Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE0624 – Laboratorio de Microcontroladores
II ciclo 2022

Laboratorio 1

GPIOs, Timers y FSM

Kenny Wu Wen C08592
Grupo 01

Profesor: MSc. Marco Villalta Fallas

17 de septiembre de 2023

Índice

Índice de figuras

1. Resumen

En este laboratorio se van a utilizar las interrupciones del microcontrolador ATtiny con el fin de crear un sistema de semáforos para autos y peatones de forma sincronizada. Primero, se va a entender como funcionan los registros del ATtiny, para luego, entender como utilizar y funcionan las interrupciones del ATtiny, después se conecta la electrónica necesaria y se escribe el firmware para el funcionamiento los semáforos.

El repositorio para el laboratorio es el siguiente: GIT

2. Nota teórica

2.1. ATtiny4313

Los ATtiny son microcontroladores pequeños de 8 bits que tienen poca memoria flash y EEPROM, tiene versiones con 6 hasta los 32 pines [2]. Para este laboratorio, se utiliza el ATtiny4313 que tiene las siguientes características según la hoja de fabricante de Atmel [1]:

- Arquitectura de RISCV
- Contador de 8 bits
- contador de 16 bits
- Cuatro canales de PWM
- Comparador analogico
- Interrupciones externas e internas
- 20 pines
- Tensión de operación de 1.8 a 5.5 V
- Memoria flash con 4K bytes de espacio
- Memoria EEPROM con 256 Bytes
- Memoria RAM con 256 Bytes item

En la figura 1, se puede observar un diagrama de pines del microcontrolador.

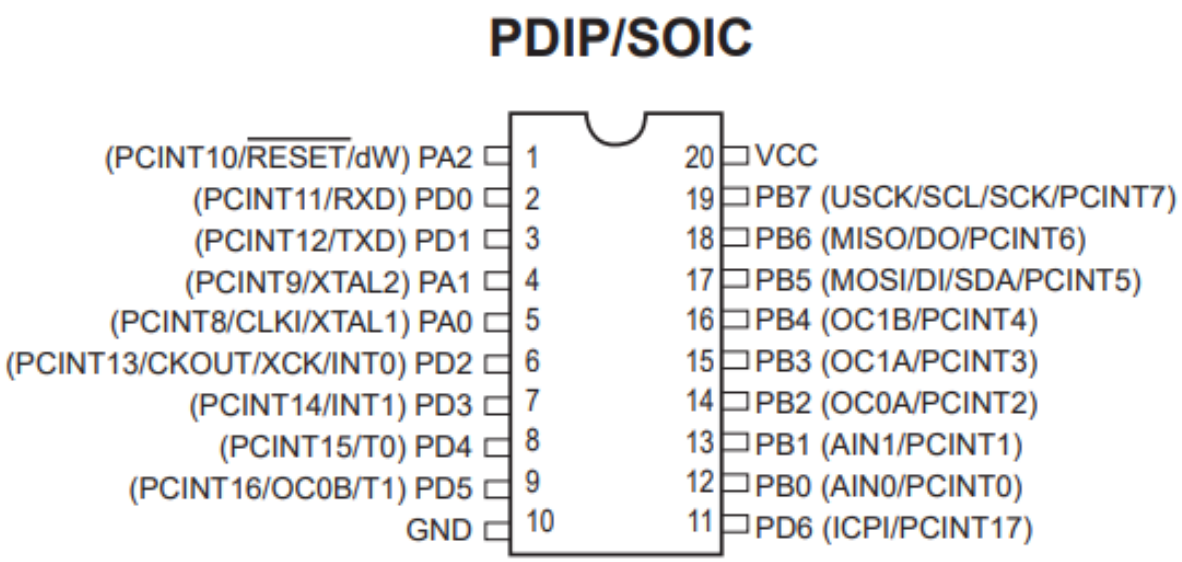

Figura 1: ATtiny4313 diagrama de pines [1]

2.1.1. GPIOs

Para el Attiny, se tiene tres registros para controlar los GPIOs los cuales son DDxn, PORTxn y PINxn.

Los bits del DDxn representa si un pin es una entrada o salida, de forma que si es uno son salidas y si son cero son entradas.

Los bits del PORTxn representan varias cosas dependiendo del valor de DDxn, si el bit PORTxn está en alto y DDxn en cero, se activa la resistencia de pull-up, para desactivarlo se coloca en bajo el bit PORTxn o DDxn en uno. Cuando DDxn es uno, los bits del PORTxn representan lógica alta o logica baja.

Los bits del PINxn se puede leer y cambiar sin importar los valores de DDxn y PORTxn. En la figura 2, se tiene un resumen de las configuraciones.

DDxn	PORTxn	PUD (in MCUCR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

Figura 2: Configuraciones de GPIOs

2.1.2. Timers

Para este laboratorio se puede usar ambos timers (8 y 16 bits), pero solo se va a utilizar el de 8 bits y ambos timers funcionan de manera parecida.

El timer de 8 bits, como se sabe llega a contar hasta $2^8 - 1 = 255$. El timer tiene varios modos de operación como ser un contador o un comparador de salidas o PWM, para nuestro caso se utilizara en modo contador o modo normal.

Para configurar el timer en modo normal hay que modificar el registro TCCR0A 3. Para que funciona en modo normal, se tiene que cambiar los bits [5:4] = 0b00 4.

TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x30 (0x50)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 3: Registro TCCR0A

Table 11-2. Compare Output Mode, non-PWM Mode

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Toggle OC0A on Compare Match
1	0	Clear OC0A on Compare Match
1	1	Set OC0A on Compare Match

Figura 4: Registro TCCR0A modos

Luego, para que el timer en modo contador/normal cuente de forma que se pueda percibir para el humano o tiempo real, se tiene que preescalar el como cuenta, para eso tenemos que modificar el registro TCCR0B 5 y sus primeros tres bits y elegir un modo según la figura 6

TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x33 (0x53)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 5: Registro TCCR0B

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Figura 6: Registro TCCR0B modo de reloj

2.1.3. Interrupciones

Una interrupción en un microcontrolador es un mecanismo que permite responder a eventos externos o internos, congelando temporalmente la ejecución del programa principal para ejecutar una subrutina de servicio de interrupción (ISR).

Existen dos tipos de interrupciones, por hardware y por software o externas e internas. Las por hardware ocurren cuando ocurre dentro del microcontrolador como un cambio en un valor interno, mientras que en software ocurre cuando una instrucción llame una rutina de servicio asociada. También se puede seperar por interrupciones internas o externas, las internas son eventos de hardware/software dentro del microcontrolador mientras que los externos son eventos que se controlan fuera del microcontrolador [3].

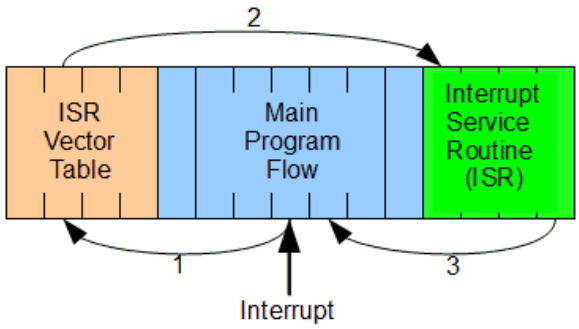

Figura 7: Interrupciones

Cuando ocurre una interrupción, se para de ejecutar el programa principal y se va a buscar una tabla de vector de interrupciones el cuál indica la prioridad de interrupciones para saber cual subrutina de interrupción ejecutar como se observa en la figura 7.

La tabla de vectores de interrupciones del ATtiny4313 es la que se muestra en la figura 8.

Vector No.	Program Address	Label	Interrupt Source
1	0x0000	RESET	External Pin, Power-on Reset, Brown-out Reset, and Watchdog Reset
2	0x0001	INT0	External Interrupt Request 0
3	0x0002	INT1	External Interrupt Request 1
4	0x0003	TIMER1 CAPT	Timer/Counter1 Capture Event
5	0x0004	TIMER1 COMPA	Timer/Counter1 Compare Match A
6	0x0005	TIMER1 OVF	Timer/Counter1 Overflow
7	0x0006	TIMER0 OVF	Timer/Counter0 Overflow
8	0x0007	USART0, RX	USART0, Rx Complete
9	0x0008	USART0, UDRE	USART0 Data Register Empty
10	0x0009	USART0, TX	USART0, Tx Complete
11	0x000A	ANALOG COMP	Analog Comparator
12	0x000B	PCINT0	Pin Change Interrupt Request 0
13	0x000C	TIMER1 COMPB	Timer/Counter1 Compare Match B
14	0x000D	TIMER0 COMPA	Timer/Counter0 Compare Match A
15	0x000E	TIMER0 COMPB	Timer/Counter0 Compare Match B
16	0x000F	USI START	USI Start Condition
17	0x0010	USI OVERFLOW	USI Overflow
18	0x0011	EE READY	EEPROM Ready
19	0x0012	WDT OVERFLOW	Watchdog Timer Overflow
20	0x0013	PCINT1	Pin Change Interrupt Request 1
21	0x0014	PCINT2	Pin Change Interrupt Request 2

Figura 8: Tabla de interrupciones

Para este laboratorio, se va a necesitar activar dos interrupciones, uno para cuando se

presiona algún boton cambiando el valor de un pin y otro para crear retraso, por lo cual se utiliza el overflow del timer0.

2.1.4. Interrupción Botón

La primera interrupción se tiene que modificar el registro *General interrupt Mask Register* (GIMSK) y modificar el bit que quiere para que sea válido como interrupción.

GIMSK – General Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x3B (0x5B)	INT1	INT0	PCIE0	PCIE2	PCIE1	–	–	–	GIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Figura 9: Registro Máscara de interrupciones generales

Para este laboratorio se utilizara las interrupciones por cambios en los pines PCIE0, los cuales son PCINT7...0, para saber cuál de los 8 pines exactamente se quiere usar como interrupción se debe modificar el registro *Pin Change Mask Register 0* (PCMSK0/PCMSK) y habilitar con un uno los pines a utilizar.

PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
0x20 (0x40)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 10: Registro Mascara de pines de cambio

2.1.5. Interrupción para retraso con timer0

Para crear el retraso por interrupción, se va utilizar el overflow del timer0, para eso hay que configurar que se habilite la interrupción por overflow del timer0, por lo que hay que modificar el registro – *Timer/Counter Interrupt Mask Register* (TIMSK) y poner en uno el bit 1, TOIE0 que activa la interrupción por overflow.

TIMSK – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x39 (0x59)	TOIE1	OCIE1A	OCIE1B	–	ICIE1	OCIE0B	TOIE0	OCIE0A	TIMSK
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 11: Registro máscara de interrupcion de contador/timer

2.1.6. avr-gcc

El compilar avr-gcc de lenguaje C, es para compilar microcontroladores AVR, como lo es el ATtiny4313. Este es una versión del compilador GCC (GNU Compiler Collection) la cuál ha sido modificada para generar código para los microcontroladores AVR [4].

2.2. Máquina de Estados Finitos (FSM)

Una máquina de estados finitos se le conoce como un modelo el cuál se divide en una cantidad de estados y la salida no depende totalmente de las señales de entrada sino, también de las anteriores entradas. Por ejemplo, en la figura 12 se puede observar dos estados, cerrado y abierto, y dependiendo de la entrada anterior, del estado en que este y la entrada actual, se cambia de un estado a otro y la salida es diferente.

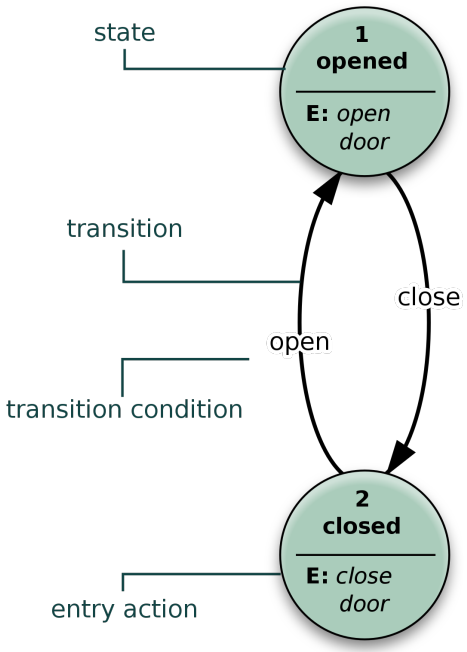

Figura 12: Ejemplo de una máquina de estados finitos simple

2.3. Sistema de semáforos

Para este laboratorio se debe desarrollar un cruce de semáforos simplificado como el de la figura 1, utilizando leds, botones y el microcontrolador ATtiny4313. Se utilizarán al menos seis leds, los leds LDPV y LDVD representarán el semáforo vehicular, mientras que LDPP y LDPD es la representacion de un semáforo peatonal. El botón B1 y/o B2 serán los puertos con el que el usuario solicitará la activación de las luces peatonales. Finalmente, el sistema deberá cumplir con el diagrama de temporización de la figura 13.

El botón B1 o B2 se puede presionar inclusive antes que se acaben los 10s del funcionamiento de LDPV, pero deberá permanecer encendido hasta que terminen los 10s. Si no se presiona el botón LDPV deberá permanecer encendido indefinidamente

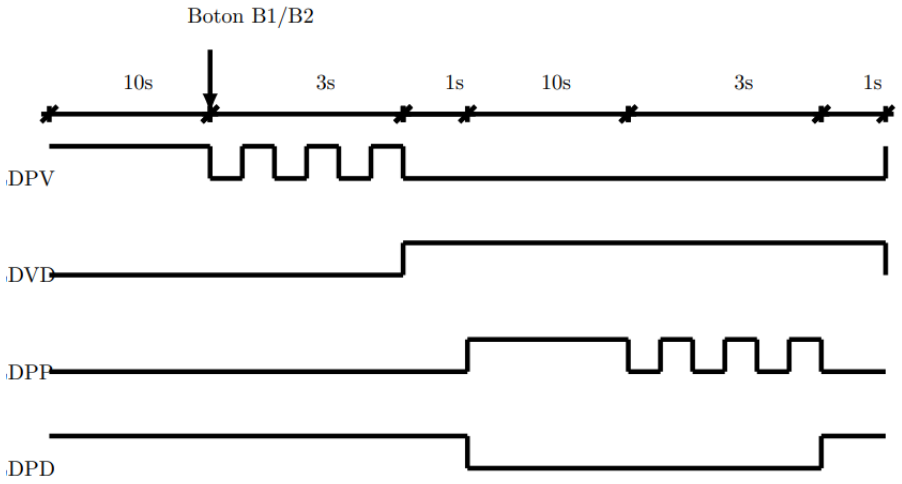

Figura 13: Diagrama de temporización

2.4. Firmware

2.4.1. Setup

Está función se encargara de inicializar los GPIOs, interrupciones y timer0 que se van a utilizar.

- Se inicializa el registro DDRB con el valor 0x38, que significa que se pone como salida los pines PCINT3, PCINT4 y PCINT5.
- Se inicializa el registro DDRD con el valor 0x1c, que significa que se pone de salida los pines PCINT13, PCINT14 y PCINT15.
- Se inicializa el registro GIMSK con el valor (1 ≪ PCIE0) que significa poner en alto el bit PCIE0 del registro.
- Se inicializa el registro PCMSK0 con el valor 0x3, que significa que los pines PCINT0 y PCINT1 son interrupciones válidas.
- Se inicializa el registro TCCR0A con 0x00, para configurarlo en modo normal
- Se inicializa el registro TCCR0B con 0b011, que significa una pre-escala de 64 al contador.
- Se inicializa el contador interno timer0 con 0 con el registro TCNT0
- Se llama el método sei() para activar las interrupciones globales.

```
void setup() {
  // Pines B
  DDRB   = 0x38; //Configuracion del puerto
  // Pines D
  DDRD   = 0x1c;
  GIMSK  |= (1 << PCIE0);
  PCMSK  |= 0x03;
  // Configuracion del timer
  TCCR0A |= 0x00; // modo normal
  TCCR0B |= 0b011; // prescala con 64
  TCNT0  &= 0x00; // Iniciar contador interno Timer0
```

```
  sei();
}
```

2.4.2. Interrupciones

Primero tenemos la interrupción cuando alguno de los botones sea presiona, lo que ocurrirá sera que la variable botón se pone en alto

```
// Interrupcion boton B0/B1
ISR (PCINT_B_vect) {
  boton  = 1;
  PCMSK  &= 0x00;
}
```

Segundo tenemos la interrupción de overflow del timer0, lo que hará sera verificar si la variable **en_delay** está en alto para aumentar el valor de **times** a uno y para terminar se desactiva la interrupción de overflow asignando 0x00 al registro TIMSK.

```
// Interrupcion overflow Timer0
ISR (TIMER0_OVF_vect) {
  // Si se usa la funcion
  // delay se aumenta times
  // para contar el tiempo
  if (en_delay) times = times + 1;
  // Se desactiva la interrupcion
  // luego
  TIMSK  &= 0x00;
}
```

2.4.3. Delay

El método de delay se va a utilizar la interrupción del overflow del timer0 para hacer un efecto de retraso de tiempo. Lo primero que hace la función es asignar **en_delay** en alto, luego entra en un loop while hasta que la variable de entrada **overflow** sea menor a la variable **times** y va a estar activando la interrupción de overflow asignando al registro TIMSK como 0x02. Al salirse del loop while, va a bajar el valor de **en_delay** y **times**.

```
void delay(int overflows) {
  // se activa el delay de interrupcion
  en_delay = 1;
  // Loop hasta que transcurra overflows
  while (times<overflows) {
    // Se activa la interrupcion
    TIMSK |= 0x02;
  }
  // Se reinicia las variables
  en_delay = 0;
  times = 0;
}
```

2.4.4. FSM

Se utilizo la la FSM de la figura 14.

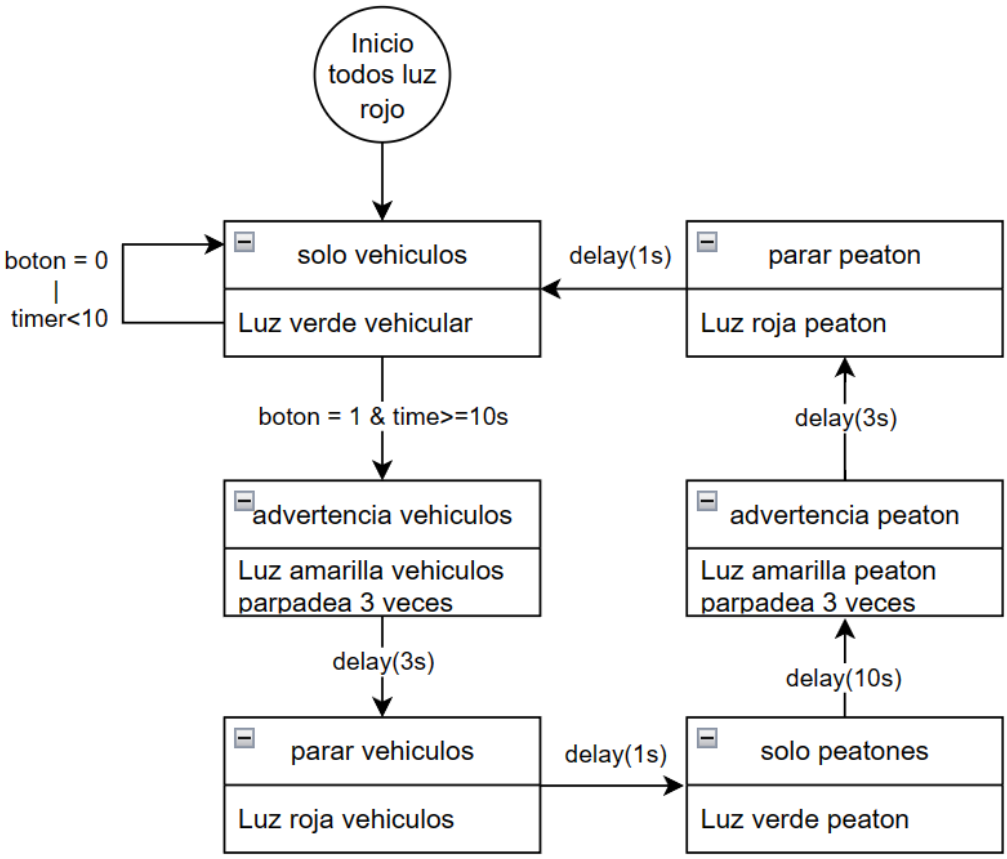

Figura 14: FSM del sistema de semáforos

Los estados:

Inicio :

Todos los semáforos en luz roja

Solo vehiculos :

Luz verde vehicular, debe pasar mínimo 10s luego de eso, si se presiono el boton en cualquier estado se pasa al estado advertencia vehiculos.

Advertencia vehiculos :

Luz amarilla vehicular parpadea tres veces y debe pasar tres segundos para pasar al siguiente estado parar vehiculos,

Parar vehiculos :

Luz roja vehicular, debe pasar un segundo y pasa al estado solo peatones.

Solo peatones :

Luz verde peatonal, debe pasar 10s y se pasa al estado advertencia peaton.

Advertencia peaton :

Luz amarilla peatonal parpadea tres veces y debe pasar tres segundos para pasar al siguiente estado parar peaton.

Parar peaton :

Luz roja peaton, debe pasar un segundo y pasa al estado solo vehiculos.

2.5. LEDs

Los LEDs (*light-emitting diode*) son diodos emisores de luz. Contienen dentro un semiconductor el cual al recibir una diferencia de potencial continua, produce luz, fenómeno conocido como electro luminiscencia [5].

Cuando se conecta un diodo LED, la tensión que aguanta es normalmente de 2 V, en caso de ser mayor se debe colocar una resistencia en serie para no quemar el LED.

En este laboratorio se van a usar LEDs amarillos de 3mm con un costo de 100 colones por unidad. Funcionan a 2V y 20mA [6].

Como los pines del PIC entregan una tensión de 5 V aproximadamente, por lo tanto la resistencia de protección va ser:

$$R_{led} = \frac{5-2}{20 \times 10^{-3}}$$
$$\boxed{R_{led} = 150\Omega} \tag{1}$$

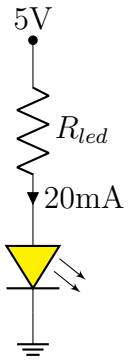

Figura 15: LED conectado a un pin del PIC

Para dos LEDs en serie con los mismas parámetros:

$$R_{led} = \frac{5 - 2 \cdot 2}{20 \times 10^{-3}}$$
$$\boxed{R_{led} = 50\Omega} \tag{2}$$

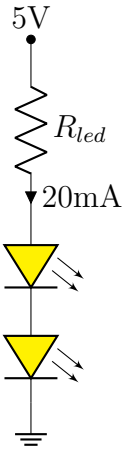

Figura 16: LED conectado a un pin del PIC

2.6. Botón

Para el botón conectado al pin de GPIO3 del PIC, se va a utilizar configuración pull-up, osea, estará lógica 1 cuando no se pulse el botón y en lógica 0 cuando se pulse. Tambíen se le conoce como lógica negativa.

Como se observa en la figura 17, la señal que va a recibir el GPIO3 va ser alta hasta que se pulse el botón. Utilizar esta configuración nos permite proteger el microcontrolador para que la corriente que le llegue sea cercana a cero. Para eso, vamos a utilizar una resistencia de:

$$R_{boton} = 1k\Omega \tag{3}$$

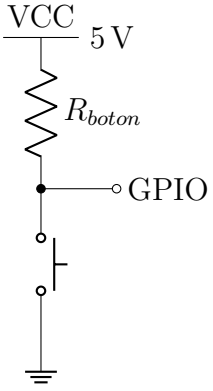

Figura 17: Configuración botón

Tambien hay que considerar el rebote al presionar el botón, ya que no es ideal el botón, por ende existen varios métodos para eliminar este efecto, tanto por software como hardware. Para este laboratorio se utilizara un método por hardware que es incluir un capacitor para que

existe un tiempo de subida y bajada de tensión de salida al presionar el boton para así eliminar el efecto de rebote [7].

Para calcular el valor del capacitor en paralelo al botón, hay que establecer el valor de T, en nuestro caso si queremos sea de $T = 100\mu$s, implica que $C = \frac{R}{T} = \frac{100 \times 1 - ^{-6}}{1 \times 10^3} = 100$nF.

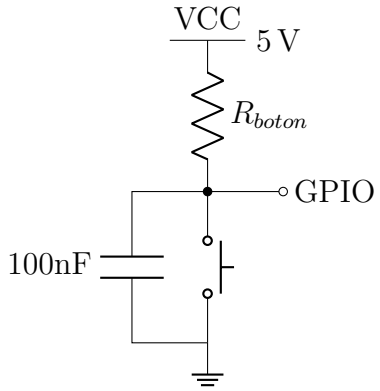

Figura 18: Configuración pull-up

3. Desarrollo/Análisis de resultados

3.1. Análisis de funcionalidad electrónica

En la figura 20, se puede observar que ninguna corriente pasa de los 20mA que aguanta los LEDs, solo hay que considerar que los LEDs de semáforo peatonal están en paralelo y al tener una resistencia interna la corriente de 33.22mA se divide en dos pero ambos LEDs reciben la misma tensión de 2.01V. Para los LEDs semáforos vehiculares solo están en serie y su corriente no sobrepasa los 20mA.

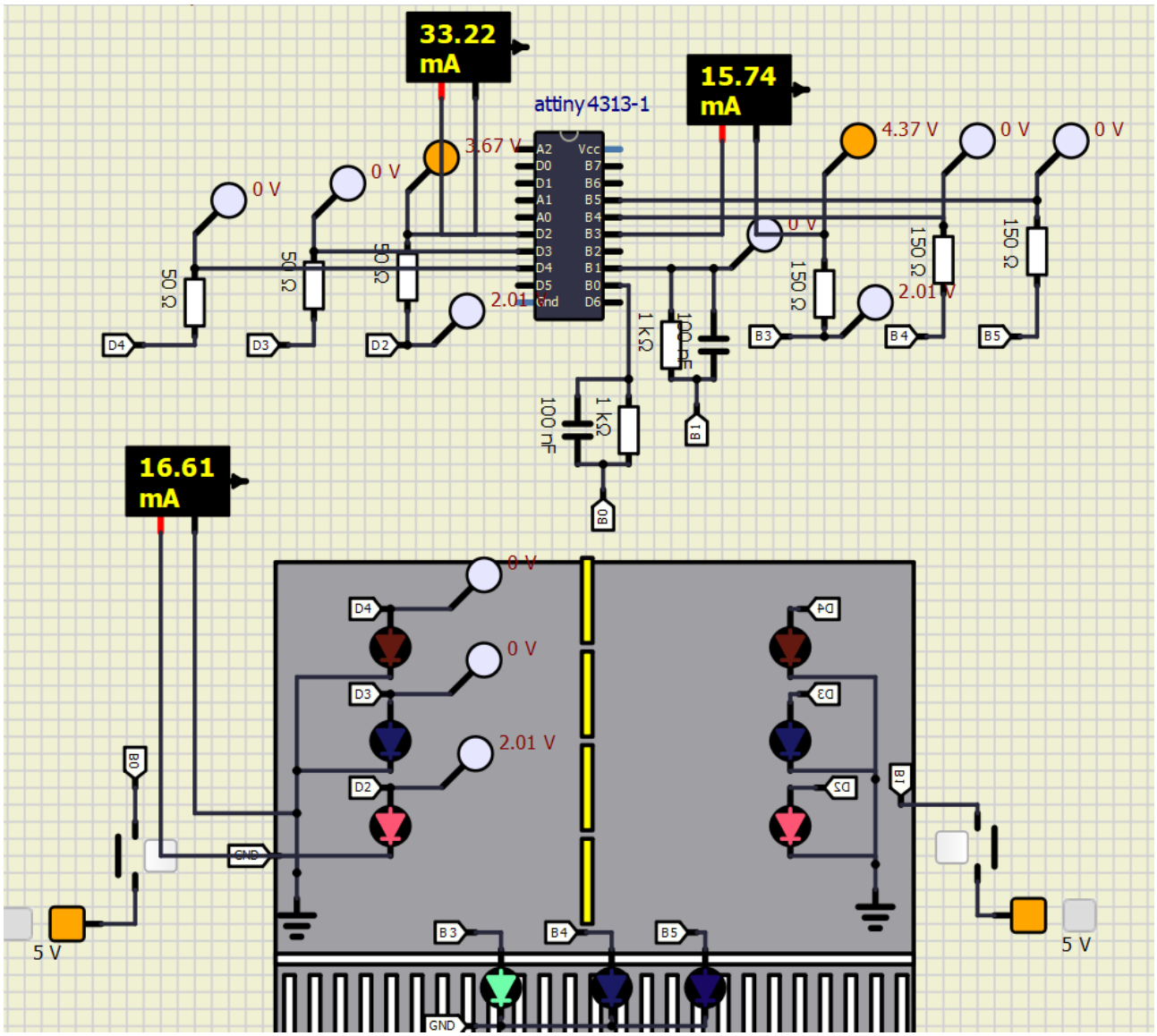

Figura 19: Análisis de corrientes y tensiones en los LEDs

Para el boton, al utilizar un oscilocopio podemos ver que al agregar el capacitor para evitar el efecto rebote, resulta en un tiempo de respuesta de 500μs como lo esperado para llegar al valor máximo como se observa en la figura **??**

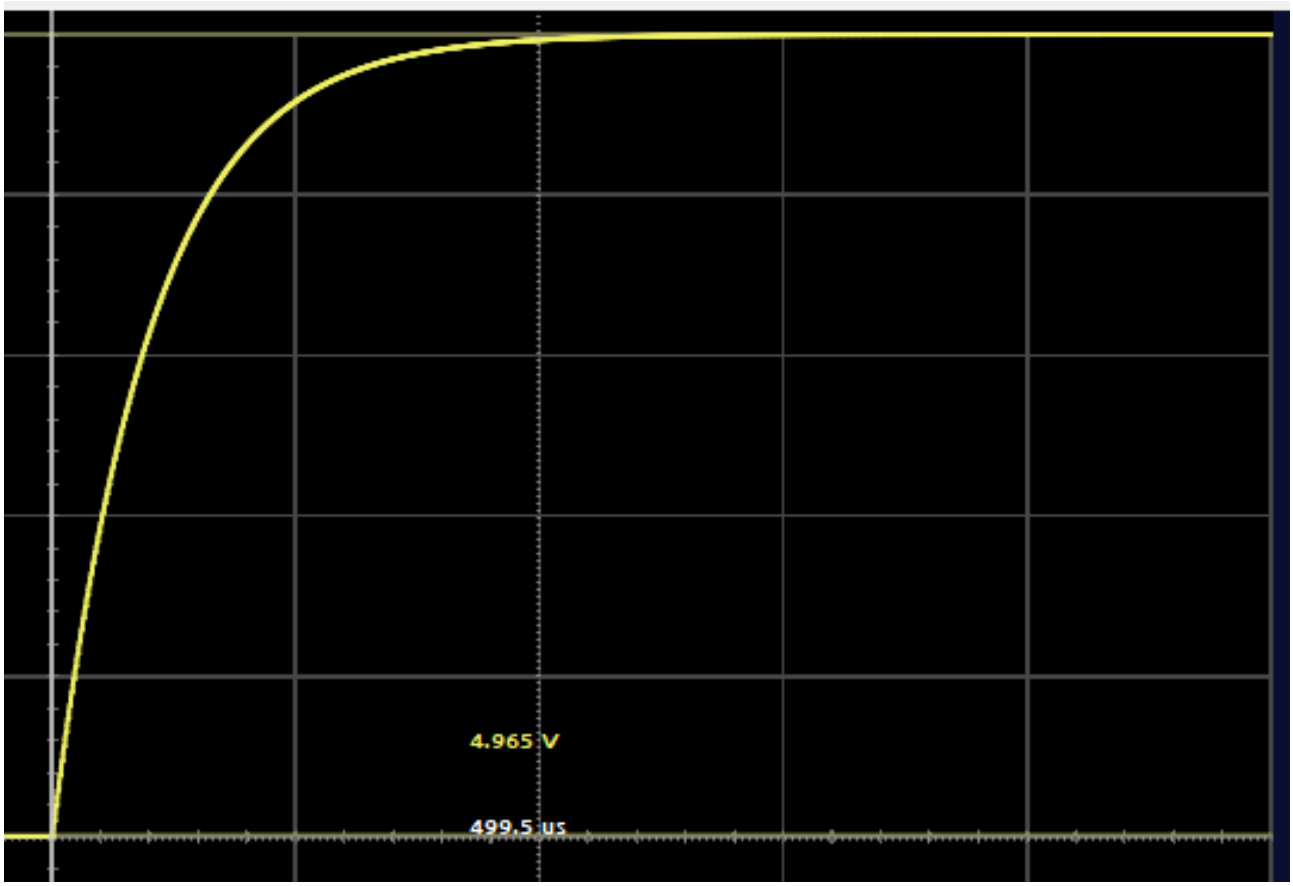

Figura 20: Análisis de anti rebote para el botón

3.2. Análisis de la funcionalidad del programa

Al ser en tiempo real el modo que funciona el circuito, es difícil a base de imágenes observar complementa el comportamiento, por lo que solo se mostrada que el sistema pasa a cada estado, desde el estado **solo vehiculo** hasta regresar a ese mismo estado.

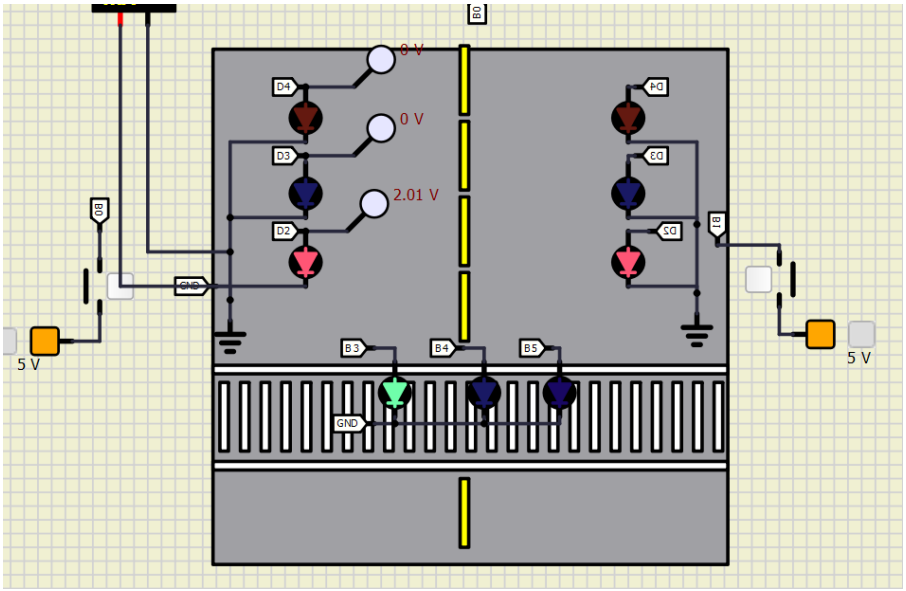

Figura 21: Estado solo vehiculo

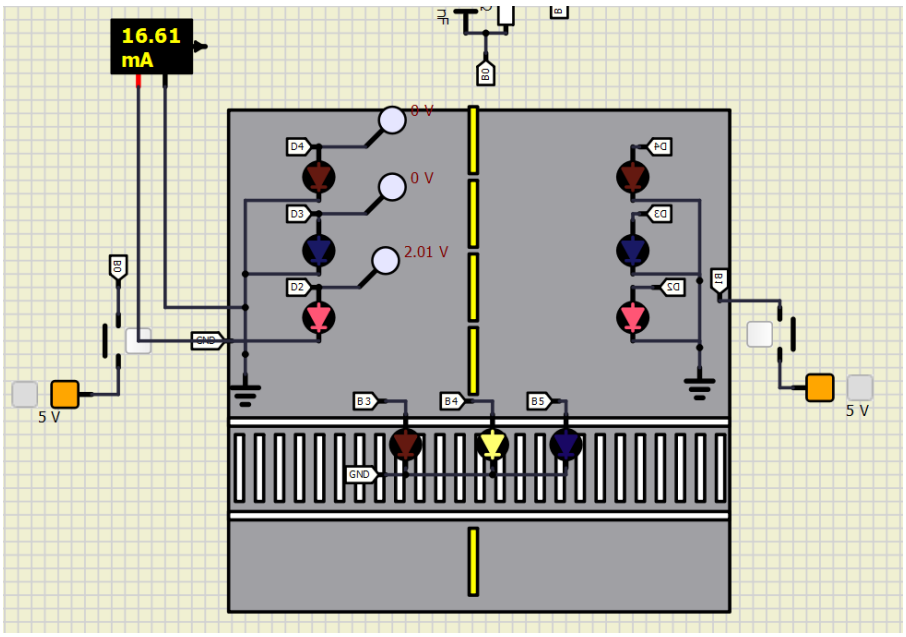

Figura 22: Estado advertencia vehiculo

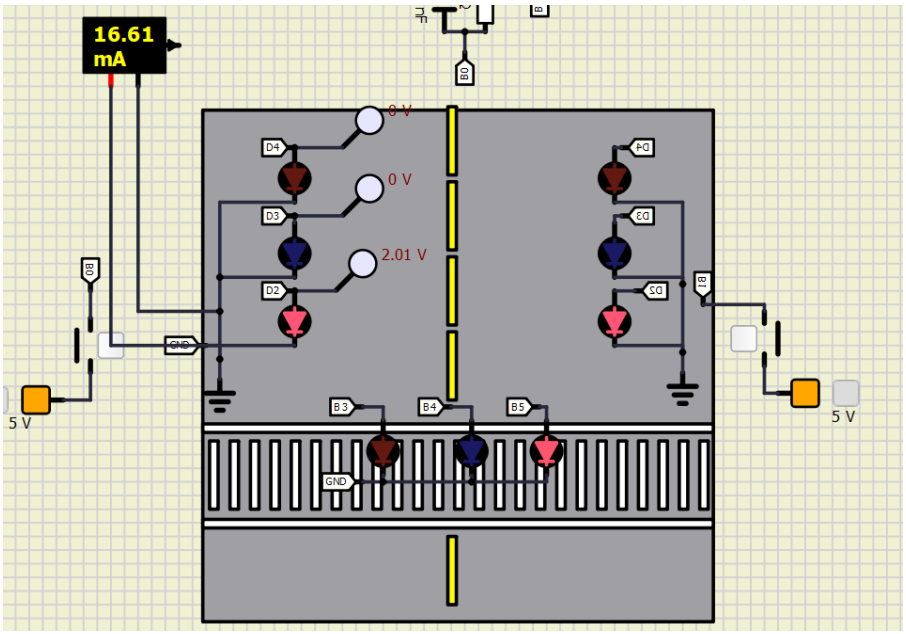

Figura 23: Estado parar vehiculo

En las anteriores figuras, se puede observar como va pasando de estado por estado luego de presionar el botón, ya que el único estado que depende del botón es el primero y los demás depende del tiempo transcurrido para pasar de estado. En los estados 22 y 25, no se puede apreciar que este parpadeando la luz amarilla, sin embargo, se puede notar que es menos brilloso el amarillo ya que la captura de la imagen es en tiempo real.

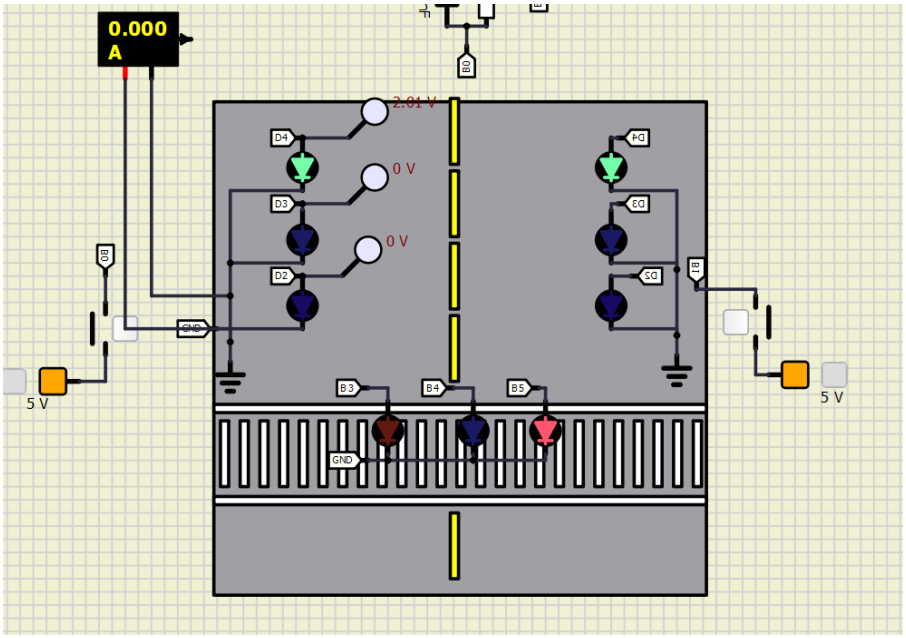

Figura 24: Estado solo peaton

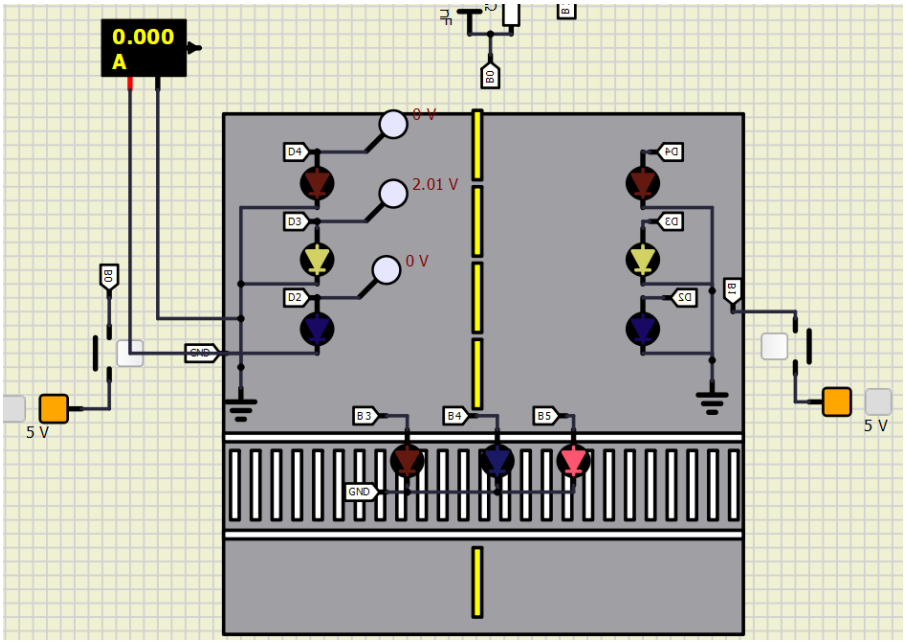

Figura 25: Estado advertencia peaton

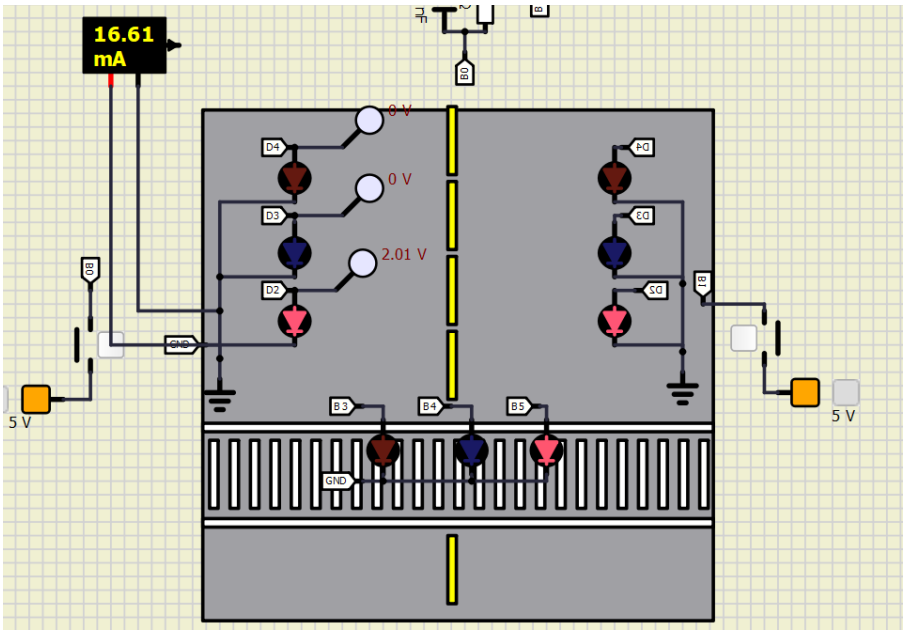

Figura 26: Estado parar peaton

4. Conclusiones y recomendaciones

- Siempre calcular la resistencia de protección para los LEDs.
- Tomar en cuenta la corriente o posibles cortos al microcontrolador, con el fin de cuidarlo.
- Tener un orden en los cables y un esquemático de como serán las conexiones antes de hacerlas.
- A la hora de escribir el firmware, tener orden en el código para entender bien que hace cada sección del código.
- Se logro crear un FSM para el sistema de semáforos.
- En el SimulIDE, se pueden observar los valores de los registros del PIC en tiempo real, para facilitar el debugeo.
- Se logro crear un firmware que las variables cambiaran por interrupciones, permitiendo que el CPU hiciera trabajo útil
- Se logro implementar un delay en base a interrupciones con el timer0 y su interrupción de overflow
- Se comprendió el funcionamiento e importancia de las interrupciones a la hora de escribir firmware.

Referencias

[1] Microchip, “Attiny4313,” https://www.microchip.com/en-us/product/ATtiny4313, accedido el 15 de septiembre de 2023.

[2] Diarioelectronicohoy, “Programador para attiny,” https://www.diarioelectronicohoy.com/blog/programador-para-attinys, accedido el 15 de septiembre de 2023.

[3] S. A. Carrasco, “Interrupciones en los microcontroladores,” https://dctrl.fi-b.unam.mx/~salva/Docs_CIBM_SEIC/Tema1_concepto_int.pdf, accedido el 15 de septiembre de 2023.

[4] G. C. Collection, “Avr-gcc,” https://gcc.gnu.org/wiki/avr-gcc, accedido el 15 de septiembre de 2023.

[5] “¿qué es un led? tipos de led, aplicaciones y usos.” Visual Led, 2023. [Online]. Available: https://visualled.com/glosario/que-es-un-led/

[6] “Led 3mm amarillo,” DBU Electronics, 2023. [Online]. Available: https://www.dbuelectronics.cr/leds/624-led-3mm-amarillo.html

[7] C. M. Maxfield”, “Cómo implementar el rebote de hardware para interruptores y relés,” https://www.digikey.com/es/articles/how-to-implement-hardware-debounce-for-switches-and-relays, 2021, [Consultado el 30 de agosto de 2023].

5. Apéndice

5.1. Precios de componentes

Cuadro 1: Precios de cada componente(Tienda)

Componente	Precio (Colones)	Cantidad
ATtiny4313	700-1500	1
1K ohm	200	2
50 ohm	300	3
150 ohm	300	3
Botón	200-500	2
Cerámico 0.1uF	80	2
LEDs 3mm	100	9