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.

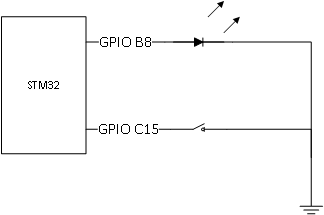


Рис.1 Структурна схема

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);

}

1. 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;  
    }  
    }
2. 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