación entre dispositivos es correcta es la siguiente: si la Placa1 recibe un mensaje de la Placa2 con un ID válido (240 para la Placa1) debe mostrar en la terminal las 8 variables de data y si el ID es distinto de 240 debe mostrar la frase: "No message"; el mismo procedimiento se utiliza para la PLaca2, si recibe un mensaje de la Placa1 con un ID valido (241 para la Placa2) mostrará las 8 variables de data del mensaje y sino el ID no es 241 mostrará "No message".

La manera de llevar a cabo la comunicación entre las dos placas es usando un protocolo MQTT (Message Queuing Telemetry Transport). De esta manera, la Placa1 va a estar subscrita a los temas en los cuales la Placa2 publica para poder recibir el mensaje que la Placa2 emite y viceversa, la Placa2 estará subscrita a los temas en los que la Placa1 publica. Como el objetivo es únicamente mostrar las 8 variables de data, cada placa va a estar subscrita a 4 temas: Board1_ID (Placa1) o Board2_ID (Placa2) donde se publica el ID del mensaje, Board1_Data1 (Placa1) o Board2_Data1 (Placa2) donde se publica las 4 primeras variables de data, Board1_Data2 (Placa1) o Board2_Data2 (Placa2) donde se publica las 4 últimas variables de data, Board1_Data3 (Placa1) o Board2_Data3 (Placa2) donde se publica el año, la región y el contador.

El resultado fue óptimo pues se logró el objetivo. La Placa1 mostraba en el terminal las variables de data de aquellos mensajes con valor de ID 240 y mostraba “No message” si recibía mensajes con un ID distinto. Lo mismo ocurría con la Placa2 y el ID 241. Con este proyecto queda demostrada la eficacia del protocolo MQTT como medio de comunicación entre placas abriendo un gran abanico de posibles aplicaciones ya que no hay un número límite de placas que se puedan subscribir a un tema o un numero límite de temas a los que se pueda suscribir pudiendo conectar varias placas a un mismo tema si se quiere mandar un mismo mensaje a todas las placas y; a la vez, que cada una este subscrita a un tema por si se quiere mandar un mensaje determinado a una placa determinada. Como se puede apreciar, el número de posibles aplicaciones es ilimitado.

ESTABLISHING WI-FI CONNECTION BETWEEN STATION DEVICES AND AN ENTERPRISE SECURITY-ENABLED NETWORK

Author: Alonso Álvarez, Raimundo

Director: Olia, Khashayar

Collaborating Entity: CSULA – Cal State University Los Angeles

PROJECT SUMMARY

INTRODUCTION

New technology has been developed in the last years, making the life easier and minimizing the human presence in the daily actions, resulting on a stronger connection between society and the technology used. This concept has become real with the Internet of Things (IoT) which is revolutionizing the way in which people connect to technology.

The Internet of Things (IoT) is a computing concept that defines the idea of physical devices, vehicles, homes appliances and other embedded items being connected to the internet which enable them to communicate to each other and exchange data. It is a new way of connection between humans and the technological environment where the items that form part of this environment are uniquely identifiable through their internal embedded system but, at the same time, they can show themselves to other devices, so they can connect and exchange data.

The Internet of Things have many different areas of application and in every one of them it has a different role. The main areas or markets where IoT plays or will play an important role on are: smart cities and buildings, environmental monitoring, health-care, smart business, security and surveillance. One example that represents the concept of the Internet of Things is the smart fridge. This fridge is connected to the cloud where it can interoperate and exchange data with the different stores when it is running out of stock; it is physical device that is connected to and exchanges data with other physical devices (stores) through the Internet.

This project is based on the concept of the Internet of Things (IoT), the wireless communications and the exchange of data between devices. When looking for technology that enables the connection and communication between two or more devices, the UART

technology comes out. UART (Universal Asynchronous Receiver-Transmitter) is a computer hardware device that monitors the communication ports of an embedded system, the UART placed in the transmitter sends the data and another UART, placed in the receiver, collects this data. This technology allows either a serial communication or a parallel communication, needing a wire-connection between devices; this is why this technology was discarded as one of the main objectives of this project is to fulfill a wireless connection between devices and an enterprise security network.

OBJECTIVE

The aim of the project is to connect multiple Wi-Fi devices (rover and stationary) to a Wi-Fi network in which an enterprise security system is used. This process involves setting up the Texas Instruments Wi-Fi enabled evaluation modules, CC3220SF-LAUNCHXL, to detect the on-campus secure Wi-Fi network then communicating with the security server to verify the integrity of the station and the network which results in installing the network certificates. The goal of the project is to stablish this secure connection between the stations (rover and stationary) and the secure network, then determine the communication quality of rover systems. The way to determine the communication quality would be to display an output if the system receives the correct input.

RESOURCES

About the resources used in this project, it is important to differentiate between the hardware and the software. The hardware consists of two CC3220SF-LAUNCHXL provided by Texas Instruments, these devices are the last update of the CC3xxx family with Wi-Fi capacities. CC3220SF is a single-chip wireless microcontroller with 1 MB Flash, 256 KB RAM and features to carry out Inter of things applications.

About the software, Code Composer Studio (CCS) is used which is an integrated development environment (IDE) that supports TI's microcontrollers and embedded processors portfolio. In addition, another development environment called Uniflash can be found in this project, it's use was specifically for one part of the project as it allows the user to flash the board with the code of the project at the same time that the certificates needed for that project are flashed. The programming language is C++, using the TI-

RTOS which is a real-time operating system and it is important to mention all the help that the experts of Texas Instruments have provided through their webpage (https://e2e.ti.com/).

METHODOLOGY

In order to correctly fulfill the objective of this project a strategy plan made out of four main points has been followed. It is important to take into account the little previous experience in terms of the hardware used or any of the devices of the CC3xxx family provided by Texas Instruments and the software used; that is why the strategy plan starts by getting to know the hardware and the software to deeply understand how they work. The four main points that have been set up are:

1. Get to know the hardware (CC3220SF-LAUNCHXL) and the software: Troubleshoot the Out of box demonstration setting up the hardware as an access point and connecting to a laptop (MacBook Air) in order to monitor the demonstration. The demonstration is based on blinking a LED and getting the location of the board by coordinates.
2. Get to know the Wi-Fi features of the hardware: Carry out a Wi-Fi provisioning setting up the hardware as station and using an iPhone application called "SimpleLink Wi-Fi starter pro" provided by Texas Instruments to monitor the access points that the board is connecting to.
3. Connect the hardware to an enterprise security network: The code used is based on the one found in the MQTT demonstration provided by Texas Instruments; that code has been modified following the interests of the project.
4. Exchange of data between two CC3220SF-LAUNCHXL and determine the quality of the communication: Using the MQTT (Message Queuing Telemetry Transport) based on a public/subscribe communicating protocol. Basically, the hardware is subscribed to a topic and everything that is published in that topic will be received by the hardware.

RESULTS AND CONCLUSIONS

The first proposal for the connecting to an enterprise security network was a Wi-Fi provisioning as it worked for a personal security network. However, it was discarded as some certificates were needed (three in total; server root CA file (server authentication), private key and client certificate (client authentication)); these certificates had to be provided by the IT department of the enterprise (Cal State LA university) you are trying to connect to. The reason why the Wi-Fi provisioning works for personal security network connections is because those networks do not require any special certificate, they work perfectly with the dummy certificates that Texas Instruments provide for the Wi-Fi provisioning demonstration.

The Cal State LA university IT department does not provide the certificate, instead they propose to use EAP TTLS MSCHAPv2 authentication method where no certificates are required and it works by using the Cal State account credentials. To implement this method a manual connection is required so the Wi-Fi provisioning was discarded. Also, disabling the server authentication has been required; in theory, this was not necessary as the EAP TTLS MSCHAPv2 authentication method does not need any certificate but it was the only way to carry out the connection between the hardware and the enterprise security network. However it has a downside which is that the board will connect and send the Cal State account credentials to any access point with the same name as the one used by Cal State LA university as it does not verify the real identity of the access point.

The message that the boards are sending and receiving is composed of 12 variables: an ID value, the year of the message, a region ID, a counter and 8 data variables. To determine the quality of the communication between devices, Board1 will display the data part (8 data variables) of the received message with the correct ID value (240 for Board1) and will display the sentence “No message” if it receives a message with an ID different from 240; the same thing happens with Board2, it will display the data part of the received message if it has the right ID value (241 for Board2), otherwise it will display the sentence “No message”.

To communicate between boards and exchange data the MQTT (Message Queuing Telemetry Transport) protocol was used. Board1 will be subscribed to the topic in which Board2 is publishing to so Board1 will receive any message published by Board2

and the same happens with Board2: it will be subscribed to the topics in which Board1 is publishing. The goal is to display the data variables if the ID value is the correct one, so it has been nesessary to set up 4 topics: Board1_ID (Board1) or Board2_ID (Board2) where the ID value of the message will be published, Board1_Data1 (Board1) or Board2_Data1 (Board2) where the first four data variables will be published, Board1_Data2 (Board1) or Board2_Data2 (Board2) where the last four data variables will be published, Board1_Data3 (Board1) or Board2_Data3 (Board2) where the year, region ID and counter will be published.

The result of the project was the one expected. Board1 displays in the terminal the data variables when it receives a message from Board2 with a valid ID (240) and "No message" if it receives a message with a different ID value and the same with Board2. This project is a way to demonstrate the efficiency of the publish/subscribe topics method (MQTT) as a communication protocol between devices and it opens the door to an entire world of applications. There is no limit number of boards you can subscribe to a topic and no limit number of topics a board can subscribe to allowing you to subscribe different boards to the same topic if you want to send the same message to a big number of boards and, at the same time, subscribe each board to independent topics if you want to send a message to an unique board. The most interesting conclusion is not the result itself but all the different applications and uses that can come out from this project.

INDEX

PART 1: MEMORY

Chapter 1: Hardware 28

1.1 OVERVIEW 28

1.2 FEATURES 29

1.2.1 Applications Microcontroller Subsystem 30
1.2.2 Wi-Fi Network Processor (NWP) Subsystem 30
1.2.3 Power-Management Subsystem 32
1.2.4 Other Features 33

1.3 MODES 33

1.3.1 Station (STA) 34
1.3.2 Access Point (AP) 34
1.3.3 Wi-Fi Direct 35

Chapter 2: Software 37

2.1 CODE COMPOSER STUDIO 37

2.2 UNIFLASH 40

2.3 SOFTWARE DEVELOPMENT KID (SDK) 41

2.4 TI-RTOS 42

Chapter 3: Out of Box Demo 45

3.1 DESCRIPTION 45

3.2 SET UP AND POWER THE BOARD 45

3.3 CONNECT MY LAPTOP TO CC3220 46

3.4 BROWSE THE ON-BOARD WEBSITE 47

3.5 RUN THE DEMONSTRATIONS 48

Chapter 4: Wi-Fi Provisioning 52

4.1 OVERVIEW 52

4.2 CERTIFICATES 53

4.3 CONNECTING TO PERSONAL SECURITY NETWORK 53

4.4 CONNECTING TO ENTERPRISE SECURITY NETWORK 57

4.5 MESSAGE 58

Chapter 5: MQTT 61

5.1 OVERVIEW 61

5.2 TOPICS 65

Chapter 6: Message and Server 68

6.1 MESSAGE 68

6.2 SERVER 69

6.3 PUBLISH MESSAGE 70

6.4 RECEIVING MESSAGE 71

Chapter 7: Conclusion 75

Chapter 8: Other Applications 78

8.1 ANOTHER WAY 78

8.2 OTHER APPLICATIONS 81

References 83

PART 2: APPENDIX

Appendix A: Tables 88

Appendix B: Block Diagrams 89

Appendix C: Project Overview 91

INDEX OF FIGURES

PART 1: MEMORY

Figure 1: Front of the CC3220SF-LAUNCHXL 28
Figure 2: Back of the CC3220SF-LAUNCHXL 29
Figure 3: Code Composer Studio interface 38
Figure 4: Resource Explorer in CCS 39
Figure 5: Uniflash's interface 40
Figure 6: Kernel's Scheduler 43
Figure 7: Board's schema 46
Figure 8: Board's network as an AP 47
Figure 9: Interface of "My SimpleLink" webpage 48
Figure 10: LED D10 turned on 49
Figure 11: LED D10 turned off 49
Figure 12: Accelerometer. First position 50
Figure 13: Accelerometer. Second position 50
Figure 14: Accelerometer. Third position 51
Figure 15: Configuration Page 54
Figure 16: Access Point browser 55
Figure 17: AP set up parameters 55
Figure 18: Provisioning parameters 56
Figure 19: Wi-Fi provisioning terminal output 56
Figure 20: MQTT schema 61
Figure 21: Set up security parameters 63
Figure 22: Disable server authentication 64
Figure 23: Subscribed topics Board 2 65
Figure 24: Subscribed topics Board 1 66
Figure 25: Message's structure publish 68
Figure 26: Message variable's conversion 69
Figure 27: Sever's parameters 69
Figure 28: Publish message function. First part 70
Figure 29: Publish message function. Second part 70
Figure 30: Receiving function 71
Figure 31: Copying into the buffer 72
Figure 32: Compare's variables. Board 1 72
Figure 33: Compare's variables. Board 2 72
Figure 34: Storing the received value 73
Figure 35: Display function 74
Figure 36: Output Board 1 75
Figure 37: Output Board2 76
Figure 38: Limit topics variable 79

Figure 39: Other application's schema 82

PART 2: APPENDIX

Figure 40: CC3220x Hardware overview 89
Figure 41: CC3220x Embedded Software overview 90
Figure 42: Project schema 91

INDEX OF TABLES

PART 1: MEMORY

Table 1: Station mode default parameters 34
Table 2: AP mode default parameters 35
Table 3: Wi-Fi Direct mode default parameters.....36
Table 4: Kernel's key components 44
Table 5: Message's structure 60
Table 6: Topic's organization 67
Table 7: Topic's organization. Another alternative 80

PART 2: APPENDIX

Table 8: Acronyms.....88

PART 1: MEMORY

Chapter 1: Hardware

1.1 OVERVIEW

The hardware used in this project is two CC3220SF-LAUNCHXL Wi-Fi enabled evaluation modules provided by Texas Instruments. The SimpleLink™ Wi-Fi® CC3220SF LaunchPad™ development kit (CC3220SF-LAUNCHXL) is a single-chip wireless microcontroller (MCU) with 1 MB of Flash Memory and 256 KB of RAM. This boards can be directly connected to a PC using a USB wire. It has two separate execution environments: an application processor ARM® Cortex®-M4 MCU with an optional 1 MB of XIP flash and a network processor MCU to run all Wi-Fi and Internet Logical Layers.

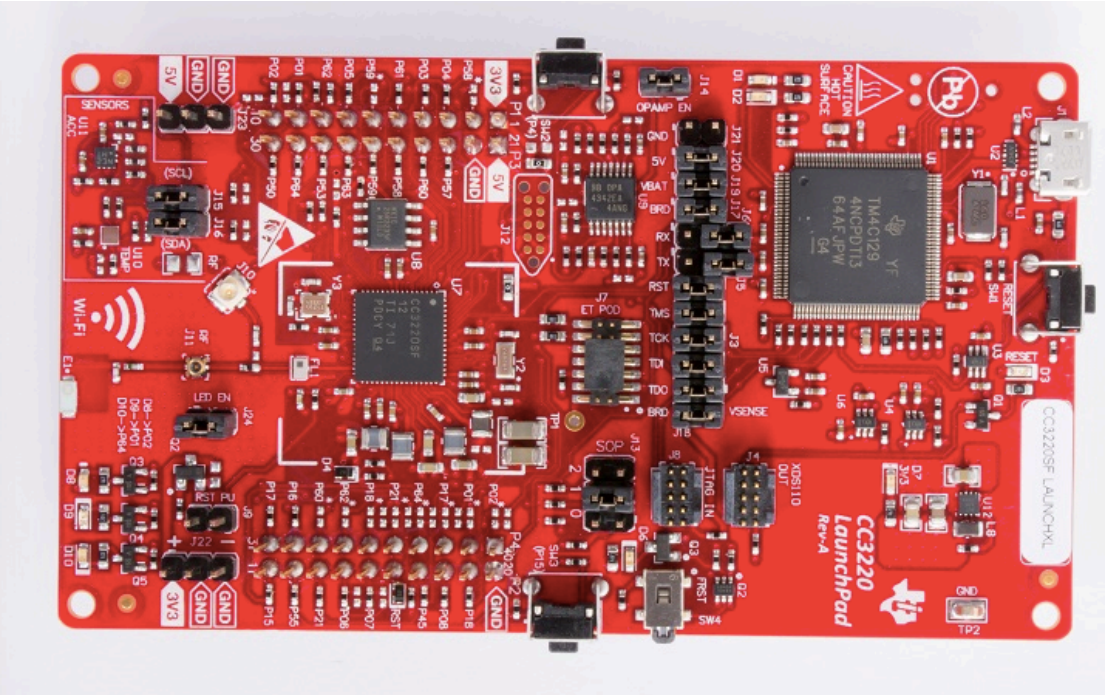


Figure 1: Front of the CC3220SF-LAUNCHXL

The CC3220x is the latest version of the Internet-on-a-chip™ family that Texas Instruments have developed, it includes some new capabilities compared to the other CC3200x families. These new features are: enhanced Wi-Fi provisioning, power consumption and file system security; Wi-Fi Access Point (AP) connection up to four stations; more concurrently opened BSD sockets (up to 16 BSD sockets with 6 secure ones); HTTPS support, RESTful API support; asymmetric keys crypto library.

Figure 2: Back of the CC3220SF-LAUNCHXL

1.2 FEATURES

All these features are stablished by Texas Instruments and can be found in the datasheet of the CC3220SF-LAUNCHXL. They are organized by the most important areas of application: Microcontroller, Wi-Fi network and Power management. In my project, I am working with the CC3220SF-LAUNCHXL, so I would only pay attention

to the features of my hardware not to all the CC3220 family such as CC3220R and CC3220S.

1.2.1 Applications Microcontroller Subsystem

The features related to the ARM® Cortex®-M4 MCU are the following:

- ARM® Cortex®-M4 at 80 MHz
- Embedded Memory
    - Flash-based wireless MCU with integrated 1MB of flash and 256 KB of RAM.
    - External serial flash
- McASP supports two I2S channels
- SD, SPI, I^2C and UART
- 8-Bit parallel camera
- Four general-purpose timers with 16-Bit PWM mode
- Watchdog timer
- 4-channel 12-Bit ADCs
- Up to 27 GPIO pins
- Debug Interfaces: JTAG, cJTAG, SWD

1.2.2 Wi-Fi Network Processor (NWP) Subsystem

One of the main feature and innovation of the CC3220x family is the Wi-Fi capabilities that it has. These Wi-Fi features are the following:

- Wi-Fi Internet-on-a-chip™ dedicated ARM MCU completely offloads Wi-Fi and Internet protocols from the application MCU
- Wi-Fi modes:
    - 802.11 b/g/n Station
    - 802.11 b/g/n Access Point supports up to four stations

    - Wi-Fi Direct® client and group owner
- WPA2 personal and enterprise security: WEP, WPA/WPA2 PSK, WPA2 enterprise (802.1x).
- IPv4 and IPv6 TCP/IP stack
- Industry-Standard BSD socket application programming interfaces (APIs)
    - 16 simultaneous TCP or UDP sockets
    - 6 simultaneous TLS and SSL sockets
- IP addressing: Static IP, LLA, DHCPv4, DHCPv6 with DAD
- SimpleLink connection manager for autonomous and fast Wi-Fi connections
- Flexible Wi-Fi provisioning with SmartConfig™ technology, AP mode and WPS2 options
- RESTful API support using the internal HTTP server
- Embedded network applications running on dedicated network processor
- Hardware features:
    - Separate execution environments and device identity
    - Hardware crypto engine for advanced fast security, including: AES, DES, 3DES, SHA2, MD5, CRC and Checksum
    - Secure sockets (SSLv3, TLS1.0, TLS1.1, TLS1.2).
    - Initial secure programming:
        - Debug security
        - JTAG and debug ports are locked
- Networking security:
    - Personal and enterprise Wi-Fi security
    - HTTPS server
    - Trusted Root-Certificate catalog
    - TI Root-of-trust
- SW IP protection:
    - Secure key storage and file system security
    - Software tamper detection
    - Cloning protection
    - Secure boot: validate the integrity and authenticity of the runtime binary during boot
- Embedded network applications running on the dedicated network processor

    - HTTP/HTTPS web server with dynamic user callbacks
    - MDNS, DNS-SD, DHCP server
    - Ping
- Recover mechanism-following the right steps can recover to factory defaults or to the factory image
- Wi-Fi TX power
    - 18.0 dBm @ 1 DSSS
    - 14.5 dBm @ 54 OFDM
- Wi-Fi RX sensitivity
    - -96 dBm @ 1 DSSS
    - -74.5 dBm @ 54 OFDM
- Application throughput
    - UDP: 16 Mbps
    - TCP: 13 Mbps

1.2.3 Power-Management Subsystem

The management of the power and how to optimize its use does not have a great importance in this project, but it is necessary to highlight the main features of the CC3220SF-LAUNCHXL in terms of power consumption and management. The most important features are the following:

- Integrated DC-CD converters support a wide range of supply voltage:
    - VBAT wide-voltage mode: 2.1 V to 3.6 V
    - VIO is always tied with VBAT
    - Preregulated 1.85-V mode
- Advanced low-power modes:
    - Shutdown: 1 µA
    - Hibernate: 4.5 µA
    - Low-power deep sleep (LPDS): 135 µA (Measured with 256-KB RAM Retention)
    - RX traffic (MCU active): 69 mA @ 54 OFDM
    - FX traffic (MCU active): 238 mA @ OFDM; Maximum power

- Idle connected in LDPS: 710 µA @ DTIM = 1

1.2.4 Other Features

Other features that are not part of the main groups are:

- Clock source
  - 40.0-MHz crystal with internal oscillator
  - 32.768-kHz crystal or external RTC
- RGK Package
  - 64-pin, 9 mm x 9 mm very thin quad flat nonleaded (VQFN) package, 0.5-mm pitch
- Operating temperature
  - Ambient temperature range: -40°C to +85°C
- Device supports SimpleLink developer's ecosystem

1.3 MODES

As it was mentioned before as part of the features of the Wi-Fi network processor subsystem, the CC3220SF-LAUNCHXL can work in three different modes and the usage of one or another depends on the objective you want to achieve. For example, if you want to troubleshoot the out of box demo, the board should wake up as an access point, so you can connect to it. There are three main modes: Access Point (AP), Station (STA) and Wi-Fi Direct.

1.3.1 Station (STA)

The Station mode is the most common operating mode for the CC3220SF-LAUNCHXL and allows the device to scan and then connect to any access point near it. Once the hardware is connected, it obtains an IP address and starts to send and receive data over the network to other devices. When the board wakes up in station mode it has some default parameters that can be reconfigured, this default settings are (the table below is provided by Texas Instruments):

Configuration	Default value
Interface	IPv4
Address	DHCP
STA TX Power	0 (no back-off, maximum TX power)
Country Code	EU (channels 1-13)
Connection policy	Auto and Auto provisioning
Calibration Mode	Normal
Server enterprise authentication	Enabled
Applications	HTTP server and MDNS

Table 1: Station mode default parameters

1.3.2 Access Point (AP)

The AP is used to set the device network configuration, the device wakes up as an access point with an SSID previously defined by the equipment manufacturer. When the device is working as an access point it creates a network by its own allowing other devices such as a PC or a smartphone to connect to its network directly and facilitate its initial configuration. The CC3220SF-LAUNCHXL can support up to four stations connecting to it when it is in access point mode, ensuring that those connections are secured.

Configuration	Default value
Interface	IPv4
Address	Static with the following parameters: IP 10.123.45.1, Subnet mask:255.255.255.1, Default Gateway: 10.123.45.1, DNS: 10.123.45.1
AP TX Power	0 (no back-off, maximum TX Power)
Country Code	EU (1-13 channels), default channel is 6
Connection policy	N/A
Calibration Mode	Normal
Applications	DHCP server and HTTP server and MDNS and DNS server

Table 2: AP mode default parameters

1.3.3 Wi-Fi Direct

The CC3220SF-LAUNCHXL supports the Wi-Fi Direct standards which enables the board to connect to other devices without being connected to an access point. The way this mode works is that one device works as a group owner (AP-like mode) and the other device works as a client (STA-like mode) by inheriting the entire STA and AP attributes. It also has some default parameters that can be summarize in Table 3 provided by Texas Instruments, these default values are:

Configuration	Default value
Interface	IPv4
STA TX power	0 (no back-off, maximum TX power)

Country code	EU (channels 1-13)
Connection policy	Auto and Auto provisioning
Calibration mode	Normal
Applications	HTTP server
Intent	3
Negotiator	2
CL address	DHCP
GO address	Static with the following parameters: IP 10.123.45.1, Subnet mask: 255.255.255.1 Default gateway: 10.123.45.1, DNS: 10.123.45.1
Device name	mysimplelink_XX (xx = Random 2 characters)
Device type	1-0050F204-1
Listen channel	Random channel between 1, 6 or 11
Operational channel	Random channel between 1,6 or 11

Table 3: Wi-Fi Direct mode default parameters

Chapter 2: Software

2.1 CODE COMPOSER STUDIO

Code Composer Studio (CCS) is an Eclipse based integrated development environment (IDE) that supports Texas Instruments microcontrollers and embedded processors portfolio. This development environment includes an entire pack of tools used to develop, build and flash different types of embedded applications; it also works with an optimizing C/C++ compiler, source code editor, project build environment, debugger, profiler, and many other features. As it is said by Texas Instruments *"Code Composer Studio combines the advantages of the Eclipse software framework with advanced embedded debug capabilities from TI resulting in a compelling feature-rich development environment for embedded developers"*.

In this project, I am using Code Composer Studio version 8. The way this environment works is very intuitive, providing some start guides and videos that will help the user to understand its performance. CCS can be used for different processor families such as MSP low power MCUs, C2000 Real-Time MCUs, TM4x MCUs, TMS570 & RM4 Safety MCUs, Sitara (Cortex A & ARM9) processor, Multicore DSP and ARM including KeyStone processor, F24x/C24x devices, C3x/C4x DSPs and SimpleLink wireless MCUs which is the one used in this project.

Code Composer Studio comes up with some new features as an integrated development environment. These features I am talking about are: Eclipse concepts, getting started view, resource explorer, app center and task view. CCS tries to simplify the main Eclipse concepts through its workbench and workspace making it easier for the user to work with different projects and to store references to all of them even if the projects themselves do not physically reside inside the workspace folder. Other features related to the Eclipse concepts are the perspective, the view, the resource and the project (project explorer) that CCS uses.

The getting started view provides the user a video explaining the main basic concepts of CCS and how to start a new project, browse examples, how to import a project. It also gives you access to the Code Composer Studio wiki that contains additional product documentation and application notes and it also offers a forum, e2e supports forum, where you can get technical advice and support. I need to highlight the work of this forum that Texas Instruments has because of the help I have received from their experts through the whole project, as an amateur user they have explained me some basics related to the CC3220SF-LAUNCHXL, its performance, Wi-Fi provisioning and MQTT, becoming an important part in the development of the project.

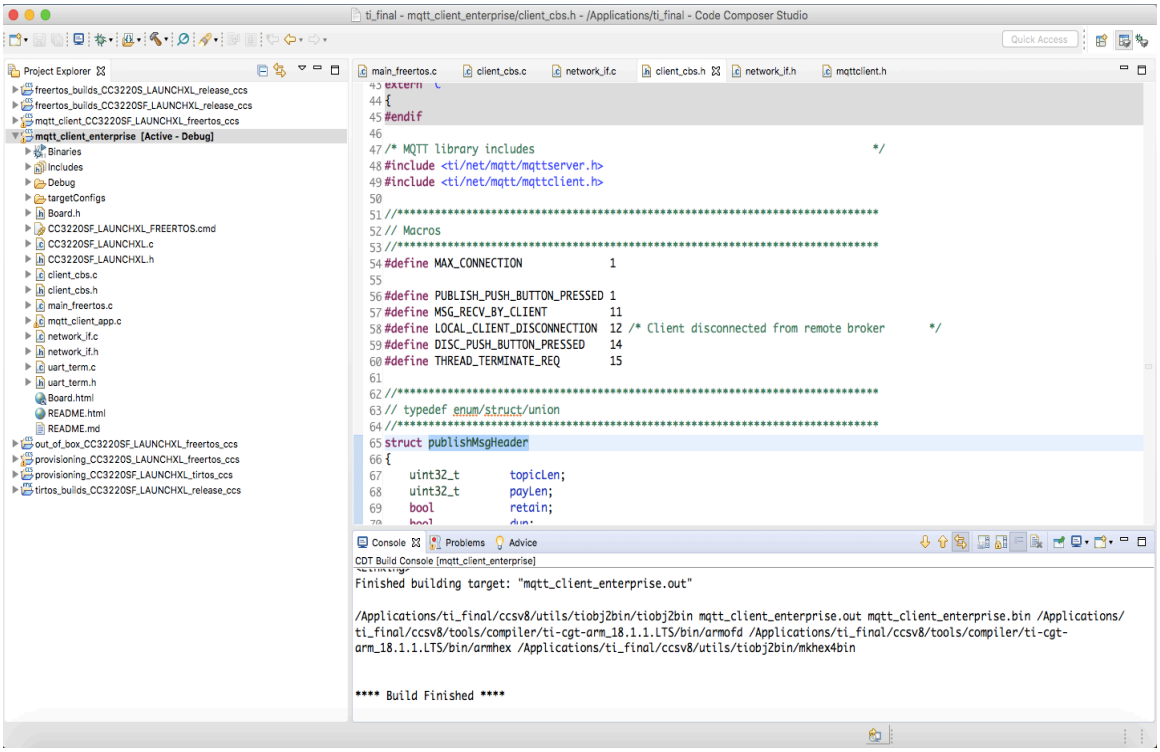


Figure 3: Code Composer Studio interface

The resource explorer is the feature that I have used the most in this project. It has the function of a browser where you can find all kind of demo applications and examples, libraries, datasheet filtering by device. You will all the documentation from the device you are using not only the resources you have already installed but those that are in the cloud giving the user the chance to download them. It can operate in two modes: the online one, by default, that means that is connected to the cloud, so it will also show the

user the content in the cloud and the offline that will just show the resources that the user has already installed.

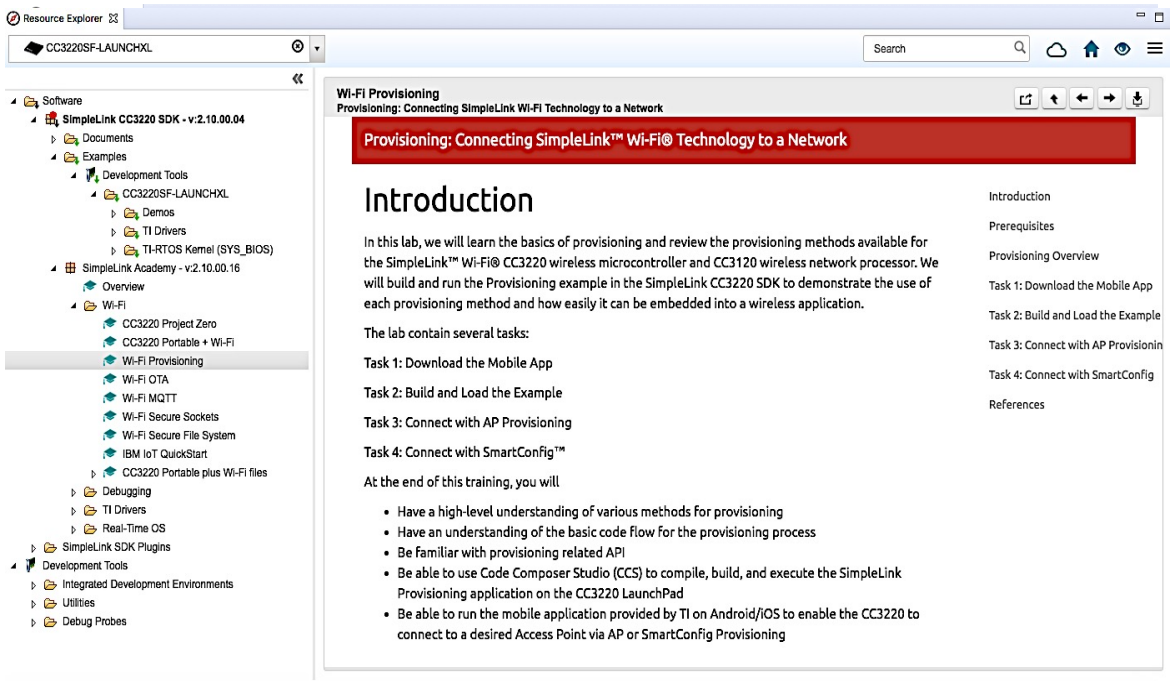


Figure 4: Resource Explorer in CCS

To do meaningful development, CCS provides the user an App center where additional resources can be found. These resources I am referring to are: Code Composer Studio Adds-on (managed tools), Standalone software (software packages that are available but not managed) and additional resources such as websites and cloud-based development tools.

The last feature of the CCS is the task view that allows the user to keep track of 'To-Do'(or Tasks) list. You can spot different variables or see their results showing them in the editor after some functions and it can also be used to track parts of the code through all the files your project can have. In this project, it has been very useful to correct some errors, thanks to the task view I could spot the right variables that made me have an error and ask the e2e forum for help or look by myself the meaning of the different values that the variables I was tracking could have.

2.2 UNIFLASH

In the last section, I have talked about the App center which is one of the features of the Code Composer Studio and provides additional resources to do a meaningful development such as standalone software. CCS Uniflash is a standalone tool used to program on-chip flash memory on TI MCUs and on-board flash memory for Sitara processors. It supports an extend variety of devices: CC31xx, CC25xx, CC26xx, CC3220xx, CC3120, Tiva, CC2000, MSP43, Hercules, PGA9xx, IWR14xx, IWR16xx, AWR14xx, AWR16xx.

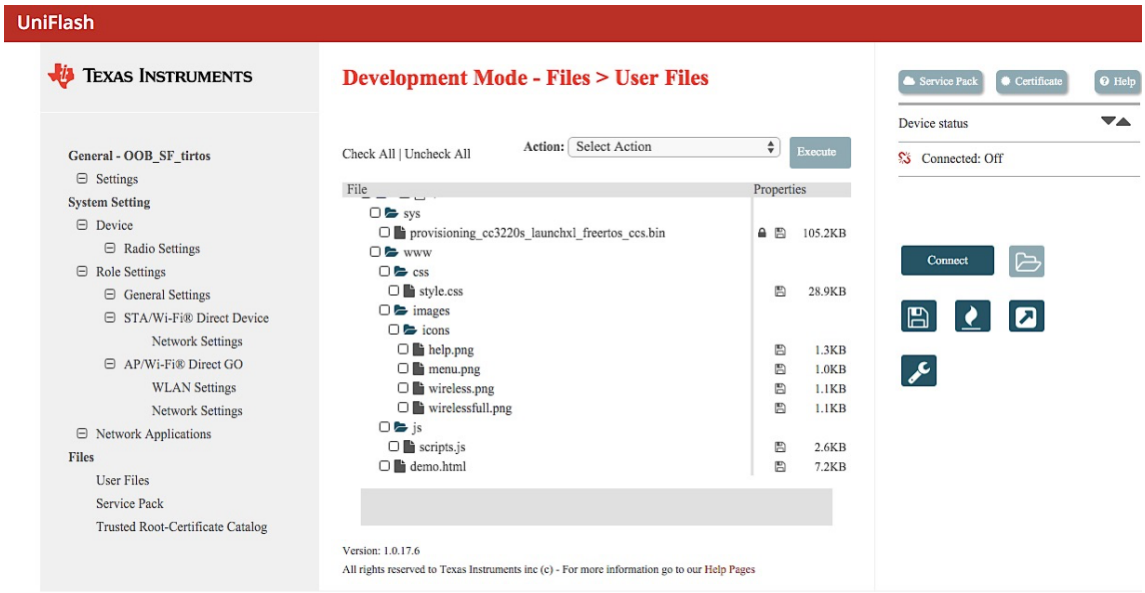


Figure 5: Uniflash's interface

One of the main advantages of the Uniflash is that you can flash the certificates you need while you flash the MCU Image at the same time, which is different from the Code Composer Studio where you need to have a separate file with the certificates required that is called by the user in the main file of the project. In this project, I have used Uniflash in the Wi-Fi provisioning part where some certificates were needed to correctly run the application.

2.3 SOFTWARE DEVELOPMENT KID (SDK)

Other software used in this project that can be treated as an addition for the CCS is the SimpleLink™ Wi-Fi® CC3220 Software Development Kit (SDK). This software is full of drivers, examples and demos for the CC3220SF-LAUNCHXL and the documentation needed to troubleshoot the examples and demos. All the examples and demo applications are supported on the integrated Cortex™-M4 with CCS IDE and no RTOS but only a few of them are supported on IAR, FreeRTOS and TI-RTOS. It contains the flash programmer, a command line tool for flashing software, configuring network and software parameters (SSID, access point channel, network profile, etc.), system files, and user files (certificates, web pages). The most important features that can be highlighted are:

- Internet-on-a-chip sample applications
    - Email from SimpleLink Wi-Fi solution
    - Information center: get time and weather from the Internet
    - Https server: host a secure web page on SimpleLink Wi-Fi solution
    - XMPP: IM chat client
    - Serial interface

- Wi-Fi sample applications
    - Easy Wi-Fi configuration
    - Station, AP modes
    - TCP/UDP
    - Security-Enterprise/Personal, TLS/SSL
    - Power management-Deep sleep, hibernate

- MCU peripheral sample applications
    - Including parallel camera, I2S audio, ADC, I2C, PWMs, JTAG Flashing and more

2.4 TI-RTOS

As Texas Instruments has quoted from wikipedia.org, "A real-time operating system (RTOS) is an operating system (OS) intended to serve real-time application process data as it comes in, typically without buffering delays". The most important point of RTOS is that it is real-time so the latencies (interrupt and thread switching) are minimal, the fact that RTOS is quick and predictable has more value than the amount of work it can perform. We can identify five main goals of the Real-Time Operating System:

1. Small latency: it is because RTOS is real time after all
2. Determinism: it is also due to the real time. You know how long things are going to take because is real time, so you can meet your deadlines
3. Structured software: RTOS is able to divide and conquer in a structure manner, adding components.
4. Scalability: RTOS must be able to scale from the simplest application to a more complex one with more things
5. Offload development: An RTOS manages many aspects of the system which allows a developer to focus on their application

As Texas Instruments says, TI-RTOS accelerates development schedules by eliminating the need to create basic system software functions from scratch scaling from a minimal footprint real-time multitasking kernel to a complete RTOS solution including protocols stacks, multi-core communications, device drivers and power management.

This operating system, TI-RTOS, is the one used in all the demos provided by Texas Instruments and it would appear in the solution code for this project.

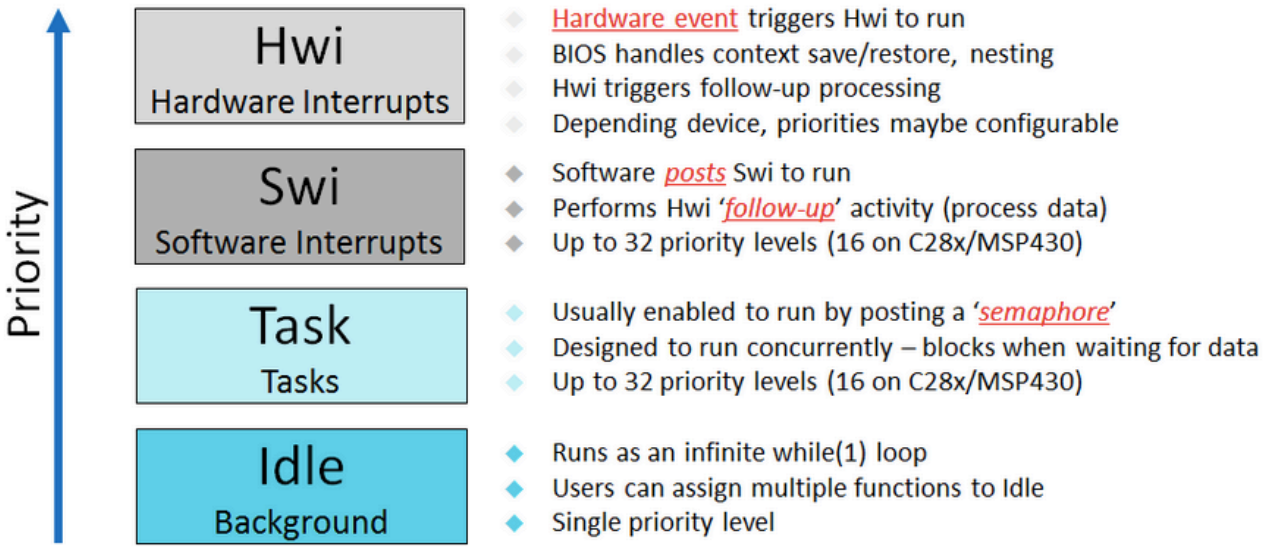


Figure 6: Kernel's Scheduler

According to professor Louis Zhu from the UCI, TI-RTOS Kernel has the following characteristics:

- Pre-emptive scheduler: The highest priority thread always runs first which means that time-slicing is not inherently.
- Event-driven: Any Kernel-configured interrupts or user calls to APIs will invoke the scheduler. Even it is not time-sliced, it can be triggered on a time bases if so desired.
- Object based: All the methods operate on self-contained objects, but it is important to highlight that when you change one object, the rest of the objects are unaffected.
- Deterministic: This characteristic comes from the one above. The scheduler works by updating queues such that all context switches take the same number of cycles.
- Intent to leave real-time analysis APIs in the program: These APIs are small and fast.

Kernel Components	Description
HWI	Hardware Interrupt management
SWI	Software Interrupt management
Task	Independent, pre-emptible thread, own stack, yield the processor
Clock	Time-triggered periodic functions
Event	Wait on multiple events (semaphore, mailbox, I/O, user-defined, ...)
Mailbox	Mailboxes for synchronized fixed-sized data exchange between tasks
Semaphore	Continuing semaphore
Gate	Protects against concurrent access to critical data structures
Heaps	Variable-sized allocation based on multiple fixed-sized buffer pools
Timer	Interface to HW timers, supporting one-shot or continuous callbacks
Diagnostics	Logs, timestamps, enable/disable diagnostics

Table 4: Kernel's key components

Chapter 3: Out of Box Demo

3.1 DESCRIPTION

The first phase of my project is getting to know the hardware, CC3220SF-LAUNCHXL, and the way to do it is to troubleshoot the demo that Texas Instruments provide for it. This Out of box demo is formed of two different parts (blinking a LED and try the accelerometer) with which the user can test the board and work with its basis. This demo does not require any background or experience and will not show the Wi-Fi capabilities that the hardware has, it is just a way to work for the first time with the board and to get an idea of its performance.

3.2 SET UP AND POWER THE BOARD

The first step to correctly run this demo is to set up the board as an access point. The CC3220SF-LAUNCHXL can work in different modes, one of these modes is access point.

I have already talked about the AP mode in the Hardware section. In the out of box demo, the board works as an access point to allow the user to connect to its network and monitor the rest of the demonstrations. When the board wakes up in this mode, it will have the default parameters showed in Table 2**.**

To set the CC3220SF-LAUNCHXL to the access point mode I need to ensure the jumper is placed to short pin 58 and VCC. This will force the device to the access point mode with the default parameters. Once pin 58 and VCC are shorted I power the board by connecting it to my PC using an USB cable. The board starts running and I press the switch number 3 (SW3) so it allows the board to wake up as an access point. To

summarize, I can say that shorting pin 58 and VCC sets the parameters to access point default parameters and then, pressing SW3 makes the board run as an access point.

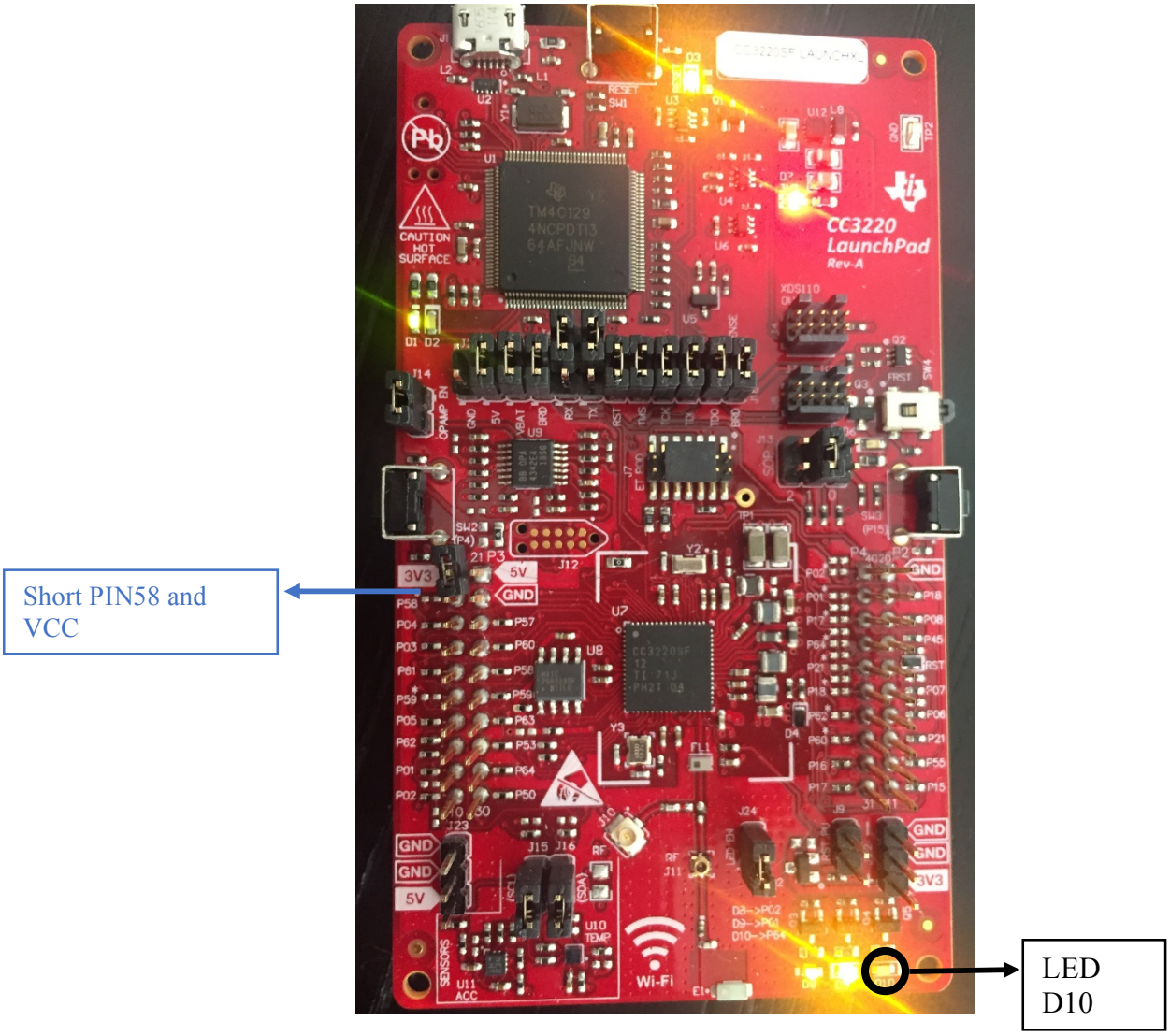


Figure 7: Board's schema

3.3 CONNECT MY LAPTOP TO CC3220

The board CC3220SF has woken up as an access point so the next step in this demo is to connect my laptop (MacBook Air) to the network that the board has created as an access point. I open the Wi-Fi settings on my laptop and select the network called

"mysimplelink-F60171", it does not require password because the default network that the board creates in access point mode is an open network.

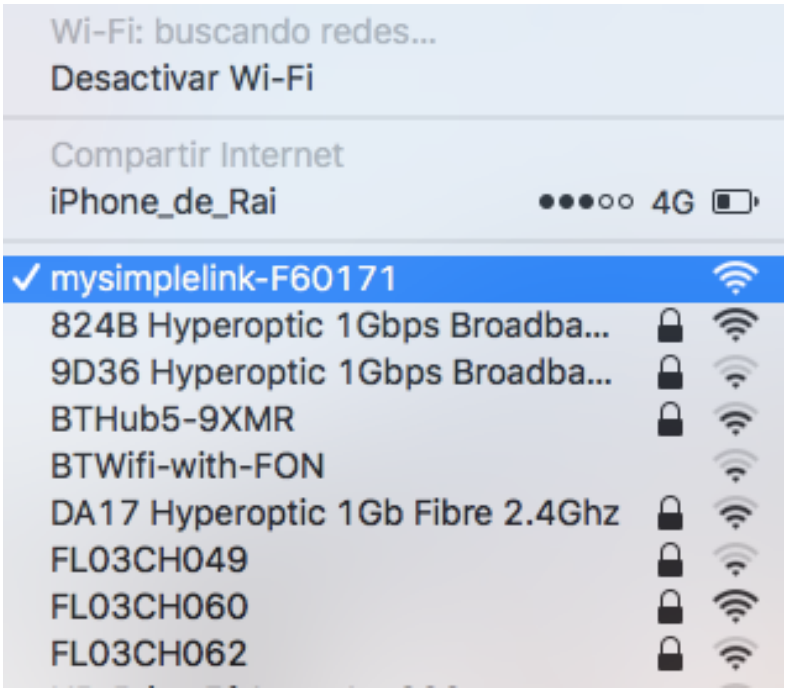


Figure 8: Board's network as an AP

3.4 BROWSE THE ON-BOARD WEBSITE

Once my smartphone is connected to the CC3220SF-LAUNCHXL the next step is to find the demos by browsing a specific website provided by Texas Instrument. I open my smartphone's web browser and type http://MySimpleLink.net in the web address line. The CC3220SF will stream the on-board web pages to my web browser, it takes a few minutes to my laptop to access the website. In this website, I can find the two demos that I can work with to test that the board works correctly

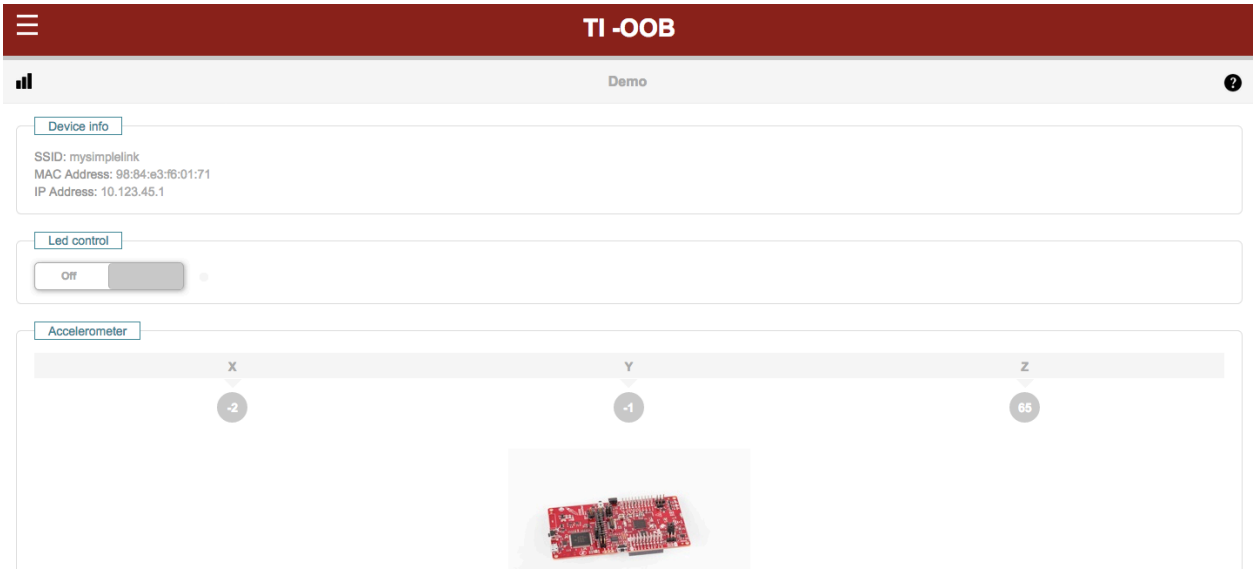


Figure 9: Interface of "My SimpleLink" webpage

3.5 RUN THE DEMONSTRATIONS

In the http://MySimpleLink.net website I can find two main demonstrations: blinking an on-board LED and the accelerometer.

The first demonstration (blinking a LED) consists of turning on and off the on-board LED D10 by clicking the switch that appears in the website. As it can be seen in Figure 10 and Figure 11, the LED D10 turns on when the online switch shows on and turns off when the online switch shows off.

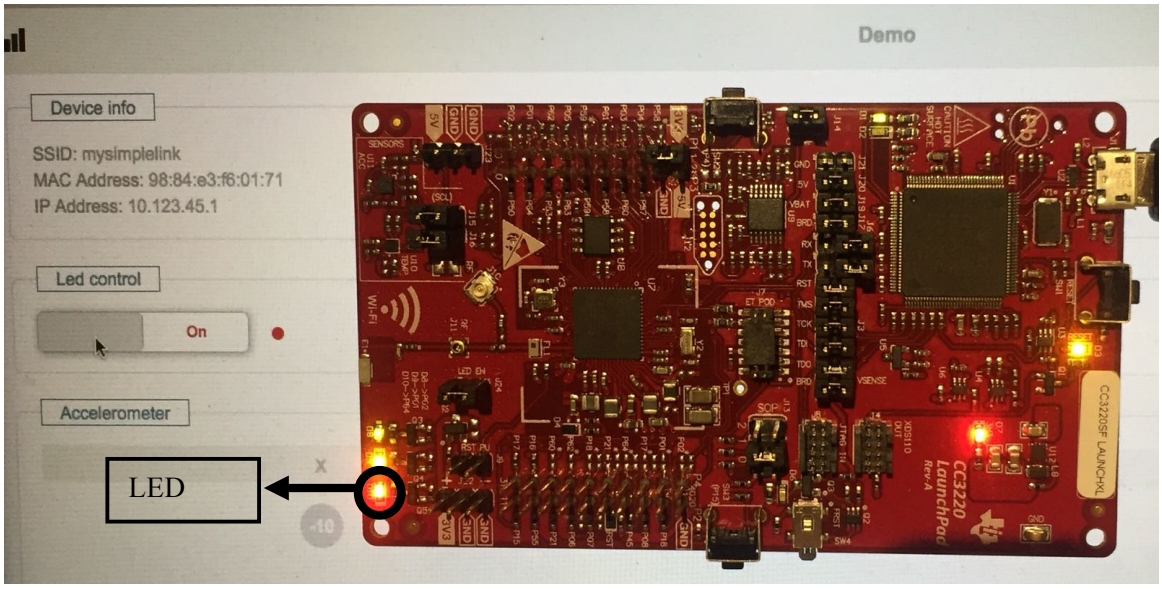


Figure 10: LED D10 turned on

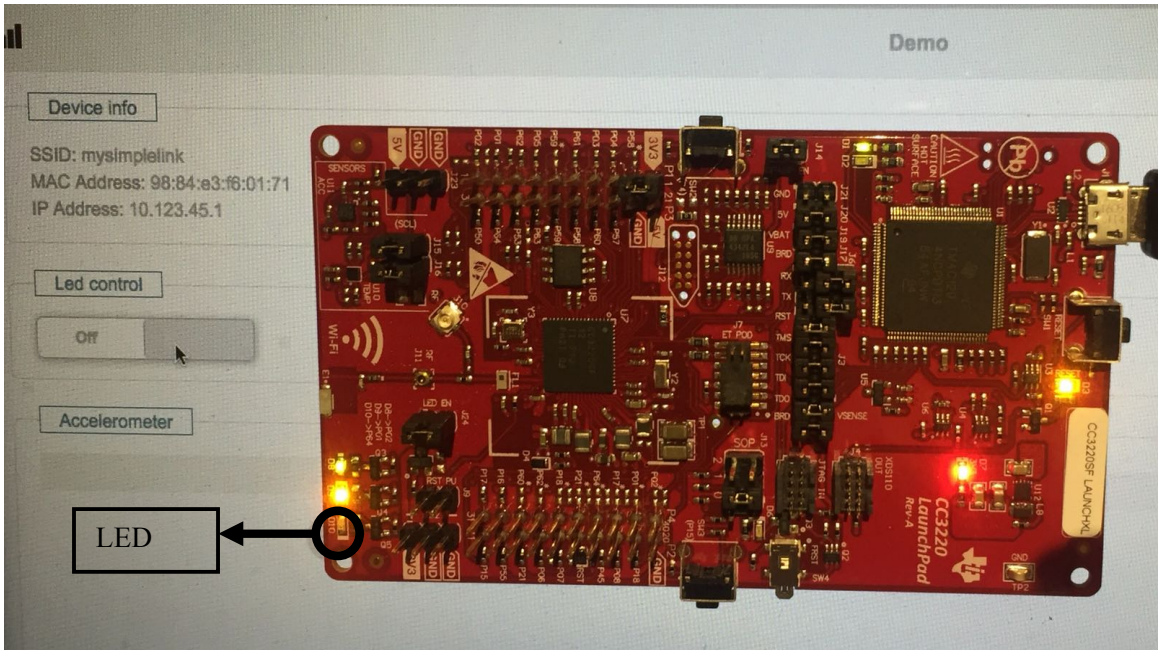


Figure 11: LED D10 turned off

The second demonstration that Texas Instruments provide to the new users is an accelerometer that identifies the position of the board by its coordinates and it shows them as ‘x’ axis, ‘y’ axis and ‘z’ axis values.

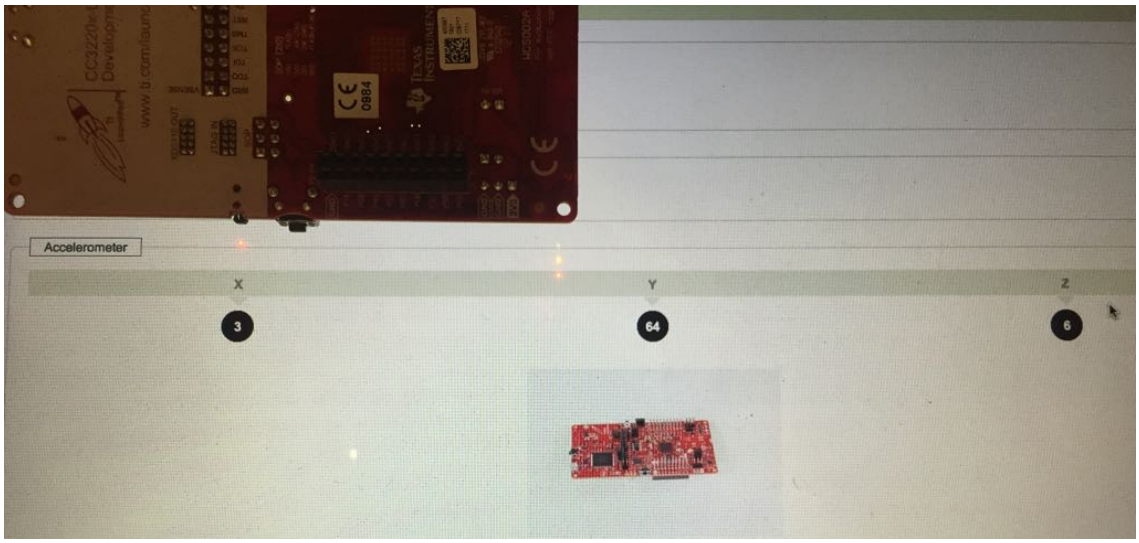


Figure 12: Accelerometer. First position

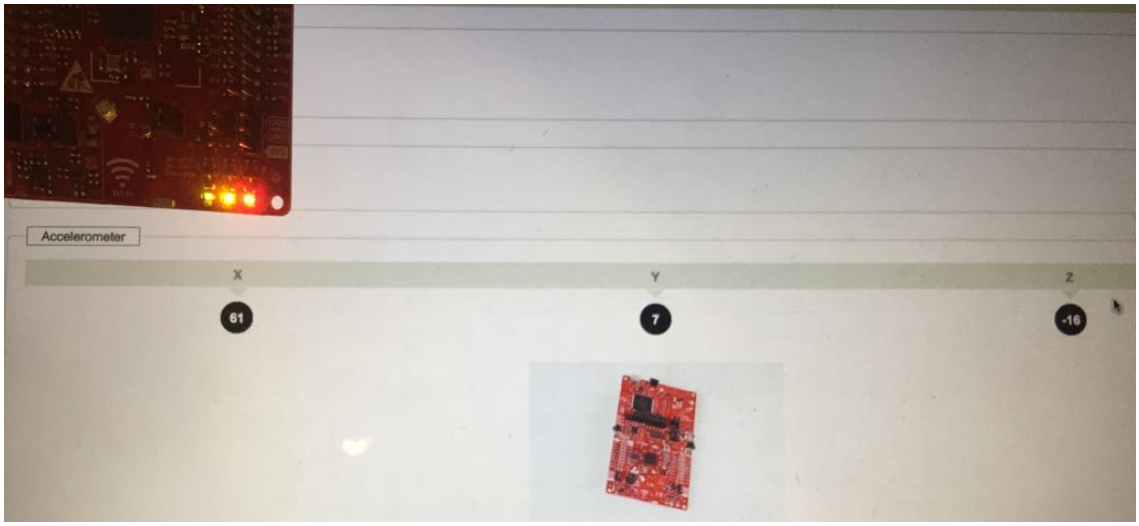


Figure 13: Accelerometer. Second position

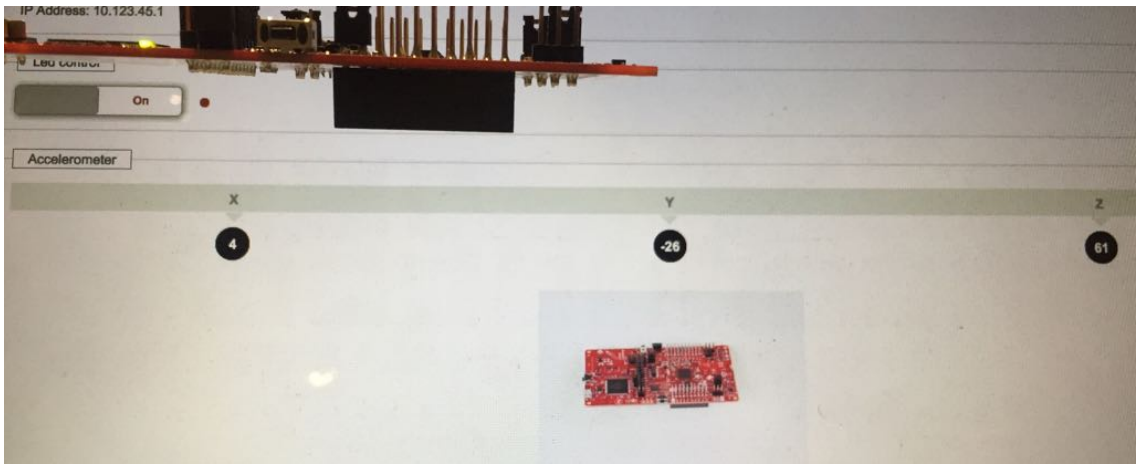


Figure 14: Accelerometer. Third position

Chapter 4: Wi-Fi Provisioning

4.1 OVERVIEW

One of the main objectives of this projects is to connect multiple Wi-Fi devices to a Wi-Fi network in which an enterprise security system is used. The importance of correctly fulfill this objective is huge because without a connection between the devices and the network there cannot be a data exchange and the rest of the objectives will not be achieved.

The SimpleLink Wi-Fi family supports a variety of different provisioning methods. In this project, I am only using a combination of two of them: Access Point + Smart Configuration.

- Access Point (AP): During this provisioning method, a Wi-Fi enabled device wakes up and temporally operates as an AP allowing a smartphone or laptop (in my case it would my smartphone, an IPhone 6) to connect to the board and transmit the network information for the desire network connection.
- Smart Configuration (SC): The proprietary of this method is Texas Instruments. In this method, the Wi-Fi enabled device is in station mode scanning for access points to connect to. At the same time, smartphone (this is my case) or tablet is used to broadcast the network credentials to the device. Another capability of the SC is that profiles can be added and saved through the SimpleLink mobile application.

The first idea I came out with was to use Wi-Fi provisioning. It is the process of connecting a Wi-Fi device (station) to a Wi-Fi network (access point) based on loading the station with the access point name (often referred to as SSID) and its security credentials. The security credentials are different depending on the type of security used by the network. In this project, I am trying to connect to a Wi-Fi network with enterprise

security, so I decided to first connect to a personal security network to see how Wi-Fi provisioning works and then try to connect to an enterprise security network (the campus Wi-Fi network of Cal State LA).

4.2 CERTIFICATES

These certificates are part of a trusted root-certificate catalog called certificate-playground which is provided by Texas Instruments. This file contains a group of known and most common trusted root CAs in the market (GoDaddy, VeriSign...), it also holds some revoked certificates known to TI. The trusted root-certificate catalog is only used in client mode as it ensures the client that the CA is trusted and known. There is a different between client and server, each server uses its own root CA to authenticate clients that is why the catalog provided by TI cannot be used by the servers.

4.3 CONNECTING TO PERSONAL SECURITY NETWORK

For the Wi-Fi provisioning, Texas Instruments provide an example code that I have modified to follow my own interests, the code is available in the TI Resource Explorer in the Demos section. I have programmed the code using CCS, when I built the program it generated a .bin file that can be used by Uniflash.

I used Uniflash ImageCreator to flash the code to the board. When loading the MCU image to the CC3220SF-LAUNCXL I used the .bin file generated by CCS and added, in addition to the "dummy" certificate chain, the dummy-root-ca-cert-key to connect to the internal HTTPS server. This last certificate that have been added as a user file is needed to create a secure connection for the SSL key exchange.

Once the code is flashed I used my smartphone (iPhone 6) to monitor the Wi-Fi provisioning through the SimpleLink Wi-Fi Starter Pro mobile application. This application allows the user, once your smartphone is connected to the board's network, to select the AP you want the board to connect to; in my case, I am connecting to iPhone_Brandom which is the name of the network created by my friend's smartphone.

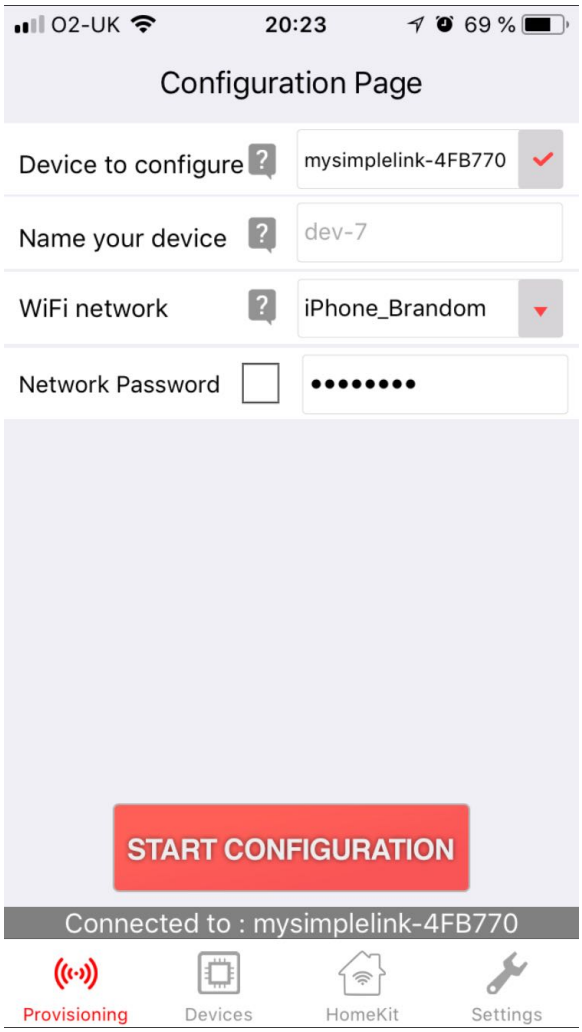


Figure 15: Configuration Page

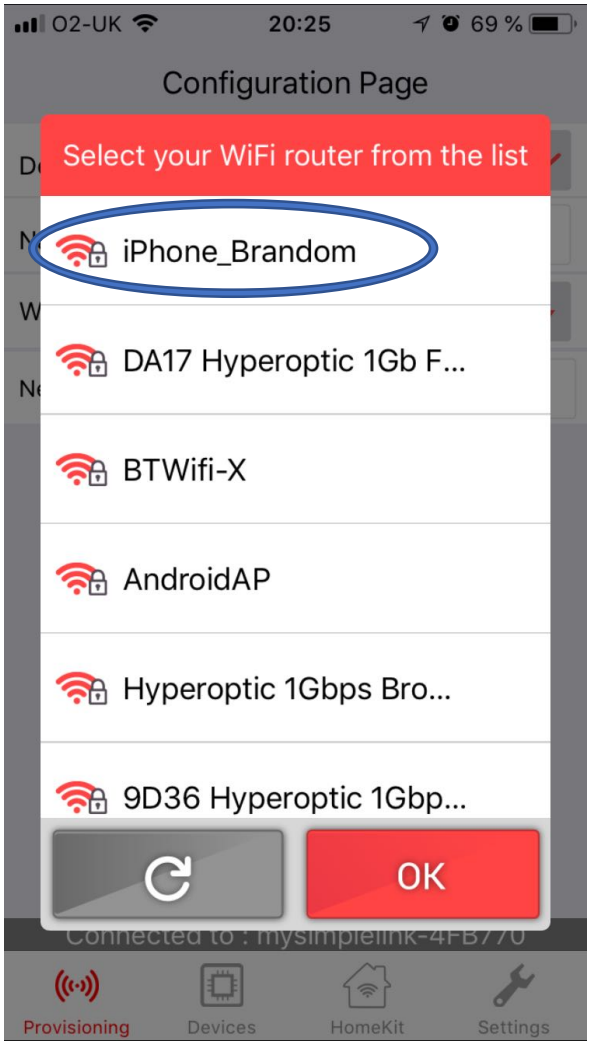


Figure 16: Access Point browser

In the code, you set the security of the AP by changing these parameters:

```
943 int32_t SetSecuredAP(const uint8_t sec_en)
944 {
945     //set secured AP parameters
946      uint8_t  val = AP_SEC_TYPE;
947      uint8_t  open = SL_WLAN_SEC_TYPE_WPA_WPA2;
948      uint8_t  password[65];
949      uint16_t  len = strlen((char *)AP_SEC_PASSWORD);
```

Figure 17: AP set up parameters

AP_SEC_TYPE is the security of the access point. It can be Open, Personal security or Enterprise security. For this first Wi-Fi provisioning part (connecting to a personal security network) I used the smartphone of one of my friends as a hotspot and its password, which is the only credential needed for personal security networks, is the one shown next to AP_SEC_PASSWORD. Also, you need to set SC_KEY which is the password that will be stored and used for Smart Configuration (once you have connected to an access point, the program will add a profile with that access point and will use the SC_KEY to automatically connect to it if Smart Configuration is enable).

```
90
91 #define AP_SEC_TYPE          (SL_WLAN_SEC_TYPE_WPA_WPA2)
92 #define AP_SEC_PASSWORD      "VazEsRod"
93 #define SC_KEY               "VazEsRod"
94
```

Figure 18: Provisioning parameters

As It can be seen in the Figure 19**,** the personal security network provisioning works correctly.

```
Starting Provisioning! mode=2 (0-AP, 1-SC, 2-AP+SC, 3-AP+SC+WAC)

[Provisioning] Profile Added: SSID: iPhone_Brandom
[Provisioning] Profile Added: SSID: iPhone_Brandom
[Provisioning] Profile confirmation: WLAN Connected!
[Provisioning] Profile confirmation: IP Acquired!
```

Figure 19: Wi-Fi provisioning terminal output

4.4 CONNECTING TO ENTERPRISE SECURITY NETWORK

The second Wi-Fi provisioning part is trying to connect to the campus network of Cal State LA which has an enterprise security system. I took the code from the first part changing the parameters needed: AP_SEC_TYPE is now SL_WLAN_SEC_TYPE_WAP_ENT which is the value for security enterprise networks. I followed the same steps as in the first part: build the code with CCS, flash it with Uniflash ImageCreator and monitoring the process through the SimpleLink application, but it did not work, the board was not connecting to the campus network.

It is important to differentiate personal security (mostly used in not such big places like homes or little businesses) and enterprise security (mostly used in big areas such as campuses or large offices) in terms of Wi-Fi security standard. The credentials needed by the last type involve installing some certificates that are used to verify the integrity of the station and the network by the interaction between client and server through the different security layers.

The certificates required for an enterprise security connection are three, they are used to authenticate the radius server and client according to their authentication settings:

- Client authentication: For this part two files are required:
    - Private Key: private key file in PEM format with the name sys/cert/private.key
    - Client Certificate: the authenticating network provides this certificate of the client. It is important that the public key matches to the private key. As the file above, it must be in PEM format. It must have the name sys/cert/client.der

- Server authentication: It is required by default, but it can be disabled. Only one file is required.
    - Server Root CA: in PEM format with the name sys/cert/ca.der

These certificates I am talking about need to be provided by the IT Department of the place which network you are trying to access. In my case and for the good of this project I contacted the IT Department of Cal State LA asking for these certificates. However, they told me that no certificates were needed and to use, instead, the EAP TTLS MSCHAPv2 as authentication method that requires my Cal State account credentials.

4.5 MESSAGE

One of the main goals of this project is to determine the communication quality of the two CC32200SF-LAUNCHXL that are connected to the enterprise security network, the way to test this communication is to set one board (let's say it is Board 1) as a transmitter and the other board (let's say it is Board 2) as a receiver. Board 1 will send a message to Board 2, if the message has the correct ID value (240 for Board 2) board 2 will have to display in the CCS terminal the "Data" section of the message; otherwise, board 2 will display the sentence "No message". At the same time, Board 2 will also send a message to Board 1 and if it has the correct ID (241 for Board 1) it will display in the CCS terminal the "Data" section of the message sent by Board 2; otherwise, it will display "No message".

The message format that the boards have to receive and send includes the following components:

1. DSRCMssgID: it works as an identifier for the Data in the message. For the Board 2, if it receives the correct ID value (240) it will print the “Data” section in the CCS terminal. For the Board 1, if it receives the ID value of 241 it will print the “Data” section in the CCS terminal. Range of DRSCMssgID = 0-32767.

2. DYear: it is an integer value representing the year according to Gregorian calendar date system, the value zero should represent an unknown value. Range of DYear: 0-4095.
3. MsgCount: it is used to provide a sequence number within a stream of messages with the same DSRCMssgId and from the same sender. A sender may initialize this element to any value in the range 0-127 when sending the first message with a given DSRCMssgID, or if the sender has changed identity since sending the most recent message with that DSRCMssgID. Depending on the application the sequence number may change with every message or may remain fixed during a stream of messages when the content within each message has not changed from the prior message sent. For this element, the value after 127 is zero. The receipt of a non-sequential MsgCount value (from the same sending device and message type) implies that one or more messages from that sending device may have been lost, unless MsgCount has been re-initialized due to an identity change. Range of MsgCount = 0-127.
4. RegionID: it is used to define regions where unique additional content may be added and used in the message set. The index values defined below represent various regions known at the time of publication. This list is expected to grow over time. The index values assigned here can be augmented by local (uncoordinated) assignments in the allowed range. Range of RegionID = 0-255.
5. Data1: first byte of the data section. Range = 0-127.
6. Data2: second byte of the data section. Range = 0-127.
7. Data3: third byte of the data section. Range = 0-127.
8. Data4: forth byte of the data section. Range = 0-127.
9. Data5: fifth byte of the data section. Range = 0-127.
10. Data6: sixth byte of the data section. Range = 0-127.
11. Data7: seventh byte of the data section. Range = 0-127.
12. Data8: eighth byte if the data section. Range = 0-127.

To summarize:

Description	Range	Units
DSRCMssgID	0-32767	N/A
DYear	0-4095	year
MsgCount	0-127	N/A
RegionID	0-255	N/A
Data1	0-127	N/A
Data2	0-127	N/A
Data3	0-127	N/A
Data4	0-127	N/A
Data5	0-127	N/A
Data6	0-127	N/A
Data7	0-127	N/A
Data8	0-127	N/A

Table 5: Message's structure

Chapter 5: MQTT

5.1 OVERVIEW

MQTT (Message Queuing Telemetry Transport) protocol is a light-weight machine-to-machine connectivity protocol based on a publish/subscribe messaging model and it is designed to be on top of the TCP/IP protocol. The key benefits or features of this protocol are: small code footprint and a low bandwidth requirement, faster response time, low power requirement, and ease of scalability. All these advantages make the MQTT an ideal candidate for a communication protocol in embedded devices intended to implement IoT (Internet of Things) applications.

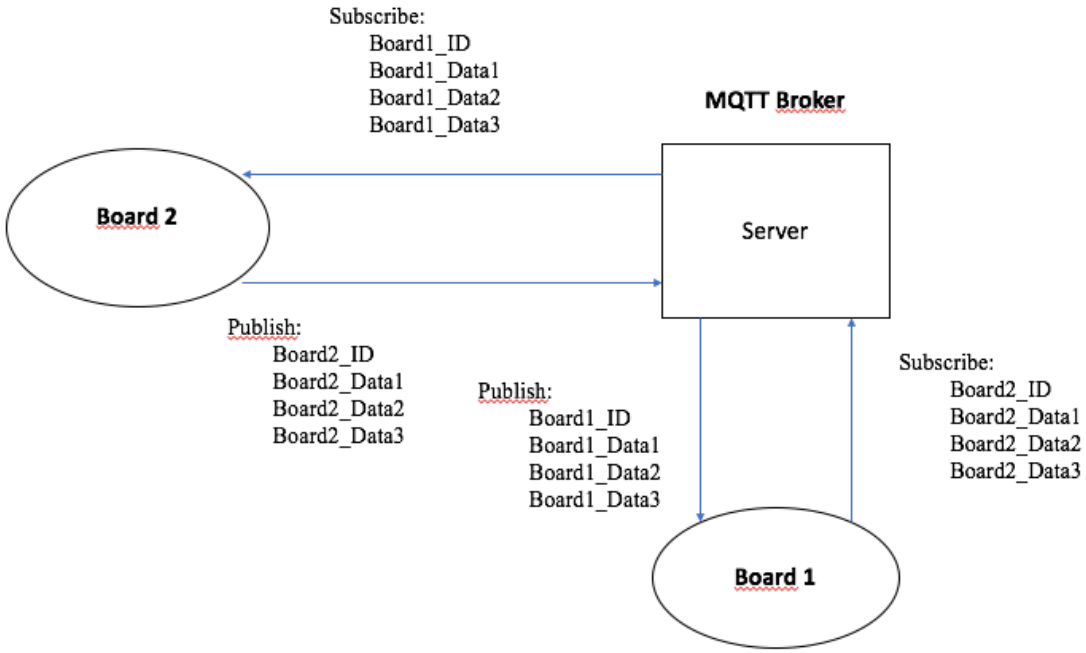


Figure 20: MQTT schema

The MQTT protocol contains a broker/server (like a central hub) connected to multiple clients, each of them has the capability of publishing on any topic to which it is subscribed (token). The broker has the responsibility of sending the message published on any topic to all the clients that are subscribed to the topic.

Texas Instruments provides a code for MQTT demonstration, I will modify the example code according to my needs to fulfill the goal of this project. TI has also developed an MQTT library that provides the user application with an intuitive and easy to use API to implement the MQTT protocol on the CC3220SF-LAUNCHXL. Different modules and APIs are used for the client and server mode. For the Client mode, the API includes: create or delete a client instance, connect to an MQTT server based on URL or IP Address, subscribe or unsubscribe to a topic, publish a topic and a message, run the client main task loop.

5.1 CONNECTION TO ENTERPRISE SECURITY NETWORK

As I said in section 4.4 I contacted the IT department of Cal State University and they told me that no certificates were needed if I used a specific authentication method. The CC3220SF board can support WPA2 enterprise (802.1x) with EAP-TTLS with TLS or EAP-PEAP0 with TLS as authentication methods (both work for this project) because both do not require a client-side certificate to be installed into your module, instead they need the user to provide his campus account credentials (in my case, I had to provide my Cal State University account credentials). The reason why connecting to an enterprise security network needs an authentication method is because the enterprise connection requires an authentication of the STA by the radius server behind the AP.

Protected EAP (PEAP) adds a TLS (Transport Layer Security) layer on top of the EAP in the same way as EAP-TLS, but it then uses the resulting TLS session as a carrier to protect other, legacy EAP methods. Commonly, EAP-PEAP uses TLS only to authenticate the server to the client but not the client to the server, only the server is

needed to have a public key certificate not the client. EAP-PEAP provides the following services to the EAP methods it protects: message authentication, message encryption, authentication the server to client, key exchange, fragmentation and reassembly and fast reconnect.

Talking about the Tunneled TLS EAP method (EAP-TTLS) we can find some similarities with the EAP-PEAP in the way it works and its features. On the EAP-TTLS, after the server is securely authenticate to the client via its CA certificate and optionally the client to the server, the server can then use the stablished secure connection ("tunnel") to authenticate the client. However, the main difference between EAP-PEAP and EAP-TTLS is that the first one is a SSL wrapper around EAP carrying EAP and the second one is a SSL wrapper around diameter TLVs (Type Length Values) carrying RADIUS authentication attributes.

In my project, I had to modify and set up the part of the code in charge of the network connection. These parameters are placed in the network_if.c and network_if.h files as shown in the Figure 20:

```
61 /* AP SSID                                                    */
62 #define SSID_NAME              "CSULA-SECURE"
63 /* Security type (OPEN or WEP or WPA)                         */
64 #define SECURITY_TYPE          SL_WLAN_SEC_TYPE_WPA_ENT
65 /* Password of the secured AP                                 */
66 #define SECURITY_KEY           "Negro123%"
67 /* Type of enterprise security                                */
68 #define SECURITY_ENT           SL_WLAN_ENT_EAP_METHOD_TTLS_MSCHAPv2
69 /* Username of the security enterprise                        */
70 #define USERNAME               "ralonso3"
71 /* Anonymous of the security enterprise                        */
72 #define ANONYMOUS               NULL
73
```

Figure 21: Set up security parameters

SSID_NAME stands for the name of the access point I am connecting to (in this case I am connecting the secure access point of Cal State campus). SECURITY_TYPE is the security of the access point (in this case is enterprise security, WPA_ENT). SECURITY_ENT stands for the authentication method I am using to connect to the network (in this case EAP TTLS_MSCHAPv2 so I do not need to use certificates and I

can connect given my Cal State LA account credentials, it could also be EAP PEAP_MSCHAPv2 by changing the parameter to SL_WLAN_ENT_EAP _METHOD_PEAP0_MSCHAPv2). USERNAME and SECURITY_KEY are the username and password of my Cal State LA account. About the ANONYMOUS parameter, I can skip it because a username has been sent to the server through the authentication method.

However, the boards were not connecting to the enterprise security network, an error value kept coming out which meant that some certificates were missing. I contacted the IT department of Cal State LA and they helped me to solve the problem. It looks like somehow a certificate to authenticate the server was needed after all because the SimpleLink Wi-Fi requires server authentication by default, so instead of installing the certificate, that the IT department could not provide me, I disabled the server authentication manually.

```
537    signed int Stat;
538
539    Stat = sl_WlanSet (SL_WLAN_CFG_GENERAL_PARAM_ID, SL_WLAN_GENERAL_PARAM_DISABLE_ENT_SERVER_AUTH, 1 ,&param);
540    if (Stat){
541        UART_PRINT("\nFail Status: Status = %d \n\r", Stat );
542    }
```

Figure 22: Disable server authentication

The downside of this way is that the server is not going to be authenticated which means that if the device scans an access point with the name "CSULA-SECURE" it will automatically connect to it and send my Cal State LA account credentials without authenticating if that access point is a real Cal State access point, so that access point even if it is a fake one will receive my account credentials.

5.2 TOPICS

MQTT is a protocol based on publish/subscribe messaging model which means that if a device A wants to send a message to device B, device A has to publish the message to one specific topic (let's say it is called "example_topic") and device B needs to be subscribed to the same topic device A has published to ("example_topic"), so every time device A publishes something, device B will automatically receive it. We could say that the topics are "created" in the server and stay there.

In this project, Board1 and Board2 work as receivers and transmitters so both of them have to publish to some topics and subscribe to other topics in order to send and receive the messages. Board1 will subscribe to the topics where Board2 is publishing to, so it can get the message Board2 is sending. And Board2 will subscribe to the topics Board1 is publishing to so it can get the message Board1 is sending.

```
122 /* Defining Number of subscription topics
123 #define SUBSCRIPTION_TOPIC_COUNT 4
124
125 /* Defining Subscription Topic Values. Board1
126 #define SUBSCRIPTION_TOPIC0      "Board1_ID"
127 #define SUBSCRIPTION_TOPIC1      "Board1_Data1"
128 #define SUBSCRIPTION_TOPIC2      "Board1_Data2"
129 #define SUBSCRIPTION_TOPIC3      "Board1_Data3"
```

Figure 23: Subscribed topics Board 2

```
141 // Defining Subscription Topic Values. Board2
142 #define SUBSCRIPTION_TOPIC0      "Board2_ID"
143 #define SUBSCRIPTION_TOPIC1      "Board2_Data1"
144 #define SUBSCRIPTION_TOPIC2      "Board2_Data2"
145 #define SUBSCRIPTION_TOPIC3      "Board2_Data3"
```

Figure 24: Subscribed topics Board 1

SUBSCRIPTION_TOPIC_COUNT stands for the number of topics I want to subscribe to (for this project, subscribing to 4 topics is enough). There is a default maximum limit of topics you can subscribe to; this limit is 12 but can be modified to as many topics as the user needs. The fact that you can subscribe to as many topics as you want has a lot of interesting applications I will talk about in Chapter 8**.** On the other hand, there is no limit to the number of topics you are publishing to and no parameter needs to be set.

Board1_ID is the topic where I am sending the value of the ID; Board1_Data1 and Board2_Data1 are where I am sending Data1, Data2, Data3 and Data4; Board1_Data2 and Board2_Data2 are where I am sending Data5, Data6, Data7 and Data8; Board1_Data3 and Board2_Data3 are where I am sending the rest of the message. It has to be noticed that the ID value has to be sent by its own, but the rest of the message could be send together, this is due to the fact that I am sending strings so if I send all the message together I will not be able to differentiate the ID from the rest of the message to, lately, check its value.

Topic	**Board subscribing**	**Board publishing**	**Content**
Board1_ID	Board1	Board2	ID of the message sent by Board2
Board1_Data1	Board1	Board2	Data1, Data2, Data3 and Data4 of

			the message sent by Board2
Board1_Data2	Board1	Board2	Data5, Data6, Data7 and Data8 of the message sent by Board2
Board1_Data3	Board1	Board2	Year, Region ID and Message Counter of the message sent by Board2
Board2_ID	Board2	Board1	ID of the message sent by Board1
Board2_Data1	Board2	Board1	Data1, Data2, Data3 and Data4 of the message sent by Board1
Board2_Data2	Board2	Board1	Data5, Data6, Data7 and Data8 of the message sent by Board1
Board2_Data3	Board2	Board1	Year, Region ID and Message Counter of the message sent by Board1

Table 6: Topic's organization

Chapter 6: Message and Server

6.1 MESSAGE

The message that is been sent for each of the board has already been described in section 4.4. I have defined a structure called "Proj_Message" with all the content of the message (ID, Data1, RegionID, etc.) defined as variables type int.

```
typedef struct {
   int DSRCMssgID;
   int Dyear;
   int MsgCount;
   int RegionID;
   int Data1;
   int Data2;
   int Data3;
   int Data4;
   int Data5;
   int Data6;
   int Data7;
   int Data8;
}Proj_Message;

Proj_Message Pro_mssg;
```

```
   Pro_mssg.DSRCMssgID = 241;
   Pro_mssg.Dyear = 2018;
   Pro_mssg.MsgCount = 127;
   Pro_mssg.RegionID = 1;
   Pro_mssg.Data1 = 8;
   Pro_mssg.Data2 = 7;
   Pro_mssg.Data3 = 6;
   Pro_mssg.Data4 = 5;
   Pro_mssg.Data5 = 4;
   Pro_mssg.Data6 = 3;
   Pro_mssg.Data7 = 2;
   Pro_mssg.Data8 = 1;
```

Figure 25: Published message's structure

However, at this point I had to face a problem due to the fact that the publish function (MQTTCLient_publish) only takes strings as it is most used by many brokers. I had to convert the different variables of the message format which were type int into strings which difficult the objective of the project as string format uses more bits and occupies more bandwidth. The values of the message are introduced as type int and are converted to type string using the function sprintf.

```
sprintf (valor_Data1, "Data1 = %d; Data2 = %d; Data3 = %d; Data4 = %d", Pro_mssg.Data1, Pro_mssg.Data2, Pro_mssg.D
sprintf (valor_Data2, "Data5 = %d; Data6 = %d; Data7 = %d; Data8 = %d", Pro_mssg.Data5, Pro_mssg.Data6, Pro_mssg.D
sprintf (valor_Data3, "Year = %d; MssgCount = %d; Region = %d", Pro_mssg.Dyear, Pro_mssg.MsgCount, Pro_mssg.Regior
sprintf (valor_ID, "%d",Pro_mssg.DSRCMssgID);
```

Figure 26: Message variable's conversion

6.2 SERVER

In first place when this project was defined, we thought about using the server of Cal State LA campus, with its IP Address and some certificates the boards could have communicated to it and used the server for the MQTT messaging protocol. However, when I contacted the IT department about using their servers and what certificates I needed to access it, they told me I was not authorized to use their servers.

Texas Instruments provides a server with the MQTT demo, it looks like each board has their own internal server. I will be using these internal servers that the boards have; this fact does not modify the project because the server is just a path, a tool only used to "carry" the topics. The parameters of the server I am using appear in Figure 26, if I wanted to connect to other server I would have to change these parameters.

```
/* Defining Broker IP address and port Number
//#define SERVER_ADDRESS          "messagesight.demos.ibm.com"
#define SERVER_ADDRESS          "m2m.eclipse.org"
#define SERVER_IP_ADDRESS       "192.168.178.67"
#define PORT_NUMBER             1883
#define SECURED_PORT_NUMBER     8883
#define LOOPBACK_PORT           1882
```

Figure 27: Server's parameters

6.3 PUBLISH MESSAGE

To send a message from a board, to publish message to a topic, it is necessary to press the switch number 2 of the board (SW2), that is how the board "sends" its messages.

```
711         /* send publish message                 *                                   */
712         lRetVal = MQTTClient_publish(gMqttClient, (char*) publish_topic_ID, strlen((char*)publish_topic_ID)
713         MQTTClient_publish(gMqttClient, (char*) publish_topic_Data1, strlen((char*)publish_topic_Data1), (c
714         MQTTClient_publish(gMqttClient, (char*) publish_topic_Data2, strlen((char*)publish_topic_Data2), (c
715         MQTTClient_publish(gMqttClient, (char*) publish_topic_Data3, strlen((char*)publish_topic_Data3), (c
```

Figure 28: Publish message function. First part

MQTTClient_publish is the function called to publish any messages. You need to enter the right parameters when calling the function. In first place, gMqttClient is the initialized MQTT client library that the user is using. The next "components" when calling the function are publish_topic_ID, publish_topic_Data1, publish_topic_Data2, publish_topic_Data3 and publish_topic_Data4 which are the topics where the message it is going to be published to. The next "component" is the length of each of the last components, this length will be used when a message is received, to set the right buffer to store the values.

```
ID), (char*)publish_ID, strlen((char*) publish_ID), MQTT_QOS_2 | ((RETAIN_ENABLE)?MQTT_PUBLISH_RETAIN:0) );
 (char*)publish_data1, strlen((char*) publish_data1), MQTT_QOS_2 | ((RETAIN_ENABLE)?MQTT_PUBLISH_RETAIN:0) );
 (char*)publish_data2, strlen((char*) publish_data2), MQTT_QOS_2 | ((RETAIN_ENABLE)?MQTT_PUBLISH_RETAIN:0) );
 (char*)publish_data3, strlen((char*) publish_data3), MQTT_QOS_2 | ((RETAIN_ENABLE)?MQTT_PUBLISH_RETAIN:0) );
```

Figure 29: Publish message function. Second part

After the topic comes the message I am publishing itself which is: publish_ID, publish_Data1, publish_Data2, publish_Data3 and publish_Data4 which are the string form of the components type int of the message. The next "component" is the length of each of the last components, this length will be used when a message is received, to set the right buffer to store the values.

6.4 RECEIVING MESSAGE

Once the publishing function is called, the message is published to the topic and all the devices that are subscribed to that topic receive the message published. The publishing part was just calling a function and enter the right parameters, but the receiving part is much more complicated. A buffer will be used to store the data and the topic where it has been published. It has to be noticed that in this project I am not just publishing one

```
        struct publishMsgHeader msgHead;

        char *pubBuff = NULL;
        //Proj_Message *pubBuffData = NULL;

        struct msgQueue queueElem;

        topicOffset = sizeof(struct publishMsgHeader);
        payloadOffset1 = sizeof(struct publishMsgHeader) + recvMetaData->topLen + 1;


        bufSizeReqd += sizeof(struct publishMsgHeader);
        bufSizeReqd += recvMetaData->topLen + 1;
        bufSizeReqd += dataLen + 1;
        pubBuff = (char *) malloc(bufSizeReqd);


        if (pubBuff == NULL)
        {
            APP_PRINT("malloc failed: recv_cb\n\r");
            return;
        }

        msgHead.topicLen = recvMetaData->topLen;
        msgHead.payLen = dataLen;
        msgHead.retain = recvMetaData->retain;
        msgHead.dup = recvMetaData->dup;
        msgHead.qos = recvMetaData->qos;
        memcpy((void*) pubBuff, &msgHead, sizeof(struct publishMsgHeader));
```

Figure 30: Receiving function

message at a time, I am publishing to four different topics at every time one board wants to send a message to the other. As it can be seen in Figure 27 and Figure 28 the first part of the message to be published is the ID, the second part is the first four data variables,

the third part the second four data variables and the last part is the Year, the Region and the counter.

```
/* copying the topic name into the buffer                          */
memcpy((void*) (pubBuff + topicOffset), (const void*)recvMetaData->topic, recvMetaData->topLen);
memset((void*) (pubBuff + topicOffset + recvMetaData->topLen), '\0', 1);
 state = (pubBuff + topicOffset);

/* copying the payload into the buffer                *                    */
memcpy((void*) (pubBuff + payloadOffset1), (const void*) data, dataLen);
memset((void*) (pubBuff + payloadOffset1 + dataLen), '\0', 1);
```

Figure 31: Copying into the buffer

To be able to control what values the buffer is storing and to make sure I get the correct values, I have defined a variable called state which will be used to store the value of the topic every time a message is received, that value will be compared with constants variables (cmp0, cmp1, cmp2, cmp4) which contains the different topics where I am sending the message so I can know which part of the message I am receiving each time.

```
67 //Topics which the board is subscribed to. Board1
68 const char *cmp0 = "Board2_ID";
69 const char *cmp1 = "Board2_Data1";
70 const char *cmp2 = "Board2_Data2";
71 const char *cmp3 = "Board2_Data3";
72
```

Figure 32: Compare's variables. Board 1

```
75 //Topics which the board is subscribed to. Board2
76 const char *cmp0 = "Board1_ID";
77 const char *cmp1 = "Board1_Data1";
78 const char *cmp2 = "Board1_Data2";
79 const char *cmp3 = "Board1_Data3";
```

Figure 33: Compare's variables. Board 2

The message that will be stored is type string. I use the function strcmp to compare the variable state (store the topic of the message I have just received) and the four different topics I have, represented by the variable compare. Once I am in the correct topic, I use the function strcpy to store the message type string into a variable because when another message is received, the buffer will kill the last value and store a different one.

```
            if (strcmp (state,cmp0) == 0){
                strcpy (ID_value,(pubBuff + payloadOffset1));
                j =0;
              Mssg_recv.DSRCMssgID = atoi(ID_value);
             }else if (strcmp (state,cmp1) == 0){
                 strcpy (Data1_value,(pubBuff + payloadOffset1));
             }else if (strcmp (state,cmp2) == 0){
                 strcpy (Data2_value,(pubBuff + payloadOffset1));
                 j=1;
             }else if (strcmp (state,cmp3) == 0){
                 strcpy (Data3_value,(pubBuff + payloadOffset1));

             }
```

Figure 34: Storing the received value

The ID value will be compared to the valid ID (240 for Board1 and 241 for Board2) and if it is right, the data values will be displayed, otherwise it will show “No message”.

```
51              UART_PRINT("\n\rMsg Recvd. by client\n\r");
52
53              if (j ==1) {
54                  if (strcmp(ID_value, valid_ID)==0){
55                        UART_PRINT ("\n The ID is: %s", ID_value);
56                        UART_PRINT ("\n  %s", Data1_value);
57                        UART_PRINT ("\n  %s", Data2_value);
58
59                  }else {
60                      UART_PRINT ("\n No message");
61                  }
62              }
63
```

Figure 35: Display function

Chapter 7: Conclusion

The way to test and verify that there is a correct communication between boards is to display some values if the board gets a correct ID. The Board1 is publishing messages to the topics where Board2 is subscribed to and Board2 is publishing to the topics where Board1 is subscribed to, which means that any message published by Board1 will be automatically received by Board2 and the other way around. To determine that the communication quality is the correct one, Board1 will display the "Data part" of the message (Data1, Data2, Data3, Data4, Data5, Data6, Data7, Data8) if it receives the correct ID from the Board2 (the correct ID value for Board1 is 240); otherwise, it will display the sentence "No message".

```
Trying to connect to AP: Hyperoptic 1Gbps Broadband B5 ...

Succes Status = 0
 lRetVal: 0
            [WLAN EVENT] STA Connected to the AP: Hyperoptic 1Gbps Broadband B5 , BSSID: 0:2:61:8b:6f:b6
[NETAPP EVENT] IP acquired by the device

Device has connected to Hyperoptic 1Gbps Broadband B5
Device IP Address is 192.168.1.64

.CONNACK:
Connection Success
.Client subscribed on Board1_ID
,Client subscribed on Board1_Data1
,Client subscribed on Board1_Data2
,Client subscribed on Board1_Data3
,.

Msg Recvd. by client

Msg Recvd. by client

Msg Recvd. by client

 The ID is: 240
                Data1 = 1; Data2 = 2; Data3 = 3; Data4 = 4
                                                          Data5 = 5; Data6 = 6; Data7 = 7; Data8 = 8
```

Figure 36: Output Board 1

It works the same way with Board2, it will display the "Data part" of the message if it receives the correct ID from the Board1 (the correct ID value for Board2 is 241); otherwise, it will display the sentence "No message".

```
 lRetVal: 0
           [WLAN EVENT] STA Connected to the AP: Hyperoptic 1Gbps Broadband B5 , BSSID: 0:2:61:8b:6f:b6
[NETAPP EVENT] IP acquired by the device

Device has connected to Hyperoptic 1Gbps Broadband B5
Device IP Address is 192.168.1.36

.CONNACK:
Connection Success
Client subscribed on Board2_ID
,Client subscribed on Board2_Data1
,Client subscribed on Board2_Data2
,Client subscribed on Board2_Data3
,.

Msg Recvd. by client
Retained

Msg Recvd. by client
Retained

Msg Recvd. by client

 The ID is: 241
                Data1 = 8; Data2 = 7; Data3 = 6; Data4 = 5
                                                    Data5 = 4; Data6 = 3; Data7 = 2; Data8 = 1Retained
```

Figure 37: Output Board2

I can conclude the MQTT based on the publish/subscribe protocol is a correct and efficient way to communicate between boards by subscribing to a topic. It gives you the flexibility to select to which board you want to send the message (subscribe that board to the topic you are publishing the message), to send the same message to more than one board (you can subscribe more than one board to the same topic) and more applications that I will talk about in the next section.

Some other important points that I can conclude from this project are:

1. Two sets of data can be received at the same time. There is a message queue to monitor the messages that are sent and receive but every time a message is published the case "MQTTClient_RECV_CB_EVENT" will be called so the first message that will be published is the first one that will be received. In this project, I am sending and receiving four messages at the same time.
2. The size of the buffer is given by the size of the structure which is determined by the message received, so I will have enough size to get the whole message. I guess, we can say the buffer do not have a maximum size as it will always be big enough to store the message that it is received

3. With MQTT the boards can be defined as transmitter, receiver or both at the same time easily. The transmitter will be the one that publishes messages to a certain topic and the receiver will be the one that is subscribed to those topics. If you want to define a board as transmitter and receiver you just have to publish to some topics and subscribe to others at the same time which can be done.
4. Many boards can be subscribed to the same topic, there is not a specified maximum number of boards that can be subscribed to a topic. This fact is very useful if you want to send the same message to a lot of boards at the same time.
5. There is not a maximum number of topics that you can publish or be subscribed to. However, the code has a macro that limits the size of the topics the device can simultaneously subscribed to; this macro can be changed to whatever number you want so there is not really a maximum number of topics you can be subscribed to.
6. You can send data as big as the size of the string that holds it. The publish function only accepts type string so the data will not be limited by its content but may be limited by the size of the string.

Chapter 8: Other Applications

8.1 ANOTHER WAY

In this project, I have shown one way to fulfill the initial objective, to connect to an enterprise security network two CC3220SF-LAUNCHXL and then determine the communication quality between both devices. About connecting to the enterprise security network, I have not noticed an alternative way to do it; I still have the problem about authenticating the server, remember that I had to disable the server authentication because the IT department did not provide me the certificates needed, in order to avoid someone to steal my Cal State University LA account details. The only way to make the connection to a network with enterprise security was manually so I could specify the authentication method I was using and to enter the correct values for the parameters required.

Once I get to the part where I need to send messages between boards to test the quality of the communication, I had to face one important problem: the publishing function only accepts variables type string as the message it publishes. This fact was a problem because the message I had to send was "made" of variables type int, so not only I had to convert the variables from type int to type string but the communication was not going to be the optimum as variables type string use more bits and occupies more bandwidth.

Another downside was the maximum number of topics you can subscribe to. The code I have developed to satisfy the objective of this project is based on the MQTT demo that Texas Instruments provides; this example code has an internal file called mqttclient.c where a variable is used to limit the number the of topics you can subscribe to, the name of the variable is MQTTCLIENT_MAX_SIMULTANEOUS_SUB_TOPICS**.**

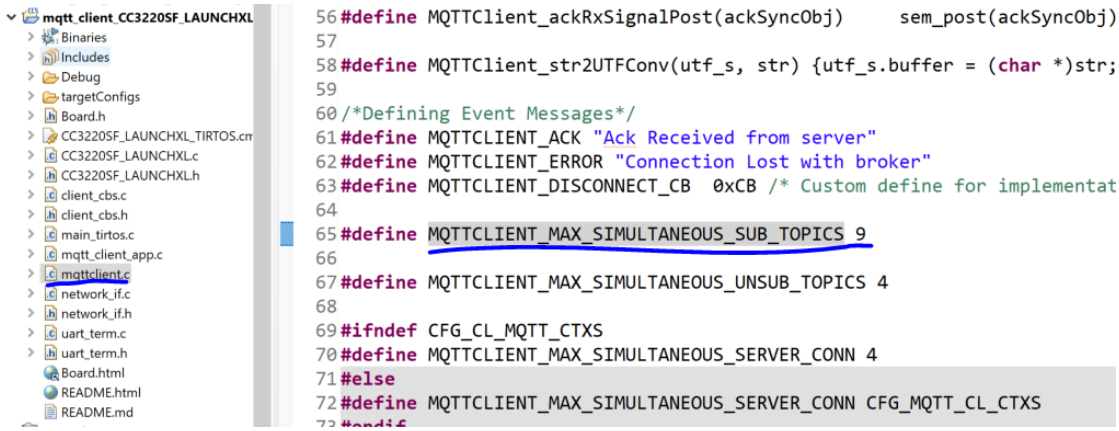


Figure 38: Limit topics variable

This maximum number can be changed, but you cannot just modify the file and save it again because it will not cause any changes. The way to change the maximum number is to install another file with the numbers of topics you want as maximum and delete the old ones. Knowing this, another alternative to this project comes out that could be more useful.

In my solution, the only part of the message that was sent separately was the ID, the rest of the message was sent in groups what made really hard to recover all the message variable by variable. The message that I was sending was not the value of the 8 data variables, the year, the region and the counter; it was a whole sentence what I was sending. For instance, instead of sending 1 (value of Data1), 34 (value of Data2) I was sending the following sentence: “Data1 = 1, Data2 = 34” which works for this project as the only thing I need to do with the message is to display the “Data” part if the ID is correct.

However, if I wanted to use the content of the message once it is received I would not be able to do it because all I would have is the sentence “Data1 =1, Data2 = 34”. To make my solution more efficient and useful I have though an alternative: modify the maximum number of topics from four to twelve (ID, 8 data variables, the year, the counter and the region) and each topic would represent one part of the message; now, when Board1 sends a message to Board2, each part of the message would be published in one of the twelve topics

New Topics	Content	Units
Board1_ID and Board2_ID	ID value of the message sent by Board2 and Board1	N/A
Board1_Data1 and Board2_Data1	Data1 of the message sent by Board2 and Board1	N/A
Board1_Data2 and Board2_Data2	Data2 of the message sent by Board2 and Board1	N/A
Board1_Data3 and Board2_Data3	Data3 of the message sent by Board2 and Board1	N/A
Board1_Data4 and Board2_Data4	Data4 of the message sent by Board2 and Board1	N/A
Board1_Data5 and Board2_Data5	Data5 of the message sent by Board2 and Board1	N/A
Board1_Data6 and Board2_Data6	Data6 of the message sent by Board2 and Board1	N/A
Board1_Data7 and Board2_Data7	Data7 of the message sent by Board2 and Board1	N/A
Board1_Data8 and Board2_Data8	Data8 of the message sent by Board2 and Board1	N/A
Board1_Year and Board2_Year	Year value of the message sent by Board2 and Board1	year
Board1_Region and Board2_Region	Region value of the message sent by Board2 and Board1	N/A
Board1_Counter and Board2_Counter	Counter value of the message sent by Board2 and Board1	N/A

Table 7: Topic's organization. Another alternative

In this new application, once the receiving message function gets the message published in the topic to which the board is subscribed it can be converted from string

type to an int type again. Now, the receiver stores the message as int type, the same type as it was sent in first place, and it can be used for any other application.

8.2 OTHER APPLICATIONS

The MQTT message protocol has opened my eyes to an extend number of different applications. In the las section I have been talking about a different point of view to approach the initial objective of this project, it was a matter of subscribing to different topics so each part of the message could go to a different topic.

However, one thing to take into account is another application for this project: you can send different messages to different boards by subscribing each of them to a different topic and you can also send one message to all of the by subscribing all the boards to one same topic. Let's put a practical case: Imagine you have 5 boards, you can subscribe board1 to topic1 "Board1", board2 to topic2 "Board2", board3 to topic3 "Board3", board4 to topic4 "Board4" and board5 to topic1 "Board5" and every time you want to send a message to any of the boards you just publish the message in the topic where the boards are subscribed; then you can subscribe board1, board2, board3, board4 and board5 to topic6 "All boards" if you want to send one message to all the boards. Another point is that you can monitor to which board you send a message depending on the ID value. The way to do this is setting a statement before calling the publishing function:

```
If (ID_vaue == desired_value) {

    MQTTClient_publish (…., desired_topic);

}
```

It is important to think about the topics you are sending messages to as if they were the board you are sending the message to so if you want to send a message to a specific board you only need to know the topic to which it is related.

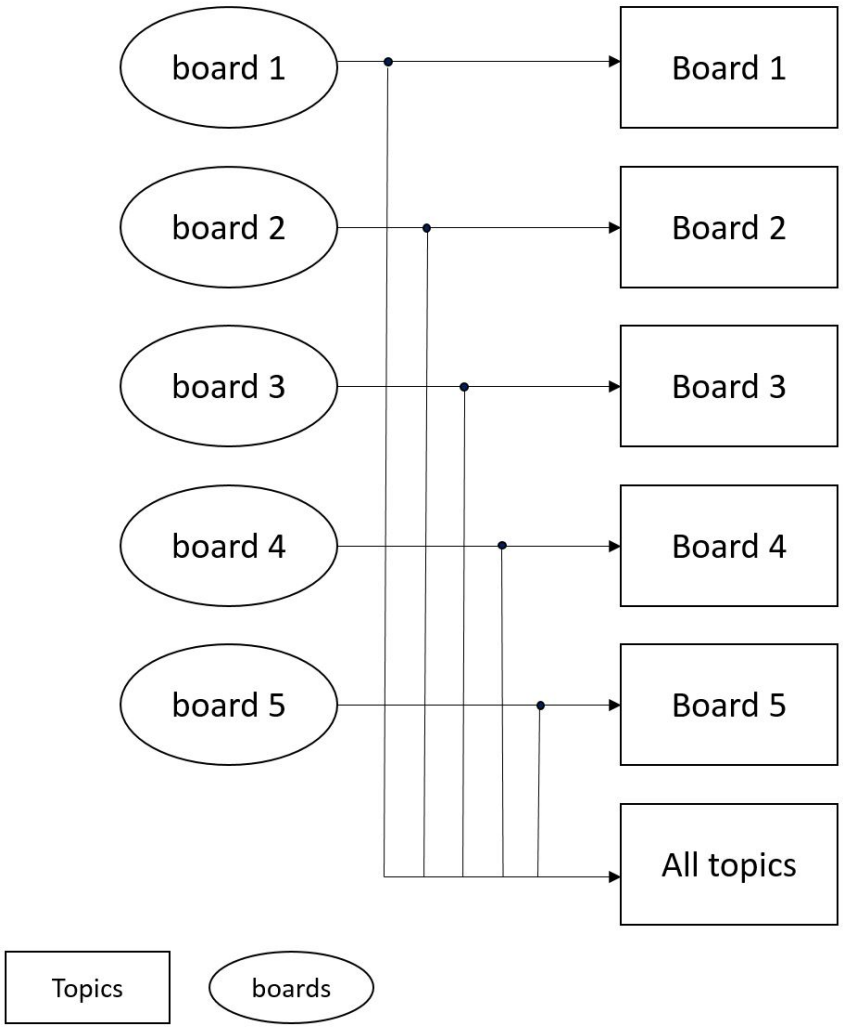


Figure 39: Other application's schema

References

1. Ti.com. (2018). *CC3220 SimpleLink ™ Wi-Fi ® Wireless and Internet-of-Things Solution, a Single-Chip Wireless MC*. [online] Available at: http://www.ti.com/lit/ds/symlink/cc3220.pdf [Accessed 17 Jun. 2018].

2. Ti.com. (2018). *CC3120, CC3220 SimpleLink ™ Wi-Fi ® and Internet of Things Network Processor Programmer 's Guide*. [online] Available at: http://www.ti.com/lit/ug/swru455e/swru455e.pdf [Accessed 17 Jun. 2018].

3. Dev.ti.com. (2018). *TI Resource Explorer*. [online] Available at: http://dev.ti.com/tirex/ [Accessed 1 May 2018].

4. Ti.com. (2018). *CCSTUDIO-WCS Code Composer Studio (CCS) Integrated Development Environment (IDE) for Wireless Connectivity | TI.com*. [online] Available at: http://www.ti.com/tool/ccstudio-wcs [Accessed 11 May 2018].

5. Ti.com. (2018). *SIMPLELINK-CC3220-SDK SimpleLink™ Wi-Fi® CC3220 Software Development Kit (SDK) | TI.com*. [online] Available at: http://www.ti.com/tool/simplelink-cc3220-sdk [Accessed 24 Jul. 2018].

6. Dev.ti.com. (2018). *General RTOS Concepts*. [online] Available at: http://dev.ti.com/tirex/content/simplelink_academy_cc32xxsdk_1_15_02_00/modules/rtos_concepts/rtos_concepts.html [Accessed 5 Jun. 2018].

7. Dev.ti.com. (2018). *TI-RTOS Basics*. [online] Available at: http://dev.ti.com/tirex/content/simplelink_academy_cc32xxsdk_1_15_02_00/modules/tirtos_basics/tirtos_basics.html [Accessed 16 Jun. 2018].

8. Dev.ti.com. (2018). *TI Resource Explorer*. [online] Available at: http://dev.ti.com/tirex/#/DevTool/CC3220SF-LAUNCHXL/?link=Software%2FSimpleLink%20CC3220%20SDK%2FSimpleLink%20Academy%2FWi-Fi%2FWi-Fi%20MQTT [Accessed 6 Jul. 2018].

9. Ti.com. (2018). *UniFlash CC3120, CC3220 SimpleLinkTM Wi-Fi® and Internet-on-a chipTM Solution ImageCreator and Programming Tool*. [online] Available at: http://www.ti.com/lit/ug/swru469b/swru469b.pdf [Accessed 30 Jun. 2018].

10. Ti.com. (2018). *SimpleLinkTM Wi-Fi® CC3220 Out-of-Box Application*. [online] Available at: http://www.ti.com/lit/ug/swru473/swru473.pdf [Accessed 13 Apr. 2018].

11. E2e.ti.com. (2018). *SimpleLink™ WiFi CC31xx/CC32xx - TI E2E Community*. [online] Available at: https://e2e.ti.com/support/wireless_connectivity/simplelink_wifi_cc31xx_cc32xx/ [Accessed 8 May 2018].

12. Ti.com. (2018). *CC3220 SimpleLinkTM Wi-Fi® and Internet of Things Solution, a Single-Chip Wireless MCU Getting Started Guide.* [online] Available at: http://www.ti.com/lit/ug/swru461b/swru461b.pdf [Accessed 24 Jul. 2018].

13. Dev.ti.com. (2018). *SimpleLink CC3120/CC3220 Host Driver: Wlan.* [online] Available at: http://dev.ti.com/tirex/content/simplelink_cc32xx_sdk_1_40_00_03/docs/wifi_host_driver_api/html/group___wlan.html#gaca3874254b60776b2dd60b9751a73697 [Accessed 12 Jun. 2018].

14. Interlinknetworks.com. (2018). *EAP-PEAP and EAP-TTLS Authentication with a RADIUS Server.* [online] Available at: https://www.interlinknetworks.com/app_notes/eap-peap.htm [Accessed 16 May 2018].

PART 2: APPENDIX

Appendix A: Tables

AP	Access Point
CA	Certificate Authority
CCS	Code Composer Studio
EAP	Extensible Authentication Protocol
IoT	Internet of Things
IT	Information Technology
MCU	Microcontroller Unit
MQTT	Message Queuing Telemetry Transport
NWP	Wi-Fi Network Processor
PEAP	Protected Extensible Authentication Protocol
RTOS	Real-Time Operating System
SC	Smart Configuration
SSL	Secure Sockets Layer
STA	Station
TCP	Transmission Control Protocol
TI	Texas Instruments
TI-RTOS	Texas Instruments Real-Time Operating System
TLS	Transport Layer Security
TTLS	Tunneled Transport Layer Security
UART	Universal Asynchronous Receiver-Transmitter
UDP	User Datagram Protocol

Table 8: Acronyms

Appendix B: Block Diagrams

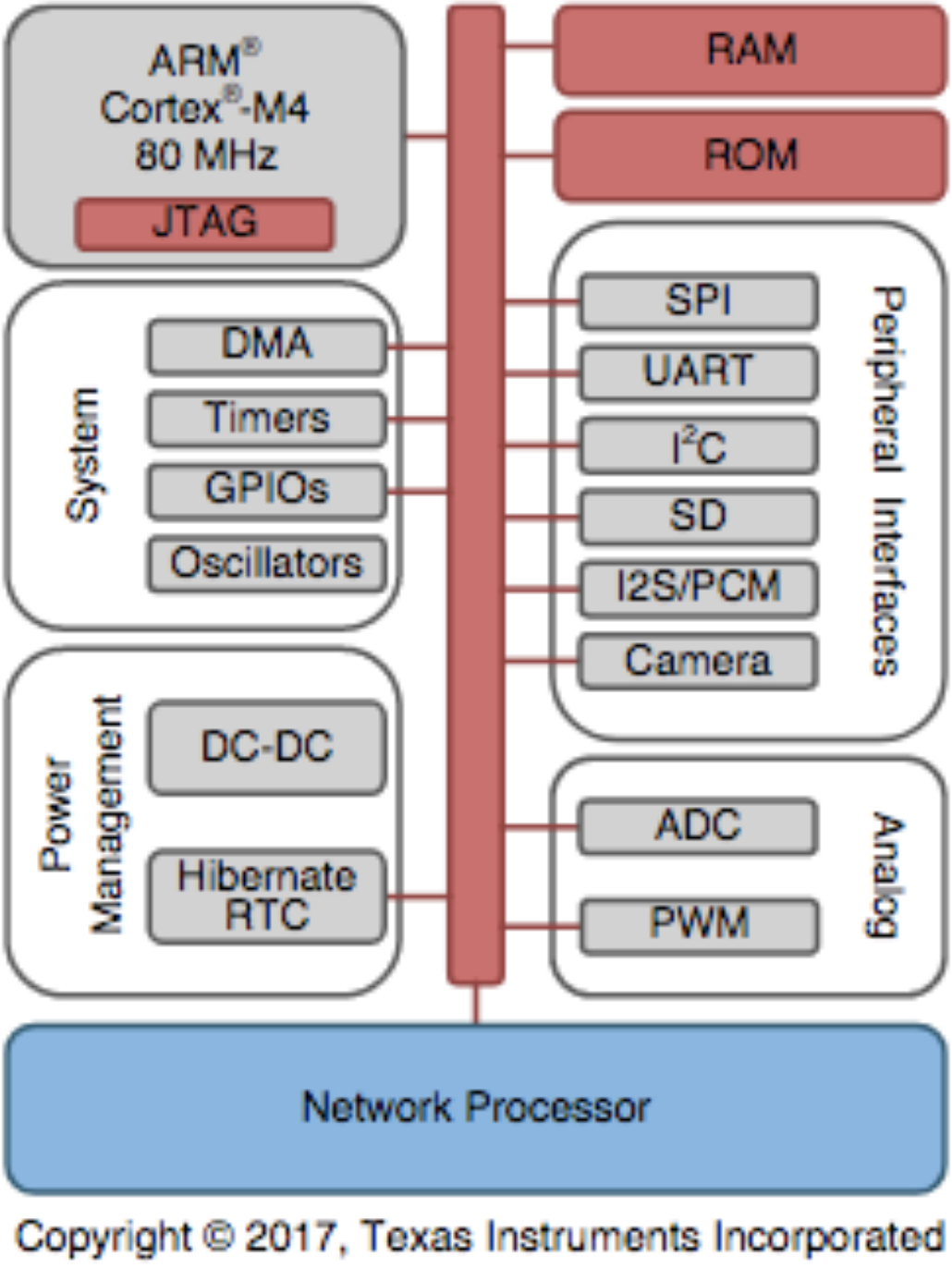


Figure 40: CC3220x Hardware overview

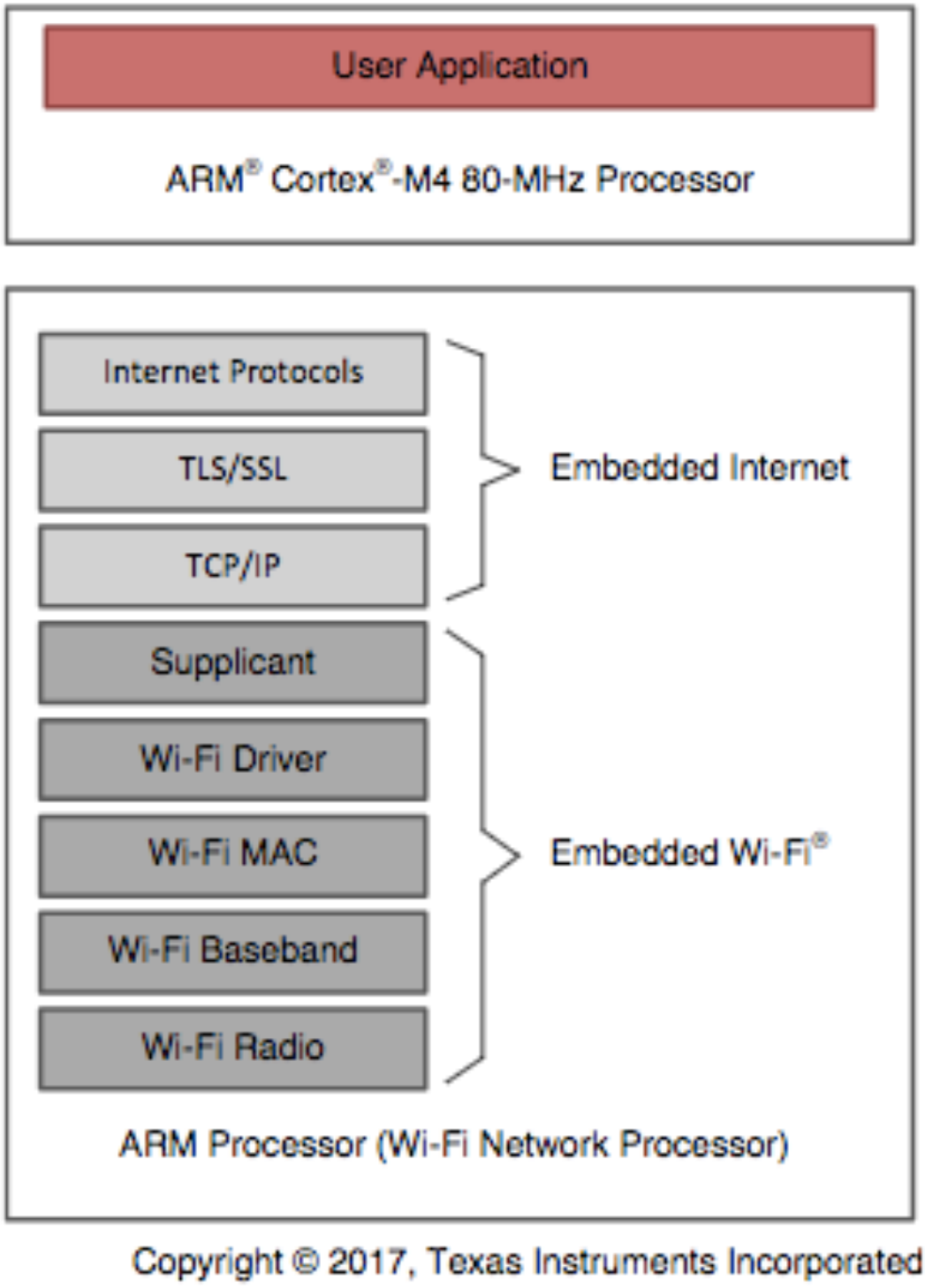


Figure 41: CC3220x Embedded Software overview

Appendix C: Project Overview

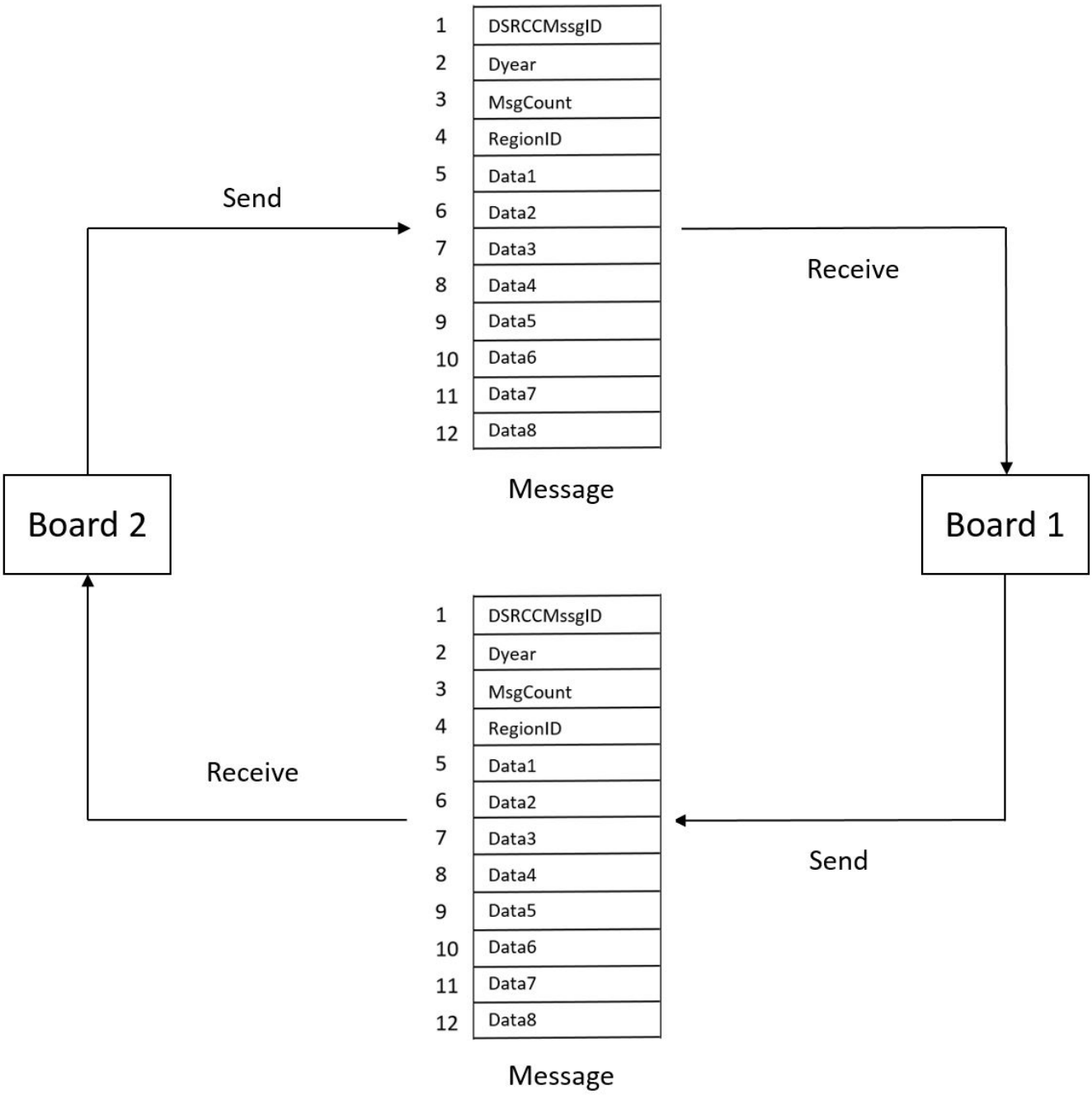


Figure 42: Project schema