ARM Counter and Timer

Part of this document is by
Prof. Shiao-Li Tsao
CS Dept., NCTU

System Clock Configuration
SYSCLK (Lab5.1)

6.4.3 Clock configuration register (RCC_CFGR) 選擇clk 來源

Address offset: 0x08

Reset value: 0x0000 0000

Access: 0 ≤ wait state ≤ 2, word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during clock source switch.

From 0 to 15 wait states inserted if the access occurs when the APB or AHB prescalers values update is on going.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	MCOPRE[2:0]			Res.	MCOSEL[2:0]			Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
	rw	rw	rw		rw	rw	rw								

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STOP WUCK	Res.	PPRE2[2:0]			PPRE1[2:0]			HPRE[3:0]				SWS[1:0]		SW[1:0]	
rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw

Bits 1:0 **SW[1:0]:** System clock switch

Set and cleared by software to select system clock source (SYSCLK).
Configured by HW to force MSI oscillator selection when exiting Standby or Shutdown mode.
Configured by HW to force MSI or HSI16 oscillator selection when exiting Stop mode or in case of failure of the HSE oscillator, depending on STOPWUCK value.

00: MSI selected as system clock

01: HSI16 selected as system clock

10: HSE selected as system clock

11: PLL selected as system clock

6.4.1 Clock control register (RCC_CR)

控制clock 開關

Address offset: 0x00

Reset value: 0x0000 0063. HSEBYP is not affected by reset.

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	PLL SAI2 RDY	PLL SAI2 ON	PLL SAI1 RDY	PLL SAI1 ON	PLL RDY	PLLON	Res.	Res.	Res.	Res.	CSS ON	HSE BYP	HSE RDY	HSE ON
		r	rw	r	rw	r	rw					rs	rw	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	HSI ASFS	HSI RDY	HSI KERON	HSION	MSIRANGE[3:0]				MSI RGSEL	MSI PLLEN	MSI RDY	MSION
				rw	r	rw	rw	rw	rw	rw	rw	rs	rw	r	rw

Bit 24 **PLLON:** Main PLL enable

Set and cleared by software to enable the main PLL.

Cleared by hardware when entering Stop, Standby or Shutdown mode. This bit cannot be reset if the PLL clock is used as the system clock.

0: PLL OFF

1: PLL ON

RCC_CR 第24bit是PLLON

RCC_PLLCFGR

6.4.4 PLL configuration register (RCC_PLLCFGR) 除頻

Address offset: 0x0C

Reset value: 0x0000 1000

Access: no wait state, word, half-word and byte access

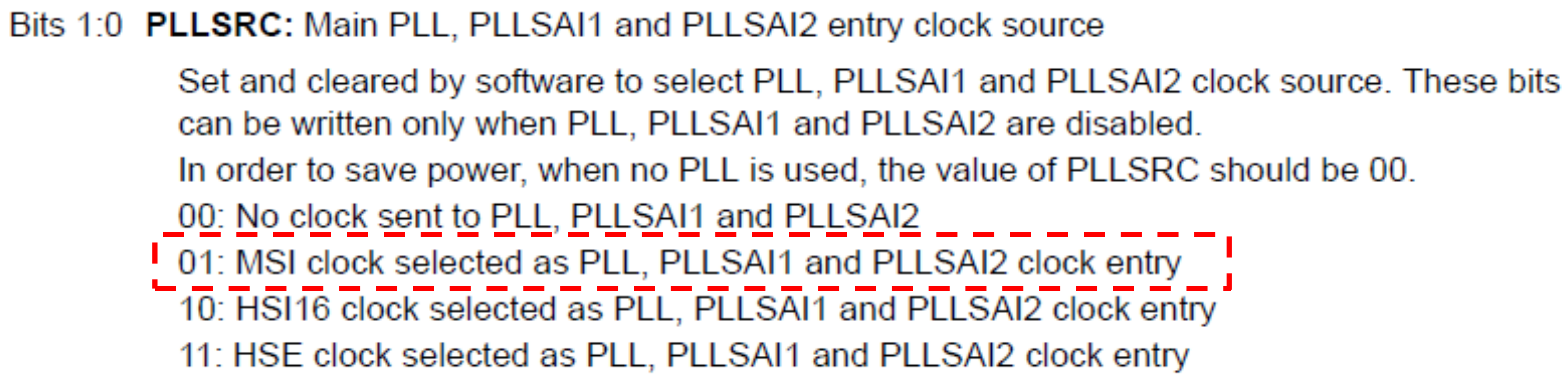
This register is used to configure the PLL clock outputs according to the formulas:

- f(VCO clock) = f(PLL clock input) × (PLLN / PLLM)
- f(PLL_P) = f(VCO clock) / PLLP
- f(PLL_Q) = f(VCO clock) / PLLQ
- f(PLL_R) = f(VCO clock) / PLLR

RCC_PLLCFGR第0,1 bit是PLLSRC
PLLM要shift 4
PLLN要shift 8
PLLR要shift 25
PLLREN要shift 24

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	PLLR[1:0]		PLL REN	Res.	PLLQ[1:0]		PLL QEN	Res.	Res.	PLLP	PLL PEN
					rw	rw	rw		rw	rw	rw			rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	PLLN[7:0]							Res.	PLLM[2:0]			Res.	Res.	PLLSRC[1:0]	
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw			rw	rw



Bits 1:0 **PLLSRC:** Main PLL, PLLSAI1 and PLLSAI2 entry clock source

Set and cleared by software to select PLL, PLLSAI1 and PLLSAI2 clock source. These bits can be written only when PLL, PLLSAI1 and PLLSAI2 are disabled.

In order to save power, when no PLL is used, the value of PLLSRC should be 00.

00: No clock sent to PLL, PLLSAI1 and PLLSAI2

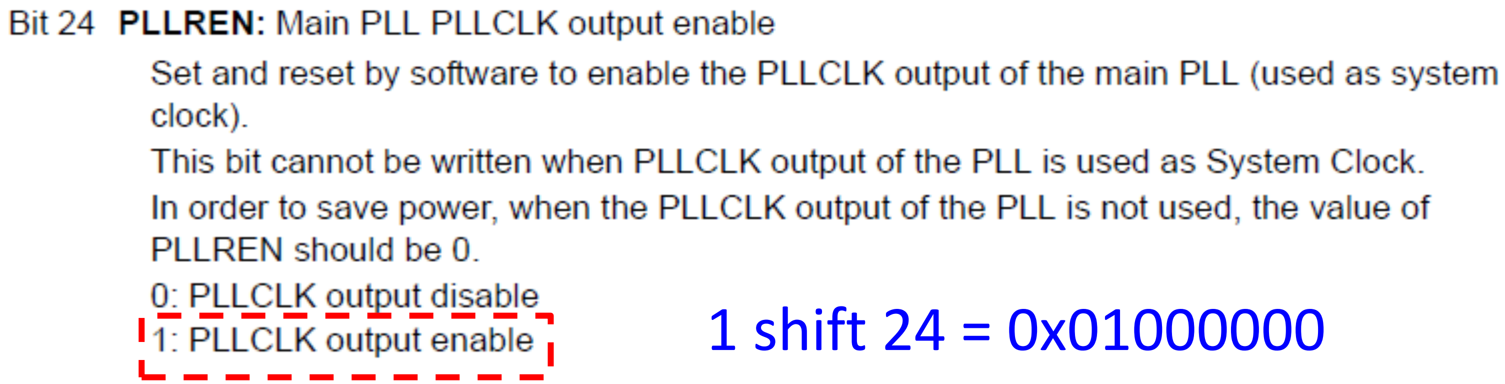
01: MSI clock selected as PLL, PLLSAI1 and PLLSAI2 clock entry

10: HSI16 clock selected as PLL, PLLSAI1 and PLLSAI2 clock entry

11: HSE clock selected as PLL, PLLSAI1 and PLLSAI2 clock entry

RCC_CR 第0,1 bit是PLLSRC

01:把MSI clock當作input拉到PLL去除頻



Bit 24 **PLLREN:** Main PLL PLLCLK output enable

Set and reset by software to enable the PLLCLK output of the main PLL (used as system clock).

This bit cannot be written when PLLCLK output of the PLL is used as System Clock.

In order to save power, when the PLLCLK output of the PLL is not used, the value of PLLREN should be 0.

0: PLLCLK output disable

1: PLLCLK output enable

1 shift 24 = 0x01000000

Bits 6:4 **PLLM:** Division factor for the main PLL and audio PLL (PLLSAI1 and PLLSAI2) input clock

M值控制

Set and cleared by software to divide the PLL, PLLSAI1 and PLLSAI2 input clock before the VCO. These bits can be written only when all PLLs are disabled.

VCO input frequency = PLL input clock frequency / PLLM with 1 <= PLLM <= 8

000: PLLM = 1
001: PLLM = 2
010: PLLM = 3
011: PLLM = 4
100: PLLM = 5
101: PLLM = 6
110: PLLM = 7
111: PLLM = 8

(M+1)

Bits 26:25 **PLLR[1:0]:** Main PLL division factor for PLLCLK (system clock)

R值控制

Set and cleared by software to control the frequency of the main PLL output clock PLLCLK. This output can be selected as system clock. These bits can be written only if PLL is disabled.

PLLCLK output clock frequency = VCO frequency / PLLR with PLLR = 2, 4, 6, or 8

00: PLLR = 2
01: PLLR = 4
10: PLLR = 6
11: PLLR = 8

2 X (R+1)

Bits 14:8 **PLLN[6:0]:** Main PLL multiplication factor for VCO ⬅ N值控制

Set and cleared by software to control the multiplication factor of the VCO. These bits can be written only when the PLL is disabled.

VCO output frequency = VCO input frequency x PLLN with 8 =< PLLN =< 86

0000000: PLLN = 0 wrong configuration

0000001: PLLN = 1 wrong configuration

...

0000111: PLLN = 7 wrong configuration

0001000: PLLN = 8

0001001: PLLN = 9

...

1010101: PLLN = 85

1010110: PLLN = 86

1010111: PLLN = 87 wrong configuration

...

1111111: PLLN = 127 wrong configuration

N

Default LED2

- **User LD2**: the green LED is a user LED connected to Arduino signal D13 corresponding to MCU I/O PA5 (pin 21) or PB13 (pin 34) depending on the STM32 target. Refer to *Table 10* to *Table 21* when:
  - the I/O is HIGH value, the LED is on
  - the I/O is LOW, the LED is off .

STM32 Nucleo-64 boards P22

Push buttons

- **B1 USER**: the user button is connected to the I/O PC13 (pin 2) of the STM32 microcontroller.
- **B2 RESET**: this push button is connected to NRST, and is used to RESET the STM32 microcontroller.
- *Note: The blue and black plastic hats that are placed on the push buttons can be removed if necessary, for example when a shield or when an application board is plugged on top of the Nucleo board. This will avoid pressure on the buttons and consequently a possible permanent target MCU RESET.*

STM32 Nucleo-64 boards P22

FPU_init()

Coprocessor access control register (CPACR) for FPU_init()

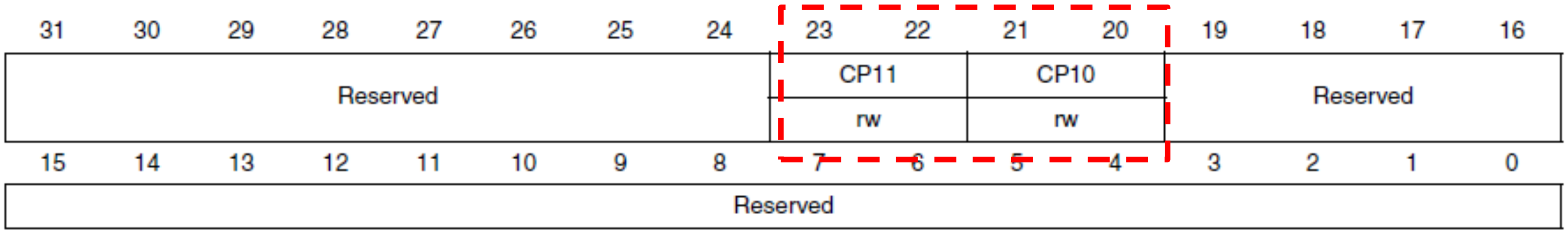
4.6.1 Coprocessor access control register (CPACR)

Address offset (from SCB): 0x88

Reset value: 0x0000000

Required privilege: Privileged

The CPACR register specifies the access privileges for coprocessors.



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								CP11		CP10		Reserved			
								rw		rw					
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved															

Bits 31:24 Reserved. Read as Zero, Write Ignore.

Bits 23:20 **CPn:** [2n+1:2n] for n values 10 and 11. Access privileges for coprocessor n. The possible values of each field are:

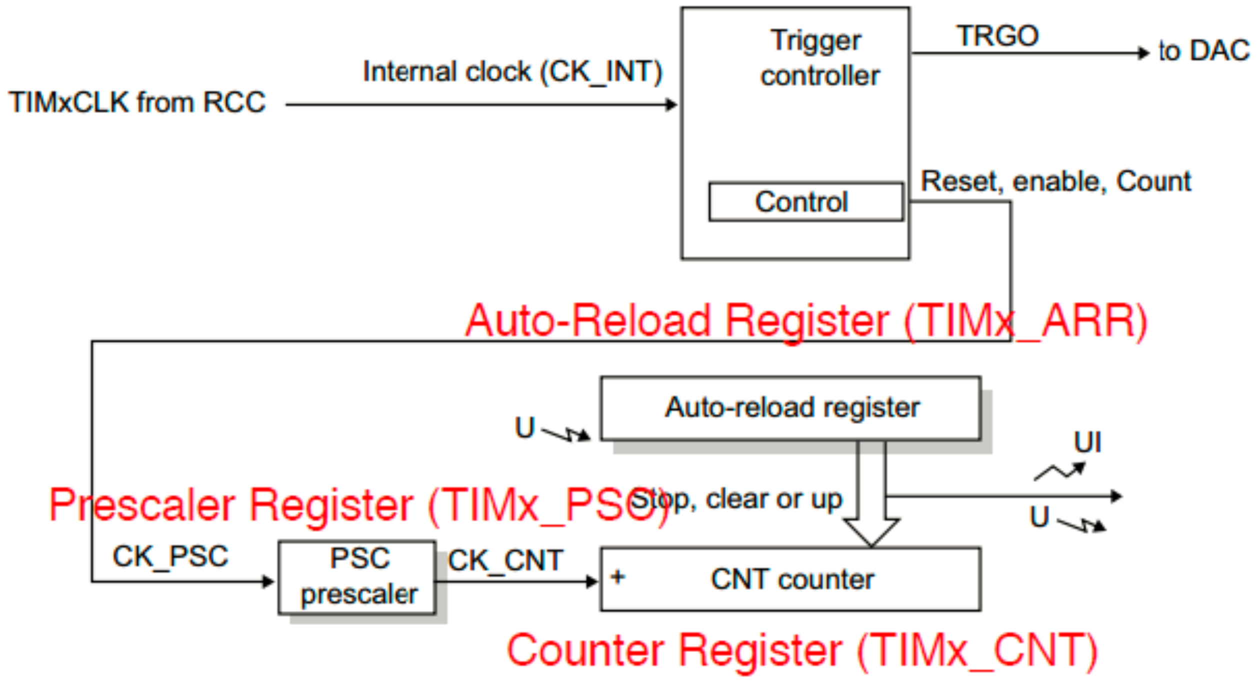
- 0b00: Access denied. Any attempted access generates a NOCP UsageFault.
- 0b01: Privileged access only. An unprivileged access generates a NOCP fault.
- 0b10: Reserved. The result of any access is Unpredictable.
- 0b11: Full access.

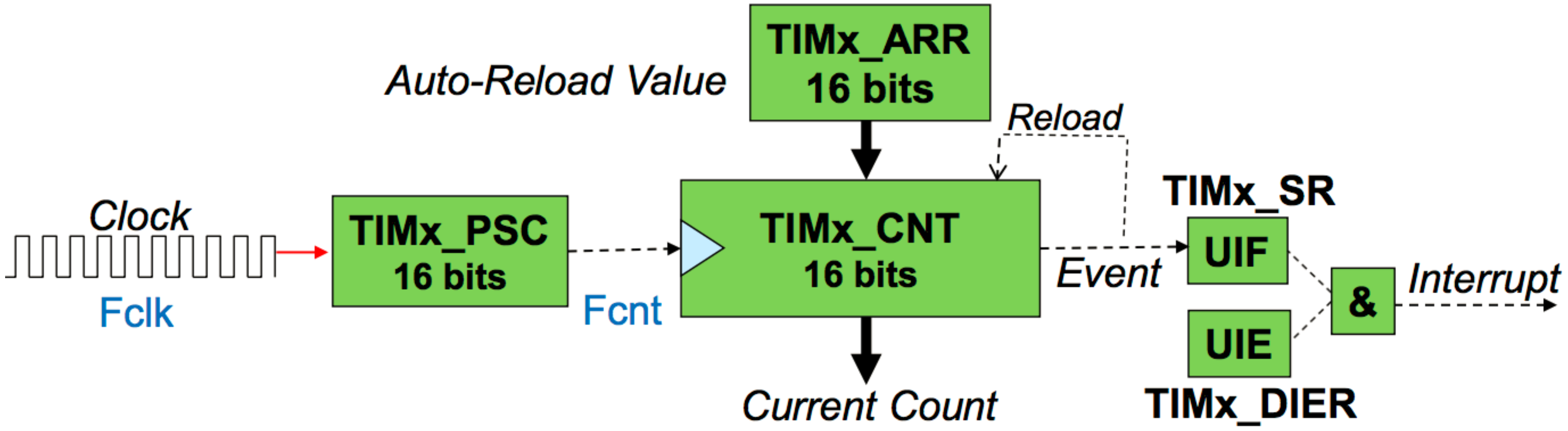
Bits 19:0 Reserved. Read as Zero, Write Ignore.

- **System control block (SCB) -** The System control block (SCB) provides system implementation information, and system control. This includes configuration, control, and reporting of the system exceptions. P220

Timer (Lab5.2)

$$\begin{tikzcd}
\boxed{\text{Enable timer}} \arrow[d] \\
\boxed{\text{Set PSC/CNT/ARR}} \arrow[d] \\
\boxed{\text{Enable counter to start}}
\end{tikzcd}$$





- Count-up mode overflow if **TIMx_CNT** reaches **TIMx_ARR**
  - On "overflow event", **UIF** flag is set and TIMx_CNT resets to 0.
  - If **UIE** = 1 (update interrupt enabled), interrupt signal is sent to NVIC
- **TIMx_PSC** prescale value multiplies input clock period (1/ Fclk) to produce counter clock period: Tcnt = 1/Fcnt = (PSC+1)×(1/Fclk)
- Periodic time interval is the ARR (Auto-Reload Register) value times counter clock period:

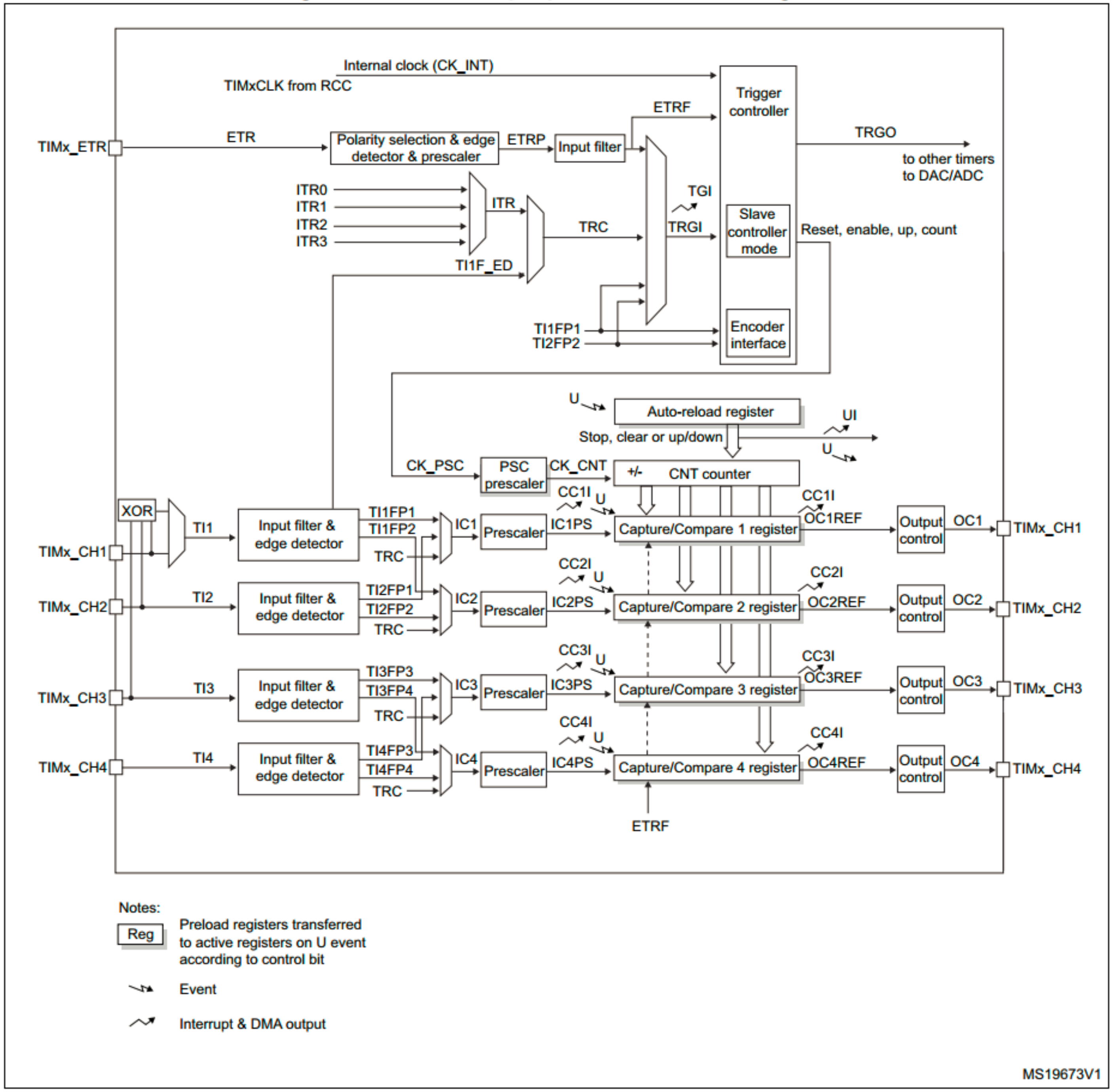
  Tout = (ARR+1)×Tcnt = (ARR+1)×(PSC+1)×(1/Fclk)

Example: For 1 second time period, given Fclk = 16MHz:

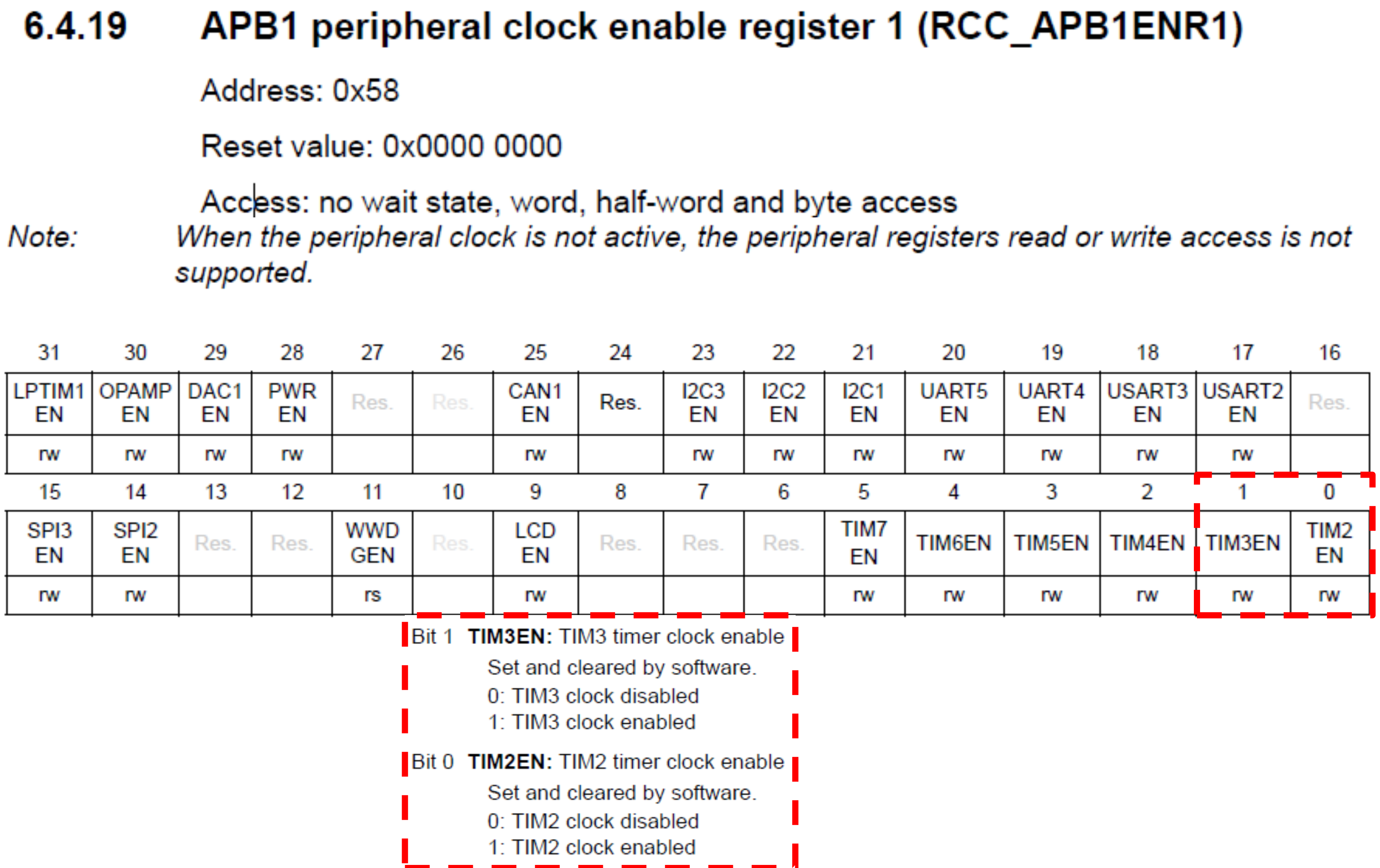
Tout = (10000 × 1600) ÷ 16000000 = 1 second

Set ARR = 9999 and PSC = 1599 *(other combinations can also be used)*

Figure 249. General-purpose timer block diagram



APB1 peripheral clock enable register 1 (RCC_APB1ENR1)



6.4.19 APB1 peripheral clock enable register 1 (RCC_APB1ENR1)

Address: 0x58

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access

Note: When the peripheral clock is not active, the peripheral registers read or write access is not supported.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LPTIM1 EN	OPAMP EN	DAC1 EN	PWR EN	Res.	Res.	CAN1 EN	Res.	I2C3 EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART3 EN	USART2 EN	Res.
rw	rw	rw	rw			rw		rw	rw	rw	rw	rw	rw	rw	

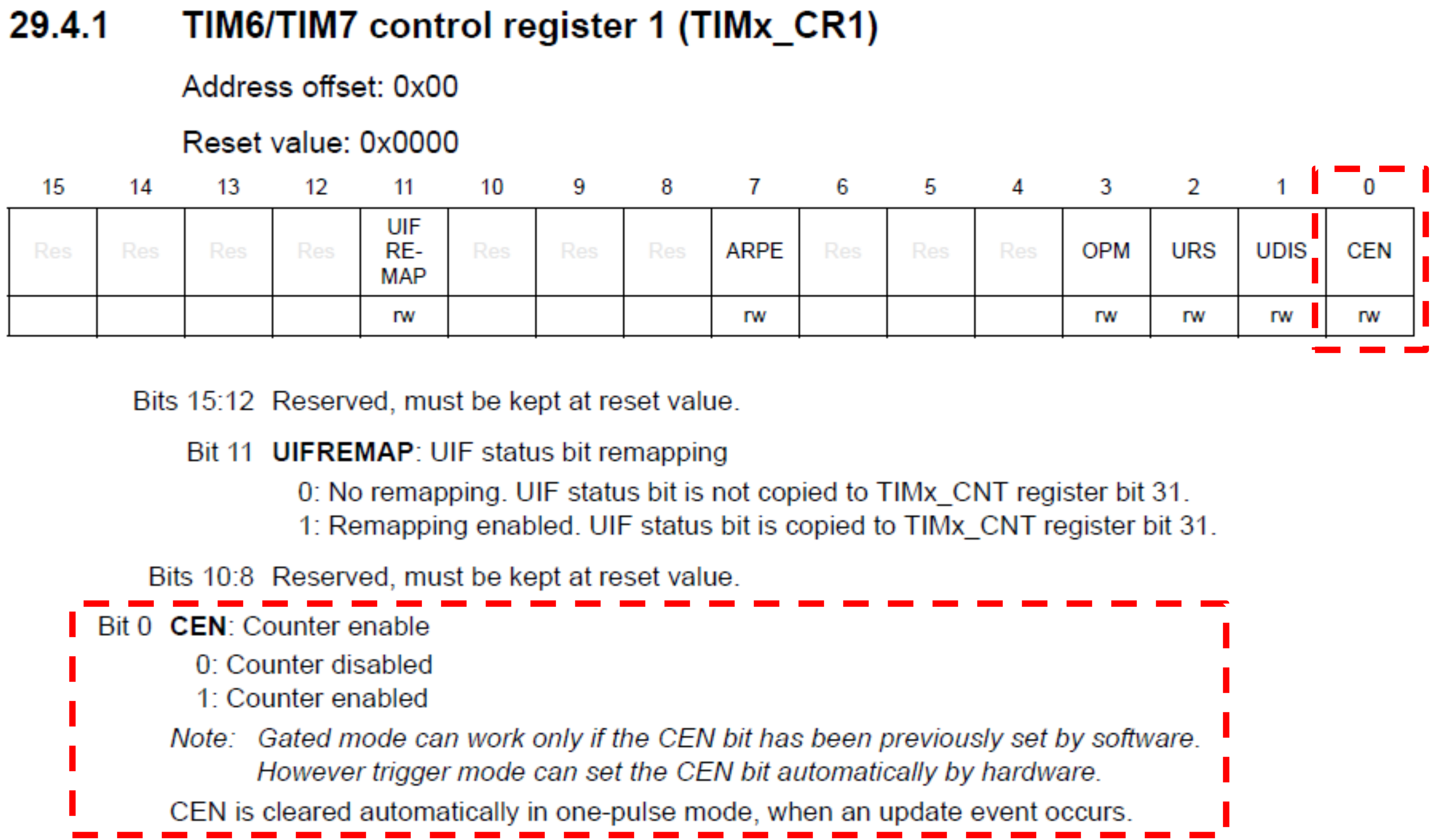
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Res.	Res.	WWD GEN	Res.	LCD EN	Res.	Res.	Res.	TIM7 EN	TIM6EN	TIM5EN	TIM4EN	TIM3EN	TIM2 EN
rw	rw			rs		rw				rw	rw	rw	rw	rw	rw

Bit 1 **TIM3EN:** TIM3 timer clock enable
Set and cleared by software.
0: TIM3 clock disabled
1: TIM3 clock enabled

Bit 0 **TIM2EN:** TIM2 timer clock enable
Set and cleared by software.
0: TIM2 clock disabled
1: TIM2 clock enabled

TIMx Control Register

- P904



29.4.1 TIM6/TIM7 control register 1 (TIMx_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	Res	Res	Res	UIF RE-MAP	Res	Res	Res	ARPE	Res	Res	Res	OPM	URS	UDIS	CEN
				rw				rw				rw	rw	rw	rw

Bits 15:12 Reserved, must be kept at reset value.

Bit 11 **UIFREMAP**: UIF status bit remapping

0: No remapping. UIF status bit is not copied to TIMx_CNT register bit 31.
1: Remapping enabled. UIF status bit is copied to TIMx_CNT register bit 31.

Bits 10:8 Reserved, must be kept at reset value.

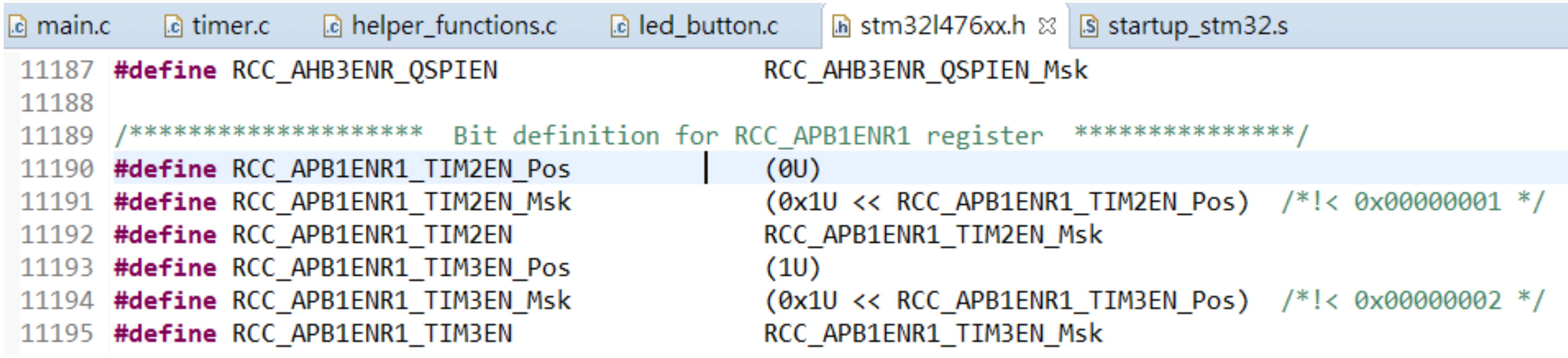
Bit 0 **CEN**: Counter enable

0: Counter disabled
1: Counter enabled

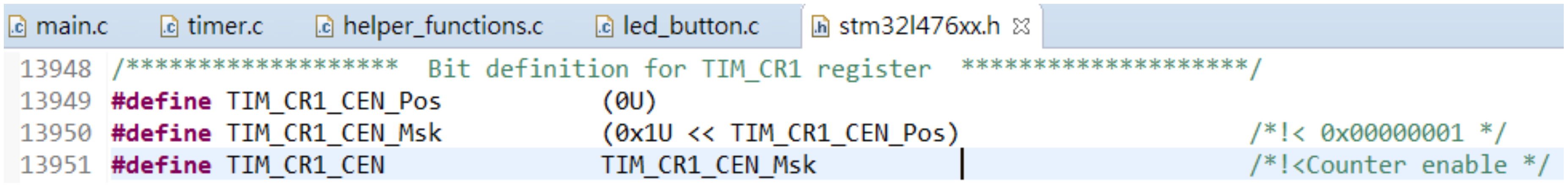
Note: Gated mode can work only if the CEN bit has been previously set by software. However trigger mode can set the CEN bit automatically by hardware.

CEN is cleared automatically in one-pulse mode, when an update event occurs.

Timer/Clock Enable



```c
#define RCC_AHB3ENR_QSPIEN                 RCC_AHB3ENR_QSPIEN_Msk

/******************  Bit definition for RCC_APB1ENR1 register  ***************/
#define RCC_APB1ENR1_TIM2EN_Pos            (0U)
#define RCC_APB1ENR1_TIM2EN_Msk            (0x1U << RCC_APB1ENR1_TIM2EN_Pos)  /*!< 0x00000001 */
#define RCC_APB1ENR1_TIM2EN                RCC_APB1ENR1_TIM2EN_Msk
#define RCC_APB1ENR1_TIM3EN_Pos            (1U)
#define RCC_APB1ENR1_TIM3EN_Msk            (0x1U << RCC_APB1ENR1_TIM3EN_Pos)  /*!< 0x00000002 */
#define RCC_APB1ENR1_TIM3EN                RCC_APB1ENR1_TIM3EN_Msk
```

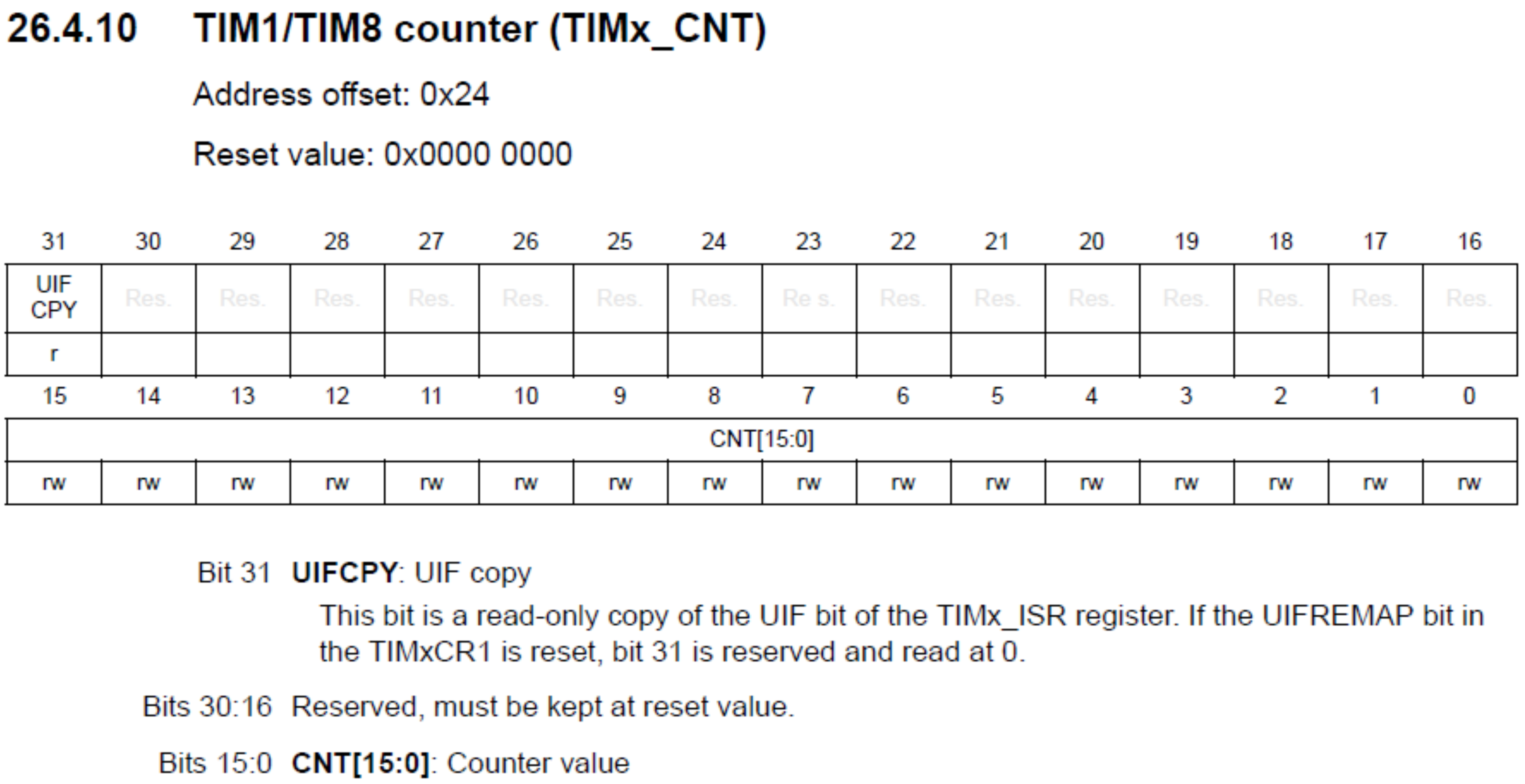


```c
/*******************  Bit definition for TIM_CR1 register  ********************/
#define TIM_CR1_CEN_Pos           (0U)
#define TIM_CR1_CEN_Msk           (0x1U << TIM_CR1_CEN_Pos)                     /*!< 0x00000001 */
#define TIM_CR1_CEN               TIM_CR1_CEN_Msk                               /*!<Counter enable */
```

- Counter Register (TIMx_CNT)
- Prescaler Register (TIMx_PSC)
- Auto-Reload Register (TIMx_ARR)

TIMx_CNT

P834, counter



26.4.10 TIM1/TIM8 counter (TIMx_CNT)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UIF CPY	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Re s.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **UIFCPY**: UIF copy

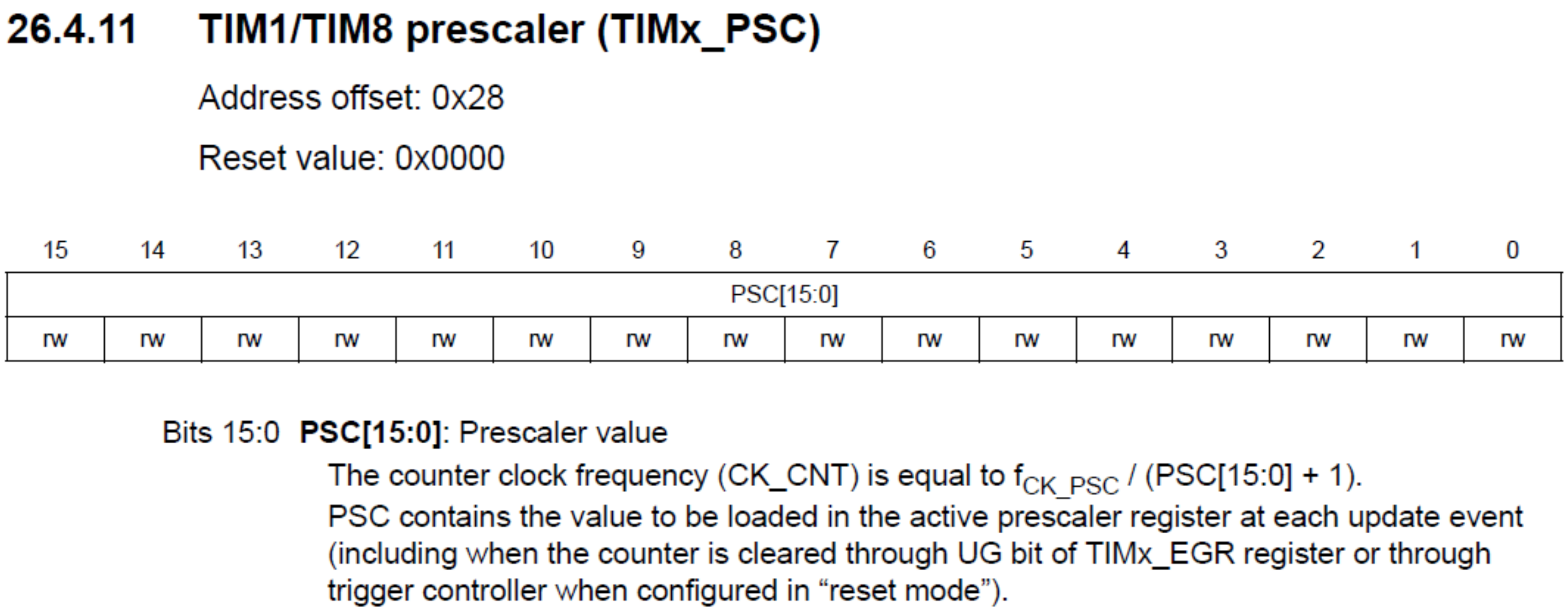
This bit is a read-only copy of the UIF bit of the TIMx_ISR register. If the UIFREMAP bit in the TIMxCR1 is reset, bit 31 is reserved and read at 0.

Bits 30:16 Reserved, must be kept at reset value.

Bits 15:0 **CNT[15:0]**: Counter value

TIMx_PSC

除頻



26.4.11 TIM1/TIM8 prescaler (TIMx_PSC)

Address offset: 0x28

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

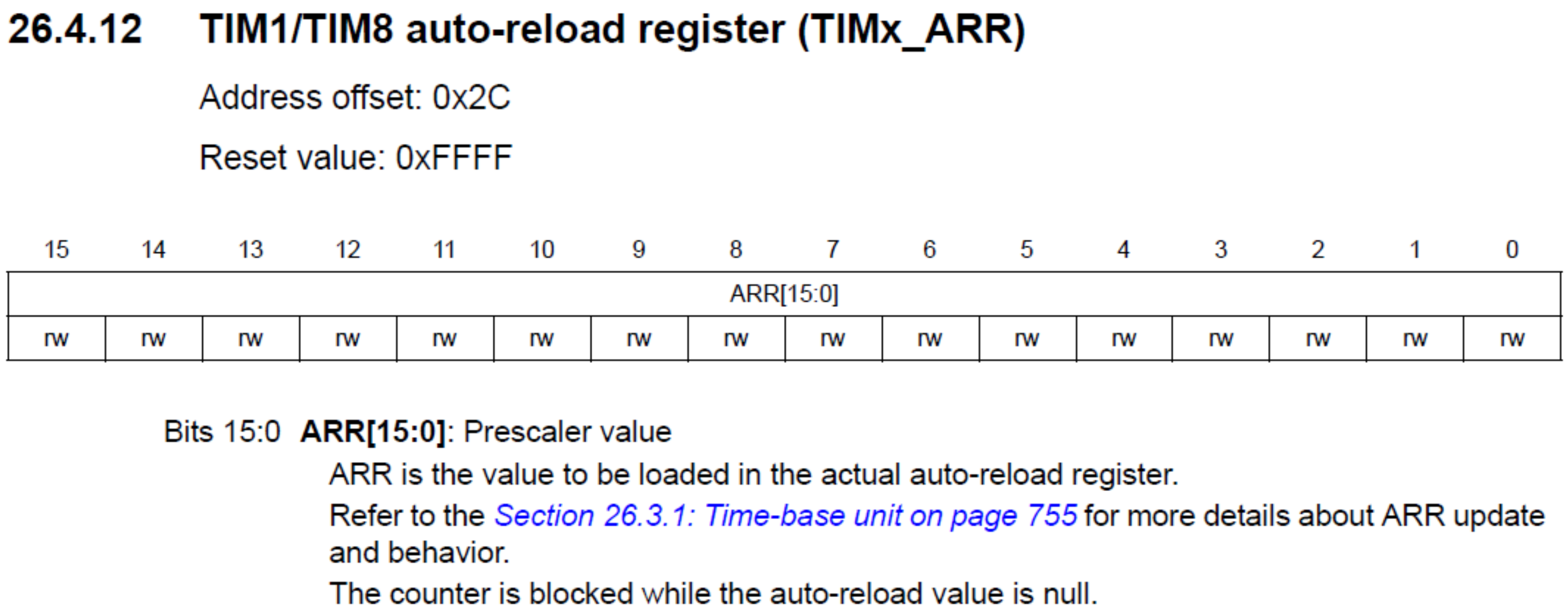
Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency (CK_CNT) is equal to f_{CK_PSC} / (PSC[15:0] + 1).

PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx_EGR register or through trigger controller when configured in “reset mode”).

TIMx_ARR

計數到多少



26.4.12 TIM1/TIM8 auto-reload register (TIMx_ARR)

Address offset: 0x2C

Reset value: 0xFFFF

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

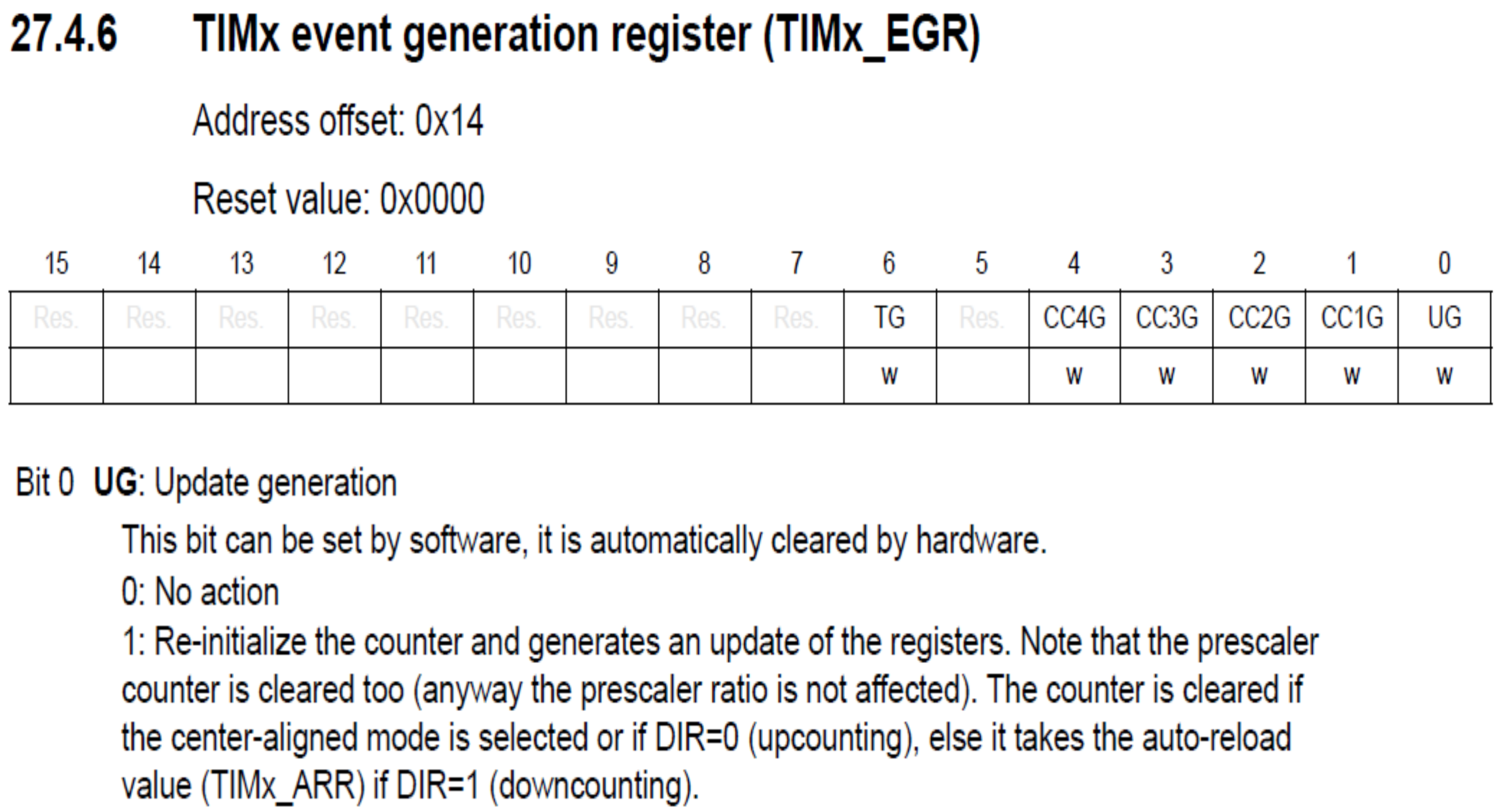
Bits 15:0 **ARR[15:0]**: Prescaler value

ARR is the value to be loaded in the actual auto-reload register.

Refer to the *Section 26.3.1: Time-base unit on page 755* for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

TIMx_EGR



27.4.6 TIMx event generation register (TIMx_EGR)

Address offset: 0x14

Reset value: 0x0000

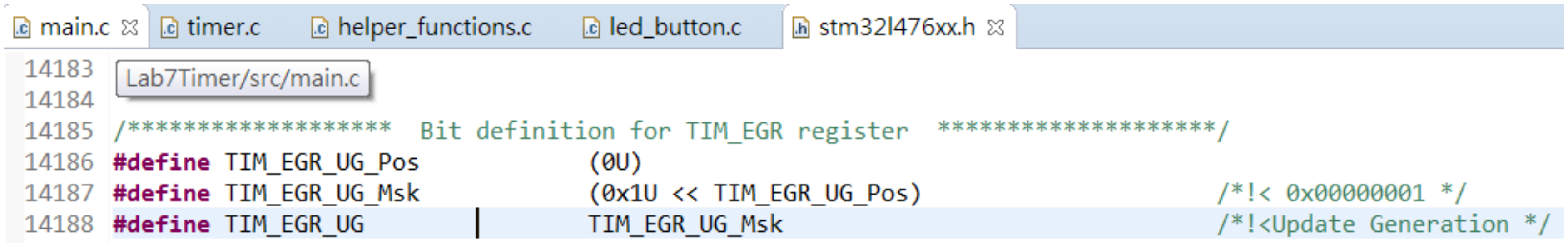
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TG	Res.	CC4G	CC3G	CC2G	CC1G	UG
									w		w	w	w	w	w

Bit 0 **UG**: Update generation

This bit can be set by software, it is automatically cleared by hardware.

0: No action

1: Re-initialize the counter and generates an update of the registers. Note that the prescaler counter is cleared too (anyway the prescaler ratio is not affected). The counter is cleared if the center-aligned mode is selected or if DIR=0 (upcounting), else it takes the auto-reload value (TIMx_ARR) if DIR=1 (downcounting).



```
/******************  Bit definition for TIM_EGR register  ********************/
#define TIM_EGR_UG_Pos            (0U)
#define TIM_EGR_UG_Msk            (0x1U << TIM_EGR_UG_Pos)                    /*!< 0x00000001 */
#define TIM_EGR_UG                TIM_EGR_UG_Msk                              /*!<Update Generation */
```

Music Keypad (Lab5.3)

表 6-10　音階頻率表(單位：Hz)

八度音	DO	DO#	RE	RE#	MI	FA	FA#	SO	SO#	LA	LA#	SI
第 0 度	65	69	73	78	82	87	93	98	104	110	116	123
第 1 度	131	139	147	156	165	175	185	196	208	220	233	247
第 2 度	262	277	294	311	330	349	370	392	415	440	466	494
第 3 度	523	554	587	622	659	698	740	784	831	880	932	988
第 4 度	1046	1109	1175	1245	1318	1397	1480	1568	1661	1760	1865	1976
第 5 度	2093	2217	2349	2489	2637	2794	2960	3136	3322	3520	3729	3951
第 6 度	4186	4435	4699	4978	5274	5587	5919	6271	6645	7040	7459	7902

Lab 5. Clock and Timer 範例程式

Code provided by NCTU CS 謝明恩 吳赫倫
Slide made by NCTU ME 助教 林穎毅
20170418

Lab5.1 Modify system initial clock(20%)

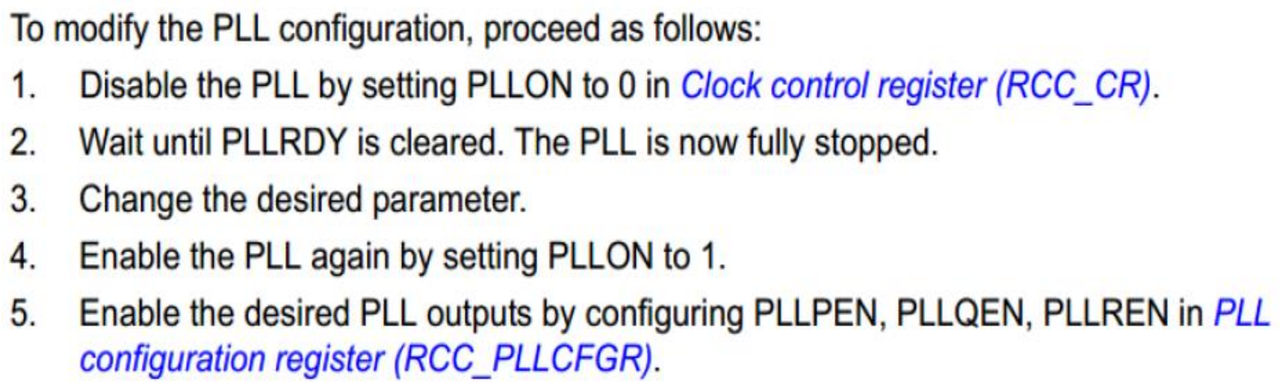
- 修改SYSCLK的clock source以及相關的prescaler使得CPU frequency(HCLK)為1MHz。觀察修改前後LED燈閃爍的頻率。
- 當使用者按下user button便依以下順序改變CPU system clock(HCLK)， 1MHz -> 6MHz -> 10MHz ->16MHz -> 40MHz ->1MHz ->…

Lab5.1 Modify system initial clock(20%)

```
main.c
extern void GPIO_init();
extern void delay_1s();
void SystemClock_Config(){
   //TODO: Change the SYSCLK source and set the corresponding
Prescaler value.
}

int main(){
  SystemClock_Config();
  GPIO_init();
  while(1){
    if (user_press_button())
    {
      //TODO: Update system clock rate
    }
    GPIOA->BSRR = (1<<5);
    delay_1s();
    GPIOA->BRR = (1<<5);
    delay_1s();
  }
}
```

Lab5.1 Modify system initial clock(20%)

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

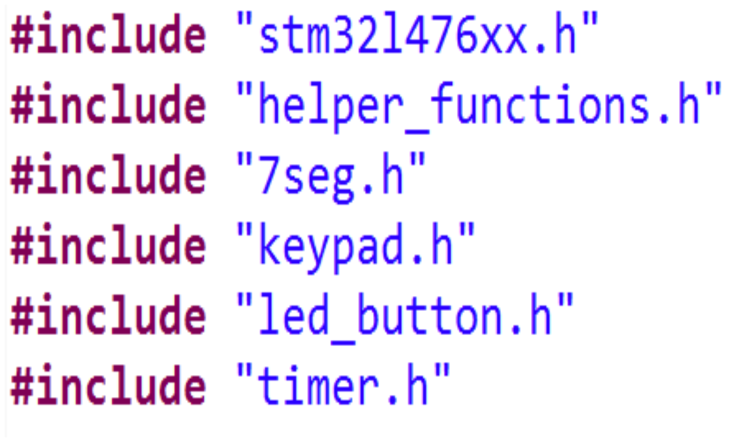


To modify the PLL configuration, proceed as follows:

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

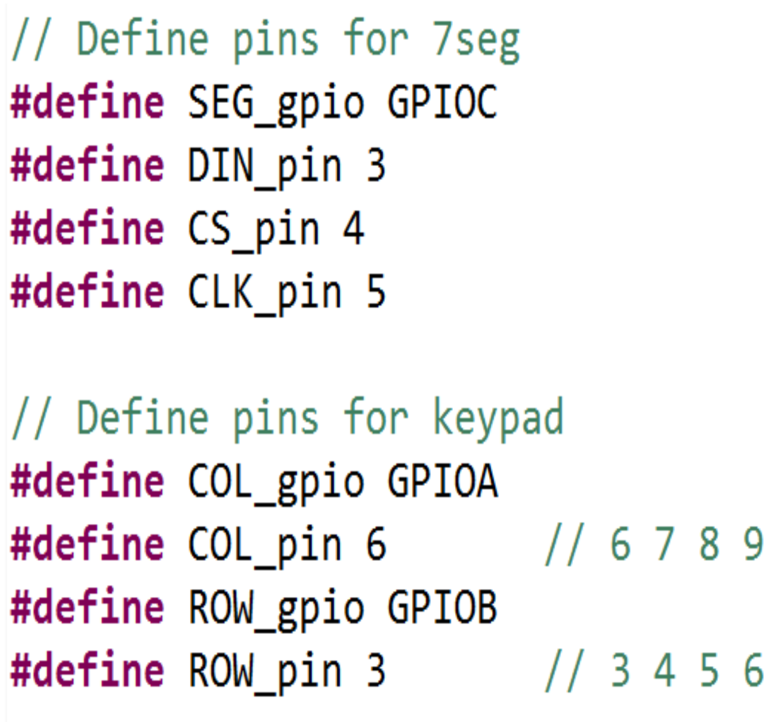
- 其中PLL clock頻率計算為f(VCO clock) = f(PLL clock input) × (PLLN / PLLM)
- 最終可輸出給system clock頻率為f(PLL_R) = f(VCO clock) / PLLR

include header file



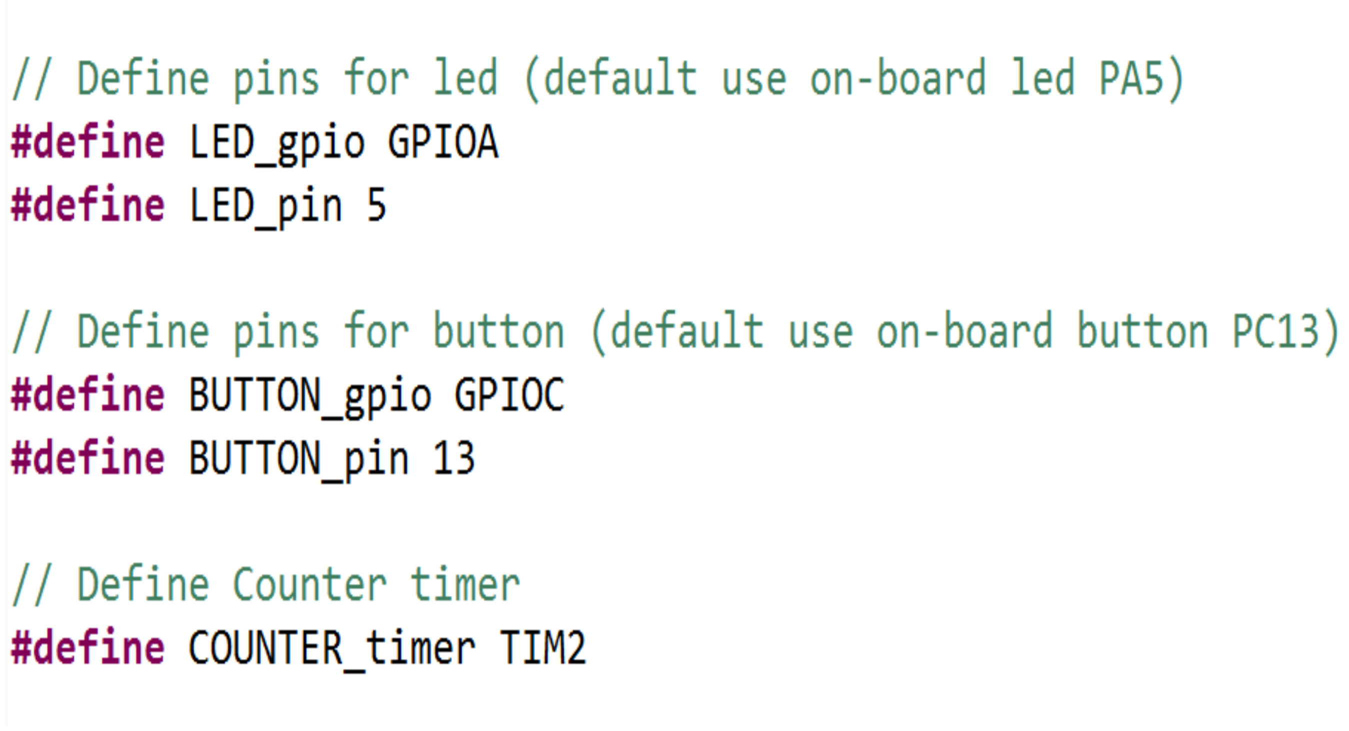
```
#include "stm32l476xx.h"
#include "helper_functions.h"
#include "7seg.h"
#include "keypad.h"
#include "led_button.h"
#include "timer.h"
```

設定Pin角跟Timer



```
// Define pins for 7seg
#define SEG_gpio GPIOC
#define DIN_pin 3
#define CS_pin 4
#define CLK_pin 5

// Define pins for keypad
#define COL_gpio GPIOA
#define COL_pin 6       // 6 7 8 9
#define ROW_gpio GPIOB
#define ROW_pin 3       // 3 4 5 6
```

設定Pin角跟Timer



```c
// Define pins for led (default use on-board led PA5)
#define LED_gpio GPIOA
#define LED_pin 5

// Define pins for button (default use on-board button PC13)
#define BUTTON_gpio GPIOC
#define BUTTON_pin 13

// Define Counter timer
#define COUNTER_timer TIM2
```

main

```
FPU_init();
```

啟用浮點數的功能

```
if(init_led(LED_gpio, LED_pin) != 0){
    // Fail to init led
    return -1;
}
if(init_button(BUTTON_gpio, BUTTON_pin) != 0){
    // Fail to init button
    return -1;
}
```

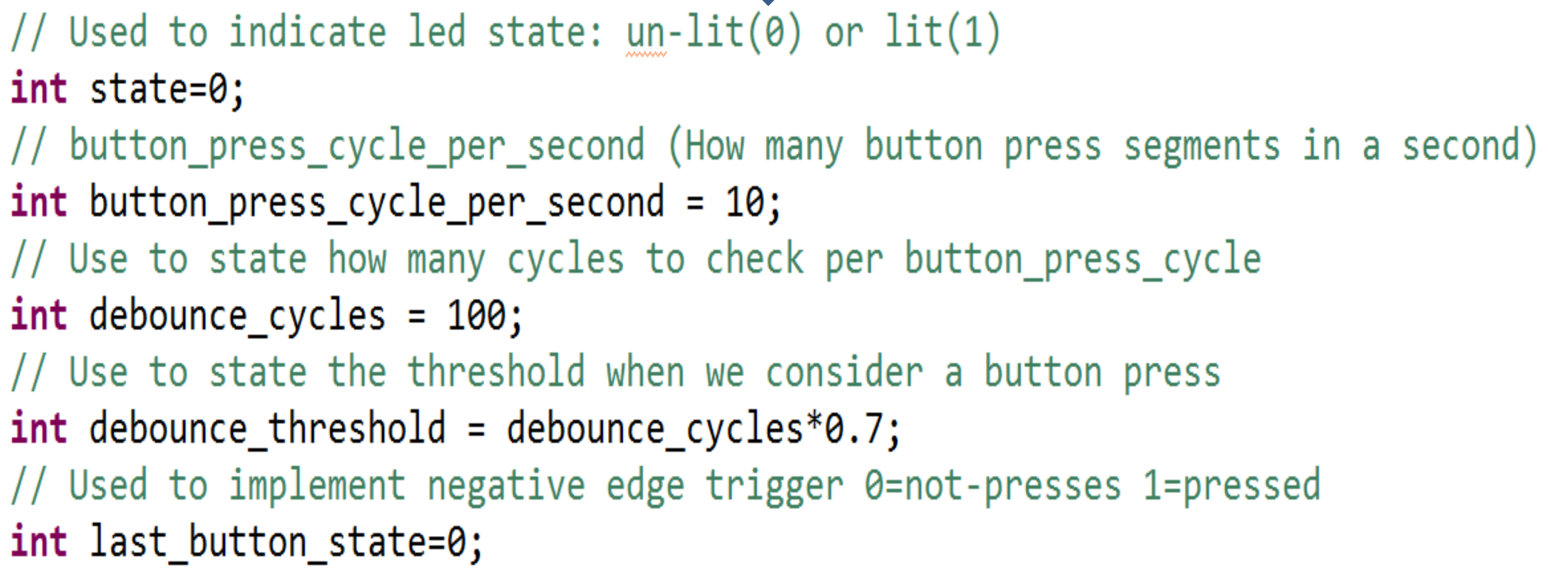
初始化需要用到的pin角

```
int speed=0, trans[5]={1, 6, 10, 16, 40};
SystemClock_Config(trans[speed]);
```

設定timer數的速度

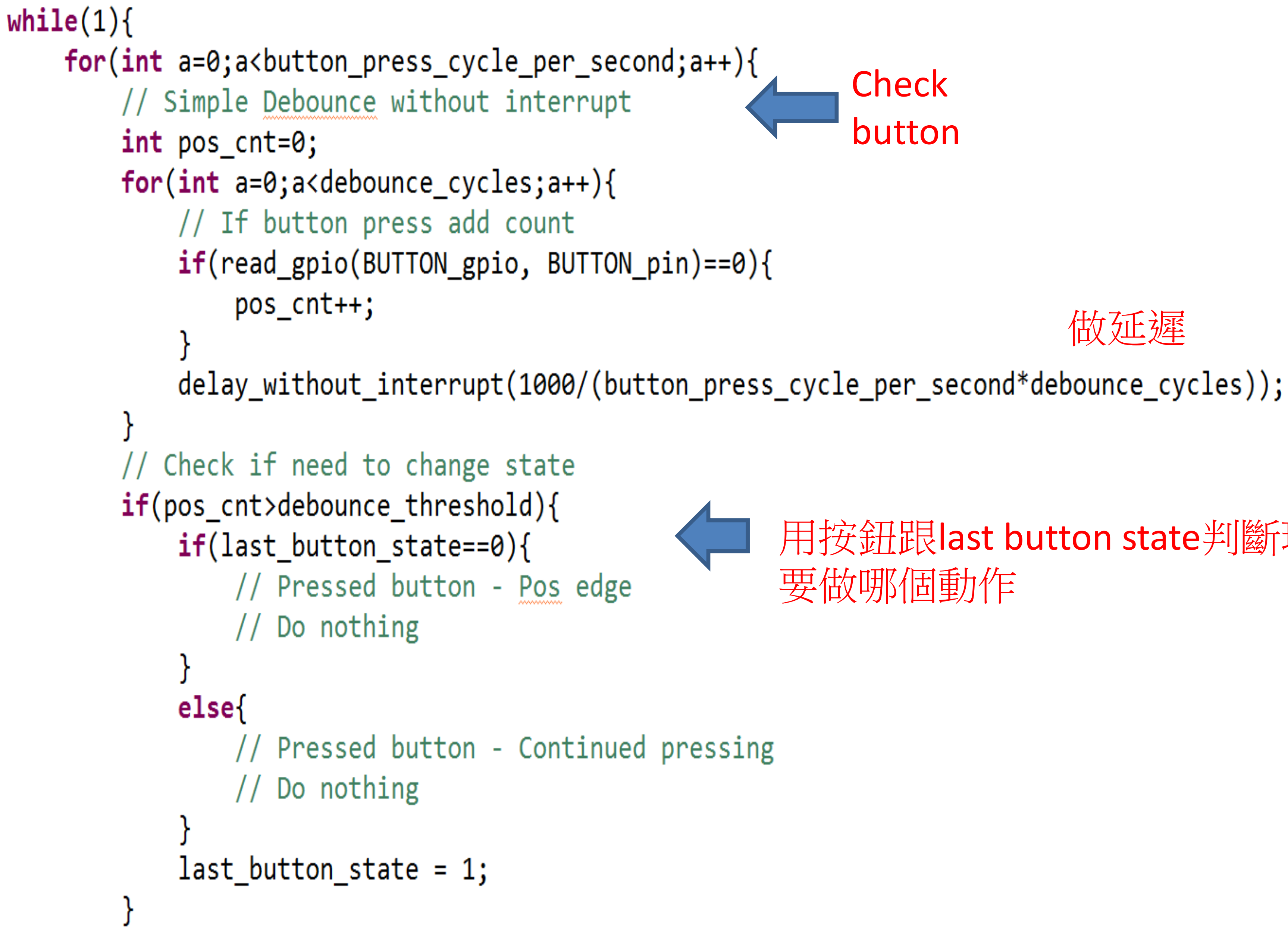
main

設定需要使用
的參數



```
// Used to indicate led state: un-lit(0) or lit(1)
int state=0;
// button_press_cycle_per_second (How many button press segments in a second)
int button_press_cycle_per_second = 10;
// Use to state how many cycles to check per button_press_cycle
int debounce_cycles = 100;
// Use to state the threshold when we consider a button press
int debounce_threshold = debounce_cycles*0.7;
// Used to implement negative edge trigger 0=not-presses 1=pressed
int last_button_state=0;
```

main



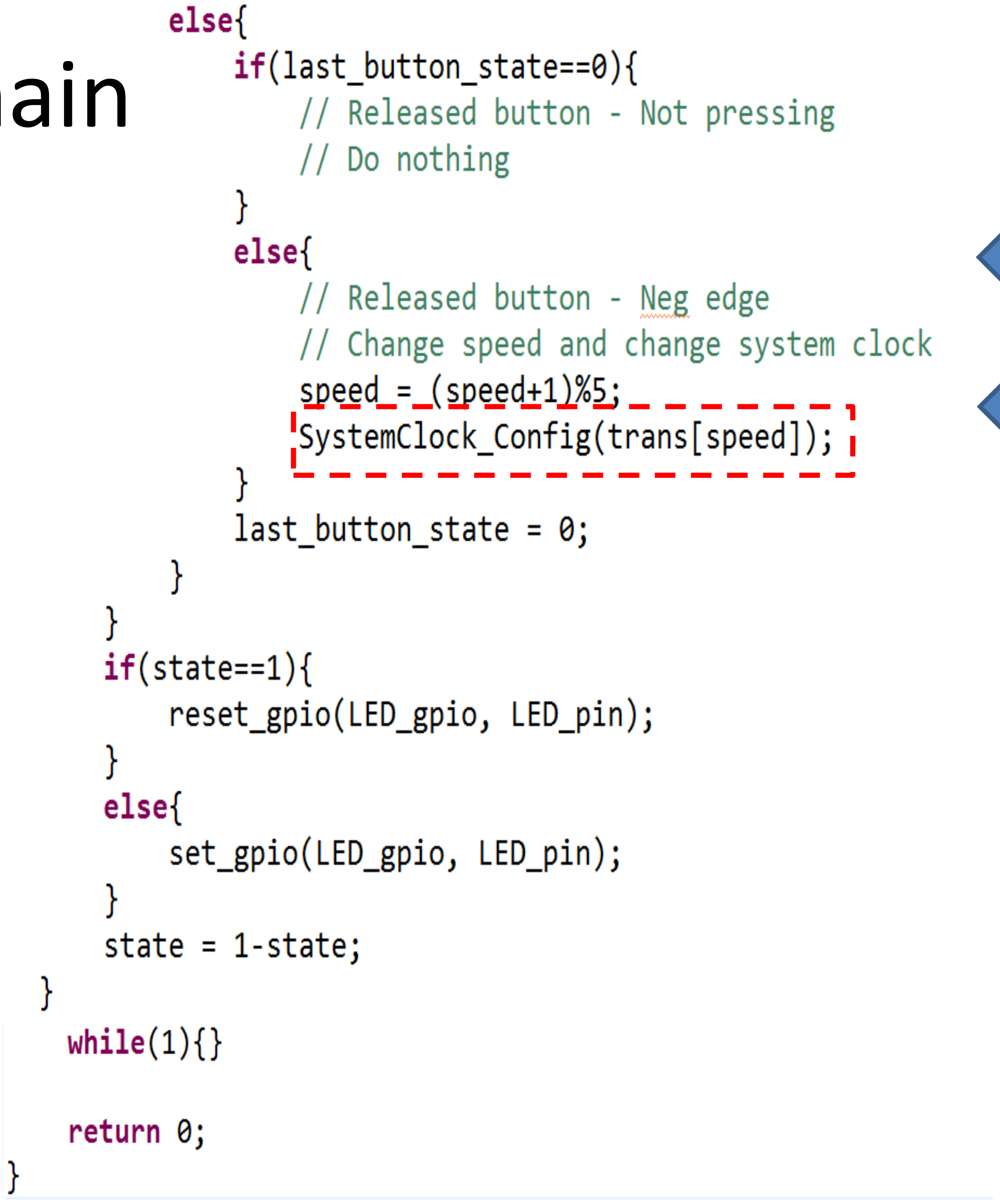
```
while(1){
    for(int a=0;a<button_press_cycle_per_second;a++){
        // Simple Debounce without interrupt
        int pos_cnt=0;
        for(int a=0;a<debounce_cycles;a++){
            // If button press add count
            if(read_gpio(BUTTON_gpio, BUTTON_pin)==0){
                pos_cnt++;
            }
            delay_without_interrupt(1000/(button_press_cycle_per_second*debounce_cycles));
        }
        // Check if need to change state
        if(pos_cnt>debounce_threshold){
            if(last_button_state==0){
                // Pressed button - Pos edge
                // Do nothing
            }
            else{
                // Pressed button - Continued pressing
                // Do nothing
            }
            last_button_state = 1;
        }
```

Check button

做延遲

用按鈕跟last button state判斷現在要做哪個動作

main

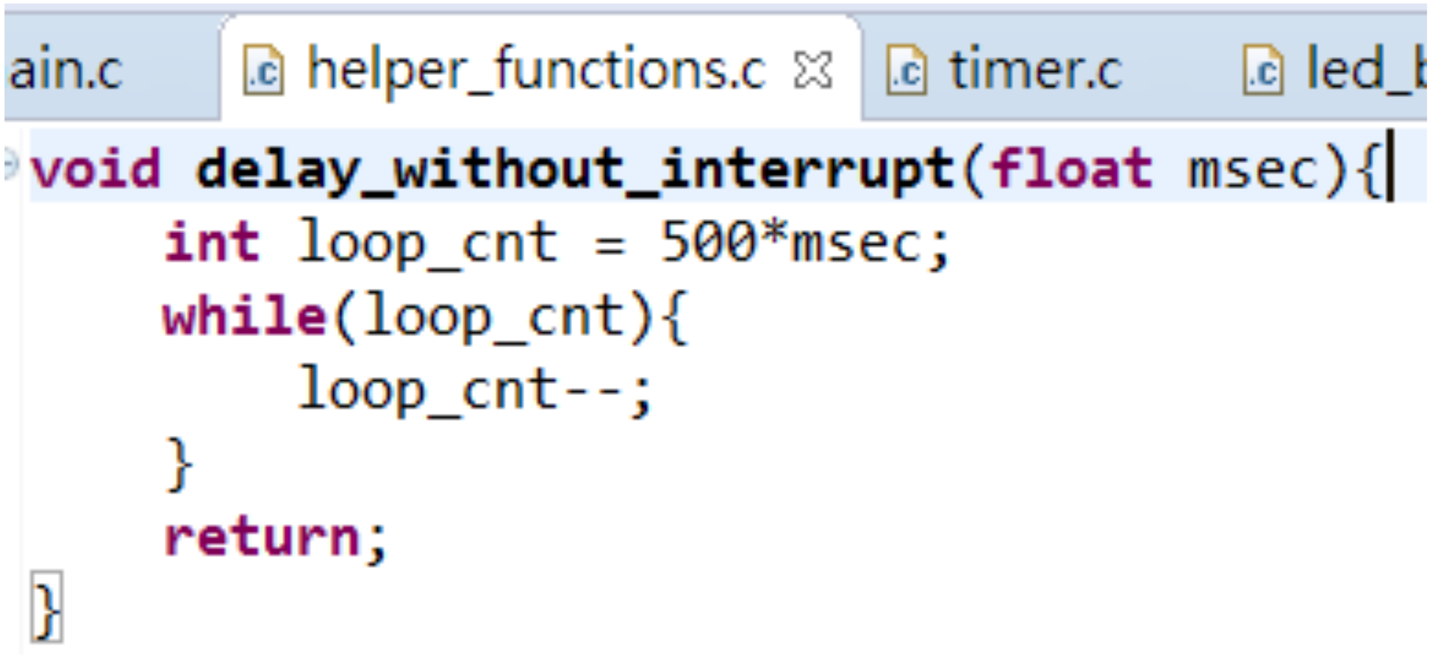


```
         else{
             if(last_button_state==0){
                 // Released button - Not pressing
                 // Do nothing
             }
             else{
                 // Released button - Neg edge
                 // Change speed and change system clock
                 speed = (speed+1)%5;
                 SystemClock_Config(trans[speed]);
             }
             last_button_state = 0;
         }
     }
     if(state==1){
         reset_gpio(LED_gpio, LED_pin);
     }
     else{
         set_gpio(LED_gpio, LED_pin);
     }
     state = 1-state;
 }
  while(1){}

  return 0;
}
```

一樣我們只在意1到0的情形

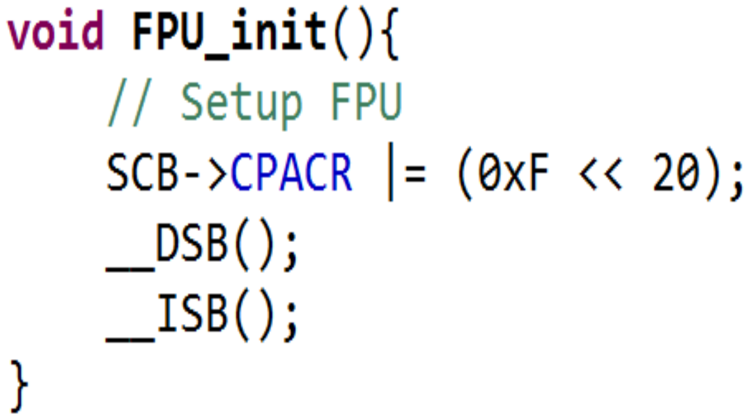
改變閃爍速度

delay_without_interrupt



```c
void delay_without_interrupt(float msec){
    int loop_cnt = 500*msec;
    while(loop_cnt){
        loop_cnt--;
    }
    return;
}
```

FPU_init

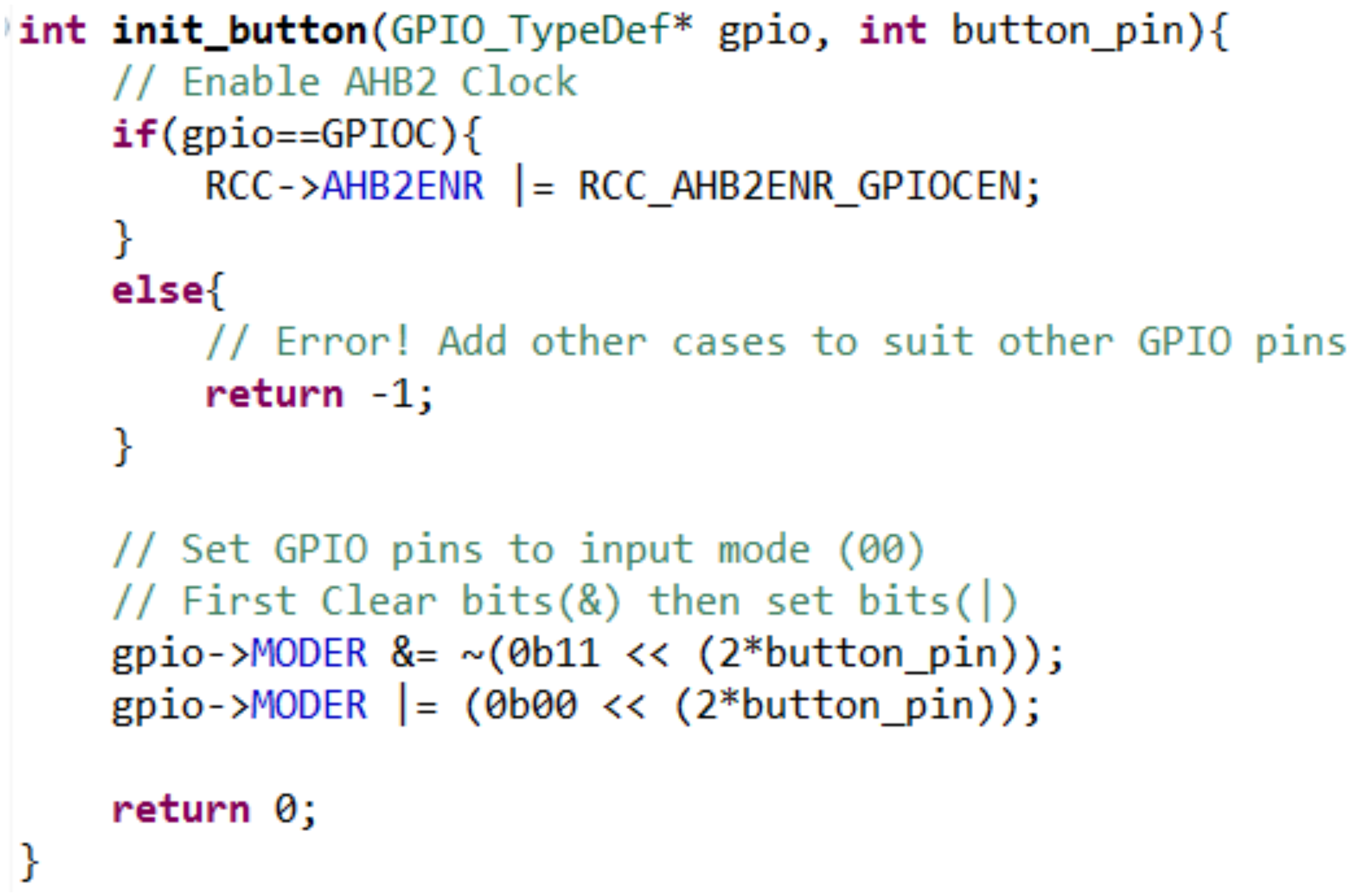


```
void FPU_init(){
    // Setup FPU
    SCB->CPACR |= (0xF << 20);
    __DSB();
    __ISB();
}
```

init_led

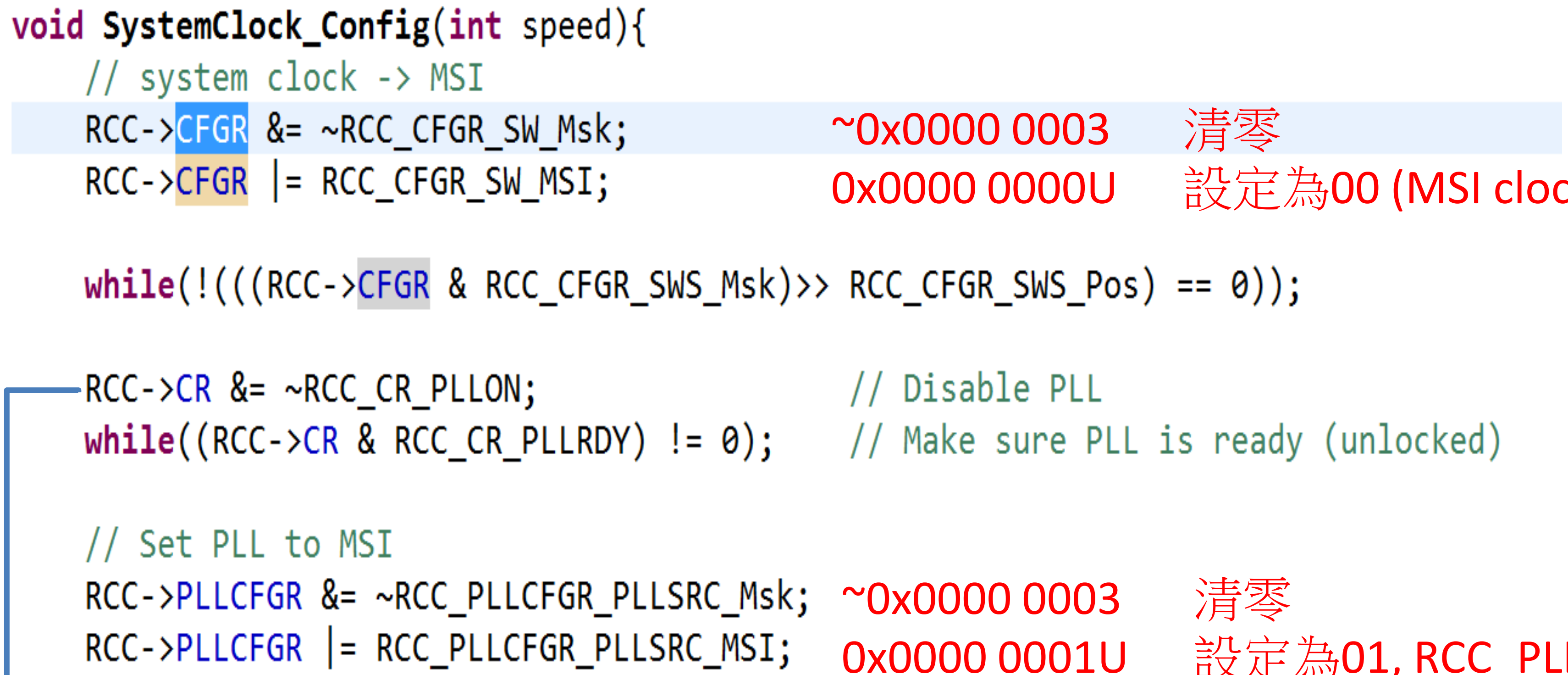
```c
int init_led(GPIO_TypeDef* gpio, int LED_pin){
    // Enable AHB2 Clock
    if(gpio==GPIOA){
        RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;
    }
    else if(gpio==GPIOB){
        RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
    }
    else{
        // Error! Add other cases to suit other GPIO pins
        return -1;
    }

    // Set GPIO pins to output mode (01)
    // First Clear bits(&) then set bits(|)
    gpio->MODER &= ~(0b11 << (2*LED_pin));
    gpio->MODER |= (0b01 << (2*LED_pin));

    return 0;
}
```

init_button



```
int init_button(GPIO_TypeDef* gpio, int button_pin){
    // Enable AHB2 Clock
    if(gpio==GPIOC){
        RCC->AHB2ENR |= RCC_AHB2ENR_GPIOCEN;
    }
    else{
        // Error! Add other cases to suit other GPIO pins
        return -1;
    }

    // Set GPIO pins to input mode (00)
    // First Clear bits(&) then set bits(|)
    gpio->MODER &= ~(0b11 << (2*button_pin));
    gpio->MODER |= (0b00 << (2*button_pin));

    return 0;
}
```

SystemClock_Config



```
void SystemClock_Config(int speed){
    // system clock -> MSI
    RCC->CFGR &= ~RCC_CFGR_SW_Msk;
    RCC->CFGR |= RCC_CFGR_SW_MSI;

    while(!(((RCC->CFGR & RCC_CFGR_SWS_Msk)>> RCC_CFGR_SWS_Pos) == 0));

    RCC->CR &= ~RCC_CR_PLLON;                  // Disable PLL
    while((RCC->CR & RCC_CR_PLLRDY) != 0);     // Make sure PLL is ready (unlocked)

    // Set PLL to MSI
    RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLSRC_Msk;
    RCC->PLLCFGR |= RCC_PLLCFGR_PLLSRC_MSI;
```

~0x0000 0003　　清零

0x0000 0000U　　設定為00 (MSI clock)

~0x0000 0003　　清零

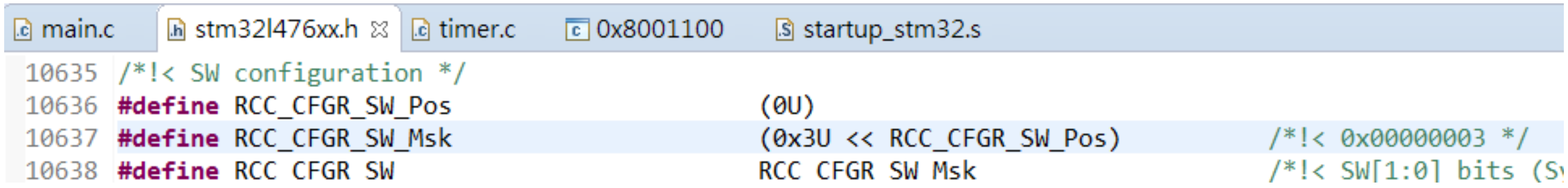
0x0000 0001U　　設定為01, RCC_PLLFGR 第0,1 bit是PLLSRC, 01:把MSI clock當作input拉到PLL去除頻

~0x0100 0000

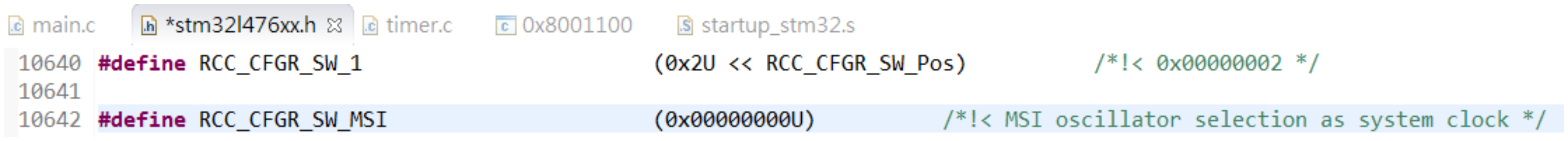
RCC_CR 第24bit是PLLON 0 off 1on

因為調整參數必須在PLL off的情形下

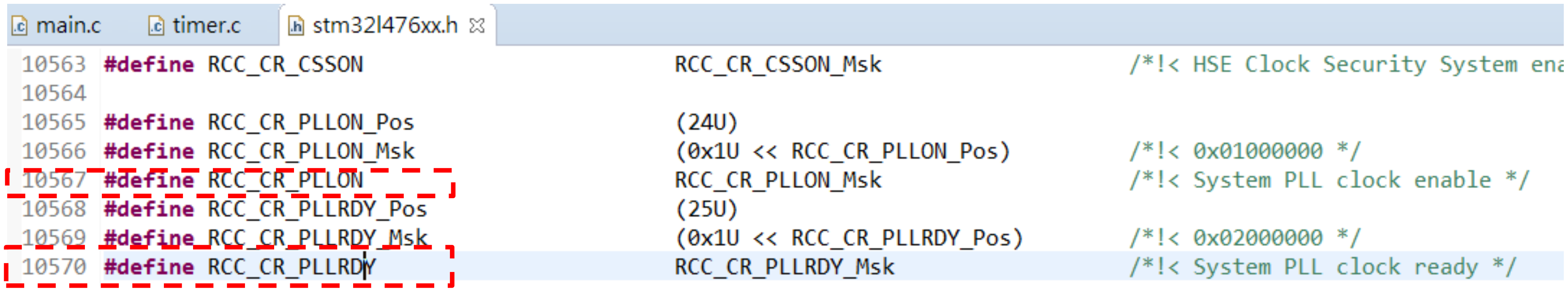
Address of CFGR & PLLON & PLLSRC



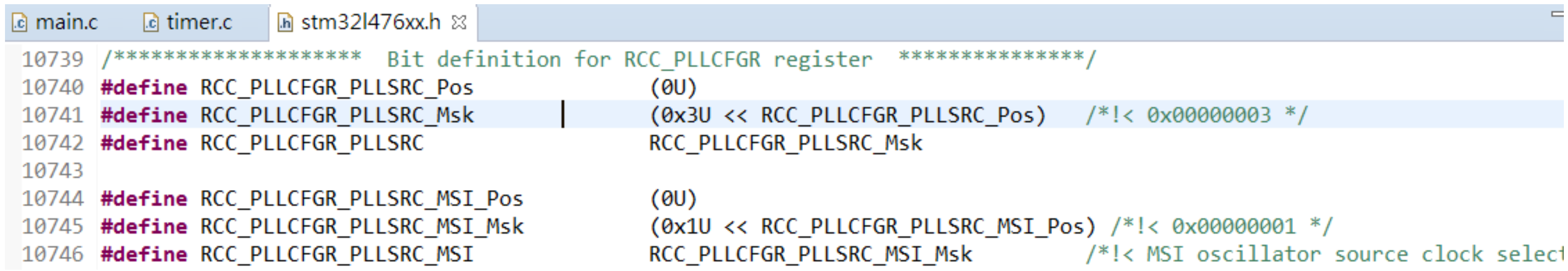
```
/*!< SW configuration */
#define RCC_CFGR_SW_Pos                      (0U)
#define RCC_CFGR_SW_Msk                      (0x3U << RCC_CFGR_SW_Pos)        /*!< 0x00000003 */
#define RCC_CFGR_SW                          RCC_CFGR_SW_Msk                  /*!< SW[1:0] bits (S
```



```
#define RCC_CFGR_SW_1                        (0x2U << RCC_CFGR_SW_Pos)        /*!< 0x00000002 */

#define RCC_CFGR_SW_MSI                      (0x00000000U)            /*!< MSI oscillator selection as system clock */
```



```
#define RCC_CR_CSSON                         RCC_CR_CSSON_Msk                 /*!< HSE Clock Security System ena

#define RCC_CR_PLLON_Pos                     (24U)
#define RCC_CR_PLLON_Msk                     (0x1U << RCC_CR_PLLON_Pos)       /*!< 0x01000000 */
#define RCC_CR_PLLON                         RCC_CR_PLLON_Msk                 /*!< System PLL clock enable */
#define RCC_CR_PLLRDY_Pos                    (25U)
#define RCC_CR_PLLRDY_Msk                    (0x1U << RCC_CR_PLLRDY_Pos)      /*!< 0x02000000 */
#define RCC_CR_PLLRDY                        RCC_CR_PLLRDY_Msk                /*!< System PLL clock ready */
```

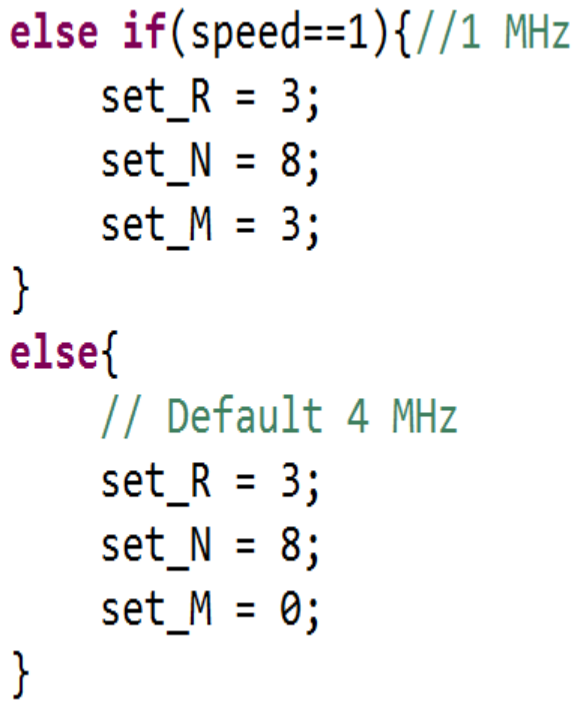


```
/********************  Bit definition for RCC_PLLCFGR register  ***************/
#define RCC_PLLCFGR_PLLSRC_Pos               (0U)
#define RCC_PLLCFGR_PLLSRC_Msk               (0x3U << RCC_PLLCFGR_PLLSRC_Pos)    /*!< 0x00000003 */
#define RCC_PLLCFGR_PLLSRC                   RCC_PLLCFGR_PLLSRC_Msk

#define RCC_PLLCFGR_PLLSRC_MSI_Pos           (0U)
#define RCC_PLLCFGR_PLLSRC_MSI_Msk           (0x1U << RCC_PLLCFGR_PLLSRC_MSI_Pos) /*!< 0x00000001 */
#define RCC_PLLCFGR_PLLSRC_MSI               RCC_PLLCFGR_PLLSRC_MSI_Msk          /*!< MSI oscillator source clock selec
```

SystemClock_Config

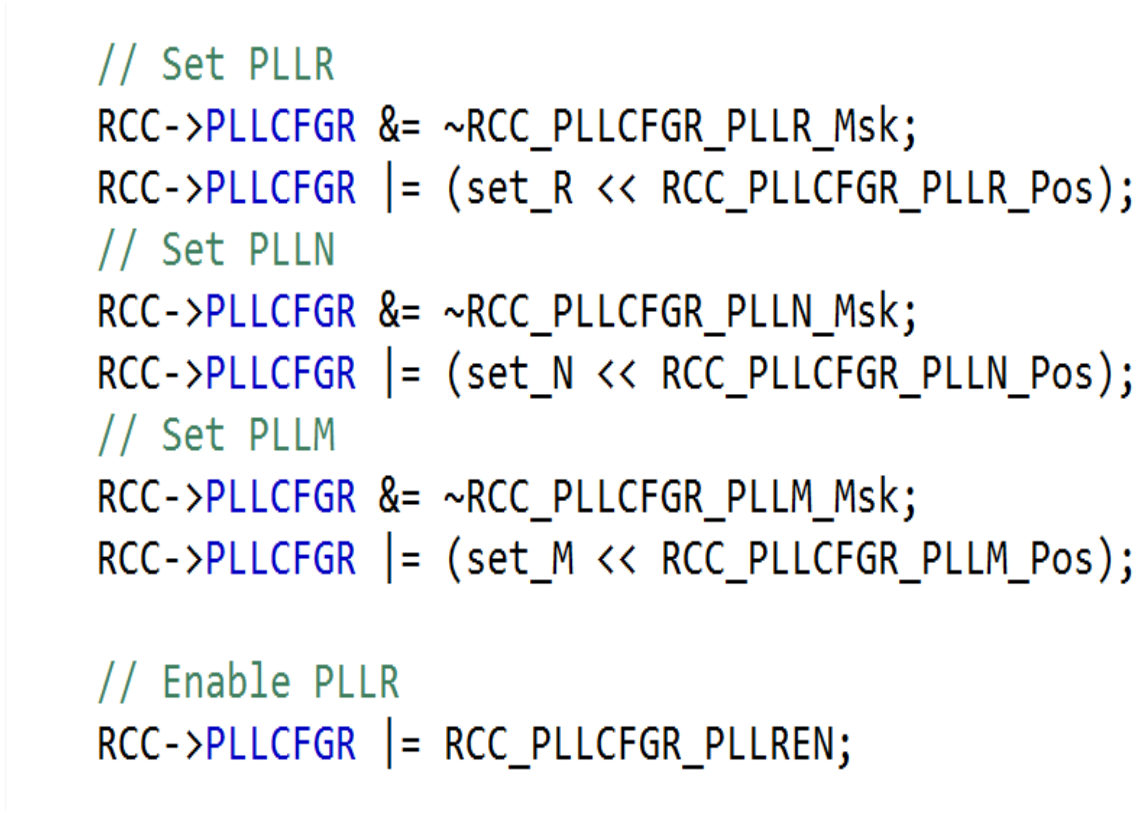
```
int set_R=0, set_N=0, set_M=0;
// Change R N M 4*N/2*(R+1)*(M+1) 4 x N / (2(R+1)) x (M+1)
if(speed==40){
    set_R = 1;
    set_N = 40;
    set_M = 0;
}
else if(speed==16){
    set_R = 0;
    set_N = 8;
    set_M = 0;
}

else if(speed==10){
    set_R = [redacted]
    set_N =
    set_M =
}
else if(speed==6){
    set_R = [redacted]
    set_N =
    set_M =
}
```

SystemClock_Config



```
else if(speed==1){//1 MHz
    set_R = 3;
    set_N = 8;
    set_M = 3;
}
else{
    // Default 4 MHz
    set_R = 3;
    set_N = 8;
    set_M = 0;
}
```

SystemClock_Config

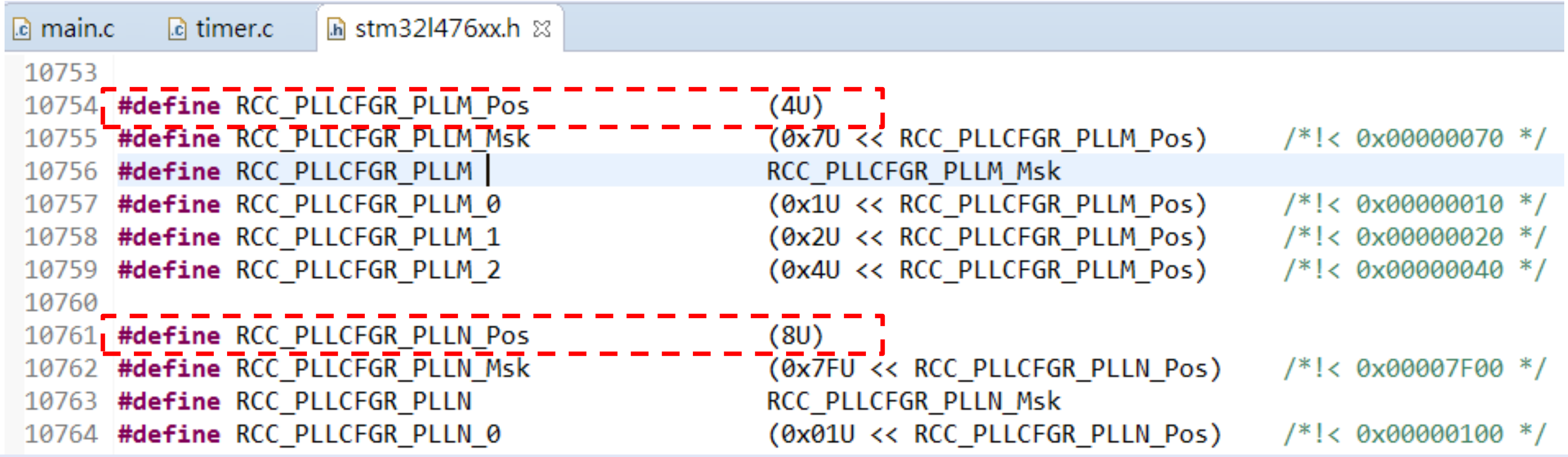


```
// Set PLLR
RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLR_Msk;
RCC->PLLCFGR |= (set_R << RCC_PLLCFGR_PLLR_Pos);
// Set PLLN
RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLN_Msk;
RCC->PLLCFGR |= (set_N << RCC_PLLCFGR_PLLN_Pos);
// Set PLLM
RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLM_Msk;
RCC->PLLCFGR |= (set_M << RCC_PLLCFGR_PLLM_Pos);

// Enable PLLR
RCC->PLLCFGR |= RCC_PLLCFGR_PLLREN;
```

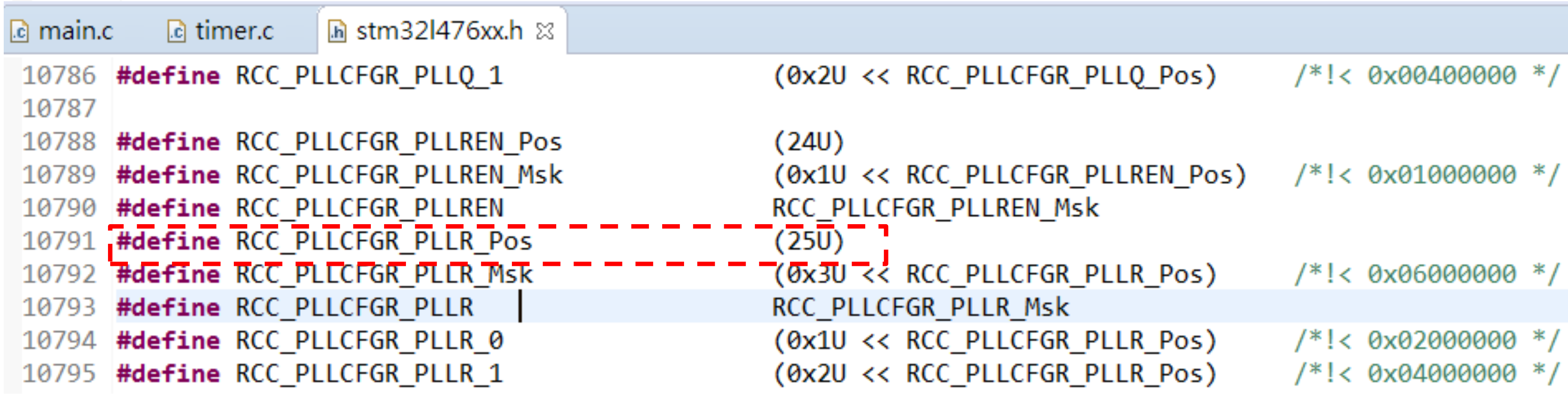
RCC_PLLCFGR第0,1 bit是PLLSRC
PLLM要shift 4, 當N=0時 PLLM=1
PLLN要shift 8,當N=8時 PLLN=8
PLLR要shift 25,當R=3時 PLLN=8

0x01000000

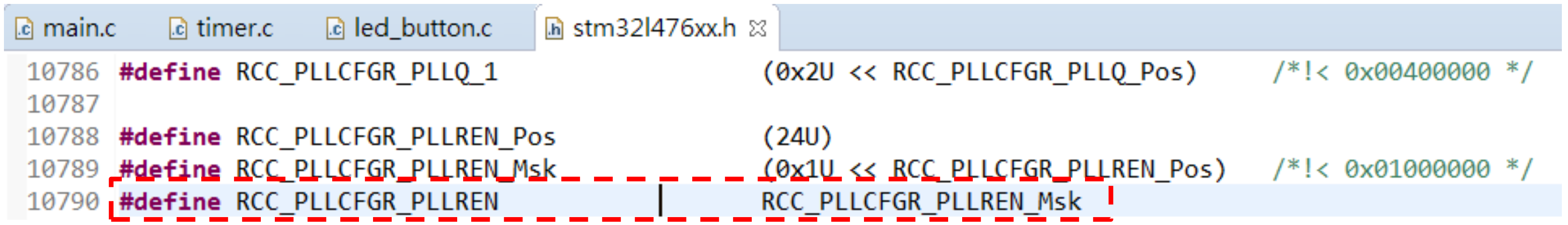
其中PLL clock頻率計算為f(VCO clock) = f(PLL clock input) 4MHz × (PLLN 8 / PLLM 1) = 32MHz
最終可輸出給system clock頻率為f(PLL_R) = f(VCO clock) 32HMz / PLLR 8 = 4MHz



```
#define RCC_PLLCFGR_PLLM_Pos                (4U)
#define RCC_PLLCFGR_PLLM_Msk                (0x7U << RCC_PLLCFGR_PLLM_Pos)      /*!< 0x00000070 */
#define RCC_PLLCFGR_PLLM                    RCC_PLLCFGR_PLLM_Msk
#define RCC_PLLCFGR_PLLM_0                  (0x1U << RCC_PLLCFGR_PLLM_Pos)      /*!< 0x00000010 */
#define RCC_PLLCFGR_PLLM_1                  (0x2U << RCC_PLLCFGR_PLLM_Pos)      /*!< 0x00000020 */
#define RCC_PLLCFGR_PLLM_2                  (0x4U << RCC_PLLCFGR_PLLM_Pos)      /*!< 0x00000040 */

#define RCC_PLLCFGR_PLLN_Pos                (8U)
#define RCC_PLLCFGR_PLLN_Msk                (0x7FU << RCC_PLLCFGR_PLLN_Pos)     /*!< 0x00007F00 */
#define RCC_PLLCFGR_PLLN                    RCC_PLLCFGR_PLLN_Msk
#define RCC_PLLCFGR_PLLN_0                  (0x01U << RCC_PLLCFGR_PLLN_Pos)     /*!< 0x00000100 */
```

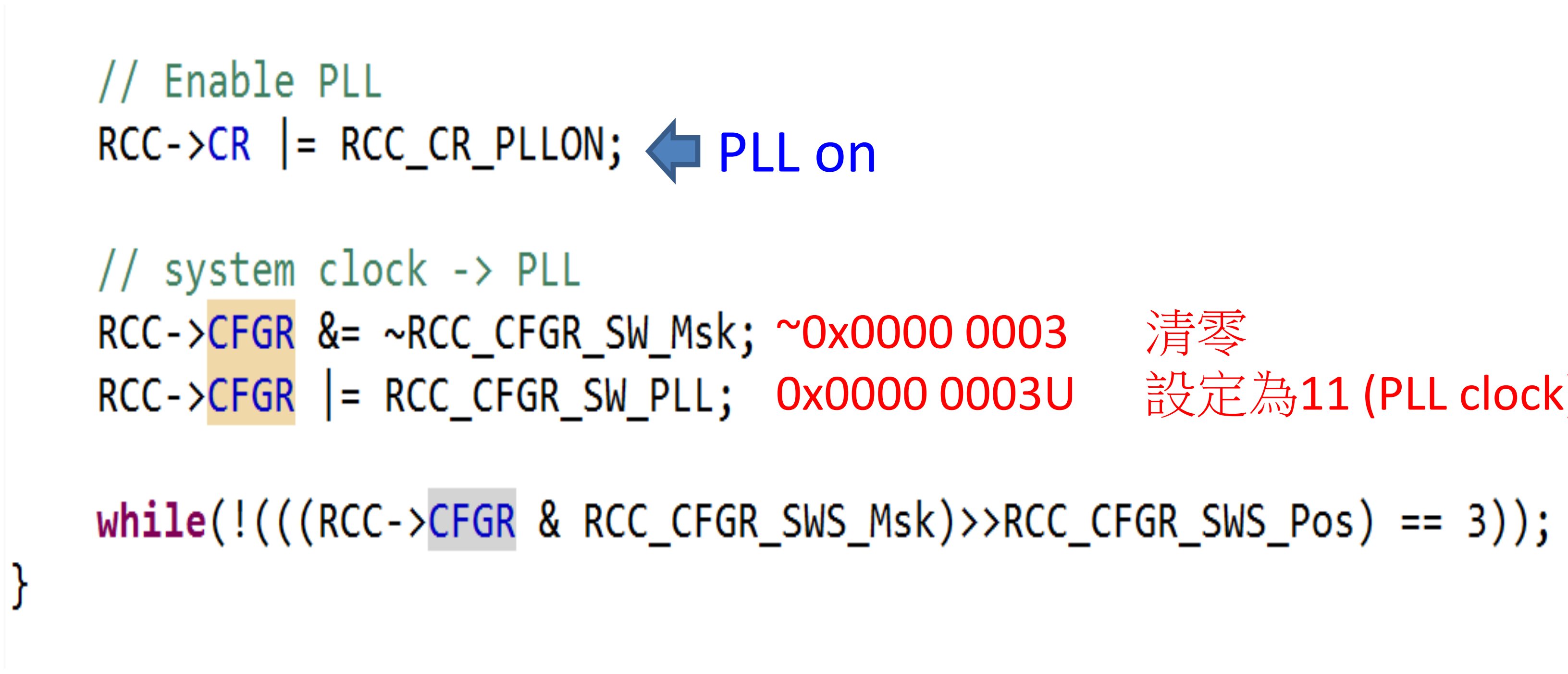


```
#define RCC_PLLCFGR_PLLQ_1                  (0x2U << RCC_PLLCFGR_PLLQ_Pos)      /*!< 0x00400000 */

#define RCC_PLLCFGR_PLLREN_Pos              (24U)
#define RCC_PLLCFGR_PLLREN_Msk              (0x1U << RCC_PLLCFGR_PLLREN_Pos)    /*!< 0x01000000 */
#define RCC_PLLCFGR_PLLREN                  RCC_PLLCFGR_PLLREN_Msk
#define RCC_PLLCFGR_PLLR_Pos                (25U)
#define RCC_PLLCFGR_PLLR_Msk                (0x3U << RCC_PLLCFGR_PLLR_Pos)      /*!< 0x06000000 */
#define RCC_PLLCFGR_PLLR                    RCC_PLLCFGR_PLLR_Msk
#define RCC_PLLCFGR_PLLR_0                  (0x1U << RCC_PLLCFGR_PLLR_Pos)      /*!< 0x02000000 */
#define RCC_PLLCFGR_PLLR_1                  (0x2U << RCC_PLLCFGR_PLLR_Pos)      /*!< 0x04000000 */
```



```
#define RCC_PLLCFGR_PLLQ_1                  (0x2U << RCC_PLLCFGR_PLLQ_Pos)      /*!< 0x00400000 */

#define RCC_PLLCFGR_PLLREN_Pos              (24U)
#define RCC_PLLCFGR_PLLREN_Msk              (0x1U << RCC_PLLCFGR_PLLREN_Pos)    /*!< 0x01000000 */
#define RCC_PLLCFGR_PLLREN                  RCC_PLLCFGR_PLLREN_Msk
```

SystemClock_Config

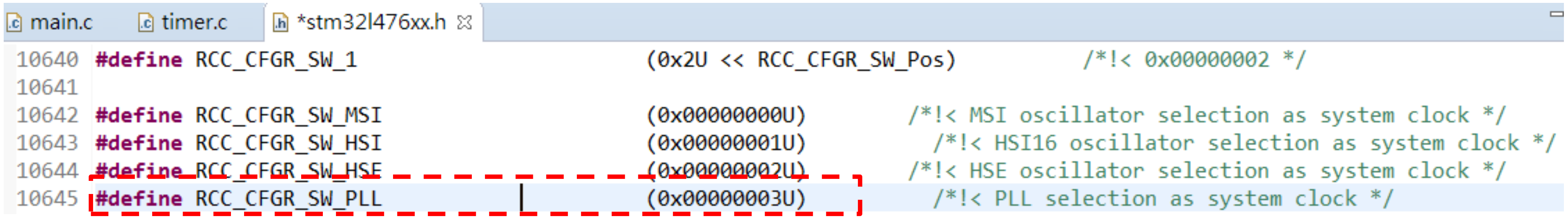


```
    // Enable PLL
    RCC->CR |= RCC_CR_PLLON;

    // system clock -> PLL
    RCC->CFGR &= ~RCC_CFGR_SW_Msk;
    RCC->CFGR |= RCC_CFGR_SW_PLL;

    while(!(((RCC->CFGR & RCC_CFGR_SWS_Msk)>>RCC_CFGR_SWS_Pos) == 3));
}
```

PLL on

~0x0000 0003　　清零

0x0000 0003U　　設定為11 (PLL clock)


main.c timer.c *stm32l476xx.h

```
#define RCC_CFGR_SW_1                        (0x2U << RCC_CFGR_SW_Pos)          /*!< 0x00000002 */

#define RCC_CFGR_SW_MSI                      (0x00000000U)       /*!< MSI oscillator selection as system clock */
#define RCC_CFGR_SW_HSI                      (0x00000001U)         /*!< HSI16 oscillator selection as system clock */
#define RCC_CFGR_SW_HSE                      (0x00000002U)       /*!< HSE oscillator selection as system clock */
#define RCC_CFGR_SW_PLL                      (0x00000003U)         /*!< PLL selection as system clock */
```

Lab5.2 Timer (30%)

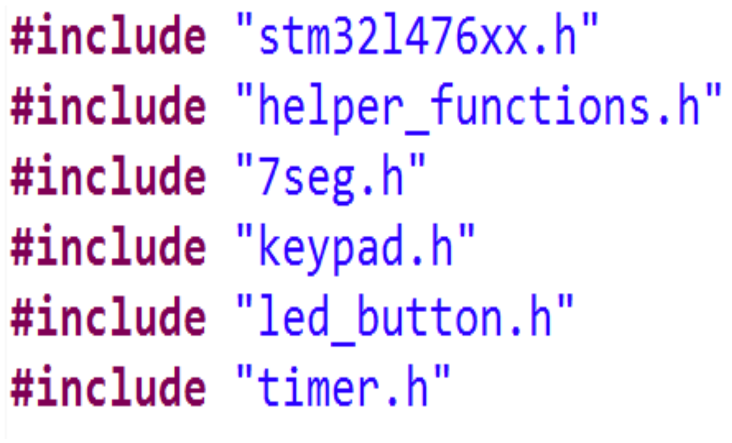
- 完成以下的main.c中的Timer_init()與Timer_start(); 並使用STM32 timer實做一個計時器會從0上數(Upcounting) TIME_SEC秒的時間。顯示到小數點以下第二位，結束時7-SEG LED停留在TIME_SEC的數字。(建議使用擁用比較高counter resolution 的TIM2~TIM5 timer)，請使用polling的方式取得 timer CNT register值並換算成時間顯示到7-SEG LED上。
- 0.01 ≤ TIME_SEC ≤ 10000.00 (超過範圍請直接顯示0.00)

- 例如 TIME_SEC 為12.7時的demo影片 ：https://goo.gl/F9hh35
- Note: 7-SEG LED驅動請利用之前Lab所實作的GPIO_init()、max7219_init()與Display ()函式呈現(須改成可呈現2個小數位)。

Lab5.2 Timer

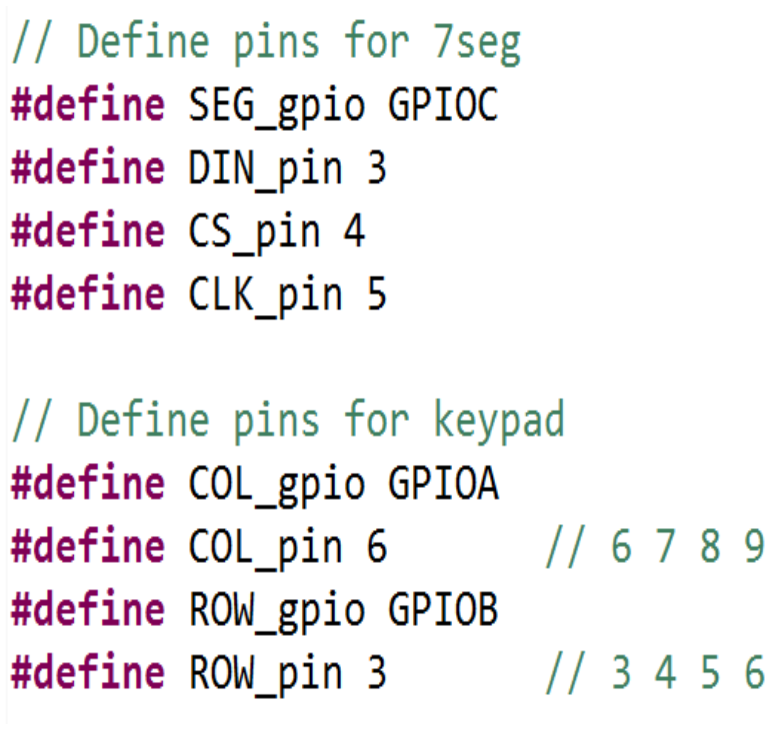
```c
main.c
#include "stm32l476xx.h"
#define TIME_SEC 12.70
extern void GPIO_init();
extern void max7219_init();
extern void Display();
void Timer_init( TIM_TypeDef *timer)
{
    //TODO: Initialize timer
}
void Timer_start(TIM_TypeDef *timer){
    //TODO: start timer and show the time on the 7-SEG LED.
}
int main()
{
   GPIO_init();
   max7219_init();
   Timer_init();
   Timer_start();
   while(1)
   {
     //TODO: Polling the timer count and do lab requirements
   }
}
```

include header file



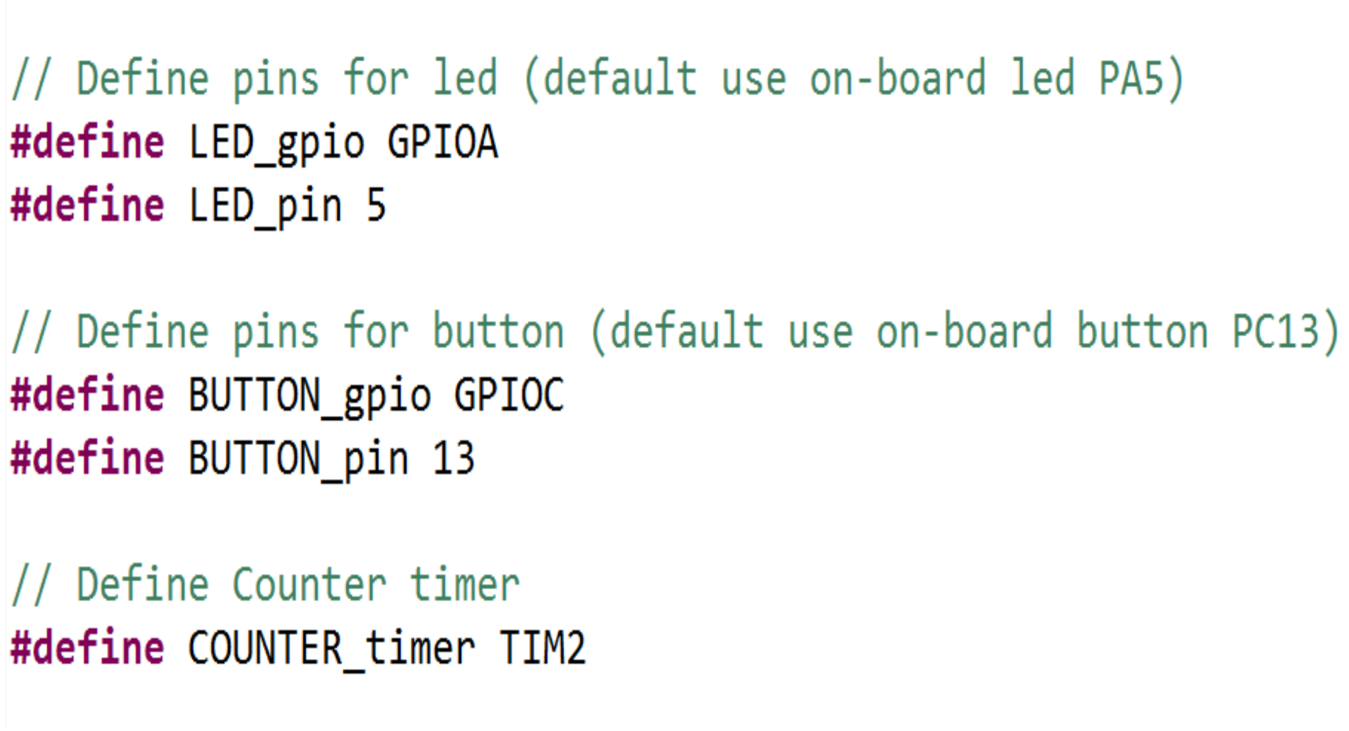
```
#include "stm32l476xx.h"
#include "helper_functions.h"
#include "7seg.h"
#include "keypad.h"
#include "led_button.h"
#include "timer.h"
```

設定Pin角跟Timer



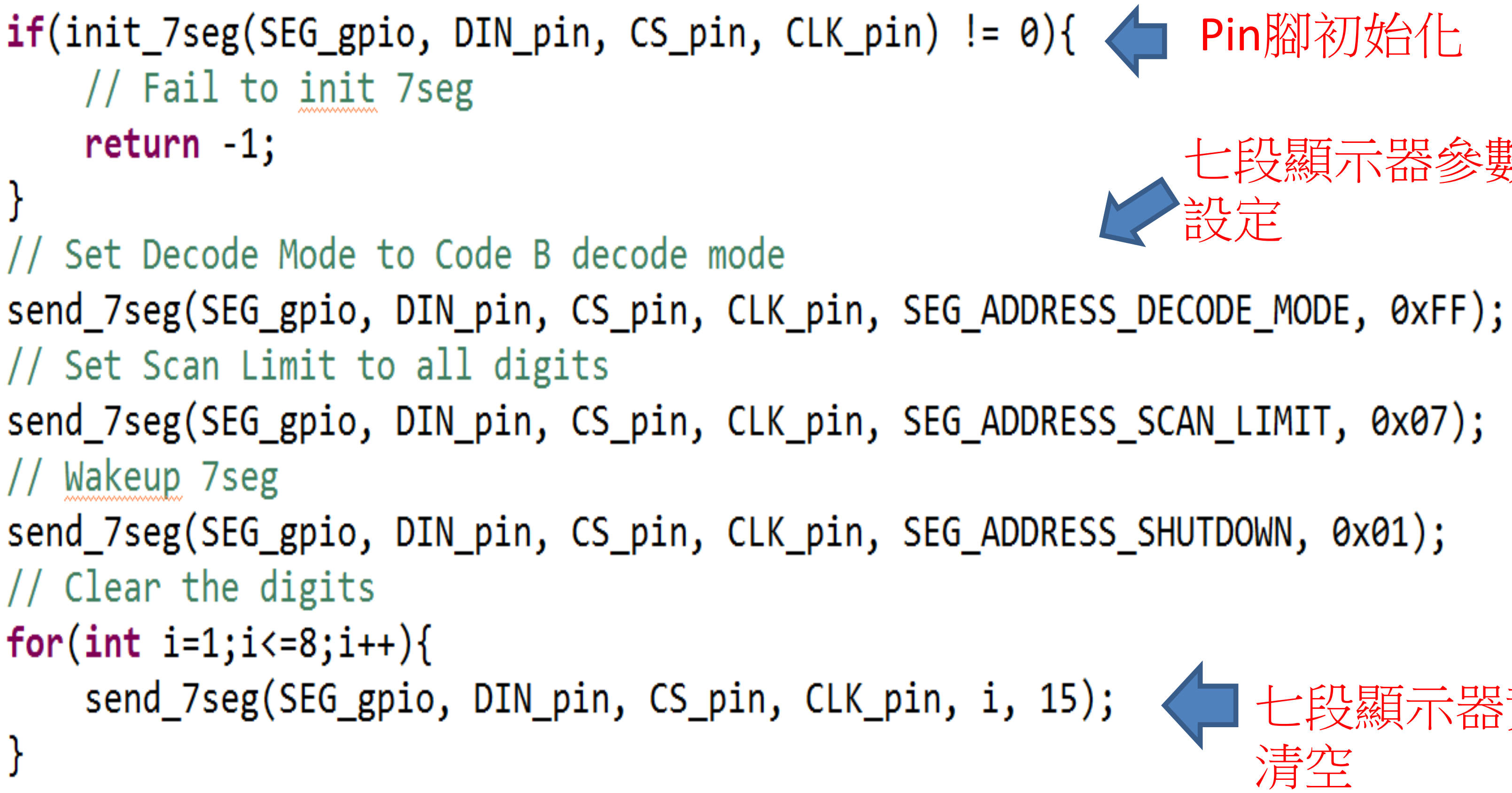
```
// Define pins for 7seg
#define SEG_gpio GPIOC
#define DIN_pin 3
#define CS_pin 4
#define CLK_pin 5

// Define pins for keypad
#define COL_gpio GPIOA
#define COL_pin 6       // 6 7 8 9
#define ROW_gpio GPIOB
#define ROW_pin 3       // 3 4 5 6
```

設定Pin角跟Timer



```
// Define pins for led (default use on-board led PA5)
#define LED_gpio GPIOA
#define LED_pin 5

// Define pins for button (default use on-board button PC13)
#define BUTTON_gpio GPIOC
#define BUTTON_pin 13

// Define Counter timer
#define COUNTER_timer TIM2
```

main



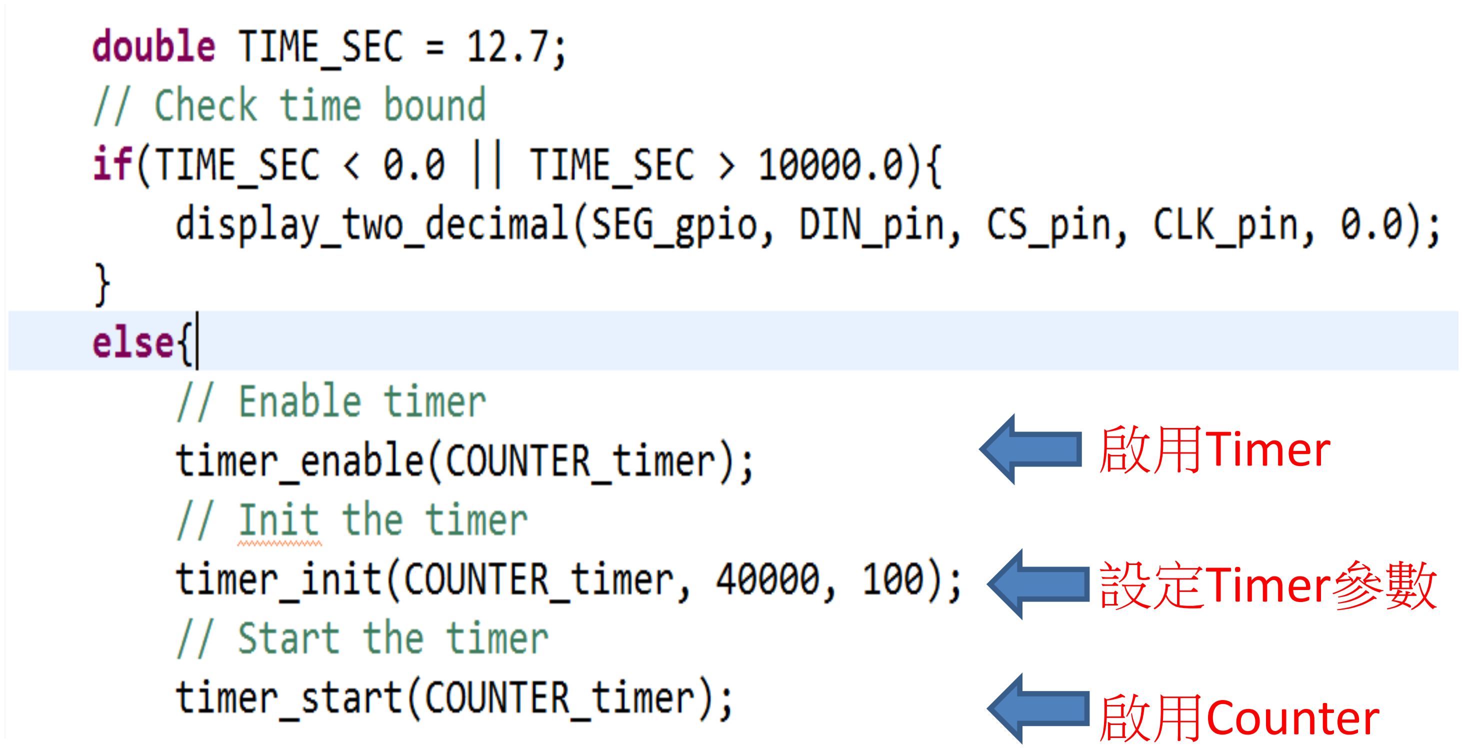
```
if(init_7seg(SEG_gpio, DIN_pin, CS_pin, CLK_pin) != 0){
    // Fail to init 7seg
    return -1;
}
// Set Decode Mode to Code B decode mode
send_7seg(SEG_gpio, DIN_pin, CS_pin, CLK_pin, SEG_ADDRESS_DECODE_MODE, 0xFF);
// Set Scan Limit to all digits
send_7seg(SEG_gpio, DIN_pin, CS_pin, CLK_pin, SEG_ADDRESS_SCAN_LIMIT, 0x07);
// Wakeup 7seg
send_7seg(SEG_gpio, DIN_pin, CS_pin, CLK_pin, SEG_ADDRESS_SHUTDOWN, 0x01);
// Clear the digits
for(int i=1;i<=8;i++){
    send_7seg(SEG_gpio, DIN_pin, CS_pin, CLK_pin, i, 15);
}
```

Pin腳初始化

七段顯示器參數設定

七段顯示器資料清空

main



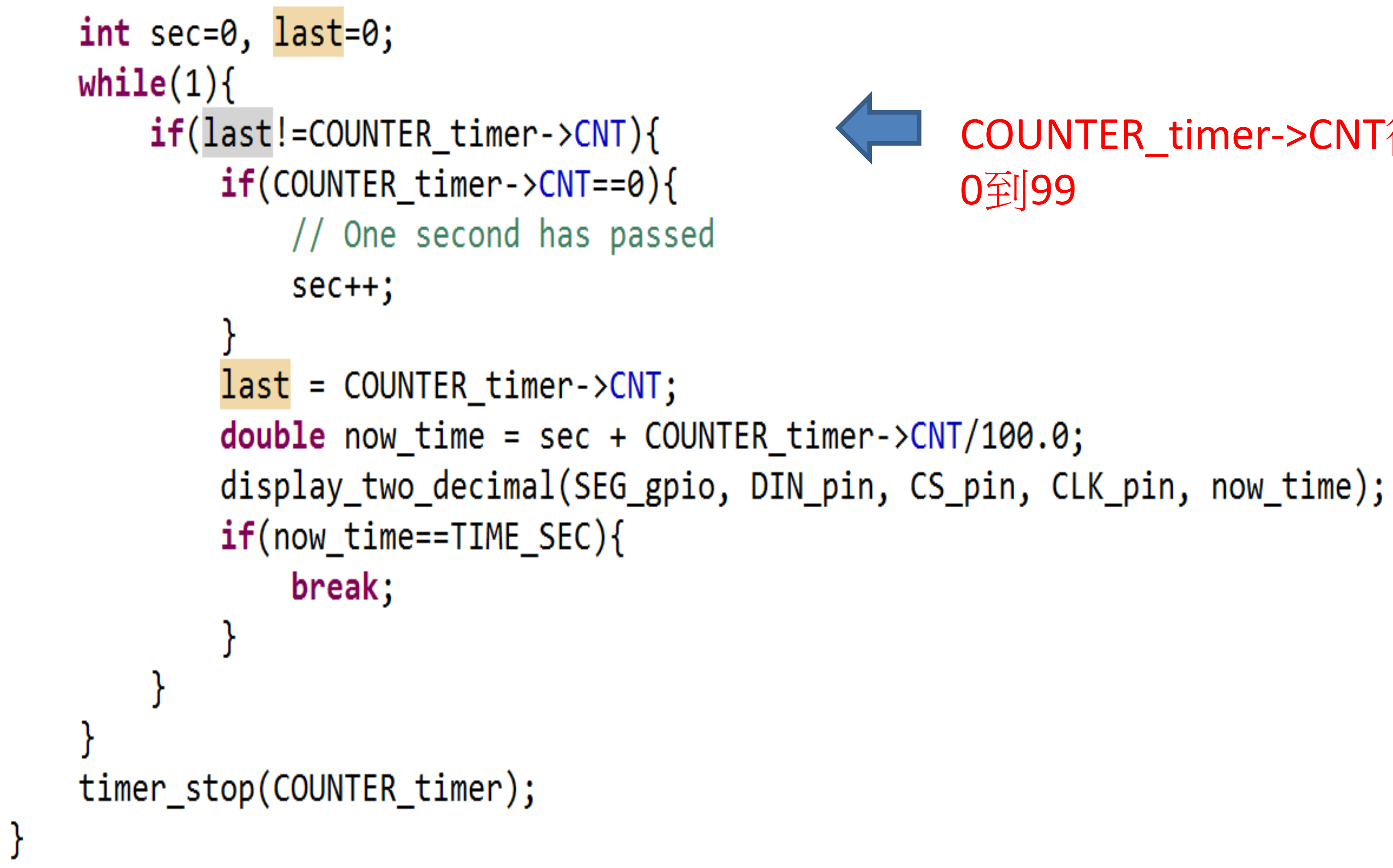
```
double TIME_SEC = 12.7;
// Check time bound
if(TIME_SEC < 0.0 || TIME_SEC > 10000.0){
    display_two_decimal(SEG_gpio, DIN_pin, CS_pin, CLK_pin, 0.0);
}
else{
    // Enable timer
    timer_enable(COUNTER_timer);
    // Init the timer
    timer_init(COUNTER_timer, 40000, 100);
    // Start the timer
    timer_start(COUNTER_timer);
```

啟用Timer

設定Timer參數

啟用Counter

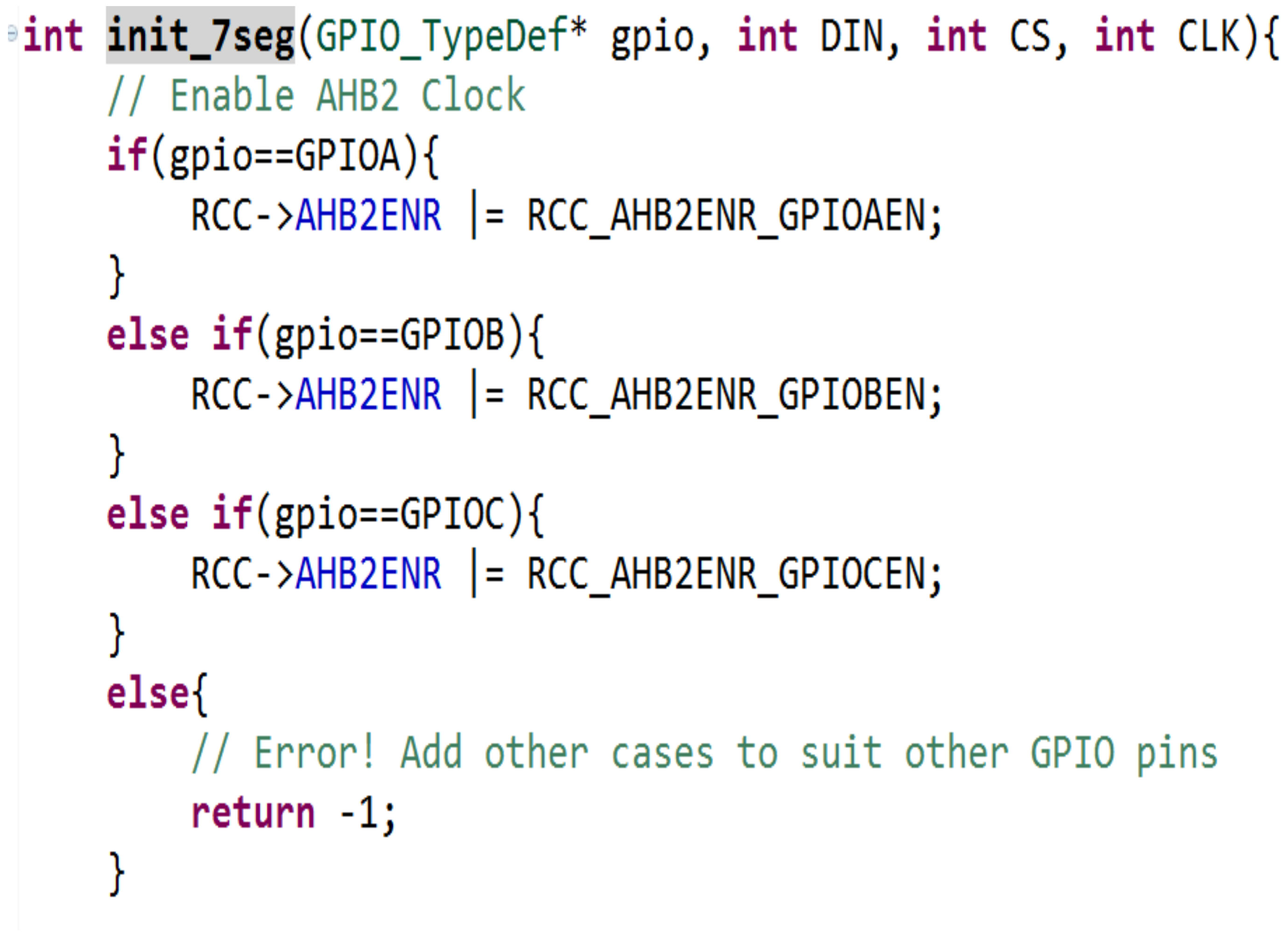
main



```
    int sec=0, last=0;
    while(1){
        if(last!=COUNTER_timer->CNT){
            if(COUNTER_timer->CNT==0){
                // One second has passed
                sec++;
            }
            last = COUNTER_timer->CNT;
            double now_time = sec + COUNTER_timer->CNT/100.0;
            display_two_decimal(SEG_gpio, DIN_pin, CS_pin, CLK_pin, now_time);
            if(now_time==TIME_SEC){
                break;
            }
        }
    }
    timer_stop(COUNTER_timer);
}
```

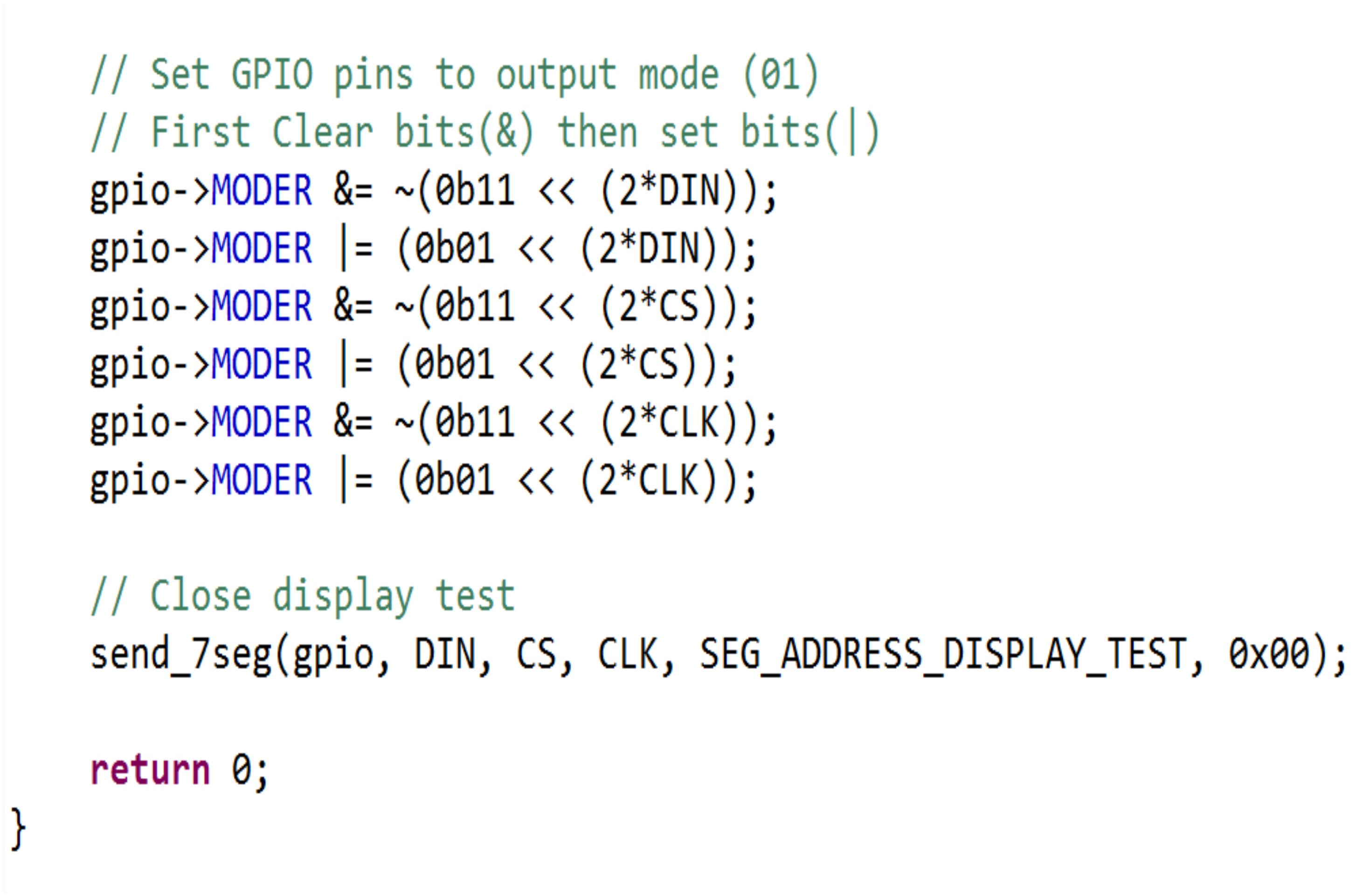
COUNTER_timer->CNT從0到99

Init_7seg



```
int init_7seg(GPIO_TypeDef* gpio, int DIN, int CS, int CLK){
    // Enable AHB2 Clock
    if(gpio==GPIOA){
        RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;
    }
    else if(gpio==GPIOB){
        RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
    }
    else if(gpio==GPIOC){
        RCC->AHB2ENR |= RCC_AHB2ENR_GPIOCEN;
    }
    else{
        // Error! Add other cases to suit other GPIO pins
        return -1;
    }
```

Init_7seg

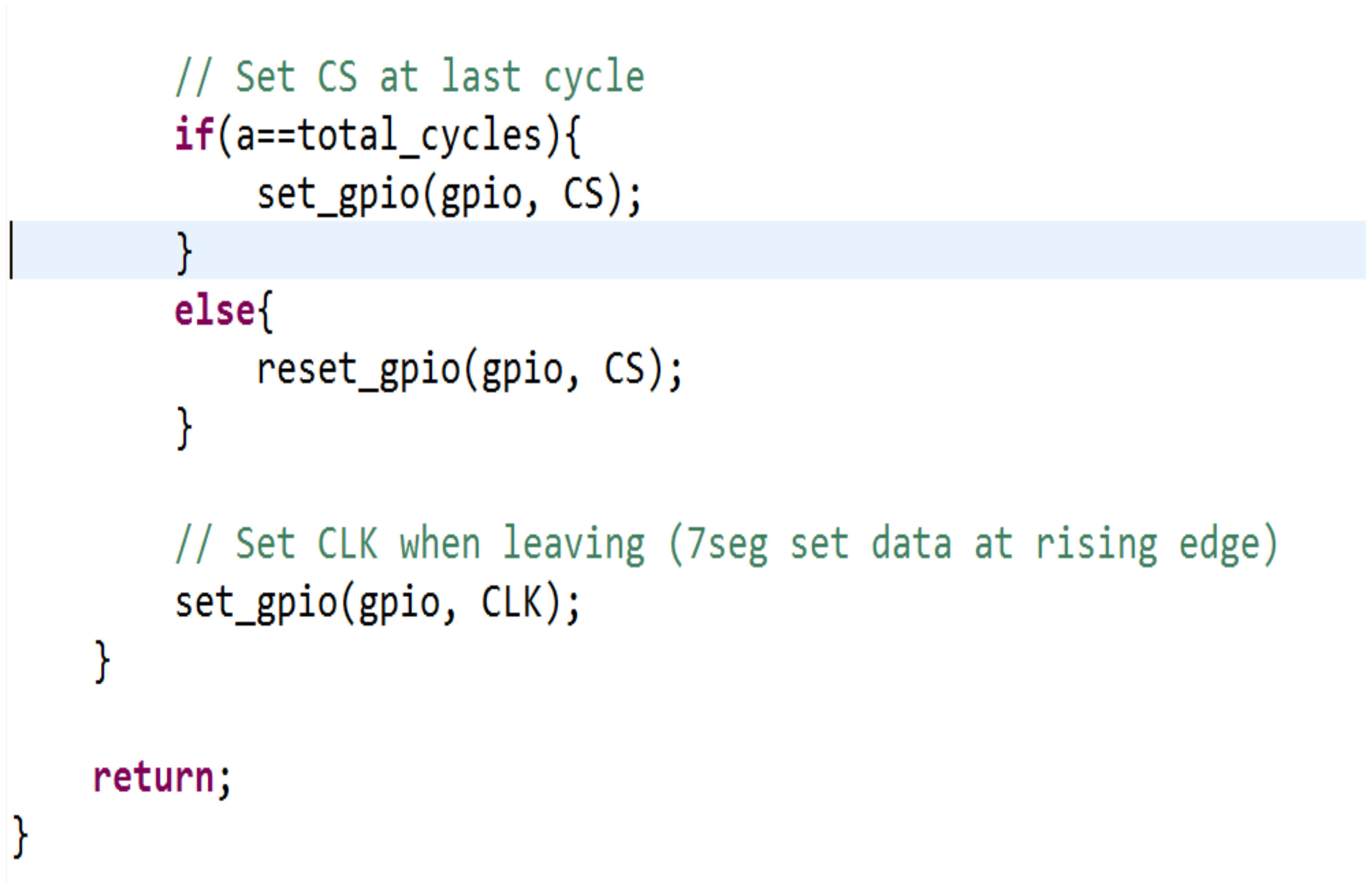


```
    // Set GPIO pins to output mode (01)
    // First Clear bits(&) then set bits(|)
    gpio->MODER &= ~(0b11 << (2*DIN));
    gpio->MODER |= (0b01 << (2*DIN));
    gpio->MODER &= ~(0b11 << (2*CS));
    gpio->MODER |= (0b01 << (2*CS));
    gpio->MODER &= ~(0b11 << (2*CLK));
    gpio->MODER |= (0b01 << (2*CLK));

    // Close display test
    send_7seg(gpio, DIN, CS, CLK, SEG_ADDRESS_DISPLAY_TEST, 0x00);

    return 0;
}
```

send_7seg

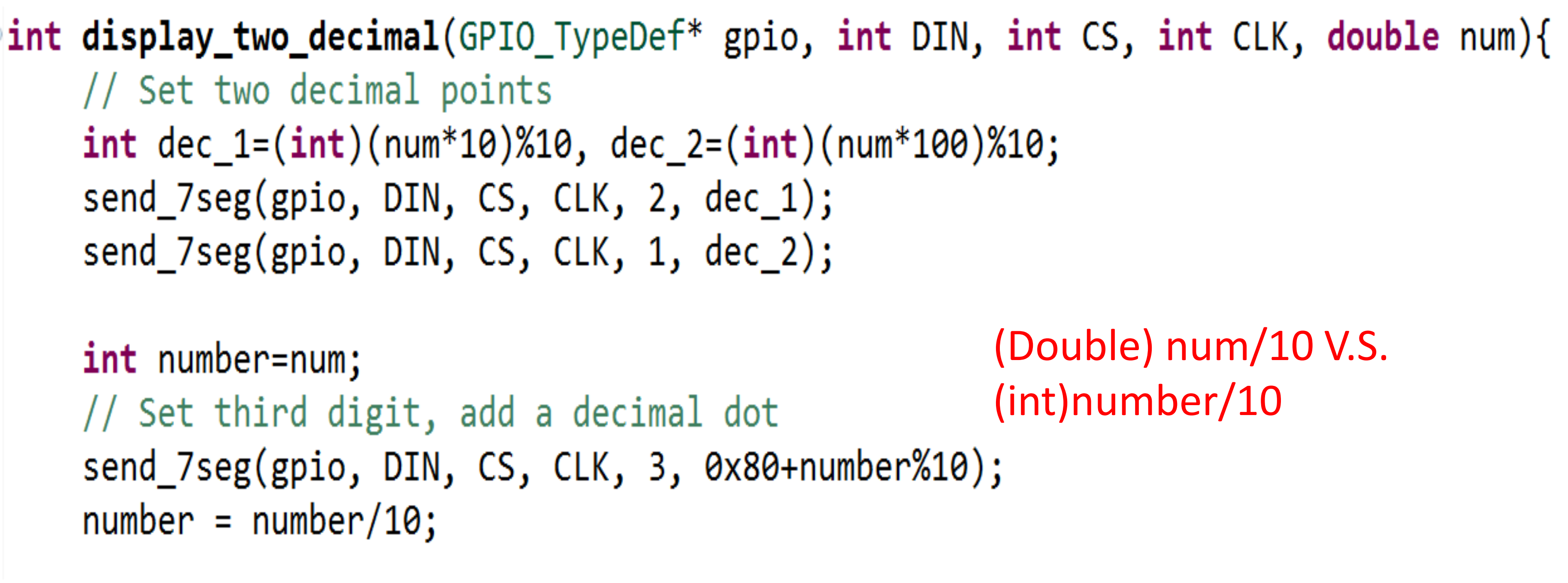
```
void send_7seg(GPIO_TypeDef* gpio, int DIN, int CS, int CLK, int address, int data){
    // The payload to send
    int payload = ((address&0xFF)<<8)|(data&0xFF);

    // Start the sending cycles
    // 16 data-bits + 1 CS signal
    int total_cycles = 16+1;

    for(int a=1;a<=total_cycles;a++){
        // Reset CLK when enter
        reset_gpio(gpio, CLK);

        // Set DIN according to data except for last cycle(CS)
        if(((payload>>(total_cycles-1-a))&0x1) && a!=total_cycles){
            set_gpio(gpio, DIN);
        }
        else{
            reset_gpio(gpio, DIN);
        }
```

send_7seg



```
        // Set CS at last cycle
        if(a==total_cycles){
            set_gpio(gpio, CS);
        }
        else{
            reset_gpio(gpio, CS);
        }

        // Set CLK when leaving (7seg set data at rising edge)
        set_gpio(gpio, CLK);
    }

    return;
}
```

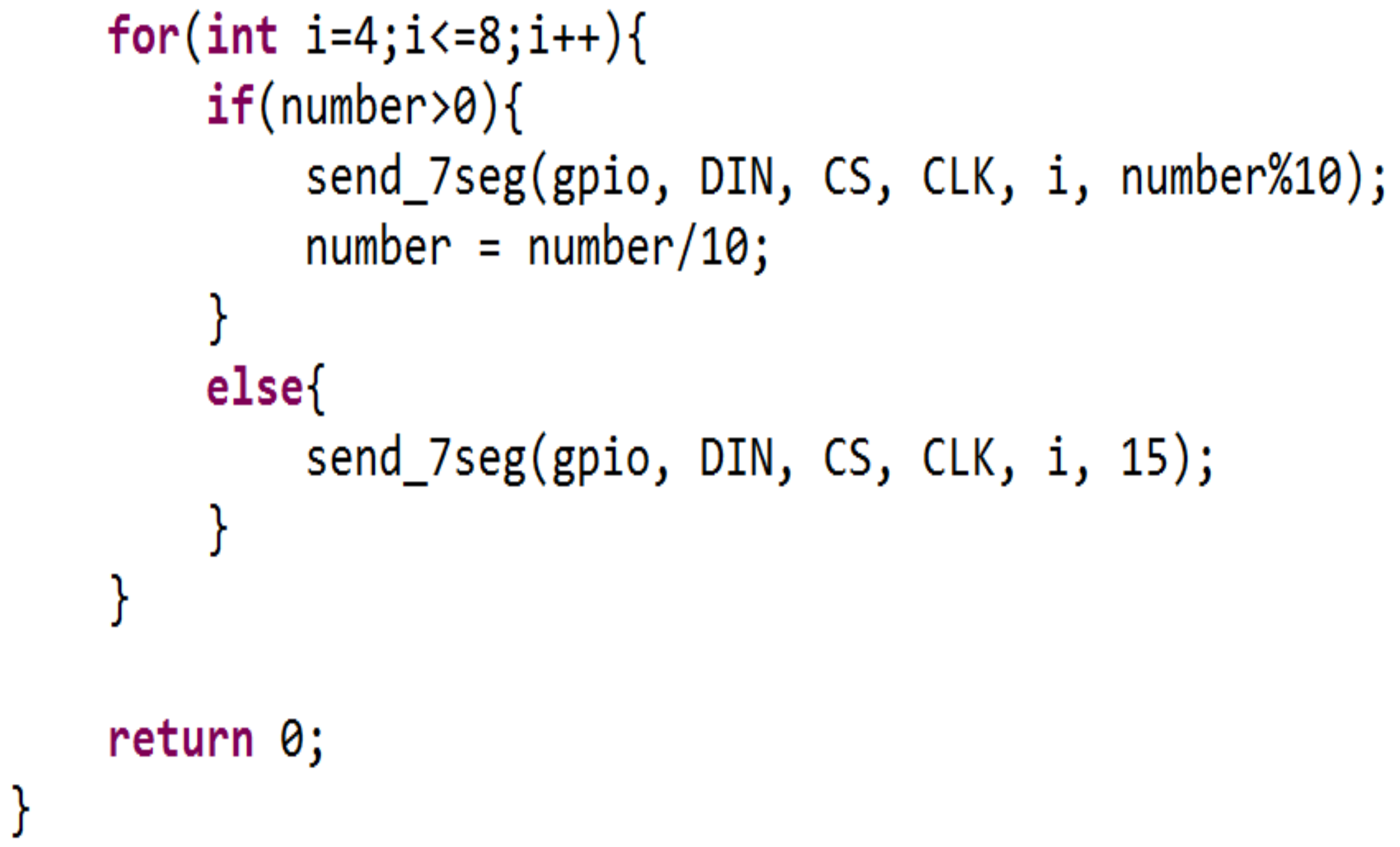
display_two_decimal



```
int display_two_decimal(GPIO_TypeDef* gpio, int DIN, int CS, int CLK, double num){
    // Set two decimal points
    int dec_1=(int)(num*10)%10, dec_2=(int)(num*100)%10;
    send_7seg(gpio, DIN, CS, CLK, 2, dec_1);
    send_7seg(gpio, DIN, CS, CLK, 1, dec_2);

    int number=num;
    // Set third digit, add a decimal dot
    send_7seg(gpio, DIN, CS, CLK, 3, 0x80+number%10);
    number = number/10;
```

(Double) num/10 V.S.
(int)number/10

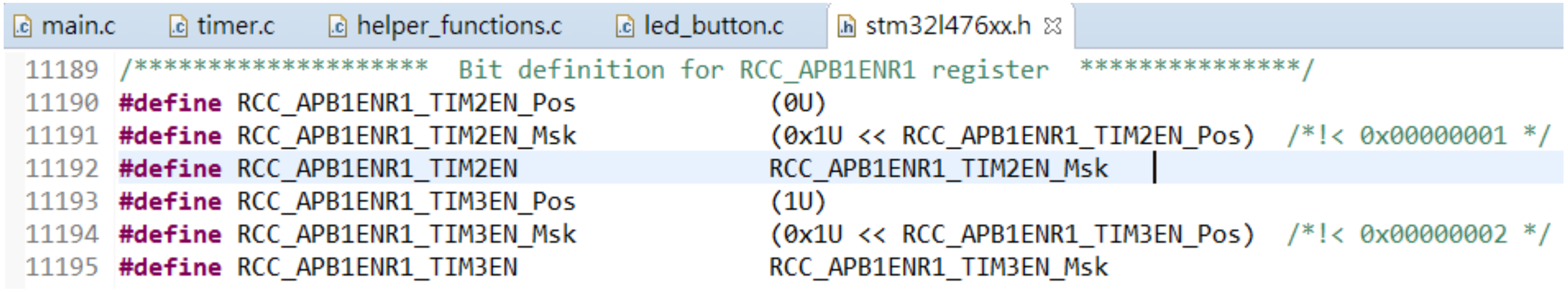
display_two_decimal



```
    for(int i=4;i<=8;i++){
        if(number>0){
            send_7seg(gpio, DIN, CS, CLK, i, number%10);
            number = number/10;
        }
        else{
            send_7seg(gpio, DIN, CS, CLK, i, 15);
        }
    }

    return 0;
}
```

timer_enable

```c
void timer_enable(TIM_TypeDef *timer){
    if(timer==TIM2){
        RCC->APB1ENR1 |= RCC_APB1ENR1_TIM2EN;     // TIM2 clock enable
    }
    else if(timer==TIM3){
        RCC->APB1ENR1 |= RCC_APB1ENR1_TIM3EN;     // TIM3 clock enable
    }
}
```



```c
/********************  Bit definition for RCC_APB1ENR1 register  ***************/
#define RCC_APB1ENR1_TIM2EN_Pos              (0U)
#define RCC_APB1ENR1_TIM2EN_Msk              (0x1U << RCC_APB1ENR1_TIM2EN_Pos)  /*!< 0x00000001 */
#define RCC_APB1ENR1_TIM2EN                  RCC_APB1ENR1_TIM2EN_Msk
#define RCC_APB1ENR1_TIM3EN_Pos              (1U)
#define RCC_APB1ENR1_TIM3EN_Msk              (0x1U << RCC_APB1ENR1_TIM3EN_Pos)  /*!< 0x00000002 */
#define RCC_APB1ENR1_TIM3EN                  RCC_APB1ENR1_TIM3EN_Msk
```

6.4.19 APB1 peripheral clock enable register 1 (RCC_APB1ENR1)

Address: 0x58

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LPTIM1 EN	OPAMP EN	DAC1 EN	PWR EN	Res.	Res.	CAN1 EN	Res.	I2C3 EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART3 EN	USART2 EN	Res.
rw	rw	rw	rw			rw		rw	rw	rw	rw	rw	rw	rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Res.	Res.	WWD GEN	Res.	LCD EN	Res.	Res.	Res.	TIM7 EN	TIM6EN	TIM5EN	TIM4EN	TIM3EN	TIM2 EN
rw	rw			rs		rw				rw	rw	rw	rw	rw	rw

Bit 1 **TIM3EN:** TIM3 timer clock enable

Set and cleared by software.

0: TIM3 clock disabled

1: TIM3 clock enabled

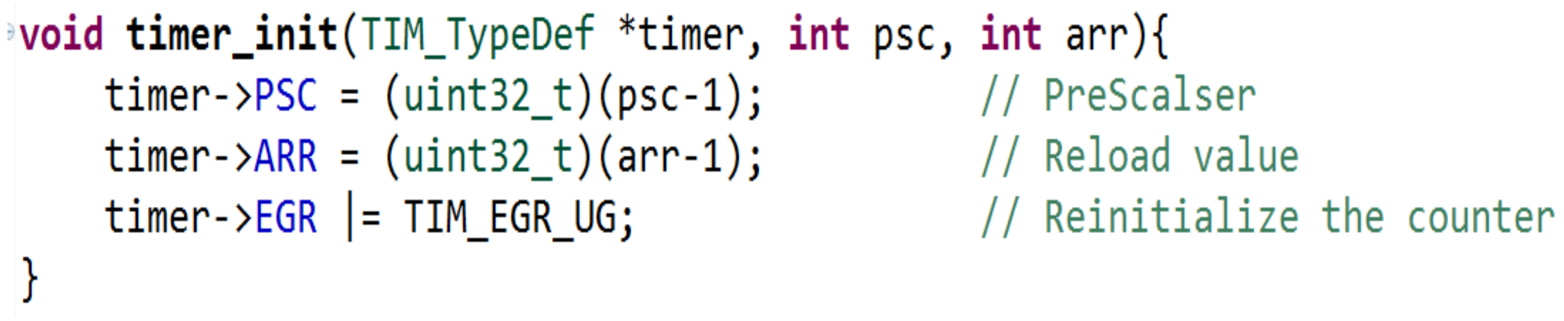
Bit 0 **TIM2EN:** TIM2 timer clock enable

Set and cleared by software.

0: TIM2 clock disabled

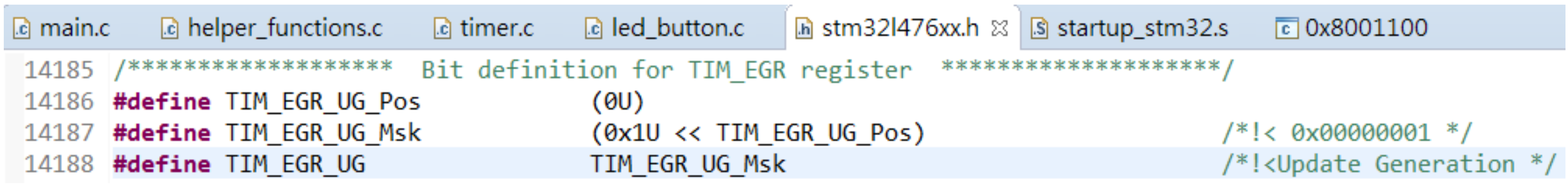
1: TIM2 clock enabled

timer_init

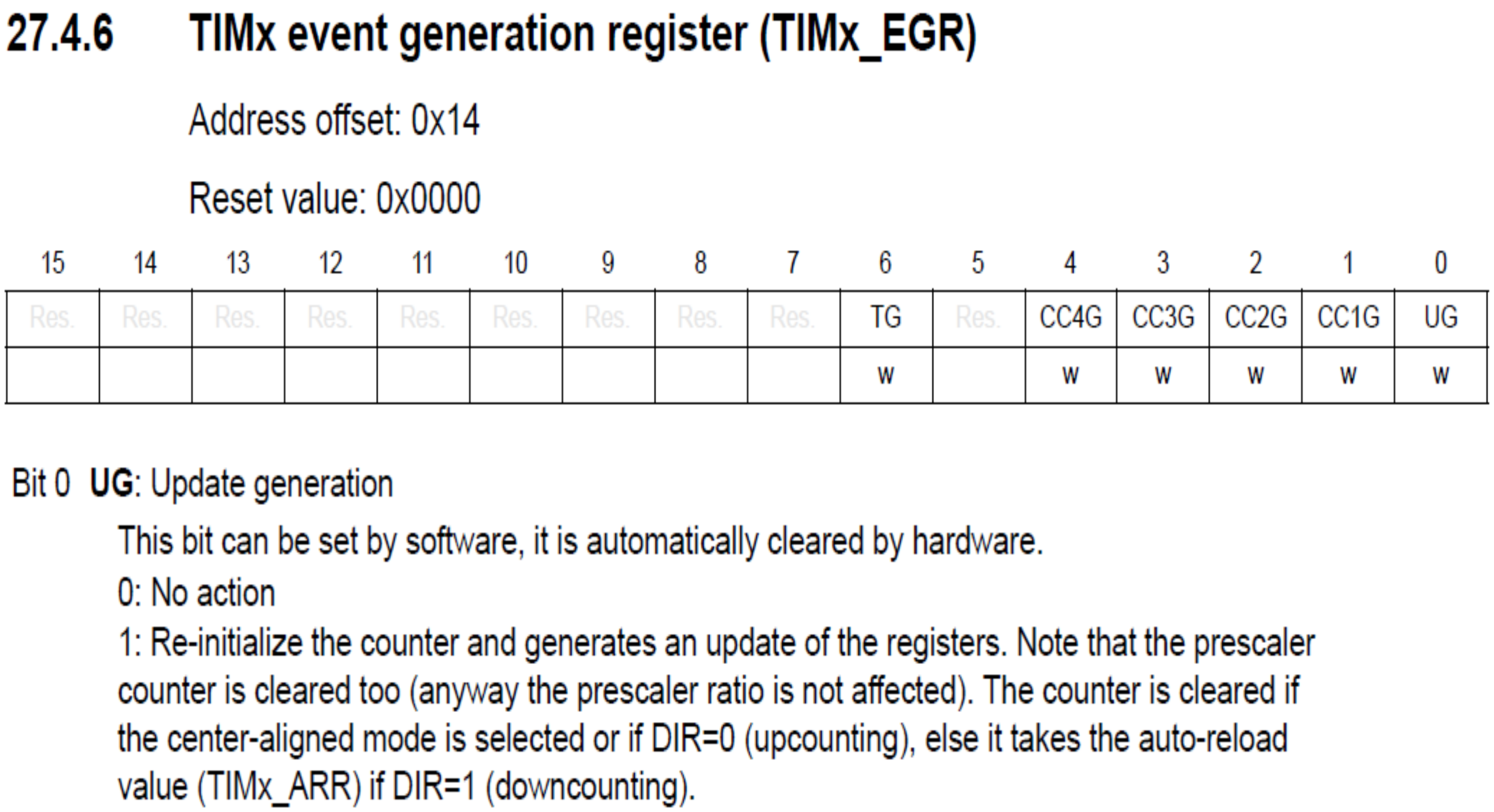


```
void timer_init(TIM_TypeDef *timer, int psc, int arr){
    timer->PSC = (uint32_t)(psc-1);          // PreScalser
    timer->ARR = (uint32_t)(arr-1);          // Reload value
    timer->EGR |= TIM_EGR_UG;                // Reinitialize the counter
}
```

PSC: prescale代表電腦跑幾次後才輸出一次counter
ARR: Auto-reload-register代表輸出多少counter後要reload
EGR: Update generation是否自動重新reload



```
/******************  Bit definition for TIM_EGR register  ********************/
#define TIM_EGR_UG_Pos          (0U)
#define TIM_EGR_UG_Msk          (0x1U << TIM_EGR_UG_Pos)                /*!< 0x00000001 */
#define TIM_EGR_UG              TIM_EGR_UG_Msk                          /*!<Update Generation */
```



27.4.6 TIMx event generation register (TIMx_EGR)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TG	Res.	CC4G	CC3G	CC2G	CC1G	UG
									w		w	w	w	w	w

Bit 0 **UG**: Update generation

This bit can be set by software, it is automatically cleared by hardware.

0: No action

1: Re-initialize the counter and generates an update of the registers. Note that the prescaler counter is cleared too (anyway the prescaler ratio is not affected). The counter is cleared if the center-aligned mode is selected or if DIR=0 (upcounting), else it takes the auto-reload value (TIMx_ARR) if DIR=1 (downcounting).

26.4.11 TIM1/TIM8 prescaler (TIMx_PSC)

Address offset: 0x28

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency (CK_CNT) is equal to f_{CK_PSC} / (PSC[15:0] + 1).
PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx_EGR register or through trigger controller when configured in “reset mode”).

26.4.12 TIM1/TIM8 auto-reload register (TIMx_ARR)

Address offset: 0x2C

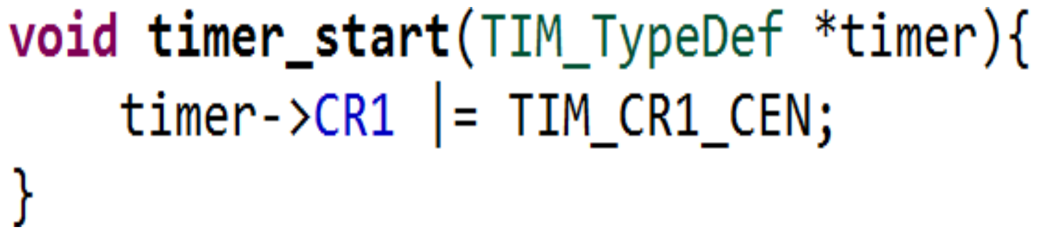
Reset value: 0xFFFF

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **ARR[15:0]**: Prescaler value

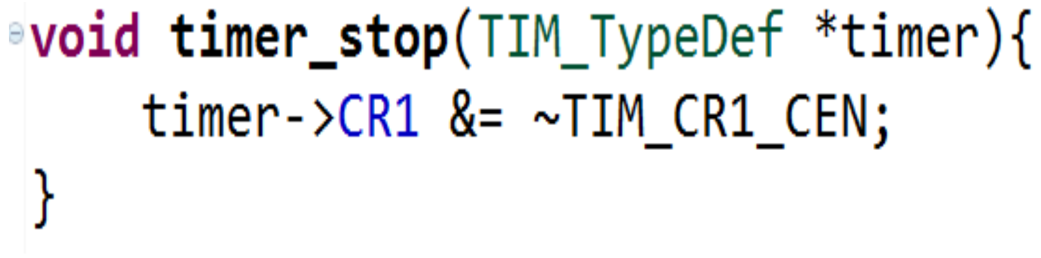
ARR is the value to be loaded in the actual auto-reload register.
Refer to the *Section 26.3.1: Time-base unit on page 755* for more details about ARR update and behavior.
The counter is blocked while the auto-reload value is null.

timer_start



```
void timer_start(TIM_TypeDef *timer){
    timer->CR1 |= TIM_CR1_CEN;
}
```

timer_stop



```
void timer_stop(TIM_TypeDef *timer){
    timer->CR1 &= ~TIM_CR1_CEN;
}
```

26.4.1 TIM1/TIM8 control register 1 (TIMx_CR1)

Address offset: 0x00

Reset value: 0x0000

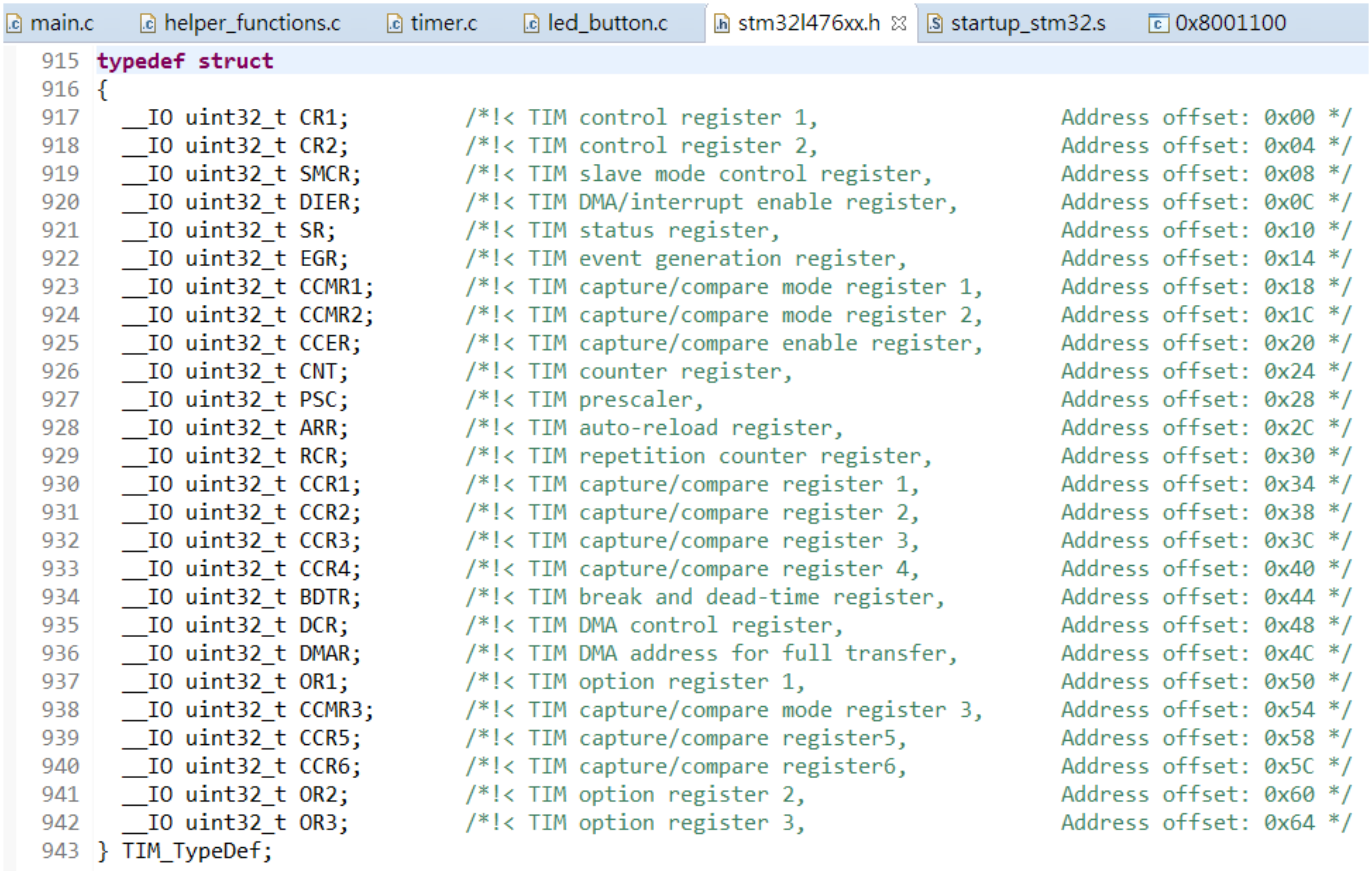
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	UIFRE MAP	Res.	CKD[1:0]		ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN
				rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 0 **CEN**: Counter enable

0: Counter disabled

1: Counter enabled

TIM_TypeDef

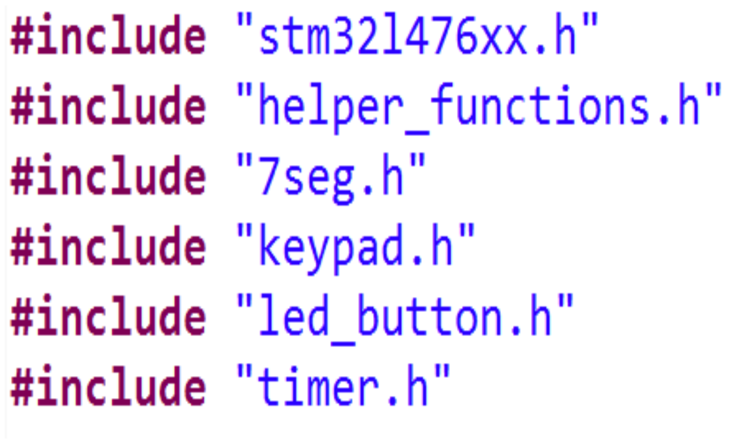


main.c helper_functions.c timer.c led_button.c stm32l476xx.h startup_stm32.s 0x8001100

```c
typedef struct
{
  __IO uint32_t CR1;         /*!< TIM control register 1,                   Address offset: 0x00 */
  __IO uint32_t CR2;         /*!< TIM control register 2,                   Address offset: 0x04 */
  __IO uint32_t SMCR;        /*!< TIM slave mode control register,          Address offset: 0x08 */
  __IO uint32_t DIER;        /*!< TIM DMA/interrupt enable register,        Address offset: 0x0C */
  __IO uint32_t SR;          /*!< TIM status register,                      Address offset: 0x10 */
  __IO uint32_t EGR;         /*!< TIM event generation register,            Address offset: 0x14 */
  __IO uint32_t CCMR1;       /*!< TIM capture/compare mode register 1,      Address offset: 0x18 */
  __IO uint32_t CCMR2;       /*!< TIM capture/compare mode register 2,      Address offset: 0x1C */
  __IO uint32_t CCER;        /*!< TIM capture/compare enable register,      Address offset: 0x20 */
  __IO uint32_t CNT;         /*!< TIM counter register,                     Address offset: 0x24 */
  __IO uint32_t PSC;         /*!< TIM prescaler,                            Address offset: 0x28 */
  __IO uint32_t ARR;         /*!< TIM auto-reload register,                 Address offset: 0x2C */
  __IO uint32_t RCR;         /*!< TIM repetition counter register,          Address offset: 0x30 */
  __IO uint32_t CCR1;        /*!< TIM capture/compare register 1,           Address offset: 0x34 */
  __IO uint32_t CCR2;        /*!< TIM capture/compare register 2,           Address offset: 0x38 */
  __IO uint32_t CCR3;        /*!< TIM capture/compare register 3,           Address offset: 0x3C */
  __IO uint32_t CCR4;        /*!< TIM capture/compare register 4,           Address offset: 0x40 */
  __IO uint32_t BDTR;        /*!< TIM break and dead-time register,         Address offset: 0x44 */
  __IO uint32_t DCR;         /*!< TIM DMA control register,                 Address offset: 0x48 */
  __IO uint32_t DMAR;        /*!< TIM DMA address for full transfer,        Address offset: 0x4C */
  __IO uint32_t OR1;         /*!< TIM option register 1,                    Address offset: 0x50 */
  __IO uint32_t CCMR3;       /*!< TIM capture/compare mode register 3,      Address offset: 0x54 */
  __IO uint32_t CCR5;        /*!< TIM capture/compare register5,            Address offset: 0x58 */
  __IO uint32_t CCR6;        /*!< TIM capture/compare register6,            Address offset: 0x5C */
  __IO uint32_t OR2;         /*!< TIM option register 2,                    Address offset: 0x60 */
  __IO uint32_t OR3;         /*!< TIM option register 3,                    Address offset: 0x64 */
} TIM_TypeDef;
```

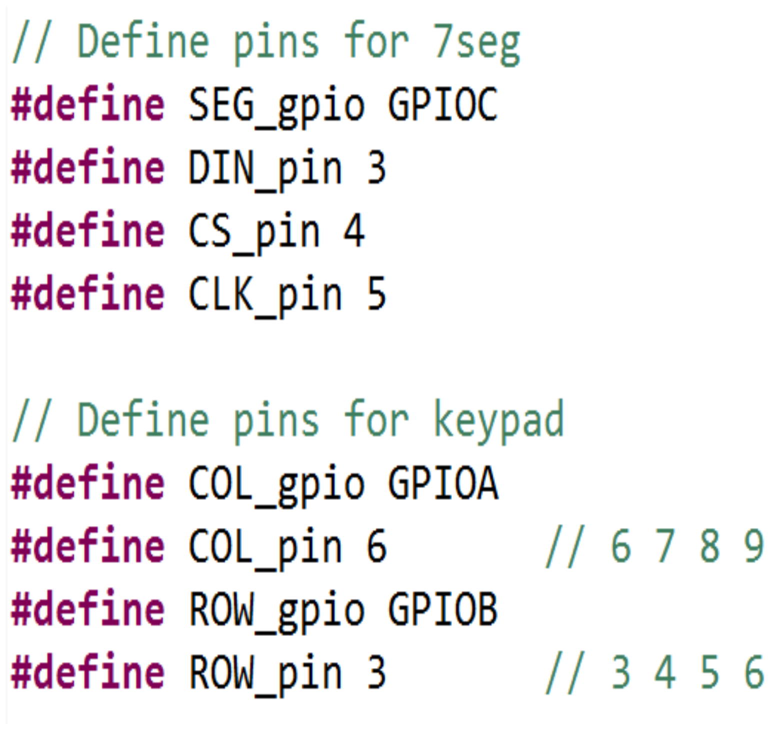
Lab5.3 Music keypad(35%)

include header file



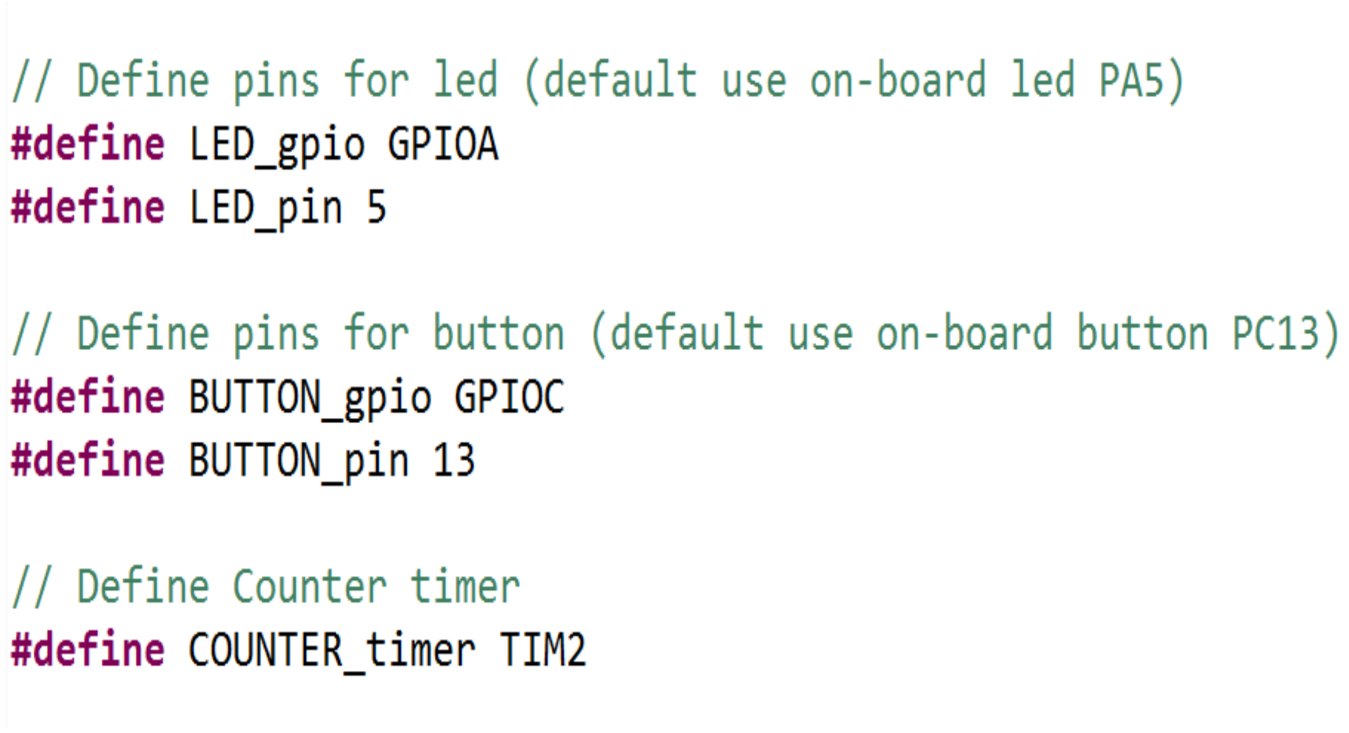
```
#include "stm32l476xx.h"
#include "helper_functions.h"
#include "7seg.h"
#include "keypad.h"
#include "led_button.h"
#include "timer.h"
```

設定Pin角跟Timer



```c
// Define pins for 7seg
#define SEG_gpio GPIOC
#define DIN_pin 3
#define CS_pin 4
#define CLK_pin 5

// Define pins for keypad
#define COL_gpio GPIOA
#define COL_pin 6       // 6 7 8 9
#define ROW_gpio GPIOB
#define ROW_pin 3       // 3 4 5 6
```

設定Pin角跟Timer



```
// Define pins for led (default use on-board led PA5)
#define LED_gpio GPIOA
#define LED_pin 5

// Define pins for button (default use on-board button PC13)
#define BUTTON_gpio GPIOC
#define BUTTON_pin 13

// Define Counter timer
#define COUNTER_timer TIM2
```

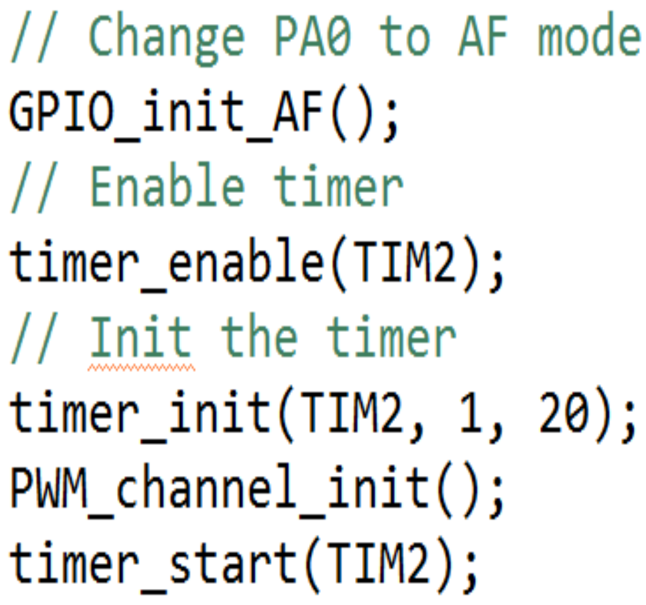
main

```
    FPU_init();

if(init_button(BUTTON_gpio, BUTTON_pin) != 0){
        // Fail to init button
        return -1;
    }
// Used to indicate led state: un-lit(0) or lit(1)
    int state=0;
    // button_press_cycle_per_second (How many button press segments in a second)
    int button_press_cycle_per_second = 20;
    // Use to state how many cycles to check per button_press_cycle
    int debounce_cycles = 50;
    // Use to state the threshold when we consider a button press
    int debounce_threshold = debounce_cycles*0.7;
    // Used to implement negative edge trigger 0=not-presses 1=pressed
    int last_button_state=0;
```

浮點數初

參數設定

main



```
// Change PA0 to AF mode - use for PWM
GPIO_init_AF();
// Enable timer
timer_enable(TIM2);
// Init the timer
timer_init(TIM2, 1, 20);
PWM_channel_init();
timer_start(TIM2);
```

把PA0設為Alternate function mode

把外接的Timer開啟

把外接的Timer參數初始化

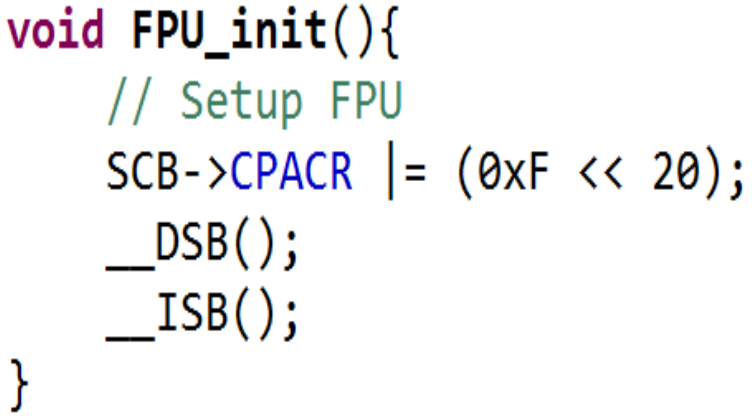
把PA0設成PWM輸出方法

把Timer的counter啟動

main

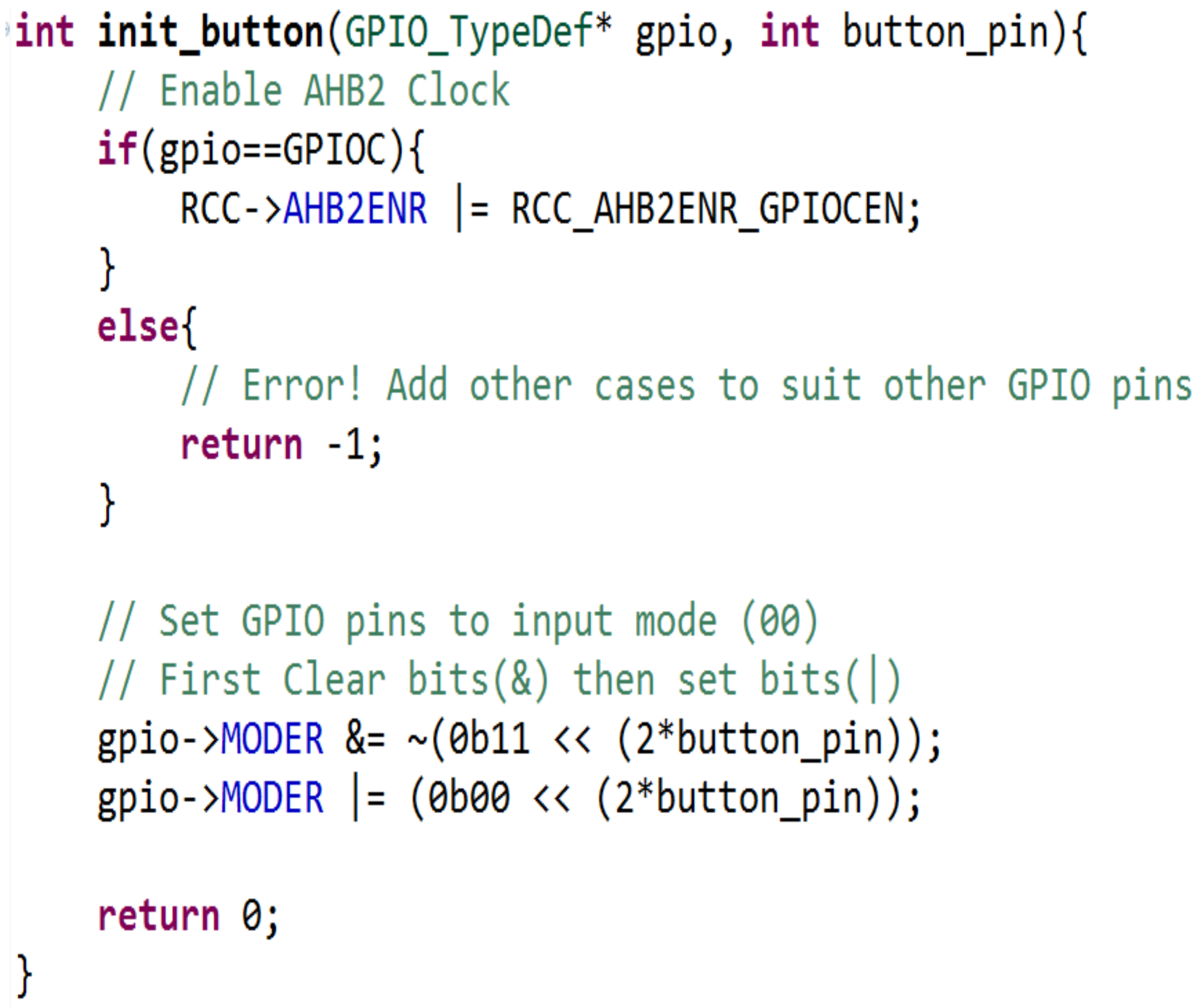
- 接下來的while loop就交給你們寫了

FPU_init



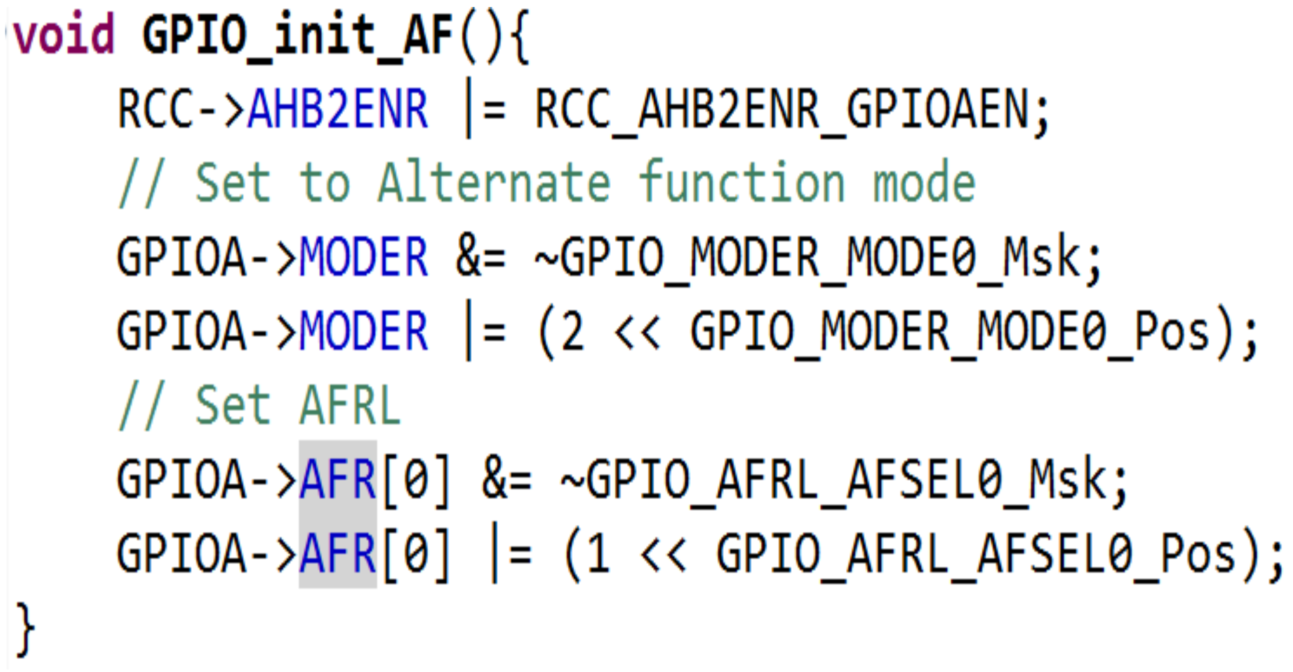
```
void FPU_init(){
    // Setup FPU
    SCB->CPACR |= (0xF << 20);
    __DSB();
    __ISB();
}
```

init_button



```
int init_button(GPIO_TypeDef* gpio, int button_pin){
    // Enable AHB2 Clock
    if(gpio==GPIOC){
        RCC->AHB2ENR |= RCC_AHB2ENR_GPIOCEN;
    }
    else{
        // Error! Add other cases to suit other GPIO pins
        return -1;
    }

    // Set GPIO pins to input mode (00)
    // First Clear bits(&) then set bits(|)
    gpio->MODER &= ~(0b11 << (2*button_pin));
    gpio->MODER |= (0b00 << (2*button_pin));

    return 0;
}
```

GPIO_init_AF()

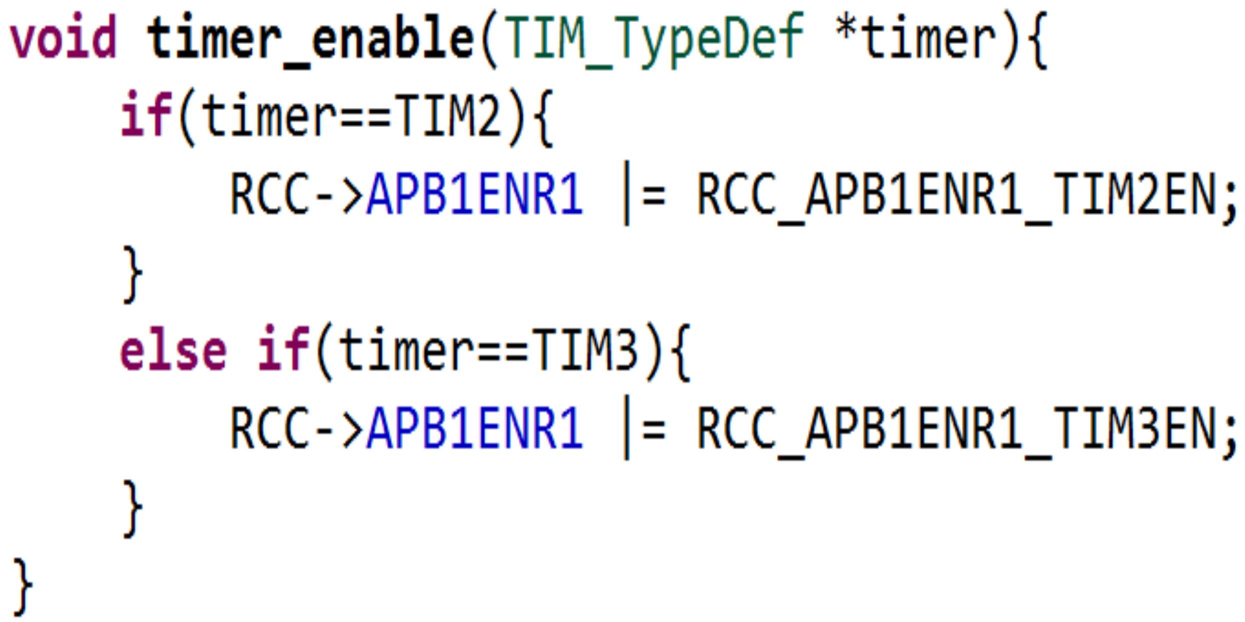


```c
void GPIO_init_AF(){
    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;
    // Set to Alternate function mode
    GPIOA->MODER &= ~GPIO_MODER_MODE0_Msk;
    GPIOA->MODER |= (2 << GPIO_MODER_MODE0_Pos);
    // Set AFRL
    GPIOA->AFR[0] &= ~GPIO_AFRL_AFSEL0_Msk;
    GPIOA->AFR[0] |= (1 << GPIO_AFRL_AFSEL0_Pos);
}
```

⇦ 把A0設成AF mode

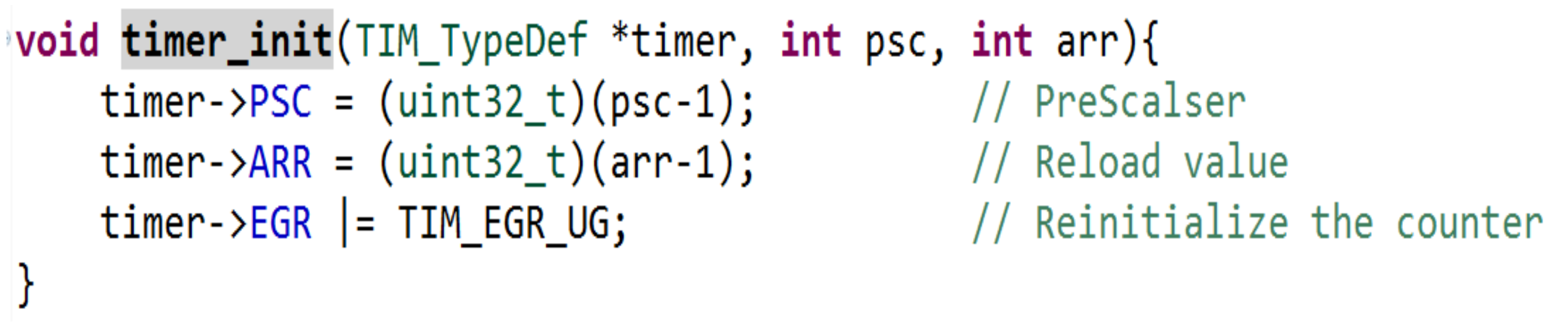
⇦ 把A0設成AF1

timer_enable



```
void timer_enable(TIM_TypeDef *timer){
    if(timer==TIM2){
        RCC->APB1ENR1 |= RCC_APB1ENR1_TIM2EN;
    }
    else if(timer==TIM3){
        RCC->APB1ENR1 |= RCC_APB1ENR1_TIM3EN;
    }
}
```

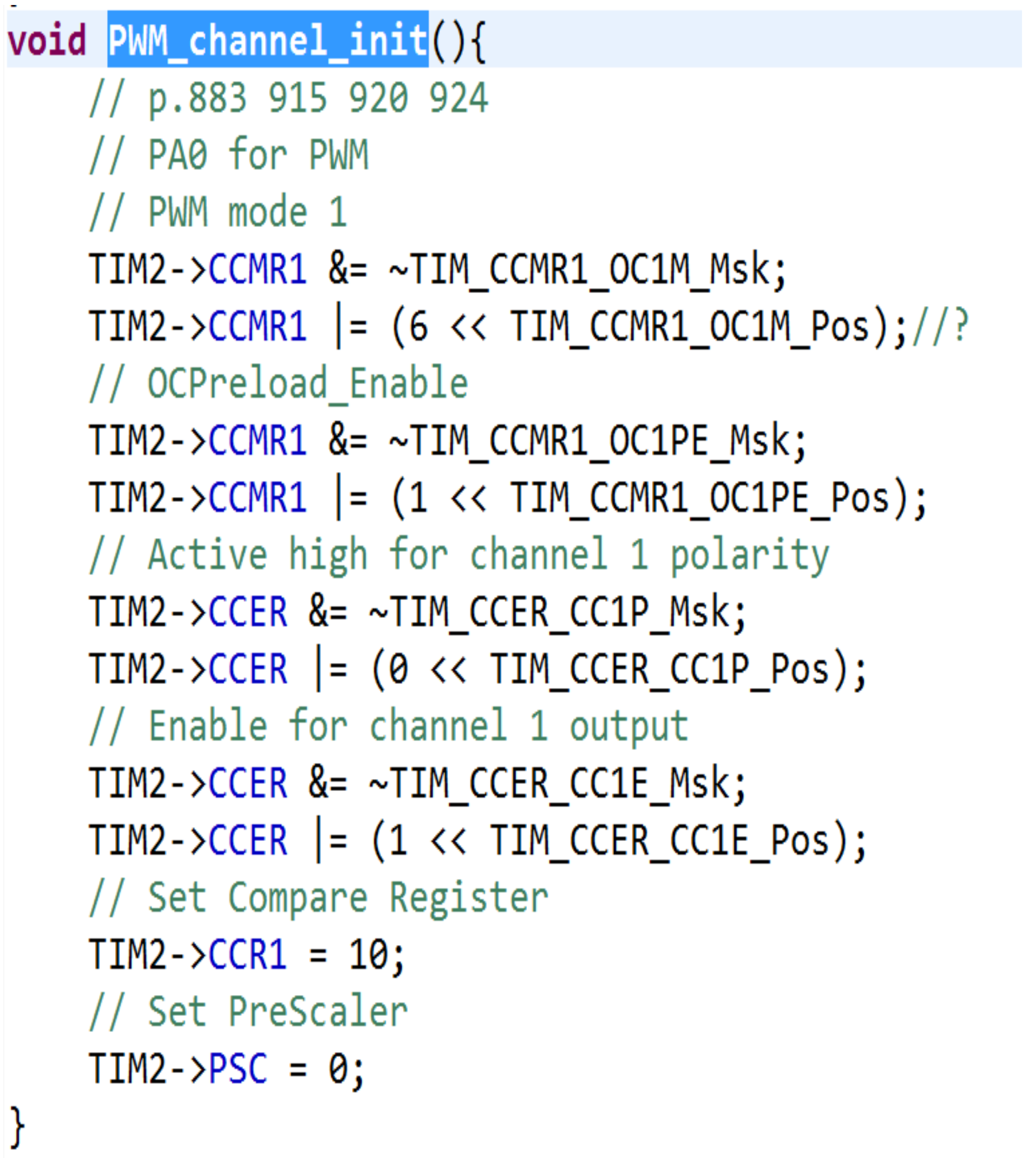
timer_init



```
void timer_init(TIM_TypeDef *timer, int psc, int arr){
    timer->PSC = (uint32_t)(psc-1);          // PreScalser
    timer->ARR = (uint32_t)(arr-1);          // Reload value
    timer->EGR |= TIM_EGR_UG;                // Reinitialize the counter
}
```

PSC :counter 多久數一次
ARR:屬到多少要reload
EGR:自動reinitialize counter

PWM_channel_init



```
void PWM_channel_init(){
    // p.883 915 920 924
    // PA0 for PWM
    // PWM mode 1
    TIM2->CCMR1 &= ~TIM_CCMR1_OC1M_Msk;
    TIM2->CCMR1 |= (6 << TIM_CCMR1_OC1M_Pos);//?
    // OCPreload_Enable
    TIM2->CCMR1 &= ~TIM_CCMR1_OC1PE_Msk;
    TIM2->CCMR1 |= (1 << TIM_CCMR1_OC1PE_Pos);
    // Active high for channel 1 polarity
    TIM2->CCER &= ~TIM_CCER_CC1P_Msk;
    TIM2->CCER |= (0 << TIM_CCER_CC1P_Pos);
    // Enable for channel 1 output
    TIM2->CCER &= ~TIM_CCER_CC1E_Msk;
    TIM2->CCER |= (1 << TIM_CCER_CC1E_Pos);
    // Set Compare Register
    TIM2->CCR1 = 10;
    // Set PreScaler
    TIM2->PSC = 0;
}
```

Output 設成PWM mode 1

CCR1 enable

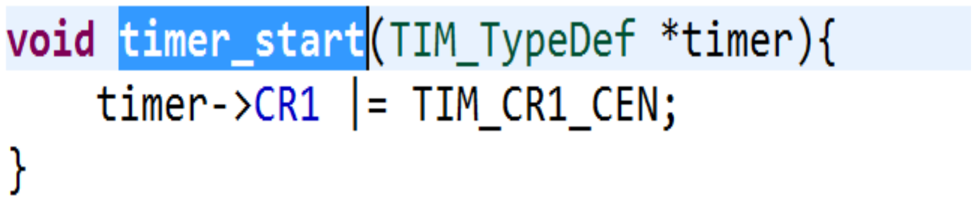
把channel 1設為 active high

把channel 1 enable

決定 duty cycle

決定 frequency

timer_start



```c
void timer_start(TIM_TypeDef *timer){
    timer->CR1 |= TIM_CR1_CEN;
}
```

Counter enable