Microprocessor Systems

ELE 271

Laboratory 7:

Timers

Due 03APR2023

By: Nick Bean ,Taylor Brookes, Toussaint Falangani

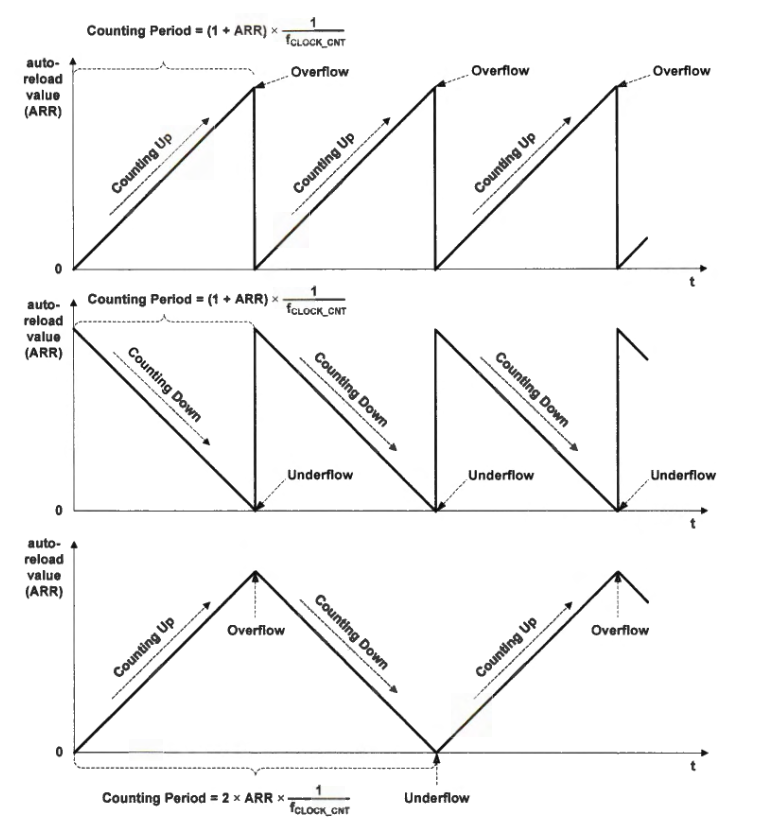
**Textbook Background**

Before delving into the lab, it is helpful to have some background knowledge about timers. A timer is a type of hardware counter that increments or decrements once every clock cycle. The timer counter (CNT) has 16 bits. The capture/compare register (CCR) holds a value that is compared to the timer counter. When a timer is in input capture mode, the timer logs the counter value into the CCR and generates an interrupt when a specific event occurs.

There are three different counting modes for the hardware timer counter: up-counting, down-counting, and center-aligned counting. In up-counting mode, the counter starts from 0 to a constant value, known as the auto-reload register (ARR), and then restarts from 0. In the down-counting mode, the counter starts from the ARR value and counts down to 0 before restarting from the ARR value. In the center-aligned counting mode, the counter alternates between up-counting and down-counting.

The period of the timer is determined by both the clock frequency of the counter (fclock\_cnt) and the ARR value. For up-counting and down-counting, the period of the sawtooth waveform is (1+ARR) \* 1/fclock\_cnt. For center-aligned counting, the period of the triangle waveform is 2 \* ARR \* 1/fclock\_cnt.

[up-counting, down-counting, and center-aligned counting periods]



The timer counter (CNT) in a clock can be slowed down by a prescaler (PSC) to create a longer output period. The output of a channel (OCREF) can be programmed to become active when the timer counter (CNT) is equal to the compare value (CCR). The polarity bit determines whether the active high or active low voltage output will be used. Active high means that when the logic is high, the voltage is high and when the logic is low, the voltage is low, and active low is the reverse. For OCREF, active high is always used, but the actual output (OC or OCN) can be active high or active low. A timer channel can have two outputs: the main output (OC) and the complementary output (OCN), which are determined by exclusive-OR between the channel reference output (OCREF) and its corresponding polarity bit in the capture/compare enable register (CCER). The polarity bit of OC and OCN are represented by the capture/compare/PWM (CCP) and compare/capture n output polarity (CCNP) bits in the CCER, respectively.

• If only OC or OCN is enabled:

OC = OCREF + Polarity bit for OC

OCN = OCREF + Polarity bit for OCN

• If both OC and OCN are enabled:

OC = OCREF + Polarity bit for OC

OCN = (not OCREF) +Polarity bit for OCN

By default for our STM32, the fclock\_psc is 80MHz. As an example, for standard up-counting, to achieve a fclock\_cnt of 2kHz, we set PSC to 39,999 (40,000 - 1). PSC, ARR, and CNT are all 16-bit values, which means their maximum value is 65535 (2^16 - 1). To achieve a 1-second on and 1-second off 50% duty cycle with our PSC value, we set ARR to 1999 (2000 - 1), and choose any value between 0 and ARR as CNT.

Pulse Width Modulation (PWM) is a method for regulating an analog variable's value. PWM involves utilizing a rectangular waveform to rapidly switch a voltage source on and off to produce a required average voltage output. Although the output is binary at any moment, the average output over a period can range from 0 to the maximum voltage. The PWM switching frequency for an LED light must be at least 120 Hz to prevent the human eye from detecting flickering effects. The PWM output signal is determined by three factors:

1. Comparison between the timer counter (CNT) (the system clock ticking) and the given reference valued stored in the compare and capture register (CCR),
2. The PWM output mode, and
3. The polarity bit.

There are two PWM modes, which work for both up-counting and down-counting:

1. PWM Mode 1: If the counter (CNT) is less than the reference signal (CCR), the timer reference output (OCREF) is then held at logic high; otherwise, it is held at logic low.
2. PWM Mode 2: The timer reference output (OCREF) in mode 2 is the opposite of mode 1. If the counter (CNT) is greater than the reference signal (CCR), OCREF is then held at active; otherwise, OCREF is held at inactive.

In up/down counting: PWM period = (1 + PSC) \* (1 + ARR) / fclock\_cnt

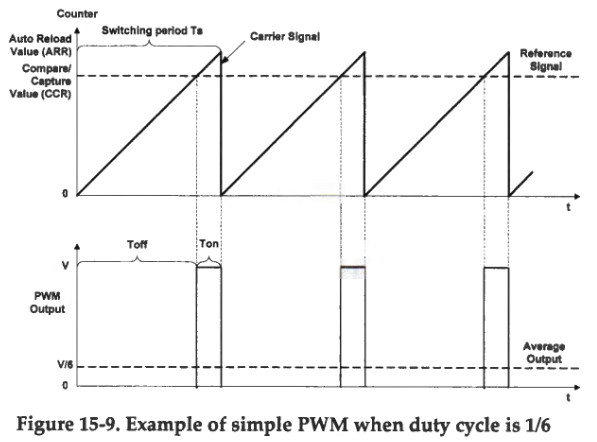
In center-counting: PWM period = (1 + PSC) \* 2 \* ARR / fclock\_cnt  
OC when polarity = 0, OC duty cycle = 1 - CCR / (ARR + 1)

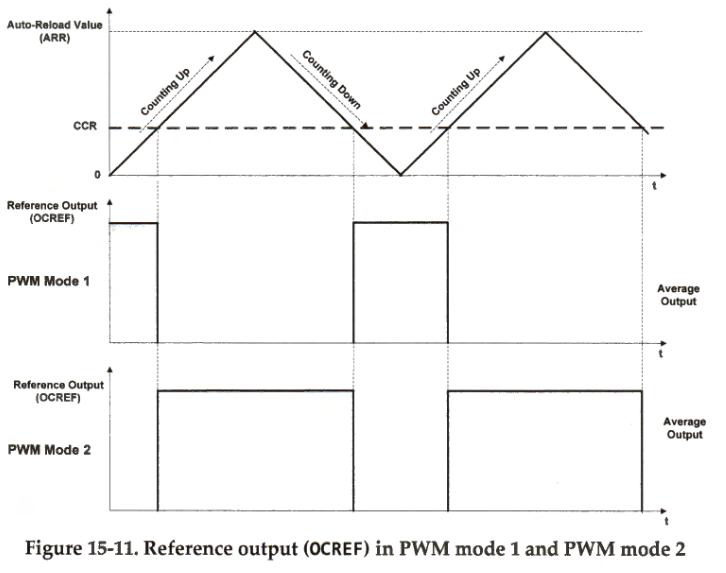
OC when polarity = 1, OC duty cycle = CCR / (ARR + 1)

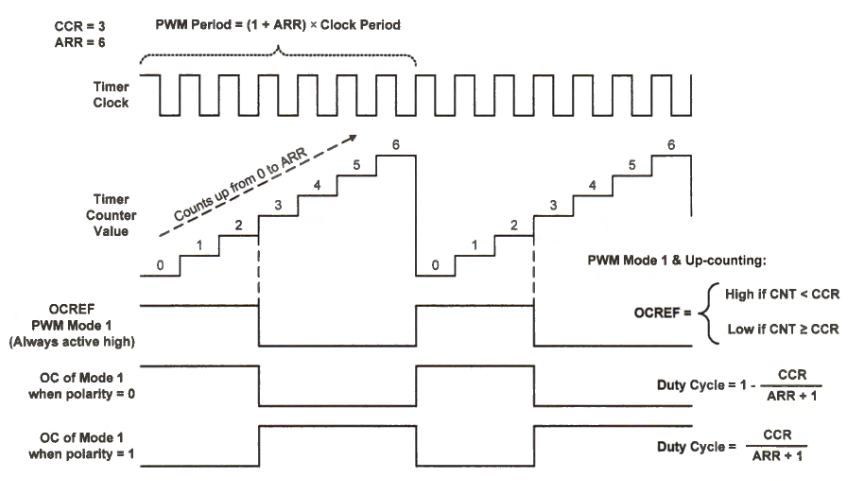
polarity bit = 0, PWM signals are called left-edge aligned (or rising-edge aligned)

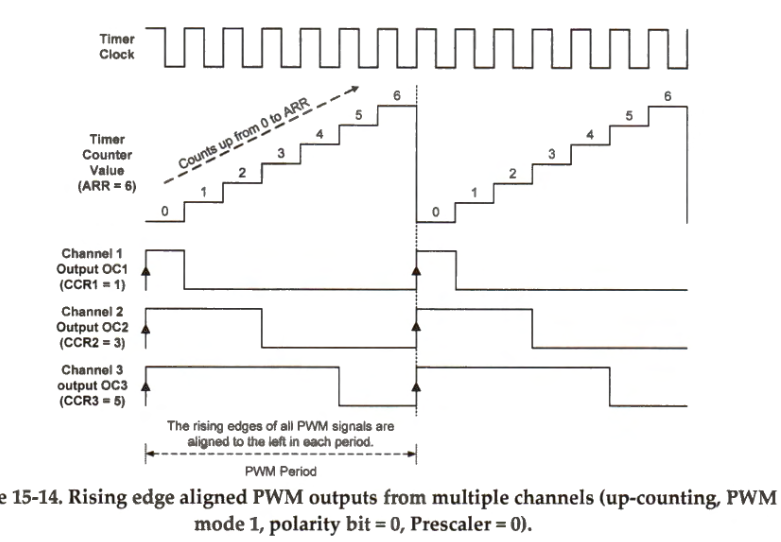
polarity bit = 1, PWM outputs become right-edge aligned, or called falling-edge aligned

Duty cycle = TimeOn / (TimeOn + TimeOff) \* 100%









**Introduction**

The purpose of this lab is to use timers to perform basic measurement operations.

* In part 1, we toggle an LED on PA.5 using Timer 2 (instead of a counter inside of a for-loop) via AF1 at 1Hz 50% duty cycle and output the waveform via the logic analyzer.
* In part 2, we switch from up-counting mode to PWM mode 1 and 2 to output 20% and 80% duty cycle 1Hz waveforms respectively and change the values of ARR, CCR1, and CCMR1 to readjust the period and on-off timing.
* In part 3, we output both the default MSI system clock and a 1/10 default MSI system clock frequency. We did this by adjusting PSC, ARR, CCR1 and CCMR1 for PA.5, and creating a config\_clock\_pin method that outputs on PA.8 the default MSI clock.
* For the Question, we are asked to answer what the shortest and longest timing waveform periods are possible using TIM2. This is achieved by selecting the default MSI system clock for the shortest, and calculating default MSI system clock / ((1 + PSC) \* (1 + ARR)).

Here are some keywords:  
AF -- Alternate Function

AFR -- Alternate Function Register

AFSEL -- Alternate Function Select

CNT -- timer counter (incrementing value)

PSC -- prescaler

ARR -- auto-reload register  
CCR -- capture/compare register

CCMR -- capture/compare mode register

CCER -- capture/compare enable register

CR -- control register

PWM -- pulse width modulation

CCP -- capture/compare/PWM

CCNP -- capture/compare n output polarity

MSI -- multi-speed internal

**Part 1**

In Part 1, we are tasked with the problem of using an internal timer to toggle an LED and to capture the timer waveform using the logic analyzer. We are given an outline on how to do so, and used this along with textbook reference and reference manual usage to create our configuration method. The following will be a step-by-step layout of how we configured this to happen, with reference manual usage (for x = 2 to 5, where x is TIMx and our value is 2) added throughout, with the textbook example at the end.

1. To begin, we reused most of the code from our previous config\_LED methods. What has been changed are MODER and AFR. MODER is set to toggle mode (10) instead of output mode (01), and AFR is set to AF1. MODER being in toggle mode means that PA.5 output will toggle when CNT = CCR, while AF1 allows us to use TIM2\_CH1 for PA.5.
2. Next, we included PSC, ARR, and CCR1. PSC scales the default clock down to a desired speed, ARR is how high the counter should count until (zero indexed), and CCR1 is the value that the toggle occurs on. Our MSI system clock frequency is 4Mhz instead of the highest possible 80Mhz clock speed. We are unsure how to achieve 80Mhz.
3. Next, we enabled Timer 2 clock with RCC\_APB1ERN1\_TIM2EN.
4. Next, we set the timer to toggle mode (0011) for bits 4 and 5.
5. Next, we enabled the output by setting bit 0 to value 1.
6. Next, we started the timer by setting bit 0 to value 1.
7. Finally, we start the timer.
8. Configure LED pin

void config\_LED(void) {

RCC -> AHB2ENR |= RCC\_AHB2ENR\_GPIOAEN; // GPIO Port A clock enabled

GPIOA -> MODER &= ~(3UL<<10); // reset MODER value

GPIOA -> MODER |= 2UL<<10; // Set the GPIO Port A mode to AF (10)

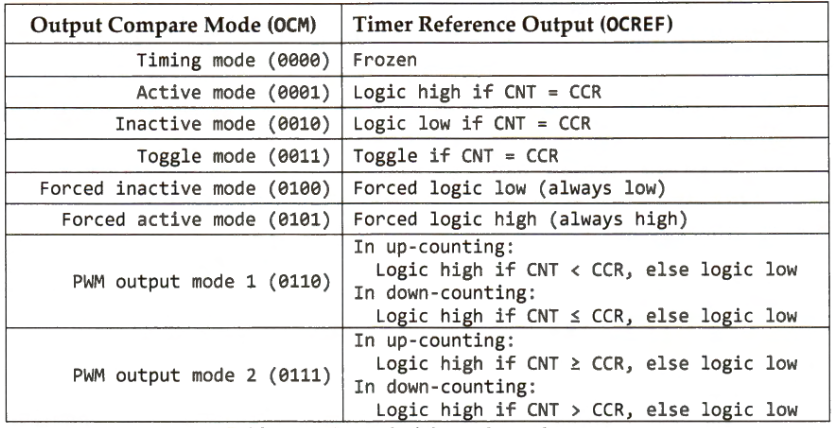
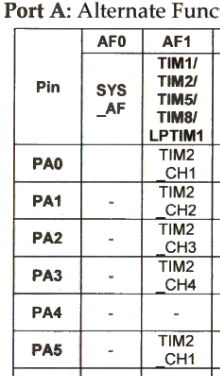
GPIOA -> AFR[0] |= 1<<20; // pg 309, Set PA.5 to AF[1]

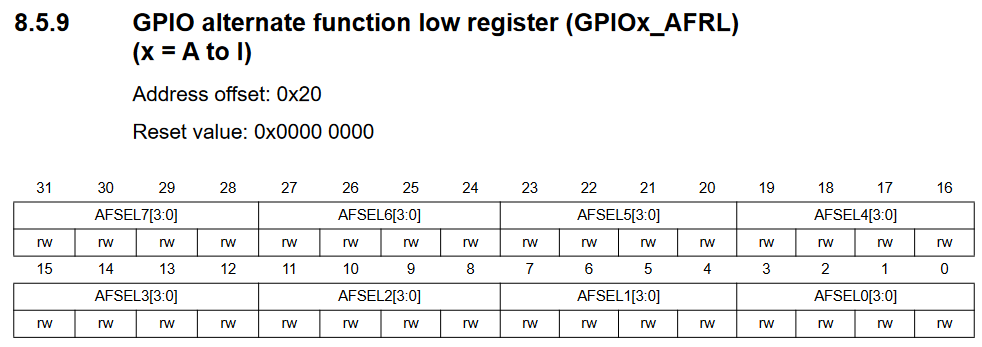
// AFSEL5[3:0] is pin 5, bits 0-3, AF1 is 0001

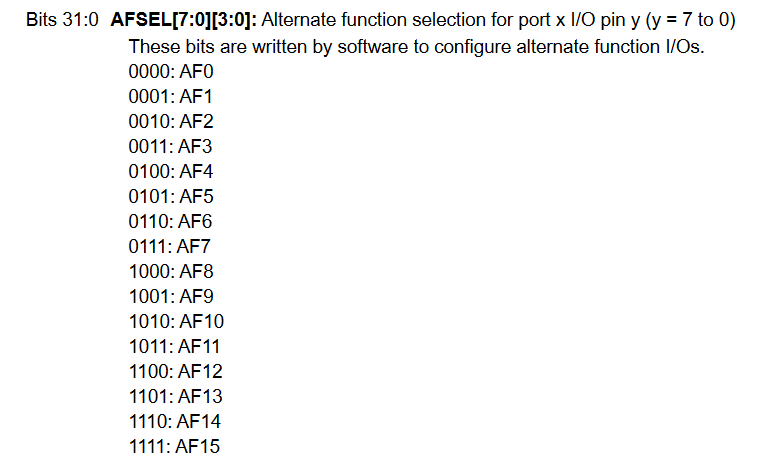
GPIOA -> OSPEEDR &= (0UL<<10); // low speed

GPIOA -> PUPDR &= ~(3UL<<10); // no pull-up pull-down

[reference images of TIM2\_CH1 location, toggle mode bit setting, AFSEL5 byte location for PA.5, AF1 bit setting]







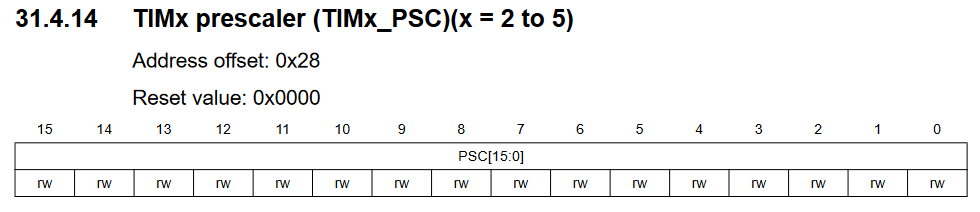
1. Use registers TIM2\_PSC, TIM2\_ARR and TIM2\_CCR1 to create a timing waveform with a period of 1 sec (frequency 1 Hz) from the default system clock of 4 MHz.

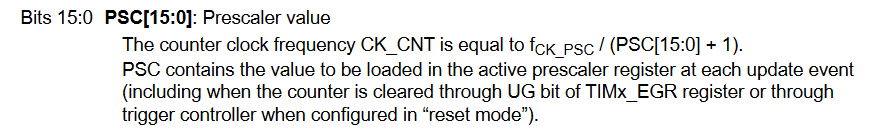
TIM2 -> PSC = 3999; // pg1077/1903, pre-scaling clock by an amount (4Mhz/(PSC + 1))

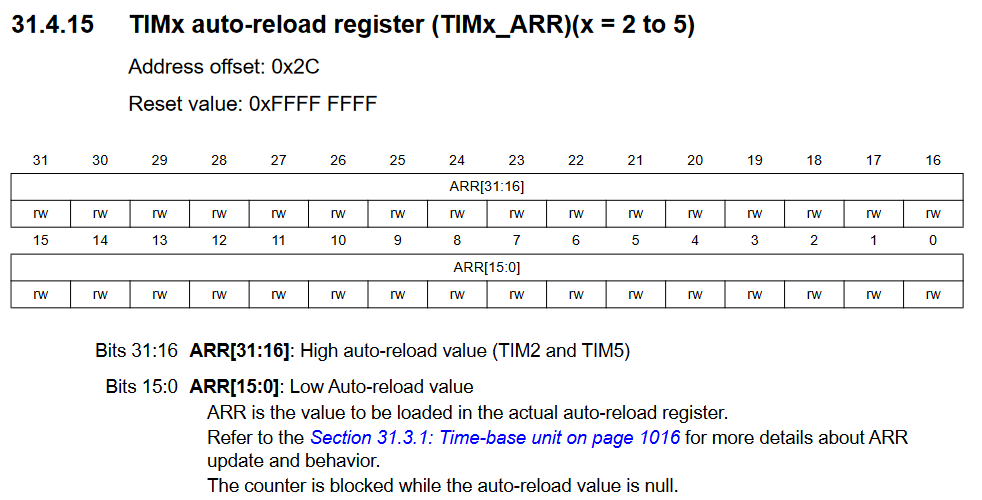
TIM2 -> ARR = 499; // pg1077, up-counting goes from 0 to auto-reload value

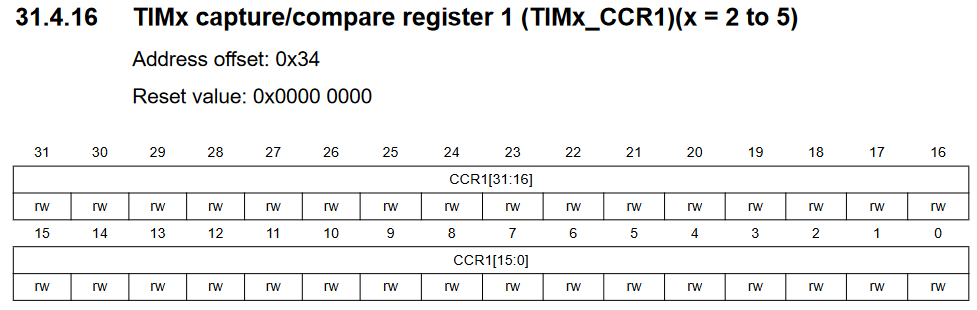
TIM2 -> CCR1 = 99; // pg1078, the value that the timer switches on, value btw 0 and ARR

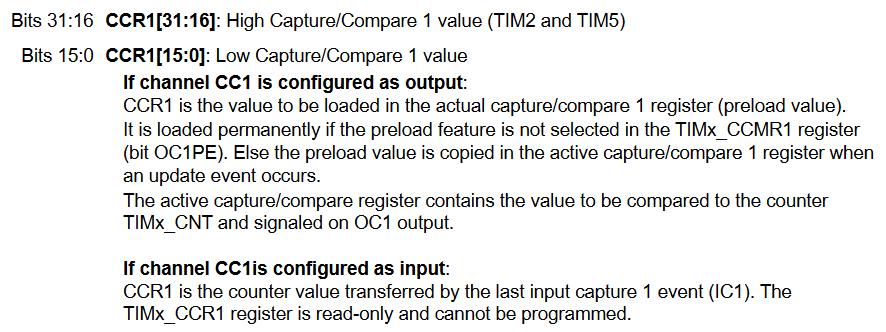
[reference images of PSC values, ARR values, and CCR1 values]











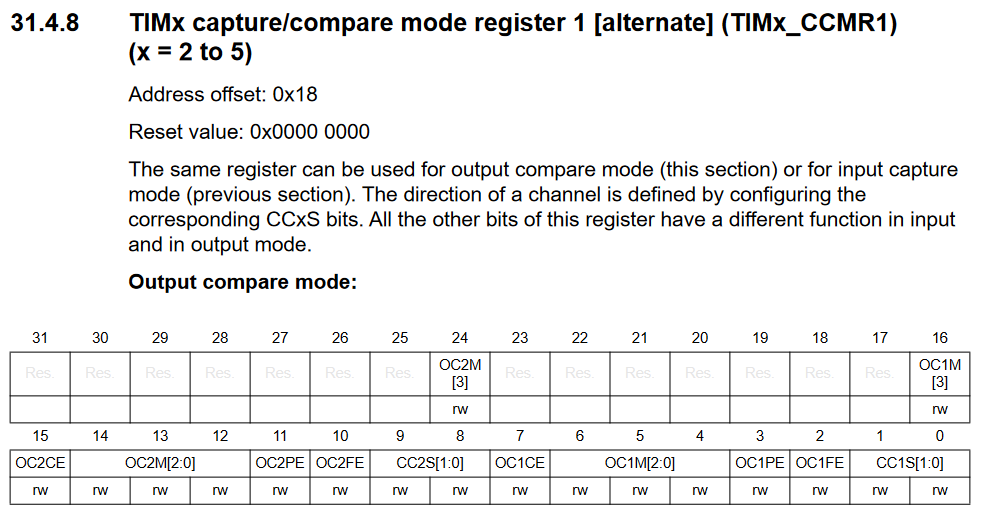
1. Enable Timer 2 clock (check which bus and register needs to be configured)

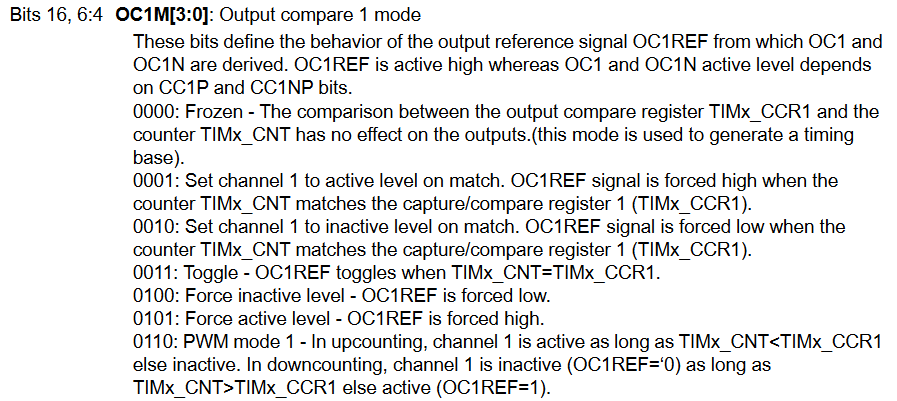
RCC -> APB1ENR1 |= RCC\_APB1ENR1\_TIM2EN; // enable timer 2 clock

1. Set the timer to output mode (see register TIM2\_CCMR1)
2. In the same register, set the timer to toggle mode to obtain a symmetric timing waveform (50% duty cycle)

TIM2 -> CCMR1 |= (3<<4); // pg1071 OC1M 0011, toggle mode, bit 4+5+6+16

[reference images of CCMR1 OC1M bit locations and bit settings]

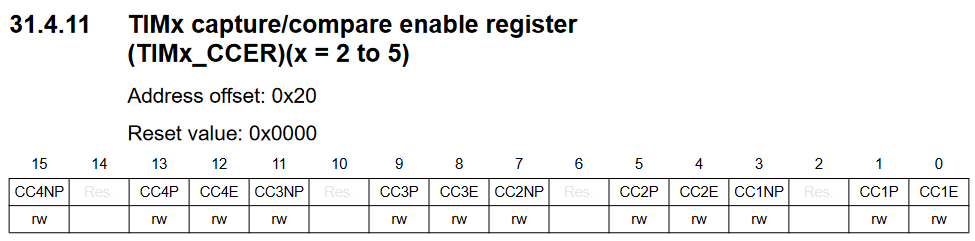


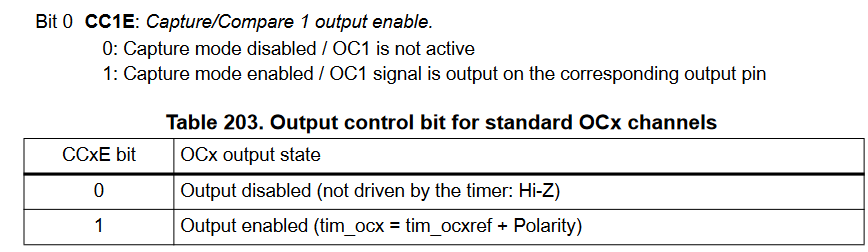


1. Enable the output of channel 1 (see register TIM2\_CCER)

TIM2 -> CCER = 1<<0; // pg1075, bit 0, OC1 is output to output pin

[reference images of CCER reference locations and OC1 active bit setting]





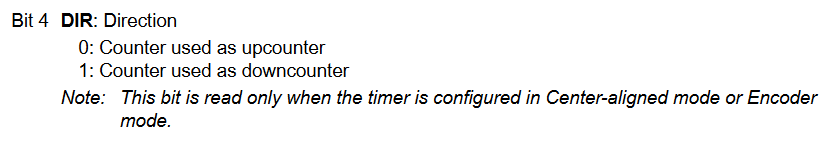
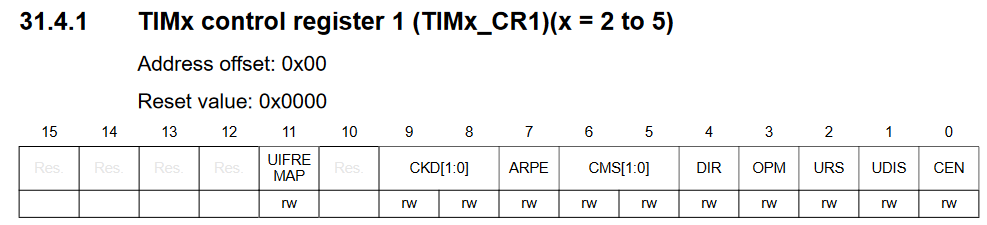
1. Start the timer (see register TIM2\_CR1)

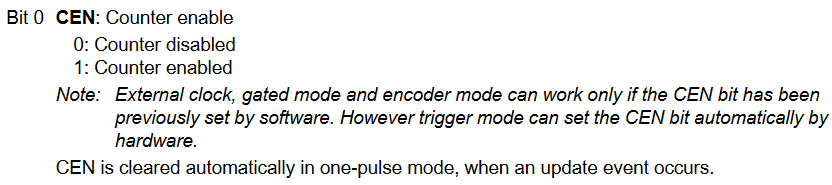
TIM2 -> CR1 = 0<<4; // up-counting mode

TIM2 -> CR1 = 1<<0; // pg 1060, enable counter AKA timer 2

}

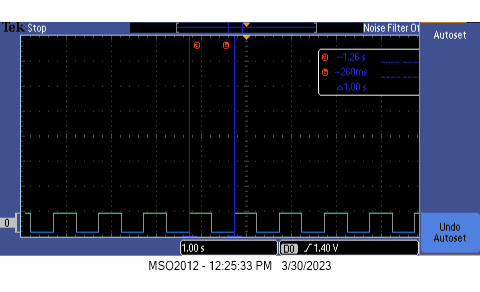
[reference image of CR1 value locations, DIR and CEN values]





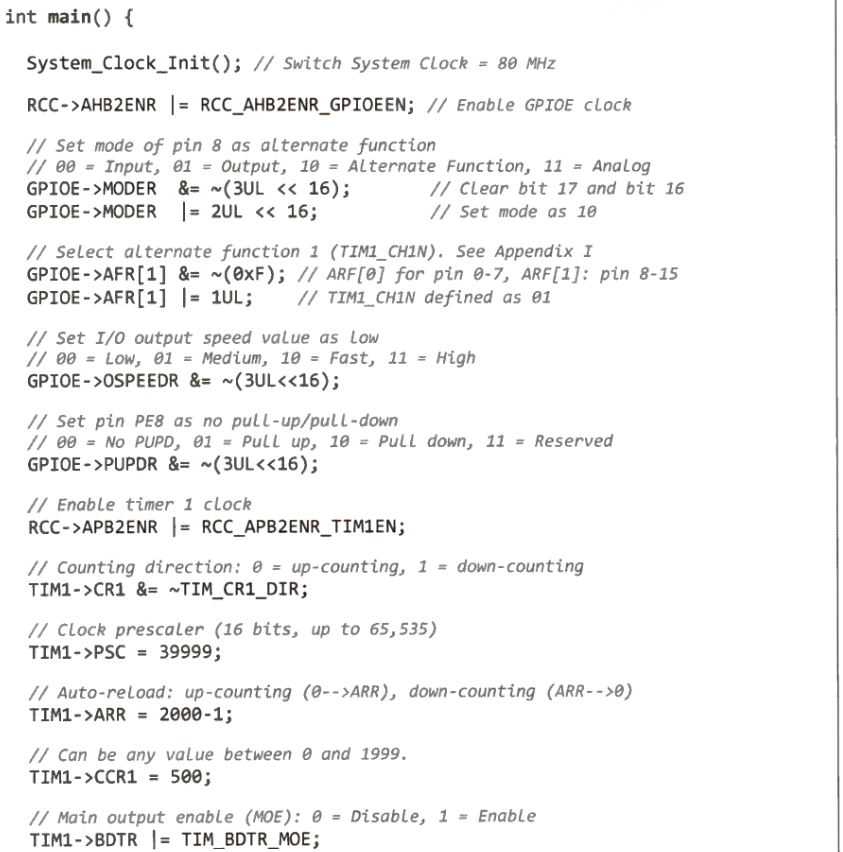
Below is our timing waveform captured with the logic analyzer. As is evident, the frequency is set to 1Hz as requested.

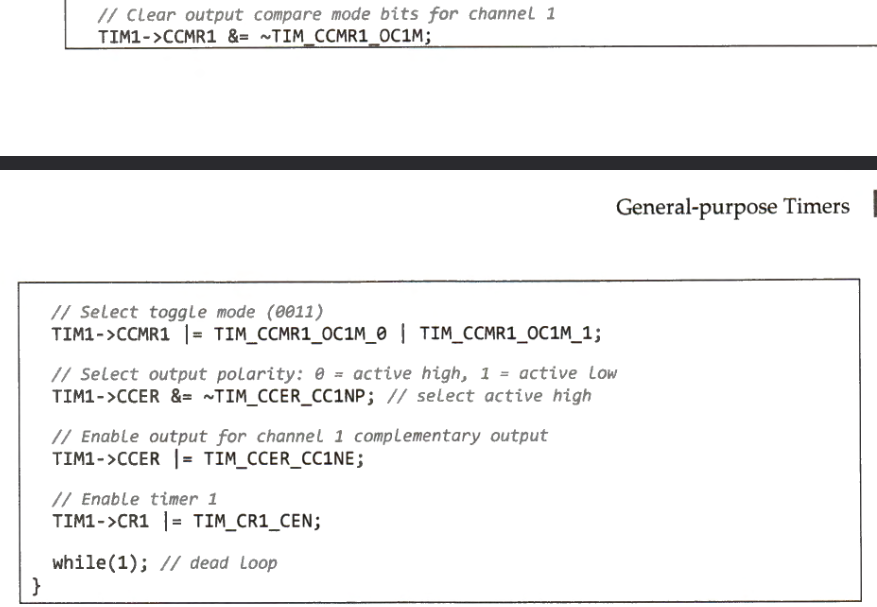
[1Hz timing waveform]



Below is the textbook configuration example.

[textbook example reference]





**Part 2**

Part 2 asks us to switch TIM2 to PWN mode, and then create a 20% duty cycle 1Hz waveform, and then create a 80% duty cycle 1Hz waveform. We changed our method by changing ARR to 999, CCR1 to 199, and PWM mode to mode 1 (bit 4+5+6+16 set to 0110). ARR is changed to 999 from 499 because we are going from up-counting to center-aligned counting (PWM), which will count from 0 to 499 then back to 0 instead of from 0 to 999 then immediately start from 0 back to 999 as in up-counting. CCR1 is set to 199 because that is 20% of the 1000 period. We then show that we obtained a 20% duty cycle through the oscilloscope. To obtain a 80% duty cycle, we changed PWN mode to mode 2 (bit 4+5+6+16 set to 0111). This changes it from activating on high to activating on low, reversing the polarity. Effectively the same thing can also be accomplished by setting CCR1 to 799, 80% of 1000, however this will output a left-aligned 80% duty instead of right-aligned 80% duty. For our purposes, either can be used.

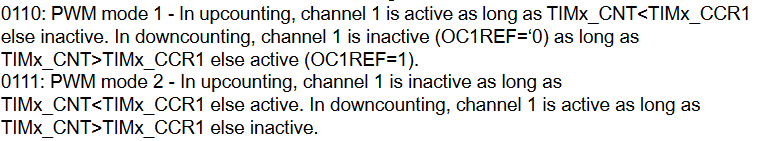
TIM2 -> ARR = 999;

TIM2 -> CCR1 = 199;

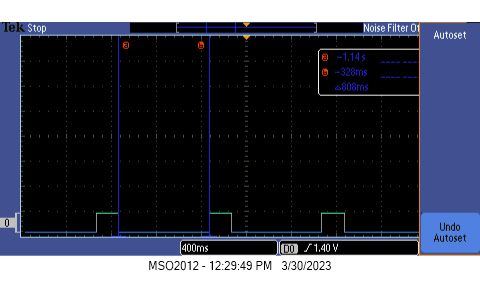
TIM2 -> CCMR1 |= (6<<4); // PWM mode 1

TIM2 -> CCMR1 |= (7<<4); // PWM mode 2

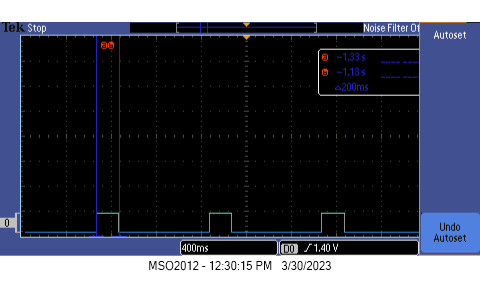
[textbook PWM mode 1 and 2 from CCMR1 reference on Part 1 bullet 5 (pg12)]



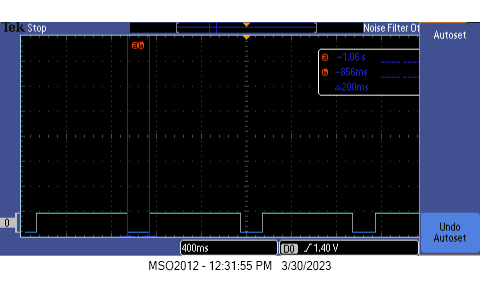
[reference image of low value 20% duty cycle, 800ms]



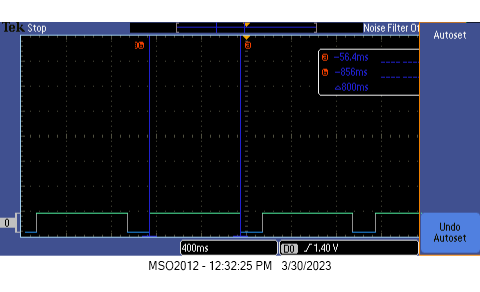
[reference image of high value 20% duty cycle, 200ms]



[reference image of low value 80% duty cycle, 200ms]



[reference image of high value 80% duty cycle, 800ms]



**Part 3**

Part 3 asks us to change the timing waveform frequency to 1/10 of default MSI system clock. Default MSI system clock is 4Mhz. That means we are looking to output 400kHz as well as the default MSI system clock of 4Mhz. We did this by changing PSC to 0, ARR to 4, CCR1 to 0, CCMR1 to toggle mode, and adding a config\_clock\_pin method from lab 6. Setting PSC to 0 means that the default clock is not scaled. CCMR1 toggle mode means that it will toggle on CCR1, giving us a period of 1/(2 \* (ARR + 1)) of the default MSI system clock. Setting ARR to 4 gives us a period of 1/10 of the default MSI system clock. The config\_clock\_pin method sets pin PA.8 to output the default MSI system clock, while pin PA.5 outputs the 1/10 scaled clock.

As shown by our logic analyzer output, the period of the 1/10 MSI system clock is 2.5 microseconds, which is 400kHz. It can also be seen that within 1 period of that output there exists 10 periods of the default MSI system clock, confirming that it has a 4MHz frequency.

TIM2 -> PSC = 0;

TIM2 -> ARR = 4;

TIM2 -> CCR1 = 0;

TIM2 -> CCMR1 |= (3<<4);

void config\_clock\_pin(void) {

RCC -> AHB2ENR |= RCC\_AHB2ENR\_GPIOAEN;

RCC -> AHB2ENR |= 0x00000001;

GPIOA -> MODER &= ~(3UL<<16);

GPIOA -> MODER |= 2UL<<16; // 01 general output , 10 alternate function

GPIOA -> OTYPER &= ~(1UL<<8);

GPIOA -> OSPEEDR &= ~(3UL<<16);

GPIOA -> PUPDR &= ~(3<<16);

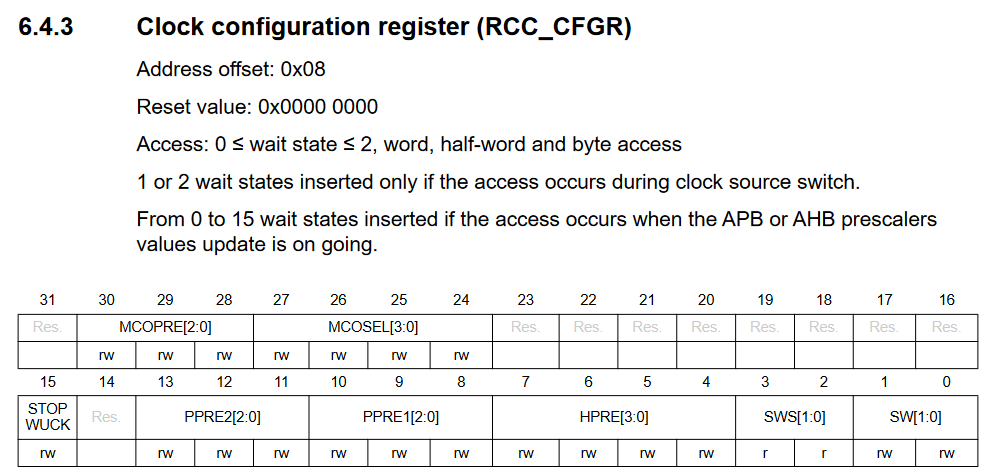
GPIOA -> AFR[1] &= ~(7<<0); // fill value with 0000 (for AF0) (pg311/1903)

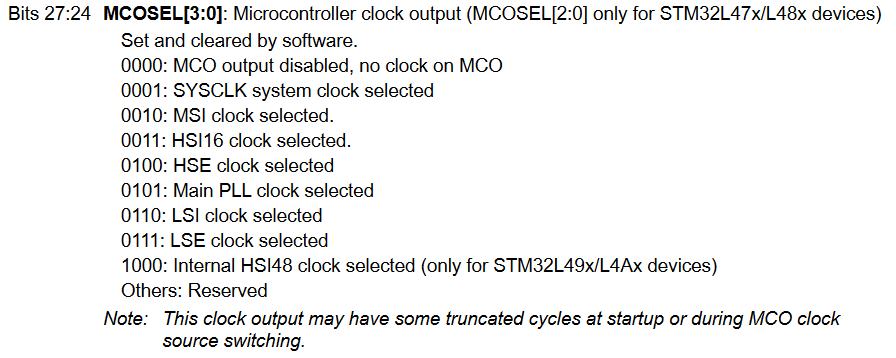
RCC -> CFGR &= ~(7<<24);

RCC -> CFGR |= (2<<24); // MCOSEL[3:0] = 0010 // MSI clock selected (pg 227/1903)

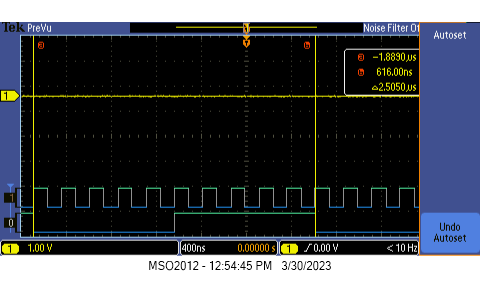
}

[reference of CFGR bit location and value for MSI clock]





[reference image of default MSI system clock frequency and 1/10 of it]



**Question**

The shortest timing waveform that can be obtained from the default system clock using general-purpose timer 2 is the default MSI system clock waveform itself, 4Mhz.

The longest timing waveform that can be obtained from the default system clock using general-purpose timer 2 is obtained from using the max values of PSC and ARR to lengthen the period. Because these are both 16-bit values, we can lengthen the clock using the period formula to: MSI clock / ((1 + PSC) \* (1 + ARR)) = 4MHz / (2^16 \* 2^16) = 1.34 cycles / day.

**Conclusion**

In our lab, we used timers to blink an LED and learned to use variables to change timer inverals. In Part 1, we used pin PA.5 to blink an LED at 1Hz, 50% duty cycle. This was accomplished by scaling the default MSI clock down to 1Hz via PSC, ARR, and CCR1 and outputting it on our logic analyzer. In Part 2, we used PWM mode 1 and 2 to output 20% and 80% duty cycle 1Hz waveforms and output them on our logic analyzer. In Part 3, we output both the default MSI system clock and a 1/10th scaled version on our logic analyzer at the same time. We answered the Question by showing that the fastest clock speed possible is that of the default MSI clock speed, and the slowest takes almost 1 day to complete and is scaled down using PSC and ARR, which are both 16-bit values. The reference manual, however, seems to say that there exists ARR low and ARR high, which, when combined, is a 32-bit value, but we are unsure if this is usable.