Laboratorio #2

Gabriel Fernando Araya Mora
B80525
gabriel.arayamora@ucr.ac.cr

GPIOs, Timers y FSM

2021-09-16

Laboratorio de Microcontroladores
(IE-0624)

1. Introducción:

Para esta práctica de laboratorio se implementa un semáforo digital por medio de un microcontrolador AT-tiny4313. Para hacer esto posible se diseña una máquina de estados con un bus de salida de 5 bits para poder tener las combinaciones necesarias de luces, y un bus de 1 bit para la entrada, el cual simboliza que alguien presionó el botón de peatones o no. Además, tiene la funcionalidad de contar con un diagrama de temporización, es decir, los estados cambian bajo tiempos definidos por el diseñador. Por último con el fin de eliminar el factor del "bouncing" del botón de entrada se utiliza un filtro pasivo hecho con una resistencia y un capacitor.

2. Nota Teórica:

2.1. AT-tiny4313

El AT-tiny4313 es un microcontrolador el cual cuenta con un procesador de tipo RISC (Reduced instruction set) avanzado, además cuenta con un reloj interno de hasta 20MHz y dos contadores, así como con la característica de las interrupciones internas y externas. La característica que diferencia a este microcontrolador del usado en la práctica anterior, es su memoria, para este caso no se tuvo que replantear la implementación para disminuir la cantidad de instrucciones.
En cuanto a los periféricos, este microcontrolador cuenta con 18 pines en entrada/salida completamente programables (GPIOs), los cuales están divididos en tres puertos, que pueden tener o no caracteristicas especiales si se programan de cierta forma.

Figure 1-1. Pinout ATtiny2313A/4313

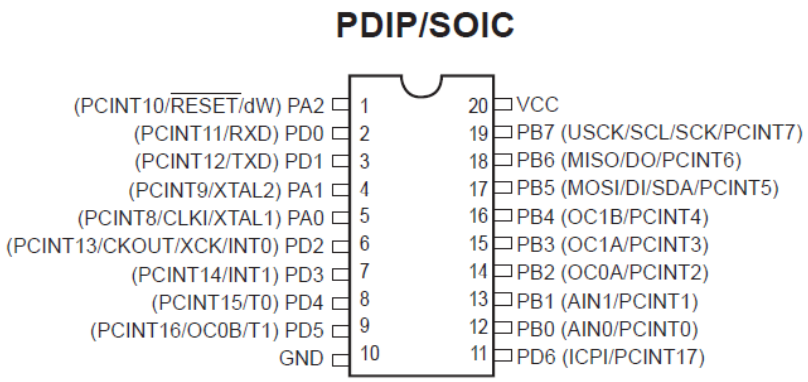


Figura 1: Esquema general del AT-tiny4313. [Inc(2011)]

2.1.1. Memoria y registros en el AT-tiny4313

Este microcontrolador cuenta con una memoria flash de $4K$ Bytes, una EEPROM de 256 Bytes, y por último una RAM de 256 Bytes. Este controlador además tiene 32×8 registros multi propósito.

2.1.2. Registro DDnx

Con este registro se le dice al microcontrolador cual va a ser la configuración de entrada/salida. Es decir mediante este registro se le puede decir al microcontrolador cuales pines van a ser entradas y cuales van a ser salidas. [Inc(2011)]

10.3.6 DDRB – Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x17 (0x37)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
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 2: Configuración general del registro DDRB. [Inc(2011)]

Como se muestra en la figura anterior, es un registro de 8bits al menos para el puerto B que es el que se piensa usar para este laboratorio. Un ejemplo de esto se muestra en seguida:

```
void main(){
    // Se designa PB0 como salida y PB1 como entrada
   DDRB = (1<<DDB0)|(0<<DDB1);
}
```

2.1.3. Puerto B

Este registro sirve para decirle al microcontrolador si queremos que el pin designado en el registro DDRB, se encuentre encendido o apagado.

Port B (PB7..PB0)

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Figura 3: Configuración general del puerto B. [Inc(2011)]

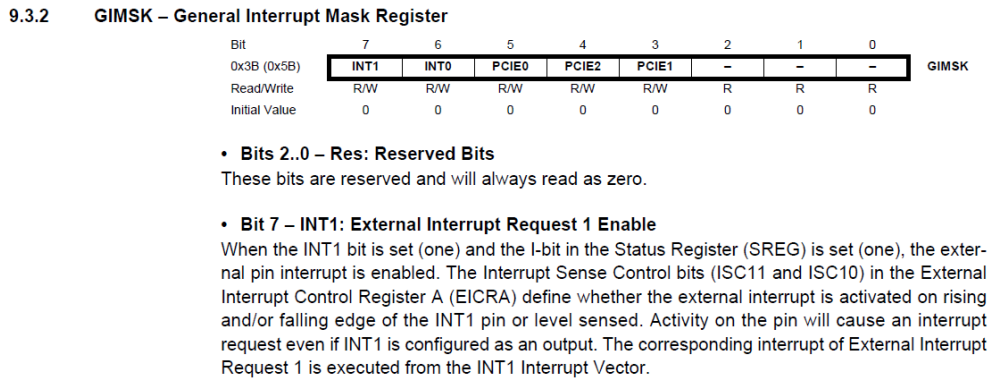
Un ejemplo de cómo se pueden encender o apagar pines se muestra en el siguiente código:

```
void main(){
    //PB0 encendido y el resto apagados
   PORTB = (1<<PB0)|(0<<PB1)|(0<<PB2)|(0<<PB3)|(0<<PB4);
}
```

Esto es especialmente útil, ya que se sabe que la corriente viaja de mayor potencial a menor potencial, y por ende se puede pensar en alguna configuración en que la corriente vaya de los pines hasta alguna tierra, pero esto ya depende del diseño físico. [Inc(2011)]

2.1.4. Interrupciones

Una interrupción se define como un evento el cual detiene el flujo de instrucciones en el procesador, con el fin de atender y resolver dicho evento, hecho esto el procesador procede a continuar donde dejó el programa antes de que ocurriera el evento gatillo. Es decir guarda el PC anterior antes de saltar a la interrupción



9.3.2 GIMSK – General Interrupt Mask Register

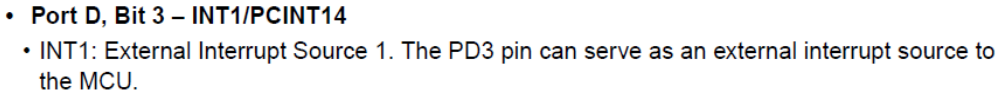
Bit	7	6	5	4	3	2	1	0	
0x3B (0x5B)	INT1	INT0	PCIE0	PCIE2	PCIE1	–	–	–	GIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

• **Bits 2..0 – Res: Reserved Bits**
These bits are reserved and will always read as zero.

• **Bit 7 – INT1: External Interrupt Request 1 Enable**
When the INT1 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled. The Interrupt Sense Control bits (ISC11 and ISC10) in the External Interrupt Control Register A (EICRA) define whether the external interrupt is activated on rising and/or falling edge of the INT1 pin or level sensed. Activity on the pin will cause an interrupt request even if INT1 is configured as an output. The corresponding interrupt of External Interrupt Request 1 is executed from the INT1 Interrupt Vector.

Figura 4: Configuración general del registro de interrupciones. [Inc(2011)]

Como se muestra en la figura anterior, por medio del registro GIMSK se puede definir que el pin INT1 reconozca interrupciones externas como el botón del semáforo peatonal. Esto se puede activar ya sea en el flanco positivo, negativo o bien cuando ocurre un cambio de nivel. Para hacer esto se usan los registros ISC11 y SC10.

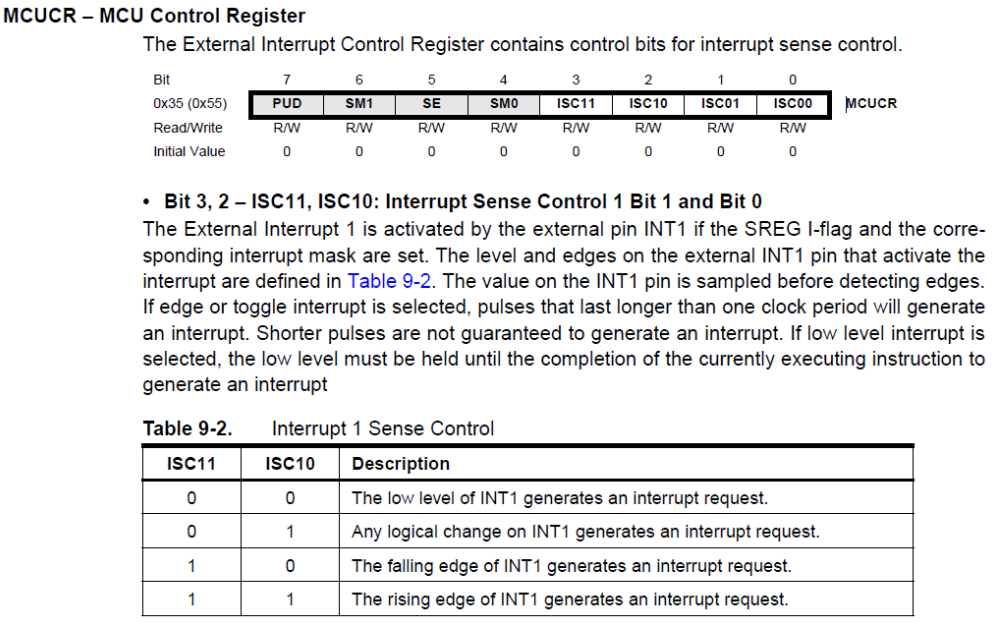


• **Port D, Bit 3 – INT1/PCINT14**
• INT1: External Interrupt Source 1. The PD3 pin can serve as an external interrupt source to the MCU.

Figura 5: INT1 como el pin de interrupciones externas. [Inc(2011)]

2.1.5. Registro MCUCR

Este registro controla las interrupciones externas y además se le dice si se quiere que revise la interrupción en flanco positivo, negativo o cuando haya un cambio de nivel. Para hacer esto se usan los registros ISC11 y ISC10. Las combinaciones se muestran en la siguiente tabla. Además se muestra un ejemplo de cómo configurar la interrupción externa en INT1.



MCUCR – MCU Control Register

The External Interrupt Control Register contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	PUD	SM1	SE	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
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	

• **Bit 3, 2 – ISC11, ISC10: Interrupt Sense Control 1 Bit 1 and Bit 0**
The External Interrupt 1 is activated by the external pin INT1 if the SREG I-flag and the corresponding interrupt mask are set. The level and edges on the external INT1 pin that activate the interrupt are defined in Table 9-2. The value on the INT1 pin is sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt

Table 9-2. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

Figura 6: Control de interrupciones. [Inc(2011)]

```
void main(){
    GIMSK |= (1<<INT1);      // habilitando en INT1 (external interrupt)
    MCUCR |= (1<<ISC11);     // se configura con flanco negativo del reloj
}
```

2.1.6. Temporizadores

Este microcontrolador cuenta con dos contadores, los cuales se pueden usar para contar el paso del tiempo. Para el semáforo se va a utilizar el modo normal de los contadores, para esto es cuestión de iniciarlos en cero.

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 7: Registro de temporización. [Inc(2011)]

Consiguientemente se debe seleccionar los relojes para hacer la escala y saber cuanto tiempo se cuenta.

Table 11-9. Clock Select Bit Description

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 8: Tabla de configuración del temporizador. [Inc(2011)]

Por último se va a activar el temporizador por interrupción, para lo cual se usa el siguiente registro TIMSK:

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	

- **Bit 4 – Res: Reserved Bit**

This bit is reserved bit in the ATtiny2313A/4313 and will always read as zero.

- **Bit 2 – OCIE0B: Timer/Counter0 Output Compare Match B Interrupt Enable**

When the OCIE0B bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter Compare Match B interrupt is enabled. The corresponding interrupt is executed if a Compare Match in Timer/Counter occurs, i.e., when the OCF0B bit is set in the Timer/Counter Interrupt Flag Register – TIFR.

- **Bit 1 – TOIE0: Timer/Counter0 Overflow Interrupt Enable**

When the TOIE0 bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter0 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter0 occurs, i.e., when the TOV0 bit is set in the Timer/Counter 0 Interrupt Flag Register – TIFR.

Figura 9: Registro de activación de interrupción por temporizador. [Inc(2011)]

Esto permite generar una interrupción interna cuando se da un overflow del contador para un objetivo dado; es decir, si el contador se pasa de un parámetro dado entonces se dispara una interrupción la cual se debe atender para continuar con el programa.

```
void(main){
    TCCR0A=0x00;   //Se usa el modo normal de operacion del timer
    TCCR0B=0x00;
    TCCR0B |= (1<<CS00)|(1<<CS02);   //prescaling usando el 1024
    sei(); // Se encienden las interrupciones globales
    TCNT0=0;
    TIMSK|=(1<<TOIE0); //habilitando la interrupcion en TOIE0
}
```

2.1.7. Subrutinas de atención de interrupciones

En secciones anteriores se abordó cómo habilitar las interrupciones globales, externas y por contador, pero no se ha mostrado cómo decirle al procesador qué hacer cuando se topa con una interrupción. En la librería interrupt.h del AT-tiny4313 existe una función especial que se encarga de tomar las interrupciones y resolverlas. La función se llama ISR (Interrupt Service Routine), la cual recibe un vector dependiendo de a qué interrupción se haga referencia. A continuación se muestra un ejemplo de estas funciones y cómo programarlas.

```
ISR (INT1_vect){        // Interrupt service routine
    boton = ON;
}

ISR (TIMER0_OVF_vect){      //Interrupt vector for Timer0
    if (intr_count == 60){    // cuenta un segundo
        if( estado == BVR ){
            (fsm[BVR].stateptr)(); // parpadear
         }
        else if( estado == BRV ){
            (fsm[BRV].stateptr)(); // parpadear
         }
    intr_count = 0;
    ++sec; // cuenta un segundo
  } else intr_count++;
}
```

En el código anterior se muestra cómo se atiende una interrupción exterior en el pin INT1, para este caso específico, el evento gatillo puede ser presionar un botón, y la subrutina de interrupciones actualiza una variable interna del procesador.
En el caso de la subrutina de atención para el overflow del temporizador, cuando se da una cuenta de un segundo, (que por el prescaling es un 60) se hace un llamado a varias funciones dependiendo del estado de la máquina.

2.2. Características eléctricas del AT-tiny4313

Tomando la hoja de datos del microcontrolador, se obtienen las características eléctricas físicas del mismo. Esto es importante ya que permiten dimensionar resistencias, y componentes periféricos. Todo lo anterior con el fin de proteger los componentes y asegurar el buen funcionamiento del diseño.
Como se ve en la siguiente figura, cada pin del AT-tiny4313 es capaz de entregar 40mA. Con el fin de proteger tanto los LEDs del diseño, y el microcontrolador, se utilizan resistencias de protección, las cuales se dimensionan en la sección siguiente. Además la corriente total que pueden entregar en total lo pines juntos es de 200mA según la hoja de datos, por lo que se debe cuidar que la suma de las corrientes cuando varios pines estén dando una salida no sobrepase los 200mA. [Inc(2011)]

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

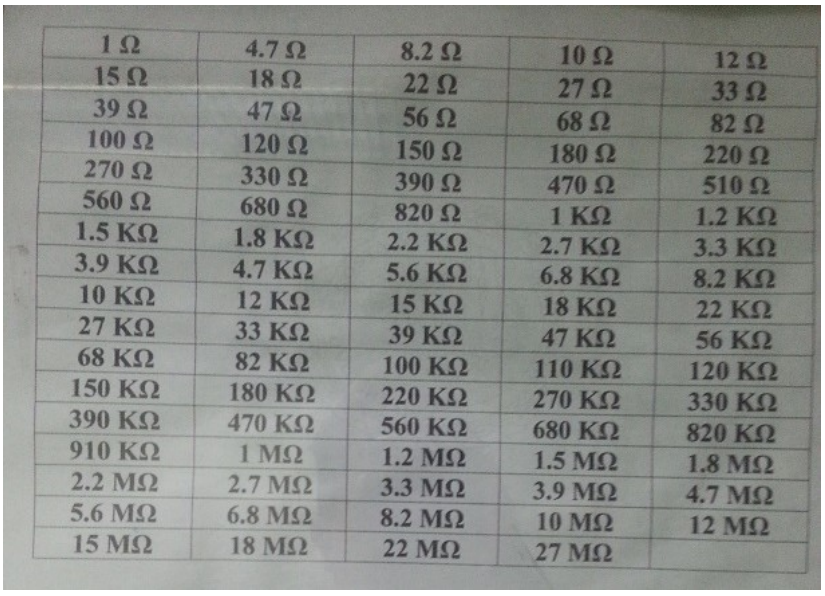
Figura 10: Características eléctricas del microcontrolador en cuestión. [Inc(2011)]

2.3. Dimensionamiento de las resistencias de protección de los LED

Para todo ingeniero eléctrico, técnico en electrónica o entusiasta, es bien sabido que los LED son elementos eléctricos sumamente frágiles y tienden a quemarse por sobre corrientes muy fácilmente. Es por esto que siempre que se tiene un LED, el mismo debe ir acompañado de una resistencia en serie. El valor de esta resistencia va a estar dado por el resto de componentes del circuito. [Dorf and Svoboda(2011)]
Como en este caso se conoce que la corriente máxima que da el microcontrolador es de 40mA, se puede usar este dato para el dimensionamiento de la resistencia. Sin embargo por temas de protección del mismo microcontrolador, nunca se debe apuntar a utilizar el máximo, sino un poco menos, por lo que para este diseño se procurará estar por debajo de los 20mA por pin en todo momento.
Para ahorrar en resistencias, se quiere usar únicamente resistencias de 100Ω y como se quiere una corriente menor a los 20mA, se aplica ley de ohm y se verifica si la tensión obtenida es menor a la máxima que puede dar el microcontrolador.

$$V = IR \longrightarrow V = 20mA \cdot 100 = 2V \tag{1}$$

Como la tensión máxima del microcontrolador es 5V, el MCU es totalmente capaz de entregar 2v por pin.
Como eventualmente este diseño se quiere hacer de forma física, el valor de resistencia de 100Ω, se encuentra en la bodega de la escuela de ingeniería eléctrica.



1 Ω	4.7 Ω	8.2 Ω	10 Ω	12 Ω
15 Ω	18 Ω	22 Ω	27 Ω	33 Ω
39 Ω	47 Ω	56 Ω	68 Ω	82 Ω
100 Ω	120 Ω	150 Ω	180 Ω	220 Ω
270 Ω	330 Ω	390 Ω	470 Ω	510 Ω
560 Ω	680 Ω	820 Ω	1 KΩ	1.2 KΩ
1.5 KΩ	1.8 KΩ	2.2 KΩ	2.7 KΩ	3.3 KΩ
3.9 KΩ	4.7 KΩ	5.6 KΩ	6.8 KΩ	8.2 KΩ
10 KΩ	12 KΩ	15 KΩ	18 KΩ	22 KΩ
27 KΩ	33 KΩ	39 KΩ	47 KΩ	56 KΩ
68 KΩ	82 KΩ	100 KΩ	110 KΩ	120 KΩ
150 KΩ	180 KΩ	220 KΩ	270 KΩ	330 KΩ
390 KΩ	470 KΩ	560 KΩ	680 KΩ	820 KΩ
910 KΩ	1 MΩ	1.2 MΩ	1.5 MΩ	1.8 MΩ
2.2 MΩ	2.7 MΩ	3.3 MΩ	3.9 MΩ	4.7 MΩ
5.6 MΩ	6.8 MΩ	8.2 MΩ	10 MΩ	12 MΩ
15 MΩ	18 MΩ	22 MΩ	27 MΩ	

Figura 11: Resistencias Disponibles en bodega. (Creación Propia)

2.4. Tratando el rebote del botón de entrada

Cuando se enciende o apaga un interruptor, ya sea un switch, o un botón, la salida del mismo no es totalmente limpia, sino que presenta rebotes debido que los materiales físicos rebotan entre ellos. Esto es un problema ya que los rebotes pueden inevitablemente introducir falsas lecturas al sistema. [Boylestad and Nashelsky(2009)]

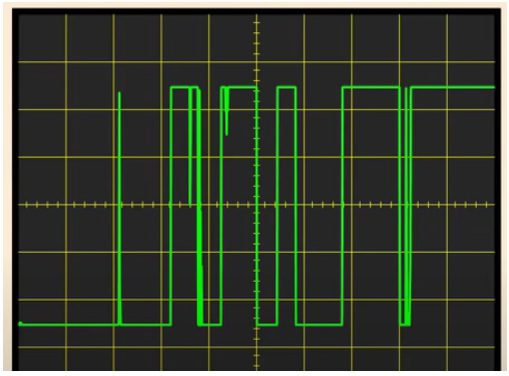


Figura 12: Efecto de rebote de un botón.[Christoffersen(2015)]

Existen diversas formas para contrarrestar este efecto, ya sea por hardware o por software. Para este diseño específico se decidió ir por la solución implementada en hardware, ya que es fácil, eficiente y barata de implementar.
Para esto se aplica un filtro con el fin de eliminar los rebotes del switch.

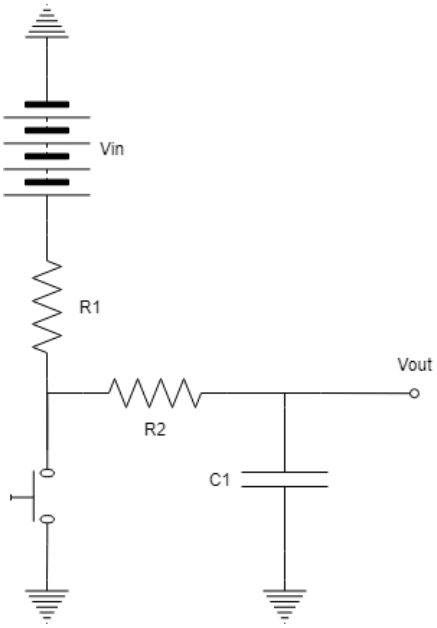


Figura 13: Filtro para eliminar el efecto de rebote del botón. (Creación Propia)

Como no se tiene una entrada sinusoidal, y además no se pretende filtrar ciertas frecuencias, no hay parámetros importantes para dimensionar el capacitor, con solo que se cargue y se descargue en un tiempo prudente es suficiente. Por esto mismo se escoge de los capacitores en bodega, uno de 100pF, lo cual se va a cargar sumamente rápido y lo mismo para la descarga, cumpliendo así el objetivo de filtrar la señal del switch.
Teniendo esto se pueden dimensionar las resistencias teniendo en cuenta el tema del tiempo.

$$\tau = RC \longrightarrow 100\Omega \cdot 100pF \approx 10nS \tag{2}$$

Este valor es lo suficientemente pequeño como para que sea imperceptible por el usuario, además que en el semáforo el tiempo de reacción del botón no importa debido a que los tiempos en que cambian las luces estarán predefinidos en la FSM .

2.5. Finite State Machine

Se define una máquina de estados como un objeto matemático capaz de realizar cálculos sobre una entrada con el fin de producir una salida basado en esos cálculos.

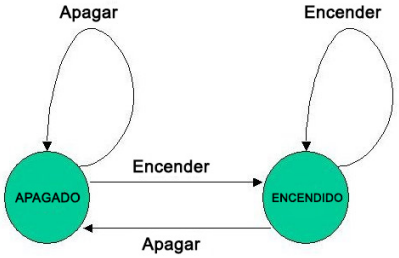


Figura 14: Diagrama FSM genérico. (Creación Propia)

2.5.1. Diseño de la máquina de estados para el semáforo digital

- **Entrada:** Si bien es cierto que se tienen dos botones, la entrada no tiene sentido que sea de 2 bits, ya que si se presionan los botones al mismo tiempo, debería pasar de estado, al igual que si se presionara un único botón. Por lo que la entrada es de un solo bit para evitar redundancias y eléctricamente se conectan los dos botones al mismo nodo, por lo que si se acciona el uno o el otro va a resultar en una entrada de 1 bit.

Entrada	Significado
0	No hay peatones
1	Una persona presionó el botón.

Cuadro 1: Codificación para el botón de entrada. (Creación Propia)

- **Salida:** Eléctricamente se tienen 7 leds; sin embargo, los semáforos peatonales cuentan con sus LEDs en paralelo, ya que ambos son iguales y deben cambiar de estado al mismo tiempo. Por lo que entonces el vector de salida será de 5 bits y se codifica de acuerdo a la siguiente tabla:

Carros			**Peatones**	
b1	**b2**	**b3**	**b4**	**b5**
Verde Carros	**Amarillo Carros**	**Rojo Carros**	**Rojo Peatones**	**Verde Peatones**

Cuadro 2: Codificación de la salida de la máquina de estados. (Creación Propia)

- **Estados:** En la siguiente tabla se muestran los estados codificados con su respectiva salida y siglas:

Estados	Siglas	Salidas
Carroz avanzando	CA	10010
Blink Verde-Red	BVR	10010/00000
Stop/Slow Carros	SC	01010
Peatones Avanzando	PA	00101
Blink Red-Verde	BRV	00101/00000

Cuadro 3: Tabla de codificación de estados. (Creación Propia)

- **Transición de estados:** En la siguiente tabla se muestra la tabla de transición de estados según las entradas:

Estados	0	1
Carroz avanzando	CA	BVR
Blink Verde-Red	SC	SC
Stop/Slow Carros	PA	PA
Peatones Avanzando	BRV	BRV
Blink Red-Verde	CA	CA

Cuadro 4: Tabla de transición de estados. (Creación Propia)

2.6. Diseño Final del semáforo digital

Tomando en consideración todas las secciones anteriores, se puede crear un diseño funcional para lo solicitado en el enunciado del laboratorio. Tomando en cuenta que se quiere simular un semáforo, es necesario utilizar 7 LEDs, y 7 resistencias para los mismos. Tres LEDs y tres resistencias van a simular el semáforo vehicular, mientras que los otros 4 están en paralelo para simular los semáforos peatonales. Los botones tienen su respectivo filtro para eliminar el rebote de los mismos.

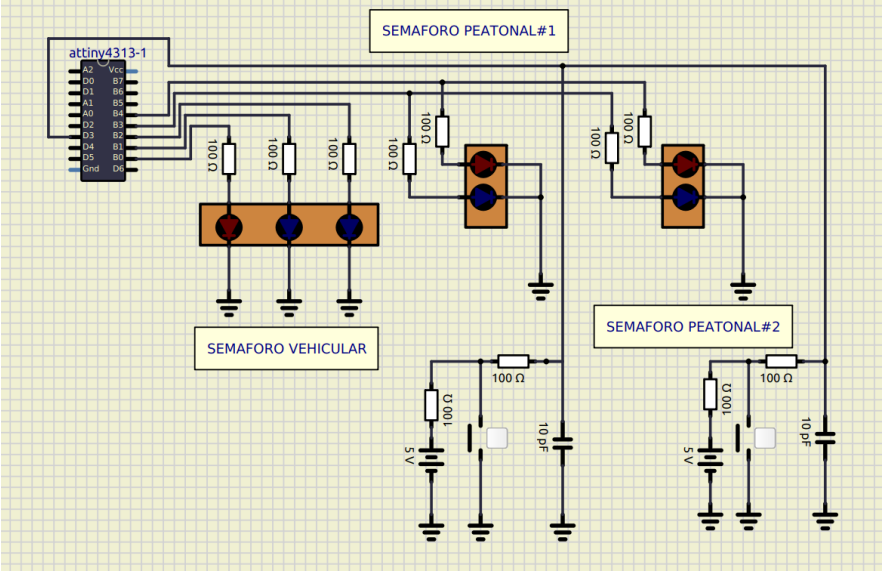


Figura 15: Esquemático del circuito montado. (Creación Propia)

Los componentes del filtro de entrada son los mismos discutidos en la sección anterior, dos resistencias, un botón, la batería y el capacitor.
Buscando vendedores de los componentes anteriormente mencionados, se tiene el problema de que no venden la cantidad exacta de componentes; sin embargo esto no presenta un problema debido a que un paquete con 100 resistencias cuesta alrededor de 6 dólares, lo que sí sube considerablemente el precio del proyecto, pero sigue siendo un proyecto que sale por aproximadamente 11 mil colones. (Las capturas de los precios se presentan en la sección de anexos)

Componente	**Cantidad**	**Precio**
AT-Tiny4313	**1**	**$ 15**
100Ω Resistor	**11**	**$1.20**
LED	**7**	**$ 5.7**
TOTAL	**$ 22**	

Cuadro 5: Tabla resumen de la cantidad de componentes y sus precios.

3. Análisis de resultados:

Primeramente se muestran los resultados de los diferentes estados principales del semáforo:

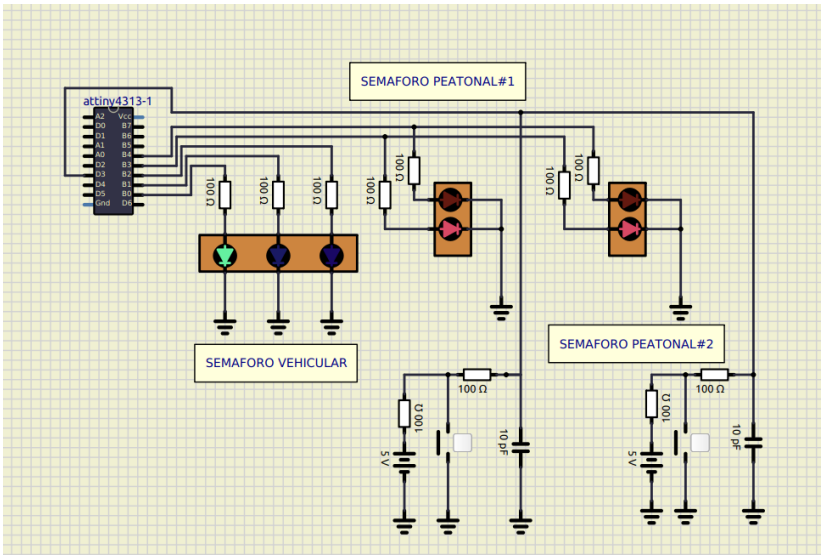


Figura 16: Carros avanzando. (Creación Propia)

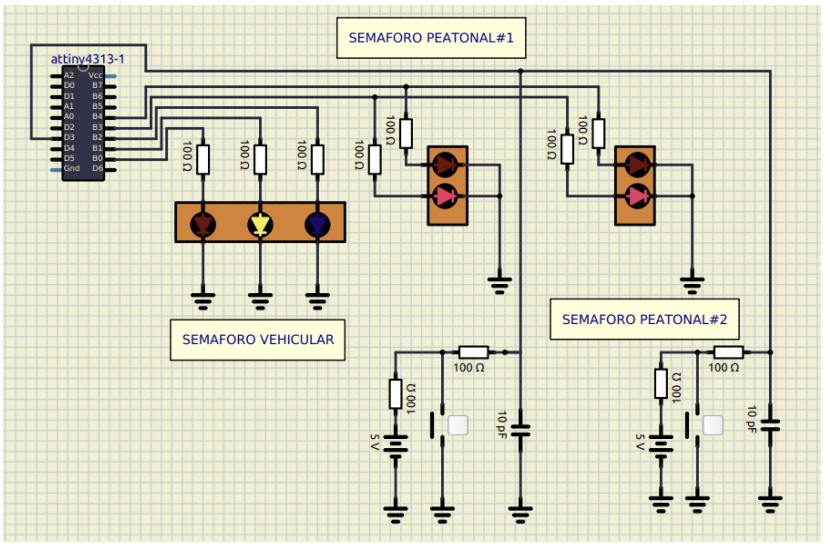


Figura 17: Stop carros (Luz amarilla). (Creación Propia)

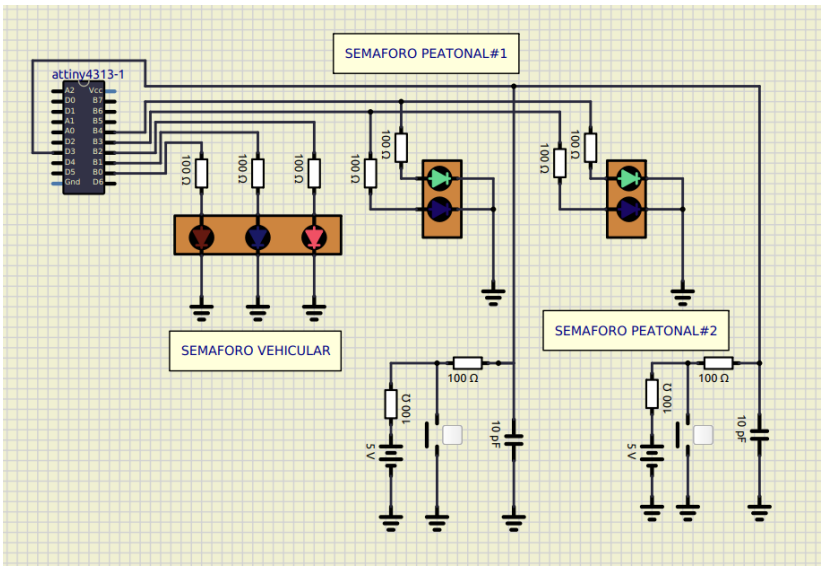


Figura 18: Carros detenidos/ peatones avanzando. (Creación Propia)

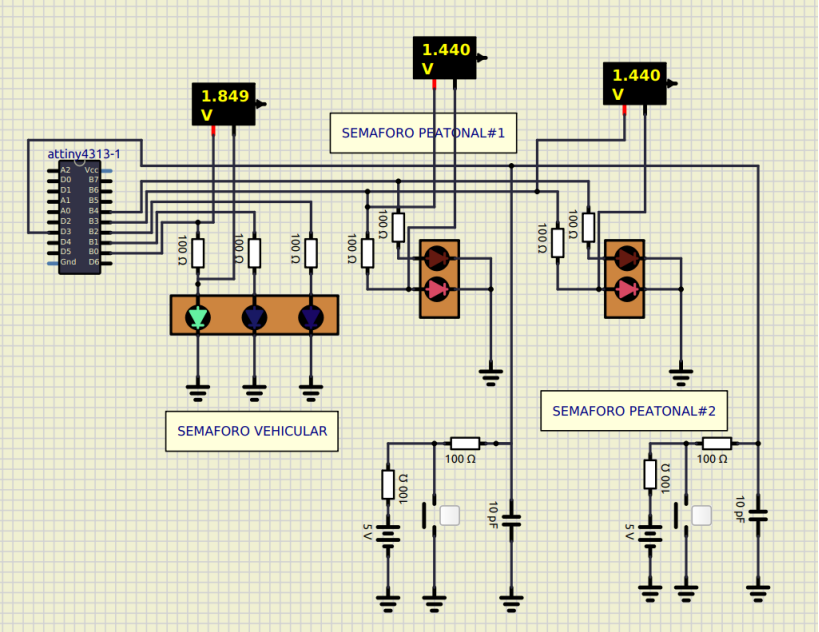


Figura 19: Medición de tensión en los pines de salida. (Creación Propia)

Como se muestra en la figura anterior, se obtuvo una representación numérica de la tensión en la resistencia de los leds del semáforo. En los semáforos peatonales hay una menor tensión debido a que se tienen dos LEDs en paralelo, por lo que se divide la corriente, y se tiene un poco menos; sin embargo, no es significativo.

$$V = IR \longrightarrow I = \frac{1{,}849}{100} \approx 18{,}49mA$$

$$V = IR \longrightarrow I = \frac{1{,}440}{100} \approx 14{,}4mA$$

Estos resultados concuerdan con lo esperado ya que se diseñó para tener una corriente menor a 20mA y se tienen estos resultados.
El comportamiento del filtro se comprobó en el laboratorio pasado, y en este se reutilizó el mismo, con los mismos componentes con el fin de ahorrar tiempo y además es algo que ya sirve de buena manera.

3.1. Código

Para implementar la máquina de estados se hace uso de una estructura especial de C, el *struct*, en el cual se definen todos los atributos que tiene dicha máquina, es decir la máquina implementada cuenta con tres atributos fundamentales, primero, cuenta con un puntero a la función de estado, las cuales se encargan de dar la salida al sistema. Después cuenta con el atributo del tiempo, como se quiere que cada estado sea efectivo en ventanas de tiempo distintas, quiere decir que el tiempo es un atributo que acompaña a cada estado y por último se debe codificar el vector de estados siguientes.

```
struct FSM{
     void (*stateptr)(void);
     unsigned char time;
     unsigned char next[2];
 };
 // Se hace una instancia de la máquina
 semaforo fsm[5] = {
     {&CarrosAvance,10,{CA,BVR}},
     {&BlinkVerdeRed,4,{SC,SC}},
     {&StopCarros,2,{PA,PA}},
     {&PeatonesAvance,10,{BRV,BRV}},
     {&BlinkRedVerde,4,{CA,CA}}
 };
```

Teniendo esto, se definen cada una de las funciones de estado, las cuales como se dijo anteriormente se encargan de encender los leds según la codificación de salida.
Por último dentro del bucle infinito, se debe obtener una lógica que permita cambiar de estado secuencialmente según pase el tiempo de cada estado y se cumpla la condición del botón encendido, además esta lógica debe ser capaz de poner los contadores en cero para reiniciar la cuenta en cuanto se pase de estado.

```
switch (estado){
    case CA:
      (fsm[estado].stateptr)(); // pone las luces  de carros avanzando
      if (boton && sec >= fsm[estado].time){
        estado = fsm[estado].next[boton];
        intr_count = 0;
        sec = 0;
      }
      else{
        estado = CA; // si no hay bonton quedese en el estado de carros avanzando
      }
      break;

    case BVR:
      if (sec == fsm[estado].time){ // despues del tiempo definido pase de estado
        estado = fsm[estado].next[boton];
        intr_count = 0;
        sec = 0;
      }
      else{
        estado = BVR; // si no se cumple la condicion de tiempo siga parpadeando
      }
      break;

      .
      .
      .

    case BRV:
      if (sec == fsm[estado].time){
        estado = fsm[estado].next[boton];
        intr_count = 0;
        sec = 0;
        boton = OFF; // devuelve el sistema a su estado inicial
      }
      else{
        estado = BRV;
      }
      break;

    default:
      break;
  }
}
```

3.2. Temporización

El diagrama de temporización mostrado en el enunciado del laboratorio se modificó un poco, ya que el mismo no cuenta con el LED de color amarillo, entonces se tuvo que modificar el mismo para incorporar el amarillo, además se le agregó un segundo más a los estados de parpadeo con el fin de que el flujo de luces se sintiera más natural y se apreciara mejor el parpadeo de los leds. Además se adjuntan ambas imágenes para poder compararlas.

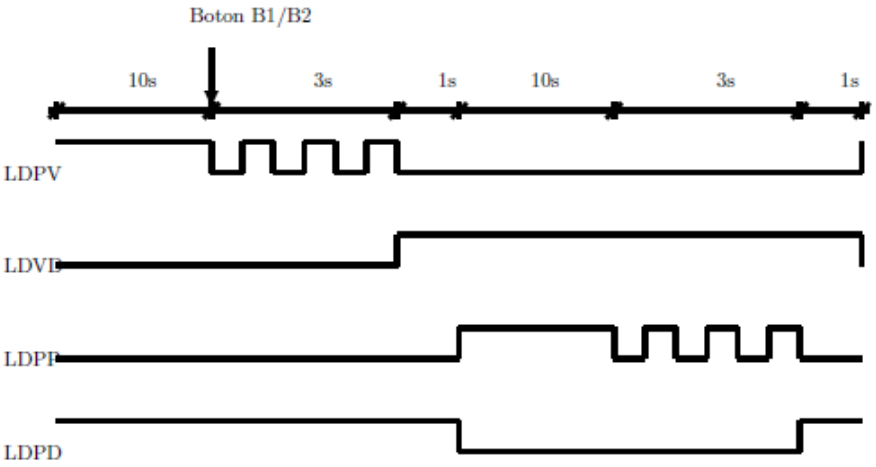


Figura 20: Diagrama de temporización dado en el enunciado.

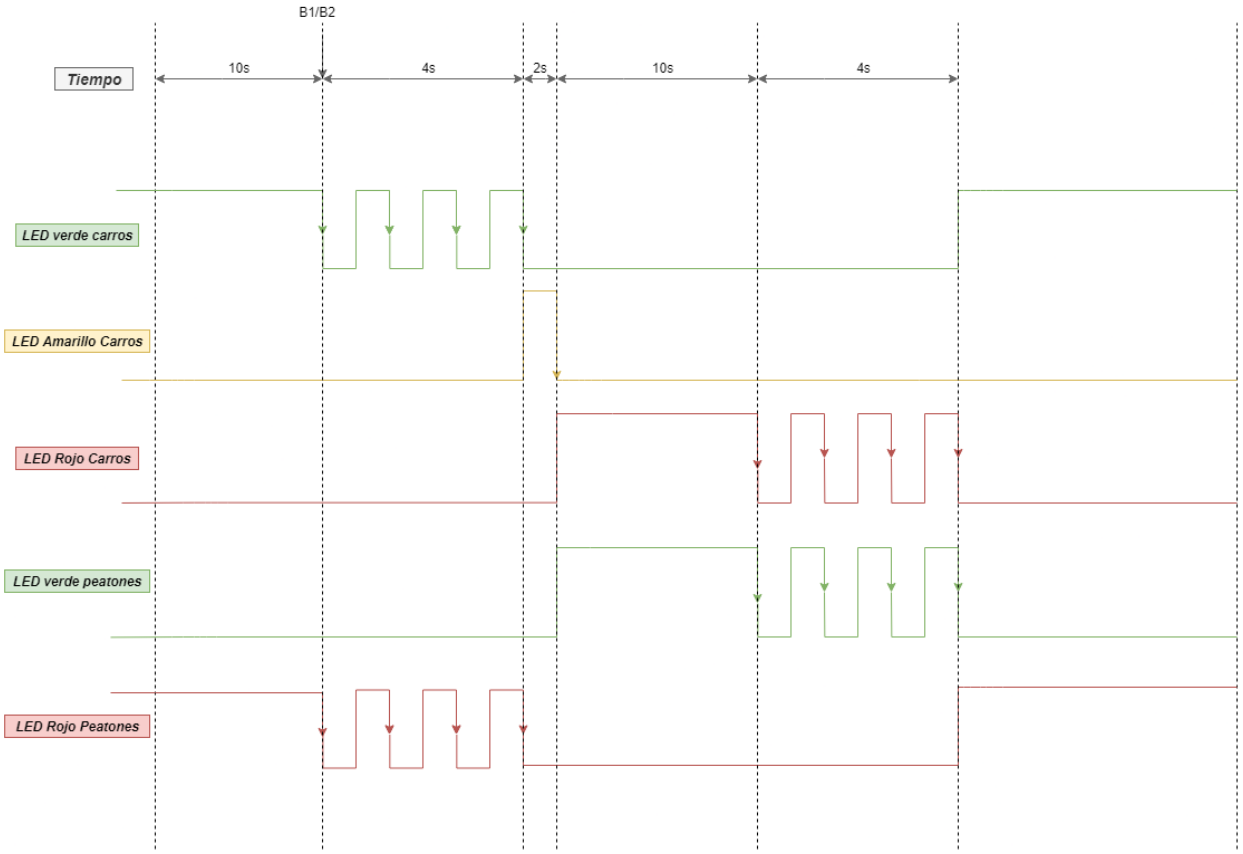


Figura 21: Diagrama de temporización propio. (Creación Propia)

4. Conclusiones y recomendaciones

Primeramente, se cumplió el objetivo de tal manera que el circuito cumple con todas las especificaciones de diseño y sirve bien.
Se tuvieron varios problemas, ya que se estaba usando el puerto INT0 para manejar las interrupciones y el mismo no sirve para el fin de este laboratorio, por lo que entonces se optó por usar el pin INT1 debido a que en la hoja de datos dice explícitamente que el mismo se usa para manejar excepciones externas.
Después, el algoritmo implementado no cuenta con un algoritmo de parpadeo sin delays, por lo que los mismos sirven, y son disparados por interrupciones de tiempo, pero el parpadeo como tal no se logró generar sin usar delays. Por último el timer se maneja con interrupciones lo cual se considera una forma muy elegante de

hacerlo; sin embargo, no es la única y posiblemente tampoco la mejor.
El problema más grande que apareció durante la implementación, fue que se estaban llamando muchas funciones en la función de atención de interrupciones, lo que si bien es funcional, no es para nada una buena práctica debido a que la atención de las interrupciones debe ser lo más rápida y concisa posible, y todas las diferentes implementaciones llevaban a llamar alguna otra función en esta rutina de interrupciones para pasar de estado, por lo que entonces se tomó la alternativa de mover los estados en el bucle de programa directamente, y crear una nueva variable global interna que funcione como bandera para la interrupción, y esta bandera es el evento detonante que mueve la máquina, sin embargo, no se desechó la idea de usar un struct, ya que se consideró que es una forma compacta y ordenada de hacer la FSM, sin embargo, el Switch/Case puede modelar una máquina de estados sin el struct.
A pesar de los problemas que se tuvo con las interrupciones y los bugs que aparecieron en el camino, se pudo depurar lo suficiente el diseño para ser funcional y cumplir las expectativas.
Además se recomienda buscar los componentes en tiendas dentro del país, para ahorrar en envío y demás.

Referencias

[Inc(2011)] Microchip Technology Inc. At-tiny data sheet, 2011.

[Dorf and Svoboda(2011)] R. Dorf and J. Svoboda. *Circuitos Eléctricos*. Alfaomega, 8 edition, 2011.

[Boylestad and Nashelsky(2009)] R. Boylestad and L. Nashelsky. *Electrónica: Teoría de Circuitos y Dispositivos Electrónicos*. Pearson Educación, 10 edition, 2009.

[Christoffersen(2015)] Jens Christoffersen. Switch bounce and how to deal with it. `https://www.allaboutcircuits.com/technical-articles/switch-bounce-how-to-deal-with-it/` , 2015.

[Podkalicki(2018)] Lukasz Podkalicki. Microcontrollers and basic bit-level operations. `https://blog.podkalicki.com/microcontrollers-and-basic-bit-level-operations` , 2018.

5. Anexos

Figura 22: Precio para el MCU

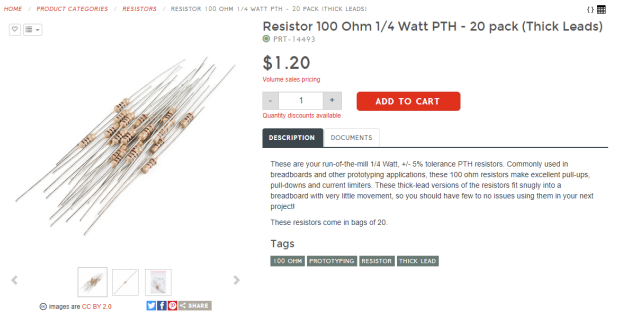


Figura 23: Precio para la resistencia de 100ohms

Figura 24: Precio para los LED

Figura 25: Precio para el capacitor de 100pF