07_2018

[Sistema de monitorização de Estações de Tratamento de Águas Residuais (ETARs)]

[Integração de Sistemas]

Manuel Lameira – nº 4829 |
Alberto Lameira – nº 13282

MESTRADO "INTERNET DAS COISAS" – IP BEJA – 2017/2018

Índice

1. Introdução

A proposta de criação do Mestrado em Internet das Coisas (doravante designado por MIOT) apresenta, entre outras justificações, o facto do Instituto Politécnico de Beja integrar a infraestrutura de investigação científica Engage-SKA. Neste contexto há um tema comum identificado em reuniões com membros do projeto Engage-SKA pertencentes à Escola Superior Agrária do IPBeja: **a água** [1].

No enquadramento dos trabalhos já realizados, nomeadamente nas disciplinas de Dispositivos para a Internet das Coisas e Sistemas Embebidos, onde se escolheu o Sistema de monitorização de Estações de Tratamento de Águas Residuais (ETARs), e é sobre estes que o presente relatório assenta. Descreve-se a metodologia da execução, assim como a sua implementação e as alterações sofridas.

Foram atualizados alguns aspetos do relatório face ao anterior, nomeadamente a adição das figuras no capítulo 3 e a substituição do capitulo 5, 6 e 7.

2. Estado da Arte

O único sistema semelhante que foi possível encontrar foi o sistema de medidor de caudal ultrassónico da empresa ecodepur [2]. A empresa faz questão de omitir diversos detalhes sobre o sistema, como a arquitetura do sistema, que tipos de sensores usam no total e como é feita a comunicação entre o equipamento e a base. No entanto, pelo que é possível analisar parte-se do princípio que este sistema é bastante semelhante ao que é idealizado neste projeto, uma vez que a forma como está desenhado o modelo indica-nos que é utilizado um sensor ultrassónico a uma distância fixa e conhecida acima da superfície da água, podendo assim ser medido o volume de água a passar pelo afluente.

Este sistema opera em uma gama de temperatura de -40ºC a +90ºC com uma frequência de 41.5KHz. É possível ser programado através de um teclado de 5 botões e contém um display de 2 linhas de 16 dígitos LCD. Funciona ainda com alimentação 230V AC.

3. Análise do Problema

O objetivo do trabalho **sistema para a monitoração de incidentes em estações de tratamento de águas residuais com georreferenciação da informação** assenta numa plataforma de *hardware* e *software* composta por microcontrolador, sistemas de comunicação de dados, a sua programação, sensores, atuadores e a eletrónica de aquisição e condicionamento de sinal.

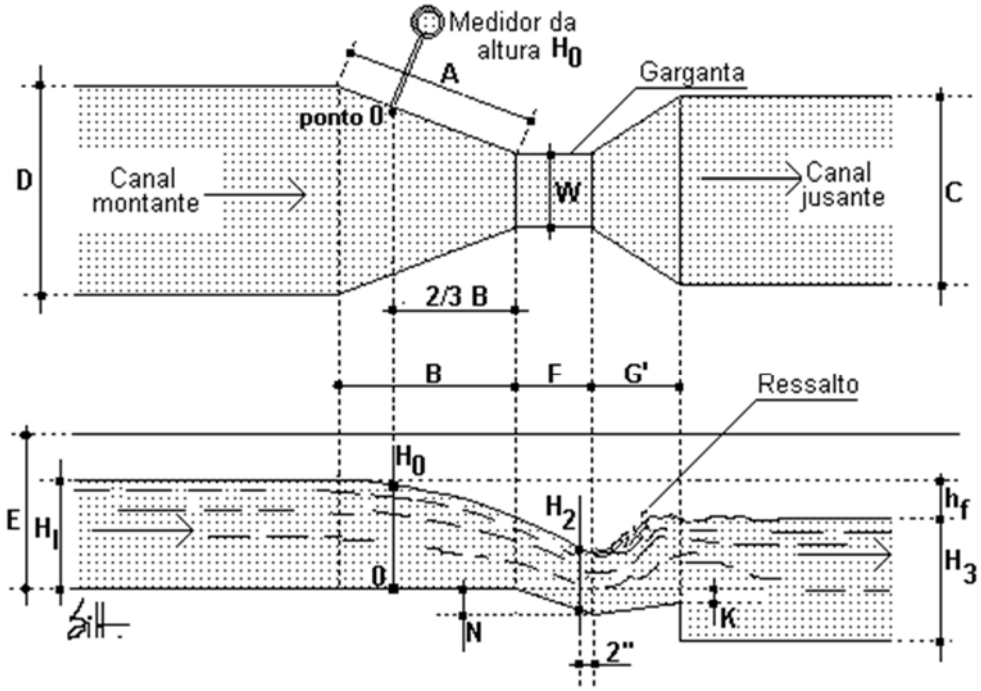
Este conjunto de equipamento será montado na obra de entrada, mais precisamente no canal Parshall. Deste modo é permitido um controlo contínuo do afluente para a fossa, em situação de normal funcionamento. Em caso excecional de descarga direta através do *By-Pass* (situação de incidente) para a linha de água, é acionado um sensor de contacto localizado na comporta do *By-Pass*, possibilitando quantificar com precisão o caudal, volume e tempo de descarga direta. Para este trabalho iremos focar-nos apenas na situação de incidente, ou seja, descargas diretas para a linha de água.

O sistema permite um maior controlo ambiental uma vez que é autónomo e de comunicação instantânea.

4. Arquitetura do Sistema

O sistema será montado na obra de entrada, mais precisamente no canal Parshall.

O canal Parshall é um dispositivo de medição de caudal na forma de um canal aberto com dimensões padronizados. A água é forçada por uma garganta (W) relativamente estreita, sendo que o nível da água a montante da garganta é o indicativo do caudal a ser medido, independentemente do nível de água a jusante. A Tabela 1 [3] mostra os valores padronizados da largura da garganta do canal de Parshall, bem como de outras dimensões do mesmo.



ESQUEMA DE UMA CALHA PARSHALL CONVENCIONAL

Tabela 1 - Dimensões padronizadas do canal Parshall (mm)

W	A	B	C	D	E	F	G'	K	N
76 (3")	466	457	178	259	381	152	305	25	57
152 (6")	621	610	294	393	457	305	610	76	114
229 (9")	880	864	380	575	610	305	457	76	114
305 (1')	1370	1340	601	845	915	610	915	76	229
457 (1 1/2)	1449	1420	762	1026	915	610	915	76	229
610 (2')	1525	1496	915	1207	915	610	915	76	229
915 (3')	1677	1645	1220	1572	915	610	915	76	229
1220 (4')	1830	1795	1525	1938	915	610	915	76	229
1525 (5')	1983	1941	1830	2303	915	610	915	76	229
Fonte: (Azvedo Netto et alli, 1998)									

A base horizontal do canal constitui um nível de referência para o nível de água a montante. Muitas vezes mede-se a altura da água num ponto situado a 2/3 do canal de aproximação da garganta, tendo-se estabelecido empiricamente a seguinte relação entre o nível de água no ponto 0 e o caudal na seção:

Q = 2,2 * W * H0^(3/2), (Q em m3/s)

Onde:

H0 = altura do nível de água no ponto 0 (m)

W = largura da garganta (m)

Pelo exposto, apenas necessitamos de medir a altura do afluente no ponto H0 para calcular o caudal. O sensor de temperatura e o de partículas em suspensão serão montados a jusante do canal de Parshall.

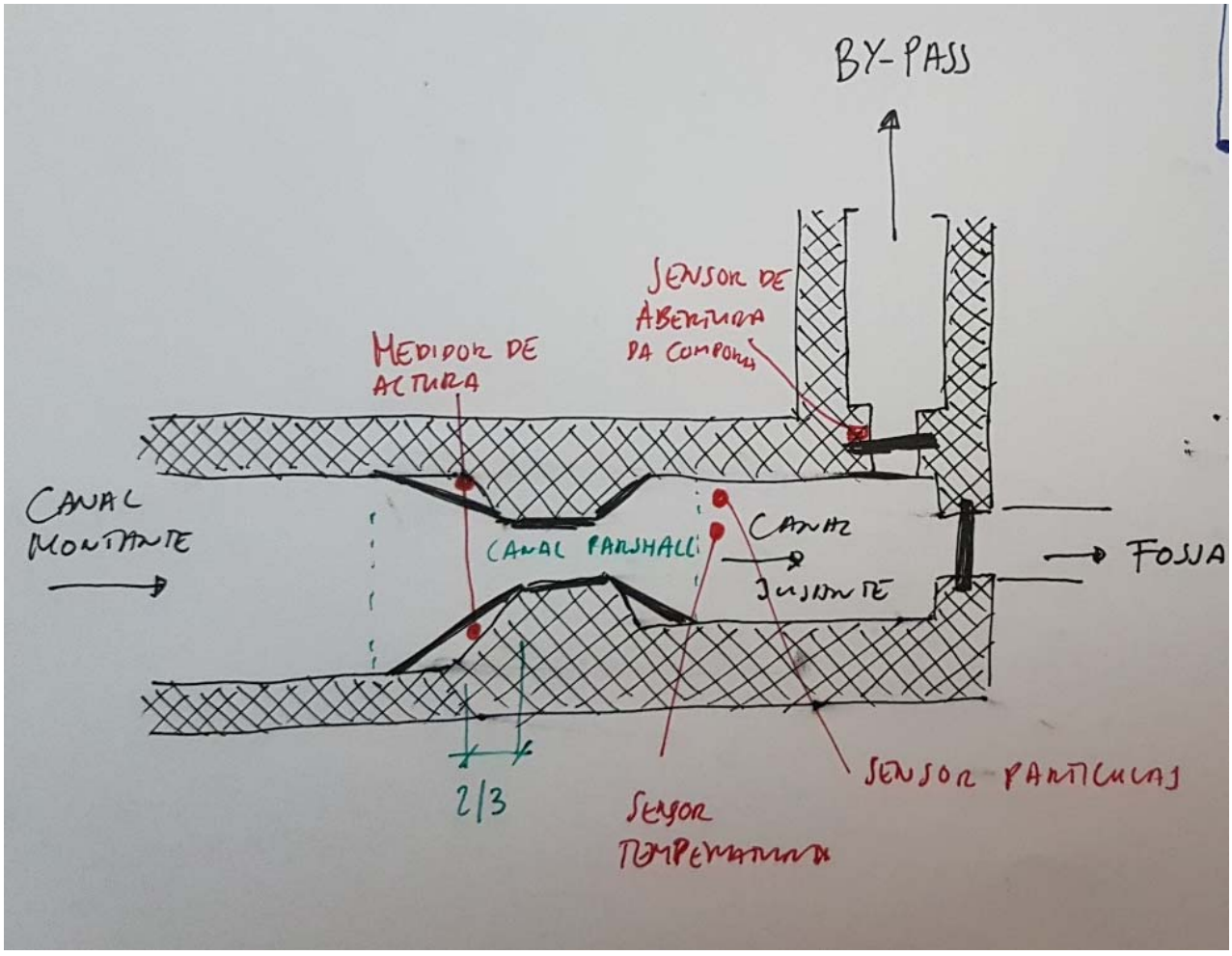


Figura 1 - Esquema de montagem dos equipamentos na Obra de Entrada

Neste caso particular, a obra de entrada em estudo apresenta um bypass adicional, que serve como "ladrão" em caso de excesso de afluente, protegendo o bom funcionamento da fossa.

Esta particularidade, obriga a uma distinção na apresentação das leituras de distância na estação recetora, de modo a diferenciar o volume de afluente escoado, ora pelo funcionamento normal, pelo bypass “ladrão” ou pela abertura da comporta, conforme ilustrado nas seguintes imagens.

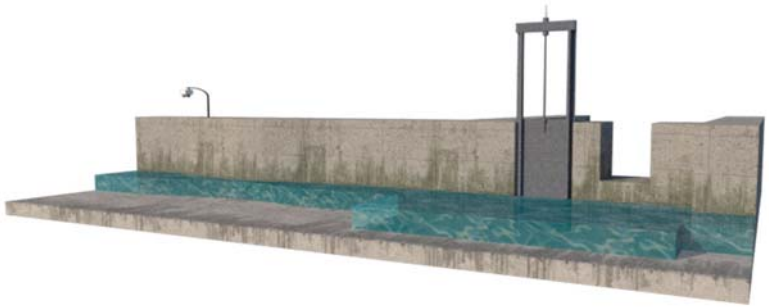


Figura 2 - Hipótese 1 - Funcionamento normal da obra de entrada

Figura 3 - Hipótese 2 - Funcionamento pelo bypass "ladrão"

Figura 4 - Hipótese 3 - Funcionamento pela abertura da comporta do bypass

4.1 Hardware

Neste subcapítulo expomos o hardware utilizado para este projeto. O equipamento físico é dividido em dois módulos, o que fica montado na obra de entrada na ETAR a efetuar a recolha e envio dos dados (emissor) e o segundo que recebe e trata os dados (recetor).

Apresenta-se também de seguida os diagramas de blocos dos dois módulos com as respetivas ligações entre os seus componentes e descreve-se as suas constituintes nos pontos seguintes. Tentou-se, sempre que possível, encontrar equipamentos com uma ótima qualidade preço para englobar no projeto de modo a manter os custos o mais baixo possível.

Como referido no artigo anterior, a estação emissora será instalada na obra de entrada, no entanto de modo a economizar energia das baterias o sistema só é ativado quando a comporta do *By-Pass* é acionada.

Para que o sistema funcione, idealizou-se um *switch* mecânico que opere por contacto.

A estação recetora será ligada a um computador (PC ou Raspberry Pi) para tratamento de dados e fornecer uma interpretação gráfica dos mesmos.

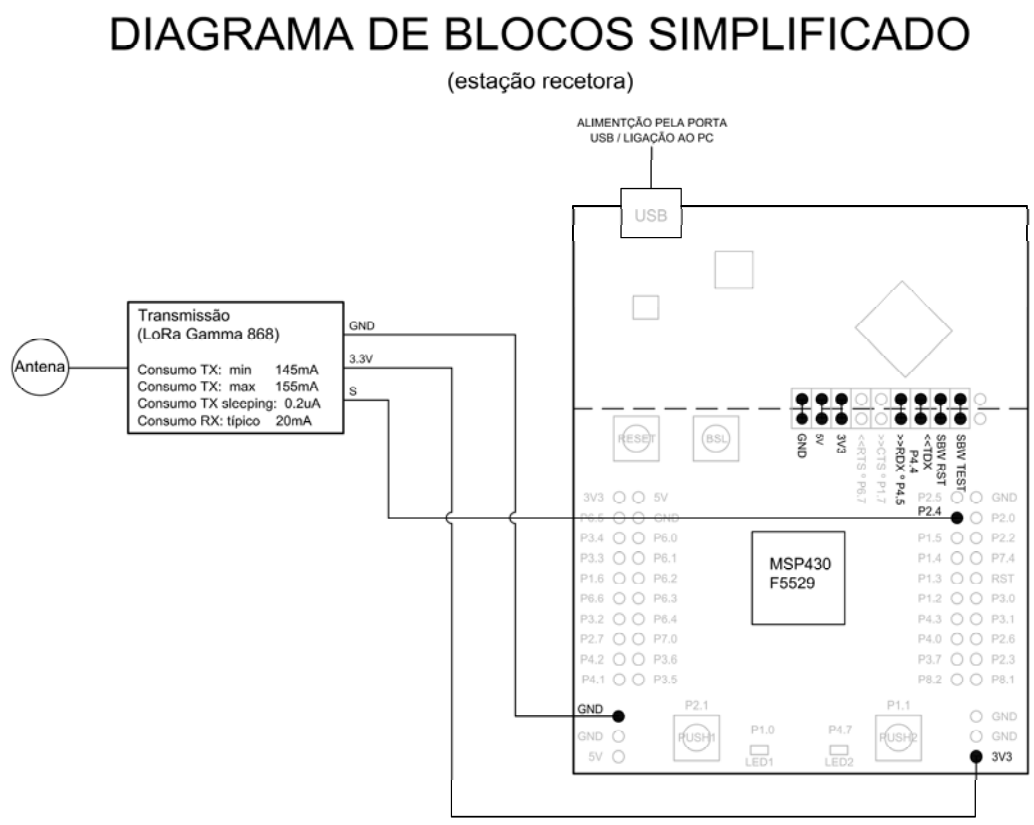


Figura 5 - Diagrama de Blocos da estação recetora

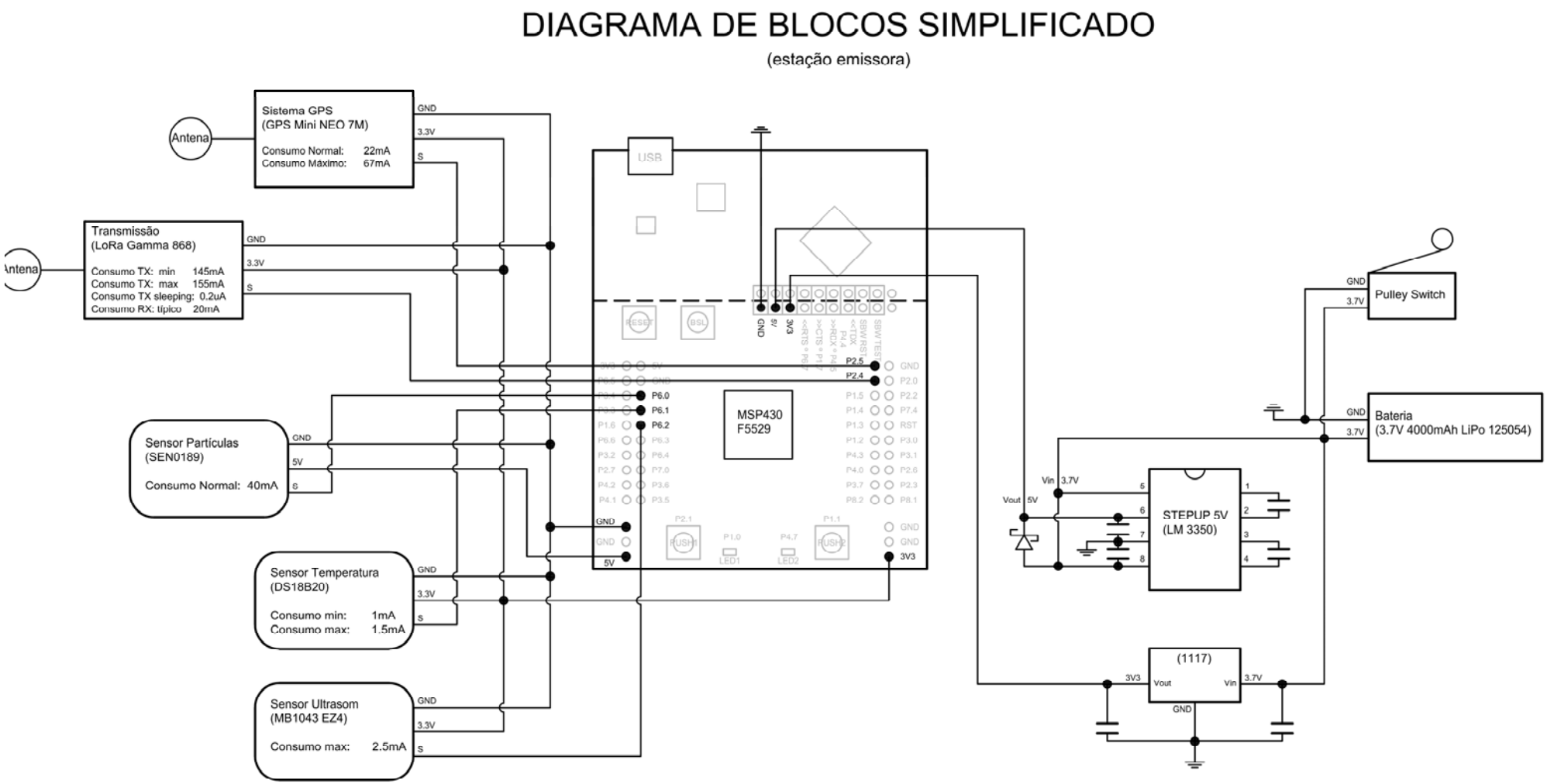


Figura 6 - Diagrama de Blocos da estação emissora

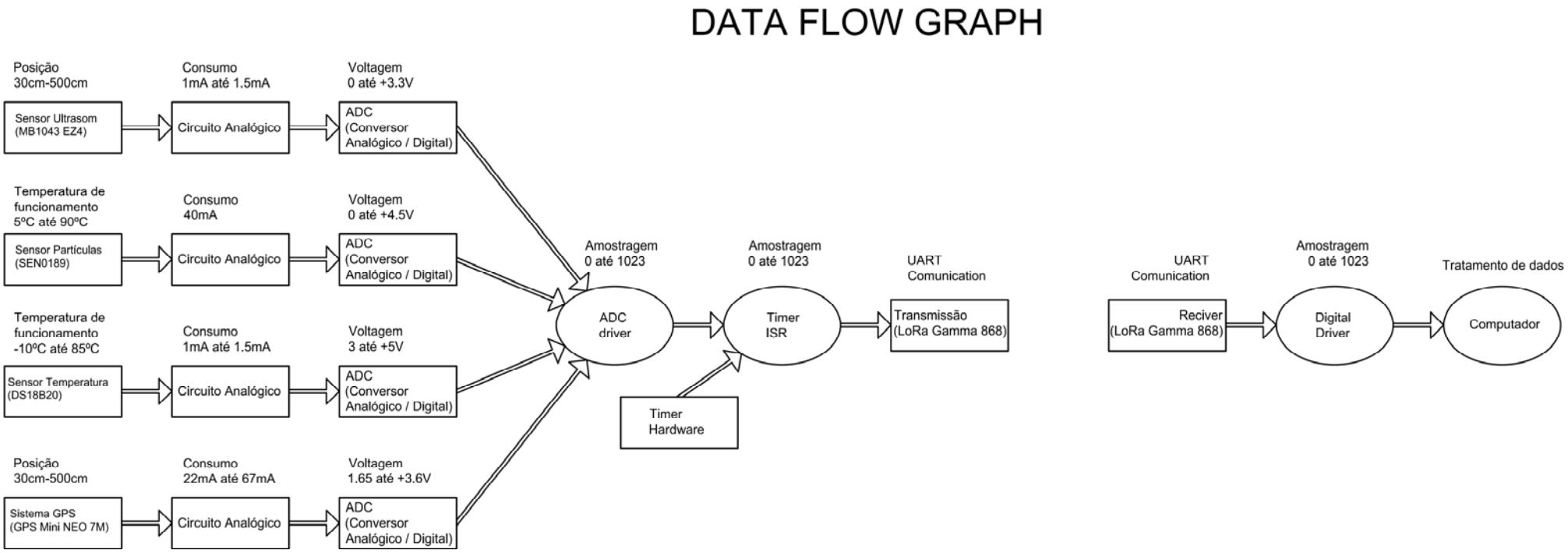


Figura 7 - Data Flow Graph

- Os Retângulos representam componentes de hardware e os ovais módulos de software.
- As setas apontam da fonte da informação para o destino

4.1.1 Sensores

Medidor de temperatura [4] – A função do medidor de temperatura é como o nome indica, medir a temperatura do afluente. Propomos estes sensores de temperatura:

Tabela 2 - Sensor Temperatura

EQUIPAMENTO	MARCA/LOCAL	COMPATIVEL	VOLTAGEM DE FUNCIONAMENTO	AMPER DE FUNCIONAMENTO	AMPLITUDE DE MEDIÇÃO	PRECISÃO	A PROVA DE AGUA	PREÇO
DS18B20	Chines / eBay	Arduino	3,2 ~ 5,25VDC	2mA (max)	-55 ~ 110 ºC	±0.5ºC @-10~+80'C	sim	2,89 €
DANFOSS MBT 153	DANFOSS	-	sob consulta	sob consulta	-50~200 ºC	+/- 1°C	sim	sob consulta
ALTA MNS2-9-IN-TS-WT-L03	MONNIT	-	3,6V	-	-40°C to +100°C	+/- 1°C	sim	sob consulta

Ir-se-á utilizar o **sensor DS18B20**, por questões de logística e aquisição.

Medidor de altura de água [5] – Para o medidor de altura utilizaremos um medidor ultrassónico. Após uma pesquisa de mercado encontrou-se os seguintes:

Tabela 3 - Sensor Altura de Água

EQUIPAMENTO	MARCA/LOCAL	VOLTAGEM DE FUNCIONAMENTO	AMPER DE FUNCIONAMENTO	DISTANCIA DE MEDIÇÃO	PRECISÃO	A PROVA DE AGUA	PREÇO
HC-SR04 ULTRASOM	chines ebay	5V	2mA	2cm ate 450cm	Até 0,3cm	Não	0,83 €
JSN SR04T ULTRASOM	chines ebay	5V	30mA	25cm ate 4500m	0,5cm	sim	6,69 €
VDM18-300/20/88/122/151 LASER	Pepperl+Fuchs	30V		80mm ate 300mm	0,10%	sim	sob consulta
ULTRASOM MB1043 EZ4	Maxbotix	2.5 – 5.5 V	2.5mA	30cm ate 500cm	1mm	não	28,10 €

Por questões de logística e aquisição, optou-se pelo sensor **Ultrasom MB1043 EZ4**.

Sensor de partículas em suspensão [6] – Como sensor de partículas em suspensão iremos utilizar um sensor de turbidez (encontrado nas maquinas de lavar loiça e maquinas de lavar roupa).

Tabela 4 - Sensor Partículas em Suspensão e Turbidez

EQUIPAMENTO	MARCA/LOCAL	VOLTAGEM DE FUNCIONAMENTO	AMPER DE FUNCIONAMENTO	INSULATION RESISTANCE	PREÇO
SEN0189	chines ebay	5V	40mA	100 M	15,10 €

Não existe muita escolha no sensor de partículas, dos encontrados, todos se baseiam na base do **SEN0189**.

Sensor de contacto [7] – Para diferenciar se o afluente esta encaminhado pelo *By-Pass* ou diretamente para a fossa, instalamos um sensor de contacto na comporta do *By-Pass* para sinalizar ao microcontrolador a utilização da via alternativa. O microcontrolador indica nas leituras de caudal que o sensor foi adicionado (por exemplo colocando um caractere "B" após a s leituras) e simultaneamente contabiliza o tempo em que o sensor esta acionado, permitindo calcular o fluxo de afluente descarregado pelo *By-Pass*.

Tabela 5 - Sensor Pulley Switch

EQUIPAMENTO	MARCA/LOCAL	VOLTAGEM DE FUNCIONAMENTO	AMPER DE FUNCIONAMENTO	PREÇO
CONJUNTO DE SWITCH MAGNÉTICO PARA PORTAS	Boxelectronica	ate 100V	ate 500mA	5,89 €
PULLEY SWITCH	Funcionamento Mecânico			

Após uma análise mais cuidada, concluiu-se que o sensor proposto não é funcional. Para o estudo em causa, apenas interessa o afluente que é encaminhado para o *By-Pass*. Utilizando o sensor inicialmente proposto, existe a necessidade de este estar constantemente ligado e consumindo energia valiosa das baterias. Optou-se por colocar um sensor de contacto do tipo ***Pulley Switch*** na comporta do *By-Pass*. Quando a comporta é aberta, o sensor abre o fornecimento de energia ao microcontrolador e consequentemente inicia o sistema.

4.1.2 Plataforma Computacional

Microcontrolador MSP430 [8] – De modo a monitorizar os vários componentes do sistema utiliza-se um microcontrolador MSP430 da Texas Instrumentes. Este microcontrolador permite, entre outras funções, a otimização do consumo de energia *(Low Power Mode)*. Microcontroladores adquiridos e disponíveis:

Tabela 6 - Microcontroladores

EQUIPAMENTO	QUANTIDADE	BIT	RAM	TIMERS	PINS
MSP430G2553	2	16	512 B	2	20
MSP430F5529	2	16	128 KB	4	40
MSP430FR5969	1	16	64 KB	5	20
MSP430FR6989	1	16	128 KB	5	40
MSP430FR2433	1	16	16 KB	4	20

Optou-se por escolher o **Microcontrolador MSP430F5529**, uma vez que satisfaz as necessidades deste projeto, tanto como estação emissora como para a estação recetora dos dados.

Fornecimento de energia [9] – A alimentação do sistema irá ser por meio de baterias e painéis solares. Como os componentes definidos, conseguimos averiguar a necessidade energética do sistema.

Tabela 7 - Fornecimento de energia

EQUIPAMENTO	MARCA/LOCAL	VOLTAGEM DE FUNCIONAMENTO	AMPER DE FUNCIONAMENTO	PREÇO
BATTERY HOLDER PRO KIT 15PCS	chines ebay	-	-	8,81 €
3.7V 4000MAH LI PO 125054	chines ebay			7.64 €

Por questões de logística e aquisição, optou-se pelo sensor **3.7V 4000mAh Li Po 125054**.

Computador – De modo a conseguir tratar os dados transmitidos pelo equipamento instalado na ETAR utiliza-se um computador com capacidade de correr uma base de dados. Quer-se poder averiguar as leituras e executar comparações como por exemplo leituras por data ou valores máximos.

4.1.3 Módulo de Comunicações

Comunicação e transmissão de dados [10] – Para a transmissão de dados optamos por um sistema repartido por emissão/ receção via radio e uma central para tratamento dos mesmos do tipo *Raspberry Pi*. Propomos:

Tabela 8 - Comunicação e transmissão de dados

EQUIPAMENTO	MARCA/LOCAL	VOLTAGEM DE FUNCIONAMENTO	AMPER DE FUNCIONAMENTO	ANTENA	DISTANCIA	PREÇO
HC-12	chines ebay	3,2 - 5,5V	1uA	914MHz	1000m	2,86 €
NRF24L01 RADIO TRANSCEIVER	chines ebay	1,9 - 3,6V	1uA	2,4GHz	15 - 610 m	1,29 €
LORA SX1278	chines ebay	1,8 - 3,6V	1uA	868MHz	2000m	10,44 €
LORA GAMMA 868	chines ebay	3.6 – 15V	155mA	869.5MHz	16000 m	33,50 €

De modo a garantir um bom funcionamento da unidade escolheu-se o equipamento **LORA GAMMA 868** com um alcance de 16km.

Sistema de georreferenciação [11] – Para o sistema de georreferenciação ponderou-se os seguintes elementos:

Tabela 9 - Sistemas GPS

EQUIPAMENTO	MARCA/LOCAL	VOLTAGEM DE FUNCIONAMENTO	AMPER DE FUNCIONAMENTO	PREÇO
SIM900	chines ebay	4,8 ate 5V	1.5mA	14,38 €
GPS MINI NEO 7M	chines ebay			6,17 €

Por questões de logística e aquisição, optou-se pelo sensor **GPS MINI NEO 7M**.

4.1.4 Custos do Projeto

Apresenta-se na tabla seguinte o material escolhido para o projeto assim como o seu custo total e sites para fornecimento.

Tabela 10 - Custos totais do projeto

EQUIPAMENTO	DESCRIÇÃO	VALOR UNITÁRIO	QT.	TOTAL	STOCK PESSOAL	LINK VENDEDOR
MICROCONTROLADOR	MSP430FR5969	12,90 €	1	12,90 €	sim	
SENSOR TEMPERATURA	Waterproof DS18B20 Sensor	6,50 €	1	6,50 €	sim	
LORA	Gamma-868-SO	15,15 €	2	30,30 €	não	ebay
ULTRASONS	MB1043 HRLV-MaxSonar-EZ4	28,10 €	1	28,10 €	não	maxbotic
SENSOR DE PARTÍCULAS EM SUSPENSÃO	SEN0189	15,15 €	1	15,15 €	não	dfrobot
SENSOR DE CONTACTO	Pulley Switch	5,89 €	1	5,89 €	sim	
BATERIA	3.7V 4000mAh Li Po 125054	7,64 €	1	7,64 €	sim	
GPS	GPS Mini NEO-7M	6,17 €	1	6,17 €	sim	
TOTAL				*112,65 €*		
TOTAL IPBEJA				**73,55 €**		

Estima-se um custo total para o projeto de 112,65€, no entanto o grupo dispõe de algum material necessário (como indicado na tabela) pelo que o custo reduz para **73,55€ (setenta e três euros e cinquenta e cinco cêntimos).**

4.2 Software

O software previsto para este projeto, nesta fase de desenvolvimento, serão os seguintes:

- Criação do código em Code Composer Studio [12] (Texas Instrumentes) para os MSP430;
- Exportação dos dados brutos para ficheiro Excel;
- Criação de base de dados tipo MongoDb [13] para consulta e manuseamento dos dados.

O IDE da Texas Instrumentes permite um controlo a nível de bit nas placas MSP430, o que possibilita colocar as mesmas em baixo consumo de energia (Low Power Mode). Também possibilita uma otimização a nível de linguagem de programação o que consequentemente é mais uma forma para a otimização do consumo energético.

Idealiza-se uma base de dados pensada para "Big Data" como o MongoDb pela incerteza do volume de dados a tratar. O MongoDB é *schemalees* mas estruturado. Quer isto dizer que não ficamos restringidos a uma organização rígida de estrutura para os dados. Esta ultima afirmação pode demonstrar-se particularmente vantajosa em situações de adição de sensores com outros parâmetros de leitura.

5. Desenvolvimento

Neste capítulo do relatório pretende-se demonstrar a implementação do sistema a montar na obra de entrada da ETAR (estação emissora) para monitorizar o caudal do afluente, temperatura e turbidez. A localização é obtida pelo modulo GPS. As medições serão enviadas por LoRa para o servidor que se encontra no Instituto Politécnico de Beja (estação recetora) onde os valores serão tratados e armazenados.

Os dados enviados pela estação emissora não são tratados, i.e., apenas envia o valor da voltagem de cada sensor. Com este método, não necessitamos de efetuar cálculos no microcontrolador MSP430 e conseguindo-se uma poupança no consumo de energia do sistema.

Do lado da estação recetora, esta um modulo LoRa ligado ao Raspberry Pi 3, que trata os dados recebidos.

No entanto, com a implantação do sistema surgiram alterações ao equipamento a aplicar, nomeadamente, não se conseguiu adquirir os módulos LoRa e assim substituíram-se pelos módulos GSM/GPRS Sim900 [14]. Esta alteração permite a eliminação do GPS, uma vez que a localização é obtida por triangulação, e representa mais uma economia de energia no sistema.

5.1 Hardware

Como *hardware* utilizámos uma placa MSP430G2553, um sensor de turbidez, um sensor analógico de temperatura (NTC thermistor) e um HC-SR04 Ultrasom como medidor de distância.

Por questões praticas alterou-se a placa MSP430FR2433 pela MSP430G2553 e não se implementou o sensor de ultrassom. Este sensor ira depender de uns LED's que indicam o estado do sistema, a cor vermelha significa comporta acionada, amarelo representa o bypass ladrão e o LED verde indica o estado normal de funcionamento do sistema. No esquema da figura 8 o sensor da comporta é representado pelo botão.

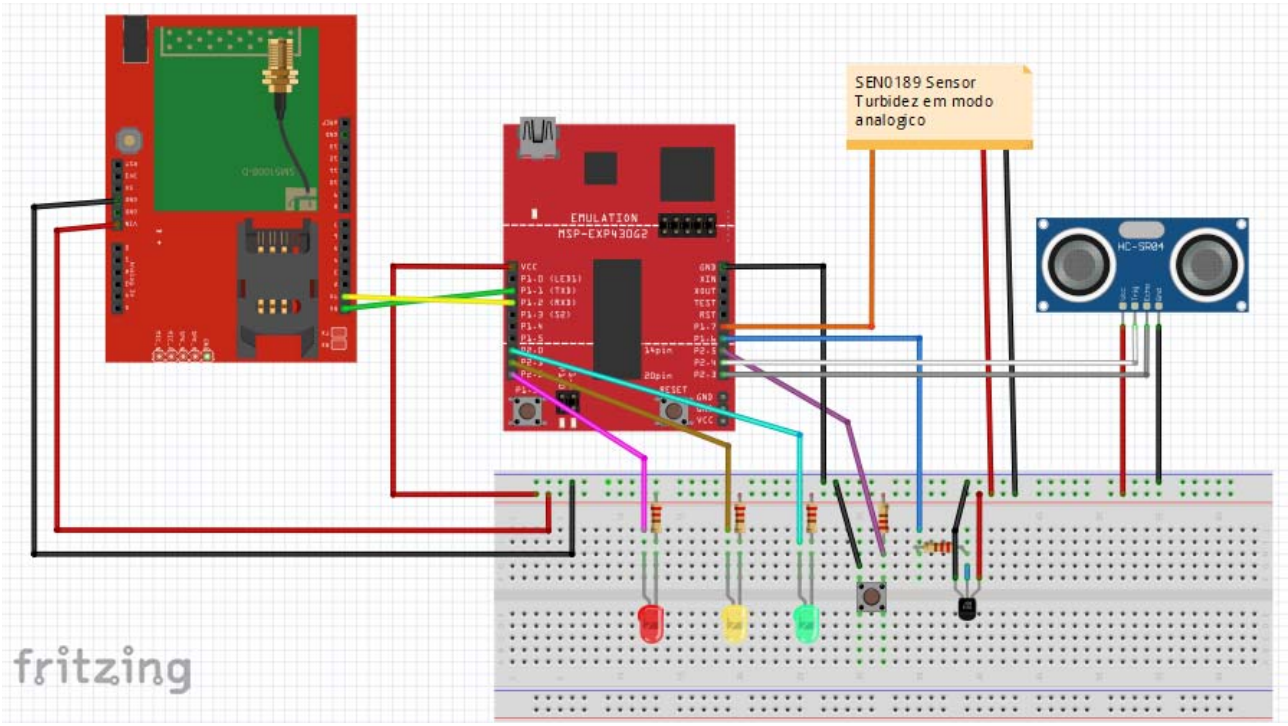


Figura 8 - Esquema de montagem do circuito a implementar

Todo o sistema foi testado em plataforma Arduíno, uma vez que permite um *Debug* mais rápido e intuitivo. Após a realização das ligações na placa de desenvolvimento MSP430G2553 e verificada a consistência do sistema, efetuada por confirmação de receção de dados via USB-TTL na interface *PuTTY [14]*, procedeu-se aos testes de comunicação com o modulo SIM900. Lamentavelmente um desses testes provocou um curto-circuito no modulo de comunicação, o que impossibilitou a implementação estudada do sistema.

Como forma de poder concluir o presente trabalho em tempo útil e por falta de módulos de comunicação *wireless* suplentes ou alternativos *(como por exemplo modulo Bluetooth ou RF433)*, decidiu-se avançar com uma ligação direta TX-RX ao Raspberry Pi 3, conforme apresentado na figura 9.

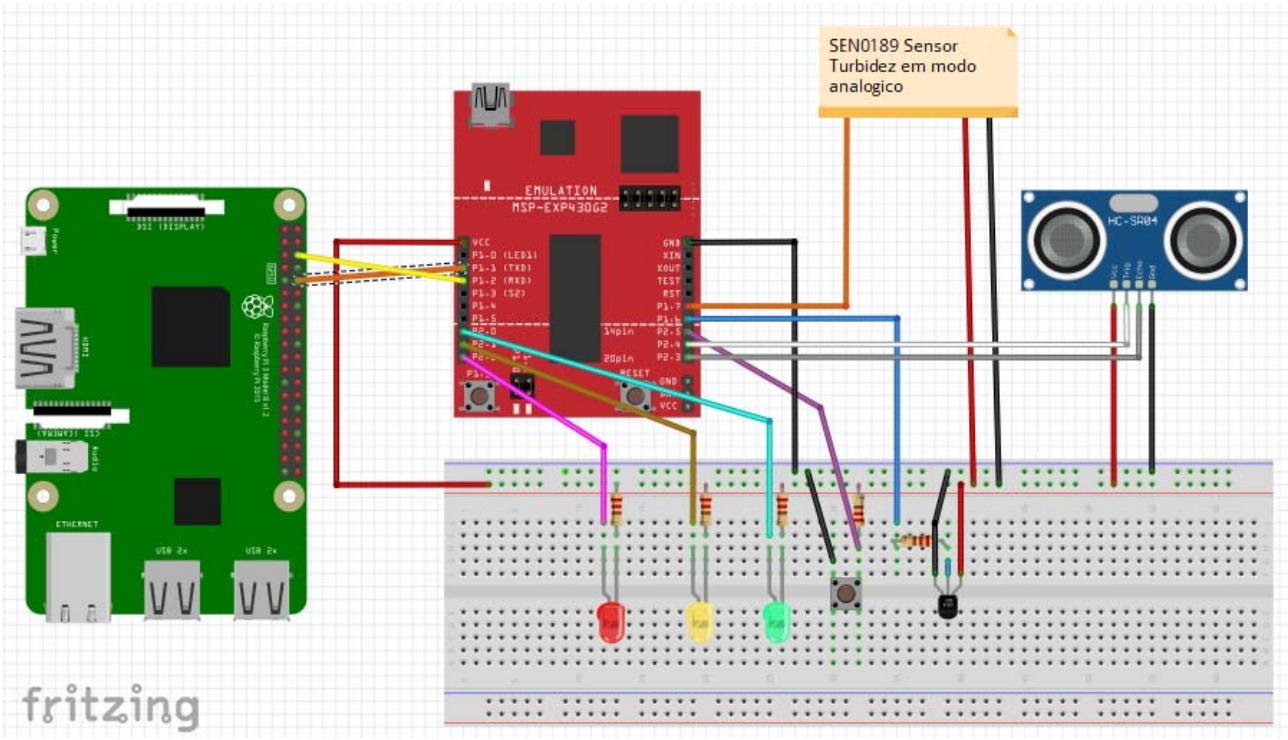


Figura 9 - Esquema de montagem alternativo do circuito a implementar

Os resultados obtidos por meio desta ligação foram os esperados, uma vez que a ligação é bastante básica.

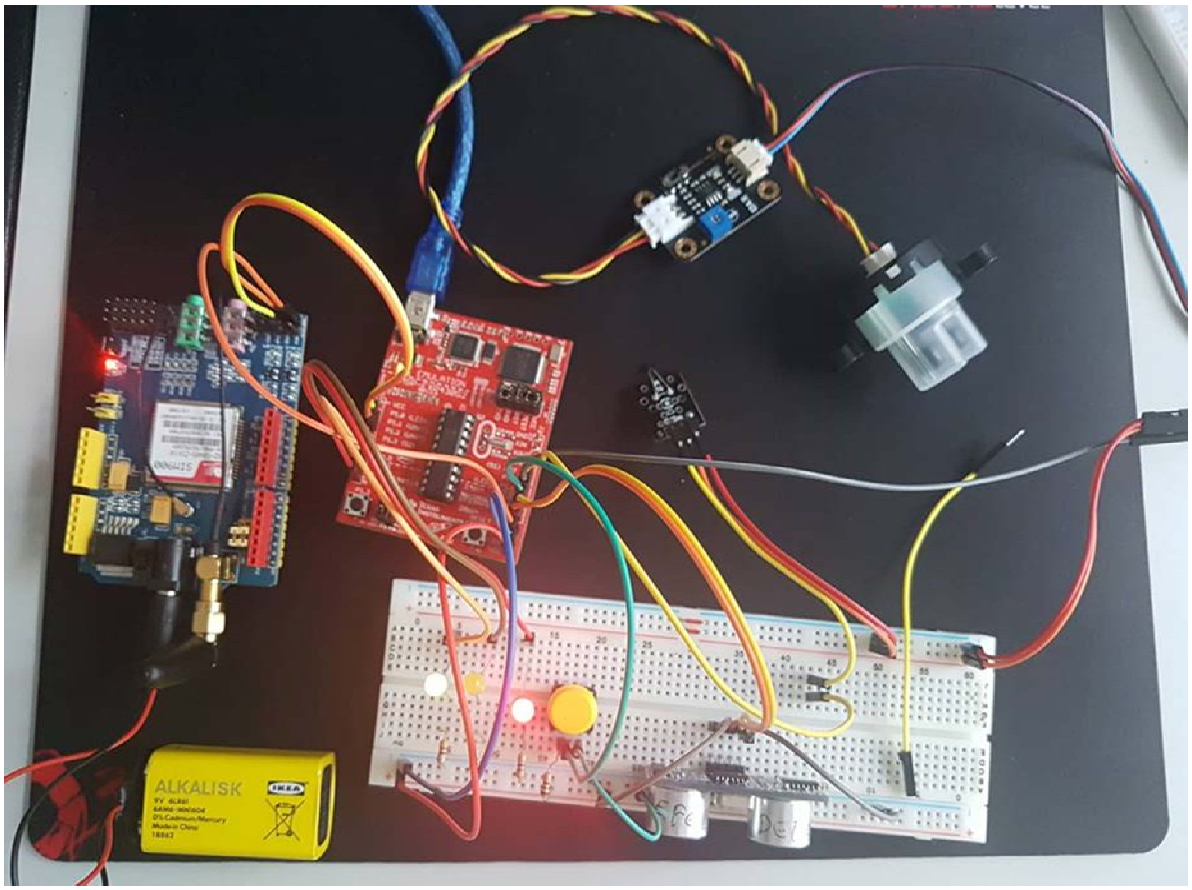


Figura 10 - Esquema físico com LED ligado e potenciómetro a 30%

5.2 Software

Para que este exemplo funcione de maneira pretendida foi utilizado a funcionalidade *Comparator*, sendo que foi escolhido como *input* o CA6 (pin P1.6) e como output o pin P1.3, este opera da seguinte forma: sempre que a corrente ultrapasse um valor referência (que neste exemplo será 0.5*VCC) a placa envia um sinal de HIGH para o pin P1.3, fazendo com que o LED acenda. O código completo utilizado para este exemplo encontra-se nos apêndices.

```
import serial
import time

port = serial.Serial("/dev/ttyS0", baudrate=9600)

printFlag = False
counter = 0;

while True:
    rcv = port.readline()
    if rcv == "Leitura\n":
        printFlag = True

    if counter == 3:
        printFlag = False
        counter = 0

    if printFlag:
        counter+= 1
        print(rcv)
```

Figura 11 - Código Python

Na figura 11 está presente o código desenvolvido para o equipamento recetor (Raspberry PI). Este código foi produzido utilizando a linguagem Python. Dado o tempo disponível apenas foi possível efetuar uma comunicação básica entre o dispositivo emissor (MSP430) e o recetor, sendo que este código está sempre a correr e por sua vez o equipamento recetor estará sempre "a ouvir" por um sinal. Em ambiente real a comunicação poderia não ser constante, havendo falhas e interrupções no sinal, pelo que seria necessário implementar um ou mais sistemas de *failsafe* tanto no lado do emissor como do recetor.

O objetivo deste código é simplesmente ouvir todas as comunicações no serial "ttyS0" (ou Serial0) e filtrar as informações recebidas de forma a ser possível trata-las posteriormente. Sempre que é recebido uma comunicação "Leitura" o sistema irá preparar-se para ler esta mesma mensagem e as duas seguintes, neste caso irá fazer apenas um print, em ambiente real estas informações seriam enviadas para uma base de dados depois de tratadas.

Nota: Para conseguir colocar o Raspberry Pi a ler as comunicações pelo Serial 0 foi necessário alterar o ficheiro de configuração *config.txt* e *cmdline.txt*, de acordo com a documentação do Raspberry Pi, que se pode consultar aqui: https://www.raspberrypi.org/documentation/configuration/uart.md.

```
{
    for(unsigned int i = 0; i < NUM_ADC_CHAN; i++)
    {
        vectAverage[i] += arraySamples[i];
    }

    if(sampling_num == (NUM_SAMPLINGS - 1))
    {
        sampling_num = 0;
        // Average
        for(unsigned int i = 0; i < NUM_ADC_CHAN; i++)
        {
            vectAverage[i] = vectAverage[i] >> 3;
        }


        // Send to console
        for (unsigned int i = 0; i < NUM_ADC_CHAN; i++)
        {
            if(i==2)
                continue;
            dataTX(sensors[i]);
            int2char (strout, vectAverage[i]);
            dataTX(strout);
            dataTX("\n");
            vectAverage[i] = 0;
        }
    }
    else if(sampling_num == (NUM_SAMPLINGS - 2)) {
         sampling_num++;
         dataTX("Leitura");
         dataTX("\n");
    }
    else
    {
        sampling_num++;
        dataTX(int2char (strout, sampling_num));
        dataTX("\r\n");
    }

    ADC10_StartSampling();
```

Figura 12 - Excerto do código C

O código para o equipamento emissor (MSP430), excerto presente na figura 12, foi baseado no código exemplo disponibilizado pelo professor, uma vez que este já estava configurado com as funções base para o projeto, nomeadamente a utilização do ADC10 com interrupções e a comunicação UART.

No excerto de código que se consegue visualizar na figura 2 está então programado a recolha de oito leituras dos dois sensores que estão ligados ao MSP430, sendo que no fim de cada ciclo o MSP430 é colocado em *low power mode*. No fim dos oito ciclos é então calculada a média das leitras e feito o envio dos valores através de comunicação UART.

6. Teste e Implementação do Sistema

O desenvolvimento de um produto segue um ciclo de analise-design-desenvolvimento-teste-produção a semelhança do indicado na figura 13.

Neste capítulo focamo-nos principalmente na última fase do ciclo de vida de um projeto, a fase de teste e implementação.

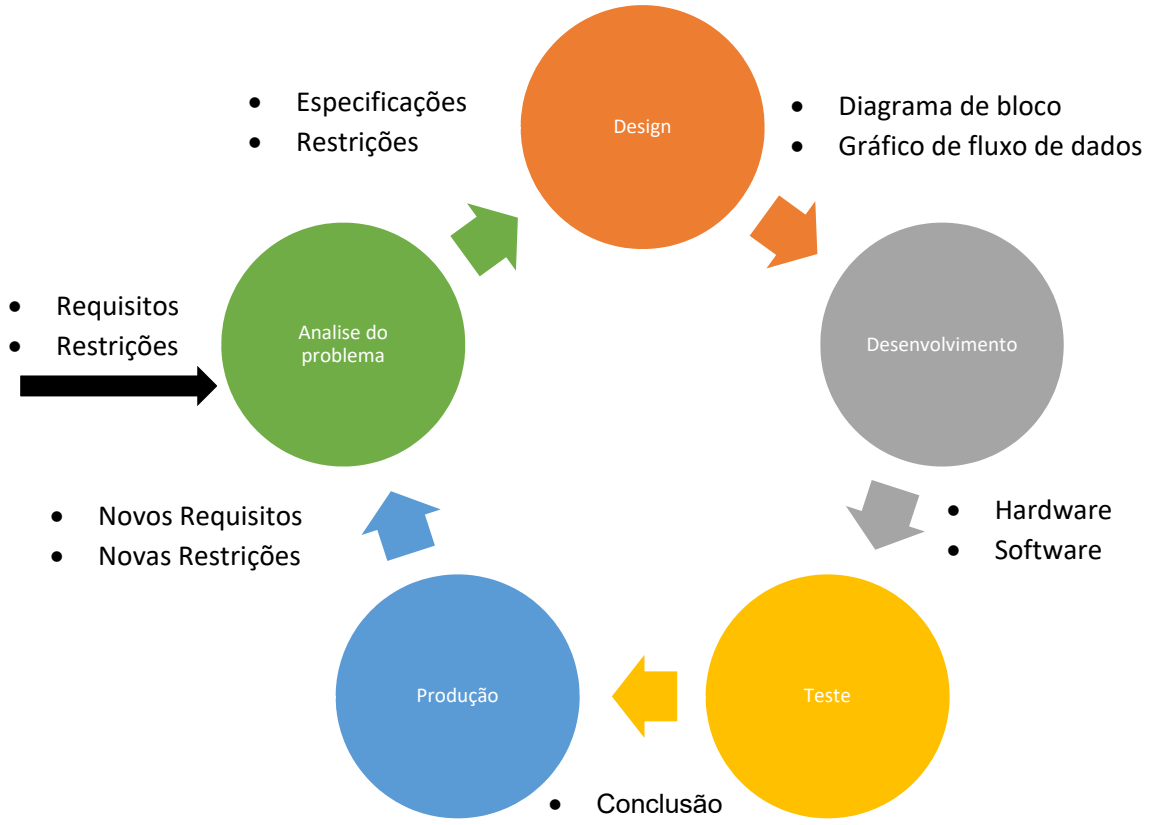


Figura 13 - Ciclo do Produto

É importante saliente que apenas utilizamos a fase de teste para o exemplo demonstrado no capítulo 5, uma vez que não temos todo o material necessário para construir o projeto.

A analise e resultados dos parâmetros de otimização são os seguintes:

Eficiência estática (utilização da memoria) = 1006kb;

Eficiência dinâmica (velocidade de execução) = N.D.;

Precisão (diferença entre a verdade esperada e medida) = Preciso;

Estabilidade (operação consistente) = Estável

Os valores apurados indicam, no nosso entender, uma boa eficiência. No entanto, o próximo passo será a colocação do resto do código do projeto e depois otimizar o programa como um todo.

Para determinar os valores supracitados utilizou-se a informação apresentada no Code Composer Studio, também se efetuou uma análise de consumo de energia utilizando-se para esse efeito o modo *EnergyTrace™ Technology* do microcontrolador MSP430FR2433.

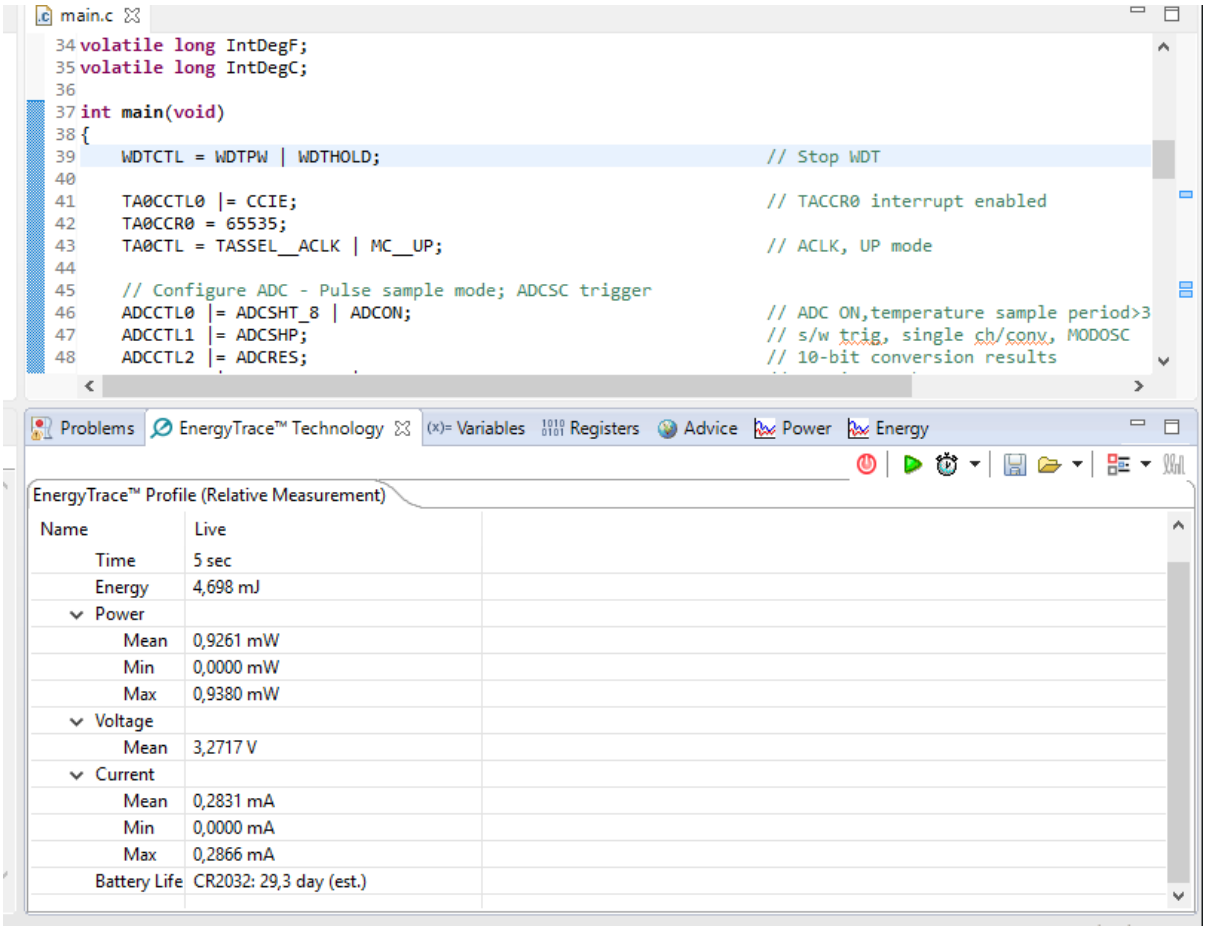


Figura 14 - Consume de energia - EnergyTrace™ Technology MSP430FR2433

É de referir que os valores demonstrados não contemplam o consumo de energia geral do sistema, uma vez que apenas conseguimos medir o consumo do MSP430. Os restantes consumos retiramos dos respetivos *datasheets* dos módulos a incluir no sistema de monotorização.

7. Conclusão

Durante a realização deste projeto foram sentidas algumas dificuldades, nomeadamente na implementação das comunicações UART e configurações dos *timers*. A conjugação e definição certa entre os vários parâmetros do código a nível de bit é notoriamente complexa. As constantes mudanças entre manual de utilizador e manual específico não ajudam a uma rápida organização de tarefas a implementar. No entanto serviu para nos familiarizarmos com o mundo dos microcontroladores avançados. Designamos avançados porque em comparação com outros microcontroladores mais *user friendly* (como por exemplo o Arduino), os MSP430 permitem afinações quase cirúrgicas, criando assim sistemas altamente precisos e adaptados ás necessidades dos projetos. Outra grande vantagem é o sistema de *Low Power Mode* que permite “hibernar” o controlador quando este não é utilizado e assim poupar as baterias. Para a implementação deste projeto, foram sacrificados dois integrados MSP430, dois USB-TTL e um modulo GSM/GPRS.

Tendo em conta as dificuldades supramencionadas e o tempo para entrega do trabalho, apenas conseguimos ligar o sistema por comunicação física ao Raspberry Pi e apresentar as leituras na linha de comandos.

Uma possível continuidade deste trabalho, devia prever outros sistemas acessórios interligados (por exemplo comunicação wireless) em todas as comportas que compõem o sistema de bypass, para se poder monitorizar qualquer descarga de afluente sem tratamento ou com tratamento insuficiente para as linhas de água.

É de salientar que o desenvolvimento e implementação deste sistema, no mínimo em todas as obras de entrada das ETAR’s a nível nacional, é urgente e de extrema necessidade para um controlo rigoroso das descargas para as linhas de água. Também é uma forma de salvaguardar o meio ambiente e garantir a qualidade da água nos meios recetores.

8. Referências bibliográficas

[1] - EnunciadoTrabalho_SE_-2017-2018.pdf

[2] - http://www.ecodepur.pt/pt/4/produtos

[3] - http://www.dec.ufcg.edu.br/saneamento/PARSHALL.html

[4] - Sensor Temperatura:

- https://www.ebay.com/itm/DS18B20-Waterproof-Digital-Temperature-Sensor-With-Adapter-Module-for-Arduino-HM/201639236501?hash=item2ef2a29395:g:ziYAAOSwCfdXpFMi
- http://www.directindustry.com/industrial-manufacturer/water-temperature-sensor-161988.html
- https://www.monnit.com/Product/MNS2-9-IN-TS-WT-L03

[5] - Sensor de distância:

- https://www.ebay.com/itm/1pcs-Ultrasonic-Module-HC-SR04-Distance-Measuring-Transducer-Sensor-for-Arduino/400985326881?epid=1638465117&hash=item5d5c968521:g:kLQAAOxyNyFS-xFw
- https://www.ebay.com/itm/Arduino-Ultrasonic-Ranging-Measuring-Transducer-Sensor-Waterproof-JSN-SR04T/272266517229?hash=item3f64595aed:g:VvcAAOSwtJZXVnh8
- http://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/
- https://www.pepperl-fuchs.com/global/en/classid_53.htm?view=productdetails&prodid=34337

[6] - Sensor de partículas em suspensão

- https://www.dfrobot.com/wiki/index.php/Turbidity_sensor_SKU:_SEN0189
- https://www.usinainfo.com.br/outros-sensores-arduino/sensor-de-turbidez-arduino-para-monitoramento-da-agua-4539.html

[7] - Sensor de contacto

- https://www.boxelectronica.com/pt/varios-sensores/838-conjunto-de-switch-magnetico-para-portas.html

[8] - Microcontroladores:

- http://processors.wiki.ti.com/index.php/MSP430F5529_LaunchPad
- http://www.ti.com/tool/MSP-EXP430F5529LP
- http://www.ti.com/tool/MSP-EXP430G2
- http://www.ti.com/product/msp430fr2422
- http://www.ti.com/tool/MSP-EXP430FR5969
- http://www.ti.com/product/MSP430FR6989

[9] - Fornecimento de energia

- https://www.ebay.com/itm/Battery-Holder-Pro-KIT-15pcs-Power-Supplies-for-Arduino-Solar-4AA-3AA-2AA-DC-EU/222242451205?hash=item33beaeaf05:g:gQgAAOSw-0xYOwYo

[10] - Comunicação e transmissão de dados:

- https://www.ebay.com/itm/433Mhz-HC-12-SI4463-Wireless-Serial-Port-Module-1000m-Replace-Bluetooth-TOP/401051275954?epid=26007567495&hash=item5d6084d2b2:g:JawAAOSwMHdXS9OD
- https://www.ebay.com/itm/Wireless-Radio-Transceiver-Module-Communication-Control-for-Arduino-DIY/322668815062?_trkparms=aid%3D555018%26algo%3DPL.SIM%26ao%3D2%26asc%3D44040%26meid%3Dbf00b29db36c45de9590955d7d66ce48%26pid%3D100005%26rk%3D1%26rkt%3D6%26sd%3D262966959176%26itm%3D322668815062&_trksid=p2047675.c100005.m1851
- https://arduino-info.wikispaces.com/Nrf24L01-2.4GHz-HowTo
- https://www.cooking-hacks.com/blog/send-data-at-extreme-long-range-using-lora-with-arduino-raspberry-pi-and-intel-galileo/
- https://www.cooking-hacks.com/blog/send-data-at-extreme-long-range-using-lora-with-arduino-raspberry-pi-and-intel-galileo/

[11] - Sistema de georreferenciação

- https://www.ebay.com/itm/SIM900-Module-Quad-Band-Development-Board-GSM-GPRS-for-Arduino-Raspberry-Pi-E7G2/253162539156?epid=2248191795&hash=item3af1a9c894:g:HhUAAOSwFuxZwP7n

[12] – http://www.ti.com/tools-software/ccs.html

[13] – https://www.mongodb.com/?_ga=2.110403590.208858189.1520880704-1686614365.1516833254

[13] – https://www.ebay.com/itm/SIM900-850-900-1800-1900-MHz-GPRS-GSM-Development-Board-Module-Kit-For-Arduino/263106376867?hash=item3d425cb4a3%3Ag%3AwJMAAOSwzXBZhuns&_sacat=0&_nkw=sim900&_from=R40&rt=nc&_trksid=p2050601.m570.l1313.TR8.TRC1.A0.H0.Xsim900.TRS0

[14] – https://putty.org/

9. Apêndices

```
/*
 * filename: ADC10_UART.c
 */

#include <msp430.h>
#include <math.h>

/*
 * The ADC10 samples channels A7 through A3 using TimerA0.CCR1 (TA1).
 * The internal reference voltage is on with the value of 1.5V and
 * is output on P1.4.
 * The sampling is triggered by the Timer0_A3 with the signal OUT1
 * (generated from CCR1 in Set/Reset output mode). The MSC is set to
 * take multiple sample and conversions, that is set to 5 samples (A7 and A3)
 * with the ADC10DTC1 register. The samples are transfered to arraySamples by
 * the DTC. The example implements a simple UART communication to send the data
 * serially to the computer.
 *
 *              MSP430G2553
 *            -----------------
 *        /|\|              XIN|-
 *         | |                 |
 *         --|RST          XOUT|-
 *           |                 |
 *     RX <---|P1.1             |
 *     TX <---|P1.2             |
 *           |                 |
 *     A4 ><--|P1.4(VREF+)  P1.6|--< A6
 *    (1.5V)  |             P1.7|--< A7
 *           |                 |
 *
 */

#define NUM_ADC_CHAN 3
#define NUM_SAMPLINGS 8

unsigned int arraySamples[NUM_ADC_CHAN];
unsigned int vectAverage[NUM_ADC_CHAN];
unsigned int matrixSamples[NUM_ADC_CHAN][NUM_SAMPLINGS];

char strout[5];

char *sensors[NUM_ADC_CHAN] = {"Temperatura: ",
                               "Turbidez: ",
                               "Vref: ",};

// WDT:
void WDT_Init(void);
// GPIO:
void GPIO_Init(void);
// BCS+:
void BCS_Init(void);
// Timer0_A3:
void Timers_Init(void);
// ADC10):
void ADC10_Init(void);
// USCI_A0:
```

```
void USCI_A0_Init(void);

void ADC10_StartSampling(void);
// Funções genéricas
char * int2char (char *strout, int num);
void dataTX(char *strout);

/**
 * main.c
 */

unsigned int sampling_num = 0;


int main(void)
{
    WDT_Init();
    GPIO_Init();
    BCS_Init();
    Timers_Init();
    ADC10_Init();
    USCI_A0_Init();


    ADC10_StartSampling();

    __enable_interrupt();
    _low_power_mode_0();

     while(1)
    {
        for(unsigned int i = 0; i < NUM_ADC_CHAN; i++)
        {
            vectAverage[i] += arraySamples[i];
        }

        if(sampling_num == (NUM_SAMPLINGS - 1))
        {
            sampling_num = 0;
            // Average
            for(unsigned int i = 0; i < NUM_ADC_CHAN; i++)
            {
                vectAverage[i] = vectAverage[i] >> 3;
            }



            // Send to console
            for (unsigned int i = 0; i < NUM_ADC_CHAN; i++)
            {
                if(i==2)
                    continue;
                dataTX(sensors[i]);
                int2char (strout, vectAverage[i]);
                dataTX(strout);
                dataTX("\n");
                vectAverage[i] = 0;
            }
```

```
        }
        else if(sampling_num == (NUM_SAMPLINGS - 2)) {
            sampling_num++;
            dataTX("Leitura");
            dataTX("\n");
        }
        else
        {
           sampling_num++;
           dataTX(int2char (strout, sampling_num));
           dataTX("\r\n");
        }

        ADC10_StartSampling();
        _low_power_mode_0();
    }

}

void WDT_Init()
{
    WDTCTL = WDTPW | WDTHOLD;
}

void GPIO_Init()
{
    // Port 1:
    // P1.0 - n/c or ACLK Output
    // P1.1 - UART RX
    // P1.2 - UART TX
    // P1.3 - n/c
    // P1.4 - V_REF+ output
    // P1.5 - n/c
    // P1.6 - ADC10 A6
    // P1.7 - ADC10 A7

    P1OUT = 0x00;   // Set all outputs to zero as default
    P1DIR = 0xFF;   // Configure all pins as outputs by default;
    P1SEL |= BIT1   // RX
          | BIT2;   // TX

    P1SEL2 |= BIT1   // RX
          | BIT2;   // TX

    // Port 2:
    // P2.0 - n/c
    // P2.1 - n/c
    // P2.2 - n/c
    // P2.3 - n/c
    // P2.4 - n/c
    // P2.5 - n/c
    // P2.6 - n/c or LFXT1: XIN
    // P2.7 - n/c or LFXT1: XOUT
    P2OUT = 0x00;   // Set all outputs to zero as default
    P2DIR = 0xFF;   // Configure all pins as outputs by default;

    // LFXT1 : P2.6 - XIN, P2.7 - XOUT; see SLAS735J pg. 53, 55
```

```
    //    P2DIR  &= ~BIT6;
    //    P2SEL  |= (BIT6 | BIT7);
    //    P2SEL2 &= ~(BIT6 | BIT7);
}


void BCS_Init()
{
    DCOCTL = CALDCO_8MHZ;   // DCO frequency select and Modulator selection:
    // are set with factory calibrated value
    BCSCTL1 = XT2OFF        // XT2 off
          | DIVA_0        // Divider for ACLK: /1
          | CALBC1_8MHZ;  // Range select: factory value for 8MHz

    BCSCTL2 = SELM_0        // Select the MCLK source: DCOCLK
          | DIVM_0        // Divider for MCLK: /1
          | DIVS_0;       // Divider for SMCLK: /1

    BCSCTL3 = LFXT1S_2;      // Low-frequency clock select and LFXT1 range
    // select: VLOCLK
}

void Timers_Init(void)
{
    // Configure Timer0_A3:
    // TA0R:
    TA0CTL = TASSEL_1  // Timer_A clock source select:

          | ID_3          // Input divider: /1
          | MC_0      // Mode control: Stop
          | TACLR;        // Timer_A clear

    // TA0CCR0:
    TA0CCR0 = 1023;

    // TA0CCR1:
    // Set CCR1 on Compare Mode Enable the CC2IFG:
    // TA0.1 Out1 (P1.3-slas590m pg 15, 82)
    TA0CCTL1 =  OUTMOD_3;    // Output mode: Set/reset
    //          | CCIE;         // Capture/compare interrupt enable
    TA0CCR1 = 512;   // Value for the comparison to the timer CCR0 register

    // Start the timer in Up Mode:
    TA0CTL |= MC_1;
}


void ADC10_Init(void)
{
    // ADC10 Control Register 0:
    ADC10CTL0 = SREF_1      // Select reference: VR+ = VREF+ and VR- = VSS
          | ADC10SHT_3    // ADC10 sample-and-hold time: 64 × ADC10CLKs
          | ADC10SR       // ADC10 sampling rate: Reference buffer supports up
          // to ~50 ksps
          | REFOUT        // Reference output on. (VeREF+ / VREF+ pin)
          //          | REFBURST      // Reference buffer on only during
          // sample-and-conversion
```

```
        | MSC          // Multiple sample and conversion
        //          | REF2_5V      // Reference-generator voltage 2.5V
        | REFON        // Reference generator on: V_REF+ takes 30us to
        // settle (slau144j pg. 38)
        | ADC10ON      // ADC10 on
        | ADC10IE;     // ADC10 interrupt enable

    // ADC10 Control Register 1:
    ADC10CTL1 = INCH_7     //  Input channel selection: A7-A6
        | SHS_1        //  Sample-and hold source select: Timer0_A.OUT1
        | ADC10DIV_3   // ADC10 clock divider: /4
        | ADC10SSEL_0  // ADC10 clock source select: ADC10OSC
        | CONSEQ_1;    // Conversion sequence mode select:
    //    Sequence-of-channels (A7 - A6)


    // Analog (Input) Enable Control Register 0:
    ADC10AE0 = BIT7
        | BIT6
        | BIT5
        | BIT4        // V_REF+ on P1.4
        | BIT3;


    // ADC10 data transfer control register 1:
    ADC10DTC1 = NUM_ADC_CHAN;       // Define the number of transfers: 3 values
}

void USCI_A0_Init(void)
{
    // UCA0 in reset mode for configuration
    UCA0CTL1 = UCSWRST;

    // UART Mode: 9600 8N1
    UCA0CTL1 |= UCSSEL_2; // USCI clock source select: SMCLK

    // From Table 36.4 in slau144j:
    // BRCLK Frequency (Hz)     Baud Rate   UCBRx   UCBRSx  UCBRFx
    // 8 000 000                9600        833     2       0
    UCA0BR0 = 0x41;     // UCBRx = 833 = 0x341
    UCA0BR1 = 0x03;

    UCA0MCTL = UCBRS_2;

    // BRCLK Frequency (Hz)     Baud Rate   UCBRx   UCBRSx  UCBRFx
    // 8 000 000              115200      69        2       0
//    UCA0BR0 = 0x45;     // UCBRx = 69 = 0x45
//    UCA0BR1 = 0x00;
//
//    UCA0MCTL = UCBRS_4;

    // USCI reset released for operation:
    UCA0CTL1 &= ~UCSWRST;

    IE2 |= UCA0RXIE;    // USCI_A0 receive interrupt enable
}
```

```
void ADC10_StartSampling(void)
{
   //while (ADC10CTL1 & ADC10BUSY);  // Wait if ADC10 core is active
   // ADC10 data transfer start address:
   ADC10SA = (unsigned int) arraySamples; // ADC10 start address
   ADC10CTL0 |= (ENC | ADC10SC);     // Enable conversion
}

char * int2char (char *strout, int num){
   char straux[4];
   unsigned int i = 0;
   while (num >= 10){
      straux[i] = num % 10;
      num = num / 10;
      ++i;
   }
   strout[0] = '0' + num;
   unsigned int j  = 1;
   //   --i;
   while (i > 0){
      strout[j++] = '0' + straux[--i];
      //       --i;
      //       ++j;
   }
   strout[j] = '\0';

   return strout;
}


#pragma vector = ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
   ADC10CTL0 &= ~ENC;
   __low_power_mode_off_on_exit();
}


void dataTX(char *strout)
{
   unsigned int i = 0;
   while(strout[i] != '\0')
   {
      while(!(IFG2 & UCA0TXIFG));
      UCA0TXBUF = strout[i];
      i++;
   }
}
```

```
import serial
import time

port = serial.Serial("/dev/ttS0", baudrate=9600)

printFlag = False
counter = 0

while True:
	rcv = port.readline()
	if rcv == "Leitura\n":
		printFlag = True

	if counter == 3:
		printFlag = False
		counter = 0

	if printFlag:
		counter += 1
		print(rcv)
```

```
// Ultrasonic rangefinder example for MSP430G2553
// - An LED is driven by P1.3
// - The trigger pulse is sent to the rangefinder via P1.2
// - The echo pulse is read from the rangefinder via P1.1
//  Special thank to Ali Gokoglu

#include <msp430.h>

void main(void)
{
   int distance_cm;

   // Disable watchdog timer
   WDTCTL = WDTPW + WDTHOLD;

   BCSCTL1 = CALBC1_1MHZ;
   DCOCTL = CALDCO_1MHZ;

   TA1CTL = TASSEL_2 + ID_0 + MC_2;

   //P2DIR = 0b00000111; // P2.0 2.1 2.2 as output LED
   P1DIR = 0b00000100; // P1.2 is an output, P1.1 is input

   //P1DIR = 0x03;

   while(1)
   {
      distance_cm = read_distance_sensor_cm();

      if (distance_cm < 50) P2OUT |= 0b00000001;       // LED on

      else P2OUT &= ~0b00000001;                       // LED off
   }
}

int read_distance_sensor_cm()
{
   int echo_pulse_duration;       // time in us
   int distance;                  // distance in cm

   // Send a 20us trigger pulse
   P1OUT |= 0b00000100;                  // trigger high
      __delay_cycles(20);                // 20us delay
   P1OUT &= ~0b00000100;                 // trigger low

   // Measure duration of echo pulse
   while ((P1IN & 0b00000010) == 0);      // Wait for start of echo pulse
   TA1R = 0;                              // Reset timer at start of pulse
   while ((P1IN & 0b00000010) > 0);       // Wait for end of echo pulse
   echo_pulse_duration = TA1R;            // Current timer value is pulse length

   distance = 0.017 * echo_pulse_duration; // Convert from us to cm

   return distance; // Return distance in cm to calling function
}
```

```
//
// Rangefinder example for MSP430G2553
// Written by Ted Burke - Last modified 19-11-2014
//
// This program uses a rangefinder to measure the distance to the
// nearest object. If the distance is below a threshold, an LED lights.
//
//  - Trigger pulses are sent to the rangefinder via P1.0
//  - Each pulses are received from the rangefinder via P1.1
//  - The LED is controlled by P2.0
//

#include <msp430.h>

int main( void )
{
   WDTCTL = WDTPW + WDTHOLD; // Disable watchdog timer

   P1DIR = 0b00000001; // Make P1.0 an output for trigger pulses
   P2DIR = 0b00000111; // Make P2.0 an output for indicator LED

   while(1)
   {
      // Send a 100us trigger pulse on P1.0
      P1OUT = 0b00000001;
      __delay_cycles(100);
      P1OUT = 0b00000000;

      // Wait for start of echo pulse on P1.1
      // (i.e. wait while P1.1 stays low)
      while ((P1IN & BIT1) == 0);

      // Delay for fixed time, then check if P1.1
      // is still high. If not, distance is less
      // than threshold
      __delay_cycles(2000);
      if ((P1IN & BIT1) == 0)
      {
         P2OUT = 0b00000111; // LED on
      }
      else
      {
         P2OUT = 0b00000000; // LED off
      }

      // Now just wait for echo pulse to end
      // (i.e. wait while P1.1 stays high)
      while ((P1IN & BIT1) > 0);
   }

   // The program never actually reaches this point
   return 0;
}
```