Project Report 2: Smart Devices Communication

Guilherme R. C. - RA: 169127

1 Purpose

The purpose of this module is to use the infrastructure provided by the previous module to enable the communication with external peripherals, namely a temperature sensor and a DC motor, via MQTT [1]. These features were validated using the IoT MQTT Panel app ([2]), on which an inituitive UI was built.

2 Materials

- EA076 kit, including a platform board with multiple peripheral connections, wire wrapping wire and tools for wire wrapping
- FRDM-KL25Z: Freedom Development Platform for Kinetis, featuring MKL25Z128VLK4 MCU – 48 MHz, 128 KB flash, 16 KB SRAM, USB OTG (FS), 80LQFP
- ESP8266 ESP-01 WiFi module (programmed with [3])
- SERIAL-TO-TTL converter (for debugging purposes)
- LM61 Temperature Sensor [4]
- RF-300FA-12350 DC motor [5]
- L293D H-bridge [6]

3 Hardware design

The hardware project, designed in EAGLE, can be found in /external/EAGLE in the .zip file submitted.

4 Software design

First things first. Before discussing the features intended to be implemented in this module, let's recall the most important limitations (at least from my point of view) that the project had and discuss how I managed to overpass them.

4.1 Handling the limitations that the project had

Quoting from the previous Report:

- **Limitation**: "Since a linear buffer is being used [for receiving the characters], the program cannot handle the case in which a second message is received while the first one is being treated. This queueing of the multiple messages feature could be implemented using a ring buffer."

  Solution: The ring buffers (generated by PE) are now being used for each UART.

- **Limitation**: "Also, both UARTs can write in the same linear buffer at any time. There's no protection and neither a single linear buffer for each one (although the program has a different one for the to-be-sent messages)."

  Solution: The problem was that I was buffering the received characters from both UARTs into the same linear buffer. The usage of unique ring buffers for each UART also solves this problem.

In addition to these implementations, I also formalized the usage of an event-driven architecture, in which main loop only task is to check if the event ring buffer has a new event to be handled or not (this architecture was already being used, but not formally).

Note: Here, with the word 'event' I'm not meaning in any way 'interruption'. I'm bringing it to a more abstract level, whose types currently are:
NEW_MESSAGE_FROM_BROKER;
NEW_MESSAGE_FROM_TERMINAL; and
NEW_TEMPERATURE_MEAS.

Enough of software architecture, let's head to the stuff that was intended to be implemented in this module.

4.2 Implementations of this module

The implementations of this module can be divided into: Temperature sensor reading, Real Time Clock (RTC) module configuration and DC motor control.

4.2.1 Temperature sensor reading

The temperature sensor generates an analog signal. Therefore, an ADC was used to read this signal. Also, a periodic Timer Interruption at each 2 ms was used to request a new ADC measurement and to enable an average reading calculation (500 samples/s), reducing the noise from the signal.

Note: The linker doesn't link the sprintf float-to-string conversion object by default as it occupies a bunch of memory. I had to enable it in the build configurations, adding the *-u _printf_float* flag to the linker flags.

4.2.2 Real Time Clock (RTC) module configuration

This module was used to generate time stamps for the temperature readings. To configure this module, I followed the steps from a document [7] hosted in the NXP community website. Thereafter, I adapted the example code provided there to my usage, creating a library to abstract the lowest level stuff.

4.2.3 DC motor control

The hardware interface with the DC motor is provided by an H-bridge. In order to control the speed of this motor, a PWM wave was generated on the 1,2EN pin of L293D [6]. To control its rotation, a combination of complementary signals was generated on the 1A and 2A pins.

4.3 Processor Expert (PE) components

The PE components used and their usage are described in the next table.

PE component	Usage
ADC	For reading the temperature sensor
AsynchroSerial	For UART communication (with UART0 and UART2)
BitIO	For controlling the direction of the DC motor
PWM	For controlling the power of the DC motor
RTC_LDD	For generating time stamps
TimerInt	For requesting to ADC a new temperature reading

4.4 Limitations

I can see 3 major limitations that the current state of the project has:

- Messages can't be queued to be sequentially delivered. I have to be aware of it and send at most one message to each UART until these messages are completely sent. Even with this limitation, the code as it is suffices the communication protocol we're dealing with, but being able to log without worrying if the last log was already sent might be really useful for debugging purposes. An output ring buffer would solve this.

- Whenever a temperature reading is published, the communication FSM expects a "OK PUBLISH" message from the MQTT Broker. If another message arrives (for example one from /power topic) before this confirmation, this one is ignored (for the sake of the proper functioning of the FSM). The case in which more than one message arrives before the confirmation is not handled and breaks the FSM.
  I really don't know how should I architect the software in order to treat these corner cases.

- Sometimes the "OK PUBLISH" message never arrives and the FSM keeps stuck waiting for it. A timeout on the microcontroller side would solve this, republishing the temperature again.
  I don't know why this happens. Since we're using TCP as our network transport layer protocol and it ensures that a packet will be received, it shoudn't happen.

5 How to control the DC motor parameters with MQTT topics

The standard adopted to control the DC motor parameters using MQTT topics is the following.

Parameter	Topic	Message
Direction	EA076/grupoD3/dir	CLOCKWISE or ANTICLOCKWISE
Power	EA076/grupoD3/power	A value between 0 and 100 *
Mode	EA076/grupoD3/mode	ON, OFF or AUTO
Threshold	EA076/grupoD3/threshold	A value between -60.0 and 270.0 **

* - representing the percentage of power
** - representing the temperature of threshold

6 UI design in IoT MQTT Panel app [2]

The following figure shows the UI design made in the IoT MQTT Panel app.

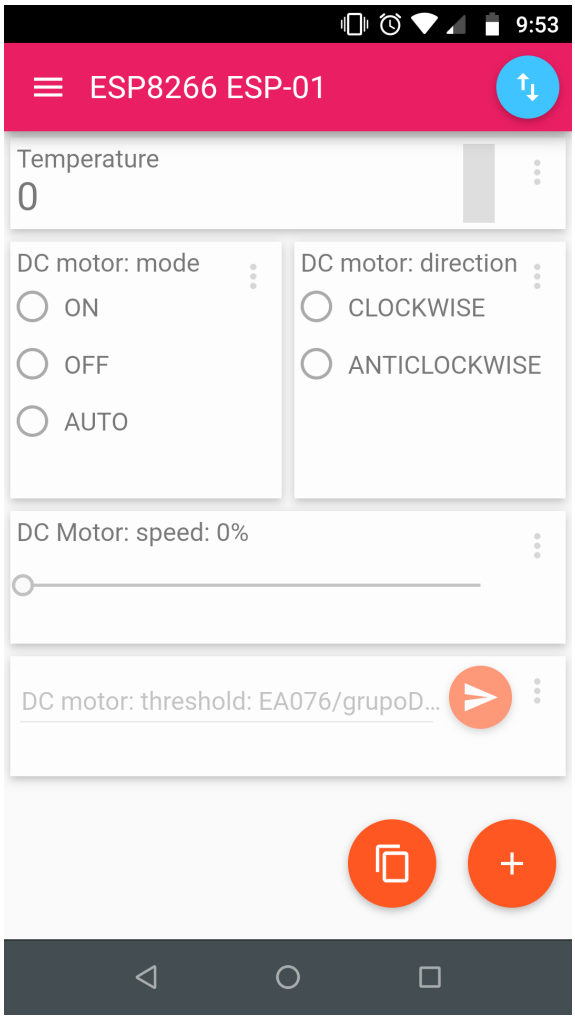

Figure 1: Screeenshot of the UI created using IoT MQTT Panel app [2]

7 Validation

Again, since I developed most of the program away from University, I used a SERIAL-TO-TTL converter acting like the ESP01 module, opening two serial connections in my computer and simulating the communication. Then, when I felt the program was ready to be deployed, I did it using the IoT MQTT Panel app connecting everything to the University's local network.

References

[1] ftp://ftp.dca.fee.unicamp.br/pub/docs/ea076/manuais/MQTT_V3.1_Protocol_Specific.pdf.

[2] https://play.google.com/store/apps/details?id=snr.lab.iotmqttpanel.prod.

[3] ftp://ftp.dca.fee.unicamp.br/pub/docs/ea076/complementos/ESP8266_CLIENTE_MQTT.pdf.

[4] http://www.ti.com/lit/ds/symlink/lm61.pdf.

[5] https://www.neuhold-elektronik.at/datenblatt/N7029.pdf.

[6] http://www.ti.com/lit/ds/symlink/l293.pdf.

[7] https://community.nxp.com/docs/DOC-94734.