Treinamento FreeRTOS

Esse texto ilustra os laboratórios que serão realizados durante o treinamento de FreeRTOS no Masters Brasil 2016.

Sumário

[2 Instalação da IDE MPLAB X, Compilador XC32 e Harmony 2](#_Toc527356272)

[3 Meu primeiro projeto com FreeRTOS 2](#_Toc527356273)

[3.1 Criando o projeto no MPLABX 2](#_Toc527356274)

[3.2 Adiconando os fontes do FreeRTOS 3](#_Toc527356275)

[3.3 Adicionando uma task 11](#_Toc527356276)

[3.3.1 Resposta: 11](#_Toc527356277)

[3.4 Manipulando prioridades 12](#_Toc527356278)

[3.4.1 Resposta 12](#_Toc527356279)

[3.5 13](#_Toc527356280)

[3.6 Utilizando delays (extra) 13](#_Toc527356281)

[3.6.1 Resposta 15](#_Toc527356282)

[4 Manipulando dados com filas 16](#_Toc527356283)

[4.1 Expandindo para duas chaves e dois leds 17](#_Toc527356284)

[4.2 Crie uma fila para enviar valores de sinal analógico (extra) 19](#_Toc527356285)

[4.2.1 Resposta 19](#_Toc527356286)

[5 Lab 3 - Semáforos 23](#_Toc527356287)

[5.1 Utilizando recursos compartilhados 23](#_Toc527356288)

[5.2 Solucionando o compartilhamento 24](#_Toc527356289)

[5.2.1 Resposta 24](#_Toc527356290)

[6 Desenvolvendo uma aplicação com requisitos de tempo real (PID) 25](#_Toc527356291)

[6.1 Configurando os periféricos 25](#_Toc527356292)

[6.2 Criando tarefas para rastreamento de referência e execução de controle em malha aberta 27](#_Toc527356293)

[6.3 Adicionando um controle PID 29](#_Toc527356294)

# Instalação da IDE MPLAB X, Compilador XC32 e Harmony

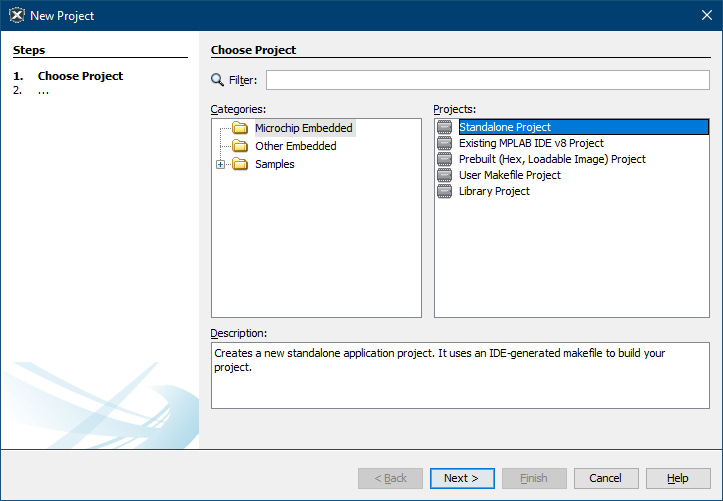
Primeiramente é necessário fazer o download e instalar todos os softwares necessários para que se possa usar o Harmony. Para esse texto, utilizei o Windows 8. Baixe e Instale o MPLAB X IDE. Para isso, [acesse esse link](http://www.microchip.com/mplabx-ide-windows-installer). Estamos usando a versão v4.01. Após a instalação do MPLABX, baixe e instale o Compilador XC32. Para baixar o compilador para o Windows, [acesse esse link](http://www.microchip.com/mplabxc32windows) e siga as instruções para instalação. Foi utilizado a versão 1.44 do XC32 para essas experiências. O código base do FreeRTOS 9.0.0 pode ser encontrado [aqui](http://www.freertos.org/a00104.html).

# Meu primeiro projeto com FreeRTOS

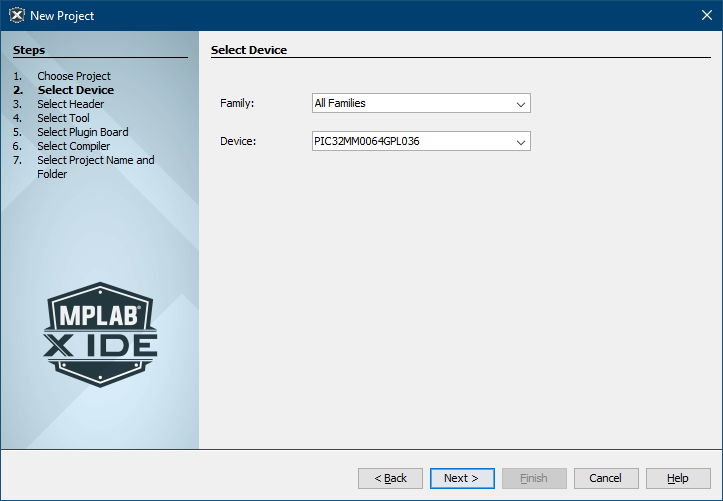
Nesta primeira etapa iremos criar um novo projeto com FreeRTOS partindo do zero.

## Criando o projeto no MPLABX

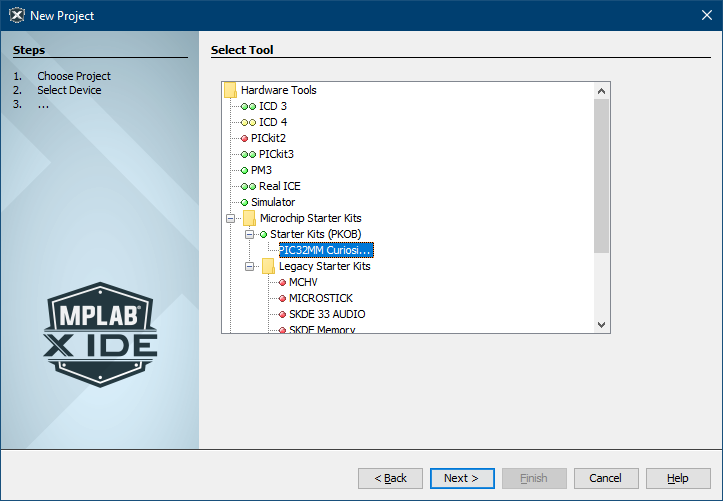
Os primeiros passos são similares a qualquer projeto com o MPLABX. De início vá até a opção File > New Project.



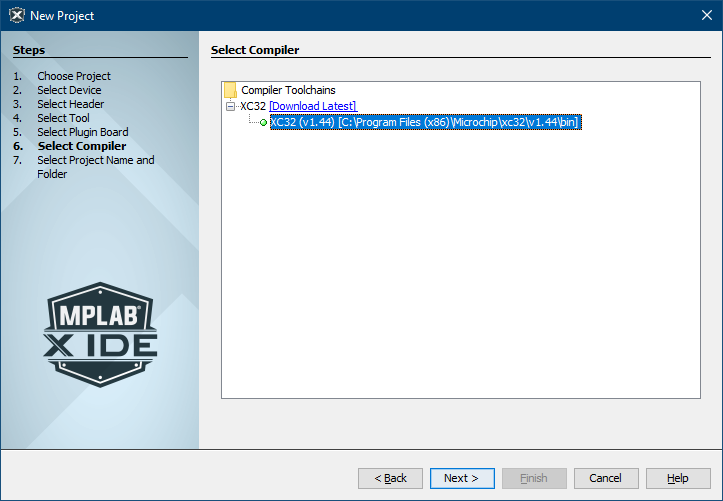
Escolha o dispositivo PIC32MM0064GPL036



Conecte a Placa PIC32MM Curiosity Development Board ao computador e selecione o Starter Kit da Curiosity



Será utilizado o XC32 como compilador do projeto.

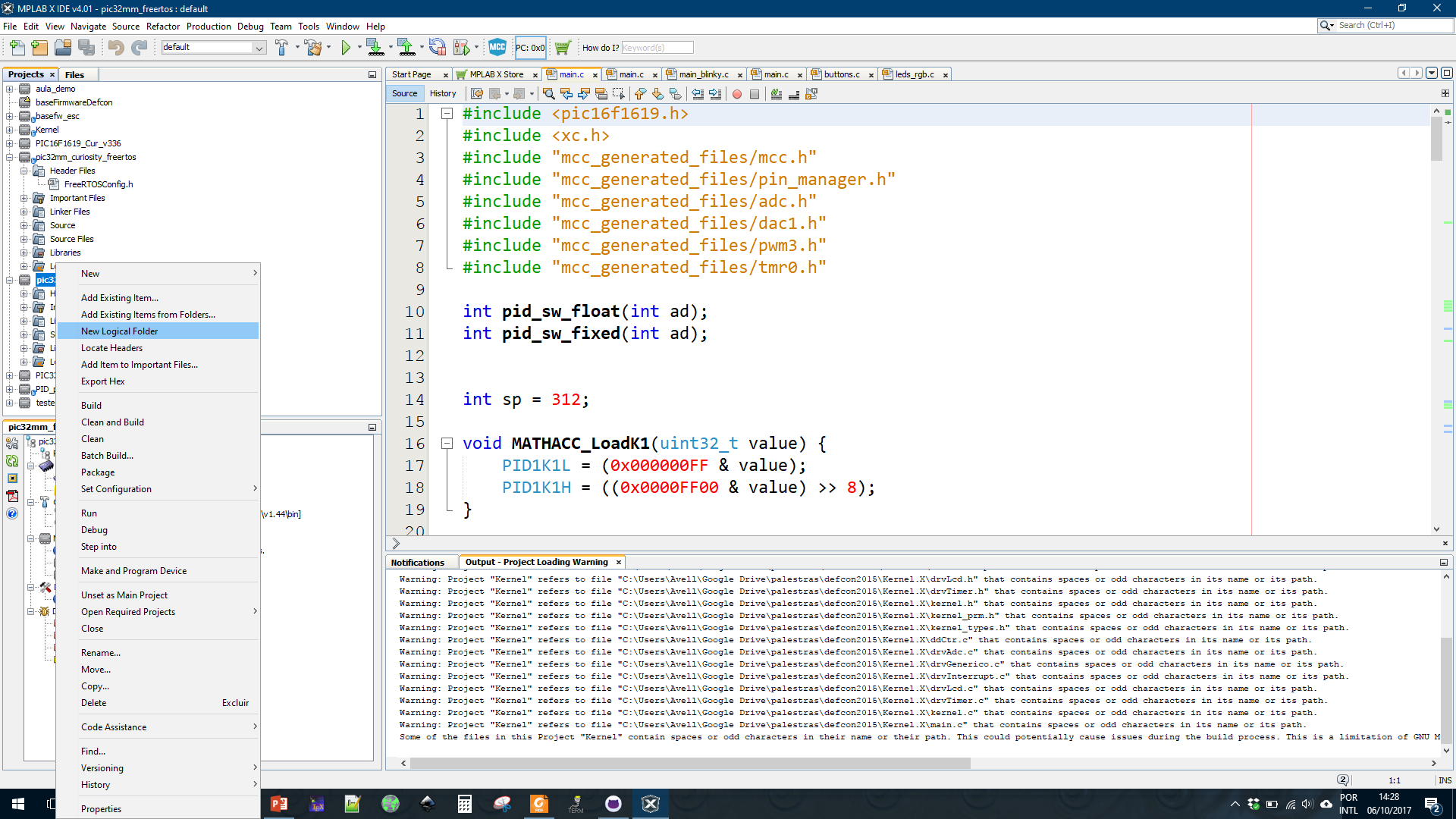


Dê um nome do projeto e salve.

## Adiconando os fontes do FreeRTOS

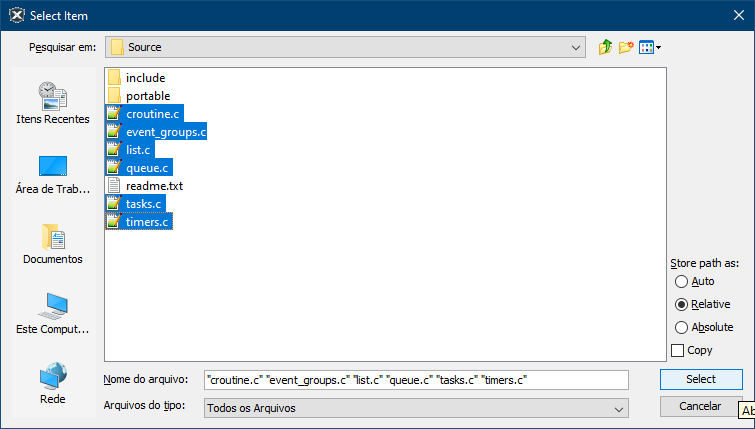
Agora vamos adicionar os arquivos do FreeRTOS na placa. O primeiro passo é copiar a pasta Source do FreeRTOS para dentro do seu projeto. Faça isso utilizando o Windows Explorer. Esta pasta está dentro da pasta FreeRTOSv9.0.0->FreeRTOS->Source.

De volta ao MPLABX, clique com o botão direito e adicione um logical folder com o nome FreeRTOS (apenas para organização no projeto)



Crie mais três logical folders dentro da Pasta FreeRTOS: include, portable e source.

Clique com o botão direito no diretório source e selecione a opção add existing items nas pastas criadas.



Adicione os arquivos da pasta Source, copiada pelo Windows Explorer para o diretório do seu projeto, dentro das pastas lógicas da seguinte forma:

Arquivos da raiz(Source) para a pasta lógica source;

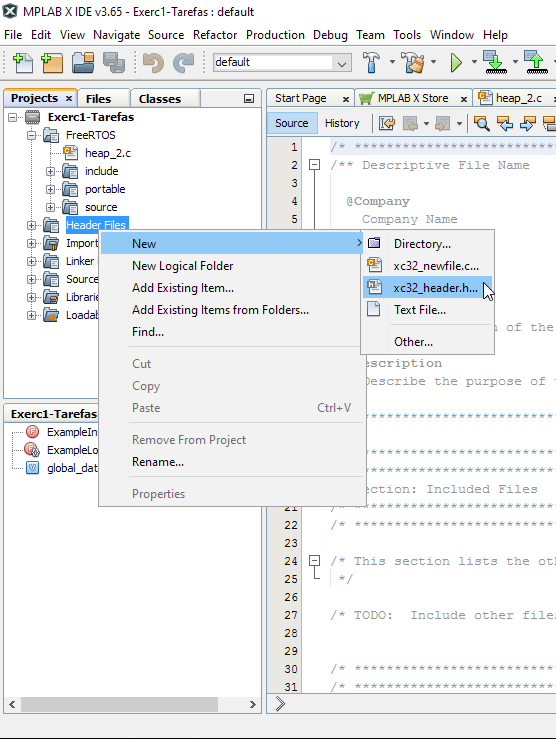
Arquivos da Source\include na pasta lógica include;

Arquivos da Source\portable\MPLAB\PIC32MX na pasta lógica portable;

E o arquivo Source\portable\MemMang\heap2.c para a pasta lógica FreeRTOS.

Obs.: Não iremos criar um arquivo main.c pois ele será gerado automaticamente pelo MCC.

Clique com o botão direito na pasta lógica Header Files, e adicione um novo arquivo de cabeçalho com o nome FreeRTOSConfig.h.

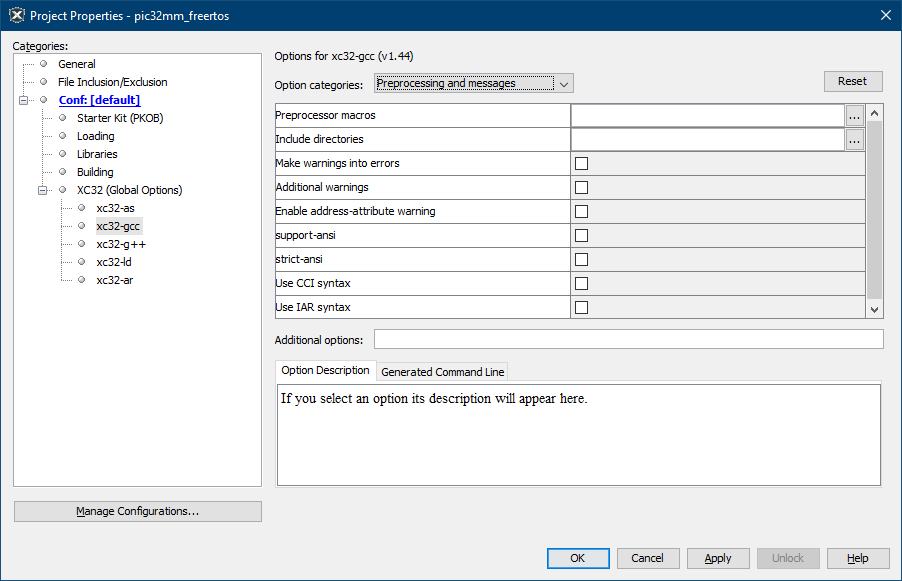


Copie e cole no arquivo o seguinte código:

|  |
| --- |
| /\*  FreeRTOS V10.0.1 - Copyright (C) 2018 Real Time Engineers Ltd.  All rights reserved  This file is modified from the example from FreeRTOS distribution.  \*/  #ifndef FREERTOS\_CONFIG\_H  #define FREERTOS\_CONFIG\_H  /\*-----------------------------------------------------------  \* Application specific definitions.  \*  \* These definitions should be adjusted for your particular hardware and  \* application requirements.  \*----------------------------------------------------------\*/  #define configUSE\_PREEMPTION 1  #define configUSE\_PORT\_OPTIMISED\_TASK\_SELECTION 1  #define configUSE\_IDLE\_HOOK 0  #define configUSE\_TICK\_HOOK 0  #define configTICK\_RATE\_HZ ( ( TickType\_t ) 1000 )  #define configCPU\_CLOCK\_HZ ( 8000000UL )  #define configPERIPHERAL\_CLOCK\_HZ ( 8000000UL )  #define configMAX\_PRIORITIES ( 5UL )  #define configMINIMAL\_STACK\_SIZE ( 100 )  #define configISR\_STACK\_SIZE ( 200 )  #define configTOTAL\_HEAP\_SIZE ( ( size\_t ) 4000 )  #define configMAX\_TASK\_NAME\_LEN ( 8 )  #define configUSE\_TRACE\_FACILITY 0  #define configUSE\_16\_BIT\_TICKS 0  #define configIDLE\_SHOULD\_YIELD 1  #define configUSE\_MUTEXES 1  #define configCHECK\_FOR\_STACK\_OVERFLOW 0  #define configQUEUE\_REGISTRY\_SIZE 0  #define configUSE\_RECURSIVE\_MUTEXES 1  #define configUSE\_MALLOC\_FAILED\_HOOK 0  #define configUSE\_APPLICATION\_TASK\_TAG 0  #define configUSE\_COUNTING\_SEMAPHORES 1  #define configGENERATE\_RUN\_TIME\_STATS 0  /\* Co-routine definitions. \*/  #define configUSE\_CO\_ROUTINES 0  #define configMAX\_CO\_ROUTINE\_PRIORITIES ( 2 )  /\* Software timer definitions. \*/  #define configUSE\_TIMERS 1  #define configTIMER\_TASK\_PRIORITY ( 2 )  #define configTIMER\_QUEUE\_LENGTH 5  #define configTIMER\_TASK\_STACK\_DEPTH ( configMINIMAL\_STACK\_SIZE \* 2 )  /\* Set the following definitions to 1 to include the API function, or zero  to exclude the API function. \*/  #define INCLUDE\_vTaskPrioritySet 1  #define INCLUDE\_uxTaskPriorityGet 1  #define INCLUDE\_vTaskDelete 1  #define INCLUDE\_vTaskCleanUpResources 0  #define INCLUDE\_vTaskSuspend 1  #define INCLUDE\_vTaskDelayUntil 1  #define INCLUDE\_vTaskDelay 1  #define INCLUDE\_uxTaskGetStackHighWaterMark 1  #define INCLUDE\_eTaskGetState 1  /\* The priority at which the tick interrupt runs. This should probably be  kept at 1. \*/  #define configKERNEL\_INTERRUPT\_PRIORITY 0x01  /\* The maximum interrupt priority from which FreeRTOS.org API functions can  be called. Only API functions that end in ...FromISR() can be used within  interrupts. \*/  #define configMAX\_SYSCALL\_INTERRUPT\_PRIORITY 0x03  #endif /\* FREERTOS\_CONFIG\_H \*/ |

Agora vamos ajustar os caminhos de compilação e do assemblador.

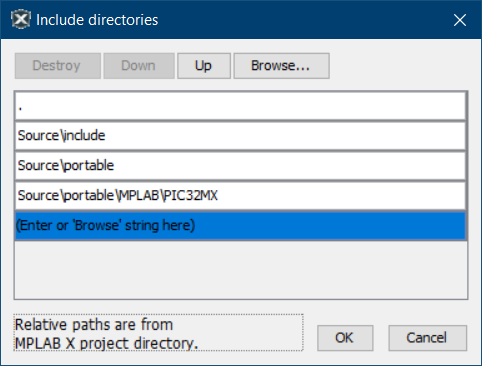
Clique com o botão direito no nome do projeto e abra a janela de propriedades. Selecione o compilador xc32-gcc, e em options categories selecione Preprocessing and messages.



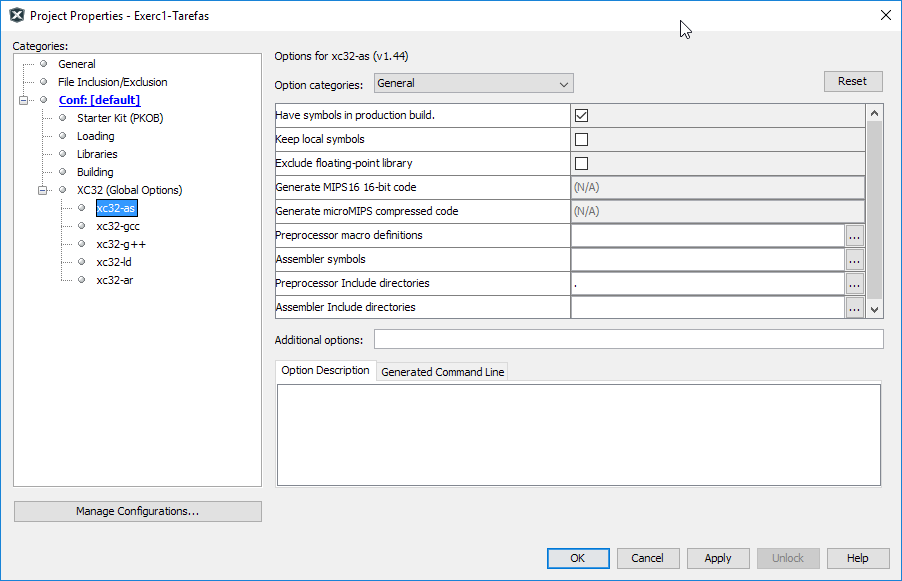
Vá até a opção include directories adicione quatro diretórios (utilizar o botão browse para os três últimos):

* **. (apenas um ponto)**
* **include**
* **portable**
* **portable\MPLAB\PIC32MX**

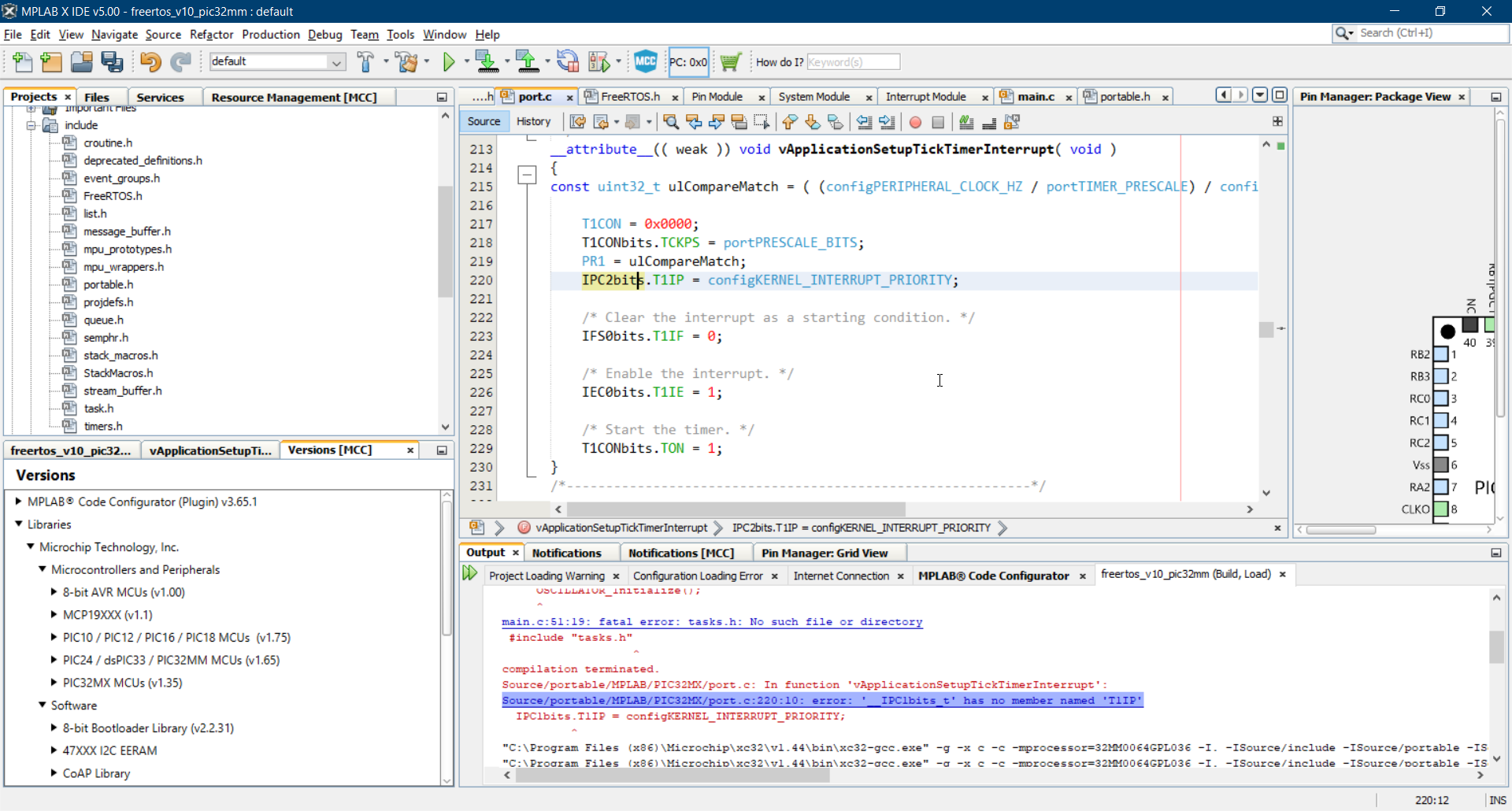
A estrutura deve ficar como na figura a seguir.



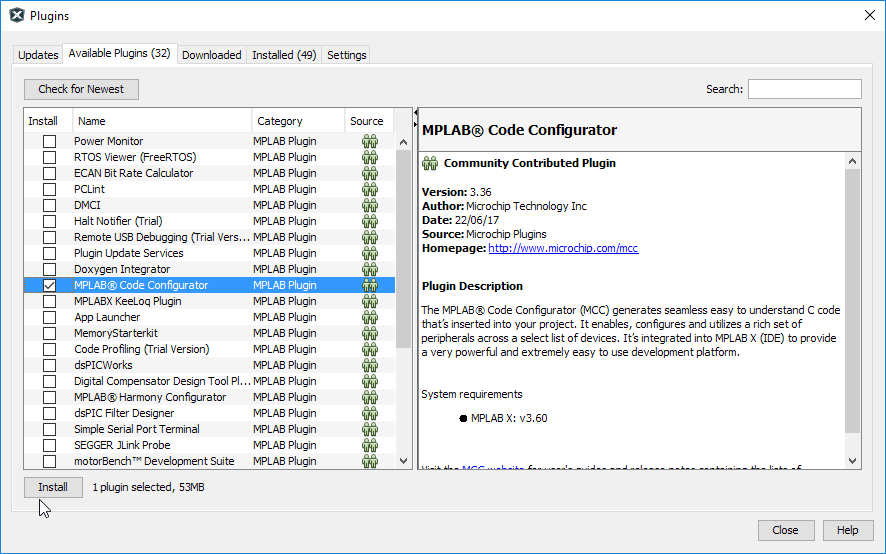
Para o assemblador **xc32-as** precisamos apenas incluir a pasta raiz: .(ponto), no Preprocessor Include directories.. Segue a imagem final:



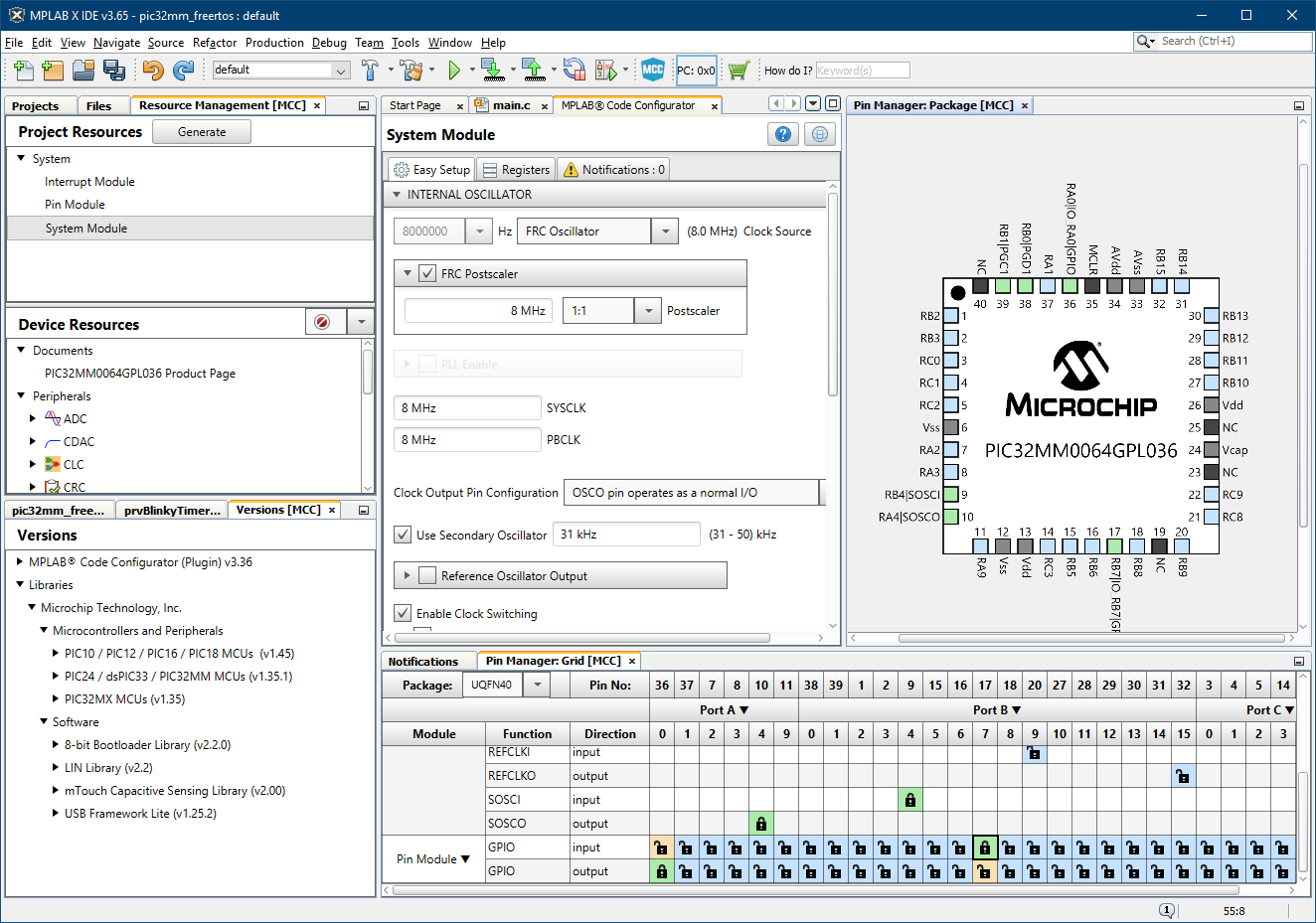
Agora vamos alterar o arquivo port.c para que funcione de acordo com a arquitetura MM. Na arquitetura MX o registro que controla o timer utiliza o registro IPC1bits, na arquitetura MM esse novo registro mudou para IPC2bits. A modificação é na linha 220 do arquivo port.c, conforme imagem a seguir.



Para simplificar o acesso aos periféricos do micro, que não são o foco deste workshop, iremos utilizar o MPLAB Code Configurator®. Verifique se ele está instalado, se não vá em Tools->Plugins e selecione a caixa referente ao MPLAB Code Configurator->Install. Prossiga com toda instalação e reinicie o MPLABX.



Abra o MCC com o ícone azul da barra de tarefas. A tela inicial do MCC é apresentada a seguir:



Vá na seção inferior (Pin Manager Grid) e configurar **RA0** como saída, e **RB7** como entrada. Note que o cadeado muda a cor do fundo de azul para verde, além de desabilitar a opção complementar (amarelo)

Clique no botão **Generate**, e salve o projeto.

O MCC irá criar o arquivo main.c e as várias bibliotecas dos periféricos necessários à configuração da placa.

Como exemplo inicial, implemente o restante do main com o seguinte código:

|  |
| --- |
| #include "FreeRTOS.h"  #include "task.h"  static void task1**(**void **\***pvParameters**)** **{**  **for(;;){**  **if(**IO\_RB7\_GetValue**()){**  IO\_RA0\_SetLow**();**  **}else{**  IO\_RA0\_SetHigh**();**  **}**  **}**  **}**  void main**(**void**)** **{**  SYSTEM\_Initialize**();**  xTaskCreate**(**task1**,** /\* The function that implements the task. \*/  "Rx"**,** /\* The text name assigned to the task - for debug only. \*/  configMINIMAL\_STACK\_SIZE**,** /\* The size of the stack to allocate to the task. \*/  **NULL,** /\* The parameter passed to the task - just to check functionality. \*/  tskIDLE\_PRIORITY**+**1**,** /\* The priority assigned to the task. \*/  **NULL);**  /\* Start the tasks and timer running. \*/  vTaskStartScheduler**();**  **}** |

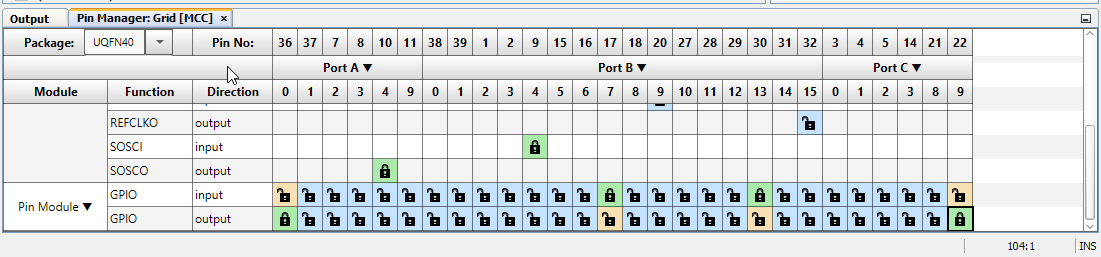
LAB1

## Adicionando uma task

Modifique o projeto e adicione uma segunda task que leia a chave 2(RB13) e acenda o led 2(RC9). Para isso abra o MCC, escolha os respectivos pinos como entrada e saída e crie a task 2 no main.c.

### Resposta:

Escolha dos terminais



Clicar em generate.

A adicionar o código no arquivo main.c para criar a nova task:

|  |
| --- |
| /\*  \* Cria a segunda task e utiliza como entrada RB13 e saída RC9  \*/  static void task2**(**void **\***pvParameters**)** **{**  **for(;;){**  **if(**IO\_RB13\_GetValue**()){**  IO\_RC9\_SetLow**();**  **}else{**  IO\_RC9\_SetHigh**();**  **}**  **}**  **}** |

Por fim modifique o código da função main adicionando a criação da task:

|  |
| --- |
| /\*  \* Adiciona a task 2 no kernel.  \*/  xTaskCreate**(**task2**,** /\* The function that implements the task. \*/  "LigaLED2"**,** /\* The text name assigned to the task - for debug only as it is not used by the kernel. \*/  configMINIMAL\_STACK\_SIZE**,** /\* The size of the stack to allocate to the task. \*/  **NULL,** /\* The parameter passed to the task - just to check the functionality. \*/  tskIDLE\_PRIORITY**+**1**,** /\* The priority assigned to the task. \*/  **NULL);** |

LAB2

## Manipulando prioridades

Modifique o código anterior para que uma das chaves tenha uma prioridade maior.

|  |
| --- |
| xTaskCreate**(**task2**,** "LigaLED2"**,** configMINIMAL\_STACK\_SIZE**, NULL,**  tskIDLE\_PRIORITY**+**2**,**  **NULL);** |

* O que acontece?
* Como resolver?

Uma alternativa é utilizar um delay para que a outra tarefa tenha oportunidade de rodar.

### Resposta

Resolvendo o problema de prioridades:

|  |
| --- |
| /\*  \* Task com token para detecção de borda  \* e acionamento mutuo das tasks, mesmo com prioridade diferente.  \* Lembrar que o vTaskDelay coloca a task em estado blocked.  \*/  static void task2**(**void **\***pvParameters**)** **{**  bool token **=** false**;**  **for** **(;;)** **{**  **if** **(**token **==** **!**IO\_RB13\_GetValue**())** **{**  vTaskDelay**(**20**/**portTICK\_PERIOD\_MS**);**  **}else{**  token **=** **!**IO\_RB13\_GetValue**();**  IO\_RC9\_Toggle**();**  **}**  **}**  **}** |

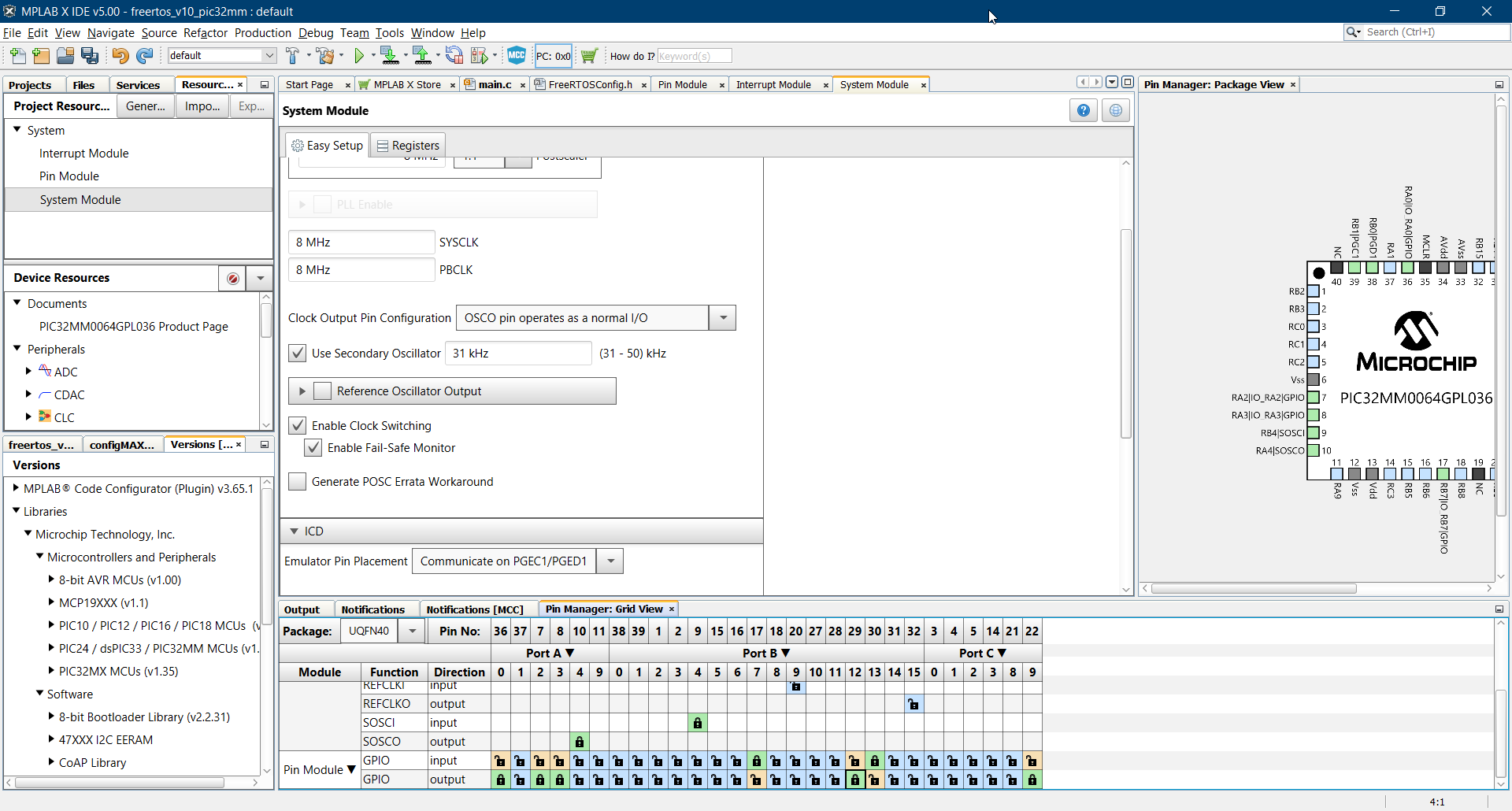
## 

LAB3

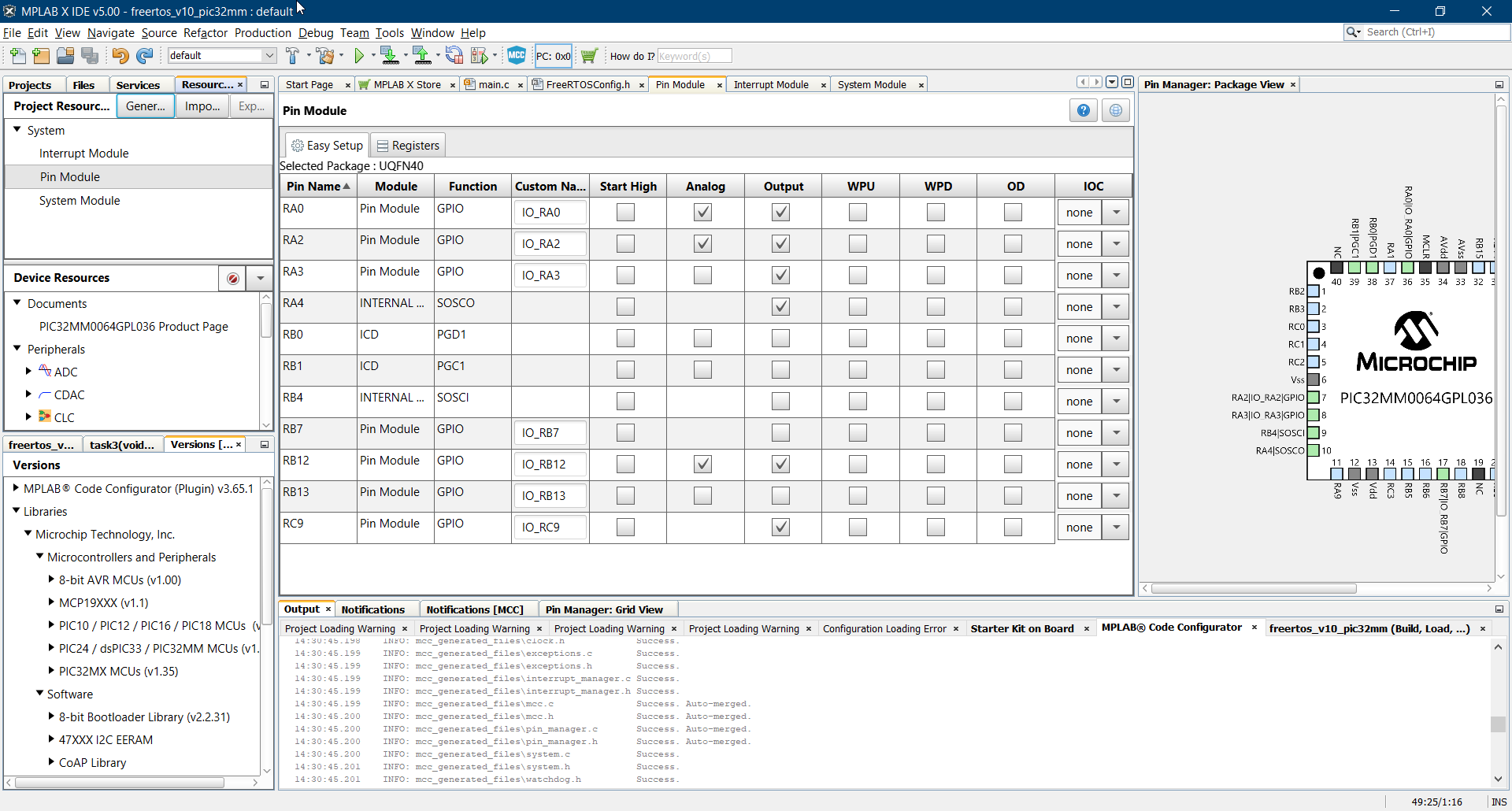
## Utilizando delays (extra)

Utilize as funções timerDelay, timerDelayUntil e crie um semáfaro.

Antes configure os terminais RA3, RB12 e RA3 como saída no MCC. Para usar o RA3 é necessário desliga-lo como CLKO.



Caso o MCC indique algum erro é provável que seja um problema de configuração no registro de Pin Module.



Lembre-se por fim de adicionar a task no programa principal.

### Resposta

|  |
| --- |
| /\*  \* Task criando o semáforo que troca de cor a cada 200 ciclos de clock  \*/  static void task3**(**void **\***pvParameters**)** **{**  TickType\_t ultimaChamada**;**  ultimaChamada **=** xTaskGetTickCount**();**    **for** **(;;)** **{**  IO\_RA3\_SetHigh**();**  IO\_RB12\_SetLow**();**  vTaskDelayUntil**(&**ultimaChamada**,** 3500 **/** portTICK\_RATE\_MS**);**  IO\_RA3\_SetHigh**();**  IO\_RB12\_SetHigh**();**  vTaskDelayUntil**(&**ultimaChamada**,** 1000 **/** portTICK\_RATE\_MS**);**  IO\_RA3\_SetLow**();**  IO\_RB12\_SetHigh**();**  vTaskDelayUntil**(&**ultimaChamada**,** 3000 **/** portTICK\_RATE\_MS**);**  **}**  **}** |

# Manipulando dados com filas

Continuaremos com o projeto anterior ou utilizando o projeto resolução do lab 1. Sinta-se a vontade para remover/comentar as tasks anteriores e os comandos de adição no main.

Utilizando fila para comandar dispositivos digitais

Insira o seguinte código, compile, grave e observe como funciona o processo de queue no FreeRTOS.

|  |
| --- |
| enum **{**  LED\_STATE\_OFF**,** LED\_STATE\_ON  **};**  #include "queue.h"  **typedef** struct **{**  unsigned int num**;** // 1, 2  unsigned int status**;** // LED\_STATE\_OFF, LED\_STATE\_ON  **}** LedInfo**;**  QueueHandle\_t ledQueue**;**  void taskLeds**(**void **\***pvParameters**)** **{**  LedInfo led**;**  **for** **(;;)** **{**  xQueueReceive**(**ledQueue**,** **&**led**,** portMAX\_DELAY**);**  **if** **(**led**.**num **==** 1**)** **{**  **if** **(**led**.**status **==** LED\_STATE\_ON**)** **{**  IO\_RA0\_SetHigh**();**  **}** **else** **{**  IO\_RA0\_SetLow**();**  **}**  **}**  **}**  **}**  void taskButtons**(**void **\***pvParameters**)** **{**  LedInfo led**;**  int num **=** 1**;**  **for** **(;;)** **{**  led**.**num **=** 1**;**  **if** **(!**IO\_RB7\_GetValue**())** **{**  led**.**status **=** LED\_STATE\_ON**;**  **}** **else** **{**  led**.**status **=** LED\_STATE\_OFF**;**  **}**  xQueueSend**(**ledQueue**,** **&**led**,** portMAX\_DELAY**);**  **}**  **}** |

Lembre-se de adicionar as tasks e criar a queue dentro do main.

Resposta

xTaskCreate(taskLeds, /\* The function that implements the task. \*/

"Led", /\* The text name assigned to the task - for debug only. \*/

configMINIMAL\_STACK\_SIZE, /\* The size of the stack to allocate to the task. \*/

NULL, /\* The parameter passed to the task - just to check functionality. \*/

tskIDLE\_PRIORITY + 1, /\* The priority assigned to the task. \*/

NULL);

/\*

\* Adiciona a task 2 no kernel.

\*/

xTaskCreate(taskButtons, /\* The function that implements the task. \*/

"Button", /\* The text name assigned to the task - for debug only as it is not used by the kernel. \*/

configMINIMAL\_STACK\_SIZE, /\* The size of the stack to allocate to the task. \*/

NULL, /\* The parameter passed to the task - just to check the functionality. \*/

tskIDLE\_PRIORITY + 2, /\* The priority assigned to the task. \*/

NULL);

ledQueue = xQueueCreate(3, sizeof(LedInfo));

## Expandindo para duas chaves e dois leds

Altere o código para incluir também o acionamento da segunda chave.

void taskLeds(void \*pvParameters) {

LedInfo led;

for (;;) {

xQueueReceive(ledQueue, &led, portMAX\_DELAY);

if (led.num == 1) {

if (led.status == LED\_STATE\_ON) {

IO\_RA0\_SetHigh();

} else {

IO\_RA0\_SetLow();

}

}

if (led.num == 2) {

if (led.status == LED\_STATE\_ON) {

IO\_RC9\_SetHigh();

} else {

IO\_RC9\_SetLow();

}

}

}

}

void taskButtons(void \*pvParameters) {

LedInfo led;

int num = 1;

for (;;) {

led.num = 1;

if (!IO\_RB7\_GetValue()) {

led.status = LED\_STATE\_ON;

} else {

led.status = LED\_STATE\_OFF;

}

xQueueSend(ledQueue, &led, portMAX\_DELAY);

led.num = 2;

if (!IO\_RB13\_GetValue()) {

led.status = LED\_STATE\_ON;

} else {

led.status = LED\_STATE\_OFF;

}

xQueueSend(ledQueue, &led, portMAX\_DELAY);

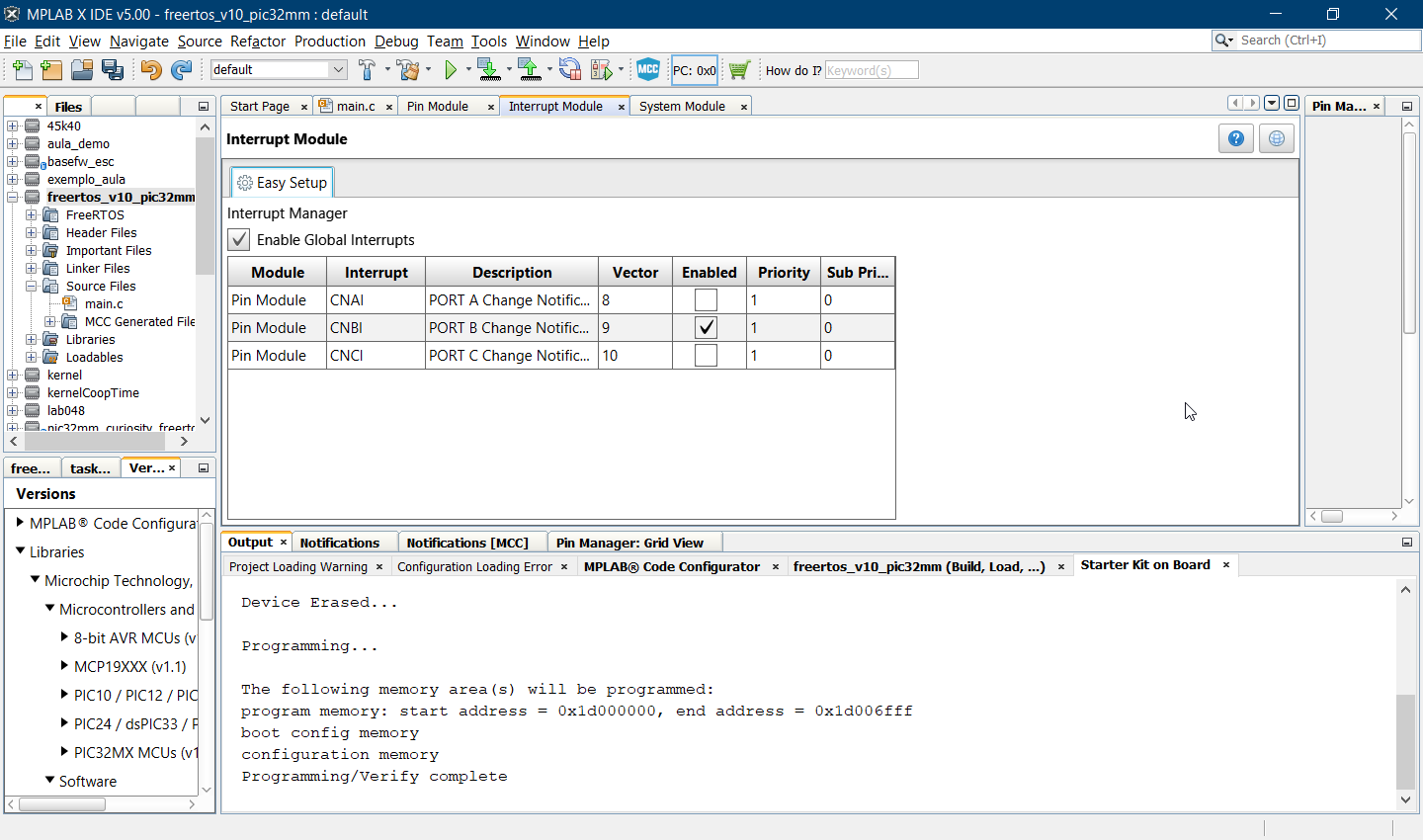
}

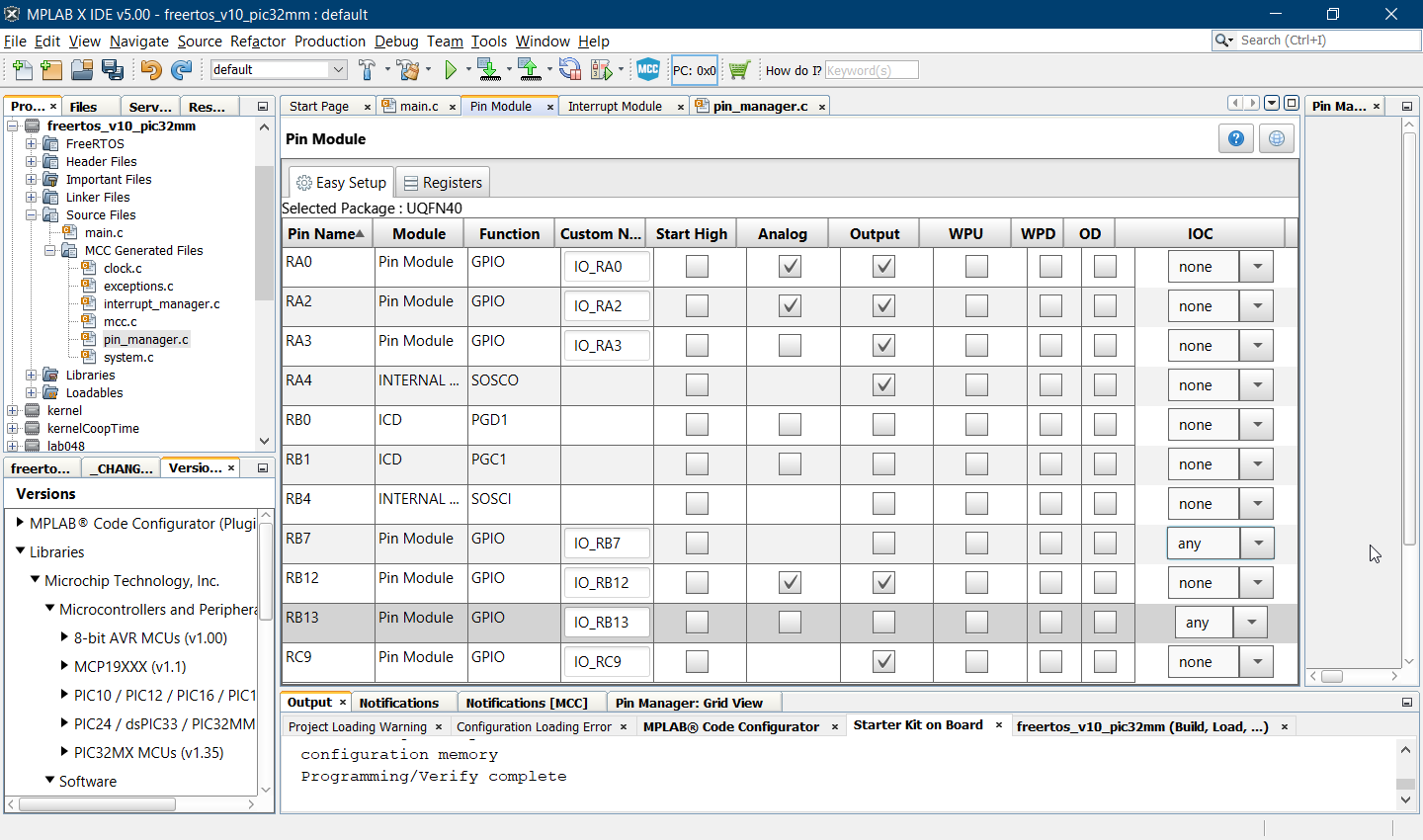
}

LAB2a

## Enviando dados da interrupção

Modifique no MCC o sistema de interrupção para os botões. Habilite a interrupção da porta B e o IOC para any.





Clique em generate. Em algumas situações o MCC pode não configurar corretamente os registros da interrupção, para isso modifique a função PIN\_MANAGER\_Initilizer() do arquivo pin\_manager.c, ligando os bits abaixo:

void PIN\_MANAGER\_Initialize(void) {

// outras definições

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Interrupt On Change for group CNCONB - config

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CNCONBbits.ON = 1;

CNCONBbits.CNSTYLE = 1;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Interrupt On Change for group CNEN0B - positive

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CNEN0Bbits.CNIE0B13 = 1; // Pin : RB13

CNEN0Bbits.CNIE0B7 = 1; // Pin : RB7

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Interrupt On Change for group CNEN1B - negative

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CNEN1Bbits.CNIE1B13 = 1; // Pin : RB13

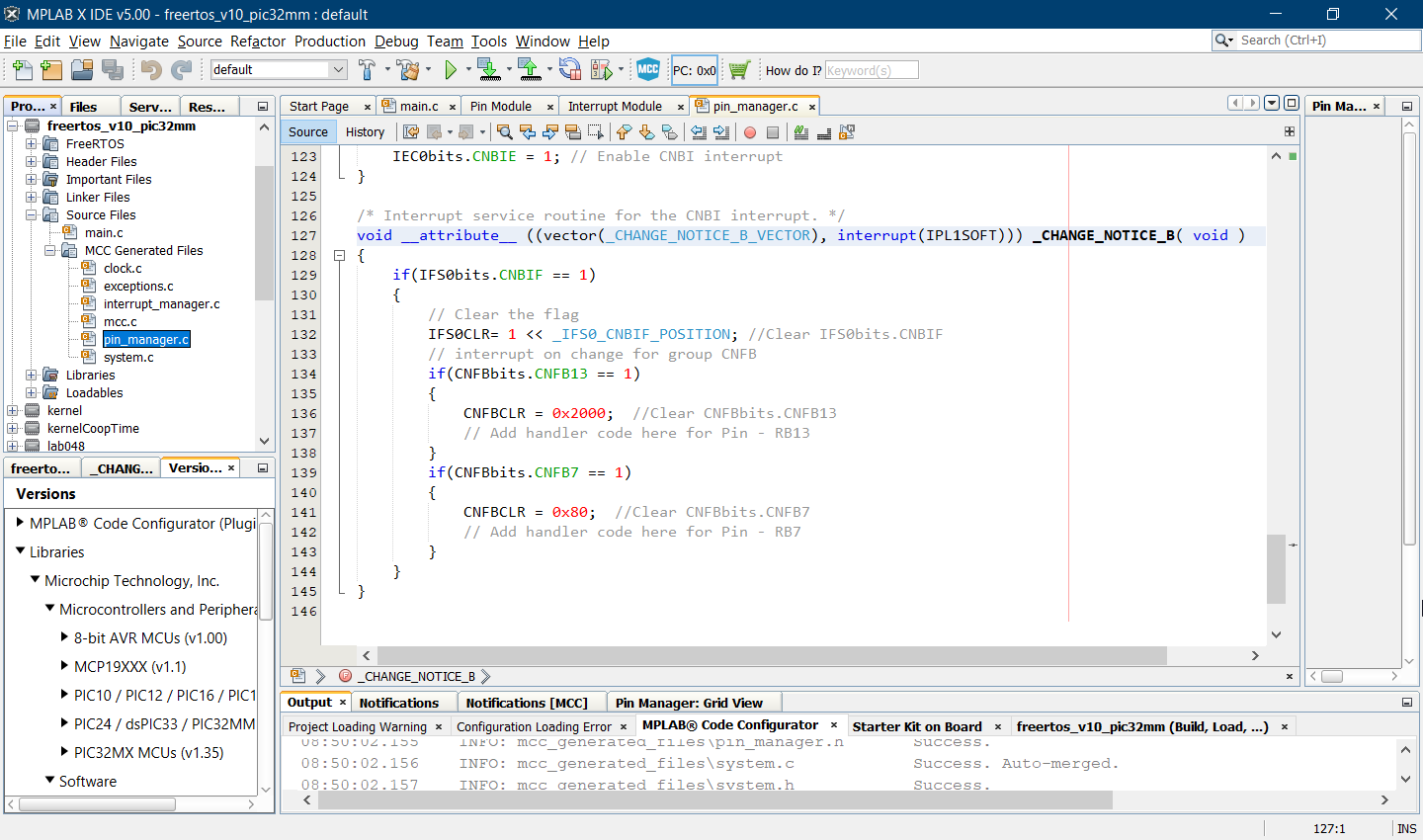
CNEN1Bbits.CNIE1B7 = 1; // Pin : RB7

IEC0bits.CNBIE = 1; // Enable CNBI interrupt

}

Remova a task buttons do programa principal e adicione o envio de mensagem pela queue através da interrupção.

Abra o arquivo pin\_manager.c dentro da pasta MCC Generated Files



Modifique o código da função de ISR para enviar os dados via fila de dentro da interrupção.

### Resposta

Atenção, uma parte do código está sendo duplicada, ou acessada via extern, para simplificar as alterações nos arquivos por questão de tempo. Num sistema real essa abordagem nunca é útil. É melhor criar um novo arquivo com as definições para evitar problemas futuros.

#include "FreeRTOS.h"

#include "queue.h"

typedef struct {

unsigned int num; // 1, 2

unsigned int status; // LED\_STATE\_OFF, LED\_STATE\_ON

} LedInfo;

enum {

LED\_STATE\_OFF, LED\_STATE\_ON

};

extern QueueHandle\_t ledQueue;

LedInfo led;

/\* Interrupt service routine for the CNBI interrupt. \*/

void \_\_attribute\_\_((vector(\_CHANGE\_NOTICE\_B\_VECTOR), interrupt(IPL1SOFT))) \_CHANGE\_NOTICE\_B(void) {

BaseType\_t xHigherPriorityTaskWoken;

/\* We have not woken a task at the start of the ISR. \*/

xHigherPriorityTaskWoken = pdFALSE;

if (IFS0bits.CNBIF == 1) {

// Clear the flag

IFS0CLR = 1 << \_IFS0\_CNBIF\_POSITION; //Clear IFS0bits.CNBIF

// interrupt on change for group CNFB

if (CNFBbits.CNFB13 == 1) {

CNFBCLR = 0x2000; //Clear CNFBbits.CNFB13

// Add handler code here for Pin - RB13

led.num = 2;

if (!IO\_RB13\_GetValue()) {

led.status = LED\_STATE\_ON;

} else {

led.status = LED\_STATE\_OFF;

}

xQueueSendFromISR(ledQueue, &led, &xHigherPriorityTaskWoken);

}

if (CNFBbits.CNFB7 == 1) {

CNFBCLR = 0x80; //Clear CNFBbits.CNFB7

// Add handler code here for Pin - RB7

led.num = 1;

if (!IO\_RB7\_GetValue()) {

led.status = LED\_STATE\_ON;

} else {

led.status = LED\_STATE\_OFF;

}

xQueueSendFromISR(ledQueue, &led, &xHigherPriorityTaskWoken);

}

if (xHigherPriorityTaskWoken) {

/\* Actual macro used here is port specific. \*/

// taskYIELD\_FROM\_ISR();

}

}

}

LAB2b

## Crie uma fila para enviar valores de sinal analógico (extra)

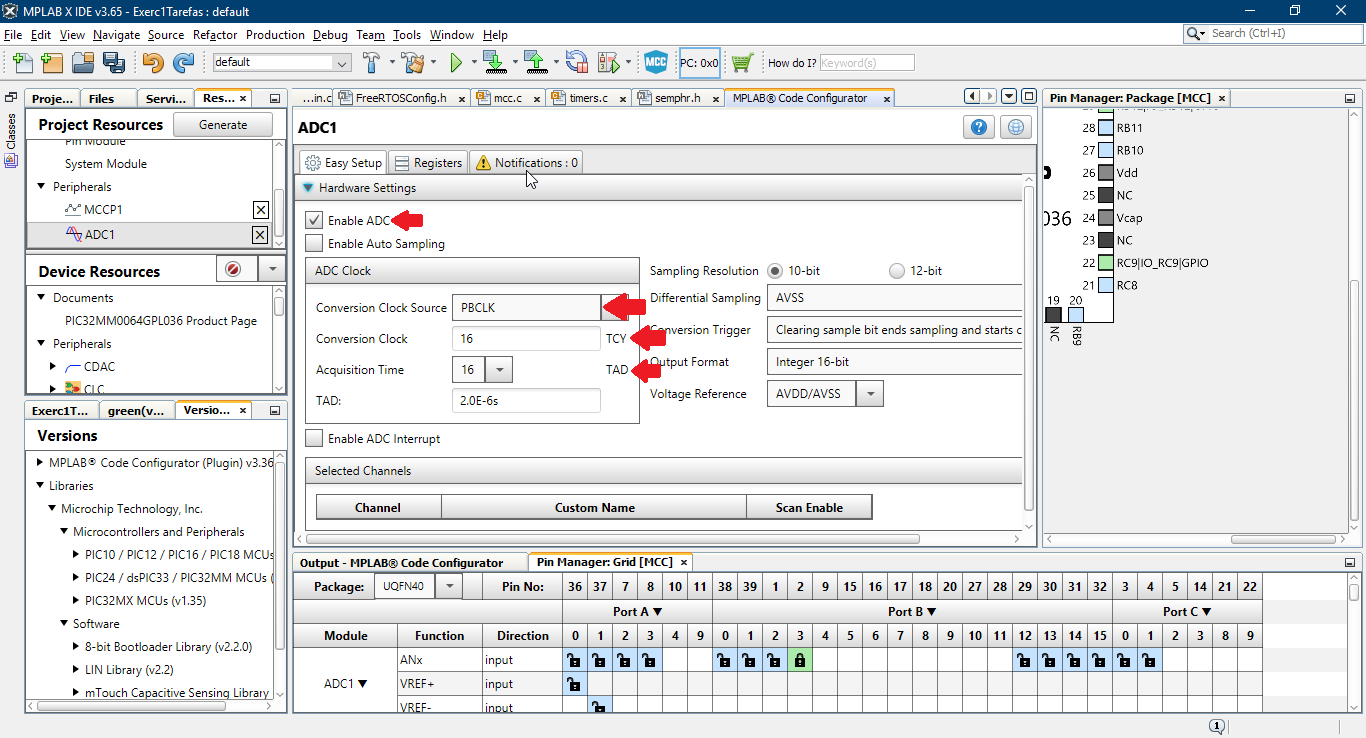
Ler o valor de um potenciômetro numa entrada analógica e enviar para uma task que aciona o PWM. A task do ADC apenas envia se o valor se alterar, se for igual ele deve dormir por 10ms.

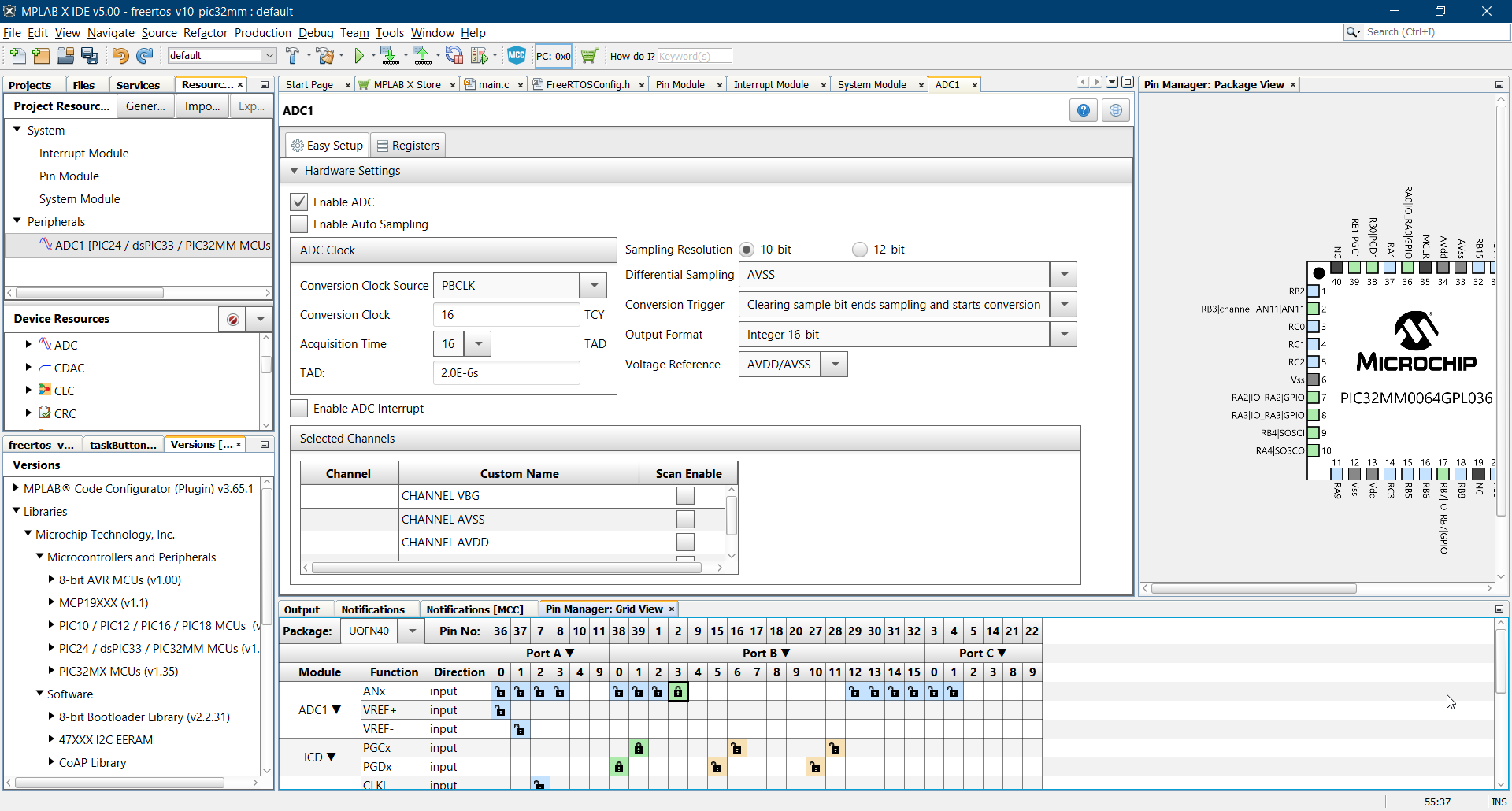
### Resposta

Agora vamos adicionar as configurações necessárias para a utilização do Conversor Analógico-Digital, ADC1.

Abra o MCC, e dê um duplo clique em ADC1(PIC24 / dsPIC32 /PIC32MM MCUS by Michochip Tech, Inc.) da janela Device Resources, à esquerda, e faças as seguintes mudanças (setas vermelhas):

* Clique em Enable ADC;
* Mude o Conversion Clock Source para PBCLK;
* Conversion Clock para 16 TCY;
* E o Acquisition Time para 16 TAD.
* No Pin Manager adicione o terminal 3 da portb (RB3) como input analógico.

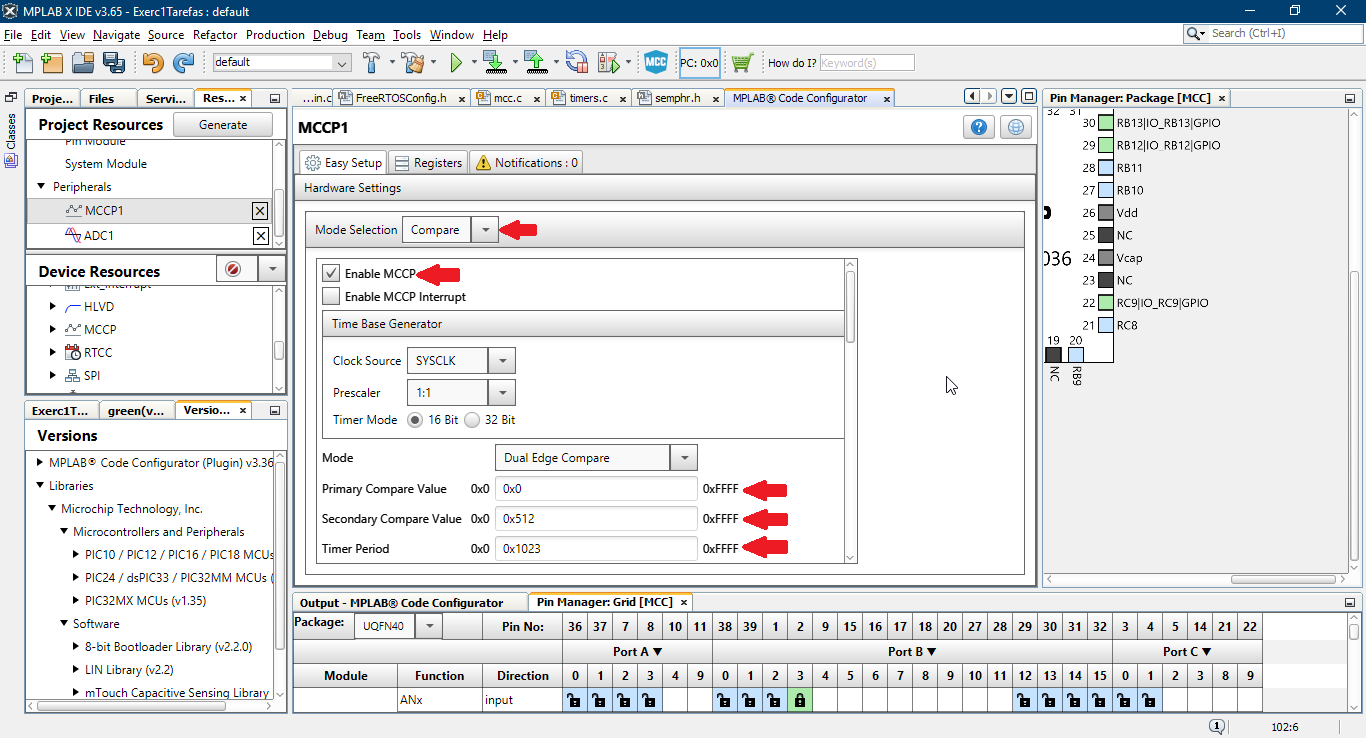




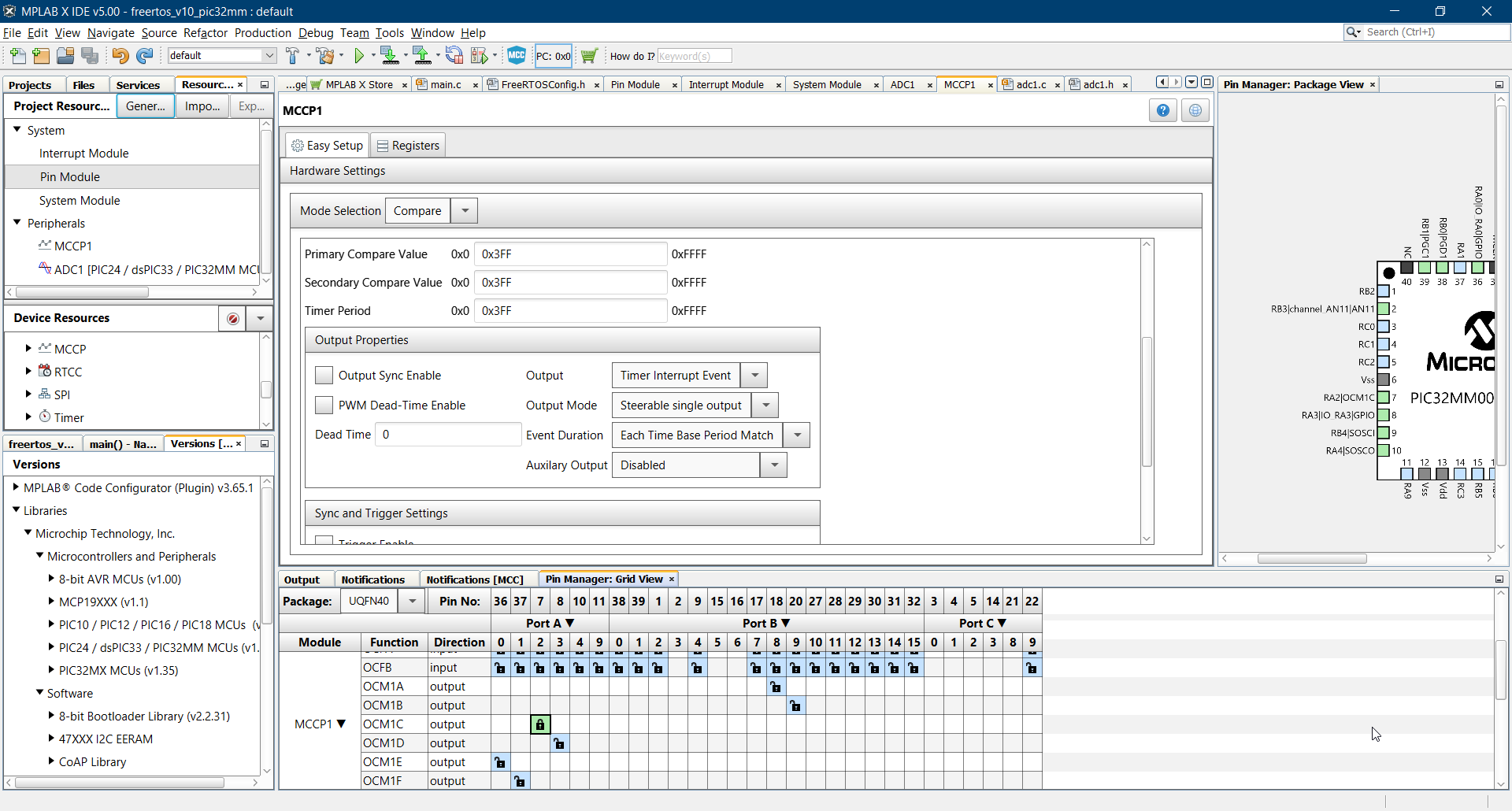
Com isso poderemos utilizar o potenciômetro da placa, localizado na porta AN11 em nosso projeto. Iremos agora configurar uma saída PWM que utilizaremos para controlar o LED RGB e demonstrar e controlar seu brilho.

Para isso voltamos ao MCC e na Janela Device Resources iremos adicionar o MCCP1 (MultiCompare/Capture/PWM/Timer) ao projeto e vamos fazer as seguintes configurações:

* Mode Selection para Compare;
* Clique em Enable MCCP;
* Em Mode selecione Dual Edge Compare;
* Mude os valores Primary Compare Value, Secondary Compare Value e Timer Period para 0x3FF em todos respectivamente.



Por último iremos selecionar em qual pino o nosso PWM irá atuar, no caso em questão no LED Azul localizado na porta RA2. O mapa final dos terminais é apresentado abaixo.



Após as configurações de hardware utilize os seguintes códigos das tasks.

|  |
| --- |
| QueueHandle\_t pwmQueue;  static void lerADC(void \*pvParameters) {  int i;  unsigned int conversion = 0;  ADC1\_ChannelSelect(ADC1\_CHANNEL\_AN11);  for (;;) {  ADC1\_Start();  for (i = 0; i < 1000; i++);  ADC1\_Stop();  while (!ADC1\_IsConversionComplete());  if (conversion != ADC1\_ConversionResultGet()) {  conversion = ADC1\_ConversionResultGet();  xQueueSend(pwmQueue, &conversion, portMAX\_DELAY);  } else {  vTaskDelay(10 / portTICK\_PERIOD\_MS);  }  }  }  static void taskPwm(void \*pvParameters) {  float t;  int readValor = 0;  MCCP1\_COMPARE\_Start();  for (;;) {  xQueueReceive(pwmQueue, &readValor, portMAX\_DELAY);  MCCP1\_COMPARE\_CenterAlignedPWMConfig(0, readValor);  }  } |

Dentro do main.c adicione o código para criar as tasks e a fila de mensagens

|  |
| --- |
| xTaskCreate**(**lerADC**,** "taskLerADC"**,** configMINIMAL\_STACK\_SIZE**,**  **NULL,** tskIDLE\_PRIORITY **+** 1**,** **NULL);**  xTaskCreate**(**taskPwm**,** "taskPWM"**,** configMINIMAL\_STACK\_SIZE**,**  **NULL,** tskIDLE\_PRIORITY **+** 1**,** **NULL);**  pwmQueue = xQueueCreate(3, sizeof(int)); |

LAB2Final

# Lab 3 - Semáforos

Iremos criar um projeto para demonstração dos problemas que levam à necessidade de utilização de semáforos.

## Utilizando recursos compartilhados

Para demonstrar o problema, iremos criar um projeto que ativa os diferentes LEDs da placa. Iremos utilizar o botão 1 para ativar o LED vermelho e o botão 2 para ativar o LED verde. Sem a utilização do semáforo a cor vai aparecer errada:

|  |
| --- |
| void red2**(**void **\***pvParameters**)** **{**  **for** **(;;)** **{**  **if** **(!**IO\_RB7\_GetValue**())** **{**  IO\_RA3\_SetHigh**();**  **}** **else** **{**  IO\_RA3\_SetLow**();**  **}**  **}**  **}**  void green2**(**void **\***pvParameters**)** **{**  **for** **(;;)** **{**  **if** **(!**IO\_RB13\_GetValue**())** **{**  IO\_RB12\_SetHigh**();**  **}** **else** **{**  IO\_RB12\_SetLow**();**  **}**  **}**  **}** |

Quanto ao main:

|  |
| --- |
| xTaskCreate**(**red2**,** "BotaoVermelho"**,** configMINIMAL\_STACK\_SIZE**,** **NULL,** tskIDLE\_PRIORITY **+** 1**,** **NULL);**  xTaskCreate**(**green2**,** "BotaoAzul"**,** configMINIMAL\_STACK\_SIZE**,** **NULL,** tskIDLE\_PRIORITY **+** 1**,** **NULL);** |

Pode-se observar que as cores se misturam, pelo fato de mais de um processo estar utilizando o mesmo recurso ao mesmo tempo.

Em vários programas isso pode ser um problema, um exemplo comum é quando 2 ou mais processos do sistema operacional tentam escrever sobre uma mesma posição de memória, o que pode causar erros nas variáveis.

## Solucionando o compartilhamento

Para se corrigir esse problema podemos utilizar os semáforos, que restringem o uso de determinado recurso apenas pelo processo que requisitar o semáforo. Para corrigir o problema dos botões podemos usar o semáforo para regular quem pode acessar o led em cada instante.

### Resposta

Os processos que utilizam o semáforo são:

|  |
| --- |
| #include "semphr.h"  SemaphoreHandle\_t buttonSem**;**  void red2**(**void **\***pvParameters**)** **{**  int antigo**,** novo **=** 0**;**  **for** **(;;)** **{**  novo **=** **!**IO\_RB7\_GetValue**();**  **if** **(**antigo **!=** novo**)** **{**  antigo **=** novo**;**  **if** **(**novo**)** **{**  xSemaphoreTake**(**buttonSem**,** portMAX\_DELAY**);**  IO\_RA3\_SetHigh**();**  **}** **else** **{**  IO\_RA3\_SetLow**();**  xSemaphoreGive**(**buttonSem**);**  **}**  **}**  **}**  **}**  void green2**(**void **\***pvParameters**)** **{**  int antigo**,** novo **=** 0**;**  **for** **(;;)** **{**  novo **=** **!**IO\_RB13\_GetValue**();**  **if** **(**antigo **!=** novo**)** **{**  antigo **=** novo**;**  **if** **(**novo**)** **{**  xSemaphoreTake**(**buttonSem**,** portMAX\_DELAY**);**  IO\_RB12\_SetHigh**();**  **}** **else** **{**  IO\_RB12\_SetLow**();**  xSemaphoreGive**(**buttonSem**);**  **}**  **}**  **}**  **}** |

É necessário inicializar o semáforo e disponibilizá-lo para que as tarefas possam requisitá-lo.

|  |
| --- |
| buttonSem **=** xSemaphoreCreateBinary**();**  xSemaphoreGive**(**buttonSem**);** |

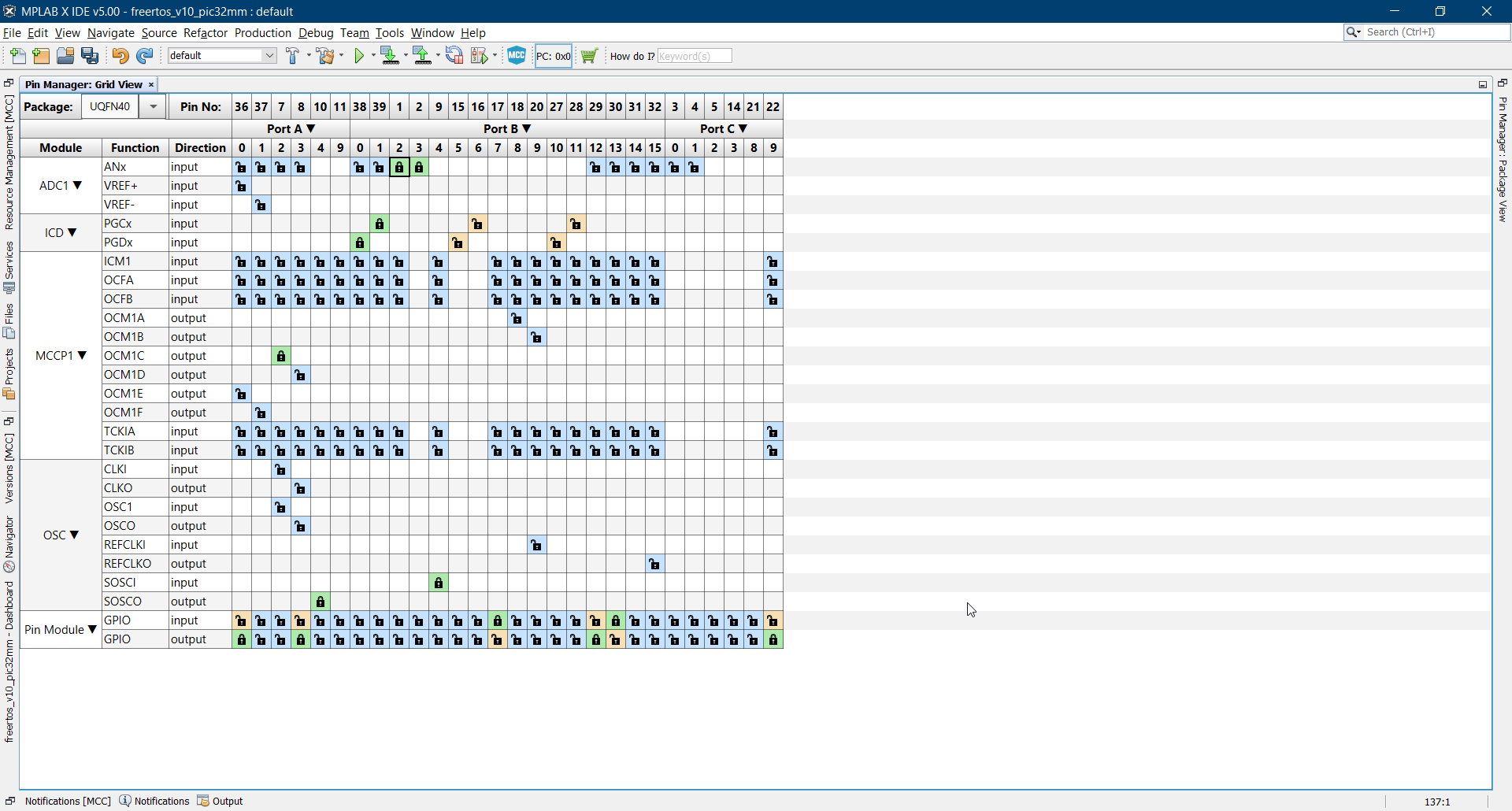
# Desenvolvendo uma aplicação com requisitos de tempo real (PID)

O resultado do laboratório 4 não pode ser visualizado/posto em prática pois as placas de desenvolvimento não estão conectadas à uma planta de controle. Os passos aqui foram desenvolvidos utilizando uma planta com uma malha RC que serve como instrumento de visualização da resposta do sistema. Além disso todos os passos são idênticos ao desenvolvimento de uma aplicação funcional.

## Configurando os periféricos

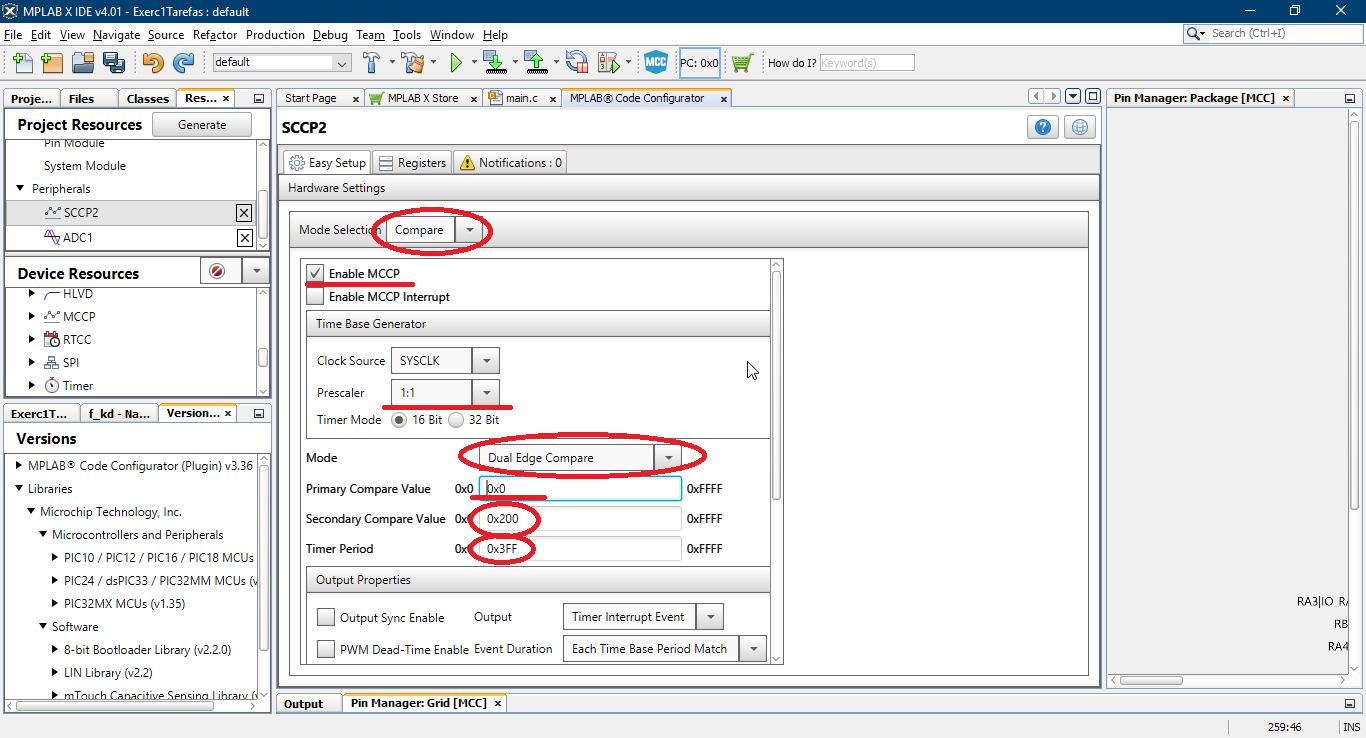
Crie o projeto conforme os exercícios anteriores.

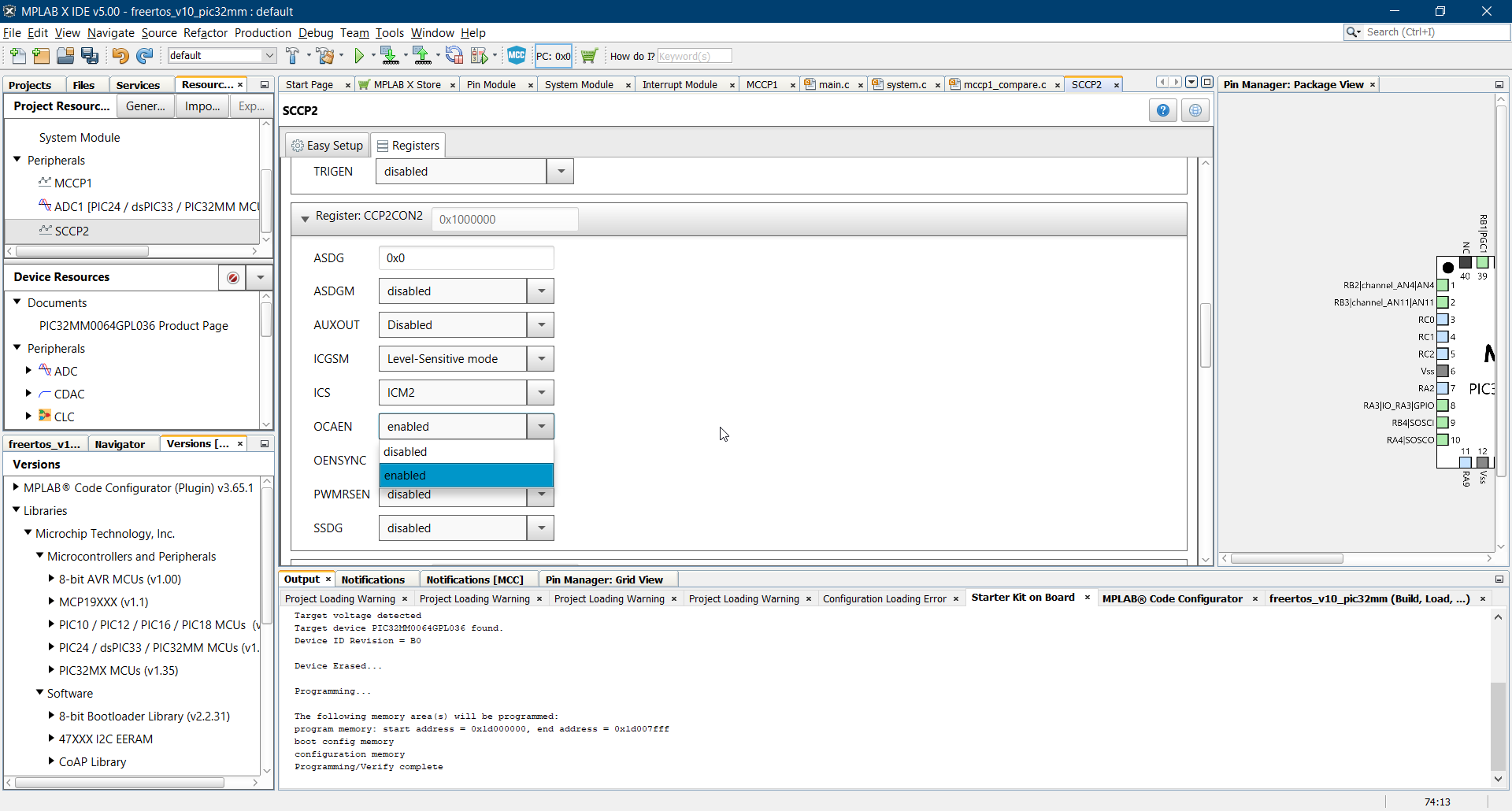
Configure o drive de conversor AD conforme demonstrado anteriormente para capturar o sinal do potenciômetro e do terminal **RB2**. Para isso associamos os pinos no pin manager, conforme a imagem abaixo:



Configure o driver do controlador PWM. Utilizaremos os drivers do SCCP2 e o MCCP1, para isso abra o MCC. Em Device Resources, expanda as opções para MCCP, e dê um duplo clique sobre SCCP2 que será adicionado ao projeto e poderá ser configurado acima, em Project Resources. Selecionando SCCP2 vamos altera-la da seguinte maneira:

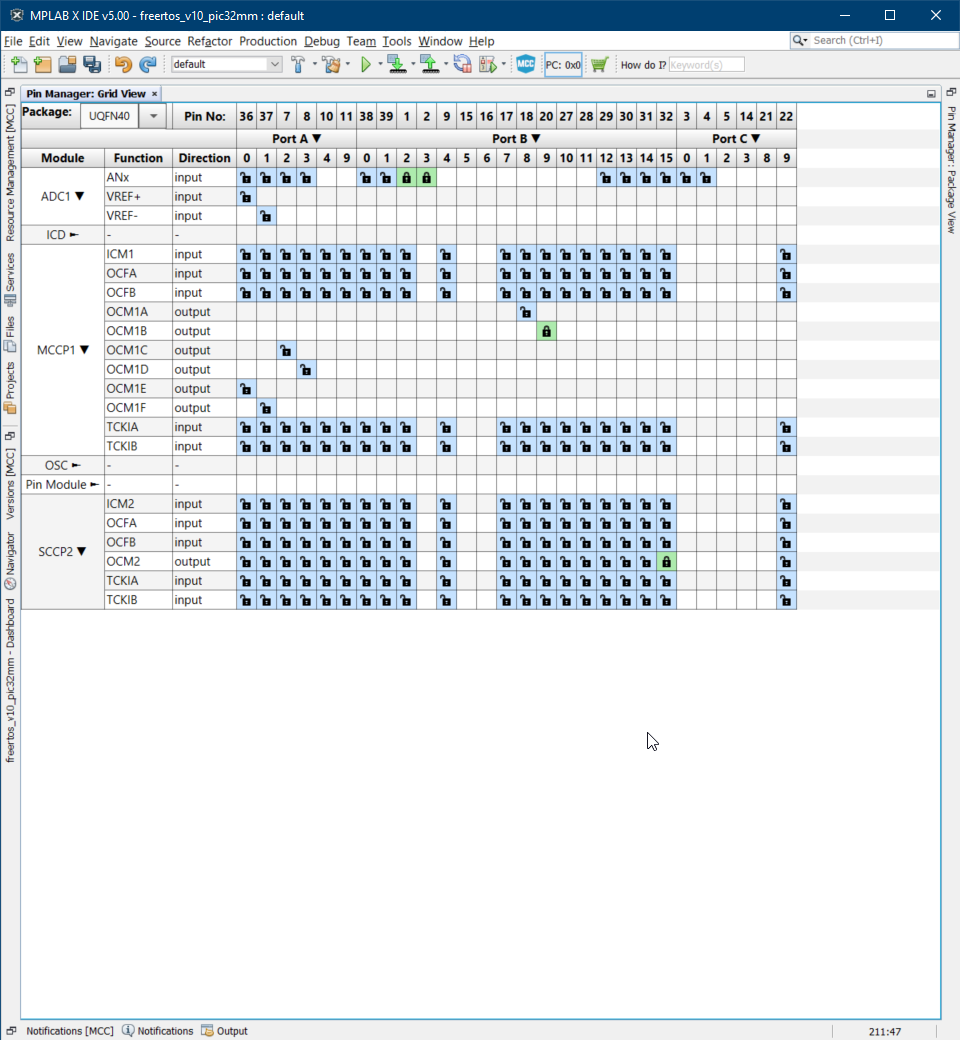
* Mode Selection para Compare;
* Tenha certeza de a caixa Enable MCCP está selecionada e que o Prescaler está em 1:1 e o Primary Compare Value em 0x3FF;
* Mude o Mode para Dual Edge Compare, o Secondary e o Time Period para 0x3FF e 0x3FF respectivamente.





Mude para a aba Registers e ligue o bit OCAEN do registro CCP2CON2 para habilitar a saída PWM

No Pin Manager, ajuste o MCCP (desligando o RA2 e ligando o RB9) e o SCCP (ligando o RB15). Confirme também que RB2 e RB3 estejam como analógicos. Segue o mapa final dos pinos



Clique em Generate e recompile o projeto.

## Criando tarefas para rastreamento de referência e execução de controle em malha aberta

Nesta atividade criaremos duas tarefas. A primeira tarefa irá rastrear o valor do potenciômetro e ajustar uma variável global SP. A segunda tarefa será responsável pelo sistema de controle. Ela deverá simplesmente colocar o valor de SP na saída PWM. O objetivo é que a variável da saída atinja o valor da referência. Utilize o código abaixo.

int32\_t sp;

void sp\_change(void \*pvParameters) {

int t;

MCCP1\_COMPARE\_Start();

for (;;) {

ADC1\_ChannelSelect(ADC1\_CHANNEL\_AN11);

ADC1\_Start();

for (t = 0; t < 1000; t++);

ADC1\_Stop();

while (!ADC1\_IsConversionComplete());

sp = ADC1\_ConversionResultGet();

MCCP1\_COMPARE\_DualCompareValueSet(0, sp);

vTaskDelay(10/portTICK\_PERIOD\_MS);

}

}

void task\_PID(void \*pvParameters) {

TickType\_t xLastWakeTime;

const TickType\_t xFrequency = 5 / portTICK\_PERIOD\_MS;

xLastWakeTime = xTaskGetTickCount();

SCCP2\_COMPARE\_Start();

for (;;) {

SCCP2\_COMPARE\_DualCompareValueSet(0, sp);

vTaskDelayUntil(&xLastWakeTime, xFrequency);

}

}

Para adicionar as tasks no FreeRTOS:

|  |
| --- |
| xTaskCreate**(**task\_PID, "PID"**,** configMINIMAL\_STACK\_SIZE**,**  **NULL,** tskIDLE\_PRIORITY **+** 1**,** **NULL);**  xTaskCreate**(**sp\_change**,** "sp"**,** configMINIMAL\_STACK\_SIZE**,**  **NULL,** tskIDLE\_PRIORITY **+** 1**,** **NULL);** |

Ajuste o potenciômetro para que três leds estejam acesos. Aperte um dos botões da placa de controle e perceba quanto tempo o sistema (terceira coluna) demora para estabilizar. Perceba também o comportamento do sinal de atuação (primeira coluna).

## Adicionando um controle PID

Edite o arquivo main.c.

Adicione o seguinte código antes da função main(). Este código realiza a função de uma malha de controle PID discretizada com a aproximação backward e utilizando o cálculo com ponto fixo.

|  |
| --- |
| #define f\_kp 10.0  #define f\_ki 50.0  #define f\_kd 0.0  #define f\_T 0.005  #define SHIFT (256)  #define MAX\_SAT 1023  #define MIN\_SAT 0  const int32\_t i\_k1 = (f\_kp + f\_ki \* f\_T + f\_kd / f\_T) \* SHIFT;  const int32\_t i\_k2 = -((f\_kp + 2 \* f\_kd / f\_T) \* SHIFT);  const int32\_t i\_k3 = (f\_kd / f\_T) \* SHIFT;  int16\_t i\_e0 = 0, i\_e1 = 0, i\_e2 = 0;  int16\_t i\_y0 = 0, i\_y1 = 0;  int32\_t sp;  int16\_t pid\_sw\_fixed(int ad) {  // Update variables  i\_y1 = i\_y0;  i\_e2 = i\_e1;  i\_e1 = i\_e0;  i\_e0 = (sp - ad);  // Processing  //the multiplication is for a number range in 4.2(6 bits) (gains from zero up to +15.75)  // times 0.3ff (10 bits)  // y0 = y1 + (i\_kp \* (e0 - e2)) +  // (i\_ki \* (e0) \* i\_T) +  // (i\_kd \* (e0 - (2 \* e1) + e2) / i\_T);  //stated in terms of errors instead of coefficients  i\_y0 = ((i\_k1 \* i\_e0) + (i\_k2 \* i\_e1) + (i\_k3 \* i\_e2));  i\_y0 = i\_y0 >> 8;  i\_y0 += i\_y1;  // Saturation  if (i\_y0 > MAX\_SAT) {  i\_y0 = MAX\_SAT;  } else if (i\_y0 < MIN\_SAT) {  i\_y0 = MIN\_SAT;  }  return i\_y0;  } |

Ambas as tasks deverão ser modificadas. A task de PID deverá realizar a leitura no canal 11 do ADC, enviar esse valor para a função de controle PID e utilizar o resultado para enviar ao PWM do SCCP2. Como o ADC será utilizando em duas tasks, sua utilização deve ser protegida por um semáforo.

|  |
| --- |
| SemaphoreHandle\_t adc\_sem;  void sp\_change(void \*pvParameters) {  int t;  MCCP1\_COMPARE\_Start();  for (;;) {  //necessário semáforo para não interromper a leitura do AD no meio  xSemaphoreTake(adc\_sem, portMAX\_DELAY);  ADC1\_ChannelSelect(ADC1\_CHANNEL\_AN11);  ADC1\_Start();  for (t = 0; t < 1000; t++);  ADC1\_Stop();  while (!ADC1\_IsConversionComplete());  sp = ADC1\_ConversionResultGet();  xSemaphoreGive(adc\_sem);  //fim leitura do AD  MCCP1\_COMPARE\_DualCompareValueSet(0, sp);  vTaskDelay(10 / portTICK\_PERIOD\_MS);  }  }  void task\_PID(void \*pvParameters) {  uint16\_t ad;  uint16\_t res;  int t;  TickType\_t xLastWakeTime;  const TickType\_t xFrequency = 5 / portTICK\_PERIOD\_MS;  xLastWakeTime = xTaskGetTickCount();  SCCP2\_COMPARE\_Start();  for (;;) {  //necessário semáforo para não interromper a leitura do AD no meio  xSemaphoreTake(adc\_sem, portMAX\_DELAY);  ADC1\_ChannelSelect(ADC1\_CHANNEL\_AN4);  ADC1\_Start();  for (t = 0; t < 1000; t++);  ADC1\_Stop();  while (!ADC1\_IsConversionComplete());  ad = ADC1\_ConversionResultGet();  xSemaphoreGive(adc\_sem);  //fim leitura do AD  //cálculo PID e atualização da saída  res = pid\_sw\_fixed(ad);  SCCP2\_COMPARE\_DualCompareValueSet(0, res);  //garantia de tempo real  vTaskDelayUntil(&xLastWakeTime, xFrequency);  }  } |

Por fim lembre-se de startar corretamente o semáforo na função main.

adc\_sem = xSemaphoreCreateBinary();

xSemaphoreGive(adc\_sem);

Por fim os códigos O exemplo apresentado aqui utiliza um PID digital, baseado na transformada Z backward. Os valores de KP, KI e KD devem ser analisados para cada tipo de planta a ser controlada. A leitura de sinal é realizada no canal 1 do ADC e a saída para a planta é feita através do PWM0 do microcontrolador. Supõe-se que exista um filtro passa baixas na saída do PWM para transformá-lo num DAC.

Os gráficos a seguir demonstram o resultado do controle PID, onde:

* A onda amarela representa a onda de saída do microcontrolador;
* A onda azul é a resposta do sistema;
* A onda verde é a referência a ser seguida.

