Lab6 STM32 Clock and Timer

實驗六 STM32 Clock and Timer

1. Lab objectives 實驗目的

- Understand the various clock source usage and modification of STM32
- Understand the principle of using STM32 timer
- Understand the principle and application of PWM for STM32
- 瞭解 STM32 的各種 clock source 使用與修改
- 瞭解 STM32 的 timer 使用原理
- 瞭解 STM32 的 PWM 使用原理與應用

2. Lab principle 實驗原理

2.1. Timer and Counter

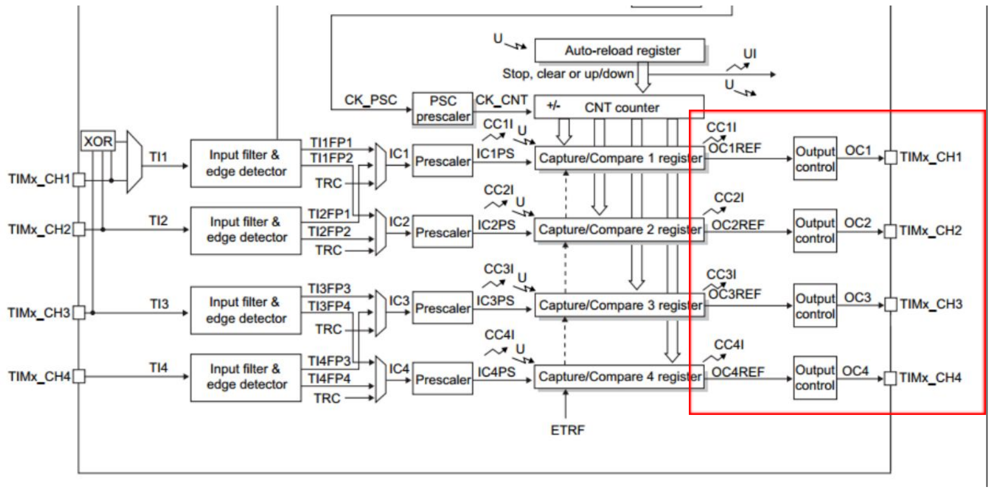
Please refer to 009-MCSL-CounterTimer lecture slide.

請參考上課 009-MCSL-CounterTimer 講義。

2.2. Timer PWM output mode

In the STM32 system, the Timer is used to generate the PWM output, which is mainly set by the capture/compare mode register (TIMx_CCMR1) and TIMx_CCRx registers and enabled by TIMx_CCER.

在 STM32 中利用 Timer 產生 PWM 輸出，主要通過 capture/compare mode register(TIMx_CCMR1) 與 TIMx_CCRx registers 設定並利用 TIMx_CCER 啟動。



The general PWM has two modes, mode1 and mode2, and the corresponding output is in the counter mode.

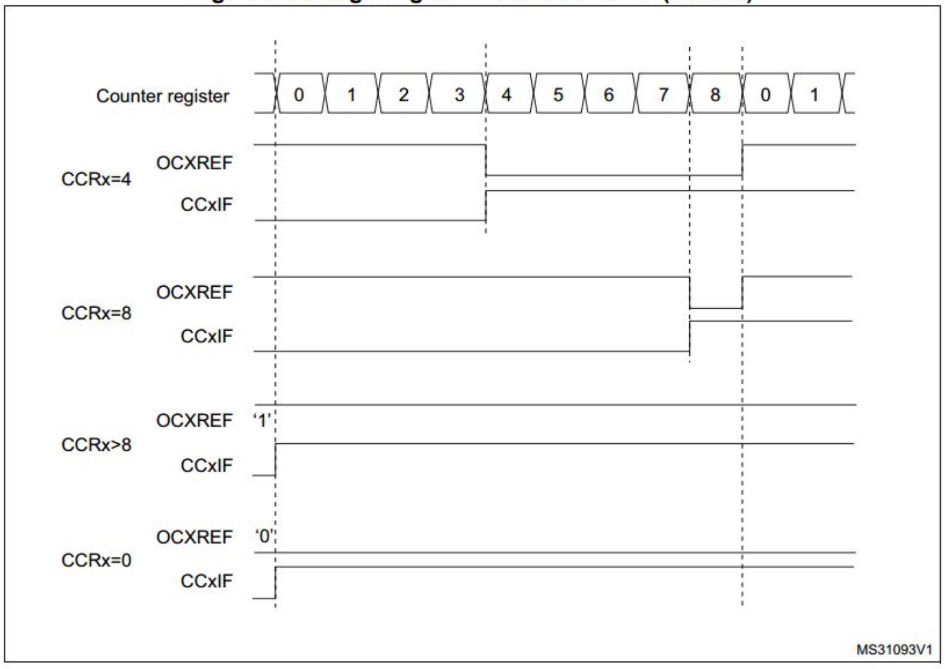
而一般 PWM 有分 mode1 與 mode2 兩種模式，而在計數器上數模式時其對應的輸出為

- PWM mode1: Channel is active as long as TIMx_CNT < TIMx_CCRx else inactive.
- PWM mode2: Channel is inactive as long as TIMx_CNT < TIMx_CCRx else active.

In addition, according to different special purposes, it can be divided into Combined PWM mode and Asymmetric PWM mode.

另外依不同特殊用途又可分 Combined PWM mode 與 Asymmetric PWM mode。

Figure 279. Edge-aligned PWM waveforms (ARR=8)



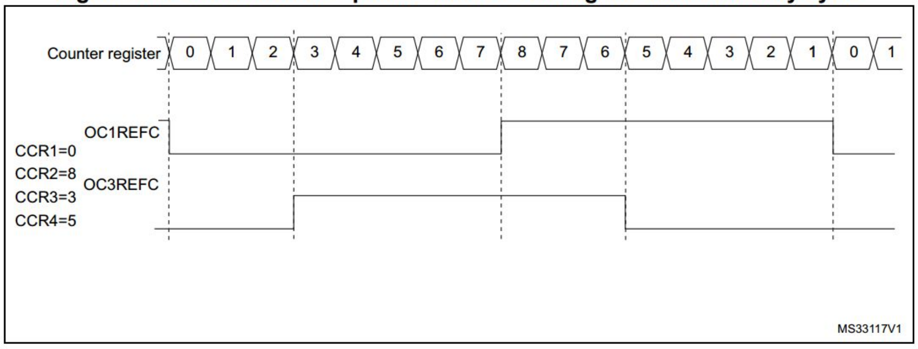
In the example above, when CCRx=4, ARR=8, the waveform of the duty cycle (the ratio of 1 level to 0 level in unit time) is 50%, CCRx=8, ARR=8 can get duty cycle= 1-1/8=87.5% waveform.

以上圖範例來說當 **CCRx**=4, **ARR**=8 可以得到 duty cycle (在單位時間內 1 準位與0 準位的比例)為 50% 的波形， CCRx=8,ARR=8 可得 duty cycle = 1 - ⅛ = 87.5% 的波形。

To output a more complex PWM and different duty cycle waveforms can also be achieved using Asymmetric PWM mode.

要輸出比較複雜的 PWM 與不同 duty cycle 波形也可以利用 Asymmetric PWM mode 來達成。

Figure 281. Generation of 2 phase-shifted PWM signals with 50% duty cycle



Reference:STM32L4x6-Advanced-Arm®-based-32bitMCUs-Reference-Manual chapter 30.3.11 PWM mode

3. Steps 實驗步驟

3.1. Modify System Clock(SYSCLK) and CPU Clock(HCLK)

Requirement:

Please use busy waiting to implement the "delay" function. When the CPU clock is 4MHz, the delay is 1 second. Then, implement an infinite loop which keeps turning on/off the LED light at the interval of single "delay". **When users press the user button**, the CPU system clock (HCLK) will be changed following the sequence. 1MHz -> 6MHz -> 10MHz -> 16MHz -> 40MHz -> 1MHz ->...

使用 busy waiting 的方式實做 "delay" 函式。 當 CPU 時鐘為 4MHz 時，延遲時間為1秒。接著，實施一個無限循環，以一個“延遲”的間隔持續打開/關閉LED燈。當使用者按下按鈕時，CPU系統時鐘（HCLK）將按照以下順序進行更改。1MHz-> 6MHz-> 10MHz-> 16MHz-> 40MHz-> 1MHz-> …

main.c

```
void GPIO_init();
void Delay1sUnder4MHz();
void Set_HCLK(int freq);

int main(){
  // Do initializations.
  int freq[] = {1, 6, 10, 16, 40};
  for(;;){
    // change LED state
    Delay1sUnder4MHz();
    // change HCLK if button pressed
   }
}
void Set_HCLK(int freq){
  // 1. change to the temporary clock source if needed
  // 2. set the target clock source
  // 3. change to the target clock source
}
```

Note:

Some CPU frequencies must be generated using the frequency multiplier and divider of the PLL module. To modify the PLL configuration, please refer to "STM32L4x6 advanced Arm®-based 32-bit MCUs, Section 6.2.5, Section 6.2.8 and Section 6.4.
必須使用PLL模塊的倍頻器和分頻器來生成某些CPU頻率。 要修改PLL配置，請參考“ STM32L4x6基於Arm®的高級32位MCU，第6.2.5節,6.2.8節及6.4節。

Hint: You can't directly generate 6MHz HCLK by PLL_N, PLL_M, PLL_R.
Trace the path from clock source to the HCLK on the clock tree and find out the solution.
您無法直接通過 PLL_N，PLL_M，PLL_R 產生 6MHz 的 HCLK。請在 clock tree 上追蹤 (trace) 時鐘源到 HCLK 的路徑，找出解決方案。

3.2. Timer 計時器

Requirement:

Please complete Timer_init() and Timer_start() in main.c below and use STM32 TIMx to implement a timer that counts up from 0 (Upcounting) and stop it at TIME_SEC seconds. While the timer is counting, you also need polling the TIMx_CNT register, and convert the counter value to time in second(s) with 2 decimal places. Then, show it on the 7-SEG LED.

請完成下面 main.c 中的 Timer_init() 和 Timer_start()，並使用 STM32 TIMx 實現一個從 0 開始的上數計時器並讓它停在 TIME_SEC 秒。在計時器運作的過程中，你必須使用輪詢的方式獲取 TIMx_CNT 暫存器的值，並轉換為秒數（取至小數點後第二位）。然後，將其顯示在7-SEG LED上。

Note:

- You can assume that 0.01 ≤ TIME_SEC ≤ 10000.00.
- It is recommended to use TIM2~TIM5 timer with higher counter resolution.
- Remember to show the decimal point.
- 您可以假設 0.01 ≤ TIME_SEC ≤ 10000.00。
- 建議使用計數器精度更高的 TIM2～TIM5 計時器。
- 記得要顯示小數點。

main.c

```c
#include "stm32l476xx.h"
#define TIME_SEC 12.70
extern void GPIO_init();
extern void max7219_init();
extern void Display();
```

```c
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
  }
}
```

Demo video with TIME_SEC = 12.7 : https://reurl.cc/WLgrrD

3.3. Modify LED brightness 調整 LED 亮度

調轉速和亮度

Requirement:

Please initialize some GPIO pins. One outputs the PWM signal generated by the timer, and the other is used to receive input from the keyboard. There are two function buttons on the keyboard, which are used to adjust the duty cycle in the range of 10% to 90%, where "#" is for increasing and "*" for decreasing, each time adjusting 5%. After the above steps are finished, feeding the PWM signal to the LED. You will find that the brightness of the LED changes with the duty cycle.

To verify your PWM output, you can also connect the oscilloscope probe to the corresponding pins. Please refer to lab6_note for more operation detail.

請初始化一些GPIO引腳。 一個用於輸出由 Timer 產生的 PWM 信號，其他用於接收來自 Keypad 的輸入。 鍵盤上有兩個用於調節佔空比的功能鍵，調節範圍從 10% 到 90%，其中 "#" 用於增加 " *" 用於減少，每次調整 5%。在完成上述步驟後，將 PWM 訊號輸入 LED 。你將會發現 LED 的亮度隨著 duty cycle 改變。

要驗證您的PWM輸出，您還可以連接示波器的探針到對應的針腳。有關更多操作細節，請參考 lab6_note。

```c
main.c

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
}
```

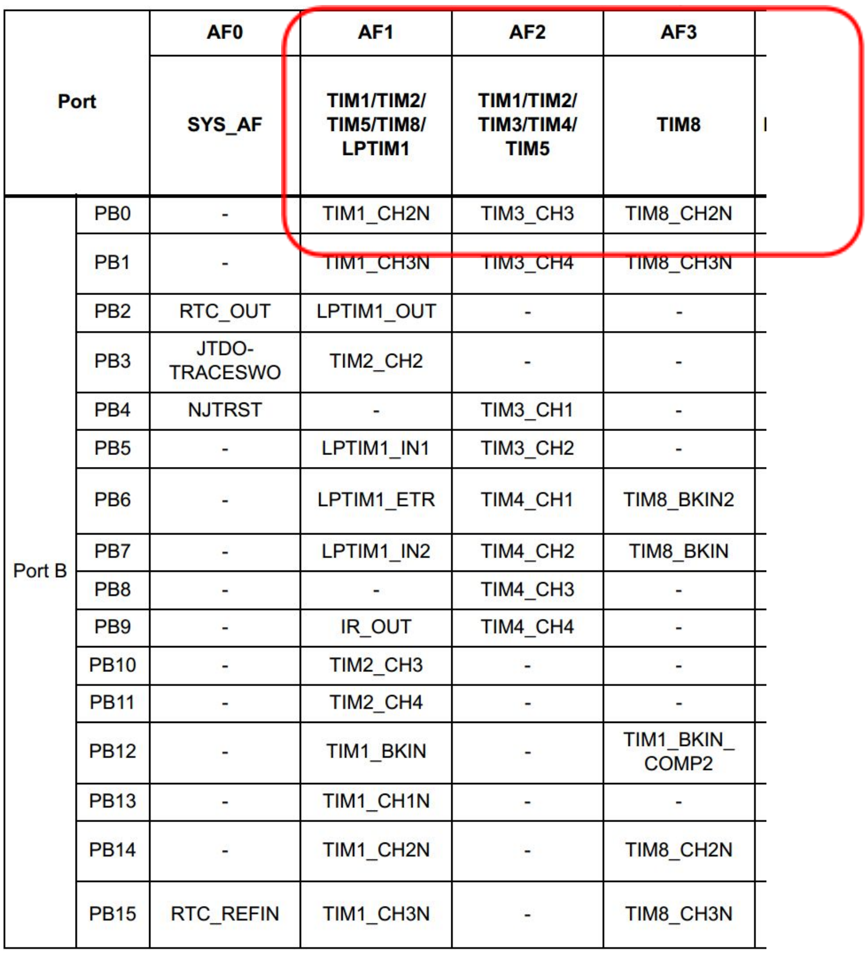
```
void PWM_channel_init(){
    //TODO: Initialize timer PWM channel
}
int main(){
    GPIO_init();
    GPIO_init_AF();
    Timer_init();
    PWM_channel_init();
    //TODO: Scan the keypad and use PWM to send the
corresponding frequency square wave to the buzzer.
}
```

Hint:

In this lab, you may need to set registers such as GPIOx_AFRH, GPIOx_AFRL, TIMx_CCER, TIMx_CCMR1, TIMx_ARR, TIMx_CCR1…

本次實驗你可能會需要設定 GPIOx_AFRH, GPIOx_AFRL, TIMx_CCER, TIMx_CCMR1, TIMx_ARR, TIMx_CCR1…等暫存器。

GPIO_AF mode:



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

When the GPIO pin is used as 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 more details, please refer to the following manuals.

- STM32L4x6-Advanced-Arm®-based-32bitMCUs-Reference-Manual for GPIO registers.
- STM32L476XX-DataSheet-Production-Data for the GPIO pins corresponding to the timer channel.

當 GPIO Pin 做 PWM output 時需將其設定為 alternate function(AF) Mode, 並根據根據所對應使用的 timer 設定 **AFRH** 與 **AFRL** register, 細節請參考下列手冊。

- 暫存器設置, STM32L4x6-Advanced-Arm®-based-32bitMCUs-Reference-Manual。
- Timer 輸出對應的 GPIO 腳位, STM32L476XX-DataSheet-Production-Data 。

4, 2, $\frac{4}{8}$, $\frac{4}{5}$

1 + 0.125 + 0.125 + 0.12 + 0.125

1.85