Proyecto 40-mqtt

Secciones

Objetivo

> El objetivo es comunicar dos plaquetas mediante MQTT.

> El conjunto permitirá lo siguiente;
>
> - Hacer que la posición del potenciómetro en una de las placas cambie la posición del servo en la otra
> - Hacer que la opresión de un pulsador en una placa, cambie el estado de un LED en la otra

Medios utilizados

- Las plaquetas base que pueden ser utilizadas son las siguientes:
  - Wemos D1 R2 (con *ESP8266*)
  - Wemos D1 R32 (con *ESP32*)
- Sobre la plaqueta base, se coloca un *shield* que es el utilizado en IAE
-
- Para conveniencia de reconocimiento, se sugiere rotular a cada conjunto de base y *shield* con los nombres *G1* y *G2*
-
- Se utilizarán servidores de MQTT para la prueba de dos tipos:
  - Servidor propio: para ello, se sugiere usar el servidor *mosquitto* en Linux
    - Se puede instalar sobre una máquina Linux o sobre Raspberry Pi corriendo *Raspbian*
  - Servidor en el Web
    - En el ejemplo, se ha utilizado ClodMQTT

Programa

El programa está dividido en cinco archivos, a saber:

> *main.cpp*: programa principal, independiente del *hardware*

> *hard.h*: archivo de inclusión con el manejo específico de *hardware*

> *defs.h*: definiciones (aprte de las que están en *platformio.ini*.

> *wifi_ruts.cpp*: rutinas de manejo de WiFi

> *wifi_ruts.h*: prototipos de las rutinas de manejo de WiFi

- Archivo ***main.cpp***
    - Entre las líneas 5 y 18, se colocan las inclusiones de archivos, tanto los del sistema, como de las bibliotecas y los propios; en este último caso, se trata de *wifi_ruts.h* y de *defs.h*
    - Entre las líneas 21 y 35, mediante compilación condicional determinada por el símbolo *MQTT*, se elige entre el servidor en el Web (*MQTT* == 1) y el servidor local (*MQTT* == 2)
    - Entre las líneas 42 y 46, se define un nuevo tipo de dato que permitirá almacenar duplas formadas por un subtópico y el mensaje asociado: dicho tipo se denomina *topic_t*
    - En las líneas 48 y 49 se instancian dos objetos del tipo *WiFiClient* y *PubSubClient*
    - En la línea 51 se define un arreglo de caracteres *client_id* que permite alojar nombre de cada cliente que se conecte a MQTT
    - Finalmente, en la línea 53 se incluye el archivo *hard.h* que posee todo el código dependiente del *hardware*.
        - Debe aclararse que es una práctica no muy común colocar código ejecutable en un archivo *header* pero se lo ha hecho para ser más fácil para el neófito.
        - Por lo tanto, se debería seguir leyendo por ese archivo, tal cual lo hace el compilador; sin embargo, por unidad de explicación se seguirá por el archhivo *main.cpp*
    - A partir de la linea 55 en adelante, se encuentra el código *independiente* del *hardware*
    - Función *callback*
        - Cada vez que el *broker* MQTT informa que hubo un cambio en un tópico al cual la estación está suscripta, llama a esta función, con el tópico *topic* y el mensaje *message*.
        - Para todas las duplas contenidas en el arreglo *topics*, se busca si coincide el tópico recibido con alguno de los allí almacenados y, si es así, se ejecuta la acción almacenada en la dupla.
    - Función *init_mqtt*
        - A través del cliente de MQTT creado en la línea 49 con el nombre *client*, se intenta conectar con el servidor elegido.
        - Para ello, en la línea 76 se indica la dirección de Internet donde se encuentra el servidor y el puerto de acceso, en la línea 77 se registra en el cliente la dirección de la rutina de *callback*.
        - Comenzando en la línea 78, se comienza un ciclo de intento de conexión; en efecto, en la línea 82 se invoca el método del cliente denominado *connect* a

quien se le pasa la identificación de este cliente, el usuario y la clave de conexión.

- Función *setup*
  - A partir de la línea 97, se realizan las siguientes acciones
    - Obtener el *string* de identificación de este cliente.
    - Inicializar la comunicación serie
    - Conectarse a WiFi
    - Inicializar el *hardware*
    - Inicializar las suscripciones deseadas en el servidor
    - Prender el led de la placa en señal de conexión Ok.
- Función *loop*
  - Llamar a la función *verify_changes* para determinar si hubo cambios en el *hardware* que haya que publicar en el servidor.
  - Llamar al método de *client* denominado *loop* que verificará si hay novedades de las suscripciones deseadas y, en caso afirmativo, llamar a la función de *callback* oportunamente registrada.

- Archivo ***hard.h***
  - Como se dijo anteriormente, este archivo es absolutamente dependiente de *hardware* y, en este caso particular, manejará los siguientes periféricos
    - LED
    - Pulsador
    - Potenciómetro
    - Servo motor
  - En la línea 5 se crea un obeto del tipo *Servo*
  - Función *get_client_id*
    - Las líneas 7 a 11 se utilizan para generar la identificación de cada cliente para el servidor.
  - Función *init_hardware*
    - Las líneas comprendidas entre 18 a 27 se realiza la inicialización de todo el *hardware*, a saber:
      - LED como salida
      - Pulsador como entrada, con resistencia de *pullup*
      - Apagar el LED
      - Anexar al objeto servo el *pin* de servicio del servo
      - Inicializar la primera posición del servo en el ángulo 0
  - Función _led_setup_done
    - Se cambia el estado del LED
  - Función *subscribe_to*
    - Recibe el subtópico

        - Arma el *string* de tópico en el formato *tópico_base/grupo/subtópico*; por ejemplo sería *espxx/g1/toggle* o *espxx/g2/angle*.
        - Envía el *string* de suscripción al servidor.

- Función *init_suscriptions*

    - Se suscribe, a través de la función anterior, a los subtópicos *toggle* y *angle*

- Función *change_led*

    - Acción relacionada con la suscripción a subtópico *toggle*
    - Recibe el mensaje (*msg*) del servidor pero no tiene efecto en la acción.
    - Cambia (*toggle*) el estado del led

- Función *write_servo*

    - Acción relacionada al subtópico *angle*
    - Recibe como mensaje (*msg*) del servidor el ángulo a girar como texto
    - Envía la orden de girar al servo del ángulo enviado como mensaje previamente convirtiendo a un número entero.

- Arreglo *topics* de estructuras tipo *topic_t*

    - Agrupa subtópico y acción a realizar

- Función *do_publish*

    - Recibe el subtópico y el mensaje
    - Arma el *string* de tópico
    - Lo publica en el servidor

- Entre las líneas 103 a 107 se definen variables globales para la observación del pulsador

- Función *verify_pushbutton*

    - Esta función trata de encontrar el cierre del pulsador como transición efectiva para informar al servidor (la de apertura se debe detectar pero no se informa).
    - De todas maneras, para tomar dichas transiciones se deben evitar los típicos rebotes (*bounce*) de cierre o apertura de contactos a través de un temporizador que, en este caso es de 50 mseg (línea 107).
    - La decisión de publicación se encuentra en las líneas 126 y 127

- Entre las líneas 136 y 139 se crean e inicializan las variables globales necesarias para la función *verify_pote*.

- Función *verify_pote*

    - Esta función verifica el valor de posición del potenciómetro.
    - El potenciómetro está conectado a 3.3v y su punto medio está conectado al conversor A/D *ANAIN*.
    - Se realiza la medición cada 100 mseg., lo cual está dado por la línea 138.

            - La medición del conversor A/D será entre 0 y RANGE, de acuerdo a la cantidad de bits del conversor.
            - El valor leído se convierte de 0 a RANGE a 0 a 180 mediante la función *map*.
            - Este valor de ángulo tendrá una resolución de *skip_angle*.
            - Si en estas condiciones, este valor de ángulo es distinto al del muestreo anterior, entonces se publica en el servidor.
        - Función *verify_changes*
            - Se llama sucesivamente a:
                - *verify_pushbutton*
                - *verify_pote*
- Archivo **_defs.h**
    - En este archivo, se colocan algunas definiciones.
    - La mayor parte de las definiciones están en *platformio.ini*
    - Entre las líneas 14 y 20, está como se arma el grupo dentro del tópico.
    - Como las acciones de ambas placas son simétricas, entonces aquí lo que se decide es quien es *ME* y *YOU*.
    - Si la placa es la G1, entonces se deberá poner en *platformio.ini* como *BOARD* el valor 1, sino se deberá colocar 2.
    -
    - Entre las líneas 31 y 33 están los strings correspondientes al tópico principal (*espxx* en nuestro caso) y como es el grupo en el tópico.
- Archivo ***wifi_ruts.cpp***
    - Códigos referidos al manejo de WiFi
- Archivo ***wifi_ruts.h***
    - Archivo de inclusión con los prototipos de las funciones del anterior

---

platformio.ini

> ***NOTA IMPORTANTE***:

> En el caso de la placa con ESP32, no funciona adecuadamente la obtención del posicionamiento del potenciómetro. Ello se debe a que la placa de *shield* IAE fué hecha para trabajar con *ESP8266* y en éste sólo existe un conversor A/D que se conecta en la posición ***0*** del la fila de conectores de analógicos. En el caso del *ESP32*, este contacto corresponde al conversor ADC2 que no funciona si se usa simultáneamente WiFi

Constantes

> A continuación, se explican las constantes que se definen en *platformio.ini*

- ***[platformio]***
    - Aquí se colocan cuál o cuales son los ambientes que se compilarán

- ***[env]***
    - Ambiente con definiciones comunes a todos los demás ambientes
    - ***framework***
        - Todos los ambientes usan el *framework* de *arduino*
    - ***build_flags***
        - *MOSQUITTO_IP*: *string* con el número de IP del servidor de *mosquitto* cuando es local
        - *MY_SSID*: *string* con el nombre de AP para conectarse a WiFi
        - *MY_PASS*: *string* con la clave de WiFi
        - *BOARD*: 1 para la placa *G1*, 2 para la placa *G2*
        - *MQTT*: 1 para el servidor en el Web, 2 para el servidor local
- ***[env:wemos_d1_mini32]***
    - Aquí se colocan las definiciones válidas para *ESP32*
    - *monitor_speed*: baud rate de conexión de la PC a la placa
    - ***build_flags***
        - Aquellas definidas en ***[env]***
        - *PIN_BUTTON*: IOport donde está conectado el pulsador
        - *PIN_LED*: IOPort donde está conectado el LED
        - *PIN_SERVO*: IOPort donde está conectado el Servo
        - *ANAIN*: IOPort de acceso al conversor AD
        - *RANGE*: Rango del conversor AD
        - _SERIAL_BAUD: Baud rate del port serie (tomado de *monitor_speed*)
- ***[env:d1_mini]***
    - Aquí se colocan las definiciones válidas para *ESP8266*
    - *monitor_speed*: baud rate de conexión de la PC a la placa
    - ***build_flags***
        - Aquellas definidas en ***[env]***
        - *PIN_BUTTON*: IOport donde está conectado el pulsador
        - *PIN_LED*: IOPort donde está conectado el LED
        - *PIN_SERVO*: IOPort donde está conectado el Servo
        - *ANAIN*: IOPort de acceso al conversor AD
        - *RANGE*: Rango del conversor AD
        - _SERIAL_BAUD: Baud rate del port serie (tomado de *monitor_speed*)

Bibliotecas

- ***[env]***
    - ***lib_deps***
        - Nombre de la/s biblioteca/s usada/s por todos los ambientes
        - En este caso, el nombre es *PubSubClient*
        - A pesar que se refiere que es solamente utilizable por *ESP8266_* se puede usar también con *ESP32*

- ***[env:wemos_d1_mini32]***
    - ***lib_deps***
        - Aquellas definidas en ***[env]***
        - *ServoESP32*: para el servo, especial para *ESP32*
- ***[env:d1_mini]***
    - No existen bibliotecas particulares para *ESP8266*
    - Sólo se requiere la ya definida en ***[env]*** y aquellas que acompañan por *default* a *Arduino* (entre ellas la biblioteca *Servo*)