Project Report 3: Local Access to Smart Devices And Data Logging

Guilherme R. C. - RA: 169127

1 Purpose

The purpose of this final module is to enable local access to the peripherals (namely temperature sensor and DC motor) and also provide the ability of registering the temperature history.

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 [1])
- SERIAL-TO-TTL converter (for debugging purposes)
- LM61 Temperature Sensor [2]
- RF-300FA-12350 DC motor [3]
- L293D H-bridge [4]
- NOKIA5110 Graphic LCD Display With PCD8544 48 × 84 Pixels Matrix LCD Controller/Driver [5] (SPI Interface)
- 4x3 Matrix Keyboard
- AT24C164 2-Wire Serial EEPROM 16 kbits (2048 x 8 bits) [6] (I2C Interface)

3 Hardware design

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

4 Software design

NOTE: The demo video can be found at
https://drive.google.com/open?id=1-Oa74LDawrevNxyNFmBwHYD1yFhVwodN.

Before digging into the features intended to be implemented in this module, I wanna explain how managed to narrow down my suspicions and solve the program-getting-stuck bug mentioned in the last report.

4.1 Program-getting-stuck bug fix

I empirically found that after publishing exactly 52 temperatures in a row the program was getting stuck. Also, I checked that I was correctly receiving 52 publishment confirmations from MQTT Broker, so it should be a problem with my code. And indeed it was. Counting the number of characters received after these 52 publishment confirmations I got the number 769, which is 6 times the size of UART RX buffer (128 bytes) plus one. This was a clear indication that the problem was occurring at the "border" of the ring buffer and 52 publishment confirmations (plus communication establishment confirmations) was exactly the amount of characters to make the indexes* fall in this corner case. I checked my indexes handling and I realized that I was doing it improperly at the border. I fixed it and now everything seems to be working as designed to.

* The indexes I meant here refer to the indexes inside a function implemented in UART0.c and UART2.c (I modified the generated codes) to check if a complete message has arrived.

4.2 Features of this module

The features implemented in this module can be divide into: (1) NOKIA5110 Graphic LCD Display, (2) 4x3 Matrix Keyboard and (3) AT24C164 2-Wire Serial EEPROM 16 kbits (2048 x 8 bits).

4.2.1 NOKIA5110 Graphic LCD Display

This display is integrated with PCD8544 48 × 84 Pixels Matrix LCD Controller/Driver, which allows us to configure and use it using SPI protocol. The PE components' configuration followed this guide [7] from Erich Styger.

I decided not to follow the proposed finite state machine in [8] and to build my own menu. Splitting the options into multiple menus enables option descriptions on the display, so the user doesn't need to refer a manual in order to discover what a key is expected to do.

4.2.2 4x3 Matrix Keyboard

In order to detect key events, the guide described in this article [9] was followed.

I decided to use the internal pull-up resistors, so I had to select the pull resistor type (that is, pull-up) and enable it in each of the pins connected to the keyboard columns. These configurations were made using the macros from the PORT Peripheral Device Driver (which can be found at Static_Code/PDD/PORT_PDD.h).

To treat the bouncing effect, the solution used was to wait 50 ms after the first detected event, so this bouncing effect would be already ceased (empirically confirmed) and new events can be correctly detected.

4.2.3 AT24C164 2-Wire Serial EEPROM 16 kbits (2048 x 8 bits)

In order to configure this device, a guide from our teacher was used. I'm not putting the link here since it would need previous authentication on UNICAMP'S Moodle. The calculations involved in the choice of SCL divider can be found /external/calculations.

Since if a single byte or whole page writing into the memory take 10 ms, it's clever to write a whole page, getting a higher bit rate of transmission. So I accumulated enough temperature readings before writing them into the memory.

I should have saved the raw temperature value read from the ADC since it takes only 2 bytes, instead of 4 that the string representation does. I didn't have enough time to make this change.

I also should have treated the EEPROM memory as a memory ring buffer. I'm not protecting in any means its complete fillment.

5 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
RTC_LDD	For configuring a trustable Real Time Clock source
AsynchroSerial	For UART communication (with UART0 and UART2)
BitIO	Multiple usages
PWM	For controlling the power of the DC motor
PDC8544 *	For driving the NOKIA5110 Graphical LCD Display
ExtInt	For identifying to which column a pressed key belongs
24AA_EEPROM *	For writing and reading from AT24164 EEPROM memory

* PDC8544 is built upon a SPI driver, whilst 24AA_EEPROM is built upon a I2C one.

6 Issue with I2C ACK Polling

As described in AT24C164 2-Wire Serial EEPROM 16 kbits (2048 x 8 bits) datasheet [6]:

"ACKNOWLEDGE POLLING: Once the internally-timed write cycle has started and the EEPROM inputs are disabled, acknowledge polling can be initiated. This involves sending a start condition followed by the device address word. The read/write bit is representative of the operation desired. Only if the internal write cycle has completed will the EEPROM respond with a zero allowing the read or write sequence to continue."

I tried to use the this feature but my program was getting stuck on ACK Polling (AT24C164.c#333). I tried to check what was happening on I2C bus using an oscilloscope.

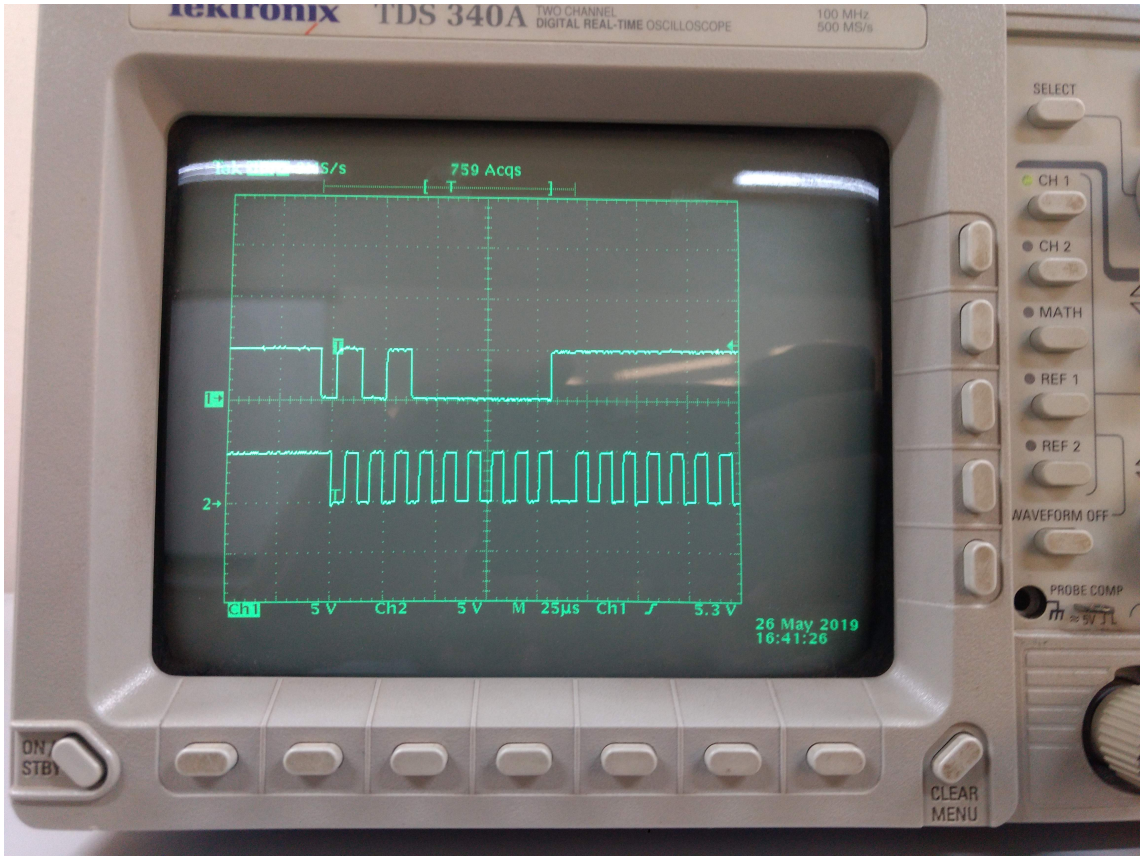

Figure 1: Photo of Processor Expert ACK Polling. Channel 1: SDA, Channel 2: SCL

After the start condition, it's possible to see that the bits sent are: 10100000. So, according to [6], $A_2A_1A_0 = 010$, $B_2B_1B_0 = 000$, but the ATC24C164 configured I2C address is actually 000 (and not 010).

In order to bypass this I put a delay after the page writing. I didn't have time to investigate it any further.

7 Validation

In order to validate the integration of all features I tested the project with a bunch of different use cases and also let it running for relative long times, like 10 minutes or above.

References

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

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

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

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

[5] ftp://ftp.dca.fee.unicamp.br/pub/docs/ea076/datasheet/Nokia5110.pdf.

[6] ftp://ftp.dca.fee.unicamp.br/pub/docs/ea076/datasheet/AT24C164.pdf.

[7] https://mcuoneclipse.com/2012/12/16/zero-cost-84x48-graphical-lcd-for-the-freedo

[8] http://www.dca.fee.unicamp.br/courses/EA076/1s2019/roteiros/ROTEIRO3.pdf.

[9] https://www.embeddedrelated.com/showarticle/519.php.