

Laboratorio de Microcontroladores

Curso UCR

Profesor: Marco Villalta Fallas

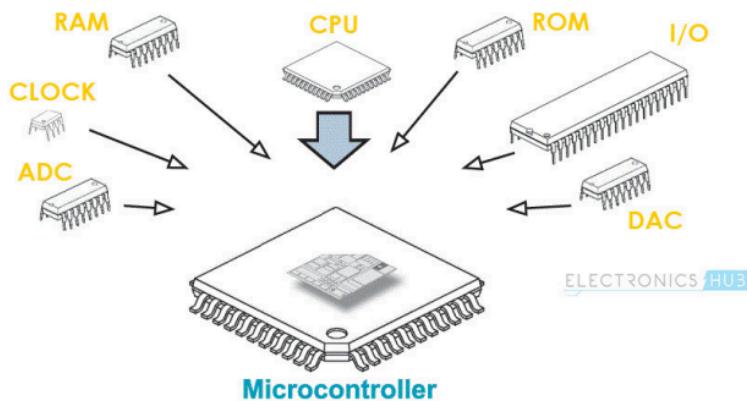
Dualok Fonseca Monge

1. Introducción:

¿Qué es un microcontrolador?

Dispositivo computacional en un solo chip tipo (VLSI-C), conocidos también como Computer on a Chip, también se les llama controladores embebidos porque microcontroladores y circuitos de soporte están integrados.

Tienen memoria RAM, ROM, circuitos de interrupción, puertos generales de entrada y salida (GPIO), comparadores analógicos, contadores, convertidores Analógicos/Digitales, watchdog timers, interfaces de comunicación. Disponibles en diferentes arquitecturas, tamaños de palabra y capacidades.



Estructura básica

- **CPU:** Central processing unit, consta de al menos una ALU y una CU, lee, decodifica y ejecuta instrucciones para realizar operaciones lógicas, aritméticas y de transferencia de datos.
- **Memoria:** -
 - *de programa* (Contiene instrucciones).
 - *de datos*: guarda la info.
- **I/O ports:** Puertos de E/S, interfaz con el mundo exterior.
- **Buses:** Grupo de conexiones agrupadas entre el CPU y otros periféricos.
- **Contadores/Temporizadores:** Proveen operaciones para contar eventos y tiempo, generaciones de funciones PWM, relojes de control.
- **Puertos seriales:** Interfaces para comunicación con el mundo exterior.
- **Interrupciones:** Proveen mecanismos para atender cambios externos, internos, de hardware y software.
- **Convertidores AC/DC**

Clasificación:

- **Por bits:** Existen microcontroladores de **8 bits** (utilizado para labores básicas); **16 bits**(tiene mayor precisión y desempeño); y **32 bits**(utilizado para aplicaciones avanzadas de control, como aplicaciones médicas, de telecomunicaciones, automatización, etc).
- **Por memoria:** Existen microcontroladores de memoria externa (no tan comunes) y de memoria interna
- **Por conjunto de instrucciones:** RISC (reduced instruction set computer) y CISC (complex instruction set computer)

Diferencias entre microprocesador y microcontrolador

Microcontrolador	Microprocesador
Más lento	Más rápido
Elemento completo y funcional	Requiere de otros periféricos
Diseñado para cumplir tareas puntuales	Tareas de gran capacidad computacional

ARM: (Advanced RISC machines), es más una arquitectura que un microcontrolador o microprocesador. Si tiene RAM y ROM externa se puede considerar un microprocesador, si tiene RAM y ROM interna; un microcontrolador.

Usos de los microcontroladores:

Sensado de luz, temperatura, humedad, velocidad, etc

Dispositivos para controlar procesos, Instrumentación industrial, Robótica, Industria automotriz. IoT.

Tarjetas de desarrollo:

Creadas para facilitar el prototipado, algunos incluyen periféricos y circuitos de alimentación.

Como evitar daños en el MCU:

- Cortos: Evitar contactos antes de energizar
- Sobre corriente: Revisar que el consumo de corriente no supere las especificaciones, se puede usar protectores o drivers de corriente
- Sobre voltaje: Revisar que el adaptador entregue tensión especificada por la placa
- Voltaje inverso: Revisar la polaridad de adaptador y conexiones
- Componentes externos: Revisar funcionamiento y conexión
- Malas prácticas: Tener lugar de trabajo limpio, fuentes de alimentación estables

Microcontroladores en IE-0624:

Microcontroladores: PIC, AVR, Arduino, STM32, ESP32.

Lenguajes: C, C++, Python.

Plataformas: SimulIDE(Simulador), Wokwi(Simulador), STM32F429 Discovery kit(real), Arduino Nano BLE33(real).

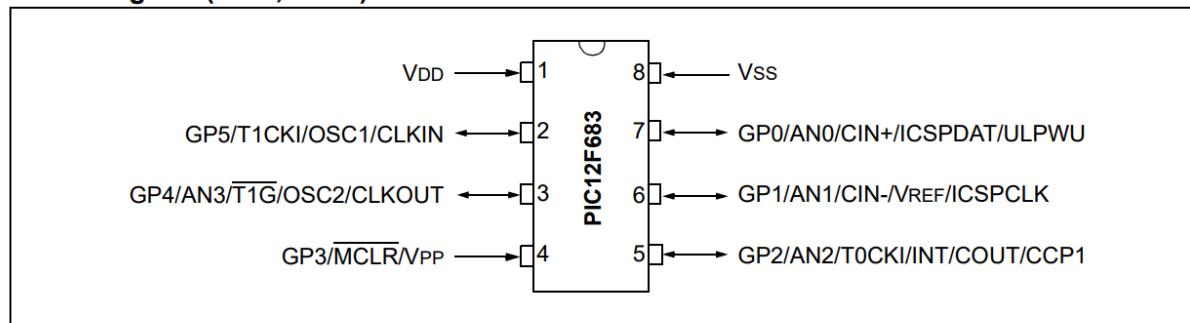
2. PIC12F683 y GPIOs:

GPIO: General purpose input/output

La gran mayoría de los pines en un MCU son GPIO, los pines que no son GPIO son VDD, VS, CLK, algunos pines de GPIO comparten otras funciones, se agrupan en puertos, en el caso del PIC12F683 utiliza un único puerto.

PIC12F683

8-Pin Diagram (PDIP, SOIC)



Cómo funcionan los GPIO:

Generalmente se configura el funcionamiento del GPIO a través de un registro

Si se configura como entrada se realiza la lectura con polling o por medio de interrupciones, si se configura como salida, se maneja el estado en el programa: 1 Alto, 0 bajo.

Registros importantes:

En este microcontrolador hay 6 GPIO y varios registros para dictar cómo funcionan estos pines.

- GPIO: Es un puerto de 6 bits bidireccional, este registro 1 es alto, 0 es bajo.

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
-n = Value at POR

W = Writable bit
'1' = Bit is set

U = Unimplemented bit, read as '0'
'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 > V_{IH}
 0 = Port pin is < V_{IL}

- TRISIO: Configura el flujo de datos de los GPIO, el tercer bit de GPIO siempre se lee como un input, por lo que el TRISIO(3) siempre lee un 1. 1 es un input y 0 es un output

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
-n = Value at POR

W = Writable bit
'1' = Bit is set

U = Unimplemented bit, read as '0'
'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

- CMCON: Registro donde se configura el comparador analógico, se debe modificar si se quiere configurar los pines GPIO0, GPIO1, GPIO2 como entradas.
 - NOTA: ANSEL y CMCON0 deben ser configurados si se quiere tratar un canal analógico como una entrada digital. Los pines configurados como analogicos leen "0"

- ANSEL: Registro donde se configura la conversión y selección analógica, se debe modificar si se quiere utilizar cualquier pin como entrada

REGISTER 4-3: ANSEL: ANALOG SELECT REGISTER

U-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-1	R/W-1	R/W-1
—	ADCS2	ADCS1	ADCS0	ANS3	ANS2	ANS1	ANS0
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 **Unimplemented:** Read as '0'

bit 6-4 **ADCS<2:0>:** A/D Conversion Clock Select bits

000 = Fosc/2

001 = Fosc/8

010 = Fosc/32

x11 = FRC (clock derived from a dedicated internal oscillator = 500 kHz max)

100 = Fosc/4

101 = Fosc/16

110 = Fosc/64

bit 3-0 **ANS<3:0>:** Analog Select bits

Analog select between analog or digital function on pins AN<3:0>, respectively.

1 = Analog input. Pin is assigned as analog input⁽¹⁾.

0 = Digital I/O. Pin is assigned to port or special function.

Note 1: Setting a pin to an analog input automatically disables the digital input circuitry, weak pull-ups and interrupt-on-change, if available. The corresponding TRIS bit must be set to Input mode in order to allow external control of the voltage on the pin.

- CONFIG

Especificaciones 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, I _{IK} (V _I < 0 or V _I > V _{DD}).....	± 20 mA
Output clamp current, I _{OK} (V _O < 0 or V _O > V _{DD}).....	± 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} x {I_{DD} - \sum I_{OH}} + \sum {(V_{DD} - V_{OH}) x I_{OH}} + \sum (V_{OL} x I_{OL}).

[†] NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operation listings of this specification is not implied. Exposure above maximum rating conditions for extended periods may affect device reliability.

Hola PIC código:

Hola PIC

```
#include <pic14/pic12f683.h>
typedef unsigned int word;
word __at 0x2007 _CONFIG = (_BODEN_OFF);
void delay (unsigned int tiempo);
void main(void)
{
    TRISIO = 0x00;//Poner todos los pines como salidas
    GPIO = 0x00;//Poner todos en bajo
    unsigned int time = 100;

    //Loop forever
    while ( 1 ){
        GPIO0 = 0x00;//Esto se puede hacer tamb con GP0=0x00(no recomendado) o enmascarando
        delay(time);
        GPIO0 = ~GPIO0;//Esto es con el metodo de campo de bit
        delay(time);
    }
}

void delay(unsigned int tiempo)
{
    unsigned int i;
    unsigned int j;

    for(i=0;i<tiempo;i++)
        for(j=0;j<1275;j++);
}
```

Configuración de los simuladores y ambiente de trabajo:

Repositorio de Git: <https://github.com/Dualock/Microcontroladores-Labs>

Se va a trabajar con el simulador SimulIDE, al inicio no podía compilar el ejecutable, hubo que instalar libfuse con el comando:

```
sudo apt install libfuse2
```

También se instaló sdcc con los comandos:

```
cd ~
```

```
mkdir tmp
```

```
cd tmp,
```

extrayendo el archivo sdcc. Luego ejecutando

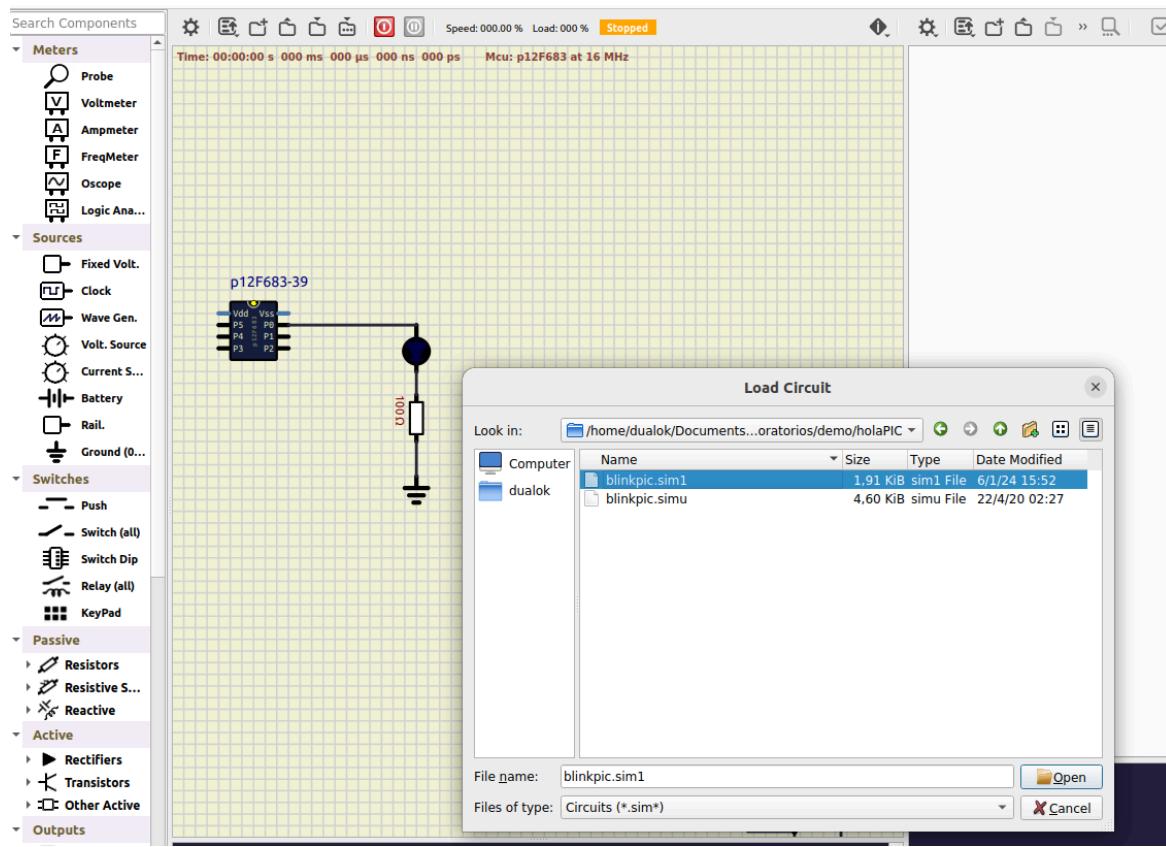
```
cd sdcc-4.0.0
```

```
cp -r * /usr/local
```

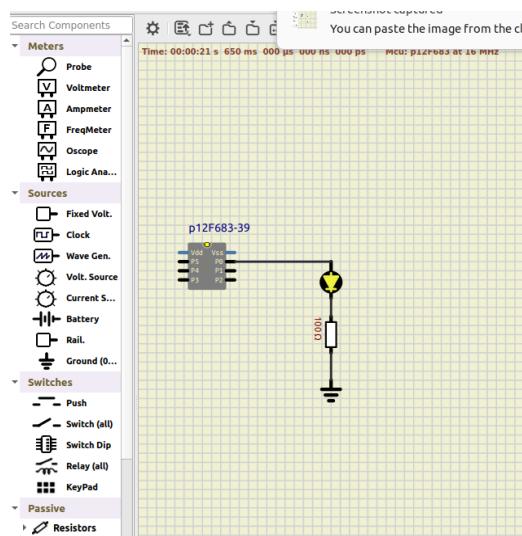
Para verificar que se instalaron correctamente, se ejecuta:

```
sdcc -v
```

Cómo utilizar el simulador: Se ejecuta el archivo ejecutable de simulIDE, en la opción open circuit, se abre el archivo.simu correspondiente, en este caso se abrió el blink.simu



Luego se selecciona el microcontrolador y se hace click en la opción load firmware y se carga el archivo .hex. Antes de esto es necesario haber programado el firmware en C y utilizar el makefile para generar el archivo.hex. Una vez cargado el firmware se comienza la simulación.



Evaluaciones para los laboratorios:

Evaluacion propuesta

- Introducción 5
 - Resumen
 - Conclusiones importantes
- Nota teórica 20
 - Información general MCU
 - Periféricos
 - Registros
 - Diseño de circuito
 - Lista de componentes y precios
 - Conceptos/temas del laboratorio
- Desarrollo/análisis 50
 - Análisis programa
 - Análisis electrónico
- Conclusiones y recomendaciones 10
- Bibliografia 5
- Git/avances 10

2.1 Laboratorio 1 - GPIOs:



Figura 2: Pantalla de 2 dígitos

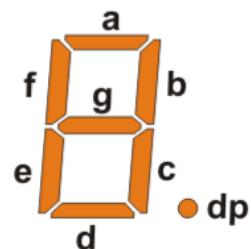
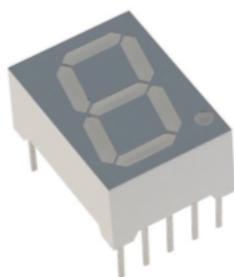
Desarrollara un simulador de tombola simplificada de bingo como el de la figura 1 utilizando dos displays de 7 segmentos para formar la pantalla de la figura 2, un botón, el microcontrolador PIC12F675/PIC12F683 y cualquier otro componente que considere necesario. El objetivo de este laboratorio es introducir al estudiante al manejo de GPIOs (*General Purpose Input Outputs*), generación de números aleatorios y el flujo de desarrollo que se pide para las prácticas dirigidas. Cada vez que se presiona el botón debe desplegarse en el display un número que represente a una de las bolas de la tombola que van del 00 al 99, se debe mantener un registro de los números que salen para no repetir. Para efectos de simplificar la simulación considere que la tombola solo tendrá 10 bolas, después de sacar 10 bolas/numeros deberá parpadear tres veces la pantalla con el número 99 y reiniciar el juego.

Solución:

Para realizar este laboratorio, se propone dividir el problema en varios pasos, los cuales incluyen: investigación, diseño del circuito y del software.

LED de 7 segmentos

Se trata de varios LEDS en la misma carcasa de plástico, donde de acuerdo a la configuración, se pueden escribir números (0 al 9) y letras. Para este laboratorio nos interesa con qué configuración de 8 bits se genera cada número



Digits to display		Display Segments						
	dp	a	b	c	d	e	f	g
0	0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0	0
2	0	1	1	0	1	1	0	1
3	0	1	1	1	1	0	0	1
4	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1
6	0	1	0	1	1	1	1	1
7	0	1	1	1	0	0	0	0
8	0	1	1	1	1	1	1	1
9	0	1	1	1	1	0	1	1

Cantidad de GPIO disponibles como salidas

Según el registro GPIO, se tienen 6 puertos disponibles, pero solo 5 son salidas, ya que el puerto GP3 solo se puede configurar como entrada. Es decir, hay que idear una forma de encender ambos LEDs de 7 segmentos con esas 5 salidas, dígase GP0, GP1, GP2, GP4 y GP5, esto se discute en la siguiente sección.

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 > V_{IH}

0 = Port pin is < V_{IL}

Técnicas a utilizar

Convertidor BCD a 7 Segmentos:

Se va a utilizar un convertidor de BCD a 7 Segmentos, esto permite mapear valores en BCD (4 bits) a sus correspondientes configuraciones de LED de 7 segmentos. De esta forma se tendrán 4 salidas para encender los LEDs, 1 entrada para el botón de push y sobra un puerto por si lo llegara a necesitar. Los números decimales codificados en binario son:

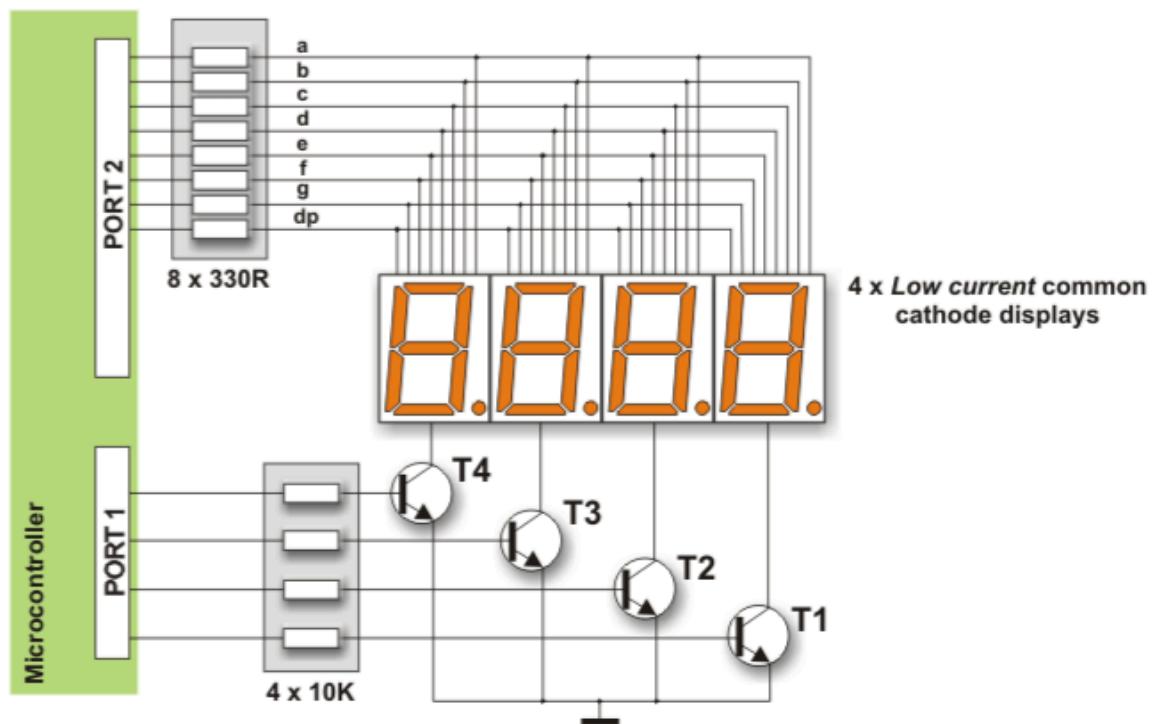
0000: 1 0011: 3 0110: 6 1001: 9

0001: 1 0100: 4 0111: 7

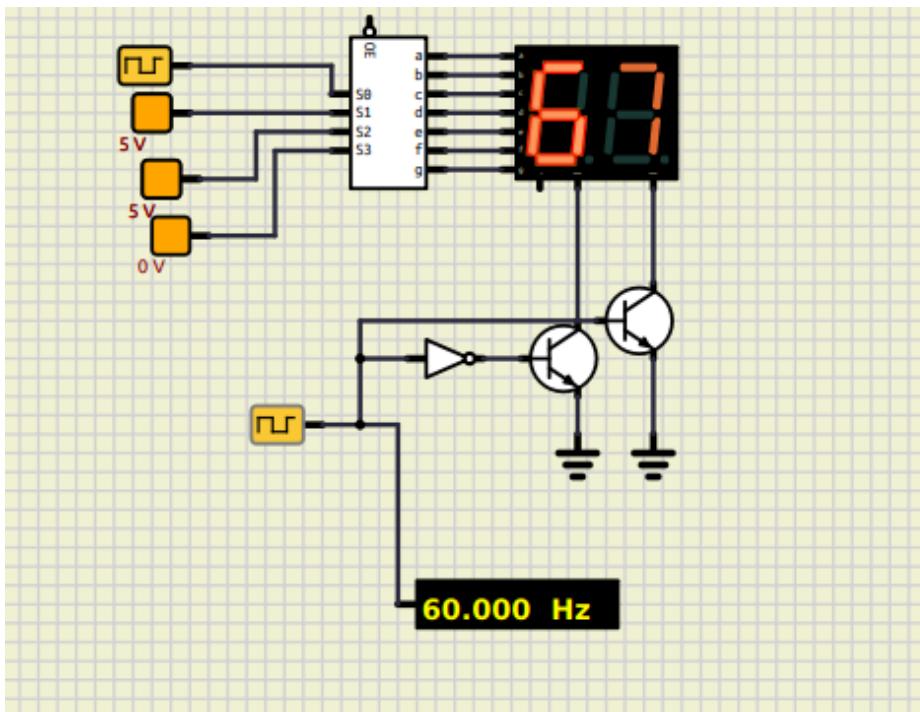
0010: 2 0101: 5 1000: 8

Multiplexación:

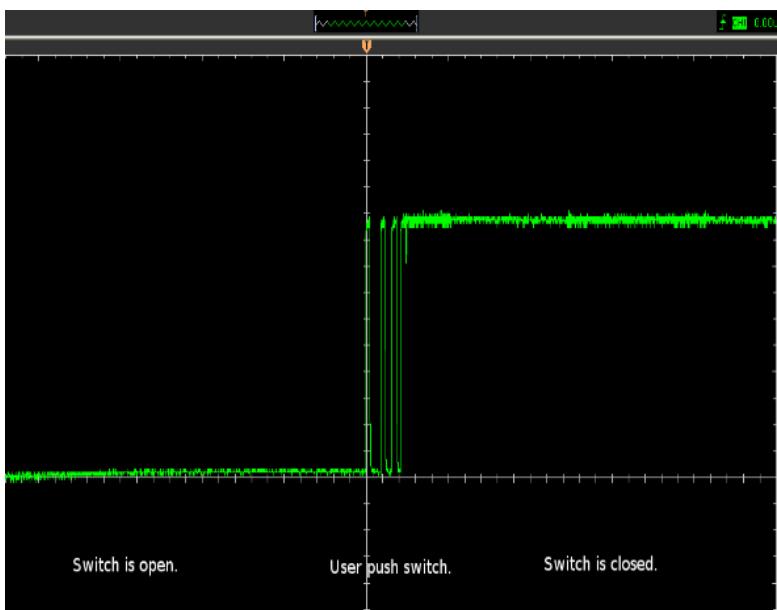
Se utilizará la técnica de multiplexación para dar la ilusión de que ambos displays de LED están encendidos al mismo tiempo, pero están conmutando con una frecuencia de 60hz haciendo uso de dos transistores BJT.



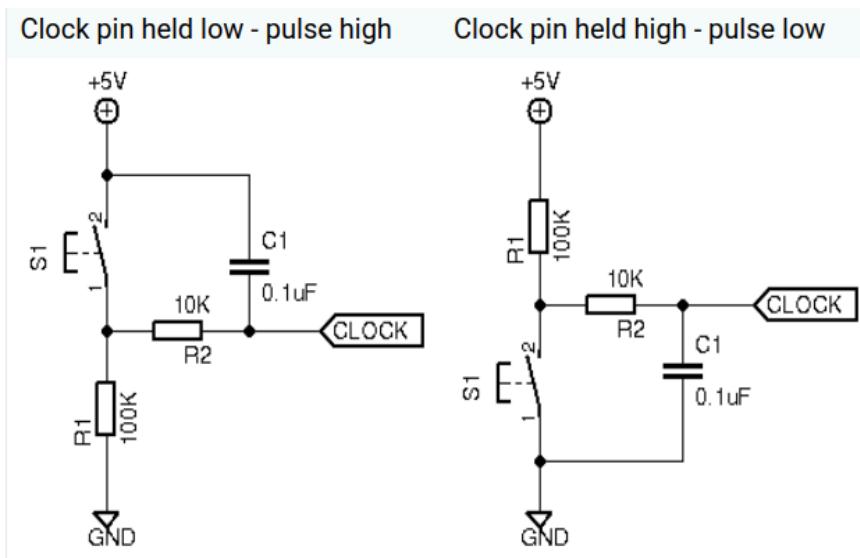
Como forma de probar este funcionamiento, se creó una simulación con un valor cambiante con la misma frecuencia de conmutación como se muestra en la figura



Debouncing: Al activar un botón en un microcontrolador, ocurre un fenómeno que no es perceptible para el ojo humano, pero que suele confundir al microcontrolador. Se trata de un rebote mecánico al activar el botón, como se muestra en la siguiente figura



Hay 2 formas de abordar este problema: La primera es utilizando el hardware, mediante un circuito RC para evitar los cambios bruscos de frecuencia y la otra es agregando tiempo dentro del software, el capacitor funciona como un circuito abierto a bajas frecuencias. En la figura, además se agregan las resistencias de pull down y pull up correspondientes.



La segunda es asignar un valor de rebote al software tal que llegue a un máximo la primera vez que es activado el botón y si el valor rebote no es cero, que baje en cada ciclo y hasta que llegue a 0 nuevamente puede ser leída la entrada como una pulsación.

Generación de números aleatorios

Para la generación de números aleatorios se hace uso de un linear feedback shift register (registro de desplazamiento con realimentación lineal), se emplea una función que recibe un número, inicialmente una semilla, de 8 bits y realimenta los bits 7, 5, 4 en una compuerta XOR en el bit menos significativo y se desplaza hacia la derecha

El polinomio de realimentación es:

$$x^7 + x^5 + x^4 + x^0$$

Comprobación de números repetidos

Se utiliza una función llamada check_num que revisa si el numero que salio en la góndola del bingo salio repetido, si salio repetido, entonces no lo guarda en el array de números y retorna el mismo índice para volver a probar con otro número, en caso de no estar repetido, lo guarda en el array de números y retorna el siguiente índice.

Modificación de registros, bits y macros necesarios del PIC

Los registros utilizados y modificados en este laboratorio fueron.

- TRISIO = 0x00 para asignar el flujo de datos y setear todos los GPIO como salidas, aunque el bit TRISIO[3] siempre lee 1, es decir siempre es una entrada.
- GPIO: Este fue el registro mas utilizado, ya que para obtener las combinaciones necesarias de salidas para representar los números en los LED de 7 segmentos, fue necesario estar cambiando el estado (alto o bajo) de este registro.
- Se modifica el registro CONFIG ubicado en la dirección 0x207, mediante la instrucción:

```
typedef unsigned int word;
word __at 0x207 __CONFIG = (_WDTE_OFF & _WDT_OFF & _MCLRE_OFF);
```

De este modo se desactiva el temporizador de vigilancia o WatchdogTimer (WDT) y el master clear (MCLR)

Lectura del estado del botón:

Se le llama PUSH_B a la entrada de lectura GP3. Para leer el botón, se utiliza el puerto GP3, como se utiliza una red de pull up, el pin siempre está en alto, hasta que se presiona el botón se registra la entrada como un cero lógico, por esta razón para la lectura se utiliza !PUSH_B. Como recomendación de la lectura sobre

debouncing, se utilizó también un valor de rebote en el código, de modo que cada vez que se presiona el botón, el valor de rebote llega a su máximo y el programa no registra más entradas hasta que ese valor de rebote llegue a 0. Cada vez que se registra una entrada se llaman las funciones para generar el número, obtener las decenas y unidades y verificar si el número ya existe.

```
else if(!PUSH_B && (valor_rebote == 0)){
    valor_rebote = 30;
    numero = lfsr(numero);
    decenas = numero/10;
    unidades = numero%10;
    index = check_num(tiros, index, numero);
}
// logica para el debouncing
else if(valor_rebote > 0){
    valor_rebote--;
}
```

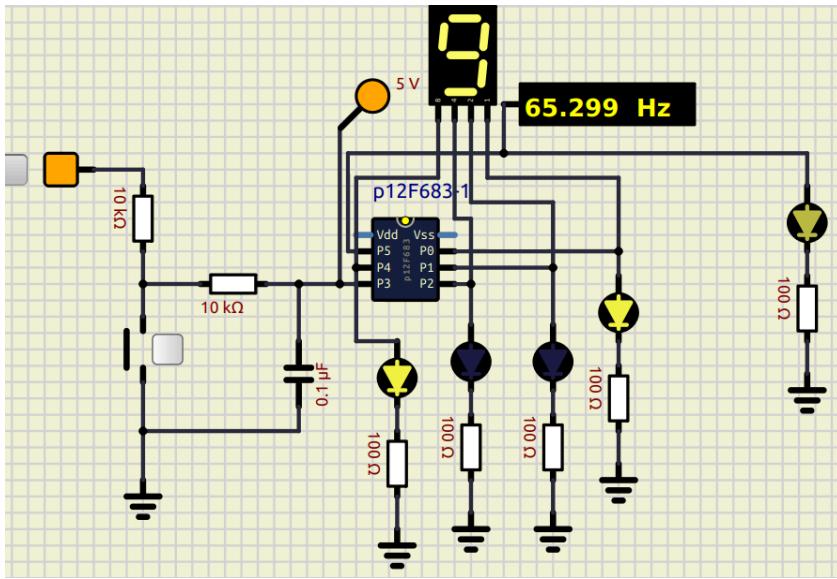
Diagrama de flujo del programa:

Circuito contador de pulsaciones:

Luego se creó otra sección del circuito para asegurar que el contador de pulsaciones del botón funciona y que cuenta hasta 10, guardando los datos en un arreglo, como se explico en la sección *Analisis de programa*,

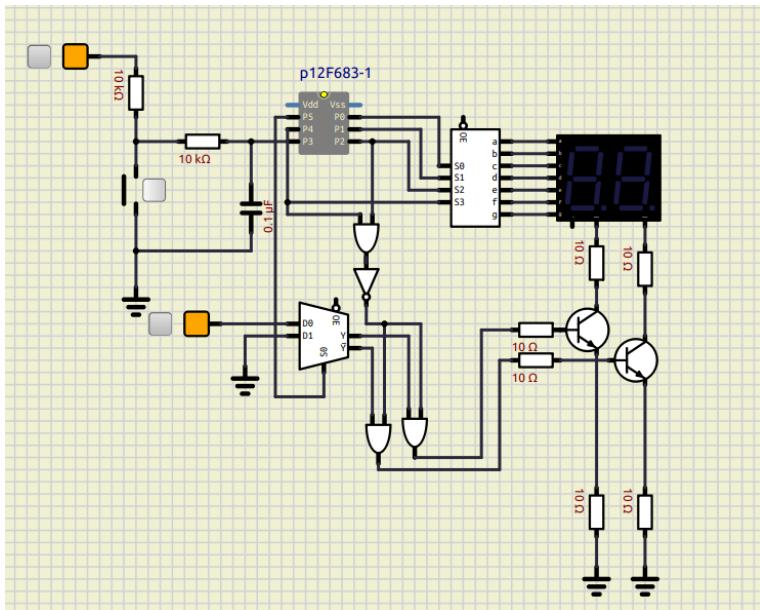
Se puede notar que para este punto, se agregó una red de pull up y un filtro RC para filtrar las altas frecuencias y así minimizar el efecto del rebote del botón

El contador se puede ver funcionando en la siguiente imagen.



Circuito completo:

Anteriormente se analizó cómo funciona la multiplexión de la salida GP5 y la red de pull up para la lectura del botón en la entrada GP3, en el circuito completo se puede observar además un Mux, que se utiliza para conmutar la salida y lógica combinacional agregada al circuito de conmutación. Nótese cómo las salidas GP4 y GP2 (correspondientes a los 2 bits más significativos del número en BCD) se conectan a una compuerta AND seguido de un inversor esto genera un 1 constante en la salida a menos que GP4 = GP2 = 1. Ya que ningún número en BCD en el alfabeto numérico decimal requiere ambos bits en alto, es decir (1 1 X X), se utilizan estos bits como método para bloquear ambos transistores. de este modo se pueden apagar ambos display de 7 segmentos al poner una combinación tipo (1 1 X X). En el código se elige hacerlo con un 15, equivalente a F en hexadecimal



3. ATtiny4: GPIO, interrupciones, timers, motores

ATtiny4313:

Es un microcontrolador de AVR de 8 bits, emplea arquitectura RISC/Harvard, tiene 4kb de memoria flash, 18 GPIOs agrupados en 3 puertos, temporizadores y contadores de 8 y 16 bits, 4 canales PWM y comunicación USI / USART.

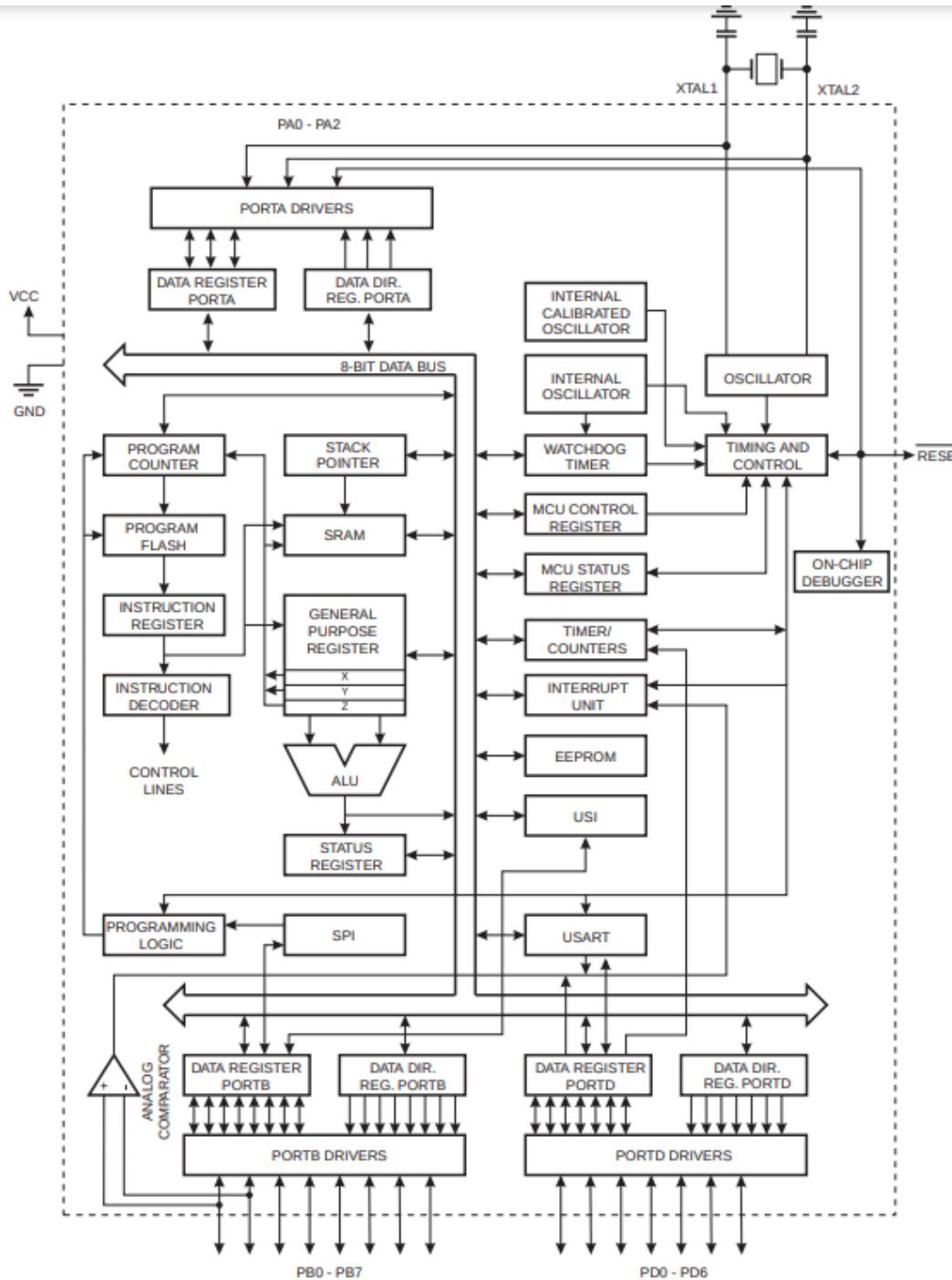
Diagrama de pines:

PDIP/SOIC

(PCINT10/RESET/dW) PA2	1	20	VCC
(PCINT11/RXD) PDO	2	19	PB7 (USCK/SCL/SCK/PCINT7)
(PCINT12/TXD) PD1	3	18	PB6 (MISO/DO/PCINT6)
(PCINT9/XTAL2) PA1	4	17	PB5 (MOSI/DI/SDA/PCINT5)
(PCINT8/CLKI/XTAL1) PA0	5	16	PB4 (OC1B/PCINT4)
(PCINT13/CKOUT/XCK/INT0) PD2	6	15	PB3 (OC1A/PCINT3)
(PCINT14/INT1) PD3	7	14	PB2 (OC0A/PCINT2)
(PCINT15/T0) PD4	8	13	PB1 (AIN1/PCINT1)
(PCINT16/OC0B/T1) PD5	9	12	PB0 (AIN0/PCINT0)
GND	10	11	PD6 (ICPI/PCINT17)

Bloques importantes

CPU, RAM, ROM/Flash, EEPROM, Temporizadores, Convertidores A/D, puertos de entrada/salida GPIO, Comunicaciones seriales.



GPIOs Lectura y escritura:

- DDxn: bit pertenece al registro DDRx: 1 para salida, 0 para entrada. Este es el registro de configuraciones
- PORTxn: Cuando está configurado como entrada, 1 activa la resistencia de pull up. Para apagar la resistencia, se debe configurar el pin como salida. Cuando está configurado como salida, 1 es alto, 0 es bajo. Este es el registro de datos
- PINxn: Escribiendo un 1 en este registro, cambia el valor de PORTxn independientemente del valor en DDRx, esto permite hacer un toggle, cambiar el estado del pin a través de este registro.
- Independientemente del valor de DDxn, el pin se puede leer en el bit PINxn
- Cuando se hacen lecturas por interrupciones, se puede configurar en el registro PCMSKn y GIMSK.

GPIOs registros:

Cada puerto/pin consiste de 3 bits de registro: DDxn, PORTxn, PINxn donde x es el puerto y n es el pin.

10.3.5 PORTB – Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x18 (0x38)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

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								
Initial Value	0	0	0	0	0	0	0	0	

10.3.7 PINB – Port B Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R/W								
Initial Value	N/A								

GPIOs registros de interrupción:

- GISMK

GIMSK – General Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	GIMSK
0x3B (0x5B)	INT1	INT0	PCIE0	PCIE2	PCIE1	-	-	-	
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	

Bit7 INT1: Bit de solicitud de interrupción externa #1 (interrupt request 1), ponerlo en 1 para activarlo.

Bit6 INT0: Bit de solicitud de interrupción externa #0 (interrupt request 0), ponerlo en 1 para activarlo.

Bit5 PCIE0: (Pin change interrupt enable 0), habilitación de interrupciones por cambios en pin #0, cualquier cambio en cualquiera de los primeros 8 bits del registro PCINT, es decir PCINT[7:0], activará la interrupción. Habilitados individualmente por el registro PCMSK0. Habilita las interrupciones del **puerto B**

Bit4 PCIE2: (Pin change interrupt enable 2), habilitación de interrupciones por cambios en pin #2, cualquier cambio en cualquiera de los pines PCINT17...11 del registro PCINT, es decir PCINT[17:11], activará la interrupción. Habilitados individualmente por el registro PCMSK2. Habilita las interrupciones del **puerto D**

Bit3 PCIE1: (Pin change interrupt enable 0), habilitación de interrupciones por cambios en pin #1, cualquier cambio en cualquiera de los pines PCINT10...8 del registro PCINT, es decir PCINT[10:8], activará la interrupción. Habilitados individualmente por el registro PCMSK1. Habilita interrupciones del **puerto A**

// Ejemplo de modificación de registro

```
GIMSK |= (1 << PCIE2) // Se habilita la interrupcion por PCIE2
```

- PCMSKn: Registro de mascarado de cambios de pin

9.3.4 PCMSK2 – Pin Change Mask Register 2

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	–	PCINT17	PCINT16	PCINT15	PCINT14	PCINT13	PCINT12	PCINT11	PCMSK2
Read/Write	R	R/W							
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 6..0 – PCINT17..11: Pin Change Enable Mask 17..11**

Each PCINT17..11 bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT17..11 is set and the PCIE1 bit in GIMSK is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT17..11 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

9.3.5 PCMSK1 – Pin Change Mask Register 1

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	–	–	–	–	–	PCINT10	PCINT9	PCINT8	PCMSK1
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:3 – Res: Reserved Bits**

These bits are reserved and will always read as zero.

- **Bits 2..0 – PCINT10..8: Pin Change Enable Mask 10..8**

Each PCINT10..8 bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT10..8 is set and the PCIE1 bit in GIMSK is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT10..8 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

9.3.6 PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
0x20 (0x40)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7..0 – PCINT7..0: Pin Change Enable Mask 7..0**

Each PCINT7..0 bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT7..0 is set and the PCIE0 bit in GIMSK is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT7..0 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

Interrupciones:

- *Polling*: Técnica de software donde se consulta el estado de un periférico, no es muy determinista
- *Interrupciones*: En lugar de hacer polling es más eficiente que el periférico indique su disposición a nuevos datos, una interrupción es una notificación al CPU que un evento ha sucedido, pueden ser disparados por varios eventos o periféricos como I/O, timers, ADC, etc. Deben ser atendidos en **rutinas cortas**.

Funcionamiento: Al dispararse una interrupción, la ejecución del programa se detiene, el estado del programa se guarda, el procesador revisa los vectores de interrupción (espacio de memoria de los registros de interrupción). Se ejecuta la ISR (Interrupt Service Routine). Luego continúa el programa.

- *Vectores de interrupción*

Vector No.	Program Address	Label	Interrupt Source
1	0x0000	RESET	External Pin, Power-on Reset, Brown-out Reset, and Watchdog Reset
2	0x0001	INT0	External Interrupt Request 0
3	0x0002	INT1	External Interrupt Request 1
4	0x0003	TIMER1 CAPT	Timer/Counter1 Capture Event
5	0x0004	TIMER1 COMPA	Timer/Counter1 Compare Match A
6	0x0005	TIMER1 OVF	Timer/Counter1 Overflow
7	0x0006	TIMER0 OVF	Timer/Counter0 Overflow
8	0x0007	USART0, RX	USART0, Rx Complete
9	0x0008	USART0, UDRE	USART0 Data Register Empty
10	0x0009	USART0, TX	USART0, Tx Complete
11	0x000A	ANALOG COMP	Analog Comparator
12	0x000B	PCINT0	Pin Change Interrupt Request 0
13	0x000C	TIMER1 COMPB	Timer/Counter1 Compare Match B
14	0x000D	TIMER0 COMPA	Timer/Counter0 Compare Match A
15	0x000E	TIMER0 COMPB	Timer/Counter0 Compare Match B
16	0x000F	USI START	USI Start Condition
17	0x0010	USI OVERFLOW	USI Overflow
18	0x0011	EE READY	EEPROM Ready
19	0x0012	WDT OVERFLOW	Watchdog Timer Overflow
20	0x0013	PCINT1	Pin Change Interrupt Request 1
21	0x0014	PCINT2	Pin Change Interrupt Request 2

Rutinas Cortas:

Si se quiere manejar una interrupción se puede usar ISR(), siempre se llama a la función sei() al inicio del programa cuando se esté configurando el microcontrolador.

```
ISR(PCINT2_vect) //SIGNAL tamb sirve pero para mi ISR tiene mas sentido
{
    //Instrucciones de la rutina de interrupcion
}
```

PCINT2_vect es una macro, que se puede consultar en el archivo de encabezado .h del microcontrolador.

Recordar que las interrupciones:

PCIE0: Se dispara si PCINT7...0 cambian su estado, es decir los pines del puerto B

PCIE1: se dispara si PCINT10...8 cambian su estado, es decir los pines del puerto A

PCIE2: se dispara si PCINT17...11 cambian su estado, es decir los pines del puerto D

Timers:

Los timers, counters permiten medir intervalos de tiempo o contar eventos internos y externos, pueden generar interrupciones, la frecuencia del contador está en función de la fuente reloj y la configuración de escala (prescaler).

Prescaler:

Circuito contador/divisor utilizado para reducir una señal de alta frecuencia y obtener una de menor frecuencia por medio de una división.

Típicamente se utiliza la entrada de reloj del sistema y se divide entre una potencia de 2 para alimentar un timer.

El attiny tiene un prescaler para el timer 0 que se configura con el registro TCCR0B (Timer counter control register B)

11.9.2 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	

Los últimos 3 bits se utilizan para selección de reloj, y se configuran como se ve en la siguiente tabla

Table 11-9. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$\text{clk}_{\text{I/O}}$ /(No prescaling)
0	1	0	$\text{clk}_{\text{I/O}}/8$ (From prescaler)
0	1	1	$\text{clk}_{\text{I/O}}/64$ (From prescaler)
1	0	0	$\text{clk}_{\text{I/O}}/256$ (From prescaler)
1	0	1	$\text{clk}_{\text{I/O}}/1024$ (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.

En el Attiny se puede seleccionar utilizar un reloj externo, un oscilador interno de 4MHz u 8 Mhz, por defecto está configurado como 8 MHz.

Timer 0 es el contador de 8 bits del Attiny, se utilizan los registros TCCR0A, TCCR0B, OCR0A, OCR0B, TIMSK y TIFR para operarlo.

TCCR0A:

Registro de control A para el contador 0, se configura la funcionalidad del pin I/O conectado a OCR0A o OCR0B. Se establecen modos de comparación (Normal, PWM, CTC)

PWM: Pulse width modulation - CTC: Collapse timer comparator

TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x30 (0x50)	COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Los últimos 2 bits WGM01 y WGM00 (Waveform generation mode), en conjunto con el bit WGM02 del registro TCCR0B controlan la secuencia del contador, el valor máximo del contador y el tipo de generación de onda a ser utilizado. Para este lab no se usa el modo PWM

Table 11-2. Compare Output Mode, non-PWM Mode

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Toggle OC0A on Compare Match
1	0	Clear OC0A on Compare Match
1	1	Set OC0A on Compare Match

Mode	WGM2	WGM1	WGM0	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on ⁽¹⁾⁽²⁾
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCR0A	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	TOP	MAX
4	1	0	0	Reserved	-	-	-
5	1	0	1	PWM, Phase Correct	OCR0A	TOP	BOTTOM
6	1	1	0	Reserved	-	-	-
7	1	1	1	Fast PWM	OCR0A	TOP	TOP

Notar que en modo Normal, el contador cuenta hasta 255 (FF) y en modo CTC (Clear timer on compare) cuenta hasta el valor que haya en OCR0A y luego se resetea.

OCR0A

Valor con el que se compara la cuenta del counter 0. También existe el registro OCR0B para tener otro valor de comparación.

11.9.4 OCR0A – Output Compare Register A

Bit	7	6	5	4	3	2	1	0	
0x36 (0x56)	OCR0A[7:0]								OCR0A
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	

The Output Compare Register A contains an 8-bit value that is continuously compared with the counter value (TCNT0). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC0A pin.

OCR0B

11.9.5 OCR0B – Output Compare Register B

Bit	7	6	5	4	3	2	1	0	
0x3C (0x5C)	OCR0B[7:0]								OCR0B
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	

The Output Compare Register B contains an 8-bit value that is continuously compared with the counter value (TCNT0). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC0B pin.

TIFR:

11.9.7 TIFR – Timer/Counter Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x38 (0x58)	TOV1	OCF1A	OCF1B	-	ICF1	OCF0B	TOV0	OCF0A	TIFR
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	

Registro de banderas por interrupción, del timer/counter0.

Bit2 OCF0B: Output Compare Flag 0 B. Se activa cuando hay una coincidencia entre el temporizador/contador y los datos en OCR0B. Se reinicia automáticamente durante la

ejecución del vector de interrupción correspondiente o manualmente escribiendo un uno lógico en la bandera.

Bit1 TOV0: Indica el fin de cuenta. El bit TOV0 se activa en un desbordamiento del Timer/Counter0. Se reinicia automáticamente en la interrupción o manualmente escribiendo un uno lógico. Cuando SREG I-bit, TOIE0 y TOV0 están activos, se ejecuta la interrupción por desbordamiento del Timer/Counter0.

Bit0 OCF0A: Igual que el OCF0B pero la comparación es con los datos en OCR0A.

Timer/Counter0: Es el contador de 8 bits del ATtiny4313, se utilizan los registros TCCR0A, OCR0A, TCCR0B y TIFR. En caso de utilizar interrupciones se debe modificar también el TIMSK.

11.9.6 TIMSK – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	TIMSK
0x39 (0x59)	TOIE1	OCIE1A	OCIE1B	-	ICIE1	OCIE0B	TOIE0	OCIE0A	
ReadWrite	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 2 – OCIE0B: Timer/Counter0 Output Compare Match B Interrupt Enable
- Bit 1 – TOIE0: Timer/Counter0 Overflow Interrupt Enable
- Bit 0 – OCIE0A: Timer/Counter0 Output Compare Match A Interrupt Enable

Se pueden tener interrupciones por comparación con OCR0A, OCR0B y overflow, habilitar interrupción global sei() y se debe atender la ISR asignada

```
ISR(TIMERO_COMPA_vect) //ISR es la macro que atiende interrupciones
{
    // Instrucciones para atender interrupcion
}
```

Watchdog timer:

Provee una forma controlada y dedicada para recuperarse de un problema de sistema, por ejemplo en caso de un problema de HW o un loop infinito, si es habilitado se debe resetear periódicamente. En caso de no resetearse, se dispara una interrupción o el MCU se reinicia

Motores:

La mayoría que se utilizan en microcontroladores son DC, la carga de un motor DC suele ser mayor a la que el MCU puede otorgar, por lo que es necesario utilizar elementos externos para el manejo de carga.

Controlar un motor consiste en controlar dirección y velocidad, si no se requiere control de velocidad o dirección, con un relé o MOSFET basta para encenderlos

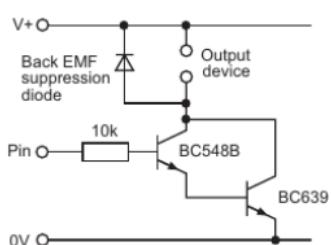
*Control on*off:* Muchas cargas requieren de un circuito comutador basado en transistor. La mayoría de casos usan configuración darlington

Un relé se utiliza para dispositivos de alto poder como motores y solenoides, puede ser alimentado por una fuente separada de (12V p.e).

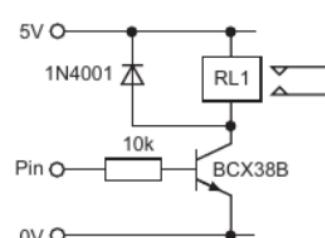
Se utilizan BJT para cargas de bajo consumo. Un MOSFET se utiliza en lugar de un darlington para cargas medidas (40V ~ 60V), resistencia entre gate y tierra como pull down.

El diodo flyback es un supresor del EMF (fuerza electromotriz) debido a cambios bruscos de ON/OFF (efecto inductivo que genera picos de voltaje), la EMF es proporcional a la velocidad del motor.

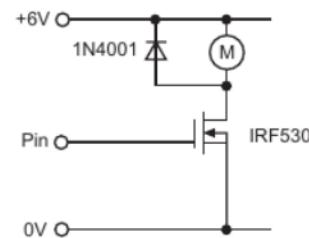
Darlington



Relé



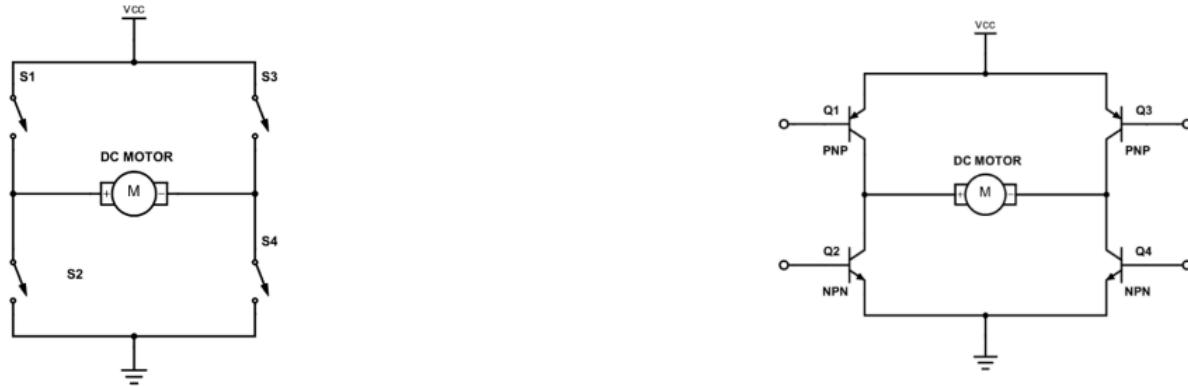
MOSFET



Esta configuración donde el motor está arriba se llama highside, existe lowside y se escoge de acuerdo a la aplicación.

Control de giro:

Se utiliza un puente H



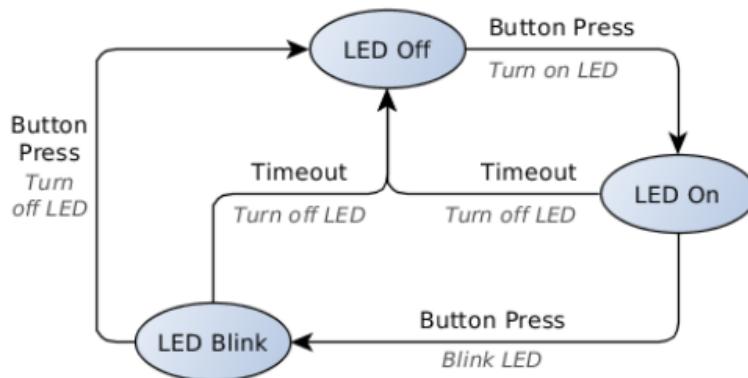
Si se cierran los interruptores 1, 4. La tensión positiva entra por la izquierda del motor y lo hace girar en un sentido; si se cierra el 2,3 la tensión entra por la derecha del motor y lo hace girar en sentido contrario.

Servomotor:

Es un motor especial, tiene un motor DC interno, un conjunto de engranajes y un circuito electrónico(Driver), el driver en conjunto con el encoder, son el circuito que gobierna posición, torque y velocidad. La señal de posicionamiento está en función de PWM.

FSM:

Tiene un número finito de estados, hacen al código más eficiente, más fácil de depurar y ayuda a organizar el flujo de programa. Promueve buenas técnicas de diseño de firmware. Permite crear un sistema manejado por eventos.



Una FSM tiene un conjunto de estados, entradas, salidas y transiciones de estado.

Entradas: evento requerido para generar una salida.

Transiciones: la flecha del diagrama de flujo representa una transición de estado.

Salidas: Acción necesaria como respuesta a una entrada.

Estados: Un estado indica que debe realizar un sistema cuando un evento sucede.

Estado	Entrada	Salida
LED Off	Timeout	Ninguna
LED On/ LED Blink	Timeout	Apagar LED
LED Off	Boton presionado	Encender LED
LED On	Boton presionado	Iniciar parpadeo LED
LED Blink	Boton presionado	Apagar LED

Existen varias formas de programar una FSM

- If else
- Switch case
- Con structs y enums
- Con lookup tables
- Con punteros a funciones

Ejemplo de una FSM

```

switch(estado){
    case led_off:
        switch(system_input){
            case button_press:
                encender led;           // Salida
                estado = led_on;      // Transicion de estado
                break;
            case timeout:
                break;                 // hacer nada
        }
        break;
    case led_on:
        switch(entrada){
            case button_press:
                parpadear led;       // Salida
                estado = led_blink;  // Transicion de estado
                break;
            case timeout:
                apagar led;          // Salida
                estado = led_off;   // Transicion de estado
                break;
        }
        break;
    case led_blink:
        switch(entrada){
            case button_press:
                apagar led;          // Salida
                estado = led_off;   // Transicion de estado
                break;
        }
}

```

Hola ATtiny:

```

#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    DDRB = 0x08; //Configuracion del puerto

    //Pepadear
    while (1) {
        PORTB = 0x00; _delay_ms(500); //Tambien se puede hacer PORTB &= ~(1 << PB3);
        PORTB = 0x08; _delay_ms(500); //Tambien se puede hacer PORTB |= (1 << PB3);
    }
}

```

4. Laboratorio 2: Lavadora automatica

GPIOs, interrupciones, timers y FSM



Figura 1: Lavadora automática

Desarrollar un control automático de lavadora simplificado (figura 1), utilizando leds, displays de 7 segmentos, un motor DC y el microcontrolador ATtiny4313 que realiza las siguientes etapas en función del nivel de carga escogido como se presenta en la tabla.

Nivel de carga	Baja	Media	Alta
Suministro de agua	1 segundos	2 segundos	3 segundos
Lavar	3 segundos	5 segundos	7 segundos
Enjuagar	2 segundos	4 segundos	5 segundos
Centrifugar	3 segundos	5 segundos	6 segundos

Tabla 1: Tiempos de secuencia de lavado

Se debe tener 3 botones/switches con leds correspondientes a la escogencia del nivel de la carga (baja, media, alta), un botón de inicio/pausa. Debe tener dos displays de 7 segmentos que indiquen el tiempo restante del total de la secuencia, un led que represente cada uno de los estados de lavado (Suministro de agua, lavar, enjuagar, centrifugar) y un motor WDM350FGA (60W, 24V).

La temporización se debe realizar utilizando uno de los temporizadores del microcontrolador.

Para hacer la lectura de los botones, como también del fin de cuenta del Timer, debe realizarlo usando interrupciones (**IMPORTANTE**).

El motor debe estar activo en las etapas de lavar, enjuagar y centrifugar.

Es opcional agregar leds adicionales componentes adicionales para representar mas adecuadamente el comportamiento de la lavadora.

División del problema:

- Interrupciones básicas.
- Temporizador descendente en display de 7 segmentos.

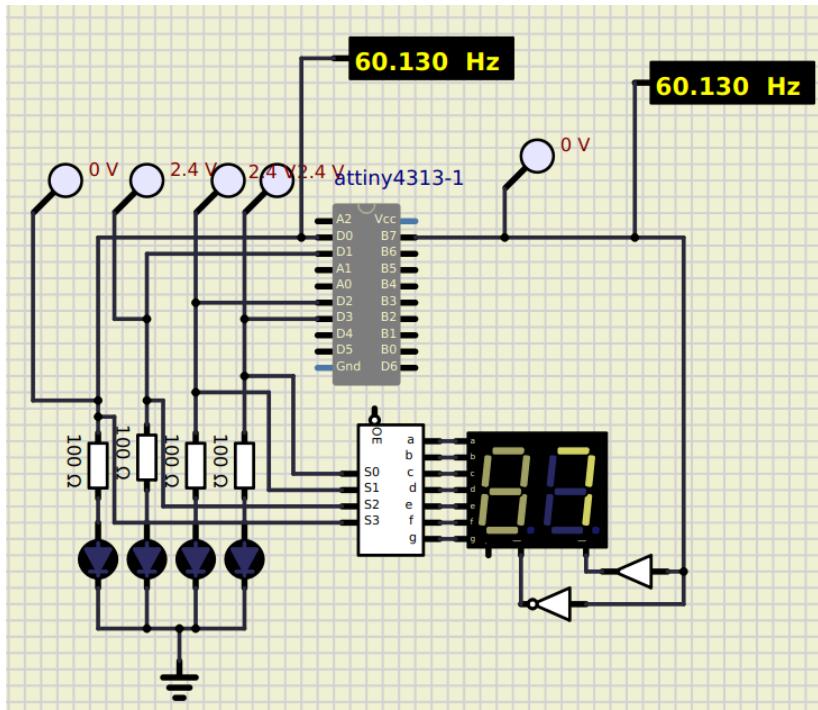
La idea es generar el temporizador que va a contar los tiempos del ciclo de lavado y los va a mostrar descendente en un display de 7 segmentos.

Configuración del display:

Se utiliza un convertidor BCD a 7 segmentos y los pines del puerto D, D3, D2, D1, D0. Se modifica el registro DDRD para establecer estos pines como salidas. Además se utiliza el pin B7 para multiplexar los display. También se dejan configurados como salidas los pines B6 para monitorear la frecuencia de conteo y B5 para apagar y encender los LED

```
/// **** Setup function ****
void setup(){
    //--- D3, D2, D1, D0 as outputs -----
    DDRD = (1 << DDD3) | (1 << DDD2) | (1 << DDD1) | (1 << DDD0) | (1 << DDD3);

    //--- B7, B6 s outputs -----
    DDRD = (1 << DDB7) | (1 << DDB6) | (1 << DDB5)
```



Configuración del temporizador e interrupciones

Lo primero es poner el temporizador a contar por segundo, se utiliza la fórmula

$$F_{oC0A} = F_{clk}/2N(1 + OCR0A)$$

Donde F_{oC0A} es el valor de frecuencia que se desea, en este caso es 1Hz, F_{clk} es la frecuencia del MCU (16Mhz), N es el factor del prescaler y $OCR0A$ es el valor de comparación. Entonces

$$1Hz = 16MHz/2(1024)(1 + OCR0A)$$

$$OCR0A = -1 + 16MHz/2(1024)$$

$$OCR0A = 7811.5s$$

Como se puede contar hasta 255, ya que el timer0 es de 8 bits, entonces se divide este tiempo entre 40 y por lo tanto, la frecuencia aumenta en un factor de 40,

contando 40 veces por segundo (40Hz). Es necesario regular esto en el código dentro de la ISR.

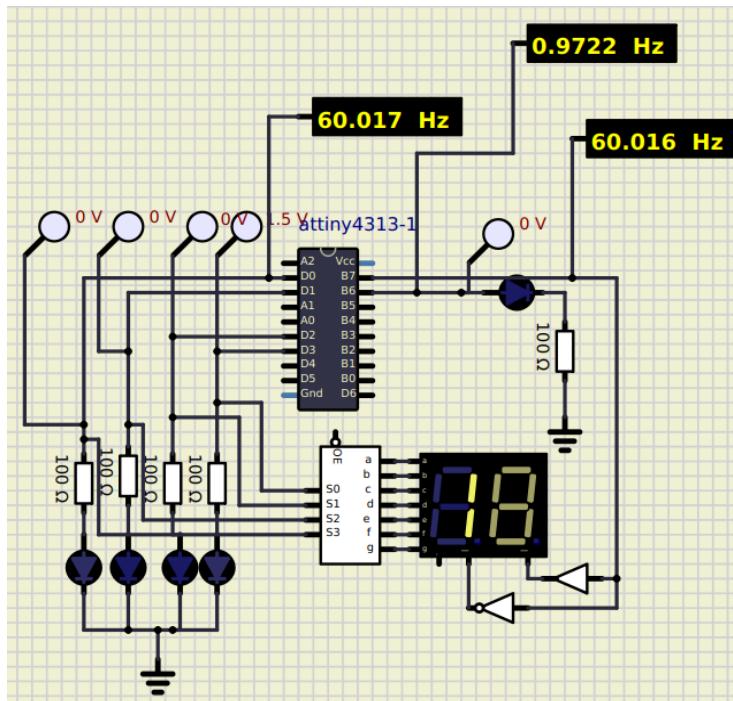
$$OCR0A = 7811.5 / 40 = 195.25s$$

Se define entonces COMPARE_TIME como 195 segundos

Luego, los registros a utilizar son TIMSK para habilitar OCIE0A, TCCR0A para poner el contador en modo CTC, TCCR0B para ajustar el prescaler a clk/1024 y OCR0A para establecer el valor de comparación. Además de llamar a la función sei() para habilitar interrupciones globales.

```
// ----- Timer0 setup -----
TCCR0A = (1 << WGM01); // CTC mode - Clear timer on compare
OCR0A =COMPARE_TIME; // Generate 40Hz interruption
TIMSK = (1 << OCIE0A); // Enable compare match A
TCCR0B = (1 << CS02) | (1 << CS00); // Set prescaler clk/1024
sei();
```

En la figura se puede observar la frecuencia de conteo en el pin B6 muy cercana a 1Hz y el conteo regresivo funciona correctamente.



-
- Botón de inicio/pausa.
 -
 - Escogencia del motor y de la configuración de encendido.
 - Encender motor
 - FSM:
 - Diagramas de flujo
 - Circuito completo

Microcontroladores

Curso Online Edx

Link: <https://learning.edx.org/course/course-v1:UTAustinX+UT.6.10x+3T2022/home>

Dualok Fonseca Monge

1. What's an embedded system:

an embedded system is a combination of electrical, mechanical, optical, chemical, computer software, all combined for a dedicated purpose, it has a computer embedded inside

Examples of embedded systems:

pacemakers: helps heart beat at a regular rate, its purpose is to help people live longer

hard drive: it has mechanical parts that rotate, electrical parts which control the motor and optical parts to read and write data from the disk

Voltage detector: allows to monitor the conditions of the operation of any system that you're testing, it has a rich interface that allows you to track information.

Motor controller: it has an interface to control the motor so it can be used in larger applications.

Game controller: it uses mechanical switches aka buttons to interface the game with the player.

Ipad: Music player

2. Numbers representation:

Base 2 system (Binary):

$$01101010 = 2^6 + 2^5 + 2^3 + 2^1 = 106$$

Hexadecimal system:

0xhhhhh

$$0x2a9f = 2 \times 16^3 + 10 \times 16^2 + 9 \times 16^1 + 15 \times 16^0 = 10911$$

Converting to binary: 0010 1010 1001 1111

hex numbers are used to represent addresses, contents of reg memory

A nibble is defined as a 4 binary digits or 1 hex number, hexadecimal digits are represented by adding \$ or 0x before the number. In C we will add the 0x

Hex Digit	Decimal Value	Binary Value
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A or a	10	1010
B or b	11	1011
C or c	12	1100
D or d	13	1101
E or e	14	1110
F or f	15	1111

Table 2.1. There are 16 hexadecimal digits.

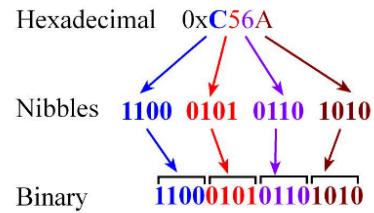


Figure 2.2. Notice that each hex digit maps into 4 binary bits.

Signed vs unsigned numbers

Signed: Min ----- 0 ----- Max

Unsigned: 0 ----- Max

Example:

Unsigned 10001101: $128 + 8 + 4 + 1 = 141$

Signed: 10001101: The MSB represents the sign of the number, 0 positive, 1 negative.

Precision and bytes:

A byte consist of 8 bits

b7|b6|b5|b4|b3|b2|b1|b0

$$= 128 \cdot b7 + 64 \cdot b6 + 32 \cdot b5 + 16 \cdot b4 + 8 \cdot b3 + 4 \cdot b2 + 2 \cdot b1 + b0$$

Precision is the number of distinct or different values, we express precision in alternatives, bytes or bits. Alternatives is the total number of possibilities. For ex; an 8 bit ADC can generate 256 different analog outputs

Example	Precision in alternatives	Precision in bits
Fingers	10	a little more than 3
Months	12	between 4 and 5
Human vertebrae	33	a little more than 5
Cards in a deck	52	between 5 and 6
Days in a year	365-366	between 8 and 9
Minutes in a day	1440	between 10 and 11
Men in a legion	5000	a little more than 12

2s complement:

flip every bit and then add 1, its the same as left every bit as it is now, from LSB to MSB until we find the first 1, then we flip the rest excluding that first 1

For ex:

What's the complement and the signed and unsigned value of 10011000?

$$\text{Complement} = (10011000)^* = 01100111 + 1 = \textcolor{red}{01101000}$$

$$\text{Unsigned Value} = 128 + 16 + 8 = 152$$

$$\text{Signed Value} = \textcolor{red}{64 + 32 + 8 = -104}$$

Another way to calculate it is $= -128 + 16 + 8 = -104 \rightarrow$ its **way easier** and faster for god sake!

Exercises:

1. Give the representations of the decimal **45** in 8-bit binary and hexadecimal

$$45/2 = 22/2 = 11/2 = 5/2 = 2/2 = 1$$

1 0 1 1 0 1

ans = bin = 0010 1101 hex = 0x2D

2. Convert the signed binary number **11011010** to decimal.

$$-128+64+16+8+2 = -38$$

Words and halfwords:

A word on the ARM cortex M will have 32 bits. There is 2^{32} different unsigned bit numbers, the smallest is 0 and the largest is about 4 million

A halfword or double byte contains 16bits. Similar to the unsigned algorithm we can use the basis to convert a decimal number into a signed binary

Number	Basis	Need it	bit	Operation
-100	-128	yes	bit 7=1	subtract -100 - -128
28	64	no	bit 6=0	none
28	32	no	bit 5=0	none
28	16	yes	bit 4=1	subtract 28-16
12	8	yes	bit 3=1	subtract 12-8
4	4	yes	bit 2=1	subtract 4-4
0	2	no	bit 1=0	none
0	1	no	bit 0=0	none

Table 2.2. Example conversion from decimal to signed 8-bit binary.

so basically, if we have a negative number we will always use the MSB as 1, so will need $-128+16+8+4$ in order to generate a 100

-128	64	32	16	8	4	2	1
1	0	0	1	1	1	0	0

When dealing with numbers in a computer it's convenient to memorize some powers of 2

exponent	decimal
2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^{10}	1024 about a thousand
2^{11}	2048
2^{12}	4096
2^{13}	8192
2^{14}	16384
2^{15}	32768
2^{16}	65536
2^{20}	about a million
2^{30}	about a billion
2^{40}	about a trillion

Here's the way to specify them in a C language

size	<code>unsigned</code>	<code>signed</code>
8 bits	<code>unsigned char</code>	<code>signed char</code>
16 bits	<code>unsigned short</code>	<code>short</code>
32 bits	<code>unsigned long</code>	<code>long</code>
64 bits	<code>unsigned long long</code>	<code>long long</code>

In C99, the number of bits has been standardized. In C99, we have these data types

size	<code>unsigned</code>	<code>signed</code>
8 bits	<code>uint_8</code>	<code>int_8</code>
16 bits	<code>uint_16</code>	<code>int_16</code>
32 bits	<code>uint_32</code>	<code>int_32</code>
64 bits	<code>uint_64</code>	<code>int_64</code>

3. Fields of usage of embedded systems

Cars, military, industry, people, phones, house, consumer electronics,

4. Components of an embedded system

Computer x86, ARM, memory, I/O(Input output) Interface -> has a lot of components -

Hardware - Electrical - Software

Abilities that are extremely important

Testing, Power management, profit, sizing

The idea of time is extremely important, we need to have the correct answer at the right time.

5. Microcontrollers

to understand the term embedded microcomputer system consider each word separately

Micro = Small - Embedded = hidden inside - Computer: has a processor and memory and means to exchange data with the real world.