ECE 1770

ELECTRONIC MICROPROCESSOR SYSTEMS

Lab 7: Systick and RTC in Assembly

TASK#	Grading criteria	Instructor Initial
1	Your program runs successfully without any compilation error when you demo it. (20)_____ The behavior of the LCD follows the requirement of this task. (40)_____ Submit your complete and correct assembly codes on Courseweb. (40)_____	
Bonus 1	The behavior of the stepper motor follows the requirement of this task. (4)_____ Submit your complete and correct assembly codes on Courseweb. (4)_____	
Bonus 2	The behavior of the clock follows the requirement of this task when the center button is pressed. (1)_____ The behavior of the clock follows the requirement of this task when the left/right button is pressed. (3)_____ The behavior of the clock follows the requirement of this task when the up/down button is pressed. (3)_____ The time displayed on LCD is correct after resetting. (5)_____ Submit your complete and correct assembly codes on Courseweb. (1)_____	

In order to receive credit for this laboratory exercise, please submit this handout to your lab instructor when you are finished.

Team Members: Group Number:	TA:

ECE1770

Lab 7: Systick and RTC in Assembly

Jingtong Hu
April 2, 2019

1. Objective

1) Understand the basic concept of system timer (SysTick)
2) Understand the basic procedure of interrupt handling
3) Program real-time clock (RTC)
4) Gain knowledge of using SysTick to generate periodic interrupts, e.g. read RTC periodically

2. Lab Assignments

1) Use SysTick to generate an interrupt every 1*ms*. In *SysTick_Handler()*, read RTC clock and display time (hours, minutes and seconds) on LCD every 1s.
2) **Bonus_1**: Set an alarm and turn the stepper motor when the alarm is triggered.
3) **Bonus_2**: Implement a clock with setting function

Note: In the project template on Courseweb, the regular task is implemented in *main.s*. Write your codes in this file and after you finish the assignment, rename this file *lab7.s*. Submit the file to the entry *Lab 7 Group Submission*. For the bonus part, please upload two separate assembly files called *lab7_bonus1.s*, *lab7_bonus2.s*, which contain the complete codes of task bonus_1 and task bonus_2, respectively. And then submit them to another entry *Lab 7 Bonus Group Submission*.

3. Clock Configuration

In this lab, in order to generate a Systick interrupt with a period of 1ms, we need to configure the clock for system timer (SysTick) first. In the *main.s* of project template, by calling the function *System_Clock_Init*, the clock for SysTick is configured to be 1M Hz. If you are interested in this function, you can refer to the file *port.c* of project template. Note the codes are written in C.

4. Systick Interrupt Configuration

After configuring the clock for systick, you need to generate systick interrupt with a period of 1ms. In order to do that, you need to configure the registers related to systick interrupt. The steps are as follows:

1. Disable Systick IRQ and Systick Counter first before you configure the related registers.
2. Set the Systick reload value register (SysTick_LOAD). Since the frequency of the clock is 1MHZ and the period of systick is 1ms, the Interrupt period should be 1000 clock cycles. So the value in SysTick_LOAD should be (1000 – 1) = 999.
3. Reset the Systick counter value (SysTick_VAL) by writing 0 to it.
4. Set the interrupt priority of Systick. In this lab, use the codes as follows:

```
PUSH {LR, R0}
BL config_NVIC_in_C
POP {LR, R0}
```

5. Enable the Systick interrupt by setting the Tickint bit in SysTick_CTRL to 1.
6. Select the processor clock by configuring the Clock Source bit in SysTick control and status register (SysTick_CTRL). Here we choose the external 1MHZ clock, which should already be configured according to the section 3 before. The Clock Source bit should be set to 1 here.
7. Enable the Systick IRQ and the Systick Timer by configuring the ENABLE bit in SysTick control and status register (SysTick_CTRL). Here we set the ENABLE bit to 1.

5. RTC Configuration

Before you use the RTC to provide the calendar time and date, you need to configure RTC first. The steps to initialize RTC are as follows:

1. Initialize the clock for RTC. You need to select **LSE** as clock source. In this lab, you just need to call the function *RTL_Clock_Init* which has been implemented in file RTC.c.
2. Disable the RTC register write protection.
3. Enter initialization mode and wait for the confirmation
4. Set the RTC to an initial value. In this lab, we do not care about the date. So you only need to set the value in Time Register (RTC_TR). In fact, you can set any time you like. But for convenience, I suggest you setting it to **14:34:59**.
5. Exit Initialization mode and wait for the confirmation.
6. Enable the RTC Registers Write Protection (RTC_WPR).

6. Writing Systick Interrupt Handler

After you configure and enable the Systick interrupt, every time a Systick interrupt signal is detected, the processor will be forced to respond to this interrupt. It will auto-stack the related registers, executing the interrupt handler, auto-unstack the values stored and return to execute the original instructions. As a programmer, you only need to be in charge of writing the Systick Interrupt Handler, which means you need to tell the processor what to do after a Systick interrupt is generated. In this lab, you need to finish the function SysTick_Handler on your own. The detailed requirement about this function will be elucidated in the Task part.

7. Pre-Lab

To finish initializing the RTC, you need to set the correct value for the related registers. The following figure shows all the registers related to RTC. The registers we care about in this lab are *RTC_TR*, *RTC_CR*, *RTC_ISR* and *RTC_WPR*. And the other registers might be used for the bonus tasks. Fill in the value in the registers necessary for the lab tasks before you write codes. It will help you to understand the meaning of each step and write codes more efficiently.

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	RTC_TR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PM	HT [1:0]		HU[3:0]				Res.	MNT[2:0]			MNU[3:0]				Res.	ST[2:0]			SU[3:0]			
	Value																																
0x04	RTC_DR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	YT[3:0]				YU[3:0]				WDU[2:0]			MT	MU[3:0]				Res.	Res.	DT [1:0]		DU[3:0]			
	Value																																
0x08	RTC_CR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ITSE	COE	OSEL [1:0]		POL	COSEL	BKP	SUB1H	ADD1H	TSIE	WUTIE	ALRBIE	ALRAIE	TSE	WUTE	ALRBE	ALRAE	Res.	FMT	BYPSHAD	REFCKON	TSEDGE	WUCKSEL[2:0]		
	Value																																
0x0C	RTC_ISR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ITSF	RECALPF	TAMP3F	TAMP2F	TAMP1F	TSOVF	TSF	WUTF	ALRBF	ALRAF	INIT	INITF	RSF	INITS	SHPF	WUT WF	ALRBWF	ALRAWF
	Value																																
0x10	RTC_PRER	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PREDIV_A[6:0]							PREDIV_S[14:0]															
	Value																																
0x14	RTC_WUTR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	WUT[15:0]															
	Value																																
0x1C	RTC_ALRMAR	MSK4	WDSEL	DT [1:0]		DU[3:0]				MSK3	PM	HT [1:0]		HU[3:0]				MSK2	MNT[2:0]			MNU[3:0]				MSK1	ST[2:0]			SU[3:0]			
	Value																																
0x20	RTC_ALRMBR	MSK4	WDSEL	DT [1:0]		DU[3:0]				MSK3	PM	HT [1:0]		HU[3:0]				MSK2	MNT[2:0]			MNU[3:0]				MSK2	ST[2:0]			SU[3:0]			
	Value																																

0x24	RTC_WPR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	KEY							
	Value																																
0x28	RTC_SSR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SS[15:0]															
	Value																																
0x2C	RTC_SHIFTR	ADD1S	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SUBFS[14:0]														
	Value																																
0x30	RTC_TSTR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PM	HT[1:0]		HU[3:0]				Res.	MNT[2:0]			MNU[3:0]				Res.	ST[2:0]			SU[3:0]			
	Value																																
0x34	RTC_TSDR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	WDU[1:0]			MT	MU[3:0]				Res.	Res.	DT [1:0]		DU[3:0]			
	Value																																
0x38	RTC_TSSSR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SS[15:0]															
	Value																																

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x3C	RTC_ CALR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CALP	CALW8	CALW16	Res.	Res.	Res.	Res.	CALM[8:0]								
	Value																																
0x40	RTC_TAMPCR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TAMP3MF	TAMP3NOERASE	TAMP3IE	TAMP2MF	TAMP2NOERASE	TAMP2IE	TAMP1MF	TAMP1NOERASE	TAMP1IE	TAMPPUDIS	TAMPPRCH[1:0]		TAMPFLT[1:0]		TAMPFREQ[2:0]			TAMPTS	TAMP3-TRG	TAMP3E	TAMP2-TRG	TAMP2E	TAMPIE	TAMP1TRG	TAMP1E
	Value																																
0x44	RTC_ ALRMASSR	Res.	Res.	Res.	Res.	MASKSS [3:0]				Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SS[14:0]														
	Value																																
0x48	RTC_ ALRMBSSR	Res.	Res.	Res.	Res.	MASKSS [3:0]				Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SS[14:0]														
	Value																																
0x4C	RTC_ OR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RTC_OUT_RMP	RTC_ALARM_TYPE
	Value																															0	0
0x50 to 0xCC	RTC_BKP0R	BKP[31:0]																															
	Value																																
	to RTC_BKP31R	BKP[31:0]																															
	Value																																

Tasks

Note: For the basic task part, I have already provided part of the codes and necessary files in the project template for lab 7 on Courseweb. Please download it to your computer. The comments are crucial hints for coding. Make full use of them. In some lines, you may need some value as mask for specific registers, which can be found in the **Pre-Lab** section if you have finished them. You may also need some other constants predefined in *stm32l476xx_constants.s*. Refer to the slides of the related classes may help you to do this. Besides, there are also many codes you can refer to in those slides. Understand those codes will be very helpful for finishing the following tasks.

Task: Display the Time in RTC Periodically

Use SysTick to generate periodic interrupts to read RTC. The systick interrupt should be generated every 1ms. After each second, which means every time the total amount of systick interrupt detected has achieved to 1000, the LCD should display the time (hours, minutes and seconds, using 24h format) in RTC. For instance, if the time in RTC is 13:12:56, display 131256 on LCD. To finish this task, in the function *SysTick_Handler*, you need to record the total amount of the occurrence of systick interrupt. When the value achieves 1000, you need to read the time in RTC, and display it on LCD. For convenience, I suggest you setting the initial time of RTC to **14:34:59** when you initialize the RTC, which has already been mentioned in section 5.

Hint: You can refer to the codes in the related slides to know about how to read time from RTC. According to those codes, you will get the seconds, minutes and hours in R2, R1, R0, respectively. All of these values are in binary form. In you want to display the seconds on LCD, you can call the function ***LCD_Display_Seconds****. Before you call this function, you should pass the value of seconds (saved in R2) to R0. Similarly, to display the minutes and hours on LDC, you need to call* ***LCD_Display_Minutes****,* ***LCD_Display_Hours****, respectively and you need to pass the value in binary form to R0 before you call these two functions.* ***Note*** *we recommend you to display hours first because the value of hours is saved in R0 after you read RTC. If you call the other two function first, the value in R0 will be overwritten since you move the other value to it before you call the function.*

Bonus 1: Spin Stepper Motor When an Alarm is Triggered

This task will add **3 points** to your final grades of this course.

In this task, you need to set an alarm every time the **center button** of joystick is pressed. The alarm should be triggered 10 seconds after the moment when the center button is pressed. When the alarm is triggered, you need to spin the stepper motor 180 degree anticlockwise with full stepping.

*Hint: Since RTC Interrupts are connected to EXTI18 Internally, you need to enable and configure the EXTI line 18 first. By doing this, when an alarm set before is triggered, the processor will respond to this interrupt and execute the RTC_Alarm_IRQHandler. So you are responsible for writing the handler to tell the processor what to do when an alarm is triggered. And do not forget to export RTC_Alarm_IRQHandler. Figure 1 shows how to export a handler for Systick. You can imitate it to export the handler for RTC alarm. Besides, when you set time for RTC or RTC alarm, the value you pass to the function is in BCD code (refer to the related slides to learn how to set time for RTC). However, the value you read from RTC is in binary from. So you need to convert the value you read from binary form to BCD code. I have provided the function to help you finish this conversion. Call function **BIN2BCD** and pass the value to R0 before you call it. The value in BCD code will be returned and saved in R0.*

```
INCLUDE core_cm4_constants.s        ; Load Cortex-M4 Definitions
INCLUDE stm32l476xx_constants.s     ; Load STM32L4 Definitions
EXPORT SysTick_Handler
```

Figure 1. Exporting handler for systick

Bonus 2: Implement a Clock with Setting Function

This task will add **5 points** to your final grades of this course.
In the regular task, we can display the time on LCD after every second. For a clock, we want to set the time as we like in addition to displaying the time. In this task, we try to implement the setting function by using the buttons of the joystick. The requirement is as follows:

1. The **center button** is used for starting and ending a time setting. When the center button is pressed at the first time, the clock will enter setting mode. And after the time is reset and the center button is pressed for the second time, the clock will exit setting mode. And the clock will keep displaying the time as usual based on the latest time just set.

2. When the clock enters setting mode, the **left/right button** can be used to move the “cursor”, which is used for choosing among hours, minutes, and seconds. The cursor is virtual, you don’t need to display the cursor. The position of the cursor will

be at the seconds by default when the clock enters setting mode. The cursor will be moved among hours, minutes, and seconds each time you press the left or right button. For example, if the cursor is at the seconds and the left button is pressed, the cursor will move to the minutes. And if the cursor is at the hours and the right button is pressed, the cursor will move to the minutes. However, if the cursor is at the seconds and the right button is pressed, the cursor will not move. If the cursor is at the hours and the left button is pressed, the cursor will not move.

3. When the clock enters setting mode, the **up/down button** can be used to change the value. Every time you press the up button, the value of the chosen part will be increased by one. And the value will be decreased by one each time the down button is pressed. However, if the value you want to change has achieved its upper bound, it should not increase even if you press the up button. For instance, assume you are changing the value of minutes. when the value of minutes has become 59 and you press the up button again, the value should not change. Similarly, when the value achieves its lower bound, the value will not change when you press the down button. **Note** that you should display the latest time on LCD every time you push the up/down button.

I will give you an example to help you learn how the time setting works. Assume the current time is **123456 (12:34:56)** and I want to set the time to **143456(14:34:56)**. First, I need to press the center button to enter the setting mode. Then I press the left button 2 times. By doing this, I move the cursor to hours. And then I press the up button, so the LCD will display **133456 (13:34:56)**. Then I press the up button for the second time. The LCD will display **143456 (14:34:56).** Since this is the new time I want to set, I press the center button to exit setting mode. So now the LCD should display **143456(14:34:56)** and it keeps displaying the latest time.

Hint: Like Bonus 1, you also need to do the conversion from binary form to BCD code before you set RTC to the new time.