PBLE-01

LPC111x – Guia de trabalho com a família ARM LPC111x

Rev. 0 – Dez/2022

UNIFEI – Universidade Federal de Itajubá
IESTI – Instituto de Engenharia de Sistemas e Tecnologia da Informação
Prof. Rodrigo de Paula Rodrigues

Índice

1 Introdução

Este documento apresenta os passos de instalação do ambiente de desenvolvimento para trabalho com a série de processadores NXP LPC111x.

A série LPC111x faz parte da família de microcontroladores ARM LPC11xx, da empresa *NXP Semiconductors*. Esta série é baseada em núcleos de processamento ARM em uma arquitetura chamada de Cortex-M0. Ela se destina a substituir processadores de 8 *bits* em aplicações de baixo consumo e com baixa densidade de sinais de entrada e saída.

Existem vários ambientes de desenvolvimento para microcontroladores ARM. Alguns deles ambientes são proprietários, enquanto outros são de acesso livre. Para os de acesso livre, alguns já atingiram níveis de maturidade que os credenciam para uso tanto em ambiente acadêmico quanto profissional. Exemplos de ambientes de desenvolvido proprietários difundidos são o *ARM Development Studio,* o *IAR Embedded Workbench* e o *Keil MDK-ARM*. Já exemplos de contrapartes gratuitas são o *GNU ARM Eclipse*, o *STM32CubeIDE* e o ambiente *EmBitz*.

Microcontroladores ARM da família LPC11xx podem ser programados (atualização de programa embarcado) tanto por meio de sua interface *SWD* quanto por meio de um carregador de arranque(*bootloader*) serial. A interface *SWD* permite tanto a programação do dispositivo quanto a depuração do código, mas exige, para tanto, o uso de um circuito programador externo com suporte a tal interface. Já para efetuar a programação desses dispositivos por meio de seu carregador de arranque, é necessário que o circuito de aplicação do dispositivo contemple uma conexão serial no padrão *UART*-232. Para este caso, é necessário utilizar um *software* de programação serial compatível.

Este guia aborda o uso do ambiente de desenvolvimento *EmBitz* e a programação de dispositivos LCP111x a partir de carregador de arranque serial.

2 Recursos necessários

Os recursos necessários são apresentados na Tabela 1.

Tabela 1 - Recursos necessários

	Item	Versão
1	Ambiente de desenvolvimento *EmBitz*	2.50
2	Programador serial *FlashMagic*	8.18

3 Instalação

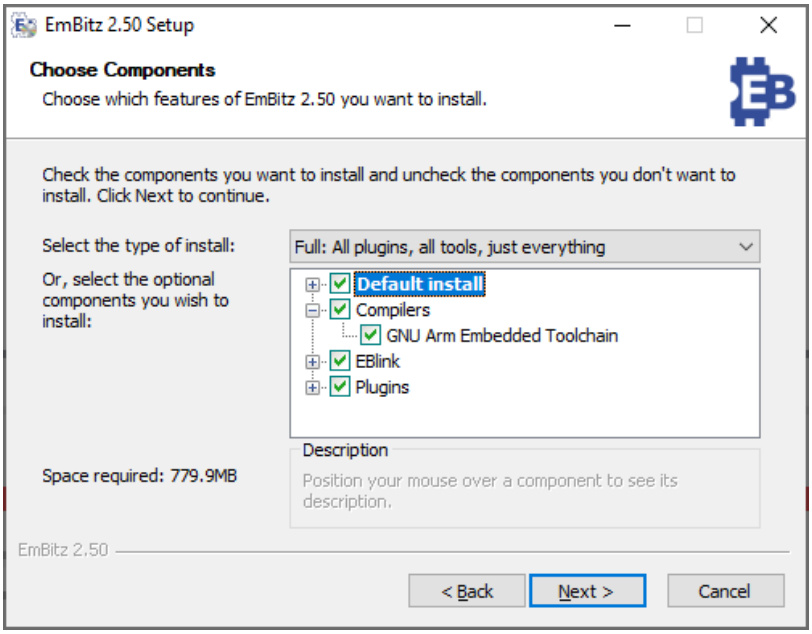
3.1 Ambiente de desenvolvimento

O ambiente de desenvolvimento EmBitz consiste em um ambiente integrado de desenvolvimento livre para a plafatormas ARM Cortex (NXP e ST, principalmente). Ele incorpora o compilador ARM GCC (*bare metal*). Dentre outros bons recursos e funcionalidades, o ambiente também suporta pontas de depuração JTAG J-Link e ST-Link.

Passos para efetuar a instalação:

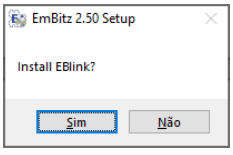
a) Execute o arquivo de instalação EmBitz_2.50.exe (ou versão mais recente);

b) No passo relativo à escolha de componentes, selecione o tipo "*Full*" e certifique-se de que a opção Compilers → GNU Arm Embeddeed ToolChain esteja selecioanda;



c) Para os passos seguintes, opte pelas opções apresentadas pela interface e prossiga até finalizar a instalação;

d) Quando questionado sobre a instalação do recurso chamado *EBLink*, é necessário responder positivamente caso se pretenda utilizar uma ponta de depuração. Em caso contrário, pode-se desprezar a instalação desse recurso.



3.2 Programador serial

Um aplicativo programador serial é necessário para efetuar a programação de microcontroladores da série LPC111x por meio de seu carregador de arranque(*bootloader*). Caso a programação seja efetuada por meio de uma ponta de depuração, não é necessário instalar o programador serial.

Passos para efetuar a instalação do aplicativo:

a) Execute o arquivo de instalação FlashMagic.exe;

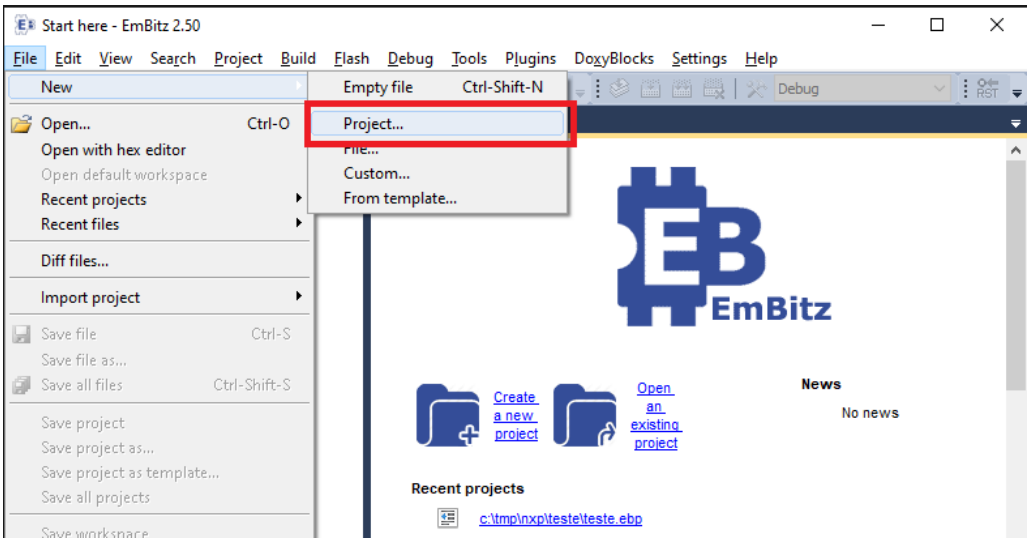
b) Siga todos os passos de instalação apresentados pelo instalador.

4 Trabalho com o ambiente de desenvolvimento

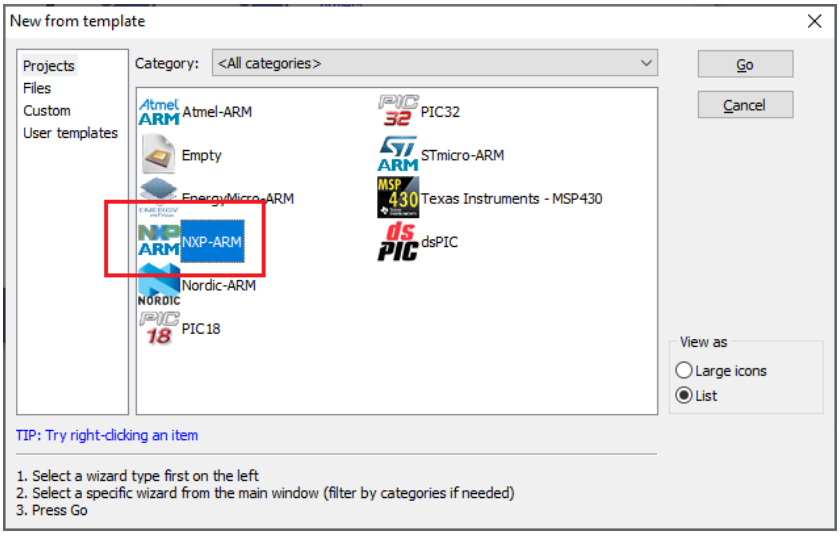
4.1 Criação de projetos

Os passos para a criação de projetos no ambiente *EmBitz* são os seguintes:

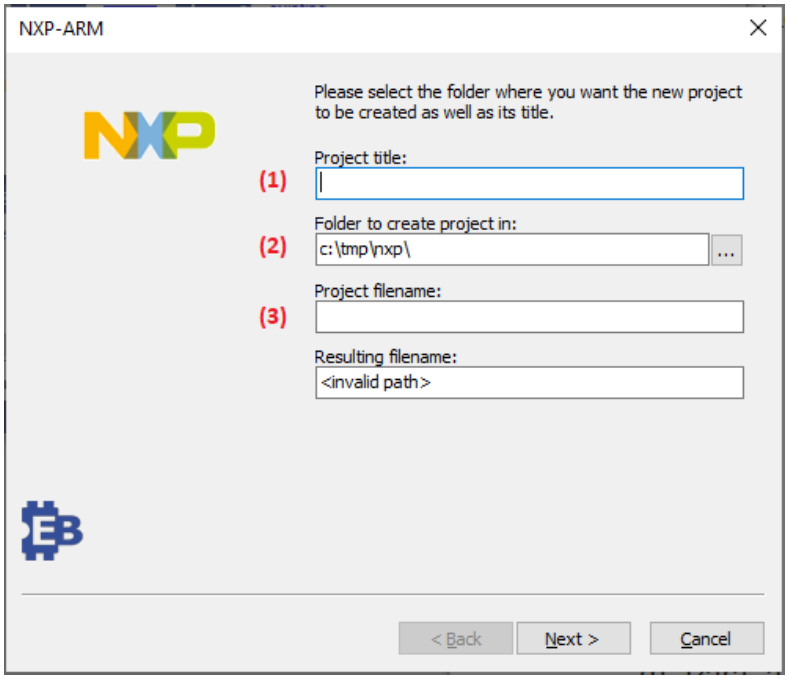
a) Com o ambiente de desenvolvimento iniciado, selecione a opção *File* > *New* > *Project* no menu de opções;



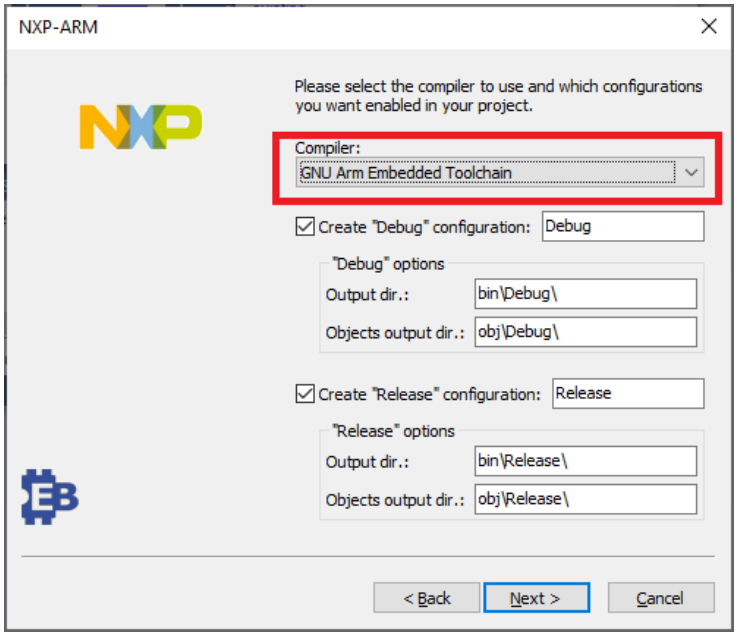
b) Na interface de escolha da categoria do projeto, opte por NXP-ARM;



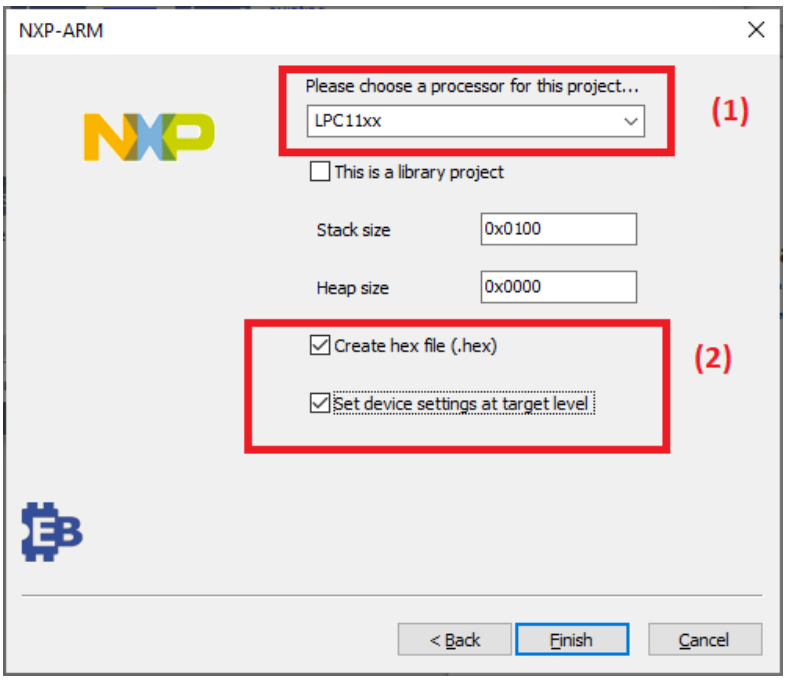
c) Selecione um título (1) para o projeto, seu caminho (2) no dispositivo de armazenamento e mesmo o nome (3) para o projeto em criação;



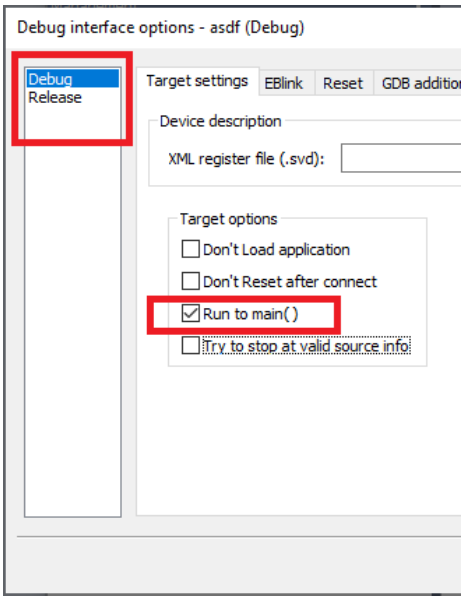
d) Para a seleção do compilador associado ao projeto, mantenha as opções no padrão apresentado pela interface;



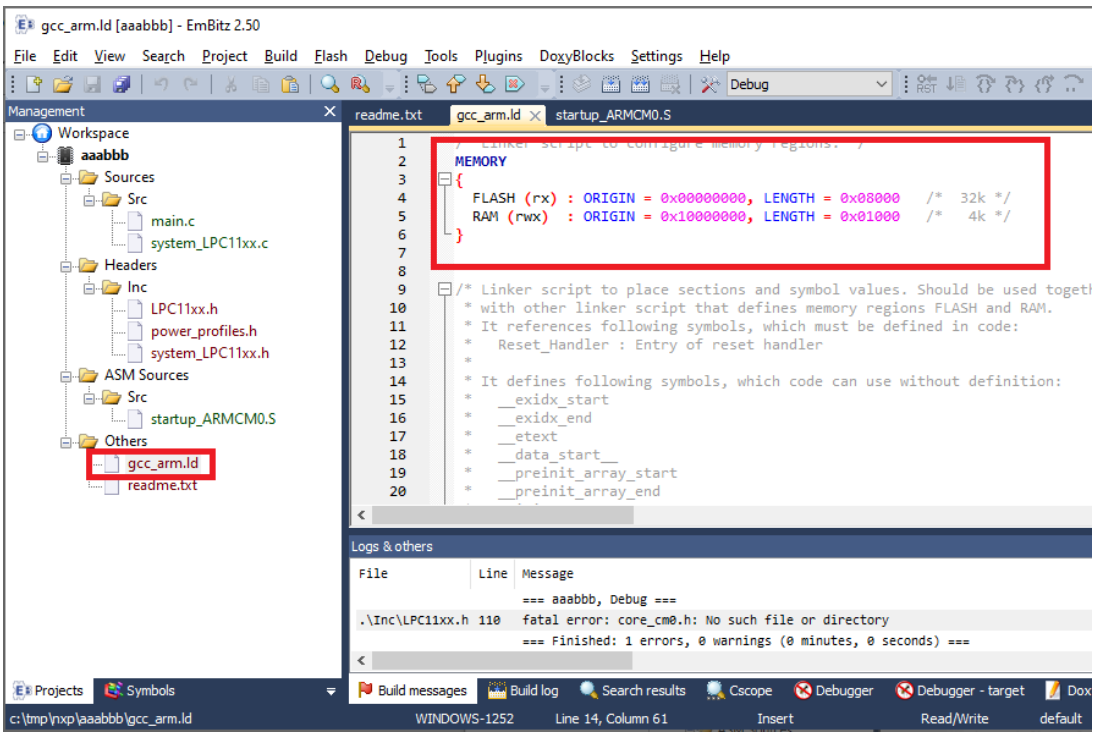
e) Efetua a escolha da série como “LPC111x” (1) e marque ambas as opções “*Create hex file (.hex)*” e “*Set device settings at target level*” (2). Ao fim, clique no botão “*Finish*”;



f) Na interface que se abre, na aba "*Target settings*", seção "Target options" marque apenas opção "*Run to main( )*" para ambos os perfis "*Debug*" e "*Release*". Mantenha os demais parâmetros inalterados.



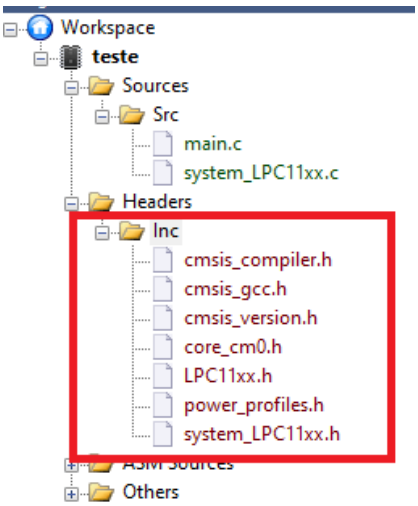
g) No painel de arquivos de projeto, selecione o arquivo *gcc_arm.ld*. Este é um arquivo de *linker* que precisa refletir a quantidade de memória *RAM* e *flash* presente no microcontrolador selecionado para o projeto. Abra tal arquivo para edição e procure, ao seu início, pela diretiva *MEMORY*.



Altere tal diretiva para refletir a quantidade de memória de seu dispositivo. Os valores padrão são compatíveis com a série LCP1114 com 32 kB de memória *ROM flash* e 4 kB de memória *RAM*. Se for necessário trabalhar com um microcontrolador com quantidades de memória diferentes, as respectivas quantidades (*length*) e suas posições em memória (*origin*) devem então ser especificadas por meio da diretiva em questão. Ao fim da alteração, o arquivo deve ser salvo.

h) Em função do padrão dos arquivos de projeto utilizado pelo ambiente de desenvolvimento *EmBitz*, é necessário adicionar a todo novo projeto criado para se trabalhar com a série LPC11xx um conjunto de arquivos de definição (arquivos .h). Estes arquivos são disponibilizados pela própria ARM e assim são externos ao ambiente *EmBitz*. Eles precisam ser manualmente inseridos no projeto.

Para se trabalhar com o compilador presente no ambiente de desenvolvimento, é necessário incluir ao projeto os seguintes arquivos: cmsis_cm0.h, cmsis_version.h, cmsis_compiler.h e cmsis_gcc.h. Desta forma, inclua tais arquivos no subdiretório de projeto “Inc”, juntamente aos demais arquivos de definições preexistentes (LPC11xx.h, power_profiles.h e system_LPC11xx.h). Ao final da inclusão, o diretório em questão deve parecer com a apresentado na figura seguinte.



4.2 Desenvolvimento do programa embarcado

Projetos criados no ambiente *EmBitz* seguem um padrão chamado CMSIS. Este padrão é sugerido pela ARM e contém todos os arquivos necessários para se configurar a operação inicial de um dispositivo ARM, cobrindo, dentre outras coisas, a estipulação de fontes dos sinais de sincronismos do núcleo e de periféricos (*clock*, temporizadores e afins). Ele também sugere um protocolo de inicialização de recursos como a posição de pilhas de dados e a direciona vetores de interrupções.

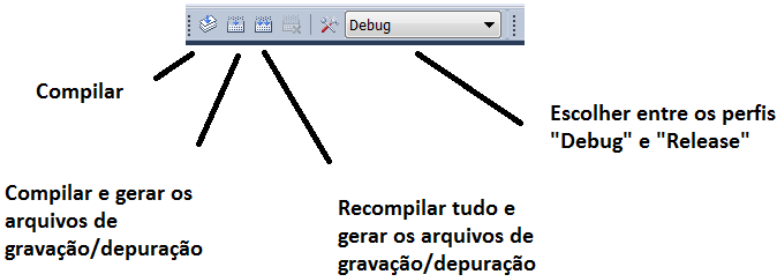
Respeitando-se o padrão CMSIS, o projeto criado possui a relação de arquivos presente na tabela abaixo (os arquivos adicionados no passo anterior foram omitidos da relação).

Arquivo	**Função/aplicação**
Source\Src	
main.c	Código-fonte para a aplicação a ser desenvolvida
system_LPC11xx.c	Rotinas de inicialização do dispositivo
Headers\Inc	
LPC11xx.h	Definições de registradores e afins do dispositivo
power_profiles.h	Definições de perfis de consume de energia
system_LPC11xx.h	Definições relativas às rotinas de inicialização
ASM Sources\Src	
startup_ARMCM0.s	Rotinas de inicialização em baixo nível
Others	
gcc_arm.ld	Arquivo de definições para o *linker*
readme.txt	Arquivo de informações sobre o *linker*

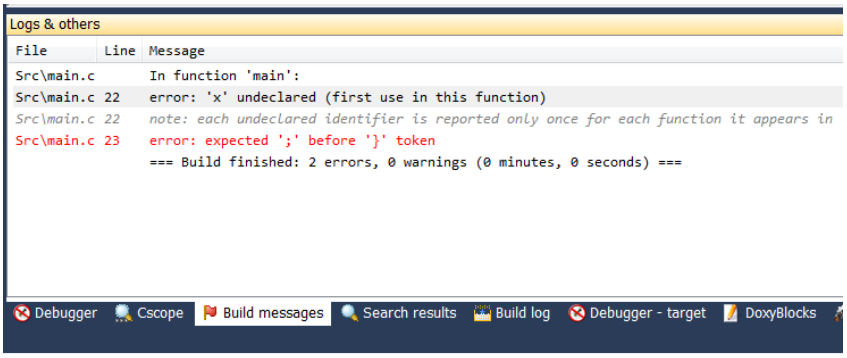
Para implementar o programa embarcado a ser desenvolvimento, utilize o arquivo principal de chamadas, o *main.c* e, quando necessário, implemente arquivos adendos para tornar o desenvolvimento modular, mais organizado e funcional.

Os demais arquivos presentes na estrutura do projeto são utilizados para se configurar a operação básica do dispositivo. Dentre eles, os arquivos *LPC11xx.h* e *system_LPC11xx.h* possuem todas as definições dos registradores e rotinas de inicialização básica do microcontrolador, respectivamente. Em especial, o arquivo *system_LPC11xx.c* possui a implementação das rotinas de inicialização básica do microcontrolador: *SystemInit* e *SystemCoreClockUpdate*. A rotina *SystemInit* é chamada após um evento de reinício (*reset*) e mesmo antes da chamada à rotina main() pois, em respeito ao padrão CMSIS, está atrelada do *vetor de reset* do dispositivo (vide arquivo em *assembly* startup_ARMCM0.s).

Para compilar um programa embarcado e mesmo gerar arquivos de gravação e depuração do código relacionado, acesse as opções presentes na barra de acesso rápido. Uma descrição das ações é apresentada na figura subsequente.



Após uma compilação, os resultados e possíveis erros podem ser acessados no painel "*Build messages*", localizado na porção inferior da interface do ambiente de desenvolvimento.



Para saber mais sobre o padrão CMSIS e entender como configurar a operação inicial do dispositivo, acesse o endereço http://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php.

Para obter mais informações sobre a programação da série LPC111x, consulte o documento “LPC111x/LPC11Cxx User manual” (UM10398).

5 Programação do embarcado

Para o contexto desde guia, a programação do *software* embarcado no dispositivo LPC111x é realizada por meio do carregador de arranque do microcontrolador. Para utilizar este carregador é necessário que seu circuito de aplicação disponibilize conexão ao periférico serial do microcontrolador, que possua um botão de reinício (*reset*), e que haja alguma forma de alternar o nível lógico em um pino do microcontrolador relacionado à operação de seu carregador de arranque, geralmente o pino PIO0_1.

Uma vez compilado e gerado o arquivo de gravação do programa embarcado (arquivo .hex relativo ao projeto desenvolvimento), os passos para programação do dispositivo por meio de seu carregador de arranque são os seguintes:

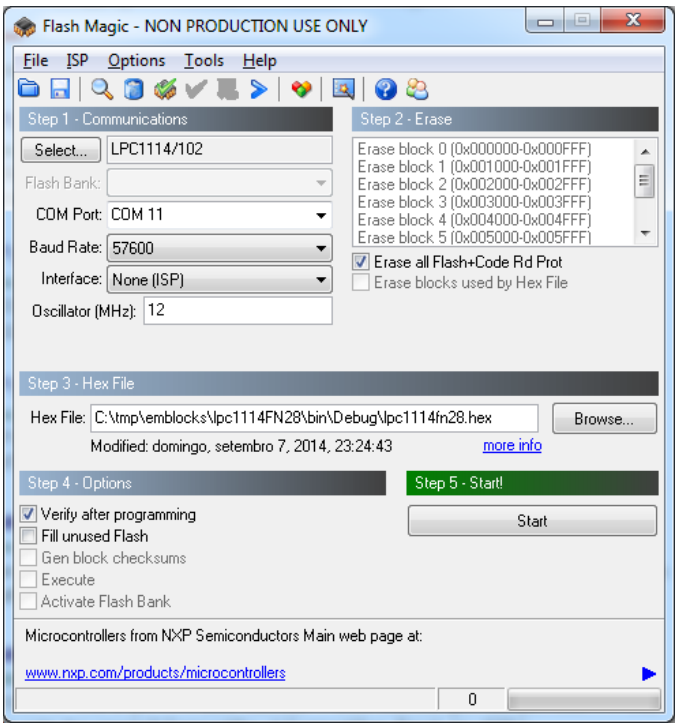
a) Abra o aplicativo *FlashMagic* e o configure da seguinte forma:

   a. Área 1 (*Step 1*):
      i. Selecione o modelo de microcontrolador utilizado (ex: LPC1114/102);
      ii. Selecione a porta serial (*COM Port*), no computador, conectada ao circuito do LPC111x;
      iii. Estipule a taxa de transferência (*Baud Rate*) para 57600;
      iv. Selecione a interface como “*None (ISP)*”;
      v. Estipule a freqüência do oscilador para 12 MHz.

   b. Área 2(*Step 2*):
      i. Marque a opção “*Erase all Flash+Code RD Prot*”.

   c. Área 3(*Step 3*):
      i. Selecione o arquivo .hex a ser programado no microcontrolador;

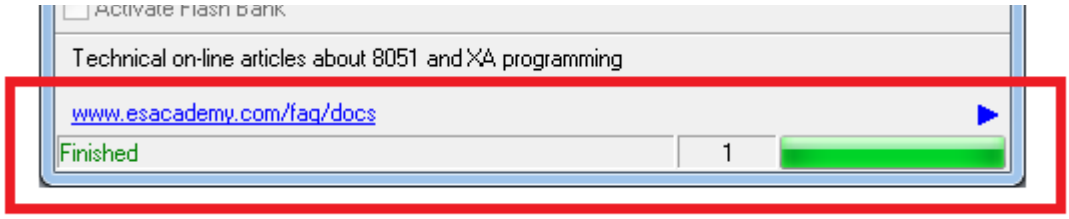
   d. Área 4(*Step 4*):
      i. Marque apenas a opção “*Verify after programming*”.



b) Com a interface devidamente configurada para gravação, aplique nível lógico baixo ao pino de controle do carregador de arranque do microcontrolador (PIO0_1) e depois aplique um sinal de reinício (pulso negativo no pino de *reset*) ao microcontrolador;

c) Neste momento, inicie o processo de gravação por meio do botão "*Star*t" na área *Step 5* do *FlashMagic*;

d) Após certo tempo, deve surgir na interface do *FlashMagic* um sinal de fim e de sucesso da operação de gravação;



e) Uma vez finalizada a operação de gravação com sucesso, para que o microcontrolador possa executar o programa gravado é necessário que um sinal de nível lógico alto seja estipulado no controle do carregador de arranque do microcontrolador (PI0_1) e um novo sinal de reinício (pulso negativo no pino de *reset*) seja aplicado ao microcontrolador.

6 Programa de exemplo: acionamento de *LED*

O seguinte trecho de programa é um exemplo de arquivo *main.c* que faz com que o pino PT0_3 de um microcontrolador da série LPC111x tenha seu estado lógico alternado continuamente. Por ele, pode-se verificar como é configurada a direção de tal como saída e como seu estado lógico de saída pode ser acessado e alterado. Todos os demais parâmetros são configurados pela rotina *SystemInit* presente no arquivo "system_LPC11xx.c" pertencente ao projeto.

main.c

```
#include <LPC11xx.h>            /* LPC11xx Peripheral Registers */
#include "system_LPC11xx.h"

#define LED_PORT LPC_GPIO0
#define LED_PIN  3


int main(void)
{
   unsigned int x;

   // Configurar o pino PT0_3 como saída
   LED_PORT->DIR  |= (1 << LED_PIN);

   while (1)
   {

      // Alternar o nível lógico no pino PT0_3
      LED_PORT->DATA ^= (1 << LED_PIN);

      // Gerar um pequeno atraso para simular um "delay"
      for( x = 0; x < 0x002fffff; x++);

   };

   return 0;
}
```

7 Programas embarcados de exemplo

7.1 Trabalho com portas de entrada e saída digitais

main.c

```
/*
**
**                          Main.c
**
**
*********************************************************************/
/*

          Este programa continuamente ler o valor lógico de entrada no pino PTO0_2 e
          envia seu inverso (complemento) ao pino PTO0_3.

          Universidade Federal de Itajubá
          Prof. Rodrigo de Paula Rodrigues
          2014-09-20

*********************************************************************/


#include <LPC11xx.h>                            /* LPC11xx Peripheral Registers */
#include "system_LPC11xx.h"

#define PINO_LED  3
#define PINO_BTN  2

int main(void)
{

          //
          // O procedimento SystemInit (CMSIS) configura o CLOCK principal
          // do dispositivo para 48 MHz a partir de um clock externo de
          // 12 MHz (cristal) e de seu circuito PLL interno.
          //
          // Para estipular outra configuração do clock principal,
          // altere as constantes SYSOSCCTRL_Val, WDTOSCCTRL_Val,
          // SYSPLLCTRL_Val, SYSPLLCLKSEL_Val, MAINCLKSEL_Val e
          // SYSAHBCLKDIV_Val presentes no arquivo system_LPC11xx.c
          //

          //
          // Configurar os demais periféricos
          //

          LPC_SYSCON->SYSAHBCLKCTRL |= (1<<6);    // habilitar o clock para o módulo GPIO (UM10398, 3.5.14)
          LPC_GPIO0->DIR  |= (1 << PINO_LED);     // configurar o pino PTO0_3 como saída digital (UM10398, 12.3.2)
          LPC_GPIO0->DIR  &= ~(1 << PINO_BTN);    // configurar o pino PTO0_2 como entrada digital (UM10398, 12.3.2)

          //
          // Laço de repetição da software
          //

          while (1)
          {

                    //
                    // Monitorar o nível de entrada no pino PTO0_2 (UM10398, 12.3.1)
                    //

                    if ( LPC_GPIO0->DATA & (1<< PINO_BTN) )
                    {
                              // Estipular nível de saída baixo (UM10398, 12.3.1)
                              LPC_GPIO0->DATA &= ~(1 << PINO_LED);
                    }
                    else
                    {
                              // Estipular nível de saída alto (UM10398, 12.3.1)
                              LPC_GPIO0->DATA |= (1 << PINO_LED);
                    }

          };

          return 0;
}
```

7.2 Trabalho com o conversor analógico para digital (ADC)

main.c

```
/*
**
**                           Main.c
**
**
***********************************************************************/
/*
   Last committed:     $Revision: 00 $
   Last changed by:    $Author: $
   Last changed date:  $Date:  $
   ID:                 $Id:  $

***********************************************************************/
/*

          Este programa continuamente amostra o sinal no pino PIO0_11 e
          envia o resultado como uma sequência hexadecimal de 3 dígitos pela
          interface serial.A forma de representação dos dados é '###<CR><LF>'.

          Obs: a interface serial está configurada como <19200, 8, N, 1>

          Universidade Federal de Itajubá
          Prof. Rodrigo de Paula Rodrigues
          2014-09-20

***********************************************************************/

#include <LPC11xx.h>                            /* LPC11xx Peripheral Registers */
#include "system_LPC11xx.h"

int main(void)
{
          unsigned long   i;
          char            r[5];


          //
          // O procedimento SystemInit (CMSIS) configura o CLOCK principal
          // do dispositivo para 48 MHz a partir de um clock externo de
          // 12 MHz (cristal) e de seu circuito PLL interno.
          //
          // Para estipular outra configuração do clock principal,
          // altere as constantes SYSOSCCTRL_Val, WDTOSCCTRL_Val,
          // SYSPLLCTRL_Val, SYSPLLCLKSEL_Val, MAINCLKSEL_Val e
          // SYSAHBCLKDIV_Val presentes no arquivo system_LPC11xx.c
          //


           //
          // Configurar os demais periféricos
          //

          // Configurar a UART em 19200, 8 bits de dados, 1 bit de parada, sem paridade, sem caracter de quedra, sem controle de
fluxo


          LPC_SYSCON->SYSAHBCLKCTRL |= (1<<16);   // habilitar o clock para o bloco IOCON (UM10398, 3.5.14)
          LPC_IOCON->PIO1_7         |= 0x01;      // configurar o pino UART TXD (UM10398, 7.4.41)
          LPC_IOCON->PIO1_6         |= 0x01;      // configurar o pino UART RXD (UM10398, 7.4.41)

          LPC_SYSCON->SYSAHBCLKCTRL |= (1<<12);   // habilitar o clock para o bloco UART (UM10398, 3.5.14)
          LPC_SYSCON->UARTCLKDIV    |= 0x9B;      // estipular o clock do módulo UART (divisor de 0x9B) para gerar BR de 19.2K
(UM10398, 3.5.16)
          LPC_UART->FCR             |= 0x01;      // habiltiar o FIFO (necessário para operar) (UM10398, 13.5.6)
          LPC_UART->LCR             |= 0x03;      // estipular um tabamho de palavra de 8 bits (UM10398, 13.5.7)
          LPC_UART->TER             |= 0x80;      // habilitar a transmissão (UM10398, 13.5.16)


          // Configurar o AD para amostrar

          LPC_SYSCON->SYSAHBCLKCTRL |= (1<<13);   // habilitar o clock para o bloco AD (UM10398, 3.5.14)
          LPC_SYSCON->PDRUNCFG      &= ~(1<<4);   // ativar o AD (UM10398,3.5.36)


          LPC_IOCON->R_PIO0_11      = 0x02;       // configurar o pino PIO0_11 (AD0) como entrada analógica (UM10398, 7.4.29)
                                                                                                  // FUNC=2, MODE=0
e ADMODE=0

          LPC_ADC->CR               = 0x0C00;     // selecionar o canal AD0 e configurar seu clock para 4 MHz (UM10398, 25.5.1)
                                                                                                  // SEL=0,
CLKDIV=12, BURST=0, CLKS=00

          //
          // Laço de repetição da aplicação
          //


          while(1)
          {

                    // iniciar uma converção ( bits START=01) (UM10398, 25.5.1)
                    LPC_ADC->CR     |= (1<<24);
```

```
                    // esperar pelo fim da conversão (bit DONE em 1) (UM10398, 25.5.4)
                    while( (LPC_ADC->GDR < 0x7FFFFFFF));


                    //
                    // Converter o resultado da conversão
                    // presente nos bits 15-6 de GDR em
                    // uma sequência de 3 dígitos hexadecimais
                    //

                    r[0] = ( LPC_ADC->GDR >> 14 ) & 0x03;    // Obter os dois bits mais significativos
                    r[1] = ( LPC_ADC->GDR >>  6 ) & 0xFF;    // Obter os 8 bits menos signiticativos
                    r[2] = r[1];
                    r[1] /= 16;                              // obter as 'dezenas hexadecimais'  relativas aos 8 bits menos
significativos
                    r[2] %= 16;                              // obter as 'unidades hexadecimais' relativas aos 8 bits menos
significativos
                    r[3] =  13;                              // adicionar a quedra de linha ao fim da sequência
                    r[4] =  10;                              // adicionar o retorno de linha

                    r[0] += '0';                             // obter os respectivo dígito ASCII para cada dígito resultante
                    r[1] += '0';
                    r[2] += '0';

                    // Tratar o caso dos valores acima de '9'

                    if ( r[1] > '9' )
                    {
                              r[1] = (r[1] - 58) + 'A';
                    }

                    if ( r[2] > '9' )
                    {
                              r[2] = (r[2] - 58) + 'A';
                    }

                    //
                    // Enviar o resultado da conversão pela interface serial
                    // no formato de 3 dígitos hexadecimais
                    //

                    for( i = 0; i < 5; i++ )
                    {
                              // Esperar pelo fim da transmissão atual (bit TEMT, UM10398, 13.5.9)
                              while ( ( LPC_UART->LSR & ( 1 << 6 ) ) == 0 );

                               // transmitir o caractere da sequência (UM10398, 25.5.2)
                              LPC_UART->THR  = r[i];

                    }

                    // Gerar um atraso entre a coleta de amostras
                    for( i = 0; i < 0x2FFFFF; i++ );

          }

          return 0;
}
```

7.3 Trabalho com periférico de comunicação UART

main.c

```
/*
**
**                           Main.c
**
**
*********************************************************************/
/*

          Este programa exemplifica o uso das funções de TX e RX
          inerentes ao periférico UART. Para tanto, qualquer caractere recebido
          pelo módulo é prontamente enviado ao transmissor.

          Obs: a interface serial está configurada como <19200, 8, N, 1>

          Universidade Federal de Itajubá
          Prof. Rodrigo de Paula Rodrigues
          2014-09-20

*********************************************************************/


#include <LPC11xx.h>                              /* LPC11xx Peripheral Registers */
#include "system_LPC11xx.h"

int main(void)
{
          char c;
```

```
	//
	// O procedimento SystemInit (CMSIS) configura o CLOCK principal
	// do dispositivo para 48 MHz a partir de um clock externo de
	// 12 MHz (cristal) e de seu circuito PLL interno.
	//
	// Para estipular outra configuração do clock principal,
	// altere as constantes SYSOSCCTRL_Val, WDTOSCCTRL_Val,
	// SYSPLLCTRL_Val, SYSPLLCLKSEL_Val, MAINCLKSEL_Val e
	// SYSAHBCLKDIV_Val presentes no arquivo system_LPC11xx.c
	//

	//
	// Configurar os demais periféricos
	//

	LPC_SYSCON->SYSAHBCLKCTRL |= (1<<16);   // habilitar o clock para o bloco IOCON (UM10398, 3.5.14)
	LPC_IOCON->PIO1_7         |= 0x01;      // configurar o pino UART TXD (UM10398, 7.4.41)
	LPC_IOCON->PIO1_6         |= 0x01;      // configurar o pino UART RXD (UM10398, 7.4.41)

	LPC_SYSCON->SYSAHBCLKCTRL |= (1<<12);    // habilitar o clock para o bloco UART (UM10398, 3.5.14)
	LPC_SYSCON->UARTCLKDIV    |= 0x9B;       // estipular o clock do módulo UART (divisor de 0x9B) para gerar BR de 19.2K
(UM10398, 3.5.16)
	LPC_UART->FCR             |= 0x01;       // habiltiar o FIFO (necessário para operar) (UM10398, 13.5.6)
	LPC_UART->LCR             |= 0x03;       // estipular um tabamho de palavra de 8 bits (UM10398, 13.5.7)
	LPC_UART->TER             |= 0x80;       // habilitar a transmissão (UM10398, 13.5.16)

	while(1)
	{

		//
		// Verificar se algum dado foi recebido (bit RDR, UM10398, 13.5.9)
		//

		if ( LPC_UART->LSR & 0x01 )
		{

			// obter o dado recebido (UM10398, 13.5.1)
			c = LPC_UART->RBR;

			// Esperar pelo fim da transmissão atual (bit TEMT, UM10398, 13.5.9)
			while ( ( LPC_UART->LSR & ( 1 << 6 ) ) == 0 );

			// Transmitir o caracter recebido  (UM10398, 13.5.2)
			LPC_UART->THR = c;

		}
	}

	return 0;
}
```

7.4 Integração com periférico I2C (RTC)

main.c

```
/*
**
**                           Main.c
**
**
*********************************************************************/
/*

	Este programa exemplifica o uso do periférico de comunicação I2C
	em modo mestre para se comunicar com um RTC DS1307.

	Obs:
	1 - para utilizar a comunicação I2C, os pinos PIO0_4 e PIO0_5
	    precisam ser conectados a resistores de pull-up;
	2 - o periférico I2C é configurado para trabalhar emm 100 kHz
	3 - o circuito de emprego do RTC é o mostrado em DS1307, "Typical Operation Circuit"

	Universidade Federal de Itajubá
	Prof. Rodrigo de Paula Rodrigues
	2014-09-28

*********************************************************************/


#include <LPC11xx.h>                            /* LPC11xx Peripheral Registers */
#include "system_LPC11xx.h"

#define K_ENDERECO_DS1307              0x68     // Endereço em 7 bits


void I2C_Transmitir ( unsigned char endereco, unsigned char *valor, unsigned char qtd );
void I2C_Receber    ( unsigned char endereco, unsigned char *valor, unsigned char qtd );

int main(void)
{
	unsigned int i;
	char                dados[2];
```

```
          //
          // O procedimento SystemIniti (CMSIS) configura o CLOCK principal
          // do dispositivo para 48 MHz a partir de um clock externo de
          // 12 MHz (cristal) e de seu circuito PLL interno.
          //
          // Para estipular outra configuração do clock principal,
          // altere as constantes SYSOSCCTRL_Val, WDTOSCCTRL_Val,
          // SYSPLLCTRL_Val, SYSPLLCLKSEL_Val, MAINCLKSEL_Val e
          // SYSAHBCLKDIV_Val presentes no arquivo system_LPC11xx.c
          //

          //
          // Configurar os demais periféricos
          //


          LPC_SYSCON->SYSAHBCLKCTRL |= (1<<16);   // habilitar o clock para o bloco IOCON (UM10398, 3.5.14)

          LPC_IOCON->PIO0_4         = 0x01;               // configurar o pino como SCL em modo I2C padrão (I2CMODE=00)
(UM10398, 7.4.11)
          LPC_IOCON->PIO0_5         = 0x01;               // configurar o pino como SDA em modo I2C padrão (I2CMODE=00)
(UM10398, 7.4.12)



          LPC_SYSCON->SYSAHBCLKCTRL |= (1<<5);    // habilitar o clock para o bloco I2C (UM10398, 3.5.14)
          LPC_SYSCON->PRESETCTRL    |= (1<<1);    // garantir que o periférico I2C não esteja em estado de reset (UM10398,
3.5.2)


          LPC_I2C->SCLH                             = 240;          // Estipular o clock I2C para 100kHz (UM10398, 15.7.5.1)
          LPC_I2C->SCLL                             = 240;

          LPC_I2C->CONSET                           |= (1<<6);      // habilitar o periférico I2C como mestre (UM10398, 15.7.1
/ 15.8.1)

          // Estipular os segundos atuais

          dados[0] = 0x00;                                                  // registro de segundos (DS1307, tabela 2)
          dados[1] = 0x10;                                                  // 10 segundos (BCD) e CH em 0 para habilitar o
clock (DS1307, tabela 2)

          I2C_Transmitir( K_ENDERECO_DS1307, (unsigned char *)&dados[0], 2 );


          while(1)
          {

                    // Posicionar o RTC em seu registro de segundos (DS1307, tabela 2)
                    dados[0] = 0x00;
                    I2C_Transmitir( K_ENDERECO_DS1307, (unsigned char *)&dados[0], 1 );

                    // Ler o valor corrente do registro de segundos do RTC (em formato BCD) para a variável 'dados[1]'
                    I2C_Receber( K_ENDERECO_DS1307, (unsigned char *)&dados[1], 1 );

                    for ( i = 0; i < 0xFFFFF; i++ );        // gerar um atraso
          }

          return 0;
}


//
// Transmitir os valores presentes no vetor 'valor' para o
// periférico escravo relacionado ao endereço 'endereco' (7 bits)
//

void I2C_Transmitir( unsigned char endereco, unsigned char *valor, unsigned char qtd )
{
          unsigned char c;


          LPC_I2C->CONSET = (1<<5);                                 // requisitar o evento de START (UM10398, 15.7.1)

          while ( LPC_I2C->STAT != 0x08 );                  // esperar pelo pelo fim do evento de START (bit SI, UM10398,
15.7.1)

          LPC_I2C->DAT        = endereco << 1;              // enviar o endereço do dispositivo escravo e espeficicação de
escrita (R/W=0) (UM10398, 15.7.3)
          LPC_I2C->CONCLR = (5<<3);                                 // limpar a sinalizações SI e START (UM10398, 15.7.6)
          while( LPC_I2C->STAT != 0x18 );                           // esperar por um estado 0x18 (ACK recebido) (UM10398,
15.10.1, tabela 236)

          //
          // Enviar os dados requisitados
          //

          for( c = 0; c < qtd; c++ )
          {
                    LPC_I2C->DAT        = valor[c];                         // transmitir o dado (UM10398, 15.7.3)
                    LPC_I2C->CONCLR = (1<<3);                   // limpar o bit SI para transmitir um byte e esperar por
um ACK (UM10398, 15.7.6 e tabela 236)

                    while( LPC_I2C->STAT != 0x28 );                   // esperar por um estado 0x28 (ACK recebido) (UM10398,
15.10.1, tabela 236)

          }
```

```
        LPC_I2C->CONSET = (1<<4);                                   // requisitar o evento de STOP (UM10398, 15.7.1)
        LPC_I2C->CONCLR = (1<<3);                                   // limpar o bit SI (UM10398, 15.7.6)

}

//
// Receber um total de "qtd" valores relacionados ao 'endereço'
// do periférico escravo (7 bits) especificado
//

void I2C_Receber( unsigned char endereco, unsigned char *valor, unsigned char qtd )
{
        unsigned char c;

        LPC_I2C->CONSET = (1<<5);                                   // requisitar o evento de START (UM10398, 15.7.1)

        while ( LPC_I2C->STAT != 0x08 );                  // esperar pelo pelo fim do evento de START (bit SI, UM10398,
15.7.1)

        LPC_I2C->DAT        = (endereco<<1)+1;            // enviar o endereço do dispositivo escravo e espeficicação de
leitura (R/W=1) (UM10398, 15.7.3)
        LPC_I2C->CONCLR = (5<<3);                                   // limpar a sinalizações SI e START (UM10398, 15.7.6)

        while( LPC_I2C->STAT != 0x40 );                             // esperar pela transmissão do endereço e recebimento de
um ACK (estado 0x40) (UM10398, 15.10.1, tabela 237)


        //
        // Receber os dados requisitados
        //

        for (c = 0; c < qtd; c++ )
        {

                //
                // Quando for receber o último byte, deve-se gerar
                // um NACK (Not Ack), caso contrário, um ACK
                //

                LPC_I2C->CONCLR = 0x2C;                                 // limpar os bits SI, START e AA para gerar um
NACK (UM10398, 15.7.6 e tabela 237)

                if ( c != (qtd-1) )
                {
                        LPC_I2C->CONSET = 0x4;                          // estipular o AA para gerar um ACK (UM10398,
15.7.1 e tabela 2.3.7)
                }


                // esperar pelo recebimento de um byte  (o estado deve assumir 0x50 ou 0x58) (UM10398, 15.10.2, tabela 237)
                while( (LPC_I2C->STAT != 0x50) && (LPC_I2C->STAT != 0x58) );

                // recuperar e armazenar o byte recebido na lista de valores
                valor[c] = LPC_I2C->DAT;
        }


        LPC_I2C->CONSET = (1<<4);                                   // requisitar o evento de STOP (UM10398, 15.7.1)
        LPC_I2C->CONCLR = (1<<3);                                   // limpar o bit SI (UM10398, 15.7.6)

}
```

7.5 Integração com periférico I2C (DAC)

main.c

```
/*
**
**                           Main.c
**
**
*********************************************************************/
/*

        Este programa exemplifica o uso do periférico de comunicação I2C
        para se comunicar com o DAC MCP4725.

        O embarcado utiliza o DAC para gerar uma forma de onda de saída senoidal.

        Mapa de pinos entre DAC e LPC1114:
        Vss(2) <-> GND
        Vdd(e) <-> 3.3V
        SDA(4) <-> PIO0_5 (com pullup para 3.3V)
        SCL(5) <-> PIO0_4 (com pullup para 3.3V)
        AO(6)  <-> GND


        Obs:
        1 - para utilizar a comunicação I2C, os pinos PIO0_4 e PIO0_5
            precisam ser conectados a resistores de pull-up;
        2 - o periférico I2C é configurado para trabalhar em 100 kHz;
        3 - é utilizado o modo de comunicação em mode rápido do DAC (MCP4725, 6.1.1).
```

```
          Universidade Federal de Itajubá
          Prof. Rodrigo de Paula Rodrigues
          2015-04-15

*********************************************************************/


#include <LPC11xx.h>                            /* LPC11xx Peripheral Registers */
#include "system_LPC11xx.h"

#define K_ENDERECO_MCP4725              0x62      // Endereço em 7 bits, podendo ser 0x60, 0x62, 0x64 ou 0x66
                                                  // Vide partnumber completo do DAC para identificar o endereço (MCP4725, 10.0)


//void I2C_Transmitir ( unsigned char endereco, unsigned char *valor, unsigned char qtd );
//void I2C_Receber  ( unsigned char endereco, unsigned char *valor, unsigned char qtd );

// Definições e variáveis globais

uint16_t guiAmostra[32] = { 2048,2447,2831,3185,
                                                                        3495,3750,3939,4056,
                                                                        4095,4056,3939,3750,
                                                                        3495,3185,2831,2447,
                                                                        2048,1648,1264,910,
                                                                        600,345,156,39,
                                                                        0,39,156,345,
                                                                        600,910,1264,1648 };          // amostras que formam uma
forma de onda senoidal

// Protótipos locais

void DAC_Transmitir ( uint16_t v_uAmostra );

// Laço principal

int main(void)
{
          unsigned int  viI;
          unsigned char vucIndice;

          //
          // O procedimento SystemIniti (CMSIS) configura o CLOCK principal
          // do dispositivo para 48 MHz a partir de um clock externo de
          // 12 MHz (cristal) e de seu circuito PLL interno.
          //
          // Para estipular outra configuração do clock principal,
          // altere as constantes SYSOSCCTRL_Val, WDTOSCCTRL_Val,
          // SYSPLLCTRL_Val, SYSPLLCLKSEL_Val, MAINCLKSEL_Val e
          // SYSAHBCLKDIV_Val presentes no arquivo system_LPC11xx.c
          //

          //
          // Configurar os demais periféri LPC_I2C->cos
          //



          LPC_SYSCON->SYSAHBCLKCTRL |= (1<<16);   // habilitar o clock para o bloco IOCON (UM10398, 3.5.14)

          LPC_IOCON->PIO0_4         = 0x01;                 // configurar o pino como SCL em modo I2C padrão (I2CMODE=00)
(UM10398, 7.4.11)
          LPC_IOCON->PIO0_5         = 0x01;                 // configurar o pino como SDA em modo I2C padrão (I2CMODE=00)
(UM10398, 7.4.12)



          LPC_SYSCON->SYSAHBCLKCTRL |= (1<<5);    // habilitar o clock para o bloco I2C (UM10398, 3.5.14)
          LPC_SYSCON->PRESETCTRL    |= (1<<1);    // garantir que o periférico I2C não esteja em estado de reset (UM10398,
3.5.2)


          LPC_I2C->SCLH                             = 240;          // Estipular o clock I2C para 100kHz (UM10398, 15.7.5.1)
          LPC_I2C->SCLL                             = 240;

          LPC_I2C->CONSET                           |= (1<<6);      // habilitar o periférico I2C como mestre (UM10398, 15.7.1
/ 15.8.1)

          //
          // No laço de repetição, varrer o vetor de amostras
          // e transmitir cada um dos valores ao DAC
          //

          vucIndice = 0;

          while(1)
          {

                    // transmitir o valor atual
                    DAC_Transmitir( guiAmostra[ vucIndice ] );

                    // Atualizar o índice e gerar contagem em módulo 32
                    vucIndice++;
                    vucIndice &= 0x1F;

                    for ( viI = 0; viI < 2000; viI++ );     // gerar um atraso
          }

          return 0;
}
```

```
//
// Transmitir os valores para o DAC
//
// Utilizar o modo de atualização especificado em (MCP4725, 6.1.1)
//

void DAC_Transmitir ( uint16_t v_uAmostra )
{

          char vcDado;

          LPC_I2C->CONSET = (1<<5);                                          // requisitar o evento de START (UM10398,
15.7.1)

          while ( LPC_I2C->STAT != 0x08 );                             // esperar pelo pelo fim do evento de START (bit SI,
UM10398, 15.7.1)

          LPC_I2C->DAT        = K_ENDERECO_MCP4725 << 1;    // enviar o endereço do dispositivo escravo e espeficicação de
escrita (R/W=0) (UM10398, 15.7.3)
          LPC_I2C->CONCLR = (5<<3);                                          // limpar a sinalizações SI e START (UM10398,
15.7.6)
          while( LPC_I2C->STAT != 0x18 );                                    // esperar por um estado 0x18 (ACK recebido)
(UM10398, 15.10.1, tabela 236)

          //
          // Enviar os dados requisitados
          //

          // Transmitir byte com os valores  <C2, C1, PD1, PD0> = <0,0,0,0>
          // O formato é apresentado em  (MCP4725, figura 6-1)

          vcDado =  ( v_uAmostra >> 8 );
          LPC_I2C->DAT        = vcDado;                                // transmitir o dado (UM10398, 15.7.3)
          LPC_I2C->CONCLR = (1<<3);                                    // limpar o bit SI para transmitir um byte e esperar por
um ACK (UM10398, 15.7.6 e tabela 236)

          while( LPC_I2C->STAT != 0x28 );                              // esperar por um estado 0x28 (ACK recebido) (UM10398,
15.10.1, tabela 236)

          //
          // Transmitir o segundo byte com a parte menos significativa da amostra
          // Vide (MCP4725, figura 6-1)
          //
          vcDado =  ( v_uAmostra & 0xFF);
          LPC_I2C->DAT        = vcDado;                                // transmitir o dado (UM10398, 15.7.3)
          LPC_I2C->CONCLR = (1<<3);                                    // limpar o bit SI para transmitir um byte e esperar por
um ACK (UM10398, 15.7.6 e tabela 236)

          while( LPC_I2C->STAT != 0x28 );                              // esperar por um estado 0x28 (ACK recebido) (UM10398,
15.10.1, tabela 236)


          LPC_I2C->CONSET = (1<<4);                                    // requisitar o evento de STOP (UM10398, 15.7.1)
          LPC_I2C->CONCLR = (1<<3);                                    // limpar o bit SI (UM10398, 15.7.6)

}
```

7.6 Trabalho com interrupções em pino de entrada digital

main.c

```
/*
**
**                          Main.c
**
**
*********************************************************************/
/*
   Last committed:     $Revision: 00 $
   Last changed by:    $Author: $
   Last changed date:  $Date:  $
   ID:                 $Id:  $

*********************************************************************/
/*

          Este programa exemplifica o tratamento de interrupção por mudança
          de estado lógico no pino PIO0_2. A cada evento de borda de subida
          em tal pino, o valor de saída no pino PTO0_3 é alternado.

          Universidade Federal de Itajubá
          Prof. Rodrigo de Paula Rodrigues
          2014-09-20

*********************************************************************/

#include <LPC11xx.h>                          /* LPC11xx Peripheral Registers */
#include "system_LPC11xx.h"

#define PINO_LED  3      // Pino GPIO0_3
```

```
#define PINO_BTN  2      // Pino GPIO0_2


//
// Manipulador da interrupção nas portas de E/S (GPIO)
//
// O vetor de tal interrupção é o 31 (UM10398, 6.4, tabela 55)
// e foi declarado como SPI_IRQHandler no arquivo "startup_ARMCM0.S"
//

void SPI_IRQHandler()
{
          // Comutar o estado de saída do pino PIO0_3 (UM10398, 12.3.1)
          LPC_GPIO0->DATA ^= (1 << PINO_LED);

          // Limpar a sinalização de interrupção (UM10398, 12.3.9)
          LPC_GPIO0->IC = (1 << PINO_BTN);

}


int main(void)
{


          //
          // O procedimento SystemInit (CMSIS) configura o CLOCK principal
          // do dispositivo para 48 MHz a partir de um clock externo de
          // 12 MHz (cristal) e de seu circuito PLL interno.
          //
          // Para estipular outra configuração do clock principal,
          // altere as constantes SYSOSCCTRL_Val, WDTOSCCTRL_Val,
          // SYSPLLCTRL_Val, SYSPLLCLKSEL_Val, MAINCLKSEL_Val e
          // SYSAHBCLKDIV_Val presentes no arquivo system_LPC11xx.c
          //


          //
          // Configurar os demais periféricos
          //

           // Configurar a porta de saída para o LED e para o BTN
          LPC_SYSCON->SYSAHBCLKCTRL |= (1<<6);     // habilitar o clock para o módulo GPIO (UM10398, 3.5.14)
          LPC_GPIO0->DIR |= (1 << PINO_LED);       // Pino PIO0_3 como saída digital (UM10398, 12.3.2)
          LPC_GPIO0->DIR &= ~(1 << PINO_BTN);      // Pino PIO0_2 como entrada digital (UM10398, 12.3.2)
          LPC_GPIO0->DATA = ~(1 << PINO_LED);      // iniciar o LED desligado (UM10398, 12.3.1)



          //
          // Configurar o pino PIO0_2 para gerar interrupções na borda de subida
          //


          NVIC_ClearPendingIRQ( EINT0_IRQn );      // Limpar interrupção pedente (UM10398, 28.6.2.1, tabela 442)
          NVIC_EnableIRQ( EINT0_IRQn );            // Limpar interrupção pedente (UM10398, 28.6.2.1, tabela 442)
          NVIC_SetPriority( EINT0_IRQn, 3 );       // Habilitar a prioridade da interrupção (a menor possível) (UM10398,
28.6.2.1, tabela 442)
          LPC_GPIO0->IS  &= ~(1 << PINO_BTN);      // para gerar interrupção por borda no pino PIO0_2 (UM10398, 12.3.3)
          LPC_GPIO0->IBE &= ~(1 << PINO_BTN);      // configurar a borda de ativação por meio do registro IEV (PIO0_2) (UM10398,
12.3.4)
          LPC_GPIO0->IEV = (1 << PINO_BTN);        // configurar a detecção de borda de subida (PIO0_2)  (UM10398, 12.3.5)
          LPC_GPIO0->IE  = (1 << PINO_BTN);        // ativar a detecção de interrupção para o pino PIO0_2 (UM10398, 12.3.6)

          //
          // Laço de repetição da aplicação
          //

          while (1){};

          return 0;
}
```

7.7 Integração com visor LCD de 16x2

main.c

```
/*
**
**                           Main.c
**
**
*********************************************************************/
/*

          Programa de exemplo de integração com visor LCD de 16x2 JHD162A
          em modo padrão de 8 bits

          Mapa de pinos para o LCD e o LPC1114FN28:
          - D0-D7   <-> GPIO1_0-GPIO1_7
```

```
          - E      <-> GPIO0_8
          - RS       <-> GPIO0_9
          - R/W      <-> GND

          O programa exibe a seguinte mensagem no visor:

          ELT031
          TESTE DO LCD


          Universidade Federal de Itajubá
          Prof. Rodrigo de Paula Rodrigues
          2015-04-19

*********************************************************************/


#include <LPC11xx.h>                            /* LPC11xx Peripheral Registers */
#include "system_LPC11xx.h"

// Definições de pinos e portas
#define LCD_PORTA_CTR                           LPC_GPIO0 // Utilizar pinos da porta GPIO0 para controlar o LCD (pinos RS e E)
#define LCD_PORTA_DADOS                         LPC_GPIO1 // Utilizar pinos da porta GPIO1 para enviar dados para o LCD
(barramento D0-D7)
#define LCD_PINO_CTR_E                          8                               // pino 8 da porta GPIO0
#define LCD_PINO_CTR_RS                         9                               // pino 9 da porta GPIO0
#define PINO_LED                                3                               // pino 3 da porta GPIO0


// Protótipos locais
void bit_set( volatile uint32_t *p_iValor, unsigned char v_ucBit );
void bit_clr( volatile uint32_t *p_iValor, unsigned char v_ucBit );
void delay_ms( unsigned int v_uiTempo );
void LCD_iniciar();
void LCD_comando( char v_cComando );
void LCD_caractere( char v_cCaractere );


// Código principal

int main(void)
{

          //
          // O procedimento SystemInit (CMSIS) configura o CLOCK principal
          // do dispositivo para 48 MHz a partir de um clock externo de
          // 12 MHz (cristal) e de seu circuito PLL interno.
          //
          // Para estipular outra configuração do clock principal,
          // altere as constantes SYSOSCCTRL_Val, WDTOSCCTRL_Val,
          // SYSPLLCTRL_Val, SYSPLLCLKSEL_Val, MAINCLKSEL_Val e
          // SYSAHBCLKDIV_Val presentes no arquivo system_LPC11xx.c
          //

          //
          // Configurar os demais periféricos
          //

          LPC_SYSCON->SYSAHBCLKCTRL |= (1<<6);                                          // habilitar o clock para o módulo
GPIO (UM10398, 3.5.14)
          LCD_PORTA_CTR->DIR              |= (1 << LCD_PINO_CTR_E);     // configurar como saídas digitais os pino de controle
do LCD
          LCD_PORTA_CTR->DIR              |= (1 << LCD_PINO_CTR_RS);
          LCD_PORTA_CTR->DIR              |= (1 << PINO_LED);
          LCD_PORTA_DADOS->DIR            |= 0xFF;                                                  // configurar como saídas
digitais os 8 primeiros pinos da porta de dados do LCD

          // Os pinos PIO1_0 a PIO1_3 não são configurados
          // para suas funções de entrada/saída digitais após um reinício.
          // Forçar suas operações como entrada/saída digitais.

          LPC_SYSCON->SYSAHBCLKCTRL |= (1<<16);                                         // habilitar o clock para o módulo
IOCON (UM10398, 3.5.14)
          LPC_IOCON->R_PIO1_0             = 0x81;
          LPC_IOCON->R_PIO1_1             = 0x81;
          LPC_IOCON->R_PIO1_2             = 0x81;
          LPC_IOCON->SWDIO_PIO1_3         = 0x81;


          // Sinalizar nível alto com o LED
          bit_set( &LCD_PORTA_CTR->DATA, PINO_LED );
          delay_ms(1000);

          // Configurar o LCD
          LCD_iniciar();

          // Enviar o texto "ELT031" para o visor LCD
          LCD_caractere('E');
          LCD_caractere('L');
          LCD_caractere('T');
          LCD_caractere('0');
          LCD_caractere('3');
          LCD_caractere('1');

          LCD_comando(0xC0);            // pular para a segunda linha

          // Enviar o texto "TESTE DO LCD"
          LCD_caractere('T');
          LCD_caractere('E');
          LCD_caractere('S');
```

```
          LCD_caractere('T');
          LCD_caractere('E');
          LCD_caractere(' ');
          LCD_caractere('D');
          LCD_caractere('O');
          LCD_caractere(' ');
          LCD_caractere('L');
          LCD_caractere('C');
          LCD_caractere('D');



          //
          // Laço de repetição da software
          //

          //
          // Manter o pino do LED piscando para sinalizar
          // que o embarcado está em operação correta
          //

          while (1)
          {
                    bit_clr( &LCD_PORTA_CTR->DATA, PINO_LED );
                    delay_ms(500);

                    bit_set( &LCD_PORTA_CTR->DATA, PINO_LED );
                    delay_ms(500);

          };

          return 0;
}

//
// Implementações locais
//

// bit_set

void bit_set( volatile uint32_t *p_uiValor, unsigned char v_ucBit )
{
          (*p_uiValor) |= (1 << v_ucBit);
}

// bit_clr

void bit_clr( volatile uint32_t *p_uiValor, unsigned char v_ucBit )
{
          (*p_uiValor) &= ~(1 << v_ucBit);
}


//
// delay_ms
//
// Rotina de atraso que leva em consideração um clock de 48 MHz
// para o núcleo
//
void delay_ms( unsigned int v_uiTempo )
{
          unsigned int vuiI1;

          do
          {
                              // Gerar um atraso aproximado a 1 milissegundo
                              for( vuiI1 = 0; vuiI1 < 3000; vuiI1++ );

                              v_uiTempo--;

          } while( v_uiTempo > 0 );

}



//
// LCD_iniciar
//
// Configurar a operação do LCD
//
void LCD_iniciar()
{

          bit_clr( &LCD_PORTA_CTR->DATA, LCD_PINO_CTR_E );
          bit_clr( &LCD_PORTA_CTR->DATA, LCD_PINO_CTR_RS );


          // Esperar pela inicialização do visor
          delay_ms(1000);

          // Enviar a sequência 0b0011****
          LCD_comando(0x30);
          delay_ms(5);

          // Enviar a sequência 0b0011****
          LCD_comando(0x30);
          delay_ms(5);

          // Enviar a sequência 0b0011****
```

```
          LCD_comando(0x30);
          delay_ms(5);

          // Requisitar modo de 8 bits, 2 linhas e fonte 5x10
          LCD_comando(0x3C);
          delay_ms(20);

          // Ligar o visor, ligar o cursor e colocá-lo piscando
          LCD_comando(0x0F);
          delay_ms(20);

          // Limpar o visor
          LCD_comando(0x01);
          delay_ms(20);

          // I/D = 1, S=0
          LCD_comando(0x06);
          delay_ms(20);
}

//
// LCD_comando
//
// Enviar um comando ao LCD
//
void LCD_comando(char v_cComando)
{
          // Sinalizar o envio de um comando
          bit_clr( &LCD_PORTA_CTR->DATA, LCD_PINO_CTR_RS );


          // Enviar o comando aos pinos de dados D0 a D7
          LCD_PORTA_DADOS->DATA = v_cComando;
          delay_ms(1);

          // Enviar o sinal de dados válidos (pulso positivo no pino E)
          bit_set( &LCD_PORTA_CTR->DATA, LCD_PINO_CTR_E );
          delay_ms(1);
          bit_clr( &LCD_PORTA_CTR->DATA, LCD_PINO_CTR_E );
}

//
// LCD_caractere
//
// Enviar um caractere ao LCD
//
void LCD_caractere( char v_cCaractere )
{
          // Sinalizar o envio de um caractere
          bit_set( &LCD_PORTA_CTR->DATA, LCD_PINO_CTR_RS );


          // Enviar o caractere aos pinos de dados D0 a D7
          LCD_PORTA_DADOS->DATA = v_cCaractere;
          delay_ms(1);

          // Enviar o sinal de dados válidos (pulso positivo no pino E)
          bit_set( &LCD_PORTA_CTR->DATA, LCD_PINO_CTR_E );
          delay_ms(1);
          bit_clr( &LCD_PORTA_CTR->DATA, LCD_PINO_CTR_E );
}
```

7.8 Integração com periférico SPI

main.c

```
/*
**
**                           Main.c
**
**
*********************************************************************/
/*

          Este programa exemplifica o uso do periférico de comunicação SPI
          para se comunicar com uma memória EEPROM 25AA256

          Obs:
          1 - o periférico SPI0 é configurado para trabalhar em 1 MHz;
          2 - o sinal de clock do periférico SPI0 foi alocado no pino PI0_6.

          Conexões dos pinos:
          LPC1114  SPI    25AA256
          PI0_2    -      /CS (1)
          PI0_6    SCK    SCK (6)
          PI0_8    MISO   SO  (2)
          PI0_9    MOSI   SI  (5)

    Os pinos /WS e /HOLD da memória foram colocados em nível alto (VCC)

          Universidade Federal de Itajubá
          Prof. Rodrigo de Paula Rodrigues
          2014-09-28
```

```
*****************************************************************/


#include <LPC11xx.h>                            /* LPC11xx Peripheral Registers */
#include "system_LPC11xx.h"

#define K_E2PROM_WRSR_CMD     0x01      // comando de escrita do registro de estado (25AA255, tabela 2-1)
#define K_E2PROM_WRITE_CMD    0x02      // comando de escrita de dados (25AA255, tabela 2-1)
#define K_E2PROM_READ_CMD     0x03      // comando de leitura de dados (25AA255, tabela 2-1)
#define K_E2PROM_WRDI_CMD     0x04      // comando de inibição de escrita de dados (25AA255, tabela 2-1)
#define K_E2PROM_RDSR_CMD     0x05      // comando de leitura do registro de estado (25AA255, tabela 2-1)
#define K_E2PROM_WREN_CMD     0x06      // comando de habilitação de escrita de dados (25AA255, tabela 2-1)

//
// Protótipos de funções
//

inline void SPI_CS_Alto()               { LPC_GPIO0->DATA |=  (1<<2); };
inline void SPI_CS_Baixo()              { LPC_GPIO0->DATA &= ~(1<<2); };

void SPI_Transmitir ( unsigned char valor  );
char SPI_Receber    ( unsigned char valor );
void SPI_Limpar_Recepcao();
char E2PROM_Disponivel();


//
// AQUI: Modificar o SSEL (PIO_2) para um pino de GPIO com controle via código.
//

int main(void)
{
          unsigned int        i;
          unsigned char       dado;

          //
          // O procedimento SystemIniti (CMSIS) configura o CLOCK principal
          // do dispositivo para 48 MHz a partir de um clock externo de
          // 12 MHz (cristal) e de seu circuito PLL interno.
          //
          // Para estipular outra configuração do clock principal,
          // altere as constantes SYSOSCCTRL_Val, WDTOSCCTRL_Val,
          // SYSPLLCTRL_Val, SYSPLLCLKSEL_Val, MAINCLKSEL_Val e
          // SYSAHBCLKDIV_Val presentes no arquivo system_LPC11xx.c
          //

          //
          // Configurar os demais periféricos
          //


          LPC_SYSCON->PRESETCTRL    |= 1;                           // garantir que o periférico SPI não esteja em estado de
reset (UM10398, 3.5.2)
          LPC_SYSCON->SYSAHBCLKCTRL |= (1<<11);   // habilitar o clock para o bloco SPI0 (UM10398, 3.5.14)


          LPC_SYSCON->SYSAHBCLKCTRL |= (1<<16);   // habilitar o clock para o bloco IOCON (UM10398, 3.5.14)

          LPC_IOCON->SCK_LOC              = 0x02;           // configurar o local do pino SCK0 como o pino PI0_6 (UM10398,
7.4.43)
          LPC_IOCON->PIO0_6         = 0x02;                 // configurar o pino como SCK0  em modo SPI (UM10398, 7.4.18)

          LPC_IOCON->PIO0_8         &= ~0x07;               // configurar o pino como MISO0 em modo SPI (UM10398, 7.4.23)
          LPC_IOCON->PIO0_8         |= 0x01;

          LPC_IOCON->PIO0_9         &= ~0x07;               // configurar o pino como MOSI0 em modo SPI (UM10398, 7.4.24)
          LPC_IOCON->PIO0_9         |= 0x01;

          //
          // O módulo SPI0 possui a função SSEL0 (CS de escolha do escravo)
          // que pode ser conectada a um pino do processador, mas seu comportamento
          // é reiniciado a cada nova transmissão. Como a memória E2PROM requer um
          // comportamento diferente de tal pino (/CS), será utilizado o PIO0_2 para
          // controlar o pino /CS da memória, mas de forma manual
          //

          LPC_SYSCON->SSP0CLKDIV          = 1;                      // Estipular o divisor do clock do módulo SSP0 para 1
(UM10398, 3.5.15)

          LPC_SSP0->CR0                             = 0x1707;       // configurar o periférico SSP0 (UM10398, 14.6.1)
                                                                                                                  // mode de 8 bits
(DDS = 7)
                                                                                                                  // modo SPI (FRF =
0)
                                                                                                                  // polaridade
(CPOL = 0) (modo spi 0,0)
                                                                                                                  // fase (CPHA = 0)
(modo spi 0,0)
                                                                                                                  // taxa de clock
serial de 23 (SCR = 23)

          LPC_SSP0->CPSR                            = 2;                    // pré-escalar de 2 (CPSDVSR = 2)(UM10398,
14.6.5)

                                                                                                                  // Obs: com o SCR
e CPSDVR escolhidos, o clock
                                                                                                                  // do periférico
SSP0 será de 1 MHz (48MHz / (2*(23+1))  (UM10398, 14.6.1, tabela 209, item SCR)
```

```
	LPC_SSP0->CR1 |= (1<<1);	// configurar o periférico SSP0 (UM10398, 14.6.2)
                                                                                                                    // loopback
desabilitado (LBM = 0)
                                                                                                                    // modo SPI (FRF =
0)
                                                                                                                    // ativar o módulo
(SSE = 1)
                                                                                                                    // modo mestre
(MS = 0)

	LPC_SYSCON->SYSAHBCLKCTRL |= (1<<6);	// habilitar o clock para o módulo GPIO (UM10398, 3.5.14)
	LPC_IOCON->PIO0_2 = 0x00;		// configurar a função de pino digital (UM10398, 7.4.6)
	LPC_GPIO0->DIR |= (1<<2);	// configurar o pino PIO0_2 como saída (UM10398, 12.3.2)
	SPI_CS_Alto();                                                  // estipular o estado inicial como alto
(UM10398, 12.3.1)

	//
	// Limpar, após a inicialização, o buffer fifo de recepção
	//
	SPI_Limpar_Recepcao();


	//
	// Desabilitar a proteção para todos os setores de memória (25AA256, 2.6)
	//

	SPI_CS_Baixo();                                                 // Iniciar a comunicação com a memória
	SPI_Transmitir( K_E2PROM_WRSR_CMD );
	SPI_Transmitir( 0x80 );
	SPI_CS_Alto();                                                  // Finalizar a comunicação com a memória

	//
	// Habilitar a escrita na memória EEPROM (25AA256, 2.4)
	//

	SPI_CS_Baixo();                                                 // Iniciar a comunicação com a memória
	SPI_Transmitir( K_E2PROM_WREN_CMD );
	SPI_CS_Alto();                                                  // Finalizar a comunicação com a memória

	// Ler o registro de estado da memória EEPROM

	SPI_CS_Baixo();                                                                 // Iniciar a comunicação com a memória
	SPI_Transmitir( K_E2PROM_RDSR_CMD  );   // enviar o comando de leitura e desprezar o retorno recebido pelo módulo SPI0
(25AA256, 2.5)
	dado = SPI_Receber( 0xFF );                                     // enviar um valor qualquer e receber o valor do registro
(25AA256, 2.5)
	SPI_CS_Alto();                                                                  // Finalizar a comunicação com a
memória

	//
	// Gravar o valor 33 na posição de memória 0x0000 da E2PROM
	//

	SPI_CS_Baixo();                                                                 // Iniciar a comunicação com a memória
	SPI_Transmitir( K_E2PROM_WRITE_CMD );   // Enviar o comando de escrita
	SPI_Transmitir( 0x00 );                                         // enviar a parte alta do endereço
	SPI_Transmitir( 0x00 );                                         // enviar a parte baixa do endereço
	SPI_Transmitir( 33   );                                         // enviar o valor a ser registrado
	SPI_CS_Alto();                                                                  // Finalizar a comunicação com a
memória

	while(1)
	{
		//
		// Esperar a memória ficar disponível
		//

		while( E2PROM_Disponivel() == 0 );
		//
		// Ler o conteúdo do endereço de memória 0x0000 da E2PROM
		//

		// Criar uma sequência de escrita
		SPI_CS_Baixo();                                                                         // Iniciar a comunicação com
a memória
		SPI_Transmitir( K_E2PROM_READ_CMD );    // enviar o comando de leitura
		SPI_Transmitir( 0x00 );                                                 // enviar a parte alta do endereço
		SPI_Transmitir( 0x00 );                                                 // enviar a parte baixa do endereço
		dado = SPI_Receber(0xFF );                                      // enviar um valor qualquer apenas para receber
o valor esperado
		SPI_CS_Alto();                                                                          // Finalizar a comunicação
com a memória

		for ( i = 0; i < 0xFFFFF; i++ );	// gerar um atraso
	}

	return 0;
}


//
// Transmitir um valor pelo dispositivo SPI0
// e desprezar o valor recebido
//

void SPI_Transmitir ( unsigned char valor )
{
	//
	// O periférico SSP0 possue um buffer FIFO de transmissão.
	// Sempre que houver espaço em tal buffer (bit TNF), pode-se
```

```
          //
          // Habilitar a escrita na memória EEPROM (25AA256, 2.4)
          //
          // requisitar o envio de um novo dado ao se escrever um valor
          // em seu registro DR. No entanto, a função implementada sumilará
          // que o buffer de transmissão possua apenas uma posição e
          // esperará pelo mesmo estar livre (bit TFE).
          //

          while ( (LPC_SSP0->SR & 0x01 ) == 0 );            // esperar pelo estado de buffer livre (bit TFE, UM10398, 14.6.4)

          LPC_SSP0->DR = valor;

          while ( (LPC_SSP0->SR & 0x14) != 0x04 );          // esperar pelo fim da transmissão e recepção correntes (bits BSY e
RNE, UM10398, 14.6.4)

          valor = LPC_SSP0->DR;                                                     // desprezar o valor recebido
}


//
// Transmitir o valor especificado como parâmetro
// e retornar a recepção atual
//

char SPI_Receber( unsigned char valor )
{

          //
          // O periférico SSP0 possue um buffer FIFO de transmissão.
          // Sempre que houver espaço em tal buffer (bit TNF), pode-se
          //
          // Habilitar a escrita na memória EEPROM (25AA256, 2.4)
          //
          // requisitar o envio de um novo dado ao se escrever um valor
          // em seu registro DR. No entanto, a função implementada sumilará
          // que o buffer de transmissão possua apenas uma posição e
          // esperará pelo mesmo estar livre (bit TFE).
          //

          while ( (LPC_SSP0->SR & 0x01 ) == 0 );            // esperar pelo estado de buffer livre (bit TFE, UM10398, 14.6.4)

          LPC_SSP0->DR = valor;

          while ( (LPC_SSP0->SR & 0x14) != 0x04 );          // esperar pelo fim da transmissão e recepção correntes (bits BSY e
RNE, UM10398, 14.6.4)

          valor = LPC_SSP0->DR;                                                     // recuperar o valor recebido pelo
periférico SSP0

          return valor;
}


//
// Limpar o buffer de recepção
//

void SPI_Limpar_Recepcao()
{
          char dado;

          while ( LPC_SSP0->SR & 0x04 ) // o bit RNE indica a presença de valores no buffer de recepção (UM10398, 14.6.4)
          {
                    dado = LPC_SSP0->DR;
          }

}


//
// Procedimento de verificação do estado de
// operação da E2PROM
//

char E2PROM_Disponivel()
{
          char dado;

          SPI_CS_Baixo();                                                           // Iniciar a comunicação com a memória
          SPI_Transmitir( K_E2PROM_RDSR_CMD  );   // enviar o comando de leitura e desprezar o retorno recebido pelo módulo SPI0
(25AA256, 2.5)
          dado = SPI_Receber( 0xFF );                               // enviar um valor qualquer e receber o valor do registro
(25AA256, 2.5)
          SPI_CS_Alto();                                                            // Finalizar a comunicação com a
memória

          // retornar o estado do bit WIP (25AA256, 2.5)
          return  ( !(dado & 0x01 ) );
}
```

7.9 Trabalho com temporizador (geração de atraso)

main.c

```
/*
**
**                           Main.c
**
**
*********************************************************************/
/*
   Last committed:     $Revision: 00 $
   Last changed by:    $Author: $
   Last changed date:  $Date:  $
   ID:                 $Id:  $

*********************************************************************/
/*

          Este programa exemplifica como utilizar o TMR de 16 bits
          para gerar uma função de atraso proporcional a 1ms.

          Universidade Federal de Itajubá
          Prof. Rodrigo de Paula Rodrigues
          2014-09-20

*********************************************************************/

#include <LPC11xx.h>                            /* LPC11xx Peripheral Registers */
#include "system_LPC11xx.h"

#define PINO_LED  3             // Pino GPIO0_3


void delay_ms( unsigned int tempo );

int main(void)
{

          //
          // O procedimento SystemInit (CMSIS) configura o CLOCK principal
          // do dispositivo para 48 MHz a partir de um clock externo de
          // 12 MHz (cristal) e de seu circuito PLL interno.
          //
          // Para estipular outra configuração do clock principal,
          // altere as constantes SYSOSCCTRL_Val, WDTOSCCTRL_Val,
          // SYSPLLCTRL_Val, SYSPLLCLKSEL_Val, MAINCLKSEL_Val e
          // SYSAHBCLKDIV_Val presentes no arquivo system_LPC11xx.c
          //


          //
          // Configurar os demais periféricos
          //


          LPC_SYSCON->SYSAHBCLKCTRL |= (1<<6);    // habilitar o clock para o módulo GPIO (UM10398, 3.5.14)
          LPC_GPIO0->DIR  |= (1 << PINO_LED);     // configurar o pino PTO0_3 como saída digital (UM10398, 12.3.2)

          //
          // Configuração do módulo TMR16 utilizada
          // pela função delay_ms
          //

          LPC_SYSCON->SYSAHBCLKCTRL |= (1<<7);    // habilitar o clock para o bloco TMR16 (UM10398, 3.5.14)
          LPC_TMR16B0->MR0          = 0xBB80;     // registro de correspondência em 48000 (UM10398, 18.7.7)
          LPC_TMR16B0->MCR          |= (1<<1);    // ao ocorrer uma correspondência, o TMR16 deve reiniciar (UM10398, 18.7.6)
          LPC_TMR16B0->CTCR         =  0x00;      // habilitar o modo temporizador do TMR16 (UM10398, 18.7.11)



          //
          // Laço de repetição da aplicação
          //

          while(1)
          {

                    // Estipular nível de saída alto (UM10398, 12.3.1)
                    LPC_GPIO0->DATA |= (1 << PINO_LED);

                    delay_ms( 1000 );



                    // Estipular nível de saída baixo (UM10398, 12.3.1)
                    LPC_GPIO0->DATA &= ~(1 << PINO_LED);

                    delay_ms( 1000 );

          };

          return 0;
}


//
```

```
// O procedimento utiliza o TMR de 16 bits para gerar atrasos
// múltiplos de 1ms
//
// É necessário que:
// -- o clock para o módulo TMR16 seja de 48MHz;
// -- o registro de correspondência esteja em 48000 (equivalente a 1ms);
// -- o timer seja reiniciado quando ocorrer uma correspondência.
//

void delay_ms( unsigned int tempo )
{

          LPC_TMR16B0->PR  = (0xFFFF & tempo);     // estipular o pré-escalar do TMR16 para a quantidade de milissegundos a
esperar (UM10398, 18.7.4).
          LPC_TMR16B0->TCR  = 0x01;                // habilitar o TMR16 (bit CEn, UM10398, 18.7.2)

          LPC_TMR16B0->EMR  = ~(0x01);             // limpar o sinal de estouro do temporizador (bit EM0, UM10398, 18.7.10)

          // esperar pelo estouro do timer (bit EM0, UM10398, 18.7.10)
          while ( !( LPC_TMR16B0->EMR & 0x01 ) );

          LPC_TMR16B0->TCR  = 0x00;                // desabilitar o TMR16 (bit CEn, UM10398, 18.7.2)
}
```

7.10 Trabalho com temporizador (função de correspondência)

main.c

```
/*
**
**                           Main.c
**
**
*********************************************************************/
/*
   Last committed:     $Revision: 00 $
   Last changed by:    $Author: $
   Last changed date:  $Date:  $
   ID:                 $Id:  $

*********************************************************************/
/*

          Este programa exemplifica como utilizar o TMR de 16 bits
          em sua função de correspondência para alternar o estado
          lógico de saída do PIO0_8 a cada 1s.

          Universidade Federal de Itajubá
          Prof. Rodrigo de Paula Rodrigues
          2014-09-20

*********************************************************************/

#include <LPC11xx.h>                            /* LPC11xx Peripheral Registers */
#include "system_LPC11xx.h"

int main(void)
{

          //
          // O procedimento SystemInit (CMSIS) configura o CLOCK principal
          // do dispositivo para 48 MHz a partir de um clock externo de
          // 12 MHz (cristal) e de seu circuito PLL interno.
          //
          // Para estipular outra configuração do clock principal,
          // altere as constantes SYSOSCCTRL_Val, WDTOSCCTRL_Val,
          // SYSPLLCTRL_Val, SYSPLLCLKSEL_Val, MAINCLKSEL_Val e
          // SYSAHBCLKDIV_Val presentes no arquivo system_LPC11xx.c
          //


          //
          // Configurar os demais periféricos
          //


          LPC_SYSCON->SYSAHBCLKCTRL |= (1<<16);   // habilitar o clock para o bloco IOCON (UM10398, 3.5.14)
          LPC_IOCON->PIO0_8          = 0x02;      // configurar o pino PIO0_8 para a função de correspondência do temporizador
TMR16 (UM10398, 7.4.23)
                                                                                                          // FUNC=2 e MODE=0


          LPC_SYSCON->SYSAHBCLKCTRL |= (1<<7);    // habilitar o clock para o bloco TMR16 (UM10398, 3.5.14)

          LPC_TMR16B0->PR           = 0x02DC;         // estipular o pré-escalar do TMR16 para 732 (UM10398, 18.7.4).
                                                                                                          // para um clock
principal de 48 MHz, um pré-escalar de 732 gera
                                                                                                          // aproximadamente
1s para o estouro do temporizador

          LPC_TMR16B0->MR0          = 0xFFFF;     // registro de correspondência (UM10398, 18.7.7)
```

```
        LPC_TMR16B0->MCR          = (1<<1);      // ao ocorrer uma correspondência, o TMR16 deve reiniciar (UM10398, 18.7.6)
        LPC_TMR16B0->EMR          = (3<<4);      // alternar o valor da saída PIO0_8 (EMC0=3) ao ocorrer uma correspondência
(UM10398, 18.7.10)
        LPC_TMR16B0->CTCR         =  0x00;       // habilitar o modo temporizador do TMR16 (UM10398, 18.7.11)

        LPC_TMR16B0->TCR          |= 0x01;       // habilitar o TMR16 (bit CEn, UM10398, 18.7.2)


        //
        // Laço de repetição da aplicação
        //

        while(1){};

        return 0;
}
```