Serial Comm

EE3376
Serial Communication Interface

- Referred to as UART in most companies (N16550 compatible)
- Serial communication where no clock is shared (thus async)
- Both receiver / transmitter share only a nominal baud rate
- 1 stop bit and start bit per 8 bits of data (20% overhead)
- Optional parity bit for error checking
- **Rule of Thumb** is that baud error should be less than 1%.

![Diagram showing serial communication interface between two systems with baud clock values.](image-url)
Universal Serial Communication Interface, UART Mode

The universal serial communication interface (USCI) supports multiple serial communication modes with one hardware module. This chapter discusses the operation of the asynchronous UART mode.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.1 USCI Overview</td>
<td>422</td>
</tr>
<tr>
<td>15.2 USCI Introduction: UART Mode</td>
<td>422</td>
</tr>
<tr>
<td>15.3 USCI Operation: UART Mode</td>
<td>424</td>
</tr>
<tr>
<td>15.4 USCI Registers: UART Mode</td>
<td>439</td>
</tr>
</tbody>
</table>
Serial Communication Interface

- TX signal stays high while idle
- When beginning a transaction, start bit always goes low for 1 bit
- Next 8 baud periods, data is sampled.
- 10th bit should be stop bit and is always high.
- Internally, data is sampled at a higher rate (16x) and averaged

![Diagram of serial communication interface](image)
Asynch – How does it work?

- Baud rates are both nominally (in name only) 9600.
- Actual bauds are not exactly the same but are close enough that the first 10/11 bits are received correctly.
- Receiver is aware of incoming frame upon sampling a low (start bit).
- Receiver is aware of successful completion if the parity matches (optional) and the last bit is high (stop bit).
- In reality, receiver samples at a higher rate of 9600 x 16 and gets average of 16 values it reads in each bit. This helps noise and mismatch.
Asynch – example wrong baud

- Case of frequency mismatch too great to receive value correctly.
- Bit 1 of transmitted word is received as stop bit – if low, (stop bit always high) than the receiver will be aware of the corruption.
- Otherwise, corruption would go undetected unless parity used.
- In this case,
  \[ R7=T7, R6=T6, R5=T5, R4=T5, R3=T4, R2=T3, R1=T2, R0=T2. \]
Figure 15-1. USCI_Ax Block Diagram: UART Mode (UCSYNC = 0)
# USCI Programming Model

## Table 15-6. USCI_A0 Control and Status Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Short Form</th>
<th>Register Type</th>
<th>Address</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>USCI_A0 control register 0</td>
<td>UCA0CTL0</td>
<td>Read/write</td>
<td>060h</td>
<td>Reset with PUC</td>
</tr>
<tr>
<td>USCI_A0 control register 1</td>
<td>UCA0CTL1</td>
<td>Read/write</td>
<td>061h</td>
<td>001h with PUC</td>
</tr>
<tr>
<td>USCI_A0 Baud rate control register 0</td>
<td>UCA0BR0</td>
<td>Read/write</td>
<td>062h</td>
<td>Reset with PUC</td>
</tr>
<tr>
<td>USCI_A0 baud rate control register 1</td>
<td>UCA0BR1</td>
<td>Read/write</td>
<td>063h</td>
<td>Reset with PUC</td>
</tr>
<tr>
<td>USCI_A0 modulation control register</td>
<td>UCA0MCTL</td>
<td>Read/write</td>
<td>064h</td>
<td>Reset with PUC</td>
</tr>
<tr>
<td>USCI_A0 status register</td>
<td>UCA0STAT</td>
<td>Read/write</td>
<td>065h</td>
<td>Reset with PUC</td>
</tr>
<tr>
<td>USCI_A0 receive buffer register</td>
<td>UCA0RXBUF</td>
<td>Read</td>
<td>066h</td>
<td>Reset with PUC</td>
</tr>
<tr>
<td>USCI_A0 transmit buffer register</td>
<td>UCA0TXBUF</td>
<td>Read/write</td>
<td>067h</td>
<td>Reset with PUC</td>
</tr>
<tr>
<td>USCI_A0 Auto baud control register</td>
<td>UCA0ABCTL</td>
<td>Read/write</td>
<td>05Dh</td>
<td>Reset with PUC</td>
</tr>
<tr>
<td>USCI_A0 IrDA transmit control register</td>
<td>UCA0IRTCTL</td>
<td>Read/write</td>
<td>05Eh</td>
<td>Reset with PUC</td>
</tr>
<tr>
<td>USCI_A0 IrDA receive control register</td>
<td>UCA0IRRCTL</td>
<td>Read/write</td>
<td>05Fh</td>
<td>Reset with PUC</td>
</tr>
<tr>
<td>SFR interrupt enable register 2</td>
<td>IE2</td>
<td>Read/write</td>
<td>001h</td>
<td>Reset with PUC</td>
</tr>
<tr>
<td>SFR interrupt flag register 2</td>
<td>IFG2</td>
<td>Read/write</td>
<td>003h</td>
<td>00Ah with PUC</td>
</tr>
</tbody>
</table>
### 15.4.1 UCAxCTL0, USCI_Ax Control Register 0

<table>
<thead>
<tr>
<th>Bit 7 UCPEN</th>
<th>Bit 6 UCPAR</th>
<th>Bit 5 UCMSB</th>
<th>Bit 4 UC7BIT</th>
<th>Bit 3 UCSPB</th>
<th>Bits 2-1 UCMODEx</th>
<th>Bit 0 UCSYNC</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
</tr>
</tbody>
</table>

**UCPEN** Bit 7  Parity enable
- 0  Parity disabled.
- 1  Parity enabled. Parity bit is generated (UCAxTXD) and expected (UCAxRXD). In address-bit multiprocessor mode, the address bit is included in the parity calculation.

**UCPAR** Bit 6  Parity select. UCPAR is not used when parity is disabled.
- 0  Odd parity
- 1  Even parity

**UCMSB** Bit 5  MSB first select. Controls the direction of the receive and transmit shift register.
- 0  LSB first
- 1  MSB first

**UC7BIT** Bit 4  Character length. Selects 7-bit or 8-bit character length.
- 0  8-bit data
- 1  7-bit data

**UCSPB** Bit 3  Stop bit select. Number of stop bits.
- 0  One stop bit
- 1  Two stop bits

**UCMODEx** Bits 2-1  USCI mode. The UCMODEx bits select the asynchronous mode when UCSYNC = 0.
- 00  UART mode
- 01  Idle-line multiprocessor mode
- 10  Address-bit multiprocessor mode
- 11  UART mode with automatic baud rate detection

**UCSYNC** Bit 0  Synchronous mode enable
- 0  Asynchronous mode
- 1  Synchronous mode
### 15.4.2 UCAxCTL1, USCI_Ax Control Register 1

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:6</td>
<td>UCSSELx: USCI clock source select. These bits select the BRCLK source clock.</td>
</tr>
<tr>
<td>5</td>
<td>UCRXEIE: Receive erroneous-character interrupt-enable.</td>
</tr>
<tr>
<td>4</td>
<td>UCBRKIE: Receive break character interrupt-enable.</td>
</tr>
<tr>
<td>3</td>
<td>UCDORM: Dormant. Puts USCI into sleep mode.</td>
</tr>
<tr>
<td>2</td>
<td>UCTXADDR: Transmit address. Next frame to be transmitted will be marked as address depending on</td>
</tr>
<tr>
<td></td>
<td>the selected multiprocessor mode.</td>
</tr>
<tr>
<td>1</td>
<td>UCTXBRK: Transmit break. Transmits a break with the next write to the transmit buffer.</td>
</tr>
<tr>
<td>0</td>
<td>UCSWRST: Software reset enable.</td>
</tr>
</tbody>
</table>

- **UCSELx**
  - Bits 7-6: USCI clock source select. These bits select the BRCLK source clock.
  - 00: UCLK
  - 01: ACLK
  - 10: SMCLK
  - 11: SMCLK

- **UCRXIE**
  - Bit 5: Receive erroneous-character interrupt-enable
  - 0: Erroneous characters rejected and UCAxRXIFG is not set
  - 1: Erroneous characters received will set UCAxRXIFG

- **UCBRKIE**
  - Bit 4: Receive break character interrupt-enable
  - 0: Received break characters do not set UCAxRXIFG.
  - 1: Received break characters set UCAxRXIFG.

- **UCDORM**
  - Bit 3: Dormant. Puts USCI into sleep mode.
  - 0: Not dormant. All received characters will set UCAxRXIFG.
  - 1: Dormant. Only characters that are preceded by an idle-line or with address bit set will set UCAxRXIFG. In UART mode with automatic baud rate detection only the combination of a break and synch field will set UCAxRXIFG.

- **UCTXADDR**
  - Bit 2: Transmit address. Next frame to be transmitted will be marked as address depending on the selected multiprocessor mode.
  - 0: Next frame transmitted is data
  - 1: Next frame transmitted is an address

- **UCTXBRK**
  - Bit 1: Transmit break. Transmits a break with the next write to the transmit buffer. In UART mode with automatic baud rate detection 055h must be written into UCAxTXBUF to generate the required break/synch fields. Otherwise 0h must be written into the transmit buffer.
  - 0: Next frame transmitted is not a break
  - 1: Next frame transmitted is a break or a break/synch

- **UCSWRST**
  - Bit 0: Software reset enable
  - 0: Disabled. USCI reset released for operation.
  - 1: Enabled. USCI logic held in reset state.
USA0BR0 and USA0BR1

15.4.3 UCAxBR0, USCI Ax Baud Rate Control Register 0

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RW</td>
<td>Rw</td>
<td>Rw</td>
<td>Rw</td>
<td>Rw</td>
</tr>
</tbody>
</table>

UCBRx

15.4.4 UCAxBR1, USCI Ax Baud Rate Control Register 1

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RW</td>
<td>Rw</td>
<td>Rw</td>
<td>Rw</td>
<td>Rw</td>
</tr>
</tbody>
</table>

UCBRx

UCBRx  7-0  Clock prescaler setting of the Baud rate generator. The 16-bit value of (UCAxBR0 + UCAxBR1 × 256) forms the prescaler value.
15.4.6 "UCAxSTAT, USCI_Ax Status Register"

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
</table>
| 7     | **UCLISTEN** Bit 7
|       | Listen enable. The UCLISTEN bit selects loopback mode.                     |
|       | 0 Disabled                                                                  |
|       | 1 Enabled. UCAxTXD is internally fed back to the receiver.                 |
| 6     | **UCFE** Bit 6
|       | Framing error flag                                                         |
|       | 0 No error                                                                 |
|       | 1 Character received with low stop bit                                     |
| 5     | **UCOE** Bit 5
|       | Overrun error flag. This bit is set when a character is transferred into UCAxRXBUF before the previous character was read. UCOE is cleared automatically when UCAxRXBUF is read, and must not be cleared by software. Otherwise, it will not function correctly. |
|       | 0 No error                                                                 |
|       | 1 Overrun error occurred                                                    |
| 4     | **UCPE** Bit 4
|       | Parity error flag. When UCPEN = 0, UCPE is read as 0.                      |
|       | 0 No error                                                                 |
|       | 1 Character received with parity error                                     |
| 3     | **UCBRK** Bit 3
|       | Break detect flag                                                          |
|       | 0 No break condition                                                        |
|       | 1 Break condition occurred                                                  |
| 2     | **UCRXERR** Bit 2
|       | Receive error flag. This bit indicates a character was received with error(s). When UCRXERR = 1, on or more error flags (UCFE, UCPE, UCOE) is also set. UCRXERR is cleared when UCAxRXBUF is read. |
|       | 0 No receive errors detected                                               |
|       | 1 Receive error detected                                                    |
| 1     | **UCADDR** Bit 1
|       | Address received in address-bit multiprocessor mode.                       |
|       | 0 Received character is data                                               |
|       | 1 Received character is an address                                         |
| 0     | **UCIDLE**
|       | Idle line detected in idle-line multiprocessor mode.                      |
|       | 0 No idle line detected                                                    |
|       | 1 Idle line detected                                                       |
|       | **UCBUSY** Bit 0
|       | USCI busy. This bit indicates if a transmit or receive operation is in progress. |
|       | 0 USCI inactive                                                             |
|       | 1 USCI transmitting or receiving                                           |
**USCI - Interrupts**

### 15.4.12 IE2, Interrupt Enable Register 2

<table>
<thead>
<tr>
<th>Bit 7-2</th>
<th>Description</th>
<th>Value</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCA0TXIE Bit 1</td>
<td>USCI_A0 transmit interrupt enable</td>
<td>0</td>
<td>Interrupt disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Interrupt enabled</td>
</tr>
<tr>
<td>UCA0RXIE Bit 0</td>
<td>USCI_A0 receive interrupt enable</td>
<td>0</td>
<td>Interrupt disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Interrupt enabled</td>
</tr>
</tbody>
</table>

Bits 7-2: These bits may be used by other modules (see the device-specific data sheet).

### 15.4.13 IFG2, Interrupt Flag Register 2

<table>
<thead>
<tr>
<th>Bit 7-2</th>
<th>Description</th>
<th>Value</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCA0TXIFG Bit 1</td>
<td>USCI_A0 transmit interrupt flag. UCA0TXIFG is set when UCA0TXBUF is empty.</td>
<td>0</td>
<td>No interrupt pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Interrupt pending</td>
</tr>
<tr>
<td>UCA0RXIFG Bit 0</td>
<td>USCI_A0 receive interrupt flag. UCA0RXIFG is set when UCA0RXBUF has received a complete character.</td>
<td>0</td>
<td>No interrupt pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Interrupt pending</td>
</tr>
</tbody>
</table>

Bits 7-2: These bits may be used by other modules (see the device-specific data sheet).
15.4.7  **UCAxRXBUF, USCI_Ax Receive Buffer Register**

<table>
<thead>
<tr>
<th></th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

**UCRXBUFx**

Bits 7-0  The receive-data buffer is user accessible and contains the last received character from the receive shift register. Reading UCAxRXBUF resets the receive-error bits, the UCADDR or UCIDLE bit, and UCAxRXIFG. In 7-bit data mode, UCAxRXBUF is LSB justified and the MSB is always reset.

15.4.8  **UCAxTXBUF, USCI_Ax Transmit Buffer Register**

<table>
<thead>
<tr>
<th></th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

**UCTXBUFx**

Bits 7-0  The transmit data buffer is user accessible and holds the data waiting to be moved into the transmit shift register and transmitted on UCAxTXD. Writing to the transmit data buffer clears UCAxTXIFG. The MSB of UCAxTXBUF is not used for 7-bit data and is reset.
15.3.5.1 IrDA Encoding

The encoder sends a pulse for every zero bit in the transmit bit stream coming from the UART as shown in Figure 15-7. The pulse duration is defined by UCIRTXPLx bits specifying the number of half clock periods of the clock selected by UCIRTXCLK.

![Figure 15-7. UART vs IrDA Data Format](image)

To set the pulse time of 3/16 bit period required by the IrDA standard the BITCLK16 clock is selected with UCIRTXCLK = 1 and the pulse length is set to 6 half clock cycles with UCIRTXPLx = 6 – 1 = 5.

When UCIRTXCLK = 0, the pulse length $t_{\text{Pulse}}$ is based on BRCLK and is calculated as follows:

$$UCIRTXPLx = t_{\text{Pulse}} \times 2 \times f_{\text{BRCLK}} - 1$$

When the pulse length is based on BRCLK the prescaler UCBRx must to be set to a value greater or equal to 5.

15.3.5.2 IrDA Decoding

The decoder detects high pulses when UCIRRXPL = 0. Otherwise it detects low pulses. In addition to the analog deglitch filter an additional programmable digital filter stage can be enabled by setting UCIRRXFE. When UCIRRXFE is set, only pulses longer than the programmed filter length are passed. Shorter pulses are discarded. The equation to program the filter length UCIRRXFLLx is:

$$UCIRRXFLLx = (f_{\text{Pulse}} - f_{\text{Wake}}) \times 2 \times f_{\text{BRCLK}} - 4$$

Where,

- $f_{\text{Pulse}}$ = Minimum receive pulse width
- $f_{\text{Wake}}$ = Wake time from any low power mode. Zero when MSP430 is in active mode.
The USA0BR0 and USA0BR1 represent a value – BR – between 1 and 64K
Baud Rate = CLK / (BR)

where ACLK = LVO at 1MHz

Example:
ACLK =1MHz, BR=104

Baud Rate = 1MHz / (104) = 9615

Baud Error = 9615 – 9600 / 9600 = .15% = good enough!
```c
void serial_init(void)
{
    P1SEL = BIT1 + BIT2;              // Select UART as the pin function
    P1SEL2 = BIT1 + BIT2;

    UCAOCTL1 |= UCSWRST;             // Disable UART module for configuration

    UCAOCTL0 = 0x00;                 // No parity, LSB first, 8-bit data, 1 stop bit, UART, Asynchronous
    UCAOCTL1 = UCSSEL_2 + UCSWRST;  // SMCLK source, keep in reset state

    //
    UCAOBRO = 130;                   // 9600 Baud rate  - Assumes 16 MHz clock
    //
    UCAOBR1 = 6;                     // 9600 Baud rate  - Assumes 16 MHz clock
    UCAOBRO = 104;                   // 9600 Baud rate  - Assumes 1 MHz clock
    UCAOBR1 = 0;
    UCAOMCTL = 0x02;                 // 2nd Stage modulation = 1, Oversampling off
                                  // Interrupts disabled
    IE2 = 0x00;

    UCAOCTL1 &= ~UCSWRST;            // Enable UART module
}
```
USCI Functions

```c
void serial_charTX(char c)
{
    while( !(IFG2 & UCA0TXIFG) ); // Wait until the transmit buffer is empty
    UCA0TXBUF = c; // Send the character through the Xmit buffer
}

char serial_charRX(void)
{
    while( !(IFG2 & UCA0RXIFG) ); // Wait until a character has been received
    return UCA0RXBUF; // Return received character
}

void serial_string(char string[])
{
    int i;
    for(i = 0; string[i] != '\0'; i++) // Send characters until end-of-string
    {
        if(string[i] == '\n') // The terminal program recognizes \r as the carriage return
            serial_charTX('\r');
        else
            serial_charTX(string[i]);
    }
}

void serial_number(int value)
{
    char string[10];
    sprintf(string, "%d", value); // Convert an integer to a character string
    serial_string(string);
}
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;  // Stop WDT
    BCSCCTL1 = CALBC1_1MHZ;    // Set DCO
    DCOCCTL = CALDCO_1MHZ;
    P1SEL1 = BIT1 + BIT2;      // P1.1 = RXD, P1.2 = TXD
    P1SEL2 = BIT1 + BIT2;      // P1.1 = RXD, P1.2 = TXD
    UCA0CTL1 |= UCSEL2_2;     // SMCLK
    UCA0BR0 = 104;            // 1MHz 9600
    UCA0BR1 = 0;              // 1MHz 9600
    UCA0MCTL = UCBRS0;        // Modulation UCBRSx = 1
    UCA0CTL1 &^ ~UCSWRST;    // **Initialize USCI state machine**
    IE2 | = UCA0RXIE;         // Enable USCI_A0 RX interrupt

    __bis_SR_register(LPM0_bits + GIE); // Enter LPM0, interrupts enabled
}

// Echo back RXed character, confirm TX buffer is ready first
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
    while(!(IFG2&UCA0TXIFG)); // USCI_A0 TX buffer ready?
    UCA0TXBUF = UCA0RXBUF;   // TX -> RXed character
}
/**
 * MSP430G2xx3 Demo - USCI_A0, 19200 UART Echo ISR, DCO SMCLK
 * Description: Echo a received character, RX ISR used. Normal mode is LPM0.
 * USCI_A0 RX interrupt triggers TX Echo.
 * Baud rate divider with 1MHz = 1MHz/19200 = ~52.1
 * ACLK = n/a, MCLK = SMCLK = CALxxx_1MHZ = 1MHz
 * 
 * \ 
 * // USCI Function
 * 
 * \ 
 * void main(void)
 * {
 *     WDTCTL = WDTPW + WDTHOLD;   // Stop WDT
 *     BCSCTL1 = CALBC1_1MHZ;      // Set DCO
 *     DCOCTL = CALDCO_1MHZ;
 *     P1SEL = BIT1 + BIT2;        // P1.1 = RXD, P1.2=TXD
 *     P1SEL2 = BIT1 + BIT2;
 *     UCSCTL1 |= UCSEL_2;         // SMCLK
 *     UCA0BR0 = 52;               // 1MHz 19200
 *     UCA0BR1 = 0;                // 1MHz 19200
 *     UCA0MCTL = UCBRS0;          // Modulation UCBRSx = 1
 *     UCA0CTL1 &= ~UCSWRST;       // **Initialize USCI state machine**
 *     IE2 |= UCA0RXIE;            // Enable USCI_A0 RX interrupt
 * 
 *     __bis_SR_register(LPM0_bits + GIE);  // Enter LPM0, interrupts enabled
 * }
 *
 * // Echo back RXed character, confirm TX buffer is ready first
 * #pragma vector=USCI0RX_VECTOR
 * 
 * @interrupt void USCI0RX_ISR(void)
 * {
 *     while (!(IFG2 & UCA0TXIFG));   // USCI_A0 TX buffer ready?
 *     UCA0TXBUF = UCA0RXBUF;        // TX -> RXed character
 * }
 */

USCI Functions

MSP430G2xx3 Demo - USCI_A0, Ultra-Low Pwr UART 9600 String, 32kHz ACLK

Description: This program demonstrates a full-duplex 9600-baud UART using
USCI_A0 and a 32kHz crystal. The program will wait in LPM3, and will
respond to a received 'u' character using 8N1 protocol. The response will
be the string 'Hello World'.
ACLK = BRCLK = LFXT1 = 32768Hz, MCLK = SMCLK = DCO ~1.2MHz
Baud rate divider with 32768Hz XTAL @9600 = 32768Hz/9600 = 3.41
/* An external watch crystal is required on XIN XOUT for ACLK */

MSP430G2xx3

-------------
| \ | XIN|--
| | 32kHz
| --| RST XOUT|--
| | P1.2/UCA0TXD--------->
| | 9600 - 8N1
| | P1.1/UCA0RXD<----------

D. Dang
Texas Instruments Inc.
February 2011
Built with CCS Version 4.2.0 and IAR Embedded Workbench Version: 5.10

#include "msp430g2553.h"

const char string1[] = { "Hello World\r\n" };
unsigned int i;
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;
P1DIR = 0xFF;
P1OUT = 0;
P2DIR = 0xFF;
P2OUT = 0;
P1SEL = BIT1 + BIT2;
P1SEL2 = BIT1 + BIT2;
P3DIR = 0xFF;
P3OUT = 0;
UCA0CTL1 |= UCSSEL_1;
UCA0BR0 = 0x03;
UCA0BR1 = 0x00;
UCA0MCTL = UCBRS1 + UCBRS0;
UCA0CTL1 &= ~UCSWRST;
IE2 |= UCA0RXIE;

__bis_SR_register(LPM3_bits + GIE);
}

#pragma vector=USCIAB0TX_VECTOR
__interrupt void USCI0TX_ISR(void)
{
    UCA0TXBUF = string1[i++];
    if (i == sizeof string1 - 1)
        IE2 &= ~UCA0TXIE;
}

#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
    if (UCA0RXBUF == 'u')
    {
        i = 0;
        IE2 |= UCA0TXIE;
        UCA0TXBUF = string1[i++];
    }
}

// Stop WDT
// All P1.x outputs
// All P2.x reset
// P2.x outputs
// All P2.x reset
// P1.1 = RXD, P1.2=TXD
// P1.1 = RXD, P1.2=TXD
// All P3.x outputs
// All P3.x reset
// CLK = ACLK
// 32kHz/9600 = 3.41
//
// Modulation UCBRSx = 3
// **Initialize USCI state machine**
// Enable USCI_A0 RX interrupt
// Enter LPM3 w/ int until Byte RXed

// TX next character
// TX over?
// Disable USCI_A0 TX interrupt

// 'u' received?
// Enable USCI_A0 TX interrupt
/// USCI Functions

#include "msp430g2553.h"

char string1[8];
char i;
char j = 0;
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;
    P1DIR = 0xFF;
    P1OUT = 0;
    P2DIR = 0xFF;
    P2OUT = 0;
    P1SEL = BIT1 + BIT2 ;
    P1SEL2 = BIT1 + BIT2 ;
    P3DIR = 0xFF;
    P3OUT = 0;

    UCA0CTL1 |= UCSSEL_1;
    UCA0BR0 = 0x03;
    UCA0BR1 = 0x00;
    UCA0MCTL = UCBRS1 + UCBRS0;
    UCA0CTL1 &~= ~UCSWRST;
    IE2 |= UCA0RXIE;

    __bis_SR_register(LPM3_bits + GIE);
}

// USCI A0/B0 Transmit ISR
#pragma vector=USCIAB0TX_VECTOR
__interrupt void USCI0TX_ISR(void)
{
    UCA0TXBUF = string1[i++];

    if (i == sizeof string1)
        IE2 &= ~UCA0TXIE;
}

// USCI A0/B0 Receive ISR
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
    string1[j++] = UCA0RXBUF;
    if (j == sizeof string1 - 1)
    {
        i = 0;
        j = 0;
        IE2 |= UCA0TXIE;
        UCA0TXBUF = string1[i++];
    }
}
Serial Peripheral Interface

- Synchronous Serial Interface – clock sent with data
- Duplex with data simultaneously in both directions
- No start/stop overhead on bandwidth, but requires 4 signals
- Data sampled on edge of clock when frame asserted
  - alternatively, clock can be controlled and gated by master
- Useful for short, high speed (faster than UART) interface
- Commonly used for DAC or RTCs or HCS12 to HCS12 comm.
Serial Peripheral Interface

MSB first (LSBFE = 0): MSB Bit 6 Bit 5 Bit 4 Bit 3 Bit 2 Bit 1
LSB first (LSBFE = 1): LSB Bit 1 Bit 2 Bit 3 Bit 4 Bit 5 Bit 6

$t_L$ = Minimum leading time before the first SCK edge
$t_T$ = Minimum trailing time after the last SCK edge
Think of an SPI connection as a rolling shift register that resides (is shared) on two chips.
IIC – 2 Wire Interface

- Very inexpensive Interface
  - Only two wires – SCL and SDA – SCL is gated
  - Synchronous – no baud generation necessary for slaves
  - Multiple slaves allowed – up to 32
  - Wired OR configuration – pull up resistor on both lines
  - Slow speed – 100KHz original, 400KHz fast, new 1MHz high speed
  - Commonly used for EEPROMs