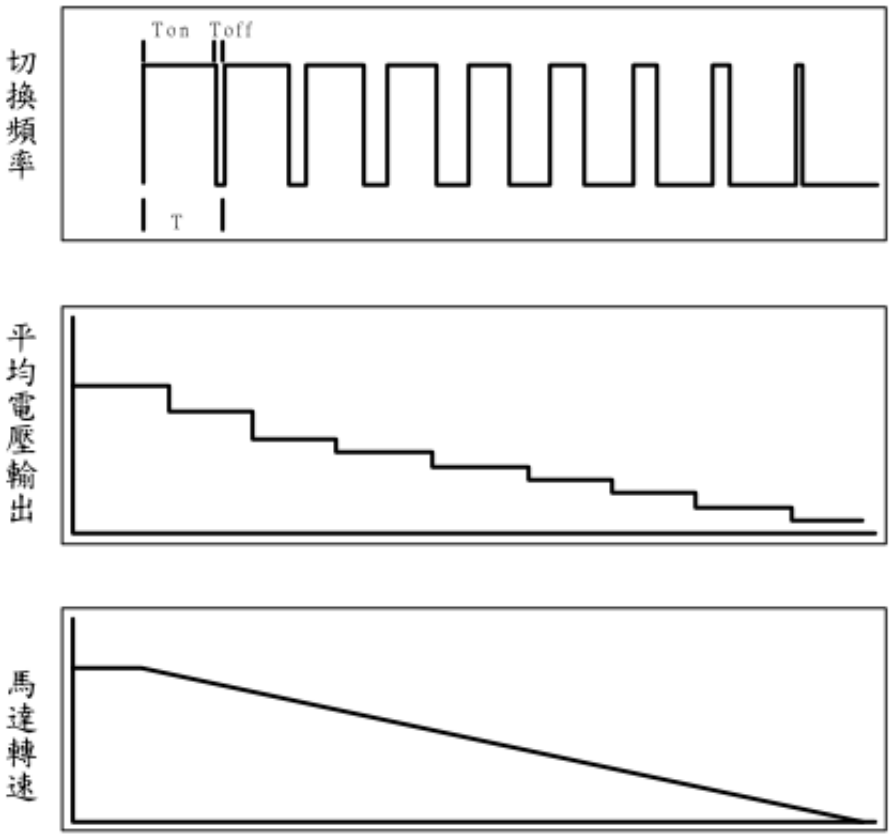
Pulse-Width Modulation (PWM)

Introduction

- Pulse-Width Modulation, 又稱pulse-duration modulation(PDM),其利用在頻率不變的狀態下, 改變工作週期大小, 使整體平均電壓值上升或下降, 藉此間歇性電壓及功率切換以節省能源及控制等效果.



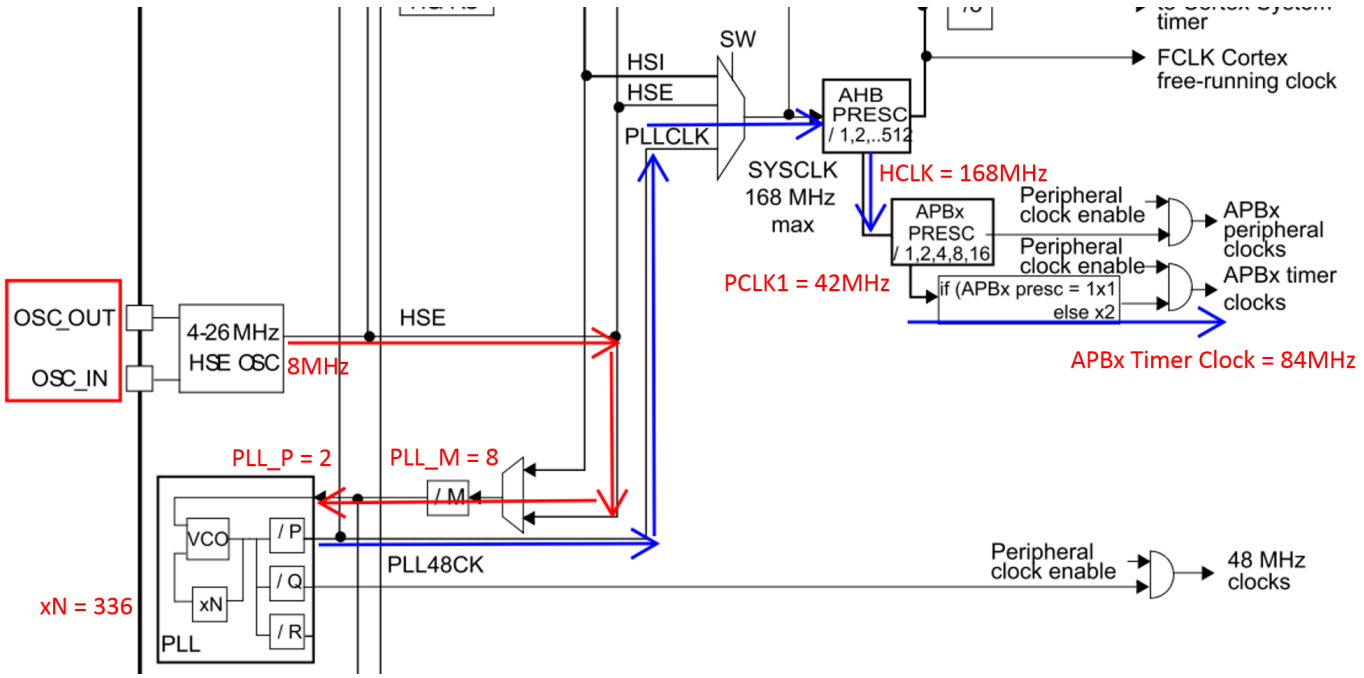
- **PWM會較省電的原因**
  因為一般類比電壓要降低電壓輸出需靠增加電阻，源頭輸出電壓"持續"都為同一電壓，不過利用電阻改變最後輸出電壓，而PWM他靠的是一段時間內輸出的頻率來模擬類比電壓，"不需要持續的輸出"，故不會將電浪費在電阻上，即可達到省電效果。

- STM32內部要產生PWM訊號時，需要透過Timer來實現。

Clock control

- 在學習Timer前, 先瞭解系統clock是如何產生的, 以及給Timer的clock值.
- A part of clock tree

1



- 我們使用Crystal resonator作為外部震盪, 其頻率為8MHz（由STM32f4 discovery user manual 可知）
2

- APBx timer clocks 計算
  - 參閱system_stm32f4xx.c
    - line 341 RCC->CR |= ((uint32_t)RCC_CR_HSEON);
      - enable HSE(High Speed External clock)
    - line 374 ~ 379 配置與啟動PLL
    - line 146 ~ 151, 可知
      - PLL_VCO = (HSE_VALUE or HSI_VALUE / PLL_M) * PLL_N
      - SYSCLK = PLL_VCO / PLL_P
      - 並定義 PLL_M = 8, PLL_N = 336, PLL_P = 2 而外部晶體頻率為8MHz
      - -> PLL_VCO = (8MHz / 8) * 336 = 336MHz
      - -> SYSCLK = 336MHz / 2 = 168MHz
    - line 365 RCC->CFGR |= RCC_CFGR_HPRE_DIV1;
      - 此行設定HCLK(High speed external clock signal) = SYSCLK / 1 = 168MHz
    - line 372 RCC->CFGR |= RCC_CFGR_PPRE1_DIV4;
      - 此行設定APB1 peripheral clocks為 HCLK / 4 = 42MHz
    - 而由上圖可知
      - APB1 timer clocks = APB1 peripheral clocks * 2 =**84MHz**
      - 該clock亦為後面Timer章節提及的 **CK_INT** 之值

Timers

- Timer 和 RTC的差別
  - Timer可被用於多種用途, 其中包含量測輸入訊號之pulse寬度, 或產生輸出波形。
  - Real-Time Clock(RTC)是負責記錄時間的專用積體電路，出現在需要長期使用時鐘的電子設備中。[3]

Basic timers 基本定時器(TIM6 和 TIM7)

[4]

- 具有 16-bit auto-reload upcounter driven by a programmable prescaler
- 用途: Synchronization circuit to trigger the DAC(內部連接至DAC)

Advanced-control timers 高級控制定時器(TIM1 和 TIM8)

[5]

- 具有 16-bit up, down, up/down auto-reload counter driven by a programmable prescaler which allowing dividing the counter clock frequency either by any factor between 1 and 65536(2^16).
- 用途: 最多有**四個**獨立通道可用於input capture, output compare, PWM generation(Edge and Center-aligned Mode) and one-pulse mode output.
- 與TIM2&TIM5之差異: 可以 Break input to put the timer's output signals in reset state or in a known state

General-purpose timers 通用定時器(TIM9 to TIM14)

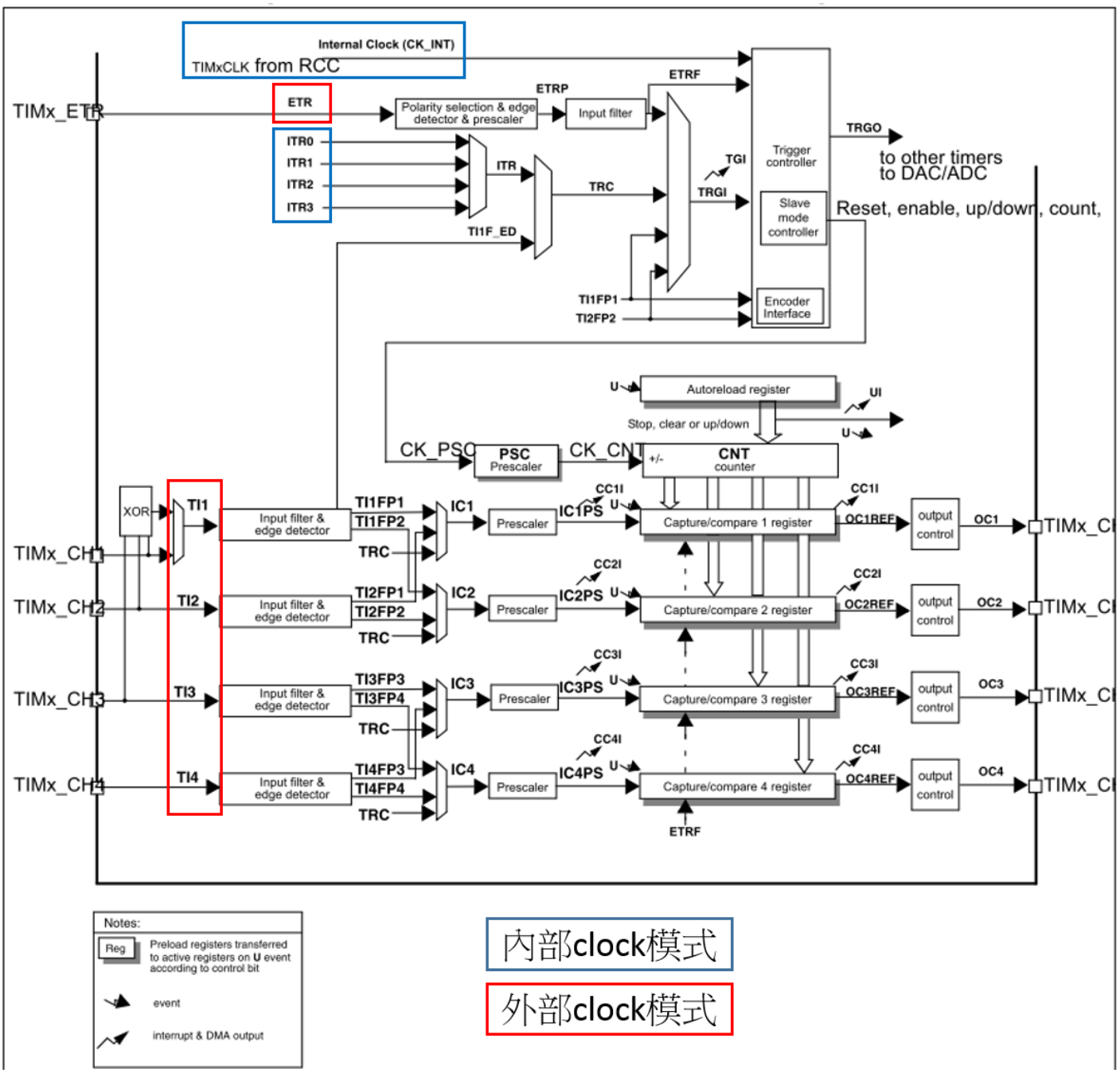
[6]

- 具有16-bit auto-reload up counter.
- 以及16-bit programmable prescaler used to divide the counter clock frequency by any factor between 1 and 65536.
- 用途: 最多有**兩個**獨立通道可用於 input capture, output compare, PWM generation(Edge and Center-aligned Mode) and one-pulse mode output.

General-purpose timers 通用定時器(TIM2 to TIM5)

[7]

- 具有16-bit (TIM3 & TIM4) or 32-bit (TIM2 & TIM5) up, down, up/dowm auto-reload counter.
- 以及16-bit programmable prescaler used to divide the counter clock frequency by any factor between 1 and 65536.
- 用途: 最多有**四個**獨立通道可用於 input capture, output compare, PWM generation(Edge and Center-aligned Mode) and one-pulse mode output.
- Block diagram



- clock來源 :
  - Internal clock(CK_INT)
  - External clock mode1: external input pin(TIx)
  - External clock mode2: external trigger input(ETR) available on TIM2, TIM3 and TIM4 only
  - Internal trigger inputs(ITRx): 使用另一個timer作為prescaler來產生除頻後之clock

Counter Modes

8

Registers

- TIMx_CNT: Counter Register, 計數器目前數到的值

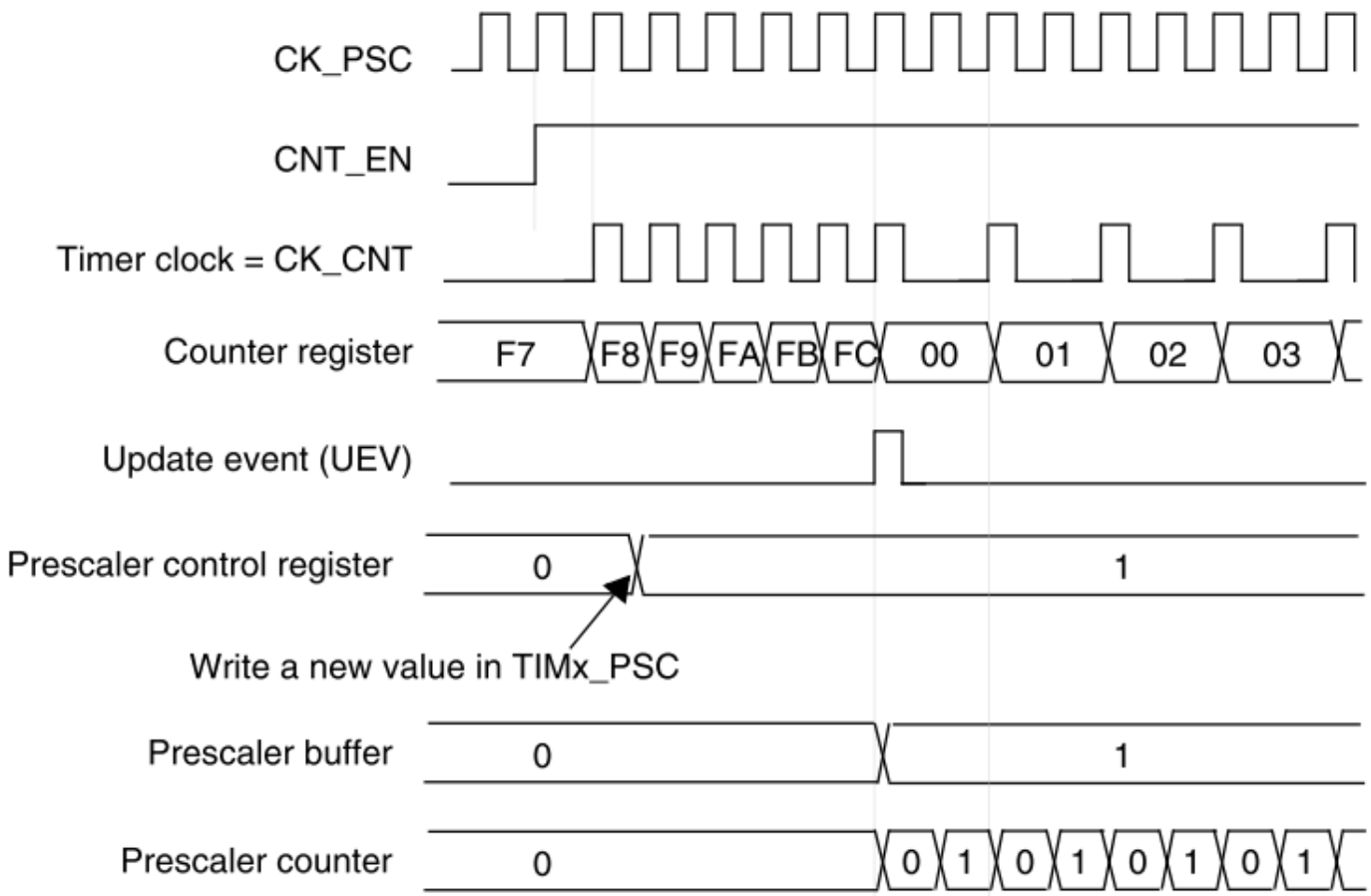
- TIMx_PSC: Prescaler Register (software R/W), 除頻數
- TIMx_ARR: Auto-Reload Register (software R/W), 存放計數起始(下數)或最大(上數)值
- UEV: 當計數器overflow or underflow, 將會產生update event(UEV), 此時會把 Auto-Reload Register (software write)內容放入 Auto-Reload shadow register (hardware write)
  - auto-reload preload enable bit(ARPE) in the TIMx_CR1 register之值設定是否直接放入shadow register,
    - ARPE = 0 直接把值轉入shadow register,
    - ARPE = 1 UEV產生時才將值轉入shadow register

Prescaler Description

9

```
TIM_TimeBaseInitStruct.TIM_Prescaler = PRES_VALUE - 1;
```

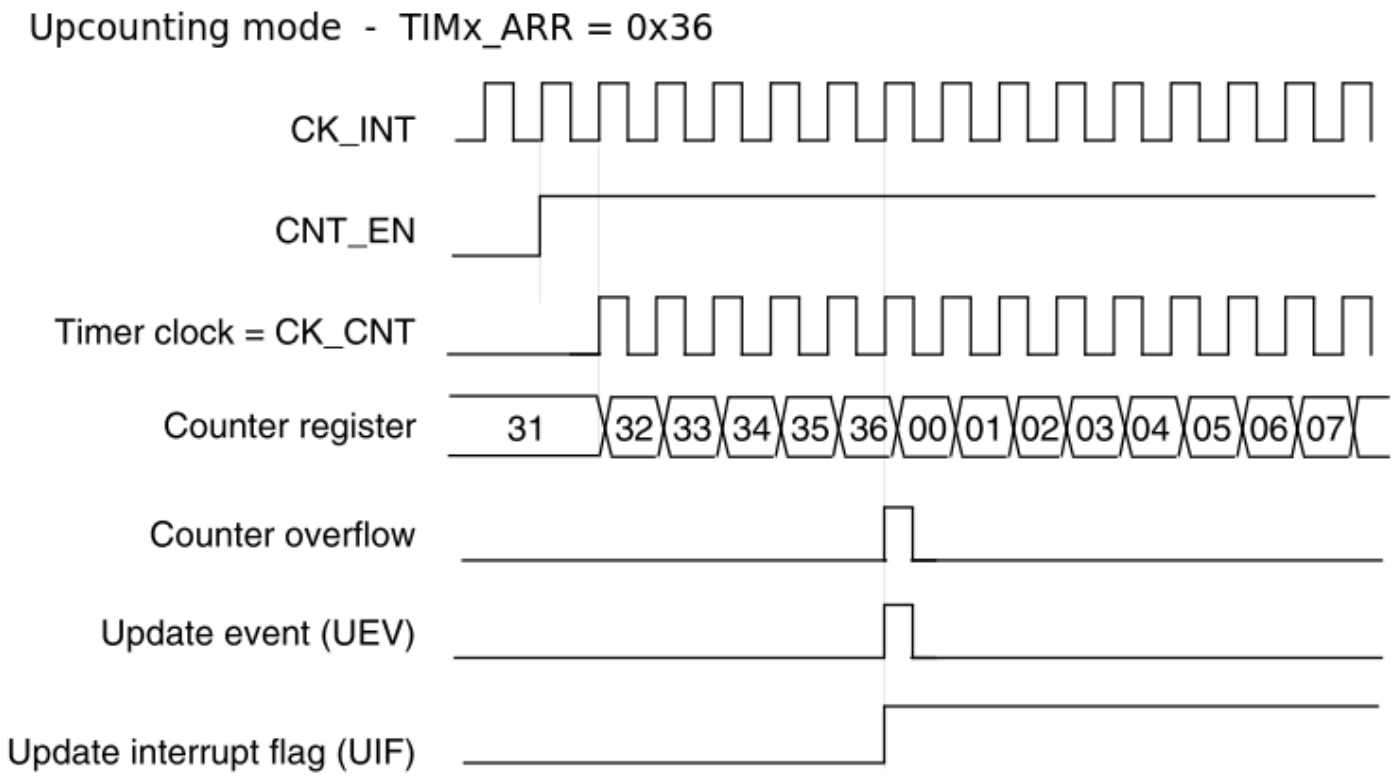
- 可除頻範圍1 ~ 65536(Set it in TIMx_PSC register)
- example: prescaler control register設為1,但須等 update event產生才會載入至prescaler buffer, 因此下一個計數週期才會依據該值來除頻.



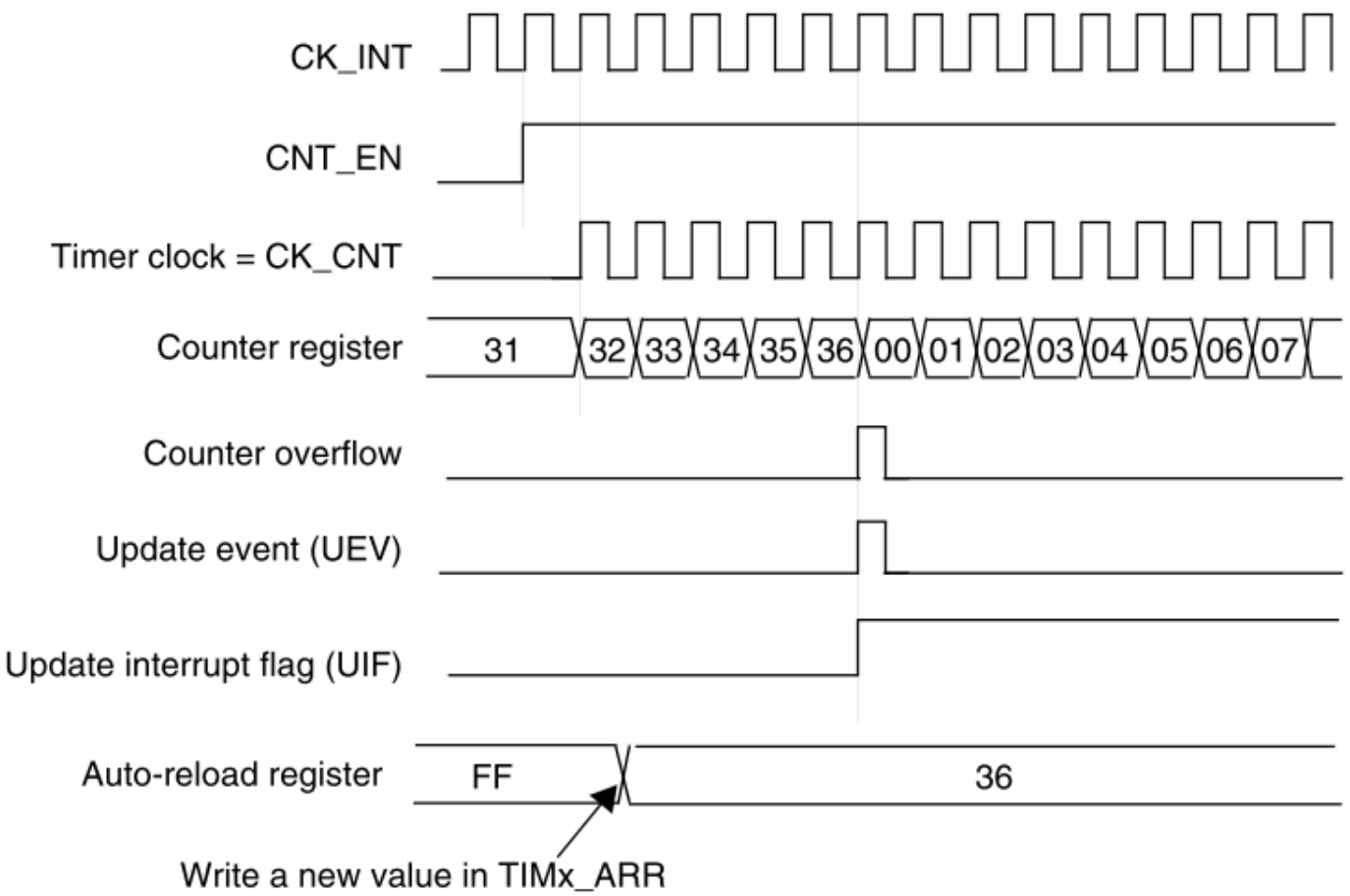
Upcounting mode

```
TIM_TimeBaseInitStruct.TIM_CounterMode = TIM_CounterMode_Up;
```

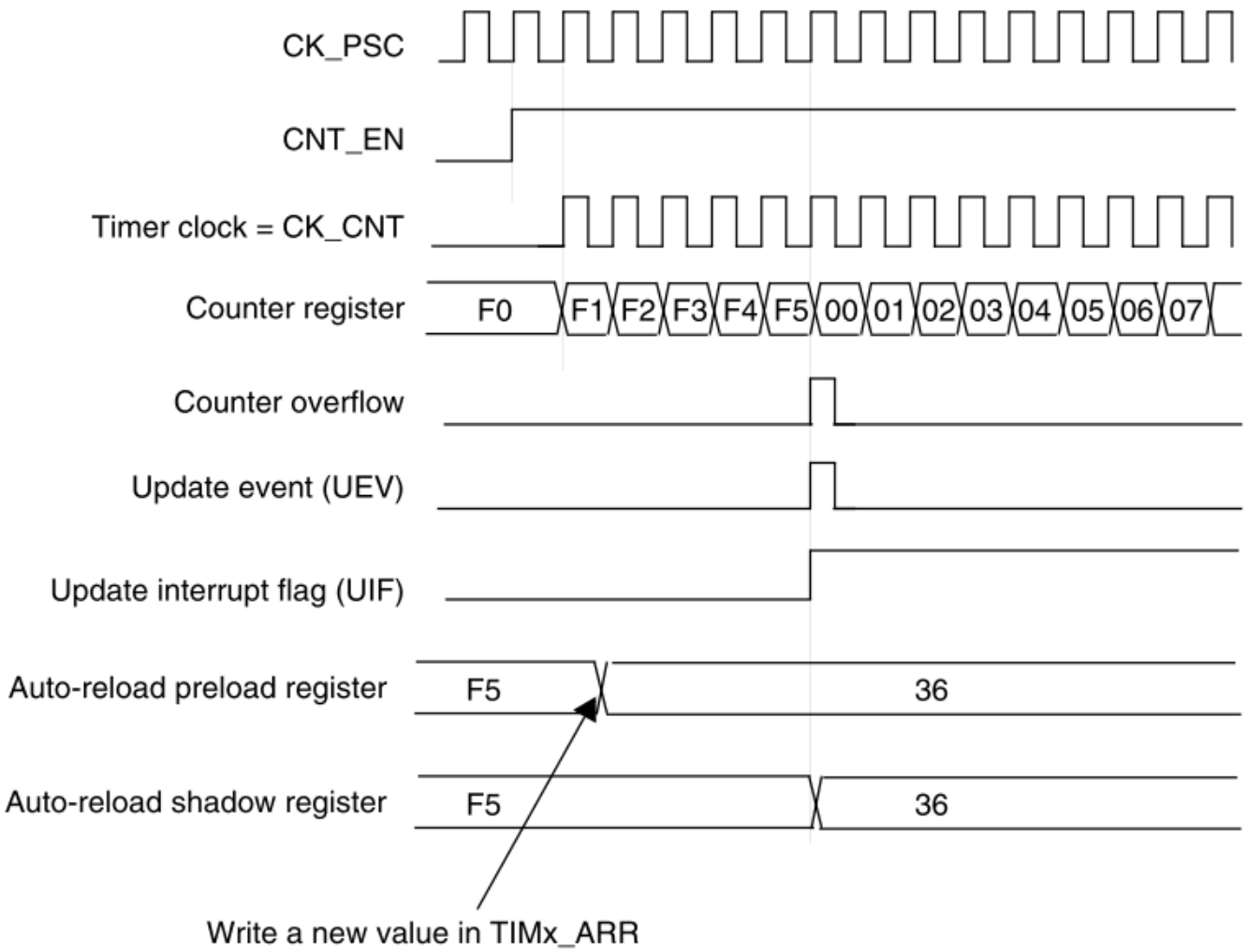
- 由0數至TIMx_ARR register之值, 然後再回到0並產生overflow event(update event), 此時auto-reload shadow register將會更新為TIMx_ARR之值.
  - 可藉由設定UDIS bit = 1來關閉UEV, 可用於避免正在更改preload register內容時, 剛好發生UEV而將舊的值轉移至shadow register中
- Example: TIMx_ARR = 0x36, prescaler buffer = 1



- Example: TIMx_ARR = 0x36, ARPE = 0



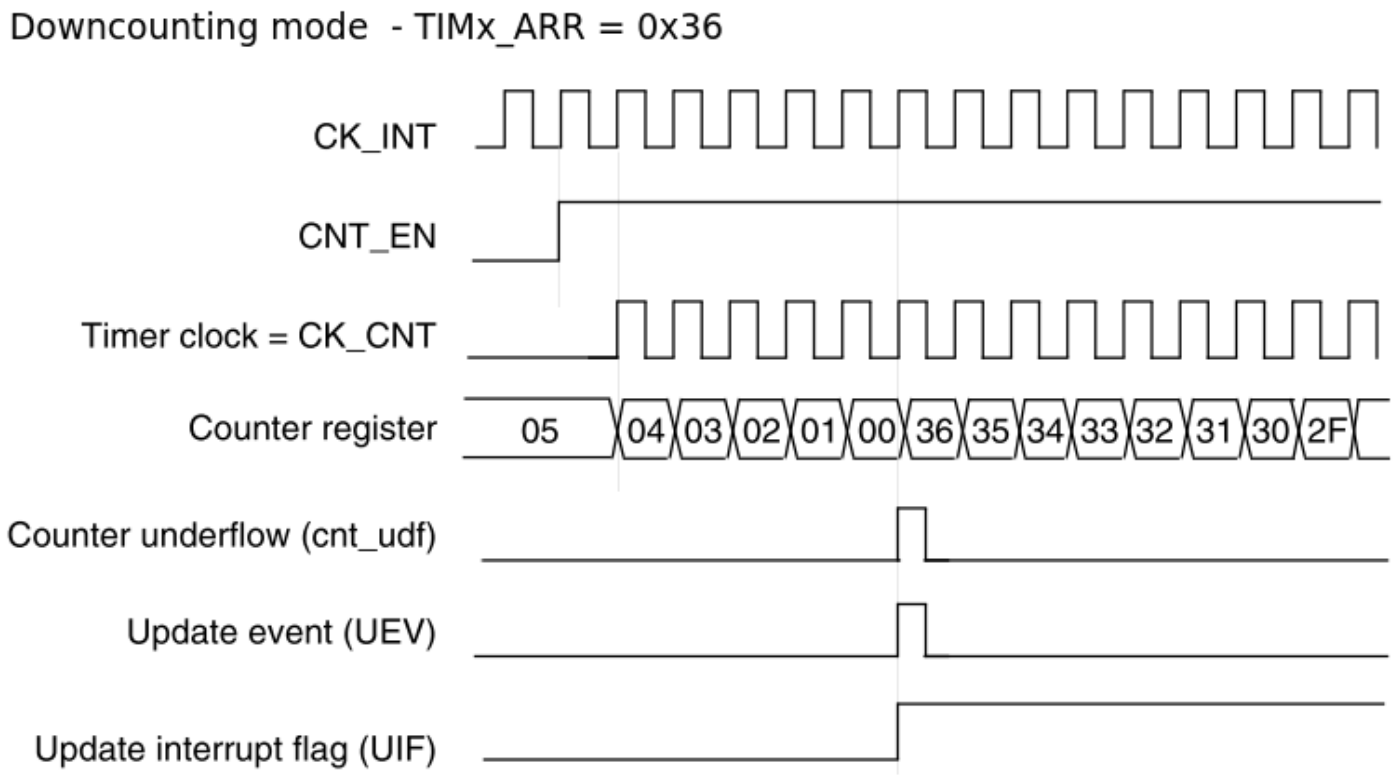
- Example: TIMx_ARR = 0x36, ARPE = 1



UEV(clear by hardware) occurs every time an overflow occurs, and UIF is a flag triggered when overflow occurs, if you don't clean the flag, it will remain triggered(clear by software).

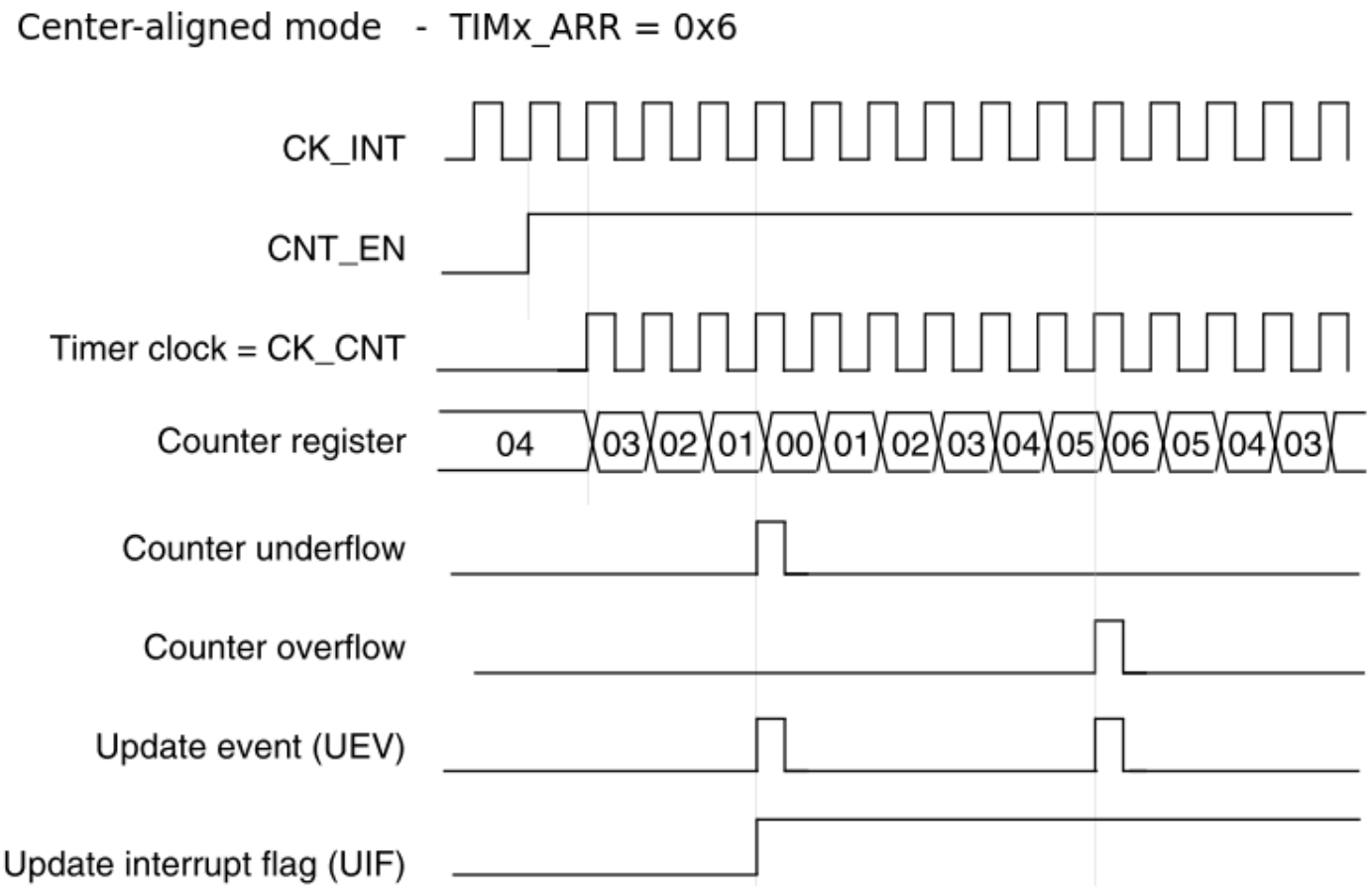
Downcounting mode

```
TIM_TimeBaseInitStruct.TIM_CounterMode = TIM_CounterMode_Down;
```



Center-aligned mode

```
TIM_TimeBaseInitStruct.TIM_CounterMode = TIM_CounterMode_CenterAligned1;
```



Functional Modes

Input capture mode

10

- 目的 : 用來計算某外部信號特定戀發生的時間點
- 用途 : 脈波寬測量, 頻攆測
- 可以設定外部觸發信號的型式，並使用外部觸發(ICx signal)來觸動一個Timer的栓鎖動作，這時候counter計數值則會存入TIMx_CCRx暫存器。

PWM input mode

11

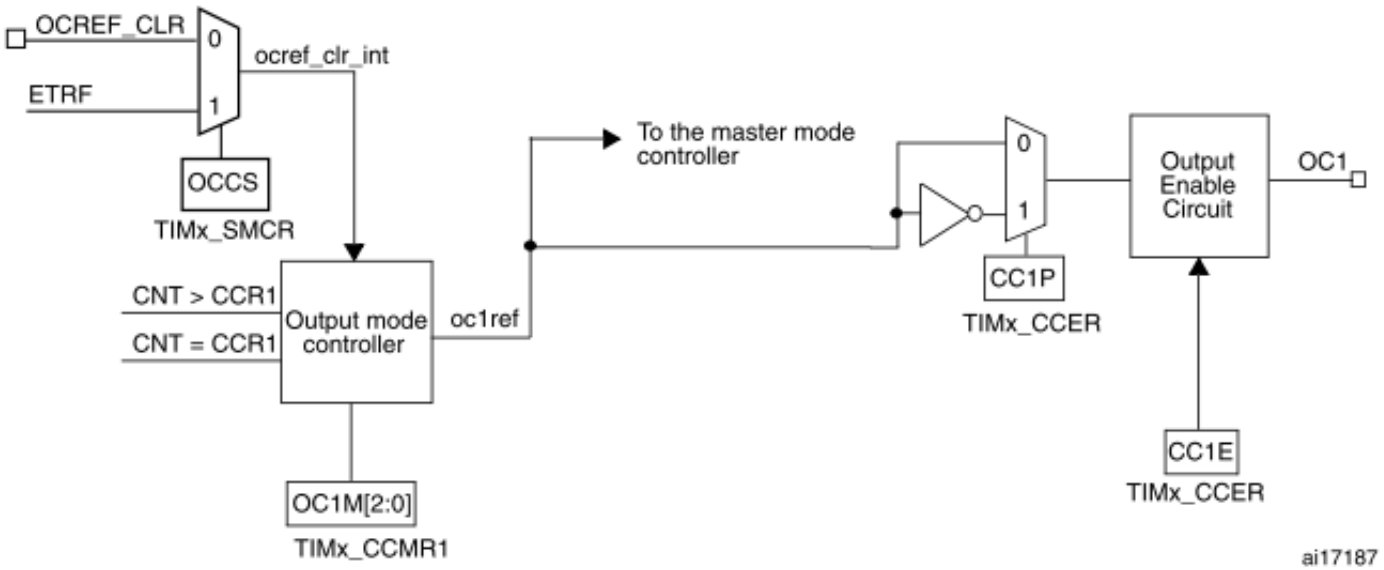
- 為Input capture mode的一個特例，此模式能夠量測到TI1上的PWM訊號週期(存於TIMx_CCR1暫存器)和工作週期(存於TIMx_CCR2暫存器)，所以會使用到兩個通道。

Forced output mode

12

```
TIM_OCInitStruct.TIM_OCPolarity = TIM_ForcedAction_InActive
TIM_OCInitStruct.TIM_OCPolarity = TIM_ForcedAction_Active
```

- 在輸出模式下，輸出比較信號能夠直接由軟體強制設為有效(active)或是無效(inactive)的狀態，而無視輸出比較暫存器和計數器之間的比較結果。



- TIMx_CCMRx = 100 -> force **inactive** level -> ocxref force **low**
- TIMx_CCMRx = 101 -> force **active** level -> ocxref force **high**
- 而OCx又會受到CCxP bit (TIM_OCPolarity)影響
    - CCxP = 0 (TIM_OCPolarity = TIM_OCPolarity_Low) -> OCx = ocxref (default)

- CCxP = 1 (TIM_OCPolarity = TIM_OCPolarity_High) -> OCx = !ocxref
- 對應

```
TIM_OCInitStruct.TIM_OCPolarity = TIM_OCPolarity_Low;
TIM_OCInitStruct.TIM_OCPolarity = TIM_OCPolarity_High;
```

Output compare mode

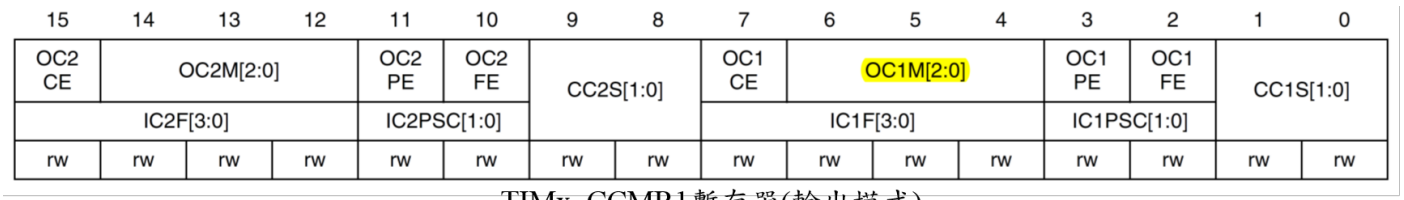
13

```
TIM_OCInitStruct.TIM_OCMode = TIM_OCMode_Toggle
```

- 此項功能是用來控制一個輸出波形或是用來指示一段給定的時間已到。

Detail of OC1M[2:0] in the TIMx_CCMR1 Register

14



TIMx_CCMR1暫存器(輸出模式)

C code define	OC1M[2:0]	說明
TIM_OCMode_Timing	000	凍結。 TIMx_CCR1與TIMx_CNT比較結果不會影響輸出。
TIM_OCMode_Active	001	當TIMx_CCR1=TIMx_CNT時，設置OC1REF為高電壓。
TIM_OCMode_Inactive	010	當TIMx_CCR1=TIMx_CNT時，設置OC1REF為低電壓。
TIM_OCMode_Toggle	011	反相(相反結果輸出)。 當TIMx_CCR1=TIMx_CNT時，反相OC1REF的電壓。
TIM_ForcedAction_InActive	100	TIMx_CCR1與TIMx_CNT比較結果不會影響輸出。 強制OC1REF為低電壓。
TIM_ForcedAction_Active	101	TIMx_CCR1與TIMx_CNT比較結果不會影響輸出。 強制OC1REF為高電壓。
TIM_OCMode_PWM1	110	向上計數：TIMx_CCR1<TIMx_CNT，OC1REF為高電壓。 向下計數：TIMx_CCR1>TIMx_CNT，OC1REF為低電壓。
TIM_OCMode_PWM2	111	向上計數：TIMx_CCR1<TIMx_CNT，OC1REF為低電壓。 向下計數：TIMx_CCR1>TIMx_CNT，OC1REF為高電壓。

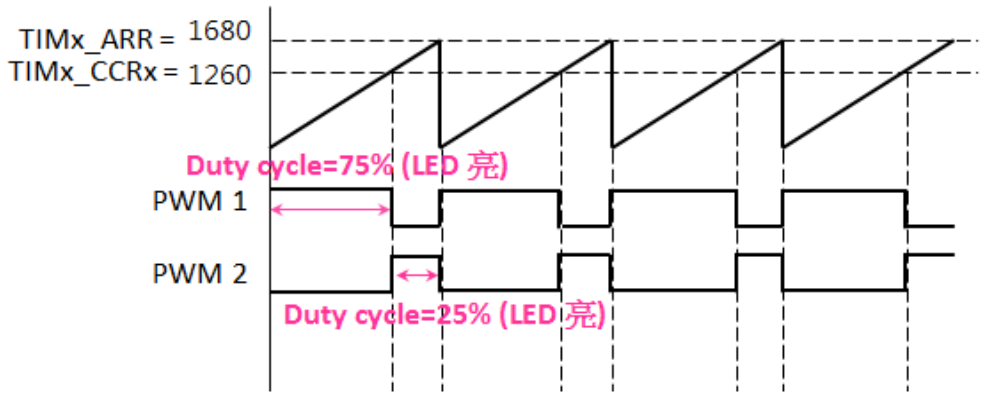
PWM mode

15

- 目的 : 產生一個由TIMx_ARR暫存器決定頻率、由TIMx_CCRx暫存器決定工作週期的PWM信號。
- PWM configuration
  - TIM_Prescaler : 將TIMxCLK除以(TIM_Prescaler + 1)

  - TIM_CounterMode : 選擇計數模式
  - TIM_Period : TIMx_ARR，counter 週期
  - TIM_OCPolarity: 設置輸出極性
  - TIM_OCMode: PWM模式
    - PWM 1 Mode :
      - 在向**上**計數，**TIMx_CNT<TIMx_CCRx**時，輸出為1，否則輸出為0；
      - 在向**下**計數，**TIMx_CNT>TIMx_CCRx**時，輸出為0，否則輸出為1。
    - PWM 2 Mode :
      - 在向**上**計數，**TIMx_CNT<TIMx_CCRx**時，輸出為0，否則輸出為1；
      - 在向**下**計數，**TIMx_CNT>TIMx_CCRx**時，輸出為1，否則輸出為0。
  - TIM_Pulse: 即TIMx_CCRx暫存器，設定脈衝寬度
  - **輸出脈波週期 = (TIM_Period + 1) * (TIM_Prescaler + 1) / TIMxCLK**

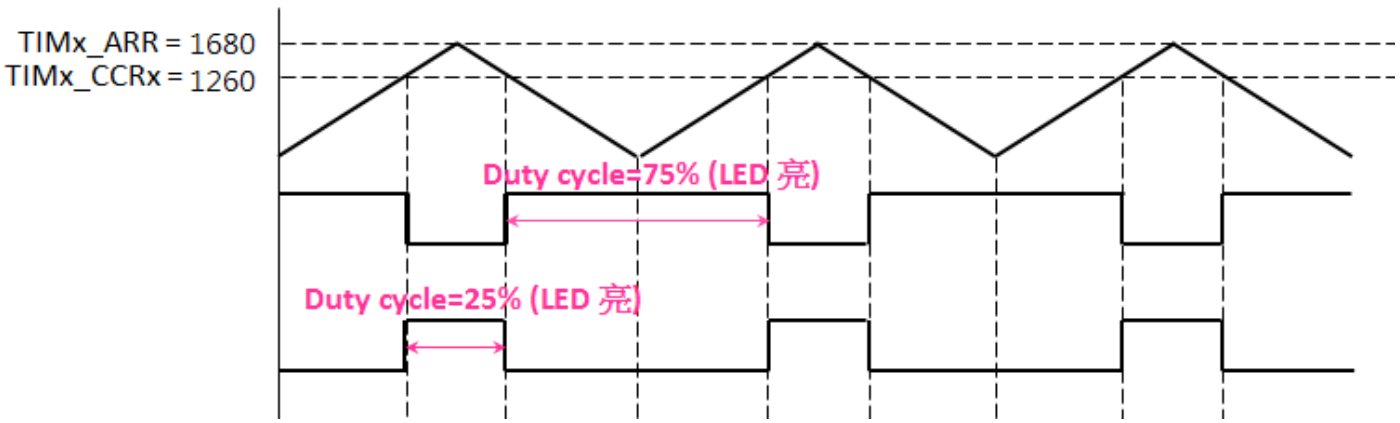
- Example: PWM mode 1, 向上計數並以LED為例：



$$duty\ cycle = \frac{TIM\_CCRx}{TIM\_ARR}$$

  - TIM_Prescaler = 500-1, TIM_Period = 1680-1 , TIMxCLK = 84MHz
  - 輸出脈波週期 = (TIM_Period + 1) * (TIM_Prescaler + 1) / TIMxCLK
    - = (1680 - 1 +1) * (500 - 1 + 1) / 84000000 = 0.01s(100Hz)
  - 0.01 s 遠小於人眼視覺暫留的時間(0.1~0.4s)，因此上圖的PWM 1 duty cycle(75%)>PWM 2 duty cycle(25%)，PWM 1 Mode看起來會比PWM 2 Mode亮。

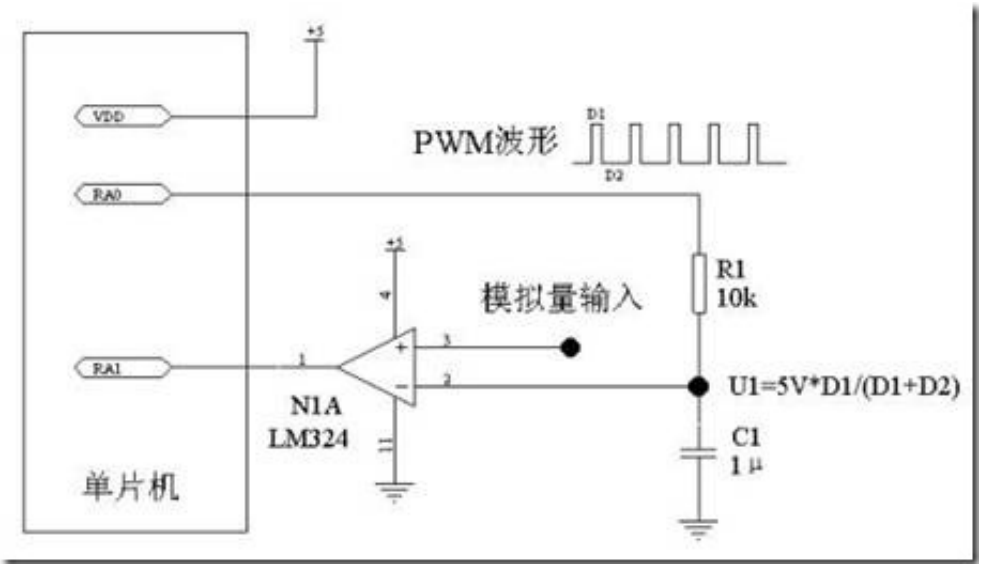
- Example: PWM mode 1, 中央對齊計數方式:



PWM realize ADC and DAC

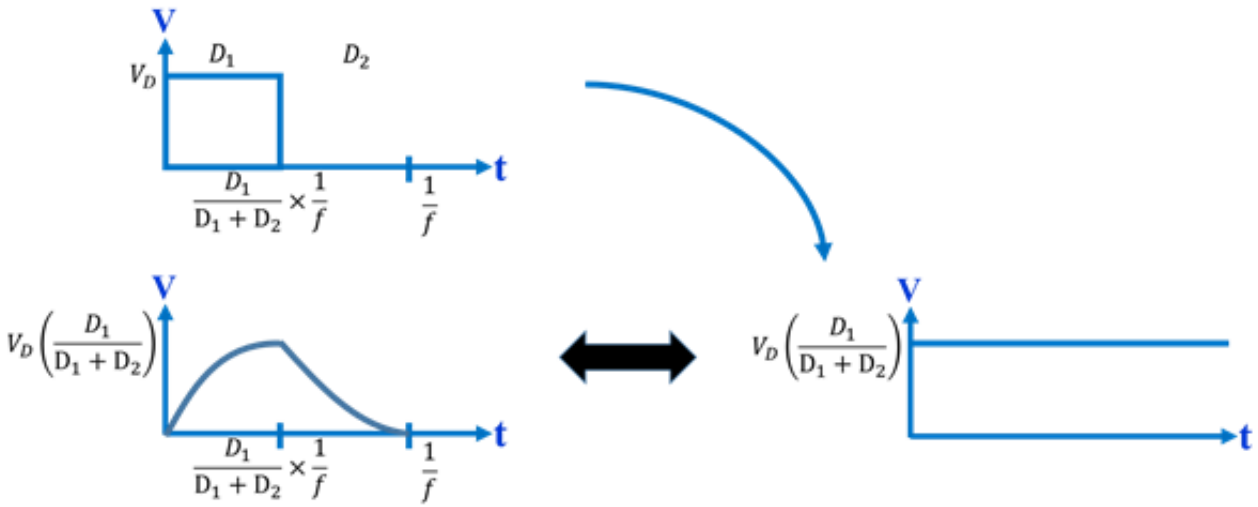
16

PWM realize ADC :



- principle brief:
  - 利用PWM可控制duty_cycle之特性，配合電阻和電容的濾波電路即可產生小於VDD之任意電壓值(U1= 5* (D1 / (D1+D2)))。
  - 再利用OPA比較類比輸入和PWM所產生的輸入壓即可得知現在所輸入類比訊號值。

RC value design:



$$\begin{cases} 充電：V_c = V_D \times \left(1 - e^{\frac{-t}{\tau}}\right) \\ 放電：V_c = V_{(10)} \times e^{\frac{-t}{\tau}} \end{cases}$$

$$\begin{cases} V_D \times \dfrac{D_1}{D_1 + D_2} = V_D \times \left(1 - e^{-\left(\frac{D_1}{D_1+D_2} \times \frac{1}{f}\right) \times \frac{1}{\tau}}\right) \\ 0 = \dfrac{D_1}{D_1 + D_2} \times V_{CC} \times e^{-\left(\frac{D_1}{D_1+D_2} \times \frac{1}{f}\right) \times \frac{1}{\tau}} \end{cases} \Rightarrow \begin{cases} V_D = \left(1 - e^{\frac{-D}{f\tau}}\right) \\ 0 = D \times V_{CC} \times e^{\frac{-(1-D)}{f\tau}} \end{cases}$$

$$\begin{cases} e^{\frac{-D}{f\tau}} = 1 - D \\ 0 = e^{\frac{-(1-D)}{f\tau}} \end{cases} \Rightarrow \begin{cases} \dfrac{-D}{f\tau} = \ln(1-D) \\ \dfrac{-(1-D)}{f\tau} \cong \infty \end{cases} \Rightarrow \begin{cases} \tau = \dfrac{-D}{f \times \ln(1-D)} \\ \tau \cong 0 \left(\tau \propto \dfrac{(1-D)}{f}\right) \end{cases}$$

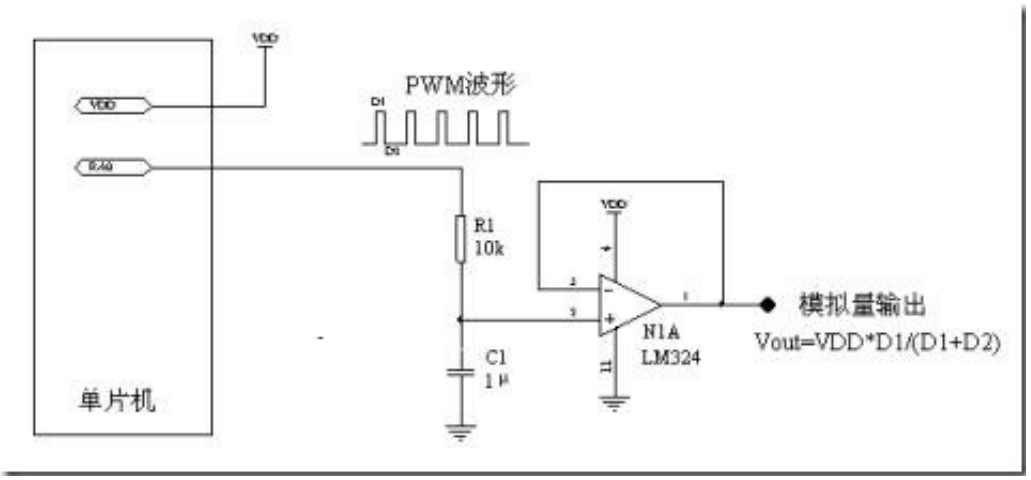
ADC transform :

- 首先讓PWM的duty_cycle從小開始增大(即是控制RCC暫存器從小到大)，當PWM所產生的電壓小於類比輸入則OPA輸出依然都是正值，而一當PWM所產生的電壓大於類比的輸入OPA即會變輸出負值，而由負轉正的瞬間RCC所存的值即是我們所要求的數位電壓值。
- ADC之解析度則是由控制PWM的duty_cycle的暫存器大小而決定，EX:STM32 TIM4 的RCC為16bit 而AD解析度就為16bit。

featrue of PWM realize ADC:

- 利用RC濾波會產生漣波誤差，因此ADC的經度較不準。
- PWM產生和RC濾波都需要時間因此其ADC轉換較慢，較適合ADC轉換速率不高的產品。
- 由於U1電壓是由小加到大，因此電壓較小的類比訊號會較快轉換(即取樣頻率不固定)。
- AD轉換頻率可藉由加快加單晶片速度而提升。

PWM realize DAC:



- principle brief:
  - 觀念和利用PWM實現AD相同，利用RC濾波電路將不同duty cycle的PWM訊號轉類比訊號，只是其OPA當作一電壓隨偶器。
  - 電壓隨偶器 : 電壓不變傳到下一級

Code_section

17

RCC_Configuration

```
RCC_AHB1PeriphClockCmd( RCC_AHB1Periph_GPIOD , ENABLE );//Enalbe AHB for GPIOD
RCC_APB1PeriphClockCmd( RCC_APB1Periph_TIM4, ENABLE );//Enable APB for TIM4
```

GPIO_Configuration

```
GPIO_InitTypeDef GPIO_InitStructure;//Create GPIO_InitStructure
GPIO_StructInit(&GPIO_InitStructure); // Reset GPIO_structure
GPIO_PinAFConfig(GPIOD, GPIO_PinSource12, GPIO_AF_TIM4); // set GPIOD_Pin12 to AF_TIM4
GPIO_PinAFConfig(GPIOD, GPIO_PinSource13, GPIO_AF_TIM4); // set GPIOD_Pin13 to AF_TIM4
GPIO_PinAFConfig(GPIOD, GPIO_PinSource14, GPIO_AF_TIM4); // set GPIOD_Pin14 to AF_TIM4
GPIO_PinAFConfig(GPIOD, GPIO_PinSource15, GPIO_AF_TIM4); // set GPIOD_Pin15 to AF_TIM4
// Setup Blue & Green LED on STM32-Discovery Board to use PWM.
GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_12 | GPIO_Pin_13| GPIO_Pin_14| GPIO_Pin_15; //PD12->L
ED3 PD13->LED4 PD14->LED5 PD15->LED6
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;            // Alt Function - Push Pull
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init( GPIOD, &GPIO_InitStructure );
```

TIM_Configuration

```
TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStruct;//Create TIM Time Base Init structure
TIM_OCInitTypeDef TIM_OCInitStruct;//Create TIM Output Compare Init structure
// Let PWM frequency equal 100Hz. ( 84MHz / 1680 /500 = 100Hz )
// Let period equal 1600. Therefore, timer runs from zero to 1600.
TIM_TimeBaseStructInit( &TIM_TimeBaseInitStruct );//reset TIM_TimeBaseStructInit
TIM_TimeBaseInitStruct.TIM_Period = 1680 - 1;
TIM_TimeBaseInitStruct.TIM_Prescaler = 500 - 1;
TIM_TimeBaseInitStruct.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit( TIM4, &TIM_TimeBaseInitStruct );
TIM_OCStructInit( &TIM_OCInitStruct );//reset TIM_OCStructInit
TIM_OCInitStruct.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStruct.TIM_OCMode = TIM_OCMode_PWM1;
// TIM4_CCR register (16 bits). Value can range from zero to 65535.
// TIM_Pulse = Compare Capture register(CCR) = 1680 ( duty_cycle = 100%)
TIM_OCInitStruct.TIM_Pulse = 1680; //(0=Always Off, >1680 =Always On)
TIM_OC1Init( TIM4, &TIM_OCInitStruct ); // set TIM_OCInitStruce to TIM4_channel1
TIM_OC2Init( TIM4, &TIM_OCInitStruct ); // set TIM_OCInitStruce to TIM4_channel2
TIM_OC3Init( TIM4, &TIM_OCInitStruct ); // set TIM_OCInitStruce to TIM4_channel3
TIM_OC4Init( TIM4, &TIM_OCInitStruct ); // set TIM_OCInitStruce to TIM4_channel4
TIM_Cmd( TIM4, ENABLE ); //Enables  TIM4 peripheral
```

PWM control

```
while(1)  // Do not exit
{
  if(brightness + n <= 0)
      who_run = (who_run + 1) % 4;
  if (((brightness + n) >= 3000) || ((brightness + n) <= 0))
    n = -n; // if  brightness maximum/maximum change direction
  brightness += n;
  //Light LEDs in turn
   switch(who_run){
       case 0:
           TIM4->CCR1 = brightness - 1; // set brightness
           break;
       case 1:
           TIM4->CCR2 = brightness - 1; // set brightness
           break;
       case 2:
           TIM4->CCR3 = brightness - 1; // set brightness
           break;
       case 3:
           TIM4->CCR4 = brightness - 1; // set brightness
           break;
   }
```

```
  for(i=0;i<10000;i++);  // delay
}
return(0); // System will implode
}
```

Demo video

1. 輪流點亮LED
2. 透過PWM使MOTOR達到快慢快慢變化

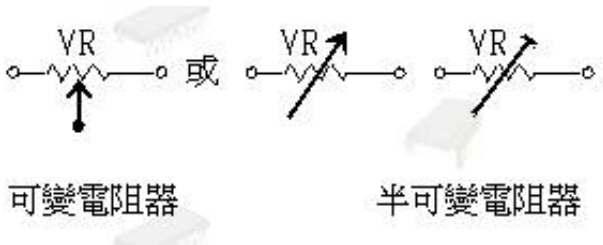
補充

PWM先高後低和先低後高 在Motor上表現有什麼差別?

> - 在Motor剛要啟動時，因為先高後低是在duty cycle的前半段先供電，而先低後高則是到duty cycle的後半段才供電，在Motor的表現上 先高後低的啟動速度會比先低後高還要快一點

可變電阻分為兩類(variable resistor):

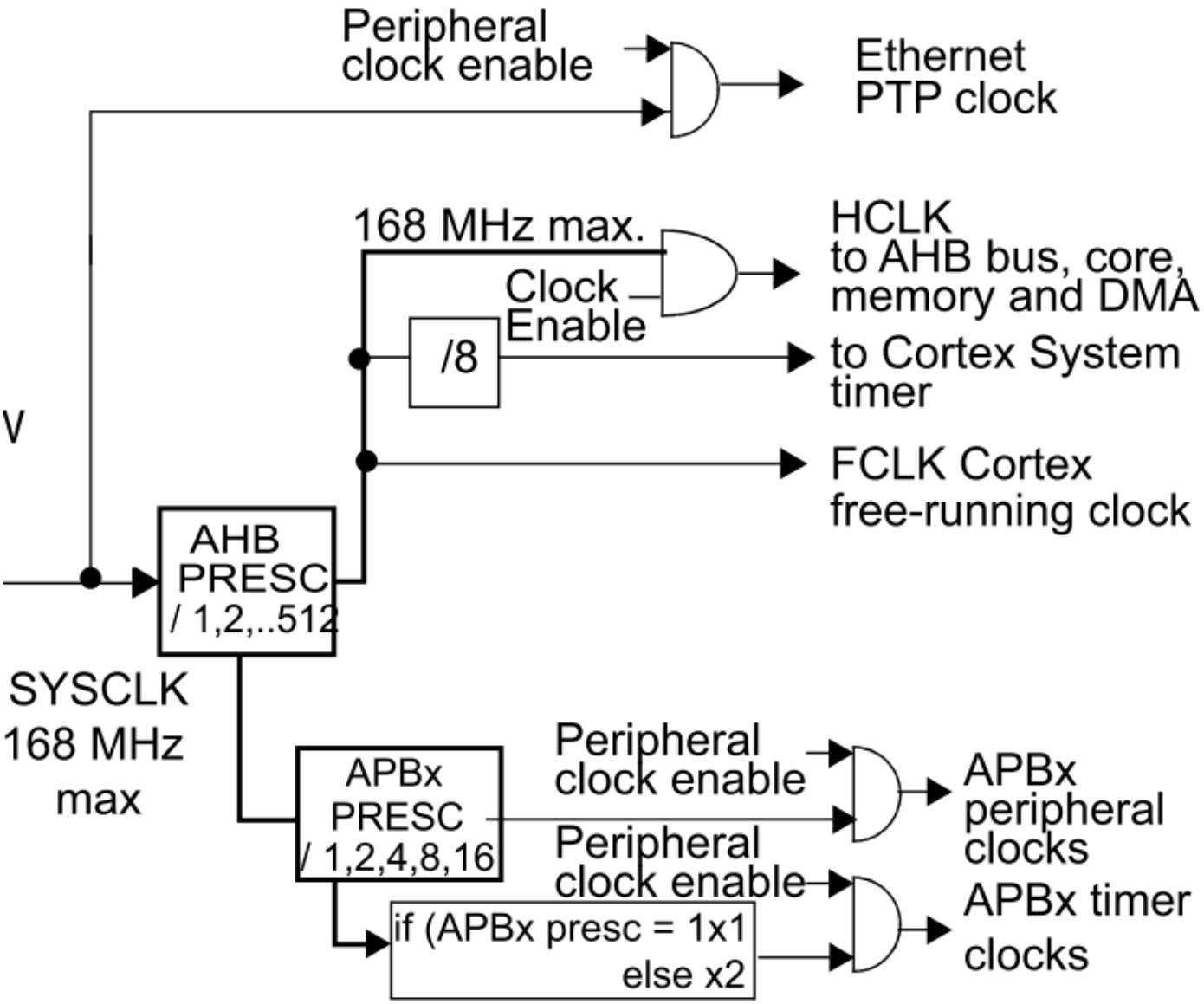
- (1)可變電阻器:電阻可調整。
- (2)半可變電阻器:為調整電阻後可以固定。



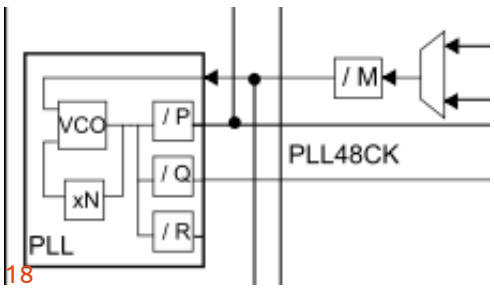
- 用處:
  - (1)可變電阻:用於阻值需要常常改變的電路用, 像音量控制, 搖桿, 類比指針式三用電表的最大值調整等.
  - (2)半可變電阻:多用於微調用或一次設定好後除非又偏差才需要調整的電路, 像類比指針式三用電表的歸零值調整等.
- Reference

Clock Tree

- Clock Tree 是把 Clock Source 分配到每個Device讓它的Skew minimize。
- 一般的邏輯閘fan out都有上限， 一推20就可能會推不上去拉，所以需要長tree 1推10，每個再推10個. 就像樹狀圖一樣，這就是clock tree。



- 為保證所有周邊裝置能夠得到期望的頻率, 所以先將頻率倍頻至最高需求頻率(Ethernet PTP clock), 其餘周邊裝置再除頻至所需之clock.
- 先倍頻在除頻係因倍頻與除頻的電路設計上的差異, 因PLL設計較為複雜, 而prescaler則以counter即可實現, 所以設計上不會直接將Oscillator產生之頻率倍頻至各裝置所需之頻率.

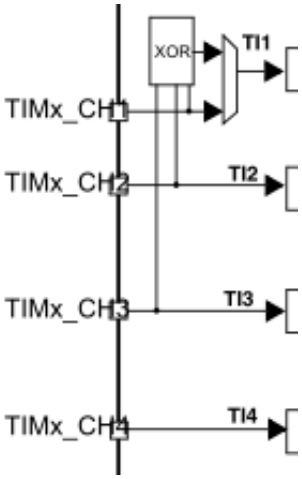


18

- PLL之前要先除頻原因為
  - 在頻率合成器中，只用除頻器並不足以產生適當的頻率，必須配合 N 倍頻電路才能產生更多樣的輸出頻率。
  - 例如以10 MHz 之參考頻率，若只用除頻器是無法獲得 3 MHz, 4 MHz, 6 MHz, 7 MHz, 8 MHz, 9 MHz 等頻率，然而若搭配 N 倍頻電路，則要產生上述頻率並不困難

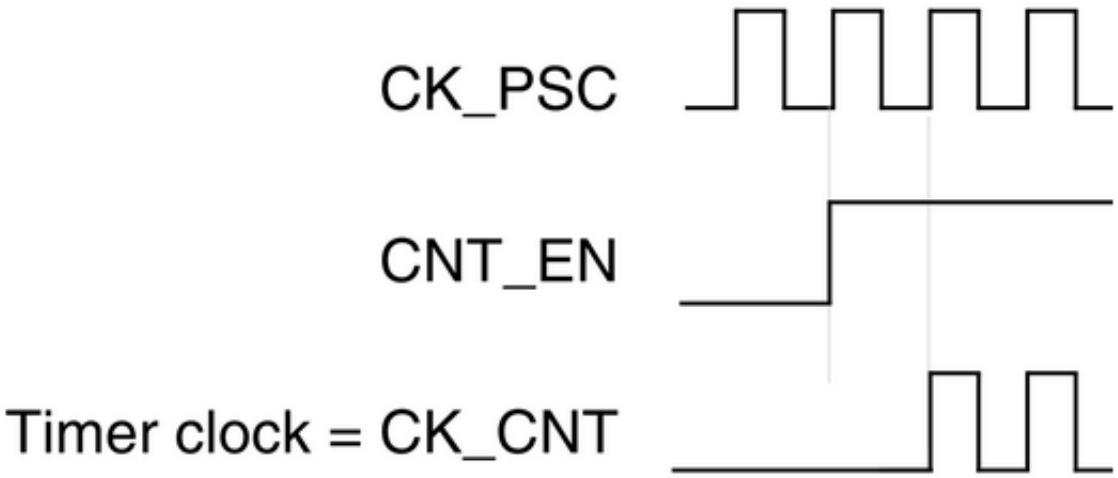
TIx XOR

19

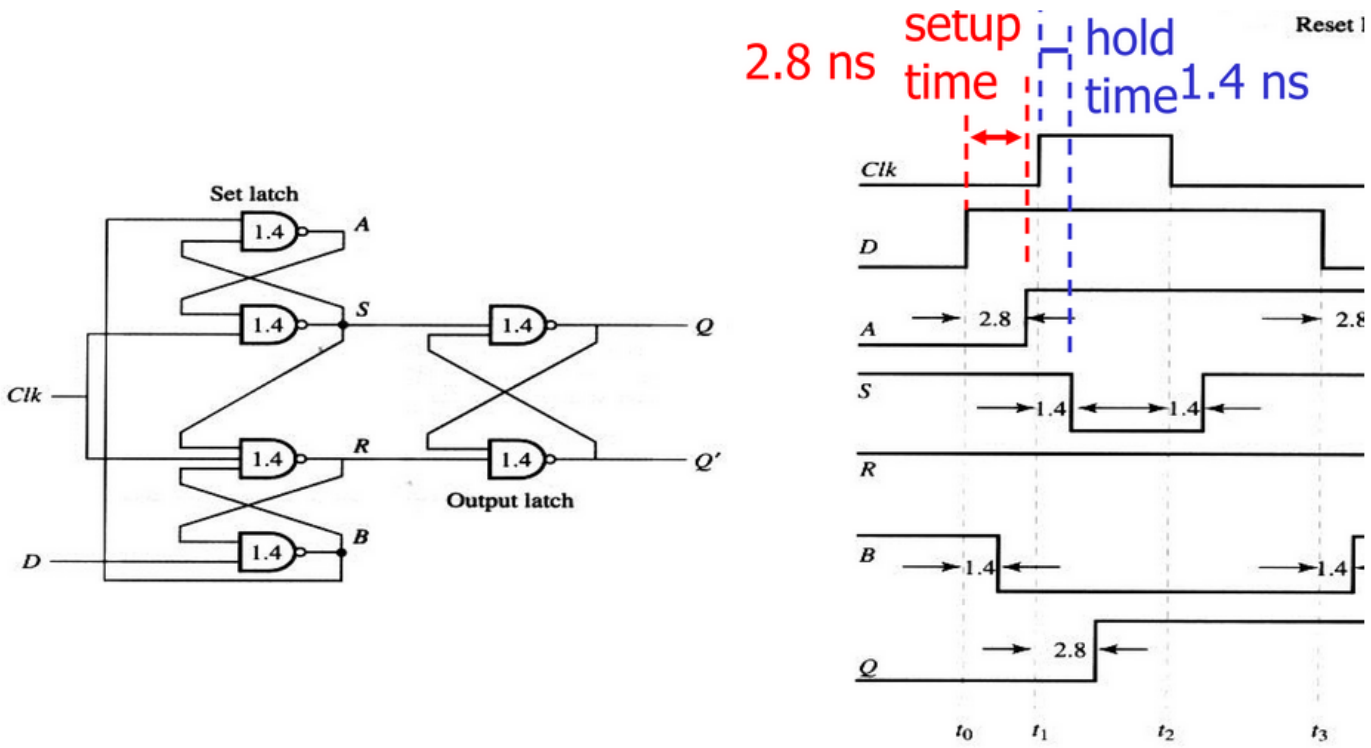


- XOR設置目的為:有三個輸入的信號(例如霍爾感測器)。
  - 霍爾感測器物理意義為電壓與磁力成正比，所以可以藉由霍爾感測器得出的電壓算出Motor的轉速。

Setup Time



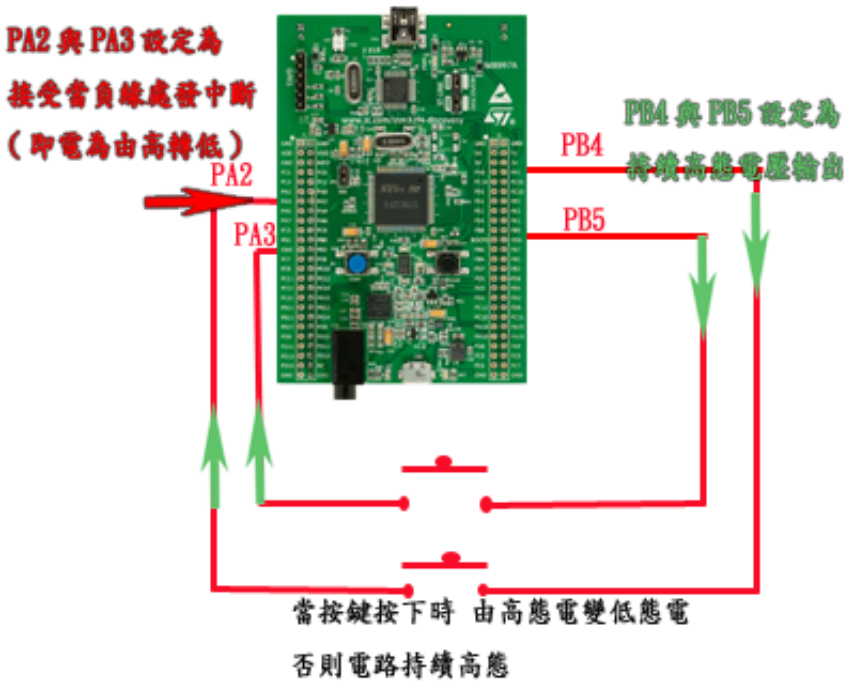
Ref. 數位IC設計, 陳培殷教授, slide 40, 41



範例程式

1. 輪流點亮LED
2. 藉由button改變duty cycle

> 此程式碼利用PB4 5作Output pin持續輸出高態電壓,PA2 3作Input Pin接收來在output pin的電壓輸入,並設定負緣觸發中斷,中斷觸發以改變duty cycle佔空比,電壓高低態變化則利用外接button,當按下button則電壓 高->低 ,PA2 3接收到變化便會觸發中斷,透過改變Timer來做duty cycle佔空比的改變,以達到LED亮暗改變.(以下圖說明)



Reference

Website

- Shadow register
- Wikipedia-PWM
- TIM_ClockDivision功用說明
- PWM實現ADC與DAC
- UNIVERSITA' DI BOLOGNA 的Timer 教材

Books

- Keil MDK ARM 原理與實作，MDK研究團隊編著，台科大圖書。
- 陳志旺（2012），STM32 嵌入式微控制器快速上手，中國：電子工業出版社。 P.155 8.3.3 計數器模式

1. Clock tree p. 212↩
2. system_stm32f4xx.c↩
3. RTC introducion↩
4. TIM6 和 TIM7 p. 678↩
5. TIM1 和 TIM8 p. 507↩
6. TIM9 to TIM14 p. 637↩
7. TIM2 to TIM5 p. 576↩
8. Counter Modes p. 579↩
9. Clock tree p. 578↩
10. Input capture mode p. 592↩
11. PWM input mode p. 593↩
12. Forced output mode p. 594↩
13. PWM input mode p. 594↩
14. TIMx_CCMR1 p. 621↩
15. PWM mode p. 595↩
16. PWM實現ADC與DAC額外補充↩
17. Demo Codes main.c↩
18. N倍頻電路↩
19. ITx XOR p. 602↩