Circuitos Digitales y Microcontroladores (E0305)
Abril 2020

CAMBIO DE SECUENCIA DE LEDS UTILIZANDO PULSADOR MECÁNICO

Autor:
Nehuen Pereyra (878/6)

1. Problema A

1.1. Consigna

Realice un programa que permita activar los bits del puerto B uno a la vez y en el siguiente orden: 0-1-2-3-4-5-6-7. Luego que vuelva a empezar.

1.2. Interpretación del problema

Para este problema utilizaremos el microcontrolador y una plaqueta con los 8 LEDs con sus respectivas resistencias.

La figura 1 muestra la secuencia de encendido de LEDs que se debe llevar a cabo.

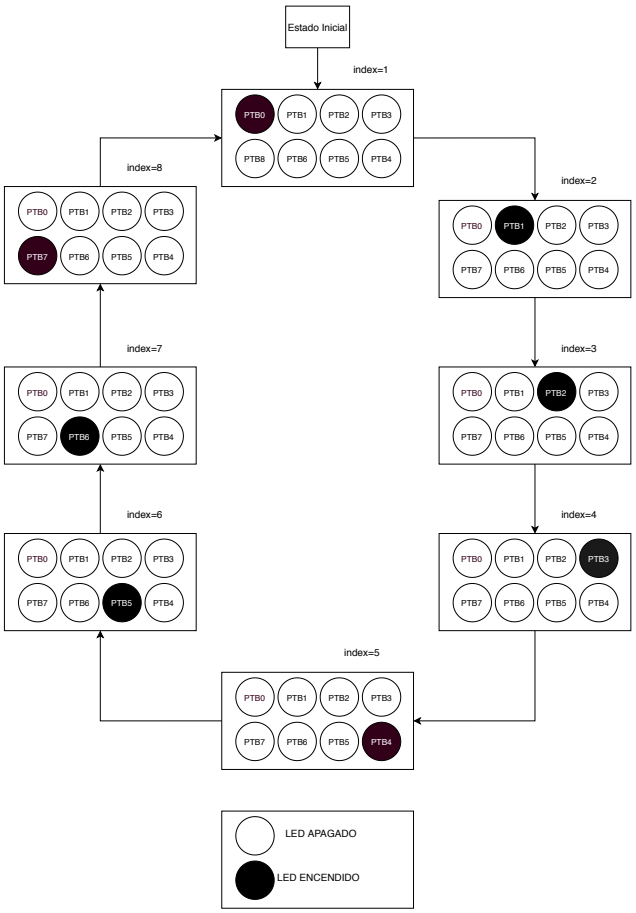


Figura 1: Se indica que bit se debe encender en cada estado

Se requiere utilizar un retardo para la correcta visualización de los LEDs que se implementará en el problema B.

1.3. Resolución del problema

La configuración de las conexiones entre los LEDs y el microcontroladores se definió en la interpretación.

Lo primero que hay que hacer en nuestro programa es configurar el PORTB como salida, esto se logra poniendo todos los bits del PTBDD en alto además el puerto se debe configurar en modo High Drive para que pueda entregar más de 4mA de corriente y esta configuración se logra estableciendo todos los bits de PTBDS en uno. Como por cada LED hacemos circular una corriente de 5mA tenemos una corriente total a través de todos los pines de 40mA que no supera la máxima corriente total que puede entregar el microcontrolador que es de 100mA (Cuadro 1).

Como ya tenemos configurado el PORTB utilizamos el registro de datos (PTBD) para ir estableciendo el estado de las salidas, en donde un 1 establece la salida en alto mientras que un 0 una salida en bajo.

El recorrido se implementó con una función en C llamada recorrido la cual recibe como parámetro un índice y evalúa que led particular tiene que prender, luego aplica un retardo para la correcta visualización del led encendido. Lo que se hace en el programa principal es establecer el índice en el valor uno que corresponde al led 0 luego se va iterando llamando a la función recorrido con el índice actual que se va incrementando en cada iteración, al llegar al último led se restablece el valor al estado inicial y continua el bucle.

El Pseudocódigo es el siguiente:

```
1  Función recorrido(index)
2          Para cada index
3                  PTBD se establece en su estado correspondiente
4          Esperar un tiempo
5  Fin recorrido()
```

Pseudocódigo de recorrido.

El pseudocódigo del programa principal sería el siguiente:

```
1  Función main()
2          Inicializo variables necesarias (ej. index = 1)
3          Configurar como salida los pines del puerto B
4          Configurar High Drive los pines del puerto B
5          Inicializar las salidas del puerto B
6          Repetir indefinidamente
7                  recorrido(index)
8                  Incremento index en uno
9                  Si index mayor o igual a 9
10                         index se establece en 1
11 Fin main()
```

Pseudocódigo del main().

En la resolución del siguiente problema se realizara la función en C para generar un retardo de tiempo.

El correcto funcionamiento de la secuencia lo podemos ver desde el chip View con la siguiente figura:

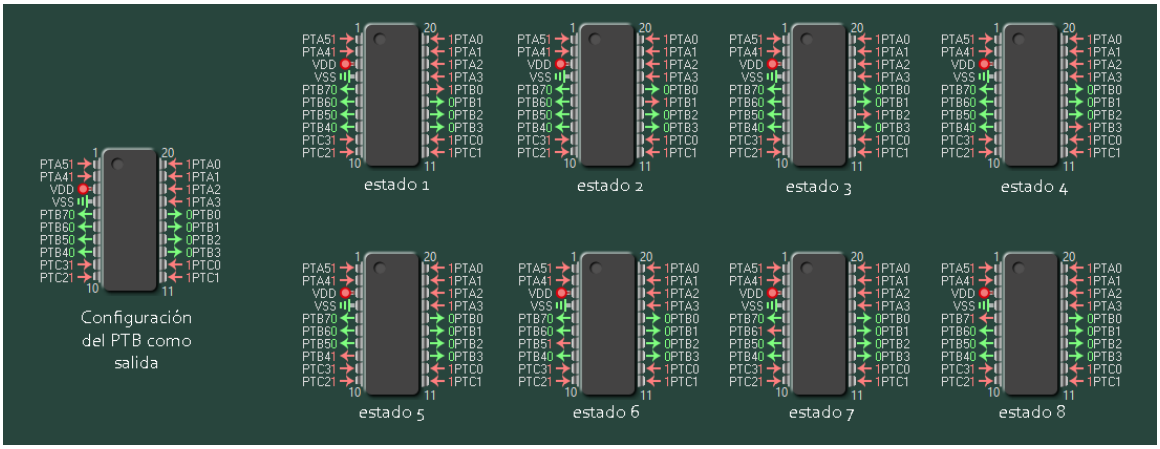


Figura 2: Secuencia que realizan los leds vista desde el chip View

2. Problema B

2.1. Consigna

Realice una función que implemente un retardo para la visualización correcta en el simulador. El valor del retardo depende de cada PC. Simule con el IDE cuantos ciclos de reloj consume el algoritmo de retardo y calcule el tiempo en ms, suponiendo un reloj de 8MHz.

2.2. Interpretación del problema

Para visualizar de manera correcta los LEDs debemos implementar un retardo por software que consiste en mantener a la CPU ocupada realizando calculo (por ejemplo decrementar un contador) y así conseguir un retardo de tiempo. La desventaja es que se consigue a costa de ocupar tiempo de CPU que se podría utilizar para realizar otras operaciones útiles.

2.3. Resolución del problema

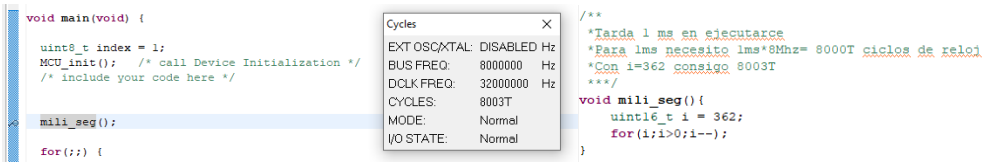
Para realizar el retardo de tiempo se implementaron dos funciones en C:

1. mili_seg() la cual tarda aproximadamente un milisegundo en ejecutarse. Para ello se simula cuántos ciclos de reloj consume el decremento de un número de 16 bits hasta llegar a cero entonces utilizamos esa cantidad de ciclos en la siguiente fórmula:

$$Retardo = \frac{\text{Cantidad de Ciclos}}{\text{Frecuencia de Bus}} * 1000[ms]$$

Con una variable unsigned short int con un valor de 362 se obtiene 8003T ciclos (ver figura 3) entonces:

$$Retardo = \frac{8003}{8000000} * 1000[ms]$$
$$Retardo \approx 1[ms]$$



(a) ciclos de reloj (b) codigo de mili_seg()

Figura 3: Ciclos necesarios para lograr un retardo de 1 ms

El pseudocódigo es el siguiente:

```
1  Función mili_seg()
2          Mientras el valor de la variable no sea cero
3                  Decrementar el valor en una unidad
4  Fin mili_seg()
```

Pseudocódigo de mili_seg().

2. delay(ms) esta función recibe la cantidad de milisegundos que va tardar el retraso (esta es una variable del tipo unsigned short int) y llama a la función mili_seg() esa cantidad de veces.

   El pseudocódigo es el siguiente:

```
1  Función delay(s milisegundos)
2          Repetir s veces
3                  mili_seg()
4  Fin delay()
```

Pseudocódigo de delay().

Se debe tener en cuenta que en el MCU real, el retardo deberá ser de más de 40 ms para que el ojo humano detecte los cambios.

3. Problema C

3.1. Consigna

Cambie el sentido de activación de la secuencia con un pulso en PC0 (simule el presionar y soltar un pulsador). Es decir, mediante el pulsador el usuario puede cambiar a la secuencia: 7-6-5-4-3-2-1-0 y viceversa.

3.2. Interpretación del problema

La secuencia de encendido de los LEDs es la misma del problema A salvo que ahora al presionar el pulsador el sentido del recorrido se invierte, no es necesario que llegue al final del recorrido para invertir la secuencia.

Para analizar cómo se detecta un pulso en PC0 veremos que tendremos que configurar el registro de direcciones del puerto C para el bit 0 (PTBDD0) en cero para que funcione como entrada y para leer la entrada utilizaremos el registro de datos del puerto C para el bit 0 (PTCD0).

En la figura 4 vemos el circuito que está implementado la lógica de control para una terminal bidireccional, en nuestro caso nos interesa ver cuando funciona como entrada, al establecer la configuración mencionada (PTCDD0 en cero) se va seleccionar la lectura de la parte que viene de la entrada además el buffer de tres estados se deshabilita por lo tanto escribir en el registro PTCD0 está deshabilitado, quedando disponible la entrada directamente que pasa por un trigger no inversor Schmitt Trigger luego por un sincronizador que está sincronizado con el reloj del sistema, esto es justamente para tener una sincronización con el reloj interno y poder leer en el instante correcto en el cual el reloj interno está funcionando. Luego habiendo establecido con el registro de direcciones en cero en el multiplexor se selecciona como entrada D0 por lo tanto cuando haga una lectura del registro de datos (PTCD0) voy a estar leyendo la entrada que aparece en el pin.

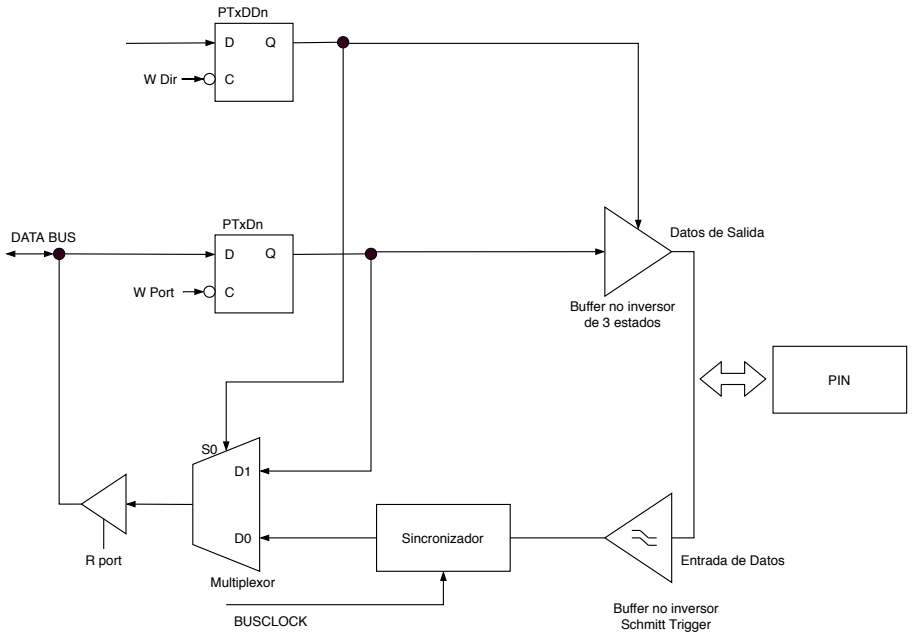


Figura 4: Lógica de control para una terminal bidireccional de un bit

Al introducir un pulsador mecánico además de configurar como entrada PTC0 se tiene que configurar una resistencia pull-up interna para que en estado normalmente abierto el pulsador tenga una tensión en alto pero al presionarlo cambia a una tensión baja y así detectar que se a pulsado utilizando el registro PTCD0.

En la figura 5 se puede ver cómo se realiza la conexión entre el pulsador y el microcontrolador.

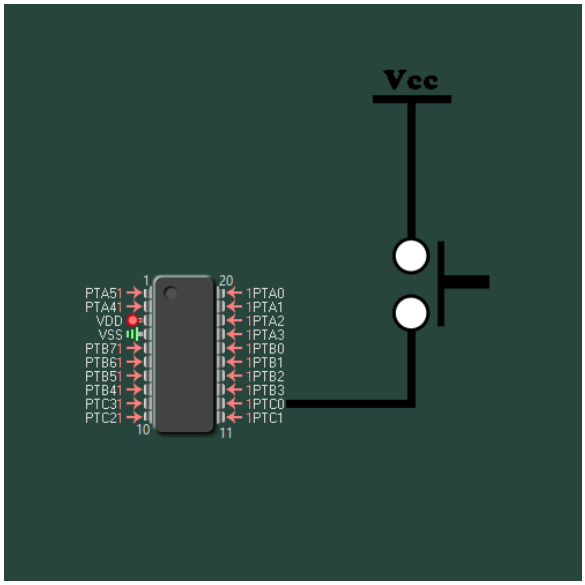


Figura 5: Conexión del microcontrolador con el pulsador mecánico

Ademas el pulsador mecánico nos trae el problema del rebote, este problema surge al variar rápidamente la tensión entre los bornes del pulsador al rebotar por el efecto de elasticidad del material hasta llegar al estado de reposo. Esto provoca que se tenga un momento de ambigüedad con respecto al estado lógico del periférico, llevando posiblemente a lecturas erróneas.

El problema del rebote se puede solucionar con el uso de hardware o software. Trabajaremos con la solución por software la cual consiste en hacer una espera hasta que la señal se estabilice. Cuando pulsamos el pulsador, desde el software se detecta que se presionó, si es cierto se espera un tiempo (este tiempo depende del pulsador ya sea por el tipo o el desgaste que posea) el cual permite que la señal se estabilice, luego se realiza un bucle mientras se mantenga presionado el pulsador, cuando se deja de presionar nuevamente se espera un tiempo hasta que se estabilice la señal y por último se realizan las operaciones que se deban realizar (ya que al precionar el boton se espera que se realice alguna acción).

3.3. Resolución del problema

Aplicamos la misma configuración del PORTB que en el problema A y utilizamos la misma función recorrido de ese problema además al pin TPC0 lo debemos configurar como entrada y con una resistencia de pull-up (por lo tanto en estado normalmente abierto estaría en estado en alto) para el tiempo de espera para que se estabilice la señal utilizamos 15 ms.

Para implementar el cambio de sentido de visualización de los LEDs se define una variable en el programa principal que se llama 'sentido' la cual indica el sentido de recorrido (en cero el sentido es igual que en el problema A, en cambio establecida en uno se invierte el sentido) esto se puede ver en la figura 6.

El pseudocódigo del programa principal sería el siguiente:

```
1  Función main()
2          Inicializo variables necesarias (ej. index = 1, sentido=0)
3          Configurar como salida los pines del puerto B
4          Configurar High Drive los pines del puerto B
5          Configurar como entrada el pin 0 del puerto C
6          Habilitar la resistencia de pull−up para el pin 0 del puerto C
7          Inicializar las salidas del puerto B
8          Repetir indefinidamente
9                  recorrido(index)
10                 Si se presiono el pulsador
11                         Esperar 15 ms aproximadamente
12                         Mientras esté pulsado no hacer nada
13                         Esperar 15 milisegundos aproximadamente
14                         Se invierte el sentido del recorrido
15                 Si sentido es igual a cero
16                         Incremento index en uno
17                         Si index es mayor o igual a 9
18                                 index se establece en 1
19                 Sino
20                         Decremento index en uno
21                         Si index es igual a cero
22                                 index se establece en 8
23 Fin main()
```

Pseudocódigo del main().

4. Problema D

4.1. Consigna

Saque conclusiones sobre las ventajas y desventajas de este tipo de retardo y cómo afecta el tiempo de respuesta del programa.

4.2. Resolución del problema

Una de las ventajas de implementar retardo por software es que es fácil de implementar (ya que es decrementar o incrementar una variable) y si los temporizadores están en uso para otras tareas (por ejemplo relacionadas con el hardware) pueden llegar a no estar libres para configurarse entonces se podría usar un retraso por software. La desventaja es, que se logra este retraso a costa de ocupar tiempo de CPU que se podría utilizar para realizar otras operaciones útiles además por como esta implementado el sistema se bloquea en el retardo pudiendo causar una mala experiencia en el usuario, si el sistema es lento al responder a eventos de cambio o se bloquea en un estado. En los próximos trabajos, se buscara mejorar esta situación.

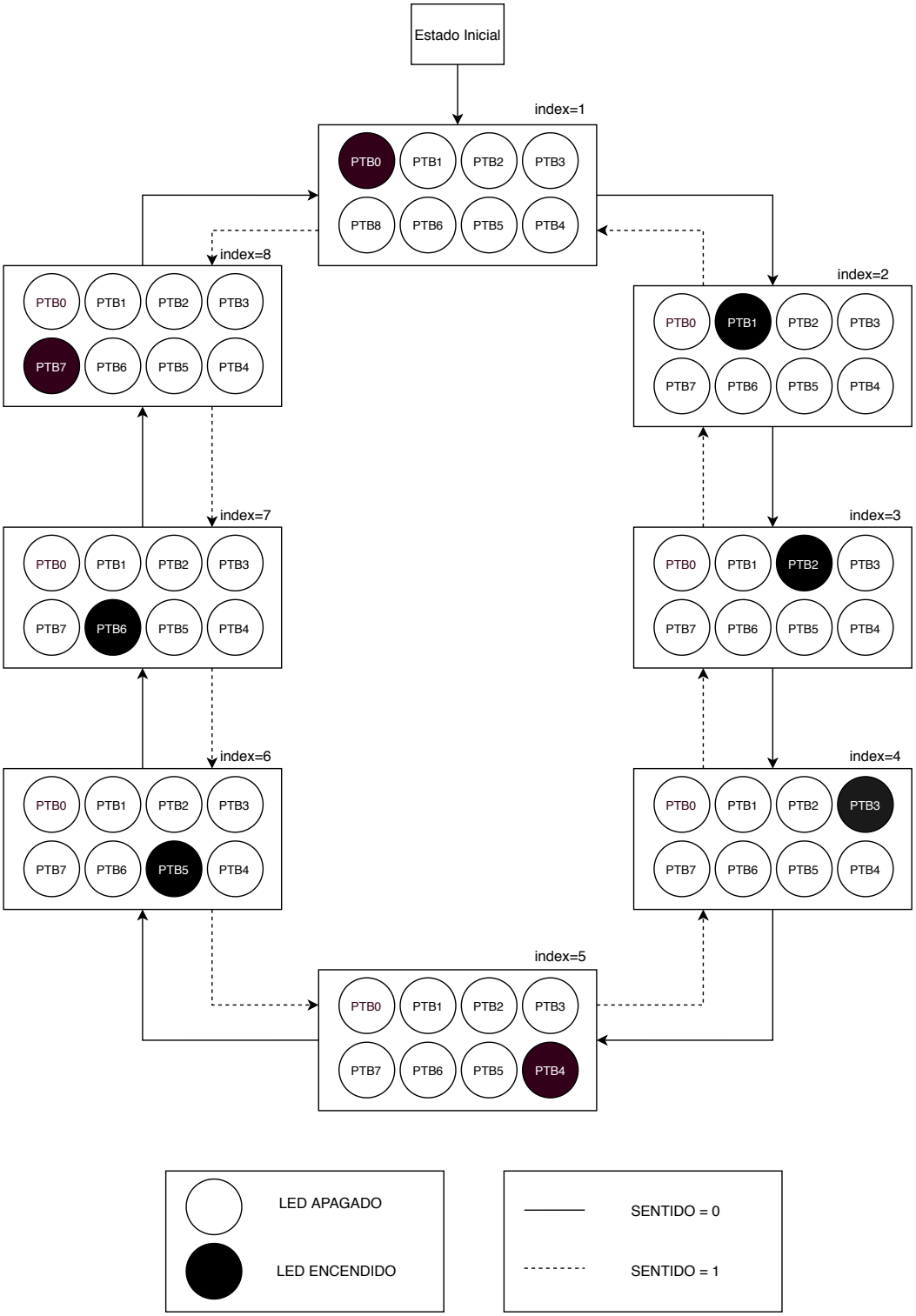


Figura 6: La variable sentido puede cambiar el orden de la secuencia

5. Apéndice: Código fuente

5.1. main.h

```
1  #ifndef TYPE_H_
2  #define TYPE_H_
3
4  typedef signed char int8_t;
5  typedef unsigned char uint8_t;
6  typedef short int int16_t;
7  typedef unsigned short int uint16_t;
8  typedef long int int32_t;
9  typedef unsigned long int uint32_t;
10 typedef long long int int64_t;
11 typedef unsigned long long uint64_t;
12
13 #endif // TYPE H
```

5.2. libtime.h

```
1  #ifdef LIBTIME_H_
2  #define LIBTIME_H_
3  #include "main.h"
4
5  void delay(uint16_t ms);
6
7  #endif // LIBTIME H
```

5.3. libtime.c

```
1  #include "main.h"
2
3  void mili_seg(){
4          uint16_t i = 362;
5          for(i;i>0;i--);
6  }
7
8  void delay(uint16_t ms){
9          uint16_t i;
10         for(i=0;i<ms;i++){
11                 mili_seg();
12         }
13 }
```

5.4. main.c problema A

```
1  #include <hidef.h> /* for EnableInterrupts macro */
2  #include "derivative.h" /* include peripheral declarations */
3  #include "main.h"
4  #include "lib_time.h"
5
6  #ifdef __cplusplus
7  extern "C"
8  #endif
9  const TIME_DELAY = 15;
10 void MCU_init(void); /* Device initialization function declaration */
11 void recorrido(uint8_t);
12 void inicializacion_recorrido();
13
14 void main(void) {
15
16   // Declaracion de variables
17   uint8_t index = 1;
18   MCU_init(); /* call Device Initialization */
19
20   inicializacion_recorrido();
21   for(;;) {
22            recorrido(index);
23            index++;
24            if(index==9) index=1;
25     /* __RESET_WATCHDOG(); By default COP is disabled with device init. When
            enabling, also reset the watchdog. */
26   } /* loop forever */
27   /* please make sure that you never leave main */
28 }
29
30 void inicializacion_recorrido(){
31         //Configuro como salida el puerto B
32         PTBDD = 0xFF;
33         //Configuro High Drive al puerto B (capacidad para entregar hasta 20mA)
34         PTBDS = 0xFF;
35         //Se inicializa las salidas del puerto B
36         PTBD = 0x00;
37 }
38
39
40 void recorrido(char index){
41         switch(index){
42         case 1: PTBD = 0x01; break;
43         case 2: PTBD = 0x02; break;
44         case 3: PTBD = 0x04; break;
45         case 4: PTBD = 0x08; break;
46         case 5: PTBD = 0x10; break;
47         case 6: PTBD = 0x20; break;
48         case 7: PTBD = 0x40; break;
49         case 8: PTBD = 0x80; break;
```

```
50          }
51          delay(TIME_DELAY);
52  };
```

5.5. main.c problema C

```
1   #include <hidef.h> /* for EnableInterrupts macro */
2   #include "derivative.h" /* include peripheral declarations */
3   #include "main.h"
4   #include "lib_time.h"
5
6   #ifdef __cplusplus
7   extern "C"
8   #endif
9   const TIME_DELAY = 15;
10  const TIME_DELAY_LED = 40;
11  void MCU_init(void); /* Device initialization function declaration */
12  void recorrido(uint8_t);
13  void inicializacion_recorrido();
14
15  void main(void) {
16
17    // Declaracion de variables
18    uint8_t index = 1;
19    uint8_t sentido = 0;
20    MCU_init(); /* call Device Initialization */
21
22    inicializacion_recorrido();
23    for(;;) {
24            recorrido(index);
25            // Pulsador
26            if(PTCD_PTCD0==0){
27                    delay(TIME_DELAY_LED);
28                    while(PTCD_PTCD0==0);
29                            delay(TIME_DELAY);
30                            //Operacion a realizar al precionar el boton
31                            sentido =~ sentido;
32                    }
33                    // Control del sentido del recorrido
34                    if(sentido==0){
35                            index++;
36                            if(index == 9)
37                                    index = 1;
38                            }else{
39                                    index--;
40                                    if(index == 0)
41                                            index = 8;
42                    }
43
```

```
44      /* __RESET_WATCHDOG(); By default COP is disabled with device init. When
            enabling, also reset the watchdog. */
45    } /* loop forever */
46    /* please make sure that you never leave main */
47  }
48
49  void inicializacion_recorrido(){
50          //Configuro como salida el puerto B
51          PTBDD = 0xFF;
52          //Configuro High Drive al puerto B (capacidad para entregar hasta 20mA)
53          PTBDS = 0xFF;
54          //Se inicializa las salidas del puerto B
55          PTBD = 0x00;
56          /**
57           * Configuracion del pulsador
58          */
59          //Se configura PTC0 como entrada
60          PTCDD_PTCDD0 = 0;
61          //Se habilita la resistencia de pull-up
62          PTCPE_PTCPE0 = 1;
63  }
64
65
66  void recorrido(char index){
67          switch(index){
68          case 1: PTBD = 0x01; break;
69          case 2: PTBD = 0x02; break;
70          case 3: PTBD = 0x04; break;
71          case 4: PTBD = 0x08; break;
72          case 5: PTBD = 0x10; break;
73          case 6: PTBD = 0x20; break;
74          case 7: PTBD = 0x40; break;
75          case 8: PTBD = 0x80; break;
76          }
77          delay(TIME_DELAY);
78  };
```

6. Tablas

Cuadro 1: Sección de tabla extraída de la página 294, Ficha de datos MC9S08SH8.

C	Characteristic		Symbol	Condition	Min	Typ[1]	Max	Unit
—	Operating voltage		V_{DD}	—	2.7	—	5.5	V
C	Output high voltage	All I/O pins, low-drive strength	V_{OH}	5 V, I_{Load} = –4 mA	V_{DD} – 1.5	—	—	V
P				5 V, I_{Load} = –2 mA	V_{DD} – 0.8	—	—	
C				3 V, I_{Load} = –1 mA	V_{DD} – 0.8	—	—	
C		All I/O pins, high-drive strength		5 V, I_{Load} = –20 mA	V_{DD} – 1.5	—	—	
P				5 V, I_{Load} = –10 mA	V_{DD} – 0.8	—	—	
C				3 V, I_{Load} = –5 mA	V_{DD} – 0.8	—	—	
C	Output high current	Max total I_{OH} for all ports	I_{OHT}	V_{OUT} < V_{DD}	0	—	–100	mA
C	Output low voltage	All I/O pins low-drive strength	V_{OL}	5 V, I_{Load} = 4 mA	—	—	1.5	V
P				5 V, I_{Load} = 2 mA	—	—	0.8	
C				3 V, I_{Load} = 1 mA	—	—	0.8	
C		All I/O pins high-drive strength		5 V, I_{Load} = 20 mA	—	—	1.5	
P				5 V, I_{Load} = 10 mA	—	—	0.8	
C				3 V, I_{Load} = 5 mA	—	—	0.8	
	Output low current	Max total I_{OL} for all ports	I_{OLT}	V_{OUT} > V_{SS}	0	—	100	mA