Trabalho prático N.º 12

Objetivos

- Compreender os mecanismos básicos que envolvem a comunicação série usando o protocolo I²C.
- Utilizar funções I²C, disponibilizadas sob a forma de uma API, para interagir com um sensor de temperatura.
- Integrar o sensor de temperatura num sistema funcional, apresentando as leituras de temperatura num sistema de visualização com *displays* de 7 segmentos.

Introdução

A interface I²C (acrónimo de *Inter-Integrated Circuit*) é uma interface de comunicação série bidirecional *half-duplex* concebida para a interligação, a pequenas distâncias, entre microcontroladores e dispositivos periféricos. O PIC32, na versão utilizada na placa DETPIC32, disponibiliza 4 módulos I²C[1] que podem operar, individualmente, 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, o módulo I²C será utilizado apenas como *master* num sistema com um único *master*[2].

Sensor de temperatura TC74

O circuito integrado TC74 da Microchip, disponível na placa DETPIC32, é um sensor digital de temperatura com interface série I²C. 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 I²C, é fixado pelo fabricante, havendo no mercado versões do mesmo sensor com 7 endereços distintos. O sensor disponível na placa DETPIC32 para ser usado neste guião tem a referência **`TC74A0-3.3VCT`** (consulte o manual do sensor de temperatura para obter o respetivo endereço).

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 1 apresenta o protocolo I²C 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 1. Protocolo I²C para leitura de 1 *byte* de um registo interno do sensor de temperatura.

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

[2] No anexo a este guião pode encontrar uma descrição mais detalhada do funcionamento e programação do módulo I²C do PIC32.

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`**). A função de envio de um byte devolve o valor de *acknowledge que recebe* do *slave* (**`ACK`**: **`0`**, **`NACK`**: **`1`**).
3) Enviar um *byte* com a identificação do comando **`RTR`** (ver manual).
4) Enviar um *restart* – este novo *start* tem que ser enviado porque a operação que se vai especificar de seguida é diferente da anterior (a primeira operação foi uma escrita, a seguinte vai ser uma leitura).
5) Enviar um *byte* com o endereço do sensor e com a indicação de uma operação de leitura (**`R/W\ = 1`**).
6) Ler um *byte* do *slave*; a função tem como argumento de entrada o valor do *acknowledge* a enviar no final da leitura (**`0`**: **`ACK`**, **`1`**: **`NACK`**). Neste caso deve ser eviado *not acknowledge* (**`NACK`**) sinalizando-se desse modo o *slave* que o *master* não pretende continuar a ler.
7) Enviar um *stop*.

API para o módulo I²C do PIC32

Esta API (*Application Programming Interface*) foi desenvolvida para facilitar a utilização do módulo I2C integrado nos microcontroladores PIC32. A API fornece funções que permitem configurar o módulo I2C, bem como realizar operações básicas de comunicação, como iniciar ou terminar transmissões e enviar ou receber dados. Estas funções abstraem detalhes de baixo nível do hardware, permitindo que o programador se concentre na lógica da aplicação.

O código-fonte (disponível no moodle da UC) está organizado nos ficheiros **`pic32_i2c.c`** e **`pic32_i2c.h`**, onde estão definidas todas as funções necessárias para operar o módulo I²C do PIC32. A tabela abaixo apresenta os protótipos das funções disponíveis, juntamente com uma breve descrição das suas funcionalidades (pode encontrar uma descrição mais detalhada no ficheiro **`pic32_i2c.h`**):

Protótipos	Descrição
`void i2c1_init(int clock_freq);`	Inicializa o módulo I²C e configura a frequência de relógio. → Ver datasheet
`void i2c1_start(void);`	Envia o evento *start*.
`void i2c1_restart(void);`	Envia o evento *restart*.
`void i2c1_stop(void);`	Envia o evento *stop*.
`char i2c1_send(char value);`	Envia um byte (retorna o valor de *acknowledge* recebido do *slave*).
`char i2c1_receive(char *value, char ack);`	Recebe um byte e armazena-o no endereço apontado por "value"; após a leitura envia o valor de *ack* (acknowledge). Retorna 0 se operação bem sucedida, ou 1 se não foi possível fazer a leitura (*timeout*).

definir previamente a variável

continuar a receber

O programa que usa esta API pode ser compilado usando o *script* **`pcompile`**. Por exemplo, se o programa tiver o nome **`teste.c`**, a linha de commando a utilizar será:

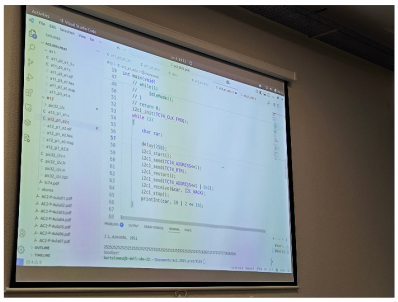
```
pcompile teste.c pic32_i2c.c
```

Este comando gera, entre outros, o ficheiro **`teste.hex`**, que poderá ser transferido para a placa DETPIC32.

Trabalho a realizar

Parte 1

O sensor de temperatura TC74 está montado na placa DETPIC32 de acordo com o esquema da figura seguinte (consulte o esquema da placa, disponível no guião prático nº 3). Os sinais **`SDA1`** (**`RD9`**) e **`SCL1`** (**`RD10`**) correspondem à ligação ao módulo I²C número 1 do PIC32[3].



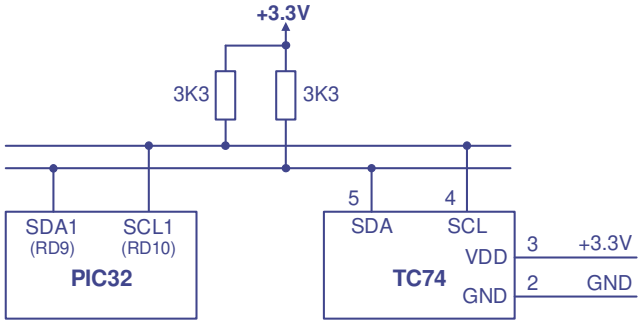


Figura 2. Ligação do sensor de temperatura TC74 ao PIC32.

1. Complete as definições em falta na listagem seguinte.

```
#include <detpic32.h>
#include "pic32_i2c.h"

#define  I2C_READ        ?
#define  I2C_WRITE       ?
#define  I2C_ACK         ?
#define  I2C_NACK        ?

#define  TC74_CLK_FREQ   ???     // Frequency in Hz (see TC74 datasheet)
#define  TC74_ADDRESS    0x??    // TC74A0 address (see TC74 datasheet)
#define  TC74_RTR        ?       // Read temperature command

int main(void)
{
   ...
   return 0;
}
```

2. Utilizando as funções disponíveis na API fornecida, complete a função **`main()`**, de modo a ler 4 valores de temperatura por segundo – os valores de temperatura lidos devem ser impressos no ecrã do PC, usando *system calls*. Utilize o *CoreTimer* para controlar a frequência de amostragem do sensor.

   O programa deve começar por inicializar o módulo I²C, com a frequência de relógio definida para o sensor. Para a leitura de um valor de temperatura deve realizar a sequência de ações que estão apresentadas na Figura 1 (conforme descrito no *datasheet* do sensor).

```
int main(void)
{
   i2c1_init(...);
   while(1)
   {
      ...
   }
   return 0;
}
```

[3] 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. Com base no código desenvolvido no exercício anterior, implemente uma função para realizar a leitura de um valor de temperatura do sensor TC74. A função deverá ter o seguinte protótipo:

```
int readTemperature(int *temp);
```

A função deverá:

- Enviar os comandos necessários via I²C para obter a temperatura do sensor, seguindo a sequência apresentada na Figura 1.
- Verificar o bit de *acknowledgment* (**`ACK`**) após cada byte enviado:
    - **`ACK=0`** indica sucesso.
    - **`ACK=1`** indica falha.
- Guardar o valor da temperatura no endereço apontado por **`temp`**, fornecido como argumento da função.
- Retornar **`0`** se todos os bytes enviados receberem **`ACK=0`** ou **`1`** caso algum receba **`ACK=1`**.

Após implementar a função **`readTempretaure()`**, teste-a, realizando 4 leituras por segundo e imprimindo os valores lidos.

Nota: o aumento, ou a diminuição, de temperatura pode ser feito, de forma artificial, somando um *offset* ao valor real lido. Para isso pode usar o *system call* **`inkey()`** para aumentar ou diminuir o valor desse *offset*.

Parte 2

1. Acrescente ao seu programa o código necessário para mostrar nos dois *displays* de 7 segmentos da placa o valor da temperatura medida pelo sensor. Para isso, considere:
    - A frequência de amostragem do sensor deve ser de **5 leituras por segundo (5 Hz)**, controlada por interrupções geradas pelo *timer* **T1**.
    - A frequência de refrescamento dos *displays* deve ser de **60 Hz**, controlada por interrupções geradas pelo *timer* **T2**.

    Certifique-se de configurar os 2 *timers* corretamente para gerar interrupções com as frequências indicadas. Não se esqueça, igualmente, de configurar os portos de I/O associados ao sistema de visualização.
2. Acrescente ao programa a funcionalidade que permita configurar e detetar limites de temperatura mínima e máxima. Para isso:
    - Implemente duas variáveis globais para armazenar os limites.
    - Implemente a verificação dos limites em cada leitura de temperatura: caso a temperatura lida ultrapasse o limite superior ou fique abaixo do limite inferior ligue o LED **`D11`** (ligado ao porto **`RC14`**).
    - Adapte a função de visualização nos *displays* de 7 segmentos de modo a mostrar "**`HI`**" se o limite superior for ultrapassado, ou "**`LO`**" se a temperatura ficar abaixo do limite inferior. O sistema deve mostrar o valor da temperarura caso os limites não sejam ultrapassados.

Elementos de apoio

- Slides das aulas teóricas (aulas 15 e 16).
- Tiny Serial Digital Thermal Sensor TC74 (disponível no moodle da UC).
- PIC32 Family Reference Manual, Section 24 – I2C.

PDF criado em 14/04/2025

Anexo

A Figura 3 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*.

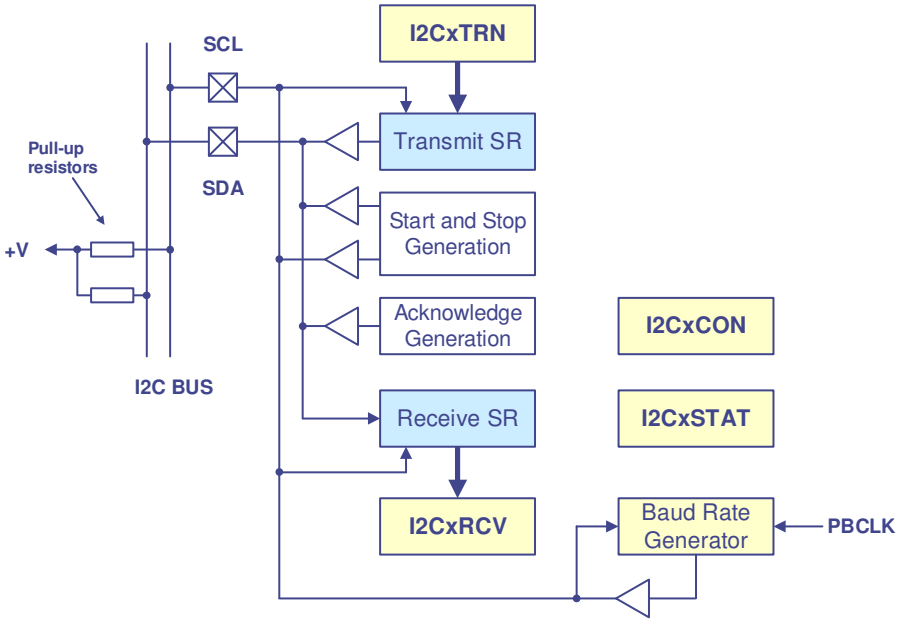


Figura 3. Diagrama de blocos simplificado do módulo I2C (*master*).

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 *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`**. O valor desta frequência é configurado 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. Essa entrada está ligada à saída *terminal count* (TC) do contador, que fica ativa sempre que o valor da contagem atinge o valor 0. Deste modo, a reposição do valor inicial de contagem é feita ciclicamente, sempre que o contador atinge o valor 0.

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 4 apresenta um diagrama simplificado do gerador de *baudrate*, que não toma em consideração as questões relacionadas com a sincronização do relógio (*clock stretching*).

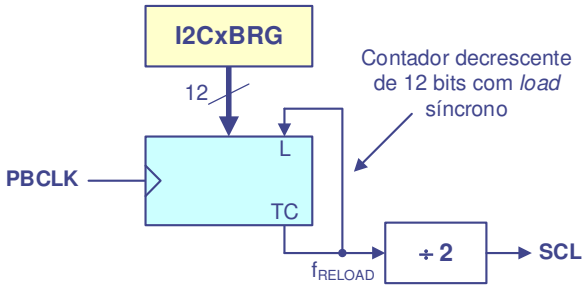


Figura 4. Diagrama de blocos simplificado do gerador de *baudrate*.

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, fazendo 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: i) escrita de um registo (de dados ou de controlo em função do passo específico); ii) 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*, o bit **SEN** 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.

Geração do *restart (repeated start)*

Para iniciar um evento de *restart* (ou *repeated start*) activa-se o bit **RSEN** (*restart enable bit*) do registo **I2CxCON**. Por *hardware*, o bit **RSEN** 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. Antes de se iniciar um evento de *restart* é necessário garantir que os 5 bits menos significativos do registo **I2CxCON** são todos 0.

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 I^2C 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.