



گزارش آزمایشگاه

رشته مهندسی برق

گزارشکار سری چهارم آزمایشگاه ریزپردازنده و مدارهای واسطه

نگارش

امیرحسین منصوری

مehشاد اکبری سریزدی - ۹۹۲۳۰۹۳

آنوشا شریعتی - ۹۹۲۳۰۴۱

استاد راهنما

مهندس ذکی زاده

آبان ۱۴۰۲ شمسی

## آزمایش سوم - بخش ۲:

در این بخش به پیاده سازی مانند قسمت قبل اما با دستوراتی متفاوت برپایه تعریف شده بودن رجیستر ها می پردازیم . در **moder** هر پین دو بیتی می باشد و بر اساس آن و دستور کار **syntax** های مربوط به یک کردن هر بیت را برای **moder** و **BSRR** به صورت زیر پیاده سازی کردیم. با توجه به بخش اول آزمایش بیت های ۱۲ تا ۱۵ رجیستر **BSRR** و **MODER** را یک میکنیم زیرا **led** های به پایه های ۱۲ تا ۱۵ پرفرال **GPIO** متصل هستند .

```
system_stm32... main.c startup_stm32... stm32f4xx_it.c
93 /* USER CODE BEGIN SysInit */
94 /* USER CODE END SysInit */
95
96 /* Initialize all configured peripherals */
97 MX_GPIO_Init();
98 /* USER CODE BEGIN 2 */
99
100 RCC_AHB1ENR |= (1<<3);
101
102 // SET_BIT(GPIOB_BSRR, (1<<12)|(1<<13)|(1<<14)|(1<<15));
103 //SET_BIT(GPIOB_MODER, (1<<24)|(1<<26)|(1<<28)|(1<<30));
104
105
106 GPIOB->MODER |= GPIO_MODER_MODER12_0;
107 GPIOB->MODER |= GPIO_MODER_MODER13_0;
108 GPIOB->MODER |= GPIO_MODER_MODER14_0;
109 GPIOB->MODER |= GPIO_MODER_MODER15_0;
110 GPIOB->BSRR |= GPIO_BSRR_BS_12;
111 GPIOB->BSRR |= GPIO_BSRR_BS_13;
112 GPIOB->BSRR |= GPIO_BSRR_BS_14;
113 GPIOB->BSRR |= GPIO_BSRR_BS_15;
114
115
116 /* USER CODE END 2 */
117
118
119 /* Infinite loop */
```



در تصویر بالا هم صحت کد را مشاهده می کنیم که بیت ها را به درستی یک کرده است و **led** ها روشن شدند.

### آزمایش سوم - بخش ۳:

در این جا به تغییر مقدار رجیستر ها در پنجره SFR به صورت زیر پرداختیم و با تغییر دادن بیت های مربوط به پایه های LED تغییرات را در روشن و خاموش شدن LED های مختلف روی برد مشاهده کردیم .

The screenshot shows an IDE with a C file named `stm32f4xx_it.c` and a peripheral register view on the right.

**Code Snippet:**

```

93  /* USER CODE BEGIN SysInit */
94
95  /* USER CODE END SysInit */
96
97  /* Initialize all configured peripherals */
98  MX_GPIO_Init();
99  /* USER CODE BEGIN 2 */
100
101  RCC_AHB1ENR |= (1<<3);
102
103  // SET_BIT(GPIOD_BSRR, (1<<12)|(1<<13)|(1<<14)|(1<<15));
104  //SET_BIT(GPIOD_MODER, (1<<24)|(1<<26)|(1<<28)|(1<<30));
105
106  GPIOD->MODER |= GPIO_MODER_MODER12_0;
107  GPIOD->MODER |= GPIO_MODER_MODER13_0;
108  GPIOD->MODER |= GPIO_MODER_MODER14_0;
109  GPIOD->MODER |= GPIO_MODER_MODER15_0;
110  GPIOD->BSRR |= GPIO_BSRR_BS_12;
111  GPIOD->BSRR |= GPIO_BSRR_BS_13;
112  GPIOD->BSRR |= GPIO_BSRR_BS_14;
113  GPIOD->BSRR |= GPIO_BSRR_BS_15;
114
115
116  /* USER CODE END 2 */
117
118
119  /* Infinite loop */

```

**Peripheral Register View:**

Register	Address	Value
MODER	0x40020...	0x55000000
MODER15 [30:2]		0x1
MODER14 [28:2]		0x1
MODER13 [26:2]		0x1
MODER12 [24:2]		0x1
MODER11 [22:2]		0x0
MODER10 [20:2]		0x0
MODER9 [18:2]		0x0
MODER8 [16:2]		0x0
MODER7 [14:2]		0x0
MODER6 [12:2]		0x0
MODER5 [10:2]		0x0
MODER4 [8:2]		0x0

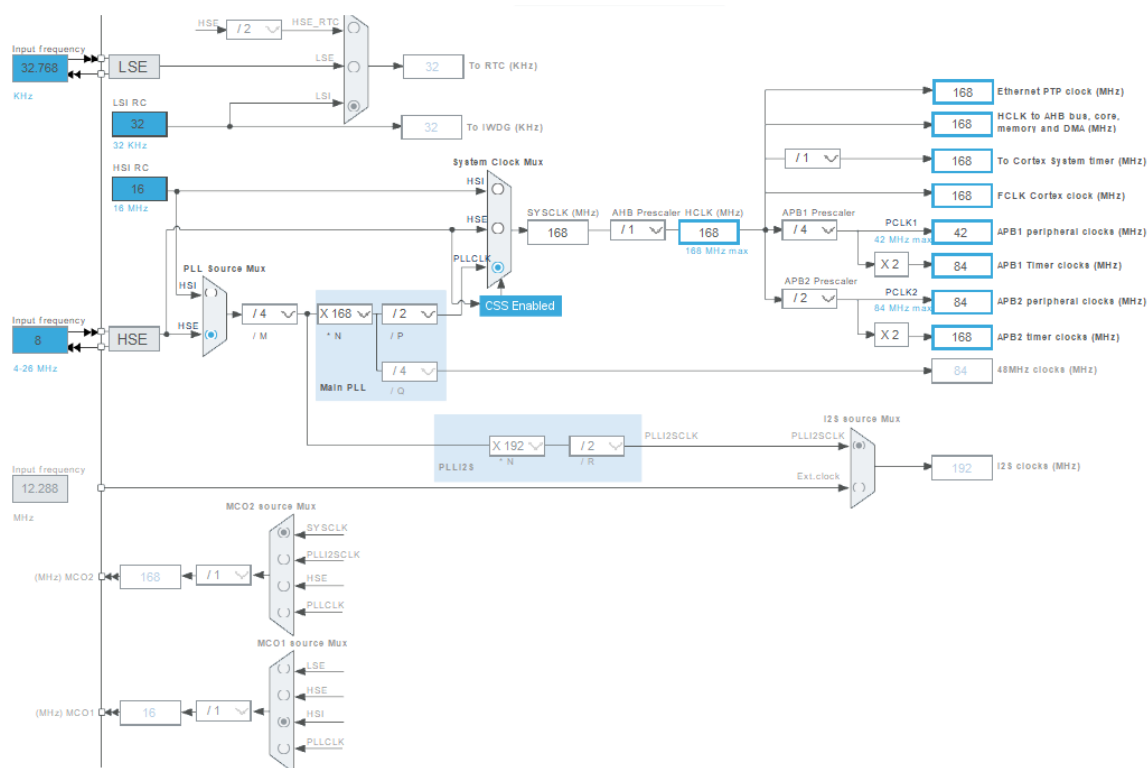
Bit field details for MODER15:

- Bit field: MODER15
- LSB: 30
- MSB: 32
- Size: 2
- Reset value: 0x0

## آزمایش سری ۴:

برای شروع کار مدل پردازنده (STM32F407VGT6) از منوی MCU انتخاب می‌کنیم.

سپس برای اینکه بتوانیم از میکرو استفاده کنیم و از مزایای Debugging آن بهره‌مند شویم باید تغییراتی را در کلاک دستگاه اعمال کنیم. برای تنظیم HSE دستگاه از یک نوسان‌ساز خارجی با فرکانس کاری ۸ MHz استفاده می‌کنیم، سپس تامین کلاک پردازنده را از PLL تعیین کرده و منبع آنرا HSE قرار می‌دهیم. حداکثر فرکانس کاری این پردازنده ۱۶۸ MHz می‌باشد. با قراردادن فرکانس کاری سایر تغییرات به صورت اتوماتیک انجام خواهد شد.



پس از این برای بهره‌مندی از امکانات دیباگینگ، مد دیباگ را به Trace Asynchronous Sw تغییر می‌دهیم.

Mode

Debug
Trace Asynchronous Sw

☐ System Wake-Up

Timebase Source
SysTick

حال که اعمال مقدماتی را انجام دادیم، کافی است گزینه Generate-code را فشار دهیم تا سایر تنظیمات توسط نرم‌افزار انجام شود.

### بخش اول:

در این بخش ابتدا با استفاده از دستور MOV برای انتقال عدد هگزادسیمال 0xFF به داخل رجیسترهای عمومی یک و دو استفاده می‌کنیم. و سپس محتویات رجیستر دوم را دوازده بار شیفت حسابی می‌دهیم. نکته قابل توجه در این بخش، قراردادن کدهای اسمبلی پیش از حلقه تکراری میکروکنترلر است.

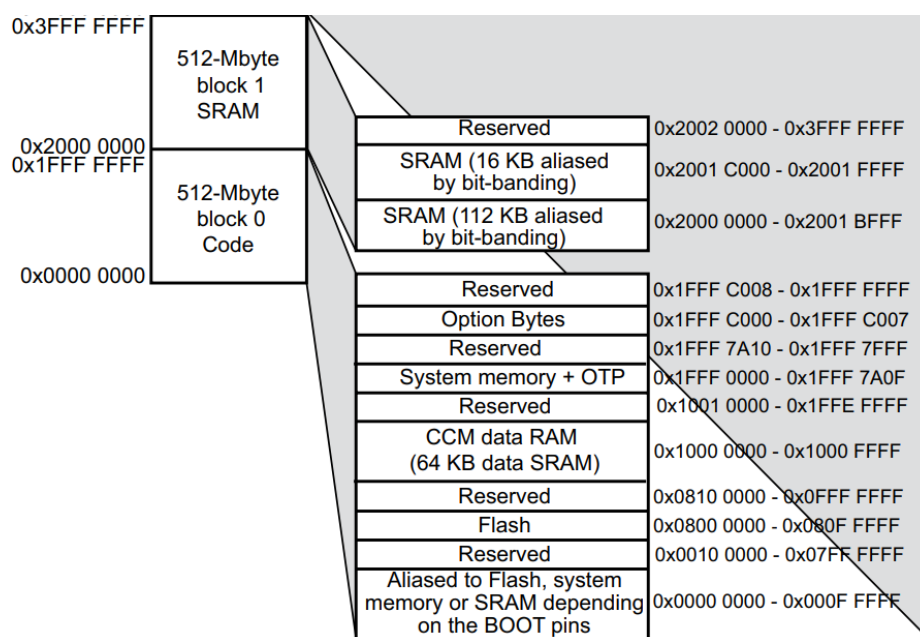
```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART2_UART_Init();
/* USER CODE BEGIN 2 */
__asm(
    "MOV R1, #0xFF \n"
    "MOV R2, #0xFF \n"
    "ASR R2, #12 \n"
);
/* USER CODE END 2 */
```

خروجی رجیسترها در مد دیباگ مطابق زیر است:

Name	Value	Description
General Registers		
r0	0	General Purpose and FPU Registers
r1	255	
r2	0	
r3	0	
r4	536871024	
r5	0	
r6	0	
r7	536969208	
r8	0	
r9	0	
r10	0	
r11	0	
r12	-520093696	
sp	0x20017ff8	
lr	134225175	
pc	0x80005cc <main+32>	

## بخش دوم:

در این بخش ابتدا به ساختار SRAM های موجود در برد موجود در آزمایشگاه می پردازیم. به طور کلی ریزپردازنده مذکور دارای ۲ حافظه اصلی با ظرفیت های ۱۶KB و ۱۱۲KB، یک باس برای ارتباط با SRAM های خارجی احتمالی و یک SRAM فرعی تحت عنوان BKP-SRAM می باشد. براساس دیتاشیت پردازنده، آدرس SRAM مطابق زیر است:



در ادامه ابتدا با استفاده از دستور `equ` آدرس های پایه SRAM و GPIOD را تعریف کرده، سپس مقادیر فوق را در رجیسترهای اول و چهارم قرار می دهیم تا از آنها برای آدرس دهی استفاده کنیم. می دانیم آدرس نسبی رجیستر `GPIO_IDR` مقدار هگزادسیمال `0x10` می باشد. با استفاده از آدرس پایه که در رجیستر اول بارگذاری شده و آدرس نسبی، مقدار رجیستر مذکور را در رجیستر صفرم بارگذاری می کنیم. در پایان مقدار رجیستر فوق را در SRAM که آدرس پایه آن در رجیستر چهارم ذخیره شده است ذخیره می کنیم.

```
__asm(
    ".EQU SR_ADDH, 0x2000    \n"
    ".EQU SR_ADDL, 0x0000    \n"
    ".EQU GD_ADDH, 0x4002    \n"
    ".EQU GD_ADDL, 0x0C00    \n"
    "MOV R4      , SR_ADDL    \n"
    "MOVT R4     , SR_ADDH    \n"
    "MOV R1      , GD_ADDL    \n"
    "MOVT R1     , GD_ADDH    \n"
    "LDR R0      , [R1,#0x10] \n"
    "STR R0      , [R4]       \n"
);
```

در مرحله بعد مقدار رجیستر مذکور را ابتدا از SFR می‌بینیم:

Register	Address	Value
> USART6		
> WWDG		
> DMA2		
> DMA1		
> GPIOH		
> GPIOE		
▼ GPIOD		
> MODER	0x40020c00	0x0
> OTYPER	0x40020c04	0x0
> OSPEEDR	0x40020c08	0x0
> PUPDR	0x40020c0c	0x0
> IDR	0x40020c10	0x0
> ODR	0x40020c14	0x0
> BSRR	0x40020c18	
> LCKR	0x40020c1c	0x0
> AFRL	0x40020c20	0x0

MSB 0

همچنین مقدار فوق که همان صفر است را می‌توان از خانه اول SRAM نیز مشاهده کرد:

Memory ×	
Monitors	
0x20000000	0x20000000 : 0x20000000 <Hex> ×
Address	0 - 3
20000000	00000000
20000010	00000000
20000020	00000000
20000030	00000000

با توجه به اینکه در این بخش هنوز پایه خاصی تنظیم نشده است، نتیجه فوق منطقی به نظر می‌رسد.