

گزارش تمرین چهارم

درس ابزار دقیق

دکتر نیروی

رضا مومنی

810199497

انتقال داده	کاربرد	صحت سنجی دیتا	Max Bitrate	نام پروتکل
دو طرفه است	ارتباط نقطه ای	صحت سنجی دیتا را پشتیبانی نمیکند	115.2kbps	UART
چند مستر و اسلیو	سنسورها و EEPROM فاصله کم	صحت سنجی دیتا دارد و دیتای قابل اعتمادی را تضمین میکند	5000kbps	I2C
MASTER SLAVE	SCADA PLC	پشتیبانی نمیکند	115.2kbps	Modbus
چند مستر و چند نود باس	Automotive systems (ECUs), industrial automation, and robotics	CAN includes error-checking mechanisms (CRC) for robustness	1000kbps	CAN

#### 1. UART (Universal Asynchronous Reception and Transmission):

- **Speed:** UART is full-duplex communication and generally faster than I2C. It can reach speeds of a few Mbps.
- **Data Validation:** UART does not inherently provide data validation; it relies on higher-level protocols or application-specific mechanisms.
- **Data Transfer:** Bi-directional asynchronous serial data transmission using two lines (TX and RX).
- **Precision:** No inherent precision; depends on the application.
- **Applications:** Commonly used for point-to-point communication (e.g., USB to a computer, sensor modules).

#### 2. I2C (Inter-Integrated Circuit):

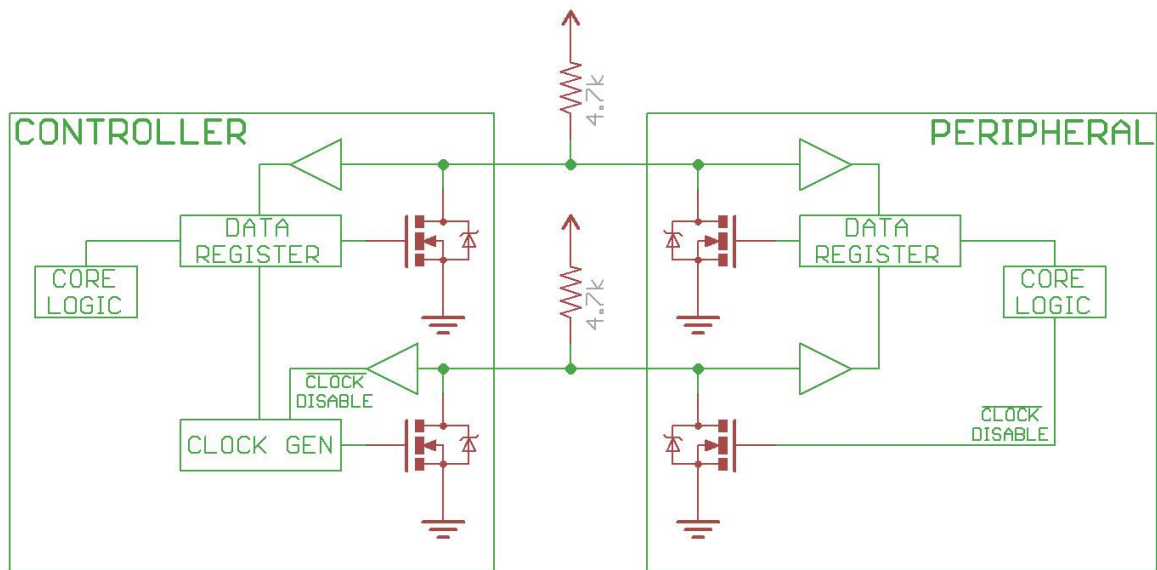
- **Speed:** I2C offers moderate data rates 100 kbps to 5 Mbps.
- **Data Validation:** I2C performs data validation, ensuring reliable data transfer.
- **Data Transfer:** Multi-master, multi-slave bus with flow control.
- **Precision:** Up to 3.4 MHz speed; suitable for precision applications.
- **Applications:** Used for connecting sensors, EEPROMs, and other peripherals within short distances.

#### 3. CAN (Controller Area Network):

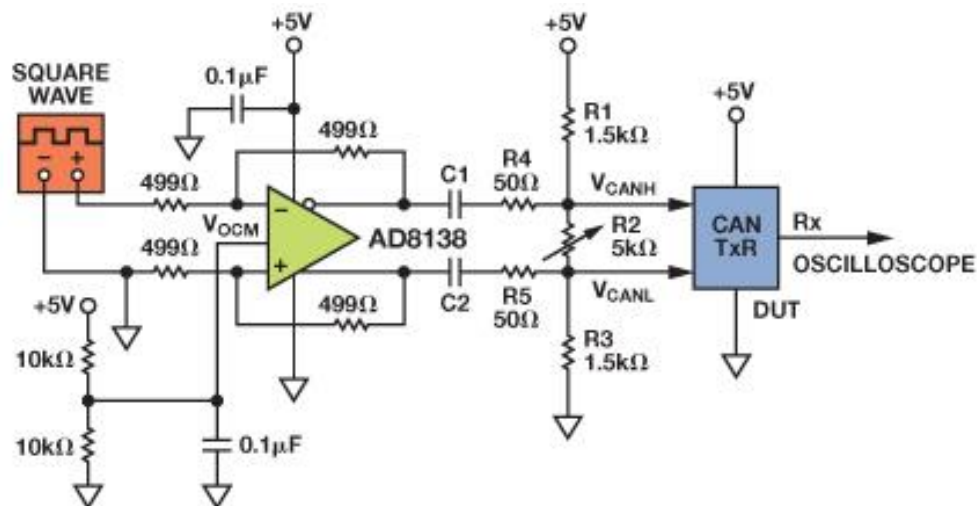
- **Speed:** CAN Bus data rates vary (low to high), with maximum speeds of 1 Mbps.
- **Data Validation:** CAN includes error-checking mechanisms (CRC) for robustness.
- **Data Transfer:** Multi-master, multi-node bus for industrial automation and automotive applications.
- **Precision:** Designed for reliability and fault tolerance.
- **Applications:** Automotive systems (ECUs), industrial automation, and robotics.

#### 4. Modbus:

- **Speed:** Modbus speed varies based on the implementation (typically slower than UART or I2C).
- **Data Validation:** Modbus lacks built-in data validation; error checking is application-specific.
- **Data Transfer:** Simple master-slave protocol (RTU or ASCII).
- **Precision:** Typically used for control and monitoring, not high precision.
- **Applications:** SCADA systems, PLCs, and industrial control.



## I2C circuit



## CAN circuit

## SIMULATION

### Question 1

810199497 % 3 = 0    =>    MC: STM32F103C6

SID % 9 = 3    =>    HCLK: 26 MHz                    |                    X1 = PA2, X2 = PA9                    |                    Y1 = PB0, Y2 = PB5

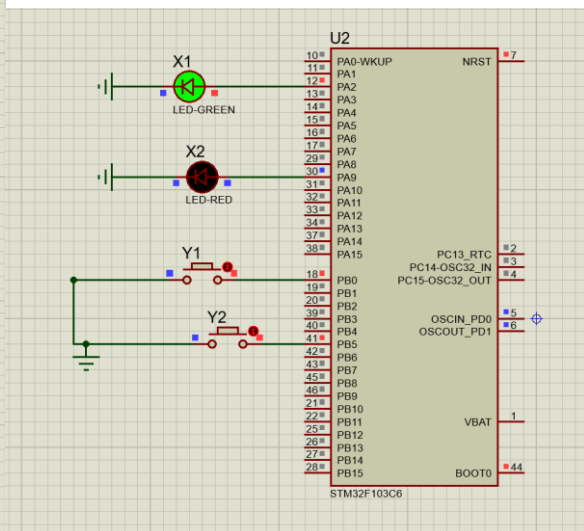
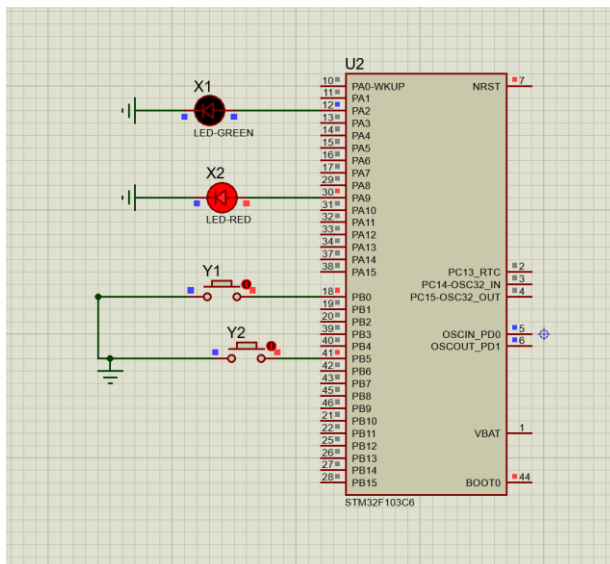
I.

بدنه اصلی کد که در while درون main نوشته شده است:

```

96   while (1)
97   {
98       /* USER CODE END WHILE */
99       led1 = HAL_GPIO_ReadPin(GPIOB, Y1_Pin);
100      led2 = HAL_GPIO_ReadPin(GPIOB, Y2_Pin);
101
102      if (led1==0){
103          HAL_GPIO_WritePin(GPIOA, x1_Pin, GPIO_PIN_SET);
104          HAL_GPIO_WritePin(GPIOA, x2_Pin, GPIO_PIN_RESET);
105      }
106
107      if (led2==0){
108          HAL_GPIO_WritePin(GPIOA, x2_Pin, GPIO_PIN_SET);
109          HAL_GPIO_WritePin(GPIOA, x1_Pin, GPIO_PIN_RESET);
110      }
111      //HAL_Delay(500);
112      /* USER CODE BEGIN 3 */
113      /* USER CODE BEGIN 3 */
114  }
115      /* USER CODE END 3 */
116  }
117

```



II.

```

59 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
60 {
61     //if (GPIO_Pin == Y2_Pin){
62         led2 = 0;
63     //}
64 }

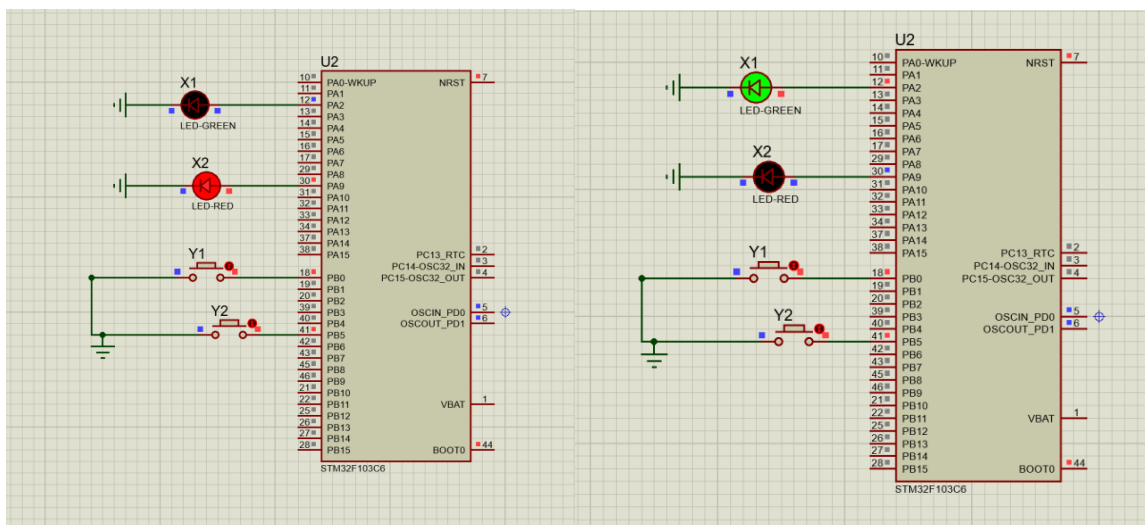
```

```

102 while (1)
103 {
104     /* USER CODE END WHILE */
105     led1 = HAL_GPIO_ReadPin(GPIOB, Y1_Pin);
106
107     if (led1==0){
108         HAL_GPIO_WritePin(GPIOA, x1_Pin, GPIO_PIN_SET);
109         HAL_GPIO_WritePin(GPIOA, x2_Pin, GPIO_PIN_RESET);
110     }
111
112     if (led2==0){
113         HAL_GPIO_WritePin(GPIOA, x2_Pin, GPIO_PIN_SET);
114         HAL_GPIO_WritePin(GPIOA, x1_Pin, GPIO_PIN_RESET);
115     }
116     led2=1;
117     /* USER CODE BEGIN 3 */
118     /* USER CODE BEGIN 3 */
119 }

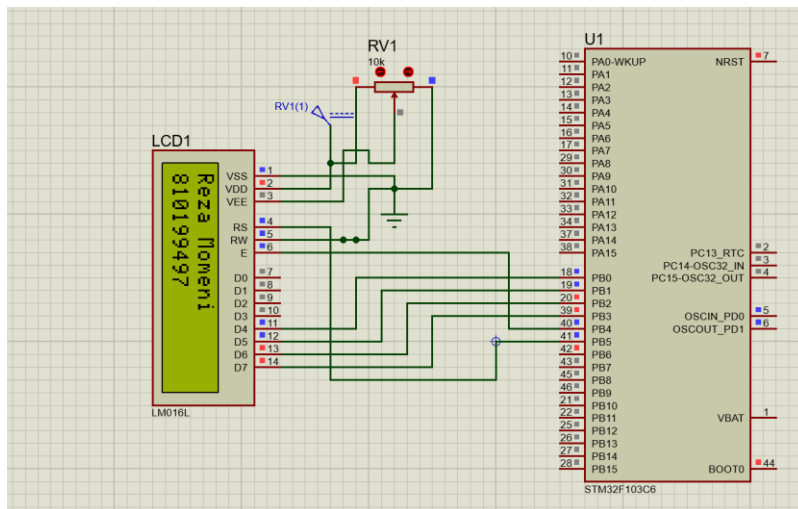
```

مزیت اصلی که اینترپت نسبت به مستقیم خواندن مقدار دارد این است که با حذف دیلی هایی ناشی از توابعی مثل `hal_gpio_readpin` میتوان به سرعت بیشتری دست پیدا کرد. همچنین اینترپت با کلاک اصلی خود میکرو کار میکند و نیازی به دیلی خارجی ندارد.



## Question 2

I.

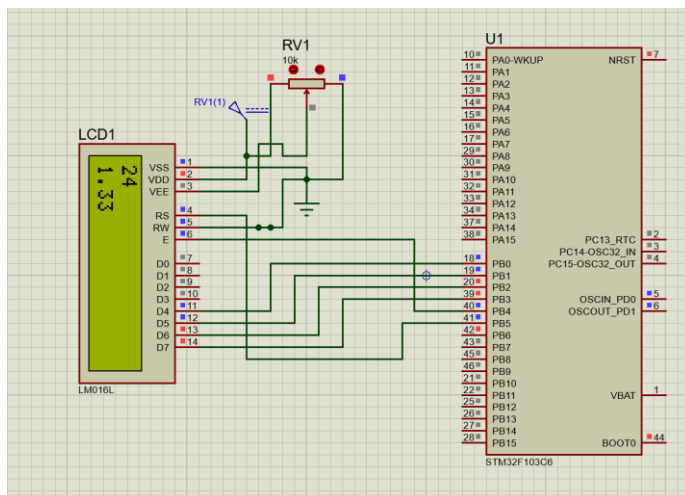


```

83  SystemClock_Config();
84  MX_GPIO_Init();
85
86  LCD16X2_Init(MyLCD);
87  LCD16X2_Clear(MyLCD);
88  LCD16X2_Set_Cursor(MyLCD, 1, 1);
89  LCD16X2_Write_String(MyLCD, "Reza Momeni");
90  LCD16X2_Set_Cursor(MyLCD, 2, 1);
91  LCD16X2_Write_String(MyLCD, "810199497");
92  /* USER CODE BEGIN SysInit */

```

II.



از آنجایی که مستقیماً نمیتوان اعداد اعشاری را نمایش داد پس با کد زیر میتوان به خروجی بالا دست پیدا کرد. یعنی ما اجزای عدد اعشاری را به صحیح تبدیل کنیم و از آن استفاده کنیم.

```
92     main = f;
93     dec = (f*100);
94     dec = dec%100;
95
96
97
98
99     LCD16X2_Init(MyLCD);
100    LCD16X2_Clear(MyLCD);
101
102    LCD16X2_Set_Cursor(MyLCD, 2, 1);
103    sprintf(fl, "%d", main);
104    LCD16X2_Write_String(MyLCD, fl);
105    LCD16X2_Write_Char(MyLCD, dot);
106    sprintf(deci, "%d", dec);
107    LCD16X2_Write_String(MyLCD, deci);
108    sprintf(in, "%d", i);|
109    LCD16X2_Set_Cursor(MyLCD, 1, 1);
110    LCD16X2_Write_String(MyLCD, in);
111    /* USER CODE BEGIN SysInit */
```



The diagram shows the following connections:

- Microcontroller (STM32F103C6) Pins:**
  - PA0-WKUP (Pin 10) to NRST (Pin 7)
  - PA1 (Pin 11) to PA15 (Pin 38)
  - PA2 (Pin 12) to PA15 (Pin 38)
  - PA3 (Pin 13) to PA15 (Pin 38)
  - PA4 (Pin 14) to PA15 (Pin 38)
  - PA5 (Pin 15) to PA15 (Pin 38)
  - PA6 (Pin 16) to PA15 (Pin 38)
  - PA7 (Pin 17) to PA15 (Pin 38)
  - PA8 (Pin 18) to PA15 (Pin 38)
  - PA9 (Pin 19) to PA15 (Pin 38)
  - PA10 (Pin 20) to PA15 (Pin 38)
  - PA11 (Pin 21) to PA15 (Pin 38)
  - PA12 (Pin 22) to PA15 (Pin 38)
  - PA13 (Pin 23) to PA15 (Pin 38)
  - PA14 (Pin 24) to PA15 (Pin 38)
  - PA15 (Pin 25) to PA15 (Pin 38)
  - PB0 (Pin 26) to PB15 (Pin 44)
  - PB1 (Pin 27) to PB15 (Pin 44)
  - PB2 (Pin 28) to PB15 (Pin 44)
  - PB3 (Pin 29) to PB15 (Pin 44)
  - PB4 (Pin 30) to PB15 (Pin 44)
  - PB5 (Pin 31) to PB15 (Pin 44)
  - PB6 (Pin 32) to PB15 (Pin 44)
  - PB7 (Pin 33) to PB15 (Pin 44)
  - PB8 (Pin 34) to PB15 (Pin 44)
  - PB9 (Pin 35) to PB15 (Pin 44)
  - PB10 (Pin 36) to PB15 (Pin 44)
  - PB11 (Pin 37) to PB15 (Pin 44)
  - PB12 (Pin 38) to PB15 (Pin 44)
  - PB13 (Pin 39) to PB15 (Pin 44)
  - PB14 (Pin 40) to PB15 (Pin 44)
  - PB15 (Pin 41) to PB15 (Pin 44)
  - NRST (Pin 7) to NRST (Pin 7)
  - VBAT (Pin 1) to VBAT (Pin 1)
  - BOOT0 (Pin 44) to BOOT0 (Pin 44)
- LCD1601 Pins:**
  - VSS (Pin 1) to VSS (Pin 1)
  - VDD (Pin 2) to VDD (Pin 2)
  - VEE (Pin 3) to VEE (Pin 3)
  - RS (Pin 4) to RS (Pin 4)
  - RW (Pin 5) to RW (Pin 5)
  - E (Pin 6) to E (Pin 6)
  - D0 (Pin 7) to D0 (Pin 7)
  - D1 (Pin 8) to D1 (Pin 8)
  - D2 (Pin 9) to D2 (Pin 9)
  - D3 (Pin 10) to D3 (Pin 10)
  - D4 (Pin 11) to D4 (Pin 11)
  - D5 (Pin 12) to D5 (Pin 12)
  - D6 (Pin 13) to D6 (Pin 13)
  - D7 (Pin 14) to D7 (Pin 14)
- Potentiometer RV1:**
  - V1(1) (Pin 1) to V1(1) (Pin 1)
  - Ground (Pin 2) to Ground (Pin 2)
  - Wiper (Pin 3) to Wiper (Pin 3)

```

105 while (1)
106 {
107     /* USER CODE END WHILE */
108     main = f;
109     dec = (f*100);
110     dec = dec%100;
111
112
113
114
115     LCD16X2_Init(MyLCD);
116     LCD16X2_Clear(MyLCD);
117
118     LCD16X2_Set_Cursor(MyLCD, 2, 1);
119     sprintf(fl, "%d", main);
120     LCD16X2_Write_String(MyLCD, fl);
121     LCD16X2_Write_Char(MyLCD, dot);
122     sprintf(deci, "%d", dec);
123     LCD16X2_Write_String(MyLCD, deci);
124     sprintf(in, "%d", i);
125     LCD16X2_Set_Cursor(MyLCD, 1, 1);
126     LCD16X2_Write_String(MyLCD, in);
127     LCD16X2_SR(MyLCD);
128     f = f*2;
129     i = i*2;
130     HAL_Delay(1000);
131     LCD16X2_SL(MyLCD);

```