بسمه تعالی

سید محمدرضا حسینی

حسین حاجی رومنان

گزارش آزمایش ۱۰ آزمایشگاه ریزپردازنده

سوالات تشریحی

سوال اول: با مطالعه پروتکل سریال USART و بلوک مربوطه در STM۳۲F۴۰۱، پارامتر های قابل تنظیم آن را ذکر کنید..

بخش USART در میکرو های STM۳۲ بسیار انعطاف پذیر برای تبادل داده به صورت تمام دوبلکس با تجهیزات خارجی ارائه می دهد. USART با استفاده از یک مولد نرخ باود کسری، طیف بسیار وسیعی از نرخ باود را ارائه می دهد. از ارتباطات یک طرفه همزمان و ارتباطات تک سیم نیمه دوبلکس نیز پشتیبانی می کند. همچنین از LIN (شبکه اتصال محلی)، پروتکل Smartcard و IrDA (داده های مادون قرمز) و عملیات مودم (CTS/RTS) پشتیبانی می کند. این امکان ارتباط بین چند پردازنده را فراهم می کند.

با توجه به موارد ذکر شده، میتوان این بلوک از میکرو را در مد های مختلف کاری راه اندازی کرد که اکثر پارامترهای هر مد مشترک هستند ولی هر مد تعدادی پاارمتر خاص خود را نیز دارند.

در زیر پارامتر های مشترک بررسی میگردد:

Baud Rate: مشخص میکند دیتا با چه نرخی از baud ارسال و دریافت شود. میتواند بین ۲۴۵ بیت در ثانیه تا ۱۰۰۰کیلوبیت در ثانیه باشد.

Word length: مشخص میکند در هر ارسال، چند بیت دیتای قابل استفاده موجود است.

Parity: مشخص میکند که آیا بیت Parity برای داده تولید شود یا خیر، در صورت تولید شدن زوج باشد یا فرد.

Sop Bit: تعداد Stop bit برای هر داده را مشخص میکند و میتواند ۰.۵، ۱، ۱/۵ یا ۲ باشد.

Data order: مشخص میکند اول MSB ارسال شود یا LSB.

پارامتر های مختص هر مد در زیر آورده شده اند:

در مورد اتصال غیر همزمان، میتوان تنظیم نمود که دو سیسیتم از بیت های کنترلی CTS یا RTS یا هردو با هم استفاده کنند یا نه.

در مد همزمان میتوان پارامتر های Clok polarity، Clock phase و Clock Last bit را مشخص نمود. همچنین جهت حرکت داده (ارسال یا دریافت یا ترکیبی از آن دو) نیز قابل تنظیم است.

در مد غیر همزمان و اتصال تک سیمه، جهت حرکت داده و تعداد Oversampling از هر داده قابل تنظیم است. در مد Multi Processor علاوه بر این موارد، پارامتر Wakeup method نیز قابل تنظیم است.

در مد IrDA علاوه بر جهت داده، Power mode و prescaler نیز قابل تنظیم هستند.

در مد LIN جهت داده و Break detect length (۱۰ یا ۱۱) قابل تنظیم است.

در مد SmartCard پارامترهای جهت داده، Nack دادن در صورت parity error و guardtime (بین ۰ تا ۲۵۵) قابل تنظیم هستند.

سوال دوم: نحوه فعال کردن وقفه دریافت داده های سریال UART را بیان کرده و ریجستر های مربوطه را به طور کامل شرح دهید.

- در ابتدا با مقداردهی به رجیسترهای CR و CFGR، منبع کلاک سیستم را HSI تعیین می کنیم..
- با نوشتن ۱ بر روی UE در رجیستر USART_CR۱ ،واحد USART را فعال میکنیم.
- به کمک M در USART_CR۱ تعداد بیتهای داده را مشخص میکنیم.
- در USART_CR۱، RE را یک میکنیم تا بخش رسیور فعال شود. همچنین RXNEIE را نیز مقدار میدهیم

30.6.4 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

تا در هنگام دریافت داده، درخواست وقفه ایجاد شود.

- در USART_CR۲ تعداد STOP BIT ها را مشخص میکنیم.

- با استفاده از فرمول زیر، مقادیر مرتبط با باود ریت مد نظر را محاسبه میکنیم:

Tx / Rx baud = fck / (۱۶ * USARTDIV)

در این رابطه USARTDIV همان Baud rate مورد نظر است. نتیجه را در رجیستر USART_BRR قرار میدهیم.

30.6.5 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

30.6.3 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.

بعد از اتمام دریافت دیتا توسط USART، بیت RXNE در رجیستر USART_SR ست شده و وقفه رخ میدهد.

30.6.1 Status register (USART_SR)

Address offset: 0x00

Reset value: 0x0000 00C0

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
Reserved						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
						rc_w0	rc_w0	r	rc_w0	rc_w0	r	r	r	r	r

در روتین وقفه، رجیستر USART_DR خوانده میشود تا دیتای دریافت شده بدست آید. با توجه به جدول زیر، این روتین باید در آدرس مرتبط نوشته شده باشد:

35	42	settable	SPI1	SPI1 global interrupt	0x0000_00CC
36	43	settable	SPI2	SPI2 global interrupt	0x0000_00D0
37	44	settable	USART1	USART1 global interrupt	0x0000_00D4
38	45	settable	USART2	USART2 global interrupt	0x0000_00D8
39	46	settable	USART3	USART3 global interrupt	0x0000_00DC
40	47	settable	EXTI15_10	EXTI Line[15:10] interrupts	0x0000_00E0

سوال سوم: کنترل خطا در ارتباط سریال USART و SPI با چه منطقی انجام میگیرد؟ شرح دهید.

در هنگام دریافت داده و توسط سخت افزار، چند بیت کنترلی قابلیت ست شدن دارند. برای USART این بیت ها در USART_SR قرار دارند.

30.6.1 Status register (USART_SR)

Address offset: 0x00

Reset value: 0x0000 00C0

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
Reserved						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
						rc_w0	rc_w0	r	rc_w0	rc_w0	r	r	r	r	r

Overrun error: این بیت در زمانی که داده ای جدید دریافت شود در حالی که هنوز داده قبلی خوانده نشده است، توسط سخت افزار تنظیم میشود.

Noise detected flag: این بیت زمانی که نویز روی یک فریم دریافتی تشخیص داده شود، توسط سخت افزار تنظیم می شود

Framing error: این بیت زمانی توسط سخت افزار تنظیم می شود که عدم همگامی[1] دو دستگاه USart، نویز بیش از حد یا یک کاراکتر break شناسایی شود.

Parity error: این بیت زمانی که خطای Parity در دیتای دریافتی وجود داشته باشد، توسط سخت افزار تنظیم می شود

و SPI نیز کنترل خطا در ارتباط را به کمک ۳ پرچم زیر انجام میدهد:

28.5.3 SPI status register (SPI_SR)

Address offset: 0x08

Reset value: 0x0002

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							FRE	BSY	OVR	MODF	CRC ERR	UDR	CHSIDE	TXE	RXNE
							r	r	r	r	rc_w0	r	r	r	r

Underrun flag: در حالت انتقال slave، این پرچم زمانی تنظیم می شود که اولین ساعت برای انتقال داده ظاهر شود در حالی که نرم افزار هنوز هیچ مقداری را در SPI_DR بارگذاری نکرده است.

---

[1] de-synchronization

Overrun flag: این پرچم زمانی تنظیم می شود که داده ای دریافت شود در حالی که داده قبلی هنوز از SPI_DR خوانده نشده است. در نتیجه، داده های دریافتی از بین می روند.

Frame error flag: این پرچم را تنها در صورتی توسط سخت افزار تنظیم میشود که I۲s در حالت Slave پیکربندی شده باشد. این فلگ تنها در صورتی تنظیم می شود که master خارجی، خط WS را در لحظه ای تغییر دهد که Slave انتظار این تغییر را نداشته باشد

برنامه نویس باید هنگام برداشتن داده، صحت دریافت (یا ارسال) داده را بررسی کند.

سوال چهارم: وظیفه کنترل داده ها و استخرا داده از بیت های کنترلی بر عهده برنامه نویس است یا ماژول سخت افزاری؟ شرح دهید که چرا از ماژول سخت افزاری برای این کار استفاده میشود؟

باس های سریع به خصوص مثل SPI و USART در فرکانس های بالا و برای اینکه بهترین عملکرد را داشته باشند، باید بتوانند به موقع تمام جریان داده را مدیریت کنند. سیستم باید از بیش از حد داده گرفتن در هنگام دریافت[۲] و کاهش داده در هنگام انتقال[۳] جلوگیری کند. مدیریت کارآمد تمام جریان‌های داده، زمانی که چارچوب زمانی برای ارسال یا دریافت داده کوتاه باشد و فرکانس کلاک قابل مقایسه با فرکانس هسته سیستم باشد، بسیار حیاتی است. برای مدیریت دریافت یا ارسال یک داده با فرکانس نزدیک به سیستم و توسط SPI، یک برنامه فرزی را در نظر بگیریم که در هر هر ۸ کلاک، نیاز باشد که پردازنده به دیتای SPI رسیدگی کند. در این مثال، حاشیه عملکرد کافی در سطح سیستم برای مدیریت چنین ارتباطاتی وجود ندارد، حتی زمانی که تمام عملکرد CPU و به صورت بهینه شده، صرف انجام این کار شود. سیستم باید پرچم‌های مرتبط را آزمایش کند و به ازی هر پرچم، عمل مرتبط را انجام دهد. حتی اگر این عملیات را به بهینه ترین حالت ممکن و به زبان اسمبلی بنوییسم، میبینم که پردازنده زمانی برای انجام تمامی کارهای لازم در ۸ سیکل ساعت را ندارد. یا مثلا اگر دریافت یا ارسال یک داده USART را در نظر بگیریم، پردازنده باید دائما منتظر زمان مناسب باشد تا دیتای مناسب را یا بر روی line قرار دهد یا از آن بخواند. حتی اگر اینکار توسط وقفه هم انجام شود، به خصوص در buad rate های بالا، پردازنده دائما در حال رفت و برگشت به وقفه های پی در پی است که صرف نظر از کد وقفه، رفتن و برگشتن از خود وقفه چندین سیکل ساعت طول میکشد. اگر قرار باشد نرم افزار بعد از اینکه یک فریم داده را دریافت کرد، بررسی کند که آیا دریافت به درستی صورت گرفته یا خطایی رخ داده است، عملا هیچ قدرت پردازشی اضافه ای برای بقیه انجام کار ها باقی نمی‌ماند. در نتیجه تمامی این کارها توسط سخت افزار و به صورت موازی با کارکرد پردازنده انجام میشود تا پردازنده فرصت انجام بقیه کارهای پردازشی را داشته باشد. حال برنامه نویس تنها لازم است که زمانی که سخت افزار داده را به طور کامل دریافت کرد، توسط وقفه، DMA یا به طور مستقیم در کد و با چک کردن بیت کنترلی، دیتای لازم را استفاده کند و توان پردازشی کنترلر را برای انجام عملیاتی بر اساس این داده استفاده کند، نه برای دریافت آن.

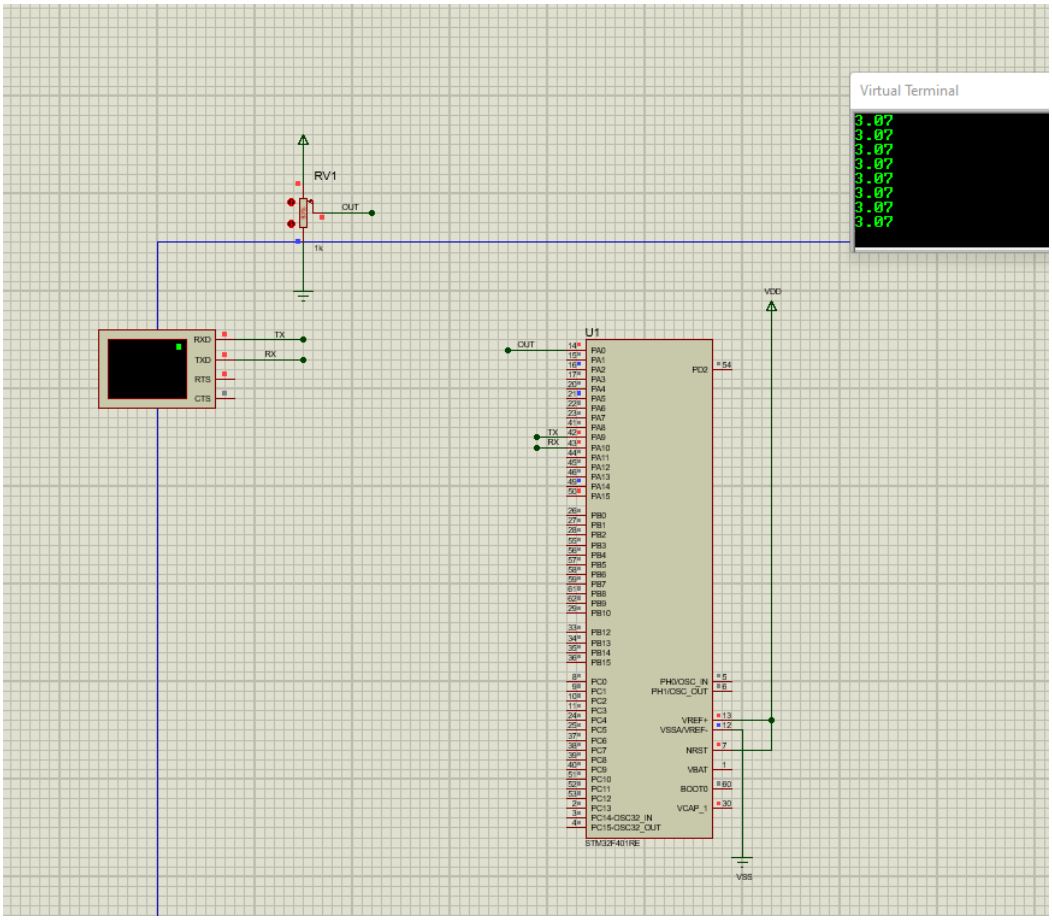
---

[۲] overrun

[۳] underrrun

سوالات پیاده سازی:

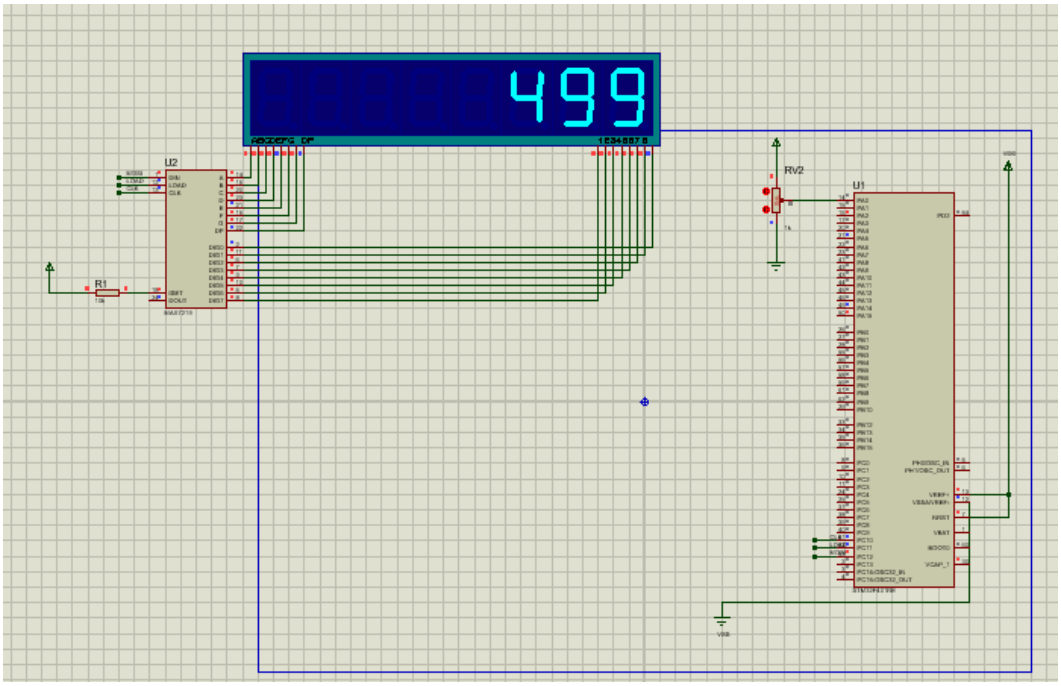
۱)



در این تمرین ADC و USART , timer استفاده شده است. در ابتدا تایمر فعال میشود تا در هر ۰/۱ ثانیه یکبار مقدار ADC از طریق pooling خوانده شود و سپس مقدار آن را با استفاده از USART که به صورت MASTER Transmitter تنظیم شده ارسال میشود و در Virtual terminal پروتئوس نتیجه با موفقیت نشان داده میشود.

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    HAL_ADC_Start(&hadc1);
    HAL_ADC_PollForConversion(&hadc1,1);
    voltage = HAL_ADC_GetValue(&hadc1);
    i = (voltage/4095)*3.3;
    sprintf(buffer,"%4.2f\r\n",i);
    HAL_UART_Transmit(&huart1,buffer,strlen(buffer), 1);
}
```

(۲



برای نوشتن مقادیر بر روی MAX۷۲۱۹ باید در ابتدا یک تابع داشته باشیم که مقدار آدرس و مقداری که باید در آن آدرس قرار بگیرد را بنویسیم .

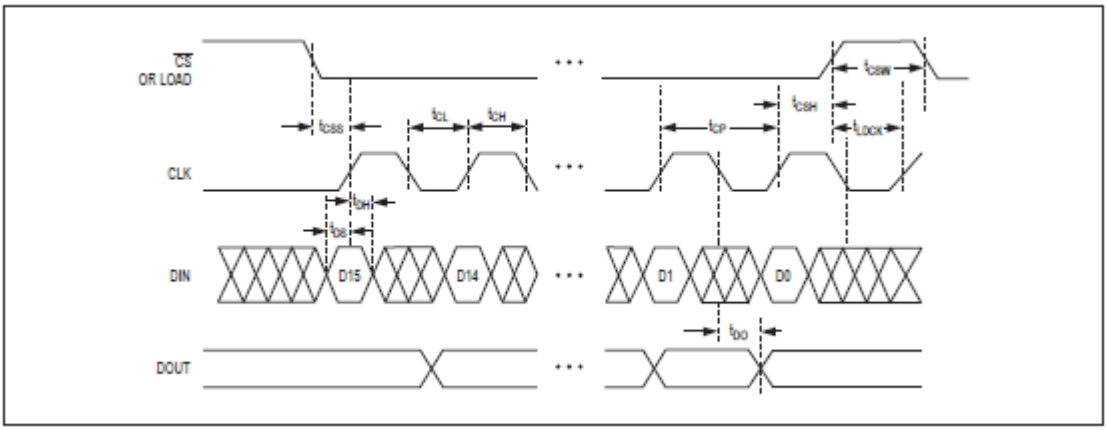


Figure 1. Timing Diagram

مطابق تصویر بالا که از دیتاشیت برداشته شده ، برای نوشتن بر روی MAX۷۲۱۹ ابتدا مقدار پین LOAD باید صفر شود و بعد از آن دیتا از طریق SPI بر روی پین دیتا قرار گیرد و پین کلاک SPI نیز باید به پین کلاک ۷۲۱۹ متصل شود و در پایان کار مقدار پین LOAD برابر یک شود. تمامی اتفاقات گفته شده در تابع transmit پیاده سازی شده که تصویر آن در زیر قابل مشاهده است .

```
void transmit(uint8_t add,uint8_t data){
uint8_t tx_buffer[0];
HAL_GPIO_WritePin(GPIOC,GPIO_PIN_11,0);
tx_buffer[0]=add;
HAL_SPI_Transmit(&hspi3,tx_buffer,1,0xffff);
tx_buffer[0]=data;
HAL_SPI_Transmit(&hspi3,tx_buffer,1,0xffff);
HAL_GPIO_WritePin(GPIOC,GPIO_PIN_11,1);
}
```

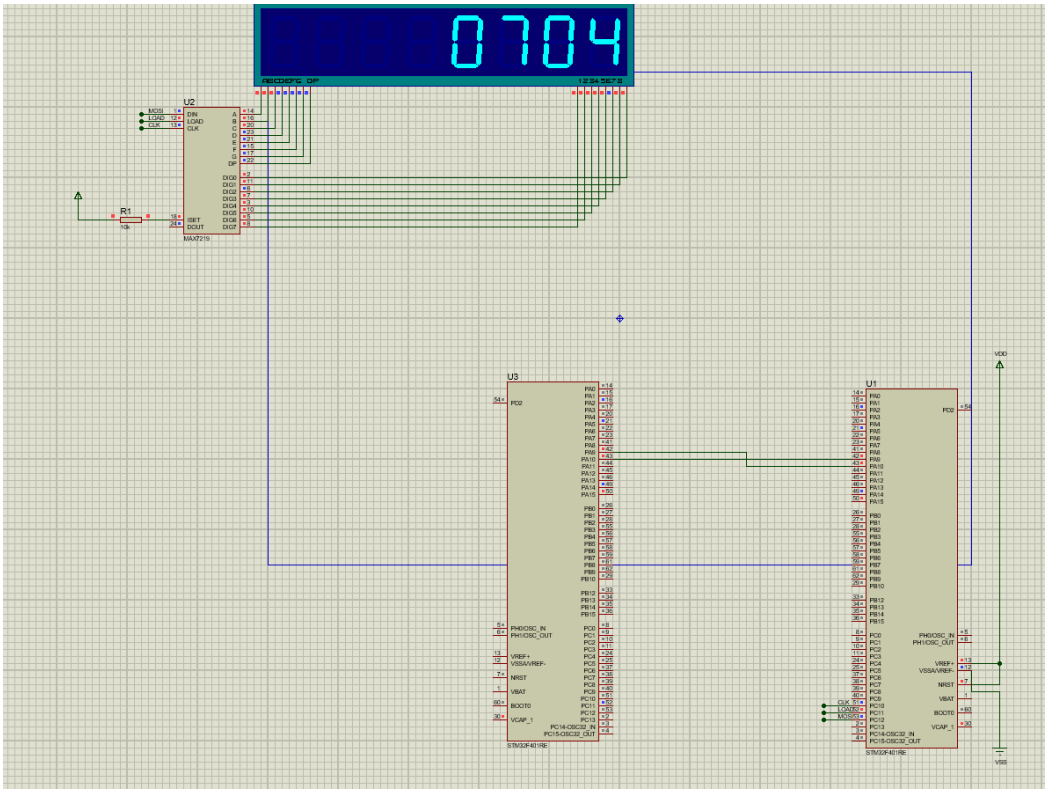
برای اینکه ۷۲۱۹ به درستی کار کند باید در ابتدا INITIALIZE شود . در تابع INIT ، ۷۲۱۹ مطابق خواست ما راه اندازی اولیه شده و از این به بعد میتوان مقادیر را بر روی آن نمایش داد.

```
void max7219_init(){
   transmit(0x09,0xff); // ENABLE ENCODER FOR ALL 7SEG
   transmit(0x0A,0x0F); // MAX INTENSITY
   transmit(0x0B,0x02); // SCAN 3 7SEG
   transmit(0x0C,0x01); // TURN ON
   transmit(0x0f,0x0); // NORMAL OPERATION
}
```

در حلقه while ، در ابتدا مقدار آنالوگ را با استفاده از ADC1 به صورت pooling میخوانیم و پس از خواندن آن را بر ۴۰۹۵ تقسیم کرده و در ۹۹۹ ضرب میکنیم تا بازه نمایش به اعداد بین ۰ تا ۹۹۹ برسد و پس از آن به ترتیب مقدار عدد خوانده شده را بر روی ۷۲۱۹ به نمایش در می آوریم.

```
  while (1)
  {
    HAL_ADC_Start(&hadc1);
    HAL_ADC_PollForConversion(&hadc1,0xffff);
    voltage = HAL_ADC_GetValue(&hadc1);
    int analog= (voltage/4095)*999;
for(int i=0;i<3;i++){
    int j=i+1;
    transmit(j,analog%10);
    analog/=10;
}
```

۳)



یکی از stm۳۲ ها نقش کانتر را داشته و برد دیگر برای نمایش اطلاعات استفاده میشود.

در کانتر یک تایمر استفاده شده که هر اینتراپت آن یک ثانیه یک بار اجرا میشود و در آن مقدار ثانیه شمار را یک واحد افزایش میابد و مقدار ثانیه و دقیقه برای آن یکی برد که برای نمایش استفاده شده ، از طریق USART ارسال میشود.

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    Second+=1;
    if (Second>59) {
    Second=0;
    Minute+=1;
    }
    //sprintf(TX_buffer,"%i\r\n",Second);
    TX_buffer[0]=(char)Second%10;
    TX_buffer[1]=(char)Second/10;
    TX_buffer[2]=(char)Minute%10;
    TX_buffer[3]=(char)Minute/10;
    HAL_UART_Transmit(&huart1,TX_buffer,sizeof(TX_buffer), HAL_MAX_DELAY);
}
```

در برد دیگر نیز مطابق تمرین ۲ ، ۷۲۱۹ مقدار دهی اولیه میشود . پس از دریافت اطلاعات از طریق USART ، اینتراپت هندلر USART در بردی که برای نمایش استفاده شده فراخوانی میشود که در آن مقدار دریافت شده بر روی ۷۲۱۹ به نمایش در می آید.

```
void transmit(uint8_t add,uint8_t data){
uint8_t tx_buffer[0];
HAL_GPIO_WritePin(GPIOC,GPIO_PIN_11,0);
tx_buffer[0]=add;
HAL_SPI_Transmit(&hspi3,tx_buffer,1,0xffff);
tx_buffer[0]=data;
HAL_SPI_Transmit(&hspi3,tx_buffer,1,0xffff);
HAL_GPIO_WritePin(GPIOC,GPIO_PIN_11,1);

}
void max7219_init(){
  transmit(0x09,0xff); // ENABLE ENCODER FOR ALL 7SEG
  transmit(0x0A,0x0F); // MAX INTENSITY
  transmit(0x0B,0x03); // SCAN 4 7SEG
  transmit(0x0C,0x01); // TURN ON
  transmit(0x0f,0x0); // NORMAL OPERATION
}
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
  {
    transmit(0x01,RX_buffer[0]);
    transmit(0x02,RX_buffer[1]);
    transmit(0x03,RX_buffer[2]);
    transmit(0x04,RX_buffer[3]);
    HAL_UART_Receive_IT(&huart1, RX_buffer, sizeof(RX_buffer));
  }
```

٤)

در این تمرین USART را در STM۳۲CUBEMX فعال میکنیم و مقدار BAUDRATE آن را برابر ۹۶۰۰ قرار میدهیم . سپس در main منتظر دریافت اطلاعات از طریق USART میمانیم . پس از دریافت اطلاعات ، اینتراپت هندلر اجرا میشود و در مقدار دریافت شده را در ۷۲۱۹ مطابق تمرین ۳ نمایش میدهیم . پس از آن مقدار دریافت شده را برای ارسال کننده مجددا ارسال میکنیم و بار دیگر منتظر دریافت اطلاعات میمانیم .

```
,
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  transmit(0x01,Buffer);
  HAL_UART_Transmit(&huart1, &Buffer, sizeof(Buffer), HAL_MAX_DELAY);
  HAL_UART_Receive_IT( &huart1, &Buffer, sizeof(Buffer));
}
```