



Using the UART Port Controller on SHARC® Processors

Contributed by Divya Sunkara

Rev 2 – July 5, 2007

Introduction

Universal asynchronous receiver transmitters (UARTs) are asynchronous serial communication peripherals that convert parallel data to serial data at the transmitter end and convert serial data into parallel data at the receiver end. These devices are capable of full-duplex communication over EIA-232E (formally referred to as RS-232) serial communication links.

This EE-Note describes UART characteristics and the programming required to configure communication between the ADSP-21371 ADSP-21375, ADSP-21367, ADSP-21368, and ADSP-21369 SHARC® processor's UART port controller and other asynchronous serial communication devices.



The ADSP-21367, ADSP-21368, ADSP-21369 SHARC processors have two UARTs, while the ADSP-21371 and ADSP-21375 SHARC processors have one. Hereafter, we will representatively refer to all of them as ADSP-21368 processors.

UART Data Frame

Figure 1 shows a typical UART data frame, which comprises a start bit followed by data bits and a stop bit. The parity bit, which can be selected for even or odd parity, is used for error detection. The total length of data bits in a frame

(excluding start, stop, and parity bits) may be of any length varying from 5 to 8. The number of stop bits per frame can be programmed to be 1 or 2. Considering all these variations, a UART data frame length can vary between 7 and 12 bits. The frame's start bit is an active low bit and is detected at the falling edge; the stop bit is an active high bit and is detected at the rising edge. Both the transmitter end and the receiver end must be given identical baud rates for proper communication between them.

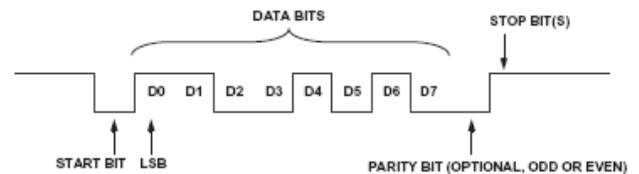


Figure 1. UART data frame

Synchronization Effects

When two UARTs communicate, the transmitter and receiver depend on the baud rate at which data is being transmitted. The transmitter transmits data at the given baud rate, and the receiver detects the start of the frame and then reads data by sampling at the same baud rate. Because the receiver does not know when the data frame arrives with respect to the receiver clock, the communication is termed asynchronous. Since the receiver has to detect the start of the frame, the logic at the receiver side is more complex than the transmitter side.

Based on the communication protocol, the receiver clock generated at the receiver side is 16 times the baud rate. [Figure 2](#) shows the protocol used by the receivers to synchronize to the data frame. The receiver detects the start of a frame at the falling edge of an active low start bit when the data signal transitions from its stop bit or its idle condition. At this falling edge, the receiver resets its clock counter and expects to find the midpoint of the start bit after eight clock cycles. The start bit is re-sampled at the midpoint in order to verify that the initially detected falling edge is not noise or a glitch. It then samples the next bit after 16 clock cycles, which is shown as the midpoint of the D0 bit in [Figure 2](#).

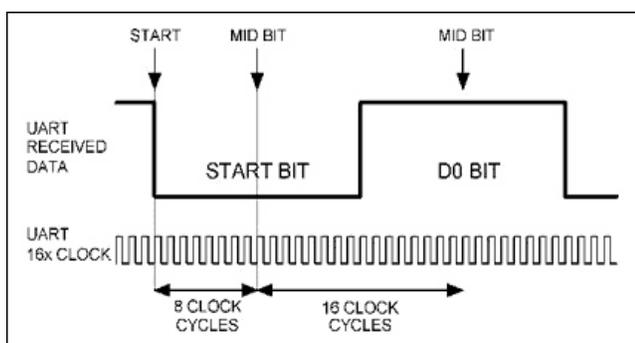


Figure 2. UART synchronization effects

Because the UART receiver re-synchronizes at the start of each frame, sampling errors do not accumulate beyond the stop bit of the previous frame. Hence, sampling errors are a concern within a data frame, but not in the entire transmitted data signal. Sampling errors occur usually due to a synchronization error between the receiver clock and the received data frame. This error builds over the entire frame and shifts the sampling point of a data bit closer to the transition edges. Due to the slow rise and fall times of these transition edges, this may lead to detection of erroneous data. The slow rise and fall times occur due to the high capacitance of the transmission wires. When the stop bit is sampled low instead of high, a framing error occurs. Since the baud rate for the UART on the processor side is derived from the processor's peripheral clock, the required generated baud

rate may not be exact. This error may further lead to timing variations on the transmitter and receiver, which can lead to faulty data sampling.

UART External Interface

The EIA-232E interface is used to communicate with other external serial communication devices (such as a PC). The EIA-232E has the same data format as that of a UART, but with different voltage level detection (see [Figure 3](#)). The standard output voltage level of an EIA-232E interface usually ranges between +12V and -12V with a dead area of +3V to -3V to absorb line noise. Hence, the EIA-232E data is bipolar with +3V to +12V, indicating an on-state or 0-state (SPACE) condition. The -3V to -12V range indicates an off-state or 1-state (MARK).

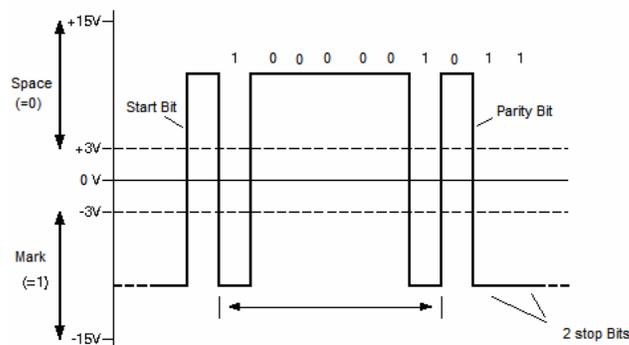


Figure 3. EIA-232E data logic levels

Because the UART port controller in the 21368 processors produce signals at lower logic levels, an EIA-232E transceiver (see [Figure 4](#)) is required to convert the UART signal levels to EIA-232E levels and vice versa. The ADM3202 in [Figure 4](#) has step-up voltage converters coupled with level-shifting transmitters and receivers that operate at a power supply voltage of 3.3V. This EIA-232E transceiver is required to communicate with other external devices. For details on the specifications and internal circuitry, refer to the ADM3202 data sheet. DB9 connectors are used to make physical connections between two serial communicating devices using EIA-232E cables.

Figure 4 shows interface connections and the devices required to allow the ADSP-21368

UART port controller communicate with external serial devices such as a PC.

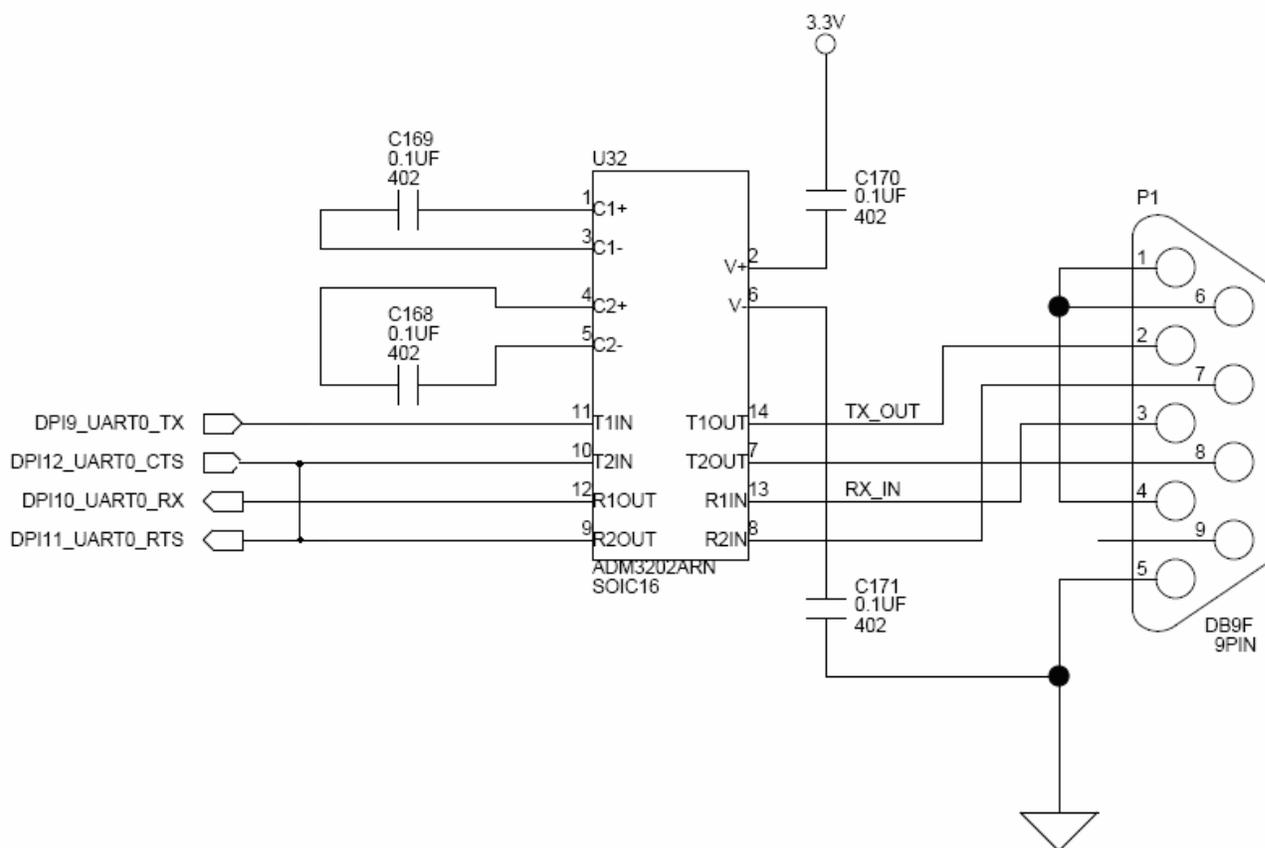


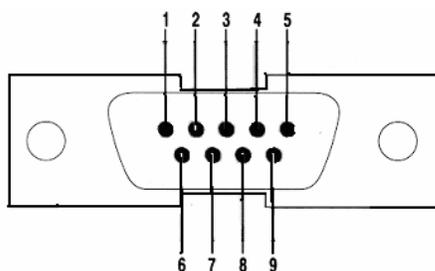
Figure 4. Connections and devices required to use the UART port controller signals on ADSP-21368 Processors

Each device capable of EIA-232E communication is termed as a DTE (data terminal equipment) or a DCE (data communication equipment) device. You must know whether the communicating devices are DCE or DTE devices in order to use proper connection cables between them. Figure 5 shows the pin labeling of a DB9 connector.

Based on these pin designations, a serial communication device with a DB9 connector (male or female) can be determined as a DTE or DCE device using a simple DC voltage

measurements on pin 2 and pin 3 with respect to pin 5 (signal ground).

When the serial communication device is powered and is not sending data, one of these pins (either pin 2 or pin 3) will have a negative voltage less than -3V and the other pin will have little or no voltage. If the voltage on pin 2 is more negative than -3V, the device is termed as a DCE device; if pin 3 has a more negative voltage, the device is termed as a DTE.



Pin	Signal
1	Data Carrier Detect
2	Received Data
3	Transmitted Data
4	Data Terminal Ready
5	Signal Ground

Pin	Signal
6	Data Set Ready
7	Request to Send
8	Clear to Send
9	Ring Indicator

Figure 5. DB9 connector pins

A straight-through cable is required for DTE-DCE device communication. DTE-DTE device communication requires a cross-over cable, commonly called a null-modem cable. DCE-DCE device communication requires a cross-over cable called a tail circuit cable. The ADSP-21369 and ADSP-21375 EZ-KIT Lite® evaluation systems have a more negative voltage on pin 2 of its DB9 connector (DCE). Windows-based PCs are DTE devices; hence, communication between an EZ-KIT Lite and a PC requires a straight-through cable connection between them. In Figure 5, pins 2 and 3 are used for transmitting or receiving and the other pins are used primarily for hardware handshaking between the communication devices. There is no handshaking support for the UART peripherals in these processors and hence handshaking signals are looped back as shown in Figure 4.

Programming Techniques

This section discusses programming techniques required for looped-back full-duplex DMA, UART interrupts, and communication with a PC for ADSP-21368 processors.

Full-Duplex DMA

Because the UART is a full-duplex peripheral, the transmitter and receiver operate simultaneously and independent of each other. When performing a full-duplex DMA in loop-back mode, (in which the transmitter output is connected to the receiver input), ensure that the receiver side of the UART is enabled before the transmitter side in order to avoid any loss of data on the receiver side. Listing 1 shows an example of full-duplex DMA on UART0 in loop-back mode.

```
#include <def21367.h>
.section/dm seg_dmda;
.var source1[8] = 0x11,
                0x22,
                0x33,
                0x44,
                0x55,
                0x66,
                0x77,
                0x88;

.var receive1[8];

.section/pm seg_pmco;
_main:

SRU2(UART0_TX_O,UART0_RX_I);
/* UART transmit signal is connected
to the receive input */

/* Sets the Baud rate for UART0 */
ustat1= UARTDLAB;
dm(UART0LCR) = ustat1;

/*enables access to Divisor register
to set baud rate for UART0 */

r0=0x8b; dm(UART0DLL) = r0;
r0=0x2;  dm(UART0DLH) = r0;

/*0x28b = 651 for divisor value and
gives a baud rate of19200 at a core
clock of 400Mhz */

/* Configure the UART0 line control
register */
ustat1 = UARTWLS8| /* word length 8 */
        UARTPEN; /* parity enable
odd parity */

dm(UART0LCR) = ustat1; /* sets UART0
Line with one stop bit, odd parity and
with baud rate of 19200 */
```

```

/* UART0 receiver is configured in DMA
mode*/
ustat1 = 0;
dm(UART0RXCTL) = ustat1; /* clears
the UART receive control register */

/* set the DMA parameter registers */
r0=receive1; dm(IIUART0RX) = r0;
r0 = 1; dm(IMUART0RX) = r0;
r0 = length(receive1);
dm(CUART0RX)=r0;
ustat1= UARTEN | UARTDEN;
dm(UART0RXCTL) = ustat1; /* Enables
UART0 in receive DMA mode */

/* UART0 transmitter is configured in
DMA mode */
ustat1 = 0;
dm(UART0TXCTL) = ustat1;
/* clears the UART0 transmit control
register */

/* Set the DMA transmit parameter
registers */
r0=source1;
dm(IIUART0TX) =r0;
r0= 1;
dm(IMUART0TX) =r0;
r0 = length(source1);
dm(CUART0TX) =r0;
ustat1 = UARTEN | UARTDEN;
dm(UART0TXCTL) = ustat1;
/* Enables UART0 transmit DMA mode; */

/* UART0 receiver Status polling */
waiting1: ustat2 = dm(UART0RXSTAT);
/* poll to ensure receive DMA of UART0
is done */
bit tst ustat2 UARTEMASTAT;
if tf jump waiting1;

ustat1 = 0;
dm(UART0RXCTL) = ustat1;
/* Disable UART0 receive */
dm(UART0TXCTL) = ustat1;
/* Disable UART0 transmit */

_main.end: jump(pc,0);

```

Listing 1. Full-duplex DMA on UART0 in loop-back mode

UART Interrupts

Two steps are required to enable UART interrupts in I/O mode. Be careful while using UART interrupts in I/O mode because all interrupts are grouped as a single receive interrupt. First, you must map the UART interrupts to one of the interrupts in the interrupt vector table. Use the DAI interrupt directly, by programming the DAI interrupt control register, or map the UART to any of the peripheral interrupt sources. This can be achieved by changing the default source of the peripheral interrupt priority control register with the UART source, using the interrupt select values of the UART receive interrupt. The receive interrupt select values for UART0 and UART1 are 0x13 and 0x14, respectively. Listing 2 illustrates the mapping of UART receive interrupt to the DAI interrupt.

```

bit set mode1 IRPTEN; /* enables
global interrupts */

bit set imask DPII; /* unmask DPI
interrupt */

ustat1 = UART0_RX_INT;
dm(DPI IRPTL RE) = ustat1; /* Enables
transmit interrupt in I/O mode */

```

Listing 2. Mapping UART receive interrupt to the DAI interrupt

Listing 3 illustrates the mapping of the interrupt to peripheral 14 interrupt using the peripheral interrupt control register. For details on programming the peripheral interrupt control register, refer to the *ADSP-21368 SHARC Processor Hardware Reference* ^[1].

```
#define MASKP14 (0x1f<<10)
#define UART0Rx (0x13<<10)

bit set model IRPTEN;

/* Maps the UART0 receive interrupt to
P14 using the programmable interrupt
controller */

ustat1 = dm(PICR2);
bit clr ustat1 MASKP14;
bit set ustat1 UART0Rx;
dm(PICR2) = ustat1;

bit set IMASK P14I; /*Unmasks the
UART Receive interrupt */
```

Listing 3. Mapping interrupt to peripheral 14 interrupt using peripheral interrupt control register

The second step in I/O mode involves enabling the UART interrupts internally by setting the corresponding bits in the UART interrupt enable register (UARTxIER). Enable the interrupt internally after all the UART settings (such as word length, parity, and so on) have been programmed, because in transmit mode as soon as the transmit buffer empty bit is enabled in the UARTxIER register, it vectors to the interrupt. If this bit is enabled before any of the UART settings are programmed, the data transmitted in the transmit interrupt service routine does not comply with the UART settings that are programmed later, leading to a communication error. In DMA mode, it is enough to enable the corresponding interrupt for transmit or receive DMA as described in step one of I/O mode. You need not enable the interrupts internally using the UARTxIER register in DMA mode for transmit or receive interrupts. For line status interrupts caused due by parity error, overrun errors, framing errors, or address detect interrupts, the corresponding interrupt in the UARTxIER register must be enabled even in DMA mode.

While using UART interrupts, do not open the UART register window (the UART registers in this window affect the peripheral's interrupt status) in VisualDSP++® while running or stepping through the code. Doing so would clear

the interrupt latches as a result of the emulator constantly trying to read these registers when the window is open. For information on bits in the UART registers, refer to the *Hardware Reference Manual*^[1].

Communication with PC

For communication with a PC, ensure that the internal settings programmed in the UART match the configurations of the COM port by using Windows® HyperTerminal. Listing 4 shows the UART initializations used for communicating with PC. The UART settings in the program match the configurations of Windows HyperTerminal shown in Figure 6.

```
.SECTION / CODE program;
.section/pm seg_pmco;
_initUART:

/* Sets the Baud rate for UART0 */
ustat1= UARTDLAB;
dm(UART0LCR) = ustat1; /*enables
access to Divisor register to set baud
rate for UART0 */

r0=0x1c; dm(UART0DLL) = r0;
r0=0x2; dm(UART0DLH) = r0; //0x21c =
540 for divisor value and gives a baud
rate of 19200 at 331.776Mhz core clock

/* Configures the UART LCR */
ustat1 = UARTWLS8| /* word length 8 */
        UARTPEN| /* parity enable
for odd parity */
        UARTSTB ; /* two stop bits */
dm(UART0LCR) = ustat1; /* sets UART0
Line with one stop bit, odd parity and
with baud rate of 19200 */

ustat1 = UARTEEN;
dm(UART0TXCTL) = ustat1; /* enable
UART0 transmitter */

ustat1 = UARTEEN;
dm(UART0RXCTL) = ustat1; /* enables
UART0 in receive mode */
_initUART.end;
```

Listing 4. Initializing a UART to communicate with a PC

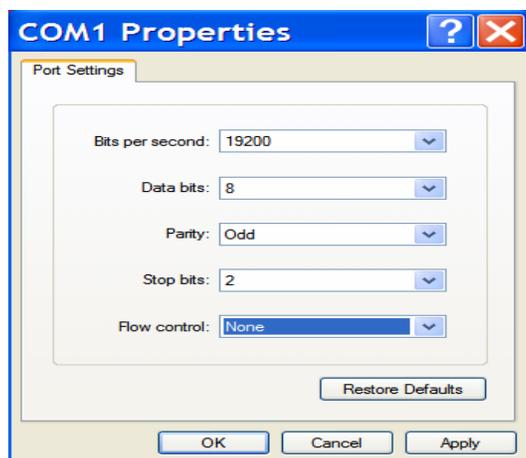


Figure 6. Configuring windows hyperterminal

Conclusion

This document discusses the characteristics of UART port controller and its external interface. It also discusses key programming techniques on different aspects of the peripheral that require attention for proper functioning.

References

- [1] *ADSP-21368 SHARC Processor Hardware Reference*, Revision 1.0, September 2006, Analog Devices, Inc.
- [2] *ADSP-21367/ADSP-21368/ADSP-21369 SHARC Processor Data Sheet*, Revision A, August 2006, Analog Devices, Inc.
- [3] *ADSP-21371: High Performance 32-bit Floating-Point SHARC Processor for Automotive Audio Preliminary Data Sheet*, Revision PrA, June 2006, Analog Devices, Inc.
- [4] *ADSP-21375: 266 MHz High Performance SHARC Processor Preliminary Data Sheet*, Revision PrB, December 2005, Analog Devices, Inc.
- [5] *ADSP-21369 EZ-KIT Lite Evaluation System Manual*, Revision 2.1, August 2006, Analog Devices, Inc.
- [6] *ADSP-21375 EZ-KIT Lite Evaluation System Manual*, Revision 1.0, September 2006, Analog Devices, Inc.
- [7] *ADM3202 – High-Speed, 2-Channel RS232/V.28 Interface Devices*, Revision D, Analog Devices, Inc.

Document History

Revision	Description
<i>Rev 2 – July 5, 2007 by Divya Sunkara</i>	Updated the external interface diagram in Figure 4. Connected the CTS and RTS signal as there is no support for handshaking
<i>Rev 1 – November 20, 2006 by Divya Sunkara</i>	Initial release