به نام خدا

گزارش کار پروژه ی پایانی درس ریز پردازنده و زبان اسمبلی

استاد : جناب آقای دکتر عطارزاده نیاکی

تارا برقیان

مهرشاد سعادتی نیا

نیم سال اول ۰۱-۱۴۰۰

فهرست

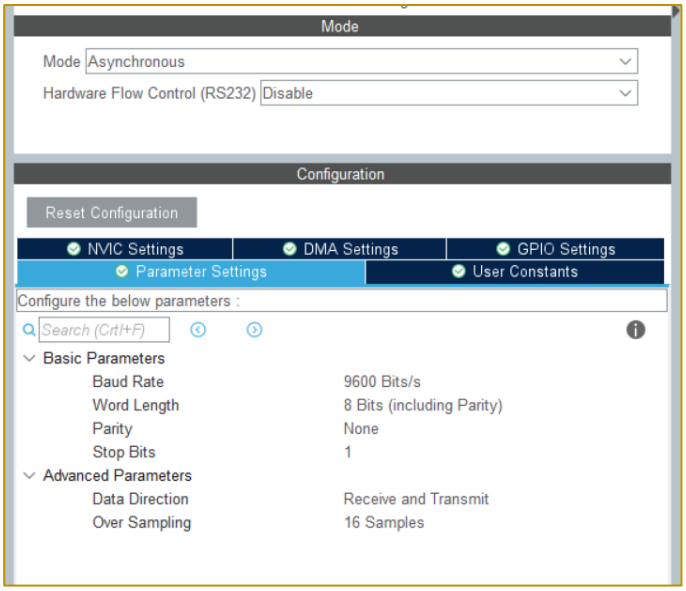
چکیده :

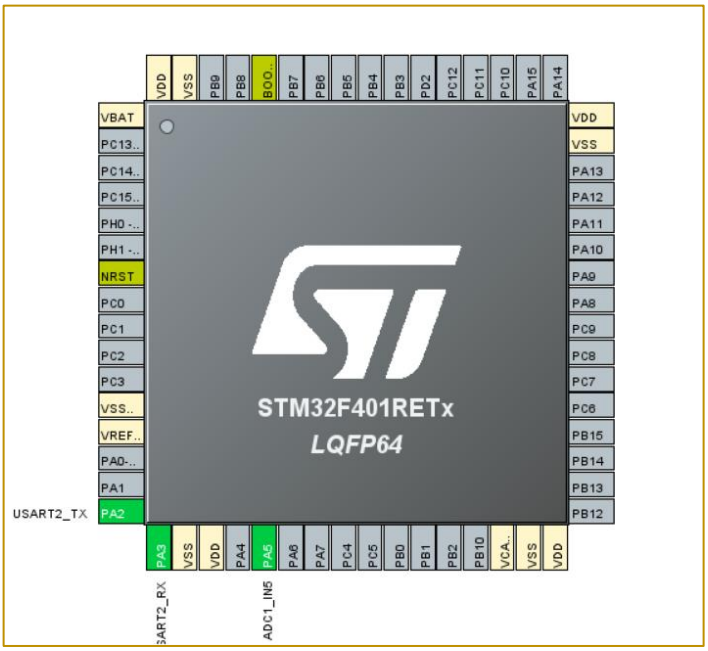
در این پروژه تلاش بر طراحی و پیاده سازی بخشی از یک سیستم کنترل کلید های هوشمند بود.

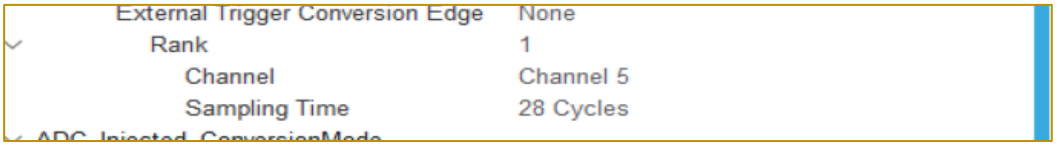
این سیستم از ۳ریزپردازنده ی مجزا تشکیل شده است و همان طور که در طرح کلی زیر مشاهده میکنید بخش اول که همان گره master است با کمک stm32cube و کتاب خانه HAL ، گره دوم که یکی از slave هاست با کمک کتابخانه CMSIS و نهایت گره اخر توسط زبان اسمبلی ۸۰۸۶ برنامه ریزی شده اند.

شرح مسیر کلی و مروری بر چالش های اولیه :

همان گونه که خواسته شد در ابتدا سعی کردیم بین دو ریزپردازنده STM32F401 یک ارتباط UART پایه برقرار کنیم. ابتدا با کمک منابعی که در درس آزمایشگاه برای یک تمرین UART داده شده بود به تنظیمات اولیه گره master پرداختیم که تصاویر مرحله به مرحله را مشاهده میکنید:







لازم به توضیح است که کتاب خانه HAL هنگام شبیه سازی در پروتئوس کمی مشکل دارد و لازم است خطی که به تنظیم مقدار baudrate میپردازد را کامنت کرده و با CMSIS خودمان بنویسیم. این مشکل در تمرین هم بود و جهت رفع آن همین راهکار را انجام دادیم :

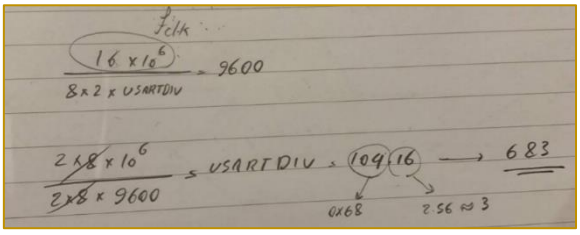
```
huart2.Instance = USART2;
//huart2.Init.BaudRate = 9600;
huart2.Init.WordLength = UART_WORDLENGTH_8B;
huart2.Init.StopBits = UART_STOPBITS_1;
huart2.Init.Parity = UART_PARITY_NONE;
huart2.Init.Mode = UART_MODE_TX_RX;
huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart2.Init.OverSampling = UART_OVERSAMPLING_16;
```

```
MX_GPIO_Init();
MX_USART2_UART_Init();
USART2->BRR = 0x0683;    /* 960
MX_ADC1_Init();
/* USER CODE BEGIN 2 */
```

```
UART_Transmit(adcValue_i);
/* USER CODE END WHILE */

//HAL_UART_Transmit(&huart2, (uint8_t *)"2", 1, 100);
```

سپس در تابع زیر تنظیمات مناسب برای میکرو دوم را انجام دادیم. نکته مهم این است که وقتی استاپ بیت ۱ باشد مشکلاتی در ارسال و دریافت وجود دارد ولی وقتی با آزمون و خطا ان را ۲ عدد قرار دادیم مشکلات حل شد. alternate فانکشن مناسب را با کمک دیتاشیت فعال شده و رجیسترهارا برای Rx Tx مقدار دهی کردیم. روش محاسبه baudrate هم در تصویر زیر آمده است. کامنت گذاری های مناسب هم در تصویر موجود است.

```
void USART2_init (void) {
  RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;  /* enable GPIOA clock */
  RCC->APB1ENR |= 0x20000;              /* enable USART2 clock */
  /* Configure PA2 for USART2_TX */
  GPIOA->OSPEEDR |= 0x20;
  GPIOA->OSPEEDR |= 0x80;
  GPIOA->AFR[0] &= ~0x0FF00;
  GPIOA->AFR[0] |= 0x07700; /* alt7 for USART2 */
  GPIOA->MODER  &= ~0x00F0;
  GPIOA->MODER  |= 0x00A0; /* enable alternate function for PA2,PA3*/
  USART2->BRR = 0x0683; /* 9600 baud @ 16 MHz */
  USART2->CR1 |= 0x0008; /* enable Tx, 8-bit data */
  USART2->CR1 |= 0x0004; /* enable Rx, 8-bit data */
  //USART2->CR1 |= USART_CR1_TCIE;
  USART2->CR2 |= (2 << 12); /* 1 stop bit */
  USART2->CR3 = 0x0000; /* no flow control */
  USART2->CR1 |= 0x2000; /* enable USART2 */

}
```

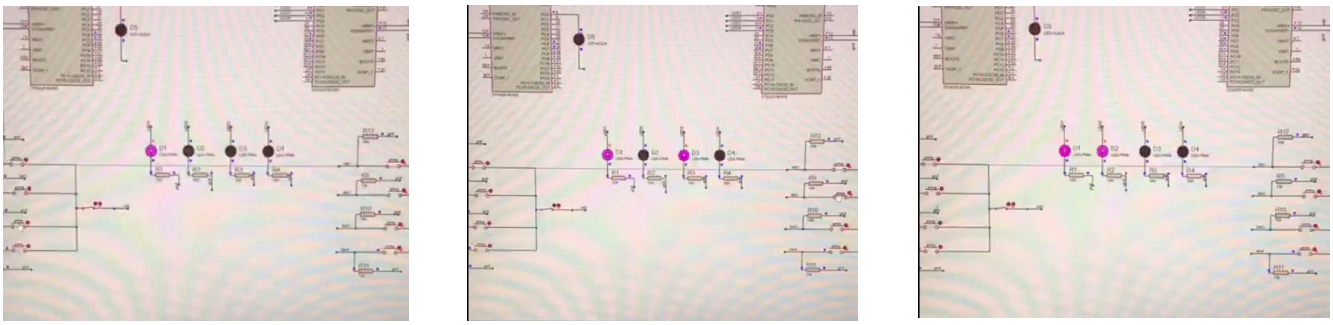


این بخش هم برای دریافت داده به صورت امتحانی گذاشتیم و تست کردیم کار میکرد.

```
char USART2_receive(){
  while (!READ_BIT(USART2->SR, USART_SR_RXNE)){}
  x = (uint8_t)(USART2->DR);
```

در این مرحله موفق شدیم یک ارتباط UART ساده پیاده سازی کنیم که نتیجه را در تصاویر زیر میبینید :

اگر تصویر نمایش داده نمیشود فیلم کوتاه و کم حجم از اینجا هم قابل مشاهده است.



فعالیت انجام شده در بخش Master با HAL:

در ابتدا به توضیح متغیر ها به ترتیب میپردازیم، هر یک از متغییر ها به ترتیب وظایف زیر را دارند :

```
43  volatile int state_sel = 0;
44  volatile int leds[4][4] = {0} ;
45  volatile uint16_t seven_seg[3] = {0x003F , 0x0006, 0x005B};
46  volatile uint8_t data_frames[4] = {'\0'};
47  volatile uint8_t read_frames[2] = {'\0'};
48  char RxBuffer[1] = {0};
49  volatile uint16_t GPIO_PINS[8] = {
50
51    GPIO_PIN_0, GPIO_PIN_1,
52    GPIO_PIN_2, GPIO_PIN_3,
53    GPIO_PIN_4, GPIO_PIN_5,
54    GPIO_PIN_6, GPIO_PIN_7
55
56  };
```

۱. وظیفه تشخیص گره های slave را دارد
۲. یک ارایه ۲ بعدی که هر ردیف آن چراغ های یکی از گره هاست.
۳. مقادیر اولیه سون سگمنت
۴. برای نگه داری فریمی که قرار است ارسال شود و ۳ بایت خواهد داشت.
۵. فریمی که قرار است به گره slave بگوید دیتا را ارسال کن

۶. بافر برای دریافت وضعیت slave و نگه داری آنها

۷. ارایه ی شامل پین های برد.

در ادامه کاربرد در کد را خواهیم دید.

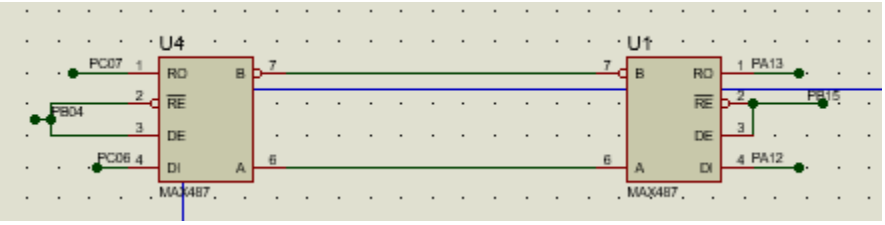
با کمک تابع زیر در main میتواند led های داخل master را کنترل کرد.

```
void write_master_leds(int count){
  for (int i= 0; i<count; i++){
    HAL_GPIO_WritePin(GPIOB, GPIO_PINS[i], leds[state_sel][i]);
  }

}
```

دو تابع هم برای در اختیار گرفتن و آزاد کردن باس در نظر گرفتیم تا بتوانیم داده ها را ارسال کنیم. در واقع همانطور که در مدار مشخص است کمک میکند تا سیم PB04 را صفر و یک کنیم.

در حالت پایه ۰ است و وقتی بخواهیم بنویسیم ۱ میکنم.

```
void assert_bus(){
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_SET);
}
void disassert_bus(){
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_RESET);

}
```



این تابع همانطور که از نامش پیداست، یک بافر با تایپ unsigned char از تایپ های uint_8 میسازد. در واقعا تمام فریم هارا به هم وصل میکند.

```
void create_dataframe(uint8_t addr, uint8_t led_number, uint8_t data){
  sprintf(data_frames, "%hhu%hhu%hhu", addr, led_number, data);
}
```

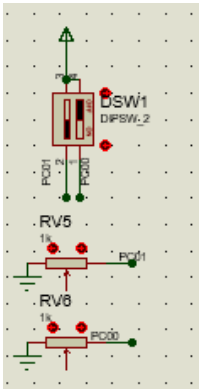
در تابع call back که از کتابخانه HAL است، زمانی فراخوانی میشود که دکمه های متصل شده به وقفه خارجی فشار داده شوند.

در این تابع ابتدا با کمک یک حلقه به جای اجرای چند شرط متوالی کلیدی که فشار داده شده بود را تشخیص دادیم و led مربوط به آن را toggle کردیم تا اگر روشن بود خاموش و اگر خاموش بود روشن گردد.

در ادمه مقادیر فریم ها مشخص میشود مثلا rw=1 یعنی در مود نوشتن هستیم. در فریم آخر دیتای مورد نظر (شماره led که قرار است toggle شود) را میگذاریم. سپس بررسی میکنیم که اگر سوییچ دیوواس ۱ را نشان میداد داده ها را ارسال کنید.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{

  uint16_t GPIOC_PINS[4] = {GPIO_PIN_2, GPIO_PIN_3, GPIO_PIN_4, GPIO_PIN_5};

  uint8_t addr = 0 ;
  uint8_t rw = 0 ;
  uint8_t data = 0;
  state_sel =  GPIOC->IDR & 0x3;

  for (int i= 0; i<4; i++){
    if(GPIO_Pin == GPIOC_PINS[i]){

      leds[state_sel][i] ^= 1;  //togle selected led in the slave

      //assign values to frames
      addr = state_sel ;
      rw =  1;
      data = i;
      create_dataframe(addr, rw, data);
      //send
      if (state_sel == 1){
      assert_bus();
      HAL_UART_Transmit(&huart6, data_frames, sizeof(data_frames), 100);
      disassert_bus();
      }
    }
  }
}
```

دلیل استفاده از این سوییچ این بود که ما ۳ حالت کلی داشتیم و نزدیک ترین دیکدر۴ تایی است. حالت ۰ یعنی هیچ دیوایسی سلکت نشده، حالت ۱ برای گره slave1 و حالت ۲ برای گره slave 8086



ابتدا در main ، پین ۴ را deassert میکنیم تا reset شود. سپس در داخل حلقه while ، تمام پین های مربوط به 7seg را پاک میکنیم. بعد با توجه به این که کدام دیوایس سلکت شده مقدار مربوطه را بر روی پین 7seg قرار میدهیم.

در انتهای main نیز دائما استیت دیوایس slave1 را دریافت و داخل بافر rx میریزیم که در بالا گفته شد سپس آنرا به int تبدیل میکنیم. در ادامه اتفاقات به میکرو بعدی منتقل میشویم که توضیح خواهیم داد. (مراجعه شود به بخش بعدی توضیحات وقفه ها)

```
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */

  CLEAR_BIT(GPIOA->ODR, 0x7F); //clear PA0-PA6
  SET_BIT(GPIOA->ODR, seven_seg[state_sel]);

  write_master_leds(4);


  HAL_UART_Receive(&huart6, (uint8_t*)RxBuffer, 1, 100);
  rcv = RxBuffer[0] - '0';
  if ((rcv > 3 && rcv < 8) && state_sel == 1){
    leds[state_sel][rcv - 4] ^= 1;

  }
```

فعالیت انجام شده در بخش Slave با CMSIS :

این تابع همانطور که از نامش پیداست برای برقراری UART است که طبق اسلاید های شما زدیم.

```
void UART_Transmit(char digit){
  USART2->DR = digit;
  while(!READ_BIT(USART2->SR, USART_SR_TC)){}
}
```

تابع usart2_init در بخش پیاده سازی اولیه توضیح داده شد و دقیقا همان تابع را در ادامه به کار گرفتیم :

```
void USART2_init (void) {
  RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
  RCC->APB1ENR |= 0x20000;
  // Configure PA2 for USART2_TX
  GPIOA->OSPEEDR |= 0x20;
  GPIOA->OSPEEDR |= 0x80;
  GPIOA->AFR[0] &= ~0x0FF00;
  GPIOA->AFR[0] |= 0x07700;
  GPIOA->MODER  &= ~0x00F0;
  GPIOA->MODER  |= 0x00A0; //enable alternate function for PA2,P
  USART2->BRR = 0x0683; // 9600 baud @ 16 MHz
  USART2->CR1 |= 0x0008; // enable Tx, 8-bit data
  USART2->CR1 |= 0x0004; // enable Rx, 8-bit data
  //USART2->CR1 |= USART_CR1_TCIE;
  USART2->CR2 |= (2 << 12); // 2 stop bits
  USART2->CR3 = 0;  // flow cntrl

  USART2->CR1 |= 0x2000;

}
```

تابع USART2_receive که در قبل توضیح دادم به حالت ز یر تغییر پیدا کرد چرا که در حال ارسال و دریافت ۳ فریم هستیم پس باید متناسب با ان توابع مربوطه تغییر پیدا کنند.

نکته قابل توجه این است که اگر فریم دوم که command است ۱ باشد، دیتا بخواند.

```
void USART2_receive(){
  while (!READ_BIT(USART2->SR, USART_SR_RXNE)){}
     address = (uint8_t)(USART2->DR);
  while (!READ_BIT(USART2->SR, USART_SR_RXNE)){}
     command = (uint8_t)(USART2->DR);
  if (command == '1'){
   while (!READ_BIT(USART2->SR, USART_SR_RXNE)){}
     data = (uint8_t)(USART2->DR);
   }
  //else

}
```

این تابع خیلی ساده است و برای تمیزی کد تابع شده، وظیفه آن صرفا toggle کردن شماره led خوانده شده است.

```
void change_state(){
  int value = data - '0';
  GPIOB->ODR ^= MASK(value);
  data = '9';

}
```

در ابتدا، هنگامی که پروژه را از پایه پیاده میکردیم، کلید ها را به صورت polling زده بودیم تا ساده تر باشد

ولی در ادامه مشاهده کردیم حجم کد زیاد شد و سرعت آن بسیار پایین آمد به خصوص که خود پروتئوس در مدارات بزرگ کندتر کار میکند لذا تصمیم گرفتیم سیستم را کلا با وقفه پیدا سازی کنیم. برای پیاده سازی وقفه هم ابتدا باس های مناسب در تابع GPIO_init و تنظیمات اولیه در تابعEXTI_init انجام شده است.

مثلا ۴ لاین وقفه فعال شده، اولیت ها ست شده، IMR و PTSR و .. تنظیم شده است.

سپس باس را برای ارسال دیتا در اختیار میگیریم و مقدار ۴ را به سمت master میفرستیم. تا با پین های led تداخل پیدا نکند. منظور از ارسال "4" این است که در سمت دیگر تغییرات slave به masterگزارش شود و درسمت دیگر با عملیت ریاضی ساده led نگاشت شده toggle میشود.

برای نمونه برای هندل کردن وقفه لاین صفر :

```
void EXTI0_IRQHandler(void){

  EXTI-> PR |= MASK(0);
  NVIC_ClearPendingIRQ(EXTI0_IRQn);

  if(GPIOC->IDR & MASK(0)){
    GPIOB->ODR ^= MASK(0);

    GPIOB->ODR |= MASK(5);
    UART_Transmit('4');
    GPIOB->ODR &= ~MASK(5);


  }
```

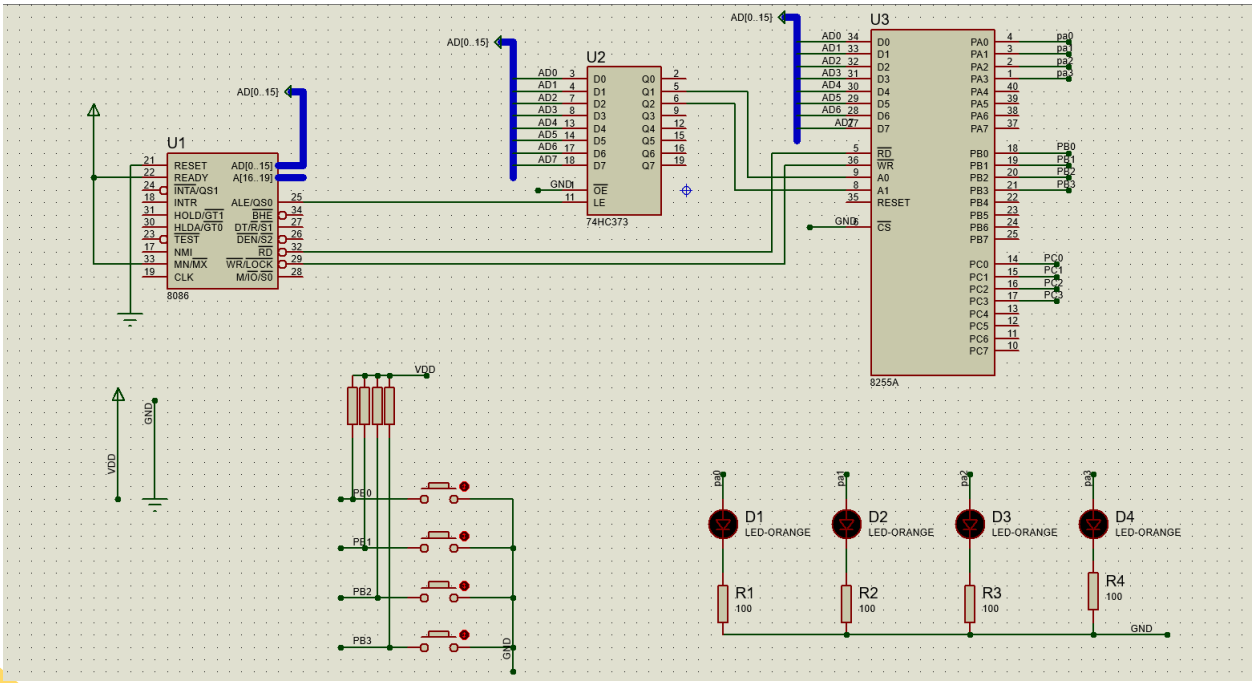
در انتها هنگام اجرا توابع مقدار دهی اولیه صدا میشوند و پین مربوط به assert کردن max487 را ریست کرده و وارد حلقه while میشویم.

```
int main(){

    GPIO_init();
    EXTI_init();

    USART2_init();
    GPIOB->ODR = 0;
    GPIOB->ODR &= ~MASK(5);
    while(1) {
      USART2_receive();
      change_state();
    }

}
```

فعالیت انجام شده در بخش ۸۰۸۶ :

این بخش پرچالش ترین بخش پروژه برای ما بود و بیشتر زمان ما بر این بخش صرف شد. در ادامه یافته ها را شرح خواهیم داد.

ابتدا سعی کردیم در یک پروژه مجزا سعی کنیم یک led ساده با 8086 روشن و خاموش کنیم. برای این کار ابتدا قطعات لازم را همانطور که در اسلایدها آموزش دادید به هم متصل کردیم مدار حاصل در صفحه ی بعدی قابل مشاهده است. ابتدا یک led و کلید داشت و در مرحله بعد ۳ تای دیگر به آن اضافه کردیم و با کمک دستورات branch در زبان اسمبلی روشن و خاموشی کلید ها را هندل کردیم.



در باره کد هم کامنت مناسب گذاشته شده است. ولی در کل آمدیم برای هر کلید یک استیت گرفتیم تا اگر فشرده شد عملیات مناسب انجام شود.

برای اینکه فشرده شدن هر کلید روی دیگران تاثیر نگذارد تصمیم گرفتیم در رجیستر BL در ابتدا همه led ها را خاموش بگیریم و هر بار هر کلید که فشار داده شد با آدرس مناسب BL را XOR کنیم. با این روش مقدار قبلی حفظ میشود و Led مرتبط با کلید هم اگر خاموش باشد، روشن و اگر روشن بود خاموش میشود. برای مثال در STATE0 ، مقدار آدرس مناسب برای همه پین ها صفر و برای پین PC0 مقدار ۱ است :

```
24  JMP _LOOP
25  _LOOP:
26  MOV DX, PORTB
27  IN AL, DX
28  CMP AL, 11111110B  ;if btn pb0 pressed then go to state0
29  JZ STATE0
30  IN AL, DX
31  CMP AL, 11111101B  ;if btn pb1 pressed then go to state1
32  JZ STATE1
33  IN AL, DX
34  CMP AL, 11111011B  ;if btn pb2 pressed then go to state2
35  JZ STATE2
36  IN AL, DX
37  CMP AL, 11110111B  ;if btn pb3 pressed then go to state3
38  JZ STATE3
39  JMP DELAY
40  JMP _LOOP ;IF NO BTN PRESSED
```

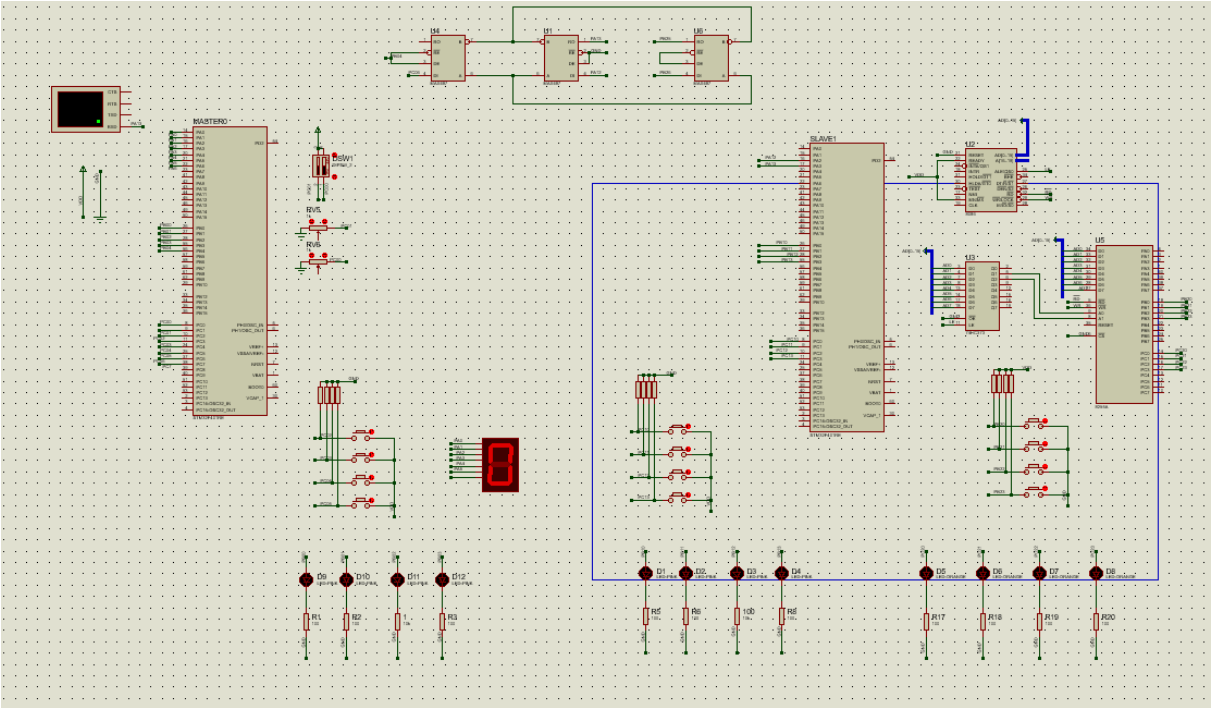
بقیه استیت ها هم به همین ترتیب.

```
43  STATE0:
44   XOR  BL, 11000001B
45   MOV AL, BL
46   MOV DX, PORTA
47   OUT DX,AL
48   JMP   LOOP
```

یک delay کوچک هم در انتهای حلقه جهت تنظیم سرعت انسان و میکرو گذاشتیم تا انهای هر حلقه یک تاخیر خیلی کم داشته باشیم و کلید فشرده شده فقط یکبار detect شود و نه بیشتر

```
70  DELAY:
71  MOV CX, 00FFH
72  P0 :
73  DEC CX
74  JNE P0
75  JMP _LOOP
76  JMP START
```

در نهایت مدار تکمیل شده را به مدار اصلی پروژه منتقل کردیم و فایل exe را در میکرو۸۰۸۶ قرار دادیم :



نکته : لازم به ذکر است وقتی مدار خیلی بزرگ میشود پرتئوس در شبیه سازی کند میشود و گاهی کلید هارا نمیگیرد، برای همین ما جدا مدار ۸۰۸۶ و کدش را در یک پوشه برای شما میگذاریم تا اگر خواستید تست کنید به مشکل برنخورید.

جدول ادوات جانبی و پین های ۸۰۸۶ :

8086	
pin	**task**
PB[0..4]	Push-Button Detection
PC[0..4]	LED Blinking
chip	**task**
74HC373	Latch
8255	IO Interfacing

جدول پین های Master :

در تمارین قبلی همانطور که در جریان هستید برخی پین های مربوط به فانکشن ها مثلا تایمر ، ADC و UART در کتابخانه HAL به درستی کار نمیکردند. برای مثال UART 2 مشکل دارد و آن زمان جناب مهندس بی طالبی گفته بودند اگر مشکل داریم با این اعداد کار کنیم :

Port C	PC6	-	--	TIM3_CH1	-	-	I2S2_MCK	-	-	USART6_TX
	PC7	-		TIM3_CH2	-	-	-	I2S3_MCK	-	USART6_RX
	PC8	-	-	TIM3_CH3	-	-	-	-	-	USART6_CK

Master Node (HAL)	
pin	**task**
PC6	UART6_RX
PC7	UART6_TX
PB[0..3]	LED[0..3]
PA[0..6]	7SEG
PC[0..5]	EXTI -> (BTN and SWITCH)

جدول پین های Slave1 :

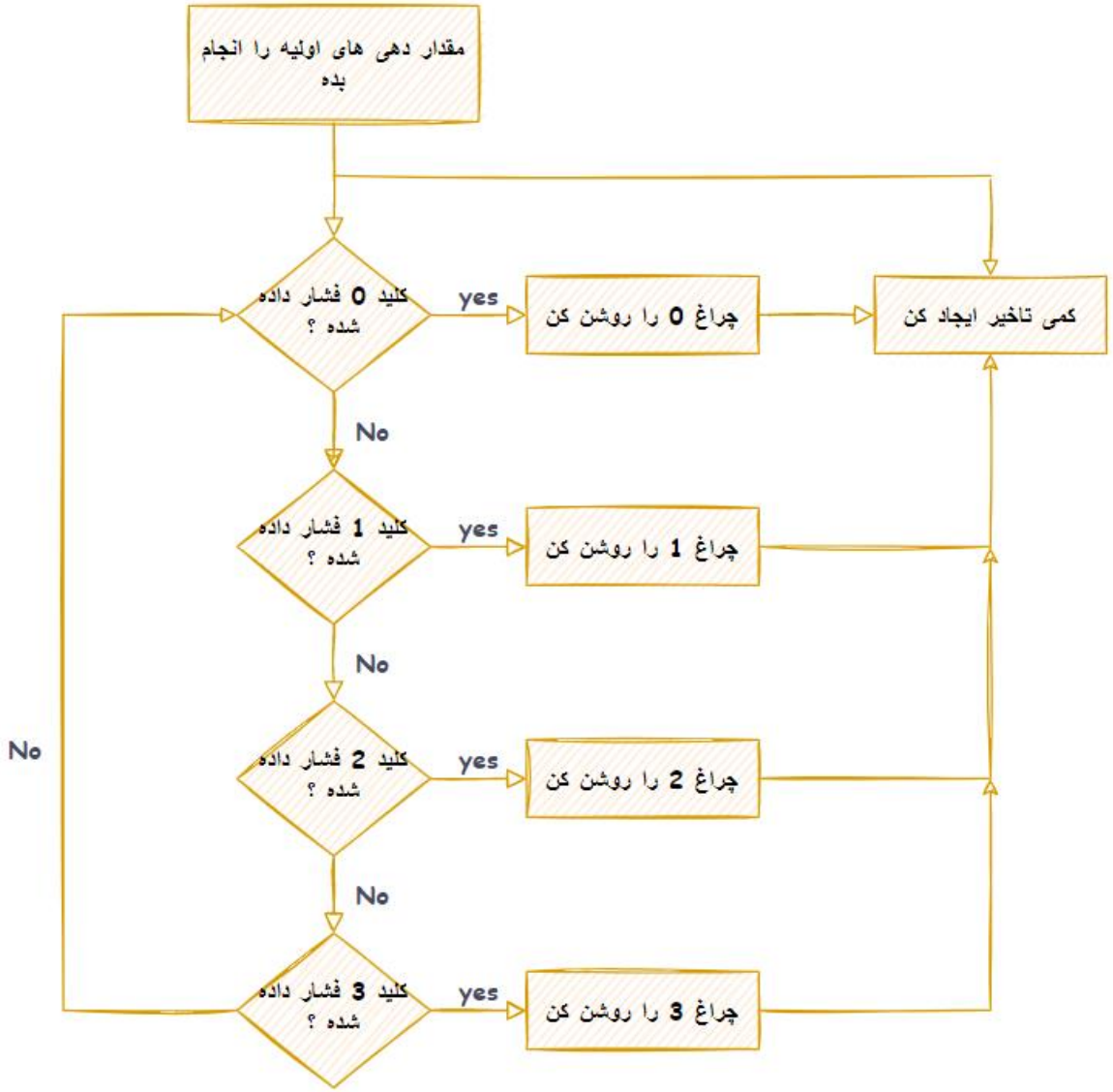
در CMSIS دیگر مشکلات HAL نبود و دستمان باز تر بود :

PA2	-	TIM2_CH3	TIM5_CH3	TIM9_CH1	-	-	-	USART2_TX
PA3	-	TIM2_CH4	TIM5_CH4	TIM9_CH2	-		-	USART2_RX

Slave1 Node (CMSIS)	
pin	**task**
PA2	UART6_TX
PA3	UART6_RX
PB[0..3]	LED[0..3]
PC[0..3]	PUSH-BUTTON

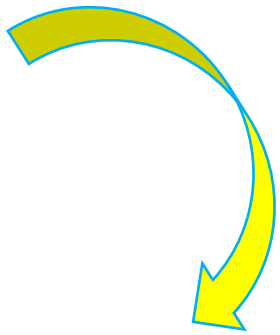
شبه کد ها و فلوچارت :

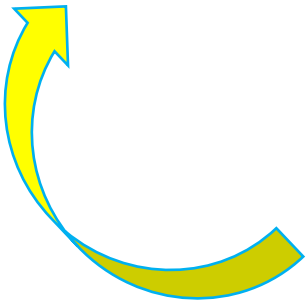
فلوچارت برای کد ۸۰۸۶ :



برای بخش های ARM

```
MASTER(0):
 if button is pressed:
 ISR:
  check button number -> find corresponding LED -> LED#
  change LEDs state
  set address byte = device selected
  set read/write byte = 1 (write)
  set data byte = LED#
  transmit address, read/write, data bytes to SLAVE
 MAIN:
  LOOP:
   update LEDs state
   listen for data from SLAVE
   if data comes -> data = LED#
   toggle LED at LED#
```





```
SLAVE(1):
 listen for packets from MASTER
 receive packets from MASTER
 packets: (address, r/w, LED#)
 if r/w is '1':
  toggle LED at LED#

 if button is pressed:
 ISR:
  check button number -> find
corresponding LED -> LED#
  toggle LED at LED#
  send LED# to MASTER
```

نکته تکمیلی :

در جهت اتصال UART برای ۸۰۸۶ هم مطالعات زیادی کردیم و تلاش هم کردیم اما متاسفانه دیتایی که فرستاده میشد دریافت نمیشد و تلاش های این بخش تقریبا بی ثمر ماندند.

پایان. 😊