Digital Input-Output and Timers – preparatory lab

Please complete this lab sheet before the assessed lab in week 4

Aim

There are two aims for this first assessed lab.

1. Extend the work in the introductory lab so that you have the understanding and skills to handle multiple digital outputs and inputs
2. Understand how to implement basic accurate timing into embedded systems through the use of the timers and develop skills to implement basic real-time control using timers.

Structure of the Lab

The lab has a non-assessed and assessed part. In this non-assessed part the lab sheet is split into exercises. Some exercises will require you to write complete code, while others will require you to modify/complete some code and then use it.

At the end of each exercise save the complete contents of you "main.c" file into a separate text file (copy and paste) so that you can access it easily at a later stage. *You might need to refer to the "main.c" that you have written for each exercise during the assessed lab session.*

The assessed part will be provided at the start of the assessed lab session and will comprise of two exercises in a similar format to this lab sheet.

Date preparatory lab sheet is available from: Week 1 Monday 0900

Date of assessed lab session: Friday Week4; Group 1 – 1100-1250; Group 2 – 1300-1450; Group 3 – 1500-1650

Learning, Assessment and Feedback process

Please use the time before the assessed lab to prepare and attempt the exercises in this lab sheet. The purpose of the take home kit is to allow you to spend as much time as possible using the kit. It also allows you to learn independently and to explore the capabilities of the kit on your own. *It is unlikely that you will be able to complete the lab in the 1h30m available unless you have prepared accordingly.*

Assessment: Your work will be marked during the lab session timetabled for this lab. *These labs will be conducted under "exam conditions" - i.e. No verbal or other communication (such as phones or email) will be allowed*. Please arrive at the start of the lab session. If you do not arrive at the start then there might be insufficient time to mark all of your work. During the lab session you will be allocated an assessor and this assessor will be responsible for you assessment. After you complete each exercise please raise your hand and the assessor will mark your work and provide. *Please ensure that any original code that you write is thoroughly commented in a similar manner to that provided as a learning resource in this module.*

The assessors primarily role in the lab is to assess your work. They will not provide you with step by step help in the lab. They will provide you with some feedback when they mark your work. Full written solutions will not be made available, but the solutions and common mistakes will be covered in the next lecture.

Unfair Means

The assessed lab component should be completed individually. The lab must be wholly your own work. Any suspicions of the use of unfair means will be investigated and may lead to penalties. See http://www.shef.ac.uk/ssid/exams/plagiarism for more information. You should be prepared to answer questions from the assessors about your solutions. *If you cannot reasonably explain how your solution works then you may lose marks.*

Special or Extenuating Circumstances

If you have medical or personal circumstances which cause you to be unable to complete this lab on time or that may have affected your performance, please complete and submit a special circumstances form along with documentary evidence of the circumstances. See http://www.shef.ac.uk/ssid/forms/special, particularly noting point 6 (Medical Circumstances affecting Examinations/Assessment).

Support

If you have any problems with the equipment please contact Dr S A Pope immediately using the contact details below. *These inquiries should only relate to problems with the equipment, such as suspected faults.* Any inquiry should include a clear description of the problem - "It isn't working" would not be acceptable on its own.

Contact details: Dr S A Pope, Email: s.a.pope@sheffield.ac.uk, Internal telephone ext.: 25186, Room: C07d AJB

Initialisation - Setup a project in Keil µVision

To start each exercise you will first need to create a project in Keil µVision using the same process as you used in the introductory lab. Connect the STM32F4Discovery to the PC and follow steps 1 - 7 in the Introductory lab sheet. You will need the same components of the STM32F4Discovery that you used in the Introductory lab. So you should select the same components in the “Manage Run-Time Environment” window in step 3. In the Introductory lab you selected the timer component “TIM”, but didn’t use it. In this lab you will use the timer functions.

The exercises start on the next page.

Exercise 1 – Timers

Up until this stage you have been using a simple FOR loop to implement a delay. However, as previously mentioned, this is not a very good way to implement a delay in embedded systems when accurate timing and efficient use of resources is usually a primary objective. A much better solution is to use the timer functions that are built into all but the most basic microcontrollers. One of the benefits of a timer is that it can run in the background, unlike a FOR loop which requires the whole program to pause for the duration of the delay. This allows us to do other tasks while we wait for the timer. This concept will become important in later labs where we take a closer look at timing and control issues in embedded systems. In this exercise you will use a timer to blink the green LED on/off at 1 second intervals.

The STM32F407 that is used in the Discovery board has 17 timers (TIM1-17) built into its architecture. Some of these have specific functions, such as TIM1 and 8 which are advanced control timers, while others are general purpose timers such as TIM2-5. We will focus on the general purpose timers here. TIM2-5 are described in section 18 (page 578) of the **STM32F - Processor Reference Manual** - you will need to refer to this document numerous times during this lab. In addition to implementing delays, these timers can be used for a multitude of things, such as measuring waveforms on input signals, e.g. the amount of time that a PWM signal is logic high (see the additional reading document for lecture 3 for information on PWM).

As with all of the components on the microcontroller we need to configure the operation of the timers before we use them. The different functions that can be set are covered in section 18.3 of the processor reference manual. For example, the timer can be selected to either count up or count down and we can select what value it counts up to or down from. We are going to use the timer in its count up mode (described on page 581), in which it counts up from zero to a pre-set value. The registers that need to be set to configure this operation are described in section 18.4 of the processor reference manual. In particular, to implement a basic delay using the upcounter, we need to configure options in CR1 (configuration register 1), PSC (prescaler), ARR (Auto Reload register) and EGR (event generation register). To setup the timer we will need to use the following settings:

CR1: Set timer as upcounter (bit 4 set to 0) and do not use repeat (bit 3 set to 1). Bit 0 is used to enable the counter.

PSC: Sets the counter prescaler to determine the counter clock frequency = prescaler clock frequency/(PSC+1). The prescaler clock frequency is the timer frequency. TIM2 runs using the APB1 clock and the timer frequency defaults to double the APB1 clock frequency. We set the APB1 clock frequency to 42MHz in point 8 of the steps described in Appendix A of the Introductory Lab. So to achieve a counter clock frequency of 10kHz we set the prescaler value PSC to 8399, i.e. 10000=84000000/(8399+1).

ARR: This is the value to set to create a delay of specific length. For example, a delay of 1 second with a counter clock frequency of 10KHz requires an ARR of 10000-1 (i.e. 9999 steps of 0.1ms). The counter flag is set when it overflows past the ARR, which would be the 10000 step, i.e. 10000x0.0001s = 1 second.

EGR: Setting bit 0 to 1 re-initialises the timer and starts its operation in the modes selected.

The code below shows a partial solution that you need to complete to setup the general purpose timer TIM2 and use it as a delay. Your task is to complete the missing part of the initialisation code below marked in blue.

You should create a new project and add this code at the start of an empty main.c file.

```
#include "stm32f4xx.h"
int main(void){
// Initialize GPIO Port for LEDs
RCC->AHB1ENR |= RCC_AHB1ENR_GPIODEN; // Enable GPIOD clock
GPIOD->MODER |= GPIO_MODER_MODER12_0; // GPIOD pin 12 output - green LED
//Initialize Timer 2
RCC->APB1ENR |= RCC_APB1ENR_TIM2EN; // Enable timer 2 clock
TIM2->CR1 &= ~0x00000016; /*Sets the counter as an upcounter*/
TIM2->CR1 Add the correct code /*Turn off repeat in CR1 - i.e. use one
pulse mode*/
TIM2->PSC = 8400-1; /*Prescaler value - the prescaler clock is running at
16MHz (the default value is set to the internal clock)*/
TIM2->ARR = Add the correct numeric value; /*sets the value in the auto-
reload register*/
TIM2->EGR = 1; /*Re-initialises the timer*/
}
```

The final step is to replace the simple FOR loops that we previously used to implement the delay. To do this each of the FOR loops used to implement the delay in exercise 1 needs to be replaced with three lines of code. When used to implement a 1 second green LED blinky these three lines of timer code are used as shown below. You need to complete the missing code marked in blue (note that there are only two different lines since the same timer code is re-used after the LED is turned off).

Add this code after the initialisation code in the main() function and when complete compile and download your program to test/observe its operation.

```
for(;;){
GPIOD->BSRR = 1<<12; // Turn on the green LED
Add the correct code //Enables the counter using the register CR1
while((TIM2->SR&0x0001)!=1){}; //TIM2 to implement a 1 second delay
Add the correct code //Resets to 0 the update interrupt flag in the
register SR - you will need to refer to the processor reference manual to
work out which bit this is in the SR register
GPIOD->BSRR = 1<<(12+16); // Turn off the green LED
Add the correct code //Enables the counter using the register CR1
while((TIM2->SR&0x0001)!=1){}; //TIM2 to implement a 1 second delay
Add the correct code //Resets to 0 the update interrupt flag in the
register SR - you will need to refer to the processor reference manual to
work out which bit this is in the SR register
}
```

The solutions are given at the end of this document

Exercise 2 - Digital inputs (User push-button)

So far you have only dealt with digital outputs. Obviously most embedded systems will have a mixture of inputs and outputs. Basic digital inputs can be treated in a similar manner as we have used for the basic digital output in the form of the LED's. Instead of writing to a pin on a port, for inputs we read from a pin on a port. For this question you will create a new project and write some code to deal with the digital input from one of the most basic input sources - a push-button switch.

The code that you will write will do the following:

The STM32F4Discovery must remain inactive (i.e. none of the four user LED's on) until the blue user push-button switch is pressed. When the push-button is pressed the green LED comes on for the duration of time that the button is held down.

When complete compile and download your program to test/observe its operation.

<u>IMPORTANT:</u> Please attempt to write your own comments for the lines of code that you write or use from the provided material. In assessments you risk losing marks if you have not sufficiently commented your code with original comments and/or the marker cannot reasonably understand its operation.

Hints:

1. You will need to find out which port and pin the push-button switch is connected to by referring to the STM32F407Discovery - User Manual.
2. You will need to work out the appropriate mode register (MODER) setting for the required GPIO port and pin combination to set it into input only mode by using the information in the Processor Reference Manual. You will already have used similar code to configure some of the GPIO pins for the LED's.
3. To read the state of the switch you just need to extract the correct bit in the relevant GPIO input data register.
4. To implement the required functionality in the main() function to check the state of the switch you will need to use an IF/ELSE statement (see the C programming guide document for information on the IF statement).

The solutions are given at the end of this document

Switch de-bounce

In exercise 2 you probably just read the value of the input data register and then made a decision on the state of the push-button switch immediately. In reality most switches suffer from "switch bounce". This means that they do not usually make a clean contact when they are pressed/released. This occurs due to a number of factors mainly related to the mechanical operation of the switch, such as the elasticity of the metal and irregular pressure on the switch. It is therefore common good practice to implement some code to "de-bounce" the state of the switch before making a definitive decision on its state. This leads to more robust code that is less prone to operational errors. The easiest way to do this is as follows:

> When the change in state of the switch is first detected, wait a short period of time using some form of delay (around 1 microsecond is sufficient for most switches). After the delay re-check the state of the switch. If the change of state is confirmed then proceed as planned (for example in exercise 4 this would be to turn on the green LED when the button is pressed), otherwise proceed as if the state of the switch hasn't changed (in exercise 4 this would be to keep the green LED turned off if the switch isn't pressed).

Assessed Lab Exercises

Group 1 should arrive at 1100, Group 2 at 1300 and Group 3 at 1500. You should spend the first 10 minutes preparing (opening Keil and starting a new project). Please make sure that you attend the correct room (check your iSheffield calender and MOLE). The lab will start at 1110 for Group 1, 1310 for Group 2 and 1510 for Group 3. You will have 1h30m to complete the exercises. The final 10 minutes will be used to mark any remaining work.

The lab will be marked out of a total of 10 and will have the following structure:

Exercise 1 – Will require you to show that you understand and can use the knowledge that we have covered so far. **[5 marks]**

Exercise 2 – Will require you to show that you can find new information and modify the code that we have written so far so that it functions differently. **[5 marks]**

In preparation you should create a new project with an empty main() function for each of the exercises (following the guidelines given in the initialisation section on page 2 of this lab). This will allow you to make maximum use of the 1h30m assessment period.

You will be allowed to view the documentation, previous labs and your previous work when conducting this lab. However, you will not be allowed to use a web-browser. Please make sure that you have downloaded all relevant information that you will need before the lab. **Only the provided documents written in English will be allowed.**

You can either bring your own laptop, or use the PC's in the labs.

Please remember to bring your STM32F4Discovery kit to the lab session.

Please refer to the lab cover sheet (first page of this document) for more details on the assessment.

IN THE NEXT NON-ASSESSED LAB: You will investigate how to communicate with an external device through serial ports and methods to control when this communication occurs.

Solutions

Exercise 1

```
#include "stm32f4xx.h"

int main(void){

	// Initialize GPIO Port for LEDs
	RCC->AHB1ENR |= RCC_AHB1ENR_GPIODEN; // Enable GPIOD clock
	GPIOD->MODER |= GPIO_MODER_MODER12_0; // GPIOD pin 12 output - green LED

	//Initialize Timer 2
	RCC->APB1ENR |= RCC_APB1ENR_TIM2EN; // Enable timer 2 clock

	TIM2->CR1 &= ~0x00000016; /*Sets the counter as an upcounter*/
	TIM2->CR1 |= 0x00000008; /*Turn off repeat - i.e. use one pulse mode*/

	TIM2->PSC = 8400-1; /*Prescaler value - the prescaler clock defaults to
	twice the APB1 which is running at 42MHz - so the timer clock is 84MHz*/

	TIM2->ARR = 10000-1; /*sets the value in the auto-reload register*/
	TIM2->EGR = 1; /*Re-initialises the timer*/


	for(;;){

		GPIOD->BSRR = 1<<12; // Turn on the green LED

		TIM2->CR1 |= 1; //Enables the counter
		while((TIM2->SR&0x0001)!=1){}; //TIM2 to implement a 1 second delay
		TIM2->SR &= ~1; //Resets the flag

		GPIOD->BSRR = 1<<(12+16); // Turn off the green LED

		TIM2->CR1 |= 1; //Enables the counter
		while((TIM2->SR&0x0001)!=1){}; //TIM2 to implement a 1 second delay
		TIM2->SR &= ~1; //Resets the flag

	}

}
```

Exercise 2

```
#include "stm32f4xx.h"

int main(void){

	// Initialize GPIO Port for LEDs
	RCC->AHB1ENR |= RCC_AHB1ENR_GPIODEN; // Enable GPIOD clock
	GPIOD->MODER |= GPIO_MODER_MODER12_0; // GPIOD pin 12 output - green LED

	// Initialize GPIO Port for push-button
	RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN; // Enable GPIOD clock
	GPIOA->MODER &= ~GPIO_MODER_MODER0; // GPIOA pin 0 input - blue user push-
	button - resets to 00 so this line isn’t strictly needed, but is robust
	coding in case of problems/faults with the processors which means that it
	doesn’t default to reset. As a general rule, when writing code for embedded
	systems you should try and make it as robust as possible so as to minimise
	the chance of faults when the embedded system is in service.

	for(;;){

		if(((GPIOA->IDR)&0x0001) == 0x0001){

			GPIOD->BSRR = 1<<12; // Turn on the green LED
		}
		else{

			GPIOD->BSRR = 1<<(12+16); // Turn off the green LED

		}

	}

}
```