



LABORATORIO DE MICROCONTROLADORES

PRÁCTICA

4

ALUMNOS

A01552401 JOSELYN CONTRERAS PEREGRINA

A01230927 SAUL GONZALEZ FIGUEROA

PROFESOR

MATIAS VÁZQUEZ

5 DE ABRIL DEL 2022

ÍNDICE

Introducción	3
Procedimiento de trabajo en laboratorio.	3
Conclusiones	15

Introducción

Durante la sesión de laboratorio nos familiarizamos con la herramienta del entorno de desarrollo integrado MPLAB y creamos nuestro primer programa. Este programa se cargó al microcontrolador PIC18 haciendo uso de la placa de desarrollo Curiosity.

La placa de desarrollo Curiosity es de Microchip Technology y presenta una plataforma de desarrollo completamente integrada, rentable de 8 bits. Admite microcontroladores PIC de 28 o 40 pines y de 8 bits con capacidad de programación de bajo voltaje, tiene programador/depurador integrado con interfaz USB y se integra perfectamente con IDE MPLAB X y con el configurador de código.

Por otro lado, MPLAB es un software profesional implementado por la empresa Microchip. MPLAB IDE es utilizado como un poderoso auxiliar para el desarrollo de sistemas basados en los microcontroladores PIC. El programa incluye un editor de texto, macro-ensamblador, compilador ANSI C, y simulador para trabajar con cualquier microcontrolador PIC. El simulador puede operar tanto en programas desarrollados en lenguaje ensamblador o ANSI C.

Procedimiento de trabajo en laboratorio.

En primera instancia se solicitó placa de desarrollo Curiosity de Microchip Technology en el laboratorio y se conectó por medio de la USB a la PC.



Figura 1. Placa de desarrollo Curiosity

Una vez así se inició MPLAB X IDE v6.00 y se creó un nuevo archivo en la categoría de Microchip embebido y proyecto independiente.

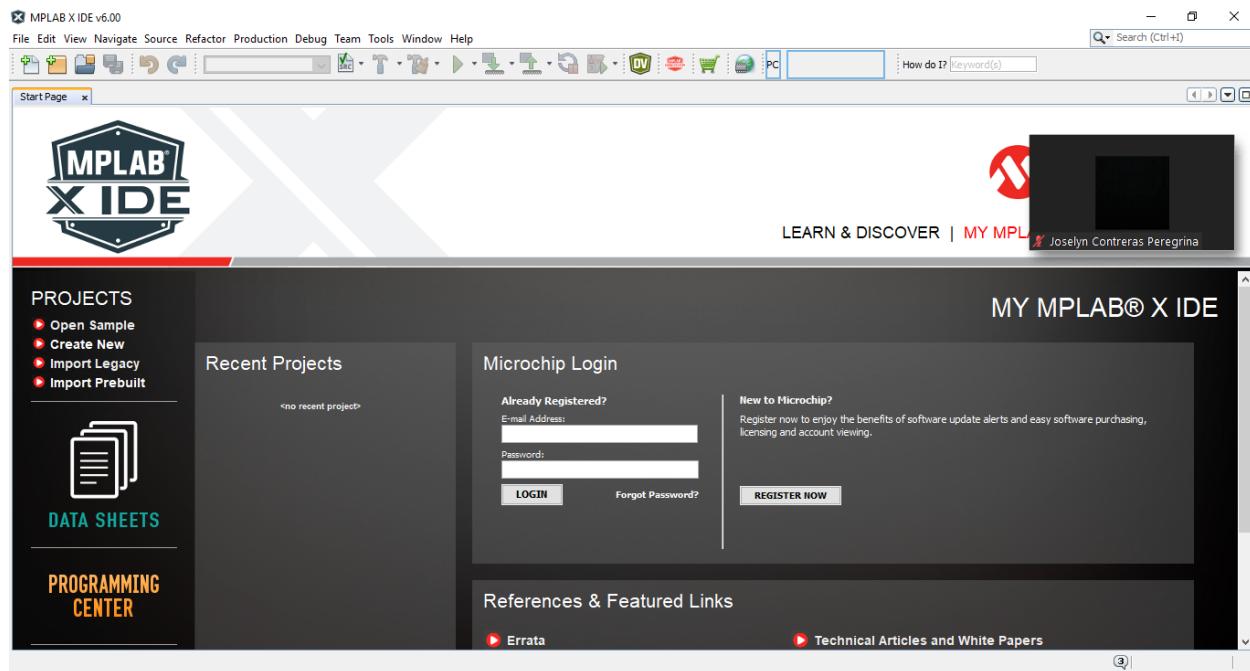


Figura 2. MPLAB X IDE iniciado

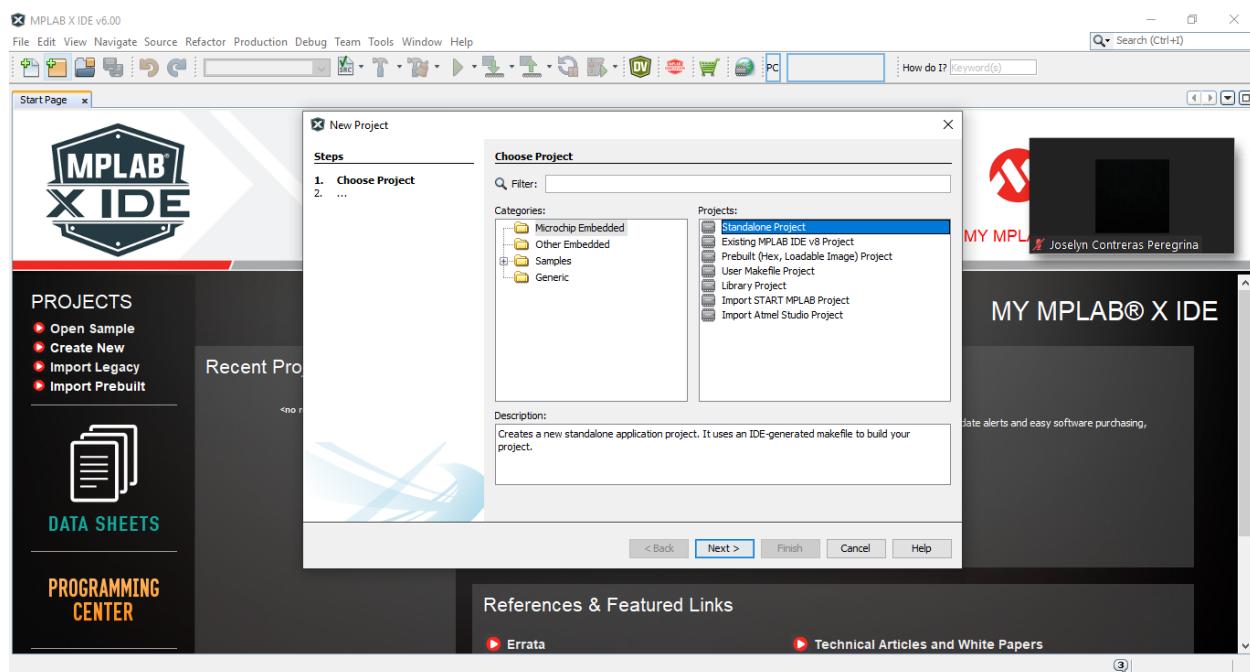


Figura 3. Creación del nuevo proyecto

Se seleccionó el dispositivo PIC18F45K50 y la herramienta Curiosity/ Starter Kits

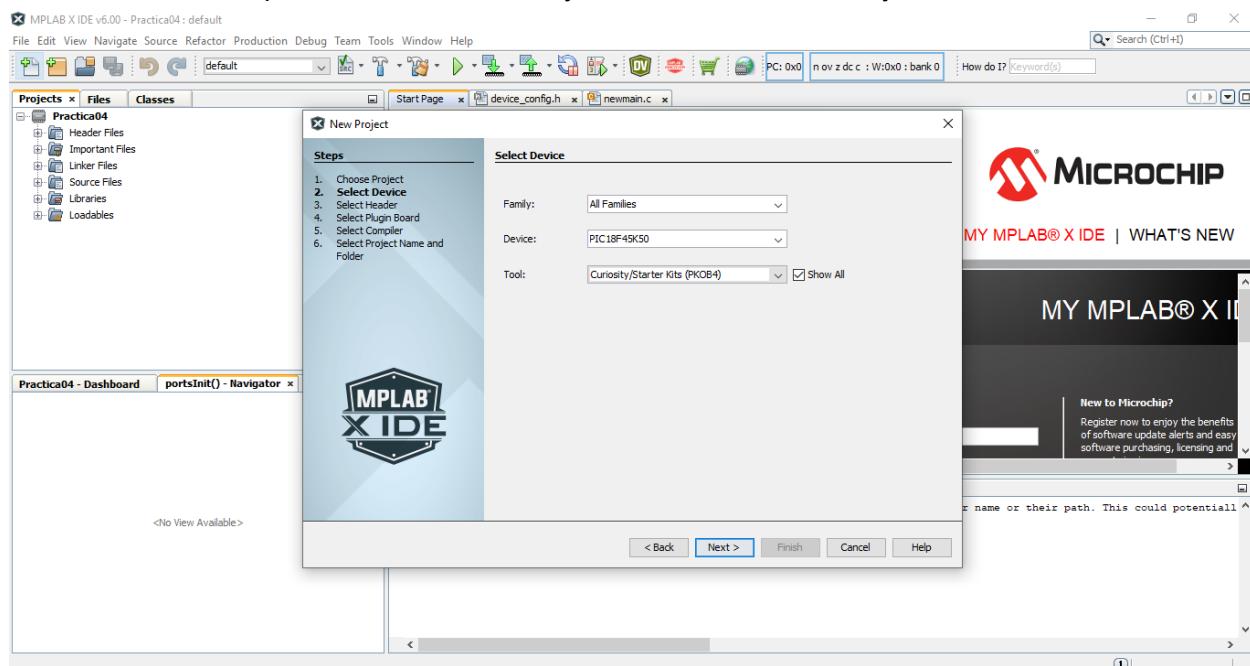


Figura 4. Configuración Select Device

Posteriormente se seleccionó el compilador instalado previamente, XC8 (v2.36)

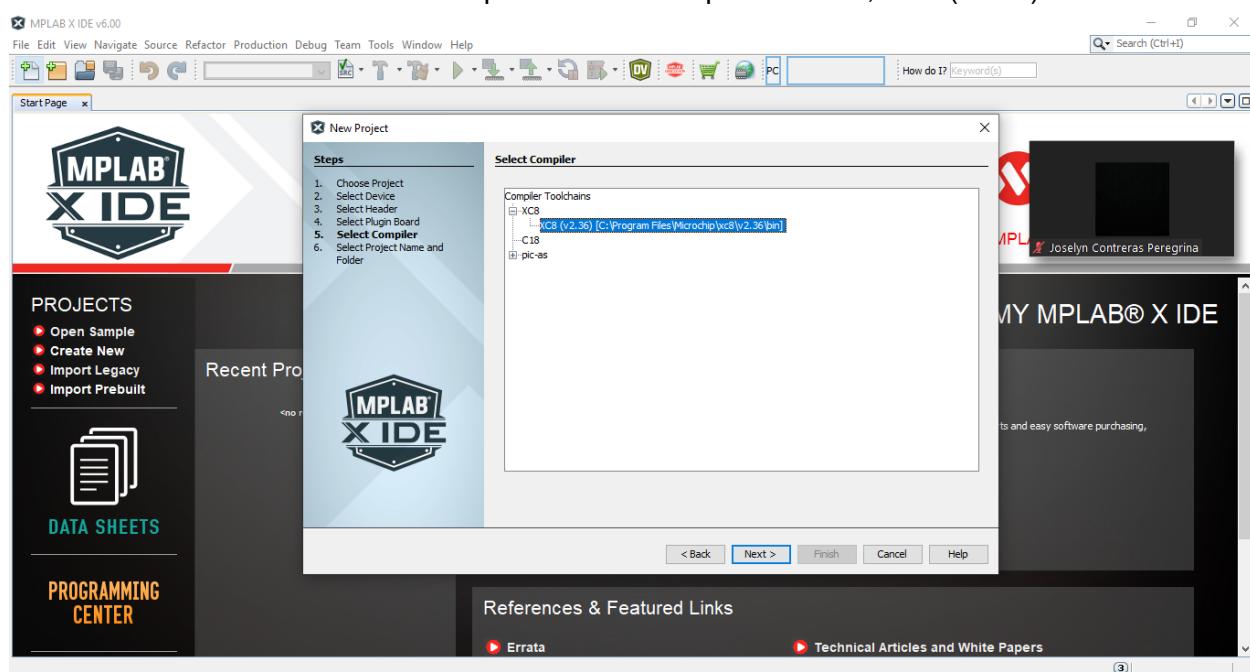


Figura 5. Configuración Compilador

Una vez así, se nombró el proyecto y se eligió su ubicación. Se mantuvieron las configuraciones de Set as main project y el Encoding ISO-8859-1

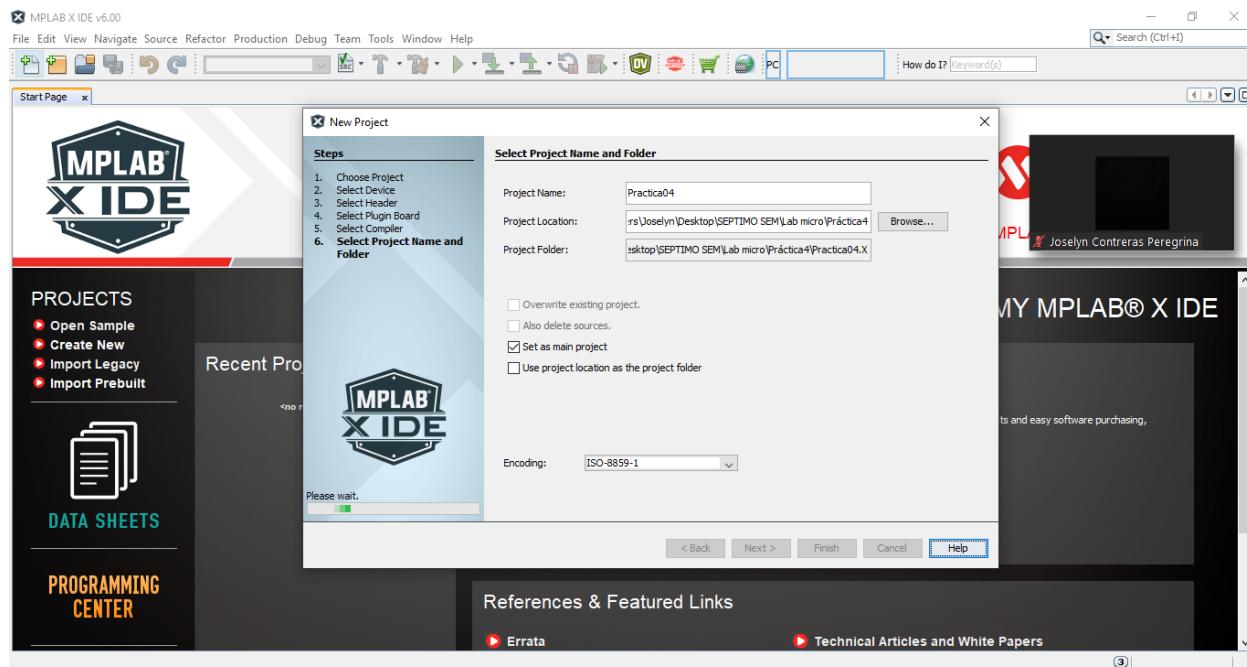


Figura 6. Nombre y folder del proyecto

Se procedió a crear un archivo main.c bombrado como newmain.c , recordando que la extensión a utilizar es c

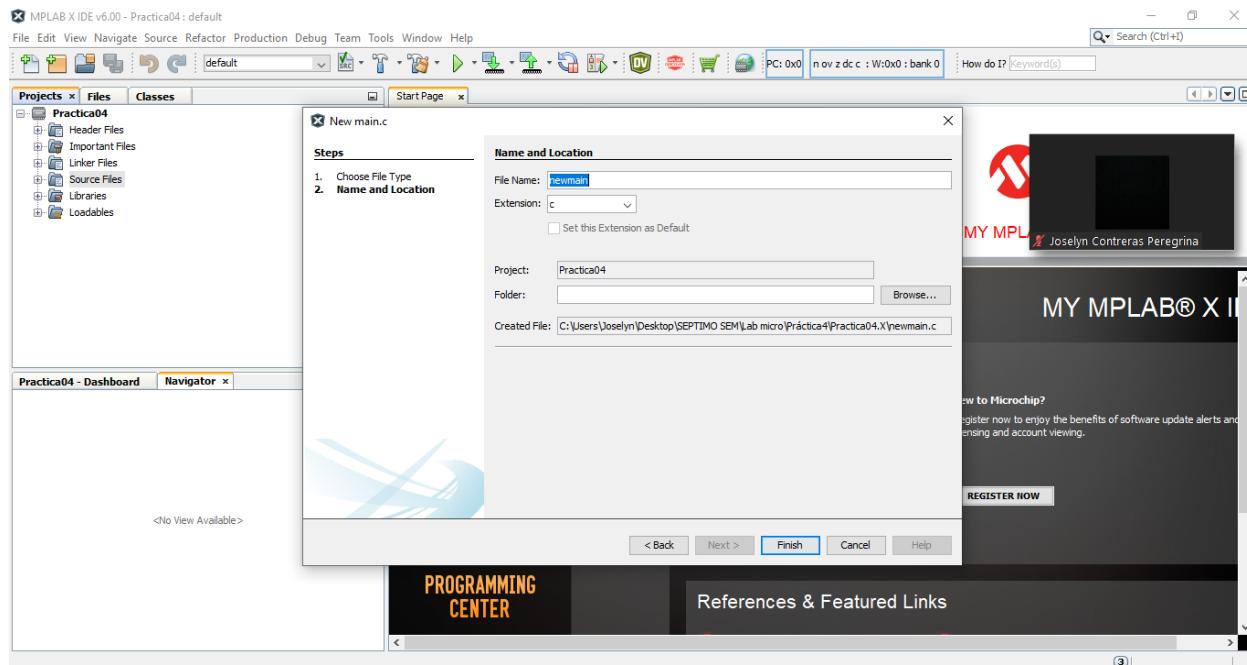


Figura 7. Creación de newmain.c

En esta ventana se copió el código proporcionado en GitHub compuesto de 46 líneas:

```
||||||||||||||||||||||||||||| LIBRARIES / HEADERS
|+++++|+
#include "device_config.h"
//||||||||||||||||||||||||| DIRECTIVES
|+++++|+
#define _XTAL_FREQ 1000000
#define ONE_SECOND 1000

//||||||||||||||||||||||||| DATA TYPES
|+++++|+
enum por_dir{ output, input };           // output = 0, input = 1
enum por_ACDC { digital, analog };      // digital = 0, analog = 1
enum resistor_state { set_ON, res_ON };  // set_ON = 0, res_ON = 1
enum led_state { led_OFF, led_ON };      // led_OFF = 0, led_ON = 1
enum butto_state { pushed, no_pushed };   // pushed = 0, no_pushed = 1

//||||||||||||||||||||| ISRs
|+++++|+
// ISR for high priority
void __interrupt ( high_priority ) high_isr( void );
// ISR for low priority
void __interrupt ( low_priority ) low_isr( void );

//||||||||||||||||||||| FUNCTION DECLARATIONS
|+++++|+
void portsInit( void );

//||||||||||||||||||||| MAIN
|+++++|+
void main( void ){
    portsInit();
    while(1){
        if(PORTBbits.RB4 == pushed)      // If button is pushed then
            LATAbits.LATA7 = led_OFF;   // turn off RA4 LED
        else                           // Otherwise
            LATAbits.LATA7 = led_ON;    // turn on RA4 LED
        LATAbits.LATA4 = led_ON;        // Turn on RA4 LED
        __delay_ms(ONE_SECOND);        // Delay function XC8 compiler
        LATAbits.LATA4 = led_OFF;      // Turn off RA4 LED
        __delay_ms(ONE_SECOND);        // Delay function XC8 compiler
    }
}

//||||||||||||||||||||| FUNCTIONS
|+++++|+
void portsInit( void ){
    ANSELA = digital;             // Set Port A as digital port
    TRISAbits.TRISA4 = output;    // RA4 as output
```

```

    TRISAbits.TRISA7 = output;           // RA7 as output
    ANSELB = digital;                  // Set Port B as digital port
    TRISBbits.TRISB4 = input;          // Set RB4 as input
}

}

```

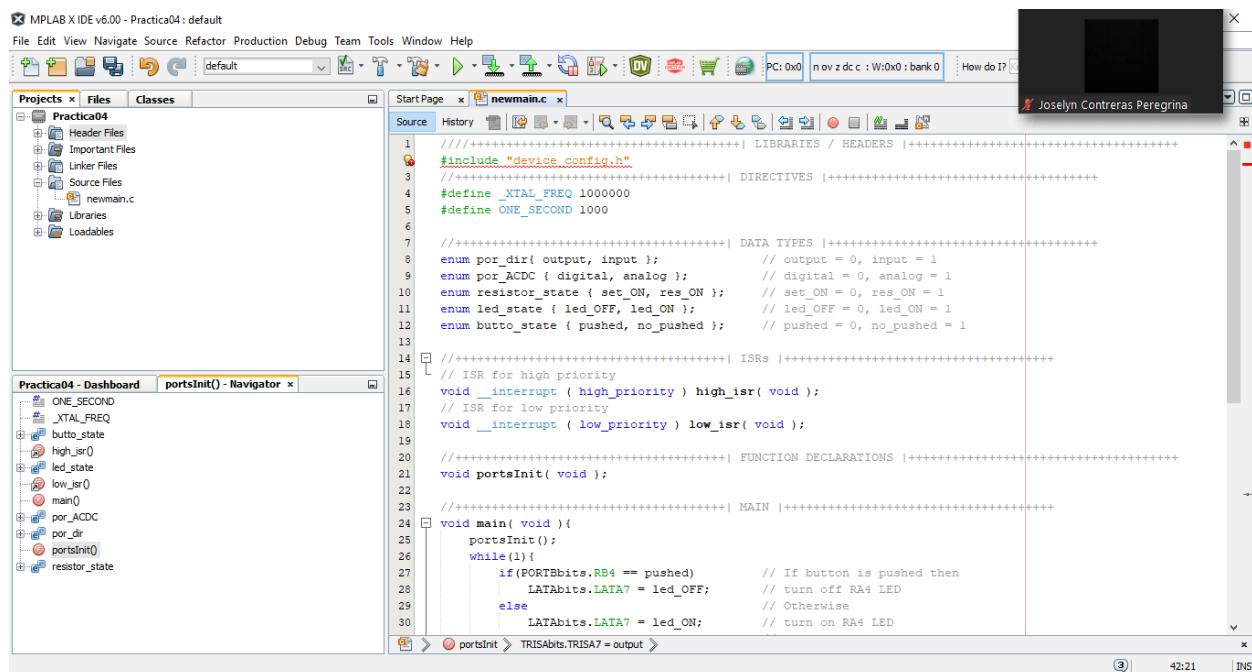


Figura 8. Código de newmain.c

Así mismo se procedió a crear un archivo xc8_header.h bombrado como device_config.h , recordando que la extensión a utilizar es h. En la nueva ventana se copió el código proporcionado en GitHub compuesto de 72 líneas:

```

// PIC18F45K50 Configuration Bit Settings
#include<xc.h>
// 'C' source line config statements

// CONFIG1L
#pragma config PLLSEL = PLL4X // PLL Selection (4x clock multiplier)
#pragma config CFGPLLLEN = OFF // PLL Enable Configuration bit (PLL Disabled (firmware controlled))
#pragma config CPUDIV = NOCLKDIV// CPU System Clock Postscaler (CPU uses system clock (no divide))
#pragma config LS48MHZ = SYS24X4// Low Speed USB mode with 48 MHz system clock (System clock at 24 MHz, USB clock divider is set to 4)

// CONFIG1H
#pragma config FOSC = INTOSCIO // Oscillator Selection (Internal oscillator)

```

```

#pragma config PCLKEN = ON      // Primary Oscillator Shutdown (Primary oscillator enabled)
#pragma config FCMEN = OFF     // Fail-Safe Clock Monitor (Fail-Safe Clock Monitor disabled)
#pragma config IESO = OFF      // Internal/External Oscillator Switchover (Oscillator Switchover
mode disabled)

// CONFIG2L
#pragma config nPWRREN = OFF   // Power-up Timer Enable (Power up timer disabled)
#pragma config BOREN = SBORDIS // Brown-out Reset Enable (BOR enabled in hardware
(SBoren is ignored))
#pragma config BORV = 190       // Brown-out Reset Voltage (BOR set to 1.9V nominal)
#pragma config nLPBOR = OFF    // Low-Power Brown-out Reset (Low-Power Brown-out Reset
disabled)

// CONFIG2H
#pragma config WDTEN = OFF     // Watchdog Timer Enable bits (WDT disabled in hardware
(SWDten ignored))
#pragma config WDTPS = 32768    // Watchdog Timer Postscaler (1:32768)

// CONFIG3H
#pragma config CCP2MX = RC1    // CCP2 MUX bit (CCP2 input/output is multiplexed with
RC1)
#pragma config PBADEN = ON     // PORTB A/D Enable bit (PORTB<5:0> pins are configured
as analog input channels on Reset)
#pragma config T3CMX = RC0     // Timer3 Clock Input MUX bit (T3CKI function is on RC0)
#pragma config SDOMX = RB3     // SDO Output MUX bit (SDO function is on RB3)
#pragma config MCLRE = ON      // Master Clear Reset Pin Enable (MCLR pin enabled; RE3
input disabled)

// CONFIG4L
#pragma config STVREN = ON     // Stack Full/Underflow Reset (Stack full/underflow will cause
Reset)
#pragma config LVP = ON        // Single-Supply ICSP Enable bit (Single-Supply ICSP enabled if
MCLRE is also 1)
#pragma config ICPRT = OFF     // Dedicated In-Circuit Debug/Programming Port Enable
(ICPORT disabled)
#pragma config XINST = OFF     // Extended Instruction Set Enable bit (Instruction set
extension and Indexed Addressing mode disabled)

// CONFIG5L
#pragma config CP0 = OFF       // Block 0 Code Protect (Block 0 is not code-protected)
#pragma config CP1 = OFF       // Block 1 Code Protect (Block 1 is not code-protected)
#pragma config CP2 = OFF       // Block 2 Code Protect (Block 2 is not code-protected)
#pragma config CP3 = OFF       // Block 3 Code Protect (Block 3 is not code-protected)

```

```
// CONFIG5H
#pragma config CPB = OFF      // Boot Block Code Protect (Boot block is not code-protected)
#pragma config CPD = OFF      // Data EEPROM Code Protect (Data EEPROM is not
code-protected)

// CONFIG6L
#pragma config WRT0 = OFF      // Block 0 Write Protect (Block 0 (0800-1FFFh) is not
write-protected)
#pragma config WRT1 = OFF      // Block 1 Write Protect (Block 1 (2000-3FFFh) is not
write-protected)
#pragma config WRT2 = OFF      // Block 2 Write Protect (Block 2 (04000-5FFFh) is not
write-protected)
#pragma config WRT3 = OFF      // Block 3 Write Protect (Block 3 (06000-7FFFh) is not
write-protected)

// CONFIG6H
#pragma config WRTC = OFF      // Configuration Registers Write Protect (Configuration
registers (300000-3000FFh) are not write-protected)
#pragma config WRTB = OFF      // Boot Block Write Protect (Boot block (0000-7FFh) is not
write-protected)
#pragma config WRTD = OFF      // Data EEPROM Write Protect (Data EEPROM is not
write-protected)

// CONFIG7L
#pragma config EBTR0 = OFF      // Block 0 Table Read Protect (Block 0 is not protected from
table reads executed in other blocks)
#pragma config EBTR1 = OFF      // Block 1 Table Read Protect (Block 1 is not protected from
table reads executed in other blocks)
#pragma config EBTR2 = OFF      // Block 2 Table Read Protect (Block 2 is not protected from
table reads executed in other blocks)
#pragma config EBTR3 = OFF      // Block 3 Table Read Protect (Block 3 is not protected from
table reads executed in other blocks)

// CONFIG7H
#pragma config EBTRB = OFF      // Boot Block Table Read Protect (Boot block is not protected
from table reads executed in other blocks)

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.
```

The screenshot shows the MPLAB X IDE interface with the project 'Practica04' open. The 'device_config.h' file is selected in the 'Files' tab. The code in the editor is as follows:

```

1 // PIC18F45K50 Configuration Bit Settings
2 #include<xc.h>
3 // 'C' source line config statements
4
5 // CONFIG1L
6 #pragma config PLLSEL = PLL4X // PLL Selection (4x clock multiplier)
7 #pragma config CFGPLLLEN = OFF // PLL Enable Configuration bit (PLL Disabled (firmware controlled))
8 #pragma config CFUDIV = NOCLKDIV// CPU System Clock Postscaler (CPU uses system clock (no divide))
9 #pragma config LS48MHZ = SYS24X4// Low Speed USB mode with 48 MHz system clock (System clock at 24 MHz, USB
10
11 // CONFIG1H
12 #pragma config FOSC = INTOSCIO // Oscillator Selection (Internal oscillator)
13 #pragma config PCLKEN = ON // Primary Oscillator Shutdown (Primary oscillator enabled)
14 #pragma config FCNEN = OFF // Fail-Safe Clock Monitor (Fail-Safe Clock Monitor disabled)
15 #pragma config IESO = OFF // Internal/External Oscillator Switchover (Oscillator Switchover mode disabled)
16
17 // CONFIG2L
18 #pragma config nPWRTEN = OFF // Power-up Timer Enable (Power up timer disabled)
19 #pragma config BOREN = SBOREN // Brown-out Reset Enable (BOR enabled in hardware (SBOREN is ignored))
20 #pragma config BORV = 190 // Brown-out Reset Voltage (BOR set to 1.9V nominal)
21 #pragma config nLPBOR = OFF // Low-Power Brown-out Reset (Low-Power Brown-out Reset disabled)
22
23 // CONFIG2H
24 #pragma config WDTEN = OFF // Watchdog Timer Enable bits (WDT disabled in hardware (SWDTEN ignored))
25 #pragma config WDTPS = 32768 // Watchdog Timer Postscaler (1:32768)
26
27 < >

```

Figura 9. Código de device_config.h

Posteriormente en Herramientas, se seleccionó la opción de Embebidos para acceder a sus configuraciones genéricas y se eligieron las opciones siguientes, Restablecimiento de depuración: Restablecer vector, Inicio de depuración: detener en el vector de reinicio.

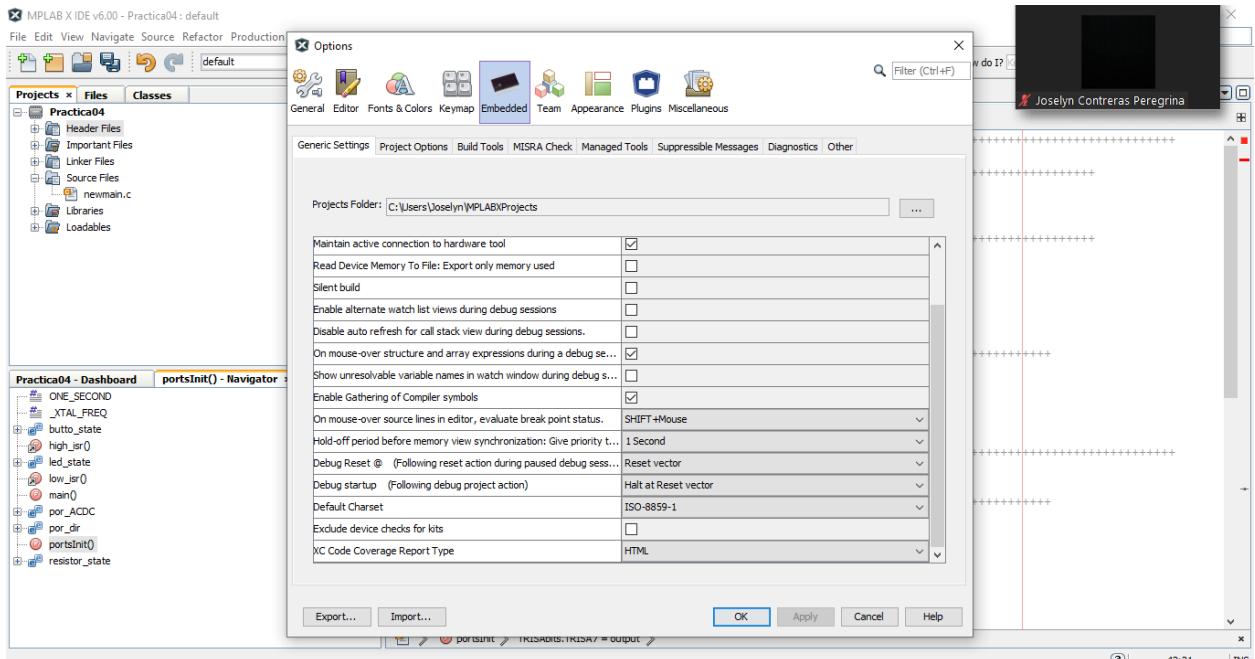


Figura 10. Configuración de embebidos

Finalmente, se configuraron las propiedades del proyecto asegurándonos que Erase All Before Program estuviese seleccionado.

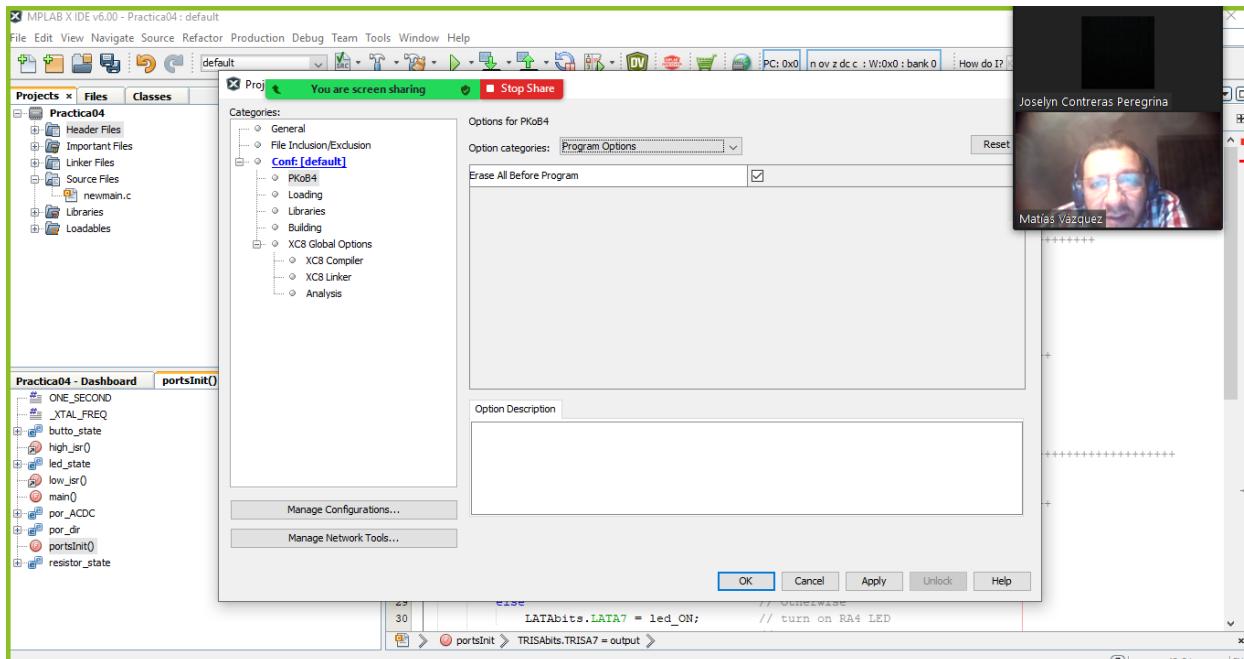


Figura 11. Propiedades de proyecto

Nos aseguramos de que la opción del modo de programación para bajo voltaje estuviese activada y se depuró el proyecto principal. Se observó el mensaje de BUILD SUCCESSFUL al cargar el programa a la placa de pruebas.

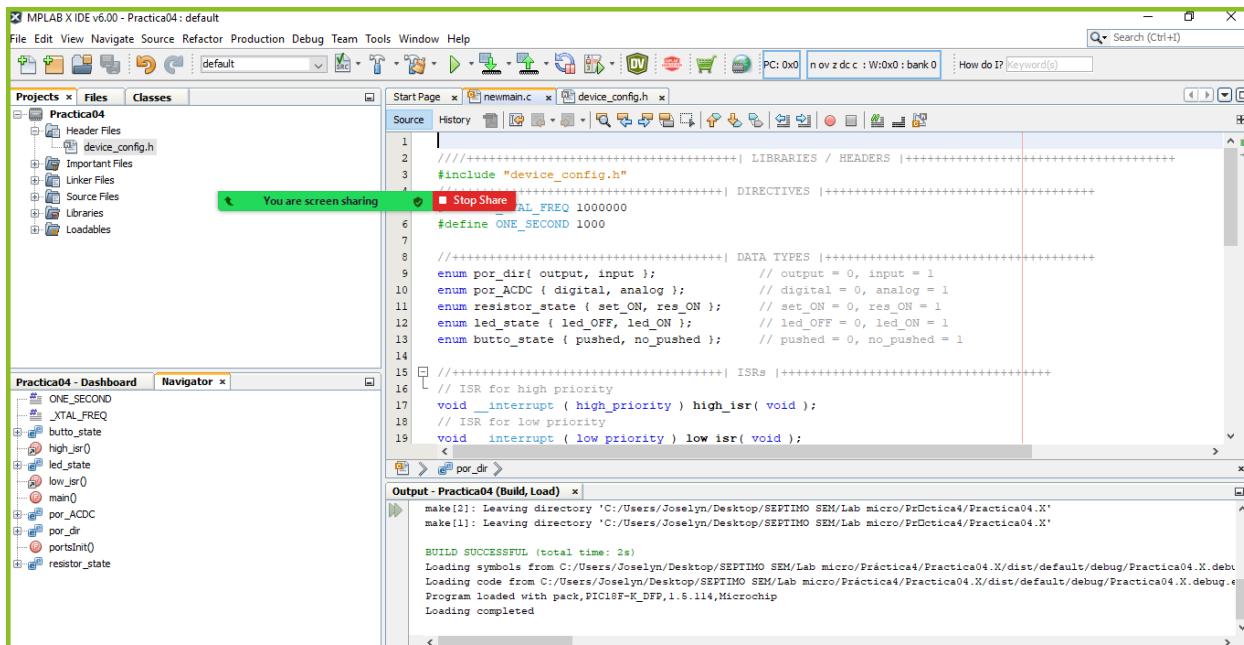


Figura 12. Carga exitosa del programa

A continuación se muestra el parpadeo del LED de la placa de pruebas

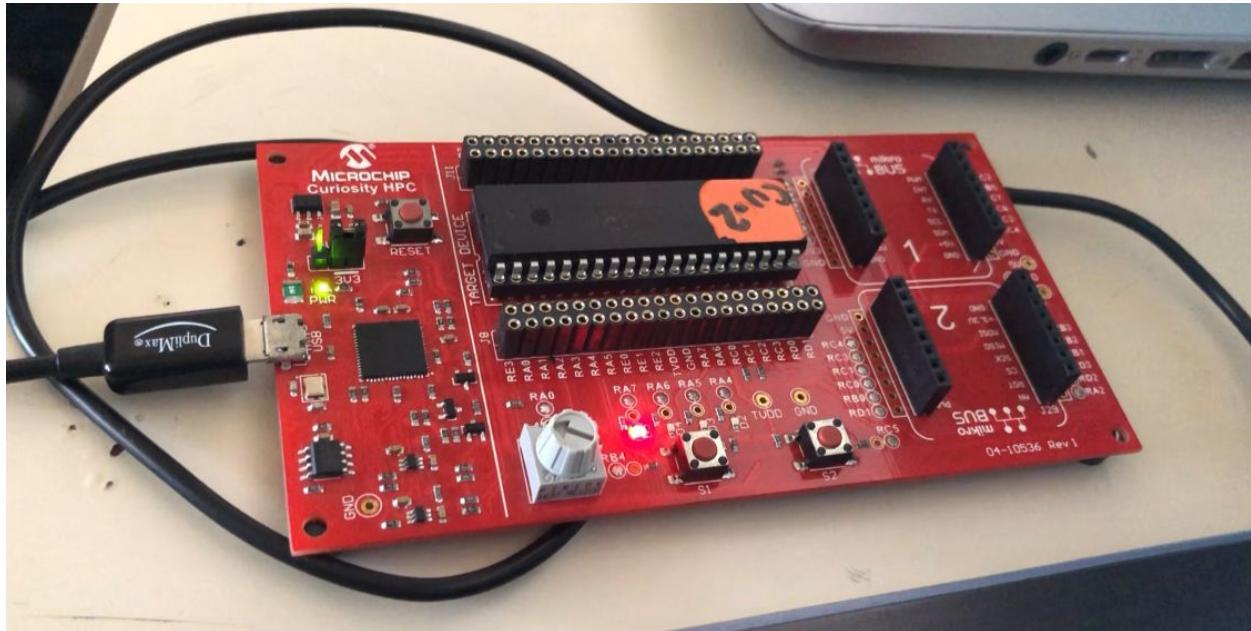


Figura 13. LED RA4 ON, LED RA7 ON

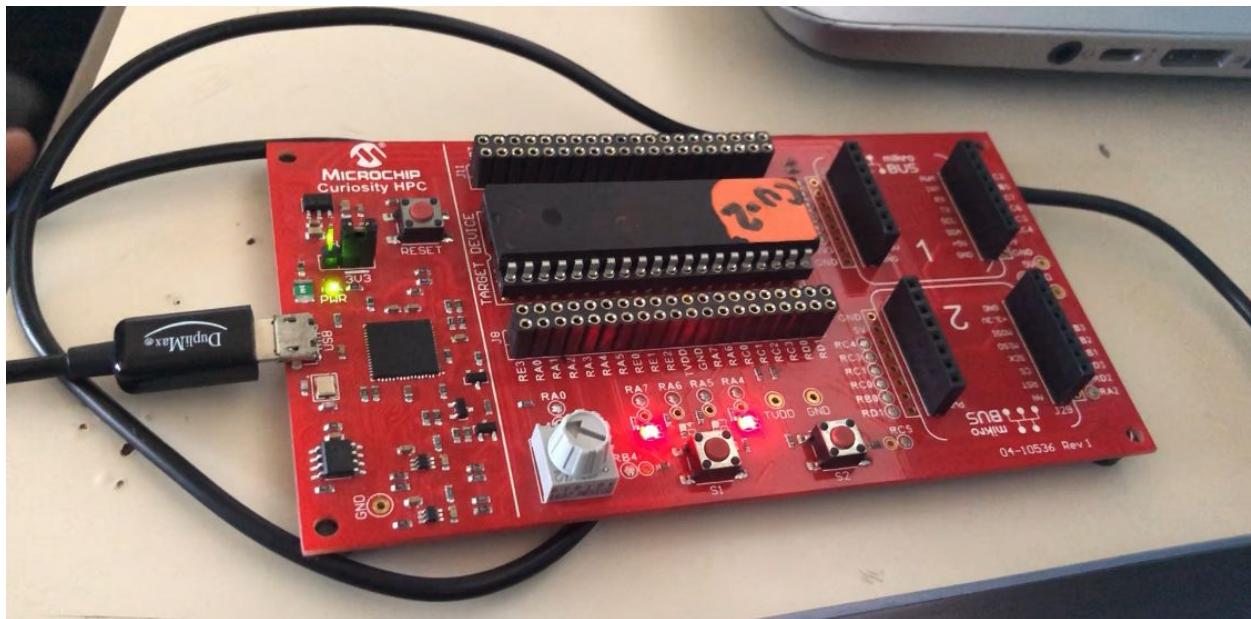


Figura 14. LED RA4 OFF, LED RA7 ON

Si el botón RB4 no es pulsado, entonces el led RA7 permanece encendido mientras que RA4 permanece encendido un segundo y apagado otro segundo. En cambio, si RB4 es pulsado, apaga RA7 y RA4 continúa encendiéndose y apagándose durante un ciclo hasta verificar la condición.

- Link Github repositorio de GitHub con los archivos del proyecto MPLAB, incluidos los archivos fuente y el main.c comentado por nosotros.
https://github.com/JoselynCP/Practica04_A01552401_A01230927.git
- Enlace al vídeo donde se muestra y explica el funcionamiento del proyecto en la placa Curiosity.
https://drive.google.com/file/d/1opdV1qiQhh8csKQTvWV3I_dtCMvAIYdU/view?usp=sharing
- Imagen de la ventana BUILD SUCCESSFUL

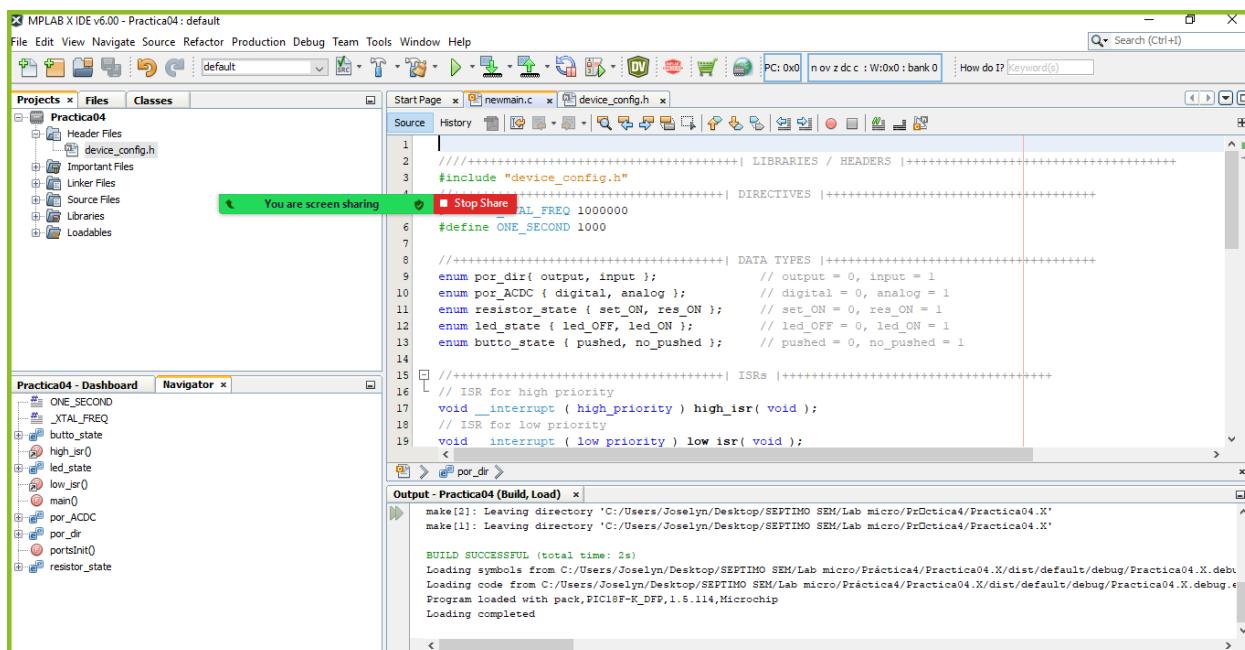


Figura 15. Carga exitosa del programa

Conclusiones

Joselyn Contreras Peregrina: Esta práctica fue muy interesante ya que permitió conjuntar el hardware con el software que vamos a continuar trabajando por el resto del semestre. Así mismo, considero que fue una práctica ideal para repasar contenido e interpretación de códigos además de que aprendimos a elaborar las configuraciones necesarias para poder cargar de forma exitosa un proyecto a nuestra placa de pruebas. A pesar de que no elaboramos todavía el código nosotros mismos, hubo que interpretarlo correctamente para no solo poder explicarlo sino también entenderlo y darnos una idea de lo que estaba sucediendo con la comunicación entre la Curiosity y el MPLAB.

Saul Gonzalez Figueroa: Lo más relevante de esta práctica fue conocer físicamente la placa Curiosity y aprender a cargar de manera exitosa el programa. El código se nos proporcionó pero al leerlo pudimos empezar a familiarizarnos con la sintaxis y la estructura que tiene.