CERRADURA ELECTRONICA

Trabajo practico n°2

Universidad Nacional de La Plata
Facultad de Ingeniería
Circuito digitales y Microcontroladores

Pérez, Francisco
Molinuevo, Gallo Francisco

Índice

Introducción

Se está bajo un problema el cual se necesita de la aplicación de varios conceptos (nuevos y ya utilizados) para poder realizar eficientemente la solución.

Se utilizará el ATMEGA328P con los componentes que este otorga como temporización (Timers) e interrupciones las cuales se extenderá la explicación del uso de estas más adelante.

El primer paso a realizar es dividir el problema en partes (módulos) otorgando mayor facilidad, organización, reutilización y mantenimiento de código.

1. **Modularización**: cada módulo en el proyecto se enfocará en una tarea específica el cual cada uno tendrá su conjunto de funciones y estructuras de datos relacionadas. Esto permite una mayor coherencia y claridad en el diseño.

Un módulo presenta una interfaz bien definida con puntos de entrada y salida para comunicarse con el resto

1.2 **Programación Modular en C**: C nos ofrece una forma de separar estos módulos mediante 2 tipos de archivo. Estos se denominan Header y archivos fuente.

Hay que tener en claro también la sintaxis para saber cómo declarar ya sean funciones o variables privadas al módulo.

No es recomendado utilizar variables globales para la comunicación entre ellos por lo tanto cada uno tendrá variables privadas y funciones para obtener estos valores o, en el caso de que sea necesario, funciones para asignarles un valor.

Si una variable y/o función contiene el modificador *"static"* esta es privada del archivo correspondiente y no es posible que sea vista por otro módulo.

El archivo Header (***.h***) nos proporciona la declaración de las funciones y variables a utilizar dentro de ese módulo mientras que el archivo fuente (***.c***) la implementación de estas funciones declaradas en él ***.h***

1.3 Módulos de software:

Analizando el problema a resolver se pueden identificar varios módulos para los cuales tendrán sus propios archivos:

1.3.1 Módulo MEF:

Representa la Maquina de Estados Finita. Las funciones declaradas en el header denominado **MEF.h** son:

```
void MEF_init(void);

uint8_t KEYPAD_update(uint8_t *);

void MEF_update(void);
```

Y la declaración de los estados dada por:

```
typedef enum {Inicio, D0, D1, D2, D3, Correcto, Incorrecto, EdicionH, EdicionM,
EdicionS} state;
```

Dentro de este módulo se tendrá algunas funciones privadas el cual principalmente son para modularizar algoritmos como la transición de estados o el vencimiento de uno. A vencimiento se denomina a que, dentro de algunos estados, luego de estar durante un tiempo determinado sin hacer nada o lo requerido, transiciona al estado 'Inicio'

Las funciones privadas son:

```
static void handleInicio(uint8_t *);
```

Esta función determina a que estado transicionar dependiendo la lectura del teclado

```
Según la tecla leída
   Si es 'A'
      Ir al estado de edición de hora
   Si es 'B'
      Ir al estado de edición de minutos
   Si es 'C'
      Ir al estado de edición de segundos
   Si es un numero
      Ir a la primera transición de estados correspondiente a la lectura de contraseña
```

```
static uint8_t MEF_readKey(uint8_t);
```

Esta función llama a la lectura del teclado del módulo del Keypad y si lee, guarda que tecla fue presionada en una posición dada de un vector. La posición en la que se guarda depende en el estado en el que este.

```
static void resetCallCountAndGoToState(uint8_t, state );
```

Por último, mediante esta función se maneja el vencimiento de un estado que depende de cada estado, si la cantidad de veces que se entró a un mismo estado es igual su número de vencimiento (en milisegundos) vuelve al estado inicio. Por ejemplo, dentro del estado de 'Correcto' luego de 3 segundos debe volver al estado 'Inicio.' Esta función es la encargada de llevar a cabo tal acción.

El módulo MEF se abordará de forma más extendida en capítulos más adelante por la gran cantidad de conceptos y situaciones a resolver.

Para finalizar, dentro de este módulo se mantendrá algunas variables para manipular la lógica general. Entre ellas una variable para ir contando cuantas veces se entró a la actualización de la MEF, otra para guardar las letras leídas y otra para definir los estados.

1.3.2 **Módulo WATCHER**:

Este módulo representa al reloj en tiempo real el cual llevará variables privadas accesibles mediante **getters()** para obtener información sobre el tiempo transcurrido. De esta forma mientras se están ejecutando otros módulos en concurrencia al reloj, este puede seguir

contando en segundos sin preocuparse a algún bloqueo que haga la desincronización del conteo. Las variables que tenemos dentro de este módulo son:

```
static uint8_t seg;
static uint8_t min;
static uint8_t hour;
```

Por otro lado, tenemos las funciones públicas de este módulo:

```
uint8_t WATCH_getSeg();
uint8_t WATCH_getMin();
uint8_t WATCH_getHour();
void WATCH_setHour(uint8_t);
void WATCH_setMinutes(uint8_t);
void WATCH_setSeconds(uint8_t);
void WATCH_update();
```

Donde tenemos las típicas funciones getters y setters para las variables privadas y la función `void WATCH_update();` el cual hace la lógica y validación de estas variables privadas (ver **Pseudocódigo 1.3.2.a**)

```
Incrementar 'seg' por 1
Si seg es igual a 60
   Reinicio los segundos
      Incremento un minuto
      Si 'min' es igual a 60
           Reinicio los minutos
           Incremento una hora
           Si 'hs' es igual a 24
                Reinicio la hora
```

Pseudocodigo 1.3.2.a *WATCH_update()*

1.3.3 Módulo sEOS:

Este módulo le corresponde al planificador de tareas el cual también se extenderá su explicación en capítulos más adelante. En este caso utilizamos este módulo para representar la lectura y reinicio de los flags (actualizados periódicamente cada cierto tiempo). Estos flags nos determinan qué acción realizar en ese instante de tiempo, llamando la función de otros módulos para hacer tal acción.

Las funciones declaradas en el header son:

```
void TIMER0_Init(void);

void MEF_update();

void sEOS_Dispatch_Tasks(void);

void update_WATCH();
```

Donde se puede observar como utiliza las funciones de otros módulos ya que este se encarga de avisarles que acción realizar (temporizadas).

El archivo fuente debe incluir al módulo de MEF y módulo de WATCH para poder conocer estas funciones. Siendo así:

```
#include "MEF.h"
#include "WATCH.h"
```

Dentro de este módulo tendremos la rutina de interrupción que genera nuestro timer, encargado de realizar el levantamiento de flag en el instante de tiempo deseado.

1.3.4 **Módulo Keypad:**

Este módulo es el responsable de hacer la lectura del teclado 4x4 (el cual será explicado posteriormente) donde contiene una única función que retorna '0' o '1' si se pulso una tecla. Se manda un parámetro como referencia que retorna, en caso de ser así, el valor ASCII de la tecla presionada correspondiente.

La función es:

```
uint8_t KEYPAD_Scan(uint8_t *);
```

1.3.5 **Módulo COMMON**

En este caso, este solo contiene un Header (***.h***) el cual no presenta ni agrega implementaciones a la resolución, se utiliza para mejorar la legibilidad del programa y contiene todos los `#includes` que se necesitan en todos o la mayoría de los archivos fuentes. Su contenido es:

```
#ifndef COMMON
#define COMMON
#include <avr/io.h>
#include "sEOS.h"
#include <avr/interrupt.h>
#define F_CPU 16000000UL
#include <util/delay.h>
#include <stdint.h>
#include <stdlib.h>

#endif
```

De esta forma, los que contengan este .h van a conocer las librerías necesarias para el manejo de ciertas características.

1.3.6 **Módulo Main**

Aquí está el loop infinito que ejecutará el microcontrolador, donde primeramente se inicializará los componentes necesarios como el Timer, la máquina de estados y el LED. Estas inicializaciones son estrictamente necesarias. Por ejemplo, es necesario tener un primer estado conocido en la máquina de estados; En el Timer es necesario la modificación de los registros de configuración del mismo y, por último, el LED tiene una forma de inicialización dada por el fabricante donde se debe cumplir rigurosamente los tiempos de estos. Se hablará más adelante sobre cada una de las funciones de inicialización.
Dentro del loop infinito estará en ejecución el despachador del que se habló anteriormente.

1.3.7 **Módulo LCD**

Este es un módulo que fue puesto a disposición por la cátedra para la correcta utilización del LED 16x2 con determinadas funciones que nos facilitan el manejo de este, entre ellas:

```
void LCDescribeDato(int val,unsigned int field_length); // Agrego Funcion para
Describir Enteros
void LCDsendChar(uint8_t);         //forms data ready to send to 74HC164
void LCDsendCommand(uint8_t);      //forms data ready to send to 74HC164
void LCDinit(void);                //Initializes LCD
void LCDclr(void);                 //Clears LCD
void LCDhome(void);                //LCD cursor home
void LCDstring(uint8_t*, uint8_t); //Outputs string to LCD
void LCDGotoXY(uint8_t, uint8_t);  //Cursor to X Y position
void CopyStringtoLCD(const uint8_t*, uint8_t, uint8_t);//copies flash string to LCD
at x,y
void LCDdefinechar(const uint8_t *,uint8_t);//write char to LCD CGRAM
void LCDshiftRight(uint8_t);       //shift by n characters Right
void LCDshiftLeft(uint8_t); //shift by n characters Left
void LCDcursorOn(void);            //Underline cursor ON
void LCDcursorOnBlink(void);       //Underline blinking cursor ON
void LCDcursorOFF(void);           //Cursor OFF
void LCDblank(void);               //LCD blank but not cleared
void LCDvisible(void);                    //LCD visible
void LCDcursorLeft(uint8_t);       //Shift cursor left by n
void LCDcursorRight(uint8_t);      //shif cursor right by n

void LCD_Init();
```

Dentro del Header tenemos varias variables públicas y privadas que no es necesario su desarrollo en el informe.

2. **Display LCD:**

Este dispositivo permite presentar información en formato de texto. Su estructura física incluye ocho pines donde se escriben o reciben los bits, un pin RW que permite elegir si la operación es de escritura o lectura, un pin Enable (E) para activar o desactivar la funcionalidad del display, y otro pin de Register Select (RS), que permite distinguir entre la escritura de datos y la selección de comandos.

2.1 **Introducción:**

Se utilizará este dispositivo con el propósito de dar visibilidad al usuario del estado operativo de la cerradura electrónica, así como también permitir la visualización de la información horaria.

2.2 **Biblioteca:**

A lo largo del trabajo práctico presentado, nos apoyamos en una biblioteca de funciones para manipular el display, la cual facilita y acelera el desarrollo del mismo. Esta cuenta con numerosas funcionalidades, sin embargo, utilizamos un pequeño porcentaje. Ver (funciones-1)

- `void LCDinit(void);` //Initializes LCD
- `void LCDclr(void);` //Clears LCD
- `void LCDhome(void);` //LCD cursor home
- `void LCDGotoXY(uint8_t, uint8_t);` //Cursor to X Y position
- `void LCDsendChar(uint8_t);` //forms data ready to send to LCD
- `void LCDstring(uint8_t*, uint8_t);` //Outputs string to LCD
- `void LCDescribeDato (int val,unsigned int field_length);` // Funcion para escribir enteros

Funciones-1

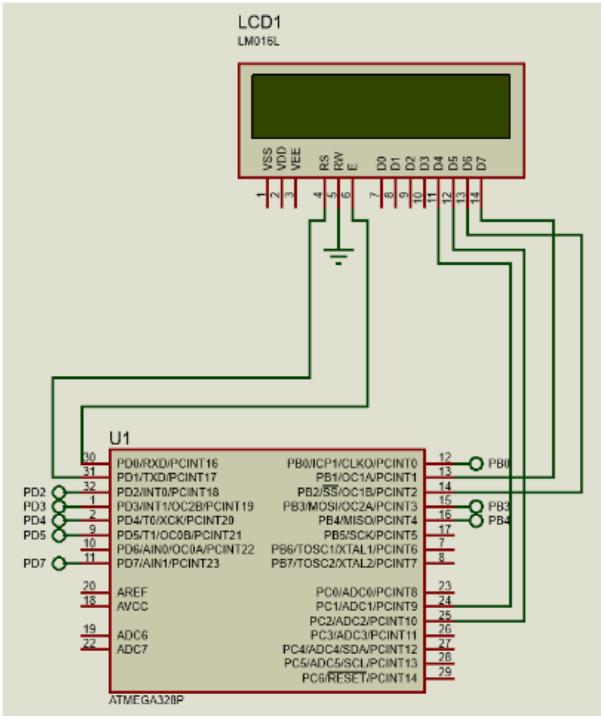
2.3 **Conexión:**

Como mencionamos anteriormente, contamos con 8 bits para intercambiar información, desde D0 a D7. De estos, solamente utilizaremos 4 y profundizaremos más sobre esto posteriormente.

El pin RW va a tierra, para así establecer el modo de escritura, lo cual significa que vamos a enviar datos al display para su visualización. Cabe destacar que al hacerlo de esta manera se pierde la capacidad de leer datos del LCD, pero no es un problema ya que no se necesita hacerlo en el desarrollo.

En cuanto al pin E del display, se conectará al PD0 del MCU para habilitar la transferencia de datos.

Por último, el pin RS del LCD se conecta al PD1 del microcontrolador, controlando así el tipo de dato que enviaremos al display. (ver *conexionado-1*)



Conexionado-1 pin-LCD

2.3.1 **4-bit mode:**

Para ahorrar terminales del microcontrolador y transmitir datos de forma más eficiente, el LCD puede configurarse para trabajar con solo 4 bits, realizando la conexión en los pines D4-D7. Con esta técnica, los datos se envían en dos paquetes de 4 bits cada uno (dos accesos de escritura), en lugar de enviar los 8 bits completos de una sola vez. Enviamos primero el nibble más alto y luego el nibble más bajo.

2.3.1.a **Inicializacion**

A destacar una función que se utiliza es la inicialización del LED, el cual requiere de una serie de pasos estrictos para su correcto funcionamiento, esta inicialización no fue desarrollada si no que es una función dentro de la librería. Esta inicialización depende del fabricante (ver ilustración-2)

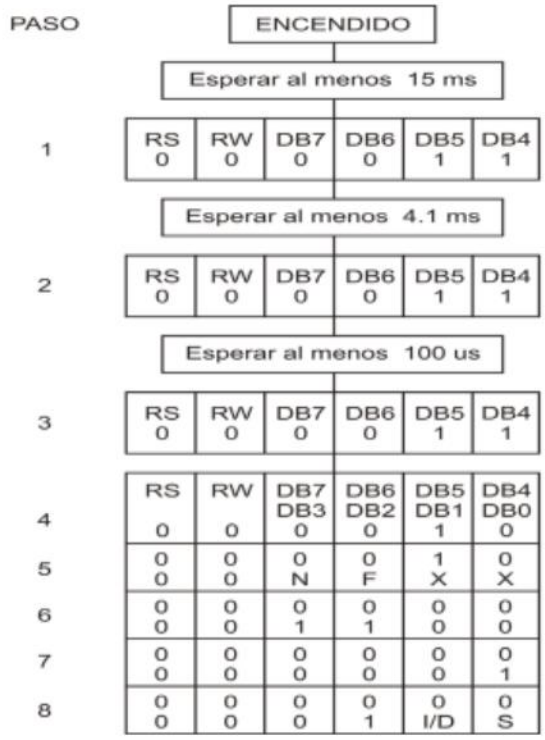


Ilustración-2

2.3.1.b **Resolución:**

En cuanto al desarrollo del programa, se utiliza a lo largo de todo el código de la MEF las funciones provistas por la librería, ya sea para limpiar la visualización del display, posicionar el cursor del LCD en una ubicación específica, escribir un dato, enviar un carácter, entre otros.

3. **Teclado matricial 4x4.**

Este teclado es una matriz de 16 pulsadores distribuidos en filas y columnas, los cuales se utilizan para enviar información.

3.1 **Introducción.**

Es de interés detectar cuando una tecla es presionada, para esto, las filas y columnas del teclado están conectadas a los pines de entrada/salida del MCU. Cuando una tecla es pulsada, se tiene la posición exacta del contacto, dado que corresponde a una combinación particular de fila y columna.

3.2 **Resolución.**

Para identificar la tecla presionada, se utiliza un algoritmo que mapea la combinación fila-columna correspondiente. (ver algoritmo-1)

```
uint8_t KEYPAD_Scan(uint8_t *key){
        // Entradas
        DDRB &= (0b11100110);
        DDRD &= (0b01111111);

        PORTB |= (0b00011001);
        PORTD |= (0b10000000);

        // Salidas
        DDRD |= (0b00111100);

        uint8_t mascarasPINB[3] = {0b00010000, 0b00001000, 0b00000001};
        uint8_t mascaraPIND = 0b10000000;

        for (uint8_t c = 0; c < 4; c++) {
            PORTD |= (0b00111100);
            PORTD &= ~(1 << (c+2));

            for (uint8_t f = 0; f < 3; f++) {
                if(!(PINB & mascarasPINB[f])){
                    *key = matriz[f][c];
                    return 1;
                }
            }
            if(!(PIND & mascaraPIND)){
                *key = matriz[3][c];
                return 1;
            }
        }
        return 0;
```

Algoritmo-1 KEYPAD_Scan();

En primer lugar, se define la matriz que contiene los caracteres y símbolos asignados a cada tecla del teclado.
La función *KEYPAD_Scan* es la encargada de realizar el escaneo.

Se configuran los pines correspondientes como entradas y salidas respectivamente. Los pines del puerto B se configuran como entradas y pull-up, y los del puerto D como salidas. Luego se definen los arreglos de máscaras que se utilizan para realizar las comparaciones durante el escaneo del teclado.

Dentro del bucle, en cada iteración, se activa una columna específica estableciendo un bit en bajo y los demás en alto. Aquí se define otro bucle, para recorrer las filas, y en cada

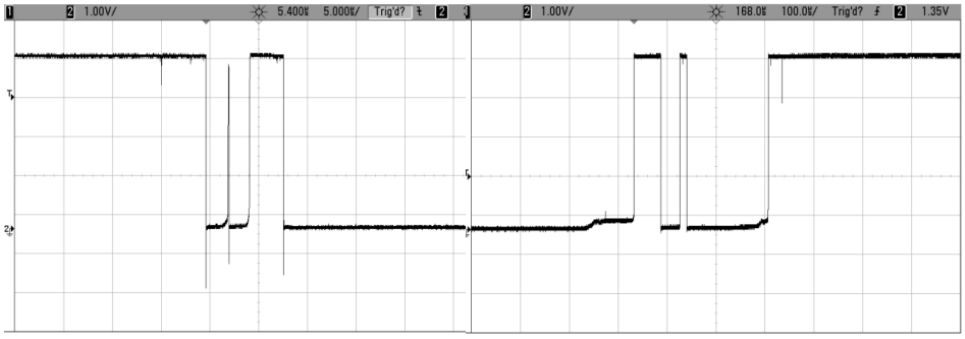
paso se verifica si se ha detectado una tecla presionada para una combinación de fila y columna en particular.

Si detectamos una tecla presionada, se asigna el valor correspondiente en la matriz a **key* y retornamos el valor 1. Si no detectamos ninguna entrada, se pasa a la siguiente iteración del bucle de columnas.

En caso de no detectar ninguna tecla presionada en la totalidad de las iteraciones, retornamos el valor 0.

3.2.1 Efecto rebote.

Este fenómeno aparece cuando los pulsadores son accionados. Los contactos deben moverse físicamente desde una posición a otra, y esto produce un “rebote” mecánico, ya que el circuito se abre y se cierra múltiples veces hasta que se estabiliza. Esto provoca fluctuaciones en la señal, y se leen valores múltiples veces en poco tiempo, lo cual es indeseado. (ver grafica-1)



Grafica-1 Efecto rebote

Se puede observar que hay un determinado tiempo hasta que la señal tiene un valor constante correcto. Por lo que deriva a una mal lectura de la pulsación.

3.2.2 Solución.

Para solucionar esto y no tener efectos adversos en el funcionamiento del programa, lo que se hizo fue llevar la cuenta de cuanto teníamos que esperar dentro del estado de la MEF para consultar el teclado, con una variable llamada *call_count_keypad*, para consultar el teclado cada un determinado tiempo.

3.3. **Funcionamiento.**

Particularmente, queremos consultar el estado del teclado cada 300 milisegundos (suficiente para eliminar el efecto rebote y dando por sentado que el usuario no presionara más de 3 veces por segundo), es decir, cuando *call-count-keypad* sea igual a tres dentro del estado de la MEF. Este chequeo se realiza en varias oportunidades dentro del código que corresponde a la implementación de la máquina de estados.

Decimos que *call_count_keypad* tiene que ser 3 ya que se definió que la actualización de estados de la MEF es cada 100ms, por lo tanto, se sabe que todo lo que está relacionado a ejecución dentro de la MEF es cada 100ms

El valor de 300 milisegundos corresponde al tiempo aproximado durante el cual se espera que se haya estabilizado el teclado después del efecto rebote.

3.4 Problemas con el escaneo del teclado.

En un primer momento, realizamos un escaneo bloqueante, el cual no era óptimo. En esta situación, la operación de escaneo detiene la ejecución del programa hasta que el botón fuese soltado. Esto significa que el programa se queda bloqueado hasta que el usuario lo deje de presionar, de esta forma, se perdería cualquier tipo de temporización ya que estamos trabajando con un solo timer.

El escaneo bloqueante es simple de implementar y entender, ya que las operaciones se realizan de manera secuencial y sincrónica. Sin embargo, tiene un impacto negativo en el rendimiento y la capacidad de respuesta del sistema.

3.4.1. Escaneo no bloqueante.

En el escaneo no bloqueante, la operación de escaneo no detiene la ejecución del programa y permite que otras instrucciones o tareas se no se vean afectadas mientras se espera el resultado del escaneo.

Permite una mayor concurrencia y eficiencia en el sistema, ya que se pueden realizar múltiples tareas simultáneamente y se evita la espera innecesaria.

Este enfoque no bloqueante permite que el programa principal continúe ejecutándose sin interrupciones mientras se verifica el estado del teclado matricial. El escaneo periódico asegura que las teclas presionadas sean detectadas en un intervalo de tiempo razonable, y la actualización del estado permite que el programa acceda a la información de las teclas presionadas.

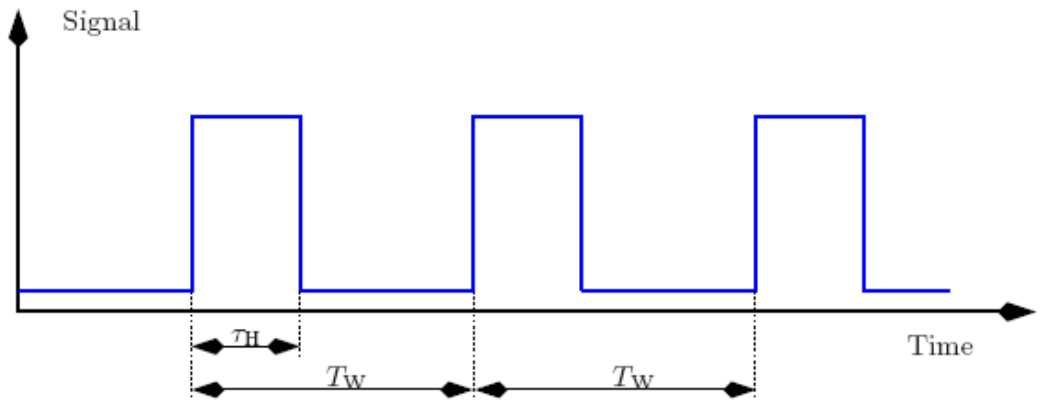
4. **Temporizadores:**

4.1 **Introducción:**

Una de las características más importantes de un MCU es la capacidad de realizar tareas temporizadas. Para esto cuentan con un periférico denominado TIMER o TEMPORIZADOR. Algunas tareas para lo cual se usa son:

- Generación de retardos
- Interrupción periódica
- Registro y conteo de eventos (COUNTER)

Las propiedades de la temporización son: el periodo, frecuencia y ciclo de trabajo. Donde el periodo se define como el tiempo que tarda la señal en hacer un ciclo entero. La frecuencia define el número de oscilaciones dentro de 1 segundo y, el ciclo de trabajo es el porcentaje de tiempo en que la señal esta activa con respecto del periodo total



Siendo Tw el periodo y th el tiempo activo de la señal.

4.2 **Interrupciones:**

Para operar correctamente con el TIMER se debe estudiar el concepto de interrupción dentro de una CPU.

Las interrupciones son un mecanismo para poder interactuar con periféricos y así darles una respuesta adecuada a estos. El TIMER, es un periférico que genera una interrupción temporizada, concretamente es una interrupción interna.

Cada interrupción debe estar en el vector de interrupciones almacenados en memoria, este vector contiene las rutinas de interrupción (ISR) de cada una.

Consultando la hoja de datos del MCU:

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External pin, power-on reset, brown-out reset and watchdog system reset
2	0x002	INT0	External interrupt request 0
3	0x0004	INT1	External interrupt request 1
4	0x0006	PCINT0	Pin change interrupt request 0
5	0x0008	PCINT1	Pin change interrupt request 1
6	0x000A	PCINT2	Pin change interrupt request 2
7	0x000C	WDT	Watchdog time-out interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 compare match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 compare match B
10	0x0012	TIMER2 OVF	Timer/Counter2 overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 capture event
12	0x0016	TIMER1 COMPA	Timer/Counter1 compare match A
13	0x0018	TIMER1 COMPB	Timer/Counter1 compare match B
14	0x001A	TIMER1 OVF	Timer/Counter1 overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 compare match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 compare match B
17	0x0020	TIMER0 OVF	Timer/Counter0 overflow
18	0x0022	SPI, STC	SPI serial transfer complete
19	0x0024	USART, RX	USART Rx complete
20	0x0026	USART, UDRE	USART, data register empty
21	0x0028	USART, TX	USART, Tx complete
22	0x002A	ADC	ADC conversion complete
23	0x002C	EE READY	EEPROM ready
24	0x002E	ANALOG COMP	Analog comparator
25	0x0030	TWI	2-wire serial interface
26	0x0032	SPM READY	Store program memory ready

Podemos ver en el vector no. 8 al 17 las fuentes de interrupción de los TIMERS. Donde a menor número, mayor prioridad.

En C, con la sintaxis ***ISR('Nombre vector de interrupción')*** se puede realizar dentro de esta función, el código que se crea necesario para el ejecución de la petición.

También debemos incluir un archivo de cabecera necesario:

```
#include <avr/interrupt.h>
```

Por otro lado, se debe habilitar las interrupciones para que la CPU esté dispuesta a atenderlas, caso contrario, algunas se perderán o se encolaran dependiendo el tipo de interrupción.

La forma de habilitarlas es mediante la función ***sei()***

4.3 **Temporizadores en AVR**

El Atmega328p cuenta con varios temporizadores, específicamente 3 (TIMER0, TIMER1, TIMER2) donde cada uno de estos contiene características específicas y diferentes modos de operación

Modos de operación:

Modo normal: El temporizador cuenta desde 0 hasta un valor de comparación y luego se reinicia. Puede generar interrupciones cuando se alcanza el valor de comparación.

Modo CTC (Comparación de temporizador con constante): El temporizador cuenta desde 0 hasta un valor de comparación y luego se reinicia. Puede generar interrupciones cuando se alcanza el valor de comparación.

Luego contiene 2 modos de operación más el cual no es necesario su explicación ya que utilizaremos el modo CTC

Fuente de reloj: Los temporizadores pueden utilizar diferentes fuentes de reloj, una de ellas puede ser el del sistema del MCU (frecuencia principal del microcontrolador) o un reloj externo.
Además, se pueden aplicar preescaladores para reducir la frecuencia de la fuente de reloj ajustando la resolución del temporizador.
En esta solución se utilizará la fuente de reloj del MCU el cual es 16 MHz y le aplicaremos un preescalador.

Registro de control (TCCRn): Cada temporizador tiene un registro de control que se utiliza para configurar el modo de operación, la fuente de reloj y el preescalador.

Registro de comparación (OCRn): Los temporizadores tienen uno o más registros de comparación que se utilizan para establecer el valor de comparación. Cuando el contador del temporizador alcanza el valor, se pueden generar interrupciones o realizar otra acción específica.

Contiene algunas funciones y modos de operación más, pero se centrará en las que se van a utilizar. Concretamente en CTC y contador de eventos.

4.3.1 **TIMER0**

En este proyecto se utilizará el TIMER0 siendo un temporizador/contador de 8-bits el cual su principal uso será para realizar interrupciones periódicas temporizando así el proyecto.

Cada vez que el timer ejecute la rutina de interrupción, sabiendo el periodo de interrupción, podes contar la cantidad de veces que entra para tener la temporización que se requiere

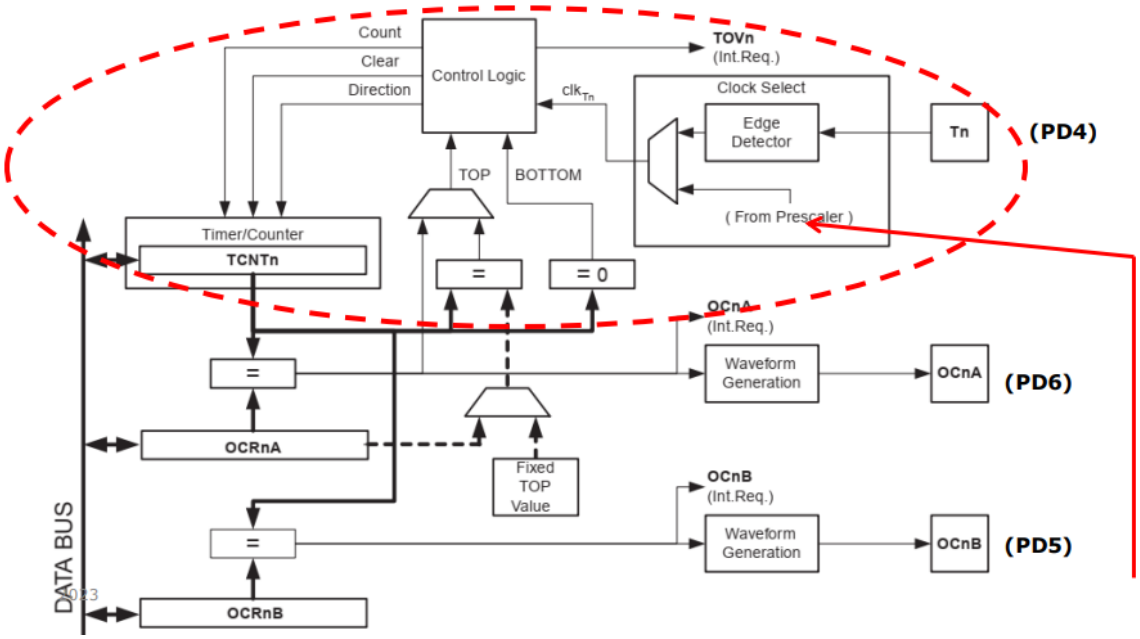


Figura 2.3.1.a Diagrama en bloques

En la figura 2.3.1.a se puede observar cómo se constituye el TIMER0. Como se explicó en el punto anterior, este contiene su registro contador, su comparador y algunas entradas. A la derecha del diagrama, la entrada CLK_{tn} es para poder utilizar una fuente de reloj (en este caso 16MHz) y aplicar un preescalador teniendo así una nueva frecuencia de trabajo.

4.3.2 **Preescalador**

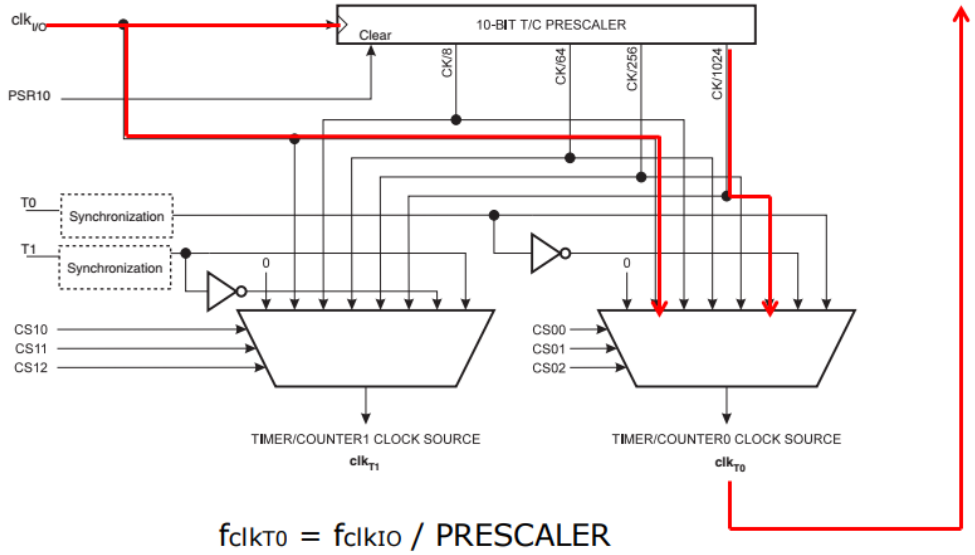


Figura 2.3.2.a Diagrama en bloques Preescalador

A veces es necesario aplicar un preescalador a la fuente de reloj cuando es deseable tener una menor frecuencia de trabajo ya que la original es muy rápida. En este caso se divide la frecuencia en 256 obteniendo así:

$$CLK_{T0}: \frac{16MHz}{256} = 62500 = 62{,}5kHz$$

4.3.3 **Registros de programación:**

Todo esto mencionado anteriormente tiene que ser programado utilizando los registros de programación. Constituido por 7 registros de 8 bits.

4.3.3.a **Registro de configuración:**

Son 2 registros de 8 bits el cual como su nombra lo indica, configura el timer. Aquí configuraremos el timer en modo CTC con la fuente de reloj del MCU aplicando un preescalador de 256.

Registros de configuración

	7	6	5	4	3	2	1	0	
	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
	7	6	5	4	3	2	1	0	
	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
	W	W	R	R	R/W	R/W	R/W	R/W	

Los bits 0 (**WGM00**), 1(**WGM01**) de TCCR0A y bit 3(**WGM02**) del TCCR0B son para aplicar el modo de operación

Concretamente para CTC, **WGM02** = 0, **WGM01** = 1 y **WGM00** = 0

El preescalador se define en los bits 0,1 y 2 de TCCR0B donde:

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Su configuración en este proyecto es de CLKio / 256 es decir **CS02** = 1 y los demás en 0.

4.3.3.b **Registro contador:**

El registro contador funciona de 2 maneras:

Modo normal:

Si el timer está configurado como “Normal” cada vez que se produzca un Overflow, es decir de 0xFF a 0x00, levanta la bandera de Overflow

Siendo TOV0 la bandera de Overflow
Obteniendo una frecuencia de Overflow determinada por:

$$f_{ovf} = \frac{fclk_{tn}}{2^8}$$

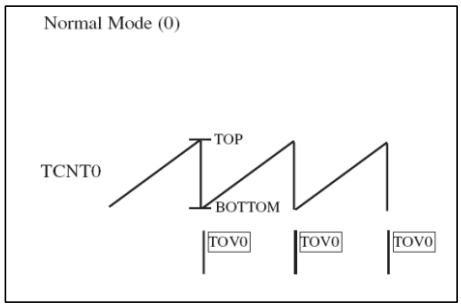
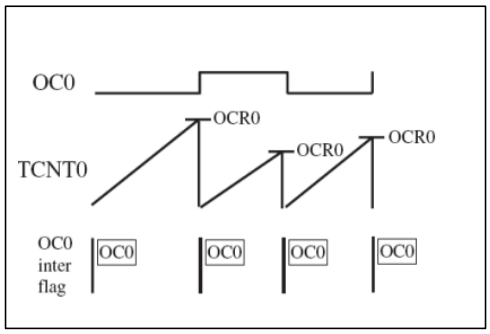


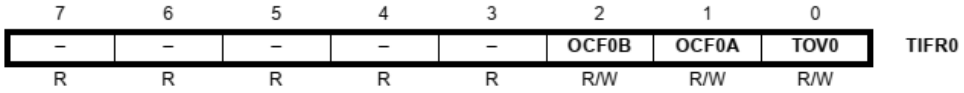
Ilustración 1

Modo CTC:

En este caso, el registro contador no producirá Overflow si no que comparara constantemente el valor del registro contador con otro registro el cual se configura asignándole un valor de 0 a 255. Una vez el contador llegue a ese valor configurado, producirá un levantamiento de un flag (no el de Overflow)



Ilustracion 2



Este es uno de los registros del TIMER y aquí se encuentran los flags, como se menciona esta el flag de Overflow y luego 2 flag para los 2 registros de comparación que contiene.

4.3.4 **Utilización modo CTC:**

Como fue mencionado, en este proyecto se utilizará el modo CTC comparando con un valor definido y una fuente de reloj obteniendo así una frecuencia de trabajo deseada y conocida.

Ya se dio a conocer que trabajaremos con una fuente de reloj provista por el MCU siendo esta 16MHz. Se opto por aplicarle un preescalador y rebajar esta frecuencia ya que los tiempos del proyecto, ya sea de actualización de estados o de flag, son del orden de los milisegundos.

Obteniendo así: $f_{clk} = 16MHz$

$$f_{timer0} = \frac{f_{clk}}{preescaler} = \frac{16MHz}{256} = 62{,}5\ kHz$$

4.3.4.b **Periodo:**

Ya se obtuvo todos los datos para calcular el periodo que se requiere con el cual el timer hará un pedido de interrupción.
Se definió un valor del registro de comparación tal de que sea un periodo cómodo de trabajar. Precisamente el valor 249, teniendo:

$$T_{isr} = \frac{250}{62{,}5kHz} = 4ms$$

Con este resultado, podemos ahora trabajar en base a que la rutina de interrupción del TIMER0 se ejecutara cada 4ms.

Para verificar que verdaderamente se está ejecutando una rutina de interrupción cada 4ms, se utilizara la opción de osciloscopio de Proteus (ver figura-osciloscopio)

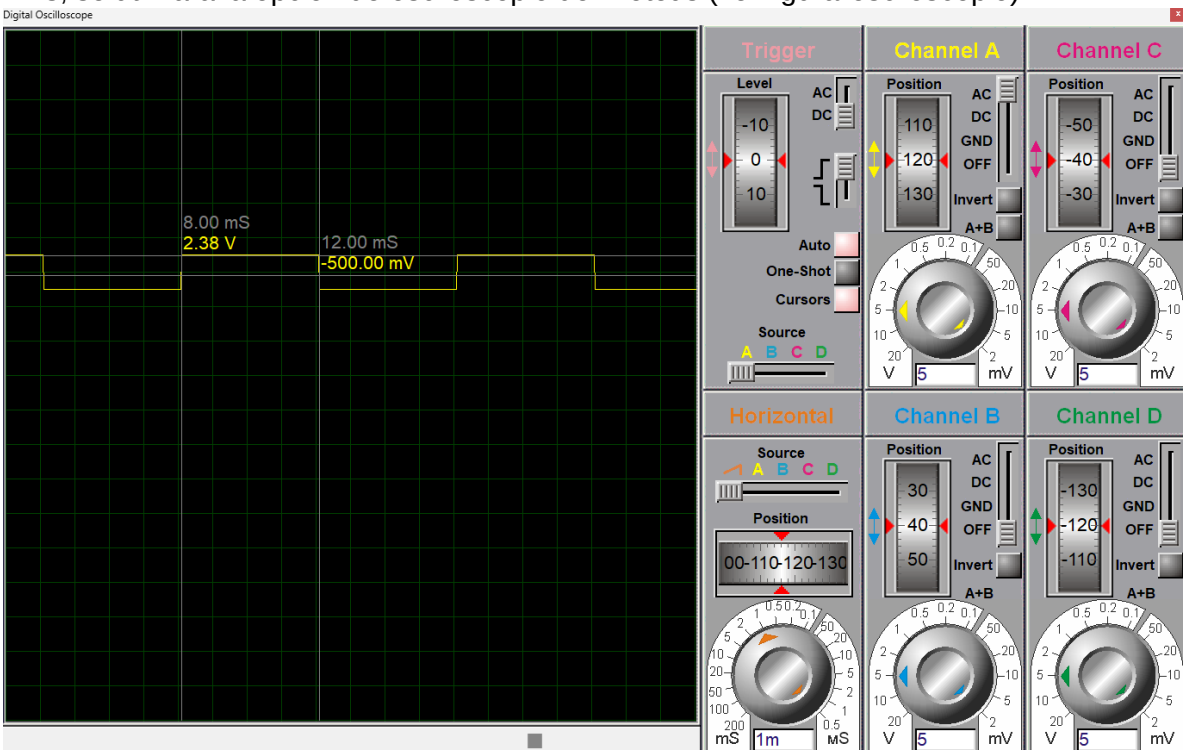


Figura-osciloscopio

Donde se puede confirmar que entre cada flanco de reloj hay 4 ms.

4.3.5 **Rutina de interrupción:**

La rutina de interrupción definida para el TIMER funcionara como un actualizador de flag (ver Pseudocodigo 4.3.5.1).
Para este proyecto tenemos 2 flag a tener en cuenta. Uno de ellos es el flag del reloj, que, cada 1sg debemos actualizarlo. El otro, se identifica como un flag para actualizar la Maquina de Estados Finita.
Por lo tanto, si el flag del reloj se quiere actualizar cada 1sg y la rutina se ejecuta cada 4ms, la cuenta se resuelve trivialmente.
Para la Maquina de Estados Finita se resolvió que un tiempo adecuado de actualización sea de 100ms.

```
Incrementar variable contadora MEF //(incremento cada 4ms)
Incrementar variable contadora RELOJ
Si variable contadora igual a 25 //100 ms
     Pongo en 1 el flag de actualización de la MEF
Si la variable contadora igual a 250 //1000ms = 1 sg
     Pongo en 1 el flag de actualización del reloj
```

Pseudocodigo 4.3.5.1 *ISR (Rutina de interrupción)*

5 **Planificador de tareas:**

5.1 **Introducción:**

Se requiere tener el conocimiento de cuál es la siguiente tarea que se necesita ejecutar en un tiempo determinado. Un planificador (scheduler) es aquel que decide cuál de esas siguientes tareas se ejecuta.

Hay 2 tipos de planificadores, uno de ellos es el no apropiativo que se basa en que una vez que una tarea está ejecutándose, hasta que no se finalice no puede ejecutar otra. Luego el apropiativo el cual puede dejar de ejecutar cierta tarea y comenzar otra.

En este proyecto se usará el no apropiativo temporizado por el TIMER0 descripto anteriormente.

5.2 **Arquitectura Time-Triggered**

Tareas planificadas por una única interrupción periódica denominada RTI (Real Time Interrupt). Donde la RTI es la que nos da la base de tiempo de todo el proyecto.
Por lo tanto, una interrupción nos marcara un tick del sistema permitiendo así planificar las tareas que corresponden ejecutar.

En este caso el scheduler es el encargado de visualizar el estado de los FLAGS y decidir qué es lo siguiente a ejecutar (ver Pseudocódigo 5.2).

```
Si el FLAG del reloj es verdadero
    Llamo a la función para actualizar el reloj
    Asigno falso el FLAG del reloj
Si el FLAG de la actualización de la MEF es verdadero
    Llamo a la función para actualizar la MEF
    Asigno falso el FLAG de la MEF
```

Pseudocodigo 5.2 *sEOS_Dispatch_Tasks()*

Dentro del módulo del despachador como se mencionó, tendremos la inicialización del TIMER0, donde en el capítulo de temporizadores se habló de que valores tendrá los registros de programación

En C, la forma de darle valores a estos registros es de la forma:

```
void TIMER0_Init(){
	DDRD |= (1<<PORTD6); //waveform generator output
	//configuración del TOPE del contador TCNT0
	OCR0A=249;                  //250
	TCCR0A =(1<<COM0A0) | (1<<WGM01); //modo CTC, Toggle on compare match
	TCCR0B |= (1 << CS02); //16Mhz / 256 = 62,5kHz
	TIMSK0 =(1<<OCIE0A); // habilitamos interrupución COMPA
	sei(); //Habilito las interrupciones

}
```

Y la rutina de interrupción de la interrupción periódica es:

```
ISR(TIMER0_COMPA_vect){ //Tisr = 250/62500 -> 4 ms
	static uint8_t count_MEF = 0;
	static uint8_t count_WATCH = 0;



	if(++count_MEF == 25) {  //Si cada 4 ms realiza la rutina de interrupcion y
quiero actualizar el estado de la MEF cada 100 ms -> 100ms/4ms = 25
		FLAG_MEF = 1;
		count_MEF=0;
	}
	if(++count_WATCH == 250 ){  //Si quiero actualizar el reloj cada 1s ->
1000ms/4ms = 250
		FLAG_WATCH = 1;
		count_WATCH=0;
	}

}
```

5.3 **Uso de tareas Multi-Estados**

Al trabajar con una MEF, debemos asegurar que las tareas se ejecuten en un plazo dado, por lo tanto, no se puede utilizar funciones bloqueantes o que hagan un retardo muy significante para la temporización.

6. **Máquina de Estados Finitos.**

Es un modelo abstracto del comportamiento del sistema que se basa en principios simples. Como el nombre indica, está compuesto por un conjunto finito de estados, como así también por entradas, salidas y transiciones.

6.1 **Introducción.**

Estos autómatas finitos describen y representan el comportamiento secuencial de un sistema, permitiendo modelar y diseñar de forma estructurada una secuencia que va a evolucionar en base a determinados eventos.

6.2 **Tipos de MEF.**

En base a cómo se definen las salidas en función de los estados y las transiciones, podemos diferenciar distintas Máquinas de Estados Finitos. La elección entre un modelo u otro depende del problema a resolver, de los requisitos del sistema y del comportamiento deseado.

6.2.1 **Modelo de Mealy.**

En este modelo, la salida depende del estado actual y de las entradas. Esto significa que las salidas pueden cambiar inmediatamente después de una transición. Es decir, la respuesta es inmediata ya que la salida la produce en forma directa después de recibir la entrada.

6.2.2 **Modelo de Moore.**

Aquí la salida depende sólo del estado actual. En este caso, las salidas se determinan sólo por el estado actual del sistema, independientemente de las entradas o transiciones que llevaron a la máquina a ese estado. Esto quiere decir que la respuesta solo depende del estado en que se encontrará la máquina después de realizar cada transición.

6.3 **Modelización:**

Ya que el enunciado había posibilidades a libre interpretación en ciertas cosas sobre el funcionamiento de la cerradura, se decidió varias cosas. Entre ellas es que, una vez se presiona una tecla para introducir la contraseña, el usuario tendrá 3 sg para introducir un nuevo digito, si no, volverá al estado de inicio. Por otro lado, es que si el usuario decide modificar la hora este no tendrá un tiempo determinado, es decir hasta que no ingrese 2 números o cancele la edición, estará en ese estado. Esto no tiene consecuencias sobre el reloj ya que en concurrencia el reloj se está ejecutando, por lo luego de salir de la edición, el reloj se actualizará correctamente. (Ver ilustración)

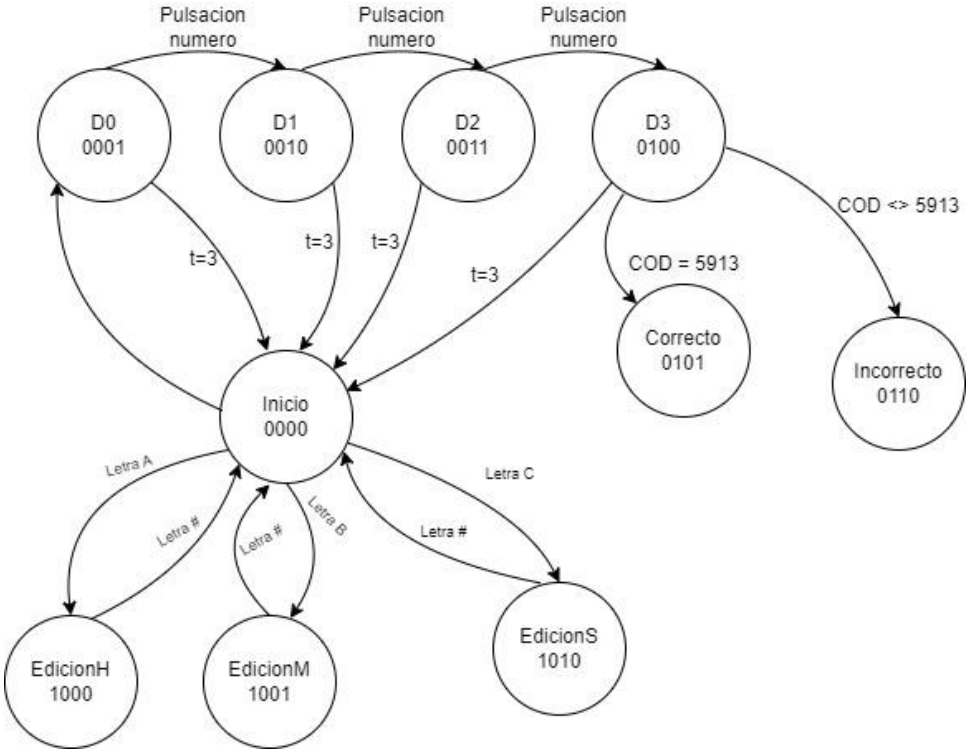


Ilustración MEF

6.4 **MEF en software:**

Definiremos varios estados en el cual en cada iteración llevara la cuenta de cuantas veces se entró al estado y, al ser temporizada, esa cantidad define los tiempos dentro de cada estado.

También debemos definir un estado inicial.

Se define una función dentro del módulo de la MEF como ya se mencionó llamado *MEF_Update()* el cual es llamado por el dispatcher cada 100 ms

Otra función dentro de la MEF es la de *MEF_init()* que define el estado “Inicio” como el inicial, además mandamos al led la estructura de como se muestra el reloj y un mensaje “CERRADO”.

6.4.1 **Implementación:**

Se implementará la transición de estados mediante la estructura switch-case

Inicio: En este estado comienza el autómata finito, aquí se muestra la hora e inicialmente un mensaje para dar visibilidad al usuario de la cerradura electrónica (ver figura-lcd-inicio). La lógica principal está en detectar si se presiona una tecla, y en caso de ser así, saber qué hacer en base a esta entrada. (ver *pseucodigo inicio*)

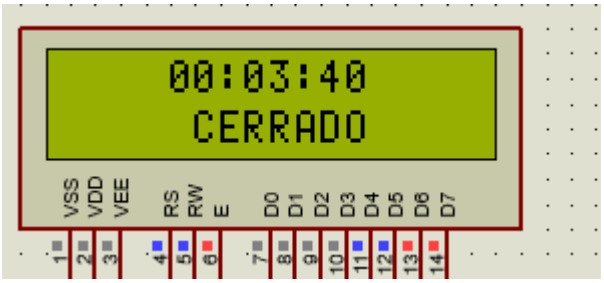


Figura-lcd-inicio

```
Si se presionó una tecla

    Si la tecla presionada es

        ‘A’: se avanza al estado de edición de la hora

        ‘B’: se avanza al estado de edición de los minutos

        ‘C’: se avanza al estado de edición de los segundos

        un número entre 0 y 9: el usuario comenzó a introducir una
contraseña y debo avanzar al siguiente estado de lectura

        cualquier otro caso: no se realiza ninguna acción
```

Pseudocodigo incio

D0, **D1** y **D2**. Estos estados secuenciales unos de otro representan como el usuario va ingresando carácter a carácter la contraseña, avanzando entre los estados a medida que esto ocurre. (ver ilustración 6.4.a) Aquí lo que se hace es colocar el puntero del LCD en la posición correcta, enmascarar las entradas del usuario y avanzar hacia el siguiente estado. Si no se recibe ninguna entrada en un tiempo menor a tres segundos, se retorna al estado inicial. (ver *Pseudocodigo inserción-contraseña*)

```
Ubicar el puntero del LCD en la posición correcta

Enviar un asterisco al display

Si se presionó una nueva tecla, avanzar al siguiente estado

Si pasaron 3 segundos, volver al estado inicial
```

Pseudocodigo inserción-contraseña

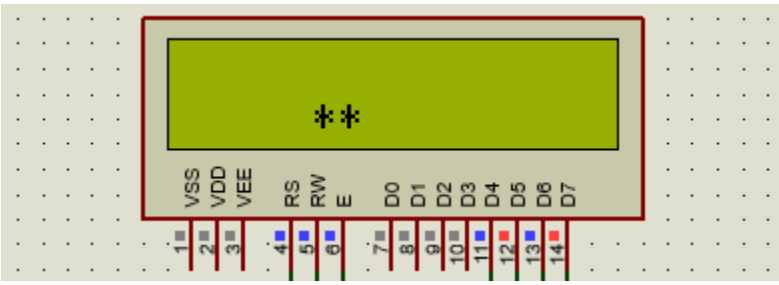


Ilustración 6.4.a

D3. Este estado es una continuación de los estados de ingreso de contraseña, y es el eslabón final en el cual se verifica si la contraseña ingresada es correcta o incorrecta.

```
Si la contraseña es correcta

    Se avanza al estado Correcto

Sino

    Se avanza al estado Incorrecto
```

Estado Correcto:

Luego de haber presionado la cuarta tecla y la combinación de las 4 ser la contraseña establecida (5913) transicionara al estado ‘Correcto’ el cual mostrará en la pantalla “ABIERTO” (ver figura-lcd-correcto) durante 3 segundo (ver Pseudocodigo 6)

Figura-lcd-correcto

```
Incremento variable que indica cantidad de entradas al estado

Si no se estuvo en el estado más de 3 segundos

    Mando al LCD el mensaje “ABIERTO”

Si no

    Voy al estado INICIO
```

Pseudocodigo-6

Estado Incorrecto:

A diferencia del estado Correcto, esta transición de la cuarta tecla presionada consecutiva al estado ‘Incorrecto’ sucede cuando la contraseña es diferente a 5913. Una vez esta la transición, se muestra en el LED “DENEGADO” (ver figura-lcd-incorrecto) durante 2 segundos y vuelvo al estado ‘Inicio’

Figura-lcd-incorrecto

El Pseudocodigo es similar al de ‘Correcto’ con la diferencia de la cantidad de tiempo que se está en ese estado y el mensaje a mostrar.

Estado Edición:

Dentro de este estado se tiene 3 variables, cada una le corresponde a la hora, los minutos y los segundos, la lógica es similar para todas. Este estado es posible luego de apretar la tecla A, correspondiente a la edición de la hora. La tecla B, correspondiente a la edición

de los minutos y la tecla C correspondiente a los segundos. Solo es posible esta transición desde el estado ‘Inicio’ por lo que la pulsación de cualquiera de estas variantes en otro estado, no realiza la transición.

Luego de pulsar una de estas variantes, el reloj se congela para proceder a la edición del numero de 2 dígitos. En el LED se verá el cursor en la posición correspondiente a editar.

Validación:

Se necesita de varias validaciones dentro de la edición ya que cada hora, minuto o segundo tiene un rango total. Es decir, una vez pulsado los 2 números que le corresponden y apretar el botón de confirmación, el numero debe estar dentro del rango. Por ejemplo, dentro de la hora, esta no pude ser mayor a 23.

Por otro lado, es estricto pulsar 2 números, ya que, al apretar 1 solo, este no será tomado como válido.

Por último, si dentro de la edición la tecla presionada es ‘#’ este cancelará la operación y volverá al estado ‘Inicio’ sin realizar ni guardar nada.

```
Si se leyó una tecla
    Si la tecla es la ‘A’
        Calcular el numero
        Si se pulsaron 2 dígitos
            Validar que este dentro del rango
            Si esta dentro del rango
                Setear el numero en las variables correspondientes
             Si no
                Volver al estado ‘Inicio’
     Si es un numero
         Validar que no pueda pulsar más de 2 números
         Guardar el digito correspondiente
     Si la tecla es ‘#’
         No guardar nada
         Volver al estado ‘Inicio’
```

Drive con video sobre el funcionamiento en el MCU otorgado por la catedra

https://drive.google.com/drive/folders/1ePmAN3aFEIUpbK15dNkJGncFVrmKQxMy?usp=share_link