

Universal Asynchronous Receiver/Transmitter (UART) on the LM3S8962

David DeLuca
11/08/2009
ECE 480 DT3

Keywords:

UART, Baud, Interrupt, FIFO, Transmitter, Receiver, Register

Abstract

In order for serial communication to take place in modern computers, peripheral devices and microcontrollers, a Universal Asynchronous Receiver/Transmitter (UART) is utilized. Due to its sequential processing of data, cost savings by avoiding using multiple lines for parallel transmissions is evident. As a critical component in a communication process, UART's can be programmed for proper timing sequences and error checking through parity bits. This allows the designer a certain amount of flexibility when programming register values.

Introduction

Although UART's have been around since the 1970's, its importance has not diminished over time. With speed and cost at the core of modern day communication, UART continues to boast its usefulness with and without electrical wires, including optical fiber, infrared (IrDA) and bluetooth signals. As the length of a cable extends, more cost is incurred. Sending data over parallel lines involves using multiple wires for one transmission, which is particularly expensive and most likely not a practical solution. However, by using a UART controller, data can be translated between serial and parallel forms. The UART takes all the bytes of data and sequentially transmits it one bit at a time. At the destination, an additional UART controller will receive the individual bits and finally reassemble the transmission into its complete form. Due to its success over the years, many modern microcontrollers such as the LM3S8962 by Texas Instruments have multiple native UART's on a board. The following pages will discuss proper programming of registers, explanation of terms, and different methods of operation for this microcontroller.

Objectives

- Provide details on how the UART works and various applications of use
- Define common terminology
- Handling of UART registers and **First in First Out** (FIFO) operation
- Sample coding to communicate between LM3S8962, MAX232, and PC

Baud-Rate

The SI unit Baud was named after Emile Baudot and is synonymous with pulses per second or symbols per second. Baud rate is expressed as the number of times a signal can change logical states per second, meaning the higher the baud rate the faster the communication channel is. In order for proper communication between sender and receiver, the baud-rate should be programmed at each location to the same value. This is easily achieved with modern UART's on microcontrollers due to Baud-rate divisors. In the LM328962, Baud-Rate generation is done through a 22-bit number that serves as the divisor. 16 of the bits are an integer, and the last six bits are for fractional adjustment. Having the six fractional bits allows for generation of all standard baud rates. Knowledge of the entire 22 bit number will allow the Baud-Rate generator to determine the bit period.

The baud-rate divisor can be defined by the following equation in the LM3S8962:

$$BRD = BRD_I + BRD_F = UARTSysClk / (16 * Baud Rate)$$

Formula 1: Baud-Rate Divisor

where BRD_I is the 16-bit integer, BRD_F is the 6-bit fraction, and UARTSysClk is the system clock connected to the UART. The system clock is divided by 16 to generate the transmit clock which will serve as error detection during receive operations. Both BRD_I and BRD_F are

stored in the UART register *UARTFBRD*, while the baud-rate divisor is located in register *UARTIBRD*. A list of all the UART registers in the LM3S8962 is located in the Appendix.

Data Transmission

All data received or transmitted must be temporarily stored so that only one bit at a time is handled. To accommodate for this, the LM3S8962 utilizes two 16-byte **First in First Out** (FIFO) buffers (typically a receiving FIFO may contain extra bits per character for status information). Both FIFOs can be accessed via the *UARTDR* register. The transmit logic performs all parallel-to-serial conversion on the data read in from the transmit FIFO. Conversely, the receive logic performs all serial-to-parallel conversion on the received bit stream. The control logic outputs the serial stream of bits beginning with a defined ‘start’ bit followed by the least significant bit (LSB) of the data stream, the parity bit (if error checking is enabled) and finally stop bits. Figure 1 demonstrates this process and the typical number of bits

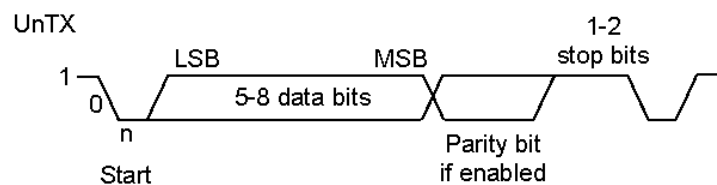


Figure 1. UART Character Frame

per function in a serial bit stream.

When UART is enabled, the *UARTLCRH* register will be searched for all necessary parameters before transmitting a data frame. Since a buffer is being used, the data will continue to transmit if the *UARTFR* flag register indicates the FIFO is non-empty. Having the ability to check both the flag register and the data register is a very beneficial tool to the programmer and allows for ease in debugging.

IrDA Serial Infrared (SIR)

The UART controller on the LM3S8962 includes an IrDA serial-Infrared (SIR) encoder/decoder block. This SIR provides the ability to convert an asynchronous UART data stream and a half-duplex serial SIR interface. The main role of this SIR encoder/decoder block is to provide a digitally encoded output with a digitally decoded input to the UART. The SIR block has two operating modes: normal IrDA mode and low-power IrDA mode. Operating in normal IrDA mode constitutes transmitting a logical '0' as a high pulse for $3/16^{\text{th}}$ the duration of the baud-rate bit period on the output pin while logical '1' is transmitted statically as a '0'. Operating in low-power IrDA mode adjusts the width of the infrared pulse transmitted to three times the period of the internally generated baud clock (nominally 1.432 MHz). Register four of the UART registers is the IrDA low-power register which contains the low-power counter divisor defined by the following formula:

$$ILPDVSR = SysClk / FIrLPBaud16$$

Formula 2: Low-Power Counter Divisor

where FIrLPBaud16 is the internally generated baud clock. To operate in low-power IrDA mode, the divisor should be in the range of: $1.42 \text{ MHz} < FIrLPBaud16 < 2.12 \text{ MHz}$. Functioning in this range results in a pulse width of $1.41 - 2.11 \mu\text{s}$ which is three times the period of FIrLPBaud16. Figure 2 illustrates the difference between data signals transmitted/received using UART with and without IrDA data modulation.

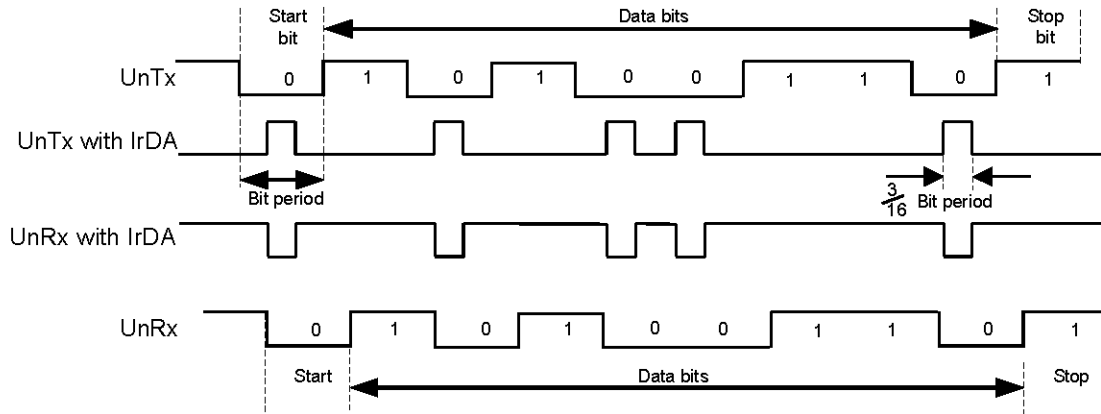


Figure 2. IrDA Data Modulation

Interrupts

Interrupts in a UART are very critical due to high probability of transmission errors in a communication system. The LM3S8962's UARTs can generate the following interrupts:

- Overrun Error
- Break Error
- Parity Error
- Framing Error
- Receive Timeout

Since the possibility of multiple interrupts occurring at one given time exists, all UART interrupt events are connected to an OR gate so that only a single interrupt event is sent to the interrupt controller at a time. The UART interrupt mask *UARTIM* defines all interrupt events that can trigger a controller-level interrupt request. When an interrupt event is handled, the event can be cleared by placing the correct bit in the *UARTICR* interrupt clear register.

Initialization/Configuration

In order to use the UARTs in the LM3S8962, the peripheral clock must be enabled by either setting the UART0 or UART1 bits in the Run Mode Clock Gating Control Register 1 (*RCGC1*). After we have set the peripheral clock, the baud-rate divisor must be programmed next. This is critical as both the *UARTIBRD* and *UARTFBRD* registers require initialization prior to the *UARTLCRH* register. Using the baud-rate generation formula (Formula 1) with a target baud-rate of 115,200, the baud-rate divisor (BRD) can be calculated to be approximately ≈ 10.8507 . Using this value, we can then calculate value to be stored in the *UARTFBRD* register as:

$$UARTFBRD[DIVFRAC] = \text{integer}(0.8507 * 64 + 0.5) = 54$$

Formula 3

where .8507 is the BRD_F and .5 is used to account for rounding error. Now that we have the BRD values calculated, the UART configuration is completed in the following sequential order:

1. Disabling the UART
2. Write integer portion of BRD to *UARTIBRD*
3. Write fractional portion of BRD to *UARTIFBRD*
4. Write desired serial parameters to *UARTLCRH* (0x0000.0060 in above case)
5. Enabling UART by setting URTEN bit in *UARTCTL*

Sample Code

The following fragment of code demonstrates basic communication between the LM3S8962 and Texas Instruments MAX232 Driver/Receiver chip. In order for communication to be read in or sent from a computer, a DB9 connector can be used to connect a serial cable to the breadboard containing the MAX232 chip.

```

//*****
**
//
// Send a string to the UART.
//
//*****
**
void
UARTSend(const unsigned char *pucBuffer, unsigned long ulCount)
{
    //
    // Loop while there are more characters to send.
    //
    while(ulCount--)
    {
        //
        // Write the next character to the UART.
        //
        UARTCharPutNonBlocking(UART0_BASE, *pucBuffer++);
    }
}

//*****
**
//
// This example demonstrates how to send a string of data to the UART.
//
//*****
**
int
main(void)
{
    //
    // Set the clocking to run directly from the crystal.
    //
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |
                    SYSCTL_XTAL_8MHZ);

    //
    // Initialize the OLED display and write status.
    //
    RIT128x96x4Init(1000000);

    //
    // Enable the peripherals used by this example.
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    //
    // Enable processor interrupts.
    //
    IntMasterEnable();

```

```

//
// Set GPIO A0 and A1 as UART pins.
//
GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

//
// Configure the UART for 115,200, 8-N-1 operation.
//
UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
                    (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                     UART_CONFIG_PAR_NONE));

//
// Enable the UART interrupt.
//
IntEnable(INT_UART0);
UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);

//
// Prompt for text to be entered.
//
// UARTSend((unsigned char *)"T", 1);
while(1)
{
    while(!UARTCharsAvail(UART0_BASE))
    {
    }
    while(UARTCharsAvail(UART0_BASE))
    {
        char tmp = UARTCharGet(UART0_BASE);
        RIT128x96x4StringDraw("M:", 12, 58, 15);
        RIT128x96x4StringDraw(&tmp, 20, 58, 15);
        UARTCharPut(UART0_BASE, 'h');
    }
    //
    // Loop forever echoing data through the UART.
    //
}
}

```

As you can see in the last set of while loops, you can continuously check to see if a character becomes available on the FIFO. When a character is found and the UART detects transmission, we can draw the data onto the LM3S8962's built-in OLED.

Appendix

Offset	Name	Type	Reset	Description
0x000	UARTDR	R/W	0x0000.0000	UART Data
0x004	UARTSR/UARTECR	R/W	0x0000.0000	UART Receive Status/Error Clear
0x018	UARTFR	RO	0x0000.0090	UART Flag
0x020	UARTILPR	R/W	0x0000.0000	UART IrDA Low-Power Register
0x024	UARTIBRD	R/W	0x0000.0000	UART Integer Baud-Rate Divisor
0x028	UARTFBRD	R/W	0x0000.0000	UART Fractional Baud-Rate Divisor
0x02C	UARTLCRH	R/W	0x0000.0000	UART Line Control
0x030	UARTCTL	R/W	0x0000.0300	UART Control
0x034	UARTIFLS	R/W	0x0000.0012	UART Interrupt FIFO Level Select
0x038	UARTIM	R/W	0x0000.0000	UART Interrupt Mask
0x03C	UARTRIS	RO	0x0000.000F	UART Raw Interrupt Status
0x040	UARTMIS	RO	0x0000.0000	UART Masked Interrupt Status
0x044	UARTICR	W1C	0x0000.0000	UART Interrupt Clear
0xFD0	UARTPeriphID4	RO	0x0000.0000	UART Peripheral Identification 4
0xFD4	UARTPeriphID5	RO	0x0000.0000	UART Peripheral Identification 5
0xFD8	UARTPeriphID6	RO	0x0000.0000	UART Peripheral Identification 6
0xFDC	UARTPeriphID7	RO	0x0000.0000	UART Peripheral Identification 7
0xFE0	UARTPeriphID0	RO	0x0000.0011	UART Peripheral Identification 0
0xFE4	UARTPeriphID1	RO	0x0000.0000	UART Peripheral Identification 1
0xFE8	UARTPeriphID2	RO	0x0000.0018	UART Peripheral Identification 2
0xFEC	UARTPeriphID3	RO	0x0000.0001	UART Peripheral Identification 3
0xFF0	UARTPCelIID0	RO	0x0000.000D	UART PrimeCell Identification 0
0xFF4	UARTPCelIID1	RO	0x0000.00F0	UART PrimeCell Identification 1
0xFF8	UARTPCelIID2	RO	0x0000.0005	UART PrimeCell Identification 2
0xFFC	UARTPCelIID3	RO	0x0000.00B1	UART PrimeCell Identification 3

Figure 3. UART Registers LM3S8962

Terminology

RC: Software can read this field. The bit or field is cleared by hardware after reading the bit/field.

RO : Software can read field and write chip reset.

R/W: Software can read/write field

R/W1C: Software can read or write this field. A write of a 0 to a W1C bit does not affect the bit value in the register. A write of a 1 clears the value of the bit in the register; the remaining bits remain unchanged.

R/W1S: Software can read or write a 1 to this field. A write of a 0 to a R/W1S bit does not affect the bit value in the register.

W1C: Software can write this field. A write of a 0 to a W1C bit does not affect the bit value in the register. A write of a 1 clears the value of the bit in the register; the remaining bits remain unchanged.

WO: Only a write by software is valid; a read of the register returns no meaningful data.

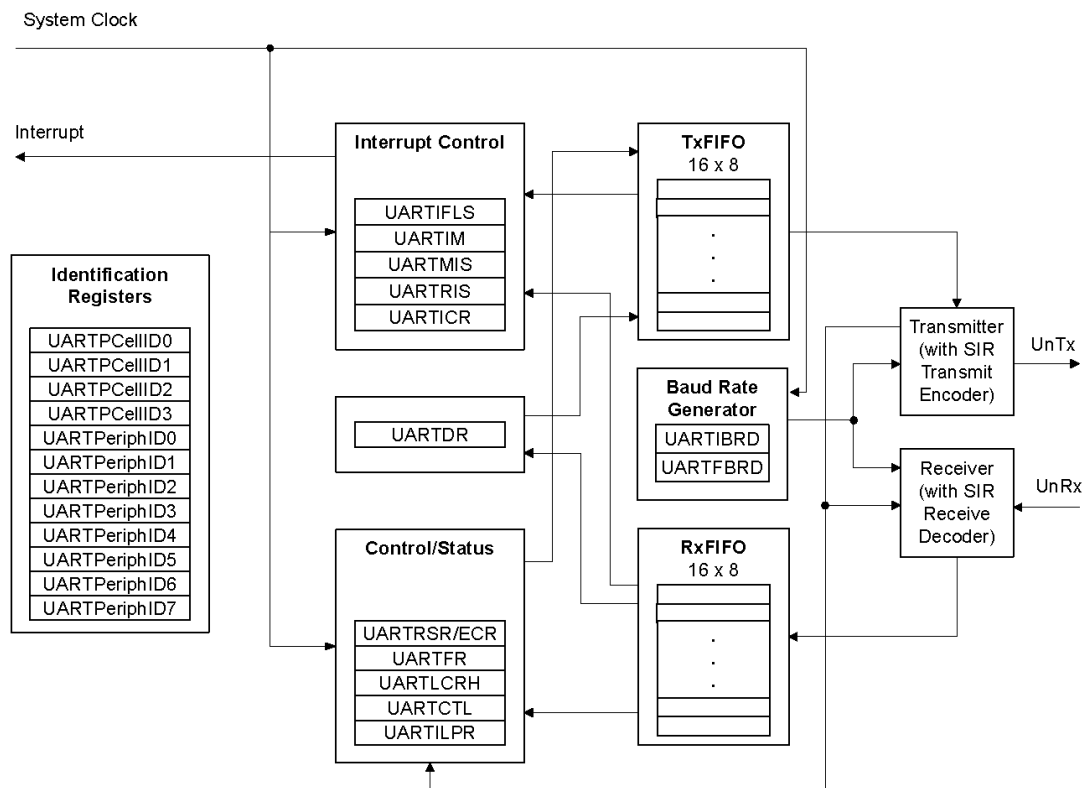


Figure 4. UART Module Block Diagram

References

- [1] Luminary Micro Technical Staff, *LM3S8962 Evaluation Board, User's Manual*, Texas Instruments, 2009. Available at:
http://www.luminarymicro.com/index.php?option=com_remository&func=download&id=523&chk=222579d07d3e13fde74ae411749ae30e&Itemid=591
- [2] A. Liakot and S. Rosalina and A. Ishak, "Design of a micro-UART for SoC application," *Computer & Electrical Engineering*, vol. 30, no. 4, p. 257-268, June 2004. [Abstract]. Available: ScienceDirect, <http://www.sciencedirect.com/>. [Accessed Nov 11, 2009].
- [3] Texas Instruments Technical Staff, *MAX232 Dual EIA-232 Drivers/Receivers*, Texas Instruments, 2009. Available at: <http://focus.ti.com/lit/ds/symlink/max232.pdf>
- [4] Lammert Bies, *Serial UART Information*, Lammert Bies 2008. Available:
<http://www.lammertbies.nl/comm/info/serial-uart.html>
- [5] PCCom Technical Staff, *Baud Rate*, PCCom. Available:
http://www.pccompci.com/Baud_Rate.html