Pulse Width Modulation
脈寬調變

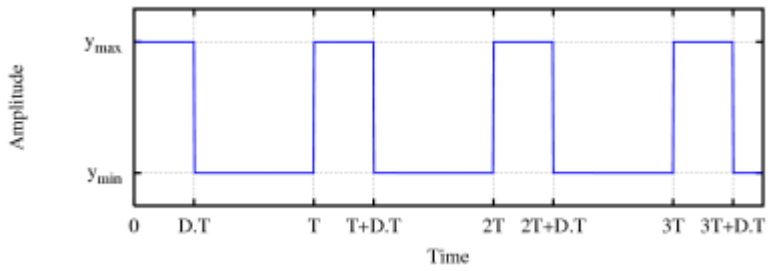
4/19/2012

Outline

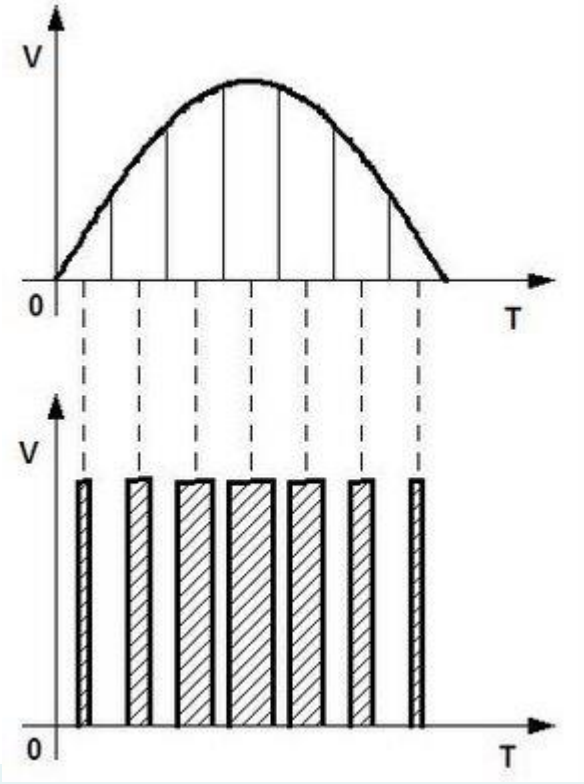
- **P**ulse **W**idth **M**odulation **脈寬調變之原理與應用**
- **實習範例：使用PWM 控制方波輸出－1K、40K、100K、 1MHz**
- **實習範例：使用PWM 控制喇叭音調　Frequency of Music Notes**
- **實習範例：使用PWM 產生雙聲道音樂與合弦**

Pulse Width Modulation 脈寬調變

- PWM (Pulse Width Modulation) 稱為脈波寬度調變，常用於直流馬達的控制、電源變換器之穩壓控制、甚至是直流轉換交流弦波的控制等，是控制直流馬達轉速最常見的方法



方波週期、高與低之寬度均為可調



NuMicro PWM Generator 脈寬調變產生器

- Four PWM Generators, each generator supports 四套脈寬調變產生器
  - One 8-bit prescaler 預分頻器
  - One clock divider 時脈除頻器
  - Two PWM-timers for two outputs (雙計數器供給雙輸出), each timer includes
    - A 16-bit PWM down-counter (16位元遞減計數器)
    - A 16-bit PWM reload value register (CNR) (16位元計數值暫存器)
    - A 16-bit PWM compare register (CMR) (16位元計數比較器)
  - One dead-zone generator 死區產生器
  - Two PWM outputs (可做雙輸出，供馬達控制使用)

NuMicro PWM Generator 脈寬調變產生器

- 8 PWM channels or 4 PWM paired channels (8個或四對脈寬調變通道)
- 16 bits resolution (16位元)
- PWM Interrupt synchronized with PWM period (脈寬調變之中斷與周期脈寬同步)
- Single-shot or Continuous mode PWM (單次或連續模式)

Overview

- 8 independent PWM outputs, PWM0~PWM7,
- or as 4 complementary PWM pairs, (PWM0, PWM1), (PWM2, PWM3), (PWM4, PWM5) and (PWM6, PWM7)
- PWM0 and PWM1 perform complementary PWM paired function;
- Each PWM interrupt source with its corresponding enable bit can cause CPU to request PWM
- The PWM generators can be configured as
  - one-shot mode to produce only one PWM cycle signal or
  - auto-reload mode to output PWM waveform continuously.

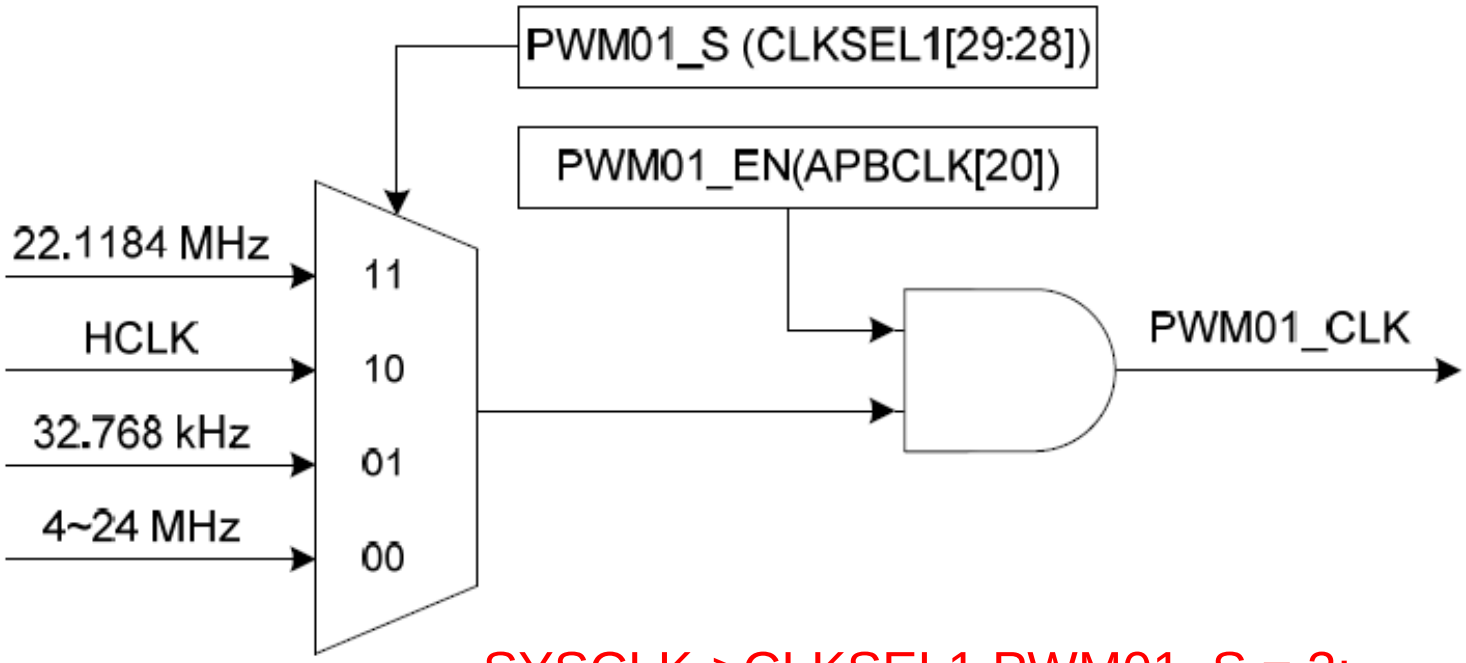
PWM function features

- PWM group has two PWM generators. Each PWM generator supports one 8-bit prescaler, one clock divider, two PWM-timers (down counter), one dead-zone generator and two PWM outputs.
- Up to 16-bit resolution
- PWM Interrupt request synchronized with PWM period
- One-shot or Auto-reload mode PWM
- Up to 2 PWM group (PWMA/PWMB) to support 8 PWM channels or 4 PWM paired channels

Capture Function Features

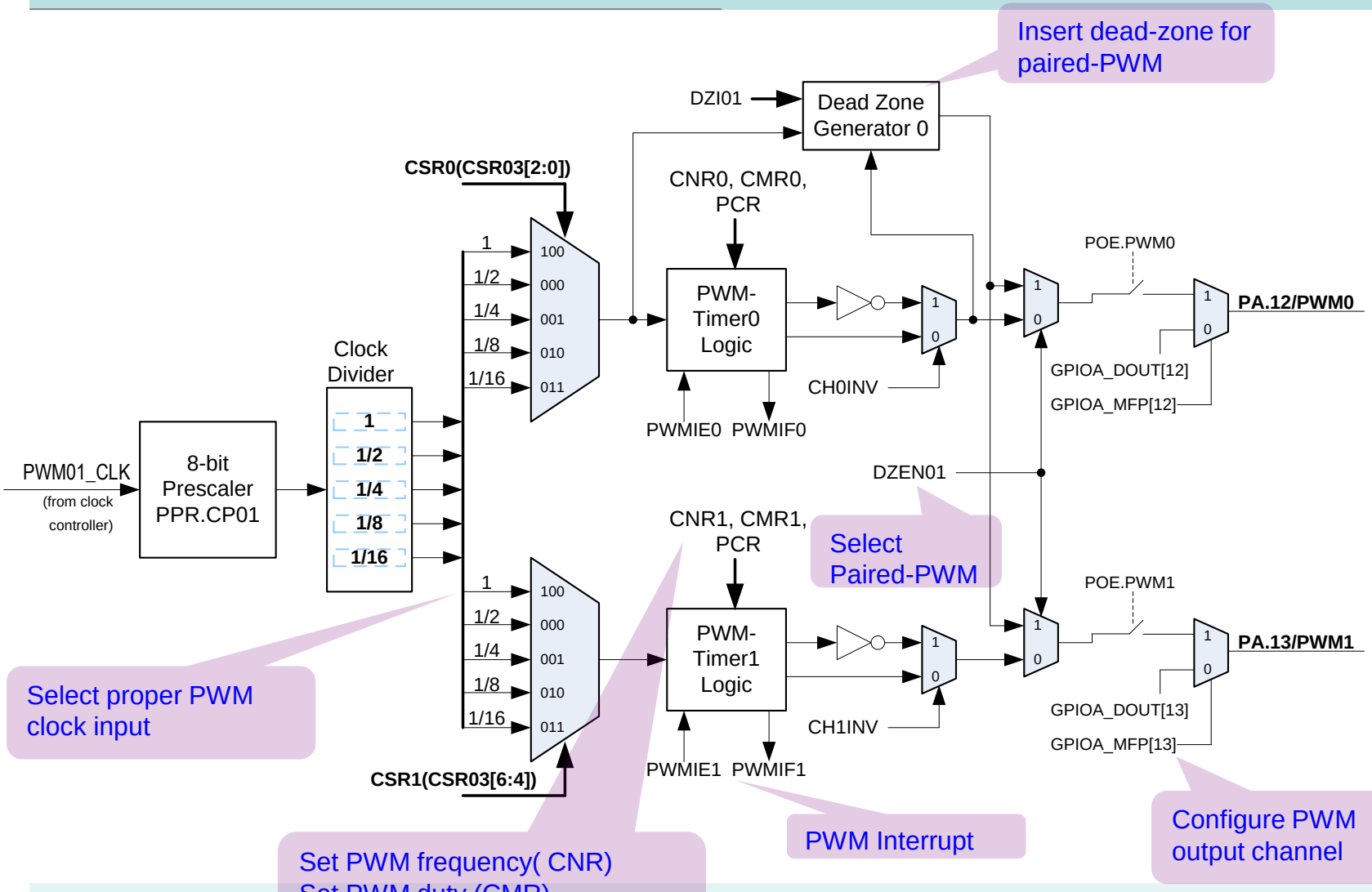
- Timing control logic shared with PWM Generators
- Support 8 Capture input channels shared with 8 PWM output channels
- Each channel supports one rising latch register (CRLR), one falling latch register (CFLR) and Capture interrupt flag (CAPIFx)

PWM Generator 0 Clock Source Control




```
SYSCLK->CLKSEL1.PWM01_S = 3;
SYSCLK->APBCLK.PWM01_EN = 1;
```

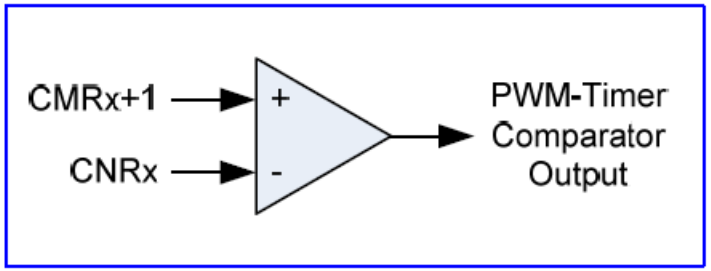
PWM Block Diagram 方塊圖

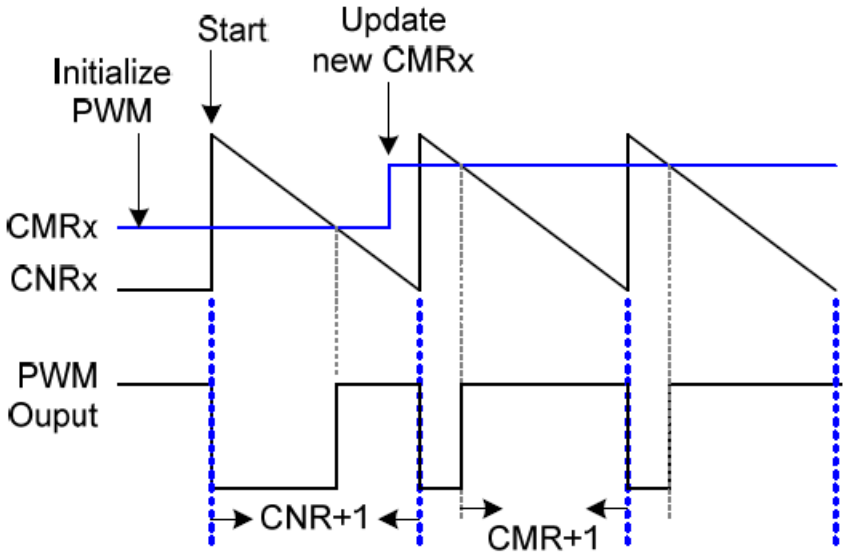


PWM-Timer Operation

- GPIO pins must be configured as PWM function.
- The PWM period and duty control are configured by PWM down-counter register (CNR) and PWM comparator register (CMR).

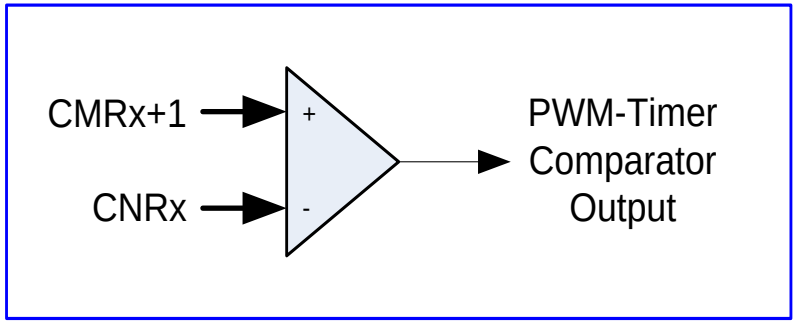
- Legend of Internal Comparator Output of PWM-Timer



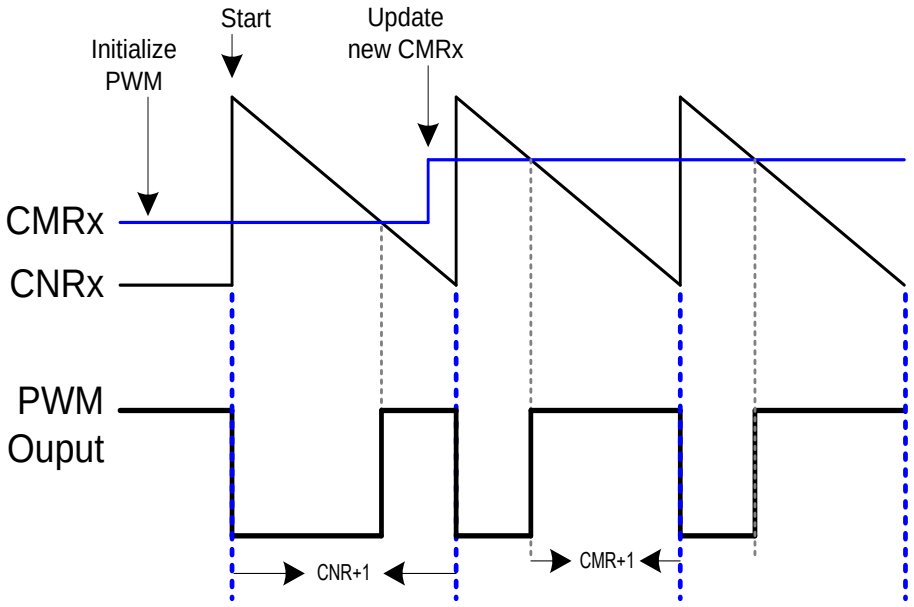


PWM Frequency & Duty Cycle 頻率與責任週期

- ***PWM Frequency*** = 頻率
  PWMxy_CLK/(prescale+1)*(clock divider)/(CNR+1);
  where xy = 01, 23, 45 or 67, the selected PWM channel.
- ***Duty ratio*** = (CMR+1)/(CNR+1) 責任週期或佔空比



CMRx+1 >= CNRx: PWM output high.
CMRx+1 < CNRx: PWM output low



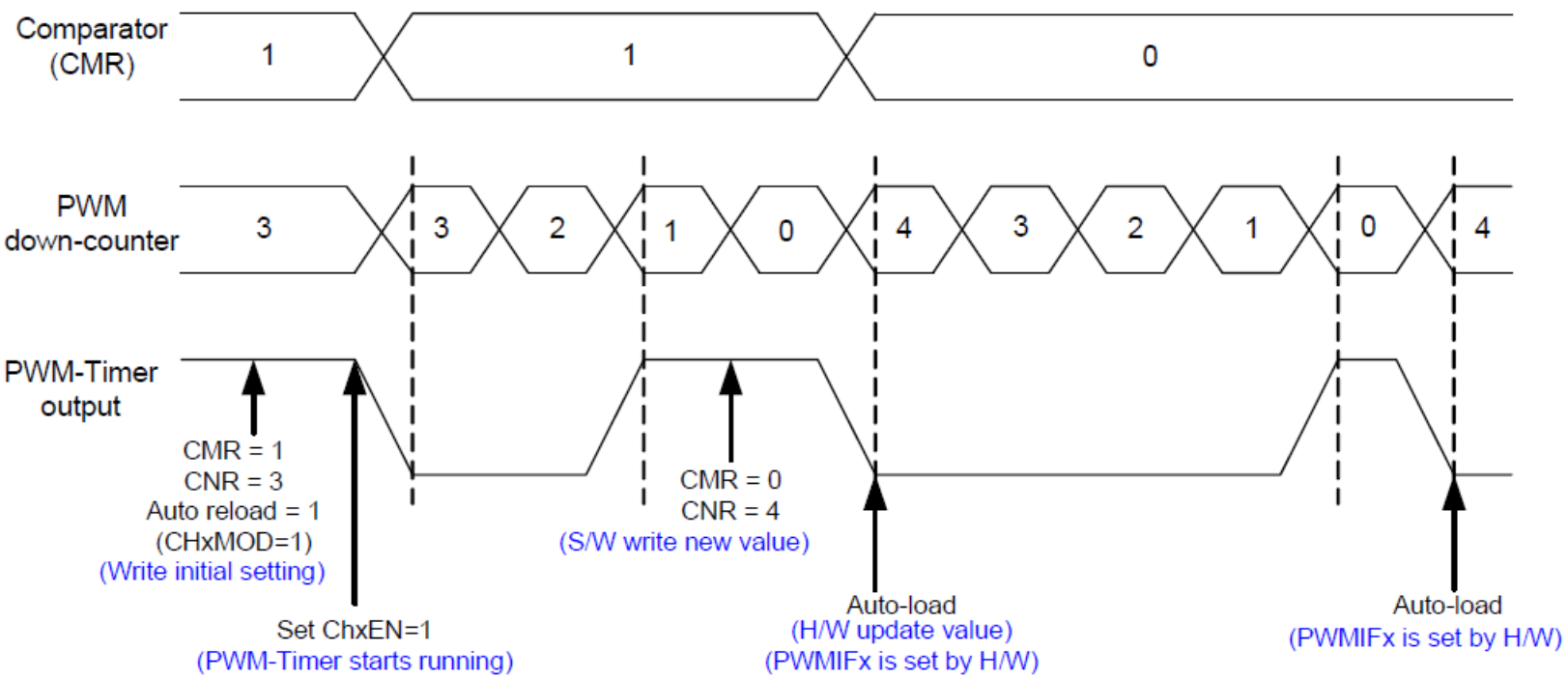
PWM-Timer Operation

- PWM frequency = PWMxy_CLK/[(prescale+1)*(clock divider)*(CNR+1)];
- ***Duty ratio*** = (CMR+1)/(CNR+1)
- CMR>=CNR, PWM=1
- CMR=0, PWM=1/(CNR+1), low width = CNR; high width = 1
- CMR<CNR, PWM=(CMR+1)/(CNR+1)

  low width= (CNR-CMR); high width = (CMR+1)

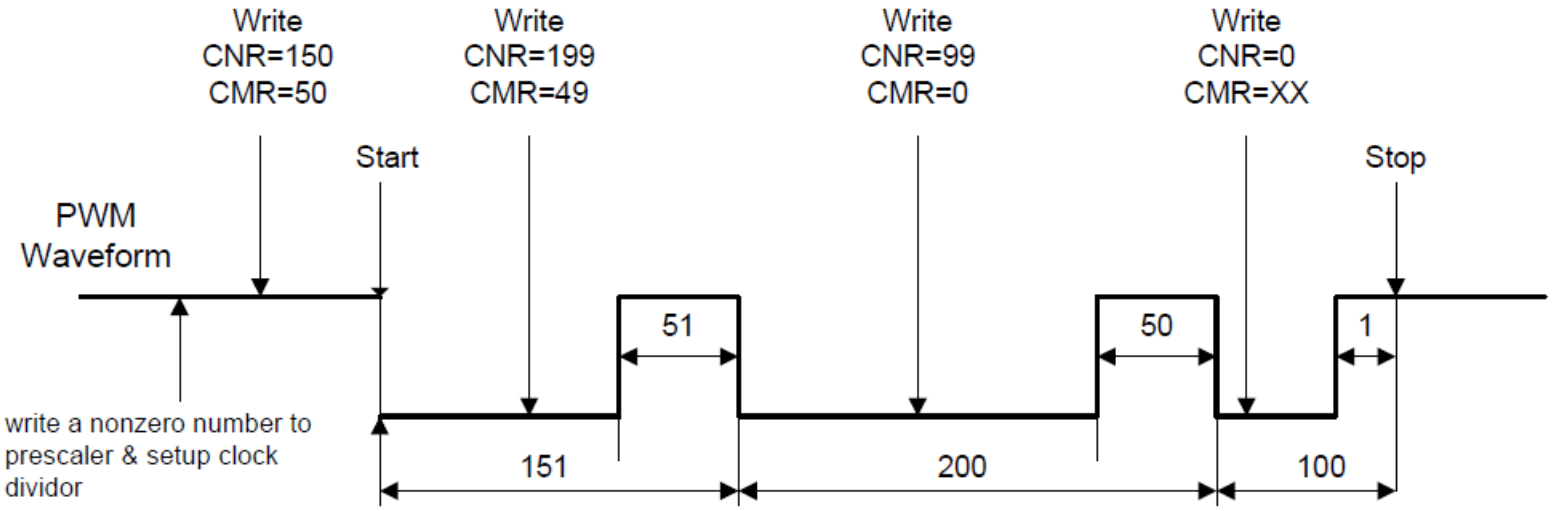
PWM-Timer Operation Timing

- CNR=4: PWM計數4,3,2,1,0
- CMR=1: 當PWM down-count大於CMR，輸出低電位；否則輸出高電位。即使CMR=0，還是可輸出1單位的高電位

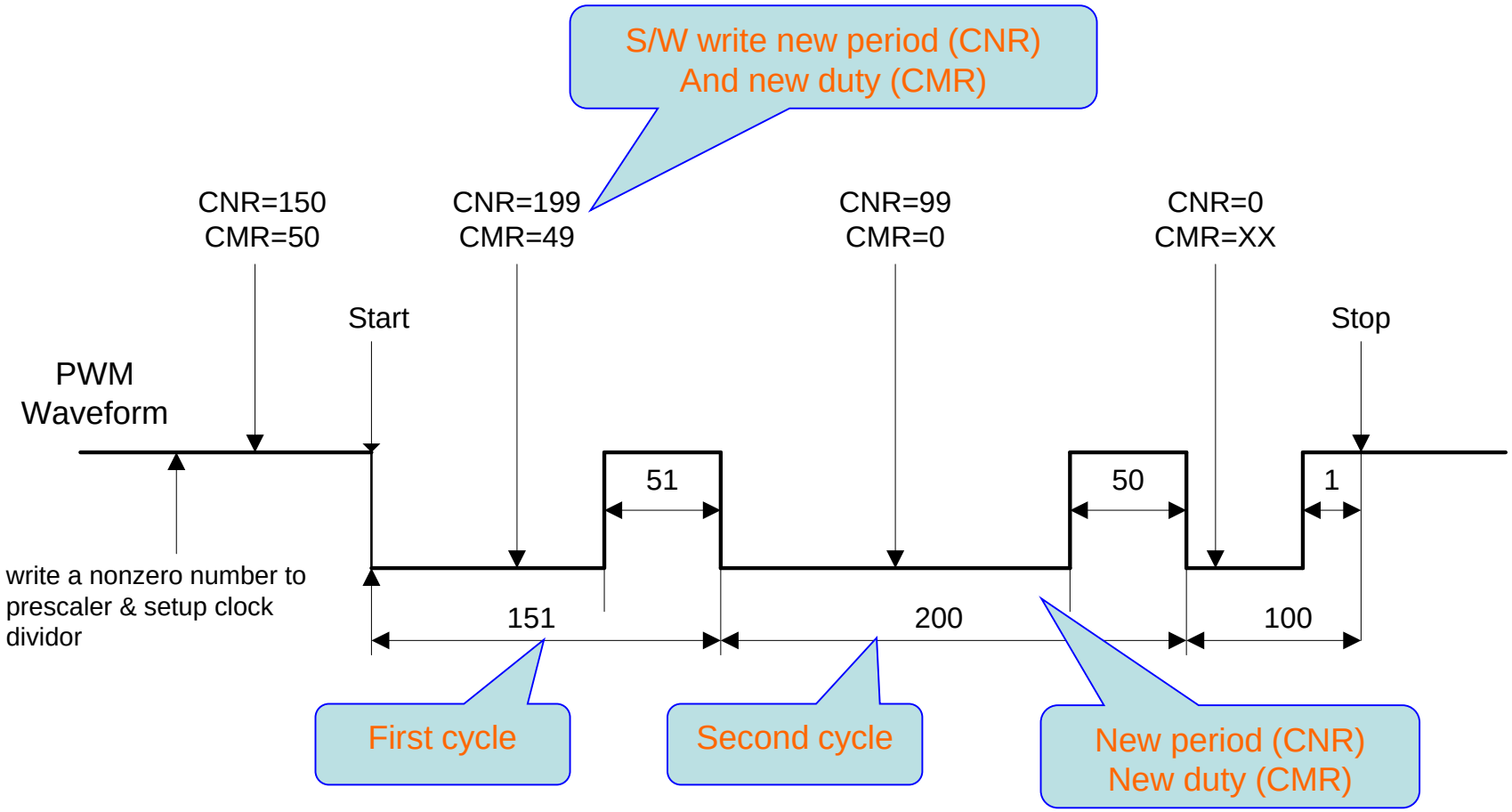


PWM Double Buffering, Auto-reload and One-shot Operation

- double buffering function: the reload value is updated at the start of next period without affecting current timer operation.
- The PWM counter value can be written into CNRx and current PWM counter value can be read from PDRx.

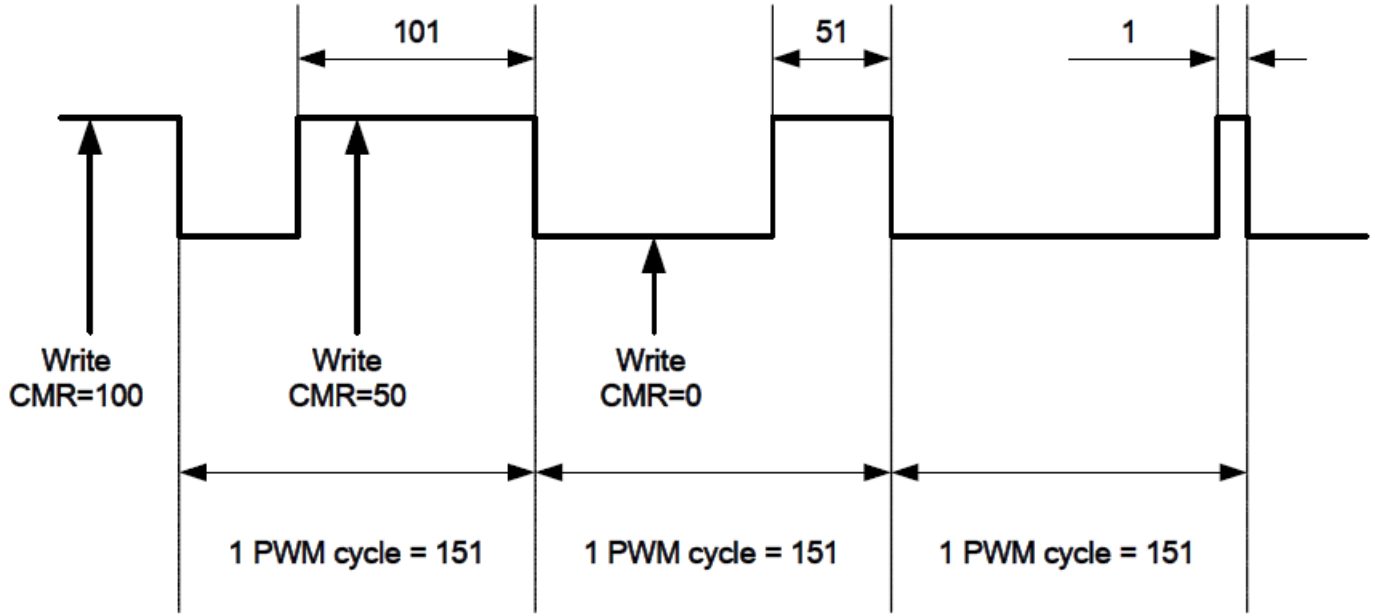


PWM double buffering scheme



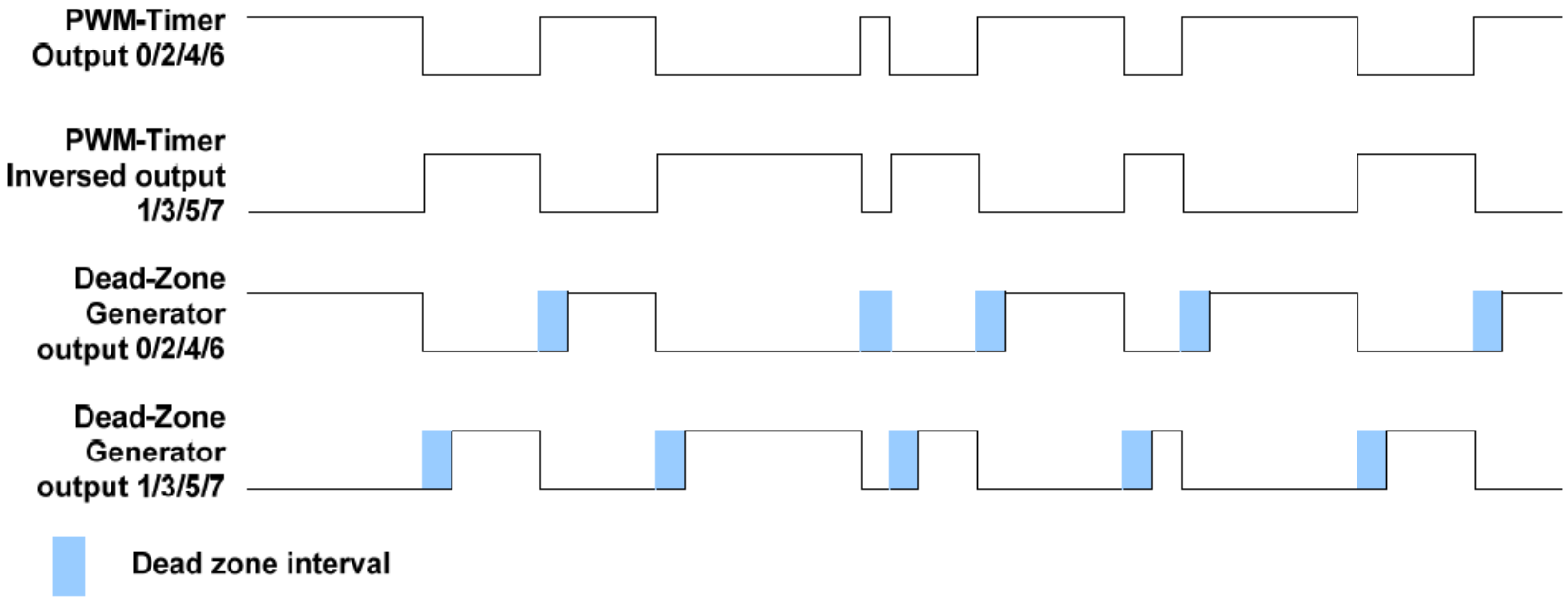
Modulate Duty Ratio

- The double buffering function allows CMRx written at any point in current cycle.
- The loaded value will take effect from next cycle.



Dead-Zone Generator

- Dead Zone 產生器的目的
  - 避免 paired-PWM 輸出之重疊，例如馬達驅動應用中, 要避免順逆方向電流同時打開
  - 方法是在每個配對PWM輸出上，插入延遲時間(稱為Dead Zone)
- 提供8-bit dead-zone timer by PWM clock

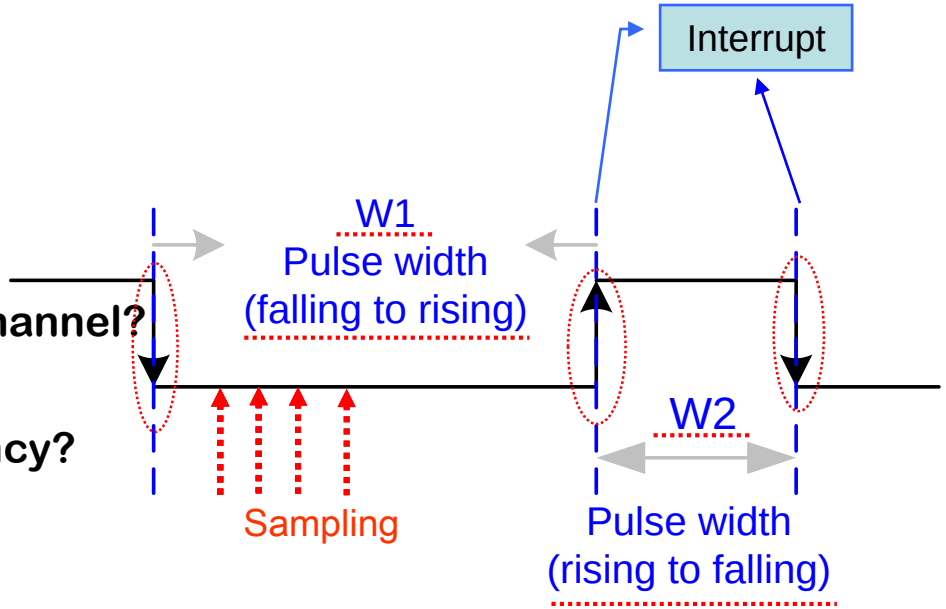


Capture Operation

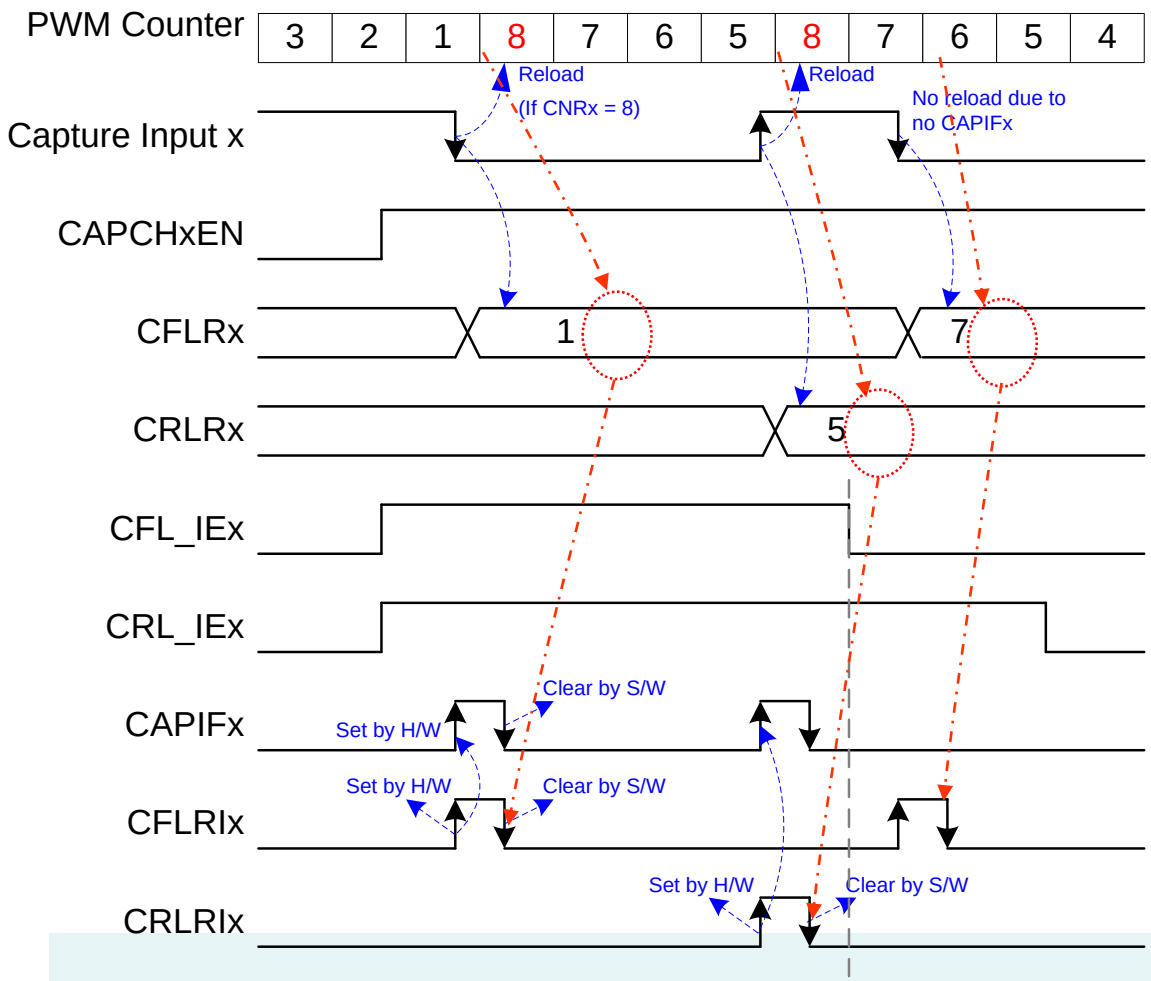
- The Capture 0 and PWM 0 share one timer that included in PWM 0
- The capture always latches PWM-counter to CRLRx when input channel has a rising transition and latches PWM-counter to CFLRx when input channel has a falling transition.
- Whenever the Capture controller issues a capture interrupt, the corresponding PWM counter will be reloaded with CNRx at this moment.

Capture Input (捕捉輸入)

- Captured by ***Rising or Falling Edge*** *(根據上下緣捕捉)*
- Measure ***Pulse Width***
  - Rising to Falling (W2)
  - Falling to Rising (W1)
  - Rising to Rising (W1+W2)
  - Falling to Falling (W1+W2)
- Capture ***Input Channel***
  - How to set and change input channel?
- Sampling ***Frequency***
  - How to set and change frequency?
- Capture Function ***Interrupt***
- Captured pulse width ***data***
  - Rising Latch Register (CRLR)
  - Falling Latch Register (CFLR)



Capture Input - Timing Diagram 捕捉時序圖



Note: X=0~7

The PWM counter will be reloaded with CNRx when a capture interrupt flag (CAPIFx) is set

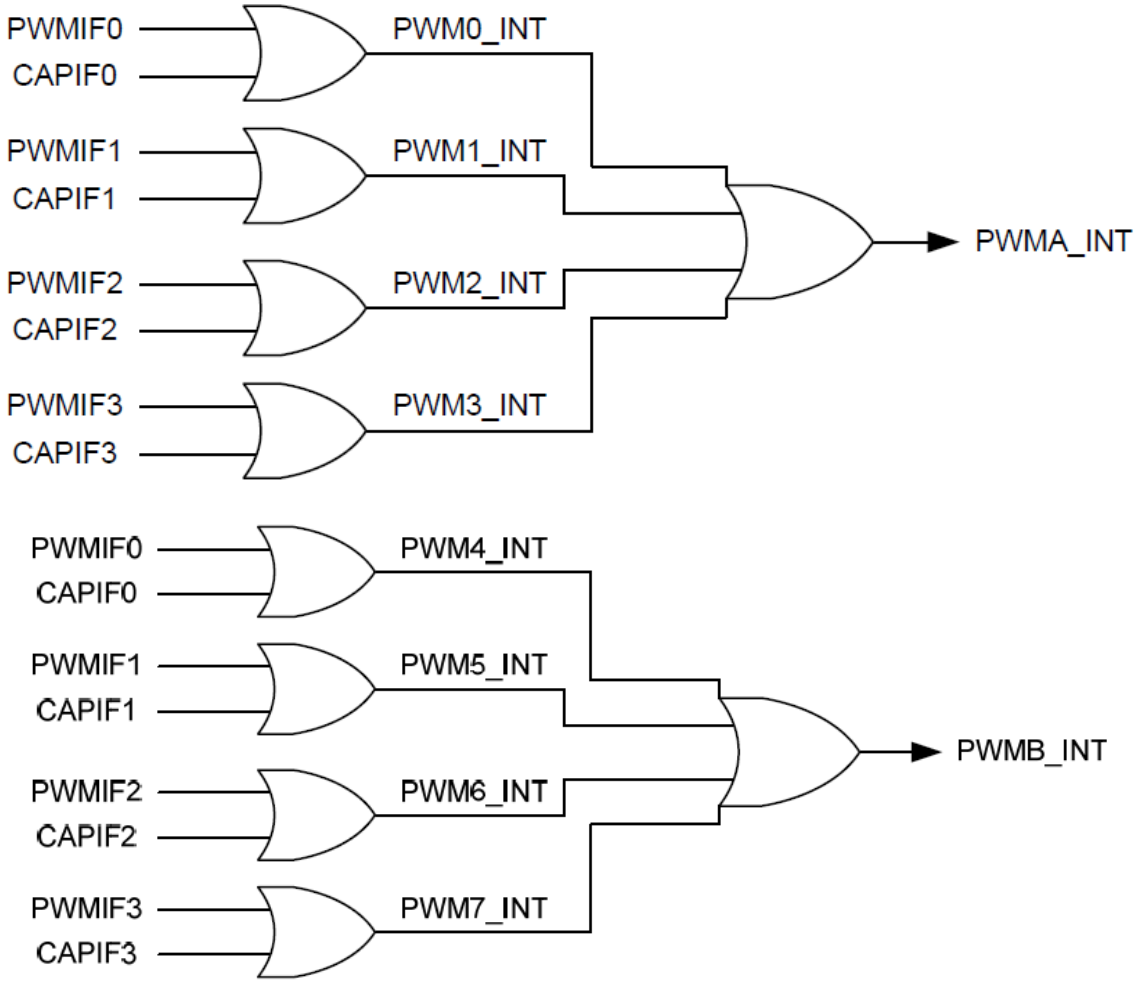
The channel low pulse width is (CNR - CRLR).

The channel high pulse width is (CNR - CFLR).

PWM-Timer Interrupt Architecture

- There are eight PWM interrupts, PWM0_INT~PWM7_INT, which are divided into PWMA_INT and PWMB_INT for Advanced Interrupt Controller (AIC).
- PWM 0 and Capture 0 share one interrupt, PWM1 and Capture 1 share the same interrupt and so on.
- PWM function and Capture function in the same channel cannot be used at the same time.
- Figure 5-47 and Figure 5-48 demonstrates the architecture of PWM-Timer interrupts.

PWM Group A/B PWM-Timer Interrupt Architecture Diagram



PWM-Timer Start Procedure

- The following procedure is recommended for starting a PWM drive.

1. Setup clock source divider select register (CSR)
2. Setup prescaler (PPR)
3. Setup inverter on/off, dead zone generator on/off, auto-reload/one-shot mode and Stop PWM-timer (PCR)
4. Setup comparator register (CMR) for setting PWM duty.
5. Setup PWM down-counter register (CNR) for setting PWM period.
6. Setup interrupt enable register (PIER) (option)
7. Setup corresponding GPIO pins as PWM function (enable POE and disable CAPENR) for the corresponding PWM channel.
8. Enable PWM timer start running (Set CHxEN = 1 in PCR)

PWM-Timer Re-Start Procedure in Single-shot mode

- After PWM waveform generated once in PWM one-shot mode, PWM-Timer will stop Automatically. The following procedure is recommended for re-starting PWM single-shot waveform.

1. Setup comparator register (CMR) for setting PWM duty.

2. Setup PWM down-counter register (CNR) for setting PWM period. After setup CNR, PWM wave will be generated

PWM-Timer Stop Procedure

- **Method 1:**
- Set 16-bit down counter (CNR) as 0, and monitor PDR (current value of 16-bit down-counter). When PDR reaches to 0, disable PWM-Timer (CHxEN in PCR). ***(Recommended)***
- **Method 2:**
- Set 16-bit down counter (CNR) as 0. When interrupt request happened, disable PW (CHxEN in PCR). ***(Recommended)***
- **Method 3:**
- Disable PWM-Timer directly ((CHxEN in PCR). ***(Not recommended)*** The reason why method 3 is not recommended is that disable CHxEN will immediately stop PWM output signal and lead to change the duty of the PWM output, this may cause damage to the control circuit of motor

Capture Start Procedure

1. Setup clock source divider select register (CSR)
2. Setup prescaler (PPR)
3. Setup channel enabled, rising/falling interrupt enable and input signal inverter on/off (CCR0, CCR2)
4. Setup auto-reload mode, Edge-aligned type and Stop PWM-timer (PCR)
5. Setup PWM down-counter (CNR)
6. Enable PWM timer start running (Set CHxEN = 1 in PCR)
7. Setup corresponding GPIO pins as capture function (disable POE and enable CAPENR) for the corresponding PWM channel.

Register Map

Register	Description
PPR	PWM Group A/B Prescaler Register
CSR	PWM Group A/B Clock Source Divider Select Register
PCR	PWM Group A/B Control Register
CNR0	PWM Group A/B Counter Register 0
CMR0	PWM Group A/B Comparator Register 0
PDR0	PWM Group A/B Data Register 0
PBCR	PWM Group A/B backward compatible Register
PIER	PWM Group A/B Interrupt Enable Register
PIIR	PWM Group A/B Interrupt Indication Register

Register Map

Register	Description
CCR0	PWM Group A/B Capture Control Register 0
CRLR0	PWM Group A/B Capture Rising Latch Register (Channel 0)
CFLR0	PWM Group A/B Capture Falling Latch Register (Channel 0)
CAPENR	PWM Group A/B Capture Input 0~3 Enable Register
POE	PWM Group A/B Output Enable for channel 0~3

PWM function

function	Description
DrvPWM_IsTimerEnabled	get PWM specified timer enable/disable state
DrvPWM_SetTimerCounter	set the PWM specified timer counter.
DrvPWM_GetTimerCounter	get the PWM specified timer counter value
DrvPWM_EnableInt	enable the PWM timer/capture interrupt and install the call back function.
DrvPWM_DisableInt	disable the PWM timer/capture interrupt.
DrvPWM_ClearInt	clear the PWM timer/capture interrupt flag.
DrvPWM_GetIntFlag	get the PWM timer/capture interrupt flag
DrvPWM_GetRisingCounter	get value which latches the counter when there‟s a rising transition.

PWM function

function	Description
DrvPWM_GetFallingCounter	get value which latches the counter when there‟s a falling transition.
DrvPWM_GetCaptureIntStatus	Check if there‟s a rising / falling transition
DrvPWM_ClearCaptureIntStatus	Clear the rising / falling transition indicator flag
DrvPWM_Open	Enable PWM engine clock and reset PWM.
DrvPWM_Close	Disable PWM engine clock and the Capture Input / PWM Output Enable function.
DrvPWM_EnableDeadZone	set the dead zone length and enable/disable Dead Zone function
DrvPWM_Enable	enable PWM timer / capture function

PWM function

function	Description
DrvPWM_SetTimerClk	configure the frequency/pulse/mode/inverter function
DrvPWM_SetTimerIO	enable/disable PWM timer/capture I/O function
DrvPWM_SelectClockSource	select PWM0&PWM1, PWM2&PWM3, PWM4&PWM5 and PWM6&PWM7 engine clock source.
DrvPWM_SelectClearLatchFlagOption	select how to clear Capture rising & falling Latch Indicator.
DrvPWM_GetVersion	Get this module‟s version.

PWM01： Test_PWM

從ADC7輸入一個類比信號，轉換成數位信號，將之顯示在LCD。

1. 觸發ADC轉換
2. 檢查ADF旗標，等待轉換完成
3. 將12bits的數位資料，以10進制顯示在LCD

將ADC的值乘以16倍，作為PWM的CMR0，改變duty cycle

PWMfreq=22.1184M/(1+1)/2/(1+65535)=84Hz

Duty cycle=(CMR+1)/65536

CMR的值可以從0-65535，duty cycle=(CMR+1)/65536

PWM01 : Test_PWM

GPA12,13,14可以作為IO引腳，也可以設定為PWM輸出

作為PWM輸出，使用前必須先設定引腳的功能

SYS->GPAMFP.PWM0_AD13=1;

SYS->GPAMFP.PWM1_AD14=1;

SYS->GPAMFP.PWM2_AD15=1;

Blue LED GPA12 PWM0

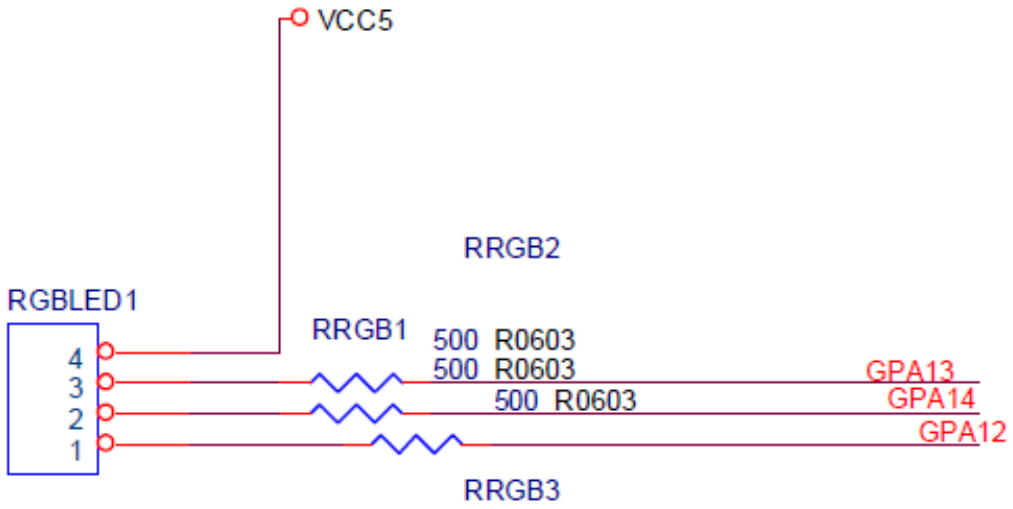
Green LED GPA13 PWM1

Red LED GPA14 PWM2

LED為低電位輸出

電壓越低越亮

電壓越高越暗



PWM01 : Test_PWM 1/x

```
int32_t main (void)
{
        char adc_value[15]="ADC Value:";
        uint16_t ADC_result;
        //Initial 12M and set HCLK=12MHz
        InitHCLK12M();
        // initial LCD pannel function
        Initial_pannel();  //(LCD_Driver.c)
        // clear LCD pannel
        clr_all_pannal(); //(LCD_Driver.c)
        // show title on row 0
        print_lcd(0, adc_value);//(LCD_Driver.c
```

PWM01 : Test_PWM 2/x

```
        //Initial PWM
        InitPWM();
        //Initial ADC
        InitADC();
while(1)
   {
        // A/D Conversion Start
        ADC->ADCR.ADST=1;  //1 = Conversion start
        // wait for ADC complete
        while(ADC->ADSR.ADF==0);  //1=A/D Conversion End Flag
```

PWM01 : Test_PWM 3/x

```
        // clear A/D Conversion End Flag
        ADC->ADSR.ADF=1;  //0=clear A/D Conversion End Flag
        //A/D Conversion Result
        ADC_result=ADC->ADDR[7].RSLT;
        // set CMR0=ADC*16
        PWMA->CMR0=ADC_result<<4;
        // convert 16 bits number to ASCII char
        uint16_ascii(ADC_result,adc_value);
        // display string from 0,10
        show_string(0,10,adc_value);
        DrvSYS_Delay(500000);  //delay 500,000us=500ms
        }
}
```

PWM01 : Test_PWM 4/x

```
void InitADC(void)
{
        // GPIOA Pin[7] Digital Input Path Disable Control
        GPIOA->OFFD|=0x00800000;
        // GPA[7] Pin Function Selection
        SYS->GPAMFP.ADC7_SS21_AD6=1;
        // ADC clock source select,
        //00=12M, 01=PLL, 10=HCLK, 11=22M
        SYSCLK->CLKSEL1.ADC_S = 0;
        // ADC clock divide number from ADC clock source
        SYSCLK->CLKDIV.ADC_N = 0;
```

PWM01 : Test_PWM 5/x

```
// Analog-Digital-Converter (ADC) Clock Enable
SYSCLK->APBCLK.ADC_EN = 1;
// A/D Converter Enable
ADC->ADCR.ADEN = 1;
// Differential Input Mode Enable
ADC->ADCR.DIFFEN = 0;
// A/D Converter Operation Mode, 00=single, 10=single-cycle scan, 11=continuous scan
ADC->ADCR.ADMD   = 0;
// Analog Input Channel Enable
ADC->ADCHER.CHEN = 0x80;
```

PWM01 : Test_PWM 6/x

```
        // A/D Conversion End Flag
        ADC->ADSR.ADF =1;
        // A/D Interrupt Enable
        //ADC->ADCR.ADIE = 1;
        //set interrupt priority
         //NVIC_SetPriority(ADC_IRQn, (1<<__NVIC_PRIO_BITS)
- 2);
        // enable ADC IRQ
        //NVIC_EnableIRQ(ADC_IRQn); //(core_cm0.h)
        // Step 6. Enable WDT module
        // A/D Conversion Start
        //ADC->ADCR.ADST=1;
}
```

PWM01 : Test_PWM 7/x

```
void InitHCLK12M(void)
{
        UNLOCKREG();
        //External 4~24 MHz High Speed Crystal Enable (write-protection bit)
        SYSCLK->PWRCON.XTL12M_EN = 1;
        //HCLK clock source select (write-protection bits)
        //000 = Clock source from external 12 MHz
        SYSCLK->CLKSEL0.HCLK_S = 0;
        LOCKREG();
}
```

PWM01 : Test_PWM 8/x

```
void uint16_ascii(uint16_t value,char text[])
{       uint8_t d1,d0;
        int8_t ia=5;
        for (ia=4; ia>0; ia--)
        {
                d1=value/10;     //value=0-65535
                d0=value-d1*10;
                text[ia]=d0+'0';  //get [4],[3],[2],[1]
                value=d1;         //next value
        }
                text[0]=value+'0'; //get [0]
                text[5]=0;          //string delimiter, /0
}
```

PWM01 : Test_PWM 9/x

```
void show_string(unsigned char x, unsigned char y, char *str)
{
        int i=y;
        do{
                Show_Word(x,i,*str++);  //display a character at (x,i)
                i++;                    // next character
                if(i>15)  break;        // max 16 character
        }  while(*str!='\0');           //
}
```

PWM01 : Test_PWM 10/x

```
void InitPWM(void)
{
        // Step 1. GPIO initial,
        //GPA_MFP[12]=1 set pin for PWM0
        SYS->GPAMFP.PWM0_AD13=1;
        // Step 2. Enable and Select PWM clock source
        //PWM_01 Clock Enable
        SYSCLK->APBCLK.PWM01_EN = 1;//Enable PWM clock
        //PWM0 and PWM1 clock source select
        //00=external, 01=32.768k, 10=HCLK, 11=22.1184M
        SYSCLK->CLKSEL1.PWM01_S = 3;
```

PWM01 : Test_PWM 11/x

```
        //Clock Prescaler 0 (PWM-timer 0 / 1 for group A and PWM-timer 4 / 5 for group B)
        //Prescaler 0~255, Setting 0 to stop output clock
        PWMA->PPR.CP01=1; //Clock input is divided by (CP01 + 1)
        //PWM Timer 0 Clock Source Divider
        //000=2, 001=4, 010=8, 011=16, 100=1
        PWMA->CSR.CSR0=0;
        //PWM-Timer 0 Auto-reload/One-Shot Mode
        //CNR and CMR will be auto-cleared after setting CH0MOD form 0 to 1.
        PWMA->PCR.CH0MOD=1;      //1:Auto-load mode
```

PWM01 : Test_PWM 12/x

```
        //PWM Timer Loaded Value, CNR determines the PWM period.
        //PWM frequency = PWMxy_CLK/[(prescale+1)*(clock divider)*(CNR+1)];
        //Duty ratio = (CMR+1)/(CNR+1).
        PWMA->CNR0=0xFFFF;
        //CMR determin the PWM duty
        PWMA->CMR0=0xFFFF;
        //PWM-Timer 0 Output Inverter Enable
        PWMA->PCR.CH0INV=0;       //Inverter->0:off, 1:on
```

PWM01 : Test_PWM 13/x

```
        //PWM-Timer 0 Enable (PWM timer 0 for group A and PWM timer 4 for group B
        PWMA->PCR.CH0EN=1; //PWM function-> 1:Enable
        // Enable PWM channel 0 output to pin
        PWMA->POE.PWM0=1; //Output to pin-> 1:Enable
}
```

PWM02：Test_PWM_DCServo

// DC Servo Motor = SG-5010

// pin1 : signal, pin2 = Vcc, pin3 = Gnd

伺服系統可以做精確的控制，允許被定位在不同的角度，通常是在0和180度之間。連續旋轉伺服系統可以被設定在不同的速度下旋轉。

驅動方法

PWM產生20ms的脈衝信號給DC伺服馬達的控制信號接腳

Duty cycle由ADC決定PWM高電位的寬度

PWMfreq=12M/(1+1)/2/(1+59999)=50Hz (20ms)

Duty cycle=(CMR+1)/60000

CMR的值可以從0-65535，duty cycle=(CMR+1)/60000

PWM02：Test_PWM_DCServo

GPA12,13,14可以作為IO引腳，也可以設定為PWM輸出

作為PWM輸出，使用前必須先設定引腳的功能

SYS->GPAMFP.PWM0_AD13=1;

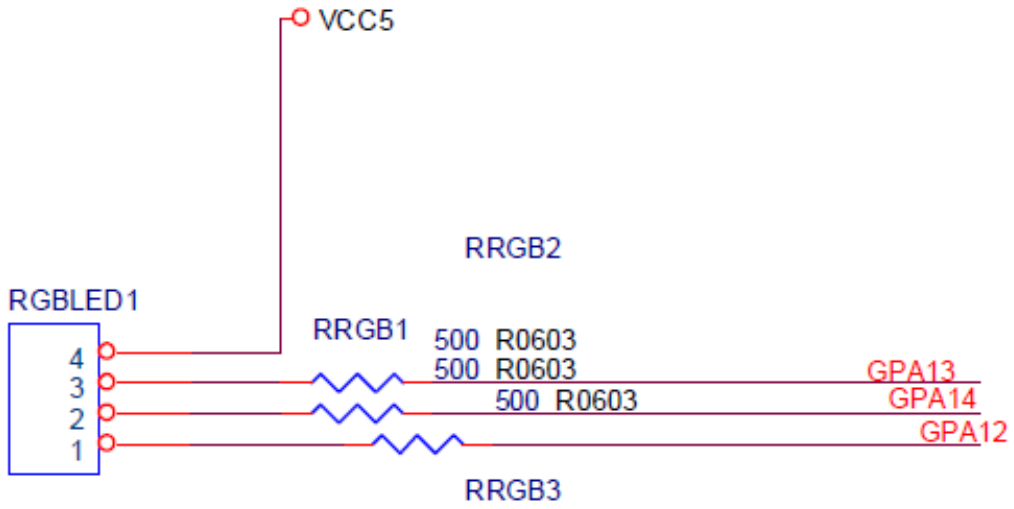
SYS->GPAMFP.PWM1_AD14=1;

SYS->GPAMFP.PWM2_AD15=1;

Blue LED　GPA12 PWM0

Green LED GPA13 PWM1

Red LED　GPA14 PWM2



PWM02 : Test_PWM_DCServo 1/x

```
int32_t main (void)
{
        uint32_t u32Temp;
        char TEXT[15]="              ";
        //Initial 12M and set HCLK=12MHz
        InitHCLK12M();
        // initial LCD pannel function
        Initial_pannel();  //(LCD_Driver.c)
        // clear LCD pannel
        clr_all_pannal(); //(LCD_Driver.c)
        // show title on row 0
        print_lcd(0, "DC Servo Motor ");
```

PWM02 : Test_PWM_DCServo 2/x

```
        print_lcd(1, "PWM period:20mS");
        print_lcd(2, "High Level:    ");
        //Initial PWM
        InitPWM();
        //Initial ADC
        InitADC();
while(1)
   {
        // A/D Conversion Start
        ADC->ADCR.ADST=1;  //1 = Conversion start
        // wait for ADC complete
        while(ADC->ADSR.ADF==0);  //1=A/D Conversion End Flag
```

PWM02 : Test_PWM_DCServo 3/x

```
        // clear A/D Conversion End Flag
        ADC->ADSR.ADF=1;
        // input ADC value
        u32Temp = 2500 + ADC->ADDR[7].RSLT * (1341) / 4096; //
        // PWM0 CMR use ADC value to control high width
        PWMA->CMR0=u32Temp;
        //high level=u32Temp/60000*20ms=u32Temp/3/1000
        sprintf(TEXT,"%d.%dmS",(u32Temp/3)/1000,
(u32Temp/3)%1000);
        print_lcd(3, TEXT);              // print string
        DrvSYS_Delay(20000);             // delay 20ms
}
}
```

PWM02 : Test_PWM_DCServo 4/x

```
void InitADC(void)
{
        // GPIOA Pin[7] Digital Input Path Disable Control
        GPIOA->OFFD|=0x00800000;
        // GPA[7] Pin Function Selection
        SYS->GPAMFP.ADC7_SS21_AD6=1;
        // ADC clock source select,
        //00=12M, 01=PLL, 10=HCLK, 11=22M
        SYSCLK->CLKSEL1.ADC_S = 0;
        // ADC clock divide number from ADC clock source
        SYSCLK->CLKDIV.ADC_N = 0;
```

PWM02 : Test_PWM_DCServo 5/x

```
// Analog-Digital-Converter (ADC) Clock Enable
SYSCLK->APBCLK.ADC_EN = 1;
// A/D Converter Enable
ADC->ADCR.ADEN = 1;
// Differential Input Mode Enable
ADC->ADCR.DIFFEN = 0;
// A/D Converter Operation Mode, 00=single, 10=single-cycle scan, 11=continuous scan
ADC->ADCR.ADMD   = 0;
// Analog Input Channel Enable
ADC->ADCHER.CHEN = 0x80;
```

PWM02 : Test_PWM_DCServo 6/x

```
        // A/D Conversion End Flag
        ADC->ADSR.ADF =1;
        // A/D Interrupt Enable
        //ADC->ADCR.ADIE = 1;
        //set interrupt priority
         //NVIC_SetPriority(ADC_IRQn, (1<<__NVIC_PRIO_BITS)
- 2);
        // enable ADC IRQ
        //NVIC_EnableIRQ(ADC_IRQn); //(core_cm0.h)
        // Step 6. Enable WDT module
        // A/D Conversion Start
        //ADC->ADCR.ADST=1;
}
```

PWM02 : Test_PWM_DCServo 7/x

```
void InitHCLK12M(void)
{
        UNLOCKREG();
        //External 4~24 MHz High Speed Crystal Enable (write-protection bit)
        SYSCLK->PWRCON.XTL12M_EN = 1;
        //HCLK clock source select (write-protection bits)
        //000 = Clock source from external 12 MHz
        SYSCLK->CLKSEL0.HCLK_S = 0;
        LOCKREG();
}
```

PWM02 : Test_PWM_DCServo 8/x

```
void InitPWM(void)
{
        // Step 1. GPIO initial,
        //GPA_MFP[12]=1 set pin for PWM0
        SYS->GPAMFP.PWM0_AD13=1;
        // Step 2. Enable and Select PWM clock source
        //PWM_01 Clock Enable
        SYSCLK->APBCLK.PWM01_EN = 1;//Enable PWM clock
        //PWM0 and PWM1 clock source select
        //00=external, 01=32.768k, 10=HCLK, 11=22.1184M
        SYSCLK->CLKSEL1.PWM01_S = 0;
```

PWM02 : Test_PWM_DCServo 9/x

```
        //Clock Prescaler 0 (PWM-timer 0 / 1 for group A and PWM-timer 4 / 5 for group B)
        //Prescaler 0~255, Setting 0 to stop output clock
        PWMA->PPR.CP01=1; //Clock input is divided by (CP01 + 1)
        //PWM Timer 0 Clock Source Divider
        //000=2, 001=4, 010=8, 011=16, 100=1
        PWMA->CSR.CSR0=0;
        //PWM-Timer 0 Auto-reload/One-Shot Mode
        //CNR and CMR will be auto-cleared after setting CH0MOD form 0 to 1.
        PWMA->PCR.CH0MOD=1;      //1:Auto-load mode
```

PWM02 : Test_PWM_DCServo 10/x

```
        //PWM Timer Loaded Value, CNR determines the PWM period.
        //PWM frequency = PWMxy_CLK/[(prescale+1)*(clock divider)*(CNR+1)];
        //Duty ratio = (CMR+1)/(CNR+1).
        PWMA->CNR0=60000-1;
        //CMR determin the PWM duty
        PWMA->CMR0=0;
        //PWM-Timer 0 Output Inverter Enable
        PWMA->PCR.CH0INV=0;       //Inverter->0:off, 1:on
```

PWM02 : Test_PWM_DCServo 11/x

```
        //PWM-Timer 0 Enable (PWM timer 0 for group A and PWM timer 4 for group B
        PWMA->PCR.CH0EN=1; //PWM function-> 1:Enable
        // Enable PWM channel 0 output to pin
        PWMA->POE.PWM0=1; //Output to pin-> 1:Enable
}
```

PWM03： Test_PWM_Tone

使用PWM4產生不同的頻率

PWM4連接到蜂鳴器，可產生不同的聲音

頻率的計算

PWMfreq=PWMclk/prescaler/divider/(CNR+1)

PWMclk=12M,22.1184M,HCLK

Prescaler=1-255, 0:stop

Divider=1,2,4,8,16

CNR=0-65535

PWM03： Test_PWM_Tone

GPB11可以作為IO引腳，也可以設定為PWM4輸出

作為PWM輸出，使用前必須先設定引腳的功能

SYS->GPAMFP.PWM0_AD13=1;

SYS->GPBMFP.PWM4=1;

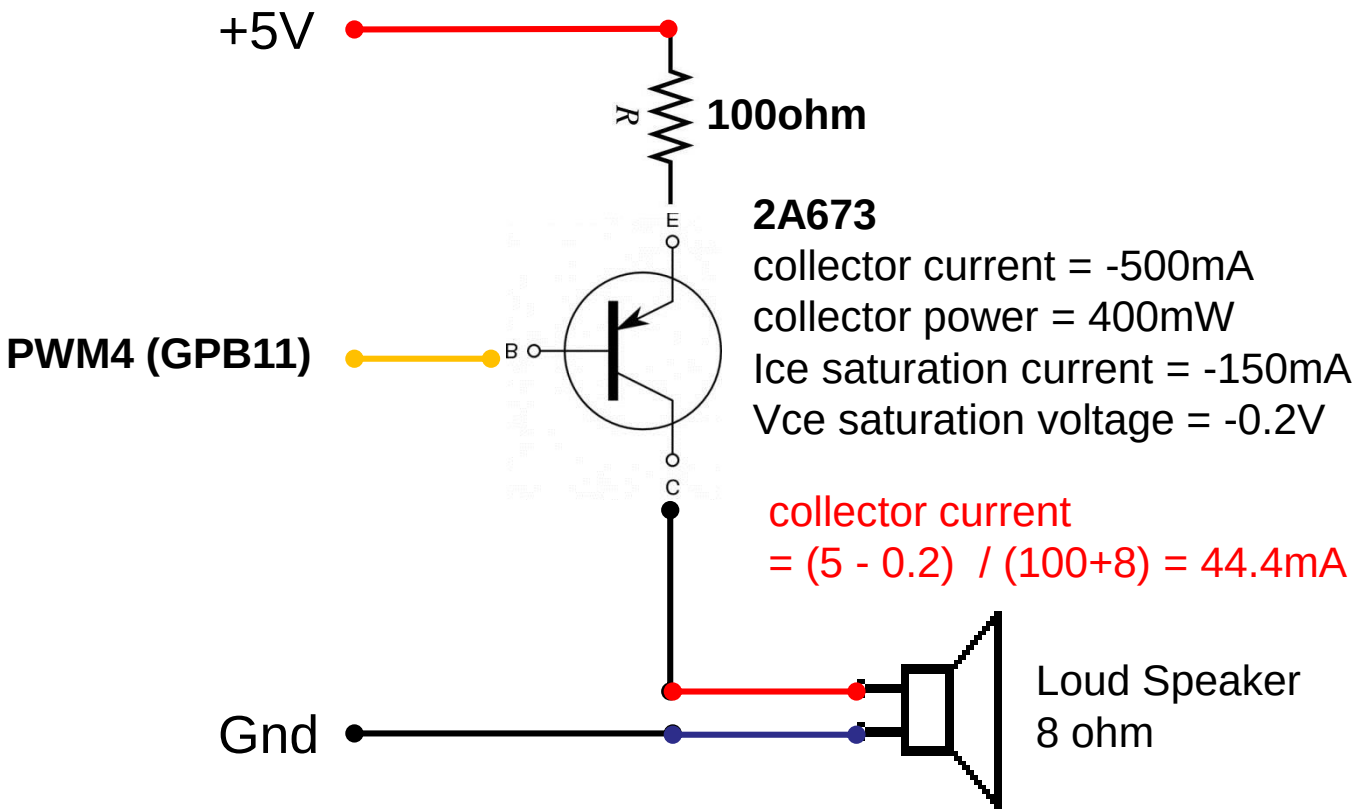
SYS->GPB11=1;

Buzzer　GPB11 PWM4

C調音階-頻率對照表

音階	n	1	2	3	4	5	6	7	8	9	10	11	12
		Do	Do#	Re	Re#	Mi	Fa	Fa#	So	So#	La	La#	Si
低音	頻率	**262**	277	294	311	330	349	370	392	415	440	464	494
	簡譜	1·		2·		3·	4·		5·		6·		7·
中音	頻率	**523**	554	587	622	659	698	740	784	831	880	932	988
	簡譜	1		2		3	4		5		6		7
高音	頻率	**1046**	1109	1175	1245	1318	1397	1480	1568	1661	1760	1865	1976
	簡譜	·1		·2		·3	·4		·5		·6		·7

External Speaker Circuit 外接喇叭電路



PWM03 : Test_PWM_Tone 1/x

```
int32_t main (void)
{
        uint8_t i, duty_cycle;
        //Enable 12Mhz and set HCLK->12Mhz
        char TEXT0[16]="PWM Musical Note";
        char TEXT1[16]="Tone :          ";
        char TEXT2[16]="Freq :       Hz";
        char TEXT3[16]="Duty :       % ";
        duty_cycle = 50; // Duty = 50%
```

PWM03 : Test_PWM_Tone 2/x

```
//Initial 12M and set HCLK=12MHz
InitHCLK12M();
Initial_pannel();          //call initial pannel function
clr_all_pannal();

print_lcd(0, TEXT0);// print TEXT on LCD
print_lcd(1, TEXT1);
print_lcd(2, TEXT2);
print_lcd(3, TEXT3);
```

PWM03 : Test_PWM_Tone 3/x

```
InitPWM0();     // initialize PWM0, output pin = GPA12

while(1)
{
  for (i=49; i<61; i++) {
 // set PWM0 with frequency & duty cycle
  PWM4_Freq(Tone_Freq(i),duty_cycle);
  sprintf(TEXT1+10,"%d",i);
  sprintf(TEXT2+10,"%d",Tone_Freq(i));
  sprintf(TEXT3+10,"%d",duty_cycle);
```

PWM03 : Test_PWM_Tone 4/x

```
            print_lcd(1, TEXT1);
            print_lcd(2, TEXT2);
            print_lcd(3, TEXT3);
            Delay(500000); // delay 500ms
            }
      }
}
```

PWM03 : Test_PWM_Tone 5/x

```
void InitPWM(void)
{
        // Step 1. GPIO initial,
        //GPA_MFP[12]=1 set pin for PWM0
        SYS->GPBMFP.TM3_PWM4=1; //set GPB_MFP11 for PWM4
        SYS->ALTMFP.PB11_PWM4=1; //set PB11_PWM4 for PWM4
        // Step 2. Enable and Select PWM clock source
        //PWM_01 Clock Enable
        SYSCLK->APBCLK.PWM45_EN = 1;//Enable PWM clock
        //PWM0 and PWM1 clock source select
        //00=external, 01=32.768k, 10=HCLK, 11=22.1184M
        SYSCLK->CLKSEL1.PWM45_S = 0;
```

PWM03 : Test_PWM_Tone 6/x

```
        //Clock Prescaler 0 (PWM-timer 0 / 1 for group A and PWM-timer 4 / 5 for group B)
        //Prescaler 0~255, Setting 0 to stop output clock
        PWMB->PPR.CP01=1; //Clock input is divided by (CP01 + 1)
        //PWM Timer 0 Clock Source Divider
        //000=2, 001=4, 010=8, 011=16, 100=1
        PWMB->CSR.CSR0=0;
        //PWM-Timer 0 Auto-reload/One-Shot Mode
        //CNR and CMR will be auto-cleared after setting CH0MOD form 0 to 1.
        PWMB->PCR.CH0MOD=1;      //1:Auto-load mode
```

PWM03 : Test_PWM_Tone 7/x

```
        //PWM Timer Loaded Value, CNR determines the PWM period.
        //PWM frequency = PWMxy_CLK/[(prescale+1)*(clock divider)*(CNR+1)];
        //Duty ratio = (CMR+1)/(CNR+1).
        PWMB->CNR0=0xFFFF;
        //CMR determin the PWM duty
        PWMB->CMR0=0xFFFF;
        //PWM-Timer 0 Output Inverter Enable
        PWMB->PCR.CH0INV=0;       //Inverter->0:off, 1:on
```

PWM03 : Test_PWM_Tone 8/x

```
        //PWM-Timer 0 Enable (PWM timer 0 for group A and PWM timer 4 for group B
        PWMB->PCR.CH0EN=1; //PWM function-> 1:Enable
        // Enable PWM channel 0 output to pin
        PWMB->POE.PWM0=1; //Output to pin-> 1:Enable
}
```

PWM03 : Test_PWM_Tone 9/x

```
void PWM4_Freq(uint32_t PWM_frequency, uint8_t PWM_duty)
{
        uint32_t PWM_Clock;
        uint8_t  PWM_PreScaler;
        uint16_t PWM_ClockDivider;
        uint16_t CNR0, CMR0;

        if (PWM_frequency==0)
                PWMB->POE.PWM0=0; //pin->0:Diasble,
        else
                PWMB->POE.PWM0=1; //pin->1:Enable
```

PWM03 : Test_PWM_Tone 10/x

```
    if (PWM_frequency!=0)
    {
        // PWM setting
        if(SYSCLK->CLKSEL2.PWM45_S == 0)
            PWM_Clock = 12000000; // Clock source = 12 MHz
        if(SYSCLK->CLKSEL2.PWM45_S == 3)
         PWM_Clock = 22118400; // Clock source = 22.1184MHz
        PWM_PreScaler = 5; // clock is divided by (PreScaler +1)
         // 0: 1/2, 1: 1/4, 2: 1/8, 3: 1/16, 4: 1
        PWM_ClockDivider = 2;
        //PWM_FreqOut = PWM_Clock / (PWM_PreScaler + 1) /
PWM_ClockDivider / (PWM_CNR0 + 1);
```

PWM03 : Test_PWM_Tone 11/x

```
        CNR0 = PWM_Clock / PWM_frequency /
(PWM_PreScaler + 1) / PWM_ClockDivider - 1;
        // Duty Cycle = (CMR0+1) / (CNR0+1)
        CMR0 = (CNR0 +1) * PWM_duty /100  - 1;
        //PWM setting
        // 0: 1/2, 1: 1/4, 2: 1/8, 3: 1/16, 4: 1
        PWMB->CSR.CSR0 = 0;
        PWMB->PPR.CP01 = PWM_PreScaler;
        PWMB->CNR0 = CNR0;          // set CNR0
        PWMB->CMR0 = CMR0;          // set CMR0
        }
}
```

PWM03 : Test_PWM_Tone 12/x

```
uint32_t Tone_Freq(uint8_t note)
{
   uint32_t note_frequency;
    switch (note) {
    case  0: note_frequency =   0; break; // null    =       0Hz
    case  1: note_frequency =  16; break; // C0      =   16.35Hz
    case  2: note_frequency =  17; break; // C0#/D0b =   17.32Hz
    case  3: note_frequency =  18; break; // D0      =   18.35Hz
    case  4: note_frequency =  19; break; // D0#/E0b =   19.45Hz
    case  5: note_frequency =  21; break; // E0      =   20.60Hz
    case  6: note_frequency =  22; break; // F0      =   21.83Hz
    case  7: note_frequency =  23; break; // F0#/G0b =   23.12Hz
```

PWM03 : Test_PWM_Tone 13/x

```
case  8: note_frequency =  24; break; // G0      =   24.50Hz
case  9: note_frequency =  26; break; // G0#/A0b =   25.96Hz
case 10: note_frequency =  27; break; // A0      =   27.50Hz
case 11: note_frequency =  29; break; // A0#/B0b =   29.14Hz
case 12: note_frequency =  31; break; // B0      =   30.87Hz
case 13: note_frequency =  33; break; // C1      =   32.70Hz
case 14: note_frequency =  35; break; // C1#/D1b =   34.65Hz
case 15: note_frequency =  37; break; // D1      =   36.71Hz
case 16: note_frequency =  39; break; // D1#/E1b =   38.89Hz
case 17: note_frequency =  41; break; // E1      =   41.20Hz
case 18: note_frequency =  44; break; // F1      =   43.65Hz
```

PWM03 : Test_PWM_Tone 14/x

```
case 19: note_frequency =  46; break; // F1#/G1b =   46.25Hz
case 20: note_frequency =  49; break; // G1      =   49.00Hz
case 21: note_frequency =  52; break; // G1#/A1b =   51.91Hz
case 22: note_frequency =  55; break; // A1      =   55.00Hz
case 23: note_frequency =  58; break; // A1#/B1b =   58.27Hz
case 24: note_frequency =  62; break; // B1      =   61.74Hz
case 25: note_frequency =  65; break; // C2      =   65.41Hz
case 26: note_frequency =  69; break; // C2#/D2b =   69.30Hz
case 27: note_frequency =  73; break; // D2      =   73.42Hz
case 28: note_frequency =  78; break; // D2#/E2b =   77.78Hz
case 29: note_frequency =  82; break; // E2      =   82.41Hz
```

PWM03 : Test_PWM_Tone 15/x

```
case 30: note_frequency =  87; break; // F2      =   87.31Hz
case 31: note_frequency =  92; break; // F2#/G2b =   92.50Hz
case 32: note_frequency =  98; break; // G2      =   98.00Hz
case 33: note_frequency = 104; break; // G2#/A2b =  103.83Hz
case 34: note_frequency = 110; break; // A2      =  110.00Hz
case 35: note_frequency = 117; break; // A2#/B2b =  116.54Hz
case 36: note_frequency = 123; break; // B2      =  123.47Hz
case 37: note_frequency = 131; break; // C3      =  130.81Hz
case 38: note_frequency = 139; break; // C3#/D3b =  138.59Hz
case 39: note_frequency = 147; break; // D3      =  146.83Hz
case 40: note_frequency = 156; break; // D3#/E3b =  155.56Hz
```

PWM03 : Test_PWM_Tone 16/x

```
case 41: note_frequency = 165; break; // E3      =  164.81Hz
case 42: note_frequency = 175; break; // F3      =  174.61Hz
case 43: note_frequency = 185; break; // F3#/G3b =  185.00Hz
case 44: note_frequency = 196; break; // G3      =  196.00Hz
case 45: note_frequency = 208; break; // G3#/A3b =  207.65Hz
case 46: note_frequency = 220; break; // A3      =  220.00Hz
case 47: note_frequency = 233; break; // A3#/B3b =  233.08Hz
case 48: note_frequency = 247; break; // B3      =  246.94Hz
case 49: note_frequency = 262; break; // C4      =  261.63Hz
case 50: note_frequency = 277; break; // C4#/D4b =  277.18Hz
```

PWM03 : Test_PWM_Tone 17/x

```
case 51: note_frequency = 294; break; // D4      =  293.66Hz
 case 52: note_frequency = 311; break; // D4#/E4b =  311.13Hz
 case 53: note_frequency = 330; break; // E4      =  329.63Hz
 case 54: note_frequency = 349; break; // F4      =  349.23Hz
 case 55: note_frequency = 370; break; // F4#/G4b =  369.99Hz
 case 56: note_frequency = 392; break; // G4      =  392.00Hz
 case 57: note_frequency = 415; break; // G4#/A4b =  415.30Hz
 case 58: note_frequency = 440; break; // A4      =  440.00Hz
 case 59: note_frequency = 466; break; // A4#/B4b =  466.16Hz
 case 60: note_frequency = 494; break; // B4      =  493.88Hz
```

PWM03 : Test_PWM_Tone 18/x

```
case 61: note_frequency = 523; break; // C5      =  523.25Hz
case 62: note_frequency = 554; break; // C5#/D5b =  554.37Hz
case 63: note_frequency = 587; break; // D5      =  587.33Hz
case 64: note_frequency = 622; break; // D5#/E5b =  622.25Hz
case 65: note_frequency = 659; break; // E5      =  659.26Hz
case 66: note_frequency = 698; break; // F5      =  698.46Hz
case 67: note_frequency = 740; break; // F5#/G5b =  739.99Hz
case 68: note_frequency = 784; break; // G5      =  783.99Hz
case 69: note_frequency = 831; break; // G5#/A5b =  830.61Hz
case 70: note_frequency = 880; break; // A5      =  880.00Hz
case 71: note_frequency = 932; break; // A5#/B5b =  932.33Hz
```

PWM03 : Test_PWM_Tone 19/x

```
case 72: note_frequency = 988; break; // B5      =  987.77Hz
 case 73: note_frequency =1047; break; // C6      = 1046.50Hz
 case 74: note_frequency =1109; break; // C6#/D6b = 1108.73Hz
 case 75: note_frequency =1175; break; // D6      = 1174.66Hz
 case 76: note_frequency =1245; break; // D6#/E6b = 1244.51Hz
 case 77: note_frequency =1319; break; // E6      = 1318.51Hz
 case 78: note_frequency =1397; break; // F6      = 1396.91Hz
 case 79: note_frequency =1480; break; // F6#/G6b = 1479.98Hz
 case 80: note_frequency =1568; break; // G6      = 1567.98Hz
 case 81: note_frequency =1661; break; // G6#/A6b = 1661.22Hz
 case 82: note_frequency =1760; break; // A6      = 1760.00Hz
```

PWM03 : Test_PWM_Tone 20/x

```
case 83: note_frequency =1865; break; // A6#/B6b = 1864.66Hz
case 84: note_frequency =1976; break; // B6      = 1975.53Hz
case 85: note_frequency =2093; break; // C7      = 2093.00Hz
case 86: note_frequency =2217; break; // C7#/D7b = 2217.46Hz
case 87: note_frequency =2349; break; // D7      = 2349.32Hz
case 88: note_frequency =2489; break; // D7#/E7b = 2489.02Hz
case 89: note_frequency =2637; break; // E7      = 2637.02Hz
case 90: note_frequency =2794; break; // F7      = 2793.83Hz
case 91: note_frequency =2960; break; // F7#/G7b = 2959.96Hz
case 92: note_frequency =3136; break; // G7      = 3135.96Hz
case 93: note_frequency =3322; break; // G7#/A7b = 3322.44Hz
```

PWM03 : Test_PWM_Tone 21/x

```
case 94: note_frequency =3520; break; // A7      = 3520.00Hz
 case 95: note_frequency =3729; break; // A7#/B7b = 3729.31Hz
 case 96: note_frequency =3951; break; // B7      = 3951.07Hz
 case 97: note_frequency =4186; break; // C8      = 4186.01Hz
 case 98: note_frequency =4435; break; // C8#/D8b = 4434.92Hz
 case 99: note_frequency =4699; break; // D8      = 4698.64Hz
 case 100:note_frequency =4978; break; // D8#/E8b =
4978.03Hz
 default: note_frequency =   0; break; // null    =       0Hz
 }
 return note_frequency;
}
```

PWM04： Test_PWM_Tone_Keypad

根據3x3鍵盤的輸入，使用PWM4產生不同的頻率

PWM4連接到蜂鳴器，可產生不同的聲音

頻率的計算

PWMfreq=PWMclk/prescaler/divider/(CNR+1)

PWMclk=12M,22.1184M,HCLK

Prescaler=1-255, 0:stop

Divider=1,2,4,8,16

CNR=0-65535

PWM04： Test_PWM_Tone

GPB11可以作為IO引腳，也可以設定為PWM4輸出

作為PWM輸出，使用前必須先設定引腳的功能

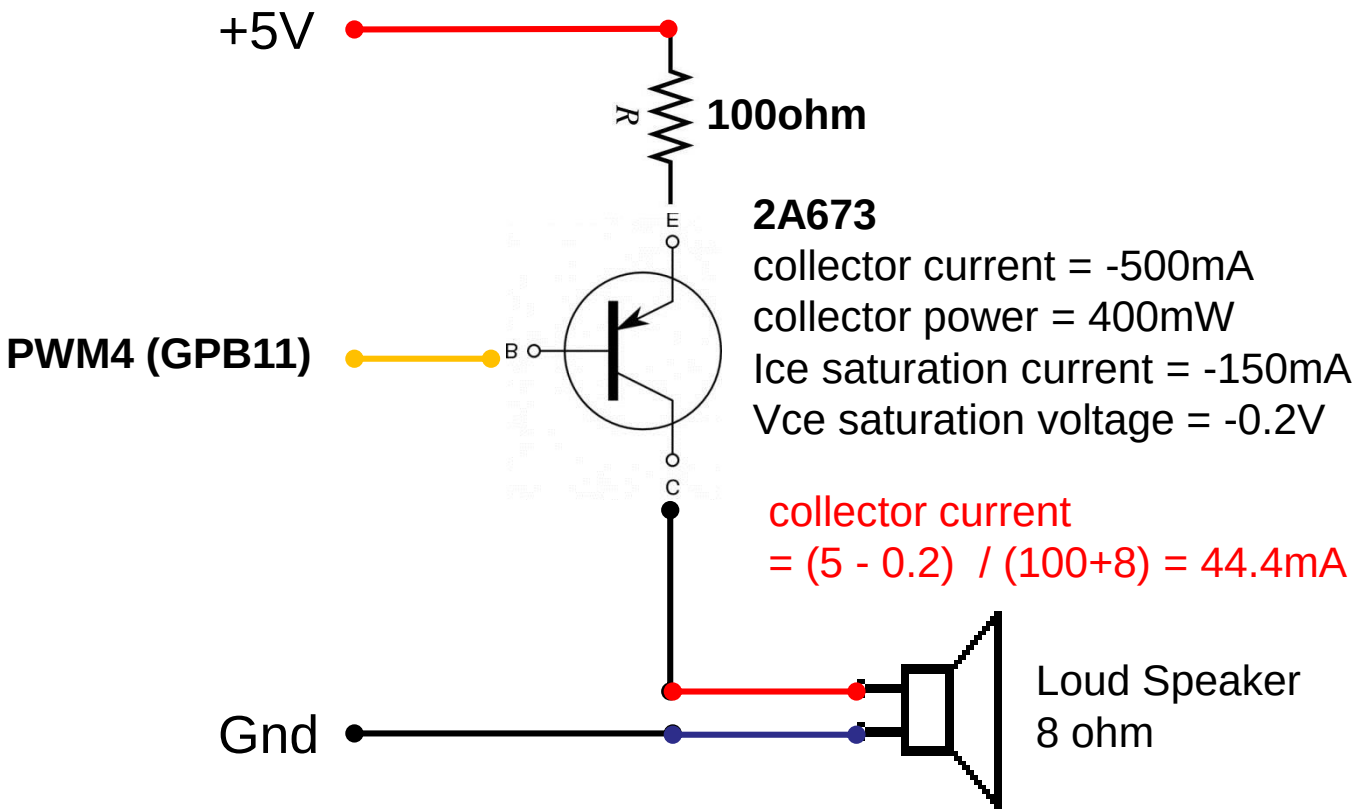
SYS->GPAMFP.PWM0_AD13=1;

SYS->GPBMFP.PWM4=1;

SYS->GPB11=1;

Buzzer　GPB11 PWM4

External Speaker Circuit 外接喇叭電路



PWM04 : Test_PWM_Tone 1/x

```
int32_t main (void)
{
        uint8_t i, duty_cycle;
        uint8_t number=0;
        uint16_t tone_freq;
        char TEXT0[16]="PWM Musical Note";
        char TEXT1[16]="Tone :          ";
        char TEXT2[16]="Freq :       Hz";
        char TEXT3[16]="Duty :       % ";
        duty_cycle = 50; // Duty = 50%
```

PWM04 : Test_PWM_Tone 2/x

```
//Initial 12M and set HCLK=12MHz
InitHCLK12M();
Initial_pannel();          //call initial pannel function
clr_all_pannal();

print_lcd(0, TEXT0);// print TEXT on LCD
print_lcd(1, TEXT1);
print_lcd(2, TEXT2);
print_lcd(3, TEXT3);
```

PWM04 : Test_PWM_Tone 3/x

```
// Initial Timer,frequency=250,4ms
InitTimer0(250);
InitPWM0();     // initialize PWM0, output pin = GPA12

while(1)
{
    if(keybufferptr)
    {
        keybufferptr=0;
        number = keybuffer[0];
    }
```

PWM04 : Test_PWM_Tone 4/x

```
switch(number)
    {
        case 1: i = 5; tone_freq = 21; break;
        case 2: i = 60; tone_freq = 494; break;
        case 3: i = 72; tone_freq = 988; break;
        case 4: i = 79; tone_freq = 1480; break;
        case 5: i = 84; tone_freq = 1976; break;
        case 6: i = 88; tone_freq = 2489; break;
        case 7: i = 91; tone_freq = 2960; break;
        case 8: i = 94; tone_freq = 3520; break;
        case 9: i = 100; tone_freq = 4978; break;
        default:i =  0; tone_freq =   0; break;
    }
```

PWM04 : Test_PWM_Tone 5/x

```
        // set PWM0 with frequency & duty cycle
          PWM4_Freq(Tone_Freq(i),duty_cycle);
          sprintf(TEXT1+10,"%d",number);
          sprintf(TEXT2+10,"%d",tone_freq);
          sprintf(TEXT3+10,"%d",duty_cycle);
          print_lcd(1, TEXT1);
          print_lcd(2, TEXT2);
          print_lcd(3, TEXT3);

        }
}
```

PWM04 : Test_PWM_Tone 6/x

```
void InitPWM(void)
{
        // Step 1. GPIO initial,
        //GPA_MFP[12]=1 set pin for PWM0
        SYS->GPBMFP.TM3_PWM4=1; //set GPB_MFP11 for PWM4
        SYS->ALTMFP.PB11_PWM4=1; //set PB11_PWM4 for PWM4
        // Step 2. Enable and Select PWM clock source
        //PWM_01 Clock Enable
        SYSCLK->APBCLK.PWM45_EN = 1;//Enable PWM clock
        //PWM0 and PWM1 clock source select
        //00=external, 01=32.768k, 10=HCLK, 11=22.1184M
        SYSCLK->CLKSEL1.PWM45_S = 0;
```

PWM04 : Test_PWM_Tone 7/x

```
        //Clock Prescaler 0 (PWM-timer 0 / 1 for group A and PWM-timer 4 / 5 for group B)
        //Prescaler 0~255, Setting 0 to stop output clock
        PWMB->PPR.CP01=1; //Clock input is divided by (CP01 + 1)
        //PWM Timer 0 Clock Source Divider
        //000=2, 001=4, 010=8, 011=16, 100=1
        PWMB->CSR.CSR0=0;
        //PWM-Timer 0 Auto-reload/One-Shot Mode
        //CNR and CMR will be auto-cleared after setting CH0MOD form 0 to 1.
        PWMB->PCR.CH0MOD=1;      //1:Auto-load mode
```

PWM04 : Test_PWM_Tone 8/x

```
        //PWM Timer Loaded Value, CNR determines the PWM period.
        //PWM frequency = PWMxy_CLK/[(prescale+1)*(clock divider)*(CNR+1)];
        //Duty ratio = (CMR+1)/(CNR+1).
        PWMB->CNR0=0xFFFF;
        //CMR determin the PWM duty
        PWMB->CMR0=0xFFFF;
        //PWM-Timer 0 Output Inverter Enable
        PWMB->PCR.CH0INV=0;       //Inverter->0:off, 1:on
```

PWM04 : Test_PWM_Tone 9/x

```
        //PWM-Timer 0 Enable (PWM timer 0 for group A and PWM timer 4 for group B
        PWMB->PCR.CH0EN=1; //PWM function-> 1:Enable
        // Enable PWM channel 0 output to pin
        PWMB->POE.PWM0=1; //Output to pin-> 1:Enable
}
```

PWM03 : Test_PWM_Tone 9/x

```
void PWM4_Freq(uint32_t PWM_frequency, uint8_t PWM_duty)
{
        uint32_t PWM_Clock;
        uint8_t  PWM_PreScaler;
        uint16_t PWM_ClockDivider;
        uint16_t CNR0, CMR0;

        if (PWM_frequency==0)
                PWMB->POE.PWM0=0; //pin->0:Diasble,
        else
                PWMB->POE.PWM0=1; //pin->1:Enable
```

PWM03 : Test_PWM_Tone 10/x

```
if (PWM_frequency!=0)
{
    // PWM setting
    if(       SYSCLK->CLKSEL2.PWM45_S == 0)
      PWM_Clock = 12000000; // Clock source = 12 MHz
    if(       SYSCLK->CLKSEL2.PWM45_S == 3)
      PWM_Clock = 22118400; // Clock source = 22.1184MHz
    PWM_PreScaler = 5; // clock is divided by (PreScaler +1)
     // 0: 1/2, 1: 1/4, 2: 1/8, 3: 1/16, 4: 1
    PWM_ClockDivider = 2;
    //PWM_FreqOut = PWM_Clock / (PWM_PreScaler + 1) /
PWM_ClockDivider / (PWM_CNR0 + 1);
```

PWM03 : Test_PWM_Tone 12/x

```
        CNR0 = PWM_Clock / PWM_frequency /
(PWM_PreScaler + 1) / PWM_ClockDivider - 1;
        // Duty Cycle = (CMR0+1) / (CNR0+1)
        CMR0 = (CNR0 +1) * PWM_duty /100  - 1;
        //PWM setting
        // 0: 1/2, 1: 1/4, 2: 1/8, 3: 1/16, 4: 1
        PWMB->CSR.CSR0 = 0;
        PWMB->PPR.CP01 = PWM_PreScaler;
        PWMB->CNR0 = CNR0;            // set CNR0
        PWMB->CMR0 = CMR0;           // set CMR0
    }
}
```

General Disclaimer

The Lecture is strictly used for educational purpose.

MAKES NO GUARANTEE OF VALIDITY

- **The lecture cannot guarantee the validity of the information found here.** The lecture may recently have been changed, vandalized or altered by someone whose opinion does not correspond with the state of knowledge in the relevant fields. Note that most other encyclopedias and reference works also have similar disclaimers.

No formal peer review

- The lecture is not uniformly peer reviewed; while readers may correct errors or engage in casual peer review, they have no legal duty to do so and thus all information read here is without any implied warranty of fitness for any purpose or use whatsoever. Even articles that have been vetted by informal peer review or *featured article* processes may later have been edited inappropriately, just before you view them.

No contract; limited license

- Please make sure that you understand that the information provided here is being provided freely, and that no kind of agreement or contract is created between you and the owners or users of this site, the owners of the servers upon which it is housed, the individual Wikipedia contributors, any project administrators, sysops or anyone else who is in *any way connected* with this project or sister projects subject to your claims against them directly. You are being granted a limited license to copy anything from this site; it does not create or imply any contractual or extracontractual liability on the part of Wikipedia or any of its agents, members, organizers or other users.
- There is **no agreement or understanding between you and the content provider** regarding your use or modification of this information beyond the Creative Commons Attribution-Sharealike 3.0 Unported License (CC-BY-SA) and the GNU Free Documentation License (GFDL);

General Disclaimer

Trademarks

- Any of the trademarks, service marks, collective marks, design rights or similar rights that are mentioned, used or cited in the lectures are the property of their respective owners. Their use here does not imply that you may use them for any purpose other than for the same or a similar informational use as contemplated by the original authors under the CC-BY-SA and GFDL licensing schemes. Unless otherwise stated , we are neither endorsed by nor affiliated with any of the holders of any such rights and as such we cannot grant any rights to use any otherwise protected materials. Your use of any such or similar incorporeal property is at your own risk.

Personality rights

- The lecture may portray an identifiable person who is alive or deceased recently. The use of images of living or recently deceased individuals is, in some jurisdictions, restricted by laws pertaining to personality rights, independent from their copyright status. Before using these types of content, please ensure that you have the right to use it under the laws which apply in the circumstances of your intended use. *You are solely responsible for ensuring that you do not infringe someone else's personality rights.*