

ESCUELA DE INGENIERÍA INFORMÁTICA

Grado en Ingeniería Informática



PERIFÉRICOS E INTERFACES

PRÁCTICAS EN LABORATORIO

Práctica 1

Interfaces paralelos en el μ C ATmega 2560

Última actualización: 16 septiembre 2024

Contenido

1	Competencias y objetivos de la práctica	4
2	Documentación previa	5
3	Introducción a la plataforma hardware de prácticas	6
4	Descripción general de la tarjeta Arduino Mega 2560	7
4.1	Componentes básicos	7
4.2	Entradas/salidas en la tarjeta Arduino Mega 2560	11
4.2.1	Entradas/salidas digitales de propósito general	13
4.2.2	Puertos de comunicación serie	16
4.2.3	Salidas de control de potencia (PWM)	19
4.2.4	Entradas analógicas (A0-A15)	20
4.2.5	Líneas de alimentación y reset	22
4.3	Sistema de interrupciones del microcontrolador ATmega 2560	23
4.3.1	Configuración y habilitación de interrupciones externas.....	24
4.3.2	Subrutina de servicio de una interrupción (ISR).....	29
4.3.3	Funciones Arduino para la gestión de interrupciones	30
5	Tarjeta de expansión	32
5.1	Componentes básicos	32
5.1.1	Visualizador de 7 segmentos.....	34
5.1.2	Teclado matricial.....	35
5.1.3	Pulsador.....	35
5.1.4	Zumbador y/o altavoz	36
5.1.5	Otros componentes.....	36
5.2	Diagrama de conexiones de la tarjeta expansión.....	37
5.3	Detalles del conexionado de los periféricos	40
5.3.1	Visualizador 7-segmentos	40
5.3.2	Teclado matricial 4x3.....	41
5.3.3	Pulsadores	42
5.3.4	Zumbador/altavoz	43
6	Entorno de programación Arduino	44
7	Realización práctica	46
7.1	Tareas previas (no entregables).....	48
7.1.1	Tarea 1. Salidas digitales	48
7.1.2	Tarea 2. Comunicación serie.....	48
7.1.3	Tarea 3. Sincronización por consulta de estado	48

7.1.4	Tarea 4. Visualización sincronizada por interrupción externa.....	49
7.2	Aplicación a desarrollar (entregable para evaluar).....	50
7.2.1	Especificaciones de diseño.....	50
7.2.2	Mejoras del diseño: Sensor de temperatura (opcional)	51
8	Entrega del informe de práctica	51

1 Competencias y objetivos de la práctica

Esta práctica de la asignatura Periféricos e Interfaces se centra en el uso del sistema de entrada/salida de un sistema empotrado (sistema que integra los componentes básicos de un computador, procesador, memoria y entrada/salida, en un mismo chip) ya que ofrece más facilidades para el desarrollo de aplicaciones a bajo nivel que, por ejemplo, un computador tipo PC. Este tipo de computadores tiene un hardware más complejo y difícilmente utilizable a bajo nivel ya que el sistema operativo impediría al usuario acceder directamente a los recursos hardware en aras de garantizar el funcionamiento fiable y seguro del computador, además de garantizar la integridad de la información perteneciente a otros usuarios. Es por ello que, en estos sistemas, el usuario accede a los recursos hardware de entrada/salida a través de llamadas al sistema operativo. Por tanto y con objeto de trabajar directamente con los interfaces de entrada/salida y familiarizarnos con ellos, es más adecuado el uso de un sistema empotrado que nos permite trabajar con todos los recursos hardware sin limitaciones.

El sistema empotrado seleccionado está basado en el microcontrolador AVR ATmega 2560 de Atmel (empresa comprada por Microchip en 2016) que cuenta con una gran cantidad de información y soporte de la comunidad de desarrolladores. Este sistema (disponible en la tarjeta Arduino Mega 2560) se complementa con una tarjeta con diferentes periféricos que se pueden conectar fácilmente a los puertos del microcontrolador. Como alternativa a trabajar con los componentes físicos directamente, se propone el uso del simulador Proteus con el que se puede diseñar y simular todas las prácticas planificadas de la asignatura presentando grandes ventajas, sobre todo en condiciones de trabajos vía telemática o fuera del laboratorio.

El microcontrolador AVR ATmega 2560 está dotado de un procesador de 8 bits, varios tipos de memorias, timers y un potente sistema de entrada/salida y multifuncional que lo hacen idóneo para el desarrollo de una gran cantidad de aplicaciones en múltiples campos (automatización, robótica, hobby, control de dispositivos, ...).

Los diferentes interfaces, visibles al programador de bajo nivel (como es nuestro caso) como un conjunto de registros, serán programados para llevar a cabo el control y el intercambio de información con los dispositivos externos o periféricos. Esto permitirá poner en práctica los conocimientos teóricos adquiridos en las clases de teoría centrados fundamentalmente en las técnicas de sincronización (consulta de estado e interrupciones) y las transferencias de datos por programa. Con la realización de esta práctica se pretende que el estudiante alcance las siguientes competencias:

1. Capacidad para el desarrollo de interfaces paralelos que permitan la conexión de dispositivos periféricos que requieran de ese tipo de interfaces haciendo uso de los recursos disponibles en el sistema empotrado.
2. Capacidad para entender los diversos aspectos software y hardware involucrados en los métodos de sincronización y transferencia de datos.
3. Capacidad para aprender y aplicar nuevos conceptos de forma autónoma e interdisciplinar.
4. Capacidad para emplear la creatividad en la resolución de los problemas.

5. Capacidad para trabajar en equipo y colaborar eficazmente con otras personas.

Para alcanzar estas competencias se plantea la consecución de los siguientes objetivos:

1. Conocer y usar un sistema empuotrado, como el disponible en la plataforma Arduino Mega 2560, para el desarrollo, ejecución y depuración de programas sencillos. Se hará uso del entorno de programación de la plataforma Arduino o el simulador Proteus, disponibles en los laboratorios. El estudiantado también dispondrá del simulador (versión estudiante o “lite”) para poder trabajar fuera del laboratorio.
2. Conocer y entender los aspectos básicos de funcionamiento y programación de algunos de los interfaces de entrada/salida del sistema empuotrado.
3. Realizar programas sencillos que impliquen operaciones de entrada/salida con los periféricos seleccionados utilizando la consulta de estado como método de sincronización.
4. Conocer y entender la gestión de interrupciones en el sistema empuotrado, así como los recursos de la plataforma para facilitar su uso.
5. Diseñar y programar una aplicación sencilla que implique operaciones de entrada/salida con los periféricos combinando la consulta de estado y las interrupciones como métodos de sincronización y las transferencias de datos por programa.

2 Documentación previa

La documentación básica a utilizar para la realización de esta práctica (además de los propios relacionados con la teoría del módulo 1 y 2) está disponible en el Campus Virtual de la ULPGC en el curso que corresponde a Periféricos e Interfaces. La documentación mínima a manejar será la siguiente:

- Documento 1: Enunciado de la práctica (este documento)
- Documento 2: Presentación de la práctica
- Transparencias del módulo 1 y 2
- Datasheet Atmel ATmega 2560 (435 páginas)
 - o http://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf
- Software: Entorno de programación de Arduino y simulador Proteus
 - <https://www.arduino.cc/en/Main/Software>
 - <https://www.hubor-proteus.com/>
- Internet: Búsqueda de cualquier otra información que se considere necesaria

3 Introducción a la plataforma hardware de prácticas

Las prácticas de la asignatura Periféricos e Interfaces se desarrollarán sobre la plataforma hardware mostrada en la Figura 1 que será simulada en el simulador Proteus. El sistema consta de dos elementos básicos: una **tarjeta Arduino Mega 2560** basada en el microcontrolador AVR ATmega 2560 de 8 bits y una **tarjeta de expansión** con múltiples componentes (teclado, visualizador 7-segmentos, pantalla OLED y LCD, memoria I2C, etc..) cuyas señales de control se llevan a conectores ubicados en la propia tarjeta de expansión con el objetivo de que sean fácilmente accesibles para interconectarlos al microcontrolador mediante cables, que pueden ser fácilmente reconfigurados de acuerdo a las necesidades de cada aplicación. Para las aplicaciones que se plantean a lo largo de las prácticas, es más que suficiente las conexiones preestablecidas cuyos detalles se irán presentando en los sucesivos enunciados de prácticas. El estudiante, salvo excepciones, solo ha de centrarse en el desarrollo de software haciendo uso del entorno de programación del simulador Proteus o el entorno gratuito de Arduino si se trabajase con hardware real.

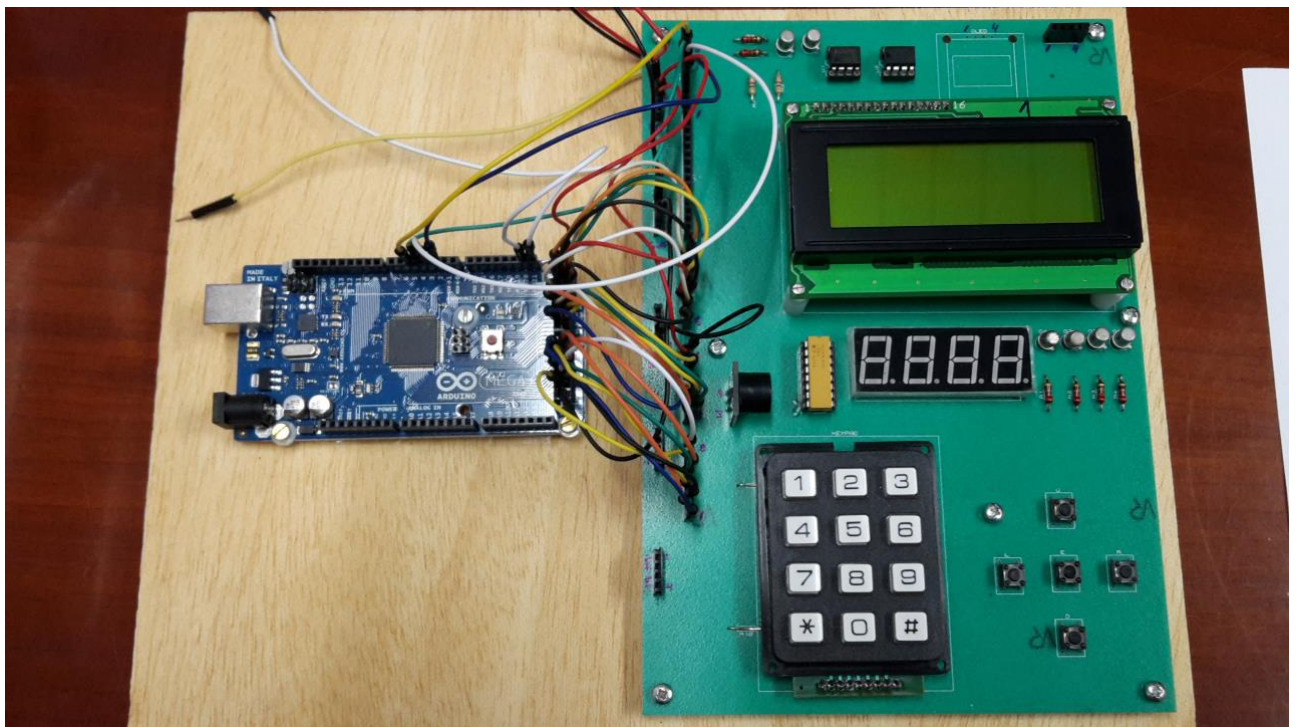


Figura 1. Arduino Mega 2560 + tarjeta de expansión

En los siguientes apartados, realizaremos una breve descripción de la estructura de las tarjetas que conforman nuestro hardware experimental y del entorno de programación Arduino. Una vez estudiado los diferentes aspectos del hardware, tendremos una visión general del sistema que nos va a permitir afrontar con éxito el desarrollo de las primeras aplicaciones planteadas en el apartado de "realización práctica".

4 Descripción general de la tarjeta Arduino Mega 2560

4.1 Componentes básicos

La tarjeta Arduino Mega 2560 es una plataforma hardware de tipo "open-source" y bajo coste dotada de un microcontrolador ATmega 2560 de 8 bits de bajo consumo con amplias capacidades de entrada y salida que le permiten un gran desempeño en múltiples campos de la informática y la automatización. Las diferentes tarjetas Arduino se integran en un entorno de programación que posibilita las tareas básicas en el desarrollo de software (edición, compilación, carga y depuración) de una forma cómoda y amigable para el usuario que le permiten trabajar en lenguaje C/C++ con el soporte de un gran número de librerías.

La figura 2 muestra una imagen de la tarjeta con los principales componentes.

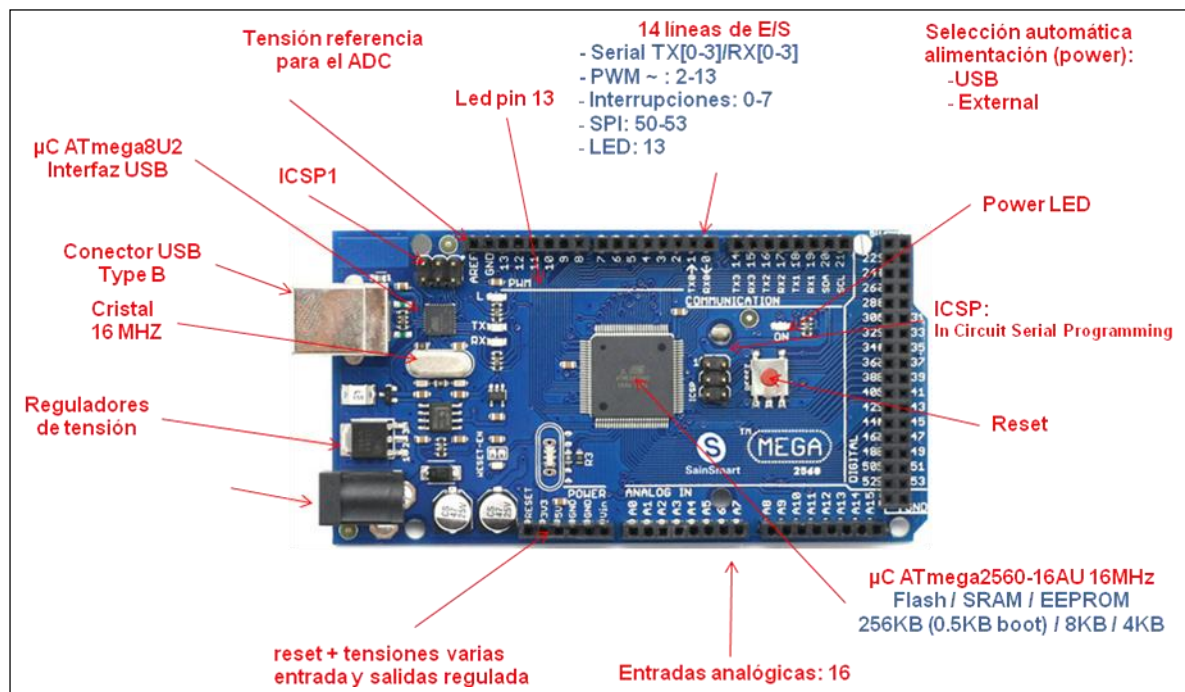


Figura 2. Componentes básicos de la tarjeta Arduino Mega 2560

El componente más importante de la tarjeta es el microcontrolador ATmega 2560 de 100 patas (o pines) que integra básicamente los siguientes elementos: procesador, memoria e interfaces de entrada y salida, entre otros. Veamos las características más importantes de cada uno de ellos:

Procesador

- 8 bits, low power, arquitectura RISC
- 131 instrucciones
- 1 instrucción por ciclo (1 MIPS per MHz)
- 32 registros de 8 bits de propósito general
- Modos de ahorro de energía: Idle, power-down, power-save, ADC noise reduction y standby

Memoria

- Memoria Flash (256 KB), SRAM (8KB) y EEPROM (4 KB)

Entrada/salida

Las líneas de entrada/salida del microcontrolador son multifuncionales, es decir, son compartidas por los diferentes interfaces que proporcionan las siguientes funcionalidades:

- 86 líneas de entrada/ salida digital de propósito general agrupadas en puertos de 8 y 6 bits
- Interrupciones externas
- USART programables (comunicación serie)
- SPI (Serial Peripheral Interface)
- TWI: two-wire interface (i2c)
- ADC: 16 canales, 10-bit resolución
- 15 salidas de timers

Otros

- 1 comparador analógico
- Timer Watchdog, programable y con oscilador interno
- JTAG

En la figura3 se muestra un diagrama de bloques del microcontrolador AT mega 2560 donde se pueden ver todos sus componentes internos y los puertos de entrada y salida.

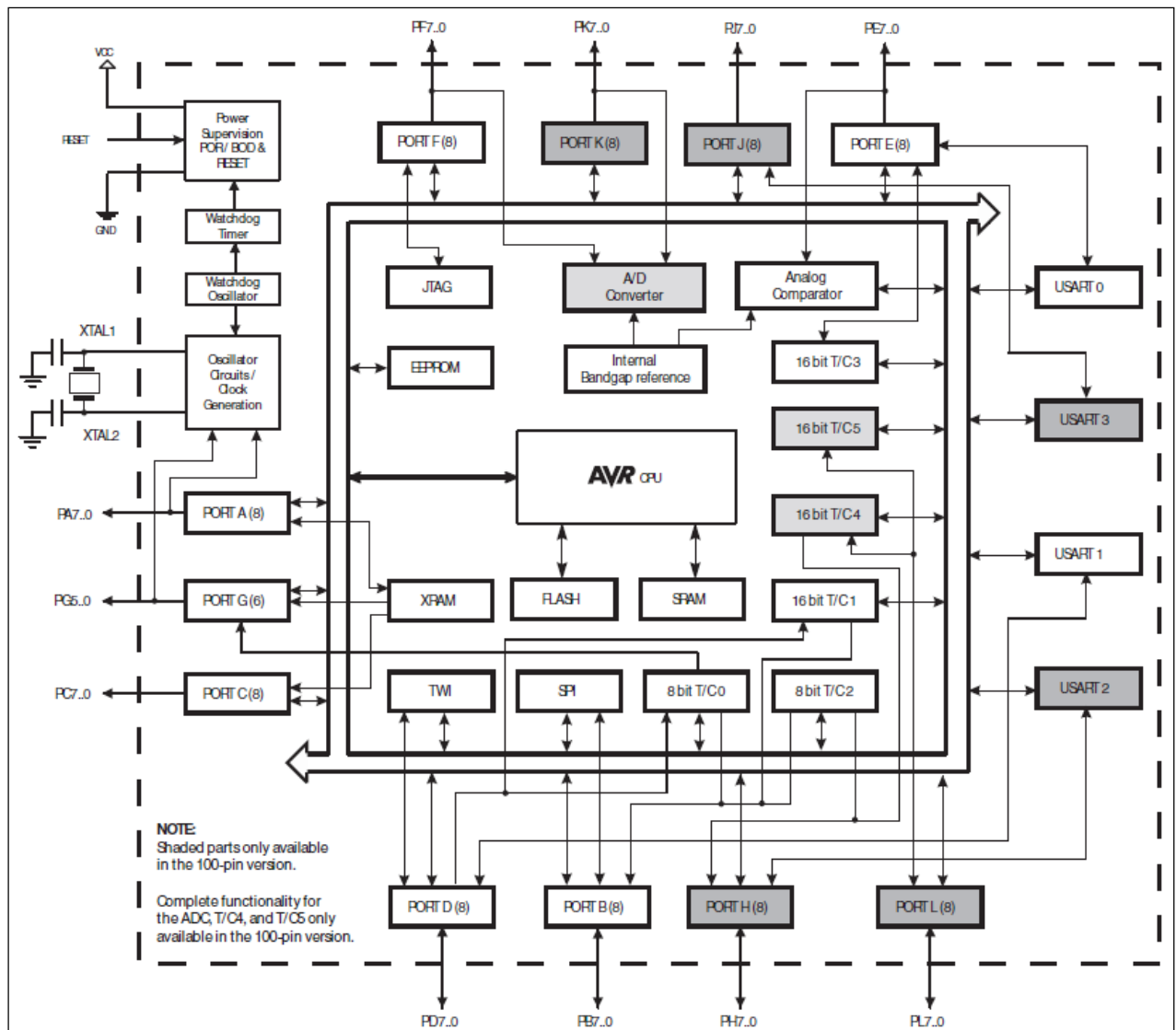


Figura 3. Diagrama de bloques del microcontrolador ATmega 2560

La figura 4 muestra el encapsulado y patillaje (pines) del microcontrolador, así como otras características en cuanto a consumos, temperatura de funcionamiento y versiones.

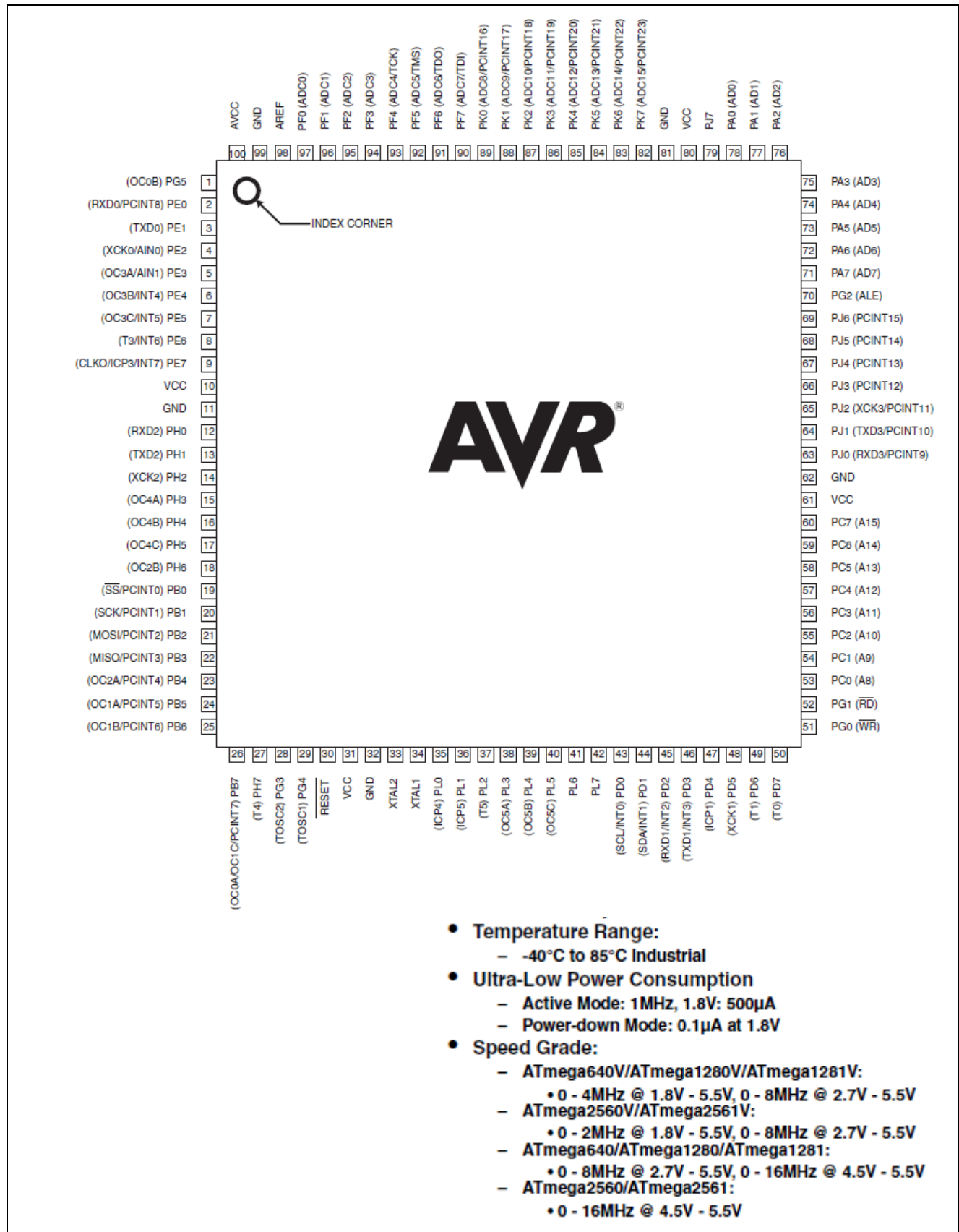


Figura 4. Encapsulado y patillaje (pines) del microcontrolador de 8 bits ATmega 2560

4.2 Entradas/salidas en la tarjeta Arduino Mega 2560

El microcontrolador ATmega 2560 puede controlar hasta un total de 86 señales de entrada-salida a través de las que se comunica con el exterior. Estas señales de entrada/salida se agrupan, en su mayoría, en grupos de 8 bits denominados puertos. Existen un total de 11 puertos (10 puertos de 8 bits y 1 puerto de 6 bits) etiquetados con una letra (A-B-C-D-E-F-G-H-J-K-L):

PA[7:0] PB[7:0] PC[7:0] PD[7:0] PE[7:0] PF[7:0] **PG[5:0]** PH[7:0] PJ[7:0] PK[7:0] PL[7:0]

Cada bit de un puerto se lleva al exterior a través de una pata (pin) del microcontrolador con lo que se obtiene un total de **86 pines** asociados a las entradas/salidas. En la tarjeta Arduino, gran parte de los pines del microcontrolador son llevados a conectores externos para que el usuario pueda acceder a ellos y usarlos para sus aplicaciones. De los 86 pines, sólo 70 tienen conexión con los conectores de la tarjeta Arduino Mega 2560. Los 16 pines que faltan (PE2-6-7, PD4-5-6, PG3-4, PH2-7, PJ2-3-4-5-6-7) no se podrán utilizar salvo que, por algún otro medio, se acceda directamente al pin del microcontrolador para realizar la conexión con el exterior. En el simulador Proteus sí tendremos acceso a todos ellos.

En la figura 5, podemos apreciar los diferentes conectores y el etiquetado de los pines con un número que luego se podrá utilizar como parámetro en las diferentes funciones del entorno de programación de Arduino para realizar operaciones de entrada-salida a través del pin. En el circuito impreso de la tarjeta se puede apreciar el rotulado de los pines y el agrupamiento en 5 grupos según la funcionalidad principal:

- DIGITAL. Entradas/salidas digitales de propósito general: pines 22-53
- COMMUNICATION. Canales de comunicación serie: pines TX0/RX0,..., SDA/SCL
- PWM. Salidas de control de potencia: pines 2-13 (12 canales)
- ANALOG IN. Entradas analógicas: pines A15-A0
- POWER. Líneas de alimentación y reset: pines Vin, GND, +5v, +3.3v, reset

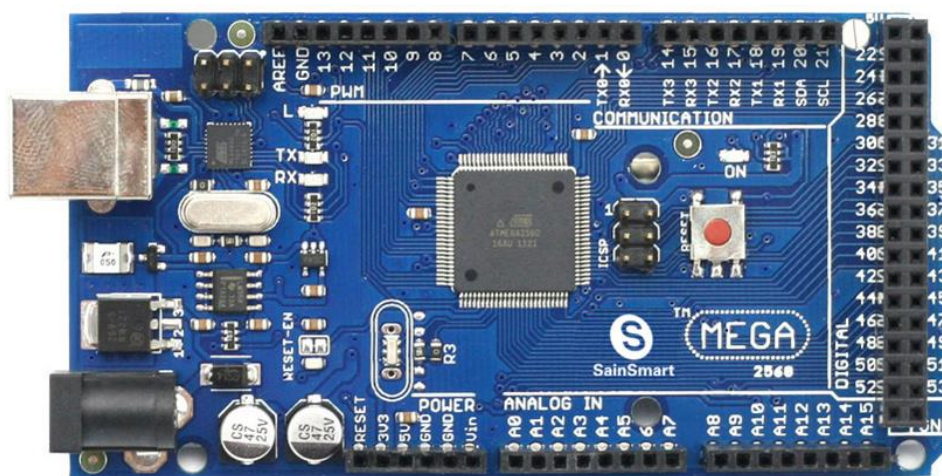


Figura 5. Tarjeta Arduino Mega 2560 (microcontrolador ATmega2560)

La mayoría de los pines del microcontrolador son **multifuncionales**. La funcionalidad queda definida por el interfaz que se utilice, previa programación (canal serie, timers, SPI, TWI, ...).

Los interfaces son visibles al programador a través de un conjunto de registros que se mapean en el espacio de direccionamiento de memoria del microcontrolador. Concretamente, el microcontrolador ATmega 2560 mapea sus registros en el rango de direcciones 0x20-0x1FF ($0x200-0x20 = 512-32 = 480$ bytes) aunque, muchas de estas direcciones, quedan libres y reservadas para usos futuros del microcontrolador. El acceso a los registros se realiza con instrucciones nativas del microcontrolador del tipo IN/OUT (acceso al espacio de direccionamiento de entrada/salida) o del tipo LOAD/STORE (acceso al espacio de direccionamiento de memoria de 64KB).

Las instrucciones IN/OUT solo pueden acceder a un espacio de entrada/salida de tamaño 64 bytes (0x00-0x3F) en el que se encuentra mapeado un cierto número de registros (port mapped I/O). Por otra parte, todos los registros están mapeados como memoria (memory mapped I/O) a los que se puede acceder con instrucciones del tipo LD/ST, en el rango de direcciones 0x020-0x1FF (480 direcciones o registros).

En la figura 6 se muestra los diferentes pines de los conectores de la tarjeta Arduino Mega 2560 especificándose las diferentes funcionalidades que pueden tener cada uno de ellos y los puertos del microcontrolador a los que se conecta.

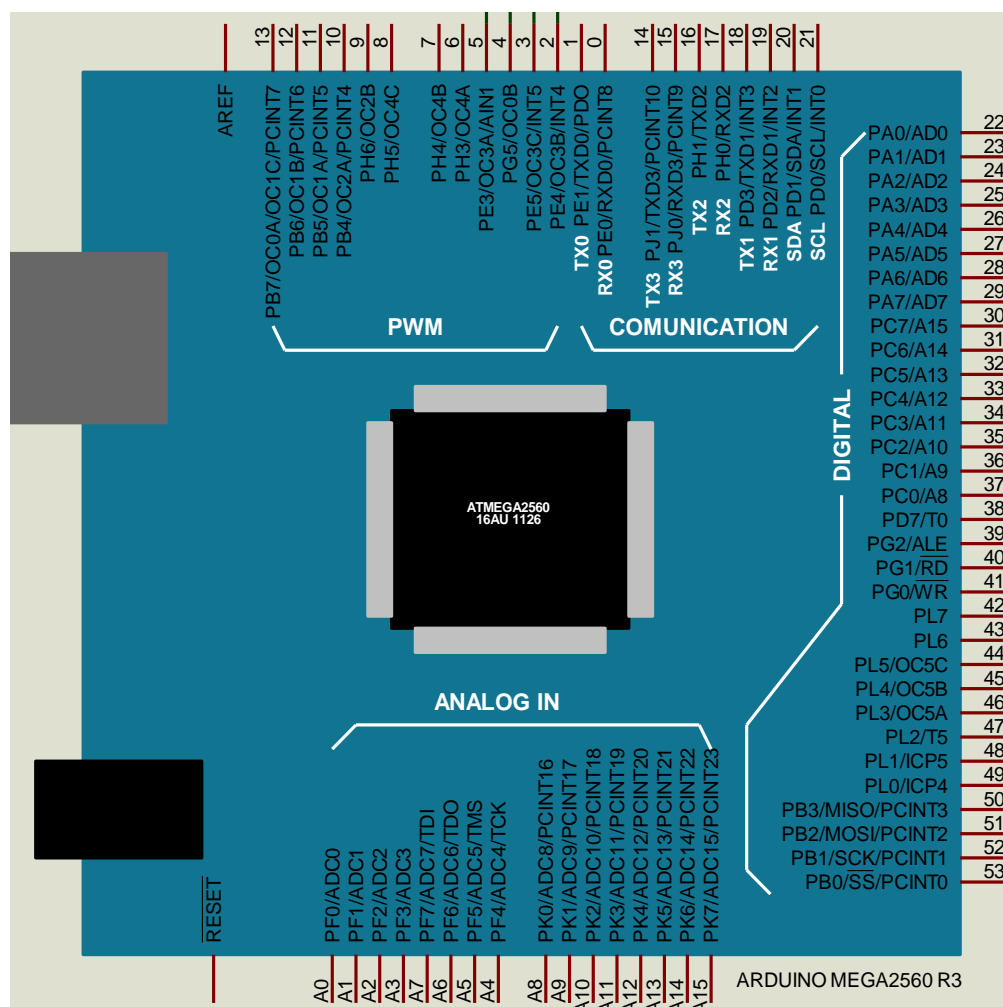


Figura 6. Multifuncionalidad de los pines del Arduino Mega 2560

4.2.1 Entradas/salidas digitales de propósito general

Las 86 líneas de entrada/salida asociadas a los puertos del microcontrolador, se pueden programar para ser utilizadas como entradas o salidas binarias (1 o 0) de propósito general. Para esta funcionalidad, cada puerto dispone de un interfaz con tres registros internos que se utilizan para configurar cada bit de un puerto como de entrada o de salida y, luego, realizar las operaciones de entrada y salida de datos binarios.

4.2.1.1 Registros internos del interfaz de un puerto

El interfaz de cada puerto x consta de tres registros: $DDRx$, $PORTx$, y $PINx$, siendo " x " la letra del puerto (A-B-C-D-E-F-G-H-J-K-L). Por ejemplo, para el interfaz del puerto B, tendríamos los registros: $DDRB$, $PORTB$ y $PINB$.

Los nombres de todos estos registros están definidos en el entorno de programación de Arduino lo que permite al programador leer y escribir en ellos desde el lenguaje de programación C/C++ de una manera fácil y directa.

Registro $DDRx$: Port x Data Direction Register

A través de este registro se define si un bit " n " del puerto " x " es de entrada o de salida.

Si $DDRx_n$ es "0", el bit n del puerto x (Px_n) será de entrada

Si $DDRx_n$ es "1", el pin n del puerto x (Px_n) será de salida

Ejemplo para el puerto A de 8 bits ($PA[7:0]$)

$DDRA = B00001111$; // Define los pines $PA[7:4]$ de entrada y los pines $PA[3:0]$ de salida

$DDRA = 0x0F$; // la misma operación, pero en hexadecimal

B: Binario

0x: Hexadecimal

$PORTx$: Port x Data Register

La salida de datos se realiza escribiendo en el registro $PORTx$. Los pines asociados a un puerto digital están en estado de alta impedancia cuando la condición de "reset" está activa. El efecto de escribir en un bit de este registro ($PORTx_n$) depende de si el bit del puerto está previamente configurado como entrada o salida. En la siguiente tabla, se resume el efecto de escribir un "1" o "0" en un bit del puerto, dependiendo de si éste es de entrada o de salida.

Operación	$PORTx_n$ de entrada ($DDRx_n=0$)	$PORTx_n$ de salida ($DDRx_n=1$)
Escribir un "1"	Conecta la resistencia de pull-up	1
Escribir un "0"	Desconecta la resistencia de pull-up	0

Ejemplos:

```
// Programación de la dirección de los pines del PORTL
```

```
DDRL=B11110000; // PL[7:4]: pines de salida      PL[3:0]: pines de entrada
```

```
// Escritura en el puerto PORTL
```

```
PORTL= B11110011; // PL[7:4]: Salidas a "1"; PL[3:2]: Res. pull-up desconectadas; PL[1:0]: Res. pull-up conectadas
```

```
// Programar solo los pines 7 y 0 del puerto B de salida
```

```
DDRB = DDRB | B1000001;
```

PINx: Port x Input Pins

La entrada de datos se realiza leyendo el puerto PINx. A través del registro PINx se puede leer el estado de los pines del puerto x, independientemente de si los pines del puerto x están programados como de entrada o de salida.

Ejemplos:

```
// Lectura de los pines del PORTA (PA[7:0])
```

```
val1 = PINA;      // lectura de los 8 pines del PORTA
```

```
// Lectura de los pines del PORTL
```

```
val2=PINL;        //lectura de los 8 pines del PORTL
```

En la figura 7 se muestra los registros del interfaz del PORTA tal y como aparecen en el manual de usuario del microcontrolador ATmega 2560. Para cada registro se especifica su nombre, el nombre de cada uno de sus bits y la dirección en el espacio de E/S o la dirección del espacio de memoria (entre paréntesis) donde se ubica o mapea el registro.

Para el caso particular del puerto A, vemos que el registro DDRA está en la dirección 0x01 cuando accedemos al registro con instrucciones del tipo IN/OUT, y en la dirección de memoria 0x21 cuando accedemos con instrucciones del tipo LD/ST.

PORTA – Port A Data Register								
Bit	7	6	5	4	3	2	1	0
0x02 (0x22)	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
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
DDRA – Port A Data Direction Register								
Bit	7	6	5	4	3	2	1	0
0x01 (0x21)	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
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
PINA – Port A Input Pins Address								
Bit	7	6	5	4	3	2	1	0
0x00 (0x20)	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Figura 7. Registros asociados al PA (puerto A): PORTA, DDRA y PINA

4.2.1.2 Funciones Arduino para el manejo de las entradas/salidas digitales

Además de manejar los puertos de entrada/salida digitales de propósito general directamente a través de los registros del interfaz, el entorno de programación Arduino proporciona otras funciones de más alto nivel que facilitan la programación desde el lenguaje de programación C/C++.

Las funciones básicas son:

// Programar un pin de entrada o de salida

pinMode(nº pin, INPUT / OUTPUT / INPUT_PULLUP)

```
pinMode(13, INPUT);      // pin 13 de entrada (INPUT=0)
```

```
pinMode(10, OUTPUT);     // pin 10 de salida (OUTPUT=1)
```

```
pinMode(6, INPUT_PULLUP); // pin 6 de entrada, rest. pull-up activa
```

// Poner un pin a "1" (HIGH) o a "0" (LOW)

digitalWrite(pin, HIGH / LOW)

```
digitalWrite(10, HIGH);   // escribe un "1"
```

```
digitalWrite(10, LOW);    // escribe un "0"
```

// Leer el estado de un pin

digitalRead(nº pin)

```
var1 = digitalRead(13); // lee pin 13 y deja el valor en var1
```

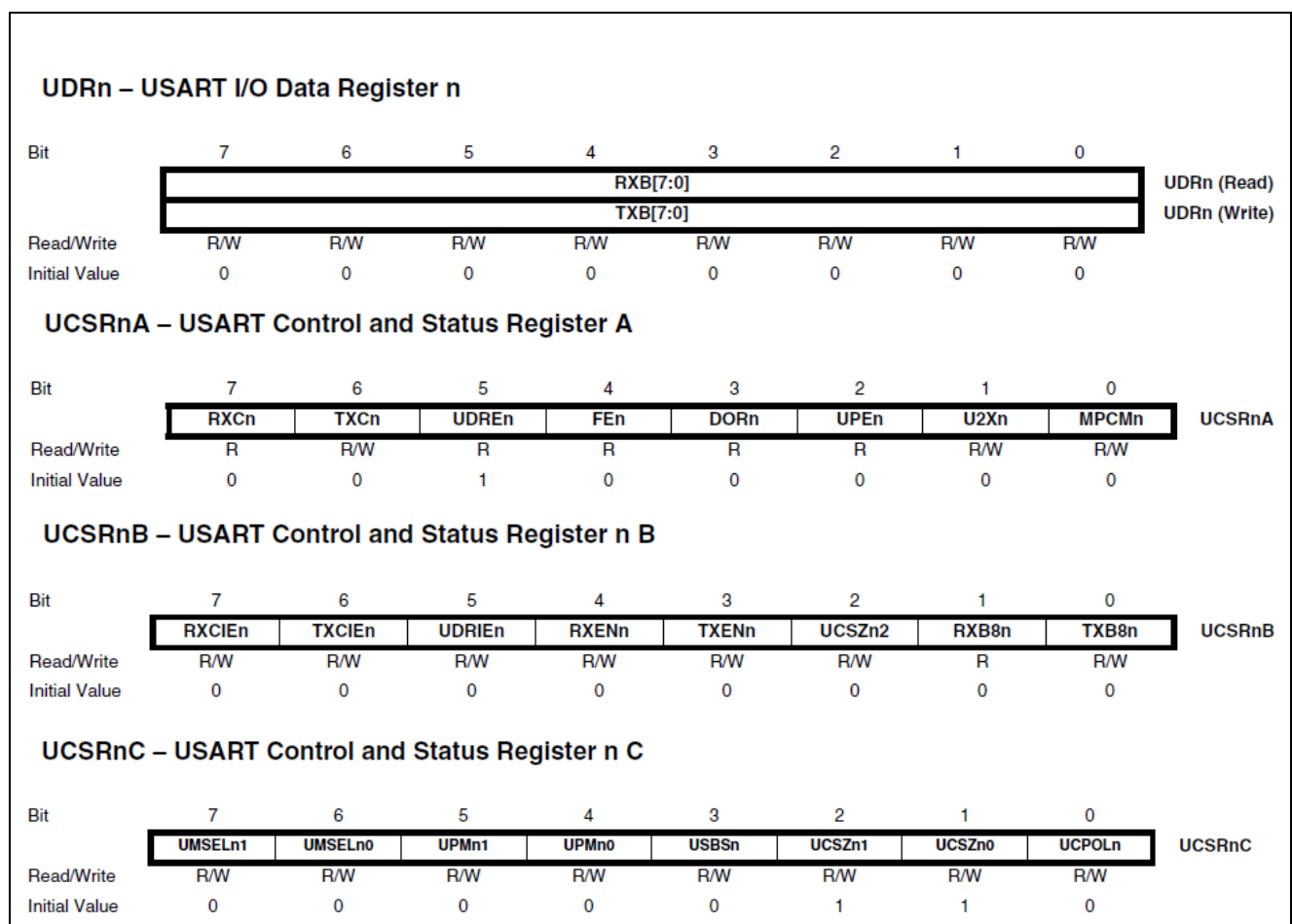
4.2.2 Puertos de comunicación serie

La tarjeta Arduino Mega 2560 dispone de 4 canales de comunicación serie: TX0/RX0, TX1/RX1, TX2/RX2 y TX3/RX3.

- TXn: Línea de transmisión del canal n (salida)
- RXn: Línea de recepción del canal n (entrada)

4.2.2.1 Interfaz de las comunicaciones serie

Cada pareja de líneas TX/RX(voltaje 0-5v) está asociada a una USART (Universal Synchronous Asynchronous Receiver and Transmitter) que es un interfaz, altamente flexible, para las comunicaciones serie dotado de múltiples registros a través de los que se configura y controla las comunicaciones serie del canal. La figura 8 muestra tales registros remitiendo al estudiante al manual del microcontrolador para un mayor detalle del funcionamiento.



UBRRnL and UBRRnH – USART Baud Rate Registers									
Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRR[11:8]				UBRRHn
	UBRR[7:0]								UBRRLn
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	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	
	0	0	0	0	0	0	0	0	

Figura 8. Registros asociados al interfaz de una USARTn

La comunicación serie consiste básicamente en el envío secuencial de los bits de un dato (de 7 u 8 bits, mayoritariamente) bit a bit, por una línea de transmisión a un ritmo constante, sincronamente con una señal reloj y de acuerdo a un protocolo. La velocidad de transmisión se mide en baudios o bits por segundo siendo frecuentes los valores de 2400, 4800, 9600, ... 115200 baudios. El protocolo de transmisión serie (ver figura 9) establece cómo se han de transmitir los datos por el canal físico (líneas de transmisión y recepción), definiendo la estructura de las palabras que se envían por el canal para garantizar la correcta recepción de los datos por el receptor, sin errores o pérdidas de datos. Para el envío de un dato, el protocolo establece la siguiente estructura o información a enviar:

- 1 START bit
- 5, 6, 7, 8, o 9 bits de datos
- 1 bit de paridad (si está habilitada)
- 1 o 2 STOP bits

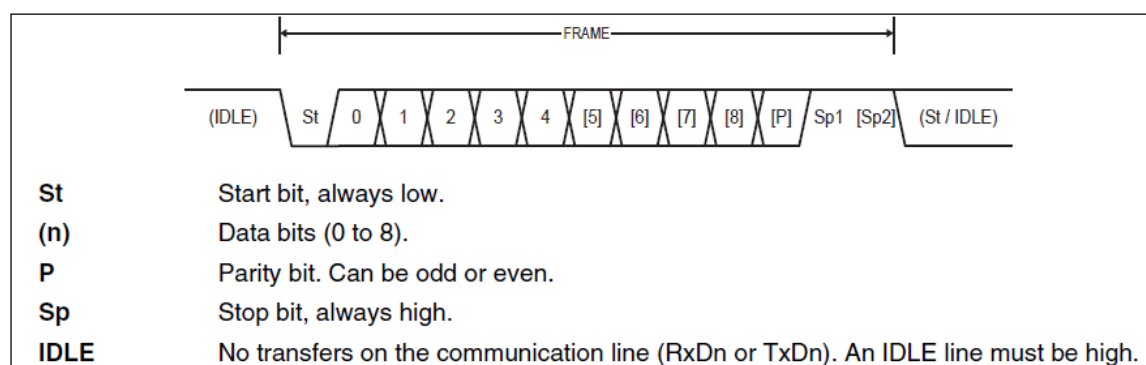


Figura 9. Transmisión serial de datos

La USART se encarga de realizar todas las operaciones de sincronización y envío de datos y avisa, a través de sus registros, de eventos diversos que puedan producirse durante la transmisión-recepción de los datos, teniendo también capacidades para la generación de interrupciones asociadas dichos eventos, tales como:

- Errores en la transmisión/recepción de datos
- Buffer de transmisión vacío
- Recepción de un dato y otros

4.2.2.2 Funciones Arduino para la comunicación serie

a) Configuración de un canal serie: parámetros

- Se especifica solo velocidad (resto por defecto: 8 bits, no parity, 1 stop bit)

`Serial.begin(speed);` //para TX0/RX0

`Serial1.begin(speed);` //para TX1/RX1

....

- Especificación de velocidad y parámetro config

`Serial.begin(speed,config);` // para TX0/RX0

`Serial3.begin(speed,config);` //para TX3/RX3

speed: 2400, 4800, 9600, ... 115200 (baudios o bits/seg)

config: SERIAL_5N1, SERIAL_8N2, SERIAL_8O2,.... (SERIAL_tamaño-dato|paridad|stop bits)

Donde:

Tamaño del dato: 5,6,7,8

Paridad: None (N), Even-par (E), Odd-impar (O)

Stop bits: 1,2

b) Transmisión de datos por el canal serie

`Serial.write()`

`Serial.print()`

`Serial.println()`

c) Recepción de datos por el canal serie

`Serial.available()`

`Serial.read()`

d) Deshabilitación del canal serie

`Serial.end()`

Para más información y uso de las funciones consultar la página web de Arduino.

4.2.3 Salidas de control de potencia (PWM)

Son salidas en las que se genera señales del tipo Pulse Width Modulation (PWM), es decir, una onda de frecuencia fija, pero de ciclo de trabajo (duty cycle) variable como se muestra en la figura10:

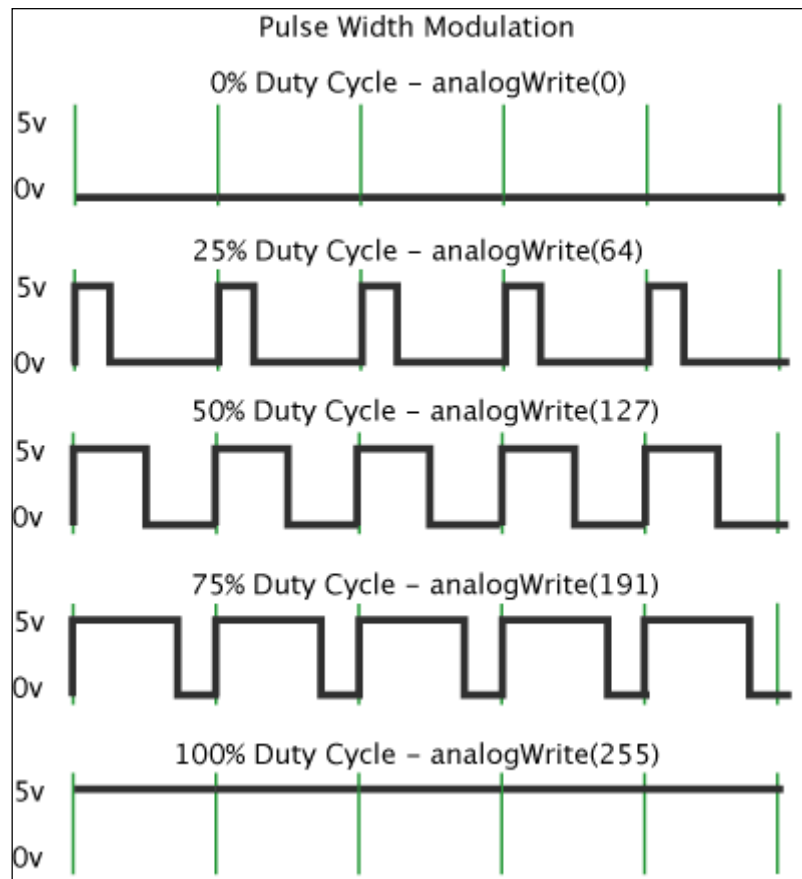


Figura 10. Señal de control de potencia PWM (Pulse Width Modulation)

El valor medio del voltaje que se aplica al dispositivo que esté conectado al pin en el que se genera una señal PWM oscila entre 0 y 5 voltios. Con esta señal se puede controlar el nivel de iluminación de un LED, velocidad de un motor, nivel de apertura y cierre de una válvula, temperatura y otras acciones.

Estas señales son generadas a través de los TIMERS disponibles en el microcontrolador. El ATmega 2560 dispone de un total de 6 timers (dos de 8 bits y cuatro de 16 bits) con los que se pueden generar diferentes tipos de señales digitales y/o establecer temporizaciones.

Los timers disponibles son:

Timer0: 8 bits (uso por las funciones: delay(), millis(), micros())

Timer1: 16 bits

Timer2: 8 bits (uso por la función tone())

Timer3-4-5: 16 bits

El uso de los timers es esencial en un uso avanzado del microcontrolador. Sin embargo, en esta práctica no profundizaremos más en ellos ya que serán objeto de un estudio más detallado en futuras prácticas. Por ahora, sólo comentaremos las funciones del entorno de programación Arduino que se utilizan para generar señales PWM.

4.2.3.1 Funciones Arduino para la generación de señales PWM

La función Arduino que nos permite la generación de señales PWM es:

```
analogWrite(pin, valor); // escritura analógica
```

Donde:

- Pin: nº de pin por donde sale la señal PWM generada por el timer. Posibles pines a utilizar:

Pines: 4-13 (timer 0)

Pines: 11-12-13 (timer 1)

Pines: 9-10 (timer 2)

Pines: 2-3-5 (timer 3)

Pines: 6-7-8 (timer 4)

Pines: 44-45-46 (timer5)

- valor: Número entre 0 (0% duty cycle) y 255 (100% duty cycle)

4.2.4 Entradas analógicas (A0-A15)

Las entradas analógicas permiten la lectura de valores analógicos en el rango de los 0 a 5 voltios mediante un proceso de digitalización basado en un conversor analógico-digital (ADC: Analog Digital Converter) de 10 bits de resolución. Ver figura11. De esta forma, 0v en una entrada analógica da lugar al valor digital 0000000000 y 5v al valor digital 1111111111, ambos representados internamente como enteros de 16 bits, con una resolución mínima de 4.88mv ($5v/1024 = 4.88mv$). Estas entradas permiten la conexión de dispositivos analógicos (ej. sensores) y, tras la conversión de los datos, tratar la información digitalmente con el correspondiente programa.

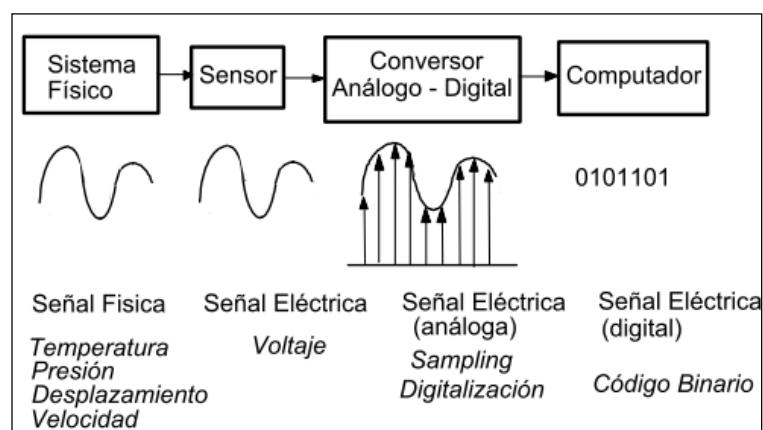


Figura 11. Conversión analógica-digital de una señal

Uso de las líneas analógicas como líneas de entrada y salida digitales

Después de un reset, los pines asociados a las entradas analógicas, A7-A0 (PORTF) y A15-A8 (PORTK), quedan programados como entradas analógicas. Si estos pines se quieren utilizar como líneas de entrada y salida digitales de propósito general será necesario redefinir su funcionalidad a través de un registro DDRx. Por ejemplo, para utilizar las entradas analógicas A7-A0 como líneas de entradas digitales tendríamos que hacer:

```
DDRF = B00000000; // pines asociados: 61(A7) ... 54(A0)
```

Nota: A las nuevas líneas de entrada digitales se les asigna un número correlativo a las digitales ya existentes (pines 22-53). Por tanto, el pin A0 sería la línea 54 y el pin A7 la línea 61 y podríamos leer esos pines con la función “digitalRead(54/61)”.

Para más información sobre la estructura interna del interfaz asociado a las entradas analógicas remitimos al estudiante al manual del microcontrolador.

4.2.4.1 Funciones Arduino para las entradas analógicas

Las funciones Arduino relacionadas con el manejo de las entradas analógicas son:

```
analogRead(pin); // pin: A0-A15
```

```
analogReference(type); // type: DEFAULT, INTERNAL1V1, INTERNAL2V56, EXTERNAL  
(pin AREF)
```

```
pinMode(pin, INPUT_PULLUP); //pin A0-A15
```

La función **analogRead(pin)** lee el valor analógico del pin y lo convierte a un valor digital de 10 bits (0-1023 en decimal).

La función **analogReference(type)** fija un voltaje de referencia para el conversor analógico digital según el valor máximo esperado para las señales a digitalizar y conectadas a las entradas analógicas. Para ello, hay varias opciones que son: 5v internos (valor por defecto), 1.1v internos, 2.56v internos y un voltaje externo (entre 0 y 5v) aplicado al pin AREF. Cuando se cambia este valor de referencia, las primeras lecturas que se hagan con analogRead() puede que no sean muy exactas.

La función **pinMode(pin, INPUT_PULLUP)** conecta una resistencia de pull-up en la entrada analógica indicada por el parámetro "pin" para que la entrada no quede al aire cuando no haya nada conectado.

Ejemplo:

```
analogReference(INTERNAL2V56); // se fija un voltaje de referencia interno de 2.56v  
pinMode(A0,INPUT_PULLUP); // conecta resistencia de pull-up  
int val = analogRead(A0); // val= 0-1023 (para entradas de 0 a 2.56 voltios)
```

4.2.5 Líneas de alimentación y reset

En la clasificación inicial de los pines del Arduino Mega 2560 tenemos un último grupo de pines asociados a la alimentación y reset. Estos pines son:

- **Vin**: Entrada de 5 voltios, voltaje regulado y estabilizado como alternativa al USB.
- **5V**: Salida de 5 voltios regulados (estables)
- **3V3**: Salida de 3.3 voltios a partir de los reguladores de tensión de la tarjeta
- **GND**: Tierra
- **IOREF**: Proporciona el voltaje de referencia con el que trabaja el microcontrolador
- **RESET**: Un "0" en este pin da lugar a un reset del microcontrolador.

Además de estos pines, existen otras conexiones en la tarjeta que son:

- **Puerto USB**: Alimentación y comunicaciones con el Computador Personal (PC)
- **Conector 7-12v**: Para alimentación externa de 7-12v mediante adaptador AC-DC

4.3 Sistema de interrupciones del microcontrolador ATmega 2560

El microcontrolador Atmega 2560 tiene un sistema de interrupciones autovectorizado (vectores fijos, es decir, vectores preasignados a las diversas fuentes de interrupción) con una tabla de interrupciones a partir de la posición 0 de memoria. Cada vector ocupa dos palabras (4 bytes) y contiene una instrucción de salto (jmp) a la rutina de servicio de la interrupción (ISR: Interrupt Service Routine). Las prioridades quedan establecidas según el número del vector siendo el vector 1 (reset) el más prioritario y el vector 57 (USART3 TX) el menos prioritario. A continuación, se muestra una parte de la tabla de vectores de interrupción y su contenido:

Vector No.	Program Address ⁽²⁾	Source	Interrupt Definition
1	\$0000 ⁽¹⁾	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	\$0002	INT0	External Interrupt Request 0
3	\$0004	INT1	External Interrupt Request 1
4	\$0006	INT2	External Interrupt Request 2
5	\$0008	INT3	External Interrupt Request 3
.....			
55	\$006C ⁽³⁾	USART3 RX	USART3 Rx Complete
56	\$006E ⁽³⁾	USART3 UDRE	USART3 Data Register Empty
57	\$0070 ⁽³⁾	USART3 TX	USART3 Tx Complete

Address	Code	Comments
0x0000	jmp RESET	RESET handler
0x0002	jmp INT0_vect	IRQ0 handler
0x0004	jmp INT1_vect	IRQ1 handler
0x0006	jmp INT2_vect	IRQ2 handler
0x0008	jmp INT3_vect	IRQ3 handler
...
0x0070	jmp USART3_TX_vect	USART3 TX Complete Handler

Figura 12. Tabla de vectores de interrupción

Las interrupciones pueden ser externas o internas. Las interrupciones externas tienen su origen en el exterior del microcontrolador y son producidas por dispositivos que cuando requieren atención del procesador interrumpen para que se les atienda de forma inmediata. Las interrupciones internas tienen su origen en los múltiples interfaces y dispositivos internos del microcontrolador y sus requerimientos son similares a las de las interrupciones externas.

Operaciones básicas en el uso de las interrupciones

Para gestionar una interrupción, interna o externa, se deben realizar las siguientes operaciones básicas:

1. Configurar y habilitar la interrupción para que pueda interrumpir. Esto se hace a través de los registros del interfaz que corresponda y del registro estado del procesador.
2. Definir la subrutina de servicio para la interrupción, ISR(intr_vect), y cargarla en memoria. Inicializar la entrada de la tabla de vectores de interrupción asociada al evento o vector de interrupción, intr_vect, para que dicha entrada contenga la instrucción:

```
jmp ISR(intr_vect)
```

Cada evento ya tiene asignado un vector intr_vect (vectores fijos) que sirve de índice para entrar en la tabla de vectores de interrupción.

Todo esto se realiza automáticamente, y de forma transparente para el usuario, al declarar la función Arduino ISR(intr_vect). Solo hemos de seleccionar el intr_vect que corresponda al evento que produce la interrupción.

El microcontrolador tiene múltiples interfaces para soportar los diferentes canales de comunicación, tanto digitales como analógicos, y todos ellos tienen capacidad para generar interrupciones cuando en su funcionamiento se producen determinados eventos.

En primer lugar, trataremos las interrupciones externas del tipo INTn y PCINTn y sus registros asociados a través de los que se configuran. Una vez configuradas y definidas las subrutinas de servicio de interrupción, el sistema de interrupciones queda listo para su funcionamiento.

4.3.1 Configuración y habilitación de interrupciones externas

La habilitación de una interrupción, normalmente, requiere de dos operaciones. La primera operación, de carácter global, afecta todas las interrupciones y se realiza a través de un bit del registro de estado del procesador. La segunda operación, de carácter más individual, está relacionada con la configuración y habilitación de cada una de las interrupciones en particular a través de los registros del interfaz de interrupciones que será objeto de descripción en los siguientes apartados.

El registro de estado del procesador tiene la siguiente estructura:

The AVR Status Register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
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	

Para las interrupciones, solo nos interesa el bit 7 de este registro. Este bit afecta a todas las interrupciones del microcontrolador, permitiéndolas o no. Para manipular este bit se dispone de las instrucciones:

SEI: Habilita interrupciones (I=1)

CLI: Deshabilita las interrupciones (I=0)

Cuando ocurre y se trata una interrupción este bit se pone a cero. Cuando se retorna de la subrutina de servicio de la interrupción este bit vuelve a ponerse a "1".

4.3.1.1 Interrupciones externas de tipo INTn

La siguiente tabla muestra los pines asociados a las interrupciones externas del tipo INTn.

Interrupción externa	Pin
INT0	21
INT1	20
INT2	19
INT3	18
INT4	2
INT5	3

Las interrupciones INTn son de carácter individual y se programan de forma independiente a través de los registros de su interfaz. Los dispositivos externos pueden hacer uso de estas interrupciones a través de los pines mencionados en la tabla. Los registros a los que hay que acceder para configurar y habilitar una interrupción externa, tipo INTn, son los siguientes:

EXTERNAL INTERRUPT CONTROL REGISTER A (EICRA)

EICRA – External Interrupt Control Register A

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

Bit (0x69)	7	6	5	4	3	2	1	0	
	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
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	
		INT3		INT2		INT1		INT0	

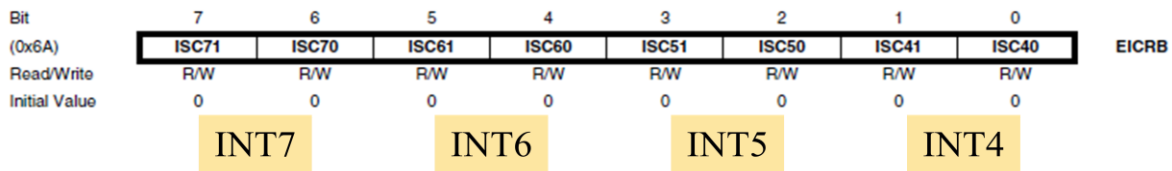
Permite configurar el funcionamiento de las interrupciones INT3-0 de acuerdo a la siguiente tabla:

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request
0	1	Any edge of INTn generates asynchronously an interrupt request
1	0	The falling edge of INTn generates asynchronously an interrupt request
1	1	The rising edge of INTn generates asynchronously an interrupt request

Note: 1. n = 3, 2, 1 or 0.
When changing the ISCn1/ISCn0 bits, the interrupt must be disabled by clearing its Interrupt Enable bit in the

EXTERNAL INTERRUPT CONTROL REGISTER B (EICRB)

EICRB – External Interrupt Control Register B



Permite configurar el funcionamiento de las interrupciones INT7-4 de acuerdo a la siguiente tabla:

Table 15-3. Interrupt Sense Control⁽¹⁾

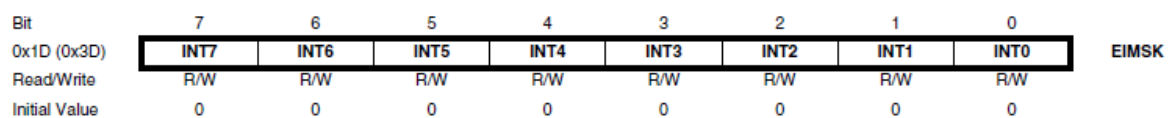
ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request
0	1	Any logical change on INTn generates an interrupt request
1	0	The falling edge between two samples of INTn generates an interrupt request
1	1	The rising edge between two samples of INTn generates an interrupt request

Note: 1. n = 7, 6, 5 or 4.

When changing the ISCn1/ISCn0 bits, the interrupt must be disabled by clearing its Interrupt Enable bit in the EIMSK Register. Otherwise an interrupt can occur when the bits are changed.

EXTERNAL INTERRUPT MASK REGISTER (EIMSK)

EIMSK – External Interrupt Mask Register



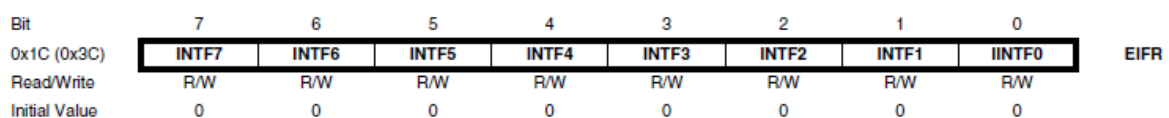
Con los bits de este registro se puede habilitar o deshabilitar una interrupción externa de tipo INTn de forma individual. A este registro se le conoce con el nombre de registro de máscaras y actúa de la siguiente forma:

Bit=1: Habilita la interrupción y puede interrumpir (desenmascara)

Bit=0: Deshabilita la interrupción y no puede interrumpir (enmascara)

EXTERNAL INTERRUPT FLAG REGISTER (EIFR)

EIFR – External Interrupt Flag Register



Este registro nos permite conocer las interrupciones que están pendientes de ser atendidas. Cuando se ejecuta la rutina de servicio de una interrupción, el bit correspondiente se pone a 0. También se puede resetear (poner a cero) si en él se escribe un "1".

4.3.1.2 Interrupciones externas de tipo PCINTn

Las interrupciones externas PCINTn (Pin Change Interrupt) son sensibles o se disparan cuando el pin cambia de estado. Tienen la particularidad de que ciertas operaciones de configuración y habilitación se hacen a nivel de grupo de interrupciones y no de forma individual.

La siguiente tabla muestra los nombres de las interrupciones y los pines asociados.

Interrupción externa	Pin
PCINT0-PCINT7	53,52,51,50,10,11,12,13
PCINT8-PCINT15	0,15,14,-,-,-,-
PCINT16-PCIN23	A8-A15

Note que ciertas interrupciones del tipo PCINTn, marcadas con el carácter "-", no tienen salida a los conectores de la tarjeta Arduino.

Los registros del interfaz para configurar y habilitar este tipo de interrupciones son:

PIN CHANGE INTERRUPT CONTROL REGISTER (PCICR)

PCICR – Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0	
(0x68)	-	-	-	-	-	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Habilita o deshabilita un grupo de interrupciones externas de la siguiente forma:

PCIE0: Un "1" habilita las interrupciones PCINT7-0. Un "0" las deshabilita.

PCIE1: Un "1" habilita las interrupciones PCINT15-8. Un "0" las deshabilita.

PCIE2: Un "1" habilita las interrupciones PCINT23-16. Un "0" las deshabilita.

PIN CHANGE INTERRUPT FLAG REGISTER (PCIFR)

PCIFR – Pin Change Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x1B (0x3B)	-	-	-	-	-	PCIF2	PCIF1	PCIF0	PCIFR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Este registro nos permite conocer los grupos de interrupciones que están pendientes de ser atendidos (pero no qué interrupción dentro del grupo). Cuando se ejecuta la rutina de servicio de una interrupción, el bit correspondiente se pone a "0". También se puede resetear (poner a cero) si en él se escribe un "1".

PIN CHANGE MASK REGISTER 0 (PCMSK0)

PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
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	

Registro de máscaras para las interrupciones PCINT7-0.

Bit=1: Habilita la interrupción y puede interrumpir (desenmascara)

Bit=0: Deshabilita la interrupción y no puede interrumpir (enmascara)

PIN CHANGE MASK REGISTER 1 (PCMSK1)

PCMSK1 – Pin Change Mask Register 1

Bit	7	6	5	4	3	2	1	0	
(0x6C)	PCINT15	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
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	

Registro de máscaras para las interrupciones PCINT15-8.

Bit=1: Habilita la interrupción y puede interrumpir (desenmascara)

Bit=0: Deshabilita la interrupción y no puede interrumpir (enmascara)

PIN CHANGE MASK REGISTER 2 (PCMSK2)

PCMSK2 – Pin Change Mask Register 2

Bit	7	6	5	4	3	2	1	0	
(0x6D)	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
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	

Registro de máscaras para las interrupciones PCINT23-16.

Bit=1: Habilita la interrupción y puede interrumpir (desenmascara)

Bit=0: Deshabilita la interrupción y no puede interrumpir (enmascara)

4.3.2 Subrutina de servicio de una interrupción (ISR)

La idea básica es declarar una función `ISR(intr_vect)` que automáticamente se carga en memoria como una subrutina de atendimento para la interrupción con vector "intr_vect".

Cuando se produce una interrupción cuyo vector de interrupción es "intr_vect", automáticamente se ejecutará la función `ISR(intr_vect)` que atenderá al dispositivo que interrumpió. El entorno de programación de Arduino facilita la definición de estas subrutinas de servicio utilizando una función con el nombre `ISR()` con la sintaxis que se muestra a continuación:

```
ISR(intr_vect) {
    // Cuerpo de la subrutina ISR()
    // Se ejecutará cuando se produzca una interrupción
    // con el n° de vector "intr_vect"
}
```

El parámetro "intr_vect" es el vector de interrupción que cada fuente de interrupción tiene ya fijado (interrupciones autovectorizadas) y al que se hace referencia de forma muy cómoda a través de un nombre o etiqueta. La siguiente tabla muestra algunos de estos nombres de vectores (del 0 al 31) que se utilizan como índice para entrar en la tabla de vectores de interrupción desde donde se salta a la subrutina de servicio de interrupción o función `ISR()` del vector.

RESET	INT7_vect	TIMER1_CAPT_vect	SPI_STC_vect
INT0_vect	PCINT0_vect	TIMER1_COMPA_vect	USART0_RX_vect
INT1_vect	PCINT1_vect	TIMER1_COMPB_vect	USART0_UDRE_vect
INT2_vect	PCINT2_vect	TIMER1_COMPC_vect	USART0_TX_vect
INT3_vect	WDT_vect	TIMER1_OVF_vect	ANALOG_COMP_vect
INT4_vect	TIMER2_COMPA_vect	TIMER0_COMPA_vect	ADC_vect
INT5_vect	TIMER2_COMPB_vect	TIMER0_COMPB_vect	EE_READY_vect
INT6_vect	TIMER2_OVF_vect	TIMER0_OVF_vect	TIMER3_CAPT_vect

Para una lista completa de vectores de interrupción y sus nombres consultar el fichero:

C:\Program Files (x86)\Arduino\hardware\tools\avr\avr\include\avr\iomxx0_1.h

Veamos ahora algunos ejemplos.

Ejemplo 1: Definición de la función ISR de la interrupción externa INT3 (pin 20)

```
ISR(INT3_vect){  
    // Cuerpo de la subrutina de servicio de interrupción a ejecutar cuando  
    // se interrumpa por el pin 20 (INT3).  
}
```

Ejemplo 2: Definición de la función ISR de la interrupción interna del TIMER1_COMPA

```
ISR(TIMER1_COMPA_vect){  
    // Cuerpo de la subrutina de servicio de interrupción a ejecutar cuando  
    // interrumpa el TIMER1_COMPA.  
}
```

Una vez visto cómo se define una función ISR(), es necesario afrontar la programación de los interfaces para que puedan generar interrupciones. Como son varios los interfaces disponibles en el microcontrolador, en esta práctica solo abordaremos el estudio del que corresponde a las interrupciones externas y de forma más superficial, el interfaz del TIMER1, solo en la programación necesaria para que genere una interrupción cada cierto tiempo.

4.3.3 Funciones Arduino para la gestión de interrupciones

El entorno de programación de Arduino ofrece un conjunto de funciones que permiten al usuario hacer uso de las interrupciones externas de una manera cómoda y sencilla, sin tener que conocer los detalles de bajo nivel del microcontrolador Atmega 2560.

Las funciones básicas Arduino para el manejo de interrupciones externas son:

Deshabilitar interrupciones

```
noInterrupts()  
  
cli()
```

Conectar/asociar una interrupción externa de tipo INTn a un pin

```
attachInterrupt(digitalPinToInterrupt(pin), miRutina, Mode)
```

pin: nº del pin por donde se interrumpe 21-20-19-18-2-3 (INT0-1-2-3-4-5)

miRutina: Función a ejecutar para atender a la interrupción

Mode: Modo de disparo de la interrupción: LOW, CHANGE, RISING, FALLING

Habilitar interrupciones

```
interrupts()  
sei()
```

Declarar una función ISR() para una interrupción

```
ISR(intr_vect) {....}
```

Desconectar una interrupción de un pin

```
detachInterrupt(digitalPinToInterrupt(pin))
```

Como se intuye de las anteriores funciones, para las interrupciones externas de tipo INTn (INT0-1-2-3-4-5) el entorno Arduino proporciona todo lo necesario para que el usuario, de forma muy cómoda, configure, habilite y declare la subrutina de servicio que se ha de ejecutar cuando por un pin externo, con capacidad de interrumpir, se interrumpa. El usuario apenas necesita conocer detalles internos del microcontrolador para utilizar este tipo de interrupciones.

Sin embargo, el manejo del resto de tipos de interrupciones requiere de una mayor implicación del programador, en el sentido que, ha de conocer la estructura interna de los interfaces, sus registros y funcionalidades para programar adecuadamente las operaciones de configuración y habilitación de interrupciones. Se puede decir que el programador ha de hacer una programación directa, de bajo nivel, accediendo al hardware de los interfaces a nivel de bits. No es necesario comentar que las interrupciones de tipo INTn se pueden gestionar de esta forma, a bajo nivel, sin hacer uso de las funciones de alto nivel de Arduino.

5 Tarjeta de expansión

5.1 Componentes básicos

Con objeto de poder facilitar y desarrollar aplicaciones con dispositivos periféricos de uso frecuente se ha diseñado y construido una tarjeta de expansión (ver figura 13) que integra varios periféricos de entrada y salida. Las señales de control de estos periféricos terminan en unos conectores que luego mediante un cableado se pueden conectar fácilmente a los pines de la tarjeta Arduino Mega 2560. La tarjeta dispone de los siguientes elementos:

- 1) Pulsadores
- 2) Visualizador de 4 dígitos 7-segmentos
- 3) Zumbador
- 4) Teclado matricial de 12 teclas (4 filas x 3 columnas)
- 5) Memoria EEPROM (I2C)
- 6) Pantalla LCD (serial/I2C)
- 7) Pantalla OLED (I2C)
- 8) Otros componentes: resistencias y transistores

En la realización de esta práctica sólo será necesario el uso de los pulsadores, el visualizador de 7 segmentos, el teclado y el zumbador, que serán objeto de descripción en los siguientes apartados, dejando la descripción del resto de componentes para futuras prácticas.

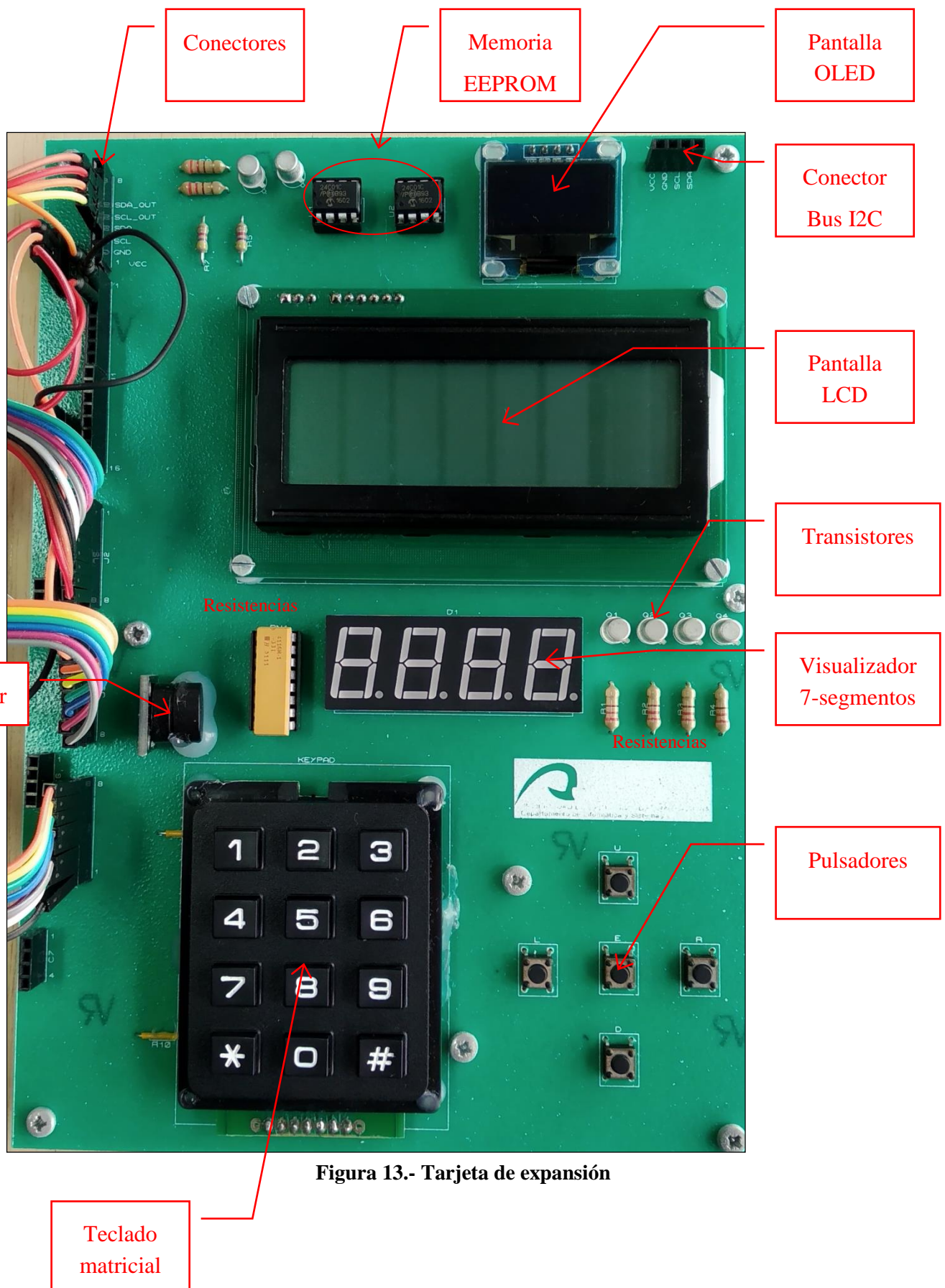


Figura 13.- Tarjeta de expansión

5.1.1 Visualizador de 7 segmentos

La tarjeta de expansión tiene un visualizador de 7-segmentos de 4 dígitos. La figura 14 muestra la estructura interna de un visualizador de 7-segmentos (HP 5082-7740) formado por los segmentos a, b, c, d, e, f y g y un punto (dot), conectados en una configuración de ánodo común o cátodo común. El utilizado en esta práctica será de cátodo común por lo que para encender los segmentos será necesario aplicar una tensión positiva a la entrada del segmento y conectar a tierra (0 voltios) la pata o pin de cátodo común. En las hojas de características del visualizador podrá consultar todos los detalles relativos a las tensiones y corrientes necesarias para el correcto funcionamiento. Según dichas hojas, será suficiente una corriente de $I_F = 20 \text{ mA}$ para que un segmento brille fuertemente. En nuestro caso y para no cargar demasiado los puertos del Arduino, limitaremos esta corriente a 8-10 mA que será suficiente para una visualización satisfactoria.

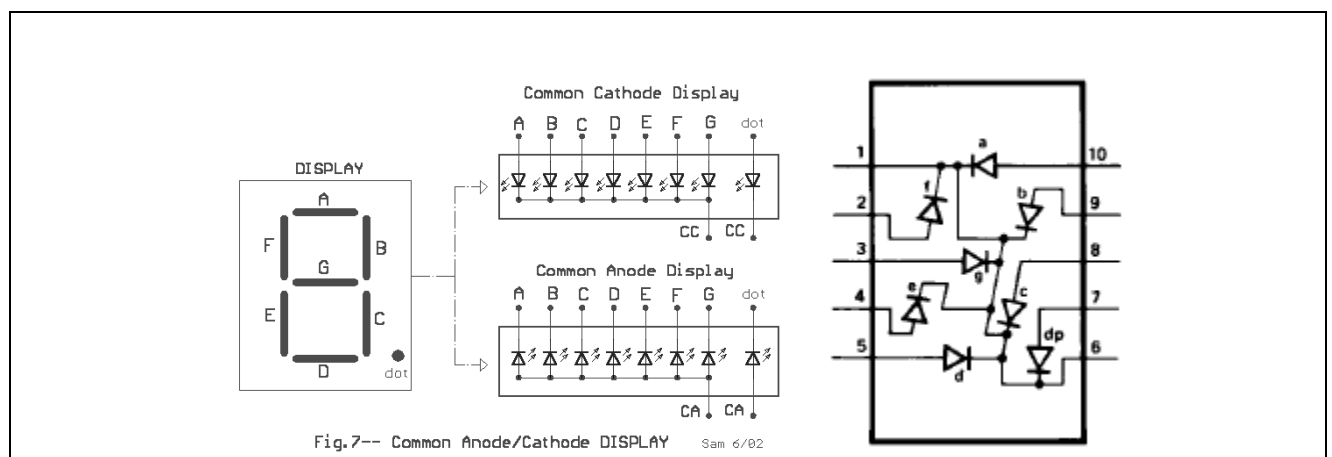


Figura 14. a) Visualizador de 7 segmentos; b) Pines del HP 5082-7740)

También existen en el mercado visualizadores de 4 dígitos, con un conexionado interno preestablecido, que simplifica enormemente el diseño cuando se necesita un visualizador de varios dígitos siendo ésta la solución adoptada en el diseño de la tarjeta de expansión. En la figura 15 se muestra el visualizador de cátodo común modelo SMA420564, de 4 dígitos (Digit1, Digit2, Digit3 y Digit4) donde se aprecia los pines asociados a los segmentos de los dígitos (A,B,C,D,E,F,G,DP) y los pines de los cátodos comunes de los dígitos (1,2,3,4).

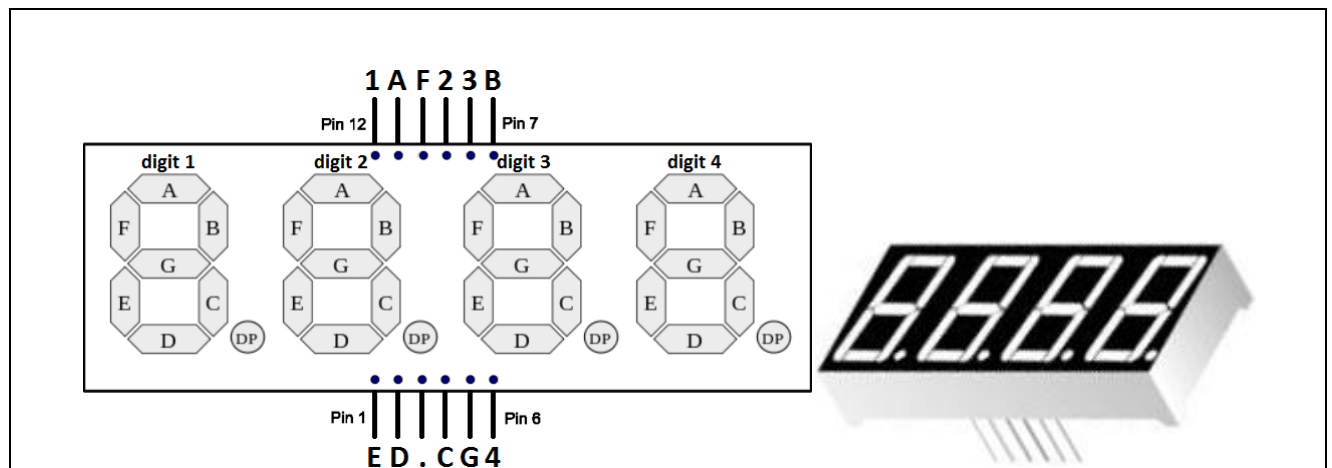


Figura 15. Visualizador de 4 dígitos SMA420564

5.1.2 Teclado matricial

En la versión más sencilla, un teclado matricial no es más que un conjunto de contactos organizados en filas columnas (forma matricial), en los que cada uno tiene asociado un pulsador (tecla) que dependiendo de su estado (pulsado o no pulsado) establece el contacto entre una fila y una columna del teclado.

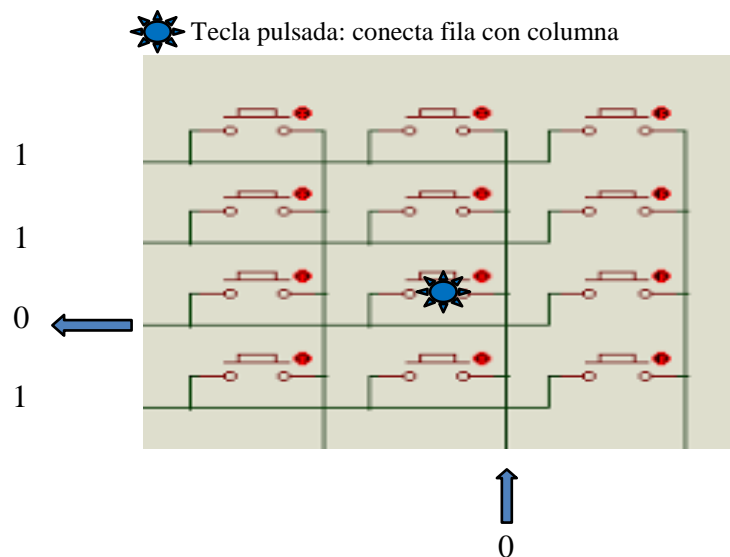


Figura 16. Teclado matricial

Para detectar las pulsaciones de las teclas, es necesario realizar un proceso de exploración continuo del teclado consistente en colocar un “0” en una columna y luego leer el estado de las filas tal y como se ilustra en la figura 16. Este proceso se repite para todas las columnas del teclado, de forma continua y con un tiempo mínimo entre exploración del orden de 5 a 10 ms con objeto de no perder pulsaciones de teclas.

5.1.3 Pulsador

Los pulsadores son elementos muy sencillos que cuando se pulsan establecen una conexión entre dos terminales. Se comercializan en distintas calidades definidas por el número de pulsaciones que son capaces de soportar sin fallos de conexión.

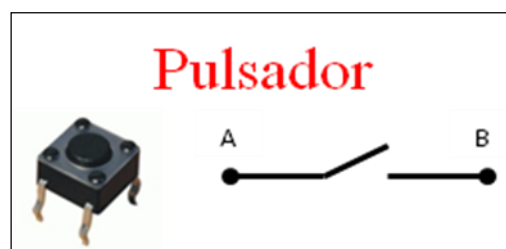


Figura 17. Pulsador

5.1.4 Zumbador y/o altavoz

Como avisador acústico se hará uso de un altavoz o zumbador. Este dispositivo transductor convierte la energía eléctrica en acústica a través de una membrana o elemento similar. Puede generar sonidos de diferentes frecuencias según la frecuencia de la señal eléctrica aplicada. Ver figura 18.

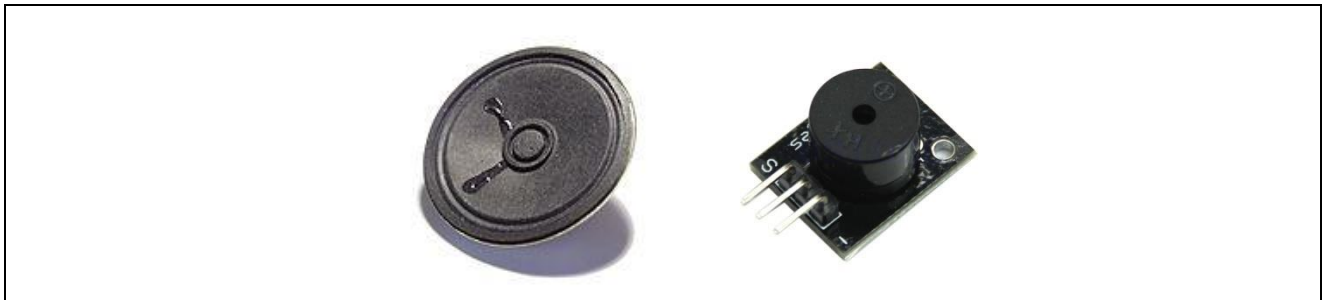


Figura 18. Altavoz/zumbador

5.1.5 Otros componentes

Además de los componentes anteriormente citados se utilizan otros que son necesarios para el funcionamiento conjunto del sistema. Básicamente, nos estamos refiriendo a resistencias discretas o en formato array (DIL: Dual In Line), transistores PNP como drivers de los visualizadores de 7 segmentos, conectores varios y cables de conexionado. Las resistencias se utilizan como limitadores de corriente mientras que los transistores, en este caso, se utilizan como interruptores que dejan pasar o no, corrientes de intensidades de hasta 500 mA.

La línea de selección de cada uno de los dígitos del visualizador (cátodo común) no debe conectarse directamente a un pin del Arduino ya que la corriente del cátodo común puede alcanzar los 96-120 mA, suponiendo que todos los segmentos estén encendidos y 12-15 mA por segmento. Estos valores de intensidad son excesivos para un pin del Arduino que según las especificaciones puede trabajar con valores típicos de hasta 20 mA. A partir de los 40 mA se corre el riesgo de quemar el pin o línea. Es aquí donde utilizamos un transistor que actúa como un interruptor controlado por una señal de mando generada por el Arduino. Así podremos controlar el paso de corrientes de colector (I_c) de 100-200 mA a partir de una señal de mando del orden de 10-20 mA aplicada a la base (I_b). Ver figura 19.

La corriente de base es la que tendría que manejar (dar o recibir) el pin del Arduino que como puede observarse es 10 veces más pequeña que la corriente del terminal de colector (proveniente del cátodo común del visualizador 7 segmentos).

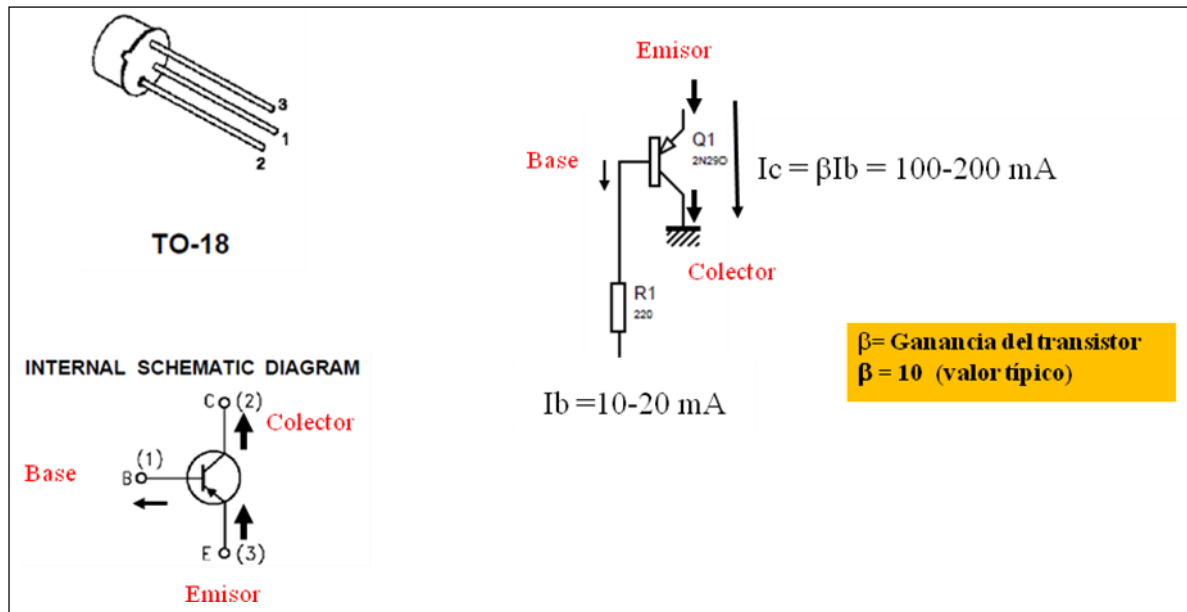


Figura 19. Transistor PNP 2N2907

5.2 Diagrama de conexiones de la tarjeta expansión

Una vez descritos los componentes básicos a utilizar en esta práctica, vamos a continuar con el esquema de conexionado de todos los componentes que es mostrado en la figura 20. Como se aprecia, todas las señales de datos y control de los periféricos son llevadas a conectores terminales convenientemente etiquetados.

La figura 21 muestra el mapeo (correlación) de señales realizado entre los conectores de la tarjeta de expansión (donde terminan todas las señales de los periféricos de la tabla) y los conectores del Arduino Mega 2560. Este mapeo ya se encuentra implementado a través de cables de conexión por lo que, salvo excepciones, el estudiante únicamente debe centrar su atención en conocer qué controla cada pin del Arduino (de acuerdo al mapeo de señales establecido) y el desarrollo de software que la aplicación a desarrollar demande.

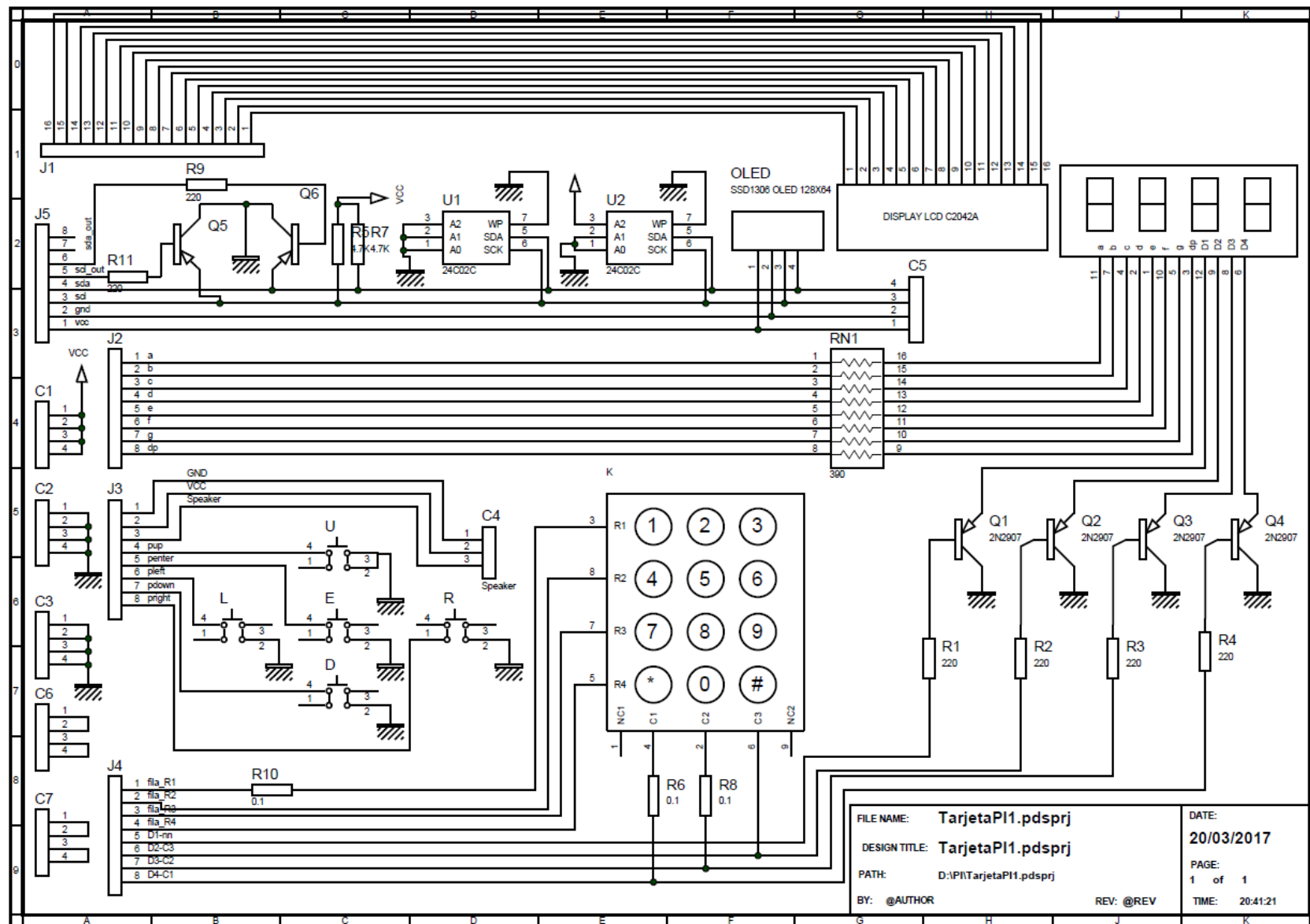


Figura 20. Diagrama de conexiones en la tarjeta prototipo

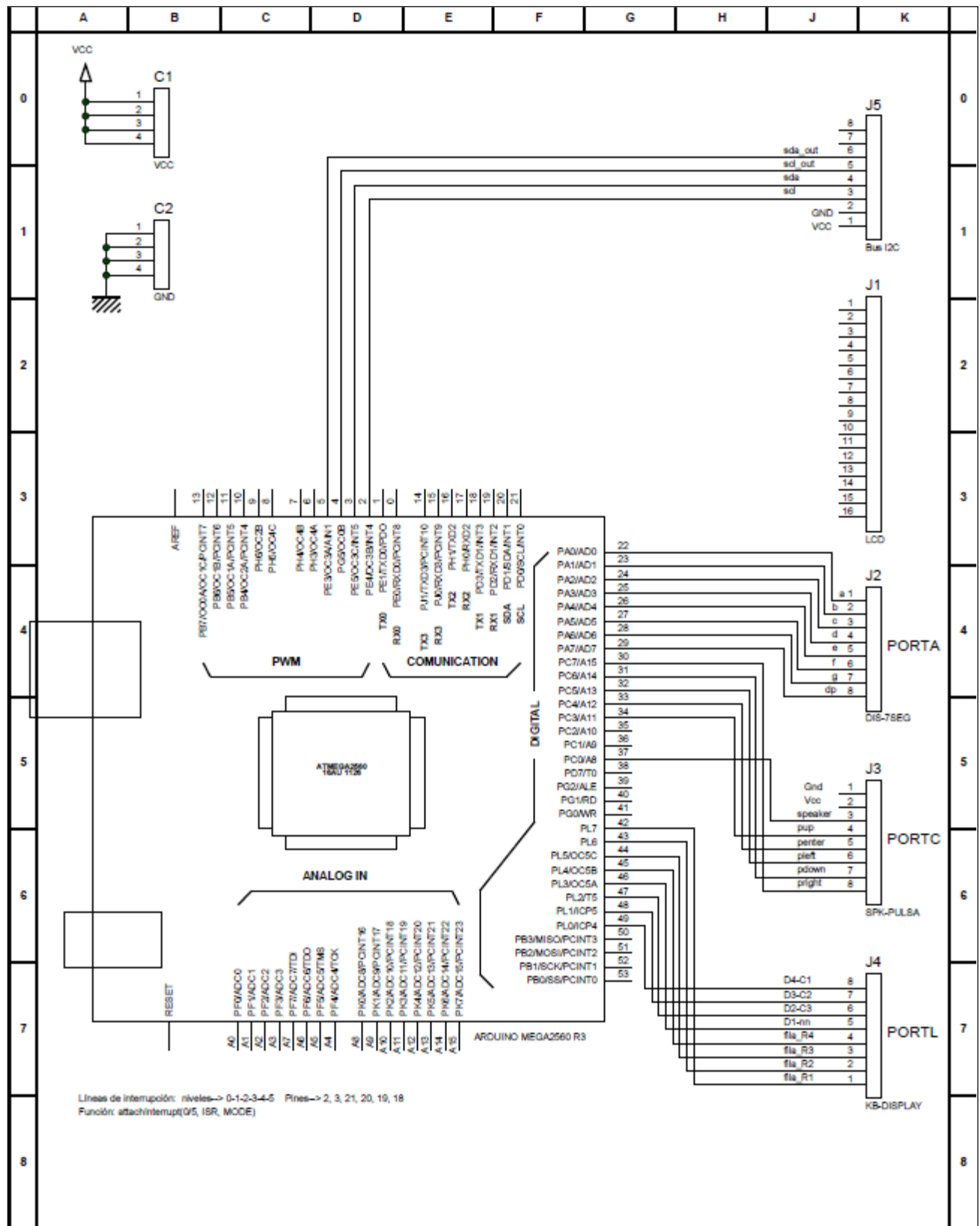


Figura 21. Interconexiones entre el Arduino Mega 2560 y la tarjeta de expansión

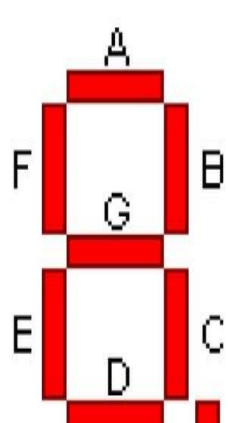
5.3 Detalles del conexionado de los periféricos

Solo se detallan los periféricos a utilizar en esta práctica. El resto de las conexiones pertenecientes a otros periféricos se deja para futuras prácticas.

5.3.1 Visualizador 7-segmentos

El control del visualizador de 7 segmentos de 4 dígitos requiere de dos grupos de señales: 1) Información a visualizar en los segmentos a-b-c-d-e-f-g-dp y 2) señales de control para los cátodos comunes del visualizador.

El primer grupo de señales lo proporciona el PORTA del Arduino, pines 22-29, de acuerdo a la información suministrada en la tabla de la figura 22 que muestra la correspondencia entre los segmentos del visualizador y los pines del PORTA. La tabla muestra los valores a enviar al puerto para visualizar los dígitos decimales 0, 1 y 2. El resto de la tabla se deja como ejercicio para el estudiante.



	Puerto PA[7:0] del Arduino								Valor
bit → pin→	PA7 29	PA 28	PA5 27	PA4 26	PA3 25	PA2 24	PA1 23	PA0 22	
Seg.→	<i>dot</i>	<i>g</i>	<i>f</i>	<i>e</i>	<i>d</i>	<i>c</i>	<i>b</i>	<i>a</i>	
0	-	0	1	1	1	1	1	1	0x3F=63
1	-	0	0	0	0	1	1	0	0x06=06
2	-	1	0	1	1	0	1	1	0x5B=91
3									
4									
5									
6									
7									
8									
9									

Figura 22. Conexiones de los segmentos del visualizador

En segundo grupo de señales, asociadas al control del cátodo común de cada uno de los dígitos del visualizador, es proporcionado por el PORTL del Arduino de acuerdo a la información mostrada en la figura 23. El cátodo común del dígito más a la derecha (unidades) está conectado al emisor de un transistor y se gobierna con la señal D4-C1(pin 49, PL0). Cuando dicha señal está a cero, el transistor se satura o conduce conectando el cátodo común del dígito D4 a tierra. Todos aquellos segmentos que tenga un “1” (o 5 voltios) a la entrada, se encenderán. De igual forma, se han

conectado el resto de los dígitos con las señales: D3-C2 (pin 48, PL1), D2-C3 (pin 47, PL2) y D1-Cn (pin 46, PL3) correspondientes al control de las decenas, centenas y unidades de millar, respectivamente. La tabla de la figura 23 resume las conexiones comentadas anteriormente a los pines PL0, PL1, PL2 y PL3. También se muestran otras señales relacionadas con el control del teclado, que se explicarán en el siguiente apartado.

PORTL			
Nº pin	Puerto	Nombre señal	Función
42	PL7-in	fila_R1	Leer fila R1 del teclado
43	PL6-in	fila_R2	Leer fila R2 del teclado
44	PL5-in	fila_R3	Leer fila R3 del teclado
45	PL4-in	fila_R4	Leer fila R4 del teclado
46	PL3-out	D1-Cnn	activar D1(mill.)
47	PL2-out	D2-C3	activar D2 (cent.) + col. 3 teclado
48	PL1-out	D3-C2	activar D3 (dec.) + col. 2 teclado
49	PL0-out	D4-C1	activar D4 (uds.) + col. 1 teclado

Figura 23. Conexiones de las señales de control D1-D2-D3-D4

5.3.2 Teclado matricial 4x3

El teclado matricial, de 4 filas por 3 columnas, será explorado sincronamente con el display de 7-segmentos. Cada vez se seleccione la activación de un dígito (poniendo un "0" en su cátodo común) también se explorará una columna del teclado de acuerdo a las conexiones realizadas. Para detectar la pulsación de cualquiera de las teclas de esa columna, bastará leer las filas del teclado (R1, R2, R3 y R4) y detectar si hay un cero en alguno de las filas. Cuando no se pulsa tecla alguna las filas estarán a un "1" lógico.

Las columnas C1, C2 y C3 están conectadas a los pines PL0, PL1 y PL2, respectivamente. Las filas R1, R2, R3 y R4 están conectadas a los pines PL7, PL6, PL5 y PL4, respectivamente. Es importante programar las entradas PL7-PL4 (pines 42-45) como entradas digitales y conectar las resistencias de pull-up internas, para que las líneas no queden al aire cuando no haya pulsaciones y estén a un "1" lógico.

En la siguiente figura se muestra cómo se detecta la tecla pulsada dependiendo de la columna (C1/PL0-C2/PL1-C3/PL3) que se explore y de la fila (R1,R2,R3,R4) en la que se detecte el "0" lógico.

	Exploración de Columna	R1 PL7	R2 PL6	R3 PL5	R4 PL4	Tecla Pulsada
		0	1	1	1	
D4-C1 = 0		1	0	1	1	4
		1	1	0	1	7
		1	1	1	0	*
		0	1	1	1	2
D3-C2 = 0		1	0	1	1	5
		1	1	0	1	8
		1	1	1	0	0
		0	1	1	1	3
D2-C3 = 0		1	0	1	1	6
		1	1	0	1	9
		1	1	1	0	#

Figura 24. Barrido del teclado y del display de 7 segmentos.

5.3.3 Pulsadores

Los pulsadores son conectados a los pines PC[7:3] del Arduino Mega 2560 de modo que cuando se pulsa un pulsador el pin se pone a "0" y cuando no está pulsado el pin queda a "1". **Será necesario activar las resistencias de pull-up internas del Arduino para definir el estado HIGH de un pulsador cuando éste no esté pulsado.** En la figura 25, se muestra la conexión de los pulsadores a los pines del PORTC del Arduino.

PORTC			
Nº pin	Puerto	Nombre señal	Función
30	PC7-in	pright	Derecha
31	PC6-in	pdown	Abajo
32	PC5-in	pleft	Izquierda
33	PC4-in	penter	Enter
34	PC3-in	pup	Arriba
35	PC2	-	
36	PC1	-	
37	PC0-out	speaker	Altavoz

Figura 25. Conexiones de los pulsadores

Para una correcta detección de las señales de los pulsadores es necesario conectar las resistencias de pull-up de los pines que correspondan para lo que se pueden utilizar las siguientes líneas de código en el apartado de setup() de nuestro programa:

```
DDRC = B00000001; // Definimos el PORTC de entrada salvo PC0 (salida)
PORTC= B11111000; // activar pull-up interno de los pines de entrada PC7-PC3
```

5.3.4 Zumbador/altavoz

El zumbador o altavoz, según de los que se disponga, permite generar una señal acústica de una frecuencia de terminada. Para ello, haremos uso de la función `tone()`:

```
tone(pin, frecuencia, duración en ms)
```

Inicialmente el zumbador de la tarjeta de expansión está ya cableado para hacer uso del pin 37 (PC0).

6 Entorno de programación Arduino

El software para implementar las diferentes actividades prácticas se desarrollará bajo el entorno de programación de Arduino haciendo uso de las utilidades que nos ofrece su lenguaje de programación para controlar los diversos aspectos del hardware. Cada programa, llamado "sketch" en el entorno de programación de Arduino, requiere de un directorio con el mismo nombre y consta de las siguientes secciones:

1.- Declaración de variables

2.- Función setup()

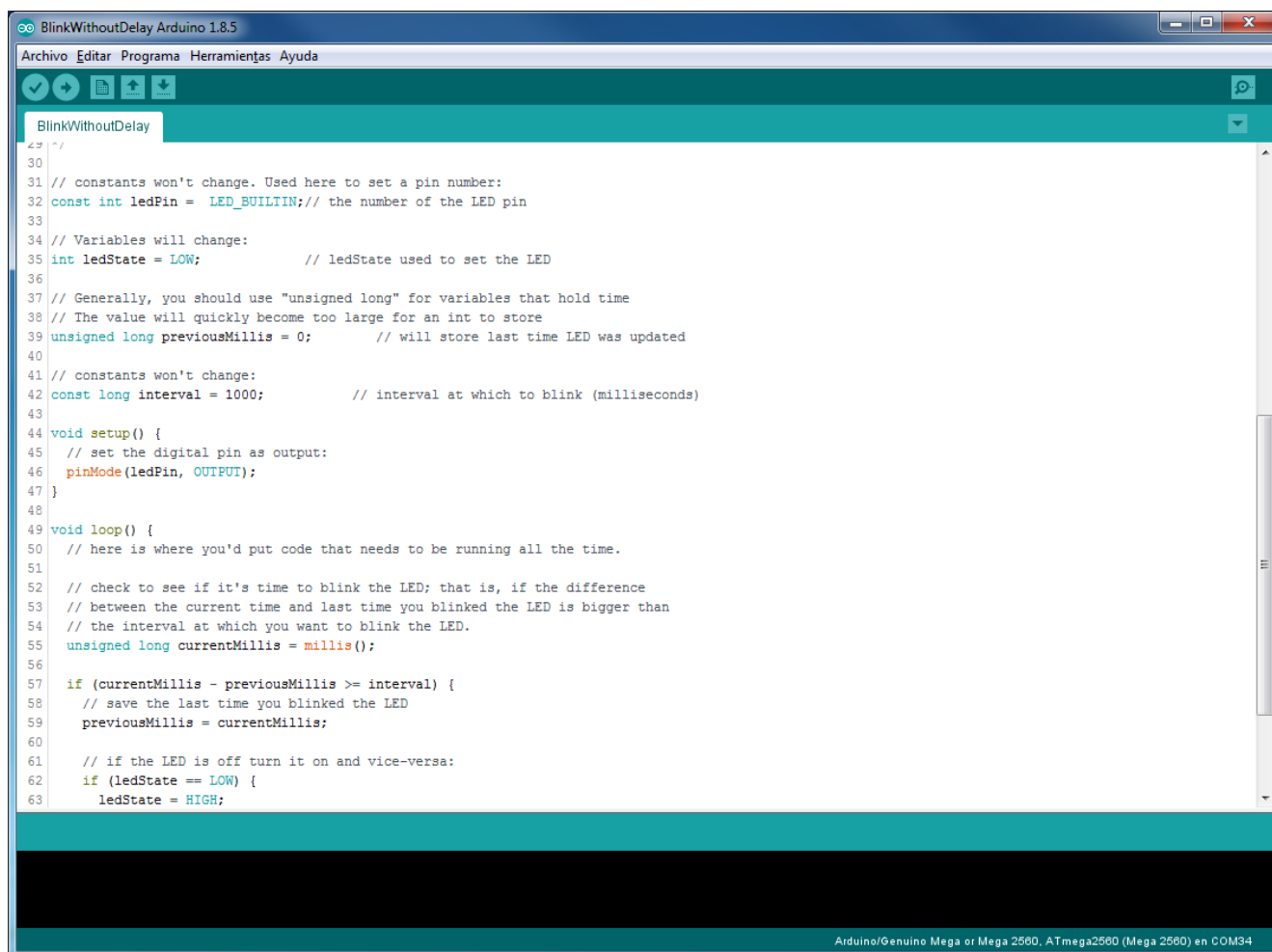
- void setup() {...}: Código de inicialización (se ejecuta sólo una vez)

3.- Opcional: otras funciones de la aplicación, librerías, ...

3.- Función loop()

- void loop() {...}: Código de ejecución indefinida

La figura muestra el entorno de programación con un programa en edición.



Ejemplo de programa:

```
/*
   Blink: Turns on an LED on for one second, then off for one second, repeatedly.
*/

// Declaración de variables
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// Inicialización: setup()
// the setup routine runs once when you press reset:
void setup() {
    // initialize the digital pin as an output.
    pinMode(led, OUTPUT);
}

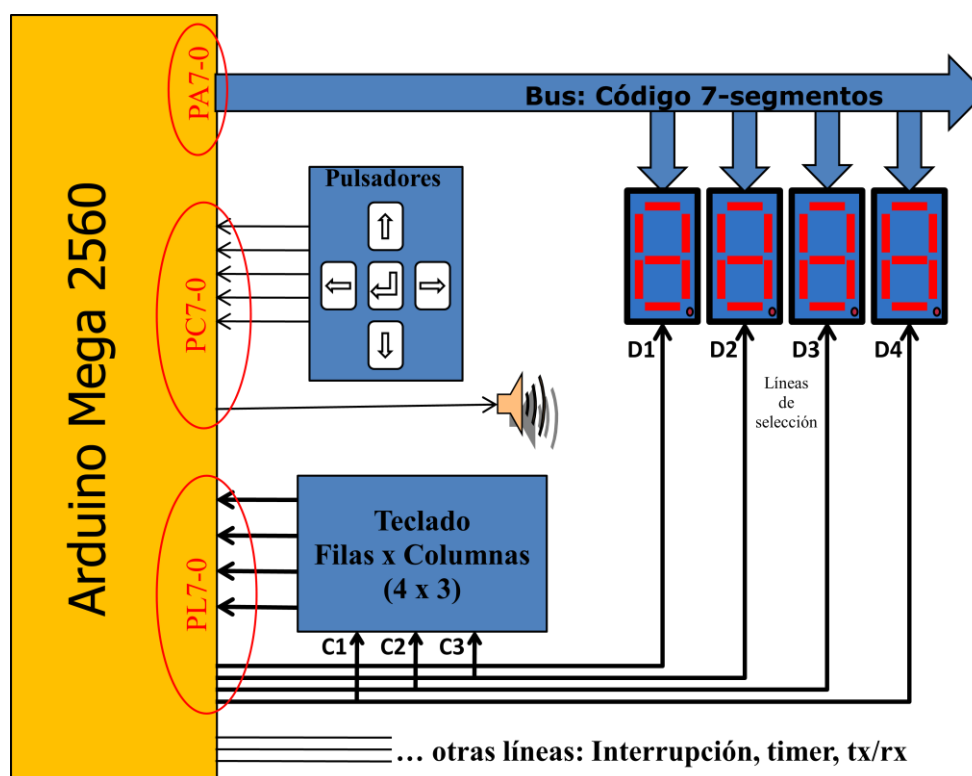
// otras funciones
void function_1() { ... }
void function_2() { ... }
ISR(vector de interrupción){.....}

// Bucle indefinido: loop()
// the loop routine runs over and over again forever:
void loop() {
    digitalWrite(led, HIGH);    // turn the LED on (HIGH is the voltage level)
    delay(1000);                // wait for a second
    digitalWrite(led, LOW);     // turn the LED off by making the voltage LOW
    delay(1000);                // wait for a second

    // posibles llamadas a funciones
    function_1();
    function_2();
    .....
}
```

7 Realización práctica

Para familiarizarse con el sistema de entrada y salida y con el entorno de programación de Arduino, plantearemos un conjunto de actividades prácticas, de complejidad creciente, de modo que una vez realizadas, se adquiera una cierta habilidad y soltura para el desarrollo de aplicaciones y/o interfaces paralelos que requieran los periféricos utilizados. El estudiante tiene completa libertad para realizar cualquier otra actividad complementaria, o de mejorar las actividades básicas planteadas, con la posibilidad de incrementar la calificación de la práctica hasta en 1 punto. El siguiente esquema muestra un diagrama de bloques de los diferentes componentes (display de 4 dígitos de 7-segmentos, pulsadores y teclado de 4x3 teclas) a utilizar en el desarrollo de la práctica y las conexiones a los puertos del microcontrolador. **La práctica se desarrollará, íntegramente, en el simulador Proteus. Diseñe el sistema propuesto en dicho simulador, o bien, descargue el proyecto base (*.pdsprj) disponible en el Campus Virtual, sección de prácticas, carpeta de la práctica 1.**



En la siguiente página se muestra un esquema tanto del microcontrolador (con un etiquetado de pines compatible con la tarjeta Arduino Mega 2560) como de la tarjeta de expansión con los periféricos disponibles en la tarjeta física o real, diseñado todo ello en el simulador Proteus. El estudiante podrá modificar este esquema para añadir nuevas funcionalidades o rediseñar aspectos que sean de su interés.

7.1 Tareas previas (no entregables)

7.1.1 Tarea 1. Salidas digitales

Hacer un programa para que haga parpadear (encender y apagar) todos los segmentos del dígito de las unidades (D4) durante 2 segundos, aproximadamente. Luego hacer que el parpadeo pase de forma secuencial al resto de los dígitos:

Unidades-D4 (parpadeo 2 seg) → decenas-D3 (parpadeo 2 seg) → centenas-D2 (parpadeo 2 seg) → millares-D1 (parpadeo 2 seg) → unidades-D4 (parpadeo 2 seg) y así, sucesivamente.

Al cambiar de dígito, generar una señal acústica para avisar al usuario. Tiene completa libertad para elegir la duración y la frecuencia del pitido (o señal acústica) que se oirá por el zumbador/altavoz.

7.1.2 Tarea 2. Comunicación serie

Uso de las comunicaciones serie (TX0/RX0). Aprovechando el código de las tareas anteriores, escribir un nuevo programa consistente en presentar al usuario un menú en el "monitor serie" del entorno de programación Arduino (Herramientas->Monitor Serie) con al menos 6 opciones con las que podamos seleccionar una de las siguientes acciones (el carácter inicial a visualizar en los dígitos del display será todos los segmentos encendidos):

1. Parpadeo de las unidades
2. Parpadeo de las decenas
3. Parpadeo de las centenas
4. Parpadeo de las unidades de millar
5. Parpadeo secuencial con todos los dígitos (tarea1)
6. Selección del carácter hexadecimal (0-F) a visualizar en el display

Entrar opción:

7.1.3 Tarea 3. Sincronización por consulta de estado

Realizar un contador electrónico de dos dígitos (decenas-unidades) que se actualiza dependiendo del pulsador que se pulse: pulsador "pup" incrementa en una unidad, pulsador "pdown" decrementa en una unidad y pulsador "pcenter" inicializa a "0". La detección de la pulsación se ha de hacer por consulta de estado. Si estando el contador a 00 se pulsa el pulsador "pdown" el contador ha de cambiar a 99. Si con este último valor se pulsa "pup" el contador deberá regresar a 00. Cada vez que cambie el estado del contador ha de oírse un pitido corto por el zumbador.

En este apartado se deberá afrontar la complejidad adicional de mostrar varios dígitos simultáneamente en el visualizador. Para ello, es necesario de un mecanismo de visualización "entrelazada" en el que se va alternando la visualización de las unidades y las decenas a una cierta frecuencia (100Hz, 10ms). Por ejemplo: en el primer ciclo (10 ms), se visualizan las unidades y se apagan las decenas; en el segundo ciclo, se apagan las unidades y se encienden las decenas; en el

tercer ciclo se visualizan las unidades y se apagan las decenas y así, sucesivamente. Si el entrelazado se hace lentamente, se observa un parpadeo en el visualizador. Sin embargo, a partir de una cierta frecuencia se ve estático porque el ojo humano no es capaz de apreciar la conmutación entre dígitos. A continuación, se muestra la estructura general del programa que le puede ayudar a realizar este apartado:

```
boolean estado=false;
Setup(): Inicialización
Loop(): Programa principal {
    Leer pulsadores y actualizar variables del contador
    If (estado) then {
        Apagar decenas, encender y visualizar unidades
        Delay 10 ms}
    Else {
        Apagar unidades, encender y visualizar decenas
        Delay 10 ms }
    estado = !estado;
}
```

7.1.4 Tarea 4. Visualización sincronizada por interrupción externa

El objetivo de esta tarea es que la visualización de los dígitos del contador de la tarea anterior, la realice una subrutina de servicio de interrupción, de forma sistemática, cada 10 ms. Para ello, generaremos una interrupción de tipo externa cada 10 ms (frecuencia 100 Hz). Utilice un generador de señal y, ajústelo adecuadamente, para obtener la señal especificada y utilícela para generar una interrupción externa de nivel 2, pin 19 de la tarjeta Arduino.

Veamos cuál sería la estructura general de un programa para habilitar y tratar estas interrupciones adecuadamente. Para más información consultar la documentación de las clases de teoría.

```
// Declaración de variables volátiles
// Las variables que se modifiquen en ISR() y se utilicen
// fuera de ISR() han de declararse "volatile"
// ejemplo:
volatile boolean estado;

// función de setup()
void setup() {
    // Habilitación de la interrupción INT2, flanco de subida (rising)
    cli();
    EICRA |= (1<<ISC21) | (1<<ISC20);
    EIMSK |= (1<<INT2);
    sei();
}
```

```
}  
ISR (INT2_vect) {  
/*  
Rutina de servicio para la visualización entrelazada en el display de 4  
dígitos de 7-segmentos y barrido del teclado (si es utilizado).  
  
Dependiendo de la frecuencia con la que se interrumpa (cada 10 ms, en  
nuestro caso) se ejecuta o se entra en esta función.  
  
En cada entrada en esta función se visualiza un dígito (y se explora la  
columna correspondiente del teclado si queremos integrarlo en la  
aplicación).  
  
Ejemplo:  
1ª interrupción --> visualiza unidades y explora 1ª columna teclado  
2ª interrupción --> visualiza decenas y explora 2ª columna teclado  
3ª interrupción --> visualiza centenas y explora 3ª columna teclado  
4ª interrupción --> visualiza unidades de millar  
5ª interrupción --> visualiza unidades y explora 1ª columna teclado  
...  
}  
void loop() {  
...  
}
```

Para implementar la visualización entrelazada se procederá de forma que, la primera vez que se interrumpa, la ISR() de la interrupción INT2 apagará todos los dígitos y activará el dígito de las unidades (solo se visualizaría las unidades del contador); en la segunda interrupción, la ISR() apagará todos los dígitos y activará el dígito de las decenas (solo se visualizará las decenas del contador) y así, sucesivamente, de forma alternada cada 10 ms entre unidades, decenas, centenas y unidades de millar, dependiendo de cuantos dígitos queramos visualizar. Observe que, si baja la frecuencia del generador de señal el display parpadeará y podrá observar el funcionamiento de la visualización entrelazada.

7.2 Aplicación a desarrollar (entregable para evaluar)

Una vez realizadas las tareas previas no entregables, se propone desarrollar la siguiente aplicación mediante un proyecto en el simulador Proteus (*.pdsprj) que, una vez terminado, se subirá al Campus Virtual para su evaluación.

7.2.1 Especificaciones de diseño

Implementar una aplicación con una funcionalidad similar a la de un Turnomatic (dispositivo para dar un turno en una cola de clientes) en el que la visualización en el display y la exploración del teclado se haga de forma entrelazada y sincronizada por una interrupción cada **5 ms** generada por el **pin 18 (INT3)**.

La aplicación se implementará de acuerdo a las siguientes especificaciones:

1. Contador de 3 dígitos
2. Control basado en 4 pulsadores con las siguientes funcionalidades:
 - a. PUP: Incrementa el contador
 - b. PDOWN: Decrementa el contador
 - c. PENTER: Puesta a cero (reset) del contador
 - d. PLEFT: El contador incrementará o decrementará su cuenta de 1 en 1
 - e. PRIGHT: El contador incrementará o decrementará su cuenta de 2 en 2.
 - f. Señal acústica por el altavoz del sistema cada vez cambie el estado del contador para así avisar a los clientes.
3. Inicialización del contador a cualquier valor, entre 000 y 999 mediante el teclado de 4x3
 - a. Para modificar la cuenta con el teclado bastará con teclear los números y terminar con la tecla “#” que hará la función de “enter o entrar”. Ejemplo; 235#
4. Mostrar en la pantalla del PC (o en el terminal virtual de Proteus) el siguiente menú de opciones para poder elegir una de las acciones mostradas:
 - 1.- Modo normal de visualización (tres dígitos): OFF-centenas-decenas-unidades
 - 2.- Modo reducido-inferior de visualización (dos dígitos): OFF-OFF-decenas-unidades
 - 3.- Modo reducido-superior de visualización (dos dígitos): decenas-unidades-OFF-OFF

7.2.2 Mejoras del diseño: Sensor de temperatura (opcional)

Como mejora se sugiere dotar al Turnomatic de la capacidad de medir y mostrar la temperatura del entorno. Para ello, se puede añadir un sensor de temperatura a través de un canal analógico y desarrollar el software necesario para realizar la medida y mostrarla. La temperatura será visualizada en el display de 7-segmentos, pero alternando entre el valor del contador y la temperatura, como lo hace cualquier panel de información. Escoja de las librerías de componentes de Proteus el sensor de temperatura que estime más adecuado.

8 Entrega del informe de práctica

Una vez desarrollada la aplicación y comprobado su correcto funcionamiento es, absolutamente necesario, subir el informe de la práctica al Campus Virtual de la asignatura, en tiempo y forma, a través del enlace disponible en la sección de prácticas. El informe consistirá en el proyecto Proteus (*.pdsprj) de la aplicación con los programas adecuadamente comentados. En principio, solo será necesario subir un fichero cuyo nombre seguirá la siguiente nomenclatura:

24-25_plab1_iniciales nombre y apellidos.pdsprj

Ejemplo:

24-25_plab1_amf.pdsprj

Fichero correspondiente al proyecto Proteus de la práctica de laboratorio 1 entregado por Anabel Medina Falcón.