ESCUELA DE INGENIERÍA INFORMÁTICA

Grado en Ingeniería Informática

PERIFÉRICOS E INTERFACES

PRÁCTICAS EN LABORATORIO

Práctica 4

Diseño e implementación de un reloj despertador

Última actualización: 14 noviembre 2023

Contenido

1 Competencias y objetivos de la práctica

La cuarta práctica de la asignatura Periféricos e Interfaces se centra en el uso de los conocimientos teóricos asociados al módulo 3 de la asignatura, relativos a los periféricos de entrada y salida de datos, si bien, es necesario el bagaje adquirido a lo largo del curso. En esta práctica, se plantea el diseño e implementación de un reloj despertador basado en el microcontrolador ATmega 2560. Para ello, se hará uso del chip DS3232 que es un reloj en tiempo real con los suficientes recursos para mantener la hora y fecha del sistema, el establecimiento de alarmas y otras funciones como medir la temperatura del chip.

La realización de la práctica por parte de los estudiantes se orienta a complementar la consecución de la competencia de título CII01 del Plan de Estudios de la Ingeniería Informática y que los capacita para: diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos asegurando su fiabilidad, seguridad y calidad conforme a principios éticos y a la legislación y normativa vigente. En base a ello, con la realización de esta práctica se pretende que los estudiantes alcancen satisfactoriamente las siguientes capacidades:

1. Capacidad para entender e interrelacionar los diferentes componentes hardware y software en el diseño de un determinado dispositivo que haga uso de un sistema empotrado basado en microcontrolador.
2. Capacidad para diseñar, implementar y verificar el correcto funcionamiento de dispositivos externos sencillos para ser conectados a un microcontrolador.
3. Capacidad para el desarrollo de programas que permitan el control básico del dispositivo haciendo uso de un lenguaje de programación.
4. Capacidad para desarrollar programas que faciliten el uso del dispositivo externo a un usuario final.
5. Capacidad para aprender y aplicar nuevos conceptos de forma autónoma e interdisciplinar.
6. Capacidad para emplear la creatividad en la resolución de los problemas.

El logro de las citadas y pretendidas capacidades pasa por el planteamiento de un conjunto de intenciones y metas que orientan el proceso de aprendizaje de los estudiantes y que constituyen los objetivos de la práctica. En concreto, se propone alcanzar los siguientes objetivos:

1. Conocer y entender la funcionalidad de los diferentes pines de los puertos de E/S del microcontrolador Atmega 2560 así como los múltiples aspectos relativos a la conexión con el hardware externo.
2. Conocer la estructura interna y uso de los diferentes interfaces del microcontrolador Atmega 2560 que sean necesarios para conectar los dispositivos periféricos de esta práctica.
3. Conocer los aspectos prácticos de funcionamiento de los diferentes componentes básicos a utilizar en la práctica.
4. Entender la integración y conexión de componentes básicos para diseñar el dispositivo propuesto y que cumpla con las funcionalidades establecidas.
5. Analizar, diseñar e implementar el software de control del dispositivo para su correcto funcionamiento en base a las funcionalidades especificadas.
6. Verificar y depurar la integración hardware/software realizada para la aplicación hasta alcanzar un funcionamiento satisfactorio y fiable. Rediseñar si fuese necesario.

2 Documentación previa

La documentación básica a utilizar para la realización de esta práctica está disponible en la página web de la asignatura ubicada en el Campus Virtual de la ULPGC. La documentación mínima a manejar será la siguiente:

- Enunciado de la práctica: este documento
- Transparencias de teoría correspondientes a los módulos 1, 2 y 3.
- Transparencias de la presentación de la práctica
- Hojas de características de los componentes electrónicos (subdirectorio "Doc" de la práctica 3)
- Página web de Arduino: http://www.arduino.cc/

3 Descripción del dispositivo: reloj despertador

3.1 Descripción general

La actividad práctica a realizar consistirá en el diseño de un reloj despertador basado en el microcontrolador ATmega 2560 que será nuestro sistema empotrado. Las funcionalidades básicas del reloj despertador se basan en el chip de reloj de tiempo real DS3232 que, una vez programado, será el encargado de mantener la fecha y hora del sistema y la gestión de hasta 2 alarmas, entre otras funciones. Una vez el dispositivo haya sido diseñado y programado, deberá funcionar como cualquier reloj despertador doméstico.

Para la realización de la práctica, se proporcionará un proyecto básico (Proteus) con todos los componentes necesarios para implementar la funcionalidad de un despertador. La figura 1, muestra un diagrama de bloques del dispositivo a implementar y que, básicamente, consta de:

1. Pantalla Liquid-Crystal Display (LCD) de 4 líneas y 20 caracteres/línea (bus serial)
2. Reloj de tiempo real RTC DS3232 (bus i2c)
3. Teclado de 4 filas x 3 columnas (interconexión paralela) y altavoz o zumbador.

Evidentemente, el dispositivo admite varios diseños como sería el uso de pulsadores en vez de un teclado, que sería una solución más acorde con los dispositivos que se encuentra en el mercado. Sin embargo, y por razones académicas, hemos optado por hacer uso de un teclado de 4x3 para que el estudiante adquiera la capacidad de control y gestión de un teclado en tiempo real, al menos para cambiar de modo de funcionamiento.

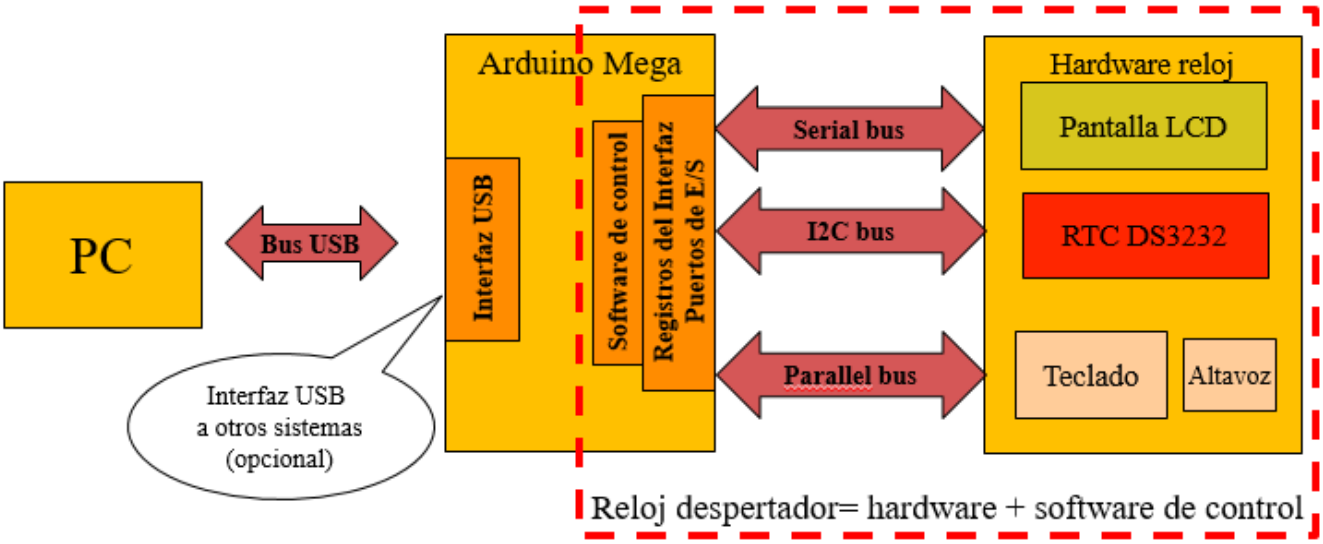


Figura 1. Diagrama de bloques del hardware del reloj-despertador

El funcionamiento básico consiste en que, una vez inicializado el reloj despertador (hora, fecha y posibles alarmas), el dispositivo empezará a visualizar hora, fecha, alarmas y temperatura del chip, de forma indefinida. En cualquier momento, se podrá entrar en el procedimiento para realizar los ajustes que se estimen oportunos: modificación de fecha, hora y/o alarmas.

El sistema se podría enriquecer añadiendo alguna otra funcionalidad como podría ser el funcionamiento en modo cronómetro, que queda como propuesta para una posible mejora.

3.2 Información en la pantalla LCD

La información a mostrar en la pantalla LCD, se hará de acuerdo a lo mostrado en la figura 2. En ella, podemos apreciar la visualización de: hora, fecha, alarmas y estado (activas/no activas) y temperatura.

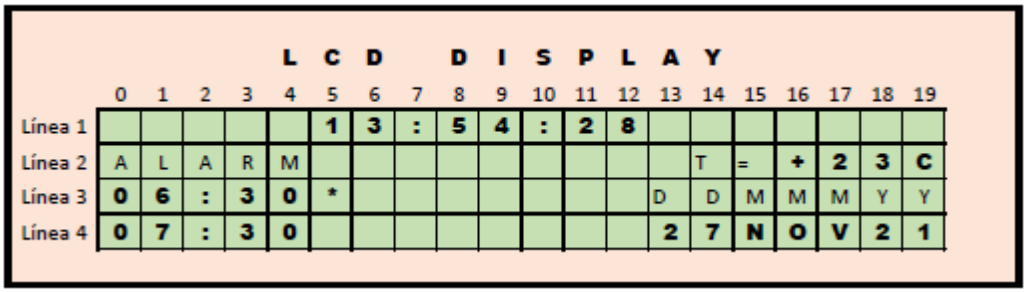


Figura 2. Información a mostrar en pantalla LCD

Por criterios de homogeneidad, en la siguiente tabla se muestran los diferentes campos a visualizar en la pantalla LCD indicándose posición y posibles valores a visualizar:

Campo	Fila	Columna	Caracteres	Descripción
Hora	1	5	8	Hora en formato → hh:mm:ss
Etiqueta (fija)	2	0	5	Etiqueta: ALARM
Etiqueta (fija)	2	14	2	Etiqueta: T=
Temperatura (T)	2	16	3	Temperatura medida por el DS3232
Etiqueta (fija)	2	19	1	Etiqueta: C
ALARMA1	3	0	5	Hora de la ALARMA 1
ACTIVE1	3	5	1	*: Activada espacio: No activada
Etiqueta	3	13	7	Etiqueta: DDMMMYY
ALARMA2	4	0	5	Hora de la ALARMA 2
ACTIVE2	4	5	1	*: Activada espacio: No activada
Fecha	4	13	7	DDMMMYY MMM → JAN-FEB-MAR-APR-MAY-JUN-JUL-AUG-SEP-OCT-NOV-DEC

Tabla 1.- Campos de información de la pantalla LCD

Como se puede deducir, toda la información que se muestra en la pantalla, se encuentra en el chip RTC DS3232 por lo que, cada segundo, como mínimo, es necesario leer la información almacenada en los registros del DS3232 y visualizarla en los diferentes campos de la pantalla en la forma adecuada. Esta tarea estará asociada a una interrupción generada por un timer que interrumpiría, al menos, una vez por segundo. En cuanto a las alarmas, cabe decir que el DS3232 dispone de dos alarmas que, si se habilitan, pueden generar interrupciones cada vez que se alcanzan las condiciones para que se produzca una alarma (coincidencia en segundos, minutos, etc.). Para el uso de interrupciones, será necesario conectar la petición de interrupción del DS3232 (pin INT'/SQW), a un pin de interrupción externa del microcontrolador Atmega 2560 (p.ej. INT0, pin 21), habilitar las interrupciones (tanto en el microcontrolador como en el DS3232) e implementar la ISR() correspondiente para atender estas interrupciones

que, en su función básica y como corresponde a una alarma, debería generar un sonido o melodía durante un tiempo determinado, incluso con repetición, hasta que el usuario desactive la alarma.

3.3 Configuración del reloj despertador

Para realizar los ajustes necesarios del reloj despertador se podría optar por alguna de las dos opciones siguientes:

1. Utilizar la pantalla de Proteus (virtual terminal) para la visualización de los diferentes menús de configuración del reloj despertador e interactuar con ellos. Solución alejada a lo que sería el diseño de un reloj despertador real pero didáctico en nuestro caso.
2. Utilizar solo el teclado (también podría ser los pulsadores) y la pantalla LCD. Solución próxima al comportamiento de un reloj despertador real pero algo más compleja.

Por defecto, utilizaremos la opción 1 para el desarrollo de la práctica.

Para entrar y salir del modo de configuración se utilizará el teclado de 4x3 de acuerdo a las siguientes secuencias de pulsaciones de teclas:

Secuencia de teclas *# → Entrada en modo configuración

Secuencia de teclas #* → Salida del modo de configuración.

La operativa exacta para la configuración del reloj despertador a través del "Virtual Terminal" de Proteus, se deja a criterio e iniciativa del estudiante. En cualquier caso, cuando en el teclado de 4x3 se pulsa la secuencia *# se debe entrar en el modo configuración y se mostrará el menú1 (principal) en el "Virtual Terminal". A través de este menú principal y submenús, el usuario interactuará para realizar los ajustes que sean necesarios para configurar e inicializar los diferentes aspectos de funcionamiento del reloj despertador. La organización del menú principal y submenús podría ser algo parecido a:

Menú1 (principal)	**Menú2 (secundario)**	**Petición de datos**	
1.- Ajustar hora	1.- Hora: 2.- Minuto 3.- Segundo 4.- Exit	Introducir hora: 13 Introducir min: 25 Introducir seg: 54	
2.- Ajustar fecha	1.- Día 2.- Mes 3.- Año 4.- Exit	Introducir día: 13 Introducir mes: 10 Introducir año: 2021	
3.- Ajustar alarma 1	...	...	
4.- Ajustar alarma 2	...	...	
....			

Tabla 2.- Menús de configuración

Con la secuencia de teclas #*, del teclado de 4x3, saldríamos del modo de configuración quedando el sistema en modo de visualización.

3.4 Componentes básicos

En este apartado se describen los componentes básicos necesarios para la implementación de la práctica. Algunos componentes básicos vistos en prácticas anteriores (Arduino ATmega 2560, display, teclado, pulsadores, altavoz o zumbador, timers, motores y memoria I2C) no serán objeto de descripción en esta práctica por lo que se remite al lector a los enunciados de prácticas anteriores.

3.4.1 Pantallas LCD

3.4.1.1 Aspectos hardware de las pantallas LCD

La tarjeta de expansión del Arduino dispone de una pantalla LCD de 20 caracteres por 4 líneas (20x4) con interfaces de tipo serial o I2C, seleccionables con un "jumper". En la práctica utilizaremos, preferiblemente, el interfaz serial. En las tarjetas de expansión disponibles en el laboratorio encontraremos dos tipos de pantallas LCD debido a que fueron adquiridas en pedidos diferentes. **Si utilizamos el simulador Proteus, también tendríamos acceso a una nueva pantalla (tipo 3) algo diferente a las instaladas físicamente en las tarjetas de expansión**. Los tipos de tarjetas son:

Tipo 1: NHD-0420D3Z (perfil bajo)

Esta pantalla se encuentra en las siguientes tarjetas experimentales:

Tarjetas: 1,2,3,4,5,6,7 y 11,12,13,14,15,16,17 **Interfaz por defecto:** Serial (TTL)

Tipos de interfaces serie (seleccionables por jumpers): Serial, I2C y SPI.

La Figura 3 muestra su aspecto visual: **perfil bajo, con separadores cortos.**

Tipo 2: C2042A (perfil alto)

Pantalla LCD con interfaz paralelo de 16 bits a la que se le ha añadido un circuito adicional (daughter board) para dotarla de un interfaz serie (TTL) o I2C, seleccionable por jumper, y de una conexión a teclado matricial (funcionalidad no usada en esta práctica). Esta pantalla se encuentra en las siguientes tarjetas experimentales:

Tarjetas: 8, 9, 10, 18, 19 **Interfaz:** Serial (jumper colocado) o I2C

Pantalla con interfaz paralelo (se le ha retirado el adaptador serial/I2C):

Tarjeta: 20 **Interfaz por defecto:** paralelo 16 bits

La Figura 4 muestra su aspecto visual: **perfil alto, con separadores largos.**

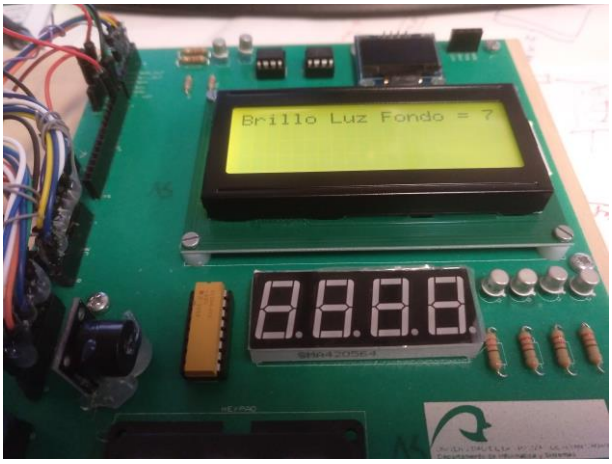


Figura 3.- Tipo 1: NHD-0420D3Z (perfil bajo)

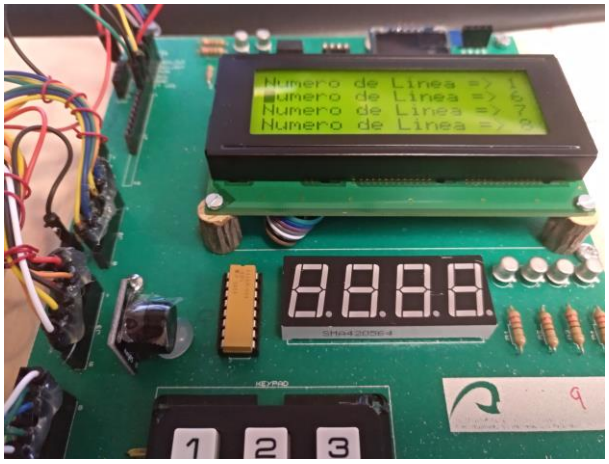
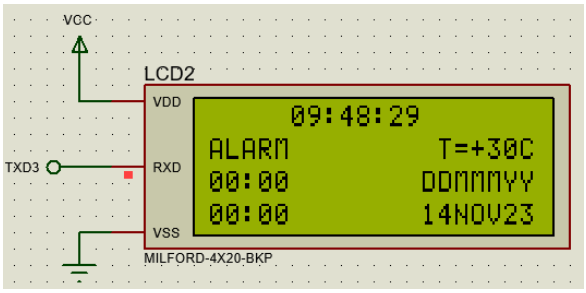


Figura 4.- Tipo 2: C2042A (perfil alto)

Tipo 3: Milford-4x20-BKP (simulador Proteus)

Pantalla de 4 líneas y 20 caracteres por línea, con interfaz serie y disponible en la librería "Serial LCDs" de Proteus.



Todas las pantallas tienen las mismas características de transmisión que coinciden con las que tiene el Arduino Mega 2560 por defecto (TX0:3/RX0:3):

9600 baudios, 8 bits, no parity, 1 stop bit

3.4.1.2 Programación y uso de la pantalla MILFORD-4x20-BKP (simulador Proteus)

La información a recibir por la pantalla se puede clasificar en:

- Códigos ASCII de caracteres: para ser visualizados
- Códigos de control: los interpreta y realiza alguna acción (borrado, parpadeo, posicionar el cursor, …). Van precedidos del carácter de control 0xFE.

Tabla de comandos			
Prefix	**Command**	**Description**	**Execution time**
-	-	Display Character Write (0x00 ~ 0xFF)	
0xFE	1	Clear Screen	10mS
0xFE	24	Scroll display one character left	
0xFE	28	Scroll display one character right	
0xFE	0	Home (and undo scrolling)	10mS
0xFE	16	Move cursor one character left	
0xFE	20	Move cursor one character right	
0xFE	14	Turn on underline cursor	
0xFE	13	Turn on blinking cursor	
0xFE	12	Turn off cursor	
0xFE	8	Blank the display (retaining data)	
0xFE	12	Restore the display (without cursor)	
0xFE	128 + address	Set display (DD) RAM address	
0xFE	64+address	Set character (CG) RAM address	

Tabla 3.- Tabla de comandos de la pantalla Milford 4x20 BKP

La pantalla tiene una memoria interna (comienza en la dirección 128) donde se escriben los caracteres que se quieran visualizar en la pantalla. La correspondencia entre la dirección de memoria y la posición del carácter en la pantalla, se muestra en la siguiente figura:

4x20 LCD

Line 1	address:	0	1	2	3	4	 19
	DD address:	128	129	130	131	132	147
Line 2	address:	64	65	66	67	68	 83
	DD address	192	193	194	195	196	211
Line 3	address:	20	21	22	23	24	 39
	DD address:	148	149	150	151	152	167
Line 4	address:	84	85	86	87	88	103
	DD address	212	213	214	215	216	231

Ejemplo: Para visualizar el carácter “A” en la penúltima columna de la línea 3 del display, se ha de escribir el dato 0x41 (código ASCII de la A) en la dirección 166 de la memoria de la pantalla LCD.

Para ello, tenemos que ejecutar previamente el comando “set display (DD) RAM address” para indicar la posición de memoria de pantalla en la que vamos a escribir y luego enviaremos el carácter.

```
Serialx.write(0xFE);  Serialx.write(166); Serialx.write("A");
```

“x” se corresponde con el número del canal serie (0,1,2 o 3) al que está conectado la pantalla. A continuación, se muestran algunas líneas de código para inicializar y hacer uso de la pantalla Milford suponiendo que esté conectada al canal serio TX3/RX3:

MILFORD

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  Serial3.begin(9600); //canal 3, 9600 baudios,
                       // 8 bits, no parity, 1 stop bit
  Serial3.write(0xFE); Serial3.write(0x01); //Clear Screen
  delay(100);

  Serial3.write(0xFE); Serial3.write(0x00); // Cursor Home
  delay(100);                                // posicionarse en línea 1
  Serial3.write("12345 Hola a Todos "); delay(1000); // linea 1

  Serial3.write(0xFE); Serial3.write(0xC0); // posicionarse en linea 2
  Serial3.write("67890 Hola a Todas "); delay(1000);  // linea 2

  Serial3.write(0xFE); Serial3.write(0x94); // posicionarse en linea 3
  Serial3.write("01234567890123456789"); delay(1000);  // linea 3

  Serial3.write(0xFE); Serial3.write(0xD4); // posicionarse en linea 4
  Serial3.write("abcdefghijklmnopqrst"); delay(1000);  // linea 4

}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(LED_BUILTIN, LOW);   // LED off
  Serial3.write(0xFE); Serial3.write(0x08); // Display off
  delay(1000);
  Serial3.write(0xFE); Serial3.write(0x0C); // Display on
  delay(1000);
}
```

La siguiente figura muestra el resultado de ejecutar el programa anterior donde la pantalla se enciende y se apaga como consecuencia de las líneas de código de la función loop().

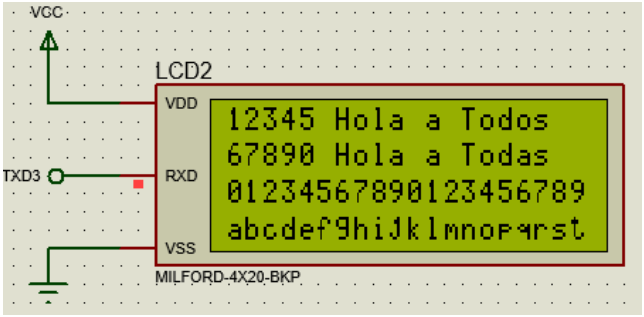


Figura 5.- Información en la pantalla LCD después de ejecutar la función de setup().

3.5 Esquema general (simulador Proteus)

A continuación, se muestran las conexiones de los diferentes componentes en el simulador Proteus.

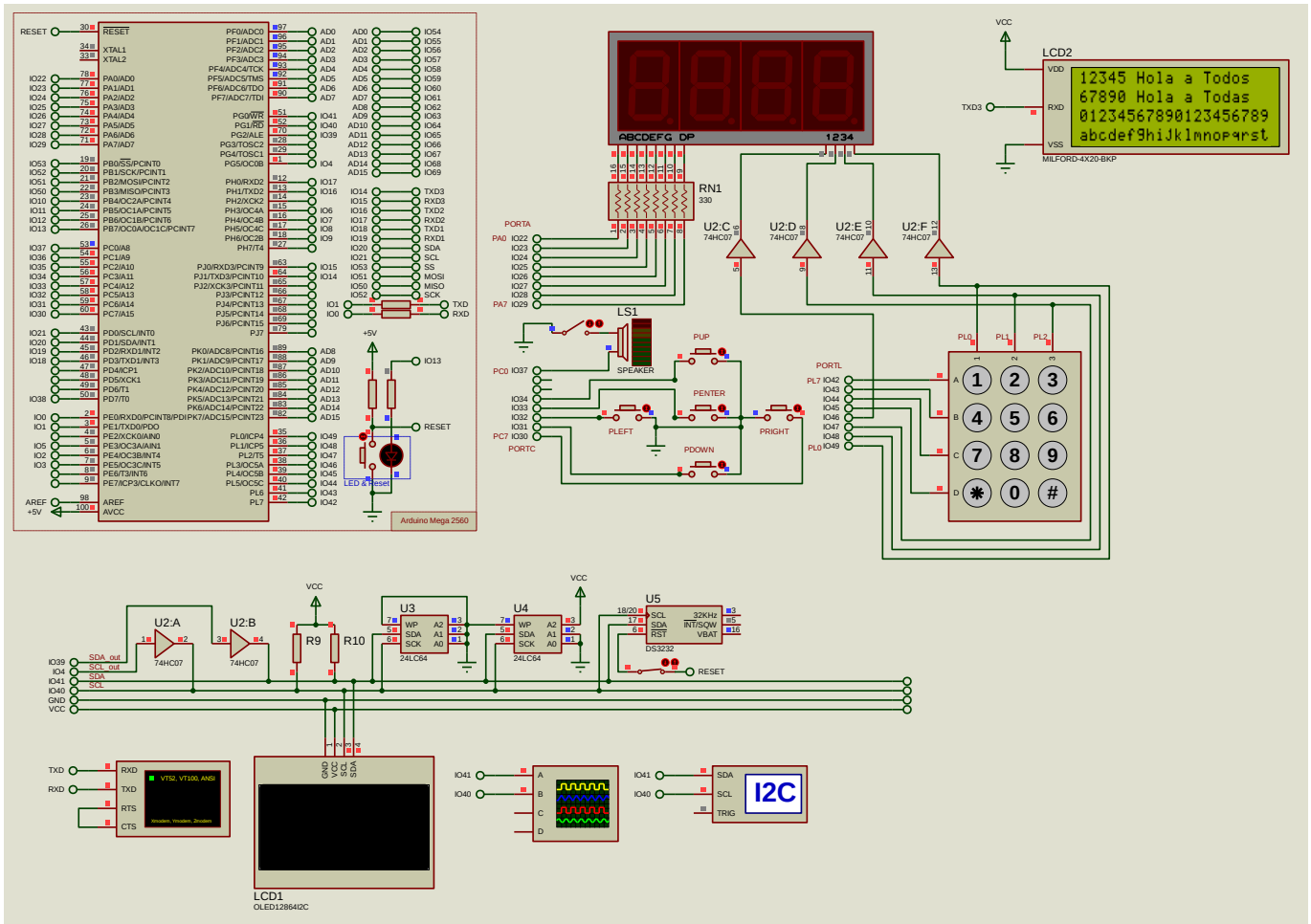
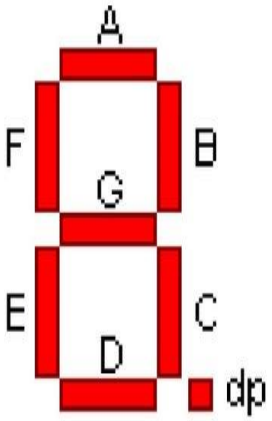


Figura 6.- Esquema general (simulador Proteus)

4 Aspectos prácticos para la implementación

4.1 Display 7-segmentos

La conexión y funcionamiento del display de 7 segmentos ya ha sido visto en prácticas anteriores por lo que sólo mostraremos la tabla de conexiones a los puertos. La visualización secuencial de los dígitos y la exploración del teclado de 4x3 se harán de forma entrelazada como en prácticas anteriores.



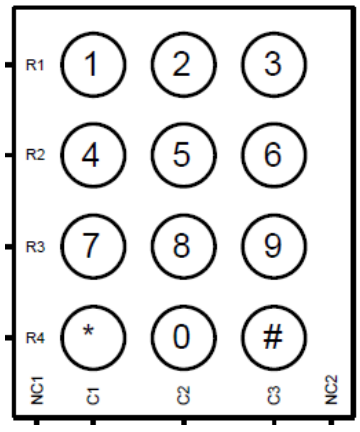
	Puertos Arduino								**Valor**
bit →	**PA7**	**PA6**	**PA5**	**PA4**	**PA3**	**PA2**	**PA1**	**PA0**	
Segmento→	***dot***	***g***	***f***	***e***	***d***	***c***	***b***	***a***	
0	-	0	1	1	1	1	1	1	0x3F=63
1	-	0	0	0	0	1	1	0	0x06=06
2	-	1	0	1	1	0	1	1	0x5B=91
3									
4									
5									
6									
7									
8									
9									

Tabla 4.- Códigos de 7 segmentos

4.2 Teclado matricial 4x3

El teclado matricial, de 4 filas por 3 columnas, será explorado síncronamente con el display de 7-segmentos de modo que cada vez se seleccione un dígito para visualizarlo se explorará una columna del teclado: poner un "0" lógico en la columna que corresponda y luego leer las filas para ver si hay algún cero en alguna de ellas. 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.

La siguiente tabla 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
D4-C1 = 0	0	1	1	1	**1**
	1	0	1	1	**4**
	1	1	0	1	**7**
	1	1	1	0	*
D3-C2 = 0	0	1	1	1	**2**
	1	0	1	1	**5**
	1	1	0	1	**8**
	1	1	1	0	**0**
D2-C3 = 0	0	1	1	1	**3**
	1	0	1	1	**6**
	1	1	0	1	**9**
	1	1	1	0	#

Tabla 5.- Barrido del teclado y del display de 7 segmentos.

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-nn	activar D1
47	PL2-out	D2-C3	activar D2 + col. 3 teclado
48	PL1-out	D3-C2	activar D3 + col. 2 teclado
49	PL0-out	D4-C1	activar D4 + col. 1 teclado

Nota: Orden de los dígitos en el display --> D1-D2-D3-D4 (menos significativo)

Tabla 6.- Pines teclado-display 7 segmentos

4.3 Pulsadores

Los pulsadores ya han sido analizados en prácticas anteriores por lo que sólo mostramos la tabla de conexiones a los pines del Arduino. **Será necesario activar las resistencias de pull-up internas del Arduino para definir un "1" lógico cuando el pulsador no esté pulsado.** Recuerde que para activar una resistencia de pull-up es necesario definir el pin de entrada y luego escribir un "1" en dicho pin.

La conexión de los pulsadores a los pines de PORTC del Arduino, responde a la siguiente tabla:

PORTC			
Nº pin	**Puerto**	**Nombre señal**	**Función**
30	PC7-in	pright	Multiplicar
31	PC6-in	pdown	Restar
32	PC5-in	pleft	Dividir
33	PC4-in	penter	Enter
34	PC3-in	pup	Sumar
35	PC2	-	
36	PC1	-	
37	PC0-out	speaker	Altavoz

Tabla 7.- Pines pulsadores y altavoz

4.4 Pantalla LCD

MILFORD-4x20-BKP (simulador Proteus)

Arduino	Conector Pantalla Serial (TTL)
Pin14-TX3	RXD
GND	VSS
+5V	VDD

Tabla 8.- Pines pantalla LCD

4.5 Organización del software de control

El software de control del reloj despertador se desarrollará bajo el entorno de programación de Proteus haciendo uso de las utilidades que nos ofrece su lenguaje de programación. Como todo programa que se desarrolle en este entorno, consta de:

1.- Declaración de variables

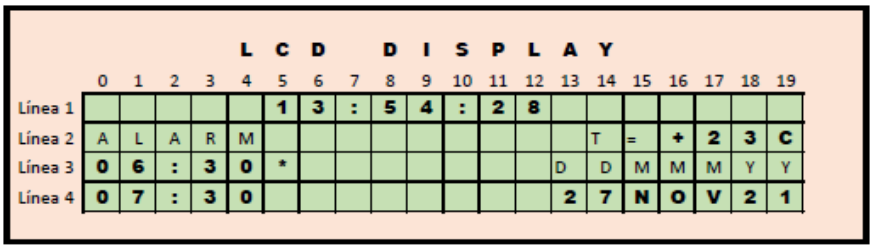
2.- Código de inicialización (se ejecuta sólo una vez): void setup() { ….}

3.- Código de funciones o métodos auxiliares

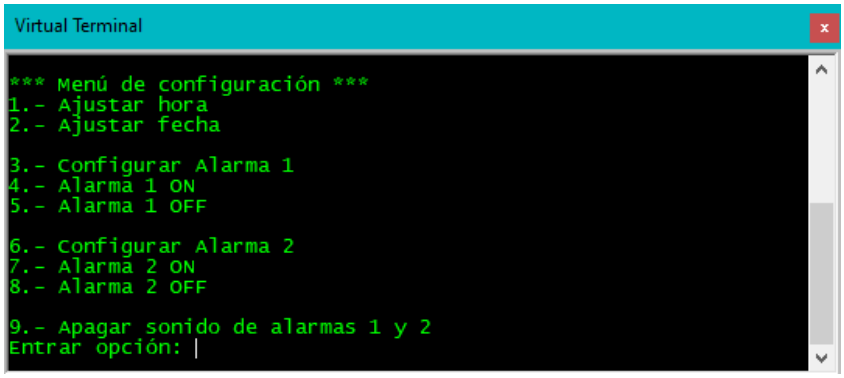
3.- Código de ejecución indefinida: void loop() { …. }

La aplicación a desarrollar permitirá elegir entre dos modos de funcionamiento del sistema:

- **Modo visualización**: En la pantalla LCD se muestra la información asociada al reloj despertador suministrada por el RTC DS3232. Para una correcta visualización de la hora, la información en la pantalla se ha de actualizar, como mínimo, cada segundo.



- **Modo configuración**: El sistema entra en un proceso de configuración en el que se puede cambiar los datos asociados al reloj despertador ubicados en el chip DS3232. La configuración se realizará a través de menús en la pantalla de Proteus (virtual terminal), similar al mostrado en la siguiente figura:



El **modo de funcionamiento del sistema** se podrá elegir mediante una secuencia de pulsaciones de teclas en el teclado de 4x3:

- Teclas *#: Entrada en modo de configuración
- Teclas #*: Salida del modo de configuración. Retorno al modo de visualización

Cuando se pulse la secuencia de teclas #*, deberá aparecer un mensaje de "… fin de configuración" en la pantalla (virtual terminal) y retornar al modo de visualización.

El software de control se organizará básicamente entorno a dos tipos de tareas: la tarea principal, asociada a la función loop() y, otras tareas asociadas, normalmente, a rutinas de servicio de interrupción ISR() tales como: a) visualización de información en la pantalla LCD, b) visualización entrelazada en el display de 7 segmentos y exploración del teclado de 4x3 y c) gestión de las alarmas 1 y 2.

Tarea 1: Tarea principal (en loop())

La tarea principal de la aplicación asociada a la función loop() detectará el modo de funcionamiento del sistema, a través de una variable global de "modo", y ejecutará el código que corresponda a dicho modo. La variable de "modo" será actualizada en la rutina de servicio de interrupción ISR() del teclado-display tan pronto se detecte un cambio de modo a través de las secuencias de pulsaciones de teclas en el teclado de 4x3. Las acciones a realizar en cada uno de los modos son:

Modo 0: Visualización

Reservado para futuras funciones. La visualización de la información en la pantalla LCD está asociada a una ISR() por lo que la funcionalidad asociada a este modo en loop() es nula.

Modo 1: Configuración

Ejecución del menú de configuración del reloj despertador a través del "Virtual Terminal" mientras se esté en este modo.

En resumen, en loop() tendremos la siguiente funcionalidad:

```
loop(){
Mientras (modo == 0) {// Modo visualización. Nada que hacer;}
Mientras (modo == 1) {// modo configuración
        // implementar el menú de configuración: ajuste de parámetros
        // cambiar hora, alarmas y fecha
}
```

Subrutinas de servicio de interrupción: ISR()

La tarea principal será interrumpida, al menos, por tres interrupciones que han de ser tratadas por las correspondientes ISR().

A) ISR (TIMER3_COMPA_vect). Rutina de servicio del Timer 3 que se ejecuta **cada 10 ms**. Se encarga de la exploración del teclado y la visualización en el display de 7 segmentos (si es necesario). Además, detecta las secuencias de teclas para el cambio de modo de funcionamiento: visualización o configuración.

B) ISR(TIMER1_OVF_vect). Rutina de servicio del Timer 1 que se ejecuta **cada 0.5 segundos**. Se encarga de actualizar la información en la pantalla del reloj a partir de los datos leídos del chip DS3232.

C) ISR(INT0_vect): Interrupción asociada al pin 21 que se conecta al pin de interrupciones del DS3232 (INT'/SQW) que se activa cuando se produce una alarma.

5 Realización práctica

5.1 Tareas previas (no entregables)

Estudio del chip RTC DS3232 y realizar un esbozo de las funciones básicas que serían adecuadas o necesarias para programar de una forma cómoda el chip DS3232, accediendo a cualquiera de sus recursos.

Rescatar las funciones del i2c de la práctica 3 e integrarlas en el nuevo proyecto para utilizarlas cuando sean necesarias. También la ISR() de la gestión entrelazada del display 7-segmentos y teclado de las prácticas anteriores.

5.2 Aplicación a desarrollar (entregable para evaluar)

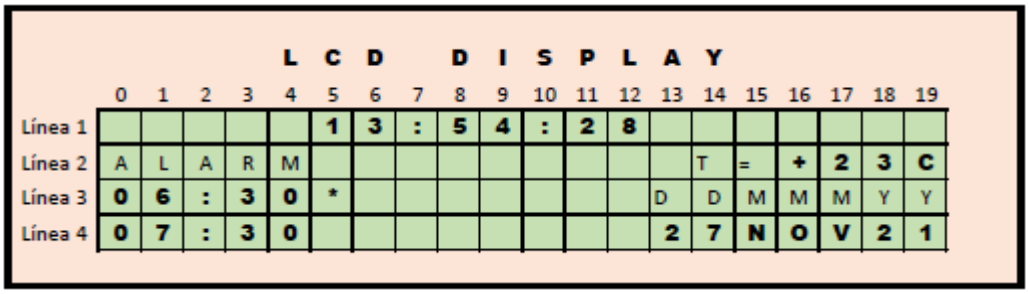
La implementación del software de control del reloj despertador, se realizará de forma incremental y en orden creciente de complejidad. Básicamente, se distinguen dos grandes bloques: visualización y configuración.

La visualización se corresponde con la actualización de la información del reloj despertador en la pantalla LCD. La información la suministra el RTC DS3232.

Las tareas de configuración del reloj despertador se realizarán a través de un menú principal y submenús que se visualizarán en el "Virtual Terminal" de Proteus.

5.2.1 Parte 1: Actualización de información en la pantalla LCD

En esta tarea, ha de actualizar la información que se muestra en la pantalla LCD del reloj despertador para lo que es necesario leer los datos del RTC DS3232:



Esta tarea se ejecuta cada 0.5 segundos y estará asociada a la rutina de servicio de interrupción del Timer 1:

```
ISR(TIMER1_OVF_vect){
// Rutina de servicio del timer 1. Se ejecuta cada 0.5 segundos (2 Hz)
// Actualiza información de la pantalla del reloj a partir de los datos
// suministrados por el chip DS3232
}
```

Especificaciones para programar el Timer 1: Modo 15 (Fast PWM, TOP = OCR1A), N=1024

Para ello, en el setup(), se ha de programar el Timer 1 para que genere una interrupción cada 0.5 segundos. También, será necesario implementar el cuerpo de la ISR() con la funcionalidad de leer la información del DS3232 y ponerla en los campos adecuados de la pantalla LCD, para obtener una visualización similar a la de la figura anterior.

5.2.2 Parte 2: Configuración de la fecha y hora

En esta tarea, se añaden nuevas funcionalidades a las establecidas en la Tarea 1: se añadirán las capacidades de configurar la fecha y la hora del reloj haciendo uso de un menú principal y submenús a través del "Virtual Terminal" de Proteus. Las acciones básicas a desarrollar son:

A) En setup(): Programar el Timer 3 para que genere una interrupción cada 10 milisegundos

   Especificaciones para programar el Timer 3: Modo 4 (CTC, TOP = OCR3A), N= 64

B) Implementar la rutina de servicio ISR(TIMER3_COMPA_vect){}

```
ISR(TIMER3_COMPA_vect){
// Rutina de servicio del Timer 3. Se ejecuta cada 10ms (100Hz)
// En esta ISR() se explora el display y el teclado
// Detecta pulsaciones, almacena códigos de tecla en un buffer y actualiza la
// variable de modo de funcionamiento, si procede.
// Secuencias de teclas:
// *#: Modo configuración
// #*: Modo visualización
}
```

C) En loop(): Mientras se esté en modo configuración, ejecutar las tareas asociadas. Cuando se abandona el modo configuración se ha de enviar un mensaje (… fin de configuración) y se pasará al modo de visualización.

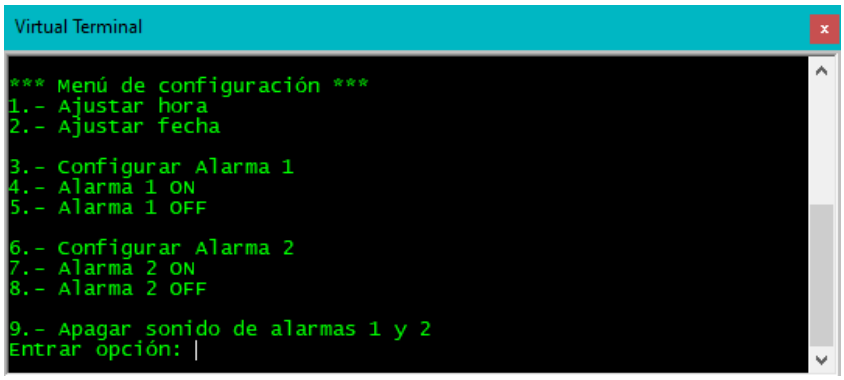
La entrada en modo de configuración, hace que se visualice en la pantalla de Proteus (virtual terminal) el menú de configuración, con el que se podrá interactuar, para ajustar la fecha y la hora del RTC DS3232.

Siéntase libre para diseñar este interfaz de usuario basado en menús.

Los cambios que se realicen en el DS3232 se verán reflejados automáticamente en la pantalla LCD gracias a la sección de actualización de información de la Tarea 1 que sabemos está asociada a una interrupción del Timer 1.

5.2.3 Parte 3: Configuración de alarmas

En esta tarea, se añaden nuevas funcionalidades a las establecidas en la Tarea 2. Consisten en añadir una nueva opción al menú para programar las alarmas 1 y 2. En la siguiente figura se muestra una posible implementación:



Cuando se produce una alarma también se generará una interrupción (previa habilitación en el DS3232) que será atendida por la correspondiente rutina de servicio ISR(). Cuando se dispare cualquiera de las alarmas se oirá un sonido particular de cada alarma (para poder distinguirlas) y que durará alrededor de 5-10 segundos, aproximadamente. La señal de petición de interrupción del chip DS3232 (INT'/SQW) se conectará a la interrupción externa INT0 (pin21) del Arduino Mega 2560. Una vez conectada, será necesario habilitar el DS3232 para que genere interrupciones por alarmas y habilitar la línea de interrupción externa INT0.

```
ISR(INT0_vect){
    // Gestión de las alarmas 1 y 2 del chip DS3232
    }
```

5.2.4 Mejoras de la aplicación (opcional)

En el apartado de mejoras puede implementar cualquier idea que se le ocurra y que mejore o amplíe la práctica base. Una posible mejora podría ser la visualización de publicidad o noticias de forma alternada con la información del reloj. Las noticias se pueden pedir al usuario a través del correspondiente menú y almacenarlas en la memoria 24LC64 con un algún tipo de flag asociado que indiquen si se visualiza o no. De esta forma, podríamos tener varias noticias almacenadas en memoria que se visualizarían de forma alternativa con el reloj en la pantalla LCD si el flag de visualización está activo o, incluso, que se visualicen dependiendo de la hora del día: imagine información de restaurantes o espectáculos que interesa que aparezcan en una franja horaria determinada.

6 Entrega de la 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 proyeto Proteus (*.pdsprj) de la aplicación con los programas adecuadamente comentados. En principio, solo será necesario subir un fichero cuyo nombre seguirá la siguiente nomenclatura:

23-24_plab4_inciales nombre y apellidos.pdsprj

Ejemplo:

23-24_plab4_amf.pdsprj

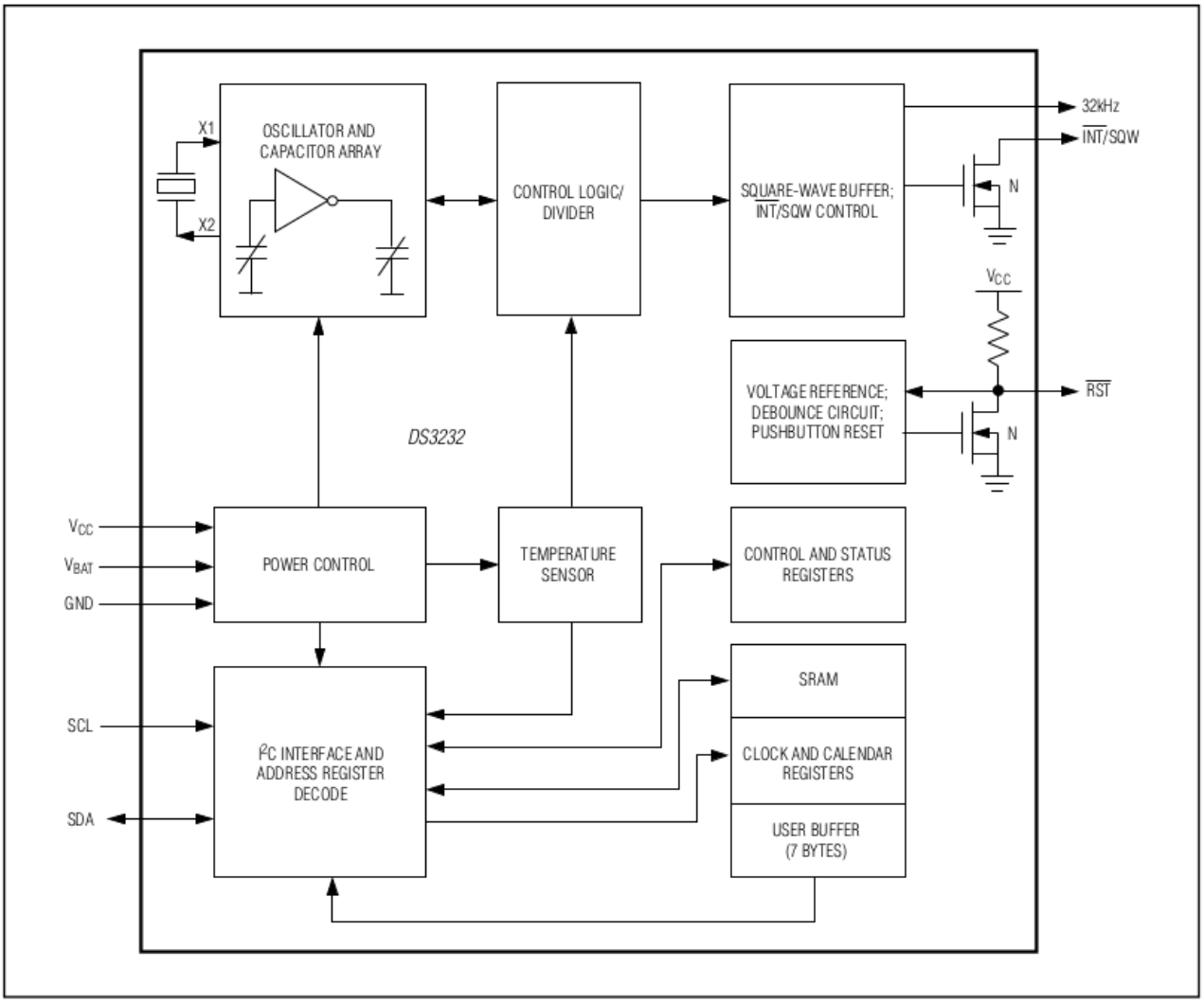
Fichero correspondiente al proyecto Proteus de la práctica 4 en laboratorio entregado por Anabel Medina Falcón.

7 Anexo: Resumen del Reloj de Tiempo Real (RTC) DS3232de la práctica

DS3232

Extremely Accurate I²C RTC with Integrated Crystal and SRAM

Block Diagram



Seguidamente se detallan los 256 registros/SRAM de 8 bits internos. La mecánica de funcionamiento es muy similar a la vista anteriormente. Tiene un puntero que hay que inicializar y que luego se autoincrementa tras la operación. Tiene un bloque relacionado con la hora y fecha, seguido de dos alarmas, dos registros de control/estado, dos registros para leer la temperatura interna y el resto de direcciones de SRAM. En la siguiente tabla se detallan los campos concretos:

Figure 1. Address Map for DS3232 Timekeeping Registers and SRAM

ADDRESS	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSB	FUNCTION	RANGE
00h	0	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Hours	1–12 + $\overline{AM}$/PM 00–23
03h	0	0	0	0	0	Day			Day	1–7
04h	0	0	10 Date		Date				Date	1–31
05h	Century	0	0	10 Month	Month				Month/ Century	01–12 + Century
06h	10 Year				Year				Year	00–99
07h	A1M1	10 Seconds			Seconds				Alarm 1 Seconds	00–59
08h	A1M2	10 Minutes			Minutes				Alarm 1 Minutes	00–59
09h	A1M3	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Alarm 1 Hours	1–12 + $\overline{AM}$/PM 00–23
0Ah	A1M4	DY/$\overline{DT}$	10 Date		Day				Alarm 1 Day	1–7
					Date				Alarm 1 Date	1–31
0Bh	A2M2	10 Minutes			Minutes				Alarm 2 Minutes	00–59
0Ch	A2M3	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Alarm 2 Hours	1–12 + $\overline{AM}$/PM 00–23
0Dh	A2M4	DY/$\overline{DT}$	10 Date		Day				Alarm 2 Day	1–7
					Date				Alarm 2 Date	1–31
0Eh	$\overline{EOSC}$	BBSQW	CONV	RS2	RS1	INTCN	A2IE	A1IE	Control	—
0Fh	OSF	BB32kHz	CRATE1	CRATE0	EN32kHz	BSY	A2F	A1F	Control/Status	—
10h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	Aging Offset	—
11h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	MSB of Temp	—
12h	DATA	DATA	0	0	0	0	0	0	LSB of Temp	—
13h	0	0	0	0	0	0	0	0	Not used	Reserved for test
14h–0FFh	x	x	x	x	x	x	x	x	SRAM	00h–0FFh

Note: *Unless otherwise specified, the registers' state is not defined when power is first applied.*

Table 2. Alarm Mask Bits

DY/$\overline{DT}$	ALARM 1 REGISTER MASK BITS (BIT 7)				ALARM RATE
	A1M4	A1M3	A1M2	A1M1	
X	1	1	1	1	Alarm once per second
X	1	1	1	0	Alarm when seconds match
X	1	1	0	0	Alarm when minutes and seconds match
X	1	0	0	0	Alarm when hours, minutes, and seconds match
0	0	0	0	0	Alarm when date, hours, minutes, and seconds match
1	0	0	0	0	Alarm when day, hours, minutes, and seconds match

DY/$\overline{DT}$	ALARM 2 REGISTER MASK BITS (BIT 7)			ALARM RATE
	A2M4	A2M3	A2M2	
X	1	1	1	Alarm once per minute (00 seconds of every minute)
X	1	1	0	Alarm when minutes match
X	1	0	0	Alarm when hours and minutes match
0	0	0	0	Alarm when date, hours, and minutes match
1	0	0	0	Alarm when day, hours, and minutes match

Control Register (0Eh)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	$\overline{EOSC}$	BBSQW	CONV	RS2	RS1	INTCN	A2IE	A1IE
POR*:	0	0	0	1	1	1	0	0

**POR is defined as the first application of power to the device, either V_{BAT} or V_{CC}.*

Special-Purpose Registers

The DS3232 has two additional registers (control and control/status) that control the real-time clock, alarms, and square-wave output.

Control Register (0Eh)

Bit 7: Enable Oscillator ($\overline{EOSC}$). When set to logic 0, the oscillator is started. When set to logic 1, the oscillator is stopped when the DS3232 switches to battery power. This bit is clear (logic 0) when power is first applied. When the DS3232 is powered by V_{CC}, the oscillator is always on regardless of the status of the $\overline{EOSC}$ bit. When $\overline{EOSC}$ is disabled, all register data is static.

Bit 6: Battery-Backed Square-Wave Enable (BBSQW). When set to logic 1 with INTCN = 0 and V_{CC} < V_{PF}, this bit enables the square wave. When BBSQW is logic 0, the $\overline{INT}$/SQW pin goes high impedance when V_{CC} < V_{PF}. This bit is disabled (logic 0) when power is first applied.

Bit 5: Convert Temperature (CONV). Setting this bit to 1 forces the temperature sensor to convert the temperature into digital code and execute the TCXO algorithm to update the capacitance array to the oscillator. This can only happen when a conversion is not already in progress. The user should check the status bit BSY before forcing the controller to start a new TCXO execution. A user-initiated temperature conversion does not affect the internal 64-second (default interval) update cycle.

A user-initiated temperature conversion does not affect the BSY bit for approximately 2ms. The CONV bit remains at a 1 from the time it is written until the conversion is finished, at which time both CONV and BSY go to 0. The CONV bit should be used when monitoring the status of a user-initiated conversion.

Bits 4 and 3: Rate Select (RS2 and RS1). These bits control the frequency of the square-wave output when the square wave has been enabled. The following table shows the square-wave frequencies that can be selected with the RS bits. These bits are both set to logic 1 (8.192kHz) when power is first applied.

SQUARE-WAVE OUTPUT FREQUENCY

RS2	RS1	SQUARE-WAVE OUTPUT FREQUENCY
0	0	1Hz
0	1	1.024kHz
1	0	4.096kHz
1	1	8.192kHz

Bit 2: Interrupt Control (INTCN). This bit controls the $\overline{INT}$/SQW signal. When the INTCN bit is set to logic 0, a square wave is output on the $\overline{INT}$/SQW pin. When the INTCN bit is set to logic 1, a match between the timekeeping registers and either of the alarm registers activates the $\overline{INT}$/SQW (if the alarm is also enabled). The corresponding alarm flag is always set regardless of the state of the INTCN bit. The INTCN bit is set to logic 1 when power is first applied.

Bit 1: Alarm 2 Interrupt Enable (A2IE). When set to logic 1, this bit permits the alarm 2 flag (A2F) bit in the status register to assert $\overline{INT}$/SQW (when INTCN = 1). When the A2IE bit is set to logic 0 or INTCN is set to logic 0, the A2F bit does not initiate an interrupt signal. The A2IE bit is disabled (logic 0) when power is first applied.

Bit 0: Alarm 1 Interrupt Enable (A1IE). When set to logic 1, this bit permits the alarm 1 flag (A1F) bit in the status register to assert $\overline{INT}$/SQW (when INTCN = 1). When the A1IE bit is set to logic 0 or INTCN is set to logic 0, the A1F bit does not initiate the $\overline{INT}$/SQW signal. The A1IE bit is disabled (logic 0) when power is first applied.

Control/Status Register (0Fh)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	OSF	BB32kHz	CRATE1	CRATE0	EN32kHz	BSY	A2F	A1F
POR*:	1	1	0	0	1	0	0	0

**POR is defined as the first application of power to the device, either V_{BAT} or V_{CC}.*

Control/Status Register (0Fh)

Bit 7: Oscillator Stop Flag (OSF). A logic 1 in this bit indicates that the oscillator either is stopped or was stopped for some period and may be used to judge the validity of the timekeeping data. This bit is set to logic 1 any time that the oscillator stops. The following are examples of conditions that can cause the OSF bit to be set:

1) The first time power is applied.
2) The voltages present on both V_{CC} and V_{BAT} are insufficient to support oscillation.
3) The $\overline{EOSC}$ bit is turned off in battery-backed mode.
4) External influences on the crystal (i.e., noise, leakage, etc.).

This bit remains at logic 1 until written to logic 0.

Bit 6: Battery-Backed 32kHz Output (BB32kHz). This bit enables the 32kHz output when powered from V_{BAT} (provided EN32kHz is enabled). If BB32kHz = 0, the 32kHz output is low when the part is powered by V_{BAT}.

Bits 5 and 4: Conversion Rate (CRATE1 and CRATE0). These two bits control the sample rate of the TCXO. The sample rate determines how often the temperature sensor makes a conversion and applies compensation to the oscillator. Decreasing the sample rate decreases the overall power consumption by decreasing the frequency at which the temperature sensor operates. However, significant temperature changes that occur between samples may not be completely compensated for, which reduce overall accuracy. When a new conversion rate is written to the register, it may take up to the new conversion rate time before the conversions occur at the new rate.

CRATE1	CRATE0	SAMPLE RATE (seconds)
0	0	64
0	1	128
1	0	256
1	1	512

Bit 3: Enable 32kHz Output (EN32kHz). This bit indicates the status of the 32kHz pin. When set to logic 1, the 32kHz pin is enabled and outputs a 32.768kHz square-wave signal. When set to logic 0, the 32kHz pin goes low. The initial power-up state of this bit is logic 1, and a 32.768kHz square-wave signal appears at the 32kHz pin after a power source is applied to the DS3232 (if the oscillator is running).

Bit 2: Busy (BSY). This bit indicates the device is busy executing TCXO functions. It goes to logic 1 when the conversion signal to the temperature sensor is asserted and then is cleared when the conversion is complete.

Bit 1: Alarm 2 Flag (A2F). A logic 1 in the alarm 2 flag bit indicates that the time matched the alarm 2 registers. If the A2IE bit is logic 1 and the INTCN bit is set to logic 1, the $\overline{INT}$/SQW pin is also asserted. A2F is cleared when written to logic 0. This bit can only be written to logic 0. Attempting to write to logic 1 leaves the value unchanged.

Bit 0: Alarm 1 Flag (A1F). A logic 1 in the alarm 1 flag bit indicates that the time matched the alarm 1 registers. If the A1IE bit is logic 1 and the INTCN bit is set to logic 1, the $\overline{INT}$/SQW pin is also asserted. A1F is cleared when written to logic 0. This bit can only be written to logic 0. Attempting to write to logic 1 leaves the value unchanged.

Aging Offset (10h)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA
POR*:	0	0	0	0	0	0	0	0

No es necesario modificar nada para conseguir la exactitud normal.

Temperature Register (Upper Byte) (11h)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA
POR*:	0	0	0	0	0	0	0	0

Parte entera de la Temperatura (8bits con signo).

Temperature Register (Lower Byte) (12h)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	DATA	DATA	0	0	0	0	0	0
POR*:	0	0	0	0	0	0	0	0

Parte fraccionaria de la Temperatura. La precisión máxima es de 0,25º Centígrados, por lo tanto los bits 6 y 7 codifican las 4 posibilidades de fracción:

Bit7	Bit6	Temperatura
Dato	Dato	Fracción ºC
0	0	0,00
0	1	0,25
1	0	0,50
1	1	0,75

SRAM (14h–FFh)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	D7	D6	D5	D4	D3	D2	D1	D0
POR*:	X	X	X	X	X	X	X	X

**POR is defined as the first application of power to the device, either V_{BAT} or V_{CC}.*

SRAM

The DS3232 provides 236 bytes of general-purpose battery-backed read/write memory. The I^2C address ranges from 14h to 0FFh. The SRAM can be written or read whenever V_{CC} or V_{BAT} is greater than the minimum operating voltage.

La **"Slave Address" es "1101 000"**. La escritura y lectura es similar a las ya explicadas; la única diferencia destacable es , que al ser solo 256 registros, la dirección del puntero es de 8 bits (word address) y por lo tanto se tiene que enviar un solo byte en lugar de dos bytes que es necesario enviar el caso de la memoria 24LC64, ya que esta es de 8Kbytes de tamaño. También, en la escritura no hay buffer intermedio, y por lo tanto es mucho más simple ya que no se producirán ciclos de espera por motivo de los retardos de escritura.

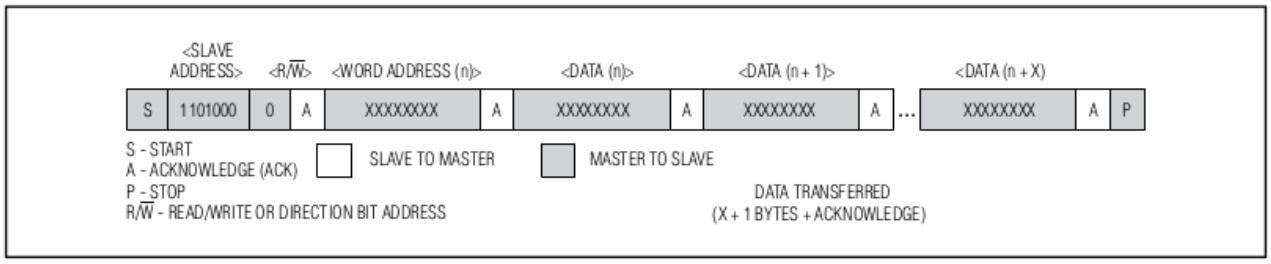


Figure 3. Data Write—Slave Receiver Mode

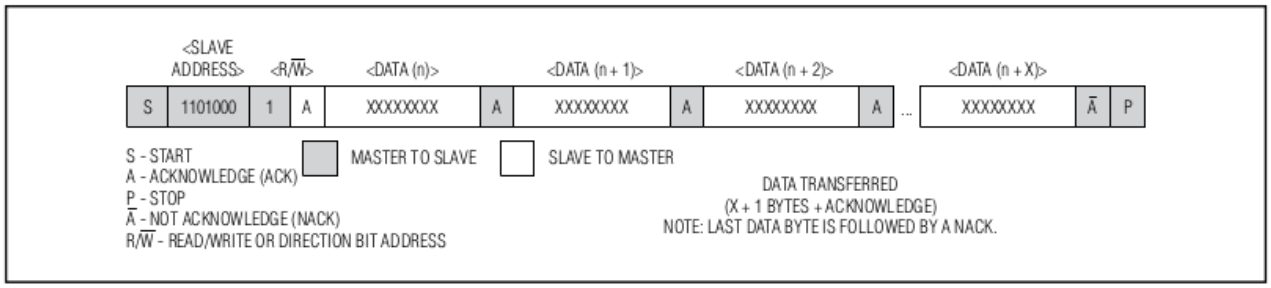


Figure 4. Data Read—Slave Transmitter Mode

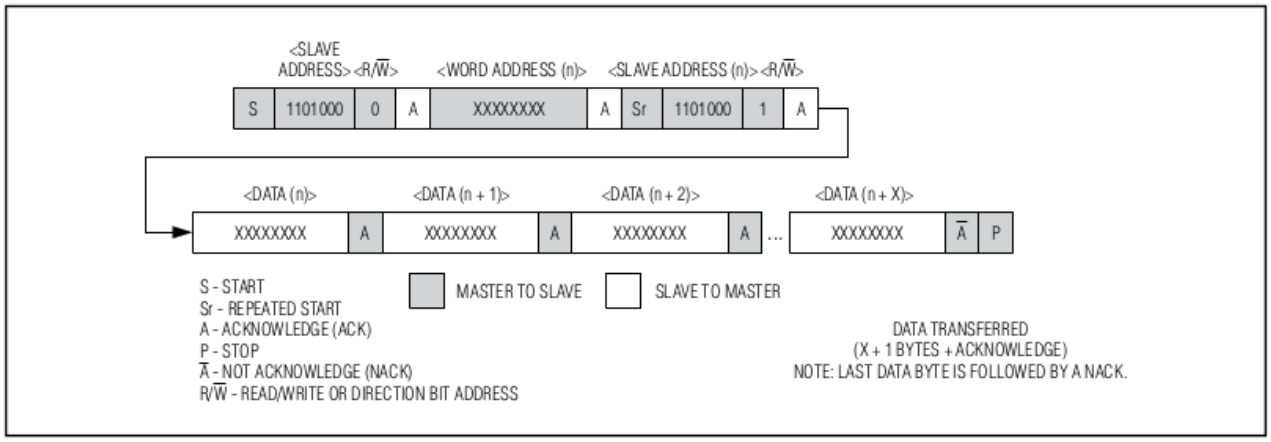


Figure 5. Data Write/Read (Write Pointer, Then Read)—Slave Receive and Transmit