Universidad de Costa Rica

Escuela de Ingeniería Eléctrica

IE-0624 Laboratorio de Microprocesadores

Laboratorio 1

José Antonio Ramos Pereira, B86485

Grupo 01

Prof. MSc. Marco Villalta Fallas.

https://github.com/JAR1224/Laboratorio_1_Jose_Ramos

Índice de contenidos

Índice de diagramas

1. Introducción

1.1. Resumen

La elaboración del proyecto consistió en plantear una solución utilizando un microcontrolador PIC12F683 y otros componentes electrónicos para simular un juego de Bingo. La aplicación debía poder generar números aleatorios entre 0-99 para poder mostrar valores de 16 bolas en dos LEDS de 7 segmentos. Luego de mostrar 16 números distintos, la aplicación debía mostrar el valor 99 en los LEDS parpadeando para indicar el fin del juego.

Los retos principales para la elaboración de la solución al problema dado fueron los siguientes:

1. Manejo de 14 pines de los LEDS teniendo solo 6 pines de IO en el microcontrolador
2. Elaboración de números aleatorios
3. Cantidad limitada de RAM para uso de memoria de datos en tiempo de ejecución

1.2. Conclusiones

Para poder superar los retos, se utilizó la siguiente metodología:

1. Para extender la capacidad de IO del microcontrolador, se utilizaron demultiplexores y registros desplazantes.
2. Para generar números aleatorios se utilizó el método de contadores por software
3. Para no sobrepasar la utilización máxima de RAM, se trató de almacenar la mayor cantidad de información posible dentro de las instrucciones mismas del programa para maximizar el uso de memoria de programa y minimizar el uso de memoria de datos. Esto debido a que la memoria de programa es más grande que la memoria RAM para el PIC12F683.

Se concluyó que por medio de las metodologías mencionadas anteriormente, se pudo implementar una solución al problema planteado que cumple con todos los requisitos de diseño requeridos por el enunciado.

2. Nota Teórica

2.1. Microcontrolador

Características generales

Las características generales son las siguientes [1]:

- CPU con arquitectura RISC
- Maneja 35 instrucciones
- Ciclo de instrucción de 200 ns

Device	Program Memory	Data Memory		I/O	10-bit A/D (ch)	Comparators	Timers 8/16-bit
	Flash (words)	SRAM (bytes)	EEPROM (bytes)				
PIC12F683	2048	128	256	6	4	1	2/1

Características generales PIC12F683 2.1: Características generales PIC12F683 [1]

Diagrama de bloques PIC

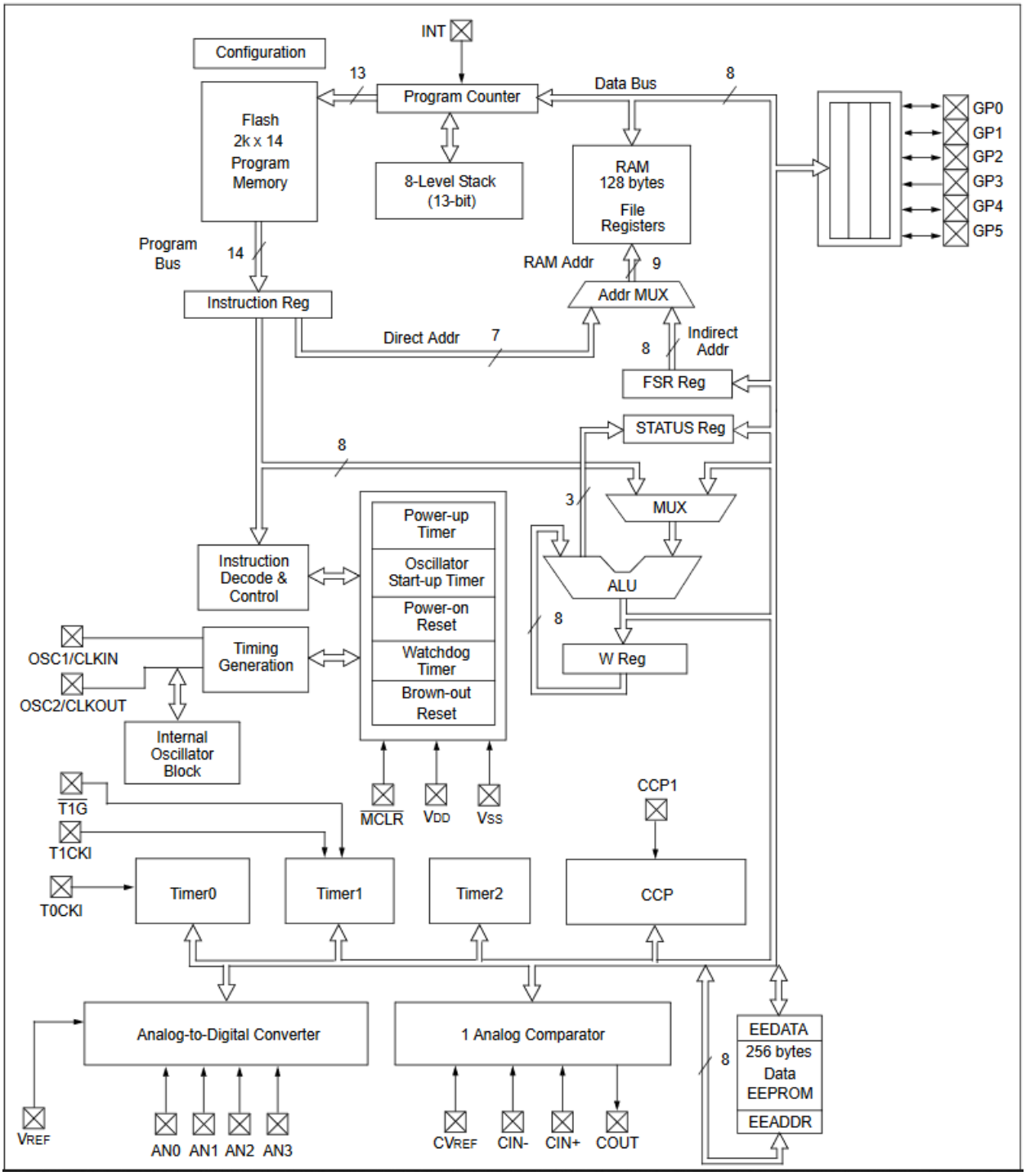


Diagrama de bloques PIC12F683 2.2: Diagrama de bloques PIC [1]

Diagrama de pines

8-Pin Diagram (PDIP, SOIC)

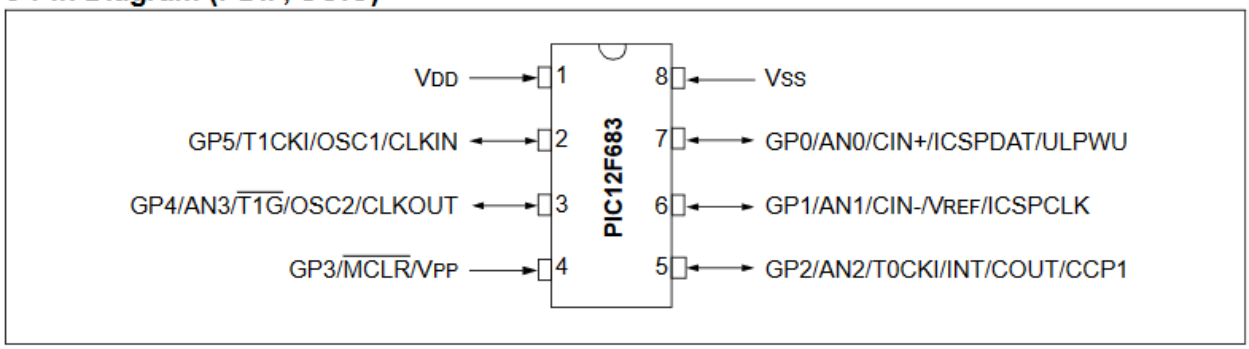


Diagrama de pines PIC12F683 2.3: Diagrama de pines [1]

Características eléctricas

15.0 ELECTRICAL SPECIFICATIONS

Absolute Maximum Ratings(†)

Ambient temperature under bias	-40° to +125°C
Storage temperature	-65°C to +150°C
Voltage on VDD with respect to VSS	-0.3V to +6.5V
Voltage on MCLR with respect to VSS	-0.3V to +13.5V
Voltage on all other pins with respect to VSS	-0.3V to (VDD + 0.3V)
Total power dissipation(1)	800 mW
Maximum current out of VSS pin	95 mA
Maximum current into VDD pin	95 mA
Input clamp current, IIK (VI < 0 or VI > VDD)	± 20 mA
Output clamp current, IOK (Vo < 0 or Vo >VDD)	± 20 mA
Maximum output current sunk by any I/O pin	25 mA
Maximum output current sourced by any I/O pin	25 mA
Maximum current sunk by GPIO	90 mA
Maximum current sourced GPIO	90 mA

Note 1: Power dissipation is calculated as follows: $P_{DIS} = V_{DD} \times \{I_{DD} - \sum I_{OH}\} + \sum \{(V_{DD} - V_{OH}) \times I_{OH}\} + \sum (V_{OL} \times I_{OL})$.

Características eléctricas PIC12F683 2.4: Características eléctricas [1]

Resumen de instrucciones

TABLE 13-2: PIC12F683 INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb			LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C, DC, Z	**1, 2**
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	**1, 2**
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z	**2**
CLRW	–	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	**1, 2**
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	**1, 2**
DECFSZ	f, d	Decrement f, Skip if 0	**1(2)**	00	1011	dfff	ffff		**1, 2, 3**
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	**1, 2**
INCFSZ	f, d	Increment f, Skip if 0	**1(2)**	00	1111	dfff	ffff		**1, 2, 3**
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	**1, 2**
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	**1, 2**
MOVWF	f	Move W to f	1	00	0000	lfff	ffff		
NOP	–	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	**1, 2**
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	**1, 2**
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C, DC, Z	**1, 2**
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		**1, 2**
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	**1, 2**
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		**1, 2**
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		**1, 2**
BTFSC	f, b	Bit Test f, Skip if Clear	**1 (2)**	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	**1 (2)**	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C, DC, Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call Subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	–	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}$, $\overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	–	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	–	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	–	Go into Standby mode	1	00	0000	0110	0011	$\overline{TO}$, $\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C, DC, Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

Resumen instrucciones PIC12F683 2.5: Resumen de instrucciones [1]

Registros

REGISTER 4-1: GPIO: GENERAL PURPOSE I/O REGISTER

U-0	U-0	R/W-x	R/W-0	R-x	R/W-0	R/W-0	R/W-0
—	—	GP5	GP4	GP3	GP2	GP1	GP0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-6 **Unimplemented**: Read as '0'

bit 5-0 **GP<5:0>**: GPIO I/O Pin bit
1 = Port pin is > VIH
0 = Port pin is < VIL

Registro GPIO 2.6: Registros GPIO [1]

REGISTER 4-2: TRISIO GPIO TRI-STATE REGISTER

U-0	U-0	R/W-1	R/W-1	R-1	R/W-1	R/W-1	R/W-1
—	—	TRISIO5(2,3)	TRISIO4(2)	TRISIO3(1)	TRISIO2	TRISIO1	TRISIO0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-6 **Unimplemented**: Read as '0'

bit 5:4 **TRISIO<5:4>:** GPIO Tri-State Control bit
1 = GPIO pin configured as an input (tri-stated)
0 = GPIO pin configured as an output

bit 3 **TRISIO<3>:** GPIO Tri-State Control bit
Input only

bit 2:0 **TRISIO<2:0>:** GPIO Tri-State Control bit
1 = GPIO pin configured as an input (tri-stated)
0 = GPIO pin configured as an output

Note 1: TRISIO<3> always reads '1'.
2: TRISIO<5:4> always reads '1' in XT, HS and LP OSC modes.
3: TRISIO<5> always reads '1' in RC and RCIO and EC modes.

Registro TRISIO 2.7: Registro TRISIO [1]

Inicializar GPIO

EXAMPLE 4-1: INITIALIZING GPIO

```
BANKSEL  GPIO       ;
CLRF     GPIO       ;Init GPIO
MOVLW    07h        ;Set GP<2:0> to
MOVWF    CMCON0     ;digital I/O
BANKSEL  ANSEL      ;
CLRF     ANSEL      ;digital I/O
MOVLW    0Ch        ;Set GP<3:2> as inputs
MOVWF    TRISIO     ;and set GP<5:4,1:0>
                    ;as outputs
```

Inicializar GPIO 2.8: Inicializar GPIO [1]

Lo que se muestra en el ejemplo proporcionado en las hojas de datos es que para usar los pines como GPIO hay que escribir un 0x00 en el registro ANSEL, un 0x07 en el registro CMCON0 y activar los pines escribiendo un 1 en los espacios de bits correspondientes del registro TRISIO.

2.2. Componentes Electrónicos Complementarios

Registro desplazante CD4094

CHARACTERISTIC	V_{DD} (V)	LIMITS		UNITS
		MIN.	MAX.	
Supply-Voltage Range (For T_A=Full Package-Temperature Range)		3	18	V
Data Setup Time, t_S	5 10 15	125 55 35	– – –	ns
Clock Pulse Width, t_W	5 10 15	200 100 83	– – –	ns
Clock Input Frequency, f_{CL}	5 10 15	dc	1.25 2.5 3	MHz
Clock Input Rise or Fall time, t_rCL, t_fCL:*	5 10 15	–	15 5 5	μs
Strobe Pulse Width, t_W	5 10 15	200 80 70	– – –	ns

Características temporales 2.9: Características temporales CD4094 [2]

Se observa que para una tensión de operación de 5V, todas las condiciones temporales son, a lo sumo, de 200ns. Como el ciclo de instrucción del PIC12F683 es de 200ns, entonces no es necesario implementar retardos adicionales dentro del código de la aplicación para satisfacer las condiciones temporales del cd4094.

Demux 4696 1:2 x3

Demux-4696	
itemtype	Demux
id	Demux-4696
Show id	false
Input High V	2.5
Input Low V	2.5
Input Imped	1e+14
Out High V	5
Out Low V	0
Out Imped	40

Características generales 2.10: Demux 4696

Leds 7 seg 4515 x2

Name	Value
id	Seven Segment-...
Show id	false
Color	yellow
NumDisplays	1
CommonCathode	true
Vertical Pins	false
Threshold	2.4
MaxCurrent	0.02
Resistance	1

Características generales 2.11: Leds 7 seg 4515

Resistencias

- 1x 50 kΩ
- 1x 10 kΩ
- 14x 100Ω

Capacitores

- 1x 100 nF

Amplificador Operacional

- 1x LMC7101

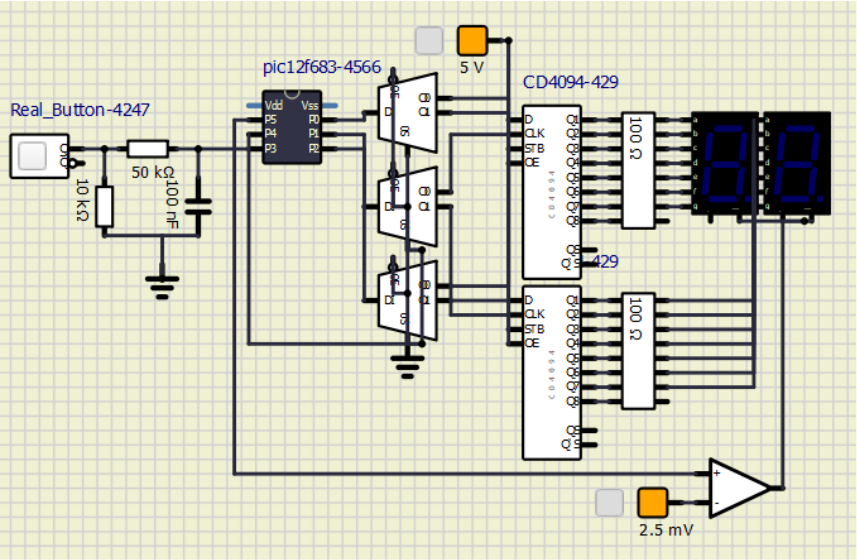
2.3. Precios

- Op Amp LMC7101: $0.66
- 2x Led 7 segmentos LSHD-5601: $1.28
- Registro desplazante CD4094: $1.3
- Demux CD40257BM: $1.57
- 16x Resistencias: $0.00
- Capacitor: $0.25
- PIC12F683: $1.92
- Total: **$8.26**

Obtenido de: [3]

2.4. Diseño del circuito

Esquemático



Esquemático 2.12: Obtenido del simulador

El circuito consta de 3 partes principales.

1) Supresor de rebotes

Al pin 3 se le conecta un supresor de rebotes junto con un botón. El conjunto de resistencias con capacitores crean un filtro pasa bajos que suprime las señales eléctricas oscilantes que envía el botón al ser presionado. Para elegir los valores del capacitor y las resistencias hay que tomar en cuenta que el tiempo de rebotes de 1 botón es de aproximadamente 1ms.

Al elegir una resistencia de 10kΩ y 50kΩ, la resistencia que ve el capacitor al cargarse y descargarse es aproximadamente 60kΩ.

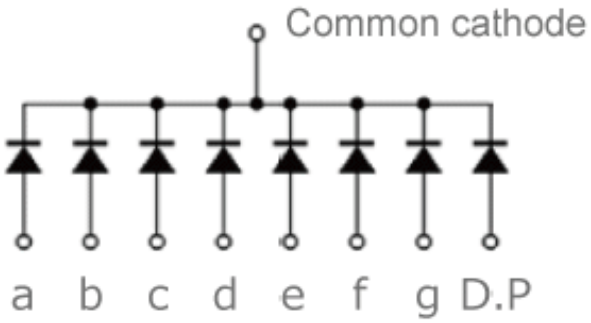
Se sabe que un capacitor dura 4 constantes de tiempo para cargarse más de 95 % de su capacidad; donde la constante de tiempo está dada por el producto de la capacitancia y la resistencia:

$$\begin{aligned} 4RC &= 4 \times 60k\Omega \times 100nF \\ &= 24ms \end{aligned} \tag{1}$$

Entonces estos valores de capacitancia y resistencia son suficientes para suprimir los rebotes y lo suficientemente bajos para ser imperceptible por el usuario.

2) Habilitador de cátodo común

El pin 5 es utilizado para poner en alto o en bajo el cátodo común de los LEDS.



Configuración cátodo común 2.13: LEDS cátodo común

Sin embargo, según las especificaciones eléctricas de los LEDS y los valores de resistencia utilizados, estos van a conducir 18 mA de corriente. Esto significa que en el caso donde se muestra un 88 en los LEDS, y todos los segmentos estén habilitados, la suma de corrientes sería de aproximadamente 252mA. Esta cifra supera la cantidad máxima de amperios que puede manejar los pines del microcontrolador (20mA), y es por eso que el pin 5 se conecta a un amplificador operacional con suficiente capacidad de corriente como para drenar los posibles 252mA.

3) Paralelización de salidas IO

Los pines 0,1,2 y 4 se usan para paralelizar datos seriales provenientes de los IO. Más específicamente, se utiliza para mandar el byte de información que ocupa los LEDS de 7 segmentos para mostrar un número de 0 al 9.

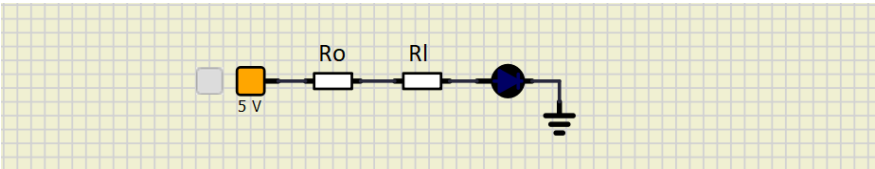
El pin 4 maneja el selector de los 3 decodificadores, seleccionando a cual de los 2 registros desplazantes se le va a mandar la información, y por ende, a cual LED se le va a cambiar el valor mostrado.

El pin 0 es el que manda la codificación de cada número del 0 al 9 a ser mostrado en los LEDS.

El pin 1 manda la señal de clk que carga en los registros desplazantes, el valor pasado por el pin 0.

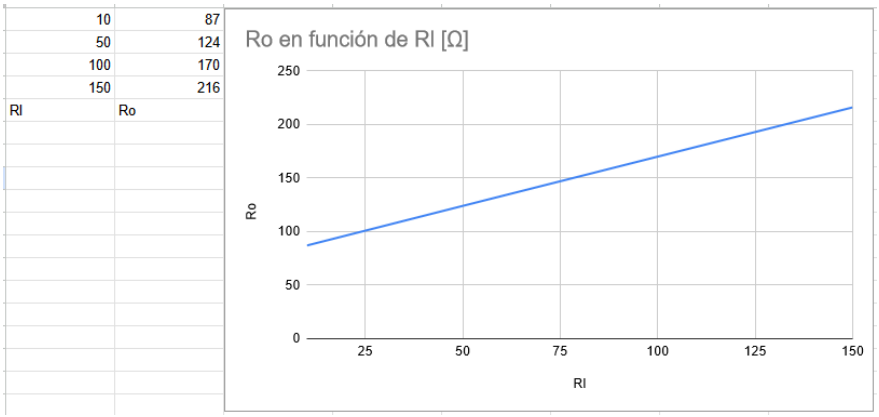
El pin 2 manda la señal de STROBE, para que los datos paralelizados sean reflejados en la salida del registro al mismo tiempo en vez de uno por uno. Esto logra que el cambio de números en los LEDS sea instantáneo.

En la salida de los registros desplazantes se le conectan resistencias de 100Ω para restringir la cantidad de corriente que pasa por los LEDS. Para poder elegir el valor ideal de estas resistencias, se tiene que cumplir que la tensión de alimentación de 5V dividido entre la resistencia total ($R_l + R_o$), de una corriente menor a 20mA para evitar quemar los LEDS; donde R_l son las resistencias conectadas a la salida de los registros y R_o es la impedancia de salida de los pines del registro.



Circuito eq. LEDS 2.14: Circuito eq. LEDS

Como la hoja del fabricante de los registros desplazantes no incluye la impedancia de salida, se tuvo que calcularla experimentalmente. Se encontró que la impedancia de salida del dispositivo no es constante, y que esta depende del valor de la resistencia R_l utilizada.



R_o en función de R_l 2.15: Impedancia de salida del CD4094

Estos datos se obtuvieron experimentalmente conectando diferentes valores de resistencias R_l a los registros desplazantes, midiendo la corriente y aplicando ley de Ohm para calcular la resistencia total $R_l + R_o$.

Como se tiene que cumplir que:

$$\begin{aligned} \frac{5V}{R_l + R_o} &< 20mA \\ 250\Omega &< R_l + R_o \end{aligned} \tag{2}$$

Entonces, según la rabla 3.1, eligiendo R_l=100Ω, se obtiene una impedancia de salida de 170Ω, y una resistencia total de 270Ω > 250Ω.

2.5. Temas de laboratorio

Generación de números aleatorios

El método utilizado en este laboratorio fue el de generación de números aleatorios por medio de contadores por software. Este método consiste en incluir un incrementador de contero dentro del programa y devolver el valor que este tenga en algún momento dado cuando suceda algún evento externo al microprocesador (como el usuario presionando un botón). El número resultante es pseudoaleatorio ya que el tiempo de ejecución de las instrucciones se

mide en nanosegundos, lo que significa que el usuario no tiene la capacidad de prever que valor está guardado en el contero en el momento de ser invocado.

3. Desarrollo y análisis

3.1. Programa

Código del programa

```
#include <pic14/pic12f683.h>

#define time 1

typedef unsigned int word ;
word __at 0x2007 __CONFIG = (_CPD_OFF, _CP_OFF, _BOREN_OFF, \
                             _MCLRE_OFF, _PWRTE_OFF, _WDTE_OFF) ;

void send_byte(unsigned int cod_7seg);
unsigned int valido(unsigned int valor);
unsigned int rand_wait();
unsigned int count_1=0;
unsigned int count_2=0;
unsigned int blink=0;

unsigned int banderas = 0b00000000;
unsigned int historial[16]={0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF, \
                            0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF};

void main(void)
```

```
{
        //Inicializar GPIO
        TRISIO = 0b00001000; //P0,1,2,4,5 como salidas, P3 como entrada
        GPIO = 0b00000000; //Pines empiezan en bajo
        ADCON0=0x00;
        ANSEL=0x00;
        CMCON0 = 0x07;

        unsigned int tx = 0x00;


        while ( 1 )
        {

                tx = rand_wait();
                GP4=0;
                if ((tx & 0x0F) == 0x00) {
                        send_byte(0b11111100);
                } else if ((tx & 0x0F) == 0x01) {
                        send_byte(0b01100000);
                } else if ((tx & 0x0F) == 0x02) {
                        send_byte(0b11011010);
                } else if ((tx & 0x0F) == 0x03) {
                        send_byte(0b11110010);
                } else if ((tx & 0x0F) == 0x04) {
                        send_byte(0b01100110);
                } else if ((tx & 0x0F) == 0x05) {
                        send_byte(0b10110110);
```

```
} else if ((tx & 0x0F) == 0x06) {
        send_byte(0b10111110);
} else if ((tx & 0x0F) == 0x07) {
        send_byte(0b11100000);
} else if ((tx & 0x0F) == 0x08) {
        send_byte(0b11111110);
} else if ((tx & 0x0F) == 0x09) {
        send_byte(0b11100110);
}
```

GP4=1;

```
if ((tx & 0xF0) == 0x00) {
        send_byte(0b11111100);
} else if ((tx & 0xF0) == 0x10) {
        send_byte(0b01100000);
} else if ((tx & 0xF0) == 0x20) {
        send_byte(0b11011010);
} else if ((tx & 0xF0) == 0x30) {
        send_byte(0b11110010);
} else if ((tx & 0xF0) == 0x40) {
        send_byte(0b01100110);
} else if ((tx & 0xF0) == 0x50) {
        send_byte(0b10110110);
} else if ((tx & 0xF0) == 0x60) {
        send_byte(0b10111110);
} else if ((tx & 0xF0) == 0x70) {
        send_byte(0b11100000);
} else if ((tx & 0xF0) == 0x80) {
```

```
            send_byte(0b11111110);
        } else if ((tx & 0xF0) == 0x90) {
            send_byte(0b11100110);
        }

        if ( (banderas & 0x02) == 0x02 ) {
                    banderas &= 0x11111101;
                    historial[0] = 0xFF;
                    historial[1] = 0xFF;
                    historial[2] = 0xFF;
                    historial[3] = 0xFF;
                    historial[4] = 0xFF;
                    historial[5] = 0xFF;
                    historial[6] = 0xFF;
                    historial[7] = 0xFF;
                    historial[8] = 0xFF;
                    historial[9] = 0xFF;
                    historial[10] = 0xFF;
                    historial[11] = 0xFF;
                    historial[12] = 0xFF;
                    historial[13] = 0xFF;
                    historial[14] = 0xFF;
                    historial[14] = 0xFF;
        }


}
```

```
}


void send_byte(unsigned int cod) {

        for (unsigned int c = 0 ; c < 8 ; c++) {
                GP0=cod & 0x01;
                GP1=1;
                GP1=0;
                cod>>=1;
        }
        GP2=1;
        GP2=0;
}

unsigned int rand_wait() {

        unsigned int rand = 0x00;

        while(1) {

                if ( (rand & 0x0F) < 0x09) {
                        rand+=0x01;
                } else if ((rand & 0xF0)>>4 < 9) {
                        rand = (rand+0x10) & 0xF0;
                } else {
                        rand = 0x00;
                }
```

```
        if ( ((banderas & 0x01) == 0x00) &&  (GP3==1) \
                                    && (valido(rand)==1) ) {
                banderas |= 0x01;
                if ( (banderas & 0x02) == 0x02 ) {
                        GP5=0;
                        blink=1;
                        return 0x99;
                }
                GP5=0;
                blink=0;
                return rand;
        } else if ( ((banderas & 0x01)==0x01) &&  (GP3==0) ) {
                banderas &= 0b11111110;
        }

        count_1+=1;
        if (count_1==0) {
                count_2+=1;
        }
        if (count_2 == time) {
                count_2=0;
                if ( blink == 1 ) {
                        GP5=~GP5;
                }
        }

}
```

```
}

unsigned int valido(unsigned int valor) {
        for (unsigned int c = 0; c < 16; c++) {
                if ( (historial[c] == 0xFF) ) {
                        historial[c] = valor;
                        return 1;
                } else if ( historial[c] == valor ) {
                        return 0;
                }
        }
        banderas |= 0x02;
        return 1;
}
```

Descripción del programa

El programa consiste de una función main(), y 3 funciones auxiliares: send_byte(), rand_wait() y valido().

void send_byte() recibe una variable tipo byte llamada cod, cuyo valor es la codificación para LEDS de 7 segmentos del valor aleatorio a mostrarse en las pantallas. El byte se manda repitiendo la siguiente secuencia 8 veces, para los 8 pines de los LEDS de 7 segmentos: se coloca el primer bit a ser mandado en el pin 0 del microcontrolador por medio de la máscara 0x01 aplicada sobre la variable cod. Luego, se escribe un 1 seguido de un 0 en el pin 1 del microcontrolador que está conectado al pin de clk del registro, cargando en el, el valor del primer bit. Por último, se desplaza a la derecha el valor de cod en 1 bit y se repiten los pasos anteriores hasta que todos los bits hayan sido cargados en el registro.

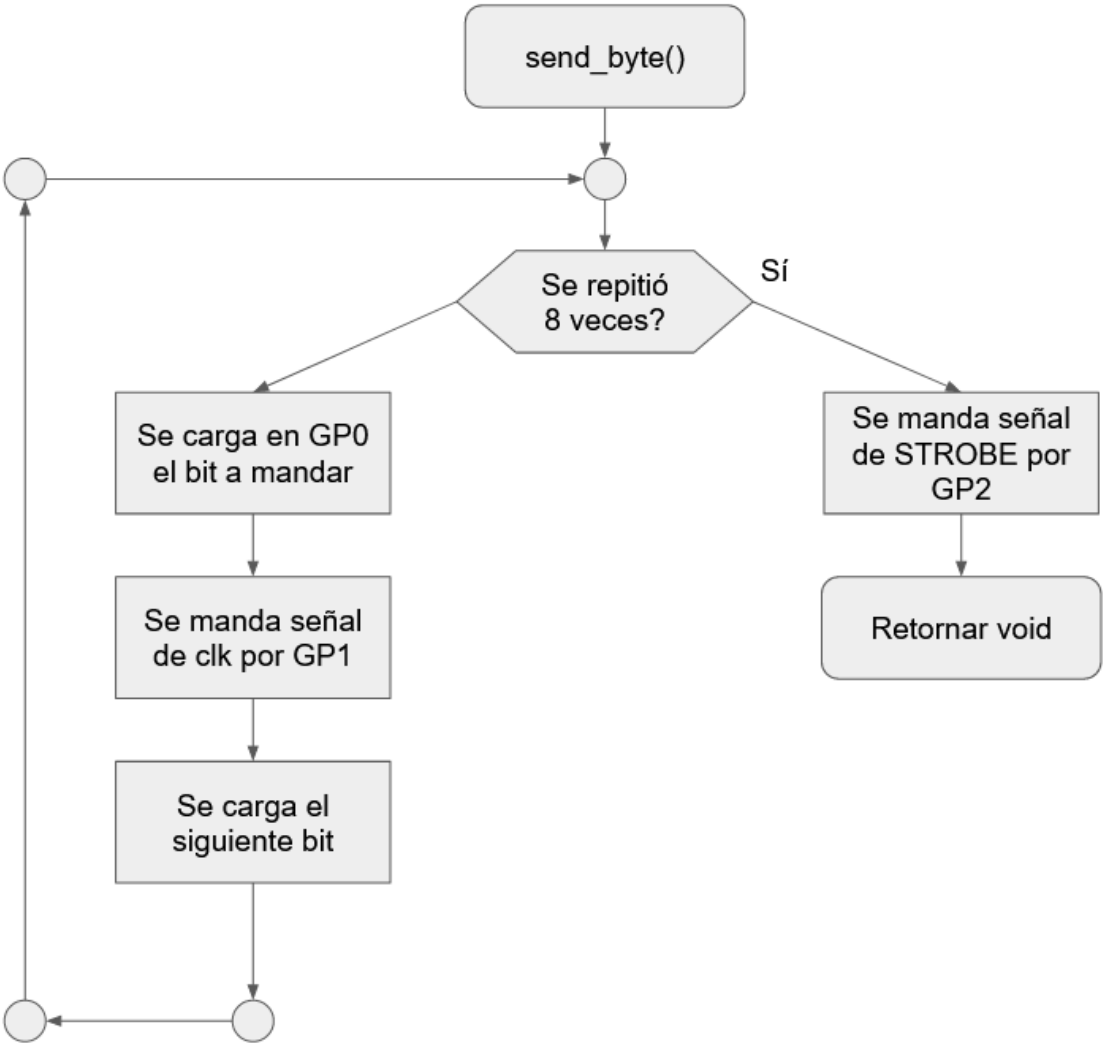
uint rand_wait() cumple dos funciones: esperar a que el usuario presione el botón y

devolver un número aleatorio en BCD, no repetido, cuando el botón haya sido presionado. Se utiliza 2 bits de bandera para poder generar una máquina de estado que le permite a la función dicernir entre los siguientes estados: botón no presionado, botón presionado y generar número aleatorio, y botón presionado y número ya fue generado. Esto hace que la función solo devuelva el número aleatorio una única vez siempre que se presione el botón aunque este se mantenga presionado por el usuario. Siempre que se encuentre en el estado *botón no presionado*, la función incrementa la variable rand en BCD que representa el número pseudoaleatorio siendo generado por medio de contadores por software. Cuando el botón es presionado, se asegura que el valor aleatorio en ese momento sea válido (no repetido) por medio de la función valido(). En caso de no ser repetido, devuelve el valor a la función main().

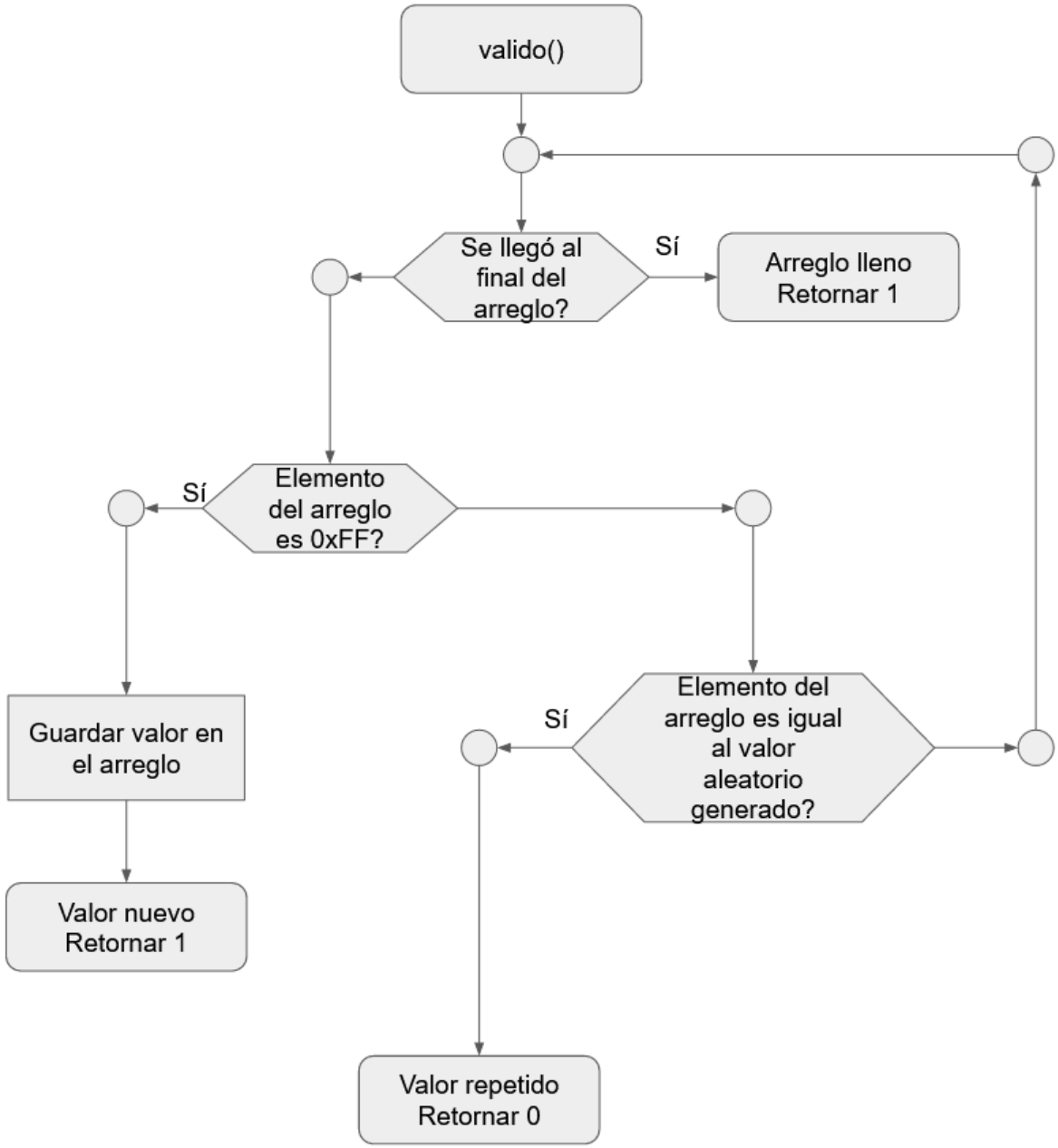
uint valido() revisa dos cosas: la primera es revisar si ya se generaron 16 números aleatorio, en cuyo caso prende una bandera para hacerle saber al main(). La función tiene acceso a un arreglo de 16 bytes que utiliza para guardar el historial de números generados. El segundo es, en caso de que no se hayan generado 16 valores aún, revisar si el valor generado es repetido o no. Si es repetido, devuelve un 0 y si no es repetido devuelve un 1.

La función main() se encarga de esperar que el usuario presione el botón y se genere un número aleatorio por medio de la función rand_wait(). Una vez que el botón haya sido presionado, en main() se revisa cual número fue generado y se le pasa a la función send_byte() la codificación para LEDS de 7 segmentos correspondiente para ser mandado a los registros desplazantes. Esto se realiza 2 veces; para el valor de las decenas y el valor de las unidades del número generado. Como el número generado viene en una variable en BCD de 1 byte, se utilizan las máscaras 0xF0 y 0x0F para accesar el valor de decenas y unidades del valor generado. Por último, este borra el arreglo de historial de números generados para generar una nueva secuencia de 16 números aleatorios.

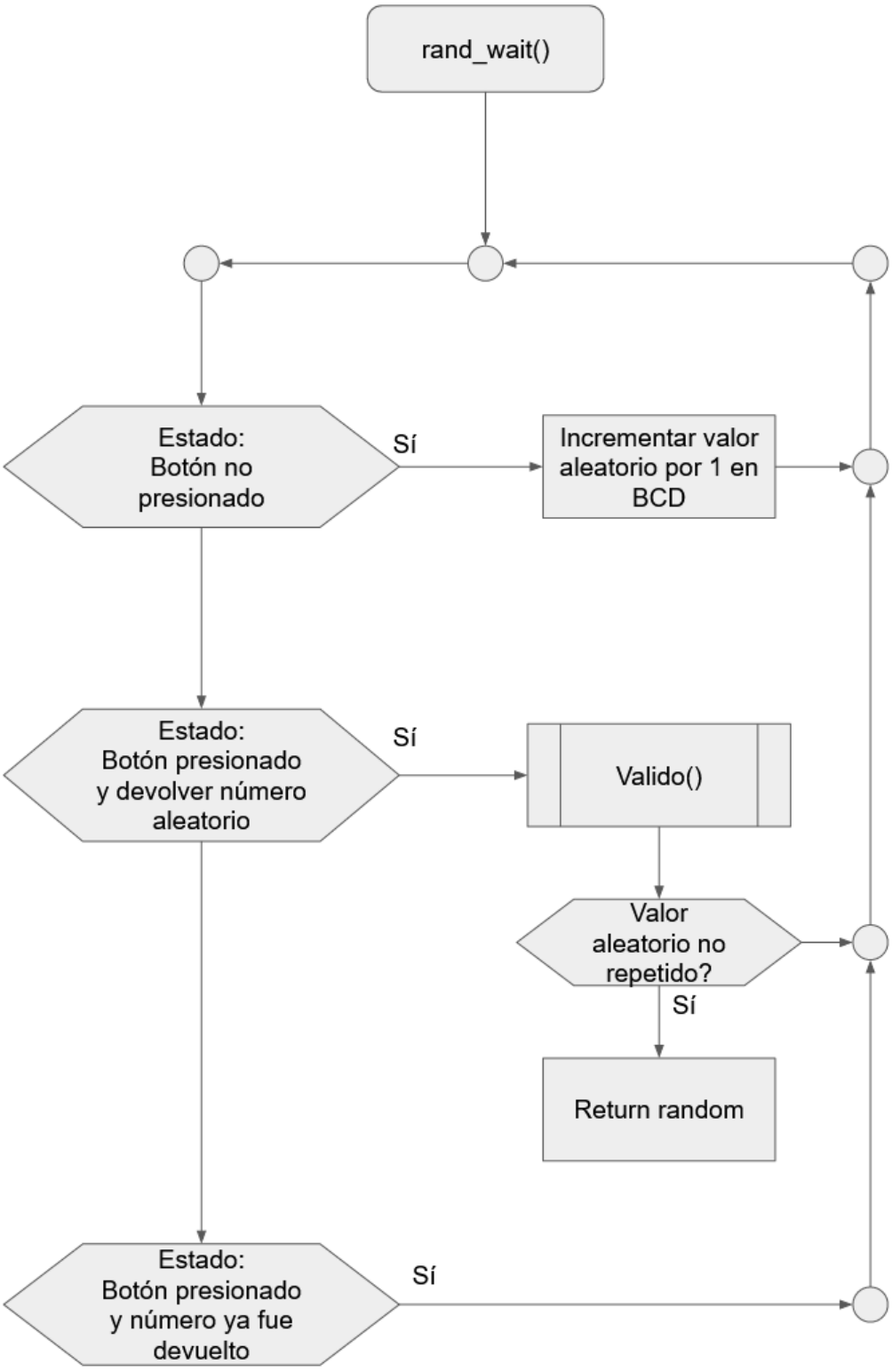
Diagrama de bloques



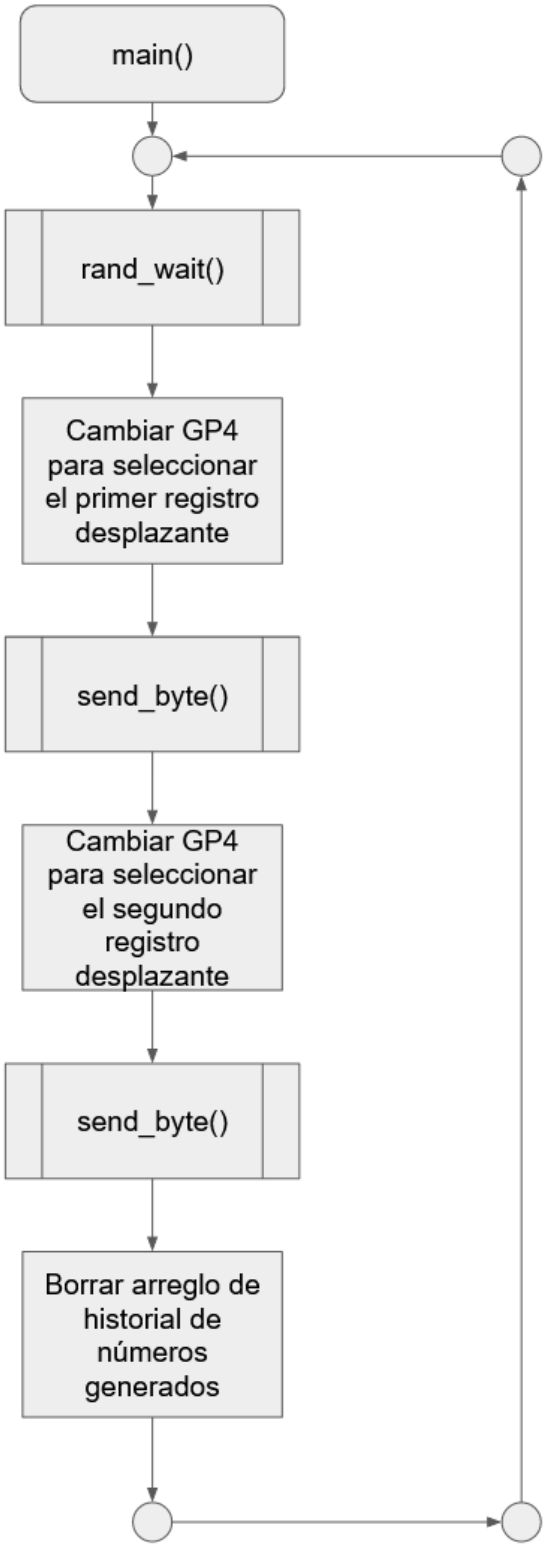
send_byte() 3.1: send_byte()



valido() 3.2: valido()



wait_rand() 3.3: wait_rand()

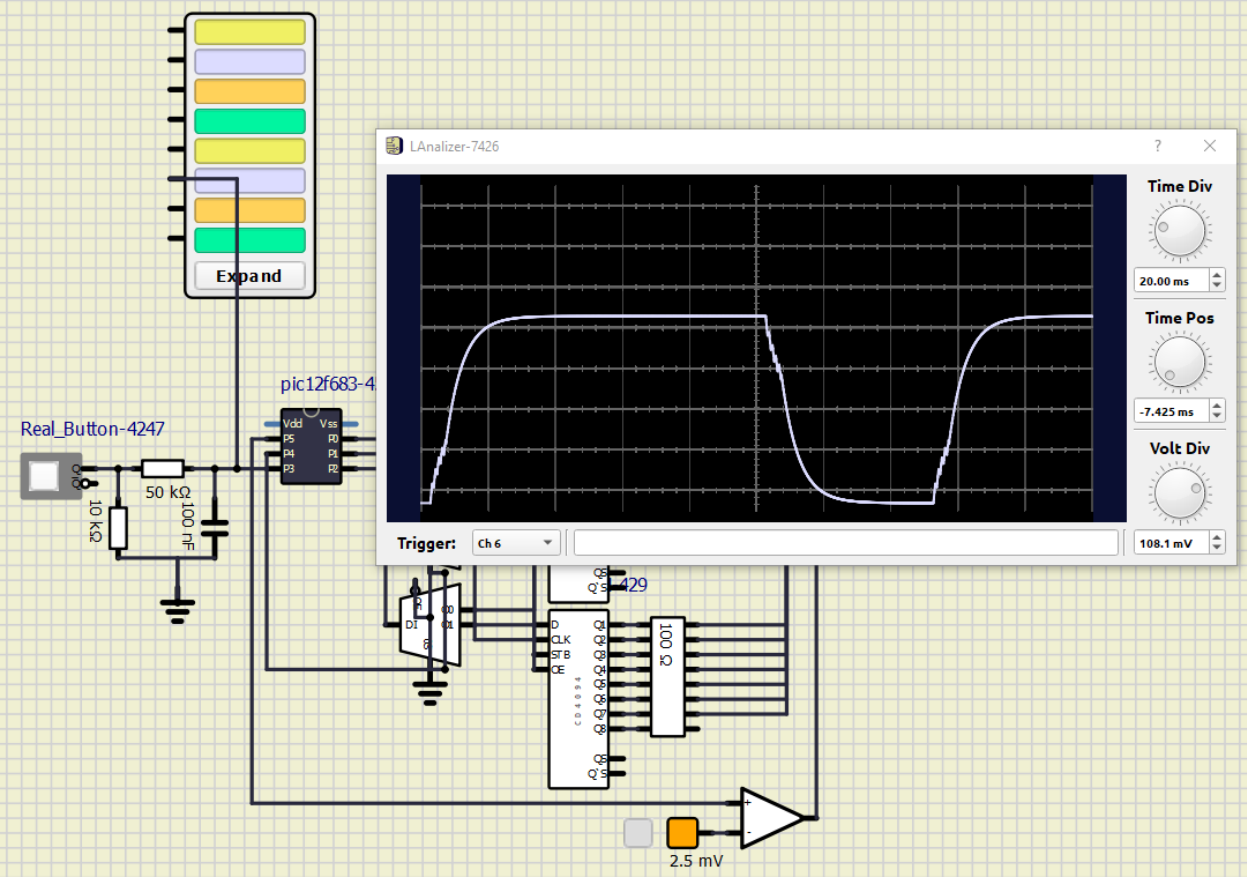

main()
rand_wait()
Cambiar GP4 para seleccionar el primer registro desplazante
send_byte()
Cambiar GP4 para seleccionar el segundo registro desplazante
send_byte()
Borrar arreglo de historial de números generados

main() 3.4: main()

3.2. Componentes

Supresor de rebotes

Según los valores escogidos de resistencia y capacitancia y según lo mencionado en la nota teórica del diseño del circuito, el supresor de rebotes debería de causar un tiempo de activación y desactivación de aproximadamente 24 ms.

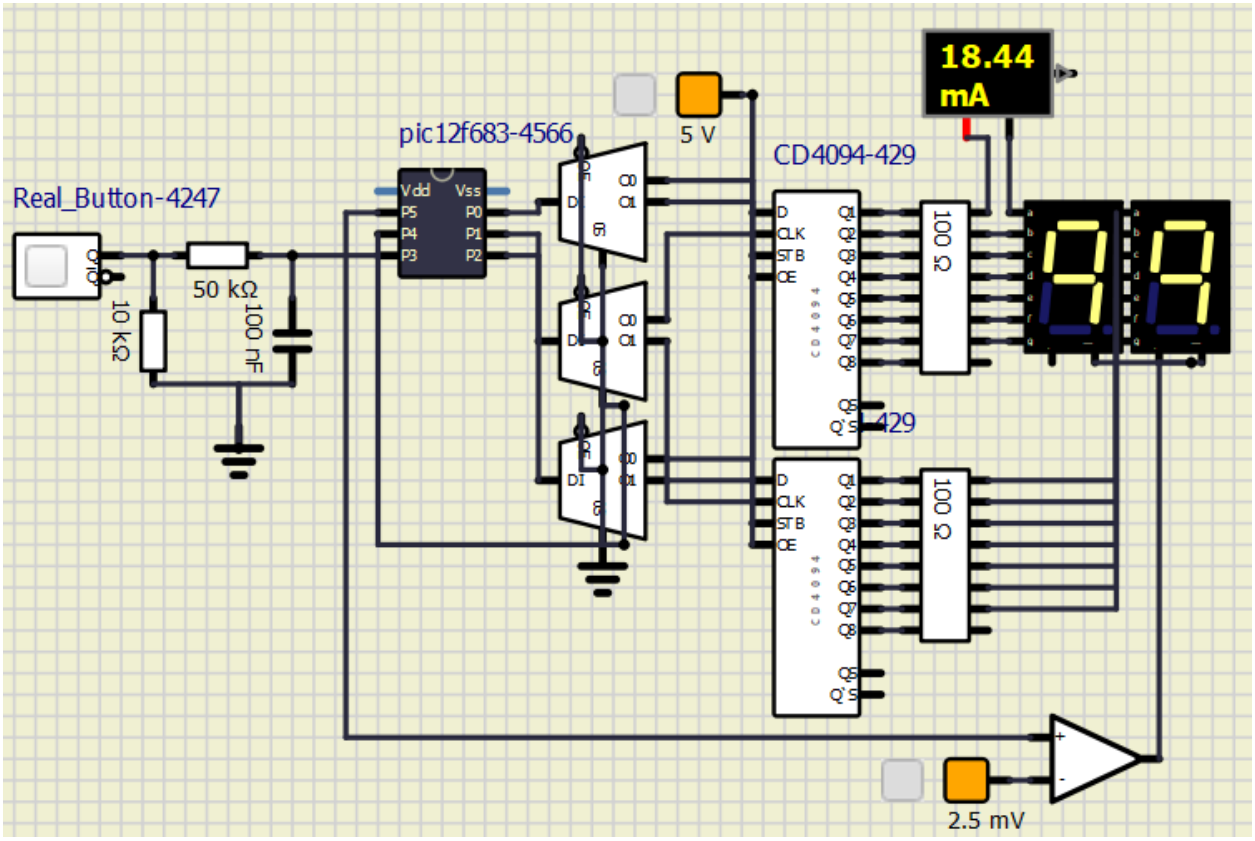


Respuesta supresor de botones 3.5: Obtenido del simulador

Como se observa de la imágen 3.5, en el analizador digital, a una resolución 20ms por división del gráfico, el tiempo de carga y descarga del capacitor es aproximadamente 20 ms. Este valor es el esperado.

LEDS 7 segmentos

Según los valores escogidos de resistencias de acuerdo con lo mencionado en la nota teórica del diseño del circuito, la corriente pasando por los LEDS debería ser marginalmente menor a 20 mA.

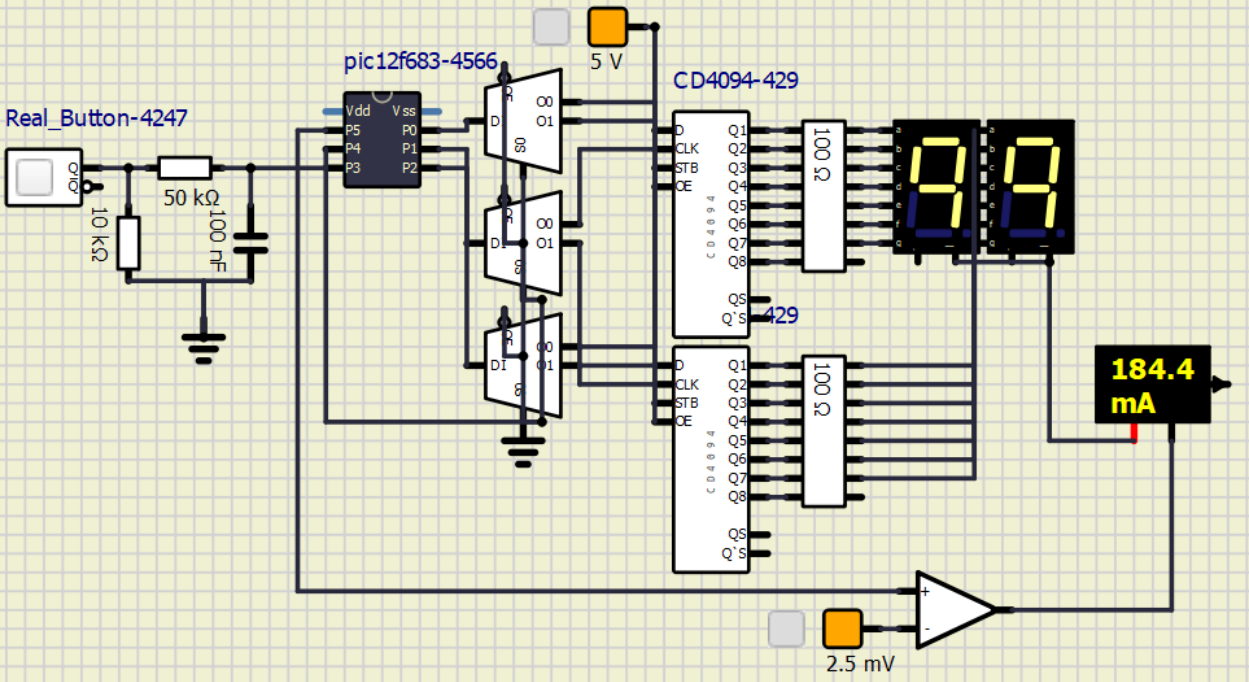


Corriente LEDS 3.6: Obtenido del simulador

Como se observa de la imágen 3.6, la corriente es de 18.44 mA, un valor esperado.

Corriente cátodo común

Como se mencionó en la nota teórica del diseño del circuito, para manejar el cátodo común de los LEDS, se ocupa un componente electrónico capaz de drenar 280 mA de corriente ya que por cada LED prendido, se va a tener que drenar a lo sumo 20 mA (18.44 en este diseño).



Corriente cátodo común 3.7: Obtenido del simulador

Como se observa en la imágen 3.7, el número 99 prende 10 LEDS, lo que debería de generar una corriente en el cátodo de 10*18.44mA=184.4mA. Este es exactamente el valor que se obtiene y es el esperado.

4. Conclusiones

Se concluyé que la solución propuesta para este primer laboratorio cumple con los requerimientos del enunciado y se comporta según el diseño teórico realizado.

4.1. Recomendaciones

Un cambio que se puede realizar para otra posible solución es manejando los LEDS de 7 segmentos por división de tiempo. En vez de usar un registro desplazante por cada conjunto de segmentos LED, se puede usar uno solo. Para lograr esto, se debe de activar cada LED por un cierto intervalo de tiempo, a una frecuencia de 60 Hz o mayor para que el ojo humano

lo perciba como si estuviese prendido todo el tiempo.

Así como esta solución simplificaría el hardware, este complicaría el software también. Con dos LEDS de 7 segmentos la diferencia no es significativa, pero con forme se agreguen más LEDS al diseño, se vuelve poco práctico utilizar un registro desplazante por cada LED.

Otra posible recomendación es realizar este proyecto en ensamblador en vez de C. Como se está utilizando un PIC12F683, con muy poca memoria ram, sería muy provechoso tener un control más preciso del uso de memoria. Este empeño no sería muy complicado ya que la arquitectura del PIC soporta instrucciones RISC, y por ende no habría que aprender muchos comandos para poder programarlo en ensamblador.

Bibliografía

[1] "Pic12f683 data sheet." https://ww1.microchip.com/downloads/en/DeviceDoc/41211D_.pdf, 2007. (Accessed on 25/03/2023).

[2] "Cd4094 data sheet." https://www.ti.com/lit/ds/symlink/cd4094b.pdf?HQS=dis-dk-null-digikeymode-dsf-pf-null-wwe&ts=1679982420960&ref_url=https://www.digikey.com/en/products/filter/logic/shift-registers/712?s=N4IgTCBcDaIMYBMAsAGAnEkBdAvkA, 2003. (Accessed on 25/03/2023).

[3] "Componentes electrónicos." https://www.digikey.com/en/products/, 2003. (Accessed on 25/03/2023).

5. GIT

https://github.com/JAR1224/Laboratorio_1_Jose_Ramos