1 Từ khóa

- Extern: Dùng để chỉ biến hàm được định nghĩa trong module khác
- Biến tĩnh thì giá trị của nó giữ trong suốt quá trình chạy nhưng chỉ được truy cập
- trong chương trình mà nó định nghĩa
- volatile mô tả một biến mà giá trị có thể thay đổi không đoán trước được. Dễ bay hơi
  - Vì port có thể thay đổi bất ngờ mà arm cortex không biết trước được. Việc khai báo biến volatile là rất cần thiết để tránh những lỗi sai khó phát hiện do tính năng optimization của compiler
- .* = →
- Union : chứa nhiều biến tại cùng một địa chỉ.
- Struct tập hợp của nhiều biến.
  - trong trường hợp struct alignment member = 1 byte.
- Con trỏ hàm: là con trỏ trò đến một hàm: int (*fp)(int);

2 Các kiểu khai báo

- enum bool {true, false} boolean;
- union {float f; struct {float f; int i;} s;} U;
- struct {float f; union {float f; int i;} u;} S;
- typedef struct {double x; double y;} point;
- typedef signed char s8;
- Biến signed char là biến có dấu :[-126:127];
- Biến usigned char là biến k dấu: [0:255];

3 Tính toán nhanh

3.1 Địa chỉ:

- Ghi giá trị 01 vào địa chỉ 2000.0FFFH: (*((unsigned int*)0x20000FFF)) = 01;
- đưa giá trị 0xAE vào ô nhớ 0x00010000: *((unsigned int*)0x00010000) = 0xAE;
- gọi một hàm ở đia chỉ 0x3000: ((void ()(void))0x3000)(*);
- Đọc giá trị kiểu int ở địa chỉ 4000.00FF vào biến a
  - #define Varialbe (*((int *)0x400000FF))
- Biến tmp có kiểu int. Đưa bit thứ 16 lên 1 tmp |= 0x00010000;
- Xóa bit thứ 4. tmp &= ~0xFFFFFFEF;

3.2 BIT BANDING:

- PortA0x40004000PortB0x40005000PortC0x40006000PortD0x40007000PortE0x40024000PortF0x40025000
- #define BITBAND(addr,bitnum) ((addr & 0xF0000000) + 0x2000000+((addr & 0xFFFFF) <<5) + (bitnum <<2))
- #define MEM_ADDR(addr) (*((volatile unsigned long *) (addr)))
- #define DEVICE_REG0 0x40000000
- Tính địa chỉ Alias của bit 16 trong thanh ghi 32 bit 4000 0010H.
  - 4200000 + 10<<5+ 16<<2 = 420000E0
  - MEM_ADDR(BITBAND(DEVICE_REG0,16)) = 0x1;
- xóa bit 4 ở địa chỉ 2000 00FFH (dùng alias):
  - MEM_ADDR(BITBAND(DEVICE_REG0,4)) = 0x0;

3.3 Con trỏ:

- int (* funcPtr) (int); con trỏ hàm có đối số kiểu int và giá trị trả về kiểu int
- (int *) funcPtr (int); hàm có đối số kiểu int và gia trị trả về là con trỏ kiểu int
- (int *) (* funcPtr) (int); con trỏ hàm có đối số kiểu int giá trị trả về là con trỏ kiểu int
- Con trỏ đến con trỏ đến 1 biến kiểu integer
  - int value = 100;
  - int *ptr = &value;
  - int **p_to_p = &ptr;

4 Hàm con C

- **Hàm void clearbuffer(char *p,unsignedunsigned int n) để xóa n byte tính từ vị trí con trỏ p đang chỉ đến.**
- **Viết hàm int countSpace(char *str) đếm số khoảng trắng trong một chuỗi. (Mã ASCII của khoảng trắng là 32).**
  - #include <iostream> #include <cstring> using namespace std;
  - int countSpace(char* str){
    - int temp; for(int i=0;i<strlen(str);i++){ if(str[i] == “ “) temp+=1;} return temp;}
  - int main(){
    - char str[100]; cout<<”nhap vao mot so bat ky: “;cin.getline(str,sizeof(str));
    - cout<<”So khoang trang la :”<<countSpace(str)<<endl;return 0;}
- **Viết chương trình con strcpy sao chép chuỗi src vào chuỗi dst (chuỗi có kết thúc bằng giá trị 0)**
  - #include <string.h> #include <stdio.h> #include <stdint.h>

- void strcpy(char* src, char* dst){ for(int I = 0, I < strlen(src),i++){ dst[i] = src[i];}}

- **Viết hàm revbits nhận vào 1 tham số kiểu char và trả về kết quả kiểu char có giá trị là đảo vị trí các bit của tham số.**
  - #include<iostream> #include<stdio.h> #include<math.h> ◦ usingnamespace std;
  - longlong DecToBin(intdecimalNumber){ longlong binaryNumber = 0; int p = 0;
    - while (decimalNumber> 0){ binaryNumber += (decimalNumber % 2) * pow(10, p);
    - ++p; decimalNumber /= 2;}▪ return binaryNumber;}
  - int BinToDec(longlongbinaryNumber){ int decNumber = 0; int p = 0;
    - while (binaryNumber> 0){ decNumber += (binaryNumber % 10) * pow(2, p);
    - ++p; binaryNumber /= 10; }return decNumber; }
  - char revbits(chararg){ int temp[100]; for (int i = 0; arg> 0; i++){ temp[i] = arg % 2; arg /= 2; cout << temp[i]; } return 0; }
  - int main(){ char arg; cout <<"Nhap vao mot ky tu tu ban phim: "; cin >> arg;
    - cout <<"Chuoi nhi phan cua ky tu vua nhap la: "<< DecToBin(arg) << endl;
    - cout <<"Chuoi nhi phan bi dao lai la: "; cout << revbits(arg); return 0; }

- **Viết hàm void initPortA(void) khởi động PORTA với PA3...0 là input, PA7...4 là output sử dụng thư viện.**
  - void initPortA(void){
  - SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);//kich hoat port A
  - GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE,GPIO_PIN_7| GPIO_PIN_6| GPIO_PIN_5| GPIO_PIN_0);
  - GPOIPinTypeGPIOInput(GPIO_PORTA_BASE,GPIO_PIN_3| GPIO_PIN_2| GPIO_PIN_1| GPIO_PIN_0);
  - GPIOPadConfigSet(GPIO_PORTA_BASE,GPIO_PIN_3| GPIO_PIN_2|GPIO_PIN_1| GPIO_PIN_0, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);}

- **Viết hàm void updateOutput(void) xuất giá trị đảo của PA3...0 ra PA7...4.**

  void updateOutput(void){
  - if(GPIOPinRead(GPIO_PORTA_BASE, GPIO_PIN_3)==0)
    - GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_7, 1<<7);
  - else GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_7, 0);
  - ….

- **Viết chương trình con int mystrtoint(char*str) biến đổi chuỗi thành số integer.**
  - #include<iostream> #include<cstring> usingnamespace stdtd; int strLen(char* string){
    - char* s = string; while (*s) s++; return s – string; }
    - int mystrtoint(char* str){ int temp=0; for (int i = 0; i < strLen(str); i++){
      - switch (str[i]){ case'0': case'1': … case ‘9’: break; default: return -1; }}
      - for (int i = strLen(str) - 1; i >= 0; --i){

        - temp += (int)(str[i] - '0') * pow(10, strLen(str) - i – 1);}
        - return temp;}
    - int main(){ char str1[100]; cout <<"Nhap vao mot chuoi bat ky: ";
        - cin.getline(str1, sizeof(str1)); cout << mystrtoint(str1) << endl; return 0; }

```
#include <stdio.h>
#include <string.h>
int factorial(int n){
   if(n<0){return -1;}
   if(n==1){ return 1;}
   return n*factorial(n-1);}
void revert_armay(char S1[]){
   int i = 0;
   for(i = 6;i>=0;i--){
      printf("%c",S1[i]);
   }
}
void hien_thi(char S2[],char S3[]){
   int i;
   for(i=0;i<7;i++){
      printf("%c",S2[i]);
   }
   printf(" ");
   for(i=0;i<4;i++){
      printf("%c",S3[i]);
   }
}
void fun1(){
   printf("This is fun1\n");
}
void fun2(){
   printf("This is fun2\n");
}
#define DEBUG 1
#ifdef DEBUG
#define DEBUG_OUT(x) printf(x)
```

```
#else
#define DEBUG_OUT(x)
#endif
int main(int argc, char **argv)
{
   //Bai` 1
   int n = 5;
   printf("%d",factorial(n));
   //Bai` 2
   char S1[] = {'m','i','d','t','e','r','m'};
   revert_armay(S1);
   //Bai` 3
   char S2[] = {'m','i','d','t','e','r','m'};
   char S3[] = {'e','x','a','m'};
   hien_thi(S2,S3);
   //Bai` 4
   void(*p)(void);
   p = &fun1;
   (*p)();
   p = &fun2;
   (*p)();
   //Bai` 5
   DEBUG_OUT("testing\n");
        return 0;
}
```

5 Tiva C ngắn

- **Ba tác vụ có tính chất như sau:**
  - void T1(void): chạy sau mỗi 1000 ms, thời gian chạytrong khoảng từ 5ms đến 100ms.
  - void T2(void): chạy sau mỗi 400ms, thời gian chạy trong khoảngtừ 1 đến 10ms.
  - void T3(void): chạy sau mỗi 200ms, thời gian chạy trong khoảngtừ 1 đến 10ms.
    - Time cuối + task trước + 1ms.
    - Task1: 1 -1001 – 2001 – 3001
    - Task 2: 102 – 502 – 902 – 1302 – 1702 – 2102 – 2502 – 2902 – 3302 – 3702
    - Task3 : 113 – 313 – 513 – 713 – 913 – 1113 – 1313

- **Viết hàm void initSystick(void) khởi động Systick Timer với chu kỳ 1 ms, cho phép ngắt.**
  - Void initSystick(void){ SysTickPeriodSet(SysCtlClockGet()/1000);
    - IntMasterEnable(); SysTickIntEnable(); SysTickEnable();
    - SysTickIntRegister(Systick_Handle);}
- **Viết chương trình thực hiện 3 tác vụ trên với sơ đồ sắp lịch và SystickTimer của câu a và b. (Chỉ viết chương trình phục vụ ngắt Systick và vòng lặp chính, đồng thời khai báo các biến toàn cục cần thiết)**
  - Int task1flag = 0; Int task2flag = 0; int task3flag = 0;
  - void SystickISRHandler(){ static int systickcount = 0; Systickcount ++;
    - If(systickcount == 1 || 1001){task1flag = 1;}
    - If(systickcount == 102 || 502||902 || 1302 || 1702) { task2flag = 1;}
    - If(systickcount == 113||313||513||713||913||1113||1313||1513||1713||1913) { task3flag = 1;}
    - If(systickcount == 2000) {systickcount = 0);}
  - int main(void){ while(1) { If(task1flag == 1) {task1(); task1flag = 0;}
    - If(task2flag == 1) {task2(); task2flag = 0;}
    - If(task3flag == 1) {task3(); task3flag = 0;} } }
- **Chương trình con void initTIVA() khởi động xung nhị p cho TIVA C với tần số 50Mhz.**
  - void initTIVA() { SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ | SYSCTL_OSC_MAIN); }
- **Chương trình con voidputstr(char *str) gửi chuỗi str lên máy tính qua UART0.**
  - voidputstr(char *str){ uint 16_t i=0; while (str[i]!=’ \0’){
    - UARTCharPut (UART0_BASE, str[i]); i++; } }
- **Chương trình con voidgetstr(char *str) nhận một chuỗi từ máy tính qua UART0 cho đến khi nhận ký tự \r (ENTER).**
  - void getstr(char *str){ static uint16_t i=0; char c;
    - while (UARTCharAvail(UART0_BASE)){
      - c=UARTCharGet(UART0_BASE); *(str+i)=c; i++; if(c==’\r’) } }
- **Cho 3 hàm void Task1(void), void Task2(void), void Task3(void). Ba hàm này có timing như sau:**

  Task 1 chạy sau mỗi 40 ms. Task 2 chạy sau mỗi 50 ms. Task 3 chạy sau mỗi 60 ms.
  - int flag1 = 0;
  - int flag2 = 0;
  - int flag3 = 0;
  - void SystickISRHandler(){ static int counter; counter ++;
    - if(counter == 40){ flag1 = 1; }

        - if(counter == 50){ flag2 = 1; }
        - if(counter == 60){ flag3 = 1; counter = 0;}}
    - int main (void){ while(1){ if (flag1 == 1){ flag1 = 0; Task1(); }
        - if (flag2 == 1){ flag2 = 0; Task2(); }
        - if (flag3 == 1){ flag3 = 0; Task3(); } } }

```
void PortF_Init(void){
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
    //PD7
    //HWREG (GPIO_PORTD_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    //HWREG (GPIO_PORTD_BASE + GPIO_O_CR) | = 0x81;
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4);
    GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4, GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD_WPU);
}
void PortFIntHandle(){
    uint32_t status = 0;
    status = GPIOIntStatus(GPIO_PORTF_BASE, true);
    GPIOIntClear(GPIO_PORTF_BASE, status);
    if((status&GPIO_INT_PIN_4)==GPIO_INT_PIN_4){
        if(~GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_4)){
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3|GPIO_PIN_1|GPIO_PIN_2,
GPIO_PIN_3);
        }
        SysCtlDelay(SysCtlClockGet()/3);
    }
}
void Interrupt_GPIO_Init(){
    GPIOIntTypeSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_FALLING_EDGE);
    GPIOIntEnable(GPIO_PORTF_BASE, GPIO_INT_PIN_4);
    //GPIOIntRegister(GPIO_PORTF_BASE, PortFIntHandle);
    IntEnable(INT_GPIOF);
    //IntMasterEnable();
    //IntPrioritySet(INT_GPIOF, 0xE0);
}
void Systick_Handle(){
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, GPIO_PIN_3);
}
void Systick_Init(){
    SysTickPeriodSet(SysCtlClockGet()/1000);
    SysTickEnable();
    SysTickIntEnable();
    //SysTickIntRegister(Systick_Handle);
    //IntMasterEnable();
}
void UART0IntHandler(void)
{
    unsigned long ulStatus;
    unsigned char received_character;
    ulStatus = UARTIntStatus(UART0_BASE, true); //get interrupt status
    UARTIntClear(UART0_BASE, ulStatus); //clear the asserted interrupts
    while(UARTCharsAvail(UART0_BASE)) //loop while there are characters in the
receive FIFO
    {
        received_character = UARTCharGet(UART0_BASE);
```

```
        UARTCharPutNonBlocking(UART0_BASE, received_character); //echo character
        if (received_character == 13) UARTCharPutNonBlocking(UART0_BASE, 10);
//if CR received,issue LF as well
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2); //blink LED
        SysCtlDelay(SysCtlClockGet() / (1000 * 3)); //delay ~1 msec
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0); //turn off LED
    }
}
void Uart_Init(){
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);
    UARTStdioConfig(0, 115200, 16000000);
    UARTFIFOLevelSet(UART0_BASE, UART_FIFO_TX4_8, UART_FIFO_RX4_8);
    UARTFIFOEnable(UART0_BASE); //enable FIFOs
    //IntMasterEnable(); //enable processor interrupts
    //IntEnable(INT_UART0); //enable the UART interrupt
    //UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT); //enable Receiver
interrupts
    //UARTIntRegister(UART0_BASE, UART0IntHandler);
    UARTprintf("\033[2J\033[HEnter Text: "); // erase screen, put cursor at home
position (0,0),prompt
}
void Timer0Handle(){
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    UARTprintf("Hello");
}
void Timer_Init(){
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
    TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
    TimerLoadSet(TIMER0_BASE, TIMER_A, (SysCtlClockGet()-1));

    //TimerIntRegister(TIMER0_BASE, TIMER_A, Timer0Handle);
    //TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    //IntEnable(INT_TIMER0A);
    TimerEnable(TIMER0_BASE, TIMER_A);
    //IntMasterEnable();
}
```

6 Tiva C lớn

6.1 Máy trạng thái

6.1.1 File .h

```
#ifndef FSM_FSM_H_
#define FSM_FSM_H_
#include "stdint.h"
extern uint16_t ledTimer;
void ledRedStateMachineUpdate(void);
void ledGreenStateMachineUpdate(void);
extern uint8_t redCounter;
extern uint8_t greenCounter;
extern uint8_t redFlag;
extern uint8_t greenFlag;
#endif
```

6.1.2 File .c

```
uint8_t redCounter = 0;
uint8_t greenCounter = 0;
uint8_t redFlag = 0;
uint8_t greenFlag = 0;
uint16_t ledTimer = 0;
enum ledNumber{LEDRED=0,LEDGREEN,LEDBLUE};
const   uint32_t    ledPin[3]   =   {GPIO_PIN_1,GPIO_PIN_2,GPIO_PIN_3};
enum ledState{OFF=0,ON};
typedef enum SW{PRESSED,RELEASED} sw_t;

void ledControl(enum ledNumber led,enum ledState State){
    if(State){
        GPIOPinWrite(GPIO_PORTF_BASE,ledPin[led], ledPin[led]);
    }
    else{
        GPIOPinWrite(GPIO_PORTF_BASE,ledPin[led], 0);
    }
}
sw_t switchState(SWnumber){
    switch(SWnumber){
    case 1:
        if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_4) == 0){
            return PRESSED;
        }
        else{
            return RELEASED;
        }
    case 2:
        if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0) == 0){
            return PRESSED;
        }
        else{
            return RELEASED;
        }
    default: return PRESSED;
    }
}
//////////////////////////////////////
RED//////////////////////////////////////////////
typedef enum {S1=0,S2,S3,S4} State_Red;
static State_Red State_R = S1;
void ledRedStateMachineUpdate(void){
    switch(State_R){
    case S1:
        if(switchState(1)==PRESSED){
            State_R = S2;
            //ledTimer = 1000;
        }
        break;
    case S2:
        if(switchState(1)==RELEASED){
            State_R = S3;
            //ledTimer = 1000;
        }
        break;
    case S3:
        if(switchState(1)==PRESSED){
            State_R = S4;
            //ledTimer = 1000;
        }
```

```
        break;
    case S4:
        if(switchState(1)==RELEASED){
            State_R = S1;
            //ledTimer = 1000;
        }
        break;
    }
    switch(State_R){
    case S1:
    case S4:
        ledControl(LEDRED,OFF);
        break;
    case S2:
    case S3:
        ledControl(LEDRED,ON);
        break;
    }
}
////////////////////////////////////////////////
GREEN//////////////////////////////////////////////////
typedef enum {S_1=0,S_2,S_3,S_4,S_5,S_6} State_Green;
static State_Green State_G;
void ledGreenStateMachineUpdate(void){
    switch(State_G){
    case S_1:
        if(switchState(2)==PRESSED){
            State_G = S_2;
            ledTimer = 3000;
        }
        break;
    case S_2:
        if(switchState(2)==RELEASED){
            State_G = S_1;
        }
        else if(ledTimer==0){
            State_G = S_3;
        }
        break;
    case S_3:
        if(switchState(2)==RELEASED){
            State_G = S_4;
        }
        break;
    case S_4:
        if(switchState(2)==PRESSED){
            State_G = S_5;
            ledTimer = 6000;
        }
        break;
    case S_5:
        if(switchState(2)==RELEASED){
            State_G = S_4;
        }
        else if(ledTimer==0){
            State_G = S_6;
        }
        break;
    case S_6:
        if(switchState(2)==RELEASED){
            State_G = S_1;
        }
```

```
        break;
    }
    switch(State_G){
    case S_1:
    case S_2:
    case S_6:
        ledControl(LEDGREEN,OFF);
        break;
    case S_3:
    case S_4:
    case S_5:
        ledControl(LEDGREEN,ON);
    }
}
```

6.1.3 File main.c

```
#include <stdbool.h>
#include <stdint.h>
#include "stdlib.h"
#include <stdbool.h>
#include "inc/hw_types.h"//macro
#include "inc/hw_uart.h"
#include "inc/hw_memmap.h"//base address of the memories and peripherals.
#include "inc/hw_gpio.h"//Defines and Macros for GPIO hardware.
#include "inc/hw_ints.h"//Macros that define the interrupt assignment
#include "inc/hw_pwm.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"//SYSCTL register
#include "driverlib/rom.h"//ROM library
#include "driverlib/rom_map.h"//ROM map
#include "driverlib/pin_map.h"//ort/Pin Mapping Definitions
#include "driverlib/gpio.h"//Defines and Macros for GPIO API
#include "driverlib/interrupt.h"//Interrupt library
#include "driverlib/timer.h"//Prototypes for the timer module
#include "driverlib/uart.h"//Defines and Macros for the UART
#include "utils/uartstdio.h"//UARTSTDIO
#include "driverlib/systick.h"//Systick
#include "driverlib/udma.h"
#include "driverlib/pwm.h"
#include "driverlib/ssi.h"
#include <string.h>
#include "Config_Port/Config_Port.h"
#include "FSM/FSM.h"
void Systick_Init(void);
void Systick_Handler(void){
    if(ledTimer>0){
        ledTimer--;
    }
    if(redCounter==0){
        redCounter = 50;
        redFlag = 1;
    }
    else{
        redCounter--;
    }
    if(greenCounter==0){
        greenCounter = 50;
        greenFlag = 1;
    }
    else{
        greenCounter--;
    }
```

```
    if(redFlag){
        redFlag = 0;
        ledRedStateMachineUpdate();
    }
    if(greenFlag){
        greenFlag = 0;
        ledGreenStateMachineUpdate();
    }
}
void Systick_Init(){
    SysTickPeriodSet(SysCtlClockGet()/1000);
    IntMasterEnable();
    SysTickEnable();
    SysTickIntEnable();
    SysTickIntRegister(Systick_Handler);
}
int main(void){
    SysCtlClockSet(SYSCTL_SYSDIV_2_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|
SYSCTL_OSC_MAIN);
    PortF_Init();
    Systick_Init();
    while(1){

    }
}
```

6.2 Tạo PWM bằng Systick timer:

```
uint32_t Duty = 25;
void Systick_Handler(){
    static uint32_t High = 0;
    static uint32_t Low = 0;
    if(Duty>99){
        GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_0, GPIO_PIN_0);
    }else if(High!=0){
        High--;
        GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_0, GPIO_PIN_0);
    }else if(Low!=0){
        Low--;
        GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_0, 0);
    }else{
        High = Duty*10-1;
        Low = 999-High;
    }
}
int main(){
    SysCtlClockSet(SYSCTL_SYSDIV_8|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|
SYSCTL_OSC_MAIN);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_0);
    Systick_Init();
    SysTickIntRegister(Systick_Handler);
    IntMasterEnable();
    while(1){
    }
}
```

7 FREERTOS

7.1 TASK

7.1.1 Lab1

```
TaskHandle_t TaskHandle2;
char *text1 = "Task 1 is running\n";
char *text2 = "Task 2 is running\n";
void vTask1(void *pvParameters){
    char *pcText;
    pcText = (char *)pvParameters;
    while(1){
        printf("%s",pcText);
        vTaskDelay(pdMS_TO_TICKS(100));
    }
}
void vTask2(void *pvParameters){
    char *pcText;
    pcText = (char *)pvParameters;
    portTickType xLastWakeTime = xTaskGetTickCount();
    while(1){
        printf("%s",pcText);
        vTaskDelayUntil(&xLastWakeTime, pdMS_TO_TICKS(100));
    }
}
int main(){
    vSetupHardware();
    UART_Init();
    xTaskCreate(vTask1, "Task 1", 200, (void *)text1, 1, NULL);
    xTaskCreate(vTask2, "Task 2", 200, (void *)text2, 1, &TaskHandle2);
    vTaskStartScheduler();
    while(1){

    }
}
```

7.1.2 Lab2

```
TaskHandle_t TaskHandle2;
char *text1 = "Task 1 is running\n";
char *text2 = "Task 2 is running\n";
void vTask1(void *pvParameters){
    char *pcText;
    pcText = (char *)pvParameters;
    UBaseType_t uxPriority = uxTaskPriorityGet(TaskHandle2);
    while(1){
        printf("%s",pcText);
        vTaskPrioritySet(TaskHandle2, uxPriority+2);
        //vTaskDelay(pdMS_TO_TICKS(100));
    }
}
void vTask2(void *pvParameters){
    char *pcText;
    pcText = (char *)pvParameters;
    portTickType xLastWakeTime = xTaskGetTickCount();
    UBaseType_t uxPriority = uxTaskPriorityGet(NULL);
    while(1){
        printf("%s",pcText);
        vTaskPrioritySet(TaskHandle2, uxPriority-2);
        //vTaskDelayUntil(&xLastWakeTime, pdMS_TO_TICKS(100));
```

```
    }
}
int main(){
    vSetupHardware();
    UART_Init();
    xTaskCreate(vTask1, "Task 1", 200, (void *)text1, 2, NULL);
    xTaskCreate(vTask2, "Task 2", 200, (void *)text2, 1, &TaskHandle2);
    vTaskStartScheduler();
    while(1){

    }
}
```

7.1.3 Lab3

```
TaskHandle_t TaskHandle2;
char *text1 = "Task 1 is running\n";
char *text2 = "Task 2 is running\n";
void vTask2(void *pvParameters);
void vTask1(void *pvParameters){
    char *pcText;
    pcText = (char *)pvParameters;

    while(1){
        printf("%s",pcText);
        xTaskCreate(vTask2, "Task 2", 200, (void *)text2, 2, &TaskHandle2);
        vTaskDelay(pdMS_TO_TICKS(100));
    }
}
void vTask2(void *pvParameters){
    char *pcText;
    pcText = (char *)pvParameters;
    portTickType xLastWakeTime = xTaskGetTickCount();

    while(1){
        printf("%s",pcText);
        vTaskDelete(TaskHandle2);
        vTaskDelayUntil(&xLastWakeTime, pdMS_TO_TICKS(100));
    }
}
int main(){
    vSetupHardware();
    UART_Init();
    xTaskCreate(vTask1, "Task 1", 200, (void *)text1, 1, NULL);
    //xTaskCreate(vTask2, "Task 2", 200, (void *)text2, 1, &TaskHandle2);
    vTaskStartScheduler();
    while(1){

    }
}
```

7.2 QUEUE

7.2.1 Lab1

```
TaskHandle_t xTask1Handle;
TaskHandle_t xTask2Handle;
QueueHandle_t xQueue;
void vTask1(void *pvParameters){
    uint32_t lValueToSend;
    bool xStatus;
    lValueToSend = (portLONG)pvParameters;
```

```
    while(1){
        xStatus = xQueueSendToBack(xQueue,&lValueToSend,0);
        if(xStatus != pdPASS){
            UARTprintf("Gui khong duoc\n");
        }
    }
}
void vTask2(void *pvParameters){
    uint32_t lReceivedValue;
    bool xStatus;
    while(1){
        xStatus = xQueueReceive(xQueue,&lReceivedValue,portMAX_DELAY);
        if(xStatus == pdPASS){
            UARTprintf("Received = %d\n",lReceivedValue);
        }else{
            UARTprintf("FAIL");
        }
    }
}
int main(){
    SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|
SYSCTL_OSC_MAIN);
    Uart_Init();
    PortF_Init();
    xQueue = xQueueCreate(5,sizeof(long));
    xTaskCreate(vTask1,"Task 1",200,(void *)100,1,&xTask1Handle);
    xTaskCreate(vTask1,"Task 1",200,(void *)200,1,&xTask2Handle);
    xTaskCreate(vTask2,"Task 2",200,NULL,2,NULL);
    //IntMasterEnable();
    vTaskStartScheduler();

    while(1){

    }
}
```

7.2.2 Lab2

```
QueueHandle_t xAQueue;
//QueueHandle_t xBQueue;
//QueueHandle_t xCQueue;
void vTask1(void *Parameters){
    portLONG lValueToSend;
    while(1){
        if(~GPIO_PORTF_DATA_R&0x01){
            lValueToSend = 1;
            xQueueSendToBack(xAQueue,&lValueToSend,0);
            vTaskDelay(pdMS_TO_TICKS(1000));
        }else if(~GPIO_PORTF_DATA_R&0x10){
            lValueToSend = 2;
            xQueueSendToBack(xAQueue,&lValueToSend,0);
            vTaskDelay(pdMS_TO_TICKS(1000));
        }else{
            lValueToSend = 3;
            xQueueSendToBack(xAQueue,&lValueToSend,0);
            vTaskDelay(pdMS_TO_TICKS(1000));
        }
    }
}
void vTask2(void *pvParameters){
    portLONG lReceiveValue;
    portBASE_TYPE xStatus;
    while(1){
```

```
        xStatus = xQueueReceive(xAQueue, &lReceiveValue, pdMS_TO_TICKS(100));
        if(xStatus == pdPASS){
            if(lReceiveValue==1){
                printf("%s","Switch 2 is pressed");
                GPIO_PORTF_DATA_R = (GPIO_PORTF_DATA_R & 0x00)+0x08;
            }else if(lReceiveValue==2){
                printf("%s","Switch 1 is pressed");
                GPIO_PORTF_DATA_R = (GPIO_PORTF_DATA_R & 0x00)+0x04;
            }else if(lReceiveValue==3){
                printf("%s","No Switch is pressed");
                GPIO_PORTF_DATA_R = (GPIO_PORTF_DATA_R & 0x00)+0x02;
            }
        }
    }
}
int main(){
    vSetupHardware();
    UART_Init();
    xAQueue = xQueueCreate(5,sizeof(portLONG));
    //xBQueue = xQueueCreate(5,sizeof(portLONG));
    //xCQueue = xQueueCreate(5,sizeof(portLONG));
    xTaskCreate(vTask1, "Task 1", 200, NULL, 1, NULL);
    xTaskCreate(vTask2, "Task 2", 200, NULL, 2, NULL);
    vTaskStartScheduler();
    while(1){

    }
}
```

7.2.3 Lab3

```
TaskHandle_t xTask1Handle;
TaskHandle_t xTask2Handle;
QueueHandle_t xQueue;
TimerHandle_t xAutoReloadTimer;
TimerHandle_t xOneShotTimer;
typedef enum {
    zero,
    one
} DataSource;
typedef struct {
    uint8_t ucValue;
    DataSource eDataSource;
} Data_t;
Data_t xStructsToSend[2] =
{
 {100,zero},
 {200,one}
};
void vTask1(void *pvParameters){
    bool xStatus;
    while(1){
        xStatus = xQueueSendToBack(xQueue,pvParameters,0);//pdPASS
        if(xStatus!=pdPASS){
            UARTprintf("Khong gui duoc");
        }
    }
}
void vTask2(void *pvParameters){
    bool xStatus;
    Data_t xReceived;
    while(1){
        xStatus = xQueueReceive(xQueue,&xReceived,portMAX_DELAY);//pdPASS
```

```
        if((xStatus==pdPASS)&&(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0)==0)){
            if(xReceived.eDataSource == zero){
                UARTprintf("From Sender 1 = %d",xReceived.ucValue);
                xTimerReset(xAutoReloadTimer,portMAX_DELAY);//pdPASS and pdFAIL
                xTimerReset(xOneShotTimer,portMAX_DELAY);
            }else{
                UARTprintf("From Sender 2 = %d",xReceived.ucValue);
            }
        }
    }
}
void prvAutoReloadTimerCallback(TimerHandle_t xTimer){
    TickType_t xTimeNow;
    xTimeNow = xTaskGetTickCount();
    static uint32_t ulExecutionCount = 0;
    ulExecutionCount = ( uint32_t ) pvTimerGetTimerID( xTimer );
    ulExecutionCount++;
    if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_4)==0){
        ulExecutionCount = 0;
    }
    vTimerSetTimerID( xTimer, ( void * ) ulExecutionCount );
    UARTprintf("%d\n",xTimeNow);
    UARTprintf("%d\n",ulExecutionCount);
    xTimerChangePeriod(xTimer,pdMS_TO_TICKS(500),0);//pdPASS and pdFAIL
    if( ulExecutionCount == 3 ){
        xTimerStop( xTimer, 0 );
    }
    //xTimerReset(xAutoReloadTimer,portMAX_DELAY);
}
void prvOneShotTimerCallback(TimerHandle_t xTimer){
    TickType_t xTimeNow;
    xTimeNow = xTaskGetTickCount();
    UARTprintf("Hello %d",xTimeNow);
}
int main(){
    SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|
SYSCTL_OSC_MAIN);
    Uart_Init();
    PortF_Init();
    xQueue = xQueueCreate(5,sizeof(Data_t));//return NULL = failed
    xTaskCreate(vTask1,"Task 1",200,&xStructsToSend[0],1,&xTask1Handle);//pdPASS
vs pdFAIL
    xTaskCreate(vTask1,"Task 1",200,&xStructsToSend[1],1,&xTask2Handle);
    xTaskCreate(vTask2,"Task 2",200,NULL,2,NULL);
    xAutoReloadTimer = xTimerCreate("Auto Reload", pdMS_TO_TICKS(1000), pdTRUE,
0, prvAutoReloadTimerCallback);//NULL = failed
    xOneShotTimer = xTimerCreate("Auto Reload", pdMS_TO_TICKS(3333), pdFALSE, 0,
prvOneShotTimerCallback);
    xTimerStart(xAutoReloadTimer,0);//pdPASS and pdFALSE
    xTimerStart(xOneShotTimer,0);
    //IntMasterEnable();
    //UARTprintf("%d",xPortGetFreeHeapSize());
    vTaskStartScheduler();
}
```

7.3 INTERRUPT

7.3.1 Lab1

```
TaskHandle_t xTask1Handle;
TaskHandle_t xTask2Handle;
SemaphoreHandle_t xBinarySemaphore;
```

```
void Timer0Handle(){
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, GPIO_PIN_3);
    portBASE_TYPE xHigherPriorityTaskWoken = pdFALSE;
    xSemaphoreGiveFromISR( xBinarySemaphore, &xHigherPriorityTaskWoken );
    portEND_SWITCHING_ISR( xHigherPriorityTaskWoken );
}
void vTask1(void *pvParameters){
    while(1){
        UARTprintf("Task 1 is running\n");
        vTaskDelay(pdMS_TO_TICKS(500));
    }
}
void vTask2(void *pvParameters){
    xSemaphoreTake(xBinarySemaphore,0);
    while(1){
        if(xSemaphoreTake(xBinarySemaphore,portMAX_DELAY)){
            UARTprintf("Handler Task");
        }
    }
}
int main(){
    SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|
SYSCTL_OSC_MAIN);
    Uart_Init();
    PortF_Init();
    Timer_Init();
    IntMasterEnable();
    vSemaphoreCreateBinary(xBinarySemaphore);
    xTaskCreate(vTask1,"Task 1",200,(void *)0,1,&xTask1Handle);
    xTaskCreate(vTask2,"Task 2",200,(void *)0,3,&xTask2Handle);
    vTaskStartScheduler();
    while(1){

    }
}
```

7.3.2 Lab2

```
volatile unsigned int Idle = 0;
void vApplicationIdleHook( void )
{
    /* This example does not use the idle hook to perform any processing. */
    Idle++;
}
/*-----------------------------------------------------------*/

void vApplicationTickHook( void )
{
    /* This example does not use the tick hook to perform any processing. */
}
/*-----------------------------------------------------------*/
TaskHandle_t xTask1Handle;
TaskHandle_t xTask2Handle;
QueueHandle_t xQueue;
QueueHandle_t xQueue2;
SemaphoreHandle_t xSemaphore;
uint8_t State;
char string[10];
int i = 0;
void vTask1(void *pvParameters){
    TickType_t xLastTimeWaken = xTaskGetTickCount();
    char led[3][10] = {
```

```
                        "ledon",
                        "ledoff",
                        "ledtoggle"
    };
    while(1){
        if(xSemaphoreTake(xSemaphore,portMAX_DELAY)==pdPASS){
            if(memcmp(led[0],string,5)==0){
                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_PIN_1);
                UARTprintf("LEDRED sang\n");
            }else if(memcmp(led[1],string,6)==0){
                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0);
                UARTprintf("LEDRED tat\n");
            }
            else if((memcmp(led[2],string,9)==0)&&(State!=2)){
                UARTprintf("LEDRED nhap nhay");
                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0);
                vTaskDelayUntil(&xLastTimeWaken, pdMS_TO_TICKS(1000));
                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_PIN_1);
                vTaskDelayUntil(&xLastTimeWaken, pdMS_TO_TICKS(1000));
                xSemaphoreGive(xSemaphore);
            }
            //UARTprintf("%s",string);
        }
    }
}
void vTask2(void *pvParameters){
    unsigned char ucReceivedValue;
    portBASE_TYPE xStatus;
    while(1){
        if(i==10){
            i = 0;
        }
        xStatus = xQueueReceive(xQueue,&ucReceivedValue,portMAX_DELAY);
        if(xStatus == pdPASS){
            string[i] = ucReceivedValue;
            i = i + 1;

        }
    }
}

void UART0IntHandler(){
    UBaseType_t xStatus;
    xStatus = UARTIntStatus(UART0_BASE, true);
    UARTIntClear(UART0_BASE, xStatus);
    portBASE_TYPE xHigherPriorityTaskWoken = pdFALSE;
    unsigned char received_character;
    while(UARTCharsAvail(UART0_BASE)){
        received_character = UARTCharGet(UART0_BASE);
        UARTCharPutNonBlocking(UART0_BASE, received_character); //echo character
        if(received_character == 13){
            i = 0;
            UARTprintf("\n\r");
            xSemaphoreGiveFromISR(xSemaphore,&xHigherPriorityTaskWoken);
            portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
        }else{

xQueueSendFromISR(xQueue,&received_character,&xHigherPriorityTaskWoken);
        }
        if(received_character == 'r'){
            State = 2;
        }else if(received_character == 'l'){
```

```
            State = 0;
        }
    }
}
int main(){
    SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|
SYSCTL_OSC_MAIN);
    Uart_Init();
    PortF_Init();
    xTaskCreate(vTask1,"Task 1",1000,(void *)0,2,&xTask1Handle);
    xTaskCreate(vTask2,"Task 2",1000,(void *)0,1,&xTask2Handle);
    xQueue = xQueueCreate(8,sizeof(char));
    xSemaphore = xSemaphoreCreateBinary();
    IntMasterEnable();
    //UARTprintf("%d",xPortGetFreeHeapSize());
    vTaskStartScheduler();
    while(1){

    }
}
```

7.4 HOOK

7.4.1 Idle Hook Function

void vApplicationIdleHook(void); ✦ configUSE_IDLE_HOOK is set to 1

7.4.2 Tick Hook Function

void vApplicationTickHook(void); ✦ configUSE_TICK_HOOK is set to 1

7.4.3 Malloc Failed Hook Function

void vApplicationMallocFailedHook(void);✦ configUSE_MALLOC_FAILED_HOOK is set to 1

7.4.4 Câu hỏi

1. Giải thích sự khác nhau cơ bản giữa hệ điều hành co-operative (non-preemptive) và pre-emptive

Hệ điều hành non preemtive là hệ điều hành có các task thực thi cho đến khi hoàn thành, không nhường cpu cho các task khác trong quá trình thực thi. Hệ điều hành preemtive là hệ điều hành có task đang thực thi sẽ dừng lại nhường cpu cho task khác nếu đang cần cpu.

2. biếnxHigherPriorityTaskWokensẽ có giá trị như thế nào? Giải thích các trường hợp có thể xảy ra

Sau khi thực hiện lệnh này thì nếu có task nào đang chờ queue mà có mức ưu tiên cao hơn task đang thực hiện trước ngắt thì biến này set lên 1. mà macro portEND_SWITCH_TASK sẽ thực hiện chuyển tác vụ tiếp theo được thực hiện là tác vụ đang chờ queue.

3. Để sử dụng được lệnh xQueueSendFromISR, mức ưu tiên của ISR mà lệnh này được gọi phải như thế nào?

Đế sử dụng các API của hệ thống thì mức ưu tiên của task này phải nhỏ hơn hoặc bằng mức giá trị đc định nghĩa bởi #define configMAX_SYSCALL_INTERUPT_PRIORITY

4. Viết lệnh khởi tạo 1 queue tên là myQueue, gồm 100 phần tử kiểu float.

   xQueueHandle myQueue; myQueue = xqueueCreate(100, sizeof(float));

5. Các lệnh sau đây có thể làm cho tác vụ bị block hay không. Giải thích

   1. portBASE_TYPExQueueSend : có thể bị block, vì khi queue đầy thì không thể gửi data và queue nên task se bị tràn cho đến khi hết thơi gian xTicktoWait(hoặc queue rỗng trong lúc task bị blocked.)
   2. portBASE_TYPExQueueReceive: âcó thể bị blocked vì nến trong queue không có data thì task sẽ đi vào trạng thái block cho đến hết thời gian xTickToWait
   3. portBASE_TYPExQueueSendFromISR: hàm này không làm task bị blocked vì hàm này chỉ dc gọi trong ngắt mà ngắt không phải 1 task.
   4. portBASE_TYPExQueueReceiveFromISR : hàm này cũng không làm tác vụ bị blocked
   5. portBASE_TYPExQueuePeek: giống như xQueueReceive nhưng khi đọc data ra thì data vẫn còn trong queue.

6. Giải thích chức năng ngắt SystemTick trong chương trình sử dụng FreeRTOS

   Chức năng của SystemTick : tạo ra các ngắt với thời gian bẽ hơn nhiều so với thời gian thực thi các hàm, mỗi khi nhảy vào ngắt thì hệ điều hành freeRtos sẽ xem sát các tác vụ nào cần thực thi, chuyển trạng thái.

7. Giải thích chức năng task IDLE: Chức năng idle task: là task có độ ưu tiên thấp nhất, để chạy các yêu cầu có độ ưu tiên thấp, ngoài ra khi idle task thực thi để đảm hệ điều hành phải có một task chạy khi không có task nào đc chạy, khi chạy task idle thì hệ điều hành sẽ giải phóng các ô nhớ đã cấp phát cho task khi có task gọi hàm vTaskDelete();

8. Lệnh taskYIELD () sẽ làm cho hệ điều hành thự c hiện context switch (lự a chọn tác vụ đang ở trạng thái READY có độ ư u tiên cao nhất để chạy mà không cần chờ đến ngắt systemTick)

7.5 Bài tập