Departamento de Engenharia Eletrotécnica

Sistemas de Aquisição de Dados

2º Semestre 2018/2019

Relatório do Trabalho Final

Trabalho Realizado por:

Diogo Lograd nº 43697

Rui Martins nº 45174

Conteúdo

Introdução

Neste trabalho pretende-se implementar um sistema inteligente com intuito de monitorar e assim melhorar a produção numa linha de montagem. Para isso será utilizado a placa de desenvolvimento da Microchip, a Explorer 16 (com PIC24FJ128GA010) para controlar a linha de montagem. Serão também utilizados vários micro-controladores ATTiny841 e um Arduino Uno2 para actuação nos motores da linha de montagem. Cada micro-controlador ATtiny tem um sensor/actuador associado. Para o Microchip comunicar com cada dispositivo é usado a comunicação I2C, utilizada para dar instruções ou receber os dados necessários para a monitorização. A porta série no Microchip, é usada para controlo dos dados, facilitar a identificação de erros por parte dos alunos e para simular o envio de mensagens para a Cloud.

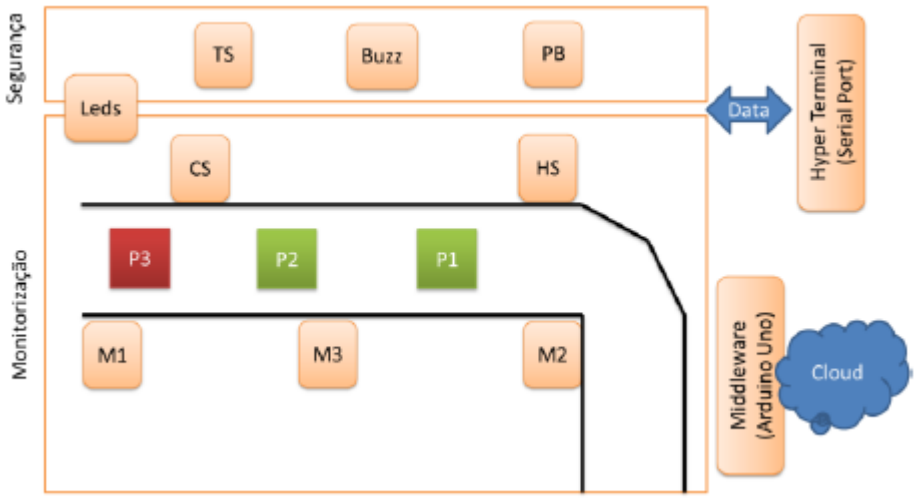

Figura 1 - Linha de montagem com sistema de monitorização

Como observado na **Erro! A origem da referência não foi encontrada.**, existem diversos equipamentos ao longo da linha de montagem, como tal todos possuem uma identificação por forma a que seja possível o seu manuseamento e controlo.

Objetivo

O trabalho divide-se em dois principais objectivos: a) Assegurar a segurança na fábrica; b) Monitorizar a produção.

Para alcançar os objectivos deste trabalho foram colocados vários sensores e actuadores na fábrica, distribuidos da seguinte forma (Figura 1):

1. **Segurança**: Indutive Sensor (IS) usado para parar a produção sempre que ocorrer um problema (actuação automática). Sensor de temperatura (TS) para detectar incêndios.

2. **Acquisição de dados e Actuação**: Três motores (M1, M2 e M3) para actuarem nos tapetes. Sensor de Ultrasom (US) e Sensor de Hall (HS) para identificar de que tapete estão a chegar as peças produzidas. Sensor de Fim de Curso (EC) para identificar que as peças produzidas de ambos os tapetes chegaram ao armazém.
3. **Porta série**: Para visualização do estado da linha, servindo também para mostrar mensagens do estado de sensores e produtos, e para receber/enviar as mensagens JSON da Cloud.

Funcionamento

Nesta linha de montagem estão a ser produzidos dois produtos distintos, que serão recebidos no armazém através das linhas de montagem A e B (representadas na figura 1), cada linha de montagem transporta um tipo de produto. A única entrada do armazém está ligada a dois tapetes, o que faz com que só possa receber um produto de cada vez, e por sua vez ambos os tapetes têm de estar sincronizados para serem actuados um de cada vez. Os sensores US e HS são usados para identificar se existem peças nos tapetes. Sempre que um dos sensores identifica uma peça, o respectivo motor (M1 ou M2) tem de ser actuado para transportar o produto. No final do curso existe um terceiro sensor (EC), que identifica a chegada do produto, devendo-se desligar os motores (M1 ou M2) e ligar o motor M3 para introduzir o produto no armazém. Por fim, faz-se a contagem dos produtos recebidos no armazém para controlo da produção, e envia-se por JSON a contagem.

Dispositivo	**Controlador**	**Endereço**
Sensor de Cor (CS)	Arduíno Uno	0x01
Sensor de Hall (HS)	Arduíno Uno	0x02
Motor 1 (M1)	Arduíno Uno	0x03
Motor 2 (M2)	Arduíno Uno	0x03
Motor 3 (M3)	Arduíno Uno	0x05
Panic Botton (PB)	Microchip	Explorer 16
Luzes de Sinalização (Leds)	Arduíno Uno	0x06
Cloud (Middleware)	Arduíno Uno	0x09
Sensor de Temperatura (TS)	Microchip	Explorer 16
Sirene de Alarme (Buzz)	ATtiny84	0x10

Tabela 1 - Lista de endereços e controladores dos equipamentos

Implementação

Inicialização ADC

Esta função vai inicializar o registo PCFG (p.10 ADC) com as portas analógicas usadas neste trabalho deixando o registo CSSL (p.10 ADC) a zero visto que não pretendemos fazer scan. (AD1CON p.6 ADC).

```
void inicializar_adc(){

     AD1PCFG = 0xFFD1;     //1FFD1    //todos os registos
necessarios para o potenciometros, LDR's e temperatura, em
hexadecimal (AN1, AN2, AN5, AN3)
     AD1CON1 = 0x0000; // SAMP bit = 0 ends sampling
     // and starts converting
     AD1CSSL = 0;     // in this example AN2 is the input
     //TRISAbits.TRISA0 = 0;    //output

     AD1CON1bits.ADON = 1; // turn ADC ON
}
```

Figura 2 - Função de inicialização do ADC

Inicialização UART

É preciso definir o Baud Rate a utilizar, neste caso será 9600, e no datasheet diz para colocar U2BRG (p.9 UART) a 25, valor este que depende da frequência do oscilador da placa. Foi ainda colocado no modo de 8 bits para poder tratar dos dados enviados. Colocou-se então o U2MODE (p.3 UART) com o valor de 0x8000.

```
void inicializar_UART(){
      U2BRG = 25; //Valor calculado atraves da formula
presente no datasheet, para um baud rate de 9600
      U2STA = 0;
      U2MODE = 0x8000; //Enable Uart for 8-bit data
      //no parity, 1 STOP bit
      U2STAbits.UTXEN = 1; //Enable Transmit
}
```

Figura 3 - Função de inicialização do UART

Colocamos no I2C2CON (p.8 I2C) o valor de 0x8000 que dá enable do módulo I2CX e configura os pinos SDAx (dados) e SCLx (clock) como pinos de porto em série. O valor do registo BRG para corresponder aos 4MHz vai ser 39 segundo o datasheet (p.14 I2C).

```
/*********************************************/
/*Nome: inicializar_I2C *********************/
/*Função: Inicializar o protocolo I2C ******/
/*********************************************/
void inicializar_I2C () {
    I2C2CON = 0x8000; //Controlar o registo I2C2CO
    I2C2BRG = 39; //BAUDRATE = 4MHZ
}
```

Figura 4 - Função de inicialização do I2C

No registo I2C2CON temos que dar ainda valor ao SEN e ao PEN (p.9 I2C). O SEN fica a 1 dando início a condição Start do SDAx e do SCLx. O PEN tem o valor de 1 também e dá início à condição Stop.

```
/*****************************************************/
/*Nome: startI2C *************************************/
/*Função: Inicia a comunicação I2C *****************/
/*****************************************************/
void startI2C(void) {
    //Inicia os pinos SDA e SCL
    I2C2CONbits.SEN = 1;
    while(I2C2CONbits.SEN);
}

/*****************************************************/
/*Nome: stopI2C **************************************/
/*Função: Termina a comunicação I2C *****************/
/*****************************************************/
void stopI2C(void) {
    //Ativa o bit PEN do registo I2C2CON para terminar a transmissão
    I2C2CONbits.PEN = 1;
    while(I2C2CONbits.PEN);
}
```

Figura 5 - Funções para iniciar e terminar a transmissão

Esta função serve para enviar informação para o I2C. O registo I2C2TRN (p.7 I2C) é o registo de transmissão, ou seja, o que se coloca neste registo é o que vai ser transmitido. O TRSTAT (p.10 I2C) é um registo que indica se está a ocorrer transmissão de dados. O TBF indica se o buffer está cheio ou não. IWCOL indica se houve alguma colisão.

```
/*Nome: write_data_I2C ********************/
/*Função: Envia dados para o I2C **********/
/*****************************************/
void write_data_I2C(unsigned char dados) {

    char data[30];

    sprintf(data, " Sent: %X \r\n", dados);

    I2C2TRN = dados;

    putString_UART(data);


    if (I2C2STATbits.IWCOL) {
        putString_UART("Erro. \r\n");
        //return 0; // Error, collision writing
    }
    //Se o bit TBF do registo I2C2STAT estiver ativo o buffer está ocupado
    while (I2C2STATbits.TBF);
    while (I2C2STATbits.TRSTAT);
}
```

Figura 4 - Função para enviar dados para o I2C

O registo RCEN vai receber o bit Enable, ou seja, vai ativar o modo Receive. Se o nosso parâmetro de entrada (bits) tiver o valor de 2 então estamos a tratar do sensor de Hall.

```
/*************************************************************/
/*Nome: request_I2C ***************************************/
/*Função: Recebe uma mensagem do I2C ***********************/
/*************************************************************/
int request_I2C(int bits) {

    putString_UART("Receiving... \r\n");

    I2C2CONbits.RCEN = 1;  //Ativa o modo receive do I2C

    while (I2C2CONbits.RCEN); //

    while (!I2C2STATbits.RBF); //

    unsigned char response = I2C2RCV;

    int resposta = (int)response;
    //I2C2CONbits.ACKEN = 1;
    //putString_UART("response \r\n");
    if (bits == 2) { //Recebe uma mensagem de 2 byte(16 bits)
        I2C2CONbits.ACKDT = 0;
        I2C2CONbits.ACKEN = 1;
        while (I2C2CONbits.ACKEN);

        I2C2CONbits.RCEN = 1;

        while (I2C2CONbits.RCEN);
        // Quando a 1 significa que o registo RBF está cheio logo o receive está completo
        while (!I2C2STATbits.RBF);

        resposta = ((I2C2RCV << 8) + response);

        I2C2CONbits.ACKDT = 1;
        I2C2CONbits.ACKEN = 1;

        while (I2C2CONbits.ACKEN);
    }
```

Figura 5 - Função para receber dados do I2C

A função em baixo descrita vai tratar os comandos a enviar. Se quisermos obter uma resposta do slave colocamos a variável stat_receive com o valor de 1, se não for necessário saber a resposta coloca-se então a variável a 0.

```
/*************************************************************/
/*Nome: Talk_to_slave*************************************/
/*Função: Inicializa uma conversa com um slave(envia e recebe dados****/
/*************************************************************/
int talk_to_slave(unsigned char addr_slave, unsigned char cmd, int stat_receive, int n_bits) {

    unsigned char addrs;
    int rec;
    char string[30];
    startI2C();


    addrs = ((addr_slave << 1) | 0);  //shift e mete o primeiro bit a 0 para enviar
    write_data_I2C(addrs);
    write_data_I2C(cmd);


    stopI2C();

    if (stat_receive) { // se tiver alguma resposta do slave
        startI2C();

        addrs = ((addr_slave << 1) | 1); //shift e mete o primeiro bit a 1 para receber

        write_data_I2C(addrs);

        rec = request_I2C(n_bits);

        sprintf(string, " Received: %d \r\n", rec);
        putString_UART(string);

        stopI2C();

        return rec;
    }

    return 0;// Error
}
```

Figura 6 - Função para a comunicação master/slave

Colocar/Receber caracter e string na UART

Funções que colocam/recebem caracteres/string na UART.

A função *putcharUART* (Figura 9) tem como tarefa guardar um caractere no registo *U2TXREG*.

```
/*********************************************/
/*Nome: putChar_UART ************************/
/*Função: Por um caracter na UART ***********/
/*********************************************/
void putChar_UART(char c){
    U2TXREG = c;
    atraso(2000);
}
```

Figura 7 - Função putcharUART

Analogamente à função *putcharUART* (Figura 9) a função *getcharUART* (Figura 10), verifica o estado do buffer de receção onde são armazenados os caracteres recebidos pela UART. Para saber se este buffer possui informação vamos verificar o buffer *URXDA* que se estiver ativado a 1, significa que o buffer de receção ainda tem dados para serem lidos. Quando é detetado um caracter esse é guardado no registo *U2RXREG*.

```
/*********************************************/
/*Nome: getCharUART *************************/
/*Função: Tirar um caracter na UART *********/
/*********************************************/
char getChar_UART(){

    while (!U2STAbits.URXDA); //verifica se ha chars para ler
    IFS1bits.U2RXIF = 0; // flag de interrupt de transmissão igual a zero

    return U2RXREG;
}
```

Figura 8 - Função getcharUART

Com o propósito de enviar mais do que um caracter para a porta serie, a função *putsrigUART* (Figura 11) permite esse envio. Nesta, a string a enviar vai ser enviada caracter a caracter para a UART através da função *putcharUART* (Figura 9).

```
/*****************************************/
/*Nome: putStringUART ********************/
/*Função: Por uma string na UART **********/
/*****************************************/
void putString_UART(char *string){    //*string -> apontador

    int i = 0;
    int length = strlen(string);  //strlen = tamanho da string

    for (i = 0; i < length; i++){
        putChar_UART(*string);   // envia o caracter
        *string++;// avança no apontador da string
    }
}
```

Figura 9 - Função putStringUART

Motores

Para controlar os motores temos de enviar para o slave certos comandos que os vão ativar ou desligar.

Comando	Acção
0x51	Liga Motor 1
0x52	Liga Motor 2
0x53	Para Motor 1
0x54	Para Motor 2

Tabela 2 - Comando dos Motores 1 e 2.

Comando	Acção
0x61	Anda para lado do armazém
0x62	Para os motores

Tabela 3 - Comando do Motor 3.

Para enviar estes comandos utilizamos a função talk_to_slave. Os comandos especificados nas tabelas são os que colocamos como parâmetro. Deixamos o stat_receive a 0 visto que não queremos resposta dos motores.

```
/*****************************************************/
/*Nome: Motor1 ***********************************/
/*Função: Consoante a variável de entrada, 1 ou 0, liga ou desliga o motor 1
/*****************************************************/
void Motor1(int estado) {
    putString_UART("Motor 1 ativo!\r\n");
    if (estado == 1)    talk_to_slave(0x0C, 0x51, 0, 1);  //liga
    else talk_to_slave(0x0C, 0x53, 0, 1);  //desliga
}

/*****************************************************/
/*Nome: Motor2 ***********************************/
/*Função: Consoante a variável de entrada, 1 ou 0, liga ou desliga o motor 2
/*****************************************************/
void Motor2(int estado) {
    putString_UART("Motor 2 ativo!\r\n");
    if (estado == 1)    talk_to_slave(0x0C, 0x52, 0, 1); //liga
    else talk_to_slave(0x0C, 0x54, 0, 1);  //desliga
}

/*****************************************************/
/*Nome: Motor3 ***********************************/
/*Função: Consoante a variável de entrada, 1 ou 0, liga ou desliga o motor 3
/*****************************************************/
void Motor3(int estado) {
    putString_UART("Motor 3 ativo!\r\n");
    if (estado == 1)    talk_to_slave(0x0D, 0x61, 0, 1); //liga
    else talk_to_slave(0x0D, 0x62, 0, 1); //desliga
}
```

Figura 10 - Funções dos motores 1, 2 e 3.

Sensores e LED's

Temos ainda comandos específicos para ativação de LEDS e tratamento de sensores.

Comando	Acção
0x00	Erro
0x81	Liga os Leds Verdes
0x82	Liga os Leds Vermelhos
0x83	Liga os Leds Amarelos
0x84	Desliga os Leds

Tabela 4 - Comandos para Panic Buttons e Leds.

Com os comandos da tabela podemos controlar 3 LEDs de 3 cores diferentes

```
/*****************************************************************************************************/
/*Nome: LEDS *****************************************************************************************/
/*Função: Consoante a variável de entrada, G, Y, R ou N liga o LED verde, amarelo, vermelho ou desliga os leds
/*****************************************************************************************************/
void LEDS(char cor) {
    putString_UART("Actua LEDs.\r\n");
    if(cor == 'G' || cor == 'g'){ //Liga leds verdes
        putString_UART("Green light.\r\n");
        talk_to_slave(0x1A, 0x81, 0, 1);
    }
    else if (cor == 'R' || cor == 'r') { //Liga leds vermelhos
        putString_UART("Red light.\r\n");
        talk_to_slave(0x1A, 0x82, 0, 1);
    }
    else if (cor == 'Y' || cor == 'y') { //Liga leds amarelos
        putString_UART("Yellow light.\r\n");
        talk_to_slave(0x1A, 0x83, 0, 1);
    }
    else if (cor == 'N' || cor == 'n') { //Desliga leds
        putString_UART("Turn of lights.\r\n");
        talk_to_slave(0x1A, 0x84, 0, 1);
    }
    else if (cor == 'E' || cor == 'e') { //Error
        putString_UART("Lights Error.\r\n");
        talk_to_slave(0x1A, 0x00, 0, 1);
    }
}
```

Figura 11 - Função para ligar e desligar LED's

É enviado um pedido com o valor 0x30 ao sensor de Hall para devolver o valor que esta a retirar no momento, este devolve uma mensagem com 2 bytes.

```
/*********************************************************************/
/*Nome: hall_sensor ***************************************************/
/*Função: Retorna o valor lido pelo sensor de hall ********************/
/*********************************************************************/
int sensor_Hall() {
    int treshold = 500;
    putString_UART("Hall sensor...\r\n");
    if(talk_to_slave(0x0E, 0x30, 1, 2) < treshold)  return 1; //peça detectada
    else if (talk_to_slave(0x0E, 0x30, 1, 2) > treshold) return 0; //peça n detectada
    else return 99;//error
}
```

Figura 12 - Função para ler o sensor de hall

Ultrason	Valor
Erro	0x00
Peça Identificada	0x1A
Peça Não Identificada	0x1B

Tabela 5 - Tabela com os valores que representa se detetou peça.

O mesmo se aplica ao sensor de Ultrasom, enviamos estes comandos com a função talk_to_slave. Os comandos especificados nas tabelas são os que colocamos como parâmetro. Deixamos o stat_receive a 1 e vamos ter como retorno 0x1A no caso de haver uma peça no tapete e 0x1B caso não consiga identificar nenhuma peça.

```
/*********************************************************************/
/*Nome: sensor_UltraSom ***********************************************/
/*Função: Retorna o valor lido pelo sensor de ultrasom ****************/
/*********************************************************************/
int sensor_UltraSom() {
    putString_UART("Sensor Ultrasom...\r\n");
    if (talk_to_slave(0x0A, 0x10, 1, 1) == 26) return 1; //peça detectada
    else if (talk_to_slave(0x0A, 0x10, 1, 1) == 27) return 0; //peça não detectada
    else return 99; //error
}
```

Figura 13 - Função para ler o sensor de ultra som

Sensor Indutivo	Valor
Erro	0x00
Porta Fechada	0x2A
Porta Aberta	0x2B

Tabela 6 - Tabela com os valores que representam o estado da porta.

No sensor indutivo o processo é semelhante ao do sensor Ultrasom, apenas sendo diferente os comandos a ser enviados.

```
/*********************************************************************/
/*Nome: sensor_indutivo ***************************************/
/*Função: Retorna o valor lido pelo sensor indutivo*****************/
/*********************************************************************/
int sensor_indutivo() {
    putString_UART("Sensor Indutivo...\r\n");
    if (talk_to_slave(0x0F, 0x20, 1, 1) == 42) return 1; //porta fechada
    else if (talk_to_slave(0x0F, 0x20, 1, 1) == 43) return 0; //porta aberta
    else return 99; //error
}
```

Figura 14 - Função para ler o sensor indutivo

Sensor de temperatura

Nesta função é definido qual o pino, na placa de aquisição de dados onde será ligado o sensor de temperatura, que no caso apresentado corresponde ao pino RB2. Essa atribuição é feita através do registo AD1CHS, registo responsável pela seleção dos canais de input/output.

```
/*******************************************/
/*Nome: sensor_temperatura ****************/
/*Função: Retorna o valor da temperatura **/
/*******************************************/
int sensor_temperatura() {
    AD1CHS = TEMPERATURE; // Sensor de temperatura ligado a AN2
    AD1CON1bits.SAMP = 1; // start sampling
    atraso(2000);
    AD1CON1bits.SAMP = 0; // end sampling
    while (!AD1CON1bits.DONE); // wait for conversion to br completed
    return ADC1BUF0; // get ADC value  (1-1024)
}
```

Figura 15 - Função que lê o valor da temperatura.

Paragem de emergência e alarme de fogo

Temos ainda comandos para estados de emergência onde podemos controlar um alarme e funções de paragem de linha montagem.

Comando	Acção
0x00	Erro
0x71	Liga Alarme
0x72	Desliga Alarme

Tabela 7 - Tabela com comandos da Sirene de Alarme.

Se quisermos ativar o alarme, para o caso onde a temperatura passou o limite definido, enviamos para o slave o comando 0x71 e 0x72 se o quisermos desligar. Quando estamos no estado 2 (estado de emergência) queremos parar todos os motores e acender os LEDs vermelhos e por isso utilizamos a função Motorx(0) que os vai desativar e LEDS('R') para acender os LEDs vermelhos. No caso do estado 1 (porta aberta) queremos parar todos os motores e por os LEDs a amarelo por isso utilizamos as funções Motorx(0) e LEDS('Y').

```
/*****************************************************************************/
/*Nome: buzzer ***************************************************************/
/*Função: Consoante a variável de entrada, 1 ou 0, liga ou desliga o buzzer ********/
/*****************************************************************************/
void buzzer(int estado) { //Activa a sirene de alarme
    putString_UART("DANGER! Get to chopper!\r\n");
    if (estado == 1)    talk_to_slave(0x1B, 0x71, 0, 1);
    else if (estado == 2) talk_to_slave(0x1B, 0x72, 0, 1);
    else talk_to_slave(0x1B, 0x00, 0, 1);
}

/*************************************************************/
/*Nome: Emergency_Stop*****************************************/
/*Função: Para todos os motores e mete led amarelo ********/
/*************************************************************/
void Emergency_Stop() {
    Motor1(0);
    Motor2(0);
    Motor3(0);
    LEDS('Y');
    putString_UART("Emergency Stop! All motors deactivated.\r\n");
}

/*************************************************************/
/*Nome: Fire_Alarm *********************************************/
/*Função: Para todos os motores e mete led vermelho *******/
/*************************************************************/
void Fire_Alarm() {
    Motor1(0);
    Motor2(0);
    Motor3(0);
    LEDS('R');
    putString_UART("Fire detected! Get to the chopper!\r\n");
}
```

Figura 16 - Funções de emergência

Loop do Main

Na função Main tratamos dos valores dos sensores e determinamos qual o estado do sistema e qual deve ser o seu comportamento.

```
//Leitura dos sensores
Room_temp = sensor_temperatura();
total_peca = peca_1 + peca_2;


if (Room_temp/10 >= temperature_threshold) { //FIRE
    Fire_Alarm();
    buzzer(1);
    while (Room_temp / 10 >= temperature_threshold);
    LEDS('G');
    buzzer(0);
}
else if(!sensor_indutivo()){ //Paragem de emergencia
    Emergency_Stop();
    while (!sensor_indutivo());
    LEDS('G');
}
else if (sensor_UltraSom() && pecaCurso ==0) { //Peça detetada no sensor de Ultra Som
    Motor1(1);
    putString_UART("Motor 1 ativo.\r\n");
    peca_1 = peca_1 + 1;
    pecaCurso = 1;
}
else if (sensor_Hall() && pecaCurso ==0) { //Peça detetada no sensor de Hall
    Motor2(1);
    putString_UART("Motor 2 ativo.\r\n");
    peca_2 = peca_2 + 1;
    pecaCurso = 1;
}
else if(fimdocurso() && pecaCurso==1){
    //Motores laterais desativados
    Motor1(0);
    putString_UART("Motor 1 desativado\r\n");
    Motor2(0);
    putString_UART("Motor 2 desativado\r\n");
    //Motor armazem ativo
    Motor3(1);
    putString_UART("Motor 3 ativo\r\n");
    atraso(300000);
    Motor3(0);
    putString_UART("Motor 3 desativado\r\n");
    pecaCurso = 0;
```

Figura 17 - Função principal do main.

Conclusão

Concluiu-se que o protocolo I2C possui uma enorme versatilidade e utilidade na comunicação entre diferentes sensores e atuadores. A metodologia Master/Slave introduz um excelente auxílio na coordenação entre diferentes tarefas necessárias de realizar, acabando por dessa forma fazer uma maior distribuição nas tarefas dos controladores.

Assim consegue-se possuir uma arquitetura em que se define um microcontrolador mais centralizado que permite distribuir e coordenar as tarefas necessárias para a realização do trabalho, enquanto que conseguimos por outro lado manter vários microcontroladores em separado para dedicarem o seu poder de processamento a outras tarefas distintas mais especificas.

Isto conduz a que a recolha de informação e a sua distribuição acabe por ser simplificada, o que corresponde a um maior controlo do sistema permitindo assim desenvolver aplicações mais complexas.