Trabalho prático N.º 10

Objetivos

- Compreender os mecanismos básicos que envolvem a comunicação série usando o protocolo I²C.
- Implementar funções básicas de comunicação série através do módulo I²C do PIC32 e utilizar essas funções para interagir com um sensor de temperatura com interface I²C.

Introdução

A interface I²C (acrónimo de *Inter-Integrated Circuit*) é uma interface de comunicação série bidirecional *half-duplex* pensada para interligação, a pequenas distâncias, entre microcontroladores e dispositivos periféricos. O PIC32, na versão usada na placa DETPIC32, disponibiliza 4 módulos I²C[1] que podem operar, cada um deles, como um dispositivo *slave*, como um dispositivo *master* num sistema com um único *master*, ou ainda como um dispositivo *master/slave* num sistema *multi-master*. Neste trabalho prático apenas iremos explorar a sua função como *master* num sistema com um único *master*. A Figura 1 apresenta o diagrama de blocos simplificado do módulo I²C do PIC32 onde se pretende, sobretudo, identificar os registos do modelo de programação e a sua função na perspectiva de utilização do módulo como único *master*.

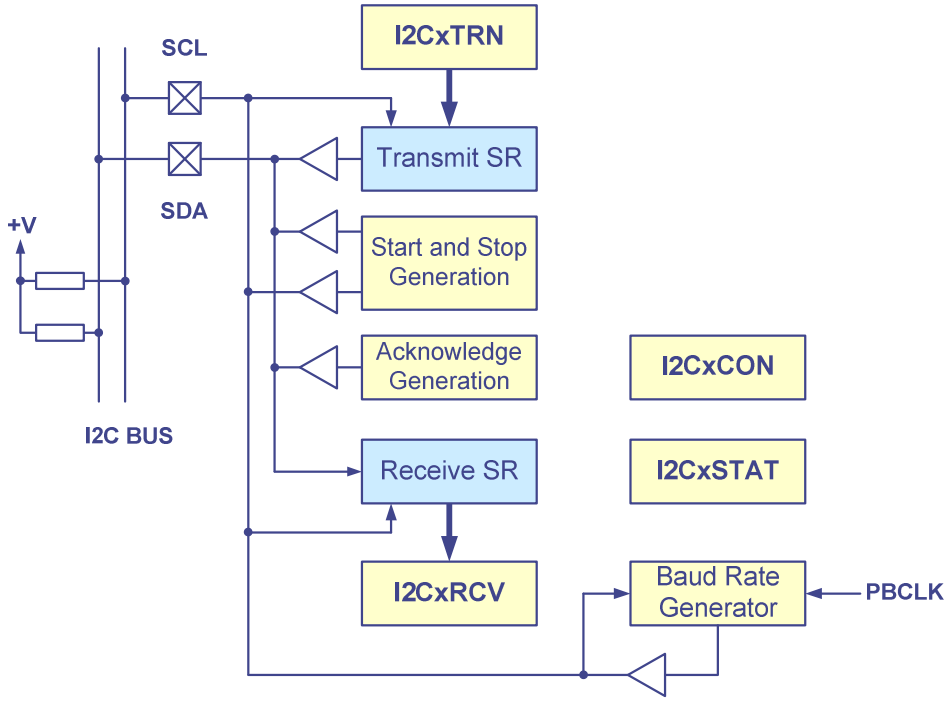

Figura 1. Diagrama de blocos simplificado do módulo I2C (*master*).

Os blocos-base do módulo I²C são dois *shift registers,* o *Transmit* SR e o *Receive* SR que fazem a conversão paralelo-série, e série-paralelo, respetivamente. Numa operação de transmissão, o *Transmit* SR envia para a linha **SDA** o *byte* armazenado no registo **`I2CxTRN`**. Por seu lado, numa operação de receção, o *byte* recebido no *Receive* SR é copiado para o registo **`I2CxRCV`**.

O módulo I²C limita-se a implementar as funções básicas que permitem a comunicação de acordo com o *standard*. A sequência de ações a realizar para a comunicação com um *slave* I²C é, assim, integralmente efetuada por *software*. Por exemplo, uma sequência de comunicação com um *slave* envolve, no início, o envio de um *start*, seguido de 8 bits com o endereço do

[1] Esses 4 módulos são numerados pelo fabricante de 1 a 5: 1, 3, 4 e 5.

slave e a operação a realizar (**RD/WR**). Esta é também a sequência de operações a realizar no programa: envio do *start* através do registo **I2CxCON**, envio do *byte* com o endereço e a operação a realizar através do registo **I2CxTRN**.

O *Baud Rate Generator* é usado, na situação em que o módulo I²C é *master*, para estabelecer a frequência do relógio na linha **SCL**. De acordo com a norma, esta frequência pode ser 100 kHz, 400 kHz ou 1 MHz, devendo ser estabelecida em função de cada um dos *slaves* com que o *master* vai comunicar.

Após *power-on* ou *reset* todos os 4 módulos I²C estão inactivos. A ativação é efetuada através do bit **ON** do registo **I2CxCON**. A ativação de um dado módulo I²C configura automaticamente os pinos correspondentes do PIC32 como *open-drain*, sobrepondo-se esta configuração à efetuada através do(s) registo(s) **TRISx**.

Gerador de *baudrate*

Como já referido anteriormente, o gerador de *baudrate* funciona como um gerador do relógio que é enviado para a linha **SCL**. A implementação deste gerador baseia-se num contador decrescente de 12 bits, em que o sinal de relógio é o **PBCLK** (*Peripheral Bus Clock*) cuja frequência é, na placa DETPIC32, 20 MHz. O contador tem uma entrada de *load* síncrona, que permite carregar o valor inicial de contagem. Adicionalmente, o contador tem uma saída de *terminal count* que fica ativa sempre que o contador atinge o valor 0. Uma vez atingido o valor 0, o contador só retoma o funcionamento quando for efetuado o *reload* do valor inicial. O valor incial de contagem é armazenado no registo **I2CxBRG**, registo este que assim determina a frequência-base do relógio na linha **SCL**. A Figura 2 apresenta um diagrama simplificado do gerador de *baudrate*, que não toma em consideração as questões relacionadas com a sincronização do relógio (*clock stretching*).

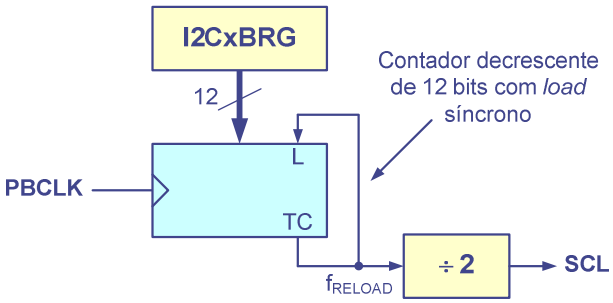

Figura 2. Diagrama de blocos simplificado do gerador de *baudrate*.

O facto de o contador efetuar o *reload* do valor inicial de forma síncrona implica que a combinação 0 (que provoca a ativação da saída **TC**) esteja fixa durante 1 ciclo de relógio do sinal **PBCLK** (sem *clock stretching*). Sendo assim, e de forma análoga ao já discutido para os *timers*, a frequência do sinal na saída **TC** (f_{RELOAD}) é dada por:

$$f_{RELOAD} = f_{PBCLK} / (I2CxBRG + 1)$$

A frequência do sinal à saída do contador é posteriormente dividida por 2 (utilizando, por exemplo, um *flip-flop* tipo T), de modo a obter-se um sinal com um *duty-cycle* de 50%. Assim, a frequência do sinal de relógio na linha SCL é:

$$f_{SCL} = f_{PBCLK} / (2 * (I2CxBRG + 1))$$

O valor da constante a colocar no registo **I2CxBRG** fica então:

$I2CxBRG = f_{PBCLK} / (2 * f_{SCL}) - 1$, ou melhor ainda (com arredondamento):

$$I2CxBRG = (f_{PBCLK} + f_{SCL}) / (2 * f_{SCL}) - 1$$

Configuração do módulo I²C

Para a configuração do módulo I²C bastam duas ações:

- Configuração do gerador de *baudrate*: cálculo da constante **I2CxBRG** e escrita no respetivo registo.
- Activação do módulo I²C (bit **ON** do registo **I2CxCON**).

Programação com o módulo I²C

O módulo I²C suporta as funções básicas de comunicação através de geradores de *start* e de *stop*, transmissão de um *byte*, receção de um *byte* e geração de *acknowledge*. Em termos gerais, a programação de cada um dos passos do protocolo consiste em duas operações básicas: 1) escrita de um registo (de dados ou de controlo em função do passo específico); 2) espera até que a operação se realize (*polling* de um bit ou conjunto de bits de um registo). Este procedimento tem que ser seguido em todas as operações básicas, uma vez que o módulo I²C não admite o começo de uma nova operação sem que a anterior tenha terminado. Por exemplo, não é possível enviar um *start* e, logo de seguida escrever no registo **I2CxTRN** para iniciar a transmissão de um *byte*, sem esperar que a operação de *start* termine. Se esta regra não for seguida, o módulo ignora, neste caso, a escrita no registo **I2CxTRN**, e ativa o bit **IWCOL** (*write collision detect bit*) do registo **I2CxSTAT**.

Geração do *start*

Para iniciar um evento de *start* activa-se o bit **SEN** (*start enable bit*) do registo **I2CxCON**. Por *hardware*, este bit passa automaticamente a zero quando a operação é completada. Este bit pode, assim, ser usado, por *polling*, para determinar a passagem para a operação seguinte.

Transmissão de um *byte* para um *slave*

A transmissão de um *byte* de dados ou de um endereço para um *slave* é efetuada escrevendo o valor a enviar no registo **I2CxTRN**. O envio desse valor demora 8 ciclos de relógio (**SCL**) e no 9º ciclo o *master* lê da linha o valor do bit de *acknowledge* colocado pelo *slave*. Esse valor é colocado no bit **ACKSTAT** do registo **I2CxSTAT**. Para garantir que esta sequência chegou ao fim, e assim passar para outra operação, pode fazer-se o *polling* do bit **TRSTAT** (*transmission status bit*) do registo **I2CxSTAT**: enquanto a sequência decorre esse bit está a **1**, passando a **0** após o 9º ciclo de relógio, isto é, após a receção do bit de *acknowledge*.

O bit **ACKSTAT** reflecte directamente o valor recebido da linha (que representa **ACK**): esse bit está a **0** se o *slave* recebeu corretamente o valor e fica a **1**, no caso contrário.

Receção de um *byte* transmitido por um *slave*

O *master* pode receber informação do *slave* após ter enviado uma sequência com o endereço e o bit **R/W** a **1**. Para isso, o *master* tem que ativar o bit **RCEN** (*receive enable bit*) do registo **I2CxCON**. No entanto, antes de ativar esse bit, é necessário garantir, de acordo com as indicações do fabricante, que os 5 bits menos significativos do registo **I2CxCON** são todos 0 (ver manual do I²C do PIC32).

Após a ativação do bit **RCEN**, o *master* inicia a geração do relógio na linha SCL e, após 8 ciclos desse relógio, o *shift-register* de receção do *master* recebeu o valor enviado pelo *slave.* Após o 8º ciclo de relógio o bit **RCEN** é automaticamente desativado, o *byte* recebido no *shift-register* é copiado para o registo **I2CxRCV** e o bit **RBF** (*receive buffer full status bit*) do registo **I2CxSTAT** é ativado. Este bit pode ser usado por *polling* para esperar que o *byte* seja recebido.

Transmissão do *acknowledge*

Após a receção de um *byte*, o *master* tem que transmitir uma sequência de *acknowledge* (**ACK**): transmite **0** (**ACK\=0**), no caso em que pretende continuar a ler informação do *slave*; transmite **1** (**ACK\=1**, i.e., **NACK**), no caso em que pretende terminar a comunicação.

O bit **ACKDT** (*acknowledge data bit*) do registo **I2CxCON** permite especificar **ACK** (**0**) ou **NACK** (**1**). O bit **ACKEN** (*acknowledge sequence enable bit*), quando ativado, inicia a sequência de *acknowledge*. Este bit é automaticamente desativado pelo *hardware* quando o *master* termina o envio da sequência de *acknowledge*, pelo que pode ser usado, por *polling*, para determinar a passagem para a operação seguinte.

Geração do *stop*

Antes de se iniciar um evento de *stop* é necessário garantir que os 5 bits menos significativos do registo **I2CxCON** são todos 0. Quando essa condição é verificada pode então gerar-se um evento de *stop* através da ativação do bit **PEN** (*stop enable bit*) do registo **I2CxCON**. À semelhança do referido para os restantes eventos, deve também esperar-se que a ação *stop* termine antes de prosseguir com outro qualquer evento. Para isso basta esperar que o bit **PEN** passe ao nível lógico 0, uma vez que ele é automaticamente colocado a 0 pelo *hardware* quando a ação termina.

Sensor de temperatura TC74

O circuito integrado TC74 da Microchip é um sensor digital de temperatura com interface série I^2C. O elemento sensorial integrado no dispositivo permite a medida de temperatura na gama -65ºC a 125ºC, com uma precisão de ±2ºC na gama 25ºC a 85ºC. O valor da temperatura é disponibilizado num registo interno de 8 bits do sensor, e é codificado em complemento para 2.

O endereço do dispositivo, para efeitos da sua interligação num barramento I^2C, é fixado pelo fabricante, havendo no mercado versões do mesmo sensor com 7 endereços distintos. O sensor disponível para ser usado nestas aulas tem o endereço **0x4D** (endereço de 7 bits – ver página 9 do manual do sensor de temperatura).

O sensor de temperatura tem internamente dois registos: o já mencionado registo de temperatura (designado por registo **TEMP**), que apenas pode ser lido e o registo de configuração (designado por registo **CONFIG**) que pode ser lido ou escrito. Para o acesso a estes dois registos são disponibilizados dois comandos: o comando **RTR** (*read temperature*) que permite a leitura do registo **TEMP** e o comando **RWCR** (*read/write configuration*) que permite a escrita ou a leitura do registo **CONFIG**. A Figura 3 apresenta o protocolo I^2C para a leitura de um registo interno do sensor de temperatura (ver página 7 do manual do sensor).

Read Byte Format

S	Address	WR	ACK	Command	ACK	S	Address	RD	ACK	Data	NACK	P
	7 Bits			8 Bits			7 Bits			8 Bits		

Slave Address

Command Byte: selects which register you are reading from.

Slave Address: repeated due to change in data-flow direction.

Data Byte: reads from the register set by the command byte.

Figura 3. Protocolo I^2C para leitura de 1 *byte* de um registo interno do sensor de temperatura.

A sequência de ações que o programa tem que efetuar para a leitura de um registo do sensor é a que resulta da descrição do protocolo apresentada na figura anterior. Se se pretender ler o valor da temperatura, a sequência de ações a realizar é:

1) Enviar um *start.*
2) Enviar um *byte* com o endereço do sensor e com a indicação de uma operação de escrita (**`R/W\ = 0`**).
3) Esperar pela receção do *acknowledge* do sensor.
4) Enviar um *byte* com a identificação do comando **RTR** (valor **`0x00`** – ver manual).
5) Esperar pela receção do *acknowledge* do sensor.
6) Enviar um *start* – este novo *start* tem que ser enviado porque a operação que se vai especificar de seguida é diferente da anterior (a anterior foi uma escrita, a seguinte vai ser uma leitura).
7) Enviar um *byte* com o endereço do sensor e com a indicação de uma operação de leitura (**`R/W\ = 1`**).
8) Esperar pela receção do *acknowledge* do sensor.
9) Ativar o bloco de receção do *master*.
10) Esperar que o *master* receba o *byte* do *slave*.
11) Enviar um *not acknowledge* (**NACK**) sinalizando-se desse modo o *slave* que o *master* não pretende continuar a ler.
12) Enviar um *stop*.

Trabalho a realizar

Parte I

1. Monte, na placa branca, o sensor de temperatura TC74, de acordo com a figura seguinte. Os sinais **`SDA1`** (**`RD9`**) e **`SCL1`** (**`RD10`**) correspondem à ligação ao módulo I²C número 1 do PIC32[2].

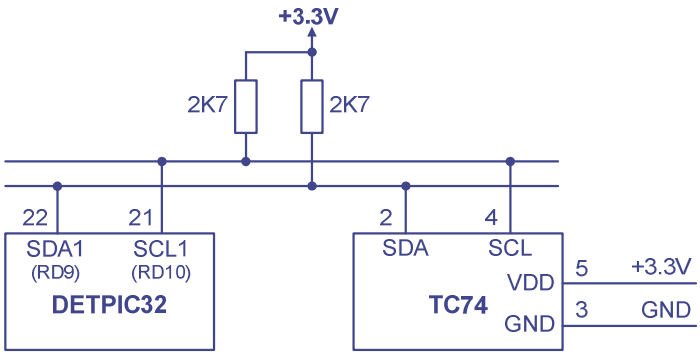

Figura 4. Ligação do sensor de temperatura TC74 à placa DETPIC32.

   Use o programa "**`testeI2C.hex`**", disponível na página de elearning da UC, para testar a correta ligação do sensor. Caso este esteja corretamente ligado o programa vai imprimir, a cada segundo, o valor lido da temperatura. Se isso não acontecer verifique as ligações.

2. A abordagem que vamos seguir neste trabalho prático é a de implementar funções específicas para cada um dos eventos do protocolo I²C. Tendo essas funções-base implementadas, a comunicação com um *slave* resume-se à evocação dessas funções pela ordem adequada, de acordo com a operação que se pretende realizar. Assim, comece por escrever uma função para inicialização do módulo I²C:

```
void i2c1_init(unsigned int clock_freq)
{
   // Config baudrate generator (see introduction for details)
   // Enable I2C1 module
}
```

[2] O PIC32 disponibiliza 4 módulos I²C iguais. No entanto, devido a um problema de *hardware* reportado pela Microchip no documento "PIC32MX7XX Family Errata", os módulos 4 e 5 requerem um procedimento de configuração inicial diferente dos restantes dois.

3. Implemente a função para gerar o evento de *start*. Essa função deve implementar a seguinte sequência de ações: 1) ativar o *start*; 3) esperar que o evento termine.

```
void i2c1_start(void)
{
   // Activate Start event (I2C1CON, bit SEN)
   // Wait for completion of the Start event (I2C1CON, bit SEN)
}
```

4. Implemente a função para gerar o evento de *stop*. Essa função deve implementar a seguinte sequência de ações: 1) assegurar-se que os 5 bits menos significativos do registo **I2CxCON** são todos 0; 2) ativar o *stop*; 3) esperar que o evento termine.

```
void i2c1_stop(void)
{
   // Wait until the lower 5 bits of I2CxCON are all 0 (the lower 5 bits
   //   of I2CxCON must be 0 before attempting to set the PEN bit)
   // Activate Stop event (I2C1CON, bit PEN)
   // Wait for completion of the Stop event (I2C1CON, bit PEN)
}
```

5. Implemente a função para transmitir um *byte*. Essa função deve implementar a seguinte sequência de ações: 1) copiar para o registo **I2C1TRN** o *byte* a transmitir; 2) esperar que o *byte* seja transmitido e o *acknowledge* recebido; 3) retornar o valor de *acknowledge* recebido.

```
int i2c1_send(unsigned char value)
{
   // Copy "value" to I2C1TRN register
   // Wait while master transmission is in progress (8 bits + ACK\)
   //    (I2C1STAT, bit TRSTAT – transmit status bit)
   // Return acknowledge status bit (I2C1STAT, bit ACKSTAT)
}
```

6. Implemente a função para receber um *byte*. Essa função deve implementar a seguinte sequência de ações: 1) esperar que os 5 bits menos significativos do registo **I2C1CON** sejam todos **0**; 2) ativar o bit **RCEN** (registo **I2C1CON**); 3) esperar que o *byte* seja recebido (**I2C1STAT** bit **RBF**); 4) esperar que os 5 bits menos significativos do registo **I2C1CON** sejam todos **0**; 5) enviar **ACK** ou **NACK** e esperar que a sequência termine; 5) retornar o valor do registo **I2C1RCV**.

```
char i2c1_receive(char ack_bit)
{
   // Wait util the lower 5 bits of I2C1CON are all 0 (the lower 5 bits
   //   of I2C1CON must be 0 before attempting to set the RCEN bit)
   // Activate RCEN bit (receive enable bit, I2C1CON register)
   // Wait while byte not received (I2C1STAT, bit RBF – receive buffer
   //   full status bit)
   // Send ACK / NACK bit. For that:
   //   1. Copy "ack_bit" to I2C1CON, bit ACKDT (be sure "ack_bit" value
   //      is only 0 or 1)
   //   2. Wait until the lower 5 bits of I2C1CON are all 0 (the lower
   //      5 bits of I2C1CON must be 0 before attempting to
   //      set the ACKEN bit.
   //   3. Start Acknowledge sequence (I2C1CON register, bit ACKEN=1)
   // Wait for completion of Acknowledge sequence (I2C1CON, bit ACKEN)
   // Return received value (I2C1RCV)
}
```

7. Até ao exercício anterior implementámos as funções necessárias para lidar com todos os eventos I²C necessários para estabelecer a comunicação com um *slave*. A comunicação com o sensor para a leitura do valor da temperatura resume-se à evocação das funções anteriores pela ordem definida no protocolo. Assim, implemente a função **`main()`** para ler 4 valores de temperatura por segundo – os valores lidos devem ser impressos no ecrã usando *system calls* (verifique na Figura 3 a sequência de ações a realizar). Poderá utilizar sequências de *escape* (ver parte final do ficheiro "**`detpic32.h`**") para melhorar a aparência (por exemplo, impressos sempre no mesmo sítio) dos valores impressos no ecrã.

```
#define  I2C_READ       1
#define  I2C_WRITE      0
#define  I2C_ACK        0
#define  I2C_NACK       1

#define  SENS_ADDRESS   0x4D    // device dependent
#define  ADDR_WR        ((SENS_ADDRESS << 1) | I2C_WRITE)
#define  ADDR_RD        ((SENS_ADDRESS << 1) | I2C_READ)
#define  TC74_CLK_FREQ  100000  // 100 KHz
#define  RTR            0       // Read temperature command

int main(void)
{
   int ack, temperature;
   i2c1_init(TC74_CLK_FREQ);
   while(1)
   {
      // Send Start event
      // Send Address + WR (ADDR_WR); copy return value to "ack" variable
      // Send Command (RTR); add return value to "ack" variable
      // Send Start event (again)
      // Send Address + RD (ADDR_RD); add return value to "ack" variable
      // Test "ack" variable; if "ack" != 0 then an error has occurred;
      //     send the Stop event, print an error message and exit loop
      // Receive a value from slave (send NACK as argument); copy
      //     received value to "temperature" variable
      // Send Stop event
      // Print "temperature" variable (syscall printInt10)
      // Wait 250 ms
   }
}
```

Parte II

1. Neste exercício pretende-se a reorganização do código que escreveu anteriormente, agrupando as funções-base de interação com o módulo I²C numa biblioteca de funções genérica que possa facilmente ser incluída num projeto de maior dimensão. Assim:
   a) Construa o ficheiro "**`i2c.h`**", com a estrutura-base que a seguir se apresenta, e onde estejam declarados os protótipos de todas as funções de interface com o módulo I²C que desenvolveu na parte 1 deste trabalho prático, bem como todos os símbolos gerais que tenham que ser usados (**`I2C_READ`**, **`I2C_WRITE`**, ...).

```
// i2c.h
#ifndef _I2C_H
#define _I2C_H

// Declare symbols here (READ, WRITE, ...)
#define  I2C_READ     1
...
// Declare function prototypes here
void i2c1_init(unsigned int clock_freq);
...
#endif
```

b) Construa o ficheiro "**i2c.c**" com o código de cada uma das funções de interação com o módulo I²C (**i2c1_init()**, **i2c1_start()**, …).

```
// i2c.c
#include <detpic32.h>
#include "i2c.h"

(...)
```

2. Escreva, num novo ficheiro (por exemplo **guiao10.c**), uma função que faça a leitura de um valor de temperatura do sensor, usando as funções de comunicação que desenvolveu anteriormente e que agora devem estar disponíveis no ficheiro "**i2c.c**". Escreva a função **main()** para teste desta função, de modo a fazer 4 leituras por segundo e a imprimir no ecrã o respectivo valor de temperatura (utilize *system calls*).

```
#include <detpic32.h>
#include "i2c.h"

int getTemperature(int *temperature)
{
   int ack;
   // Send Start event
   // Send Address + WR (ADDR_WR) and copy return value to "ack" variable
   ...
   (see exercise 7, function main())
   ...
   // Send Stop event
   return ack;
}

int main(void)
{
   ...
}
```

Nota:

Para compilar o projeto com mais do que um ficheiro pode usar o *script* "**pcompile**" colocando a lista dos ficheiros que fazem parte do projeto separados por um espaço. Exemplo:

```
pcompile guiao10.c i2c.c
```

O comando anterior compila os dois ficheiros e, se não houver erros de sintaxe, gera um ficheiro com o nome do primeiro ficheiro substituindo a extensão "**.c**" pela extensão "**.hex**". Para o exemplo acima, o ficheiro de saída seria "**guiao10.hex**".

3. Retome agora o código que desenvolveu no último exercício do trabalho prático n.º 7 e acrescente a biblioteca **i2c.c** e a função de leitura da temperatura que escreveu no exercício anterior. Faça as alterações ao programa que permitam mostrar nos dois *displays* o valor da temperatura, sempre que a combinação binária nos bits RB1 e RB0 seja "11". A leitura do sensor deverá ser feita 4 vezes por segundo, na RSI de um timer.

Elementos de apoio

- Slides das aulas teóricas.
- Tiny Serial Digital Thermal Sensor TC74 (disponível no site da disciplina).
- PIC32 Family Reference Manual, Section 24 – I2C.