Circuitos Digitales y Microcontroladores

GRUPO 6

Chanquía, Joaquín (02887/7)

Forden Jones, Ian E. W. (02543/3)

Ollier, Gabriel (02958/4)

ÍNDICE

1. INTRODUCCIÓN

1.1 - Enunciado

- Diseño de hardware:

1. Se desea conectar 8 diodos LED de diferentes colores al puerto B del MCU y encenderlos con una corriente de 10mA en cada uno. Realice el esquema eléctrico de la conexión en Proteus 8. Calcule la resistencia serie para cada color teniendo en cuenta la caída de tensión VLED (rojo=1.8V, verde=2.2V amarillo=2.0V azul=3.0V). Verifique que la corriente por cada terminal del MCU no supere la capacidad de corriente de cada salida y de todas las salidas del mismo puerto en funcionamiento simultáneo.

2. Se desea conectar un pulsador a una entrada digital del MCU y detectar cuando el usuario presiona y suelta el pulsador. Muestra el esquema de conexión y determina la configuración del MCU que corresponda. Investigue sobre el efecto de rebote que producen los pulsadores e implemente un método para eliminar este efecto en su algoritmo de detección (puede encontrar información útil en la bibliografía).

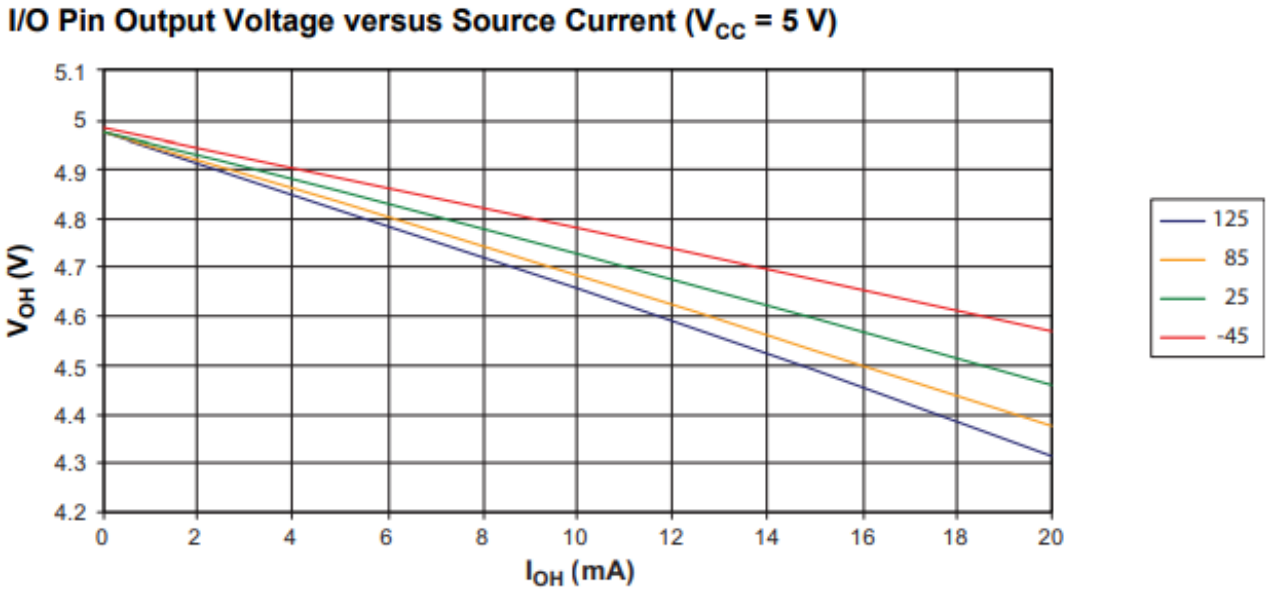
- Diseño de software:

3. Realice el programa para que el MCU encienda los LEDs del puerto B con la siguiente secuencia de encendido repetitiva: b0 y b7 – b1 y b6 – b2 y b5 – b3 y b4. Luego, cuando el usuario presione y suelte el pulsador debe cambiar a la secuencia: b7-b6-b5-b4-b3-b2-b1-b0. Si presiona y suelta nuevamente vuelve a la secuencia original y así sucesivamente. Elija un retardo adecuado para la visualización en el simulador. Justifique.

1.2 - Interpretación

Lo que se debe hacer en este trabajo es conectar un pulsador al microcontrolador el cual, una vez se suelte trás haber sido presionado, debe causar que cambie el patrón de iluminación de 8 LEDs que también se encuentran conectados al microcontrolador. Hay 4 colores de LED distintos y se asume que se han de usar 2 de cada color a la hora de resolver el trabajo y el orden en que se disponen los mismos en el conexionado eléctrico es en base al criterio estético de quien haya realizado tal disposición.

Hay 2 patrones distintos de iluminación los cuales se deben intercalar cada vez que se presione y suelte el pulsador y cada LED debe tener conectado en serie una resistencia apropiada que limite su corriente a los 10mA requeridos para encenderlo. Para obtener el valor de dichas resistencias se debe utilizar los valores de caída de tensión para cada LED de un color determinado, los cuales han sido provistos en el enunciado del trabajo, y se debe conocer el valor de la tensión de salida de cada pin cuando circulan 10mA, asumiendo que la fuente de alimentación es de 5V. Este valor se encuentra en el siguiente grafico sacado del Datasheet del ATmega328P:



Con lo cual, teniendo en cuenta que los LEDs se encienden con una corriente de 10mA y asumiendo que se estará trabajando en una temperatura ambiente de 25°C, el voltaje de salida VOH que se utilizará para calcular el valor de las resistencias es de 4,75V

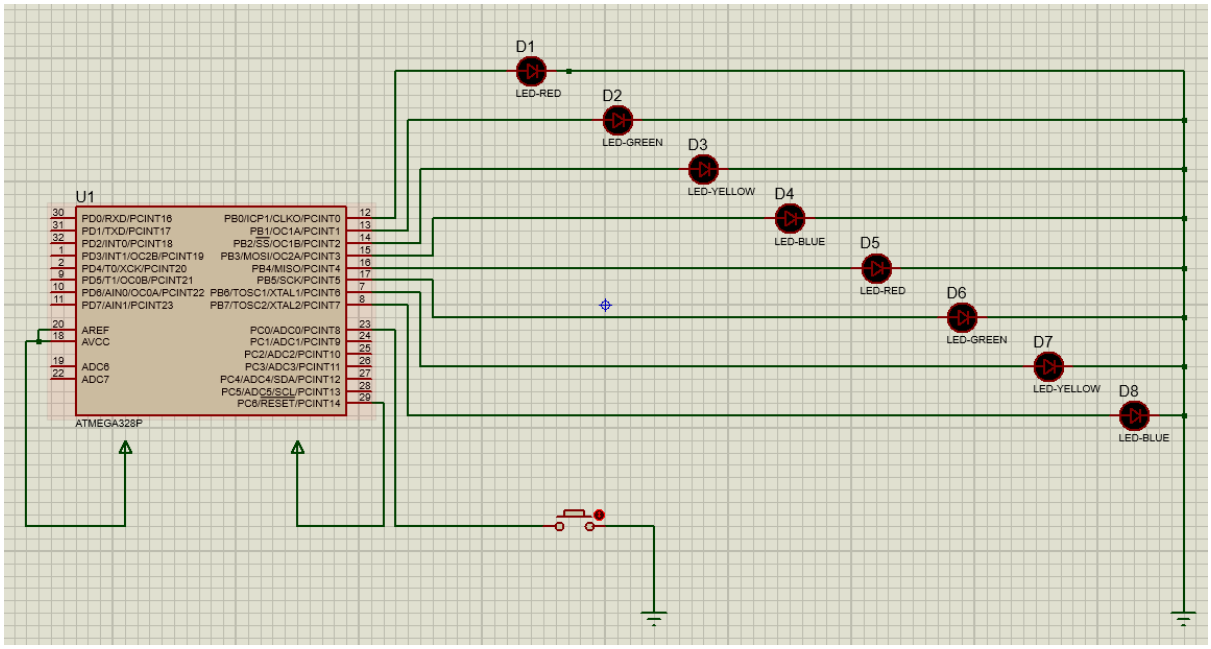
En cuanto a cómo resolver el problema que genera el efecto de rebote del pulsador se asume, en base a que se utiliza la palabra "algoritmo", que la solución debe ser realizada mediante software de modo tal que también se pueda ahorrar gastos al no requerir la agregación de otros componentes electrónicos al circuito.

Por último, se sobreentiende que se debe elegir un retardo para el código de modo tal que se puedan ver claramente los patrones de iluminación de los LEDs y cómo cambian los mismos al presionar el pulsador a la hora de simular el funcionamiento del circuito.

2. RESOLUCIÓN

2.1 - Solución Hardware

Lo primero que se hace al momento de diseñar el hardware es conectar los LEDs a los pines del puerto B, los cuales estarán en modo salida, y el pulsador a uno de los pines de otro de los puertos (en este caso se elige conectarlo a PC0), el cual estará en modo entrada.



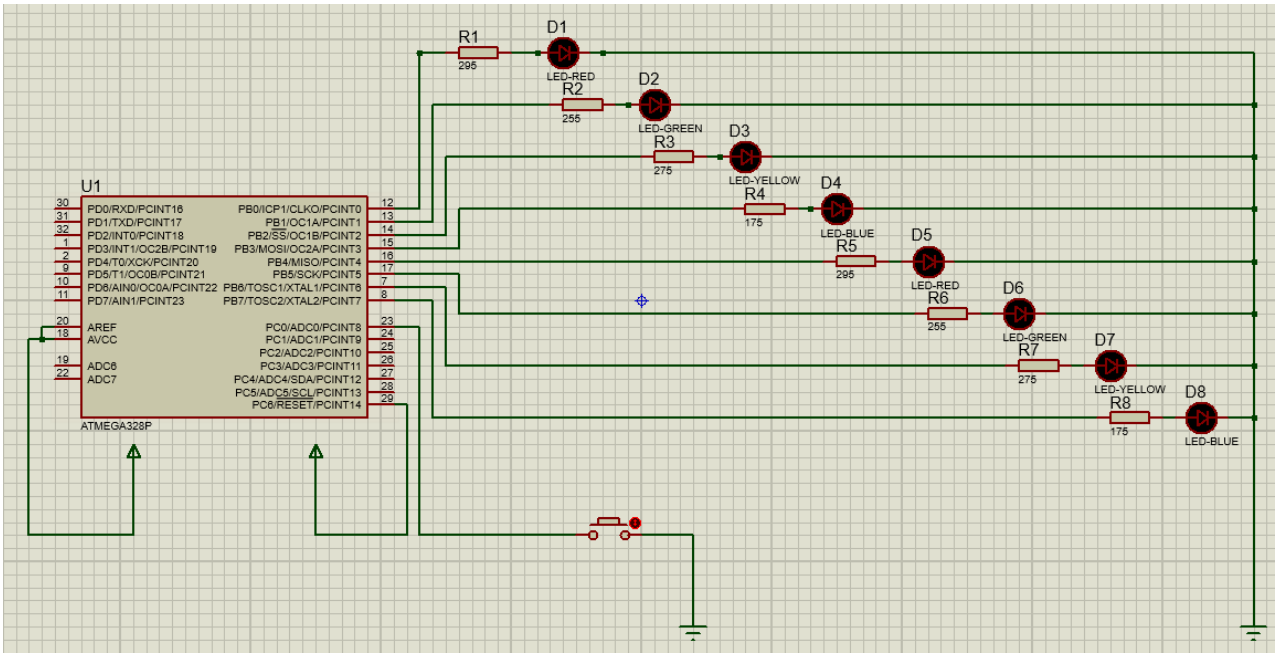
En la conexión elegida los LEDs se encenderán cuando su pin asociado tenga un valor alto de tensión a la salida. Además, el pulsador debe tener una resistencia en serie para limitar la corriente que atraviesa el pin, pero para ahorrarse el tener que conseguir otra resistencia además de las de los LEDs, simplemente se habilitará mediante software el pull-up interno del microcontrolador.

Una vez se tiene lo anterior, lo siguiente es colocar las resistencias en serie a cada LED para limitar la corriente a 10mA. Los siguientes son los cálculos del valor de la resistencia correspondiente a cada color:

$$(V_{salida} - V_{diodo}) / I_{diodo} = R_{diodo}$$

- Para un LED rojo: (4,75V - 1,8V) / 10mA = 295kΩ
- Para un LED verde: (4,75V - 2,2V) / 10mA = 255kΩ
- Para un LED amarillo: (4,75V - 2V) / 10mA = 275kΩ
- Para un LED azul: (4,75 V - 3V) / 10mA = 175kΩ

Se conecta la resistencia correspondiente a cada LED en Proteus.



Por último, para solucionar el efecto de debounce del pulsador, si bien se podría resolver por hardware (agregando, por ejemplo, una resistencia y un capacitor formando una T con el pulsador), en este trabajo la solución está realizada por software con la intención de ahorrar gastos al no tener que comprar otra resistencia y un capacitor para añadir al circuito.

2.2 - Solución Software

La primera sección del programa se encarga de incluir bibliotecas necesarias para el funcionamiento del código y de la definición de constantes que permiten un cambio rápido de los tiempos en varias secciones del código al momento de probar el programa.

```
// Inclusión de cabeceras de bibliotecas de código
#include <avr/io.h> // Definición de registros del microcontrolador
#define F_CPU 16000000UL // Especifico la frecuencia de reloj del MCU en 16MHz
#include <util/delay.h> // Retardos por software – Macros: depende de F_CPU
#include <stdint.h> // Incluye las definiciones de variables enteras

// Se definen constantes del programa
#define CHECK_TIME 5 // Tiempo de chequeo del botón
#define REBOUND_TIME 40 // Tiempo de espera por los posibles rebotes mecánicos
#define LIGHT_TIME 200 // Tiempo que se mantiene encendido cada configuracion de
LEDS

//Definición de funciones
void checkButton(uint8_t * mode);
void changeLights(uint8_t mode);
void mode0(uint8_t state);
void mode1(uint8_t state);
```

Programa principal

En este programa se inicializan los puertos del microcontrolador y se ingresa al loop que luego ejecutará las funciones del programa. Existen dos funciones, una encargada de mirar el estado del botón y cambiar el modo de ejecución de las luces en caso de ser necesario y otra encargada de cambiar entre las distintas configuraciones de luces. Al final del loop se incluye un retardo que es el único que hay en todo el programa para evitar bloqueos innecesarios dentro de las funciones que eviten el correcto funcionamiento del resto de las mismas.

int main (void) {

Se inicializan los puertos:

Se indica el pin 0 del puerto C como entrada

Se activa el modo de pull up interno en el PINC0

Se indican todos los pines del puerto B como salidas

while(1)

{

Función para ver el estado del botón

Función para cambiar el estado de las luces

Espera de 1 ms

}

Final de programa

} Fin programa principal

Código del programa principal

Este código cuenta con una única variable que no forma parte de las declaradas en la biblioteca de avr/io.h, esta es "mode" encargada de indicar en qué patrón de funcionamiento se encuentran las luces, si esta toma un valor de 0 los leds se prenden uno por uno desde el que se encuentra conectado al puerto b7 hasta el que se encuentra en el puerto b0. En caso de que mode tome un valor 1 se ejecuta la otra secuencia, la cual prende los leds de a pares, comenzando por los puertos b7 y b0 y yendo hacia el centro, terminando en los puertos b3 y b4.

```
// Programa principal
int main (void)
{
    // Inicializacion de puertos
    DDRC &=~(1<< PINC0); // Se indica el pin 0 del puerto C como entrada
    PORTC |= (1<< PINC0); // Se activa el modo de pullup interno en el PINC0
    DDRB = 0xFF; // Se indican todos los pines del puerto B como salidas
    uint8_t mode = 1;
    while(1)
    {
        checkButton(&mode); // Funcion para ver el estado del boton
        changeLights(mode); //Funcion para cambiar el estado de las luces
        _delay_ms(1); // Realizo un delay para que cada funcion se lleve a cabo cada 1 ms
    }
    // Final de programa
    return 0;
}
```

Función para ver el estado del botón}

Esta función se encarga de mirar el estado del botón y de cambiar de modo el funcionamiento de las luces en caso de ser necesario. También tiene en cuenta un tiempo de pausa ante posibles rebotes mecánicos. Dentro de esta hay un contador que lleva la cuenta de la cantidad de veces que se entró a la misma y solo deja que se acceda al código que lleva a cabo el chequeo del botón cuando este alcanza un valor definido al comienzo del programa.

Se eligió un tiempo de 5 milisegundos para el chequeo del botón ya que este permitía que se revisara el botón de manera rápida pero sin la necesidad de realizar el chequeo para cada una de las iteraciones del loop. Luego de realizar pruebas con el programa quedó en evidencia que este tiempo alcanzaba para detectar los cambios por más rápido que estos sean.

El programa chequea si el valor de PINC0 cambió, y de ser así, comienza un conteo de tiempo definido al comienzo del programa en el que no revisa el estado del botón nuevamente hasta pasado el mismo. Este tiempo se eligió de 40 ms dado que ese es el tiempo mencionado en la documentación dada sobre "CMOS Switch Debouncers" en el PDF explicación del tp1. Cuando este tiempo termina se cambia realmente cual es el valor del botón y se termina el conteo. En caso de que el conteo termine y el valor del botón ahora pase a ser un 1 (se dejó de presionar el mismo) se realiza el cambio de modo de funcionamiento de las luces.

```
void checkButton{
    Se aumenta el contador
    Si se inicializó el contador anti rebote, se aumenta el valor del mismo
    Cuando pase el tiempo establecido para revisar el estado del botón {
        Se reinicia el contador
        Si el anterior estado estable es 1{
            Si el PINC0 está en 0 y no se activó el conteo {
                Inicializo el contador anti-rebotes
            }
            Cuando pase el tiempo anti-rebotes {
                Se indica que el actual estado estable es un 0
                Se indica que terminó el conteo
            }
        } Else {//si el anterior estado estable es 0
            Si el PINC0 está en 1 y no se activó el conteo {
                Inicializo el contador anti-rebotes
            }
            Cuando pase el tiempo anti-rebotes {
                Se indica que el actual estado estable es un 1
                Se indica que terminó el conteo
                Se cambia el modo de las luces
            }
        }
    }
} Fin función para ver el estado del botón
```

Código de la función del botón

Variables:

Cont: Permite llevar cuenta de las iteraciones en las que se entró a la función para realizar el código de chequeo cada un tiempo establecido por "CHECK_TIME" definida anteriormente.

Wait: Permite saber si se está realizando un conteo antirebotes si su valor no es 0. Se incrementa solo en este caso y vuelve al valor 0 cuando se termine la cuenta para que este no se siga incrementando.

Last: Esta variable mantiene el último valor estable del pulsador, esta se cambia solo después de que pase el tiempo antirrebotes.

```
void checkButton(uint8_t * mode){
    static uint8_t cont = 0;
    static uint8_t wait = 0;
    static uint8_t last = 1;
    cont++;
    if (wait > 0) { // El contador anti rebote solo se incrementa si fue inicializado
        wait++; // Este contador sirve para contar el tiempo anti-rebote
    }
    if (cont >= CHECK_TIME){ //  Cuando pase el tiempo establecido para revisar el estado del boton
        cont = 0; // Reinicio el contador
        if (last){ // Si el anterior estado estable es 1
            if ((!(PINC & (1<<PINC0))) && (!wait)){ // Si el PINC0 esta en 0 y no se activo el conteo
                wait = 1; // Al poner el contador >0 se indica que comenzo una cuenta de tiempo anti-rebote
            }
            if (wait >= REBOUND_TIME){ //cuando pase el tiempo anti-rebote
                last = 0; // Se marca que ahora el valor que toma el PINC0 es un 0
                wait = 0; // Se indica que termino el conteo
            }
        }else{ // Si el anterior estado estable es 0
            if ((PINC & (1<<PINC0)) && (!wait)){ // Si el PINC0 esta en 1 y no se activo el conteo
                wait = 1; // Al poner el contador >0 se indica que comenzo una cuenta de tiempo anti-rebote
            }
            if (wait >= REBOUND_TIME){ // Cuando pase el tiempo anti-rebote
                last = 1; // Se marca que ahora el valor que toma el PINC0 es un 1
                wait = 0; // Se indica que termino el conteo
                *mode ^= 1; // En este punto se solto el pulsador por lo que debe cambiar el modo de luces
            }
        }
    }
}
```

Función para cambiar el estado de las luces

Esta función se encarga de organizar el cambio de luces en el programa. Cada cierto tiempo establecido por "LIGHT_TIME" una constante definida al comienzo del programa que indica en ms, cuánto tiempo se mantiene encendida cada configuración de leds. La función recibe en qué modo se encuentran las luces actualmente y en base a eso ejecuta una u otra secuencia de luces.

Para cambiar dentro de una misma secuencia se mantiene un contador que aumenta cuando se sale del código y al entrar al mismo, en la siguiente iteración, se verifica que no exceda la cantidad máxima de configuraciones posibles.

void changeLights{
Se aumenta el contador
Cuando pase el tiempo establecido para cambiar las luces {
Se reinicia el contador
Si se superó la cantidad máxima de estados se reinicia
Si es el modo 1{
Se cambia al estado correspondiente del modo 1
El modo 1 enciende los LEDS de a pares de afuera hacia adentro
}
Else si es el modo 0{
Se cambia al estado correspondiente del modo 0
El modo 0 enciende uno por uno todos los pines del puerto B
}
Se aumenta el indicador de estado;
}
} Fin función para cambiar el estado de las luces

Código de la función de las luces

Variables:

Cont: Permite llevar cuenta de las iteraciones en las que se entró a la función y cada un tiempo establecido por "LIGHT_TIME", entra al código de cambio de configuración.

State: esta variable toma un valor entre 0 y 8, e indica mediante este qué configuración de luces se debe mostrar. El valor se reinicia al llegar a 8 para evitar usar modos indefinidos.

```
void changeLights(uint8_t mode){
    static uint8_t cont = 0;
    static uint8_t state = 0;
    cont++;
    if (cont >= LIGHT_TIME){ // cuando pase el tiempo establecido para cambiar las luces
        cont = 0; //reinicio el contador
        state %= 8; // reinicio el estado cuando supere los 8 estados posibles
        if (mode)
        {
            mode1(state % 4); //Si es el modo 1 se envian solo 4 estados posibles
        }
        else
        {
            mode0(state);
        }
        state++;
    }
}
```

Para una mayor modularización, los códigos de los modos de operación se llevaron a cabo cada uno en una función separada, la cual recibe en qué estado debe ponerse la configuración. Al enviar el parámetro de estado para el modo 1 se realiza una operación módulo 4, ya que este modo solo cuenta con esta cantidad de estados.

Las funciones de modo de operación utilizan un switch para cambiar el estado de los puertos dependiendo del parámetro de estado que reciban, dada nuestra implementación de hardware un 1 en un puerto significa una luz encendida y viceversa.

```
void mode0(uint8_t state){ // El modo 0 enciende uno por uno todos los pines del puerto B
    switch (state) { // Se selecciona la configuracion de LEDS segun el estado que se recibio
        case 0:
        PORTB = 0b10000000; //
        break;
        case 1:
        PORTB = 0b01000000; //
        break;
        case 2:
        PORTB = 0b00100000; //
        break;
        case 3:
        PORTB = 0b00010000; //
        break;
        case 4:
        PORTB = 0b00001000; //
        break;
        case 5:
        PORTB = 0b00000100; //
        break;
        case 6:
        PORTB = 0b00000010; //
        break;
        case 7:
        PORTB = 0b00000001; //
        break;
        default:
        break;
    }
}
```

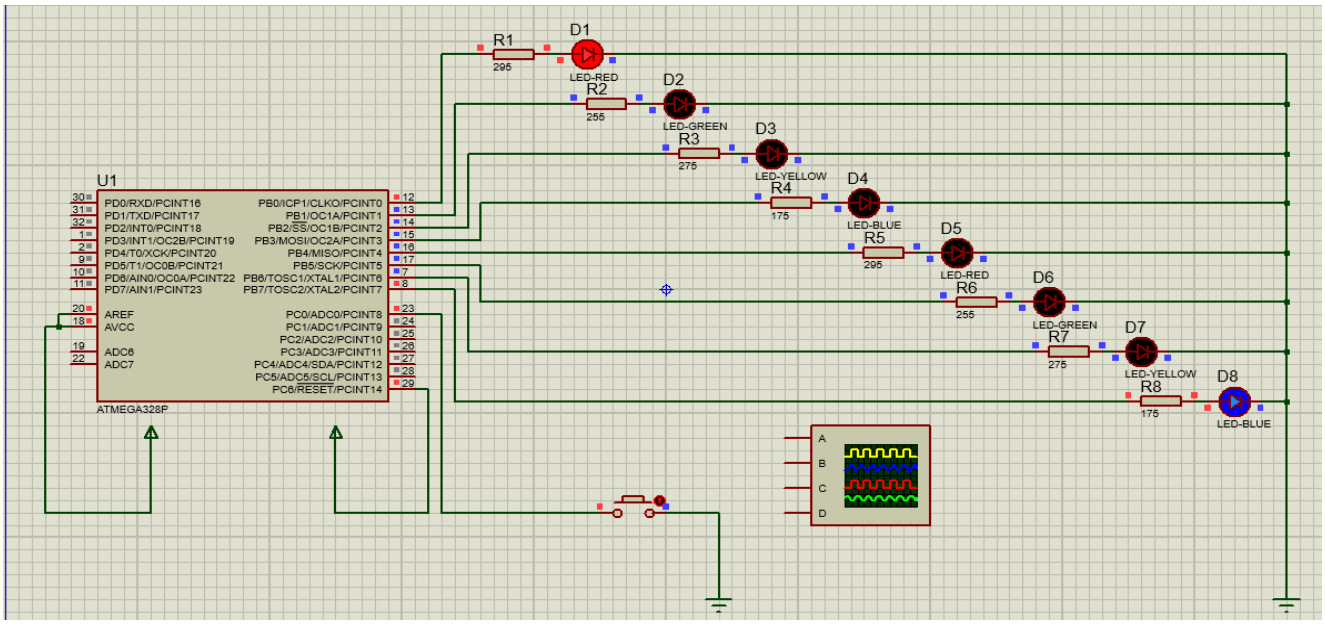
Función mode 0

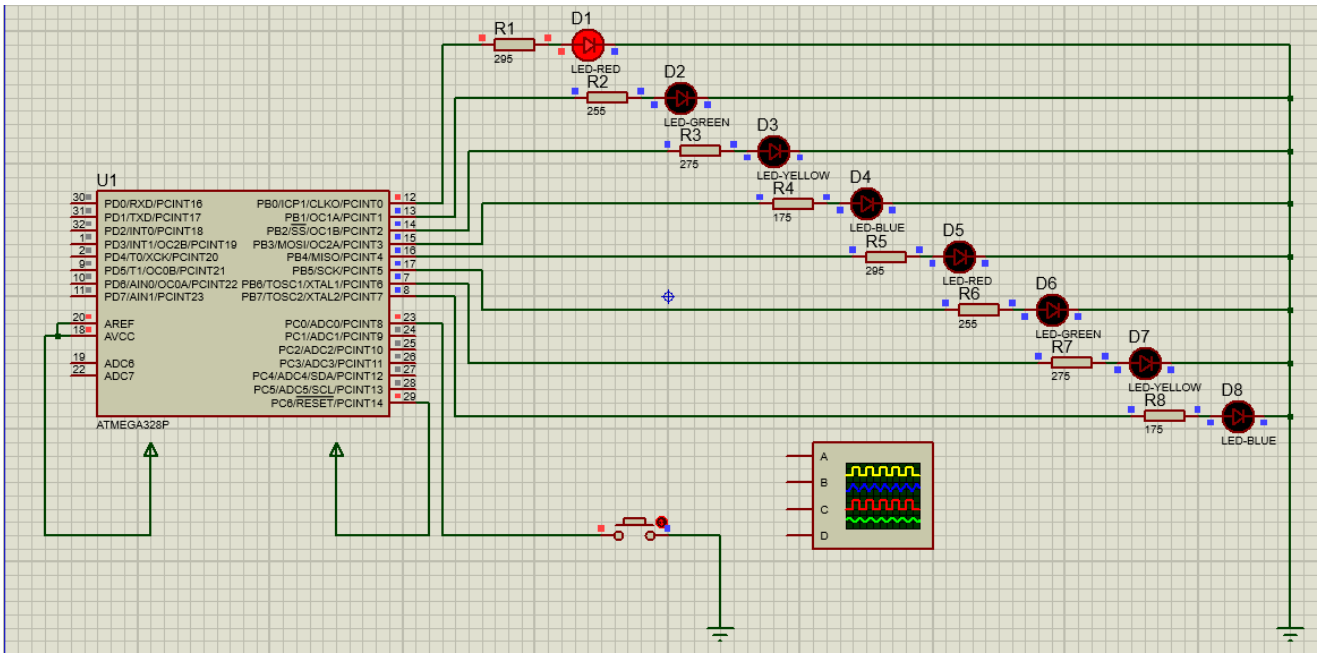
```
void mode1(uint8_t state){ // El modo 1 enciende los LEDS de a pares desde el mas exterior hasta el central
    switch (state) { // Se selecciona la configuracion de LEDS segun el estado que se recibio
        case 0:
        PORTB = 0b10000001; //
        break;
        case 1:
        PORTB = 0b01000010; //
        break;
        case 2:
        PORTB = 0b00100100; //
        break;
        case 3:
        PORTB = 0b00011000; //
        break;
        default:
        break;
    }
}
```

Función mode 1

3. VALIDACIONES

A la hora de correr el programa se observa el funcionamiento de la secuencia repetitiva por pares de leds que se encienden en orden desde el exterior del circuito hasta el centro del mismo, siempre y cuando se mantenga presionado el pulsador no se provocará ningún cambio en la secuencia dado que solo se busca el cambio al momento de soltar el mismo, entonces al momento de soltar el pulsador se observa con claridad el cambio de secuencia que pasan a encender de forma escalonada, está alteración de secuencias se observa de forma repetitiva en cuanto el usuario presione y suelte el pulsador.





Al momento del cambio de secuencia se genera un retardo el cual el programa utiliza para controlar rebotes o que se presione y suelte varias veces en simultáneo ya que se asume que esto tiene que provocar un solo cambio y no reiterados cambios proporcionales a las veces presionado. La secuencia de imágenes siguiente demuestra la condición mencionada anteriormente, mostrando el estado inicial con el resultado obtenido luego de soltar repetidas veces el pulsador simultáneamente.

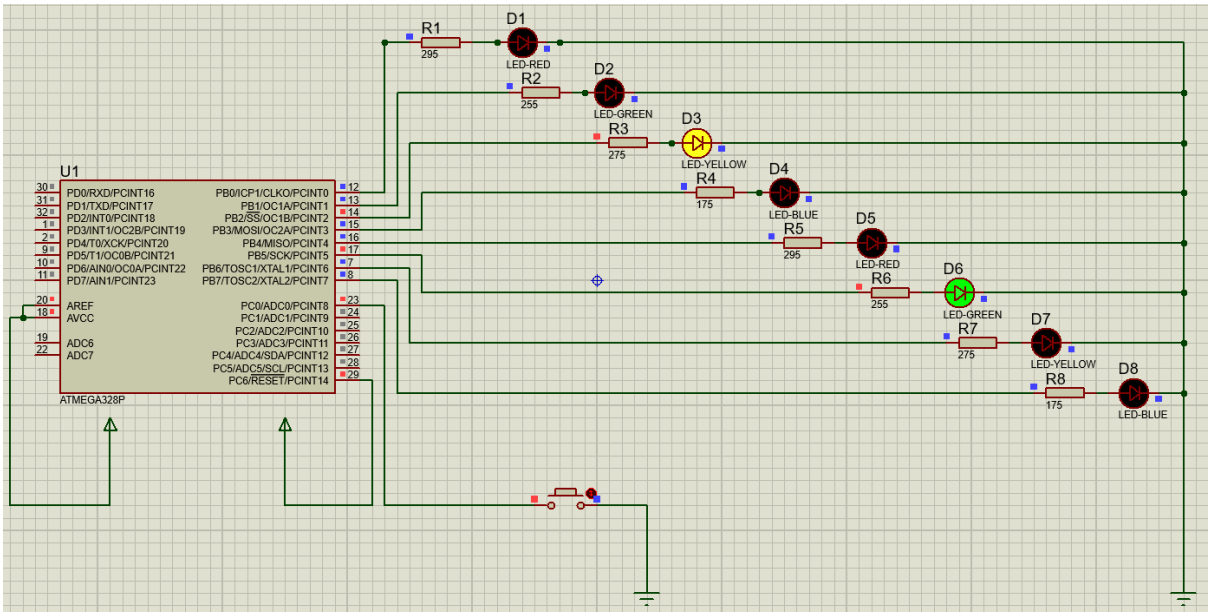


Imagen secuencia 1 inicial

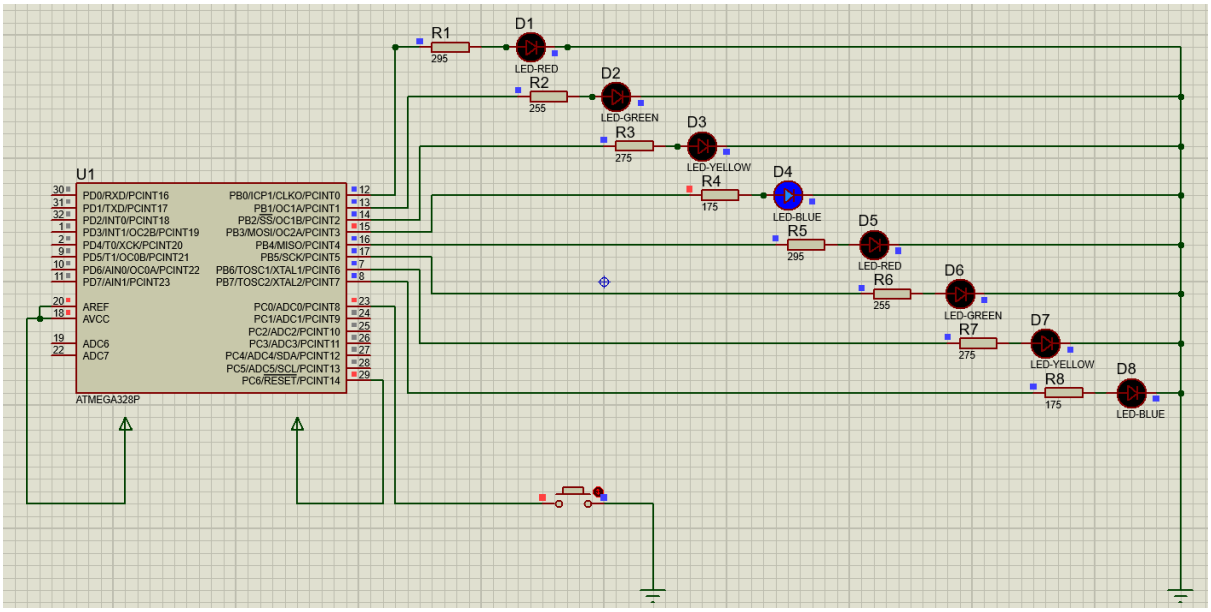
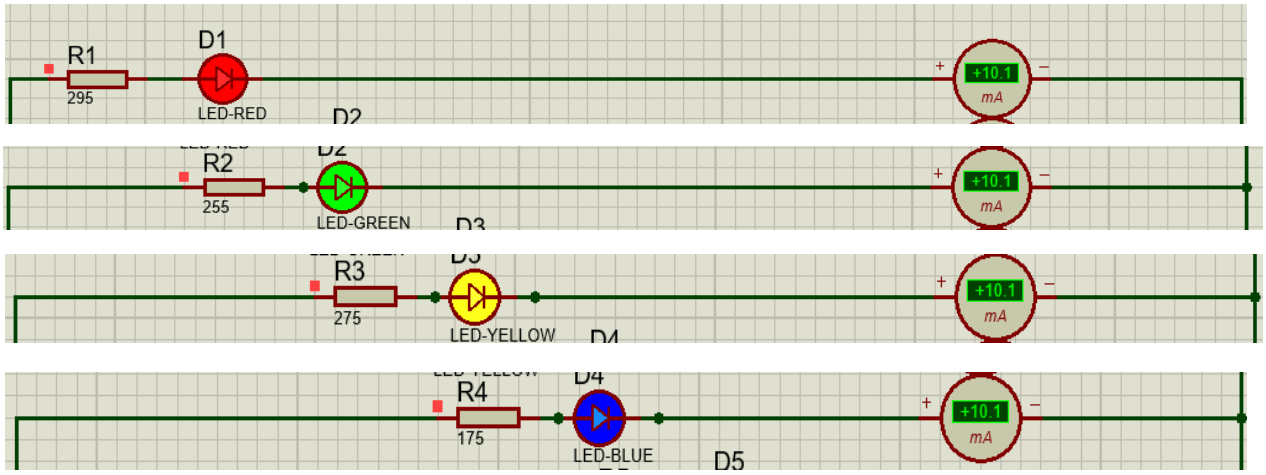
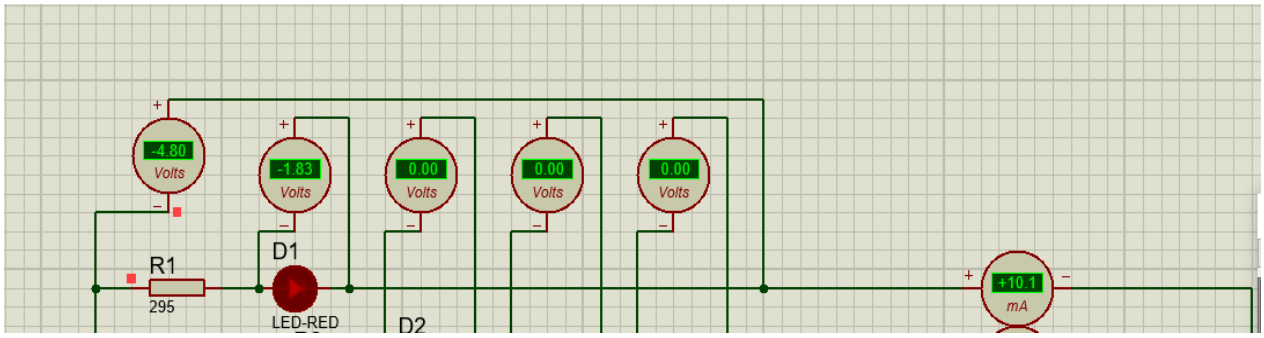


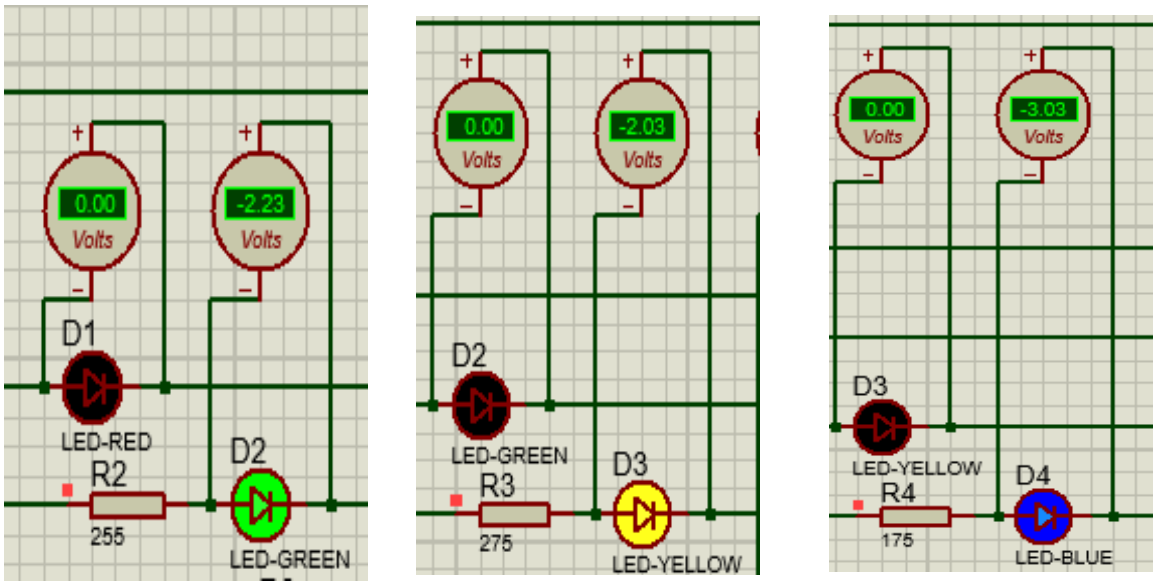
Imagen secuencia 2 final

En los cálculos anteriores se impuso una corriente de 10mA, pero al medir valores la corriente es de 10.1mA. Se verá más adelante que los valores de tensión calculados anteriormente tampoco coinciden exactamente con los medidos en Proteus y esto se debe a que la tensión de salida de cada pin esta levemente por encima de 4,75V como se asumió anteriormente. Esta diferencia de valores se debe a las limitaciones del mismo Proteus que decide aproximar el valor de dicha tensión.



Luego, se utiliza un medidor de tensión para comprobar que las tensiones de salida coinciden con las calculadas y se puede observar que la tensión de salida que nos da el microprocesador alimentado es de no son 4,75V sino 4,8V como se había mencionado anteriormente. En las imágenes a continuación también se muestran las mediciones de las caídas de tensión en cada uno de los LEDs para comprobar los correctos valores del enunciado.



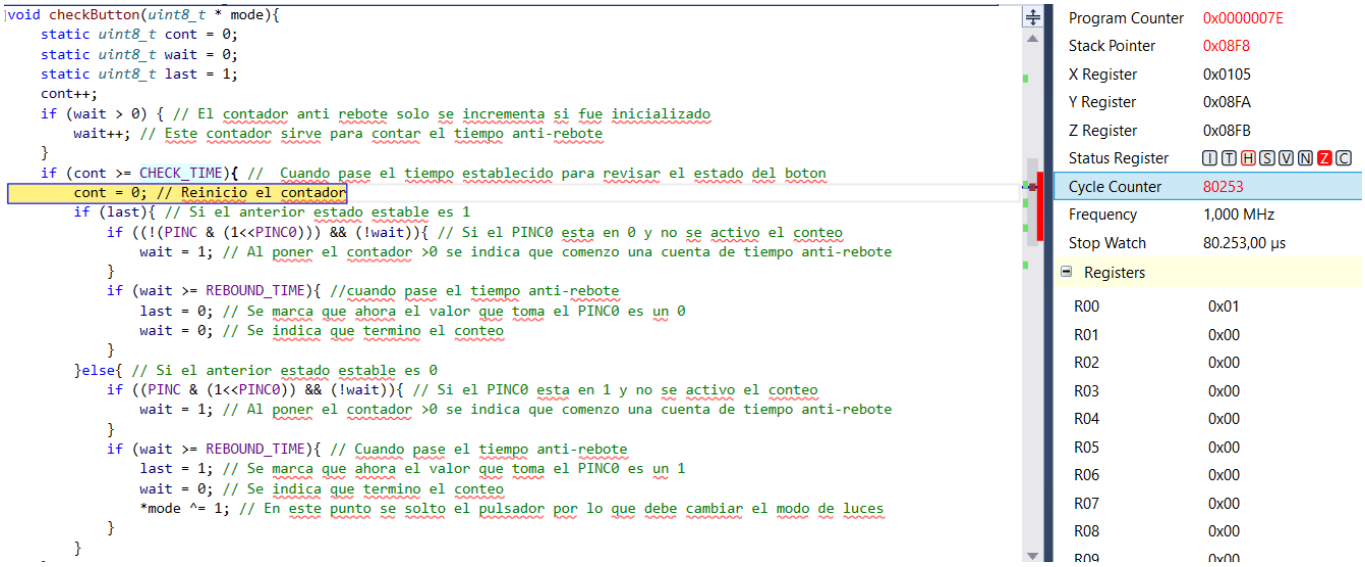


En cuanto a la temporización del programa se hace control mediante el contador de ciclos que nos proporciona el simulador Microchip Studio, asumiendo una frecuencia del reloj del MCU en 16 MHz, también se muestran los tiempos definidos para tres funciones específicas que son el tiempo para chequear estado del botón, el tiempo por

```
#include <avr/io.h> // Definición de Registros del microcontrolador
#define F_CPU 16000000UL // Especifico la frecuencia de reloj del MCU en 16MHz
#include <util/delay.h> // Retardos por software - Macros: depende de F_CPU
#include <stdint.h> // Incluyo las definiciones de variables enteras

// Defino constantes del programa
#define CHECK_TIME 5 // Tiempo de chequeo del boton
#define REBOUND_TIME 40 // Tiempo de espera por los posibles rebotes mecanicos
#define LIGHT_TIME 10 // Tiempo que se mantiene encendido cada configuracion de LEDS
```

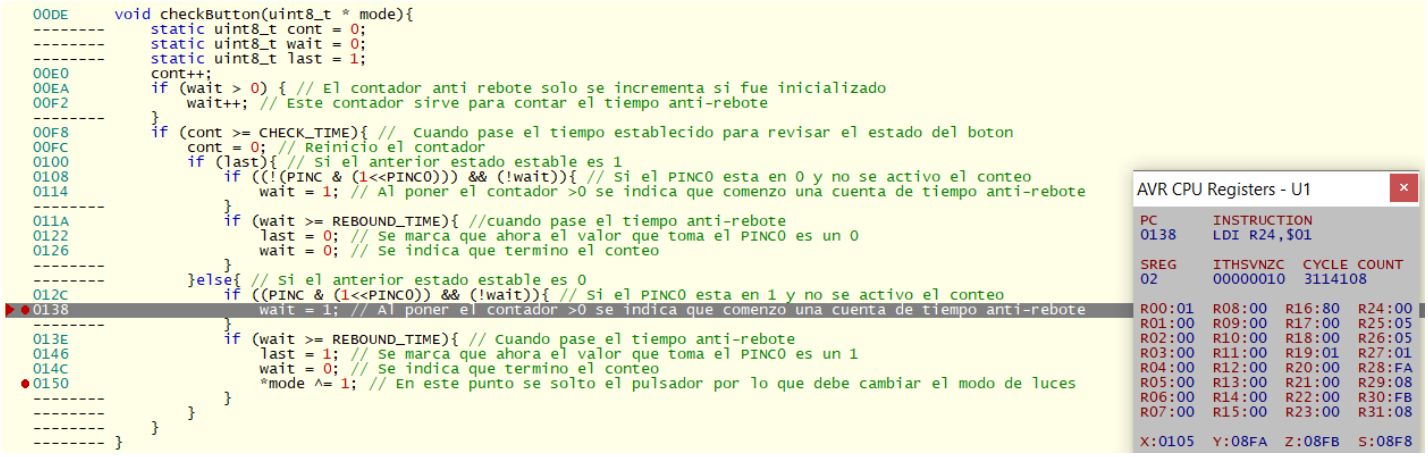
Se verifica el tiempo que le toma al proceso volver a chequear el estado del botón, dentro de la función check button, el cual arroja una cantidad de 80253 ciclos de reloj que se traduce a 5 ms como el tiempo establecido.



```
void checkButton(uint8_t * mode){
    static uint8_t cont = 0;
    static uint8_t wait = 0;
    static uint8_t last = 1;
    cont++;
    if (wait > 0) { // El contador anti rebote solo se incrementa si fue inicializado
        wait++; // Este contador sirve para contar el tiempo anti-rebote
    }
    if (cont >= CHECK_TIME){ //  Cuando pase el tiempo establecido para revisar el estado del boton
        cont = 0; // Reinicio el contador
        if (last){ // Si el anterior estado estable es 1
            if ((!(PINC & (1<<PINC0))) && (!wait)){ // Si el PINC0 esta en 0 y no se activo el conteo
                wait = 1; // Al poner el contador >0 se indica que comenzo una cuenta de tiempo anti-rebote
            }
            if (wait >= REBOUND_TIME){ //cuando pase el tiempo anti-rebote
                last = 0; // Se marca que ahora el valor que toma el PINC0 es un 0
                wait = 0; // Se indica que termino el conteo
            }
        }else{ // Si el anterior estado estable es 0
            if ((PINC & (1<<PINC0)) && (!wait)){ // Si el PINC0 esta en 1 y no se activo el conteo
                wait = 1; // Al poner el contador >0 se indica que comenzo una cuenta de tiempo anti-rebote
            }
            if (wait >= REBOUND_TIME){ // Cuando pase el tiempo anti-rebote
                last = 1; // Se marca que ahora el valor que toma el PINC0 es un 1
                wait = 0; // Se indica que termino el conteo
                *mode ^= 1; // En este punto se solto el pulsador por lo que debe cambiar el modo de luces
            }
        }
```

Program Counter	0x0000007E
Stack Pointer	0x08F8
X Register	0x0105
Y Register	0x08FA
Z Register	0x08FB
Status Register	I T H S V N Z C
Cycle Counter	80253
Frequency	1,000 MHz
Stop Watch	80.253,00 µs
Registers	
R00	0x01
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00

Luego se controla el tiempo del rebote mecánico donde se deben ignorar posibles cambios de estado en los valores, desde el inicio del contador (imagen 1) hasta que se suelta el pulsador y se produce el cambio de estado (imagen 2) se tiene el verdadero contador de ciclos con el resultado de la diferencia entre los valores de los ciclos de reloj obtenidos al comienzo del contador y al finalizar el mismo, lo cual nos da un total de 642165 ciclos que para la frecuencia mencionada equivalen a 40 ms



```
00DE     void checkButton(uint8_t * mode){
--------     static uint8_t cont = 0;
--------     static uint8_t wait = 0;
--------     static uint8_t last = 1;
00E0         cont++;
00EA         if (wait > 0) { // El contador anti rebote solo se incrementa si fue inicializado
00F2             wait++; // Este contador sirve para contar el tiempo anti-rebote
--------     }
00F8         if (cont >= CHECK_TIME){ //  Cuando pase el tiempo establecido para revisar el estado del boton
00FC             cont = 0; // Reinicio el contador
0100             if (last){ // Si el anterior estado estable es 1
0108                 if ((!(PINC & (1<<PINC0))) && (!wait)){ // Si el PINC0 esta en 0 y no se activo el conteo
0114                     wait = 1; // Al poner el contador >0 se indica que comenzo una cuenta de tiempo anti-rebote
--------                 }
011A                 if (wait >= REBOUND_TIME){ //cuando pase el tiempo anti-rebote
0122                     last = 0; // Se marca que ahora el valor que toma el PINC0 es un 0
0126                     wait = 0; // Se indica que termino el conteo
--------                 }
--------             }else{ // Si el anterior estado estable es 0
012C                 if ((PINC & (1<<PINC0)) && (!wait)){ // Si el PINC0 esta en 1 y no se activo el conteo
0138                     wait = 1; // Al poner el contador >0 se indica que comenzo una cuenta de tiempo anti-rebote
--------                 }
013E                 if (wait >= REBOUND_TIME){ // Cuando pase el tiempo anti-rebote
0146                     last = 1; // Se marca que ahora el valor que toma el PINC0 es un 1
014C                     wait = 0; // Se indica que termino el conteo
0150                     *mode ^= 1; // En este punto se solto el pulsador por lo que debe cambiar el modo de luces
--------                 }
--------             }
--------     }
-------- }
```

AVR CPU Registers - U1

PC	INSTRUCTION		
0138	LDI R24,$01		
SREG	ITHSVNZC	CYCLE COUNT	
02	00000010	3114108	
R00:01	R08:00	R16:80	R24:00
R01:00	R09:00	R17:00	R25:05
R02:00	R10:00	R18:00	R26:05
R03:00	R11:00	R19:01	R27:01
R04:00	R12:00	R20:00	R28:FA
R05:00	R13:00	R21:00	R29:08
R06:00	R14:00	R22:00	R30:FB
R07:00	R15:00	R23:00	R31:08
X:0105	Y:08FA	Z:08FB	S:08F8

Imagen 1

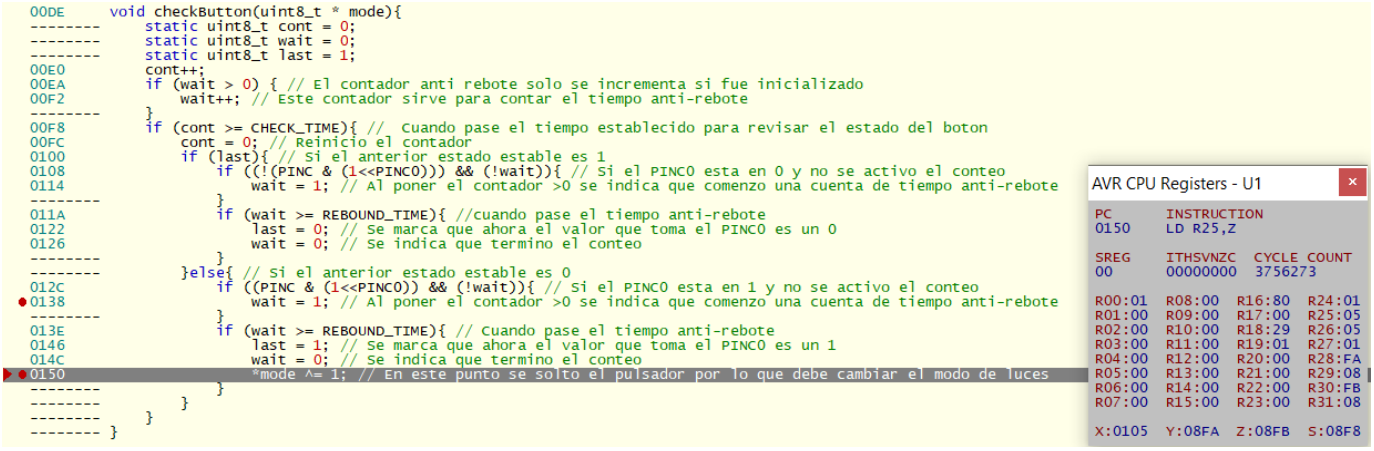
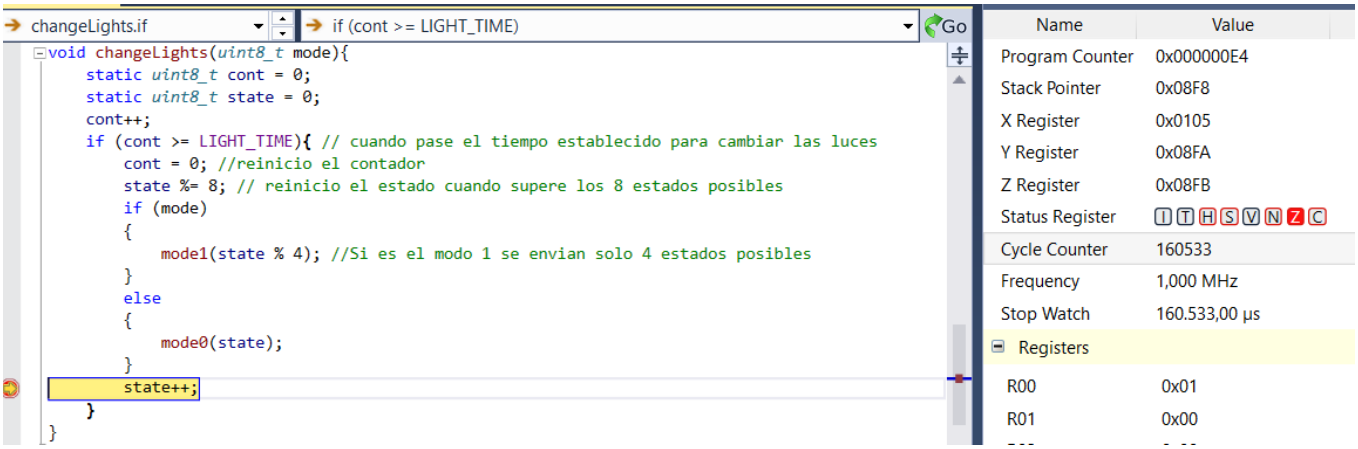


Imagen 2

Por último en cuanto al proceso del cambio de luz y de estado, podemos observar que el contador de ciclo se detiene en 160533, lo que calculando con la frecuencia establecida previamente equivale a 10 ms lo cual coincide con el tiempo definido para la duración del led encendido.



Podemos concluir que para nuestra frecuencia de reloj establecida en 16MHz la comprobación de los tiempos en ejecución es exitosa ya que coinciden con los valores previamente definidos.

4. CÓDIGO

```
// Inclusión de cabeceras de bibliotecas de código
#include <avr/io.h> // Definición de Registros del microcontrolador
#define F_CPU 16000000UL // Especifico la frecuencia de reloj del MCU en 16MHz
#include <util/delay.h> // Retardos por software – Macros: depende de F_CPU
#include <stdint.h> // Incluyo las definiciones de variables enteras

// Defino constantes del programa
#define CHECK_TIME 5 // Tiempo de chequeo del boton
#define REBOUND_TIME 40 // Tiempo de espera por los posibles rebotes mecanicos
#define LIGHT_TIME 10 // Tiempo que se mantiene encendido cada configuracion de LEDS

//Definicion de funciones
void checkButton(uint8_t * mode);
void changeLights(uint8_t mode);
void mode0(uint8_t state);
void mode1(uint8_t state);

// Programa principal
int main (void)
{
	// Inicializacion de puertos
	DDRC &=~(1<< PINC0); // Se indica el pin 0 del puerto C como entrada
	PORTC |= (1<< PINC0); // Se activa el modo de pullup interno en el PINC0
	DDRB = 0xFF; // Se indican todos los pines del puerto B como salidas
	uint8_t mode = 1;
	while(1)
	{
		checkButton(&mode); // Funcion para ver el estado del boton
		changeLights(mode); //Funcion para cambiar el estado de las luces
		_delay_ms(1); // Realizo un delay para que cada funcion se lleve a cabo
cada 1 ms
	}
	// Final de programa
	return 0;
}
```

```
void checkButton(uint8_t * mode){
	static uint8_t cont = 0;
	static uint8_t wait = 0;
	static uint8_t last = 1;
	cont++;
	if (wait > 0) { // El contador anti rebote solo se incrementa si fue
inicializado
		wait++; // Este contador sirve para contar el tiempo anti-rebote
	}
	if (cont >= CHECK_TIME){ //  Cuando pase el tiempo establecido para revisar el
estado del boton
		cont = 0; // Reinicio el contador
		if (last){ // Si el anterior estado estable es 1
			if ((!(PINC & (1<<PINC0))) && (!wait)){ // Si el PINC0 esta en 0
y no se activo el conteo
				wait = 1; // Al poner el contador >0 se indica que comenzo
una cuenta de tiempo anti-rebote
			}
			if (wait >= REBOUND_TIME){ //cuando pase el tiempo anti-rebote
				last = 0; // Se marca que ahora el valor que toma el PINC0
es un 0
				wait = 0; // Se indica que termino el conteo
			}
		}else{ // Si el anterior estado estable es 0
			if ((PINC & (1<<PINC0)) && (!wait)){ // Si el PINC0 esta en 1 y
no se activo el conteo
				wait = 1; // Al poner el contador >0 se indica que comenzo
una cuenta de tiempo anti-rebote
			}
			if (wait >= REBOUND_TIME){ // Cuando pase el tiempo anti-rebote
				last = 1; // Se marca que ahora el valor que toma el PINC0
es un 1
				wait = 0; // Se indica que termino el conteo
				*mode ^= 1; // En este punto se solto el pulsador por lo
que debe cambiar el modo de luces
			}
		}
	}
}

void changeLights(uint8_t mode){
	static uint8_t cont = 0;
	static uint8_t state = 0;
	cont++;
	if (cont >= LIGHT_TIME){ // cuando pase el tiempo establecido para cambiar las
luces
		cont = 0; //reinicio el contador
		state %= 8; // reinicio el estado cuando supere los 8 estados posibles
		if (mode)
		{
			mode1(state % 4); //Si es el modo 1 se envian solo 4 estados
posibles
		}
		else
		{
			mode0(state);
		}
		state++;
	}
}
```

```
void mode0(uint8_t state){ // El modo 0 enciende uno por uno todos los pines del
puerto B
	switch (state) { // Se selecciona la configuracion de LEDS segun el estado que
se recibio
		case 0:
		PORTB = 0b10000000; //
		break;
		case 1:
		PORTB = 0b01000000; //
		break;
		case 2:
		PORTB = 0b00100000; //
		break;
		case 3:
		PORTB = 0b00010000; //
		break;
		case 4:
		PORTB = 0b00001000; //
		break;
		case 5:
		PORTB = 0b00000100; //
		break;
		case 6:
		PORTB = 0b00000010; //
		break;
		case 7:
		PORTB = 0b00000001; //
		break;
		default:
		break;
	}
}
void mode1(uint8_t state){ // El modo 1 enciende los LEDS de a pares desde el mas
exterior hasta el central
	switch (state) { // Se selecciona la configuracion de LEDS segun el estado que
se recibio
		case 0:
		PORTB = 0b10000001; //
		break;
		case 1:
		PORTB = 0b01000010; //
		break;
		case 2:
		PORTB = 0b00100100; //
		break;
		case 3:
		PORTB = 0b00011000; //
		break;
		default:
		break;
	}
}
```

5. CONCLUSIÓN

La implementación de la solución al problema en la parte del software mediante procedimientos no bloqueantes, es decir, mediante procedimientos que no detienen la ejecución funcional del programa es una buena práctica que se ha de ejercer en el ámbito profesional con la intención de mejorar el rendimiento de las aplicaciones. Si bien la simulación muestra el funcionamiento tanto del modelado eléctrico como del código, hay que tener en cuenta que en una aplicación real los tiempos con los que se manejan las revisiones de, por ejemplo, cuando se pulsa o se suelta el pulsador, han de ser más grandes debido a que el tiempo en el que fluye la simulación está muy ralentizado con respecto al tiempo real. Por último, se concluye que la resolución del problema que se describe en este informe cumple con todos los requisitos pedidos en el enunciado con un rendimiento muy bueno, aunque no se puede asegurar que está sea la mejor solución que existe a dicho problema.