МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМ. ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
КАФЕДРА АВТОМАТИЗОВАНОГО УПРАВЛІННЯ ТЕХНІЧНИМИ СИСТЕМАМИ

Лабораторна робота № 3

Варіант 1

По дисципліні «Програмування мікроконтролерних систем»

Тема: «Система переривань мікроконтролера. Програмування мікроконтролера для формування сигналів потрібної форми з використанням переривань таймерів»

Виконали:

студенти групи ІТ-51

Бессмертний Р.С.

Цитовцева А.С.

_____

(підпис, дата)

Перевірив:

ст. викладач кафедри АУТС
Катін П. Ю.

_____

(підпис, дата)

Київ-2018

Мета: Розробка і дослідження архітектури програмних рішень з використанням системи переривань на прикладі таймерів мікроконтролера.

Хід роботи:

Варіант 1

Побудувати структурну схему, що містить кнопку на GPIOB і два світлодіода на GPIOA. На першому світлодіоді сформувати сигнал, що має тривалість 0.5 ms і період 1 ms. На другому світлодіоді сформувати сигнал ШИМ, що дає можливість керувати потужністю від 10 % до 100% при натисканні кнопки з кроком у 5%.

Написати і налагодити програми. Протестувати програми на реальному обладнанні, зробити вимірювання з використанням осцилографу. При необхідності зробити корекцію у програмних і апаратних налаштуваннях.

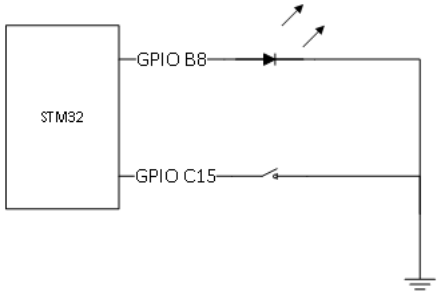

Рис.1 Структурна схема

1. Ініціалізація обробників кнопок та системного таймера разом з налаштуванням його тактування в методі Main.

```
btn_init_in_c (GPIO_Pin_15);
btn_init_in_c (GPIO_Pin_13);
SetSysClockToHSE ();
TIMER4_PWM_init (PERIOD); // pwm period
TIMER2_init (2); // button check period
```

Функції:

```
void SetSysClockToHSE (void)
{
ErrorStatus HSEStartUpStatus;
/ * SYSCLK, HCLK, PCLK2 and PCLK1 configuration* /
/ * RCC system reset (for debug purpose) * /
RCC_DeInit ();

/ * Enable HSE * /
RCC_HSEConfig (RCC_HSE_ON);

/ * Wait till HSE is ready * /
HSEStartUpStatus = RCC_WaitForHSEStartUp ();

if (HSEStartUpStatus == SUCCESS)
{
/ * HCLK = SYSCLK * /
RCC_HCLKConfig (RCC_SYSCLK_Div1);
```

```
/ * PCLK2 = HCLK * /
RCC_PCLK2Config (RCC_HCLK_Div1);

/ * PCLK1 = HCLK * /
RCC_PCLK1Config (RCC_HCLK_Div1);

/ * Select HSE as system clock source * /
RCC_SYSCLKConfig (RCC_SYSCLKSource_HSE);

/ * Wait till PLL is used as a system clock source * /
while (RCC_GetSYSCLKSource ()! = 0x04)
{}
}
else
{/ * If the HSE file is start-up, the application will have a wrong clock configuration.
User can add here some code to deal with this error * /

/ * Go to infinite loop * /
while (1)
{}
}
}

void TIMER4_PWM_init (int period)
{
GPIO_InitTypeDef port;
TIM_TimeBaseInitTypeDef timer;
TIM_OCInitTypeDef timerPWM;

RCC_APB2PeriphClockCmd (RCC_APB2Periph_GPIOB, ENABLE);
RCC_APB1PeriphClockCmd (RCC_APB1Periph_TIM4, ENABLE);

GPIO_StructInit (& port);
port.GPIO_Mode = GPIO_Mode_AF_PP;
port.GPIO_Pin = GPIO_Pin_6;
port.GPIO_Speed = GPIO_Speed_2MHz;
GPIO_Init (GPIOB, & port);

TIM_TimeBaseStructInit (& timer);
timer.TIM_Prescaler = 4;
timer.TIM_Period = period;
timer.TIM_ClockDivision = 0;
timer.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit (TIM4, & timer);

TIM_OCStructInit (& timerPWM);
timerPWM.TIM_OCMode = TIM_OCMode_PWM1;
timerPWM.TIM_OutputState = TIM_OutputState_Enable;
timerPWM.TIM_Pulse = 10;
timerPWM.TIM_OCPolarity = TIM_OCPolarity_High;
TIM_OC1Init (TIM4, & timerPWM);

TIM_Cmd (TIM4, ENABLE);
}

void TIMER2_init (int period)
{
//TIMER2
TIM_TimeBaseInitTypeDef TIMER_InitStructure;
NVIC_InitTypeDef NVIC_InitStructure;

RCC_APB1PeriphClockCmd (RCC_APB1Periph_TIM2, ENABLE); // Enable Timing TIM4 timer
```

```
TIM_TimeBaseStructInit (& TIMER_InitStructure);
TIMER_InitStructure.TIM_CounterMode = TIM_CounterMode_Up; // Account mode
TIMER_InitStructure.TIM_Prescaler = 8000; // Timer frequency separator
// You must also consider the configured dividers RCC_HCLKConfig (RCC_SYSCLK_Div1); RCC_PCLK1Config (RCC_HCLK_Div1);
// In our case, both = RCC_SYSCLK_Div1, that is, the timer divider comes in the frequency of external quartz (8MHz)
TIMER_InitStructure.TIM_Period = period; // The period through which the overflow interrupt is executed // F = 8000000/8000/500 = 2 times / sec.
TIM_TimeBaseInit (TIM2, & TIMER_InitStructure);
TIM_ITConfig (TIM2, TIM_IT_Update, ENABLE); // Enable timer overrun interrupt
TIM_Cmd (TIM2, ENABLE); // Turn on the timer

/ * NVIC Configuration * /
/ * Enable the TIM4_IRQn Interrupt * /
NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 3;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init (& NVIC_InitStructure);
}
```

2. Безкінечний цикл, в якому відповідно до натиснутої кнопки збільшується або зменшується яркість світлодіода.

```
while(1)
	{
		switch(finite_state)
		{
			case C15_PRESS:
				if (pulse_width < FULL_POWER)
				{
					pulse_width += POWER_STEP;
				}
	TIM4->CCR1 = PERIOD * FULL_POWER * pulse_width;
				finite_state = DEFAULT;
				break;
			case C13_PRESS:
				if (pulse_width > MIN_POWER)
				{
					pulse_width -= POWER_STEP;
				}
	TIM4->CCR1 = PERIOD * FULL_POWER * pulse_width;
				finite_state = DEFAULT;
				break;
			case SIN_UPDATE:

				pulse_width = sin_array[sin_counter];

				TIM4->CCR1 = PERIOD * pulse_width / 255;

				if (sin_counter < MAX_SIN-1) //occurs every TIM4 tic
				{
					sin_counter++;
				}
				else sin_counter=0;

				finite_state = DEFAULT;
				break;
```

```
        }
    }
```

3. Робота з генератором описана у наступних чотирьох методах.

```
PWM_Generator update_PWM(PWM_Generator gen)
{
    //power_counter goes from MIN_POWER to FULL_POWER with POWER_STEP
    if (gen.power_counter < gen.FULL_POWER)
    {
        gen.power_counter += gen.POWER_STEP;
    } else
    {
        gen.power_counter = 0;
    }
    return gen;
}
int PWM_signal(PWM_Generator gen)
{
    //until power_counter hits current_power, the lsignal is on. Then it's off
    if (gen.power_counter < gen.current_power){
        return 1;
    } else
    {
        return 0;
    }
}
PWM_Generator increase_power(PWM_Generator gen)
{
    //increase current power by CONTROL_STEP
    if (gen.current_power < gen.FULL_POWER)
    {
        gen.current_power += gen.CONTROL_STEP;
    }
    return gen;
}
PWM_Generator decrease_power(PWM_Generator gen)
{
    //decrease current power by CONTROL_STEP

    if (gen.current_power > gen.MIN_POWER)
    {
        gen.current_power -= gen.CONTROL_STEP;
    }
    return gen;
}
```

Висновки: В даній лабораторній роботі ми працювали з налаштуванням системного таймеру, його тактування та за допомогою їх переривань налаштовували період сигналу для регуляції потужності сигналу.

https://github.com/roman-bessmertnyi/Seventh-semester/tree/master/Subjects/%D0%9F%D0%9C%D0%A1/lab3