# LPC2148 UART Programming

In this tutorial, we are going to discuss the LPC2148 UART communication. LPC2148 has two inbuilt USARTs. We are going to discuss the only UART0. After this tutorial, you should be able to extend it to UART1. After understating the basics of LPC2148 UART module, We will discuss how to use the ExploreEmbedded libraries to communicate with any of the UART devices.

# UART module

UART module and registers. LPC2148 has 2-UARTs numbering 0-3, similarly, the pins are also named as RXD0-RXD1 and TXD0-TXD1.As the LPC2148 pins are multiplexed for multiple functionalities, first they have to be configured as UART pins.  
Below table shows the multiplexed UART's pins.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Port Pin | Pin Number | PINSEL\_FUNC\_0 | PINSEL\_FUNC\_1 | PINSEL\_FUNC\_2 | PINSEL\_FUNC\_3 |
| P0.0 | 19 | GPIO | **TXD0** | PWM1 |  |
| P0.1 | 21 | GPIO | **RXD0** | PWM3 | EINT0 |
| P0\_8 | 33 | GPIO | **TXD1** | PWM4 | AD1.1 |
| P0.9 | 34 | GPIO | **RXD1** | PWM6 | EINT3 |

# UART Registers

The below table shows the registers associated with LPC2148 UART.

|  |  |
| --- | --- |
| Register | Description |
| UxRBR | Contains the recently received Data |
| UxTHR | Contains the data to be transmitted |
| UxFCR | FIFO Control Register |
| UxLCR | Controls the UART frame formatting(Number of Data Bits, Stop bits) |
| UxDLL | Least Significant Byte of the UART baud rate generator value. |
| UxDLM | Most Significant Byte of the UART baud rate generator value. |

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

# UART Register Configuration

Now lets see how to configure the individual registers for UART communication.

## UxFCR ( FIFO Control Register )

LPC2148 has inbuilt 16byte FIFO for Receiver/Transmitter. Thus it can store 16-bytes of data received on UART without overwriting. If the data is not read before the Queue(FIFO) is filled then the new data will be lost and the OVERRUN error bit will be set.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| UxFCR |  |  |  |  |  |
| 31:8 | 7:6 | 5:3 | 2 | 1 | 0 |
| RESERVED | RX TRIGGER | RESERVED | TX FIFO RESET | RX FIFO RESET | FIFO ENABLE |

**Bit 0 – FIFO:**  
This bit is used to enable/disable the FIFO for the data received/transmitted.  
0--FIFO is Disabled.  
1--FIFO is Enabled for both Rx and Tx.

**Bit 1 – RX\_FIFO:**  
This is used to clear the 16-byte Rx FIFO.  
0--No impact.  
1--CLears the 16-byte Rx FIFO and the resets the FIFO pointer.

**Bit 2 – Tx\_FIFO:**  
This is used to clear the 16-byte Tx FIFO.  
0--No impact.  
1--Clears the 16-byte Tx FIFO and the resets the FIFO pointer.

**Bit 7:6 – Rx\_TRIGGER:**  
This bit is used to select the number of bytes of the receiver data to be written so as to enable the interrupt/DMA.  
00-- Trigger level 0 (1 character or 0x01)  
01-- Trigger level 1 (4 characters or 0x04)  
10-- Trigger level 2 (8 characters or 0x08)  
11-- Trigger level 3 (14 characters or 0x0E)

## UxLCR ( Line Control Register )

This register is used for defining the UART frame format ie. Number of Data bits, STOP bits etc.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| UxLCR |  |  |  |  |  |  |
| 31:8 | 7 | 6 | 5:4 | 3 | 2 | 1:0 |
| Reserved | DLAB | Break Control | Parity Select | Parity Enable | Stop Bit Select | Word Length Select |

**Bit 1:0 – WLS : WordLenghtSelect**  
These two bits are used to select the character length  
00-- 5-bit character length  
01-- 6-bit character length  
10-- 7-bit character length  
11-- 8-bit character length

**Bit 2 – Stop Bit Selection:**  
This bit is used to select the number(1/2) of stop bits   
0-- 1 Stop bit  
1-- 2 Stop Bits

**Bit 3 – Parity Enable:**  
This bit is used to Enable or Disable the Parity generation and checking.  
0-- Disable parity generation and checking.  
1-- Enable parity generation and checking.

**Bit 5:4 – Parity Selection:**  
These two bits will be used to select the type of parity.  
00-- Odd parity. Number of 1s in the transmitted character and the attached parity bit will be odd.  
01-- Even Parity. Number of 1s in the transmitted character and the attached parity bit will be even.  
10-- Forced "1" stick parity.  
11-- Forced "0" stick parity

**Bit 6 – Break Control**  
0-- Disable break transmission.  
1-- Enable break transmission. Output pin UARTn TXD is forced to logic 0

**Bit 8 – DLAB: Divisor Latch Access Bit**  
This bit is used to enable the access to divisor latch.  
0-- Disable access to divisor latch  
0-- Enable access to divisor latch

## UxLSR (Line Status Register)

The is a read-only register that provides status information of the UART TX and RX blocks.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| UxLSR |  |  |  |  |  |  |  |  |
| 31:8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Reserved | RXFE | TEMT | THRE | BI | FE | PE | OE | RDR |

**Bit 0 – RDR: Receive Data Ready**  
This bit will be set when there is a received data in RBR register. This bit will be automatically cleared when RBR is empty.

0-- The UARTn receiver FIFO is empty.  
1-- The UARTn receiver FIFO is not empty.

**Bit 1 – OE: Overrun Error**  
The overrun error condition is set when the UART Rx FIFO is full and a new character is received. In this case, the UARTn RBR FIFO will not be overwritten and the character in the UARTn RSR will be lost.  
0-- No overrun  
1-- Buffer over run

**Bit 2 – PE: Parity Error**  
This bit is set when the receiver detects a error in the Parity.  
0-- No Parity Error  
1-- Parity Error

**Bit 3 – FE: Framing Error**  
This bit is set when there is error in the STOP bit(LOGIC 0)  
0-- No Framing Error  
1-- Framing Error

**Bit 4 – BI: Break Interrupt**  
This bit is set when the RXDn is held in the spacing state (all zeroes) for one full character transmission  
0-- No Break interrupt  
1-- Break Interrupt detected.

**Bit 5 – THRE: Transmitter Holding Register Empty**  
THRE is set immediately upon detection of an empty THR. It is automatically cleared when the THR is written.  
0-- THR register is Empty  
1-- THR has valid data to be transmitted

**Bit 6 – TEMT: Transmitter Empty**  
TEMT is set when both UnTHR and UnTSR are empty; TEMT is cleared when any of them contain valid data.  
0-- THR and/or the TSR contains valid data.  
1-- THR and the TSR are empty.

**Bit 7 – RXFE: Error in Rx FIFO**  
This bit is set when the received data is affected by Framing Error/Parity Error/Break Error.  
0-- RBR contains no UARTn RX errors.  
1-- RBR contains at least one RX error.

## UxTER (Transmitter Enable register)

This register is used to Enable/Disable the transmission

|  |  |  |
| --- | --- | --- |
| UxTER |  |  |
| 31:8 | 7 | 6-0 |
| Reserved | TXEN | Reserved |

**Bit 7 – TXEN: Trsnamitter Enable**When this bit is 1, the data written to the THR is output on the TXD pin.  
If this bit is cleared to 0 while a character is being sent, the transmission of that character is completed, but no further characters are sent until this bit is set again.  
In other words, a 0 in this bit blocks the transfer of characters.

* Note: By default this bit will be set after Reset.

## Baudrate Calculation

LPC1768 generates the baud rate depending on the values of DLM,DLL.  
**Baudrate = PCLK/ (16 \* ( 256 \* DLM + DLL) \* (1+ DivAddVal/MulVal))**

DivAddVal/MulVal == 0

Using the above parameters , DLL/DLM is calculated as below.  
**(256 \* DLL + DLM) = PCLK / (16\* Baudrate).**

# Steps for Configuring UART0

Below are the steps for configuring the UART0.

1. .Configure the GPIO pin for UART0 function using PINSEL register.
2. .Configure the FCR for enabling the FIXO and Reste both the Rx/Tx FIFO.
3. .Configure LCR for 8-data bits, 1 Stop bit, Disable Parity and Enable DLAB.
4. .Calculate the DLM,DLL values for required baudrate from PCLK.
5. .Update the DLM,DLL with the calculated values.
6. .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

|  |
| --- |
| void uart\_init(uint32\_t baudrate) |
| { |
| uint32\_t var\_RegValue\_u32; |
|  |
| PINSEL0 |= (1<<RX0\_PIN) | (1<<TX0\_PIN); //Configure P0.0/P0.1 as RX0 and TX0 |
|  |
| U0FCR = (1<<SBIT\_FIFO) | (1<<SBIT\_RxFIFO) | (1<<SBIT\_TxFIFO); // Enable FIFO and reset Rx/Tx FIFO buffers |
| U0LCR = (0x03<<SBIT\_WordLenght) | (1<<SBIT\_DLAB); // 8bit data, 1Stop bit, No parity |
|  |
| var\_RegValue\_u32 = ( pclk / (16 \* baudrate )); |
| U0DLL = var\_RegValue\_u32 & 0xFF; |
| U0DLM = (var\_RegValue\_u32 >> 0x08) & 0xFF; |
|  |
| util\_BitClear(U0LCR,(SBIT\_DLAB)); // Clear DLAB after setting DLL,DLM |

}

# Steps for transmitting a char

1. Step1: Wait till the previous char is transmitted ie. till THRE becomes high.
2. Step2: Load the new char to be transmitted into THR.

|  |
| --- |
| void uart\_TxChar(char ch) |
| { |
| while(util\_IsBitCleared(U0LSR,SBIT\_THRE)); // Wait for Previous transmission |
| U0THR=ch; // Load the data to be transmitted |

}

# Steps for Receiving a char

1. Step1: Wait till the a char is received ie. till RDR becomes high.
2. Step2: Copy the received data from receive buffer(RBR).

**Code snippet**

char uart\_RxChar()

{

char ch;

while(util\_IsBitCleared(U0LSR,SBIT\_RDR)); // Wait till the data is received

ch = U0RBR; // Read received data

return ch;

}

**Below is the code for transmitting and receiving chars at 9600 baud**

#include <lpc214x.h>

#include "stdutils.h"

#include "systemInit.h"

#define SBIT\_WordLenght 0x00u

#define SBIT\_DLAB 0x07u

#define SBIT\_FIFO 0x00u

#define SBIT\_RxFIFO 0x01u

#define SBIT\_TxFIFO 0x02u

#define SBIT\_RDR 0x00u

#define SBIT\_THRE 0x05u

#define TX0\_PINSEL 0

#define RX0\_PINSEL 2

/\* Function to initialize the UART0 at specifief baud rate \*/

void uart\_init(uint32\_t baudrate)

{

uint32\_t var\_RegValue\_u32;

PINSEL0 |= (1<<RX0\_PINSEL) | (1<<TX0\_PINSEL); //Configure P0.0/P0.1 as RX0 and TX0

U0FCR = (1<<SBIT\_FIFO) | (1<<SBIT\_RxFIFO) | (1<<SBIT\_TxFIFO); // Enable FIFO and reset Rx/Tx FIFO buffers

U0LCR = (0x03<<SBIT\_WordLenght) | (1<<SBIT\_DLAB); // 8bit data, 1Stop bit, No parity

var\_RegValue\_u32 = ( pclk / (16 \* baudrate ));

U0DLL = var\_RegValue\_u32 & 0xFF;

U0DLM = (var\_RegValue\_u32 >> 0x08) & 0xFF;

util\_BitClear(U0LCR,(SBIT\_DLAB)); // Clear DLAB after setting DLL,DLM

}

/\* Function to transmit a char \*/

void uart\_TxChar(char ch)

{

while(util\_IsBitCleared(U0LSR,SBIT\_THRE)); // Wait for Previous transmission

U0THR=ch; // Load the data to be transmitted

}

/\* Function to Receive a char \*/

char uart\_RxChar()

{

char ch;

while(util\_IsBitCleared(U0LSR,SBIT\_RDR)); // Wait till the data is received

ch = U0RBR; // Read received data

return ch;

}

int main()

{

char ch,a[]="\n\rExploreEmbedded";

int i;

SystemInit();

uart\_init(9600); // Initialize the UART0 for 9600 baud rate

uart\_TxChar('h'); //Transmit "hello" char by char

uart\_TxChar('e');

uart\_TxChar('l');

uart\_TxChar('l');

uart\_TxChar('o');

for(i=0;a[i];i++) //transmit a predefined string

uart\_TxChar(a[i]);

while(1)

{

//Finally receive a char and transmit it infinitely

ch = uart\_RxChar();

uart\_TxChar(ch);

}

}

### Using Explore Embedded Libraries

In the above tutorial, we discussed how to configure and use the inbuilt LPC2148 UART.  
Now we will see how to use the ExploreEmbededd UART libraries to communicate on all the four UART channels.  
For this you have to include the uart.c/uart.h files and associated gpio/stdutils files.

As LPC1768 has four inbuilt UART channels, the interfaces are suffixed with channel number as shown below.  
UART0\_Printf()  
UART1\_Printf()

1. Note:Refer the uart.h file for more info

#include "uart.h"

#include "stdutils.h"

#include "systemInit.h"

int main()

{

SystemInit();

/\* Initialize both the UARTs with different Baud rate \*/

UART0\_Init(9600);

UART1\_Init(19200);

while(1)

{

UART0\_Printf("Welcome to LPC2148 UART Programming on channel Zero at 9600 baud\n\r");

UART1\_Printf("Welcome to LPC2148 UART Programming on channel One at 19200 baud\n\r");

}

}

# Testing

## Using the Terminal Software

After generating the hex/bin file, flash it to the controller. Now connect the LPC1768 to your system using a Usb to Serial converter.

