Laboratorio 2: GPIOs, Timers y FSM

1st Ronny Granados Pérez
Escuela de Ingeniería Eléctrica
Universidad de Costa Rica.
San José, Costa Rica
C03505
ronny.granadosperez@ucr.ac.cr

I. Introducción/Resumen

Este laboratorio fue realizado con el microcontrolador ATTiny4313 y el objetivo era construir un semáforo peatonal que se basara en las distintas interrupciones que ofrece este dispositivo. Para su elaboración se utilizó lenguaje C y dentro de este código se especificaron las interrupciones y se construyó una máquina de estados que modelara las distintas acciones que debía hacer el semáforo. La implementación en hardware se realizó mediante LEDs color rojo y verde pues por efectos de simplificación del modelo solamente se utilizaron las luces roja y verde del semáforo vehicular.

II. Nota Teórica

II-A. ATTiny4313

Este dispositivo con arquitectura AVR de 8 bits de bajo consumo, cuenta con 18 pines I/O programables, distribuidos en tres puertos diferentes (A 3-bits, B 8-bits y D 7 bits) los cuales cuentan con acceso a distintos registros del microcontrolador con los cuales se pueden realizar múltiples funciones, las utilizadas para efectos de este laboratorio fueron la generación de interrupciones externas, la detección de flancos negativos, el control de salida de los pines y el conteo utilizando el reloj interno del dispositivo para interrupciones de conteo. Este microcontrolador cuenta con 120 instrucciones internas y su gran variedad de modos de operación permiten una gran versatilidad en su uso [1].

Su diagrama de pines se muestra en la hoja de datos y se muestra a continuación:

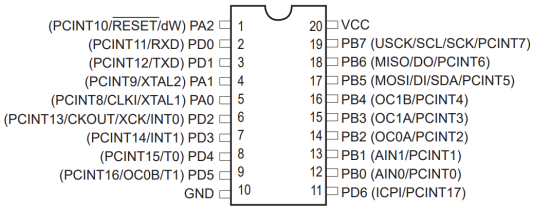

Figura 1. Diagrama de pines del ATTiny4313 [1].

Las especificaciones eléctricas generales de este dispositivo se ajuntan también en la hoja de datos:

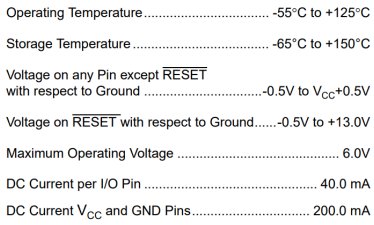

Figura 2. Características eléctricas del ATTiny4313 [1].

II-B. Interrupciones

Las interrupciones fueron una parte de este laboratorio. Básicamente consisten en señales internas o externas que le indican al sistema operativo que debe de tenerse o realizar una tarea distinta a la que estaba siendo realizada [2]. Como se mencionó, pueden ser generadas internamente por señales del CPU o de forma externa debido a algún cambio en la señal recibida por algún pin. Son muy útiles porque permiten tener un mejor flujo de trabajo ya que la última instrucción antes de la interrupción es almacenada en la cola del procesador mientras se resuelve dicha tarea para después reanudar la ejecución normal del programa. Para el caso de este proyecto, la interrupción externa fue implementada mediante la detección de un flanco negativo en uno de los pines del puerto D, mientras que la interna fue generada mediante la escalación de la señal de reloj interno ($8\,\mathrm{MHz}$) en un factor de 1024 y generando una interrupción cada vez que se completaba un ciclo de reloj.

II-B1. Interrupción externa: Se manejaron mediante el bit **INT0** fijado en 1 del registro **GIMSK** que se encuentra en el pin **PD2** y con el bit **ISC01** en 1 del registro **MCUCR** se pudo detectar los flancos negativos que fueran provocados mediante la alteración de la señal que llega a ese pin. El código en lenguaje C que realizaba esta tarea es el siguiente:

```
GIMSK |= (1 << INT0);
MCUCR |= (1 << ISC01);
```

Al momento en que se atiende la subrutina se cambia el estado binario de una variable entera volátil llamada **boton**.

```
ISR(INT0_vect){
    boton = !boton;
}
```

II-B2. Interrupción interna: Estas se manejaron mediante el Timer 1 del microcontrolador, este se debió establecer en modo CTC (Clear Timer Compare) poniendo en alto el bit WGM12, este modo se utiliza con el registro **OCR1A**, a este se le fija un valor que se comparará a cada instante con el valor del Timer y se resetea una vez que los valores coincidan. También, a como ya se mencionó, se tuvo que escalar el reloj interno del MCU en un factor de 1024 mediante los bits **CS12** y **CS10**.

La interrupción ocurrirá cada cierto periodo que para efectos de este reporte se llamará T_{OCnA}. La frecuencia respectiva se obtiene mediante una ecuación que se encuentra en la hoja de datos (pág 99, [1]):

$$f_{OCnA} = \frac{f_{clk}}{2 \cdot N \cdot (1 + OCR1A)} \tag{1}$$

Para efectos de simplicidad y para implementarlo de manera más sencilla en código se fija un periodo de $10\,\text{ms}$, lo que significa una frecuencia de $f_{OCnA} = 100\,\text{Hz}$. Sabiendo que la frecuencia interna de reloj es de $8\,\text{MHz}$ y el prescaler se fijó en 1024, por lo que se puede despejar el valor del registro **OCR1A**, tal que $OCR1A = 38$.

Obtenido este valor, el código correspondiente para generar la interrupción es el siguiente:

```
TCCR1B = 0x00;
TCCR1B |= (1 << WGM12) | (1 << CS12) | (1 << CS10);
TIMSK |= (1 << OCIE1A);
OCR1A = 38;
```

Se sabe entonces que cada $10\,\text{ms}$ se genera una interrupción que es atendida por el programa, entonces, dentro de la subrutina se agrega un acumulador que se incrementa en 1 cada vez que es atendida, esto con el fin de poder estimar cuantos ciclos equivalen a $1\,\text{s}$. Idealmente serían 100 ciclos pero el simulador puede agregar muchos desperfectos al MCU, entonces, para implementar los delays, se construyó una función que recibe como argumento la cantidad de tiempo en segundos que se quieren agregar al programa `delay(float seconds)`. Esta función es la siguiente:

```
void delay(float seconds) {
    segs = 0;
    while (segs < 188*seconds) {
        // No haga nada
    }
}
```

El factor de **188** que está dentro del ciclo while inicialmente se fijó en 1 y se intentó hacer parpadear un LED y se pasó como argumento a la función **3000**, esto provocó que el LED se encendiera a los $16\,\text{s}$, entonces por regla de 3 se calcula el factor correcto:

$$\frac{16}{3000} = \frac{1}{x}$$
$$x = 188$$

Por lo que se sabe entonces que ahora con este factor se escaló la función para poder agregar retardos en unidades de segundos.

III. Desarrollo/Análisis de Resultados

III-A. Hardware

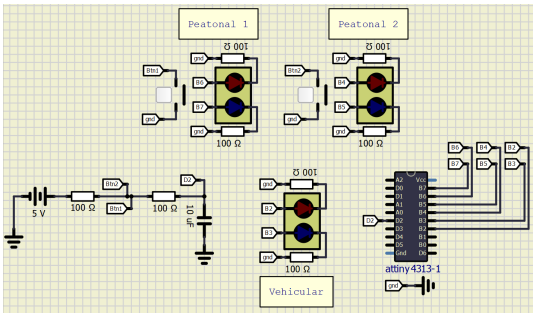

Figura 3. Diseño final del semáforo.

El diseño del semáforo realizado en el simulador hace uso de *Tunnels*, elementos que ofrece para evitar las confusiones debido a muchos cables dentro del diseño. La forma en que se conectaron los botones fue haciendo uso de resistencias pull-up y con un capacitor para evitar los rebotes del pulso. En lo que a los componentes se refiere, no se utilizó ningún componente fuera de lo convencional, a excepción del MCU. Para visualizar esto mejor, se adjunta una tabla (tabla I) con dichos componentes y sus precios (para los precios se usó como referencia Amazon).

Cuadro I
Componentes electrónicos necesarios para el diseño

Componente	Cantidad	Precio (Unidad)
ATTiny4313	1	$2.05
Resistencia 100 Ω	8	$5.99 (100 pcs)
LED Rojo	3	$6.99 (100 pcs)
LED Verde	3	$6.99 (100 pcs)
Capacitor 0,1 µF	1	$5.68 (25 pcs)
Switch	2	$6.98 (12 pcs)

Construido el diseño en el simulador, lo que resta es elaborar la lógica detrás del semáforo en lenguaje C. Para esto se tuvo que primero indicarle al MCU cuáles pines iban a actuar como salidas para poder hacer las conexiones de los LED, para el ATTiny4313 la forma de hacer esto es mediante desplazamiento de bits o de forma directa utilizando un número binario de 8 bits en el registro DDRB, haciendo lo siguiente con cada uno de los pines utilizados:

```
DDRB |= (1 << RED_PEATON1) | (1 << GREEN_PEATON1);
```

En donde `RED_PEATONn` es un `#define` que se colocó al inicio del código indicando cada uno de los pines. Como se trabaja con el puerto B, la forma de hacerlo es **PBn**, siendo **n** el número del puerto.

III-B. Máquina de Estados

El diseño propuesto para la máquina de estados que controla el semáforo peatonal es el siguiente:

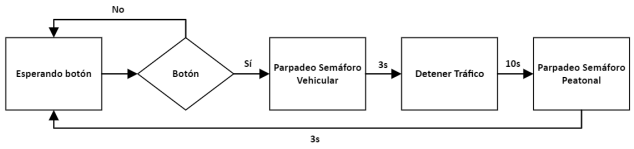

Figura 4. Diagrama de bloques de la máquina de estados para el semáforo.

Y a continuación se describen cada uno de los estados:

- **Esperando botón:** la luz verde vehicular y las rojas peatonales están activas. En este estado los carros pueden circular con normalidad y el tiempo que el programa permanece acá depende el usuario y cuándo decida tocar el botón. Esta es la única entrada del sistema y presionarlo es la forma de generar la transición al estado siguiente.
- **Parpadeo Semáforo vehicular:** durante 3 s la luz verde del semáforo vehicular parpadea.
- **Detener tráfico:** el semáforo vehicular cambia a rojo y un segundo después el semáforo peatonal cambia a verde y se mantiene este estado por 10 segundos.
- **Parpadeo Semáforo Peatonal:** en este estado parpadean por 3 segundos las luces verdes del semáforo peatonal y cambian a rojo, un segundo después se transiciona al estado inicial a esperar a que el botón vuelva a ser presionado.

III-C. Resultados

La implementación de esta máquina de estados y del programa en general fue exitoso, sin embargo, debido a que hay estados donde el semáforo parpadea, no se puede mostrar con imágenes su correcto funcionamiento. Para poder abordar esto, se realizó una grabación de pantalla que puede ser accedida en este **LINK**, siendo este el adjunto más importante de los resultados obtenidos. Sin embargo, se agregarán imágenes a continuación de los estados de **Esperando Botón** y **Detener tráfico** que es donde las luces están estáticas:

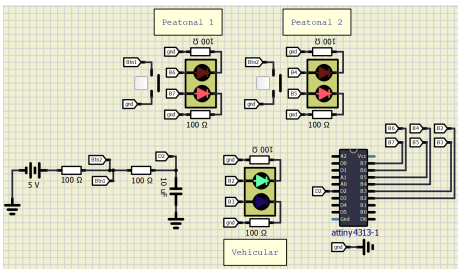

Figura 5. Estado **Esperando Botón**.

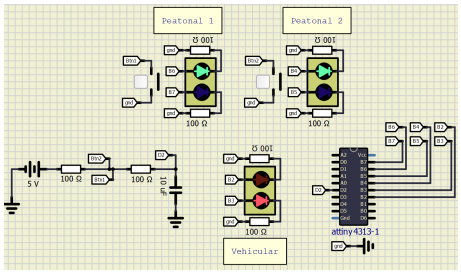

Figura 6. Estado **Detener tráfico**.

IV. Conclusiones y Recomendaciones

Se puede concluir que la implementación de la máquina de estados y el manejo de las interrupciones fue exitoso, se comprendió cómo tratar estos mecanismos en un MCU y podrán ser utilizados en otros dispositivos en proyectos futuros. A modo de recomendación está leer a detalle la hoja de datos y comprender primero qué es lo que se quiere implementar para saber qué secciones de esta hoja de datos consultar pues usualmente su extensión es de varias decenas de páginas.

REFERENCIAS

[1] ATMEL. ATtiny2313A/4313 Data Sheet. https://ww1.microchip.com/downloads/en/DeviceDoc/doc8246.pdf, 2011.

[2] Bhanu Priya. What are different types of interrupts? https://www.tutorialspoint.com/what-are-different-types-of-interrupts, 2023.