A/P Lawrence Ong (lawrence.ong@newcastle.edu.au)
School of Engineering
College of Engineering, Science and Environment
The University of Newcastle

Assignment 5

Instructions

- Complete all sections.
- Work individually.
- Section 1–Section 2 contain 7 multiple-choice questions (MCQ).
  - Enter your answers to these questions on Canvas → Assignment 5 MCQ.
  - Multiple-choice questions are due 11:55pm on Sunday 29 May 2022 (end of week 12).
  - Late submissions will have 10% deducted per day.
- Section 3 requires you to write code for an embedded system.
  - Submit project ZIP file to Canvas → Assignment 5 Demo.
  - Project ZIP file is due 11:55pm on Sunday 29 May 2022 (end of week 12).
  - Late submissions will have 10% deducted per day.
  - Demonstrate your code for during your lab session in week 13. You must use the same code that you submit to Canvas.
- Total of 25 marks possible.

Assessment

This assignment will be assessed using the following criteria:

Component	Unsatisfactory	Satisfactory
Multiple Choice Questions (7 Marks)	Incorrect answer (0 Mark each)	Correct answer (1 Mark each)
Demonstration (18 Marks)	Requirement not implemented (0 Mark each)	Requirement implemented (2 Marks each)

Objectives

The main objective of this assignment is to integrate the following functional units of a microcontroller in one application:

- GPIO
- Interrupts
- DMA
- ADC
- Timers
- Serial communications

There are three parts to this assignment, with the weighting and marking criteria given for each of the three parts. Each part of the assignment need to be enabled in the source code in turn to run the code for each part. This is similar to previous assignments by modifying defines in the header file.

The first two sections will explore functionalities of timers and SPI respectively. The third section will integrate the above-mentioned functional units in an application. The *Hardware Abstraction Layer* (HAL) will be used to access the functionality of these units. The STM32CubeIDE's device configuration tool will also be used to setup and initialise the functions.

Project Zip File Submission

1. Clean up your STM32CubeIDE project before submission. To do this, right click the project name **`Ass-5`** in the left window named Project Explorer and choose **`Clean Project`**.
2. Generate a ZIP file of your project following these instructions:
   - Right click on the project and select **`Export...`**
   - Select under **`General`** the option **`Archive File`** and click **`Next`**.
   - The whole project **`Ass-5`** should have been selected by default. Click **`Browse`** to select where to save the ZIP file; this should be outside the STM32CubeIDE workspace.
   - The default options should be to **`Save in zip format`**, **`Create the directory structure for files`**, and **`Compress the contents of the file`**.
   - Click **`Finish`** to generate the ZIP file.
3. Submit the ZIP file in the drop box provided in the Assignment 5 Demo folder on Canvas.

Note: Besides the demonstration during the lab session, the project will be compiled as part of the marking process. If it is not possible to compile the project, you will be contacted.

1 Timers (3 Marks)

This exercise uses the STM32CubeIDE configuration tool to configure a timer. The HAL library will also be used to initiate the timer in an interrupt mode. Upon the completion of each period of the timer, an interrupt is fired, and an interrupt service routine is written to toggle an LED.

1.1 Project Creation

Duplicate the project from Assignment 4. Follow the instructions in the file **`Duplicate-projects-with-semihosting.pdf`** for duplicating a project. You will need to change the name of some files and functions.

1.2 STM32CubeIDE Device Configuration Tool

Configure Timer 3 using the Device Configuration Tool as follows:

- Clock Source: Internal Clock
- Prescaler (PSC): See question below
- Counter Period (AutoReload Register): See question below
- In the NVIC setting tab, enable TIM3 global interrupt

Multiple Choice Question *(1 Mark)*

What values can be set for the Prescaler (PSC) and AutoReload Register (ARR) of Timer 3 so that the timer's counter overflows every 0.5000s? Assume that the timer's clock frequency is 84MHz. (Select ALL correct answers)

☐ a) PSC = 41,999 and ARR = 999

☐ b) PSC = 42,000 and ARR = 1000

☐ c) PSC = 83,000 and ARR = 499

☐ d) PSC = 84,000 and ARR = 500

1.3 Interrupt Mode

Structure your code similar to Assignments 3 and 4:

- Write your code for each section in this assignment in a separate sources file: **`Ass-05-Q1.c`**, **`Ass-05-Q2.c`**, and **`Ass-05-Q3.c`**.
- Keep a common header file **`Ass-05.h`** for the declaration of common variables.
- The **`main(void)`** function in **`main.c`** should call **`Ass_5_main(void)`** before the **`while(1)`** loop.
- Each of the source file **`Ass-05-Q1.c`**, **`Ass-05-Q2.c`**, and **`Ass-05-Q3.c`** implements a version of **`Ass_5_main(void)`**.

The routine simply starts the timer in the interrupt mode.

Use this code for **`Ass_5_main(void)`** in **`Ass-05-Q1.c`**:

```
void Ass_5_main (void)
{
  HAL_TIM_Base_Start_IT(&htim3);

  while (1) {
  }

}
```

Each time when the timer's counter overflows, an interrupt is generated, and HAL's TIM Period Elapsed Callback function is executed. Locate this function in
`Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_tim.c`.
Write this function in **`Ass-05-Q1.c`** to toggle the blue LED.

Multiple Choice Question *(1 Mark)*

What is the name of HAL's default callback function that is executed when the timer's period elapses in the interrupt mode?

☐ a) **`TIM_DMAPeriodElapsedCplt()`**

☐ b) **`HAL_TIM_TriggerCallback()`**

☐ c) **`HAL_TIM_PeriodElapsedCallback()`**

☐ d) **`HAL_TIM_PeriodElapsedCpltCallback_IT()`**

Compile and run the program. Check that the blue LED toggles every half a second.

1.4 Changing the Timer's Period during Runtime

The timer's period can be changed during runtime by changing the the appropriate registers. These values are updated upon each update event.

Add the following code after starting the timer in **`Ass-05-Q1.c`**:

- Delay for 5s.
- Change a register for Timer 3 so that the blue LED toggles every 0.25s.

Multiple Choice Question *(1 Mark)*

Which registers can be used to change the period of Timer 3? (Select ALL correct answers)

☐ a) **`TIM3->DCR`**

☐ b) **`TIM3->ARR`**

☐ c) **`TIM3->PSC`**

☐ d) **`TIM3->DMAR`**

Compile and run the program. Check that the blue LED toggles every 0.5s for a period of 5s. Then, it toggles every 0.25s.

2 SPI (4 Marks)

This exercise uses the STM32CubeIDE configuration tool to configure an SPI unit. The SPI interface will be used to configure and read the three-axis accelerometer on the discovery board.

2.1 Configuring the GPIO Pins

The accelerometer chip that is available on the STM32F407G-DISC1 discovery board is LIS3DSH (a MEMS chip). Refer to the pin connections for this chip (look for MEMS) on the STM32F407G-DISC1 schematic.

Multiple Choice Question *(1 Mark)*

Which GPIO pin of STM32F407 is connected to LIS3DSH's *chip select* line?

☐ a) PA5

☐ b) PA6

☐ c) PE1

☐ d) PE3

Locate the GPIO pins that connect to the clock, the data input, and the data output lines of the accelerometer chip. In the Pinout view of the Device Configuration Tool, select the **`SPI_XXX`** options for these three pins.

For the fourth pin, that is, the GPIO pin connect to chip select identified above, configure it as follows:

- GPIO_Output
- GPIO output level: High
- GPIO mode: Output Push Pull
- GPIO Pull-up/Pull-down: No pull-up and no pull-down
- User label: SPI1_CS

2.2 SPI Configurations

Under Connectivity $\rightarrow$ SPI1, configure the following, leaving the rest as their default values:

- Mode: Full-Duplex Master
- Hardware NSS signal: Disable
- Prescaler (for Baud Rate): 16

Multiple Choice Question *(1 Mark)*

Why is the prescaler set to 16?

- ☐ a) Because the LIS3DSH chip operates at a maximum SPI clock frequency of 10MHz.
- ☐ b) Because we need to match the APB2 clock frequency.
- ☐ c) Because the SPI unit is not able to transmit data fast enough.
- ☐ d) To lower power consumption.

2.3 Code: Initialisation

Configure the LIS3DSH chip to sample the accelerometer on the Y-axis at a rate of 3.125Hz by writing to its register at the address 0x20. Note that this is the internal address of the LIS3DSH chip, not the address of the STM32 microcontroller.

Create a source file **`Ass-05-Q2.c`** with the following **`Ass_5_main()`** routine:

```
#define SPI_RD 0x80          // The bit to set SPI to read.
uint8_t auchSpiTxBuf[2], auchSpiRxBuf[2];

void Ass_5_main (void)
{

  // Now set up the MEMS chip for 3.125 Hz sampling rate.
  // Initial setup of the MEMS sample rate
  auchSpiTxBuf[0] = 0x20;    // Control Reg 0x20
  auchSpiTxBuf[1] = 0x12;    // Sampling at 3.125 Hz, Y accelerometer

  // Now set the chip select low
  HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_RESET);
  // Now write the set up data.
  HAL_SPI_Transmit(&hspi1, auchSpiTxBuf, 2, 50);  // Send the set up
 command.
  //Now set the CS pin high again
  HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_SET);

  // We can now read the date back to check that the write was successful
  // Set up the auchSpiTxBuf for a read.
  auchSpiTxBuf[0] = 0x20 | SPI_RD;    // Control Reg 0x20 to be read

  // Now set the chip select low
  HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_RESET);
  // Now write the address.
  HAL_SPI_Transmit(&hspi1, auchSpiTxBuf, 1, 50);
  // Now Read the data
  HAL_SPI_Receive(&hspi1, auchSpiRxBuf, 1, 50);
  // Now set the CS pin high again
  HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_SET);

  // Now print out the value to see if it is correct
```

```
    printf("Control␣value␣read␣=␣0x%x\n", auchSpiRxBuf[0]);

    while(1) {
    }

  }
```

Compile and run the program. Check that the correct value `0x12` is printed on screen.

Multiple Choice Question *(1 Mark)*

What is the name of the register at 0x20 in LIS3DSH?

- ☐ a) INFO1
- ☐ b) CTRL_REG4
- ☐ c) STATUS
- ☐ d) SETT2

2.4 Code: Accelerometer Reading

We now read the accelerometer's Y-axis data from the register at 0x2b (OUT_Y_H register). Use the following code within the `while(1)` loop in `Ass_5_main()`:

```
HAL_GPIO_TogglePin(BLUE_GPIO_Port, BLUE_Pin); //LD6 has been labelled BLUE
// Set up the auchSpiTxBuf for a read.
auchSpiTxBuf[0] = 0x2b | SPI_RD;    // Control Reg 0x2B Y value to be read

// Now set the chip select low
HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_RESET);
// Now write the address.
HAL_SPI_Transmit(&hspi1, auchSpiTxBuf, 1, 50);
// Now Read the data
HAL_SPI_Receive(&hspi1, auchSpiRxBuf, 1, 50);
//Now set the CS pin high again
HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_SET);

HAL_Delay(200);
// Now print out the value to see if it is correct
printf("Value␣read␣=␣%d\n", (int8_t)auchSpiRxBuf[0]);
```

Compile and run the program. Tilt the discovery board forward and backward and observe the values printed in the console.

Multiple Choice Question *(1 Mark)*

Why do we read only from the OUT_Y_H register?

☐ a) The OUT_Y_L is not used.

☐ b) The OUT_Y_L is not accurate.

☐ c) OUT_Y_H is sufficient for a coarse reading.

☐ d) None of the above.

3 Christmas Lights (18 Marks)

For this section, you are required to write code for a Christmas-light application. The functionalities required are as follows:

- There are two modes of operation:
  - Mode 1: LED rotates in the clockwise or anti-clockwise direction. Only one LED is lit at any time.
  - Mode 2: Design your own lighting sequence, but you need to distinguish the clockwise and the anticlockwise directions.
- Pressing the blue button on the discovery board toggles between the two LED modes.
- Tilting the discovery board toward the right results in the LEDs rotating in the clockwise direction. Tilting the discovery board toward the left results in the LEDs rotating in the anti-clockwise direction.
- The speed of rotation is controlled by the ADC. Only one potentiometer is required. Turning the potentiometer knob to the extreme anti-clockwise position gives the lowest speed. The speed increases as the knob is turned clockwise.
- At least four distinguishable speeds must be observed.

In addition to the above functionality requirements, your code needs to implement the following:

- Adjust the timer's period based on the ADC value.
- Use the timer's period to control the LED rotation speed.
- Read the ADC values using the DMA mode and triggered by another timer.
- Debounce the blue button.

Hints:

- For the ADC, as PA5–PA7 are used for the SPI connections, use a jumper to connect the potentiometer to PC2 instead.
- Refer to slide 25 of Topic 8 to configure the ADC in the DMA mode triggered by a timer. Adjust the prescaler and counter period so that the ADC samples more frequently, around 0.25s.
- To configure the blue button, refer to the STM32F407G-DISC1 schematic. Look for the USER & WAKE-UP Button. Note that the blue button operates differently compared to the joystick. The blue button is grounded when not pressed, and pulled to VDD when pressed. You need to configure the pull-up/pull-down setting correctly.
- Configure the MEMS chip to read the X-axis so as to detect the left/right tilt.
- Use the timer's callback function to change the LEDs.

Demonstration *(18 Marks)*

The above functionalities and requirements will be marked by the lab demonstrator during the lab session.