Exercise Manual
For

CE3103
Embedded Programming

Practical Exercise #5
RTOS Based Programming:
Task with Software Timer
Preemptive and Non-Preemptive Multitasking

Venue: Hardware Laboratory 2
(Location: N4-01b-05)

COMPUTER ENGINEERING COURSE

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

CE3103
Embedded Programming
Exercise #5

Learning Objectives

In this lab, you will develop RTOS based application programs to compare the difference between preempt and non-preempt multi-tasking, and learn the important of priority in multitasking.

In these lab exercises, you will use RTOS to develop tasks to control the following device on the embedded board: a UART interface to send message to PC and receive messages from PC, and a button interface to play a song from a buzzer.

Equipment and accessories required

i) An embedded development board with a STM32F103VET6 microcontroller
ii) FlyMcu - An In-System Programming (ISP) software to upload machine codes to flash memory)
iii) Tera Term
iv) Keil's µVision IDE
References: Keil Microcontroller Development Kit (MDK-Lite)
http://www.keil.com/arm/mdk.asp
http://www.keil.com/uvision/

---

1. Introduction

1.1 A Musical Button

Since you have learned to turn on the buzzer in Practical Exercise #4, you will implement a button (the USER button next to RESET button). Some sample codes on playing different frequency tones are given. You are required to play the given song for a few seconds. To achieve delay in microseconds, SysTick interrupt is used.

1.2 UART Communication

In the embedded development board, we have two USB communication ports. The first one is used for program downloading to the Flash. The second one is used for serial communication. See figure below:

From the schematic diagram, a USB Serial Communication is provided.

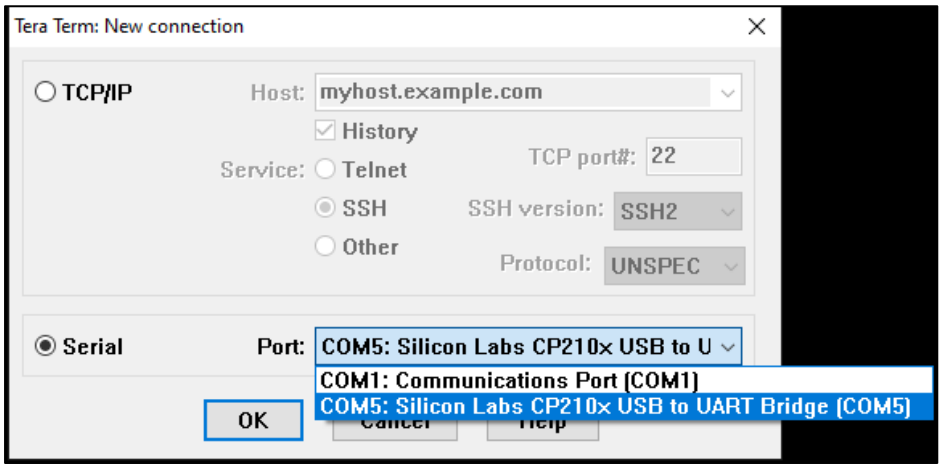
The ports and pins connected to the microcontroller are _____ and _____.

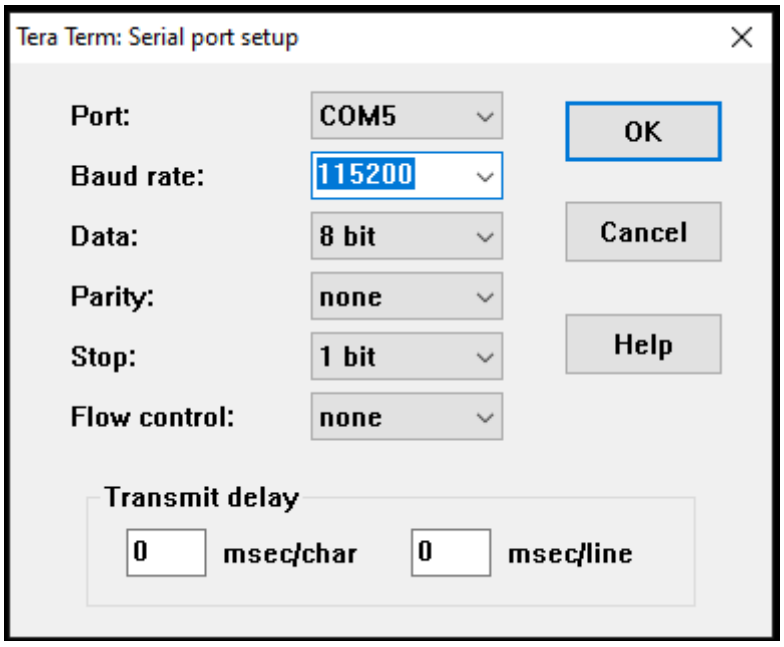
The UART initial configuration is given in Apendix A. You may select appropriate baud rate in your implementation.

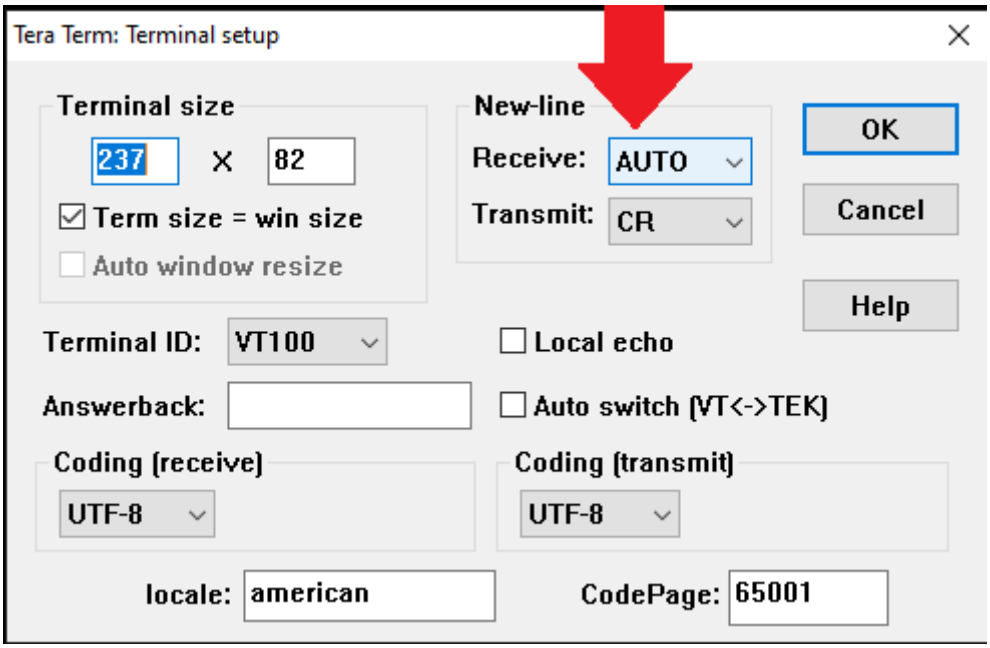
To send a 8-bit character to PC, you just need to put it at data register (DR). When the transmit enable bit (TE) is set, the data in the transmit shift register is output on the TX pin and the corresponding clock pulses are output on the CK pin.

```
void usart3_send(u8 data)

{

        USART3->DR = data;

        while((USART3->SR&0x40)==0);

}
```

Once the data can be transmitted via USB. You can use Tera Term to receive the data at PC.







To send an character back to the device, you need to use USART3_IRQHandler()

```
int USART3_IRQHandler(void)
{
	static u8 Count=0,i;
	u8 Usart_Receive;

	static u8 rxbuf[10];

	if(USART_GetITStatus(USART3, USART_IT_RXNE) != RESET)
	{
		Usart_Receive = USART_ReceiveData(USART3);


	  // USART_Receive is the received data

	}
return 0;
}
```

2. Practice Exercises

- In these exercises, you will learn how to create a button and play the given song via buzzer. You also will change the LED blinking task in lab 4 to a software timer task. Another UART task is create to send your name to PC terminal.

2.1 Practice A – Creating A Musical button

- playsong() is given in Appendix B
- Implement a button to play the song when the button is pressed.
  - Configure the button as a GPIO input
  - Create a task to read the button value
  - When the button is pressed, play a song

2.1 Practice B – Creating a software timer task

- Modify the LED blinking task to software timer task.
- You may use button to change the timer period.

2.2 Practice C – Create a UART task

- Using the given code of UART configuration to send your name from the board to PC
- Sending your name to PC every one second interval and show the received message via Tera Term
- Receive a character from PC and display it on OLED
- You may create a separated header file (.h file) and a .c file

2.3 Practice D – Preemptive vs Non-Preemptive

- Compare preemptive multitasking and non-preemptive multitasking
- Compare the difference between these two multitasking modes.

2.4 Practice E – Priority Task

- Changing the priority levels of musical button task and UART task
- Compare the difference between
  - When button task has higher priority level
  - When button task has lower priority level
  - When both tasks have the same priority level

Appendix A: UART Configuration

```
void uart_init(u32 baudRate)
{
GPIO_InitTypeDef GPIO_InitStructure;
USART_InitTypeDef USART_InitStructure;
NVIC_InitTypeDef NVIC_InitStructure;

RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3, ENABLE);
GPIO_PinRemapConfig(GPIO_PartialRemap_USART3, ENABLE);

//USART_TX
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10; //C10
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
GPIO_Init(GPIOC, &GPIO_InitStructure);

//USART_RX
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11;//PC11
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
GPIO_Init(GPIOC, &GPIO_InitStructure);

//Usart NVIC
NVIC_InitStructure.NVIC_IRQChannel = USART3_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=2;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);

//USART
USART_InitStructure.USART_BaudRate = baudRate;
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
USART_InitStructure.USART_Parity = USART_Parity_No;
USART_InitStructure.USART_HardwareFlowControl                                    =
USART_HardwareFlowControl_None;
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
USART_Init(USART3, &USART_InitStructure);
USART_ITConfig(USART3, USART_IT_RXNE, ENABLE);
USART_Cmd(USART3, ENABLE);
}
```

Appendix B: Play Song

```
static u32 notes[] = {
        2272, // A - 440 Hz
        2024, // B - 494 Hz
        3816, // C - 262 Hz
        3401, // D - 294 Hz
        3030, // E - 330 Hz
        2865, // F - 349 Hz
        2551, // G - 392 Hz
        1136, // a - 880 Hz
        1012, // b - 988 Hz
        1912, // c - 523 Hz
        1703, // d - 587 Hz
        1517, // e - 659 Hz
        1432, // f - 698 Hz
        1275, // g - 784 Hz
};

static u8* song =
(uint8_t*)"e2,d2,e2,d2,e2,B2,d2,c2,A2_C2,E2,A2,B2_E2,G2,B2,c4_E2,e2,d2,e2,d2,e2,B2,d2
,c2,A2_C2,E2,A2,B2_E2,c2,B2,A4";

static uint32_t getNote(uint8_t ch)
{
    if (ch >= 'A' && ch <= 'G')
        return notes[ch - 'A'];

    if (ch >= 'a' && ch <= 'g')
        return notes[ch - 'a' + 7];

    return 0;
}

static uint32_t getDuration(uint8_t ch)
{
    if (ch < '0' || ch > '9')
        return 400;
    /* number of ms */
    return (ch - '0') * 200;
}
static uint32_t getPause(uint8_t ch)
{
    switch (ch) {
    case '+':
        return 0;
    case ',':
        return 5;
    case '.':
        return 20;
    case '_':
        return 30;
    default:
        return 5;
    }
}
static void playNote(uint32_t note, uint32_t durationMs)
```

```
{
    uint32_t t = 0;

    if (note > 0) {
        while (t < (durationMs*1000)) {
            BUZ = 1;     // Turn on your buzzer (Please Edit)
            delay_us(note/2);
            BUZ = 0;     // Turn off your buzzer (Please Edit)
            delay_us(note/2);

            t += note;
        }
    }
    else {
            delay_ms(durationMs); // ms timer
    }
}

static void playSong(uint8_t *song) {
    uint32_t note = 0;
    uint32_t dur  = 0;
    uint32_t pause = 0;

    /*
     * A song is a collection of tones where each tone is
     * a note, duration and pause, e.g.
     * "E2,F4,"
     */

    while(*song != '\0') {
        note = getNote(*song++);
        if (*song == '\0')
            break;
        dur  = getDuration(*song++);
        if (*song == '\0')
            break;
        pause = getPause(*song++);

        playNote(note, dur);
        delay_us(pause);
    }
}
```