UNIVERSIDAD
NACIONAL
DE LA PLATA

Facultad de Ingeniería - UNLP
Circuitos Digitales y Microcontroladores (E1305 / E0305)
Curso 2022 - Trabajo Práctico Nº2

Grupo 2

Alumnos: Santana Valentin, Guerrero Nicolás

Índice:

1. Reloj funcional y su visualización en el LCD

1.1 Propuesta

Implementar un reloj que tenga formato fecha y hora a partir de una función LCD, que controla el display, un teclado matricial 4x4 y un atmega328p proporcionados por la cátedra.

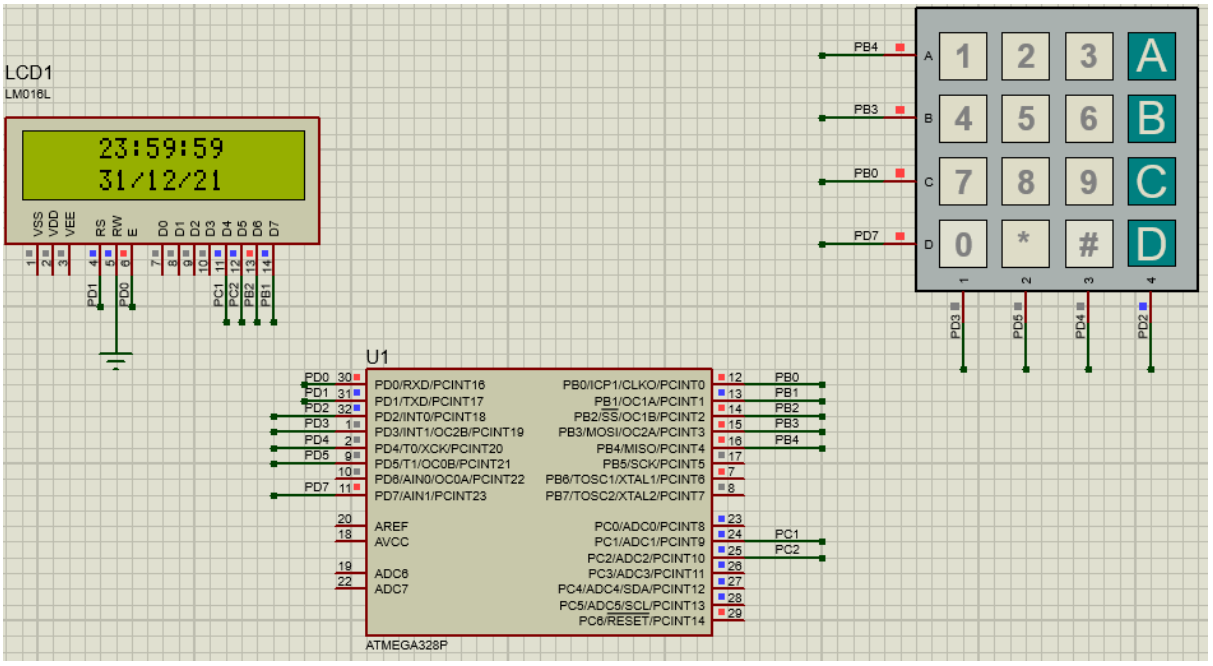

Figura 1.1 Conexión de los componentes en proteus.

1.2 Resolución del problema

Se nos fue facilitada una librería que nos permite interactuar con el display. La misma contenía un archivo cabecera “LCD.h” junto con su respectiva implementación en el archivo “LCD.c”. Las funciones que utilizamos en nuestro programa fueron:

- **LCDinit(void)** → Inicializa el LCD.
- **LCDGotoXY(uint8_t, uint8_t)** → Envía el cursor a la posición x,y del display.
- **LCDSendchar(uint8_t)** → Envía un char al lector del display y muestra en pantalla.
- **LCDescribeDato(int ,unsigned int)** → Escribe un entero en el display, especificando la cantidad de espacios a ocupar.

Lo primero que planteamos fue dividir el problema en varias etapas:

1- Implementar un contador que simule un reloj a partir de un llamado por interrupción.

2- Se creó un procedimiento que permita obtener que tecla se presiono en el keypad

3- Se creó una máquina de estado que simule las diferentes opciones que presentaba el reloj para configurar su fecha y hora.

4- Por último se creó un archivo “SEOS” que se encargará de que todos los archivos anteriormente creados puedan trabajar a la vez de una manera organizada.

2. Creación del Reloj

2.1 Reloj

Lo primero que se hizo fue crear un reloj con su respectivo formato(Figura 2.1). Para ello se utilizó el periférico **timer 0** que cada tantas pulsaciones activará el vector de interrupciones donde se actualiza el contador del reloj segundo por segundo. Lo que se buscó con esto es, que una tarea temporizada se encargue de que cada un segundo se active una interrupción, cuya función es simular el funcionamiento de un reloj, es decir, aumentar segundo a segundo hasta lograr alcanzar la barrera de los 59 segundos, donde incrementa en uno los minutos y reinicia los segundos a 0, de la misma forma con la hora, día, mes y año.

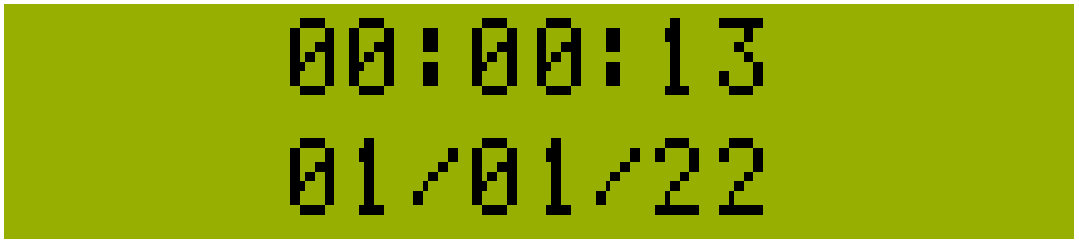

Figura 2.1 Hora y fecha en el display del lcd.

2.2 Timer 0

El **Timer 0** es un periférico proporcionado por el MCU que nos permite crear tareas temporizadas. En nuestro caso usaremos el timer para activar el vector de interrupciones. Para esto se seteó el timer en modo CTC, con un módulo de 156 y un preescaler de 1024. Estos datos se sacaron en base a la siguientes conclusiones:

Primero intentamos averiguar cuál sería el máximo tiempo que el timer podría contar antes de llegar a overflow y activar la interrupción, por lo tanto, elegimos el máximo preescaler posible y utilizamos la siguiente ecuación:

$$(Preescaler/FrecuenciaMCU) * 2^8 = overflow$$

$$(1024/16) * 2^8 = 16384\,ns$$

Como resultado obtuvimos 16384 ns como el máximo tiempo que el timer tardaría en llegar a overflow. Una vez esto, como el timer por su cuenta no puede contar 1 segundo, es decir, 1000000 ns, lo que nos propusimos fue buscar un número que multiplicado por otro nos diera 1 segundo y, que como única condición, estos números estuvieran dentro del límite del timer. Por lo tanto, 10000ns multiplicados por 100 daría como resultado el retardo de un segundo.

Una vez obtenido el tiempo que usaremos en el timer(10000 ns), es necesario encontrar el modulo que de como resultado el overflow del timer en 10000 ns, por lo tanto usamos la siguiente ecuación:

$$ns = OCR0A * 1024/16$$

$$10000\,ns = OCR0A * 1024/16$$

$$156.25 = OCR0A$$

Como el módulo no admite punto flotante es necesario redondear el resultado, por lo tanto, 156 será el módulo a usar. El problema en hacer esto es que el tiempo que tardará el timer en llegar a overflow no será un segundo exacto, sino que habrá un error el cual se puede averiguar de la siguiente forma:

$$error = ns - ((OCR0A.preescaler)/frecuenciaMCU)$$

$$error = 10000 - ((156.1024)/16)$$

$$error = 16\,ns$$

El error por cada interrupción de 10000 ns será de 16 ns.

Una vez conseguido el módulo hay que tener en cuenta que cada 100 interrupciones se va a cumplir un segundo, con un error de 16 ns * 100, es decir, 1600 ns. Por lo tanto, la interrupción número 100 se generará a los 0,9984 segundos.

2.3 Interrupción

La interrupción será la encargada de simular la suma de los segundos en un reloj. Hay que tener en cuenta que el timer llamara a la interrupción cada 10 ms, por lo tanto fue necesario agregar un contador a la interrupción que contara hasta 100 interrupciones antes de sumar un segundo al reloj, por lo tanto el programa quedaría de la siguiente forma:

```
ISR(TIMER0_COMPA_vect)
{
   cuento cada vez que se activa la interrupción
   pregunto si el contador llego a 100
         sumo segundo y pregunto si llego a 60
               seteo segundo a 0
               sumo minuto y pregunto si llego a 60
                     seteo minutos a 0
                     sumo hora y pregunta si llegó a 24
                           seteo hora a 0
                           sumo dia y pregunto si llegó a 32
                                 sumo mes y seteo dia a 1
                           sino pregunto si dia llego a 31
                                 corroboro que el mes actual pertenece a
                                 los meses de 30 días
                                       sumo mes y seteo dia a 1
                           sino pregunto si dia llego a 30
                                 corroboro que el mes actual pertenece a
                                 el mes de 29 días
                                       sumo mes y seteo dia a 1
                           sino pregunto si dia llego a 29
                                 corroboro que el mes actual pertenece a
                                 el mes de 28 días
                                       sumo mes y seteo dia a 1
                           pregunto si mes llegó a 13
                                 seteo mes en 1 y sumo año.
         reinicio contador a 0
   Llamo al planificador  (función explicada más adelante).
}
```

3. Periférico keypad

3.1 Keypad

Se nos pidió llevar a cabo el funcionamiento de un teclado matricial de 4x4 cuyo funcionamiento es el próximo:
Tecla 'A': Comienza la modificación del reloj y la fecha, comenzando por el año y cada vez que sea presionada, pasa por cada uno de los campos de manera secuencial hasta llegar a los segundos. Si se presiona nuevamente en estos últimos, el display se actualizará con los nuevos valores ingresados.
Tecla 'B': Realiza el incremento del valor actual de un campo.
Tecla 'C': Realiza el decremento del valor actual de un campo.
Tecla 'D': Cancela cualquier edición realizada y se vuelve al estado por defecto.

La conexión al atmega328p deberá ser la que se muestra en la *Figura 3.1* y la detección de la presión de alguna de las teclas estará dada cuando una fila(entrada) y una columna(salida) se analicen y posean un bit en 0.

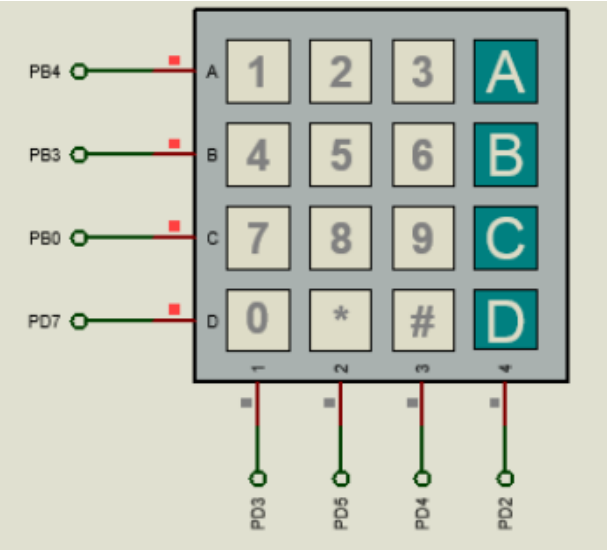

Figura 3.1 Conexión del keypad en proteus.

3.2 Implementación

Para la implementación del teclado matricial realizamos una matriz de char la cual posee todos los caracteres de las teclas que vamos a utilizar y una variable global tecla. Junto a estas existen las siguientes funciones:

• **uint8_t KEYPAD_Scan(uint8_t *)**

Efectúa un sondeo asignando la tecla (en caso de haber sido presionada) a una variable pasada por parámetro y retornando 1. De no ser así se retorna 0.

Esto se logra haciendo un barrido de las columnas cuyo pseudocódigo es el siguiente:

Hago las asignaciones necesarias
Para cada columna
    Seteo el puerto correspondiente para hacer los corrimientos
    Espero 1 milisegundo
    Evaluó si el pin de la fila está en bajo
        Asigno la tecla a la variable pasada por parámetro
        Retorno 1
.
.
Retorno 0

Asignaciones necesarias y lógica de pines:

```
PORTB|= 0x19;  // pines PB4,3,0 en alto
PORTD|= 0x80;  // pin PD7 en alto
DDRD&=~(0x3C); // pines PD2,3,4,5 configurados como entrada
DDRD|=(0X20>>c); // 00100000 Movimiento en columnas

if(!(PINB&(0x10))) //00010000 // si el pin PB4 está en bajo
if(!(PINB&(0x08)))  // si el pin PB3 está en bajo
if(!(PINB&(0x1)))  // si el pin PB0 está en bajo
if(!(PIND&(0x80)))  // si el pin PD7 está en bajo
```

• **uint8_t RetornarTecla(void)**
Retorna la variable global “tecla”

• **void KEYPAD_Actualizar(void)**
Si se presiono la una tecla
    Se la asigno a la variable global “tecla”
Sino le asigno “0” a tecla

4. Máquina de estados finitos(MEF)

4.1 MEF - diagrama

A continuación se expone el diseño del diagrama de estados con sus respectivas relaciones.

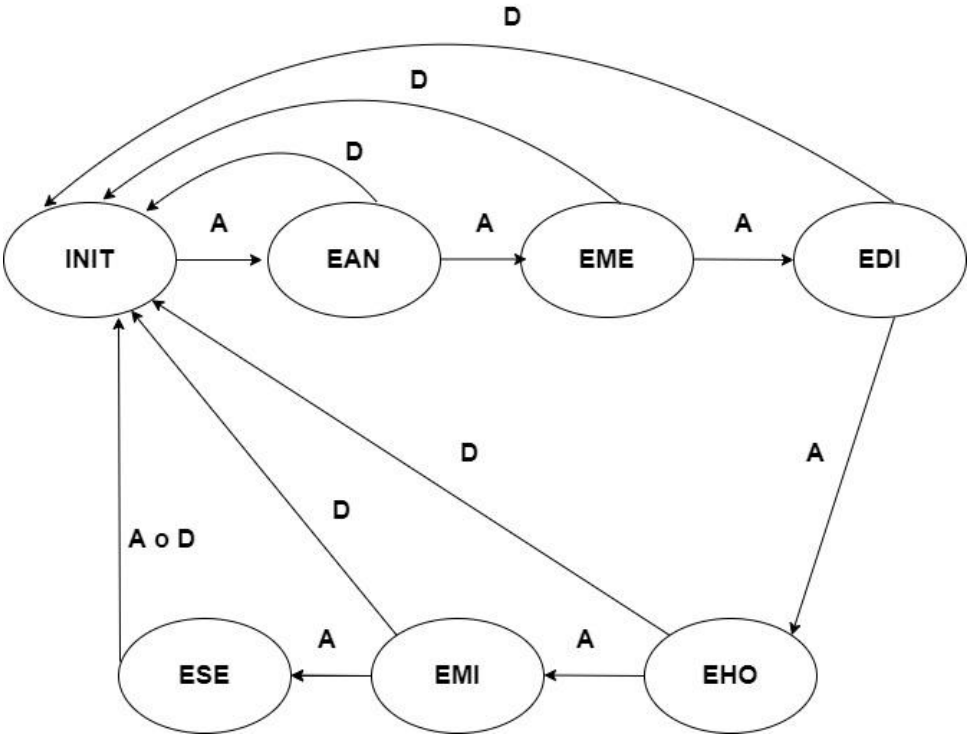

Figura 4.1 Diagrama de estados.

Como se puede ver en el diagrama los cambios de estado están dados al oprimir las teclas ‘A’ o ‘D’. En el caso de ‘A’ se pasa de uno a otro de manera secuencial, pudiendo dar toda la vuelta y llegando nuevamente al estado inicial, mientras que al presionar ‘D’ se vuelve siempre al estado mencionado anteriormente.

4.2 Desarrollo

Se pedía la implementación de una máquina de estados temporizada, y de la misma manera en que resolvimos hasta ahora, la modularizamos y creamos los archivos MEF.c y MEF.h
Dentro de MEF.h se encuentra además de las cabeceras de las funciones, un tipo de dato enum que contiene todos los estados que nuestra máquina va a utilizar, los cuales son:

INIT: Estado inicial.
EAN: Estado en el cual se modifica el campo año.
EME: Estado en el cual se modifica el campo mes.
EDI: Estado en el cual se modifica el campo día.
EHO: Estado en el cual se modifica el campo hora.
EMI: Estado en el cual se modifica el campo minutos.
ESE: Estado en el cual se modifica el campo segundos.

Dentro de MEF.c, se encuentra un vector de funciones que está relacionado con el enum mencionado anteriormente y dependiendo del estado del mismo se ejecutará una función particular.

4.3 Pseudocódigo

El pseudocódigo de las funciones del vector de estados para EAN, EME, EDI, EHO, EMI es el siguiente:

Función de estado

Si se activó la bandera entonces

Aviso que el valor al que hace referencia el estado debe parpadear

Obtengo la tecla presionada

Si toque la A

Cambio de estado

Si toque la B

Incremento el valor correspondiente

Envío el nuevo valor

Si toque la C

Decremento el valor correspondiente

Envío el nuevo valor

Si toque la D

Obtengo el valor del reloj interno

Vuelvo al estado inicial

La única diferencia por la que las funciones de INIT y ESE no están incluidas en el pseudocódigo anterior es porque ambas tienen un comportamiento diferente.

En el caso de INIT:

Si se activó la bandera entonces

Obtengo la tecla presionada

Si toque la A entonces

Obtengo el tiempo al momento de presionar la tecla

Cambio de estado

En el caso de ESE la única diferencia es que si toqué la A, además de cambiar de estado, aplico el nuevo tiempo en caso de pasar las validaciones correspondientes.

4.4 Funciones utilizadas

Las funciones que posee nuestra máquina de estados (además de las del vector de funciones mencionado anteriormente) son:

- **void MEF_Inicializar(void)**
- **void MEF_Actualizar(void)**
- **static void aplicarTiempo(time);**

La función **MEF_Inicializar()** tiene como único objetivo setear el estado actual de la máquina al inicial.

La función **MEF_Actualizar()**, es la que controla prácticamente toda la máquina de estados ya que su función es como lo dice su nombre actualizar la misma. Para esto, mediante la activación de un flag cada cierto tiempo, llama a la función actual del vector de funciones y revisa si una tecla fue presionada, y dependiendo de cual pasará a otro estado o realizará las tareas correspondientes.

Por último **aplicarTiempo()** es una función que valida que los campos ingresados durante el recorrido de todos los estados sean acordes a la realidad y no tengan valores ilógicos. En caso de no ser así los cambios no serán aplicados.

5. Simple Embedded Operating System(sEOS)

Si bien, ya hemos definido los componentes que conforman el sistema del reloj aún queda por resolver quién se encargará de la ejecución de las tareas de forma organizada y temporizada. Esto es necesario ya que pueden surgir problemas a la hora de ejecutar varias tareas a la vez.

El sEOS es un planificador de tareas cooperativo que se encarga de ejecutar tareas que serán disparadas por ticks de reloj. Será el encargado de que cada cierto tiempo se realicen las tareas de teclado, visualización del lcd y MEF. Para esto hemos creado un archivo cabecera sEOS.h donde se encuentran las funciones públicas y un sEOS.c donde se destaca dos funciones entre las demás: el Planificador y el Despachador.

5.1 Planificador

El planificador se encargará de contar los ticks de reloj que se producen cada vez que el timer realiza una interrupción de 10 ms. Este tiene un contador para cada tarea, es decir, existe un contador para teclado, LCD, y la MEF por lo que solo es posible estar haciendo una tarea a la vez. A su vez, existen lo que se denomina un máximo establecido para cada contador, por lo cual una vez el contador llegue a su correspondiente límite, el planificador activará la señal de flag que la recibirá el despachador.

Los contadores y los máximos fueron inicializados de las siguiente manera:

```
#define INIT_KEYPAD 20
#define INIT_MEF 10
#define INIT_LCD 1
#define INIT_BLANCO 100

static int8_t contKeypad=19;
static int8_t contMEF=9;
static int8_t contLCD=0;
static int8_t contBlanco=99;
```

Figura 5.1 Contadores y constantes de máximos.

Como la interrupción se genera cada 10 ms cada contador se deberá multiplicar por esta duración para saber cuánto tiempo tardará en ejecutar la tarea asignada; de esta manera queda definido de la siguiente forma:

- La información del LCD se actualizará cada 10 ms.
- La comprobación del keypad se hará cada 200 ms.
- La MEF se ejecutará cada 100 ms.
- El INIT_BLANCO es la constante que, cuando su contador llegue a 100, activará el flag que permitirá que los datos mostrados en el lcd parpadeen de acuerdo a qué posición esté ubicado el usuario. Esto se hará cada un segundo.

5.2 Despachador

El despachador se encarga de decidir que tarea se realizará. Para lograr esto estará verificando los flags de tarea y cuando alguno esté activo (activado por el planificador) este invocará la tarea a realizar y reiniciará el flag para que el planificador la pueda volver a activar en los siguientes ticks de la interrupción.

5.3 Funciones time

Dentro del archivo sEOS.c existen además tres funciones que tienen relación directa con el display del lcd.
En primer lugar **sEOS_time1()** se utiliza para obtener el tiempo del reloj interno del programa, así en caso de una cancelación en la MEF presionando la 'D', se puede actualizar el lcd con el tiempo correspondiente. Para llevar a cabo esto, cambia de condición un flag.
La función **sEOS_time2()** en cambio, obtiene el valor del tiempo congelado del reloj interno y activa un flag para mostrarlo en el lcd y así modificarlo.
Por último la función **sEOS_timeBlank()** es la encargada, en parte, de generar el efecto de parpadeo. Para esto, mediante un bucle for, verifica cual es la posición que se está editando actualmente y envía a la misma un espacio en blanco que se mostrará en esa ubicación el lcd, manteniendo los demás campos.
Estas últimas dos funciones se alternan en el momento de la modificación, generando el resultado deseado.

6. Inicialización de los componentes

6.1 Archivo main.c

El archivo "main.c" es el encargado de inicializar tanto el timer, lcd, como la máquina de estados(MEF).

Además, es el encargado de inicializar el reloj con la función **sEOS_time1()**, la cual se ocupará de mandar el primer dato a la variable time para que el reloj inicialice en 23:59:59 de 31/12/21.

Una vez inicializado los componentes, el main solo se encargará de estar llamando al despachador quien será el encargado de elegir qué tarea comenzará su ejecución.

```
int main(void)
{
    init();
    LCDinit();
    MEF_Inicializar();
    sEOS_time1(t);
    while(1)
    {
        sEOS_Despachador();
    }
}
```

Figura 6.1 Parte del código de "main.c".

6.2 Función LCD_Actualizar

Quizás sea una de las funciones más importantes del programa, ya que su correcto funcionamiento es necesario para una buena experiencia de usuario. Su propósito es mostrar en el lcd los valores actuales de reloj, ya sea el interno, el que se está modificando o el que posee los espacios en blanco. Si bien esta función, para una mejor disposición debería estar en lcd.c, preferimos dejarla en el main para no mezclarla con la librería previamente administrada por la cátedra.

```
void LCD_Actualizar(time t){

    LCDGotoXY(4,1);
    LCDescribeDato(t.date,2);
    LCDsendChar('/');
    LCDescribeDato(t.month,2);
    LCDsendChar('/');
    LCDescribeDato(t.year,2);
    LCDGotoXY(4,0);
    LCDescribeDato(t.hour,2);
    LCDsendChar(':');
    LCDescribeDato(t.minute,2);
    LCDsendChar(':');
    LCDescribeDato(t.second,2);
}
```

Figura 6.2 Código de la función LCD_Actualizar

Anexo

Archivos cabecera

Main.h

```
#ifndef MAIN_H
#define MAIN_H

#include <avr/io.h>
#include <lcd.h>
#include <stdio.h>
#include "MEF.h"
#include "sEOS.h"
#include "Clock.h"
#include "Keypad.h"

void LCD_Actualizar(time);

#endif
```

Keypad.h

```
#ifndef KEYPAD_H
#define KEYPAD_H

#include <inttypes.h>
#include <avr/io.h>
#define F_CPU 16000000
#include <util/delay.h>

uint8_t KEYPAD_Scan(uint8_t *);
uint8_t RetornarTecla(void);
void KEYPAD_Actualizar(void);

#endif
```

Clock.h

```
#ifndef CLOCK_H
#define CLOCK_H
```

```
#include <inttypes.h>

typedef struct{
      unsigned char second ;
      unsigned char minute;
      unsigned char hour;
      unsigned char date;
      unsigned char month;
      unsigned char year;
}time;

static volatile time t={59,59,23,31,12,21};

void init(void);
char not_leap(void);
time obtenerTime(void);
void setearTime(time);

#endif
```

MEF.h

```
#ifndef _MEF_H
#define _MEF_H

#include "Keypad.h"
#include "sEOS.h"
#include "Clock.h"

typedef enum{INIT,EAN,EME,EDI,EHO,EMI,ESE} enum_estados;
void MEF_Actualizar(void);
void MEF_Inicializar(void);
void fINIT(void);
void fEAN(void);
void fEME(void);
void fEDI(void);
void fEHO(void);
void fEMI(void);
void fESE(void);

#endif
```

sEOS.h

```
#ifndef SEOS_H
#define SEOS_H

#include <inttypes.h>
#include "Clock.h"
#include "main.h"
#include "Keypad.h"
#include "MEF.h"
#include "lcd.h"

void sEOS_Planificador(void);
void sEOS_Despachador(void);
void sEOS_time1(time);
void sEOS_time2(time);
void sEOS_timeBlank(time, uint8_t[]);

#endif
```

lcd.h

```
#ifndef LCD_H
#define LCD_H

#include <inttypes.h>

      //#include "main.h"


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



#define LCD_RS   1     //define MCU pin connected to LCD RS
#define LCD_RW   1     //define MCU pin connected to LCD R/W
#define LCD_E    0     //define MCU pin connected to LCD E
#define LCD_D0   0     //define MCU pin connected to LCD D0
#define LCD_D1   1     //define MCU pin connected to LCD D1
```

```
#define LCD_D2   2     //define MCU pin connected to LCD D1
#define LCD_D3   3     //define MCU pin connected to LCD D2
#define LCD_D4   1     //define MCU pin connected to LCD D3
#define LCD_D5   2     //define MCU pin connected to LCD D4
#define LCD_D6   2     //define MCU pin connected to LCD D5
#define LCD_D7   1     //define MCU pin connected to LCD D6
#define LDP1 PORTB     //define MCU port connected to LCD data pins
#define LDP2 PORTC     //define MCU port connected to LCD data pins
#define LCP PORTD      //define MCU port connected to LCD control
pins
#define LDDR1 DDRB     //define MCU direction register for port
connected to LCD data pins
#define LDDR2 DDRC     //define MCU direction register for port
connected to LCD data pins
#define LCDR DDRD      //define MCU direction register for port
connected to LCD control pins

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

#define LCD_CLR              0    //DB0: clear display
#define LCD_HOME             1    //DB1: return to home position
#define LCD_ENTRY_MODE       2    //DB2: set entry mode
#define LCD_ENTRY_INC        1    //DB1: increment
#define LCD_ENTRY_SHIFT      0    //DB2: shift
#define LCD_ON_CTRL          3    //DB3: turn lcd/cursor on
#define LCD_ON_DISPLAY       2    //DB2: turn display on
#define LCD_ON_CURSOR        1    //DB1: turn cursor on
#define LCD_ON_BLINK         0    //DB0: blinking cursor
#define LCD_MOVE             4    //DB4: move cursor/display
#define LCD_MOVE_DISP        3    //DB3: move display (0-> move
cursor)
#define LCD_MOVE_RIGHT       2    //DB2: move right (0-> left)
#define LCD_FUNCTION         5    //DB5: function set
#define LCD_FUNCTION_8BIT    4    //DB4: set 8BIT mode (0->4BIT
mode)
#define LCD_FUNCTION_2LINES  3    //DB3: two lines (0->one line)
#define LCD_FUNCTION_10DOTS  2    //DB2: 5x10 font (0->5x7 font)
#define LCD_CGRAM            6    //DB6: set CG RAM address
#define LCD_DDRAM            7    //DB7: set DD RAM address
// reading:
```

```
#define LCD_BUSY            7    //DB7: LCD is busy
#define LCD_LINES               2     //visible lines
#define LCD_LINE_LENGTH         16    //line length (in
characters)
// cursor position to DDRAM mapping
#define LCD_LINE0_DDRAMADDR     0x00
#define LCD_LINE1_DDRAMADDR     0x40
#define LCD_LINE2_DDRAMADDR     0x14
#define LCD_LINE3_DDRAMADDR     0x54
// progress bar defines
#define PROGRESSPIXELS_PER_CHAR  6


void LCDescribeDato(int val,unsigned int field_length); // Agrego
Funcion para escribir 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
// displays a horizontal progress bar at the current cursor
location
// <progress> is the value the bargraph should indicate
// <maxprogress> is the value at the end of the bargraph
// <length> is the number of LCD characters that the bargraph
```

```
should cover
//adapted from AVRLIB - displays progress only for 8 bit variables
void LCDprogressBar(uint8_t progress, uint8_t maxprogress, uint8_t
length);
void LCD_Init();
void LCD_Update();



#endif
```

Archivos ".c"

Main.c

```
#include "main.h"

uint8_t key;

int main(void)
{
	init();
	LCDinit();
	MEF_Inicializar();
	sEOS_time1(t);
	while(1)
	{
		sEOS_Despachador();
	}
}


void LCD_Actualizar(time t){

	LCDGotoXY(4,1);
	LCDescribeDato(t.date,2);
	LCDsendChar('/');
	LCDescribeDato(t.month,2);
	LCDsendChar('/');
	LCDescribeDato(t.year,2);
```

```
	LCDGotoXY(4,0);
	LCDescribeDato(t.hour,2);
	LCDsendChar(':');
	LCDescribeDato(t.minute,2);
	LCDsendChar(':');
	LCDescribeDato(t.second,2);
}
```

Keypad.c

```
#include "Keypad.h"

const char keypad[4][4] = {
	{'2','3','1','A'},
	{'5','6','4','B'},
	{'8','9','7','C'},
	{'*','#','0','D'}
};

static uint8_t tecla;

uint8_t KEYPAD_Scan (uint8_t *key){
	uint8_t c;

	PORTB|= 0x19;
	PORTD|= 0x80;

	for (c=0;c<4;c++)
	{
		DDRD&=~(0x3C);
		DDRD|=(0X20>>c);
		_delay_ms(1);
		if(!(PINB&(0x10))){
			*key = keypad[0][c];
			return 1;
		}
		if(!(PINB&(0x08))){
			*key = keypad[1][c];
			return 1;
		}
		if(!(PINB&(0x1))){
			*key = keypad[2][c];
```

```
                return 1;
            }
            if(!(PIND&(0x80))){
                *key = keypad[3][c];
                return 1;
            }

        }
        return 0;
}

void KEYPAD_Actualizar(void){
        uint8_t key=' ';
        if(KEYPAD_Scan(&key)){
            tecla=key;
        }
        else tecla=0;
}

uint8_t RetornarTecla(void){
        return tecla;
}
```

Clock.c

```
#include <avr/interrupt.h>
#include <avr/io.h>
#include "sEOS.h"
#include "Clock.h"

static uint8_t counter = 0;

void init(void)
{
        DDRC = 0xFF;
        TCCR0A = (1 << WGM01);
        OCR0A = 156; //0.01 sec
        TIMSK0 = (1 << OCIE0A);
        TCCR0B = (1 << CS02) | (1 << CS00); // prescaler 1024
        sei();
}
```

```
ISR(TIMER0_COMPA_vect)
{
	counter++;
	if(counter > 100){
		if (++t.second==60)
		{
			t.second=0;
			if (++t.minute==60)
			{
				t.minute=0;
				if (++t.hour==24)
				{
					t.hour=0;
					if (++t.date==32)
					{
						t.month++;
						t.date=1;
					}
					else if (t.date==31)
					{
						if ((t.month==4) || (t.month==6)
|| (t.month==9) || (t.month==11))
						{
							t.month++;
							t.date=1;
						}
					}
					else if (t.date==30)
					{
						if(t.month==2)
						{
							t.month++;
							t.date=1;
						}
					}
					else if (t.date==29)
					{
						if((t.month==2) && (not_leap()))
						{
							t.month++;
```

```
                                    t.date=1;
                              }
                        }
                        if (t.month==13)
                        {
                              t.month=1;
                              t.year++;
                        }
                  }
            }
      }

      counter = 0;
  }
  sEOS_Planificador();
}

char not_leap(void)      //check for leap year
{
  if (!(t.year%100))
  {
      return (char)(t.year%400);
  }
  else
  {
      return (char)(t.year%4);
  }
}

time obtenerTime(void){
  return t;
}

void setearTime(time newTime){
  t=newTime;
}
```

MEF.c

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

```
static void aplicarTiempo(time);

void (*MEF_funciones[])(void) =
{fINIT,fEAN,fEME,fEDI,fEHO,fEMI,fESE};

static enum_estados estado;
static uint8_t flag = 0;
static time ti;
static uint8_t blanco[6]={0};


void MEF_Inicializar(void){
	estado = INIT;
}

void MEF_Actualizar(void){
	if (flag)
		flag=0;
	else
		flag=1;
	(*MEF_funciones[estado])();
}

void fINIT(void)
{
	if(flag==1){
		uint8_t tecla = RetornarTecla();
		if(tecla == 'A'){
			ti=obtenerTime();
			sEOS_time2(ti);
			estado = EAN;
		}
	}
}

void fEAN(void){
	if(flag==1){
		blanco[0]=' ';
		sEOS_timeBlank(ti,blanco);
		uint8_t tecla = RetornarTecla();
		switch(tecla){
			case 'A':
```

```
                        estado = EME;
                        break;
                case 'B':
                        ti.year++;
                        if(ti.year==100){
                                ti.year=0;
                        }
                        sEOS_time2(ti);
                        break;
                case 'C':
                        ti.year--;
                        if(ti.year==255){
                                ti.year=99;
                        }
                        sEOS_time2(ti);
                        break;
                case 'D':
                        ti=obtenerTime();
                        sEOS_time1(ti);
                        estado = INIT;
                        break;
        }
    }
}

void fEME(void){
    if(flag==1){
        blanco[1]=' ';
        sEOS_timeBlank(ti,blanco);
        uint8_t tecla = RetornarTecla();
        switch(tecla){
            case 'A':
            estado = EDI;
            break;
            case 'B':
            ti.month++;
            if(ti.month==13){
                ti.month=1;
            }
            if((ti.month==2)&(ti.date>29)){
                ti.date=1;
            }
```

```
                if( ((ti.month==4) || (ti.month==6) ||
(ti.month==9) || (ti.month==11))){
                     if(ti.date>30){
                          ti.date=1;
                     }
                }
                sEOS_time2(ti);
                break;
                case 'C':
                ti.month--;
                if(ti.month==0){
                     ti.month=12;
                }
                if((ti.month==2)&(ti.date>29)){
                     ti.date=1;
                }
                if( ((ti.month==4) || (ti.month==6) ||
(ti.month==9) || (ti.month==11))){
                     if(ti.date>30){
                          ti.date=1;
                     }
                }
                sEOS_time2(ti);
                break;
                case 'D':
                ti=obtenerTime();
                sEOS_time1(ti);
                estado = INIT;
                break;
           }
      }
}

void fEDI(void){
      if(flag==1){
           blanco[2]=' ';
           sEOS_timeBlank(ti,blanco);
           uint8_t tecla = RetornarTecla();
           switch(tecla){
                case 'A':
                estado = EHO;
                break;
```

```
                case 'B':
                ti.date++;
                if((ti.month==2)&(ti.date==30)){
                      ti.date=1;
                }
                if( ((ti.month==4) || (ti.month==6) ||
(ti.month==9) || (ti.month==11))){
                      if(ti.date==31){
                            ti.date=1;
                      }
                }else if(ti.date==32){
                      ti.date=1;
                }
                sEOS_time2(ti);
                break;
                case 'C':
                ti.date--;
                if((ti.month==2)&(ti.date==0)){
                      ti.date=29;
                }
                      if( ((ti.month==4) || (ti.month==6) ||
(ti.month==9) || (ti.month==11))){
                            if(ti.date==0){
                                  ti.date=30;
                            }
                            }else if(ti.date==0){
                            ti.date=31;
                      }
                sEOS_time2(ti);
                break;
                case 'D':
                ti=obtenerTime();
                sEOS_time1(ti);
                estado = INIT;
                break;
          }
     }
}

void fEHO(void){
     if(flag==1){
          blanco[3]=' ';
```

```
            sEOS_timeBlank(ti,blanco);
            uint8_t tecla = RetornarTecla();
            switch(tecla){
                  case 'A':
                  estado = EMI;
                  break;
                  case 'B':
                  ti.hour++;
                  if(ti.hour==24){
                        ti.hour=0;
                  }
                  sEOS_time2(ti);
                  break;
                  case 'C':
                  ti.hour--;
                  if(ti.hour==255){
                        ti.hour=23;
                  }
                  sEOS_time2(ti);
                  break;
                  case 'D':
                  ti=obtenerTime();
                  sEOS_time1(ti);
                  estado = INIT;
                  break;
            }
      }
}

void fEMI(void){
      if(flag==1){
            blanco[4]=' ';
            sEOS_timeBlank(ti,blanco);
            uint8_t tecla = RetornarTecla();
            switch(tecla){
                  case 'A':
                  estado = ESE;
                  break;
                  case 'B':
                  ti.minute++;
                  if(ti.minute==60){
                        ti.minute=0;
```

```
                }
                sEOS_time2(ti);
                break;
                case 'C':
                ti.minute--;
                if(ti.minute==255){
                      ti.minute=59;
                }
                sEOS_time2(ti);
                break;
                case 'D':
                ti=obtenerTime();
                sEOS_time1(ti);
                estado = INIT;
                break;
          }
     }
}

void fESE(void){
     if(flag==1){
          blanco[5]=' ';
          sEOS_timeBlank(ti,blanco);
          uint8_t tecla = RetornarTecla();
          switch(tecla){
                case 'A':
                aplicarTiempo(ti);
                sEOS_time1(ti);
                estado = INIT;
                break;
                case 'B':
                ti.second++;
                if(ti.second==60){
                      ti.second=0;
                }
                sEOS_time2(ti);
                break;
                case 'C':
                ti.second--;
                if(ti.second==255){
                      ti.second=59;
                }
```

```
                    sEOS_time2(ti);
                    break;
                    case 'D':
                    ti=obtenerTime();
                    sEOS_time1(ti);
                    estado = INIT;
                    break;
              }
        }
}



static void aplicarTiempo(time newTime){
      if(newTime.hour<25){
            if(newTime.minute<60){
                  if(newTime.second<60){
                        if(newTime.month<13){
                              if(newTime.date<32){
                                    if ((newTime.month==4) ||
(newTime.month==6) || (newTime.month==9) || (newTime.month==11)){
                                          if(newTime.date<31)
setearTime(newTime);
                                    }else if(newTime.month==2){
                                          if(not_leap()){
                                                if(newTime.date<29)
setearTime(newTime);
                                          }else if(newTime.date<30)
setearTime(newTime);
                                    }else setearTime(newTime);
                              }
                        }
                  }
            }
      }
}
```

sEOS.c

```
#include "sEOS.h"

#define INIT_KEYPAD 20
```

```
#define INIT_MEF 10
#define INIT_LCD 1
#define INIT_BLANCO 100

static int8_t contKeypad=19;
static int8_t contMEF=9;
static int8_t contLCD=0;
static int8_t contBlanco=99;

static uint8_t FlagKeypad=0;
static uint8_t FlagMEF=0;
static uint8_t FlagLCD=0;
static uint8_t FlagBlanco=0;

static time tiempo1,tiempo2,tiempoBlanco;
static uint8_t cond=1;

void sEOS_Planificador(void){
	if(++contMEF==INIT_MEF){
		FlagMEF=1;
		contMEF=0;
	}
	if(++contLCD==INIT_LCD){
		FlagLCD=1;
		contLCD=0;
	}
	if(++contKeypad==INIT_KEYPAD){
		FlagKeypad=1;
		contKeypad=0;
	}
	if(++contBlanco==INIT_BLANCO){
		FlagBlanco=1;
		contBlanco=0;
	}
}

void sEOS_Despachador(void){
	if(FlagKeypad==1){
		KEYPAD_Actualizar();
		FlagKeypad=0;
	}
	if(FlagMEF==1){
```

```
            MEF_Actualizar();
            FlagMEF=0;
      }
      if((FlagLCD==1)&(cond==1)){
            LCD_Actualizar(tiempo1);
            FlagLCD=0;
            tiempo1=obtenerTime();
      }else if((FlagLCD==1)&(cond==0)){
            LCD_Actualizar(tiempo2);
            FlagLCD=0;
            cond = 2;
      }
      if((FlagBlanco==1)&(cond==2)){
            LCD_Actualizar(tiempoBlanco);
            FlagBlanco=0;
            cond = 0;
      }
}



void sEOS_time1(time t){
      tiempo1=t;
      cond=1;
}

void sEOS_time2(time t){
      tiempo2=t;
      cond=0;
}

void sEOS_timeBlank(time t, uint8_t b[]){
      for (int i = 0; i<6;i++)
      {
            if((i==0) &(b[i]==' ')){
                  LCDGotoXY(10,1);
                  LCDsendChar(b[i]);
                  LCDsendChar(b[i]);
                  b[i]='0';
            }else if((i==1) &(b[i]==' ')){
                        LCDGotoXY(7,1);
                        LCDsendChar(b[i]);
```

```
                        LCDsendChar(b[i]);
                        b[i]='0';
                    }else if((i==2) &(b[i]==' ')){
                                LCDGotoXY(4,1);
                                LCDsendChar(b[i]);
                                LCDsendChar(b[i]);
                                b[i]='0';
                        }else if((i==3) &(b[i]==' ')){
                                LCDGotoXY(4,0);
                                LCDsendChar(b[i]);
                                LCDsendChar(b[i]);
                                b[i]='0';
                                }else if((i==4) &(b[i]==' ')){
                                      LCDGotoXY(7,0);
                                      LCDsendChar(b[i]);
                                      LCDsendChar(b[i]);
                                      b[i]='0';
                                }else if((i==5) &(b[i]==' ')){
                                            LCDGotoXY(10,0);
                                            LCDsendChar(b[i]);
                                            LCDsendChar(b[i]);
                                            b[i]='0';
                                            }
       }
      tiempoBlanco=t;
}
```

lcd.c

```
#include "lcd.h"

#include <inttypes.h>
#define F_CPU 16000000
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <util/delay.h>


extern volatile unsigned int temp;

const uint8_t LcdCustomChar[] PROGMEM=//define 8 custom LCD chars
{
```

```
      0x00, 0x1F, 0x00, 0x00, 0x00, 0x00, 0x1F, 0x00, // 0. 0/5
full progress block
      0x00, 0x1F, 0x10, 0x10, 0x10, 0x10, 0x1F, 0x00, // 1. 1/5
full progress block
      0x00, 0x1F, 0x18, 0x18, 0x18, 0x18, 0x1F, 0x00, // 2. 2/5
full progress block
      0x00, 0x1F, 0x1C, 0x1C, 0x1C, 0x1C, 0x1F, 0x00, // 3. 3/5
full progress block
      0x00, 0x1F, 0x1E, 0x1E, 0x1E, 0x1E, 0x1F, 0x00, // 4. 4/5
full progress block
      0x00, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x00, // 5. 5/5
full progress block
      0x03, 0x07, 0x0F, 0x1F, 0x0F, 0x07, 0x03, 0x00, // 6. rewind
arrow
      0x18, 0x1C, 0x1E, 0x1F, 0x1E, 0x1C, 0x18, 0x00  // 7.
fast-forward arrow
};


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

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

```
      LDP=ch;
      LCP|=1<<LCD_RS;
      LCP|=1<<LCD_E;
      _delay_ms(1);
      LCP&=~(1<<LCD_E);
      LCP&=~(1<<LCD_RS);
      _delay_ms(1);
#endif
}
void LCDsendCommand(uint8_t cmd) //Sends Command to LCD
{
#ifdef LCD_4bit
      //4 bit part
      //LDP=(cmd&0b11110000);
      LCD_DATAWR(cmd&0b11110000);
      LCP|=1<<LCD_E;
      _delay_ms(1);
      LCP&=~(1<<LCD_E);
      _delay_ms(1);
      //LDP=((cmd&0b00001111)<<4);
      LCD_DATAWR((cmd&0b00001111)<<4);
      LCP|=1<<LCD_E;
      _delay_ms(1);
      LCP&=~(1<<LCD_E);
      _delay_ms(1);
#else
      //8 bit part
      LDP=cmd;
      LCP|=1<<LCD_E;
      _delay_ms(1);
      LCP&=~(1<<LCD_E);
      _delay_ms(1);
#endif
}
void LCDinit(void)//Initializes LCD
{
#ifdef LCD_4bit
      //4 bit part
      _delay_ms(15);
      //LDP=0x00;
      LCD_DATAWR(0x00);
      LCP=0x00;
```

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


#else
    //8 bit part
```

```
   _delay_ms(15);
   LDP=0x00;
   LCP=0x00;
   LDDR|=1<<LCD_D7|1<<LCD_D6|1<<LCD_D5|1<<LCD_D4|1<<LCD_D3
              |1<<LCD_D2|1<<LCD_D1|1<<LCD_D0;
   LCDR|=1<<LCD_E|1<<LCD_RW|1<<LCD_RS;
//---------one------
   LDP=0<<LCD_D7|0<<LCD_D6|1<<LCD_D5|1<<LCD_D4|0<<LCD_D3
              |0<<LCD_D2|0<<LCD_D1|0<<LCD_D0; //8 it mode
   LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS;
   _delay_ms(1);
   LCP&=~(1<<LCD_E);
   _delay_ms(1);
   //-----------two-----------
   LDP=0<<LCD_D7|0<<LCD_D6|1<<LCD_D5|1<<LCD_D4|0<<LCD_D3
              |0<<LCD_D2|0<<LCD_D1|0<<LCD_D0; //8 it mode
   LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS;
   _delay_ms(1);
   LCP&=~(1<<LCD_E);
   _delay_ms(1);
   //-------three-------------
   LDP=0<<LCD_D7|0<<LCD_D6|1<<LCD_D5|1<<LCD_D4|0<<LCD_D3
              |0<<LCD_D2|0<<LCD_D1|0<<LCD_D0; //8 it mode
   LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS;
   _delay_ms(1);
   LCP&=~(1<<LCD_E);
   _delay_ms(1);
   //--------8 bit dual line----------
   LDP=0<<LCD_D7|0<<LCD_D6|1<<LCD_D5|1<<LCD_D4|1<<LCD_D3
              |0<<LCD_D2|0<<LCD_D1|0<<LCD_D0; //8 it mode
   LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS;
   _delay_ms(1);
   LCP&=~(1<<LCD_E);
   _delay_ms(1);
//-----increment address, invisible cursor shift------
   LDP=0<<LCD_D7|0<<LCD_D6|0<<LCD_D5|0<<LCD_D4|1<<LCD_D3
              |1<<LCD_D2|0<<LCD_D1|0<<LCD_D0; //8 it mode
   LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS;
   _delay_ms(1);
   LCP&=~(1<<LCD_E);
   _delay_ms(5);
        //init custom chars
```

```
	uint8_t ch=0, chn=0;
	while(ch<64)
	{
		LCDdefinechar((LcdCustomChar+ch),chn++);
		ch=ch+8;
	}

#endif
}
void LCDclr(void)					//Clears LCD
{
	LCDsendCommand(1<<LCD_CLR);
}
void LCDhome(void)				//LCD cursor home
{
	LCDsendCommand(1<<LCD_HOME);
}
void LCDstring(uint8_t* data, uint8_t nBytes)	//Outputs string
to LCD
{
register uint8_t i;

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

	// print data
	for(i=0; i<nBytes; i++)
	{
		LCDsendChar(data[i]);
	}
}
void LCDGotoXY(uint8_t x, uint8_t y)  //Cursor to X Y position
{
	register uint8_t DDRAMAddr;
	// remap lines into proper order
	switch(y)
	{
	case 0: DDRAMAddr = LCD_LINE0_DDRAMADDR+x; break;
	case 1: DDRAMAddr = LCD_LINE1_DDRAMADDR+x; break;
	case 2: DDRAMAddr = LCD_LINE2_DDRAMADDR+x; break;
	case 3: DDRAMAddr = LCD_LINE3_DDRAMADDR+x; break;
	default: DDRAMAddr = LCD_LINE0_DDRAMADDR+x;
```

```
	}
	// set data address
	LCDsendCommand(1<<LCD_DDRAM | DDRAMAddr);

}
//Copies string from flash memory to LCD at x y position
//const uint8_t welcomeln1[] PROGMEM="AVR LCD DEMO\0";
//CopyStringtoLCD(welcomeln1, 3, 1);
void CopyStringtoLCD(const uint8_t *FlashLoc, uint8_t x, uint8_t
y)
{
	uint8_t i;
	LCDGotoXY(x,y);
	for(i=0;(uint8_t)pgm_read_byte(&FlashLoc[i]);i++)
	{
		LCDsendChar((uint8_t)pgm_read_byte(&FlashLoc[i]));
	}
}
//defines char symbol in CGRAM
/*
const uint8_t backslash[] PROGMEM=
{
0b00000000,//back slash
0b00010000,
0b00001000,
0b00000100,
0b00000010,
0b00000001,
0b00000000,
0b00000000
};
LCDdefinechar(backslash,0);
*/
void LCDdefinechar(const uint8_t *pc,uint8_t char_code){
	uint8_t a, pcc;
	uint16_t i;
	a=(char_code<<3)|0x40;
	for (i=0; i<8; i++){
		pcc=pgm_read_byte(&pc[i]);
		LCDsendCommand(a++);
		LCDsendChar(pcc);
		}
```

```
}

void LCDshiftLeft(uint8_t n)     //Scrol n of characters Right
{
      for (uint8_t i=0;i<n;i++)
      {
            LCDsendCommand(0x1E);
      }
}
void LCDshiftRight(uint8_t n)    //Scrol n of characters Left
{
      for (uint8_t i=0;i<n;i++)
      {
            LCDsendCommand(0x18);
      }
}
void LCDcursorOn(void) //displays LCD cursor
{
      LCDsendCommand(0x0E);
}
void LCDcursorOnBlink(void)//displays LCD blinking cursor
{
      LCDsendCommand(0x0F);
}
void LCDcursorOFF(void)     //turns OFF cursor
{
      LCDsendCommand(0x0C);
}
void LCDblank(void)         //blanks LCD
{
      LCDsendCommand(0x08);
}
void LCDvisible(void)       //Shows LCD
{
      LCDsendCommand(0x0C);
}
void LCDcursorLeft(uint8_t n)    //Moves cursor by n poisitions
left
{
      for (uint8_t i=0;i<n;i++)
      {
            LCDsendCommand(0x10);
```

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


//**********   Inicio de Comando Agregado por Fido

void LCDescribeDato(int val,unsigned int field_length)
{

/*************************************************************
	This function writes a integer type value to LCD module

	Arguments:
	1)int val  : Value to print

	2)unsigned int field_length :total length of field in which
the value is printed
	must be between 1-5 if it is -1 the field length is no of
digits in the val


*************************************************************/

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

```
      else
            j=5-field_length;

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

//**********   Final de Comando Agregado por Fido


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

      // draw a progress bar displaying (progress / maxprogress)
      // starting from the current cursor position
      // with a total length of "length" characters
      // ***note, LCD chars 0-5 must be programmed as the bar
characters
      // char 0 = empty ... char 5 = full

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

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

```
                // this is a partial or empty block
                if( ((i*(uint16_t)PROGRESSPIXELS_PER_CHAR)) >
pixelprogress )
                {
                      // this is an empty block
                      // use space character?
                      c = 0;
                }
                else
                {
                      // this is a partial block
                      c = pixelprogress % PROGRESSPIXELS_PER_CHAR;
                }
          }
          else
          {
                // this is a full block
                c = 5;
          }

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