

Circuitos Digitales y Microcontroladores

Tp2 - Ejercicio Entregable

Petrigh Lautaro - Canessa Lucio E.



UNIVERSIDAD
NACIONAL
DE LA PLATA

Índice

Introducción.....	3
Problema.....	3
Interpretación.....	4
Resolución del problema.....	5
Teclado.....	5
LCD.....	8
Conexión.....	9
Clock.....	11
Timer.....	11
Máquina de estado.....	14
IMPLEMENTACIÓN DE LA MEF.....	18
-Introducción.....	18
-IMPLEMENTACION EN PSEUDOCODIGO.....	20
Validación.....	27
Código.....	30

Introducción

En este nuevo proyecto, además de utilizar los conocimientos previos sobre el uso de MCU y sus herramientas, deberemos de utilizar y desarrollar nuevos métodos para poder cumplir con la resolución al problema, entre ellos se encuentra el uso de temporizadores y máquinas de estado finito.

Problema

Implementar con el MCU una cerradura electrónica como la mostrada 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 máquina de estados temporizadas con Timer. A continuación se enumeran los requerimientos que debe satisfacer el sistema:

- a) Cuando el equipo se inicia deberá mostrar en la primera línea del LCD un reloj funcionando con el formato HH:MM:SS (horas, minutos y segundos) inicializado en la hora de compilación y en la segunda línea el estado de la cerradura "CERRADO".
- b) El sistema debe tener la clave numérica por defecto 5913 de manera de poder activar o desactivar la cerradura. Si el usuario presiona la clave correcta se mostrará en la segunda línea "ABIERTO" durante 3 seg y luego volverá automáticamente al estado por defecto. Si la clave es incorrecta se mostrará el estado "DENEGADO" durante 2 seg y luego volverá automáticamente al estado por defecto. No se mostrarán en LCD las teclas presionadas, en su lugar se mostrarán caracteres '*'.
- c) Para modificar la hora del reloj, en el estado por defecto, se deberán introducir los campos HH, MM y SS por separado, siguiendo el siguiente procedimiento: Presionar 'A' para cambiar los dos dígitos de la hora. Para guardar los cambios y terminar, volver a presionar 'A'. De la misma manera se utilizarán las teclas 'B' y 'C' para modificar los minutos y los segundos respectivamente. Si desea cancelar se puede presionar '#' lo que

interrumpe el ingreso y vuelve al estado por defecto sin guardar los cambios. Mientras la modificación del reloj se esté ejecutando, el cursor del LCD deberá estar habilitado y parpadeando cada 1 seg hasta que el usuario finalice el ingreso de datos.

d) Deberá verificar los datos ingresados por el usuario ya que el campo hora debe ser entero entre 00 y 23 mientras que los campos minutos y segundos deben estar entre 00 y 59. En el caso de detectar inconsistencias debe simplemente ignorar los datos ingresados sin presentar mensajes de error.



Fig. 1 - Cerradura electrónica

Interpretación

Se busca implementar una cerradura electrónica utilizando un MCU Atmega328p, un display LCD de 2 líneas y un teclado matricial 4x4. El sistema incluye un reloj en la primera línea del LCD, un estado de cerradura en la segunda línea y la capacidad de activar/desactivar la cerradura con una clave numérica. También se permite la modificación

de la hora del reloj mediante la introducción de los campos HH, MM y SS (siendo HH los dos dígitos de la hora, MM los dos dígitos de los minutos y SS los dos dígitos de los segundos) por separado, siguiendo un procedimiento específico. Además, el sistema deberá verificar y procesar adecuadamente los datos ingresados por el usuario, mostrando o no en dicho casos mensajes de error en caso de inconsistencias.

Resolución del problema

Para llevar a cabo la resolución del problema, habrá que abordar varios aspectos. Ya sea la comprensión, desarrollo y configuración de las herramientas provistas por la cátedra; La implementación del sistema y sus correspondientes validaciones y conclusiones.

Empezaremos con la explicación del desarrollo de las herramientas.

Teclado

Para la implementación de la alarma tendremos a un teclado de 4x4 el cual será conectado al microcontrolador, para manipularlo utilizaremos la librería "teclado.h" la cual actúa como driver del teclado. El teclado matricial funciona como una matriz de botones que conecta cada fila de con cada columna de la siguiente manera:

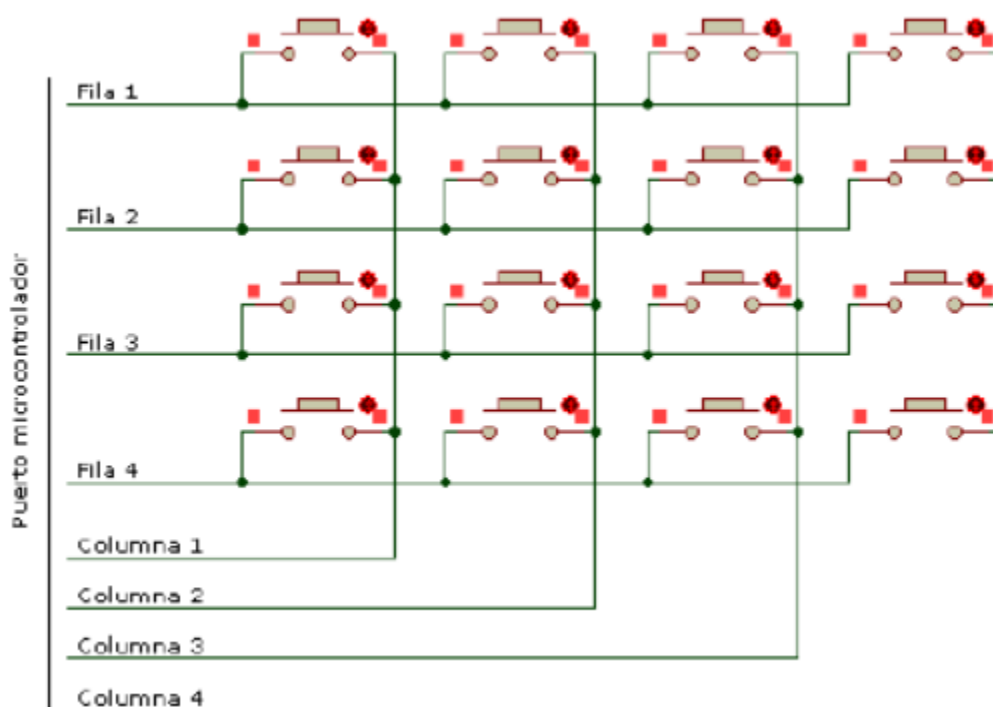


Fig. 2 - Esquema del teclado matricial

esta configuración nos permite tener 1 salida y 1 entrada por cada fila y columna respectivamente, lo cual nos deja con 8 variables (4 de salida y 4 de entrada) para 16 botones en total, para conocer qué botón es presionado, utilizaremos el método de barrido el cual implica en primer lugar, colocar las 4 salidas o filas en alto, e ir iterando una a una cambiandola a bajo, dentro de cada iteración, chequeamos qué columna está en bajo, y de esta forma conocemos fila y columna del botón presionado, información suficiente para saber qué botón es, en caso de no haberse presionado ningún boton retornaremos el valor 0xFF. Para el conexionado del teclado contamos hacemos del uso de los puertos B y D, utilizando los pines 4, 3 y 0 del puerto B como salida, además de el pin 7 del puerto D para la última fila, como entradas seteamos los pines 3, 5, 4 y 2 del puerto D. Esta conexión fue dada por la cátedra en conjunto con el conexionado del display, por lo que nuestro barrido lo partimos en 2 partes, un *for loop* que itera por cada fila del puerto B consultado por cada columna del puerto D y al final una última iteración de cada columna en el puerto D utilizando la fila conectada en el pin 7 del puerto D también. Este algoritmo puede ser consultado constantemente conocido como método de *polling*, el cual nos dirá qué botón

fue presionado. Además del barrido, nuestro driver cuenta con algoritmos de antirebote y detección múltiple, el cual permite tener una interacción más acorde a los requerimientos, que sin estos algoritmos obtendremos valores basura que aparecen por las características físicas de la implementación.

Haciendo la implementación en pseudocódigo, la implementación del barrido tendría la siguiente lógica expresada en la figura 3.

```
int Barrido( número de 8 bits)
    Configuro entradas con pull-ups internos
    Configuro Salidas en alto
    Para cada fila (pines 4, 3 y 0 puerto B)
        Setteo la fila correspondiente en bajo
        Para cada columna (pines 3, 4, 5 y 2 puerto D)
            Si la entrada es 0
                retorno valor tecla (fila*4 + columna)
            Setteo la fila correspondiente en alto
        Setteo la pin 7 puerto D en bajo
    Itero por cada columna si la entrada es 0
        retorno valor tecla (12*columna)
    Setteo la pin 7 puerto D en alto
    retorno 0xFF en caso de no retornar en ningún otro caso
```

Figura 3 - Implementación del barrido

En nuestro archivo timer.c tenemos un arreglo estático que contiene una matriz de 16 caracteres, cada uno siendo el que se encuentra en el teclado, la posición de cada carácter

está en función de las filas y columnas, permitiendo obtener el carácter si al conocemos la fila y la columna, y obteniendo su posición mediante la $(\text{Nro fila} * 4 + \text{Nro columna})$ esto es porque las filas y columnas las iteramos con 2 variables que van de 0 a 3, estas variables a sí mismo las utilizamos como índice para recorrer 2 vectores más, cada uno conteniendo el valor en hexadecimal que nos permite colocar en alto o bajo el bit correspondiente para el caso de las salidas, o nos permite leer el bit correspondiente para el caso de la entrada, ambos vectores también se inicializan estáticos con sus correspondientes valores junto con el vector de caracteres. Además la librería cuenta con una función que nos permite garantizar el antirebote o evitar detecciones múltiples llamada `KEYPAD_Scan`, esta función recibe como parámetro un puntero, el cual apunta a una variable de 8 bits, esta variable contiene el valor del carácter presionado para poder ser almacenado y validado tanto para la contraseña como con el cambio de la hora.

LCD

Las librerías y conexiones de este periférico fueron otorgados por la cátedra, pero aun así haremos algunos comentarios al respecto.

El display cuenta con 2 renglones de 16 caracteres cada 1, el cual está conectado a los puertos C (Pin 1 y 2) y B (Pin 1 y 2) del microcontrolador para el bus de datos del display y el puerto D (Pin 0 y 1) del microcontrolador está conectado a los pines de Enable y el pin de selección Datos/Comandos del display.

El display cuenta con un cursor que se va moviendo a medida que uno envía caracteres utilizando la función `LCDsendChar(c)`, o strings utilizando la función `LCDstring(s,t)` o sino podemos posicionar deliberadamente nosotros con llamando a la función `LCDGotoXY(x,y)` definidas en la librería `lcd.h`

Decisiones tomadas a la hora de manejar el display fue el de evitar el uso de algunas funciones ya que la librería se maneja con retardo, como algunos de nuestros métodos hace más de un llamado al lcd cuando lo utilizamos, como por ejemplo borrar el display y de ahí escribir, evitamos el uso de la función `LCDclr()` ya que esta tiene un retardo mayor y ocasionalmente no funcionaba como era deseado. Por esta razón recurrimos a escribir 16 caracteres " " sobre el renglón que se desea borrar.

Conexión

Una vez implementado y explicado el funcionamiento del teclado y del led, procederemos a hacer el conexionado de los elementos al MCU.

Como hemos explicado anteriormente en el teclado, utilizaremos los puertos B y D, en los cuales se conectarán los puertos B4, B3, B0 y D7 para representar las filas del teclado. Y se conectarán los puertos D3, D5, D4 y D2 para las columnas del mismo.

Para el caso del LCD, usaremos el mismo conexionado que hemos visto de la guía de trabajos prácticos N°2.

En la figura 4 se podrá ver la conexión de todos los elementos utilizados en este proyecto.

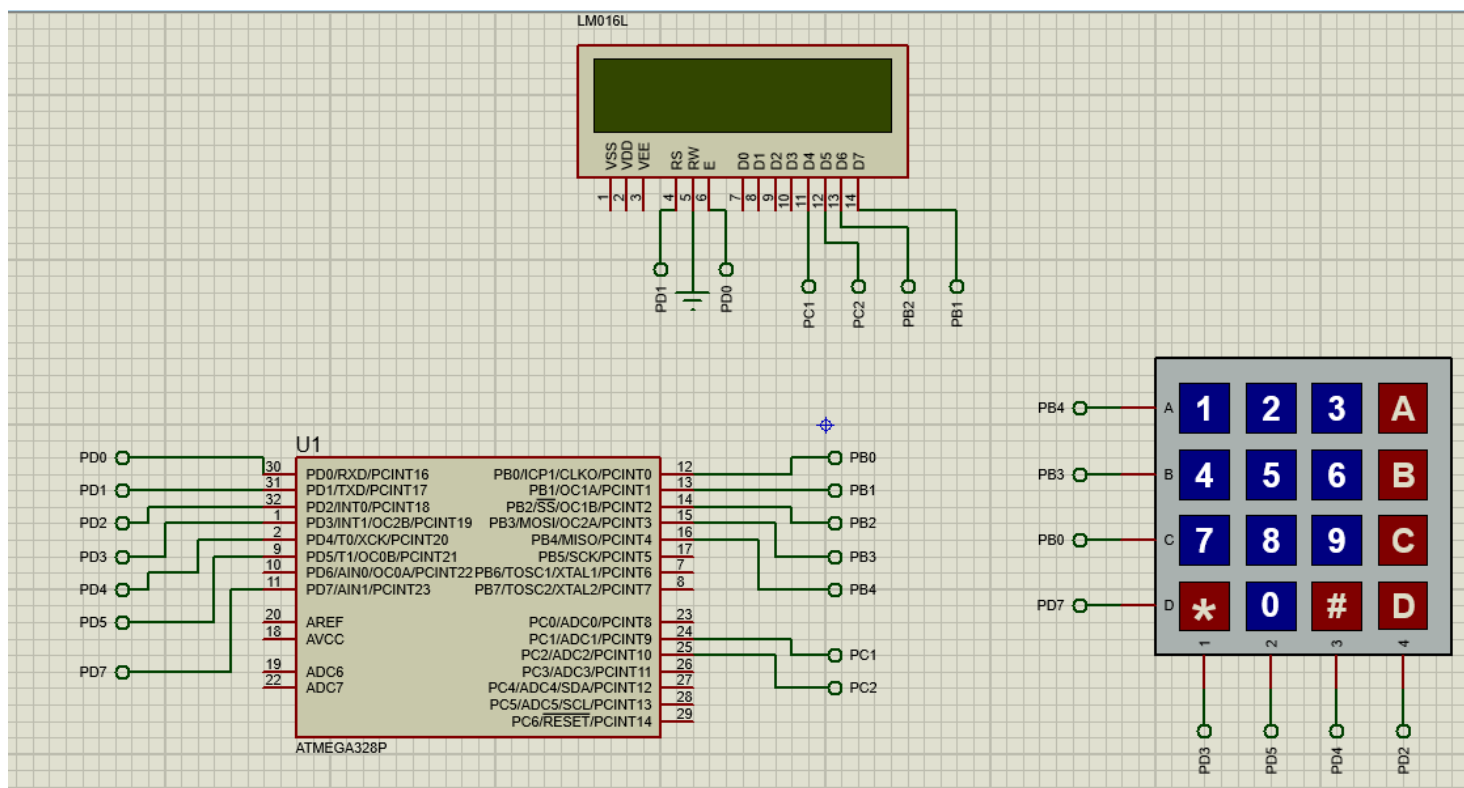


Figura 4 - Conexión del teclado y el LCD al MCU

Clock

Para la implementación del reloj de la alarma, optamos por escribir una librería que englobe su comportamiento, para ello definimos las funciones:

- `CLOCK_Init(h, m, s)`: Inicializa el reloj con los valores pasados como parámetros, éste será llamado con la hora de compilación.
- `CLOCK_Update()`: Avanza 1 segundo el reloj
- `CLOCK_GetHora(uint8_t*)`: devuelve la hora en un arreglo de caracteres [HH:MM:SS]
- `CLOCK_ModHora(horas)`: Modifica la hora en caso de que sea válida [0-23]
- `CLOCK_ModMin(minutos)`: Modifica los minutos en caso de ser válidos [0-59]
- `CLOCK_ModSeg(segundos)`: Modifica los segundos en caso de ser válidos [0-59]

Además de las funciones, la librería cuenta con 3 variables internas estáticas que corresponden al almacenamiento de las horas, minutos y segundos.

Para que el reloj se actualiza cada 1 un segundo utilizamos una arquitectura *time-triggered* la cual cuenta con una interrupción periódica que llamará a la función `CLOCK_Update` cada una cantidad calculada de tiempo (1 segundo). Esta arquitectura será más desarrollada en el apartado siguiente.

Timer

Nuestra implementación maneja varios “procesos” y cada proceso cuenta con distintas funciones, para organizar estos eventos utilizamos una arquitectura *Time-Triggered* la cual nos permite coordinarlos en base a interrupciones periódicas, estas interrupciones pueden

hacer que actualicen el reloj cada 1 segundo, independientemente de lo que el usuario esté haciendo, facilitando la implementación a la hora de manejar varias máquinas de estado ya que una de ellas es en función del tiempo.

Para implementar todo lo mencionado previamente, hacemos uso del temporizador de 8 bits TIMER 0 integrado en el ATmega328p. Este fue configurado en modo CTC, en el cual el temporizador cuenta desde 0 hasta un valor de comparación predefinido y luego se reinicia a 0. Es decir, cuando el temporizador alcanza el valor de comparación, se produce una coincidencia y se ejecuta una acción específica, como una interrupción o un cambio en la salida del temporizador. Podemos conocer el tiempo entre cada interrupción mediante una simple función

$$[(\text{Valor de comparación} + 1) * \text{Preescalar} / \text{Frecuencia del microprocesador}]$$

- La frecuencia del microprocesador la definimos a 16MHz.
- El valor de comparación es asignado a la variable OCR0A la cual establecimos en 249.
- El preescalar se define en el registro TCCR0B el cual seteamos CS01 y CS00 en 1 mientras que CS02 en 0, indicándole al temporizador que utilizamos el preescalar de 64.
- El modo CTC se debe almacenar en el registro TCCR0A seteando en 1 el bit 6 del registro.
- Finalmente configuramos las interrupciones habilitando la interrupción COMPA

Una vez inicializado el TIMER 0, tendremos interrupciones cada

$$[(249 + 1) * 64 / 16\text{MHz}] = 1\text{ms}$$

Las cuales hacemos uso de ellas en la función Manejador de interrupción del comparador del TIMER 0, en ella actualizaremos la máquina de estados de la alarma cada 10 ms, y actualizaremos el reloj cada 1 segundo (1000 ms), a la hora de implementar la llamada a actualizar el reloj, en lugar de hacerlo en la iteración 100 correspondiente a 1000ms, llamamos cada 89 iteraciones, si bien el cálculo nos dice

que esto actualizará el reloj cada 890 ms, en la realidad termina siendo actualizado en un tiempo un poco mayor a 1000 ms, esto se da porque los tiempos de retardo que se dan durante la ejecución del programa

Máquina de estado

MODELADO DEL SISTEMA

Para poder plantear el sistema dado, debemos de usar una máquina de estado. El uso de este nos permitirá:

- **Gestión de eventos:** La cerradura electrónica debe responder a eventos como la introducción de la clave, la modificación de la hora y el cancelamiento de la operación. Una máquina de estados permite gestionar estos eventos de manera ordenada y controlada, definiendo cómo el sistema responde a cada uno de ellos en función del estado en el que se encuentre.
- **Comportamiento secuencial:** El funcionamiento de una cerradura electrónica implica una secuencia de pasos y acciones bien definidos. Una máquina de estados proporciona una estructura lógica que permite representar y controlar esta secuencia, asegurando que los pasos se realicen en el orden correcto y según las reglas establecidas.
- **Cambios de estado:** La cerradura electrónica debe mantener un estado determinado, como "CERRADO", "ABIERTO" o "DENEGADO", y cambiar de un estado a otro en función de las acciones del usuario y los eventos que ocurran. La máquina de estados facilita el control y la gestión de estos cambios de estado, asegurando que el sistema responda correctamente y se mantenga en el estado adecuado en todo momento.
- **Modularidad y mantenibilidad:** El uso de una máquina de estados permite dividir la lógica del programa en diferentes estados y transiciones, lo que mejora la modularidad y facilita el mantenimiento del código. Cada estado y transición puede

ser diseñado, probado y modificado de forma independiente, lo que facilita la incorporación de nuevas funcionalidades o la corrección de errores sin afectar al funcionamiento global del sistema.

Ahondando más en la implementación de la misma, sabemos que:

1. El sistema tendrá un estado default que será "CERRADO", en la cual indicará la hora (ya sea la hora de compilación o una ingresada por el usuario) acompañado de un mensaje que indicará el nombre del mismo estado.
2. Si el usuario llegara a apretar un número en el teclado, se pasaría al estado "PASSWORD", la cual no solo se ingresa la clave, sino que además la validará y procederá a ir a uno de los próximos dos estados: "ABIERTO" o "DENEGADO".
3. Una vez validada la clave, si llegara a ser la clave correcta, pasará al estado de "ABIERTO". En la cual se indicará un mensaje con el texto "ABIERTO" por 3 segundos, y luego procederá a ir al estado por default.
4. Si la clave fue validada y es incorrecta, pasará al estado de "DENEGADO". En el cual se indicará el mismo mensaje que su nombre de estado por 2 segundos y luego procederá a ir al estado default.
5. Si estando en estado default, se llegara a presionar los botones "A", "B" o "C". Entonces pasaría al estado de "HORA", "MINUTO" o "SEGUNDOS", lo cual se podría modificar los dígitos de la hora (Si fuera "A"), los dígitos de los minutos (si fuera "B") o los dígitos de los segundos (si fuera "C"). Una vez hecho el cambio, al apretar el mismo botón por el cual se redirecciono al estado actual, procederá a verificar el horario por él efectuado y, si no hubiera ningún error, se redirigirá al estado por default con el horario provisto por el usuario.

Cabe aclarar algunas cuestiones:

- Cuando se presione alguna tecla numérica para ingresar la contraseña, la misma se mostrará como el carácter "*".

- Una vez que se ingresa el primer carácter, el usuario tendrá a su disposición 30 segundos para ingresar los otros 3 caracteres. En dicho caso que haya pasado ese tiempo, el sistema se redirigirá al estado "DENEGADO".

Además, no podrá apretar ninguna otra tecla que no sea numérica hasta que termine de ingresar toda la clave.

- Al ingresar los dígitos del horario a cambiar (ya sea hora, minutos o segundos), el usuario puede cancelar el cambio y volver al estado por defecto apretando el carácter "#".
- Si el usuario ingresa una hora cuya combinación de los dígitos sea mayor que el valor 23, o si ingresara minutos o segundos los cuales sus valores sean mayor a 59, al apretar su correspondiente tecla para aceptar los cambios el sistema ignorará los valores ingresados y seguirá usando el horario establecido previamente al intento de modificarlo.
- Además, se desea que al cambiar el horario, el cursor parpadee cada 1 segundo en el dígito a cambiar hasta que se efectúe un cambio de estado.

Una vez aclarado sobre cómo tendría que ser el sistema y sus cuestiones generales, procederemos a hacer el modelado del mismo, como podemos ver en la figura 5.

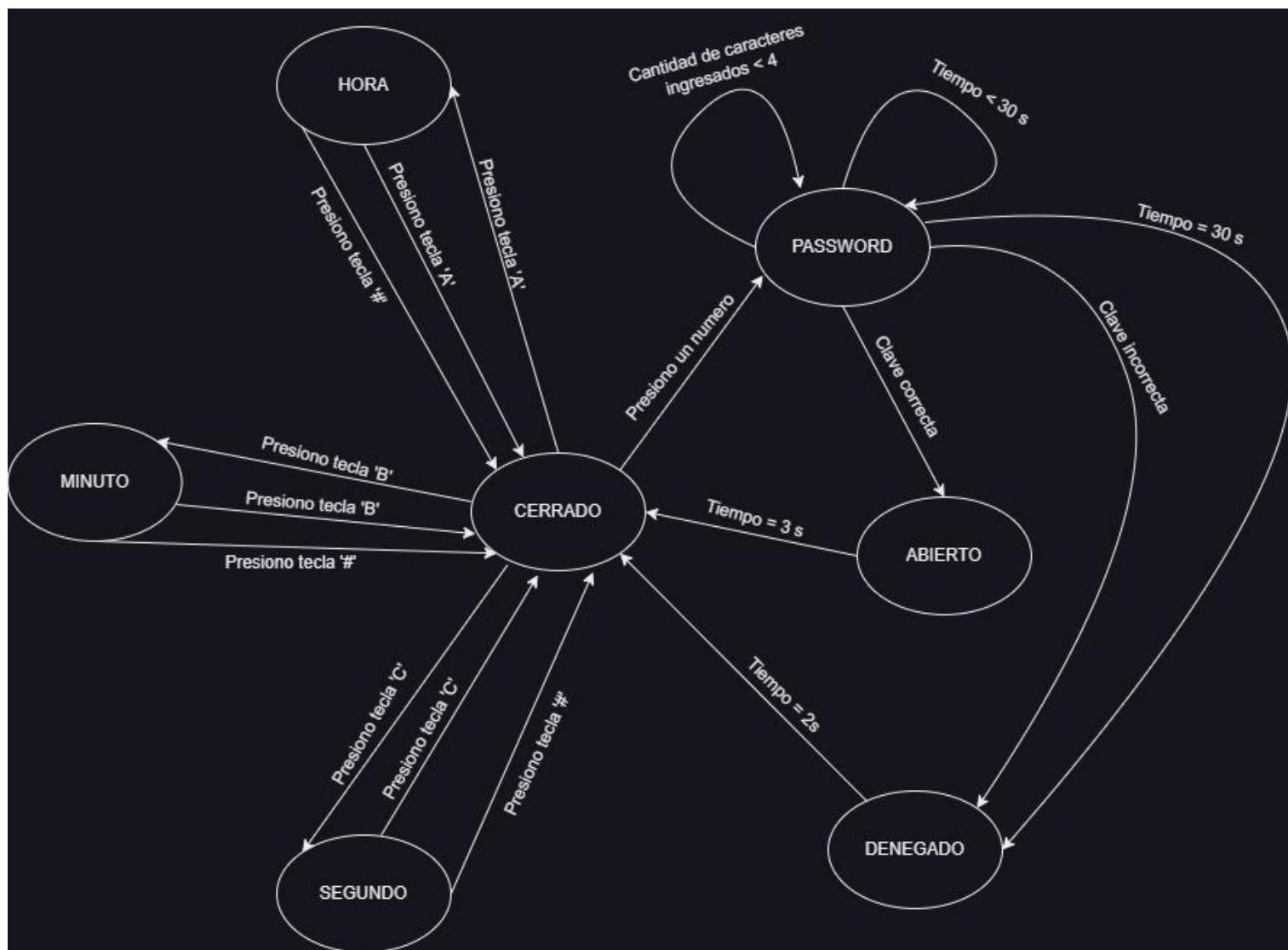


Figura 5 - Diagrama de estado de la MEF

IMPLEMENTACIÓN DE LA MEF

-Introducción

Para implementar la MEF que controlará todo el sistema vamos a desarrollar una librería. Esta misma se comunicará con otras para recibir sus entradas y enviar sus salidas.

Se usarán las cuatro librerías vistas en las secciones anteriores: Para determinar si el usuario ha presionado una tecla (entrada de la MEF), para actualizar el display LCD (salida de la MEF), para las operaciones con el horario y para las interrupciones.

Para sus variables internas hemos declarado los posibles estados (lo cual se utilizó un enumerador, ya que es un tipo de dato que puede tomar valores finitos que solo son modificables a la hora de su creación) llamado **MEF_STATE**, un puntero que poseerá el valor de la entrada por teclado (**key**), la cantidad de iteraciones que tendrá la MEF a cada llamado de actualización de la misma y que se reinicia al cambiar de estado (**State_count_time**), el propio estado actual del sistema (**System_state**) y un arreglo de 4 posiciones indicandonos la contraseña que se ingresará (**password[4]**).

Nuestra librería tendrá 2 métodos públicos: Para inicializar la MEF llamado *Cerradura_Init()* y para actualizar la MEF llamado *Cerradura_Update()*.

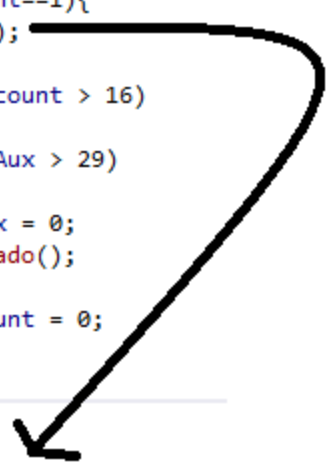
Para el caso de *Cerradura_Init()*, lo que hace es iniciar la propia MEF en el estado por defecto (es decir, en el estado "CERRADO") y reiniciar la cantidad de llamadas de actualización de la MEF.

En lo que respecta al segundo método (*Cerradura_Update()*), este será el encargado de leer las entradas y transicionar entre estados en caso de ser necesario.

Lo primero que se hace en este método es incrementar el valor del `state_call_count`, que representa cuantos ticks del sistema estoy dentro de un mismo estado. Dicho valor es utilizado para la optimización de las salidas: gran cantidad de los "outputs" de la máquina

son estáticos, es decir, para algunos estados, la salida (el contenido del display) se mantiene igual. Entonces, para no sobrecargar el LCD mandando a escribir siempre lo mismo en cada update, escribimos las salidas estáticas una sola vez, el primer tick que estoy en el estado.

Por ejemplo, en el estado abierto, que simplemente debe escribir en el LCD "ABIERTO", se imprimiría en el LCD 30 veces la misma frase en caso de no estar nuestra optimización, teniendo en cuenta que la máquina se actualiza cada 10 ms y la salida debe permanecer por 3 seg antes de volver al estado por defecto. En la figura 6 puede verse la implementación de esta parte.



```

case ABIERTO:
    if(State_call_count==1){
        stateAbierto();
    }
    if (++State_call_count > 16)
    {
        if (++counterAux > 29)
        {
            counterAux = 0;
            stateCerrado();
        }
        State_call_count = 0;
    }
    break;

static void stateAbierto(void){
    LCDhome();
    LCDstring((uint8_t*)"          ",16);
    LCDGotoXY(4,1);
    LCDcursorOFF();
    LCDstring(abierto,8);
}

```

Figura 6 -Optimización en la escritura del LCD en el estado ABIERTO

Por otro lado, para el estado por defecto, si bien no hace falta volver a escribir la palabra en cada update, se debe actualizar la hora cada segundo (la salida es la hora, pero la hora

es dinámica). Nuevamente, durante 10 ejecuciones de la máquina, la hora se mantiene constante. Para optimizar las escrituras en el LCD, la máquina tiene un contador que si se permanece en el estado por defecto por 100 ms, actualiza la hora. En la figura 7 se puede ver lo antes descrito.

```
static void BreakCerrado(void)
{
    if(State_call_count==1){
        LCDstring((uint8_t*)"
        LCDGotoXY(4,0);
        CLOCK_GetHora(hora);
        LCDstring(hora, 8);
        LCDGotoXY(4,1);
        LCDstring(cerrado, 8);
        LCDhome();
        LCDGotoXY(3,0);
    }
    if(horaActual >= 10){
        CLOCK_GetHora(hora);
        LCDGotoXY(3,0);
        LCDstring(hora, 8);
        horaActual = 0;
        LCDhome();
        LCDGotoXY(3,0);
    }
}
```

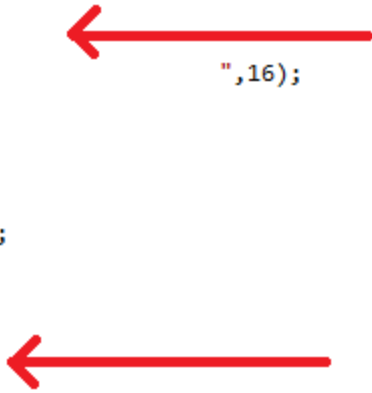


Figura 7 - Optimización en la escritura del LCD en el estado CERRADO

-IMPLEMENTACION EN PSEUDOCODIGO

La implementación de la máquina de estados (MEF) en software, es del tipo SWITCH-CASE, donde el método Cerradura_Update() consulta en qué estado está el sistema actualmente. Una vez que encuentra el estado actual, procesa las posibles entradas. Si esta entrada genera un cambio en el estado que amerite una transición a otro, se realiza la misma, por lo que en la siguiente ejecución del Cerradura_Update() el estado habrá cambiado. En la figura 8 se puede ver un pseudocódigo de la función.

```
Cerradura_Update()
begin
  Pregunto en qué estado estoy
  Si estoy en CERRADO:
    //Hago lo correspondiente al estado CERRADO
  Si estoy en PASSWORD:
    //Hago lo correspondiente al estado PASSWORD
  Si estoy en ABIERTO:
    //Hago lo correspondiente al estado ABIERTO
  Si estoy en DENEGADO:
    //Hago lo correspondiente al estado DENEGADO
  Si estoy en HORA:
    //Hago lo correspondiente al estado HORA
  Si estoy en MINUTO:
    //Hago lo correspondiente al estado MINUTO
  Si estoy en SEGUNDOS:
    //Hago lo correspondiente al estado SEGUNDOS
end
```

Figura 8 - Pseudocódigo de la Cerradura_Update()

En cualquiera de todos los estados hay ciertas “tareas” a realizar. Todos los estados procesan las entradas (el teclado y el tiempo pasado) y en caso de ser necesario, marcan la transición a nuevos estados. Además, los estados definen las salidas, por lo que en su implementación deben actualizarse las mismas.

En nuestra implementación, hemos modelado los estados principales de la figura 5. Esto puede llevar a confusiones ya que, en algunos casos, si bien el estado en sí no cambia, el contenido del display sí. Por ejemplo, en el estado CERRADO, la salida es la hora y el mensaje de “CERRADO”, pero la hora cambia cada segundo. Entonces, internamente, cada 1 segundo se actualiza la hora, pero a un nivel esquemático se sigue mostrando “la hora” con el mensaje. A esto nos referimos previamente con lo de salidas “estáticas” y “dinámicas”. La hora, es una salida dinámica ya que se modifica pese a que seguimos en el mismo estado general.

En función de lo explicado anteriormente, a continuación, se puede observar un pseudocódigo para la implementación de cada estado (figuras 9 hasta 15)

```

Si estoy en el estado CERRADO
  Si se presiono una tecla
    Si presiono una tecla numérica
      Cambio de estado a PASSWORD
    Si presiono la tecla 'A'
      Cambio de estado a HORA
    Si presiono la tecla 'B'
      Cambio de estado a MINUTO
    Si presiono la tecla 'C'
      Cambio de estado a SEGUNDOS
  Sino imprimo la hora y "CERRADO"

```

Figura 9 - Pseudocódigo del estado CERRADO

```

Si estoy en el estado PASSWORD
  Si ingreso una tecla válida y todavía no ingresé 4 dígitos
    Guardo el carácter e imprimo en el LCD un '*'
  Si ya ingrese los 4 dígitos
    Compruebo si la clave es correcta
      Si es correcta voy al estado ABIERTO
      Sino voy al estado DENEGADO
  Si pasa más de 15 segundos en este estado
    Cambio a estado DENEGADO

```

Figura 10 - Pseudocódigo del estado PASSWORD

```

Si estoy en el estado HORA
  Posiciono el cursor sobre la decena de la hora para ser modificada
  Activo el parpadeo de 1 segundo sobre el cursor
  Si se ingresó una tecla numérica
    Si es el primer dígito
      Sumo uno a la cantidad de dígitos actual
      Se almacena el valor en la posición 0 del vector "hora"
    Sino (es el segundo dígito)
      Se almacena el valor en la posición 1 del vector "hora"
  Sino, si la tecla presionada es igual a 'A'

```

Verifico y actualizo la hora en caso de ser correcta
 Apago el cursor
 Cambio a estado CERRADO
 Sino, si la tecla presionada es igual a '#'
 Apago el cursor
 Cancelo los cambios yendo directamente al estado CERRADO

Figura 11 - Pseudocódigo del estado HORA

Si estoy en el estado MINUTO
 Posiciono el cursor sobre la decena del minuto para ser modificada
 Activo el parpadeo de 1 segundo sobre el cursor
 Si se ingresó una tecla numérica
 Si es el primer dígito
 Sumo uno a la cantidad de dígitos actual
 Se almacena el valor en la posición 3 del vector "hora"
 Sino (es el segundo dígito)
 Se almacena el valor en la posición 4 del vector "hora"
 Sino, si la tecla presionada es igual a 'B'
 Verifico y actualizo la hora en caso de ser correcta
 Apago el cursor
 Cambio a estado CERRADO
 Sino, si la tecla presionada es igual a '#'
 Apago el cursor
 Cancelo los cambios yendo directamente al estado CERRADO

Figura 12 - Pseudocódigo del estado MINUTO

Si estoy en el estado SEGUNDOS
 Posiciono el cursor sobre la decena del segundo para ser modificada
 Activo el parpadeo de 1 segundo sobre el cursor
 Si se ingresó una tecla numérica
 Si es el primer dígito
 Sumo uno a la cantidad de dígitos actual
 Se almacena el valor en la posición 6 del vector "hora"
 Sino (es el segundo dígito)
 Se almacena el valor en la posición 7 del vector "hora"

```
Sino, si la tecla presionada es igual a 'C'  
  Verifico y actualizo la hora en caso de ser correcta  
  Apago el cursor  
  Cambio a estado CERRADO  
Sino, si la tecla presionada es igual a '#'  
  Apago el cursor  
  Canelo los cambios yendo directamente al estado CERRADO
```

Figura 13 - Pseudocódigo del estado SEGUNDOS

```
Si estoy en el estado ABIERTO  
  Si es la primera vez que estoy en el estado  
    Imprimo en el LCD "ABIERTO"  
Sino, si ya pasaron los 3 segundos  
  Cambio al estado CERRADO
```

Figura 14 - Pseudocódigo del estado ABIERTO

```
Si estoy en el estado DENEGADO  
  Si es la primera vez que estoy en el estado  
    Imprimo en el LCD "DENEGADO"  
Sino, si ya pasaron los 2 segundos  
  Cambio al estado CERRADO
```

Figura 15 - Pseudocódigo del estado DENEGADO

Todos los cambios de estados y salidas anteriormente descritos se encuentran implementados en el MEF.c como funciones privadas, el objetivo de esto es obtener encapsulación y ocultamiento de datos en la implementación del código. Es decir, desde afuera del archivo MEF.c solamente se conocen los métodos Cerradura_Init() y Cerradura_Update(), pero dentro de ella si aparecen todas las funciones para simular las salidas y los cambios de estados.

Algo similar ocurre con las variables dentro del archivo MEF.c: todas son privadas. Esto permite compartir las variables entre las funciones del archivo, pero que sean inaccesibles para afuera.

Tanto para declarar una función o una variable privada dentro de un archivo .c estas deben declararse como 'static' y el prototipo de función estar presente en ese mismo archivo. Para hacerlas públicas, se declaran sin el static y el prototipo de función debe estar presente en el .h como ocurre con las ya mencionadas Cerradura_Init() y Cerradura_Update().

Para poder ver toda la implementación en código por favor diríjase en la sección de "código" en este mismo informe.

-Inicialización del sistema

Lo último que queda por explicar es el archivo desde dónde se comienza la ejecución del programa: el main.c

En él, se incluyen todas las librerías que hemos estado mencionando, además de las que provee ya el mismo lenguaje (ver figura 16) y, luego de ejecutarse el Init de cada una de ellas y de habilitar las interrupciones globales (paso imprescindible para que el timer genere sus interrupciones periódicas y el sistema funcione), se quedará en un loop infinito llamando al método de la librería MEF para ir actualizando el estado del mismo (ver figura 18).

Además, por única vez, habrá una función que establecerá el valor del horario en base a la hora de compilación (ver figura 17).

```

#include "main.h"
#include "mef.h"
#include "clock.h"
#include "lcd.h"
#include "timer.h"
#include "stdio.h"
#include "stdlib.h"
#include "stdint.h"

```

Figura 16 - Librerías usadas en el main

```

uint8_t horaCompi, minutoCompi, segundoCompi;

void obtener_hora_actual() {
    const char* compilationTime = __TIME__; // Store the compilation time in a variable

    horaCompi = ((atoi(&compilationTime[0]))*10 + (atoi(&compilationTime[1]))) / 10;
    minutoCompi = ((atoi(&compilationTime[3]))*10 + (atoi(&compilationTime[4]))) / 10;
    segundoCompi = ((atoi(&compilationTime[6]))*10 + (atoi(&compilationTime[7]))) / 10;
}

```

Figura 17 - Función y variables que establecen la hora en base a la hora de compilación

```

int main(void)
{
    // Inicializar LCD
    LCD_Init();
    // inicializar clock
    obtener_hora_actual();
    CLOCK_Init(horaCompi,minutoCompi,segundoCompi);
    // inicializar mag de estados y buffers
    CERRADURA_Init();
    // configurar timer
    Timer0Init();
    // activar interrupciones
    sei();
    while(1)
    {
    }
    return 0;
}

```

Figura 18 - Código del método main

Validación

Para validar el correcto funcionamiento de nuestra cerradura electrónica, procederemos a mostrar los resultados de algunas pruebas las cuales demuestran la satisfactoria ejecución del trabajo.

1. Verificación del reloj: En esta sección nos aseguraremos de que el reloj se muestre correctamente en la primera línea del LCD en formato HH:MM:SS, como así también que la hora de inicialización es la de compilación. Finalmente adjuntamos un video mostrando el correcto avance del reloj actualizando en tiempo real:

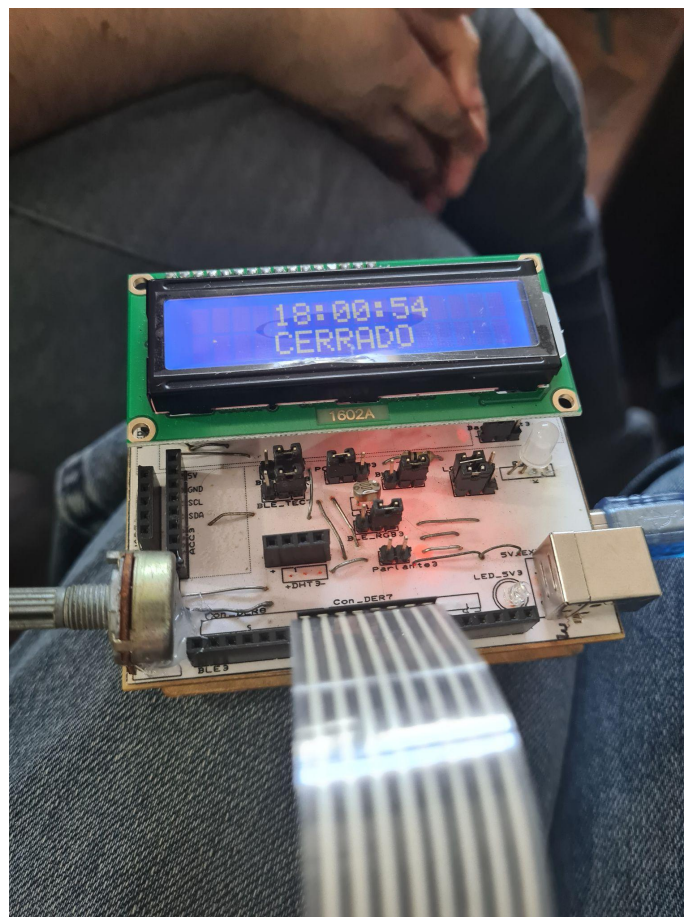
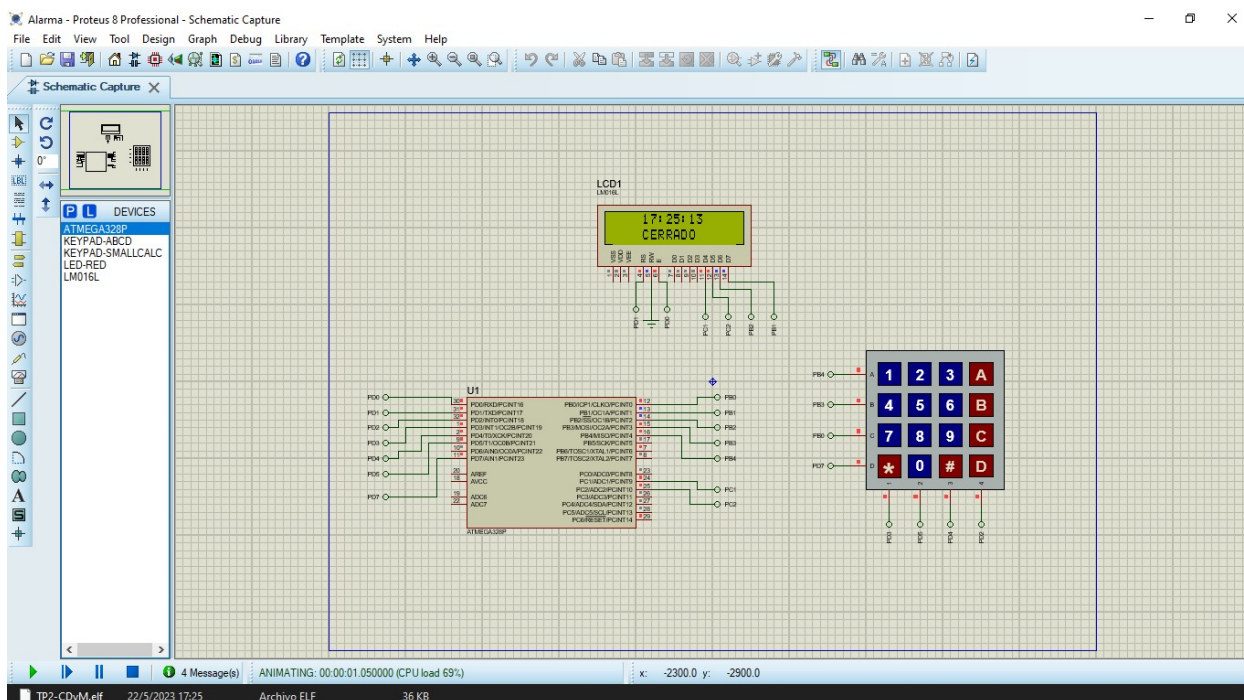


Figura 19 - Implementación real con tiempo correcto

Para la demostración del tiempo de compilación utilizaremos el simulador proteus ya que este nos permite ver el tiempo transcurrido de ejecución



caso de que el la hora no sea válida, volverá a la hora previa al cambio de hora. Mismo procedimiento para minutos y segundos utilizando las teclas B y C respectivamente

4. Verificación de datos ingresados por el usuario: Demostraremos que al ingresar la contraseña se mostrará asteriscos en el display y al ingresar 4 dígitos se hará la comprobación, Abierto en caso de que la contraseña ingresada sea 5913 o Denegado en caso contrario. Estos se ve en *video de validacion 1*, además será adjunto *video de validacion 2*, el cual muestra el caso de ingresar una contraseña parcial, y luego de pasados unos segundos se mostrará un cartel de Denegado. Si bien este requerimiento no es parte del trabajo práctico, fue agregado por una cuestión de funcionalidad del sistema.

Los dos videos se encontraran en el siguiente link (por problemas del peso de los archivos no lo hemos podido adjuntar): [InfinioKneza/TP2-CDyM: Entrega del tp 2 de Circuitos Digitales y Microncontroladores - Implementar con el MCU una cerradura electronica \(github.com\)](https://github.com/InfinioKneza/TP2-CDyM)

5. Validación del TIMER: En esta sección mostraremos el tiempo entre interrupciones para demostrar que los cálculos fueron correctos y que los tiempos de espera de los mensajes o el reloj son correctos

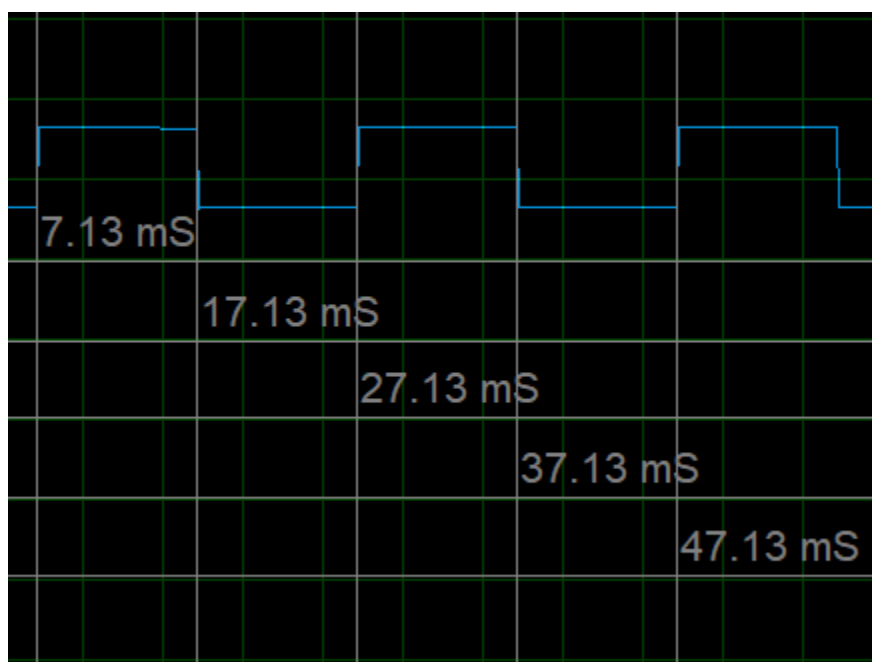


Figura 21 - Intervalo de actualización de la mef

En la figura 21 podemos ver el tiempo entre cada llamado a la mef para que se actualice, el error no se llega a registrar en el corto tiempo de ejecución sumado a la resolución del simulador. Pero podemos afirmar con certeza que la configuración del timer fue la correcta

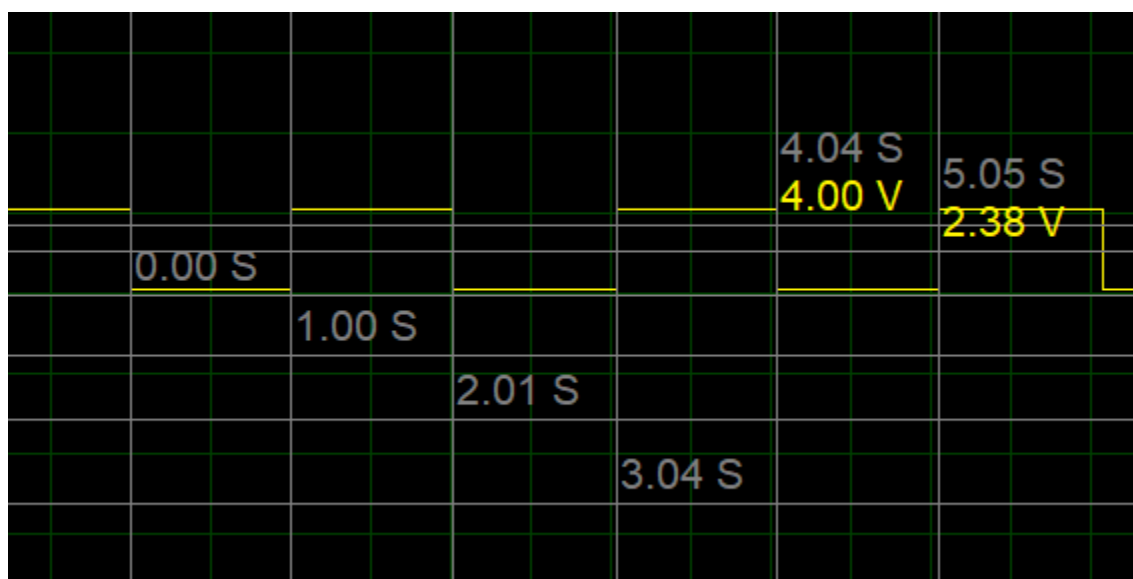


Figura 21 - Intervalo de actualización del reloj

En la figura 21 podemos ver el tiempo para el cual el reloj aumenta 1 segundo, si bien tenemos un error de hasta 30 ms, notamos que el timer funciona correctamente con el update de la máquina de estados, por lo que el error que se da en el reloj son los retardos que aparecen al ejecutar otras instrucciones, además de estar trabajando con variables de 8 bits, lo cual no nos permiten ajustar demasiado estos valores. Aun así, tomamos como satisfactorio este error para la resolución de este trabajo práctico.

Código

Librería Timer

```
#ifndef _TIMER_H
#define _TIMER_H

#include "mef.h"
#include "clock.h"
#include "avr/interrupt.h"

void Timer0Init(void);

#endif

/*-----*
--- END OF FILE ---
*/
```

```

/*
 * TP2_Timer0.c
 *
 * Created: 15/04/2022 11:58:35
 * Author : J.Juarez
 */

#include "timer.h"

static uint8_t cont_MEF = 0;
static uint8_t clock_MEF = 0;
//CONFIGURACION DEL TIMER0
//Activación de interrupción periódica y
//Generación de señal periódica en terminal OC0A (PD6)
void Timer0Init(void){
    DDRC |= (1<<DDC4) | (1<<DDC5);
    //configuración del TOPE del contador TCNT0
    OCR0A=249;           //249+1
    TCCR0A =(1<<COM0A0) | (1<<WGM01); //modo CTC, Toggle on compare match
    TCCR0B =(1<<CS01) | (1<<CS00); //CTC CLK/64 =16MHz/64 =250KHz
    TIMSK0 =(1<<OCIE0A); // habilitamos interrupción COMPA
}

//MANEJADOR DE INTERRUPTON DEL COMPARADOR A DEL TIMER0
//se activa periodicamente cuando TCNT0==OCR0A modo CTC
ISR(TIMER0_COMPA_vect){ //interrupción periódica de periodo Tisr=250/250KHz=1ms o fisr=250KHz/250=1KHz
    if(++cont_MEF == 10 ){
        if(++clock_MEF==89){ //Actualiza en la iteracion 890 por retardos del programa
            CLOCK_Update();
            clock_MEF=0;
            PORTC ^= (1<<PORTC4);
        }
        cont_MEF=0;
        CERRADURA_Update();
        PORTC ^= (1<<PORTC5);
    }
}

```

Librerías Clock (El reloj del horario, no confundir con el reloj del MCU)

```

#ifndef CLOCK_H_
#define CLOCK_H_

#define F_CPU 16000000UL //Frecuencia de reloj
#include <avr/io.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdint-gcc.h>

void CLOCK_Init(uint8_t h, uint8_t m, uint8_t s); //Inicia el reloj
void CLOCK_Update(); //Aumenta 1 segundo, en caso de overflow, aumenta minutos, en caso de overflow, aumenta horas, en caso de overflow, reinicia
void CLOCK_GetHora(uint8_t*); //Guarda en el puntero pasado por parametro la hora actual
void CLOCK_ModHora(uint8_t horas); //Modifica las horas si el parametro esta entre 0 y 23
void CLOCK_ModMin(uint8_t minutos); //Modifica los minutos si el parametro esta entre 0 y 59
void CLOCK_ModSeg(uint8_t segundos); //Modifica los segundos si el parametro esta entre 0 y 59

#endif /* CLOCK_H_ */

#include "clock.h"

static uint8_t seconds=0; //Almacenamiento de horas, minutos y segundos
static uint8_t minutes=0;
static uint8_t hours=0;

void CLOCK_Init(uint8_t h, uint8_t m, uint8_t s){ //Inicia el reloj, sera llamado con la hora de compilacion
    CLOCK_ModHora(h);
    CLOCK_ModMin(m);
    CLOCK_ModSeg(s);
}

```

```
void CLOCK_Update(){ //Aumenta los segundos
    seconds++;
    if (seconds==60) //En caso de llegar a 60s, aumenta minutos y reinicia segundos
    {
        seconds=0;
        minutes++;
    }
    if (minutes==60) //En caso de llegar a 60 min, reinicia minutos y aumenta horas
    {
        minutes=0;
        hours++;
    }
    if (hours==24) //En caso de llegar a 24 horas, reinicia horas
    {
        hours=0;
    }
}
```

```
void CLOCK_GetHora(uint8_t* hAct){ //Guarda en el puntero de parametro [HH:MM:SS]
    hAct[0] = (hours/10) + '0';
    hAct[1] = (hours%10) + '0';
    hAct[2] = ':';
    hAct[3] = (minutes/10) + '0';
    hAct[4] = (minutes%10) + '0';
    hAct[5] = ':';
    hAct[6] = (seconds/10) + '0';
    hAct[7] = (seconds%10) + '0';
}
```

```
void CLOCK_ModHora(uint8_t h){ //Modifica la hora
    if(h<24 && h>=0){ //Verifica valores validos entre 0 y 23 horas
        hours=h;
    }
}
```

```

void CLOCK_ModMin(uint8_t h){ //Modifica los minutos
    if(h<60 && h>=0){ //Verifica valores validos entre 0 y 59 minutos
        minutes=h;
    }
}

void CLOCK_ModSeg(uint8_t h){ //Modifica los segundos
    if(h<60 && h>=0){ //Verifica valores validos entre 0 y 59 segundos
        seconds=h;
    }
}

```

Librerias teclado

```

#ifndef TECLADO_H_
#define TECLADO_H_

#include <avr/io.h>

#define TECLADO_DDRD DDRD
#define TECLADO_PORTD PORTD
#define TECLADO_PIND PIND
#define TECLADO_DDRB DDRB
#define TECLADO_PORTB PORTB
#define TECLADO_PINB PINB

void TECLADO_Init();
uint8_t KEYPAD_Scan(uint8_t*);

#endif /* TECLADO_H_ */

#include "teclado.h"
static uint8_t UpdateTeclado(void);
static const uint8_t matriz[16]={ '1','2','3','A','4','5','6','B','7','8','9','C','*','0','#','D'};
static const uint8_t cols[4] = {0x08, 0x20, 0x10, 0x04};
static const uint8_t fils[4] = {0xEF, 0xF7, 0xFE, 0x7F};
static uint8_t r,c;

```

```

static uint8_t UpdateTeclado(void){
    TECLADO_DDRD |= 0x80; //PIND7 como salida
    TECLADO_DDRD &= 0xC3; //PIND2,3,4,5 como entrada
    TECLADO_PORTD|= 0x3C; //Pull-up interno
    TECLADO_PORTD|= 0x80; //envia 1XXXXXXX al puerto D
    TECLADO_DDRB |= 0x19; //PINB4,3,0 como salida
    TECLADO_PORTB|= 0x19; //envia XXX11XX1 al puerto B

    //Barrido de las filas con puerto B
    for(r=0;r<3;r++){
        TECLADO_PORTB &= fils[r];
        for(c=0; c<4; c++){
            if(!(TECLADO_PIND & cols[c])){
                return matriz[r*4+c];
            }
        }
        TECLADO_PORTB |= ~(fils[r]);
    }

    //Barrido de las filas con puerto D
    TECLADO_PORTD &= fils[3];
    for(c=0; c<4; c++){
        if(!(TECLADO_PIND & cols[c])){
            return matriz[12+c];
        }
    }
    TECLADO_PORTD |= ~(fils[4]);

    return 0xFF;
}

```

```
uint8_t KEYPAD_Scan(uint8_t *pkey){
    static uint8_t Old_key, Last_valid_key=0xFF;
    uint8_t Key;
    Key = UpdateTeclado();
    if(Key==0xFF){
        Old_key=0xFF;
        Last_valid_key=0xFF;
        return 0;
    }
    if(Key==Old_key){
        if(Key!=Last_valid_key){
            *pkey=Key;
            Last_valid_key=Key;
            return 1;
        }
    }
    Old_key=Key;
    return 0;
}
```

Librerías LCD

Para este caso, hemos usado las mismas librerías provistos por la cátedra. Por lo tanto no mostraremos en este informe las librerías, aunque si lo hemos implementado.

Librerías MEF

```

#ifndef MEF_H_
#define MEF_H_

#include "clock.h"
#include "lcd.h"
#include "timer.h"
#include "teclado.h"

typedef enum {CERRADO, PASSWORD, ABIERTO, DENEGADO, HORA, MINUTO, SEGUNDOS} MEF_STATE;
uint8_t* key;
uint8_t State_call_count;
MEF_STATE System_state;
uint8_t password[4];

void CERRADURA_Init();
void CERRADURA_Update(void);

#endif /* MEF_H_ */

```

```

#include "mef.h"
static uint8_t cantDigitos;
static uint8_t horaIngresada;
static uint8_t cerrado[] = "CERRADO ";
static uint8_t abierto[] = "ABIERTO ";
static uint8_t denegado[] = "DENEGADO";
static uint8_t cantTecla = 0;
static uint8_t hora[8];
static uint8_t horaActual= 0;
static uint8_t counterAux=0;
static void BreakCerrado(void);
static void ingPass(void);
static void ingPassContinue(void);
static void stateCerrado(void);
static void stateAbierto(void);
static void stateDenegado(void);
static void cambiar_Hora(MEF_STATE);
static void LCDHora(void);
static void LCDMinutos(void);
static void LCDSegundos(void);
static uint8_t Verificar_Password(void);

```

```

void CERRADURA_Update(void)
//llamada cada 100 ms
{
    //Contar numero de interrupciones
    State_call_count++;
    horaActual++;

    switch(System_state)
    {
        case CERRADO:
            //Se fija si se apreta por teclado
            if (KEYPAD_Scan(key)){
                //Se fija si ingresa contraseña
                if( (*key == '0') || (*key == '1') || (*key == '2') || (*key == '3') || (*key == '4') ||
                    (*key == '5') || (*key == '6') || (*key == '7') || (*key == '8') || (*key == '9') ){
                    ingPass();
                    break;
                }

                //Se fija si se ingresa algun cambio de horario
                if (*key == 'A'){
                    cambiar_Hora(HORA);
                    break;
                }
                if (*key == 'B'){
                    cambiar_Hora(MINUTO);
                    break;
                }

                if (*key == 'C'){
                    cambiar_Hora(SEGUNDOS);
                    break;
                }
            }else{
                BreakCerrado();
            }
        break;
    }
}

```

```

case PASSWORD:
    if(KEYPAD_Scan(key) && cantTecla<4){
        if (*key != 'A' && *key != 'B' && *key != 'C' && *key != 'D' && *key != '*' && *key != '#'){
            ingPassContinue(); //Si se ingreso una tecla distinta de A, B, C, D, * o #, entonces agrega a la contrasena
        }
    }
    if(cantTecla == 4){
        //Verifico que ingreso bien la contraseña
        if (Verificar_Password()){
            System_state = ABIERTO;
        } else {
            System_state = DENEGADO;
        }
        State_call_count = 0;
        counterAux = 0;
        break;
    }

//Esperar 15 segundos para que ingrese la clave, sino denegado
if(++State_call_count > 99){
    if(++counterAux > 30){
        System_state = DENEGADO;//si bien la mef es llamada cada 10ms ...
        counterAux = 0; //... las cuentas seran dentro de 19 seg porque tarda algunos ciclos de reloj en otras instrucciones
    }
    State_call_count = 0;
    break;
}
break;

```



```

case ABIERTO:
    if(State_call_count==1){ //Muestra en el display "ABIERTO" la primera vez que entra
        stateAbierto();
    }
    if (++State_call_count > 16) // Necesitamos contar hasta 3000 ms
    {
        // La mef se actualiza cada 10ms, por lo que necesitamos 300 llamados
        if (++counterAux > 29) // Como State_call_count es uint_8 solo llega a 255
        {
            // Por lo que anidamos 2 If contando hasta 16 y luego 29
            counterAux = 0; // Si bien 16x29x10ms = 4640 ms, termina entrando a los 3000ms
            stateCerrado(); //Esto se da por los retardos del programa
        }
        State_call_count = 0;
    }
    break;

case DENEGADO:
    if(State_call_count==1){ //Muestra en el display "DENEGADO" la primera vez que entra
        stateDenegado();
    }
    if (++State_call_count > 16) // Necesitamos contar hasta 3000 ms
    {
        // La mef se actualiza cada 10ms, por lo que necesitamos 300 llamados
        if (++counterAux > 19) // Como State_call_count es uint_8 solo llega a 255
        {
            // Por lo que anidamos 2 If contando hasta 16 y luego 19
            counterAux = 0; // Si bien 16x19x10ms = 3040 ms, termina entrando a los 2000ms
            stateCerrado(); //Esto se da por los retardos del programa
        }
        State_call_count = 0;
    }
    break;

case HORA:
    if(KEYPAD_Scan(key)){
        if (*key != 'A' && *key != 'B' && *key != 'C' && *key != 'D' && *key != '*' && *key != '#')
        {
            //Si se ingreso una tecla distinta de A, B, C, D, * o #, entonces sumo a la hora
            if (!cantDigitos) //Primer dígito
            {
                cantDigitos++;
                hora[0]=*key; //Decena hora almacenada en hora[0] Hh:mm:ss
                horaIngresada= (*key - '0') *10; //Transformo de caracter a numero para poder validarlo mas tarde
            }else if(cantDigitos==1) //Segundo dígito
            {
                cantDigitos++;
                hora[1]=*key; //Unidad hora en hora[1] hH:mm:ss
                horaIngresada+= (*key-'0'); //Transformo de caracter a numero para poder validarlo mas tarde
            }
            LCDHora(); //Feedback al usuario de hora ingresada
        }
    }
}

```

```

    }else if(*key == 'A') //Guardar
    {
        CLOCK_ModHora(horaIngresada); //Verifico y actualizo la hora en caso de ser correcta
        LCDcursorOFF();
        stateCerrado(); //Cambio a estado CERRADO
    }else if(*key == '#') //Cancelar
    {
        LCDcursorOFF(); //Apago el cursor
        stateCerrado(); //Cancelo los cambios, solo cambio de estado
    }
}
break;

case MINUTO:
    if(KEYPAD_Scan(key)){
        if (*key != 'A' && *key != 'B' && *key != 'C' && *key != 'D' && *key != '*' && *key != '#')
        {
            //Si se ingreso una tecla distinta de A, B, C, D, * o #, entonces sumo a los minutos
            if (!cantDigitos) //Primer digito
            {
                cantDigitos++;
                hora[3]=*key; //Decena minutos almacenada en hora[3] hh:Mm:ss
                horaIngresada= (*key - '0') *10; //Transformo de caracter a numero para poder validarlo mas tarde
            }else if(cantDigitos==1) //Segundo digito
            {
                cantDigitos++;
                hora[4]=*key; //Unidad minutos en hora[4] hh:mM:ss
                horaIngresada+= (*key-'0'); //Transformo de caracter a numero para poder validarlo mas tarde
            }
            LCDMinutos(); //Feedback al usuario de hora ingresada
        }else if(*key == 'B') //Guardar
        {
            CLOCK_ModMin(horaIngresada); //Verifico y actualizo la hora en caso de ser correcta
            LCDcursorOFF();
            stateCerrado(); //Cambio a estado CERRADO
        }

        }else if(*key == '#') //Cancelar
        {
            LCDcursorOFF();
            stateCerrado(); //Cambio a estado CERRADO
        }
    }
break;

case SEGUNDOS:
    if(KEYPAD_Scan(key)){
        if (*key != 'A' && *key != 'B' && *key != 'C' && *key != 'D' && *key != '*' && *key != '#')
        {
            //Si se ingreso una tecla distinta de A, B, C, D, * o #, entonces sumo a los minutos
            if (!cantDigitos)
            {
                cantDigitos++;
                hora[6]=*key; //decena segundos en hora[6] hh:mm:ss
                horaIngresada= (*key - '0') *10; //Transformo de caracter a numero para poder validarlo mas tarde
            }else if(cantDigitos==1)
            {
                cantDigitos++;
                hora[7]=*key; //unidad segundos en hora[7] hh:mm:ss
                horaIngresada+= (*key-'0'); //Transformo de caracter a numero para poder validarlo mas tarde
            }
            LCDSegundos(); //Feedback al usuario de hora ingresada
        }
    }

```

```

    }else if(*key == 'C') //Guardar cambios
    {
        CLOCK_ModSeg(horaIngresada); //verifico y actualizo los segundos
        LCDcursorOFF();
        stateCerrado(); //Cambio estado a cerrado
    }else if(*key == '#') //Cancelar
    {
        LCDcursorOFF();
        stateCerrado(); //Cambio estado a cerrado sin actualizar los segundos
    }
}
break;
}
}

```

```

void cambiar_Hora(MEF_STATE state){ //Pasaje de estado a Cambiar Hora
    System_state = state;
    State_call_count = 0;
    cantDigitos = 0;
    horaIngresada= 100; //numero grande (23<59<100) el cual no puede ser guardado en caso de apretar 2 veces seguidas A, B o C
    LCDhome();
    LCDstring((uint8_t*)"          ",16); //Limpi display
    LCDGotoXY(4,0);
    LCDstring(hora, 8); //Escribo hora
    switch(state){ //segun fue presionado A, B o C la funcion fue llamada con un state distinto
        case HORA:
            LCDGotoXY(4,0);
            LCDcursorOnBlink(); //Posiciono el cursor sobre la decena de la hora para ser modificada
            break;
        case MINUTO:
            LCDGotoXY(7,0); //Posiciono el cursor sobre la decena de los minutos para ser modificada
            LCDcursorOnBlink();
            break;
        case SEGUNDOS:
            LCDGotoXY(10,0); //Posiciono el cursor sobre la decena de los segundos para ser modificada
            LCDcursorOnBlink();
            break;
        default: //Default esta porque nunca es llamado para los state CERRADO, ABIERTO, PASSWORD, etc
            LCDGotoXY(4,0);
            LCDcursorOnBlink();
            break;
    }
    LCDcursorOFF(); //Apaga cursor
}
}

```

```
void LCDHora(void){ //Llamado desde el cambiar hora para cuando el usuario ingresa una nueva hora
    LCDhome();
    LCDstring((uint8_t*)"                ",16);
    LCDGotoXY(4,0);
    LCDstring(hora, 8);
    LCDGotoXY(5,0);
}

void LCDMinutos(void){ //Llamado desde el cambiar hora para cuando el usuario ingresa minutos
    LCDhome();
    LCDstring((uint8_t*)"                ",16);
    LCDGotoXY(4,0);
    LCDstring(hora, 8);
    LCDGotoXY(8,0);
}

void LCDSegundos(void){ //Llamado desde el cambiar hora para cuando el usuario ingresa segundos
    LCDhome();
    LCDstring((uint8_t*)"                ",16);
    LCDGotoXY(4,0);
    LCDstring(hora, 8);
    LCDGotoXY(11,0);
}

void CERRADURA_Init(){ //Inicializa en estado CERRADO y el count de ticks en 0
    System_state = CERRADO;
    State_call_count = 0;
}

uint8_t Verificar_Password(void){ //Checkea que las psw sea 5193
    if (password[0] == '5' && password[1] == '9' && password[2] == '1' && password[3] == '3'){
        return 1;
    }
    return 0;
}
```

```
static void BreakCerrado(void)
{
    if(State_call_count==1){ //Primera vez que entra
        LCDstring((uint8_t*)"          ",16);
        LCDGotoXY(4,0);
        CLOCK_GetHora(hora); //Muestra hora en el primer renglon
        LCDstring(hora, 8);
        LCDGotoXY(4,1);
        LCDstring(cerrado, 8); //Muestra CERRADO en el segundo renglon
        LCDhome();
        LCDGotoXY(3,0); //Se posiciona para actualizar la hora
    }
    if(horaActual >= 10){ //Cada 100ms
        CLOCK_GetHora(hora); //Muestra la hora
        LCDGotoXY(3,0);
        LCDstring(hora, 8);
        horaActual = 0; //Reinicia counter
        LCDhome();
        LCDGotoXY(3,0); //Se posiciona para actualizar la hora
    }
}
```

```

static void ingPass(void){
    State_call_count=0; //Cambia estado a PASSWORD donde estaremos ingresando la contrasena
    System_state = PASSWORD;
    password[cantTecla] = *key; //Guarda el numero presionado
    cantTecla++;
    LCDstring((uint8_t*)"          ",16);
    LCDGotoXY(0,1);
    LCDstring((uint8_t*)"          ",16); //Limpia todo el display
    LCDGotoXY(6,1);
    LCDsendChar('*'); //Coloca * en lugar del digito presionado
}

static void ingPassContinue(void){ //Llamado al presionar el segundo, tercer y cuarto digito de la contrasena
    password[cantTecla] = *key; //Guarda el valor presionado
    cantTecla++;
    LCDsendChar('*'); //Manda * en lugar del digito
}

static void stateCerrado(void){ //Cambia a estado cerrado
    LCDcursorOFF();
    State_call_count=0;
    cantTecla=0; //Reinicia counters
    horaActual=0;
    System_state=CERRADO;
}

static void stateAbierto(void){ //Muestra en el display el mensaje DENEGADO
    LCDhome();
    LCDstring((uint8_t*)"          ",16);
    LCDGotoXY(4,1);
    LCDcursorOFF();
    LCDstring(abierto,8);
}

static void stateDenegado(void){ //Muestra en el display el mensaje DENEGADO
    LCDhome();
    LCDstring((uint8_t*)"          ",16);
    LCDGotoXY(4,1);
    LCDcursorOFF();
    LCDstring(denegado,8);}

```