Desenvolvendo aplicações com Bluetooth® Low Energy RN4020

Autor: Fábio Souza - Embarcados: www.embarcados.com.br

Esse texto ilustra os laboratórios que serão realizados durante o treinamento: Desenvolvendo aplicações com Bluetooth® Low Energy RN4020, no Masters Brasil 2016.

Objetivo

- Desenvolver aplicações com a placa Curiosity e a BLE2 Click
- Entender como fazer a interface do RN4020 com um microcontrolador
- Desenvolver aplicações para interface com smartphone

Recursos necessários

Para o desenvolvimento desse laboratório serão necessários os seguintes recursos:

Hardware:

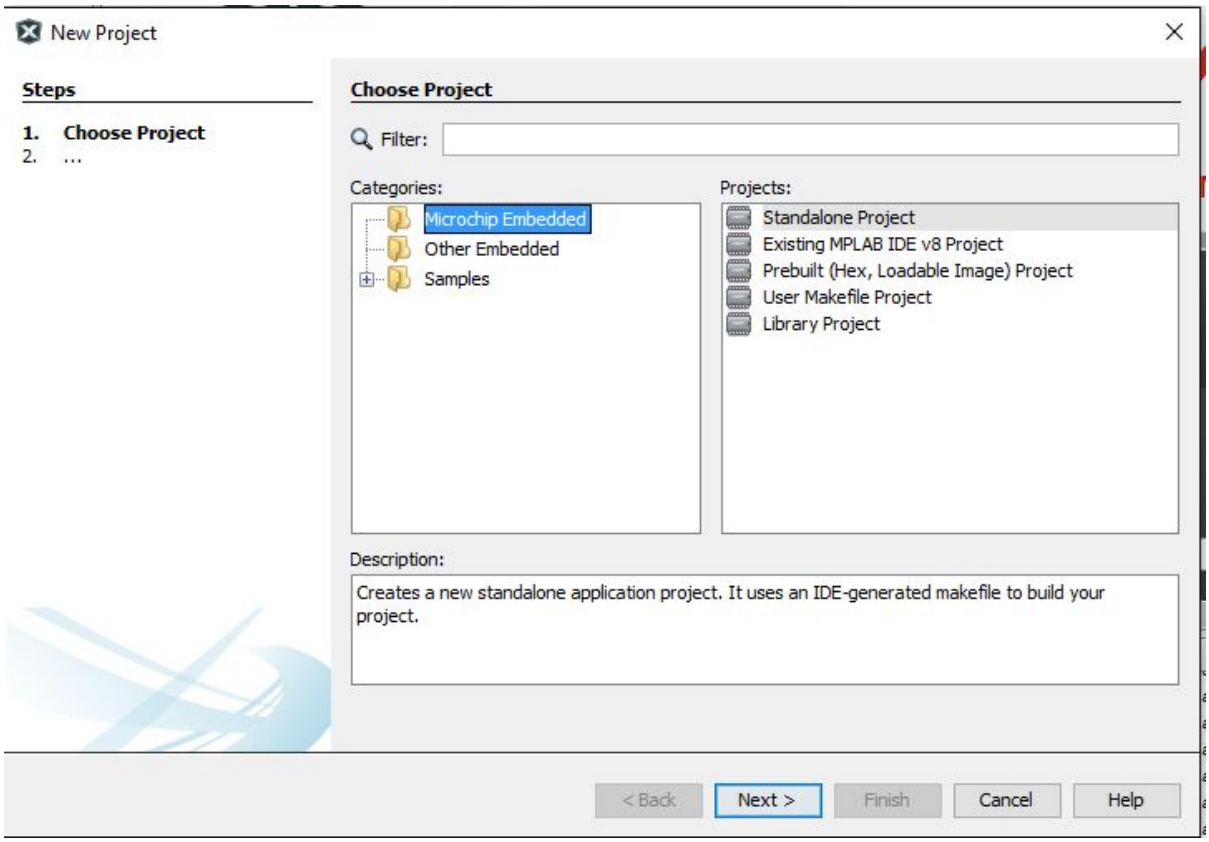
- DM164137 - Curiosity
- TMIK037 - BLE2 click

Software:

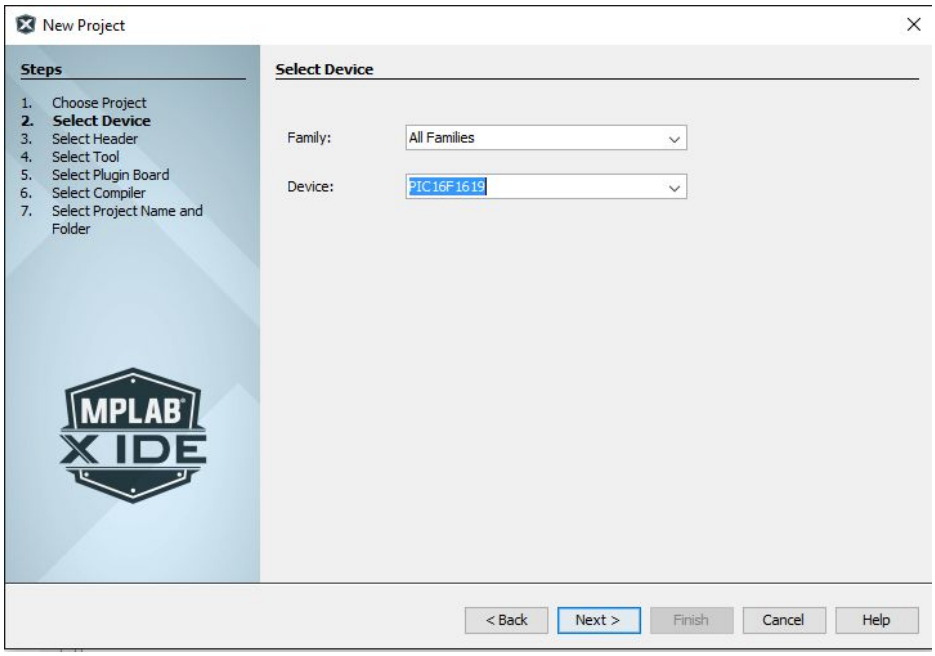
- MPLAB X IDE V3.40 ou superior
- MPLAB XC8 V1.38
- MPLAB® Code Configurator (MCC) V 2.25.2
- Aplicativo para Smartphone: Bluetooth Smart Discover:
    - IOS
    - Android

Lab 1 - Criando estrutura do projeto com MPLAB X, MCC e Compilador XC8 para interface com o módulo RN4020 usando a placa Curiosity

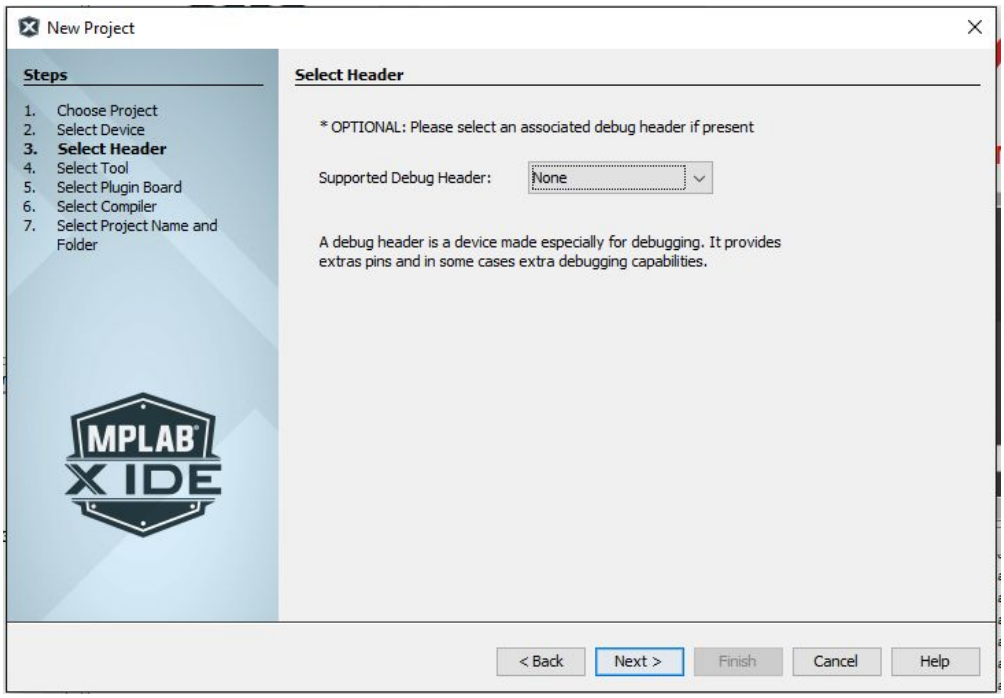
Abra o MPLAB X IDE, vamos criar um novo projeto. Para isso acesse File-> New Project. Será aberta a seguinte janela:



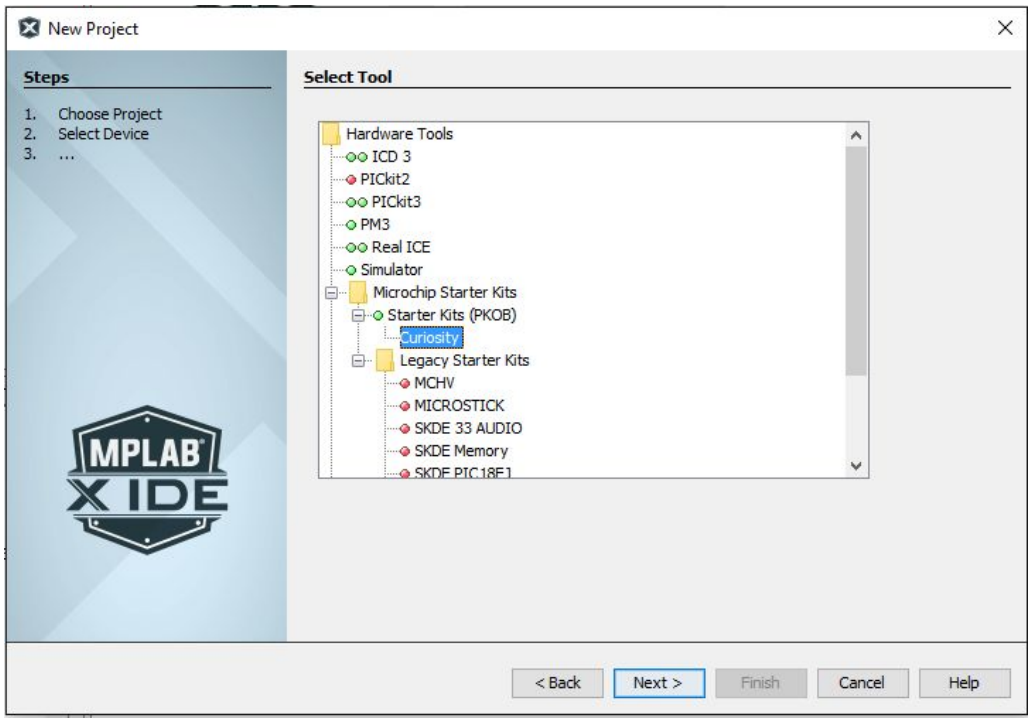
Escolher a categoria **Microchip Embedded** e **Standalone Project**. Clicando em Next, será exibida a janela a seguir. Selecione na lista o PIC16F1619, que é o microcontrolador presente na placa Curiosity:



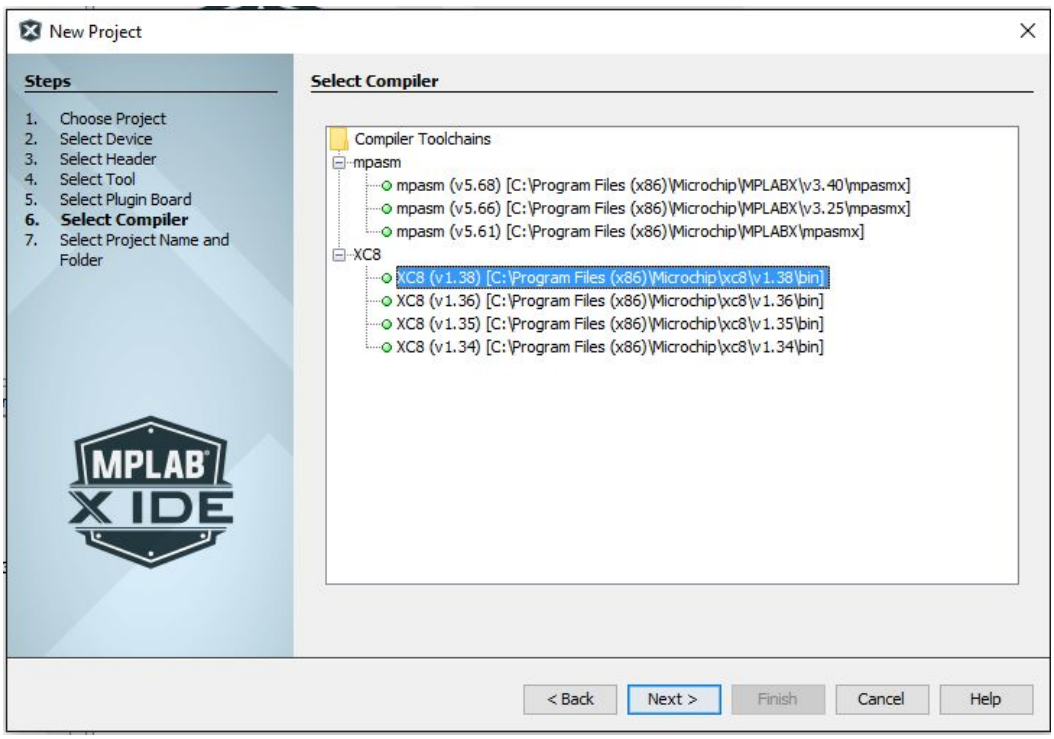
Após a seleção do microcontrolador, clicar em NEXT. Na tela de seleção de Header não é necessário nenhuma alteração, apenas clicar em NEXT



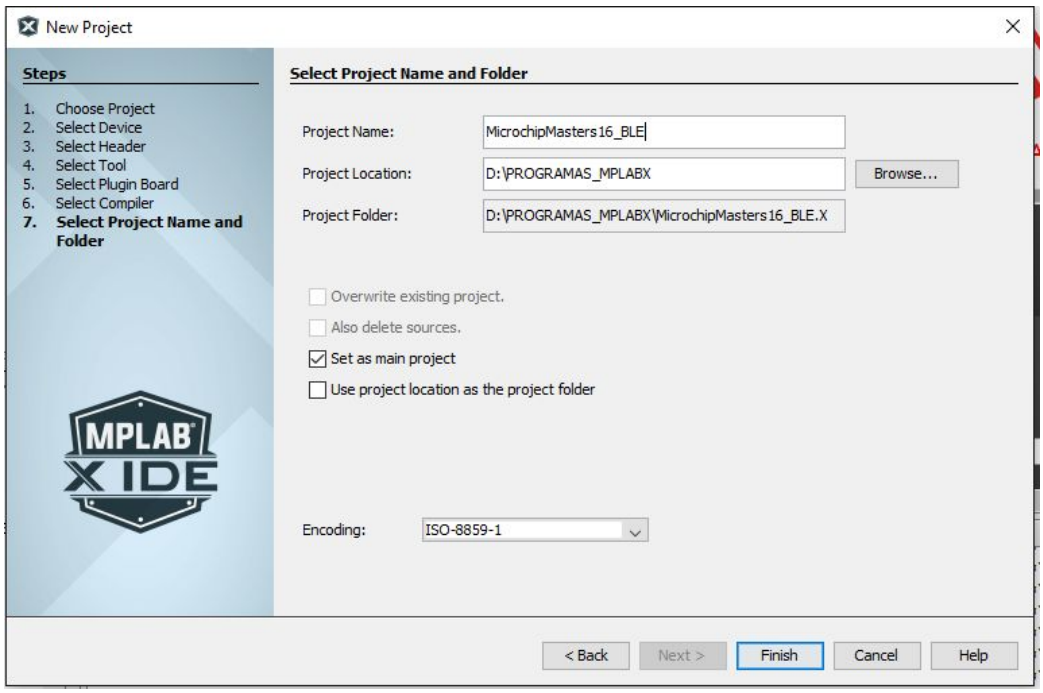
Agora vamos selecionar as ferramentas. Certifique que a placa Curiosity está plugada no computador, e a selecione na lista:



Após a seleção da ferramenta e clicando em NEXT, será aberta a janela para escolha do compilador. Para esse laboratório iremos utilizar o XC8 V1.38:



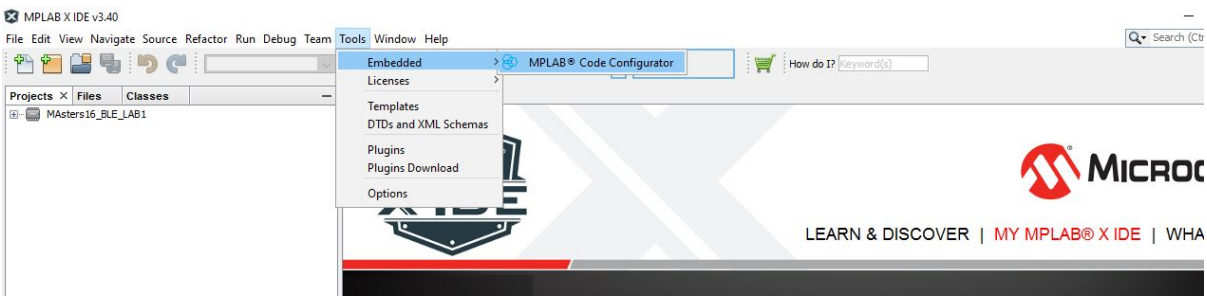
Por último, vamos nomear o projeto, escolher uma pasta para salvá-lo e finalizar o assistente.



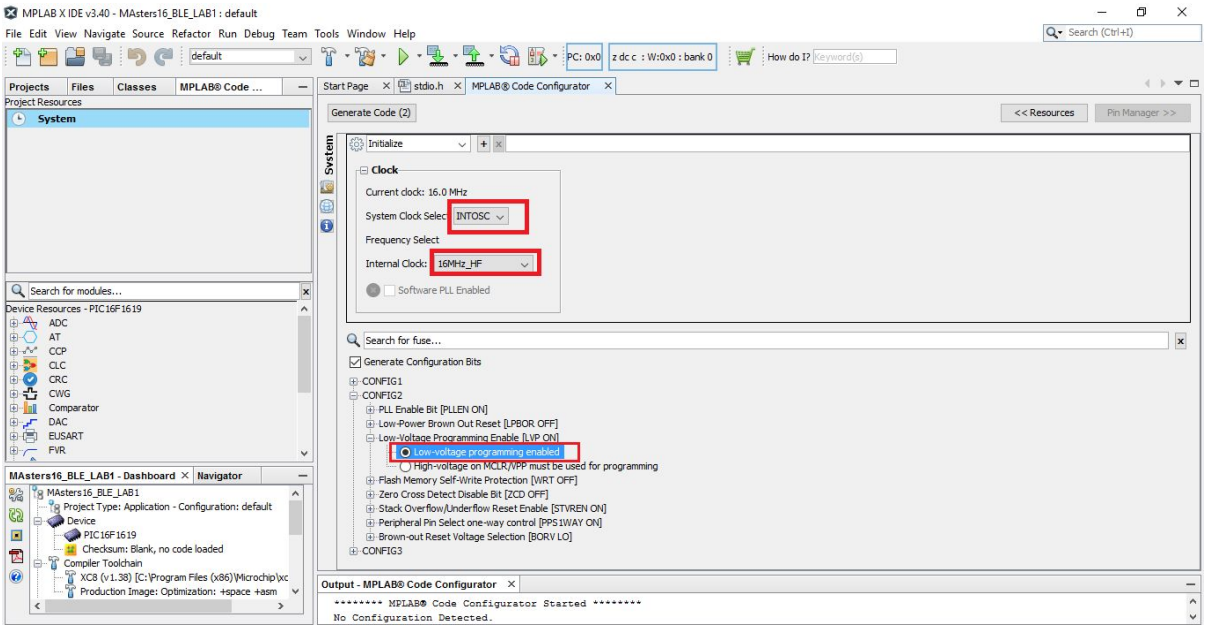
Pronto, criamos a estrutura do projeto para nosso laboratório.

Utilizando o MCC para configurações do Microcontrolador

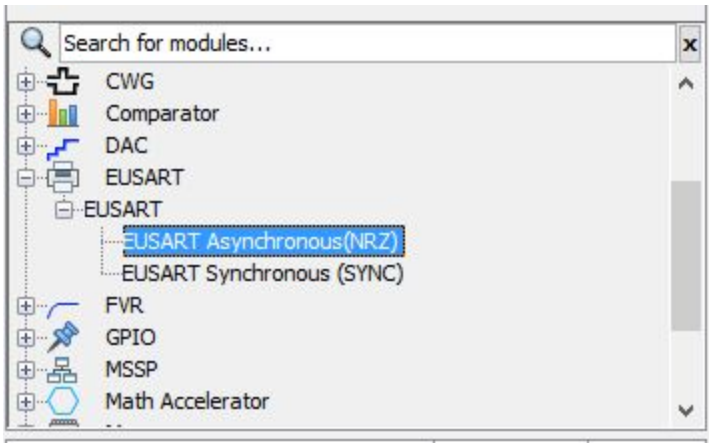
Para auxiliar na configuração do microcontrolador e seus periféricos, vamos utilizar o MPLAB Code Configurator. Para abri-lo acesse Tool->Embedded->MPLAB Code Configurator:



Após sua abertura, o primeiro passo é configurar o sistema. Para isso acesse o System e faça as seguintes configurações:



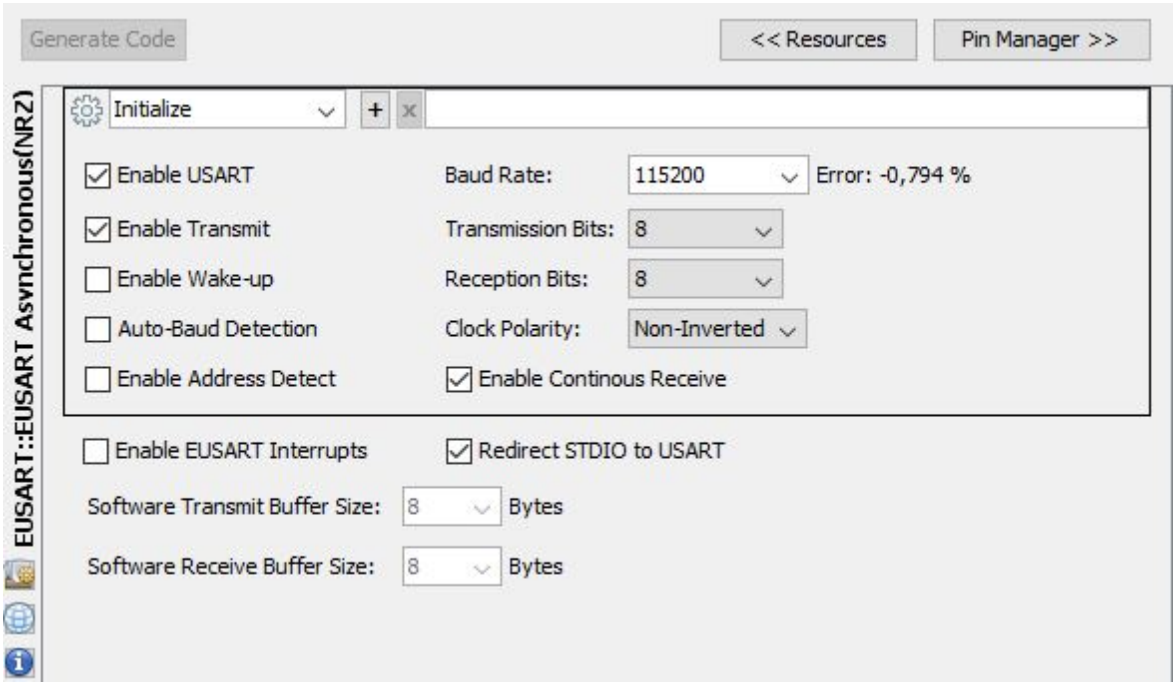
Como a comunicação com o RN4020 é feita através da UART, precisamos configurar o periférico de comunicação EUSART. Para isso é só dar duplo clique no EUSART Assynchronous(NRZ), presente na lista de Device Resources:



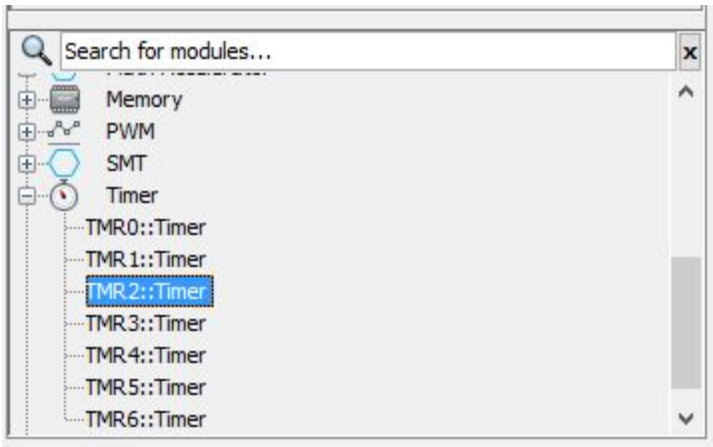
O RN4020 possui as seguintes características para comunicação serial:

Parameter	Value
Baud Rate	115200
Data Bits	8
Parity	None
Stop Bits	1
Flow Control	None

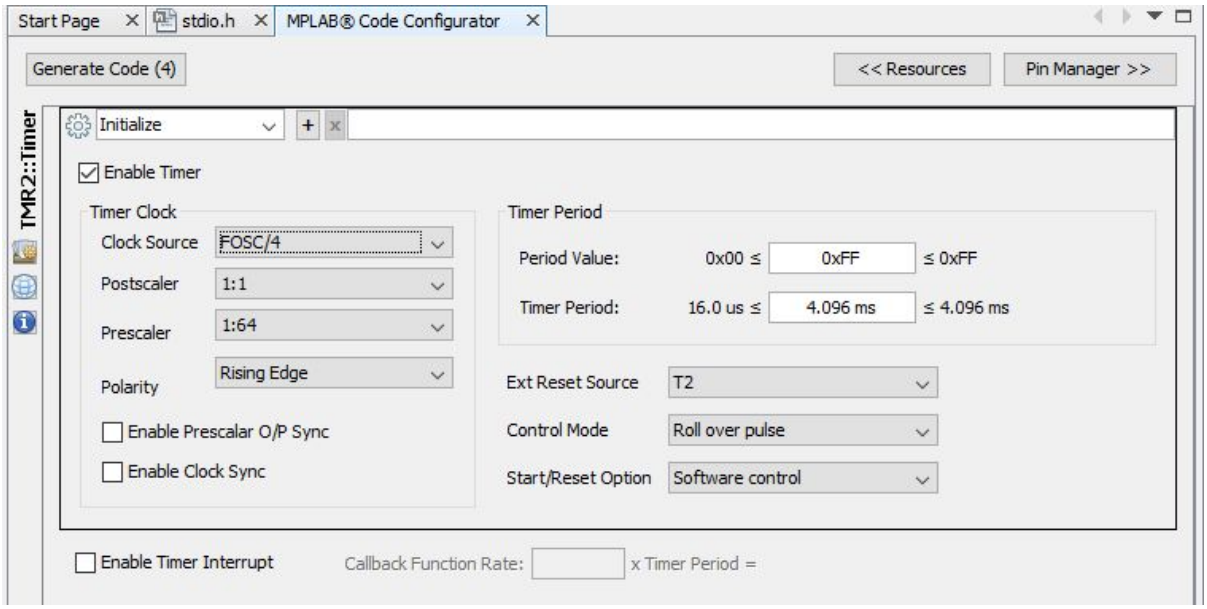
Assim, vamos configurar a EUSART da seguinte forma:



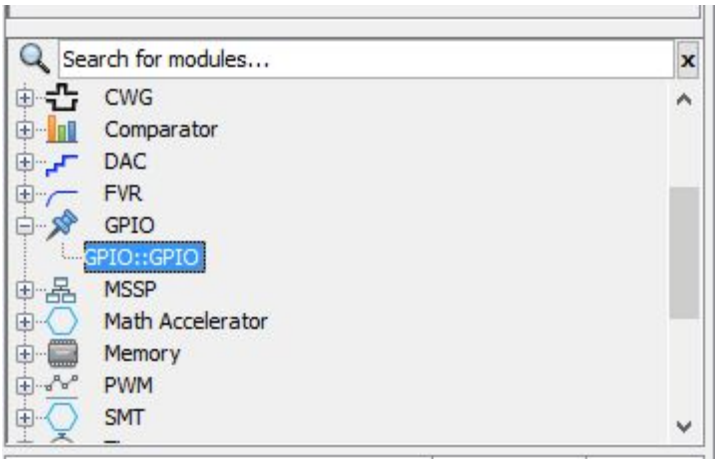
Aproveitando, vamos configurar o TMR2 que servirá de base de tempo para envio das mensagens em nossa aplicação. Em Timer, selecione o TMR2:



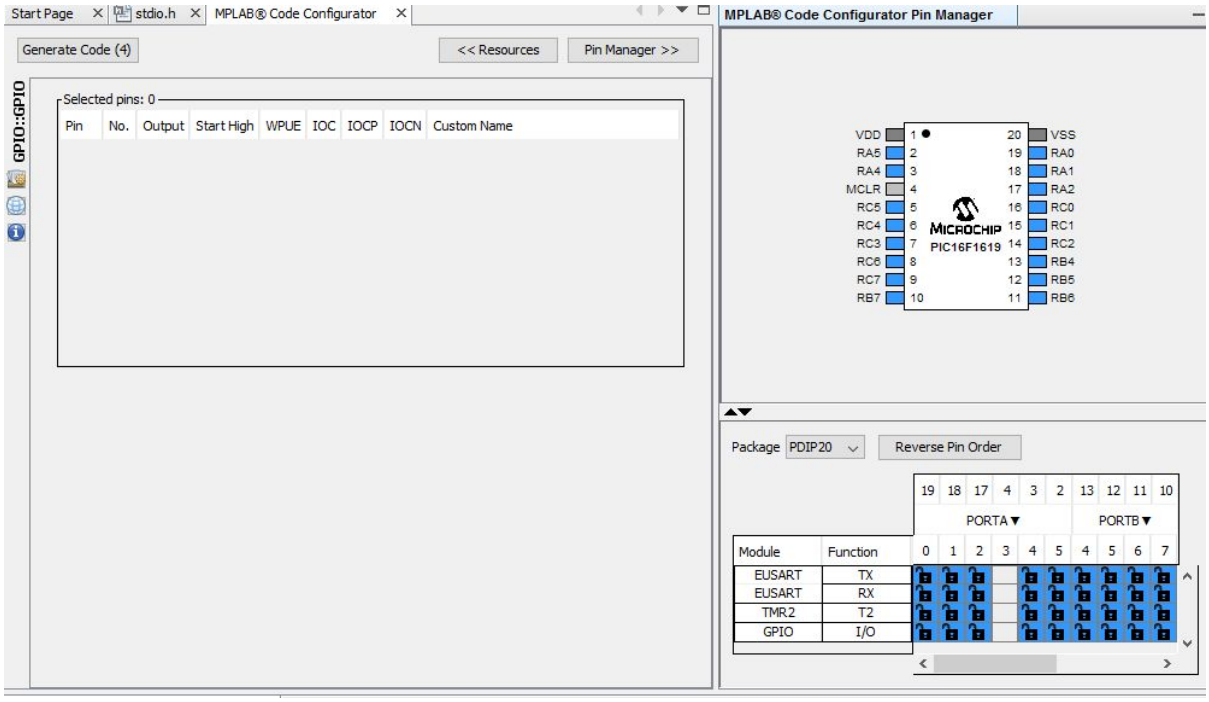
Após a seleção do TMR2, fazer a seguinte configuração para o módulo do TMR2:



Por último, vamos configurar o GPIO:



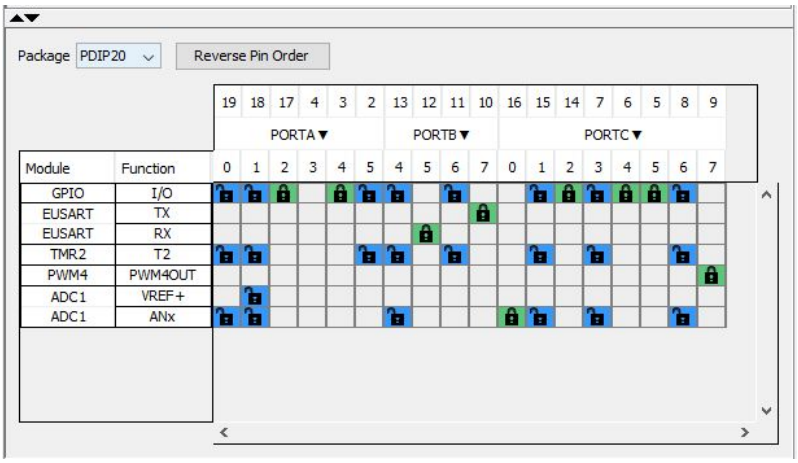
Será aberta a seguinte janela:

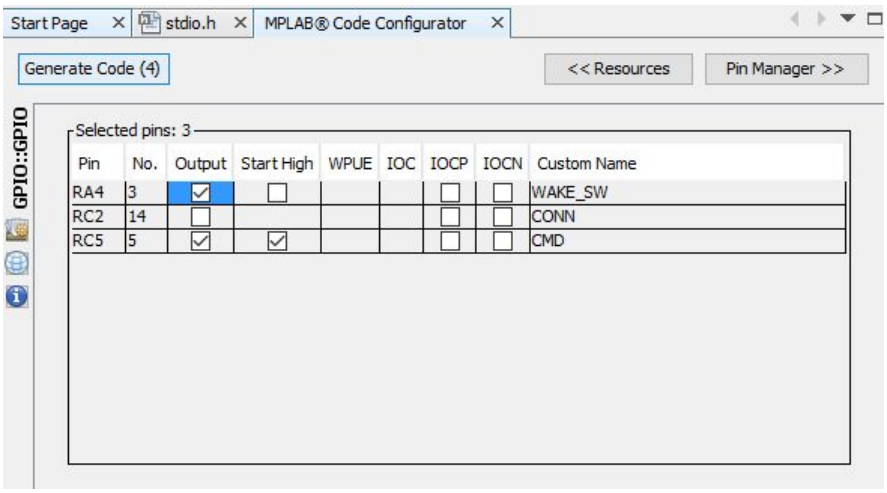


Sabemos que os pinos da BLE2 Click Board estão ligados da seguinte forma no PIC16F1619:

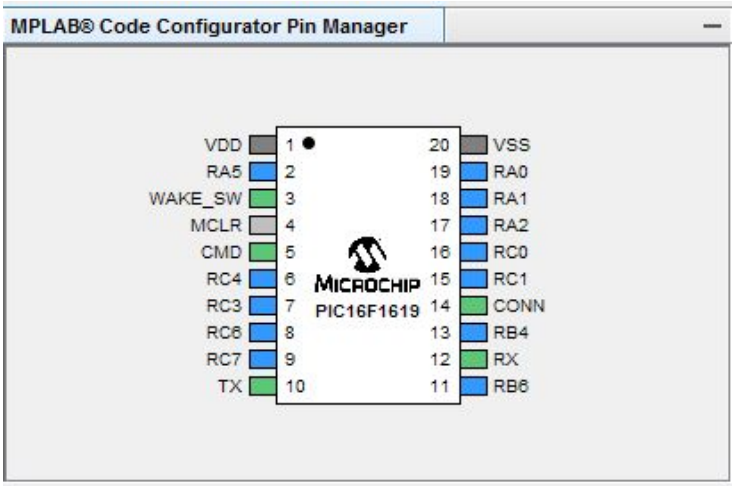
PIC16F16198	**BLE2 Click**
RA4	SWAKE
RB5	TX
RB7	RX
RC2	CONN
RC5	CMD/MLDP

Configurar os pinos da seguinte forma:

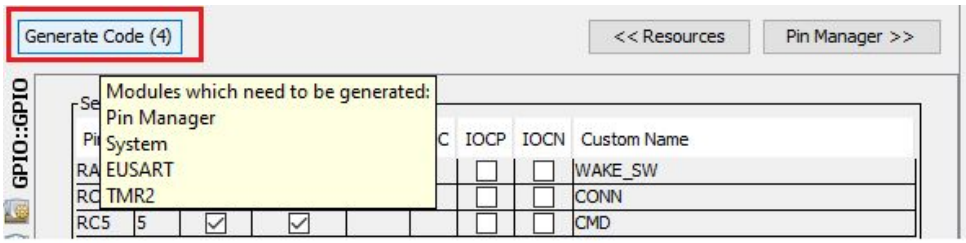




Note que os pinos são atualizados no Package:



As configurações dos periféricos estão prontas. Agora é só clicar no botão Generate. O MCC irá gerar os arquivos automaticamente:



O MCC perguntará se deseja criar o arquivo main.c. Clicar em Yes.

Note que foram gerados diversos arquivos e inclusive o arquivo main.c, onde se encontra a função main.

Abra o arquivo main.c e insira as funções para envio e recebimento de dados pela serial:

```
#include "mcc_generated_files/mcc.h"

#define cmdLONGDELAY    640  // 640ms

char rxBuffer[32];

void mygets( void);
void sendCMD( const char * cmd);

/*
                         Main application
 */
void main(void)
{
    // initialize the device
    SYSTEM_Initialize();



    while (1)
    {
        uint8_t time;              // auxiliary time variable

        if ( TMR2_HasOverflowOccured()) {
            time++;
        }

        if ( time >= 50)    // 200ms
        {

        }

    }
}

void mygets( void)
{
    char c, i=0;
    char *p = rxBuffer;
    while ( (i++ < sizeof(rxBuffer)) && (( c = getch()) != '\n')) {
        *p++ = c;
    }
    *p = '\0';
}

void sendCMD( const char * cmd)
{
    puts( cmd);
    mygets();
```

```
}
```

Excelente! Já temos a estrutura para nossa aplicação. Antes de enviar comandos para o RN4020, vamos colocar uma instrução para fazer o LED D6 da Curiosity inverter o seu estado a cada 200 ms. Esse LED está ligado no pino RA2.

Exercício

Com o auxílio do MCC, configurar o LED D6 para saída. Inserir no código o comando para inversão do LED a cada 200 ms.

Lab 2 - Primeiros comandos para o RN4020

Agora que já criamos a estrutura para nosso projeto, vamos aprender a manipular o RN4020. Para nossa aplicação vamos usar o RN4020 em modo de comando. Sendo assim, o pino CM/MLDP do RN4020 deve ser colocado em nível 1. Além disso é necessário habilitar o módulo colocando o pino WAKE_SW em nível 1.

7	WAKE_SW	Deep Sleep Wake; active-high to wake module from Deep Sleep.	Input; weak pull-down
8	CMD/MLDP	CMD – Command Mode – Module enters Command mode where UART commands and responses sent over UART are exchanged between the RN4020 command interpreter and the MCU host. MLDP Mode – Data Mode – Data through UART is sent over the Bluetooth Low Energy connection to the remote device using MLDP data service.	Input; active-high to enter Command
9	GND	Ground.	Ground
10	CONNECTION_LED SCK PIO[1]	Default state is output: Active-high indicates the module is connected to a remote device. Active-low indicates a disconnected state. Configurable as PIO[1] via software command. SCK for diagnostics and factory calibration if pin 17 is asserted.	• Green LED • PIO[1] • SCK

Com o módulo configurado para comando e ativo, é necessário dar um reset através do comando SF,2. Esse comando executa um reset de fábrica, trazendo as configurações iniciais do módulo:

Após o reset, vamos configurar o módulo. Primeiro vamos configurar qual serviço será habilitado no módulo usando o comando SS. A tabela a seguir exibe os serviços disponíveis:

Service	Bitmap	Used in Profiles
Device Information	0x80000000	Blood Pressure, Cycling Speed Cadence, Glucose, Health Thermometer, Heart Rate, Running Speed Cadence
Battery	0x40000000	
Heart Rate	0x20000000	Heart Rate
Health Thermometer	0x10000000	Health Thermometer
Glucose	0x08000000	Glucose
Blood Pressure	0x04000000	Blood Pressure
Running Speed Cadence	0x02000000	Running Speed Cadence
Cycling Speed Cadence	0x01000000	Cycling Speed Cadence
Current Time	0x00800000	Time
Next DST Change	0x00400000	Time
Reference Time Update	0x00200000	Time
Link Loss	0x00100000	Proximity
Immediate Alert	0x00080000	Find Me, Proximity
TX Power	0x00040000	Proximity
Alert Notification	0x00020000	Alert Notification
Phone Alert Status	0x00010000	Phone Alert Status
Scan Parameters	0x00004000	Scan Parameters
User Defined Private Service	0x00000001	User Defined Private Profile

Para esse exemplo, vamos configurar apenas o Device Information: 0x80000000.

O próximo comando, SR, irá configurar as características do módulo. Para a nossa aplicação vamos apenas configurar o auto advertise: 0x2000000

Auto Advertise	0x20000000	This setting only applies to a peripheral device. If set, the device starts advertisement after a power cycle, reboot, or disconnection. If cleared, the device starts advertisement after receiving command "A" from the UART in Command mode.

Agora vamos nomear o dispositivo através do comando S-,<string>. Esse comando atribui um nome amigável e coloca ao final do nome os 2 últimos bytes do MAC.

Para efetivar as alterações é necessário um reboot usando o comando R,1.

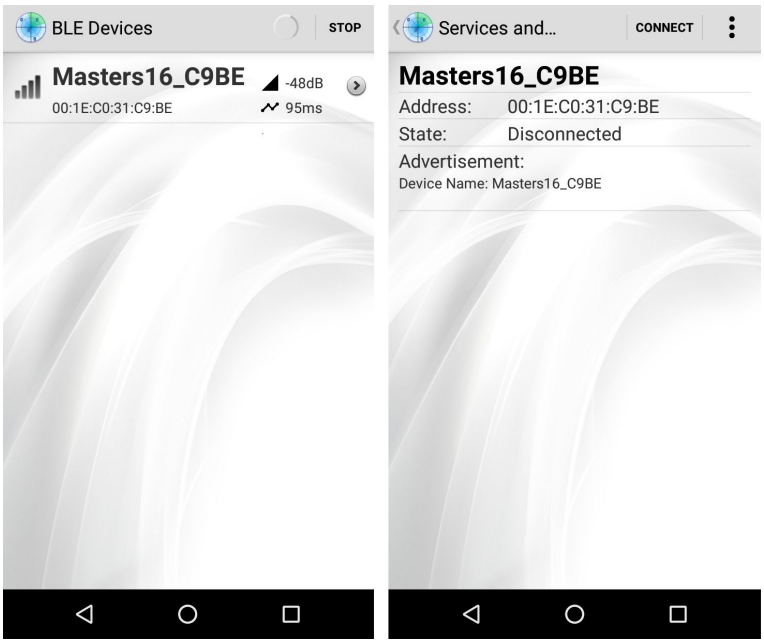
Pronto agora é só aguardar a conexão.

A seguir é apresentada a sequência de configuração na função main:

```
void main(void)
{
    // initialize the device
    SYSTEM_Initialize();

    // initialize RN4020
    CMD_SetHigh();           //command mode
    WAKE_SW_SetHigh();       // enter command mode
    mygets();                //wait response
    sendCMD( "SF,2");        // factory reset
    sendCMD( "SS,80000000"); // service
    sendCMD( "SR,20000000"); // set features
    sendCMD( "S-,Masters16"); // change name
    sendCMD( "R,1");         // reboot with new settings
    mygets();
    __delay_ms( cmdLONGDELAY);   // wait for CMD prompt
    while( !CONN_GetValue());   // wait for a connection

    while (1)
    {
        uint8_t time;              // auxiliary time variable

        if ( TMR2_HasOverflowOccured()) {
           time++;
        }

        if ( time >= 50)    // 200ms
        {
            time = 0;
            LED_D6_Toggle();
        }
    }
}
```

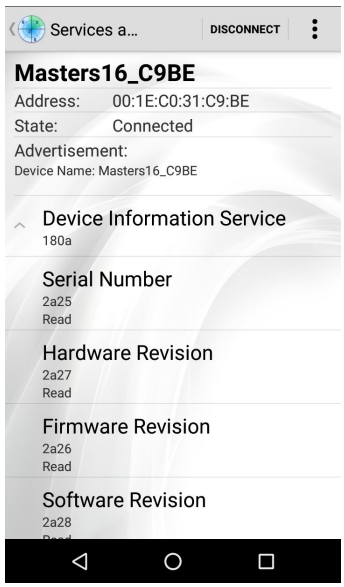
Compile o código e grave na placa.

Abra o aplicativo Smart Discovery no celular. Procure pelo nome configurado e conecte ao dispositivo:



Exercício

Leia as informações do RN4020 utilizando o app no smartphone



Lab 3 - Battery Service

Agora vamos explorar o serviço de bateria do Bluetooth.

Para habilitar esse serviço é necessário enviar o comando:

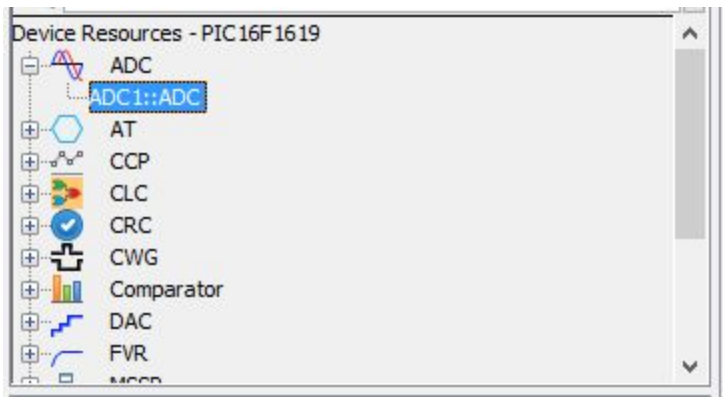
SS, C0000000

Este comando habilita o serviço de bateria e as informações do dispositivo. Altere o código de configuração do RN4020 para a seguinte forma:

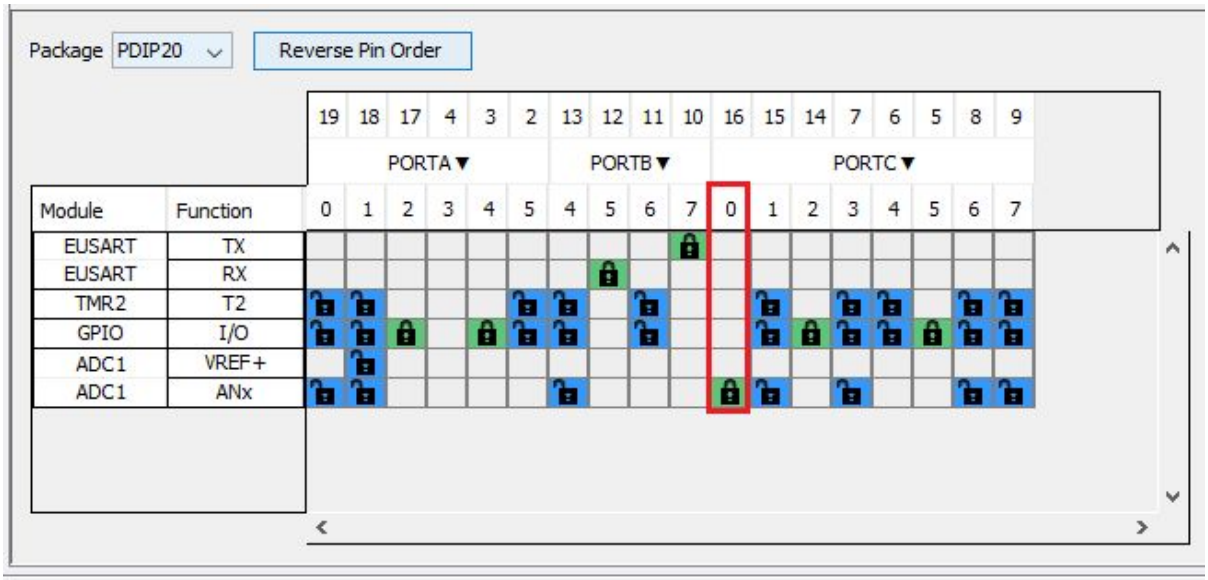
```
// initialize RN4020
    CMD_SetHigh();           //command mode
    WAKE_SW_SetHigh();       // enter command mode
    mygets();                //wait response
    sendCMD( "SF,2");        // factory reset
    sendCMD( "SS,C0000000"); // service
    sendCMD( "SR,20000000"); // set features
    sendCMD( "S-,Masters16"); // change name
    sendCMD( "R,1");         // reboot with new settings
    mygets();
    __delay_ms( cmdLONGDELAY);   // wait for CMD prompt
    while( !CONN_GetValue());   // wait for a connection
```

Para simular o valor da bateria, vamos configurar nossa aplicação para leitura do potenciômetro presente na placa Curiosity.

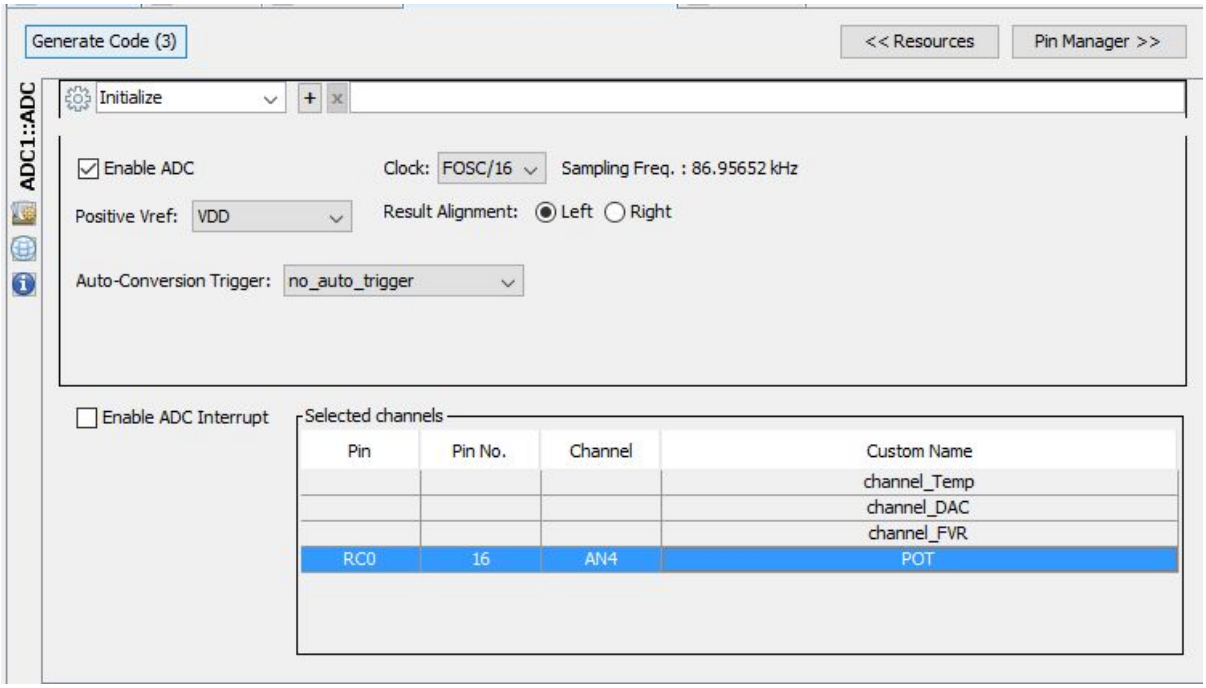
Abra o MCC e encontre o módulo ADC na lista de Device Resources:



Antes de configurar o módulo ADC, configure o pino RC0 como entrada do ADC1:



Agora configure o ADC1 da seguinte forma:



Obs. Nomeie o canal AN4 com o nome POT

Clique no botão Generate.

Para fazer a leitura do potenciômetro, vamos adicionar o seguinte trecho de código no loop principal:

```
while (1)
    {
        uint8_t time;              // auxiliary time variable

        if ( TMR2_HasOverflowOccured()) {
            time++;
        }

        if ( time >= 50)    // 200ms
        {
             uint8_t value;
            time = 0;
            LED_D6_Toggle();

             value =  ADC1_GetConversion(POT)>>9; // 0 to 127



        }
    }
```

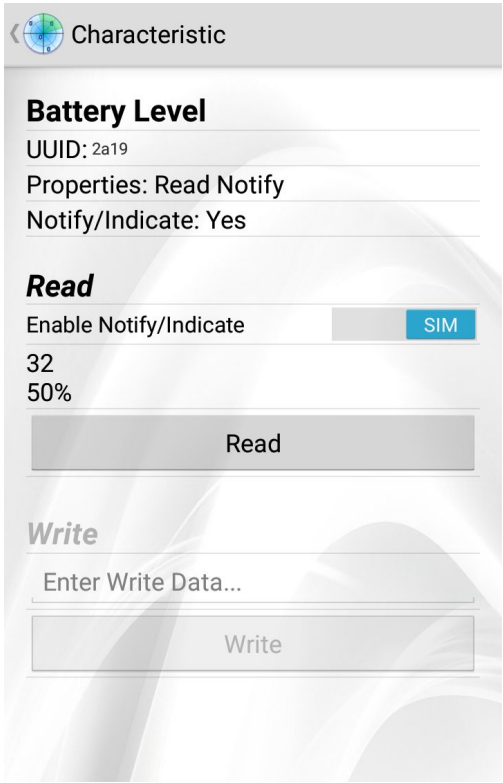
Para enviar o valor da bateria para a aplicação no celular, vamos usar o comando SUW. Esse comando escreve um valor no server UUID. O endereço do serviço de bateria é o 0X2A19.

Sendo assim, para a escrita do valor lido do potenciômetro, devemos enviar o comando SUW, logo após a leitura do potenciômetro, conforme exibido no trecho de código abaixo:

```
 while (1)
    {
        uint8_t time;              // auxiliary time variable

        if ( TMR2_HasOverflowOccured()) {
            time++;
        }

        if ( time >= 50)    // 200ms
        {
            uint8_t value;
            time = 0;
            LED_D6_Toggle();

            value =  ADC1_GetConversion(POT)>>9; // 0 to 127

             printf( "SUW,2A19,%02x\r\n", value); // update the battery level
            mygets();                            //wait response


        }
    }
```

Compile o programa e grave na placa Curiosity.

Abra o aplicativo Smart Discover, conecte a placa e leia o serviço de bateria(Battery Level):



Exercício

O serviço de bateria só recebe valores de 0 a 100%. Limite o valor enviado para essa faixa de valores.

Lab 4 - Heart Rate Service

Para configurar o Heart Rate Service, deve-se enviar o comando: SS,20000000.

Altere o código de configuração do RN4020 para a seguinte forma:

```
// initialize RN4020
    CMD_SetHigh();           //command mode
    WAKE_SW_SetHigh();       // enter command mode
    mygets();                //wait response
    sendCMD( "SF,2");        // factory reset
    sendCMD( "SS,20000000"); // service: Heart Rate
    sendCMD( "SR,20000000"); // set features
    sendCMD( "S-,Masters16"); // change name
    sendCMD( "R,1");         // reboot with new settings
    mygets();
    __delay_ms( cmdLONGDELAY);   // wait for CMD prompt
    while( !CONN_GetValue());   // wait for a connection
```

Vamos utilizar as mesmas configurações do potenciômetro para simular o Heart Rate Measurement.

Para envio dos valores deve-se usar a seguinte estrutura:

SUW,2A37,1FHHHHEEEERRRR

Onde:

HHHH - Heart Rate
EEEE - Energy Expended
RRRR - RR-Interval

Também é possível enviar a localização do sensor:

SUW,2A38,00

Onde o sensor pode estar localizado nas seguintes regiões:

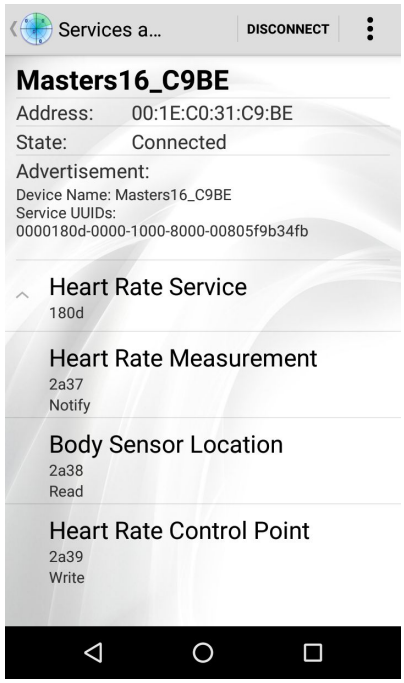
Names	Field Requirement	Format	Minimum Value	Maximum Value	Additional Information
Body Sensor Location	Mandatory	8bit	N/A	N/A	Enumerations

Key	Value
0	Other
1	Chest
2	Wrist
3	Finger
4	Hand
5	Ear Lobe
6	Foot
7 - 255	Reserved for future use

Assim, para testar esse serviço, inserir o seguinte código no loop principal:

```
while (1)
    {
        uint8_t time;               // auxiliary time variable

        if ( TMR2_HasOverflowOccured()) {
            time++;
        }

        if ( time >= 50)    // 200ms
        {
            uint8_t value;
            time = 0;
            LED_D6_Toggle();

            value =  ADC1_GetConversion(POT)>>9; // 0 to 127

            //Heart Rate
            printf( "SUW,2A37,1F%02x00%02x00%02x00\n", value,180,10);
            mygets();

            //Body Sensor Location
            printf( "SUW,2A38,%02x\n",1);
            mygets();
        }
    }
```

Abra o aplicativo Smart Discover, conecte a placa e leia as informações:



Exercício

Crie uma função para uso o Heart Rate.

Lab 5 - Health Thermometer

Para configurar o Health Thermometer Service, deve-se enviar o comando: SS,10000000.

Altere o código de configuração do RN4020 para a seguinte forma:

```
// initialize RN4020
    CMD_SetHigh();           //command mode
    WAKE_SW_SetHigh();       // enter command mode
    mygets();                //wait response
    sendCMD( "SF,2");        // factory reset
    sendCMD( "SS,10000000"); // service: Heart Rate
    sendCMD( "SR,20000000"); // set features
    sendCMD( "S-,Masters16"); // change name
    sendCMD( "R,1");         // reboot with new settings
```

```
    mygets();
    __delay_ms( cmdLONGDELAY);    // wait for CMD prompt
    while( !CONN_GetValue());    // wait for a connection
```

Vamos utilizar as mesmas configurações do potenciômetro para simular o Health Thermometer.

Para envio dos valores deve-se usar a seguinte estrutura:

SUW,2A1C,BBTTTT0000

Onde:

BB- São bits de Controle 01 = Fahrenheit; 00 = Celsius
TTTT - Temperatura

Também é possível enviar o tipo da temperatura:

SUW,2A1D,00

Onde o valor pode ser:

Names	Field Requirement	Format	Minimum Value	Maximum Value	Additional Information
Temperature Text Description	Mandatory	8bit	N/A	N/A	Enumerations

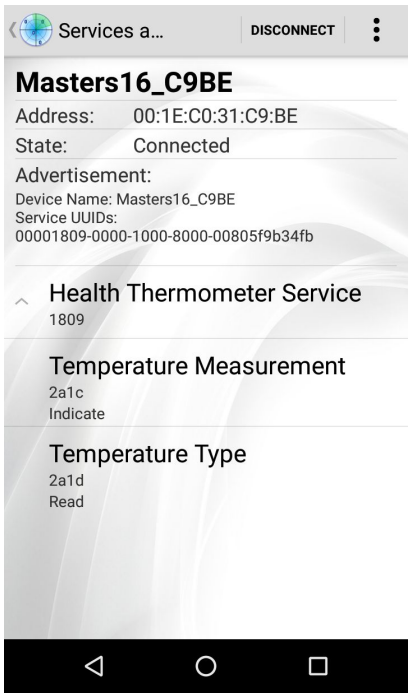
Key	Value
1	Armpit
2	Body (general)
3	Ear (usually ear lobe)
4	Finger
5	Gastro-intestinal Tract
6	Mouth
7	Rectum
8	Toe
9	Tympanum (ear drum)
10 - 255	Reserved for future use
0 - 0	Reserved for future use

Assim, para testar esse serviço, inserir o seguinte código no loop principal:

```
while (1)
    {
        uint8_t time;              // auxiliary time variable

        if ( TMR2_HasOverflowOccured()) {
            time++;
```

```
        }

        if ( time >= 50)    // 200ms
        {
            uint8_t value;
            time = 0;
            LED_D6_Toggle();

            value =  ADC1_GetConversion(POT)>>9; // 0 to 127

           //Health Thermometer
           printf( "SUW,2A1C,01%02x000000\n",value);
           mygets();

           //temperature type
           printf( "SUW,2A1D,%02x\n",2);
           mygets();


        }
    }
```

Abra o aplicativo Smart Discover, conecte a placa e leia as informações:



Exercício

Crie uma função para facilitar o uso de serviço.