Ensamblador, Interrupciones & Temporizadores

Daniela Arely Morales Hernández
Ana Carolina Sandoval Mejía 152808
Stephanie Lizeth Malvaes Diaz 135515

I. Introducción

Conocer y manejar los conceptos básicos del comportamiento del micro-controlador es vital para comprender el desarrollo de este laboratorio. Es por esto que en esta práctica pusimos énfasis en recordar el conocimiento adquirido en clases anteriores como lo son: acceso y declaración de pines, comandos del ensamblador, acceso del micro-controlador a través del ensamblador, entre otros. Estos conceptos son presentados en el apartado de “Marco Teórico”. En un principio, nos familiarizamos con el lenguaje de ensamblador para resolver algunos problemas de variables, luego asignamos los pines de lectura y de escritura para cambiar el estado del led usando interrupciones, además de imprimir los estados en la consola.para finalizar, diseñamos un arreglo de leds para representar una sucesión de 4 bits. Posteriormente programamos el micro-controlador para que los leds parpadearan cada 10 ms empleando el timer de 8 bits, así como puertos tipo PWM. Para lo cual, fue fundamental calcular el periodo y el contador del mismo.

II. Conceptos.

Ensamblador

El lenguaje ensamblador es el lenguaje de programación utilizado para escribir programas informáticos de bajo nivel, que al ser programados sobre hardware suelen ser más rápidos y ocupan menos recursos.
Es un código estructurado desarrollado sobre un archivo de programación (.ASM)
Son una serie de instrucciones ejecutables que son cargadas en la memoria de un sistema en código de operación, que suelen tener una correspondencia de 1 a 1 entre las instrucciones simples.

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

Temporizadores

Muchas aplicaciones necesitan contar un evento o generar tiempos de retardo para su funcionalidad. Para ésto, 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.

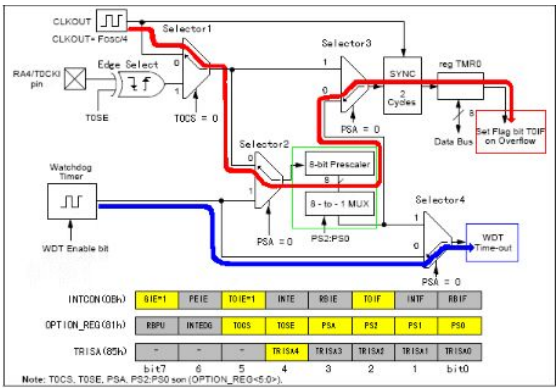

Fig 1. Diagrama del temporizador.

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

Ensamblador

Se programó en lenguaje C tres problemas, sacar el discriminante de tres variables, el promedio de 10 números y controlar el encendido u apagado de un LED.

Posteriormente se realizó las mismas tareas en lenguaje ensamblador y ver el AVR generado.

Interrupciones

Sensor infrarrojo: Se conectó en el arduino en lugar del botón usado anteriormente, un led infrarrojo y un fotodiodo puestos frente a frente, y conectado el led a 5 V del lado positivo y a una resistencia de 330 ohms conectada a tierra, del lado negativo, y el fotodiodo a 5V de lado positivo y a una resistencia de 10k, conectada a tierra, de lado negativo, con el mismo código usado con el botón.
Este tenía implementado un contador, que aumentaba cada vez que se ponía algo entre el led y el diodo.

Temporizadores

Modo normal

En la práctica primero se nos pidió que hiciéramos parpadear un LED usando el modo normal a 2 Hz. En este modo el contenido del timer/contador aumenta cada ciclo de
reloj. El contador se incrementa hata alcanzar su máximo 0xFF, cuando
él pasa de 0xFF a 0x00, la bandera TVO0 se activa. Esta bandera se
puede monitorear. A continuación se muestran fragmentos del código y sus explicaciones.

```
/*
int flag=LOW;
void setup() {
  Serial.begin(9600);
  DDRB = DDRB | B10000000;
  cli();
  TCCR1B= 0;
  TCCR1A=0;
  TCCR1B |= (1 << CS12);
  TCNT1= 3036;
```

```
  TIMSK1 |= (1 << TOIE1);
  sei();

}
*/
```

En esta parte del código se configura el modo normal haciendo TCCR1A y TCCR1B iguales a cero, TCNT1 se hace igual a 3036 para que el temporizador cuente a 2Hz y TIMSK1 se hace de esa manera para habilitar la interrupción de desbordamiento del temporizador.

```
/*
void loop() {
  // put your main code here, to run
repeatedly:


}
ISR(TIMER1_OVF_vect){

  digitalWrite(13, flag);
  flag=!flag;
}
/*
```

ISR funciona simplemente para prender o apagar el LED cada 2 Hz.

Modo CTC

Al igual que en el modo normal, en el modo CTC, el timer es incrementado con un reloj. Pero cuenta hasta que el contenido del registro TCNT0 llegue a ser igual al contenido de OCR0. En esta parte se nos pide que hagamos parpadear el LED a 4 Hz con el modo CTC.

```
/*
int flag=LOW;
void setup() {
  Serial.begin(9600);
  DDRB = DDRB | B10000000;
  cli();
  TCCR1B= 0;
  TCCR1A=0;
  TCCR1B |= B00001101;
  OCR1A= 0x0F42;
  TIFR1 |= (1 << OCF1A);
  TIMSK1 |= (1 << OCIE1A);
  sei();

}

void loop() {
}

ISR(TIMER1_COMPA_vect){
  digitalWrite(13,flag);
  flag=!flag;
}
/*
```

TCCR1A y TCCR1B se utilizan para configurar el modo CTC. OCR1A con valor 0x0F42 se utiliza para configurar el timer a 4 Hz.

Semáforo

En esta parte se nos pide que implementemos un semáforo con ciertas características.

```
/*
int counter=0;
void setup() {
  Serial.begin(9600);
  DDRB = DDRB | B10100100;

  cli();
  TCCR1B= 0; TCCR1A=0;
  TCCR1B |= B00001101;
  OCR1A= 0x3D09;
  TIMSK1 |= (1 << OCIE1A);
  sei();
}
*/
```

En esta parte configuramos el temporizador.

```
/*
void loop() {
}

ISR(TIMER1_COMPA_vect){
  Serial.println("ISR triggered");
  Serial.println(counter);

  if(counter==0){
        digitalWrite(2, HIGH); //verde
        digitalWrite(5, LOW); //amarillo
        digitalWrite(7, LOW); //rojo
  }else if(counter==12){
        digitalWrite(5, HIGH);
  }else if(counter== 15){
        digitalWrite(2, LOW); //verde
        digitalWrite(5, LOW);
        digitalWrite(7, HIGH);
  }else if(counter==24){
        counter=-1;
  }
counter= counter+1;
}
/*
```

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.

IV. Resultados

Ensamblador

En esta parte no pudimos comparar hacer las comparaciones de los AVR debido a que nuestro equipo no tenía habilitada esta opción.

Interrupciones

En la primera parte de este inciso, se implementó el código y la conexión si problema alguno, el botón funciona como contador, aunque al apretarlo algunas veces en lugar de contar de uno en uno, se saltó algún número. En la segunda parte, hubo un problema con la alineación del led infrarrojo y el fotodiodo: Funcionó el contador, sin embargo, era fácil que se desalinearán al poner algo en medio y entonces paraba el contador.
Limitaciones:
Si tienes un proceso, como un while, que demora mucho el contador no aumentará pues el microprocesador no puede hacer varias funciones a la vez.
Otra limitación es que si tenemos un delay muy grande, el contador tampoco aumentará al presionar el pushbutton pues esperará hasta que acabe el delay.

Temporizador

En los tres incisos pudimos hacer que los LEDs se prendieran con las especificaciones de la práctica correctamente.

Limitaciones:
La frecuencia del temporizador depende de la frecuencia del oscilador, además, la primera vez es un poco complicado configurar los temporizadores y los retardos pues se deben conocer los pines que hay que modificar y se necesitan calcular ciertas fórmulas para los retardos.

V. Conclusiones

Stephanie Malvaes: En esta práctica fue más claro el uso del lenguaje ensamblador, y de c++, el uso del

arduino, y de los DDR”X”, para configurar entradas y salidas, las conexiones, asignación de valores a puertos. Además cosas nuevas como interrupciones, temporizadores, uso del led infrarrojo y fotodiodo como sustituto de un botón que activa un contador externo. fue una práctica interesante y aclaró los temas vistos en teoría.

Daniela Morales: Fue una práctica bastante extensa debido a que armar los diferentes dispositivos que conectamos y manejar el lenguaje requerido fue una tarea bastante complicada. Aprender el manejo de las interrupciones y sus diferencias con los comando que vienen predeterminadas como attached Interrupted, lo cual averiguamos que internamente hace el mismo procedimiento que nosotros desarrollamos

Ana Carolina Sandoval Mejía: Esta práctica es importante para entender las diferencias entre ensamblador y C++, además nos ayuda a entender cómo funcionan y cómo se pueden configurar los relojes en los temporizadores, incluyendo timers y retardos. Esta fue una práctica larga y complicada pues tuvimos que trabajar con interrupciones antes de ver el tema en clase, además los temporizadores me resultaron algo difíciles de entender y configurar. Otro problema que tuvimos fue que al principio cuando no nos funcionó la interrupción con el LED infrarrojo no sabíamos si esto era debido a que el LED infrarrojo no prendía (su luz no es visible) o debido a que el fotodiodo no estaba en línea de vista con la luz emitida del LED.

VI. Referencias

http://bbeltran.cs.buap.mx/Interrupciones.pdf
https://hispavila.com/total/3ds/chipspic/tmr0.html