# KOCAELİ UNIVERSITY ENGINEERING FACULTY


# DISTANCE MESUREMENT
# WITH BLUETOOTH SIGNALS

**Furkan YARDIMCI**
**2022-2023**

**DEPARTMENT: Electronic and Communacition Engineering**
**TEACHER: Dr. Lecturer Ayhan KÜÇÜKMANİSA**

**KOCAELİ**

## TABLE OF CONTENTS



## LIST OF IMAGES

## LIST OF TABLES

## PURPOSE

The purpose of this document is to provide a guide for the application of distance measurement via Bluetooth signal with MSP430 and HC-05.

## 1. İNTRO

With this document, you will have a strong document on how to measure distance with MSP430 and HC-05. There is more than one way to measure distance with Bluetooth signals, in this document we will examine the distance measurement with the RSSI (Received signal strength indication) value, which is the simplest to implement but gives an approximate distance value. To obtain the RSSI value with the HC-05, we will use AT commands, and to obtain the RSSI value and distance information, we will use a formula including environment variables.

We have a central circuit in the project, and after defining the address of the device whose distance we want to measure to the circuit, we can see the distance information continuously on the LCD display on the main circuit. In the next part, we will talk about the first components we have to use, their functions in the project, the code and schematic of the project.

## 2. COMPONENTS

### 2.1 MSP430

Texas Instruments MSP430G2553 is part of the MSP430 family of ultra-low-power microcontrollers featuring different sets of peripherals. The MSP430G2553 mixed signal microcontroller features a 16-bit Semi RISC CPU, 16-bit registers, 16KB non-volatile memory, 512 bytes RAM. The G2x53 series is popular due to the low price and various package sizes.

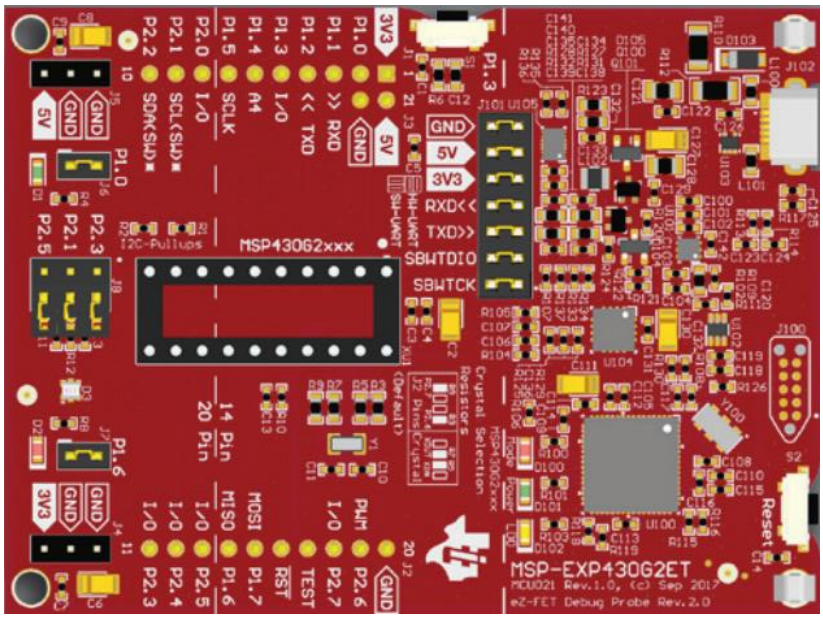



**Image 1: MSP430G2553**

## 2.2 HC-05 BLUETOOTH MODULE

HC-05 Bluetooth Module is an easy-to-use Bluetooth SPP (Serial Port Protocol) module, designed for transparent wireless serial connection setup. Its communication is via serial communication which makes an easy way to interface with controller or PC. HC-05 Bluetooth module provides switching mode between master and slave mode which means it able to use neither receiving nor transmitting data. Master, slave modes and lots of modes are, arranged with AT command of the HC-05, which is so important for this project. With AT commands we will receive RSSI value and process it to obtain measurement value.

### 2.2.1 AT COMMANDS

By default, the HC-05 is in data mode. In this mode, the module acts like a serial bridge. If you want to configure an HC-05 as you want you have to put it into AT command mode, to put into AT command mode the KEY pin (34) must be set high at start (Guide). The HC-05 is now in command mode if the red LED flashes once every two seconds. You can see all AT commands with this link.

We will use "AT+INQ" command, which gives us RSSI value and after inquiring the desired device AT+INQC should be used to close the inquire process. To use these commands first you need to configure your device. Your module must be in master mode, your connection mode must be 0 and finally you must pair and bind it with your desired Bluetooth signal source, which can be HC-05 or phone. You can check your device modes with "AT+ROLE?" and "AT+CMODE?" commands, and you can configure it with "AT+ROLE = 1", "AT+CMODE = 0", "AT+PAİR = address of the device", "AT+BIND = address of the device" these commands. After these configurations your module is ready. I used my phone as the signal source, which means this project will give me information about how far my phone from the circuit is.

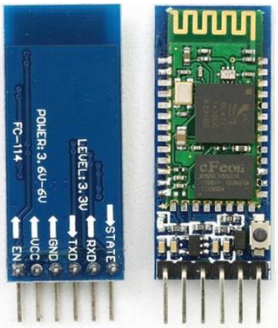



**Image 2: HC-05**

| | | |
|---|---|---|
| 1 | AT+ROLE=1 | Set module to master |
| 2 | AT+CMODE=0 | Connect the module to the specified Bluetooth address. |
| 3 | AT+PAIR=" address" | make pair with the remote Bluetooth |
| 4 | AT+BIND=" address" | Connect with the remote Bluetooth device |
| 5 | AT+INQ | Inquire Bluetooth device around |
| 6 | AT+INQC | Cancel inquire Bluetooth device |

**Table 1: Used AT Commands**

## 2.3 LCD DİSPLAY

LCD (Liquid Crystal Display) screen is an electronic display module and find a wide range of applications. A 16x2 LCD display is very basic module and is very commonly used in various devices and circuits. These modules are preferred over seven segments and other multi segment LEDs. The reasons being: LCDs are economical; easily programmable; have no limitation of displaying special & even custom characters (unlike in seven segments), animations and so on. A 16x2 LCD means it can display 16 characters per line and there are 2 such lines. In this LCD each character is displayed in 5x7 pixel matrix. This LCD has two registers, namely, Command and Data.

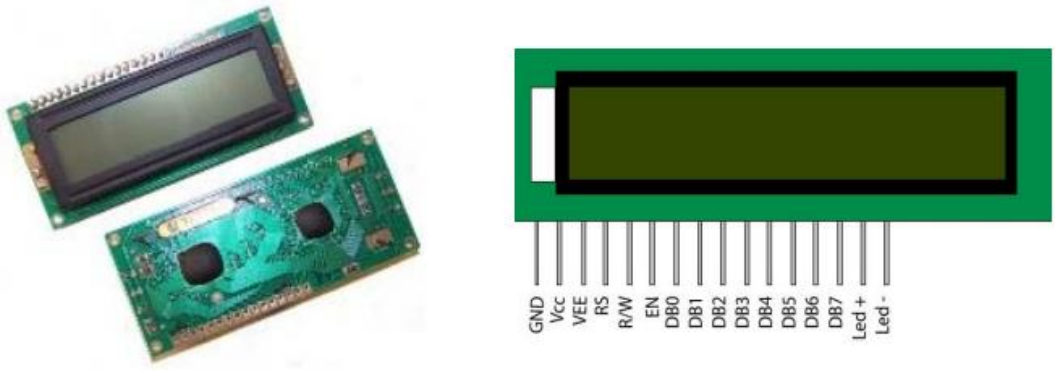



**Image 3: 2x16 LCD Display**

### 2.3.1 PIN CONFIGURATION

| SM PMOD Reference Designator | Function |
|---|---|
| Pin 1 ( Vss ) | Function as Ground Terminal. |
| Pin 2 ( Vcc ) | Function as Positive Supply ( 2.7V to 5.5V ). |
| Pin 3 ( Vdd ) | Function as Contrast adjustment ( Ground to Vcc ). |
| Pin 4 ( RS ) | Function as Register Select ( If 0 is refer to Instruction Register and if 1 is refer to Data Register ). |
| Pin 5 ( R/W ) | Its function to Read or Write Signal ( if 1 mean to Read and if 0 mean to Write ). |
| Pin 6 ( E ) | Function as Enable. |
| Pin 7 to Pin 14 ( DB0 – DB7 ) | : Refer to Bi-directional data bus, data transfer is performed one, thru DB0 to DB7, in this case of interface data length is 8- bits; and twice, through DB4 to DB7 in this case of interface data length is 4-bits ( Upper nibble first and then Lower nibble). |
| Pin 15 ( K ) | Function to Back light LED cathode terminal. |
| Pin 16 ( A ): | Function to Back light LED anode terminal. |

**Table 2: pins of LCD display**

### 2.3.2 Control and Display Commands

There is a character LCD controller of Hitachi company named HD44780 on most character LCDs available in the market. This controller acts as a bridge between the LCD and the microcontroller/FPGA. In other words, we do not directly interfere with the pixels on the LCD with the microcontroller/FPGA. We ensure that the characters we want are displayed through the controller. Since the HD44780 is a general-purpose controller, most character LCD manufacturers use this controller in their LCDs of various sizes and features.

3 bits are very important for sent data or instruction

**RS:** Register Select pin. In the logic 0 state, a command is sent to the HD44780 from the bus. If logic 1 is set, data is written or read from the data bus to the HDD780. It is necessary to write or receive commands to make adjustments to the HD44780, and to send or receive character data.

**R/W:** Read write pin. In the logic 1 state, reading is taken from HD44780. If the logic is 0, the HD44780 is written to. Since the HD44780 is generally written to, this pin is usually connected directly to gnd in applications.

**EN:** It is the enable pin. In case of logic 1, read-write operation is performed to HD44780. Cannot be done in the logic 0 state. D0:D7: The bus of HD44780 controller is 8 bits wide, used for reading, write operations.

To send instructions to LCD via controller, RS and R/W bits must be 0 and to send data RS must be 1. In this mode, we can send commands to LCD's instruction register. The commands are 8 bit wide and given in the list below.

| Sr.No. | Hex Code | Command to LCD instruction Register |
|---|---|---|
| 1 | 01 | Clear display screen |
| 2 | 02 | Return home |
| 3 | 04 | Decrement cursor (shift cursor to left) |
| 4 | 06 | Increment cursor (shift cursor to right) |
| 5 | 05 | Shift display right |
| 6 | 07 | Shift display left |
| 7 | 08 | Display off, cursor off |
| 8 | 0A | Display off, cursor on |
| 9 | 0C | Display on, cursor off |
| 10 | 0E | Display on, cursor blinking |
| 11 | 0F | Display on, cursor blinking |
| 12 | 10 | Shift cursor position to left |
| 13 | 14 | Shift the cursor position to the right |
| 14 | 18 | Shift the entire display to the left |
| 15 | 1C | Shift the entire display to the right |
| 16 | 80 | Force cursor to the beginning ( 1st line) |
| 17 | C0 | Force cursor to the beginning ( 2nd line) |
| 18 | 38 | 2 lines and 5×7 matrix (8bit mode) |
| 19 | 28 | 2 lines and 5×7 matrix (4bit mode) |

**Table 3: LCD instructions**

### 2.3.3 INITIALIZATION OF LCD

According to datasheet some process must be followed beginning of the device start to function properly. This process steps are given below. To send instructions RS bit should be 0 and to write R/W bit should be 0 as mentioned above and enable pin must be high at least 230ns to send or write any data to lcd. Waits are given due to this information.

| | | |
|---|---|---|
| 1 | EN = 0 | Wait 15ms or longer before enable the device. |
| 2 | EN = 1 | Send 0x3 for 240ns. |
| 3 | EN = 0 | Wait 4.1ms or longer. |
| 4 | EN = 1 | Send 0x3 for 240ns. |
| 5 | EN = 0 | Wait 100us or longer. |
| 6 | EN = 1 | Send 0x3 for 240ns. |
| 7 | EN = 0 | Wait 40us or longer. |
| 8 | EN = 1 | Send 0x2 for 240ns. |
| 9 | EN = 0 | Wait 40us or longer. |

**Table 4: LCD initialization**

After initialization process, we can send data or instruction to LCD. First, we will send instructions. Used instructions are listed below.

| | | |
|---|---|---|
| 1 | 0C | Display on, cursor off |
| 2 | 06 | Increment cursor (shift cursor to right) |
| 3 | 80 | Force cursor to the beginning ( 1st line) |
| 4 | 28 | 2 lines and 5×7 matrix (4bit mode) |

**Table 5: LCD instructions used**

### 2.3.4 USAGE OF 4-BİT MODE

8 bit data length is expensive in terms of pin usage so LCD used in 4 bit mode in this project. To do this 0x28 instruction must be send to LCD when RS and R/W bits are 0. In this mode, data is sent in nibbles. The msb nibble must be sent first, followed by the lsb nibble. Regardless of whether the information sent is data or instructions, a certain process must be followed. After sending msb nibble, wait 1 microsecond and then send lsb nibble and wait at least 40 microseconds at the end of each 8-bit data packet. This process must be followed to send instruction or data to LCD.

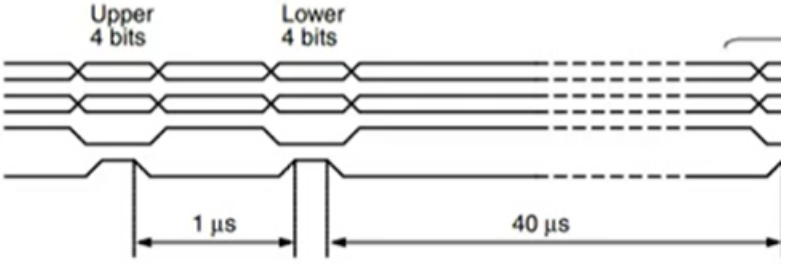



**Image 4: 4-bit mode working process**

### 2.3.5 ALPHA NUMERİC TABLE

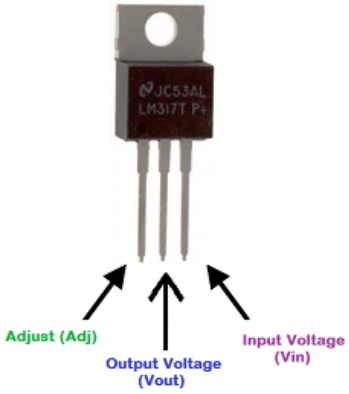
The 2x16 LCD display has the power to write 2x16=32 characters in total. It uses a 5x7 matrix to write each character to the screen. The upper and lower nibble values of these characters are given below. To be able to read these characters on the screen, RS = 1 and R/W = 0 and when RS = 1, we send data to LCD's data register as mentioned early.


**Img 5: Alphanumeric table**

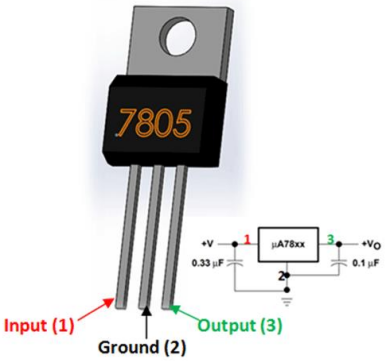
## 2.4 REGULATORS

A voltage regulator is **a component of the power supply unit that ensures a steady constant voltage supply through all operational conditions**. It regulates voltage during power fluctuations and variations in loads. It can regulate AC as well as DC voltages.

MSP needs 3.3 volts to operate, LCD display and HC-05 5 volts. In the project, LM317T and LM785 regulators were used to produce 3.3v and 5v, respectively.




**Img 6: LM317T**




**Img 7: LM7805**

## 3. PROCESSING LOGIC

First, let's talk about the general working logic. Each Bluetooth device has its own unique address. The HC-05 returns a set of information with the AT+INQ command described above. For example, if we examine the following information, "+INQ:2:72:D2224,3E0104,FFBC" we first see the address of the device, then the type of the device, and finally the RSSI value. Since we measure distance with RSSI value, we need to make this information hand-processable somehow. To perform this operation, I created a series of functions that I will talk about later. Thanks to these functions, I extract the RSSI value from the information in the form of a string, convert it to a hexadecimal and then a decimal number, and finally, I apply the necessary formula on it and generate the distance information. The formula is given below.

$$Distance = 100x10^{n}\ [cm], \qquad \text{n} = \frac{\text{refdb} - \text{currentdb}}{10\text{x}(envconstant)}$$

The ambient constant is a quantity that changes according to the environment and gives information about the parameters such as attenuation and reflection of the signal. Refdb represents the average dB power at 1m distance. With reference to this dB value, other measurement results are calculated.

## 3.1 PROCESSİNG STEPS AND CODE

When the circuit starts to work for the first time, the units connected to the circuit such as HC-05, LCD, MSP430 are initialized. The starting operations of HC-05 and LCD were previously explained above. We perform this operation with the WD_OSCAL_AND_INITS() function. In this function, the initial values of the registers for wd, oscillator, lcd, hc-05 and LEDs are set. Then the program goes into infinite loops. In the first loop, it does INQ and INQC operations until it finds the mac address of my phone. Detecting my device is done with the ft_strnstr() function. This function searches for a substring in a string and returns the starting address if it finds that string and returns NULL otherwise. After the device is found, it takes the RSSI value as a string from the returned string with the Get_RSSI() function, and then converts this value to a numerical value, processes it and writes the distance information to the screen with LCD.h functions. The whole program is given below

### 3.1.1 MAIN.C

```
#include "muhtas1.h"

int     i;
char    BUFF[80]={0};

int     main( void )
{
 int     flag=0;
 char    *HexRssi;
 WD_OSCAL_AND_INITS();

 while (1)
 {
  while (!(flag = ft_strnstr(BUFF, "1819:D6:8BC2F9", 80)))
  {
   i = 0;
   TX("AT+INQ\r\n");
   __delay_cycles(1500000);
   TX("AT+INQC\r\n");
  }
  LCD_Temizle();
  LCD_Yazi_Yaz("Distance:");
  LCD_Git_XY(1,10);
  HexRssi = Get_RSSI(BUFF, flag);
  HexRssi = Calculate_Dist(HexRssi);
  LCD_Yazi_Yaz(HexRssi);
  Re_Arrange(BUFF,HexRssi,&flag);
 }
}

#pragma vector=USCIAB0RX_VECTOR
```

```
__interrupt void RX_ISR(void)
{
   BUFF[i] = UCA0RXBUF;
   i++;
}
```

### 3.1.2 LIBFT.H

```
#ifndef LIBFT_H
#define LIBFT_H

#include "math.h"
#include "stdlib.h"

int     ft_strnstr(const char *haystack, const char *needle, int len);
int       ft_strlen(const char *s);
int     ft_strncmp(const char *s1, const char *s2, int n);
int       ft_strlcpy(char *dst, char *src, int dstsize);
void     *ft_memset(void *b, int c, int len);

int     ft_hex2_dec(const char *s);
char    *ft_ftoa(float Dist);
#endif
```

### 3.1.3 LIBFT.C

```
#include "libft.h"

int       ft_strlen(const char *s)
{
          int       i;

          i = 0;
          while (*(s + i))
                    i++;
          return (i);
}

int ft_strnstr(const char *haystack, const char *needle, int len)
{
          unsigned int    counter;
          unsigned int    len_needle;

          len_needle = ft_strlen(needle);
          if (len_needle == 0)
                    return (0);
          if (len == 0)
                    return (0);
          counter = 0;
          while (counter <= (len - len_needle) && haystack[counter] != '\0')
```

```
	{
		if (ft_strncmp(&haystack[counter], needle, len_needle) == 0)
			return (counter);
		counter++;
	}
	return (0);
}

int	ft_strncmp(const char *s1, const char *s2, int n)
{
	int		dif;
	int	i;

	i = 0;
	dif = 0;
	if (n == 0)
		return (0);
	while ((s1[i] || s2[i]) && i < n)
	{
		dif = (unsigned char)s1[i] - (unsigned char)s2[i];
		if (dif != 0)
			break ;
		i++;
	}
	return (dif);
}

int	ft_strlcpy(char *dst, char *src, int dstsize)
{
	int	i;
	int	src_size;

	i = 0;
	src_size = ft_strlen(src);
	if (dstsize)
	{
		while (src[i] && i < (dstsize - 1))
		{
			dst[i] = src[i];
			i++;
		}
		dst[i] = '\0';
	}
	return (src_size);
}

void	*ft_memset(void *b, int c, int len)
{
	int	i;
```

```
        i = 0;
        while (i < len)
        {
                ((unsigned char *)b)[i] = (unsigned char)c;
                i++;
        }
        return (b);
}

int     ft_hex2_dec(const char *s)
{
  unsigned int   num = 0;
  int   i = -1;

  while (++i < 4)
  {
    if(*s-48 < 10)
      num = (*s-48) + num * 16;
    else
      num = (*s-55) + num * 16;
    s++;
  }
  return (num);
}

char    *ft_ftoa(float Dist)
{
  char *array = (char *)malloc(sizeof(char)*8);
  int num = 0;
  int i = 0;

  num = (int)Dist;
  while(i<3)
  {
    array[2-i]=num % 10 + 48;
    num /= 10;
    i++;
  }
  num = (int)(fmod(Dist,1)*1000);
  array[3] = '.';
  i = 0;
  while(i<3)
  {
    array[6-i]=num%10 + 48;
    num /= 10;
    i++;
  }
  array[7]=0;
  return (array);
}
```

### 3.1.4 MUHTAS1.H

```
#ifndef MUHTAS1_H
#define MUHTAS1_H

#include "io430.h"
#include "intrinsics.h"
#include "stdlib.h"
#include "libft.h"
#include "math.h"
#include "LCD.h"

#define GREENL  BIT0
#define YELLOWL BIT3
#define REDL    BIT0

void    WD_OSCAL_AND_INITS(void);
void    Uart_Init(void);
void    Re_Arrange(char *a, char *Device_val, int *flag);
void    Led_Control(float distance);

void    TX(const char *s);
void    putc(const unsigned c);

char    *Get_RSSI(char *a, int index);
char    *Calculate_Dist(char *HexRssi);

#endif
```

### 3.1.5 MUHTAS1.C

```
#include "muhtas1.h"

void    WD_OSCAL_AND_INITS(void)
{
  WDTCTL = WDTPW + WDTHOLD;
  LCD_Ayarla();
  P1DIR |= GREENL+YELLOWL;
  P2DIR |= REDL;
   P1OUT &= ~(GREENL+YELLOWL);
  P2OUT |= REDL;
  if (CALBC1_1MHZ == 0xFF)
  {
    while (1);
  }
  DCOCTL = 0;
  BCSCTL1 = CALBC1_1MHZ;
  DCOCTL = CALDCO_1MHZ;
  Uart_Init();
```

```
  __bis_SR_register(GIE);
}

void    Uart_Init(void)
{
  P1SEL = BIT1 + BIT2;
  P1SEL2 = BIT1 + BIT2;
  UCA0CTL1 |= UCSSEL_2;
  UCA0BR0 = 26;
  UCA0BR1 = 0;
  UCA0MCTL = UCBRF_0 + UCBRS_0;
  UCA0CTL1 &=~ UCSWRST;
  IE2 |= UCA0RXIE;
}

void    putc(const unsigned c)
{
  while (!(IFG2&UCA0TXIFG));
    UCA0TXBUF = c;
}

void    TX(const char *s)
{
  while (*s)
    putc(*s++);
}

char    *Get_RSSI(char *a, int index)
{
  char *b = (char *)malloc(sizeof(char)*5);
  if(!b)
      return NULL;
    ft_strlcpy(b, a + index + 22, 5);
    return (b);
}

void    Re_Arrange(char *a, char *Device_val, int *flag)
{
   *flag=0;
   ft_memset(a,0,80);
   free(Device_val);
}

char    *Calculate_Dist(char *HexRssi)
{
  int ref_db = -55;
  int current_db;
  float n=0;
  float distance;
```

```
  current_db = ft_hex2_dec(HexRssi)-65535;
  free(HexRssi);
  n = (ref_db - current_db) / (10*1.8);
  distance = 100*pow(10,n);
  Led_Control(distance);
  return (ft_ftoa(distance));
}

void    Led_Control(float distance)
{
  if (distance < 100)
  {
    P1OUT&=~GREENL;
    P1OUT&=~YELLOWL;
    P2OUT|=REDL;
  }
  else if (distance >= 100 && distance < 200)
  {
    P1OUT&=~GREENL;
    P1OUT|=YELLOWL;
    P2OUT&=~REDL;
  }
  else
  {
    P1OUT|=GREENL;
    P1OUT&=~YELLOWL;
    P2OUT&=~REDL;
  }
}
```

### 3.1.6 LCD.H

```
#ifndef __LCD_H
#define __LCD_H

#ifdef __cplusplus
 extern "C" {
#endif

#define LCD_DATA_PORT                   P1OUT
#define LCD_DATA_PORT_DIR               P1DIR
#define LCD_DATA_PORT_SEL               P1SEL
#define LCD_DATA_PORT_SEL2              P1SEL2
#define LCD_CONTROL_PORT              P2OUT
#define LCD_CONTROL_PORT_DIR            P2DIR
#define LCD_CONTROL_PORT_SEL              P2SEL
#define LCD_CONTROL_PORT_SEL2             P2SEL2
#define LCD_RS_BIT                      BIT7
#define LCD_EN_BIT                      BIT6
```

```
#defineEN_AC_KAPA()    LCD_EN(1),LCD_EN(0)
#defineLCD_RS(x)           ( (x) ? (LCD_CONTROL_PORT |= LCD_RS_BIT) :
(LCD_CONTROL_PORT &= ~LCD_RS_BIT) )
#define  LCD_EN(x)          ( (x) ? (LCD_CONTROL_PORT |= LCD_EN_BIT) :
(LCD_CONTROL_PORT &= ~LCD_EN_BIT) )

void LCD_Komut_Yaz(unsigned char);
void LCD_Temizle(void);
void LCD_Yazi_Yaz(const char*);
void LCD_Git_XY(char,char);
void LCD_Ayarla(void);
void LCD_Karakter_Yaz(char);

#ifdef __cplusplus
}
#endif

#endif /*__LCD_H */
```

### 3.1.7 LCD.C

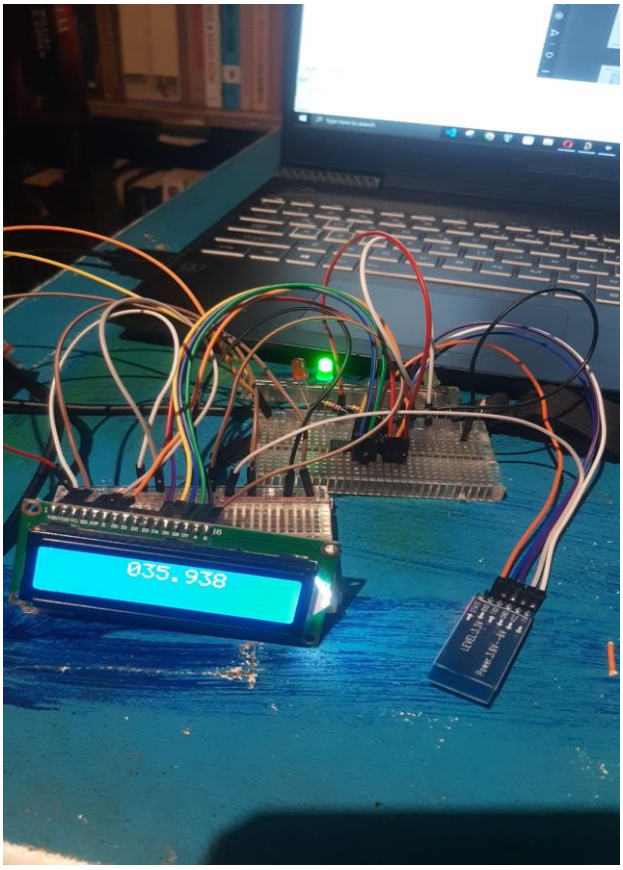
```
#include "io430.h"
#include "LCD.h"
void LCD_Karakter_Yaz(char veri)
{
 unsigned char bSayac;
 LCD_RS(1);
 for(bSayac=0;bSayac<40;bSayac++);
 LCD_DATA_PORT = (LCD_DATA_PORT & 0x0F)|((veri) & 0xF0);
 EN_AC_KAPA();
 LCD_DATA_PORT = (LCD_DATA_PORT & 0x0F)|(veri<<4 & 0xF0);
 EN_AC_KAPA();
}
void LCD_Komut_Yaz(unsigned char komut)
{
 unsigned char bSayac;
 LCD_RS(0);
 for(bSayac=0;bSayac<40;bSayac++);
 LCD_DATA_PORT = (LCD_DATA_PORT & 0x0F)|((komut) & 0xF0);
 EN_AC_KAPA();
 LCD_DATA_PORT = (LCD_DATA_PORT & 0x0F)|(komut<<4 & 0xF0);
 EN_AC_KAPA();
}
void LCD_Temizle(void)
{
 unsigned int  bSayac;
 LCD_Komut_Yaz(0x01);
 for(bSayac=0;bSayac<10000;bSayac++);
}
void LCD_Yazi_Yaz(const char* yazi)
```

```
{
 while(*yazi)
 LCD_Karakter_Yaz(*yazi++);
}
void LCD_Git_XY(char x, char y)
{
 if(x==1)
 LCD_Komut_Yaz(0x80+((y-1)%16));
 else
 LCD_Komut_Yaz(0xC0+((y-1)%16));
}
void LCD_Ayarla()
{
 unsigned int bSayac1,bSayac2;
 LCD_DATA_PORT_DIR      |=    BIT4 + BIT5 + BIT6 + BIT7;
 LCD_DATA_PORT_SEL      &= ~(BIT4 + BIT5 + BIT6 + BIT7);
 LCD_DATA_PORT_SEL2  &= ~(BIT4 + BIT5 + BIT6 + BIT7);

 LCD_CONTROL_PORT_DIR   |=    LCD_EN_BIT + LCD_RS_BIT;
 LCD_CONTROL_PORT_SEL   &= ~(LCD_EN_BIT + LCD_RS_BIT);
 LCD_CONTROL_PORT_SEL2  &= ~(LCD_EN_BIT + LCD_RS_BIT);

 LCD_DATA_PORT = 0x0F;
 LCD_CONTROL_PORT = 0x00;
 for(bSayac1=0;bSayac1<1000;bSayac1++)
 for(bSayac2=0;bSayac2<500;bSayac2++);
 LCD_RS(0);
 LCD_EN(0);
 for(bSayac1=0;bSayac1<20000;bSayac1++);
 LCD_Komut_Yaz(0x28);
 LCD_Komut_Yaz(0x0C);
 LCD_Komut_Yaz(0x06);
 LCD_Komut_Yaz(0x80);
 LCD_Komut_Yaz(0x28);
 LCD_Temizle();
}
```

# 4. PCB AND SCHEMATİC

## 4.1 PRELIMINARY

Before printing the pcb, I set up the project on the bread board and observed that it was working.




**IMG 8: Project on breadboard**

## 4.2 PCB MAKİNG PROCESS

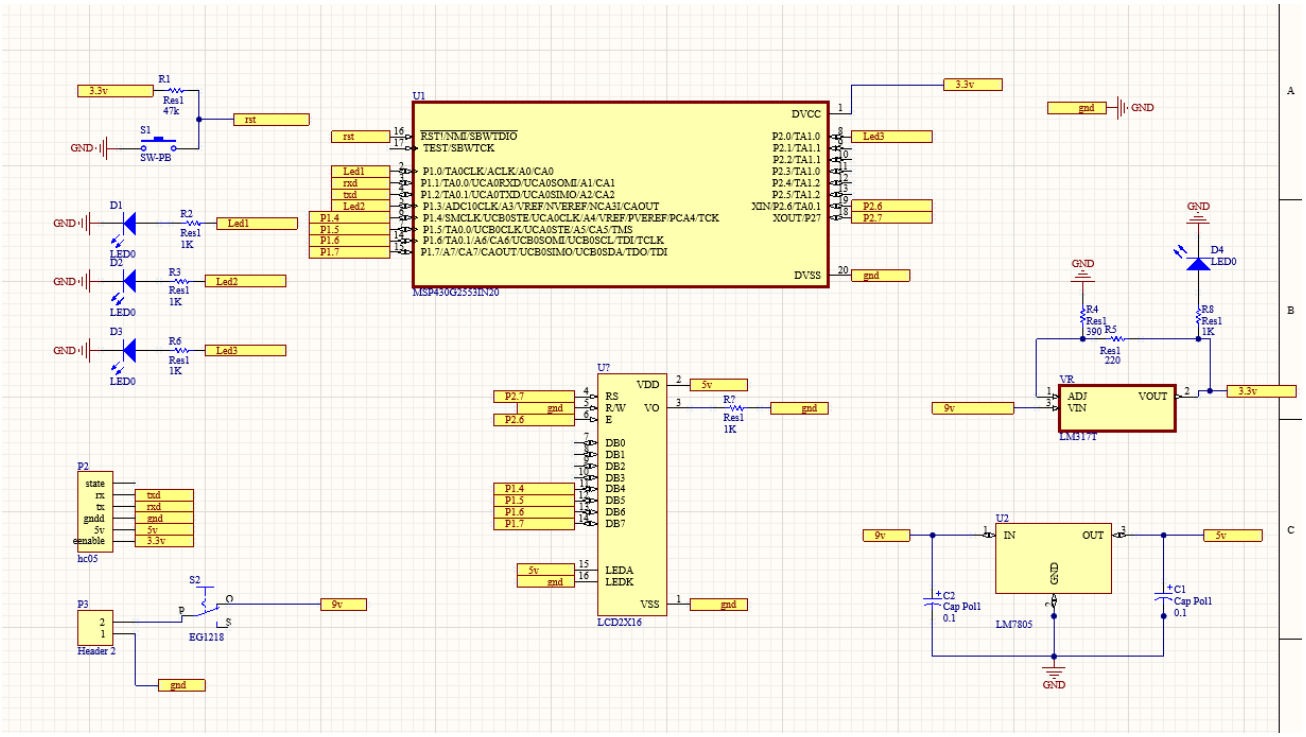
CONSTRUCTION OF THE PRINT CIRCUIT Materials needed for the printed circuit board

- Double-sided copper plate
- Coated (oil paper)
- Iron - Dish sponge
- Acetate pen
- Perhydrol acid
- Salt Spirit
- A plastic container to put the plaque in - Glove

After setting the materials required for printed circuit boarding, we can immediately proceed to the processing steps. In this context, as the first step, I printed the reverse image of the printed circuit drawing prepared with the help of the computer printed circuit drawing program, namely Altium, on glossy paper from the laser printer. The point to be considered in this step is the quality of the print. For this, the toner must be

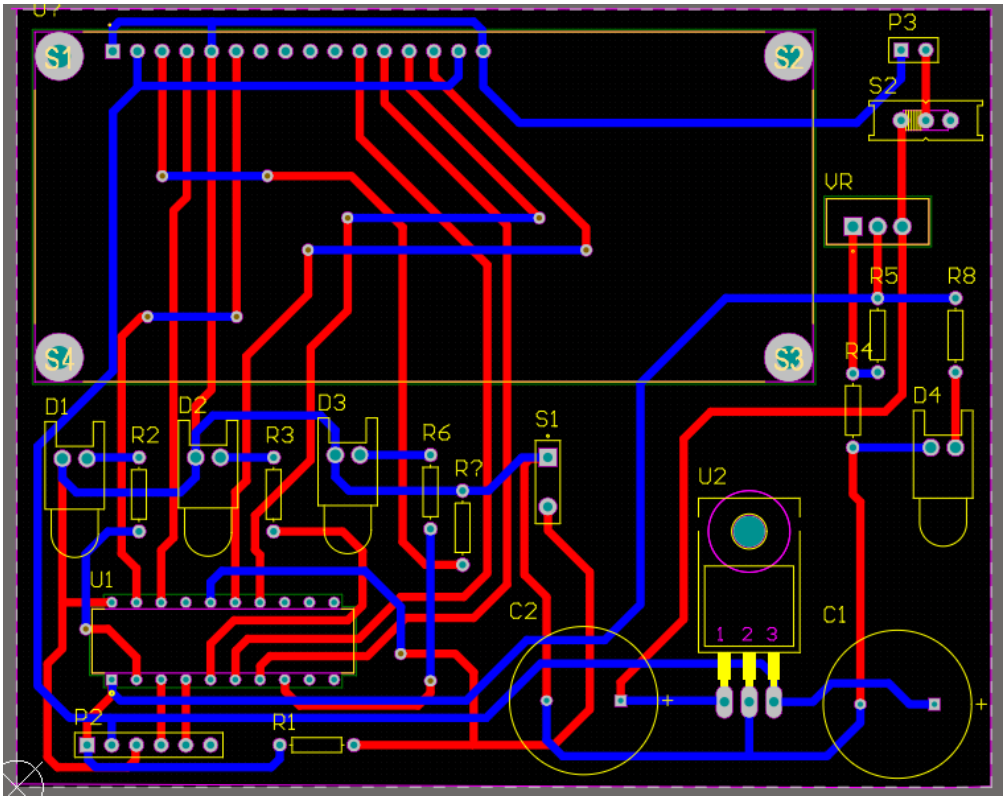
of good quality. As the second step in the construction of the printed circuit board, the copper plate prepared in accordance with the size of the circuit diagram is cleaned with detergent and a dish sponge. In the third step, the front side of the print out on oiled paper is placed on the front of the copper plate. After insertion, the edges can be folded to the back of the copper plate. Here it is very important to align the plaque and paper well. One of the most difficult steps when making a printed circuit board is ironing. At this stage, it will be necessary to process the iron without adding water. Otherwise, the paper may get wet. For the ironing process, the iron is set to the cotton setting and ironed for 5-10 minutes. After we run the iron on the plate with pressure, we should finish the ironing work without giving the plate a chance to swell. After waiting a few minutes for the copper plate to cool, we carefully remove it. At this stage, we must be very careful not to damage the roads. The printed circuit diagram is compared with the existing printed circuit and the missing parts are filled with a printed circuit pen (transparency pen). Likewise, if there are short circuits and unnecessary places, we separate them and bring them to a more complete form. Another issue that we should pay attention to in the production of printed circuit boards is that the printed circuit board should not be kept in acid for too long. It would be beneficial not to look inside the container because harmful gases are released when the plaque is thrown into the acid-filled plastic container. It will be sufficient to keep the copper plate in an acidic container for 5-10 minutes. Slight shaking of the container during this time will speed up the process. As a result of the observations and after the paths are revealed on the copper plaque, it is removed from the container and cleaned with the help of a dish sponge and detergent. Finally, the plate is drilled with a printed circuit drill and the process is terminated.

### 4.3 SCHEMATİC



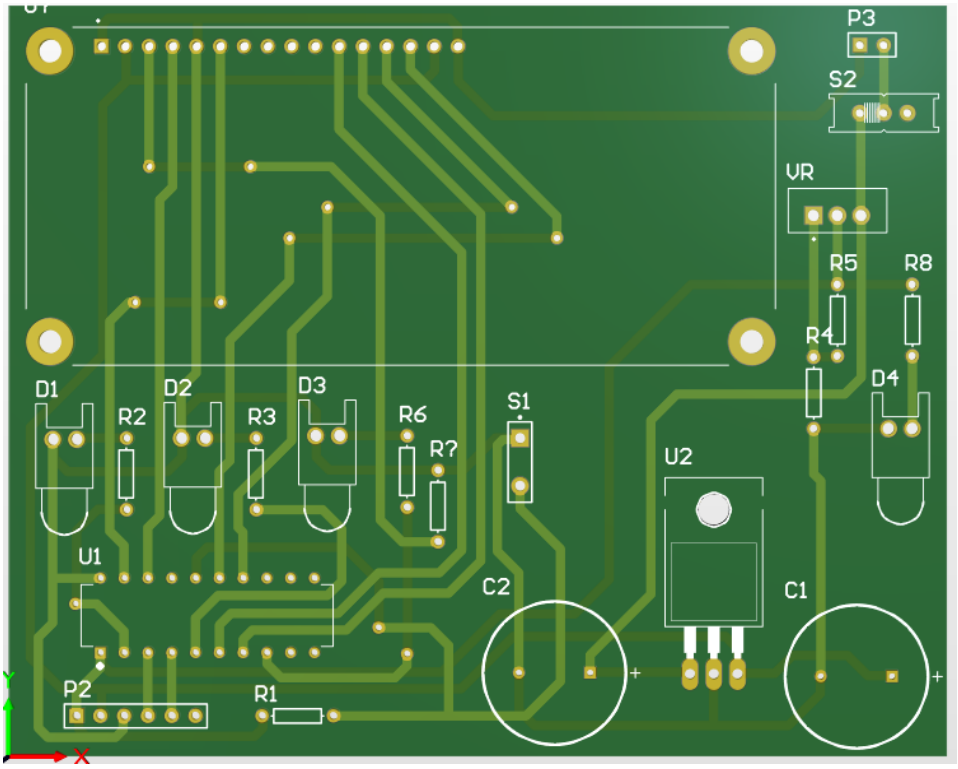

**Img 9: Schematic of project**

### 4.4 PCB



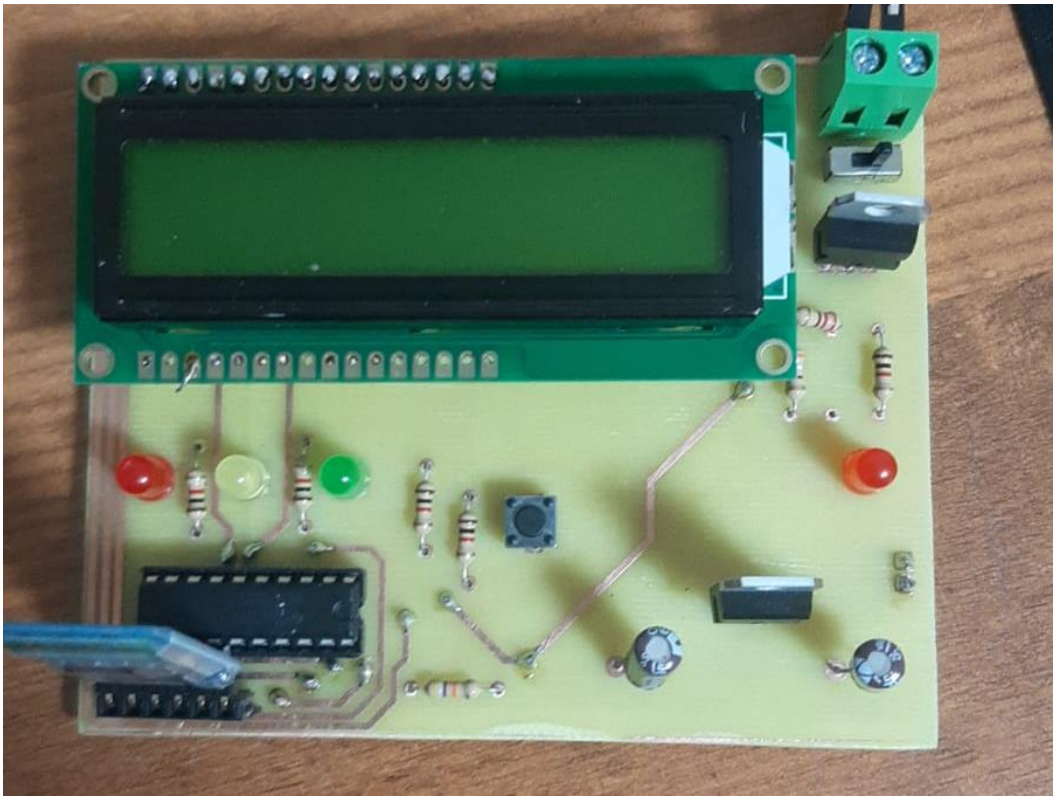

**Img 10: PCB of the project**

### 4.5 3D LAYOUT MODE

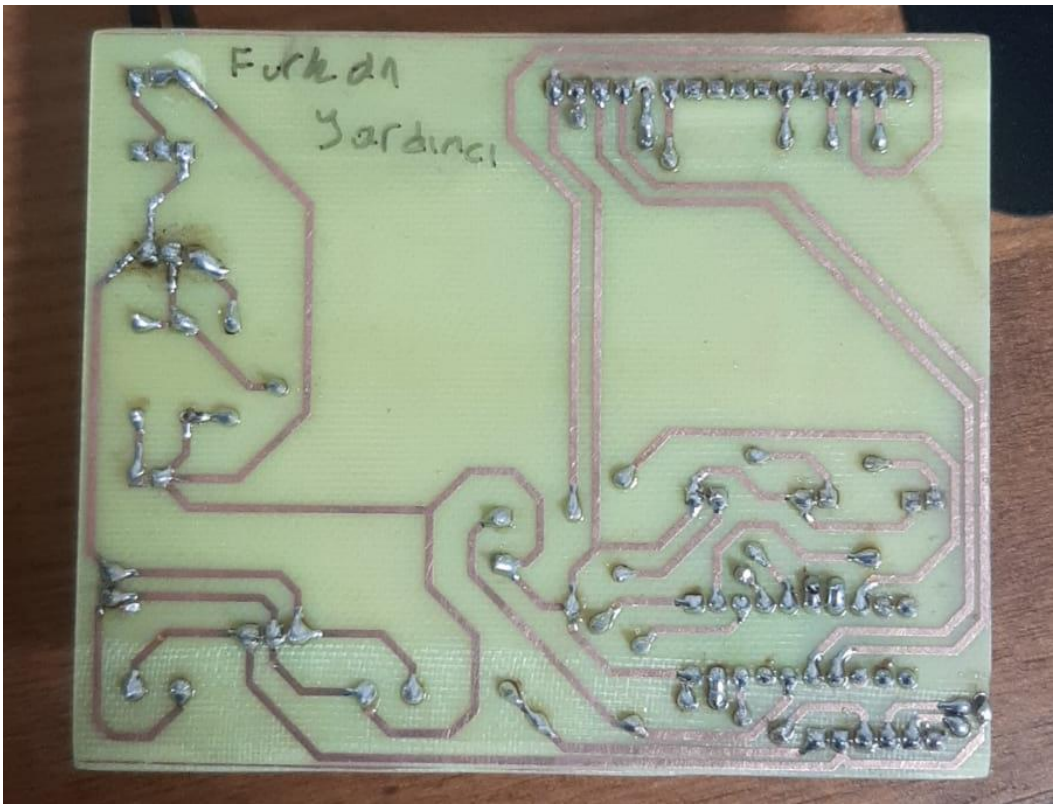



**Img 11: 3D wiev of the project**

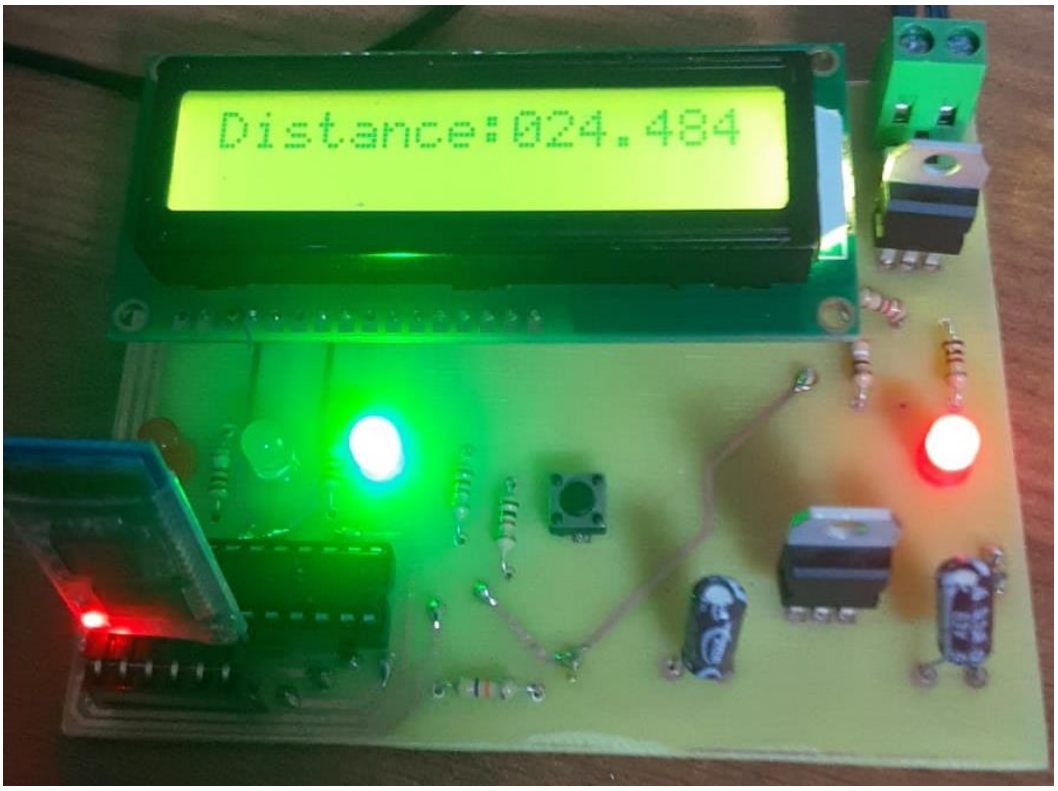
## 4.6 FINISHED WIEV OF PROJECT



**Img 12: Front view**




**Img 13: Back view**




**Img 14: While working**

## 5. REFERENCES

**https://www.ti.com/lit/ug/swmu005/swmu005.pdf?ts=1671400775862&ref_url=https%253A%252F%252Fwww.google.com%252F**

**https://www.youtube.com/watch?v=8YYZVVcnVpM&t=141s**

**https://linuxhint.com/interface-lcd-4-bit-8-bit-modes-arduino/**

**https://electronics.stackexchange.com/questions/180962/hc-05-bluetooth-atinq-command-not-working**

**https://www.electronicsforu.com/technology-trends/learn-electronics/16x2-lcd-pinout-diagram**

**https://s3-sa-east-1.amazonaws.com/robocore-lojavirtual/709/HC-05_ATCommandSet.pdf**

**https://www.youtube.com/watch?v=wc74sfv1Xpc**

**https://www.emo.org.tr/ekler/bac42456e18a005_ek.pdf**