GF's Godavari College of Engineering, Jalgaon.

Dept. of Electronics & Telecommunication Engineering

Lab Manual

EMBEDDED SYSTEMS (EL-II)

BE (E&TC) Semester-II

Prepared by
Vijay D. Chaudhari (Assistant Professor)

Yr. 2017-18

INDEX

Experiment No. 1 Date-

Aim: Study of IDE (Integrated Development Environment).

Objective:

To understand characteristic, features, uses, advantages and examples of IDE.

Prerequisites:

Basic knowledge of Microprocessor and Microcontroller.

Theory:

1. **Characteristics**

2. **Features**

3. **Uses**

4. **Advantages**

5. **Examples of IDE**

Conclusion:

Experiment No. 2 Date-

Aim: Embedded C program to explore the timer (on chip) using ARM.

Objective:

This program demonstrates blinking of LED using GPIO after specified interval & creating rotating LED appearance.

Requirements:

1. OASIS ARM7(TITAN) BOARD
2. RS232 Serial cable
3. 9 Volt DC Power Supply
4. Triton IDE

Pin Assignment: LED Display Interface

Sr.No	Signal	Description
1	Pin 45 (P0.15)	L8
2	Pin 46 (P0.16)	L7
3	Pin 47 (P0.17)	L6
4	Pin 53 (P0.18)	L5
5	Pin 54 (P0.19)	L4
6	Pin 55 (P0.20)	L3
7	Pin 1 (P0.21)	L2
8	Pin 2 (P0.22)	L1

Theory:

A *timer* is a device that generates a signal pulse at specified time intervals. A time interval is a "real-time" measure of time, such as 3 milliseconds. These devices are extremely useful in systems in which a particular action, such as sampling an input signal or generating an output signal, must be performed every *X* time units.

Internally, a simple timer may consist of a register, counter, and an extremely simple controller. The register holds a count value representing the number of clock cycles that equals the desired real-time value. This number can be computed using the simple formula:

Number of clock cycles = Desired real-time value / Clock cycle

For example, to obtain a duration of 3 milliseconds from a clock cycle of 10 nanoseconds (100 MHz), we must count (3x10-6 s / 10x10-9 s/cycle) = 300 cycles. The counter is initially loaded with the count value, and then counts down on every clock cycle until 0 is reached, at which point an output signal is generated, the count value is reloaded, and the process repeats itself. To use a timer, we must configure it (write to its registers), and respond to its output signal.

Features:

A 32-bit Timer/Counter with a programmable 32-bit Prescaler.

• Counter or Timer operation

• Up to four 32-bit capture channels per timer, that can take a snapshot of the timer value when an input signals transitions. A capture event may also optionally generate an interrupt.

• Four 32-bit match registers that allow:

– Continuous operation with optional interrupt generation on match.

– Stop timer on match with optional interrupt generation.

– Reset timer on match with optional interrupt generation.

• Up to four external outputs corresponding to match registers, with the following capabilities:

– Set low on match.

– Set high on match.

– Toggle on match.

– Do nothing on match.

The Timer/Counter is designed to count cycles of the peripheral clock (PCLK) or an externally-supplied clock, and can optionally generate interrupts or perform other actions at specified timer values, based on four match registers. It also includes four capture inputs to trap the timer value when an input signal transitions, optionally generating an interrupt.

Algorithm:

Program:

```
#include <LPC21XX.h>

#define ALL 0x8000
```

```
#define DELAY 0x100000
void Delay();
unsigned int rp();//define fuction
unsigned int n=ALL; //globally set n=all
int main()
{
        {
        *IODIR0=0x7F8000;
        while(1)
                {
                *IOSET0=n;
                Delay();
                *IOCLR0=n;
                Delay();
                rp();
                }
        }
}
void Delay()
{
        unsigned int cnt;
        for(cnt=0;cnt<DELAY;cnt++);
}
unsigned int rp()
```

```
{
        if (n<=0x400000 && n>=0x8000)
        n<<=1;
        else
        n=0x8000;
        return n;
}
```

Application of Experiment:

1. Interval Timer for counting internal events.
2. Pulse Width Demodulator via Capture inputs.
3. Free running timer.

Procedure:

1. Connect 9 V DC Power supply to the OASIS TITAN Board.
2. Connect the Board with the COM port of the PC using the serial cable.
3. Generate .hex file using Triton IDE.
4. Download the .hex file.
5. Put the board in RUN mode and observe the output

Result:

Conclusion:

Oral questions:

1. If timer interval is a "real time" measure of time, what is the use of such devices?
2. What are the main parts of timer?
3. What is PCLK?
4. What is the purpose of IOSET and IOCLR?

Experiment No. 3 Date-

Aim: Embedded C program for interfacing LED and Switch with ARM.

Objective:

The LED will glow when corresponding switch is pressed.

Requirements:

1. OASIS ARM7(TITAN) BOARD
2. RS232 Serial cable
3. 9 Volt DC Power Supply
4. Triton IDE

Pin Assignment: LED interface (As mentioned In previous expt.)

Theory:

This program demonstrates glow of LED when corresponding switch is pressed.

Algorithm:

Program: Embedded C program for interfacing LED and switch

```
/*****************************************************************
      Switches_LEDS_main.c ******************************************/
#include<LPC21XX.h>
#define ALL_LEDS 0x0003C000
#define LED1 0x00008000
#define LED2 0x00010000
#define LED3 0x00020000
#define LED4 0x00040000
#define switch1 0x00080000
#define switch2 0x00100000
#define switch3 0x00200000
#define switch4 0x00400000

int main(void)
{
      *IODIR0 = ALL_LEDS;
      while(1)
      {
            if(!(*IOPIN0 & switch1))
            {
                  *IOSET0 = LED1;
            }
            else
            {
                  *IOCLR0=LED1;
            }
            if(!(*IOPIN0 & switch2))
            {
                  *IOSET0 = LED2;
            }
            else
            {
                  *IOCLR0=LED2;
            }
            if(!(*IOPIN0 & switch3))
            {
                  *IOSET0 = LED3;
            }
            else
            {
                  *IOCLR0=LED3;
            }
            if(!(*IOPIN0 & switch4))
            {
                  *IOSET0 = LED4;
            }
            else
            {
                  *IOCLR0=LED4;
            }
            //*IOCLR0 = 0x00000000;
      }
      return 0;
}
```

Application of Experiment:

1. Door-lock control
2. Safe box
3. Simple access controller
4. Vehicle control
5. ATM

Procedure:

1. Connect 9 V DC Power supply to the OASIS TITAN Board.
2. Connect the Board with the COM port of the PC using the serial cable.
3. Generate .hex file using Triton IDE.
4. Download the .hex file.
5. Put the board in RUN mode.
6. Press any key on on-board Keypad and you can observe the output on LCD.

Result:

Conclusion:

Oral questions:

Experiment No. 4 Date-

Aim: Embedded C program for interfacing LCD and ARM

Objective:

This program displays given string to LCD using ARM.

Requirements:

5. OASIS ARM7(TITAN) BOARD
6. RS232 Serial cable
7. 9 Volt DC Power Supply
8. Triton IDE

Pin Assignment: LCD display interface

Sr. No.	Signal	Description
1	Pin 16 (P1.16)	Data 0
2	Pin 12 (P1.17)	Data 1
3	Pin 8 (P1.18)	Data 2
4	Pin 4 (P1.19)	Data 3
5	Pin 48 (P1.20)	Data 4
6	Pin 44 (P1.21)	Data 5
7	Pin 40 (P1.22)	Data 6
8	Pin 36 (P1.23)	Data 7
9	Pin 13 (P0.28)	RS
10	Pin 14 (P0.29)	EN
11	GND	WR

Theory:

This program demonstrates LCD display using keypad interfacing with ARM7 through GPIO. For this experiment we use 16*2 LCD display which shows the given string in program on the display.

Algorithm:

Program:

```
#include <LPC21XX.h>

#include <board.h>

int main(void)

{

        char key;

        q_lcdinit(TITAN);

        q_printf("%s \r\nkey;","Hello");

        q_displaylcd("Hello", 5);

}

}
```

Application of Experiment:

1. Door-lock control
2. Safe box
3. Simple access controller
4. Vehicle control
5. ATM

Procedure:

7. Connect 9 V DC Power supply to the OASIS TITAN Board.
8. Connect the Board with the COM port of the PC using the serial cable.
9. Generate .hex file using Triton IDE.
10. Download the .hex file.
11. Put the board in RUN mode.
12. Press reset key and you can observe the output string on LCD.

Result:

Conclusion:

Oral questions:

1. Why we use q_lcdinit()?
2. How many types of LCDs are there? Which LCD we are using here?

Experiment No. 5 Date-

Aim: Embedded C program for interfacing Keypad & LCD using ARM

Objective:

This program displays key typed on keypad to LCD using ARM.

Requirements:

9. OASIS ARM7(TITAN) BOARD
10. RS232 Serial cable
11. 9 Volt DC Power Supply
12. Triton IDE

Pin Assignment: LCD display interface

Sr. No.	Signal	Description
1	Pin 16 (P1.16)	Data 0
2	Pin 12 (P1.17)	Data 1
3	Pin 8 (P1.18)	Data 2
4	Pin 4 (P1.19)	Data 3
5	Pin 48 (P1.20)	Data 4
6	Pin 44 (P1.21)	Data 5
7	Pin 40 (P1.22)	Data 6
8	Pin 36 (P1.23)	Data 7
9	Pin 13 (P0.28)	RS
10	Pin 14 (P0.29)	EN
11	GND	WR

Pin Assignment: Keypad interface

Sr. No.	Signal	Description
1	Pin 22 (P0.2)	ROW 1
2	Pin 26 (P0.3)	ROW 2

3	Pin 27 (P0.4)	ROW 3
4	Pin 29 (P0.5)	ROW 4
5	Pin 30 (P0.6)	COL 1
6	Pin 31 (P0.7)	COL 2
7	Pin 33 (P0.8)	COL 3
8	Pin 34 (P0.9)	COL 4

Theory:

This program demonstrates LCD display using keypad interfacing with ARM7 through GPIO. This will shows the interfacing of 4*4 keypad with ARM7, keypad such as those found on telephone and calculators are almost always matrix switches. Pressing a button established a connection between its column and row terminals. For example, pressing the 'S' button rows for practical purpose. We use 16 key model for reading the keypad. We can determine the row number and column number, and from that identify which button has been pressed.

LCD: For this experiment we use 16*2 LCD display which shows the number pressed on keypad.

Algorithm:

Program:

```
#include <LPC21XX.h>

#include <board.h>

int main(void)

{

        char key;

        q_keyinit(TITAN);
```

```
        q_lcdinit(TITAN);
        while(1)
        {
                while((key = q_keyread()) == 0);
                key = key + 0x30;
                q_clrscreen();
                q_printf("%x \r\n", key);
                q_displaylcd(&key, 1);
        }
}
```

Application of Experiment:

1. Door-lock control
2. Safe box
3. Simple access controller
4. Vehicle control
5. ATM

Procedure:

13. Connect 9 V DC Power supply to the OASIS TITAN Board.
14. Connect the Board with the COM port of the PC using the serial cable.
15. Generate .hex file using Triton IDE.
16. Download the .hex file.
17. Put the board in RUN mode.
18. Press any key on on-board Keypad and you can observe the output on LCD.

Result:

Conclusion:

Oral questions:

1. Why we use q_keyinit()?
2. Why we use q_lcdinit()?
3. How many types of LCDs are there? Which LCD we are using here?
4. Which keypads are available? What keypad we are using?

Experiment No. 6 Date-

Aim: Embedded C program for interface of Stepper Motor using ARM.

Objective:

This program demonstrates interfacing of Stepper Motor with ARM and rotates it clockwise and anticlockwise.

Requirements:

1. OASIS ARM7(TITAN) BOARD
2. RS232 Serial cable
3. 9 Volt DC Power Supply
4. Triton IDE

Pin Assignment:

Sr. No.	Signal	Description
1	Pin 37 (P0.11)	Data 0
2	Pin 38 (P0.12)	Data 1
3	Pin 39 (P0.13)	Data 2
4	Pin 41 (P0.14)	Data 3
5	5.0V	VCC(5)
6	GND	GND

Theory:

A stepper motor is widely used device that translates electrical pulses into mechanical movement. Stepper motor commonly have a permanent magnet rotor (also called shaft) surrounded by stator. Stepper motor also called variable reluctance stepper motor that do not have PM rotor. The most common stepper motor has a four stator winding that are paired with center tapped. This stepper motor is known as four phase or unipolar stepper motor. The centre tapped allows a change of current direction in each two coils when a winding is grounded. Therefore resulting in polarity change of the stator. Stepper motor shaft moves in a fixed repeatable increment, which allows one to move it to a precise position. The simple motor model at a centre is a bar magnet to rotate surround by a 4 electron magnet space 90 degree to each other. The electromagnets are wound over soft iron poles in motor terminology. The bar magnet is rotor and the surrounding electromagnet from the stator.

The motor is unpowered and the rotor has automatically rotated to the position of minimum magnetic energy i.e., the permanent magnets rotates position itself so its fault has the shortest air path and longest iron path.

Now let's energies winding A. We will set the priority of the current through 'A' so that the inward falling pole is attracting the motor North Pole. & the rotor is held in place the external towards necessary to over side the magnetic attraction & move the rotor is known as winding torque. Similarly we apply current to winding A, B, C, D in that order with the polarity so that the order with inside of its winding is a magnetic south pole. We may keep this up by energizing then A,B,C,D so long as we designed the rotor to continue stepping clockwise.

Algorithm:

Program:

```
#include <LPC21xx.h>

#include<board.h>

int main(void)

{

        unsigned long int i,j;

        *PINSEL0 = *PINSEL0 &(0XC03FFFFF);

        *IODIR0 = *IODIR0 | (0X00007800);

        q_lcdinit(TITAN);


        while(1)

        {

                q_displaylcd("clkwise directxn",16);

                for(j=0;j<12;j++)
```

```
        {
                *IOSET0 = 0x00002800;
                for(i=0;i<25000;i++);
                *IOCLR0 = 0x00002800;
                for(i=0;i<25000;i++);
                *IOSET0 = 0x00003000;
                for(i=0;i<25000;i++);
                *IOCLR0 = 0x00003000;
                for(i=0;i<25000;i++);
                *IOSET0 = 0x00005000;
                for(i=0;i<25000;i++);
                *IOCLR0 = 0x00005000;
                for(i=0;i<25000;i++);
                *IOSET0 = 0x00004800;
                for(i=0;i<25000;i++);
                *IOCLR0 = 0x00004800;
                for(i=0;i<25000;i++);
}
        q_clrscreen();
        q_displaylcd("anticlk directxn",16);

        for(j=0;j<12;j++)
        {
                *IOSET0 = 0x00002800;
```

```
                    for(i=0;i<25000;i++);
                    *IOCLR0 = 0x00002800;
                    for(i=0;i<25000;i++);
                    *IOSET0 = 0x00004800;
                    for(i=0;i<25000;i++);
                    *IOCLR0 = 0x00004800;
                    for(i=0;i<25000;i++);
                    *IOSET0 = 0x00005000;
                    for(i=0;i<25000;i++);
                    *IOCLR0 = 0x00005000;
                    for(i=0;i<25000;i++);
                    *IOSET0 = 0x00003000;
                    for(i=0;i<25000;i++);
                    *IOCLR0 = 0x00003000;
                    for(i=0;i<25000;i++);
                }
                q_clrscreen();
            }
                return 0;
}
```

Application of Experiment:

1. Disk drives
2. Dot matrix printers
3. Robotics

4. In position control.

Procedure:

1. Connect 9 V DC Power supply to the OASIS TITAN Board.
2. Connect the Board with the COM port of the PC using the serial cable.
3. Generate .hex file using Triton IDE.
4. Download the .hex file.
5. Connect the stepper Motor as per the Pin diagram given in user manual.
6. Now set the board into RUN mode and observe the output.

Result:

Conclusion:

Oral questions:

1. What is q_lcdinit?
2. What is PINSEL?
3. What is the relation between steps per second and RPM?
4. What do you mean by step sequences for stepper motor?

Experiment No. 7 Date-

Aim: To write a program for interfacing on-board Keypad & LCD, using uC/OS-II.

Objective:

Program to display a key pressed on Keypad to LCD.

Requirements:

1. OASIS ARM7(TITAN) BOARD
2. RS232 Serial cable
3. 9 Volt DC Power Supply
4. Triton IDE

Theory:

KERNEL: It is the heart of an operating system. Part of multi-tasking system, responsible for task management & communication between tasks. It adds overhead to system as services provided by its required execution time. It requires extra ROM & additional RAM for kernel data structure. Scheduler: Scheduler is also called the dispatcher & is part of kernel, responsible for determining, which task runs most RT-kernels are priority based on its importance.

- **Non-preemptive Kernel:**

1) Each task explicitly give up control of CPU.
2) Co-operative multi-tasking to share CPU.

- **Preemptive Kernel:**

1) HPT ready to run is always given the control of cpu.
2) When task makes HPT ready to run, current task is preempted.

Scheduler: Scheduler is also called the dispatcher & is part of kernel, responsible for determining, which task runs most RT-kernels are priority based on its importance. Two types of priority based kernel exist:

1) Non-preemptive
2) Preemptive.

This program demonstrates LCD display using keypad interfacing with ARM7 through GPIO. This will shows the interfacing of 4*4 keypad with ARM7, keypad such as those found on telephone and calculators are almost always matrix switches. Pressing a button established a connection between its column and row terminals. For example, pressing the 'S' button rows for practical purpose. We use 16

key model for reading the keypad. We can determine the row number and column number, and from that identify which button has been pressed.

LCD: For this experiment we use 16*2 LCD display which shows the number pressed on keypad.

Algorithm:

1. Initialize the stack space for tasks
2. Assign the priority to each task.
3. Initialize the LCD.
4. Scan the keyboard for pressed key.
5. Display the pressed Key on LCD.

Program:

```
#include<board.h>

#include<ucos.h>

#include<lcd.h>

#include<keyboard.h>

void init_timer();

OS_STK   TaskStk[2][100];

OS_EVENT *MBox;

void t1()

{

        char err,*string;

        while(1)

        {

                OSTimeDly(5);

                string=OSMboxPend(MBox,0,&err);

                q_displaylcd(string,1);

        }
```

```
}
void t2()
{
        char key;
        while(1)
        {
                OSTimeDly(5);
                while((key=q_keyread())==0);
                q_clrscreen();
                key = key+0x30;
                OSMboxPost(MBox,&key);
        }
}
int main(void)
{
        init_timer();
        q_keyinit(TITAN);
        q_lcdinit(TITAN);
        q_inccursor();
        OSInit();
        MBox=OSMboxCreate((void *)0);
        TaskCreate(t1,0,"High",0);
        TaskCreate(t2,0,"Low",1);
        OSStart();
```

```
    return 0;
}
```

Application of Experiment:

1. We can assign priority to tasks by using keypad and observe the result on LCD after execution of tasks routines.
2. Door-lock control
3. Safe box
4. Simple access controller
5. Vehicle control
6. ATM

Procedure:

1. Connect 9 V DC Power supply to the OASIS TITAN Board.
2. Connect the Board with the COM port of the PC using the serial cable.
3. Generate .hex file using Triton IDE.
4. Download the .hex file.
5. Put the board in RUN mode.
6. Press any key on on-board Keypad and you can observe the output on LCD.

Result:

Conclusion:

Oral questions:

1. How many types of LCDs are there? Which LCD we are using here?
2. Which keypads are available? What keypad we are using?
3. Which header files we need to include for this program?
4. What is the purpose of OSMboxPend() & OSMboxPost()?

Experiment No. 8 Date-

Aim: To implement Context Switching using uC/OS-II

Objective:

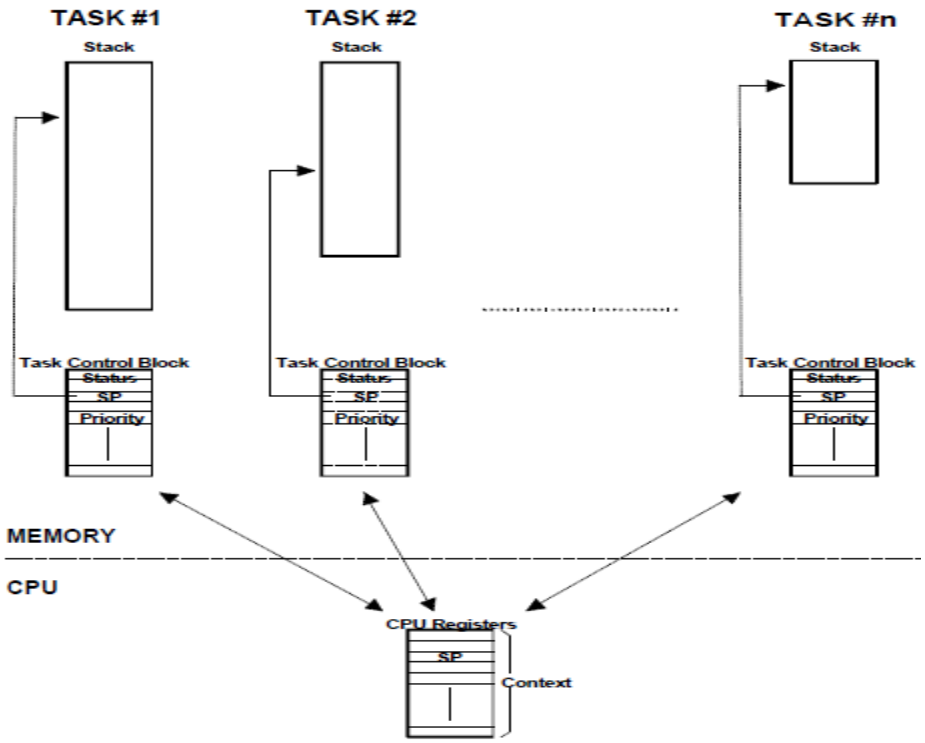
This program demonstrates Context Switching between tasks using RTOS.

Requirements:

1. OASIS ARM7(TITAN) BOARD
2. RS232 Serial cable
3. 9 Volt DC Power Supply
4. Triton IDE

Theory:

The kernel is the part of a multitasking system responsible for the management of tasks (that is, for managing the CPU's time) and communication between tasks. The fundamental service provided by the kernel is context switching. When a multitasking kernel decides to run a different task, it simply saves the current task's *context* (CPU registers) in the current task's context storage area – it's stack. See Figure below.



Once this operation is performed, the new task's context is restored from its storage area and then resumes execution of the new task's code. This process is called a *context switch* or a *task switch*. Context switching adds overhead to the application. The more registers a CPU has, the higher the

overhead. The time required to perform a context switch is determined by how many registers have to be saved and restored by the CPU. Performance of a real-time kernel should not be judged on how many context switches the kernel is capable of doing per second. The actual mechanism for swapping tasks is called a *context switch*. P*reemptive* RTOS context switching occurs periodically when a timer interrupt is raised.

Algorithm:

Program:

```
#include<ucos.h>

#include<board.h>

#define  TASK_STK_SIZE  100

#define  NO_TASKS    3

void init_timer();

OS_STK   TaskStk[NO_TASKS][TASK_STK_SIZE];

//OS_STK   TaskStartStk[TASK_STK_SIZE];

void Task1()

{

                    while(1)
```

```
        {
            q_printf("high priority task \n");
            OSTimeDly(5);
        }
}
void Task2()
{
    while(1)
    {
        q_printf("low priority task\n");
        OSTimeDly(10);
    }
}
int main(void)
{
    init_timer();
    OSInit();
    TaskCreate(Task1,(void*)0,"Hi",1);
    TaskCreate(Task2,(void*)0,"Low",2);
    OSStart();
    return 0;
}
```

Application of Experiment:

1. To classify computing tasks in terms of their timing constraints.

2. Scheduling as per priority
3. In controlling domestic and consumer appliances depending upon the service requirements

Procedure:

1. Connect 9 V DC Power supply to the OASIS TITAN Board.
2. Connect the Board with the COM port of the PC using the serial cable.
3. Generate .hex file using Triton IDE.
4. Download the .hex file.
5. Put the board in RUN mode and you can observe the LED output.

Result:

Conclusion:

Oral questions:

1. What do you mean by context switching?
2. How context switching is used in Inter task communication?
3. What is the purpose of stack in context switching?
4. When preemptive RTOS context switching occurs?
5. Whether context switching adds overhead to the application?

Experiment No. 9 Date-

Aim: Implementation of Task Synchronization using semaphore for any given task using RTOS.

Objective:

To describe the semaphore behavior between two different tasks for task synchronization and task scheduling.

Requirements:

It includes component, equipment, software, hardware, Kits, PCs required to conduct experiments.

Theory:

A Semaphore is an integer variable, which is used for synchronization. Value of semaphore can be modified by two uninterruptable operations, P & V semaphore variable. P stands for "WAIT"->Decrement block until semaphore is free. V stands for "POST"-> Increment allows another to enter.

Types of Semaphore:

1) Binary Semaphore:

- Integer value can range only between 0 & 1 simpler to implement
- Only one thread / process allowed entry at a time.
- Counter is initialized to '1'.

2) Counting Semaphore:

- Integer value can range over unrestricted domain.
- Allows threads/process to enter as long as units are available.
- Counter is initialized to '1'.
- Can be implemented as Binary.

Configuration Constants:

Sr No.	Semaphore Services	Enabled when set to '1' OS_CFG.h
1	OSSEMACCEPT()	OS_SEM_ACCEPT_EN
2	OSSEMCREAT()	
3	OSSEMDEL()	OS_SEM_DEL_EN
4	OSSEMPEND()	
5	OSSEMPOST()	
6	OSSEMQUERY()	OS_SEM_QUERY_EN

Algorithm:

Program:

```
#include<board.h>
#include<ucos.h>
#include<LPC21xx.h>
OS_STK TaskStk[2][100];
void inituart();
void init_timer();
OS_EVENT*sem;
char gvar;
void Txtask();
void Rectask();
int main(void)
{
        init_timer();
        inituart();
        OSInit();
        sem=OSSemCreate(0);
        TaskCreate(Txtask,0,"Hiprio",0);
        TaskCreate(Rectask,0,"Loprio",1);
        OSStart();
```

```
        return 0;
}
void Rectask()
{
        while(1)
        {
                while(!(*U0LSR&0x1));
                gvar = *U0RBR;
                OSSemPost(sem);
        }
}
void Txtask()
{
        char err;
        while(1)
        {
                OSSemPend(sem,0,&err);
                while(!(*U0LSR&0x20));
                *U0THR = gvar;
        }
}
void inituart()
{
        *PINSEL0 |=0x5;
```

```
    *U0LCR=0x80;

    *U0DLL=0x61;

    *U0DLM=0;

    *U0LCR=0x3;

}
```

Application of Experiment:

6. In parallel computing for HPC (High Performance Computing)
7. To create a lock for synchronizing the access of data
8. Used in Pipe, Mailboxes and Message Queues to handle messages

Procedure:

1. Connect 9 V DC Power supply to the OASIS TITAN Board.
2. Connect the Board with the COM port of the PC using the serial cable.
3. Generate .hex file using Triton IDE.
4. Download the .hex file.
5. Put the board in RUN mode and you can observe the LED output.

Result:

Conclusion:

Oral questions:

1. What is Semaphore and its types?
2. Why we use OS_EVENT*sem?
3. Why we use void inituart()?
4. Why we use P, V semaphore?
5. What is Scheduler?
6. What is Kernel?
7. Why we use OS_STK TaskStk?