Proyecto 41-mqtt_01

Austral 2023 - Informatica Electronica - Austral
EAMartinez

El objetivo de este proyecto es mostrar como una plaqueta realizada con ESP32 y conteniendo 4 LEDs y un pulsador, puede comunicarse con un *broker* MQTT de manera de informar la opresión del pulsador y recibir las ordenes de prendido y apagado de los 4 LEDs que posee cada placa. De esta manera, a través del *broker* se permite que esta placa (y hasta un total 4 placas de esta misma caracteristica) puedan formar parte de un conjunto de IoT donde el *broker* MQTT es el elemento central de comunicación del flujo a través de todo el sistema

Hardware

Placa de desarrollo de ESP32
Pulsador Tact
3 LEDs externos: 1 rojo, 1 amarillo, 1 verde
3 Resistores de 220 ohm
1 LED interno
Cables Dupont

Conexiones

Conectar el riel negativo del *protoboard* (el azul) al GND en placa de desarrollo de ESP32.

Para cada LED externo:
* De GND al cátodo del LED
* Del ánodo del LED al resistor de 220 ohm
* Del otro extremo del resistor de 220 ohm al GPIO respectivo

Conectar a los siguientes GPIOs:
* LED rojo a 'LED_RED'
* LED amarillo a 'LED_YEL'
* LED verde a 'LED_GRN'

Desde GPIO 'PUSH' a un extremo del pulsador Tact
Del otro extremo del pulsador Tact al riel negativo

Para identifiación de plaqueta usamos dos GPIOs *IB0* e *IB1*:

IB1	IB0	board
GND	GND	0
GND	Open	1
Open	GND	2
Open	Open	3

platformio.ini

Definiciones de salidas

- 1 por cada LED externo
- 1 para el LED interno

Definiciones de entrada

- 1 para el pulsador Tact
- 2 bits para la enumeración de plaqueta
  [Nota: la numeración de plaqueta va de 0 a 3: si se deja sin conectar IB0 e IB1, da el número 3]

***strings* de identificación para MQTT:**

En la conversación con el broker MQTT, todos los *strings* están precedidos por 'MAIN_NAME' el cual es seguido por 'SUB_NAME' que es diferente para cada grupo que se comunique a través del broker.

Para la identificación inicial en el *broker* (*broker log*) y para el proceso de suscripción, es continuado por el número de plaqueta o equipo que va de 00 a 99.

Para publicación de un *tópico*, a los items previamente mencionados se le agrega el *string* del tópico.

A continuación, aqui hay ejemplos para el caso de *SUB_NAME*="inel00", *plaqueta*= 00, *tópico*="button":

broker log: *"AustralFI_inel00_00"*
publicación tópico: *"AustralFI/inel00/00/button"*

En el caso de suscripción, no hay un formato fijo ya que depende de la existencia de *comodines* de MQTT

Simbolos definidos

Como es usual, en *platformio.ini* se suelen definir los GPIO espe©ificos que se van a usar, que, por todo lo ya expkicado, no necesitan ser comentados

- LED_RED
- LED_YEL
- LED_GRN
- LED_INT
- PUSH
- IB0
- IB1

También se definen los siguientes símbolos:

- MAIN_NAME: Nombre principal del *string* de identificación y de suscripción
- SUB_NAME: Nombre adicional del *string* de identificación y de suscripción
- MQTT: entre 0 y 2, para elegir cual de los *brokers* se va a usar
- WIFI: entre 0 y 2, para elegir cual par de identificación y clave de acceso a WiFi se va a usar
- SUB_LIST: entre 0 y 2, para elegir cual lista de suscripciones se va a elegir

Bibliotecas.

Se colocan aquí dos bibliotecas a cargar

- **PubSubClient**: implementa el acceso a brokers MQTT

- **TelnetStream**: Para poder enviar por Telnet notificaciones o información de *debugging*; en este caso, especificamente vinculadas con las rutinas de accion de las suscripciones en el archivo *mqtt_actions.cpp*.

Estructura del programa

El programa, que se encuentra en el directorio *src*, está estructurado mediante módulos de C de compilación separada, con lo cual posee una estructura jerárquica claramente establecida:

main.cpp

En efecto, el programa principal está realizado en el archivo *main.cpp* que, como se puede observar tiene una estructura muy sencilla; siendo el programa principal, todas las acciones que realiza están vinculadas a tres objetivos claramente definidos:

- El acceso a la conexión WiFi. (*wifi_ruts.h*)
- El manejo del hardware propio de la plaqueta, que básicamente son 4 LEDs y un pulsador (*hw.h*)
- La comunicación con el *broker* MQTT (*mqtt.h*)

Por ello se incluyen los archivos *.h* que se mencionan.

Existen solo dos funciones en dicho programa principal:

- *setup*: permite la inicialización de los módulos que depende de este programa
- *loop*: Se realizan dos acciones: Si se oprimió el pulsador, entonces se publica la novedad al *broker* y, de cualquier manera, se verifica si se recibió alguna novedad del *broker* mediante *test_mqtt*.

Manejo de WiFi

Aparte del archivo de encabezamiento *wifi_ruts.h* que posee los prototipos de las funciones de manejo de WiFi, se encontará el archivo *wifi_ruts.cpp* que justamente tiene la implementación de dichos prototipos.
También existe un archivo de configuración *wifi_data.h* donde se pueden colocar la identificación y la clave para el acceso de hasta tres lugares que sean usuales en el trabajo de desarrollo del grupo; cual de esos lugares en definitiva se conectará en el momento de ejecución, depende del símbolo *WIFI* que se encuentra definido en *plaformio.ini*.

Manejo de hardware de la plaqueta

Como se mostró anteriormente, el archivo donde están los prototipos de las funciones utilizadas por *main.cpp* de manejo del hardware es *hw.h*; sin embargo, no todo el manejo de *hardware* está declarado en este archivo, ya que solo está el que necesita el programa principal; específicamente, el manejo de los LEDs, que forma parte de la entrada desde el *broker*, se encuentran los prototipos en el archivo *hw_actions.h*.

Obviamente, la implementación de las funciones se encuentran en *hw.cpp* y en *hw_actions.cpp*, respectivamente

Manejo de la comunicación con el *broker* MQTT

Los nombres de los archivos vinculados a este ítem comienzan con *mqtt*, de los cuales pueden destacarse los siguientes:

mqtt.cpp: implementa las funciones principales cuyos prototipos están en *mqtt.h*, archivo que, a su vez, fué incluído dentro de *main.cpp* como ya se explico más arriba.
Las tres funciones públicas que implementa este archivo son las llamadas desde *main.cpp*

Utilizando este archivo, se van a recibir las novedades del *broker* a que se han suscripto y estas novedades, en este caso particular, sirven para cambiar el estado de los 4 LEDs; por lo tanto, esas acciones se realizarán mediante funciones cuyos prototipos están definidos en *mqtt_actions.h* y que están implementadas en *mqtt_actions.cpp*; de hecho, la inclusión de enviar información por Telnet se incluyo

específicamente para enviar la información de la rutina de acción vinculada a cada una de las suscripciones cada vez que se recibia la información de un cambio en alguna de las suscriopciones, para el caso que el ESP32 no estuviese vinculado al monitor de PlatformIO mediante USB.

Se dispone de un archivo adicional referido a *mqtt* cual es *mqtt_def.h*: este archivo, incluído en *mqtt.cpp*, es particularmente importante, ya que sirve para hacer configuraciones en relación con el *broker*.

Configuración del proyecto

El proyecto se configura mendiante tres elementos:

Configuración de *harwdare*

Este proyecto permite tener hasta 4 plaquetas con el mismo programa intercomunicadas en la nube IoT; para ello, y para que su función en IoT sea diferenciada de sus hermanas, es necesario identificarlas, por lo cual se dispone de dos bits con las que se las puede numerar de 0 a 3, como se dijo previamente; para ello, ver como conectar esos dos bits bajo el subtítulo *Conexiones*.

Configuración mediante el archivo *mqtt_def.h*

C1 -> List of topics and actions: En este arreglo de estructuras, se encuentran los pares formados por *tópico* y *acción* que se pueden ofrecer para suscribirse en el *broker*. Solo hace falta configurar en este item si se agregan tópicos adicionales.

C2 -> Subscription lists: aquí aparece la lista de suscripciones que se desea realizar; obsérvese que, para generar el *string* de suscripción, se comienza con *'MAIN_NAME/'SUB_NAME'/* y se continúa con el sub-string indicado en cada ítem de esta lista o tabla; obviamente, los caracteres reservados **#** y **+** son los consasbidos comodines de MQTT (En caso de duda, ver la función *init_subscriptions* en el archivo *mqtt.cpp*). Notar que, con la finalidad de poder probar cómodamente otras suscripciones posibles y comprender como funciona MQTT, se ha pensado preparar hasta 3 juegos de suscripciones que se pueden elegir desde *platformio.ini* mediante el símbolo *SUB_LIST*.

C3 -> Macros for id strings: Aqui se usan las definiciones que estan en *platformio.ini* de los símbolos *MAIN_NAME*, *SUB_NAME* y, además, el número de placa o equipo en cuestión para generar los *strings* de identificación para el *broker*, tanto sea para el *log* en el *broker*, como para la publicación de un tópico o para la suscripción.

C4 -> Broker selection: También para comprender que el *broker* MQTT puede estar ubicado en distintos lugares, existen 3 selecciones del *broker* en particular a usar, la cual se puede seleccionar mediante el símbolo *MQTT* que se define en *platformio.ini*. Aquí, en realidad, se debe colocar la forma de acceso y validación de cada uno de los 3 *brokers* a usar.

Configuración mediante *platformio.ini*

Esta configuración acompaña las anteriores y ya se explicó como influye en la configuración total.

Luego que Ud. cargue este proyecto en su computadora, va a encontrar las siguientes configuraciones que aparecen por *default* y que son las que quedaron luego que el profesor mostrase el funcionamiemto en clase:

MQTT = 0 -> apunta al *mosquitto* corriendo en la máquina. Si va a usar esta configuración, recuerde de cambiar en *mqtt_def.h* el número de IP de la máquina donde está corriendo *mosquitto* a usar

WIFI = 1 -> apunta a la definición del WiFi de la Facultad; si está trabajando en otro lugar, recuerde en cambiar la identificación y la palabra clave de ese lugar en el archivo *mqtt_def.h* y también cambiar en *platformio.ini* en el símbolo **WIFI** el número que apunta a ese lugar.

SUB_LIST = 1 apunta a la lista de suscripciones que se usaron para mostar en clase. Se recomienda que, si recién empieza a trabjar con *MQTT* y está haciendo las primeras pruebas, coloque en este parámetro el valor **0** que permite suscribirse a todo.

Prueba con broker MQTT mosquitto

Para ejemplificar la prueba realizada, se ha usado un broker *mosquitto* sobre una computadora con Ubuntu 18.04 que se encontraba en la misma red que los microcontroladores ESP32 utilizados.
Las configuraciones son las que se muestran en el código que esta en *github* tal cual queda después de clonarse.
El único cambio que deberia hacerse es el número de IP del broker *mosquitto*

Broker mosquitto

Este broker está disponible para ser cargado en cualquiera de los sistemas operativos
Debe cargarse tanto el *broker mosquitto* asi como *mosquitto-clients*: bajo Ubuntu, se instaló sencillamente con la siguiente línea de comando:

```
$ sudo apt install mosquitto mosquitto-clients
```

Normalmente, el solo hecho de cargarlo ya lo hace correr y queda esperando conexiones; obviamente, debe cambiar en el archivo *mqtt_def.h* la dirección de IP donde se encuentra la máquina donde fue cargado mosquitto y asegurarse que en *platformio.ini* la configuración MQTT apunta al caso configurado.

Placas de hardware

Se han construido 4 placas de hardware con ESP32 una de las cuales posee 3 LEDs (rojo, amarillo, verde); obviamente, las cuatro placas utilizan el led interno de los ESP32 asi como las 4 placas poseen un pulsador; se muestra foto más abajo.

La forma mas rápida de construir una placa única que pueda utilizarse en esta prueba es solamente conectar el pulsador entre el GPIO 'PUSH' y GND (lo cual puede hacerse solamente con dos cables); la identificacion de placa en ese caso sera la 03, ya que los GPIO IB0 e IB1 quedan en sin conexión externa y están con *pull_up* interno; de hecho, si el alumno quiere hacer parte de las purebas hechas en clase, esta es la forma rápida que se recomienda para construir una sola placa que se conecte al sistema.

Todas las placas, cuando arrancan, se conectan al broker *mosquitto*, leen su número de placa y se suscriben a las novedades como se ve en el archivo *mqtt_def.h*; para el caso recomendado para las primeras pruebas, que corresponde a la configuración SUB_LIST = 0, va a recibir las novedades de cualquier origen.

Obviamente, aquellas placas que no tienen LEDs externos no podran mostrar la suscripcion que reciben.

La única información que se publica desde la placa es el cierre del pulsador, que corresponde al tópico *button*.

Se aconseja para probar cada placa, conectarlas sucesivamente a la PC por el cable USB, cargar el programa y poner a correr el monitor serie; una vez que esté corriendo, hacer reset en la plaqueta de ESP32 y observar la información que aparece en la pantalla porque le dará datos valiosos para una eventual busca de problemas. Ver foto más abajo de una captura de pantalla en estas condiciones.

Comandos de los clientes de *mosquitto*

Existen dos clientes distintos de *mosquitto* que pueden usarse para *debugging* de la instalación::

- Uno para publicar novedades, que se denomina *mosquitto_pub*
- Uno para suscribirse a novedades, que se denomina *mosquitto_sub*

Para las pruebas conviente tener abiertas dos terminales, una para poder publicar y la otra para poder observar las novedades suscriptas; se aconseja tener abierta una tercera terminal con el monitor serie conectado al ESP32

El comando para publicación será de este tipo (se muestran todas las publicaciones posibles teniendo en cuenta las suscripciones realizadas en el ESP32); obsérvese que se ha usado para la publicacion un numero de plaqueta o de equipo 99, para indicar que no se trata de ninguna de las placas posibles. Para cada publicación, debe enviarse el comando interactivamente, luego de modificar los argumentos

```
mosquitto_pub -t AustralFI/inel00/99/tr -m any        <- tr significa Toggle Red
mosquitto_pub -t AustralFI/inel00/99/ty -m any        <- ty significa Toggle Yellow
mosquitto_pub -t AustralFI/inel00/99/tg -m any        <- tg significa Toggle Green
mosquitto_pub -t AustralFI/inel00/99/ti -m any        <- ti significa Toggle Internal
mosquitto_pub -t AustralFI/inel00/99/sr -m 0          <- sr significa Set Red a 0
mosquitto_pub -t AustralFI/inel00/99/sr -m 1          <- sr significa Set Red a 1
mosquitto_pub -t AustralFI/inel00/99/sy -m 0          <- sy significa Set Yellow a 0
mosquitto_pub -t AustralFI/inel00/99/sy -m 1          <- sy significa Set Yellow a 1
mosquitto_pub -t AustralFI/inel00/99/sg -m 0          <- sg significa Set Green a 0
mosquitto_pub -t AustralFI/inel00/99/sg -m 1          <- sg significa Set Green a 1
mosquitto_pub -t AustralFI/inel00/99/clear -m any     <- clear significa apagar los
LEDs
```

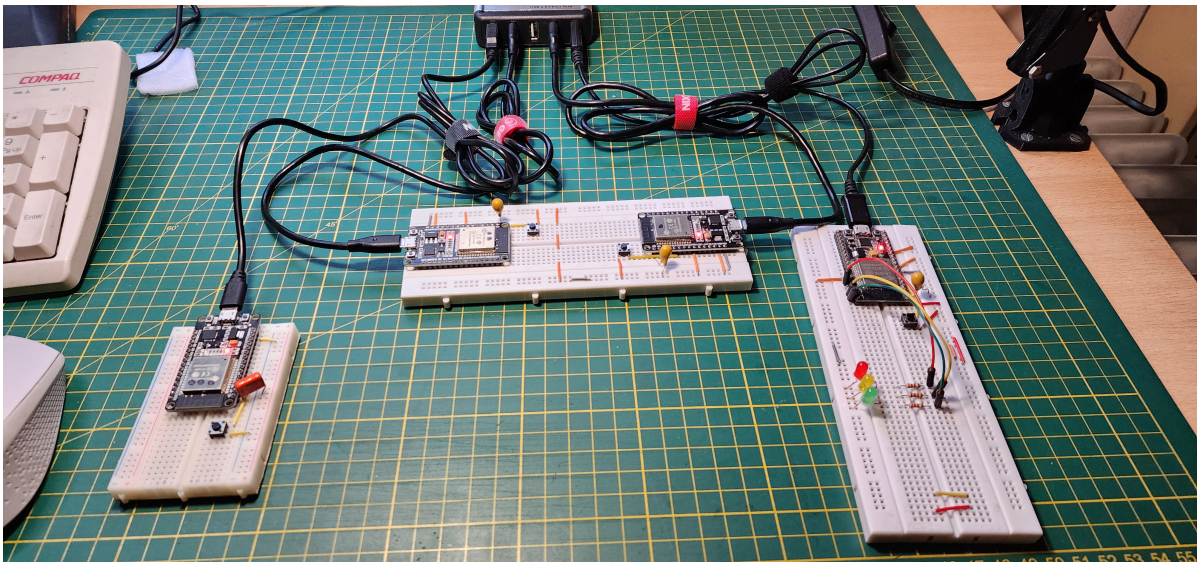
En el caso del comando de suscripción, debe enviarse y quedará en espera de la llegada de una novedad; se muestra la llegada de algunas novedades, tanto de las placas por la opresión del pulsador, como de las enviadas por *mosquitto_pub*.

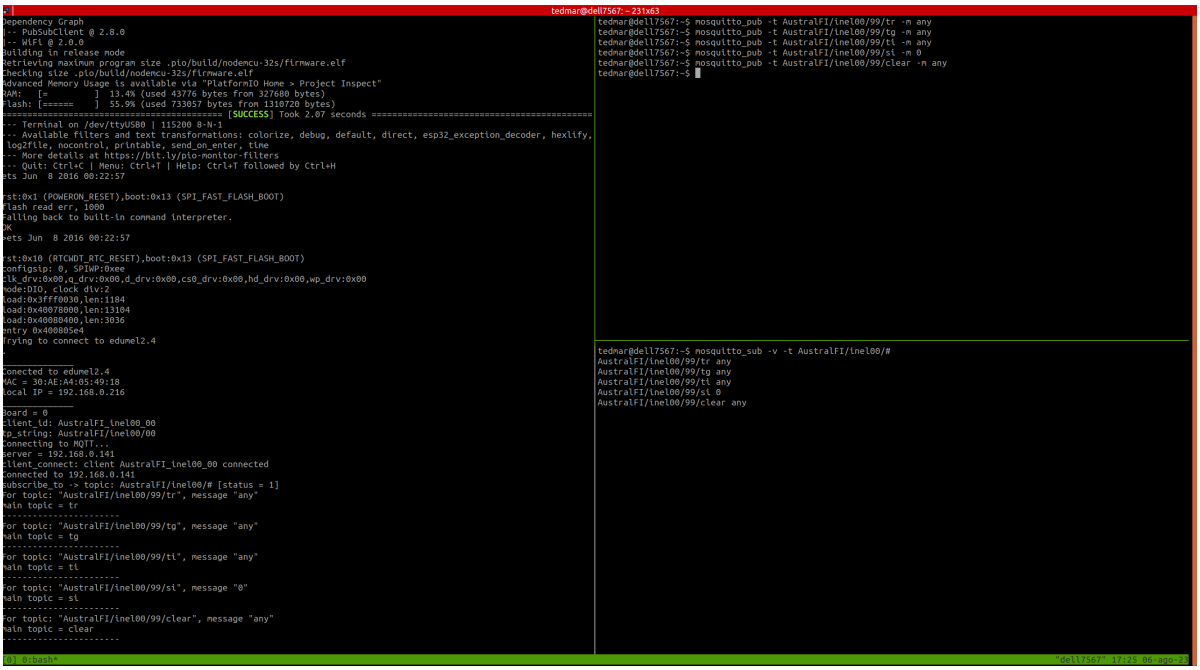
```
mosquitto_sub -v -t AustralFI/inel00/#

AustralFI/inel00/00/button button
AustralFI/inel00/03/button button
AustralFI/inel00/99/tr 0
AustralFI/inel00/99/tg 0
AustralFI/inel00/99/sy 1
AustralFI/inel00/02/button button
```

Conexión a WiFi

No olvide cambiar en *mqtt_def.h* la identificación así como la clave de los puntos de conexión de WiFi donde Ud. desarrolla su actividad (hay hasta tres disponibles) y cambiar concordantemente la defición de cual está usando en *platformio.ini* mediante el símbolo *WIFI*

Fotos





Tutoriales

MQTT

[MQTT: The Standard for IoT Messaging - from mqtt.org]

[10 Free Public MQTT Brokers(Private & Public)]

[mosquitto download]

Node-Red

[Node-Red Tutorials - From nodered.org]

[Step by step guide to install Node-RED on Ubuntu 18.04 LTS]

Videos

MQTT

[Desentrañando MQTT - ¿Cómo funciona?]

[MQTT & ESP32.: (1). Ejemplo sensor de temperatura DHT22]

Node-Red

[Introduction - Node-RED Essentials - From nodered.org]

[Qué es Node-RED? - Introducción, Node-RED en la Industria, Node-RED e IoT]