PROYECTO CONTROL DE ACCESO

DESCRIPCIÓN

Diseño e implementación de un circuito basado en un microcontrolador PIC16F1937 que hace la función de control de acceso a un recinto, marcando una clave de ingreso de 4 dígitos. Para esto el circuito cuenta con un Teclado Matricial (4X3) por el cual se ingresa la clave de acceso y una pantalla LCD, en la cual se muestran los diversos mensajes como la hora (en formato de 24 hrs), la temperatura, el código enmascarado con "*", entre otros.

Al pulsar cualquiera de las teclas del Teclado, aparece un pequeño menú de opciones, donde el usuario puede elegir entre:

- Ingresar Clave: Esta opción permite ingresar la clave actual para acceder.
- Cambiar Clave: Esta opción permite cambiar la clave actual por una nueva. Se preguntará primero por la clave actual para confirmar el cambio de clave.

El circuito cuenta también con un Led RGB que indica que el acceso se autorizó o fue negado. Además, también tiene un Zumbador (Buzzer) que suena sólo cuando el acceso fue negado. Se agregó un Relé que controla una carga (por ejemplo un bombillo), el cual simulará la apertura de la puerta.

Mientras no se está ingresando ningún código, en la pantalla se muestra el nombre de una empresa ficticia, la Hora en formato de 24 Hras y la Temperatura.

La base de tiempo para el reloj se implementó con un integrado externo que lleva la base de tiempo (RTC), el DS1307 con protocolo de comunicación I2C.

Para la parte de la temperatura, se utilizó el sensor Analógico LM35.

FUNCIONALIDADES ESPECÍFICAS

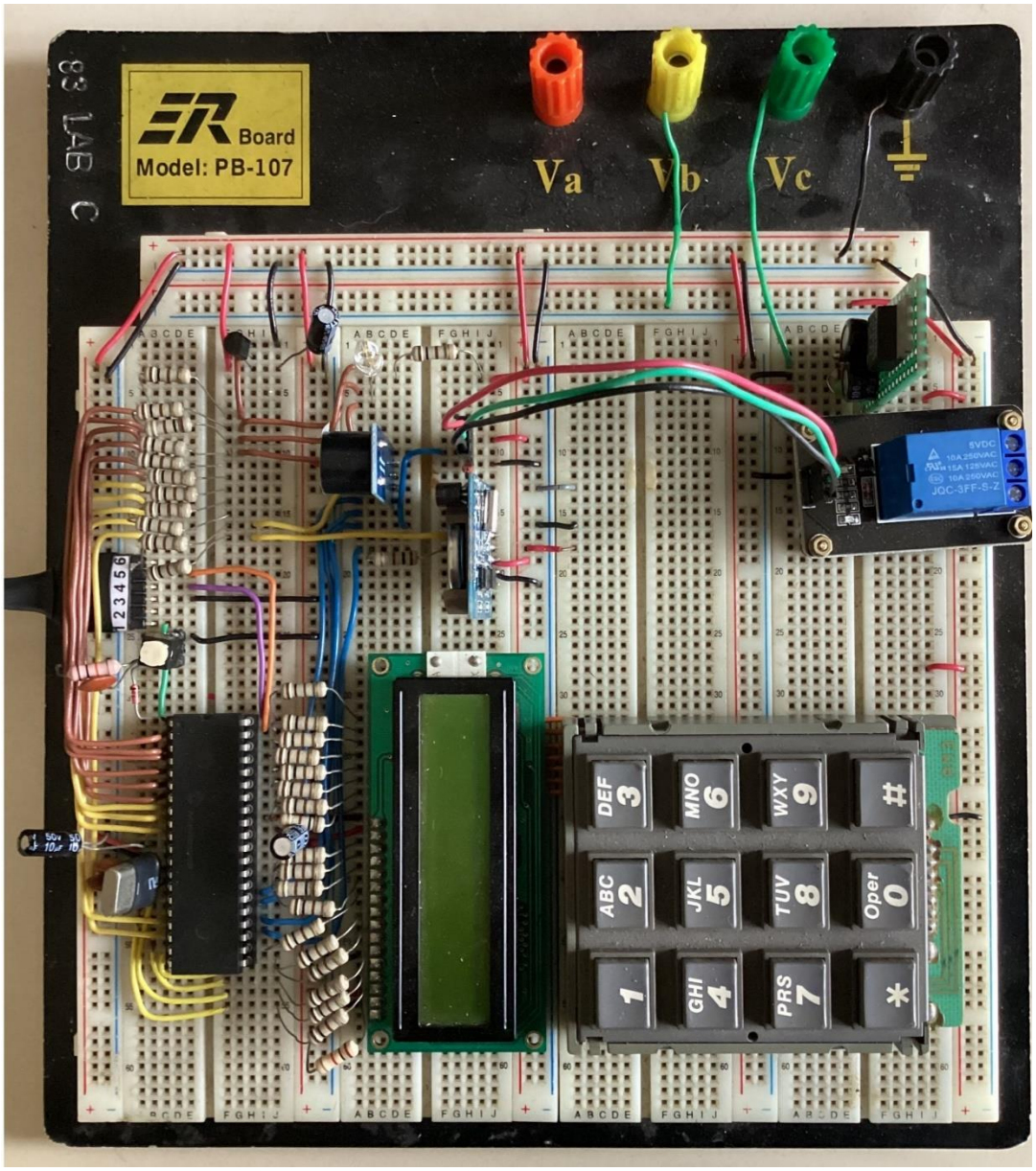
- Mostrar el nombre de la empresa.
- Mostrar hora actual usando el RTC y la temperatura usando sensor analógico.
- Menú con 2 opciones: ¨INGRESAR CLAVE ¨ o ¨CAMBIAR CLAVE¨. En este menú, si se pulsa cualquier tecla diferente de ¨1¨ o ¨2¨, se vuelve al programa principal.
  - Para acceder a ¨INGRESAR CLAVE ¨, se debe pulsar la tecla ¨1¨.
  - Para acceder a ¨CAMBIAR CLAVE ¨, se debe pulsar la tecla ¨2¨.
- LED RGB y Buzzer (Zumbador):
  - LED Azul: Para indicar que el teclado está en uso.

- LED Verde: Para indicar que el acceso fue aprobado o si el cambio de clave fue correcto.
- LED Rojo: Para indicar que el acceso fue denegado o que el cambio de clave fue incorrecto.
- Buzzer: Suena cuando el acceso fue denegado o el cambio de clave fue incorrecto.

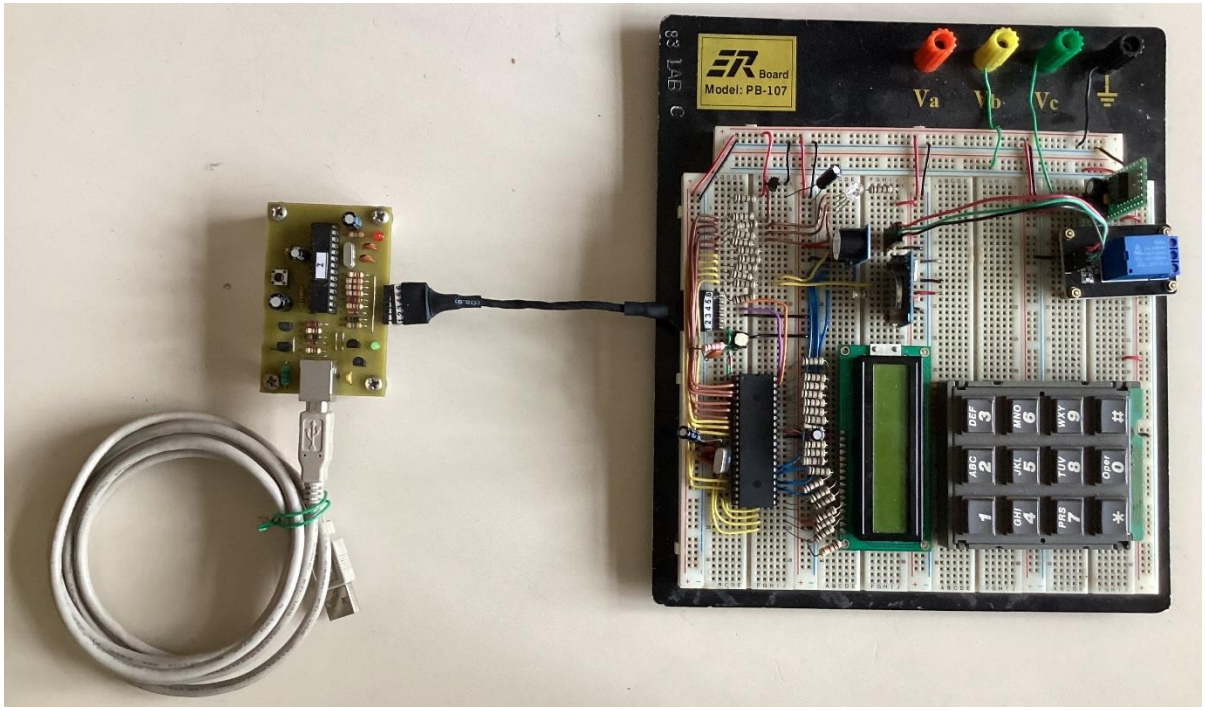
- Relé: Simula si se abrió o no la puerta. En caso de que el acceso fue permitido se abre la puerta (en la simulación un bombillo se prende), en caso de que fue denegado, no se abre (bombillo apagado).
- La clave es almacenada en la memoria EEPROM del PIC.

CLAVE POR DEFECTO: 1234

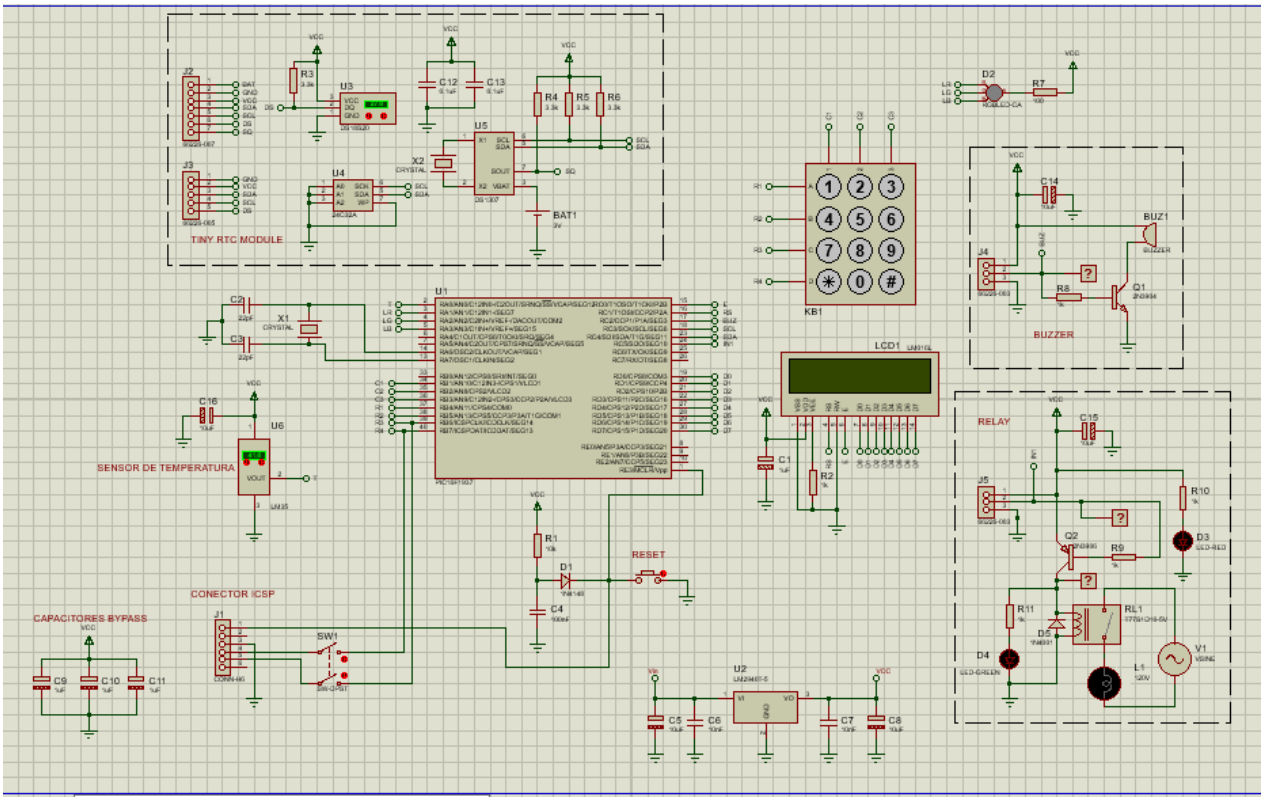
Montaje en Protoboard



Montaje en Protoboard con Programador PICkit2 Clone

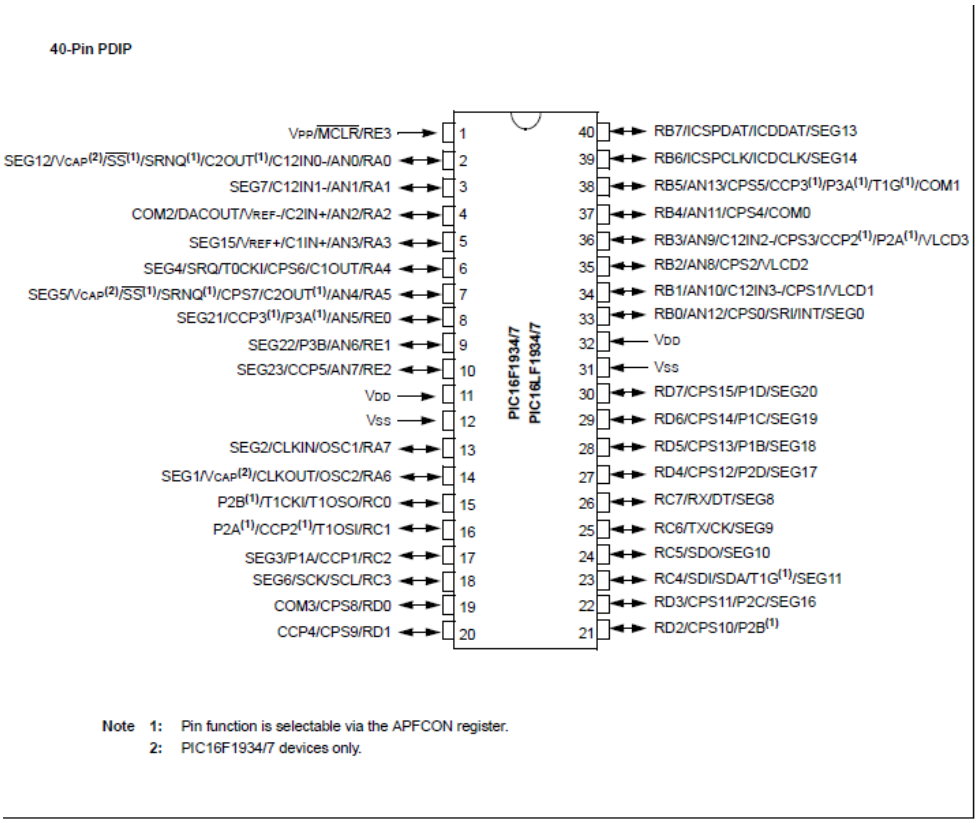


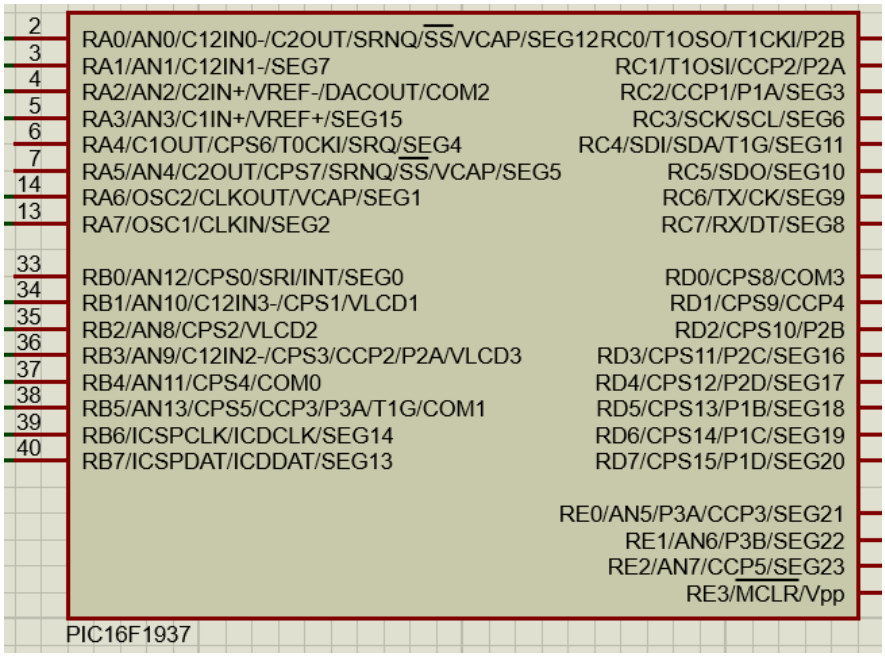
Esquemático en Proteus



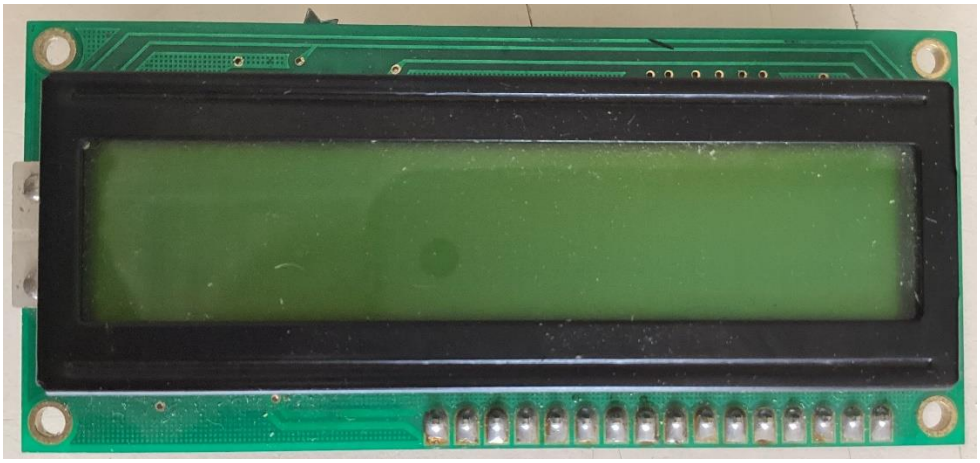
COMPONENTES

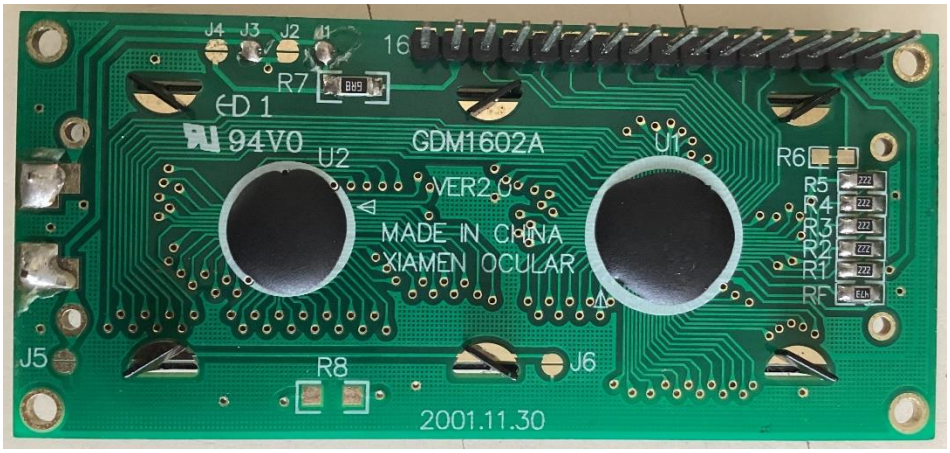
PIC16F1937

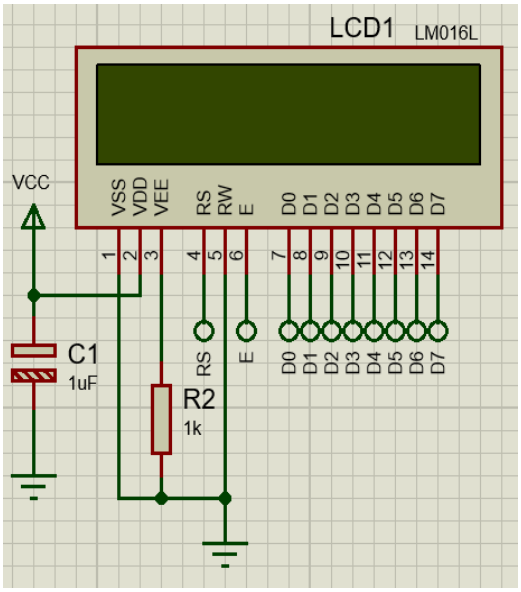




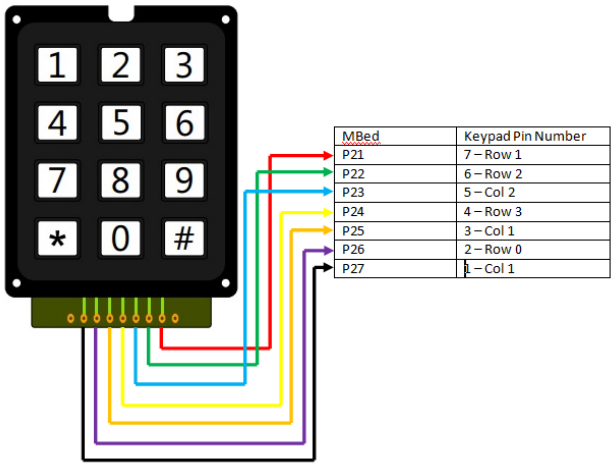
LCD



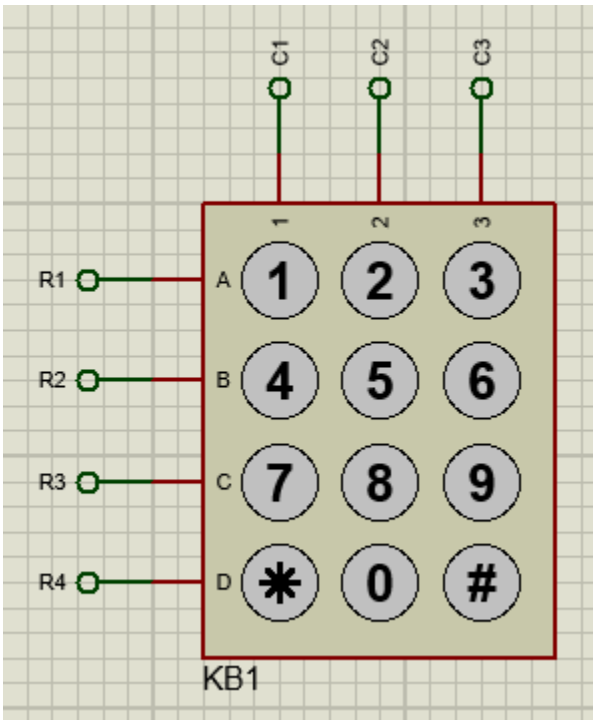

J4 J3 J2 J1
16
R7
D 1
94V0
U2
GDM1602A
VER2.0
U1
R6
R5
R4
R3
R2
R1
RF
MADE IN CHINA
XIAMEN OCULAR
J5
R8
J6
2001.11.30


LCD1
LM016L
VCC
VSS
VDD
VEE
RS
RW
E
D0
D1
D2
D3
D4
D5
D6
D7
1
2
3
4
5
6
7
8
9
10
11
12
13
14
C1
1uF
R2
1k

Teclado



MBed	Keypad Pin Number
P21	7 – Row 1
P22	6 – Row 2
P23	5 – Col 2
P24	4 – Row 3
P25	3 – Col 1
P26	2 – Row 0
P27	1 – Col 1



DS1307
64 x 8, Serial, I²C Real-Time Clock

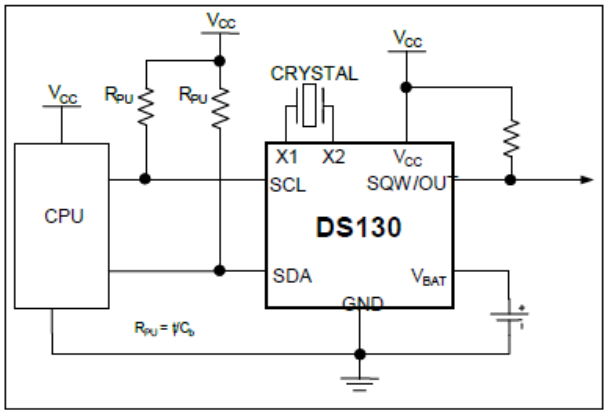
GENERAL DESCRIPTION

The DS1307 serial real-time clock (RTC) is a low-power, full binary-coded decimal (BCD) clock/calendar plus 56 bytes of NV SRAM. Address and data are transferred serially through an I²C, bidirectional bus. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with AM/PM indicator. The DS1307 has a built-in power-sense circuit that detects power failures and automatically switches to the backup supply. Timekeeping operation continues while the part operates from the backup supply.

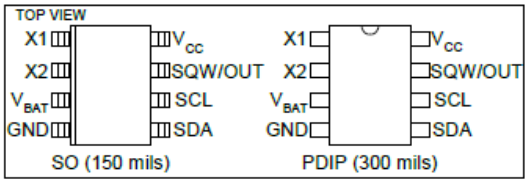
FEATURES

- Real-Time Clock (RTC) Counts Seconds, Minutes, Hours, Date of the Month, Month, Day of the week, and Year with Leap-Year Compensation Valid Up to 2100
- 56-Byte, Battery-Backed, General-Purpose RAM with Unlimited Writes
- I²C Serial Interface
- Programmable Square-Wave Output Signal
- Automatic Power-Fail Detect and Switch Circuitry
- Consumes Less than 500nA in Battery-Backup Mode with Oscillator Running
- Optional Industrial Temperature Range: -40°C to +85°C
- Available in 8-Pin Plastic DIP or SO
- Underwriters Laboratories (UL) Recognized

TYPICAL OPERATING CIRCUIT



PIN CONFIGURATIONS



LM35

LM35 Precision Centigrade Temperature Sensors

FEATURES

- Calibrated Directly in ° Celsius (Centigrade)
- Linear + 10 mV/°C Scale Factor
- 0.5°C Ensured Accuracy (at +25°C)
- Rated for Full −55°C to +150°C Range
- Suitable for Remote Applications
- Low Cost Due to Wafer-Level Trimming
- Operates from 4 to 30 V
- Less than 60-µA Current Drain
- Low Self-Heating, 0.08°C in Still Air
- Nonlinearity Only ±¼°C Typical
- Low Impedance Output, 0.1 Ω for 1 mA Load

DESCRIPTION

The LM35 series are precision integrated-circuit temperature sensors, with an output voltage linearly proportional to the Centigrade temperature. Thus the LM35 has an advantage over linear temperature sensors calibrated in ° Kelvin, as the user is not required to subtract a large constant voltage from the output to obtain convenient Centigrade scaling. The LM35 does not require any external calibration or trimming to provide typical accuracies of ±¼°C at room temperature and ±¾°C over a full −55°C to +150°C temperature range. Low cost is assured by trimming and calibration at the wafer level. The low output impedance, linear output, and precise inherent calibration of the LM35 make interfacing to readout or control circuitry especially easy. The device is used with single power supplies, or with plus and minus supplies. As the LM35 draws only 60 µA from the supply, it has very low self-heating of less than 0.1°C in still air. The LM35 is rated to operate over a −55°C to +150°C temperature range, while the LM35C is rated for a −40°C to +110°C range (−10° with improved accuracy). The LM35 series is available packaged in hermetic TO transistor packages, while the LM35C, LM35CA, and LM35D are also available in the plastic TO-92 transistor package. The LM35D is also available in an 8-lead surface-mount small-outline package and a plastic TO-220 package.

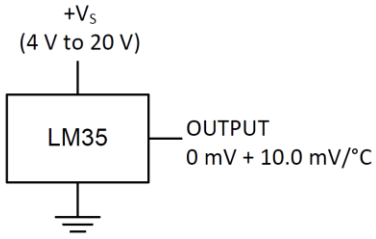
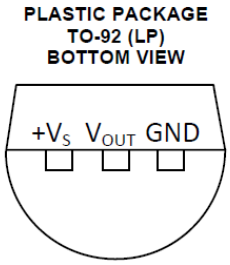
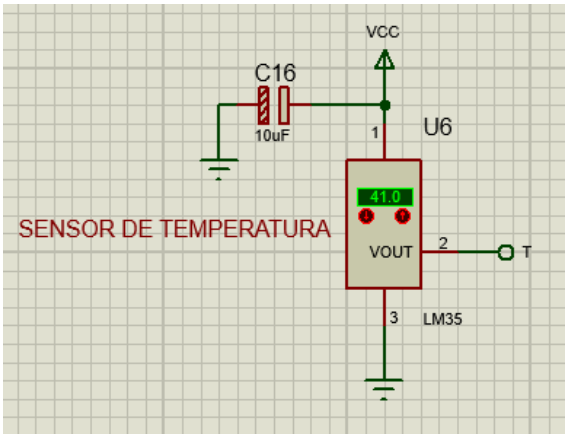
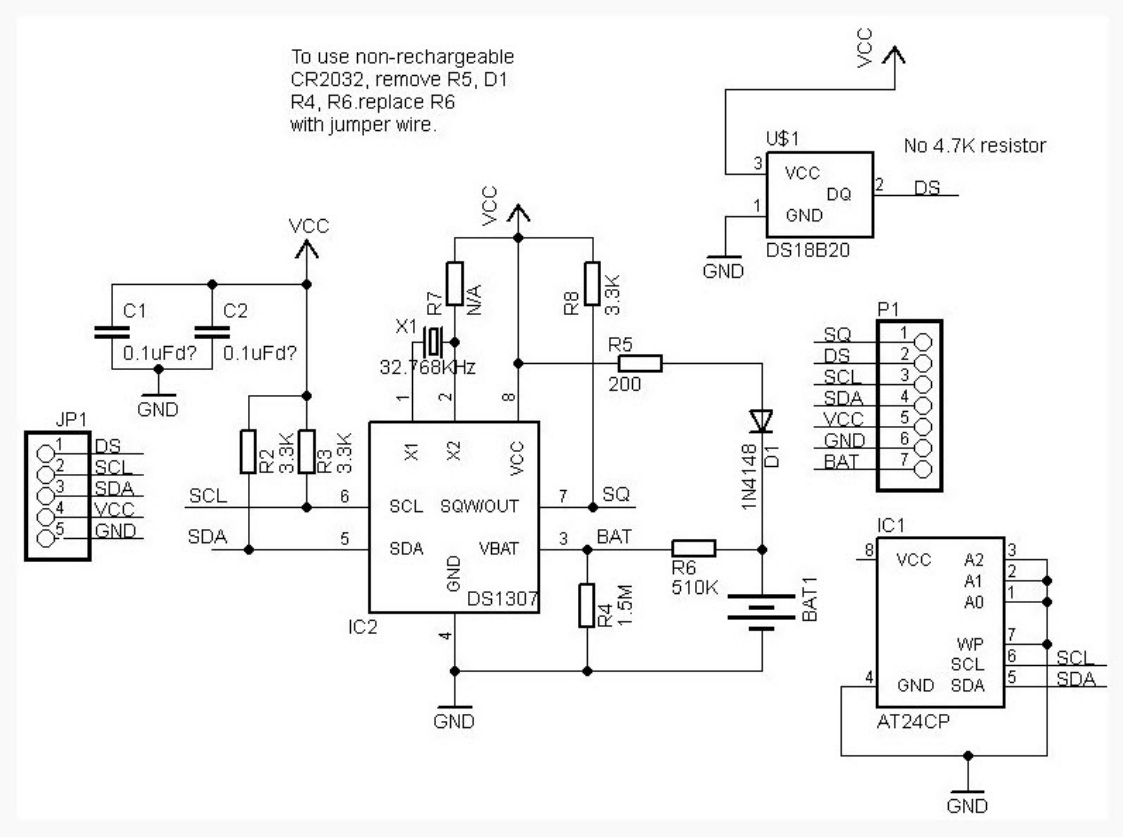


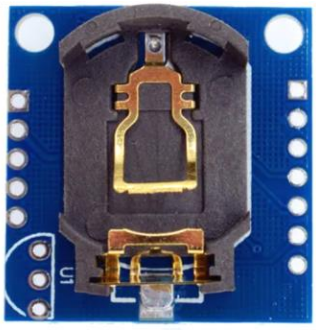
Figure 1. Basic Centigrade Temperature Sensor (+2°C to +150°C)

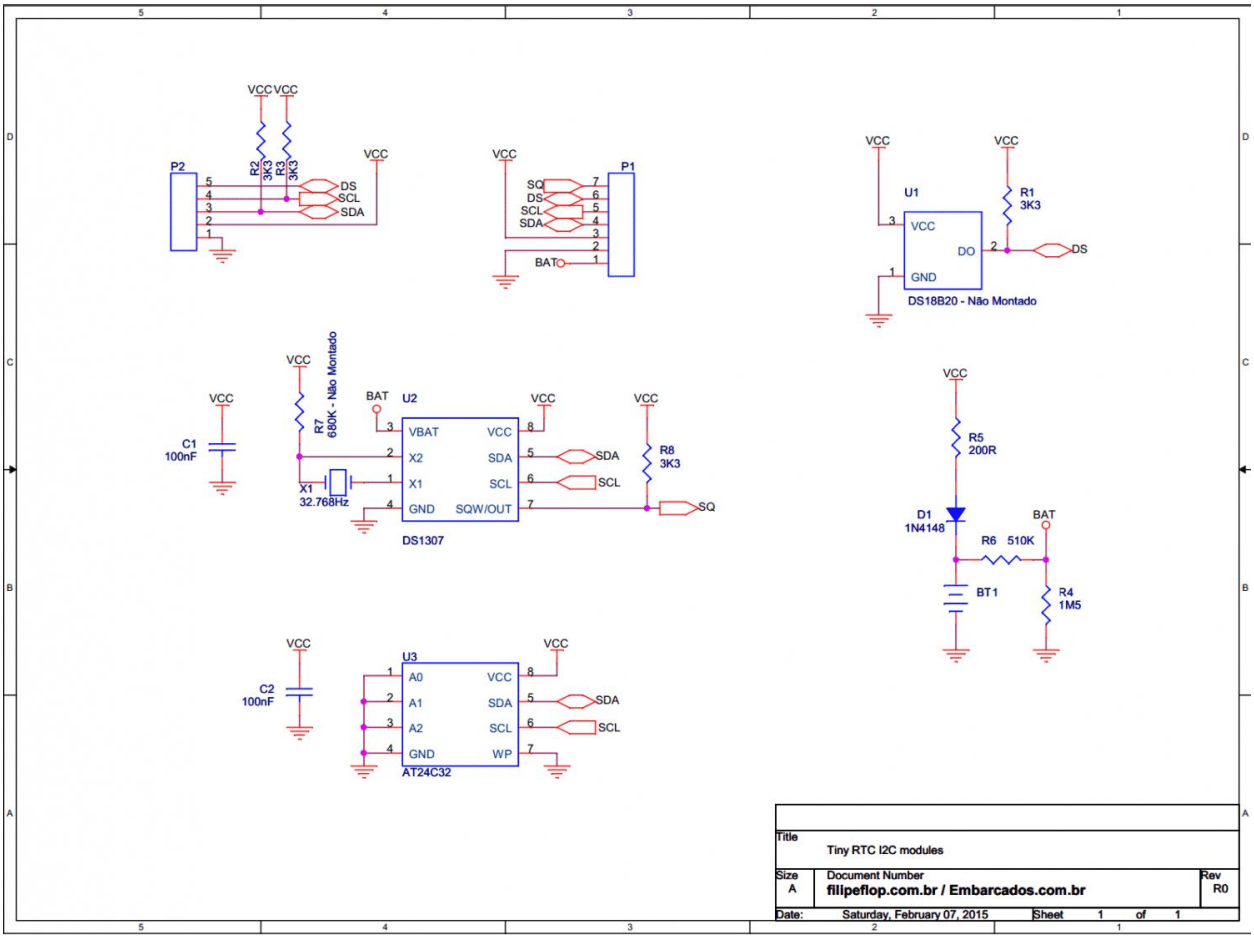


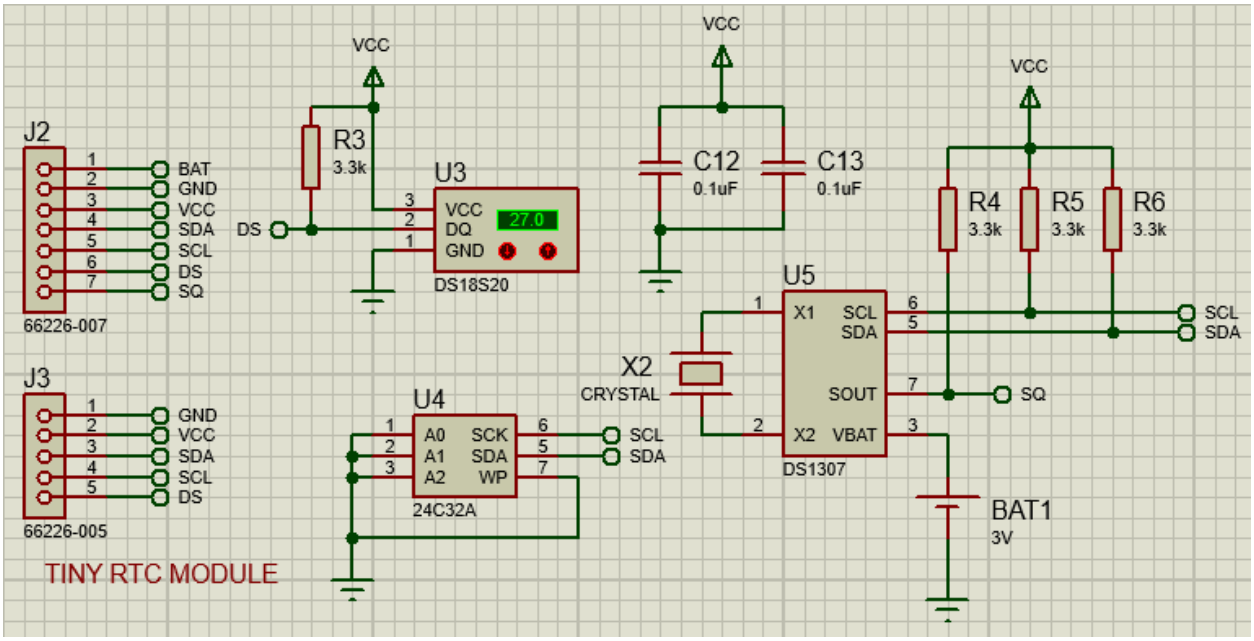


Tiny RTC Module

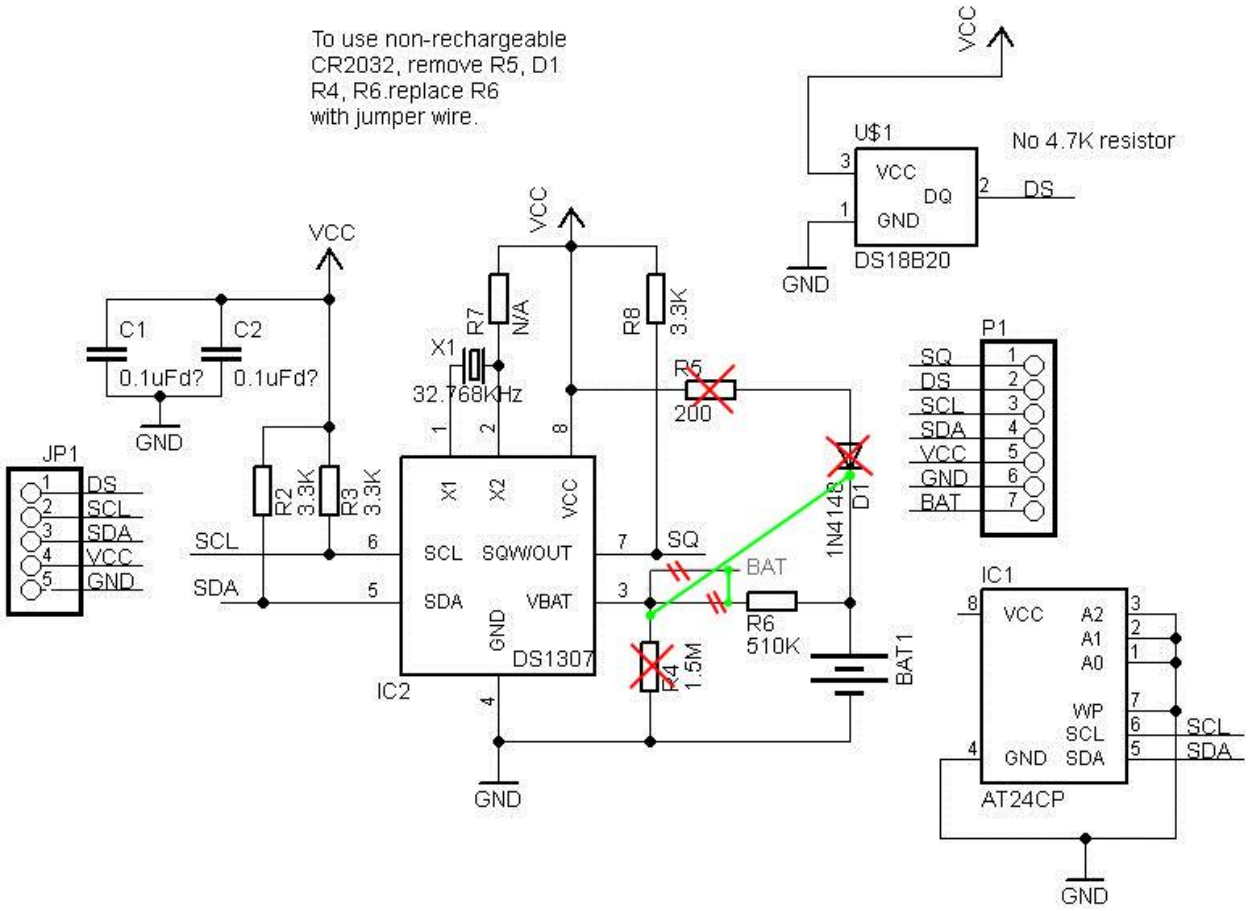





P2
R2 3K3
R3 3K3
VCC
DS
SCL
SDA
P1
SQ
DS
SCL
SDA
BAT
U1
R1 3K3
DO
GND
DS18B20 - Não Montado
C1 100nF
R7 680K - Não Montado
X1 32.768Hz
U2
VBAT
X2
X1
GND
VCC
SDA
SCL
SQW/OUT
R8 3K3
DS1307
R5 200R
D1 1N4148
R6 510K
BT1
R4 1M5
C2 100nF
U3
A0
A1
A2
GND
WP
AT24C32
Title
Tiny RTC I2C modules
Size A
Document Number
filipeflop.com.br / Embarcados.com.br
Rev R0
Date: Saturday, February 07, 2015
Sheet 1 of 1


J2
BAT
GND
VCC
SDA
SCL
DS
SQ
66226-007
J3
GND
VCC
SDA
SCL
DS
66226-005
TINY RTC MODULE
R3 3.3k
DS
U3
DQ
27.0
DS18S20
U4
A0
A1
A2
SCK
SDA
WP
24C32A
C12 0.1uF
C13 0.1uF
X2
CRYSTAL
U5
X1
X2
SCL
SDA
SOUT
VBAT
DS1307
R4 3.3k
R5 3.3k
R6 3.3k
BAT1
3V

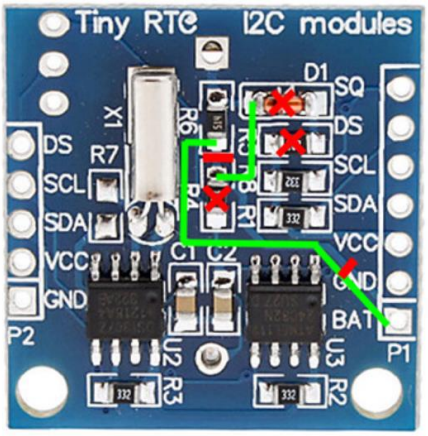
Tiny RTC Module modifications



Cut track

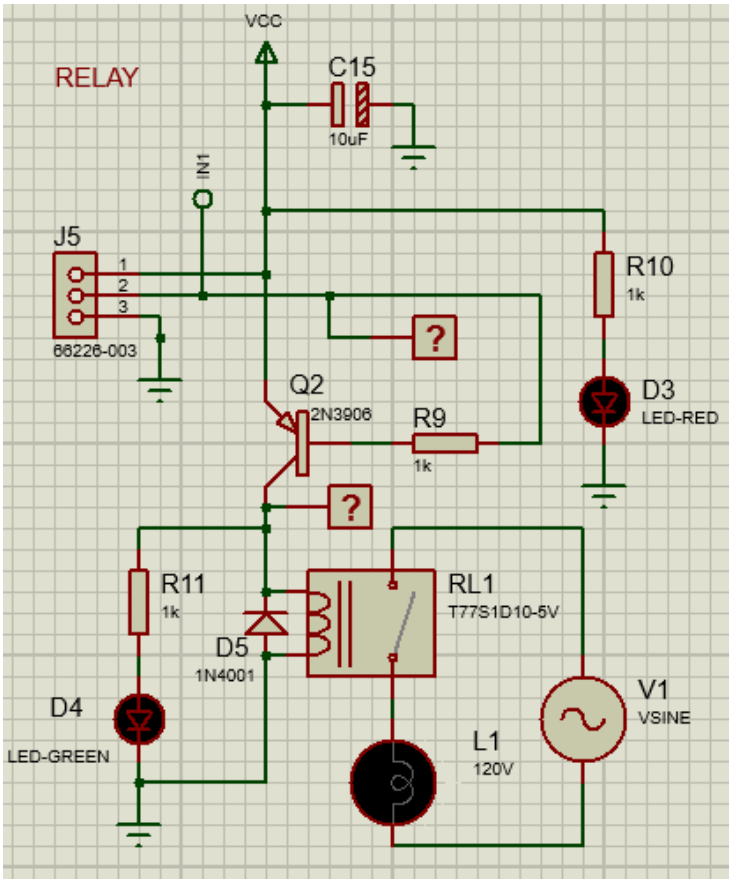
Remove component

New connection

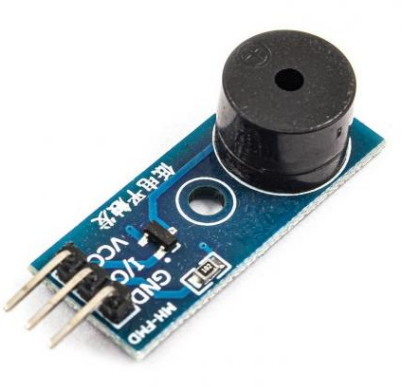


Relay


VCC
IN1
GND
NC
COM
NO
SRD-05VDC-SL-C
SONGLE

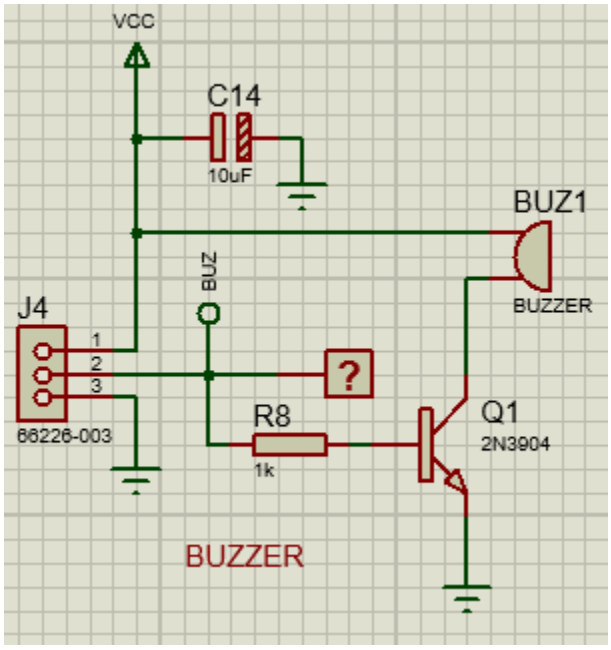

VCC
RELAY
C15
10uF
IN1
J5
1
2
3
66226-003
R10
1k
?
Q2
2N3906
R9
1k
D3
LED-RED
?
R11
1k
RL1
T77S1D10-5V
D5
1N4001
D4
LED-GREEN
V1
VSINE
L1
120V

Buzzer

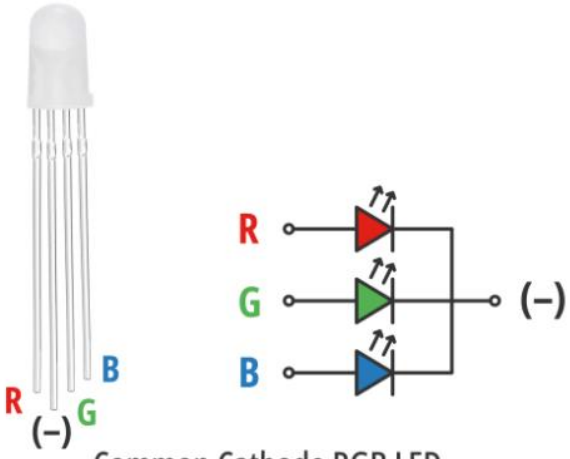



MH-FMD
GND
I/O
VCC
低电平触发

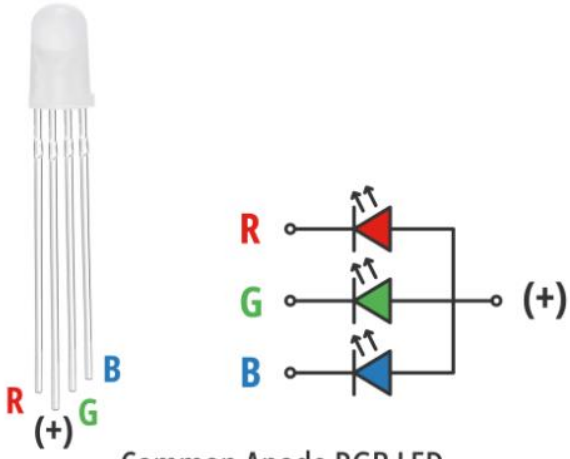

Buzzer module
Low level trigger


VCC
C14
10uF
BUZ1
BUZZER
BUZ
J4
1
2
3
66226-003
?
R8
1k
Q1
2N3904
BUZZER

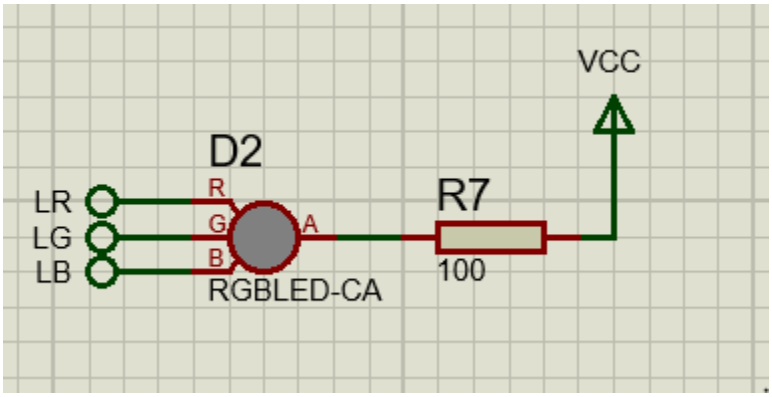
LED RGB



Common Cathode RGB LED



Common Anode RGB LED



CÓDIGO

```c
/***********LIBRERIAS************/
#include <xc.h>
#include <stdio.h>
#include <string.h>
#include "lcd.h" // libreria para pantalla LCD
#include "kbd4x3.h" // libreria para teclado 4x3
#include "eeprom.h" // libreria para la EEPROM
#include "mcc_generated_files/mcc.h" // MCC

/**************DEFINICIONES*********/
#define _XTAL_FREQ 4000000
#define TRIS_LR TRISAbits.TRISA1 // LED RED
#define TRIS_LG TRISAbits.TRISA2 // LED GREEN
#define TRIS_LB TRISAbits.TRISA3 // LED BLUE
#define TRIS_BUZ TRISCbits.TRISC2 // BUZZER
#define TRIS_REL TRISCbits.TRISC5 // RELAY
#define LAT_LR LATAbits.LATA1 // LED RED
#define LAT_LG LATAbits.LATA2 // LED GREEN
#define LAT_LB LATAbits.LATA3 // LED BLUE
#define LAT_BUZ LATCbits.LATC2 // BUZZER
#define LAT_REL LATCbits.LATC5 // RELAY

/**********VARIABLES GLOBALES**************/
// Clave por defecto "1234"
// Para cambiar la clave, solo se deben modificar los primeros 4 bytes de la clave,
// el resto de bytes se deja en 0xFF
__EEPROM_DATA('1','2','3','4',0xFF,0xFF,0xFF,0xFF); // almacenar clave por defecto en EEPROM

char tecla;                 // Almacena el valor de la tecla presionada
char clave[5];              // Almacena la clave ingresada por el usuario
char clave_enter[5]; // guardar clave en la EEPROM
uint8_t cont_key = 0; // contador de la clave ingresada
uint8_t estado = 0; // estado para saber si INGRESAR CLAVE O CAMBIAR CLAVE
char buffer[20]; // buffer para lcd
uint16_t convertedValue; // valor ADC convertido
float temp; // Almacenar valor temperatura LM35
char seg,min,hor; // guardar datos del RTC

/*******FUNCIONES EEPROM**********/
void Escribir_Clave(char* str);
void Leer_Clave(char* str);

/**********PROGRAMA PRINCIPAL*****************/
void main() {

    // configuracion puertos y pines
    ANSELB = 0; // PORTB como digital
    TRIS_LR = 0; // LED RED como salida
    TRIS_LG = 0; // LED GREEN como salida
    TRIS_LB = 0; // LED BLUE como salida
    TRIS_BUZ = 0; // BUZZER como salida
```

```
    TRIS_REL = 0; // RELAY como salida
    TRISCbits.TRISC3 = 1; // SCL como entrada
    TRISCbits.TRISC4 = 1; // SDA como entrada
    LAT_LR = 1; // Led RED inicialmente apagado (CA)
    LAT_LG = 1; // Led GREEN inicialmente apagado (CA)
    LAT_LB = 1; // Led BLUE inicialmente apagado (CA)
    LAT_BUZ = 0; // BUZZER inicialmente apagado
    LAT_REL = 1; // puerta cerrada inicialmente

    // resistencias pull up en las columnas del teclado para las interrupciones
    OPTION_REGbits.nWPUEN = 0;
    WPUB = 0b00001110; // pull ups activadas en RB1,RB2,RB3 (columnas)

    // columnas teclado como salidas con valor inicial de 0 para detectar interrupcion
    C1_DIR = 0;
    LATBbits.LATB1 = 0;
    C2_DIR = 0;
    LATBbits.LATB2 = 0;
    C3_DIR = 0;
    LATBbits.LATB3 = 0;

    // interrupciones
    INTCONbits.GIE = 1; // habilitar interrupciones globales
    INTCONbits.IOCIE = 0; // limpiar flag interrupcion
    INTCONbits.IOCIE = 1; // habilitar interrupciones IoC (interrupt on change)
    //IOCBP = 0b11110000; // Interrupciones por flanco de subida
    IOCBN = 0b11110000; // Interrupciones por flanco de bajada
    IOCBF = 0; // flag

    // MSSP
    SSPCON1bits.SSPEN = 1; // habilitar puerto MSSP
    SSPCON1bits.SSPM3 = 1;
    SSPCONbits.SSPM2 = 0; // Modo I2C Maestro
    SSPCONbits.SSPM1 = 0;
    SSPCON1bits.SSPM0 = 0;
    SSPADD = 0x4F; // 100KHz Baud Rate

    // Inicializaciones
    ADC_Initialize(); // inicializar ADC
    Lcd_Init();       // Inicializa la pantalla lcd
    Lcd_Clear(); // limpiar lcd

    __delay_ms(200); // delay de estabilizacion

    // Trama Escritura RTC -> Reloj 24 hrs
    // Hora de prueba: 15:00:00 (se puede modificar para configurar la hora adecuada)
    SSPCON2bits.SEN = 1; // generar senal de start
    while(SSPCON2bits.SEN); // esperar que se genere senal
    SSPBUF = 0b11010000; // direccion +0 de escritura
    while(SSPSTATbits.BF); // esperar que se envie el dato
    while(SSPSTATbits.R_nW); // esperar que transcurra la senal de ACK
    SSPBUF = 0; // direccion 0x00 (segundos)
    while(SSPSTATbits.BF); // esperar que se envie el dato
```

```
105         while(SSPSTATbits.R_nW); // esperar que transcurra la senal de ACK
106         SSPBUF = 0; // 0 segundos
107         while(SSPSTATbits.BF); // esperar que se envie el dato
108         while(SSPSTATbits.R_nW); // esperar que transcurra la senal de ACK
109         SSPBUF = 0; // 0 minutos
110         while(SSPSTATbits.BF); // esperar que se envie el dato
111         while(SSPSTATbits.R_nW); // esperar que transcurra la senal de ACK
112         SSPBUF = 0b00010101; // 15 horas
113         while(SSPSTATbits.BF); // esperar que se envie el dato
114         while(SSPSTATbits.R_nW); // esperar que transcurra la senal de ACK
115         SSPCON2bits.PEN = 1; // genera senal de stop
116         while(SSPCON2bits.PEN); // esperar senal de stop
117 
118         // mensaje inicial
119         Lcd_Set_Cursor(1,2);
120         Lcd_Write_String("ControlTech CA");
121 
122         __delay_ms(200); // delay de estabilizacion
123 
124         while (1) {
125 
126             // Trama Lectura RTC -> Reloj 24 hrs
127             SSPCON2bits.SEN = 1; // genera senal de start
128             while(SSPCON2bits.SEN); // esperar que se genere senal
129             SSPBUF = 0b11010000; // direccion +0 de escritura
130             while(SSPSTATbits.BF); // esperar que se envie el dato
131             while(SSPSTATbits.R_nW); // esperar que transcurra la senal de ACK
132             SSPBUF = 0; // direccion 0x00 (segundos)
133             while(SSPSTATbits.BF); // esperar que se envie el dato
134             while(SSPSTATbits.R_nW); // esperar que transcurra la senal de ACK
135             SSPCON2bits.RSEN = 1; // generar senal de restart
136             while(SSPCON2bits.RSEN); // esperar que se genere la senal
137             SSPBUF = 0b11010001; // Direccion +1 de lectura
138             while(SSPSTATbits.BF); // esperar que se envie el dato
139             while(SSPSTATbits.R_nW); // esperar que transcurra la senal de ACK
140             SSPCON2bits.RCEN = 1; // Modo de recepcion
141             while(SSPSTATbits.BF==0); // esperar a que llegue el dato
142             seg = SSPBUF; // segundos
143             SSPCON2bits.ACKDT = 0; // dato ACK
144             SSPCON2bits.ACKEN = 1; // genera senal de ACK o NACK
145             while(SSPCON2bits.ACKEN); // esperar que se envie la senal
146             SSPCON2bits.RCEN = 1; // Modo de recepcion
147             while(SSPSTATbits.BF==0); // esperar a que llegue el dato
148             min = SSPBUF; // minutos
149             SSPCON2bits.ACKDT = 0; // dato ACK
150             SSPCON2bits.ACKEN = 1; // genera senal de ACK o NACK
151             while(SSPCON2bits.ACKEN); // esperar que se envie la senal
152             SSPCON2bits.RCEN = 1; // Modo de recepcion
153             while(SSPSTATbits.BF==0); // esperar a que llegue el dato
154             hor = SSPBUF; // horas
155             SSPCON2bits.ACKDT = 1; // dato NACK
156             SSPCON2bits.ACKEN = 1; // genera senal de ACK o NACK
```

```
157             while(SSPCON2bits.ACKEN); // esperar que se envie la senal
158             SSPCON2bits.PEN = 1; // genera senal de stop
159             while(SSPCON2bits.PEN); // esperar senal de stop
160 
161             // Lectura del ADC (Temperatura LM35)
162             ADC_StartConversion(); // iniciar conversiï¿½n ADC
163             while (!ADC_IsConversionDone()); // esperar conversiï¿½n
164             convertedValue = ADC_GetConversionResult(); // guardar resultado
165             temp = (float)(convertedValue*5*100)/1024; // formula temperatura
166 
167             // mostrar hora en LCD
168             sprintf(buffer,"%d%d:%d%d:%d%d",hor>>4,hor&=0x0F,min>>4,min&=0x0F,seg>>4,seg&=0x0F);
169             Lcd_Set_Cursor(2,1);
170             Lcd_Write_String(buffer);
171 
172             // mostrar temperatura en LCD
173             Lcd_Set_Cursor(2,10);
174             sprintf(buffer,"%0.2f\xDF",temp);
175             Lcd_Write_String(buffer);
176             Lcd_Set_Cursor(2,16);
177             Lcd_Write_String("C");
178 
179             __delay_ms(1000); // lectura de tiempo y temp cada segundo
180         }
181     }
182 
```

```
183 /*************FUNCION INTERRUPCION TECLADO************/
184 void __interrupt() IOC(void) {
185     Keypad_Init();   // Inicializa el teclado matricial 4x3
186     Lcd_Clear();
187     LAT_LB = 0; // Led azul encendido
188     __delay_ms(500); // delay de estabilizacion
189 
190     while(IOCBF != 0) {
191         switch(estado) {
192             case 0: // Menu de opciones
193                 Lcd_Set_Cursor(1,1);
194                 Lcd_Write_String("1:INGRESAR CLAVE");
195                 Lcd_Set_Cursor(2,1);
196                 Lcd_Write_String("2:CAMBIAR CLAVE");
197 
198                 tecla = Keypad_Get_Char();
199                 if(tecla != 0) {
200                     switch(tecla) {
201                         case '1':
202                             estado = 1; // modo ingresar clave
203                             break;
204                         case '2':
205                             estado = 2; // modo cambiar clave
206                             break;
207                         default:
208                             estado = 0; // menu de opciones
209 
```

```
                        // si se pulsa otra tecla diferente de 1 o 2, se vuelve al programa principal
                        WPUB = 0b11111110; // activar pull up en filas para deshabilitar teclado
                        __delay_ms(200);
                        // columnas vuelven a estado original 0
                        LATBbits.LATB1 = 0;
                        LATBbits.LATB2 = 0;
                        LATBbits.LATB3 = 0;
                        // desactivar pull ups en filas y activar en columnas
                        WPUB = 0b00001110;
                        IOCBF = 0; // reiniciar flag
                        break;
                    }
                    Lcd_Clear();
                }
                break;
            case 1: // Modo Ingresar Clave
                Lcd_Set_Cursor(1,2);
                Lcd_Write_String("INGRESAR CLAVE");

                while(cont_key < 4)
                {
                    tecla = Keypad_Get_Char(); // Lee el dato de la tecla presionada
                    if(tecla != 0)  // Verifica si se ha presionado alguna tecla
                    {
                        clave[cont_key] = tecla;   // Almacena cada tecla presionada en el arreglo
                        Lcd_Set_Cursor(2,7+cont_key);
                        Lcd_Write_Char('*'); // Muestra la tecla presionada.
```

```
                        cont_key++;  // Incrementa el contador
                    }
                }
                /*hay que activar pull ups en filas para que el teclado no funcione y se bloquee
                durante el chequeo de clave. Esto evita bugs ya que si se presionan teclas durante esta fase,
                la interrupcion no es detectada correctamente en ciertas teclas */

                WPUB = 0b11111110; // activar pull up en filas para deshabilitar teclado
                __delay_ms(200);
                Lcd_Clear();   // Limpia la pantalla lcd
                Leer_Clave(clave_enter); // Lee la clave almacenada en la memoria EEPROM
                LAT_LB = 1; // LED BLUE apagado

                if(!strcmp(clave, clave_enter)) // Compara si la clave es la correcta
                {
                    Lcd_Set_Cursor(1,6);
                    Lcd_Write_String("ACCESO");
                    Lcd_Set_Cursor(2,5);
                    Lcd_Write_String("APROBADO");
                    LAT_LG = 0; // LED GREEN encendido
                    LAT_REL = 0; // Puerta abierta
                    __delay_ms(10000); // 10s
                }
                else   // Sino es la clave correcta, no permite el acceso
                {
                    Lcd_Set_Cursor(1,6);
                    Lcd_Write_String("ACCESO");
```

```
                Lcd_Set_Cursor(2,5);
                Lcd_Write_String("DENEGADO");
                LAT_LR = 0; // LED RED encendido
                LAT_BUZ = 1; // BUZZER encendido
                LAT_REL = 1; // Puerta cerrada
                __delay_ms(5000); // 5s
            }
            cont_key = 0;          // Reinicia el contador
            estado = 0; // reiniciar estado
            Lcd_Clear();  // Limpia la pantalla lcd
            LAT_LG = 1; // LED GREEN apagado
            LAT_LR = 1; // LED RED apagado
            LAT_BUZ = 0; // BUZZER apagado
            LAT_REL = 1; // Puerta cerrada
            // columnas vuelven a estado original 0
            LATBbits.LATB1 = 0;
            LATBbits.LATB2 = 0;
            LATBbits.LATB3 = 0;
            // desactivar pull ups en filas y activar en columnas
            WPUB = 0b00001110;
            IOCBF = 0; // reiniciar flag
            break;
        case 2: // Modo cambiar clave
            Lcd_Set_Cursor(1,3);
            Lcd_Write_String("CLAVE ACTUAL");

            while(cont_key < 4)
```

```
            {
                tecla = Keypad_Get_Char(); // Lee el dato de la tecla presionada
                if(tecla != 0)  // Verifica si se ha presionado alguna tecla
                {
                    clave[cont_key] = tecla;   // Almacena cada tecla presionada en el arreglo
                    Lcd_Set_Cursor(2,7+cont_key);
                    Lcd_Write_Char('*'); // Muestra la tecla presionada.
                    cont_key++;  // Incrementa el contador
                }
            }
            WPUB = 0b11111110; // activar pull up en filas para deshabilitar teclado
            __delay_ms(200);
            Lcd_Clear();   // Limpia la pantalla lcd
            Leer_Clave(clave_enter); // Lee la clave almacenada en la memoria EEPROM

            if(!strcmp(clave, clave_enter)) // Compara si la clave es la correcta
            {
                Lcd_Set_Cursor(1,3);
                Lcd_Write_String("CLAVE NUEVA");
                cont_key = 0;

                WPUB = 0b00001110; // activar pull ups columnas, desactivar en filas para que teclado funcione

                while(cont_key < 4) {
                    tecla = Keypad_Get_Char();
```

```
                    if (tecla != 0) {
                        clave[cont_key] = tecla;
                        Lcd_Set_Cursor(2,7+cont_key);
                        Lcd_Write_Char('*');
                        cont_key++;
                    }
                }
                WPUB = 0bllllllll0; // activar pull up en filas para deshabilitar teclado
                __delay_ms(200);
                Lcd_Clear();   // Limpia la pantalla lcd
                Escribir_Clave(clave);

                Lcd_Set_Cursor(1,2);
                Lcd_Write_String("CLAVE CAMBIADA");
                Lcd_Set_Cursor(2,2);
                Lcd_Write_String("CORRECTAMENTE");
                LAT_LB = 1; // LED BLUE apagado
                LAT_LG = 0; // LED GREEN encendido
                __delay_ms(4000); // 4s
            } else {
                Lcd_Set_Cursor(1,6);
                Lcd_Write_String("CLAVE");
                Lcd_Set_Cursor(2,4);
                Lcd_Write_String("INCORRECTA");
                LAT_LB = 1; // LED BLUE apagado
                LAT_BUZ = 1; // BUZZER encendido
```

```
                    LAT_LR = 0; // encender LED RED
                    __delay_ms(3000); // 3s
                }
                cont_key = 0;          // Reinicia el contador
                estado = 0; // reiniciar estado
                Lcd_Clear();  // Limpia la pantalla lcd
                // columnas vuelven a estado original 0
                LATBbits.LATB1 = 0;
                LATBbits.LATB2 = 0;
                LATBbits.LATB3 = 0;
                // desactivar pull ups en filas y activar en columnas
                WPUB = 0b00001110;
                LAT_LR = 1; // LED RED apagado
                LAT_LG = 1; // LED GREEN apagado
                LAT_BUZ = 0; // BUZZER apagado
                IOCBF = 0; // reiniciar flag
                break;
            }
        }
        LAT_LB = 1; // LED BLUE apagado
        Lcd_Clear();
        // volver a mostrar mensaje inicial
        Lcd_Set_Cursor(1,2);
        Lcd_Write_String("ControlTech CA");
        INTCONbits.IOCIF = 0; // reiniciar flag interrupcion
    }
```

```
368
369   /********FUNCIONES EEPROM***************/
370   void Escribir_Clave(char* str) // funcion para guardar clave en la EEPROM
371   {
372       for(uint8_t pos=0; pos<4; pos++)
373       {
374           EEPROM_Write(pos, str[pos]);
375       }
376   }
377
378   void Leer_Clave(char* str) // funcion para leer la clave guardada en la EEPROM
379   {
380       for(uint8_t pos=0; pos<4; pos++)
381       {
382           str[pos] = (char)EEPROM_Read(pos);
383       }
384   }
```