گزارش تمرین چهارم

درس ابزار دقیق

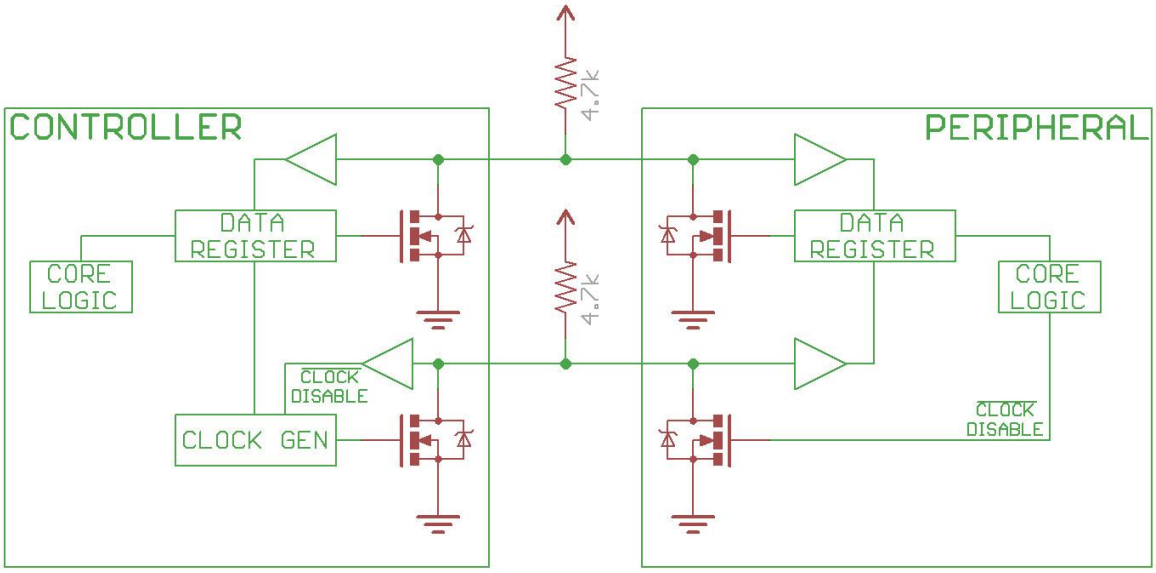
دکتر نیری

رضا مومنی

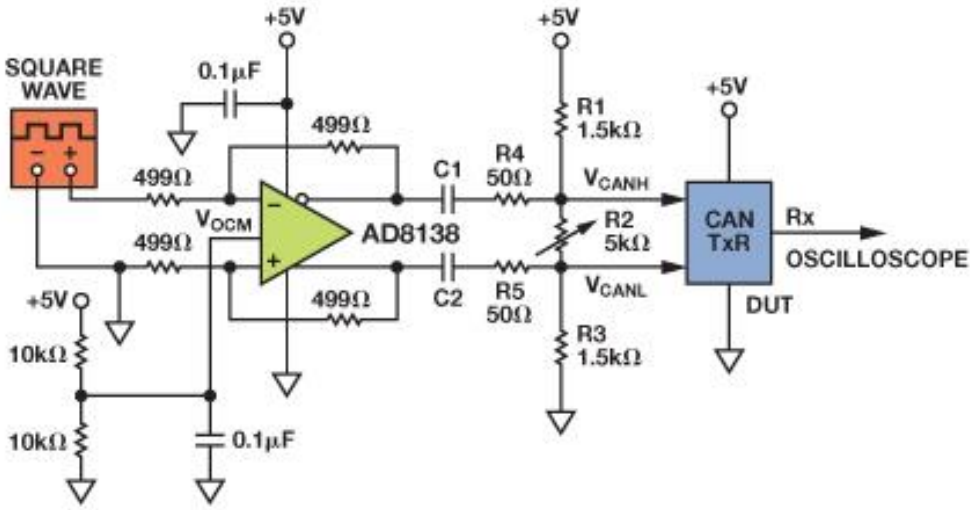
810199497

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

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

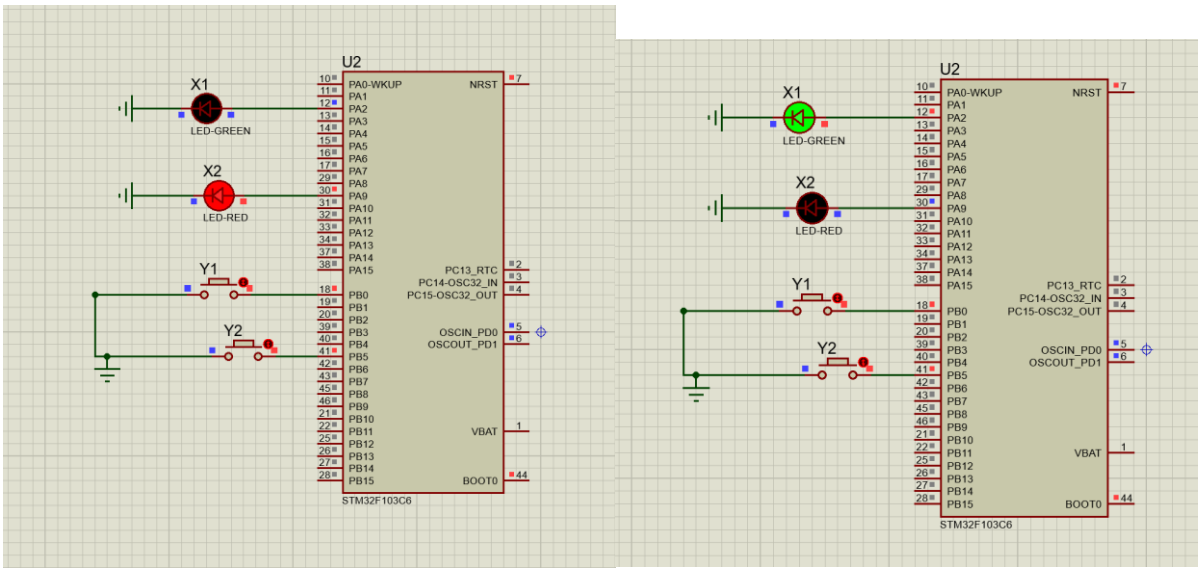
810199497 % 3 = 0 => MC: STM32F103C6

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:

```c
  while (1)
  {
    /* USER CODE END WHILE */
      led1 =  HAL_GPIO_ReadPin(GPIOB, Y1_Pin);
    led2 =  HAL_GPIO_ReadPin(GPIOB, Y2_Pin);

    if (led1==0){
      HAL_GPIO_WritePin(GPIOA, x1_Pin, GPIO_PIN_SET);
      HAL_GPIO_WritePin(GPIOA, x2_Pin, GPIO_PIN_RESET);
    }

    if (led2==0){
      HAL_GPIO_WritePin(GPIOA, x2_Pin, GPIO_PIN_SET);
      HAL_GPIO_WritePin(GPIOA, x1_Pin, GPIO_PIN_RESET);
      }
    //HAL_Delay(500);
      /* USER CODE BEGIN 3 */
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

```

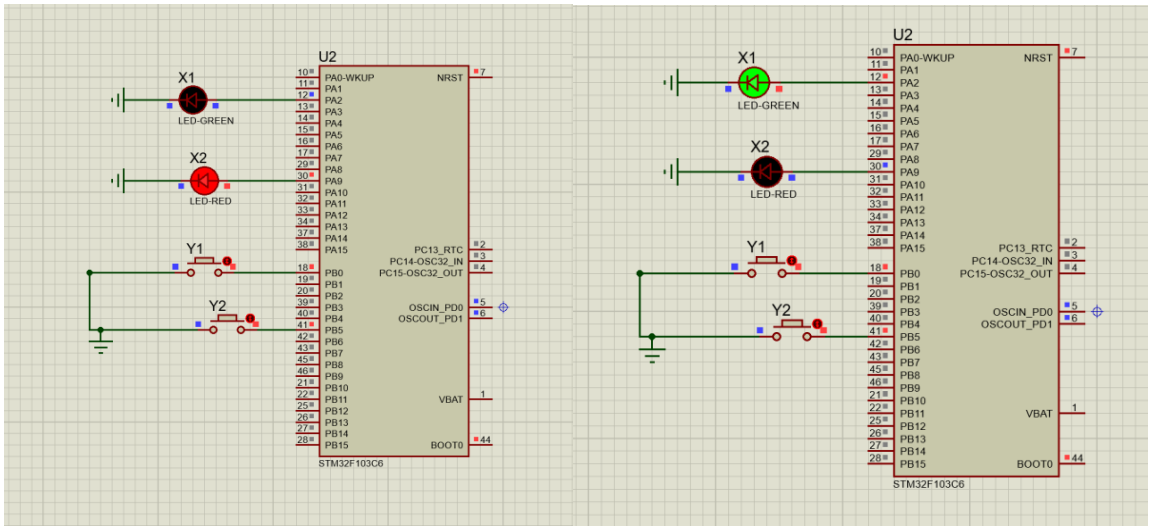


II.

```c
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    //if (GPIO_Pin == Y2_Pin){
        led2 = 0;
    //}
}
```

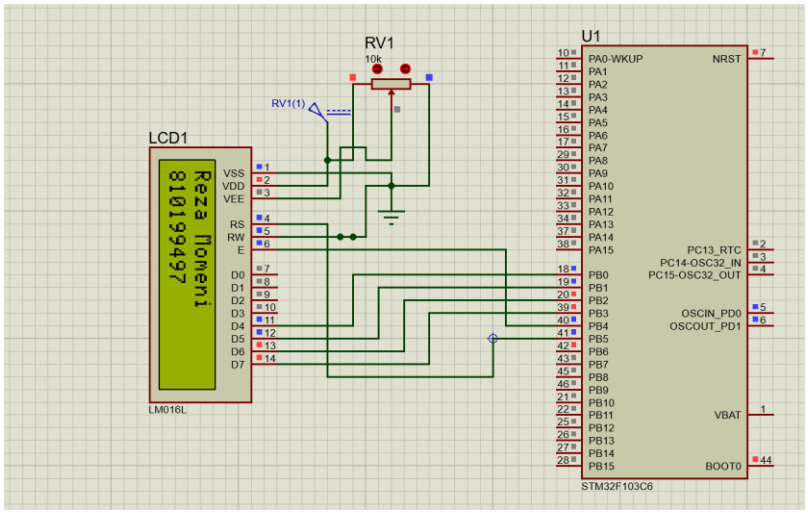
```c
while (1)
{
  /* USER CODE END WHILE */
    led1 =  HAL_GPIO_ReadPin(GPIOB, Y1_Pin);

          if (led1==0){
              HAL_GPIO_WritePin(GPIOA, x1_Pin, GPIO_PIN_SET);
              HAL_GPIO_WritePin(GPIOA, x2_Pin, GPIO_PIN_RESET);
          }

          if (led2==0){
              HAL_GPIO_WritePin(GPIOA, x2_Pin, GPIO_PIN_SET);
              HAL_GPIO_WritePin(GPIOA, x1_Pin, GPIO_PIN_RESET);
              }
          led2=1;
  /* USER CODE BEGIN 3 */
  /* USER CODE BEGIN 3 */
}
```

مزیت اصلی که اینتراپت 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 اینتراپت با کلاک اصلی خود میکرو کار میکنه و نیازی به دیلی خارجی نداره.



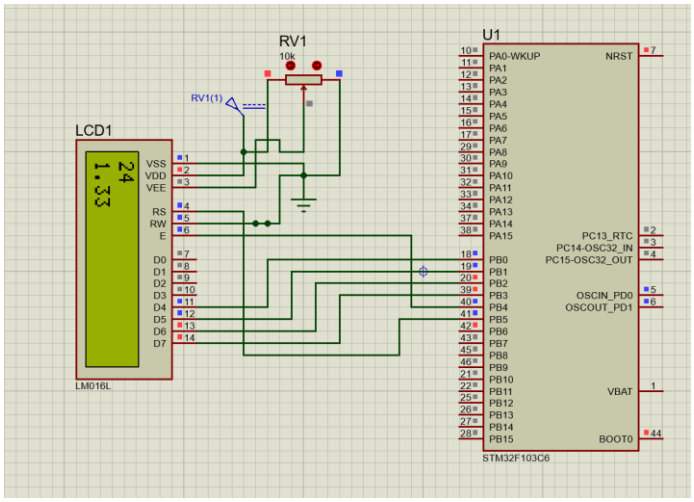
Question 2

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 */
```

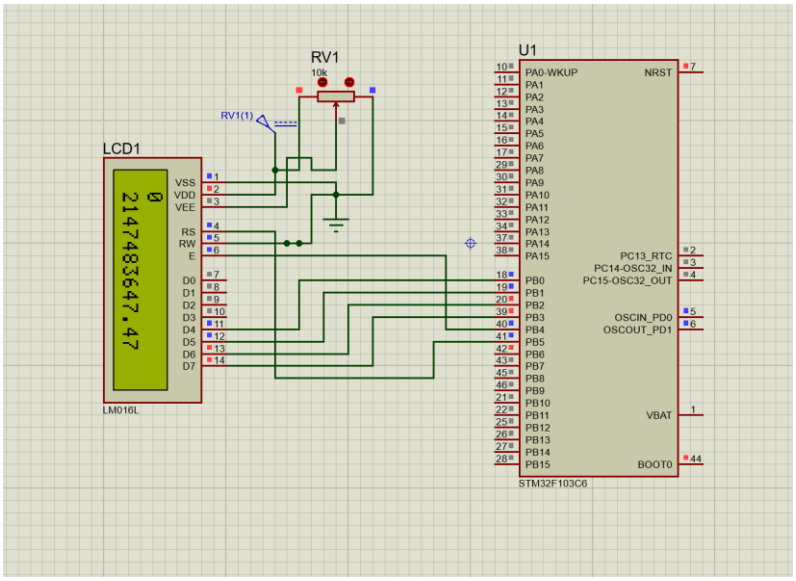
II.



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

```
92    maiin = 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", maiin);
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 */
```

III.



بعد از مدتی همانطور که مشاهده می شود عدد صحیح 0 شده و عدد اعشاری بدون تغییر باقی میماند. این به این دلیل است که تعداد بیت ظرفیت این متغیرها پر شده و overflow رخ داده است. برای داشتن متغیری با ظرفیت بیشتر میبایست از تایپ های دیگری از متغیرها مثل uint16 و long و double استفاده کنیم که ظرفیت بیشتری دارند.

```
105   while (1)
106   {
107     /* USER CODE END WHILE */
108       maiin = 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", maiin);
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);
```