Circuitos Digitales y Microcontroladores (E0305)
Mayo 2020

TRABAJO PRACTICO NRO. 2

Autor:
Nehuen Pereyra (878/6)

1. Interpretación del problema

El problema a resolver utilizando el microcontrolador MC9S08SH8 es poder ver en memoria (por falta del LCD incluido del kit de desarrollo) un reloj en formato HH:MM:SS y poder modificar sus valores de hora, minutos y segundos a través de un teclado matricial. Las funcionalidad a implementar son: ver en memoria un reloj en formato HH:MM:SS y ajustar la hora, minutos o segundos.

1.1. Detalle del problema a resolver

El reloj se debe iniciar con el valor 12:00:00 y comienza a mostrar el tiempo, luego con el teclado matricial se pueden presionar las teclas: A para cambiar la hora, B para los minutos y C para los segundos. Mientras se está cambiando uno de estos parámetros el dígito a ingresar se debe visualizar parpadeando (el parpadeo debe suceder cada segundo), los números ingresados por teclado no se pueden borrar y al finalizar de ingresarlos para confirmar el ingreso se debe presionar nuevamente la letra que se utilizó para comenzar el proceso de modificación, para luego visualizar el reloj con la nueva hora ingresa. El proceso se puede cancelar en cualquier momento presionando la letra D. Mientras se realiza el cambio el reloj sigue funcionando por si se descarta el cambio. La visualización de la hora se realiza a través de la herramienta View Memory en formato ASCII. Mientras que para simular el teclado matricial se trabaja sobre las entradas del microcontrolador desde el ChipView. Para resolver este problema tendremos que utilizar el módulo RTC que brinda el microcontrolador para generar una base de tiempo en base a las interrupciones periódicas que genera. Para controlar el momento del ingreso de los datos de la hora, minutos o segundos tendremos que implementar una máquina de estados, la cual en estado por default solo muestra el tiempo y cuando el usuario presiona alguna de las teclas mencionadas anteriormente cambie el estado al que corresponda (en la sección de la máquina de estados finita se detalla el funcionamiento) para luego de la confirmación o cancelación de la operación se realicen los cambios sobre el reloj y se vuelva al estado por default con los nuevos valores del tiempo.

2. Implementación

Se intentó de modularizar todo el problema para hacerlo más independiente y reutilizable. Se separaron en dos carpetas los distintos módulos que utilizaremos, la primera llamada io de (input/output) donde encontramos:

- **El teclado**: Se configura aquí el teclado matricial para su inicialización y funcionamiento.
- **Un buffer de entrada**: Este sirve para ir almacenando las teclas presionadas por el usuario y que luego serán accedidas para realizar acciones.
- **Una Terminal**: En donde se podrá visualizar la salida de la hora.

La segunda carpeta llamada Lib posee:

- **El Reloj**: Aquí se modela el reloj el cual se puede visualizar, modificar y obtener la hora actual.
- **Las Tareas**: Este módulo se encarga de planificar y despachar las tareas. En base a las distintas tareas que se van a estar ejecutando en concurrente este módulo se encarga de activar los flags de las distintas tareas y controlar que una vez se finalicen sean limpiados los flags.

- **La máquina de estados finitos**: Este módulo se encarga de realizar el control de la máquina de estado la cual se debe definir y luego inicializar. Desde el planificador de tarea se llama a actualizar la máquina de estados cada cierto tiempo que en el apartado de planificación se detalla en la siguiente sección.

El archivo main se encarga de inicializar el microcontrolador y los distintos módulos para luego quedar ejecutando las interrupciones que se van generando por el RTC.

2.1. Planificación

Como el programa consiste en un conjunto de tareas que se ejecutan concurrentemente de manera cooperativa (no hay formas de interrumpirlas) cada una de ellas va cediendo el control. Todas utilizan el mismo buffer de entrada, en el cual el teclado va encolando las teclas presionadas por el usuario y luego desde el programa pueden acceder o desencolar la tecla. Se explicará con más detalle el funcionamiento del teclado, el buffer y la terminal donde se muestra la hora en sus respectivas secciones.

Las tareas a ejecutar son:

- Incremento de la hora del reloj.
- Lectura del keypad si corresponde cargar en el buffer.
- Actualizar el estado de la máquina de estados finitos.
- Ejecutar la tarea principal.

Estas se pueden observar en el archivo tasks.c que en la última sección se encuentra. Como se mencionó anteriormente este módulo además de despachar las tareas también planifica cuando lanzarlas para esto se utilizó el reloj de tiempo real del microcontrolador (descrita en la siguiente sección) como generador de interrupciones periódicas y se eligió un periodo de 100 ms, ya que el tiempo de respuesta del dispositivo no necesita ser menor a 100 ms. Se podría haber puesto un menor tiempo de respuesta, pero esto aumenta el riesgo de que la CPU reciba una interrupción cuando aún no ha terminado de atender la actual, provocando un desfase en la hora del dispositivo.

Pseudocódigo de la implementación de schedulet_tasks() y dispach_tasks().

```
1  // Se ejecuta cada 100ms
2  Función schedule_tasks()
3          Incrementar contador de reloj
4          Si el contador de reloj es igual a 10
5                  Establecer el contador en cero
6                  Activar flag de reloj
7                  Activar flag de parpadeo
8          Activar flag de la maquina de estados
9          Activar flag para realizar el barrido del teclado
10 Fin schedule_tasks()
```

Pseudocódigo de schedulet_tasks().

```
1  // Se ejecuta cada 100ms
2  Función dispach_tasks()
3          Si el flag del reloj esta activo
4                  Actualizar reloj
5                  Avisar que fue atendido el pedido de interrupción por parte del reloj
6          Si el flag del barrido del teclado esta activo
7                  Realizar barrido del teclado y almacenar la tecla en el buffer
8                  Avisar que fue atendido el pedido de interrupción
9          Si el flag de la maquina de estado esta activo
10                 Actualizar el estado de la maquina de estados finitos
11                 Avisar que fue atendido el pedido de interrupción por parte de la
                       mef
12 Fin dispach_tasks()
```

Pseudocódigo de dispach_tasks().

2.2. RTC (Reloj de tiempo real)

Se utilizó del módulo RTC (Real Time Counter) del microcontrolador, para generar las interrupciones periódicas. El RTC está formado por un contador de ocho bits, un comparador de ocho bits, predivisores de frecuencia binarios y decimales, dos fuentes de clock y una interrupción periódica programable.

El RTC puede utilizar tres fuentes de clock para su funcionamiento:

- Oscilador de bajo consumo de 1 kHz.
- Clock externo.
- Clock interno de 32 kHz.

Se puede configurar mediante un conjunto de registros. Cada registro está compuesto por ocho bits y en un mismo registro hay grupos de bits relacionados. A continuación, se listan los registros y los grupos de bits de cada uno, empezando por el bit menos significativo:

Nombre		7	6	5	4	3	2	1	0
RTCSC	R	RTIF	RTCLKS		RTIE	RTCPS			
	W								
RTCCNT	R	RTCCNT							
	W	-							
RTCMOD	R	RTCMOD							
	W								

Figura 1: Registros del módulo RTC.

- **RTC Status and Control Register (RTCSC)**: contiene cuatro bits para configurar el predivisor (RTCPS), un bit para habilitar la interrupción (RTIE), dos bits para seleccionar la fuente de reloj (RTCLKS) y el flag de estado de la interrupción (RTIF).
- **RTC Counter Register (RTCCNT)**: contiene el valor del contador de 8 bits.
- **RTC Modulo Register (RTCMOD)**: contiene el valor que al tomar RTCCNT se emite una interrupción.

Como fuente de reloj se eligió la interna de 1kHz que tiene un error del 20 %. Para reducir este error se podría utilizar un oscilador de cuarzo externo, pero por lo pedido por la catedra se considera suficiente utilizar el reloj interno. Para esto RTCLKS debe tomar el valor 0.

La frecuencia de interrupciones en función a la frecuencia del clock está dada por:

$$F_{TIF} = \frac{f_{source}}{PREDIV \cdot (RTCMOD + 1)} \tag{1}$$

Se necesita temporizar 100 ms.

$$10Hz = \frac{1000Hz}{PREDIV \cdot (RTCMOD + 1)} \tag{2}$$

Eligiendo RTCMOD = 0, se obtiene que PREDIV debe valer 100.

Para habilitar la interrupción, se setea RTIE a 1.

Estas configuraciones no se realizaron de forma manual, sino que se utilizó la funcionalidad Device Initialization que provee CodeWarrior. Se puede ver en la figura 2 la configuración del RTC con las configuraciones mencionadas previamente ya ingresadas.

Por último, se establece cuál es la función que maneja la interrupción. El asistente de Device Initialization crea una función llamada isrVrtc() Dentro de esta función se hace dos llamadas a las funciones: schedule_tasks() y dispach_tasks() las cuales estan implementadas en tasks.c luego establece RTIF en uno para indicar que esa interrupción ya fue atendida y dar paso a nuevas interrupciones, se puede ver en el siguiente codigo:

```
1  /* Se ejecuta cada 100 ms */
2  __interrupt void isrVrtc(void)
3  {
4          schedule_tasks();
5          dispach_tasks();
6          RTCSC_RTIF = 1;
7  }
8  /* end of isrVrtc */
```

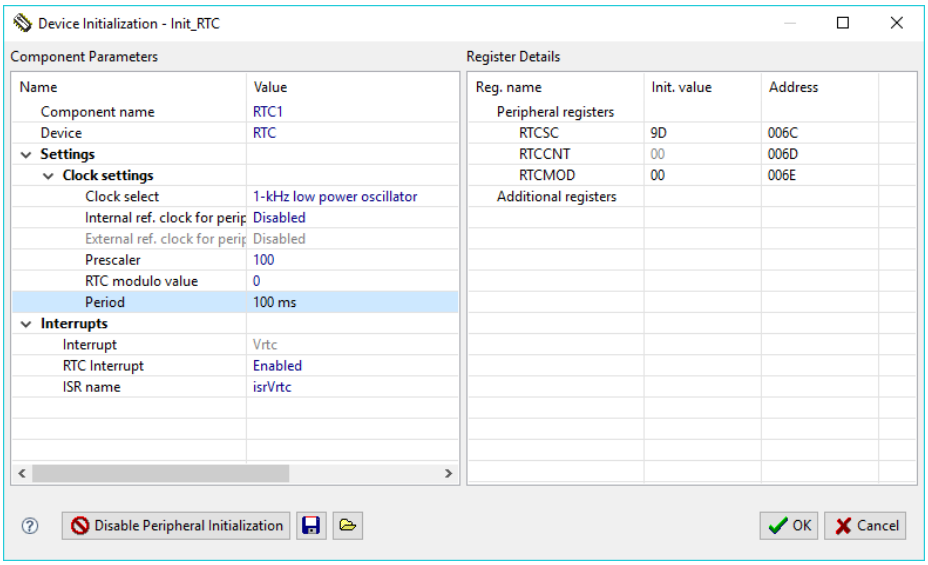


Figura 2: Configuración del módulo RTC.

2.3. Keypad

El teclado matricial de 4x4 teclas mostrado por la cátedra posee un conector con ocho pines (cuatro para las filas y cuatro para las columnas), en la figura 3 podemos observar el teclado y como se conecta al microcontrolador.

Para detectar la tecla pulsada se utiliza el método de barrido (el cual requiere analizar las filas y la respuesta de las columnas) como ya conocemos el patrón que sigue al ir rotando un valor bajo por la salida y su reflejo en las entradas, podremos detectar la tecla en base a esto. Entonces a la hora de utilizar el keypad tenemos una parte de inicialización y otra de detección en la primera debemos configurar la parte baja de PTB como salida [0:3] y la parte alta como entrada [4:7] (PTBDD = 0x0F) por lo tanto, hay que habilitar las resistencias de pull-up para las entradas (PTBPE = 0xF0) y como vamos a trabajar desde el simulador podemos no tener en cuenta el efecto del rebote, pero a la hora de implementarlo con el kit real se debe considerar este problema y resolverlo como en el informe 1. Mientras en la segunda parte detectamos si existe o no una tecla presionada. Estas dos partes están definidas en la cabecera del módulo Keypad.

2.4. Buffer

Se incorporó al sistema un buffer circular el cual permite almacenar los caracteres ingresados por teclado. Por lo tanto cuando desde algún módulo se requiera acceso a valores de las teclas ingresadas por el usuario van a poder ser accedidas desde este buffer. La siguiente lista presenta su interfaz:

- **buffer_peek()**: Retorna el primer elemento agregado a la cola, sin quitarlo. En particular, se utiliza para saber qué carácter se presionó y de ese modo actualizar la maquina de estados finitos.
- **buffer_pop()**: Saca de la cola el carácter más antiguo ingresado por teclado.

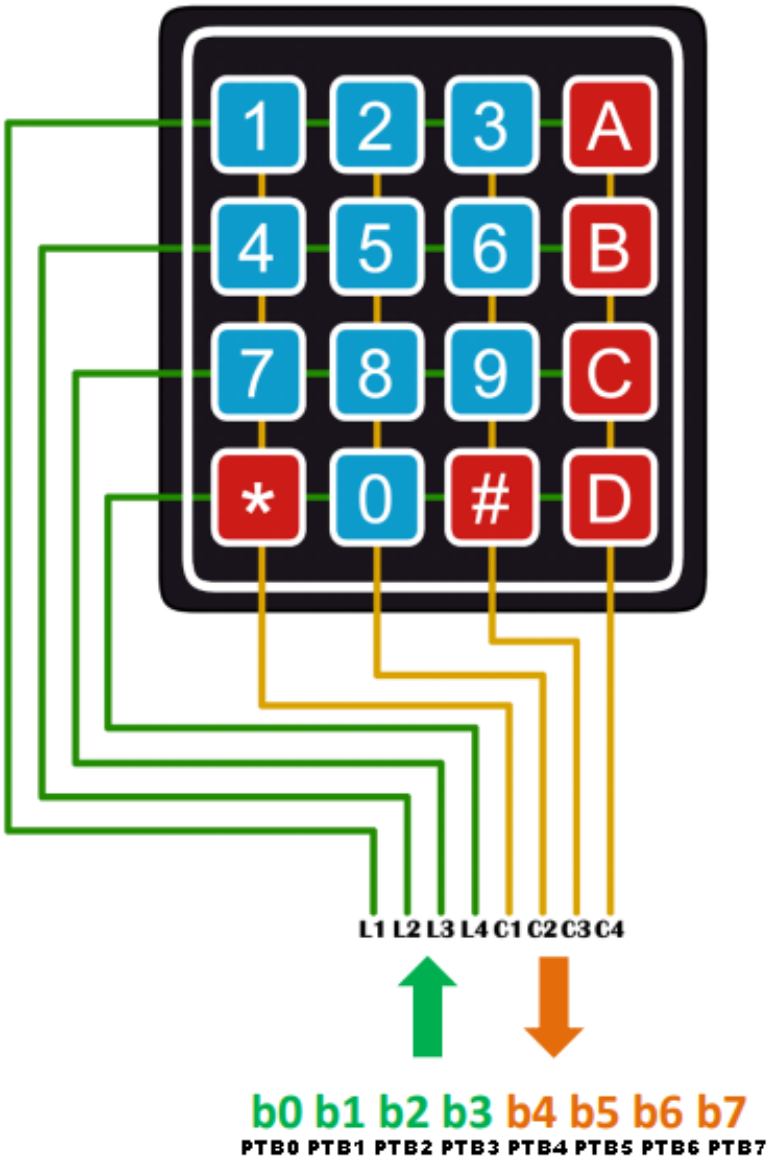


Figura 3: Conexión del teclado matricial con el PORTB del MCU.

- **buffer_push()**: Agrega un elemento a la cola. Para este problema, se utiliza para agregar el último carácter ingresado por teclado.
- **buffer_empty()**: Determina si hay caracteres ingresados hasta el momento.

2.5. Terminal

Este es el módulo más simple del sistema y viene a ser un reemplazo del LCD para mostrar los datos que nos pide el problema. Este módulo está compuesto por un arreglo de 9 posiciones llamado linea (las necesarias para poder mostrar el formato HH:MM:SS además del fin linea). La siguiente lista presenta su interfaz:

- **terminal_init()**: Permite enviar un string para inicializar el arreglo con ese vector.
- **terminal_set_string()**: Permite setear un string.
- **terminal_set_ digit()**: Permite enviar un dígito y un índice para establecer el arreglo en esa posición del índice enviado el valor del dígito indicado.

Para ver el valor de la linea debemos utilizar la herramienta de View Memory sobre esa variable y elegir el formato ASCII para visualizarlo correctamente.

2.6. Maquina de estados finitos

Esta tarea implementa el control del ajuste de la hora del reloj. Se implementa con una máquina de estados finitos que se ejecuta en simultáneo con el resto de las tareas (mientras se ejecuta cualquiera de los estados, el reloj se seguirá actualizando). El ajuste del reloj consiste en dos funcionalidades principales:

- Actualizar la hora y mostrarla por pantalla.
- Ajustar la hora según los datos ingresados por el usuario.

La primera funcionalidad consiste en el estado S0 que es el estado por default en donde a través de la terminal se va ir mostrando la hora en formato HH:MM:SS. Cuando el usuario presione una tecla dependiendo cual sea (ver figura 4) se pasa al estado correspondiente (S1 para el cambio de hora, S2 para el cambio de minutos y S3 para el cambio de segundos) en donde comienza a titilar cada 1 segundo el dígito a ingresar (el usuario tiene 2 minutos para ingresar los dígitos, si pasas ese tiempo se vuelve al estado S0) una vez ingresados los dígitos se comprueba que se confirme la operación apretando la tecla correspondiente al cambio a realizar para luego volver al estado S0 con la hora ajustada al nuevo valor, este sería el flujo normal del programa, si en cambio el usuario presiona la letra D en cualquier momento, se pasa del estado actual a S0 manteniendo la hora sin modificar. Además, se debe tener en cuenta que una hora válida es un número entre 0 a 23, mientras que los segundos y minutos tienen que ser un número entre 0 y 59. El usuario al entrar en uno de los estados de modificación sea de la hora, minutos o segundos solo se van a detectar las teclas correspondientes a dígitos o la confirmación de la operación, si ingresa un número no válido al confirmar la operación, se pasa al estado S0 sin modificar el ajuste del reloj. Mientras que se guardaran los números en variables internas al reloj, solo cuando estos sean válidos.

Pseudocódigo de la implementación de ingreso_digitos() y mef_actualizar().

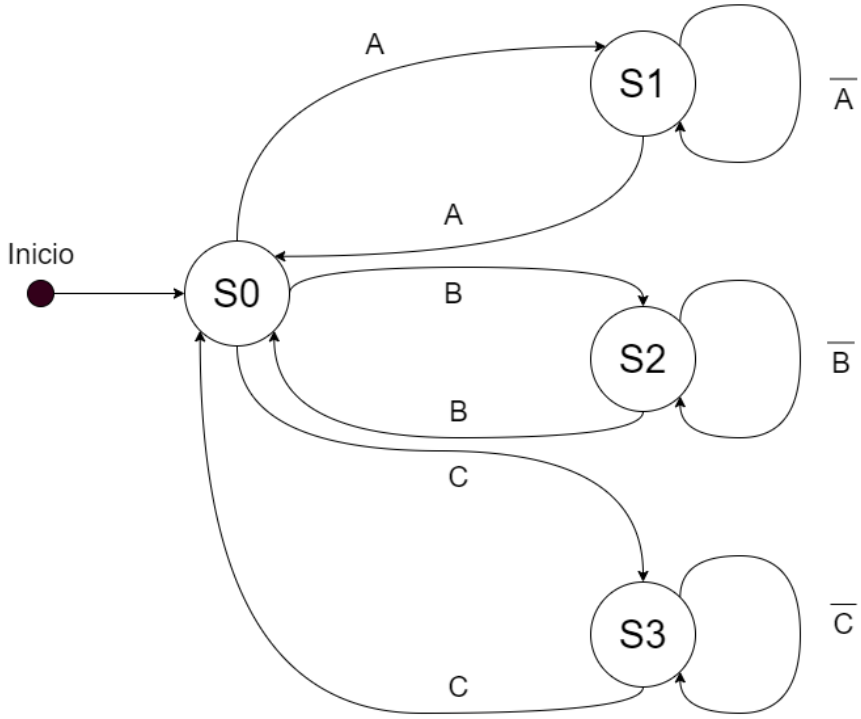


Figura 4: Estados de la maquina de estados finitos.

```
1  // El parametro tiempo es un indice para posicionarse en la terminal para mostrar o titilar
       el digito
2  Función ingreso_digitos(tiempo, key)
3          Se obtiene un caracter de buffer de entrada
4          Se comprueba que no se halla excedido el tiempo maximo de ingreso de caracteres
5          Comienza a titilar el digito correspondiente al caracter a ingresar, utilizando la
               posición en base al parametro: tiempo más la cantidad de elementos del arreglo
                temporal
6          Si el caracter leido es un numero y no se a excedido los 2 caracteres en el arreglo
               temporal
7                  Se almacena en un arreglo temporal el caracter
8                  Se muestra en la terminal el caracter ingresado, utilizando la posición en
                       base al parametro:tiempo más la cantidad de elementos del arreglo
                       temporal
9          Sino si el caracter es igual a la tecla de confirmación (parametro: key) y el arreglo
               temporal posee 2 caracteres
10                 Se retorna el valor uno
11         Sino el caracter leido es D
12                 Se retorna el valor menos uno
13         Sino no se a precionado caracter o no es uno de los indicados previamente
14                 Se retorna el valor cero
15 Fin ingreso_digitos()
```

Pseudocódigo de ingreso_digitos().

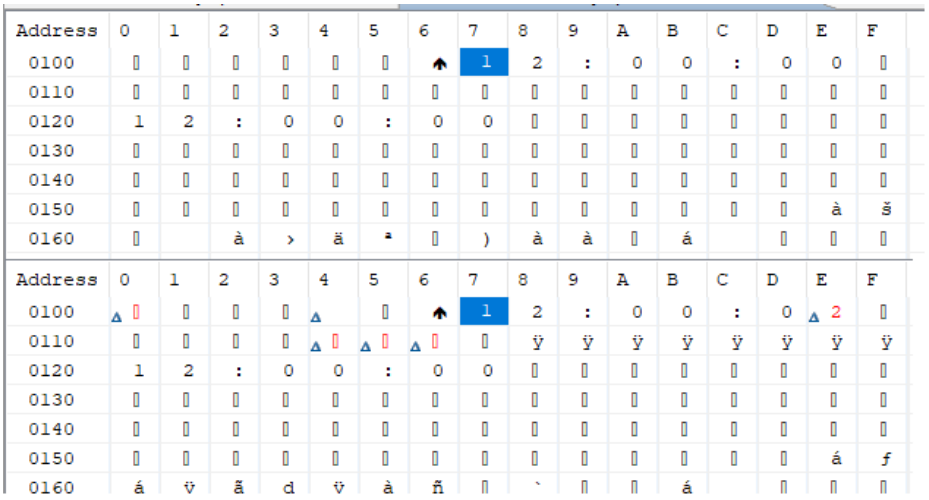
```
1  Función mef_actualizar()
2          Se lee un caracter del buffer de entrada
3          Se incrementa el contador de tiempo maximo de ingreso de caracter
4          Si el estado actual de la mef es S0
5                  Comprueba si el caracter es alguna de las teclas para ajustar la hora y
                       cambia al estado correspondiente
6          Si el estado actual de la mef es S1
7                  Llama y guarda el valor que retorna ingresar_digitos() pasando como
                       parametro la posición de los digitos de la hora y la tecla de confirmació
                       n que en este caso es la A
8                  Si los digitos ingresados son validos actualiza la hora
9                  Si se ingreso la letra D establece el estado en S0
10         Si el estadoa ctual de la mef es S2
11                 Llama y guarda el valor que retorna ingresar_digitos() pasando como
                       parametro la posición de los digitos de los minutos y la tecla de
                       confirmación que en este caso es la B
12                 Si los digitos ingresados son validos actualiza los minutos
13                 Si se ingreso la letra D establece el estado en S0
14         Si el estado actual de la mef es S3
15                 Llama y guarda el valor que retorna ingresar_digitos() pasando como
                       parametro la posición de los digitos de los segundos y la tecla de
                       confirmación que en este caso es la C
16                 Si los digitos ingresados son validos actualiza los segundos
17                 Si se ingreso la letra D establece el estado en S0
18         En cualquiera de los casos
19                 Actualiza la visualización del reloj
20 Fin mef_actualizar()
```

Pseudocódigo mef_actualizar().

3. Validación

Para verificar el funcionamiento mostraremos 3 etapas esenciales: el reloj funcionando, el cambio de la hora (que se puede aplicar también a los minutos como a los segundos) y la cancelación de la operación por parte del usuario.

En la figura 5 podemos ver al reloj funcionando en la View Memory en donde se van incrementando los segundos.



Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0100	▯	▯	▯	▯	▯	▯	▯	1	2	:	0	0	:	0	0	▯
0110	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯
0120	1	2	:	0	0	:	0	0	▯	▯	▯	▯	▯	▯	▯	▯
0130	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯
0140	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯
0150	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	à	š
0160	▯		à	›	ä	ª	▯	)	à	à	▯	á		▯	▯	▯

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0100	▯	▯	▯	▯	▯	▯	▯	1	2	:	0	0	:	0	2	▯
0110	▯	▯	▯	▯	▯	▯	▯	▯	ÿ	ÿ	ÿ	ÿ	ÿ	ÿ	ÿ	ÿ
0120	1	2	:	0	0	:	0	0	▯	▯	▯	▯	▯	▯	▯	▯
0130	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯
0140	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯
0150	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	á	ƒ
0160	á	ÿ	ã	d	ÿ	à	ñ	▯	`	▯	▯	á		▯	▯	▯

Figura 5: Funcionamiento normal del reloj.

En la figura 6 se puede ver que al ingresar el valor A (01111110) empieza a parpadear el primer dígito ingresando el valor 2 (11011110) y luego el segundo dígito empieza a parpadear indicando que se debe ingresar este valor por lo que ingreso el valor 3 (10111110), una vez ingresado confirmó la operación ingresando A (01111110).

En la figura 7 se puede observar la cancelación de la operación de cambiar la hora, primero ingreso para cambiar la hora, presionando A (01111110) comienza a parpadear el primer dígito entonces luego apretó D (01110111) para volver al estado S0 o por default donde se muestra nuevamente el reloj.

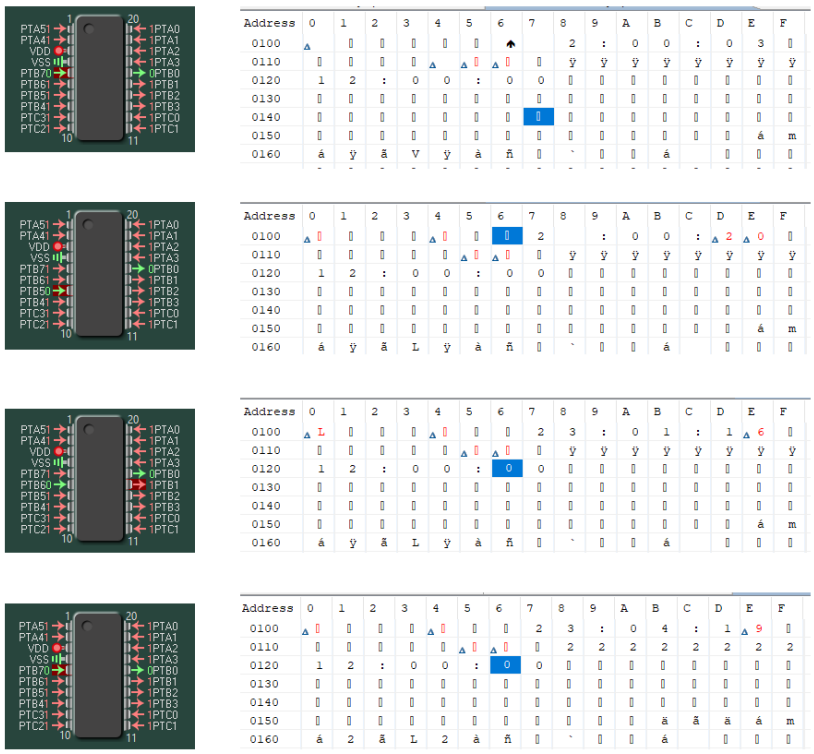


Figura 6: Modificación de la hora.

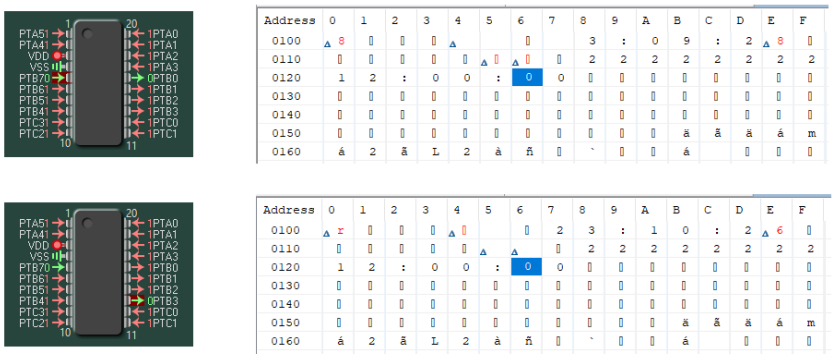


Figura 7: Cancelación del ingreso de la hora.

4. Apéndice: Código fuente

Módulos de la sección entrada salida

4.1. keypad.h

```
1  #ifndef KEYPAD_H_
2  #define KEYPAD_H_
3
4  #include <mc9s08sh8.h>
5  #include "type.h"
6  #include "derivative.h"
7
8  void keypad_init(void);
9  uint8_t keypad_scan(uint8_t *pkey);
10
11
12 #endif /* KEYPAD_H_ */
```

4.2. keypad.c

```
1  #include "derivative.h" /* include peripheral declarations */
2  #include "type.h"
3  #include "io/keypad.h"
4
5  #define TECLADO_MATRICIAL PTBD
6
7  /* Inicializa el teclado */
8  void keypad_init(){
9          PTBDD = 0x0F;
10         PTBPE = 0xF0;
11 }
12
13 /* Parametros: Recive como parametro un puntero a un caracter donde retorna la
       tecla precionada
14  * Devuelve: Si se preciono una tecla un 0 y si no se preciono ninguna un -1
15  * */
16
17 uint8_t barrido(uint8_t *pressed_key){
18
19         uint8_t key = -1;
20         switch(TECLADO_MATRICIAL){
21                                 case 0b11101110:
22                                         key='1';
23                                         break;
24                                 case 0b11011110:
25                                         key='2';
26                                         break;
```

```
27                                      case 0b10111110:
28                                              key='3';
29                                              break;
30                                      case 0b01111110:
31                                              key='A';
32                                              break;
33                                      case 0b11101101:
34                                              key='4';
35                                              break;
36                                      case 0b11011101:
37                                              key='5';
38                                              break;
39                                      case 0b10111101:
40                                              key='6';
41                                              break;
42                                      case 0b01111101:
43                                              key='B';
44                                              break;
45                                      case 0b11101011:
46                                              key='7';
47                                              break;
48                                      case 0b11011011:
49                                              key='8';
50                                              break;
51                                      case 0b10111011:
52                                              key='9';
53                                              break;
54                                      case 0b01111011:
55                                              key='C';
56                                              break;
57                                      case 0b11100111:
58                                              key='*';
59                                              break;
60                                      case 0b11010111:
61                                              key='0';
62                                              break;
63                                      case 0b10110111:
64                                              key='#';
65                                              break;
66                                      case 0b01110111:
67                                              key='D';
68                                              break;
69                              }
70
71              if(key != -1){
72                      *pressed_key = key;
73                      return 0;
74              }
75              return key;
76      }
```

4.3. buffer.h

```
1  #ifndef BUFFER_H_
2  #define BUFFER_H_
3
4  #include "type.h";
5  #define BUFFER_VACIO 'X';
6
7  uint8_t buffer_peek(void);
8  uint8_t buffer_pop(void);
9  void buffer_push(uint8_t);
10 uint8_t buffer_empty(void);
11
12 #endif /* BUFFER_H_ */
```

4.4. buffer.c

```
1  #include "io/buffer.h";
2
3  /* Tama del Buffer */
4  #define TAMANIO 8
5
6  /* Buffer configurado como una cola. */
7  static uint8_t buffer[TAMANIO];
8
9  /* Indice de la posicion del ultimo elemento. */
10 static uint8_t ultimo = 0;
11
12 /* Indice del primer elemento agregado. */
13 static uint8_t primero = 0;
14
15 /* Cantidad de elementos en el buffer*/
16 static uint8_t cantidad = 0;
17
18
19 /*
20  * Retorna el primer elemento agregado al buffer, sin quitarlo.
21  * Devuelve: El primer caracter agregado a la cola.
22  */
23 uint8_t buffer_peek(void) {
24         if (cantidad == 0){
25                 return BUFFER_VACIO;
26         }
27         else
28                 return buffer[primero];
29 }
30
31 /*
32  * Saca el primer elemento agregado a la cola.
```

```
33  * Devuelve: El primer caracter agregado a la cola.
34  */
35 uint8_t buffer_pop(void) {
36         uint8_t caracter;
37         if (cantidad == 0){
38                 return BUFFER_VACIO;
39         }
40         else {
41                 cantidad--;
42                 caracter = buffer[primero];
43                 primero = (primero + 1) & 0x7;
44                 return caracter;
45         }
46 }
47
48 /*
49  * Agrega un elemento a la cola.
50  * Parametro:
51  * caracter: Caracter a agregar.
52  */
53 void buffer_push(uint8_t cararacter) {
54         if(cararacter == 0)
55                 return;
56         buffer[ultimo] = cararacter;
57         cantidad++;
58         if (cantidad > TAMANIO)
59         {
60                 cantidad = TAMANIO;
61                 primero = (uint8_t)(primero + 1) & 0x7;
62         }
63     ultimo = (uint8_t)(ultimo + 1) & 0x7;
64 }
65
66 /*
67  * Comprueba si el buffer esta vacio.
68  * Devuelve: Valor uno si esta vacia, cero caso contrario.
69  */
70 uint8_t buffer_empty(void) {
71         return !cantidad;
72 }
```

4.5. terminal.h

```
1  #ifndef TERMINAL_H_
2  #define TERMINAL_H_
3  #include "type.h";
4
5  uint8_t terminal_init(uint8_t *string);
6  uint8_t terminal_set_string(uint8_t *string);
7  uint8_t terminal_set_digit(uint8_t a, uint8_t b);
```

```
8
9
10 #endif /* TERMINAL_H_ */
```

4.6. terminal.c

```
1  #include "io/terminal.h"
2  #include "type.h"
3  #include <string.h>
4
5  /* Representa la linea a mostrar en el formato HH:MM:SS */
6  static uint8_t linea[9];
7
8  /* Recibe como parametro una cadena de caracteres
9   * Devuelve: el tama de la cadena
10  * */
11 uint8_t size(uint8_t *arr){
12         return (sizeof arr / sizeof *arr);
13 }
14
15 /* Inicializa la terminal con una cadena recibida por parametro */
16 uint8_t terminal_init(uint8_t* string){
17         if(size(string)<9){
18                 strcpy(linea,string);
19                 return 0;
20         }
21         return -1;
22 }
23
24 /* Setea el valor de linea con la cadena recibida por parametro
25  * Devuelve: Si la cadena era del tama correcto retorna 0 sino -1
26  * */
27 uint8_t terminal_set_string(uint8_t* string){
28         if(size(string)<9){
29                 strcpy(linea,string);
30                 return 0;
31                 }
32         return -1;
33 }
34
35 /* Setea el digito en la posici de index recibido por parametro
36  * Devuelve: Si el indice estaba dentro del rango con un 0 sino un -1
37  * */
38 uint8_t terminal_set_digit(uint8_t digit, uint8_t index){
39         if(index < 9){
40                         linea[index] = digit;
41                         return 0;
42                 }
43         return -1;
44 }
```

Módulos de la sección Lib

4.7. tasks.h

```
1  #ifndef TASKS_H_
2  #define TASKS_H_
3
4  #include "type.h"
5  #include "lib/reloj.h"
6  #include "lib/mef.h"
7  #include "io/keypad.h"
8  #include "io/buffer.h"
9
10 void schedule_tasks(void);
11 void dispach_tasks();
12
13 #endif /* TASKS_H_ */
```

4.8. tasks.c

```
1  #include "lib/tasks.h";
2
3  /* Flags a controlar */
4  volatile uint8_t flagReloj = 0;
5  volatile uint8_t flagTitileo = 0;
6  volatile uint8_t flagKeypad = 0;
7  volatile uint8_t flagMef = 0;
8
9  /* Determina a que tiempo se van a ejecutar cada tarea*/
10 static uint8_t contador_reloj = 0;
11
12 /* Planifica las tareas a ejecutar con su respectivo tiempo*/
13 void schedule_tasks(void){
14         contador_reloj++;
15
16         /* Se activa el flag del reloj cada 1 segundo */
17         if(contador_reloj==10){
18                 contador_reloj = 0;
19                 flagReloj = 1;
20                 flagTitileo=1;
21         }
22
23         flagMef = 1;
24         flagKeypad = 1;
25 }
26
27 /* Se encarga de ejecutar las tareas*/
```

```
28 void dispach_tasks(){
29         uint8_t key;
30
31         if(flagReloj==1){
32                 reloj_actualizar();
33                 flagReloj = 0;
34          }
35
36          if(flagKeypad==1){
37                 if(barrido(&key)!=-1){
38                         buffer_push(key);
39                 }
40                 flagKeypad=0;
41          }
42
43          if(flagMef==1){
44                 mef_actualizar();
45                 flagMef=0;
46          }
47 }
```

4.9. reloj.h

```
1  #ifndef RELOJ_H_
2  #define RELOJ_H_
3
4  #include "type.h"
5  #include <string.h>
6  #include <stdio.h>
7  #include <stdlib.h>
8
9  void reloj_init(void);
10 void reloj_actualizar(void);
11 uint8_t reloj_modificar_segundo(uint8_t a, uint8_t b);
12 uint8_t reloj_modificar_minuto(uint8_t a, uint8_t b);
13 uint8_t reloj_modificar_hora(uint8_t a, uint8_t b);
14 uint8_t* reloj_get_string();
15
16 #endif /* RELOJ_H_ */
```

4.10. reloj.c

```
1  #include "lib/reloj.h";
2
3  /* Almacena los valores de hora, minutos y segundos del reloj */
4  static uint8_t reloj[] = {0,0,0};
5
```

```
6  /* Realiza la conversi de string a entero de dos caracteres enviados por parametro */
7  uint8_t strToInt(uint8_t a, uint8_t b){
8          uint8_t d0 = a-48, d1 = b-48;
9          if((d0>=0)&&(d1>=0)&&(d0<=9)&&(d1<=9)){
10                 return (((d0)*10)+(d1));
11         }
12         return -1;
13 }
14
15 /* Inicializa el reloj */
16 void reloj_init(void){
17         reloj[0] = 0;
18         reloj[1] = 0;
19         reloj[2] = 12;
20 }
21
22 /* Actualiza el reloj */
23 void reloj_actualizar(){
24         reloj[0]++;
25         if(reloj[0]==60){
26                 reloj[0] = 0;
27                 reloj[1]++;
28                 if(reloj[1]==60){
29                                 reloj[1] = 0;
30                                 reloj[2]++;
31                                 if(reloj[2]==24){
32                                                 reloj[2] = 0;
33                                         }
34                         }
35         }
36 }
37
38 /* Devuleve: Un string que es la hora a mostrar*/
39 uint8_t* reloj_get_string(){
40         uint8_t string[9];
41         // Horas
42         string[0] = (reloj[2] / 10)+'0';
43         string[1] = (reloj[2] % 10)+'0';
44         string[2] = ':';
45         //Minutos
46         string[3] = (reloj[1] / 10)+'0';
47         string[4] = (reloj[1] % 10)+'0';
48         string[5] = ':';
49         //Segundos
50         string[6] = (reloj[0] / 10)+'0';
51         string[7] = (reloj[0] % 10)+'0';
52         string[8] = '\0';
53         return string;
54 }
55
56 /* Recibe: dos caracteres que representan un numero
57  * Comprueba que este en el rango de los segundos
```

```
58  * Devuelve: Si se logro modificar con exito un 0 sino un -1
59  * */
60  uint8_t reloj_modificar_segundo(uint8_t a, uint8_t b){
61          uint8_t segundo = strToInt(a,b);
62          if(segundo>=0 && segundo<=24){
63                  reloj[2] = segundo;
64                  return 0;
65          }
66          return -1;
67  }
68
69  /* Recibe: dos caracteres que representan un numero
70   * Comprueba que este en el rango de los minutos
71   * Devuelve: Si se logro modificar con exito un 0 sino un -1
72   * */
73  uint8_t reloj_modificar_minuto(uint8_t a, uint8_t b){
74          uint8_t minuto = strToInt(a,b);
75          if(minuto>=0 && minuto<=60){
76                  reloj[1] = minuto;
77                  return 0;
78          }
79          return -1;
80  }
81
82  /* Recibe: dos caracteres que representan un numero
83   * Comprueba que este en el rango de las horas
84   * Devuelve: Si se logro modificar con exito un 0 sino un -1
85   * */
86  uint8_t reloj_modificar_hora(uint8_t a, uint8_t b){
87          uint8_t hora = strToInt(a,b);
88          if(hora>=0 && hora<=24){
89                  reloj[0] = hora;
90                  return 0;
91          }
92          return -1;
93  }
```

4.11. mef.h

```
1  #ifndef MEF_H_
2  #define MEF_H_
3
4  #include "io/keypad.h";
5  #include "type.h";
6  #include "io/buffer.h";
7  #include "io/terminal.h";
8  #include "lib/reloj.h";
9
10 void mef_inicializar(void);
11 void mef_actualizar(void);
```

```
12
13 #endif /* MEF_H_ */
```

4.12. mef.c

```
1  #include "lib/mef.h"
2
3  /* Defino los estados de la maquina de estados finitos
4   * S0: Estado en reposo en donde muestra el reloj.
5   * S1: Estado en el cual permite cambiar la hora.
6   * S2: Estado en el cual permite cambiar los minutos.
7   * S3: Estado en el cual permite cambiar los segundos.
8   * */
9  typedef enum{S0, S1, S2, S3} state;
10
11 /* Esta variable representa el estado actual de la MEF */
12 state estado;
13 struct temp {
14         uint8_t len;
15         uint8_t contenido[2];
16 };
17
18 /* Digitos siendo ingresados */
19 static struct temp digitos_temp;
20
21 /* Se utiliza estre flag para hacer titilar un digito */
22 extern volatile uint8_t flagTitileo;
23
24 /* Contador de tiempo maximo de ingreso de los digitos */
25 static int8_t contador_maximo_tiempo = 0;
26
27 /* Se inicializa la MEF */
28 void mef_inicializar(){
29         estado = S0;
30 }
31
32 /* Funcione de Test
33  * Paramtros: recibe que tipo de digitos se van a ingresar
34  * -> Para Horas debe ser 0,'A' para asi establecer la posici de la terminal a
        ingresar el digito y la tecla a presionar.
35  * -> Para Horas debe ser 3,'B' para asi establecer la posici de la terminal a
        ingresar el digito y la tecla a presionar.
36  * -> Para Horas debe ser 7,'C' para asi establecer la posici de la terminal a
        ingresar el digito y la tecla a presionar.
37  * Devuelve:
38  * -> 1 si ya se ingresaron los 2 digitos y se preciono la tecla para confirmar el
        ingreso.
39  * -> -1 si se desea cancelar la operaci
40  * -> 0 si se preciono cualquier otra tecla.
41  * */
```

```
42 static int8_t ingreso_digitos(uint8_t tiempo, uint8_t key) {
43
44         uint8_t caracter = buffer_pop();
45         /* Se le da al usuario 2 minutos como maximo para ingresar los caracteres
               */
46         if(contador_maximo_tiempo == 1200){
47                 return -1;
48         }
49         /* Parpadeo del digito a ingresar */
50         if(flagTitileo == 1){
51                 terminal_set_digit('␣',digitos_temp.len+tiempo);
52                 flagTitileo=0;
53         }
54         if('0' <= caracter && caracter <= '9' && (digitos_temp.len < 2)){
55                 digitos_temp.contenido[digitos_temp.len++] = caracter;
56                 /* Escribir en pantalla*/
57                 terminal_set_digit(caracter,digitos_temp.len+tiempo-1);
58         }else if((caracter == key)&&(digitos_temp.len==1)) {
59                 return 1;
60         }else if(caracter == 'D') {
61                 return -1;
62         }else{
63                 return 0;
64         }
65 }
66
67 /* Actualizaci de la maquina de estados finistos */
68 void mef_actualizar(){
69         uint8_t key = buffer_peek();
70         uint8_t retorno;
71         /*Se incrementa cada 100 ms*/
72         contador_maximo_tiempo++;
73         switch (estado) {
74                 /*Estado inicial*/
75                 case S0:
76                         digitos_temp.len = 0;
77                         if(key=='A'){
78                                 estado = S1;
79                         }
80                         if(key=='B'){
81                                 estado = S2;
82                         }
83                         if(key=='C'){
84                                 estado = S3;
85                         }
86                         break;
87                 /*Estado cambiar horas*/
88                 case S1:
89                         retorno = ingreso_digitos(0,'A');
90                         if( retorno == 1){
91                                 reloj_modificar_hora(digitos_temp.contenido[0],
                                        digitos_temp.contenido[1]);
```

```
92                                      estado = S0;
93                              }else if (retorno == -1){
94                                      estado = S0;
95                              }
96                              break;
97                      /*Estado cambiar minutos*/
98                      case S2:
99                              retorno = ingreso_digitos(3,'B');
100                             if( retorno == 1){
101                                     reloj_modificar_hora(digitos_temp.contenido[0],
                                            digitos_temp.contenido[1]);
102                                     estado = S0;
103                             }else if (retorno == -1){
104                                     estado = S0;
105                             }
106                             break;
107                     /*Estado cambiar segundos*/
108                     case S3:
109                             retorno = ingreso_digitos(7,'C');
110                             if( retorno == 1){
111                                     reloj_modificar_hora(digitos_temp.contenido[0],
                                            digitos_temp.contenido[1]);
112                                     estado = S0;
113                             }else if (retorno == -1){
114                                     estado = S0;
115                             }
116                     break;
117                     default:
118                             terminal_set_string(reloj_get_string());
119                     break;
120             }
121     }
```

Programa principal

4.13. type.h

```
1   #ifndef TYPEE_H_
2   #define TYPEE_H_
3
4   typedef signed char int8_t;
5   typedef unsigned char uint8_t;
6   typedef short int int16_t;
7   typedef unsigned short int uint16_t;
8   typedef long int int32_t;
9   typedef unsigned long int uint32_t;
10  typedef long long int int64_t;
11  typedef unsigned long long uint64_t;
12
13  #endif // TYPE H
```

4.14. main.c

```
1  #include <hidef.h> /* for EnableInterrupts macro */
2  #include "derivative.h" /* include peripheral declarations */
3
4  #ifdef __cplusplus
5  extern "C"
6  #endif
7  void MCU_init(void); /* Device initialization function declaration */
8
9
10 void main(void) {
11
12   /* Inicializaci de los modulos */
13   MCU_init();
14   keypad_init();
15   reloj_init();
16   mef_inicializar();
17   terminal_init("12:00:00");
18
19   for(;;);
20  }
```