Application Note AN16002

Programming Guide

For SystemBase's SB16C105x UART

SystemBase Co., Ltd.

Document Information

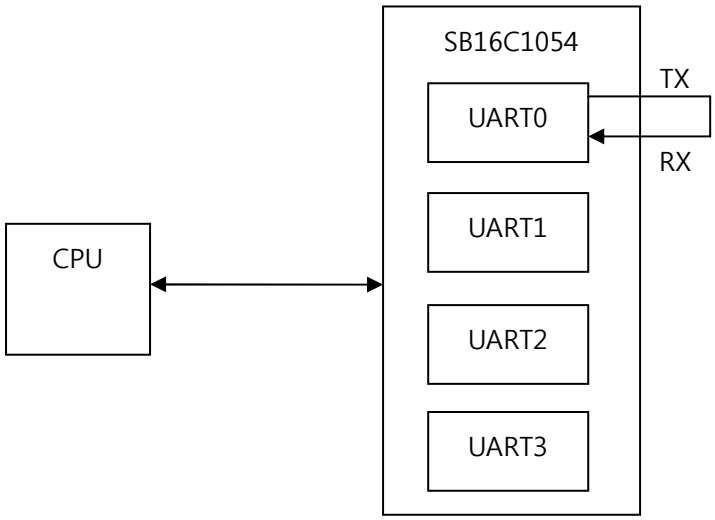
Information	Content
Abstract	This document provides a guideline for programming SystemBase's SB16C105x UARTs.
Revision History	V1.0, Written by Hwanjun Noh @ 26th May 2009

SystemBase

1. Introduction

UARTs are one of simple logics of IC. Still, people who use them for the first time have difficulties in using them. This paper will provide a brief instruction for setting and testing UART assuming SystemBase SB16C1054 is used.

Usually 8051 or ARM CPU is used to control UART. We will be testing our UART by initializing the UART and transceiving data through CPU data bus. In this paper, we will use one of the four UARTs to send/receive data as below.



2. Basic Techniques

2.1 UART Initialization

1. Set LCR[7] = 1 to enable DLAB. This allows DLL, DLM Registers to be accessed.
2. Set baud rate by configuring DLL, DLM.
3. Set LCR[7] = 0, so normal registers can be accessed.
4. Set LCR to configure serial data communication frame.
5. Clear FIFO with FCR.
6. Enable FIFO with FCR. Set FIFO trigger level if necessary.
7. Enable interrupts using IER.
8. Start data communication.

2.2 Interrupts

When interrupts occur, interrupts can be read from IIR. After resolving the interrupt, you need read the IIR again to read interrupt with next highest priority until all interrupts are resolved

through ISR(Interrupt Service Routine).

2.3 Registers

RBR and THR both have the same address '0'. However, you should notice that their access permissions are different. When you read address '0', you access RBR but when you write into address '0', you access THR. Also, DLL and DLM can only be accessed when LCR[7] = 1. When LCR[7] = 0, RHR, THR and IER are accessed.

3. Programmer's Guide

The base set of registers that is used during high-speed data transfer has a straightforward access method. The extended function registers require special access bits to be decoded along with the address lines. The following guide will help with programming these registers. Note that the descriptions below are for individual register access. Some streamlining through interleaving can be obtained when programming all the registers.

Command	Action
Set Baud Rate to VALUE1, VALUE2	Read LCR, save in temp Set LCR to 80h Set DLL to VALUE1 Set DLM to VALUE2 Set LCR to temp
Set Xon1, Xoff1 to VALUE1, VALUE2	Read LCR, save in temp Set LCR to BFh Set Xon1 to VALUE1 Set Xoff1 to VALUE2 Set LCR to temp
Set Xon2, Xoff2 to VALUE1, VALUE2	Read LCR, save in temp Set LCR to BFh Set Xon2 to VALUE1 Set Xoff2 to VALUE2 Set LCR to temp
Set Software Flow Control Mode to VALUE	Read LCR, save in temp Set LCR to BFh Set EFR to VALUE Set LCR to temp
Set flow control threshold for 64-byte FIFO Mode	1) Set FCR to '0000_xxx1' → Set FUR to 8, set FLR to 0 2) Set FCR to '0101_xxx1' → Set FUR to 16, set FLR to 8 3) Set FCR to '1010_xxx1' → Set FUR to 56, set FLR to 16 4) Set FCR to '1111_xxx1'

	→ Set FUR to 60, set FLR to 56
Set flow control threshold for 256-byte FIFO Mode	Set FCR to 'xxxx_xxx1' Read LCR, save in temp Set LCR to BFh Set PSR to A5h Set AFR to 01h Set FUR to Upper Threshold Value Set FLR to Lower Threshold Value Set PSR to A4h Set LCR to temp
Set TX FIFO / RX FIFO Interrupt Trigger Level for 64-byte FIFO Mode	1) Set FCR to '0000_xxx1' → Set RTR to 8, set TTR to 8 2) Set FCR to '0101_xxx1' → Set RTR to 16, set TTR to 16 3) Set FCR to '1010_xxx1' → Set RTR to 56, set TTR to 32 4) Set FCR to '1111_xxx1' → Set RTR to 60, set TTR to 56
Set TX FIFO / RX FIFO Interrupt Trigger Level for 256-byte FIFO Mode	Set FCR to 'xxxx_xxx1' Read LCR, save in temp Set LCR to BFh Set PSR to A5h Set AFR to 01h Set TTR to TX FIFO Trigger Level Value Set RTR to RX FIFO Trigger Level Value Set PSR to A4h Set LCR to temp
Read Flow Control Status	Read LCR, save in temp1 Read MCR, save in temp2 Set LCR to ('0111_1111' **AND** temp1) Set MCR to ('0100_0000' **OR** temp2) Read FSR, save in temp3 Pass temp3 back to host Set MCR to temp2 Set LCR to temp1
Read TX FIFO / RX FIFO Count Value	Read LCR, save in temp1 Read MCR, save in temp2 Set LCR to ('0111_1111' **AND** temp1) Set MCR to ('0100_0000' **OR** temp2) Read TCR, save in temp3 Read RCR, save in temp4 Pass temp3 back to host Pass temp4 back to host Set MCR to temp2 Set LCR to temp1
Read 256-byte TX FIFO Empty Status /	Set FCR to 'xxxx_xxx1' Read LCR, save in temp1

RX FIFO Full Status	Set LCR to BFh Set PSR to A5h Set AFR to 01h Set PSR to A4h Set LCR to temp1 Read ISR, save in temp2 Pass temp2 back to host
Enable Xoff Re-transmit	Read LCR, save in temp1 Set LCR to not BFh Read MCR, save in temp2 Set MCR to ('0100_0000' **OR** temp2) Set MCR to ('0100_0100' **OR** temp2) Set MCR to ('1011_1111' **AND** temp2) Set MCR to temp2 Set LCR to temp1
Disable Xoff Re-transmit	Read LCR, save in temp1 Set LCR to not BFh Read MCR, save in temp2 Set MCR to ('0100_0000' **OR** temp2) Set MCR to ('1011_1011' **AND** temp2) Set MCR to temp2 Set LCR to temp1

Set Prescaler Value to Divide-by-1 or 4	Read LCR, save in temp1
	Set LCR to BFh
	Read EFR, save in temp2
	Set EFR to ('0001_0000' **OR** temp2)
	Set LCR to 00h
	Read MCR, save in temp3
	
	if Divide-by-1 = OK then
	Set MCR to ('0111_1111' **AND** temp3)
	else
	Set MCR to ('1000_0000' **OR** temp3)
	
	Set LCR to BFh
	Set EFR to temp2
	Set LCR to temp1

4. Sample Code

4.1 Interrupt Driven Program

The sample code illustrates a simple interrupt driven program. The program will receive input from keyboard and send the received data. The received data will be handled in ISR function when the interrupt occurs.

```
#define <stdio.h>
```

```
#define <conio.h>

#define COM1    0x03f8
#define INTVEC  0x0c

#define THR     0
#define RBR     0
#define IER     1
#define ISR     2
#define LCR     3
#define LSR     5

#define DLL     0
#define DLM     1

void interrupt (*oldportisr) ();

void interrupt ISR(void)
{
        int c;
        int data;

        //Here, we're assuming that the interrupts only occur when received data is available. To
        //handle more interrupts, we would need to read the IIR and handle them accordingly.

        //int interrupt

        //interrupt = inportb(COM1 + ISR);
        //switch(interrupt){
        //      case 0x02:
        //              Action;
        //              break;
        //      case 0x04:
        //              Action;
        //              break;
        //      ...
        //}

        do{
```

```
                //First we check LCR to see if data is available
                c = inportb(COM1 + LCR)
                if(c & 1){
                        //The data is available. So we read RBR and print it
                        data = inportb(COM1 + RBR);
                        printf("Received Data: %d", data);
                }
        } while (c & 1)

        outportb(0x20,0x20);
}

void main(void)
{
        int c;

        //Once we know the IRQ the next step is to find it's interrupt vector or software interrupt
        //as some people may call it. Basically any 8086 processor has a set of 256 interrupt
        //vectors numbered 0 to 255. Each of these vectors contains a 4 byte code which is an
        //address of the Interrupt Service Routine (ISR). Fortunately C being a high level language,
        //takes care of the addresses for us. All we have to know is the actual interrupt vector.
        //INT   IRQ     Common Uses
        // 0B     3      Serial Comms. COM2/COM4
        // 0C     4      Serial Comms. COM1/COM3

        //Save old interrupt vector so it can be restored in the end
        oldportisr = getvect(INTVECT);
        // Set up the vector using so function ISR can lead us to a set of instructions which
        // would service the interrupt.
        setvect(INTVEC, ISR);
        //Selects which interrupts we want to disable
        //The Programmable Interrupt Controller handles hardware interrupts. Most PC's will have
        //two of them located at different addresses. One handles IRQ's 0 to 7 and the other
        //IRQ's 8 to 15. Mainly Serial communications interrupts reside on IRQ's under 7, thus
        //PIC1 is used, which is located at 0x20. Masking is done in 0x21.
        //0xEF is 11101111b. The highest bit shows IRQ 7 while the lowest bit shows IRQ 0.
        //Writing 0xEF would mean that we would disable all IRQs except IRQ 4.
        //In case some other IRQs have been enabled already, we should read the PIC and keep
        //them enabled as well. This could be done by reading the PIC and & them with 0xEF.
```

```
//Look for PIC for more information.
outportb(0x21, (inportb(0x21) & 0xEF));

//This will set baud rate.
//First we enable DLAB by setting LCR[7] = 0
//Then we set DLM and DLL to appropriate value to set desired baud rate.
outportb(COM1 + LCR, 0x80);
outportb(COM1 + DLM, 0x00);
outportb(COM1 + DLL, 0x02);

//We set LCR to set communication frame.
//8 Bits, No Parity, 1 Stop Bit.
outportb(COM1 + LCR, 0x03);

//Reset FIFO and enable FIFO.
outportb(COM1 + FCR, 0x07);

/*
//This code illustrates how to set 256-byte FIFO.
//Enable Page 3,4
outportb(COM1 + LCR, 0xBF); //enable page 3,4
//Select Page 4
outportb(COM1 + PSR, 0xA5);
//Enable Deep FIFO
outportb(COM1 + AFR, 0x01);
//Set FIFO Trigger Levels
//TX trigger level 128
outportb(COM1 + TTR, 0x80);
//RX trigger level 255
outportb(COM1 + RTR, 0xFF);
*/

//Enable Receive Interrupt
outportb(COM1 + IER, 0x01);

//We send the input data until ESC is pressed
//That is it for the main function.
//The received data will be handled by ISR function
do {
```

```
                if (kbhit()){
                        c = getch();
                        outportb(PORT1, c);
                        }
                outportb(COM1 + THR, c);
        } while (c != 27)

        //Disable interrupt
        outportb(COM1 + IER, 0x00);
        //Mask IRQ
        outportb(0x21, (inportb(0x21) | 0x10));
        //Restore interrupt vector
        setvect(INTVECT, oldisr);
}
```

4.2 Using Global Interrupts

As there are four independent 1-channel UARTs in SB16C1054, there are four interrupts. Interrupts are assigned INT0, INT1, INT2, and INT3 for each channel. Each interrupt has six prioritized level's interrupt generation capability. The IER enables each of the six types of interrupts and INT signal in response to an interrupt generation. When an interrupt is generated, the ISR indicates that an interrupt is pending and provides the type of interrupt. SB16C1054 can handle for four interrupts with one global interrupt. Global interrupt treats four of each interrupt as one interrupt, so it is useful when external system has few interrupt resource.

Add following code to use Global Interrupt.

```
#define PSR     0
#define AFR     1
#define GICR    1
#define GISR    2
#define MCR     4

void main(void)
{

        ...
        ...
        //First we set GICR[0] = 1 to enable Global Interrupt.
        //To Access GICR, we must set LCR[7] = 0, MCR[6] = 1. This is written in the Datasheet.
```

```
        //Note that we didn't want to alter the contents of MCR other than MCR[6].
        outportb(COM1 + LCR, 0x80);
        outportb(COM1 + MCR, inportb(COM1 + MCR) | 0x40);
        //Set GICR[0] = 1 to enable Global Interrupt.
        outportb(COM1 + GICR, 0x01);
        //MCR[6] = 0
        outportb(COM1 + MCR, inportb(COM1 + MCR) & 0xBF);
        //When Global Interrupt is used, INT0 will be used as GINT. To do this, we set AFR[4] = 1.
        //To Access AFR, we set LCR = 0xBF and PSR[0] = 1.
        outportb(COM1 + LCR, 0xBF);
        outportb(COM1 + PSR, 0xA5);
        //We set AFR[4] = 1.
        //Also, we set Global Interrupt Polarity with AFR[5] = 1. This means, GINT pin outputs '1'
        //when interrupt is generated
        outportb(COM1 + AFR, 0x30);
        …
        …
}

void interrupt ISR(void)
{
        //When interrupt is generated through GINT, we should find out which of the four UARTs
        //generated the interrupt. To do this, we read GISR.
        int InterruptNumber;

        //Reading GISR requires same procedure as reading writing GICR.
        outportb(COM1 + LCR, 0x80);
        outportb(COM1 + MCR, inportb(COM1 + MCR) | 0x40);
        //Read GISR
        InterruptNumber = inportb(COM1 + GISR);
        outportb(COM1 + MCR, inportb(COM1 + MCR) & 0xBF);

        //Depending on which UART generated the interrupt, we would implement different logic
        if(InterruptNumber & 0x01){
        }
        else if(InterruptNumber & 0x02){
        }
        else if(InterruptNumber & 0x04){
        }
```

```
        else if(InterruptNumber & 0x08){
        }
        ...
        ...
}
```