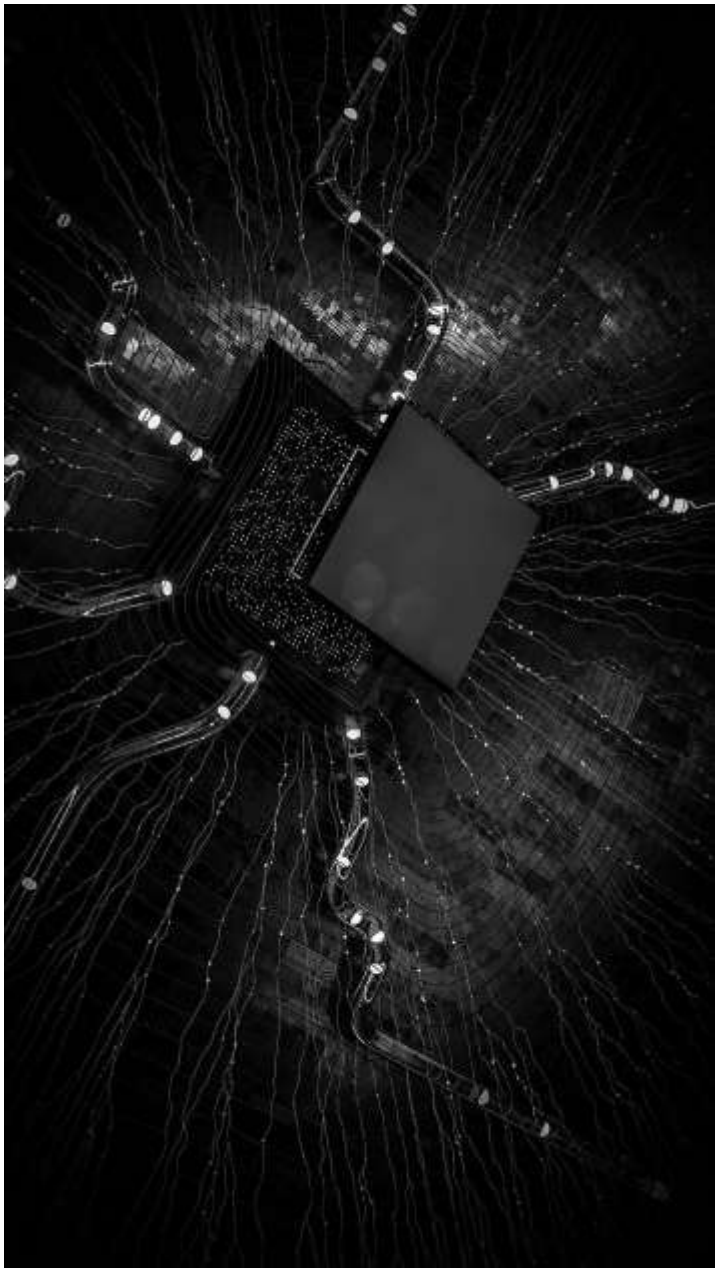


آزمایشگاه ریزپردازنده

تمرین هشت

سینا عربی – سالار جهانشیری – امیرعلی وکیلی

سوالات تحلیلی:

منابع کلاک میکروکنترلر STM32F401 را نام برده و هرکدام را توضیح دهید.

منابع کلاک میکروکنترلر STM32F401 عبارتند از:

1. HSI (High-Speed Internal) که یک سیگنال ساعت داخلی با فرکانس 16 مگاهرتز است. این سیگنال می‌تواند به عنوان منبع کلاک اصلی یا منبع کلاک فرعی استفاده شود.

2. HSE (High-Speed External) که یک سیگنال ساعت خارجی است و معمولاً با فرکانس‌های 8، 12 یا 16 مگاهرتز در دسترس است. این منبع کلاک برای اتصال به ساعت‌های خارجی مانند کریستال یا ساعت‌های داخلی دیگر استفاده می‌شود.

3. PLL (Phase-Locked Loop) که به منظور تولید سیگنال‌های ساعت با فرکانس‌های بالاتر از منابع HSI یا HSE استفاده می‌شود. این منبع کلاک به صورت دقیق و پایدار فرکانس‌های بالاتر را تولید می‌کند.

4. MSI (Medium-Speed Internal) که یک سیگنال ساعت داخلی با فرکانس قابل تنظیم بین 100 کیلوهرتز تا 48 مگاهرتز است. این منبع کلاک برای استفاده در حالت‌های مصرف پایین و اقتصادی مناسب است.

این منابع کلاک به کاربر اجازه می‌دهند تا با توجه به نیازهای خود، منبع کلاک مناسب را برای میکروکنترلر STM32F401 انتخاب و پیکربندی کنند.

Interrupt vector table چیست و عملکرد آن در پردازنده ARM چگونه است؟

جدول بردار سمتی (Interrupt Vector Table) یک جدول است که در معماری پردازنده ARM قرار دارد و برای مدیریت و پردازش ایستگاه‌های وقفه (interrupts) استفاده می‌شود. این جدول شامل آدرس‌های حافظه است که به ترتیب با ایستگاه‌های وقفه متناظر در پردازنده مرتبط شده‌اند.

عملکرد جدول بردار سمتی در پردازنده ARM به صورت زیر است:

1. هنگامی که یک ایستگاه وقفه رخ می‌دهد، پردازنده به صورت خودکار به آدرس متناظر در جدول بردار سمتی پرش می‌کند.

2. در آدرس مذکور، یک روتین خاص (روتین وقفه) قرار دارد که برای پردازش ایستگاه وقفه استفاده می‌شود. این روتین شامل دستوراتی است که باید در صورت وقوع وقفه اجرا شود.

3. پس از اجرای روتین وقفه، پردازنده به آدرس بعدی در جدول بردار سمتی پرش می‌کند تا بتواند وقفه بعدی را پردازش کند.

با استفاده از جدول بردار سمتی، پردازنده قادر است به صورت خودکار و به ترتیب، ایستگاه‌های وقفه را پردازش کند و به همین دلیل، این سیستم مناسب برای مدیریت چندین ایستگاه وقفه در پردازنده ARM است.

دستور کار بخش اصلی:

در بخش‌های اول و دوم با استفاده از توابع مربوط به LCD و با استفاده از حل، پورت‌های مربوطه را آماده‌سازی و طبق شماتیک تعریف شده در کلاس برای اتصال پورت های بورد Nucleo-64 به پایه‌های LCD عمل می‌کنیم.

بخش سوم:

در این بخش و بخش های بعدی بیشتر به جزئیات پیاده سازی بخش دو و روتین های وقفه خواهیم پرداخت.

```
LCD_Init();
show_freq_clock();

Configure_PA0();
Configure_PA1();

size_t length = sizeof(message);

if (length <= LCD_LENGTH)
{
    // Print message on the second line
    LCD_Puts(0, 1, message);
}
else
{
    int first_index = 0;
    char temp[LCD_LENGTH];
    while (1)
    {
        while (first_index + LCD_LENGTH - 2 != length)
        {
            memcpy(temp, &message[first_index], LCD_LENGTH - 1);
            temp[LCD_LENGTH - 1] = '\0';
            LCD_Puts(0, 1, temp);
            HAL_Delay(250);
            ++first_index;
            show_freq_clock();
        }
        first_index = 0;
    }
}
```

این بخش ربط به main برنامه دارد و در ابتدا در این تابع، LCD مقداردهی اولیه میشود و فرکانس اولیه کلاک بر روی LCD نمایان میشود،حال Configuration های مربوط به پین های

A را داریم که در ادامه در بخش چهارم به این میپردازیم، حال طول message که باید بر روی LCD نمایش دهیم را در متغیر length ذخیره میکنیم، اگر طول آن از طول کارکتر های مجازی مه میتوان بر روی LCD نمایش داد کمتر بود message را بر روی LCD نمایش میدهیم در غیر این صورت با استفاده از آرایه temporary در یک حلقه بی انتها تا وقتی که اندیس اولیه آرایه به علاوه طول LCD منها دو برابر با طول رشته نشده، رشته را بر روی LCD نمایش میدهیم و هربار در این حلقه اندیس اولیه را یکی به جلو میبریم و آرایه temporary را پر میکنیم که در هر چرخه با یک delay و نشان دادن فرکانس دوباره کلاک چرخه را تمام میکنیم، که در این صورت باعث میشود در هر چرخه یک اندیس(پیکسل) پیام به سمت چپ حرکت کند.

بخش چهارم:

```
void Configure_PA0(void)
{
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN | RCC_AHB1ENR_GPIOCEN | RCC_AHB1ENR_GPIOBEN;
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;

    GPIOA->MODER &= ~(3 << 0);

    // SYSCFG->EXTICR[0] &= (0x0 << 0); // Clear EXTI0 field (bits 0-3)
    SYSCFG->EXTICR[0] &= SYSCFG_EXTICR1_EXTI0_PA;
    // Connect

    EXTI->FTSR |= (1 << 0); // Falling trigger
    EXTI->IMR |= (1 << 0);  // Clear mask
    NVIC_EnableIRQ(EXTI0_IRQn);
}

void Configure_PA1(void)
{
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN | RCC_AHB1ENR_GPIOCEN | RCC_AHB1ENR_GPIOBEN;
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;

    GPIOA->MODER &= ~(3 << 2);

    // SYSCFG->EXTICR[0] &= (0x0 << 4); // Clear EXTI1 field (bits 4-7)

    SYSCFG->EXTICR[0] &= SYSCFG_EXTICR1_EXTI1_PA;

    EXTI->FTSR |= (1 << 1); // Falling trigger
    EXTI->IMR |= (1 << 1);  // Clear mask

    NVIC_EnableIRQ(EXTI1_IRQn);
```

در ابتدای این بخش همانطور که قبلا گفته شده بود، دو Configuration برای ست کردن ورودی push button ها در پورت A با پین های 0 و 1 را داریم که در بدنه این تابع ها در ابتدا

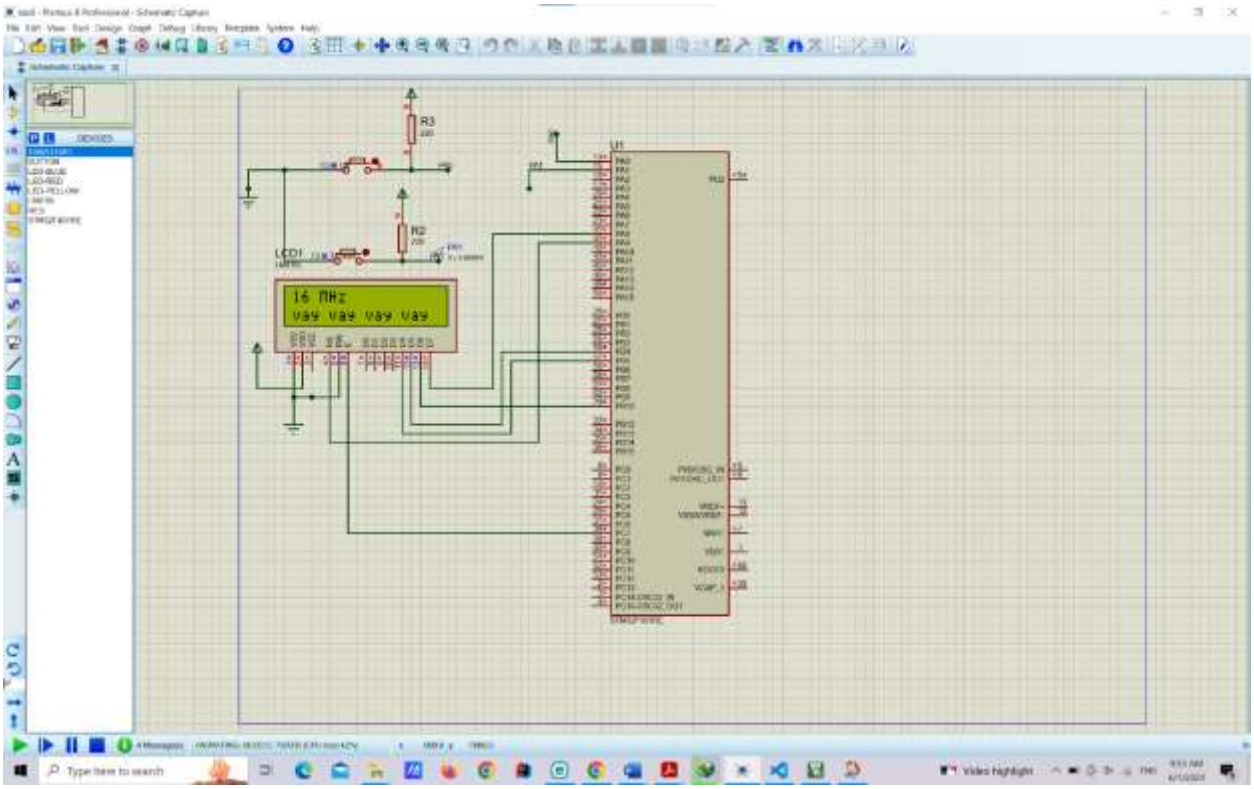
کلاک را ست میکنیم و بعد با استفاده از رجیستر GPIO مقدار چهار بیت اول را صفر میکنیم برای input mode پین های 0 و 1و در آخر هم با استفاده از extrenal interrupts کلاک را به لبه پایین رونده ست میکنیم.

```
void Set_Clock(bool increase)
{

    BaseClock += increase ? 1 : -1;
    RCC->PLLCFGR = (16 << RCC_PLLCFGR_PLLM_Pos) | (BaseClock * 4 << RCC_PLLCFGR_PLLN_Pos) | (1 << RCC_PLLCFGR_PLLP_Pos);

    RCC->CR |= RCC_CR_PLLON;
    while ((RCC->CR & RCC_CR_PLLRDY) == 0)
        ;

    RCC->CFGR |= (RCC_CFGR_SW_PLL | (4 << RCC_CFGR_PPRE1_Pos));

    while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_PLL)
        ;

    SystemCoreClockUpdate();
}
```

در این تابع مقدار کلاک را با توجه به ورودی که از پورت ها با توجه به push button ها میگیریم تغییر میدهیم، که در ابتدا یک ورودی Boolean دارد اگر true بود مقدار کلاک را افزایش و در غیر این صورت کاهش میدهیم که ازین تابع درinterrupt handler های بخش بعد استفاده میشود که بهش خواهیم پرداخت.

```c
void show_freq_clock(void)
{
    size_t length = sizeof(message);

    char systemClockFreqStr[20]; // Buffer to hold the resulting string

    uint32_t systemClockFreq = HAL_RCC_GetHCLKFreq();
    systemClockFreq /= 1000000; // Convert to MHz

    // Format the system clock frequency and concatenate with " MHz"
    snprintf(systemClockFreqStr, sizeof(systemClockFreqStr), "%lu", systemClockFreq);
    strncat(systemClockFreqStr, messageMHz, sizeof(systemClockFreqStr) - strlen(systemClockFreqStr) - 1);

    // Clear the LCD before writing
    LCD_Clear();

    // Print frequency on the first line
    LCD_Puts(0, 0, systemClockFreqStr);

    delay(100);
}
```

در این تابع فرکانس کلاک را بر روی LCD نمایش میدهیم که در ابتدا این مقدار را از سیستم دریافت میکنیم و بعد آن را به MHz تبدیل میکنیم و بعد از clear کردن LCD با استفاده از تابع LCD_Puts آن را نمایش میدهیم و در آخر یک delay هم ایجاد میکنیم.



بخش پنجم:

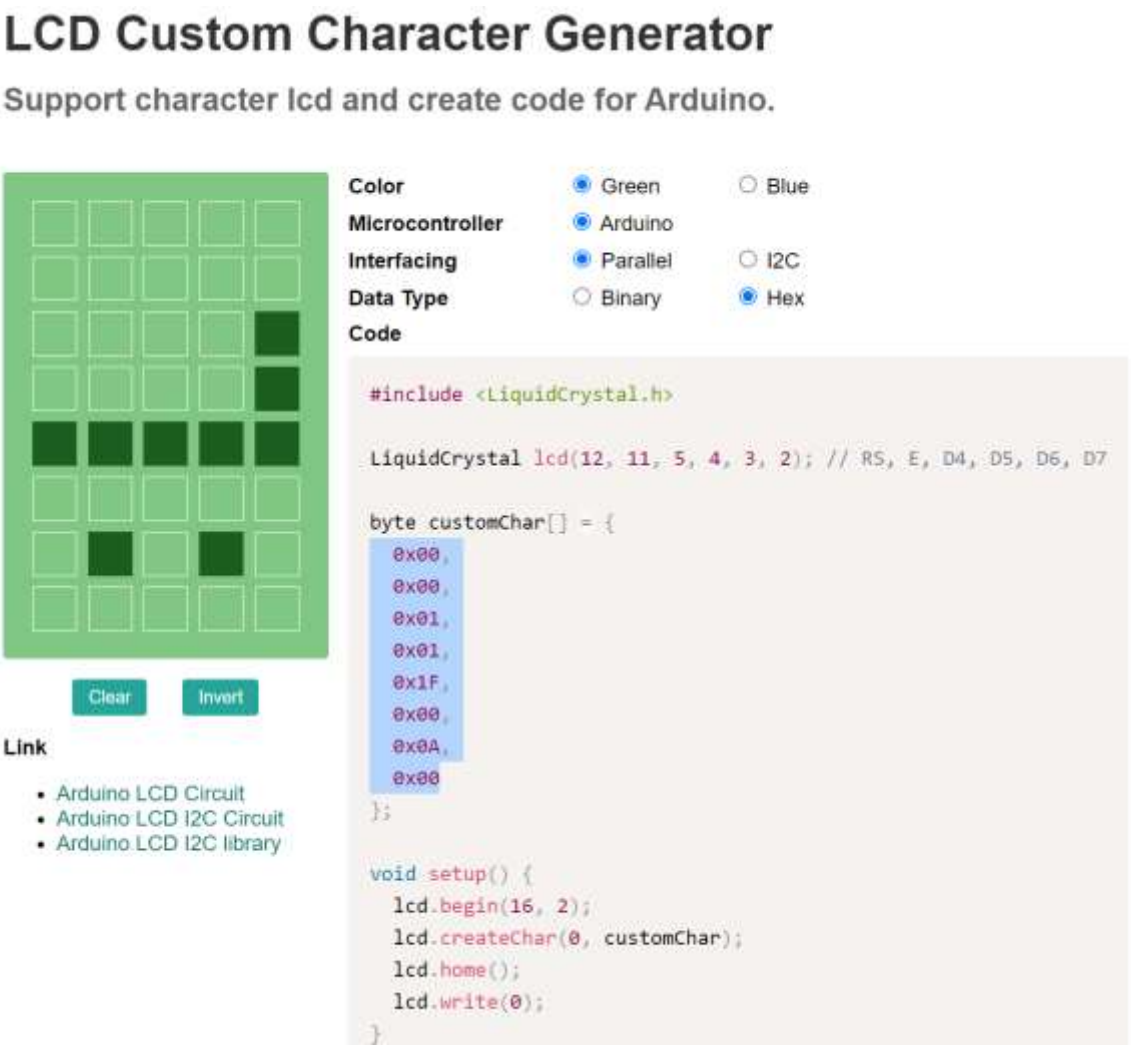
```
void EXTI0_IRQHandler(void)
{

    Set_Clock(true);
    EXTI->PR |= (1 << 0);             // Clear pending interrupt
    NVIC_ClearPendingIRQ(EXTI0_IRQn); // Not necessary if using debouncing circuitry
}

/* Set interrupt handlers */
/* Handle PA1 interrupt */
void EXTI1_IRQHandler(void)
{

    Set_Clock(false);
    EXTI->PR |= (1 << 1);             // Clear pending interrupt
    NVIC_ClearPendingIRQ(EXTI1_IRQn); // Not necessary if using debouncing circuitry
}
```

در این بخش پیاده سازی روتین های وقفه را برای دو پین A0 و A1 در پورت A را داریم که همانطور که در قبل دیدیم در Configuration این دو پین به روتین وقفه مختص خود وصل میشود، حال اگر در پین صفر ورودی گرفتیم به این معنی است که فرکانس کلاک را باید یک MHz افزایش دهیم که تابع set_Clock را که قبلا توضیح داده بودیم با ورودی true فراخوانی میکنیم و بعد از interrupt خارج میشویم، برای پین یک هم تابه set_clock را با ورودی false فراخوانی میکنیم و باز با استفاده از external interrupts از اینتراپت خارج میشویم.

دستور کار بخش امتیازی:

همانطور که می‌دانیم LCD از کاراکترهای فارسی پشتیبانی نمی‌کند، به همین علت نیازمندیم تا نمایش پیکسلی حروف فارسی روی LCD را تعریف کنیم. برای این کار به کمک لینک زیر و با تعریف شکل ظاهری هر کاراکتر، نمایش هگز آن را تعریف می‌کنیم.

https://maxpromer.github.io/LCD-Character-Creator/



در ادامه با استفاده از LCD_CreateChar حروف مورد نیاز با یک کلید عددی معرفی شده و به کمک LCD_PutCustom نمایش داده می‌شود.

```
void print_persian_style(void)
{
    int sc = 0;
    LCD_CreateChar(0, alef_hat);
    LCD_CreateChar(1, ze);
    LCD_CreateChar(2, re);
```

```
    LCD_CreateChar(3, ye_middle_2);

    LCD_PutCustom(1, 1, 1);
    LCD_PutCustom(2, 1, 3);
    LCD_PutCustom(3, 1, 2);
    LCD_PutCustom(4, 1, 1);
    LCD_PutCustom(5, 1, 0);
}
```