Lab6 STM32 Clock and Timer
實驗六 STM32 Clock and Timer

Group7 0616231 彭世丞

6-1. Modify system initial clock

1. 實驗目的

Understand the various clock source usage and modification of STM32
瞭解 STM32 的各種 clock source 使用與修改

2. 實驗過程

a. Please use the GPIO_init and Delay1sWith4MHz implemented by the previous lab to call the previous assembly function or re-implement with C to initialize GPIO and delay.

b. Modify the clock source of SYSCLK and the associated prescaler so that the CPU frequency (HCLK) is 1MHz.

c. Observe the frequency at which the LEDs flash before and after modification.

d. When the user presses the user button, the CPU system clock (HCLK) is changed in the following order, 1MHz -> 6MHz -> 10MHz -> 16MHz -> 40MHz -> 1MHz ->...

e. 請利用先前 lab 所實作的 GPIO_init 與 Delay1sWith4MHz，可呼叫之前的 assembly function 或是用 C 重新實作，初始化 GPIO 與 delay。

f. 修改 SYSCLK 的 clock source 以及相關的 prescaler 使得 CPU frequency(HCLK) 爲 1MHz。

g. 觀察修改前後 LED 燈閃爍的頻率。

h. 當使用者按下 user button 便依以下順序改變 CPU system clock(HCLK)，1MHz -> 6MHz -> 10MHz ->16MHz -> 40MHz ->1MHz ->...

```
void GPIO_init();
void Delay1sWith4MHz();
void SystemClock_Config(){
    //TODO: Change the SYSCLK source and set the corresponding
Prescaler value.
}

Int main(){
 SystemClock_Config();
 GPIO_init();
 while(1){
     // make LED light
     Delay1sWith4MHz();
     // make LED dark
     Delay1sWith4MHz();
   }
}
```

Note: Some CPU frequency settings must be made by the multiplier and divider in PLLCLK. In this case, change the SYSCLK source to PLLCLK and set the RCC_PLLCFGR register setting according to the following procedure.

Note: 有些 CPU 頻率設定須由 PLLCLK 內的倍頻器與除頻器達成，此時須將 SYSCLK source 改成 PLLCLK 並依以下流程設定 RCC_PLLCFGR register 設定。

The PLL clock frequency is calculated as f(VCO clock) = f(PLL clock input) × (PLLN / PLLM) The final output to the system clock frequency is f(PLL_R) = f(VCO clock) / PLLR

其中 PLL clock 頻率計算爲 f(VCO clock) = f(PLL clock input) × (PLLN / PLLM) 終可輸出給 system clock 頻率爲 f(PLL_R) = f(VCO clock) / PLLR

3. 寫code過程

這次的作業用到非常多以前已經寫好東西，包括delay的程式、各種initial等等。所以，需要寫的東西其實只有SystemClock_Config的 function，和main function。

首先，我用想要造成的MHz result，去算出個別需要的PLLN, PLLM 和 PLLR，並把他們儲存成陣列，這樣到時候需要修改RCC->PLLCFGR的值就直接引入陣列即可。

SystemClock_Config的function，我就照著以下的做法實作，就完成了。

To modify the PLL configuration, proceed as follows:

1. Disable the PLL by setting PLLON to 0 in *Clock control register (RCC_CR)*.
2. Wait until PLLRDY is cleared. The PLL is now fully stopped.
3. Change the desired parameter.
4. Enable the PLL again by setting PLLON to 1.
5. Enable the desired PLL outputs by configuring PLLPEN, PLLQEN, PLLREN in *PLL configuration register (RCC_PLLCFGR)*.

Main function就更簡單了，先讓ODR打開關上，中間插入delay，然後再每個中間插入檢查buttom press的function，有的話就跑SystemClock_Config的function，藉由更改不同的RCC->PLLCFGR來讓閃爍頻率改變。

6-2. Timer 計時器

1. 實驗目的

Understand the principle of using STM32 timer
瞭解 STM32 的 timer 使用原理

2. 實驗過程

Complete Timer_init() and Timer_start() in main.c below and use STM32 timer to do a timer that counts up from 0 (Upcounting) for TIME_SEC seconds. The second digit below the decimal point is displayed, and the 7-SEG LED stays at the TIME_SEC number at the end. (It is recommended to use TIM2~TIM5 timer with higher counter resolution). Please use polling to get the timer CNT register value and convert it into time display to 7-SEG LED.

0.01 ≤ TIME_SEC ≤ 10000.00 (Please display 0.00 directly beyond the range)
Note: The 7-SEG LED driver should be rendered using the GPIO_init(), max7219_init() and Display () functions previously implemented by Lab (which must be changed to render 2 decimal places).

完成以下的 main.c 中的 Timer_init() 與 Timer_start(); 並使用 STM32 timer 實做一個計時器會從 0 上數(Upcounting) TIME_SEC 秒的時間。顯示到小數點以下第二位，結束時 7-SEG LED 停留在 TIME_SEC 的數字。

(建議使用擁用比較高counter resolution的 TIM2~TIM5 timer)，請使用 polling 的方式取得 timer CNT register 值並換算成時間顯示到 7-SEG LED 上。
0.01 ≤ TIME_SEC ≤ 10000.00 (超過範圍請直接顯示0.00)

Note: 7-SEG LED 驅動請利用之前 Lab 所實作的 GPIO_init()、max7219_init() 與 Display () 函式呈現(須改成可呈現 2 個小數位)。

```
#include "stm321476xx.h"-
// You can use your way to store TIME_SEC. Maybe it is `int`or
`float` or any you want
#define TIME_SEC 12.70
extern void GPIO_init();
extern void max7219_init();
extern void Display();
void Timer_init( TIM_TypeDef *timer)
{
    //TODO: Initialize timer
}
void Timer_start(TIM_TypeDef *timer){
    //TODO: start timer and show the time on the 7-SEG LED.
}
int main()
{
   GPIO_init();
  max7219_init();
  Timer_init();
  Timer_start();
  while(1)
  {
    //TODO: Polling the timer count and do lab requirements
  }
}
```

3. 寫code過程

Timer initialize:

```
RCC->APB1ENR1 |= RCC_APB1ENR1_TIM2EN;
TIM2->CR1 &= 0x0000;
TIM2->PSC = 39999U;
TIM2->ARR = 99U;
TIM2->EGR = 0x0001;
```

Timer start:

```
TIM2->CR1 |= TIM_CR1_CEN;
TIM2->SR &= ~(TIM_SR_UIF);
```

Main function:
反正就是分成傳 毫秒和秒，再傳到處理時間的function
處理時間的function 大致想法如下：

```
for(int i = 1;i <= len;i++)
  if(i<3)max7219_send(i,milisecond % 10); milisecond /= 10;
  else
      max7219_send(i,second % 10); second /= 10;
      if(second == 0)break;
```

但要記得處理 秒.毫秒 中間的那個的 “點”。

6-3. Music keypad && 6-4. Modify LED brightness

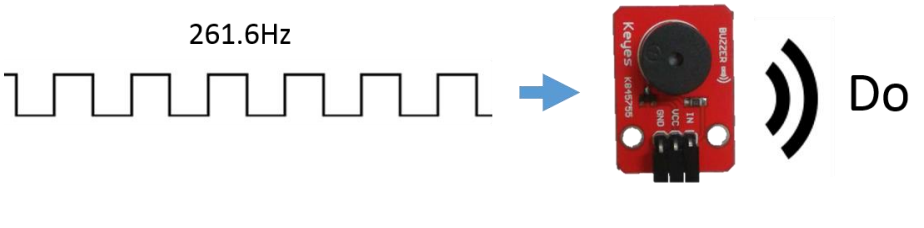
1. 實驗目的

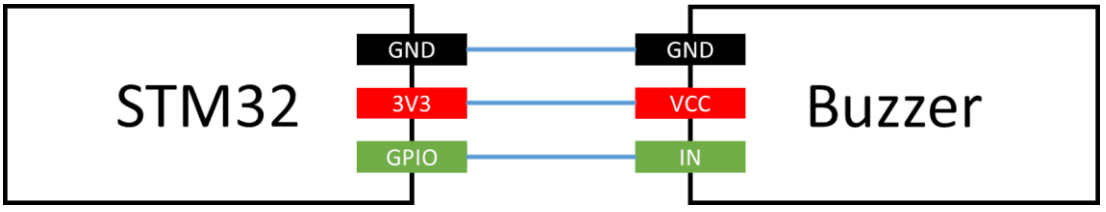
Understand the principle and application of PWM for STM32
瞭解 STM32 的 PWM 使用原理與應用

2. 實驗過程

The buzzer is divided into an active (self-excited) buzzer and a passive (excited) buzzer. The active buzzer directly designs the drive circuit into the buzzer, so it is only necessary to provide a DC voltage to make a sound, but the disadvantage is that the frequency of the sound cannot be changed. The external buzzer needs to provide an oscillating waveform to make a sound, and the frequency of the sound is the frequency of the input wave. Our LAB is using a passive buzzer.

蜂鳴器分爲有源(自激式)蜂鳴器和無源(他激式)蜂鳴器。有源蜂鳴器將驅動 電路直接設計到蜂鳴器中，因此只需提供直流電壓就可以發出聲音，但其缺點 是聲音的頻率無法更改。無源蜂鳴器外部需提供震盪波形才會發出聲音，其聲 音的頻率就是輸入波的頻率。我們這次 LAB 使用的是無源蜂鳴器。





The buzzer's VCC is connected to 3.3V, GND is connected to GND, and IN is connected to the GPIO pin.
蜂鳴器的VCC接3.3V、GND接GND、IN接GPIO腳位。

Please use the timer to generate and output the PWM signal with 50% Duty cycle, and use the keypad in Lab6 as the keyboard. When the user presses the different keypad buttons, the PWM square wave of the specific frequency (refer to the following table) is given to the buzzer. Do not make a sound when there is no button or press to a button that has no function. This lab will need to set registers such as GPIOx_AFRH, GPIOx_AFRL, TIMx_CCER, TIMx_CCMR1, TIMx_CCR1...

Note: Refer to STM32L4x6-Advanced-Arm®-based-32bitMCUs-Reference-Manual to understand the functions of these registers to complete the lab. Use STM32L476XX-DataSheet-Production-Data to find the pin corresponding to the timer channel.

請利用 timer 產生並輸出Duty cycle爲 50% 的 PWM 訊號，並以 Lab5 中的

keypad 爲鍵盤，當使用者在按下不同 keypad 按鍵時產生特定頻率(參考下表)的 PWM 方波給蜂鳴器，沒按鍵或按到沒功能的鍵時請不要發出聲音。本次實驗會需要設定 GPIOx_AFRH、GPIOx_AFRL、TIMx_CCER、TIMx_CCMR1、TIMx_CCR1...等 registers。

Note: 參考 STM32L4x6-Advanced-Arm®-based-32bitMCUs-Reference-Manual 瞭解這些register的功能完成此次實驗。並利用 STM32L476XX-DataSheet-Production-Data 找到timer channel所對應的腳位。

	X0	X1	X2	X3
Y0	Do	Re	Mi	
Y1	Fa	So	La	
Y2	Si	HDo		
Y3				

Keypad corresponds to the phonetic name

音名	Do	Re	Mi	Fa	So	La	Si	HDo
頻率(Hz)	261.6	293.7	329.6	349.2	392.0	440.0	493.9	523.3

Phonetic frequency correspondence table

Note: When the GPIO pin is set to PWM output, it needs to be set to the alternate function (AF) Mode, and set the AFRH and AFRL register according to the timer used. For details of the setting method, please refer to reference manual and datasheet.

Port		AF0	AF1	AF2	AF3
		SYS_AF	TIM1/TIM2/ TIM5/TIM8/ LPTIM1	TIM1/TIM2/ TIM3/TIM4/ TIM5	TIM8
Port B	PB0	-	TIM1_CH2N	TIM3_CH3	TIM8_CH2N
	PB1	-	TIM1_CH3N	TIM3_CH4	TIM8_CH3N
	PB2	RTC_OUT	LPTIM1_OUT	-	-
	PB3	JTDO-TRACESWO	TIM2_CH2	-	-
	PB4	NJTRST	-	TIM3_CH1	-
	PB5	-	LPTIM1_IN1	TIM3_CH2	-
	PB6	-	LPTIM1_ETR	TIM4_CH1	TIM8_BKIN2
	PB7	-	LPTIM1_IN2	TIM4_CH2	TIM8_BKIN
	PB8	-	-	TIM4_CH3	-
	PB9	-	IR_OUT	TIM4_CH4	-
	PB10	-	TIM2_CH3	-	-
	PB11	-	TIM2_CH4	-	-
	PB12	-	TIM1_BKIN	-	TIM1_BKIN_ COMP2
	PB13	-	TIM1_CH1N	-	-
	PB14	-	TIM1_CH2N	-	TIM8_CH2N
	PB15	RTC_REFIN	TIM1_CH3N	-	TIM8_CH3N

PortB AF mode selection table

```
extern void GPIO_init();
void GPIO_init_AF(){
//TODO: Initial GPIO pin as alternate function for buzzer. You can
choose to use C or assembly to finish this function.
}
void Timer_init(){
    //TODO: Initialize timer
}
void PWM_channel_init(){
    //TODO: Initialize timer PWM channel
}
int
  main(){ GPIO_init();
  GPIO_init_AF();
  Timer_init();
  PWM_channel_init();
    //TODO: Scan the keypad and use PWM to send the corresponding
frequency square wave to buzzer.
}
```

In the previous lab, the keypad adds two function buttons to adjust the Duty cycle of the PWM output (range 10%~90%, 5% adjustment per button). Use LED to replace buzzer. If successful, you should see that the LED's brightness changes with the duty cycle. Note: Pay attention to the relationship between frequency and duty cycle to set timer ARR and CCR registers.

在前一實驗中的 keypad 增加 2 個功能按鈕用以調整 PWM 輸出的 Duty cycle (範圍10%~90%，每按一次鍵調整5%)。

使用 LED 去替換蜂鳴器。如果成功應會看到 LED 亮度隨著 duty cycle 不同而有變化。

Note: 須注意頻率與 duty cycle 的關係來設定 timer ARR 與 CCR registers。

3. 寫code過程

因爲6-3和6-4實在太像，我就一起寫了，主要會以6-3敘述，6-4都是用差不多的觀念。
引用上次寫的keyboard的東東(但要稍作修改)，所以只要自己再實作Timer, GPIO_AF 和PWM即可。
GPIO_AF中，因爲我用的是PB3，又PB的預設是alternate function mode，故不用再設一次Moder，只要設AFR就好。

Timer initialize:

```
RCC->APB1ENR1 |= RCC_APB1ENR1_TIM2EN;
TIM2->CR1 &= 0x0000;
TIM2->ARR = (uint32_t)100;
TIM2->PSC = (uint32_t)39999;
TIM2->EGR = TIM_EGR_UG;
```

PWM initialize:

```
TIM2->CCMR1 &= ~TIM_CCMR1_OC2M;
TIM2->CCMR1 |= (0b0110 << TIM_CCMR1_OC2M_Pos);
TIM2->CCMR1 &= ~TIM_CCMR1_OC2PE;
TIM2->CCMR1 |= (0b1 << TIM_CCMR1_OC2PE_Pos);
TIM2->CR1 |= TIM_CR1_ARPE;
TIM2->CCR2 = (uint_32t)50;
TIM2->CCER |= TIM_CCER_CC2E;
```

我查了好多資料，在此附上比較厲害完整的網站，這樣之後想回頭看就不用再找了

https://www.itread01.com/content/1546328226.html

http://mcu.eetrend.com/content/2017/100009084.html

https://blog.csdn.net/ABAP_Brave/article/details/52850549

https://www.cnblogs.com/kuotian/p/5631652.html

實驗心得與結語

這次lab6的四個小題讓我學會了怎麼靈活的使用Timer和AFR，對於組合語言的整體寫法也更有sense了。

過程中查了好多資料，因爲老師教的內容完全不夠，特別是第三和第四小題。感覺自己的實力又提升了，希望明天的改程式題不要太難。