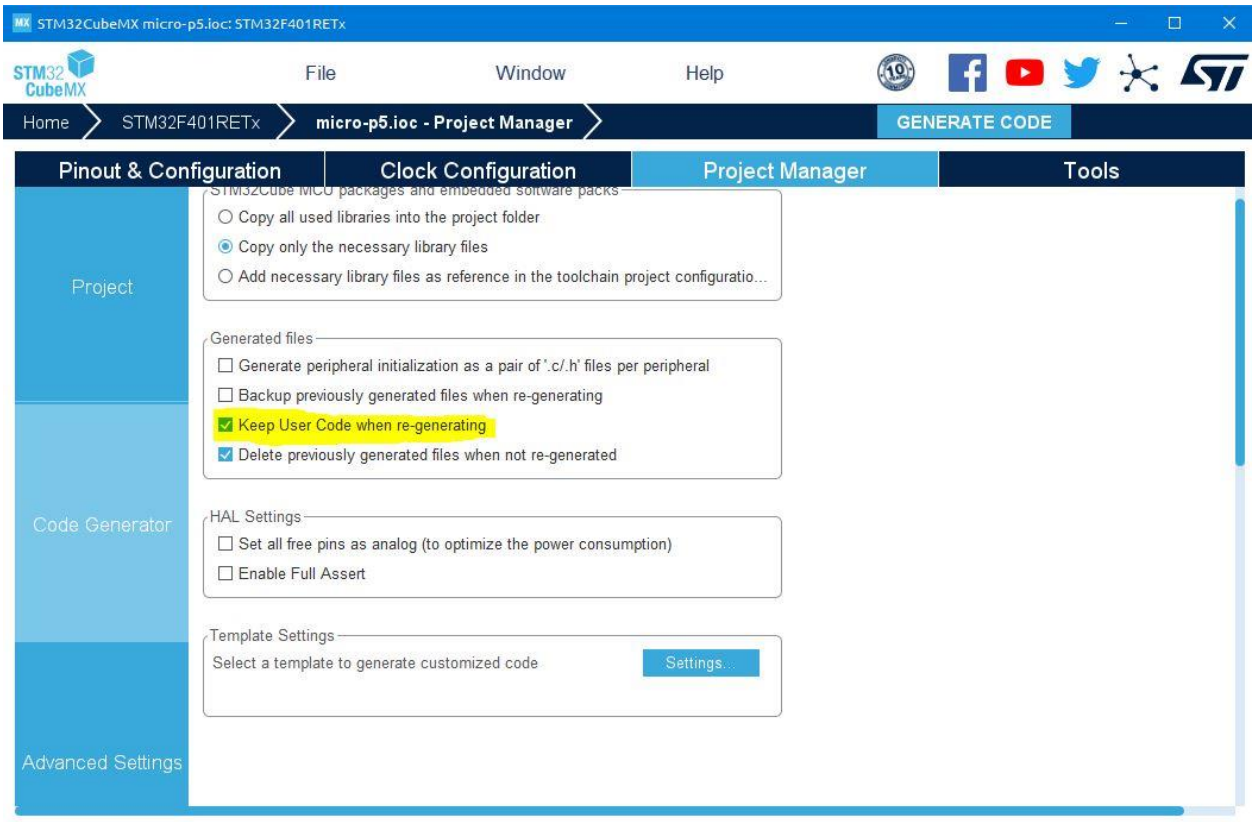
گزارش دستورکار سری پنجم ریزپردازنده

امیرحسین ادواری ۹۸۲٤۳۰۰٤ – زهرا حیدری ۹۸۲٤۳۰۲۰

بخش تحلیلی

سوال اول)

در cubemx قابلیتی وجود دارد که با فعالسازی آن هنگام تولید مجدد کد، کدهای اضافه شده کاربر از بین نمی‌روند و صرفا تغییرات جدید اعمال می‌شوند برای فعالسازی این قابلیت بایستی در تب Project Manager در قسمت Code Generator تیک مربوط به این فیچر فعال شود (در تصویر زیر هایلایت شده است) بدین‌ترتیب پس از اجرای مجدد فرایند تولید کد، کدهای توسعه داده شده فعلی از بین نمی‌روند.



سوال دوم)

این کلیدواژه قابلیتی در سطح کامپایلر است که امکان تعریف مجدد(Override) یک تابع را فراهم می‌آورد. به‌گونه‌ای که اگر تابعی به صورت __weak تعریف شود، مادامی که پیاده‌سازی جدیدی برای آن اضافه نشود از همان نسخه __weak استفاده خواهد شد اما درصورتیکه این تابع مجددا پیاده‌سازی شود از پیاده‌سازی جدید آن در اجرا استفاده خواهد شد.

بسیاری از توابع مربوط به interrupt ها (isr ها) در HAL به صورت __weak تعریف شده اند، این امر این امکان را به‌وجود می‌آورد که کاربر کتابخانه بدون تغییر کدهای کتابخانه پیاده‌سازی مربوط به ISR ها را بسته به نیاز خود بازتعریف کرده و مجددا متناسب با هدف‌مطلوب خود پیاده‌سازی کند.

سوال سوم)

سازوکار دسترسی و به‌کارگیری وقفه‌ها در HAL بدین‌گونه است که کاربر بایستی تابع

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)

را که توسط خود HAL به‌صورت __weak تعریف شده‌است را بازتعریف کند. ورودی این تابع عملا نشان‌دهنده پین‌ای می‌باشد که وقفه توسط آن صادر شده است، لذا توسط بازتعریف این تابع بسته به اینکه ورودی‌آن چیست، می‌توان از وقفه‌ها استفاده کرد. بطور مثال میتوان این تابع را به‌شکل زیر تعریف نمود:

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){

  if(GPIO_Pin == GPIO_PIN_0 ){
    systemState = COUNTING;
  }
  if(GPIO_Pin == GPIO_PIN_1 ){
    if(systemState == OFF)
      return;
    systemState = STOPPED;
  }
}
```

به‌عبارتی این روتین توسط آرگومان خود شناسایی می‌کند که کدام وقفه صادر شده، سپس اقدامات لازم برای هندل کردن آن را انجام می‌دهد.

ارتباط این سازوکار با CMSIS بدین‌گونه است که هندلرهای تعریف شده در CMSIS که به‌شکل

EXTIx_IRQHandler()

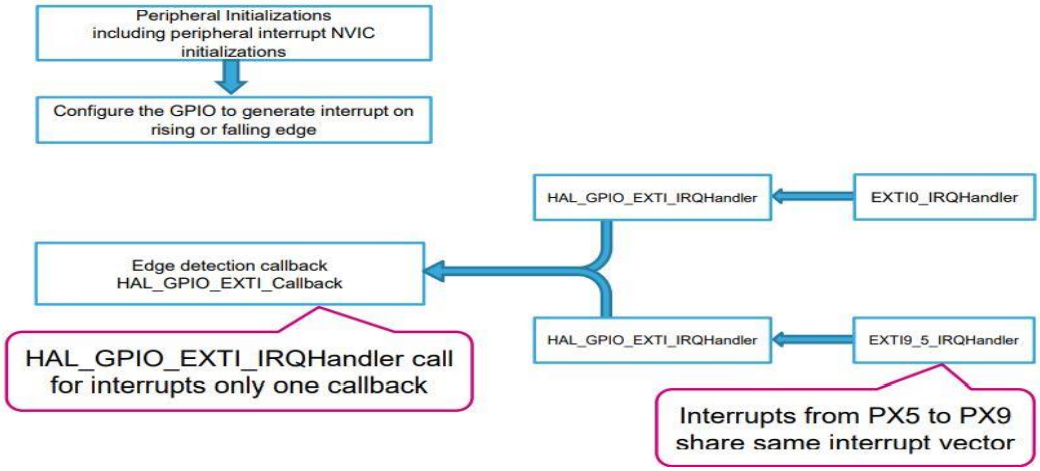
هستند، در بدنه خود هندلر تعریف شده در HAL را فراخوانی می‌کنند، و متناسب با اینکه هندلر کدام وقفه اجرا شده‌است (کدام EXTIx_IRQHandler کال شده است)؛ پارامتر متناظر با پین آن‌را به روتین مربوطه در HAL که به شکل

void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)

میباشد، پاس می‌کنند. این روتین نیز در دل خود پس از اجرای دستورات مربوط به تنظیم و کانفیگ خطوط وقفه(که در محیط گرافیگی cubemx قابل تعیین و تغییر است) ، روتین

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)

را با همان پارامتر ورودی خودش کال می‌کند.



(تصویر از اسلایدهای مربوط به HAL آپلودشده در درس‌افزار برداشته شده است)

بخش عملی

(بسیاری از توابعی که در انجام این پروژه استفاده شده‌است، در پروژه سوم شرح داده شده

(به خصوص توابع مربوط به LCD لذا از شرح مجدد آن‌ها می‌پرهیزیم)

ورودی‌خروجی‌ها

ورودی‌خروجی‌ها به شرح زیر هستند:

Symbol	PA 0	PA 1	PA 2	PA 3	PA 4	PA 5	PA 6	PA 7	PA 8	PA 9	PA 10	PB 0	PB 1	PB 2
Signal	D0	D1	D2	D3	D4	D5	D6	D7	RS	RW	E	B1	B2	B3
Type	O	O	O	O	O	O	O	O	O	O	O	I	I	I

O = output, I = input (inputs have been set to pull-down)

خطوط B3 B2 و نیز B1 خطوطی هستند که وقفه صادر میکنند.

منطق کار بدین‌صورت است که، برای هر حالت در تایمر، یک State در برنامه تعریف می‌کنیم و با توجه به این State ها هنگام صدور فرمان‌ها، اقدامات متناسب را انجام می‌دهیم. که در ادامه شرح داده خواهد شد.

1. Defines

```
// GPIOA -> For outputs : ( 0 to 7 for DataBus )
#define RS (8)
#define RW (9)
#define E  (10)
// GPIOB -> For interrupt lines
#define B1 (0)
#define B2 (1)
#define B3 (2)
#define MASK(x)    (1UL << (x))
```

ورودی‌ها صرفا دکمه‌ها هستند و خروجی‌ها نیز سیگنال‌های ورودی به LCD که در پروژه 3 شرح داده شده‌اند.

2. Vars-States

```
volatile uint16_t msec = 0;
volatile uint16_t sec = 0;
volatile uint16_t min = 0;

// Timer States
enum State { STOPPED, COUNTING, STARTUP, OFF };
// Initial State
enum State systemState = STARTUP;
```

در این قسمت State های مختلف و نیز متغیرهای مربوط به ذخیره میلی‌ثانیه، ثانیه و دقیقه فعلی را تعریف می‌کنیم.

3. Long_push()

```
// Check whether pushing was long.
bool long_push(){
    HAL_NVIC_DisableIRQ(TIM2_IRQn);
    uint32_t start = HAL_GetTick();
    uint32_t delay = 300;

    bool long_push = true;

    while( (HAL_GetTick() - start) < delay){
            if(!B3_clicked()){
                long_push = false;
                break;
            }

        if(!long_push)
            break;
    }
    HAL_NVIC_EnableIRQ(TIM2_IRQn);

    return long_push;
}
```

این تابع با استفاده از Systickبررسی میکند که آیا فشردن دکمه سوم طولانی بوده یا خیر، بدینصورت که با فعالسازی Systick و فراخوانی پشت سر هم getTick() بررسی میکنیم اگر تا یک زمان مشخصی دکمه B3 فشرده شده است(پین مربوطه 1 است) بولین long_push درست میماند، در صورتیکه پیش از موعد تعیین شده مقدار B3 صفر شود، long_push نادرست شده و از وایل خارج می‌شویم. در نهایت بولین long_push را برمی‌گردانیم. سپس در هندلر با استفاده از این تابع نوع فشردن را تشخیص می‌دهیم.

4. B3_clicked()

```
// Check for B3-Push
bool B3_clicked(){
    return HAL_GPIO_ReadPin(GPIOB, MASK(B3) ) == GPIO_PIN_SET;
}
```

این تابع صرفا بررسی میکند که دکمه B3 فشرده شده‌است یا خیر.

5. PrintTime(msec, sec, min)

```
// Prints the given time ( in timer-format) on LCD
void printTime(uint16_t min, uint16_t sec, uint16_t msec){
    commitCommand(0xC0);

    // Minutes:
    commitChar( intToChar(min/10) );
    commitChar( intToChar(min%10) );

    commitChar(':');

    // Seconds:
    commitChar( intToChar(sec/10) );
    commitChar( intToChar(sec%10) );

    commitChar(':');

    // MiliSeconds:
    commitChar( intToChar(msec/100) );
    commitChar( intToChar((msec%100)/10 ) );
    commitChar( intToChar(msec%10) );
}
```

این تابع سه عدد مربوط به میلی‌ثانیه، ثانیه و نیز دقیقه را دریافت کرده و آن‌ها را به صورت صحیح و در فرمت یک تایمر توسط توابع LCD روی نمایشگر نشان می‌دهد.

6. InterruptHandler

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
    // On System Startup we shouldnt accept any commands
    if ( systemState == STARTUP )
        return;

    // Count if the system is not off
    if(GPIO_Pin == GPIO_PIN_0 ){
        if( systemState != OFF )
            systemState = COUNTING;
    }
    // Stop Timer
    if(GPIO_Pin == GPIO_PIN_1 ){
        if(systemState == OFF)
            return;
        systemState = STOPPED;
    }
    // LongPush -> Turn off the system, else -> Reset Timer !
    if(GPIO_Pin == GPIO_PIN_2 ){
            HAL_NVIC_SetPendingIRQ(EXTI2_IRQn);
            min = sec = msec = 0;
            if ( long_push() ) {
                systemState = OFF;
                clear_line_2();
            }
            else {
                systemState = STOPPED;
                printTime(0, 0, 0);
            }
            while( B3_clicked() );
            HAL_NVIC_ClearPendingIRQ(EXTI2_IRQn);
    }
}
```

در صورتیکه سیستم در Startup باشد هیچ فرمانی را نمی‌پذیریم. چنانچه دکمه 1 فشرده شود و سیستم Off نباشد، به حالت Counting می‌رویم؛ به همین ترتیب اگر دکمه 2 فشرده شود و در حالت Off نباشیم، به حالت Stopped می‌رویم. اگر کلید 3 ممتد فشرده شود (long_push یک برگرداند) به حالت Off رفته و LCD را پاک می‌کنیم. در غیر اینصورت، صرفا وارد حالت Stopped رفته و مقادیر min ، msec و sec را ریست کرده و روی LCD چاپ می‌کنیم. (در هر دو حالت فوق متغیرهای ثانیه و میلی‌ثانیه و دقیقه ریست می‌شوند)

7. TimerInterruptHandler

```
// TIM2 -> For counting MiliSeconds, TIM3 -> For counting halfSeconds (in order to print and clear 'TURN OFF' )
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){

    if (htim->Instance == TIM2){
        // Counter Timer :
        if( systemState == COUNTING ){
                msec+=8;
            if(msec == 1000){
                sec++;
                msec = 0;
            }
            if(sec == 60){
                min++;
                sec = 0;
            }
            printTime( min, sec, msec);
        }
    }
    // Timer for OFF state :
    else if (htim->Instance == TIM3){
        static bool turnOffPrinted = false;
        if(systemState == OFF ){
            if(turnOffPrinted){
                clear_line_2();
            }
            else {
                printInLine("TURN OFF");
            }
            turnOffPrinted = !turnOffPrinted;
        }
    }
}
```

اگر اینتراپت توسط تایمر 2 صادر شود و نیز در حالت Counting باشیم، مقادیر میلی‌ثانیه، ثانیه، و دقیقه را متناسب با مقادیرشان، increment می‌کنیم (در مورد میلی-ثانیه خطا و نیز تاخیر مربوط به نوشتن روی lcd را نیز در نظر گرفتیم، لذا آنرا 8 واحد، 8 واحد افزایش می‌دهیم) در صورتیکه msec به 1000 و sec به 60 برسد، ریستشان کرده و پارامتر فوق آن‌ها را increment می‌کنیم. در نهایت مقادیر را روی lcd چاپ میکنیم.

اگر اینتراپت توسط تایمر 3 صادر شود (که مختص نمایش و پاک کردن عبارت TURN OFF در حالت Off می‌باشد) توسط یک متغیر استاتیک بررسی می‌کنیم در فراخوانی گذشته LCD پاک شده یا اینکه TURN OFF چاپ شده است، سپس عکس عمل گذشته را انجام می‌دهیم (این عمل در دوره‌های تقریبا نیم‌ثانیه‌ای صورت می‌گیرد)

8. Main

```c
int main(void)
{
  /* USER CODE BEGIN 1 */
  /* USER CODE END 1 */
  /* MCU Configuration--------------------------------------------------------*/
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();
  /* USER CODE BEGIN Init */
  /* USER CODE END Init */
  /* Configure the system clock */
  SystemClock_Config();
  /* USER CODE BEGIN SysInit */
  /* USER CODE END SysInit */
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM2_Init();
  MX_TIM3_Init();
  /* USER CODE BEGIN 2 */
    init_lcd();
    printInLine("Welcome");
    delay(30000);
    clear_line_1();
    printTime(0, 0, 0);
    systemState = STOPPED;
    HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
    HAL_TIM_Base_Start_IT(&htim3);
    HAL_TIM_Base_Start_IT(&htim2);
  /* USER CODE END 2 */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

در این تابع پس از Initialization های مربوط به HAL که auto-generate هستند، lcd را initialize کرده، سپس welcome را برای مدت کوتاهی نشان می‌دهیم، سپس آنرا پاک کرده و زمان صفر را در فرمت تایمر چاپ می‌کنیم و سیستم را به حالت Stopped میبریم. در نهایت، اولویت Systick را صفر میکنیم (تا در ISR ها امکان اجرا داشته باشیم لازم به ذکر است اولویت EXTI ها را به 1 تغییر داده‌ایم. لذا اولویت SysTick بیشتر خواهد بود) و تایمرها را فعال می‌کنیم.

پروتئوس