Trabajo Práctico N°02

Implementación de un reloj con fecha y hora editables

Circuitos Digitales y Microcontroladores – UNLP

NOTA 10

CALDERÓN Sergio Leandro, ERCOLI Juan Martín

23 de mayo de 2022

Índice

0 – Consigna general

Implementar con el MCU un reloj con fecha y hora como el mostrado en la Fig. 1. Para esto se dispone de un display LCD de 2 líneas, un teclado matricial 4x4 y el Atmega328p. La implementación deberá hacerse con máquinas de estados finitos temporizadas con Timer.

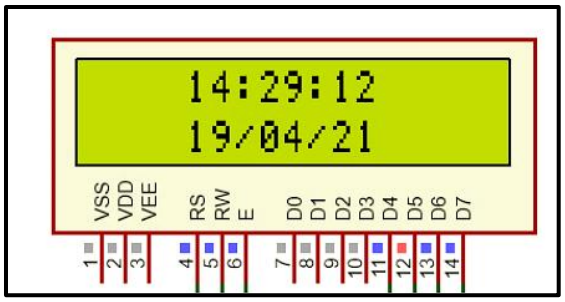

Fig. 1. Visualización esperada para el presente trabajo.

En los apartados 2 al 5 se explican cada uno de los requerimientos pedidos.

1 – Componentes

1.1 – LCD

Para la visualización de la fecha y hora se utilizó un LCD de 2 líneas configurado en modo 4 bits. El manejo del display se realiza por medio de la biblioteca de funciones provista por la cátedra (ver apéndice A, archivo LCD.h).

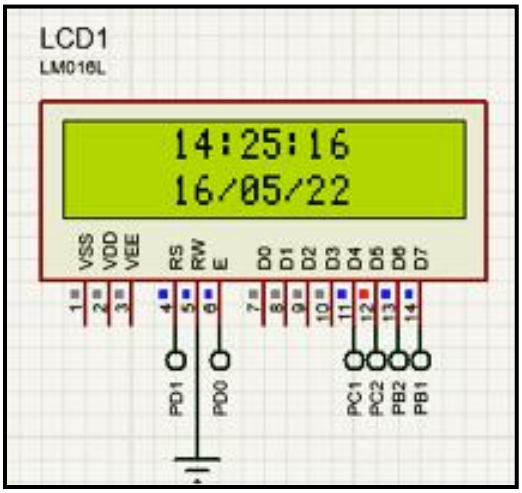

Fig 1.1. Conexionado del LCD para su manipulación desde el MCU.

Las conexiones del display se encuentran hechas para manejarlo con 4 lineas de datos y 2 de control, ya que el pin RW se conecta a tierra (solo escritura).

Tabla 1.1. Configuración de pines del LCD.

PIN	Conectado a	Cumple la función de
V_{SS} y V_{DD}	Alimentación de +5V	Alimentar al LCD.
RS	PD1	Register select (control).
RW	Tierra	RW = 0 → LCD Writable. RW = 1 → LCD Readable.
E	PD0	RW = 0 → Aplicando un flanco HIGH to LOW a E se escribe el dato que está en las líneas de datos. RW = 1 → Aplicando un flanco HIGH to LOW a E el LCD entrega los datos en las líneas de datos. (Control).
D7, D6, D5, D4	PB1, PB2, PC2, PC1	Líneas de datos.

1.2 – Teclado matricial

Para controlar la edición de la fecha y hora se utilizó un teclado matricial 4x4 cuya apariencia en Proteus fue editada para que sea semejante al teclado que se utiliza en el kit de la cátedra. Su implementación interna para la lectura de teclas se explica en la sección 3.3.

Tabla 1.2. Configuración de pines del teclado matricial.

PIN	Conectado a
Filas A, B, C, D	PB4, PB3, PB0, PD7
Columnas 1, 2, 3, 4	PD3, PD5, PD4, PD2

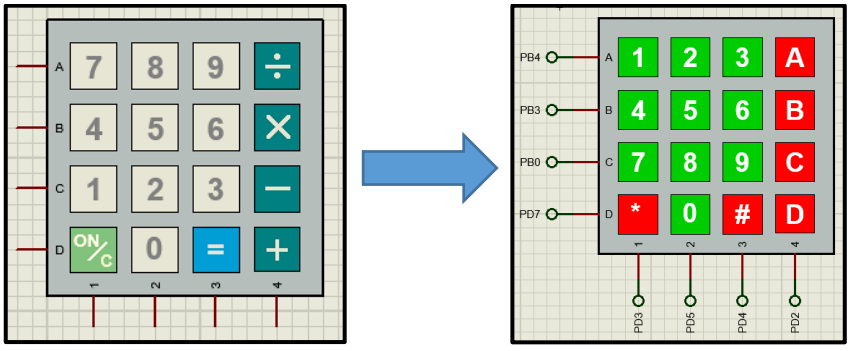

Fig. 1.2. Cambio de apariencia del componente "KEYPAD-SMALLCALC" en Proteus.

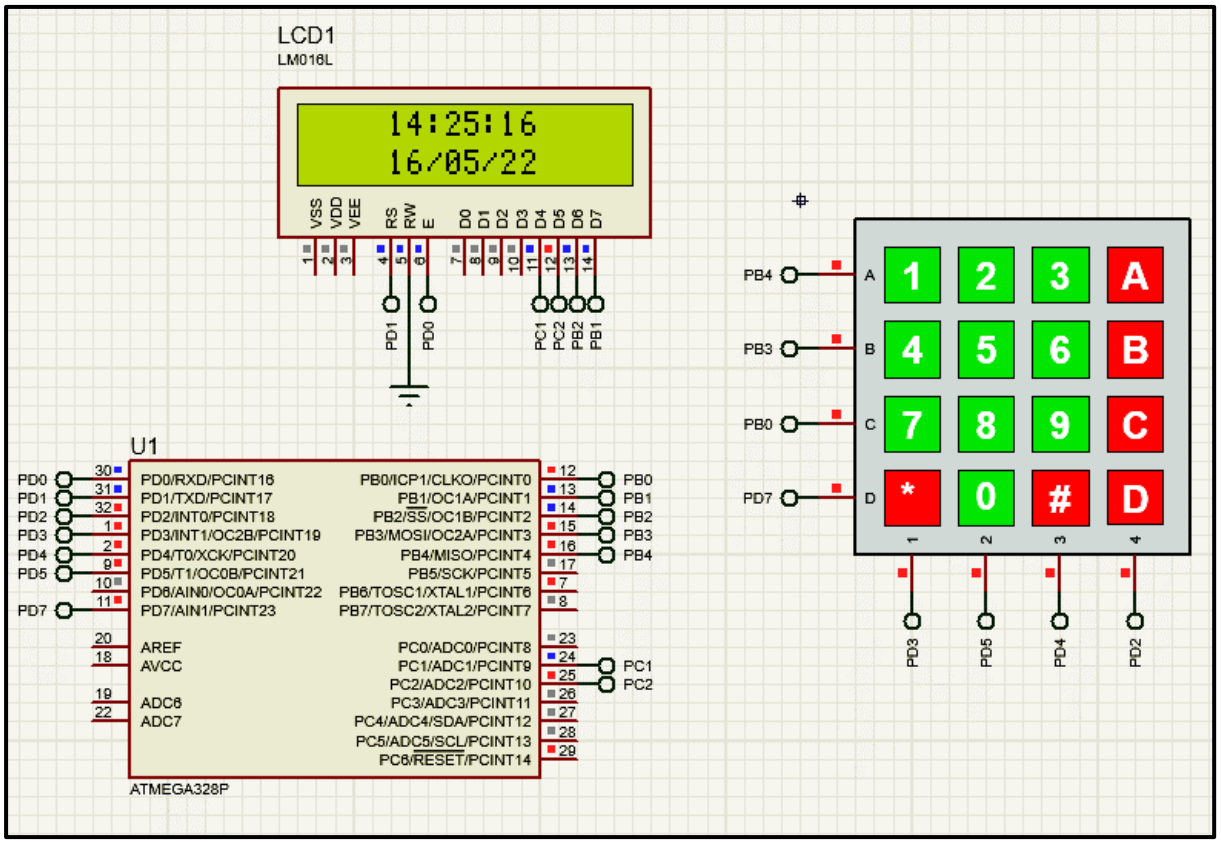

Fig. 1.3. Conexionado resultante entre MCU, LCD y teclado matricial.

2 – Visualización por defecto

2.1 – Enunciado

En el estado por defecto se deberá mostrar en la primera línea del LCD un reloj funcionando con el formato HH:MM:SS (horas, minutos y segundos) y en la segunda línea la fecha en formato DD/MM/AA (día, mes y año). El valor inicial del reloj se establece en tiempo de compilación por ejemplo 23:59:59 de 31/12/21.

2.2 – Interpretación

Se pide implementar un reloj y visualizar la fecha y hora del mismo en el display del LCD. En concreto, la hora deberá mostrarse en la primera línea con el formato HH:MM:SS, siendo HH la hora, MM los minutos y SS los segundos actuales. Por otra parte, la fecha, compuesta por el día, mes y año, deberá mostrarse en la segunda línea con sus parámetros separados por barras (formato DD/MM/AA).

El reloj deberá inicializarse en un valor de tiempo prestablecido en el código fuente.

2.3 – Resolución

2.3.1 – Estructura de datos

Para la implementación interna del reloj, se definió una estructura "Tiempo" que posee 6 campos tipo entero sin signo para almacenar cada uno de los parámetros a mostrar.

```
typedef struct {
    uint8_t anio;
    uint8_t mes;
    uint8_t dia;
    uint8_t hora;
    uint8_t minuto;
    uint8_t segundo;
} Tiempo;
```

Fig. 2.1. Estructura "Tiempo" definida para el reloj.

Para asignar valores a los campos se declaró una variable de este tipo, la cual se encuentra inicializada por defecto en una fecha y hora particular para posteriores validaciones.

2.3.2 – Rangos de validez

Cuando se actualiza el tiempo es necesario verificar que cada parámetro modificado se encuentre en su rango de validez correspondiente. A continuación, se indican los mismos:

Tabla 2.1. Valores mínimos y máximos permitidos de cada campo de Tiempo.

Campo	Mínimo	Máximo
segundo	0	59
minuto	0	59
hora	0	23
dia	1	28, 29, 30 o 31 (según mes y año)[1]
mes	1	12
anio	0	199

Si al incrementar un parámetro se supera el valor máximo, se debe resetear dicho parámetro a su valor mínimo e incrementar el parámetro de siguiente orden en 1, pudiéndose repetir este proceso hasta el campo de año, el cual al alcanzar el valor 100 se reinicia a 0.

[1] El máximo es 28 si el mes es febrero y no es un año bisiesto, 29 si es febrero y año bisiesto, 30 si el mes es abril, junio, septiembre o noviembre, 31 para los demás meses.

A continuación, se muestra el pseudocódigo correspondiente a esta función, la cual debe ser invocada cada vez que transcurre 1 segundo de tiempo:

```
Incrementar campo de segundos
Si se alcanzaron los 60 segundos
    Reiniciar segundos a cero
    Incrementar campo de minutos
    Si se alcanzaron los 60 minutos
        Reiniciar minutos a cero
        Incrementar campo de horas
        Si se alcanzaron las 24 horas
            Reiniciar horas a cero
            Incrementar campo de días
            Si se superó el día correspondiente al mes y año
                Reiniciar el campo día a 1
                Incrementar campo de meses
                Si se alcanzaron los 13 meses
                    Reiniciar el campo mes a 1
                    Incrementar campo de años
                    Si se alcanzaron los 200 años
                        Reiniciar años a cero
```

Pseudocódigo 2.1. Función privada *RELOJ_IncSecond()* con rangos controlados.

2.3.3 – Configuración del Timer

La unidad mínima de tiempo utilizada en la estructura homónima es el segundo. La actualización del reloj debe poder producirse de forma periódica cada dicho intervalo de tiempo, a partir de interrupciones producidas por un Timer.

Sin embargo, para otros requerimientos posteriores se necesita poder realizar lecturas de teclado también de forma periódica, por ende, el período de interrupción debe ser menor, del orden de los milisegundos. Conociendo que la frecuencia base del microcontrolador del kit es 16 MHz, se puede calcular cuáles son las frecuencias de Prescaler disponibles para usar.

Tabla 2.2. Prescalers disponibles para Timer 0 y 1 (págs. 87 y 110, Cap. 14-15)

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(no prescaling)
0	1	0	$clk_{I/O}$/8 (from prescaler)
0	1	1	$clk_{I/O}$/64 (from prescaler)
1	0	0	$clk_{I/O}$/256 (from prescaler)
1	0	1	$clk_{I/O}$/1024 (from prescaler)

Tabla 2.3. Prescalers disponibles para Timer 2 (pág. 131, Cap. 17)

CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk_{T2S}/(no prescaling)
0	1	0	clk_{T2S}/8 (from prescaler)
0	1	1	clk_{T2S}/32 (from prescaler)
1	0	0	clk_{T2S}/64 (from prescaler)
1	0	1	clk_{T2S}/128 (from prescaler)
1	1	0	clk_{T2S}/256 (from prescaler)
1	1	1	clk_{T2S}/1024 (from prescaler)

Como se puede observar en las Tablas 2.2 y 2.3, la frecuencia se puede dividir hasta por N = 1024. En dicho caso, para una frecuencia del sistema de 16 MHz, resulta una frecuencia mínima de Prescaler de 15625 Hz, es decir, un período máximo de 64 μs (T = 1/f).

Por otra parte, cada Timer tiene asociado un registro contador (TCNTn) y otro de comparación (OCRnx), ambos de 8 bits para Timer 0 y 2, y de 16 bits para Timer 1. El contador se actualiza en cada ciclo de la señal de reloj del Prescaler. Además, existen varios modos de funcionamiento del Timer, de los cuales destacamos dos:

- Normal: produce interrupción cuando el registro contador desborda.
- Modo CTC: produce interrupción cuando el contador coincide con el de comparación.

El transcurso de un segundo completo ocurriría luego de 15625 pulsos de la frecuencia de Prescaler, ya que dicha frecuencia es justamente 15625 Hz. Sin embargo, también se deberán realizar lecturas del teclado matricial, generalmente en el orden de los milisegundos. Por tal motivo, si se desea mantener la exactitud del reloj, se deberá elegir un submúltiplo de 15625:

Tabla 2.4. Posibilidades de conteo para un segundo exacto.

Conteo hasta	Tiempo transcurrido	Bits necesarios
3125	0,2 s = 200 ms	16
625	0,04 s = 40 ms	16
125	0,008 s = 8 ms	8

Se decidió que la interrupción se realice cada 40 ms, de modo que el incremento de segundo del reloj sea cada 25 interrupciones. Timer 1 es el único de los 3 presentados que posee registros de 16 bits para el conteo y comparación. Además, como $625 < 2^{16}$, el modo de funcionamiento debe ser CTC, si se inicializa el registro contador TCNT1 en cero.

Otra posibilidad sería producir interrupciones cada 8 ms con Timer 0 en modo CTC, pero como se verá más adelante en la simulación (sección 2.4), este tiempo NO es suficiente para realizar ciertas tareas de forma temporizada: impresión de la fecha y hora completa.

Tabla 2.5. Elección del modo de funcionamiento para Timer 1.

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, phase correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, phase correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, phase correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP

Como se observa en la Tabla 2.5, para establecer el modo CTC debe encontrarse activo el bit WGM12, e inactivos los bits WGM10, WGM11 y WGM13. Luego, de manera análoga que en Timer 0, según la Tabla 2.3, para un Prescaler de N=1024, deben activarse los bits CS10 y CS12, y desactivarse el bit CS11. Todos estos bits se encuentran en el registro TCCR1B:

```
TCCR1B = (1<<CS10) | (1<<CS12) | (1<<WGM12);
```

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Fig. 2.5. Bits del registro TCCR1B (Timer/Counter Control Register B)

En concordancia con el planteo anterior, en la inicialización debe reiniciarse el contador y asignarse el valor de comparación: **TCNT1 = 0; OCR1A = 624;**

Respecto a las interrupciones, se identifican 3 para el Timer. Cada una se puede habilitar o desactivar estableciendo en 1 el bit correspondiente en el registro TIMSK1:

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Fig. 2.6. Bits del registro TIMSK1 (Timer Interrupt Mask Register)

Para el modo de funcionamiento CTC utilizando al registro OCR1A para la comparación debe habilitarse el bit OCIE1A: **TIMSK1 |= (1<<OCIE1A);**

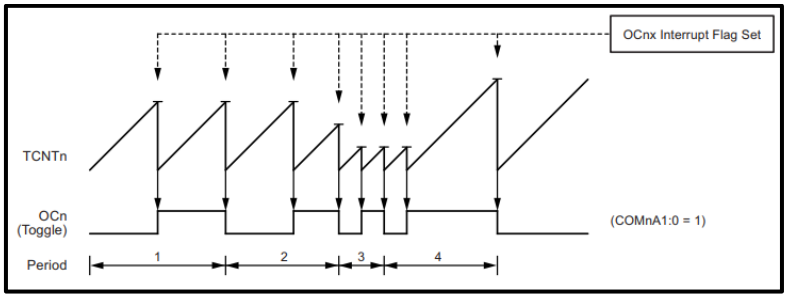

Fig. 2.7. Evolución en el tiempo de TCNTn y OCn en el modo CTC.

Por último, deben habilitarse las interrupciones globales mediante la función *sei()*. A modo de resumen, se muestra el pseudocódigo de inicialización de Timer 1 a continuación:

Deshabilitar todas las interrupciones en TIMSK1

Establecer CTC como modo de funcionamiento en TCCR1B

Establecer Prescaler de N = 1024 en TCCR1B

Reiniciar a cero el registro contador TCNT1

Asignar el valor 624 al registro OCR1A

Habilitar interrupción por Compare Match A en TIMSK1

Habilitar interrupciones globales

Pseudocódigo 2.2. Función pública *RELOJ_Init()* aplicada para Timer 1.

2.3.4 – Manejo de la interrupción

De acuerdo a la información provista por el fabricante, para ATmega328P se tiene un vector de interrupciones, en el cual la interrupción habilitada en la sección anterior (Timer 1 compare match A) se corresponde a la posición 12, identificada como TIMER1 COMPA.

Tabla 2.6. Vector de interrupciones del ATmega328P (pág. 49, Cap. 11)

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External pin, power-on reset, brown-out reset and watchdog system reset
2	0x002	INT0	External interrupt request 0
3	0x0004	INT1	External interrupt request 1
4	0x0006	PCINT0	Pin change interrupt request 0
5	0x0008	PCINT1	Pin change interrupt request 1
6	0x000A	PCINT2	Pin change interrupt request 2
7	0x000C	WDT	Watchdog time-out interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 compare match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 compare match B
10	0x0012	TIMER2 OVF	Timer/Counter2 overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 capture event
12	0x0016	TIMER1 COMPA	Timer/Counter1 compare match A
13	0x0018	TIMER1 COMPB	Timer/Counter1 compare match B
14	0x001A	TIMER1 OVF	Timer/Counter1 overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 compare match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 compare match B
17	0x0020	TIMER0 OVF	Timer/Counter0 overflow

En consecuencia, en el archivo del programa principal se debe indicar el código a ejecutar como rutina de interrupción para dicha fuente: **ISR(TIMER1_COMPA_vect) { }**

De manera periódica, cada 40 ms, se invoca la rutina mencionada, en la cual únicamente se activa un FLAG_TIMER que se consulta en el programa principal luego de haberse inicializado todos los componentes, para realizar acciones de forma temporizada, reiniciar el Flag propiamente dicho y suspender el uso de CPU hasta la próxima iteración.

En este caso, la acción temporizada es actualizar una máquina de estados (explicado más adelante) e incrementar un contador auxiliar de milisegundos del reloj de a 40 unidades, de modo que al alcanzar los 1000 ms se actualicen los campos de la estructura de Tiempo (visto en el pseudocódigo 2.1) y se reinicie a cero dicho contador auxiliar de 16 bits.

```
Sumar X al contador "ms"
Si "ms" es mayor o igual a 1000
    Incrementar 1 segundo de Tiempo
    Asignar a "ms" el resto de dividirlo por 1000
```

Pseudocódigo 2.3. Función pública *RELOJ_IncMs*. El valor X=40 se pasa como parámetro.

2.3.5 – Estado por defecto

Los campos de la estructura Tiempo deben visualizarse en el display LCD. Esto se realiza en el estado “DEFAULT” de una máquina de estados finitos (MEF) temporizada, que es actualizada cada 40 ms como se mencionó anteriormente. Sin embargo, el mensaje mostrado por pantalla (reloj) sólo es modificado cada 1 segundo, por ende, deben enviarse los caracteres a escribir cada dicho intervalo de tiempo, involucrando otro contador auxiliar.

Para la visualización, primero se obtiene una copia de la variable de Tiempo almacenada en el módulo de Reloj mediante una función pública *RELOJ_getCurrent*. Luego, se posiciona el cursor del LCD en la primera fila y se escriben la hora, minuto y segundo, separados por ‘:’ usando 2 caracteres para cada uno. Por último, se posiciona el cursor en la segunda fila y se escriben el día, mes y año, separados por ‘/’ usando nuevamente 2 caracteres para cada campo. Al escribir datos o caracteres, el cursor se desplaza automáticamente.

Guardar referencia de tiempo actual en “aux”

Ir a la posición (4,0) del LCD

Escribir dato aux.hora con 2 caracteres

Enviar carácter ‘:’ al LCD

Escribir dato aux.minuto con 2 caracteres

Enviar carácter ‘:’ al LCD

Escribir dato aux.segundo con 2 caracteres

Ir a la posición (4,1) del LCD

Escribir dato aux.dia con 2 caracteres

Enviar carácter ‘/’ al LCD

Escribir dato aux.mes con 2 caracteres

Enviar carácter ‘/’ al LCD

Escribir dato aux.anio con 2 caracteres

Pseudocódigo 2.4. Función privada *mostrarFechaHora()* de la MEF.

2.4 – Simulación

El reloj se inicializa con la fecha 31/12/2021 y hora 23:59:50, es decir, próximo al cambio de año, para poder verificar que los parámetros se actualizan de manera correcta.

La temporización se comprueba mediante el uso de *breakpoints*. A continuación, se adjuntan capturas de la simulación realizada en Proteus, usando una frecuencia de 16 MHz.

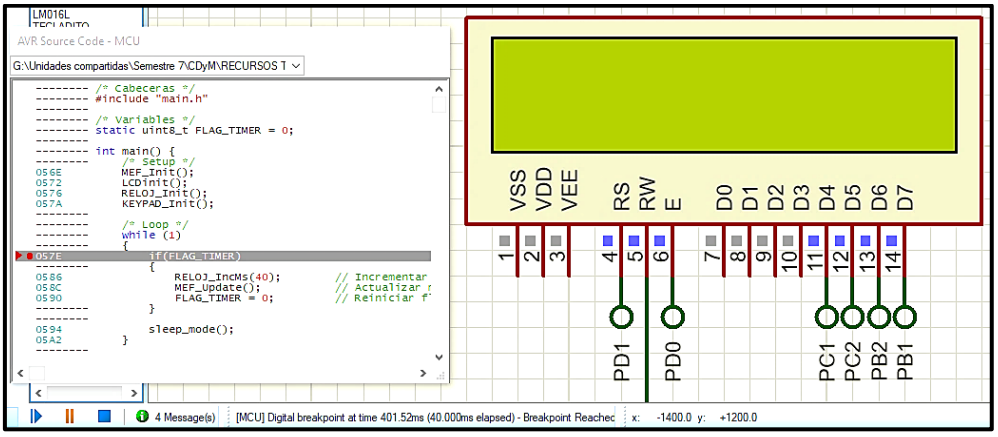

Fig. 2.8. Medición del período de interrupción. Verifica 40 ms = (OCR1A+1) ∗ 64 μs

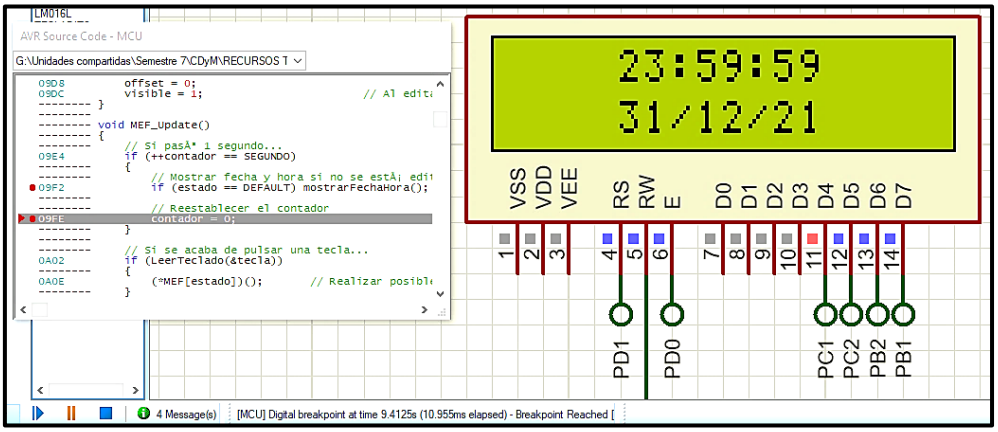

Fig. 2.9. Medición del tiempo de escritura en pantalla, aprox. 11 ms, mayor a 8 ms.

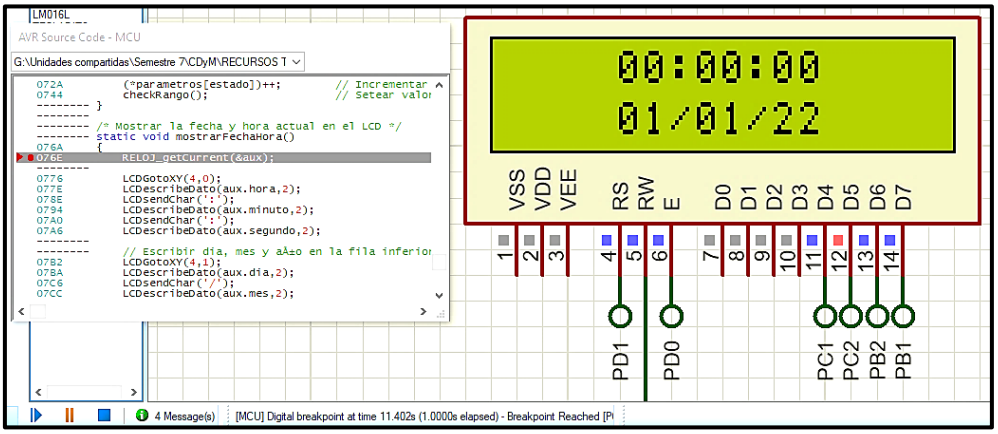

Fig. 2.10. Medición del tiempo de visualización. Verifica 1000 ms = 25 x 40 ms, y rangos válidos.

3 – Etapas de modificación

3.1 – Enunciado

Para modificar los datos (fecha y hora) se deberá presionar 'A'. Para cancelar y no aceptar la modificación se deberá presionar 'D' en cuyo caso se volverá al estado por defecto.

El proceso de modificación es secuencial, empezando por el año y siguiendo con el mes, el día, luego la hora, minutos y por último segundos. Para pasar de un campo al siguiente se deberá presionar 'A'. Si es el campo de segundos, al presionar 'A' se establecerán los nuevos datos al sistema volviendo al estado por defecto.

3.2 – Interpretación

Se pide que, estando en el estado por defecto, al oprimir la tecla 'A' del teclado matricial se ingrese a un modo de configuración de los datos del reloj, es decir se debe permitir la modificación de la fecha y hora visibles en el display del LCD. La modificación de cada parámetro (campo) del reloj se hará de forma secuencial, comenzando por el año, luego el mes, siguiendo el día, a continuación, la hora, los minutos y por último los segundos. Se podrá pasar de un campo al siguiente presionando otra vez la tecla 'A' independientemente del campo en el que se encuentre. Se guardarán los datos modificados una vez que se presione la tecla 'A' en el campo de segundos, sobrescribiendo el tiempo actual y volviendo al estado por defecto.

Si se desea descartar los cambios ya realizados y la modificación de los parámetros restantes, se deberá presionar la tecla 'D' independientemente del campo en el que se encuentre, esto resolverá a que se vuelva al estado por defecto mostrando el tiempo actual, por lo que es necesario que el reloj siga actualizándose internamente para evitar su atraso. Se entiende además que cada etapa de la edición de campos se puede representar con un estado diferente.

3.3 – Resolución

3.3.1 – Identificación de tecla presionada

El teclado matricial está compuesto por 16 pulsadores, organizados en 4 filas y 4 columnas. Para identificar la posición de una tecla presionada, se configuran las filas como entradas con Pull-Up y, por otro lado, las columnas como salidas inicialmente desactivadas.

En cada encuesta de teclado, se prueba activando una columna por vez, y se lee la entrada asociada a cada fila para identificar si la tecla de dicha columna y fila está pulsada.

La implementación de la encuesta se realizó como una función del módulo Teclado, que devuelve si hubo tecla presionada o no (0 o 1), y la tecla en cuestión por referencia. Existe una matriz de caracteres para conocer cuál corresponde a cada posición (fila, columna).

```
Desactivar salidas escribiendo un 1 en c/u
Por cada columna "C" del teclado matricial
    Se activa la salida asociada escribiendo un 0
    Por cada fila "F" del teclado matricial
        Se lee el estado del pulsador de la columna C y fila F
        Si no se encuentra en reposo
            Pasar carácter asociado a la posición por referencia
            Retornar que hay tecla presionada (valor 1)
Retornar que no hay tecla presionada (valor 0)
```

Pseudocódigo 3.1. Función privada *KEYPAD_Scan* del teclado.

La función anterior indica si una tecla se encuentra presionada en el instante en que la encuesta es realizada. Sin embargo, al no contar con registro de la última tecla presionada no permitirá distinguir si se presionó nuevamente la tecla o se mantuvo la presionada; y tampoco cuenta con doble verificación por lo que no eliminará el efecto rebote de los pulsadores.

Para resolver las 2 problemáticas mencionadas, en el siguiente apartado se presenta una segunda función que utiliza a la anterior para encuestar al teclado y tiene variables auxiliares.

3.3.2 – Detección correcta de presión de tecla

Para realizar acciones sobre el reloj se debe detectar si una tecla ha sido presionada, siendo necesaria en cada actualización de la MEF temporizada realizada cada 40 ms. Las múltiples detecciones deben evitarse, por ejemplo, al presionar la tecla 'A' para avanzar al próximo campo a editar. A continuación, se presenta la función pública *KEYPAD_Update* del teclado, que tiene un parámetro por referencia "tecla" para conocer el valor de la tecla presionada y retorna 1 si está confirmado que se presionó una tecla, 0 en caso contrario.

Se utilizan 2 variables *static* en la función: tecla anterior y tecla confirmada.

```
Inicializar tecla anterior como "no válida".
Inicializar tecla confirmada como "no válida".

Encuestar teclado (KEYPAD Scan)
Si hay tecla presionada en este momento
    Si es una detección consecutiva de la tecla
        Si aún no se confirmó la tecla como presionada
            Pasar tecla actual por referencia
            Asignar tecla actual como confirmada
            Retornar que se presionó una tecla (valor 1)
    Sino
        Asignar tecla actual como anterior
Sino
    Asignar tecla anterior y confirmada como "no válida"
Retornar que no se presionó una tecla (valor 0)
```

Pseudocódigo 3.2. Función pública *KEYPAD_Update* del teclado.

Luego si se confirma la presión de una tecla, se realiza un posible cambio de estado de la MEF invocando la función de estado con un puntero a función.

3.3.3 – Estados de la MEF

Para representar el estado en que se encuentra el reloj se diseñó una máquina de estados de Mealy, cuyo estado predeterminado (DEFAULT) se ha explicado en la sección 2.3.5.

Se tienen 7 estados en total, de los cuales 6 corresponden a edición de parámetros. En una primera instancia se puede plantear la secuencia de la siguiente forma:

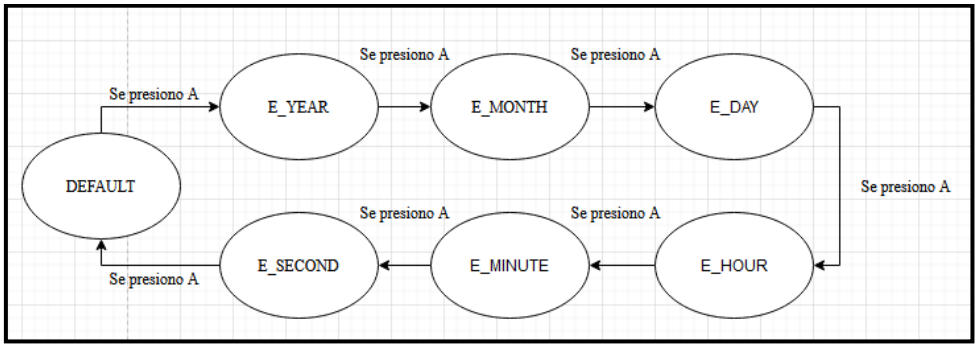

Figura 3.1. Secuencia ordenada de estados al presionar 'A'.

En cada actualización de la MEF, se realiza la invocación a la función de estado correspondiente por medio de un puntero a función. Es importante destacar que todas las ediciones sobre los campos se realizan sobre una variable "aux" y los cambios se escriben en la variable de tiempo del módulo Reloj, a través de una función *RELOJ_updateCurrent*, cuando la MEF se encuentre en el estado E_SECOND y se presione la tecla 'A'.

En el caso de presionar la tecla 'D', siempre se retornará al estado DEFAULT, y la variable "aux" será sobrescrita por una copia del tiempo actual del Reloj al momento de mostrarlo en el display del LCD, descartándose así cualquier cambio realizado.

4 – Edición controlada de parámetros

4.1 – Enunciado

Cada campo a modificar tendrá su rango de validez, por ejemplo, la hora tiene un rango de 0 a 23, los minutos y segundos de 0 a 59, el año de 0 a 99, los meses de 1 a 12. Los días dependen de cada mes. Para incrementar el valor actual de un campo se deberá presionar 'B' y para decrementarlo se deberá presionar 'C'.

4.2 – Interpretación

Al ingresar a la configuración de cada uno de los campos del reloj se deberá controlar que el mismo no exceda el rango de validez en concordancia a lo visto en el apartado 2.3.2.

Para aumentar en 1 el valor actual del campo se deberá presionar la tecla 'B', y para decrementarlo en 1 se deberá presionar la tecla 'C'. Si al incrementar se excede el rango permitido se debe establecer el parámetro en su valor mínimo; y si al decrementar el valor queda por debajo del mínimo permitido se establece dicho campo en su valor máximo.

4.3 – Resolución

4.3.1 – Entradas y salidas de la MEF

A partir de este nuevo requerimiento y el explicado en el apartado 3, se definieron y completaron todas las transiciones entre estados a realizar según el estado actual y la tecla que se haya presionado, como también las acciones para cada una (máquina de Mealy).

Las entradas en cada iteración son: 000 (no se presionó tecla), 001 (se presionó ‘A’), 010 (se presionó ‘B’), 011 (se presionó ‘C’) y 100 (se presionó ‘D’). Por otro lado, se tienen 4 salidas o eventos: 00 (sin cambios), 01 (incrementar), 10 (decrementar) y 11 (sobrescribir).

El diagrama de estados de la máquina se muestra a continuación:

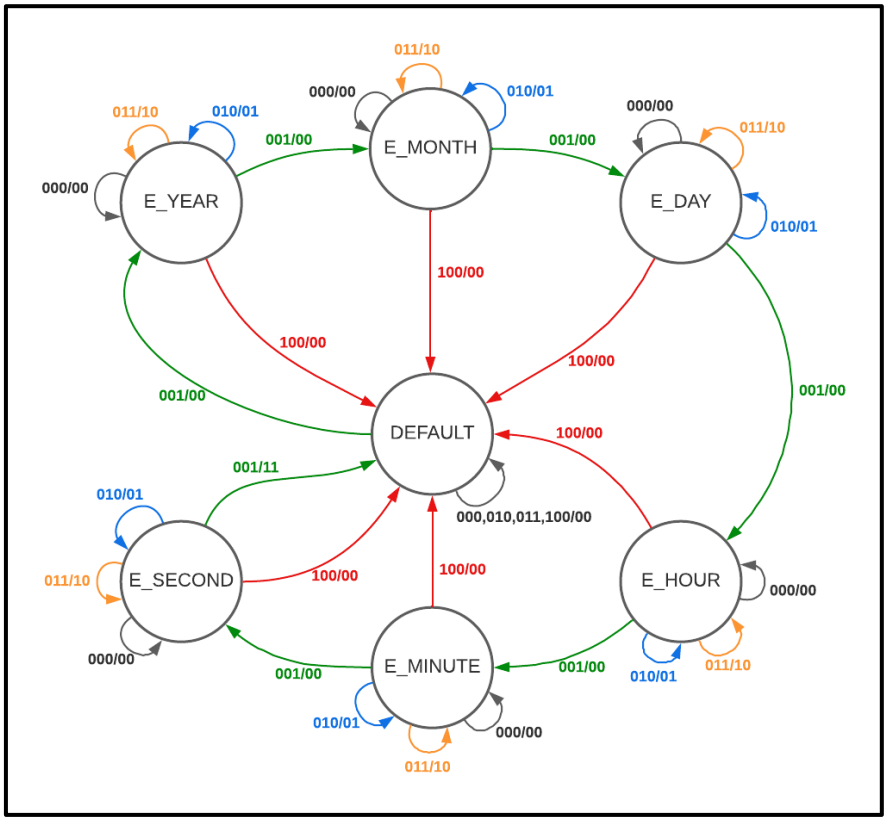

Fig. 4.1. Diagrama de estados de la MEF de Mealy.

Se decidió elegir MEF de Mealy para tener una menor cantidad de estados y porque las acciones de incrementar, decrementar y cancelar se pueden codificar de forma genérica aprovechando la conversión del estado a entero mediante enumerativos.

En cada función de estado se solicita al teclado que devuelva si hubo tecla presionada y cuál es su carácter en caso afirmativo usando la función explicada en el apartado 3.3.2, para luego realizar alguna de las acciones mencionadas, pudiendo cambiar o no de estado.

Tabla 4.1. Descripción de la MEF por tabla de transición de estados.

Estado / Entradas	000	001	010	011	100
DEFAULT	DEFAULT / 00	E_YEAR / 00	DEFAULT / 00	DEFAULT / 00	DEFAULT / 00
E_YEAR	E_YEAR / 00	E_MONTH / 00	E_YEAR / 01	E_YEAR / 10	DEFAULT / 00
E_MONTH	E_MONTH / 00	E_DAY / 00	E_MONTH / 01	E_MONTH / 10	DEFAULT / 00
E_DAY	E_DAY / 00	E_HOUR / 00	E_DAY / 01	E_DAY / 10	DEFAULT / 00
E_HOUR	E_HOUR / 00	E_MINUTE / 00	E_HOUR / 01	E_HOUR / 10	DEFAULT / 00
E_MINUTE	E_MINUTE / 00	E_SECOND / 00	E_MINUTE / 01	E_MINUTE / 10	DEFAULT / 00
E_SECOND	E_SECOND / 00	DEFAULT / 11	E_SECOND / 01	E_SECOND / 10	DEFAULT / 00

El detalle de implementación de cada función de estado se dará una vez contemplados todos los requerimientos, en el apartado 5 del presente informe.

4.3.2 – Chequeo de rango

Para resolver el problema de determinar la validez del valor del campo se utilizó una función privada de la MEF llamada *checkRango* que utiliza el algoritmo listado a continuación.

Dentro de la función se poseen 2 vectores constantes delimitan los máximos y mínimos para cada uno de los campos. Dichos valores de cota se encuentran ubicados en el vector de forma tal que el índice se corresponda con el ordinal del enumerativo “estado”. Sin embargo, para el caso de los días el máximo depende del mes y año, que ya se encuentran modificados para tal entonces, por lo que debe actualizarse dicho máximo antes de realizar el chequeo:

```
Si se está editando el día
        Actualizar máximo para días según mes y año

Obtener campo actual de “aux” según estado (enumerativo)
Si es menor al mínimo o su valor es 255 (-1)
        Asignar valor máximo para dicho campo
Sino
        Asignar valor mínimo para dicho campo
```

Pseudocódigo 4.1. Función privada *checkRango()* de la MEF.

4.3.3 – Incremento de parámetro

Cuando el reloj se encuentre en algún estado de edición y se presiona la tecla ‘B’, se incrementa el valor del campo correspondiente en 1. Se describe a continuación el algoritmo:

Obtener campo actual de “aux” según estado

Incrementar valor del campo

Realizar chequeo de rango

Escribir campo en LCD

Pseudocódigo 4.2. Función privada *incrementar()* de la MEF.

4.3.4 – Decremento de parámetro

De manera análoga a la función de incremento, cuando se presiona la tecla ‘C’ en un estado de edición, se decrementa el valor del campo corresponde en 1, controlando el rango.

Obtener campo actual de “aux” según estado

Decrementar valor del campo

Realizar chequeo de rango

Escribir campo en LCD

Pseudocódigo 4.3. Función privada *decrementar()* de la MEF.

4.3.5 – Escritura del campo editado

Para escribir el valor del parámetro modificado en el LCD, existen 2 vectores constantes con las coordenadas X e Y del primer digito de cada campo. Cada valor está ubicado de modo que el índice en el vector coincida con el ordinal del estado como enumerativo.

Posicionar cursor en la coordenada (X,Y) según estado

Obtener valor del campo actual de “aux” según estado

Escribir dato en LCD usando 2 caracteres

Pseudocódigo 4.4. Función privada *mostrarParametro()* de la MEF.

4.3.6 – Casos especiales

Como se mencionó anteriormente, el valor máximo permitido para el campo día está determinada por el mes y año que se hayan seleccionado en el proceso de edición. Sin embargo, el control de rango únicamente al incrementar o decrementar el parámetro NO es suficiente para garantizar que la nueva fecha y hora cargada por el usuario sea válida.

En un primer caso se tiene cuando el día actual del reloj (el mostrado durante el estado DEFAULT y copiado al presionar ‘A’) excede el máximo permitido para el mes que posteriormente haya seleccionado el usuario en el estado E_MONTH. Por ejemplo, la fecha original es 31/12/2021, y luego el usuario cambia el mes ‘12’ por ‘11’ y presiona ‘A’ para ingresar al estado E_DAY; como aún no se incrementó ni decrementó el campo “día”, este último no tuvo una corrección de su valor (sigue siendo ‘31’). Se decidió entonces llamar a la función *checkRango* siempre que se realice la transición del estado E_MONTH a E_DAY, de modo que el día es corregido al valor 1 si ocurre la problemática descrita.

En segundo lugar, se tiene el caso cuando la fecha original del reloj es 29 de febrero, y el usuario cambia el año por otro no bisiesto. Al momento de ingresar al estado E_DAY, si no se aplica una corrección inmediata el usuario podrá guardar, por ejemplo, la fecha 29 de febrero del año 2021, que no es bisiesto. Mediante la solución vista para el primer caso también se soluciona este problema, ya que la función *checkRango* actualiza el máximo según mes y año.

4.4 – Simulación

A continuación, se muestran capturas de diferentes validaciones respecto a la corrección de parámetros en edición, en base a lo explicado en los apartados anteriores:

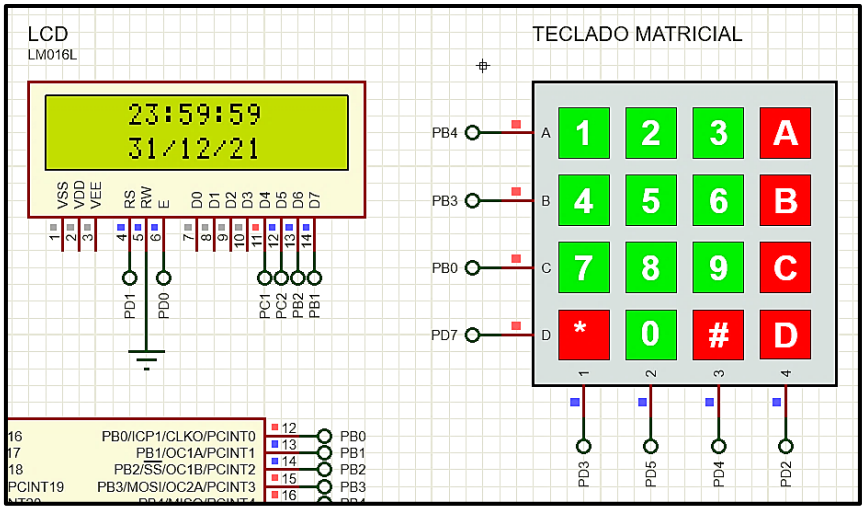

Fig. 4.2. Fecha y hora mostrada en el estado DEFAULT, instante previo a presionar ‘A’.

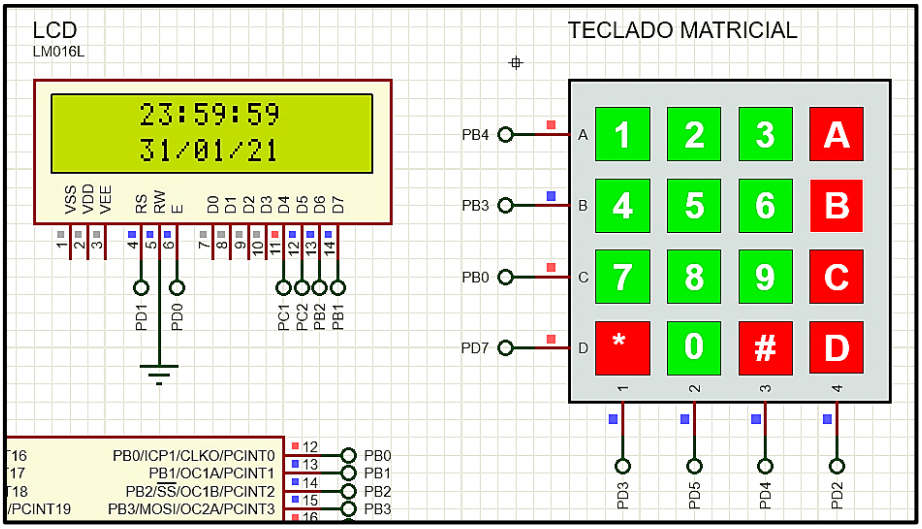

Fig. 4.3. Visualización en estado E_MONTH, luego de presionar 'B'. Cambió mes 12 a 1.

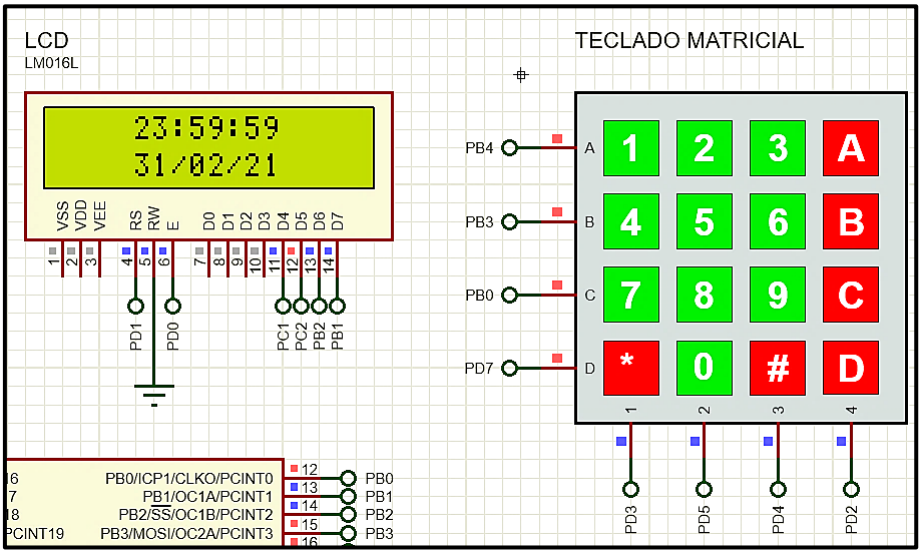

Fig. 4.4. Visualización en estado E_MONTH, luego de presionar 'B' otra vez. Cambió mes 1 a 2.

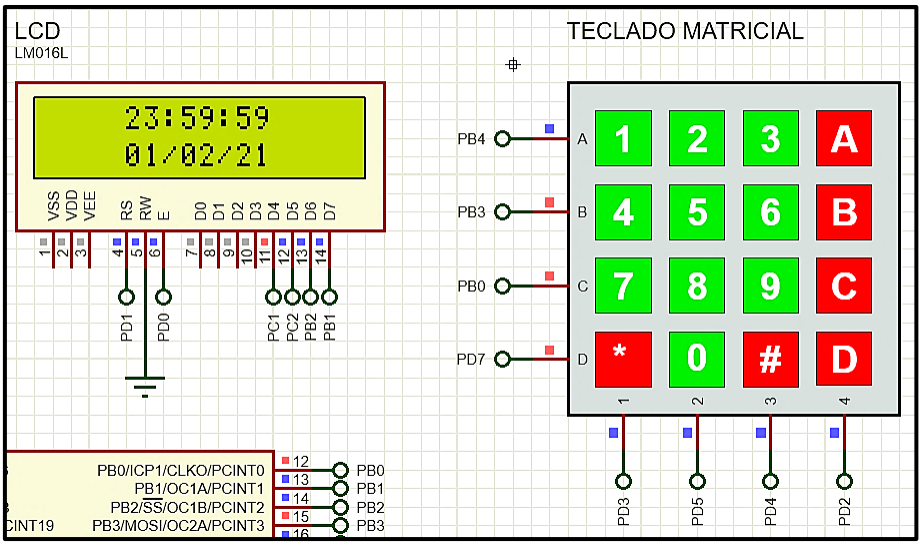

Fig. 4.5. Presión de la tecla 'A' y transición al estado E_DAY. Se corrigió día 31 a 1.

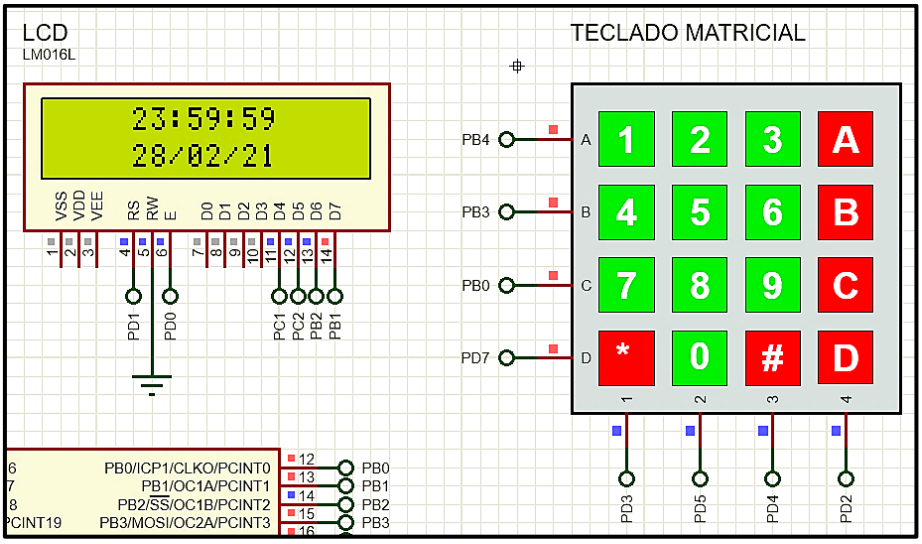

Fig. 4.6. Visualización en estado E_DAY luego de presionar 'C'. Se cambió día 1 a 28.

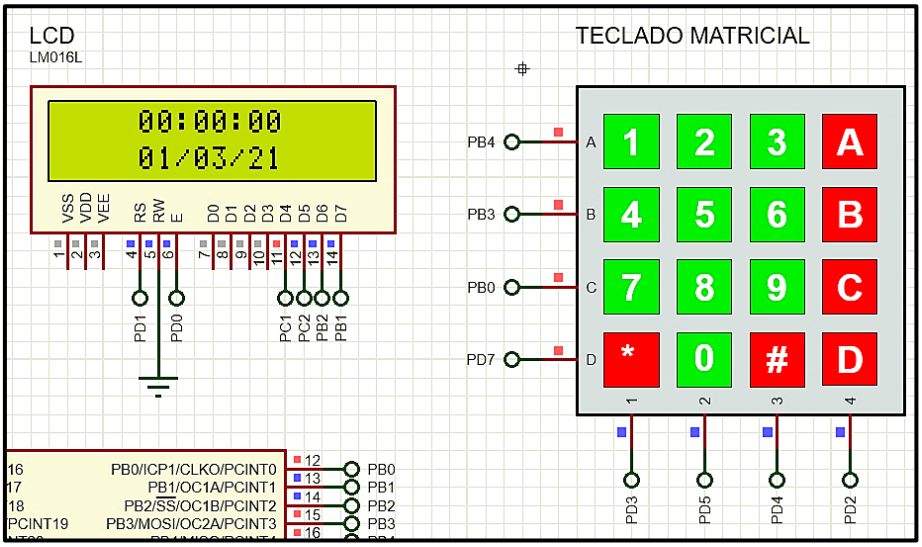

Fig. 4.7. Visualización en estado DEFAULT, 1 segundo después de guardar los cambios.

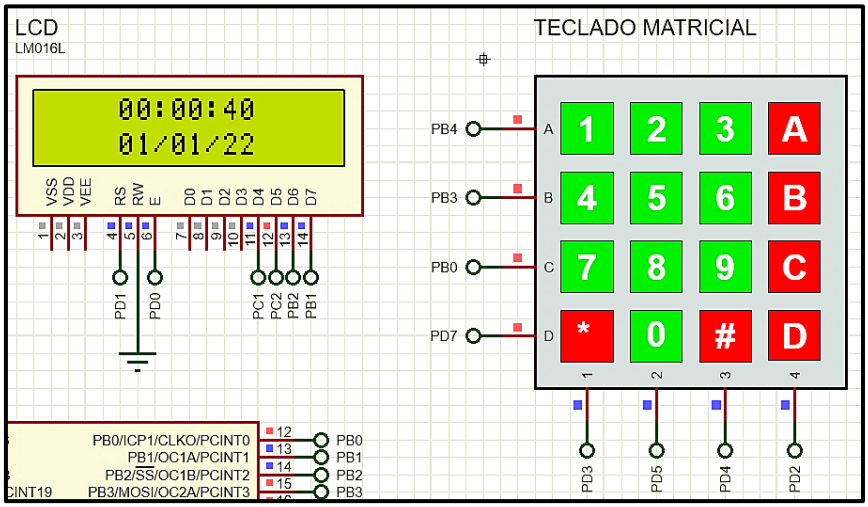

Fig. 4.8. Fecha y hora luego de cancelar los cambios de la Fig. 4.6. Pasaron 41 segundos en edición.

5 – Intermitencia del parámetro actual

5.1 – Enunciado

Mientras transcurre la modificación de un campo, el valor del mismo deberá mostrarse parpadeando cada un segundo hasta que el usuario finalice presionando 'A' para pasar al campo siguiente o para finalizar, o presionando 'D' para cancelar.

5.2 – Interpretación

Al momento de modificar un campo, el valor del mismo deberá mostrarse parpadeando, es decir se debe alternar, cada determinado periodo de tiempo, el valor del campo entre 2 espacios en blanco y luego el valor que le corresponde.

El enunciado propone que el parpadeo se realice cada un segundo, pero se decidió realizar cada 400 ms de forma tal que no brindar una apariencia "lenta" para el usuario al momento de modificar los campos. Entonces el valor del parámetro en edición parpadea cada 400 ms hasta que el usuario presione la tecla 'A' y pase al siguiente campo o finalice la edición, o presionando la tecla 'D' para cancelar la edición.

5.3 – Resolución

5.3.1 – Ocultamiento de parámetro

Para ocultar el campo en edición se ubica el cursor del LCD en la fila y columna correspondiente al mismo, utilizando los 2 vectores constantes de coordenadas mencionado en el apartado 4.3.5, y se envían 2 caracteres en blanco para sobrescribir el valor mostrado.

Luego para posibilitar la intermitencia se actualiza una variable "visible" de la MEF.

```
Posicionar cursor en la coordenada (X,Y) según estado

Escribir 2 veces el carácter ' ' en LCD

Asignar el valor 0 a la variable "visible"
```

Pseudocódigo 5.1. Función privada *ocultarParametro()* de la MEF.

La muestra del parámetro se realiza con la función *mostrarParametro* antes vista, agregando la asignación del valor 1 a la variable "visible" luego de la escritura en el LCD.

5.3.2 – Alternancia entre visible y oculto

Existe además una función privada *toggleParametro* que debe invocarse de manera periódica cada un tiempo de alternancia, en nuestro caso 400 ms (10 llamados a *MEF_Update*).

```
Si "visible" es 1
        Ocultar el campo actual
Sino
        Mostrar el campo actual
```

Pseudocódigo 5.2. Función privada *toggleParametro()* de la MEF.

5.3.3 – Funciones de estado

De manera general, se tiene una variable privada a la MEF llamada "veces_en_estado" que permite identificar cuándo se ingresa a un estado por primera vez. Además, también se guarda cuál fue el último estado accedido para reiniciar la variable a 1 si corresponde.

A continuación, se explicará brevemente las tareas que se realizan en cada llamado de actualización de la MEF según el estado actual, invocando a la función asociada.

5.3.3.1 – Estado DEFAULT

En el primer ingreso al estado (veces_en_estado = 1), se debe mostrar la fecha y hora actuales del módulo Reloj, como se mostró en la sección 2.3.5. Para controlar que se realice cada un segundo, se reinicia el contador de *veces_en_estado* a 0 si el mismo alcanza el valor constante SEGUNDO, que es igual a 25 (1000 / FRECUENCIA_LLAMADO_MS).

Luego se consulta si se presionó una tecla. Sólo se verifica si la tecla es 'A', y en caso afirmativo se realiza una transición al estado E_YEAR para editar el año.

```
Si "veces_en_estado" es 1
        Mostrar fecha y hora actuales en el LCD
Sino si "veces_en_estado" es 25
        Reiniciar "veces_en_estado" a cero
Si se presionó una tecla y es 'A'
        Realizar transición al siguiente estado
```

Pseudocódigo 5.3. Función de estado privada *fsDEFAULT()* de la MEF.

5.3.3.2 – Estados de edición

En las funciones de estado restantes se puede apreciar un algoritmo similar que se repite, por tal motivo se procederá a describir el mismo de forma genérica y de ser necesario se detallarán los casos específicos. Se utiliza la variable "veces_en_estado" para realizar el parpadeo. Dicha variable se incrementa hasta 10 (400 / FRECUENCIA_LLAMADO_MS).

```
Si "veces_en_estado" es 1
        Cambiar visibilidad del parámetro
Sino si "veces_en_estado" es 10
        Reiniciar "veces_en_estado" a cero
Según el valor de la tecla presionada
        Caso 'A':
                Se escribe el parámetro actual por si quedó oculto
                Se realiza la transición al estado siguiente
        Caso 'B':
                Incrementar el valor del parámetro
        Caso 'C':
                Decrementar el valor del parámetro
        Caso 'D':
                Cancelar la edición
```

Pseudocódigo 5.4. Forma genérica de las funciones de estado E_* de la MEF.

Como primer caso especial se tiene a la función de estado *fsE_MONTH*, cuando se presiona la tecla 'A' se agrega, luego de cambiar el estado, la invocación a la función privada *checkRango()* que permite corregir el día si no existe para el mes y año seleccionados.

En segundo lugar, en la función de estado *fsE_SECOND*, cuando se presiona la tecla 'A' se agrega al inicio la invocación de la función pública *RELOJ_updateCurrent* que recibe un parámetro por referencia a la variable de Tiempo "aux" y cumple el objetivo de actualizar los campos de la estructura de tiempo actual con los de la estructura editada. Es decir, si se presiona la tecla 'A' en el estado E_SECOND se escriben los datos nuevos en el reloj.

5.4 – Simulación

A continuación, se muestran dos capturas que validan el período de parpadeo:

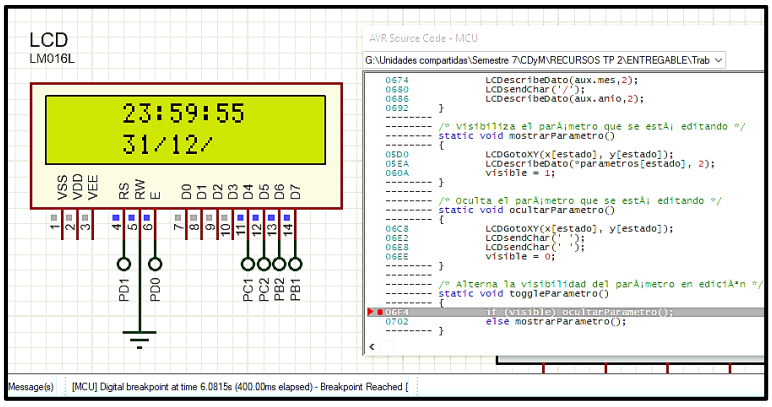

Fig. 5.1. Verificación del lapso de tiempo de ocultamiento de parámetro. Son 400 ms.

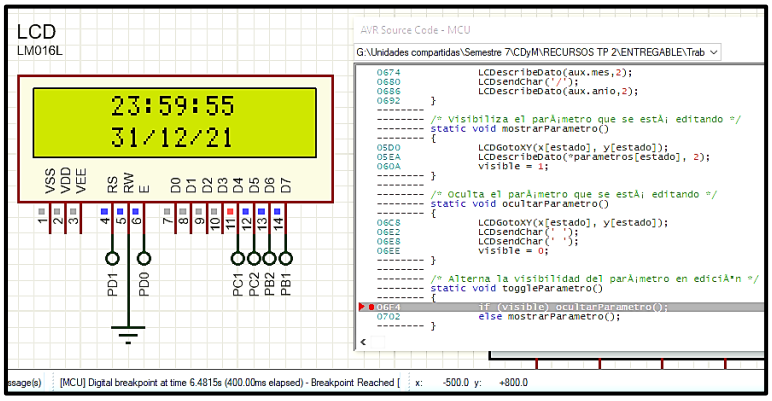

Fig. 5.2. Verificación del lapso de tiempo de muestra de parámetro. Nuevamente 400 ms.

6 – Conclusiones

6.1 – Ventajas de la solución

- Respecto a la exactitud del reloj, se puede afirmar que, si la frecuencia del sistema se mantiene constante en 16 MHz, no se producirá atraso alguno en la variable de tiempo, debido a que el valor de comparación OCR1A no fue redondeado, y además el período de interrupción es submúltiplo de 1000 ms y suficiente para que la actualización de la MEF se realice de manera completa en cada iteración del programa principal.

- Las acciones realizadas al presionar una tecla se encuentran generalizadas en su mayoría, gracias a que el estado representado como enumerativo tiene asociado un valor ordinal que se puede utilizar como índice para vectores de coordenadas y parámetros.

- La actualización del reloj se produce de manera continua e independiente de la MEF, el tiempo no se para durante el proceso de edición si se descartan los cambios.

6.2 – Limitaciones encontradas

6.2.1 – Incremento y decremento unitario

La resolución respecto a la edición de los parámetros tiene la particularidad de que los incrementos y decrementos de los mismos solo se pueden hacer de a una unidad a la vez. Esto significa que si se desea alterar el valor del campo en un sentido en N unidades entonces se debe oprimir la tecla 'B' o 'C', según corresponda, unas N veces.

Una posibilidad sería que al mantener presionado la tecla ('B' o 'C') se realicen los incrementos/decrementos de manera continua. Sin embargo, si se modificara la función *KEYPAD_Update*, el cambio involucraría también a las teclas 'A' y 'D', pudiendo en el primer caso provocar cambios de estado no deseados. Para diferenciarlo, se podría tener otra función pública que encueste las teclas 'B' y 'C', pero reduciría la reusabilidad de código.

6.2.2 – Imposibilidad de generalizar estados

Ante la repetición de código en los estados de edición, se intentó usar una estructura de transiciones, con entradas y salidas codificadas (según Tabla 4.1), pero reaparecen problemas como parámetros que quedan ocultos y días excedidos de rango (sección 4.3.6); y no se puede forzar a invocar *mostrarParametro* y *checkRango* en, por ejemplo, el estado DEFAULT.

6.3 – Comparación con otras alternativas

Otra posibilidad para la producción de interrupciones periódicas es la utilización de Timer 0, que puede realizarse cada 8 ms si se desea mantener la exactitud del reloj. Como se verificó en la Figura 2.9, la impresión de la fecha y hora completa demora más de 10 ms. Una alternativa es imprimir la primera línea en un llamado a la MEF, y luego la segunda en el siguiente llamado. Sin embargo, existe la posibilidad de que al presionar 'A' quede actualizada la hora en pantalla, pero no la fecha, justo cuando se produce el cambio de día.

Si se persiste en utilizar Timer 0 aunque con interrupciones cada 15 ms, por ejemplo, se perderá precisión en el reloj porque el valor de comparación OCR0A deberá ser redondeado, y además 1000 ms no es divisible por 15 ms. El próximo divisor, 20 ms, requiere 16 bits.

7 – Bibliografía

- Atmel Corporation (2015). *ATmega 328P Datasheet.* Capítulos 11, 14, 15 y 17. https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

Apéndice A – Archivos de cabecera

A continuación, se muestran los archivos de cabecera del proyecto realizado.

main.h

```
#ifndef MAIN_H_
#define MAIN_H_

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>

#define F_CPU 16000000UL
#include <util/delay.h>

// Componentes
#include "lcd.h"
#include "reloj.h"
#include "teclado4x4.h"
#include "MEF.h"

// Constantes
#define FRECUENCIA_LLAMADO_MS 40

#endif /* MAIN_H_ */
```

lcd.h

```
#ifndef LCD_H
#define LCD_H

#include <inttypes.h>

#include "main.h"

//Uncomment this if LCD 4 bit interface is used
//*****************************************
#define LCD_4bit
//*****************************************

#define LCD_RS1      //define MCU pin connected to LCD RS
#define LCD_RW1      //define MCU pin connected to LCD R/W
#define LCD_E        0      //define MCU pin connected to LCD E
#define LCD_D00      //define MCU pin connected to LCD D0
#define LCD_D11      //define MCU pin connected to LCD D1
#define LCD_D22      //define MCU pin connected to LCD D1
#define LCD_D33      //define MCU pin connected to LCD D2
#define LCD_D41      //define MCU pin connected to LCD D3
#define LCD_D52      //define MCU pin connected to LCD D4
#define LCD_D62      //define MCU pin connected to LCD D5
#define LCD_D71      //define MCU pin connected to LCD D6
#define LDP1 PORTB   //define MCU port connected to LCD data pins
#define LDP2 PORTC   //define MCU port connected to LCD data pins
#define LCP PORTD    //define MCU port connected to LCD control pins
#define LDDR1 DDRB   //define MCU direction register for port connected to LCD data pins
#define LDDR2 DDRC   //define MCU direction register for port connected to LCD data pins
#define LCDR DDRD    //define MCU direction register for port connected to LCD control pins

#define LCD_DATAWR(Data) {PORTB = (PORTB & 0xF9) | ((Data & 0x40)>>4) | ((Data & 0X80)>>6);
PORTC = (PORTC & 0xF9) | ((Data & 0X10) >> 3) | ((Data & 0X20) >> 3);}
```

```
#define LCD_CLR             0      //DB0: clear display
#define LCD_HOME            1      //DB1: return to home position
#define LCD_ENTRY_MODE      2      //DB2: set entry mode
#define LCD_ENTRY_INC       1      //DB1: increment
#define LCD_ENTRY_SHIFT     0      //DB2: shift
#define LCD_ON_CTRL         3      //DB3: turn lcd/cursor on
#define LCD_ON_DISPLAY      2      //DB2: turn display on
#define LCD_ON_CURSOR       1      //DB1: turn cursor on
#define LCD_ON_BLINK        0      //DB0: blinking cursor
#define LCD_MOVE            4      //DB4: move cursor/display
#define LCD_MOVE_DISP       3      //DB3: move display (0-> move cursor)
#define LCD_MOVE_RIGHT      2      //DB2: move right (0-> left)
#define LCD_FUNCTION        5      //DB5: function set
#define LCD_FUNCTION_8BIT   4      //DB4: set 8BIT mode (0->4BIT mode)
#define LCD_FUNCTION_2LINES 3      //DB3: two lines (0->one line)
#define LCD_FUNCTION_10DOTS 2      //DB2: 5x10 font (0->5x7 font)
#define LCD_CGRAM           6      //DB6: set CG RAM address
#define LCD_DDRAM           7      //DB7: set DD RAM address
// reading:
#define LCD_BUSY            7      //DB7: LCD is busy
#define LCD_LINES           2      //visible lines
#define LCD_LINE_LENGTH     16     //line length (in characters)
// cursor position to DDRAM mapping
#define LCD_LINE0_DDRAMADDR 0x00
#define LCD_LINE1_DDRAMADDR 0x40
#define LCD_LINE2_DDRAMADDR 0x14
#define LCD_LINE3_DDRAMADDR 0x54
// progress bar defines
#define PROGRESSPIXELS_PER_CHAR     6

void LCDescribeDato(int val, unsigned int field_length); //Escribe un número en LCD
void LCDsendChar(uint8_t);          //forms data ready to send to 74HC164
void LCDsendCommand(uint8_t);       //forms data ready to send to 74HC164
void LCDinit(void);                 //Initializes LCD
void LCDclr(void);                  //Clears LCD
void LCDhome(void);                 //LCD cursor home
void LCDstring(uint8_t*, uint8_t);  //Outputs string to LCD
void LCDGotoXY(uint8_t, uint8_t);   //Cursor to X Y position
void CopyStringtoLCD(const uint8_t*, uint8_t, uint8_t);  //copies flash string to LCD at x,y
void LCDdefinechar(const uint8_t *,uint8_t);             //write char to LCD CGRAM
void LCDshiftRight(uint8_t);        //shift by n characters Right
void LCDshiftLeft(uint8_t);         //shift by n characters Left
void LCDcursorOn(void);             //Underline cursor ON
void LCDcursorOnBlink(void);        //Underline blinking cursor ON
void LCDcursorOFF(void);            //Cursor OFF
void LCDblank(void);                //LCD blank but not cleared
void LCDvisible(void);              //LCD visible
void LCDcursorLeft(uint8_t);        //Shift cursor left by n
void LCDcursorRight(uint8_t);       //shif cursor right by n
// displays a horizontal progress bar at the current cursor location
// <progress> is the value the bargraph should indicate
// <maxprogress> is the value at the end of the bargraph
// <length> is the number of LCD characters that the bargraph should cover
//adapted from AVRLIB - displays progress only for 8 bit variables
void LCDprogressBar(uint8_t progress, uint8_t maxprogress, uint8_t length);
void LCD_Init();
void LCD_Update();



#endif
```

reloj.h

```
#ifndef RELOJ_H_
#define RELOJ_H_

#include "main.h"

/* Declaración de tipos */
typedef struct {
	uint8_t anio;
	uint8_t mes;
	uint8_t dia;
	uint8_t hora;
	uint8_t minuto;
	uint8_t segundo;
} Tiempo;

/* Prototipos de funciones públicas */
void RELOJ_Init();                                  // Inicializa el reloj con Timer 1
void RELOJ_getCurrent(Tiempo*);                     // Devuelve una copia del tiempo actual
void RELOJ_updateCurrent(Tiempo*);                  // Sobrescribe el tiempo actual
void RELOJ_IncMs(uint8_t);                          // Incrementa milisegundos al reloj
uint8_t UTIL_DiaMaximo(uint8_t, uint16_t);          // Devuelve día máximo según mes y año
uint8_t UTIL_NoBisiesto(uint16_t);                  // Devuelve 0 si el año es bisiesto
#endif /* RELOJ_H_ */
```

teclado4x4.h

```
#ifndef TECLADO4X4_H_
#define TECLADO4X4_H_

/* Cabeceras */
#include "main.h"

/* Constantes */
#define REPOSO 0
#define PRESIONADO 1
#define TECLA_NO_VALIDA 0xFF

/* Prototipos de funciones públicas */
void KEYPAD_Init();                                 // Inicializa el teclado matricial

/* Encuesta el teclado para determinar si se presionó una tecla y pasa el valor de la misma
por referencia. Se retorna 0 si no se presionó ninguna tecla o 1 si se presionó alguna. */
uint8_t KEYPAD_Update(uint8_t *tecla);

#endif /* TECLADO4X4_H_ */
```

MEF.h

```
#ifndef MEF_H_
#define MEF_H_

#include "main.h"

// Constantes
#define SEGUNDO 1000 / FRECUENCIA_LLAMADO_MS
#define OFFSET_PARPADEO 400 / FRECUENCIA_LLAMADO_MS

// Prototipos de función
void MEF_Init();                    // Inicializa la MEF
void MEF_Update();                  // Actualiza la MEF según su estado actual

#endif /* MEF_H_ */
```

Apéndice B – Archivos .C

A continuación, se muestran los archivos .C que conforman el programa completo, los cuales se compilan y utilizan para el funcionamiento del sistema en el simulador Proteus.

main.c

```
/* Cabeceras */
#include "main.h"

/* Variables */
static uint8_t FLAG_TIMER = 0;

int main() {
	/* Setup */
	MEF_Init();
	LCDinit();
	RELOJ_Init();
	KEYPAD_Init();

	/* Loop */
	while (1)
	{
		if(FLAG_TIMER)
		{
			RELOJ_IncMs(FRECUENCIA_LLAMADO_MS);// Incrementar tiempo actual
			MEF_Update();                      // Actualizar máquina de estados
			FLAG_TIMER = 0;                    // Reiniciar flag
		}

		sleep_mode();                          // Suspender uso de CPU
	}

	return 0;
}

/*****************************************************************
  ISR TIMER : se interrumpe cada 40 ms
*****************************************************************/
ISR(TIMER1_COMPA_vect) {
    FLAG_TIMER = 1;
}
```

lcd.c

```
#include "lcd.h"

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

extern volatile unsigned int temp;

const uint8_t LcdCustomChar[] PROGMEM=            //define 8 custom LCD chars
{
	0x00, 0x1F, 0x00, 0x00, 0x00, 0x00, 0x1F, 0x00, // 0. 0/5 full progress block
	0x00, 0x1F, 0x10, 0x10, 0x10, 0x10, 0x1F, 0x00, // 1. 1/5 full progress block
	0x00, 0x1F, 0x18, 0x18, 0x18, 0x18, 0x1F, 0x00, // 2. 2/5 full progress block
	0x00, 0x1F, 0x1C, 0x1C, 0x1C, 0x1C, 0x1F, 0x00, // 3. 3/5 full progress block
```

```
        0x00, 0x1F, 0x1E, 0x1E, 0x1E, 0x1E, 0x1F, 0x00, // 4. 4/5 full progress block
        0x00, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x00, // 5. 5/5 full progress block
        0x03, 0x07, 0x0F, 0x1F, 0x0F, 0x07, 0x03, 0x00, // 6. rewind arrow
        0x18, 0x1C, 0x1E, 0x1F, 0x1E, 0x1C, 0x18, 0x00  // 7. fast-forward arrow
};


void LCDsendChar(uint8_t ch)              // Sends Char to LCD
{

#ifdef LCD_4bit
        //4 bit part
        //LDP=(ch&0b11110000);
        LCD_DATAWR(ch&0b11110000);
        LCP|=1<<LCD_RS;
        LCP|=1<<LCD_E;
        _delay_us(40);
        LCP&=~(1<<LCD_E);
        LCP&=~(1<<LCD_RS);
        _delay_us(40);
        //LDP=((ch&0b00001111)<<4);
        LCD_DATAWR((ch&0b00001111)<<4);
        LCP|=1<<LCD_RS;
        LCP|=1<<LCD_E;
        _delay_us(40);
        LCP&=~(1<<LCD_E);
        LCP&=~(1<<LCD_RS);
        _delay_us(40);
#else
        //8 bit part
        LDP=ch;
        LCP|=1<<LCD_RS;
        LCP|=1<<LCD_E;
        _delay_ms(1);
        LCP&=~(1<<LCD_E);
        LCP&=~(1<<LCD_RS);
        _delay_ms(1);
#endif
}

void LCDsendCommand(uint8_t cmd)          // Sends Command to LCD
{
#ifdef LCD_4bit
        //4 bit part
        //LDP=(cmd&0b11110000);
        LCD_DATAWR(cmd&0b11110000);
        LCP|=1<<LCD_E;
        _delay_ms(1);
        LCP&=~(1<<LCD_E);
        _delay_ms(1);
        //LDP=((cmd&0b00001111)<<4);
        LCD_DATAWR((cmd&0b00001111)<<4);
        LCP|=1<<LCD_E;
        _delay_ms(1);
        LCP&=~(1<<LCD_E);
        _delay_ms(1);
#else
        //8 bit part
        LDP=cmd;
        LCP|=1<<LCD_E;
        _delay_ms(1);
        LCP&=~(1<<LCD_E);
        _delay_ms(1);
#endif
}
```

```
void LCDinit(void)                          //Initializes LCD
{
#ifdef LCD_4bit
       //4 bit part
       _delay_ms(15);
       //LDP=0x00;
       LCD_DATAWR(0x00);
       LCP=0x00;
       DDRC|=0x06;
       DDRB|=0x06;
       LDDR1|=1<<LCD_D7|1<<LCD_D6;
       LDDR2|=1<<LCD_D4|1<<LCD_D5;
       //LDDR|=1<<LCD_D7|1<<LCD_D6|1<<LCD_D5|1<<LCD_D4;
       LCDR|=1<<LCD_E|1<<LCD_RW|1<<LCD_RS;
       //---------one------
       //LDP=0<<LCD_D7|0<<LCD_D6|1<<LCD_D5|1<<LCD_D4; //4 bit mode
       LCD_DATAWR(0b00110000);
       LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS;
       _delay_ms(1);
       LCP&=~(1<<LCD_E);
       _delay_ms(1);
       //-----------two-----------
       //LDP=0<<LCD_D7|0<<LCD_D6|1<<LCD_D5|1<<LCD_D4; //4 bit mode
       LCD_DATAWR(0b00110000);
       LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS;
       _delay_ms(1);
       LCP&=~(1<<LCD_E);
       _delay_ms(1);
       //-------three-------------
       //LDP=0<<LCD_D7|0<<LCD_D6|1<<LCD_D5|0<<LCD_D4; //4 bit mode
       LCD_DATAWR(0b00100000);
       LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS;
       _delay_ms(1);
       LCP&=~(1<<LCD_E);
       _delay_ms(1);
       //--------4 bit--dual line---------------
       LCDsendCommand(0b00101000);
       //-----increment address, invisible cursor shift------
       LCDsendCommand(0b00001100);
       //init 8 custom chars
       uint8_t ch=0, chn=0;
       while(ch<64)
       {
              LCDdefinechar((LcdCustomChar+ch),chn++);
              ch=ch+8;
       }
#else
       //8 bit part
       _delay_ms(15);
       LDP=0x00;
       LCP=0x00;
       LDDR|=1<<LCD_D7|1<<LCD_D6|1<<LCD_D5|1<<LCD_D4|1<<LCD_D3
                     |1<<LCD_D2|1<<LCD_D1|1<<LCD_D0;
       LCDR|=1<<LCD_E|1<<LCD_RW|1<<LCD_RS;
       //---------one------
       LDP=0<<LCD_D7|0<<LCD_D6|1<<LCD_D5|1<<LCD_D4|0<<LCD_D3
                     |0<<LCD_D2|0<<LCD_D1|0<<LCD_D0;     //8 it mode
       LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS;
       _delay_ms(1);
       LCP&=~(1<<LCD_E);
       _delay_ms(1);
       //-----------two-----------
       LDP=0<<LCD_D7|0<<LCD_D6|1<<LCD_D5|1<<LCD_D4|0<<LCD_D3
                     |0<<LCD_D2|0<<LCD_D1|0<<LCD_D0;     //8 it mode
       LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS;
```

```
        _delay_ms(1);
        LCP&=~(1<<LCD_E);
        _delay_ms(1);
        //-------three-------------
        LDP=0<<LCD_D7|0<<LCD_D6|1<<LCD_D5|1<<LCD_D4|0<<LCD_D3
                        |0<<LCD_D2|0<<LCD_D1|0<<LCD_D0; //8 it mode
        LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS;
        _delay_ms(1);
        LCP&=~(1<<LCD_E);
        _delay_ms(1);
        //--------8 bit dual line----------
        LDP=0<<LCD_D7|0<<LCD_D6|1<<LCD_D5|1<<LCD_D4|1<<LCD_D3
                        |0<<LCD_D2|0<<LCD_D1|0<<LCD_D0; //8 it mode
        LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS;
        _delay_ms(1);
        LCP&=~(1<<LCD_E);
        _delay_ms(1);
        //-----increment address, invisible cursor shift------
        LDP=0<<LCD_D7|0<<LCD_D6|0<<LCD_D5|0<<LCD_D4|1<<LCD_D3
                        |1<<LCD_D2|0<<LCD_D1|0<<LCD_D0; //8 it mode
        LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS;
        _delay_ms(1);
        LCP&=~(1<<LCD_E);
        _delay_ms(5);
        //init custom chars
        uint8_t ch=0, chn=0;
        while(ch<64)
        {
                LCDdefinechar((LcdCustomChar+ch),chn++);
                ch=ch+8;
        }
#endif
}

void LCDclr(void)                           //Clears LCD
{
        LCDsendCommand(1<<LCD_CLR);
}

void LCDhome(void)                          //LCD cursor home
{
        LCDsendCommand(1<<LCD_HOME);
}

void LCDstring(uint8_t* data, uint8_t nBytes)     //Outputs string to LCD
{
        register uint8_t i;

        // check to make sure we have a good pointer
        if (!data) return;

        // print data
        for(i=0; i<nBytes; i++) LCDsendChar(data[i]);
}

void LCDGotoXY(uint8_t x, uint8_t y)        //Cursor to X Y position
{
        register uint8_t DDRAMAddr;
        // remap lines into proper order
        switch(y)
        {
                case 0: DDRAMAddr = LCD_LINE0_DDRAMADDR+x; break;
                case 1: DDRAMAddr = LCD_LINE1_DDRAMADDR+x; break;
                case 2: DDRAMAddr = LCD_LINE2_DDRAMADDR+x; break;
                case 3: DDRAMAddr = LCD_LINE3_DDRAMADDR+x; break;
```

```
                default: DDRAMAddr = LCD_LINE0_DDRAMADDR+x;
        }
        // set data address
        LCDsendCommand(1<<LCD_DDRAM | DDRAMAddr);

}

//Copies string from flash memory to LCD at x y position
//const uint8_t welcomeln1[] PROGMEM="AVR LCD DEMO\0";
//CopyStringtoLCD(welcomeln1, 3, 1);
void CopyStringtoLCD(const uint8_t *FlashLoc, uint8_t x, uint8_t y)
{
        uint8_t i;
        LCDGotoXY(x,y);
        for(i=0;(uint8_t)pgm_read_byte(&FlashLoc[i]);i++)
        {
                LCDsendChar((uint8_t)pgm_read_byte(&FlashLoc[i]));
        }
}

//defines char symbol in CGRAM
void LCDdefinechar(const uint8_t *pc,uint8_t char_code){
        uint8_t a, pcc;
        uint16_t i;
        a=(char_code<<3)|0x40;
        for (i=0; i<8; i++){
                pcc=pgm_read_byte(&pc[i]);
                LCDsendCommand(a++);
                LCDsendChar(pcc);
        }
}

void LCDshiftLeft(uint8_t n)        //Scroll n of characters Right
{
        for (uint8_t i=0;i<n;i++) LCDsendCommand(0x1E);
}

void LCDshiftRight(uint8_t n)       //Scroll n of characters Left
{
        for (uint8_t i=0;i<n;i++) LCDsendCommand(0x18);
}

void LCDcursorOn(void)              //displays LCD cursor
{
        LCDsendCommand(0x0E);
}

void LCDcursorOnBlink(void)         //displays LCD blinking cursor
{
        LCDsendCommand(0x0F);
}

void LCDcursorOFF(void)             //turns OFF cursor
{
        LCDsendCommand(0x0C);
}
void LCDblank(void)                 //blanks LCD
{
        LCDsendCommand(0x08);
}

void LCDvisible(void)               //Shows LCD
{
        LCDsendCommand(0x0C);
}
```

```
void LCDcursorLeft(uint8_t n)      //Moves cursor by n poisitions left
{
	for (uint8_t i=0;i<n;i++) LCDsendCommand(0x10);
}

void LCDcursorRight(uint8_t n)     //Moves cursor by n poisitions left
{
	for (uint8_t i=0;i<n;i++) LCDsendCommand(0x14);
}
//adapted fro mAVRLIB

void LCDescribeDato(int val,unsigned int field_length)
{
	char str[5]={0,0,0,0,0};
	int i=4,j=0;
	while(val)
	{
		str[i]=val%10;
		val=val/10;
		i--;
	}
	if(field_length==-1) while(str[j]==0) j++;
	else j=5-field_length;

	if(val<0) LCDsendChar('-');
	for(i=j;i<5;i++) LCDsendChar(48+str[i]);
}

void LCDprogressBar(uint8_t progress, uint8_t maxprogress, uint8_t length)
{
	uint8_t i;
	uint16_t pixelprogress;
	uint8_t c;

	// total pixel length of bargraph equals length*PROGRESSPIXELS_PER_CHAR;
	// pixel length of bar itself is
	pixelprogress = ((progress*(length*PROGRESSPIXELS_PER_CHAR))/maxprogress);

	// print exactly "length" characters
	for(i=0; i<length; i++)
	{
		// check if this is a full block, or partial or empty
		// (u16) cast is needed to avoid sign comparison warning
		if( ((i*(uint16_t)PROGRESSPIXELS_PER_CHAR)+5) > pixelprogress )
		{
			// this is a partial or empty block
			if( ((i*(uint16_t)PROGRESSPIXELS_PER_CHAR)) > pixelprogress )
			{
				// this is an empty block
				// use space character?
				c = 0;
			} else {
				// this is a partial block
				c = pixelprogress % PROGRESSPIXELS_PER_CHAR;
			}
		} else {
			// this is a full block
			c = 5;
		}

		// write character to display
		LCDsendChar(c);
	}
}
```

reloj.c

```
#include "reloj.h"

// Variables privadas al componente
static Tiempo t = {21,12,31,23,59,50};
static uint16_t ms = 0;

/* Prototipos de funciones privadas */
static void RELOJ_IncSecond();

void RELOJ_Init(void)
{
	// Asegurarse que las interrupciones del Timer están deshabilitadas
	TIMSK1 &= ~((1<<TOIE1)|(1<<OCIE1A)|(1<<OCIE1B));

	// Preescaler de N = 1024 y modo CTC
	TCCR1B = (1<<CS10)|(1<<CS12)|(1<<WGM12);

	TCNT1 = 0;				// Reiniciar contador
	OCR1A = 624;				// Valor para la comparación

	TIMSK1 |= (1<<OCIE1A);			// Habilitar interrupción por Match Compare
	sei();					// Habilitar interrupciones globales
}

void RELOJ_getCurrent(Tiempo* copia)
{
	copia->anio = t.anio;
	copia->mes = t.mes;
	copia->dia = t.dia;
	copia->hora = t.hora;
	copia->minuto = t.minuto;
	copia->segundo = t.segundo;
}

void RELOJ_updateCurrent(Tiempo* nuevo)
{
	t.anio = nuevo->anio;
	t.mes = nuevo->mes;
	t.dia = nuevo->dia;
	t.hora = nuevo->hora;
	t.minuto = nuevo->minuto;
	t.segundo = nuevo->segundo;
}

void RELOJ_IncMs(uint8_t quantity)
{
	ms += quantity;
	if (ms >= 1000)
	{
		RELOJ_IncSecond();
		ms = ms % 1000;
	}
}

uint8_t UTIL_NoBisiesto(uint16_t anio)
{
	// Si es un año divisible por 100 y no es divisible por 400 (ej: 1900, 2100)
	if (!(anio%100)) return anio % 400;

	// Si no es divisible por 4 (ej: 2019, 2021)
	return anio % 4;
}
```

```
uint8_t UTIL_DiaMaximo(uint8_t mes, uint16_t anio)
{
	if ((mes == 4) || (mes == 6) || (mes == 9) || (mes == 11)) return 30;
	else if (mes == 2) {
		if (UTIL_NoBisiesto(anio)) return 28;
		else return 29;
	}

	return 31;
}

/* Suma 1 segundo al tiempo actual, incrementando otros parámetros si corresponde */
static void RELOJ_IncSecond()
{
	// Si transcurrió 1 minuto...
	if (++t.segundo == 60)
	{
		t.segundo=0;
		// Si transcurrió 1 hora...
		if (++t.minuto==60)
		{
			t.minuto=0;
			// Si transcurrió 1 día...
			if (++t.hora==24)
			{
				t.hora=0;
				t.dia++;

				// Si transcurrió el mes completo...
				if (t.dia > UTIL_DiaMaximo(t.mes, t.anio))
				{
					t.mes++;
					t.dia=1;

					// Si transcurrió un 1 año...
					if (t.mes==13)
					{
						t.mes=1;
						t.anio++;

						// Si transcurrieron 200 años...
						if (t.anio==200) t.anio=0;
					}
				}
			}
		}
	}
}
```

teclado4x4.c

```
#include "teclado4x4.h"

// Matriz de caracteres asociados a cada posición del teclado
static const uint8_t caracteres[4][4] =
{
	{'1','2','3','A'}, {'4','5','6','B'}, {'7','8','9','C'}, {'*','0','#','D'}
};

// Vectores de bits utilizados de cada puerto
static const uint8_t filas[3] = {PINB4, PINB3, PINB0};
static const uint8_t columnas[4] = {PORTD3, PORTD5, PORTD4, PORTD2};

// Prototipos de funciones privadas
static uint8_t KEYPAD_Scan(uint8_t*);

void KEYPAD_Init()
{
	// Se configuran las filas como entradas con Pull-Up
	DDRB &= ~((1<<PORTB0) | (1<<PORTB3) | (1<<PORTB4));
	DDRD &= ~(1<<PORTD7);
	PORTB |= (1<<PORTB0) | (1<<PORTB3) | (1<<PORTB4);
	PORTD |= (1<<PORTD7);

	// Se configuran las columnas como salidas
	DDRD |= (1<<PORTD3) | (1<<PORTD5) | (1<<PORTD4) | (1<<PORTD2);
}

static uint8_t KEYPAD_Scan(uint8_t *key)
{
	uint8_t reposo;

	// Se desactivan todas las salidas (columnas)
	PORTD |= (1<<PORTD3) | (1<<PORTD5) | (1<<PORTD4) | (1<<PORTD2);

	// Por cada columna del teclado...
	for (uint8_t c=0; c < 4; c++)
	{
		// Se prueba activando con 0 la columna actual
		PORTD &= ~(1<<columnas[c]);

		// Se leen las filas 0 a 2 (asociadas a Puerto B)
		for (uint8_t f=0; f < 3; f++)
		{
			// Leer pulsador
			reposo = PINB & (1<<filas[f]);
			if (!reposo)
			{
				*key = caracteres[f][c]; // Pasar caracter asociado
				return PRESIONADO;       // Devolver que se presionó una tecla
			}
		}

		// Se lee la última fila (asociada a Puerto D)
		reposo = PIND & (1<<PIND7);
		if (!reposo)
		{
			*key = caracteres[3][c];      // Pasar caracter asociado
			return PRESIONADO;            // Devolver que hay tecla presionada
		}
	}

	return REPOSO;                          // Devolver que no hay tecla presionada
}
```

```
uint8_t KEYPAD_Update(uint8_t* tecla)
{
	static uint8_t tecla_anterior = TECLA_NO_VALIDA;
	static uint8_t tecla_confirmada = TECLA_NO_VALIDA;
	uint8_t tecla_actual;

	// Se encuesta el teclado
	uint8_t estado_actual = KEYPAD_Scan(&tecla_actual);

	// Si hay tecla presionada en este momento...
	if (estado_actual == PRESIONADO)
	{
		// Si es una detección consecutiva de la tecla...
		if (tecla_actual == tecla_anterior)
		{
			// Si aún no se devolvió la tecla por referencia
			if (tecla_actual != tecla_confirmada)
			{
				*tecla = tecla_actual;			// Pasar por referencia
				tecla_confirmada = tecla_actual;	// Evitar doble detección
				return PRESIONADO;		// Devolver que se presionó una tecla
			}
		} else tecla_anterior = tecla_actual;		// Guardar para próxima iteración
	} else {
		tecla_anterior = TECLA_NO_VALIDA;			// Borrar teclas guardadas
		tecla_confirmada = TECLA_NO_VALIDA;
	}

	return REPOSO;					// Devolver que no se presionó una tecla
}
```

MEF.c

```
#include "MEF.h"

// Declaración de tipos
typedef enum {E_YEAR, E_MONTH, E_DAY, E_HOUR, E_MINUTE, E_SECOND, DEFAULT} MEF_Estado;

/* Prototipos de funciones privadas */
static void fsE_YEAR(void);
static void fsE_MONTH(void);
static void fsE_DAY(void);
static void fsE_HOUR(void);
static void fsE_MINUTE(void);
static void fsE_SECOND(void);
static void fsDEFAULT(void);
static void cancelar();
static void checkRango();
static void decrementar();
static void incrementar();
static void mostrarFechaHora();
static void mostrarParametro();
static void ocultarParametro();
static void toggleParametro();

/* Variables privadas de la clase */
static MEF_Estado estado;
static Tiempo aux;
static uint8_t tecla, visible, veces_en_estado;
static const uint8_t x[] = {10, 7, 4, 4, 7, 10};
static const uint8_t y[] = {1, 1, 1, 0, 0, 0};
static uint8_t* parametros[] = {&(aux.anio), &(aux.mes), &(aux.dia), &(aux.hora),
                                &(aux.minuto), &(aux.segundo)};
```

```
static void (*MEF[])(void) = {fsE_YEAR, fsE_MONTH, fsE_DAY, fsE_HOUR, fsE_MINUTE, fsE_SECOND,
                              fsDEFAULT};

void MEF_Init()
{
	estado = DEFAULT;			// Estado inicial es DEFAULT
	visible = 1;				// Al editar, primero se ocultará y luego se mostrará
	veces_en_estado = 0;
}

void MEF_Update()
{
	static MEF_Estado ultimo = DEFAULT;

	if (estado == ultimo) veces_en_estado++;
	else {
		veces_en_estado = 1;
		ultimo = estado;
	}

	// Invocar a la función del estado actual
	(*MEF[estado])();
}


/* ************************************************
FUNCIONES LLAMADAS EN CADA ACTUALIZACIÓN DE MEF
* ************************************************/

/* Estado "Editando Año" */
static void fsE_YEAR(void)
{
	if (veces_en_estado == 1) toggleParametro();
	else if (veces_en_estado == OFFSET_PARPADEO) veces_en_estado = 0;

	if (KEYPAD_Update(&tecla)) switch(tecla) {
	case 'A':
		mostrarParametro();	// Visibilizar por si quedó oculto debido a parpadeo
		estado = E_MONTH;	// Transicionar al estado "Editando Mes"
		break;
	case 'B':
		incrementar();		// Incrementar valor del parámetro (controlando rango)
		break;
	case 'C':
		decrementar();		// Decrementar valor del parámetro (controlando rango)
		break;
	case 'D':
		cancelar();		// Descartar cambios
		break;
     }
}


/* Estado "Editando Mes" */
static void fsE_MONTH(void)
{
	if (veces_en_estado == 1) toggleParametro();
	else if (veces_en_estado == OFFSET_PARPADEO) veces_en_estado = 0;

	if (KEYPAD_Update(&tecla)) switch(tecla) {
	case 'A':
		mostrarParametro();	// Visibilizar por si quedó oculto debido a parpadeo
		estado = E_DAY;		// Transicionar al estado "Editando Dia"
		checkRango();	// Corregir día si no existe para el mes recién establecido
		break;
```

```
        case 'B':
                incrementar();        // Incrementar valor del parámetro (controlando rango)
                break;
        case 'C':
                decrementar();        // Decrementar valor del parámetro (controlando rango)
                break;
        case 'D':
                cancelar();           // Descartar cambios
                break;
    }
}

/* Estado "Editando Día" */
static void fsE_DAY(void)
{
        if (veces_en_estado == 1) toggleParametro();
        else if (veces_en_estado == OFFSET_PARPADEO) veces_en_estado = 0;

        if (KEYPAD_Update(&tecla)) switch(tecla) {
        case 'A':
                mostrarParametro();   // Visibilizar por si quedó oculto debido a parpadeo
                estado = E_HOUR;      // Transicionar al estado "Editando Hora"
                checkRango(); // Corregir día si no existe para el mes recién establecido
                break;
        case 'B':
                incrementar();        // Incrementar valor del parámetro (controlando rango)
                break;
        case 'C':
                decrementar();        // Decrementar valor del parámetro (controlando rango)
                break;
        case 'D':
                cancelar();           // Descartar cambios
                break;
    }
}

/* Estado "Editando Hora" */
static void fsE_HOUR(void)
{
        if (veces_en_estado == 1) toggleParametro();
        else if (veces_en_estado == OFFSET_PARPADEO) veces_en_estado = 0;

        if (KEYPAD_Update(&tecla)) switch(tecla) {
        case 'A':
                mostrarParametro();   // Visibilizar por si quedó oculto debido a parpadeo
                estado = E_MINUTE;    // Transicionar al estado "Editando Minutos"
                break;
        case 'B':
                incrementar();        // Incrementar valor del parámetro (controlando rango)
                break;
        case 'C':
                decrementar();        // Decrementar valor del parámetro (controlando rango)
                break;
        case 'D':
                cancelar();           // Descartar cambios
                break;
    }
}

/* Estado "Editando Minutos" */
static void fsE_MINUTE(void)
{
        if (veces_en_estado == 1) toggleParametro();
        else if (veces_en_estado == OFFSET_PARPADEO) veces_en_estado = 0;
```

```
        if (KEYPAD_Update(&tecla)) switch(tecla){
        case 'A':
               mostrarParametro();   // Visibilizar por si quedó oculto debido a parpadeo
               estado = E_SECOND;    // Transicionar al estado "Editando Segundos"
               break;
        case 'B':
               incrementar();        // Incrementar valor del parámetro (controlando rango)
               break;
        case 'C':
               decrementar();        // Decrementar valor del parámetro (controlando rango)
               break;
        case 'D':
               cancelar();           // Descartar cambios
               break;
     }
}

/* Estado "Editando Segundos" */
static void fsE_SECOND(void)
{
        if (veces_en_estado == 1) toggleParametro();
        else if (veces_en_estado == OFFSET_PARPADEO) veces_en_estado = 0;

        if (KEYPAD_Update(&tecla)) switch(tecla){
        case 'A':
               mostrarParametro();        // Visibilizar por si quedó oculto debido a parpadeo
               RELOJ_updateCurrent(&aux);   // Se hacen efectivos los cambios
               estado = DEFAULT;            // Transicionar al estado por defecto
               break;
        case 'B':
             incrementar();          // Incrementar valor del parámetro (controlando rango)
               break;
        case 'C':
               decrementar();        // Decrementar valor del parámetro (controlando rango)
               break;
        case 'D':
               cancelar();           // Descartar cambios
               break;
     }
}

/* Estado por defecto */
static void fsDEFAULT(void)
{
        if (veces_en_estado == 1) mostrarFechaHora();
        else if (veces_en_estado == SEGUNDO) veces_en_estado = 0;

        // Transicionar al estado "Editando Año"
        if (KEYPAD_Update(&tecla) && tecla == 'A') estado = E_YEAR;
}


/* ***************************************************
FUNCIONES CON ACCIONES COMUNES PARA VARIOS ESTADOS
*************************************************** */

/* Descarta los cambios de fecha y hora, y muestra los valores originales */
static void cancelar()
{
        estado = DEFAULT;            // Transicionar al estado por defecto
        visible = 1;
}
```

```
/* Corrige el parámetro en edición respecto a sus rangos válidos */
static void checkRango()
{
	static uint8_t maximos[] = {199, 12, 31, 23, 59, 59};
	static const int8_t minimos[] = {0, 1, 1, 0, 0, 0};

	// Si se está editando el dia...
	if (estado == E_DAY)
	{
		// Actualizar máximo para el mes y año asignado
		maximos[estado] = UTIL_DiaMaximo(aux.mes, aux.anio);
	}

	uint8_t* actual = parametros[estado];
	// Setear valor máximo si es menor al mínimo
	if (*actual < minimos[estado] || *actual == 255) *actual = maximos[estado];
	// Setear valor mínimo si es mayor al máximo
	else if (*actual > maximos[estado]) *actual = minimos[estado];
}

/* Resta 1 al valor del parámetro en edición, respetando el rango */
static void decrementar()
{
	(*parametros[estado])--;		// Decrementar parámetro
	checkRango();				// Setear valor máximo si se salió de rango
	mostrarParametro();			// Forzar escritura del campo
	veces_en_estado = 1;			// Evitar que se oculte rápidamente
}

/* Suma 1 al valor del parámetro en edición, respetando el rango */
static void incrementar()
{
	(*parametros[estado])++;		// Incrementar parámetro
	checkRango();				// Setear valor mínimo si se salió de rango
	mostrarParametro();			// Forzar escritura del mismo
	veces_en_estado = 1;			// Evitar que se oculte rápidamente
}

/* Mostrar la fecha y hora actual en el LCD */
static void mostrarFechaHora()
{
	RELOJ_getCurrent(&aux);

	LCDGotoXY(4,0);			// Escribir hora, minuto y segundo en la fila superior
	LCDescribeDato(aux.hora,2);
	LCDsendChar(':');
	LCDescribeDato(aux.minuto,2);
	LCDsendChar(':');
	LCDescribeDato(aux.segundo,2);

	LCDGotoXY(4,1);				// Escribir dia, mes y año en la fila inferior
	LCDescribeDato(aux.dia,2);
	LCDsendChar('/');
	LCDescribeDato(aux.mes,2);
	LCDsendChar('/');
	LCDescribeDato(aux.anio,2);
}

/* Visibiliza el parámetro que se está editando */
static void mostrarParametro()
{
	LCDGotoXY(x[estado], y[estado]);		// Situar cursor
	LCDescribeDato(*parametros[estado], 2);		// Escribir número con 2 dígitos
	visible = 1;					// Indicar que está visible
}
```

```
/* Oculta el parámetro que se está editando */
static void ocultarParametro()
{
	LCDGotoXY(x[estado], y[estado]);		// Situar cursor
	LCDsendChar(' ');				// Borrar decena
	LCDsendChar(' ');				// Borrar unidad
	visible = 0;					// Indicar que no está visible
}

/* Alterna la visibilidad del parámetro en edición */
static void toggleParametro()
{
	if (visible) ocultarParametro();		// Si está visible, ocultarlo
	else mostrarParametro();			// Sino, mostrarlo
}
```