آزمایش 9

طاها موسوی 98243058

نیلوفر مرادی جم 97243063

گروه 2

سوالات تحلیلی:

1 – چالش هایی که برای ایجاد یک ارتباط سریال غیر همزمان وجود دارد را با ذکر راهکار بیان نمایید.

1- سیگنال های سطح منطقی (0 تا 1.65 ولت، 1.65 ولت تا 3.3 ولت) به نویز و کاهش سیگنال حساس اند.

راهکار:

1- افزایش سطح ولتاژ برای افزایش noise margin. برای مثال از مثبت منفی 3 به مثبت منفی 15 ولت رسانده شده.

2- Differential signaling: به جای استفاده از یک سیم که نسبت به زمین ولتاژ را تعیین می‌کند تا 0 و 1 منطقی داشته باشیم، با دو سیم داده را جابجا می‌کنیم و اختلاف پتانسیل بین دو سیم موزد استفاده قرار می‌گیره.

2- برای حالت هایی که چند دیوایس می‌خواهند با هم صحبت کنند به این شکل خوب پشتیبانی نمیشه.

راهکار:

1 – با یک فرستنده چند گیرنده را هدایت کنیم.

2- فرستنده ها و گیرنده ها را به یک خط داده متصل کنیم.

2 – ارسال و دریافت با UART به کمک DMA چه مزایایی دارد؟

DMA برای انتقال داده ها از رجیستر داده USART RX به حافظه کاربر در سطح سخت افزار استفاده می شود. هیچ تعامل برنامه ای در این مرحله مورد نیاز نیست، به جز پردازش داده های دریافتی توسط برنامه زمانی که لازم باشد.

- انتقال از دستگاه جانبی USART به حافظه در سطح سخت افزار بدون تعامل CPU انجام می شود
- می تواند به راحتی با سیستم عامل ها کار کند
- بهینه شده برای بالاترین Baud rate > Mbps1 و برنامه های کم مصرف
- در صورت انفجار بزرگ داده ها، افزایش اندازه بافر داده می تواند عملکرد را بهبود بخشد.

3 – 4 مورد از مزایا و 4 مورد از معایب SPI را در مقایسه با I^2C شرح دهید

مزایا:

1- خطوط داده مجزای MISO/MOSI به این معنی است که برخلاف عملیات -I2C hald duplex، قادر به برقراری ارتباط کامل دوطرفه است، به این معنی که ارسال و دریافت داده باید به طور متناوب ارسال شود.

2- سرعت: I2C در ابتدا نرخ انتقال داده را 100 کیلوبیت بر ثانیه تعریف می کرد، اگرچه ما شاهد افزایش سرعت آن تا 400 کیلوبیت بر ثانیه یا حتی تا 5 مگابیت در ثانیه در حالت فوق سریع بوده ایم. با این حال، SPI یک سرعت ارتباطی بالا یا هیچ کدام را تعریف نمی کند و می تواند با سرعت 10 مگابیت در ثانیه یا بیشتر پیاده سازی شود.

3- هیچ بیت شروع و توقفی وجود ندارد، بنابراین داده ها می توانند به طور مداوم بدون وقفه جریان داشته باشند.

4- هیچ سیستم آدرس دهی پیچیده ای مانند I2C وجود ندارد

معایب:

1- آشکارترین تفاوت بین I2C و SPI این است که I2C به عنوان یک گذرگاه 2 سیم کار می کند و برای انتقال و همگام سازی داده ها فقط به خطوط داده سریال (SDA) و ساعت سریال (SCK) نیاز دارد. SPI، از سوی دیگر، برای کنترل یک Slave به چهار سیم نیاز دارد: SCK، Master out Slave in (MOSI)، Master in Slave Out (MISO) و Slave Select (SS) .

2- هنگامی که کاربران به بیش از یک دستگاه slave نیاز دارند، SPI یک پین SS اضافی را برای هر یک پیاده سازی می کند. هنگامی که یک سیستم I2C نیاز به پیاده سازی دستگاه های slave جدید دارد، آنها می توانند به سادگی با استفاده از یک سیستم آدرس دهی 7 بیتی برای شناسایی هر ماژول، به گذرگاه موجود متصل شوند. این طرح I2C به پیکربندی آدرس مناسب نیاز دارد اما از بار سیم کشی اضافی برای هر دستگاه جلوگیری می کند. اما در spi به این صورت نمیشود عمل کرد.

3- هیچ تاییدی مبنی بر اینکه داده ها با موفقیت دریافت شده است. اما I2C این را دارد.

4- این فقط برای یک single master اجازه می دهد.

رفرنس های سوالات تحلیلی:

- کلاس درس و اسلاید های درسی

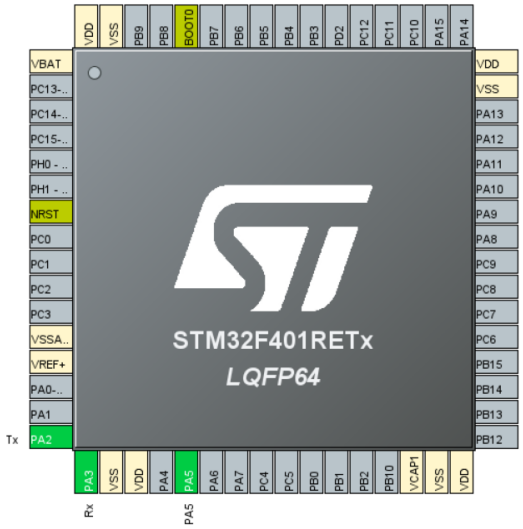
https://stm32f4-discovery.net/2017/07/stm32-tutorial-efficiently-receive-uart-data-using-dma/

https://www.arrow.com/en/research-and-events/articles/spi-vs-i2c-protocols-pros-and-cons

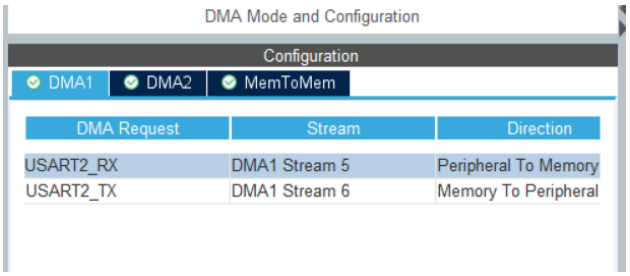
https://aticleworld.com/difference-between-i2c-and-spi/

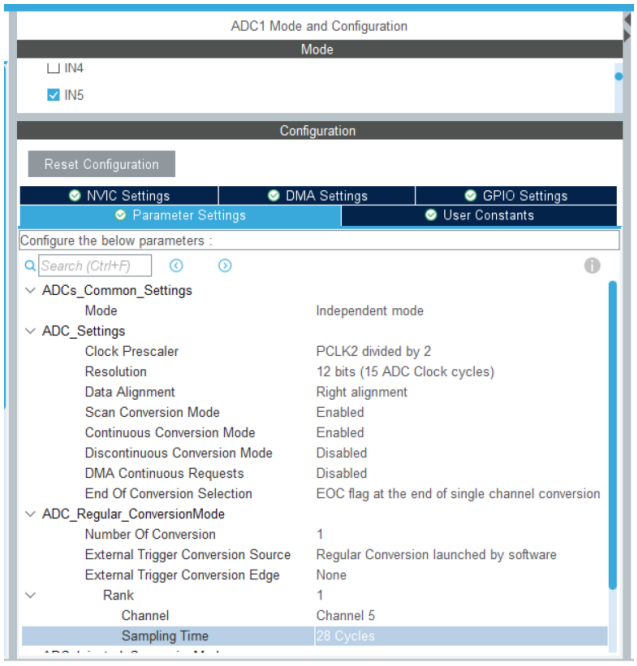
دستور کار:

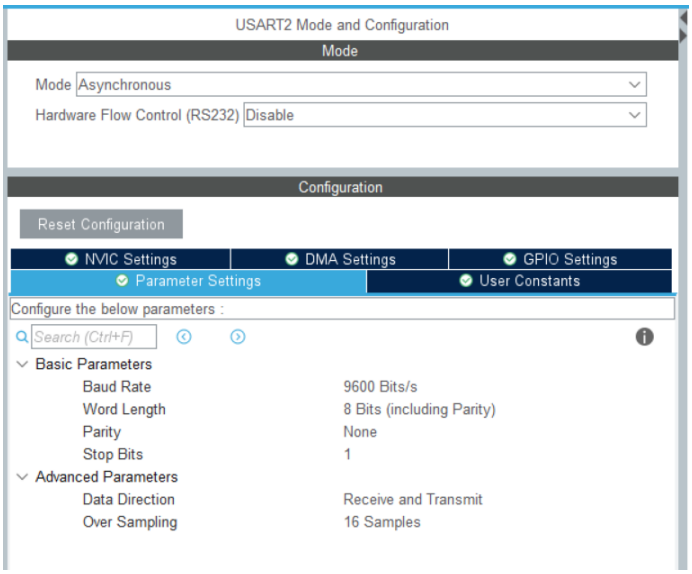
طرح کلی برای میکروی اول:

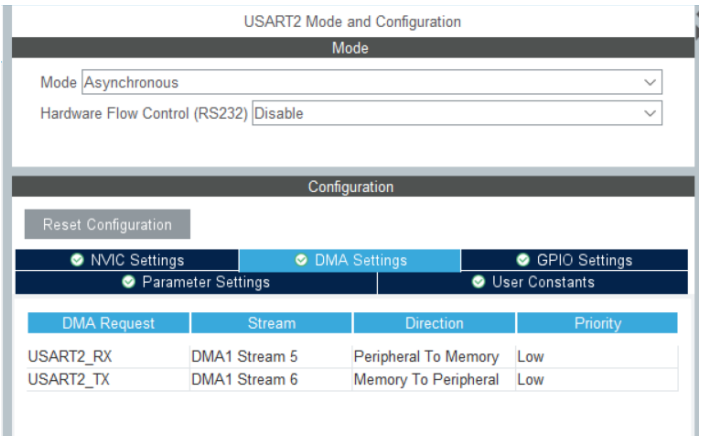


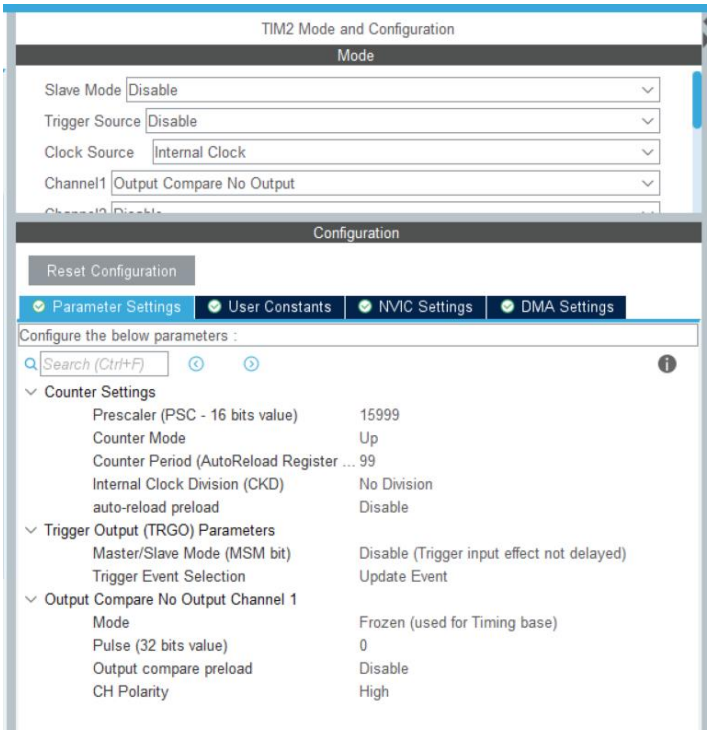
میکروی اول به شکل زیر کانفیگور شده است:






USART2 Mode and Configuration
Mode
Mode Asynchronous
Hardware Flow Control (RS232) Disable
Configuration
Reset Configuration
NVIC Settings
DMA Settings
GPIO Settings
Parameter Settings
User Constants
Configure the below parameters :
Search (Ctrl+F)
Basic Parameters
Baud Rate 9600 Bits/s
Word Length 8 Bits (including Parity)
Parity None
Stop Bits 1
Advanced Parameters
Data Direction Receive and Transmit
Over Sampling 16 Samples


USART2 Mode and Configuration
Mode
Mode Asynchronous
Hardware Flow Control (RS232) Disable
Configuration
Reset Configuration
NVIC Settings
DMA Settings
GPIO Settings
Parameter Settings
User Constants
DMA Request
Stream
Direction
Priority
USART2_RX
DMA1 Stream 5
Peripheral To Memory
Low
USART2_TX
DMA1 Stream 6
Memory To Peripheral
Low


TIM2 Mode and Configuration
Mode
Slave Mode Disable
Trigger Source Disable
Clock Source Internal Clock
Channel1 Output Compare No Output
Configuration
Reset Configuration
Parameter Settings
User Constants
NVIC Settings
DMA Settings
Configure the below parameters :
Search (Ctrl+F)
Counter Settings
Prescaler (PSC - 16 bits value) 15999
Counter Mode Up
Counter Period (AutoReload Register ... 99
Internal Clock Division (CKD) No Division
auto-reload preload Disable
Trigger Output (TRGO) Parameters
Master/Slave Mode (MSM bit) Disable (Trigger input effect not delayed)
Trigger Event Selection Update Event
Output Compare No Output Channel 1
Mode Frozen (used for Timing base)
Pulse (32 bits value) 0
Output compare preload Disable
CH Polarity High

```
339
340  void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
341  {
342
343    HAL_GPIO_TogglePin(GPIOC,GPIO_PIN_0);
344    adc_val = HAL_ADC_GetValue(&hadc1);
345    adc_val = adc_val * 5 /4096 ;
346
347    sprintf(buffer, "%02d", (int)adc_val);
348
349
350
351    HAL_UART_Transmit_DMA(&huart2,(uint8_t*) buffer, 100);
352    __HAL_UART_ENABLE_IT(&huart2, UART_IT_TXE);
353
354    HAL_Delay(100);
355    HAL_ADC_Stop(&hadc1);
356
357  }
358
359  void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart)
360  {
361      HAL_UART_DeInit(&huart2);
362  }
363
364
365  /* USER CODE END 4 */
366
```

محاسبه ولتاژ، ارسال آن از طریق uart به dma در تابع adc conccpltCallback از HAL

استفاده از ADC در این قسمت، شبیه تکلیف قبلی است. فقط اینجا باید بعد از اینکه مقدار را از ADC گرفتیم و convertکردیم، آن را روی UART به میکروکنترلر دیگر بفرستیم. این کار را با استفاده از Transmit_UART_HAL() انجام میدهیم. به دلیل استفاده از dma از hal_uart_transmit_dma استفاده میکنیم.

در micro2 باید با استفاده از manual reference رجیسترهای متناظر برای راه اندازی UART و SPI را کانفیگور کنیم.

```
25 void UART2_init(void){
26   RCC->APB1ENR |= 0x20000;  // Enable UART2 CLOCK
27   RCC->AHB1ENR |= 0x01; // Enable GPIOA CLOCK
28
29   GPIOA->MODER |= 0x000000A0;  // bits 7-4 = 1010 = 0xA --> Alternate Function for Pin PA2 & PA3
30   GPIOA->OSPEEDR |= 0x000000F0;  // bits 7-4 = 1111 = 0xF --> High Speed for PIN PA2 and PA3
31   GPIOA->AFR[0] |= 0x07700;  // bits 15-8=01110111=0x77  --> AF7 Alternate function for USART2 at Pin PA2 & PA3
32
33   USART2->BRR = 0xD05;   // Baud rate = 4800bps, CLK = 16MHz
34
35   USART2->CR1 = 1<<13;  // Enable USART
36   USART2->CR1 &= ~(1<<12);  // M = 0; 8 bit word length
37
38   USART2->CR1 |= (1<<2); // RE=1.. Enable the Receiver
39   USART2->CR1 |= (1<<3);  // TE=1.. Enable Transmitter
40
41   USART2->CR2 &= (1<<13);  // bits 13,12 = 10; 2 stop bits
42 }
43
```

همچنین تابع زیر برای خواندن داده استفاده میشود:

```
unsigned int UART2_read(void) {
  while (!(USART2->SR & 0x0020));  // wait for RXNE bit to set
  return USART2->DR;
}
```

19.6.3 Baud rate register (USART_BRR)

Note: The baud counters stop counting if the TE or RE bits are disabled respectively.

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value

Bits 15:4 **DIV_Mantissa[11:0]**: mantissa of USARTDIV
These 12 bits define the mantissa of the USART Divider (USARTDIV)

Bits 3:0 **DIV_Fraction[3:0]**: fraction of USARTDIV
These 4 bits define the fraction of the USART Divider (USARTDIV). When OVER8=1, the DIV_Fraction3 bit is not considered and must be kept cleared.

19.6.4 Control register 1 (USART_CR1)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OVER8	Reserved	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
rw	Res.	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 13 **UE**: USART enable
When this bit is cleared, the USART prescalers and outputs are stopped and the end of the current byte transfer in order to reduce power consumption. This bit is set and cleared by software.
0: USART prescaler and outputs disabled
1: USART enabled

Bit 12 **M**: Word length
This bit determines the word length. It is set or cleared by software.
0: 1 Start bit, 8 Data bits, n Stop bit
1: 1 Start bit, 9 Data bits, n Stop bit
Note: The M bit must not be modified during a data transfer (both transmission and reception)

Bit 3 **TE**: Transmitter enable
This bit enables the transmitter. It is set and cleared by software.
0: Transmitter is disabled
1: Transmitter is enabled
Note: During transmission, a "0" pulse on the TE bit ("0" followed by "1") sends a preamble (idle line) after the current word, except in smartcard mode.
When TE is set, there is a 1 bit-time delay before the transmission starts.

Bit 2 **RE**: Receiver enable
This bit enables the receiver. It is set and cleared by software.
0: Receiver is disabled
1: Receiver is enabled and begins searching for a start bit

19.6.5 Control register 2 (USART_CR2)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LINEN	STOP[1:0]		CLKEN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	Res.	ADD[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

Bits 31:15 Reserved, must be kept at reset value

Bit 14 **LINEN**: LIN mode enable
This bit is set and cleared by software.
0: LIN mode disabled
1: LIN mode enabled
The LIN mode enables the capability to send LIN Synch Breaks (13 low bits) using the SBK bit in the USART_CR1 register, and to detect LIN Sync breaks.

Bits 13:12 **STOP**: STOP bits
These bits are used for programming the stop bits.
00: 1 Stop bit
01: 0.5 Stop bit
10: 2 Stop bits
Note: 11: 1.5 Stop bit

```
51
52  void SPI1_init(void){
53    RCC->APB2ENR |= RCC_APB2ENR_SPI1EN; // enable SPI1 clock
54
55    // set Alternate Function mode for PA5 (sclk) & PA7 (MOSI:DIN for max7227)
56    GPIOA->MODER &= ~0x00000300U;
57    GPIOA->MODER |= 0x00008800;
58
59    // set to AF5(SPI1)
60    GPIOA->AFR[0] &= ~0xF0F00000;
61    GPIOA->AFR[0] |= 0x50500000;
62
63    // set PA4 to GPIO output mode
64    GPIOA->MODER &= ~0x00000300U;
65    GPIOA->MODER |= 0x00000100;
66
67    SPI1->CR1 = 0x0;
68    SPI1->CR1 &= ~(SPI_CR1_CPHA | SPI_CR1_CPOL) ; // reset CPHA and CPOL
69    SPI1->CR1 |= SPI_CR1_MSTR ; // set master selection
70    SPI1->CR1 |= (SPI_CR1_BR_0 | SPI_CR1_BR_1 | SPI_CR1_BR_2) ; // set baud rate to 111 : fpclk(16MHz)/256
71    SPI1->CR1 |= SPI_CR1_SSI | SPI_CR1_SSM ;// software slave manage mode
72    SPI1->CR1 |= SPI_CR1_SPE; // enable SPI_1
73
74    MAX_send_data(0xC01); // turn on
75    MAX_send_data(0xA1A); // set light intensity
76    MAX_send_data(0xB05); // scan limit
77    MAX_send_data(0x9ff); // decoding mode
78  }
79
```

رجیسترهای استفاده شده در spi_cr1:

Bit 9 **SSM:** Software slave management
When the SSM bit is set, the NSS pin input is replaced with the value from the SSI bit.
0: Software slave management disabled
1: Software slave management enabled
Note: This bit is not used in I^2S mode and SPI TI mode

Bit 8 **SSI:** Internal slave select
This bit has an effect only when the SSM bit is set. The value of this bit is forced onto the NSS pin and the IO value of the NSS pin is ignored.
Note: This bit is not used in I^2S mode and SPI TI mode

Bit 7 **LSBFIRST:** Frame format
0: MSB transmitted first
1: LSB transmitted first
Note: This bit should not be changed when communication is ongoing.
It is not used in I^2S mode and SPI TI mode

Bit 6 **SPE:** SPI enable
0: Peripheral disabled
1: Peripheral enabled
Note: This bit is not used in I^2S mode.
When disabling the SPI, follow the procedure described in Section 20.3.8.

Bits 5:3 **BR[2:0]:** Baud rate control
000: $f_{PCLK}/2$
001: $f_{PCLK}/4$
010: $f_{PCLK}/8$
011: $f_{PCLK}/16$
100: $f_{PCLK}/32$
101: $f_{PCLK}/64$
110: $f_{PCLK}/128$
111: $f_{PCLK}/256$
Note: These bits should not be changed when communication is ongoing.
They are not used in I^2S mode.

Bit 2 **MSTR:** Master selection
0: Slave configuration
1: Master configuration
Note: This bit should not be changed when communication is ongoing.
It is not used in I^2S mode.

Bit1 **CPOL:** Clock polarity
0: CK to 0 when idle
1: CK to 1 when idle
Note: This bit should not be changed when communication is ongoing.
It is not used in I^2S mode and SPI TI mode.

Bit 0 **CPHA:** Clock phase
0: The first clock transition is the first data capture edge
1: The second clock transition is the first data capture edge
Note: This bit should not be changed when communication is ongoing.
It is not used in I^2S mode and SPI TI mode.

تابع data_send_MAX() به صورت زیر پیاده سازی شده است.

```
82
83  void MAX_send_data(unsigned int data ) {
84    while (!(SPI1->SR & SPI_SR_TXE)); // wait til TE empty
85    GPIOA->BSRR = 0x00100000; //slave clk to low
86    SPI1->DR = data >> 8 ;  // write command
87    while (!(SPI1->SR & SPI_SR_TXE));
88    SPI1->DR = data & 0xff;  // write 8 bit data
89    while (SPI1->SR & SPI_SR_BSY); // wait for data send
90    GPIOA->BSRR = 0x0000010; //slave clk to high
91  }
92
```

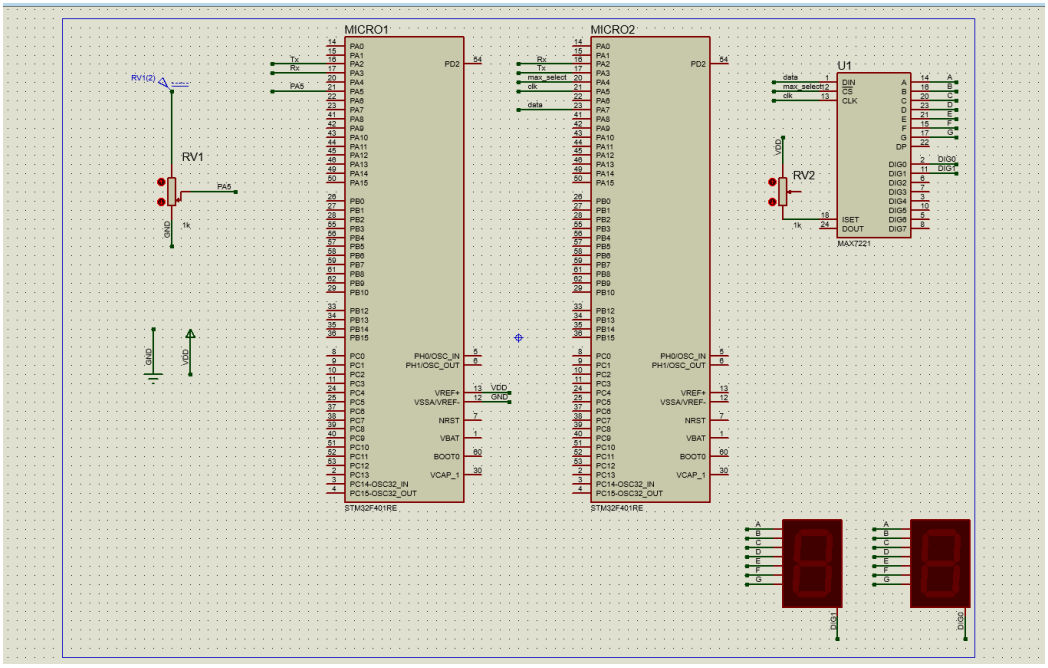
تابع digits_write() نیز به صورت زیر پیاده سازی شده است.

```
93  // write 2 digits to MAX7227
94  void write_digits(void){
95      unsigned int digit_1 = UART2_read() - '0';
96      unsigned int digit_0 = UART2_read() - '0';
97
98      MAX_send_data(mask(8) + digit_0);
99      MAX_send_data(mask(9) + digit_1);
100 }
101
```

و تابع main() در micro2 به شکل زیر است:

```
15  int main (void) {
16    UART2_init();
17    SPI1_init();
18
19    while(1) {
20      write_digits(); // read from UART and sent digits to max7227
21    }
22  }
23
```

پروتئوس:



رفرنس دستور کار:

کلاس درس و اسلاید های درسی

دیتا شیت و رفرنس منوآل