Práctica 2
Lenguaje Ensamblador, Interrupciones y Temporizadores

Stephanie Lizeth Malvaes Diaz

Daniela Arely Morales Hernández
Ingeniería en Telecomunicaciones
Instituto Tecnológico Autónomo de México
Email: daniela.morales@itam.mx

***Resumen*—Se analizará las diferencias entre el uso de lenguaje ensamblador y programación en arduino. Será muy importante entender las arquitecturas de computadoras y los sistemas en tiempo real así cómo entender el uso y configuración de los temporizadores y los sistemas de interrupciones.**

I. Introducción

I-A. Lenguaje ensamblador

El lenguaje ensamblador es un sistema de codificación de instrucciones de bajo nivel que usa registros y localidades de memoria de los microprocesadores. Es usado para tener tiempos de ejecución de muy alta exactitud dado que usa ciclos de reloj para ejecutar las instrucciones, por ello es usado en aplicaciones de tiempo real dado que se necesita garantizar la ocurrencia de eventos en momentos determinados.

I-B. Interrupciones

El uso de interrupciones es atender tareas momentáneamente durante el flujo principal del programa es una utilidad necesaria en los sistemas en tiempo real. Los dos tipos principales de interrupciones son aquellos basados en temporizadores y las controladas por estímulos externos.

I-C. Temporizadores

Muchas aplicaciones necesitan contar un evento o generar tiempos de retardo para su funcionalidad. Para esto, utilizamos un reloj, interno o externo, si es interno la frecuencia del oscilador es alimentada al timer, si es externo se necesitará alimentar pulsos a través de uno de los pines del AVR.

II. Marco Teórico

En la práctica utilizamos el conjunto reducido de instrucciones del microcontrolador ATMEGA. El software de Arduino es un IDE (entorno de desarrollo integrado), es decir, una aplicación informática que proporciona servicios para facilitar al programador el desarrollo de software y cargar el programa compilado en la memoria flash del hardware. Consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI), pues suele ofrecer muchas características para la creación, modificación, compilación, implementación y depuración de software. No hay un lenguaje propio de programación de Arduino,la programación se hace en C++ pero Arduino ofrece librerías ya incluidas en el IDE que facilitan la programación de los pines de entrada y salida y de los puertos de comunicación, así como otras librerías para operaciones específicas. Otra diferencia con C++ es la estructuctura del programa ya que en lugar de la función `main()`, usa las funciones `setup()` y `loop()`. La programación en arduino puede dividirse en tres partes principales: funciones, valores (variables y constantes) y estructura.

- Funciones
  - Digital I/O
  - Analog I/O
  - Zero, Due, Mkr Family
  - Advanced I/O
  - Time
  - Math
  - Trigonometry
  - Characters
  - Random numbers
  - Bits and bytes
  - External Interrupts
  - Interrupts
  - Communication
  - USB
- Variables
  - Constants
  - Conversion
  - Data Types
  - Variable Scope and Qualifiers
  - Utilities
- Estructura
  - Sketch (Setup y loop)
  - Control structure
  - Further syntax
  - Arithmetic operators
  - comparison operators
  - boolean operators
  - pointer access operators
  - bitwise operators
  - compound operators

Tenemos variables globales y locales: Una variable global puede ser vista y utilizada por cualquier función y estamento de un programa. Esta variable se declara al comienzo del programa, antes de `setup()`. Ocupa un espacio de memoria permanente por lo que supone uso ineficiente de memoria. La variable local se define dentro de una función o como parte de un bucle. Sólo es visible y sólo puede utilizarse dentro de la función en la que se declaró. El modificador de variable static,

es utilizado para crear variables que solo son visibles dentro de una función, sin embargo, al contrario que las variables locales que se crean y destruyen cada vez que se llama a la función, las variables estáticas mantienen sus valores entre las llamadas a las funciones.

II-A. Lenguaje ensamblador

Consiste en una serie de mnemónicos los cuales representan instrucciones básicas. Representan los códigos máquina binarios para poder programar una arquitectura de procesador. Entre los tipos de instrucciones que hay encontramos

1. Instrucciones aritméticas y lógicas
2. Instrucciones de desvío (salto)
3. Instrucciones de transferencia de datos
4. Instrucciones de bit y prueba de bit

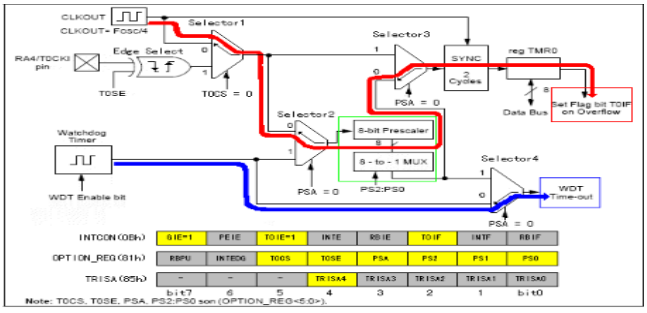
II-B. Interrupciones

Es el rompimiento en la secuencia de un programa para ejecutar una rutina de servicios que al finalizar regresa al punto de donde se interrumpió.

- Interrupciones por software: Las programadas por el usuario que son generalmente usada para realizar tareas de entrada y salida
- Interrupciones por hardware: provocadas por dispositivos externos al procesador, pueden ocurrir en cualquier momento Cuando se genera una interrupción el CPU termina de ejecutar la instrucción en curso y evalúa la fuente de la interrupción

II-C. Temporizadores

Muchas aplicaciones necesitan contar un evento o generar tiempos de retardo para su funcionalidad. Para esto, utilizamos un reloj, interno o externo, si es interno la frecuencia del oscilador es alimentada al timer, si es externo se necesitará alimentar pulsos a través de uno de los pines del AVR.



El Preescaler que, está en el centro de la figura, puede ser usado por el TMR0 o por el WDT. La figura anterior muestra el preescaler conectado a TMR0. El bit PSA (bit 3) del `OPTION_REG` determina a cuál de los dos es conectado el preescaler. El preescaler es un contador programable cuyo rango es determinado por los bits PS0, PS1, PS2 (bits 0, 1 y 2) de `OPTION_REG`. TMR0 es un contador binario de 8 bit que puede contar hasta 256. Cuando el contador rebasa la cuenta de 255 (FFh) a 0 (00h) ocurre una interrupción por desbordamiento y el bit T0IF (bit 2) del registro INTCON es puesto a 1. El hardware está diseñado tal que cuando ambos el GIE (bit 7) y TOIE (bit 5) del registro INTCON son H ("1") la interrupción ocurre y el PC (program counter) va la dirección 004h, para comenzar la operación de programa.

III. Desarrollo

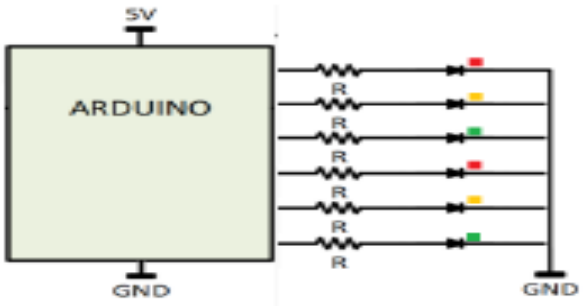
III-A. Ensamblador

En la primera parte de la práctica se realizó un código en arduino que hiciera parpadear 4 leds en diferentes frecuencias. Ese mismo funcionamiento lo tuvimos que implementar en lenguaje ensamblador. Para realizar los cambios en arduino tuvimos que usar la función `delay` para lograr los diferentes intervalos de parpadeo. A continuación se muestran ambos programas:

El código de esta sección se encuentra en el repositorio como P2b

También tuvimos que generar una rutina que usara localidades de memoria y registros durante un segundo.

Por último tuvimos que programar el funcionamiento de un semáforo con diferentes leds basados en el siguiente diagrama.



El código de esta sección se encuentra en el repositorio como P2d

III-B. Interrupciones

Se programo una interrupción que hiciera parpadear un led a una frecuencia de 1 Hz. El código para programar una interrupción es el siguiente el cual va integrado a la parte de la sección de `void` dentro del arduino.

```
void setup()
{
cli();
DDRD &=~(1 << DDD1);
PORTD |= (1 << PORTD1);
EICRA |= (1 << ISC10);
EIMSK |= (1 << INT1);
sei();
}
ISR(INT1_vect)
{
...//Códigoaejecutaralactivarselainterrupción
}
```

»Se tuvo que agregar un push-button que funcionara como contador cada que fuera pulsado sin interferir con la interrupción anterior.

»Una vez que esto funcionará se remplazo el push-button con un lector óptico. Se tuvo que cambiar el código para cambiar al recibir cierta intensidad de luz para poder percibir el cambio.

El código de esta sección se encuentra en el repositorio como P2e

»*III-C. Temporizadores*

»Para habilitar un temporizador se deben habilitar primero las interrupciones vigentes. Configurar los registros de control y establecer el preescalador. Todo lo anterior se realizar al insertar el siguiente código para activar el temporizador en modo normal.

```
void setup()
{
cli();
TCCR1B = 0; TCCR1A = 0;
TCCR1B || = (1 << CS12);
TCNT1 = 3036;
TIMSK1|| = (1 << TOIE1);
sei();
}
ISR(TIMER1_OVF_vect)
{
...//Códigoaejecutaralactivarselainterrupción
}
```

»Para acabar esta practica tuvimos que implementar el semáforo anterior pero ahora programado con interrupciones. En esta parte del código hicimos un contador de 1 a 24, cuando el contador va de 0 a 11 se enciende solamente el LED verde, cuando el contador va de 12 a 14 se enciende además el amarillo, cuando el contador va de 15 a 23 se apagan los LEDs verde y amarillo y se enciende el rojo, finalmente, cuando el contador llega a 24, éste se reinicia.

El código de esta sección se encuentra en el repositorio como P2r

»IV. Conclusiones

»Durante la práctica el mayor reto resultó ser entender el conceptos de las interrupciones y de los temporizadores. Fue un poco complejo entender la forma en que se deben programar y usar sus características. Posterior a eso lo más complicado fueron los distintos circuitos que tuvimos que armar durante la práctica ya que cada inciso nos pedía una modificación al circuito.

»Referencias

[1] »https://www.arduino.cc/reference/en/
[2] »https://aprendiendoarduino.wordpress.com/tag/avr/
[3] »https://aprendiendoarduino.wordpress.com/2016/11/08/entradas-y-salidas-arduino/
[4] »https://jaimedearcos.github.io/arduino/2017/01/04/Timers/
[5] »https://www.arduinohobby.com/interrupciones-timer-arduino/
[6] »https://aprendiendoarduino.wordpress.com/2016/11/13/interrupciones/