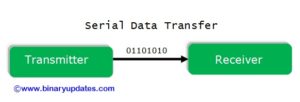
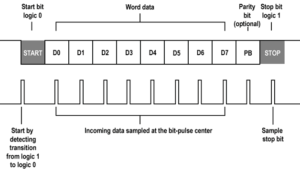
**UART in LPC2148 ARM7 Microcontroller- Serial Communication**

In this tutorial, we will communicate microcontroller and PC over serial communication using UART in LPC2148 ARM7 Microcontroller. Before we move on to UART in LPC2148 ARM7. Let’s first discuss UART in general. UART (Universal Asynchronous Receiver/Transmitter) is one of the earliest mode of communication applied to computer (somewhere in 1960s). The information is transmitted one binary bit at a time; as such it is a serial communication method. These bits are grouped together in the form of ‘**Frames**’ (a set format) for conveying one meaningful piece of data (e.g. character byte). UART is asynchronous because it doesn’t require a transmitter provided clock to synchronize the transmission and receipt of data.

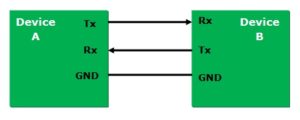
[](http://www.binaryupdates.com/wp-content/uploads/serial_data_transfer_uart_lpc2148.jpg)Serial Data Transfer In UART

**Serial Data Transmission in UART**

Just because there is no clock signal per se, a start bit is added sent first to tell the receiver to listen out for data. The receiver monitors for a logic HIGH falling to logic LOW. The receiver synchronizes its own bus clock to that make up the word being sent, with bit zero, the least significant bit (LSB) being sent first. The bits are sent as pulses on the wire at specific time intervals, set at both ends of links to previously agreed values. The receiver looks at the voltage on the wire at these times; if it sees logic high, it records a binary digit 1 or 0 if the line is low. The receiver checks half way between the start and the end of the pulse to ensure it does not miss-read the voltage on the line during the brief interval while the voltage is rising or falling.

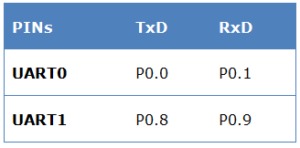
[](http://www.binaryupdates.com/wp-content/uploads/Serial-Data-Transmission-in-UART.png)Serial Data Transmission in UART

If two devices use a parity bit for rudimentary error checking, that is calculated and sent next, in sync with data that has been transmitted thus far. Finally, one stop bit is sent by the transmitter. Word length, parity availability and type, and numbers of stop bits all have to be agreed in advance for successful communication because UART uses two wires. The transmitter of device-A connected to receiver of device-B and receiver of device-A connected to transmitter of device-B. This is how devices can send data simultaneously to each other, a mode of communication called ‘**full duplex**’.

[](http://www.binaryupdates.com/wp-content/uploads/Serial-Device-Communication.jpg)Serial Device Communication

**INTRODUCTION: UART in LPC2148 ARM7 Microcontroller**

As we all know UART is widely used serial communication protocol in embedded system based applications. Almost all microcontrollers have built-in on-chip UART support. LPC2148 ARM7 core supports two UART in it, UART0 and UART1. UART0 can be used as general purpose UART and also can support ISP Programming through it, whereas UART1 has additional modem support. Both have built in baud rate generator and 16-byte transmit and receive FIFOs. For UART0 the TxD Pin is at P0.0 and RxD Pin is at P0.1 and similarly for UART1 the TxD Pin is at P0.8 and RxD Pin is at P0.9 as shown in table below;

[](http://www.binaryupdates.com/wp-content/uploads/List-UART-in-LPC2148.png)List of UART in LPC2148

**RS232 Level Converter**

Most of microchips work on TTL or CMOS voltage level which can’t be used to communicate over RS-232 protocol. In this case voltage or level converter is needed which can convert TTL to RS-232 and RS-232 to TTL voltage levels. The most commonly used RS-232 level converter is MAX3232 chip. This chip includes charge pump which can generate RS232 voltage levels (-10V and +10V) from 5V power supply. It also includes two receiver and two transmitters and is capable of full-duplex UART communication. RS232 communication enables point-to-point data transfer, which often used in data acquisition applications and for data transfer between microcontroller and PC.

In example project, we will use RS232 cable (USB to Serial Converter) to accomplish communication between LPC2148 Microcontroller and PC. **\**Make sure to install appropriate drivers before you make use of it.***

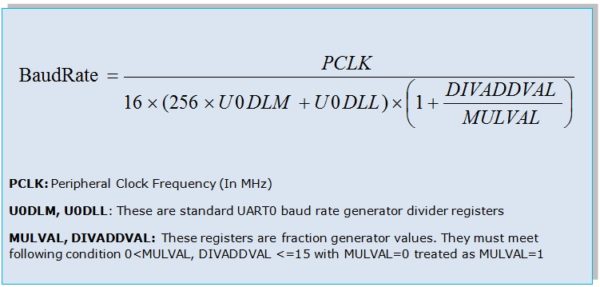
**Registers Description: UART in LPC2148 ARM7**

Before we use any pin from LPC2148 for serial communication. We have to configure and initialize UART0 Peripheral by setting register bits. Let’s quickly revise functions of UART0 registers which will be used in our code later on. [*It is recommended to keep user manual open to collect more details about each register if in case needed*].

|  |  |
| --- | --- |
| **Registers** | **Description/Function** |
| **U0THR** | **Transmit Hold Register:** This register contains 8-bit write data which can be transmitted through UART0. This is write only register. |
| **U0RBR** | **Receive Buffer Register:** This register contains 8-bit received data from UART0. This data is nothing but top most byte of Rx FIFO. When we use 5, 6 or 7-bit data then remaining bits are padded with 0’s by default. This is read only register. |
| **U0LCR** | **Line Control Register:** The value or settings in this register configure the UART0 block. As this is an 8-bit register. There are several parameters configured through this register such as word length, stop bit, parity enable, parity select, break control, divisor latch access bit. This register setting plays important role while initializing UART0 before using it. |
| **U0DLL & U0DLM** | **U0DLL & U0DLM are standard** UART0 baud rate generator divider registers. Each of this register holds 8-bit values. Together these registers form a 16-bit divisor value which will be used for baud rate generation. This will be discussed further while code explanation with respect to real world example. |
| **U0FDR** | **Fractional Divider Register:** This is another very important register, which plays significant role in baud rate generation. In this 8-bit register, first four bits i.e. **Bit[3 to 0]-DIVADDVAL:** This is the Prescale Divisor value. If this value is 0 then fractional baud rate generator have no effect on UART0 baud rate. The remaining 4-bits i.e. **Bit[4 to 7]-MULVAL:** This defines Prescale Multiplier value. Even if fractional baud rate generator is not used the value in this register must be more than or equal to ‘1’. |

**Calculate Baud Rate for UART in LPC2148 ARM7**

Baud Rate for UART in LPC2148 ARM7 Microcontroller can be calculated by given equation:

[](http://www.binaryupdates.com/wp-content/uploads/Baud-Rate-Calculation-in-LPC2148.jpg)Baudrate Calculation in LPC2148

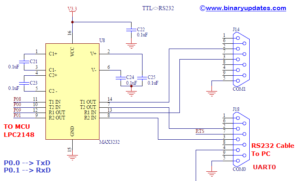
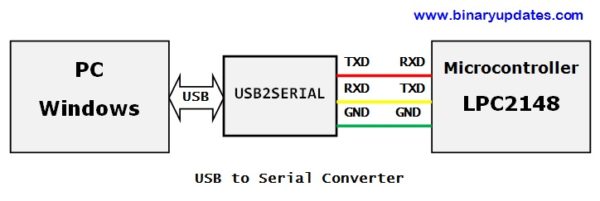
**Example Project:**

Let’s look at real world example where we will send string from LPC2148. This message will be received and displayed on terminal emulator or serial console which configured to appropriate COM Port of PC. I believe now you’re familiar with how UART works. In our program we will use following configuration to establish proper communication:

* **Baud Rate = 9600 baud** (with PCLK=60Mhz)
* **Data Length = 8 bits**
* **No Parity Bit**
* **and 1 Stop Bit**

To work out on this project we need following things to be setup:  
**Software Requirements:** Install Keil uVision4, Flash Magic, PuTTY (terminal emulator or serial console)  
**Hardware Requirements:** LPC2148 Development Board, RS232 Cable (USB Serial Converter), Power Adapter (9V-500mA).

**Circuit Diagram: UART in LPC2148 ARM7 Microcontroller**

[](http://www.binaryupdates.com/wp-content/uploads/Circuit-Diagram-UART-in-LPC2148-ARM-Microcontroller.png)Circuit Diagram-UART in LPC2148 ARM Microcontroller [](http://www.binaryupdates.com/wp-content/uploads/Connection-between-LPC2148-and-PC.jpg)Connection between LPC2148 and PC

**Source Code: UART in LPC2148 ARM7 Microcontroller**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69 | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* PROJECT NAME: UART in LPC2148 ARM7              \*/  /\* Device:       LPC2148                 \*/  /\* Filename:     uart.c                                   \*/  /\* Language:     C                                        \*/  /\* Compiler:     Keil ARM              \*/  /\* For more detail visit www.binaryupdates.com        \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include <lpc214x.h>    void initClocks(void);  void initUART0(void);  void U0Write(char data);  void Send\_String(char\* StringPtr);    char String[]="Hello from BINARYUPDATES.COM !!! \n\r\n";  unsigned int delay;    int main(void)  {  initClocks(); // Set CCLK=60Mhz and PCLK=60Mhz  initUART0();  while(1)  {  Send\_String(String);    //Pass the string to the USART\_putstring function and sends it over the serial  for(delay=0; delay<500000; delay++); // delay  }  }    void initUART0(void)  {  PINSEL0 = 0x5;  /\* Select TxD for P0.0 and RxD for P0.1 \*/  U0LCR = 0x83; /\* 8 bits, no Parity, 1 Stop bit | DLAB set to 1 \*/  U0DLL = 110;  U0DLM = 1;  U0FDR = 0xF1; /\* MULVAL=15(bits - 7:4) , DIVADDVAL=0(bits - 3:0)\*/  U0LCR &= 0x0F; // Set DLAB=0 to lock MULVAL and DIVADDVAL  //BaudRate is now ~9600 and we are ready for UART communication!  }    void U0Write(char data)  {  while (!(U0LSR & (1<<5))); // wait till the THR is empty  // now we can write to the Tx FIFO  U0THR = data;  }    void initClocks(void)  {  PLL0CON = 0x01;   //Enable PLL  PLL0CFG = 0x24;   //Multiplier and divider setup  PLL0FEED = 0xAA;  //Feed sequence  PLL0FEED = 0x55;  while(!(PLL0STAT & 0x00000400)); //is locked?  PLL0CON = 0x03;   //Connect PLL after PLL is locked  PLL0FEED = 0xAA;  //Feed sequence  PLL0FEED = 0x55;  VPBDIV = 0x01;    // PCLK is same as CCLK i.e.60 MHz  }    void Send\_String(char\* StringPtr){  while(\*StringPtr != 0x00){  U0Write(\*StringPtr);  StringPtr++;}  } |

# LPC2148 Serial Communication Tutorial

Coming to UART in LPC2148, the LPC214x series of MCUs have two UART blocks called UART0 and UART1. Each UART block is associated with two pins, one for transmission and the other for receiving.

In UART0 block, the TXD0 (Transmit) and RXD0 (Receive) pins in the device are P0.0 and P0.1 respectively. In case of UART1, the TXD1 and RXD1 pins are P0.8 and P0.9 respectively.

|  | **UART 0** | **UART 1** |
| --- | --- | --- |
| TX | P0.0 | P0.8 |
| RX | P0.1 | P0.9 |

Both the UART modules are identical, except the UART1 block has an additional full modem interface. This includes all the pins for RS232 compatibility like flow control pins (CTS, RTS) etc. Both UART0 & UART1 blocks internally have a 16-byte FIFO (First In First Out) structure to hold the Rx and Tx data.  In this tutorial we will see only UART0. But If you understand the UART0 obviously you can play with UART1.

# Registers Used For UART

This below table shows the registers used for UART.

| **Registers** | **Description** |
| --- | --- |
| UxTHR | Contains the data to be transmitted |
| UxRBR | Contains the recently received Data |
| UxLCR | Controls the UART frame formatting(Number of Data Bits, Stop bits) |
| UxFDR | Controls the clock pre-scaler for the baud rate generation |
| UxDLL | Least Significant Byte of the UART baud rate generator value |
| UxDLM | Most Significant Byte of the UART baud rate generator value |
| UxLSR | Provides status information on the UART0 TX and RX blocks |
| UxFCR | Controls the operation of the UART0 Rx and TX FIFOs |

**Note:** x-> 0/1 (UART0/UART1)

## UxTHR (UART 0/1 Transmit  Holding Register)

The UxTHR is the top byte of the UART0/1 TX FIFO. The top byte is the newest character in the TX FIFO and can be written via the bus interface. The LSB represents the first bit to  
transmit.  
The Divisor Latch Access Bit (DLAB) in UxLCR must be zero in order to access the  
UxTHR. The UxTHR is always Write Only.

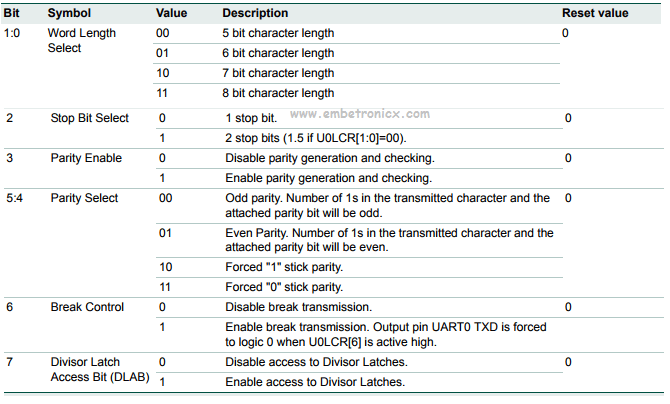
## UxRBR (UART 0/1 Receiver Buffer Register)

The UxRBR is the top byte of the UART0/1 Rx FIFO. The top byte of the Rx FIFO contains the oldest character received and can be read via the bus interface. The LSB (bit 0) represents the “oldest” received data bit. If the character received is less than 8 bits, the unused MSBs are padded with zeroes.

The Divisor Latch Access Bit (DLAB) in UxLCR must be zero in order to access the  
UxRBR. The UxRBR is always Read Only.

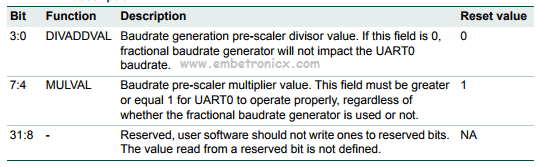
## UxLCR (UART 0/1 Line Control Register)

The Line Control Register is used to set the format of the data which is transmitted or received. The value or settings in this register configure the UART0/1 block. As this is an 8-bit register. There are several parameters configured through this register such as word length, stop bit, parity enable, parity select, break control, divisor latch access bit. This register setting plays important role while initializing UART0/1 before using it.

[](https://i2.wp.com/www.embetronicx.com/wp-content/uploads/2017/07/line-control-register.png)

## UxFDR (Fractional Divider Register)

This register is used to set the prescale value for baud rate generation. The input clock is the peripheral clock and output is the desired clock defined by this register. This register actually holds to different 4-bit values (a divisor and a multiplier) for prescaling which are:

[](https://i2.wp.com/www.embetronicx.com/wp-content/uploads/2017/07/Fractional-Divider-Register.png)

**Important:** If the fractional divider is active (DIVADDVAL > 0) and DLM = 0, the value of the DLL register must be 3 or greater

**DIVADDVAL:** This is the Prescale Divisor value. If this value is 0 then fractional baud rate generator have no effect on UART0/1 baud rate.

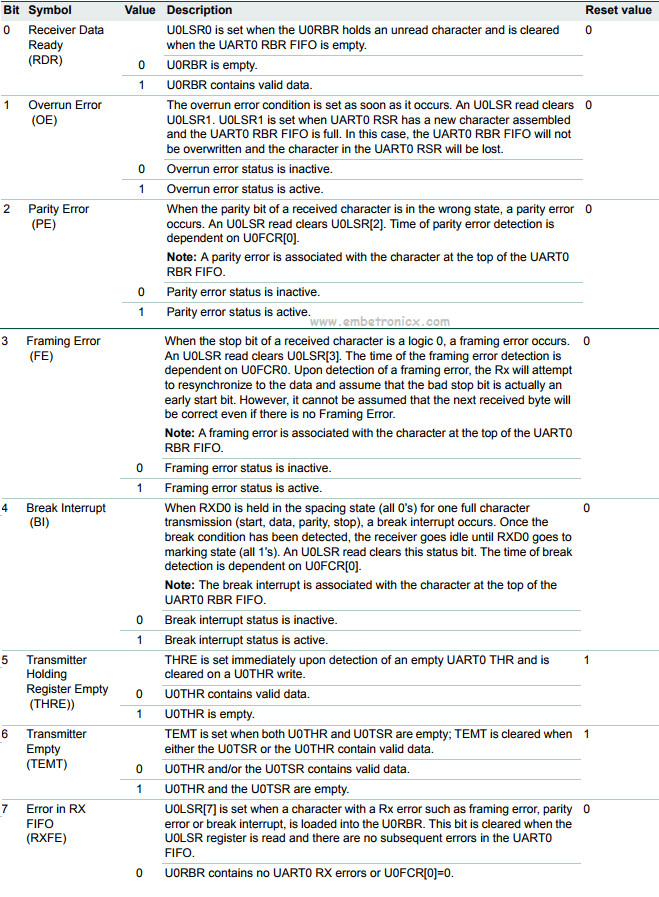
**MULVAL:** This defines Prescale Multiplier value. Even if fractional baud rate generator is not used the value in this register must be more than or equal to ‘1’.

## UxDLL & UxDLM (UART 0/1 Divisor Latch registers)

These are standard UART0/1 baud rate generator divider registers. Each of this register holds 8-bit values. Together these registers form a 16-bit divisor value which will be used for baud rate generation. UxDLM holds the upper 8-bits and UxDLL holds the lower 8-bits and the formation is “[UxDLM:UxDLL]“. Since these form a divisor value and division by zero is invalid, the starting value for UxDLL is 0x01 (and not 0x00) i.e the starting value in combined formation is “[0x00:0x01]” i.e 0x0001. Please keep this in mind while doing baud-rate calculations. In order to access and use these registers properly, DLAB bit in UxLCR must be first set to 1.

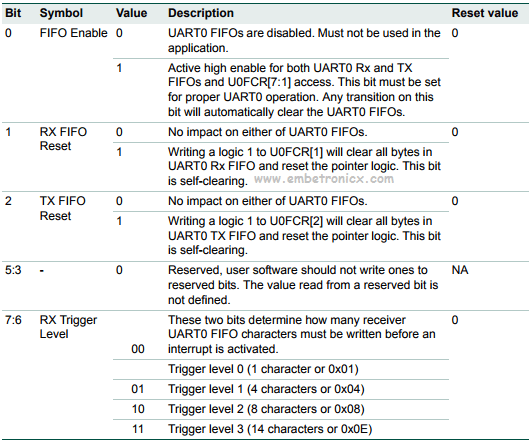
## UxLSR (UART 0/1 Line Status Register)

The Line Status Register gives us the information about the RX and TX blocks in UART0/1.

[](https://i1.wp.com/www.embetronicx.com/wp-content/uploads/2017/07/line-status-register.png)

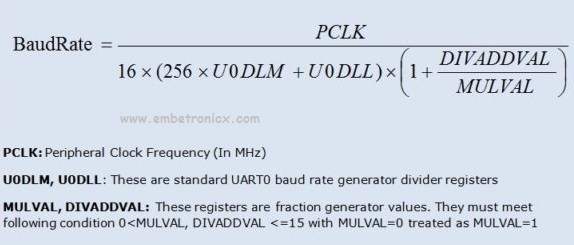
## UxFCR (UART 0/1 FIFO Control Register)

The FIFO Control Register controls the operation of the RX and TX FIFOs in UART0/1.

[](https://i0.wp.com/www.embetronicx.com/wp-content/uploads/2017/07/fifo-control-register.png)

# Baud Rate Calculation

By using this below formula we will calculate the Baud Rate.

[](https://i0.wp.com/www.embetronicx.com/wp-content/uploads/2017/07/Baud-Rate-Calculation-in-LPC2148-600x287.jpg)

## Example Calculation 1 (Baud Rate 9600 And PCLK = 60MHz)

First Assume **U0DLM = 0 , DIVADDVAL = 0 & MULVAL = 1 with PCLK = 60 x 106 Hz**

Substitute into that above formula.

U0DLL = 60000000/(16\*9600)

U0DLL = 390.625 (~390)

Now the U0DLL value is greater than 255. So we cant store it into this U0DLL Register. Because it is 8-but register.

Now we will change our assumed values.

Now assume **U0DLM =1**

9600 = 60000000/(16\*(256+U0DLL))

4096+16U0DLL = 60000000/9600

16U0DLL = 6250 - 4096

U0DLL = 2154/16

U0DLL = 134.625 (~135)

This U0DLL (135) gives 9591 Baud Rate. This is less than 9600. But we can take this value. Because less difference only.

Finally,

PCLK = 60MHz

U0DLL = 135

DIVADDVAL = 0

MULVAL =1

U0DLM = 1

Baud Rate = 9600

## Example Calculation 2 (Baud Rate 9600 And PCLK = 30MHz)

First Assume **U0DLM = 0 , DIVADDVAL = 0 & MULVAL = 1 with PCLK = 30 x 106 Hz**

U0DLL = 30000000/(16\*9600)

U0DLL = 195.3125 (~195)

This U0DLL (195) gives 9615.38 Baud Rate. This is greater than 9600. But we can take this value. Because less difference only.

PCLK = 30MHz

U0DLL = 195

DIVADDVAL = 0

MULVAL =1

U0DLM = 0

Baud Rate = 9600

**Note** : 9600 is pretty famous Baud rate in embedded projects. Some of the standard Baud rates that can be used are : 2400 , 4800 , 7200 , 9600 , 14400 , 19200 , 28800 , 38400 , 57600 , 115200 , 230400 , etc..

# Programming Explanation

## Initializing UART

Below are the steps for configuring the UART0.

1. .Configure the GPIO pin for UART0 function using PINSEL register.
2. .Configure LCR for 8-data bits, 1 Stop bit, Disable Parity and Enable DLAB.
3. .Update the DLM,DLL with the calculated values.
4. .Finally clear DLAB to disable the access to DLM,DLL.

After this, the UART will be ready to Transmit/Receive Data at the specified baudrate.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | void ser\_int()  {  PINSEL0=0X5;  U0LCR=0X83;            //8-data bits, 1 Stop bit, Disable Parity and Enable DLAB  U0DLL=135;  U0DLM=1;  U0LCR=0X03;            //cleared DLAB  } |

## Transmit (TX)

1. Wait till the previous char is transmitted ie. till THRE becomes high in U0LSR Register.
2. Load the new char to be transmitted into U0THR.



|  |  |
| --- | --- |
| 1  2  3  4  5 | void tx(unsigned char c)  {  while((U0LSR&(1<<5))==0);    //Checking THRE bit  U0THR=c;  } |

## Receive (RX)

1. Wait till the a char is received ie. till RDR becomes high in U0LSR.
2. Copy the received data from receive buffer(RBR).



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | char rx()  {  unsigned char a;  while((U0LSR&(1<<0))==0);         //Checking RDR bit  a=U0RBR;  return a;  } |

# Code 1

This is like a echo program. Whatever typing it will resend via UART. Here PCLK = 60MHz, BaudRate=9600.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | #include<lpc214x.h>    void pll()  {  PLL0CON=0X01;  PLL0CFG=0X24;  PLL0FEED=0XAA;  PLL0FEED=0X55;  while((PLL0STAT&(1<<10))==0);  PLL0CON=0X03;  PLL0FEED=0XAA;  PLL0FEED=0X55;  VPBDIV=0x01; //pclk=60mhz  }    void ser\_int()  {  PINSEL0=0X5;  U0LCR=0X83;  U0DLL=135;  U0DLM=1;  U0LCR=0X03;  }    void tx(unsigned char c)  {  while((U0LSR&(1<<5))==0);  U0THR=c;  }    char rx()  {  unsigned char a;  while((U0LSR&(1<<0))==0);  a=U0RBR;  return a;  }    int main(void)  {  unsigned char b;  pll();  ser\_int();  while(1) {  b=rx();  tx(b);  }  } |

# Code 2

This is also the same echo program. But Here PCLK = 30MHz, BaudRate=9600.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | #include<lpc214x.h>    void pll()  {  PLL0CON=0X01;  PLL0CFG=0X24;  PLL0FEED=0XAA;  PLL0FEED=0X55;  while((PLL0STAT&(1<<10))==0);  PLL0CON=0X03;  PLL0FEED=0XAA;  PLL0FEED=0X55;  VPBDIV=0x02; //pclk=30mhz  }    void ser\_int()  {  PINSEL0|=0x05;  U0LCR=0x83;  U0DLL=195;  U0DLM=0;  U0LCR=0x03;    }    void tx(unsigned char c)  {  while((U0LSR&(1<<5))==0);  U0THR=c;  }    char rx()  {  unsigned char a;  while((U0LSR&(1<<0))==0);  a=U0RBR;  return a;  }    int main(void)  {  unsigned char b;  pll();  ser\_int();    while(1) {  b=rx();  tx(b);  }  } |

That is all guys…..You can also try this with UART1. If you have any doubt please let us know.

# LPC2148 UART Tutorial

December 3, 2016 By [Administrator](https://www.electronicshub.org/author/elktros/) [2 Comments](https://www.electronicshub.org/lpc2148-uart-tutorial/#comments)

The simplest way to communicate between a computer and a microcontroller is through UART i.e. Universal Asynchronous Receiver Transmitter. It is a type of Serial Communication Protocol and it uses only two wires for transferring the data. Normally, we need to connect the microcontroller to the computer using a COM Port. But the modern day computers and laptops doesn’t include a COM port as the USB ports have replaced them.

Hence now – a – days, the interface between a microcontroller and the computer is through USB Port with the help of USB – to – RS232 modules that are based on the USB – to – UART Bridge ICs like CP210x from Silicon Labs or FT232R from FTDI.

Table of Contents

* [Basics of UART](https://www.electronicshub.org/lpc2148-uart-tutorial/#Basics_of_UART)
* [UART in LPC2148](https://www.electronicshub.org/lpc2148-uart-tutorial/#UART_in_LPC2148)
* [Registers associated with UART in LPC2148](https://www.electronicshub.org/lpc2148-uart-tutorial/#Registers_associated_with_UART_in_LPC2148)

### Basics of UART

Before going to the UART Programming in LPC2148 MCUs, let us first see some basics of the UART Protocol. As mentioned earlier, the UART Protocol uses only two wires (or pins in a device like microcontroller) to transmit the data. In that, one is for transmitting the data and the pin is called TX pin in the device. The other pin is used to receive the data and is called RX pin.

As UART is a serial communication, the data is transmitted in a series of packets. Usually, a packet consists of 4 parts: a start bit, the actual data, a parity bit and stop bits. The following image shows a typical structure of the data packet in UART.

### UART in LPC2148

Coming to UART in LPC2148, the LPC214x series of MCUs have two UART blocks called UART0 and UART1. Each UART block is associated with two pins, one for transmission and the other for receiving.

In UART0 block, the TXD0 (Transmit) and RXD0 (Receive) pins in the device are P0.0 and P0.1 respectively. In case of UART1, the TXD1 and RXD1 pins are P0.8 and P0.9 respectively.

|  |  |  |  |
| --- | --- | --- | --- |
| UART0 | | UART1 | |
| TXD0 | P0.0 | TXD1 | P0.8 |
| RXD0 | P0.1 | RXD1 | P0.9 |

Both the UART modules are identical, except the UART1 block has an additional full modem interface. This includes all the pins for RS232 compatibility like flow control pins (CTS, RTS) etc.

Both the UART blocks have 16 byte Receive and Transmit FIFO structures to hold the transmit and receive data. In order to control the data access and assembly, the UART blocks have two registers each.

For the transmitter operation, the TX has two special registers called Transmit Holding Register (THR) and Transmit Shift Register (TSR). In order to transmit the data, it is first sent to THR and then moved to TSR.

For the receiver operation, the RX has two special registers called Receiver Buffer Register (RBR) and Receive Shift Register (RSR). When the data is received, it is first stored in the RSR and then moved to RBR.

### Registers associated with UART in LPC2148

There are many registers involved with the UART blocks. UART0 and UART1 have a similar register structure and here we are mentioning some of the important registers of the UART0 block. In order to access the registers in UART1 block, simply replace the ‘0’ with ‘1’.

**UART0 Receiver Buffer Register (U0RBR)**: The Receiver Buffer Register consists of the top byte of the RX FIFO. This is the data that is first arrived and Bit 0 contains the oldest data. In case the received data is less than 8 bits, the remaining bits are padded with 0’s. In order to access the U0RBR register, the Divisor Latch Access bit (DLAB) in the UART0 Line Control Register (U0LCR) must be set to 0. With reference to the user manual, it is advised that the U0LSR register is read first and then the U0RBR register is read.

**UART0 Transmit Holding Register (U0THR)**: The Transmit Holding Register consists of the top byte of the TX FIFO i.e. the oldest data. Bit 0 (LSB) is the first data to be transmitted. In order to access the U0THR register, the Divisor Latch Access Bit (DLAB) bit in the UART0 Line Control Register (U0LCR) must be made zero.

**UART0 Divisor Latch Registers (U0DLL and U0DLM)**: The Divisor Latch Registers determine the baud rate of the UART0. The Divisor Latch is a part of the Baud Rate Generator. The U0DLL and U0DLM registers contains the lower and higher 8 – bits of the divisor and together they form the 16 – bit divisor value. Since it contains the divisor value, the U0DLL cannot contain 0x00 as division by zero is invalid. Hence, the minimum value for the U0DLM:U0DLL combination is 0x00:0x01. In order to access the Divisor Latch Registers, the Divisor Latch Access Bit (DLAB) bit in the UART0 Line Control Register (U0LCR) must be set to 1.

**UART0 Fractional Divider Register (U0FDR)**: The Fractional Divider Register contains the bits that control the prescale for the baud rate generator. U0FDR contains the divisor and multiplier values for prescaling. Both the divisor (DIVADDVAL) and multiplier (MULVAL) values are stored as 4 – bit values. Bit 0 to Bit 3 in U0FDR contains the pre – scaler divisor value for baud rate generation. For the baud rate generator to have an impact on the baud rate of the UART, the divisor value must not be 0. Bit 4 to bit 7 in U0FDR contains the pre – scaler multiplier value. For proper working of the UART0, the value in multiplier must be greater than or equal to 1. This is also applicable even if the baud rate generator is not used.

**UART0 Interrupt Enable Register (U0IER)**: The Interrupt Enable Register is used to enable or disable the interrupts corresponding to UART0. Bit 0 is for RBR Interrupt, Bit 1 is for THRE interrupt, Bit 2 is for RX Line Status Interrupt, Bit 8 is for End of Auto – baud interrupt and Bit 9 is for Auto – baud Time out interrupt. When a bit is 0, the interrupt is disabled and when the bit is 1, the interrupt is enabled.

**UART0 Interrupt Identification Register (U0IIR)**: The Interrupt Identification Register is used to provide the code for priority and status of the pending interrupt.

**UART0 FIFO Control Register (U0FCR)**: The FIFO Control Register controls the operation of the RX and TX FIFOs in UART0. Bit 0 is used to enable or disable the FIFO. Bit 1 is used to reset the RX FIFO. Bit 2 is used to reset the TX FIFO. Bits 6 and 7 are used to control when the interrupt must occur i.e. after how many receiver characters.

**UART0 Line Control Register (U0LCR)**: The Line Control Register is used to set the format of the data which is transmitted or received.

|  |  |  |  |
| --- | --- | --- | --- |
| Bit | Name | Value | Description |
| 1:0 | Word Length | 00 | 5 bit data length |
| 01 | 6 bit data length |
| 10 | 7 bit data length |
| 11 | 8 bit data length |
| 2 | Stop Bit | 0 | 1 stop bit |
| 1 | 2 stop bits |
| 3 | Parity Enable | 0 | Disable Parity |
| 1 | Enable Parity |
| 5:4 | Parity Select | 00 | Odd Parity |
| 01 | Even Parity |
| 10 | Forced 1 parity |
| 11 | Forced 0 parity |
| 6 | Break Control | 0 | Disable break transmission |
| 1 | Enable break transmission |
| 7 | Divisor Latch Access Bit (DLAB) | 0 | Disable access to Divisor Latches |
| 1 | Enable access to Divisor Latches |

**UART0 Line Status Register (U0LSR)**: The Line Status Register gives us the information about the RX and TX blocks in UART0.

|  |  |  |  |
| --- | --- | --- | --- |
| Bit | Symbol | Value | Description |
| 0 | Receiver Data  Ready | 0 | U0RBR is empty |
| 1 | U0RBR contains valid data |
| 1 | Overrun Error | 0 | Overrun error status is inactive |
| 1 | Overrun error status is active |
| 2 | Parity Error | 0 | Parity error status is inactive |
| 1 | Parity error status is active |
| 3 | Framing Error | 0 | Framing error status is inactive |
| 1 | Framing error status is active |
| 4 | Break Interrupt | 0 | Break interrupt status is inactive |
| 1 | Break interrupt status is active |
| 5 | Transmitter Holding Register Empty | 0 | U0THR contains valid data |
| 1 | U0THR is empty |
| 6 | Transmitter Empty | 0 | U0THR and/or the U0TSR contains valid data |
| 1 | U0THR and the U0TSR are empty |
| 7 | Error in RX FIFO | 0 | U0RBR contains no UART0 RX errors |
| 1 | At least one UART0 RX error in U0RBR |

**UART0 Transmit Enable Register (U0TER)**: The Transmit Enable Register is used to enable the data transmission i.e. it is used to enable implementation of software flow control. The Bit 7 in U0TER register i.e. TXEN is the only bit available for user. When this bit is high, the UART0 TX will keep on sending data. When this bit becomes 0, the data transmission will stop.

**Baud Rate Generation in UART**

The formula for calculating UART baud rate is given below

Baud Rate =

The following rules must be followed for the Baud rate generation.

Minimum value for MULVAL must be 1 i.e. 0 < MULVAL <= 15.

The value for DIVADDVAL can be between 0 and 15 with both the extremes included i.e.

0 <= DIVADDVAL <= 15.

In order to get a baud rate of 9600 (which is the most common baud rates) at a peripheral clock frequency of 60 MHz, the following calculations must be followed.

Try with DLM (U0DLM) = 0, DIVADDVAL = 0, MULVAL = 1. By substituting these values in above formula, then we get DLL (U0DLL) as 390.625. Since U0DLL is an 8 – bit register, this value is out of range.

By continuing this trial and error method, we end up at the following settings in order to get a baud rate of 9600 (approximately) with PCLK being 60 MHz.

U0DLM = 1

U0DLL = 110

DIVADDVAL = 1

MULVAL = 15

**UART Programming in LPC2148**

We will see the configuring and initialization of the UART0 block in the following example where we send some data to the Computer.

#include <lpc214x.h>

#define New\_Line 0xA

void UARTWrite(char data);

int main(void)

{

char msg[] = { ‘E’,’l’,’e’,’c’,’t’,’r’,’o’,’n’,’i’,’c’,’s’,’ ‘,’H’,’u’,’b’,’\0′ };

int c=0; // counter

/\* Configuring PLL for generation of 60 MHz CCLK and PCLK \*/

PLL0CON = 0x01;

PLL0CFG = 0x23;

PLL0FEED = 0xaa;

PLL0FEED = 0x55;

while(!(PLL0STAT & 0x00000400));

PLL0CON = 0x03;

PLL0FEED = 0xaa;

PLL0FEED = 0x55;

VPBDIV = 0x01;

/\*Configure and Initialize UART \*/

PINSEL0 = 0x5;  /\*Select TxD for P0.0 and RxD for P0.1\*/

U0LCR = 0x43; /\*8 data bits, no Parity, 1 Stop bit | DLAB is set to 1\*/

U0DLL = 0x6e; /\* DLL set to 110\*/

U0DLM = 0x01; /\* DLM set to 1\*/

U0FDR = 0xf1; /\* MULVAL bits – 7:4) is set to 15 and DIVADDVAL (bits – 3:0) is set to 1\*/

U0LCR &= 0x0f; // Set DLAB=0 to lock MULVAL and DIVADDVAL

while (1)

{

while( msg[c]!=’\0′ )

{

UARTWrite(msg[c]);

c++;

}

UARTWrite(New\_Line);

c=0;

}

}

void UARTWrite(char data)

{

while ( !(U0LSR & 0x02 ) ); // wait till the THR is empty

U0THR = data;

}