BIRZEIT UNIVERSITY

Electrical and Computer Engineering

Computer Design Lab – ENCS4110

Serial Communication

Objectives:

1. To be familiar with the USART (RS-232) protocol.

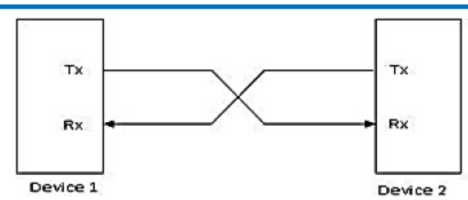
2. To be able to transfer data from LPC-PC, PC-LPC and LPC-LPC.

3. To test serial communications with virtual serial ports with Keil and Proteus programs tools.

Introduction

Serial communications is the process of sending data one bit at one time, sequentially, over a communications channel or computer bus. This is in contrast to parallel communications, where all the bits of each symbol are sent together. Serial communications is communications and most computer networks, where the cost of cable and synchronization difficulties makes parallel communications impractical.

Serial computer buses are becoming more common as improved technology enables them to transfer data at higher speeds. Examples of serial communication architectures includes: RS-232, RS-485, Universal Serial Bus (USB), SPI, I2C and others. Serial interfaces have certain advantages over parallel interfaces. The most significant advantage is simpler wiring. In addition, serial interface cables can be longer than parallel interface cables.



Serial Communication Between 2 devices

USART Protocol

USART stands for the Universal Synchronous/Asynchronous Receiver/Transmitter.

- Universal means that it can be used with a wide scope of devices
- Synchronous devices that communicate with each other require an external synchronization line (the clock).
- Asynchronous The Asynchronous mode (without the common clock) is easier to implement, although it is generally slower than the synchronous. It is also the older way – The COM port in the PC uses the UART, therefore it named as UART (without S).
- Receiver/Transmitter means that this device can receive and transmit (send) data simultaneously. It is also called the two-way or full duplex communication.

USART Asynchronous Mode

Asynchronous transmission allows data to be transmitted without the sender having to send a clock signal to the receiver. In the case, the sender and receiver must agree on timing parameters(Baud Rate) prior transmission and special bits are added to each word to synchronize the sending and receiving units. The sender sends a Start bit, 5-8 data bits, an optional parity bit, and 1-2 stop bits.

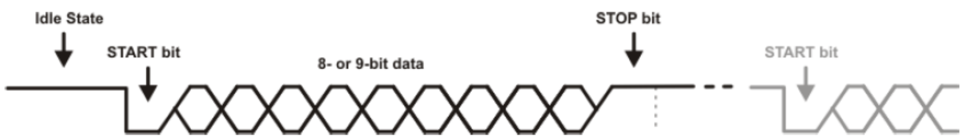
TTL Level

Most microcontrollers with USART uses TTL (Transistor-transistor Logic) level.

Logic	Voltage
Low	0V
High	5V

Voltage level for TTL level UART

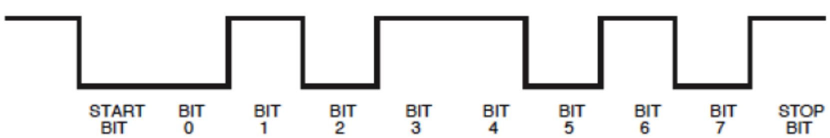
The USART transmits and receives data using standard non-return-to-zero (NRZ) format. As seen in figure below, this mode does not use clock signal, while the data format being transferred is very simple:



Briefly, each data is transferred in the following way:

- In idle state, data line has high logic level (1);
- Each data transmission starts with START bit which is always a zero (0);
- Each data is 8- or 9-bit wide (LSB bit is first transferred); and
- Each data transmission ends with STOP bit which always has logic level which is always a one (1).

For example Data is H'90' = B'01011010' (LSB bit is first transferred)



Each USART contains a shift register which is the fundamental method of conversion between serial and parallel forms. After waiting a further bit time, the state of the line is sampled and the resulting level clocked into a shift register. After the required number of bit periods for the character length (8 ,9 bits typically) have elapsed, the contents of the shift register is made available.

RS-232

The RS-232 standard defines the voltage levels that correspond to logical one and logical zero levels for the data transmission and the control signal lines. Valid signals are plus or minus 3 to 15 volts - the range near zero volts is not a valid RS-232 level.

Logicc	Voltage
Low	+3 to +15V
High	-3 to -15V

Voltage level for RS-232

Pin	Signal abbreviation	Signal Name	DTE (PC)
1	DCD	Data Carrier Detect	In
2	RXD	Receive Data	In
3	TXD	Transmit Data	Out
4	DTR	Data Terminal Ready	Out
5	GDN	Signal Ground	-
6	DSR	Data Set Ready	In
7	RTS	Request to Send	Out
8	CTS	Clear to Send	In
9	RI	Ring Indicator	In

UART Programming

The LPC2138 has two UARTs called UART0 and UART1. The two are identical except that UART1 has some additional functions that make it easier to control a modem. The interface to the outside world is by way of two pins: the transmit pin is TXD0 and the receive pin is RXD0. UART0 has eleven registers which are shown in the table below.

Name	Description	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	Access	Reset Value*	Address
U0RBR	Receiver Buffer Register	MSB			READ DATA				LSB	RO	un-defined	0xE000C000 DLAB = 0
U0THR	Transmit Holding Register	MSB			WRITE DATA				LSB	WO	NA	0xE000C000 DLAB = 0
U0DLL	Divisor Latch LSB	MSB							LSB	R/W	0x01	0xE000C000 DLAB = 1
U0DLM	Divisor Latch MSB	MSB							LSB	R/W	0	0xE000C004 DLAB = 1
U0IER	Interrupt Enable Register	0	0	0	0	0	Enable Rx Line Status Interrupt	Enable THRE Interrupt	Enable Rx Data Available Interrupt	R/W	0	0xE000C004 DLAB = 0
U0IIR	Interrupt ID Register	FIFOs Enabled		0	0	IIR3	IIR2	IIR1	IIR0	RO	0x01	0xE000C008
U0FCR	FIFO Control Register	Rx Trigger		Reserved			Tx FIFO Reset	Rx FIFO Reset	FIFO Enable	WO	0	0xE000C008
U0LCR	Line Control Register	DLAB	Set Break	Stick Parity	Even Parity Select	Parity Enable	Number of Stop Bits	Word Length Select		R/W	0	0xE000C00C
U0LSR	Line Status Register	Rx FIFO Error	TEMT	THRE	BI	FE	PE	OE	DR	RO	0x60	0xE000C014
U0SCR	Scratch Pad Register	MSB							LSB	R/W	0	0xE000C01C
U0TER	Transmit Enable	TxEn	Reserved [6:0]							R/W	0x80	0xE000C030

*Reset Value refers to the data stored in used bits only. It does not include reserved bits content.

To use the UART for simple transmission and reception we need to set up the prescaler divider registers (U0DLM and U0DLL), the line control register (U0LCR), pclk (VPBDIV), and the pin select register to enable the UART pins for transmit and receive (PINSEL0). The remaining register reset states allow the UART to run.

To transmit data you write the U0 transmit holding register (U0THR) and to receive data you read it from the U0 Receive Buffer Register (U0RBR).

The Line Status Register can be used to determine if the UART has a character to read or if its transmit buffer is empty. Since the processor can read and write much faster than the UART can transmit data, you must always check the bits in these registers to determine the UART status before reading or writing.

Line Control Register

The Line Control Register U0LCR bit description is shown in table 5.2 below. This register is used to determine such things as the number of stop bits, even or odd parity, etc.

U0LCR	Function	Description	Reset Value
1:0	Word Length Select	00: 5 bit character length 01: 6 bit character length 10: 7 bit character length 11: 8 bit character length	0
2	Stop Bit Select	0: 1 stop bit 1: 2 stop bits (1.5 if U0LCR[1:0]=00)	0
3	Parity Enable	0: Disable parity generation and checking 1: Enable parity generation and checking	0
5:4	Parity Select	00: Odd parity 01: Even parity 10: Forced "1" stick parity 11: Forced "0" stick parity	0
6	Break Control	0: Disable break transmission 1: Enable break transmission. Output pin UART0 TxD is forced to logic 0 when U0LCR6 is active high.	0
7	Divisor Latch Access Bit	0: Disable access to Divisor Latches 1: Enable access to Divisor Latches	0

For our applications in this lab we will have an 8-bit character length, 1 stop bit, disabled parity checking, disabled break control, and we will enable access to the Divisor latches. For example :

U0LCR = 0x83; //8-bit, 1 stop bit, no parity, enable divisor latch

Pin Select Register

The Pin Function Select (PINSEL) Registers enable you to select which pin functions you would like to use, you need to use one of the the three PINSEL registers: PINSEL0, PINSEL1 and PINSEL2. Which register you would use depends on which pin you want to modify.

PINSEL0 contains GPIO pins 0.0 to 0.15

PINSEL1 contains GPIO pins 0.16 to 0.31

PINSEL2 contains GPIO pins 1.16 to 1.31

Each pin of the controller is assigned a 2-bit address from the PINSEL register. For example, P0.0 uses the first two bits in **PINSEL0**, P0.1 uses the next two bits, and so on. The whole description is as following:

PINSEL0

Pin	0.15	0.14	0.13	0.12	0.11	0.10	0.9	0.8	0.7
Bits	31..30	29..28	27..26	25..24	23..22	21..20	19..18	17..16	15..14

PINSEL1

Pin	0.31	0.30	0.29	0.28	0.27	0.26	0.25	0.24	0.23
Bits	31..30	29..28	27..26	25..24	23..22	21..20	19..18	17..16	15..14

PINSEL2

Pin	1.31	1.30	1.29	1.28	1.27	1.26	1.25	1.24	1.23
Bits	31..30	29..28	27..26	25..24	23..22	21..20	19..18	17..16	15..14

Now let us see what we have to do to select a specific function of a pinTo select a specific function you simply assign one the following 2-bit values to the appropriate location in your PINSEL register:

Binary Value	Selected Function
00	Primary (default) function (always GPIO)
01	First alternate function
10	Second alternate function
11	Third alternate function

To enable UART0 I/O pins bits 3, 2, 1, and 0 should be set to 0101. (UART1 needs bits 19, 18, 17, and 16 set to 0101.

PINSEL0 = 0x5;

Baud Rate Prescale Registers

Both UARTs have their own baud rate generators. The input to these generators is the peripheral clock (pclk).

VPBDIV = 0x1; //make pclk = cclk = 30 MHz

then, the master formula for calculating baud rate is given as :

$$\textbf{BAUD\_RATE} = \mathbf{PCLK\ IN\ HERTZ}/(\mathbf{16} \times (\mathbf{256} \times \mathbf{DLM} + \mathbf{DLL}) \times (\ \mathbf{1} + \mathbf{DIVADDVALMULVAL}))$$

The quickest and also the dirtiest(accuracy wise) method without using any algorithm or finetuning is to set DLM=0 and completely ignore the fraction by setting it to 1. In this case MULVAL=1 and DIVADDVAL=0 which makes the Fraction Multiplier = 1. Now we are left with only 1 unknown in the equation which is U0DLL and hence we simplify the equation and solve for U0DLL.

$$\textbf{U0DLL} = \mathbf{PCLK\ IN\ HERTZ}/(\mathbf{16} \times \mathbf{DESIRED\_BAUDRTE})$$

Now for a baud rate of 9600 baud we want a divisor = pclk/(16 x 9600) = 195.3125. Rounding this to 195 = 0xC3 gives a true baud rate of 9615.

U0DLM = 0;
U0DLL = 0xC3;

Line Status Register

The line status register bit 0 and bit 5 definitions are shown in the table below.

U0LSR	Function	Description	Reset Value
0	Receiver Data Ready (RDR)	0: U0RBR is empty 1: U0RBR contains valid data U0LSR0 is set when the U0RBR holds an unread character and is cleared when the UART0 RBR FIFO is empty.	0
5	Transmitter Holding Register Empty (THRE)	0: U0THR contains valid data. 1: U0THR is empty. THRE is set immediately upon detection of an empty UART0 THR and is cleared on a U0THR write.	1

Bit 0 in this register may be tested to determine if a character has been received and is in the buffer. Bit 5 can be used to determine if the transmit holding register is empty. To transmit a character we can poll bit 5 as shown below:

```
while (!(U0LSR & 0x20)); //Test bit 5

U0THR = ch; //When bit 5 = 1, load data to transmit ch
```

Likewise, to receive a character in a polling mode:

```
while (!(U0LSR & 0x01)); //Test bit 0

 return (U0RBR); //When bit 0 = 1, get data from buffer
```

Lab Work 1

You are going to use these keywords when you search for parts in Proteus:

Part	Keyword
Microcontroller	LPC2138

Write a program that sends a text from LPC to PC.

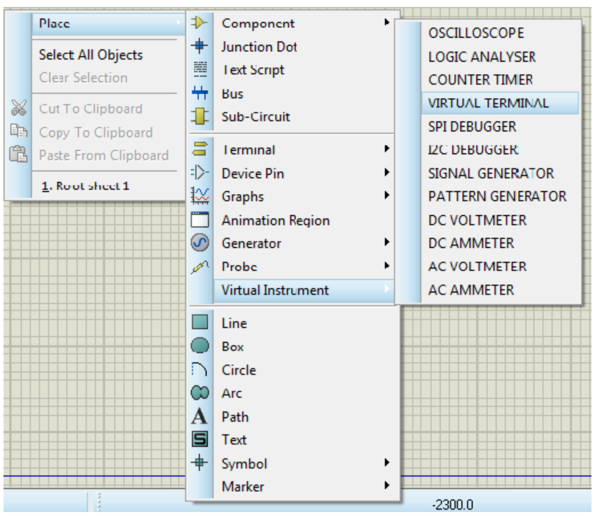
Keil:

```
#define NEW_LINE 0x0D
#include <LPC213x.H>
void UART0_init(void);
int putchar (int ch);
void UART0_write_str(char *msg);

int main() {
        char msg1[]="Welcome to Serial Communication Lab :)";
        char msg2[] = "End";
        UART0_init();
        UART0_write_str(msg1);
        putchar(NEW_LINE);
        UART0_write_str(msg2);

        while(1);

}
```

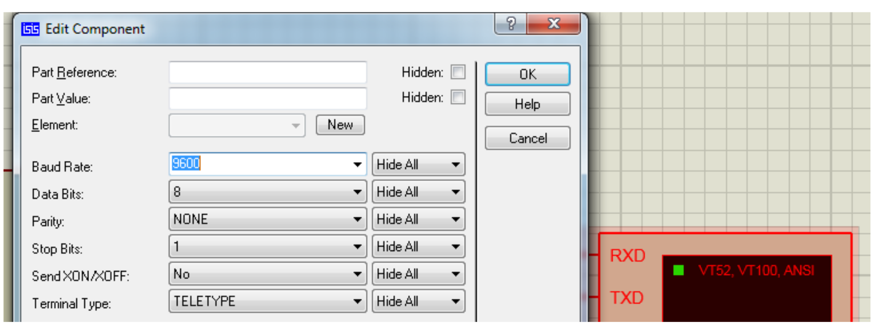
```
void UART0_init(void){
 PINSEL0 = 0x5; /* Enable RxD0 and TxD0 */
 U0LCR = 0x83; /* 8 bits, no Parity, 1 Stop bit */
 U0DLL = 97; /* 9600 Baud Rate @ 12MHz VPB Clock */
 U0LCR = 0x03; /* DLAB = 0 */
}

int putchar (int ch) /* Write single character to Serial Port */
{
   while (!(U0LSR & 0x20));
     return (U0THR = ch);
}

void UART0_write_str(char *str){  /* send string to the serial port */
		int i=0;
		while(str[i]!='\0'){
			putchar(str[i]);
			i++;
		}
}
```

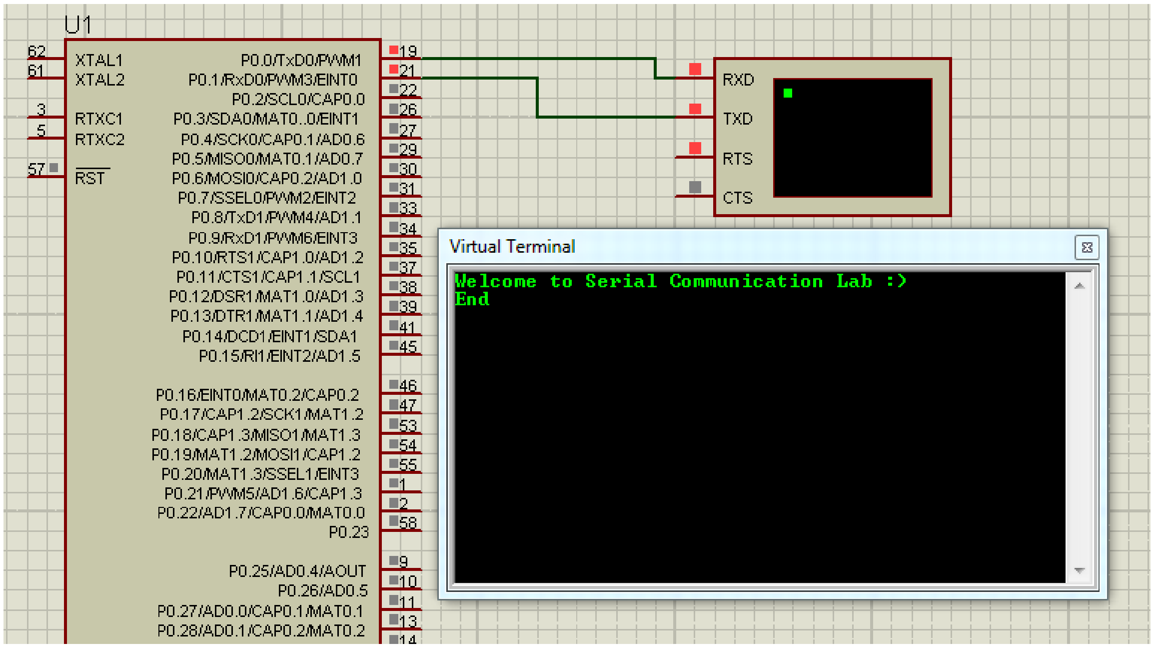
Proteus:

We will use virtual terminal (V.T.) for implementing PC.



Double click << Change the baud rate to 9600





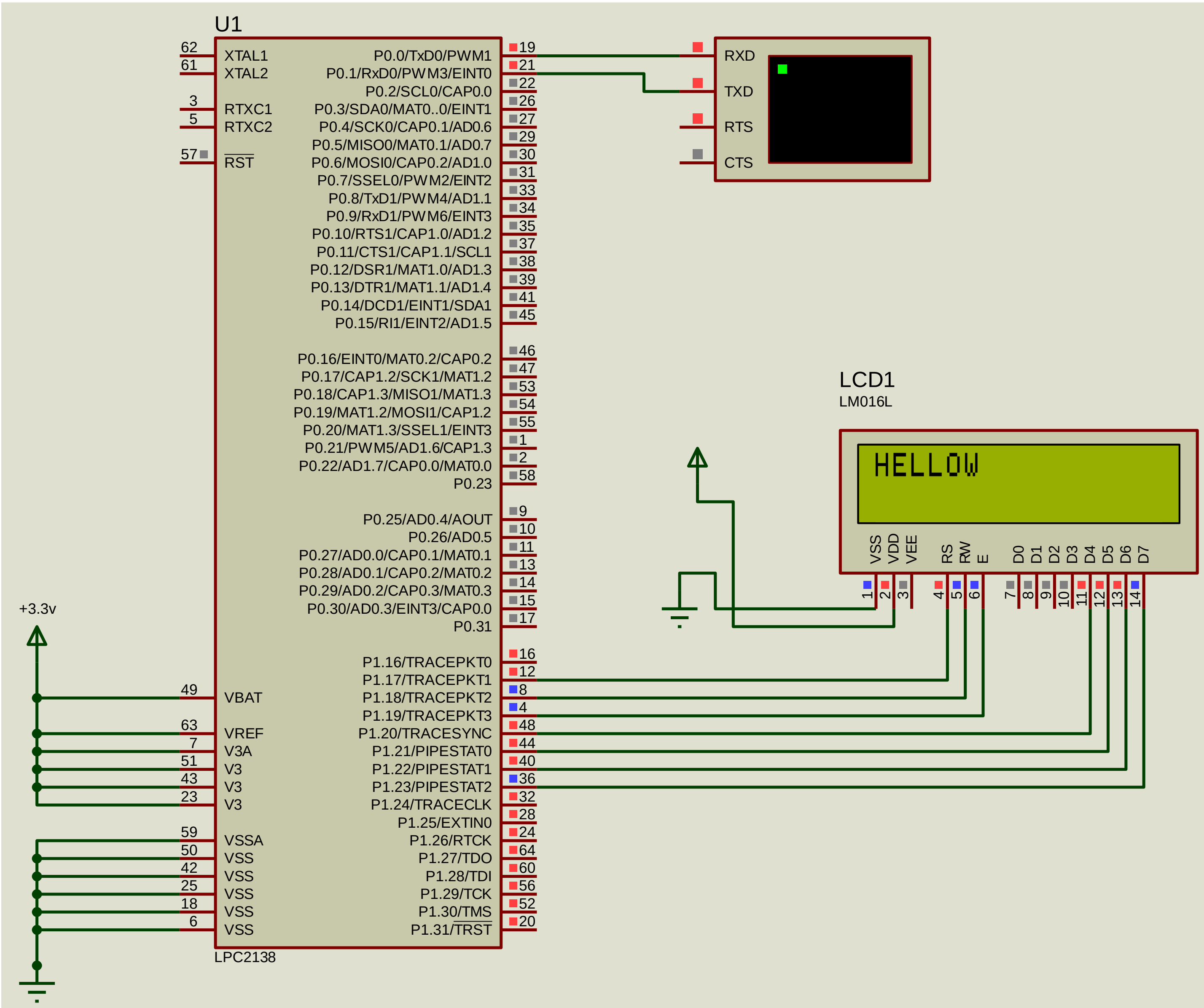
Lab Work 2

Write a program that sends a text from PC to LPC and view on LCD

Keil:

```
#include <lpc213x.H>

char UART0_read(void) {
    while (!(U0LSR & 0x1));    //wait until the receiver buffer is full
     return(U0RBR);
}

Int main() {
   unsigned char name[20] = "ENCS4110";
   unsigned char msg1[50] = "Welcome to Serial Communication Lab:)";
   unsigned char msg2[5] = "write here:";

    IO1DIR = 0x00FE0000; /* LCD pins set as output P1.16 ..P1.19        */
    UART0_init();
    Delay(100);
    UART0_write_str(msg1);
   putchar(NEW_LINE);
    UART0_write_str(msg2);

    LCD_Init();         /* Initialise LCD          */
   Delay(100);
   While(1)
     {
       LCD_Data(UART0_read());
       Delay(50);
     }
}
```

Proteus:


U1
LPC2138
XTAL1
XTAL2
RTXC1
RTXC2
RST
VBAT
VREF
V3A
V3
V3
V3
VSSA
VSS
VSS
VSS
VSS
VSS
P0.0/TxD0/PWM1
P0.1/RxD0/PWM3/EINT0
P0.2/SCL0/CAP0.0
P0.3/SDA0/MAT0..0/EINT1
P0.4/SCK0/CAP0.1/AD0.6
P0.5/MISO0/MAT0.1/AD0.7
P0.6/MOSI0/CAP0.2/AD1.0
P0.7/SSEL0/PWM2/EINT2
P0.8/TxD1/PWM4/AD1.1
P0.9/RxD1/PWM6/EINT3
P0.10/RTS1/CAP1.0/AD1.2
P0.11/CTS1/CAP1.1/SCL1
P0.12/DSR1/MAT1.0/AD1.3
P0.13/DTR1/MAT1.1/AD1.4
P0.14/DCD1/EINT1/SDA1
P0.15/RI1/EINT2/AD1.5
P0.16/EINT0/MAT0.2/CAP0.2
P0.17/CAP1.2/SCK1/MAT1.2
P0.18/CAP1.3/MISO1/MAT1.3
P0.19/MAT1.2/MOSI1/CAP1.2
P0.20/MAT1.3/SSEL1/EINT3
P0.21/PWM5/AD1.6/CAP1.3
P0.22/AD1.7/CAP0.0/MAT0.0
P0.23
P0.25/AD0.4/AOUT
P0.26/AD0.5
P0.27/AD0.0/CAP0.1/MAT0.1
P0.28/AD0.1/CAP0.2/MAT0.2
P0.29/AD0.2/CAP0.3/MAT0.3
P0.30/AD0.3/EINT3/CAP0.0
P0.31
P1.16/TRACEPKT0
P1.17/TRACEPKT1
P1.18/TRACEPKT2
P1.19/TRACEPKT3
P1.20/TRACESYNC
P1.21/PIPESTAT0
P1.22/PIPESTAT1
P1.23/PIPESTAT2
P1.24/TRACECLK
P1.25/EXTIN0
P1.26/RTCK
P1.27/TDO
P1.28/TDI
P1.29/TCK
P1.30/TMS
P1.31/TRST
+3.3v
RXD
TXD
RTS
CTS
LCD1
LM016L
HELLOW
VSS
VDD
VEE
RS
RW
E
D0
D1
D2
D3
D4
D5
D6
D7

Lab work3:

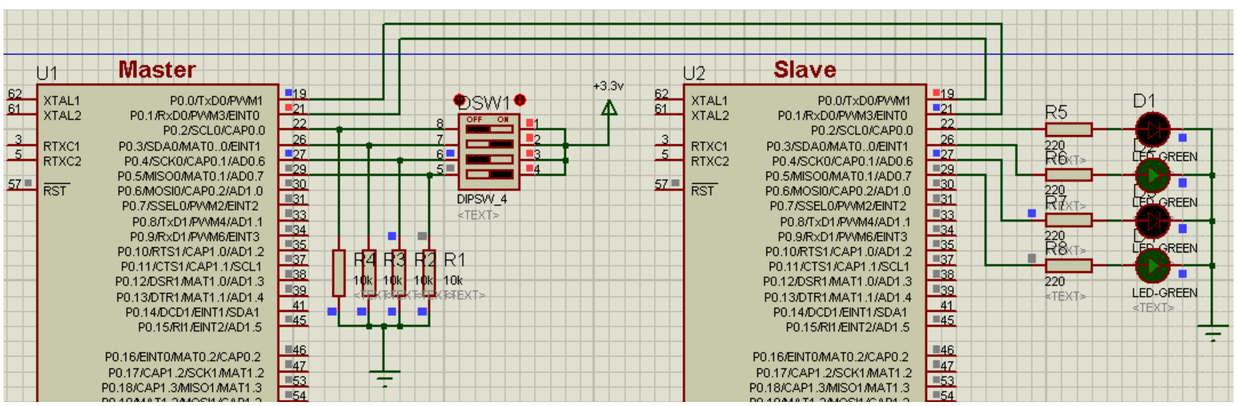
Write a program that sends the state of switch from master LPC to slave LPC(Leds).

Keil:

```c
#include <lpc213x.h>

#define NEW_LINE 0x0D

int main(){
    // Master
    IO0DIR &= (~0x3C);
    UART0_init();
    while(1){
      UART0_write(IO0PIN);
    }
}
```

```c
#include <lpc213x.h>

#define NEW_LINE 0x0D

int main(){
    // Slave
    IO0DIR |= 0x3C;
    UART0_init();
    while(1){
      IO0PIN = UART0_read();
    }
}
```

Proteus:



<u>Post Lab (ToDo)</u>

Write a program (and circuit) that shines the led connected to LPC if the PC (V.T.) entered “hi” word.