Hernández Diaz Sebastian

Laboratorio de Microcomputadoras

Práctica No. 9

Programación C, Convertidor A/D e Interrupciones

Objetivo. Realización de programas usando programación en lenguaje C, utilización del puerto serie, convertidor analógico digital e introducción a aplicaciones con interrupciones.

Desarrollo. Realizar los siguientes ejercicios.

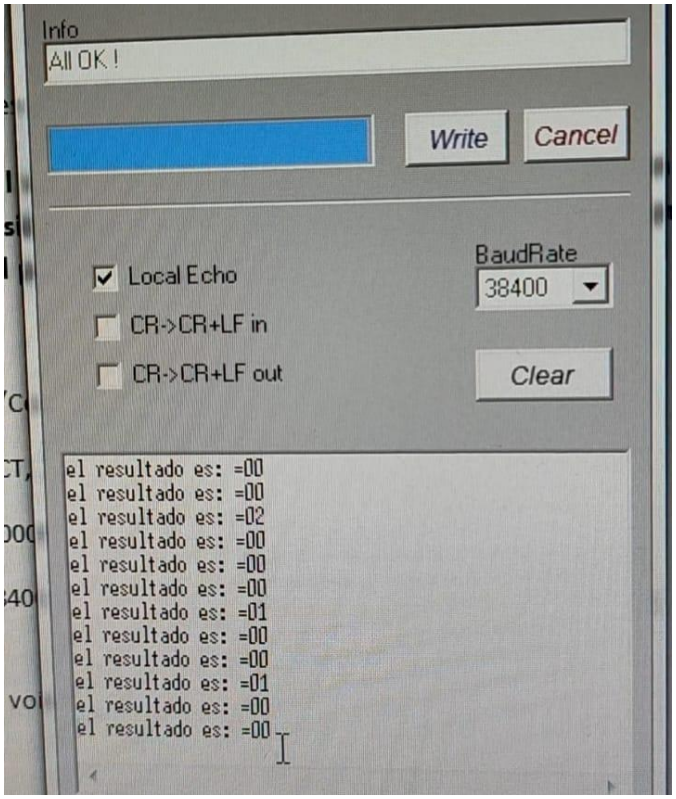
1.- Programa el cual obtenga una señal analógica a través del canal de su elección, se realice la conversión y el resultado de esta, la muestre en un puerto paralelo y a su vez lo trasmita al puerto serie.

El código usado para esta actividad fue el siguiente:

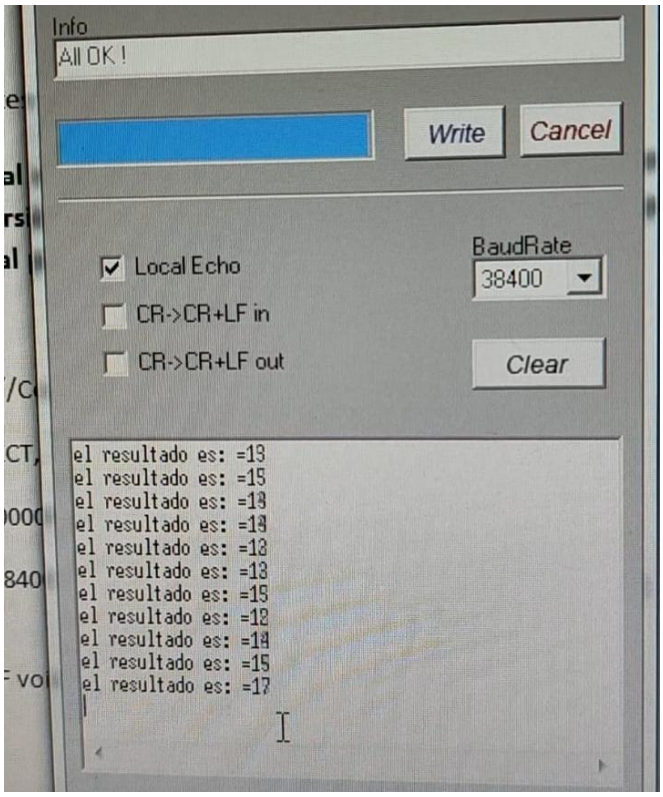
```
#include <16f877.h>
#device ADC = 8,                                    // Convertidor analogico digital con 8
#fuses HS, NOPROTECT,                               // Transmision
#use delay(clock = 20000000)                        // Reloj a 20mhz
#use rs232(baud = 38400, xmit = PIN_C6, rcv = PIN_C7) // Transmision rs232. PIN transmision y recepcion
#org 0x1F00, 0x1FFF void loader16F877(void){}       // for the 8k 16F876/7
int converse;
void main()
{
   setup_port_a(ALL_ANALOG);      // Todo analogico
   setup_adc(ADC_CLOCK_INTERNAL); // ADC que trabaje con reloj interno
   set_adc_channel(5);            // Habilitar canal 5

   while (1)
   {
      delay_ms(20);                          // Retardo de 20ms
      converse = read_adc();                 // Guardamos conversion en converse
      printf("el resultado es: =%x\n", converse); // Imprimimos
      output_b(converse);                    // Salida en puerto B
   }
}
```

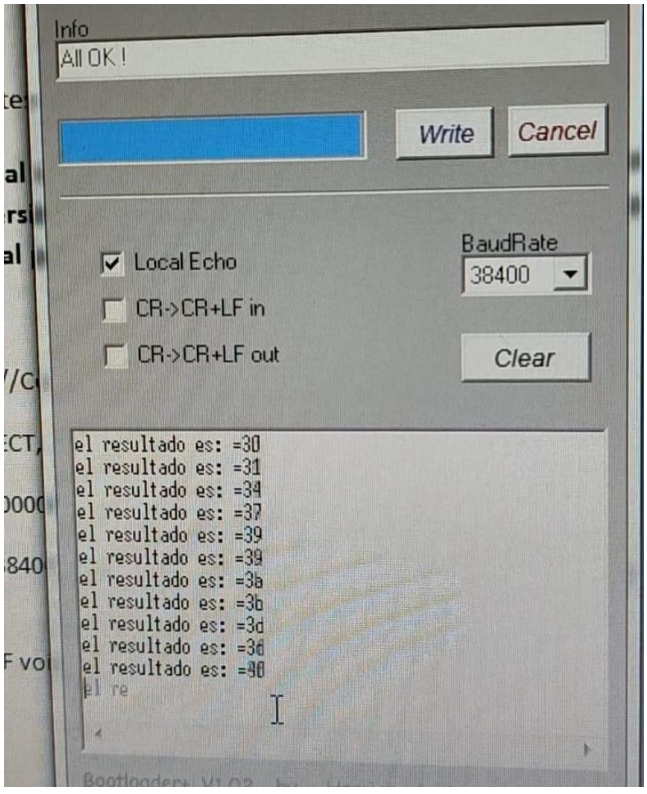
Este código fue diseñado para que el microcontrolador PIC16F877 lea continuamente un valor analógico proveniente del canal 5 de su convertidor analógico-digital (ADC) con una resolución de 8 bits, usando su reloj interno. Cada 20 milisegundos realiza una lectura del ADC, guarda el resultado en la variable converse, lo imprime por el puerto serial RS232 en formato hexadecimal y al mismo tiempo envía el valor al puerto B del microcontrolador como salida digital, permitiendo así monitorear el valor leído tanto por comunicación serial como mediante dispositivos conectados al puerto B.

La ejecución del código es la siguiente:

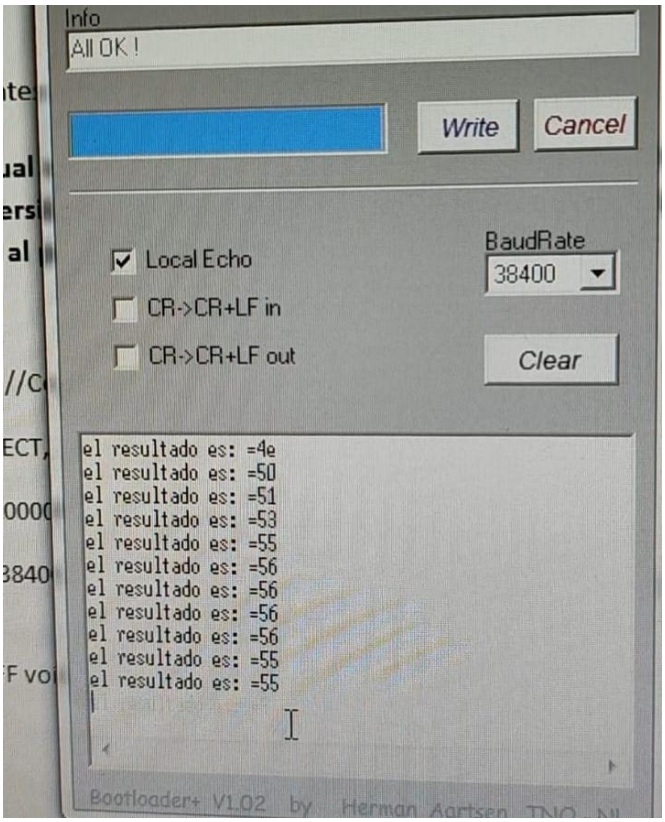


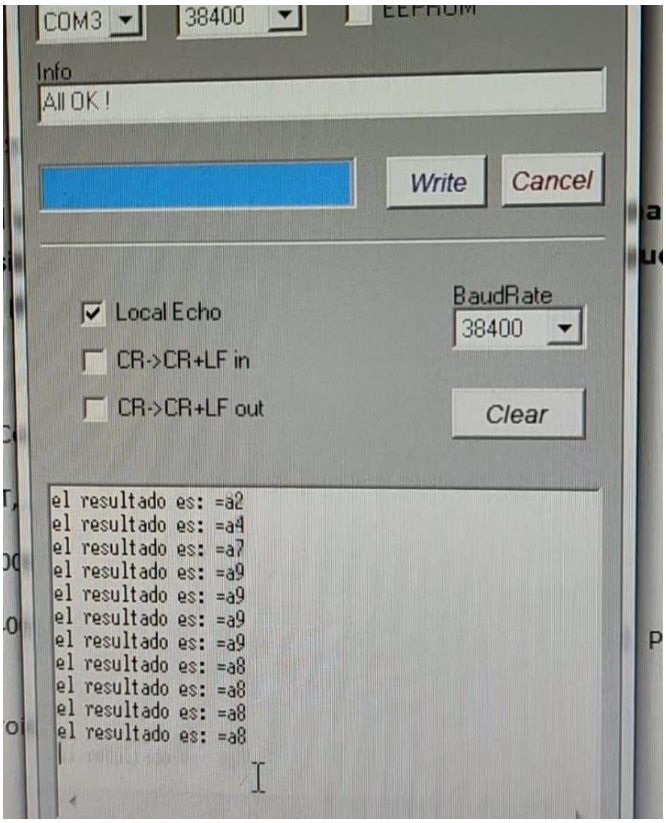
Hernández Diaz Sebastian





Hernández Diaz Sebastian





Por último, para poder utilizar este código en lenguaje ensamblador se tiene que tener en cuenta lo siguiente:

setup_port_a(ALL_ANALOG), se debe escribir 0x00 en el registro ADCON1, que indica que todos los pines RA0–RA5 serán entradas analógicas.

Se debe acceder a banco 1 para escribir en ADCON1.

setup_adc(ADC_CLOCK_INTERNAL), en el registro ADCON0, se deben configurar los bits ADCS1:ADCS0 en 11 para reloj interno.

Además, se debe habilitar el módulo ADC (ADON = 1 en el bit 0 de ADCON0).

set_adc_channel(5), configurar bits CHS2:CHS0 de ADCON0 con 101 (para seleccionar AN5).

Esto también se hace en ADCON0, asegurándose de que ADON sigue activado.

Configuración de TRISB

Cambiar a banco 1 (BSF STATUS, RP0)

Escribir TRISB = 0x00 para que todos los pines sean salidas

delay_ms(20), una rutina de retardo basada en bucles decfsz, calibrada para 20 ms según una frecuencia de reloj de 20 MHz.

2.- Utilizando la interrupción del TIMER0, realizar un programa que transmita el resultado de la conversión cada 10 segundos.

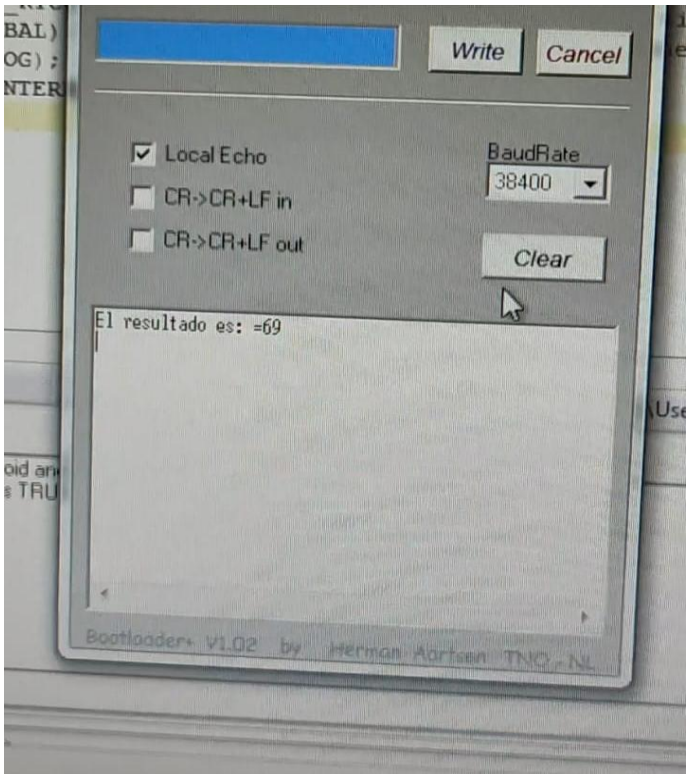
El código usado para esta actividad fue el siguiente:

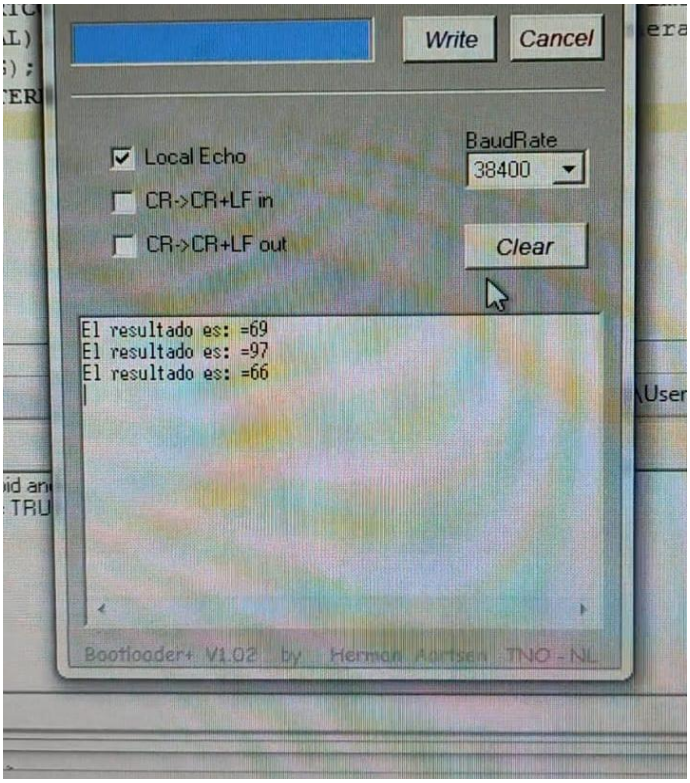
```
#include <16f877.h>                              // Incluye el archivo de cabecera para el PIC16F877
#device ADC = 8,                                 // Configura el ADC con resolución de 8 bits
#fuses HS, NOPROTECT,                            // Configura los fusibles: HS para oscilador de alta velocidad, NOPROTECT desactiva protección de lectura
#use delay(clock = 20000000)                     // Configura el reloj a 20 MHz
#use rs232(baud = 38400, xmit = PIN_C6, rcv = PIN_C7) // Configura la comunicación RS232 a 38400 bps, usando los pines C6 para transmisión y C7 para recepción
#org 0x1F00, 0x1FFF void loader16F877(void){}     // Define la región de memoria para la función loader16F877 (para el PIC16F877)

int conv;      // Variable para almacenar la conversión del ADC
long cont = 0; // Contador que se incrementará para activar la interrupción cada 10 segundos
```

```c
// Función de interrupción
#int_RTCC
clock_isr()
{
   cont++; // Incrementa el contador en cada interrupción
   // Verifica si han pasado 10 segundos (763 ciclos del reloj)
   if (cont == 763)
   {
      printf("El resultado es: =%x\n", conv); // Muestra el valor de la conversión en formato hexadecimal
      cont = 0;                               // Resetea el contador
   }
}

void main()
{
   set_timer0(0);                             // Inicializa el temporizador 0 a cero
   setup_counters(RTCC_INTERNAL, RTCC_DIV_256); // Configura el temporizador 0 con fuente de reloj interna y prescaler de 256
   enable_interrupts(INT_RTCC);               // Habilita la interrupción por el temporizador 0
   enable_interrupts(GLOBAL);                 // Habilita las interrupciones globales
   setup_port_a(ALL_ANALOG);                  // Configura todos los pines del puerto A como entradas analógicas
   setup_adc(ADC_CLOCK_INTERNAL);             // Configura el ADC con reloj interno
   set_adc_channel(5);                        // Selecciona el canal 5 del ADC para leer la entrada analógica
   while (1)                                  // Bucle infinito
   {
      delay_us(20);      // Espera 20 microsegundos para estabilizar la señal
      conv = read_adc(); // Lee el valor analógico del canal seleccionado y lo almacena en 'conv'
   }
}
```

Este código fue diseñado para configurar un PIC16F877 con un ADC de 8 bits, de manera que lea un valor analógico del canal 5 de manera continua, cada 20 microsegundos. Utiliza el temporizador 0 para generar una interrupción cada 10 segundos, mediante un contador que se incrementa en cada interrupción. Al alcanzar un valor específico, el sistema transmite el valor leído del ADC a través de comunicación RS232 en formato hexadecimal. La configuración del microcontrolador incluye un reloj de 20 MHz, una fuente interna para el ADC y una transmisión serial a 38400 bps, permitiendo monitorear el valor analógico de forma periódica y remota.

La ejecución del código es la siguiente:





Por último, para poder utilizar este código en lenguaje ensamblador se tiene que tener en cuenta lo siguiente:

setup_port_a(ALL_ANALOG), configurar el registro ADCON1 a 0000 (todos los canales como analógicos).

setup_adc(ADC_CLOCK_INTERNAL), en el registro ADCON0, poner ADCS1:ADCS0 = 11 y habilitar el ADC (ADON = 1).

También debe configurarse el canal de entrada, ver siguiente punto.

set_adc_channel(5), En ADCON0, poner CHS2:CHS0 = 101 (para canal 5).

Asegurarse de mantener ADON = 1.

output_b(conv), Asegurarse de que PORTB es salida (TRISB = 0x00 en banco 1).

Copiar el valor de ADRESH a PORTB.

delay_us(20), Implementar una rutina de retardo muy corto usando un par de instrucciones nop o un bucle mínimo de decfsz.

3.- Realizar un programa el cual constantemente transmita el resultado de la conversión a la terminal, y cada 30 segundos interrumpa la ejecución de este y envíe el siguiente texto “Laboratorio de Microcomputadoras”.

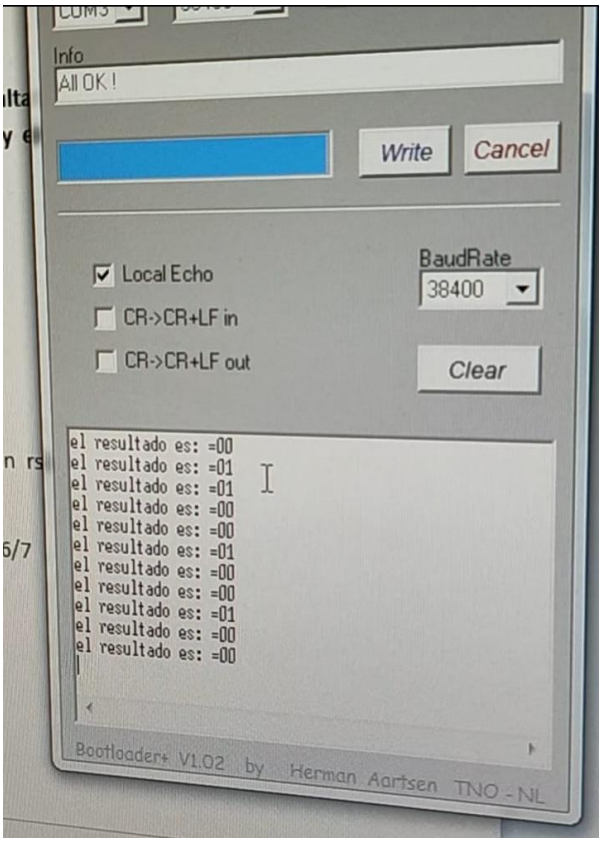
El código usado para esta actividad fue el siguiente:

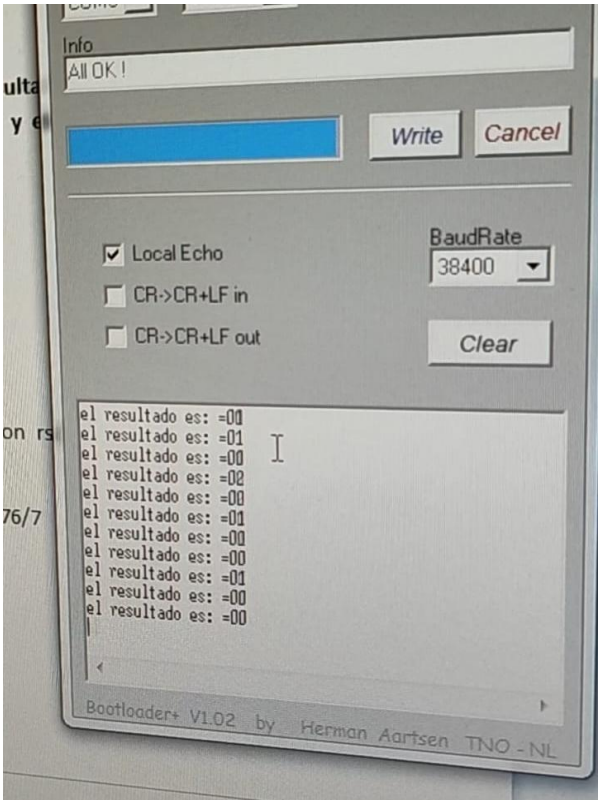
```
#include <16f877.h>                              // Incluye el archivo de cabecera para el PIC16F877
#device ADC = 8,                                 // Configura el ADC con una resolución de 8 bits
#fuses HS, NOPROTECT,                            // Configura los fusibles: HS para un oscilador de alta velocidad
#use delay(clock = 20000000)                     // Configura el reloj del microcontrolador a 20 MHz
#use rs232(baud = 38400, xmit = PIN_C6, rcv = PIN_C7) // Configura la comunicación RS232 con una tasa de 38400 bps, usando los pines C6 para transmisión y C7 para recepción
#org 0x1F00, 0x1FFF void loader16F877(void){}     // Define la región de memoria para la función loader16F877 (para el PIC16F877)

int conv;      // Variable para almacenar el valor de la conversión del ADC
long cont = 0; // Contador que se incrementará para activar la interrupción cada 30 segundos
```

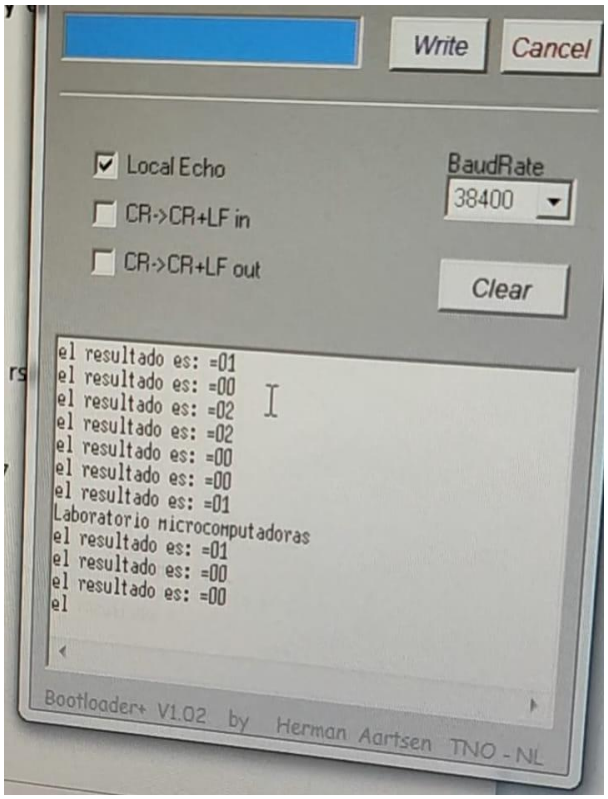
```c
// Función de interrupción por el temporizador
#int_RTCC
clock_isr()
{
   cont++; // Incrementa el contador cada vez que se activa la interrupción
   // Verifica si han pasado 30 segundos (769*3 ciclos del reloj)
   if (cont == (769 * 3))
   {
      printf("Laboratorio microcomputadoras\n"); // Imprime un mensaje en la terminal
      cont = 0;                                  // Resetea el contador para el siguiente ciclo
   }
}

void main()
{
   set_timer0(0);                                // Inicializa el temporizador 0 a cero
   setup_counters(RTCC_INTERNAL, RTCC_DIV_256); // Configura el temporizador 0 con reloj interno y prescaler de 256
   enable_interrupts(INT_RTCC);                  // Habilita la interrupción por el temporizador 0
   enable_interrupts(GLOBAL);                    // Habilita las interrupciones globales
   setup_port_a(ALL_ANALOG);                     // Configura todos los pines del puerto A como entradas analógicas
   setup_adc(ADC_CLOCK_INTERNAL);                // Configura el ADC con reloj interno
   set_adc_channel(2);                           // Selecciona el canal 2 del ADC para leer la entrada analógica
   while (1)
   {                                            // Bucle infinito
      delay_ms(20);                             // Espera 20 milisegundos
      conv = read_adc();                        // Lee el valor del ADC y lo almacena en la variable 'conv'
      printf("el resultado es: =%x\n", conv); // Imprime el valor leído del ADC en formato hexadecimal
      output_b(conv);                           // Envía el valor leído al puerto B
   }
}
```

Este código configura el PIC16F877 para realizar lecturas periódicas del canal 2 del ADC, cada 20 milisegundos. El valor leído del ADC se almacena en la variable conv, y se muestra en la terminal en formato hexadecimal a través de comunicación RS232. Además, el código usa un temporizador (con interrupciones) que, cada 30 segundos, imprime un mensaje en la terminal. El valor del ADC también se envía al puerto B del microcontrolador, lo que permite visualizarlo externamente o conectarlo a otros dispositivos. Las interrupciones por el temporizador se manejan en la función clock_isr(), y el contador se restablece después de cada 30 segundos.

La ejecución del código es la siguiente:







Por último, para poder utilizar este código en lenguaje ensamblador se tiene que tener en cuenta lo siguiente:

Configuración del ADC (Convertidor Analógico Digital)

Se debe configurar el módulo ADC para que todos los pines del puerto A estén en modo analógico. Esto se hace manipulando los registros ADCON1 y ADCON0. Además, se debe seleccionar el canal 2 como entrada analógica, habilitar el ADC y definir que utilice el reloj interno como fuente de tiempo para las conversiones.

Configuración del UART (Comunicación Serial RS232)

Se debe establecer la configuración del puerto serial:

- Activar el transmisor y receptor.
- Configurar la velocidad de transmisión a 38400 baudios usando el registro SPBRG.
- Habilitar los bits correspondientes en los registros TXSTA y RCSTA.
- Usar los pines RC6 (TX) y RC7 (RX) como líneas de transmisión y recepción.

Manejo de interrupciones

Se necesita una rutina de interrupción que se ejecute cada vez que el temporizador se desborda. En esta rutina:

- Se debe incrementar un contador cada vez que se genera la interrupción.
- Cuando el contador alcance un valor específico (aproximadamente equivalente a 30 segundos), se debe enviar un mensaje por el puerto serial y luego reiniciar el contador.

Retardo de 20 ms entre conversiones

Es necesario implementar un retardo entre lecturas del ADC. Esto puede hacerse con un ciclo de bucle que consuma el tiempo equivalente a 20 milisegundos, considerando que el sistema opera a 20 MHz.

4.- Utilizando la interrupción por cambio de nivel del puerto paralelo, realizar un programa que reconozca un flanco positivo en los pines PB4, PB5, PB6 o PB7 del puerto B, y cuando se presente, envíe a la terminal el siguiente texto; de acuerdo a la entrada en la que ha ocurrido el evento.

Interrupción PB4 Activada

Interrupción PB5 Activada

Interrupción PB6 Activada

Interrupción PB7 Activada

Cuando se detecte la transición de alto a bajo, se debe mostrar:

Pulso de bajada

El código usado para esta actividad fue el siguiente:

```
#include <16f877.h>                          // Incluye la cabecera del microcontrolador PIC16F877
#device ADC = 8                              // Configura el convertidor analógico-digital (ADC) con una resolución de 8 bits
#fuses HS, NOPROTECT                         // Usa oscilador de alta velocidad y desactiva la protección del código
#use delay(clock = 20000000)                 // Establece la frecuencia del reloj del sistema a 20 MHz
#use rs232(baud = 38400, xmit = PIN_C6, rcv = PIN_C7) // Configura la comunicación serial (RS232) a 38400 baudios
#org 0x1F00, 0x1FFF void loader16F877(void){} // Reserva espacio de memoria para el bootloader (del 0x1F00 al 0x1FFF)
```

```c
int var1; // Variable entera para almacenar el valor del puerto B

// ---------- INTERRUPCIÓN ----------
#int_rb // Define la rutina de interrupción para cambio de estado en pines RB4-RB7
int_p() // Función que se ejecuta cuando hay cambio de nivel lógico en RB4-RB7
{
    if (input(pin_b4))            // Si RB4 está en alto
        printf("\nPB4 activado"); // Imprime mensaje de que RB4 fue activado
    else
        printf("\nPB4 desactivado"); // Si está en bajo, imprime desactivado

    if (input(pin_b5))
        printf("\nPB5 activado");
    else
        printf("\nPB5 desactivado");

    if (input(pin_b6))
        printf("\nPB6 activado");
    else
        printf("\nPB6 desactivado");

    if (input(pin_b7))
        printf("\nPB7 activado");
    else
        printf("\nPB7 desactivado");

    printf("\n\n-------------------------------\n\n");
}
```

```c
void main()
{
    setup_counters(RTCC_INTERNAL, RTCC_DIV_256); // Configura el Timer0 como contador con reloj interno y divisor 1:256
    enable_interrupts(INT_RB);                   // Habilita la interrupción por cambio en pines RB4-RB7
    enable_interrupts(GLOBAL);                   // Habilita las interrupciones globales

    while (1)
    {
        var1 = input_b(); // Lee el valor actual de todo el puerto B
        output_a(var1);   // Envía ese valor leído directamente al puerto A
    }
}
```

Este programa permite monitorear los pines RB4 a RB7 del PIC16F877. Cada vez que uno de esos pines cambia de estado (de bajo a alto o viceversa), se genera una interrupción, y el microcontrolador envía por puerto serial (RS232) un mensaje indicando qué pin se activó o desactivó. Además, en el bucle principal, el valor completo del puerto B se copia continuamente al puerto A, lo cual puede usarse para reflejar en tiempo real los cambios de entrada del puerto B en salidas del puerto A (por ejemplo, para pruebas con LEDs o para depuración visual).

Por último, para poder utilizar este código en lenguaje ensamblador se tiene que tener en cuenta lo siguiente:

Configurar la interrupción por cambio en PORTB (RB4–RB7)

- Configurar el puerto B como entrada (al menos los pines RB4 a RB7).
- Activar la interrupción por cambio de nivel en los pines RB4 a RB7, lo que implica:
    - Habilitar RBIE en el registro INTCON.
    - Habilitar las interrupciones globales (GIE).
    - Asegurarse de que PORTB esté correctamente inicializado para que las transiciones se detecten.

Escribir la rutina de interrupción INT_RB

- Leer el estado de los pines RB4, RB5, RB6 y RB7 usando PORTB.
- Comparar cada bit individual para verificar si está en alto o bajo.
- Enviar un mensaje por el puerto serial (RS232) dependiendo del estado de cada pin.
- Restablecer el flag de interrupción por cambio en PORTB (RBIF en INTCON).

Ciclo principal del programa (loop infinito)

En el bucle principal del código:

- Leer continuamente el valor de PORTB (todos los 8 bits).
- Escribir ese mismo valor en PORTA.
  Esto requiere configurar el puerto A como salida y el puerto B como entrada en los registros TRISA y TRISB.

Manejo de interrupciones

- Guardar y restaurar el contexto del microcontrolador dentro de la ISR (es decir, guardar W, STATUS y PCLATH si son modificados).
- Asegurarse de restablecer los flags de interrupción (como RBIF) antes de salir de la ISR.
- Utilizar RETFIE para regresar correctamente de la interrupción.

Hernández Diaz Sebastian

Conclusiones

Durante el desarrollo de estos ejercicios se aplicaron diversos conceptos fundamentales en la programación de microcontroladores PIC, combinando manejo de puertos analógicos y digitales, configuración y uso del convertidor analógico-digital (ADC), transmisión de datos vía RS232, así como la implementación de interrupciones tanto por temporizador como por cambio de nivel en pines del puerto B.

En el primer ejercicio, se comprendió cómo configurar el módulo ADC del PIC16F877 para leer señales analógicas desde un canal específico, y se aprendió a convertir esta señal en un valor digital con resolución de 8 bits. Posteriormente, se mostró este valor simultáneamente en un puerto paralelo y se transmitió a través del puerto serial, reforzando el uso de funciones de entrada/salida como input_b()/output_d() y el uso del módulo rs232.

El segundo ejercicio introdujo el uso de interrupciones por temporizador (TIMER0), lo cual fue esencial para generar eventos periódicos sin bloquear el flujo principal del programa. Se configuró correctamente el prescaler y se utilizó el vector #int_RTCC para ejecutar una función cada 10 segundos que transmitiera el resultado del ADC. Esto permitió entender cómo trabajar con eventos temporizados y cómo mantener la precisión en tiempo real.

En el tercer ejercicio, se amplió la lógica anterior para interrumpir la transmisión continua de los datos del ADC y, cada 30 segundos, enviar un mensaje específico a la terminal serial (“Laboratorio de Microcomputadoras”). Esta actividad profundizó en el uso combinado del ADC, temporizadores e interrupciones globales, demostrando cómo se pueden coordinar múltiples tareas en un solo programa de manera eficiente.

Finalmente, el cuarto ejercicio abordó la interrupción por cambio de nivel en los pines RB4 a RB7 del puerto B. Se logró detectar flancos de subida y bajada en estos pines, mostrando mensajes adecuados en la terminal dependiendo de la transición detectada. Este ejercicio consolidó el conocimiento sobre el manejo de entradas digitales con interrupciones, una técnica ampliamente utilizada en sistemas embebidos para reducir el consumo de recursos y aumentar la capacidad de respuesta del sistema ante eventos externos.

Bibliografía

Microchip Technology Inc. (2001). *PIC16F87X Data Sheet: 28/40-Pin Enhanced Flash Microcontrollers*. Retrieved from https://ww1.microchip.com/downloads/en/DeviceDoc/30292c.pdf

Barnett, R., Cox, S., & O'Cull, L. (2003). *Embedded C Programming and the Atmel AVR*. Delmar Cengage Learning.

Ibrahim, D. (2013). *PIC Microcontroller Projects in C: Basic to Advanced*. Newnes/Elsevier.

Valdés-Pérez, R. (2008). *Sistemas digitales con microcontroladores PIC: Fundamentos y aplicaciones con MPLAB C*. Alfaomega Grupo Editor.

Dogan Ibrahim. (2014). *Advanced PIC Microcontroller Projects in C: From USB to RTOS with the PIC18F Series*. Newnes.

Skvarcius, R. (2002). *Using the PIC16F877 Microcontroller: Theory and Applications*. Prentice Hall.

Microchip Technology Inc. (n.d.). *MPLAB® X IDE User's Guide*. Retrieved from https://www.microchip.com/en-us/tools-resources/develop/mplab-x-ide