Taller de Proyecto 1

Trabajo Práctico N°4

Alumnos:
Pedro Molina Prozzi
Tomás Pi Puig
Evaristo Compagnucci

Propuesta

Se desea realizar un prototipo de sistema embebido con el objetivo de automatizar la tarea de subir y bajar las cortinas, más específicamente cortinas Blackout. El sistema permitirá programar el izado y arriado de la cortina de modo automático, es decir, cambiará el estado de la cortina dependiendo de la iluminación.

Para el modo automático se propone utilizar un sensor LDR, este será conectado en un circuito de tipo divisor de tensión para calcular la intensidad de la iluminación dependiendo de la variación de la corriente en la resistencia, para esto se utilizará el ADC incorporado en el MCU STM32F103C6. Se agregara un LED en paralelo que servirá de indicador, ya que, variará la intensidad de la luz emitida dependiendo de la luz que esté recibiendo el LDR. Este modo permitirá mantener un nivel de iluminación (x%) en la habitación, dentro de la medida de lo posible. Para la modificación del porcentaje de luz deseado, se dispondrá de una conexión serie (interfaz UART) con una PC u otro dispositivo, que funcionará como interfaz de usuario. El tipo y opciones del menú quedan a definir en una etapa de desarrollo más avanzada.

También se implementará un LCD (1602A) que mostrará la intensidad de la iluminación recibida por el LDR y el horario actual, para esto se utilizaran GPIOs del puerto B.
Para el control del motor que se encargará de mover la cortina se utilizara un motor paso a paso (modelo a definir) en conjunto con el controlador L298N (motor y driver específicos a definir). El motor estará alimentado por una fuente de alimentación externa.

Para el diseño del circuito PCB se fabricará siguiendo los siguientes requerimientos:
Simple faz de tamaño máximo: 20x20cm
Pads o vías: 1.8 mm de corona, 0.7mm los agujeros
Ancho de pista y separación: 1 mm.
Sin serigrafía de componentes y sin máscara antisoldante.

Aclaraciones: en la implementación final se utilizó un DC motor en lugar de un motor paso a paso. Tampoco se colocó el led en paralelo al LDR porque decidimos que no aporta funcionalidad. Tampoco se muestra la intensidad de la iluminación en el LDR ya que este tipo de sensor no es el óptimo para medir luxes y además esta unidad de medida no es conocida para el usuario común, por lo tanto no le vimos utilidad, es su lugar se muestra la configuración actual del sistema.

1.Interpretación

En este trabajo práctico desarrollaremos un sistema el cual nos servirá para controlar el pliegue y repliegue de un black out dependiendo de la iluminación deseada en la habitación.Es importante aclarar que nuestro planteo permitirá al usuario definir una iluminación que se mantendrá en un rango determinado, específicamente definimos 3 rangos de iluminación, esto con la intención de que el mínimo cambio de luz no obligue al motor a moverse constantemente.Para el desarrollo de dicho proyecto, utilizaremos los siguientes componentes:

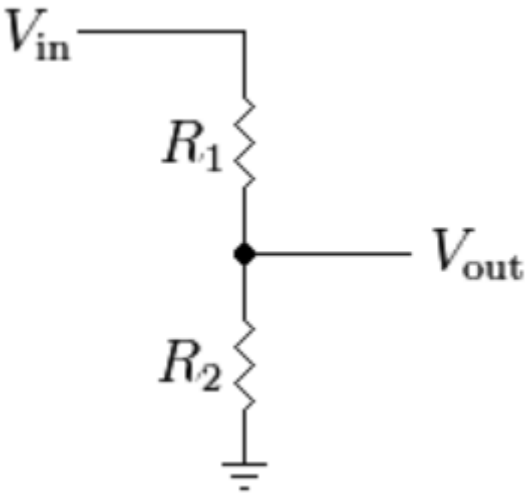
- Un microcontrolador STM32F103C6
- Una pantalla LCD de dos líneas de 16 caracteres c/u
- Un motor (DC motor, 12V con caja reductora, 200 rmp)
- Modulo controlador L298N H-bridge
- LDR (GL5528)

Programaremos el microcontrolador utilizando lenguaje C y el compilador de Keil y para la simulación del sistema utilizaremos Proteus.
En el desarrollo, crearemos una solución modular, encapsulando así, la funcionalidad de cada componente para simplificar la comunicación entre los dispositivos.

Propuesta e implementación

Medición de luz

Para la medición de la luz dentro de la habitación se utilizó una resistencia LDR (GL5528), conectamos el fotoresistor en configuración divisor de tensión con una resistencia de 10kΩ como se muestra en la siguiente imagen:



siendo R1 el LDR.
Esta configuración nos permite utilizar la tensión entre las resistencias, que variara dependiendo del valor del fotoresistor según la siguiente expresión:

$$V_{out} = \frac{R_2}{R_1 + R_2} \cdot V_{in}$$

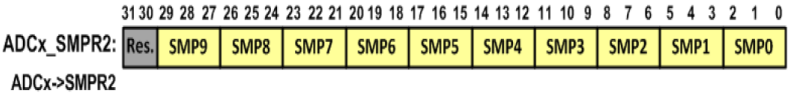
Para medir dicha tensión haremos uso del ADC del microcontrolador utilizando la siguiente configuración:
Para configurar el ADC (ADC1) primero se debe activar el clock del adc, se configura el pin PA1 como entrada analógica, es decir, poner un 0x0 en el pin 1 del registro GPIOA->CRL.

MODEx bits	Direction	Max speed
00	Input	

Input (MODE=00)

CNFx bits	Configuration	Description
00	Analog mode	Select this mode when you use a pin as an ADC input.
01	Floating input	In this mode, the pin is high-impedance.
10	Input with pull-up/pull-down	The value of ODR chooses if the pull-up or pull-down resistor is enabled. (1: pull-up, 0:pull-down)
11	reserved	

Luego se coloca en 1 el bit ADON del registro ADC1->CR2 para prender el ADC y después se selecciona la frecuencia con la que vamos a muestrear, en nuestro caso elegimos la máxima frecuencia, por lo tanto, ponemos un 001 en SMP1 del registro ADC1->SMPR2 .



- **SMPn:** Nth Channel Sample time selection

SMP	Sample time	SMP	Sample time
000	1.5 ADC clock cycles	100	41.5 ADC clock cycles
001	7.5 ADC clock cycles	101	55.5 ADC clock cycles
010	13.5 ADC clock cycles	110	71.5 ADC clock cycles
011	28.5 ADC clock cycles	111	239.5 ADC clock cycles

Con la configuración explicada ya podemos leer valores utilizando el ADC.
Para una mayor comprensión de los valores leídos por el ADC estos se convertirán a valores de tensión (Volts), con la siguiente expresión:

$$Vin = ADC \frac{Vref}{ADC\ step}$$

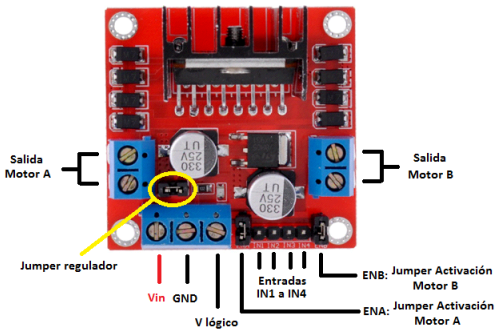
$$Vin = ADC \frac{3.3V}{(2^{12}-1)}$$

Finalmente podremos utilizar la tensión medida para saber si debemos permitir que entre más o menos luz del exterior modificando el estado de la persiana.

Control de persiana

Para el izado y arriado de la cortina se hará uso de un motor de corriente continua, y para controlar el mismo se utilizará el módulo controlador de motores L298N H-bridge, este nos permite controlar y alimentar el motor de una manera sencilla y utilizando pocos pines.

L298N



El L298N puede trabajar con tensiones hasta 35V y una intensidad de hasta 2A por canal por canal, con lo que puede manejar hasta 4A en total y unos 25W.
Este módulo nos permite controlar la dirección de giro del motor.

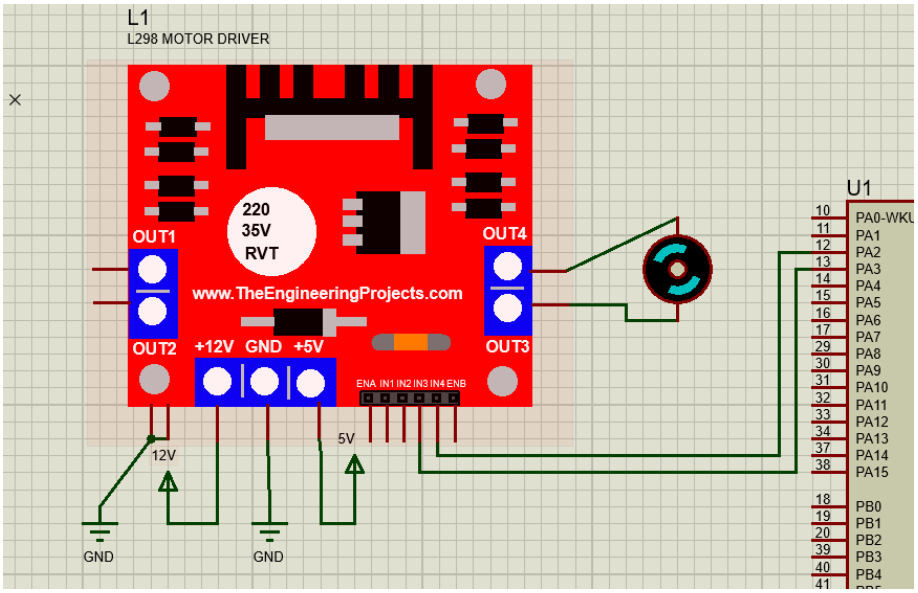
La entrada de tensión Vin admite tensiones entre 3V y 35V, y justo a su derecha en la imagen tenemos el pin que debemos conectar a GND.
La tercera conexión de ese grupo V lógico puede funcionar de dos maneras:

- Si el **jumper del regulador** está **cerrado** activaremos el regulador de tensión del **L298N**, y en **V lógico** tendremos una salida de 5V, que podremos usar para lo que queramos, por ejemplo para alimentar una placa Arduino.
- Si el **quitamos** el **jumper** desactivaremos el regulador, necesitaremos alimentar la parte lógica del módulo, así que tendremos que meter una tensión de 5V por la conexión **V lógico** para que el módulo funcione.
- Si introducimos corriente por V lógico con el jumper de regulación puesto podríamos dañar el módulo.
- Además el **regulador** sólo funciona con tensiones hasta **12V** en **Vin**, por encima de este valor tendremos que quitar el jumper y alimentar la parte lógica del módulo desde otra fuente.

Los pines IN1 e IN2 nos sirven para controlar el sentido de giro del motor A, y los pines IN3 e IN4 el del motor B. Funcionan de forma que si IN1 está a HIGH e IN2 a LOW, el motor A gira en un sentido, y si está IN1 a LOW e IN2 a HIGH lo hace en el otro. Y lo mismo con los pines IN3 e IN4 y el motor B.

Para controlar la velocidad de giro de los motores tenemos que quitar los jumpers y usar los pines ENA y ENB. Si tenemos los jumpers colocados, los motores girarán a máxima velocidad.

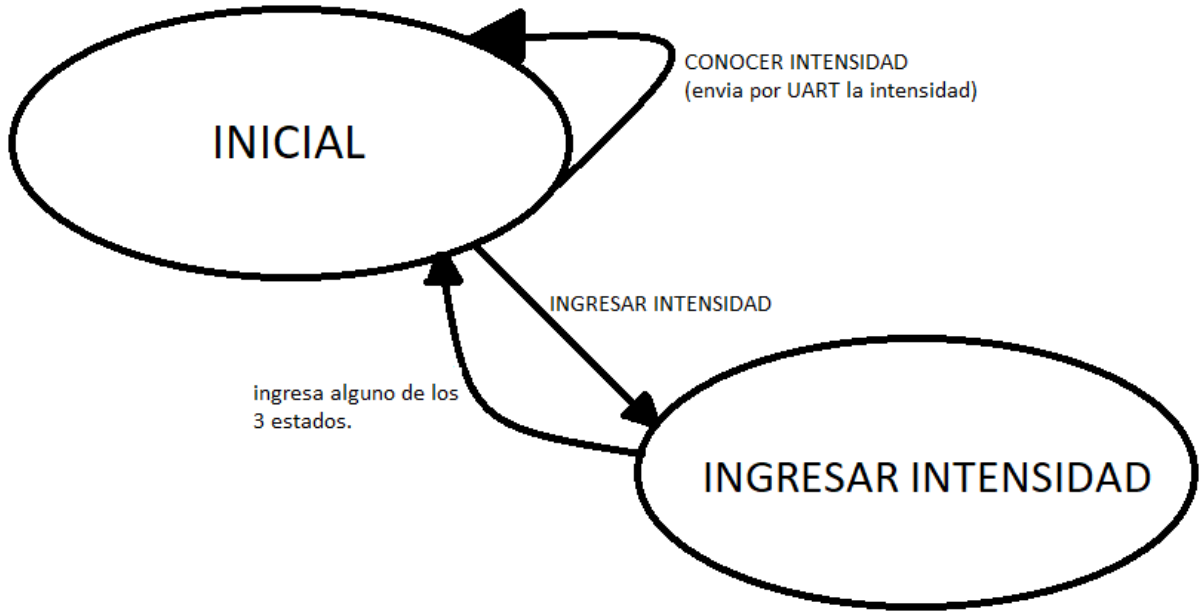
En nuestra implementación los motores girarán a máxima velocidad (12V, 200rpm).
Y quedará conectado como se muestra en la siguiente imagen.



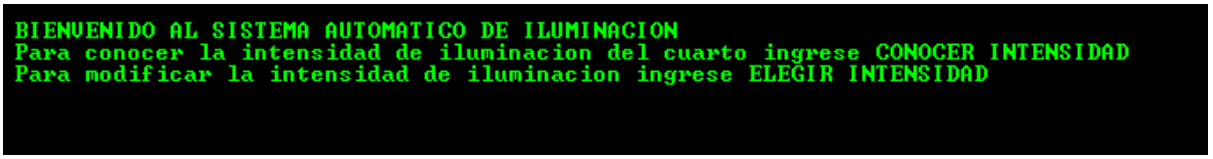
Controlando el motor con los pines PA2 y PA3 (configurados como push-pull output) para modificar el sentido de giro.
Y se detendrá o movera dependiendo de las directivas que de la mef implementada para la lógica del sistema.

Máquina de Estados

para el desarrollo del sistema decidimos utilizar una máquina de estados muy sencilla, en ella contaremos con dos estados el inicial y uno para cambiar la iluminación deseada, también se debe aclarar que la mef además se encarga de actualizar y mostrar la hora, el estado de iluminación que se esté manteniendo y de revisar que la iluminación sea acorde con la seleccionada, y si esto no sucede de accionar el motor.



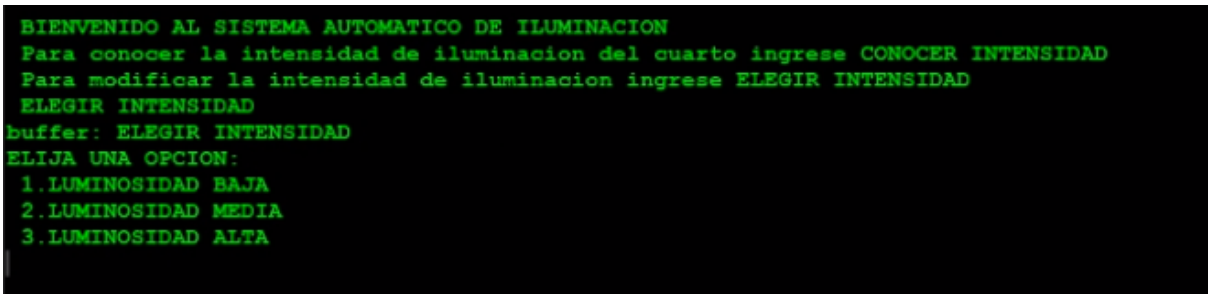
La comunicación del usuario con el sistema se realiza a través de la librería dedicada a la UART. Al iniciar el sistema este enviará un mensaje de bienvenida y explica los dos comandos que se pueden ingresar para interactuar con el sistema.



Aquí podemos ver que los dos comandos disponibles son CONOCER INTENSIDAD y ELEGIR INTENSIDAD.

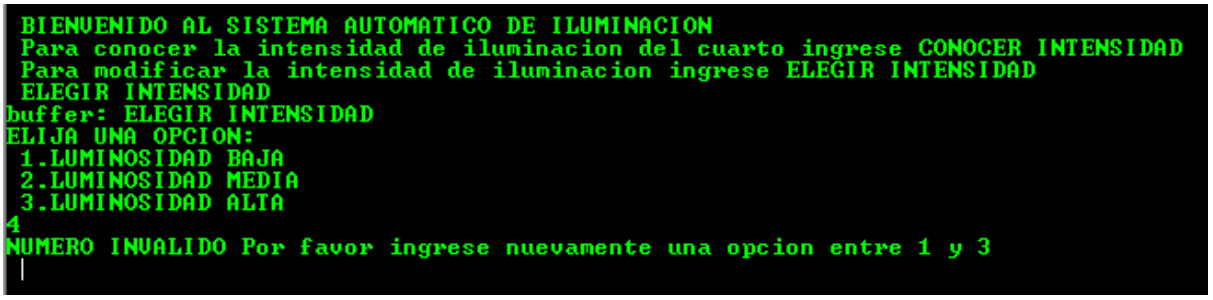
Al ingresar el mensaje CONOCER INTENSIDAD se imprimirá el mensaje correspondiente y se volverá al estado inicial.

al ingresar el mensaje INGRESAR INTENSIDAD pasará al estado ingresar intensidad y se pedirá que elija alguno de los 3 estados posibles



En este caso al seleccionar una de las tres opciones (esto se realiza enviando 1, 2 o 3 mediante la UART), el programa colocará el estado de la cortina y dependiendo de si la luz es la correcta o no se moverá el motor.

Si se ingresa algún valor no válido se mostrará un mensaje indicando y pidiendo que lo vuelva intentar hasta que se ingrese un valor válido.



La mef tiene una temporización de 100 ms, esto se realiza gracias a la librería del timer y al llegar al tiempo estipulado se activa un flag que es leído en el main y este permite actualizar la mef.

```
flag_mef = timer_getFlag();
if (flag_mef){
  MEF_Update();
  timer_resetFlag();
  }
```

Luego para actualizar el tiempo cada un segundo, dentro de la mef hay un contador que cuenta las entradas a la función de MEF Update hasta llegar a 10.

```
if (++cantTiempo == 10){ //actualizo cada 1s
   actualizarTiempo();
   prepararHora();
   cantTiempo =0;
}
```

Para controlar que la iluminación se mantenga a la definida por el usuario cada vez que se entra a la mef update se revisa que la iluminación sea la correcta, si no es así se activa el motor. Por otro lado, si el estado es adecuado con la iluminación se detiene el motor. Las iluminaciones van por un rango de valores para evitar que una leve variación de la luz genere un movimiento del black out.

Para la definición de los rangos utilizamos como referencia la siguiente tabla:

Iluminancia	Abr.	Ejemplo
0,000008 lux	8 µlx	Luz de la estrella Sirio (Vista desde la tierra)
0,0001 lux	100 µlx	Cielo nocturno nublado, luna nueva
0,001 lux	1 mlx	Cielo nocturno despejado, luna nueva
0,01 lux	10 mlx	Cielo nocturno despejado, cuarto creciente o menguante
0,25 lux	250 mlx	Luna llena en una noche despejada[1]
1 lux	1 lx	Luna llena a gran altitud en latitudes tropicales[2]
3 lux	3 lx	Límite oscuro del crepúsculo bajo un cielo despejado[3]
100 lux	1 hlx	Pasillo en una zona de paso
300 lux	3 hlx	Sala de reuniones
500 lux	5 hlx	Oficina bien iluminada
600 lux	6 hlx	Salida o puesta de sol en un día despejado.
1000 lux	1 klx	Iluminación habitual en un estudio de televisión
32.000 lux	32 klx	Luz solar en un día medio (mín.)
100.000 lux	100 klx	Luz solar en un día medio (máx.)

a partir de esta, definimos:

- luminosidad baja: ~0 lux - ~40 lux (0mV- 2154 mV)
- luminosidad media: ~60 lux - ~150 lux (2400 mV - 2817 mV)
- luminosidad alta: ~205 lux - sin límite (2918 mV - 9999 mV)

Los rangos de tensión fueron definidos observando cuál era la tensión medida por el adc para dichos rangos en luxes.

UART

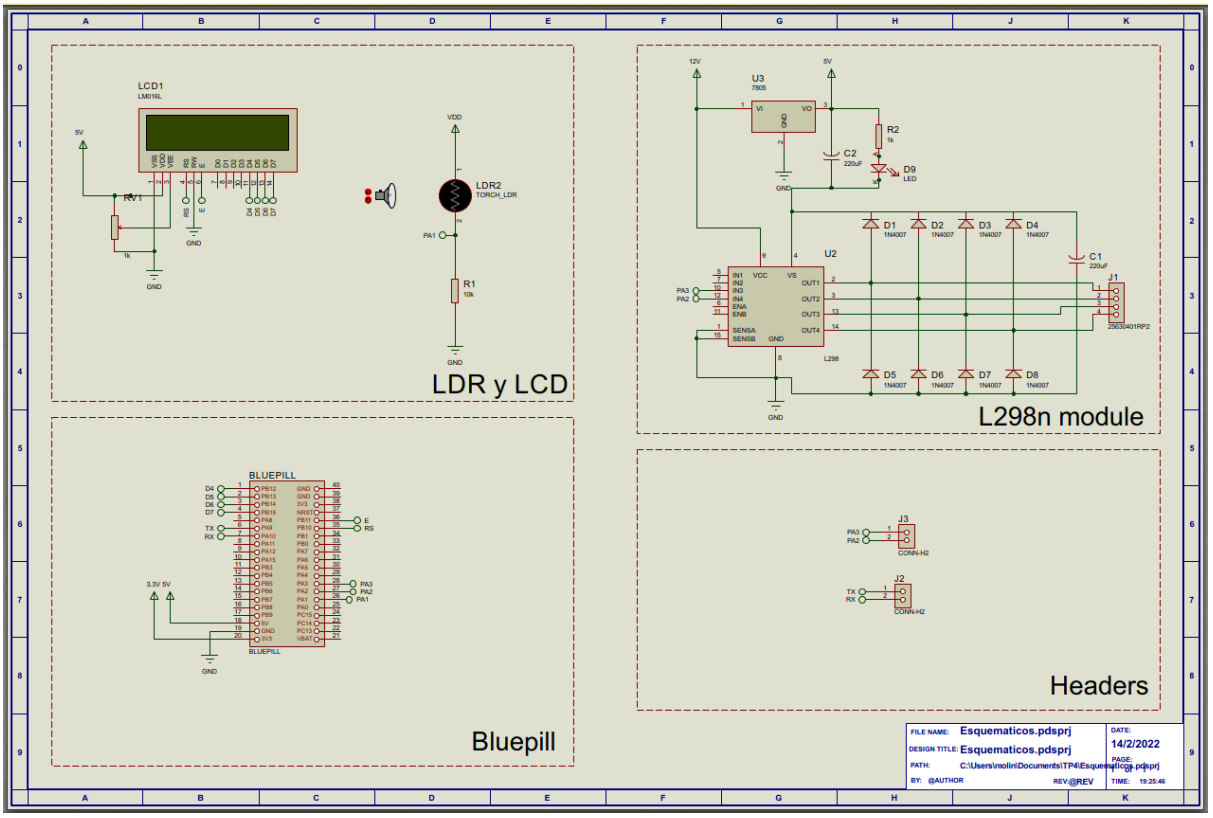
Para la UART creamos una librería que se encargue de generar una interrupción cuando se recibe un carácter. Esta guarda en un buffer los caracteres hasta que llegue un enter. Cuando llega esto activa un flag que es con el que se comunica con la mef para avisar que llego un nuevo mensaje.

```
void USART1_IRQHandler() { /* USART1 interrupt routine */
char c = USART1->DR; /* get received data */
 if (c != '\n'){
     if (c == '\r'){  //SI ES CARACTER DE FIN DE CADENA
            if (getTXindice_escritura() >0){
               usart1_Write_Char_To_Buffer('\0');
               } else {
                     usart1_Write_Char_To_Buffer('\r');
                     usart1_Write_Char_To_Buffer('\0');
           }
            setLlegoMensaje(1); //llegoMensaje=1;
            setTXindice_escritura(0);
            } else {
                usart1_Write_Char_To_Buffer(c); //AGREGO AL BUFFER
           }
  }
}
```

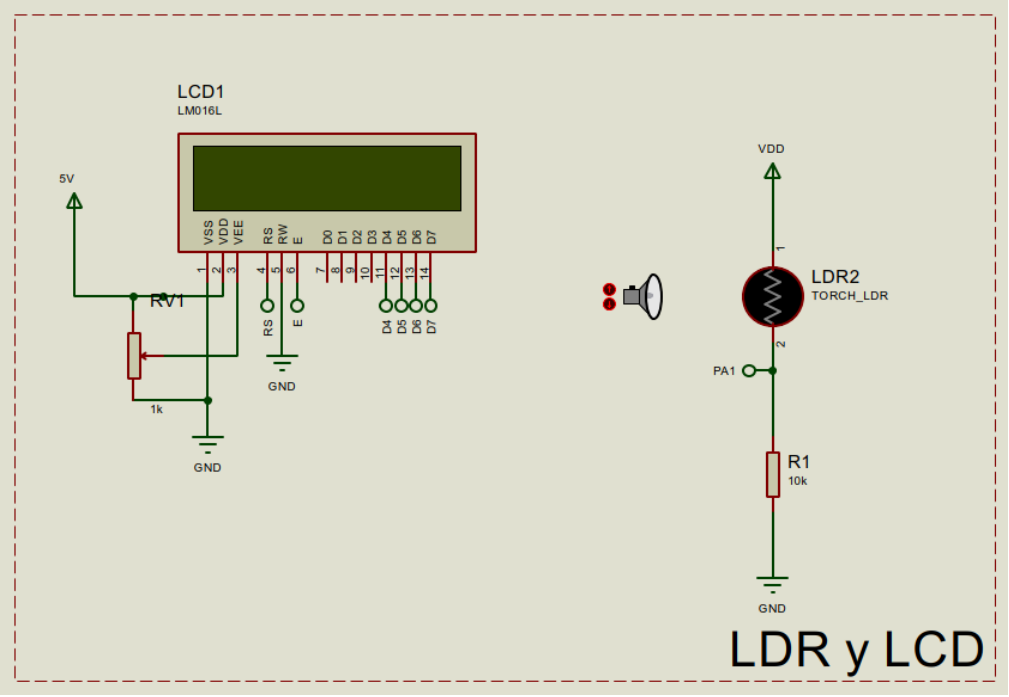
En cambio, para enviar un mensaje se realiza por medio de polling.

```
void usart1_sendByte(unsigned char c){

      USART1->DR=c;
   while((USART1->SR&(1<<6))==0); //wait until the TC flag is set
    USART1->SR &= ~(1<<6);

}
void usart1_sendStr(char* str){
   while(*str!=0){ //while char act is not '\0'
      usart1_sendByte(*str); //print it
      str++; //go to next char
   }
}
```

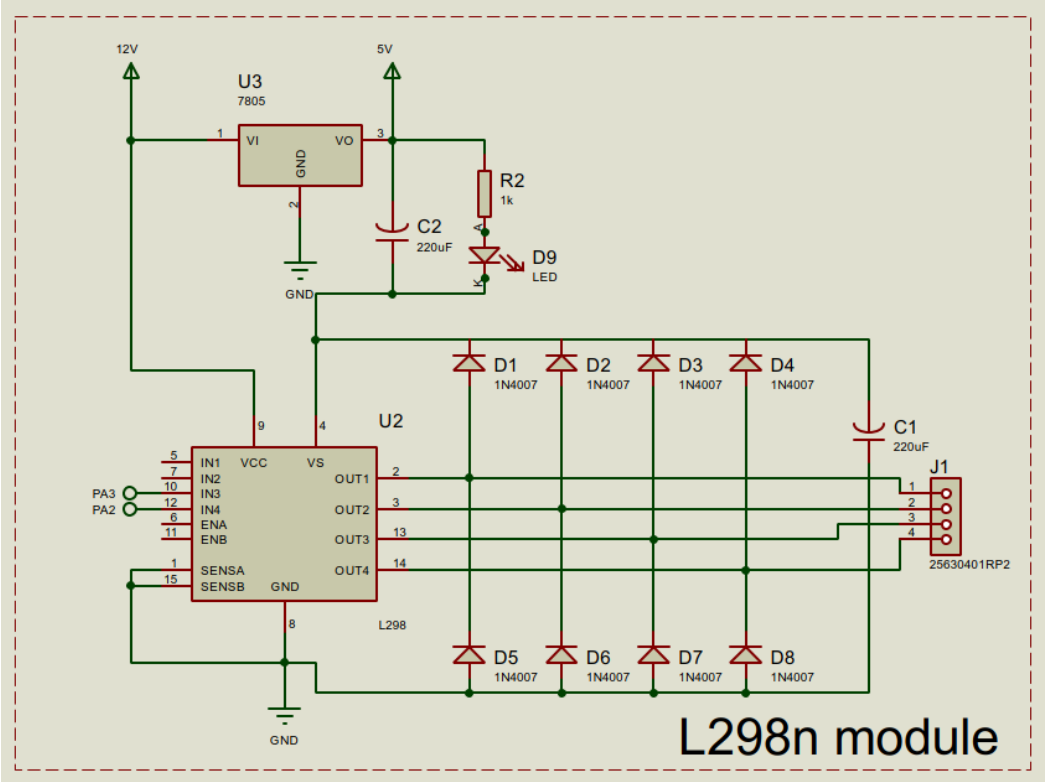
ESQUEMÁTICO



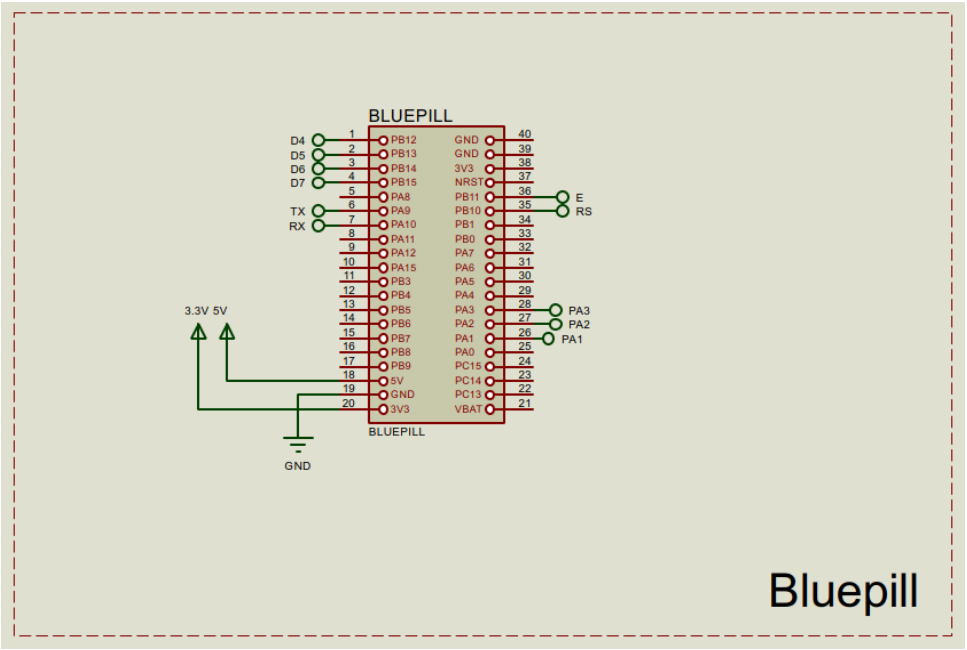
A continuación, se muestra el conexionado del LCD en configuración 4bits, y el divisor resistivo que contiene el fotoresistor utilizado en el proyecto, este divisor está conectado a nuestro ADC, el cual nos permite hacer las lecturas necesarias.



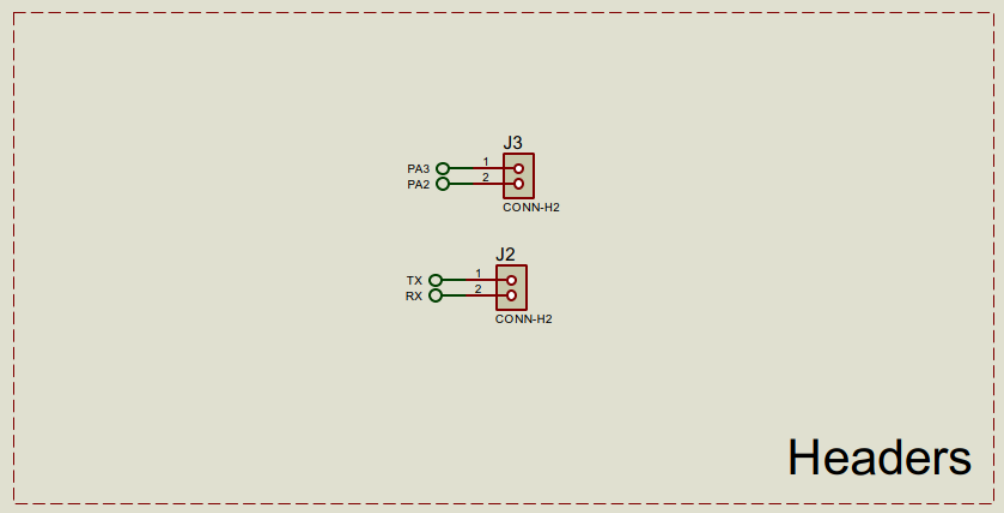
A continuación se muestra el esquemático de nuestro módulo controlador de motores, L298n.



La imagen mostrada a continuación representa los conectores que se utilizaran para conectar la bluepill a nuestro PCB.



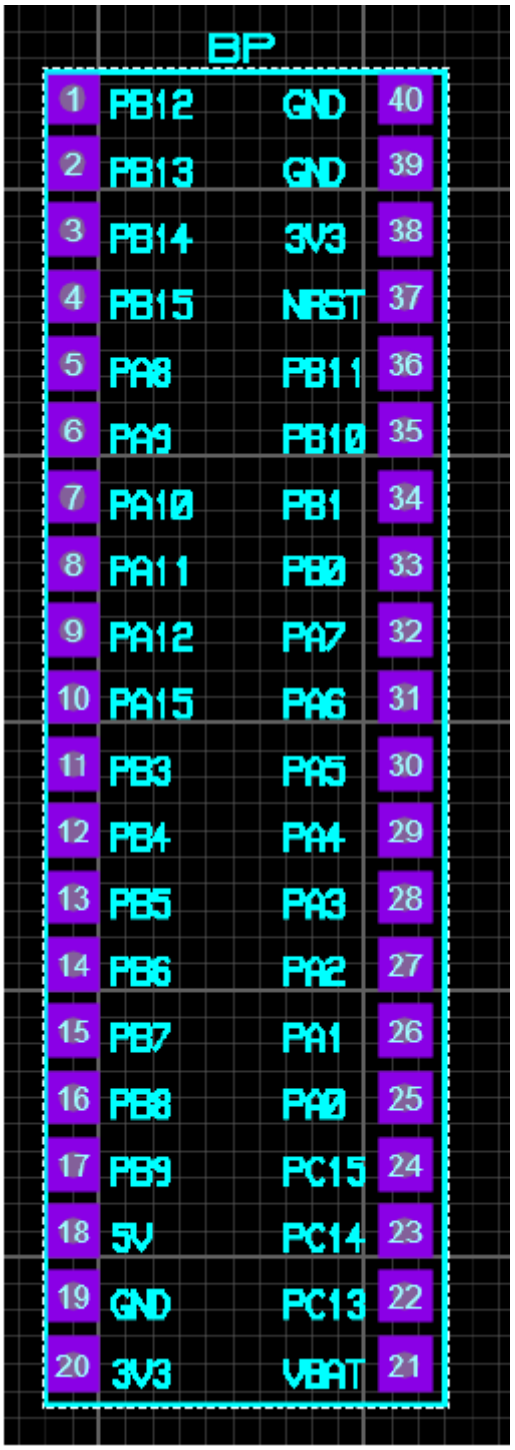
Los conectores mostrados a continuación son los que se utilizaran para la comunicación serie con USART y para la conexión entre el micro y el módulo L298n.



El DC motor no se agregó en el esquemático, va conectado a OUT3 Y OUT4.

PCB

Para el diseño PCB utilizamos los paquetes creados en el anterior trabajo práctico por lo que para la BLUEPILL se utilizó el siguiente diseño:



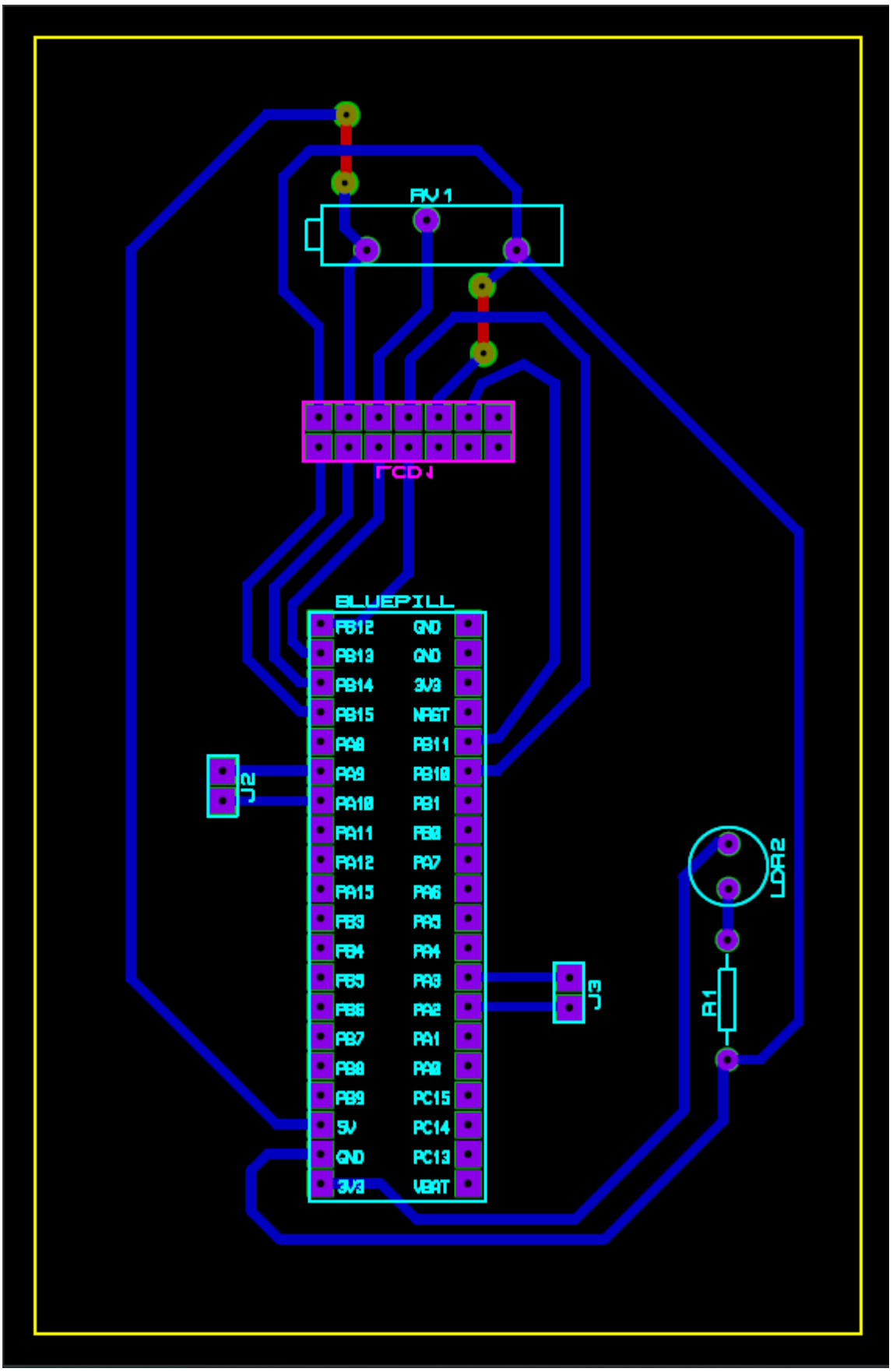
Para lo demás se utilizaron diseños PCB ya incluidos en los componentes por defecto.

Si bien la plaqueta simple faz posee un tamaño máximo de 20cm x 20cm al ubicar eficientemente los componentes se utilizó un tamaño de 7cm x 11cm.

Con respecto al esquemático cabe aclarar que para el diseño PCB no se incluye la plaqueta L298N ya que el motor no se ubicaría en la plaqueta sino en la cortina, es por esto que tomamos la decisión de agregar un conector de dos pines (J3) que es donde irá conectado este. Es posible ver el módulo en el esquemático, pero en el PCB se verá solo el conector de dos pines.

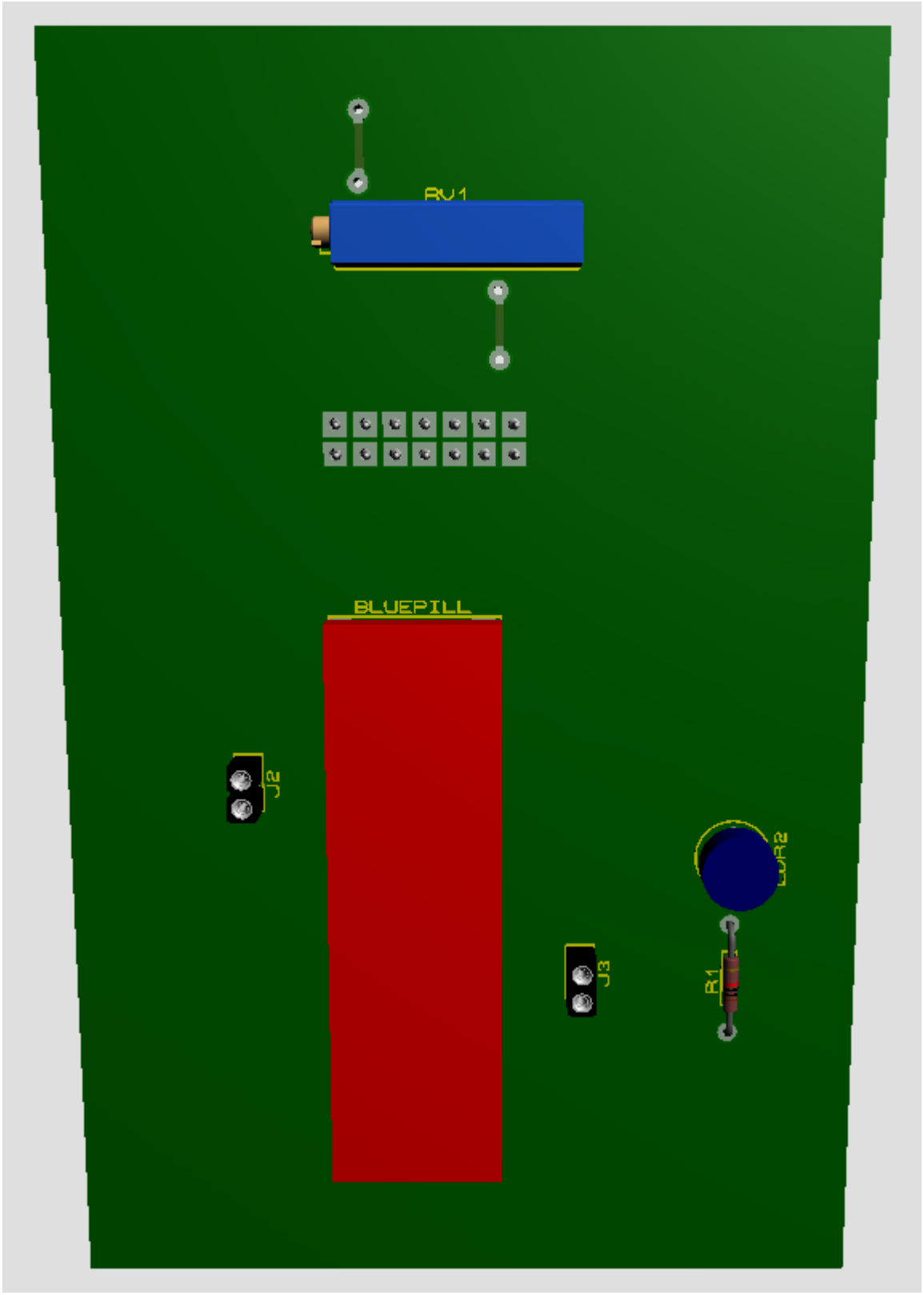
Las pistas utilizadas son de 0.7mm (el mínimo indicado por la consigna) y vías de 70x30 (esto significa 1.8 mm de corona y 0.7mm los agujeros). La separación entre pistas-pistas y pistas-vías también se ajusta de acuerdo a los límites propuestos en la consigna el cual es de 0.7mm.

Ya que nuestra plaqueta es simple faz sólo poseemos conductor de un solo lado de la plaqueta, por lo tanto, los cruces que se hagan del lado de los componentes deberían ser conectados por un cable. En nuestro PCB los conductores rojos representan estos cables que se encuentran del lado superior de la plaqueta. En nuestro diseño solo ocurre en dos casos el cual es el mínimo que se puede llevar a cabo en este diseño y son lo más cortos posibles.

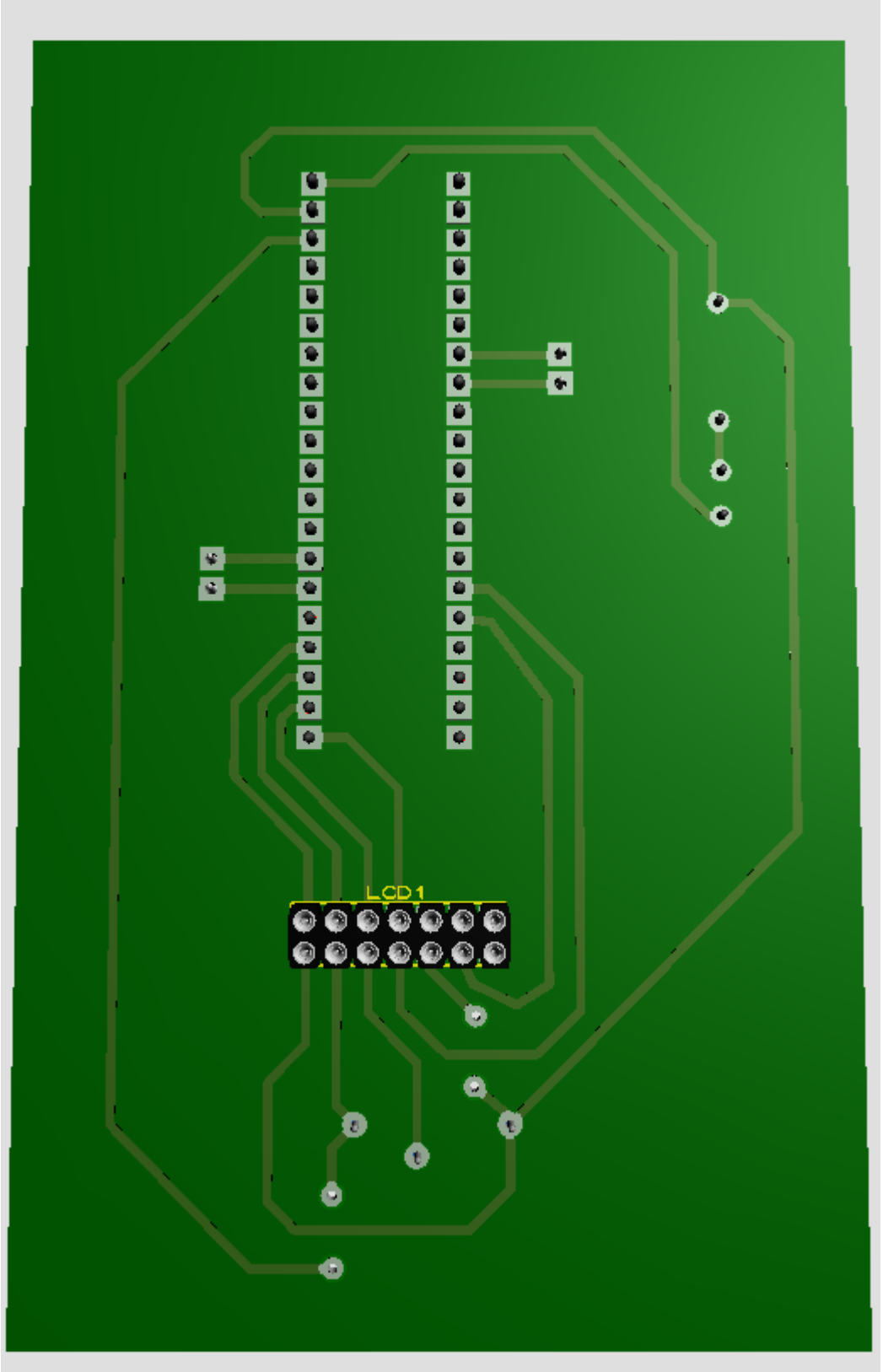

RV1
LCD1
BLUEPILL
PB12
PB13
PB14
PB15
PA8
PA9
PA10
PA11
PA12
PA15
PB3
PB4
PB5
PB6
PB7
PB8
PB9
5V
GND
3V3
GND
GND
3V3
NRST
PB11
PB10
PB1
PB0
PA7
PA6
PA5
PA4
PA3
PA2
PA1
PA0
PC15
PC14
PC13
VBAT
J2
J3
R1
LDR2

VISTA 3D

Capa de componentes



Capa de Soldadura



Main.c

```
#include <stm32f103x6.h>
#include "dc_motor.h"
#include "delay.h"
#include "adc.h"
#include "stdio.h"
#include "usart.h"
#include "mef.h"
#include "timer.h"
#include "stdlib.h"
#include "lcd4bits.h"

volatile uint32_t result;
static uint8_t flag_mef = 0;


void USART1_IRQHandler  () { /* USART1 interrupt routine */
char c = USART1->DR; /* get received data */
 if (c != '\n'){
       if (c == '\r'){  //SI ES CARACTER DE FIN DE CADENA
                if (getTXindice_escritura() >0){
                    usart1_Write_Char_To_Buffer('\0');
                    } else {
                              usart1_Write_Char_To_Buffer('\r');
                              usart1_Write_Char_To_Buffer('\0');
                }
                 setLlegoMensaje(1); //llegoMensaje=1;
                 setTXindice_escritura(0);
                 } else {
                      usart1_Write_Char_To_Buffer(c); // AGREGO AL
BUFFER
                }
   }
 }

int main (void)
{
    //RCC->APB2ENR|= 0xFC | (1<<9) | (1<<14) | (1<<11); //enable
clock for GPIO, ADC1 clock, usart1 and TIM1.
```

```
    timer_init();
   RCC->APB2ENR |= 0xFC | (1<<9) | (1<<14) ; //enable clock for
GPIO, USART1
   GPIOA ->CRL |= 0x44443304;   /* PA2,PA3: output push-pull - PA1
analog input*/


   LCDinit();
   adc_init();
   usart1_init();
   MEF_Init();
   delay_us(1000);
    usart1_sendStr("\r\n BIENVENIDO AL SISTEMA AUTOMATICO DE
ILUMINACION \r\n Para conocer la intensidad de iluminacion del
cuarto ingrese CONOCER INTENSIDAD \r\n Para modificar la
intensidad de iluminacion ingrese ELEGIR INTENSIDAD \r\n \0");
    while(1){
      //SI TIMER DE 1 SEGUNDO SE ACTIVO
        //ACTUALIZAR HORA
        flag_mef = timer_getFlag();
       if (flag_mef){
          MEF_Update();
          timer_resetFlag();
          }
      }
}
```

Adc.c

```
#include "adc.h"
volatile uint32_t volts;

void adc_init(void){
     GPIOA->CRL&=0xFFFFFF0F;  // PA1(ADC_IN1) as analog input
     ADC1->CR2=1; //ADON=1 power-up
     ADC1->SMPR2=1<<3; //SMP1=001
     delay_us(1000); //wait a little bit to make sure adc is
stable
```

```
}


uint32_t adc_read(void){
      ADC1->SQR3=1; //choose channel 1 as input
      ADC1->CR2=1; //ADON=1 (start conversion)
      while((ADC1->SR&(1<<1))==0); //wait until the EOC flag is set

      volts=(ADC1->DR * 3300)/4096; //res*Vref / steps = ~ Vin[mV]
      return volts;
}
```

Adc.h

```
#include <stm32f103x6.h>
#include <stdio.h>
#include <delay.h>
#include <math.h>

void adc_init(void);
uint32_t adc_read(void);
```

Mef.c

```
#include "mef.h"
#include "usart.h"
#include <string.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "lcd4bits.h"
#include "adc.h"
#include "dc_motor.h"
```

```
typedef enum {invalido,ingresar_porcentaje} state_name;

//constantes

//Variables Privadas
static state_name actual_state;
static uint32_t valor_LDR[3][2]={{2154,0},{2817,2400},{9999,2910}}
;
static uint8_t opcion=0;
//static uint8_t time_state= 0;
static uint8_t hora=0,min=0,seg=0,cantTiempo = 9;
static uint8_t stringTime[8]= {'0','0',':','0','0',':','0','0'};
unsigned char* opcionIntensidad[3]={"BAJA \0","MEDIA\0","ALTA
\0"};
uint8_t opcionBytes[3]={4,7,6};


char num='0';
int mensaje;
int num_digits;
unsigned char snum[6];
uint32_t datoADC;


//funciones privadas
void actualizarTiempo(void);
void prepararHora(void);
uint8_t verificarStringValido(unsigned char*);
void processInvalido(void);
void processIngresarPorcentaje(void);
void processConsultarPorcentaje(void);
uint32_t rangoAceptable(uint32_t adc);

uint32_t rangoAceptable(uint32_t adc){
if(adc > valor_LDR[opcion][1]){
    if(adc < valor_LDR[opcion][0]){
         return 1;
    }
}
return 0;
}
```

```
char* itoa(int num, char* buffer, int base) {
    int curr = 0;

    if (num == 0) {
        // Base case
        buffer[curr++] = '0';
        buffer[curr] = '\0';
        return buffer;
    }

    num_digits = 0;

    if (num < 0) {
        if (base == 10) {
            num_digits ++;
            buffer[curr] = '-';
            curr ++;
            // Make it positive and finally add the minus sign
            num *= -1;
        }
        else
            // Unsupported base. Return NULL
            return NULL;
    }

    num_digits += (int)floor(log(num) / log(base)) + 1;

    // Go through the digits one by one
    // from left to right
    while (curr < num_digits) {
        // Get the base value. For example, 10^2 = 1000, for the
third digit
        int base_val = (int) pow(base, num_digits-1-curr);

        // Get the numerical value
        int num_val = num / base_val;

        char value = num_val + '0';
        buffer[curr] = value;

        curr ++;
```

```
        num -= base_val * num_val;
    }
    buffer[curr] = '\0';
    return buffer;
}

void MEF_Init(){
     actual_state = invalido;
}


void MEF_Update(){
    if (++cantTiempo == 10){     //actualizo cada 1s
          actualizarTiempo();
          prepararHora();
          cantTiempo =0;
     }

    datoADC=adc_read();  //LEO LA RESISTENCIA
    if(rangoAceptable(datoADC)){ //SI LA RESISTENCIA ES IGIUAL A
LCDREAD O CERCA
    //NO SE MUEVE
    dc_motor_stop();
    }else{
    //SE MUEVE
    if(datoADC>valor_LDR[opcion][0]){ //si e mayor al limite
superior
       dc_motor_clockwise();     //CIERRA
     }else{
       dc_motor_anticlockwise(); //ABRE
     }
}


     //SI LLEGO MENSAJE
    mensaje=getLlegoMensaje();
     if(mensaje){
     switch(actual_state){
          case invalido:
               processInvalido();
          break;
          case ingresar_porcentaje:
```

```
                processIngresarPorcentaje();
            break;
      }
      setLlegoMensaje(0);
      }

}

void prepararHora(){
      //stringTime[1] =  '0' + (uint8_t)((hora)%  (uint8_t)10);
      stringTime[4] =  '0' + (uint8_t)((min)%  (uint8_t)10);
      stringTime[7] =  '0' + (uint8_t)((seg)%  (uint8_t)10);
      stringTime[0] = ((hora/10)% 10) + '0';
      stringTime[3] = ((min/10)% 10) + '0';
      stringTime[6] = ((seg/10)% 10) + '0';

      //imprimir la hora en el led
      LCDGotoXY(0, 0);
      LCDstring(stringTime, 8);
}
void actualizarTiempo(){

      if(++seg == 60)
      {
            seg = 0;
            if(++min == 60)
            {
                  min =0;
                  if(++hora == 24)
                  {
                        hora=0;
                  }
            }
      }
}

//RETORNA 1 SI ES INVALIDO Y 0 SI NO ES ESTADO INVALIDO
uint8_t verificarStringValido(unsigned char* str){
       usart1_sendStr("buffer: ");
       usart1_sendStr(str);
      usart1_sendStr("\r\n\0");
      if(strncmp(str,"CONOCER INTENSIDAD\0",(sizeof(unsigned
```

```
char))*22) == 0){
        usart1_sendStr("LA INTENSIDAD ES: \0");
       usart1_sendStr(opcionIntensidad[opcion]);
        usart1_sendStr("\r\n\0");
        actual_state=invalido;
        return 0;
        }else{
           if(strncmp(str,"ELEGIR INTENSIDAD\0",(sizeof(unsigned
char))*22) == 0){
            actual_state=ingresar_porcentaje;
         usart1_sendStr("ELIJA UNA OPCION: \r\n\0" );
            usart1_sendStr(" 1.LUMINOSIDAD BAJA \r\n 2.LUMINOSIDAD
MEDIA\r\n 3.LUMINOSIDAD ALTA\r\n\0" );
            return 0;
            }
       }
      return 1;
 }



void processInvalido(){
   if(verificarStringValido(getTX_Buffer())){
      usart1_sendStr("\r\n BIENVENIDO AL SISTEMA AUTOMATICO DE
ILUMINACION \r\n Para conocer la intensidad de iluminacion del
cuarto ingrese CONOCER INTENSIDAD \r\n Para modificar la
intensidad de iluminacion ingrese ELEGIR INTENSIDAD \r\n \0");
      }

}

void processIngresarPorcentaje(){
     int comando_invalido = 1;
   num=getTX_Buffer()[0];
                           LCDGotoXY(0, 1);

                           switch(num)
                           {
                                 case '1' :
                                       opcion=0;
usart1_sendStr("OPCION VALIDA: Su intensidad de iluminacion es
BAJA \0");
```

```
                                        LCDstring("BAJA ",5);
                                    break;
                                    case '2':
                                        opcion=1;
usart1_sendStr("OPCION VALIDA: Su intensidad de iluminacion es
MEDIA\0");
                                        LCDstring("MEDIA",5);
                                    break;
                                    case '3':
                                        opcion=2;
usart1_sendStr("OPCION VALIDA: Su intensidad de iluminacion es
ALTA\0");
                                        LCDstring("ALTA ",5);
                                    break;
                                    default:
                                     //itoa(num,snum,10);
                                     //usart1_sendStr(snum);
                                    usart1_sendStr("NUMERO INVALIDO
Por favor ingrese nuevamente una opcion entre 1 y 3 \r\n \0");
                                    comando_invalido = 0;
                                    break;
                                }
                                if (comando_invalido) {
                                    usart1_sendStr("\r\n\0");
                                    actual_state=invalido;

                                }

}
```

Mef.h

```
#ifndef mef
#define mef
#include <stm32f103x6.h>
#include <string.h>
#define MAX_TIME 999999
#define RANGO 0.1 //10% del valor
#define MAXLUX 3014 //el valor en voltios de la habitacion
completamente iluminada 500LUXES
```

```
#define MIDLUX  2655
#define MINLUX   801

void MEF_Update(void);
void MEF_Init(void);

#endif
```

Lcd4bits.c

```
/* Main.c file generated by New Project wizard
 *
 * Created:   Tue Sep 7 2021
 * Processor: STM32F103C6
 * Compiler:  Keil for ARM
 */
#include <stm32f103x6.h>
#include <lcd4bits.h>
#include <delay.h>


//Cabecera de función privada
 void LCDputValue(uint8_t value);

/*Outputs string to LCD.
Receives the String + the size of the String in Characters*/
void LCDstring(uint8_t* data, uint8_t nBytes)
{
uint8_t i;

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

     for(i=0; i<nBytes; i++){          //Print data in LCD
          LCDsendChar(data[i]);
     }
}

//Inicializacion del LCD en modo 4 bits
//¡¡Setea los pines!!
void LCDinit(){
```

```
  LCD_PORT = 0x33333344;              /* PB10-PB15 as outputs */
  LCD_PIN_OUT &= ~(1<<LCD_EN);   //LCD_EN=0

  delay_us(300);                              //Delay de 3ms
  LCDsendCommand(0x33);                       //Send $33 for init
  LCDsendCommand(0x32);                       //Send $32 for init
  LCDsendCommand(0x28);                       //Init LCD 2 line, 5x7
Matrix
  LCDsendCommand(0x0c);                       //Display On, Curson On
  LCDsendCommand(0x01);                       //Clear LCD

  delay_us(2000);                             //Delay de 2ms
  LCDsendCommand(0x06);//Shift Cursor Right
}

//Codigo para enviar un comando al LCD
void LCDsendCommand (uint8_t cmd)
{
   LCD_PORT_BRR = (1<<LCD_RS); /* RS = 0 for command */
   LCDputValue(cmd);
}

//Codigo para enviar un Char al LCD
void LCDsendChar (uint8_t data)
{
   LCD_PORT_BSRR = (1<<LCD_RS); /* RS = 1 for data */
   LCDputValue(data);
}

/* Codigo para enviar valores al LCD.
Funciona en 4 bits. Primero envia la parte superior,
y luego envia la parte inferior del dato.
*/
void LCDputValue(unsigned char value)
{
   LCD_PORT_BRR = 0xF000;                              /* clear
PA0-PA3  PB12-PB15*/
   LCD_PORT_BSRR = (value<<8)&0xF000;            /* put high
nibble on PA0-PA3 PB12-PB15 */
   LCD_PIN_OUT |= (1<<LCD_EN);                   /* EN = 1 for
H-to-L pulse */
```

```
   delay_us(1);                                                    /*
make EN pulse wider. You can use delay_us(2); too */
   LCD_PIN_OUT &= ~ (1<<LCD_EN);                 /* EN = 0 for
H-to-L pulse */
   delay_us(2000);                                                 /*
wait */

   LCD_PORT_BRR = 0xF000;                                /* clear
PA0-PA3 PB12-PB15*/
   LCD_PORT_BSRR = (value<<12)&0xF000;                   /* put low
nibble on PA0-PA3 PB12-PB15*/
   LCD_PIN_OUT |= (1<<LCD_EN);                           /* EN = 1
for H-to-L pulse */
   delay_us(1);                                                    /*
make EN pulse wider */
   LCD_PIN_OUT &= ~(1<<LCD_EN);                  /* EN = 0 for
H-to-L pulse */
   delay_us(2000);                                                 /*
wait */
}

//Cursor to X Y position
void LCDGotoXY(uint8_t x, uint8_t y){
      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);
}
//Clears LCD
void LCDclr(void){
      LCDsendCommand(1<<LCD_CLR);
}
//Cursor OFF
void LCDcursorOFF(void){
```

```
	LCDsendCommand(0x0C);
}

void LCDSendInt(unsigned int i){
	char str[10];
	sprintf(str,"%d",i);
	LCDstring(str,strlen(str));
}
```

Lcd4bits.h

```
#ifndef LCD_H
#define LCD_H

#include <stm32f103x6.h>
//#include <utils.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define LCD_CLR 0     //DB0: clear display

//PARA CAMBIAR EL PUERTO CAMBIAR GPIOx DONDE x CORRESPONDE AL
PUERTO
#define LCD_PORT           GPIOB->CRH
#define LCD_PIN_IN         GPIOB->IDR            //Input Data
Register
#define LCD_PIN_OUT   GPIOB->ODR            //Output Data Register

//Since BRR and BSRR are opposite of each other,
//you can use both if you don't want to do the bit shift left
operation
#define LCD_PORT_BSRR GPIOB->BSRR
#define LCD_PORT_BRR  GPIOB->BRR

#define LCD_RS 10
#define LCD_RW 12
#define LCD_EN 11
```

```
// 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
#define LCD_DDRAM 7   //DB7: set DD RAM address -> 1<<LCD_DDRAM =>
0x80


/*CABECERAS*/
uint8_t string_len(char* data);
void LCDinit(void);              //Initializes LCD
void LCDsendChar(uint8_t);       //forms data ready to send to
74HC164
void LCDsendCommand(uint8_t);    //forms data ready to send to
74HC164
//void lcd_putValue (uint8_t value); la moví porque es privada
void LCDstring(uint8_t*, uint8_t);    //Outputs string to LCD
void LCDSendInt(unsigned int i); //Outputs int as string to LCD

//TODO - Son necesarias para la MEF
void LCDGotoXY(uint8_t, uint8_t);     //Cursor to X Y position
void LCDclr(void);                    //Clears LCD
void LCDcursorOFF(void);         //Cursor OFF

/*
//En un futuro owo
void LCDhome(void);              //LCD cursor home
void LCDescribeDato(int val,unsigned int field_length); // Agrego
Funcion para escribir Enteros
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 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
```

```
*/

#endif
```

UART.c

```
#include <usart.h>

volatile uint8_t TX_Buffer[]= "Bienvenidos a mi casa";
static uint8_t llegoMensaje=0;
static uint8_t TXindice_escritura=0;


void usart1_Write_Char_To_Buffer(char data) {
   if(TXindice_escritura < TX_BUFFER_LENGTH){
      TX_Buffer[TXindice_escritura] = data;
      TXindice_escritura++;
      } else{
      //MENSAJE DE ERROR DE BUFFER
      usart1_sendStr("ERROR\n\0");
      }
}




void usart1_init(){
     GPIOA->ODR|=(1<<10); //pull-up PA10
     GPIOA->CRH =0x444448B4; //RX1=input with pull-up (PA10), TX1
= alt func output 50MHz (PA9)
     USART1->CR1=0x202C; // 0010 0000 0010 1100 -> Enable USART,
8N1, Receiver enable - TX ENABLE-  RECIEVE INTERRUMPTION
     USART1->BRR=7500; //ACA EN REALIDAD VA 72MHz/9600bps=7500;
for some reason USART1 - PERO EN REALIDAD USAMOS 1 MHZ PQ TODO SE
ROMPIA SINO
        NVIC_EnableIRQ(USART1_IRQn);
}
//doesn't use 36MHz clock by default and uses the 72MHz

void usart1_sendByte(unsigned char c){
```

```
        USART1->DR=c;
      while((USART1->SR&(1<<6))==0); //wait until the TC flag is
set
       USART1->SR &= ~(1<<6);

}
void usart1_sendStr(char* str){
      while(*str!=0){ //while char act is not '\0'
            usart1_sendByte(*str); //print it
            str++; //go to next char
      }
}

void usart1_sendInt(unsigned int i){
      char str[10]; //local variable to store string
      sprintf(str,"%d",i); //transform i to a string and store it
in str
      usart1_sendStr(str); //print it in the USART1 str
}

uint8_t* getTX_Buffer(){
       return TX_Buffer;
 }

 uint8_t getLlegoMensaje(){
       return llegoMensaje;
 }

 void setLlegoMensaje(uint8_t d){
    llegoMensaje=d;
    }

  void setTXindice_escritura(uint8_t d){
    TXindice_escritura=d;
}

uint8_t getTXindice_escritura(){
   return TXindice_escritura;
   }
```

USART.H

```
#include <stm32f103x6.h>
#include <stdint.h>
#include <stdio.h>
#include "mef.h"
#define TX_BUFFER_LENGTH 25

void usart1_init(void);
void usart1_sendByte(unsigned char c);
void usart1_sendStr(char* str);
void usart1_sendInt(unsigned int i);
uint8_t* getTX_Buffer(void);
void setLlegoMensaje(uint8_t d);
void setTXindice_escritura(uint8_t d);
uint8_t getTXindice_escritura(void);
void usart1_Write_Char_To_Buffer(char data);
uint8_t getLlegoMensaje(void);
```

TIMER.C

```
#include "timer.h"
#include <stm32f103x6.h>
#include <stdio.h>


static volatile uint8_t flag = 0;

void timer_init(){
	SysTick_Config (7200000);		//timer cada 25 ms con 1 mhz
de frecuencia 1800000
}

uint8_t timer_getFlag(){
	return flag;
}
```

```
void timer_resetFlag(){
	flag = 0;
}
void SysTick_Handler()
{
	flag = 1;
}
```

TIMER.H

```
#include <stm32f103x6.h>
#include <stdint.h>
#include "delay.h"



void timer_init(void);
uint8_t timer_getFlag(void);
void SysTick_Handler(void);
void timer_resetFlag(void);
```

DC_MOTOR.C

```
#include "dc_motor.h"


void dc_motor_anticlockwise() //giro antihorario
{
	GPIOA->ODR &= ~(1<<2);
	GPIOA->ODR |= (1<<3);
}

void dc_motor_clockwise() //giro horario
{
```

```
      GPIOA->ODR &= ~(1<<3);
      GPIOA->ODR |= (1<<2);
}

void dc_motor_stop(){
      GPIOA->ODR &= ~(1<<3);
      GPIOA->ODR &= ~(1<<2);
}
```

DC_MOTOR.H

```
#include <stm32f103x6.h>

void dc_motor_anti_clockwise(void);
void dc_motor_clockwise(void);
void dc_motor_stop(void);
```

DELAY.C

```
#include "delay.h"
void delay_us (uint32_t t)
{
   volatile unsigned long l = 0;
   uint32_t i;
   for( i= 0; i < t; i++)
   for(l = 0; l < 6; l++)
   { }
}
```

DELAY.H

```
include <stdint.h>
void delay_us (uint32_t );
```

Validación

En el video de validación se muestra:
- Temporización del reloj.
- El correcto funcionamiento del menú
- Como cambia de sentido o para el motor cuando sale del rango seteado y vuelve a este (MEDIO en la simulación).
-Chequeo de comandos inválidos.
Video: https://youtu.be/_YX3cRyn-3c
Github: https://github.com/Pedro-Molina/TP4_TP1