

*Desenvolvimento de uma estação  
meteorológica em bare-metal e RTOS  
com um dashboard em Qt.*

Trabalho prático de Sistemas Embebidos e de Tempo Real

Rúben Guimarães nº11156

Kyrylo Yavorenko nº10355

*Escola Superior de Tecnologia, IPCA  
Barcelos*

11 de Junho de 2018

# Conteúdo

<b>Introdução</b>	<b>2</b>
<b>Resumo</b>	<b>3</b>
<b>Objetivos</b>	<b>4</b>
<b>Parte I</b>	<b>5</b>
Arquitetura . . . . .	5
Recursos usados no projeto . . . . .	6
Esquema do projeto . . . . .	8
Desenvolvimento . . . . .	10
<b>Parte II</b>	<b>15</b>
Arquitetura . . . . .	15
Desenvolvimento . . . . .	16
<b>Conclusão</b>	<b>18</b>
<b>Bibliografia</b>	<b>20</b>

# Introdução

O trabalho prático abordado neste relatório foi desenvolvido no âmbito da unidade curricular Sistemas Embebidos e de Tempo Real do curso de Engenharia de Sistemas Informáticos, lecionada pelo docente António Moreira. Inicialmente o docente desafiou os alunos a aplicarem os conceitos de programação de sistemas embebidos e de tempo real lecionados durante o decorrer da unidade curricular, surgindo assim este projeto, dividido em duas partes, (baremetal e usando o sistema operativo FreeRTOS), que adquire de diversas fontes de sinais analógicos e digitais informação replicando o funcionamento de uma estação meteorológica.

Numa segunda fase pediu ao alunos que pegassem no projeto já desenvolvido e enviassem os dados por uma porta serial para o computador. Criassem um *dashboard* em Qt para mostrar os dados adquiridos no projeto inicial.

# Resumo

Neste trabalho desenvolvemos uma pequena estação meteorológica recorrendo à plataforma de prototipagem eletrónica open-source Arduino. Esta estação consiste num conjunto de sensores que obtém dados sobre o estado do tempo que depois são enviados para o Arduino para serem processados, sendo por fim mostrados ao utilizador, quer através de um lcd de 16x2, quer através da consola do IDE do Arduino.

Na segunda parte criamos um *dashboard* recorrendo a C++ e QML que apresenta-se os dados de forma minimalista.

# Objetivos

Os objetivos definidos para o projeto pelo docente foram:

- Desenvolvimento de um programa usando a tipologia baremetal.
- Desenvolvimento de um programa usando o sistema operativo FreeRTOS usando 3 tasks.
- Utilizar os seguintes sensores:
  - Sensor de água.
  - 3 LDR's para calcular a posição do sol.
  - Sensor de humidade.
  - Dois sensores de temperatura (para o ar e o solo).
  - Barómetro.
  - Anemômetro
- Criação de um *dashboard* em Qt.

# Parte I

## Arquitetura

Podemos consultar na figura seguinte um diagrama com a arquitetura do projeto.

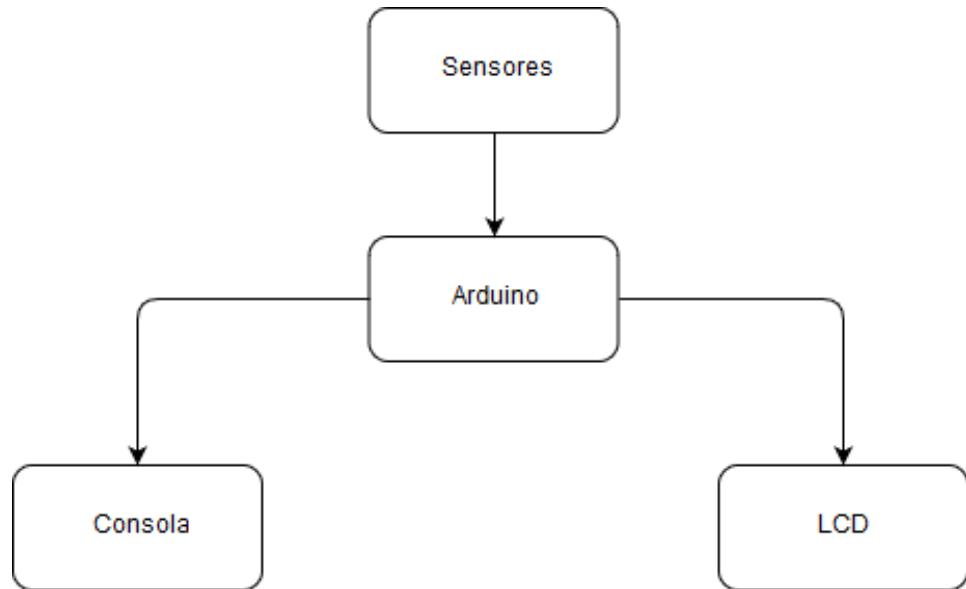


Figura 1: Diagrama da arquitetura do projeto

## Recursos usados no projeto

Para o desenvolvimento do projeto foram utilizados os seguintes recursos:

### Software:

- *Arduino IDE 18.5* para o desenvolvimento do código usado.
- *GitHub Desktop* para atualizar o repositório com o código do projeto.
- *Fritzing 0.9.3b* para criar os protótipos dos esquema do projeto.
- *LATEX*

### Hardware:

- *Arduino Mega 2560*, placa que contem o micro-controlador que controla todo o nosso projeto.
- *DHT11*, sensor usado para obter leituras de umidade e temperatura do ambiente.
- *DS18B20*, sensor à prova de água usado para obter temperaturas de água.
- *BMP180*, sensor que obtém leituras da pressão atmosférica e da temperatura ambiente. Com os valores obtidos conseguimos fazer uma estimativa da altitude.
- *Water Sensor*, sensor usado para saber se existe água ou não.
- *Buzzer*, usado para dar um feedback sonoro caso seja detetada água no sensor de água.
- *QTR-8RC*, sensor usado para contar as rotações por minuto de um cata-vento.

- *LCD 16x2*, usado para mostrar os valores obtidos pelos sensores.
- *Light Dependent Resistor*, usado para descobrir qual das posições (sul, este ou oeste) o sol de encontra.
- *Breadboard*, usada para montar o circuito.
- *Diversas resistências*, usadas para proteger componentes ou para servir de pull-up.
- *Diversos fios*, usados para efetuar as ligações entre os componentes, breadboard e o arduino.

## Esquema do projeto

Na imagem seguinte podemos verificar o esquema do projeto desenvolvido na plataforma fritzing.

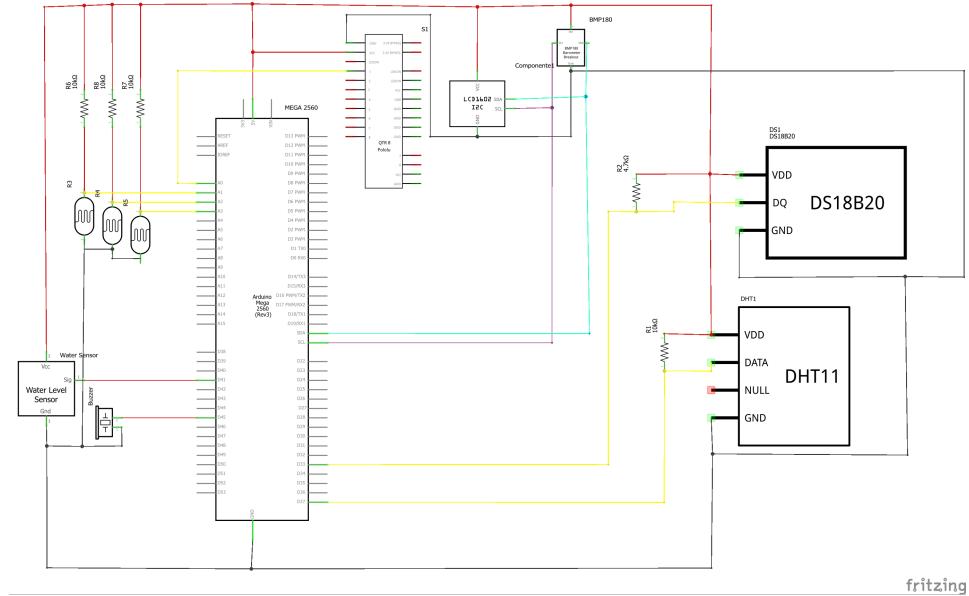


Figura 2: Esquema do projeto

Na imagem seguinte podemos ver o projeto finalizado.

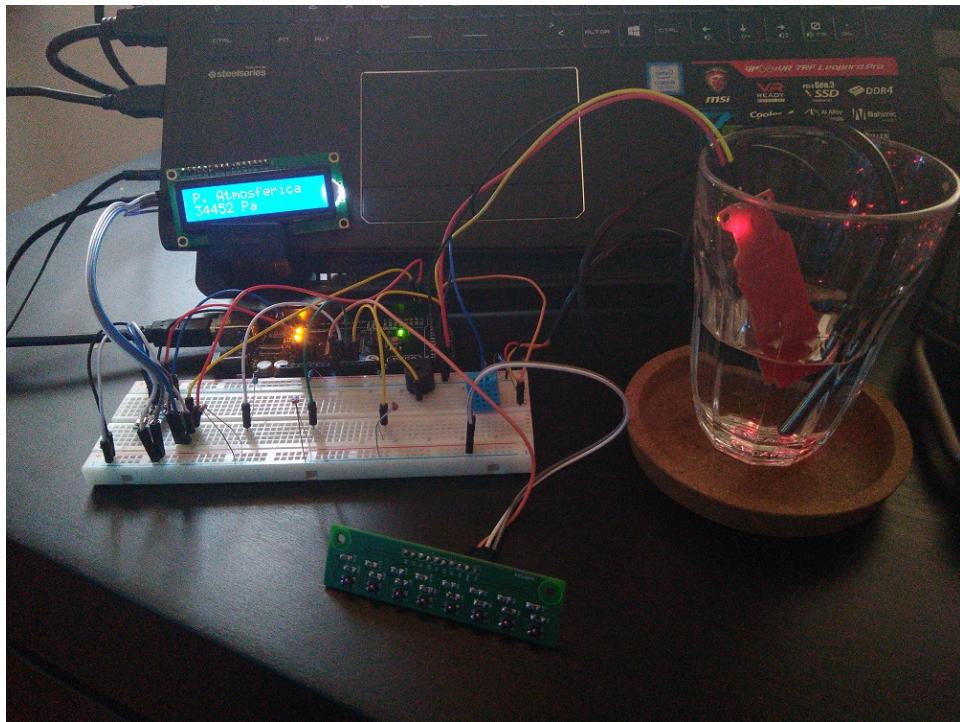


Figura 3: Foto do projeto finalizado.

## Desenvolvimento

Inicialmente decidimos que a nossa abordagem ao projeto seria começar por bare-metal, testar todos os sensores e as ligações nesta metodologia de código e só depois passar à parte em que iríamos recorrer ao sistema operativo FreeRTOS.

Começamos por testar o sensor de temperatura DS18B20 recorrendo à biblioteca OneWire.h [1] para facilitar a leitura dos dados dos sensores. [2] [3]. De seguida, passamos ao sensor DHT11 que nos permite medir a percentagem de humidade do ambiente e a temperatura, para o qual adicionamos a biblioteca DHT.h [4]. [5]. Depois passamos ao barómetro BMP180, este sensor permite-nos medir a pressão atmosférica, a temperatura e com a pressão atmosférica podemos calcular aproximadamente a altitude que encontra. Para obter os valores recorremos à biblioteca Wire.h (nativa do arduino) e a Adafruit\_BMP085.h [6]. [7]. No final do circuito destes sensores montados e o código escrito, imprimimos o valor deste na consola para verificar se estava tudo a funcionar corretamente.

O proximo passo foi testar o sensor de detecção de água, este foi bastante simples tendo em conta que devolve um sinal HIGH (5V) caso seja encontrada água nos contactos do sensor. [8]. Depois montamos os 3 LDR's que servem para determinar a posição do sol, sendo estes também simples de instalar tendo em conta que já tínhamos experiência por termos feito circuitos com a utilização de LDR's no decorrer da cadeira. Só tivemos que os ligar e determinar no código qual dos LDR's estava a obter um valor de resistência mais baixo.

Por fim, faltava utilização de LED's infravermelhos para ler as rotações de um cata-vento, primeiro testamos utilizar simples LED's IR e contar as vezes que este era interrompido num certo período de tempo. Não obtendo sucesso recorremos ao sensor QTR-8RC mas utilizamos apenas um dos seus conjuntos de LED's. Para obter a leitura das Rotações por minuto fizemos uma função que durante 5 segundos consulta o valor obtido na porta analógica do LED receptor de 50 em 50 milissegundos. Caso este valor seja inferior ao do threshold definido (800) conta uma volta. Também tivemos que definir uma flag para evitar as mesmas leitura (caso o cata-vento estivesse parado). Para calcular um valor das rotações por minuto efetuamos uma regra de 3 simples para efetuar uma estimativa das rotações num minuto.

Como podemos verificar na imagem seguinte, estávamos a obter os valores dos sensores todos na consola.

```
*** Valores Lidos ***
Humidade ambiente: 30%
Temperatura ambiente: 19°C
Temperatura água: 17.00°C
Temperatura BMP: 17.60°C
Pressão atmosferica: 34460Pa
Altitude: 111m
Água no sensor ? Não!
Velocidade vento: 36 RPM
Posição do sol: Sul
```

Figura 4: Valores obtidos na consola do IDE.

O único aspecto que nos faltava para terminar a implementação em bare metal era mostrar os valores obtidos num LCD 16x2. Para facilitar a utilização do LCD recorremos a um modulo I2C que permite simplicar a ligação ao arduino usando as portas de comunicação SDA e SCL. Para utilizar o LCD recorremos a biblioteca LiquidCrystal\_I2C.h [9]. Podemos verificar na imagem seguinte o LCD a mostrar valores obtidos.



Figura 5: LCD a mostrar o valor da altitude.

Ao desenvolver o código, até ao momento, tivemos o cuidado de poupar o máximo de memória possível, até porque sabíamos que na implementação em FreeRTOS iríamos necessitar de toda a memória disponível. [10]

O passo seguinte e final foi migrar o código desenvolvido até ao momento para utilizar o sistema operativo FreeRTOS. Para utilizar este tivemos que utilizar a biblioteca Arduino\_FreeRTOS.h [11] e depois criamos 3 tarefas diferentes:

- *GetSensorValuesTask* usada para obter os valores dos sensores.
- *ShowValuesLCDTask* usada para mostrar os valores obtidos no LCD.
- *PrintValuesOnConsoleTask* usada para mostrar os valores obtidos na consola do IDE.

Na imagem seguinte podemos verificar a criação das tarefas com os seus parâmetros únicos, tanto de tamanho, como de prioridade.

```
xTaskCreate(
    ShowValuesLCDTask
,  (const portCHAR *)"ShowValuesLCDTask"
,  512
,  NULL
,  2
,  NULL );

xTaskCreate(
    GetSensorValuesTask
,  (const portCHAR *)"GetSensorValuesTask"
,  512
,  NULL
,  3
,  NULL );

xTaskCreate(
    PrintValuesOnConsoleTask
,  (const portCHAR *)"PrintValuesOnConsoleTask"
,  512
,  NULL
,  1
,  NULL );
```

Figura 6: Criação das tarefas.

Cada uma das tarefas reutiliza o mesmo código desenvolvido no bare metal, apenas às vezes com algumas alterações, como por exemplo, evitar a criação de variáveis dentro do ciclo infinito da tarefa para impedir ultrapassar o tamanho da stack e substituição dos delay's pelos vTaskDelay's. Um dos problemas que nos surgiu nesta fase foi a escolha do tamanho da stack de cada tarefa. Para alcançarmos o tamanho ideal começamos por definir um valor alto para cada tarefa (ex: 1048 bits) e ir reduzindo até a tarefa deixar de funcionar. Chegamos ao valor mínimo de 512 bits para cada stack das tarefas.

# Parte II

## Arquitetura

Podemos consultar nas figuras os diagramas com a arquitetura do projeto da parte II.

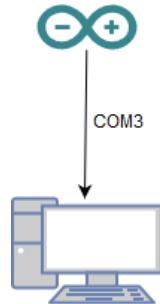


Figura 7: Arquitetura do hardware.

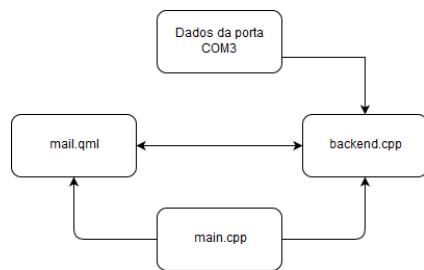


Figura 8: Arquitetura do software.

## Desenvolvimento

Começamos por criar no Qt Creator um novo projeto do tipo Qt Quick Application - Empty. Neste criamos um header "backend.h" e a implementação deste backend.cpp. Nos resources criamos o ficheiro main.qml que vai conter o código qml que personaliza a nossa aplicação e uma pasta para guardarmos os recursos necessários para a aplicação.

De seguida fizemos a ligação no ficheiro main.cpp do *backend* e do *frontend* do projeto carregando o engine do Qt. Neste passo tivemos que alterar fazer pequenas alterações no código do arduino para enviar uma string de dados formatada para a porta COM3. No backend começamos por configurar a comunicação com a porta COM3 tal como podemos verificar no excerto de código seguinte.

```
// Configura a porta COM3
this->mCOM = new QSerialPort();
this->mCOM->setBaudRate(QSerialPort::Baud9600);
this->mCOM->setStopBits(QSerialPort::OneStop);
this->mCOM->setFlowControl(QSerialPort::NoFlowControl);
this->mCOM->setParity(QSerialPort::NoParity);
this->mCOM->setDataBits(QSerialPort::Data8);
this->mCOM->setPortName("COM3");
this->mCOM->open(QSerialPort::ReadWrite);

connect(mCOM, SIGNAL(readyRead()), this, SLOT(dataReceived()));
```

Figura 9: Configuração da comunicação.

De seguida criamos no main.qml as diversas estruturas necessárias para atualizar ou modificar o *dashboard* com os dados recebidos. Depois desta parte concluída e configurada recorrendo a linguagem qml. Criamos diversas variáveis do tipo QObject no backend.cpp e fizemos a ligação destas ao's

objectName's correspondentes para permitir atualizar os dados do dashboard. Podemos verificar o resultado final do *dashboard* preenchido com dados reais na imagem seguinte.

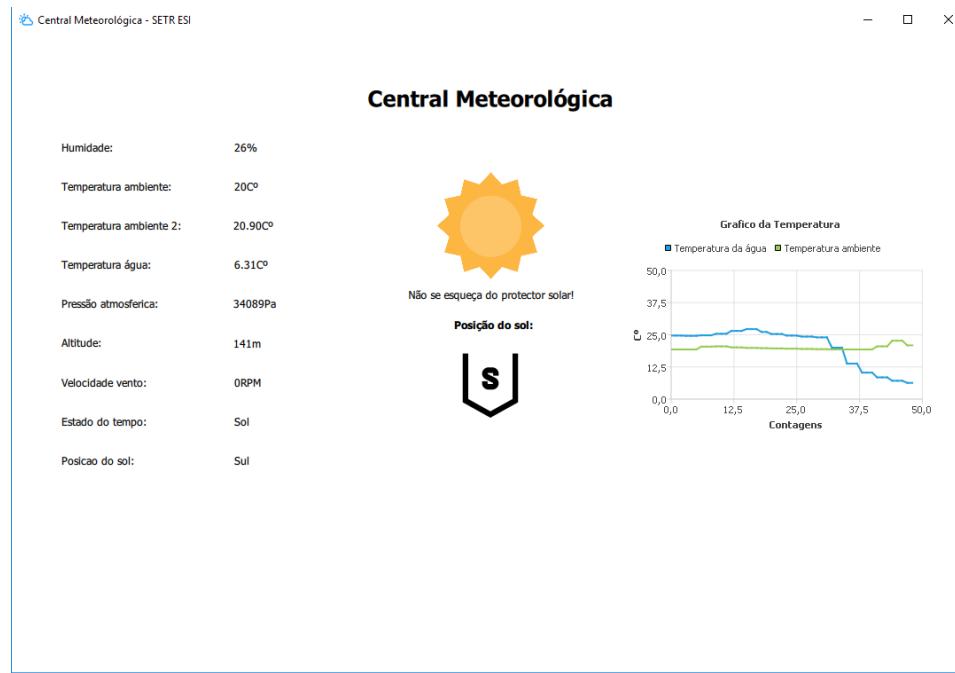


Figura 10: Configuração da comunicação.

# Conclusão

Este trabalho permitiu-se aplicar os conhecimentos adquiridos durante o decorrer da unidade curricular de Sistemas Embebidos e de Tempo Real. Acha-mos que o trabalho foi muito proveitoso porque tivemos pela primeira vez no curso um contacto com programação de mais baixo nível e onde trabalhássemos diretamente com o hardware. Foi muito interessante ter que ter cuidados com o tamanho das variáveis, tendo em conta que estávamos a trabalhar num ambiente onde a memória era muito escassa.

A parte de desenvolvimento da *dashboard* foi de tal forma vantajoso tendo em conta que tivemos a oportunidade de trabalhar com uma ferramenta usada por grandes empresas da área. As principais dificuldades nesta parte foi por vezes a comunicação entre o C++ e o qml que nos criou bastantes constrangimentos.

De forma geral achamos que o trabalho correu suavemente, tendo em conta que ultrapassamos todas as dificuldades encontradas e cumprimos todos os objetivos definidos pelo professor.

# Bibliografia

- [1] PaulStoffregen. *OneWire.h*. 06 Abril, 2018. <https://github.com/PaulStoffregen/OneWire>
- [2] dfrobot. *Waterproof DS18B20 Digital Temperature Sensor*. 06 Abril, 2018. <https://github.com/PaulStoffregen/OneWire>
- [3] sparkfun. *DS18B20*. 06 Abril, 2018. [https://github.com/sparkfun/simple\\_sketches/blob/master/DS18B20/DS18B20.ino](https://github.com/sparkfun/simple_sketches/blob/master/DS18B20/DS18B20.ino)
- [4] Adafruit. *Adafruit Unified Sensor Driver*. 06 Abril, 2018. [https://github.com/adafruit/Adafruit\\_Sensor](https://github.com/adafruit/Adafruit_Sensor)
- [5] lady ada. *Connecting to a DHTxx Sensor*. 06 Abril, 2018. <https://learn.adafruit.com/dht/connecting-to-a-dhtxx-sensor>
- [6] Adafruit. *A powerful but easy to use BMP085/BMP180 Library*. 06 Abril, 2018. <https://github.com/adafruit/Adafruit-BMP085-Library>
- [7] Adilson Thomsen. *Controlando temperatura e pressão com o BMP180*. 06 Abril, 2018. <https://www.filipeflop.com/blog/temperatura-pressao-bmp180-arduino/>

- [8] tutorialspoint. *Arduino - Water Detector / Sensor*. 20 Abril, 2018. [https://www.tutorialspoint.com/arduino/arduino\\_water\\_detector\\_sensor.htm](https://www.tutorialspoint.com/arduino/arduino_water_detector_sensor.htm)
- [9] fdebrabander. *Library for the LiquidCrystal LCD display connected to an Arduino board*. 20 Abril, 2018. <https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library>
- [10] Bill Earl. *Optimizing SRAM*. 30 Abril, 2018. <https://learn.adafruit.com/memories-of-an-arduino/optimizing-sram>
- [11] feilipu. *A FreeRTOS Library for all Arduino AVR Devices (Uno, Leonardo, Mega, etc)* . 06 Abril, 2018. [https://github.com/feilipu/Arduino\\_FreeRTOS\\_Library](https://github.com/feilipu/Arduino_FreeRTOS_Library)