embedded/RTC

Introduction

Real-Time Clock(RTC)是負責記錄時間的元件，出現在需要長期使用時鐘的電子設備中。例如學校定時關閉冷氣的裝置，以及手機上的鬧鈴功能。

System Overview

基本上RTC（Real-time clock）本身就是一個真正的時鐘，利用原本STM本身所內建的振盪器再利用Prescaler降成1Hz讓RTC使用。利用硬體達成的binary-coded decimal (BCD) format，可以把下列與時間有

關的資訊儲存，而且不需要任何軟體轉換，因為硬體就已經把資料轉成一般的日期格式。

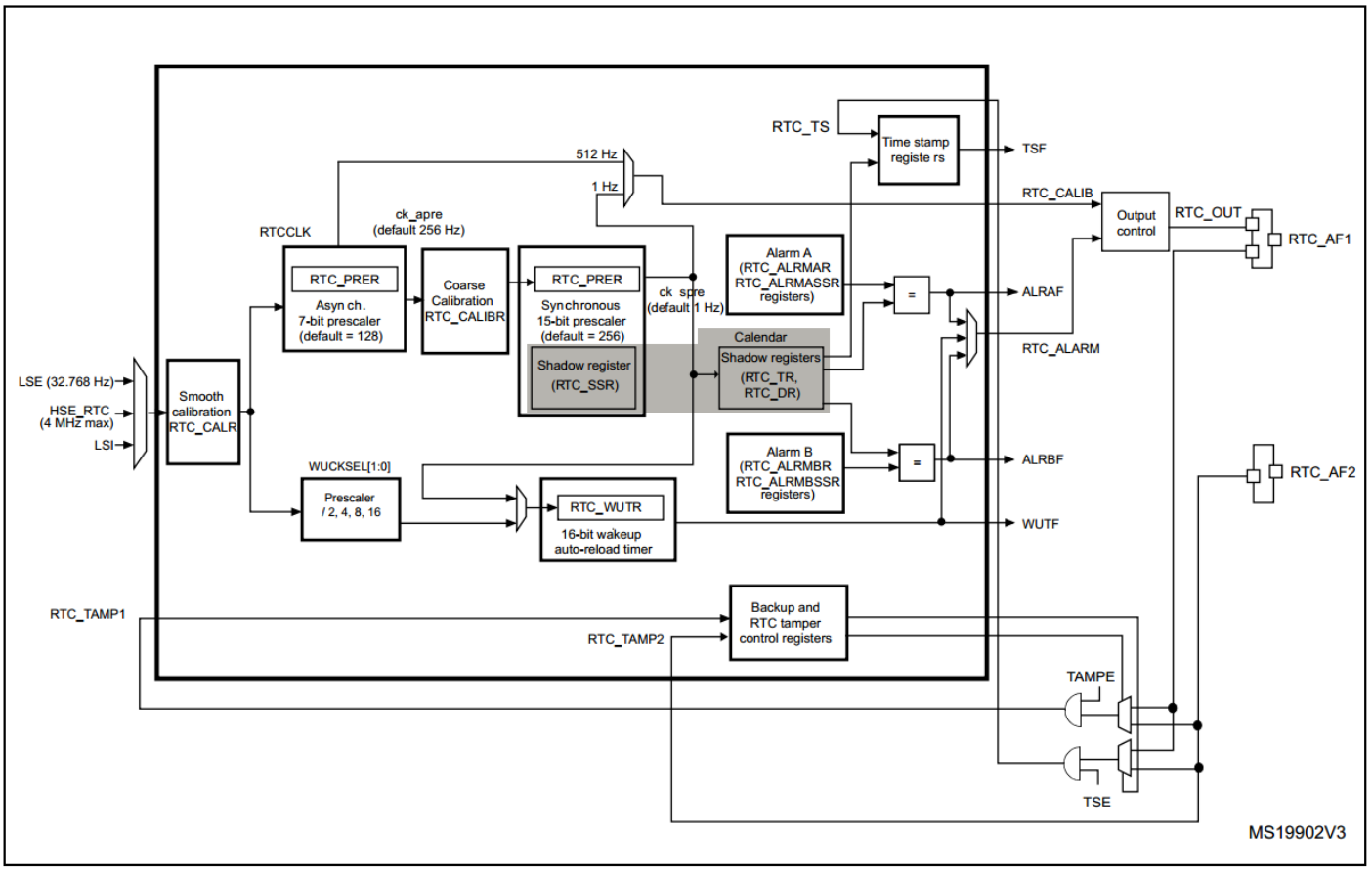
- sub-seconds
- seconds
- minutes
- hours in 12-hour or 24-hour format
- day of the week (day)
- day of the month (date)
- month
- year

可用的功能

- Alarm A & Alarm B
- Auto wakeup
- Timestamp
- Tamper detection

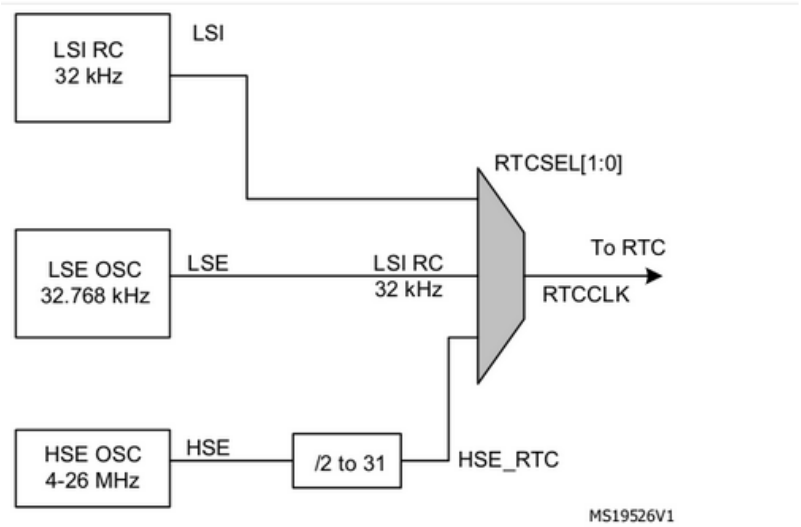
Block Diagram

1



Functional Description

Clock Source



STM32的 Clock 有分成「SYSCLK」和「Secondary Clock」：

SYSCLK為系統的clock有三種來源：

1. **HSI** (High-Speed Internal) ：16MHz,RC電路
2. **HSE** (Low-Speed External)：4-26MHz (一般選用8MHz),石英震盪
3. Main **PLL** clock

Secondary Clock:

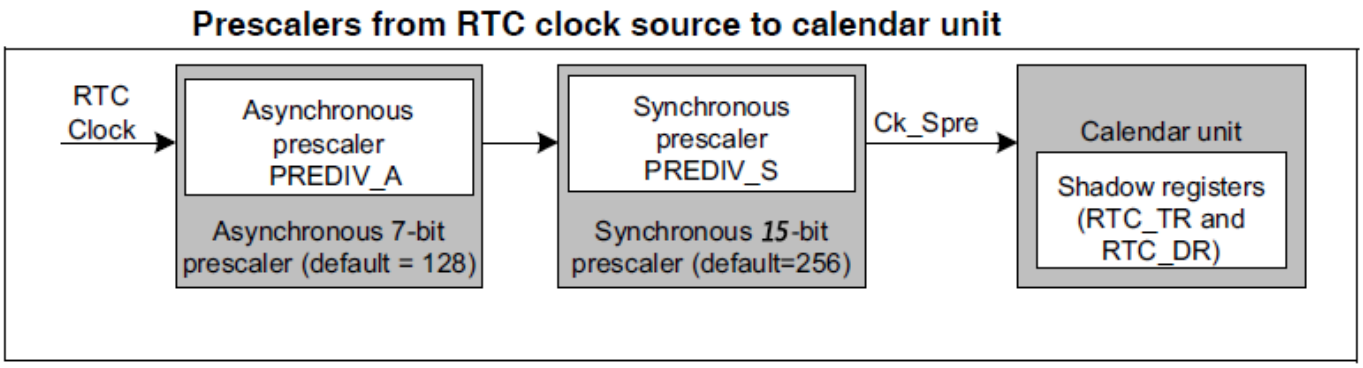
1. **LSI** (Low-Speed Internal) ：40kHz,RC電路震盪
2. **LSE** (High-Speed External)：32.768kHz,石英震盪

—-> RTC 主要是使用 **HSE** , **LSI** , **LSE** 三種來源 [2]

比較「HSE」,「LSI」,「LSE」三種輸入來源：

- **HSE - 較為耗能， 可以處理像是USB或TV訊號的clock，需要和和另一個clock穩定同步。**
- **LSI - 是一個低功耗的clock，可以在停機或待機模式下保持運行，用在auto-wakeup(簡稱AWU)與watchdog看門狗(簡稱IMDG)。**
- **LSE - 它是一個低功耗且精準的clock，適合用在時間的精確計算。**

Prescaler



`ck_spre` 一般而言要降為1Hz的頻率，因為省電的因素STM設計了兩個Prescaler。7 bit非同步

prescaler(`PREDIV_A`)和15 bit同步的prescaler(`PREDIV_S`)。這兩個prescaler要在RTC_PRER的暫存器設定。*ST在Reference有說明當兩個Prescaler都使用時，建議讓非同步的Prescaler讓他有較大的值，以讓系統更省電*

所以`ck_spre`與兩個prescaler的關係為:

$$ck\_spre = \frac{RTCCLK}{(PREDIV\_A + 1) \times (PREDIV\_S + 1)}$$

例如:

LSE: 32.768kHz / (**127**+1) / (**255**+1) = 1Hz

LSI: 32kHz / (**127**+1) /(**249**+1) =1Hz
3

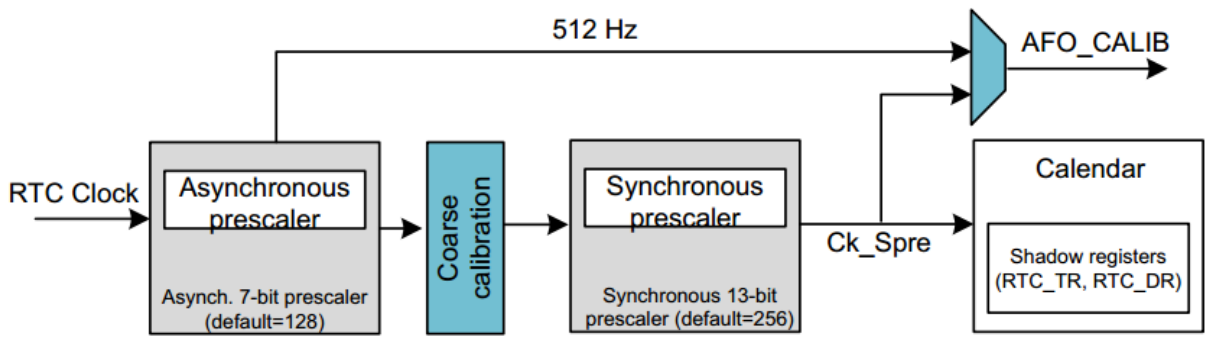
Calibration

RTC裡面有兩個校正功能，一個是coarse calibration另一個是smooth calibration。[4]

這兩個校正不能同時被使用，而前者只能固定修正，後者則可以動態的校正。後者也可以較大範圍及更精準的校正。

1、RTC coarse calibration ：

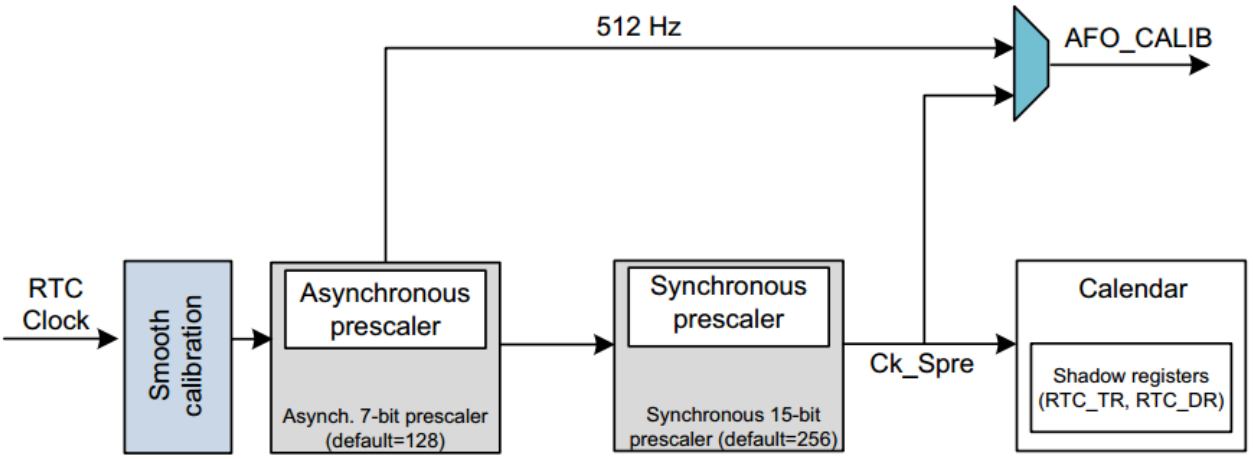
- **被使用在補償石英震盪器的校正。**
- **在非同步分頻器(ck_apre)的輸出，藉由增加或減少時間的週期達到校正，最大範圍的校正為63ppm~126ppm。**
- **只能在初始化Calendar時候修改，是一個開迴路控制**(opened loop control)(又稱非回授控制系統)。
- 可以利用AFO_CALIB去計算clock deviation。不能以圖中512Hz的訊號去確認coarse calibration的輸出結果，只能使用ck_spre確認校正的結果。
- reference clock calibration 和 coarse calibration 不能同時使用。



2、RTC smooth calibration：

- **可以補償石英震盪器的偏差，此偏差可能是晶體老化或者溫度所造成。**
- **利用每個RTCCLK pulse為單位做出小幅調整，來修正RTC的clock頻率。**

- 最大範圍的校正為-487.1ppm~+488.5ppm。
- 與coarse calibration不同，是屬於**閉迴路控制**(closed loop control)(又稱回授控制系統)。
- 可以使用AFO_CALIB來計算時間偏差，而且可直接檢查512Hz和1Hz的校正輸出。

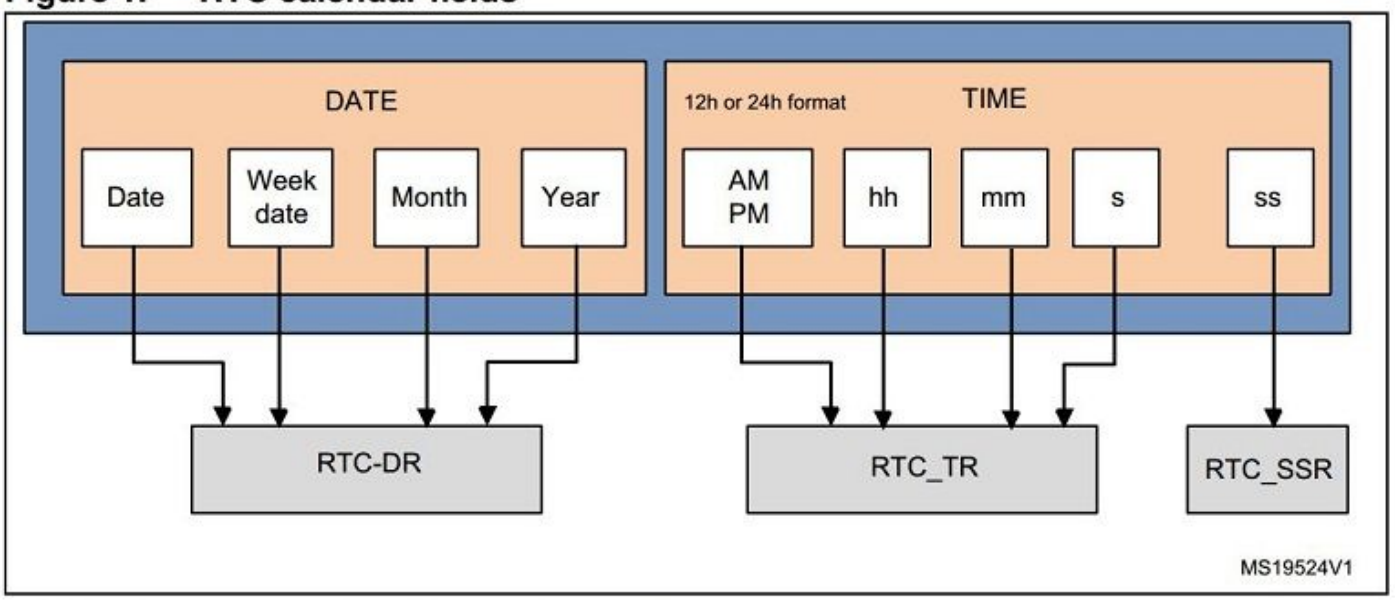


3、RTC reference clock detection：

- **利用外部時鐘校正，其時鐘源必須要比LSE的時鐘還精準。**[5]

RTC Calendar

Figure 1. RTC calendar fields



1. RCT_DR, RTC_TR are RTC Date and Time registers.
2. The sub-second field is the value of the synchronous prescaler's counter. This field is not writable.

讀取Calendar:

當在讀取Calendar時其實並不是真正直接讀取calendar register，其實是在讀shadow register，如果想要直接讀取calendar必須

要設BYPSHAD控制位元為1(在RTC_C Rregister)才能繞過shadow register而直接讀取calender register。

Shadow Register:

- 通常初始化和讀取的動作採用Shadow Register。
- 每兩個RTCCLK時間週期，RTC的calendar register (RTC_SSR, RTC_TR, and RTC_DR)的值會被自動更新到Shadow Register。

6

Alternate Function Outputs & Inputs

Alternate function 可以將RTC某些功能對應到輸出的接腳(GPIO Pin)，這些輸出可以被選擇成**tamp event** 或 **time stamp event**，

甚至RTC Calibration 的訊號都可以藉由這個功能輸出。 7

可以選擇兩個輸出:

- 輸出接腳 **RTC_AF1(PC13)** :
  - RTC_ALARM output: 1. RTC Alarm A 2. RTC Alarm B 3. RTC Wakeup
  - RTC_CALIB output
  - RTC_TAMP1
  - RTC_TS
- 輸出接腳 **RTC_AF2 (PI8)** :
  - RTC_TAMP1
  - RTC_TAMP2
  - RTC_TS

PI8接腳參考: datasheet p.10

Low Power Modes

- RTC 設計為最小的耗能，在休眠模式下，RTC將繼續動作。 8
- 當clock 的來源是LSE 或 LSI，在停止模式與待機模式下，RTC仍然活躍的動作。
- 在低功率(low power mode)模式下，Alarm, tamper event, time stamp event, and wakeup 依然會被中斷所啟動。

Mode	Description
Sleep	No effect RTC interrupts cause the device to exit the Sleep mode.
Stop	The RTC remains active when the RTC clock source is LSE or LSI. RTC alarm, RTC tamper event, RTC time stamp event, and RTC Wakeup cause the device to exit the Stop mode.
Standby	The RTC remains active when the RTC clock source is LSE or LSI. RTC alarm, RTC tamper event, RTC time stamp event, and RTC Wakeup cause the device to exit the Standby mode.

比較三種低功耗

- **睡眠模式（sleep mode）**：CPU停止工作，週邊設備繼續工作。
- **停機模式（stop mode）**：CPU停止工作，HSI、HSE進入關閉模式，STM32工作狀態仍然保留。（RTC正常運行）
- **待機模式（standby mode）**：STM32所有SRAM與暫存器內容，全部遺失（RTC正常運行）從待機模式喚醒相當於重新復歸。

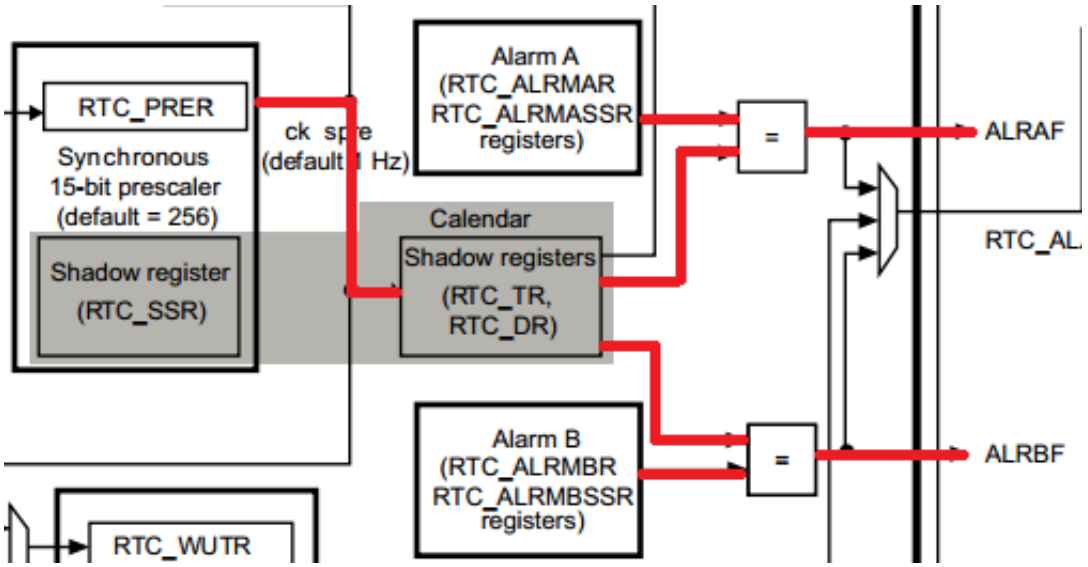
電流消耗

- **一般運行模式 ： 8.5mA**
- **低功耗模式**
  - 睡眠模式 ： 0.5mA
  - 停機模式 ：24uA
  - 待機模式：2uA

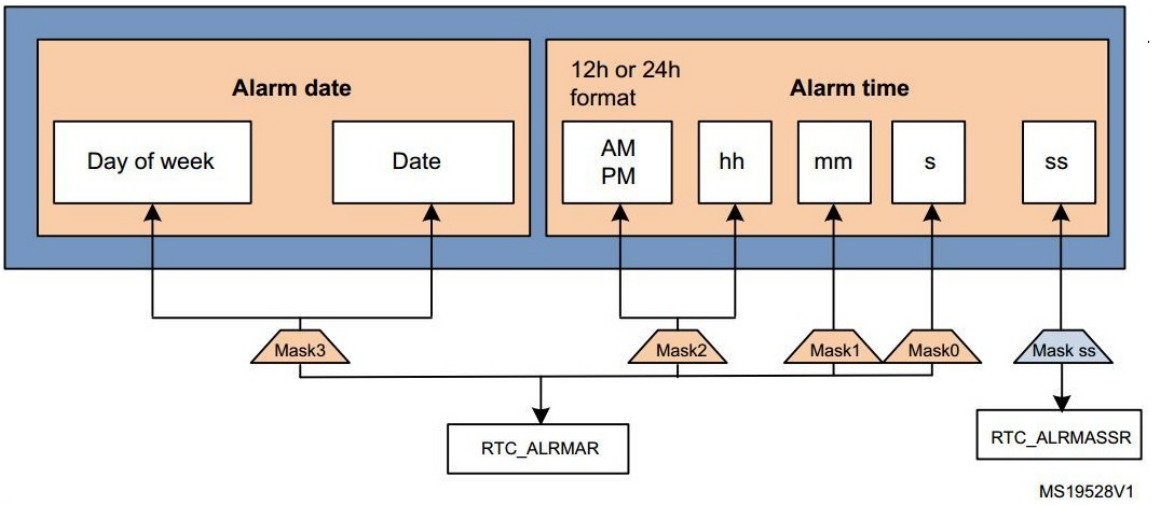
Interrupt Application

Alarm

- 有兩個時鐘Alarm A及Alarm B，可以當作鬧鐘使用。而且相關的register也都會有兩組。[9]
- 可利用Mask達成不同時間的鬧鈴效果。



- 以設定來說，生效的時候會透過EXTI_Line17的外部中斷RTC Alarm，讓ALRAF(ALRBF)的bit設成1 (Reference Manual p.380)



利用MSKx bits設定遮罩改變Alarm的行為: [10]

As an example, to configure the alarm time to 23:15:07 on Monday (預設MSK是0000)

MSK3	MSK2	MSK1	MSK0	Alarm behavior
0	0	0	0	**All fields are used in alarm comparison:** Alarm occurs at 23:15:07, each Monday.
0	0	0	1	**Seconds do not matter in alarm comparison** The alarm occurs every second of 23:15, each Monday.
0	0	1	0	**Minutes do not matter in alarm comparison** The alarm occurs at the 7th second of every minute of 23:XX, each Monday.
0	0	1	1	**Minutes and seconds do not matter in alarm comparison**
0	1	0	0	**Hours do not matter in alarm comparison**
0	1	0	1	**Hours and seconds do not matter in alarm comparison**
0	1	1	0	**Hours and minutes do not matter in alarm comparison**
0	1	1	1	**Hours, minutes and seconds do not matter in alarm comparison** The alarm is set every second, each Monday, during the whole day.
1	0	0	0	**Week day (or date, if selected) do not matter in alarm comparison** Alarm occurs all days at 23:15:07.
1	0	0	1	**Week day and seconds do not matter in alarm comparison**
1	0	1	0	**Week day and minutes do not matter in alarm comparison**
1	0	1	1	**Week day, minutes and seconds do not matter in alarm comparison**
1	1	0	0	**Week day and hours do not matter in alarm comparison**
1	1	0	1	**Week day, hours and seconds do not matter in alarm comparison**
1	1	1	0	**Week day, hours and minutes do not matter in alarm comparison**
1	1	1	1	**Alarm occurs every second**

Periodic Wakeup Unit

- 周期性的喚醒(wakeup)可以達成類似一個定時倒數的timer，一個downcounting且會auto-reload 的timer。當它的counter數到0的時候，在打開中斷的情況下會定時觸發中斷。
- 與Alarm不同，Wakeup是定時觸發中斷，Alarm則是在特定時間才會觸發。
- 會有兩個因素影響wakeup unit的中斷觸發週期，一個是timer儲存數字的大小，一個是clock source。

設定wakeup period [11]

有三種組態：

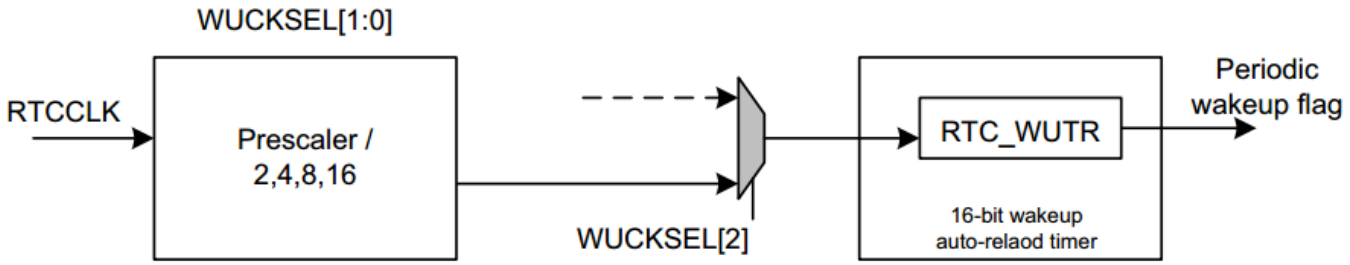
Configuration 1:**short wakeup** periods

Configuration 2:**medium wakeup** periods

Configuration 3: **long wakeup** periods

Configuration 1：

Prescalers connected to the timebase/wakeup unit



在這個設定中，wakeup unit，會接受由RTCCLK再經過Prescaler的clock。RTC_WUTR(RTC Wakeup Timer Register)為一個

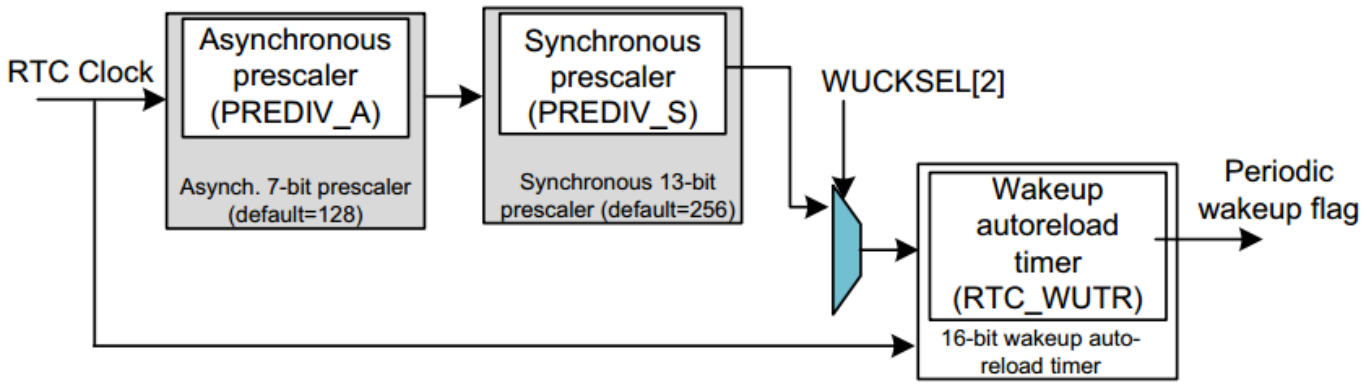
auto-reload的down counting timer，這表示當使用者在初始化時設了一個值x，會在x數到0的時候觸發wakeup flag。

EX：當RTCCLK= 32768 Hz，代表著最小的timebase resolution為61.035μs(數一次的時間)，最大則是488.28 μs。

- 所以這代表著最小可觸發wakeup flag時間為**(0x0001 + 1) x 61.035 μs = 122.07 μs**
- 最大可觸發wakeup flag時間為 **(0xFFFF+ 1) x 488.28 μs = 2 s**

註:STM32禁止timer初始值設為0

Prescalers connected to the wakeup unit for configurations 2 and 3



2和3的clock source相同，都是用來計算calendar的ck_spre。

Configuration 2：

- The minimum timebase/wakeup period is **(0x0000 + 1) x 61.035 μs = 122.07 μs.**
- The maximum timebase/wakeup period is**(0xFFFF+ 1) x 32 s = 131072 s (more than 36 hours).**

Configuration 3：

其Clock Source與 Configuration 2 一樣，差在2最大可以從0xFFFF倒數至0x0000，3則是0x1FFFF至0x00000

- The minimum timebase/wakeup period is: **(0x10000 + 1) x 61.035 μs = 250.06 ms**

- The maximum timebase/wakeup period is: **(0x1FFFF+ 1) x 32 s = 4194304 s (more than 48 days).**

Min. and max. timebase/wakeup period when RTCCLK= 32768Hz

Configuration	Minimum period	Maximum period
1	122.07 µs	2 s
2	122.07 µs	more than 36 hours
3	250.06 ms	more than 48 days

Time-stamp Function

實體世界裡有「郵戳為憑」來證明信件的時間，在網路世界裡如何來證明電子文件或交易的時間?

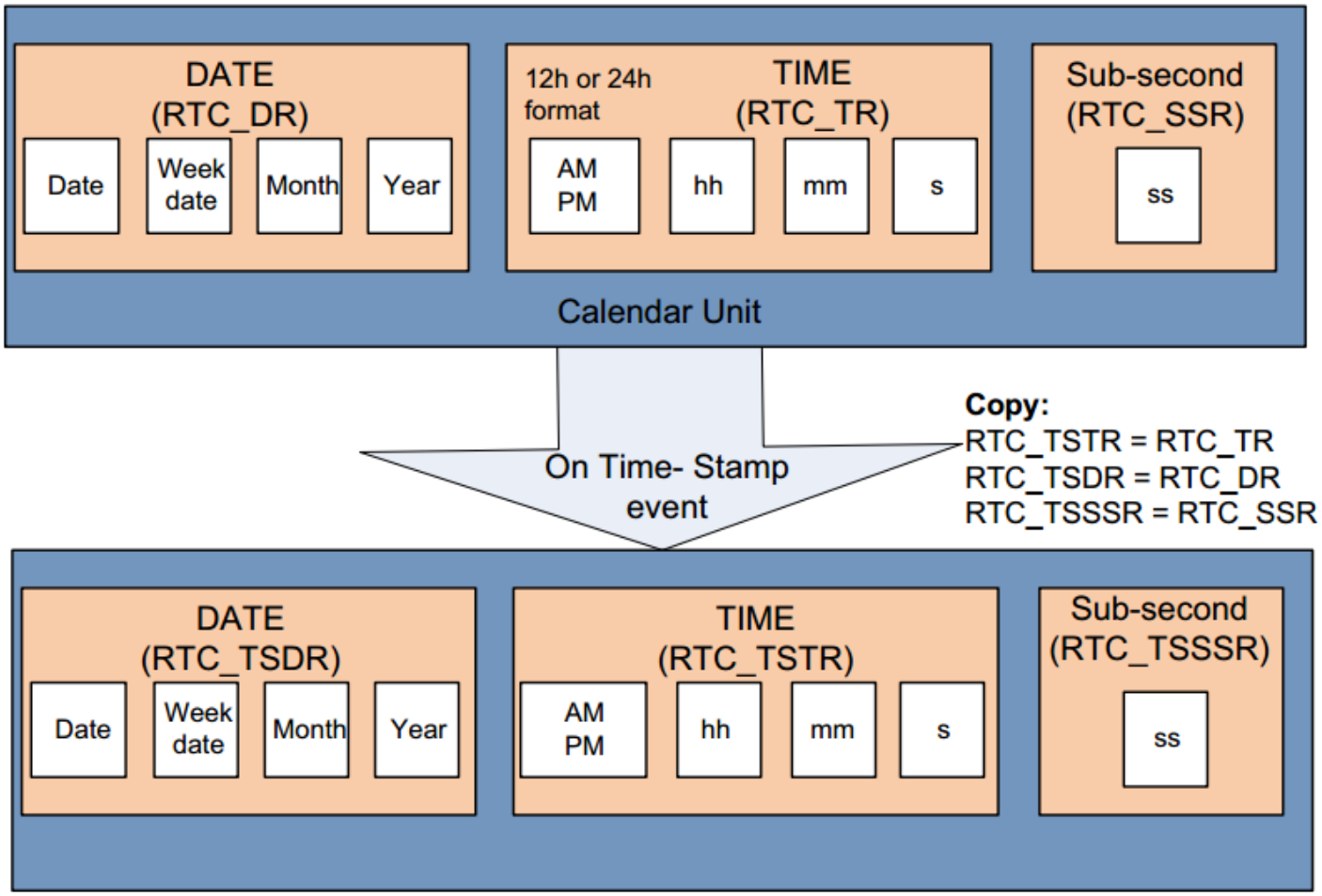
利用Time-stamp function 可以為任何電子文件或電子交易提供準確的時間證明，並且驗證其內容自蓋上時戳後是否曾被人修改過。[1][2]

Time-stamp function 提供自動幫你儲存日曆的功能

RTC_TSDR register : 會去讀取RTC_DR 裡面的年、月、日、周的資料並儲存

RTC_TSTR register : 會去讀取RTC_TR 裡面的秒、時、分的資料並儲存

RTC_TSSR register : 會去讀取RTC_SSR 裡面的Sub-second 的資料並儲存



Tamper Detection Function

Tamper Detection是一個可以檢測系統是否有被竄改的功能。在Tamper發生時，RTC Backup Register會全部自動reset，

達到保護系統的作用。而也可以在Tamper發生時，讓Timestamp發生。 [13]

Backup registers

在主電源 VDD(1.8~3.6V) 關掉以後，這個backup registers會動作在 VBAT (1.65~3.6V)的低功率電源(low power mode)，

所以它不會被system reset重設，它只會被tamper detection重設。或者因為backup domain reset。

如先前提到的，RTC_AF1和RTC_AF2可以被對映成不同接腳的輸入(alternate function)以偵測Tamper Event。

- TAMPER1 可以被對映到: RTC_AF1(PC13) 或RTC_AF2 (PI8).
- TAMPER 2可以被對映到: RTC_TAMP2 pin(PI8).

補充 : 何謂 Backup Domain ? [14]

Backup Domain 有下列幾個主要裝置:

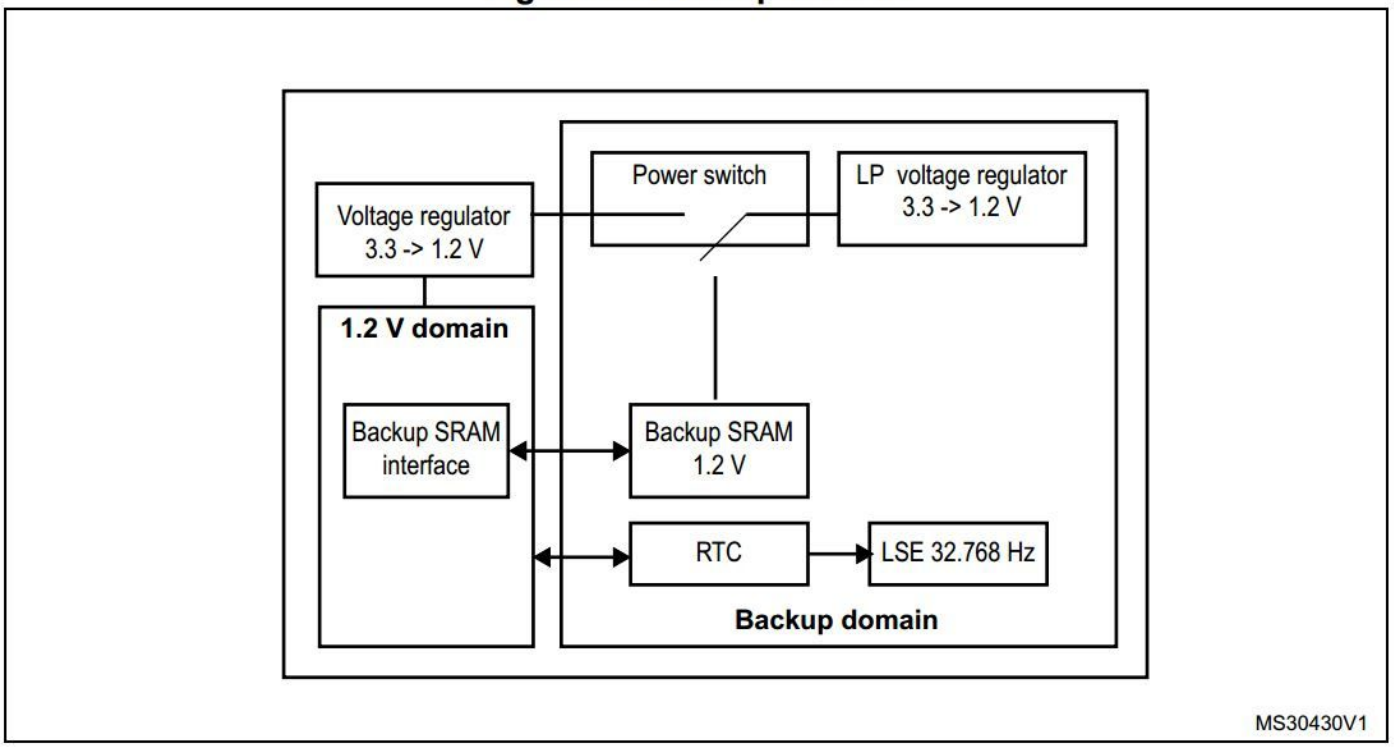
- The RTC
- The LSE oscillator
- The backup SRAM when the low power backup regulator is enabled

> PC13 to PC15 I/Os, plus PI8 I/O (when available)

當VDD停止供應電源時，Backup Domain會切到standby mode的電壓，換電池供電。在reset後，RTC registers, RTC backup register and backup SRAM會被保護以避免寫入。

SRAM:靜態隨機存取存儲器（Static Random-Access Memory, SRAM）是隨機存取儲存器的一種。所謂的「靜態」，是指這種存儲器只要保持通電，裡面儲存的數據就可以恆常保。

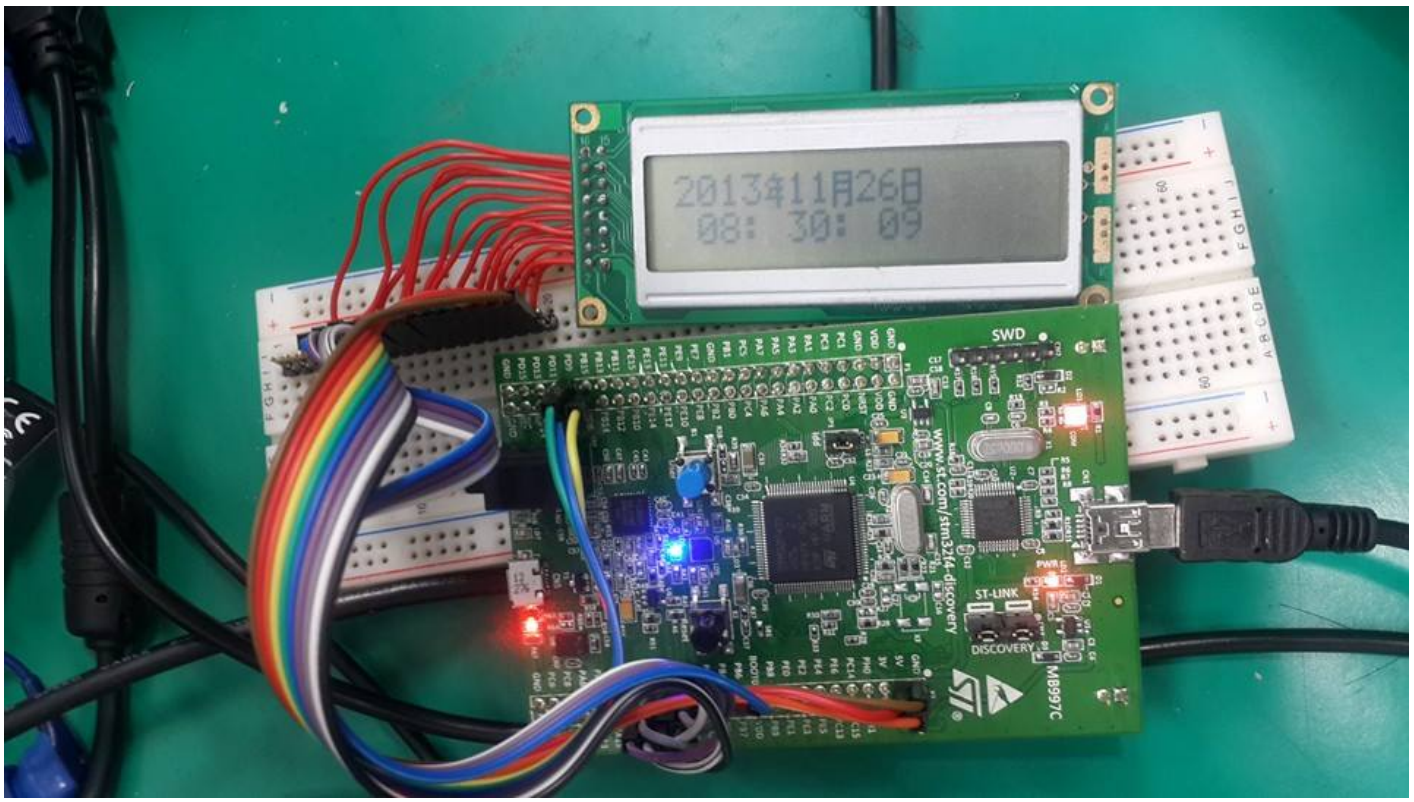
Figure 11. Backup domain



Example of Code

下面的範例為先初始化RTC的Calendar再去設定Alarm A。一開始開機時時間為8:29:55，LED3,4,5會在開機五秒前輪流閃爍。但是在第五秒後也就是8:30:00，Alarm A觸發中斷將所有LED燈都關掉。LED6 為auto-wakeup 每秒閃爍直到alarm觸發中止。

[YouTube連結–實現RTC]



Initialize RTC

```
RTC_InitTypeDef RTC_InitStructure;

RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR, ENABLE);     /* Enable the PWR clock */
PWR_BackupAccessCmd(ENABLE);                            /* Allow access to RTC */

RCC_LSICmd(ENABLE);                                     /* Enable the LSI OSC */
while(RCC_GetFlagStatus(RCC_FLAG_LSIRDY) == RESET);     /* Wait till LSI is ready */
RCC_RTCCLKConfig(RCC_RTCCLKSource_LSI);                 /* Select the RTC Clock Source */

RCC_RTCCLKCmd(ENABLE);                                  /* Enable the RTC Clock */
RTC_WaitForSynchro();                   /* Wait for RTC APB registers synchronisation */

/* Configure the RTC data register and RTC prescaler */
RTC_InitStructure.RTC_AsynchPrediv = 127;
RTC_InitStructure.RTC_SynchPrediv = 249;
RTC_InitStructure.RTC_HourFormat = RTC_HourFormat_24;
RTC_Init(&RTC_InitStructure);
```

15

Setting Time

```
/* set 8:29:55 */
RTC_TimeTypeDef RTC_TimeStruct;
RTC_TimeStruct.RTC_Hours = 0x08;
RTC_TimeStruct.RTC_Minutes = 0x29;
RTC_TimeStruct.RTC_Seconds = 0x55;

RTC_SetTime(RTC_Format_BCD, &RTC_TimeStruct);
```

Initialize RTC Alarm

```
EXTI_InitTypeDef EXTI_InitStructure;
NVIC_InitTypeDef NVIC_InitStructure;

/* EXTI configuration */
EXTI_ClearITPendingBit(EXTI_Line17);
EXTI_InitStructure.EXTI_Line = EXTI_Line17;
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
EXTI_InitStructure.EXTI_LineCmd = ENABLE;
```

```
EXTI_Init(&EXTI_InitStructure);

/* Enable the RTC Alarm Interrupt */
NVIC_InitStructure.NVIC_IRQChannel = RTC_Alarm_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

Setting Alarm Time

```
RTC_AlarmTypeDef RTC_AlarmStructure;
RTC_AlarmCmd(RTC_Alarm_A, DISABLE);   /* disable before setting or cann't write */

/* set alarm time 8:30:0 everyday */
RTC_AlarmStructure.RTC_AlarmTime.RTC_H12     = RTC_H12_AM;
RTC_AlarmStructure.RTC_AlarmTime.RTC_Hours   = 0x08;
RTC_AlarmStructure.RTC_AlarmTime.RTC_Minutes = 0x30;
RTC_AlarmStructure.RTC_AlarmTime.RTC_Seconds = 0x0;
RTC_AlarmStructure.RTC_AlarmDateWeekDay = 0x31; // Nonspecific
RTC_AlarmStructure.RTC_AlarmDateWeekDaySel = RTC_AlarmDateWeekDaySel_Date;
RTC_AlarmStructure.RTC_AlarmMask = RTC_AlarmMask_DateWeekDay; // Everyday
RTC_SetAlarm(RTC_Format_BCD, RTC_Alarm_A, &RTC_AlarmStructure);

/* Enable Alarm */
RTC_ITConfig(RTC_IT_ALRA, ENABLE);
RTC_AlarmCmd(RTC_Alarm_A, ENABLE);
RTC_ClearFlag(RTC_FLAG_ALRAF);
```

RTC_Alarm_IRQHandler

```
vTaskSuspend( pvLEDTask );
RTC_ClearITPendingBit(RTC_IT_ALRA);
EXTI_ClearITPendingBit(EXTI_Line17);
STM_EVAL_LEDOff(LED4);
STM_EVAL_LEDOff(LED3);
STM_EVAL_LEDOff(LED5);
```

RTC_WKUP_IRQHandler

```
if(RTC_GetITStatus(RTC_IT_WUT) != RESET)
```

```
{
  RTC_ClearITPendingBit(RTC_IT_WUT);
  EXTI_ClearITPendingBit(EXTI_Line22);
  STM_EVAL_LEDToggle(LED6);
}
```

Complete Code

LCD的程式碼, 有介紹LCD的腳位, 可顯示字串在畫面上

```
git clone https://github.com/zxc2694/stm32f4_LCD.git
cd stm32f4_LCD
make
make flash
```

利用LCD來做RTC的應用, 可以實現時間和日期的顯示

```
git clone https://github.com/fboris/stm32_RTC_demo.git
cd stm32_RTC_demo
make
make flash
```

LCD Introduction

LCD(Liquid Crystal Display)為液晶顯示器，功耗極低，常在控制周邊時，用來顯示數據的變化。裡面已經有內建文字圖形（只有英文字母大小寫、阿拉伯數字、標點符號），只要輸入**ASCII碼**，便會將該字的圖形顯示於LCD，如果要顯示中文字可以利用CGRAM，自己做出想要的字。

我們的LCD則是用來顯示RTC的時間與日期，而下面有個簡單介紹，顯示0到9字的程式碼。

Pin Connection

規格參考 ：http://www.sdec.com.tw/products.php?no=50

LCD使用了 **14 pin** .

- LCD pin 1 = ground
- LCD pin 2 = V+
- LCD pin 3 = ground (or use variable resistor)
- LCD pin 4 ~ 13 connection ：
  - RS = PD.08 (PIN 8 of GPIOD)
  - R/W = PD.09
  - E = PD.10
  - DB0 = PD.00

- DB1 = PD.01
- DB2 = PD.02
- DB3 = PD.03
- DB4 = PD.04
- DB5 = PD.05
- DB6 = PD.06
- DB7 = PD.07

Control Method

命令	指令編碼									
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1.清除顯示器	0	0	0	0	0	0	0	0	0	1
2.游標歸位	0	0	0	0	0	0	0	0	1	X
3.進入模式設定	0	0	0	0	0	0	0	1	I/D	S
4.顯示器ON/Off控制	0	0	0	0	0	0	1	D	C	B
5.顯示器或游標移動	0	0	0	0	0	1	S/C	R/L	X	X
6.功能設定	0	0	0	0	1	DL	N	F	X	X
7.CGRAM 位址設定	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0
8.DDRAM 位址設定	0	0	1	AD6	AD5	AD4	AD3	AD2	AD1	AD0

RS	R/W	功　　能
0	0	寫命令到 LCD
0	1	讀取忙碌旗號和位址計數器 AC(記錄目前游標位址)內容
1	0	寫資料到 DDRAM(要顯示的文字)或 CGRAM(要造字的字形)
1	1	從 DDRAM 或 CGRAM 讀取資料

Initial LCD Code

```
/* Initialization can decide whether to open a cursor, blink and display or not. */
void Init_LCD()
{
        vTaskDelay(100);
        LCD_CMD(0x003f);
        vTaskDelay(10);
        LCD_CMD(0x000c);
        vTaskDelay(10);
        LCD_CMD(0x0001);
        vTaskDelay(10);
}
```

Command Register Code

```
/* Write command register function */
void LCD_CMD(uint16_t cmd)
{
        int i;
        GPIO_SetBits(LCD_DBPORT, cmd);
        RS_0;
        RW_0;
        E_1;
        vTaskDelay(1);
        E_0;
        GPIO_ResetBits(LCD_DBPORT, cmd);
}
```

Data Register Code

```
/* Write data register function */
void LCD_DATA(uint16_t data1)
{
        int i;
```

```
        GPIO_SetBits(LCD_DBPORT, data1);
        RS_1;
        RW_0;
        E_1;
        vTaskDelay(1);
        E_0;
        GPIO_ResetBits(LCD_DBPORT, data1);
}
```

Display String

```
/* Testing LCD can display 0~9 */
void LCD_display(char row,char column, char display[])
{
        char *str;
        uint16_t address;
        str=display;
        address=0x0080+0x0040*(row-1)+(column-1);
        LCD_CMD(address);
        vTaskDelay(10);
        while(*str!=0){
        LCD_DATA(*str++);
        vTaskDelay(5);
        }
        str=display;
}
```

Q & A

1.潤年是否可調整？

Ans:

> 可以, RTC提供了自動調整潤年的功能。
> 參考手冊 p.536.(Compensations for 28-, 29- (leap year), 30-, and 31-day months are performed automatically.)

2.為什麼要存在shadow register?

Ans:

- 有些暫存器是兩層的,第一層是給CPU存取,第二層是給硬體存取,像是我們使用的**shadow register**與**硬體**其實它們兩個就是上下層的關係,不過是屬於同一個暫存器（**calendar register**）。CPU在寫入暫存器時,會先寫入在上層的shadow register,隨後硬體更新之後才會在下層動作, 所以shadow register 可以

直接對硬體存取 。

- 而當我們讀取calendar register時, APB1的clock頻率必須等於或大於RTC的clock頻率才行, **存在shadow register是為了可以確保同步機制的安全**, 如果APB1 clock頻率小於RTC clock頻率的話,CPU可能會讀到兩次的時間與日期。所以在初始化時,必須採用shadow register,才不會造成同步上的問題。

  參考手冊 p.539

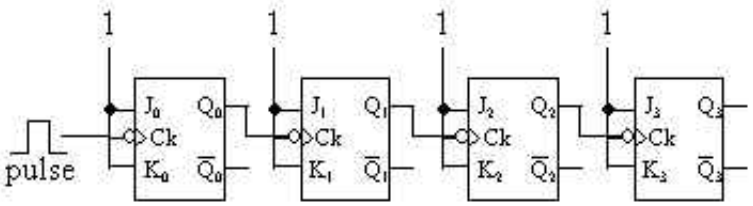
3.下數(down counter)如何實現？

Ans:

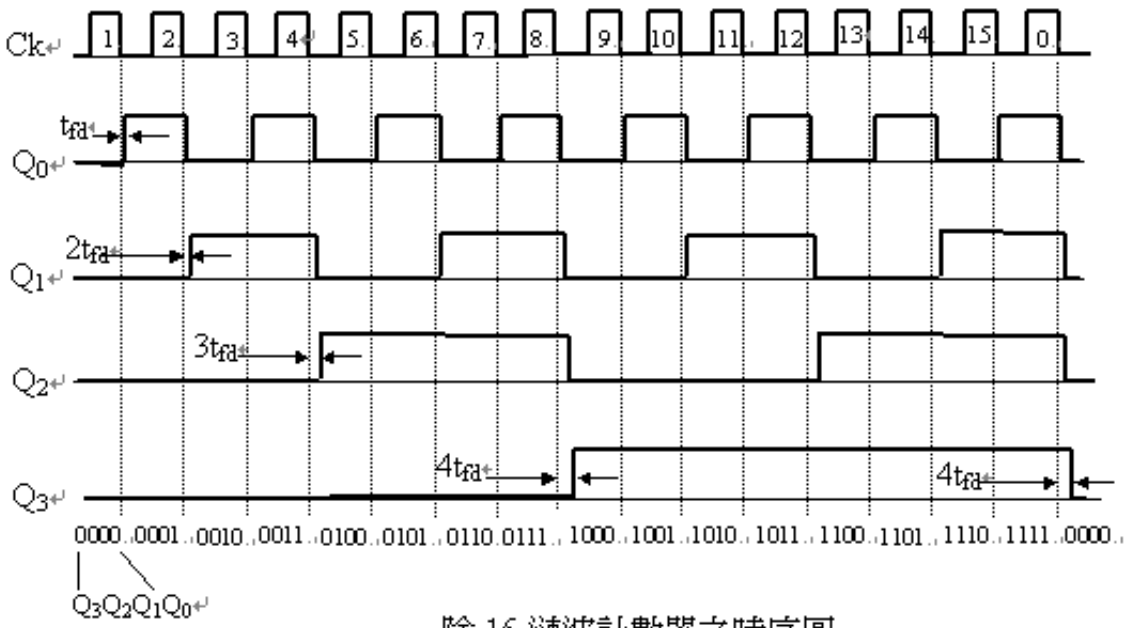
- 計數器由正反器構成，可以記錄狀態的變遷，或可說是正反器隨時脈的變化次數做故定狀態的循環。
- 要變成**下數計時器**只要在負緣觸發改成正緣觸發即可, 等於**時脈輸入端加了反閘**。

補充：

上數計時器

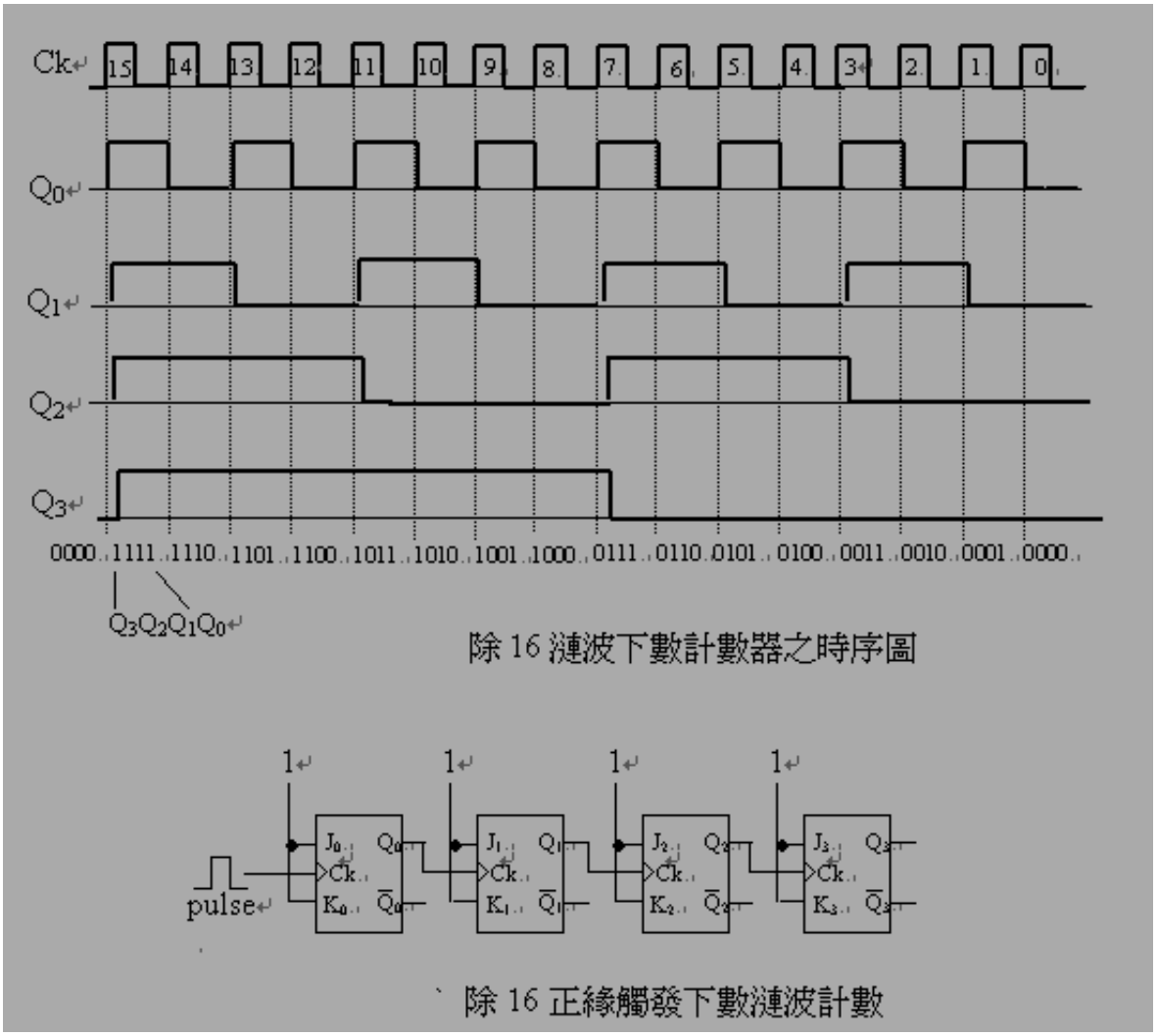


除 16(2^n 模)漣波計數器



除 16 漣波計數器之時序圖

下數計時器



4.Backup domain 為什麼從1.65V轉成1.2V？

Ans:

因一般電池的電壓為1.5~1.2V, 而Backup domain只需在1.2V以上即可工作。

ps. 一般電池的電壓為1.5v(鹼性電池)至1.2(一般及充電電池)

而有關於1.65v的部份應該是跟debugger的界面有關係，st的st-link供應的電壓為1.65v至5v，相似的設計在ti的msp 430 usb debugger 也可以看到類似的設計(1.8v~3.6v)

ST-LINK:http://www.st.com/web/catalog/tools/FM146/CL1984/SC724/SS1677/PF251168

MSP430 USB Debugging Interface :http://www.ti.com/tool/cc-debugger

Reference

RTC application note

STM32F405xx/07xx, STM32F415xx/17xx, STM32F42xxx and STM32F43xxx Reference Manual

STM32F407 Datasheet

LCD introduction (選左邊清單的“LCD控制”)

電池規格

---

1. Block Diagram p.780↩
2. Clock Source p.150↩
3. prescaler p.781↩
4. Calibration p.787↩
5. RTC reference clock p.786↩
6. Calendar p.780↩
7. Alternate Function Outputs & Inputs p.227↩
8. Low Power Modes p.126↩
9. Alarm p.781↩
10. Alarm application note p.11↩
11. Periodic Wakeup Unit p.782↩
12. Time-stamp Function p.790↩
13. Tamper Detection Function p.792↩
14. Tamper Detection Function p.115↩
15. Device overview p.18↩