Beverly Yee

u0770041

Prelab 03

1. List two things you can learn from a peripheral’s functional description in the peripheral reference manual?

**Basic theory of operation.**

**Purpose of each operating mode.**

1. What is the title of the first sub-section in the functional description of timers 2 and 3?

**Hint:** Not mentioned in the lab manual; look it up!

**Time-base unit.**

1. What is the purpose of the Prescaler (PSC) register?

**Divide the input clock frequency to the timer. It can divide by any integer value that fits a 16-bit width and is 0-indexed (count starts at 0 instead of 1).**

1. What is the purpose of the Auto-Reload (ARR) register?

**Create a trigger point with a value such that when the timer hits that value, it resets and begins a new count period.**

1. What is the purpose of the Capture/Compare (CCRx) register while the timer is operating in Output Compare mode?

**Measures the output of a GPIO pin whenever the timer’s counter matches the value stored in the register.**

1. What does the duty-cycle of a PWM signal represent?

**An analog voltage that ranges between the lowest and highest voltage of a digital signal. The ratio of the on-time and off-time in a square wave.**

1. What is the purpose of the Alternate Function mode for a GPIO pin?

**Allow a pin to connect directly to the internal peripherals of the STM32FO board.**

1. In what document can you find the documentation for what GPIO pins have which alternate functions?

**The datasheet specific to the board used.**

3.1 — Using Timer Interrupts.

In this exercise, you’ll set up a timer such that the update event (UEV) triggers an interrupt at a specific period. Timer peripherals allow for greater flexibility in choosing an interrupt period over manually counting in the SysTick handler. To complete this exercise, carefully review the control register maps in the peripheral reference manual to determine the proper option bits to set and reset.

1. Enable the timer 2 peripheral (TIM2) in the RCC.

* This lab will use timers 2 and 3; while all of the timer peripherals can produce interrupts, their configuration registers often have slight differences.

1. Configure the timer to trigger an update event (UEV) at 4 Hz.

* The default processor frequency of the STM32F072 is 8 MHz. Use this value when calculating the timer parameters.
* A typical approach is to set the timer’s base frequency and the target period into the same units.
  + See example 3.1 and use the prescaler (PSC) register.
* Once the timer and target period are in similar units, set the auto reload register (ARR) to count the number of units to reach the target.
  + Pay attention to the size (in bits) of the timer. For example, a 16-bit timer can only count up to 65535. If your target ARR is outside of that range, you’ll need to adjust the prescaler (change units) to scale the ARR appropriately.

1. Configure the timer to generate an interrupt on the UEV event.

* Use the DMA/Interrupt Enable Register (DIER) to enable the update interrupt.

1. Configure and enable/start the timer

* Although the RCC enabled the timer’s clock source, the timer has its own enable/start bit in the control registers.
* Note that you should not enable a timer until you’ve finished setting all the basic parameters and options. This is different than the RCC enable, which you should always enable first!)

1. Set up the timer’s interrupt handler and enable in the NVIC.

* The timer will only have a single generic interrupt reserved in the vector table (not a specific interrupt about the UEV).
* Look for an interrupt containing the name of the timer you are using.

1. Toggle between the green (PC8) and orange (PC9) LEDs in the interrupt handler.

* Remember to initialize the LED GPIO pins in your main function.
* To get the alternating flash pattern, set one of the LEDs active in the GPIO initialization.
* Don’t forget to clear the pending flag for the update interrupt in the status register.

1. Compile and load the application onto the Discovery board.
2. Measure the timing between the UEV interrupt with the logic analyzer (i.e., verify that the cycle is roughly 4 Hz). Make sure to save a screenshot for your postlab

/\* USER CODE BEGIN Header \*/

/\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @file : main.c

\* @brief : Main program body

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @attention

\*

\* Copyright (c) 2024 STMicroelectronics.

\* All rights reserved.

\*

\* This software is licensed under terms that can be found in the LICENSE file

\* in the root directory of this software component.

\* If no LICENSE file comes with this software, it is provided AS-IS.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\* Includes ------------------------------------------------------------------\*/

#include "main.h"

/\* Private function prototypes -----------------------------------------------\*/

void SystemClock\_Config(void);

void SetPins(void);

void SetTimers(void);

/\*\*

\* @brief The application entry point.

\* @retval int

\*/

int main(void)

{

/\* MCU Configuration--------------------------------------------------------\*/

/\* Reset of all peripherals, Initializes the Flash interface and the Systick. \*/

HAL\_Init();

/\* Configure the system clock \*/

SystemClock\_Config();

SetPins();

SetTimers();

/\* Infinite loop \*/

while (1)

{

}

}

/\*

\* timer2 interrupt handler (3.1 Q5)

\*/

void TIM2\_IRQHandler(void){

// toggle green and orange LEDs when the interrupt is triggered

GPIOC->ODR ^= GPIO\_ODR\_8;

GPIOC->ODR ^= GPIO\_ODR\_9;

// clear the pending flag in SR (3.1 Q6)

TIM2->SR &= ~0x00000001;

}

/\* timer setup function \*/

void SetTimers(void){

// enable the timer clock

RCC->APB1ENR |= RCC\_APB1ENR\_TIM2EN;

// configure timer to trigger an update event at 4Hz (default processor frequency = 8MHz)

TIM2->PSC = 7999; // prescaler value = 7999 (3.1 Q2)

TIM2->ARR = 250; // auto-reload value = 250 (3.1 Q2)

TIM2->DIER |= 0x00000001; // enable the update interrupt (3.1 Q3)

TIM2->CR1 |= TIM\_CR1\_CEN\_Msk; // enable the timer (3.1 Q4)

// find the IRQn\_Type for TIM2\_IRQn, enable it (3.1 Q5)

NVIC\_EnableIRQ(TIM2\_IRQn);

}

/\*\*

\* Set the pins according to the lab instructions

\* Helper method so the main method doesn't look overcrowded

\*/

void SetPins(void){

// enable the GPIOC peripheral clock

RCC->AHBENR |= RCC\_AHBENR\_GPIOCEN;

// set the general purpose output for the LEDs

// bits = 01

GPIOC->MODER |= GPIO\_MODER\_MODER8\_0;

GPIOC->MODER |= GPIO\_MODER\_MODER9\_0;

// LEDs have push-pull output type = both bits cleared

GPIOC->OTYPER &= ~(GPIO\_OTYPER\_OT\_8);

GPIOC->OTYPER &= ~(GPIO\_OTYPER\_OT\_9);

// low speed = both bits cleared

GPIOC->OSPEEDR &= ~(GPIO\_OSPEEDER\_OSPEEDR8);

GPIOC->OSPEEDR &= ~(GPIO\_OSPEEDER\_OSPEEDR9);

// no pull-up/down resistors = both bits cleared

GPIOC->PUPDR &= ~(GPIO\_PUPDR\_PUPDR8);

GPIOC->PUPDR &= ~(GPIO\_PUPDR\_PUPDR9);

// set green LED (PC9) to high

GPIOC->ODR &= ~(GPIO\_ODR\_8);

GPIOC->ODR |= GPIO\_ODR\_9;

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

/\*\* Initializes the RCC Oscillators according to the specified parameters

\* in the RCC\_OscInitTypeDef structure.

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_NONE;

if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

{

Error\_Handler();

}

/\*\* Initializes the CPU, AHB and APB buses clocks

\*/

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

|RCC\_CLOCKTYPE\_PCLK1;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_HSI;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV1;

if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_0) != HAL\_OK)

{

Error\_Handler();

}

}

/\* USER CODE BEGIN 4 \*/

/\* USER CODE END 4 \*/

/\*\*

\* @brief This function is executed in case of error occurrence.

\* @retval None

\*/

void Error\_Handler(void)

{

/\* USER CODE BEGIN Error\_Handler\_Debug \*/

/\* User can add his own implementation to report the HAL error return state \*/

\_\_disable\_irq();

while (1)

{

}

/\* USER CODE END Error\_Handler\_Debug \*/

}

#ifdef USE\_FULL\_ASSERT

/\*\*

\* @brief Reports the name of the source file and the source line number

\* where the assert\_param error has occurred.

\* @param file: pointer to the source file name

\* @param line: assert\_param error line source number

\* @retval None

\*/

void assert\_failed(uint8\_t \*file, uint32\_t line)

{

/\* USER CODE BEGIN 6 \*/

/\* User can add his own implementation to report the file name and line number,

ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

/\* USER CODE END 6 \*/

}

#endif /\* USE\_FULL\_ASSERT \*/

3.2 — Configuring Timer Channels to PWM Mode.

This exercise sets up a PWM output to dim the LEDs on the Discovery board: changing the waveform’s duty cycle controls the apparent brightness of the LED. For this exercise you will be using capture/compare channels 1 & 2 of timer 3.

1. Enable the timer 3 peripheral (TIM3) in the RCC.
2. The timer’s update period determines the period of the PWM signal; configure the timer to a UEV period related to 800 Hz (T = 1/f).

* Follow the previous exercise as a template, but do not enable/start the timer.
* Set the prescaler to a reasonable range of counter values between zero and the top limit; otherwise your timer will be too granular to be able to make fine adjustments to the duty-cycle of the PWM.
* Do not enable the update interrupt or set up the handler: we will not be using interrupts for this part.

1. Use the Capture/Compare Mode Register 1 (CCMR1) register to configure the output channels to PWM mode.
2. The CCMR1 register configures channels 1 & 2, and the CCMR2 register for channels 3 & 4.
3. Examine the bit definitions for the CC1S[1:0] and CC2S[1:0] bit fields; ensure that you set the channels to output.
4. Examine the bit definitions for the OC1M[2:0] bit field; set output channel 1 to PWM Mode 2.
5. Use the OC2M[2:0] bit field to set channel 2 to PWM Mode 1.

* The OC2M bits operate identically to the OC1M, so they have the same documentation.
* You will see the difference between the different PWM modes in a later exercise.

1. Enable the output compare preload for both channels.
2. Set the output enable bits for channels 1 & 2 in the CCER register.
3. Set the capture/compare registers (CCRx) for both channels to 20% of your ARR value.

The previous sections mention that the frequency of a PWM signal isn’t nearly as important as the duty-cycle, provided that the frequency is high enough; this is true for dimming LEDs, although the lower limit isn’t how fast the LED can respond to the electrical signal, but how fast the human eye can distinguish separate blinks.

When both the light and viewer are stationary, the human eye has difficulty seeing the blinking transitions past 70 Hz; many people, however, may see noticeable flicker in moving lights even above 500 Hz. You’ll therefore be using 800 Hz as the base frequency for the PWM.

// configure CC1S and CC2S

TIM3->CCMR1 |= TIM\_CCMR1\_CC1S\_Pos | TIM\_CCMR1\_CC2S\_Pos;

// set OC1M to PWM mode 2 (111)

TIM3->CCMR1 |= TIM\_CCMR1\_OC1M\_Msk;

// set OC2M to PWM mode 1 (110)

TIM3->CCMR1 |= TIM\_CCMR1\_OC2M\_1 | TIM\_CCMR1\_OC2M\_2;

// enable output preload for both channels

TIM3->CCMR1 |= TIM\_CCMR1\_OC1PE | TIM\_CCMR1\_OC2PE;

// enable output in CCER for the channels

TIM3->CCER |= TIM\_CCER\_CC1E | TIM\_CCER\_CC2E;

3.3 — Configuring Pin Alternate Functions.

All four of the LEDs on the Discovery board connect to timer capture/compare channels. This enables us to control their apparent brightness using PWM. In the previous exercise you used two of the LEDs in timer 2’s interrupt. For this portion of the lab, you’ll use the remaining two.

* 1. Look up the alternate functions of the red (PC6) and blue (PC7) LEDs by following examples 3.2 and 3.3.
* Alternate functions that connect to the capture/compare channels of timers have the form: “TIMx\_CHy”.
  1. Configure the LED pins to alternate function mode and select the appropriate function number in alternate function registers.
* Alternate function numbers for each pin are listed in table 15 of the device datasheet.
* The alternate function registers are defined as an array in stm32f0xb.h. You’ll need to check the register map in the peripheral manual to determine what alternate function register to modify for the pins you are using.
  1. Although we configured the matching capture/compare channels first in this lab, typically you choose pins first and then work with the timer channels available.
  2. Compile and load your application onto the Discovery board.

3.4 — Measuring PWM Output.

In exercise 3.2 you configured channel 1 to PWM mode 2 and channel 2 to PWM mode 1. In this exercise you will be exploring the difference between the two modes and the effect of the CCRx register on the output duty cycle.

1. Connect the Saleae logic analyzer to pins PC6 and PC7 and start a capture with the PWM running.
2. Considering that both channels have their CCRx values set to 20% of the ARR, what is the difference between the two PWM modes?
3. Experiment with a variety of CCRx values for both channels.

* What does increasing the CCRx value do for each PWM mode?
* The maximum value that can be used in the CCRx register is the ARR value. What is the relationship between PWM duty cycle and the CCRx, ARR registers?

1. Take a screenshot of the effect of the duty cycle on the waveform using your logic analyzer; include this screenshot with your postlab submission.

Beverly Yee

u0770041

Prelab 03

Postlab Questions

1. Using a timer clock source of 8 MHz, calculate PSC and ARR values to get a 60 Hz interrupt.

* This is tricky because precisely 60 Hz is impossible with our system; instead, think about the process and minimize the error. Many combinations of PSC and ARR values work—not just one!

1. Look through the Table 13 "STM32F072x8/xB pin definitions" in the chip datasheet and list all pins that can have the timer 3 capture/compare channel 1 alternate function.

* If the pin is included on the LQFP64 package that we are using, list the alternate function number that you would use to select it.

1. List your measured value of the timer UEV interrupt period from first experiment. Timers, PWM and GPIO Alternate Functions 17
2. Describe what happened to the measured duty-cycle as the CCRx value increased in PWM mode 1.
3. Describe what happened to the measured duty-cycle as the CCRx value increased in PWM mode 2.
4. Include at least one logic analyzer screenshot of a PWM capture.
5. What PWM mode is shown in figure 3.6 of the lab manual (PWM mode 1 or 2)?