EIE

Escuela de
Ingeniería Eléctrica

Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

UNIVERSIDAD DE COSTA RICA

IE0624 - Laboratorio de Microcontroladores

LABORATORIO 1 : GPIOs, Timers y FSM

Oscar Fallas Cordero. B92861

Entrega: 17/Septiembre/2023

---

1. Introducción

El siguiente laboratorio permite la familiarización de una nueva familia de microcontroladores, en específico el Attiny 43-13, utilizando su diseño para crear un semáforo simplificado. Entre las tareas más importantes de este laboratorio se incluyen conocimiento y manipulación de pines de entrada y salida del microcontrolador, estudio de las interrupciones internas y externas, manipulación de la interrupción para el uso de timer, y construcción de maquinas de estado.

2. Nota Teórica

2.1. Microcontrolador ATinny

Los microcontroladores Attiny son dispositivos electrónicos altamente versátiles y compactos que desempeñan un papel fundamental en el mundo de la electrónica y la informática embebida. Fabricados por Microchip Technology, estos diminutos chips ofrecen un potente conjunto de capacidades de procesamiento y control en un espacio extremadamente reducido. El microcontrolador ATtiny 43-13 tiene un diseño descrito mediante el siguiente diagrama:

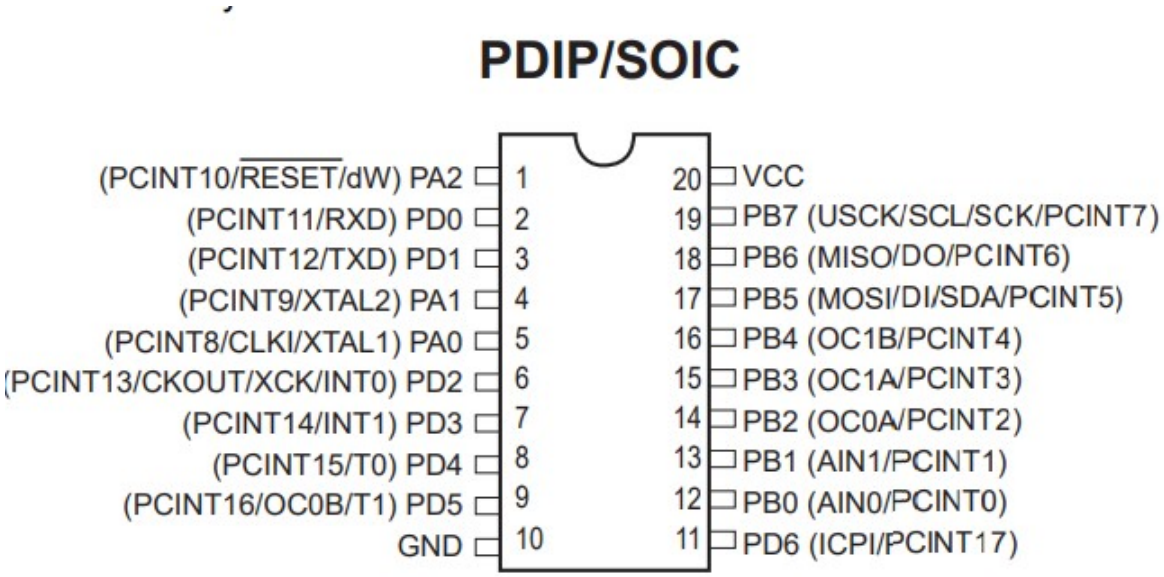


Figura 1: Diagrama de pines ATtiny 43-13 [1]

Entre las caracterísiticas más imporates a analizar del diagrama tenemos:

- El microcontrolador posee un número más elevado de pines, separados por familias o registros tipo A, B y D. A continuación podemos apreciar como se contruye cada uno de estos registros:

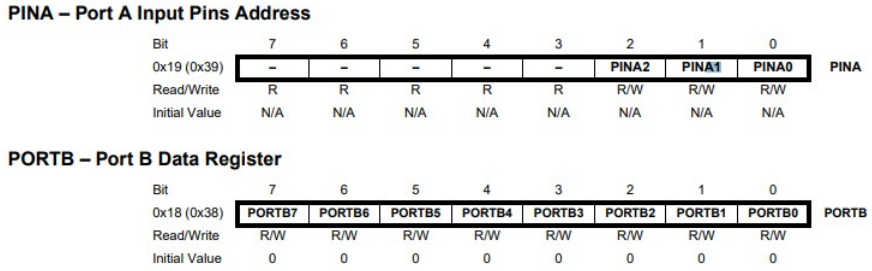


Figura 2: Descripción de Registros de Pines A y B[1]

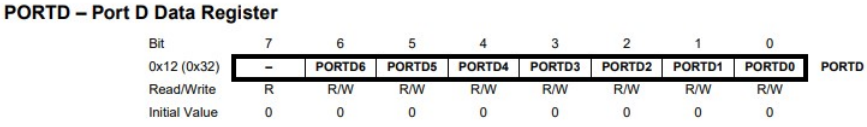
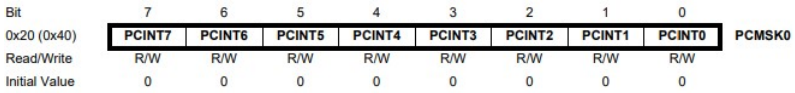


Figura 3: Descripción de Registros de Pines D[1]

  La manipulación lógica de valores en estos pines se puede realizar ingresando un 1 lógico al pin específico y en su caso contrario un 0 lógico.

- La asinación de entrada y salida de los pines se por los registros DDRx, el cuál asigna un 1 lógico a la salida y un 0 a la entrada.

- El Attiny4313 cuenta con varios pines de entrada/salida que pueden ser configurados para generar interrupciones externas. Cuando ocurre un cambio de nivel en uno de estos pines, el microcontrolador puede suspender temporalmente la ejecución del programa principal para atender la interrupción. La interrupción utilizada en este laboratorio fue la interrupción interna INT0, para utilizar esta interrupción se utilizó el registro PCMSK0.



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 4: Registro para manipulación de INT0[1]

Un aspecto importante a tomar en cuenta, es la polaridad de la interrupción, en mi caso utilicé que la interrupción se activacé en un flanco decreciente de reloj. Esto se hace mediante la manipulación del registro MCUCR.

- Los temporizadores pueden configurarse para generar interrupciones en momentos específicos, lo que resulta especialmente útil para llevar a cabo tareas sincronizadas en aplicaciones como la generación de pulsos, la medición de frecuencias, o el control de motores paso a paso. Al programar el Timer/Counter con un valor de cuenta deseado, se puede precisar cuándo se ejecutará una acción o interrupción. Entre los aspectos más importantes para el uso de timers, son los registros de comparación(0CR0A y OCR0B) y el registro principal(TCNT0). Ambos descritos en.

3. Desarrollo/Análisis de Resultados

3.1. Construcción del circuito

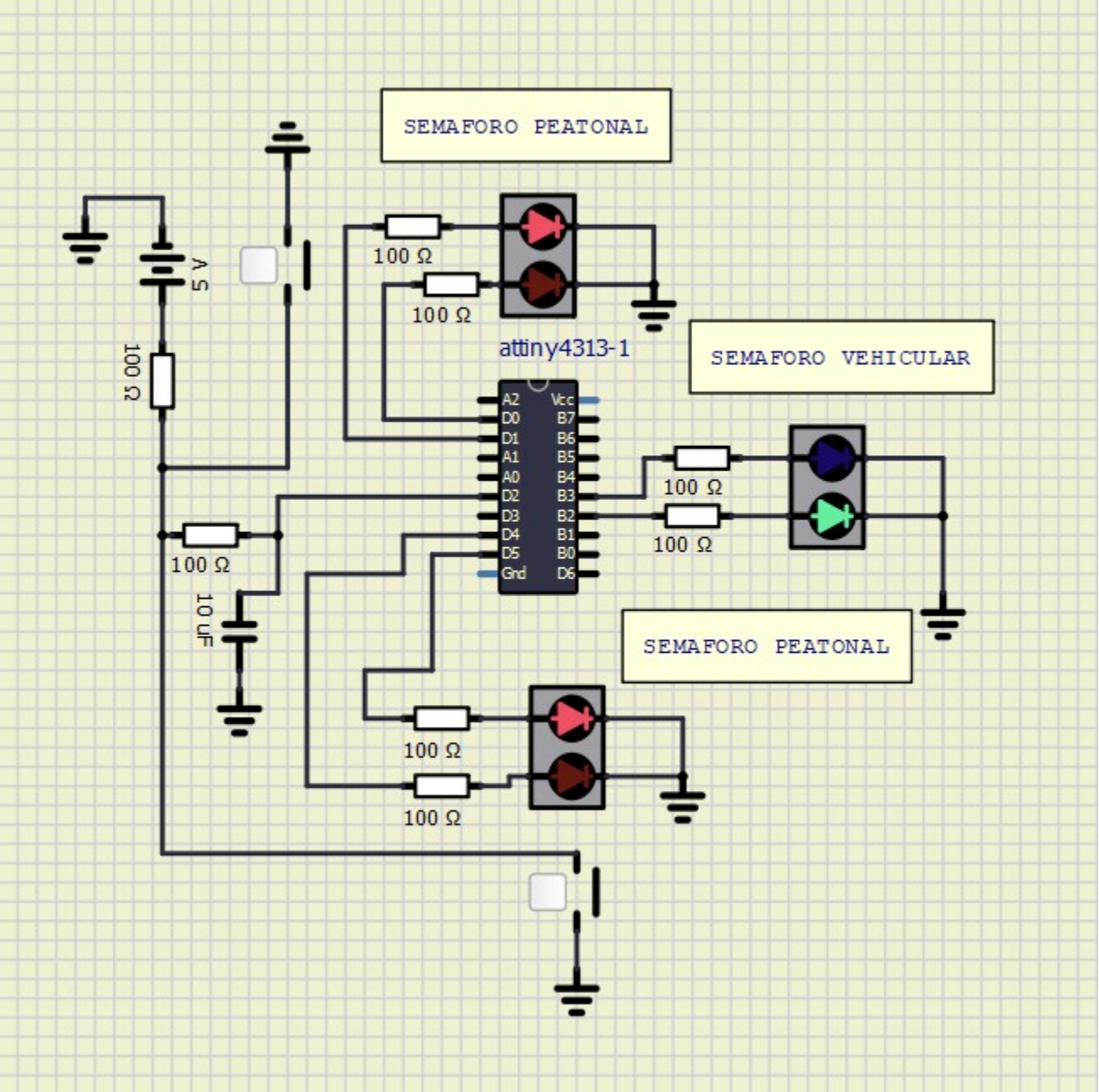


Figura 5: Diseño del circuito

Para la construcción del circuito primero debemos tomar en cuenta las máximas capacidades de tensión y corriente que se pude proporcionar a los pines, con el fin de una manipulación correcta. De acuerdo con la hoja del fabricante,

22.1 Absolute Maximum Ratings*

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on any Pin except $\overline{\text{RESET}}$ with respect to Ground	-0.5V to V_{CC}+0.5V
Voltage on $\overline{\text{RESET}}$ with respect to Ground	-0.5V to +13.0V
Maximum Operating Voltage	6.0V
DC Current per I/O Pin	40.0 mA
DC Current V_{CC} and GND Pins	200.0 mA

Figura 6: Especificaciones máximas para el ATtiny 4313[1]

Podemos notar que la corriente máxima que se le puede inyectar a los pines es de 40mA, por lo tanto utilizando dos resistencias de 100Ω obtenemos una corriente que no causará problema.

$$R = \frac{5V}{200\Omega} = 25mA$$

También un efecto que olvidé tomar en cuenta es el efecto rebote que pueden causar los interruptores, por lo que añadí un capacitor de $100\mu F$ y así evitar las falsas lecturas. Después de esto es una simple conexión de un pin configurado de salida a los leds de cada uno de los tres semáforos, con una resistencia de 100Ω para reducir la corriente. A continuación se presentan los costos de la construcción del circuito:

Componente	Precio (¢)
ATtiny 43-13	945
8 Resistencias	360
6 LED	1200
Total	2505

Cuadro 1: Tabla de valor de componentes

3.2. Código implementado

El código se basa en el siguiente diagrama de estados:

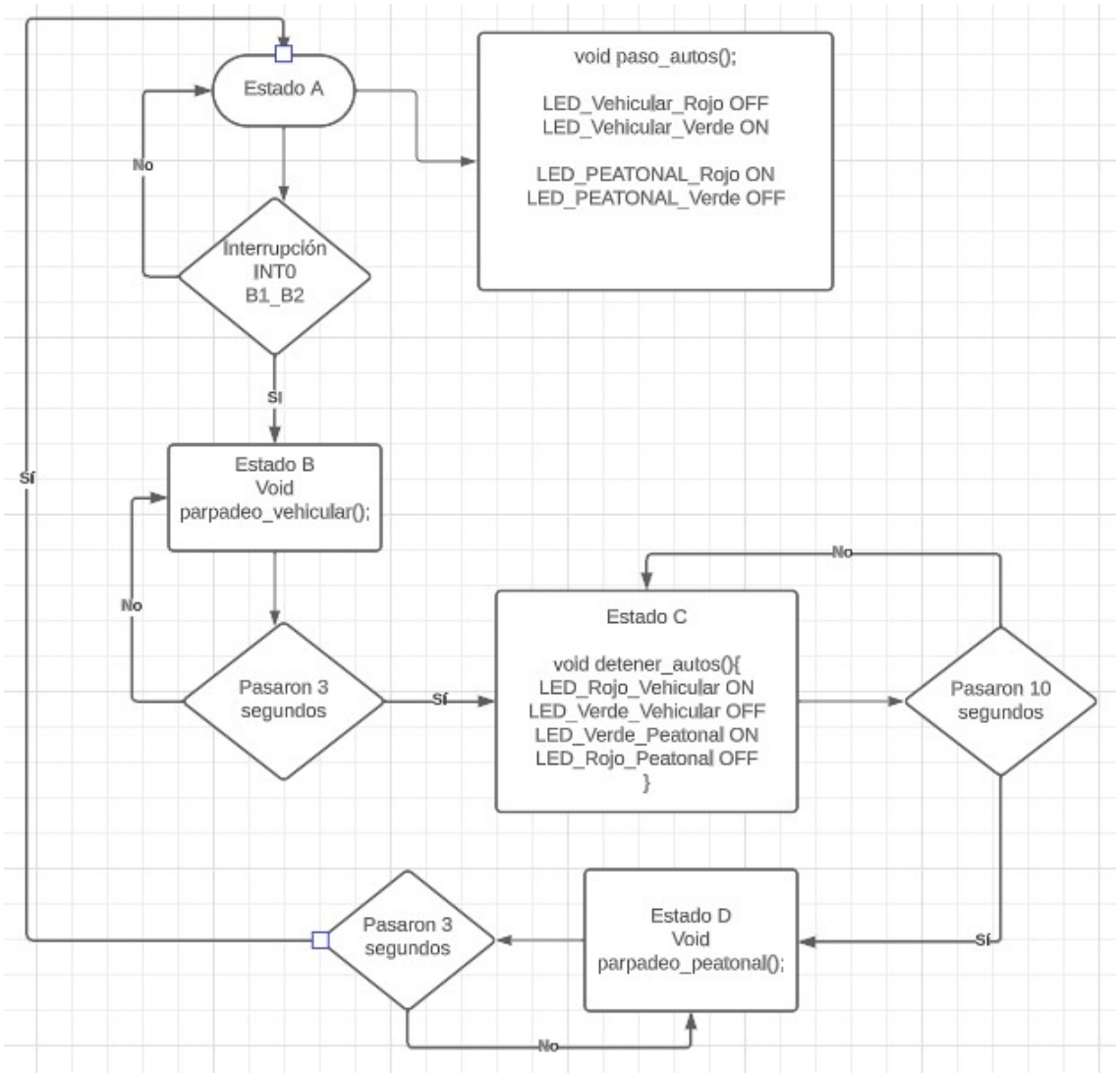


Figura 7: Diagrama de flujo semaforo.c

Ahora se procederá a explicar los puntos más importantes de configuración. Primero, importamos la librería general de AVR, con el fin poder utilizar las macros que controlan el microcontrolador, también la librería de delay que nos permite realizar puntos de detenimiento en el código en escala de ms, por último la librería de AVR específica para el manejo de interrupciones. Despues de esto, para una mayor facilidad de manipulación agrego macros que asignar los pines conectados a los LEDS, y defino cuatro estados para la FSM que se puede apreciar mejor su funcionamiento en la figura 7. Finalmente, asignó las variables necesarias para la lógica y manipulación.

```
1 #include <avr/io.h>
2 #include <util/delay.h>
3 #include <avr/interrupt.h>
4
5 #define VEHICULAR_ROJO PINB3
```

```
#define VEHICULAR_VERDE PINB2
#define PEATONAL_A_ROJO PIND1
#define PEATONAL_A_VERDE PIND0
#define PEATONAL_B_ROJO PIND5
#define PEATONAL_B_VERDE PIND4

#define A 1
#define B 2
#define C 3
#define D 4

int B1_B2;
int state;
int timer_counter;
int ten_sec_c;
int three_sec_c;
int ten_sec;
int three_sec;

```

Ahora, se realizaron funciones que me permiten manipular los LEDS a conveniencia según el estado en el que se encuentra, lo anterior se hace mediante enmascarado de bits.

```
void paso_autos() {
    PORTB &= ~(1 << VEHICULAR_ROJO);
    PORTB |= (1 << VEHICULAR_VERDE);
    PORTD &= ~(1 << PEATONAL_A_VERDE) | ~(1 << PEATONAL_B_VERDE);
    PORTD |= (1 << PEATONAL_A_ROJO) | (1 << PEATONAL_B_ROJO);
}

void detener_autos() {
    PORTB |= (1 << VEHICULAR_ROJO);
    PORTB &= ~(1 << VEHICULAR_VERDE);
    PORTD &= ~(1 << PEATONAL_A_ROJO) & ~(1 << PEATONAL_B_ROJO);
    PORTD |= (1 << PEATONAL_A_VERDE) | (1 << PEATONAL_B_VERDE);
}

void parpadeo_vehicular() {
    PORTB ^= (1 << VEHICULAR_VERDE);
}

void parpadeo_peatonal() {
    PORTD ^= (1 << PEATONAL_A_VERDE);
    PORTD ^= (1 << PEATONAL_B_VERDE);
}

void reinicio() {
    PORTB &= ~(1 << VEHICULAR_ROJO) & ~(1 << VEHICULAR_VERDE);
    PORTD &= ~(1 << PEATONAL_A_ROJO) & ~(1 << PEATONAL_A_VERDE) & ~(1 <<
   PEATONAL_B_ROJO) & ~(1 << PEATONAL_B_VERDE);
}

```

Para la configuración de las interrupciones realizó el enmascarado de los registros GIMSK y MCUCR como se explica en la nota teórica. Por otra parte, para la configuración de timer el punto más importante a tener en consideración es la configuración de registro TCCROB, que se utiliza para realizar un preescalado del reloj principal del microcontrolador. Este preescalado permite el paso del reloj de 16MHz a 16KHz.

```
void external_interrupt() {
    GIMSK |= (1 << INT0); // Habilitar la INT0 (interrupcion externa)
    MCUCR |= (1 << ISC01); // Configurar como flanco descendente
}

void timer_config() {
    TCCR0A = 0x00; // Modo normal
    TCCR0B = 0x00;
    TCCR0B |= (1 << CS00) | (1 << CS02); // Prescaler de 1024
    sei();
    TCNT0 = 0;
    TIMSK |= (1 << TOIE0);
    // Habilitar la interrupcion del timer0
}

```

Aquí, se configura el uso de las interrupciones, en el caso de la interrupción interna 0 cada vez que se accione será cual el interruptor del semáforo peatonal sea accionado. Por otra parte, para el uso del timer se establecen dos conteos de importancia uno para los diez segundos entre el estado B y C, y otro de tres segundos entre los estados A con B y C con D. Para el cálculo del desborde del contador se realizó lo siguiente, dado que se utiliza un contador de 8 bits se puede tener un valor maximo de hasta 255,

$$\frac{1}{16kHz} = 6{,}2\mu s.(255) = 15{,}81ms$$

$$\frac{1}{15{,}81ms} = 63{,}2$$

Por lo tanto, el contador necesita al rededor de 63 desbordes para contar un segundo.

```
ISR(INT0_vect) // Interrupt service routine
{
    B1_B2 = 1;
}

ISR(TIMER0_OVF_vect) {
    if (timer_counter == 63){
      timer_counter = 0;
      ++three_sec_c;
      ++ten_sec_c;
    } else timer_counter++;

    if (ten_sec_c >= 10){
      ten_sec = 1;
    }

    if (three_sec_c >= 3){
      three_sec = 1;
    }
  }

```

Finalmente, la función principal, que simplemente llama a las configuraciones de interrupciones, configura los pines de entrada y salida. Además, establece como estado inicial del programa el estado A, y pone las variables necesarias en 0. Por último, utiliza un bucle infinito con la máquina de estados descrita en la figura 7

```
int main() {
```

```
    reinicio();
    external_interrupt();
    timer_config();

    DDRB |= (1 << VEHICULAR_ROJO) | (1 << VEHICULAR_VERDE); // Configurar B2 y B3 como salidas
    DDRD |= (1 << PEATONAL_A_ROJO) | (1 << PEATONAL_A_VERDE) | (1 << PEATONAL_B_ROJO) | (1 << PEATONAL_B_VERDE);

    state = A;
    B1_B2 = 0;

    while (1) {
      switch (state)
      {
      case A:
        paso_autos();
        if(B1_B2 == 1){
          timer_counter = 0;
          three_sec_c = 0;
          three_sec = 0;
          state = B;
        }
        break;

      case B:
        parpadeo_vehicular();
        _delay_ms(500);
        if(three_sec == 1){
          PORTB &= ~(1 << VEHICULAR_VERDE);
          PORTB |= (1 << VEHICULAR_ROJO);
        if(three_sec_c == 4){
          timer_counter = 0;
          ten_sec_c = 0;
          ten_sec = 0;
          state = C;
        }
        }
      break;

      case C:
        reinicio();
        detener_autos();
        if(ten_sec == 1){
          reinicio();
          timer_counter = 0;
          three_sec_c = 0;
          three_sec = 0;
          state = D;
          B1_B2 = 0;

        }
      break;

      case D:
        parpadeo_peatonal();
        _delay_ms(500);
        if(three_sec == 1){
          PORTD |= (1 << PEATONAL_A_VERDE);
```

```
59            PORTD |= (1 << PEATONAL_B_VERDE);
60            PORTD &= ~(1 << PEATONAL_A_ROJO);
61            PORTD &= ~(1 << PEATONAL_B_ROJO);
62          if(three_sec_c == 4){
63            reinicio();
64            state = A;
65          }
66          }
67        break;
68
69        default:
70          state = A;
71          break;
72        }
73      }
74  }
75
```

Para visualizar la funcionabilidad completa del circuito puede dirigirse al siguiente link.

4. Observaciones y Recomendaciones

- Es importante realizar un diseño antes de construir una implementación como esta, debido a que brinda una idea más clara de la relación de la entrada y salida y que complementos pueden llegar a surgir.
- El estudio de la hoja del fabricante es de vital importancia, dado que nos da las referencias de parámetros adecuados de manipulación de las entradas y salidas del microcontrolador, así como características importantes como funcionamiento de los registros.
- El uso de simuladores para realizar el aprendizaje y primeros diseños da una perspectiva real de como se construye un sistema, sin poner el riesgo el equipo.
- Las interrupciones de un microcontrolador funcionan para detener o realizar una acción ante un cambio en el comportamiento de entrada de un PIN.
- El uso de los temporizadores y contadores del microcontrolador funcionan para poder escalar los tiempos en unidades reales de tiempo.
- Para acceder al repositorio del laboratorio puede utilizar el siguiente enlace a oscarfc164/IE0624

Referencias

[1] A. Corporation, *8-bit AVR Microcontroller with 2/4 Bytes In-System Programmable Flash*. 2011.