Sanjivani Rural Education Society's

SANJIVANI COLLEGE OF ENGINEERING,

KOPARGAON - 423 603

(An Autonomous Institute)

Department ECE Engineering

PRACTICAL / TERM-WORK REPORT

Subject ES & RTOS

Name of the Student Wakte Sakshi Vijay

Class TY ECE Division -

Batch No. T3 Roll No. 73

Name of the Experiment / Title of Term Work WiFi Module ESP8266 Interfacing & Display the Data on ThingSpeak IOT Platform

Experiment / Term - Work No.

Conducted on :

Expected date of Submission :

Actual date of Submission :

Checked by Prof. Y R. Khandekar

(Name of the Teacher)

Remarks 09/10

Signature Date

Page No.
Date

Experiment No-

WiFi Module ESP8266 Interfacing and Display the Data on ThingSpeak IOT Platform.

Aim :

Interfacing of Wi-Fi (ESP8266) Module & Sensor.

Learning Objectives :

To understand interfacing of analog as well as digital input & output devices with the NodeMCU board.

Equipments Required :

ESP8266
LM35
Breadboard
Connecting Wires.

Theory :

What is LM35?

LM35 is a temperature sensor that outputs an analog signal which is proportional to the instantaneous temperature.
The output voltage can easily be interpreted to obtain a temperature reading in Celsius.
The advantage of m35 over thermistor is it does not required any external calibration.

VCC: supply voltage
OUT: It gives analog output voltage which is proportional to temperature
GND: Ground.

What is esp8266?

→ An ESP8266 Wifi module is a SOC microchip mainly used for development of end-point IOT application. It is referred is as a stand-alone wireless transreceiver, available at low price. It is used to enable internet connection to various application or embedded system.

There are different types of ESP modules designed They are,

- ESP8260-01 designed with 8 pins (GPIO pin 2)
- ESP8266-02 designed with 8 pins (GPIO pins-3)
- ESP8266-03 designed with 14 pins (GPIO pins-7)
- ESP8266-04 designed with 14 pins (GPIO pins-7)

Specification & construction:

- Operating voltage : 2.5 to 3.3V
- Operating current : 800mA
- 3.3V 600mA on-board voltage regulation
- ESP8266 comes up with 2 switches one is reset and another one is flash button.
- ESP8266 board also has 12C pins which can be used both as 12C master and 12C slave
- ESP8266 NodeMCU has 17 GPIO pins which can be assigned to various function such as UART, PWM, 12C, IR and button via programming.

Procedure:

A. Connection:

Connect 3.3V pin of ESP8266 to first pin of LM35.

Connect A0 pin of ESP8266 to second pin of LM35.

Connect GND pin of ESP8266 to third pin of LM35

B. Install Arduino IDE latest Version (1.8.19) or (2.2.1):

1. While version 2 works well with Arduino, there are still some bugs & some features that are not supported yet for the ESP8266.

2. Select your operating system & download the software. For windows we recommend downloading the "Windows ZIP file".
3. Grab the folder you've just downloaded & unzip it. Run the executable file called arduino.exe.

C. Installing the ESP8266 Node MCU in Arduino IDE:

1. Go the file > Preferences.
2. Enter the following into the "Additional Board Manager URL's" field https://arduino.esp8266.com/stable/package_esp8266com_index.json.
3. Then, click the "OK" button
4. Open the Board Manager Go to Tools > Board > Board Manager.
5. Search for ESP8266 and install 'ESP8266 by ESP8266 community"
6. That's it. It will be installed after a few seconds.
7. After this, restart your Arduino IDE
8. Then go to Tools > Board & check that you have ESP8266 boards available

9. Select Node MCU 1.0 (ESP-12E Module).
10. Now, you're ready to start programming your ESP8266 using Arduino IDE

D. ThingSpeak Library:
Add ThingSpeak latest version 2.0.1 library from Library Manager.

E Installing the ESP8266 Node MCU USB Drivers:
1. After connecting the ESP8266 board to your computer, if the COM port in Arduino IDE is grayed out, it means you don't have the required USB drivers installed on your computer
2. Most ESP8266 boards either user the CP2101 or CH340 drivers. check the USB to UART converter on your board & install the corresponding drivers.

F Upload the code:
1. Connect your ESP8266 development board to your computer using a USB cable.
2. Go to Tools > Board, scroll down to the ESP8266 section & select your ESP8266 board. If you don't know what's your board, the Generic ESP8266 Module usually works fine for most boards.
3. Go to Tools > Port & select a COM port available, If the COM port is grayed out,

this means you don't have the required USB drivers. Check the section Installing USB Drivers before proceding.

4. Press the upload button
5. The code should be successfully uploaded to the board after a few seconds.

Testing :

This monitors the current room temperature by using LM35 temperature sensors & updates it to Thingspeak IOT server by using ESP8266 WiFi SoC. Output of temperature sensors i.e LM35 is connected to analog input pin A0 of ESP8266.

After uploading the code to ESP8266 module, open the serial monitor and ~~pass~~ press RST pin of ESP8266. The module will connect to Thingspeak server through internet. Now open Thingspeak server and observe the monitored temperature value on created channel graph

Page No.
Date

ThingSpeak is a very good platform for IoT based projects. By using channels & web pages provided by ThingSpeak we can manitor any data over the Internet from anywhere and we can also control our system over the internet. ThingSpeak 'Collects' data from sensors, 'Analyze & Visualize' data & 'Acts' by triggering a reaction. Here we are explaning about how to setup ThingSpeak account for this project.

1. First, we need to create an account on Thingspeak.com, then Sign In and click on 'Get started'.
2. Now go to the 'Channels' ~~means~~ menu and click on the 'New Channel' option in the same page.
3. Now we will see a form for creating the channel. fill the Name and Description As per your choice. Then fill 'Temperature' in 'field 1' field. Tick the check box 'Make Public' option below the form and finally Save the channel. Now your new channel is ready.
4. Now click on 'API keys' tab and note the Write & Read API key, here we are only using Write key. We need to copy & paste this key in the code.
5. Now we need to upload the program to ESP8266 using Arduino IDE

6. After uploading, open "PRIVATE VIEW" icon in ThingSpeak website & observe the monitored temperature value on graph as shown below.

Keywords:
Here are some keywords that are used in the API. Understanding of these terms will make the API documentation easier to understand.

Channel:
Channel can be said as a stream of data It identified by a numerical channel ID using which data can be inserted or retrieved using ThingSpeak APIs

Field:
Each channel is having 8 Field which can hold any type of data. For eg. you may store temperature, humidity, RFID data (alphanumeric) in each channel.

Status:
It is short status message to argument the data stored in a channel.

Location:
In addition to above 8 field, we can store Gps location or coordinates. For eg. we can

stored the location of the place from where the data is coming. It is having latitude, longitude and elevation.

Write API key:
A 16-digit API key code that allows an application to write data to a channel. You should not share this API key publicly. because anyone having this key can write data to your channel.

Read API key:
A 16-digit API key code that allows an application to ready the data stored in a channel. You should not share this API key publicly, because anyone having this key can read data from your channel.

Program:

```
#include "Thingspeak.h"
#include <ESP8266WiFi.h>
 const int LM35=A0;
 //-----Enter you Wifi Details-----//
 char ssid[] = "redmi12";  //SSID
 char pass[] = "xyz@123"; // Password
 //-------------------------//

WiFiClient client;
```

```
unsigned long myChannelNumber = 2587047;
                                //Channel ID here.
const int fieldNumber = 1;
const char * myWriteAPIkey = "TBkD4NSHVU-
                    2MZ4QR"; // Your Write
                              API Key here.

void setup() {
  Serial.begin (115200);
  WiFi.mode (WIFI_STA);
  ThingSpeak.begin (client);
}

void loop() {
if (WiFi.status () != WL_CONNECTED) {
Serial.print ("Attempting to connect to
    SSID:");
Serial.println (ssid);
while (WiFi.status () != WL_CONNECTED) {
WiFi.begin (ssid, pass);
Serial.print (".");
delay (5000);
}

Serial.println ("\n connected.");
}
int ADC;
float temp;
ADC = analogRead (LM35); /* Read Temperature */
temp = (ADC * 3);  /* Convert adc value to
                      equivalent voltage */
```

er = 2587047;
nel ID here

```
temp = (temp /10);  /* LM35 gives output
                       of 10 mV/°C */
Serial.print("Temperature = ");
Serial.print(temp);
Serial.println("*C");
delay(1000);
ThingSpeak.writeField(myChannelNumber,
  FieldNumber, temp, myWriteAPIkey);
delay(1000);
}
```

Sanjivani Rural Education Society's

SANJIVANI COLLEGE OF ENGINEERING,

KOPARGAON - 423 603

(An Autonomous Institute)

Department ECE. Engineering

PRACTICAL / TERM-WORK REPORT

Subject ES & RTOS.

Name of the Student Wakte Sakshi Vijay

Class TY ECE Division -

Batch No. T3 Roll No. 73.

Name of the Experiment / Title of Term Work Simulation of Multitasking with µCOS-II on ARM7 microcontroller for three tasks to blink 3 LEDS with 3 different rates.

Experiment / Term - Work No.

Conducted on :

Expected date of Submission :

Actual date of Submission :

Checked by Prof. Y. R. Khandekar.

(Name of the Teacher)

Remarks

Signature Date

Page No.
Date

Title : Simulation of Multitasking with uCOS-II on ARM7 microcontroller for three tasks to blink 3 LEDS with 3 different rates.

Aim : Write a program for implementation of multitasking in uCOS II RTOS for [Any one of following]

1. 3 tasks, each will blink LED with different rates.
2. 3 tasks, each will send a message on hyper terminal with different delays.

Learning Objectives :

To understand

- Creating tasks under uCOS II RTOS.
- How to write a program using uCOS II RTOS to perform multiple tasks with different priorities.
- How to create, execute and debug project in keil development enviroment.

Equipments required :

Trainer kit of LPC2148, serial cable, power supply, LCD-KEYBOARD interfacing card, LED interfacing card [GPIO module] PC with WinARM installed in it etc.

Theory:

Explain µCOS II RTOS API function like OSInit(), OSTaskCreate(), OSStart(), OSTimeDlyHMSM()

→ 1. OSInit():
µCOS-II requires OSInit() to be called before any other services. This function initializes the variable and data structure of µclas II OSInit() creates and idle task OS_TaskIdle() which is ready to run.

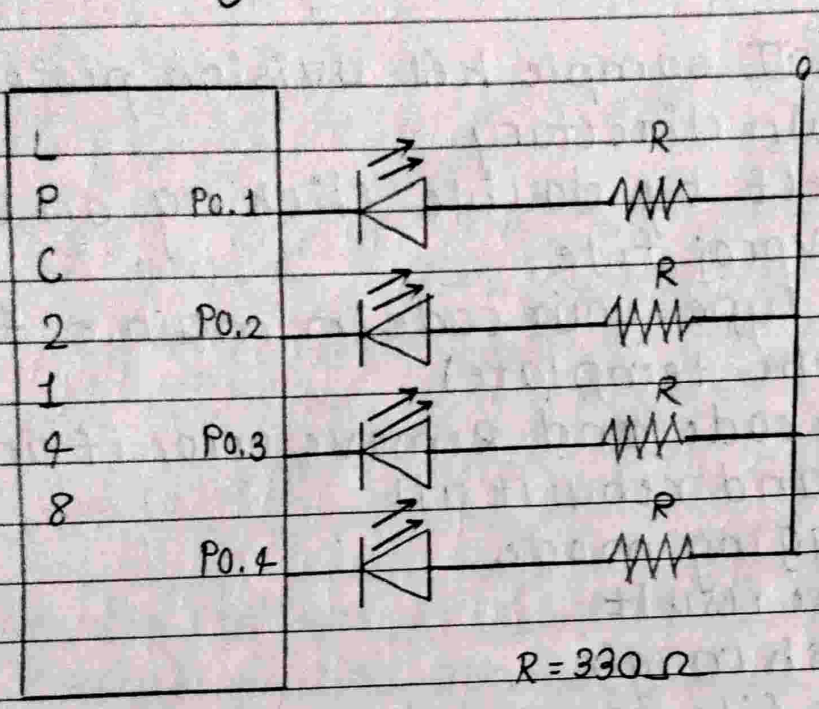
2. OSTaskcreate():
It creates a task to be managed by µcos II Task can be created before that start of multitasking or by a running task. A task cannot be created in an ISR. A talk always contains a finite loop which doesn't return a value. Dependening on nature of stack frame the task can have interrupt.

3. OSstart():
OSstart begins the multitasking. Call this as a part of your intialization fun^ but make sure that you have called OSInit()

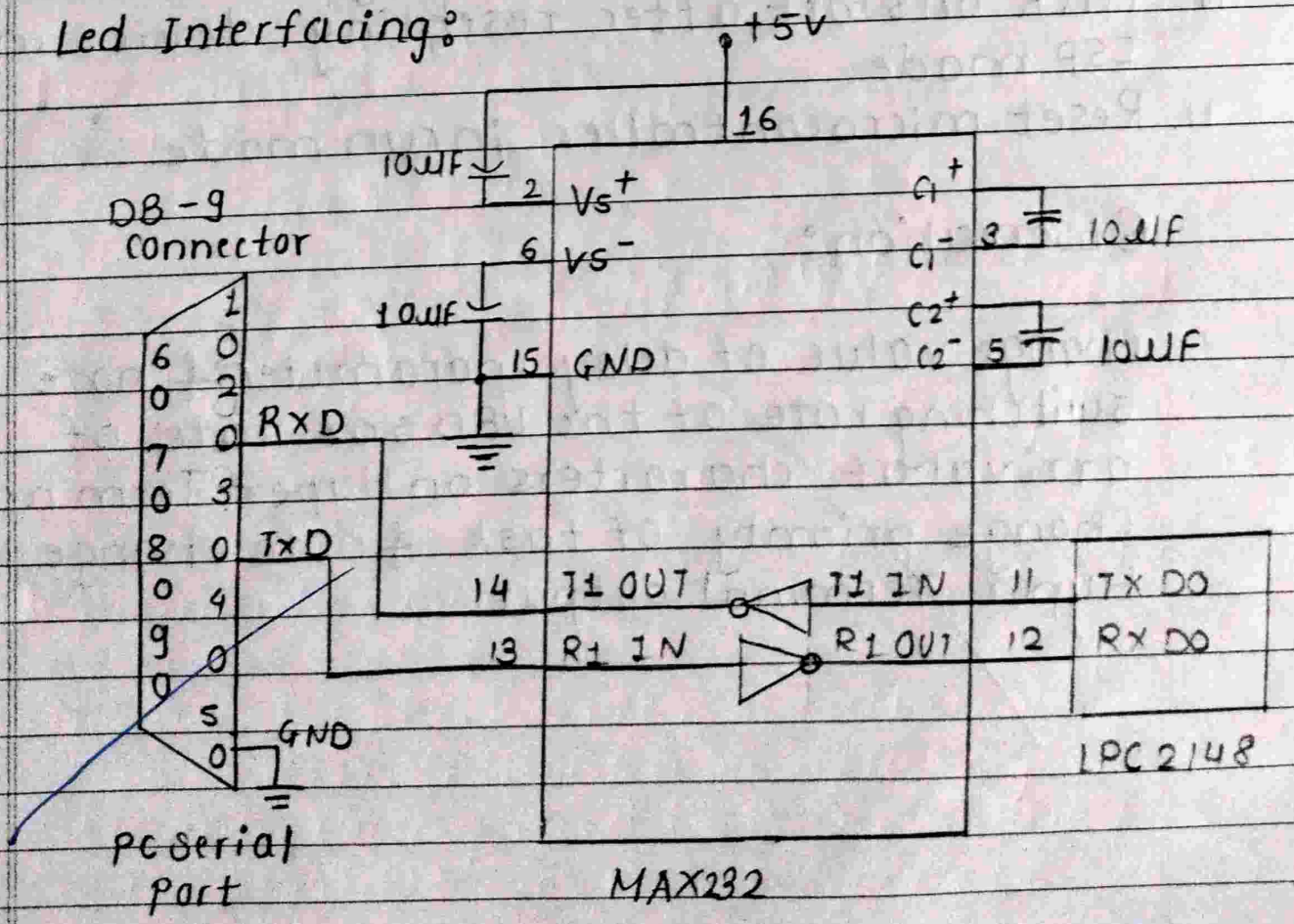
4. OSTimeDelyHMSM():
The HMSM Stands for hours, Minutes, seconds, and Milisecond.
- It is time function
It is delays execution of task for the specified number of clock tick. No delay will result of ticks is '0'. If ticks is >0 then a context switch will result.

Page No.
Date

Interfacing Diagram.



Led Interfacing:



Procedure:

1. Copy ucos II sample keil uVision project to your own directory
2. Open project by double clicking on LPC214x _ucosii.uvproj file.
3. Open and type your code in main.c file (fill in the template)
4. Compile a code and Remove error if any
5. Built all and rebuilt all
6. Start debugging mode
7. verify the result
8. Open flash magic
9. Select hex file generated in your own directory
10. Click on start after resebting device in ISP mode.
11. Reset microcontroller in run mode

Observation:

1. Change value of delay parameter & note switching rate of the LED's or rate of arrival of characters on Hyper Terminal
2. Change priority of task & note change in Hyper Terminal output.

Program:

```
#include "config.h"
#include "stdlib.h"
#include "serial.h"

/* Each task must have its own stack spoce. A
 stack must be declared as being of type OS_STK
 and must consist of contiguous memory
 locations.*/

OS_STK t1[100], t2[100], t3[100];
// Each task has stack entries of this type
The data structure is located in file src/
ucos_cpu.h.

 void task1()
 {
  while(1)
    {
     IOOSET = 1<<5;
     OSTimeDlyHMSM(0,0,1,0);
     IOOCLR = 1<<5;
   OSTimeDlyHMSM(0,0,1,0); // allows a task
 to delay itself for a user specified amount
 of second and miliseconds.
 }
}

 voidtask2()
  {
```

```
while(1)
{
 IOOSET = 1<<6;
 OSTimeDly HMSM (0,0,2,0);
 IOOCLR = 1<<6;
 OSTimeDlyHMSM (0,0,2,0);
}
}
Void task3()
{
 while(1)
 {
  IOOSET = 1<<7;  //LED ON
  OSTimeDlyHMSM (0,0,3,0);
  IOOCLR = 1<<7;  //LED OFF
  OSTimeDlyHMSM (0,0,3,0);
}
}
int main()
{
 PINSEL0 = 0X00000000; // configure PORT1 as GPIO
 IOODIR = 0XFFFFFFFF;
 TargetInit();
 //init_serial();
 OSInit(); // call this fun^n first, initialize u c/osII
 &must be called before calling OSStart()
 OSTask Create (task1, NULL, &t1[99], 1); //Ready
 State, Stack grows from High to low memory
 OSTaskcreate (task2, Null, &t2[99], 2);
 OSTask Create (task3, Null, &t3[99], 3);
 OSstart(); // Running
}
```

Sanjivani Rural Education Society's

SANJIVANI COLLEGE OF ENGINEERING,

KOPARGAON - 423 603

(An Autonomous Institute)

Department ECE. Engineering

PRACTICAL / TERM-WORK REPORT

Subject ES & RTOS.

Name of the Student Wakte Sakshi Vijay

Class TY ECE Division -

Batch No. T3 Roll No. 73

Name of the Experiment / Title of Term Work Implementation of semaphore service for signaling & synchronization application with COS II on ARM7 controller.

Experiment / Term - Work No.

Conducted on :

Expected date of Submission :

Actual date of Submission :

Checked by Prof. Y.R. khandekar.

(Name of the Teacher)

Remarks

Signature Date

Page No
Date

Title : Implementation of semaphore service for signaling and synchronization application with Cos-II on ARM7 controller.

Aim :
1. Write a program for implementation of semaphore as a signaling device for two tasks. First will send 'A' & second will send 'B' continuously to HyperTerminal.
2. Write a program for implementation of semaphore as a synchronization device between 2 tasks. First will send 'A' & second will send 'B' continuously to HyperTerminal.
3. Write a program for implementation of semaphore as a signaling device between 2 tasks. First will generate 'A' to 'z' characters and second task will send the same to HyperTerminal.

Learning Objectives :
To understand
- Configuration of semaphore in two different modes.
- Solution to shared data problem.

Equipments Requireds :

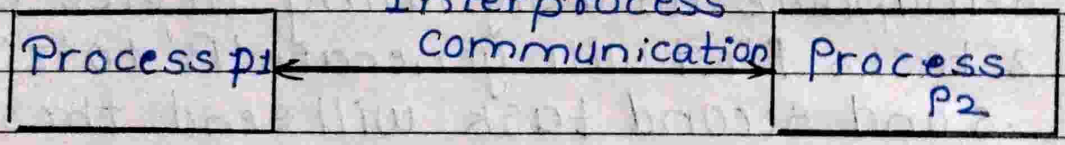
Trainer kit of LPC2148, serial cable, power supply, PC with WinARM or keil uvision ARM

installed init etc.

Theory :

Explain Inter process / task communication (IPC):

Interprocess Communication is the mechanism provided by the operating system that allows processes to communicate with each other. This communication could involve a process letting another process know that same event has occured or the transfering of data from one process to another.



Inter process communication is used for exchanging useful information beth numerous thread in one or more processes or program.

Semaphores & its modes in brief only.

- A Semaphore (sometime called a semaphore token) is a kernel object that one or more thread of execution can acquire or release for the purpose of synchronization or mutual exclusion.
- When a semphore is first created, the kernel assign to it an associated semaphore contral

block (SCB), a unique ID, a value (binary or count), and task-waiting list.

- A Semaphore is like a key that allows a task to carry out some operation to access a resource.
- Types of Semaphore:
A kernel can support many different types of semaphore, including
1. Binary 2. Counting 3. Mutux Semaphore.

1. Binary Semaphore:
- A Binary Semaphore can have a value of either 0 or 1.
- When a binary semaphore's value is 0, the semphore is considered unavailable (or empty) when the value is 1, the binary semaphore is considered available. (or full).

2 Counting Semaphore:
- A counting semaphore uses a count to allow it to be acquired or released multiple times.
- As with binary semaphore, counting semaphore are global resource that can be shared by all tasks need them.

3. Mutual Exclusion (Mutex):
- A Mutual exclusion (mutex) semaphore, is a special binary semaphore that support ownership, recursive access, task deletion safety, and one or more protocol for avoiding problem inherent to mutual exclusion.

Page No.
Date

RTOS configuration parameter.

RTOS (Real-Time Operating System) configuration parameter are typically defined in an operating system (OS) configuration file, often named 'os_cfg.h' or free RTOS config.h', depending on the RTOS being used These files allows customization of various aspects of the RTOS behavior, such as memory allocation, task scheduling, interupt priorities.

UCOS II RTOS API function in brief

a. OSSemCreate() b. OSSemPend()
c. OSSemPost() d. OSSemAccept()

→ OSSemCreate():

- This function is used to create a binary semaphore.
- Parameter typically include an address to store the semaphore control block, an initial semaphore value (0 or 1) and an error pointer.
- Example: OSSemCreate(OS_EVENT *pevent, INT16U value, INT8U *perr);

OSSemPend(): This function is used to wait (or pend) on a semaphore.

- It decrement the semaphore count, blocking the task if the semaphore count is zero.
- Example: OSSemPend(OS_EVENT *pevent, INT32U timeout, INT8U

OSSemPost():

- This function is used to signal (or post) a semaphore.
- It increments the semaphore count potentially unblock task waiting on the semaphore.
- Example: OSSemPost(OS_EVENT *pevent, INT8U * perr);

OSSemAccept():

- This function is used to check the current value of semaphore without blocking
- It return the current semaphore count without changing it.
- Example: OSSemAccept(OS_EVENT *pevent);

Page No.
Date

Procedure:

1. Copy ucos II sample keil uvision project to your own directory.
2. Open project by double clicking on LPC214x_ucosii.uvproj file.
3. Open and Type your code in main.c file
4. Compile a code and Remove error if any
5. Built all and rebuilt all
6. Start debugging mode
7. Verify the result
8. Open flash magic
9. Select hex file generated in your own directory
10. Click on start after resetting device in ISP mode.
11. Reset microcontroller in run mode.

Observation:

1. Change value of delay parameter & note switching rate of the LEDs or rate of arrival of characters on HyperTerminal.
2. Change priority of task & note change in HyperTerminal output.

Page No.
Date

Program:

```
#include "config.h"
#include "stdlib.h"
#include "serial.h"
OS_STK t1[100], t2[100];
OS_EVENT *sem;
INT8U err;
 void task1()
{
  while (1)
   {
    OSSemPend (sem, 0, &err);
    printf ("A");
    OSSemPost( sem);

  OSTimeDlyHMSM (0,0,0,100);
   }
}
  void task2()
{
  while (1)
   {
    OSSemPend (sem, 0, &err);
    printf ("B");
    OSSemPost (sem);
   OSTimeDlyHMSM (0,0,0,200);
   }
}
```

```
int main()
{
    TargetInit();
    init_serial();
    OSInit();
    sem = OSSemCreate(1);

OSTaskCreate(task1, NULL, &t1[99], 2);
OSTaskCreate(task2, NULL, &t2[99], 1);

    OSStart();
}
```

Sanjivani Rural Education Society's

SANJIVANI COLLEGE OF ENGINEERING,

KOPARGAON - 423 603

(An Autonomous Institute)

Department ECE Engineering

PRACTICAL / TERM-WORK REPORT

Subject ES & RTOS.

Name of the Student Wakte Sakshi Vijay

Class TY ECE Division

Batch No. T3 Roll No. 73

Name of the Experiment / Title of Term Work Queue implementation for message passing on LPC2148.

Experiment / Term - Work No.

Conducted on :

Expected date of Submission :

Actual date of Submission :

Checked by Prof. Y. R. Khandekar.
(Name of the Teacher)

Remarks

Signature Date

Page No.
Date

Title : Queue implementation for message passing on LPC2148.

Aim: Write a program to send a 10 digit message from one task to another using message queue if queues is not full & display the result on HyperTeximinal if queue is not empty.

Learning Objectives:
To understand.
- Message Queue configuration.

Equipments Required:
Trainer kit of LPC2148, serial cable, power supply, PC with WinARM / keil uvision ARM installed in it etc.

Theory :
1. Message Queue Concept.
- A Message Queue (or queue) is kernel Object (ie data structure)
- allow a task or an ISR to send pointer sized variable to another task. (Unlike in Semaphore, instead of keys the task acquire pointers to message.)
- The Message Queue is an Interprocess Communication Mechanism

Different Message Queue API in brief only.

1. OSQCreate():
This function is used to create a new message queue. It typically takes parameter such as the maximum no. of message the queue can hold and the size of each message.

2. OSQPend():
This Function is used by a task to wait for a msg to be available in the queue. It block the task until a msg is recieved or a tetimeaut occur.

3. OSQPost():
This function is used to send or message to the back of the queue. if there are tasks waiting for message, it unblock the highest priority waiting task.

4. OSQPostfront():
This function is similar to OSQPost(), but it adds the msg to front of queue instead of back.

5. OSQAccept():
This function check if there are msg available in queue without blocking the task.

6. OSQFlush():
This function is used to flush all message from the queue. this can be useful for cleaning up the queue or resetting its state.

Page No.
Date

Procedure:

1. Copy uCos II sample keil uVision project to your own directory
2. Open project by double clicking on LPC214x_uCosii.uvproj file
3. Open and Type your code in main.c file (fill in the template).
4. Compile a code and Remove errors if any
5. Built all and rebuilt all
6. Start debugging mode.
7. Verify the result.
8. Open flash magic
9. Select hex file generated in your own directory
10. Click on start after resetting device in ISP mode.
11. Reset microcontroller in run mode.

Observation:

- Observe effect of change in priority and delay on o/p.
- Try different character as a message in mailbox.

Page No.
Date

Program:

```
#include "config.h"
#include "stdlib.h"
#include "serial.h"
OS_STK t1[100], t2[100];
OS_EVENT *msgq;
INT8U i, err, dat[10] = {1,2,3,4,5,6,7,8,9,
        0};
    void *q[10];
    void task1()
    {
      while(1)
      {
        for(i=0; i<=11; i++)
        {
          err = OSQPost(msgq, (void *)&
          dat[i]);
        if(err = OS_Q_FULL)
        printf("queue is full.");
        else
        printf("sent %d", dat[i]);

    OSTimeDlyHMSM(0,0,0,100);
    }}}

    void task2(){
    INT8U *msg;
      while(1)
    {
```

```
msg = OSQPend (msgq, 10, &err);

if (err = OS_NO_ERR)
printf ("Received = %d", *msg);

else
 printf ("Not received");

OSTimeDlyHMSM (0,0,1,0);
 }
}
int main()
{
  TargetInit();
  init_serial();
   OSInit();
 msgq= OSQCreate (q, 10);

OSTaskCreate (task1, NULL, &t1[99], 1);

OSTaskCreate (task2, NULL, &t2[99], 2);

 OSStart();
}
```

Sanjivani Rural Education Society's

SANJIVANI COLLEGE OF ENGINEERING,

KOPARGAON - 423 603

(An Autonomous Institute)

Department ECE Engineering

PRACTICAL / TERM-WORK REPORT

Subject EG & RTOS.

Name of the Student Wakte Sakshi Vijay

Class TY ECE Division -

Batch No. T3 Roll No. 73

Name of the Experiment / Title of Term Work Mailbox implementation for message passing on Arm 7.

Experiment / Term - Work No.

Conducted on :

Expected date of Submission :

Actual date of Submission :

Checked by Prof. Y. R. Khandekar.

(Name of the Teacher)

Remarks

09/10 Syntax for call function

Signature Date

Page No.
Date

Title : Mailbox implementation for message passing on Arm 7.

Aim : Write a program to send a digit message from one task to another using Mailbox & display the result on Hyper Terminal.

Learning Objectives :
To understand
- Mailbox Configuration
- Different IPc methods

Equipment Required :
Trainer kit of LPC2148, Serial cable, power supply. Pc with KlinARM or Keil Uvision ARM installed in it etc.

Theory : Explain:
1. UART0 registers [U0RBR, U0THR, U0DLL, U0DLM, U0FGR, U0CLR, U0LSR] figures only.

1. U0RBR : Receive Buffer Register.
This register is used to read incoming data the from the UART Receive buffer. for example if a byte is received over the UART, it will be stored in the U0RBR register, and the microcontroller can then read this data from the register.

2. U0THR : Tranomit Holding Register.
This register is used to write data to be transmitted over the UART. For example, if the microcantraller wants to send, a byte of data over the UART, it can write this data to the U0THR register and the UART will transmit this data.

3. U0DLL : Divisor latch LSB:
This register along with the U0DLM register, is used to set the baud rate. the UART. By writing the appropriate value to the U0DLL and U0DLM register the controller can the baud rate to the desired value.

4. U0DLM : Divisor latch MSB.
This register along with the U0DLL register is used to set the baud rate of the UART. The baud rate is used to determine this rate at which data is trasmitted over the UART.

5. U0FCR : FIFO Contral Register:
This register is used to contral the FIFO (First In first-out) buffer of the UART. The FIFO buffer is used to store data before it is transmitted over the UART or after it is Received The U0FCR Register can be used to enable or disable the FIFO buffer and to set the trigger level at which the buffer

6. U0CLR: Line Control Register:
This register is used to control various aspect of the serial communication such as setting the data format (number of data bits, parity and stop bits and enabling or disabling the divisor latch Access.

7. Mail Box APIs brief only.

- OSMboxCreate():
This Function is used to create a mailbox, which is a data structure that allows one task to send a message to another risk It initialize the mailbox and return a handle to it which can be used in other mailbox API function.

- OSMboxPend():
This function is used by a task to wait for a message to be sent to it through a mailbox if no message is available, the task will block until a message is sent once a message is received, the function returns the message.

OSMboxPost():
This function is used to send a message to a task through a mailbox. It takes the handle of the mailbox and the message to be sent as arguments. if a task is waiting

for a message through the mailbox, it will unblock and receive the message.

- OSMboxAccept():

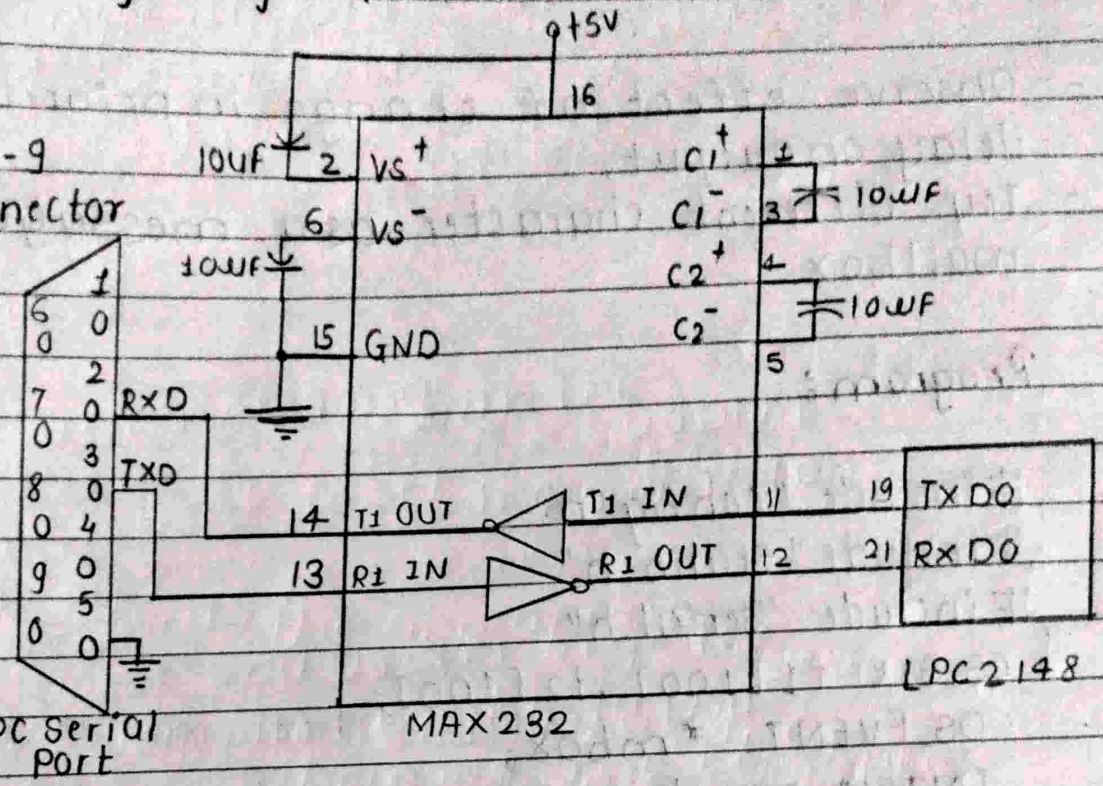
This function is similar to OSMboxPend(), but it does not block if no message is available. If a message is available, it returns the message if no message is available, it returns NULL immediately

- OSMboxQuery():

This function is used to query the state of a mailbox. it can be used to check if a mailbox is empty or full, or to get information about the last task that sent or received a message through the mailbox.

Page No.
Date

Interfacing Diagram:



Procedure:

1. Copy uCOS II sample keil uvision project to your own directory
2. Open project by double clicking on LPC214X _uCosii.uvproj file
3. Open and Type your code in main.c file
4. Compile a code and Remove error if any
5. Built all and rebuilt all
6. Start debugging mode.
7. Verify the result.
8. Open flash magic
9. Select hex file generated in your own directory
10. Click on start after resetting device in ISP mode
11. Reset microcontroller in run mode.

Page No.
Date

Observation:

- Observe effect of change in priority and delay on output.
- Try different character as a message in mailbox.

Program:

```
#include "config.h"
#include "stdlib.h"
#include "serial.h"
OS_STK t1[100], t2[100];
OS_EVENT *mbox;
INT8U err, key = '3';
Void task1()
{
  While(1)
  {
    err = OSMboxPost (mbox, (void *) key);
    printf("S");
  OSTimeDlyHMSM (0,0,0,200);
  } }
  void task2() {
  INT8U *msg;
  while (1) {
    msg = (INT8U *)
    (OSMboxPend (mbox, 10, &err));
    if (err == OS_NO_ERR)
```

```
printf ("Y %.c ", msg);
  else

  printf (" N");

  OSTimeDly HMSM (0,0,0,100);
 }
}
 int main ()
 {

    TargetInit ();
    init_serial ();
    os_Init ();
    mbox = OSMboxCreate (0);

OSTaskCreate (task1, NULL, &t1[99], 1);

OSTaskCreate (task2, NULL, &t2[99], 2);

  OSStart ();
}
```

Sanjivani Rural Education Society's

SANJIVANI COLLEGE OF ENGINEERING,

KOPARGAON - 423 603

(An Autonomous Institute)

Department ECE Engineering

PRACTICAL / TERM-WORK REPORT

Subject ES & RTOS.

Name of the Student Wakte Sakshi Vijay

Class TYECE Division -

Batch No. T3 Roll No. 73.

Name of the Experiment / Title of Term Work Using Device driver for GPIO, write a program to blink LED

Experiment / Term - Work No.

Conducted on :

Expected date of Submission :

Actual date of Submission :

Checked by Prof. Y. R. Khandekar.
(Name of the Teacher)

Remarks

Signature Date

Page No.
Date

Experiment No-

Title : Using Device driver for GPIO, write a program to blink LED.

Aim : Write a device driver for LED, compile kernel module and load it in kernel space.
Write program to blink LED using ARM9 target board, build it, and execute it using the embedded linux.

Learning Objectives :
- To understand device driver writing procedure for hardware like, LED.
- To understand procedure to use device driver of the system.

Equipments required :
Trainer kit of ARM9 controller, serial cable, power supply, CCS server, two Ethernet cables, Ethernet switch.

Theory :
Explain Module Utilities :
insmod :
The insmod command is used to insert modules into the kernel. kernel modules are usually used to add support for new hardware (as device driver) and /or filesystem, or for adding system

calls. This command insert the kernel object file (.ko) into kernel.
Syntax:
insmod [file name] [module-options...]

- lsmod:
The lsmod command is used to display the status of modules in the Linux kernel.
It result in a list of loaded modules. lsmod is a trivial program which nicely formats the contents of the /proc/modules, showing what kernel modules are currently loaded.
syntax:
lsmod.

- rmmod:
The rmmod command is used to remove a module from the kernel. Most of the users use modprobe with the -r option instead of using rmmod.
Syntax:
rmmod [-f] [-s] [-v] [modulename].

Procedure:
1. Insert and remove the device driver module from the kernel and verification.
Do the activity on console window of arm9 target on the hyper terminal.

[root@EDUTECH]cd /user/driver/
[root@EDUTECH]ls
[root@EDUTECH]rmmod led.ko (led driver removed from the kernel)
[root@EDUTECH] cd /home/Example/
[root@EDUTECH]./ledapp.o (it will not work)
[root@EDUTECH] cd /user/driver
[root@EDUTECH] insmod led.ko
[root@EDUTECH] cd /home/Example/
[root@EDUTECH]./ledapp.o (it will work)

2. Developing Device Driver.

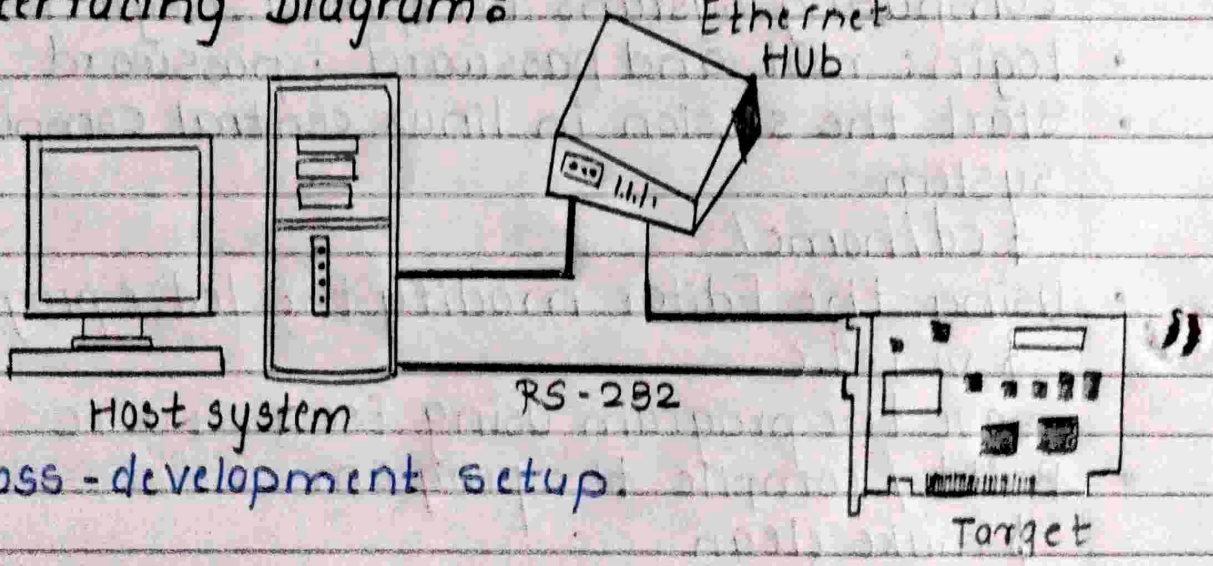
- open the putty utility and login in the central computing systems (if opened, neglect it)
- login: root and password : password
- Start the session in linux central computing system
  $ cd /home/
- Using the Editor modify the led.c program
  $ vi led.c
  save the program using Esc +: +w +c
- Build / compile the program
  $ make clean
  $ make
- Copy it to the /tftpboot
  $ cp led.ko /tftpboot
- Copy led.ko to target board and change its permission

[root@EDUTECH] cd /home/driver/
[root@EDUTECH] tfp -g -r led.ko 172.16.3.xx
(central computing systems' IP add)
[root@EDUTECH] chmd 777 led.ko

- remove the older device driver from the kernel.
  [root@EDUTECH] rmmod led.ko
- insert the device driver to the kernel
  [root@EDUTECH] insmod led.ko

Now execute the LED application
[root@EDUTECH] cd /home/Example/
[root@EDUTECH] ./ledapp.o

Interfacing Diagram:



Cross-development setup.

Observation:

- Observe blinking rate of LED, Change the rate and observe its effect
- Observe different messaged printed on screen throughout the procedure of driver insertion use and removal.

Sanjivani Rural Education Society's

SANJIVANI COLLEGE OF ENGINEERING,

KOPARGAON - 423 603

(An Autonomous Institute)

Department ECE Engineering

PRACTICAL / TERM-WORK REPORT

Subject ES & RTOS.

Name of the Student Wakte Sakshi Vijay.

Class TY ECE Division -

Batch No. T3. Roll No. 73

Name of the Experiment / Title of Term Work Writing "Hello World" device Driver. Loading into & removing from kernel.

Experiment / Term - Work No.

Conducted on :

Expected date of Submission :

Actual date of Submission :

Checked by Prof. Y R Khandekar.
(Name of the Teacher)

Remarks

Signature Date

Page No.
Date

Experiment No-

Title: Writing "Hello World" device Driver. Loading into & removing from kernel.

Aim: Writing a device driver to print "Hello World" on loading in to kernel and print "good bye world" on removal from kernel.

Learning Objectives:

- To understand concept of device driver.
- To understand user mode and kernel mode execution.
- To understand procedure to inset and remove kernel module.

Equipments Required:

Trainer kit of ARM9 controller, serial cable, power supply, ccs server, two Ethernet cables, Ethernet switch.

Theory:

Explain Device driver concept:

- Loadable Modules:
- One of the good features of Linux is the ability to extend at runtime the set of features offered by the kernel. This means that you can add functionality to the kernel (and remove functionality as well) while the system is up and running.
- Each piece of code that can be added to the kernel at runtime is called a Module.

- The Linux kernel offers support for quite a few different types (or classes) of modules, including but not limited to, device driver.
- Each module is made up of object code (not linked into a complete executable) that can be dynamically linked to the running kernel by the insmod program and can be unlinked by the rmmod program.
- insmod is a trivial program to insert a module into the kernel.
- rmmod is a trivial program to remove a module from the kernel.

- Device Driver Architecture:
  Device Driver are software component that allows the operating system to communicate with hardware devices. They typically consist of entry points for handling device request, managing device resource, and interfacing with the kernel.

Minimal Device Driver example:
A Minimal device Driver example would include code to register the device with the operating system, handle device initialization and cleanup and implement the necessary function to communicate with the device.

Procedure:

1. Insert and remove the device driver module from the kernel and Verification.
   Do the activity on console window of arm9 target on the hyper terminal.
   [root@EDUTECH] cd /user/driver/
   [root@EDUTECH] ls
   [root@EDUTECH] insmod hello.ko
   [root@EDUTECH] rmmod hello.ko

2. Developing Device Driver.
   - open the putty utility and login in the central computing system (if opened, neglect it).
   - login: root and password: password
   - Start the session in linux central computing system.
     $ cd/home/
   - Using the Editor modify the hello.c program
     $ vi hello.c
     save the program using Esc + : + w + q
   - Build / compile the program
     $ make clean (to remove old files)
     $ make (it will compile the device driver program).
   - Copy it to the /tftpboot
     $ cp hello.ko /tftpboot
   - Copy hello.ko to target board and change its permission.

[root@EDUTECH] cd /home1/driver1
[root@EDUTECH] tftp -g -r hello.ko 172.16.3.XX
(central computing systems' IP add)
[root@EDUTECH] chmod 777 hello.ko

- remove the older device driver from the kernel.
  [root@EDUTECH] rmmod hello.ko
- insert the device driver to the kernel.
  [root@EDUTECH] insmod hello.ko

Note : We will denote home directory by "HOME$" throughout this tutorial.

Interfacing Diagram:

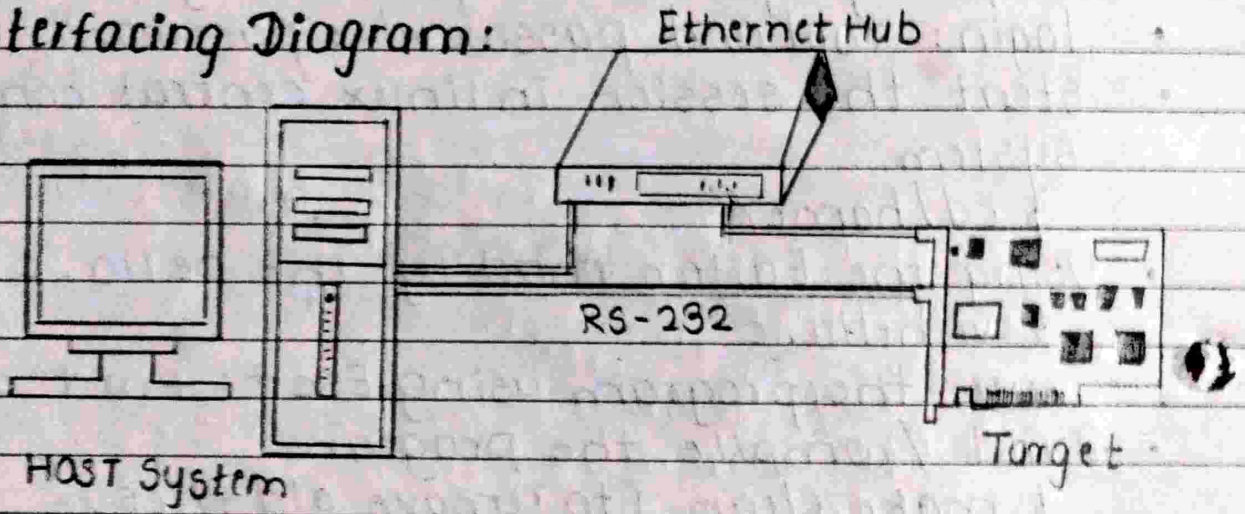


Figure 12-1 Cross-development setup.

Observation:
Observe & note different message printed on terminal screen when driver is inserted or removed.