МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СИКОРСЬКОГО»

КАФЕДРА КОНСТРУЮВАННЯ ЕЛЕКТРОННО-ОБЧИСЛЮВАЛЬНОЇ АПАРАТУРИ

ЗВІТ

з лабораторної робіти № 5
по курсу «Обчислювальні та МП засоби в РЕА-2»

Виконав:
студент гр. ДК-82
Дмитрук О.О.

Перевірив:
ст. викладач
Бондаренко Н.О.

Київ – 2021

Завдання

1. Реалізувати секундомір, з 3-ма кнопками, які відповідають за скидання секундоміру, старт та стоп його лічби. Використовувати переривання та системний таймер SysTick. Використовувати бібліотеку CMSIS.

Теоретична частина

1. Interrupts (переривання)

Переривання - використання спеціального блоку в MCU, який виявляє певну подію (натискання кнопки, закінчення заданого інтервалу часу, отримання повідомлення з комунікаційного порту, т. і.) та запускає у відповідну підпрограму **ISR** (*Interrupt Service Routine*)

Переваги переривань це:

Ефективність - код виконується лише тоді, коли це необхідно.

Швидкість - апаратний механізм виявлення події і запуску реакції

Різниця між перериванням та подією наступна:

Event – це апаратна подія в MCU, у внутрішньої периферії або у зовнішньому пристрою.

Подія може викликати переривання, або просто підняти прапорець, розбудити процесор, запустити яку-небудь периферію.

Interrupt - *Переривання /Виключення*- це **сигнал**, по якому процесор "дізнається" про вчинення асинхронної (*asynchronous*) або синхронної (*synchronous*) події (*Event*). При цьому виконання поточної послідовності команд (програми) припиняється (переривається), а замість неї починає виконуватися інша послідовність, яка відповідає даному перериванню (події).

Переривання поділяють на синхронні та асинхронні.

Переривання можна маскувати (забороняти проходити сигналу). Також є переривання які не маскуються (NMI) .

Всього в STM32 255 переривань та виключень. Кожне переривання має свій порядковий номер в таблиці векторів переривань. Перші 16 переривань – системні, а починаючи з номеру 16 – зовнішні. По суті таблиця містить в собі мітки на певні підпрограми, які реалізуються розробником, по яким виконується перехід при виконанні переривання.

Щоб написати підпрограму обробки переривання, потрібно правильно визначити назву функції. Назви можна найти в стартап файлі, який ми підключаємо при підключенні бібліотеки CMSIS.

```
83    DCD     FLASH_IRQHandler          ; FLASH
84    DCD     RCC_IRQHandler            ; RCC
85    DCD     EXTI0_IRQHandler          ; EXTI Line0
86    DCD     EXTI1_IRQHandler          ; EXTI Line1
87    DCD     EXTI2_IRQHandler          ; EXTI Line2
88    DCD     EXTI3_IRQHandler          ; EXTI Line3
89    DCD     EXTI4_IRQHandler          ; EXTI Line4
90    DCD     DMA1_Stream0_IRQHandler   ; DMA1 Stream 0
91    DCD     DMA1_Stream1_IRQHandler   ; DMA1 Stream 1
```

Рис.1. приклад визначених імен функцій обробників переривань.

Обробники переривань поділяються на згруповані та незгруповані.

Кожен незгрупований обробник переривань відповідає за власне переривання. Що на рахунок згрупованих, наприклад EXTI15_10_IRQHandler, такі обробники будуть виконувати обробку, при виконані хоча б одного переривання по лінії (EXTI10-15) , тобто якщо хоча б 1-не з 6-ти переривань виконається, то буде виконане EXTI15_10_IRQHandler.

Для того щоб ініціалізувати переривання, потрібно виконати наступне.

1. Включити тактування переривань.
2. Виконати налаштування мультиплексорів, в регістрі EXTICR, які відповідають за вибірку GPIO пінів, сигнал з яких буде зчитуватись.
3. Розмаскувати переривання, регістр IMR

4. Налаштувати, на який фронт буде генеруватись переривання, якщо на зростання то це регістр RTSR, якщо спадання, то регістр FTSR.
5. Налаштувати пріоритет переривань, в системі NVIC.
6. Очистити біти черги (Pending) в NVIC.
7. Активувати переривання в NVIC

Загалом це вся процедура по налаштуванню переривань. Також варто памятати, що при налаштуванні пріоритетності, якщо під час виконання низькопріоритетного переривання, виконається високопріоритетне переривання, то система буде чекати поки завершиться низькопріор. а тоді вже дасть високопріоритет. перериванню обробитись.

Всю інформацію, про налаштування регістрів можна найти в reference manual-і.

2. Таймери в STM32

Таймер - це лічильник, що працює не залежно від процесору.

Функції таймерів :

Вимірювання тривалості (або) періоду імпульсів

Підтримка Енкодерів

Формування періодичних переривань, DMA-запитів

Генерування періодичних послідовностей імпульсів

Генерування ШІМ сигналів

Формування поодиноких імпульсів різної тривалості

Всього є 15 таймерів в stm32f401RE. Вони поділяються на :

Базові (16-бітні) - відраховують інтервали часу і генерують переривання при досягненні заданого значення - підтримують DAC, формують запити DMA.

Таймери заг. призначення (16/32 бітні) - мають усі можливості базових таймерів здатні формувати ШИМ (PWM) сигнали, рахувати імпульси на певних входах, підтримувати енкодери датчики Холла, декілька таймерів можна сінхронизувати між собою.

Таймери з розвиненим управлінням - виконують всі функції менш розвинутих таймерів підтримують управління трифазним електроприводом

Системний таймер (Systick) – 24-бітний таймер, який є складовою частиною ядра Cortex-M4

Таймери можуть рахувати в 3-х режимах:

Up-counting – рахунок вверх. При досягненні певного значення, яке міститься в регістрі ARR, виконується переривання виконання події, і по наступному актив. фронту такт. сигналу виконується обнулення лічильника, рахунок починається з початку.

Down-counting – рахунок вниз. При обнулені лічильника виконується переривання виконання події, і по наступному фронту такт. сигналу виконується перезагрузка лічильника значенням регістру ARR.

Center-aligned-counting – Виконується рахунок вверх до певного значення, після рахунок вниз до нуля, і після досягнення нуля лічильником, виконується переривання.

Лічильники тактуються наступним чином:

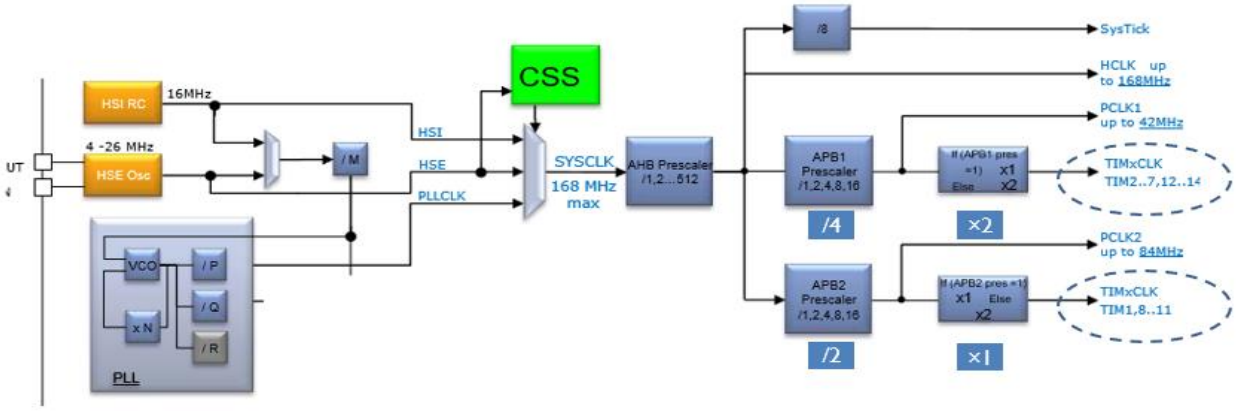

Рис.2. тактування таймерів

Розглянемо налаштування системного таймеру, так як ми будемо його використовувати.

Сист. таймер має наступні регістри для його налаштування:

1. **SysTick->CTRL = 0**; – SysTick Control and Status Register.
   В цьому полі відбувається скидання налаштувань системного таймеру SysTick .
2. **SysTick->LOAD** – SysTick Reload Value Register.
   В цьому полі відбувається налаштування регістру перезавантаження (Reload Value Register). Відбувається завантаження необхідного числа затримки.
3. **SysTick->VAL = 0**; – SysTick Current Value Register.
   В цьому полі відбувається завантаження поточного значення, з якого буде починатися рахунок таймера. Якщо SysTick->VAL = 0, то значення завантажується з регістра SysTick->LOAD, в іншому разі, поточне значення регістра залишиться незмінним.
4. **SysTick->CTRL= SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_TICKINT_Msk | SysTick_CTRL_ENABLE_Msk**;

Тут відбувається налаштування 3-х бітів: **Біт SysTick_CTRL_ENABLE_Msk** Він є бітом дозволу на рахунок для таймера. Якщо SysTick_CTRL_ENABLE_Msk = 1, то таймер може рахувати. Тоді ж він автоматично завантажує свої регістри рахунку значеннями з регістру LOAD - регістр попереднього завантаження, з якого таймер бере значення для перезавантаження при обнуленні. В цей регістр можна завантажити необхідне число затримки (до 24 біт)).
Якщо SysTick_CTRL_ENABLE_Msk = 0, тоді - не може рахувати.

2. **Біт SysTick_CTRL_TICKINT_Msk.** Цей біт є бітом дозволу переривань. Переривання буде генеруватися тоді, коли лічильник таймера порахує до «0». Тоді в обробнику переривань починається їх обробка. Для обробки переривань в коді програми необхідно визначити обробник переривань з ім’ям SysTick_Handler().
Якщо SysTick_CTRL_TICKINT_Msk = 1, то переривання таймера буде відбуватися при його обнуленні.

3. **Біт SysTick_CTRL_CLKSOURCE_Msk**. Цей біт визначає джерело тактів.
Якщо SysTick_CTRL_CLKSOURCE_Msk = 0, то тактування відбувається за допомогою зовнішнього еталонного генератора.
Якщо SysTick_CTRL_CLKSOURCE_Msk = 1, то тактування відбувається з частоти процесора.

Розглянувши необхідну теорію можемо перейти до реалізації секундоміру.

Виконання роботи

Розпочнемо з ініціалізації портів GPIO, функція init_gpio() (рис.3.)

```
3  void init_gpio () {
4      // ports B and C clock enable
5      RCC->AHB1ENR |= RCC_AHB1ENR_GPIOBEN | RCC_AHB1ENR_GPIOCEN;
6
7      // configure pin C(0-3) to input (mode_bits = 00)
8      GPIOC->MODER &= ~(GPIO_MODER_MODE0_0|GPIO_MODER_MODE0_1);
9
10     GPIOC->MODER &= ~(GPIO_MODER_MODE1_0|GPIO_MODER_MODE1_1);
11
12     GPIOC->MODER &= ~(GPIO_MODER_MODE2_0|GPIO_MODER_MODE2_1);
13
14     GPIOC->MODER &= ~(GPIO_MODER_MODE3_0|GPIO_MODER_MODE3_1);
15
16     // configure pin B(0-8) to output (mode_bits = 01)
17     GPIOB->MODER |= GPIO_MODER_MODE0_0;
18     GPIOB->MODER &= ~GPIO_MODER_MODE0_1;
19
20     GPIOB->MODER |= GPIO_MODER_MODE1_0;
21     GPIOB->MODER &= ~GPIO_MODER_MODE1_1;
22
23     GPIOB->MODER |= GPIO_MODER_MODE2_0;
24     GPIOB->MODER &= ~GPIO_MODER_MODE2_1;
25
26     GPIOB->MODER |= GPIO_MODER_MODE3_0;
27     GPIOB->MODER &= ~GPIO_MODER_MODE3_1;
28
29     GPIOB->MODER |= GPIO_MODER_MODE4_0;
30     GPIOB->MODER &= ~GPIO_MODER_MODE4_1;
31
32     GPIOB->MODER |= GPIO_MODER_MODE5_0;
33     GPIOB->MODER &= ~GPIO_MODER_MODE5_1;
34
35     GPIOB->MODER |= GPIO_MODER_MODE6_0;
36     GPIOB->MODER &= ~GPIO_MODER_MODE6_1;
37
38     GPIOB->MODER |= GPIO_MODER_MODE7_0;
39     GPIOB->MODER &= ~GPIO_MODER_MODE7_1;
40
41     GPIOB->MODER |= GPIO_MODER_MODE8_0;
42     GPIOB->MODER &= ~GPIO_MODER_MODE8_1;
43 }
44
```

Рис.3. Функція налаштування GPIO

До пінів 0-3 порту С будуть підключені кнопки. До пінів 0-8 порту В підключений 2-розряд. семисигментний індикатор та 2 транзистори, які будуть керувати динамічною індикацією індикаторів.

Вмикаємо тактування портів В та С (5 сточка коду)

Налаштовуємо на вхід піни 0-3 порта С (7-14 сточки)

Налаштовуємо на вихід піни 0-8 порта В (16-42 строчки)

Всі вище розглянуті регістри не потрібно пояснювати, так як ми працювали з ними в попередніх лаб. роботах.

Перейдемо до функції налаштування системного таймеру init_systick() (рис.4.)

```
90  void init_systick(){
91      // Reset Systick
92      SysTick->CTRL = 0;
93
94      // Value which load to counter after overflow
95      SysTick->LOAD = (FREQ_DIV_VAL - 1);
96
97      // Value from which count to LOAD
98      SysTick->VAL = 0;
99
100     SysTick->CTRL |= SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_TICKINT_Msk | SysTick_CTRL_ENABLE_Msk;
101 }
```

Рис.4. функція налаштування таймеру

Виконуємо обнулення регістру конфігурації (строчка 92)

Загружаємо в регістр LOAD константу , яка містить значення 16-мільйоннів - 1 (тактування сист. тайм. виконується внутрішнім генератором, з частотою 16 мгц, тому потрібно поділити цю частоту, щоб отримати сигнал 1 герц) (строчка 95)

Завантажуємо в регістр VAL (лічильник) нуль, для того щоб по наступному актив. фронту почалась лічба. (строчка 98)

Встановлюємо певні біти в регістрі CTRL, щоб налаштувати таймер (строчка 100). За що дані біти відповідають, згадано вище.

Перейдемо до функції налаштування переривань irq0_3_enable() (рис.5.)

```
45  void irq0_3_enable () {
46      // SYSCFGREG clock enable
47      RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
48
49      // make PC(0-3) as IRQ interrupt source
50      SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI0_PC;
51      SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI1_PC;
52      SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI2_PC;
53      SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI3_PC;
54
55      // enable (0-3) interrupt (it is about mask)
56      EXTI->IMR |= EXTI_IMR_IM0;
57      EXTI->IMR |= EXTI_IMR_IM1;
58      EXTI->IMR |= EXTI_IMR_IM2;
59      EXTI->IMR |= EXTI_IMR_IM3;
60
61      // use positive front (Rising trigger)
62      EXTI->RTSR |= EXTI_RTSR_TR0;
63      EXTI->RTSR |= EXTI_RTSR_TR1;
64      EXTI->RTSR |= EXTI_RTSR_TR2;
65      EXTI->RTSR |= EXTI_RTSR_TR3;
66
67      /* NVIC(Nested Vectored Interrupt Controller) */
68      // set priorities
69      NVIC_SetPriority(EXTI0_IRQn,0); // set highest priority
70      NVIC_SetPriority(EXTI1_IRQn,1);
71      NVIC_SetPriority(EXTI2_IRQn,2);
72      NVIC_SetPriority(EXTI3_IRQn,3);
73
74      // clear pendings
75      NVIC_ClearPendingIRQ(EXTI0_IRQn);
76      NVIC_ClearPendingIRQ(EXTI1_IRQn);
77      NVIC_ClearPendingIRQ(EXTI2_IRQn);
78      NVIC_ClearPendingIRQ(EXTI3_IRQn);
79
80      // enable interrupts
81      NVIC_EnableIRQ(EXTI0_IRQn);
82      NVIC_EnableIRQ(EXTI1_IRQn);
83      NVIC_EnableIRQ(EXTI2_IRQn);
84      NVIC_EnableIRQ(EXTI3_IRQn);
85
86      // enable all interrupts (CMSIS function)
87      __enable_irq();
88  }
```

Рис.5. функція налаштування переривань

Порядок налаштування згадано вище в теоретичних відомостях.

Включимо тактування переривань (строчка 47).

Виконаємо налаштування мультиплексорів, в регістрі EXTICR, які відповідають за вибірку GPIO пінів, сигнал з яких буде зчитуватись (строчка 50-53)

Розмаскуємо переривання, (строчка 56-59)

Налаштуємо, на детект переднього фронту сигналу з кнопок, по яким буде генеруватись переривання (строчка 62-65)

Налаштуємо пріоритет переривань, в системі NVIC (строчка 69-72)

Очистимо біти черги (Pending) в NVIC (строчки 81-84).

Активуємо переривання в NVIC (строчки 87)

Перейдемо до опису обробки переривань. На рис.6. зображено обробка кнопки, яка відповідає за скидання секундоміру.

```
1  #include "header.h"
2  extern int sec0, sec1;
3  // EXTI0_IRQ is responsible for the reset time
4  void EXTI0_IRQHandler () {
5      // disable this interrupt
6      EXTI->IMR &= ~ EXTI_IMR_IM0;
7
8      // change time
9      sec0 = sec1 = 0;
10     SysTick->CTRL &= ~SysTick_CTRL_ENABLE_Msk;
11     SysTick->VAL = (FREQ_DIV_VAL - 1);
12
13     // clear pending bit
14     EXTI->PR|= EXTI_PR_PR0;
15
16     // enable this interrupt
17     EXTI->IMR |= EXTI_IMR_IM0;
18 }
```

Рис.6. Обробка переривання EXTI0

Виконуємо маскування переривання, щоб не відбулось зациклювання і інших проблем (строчка 6)

Скидаємо змінні sec0, sec1 які відповідають за одиниці і десятки секунд відповідно. (строчка 9)

Скидаємо біт дозволу рахунку таймеру (строчка 10)

Очищаємо біт черги, та розмасковуємо переривання (строчки 14 та 17 відповідно)

Лістинг коду строчок 6, 14 та 17 виконуємо постійно в кожних функціях обробки переривань.

На рис.7. зображ. обробка кнопок які відповідають за стоп і старт лічби відповідно.

```
20  // EXTI1_IRQ is responsible for stop timer
21  void EXTI1_IRQHandler () {
22      // disable this interrupt
23      EXTI->IMR &= ~ EXTI_IMR_IM1;
24
25      // stop systick
26      SysTick->CTRL &= ~SysTick_CTRL_ENABLE_Msk;
27
28      // clear pending bit
29      EXTI->PR|= EXTI_PR_PR1;
30
31      // enable this interrupt
32      EXTI->IMR |= EXTI_IMR_IM1;
33  }
34
35  // EXTI2_IRQ is responsible for start timer
36  void EXTI2_IRQHandler () {
37      // disable this interrupt
38      EXTI->IMR &= ~ EXTI_IMR_IM2;
39
40      // start sstick
41      SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;
42
43      // clear pending bit
44      EXTI->PR|= EXTI_PR_PR2;
45
46      // enable this interrupt
47      EXTI->IMR |= EXTI_IMR_IM2;
48  }
49
```

Рис.7. Функції обробки стоп та старту лічби лічильнику.

На строчці 26 виконується стоп лічильнику (скидання біта дозволу рахунку)

На строчці 41 виконується старт лічильнику (встановлення біта дозволу рахунку)

Розглянемо Функції які виконуються в нескінченному циклі, а саме обробка чисел, які відповідають за певні розряди fixNumber (рис.8), та вивід на семисигментний індикатор секунд (рис.9)

```
154 void fixNumber(int *sec0, int *sec1){
155     if(*sec0 > 9) {
156         *sec0 = 0;
157         *sec1 += 1;
158     }
159
160     if(*sec0 < 0){
161         *sec0 = 9;
162         *sec1 -= 1;
163     }
164 }
165
```

Рис.8. Обробка розрядів секунд

Тут все просто, якщо одиниці секунд стають більші чим число 9, виконується інкремент десятків, і обнулення одиниць.

```
103 void switchNum(int sec0, int sec1){
104     int num = 0;
105     int i = 0;
106 
107     for(i = 0; i < 2; i ++){
108         GPIOB->ODR ^= GPIOB->ODR;
109         switch (i){
110             case 0: GPIOB->ODR |= (1 << 7);
111             num = sec0;
112             break;
113 
114             case 1: GPIOB->ODR |= (1 << 8);
115             num = sec1;
116             break;
117         }
118 
119         switch(num) {
120           case 0:
121               GPIOB->ODR |= ZERO;
122               break;
123           case 1:
124               GPIOB->ODR |= ONE;
125               break;
126           case 2:
127               GPIOB->ODR |= TWO;
128               break;
129           case 3:
130               GPIOB->ODR |= THREE;
131               break;
132           case 4:
133               GPIOB->ODR |= FOUR;
134               break;
135           case 5:
136               GPIOB->ODR |= FIVE;
137               break;
138           case 6:
139               GPIOB->ODR |= SIX;
140               break;
141           case 7:
142               GPIOB->ODR |= SEVEN;
143               break;
144           case 8:
145               GPIOB->ODR |= EIGHT;
146               break;
147           case 9:
148               GPIOB->ODR |= NINE;
149               break;
150           }
151     }
152 }
```

рис.9. Функція виводу на семисигментні індикатори секунд

Виконується передача одиниць і десятків секунд. Потім в циклі виконується вивід по черзі розрядів секунд. (динамічна індикація) Спочатку виводяться одиниці секунд, потім десятки. Це відбувається так швидко, що людське око не взмозі побачити затухання одного з розрядів семисиг. індикатору. Маски, One, Two і т.п. взяті з попередньої лаб. роботи. Вони містять в собі значення,

які відповідають за загоряння певних сегментів. 7 і 8 біти в регістрі ODR відповідають за транзистори, які керують катодами світлодіодів.

На рис.10 зображено обробка переривання виконання події системного лічильника.

```
65 void SysTick_Handler() {
66     sec0++;
67 }
```

рис.10. Обробка переривання системного лічильника

Тут все просто, з частотою 1 герц буде інкрементуватись змінна, яка відповідає за одиниці секунд.

Лістинг головної функції main зображ. на рис.11.

```
1  #include "header.h"
2      int sec0, sec1;
3  int main () {
4       sec0 = sec1 = 0;
5      // init functions
6      init_gpio ();
7      init_systick ();
8      irq0_3_enable ();
9
10     // endless loop
11     while (1) {
12         fixNumber(&sec0, &sec1);
13         switchNum(sec0, sec1);
14     }
15 }
16
```

Рис.11. Головна функція main