SJSU EE138 Chapter 10 UART

10.1 Asynchronous Serial Communication

Asynchronous serial communication has been in use since the early days of data communication for, such as, teletype equipment and user terminals for mainframe computers. It is also known as RS-232 protocol.

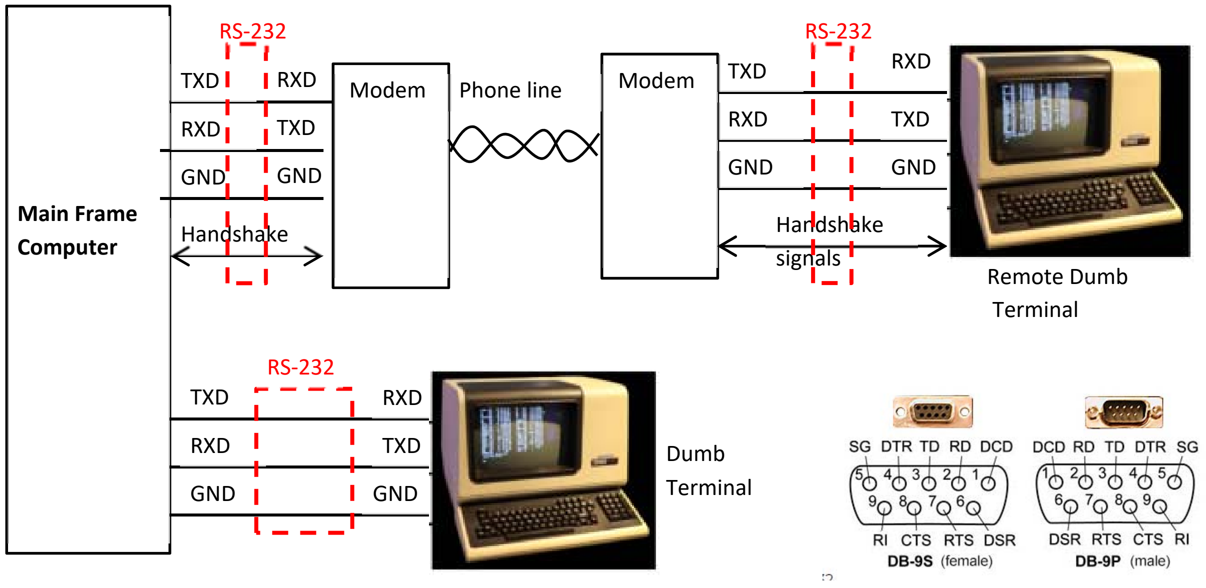


Figure 10.1 shows a typical data frame for serial communication.

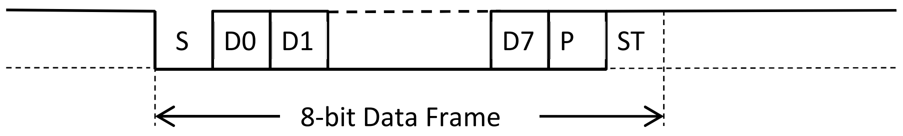


Figure 10.1 Data frame

In this figure, S stands for Start bit, P for Parity bit, and ST for STOP bit. The line is normally high (while idling). A high to low transition signals the starting of a data frame. This bit 'S' is called the start bit. The time period of a single bit determines the bit rate (Baud rate). The number of data bit is typically 7 or 8.

A parity bit 'P' follows the data bits. The parity bit is for error checking. The parity bit is either 0 or 1 such that the number of '1's in the data bits (including parity bit) is either even (even parity) or odd (odd parity). If one data bit is corrupted during the transmission, the receiver can detect this error by checking the parity bit. If, however, two (or any even number) bits are corrupted, the parity bit would not be able to detect this condition. Parity bit can be omitted when configuring the serial interface hardware.

A STOP bit follows the parity bit. This bit signals end of data frame.   Stop bit can be 1 or 2 bits in length.

For a communication channel, both the transmitter and the receiver must have the same configuration in terms of baud rate (the frequency of the data stream), # of data bits, parity (odd, even, or not at all), and # of stop bits. It is possible to add additional bits to the data field so long as both transmitting end and receiving end use the same data frame definition.

This serial communication operation is not synchronized by using a common clock signal or any additional handshaking signal (and hence the term 'Asynchronous communication').   The synchronization is entirely done by the receiver using the falling edge of the START bit as the starting of the data frame.

A programmable hardware circuit for implementing this serial communication protocol is called Universal Asynchronous Receiver and Transmitter (UART).   UART is for one-to-one communication, meaning it is for communication between just two devices. A UART typically supports two communication lines: one for receiving (RxD) and one for transmitting (TxD).

10.2 Universal Asynchronous Receiver and Transmitter (UART) in MSP432

The Enhanced Universal Serial Communication Interface (eUSCI_A) in MSP432 supports the operation of UART. Figure 10.2 is a block diagram of UART function in eUSCI. There are five blocks highlighted in red that are important for programmers. These blocks are:

- **Transmit shift register** – This register is a shift-register. The transmitting data byte is shifted-out, one bit at a time, at the baud rate.   This register is not accessible to software but it is important for the programmers to understand its operation.
- **Transmit buffer register** – The transmitting data byte should be written to this register by the software.   This data is automatically copied to the shifter-register after the current transmission is completed.   When the data is copied to the shift register, a transmit buffer empty flag is set. In response to this flag, the software should write another data byte into the buffer if there are more data to send.   Writing a new byte to transmit buffer clears the buffer empty flag.
- **Receive shift register**– This register is a shift-register. The receiving data byte is shifted-in to this register, one bit at a time, at the baud rate.   This register is not accessible to software but it is important for the programmers to understand its operation.
- **Receive buffer register**– After a complete byte is shifted into the receive shift register, it is automatically copied to the Receive buffer register.   When the data is copied to the buffer register, a receive buffer full flag is set. In response to this flag, the software should read the just received data byte.   Failing to do so before the next byte is received, an overrun flag will be set. Reading the receive buffer clears the buffer full flag.

- **Baud rate generator** – This circuit generates the clock signal used by the shift registers. The frequency of the clock is the baud rate. The baud rate frequency is programmable. This circuit is also responsible for synchronizing the receive clock with incoming data stream and for generating other timing critical control signals.

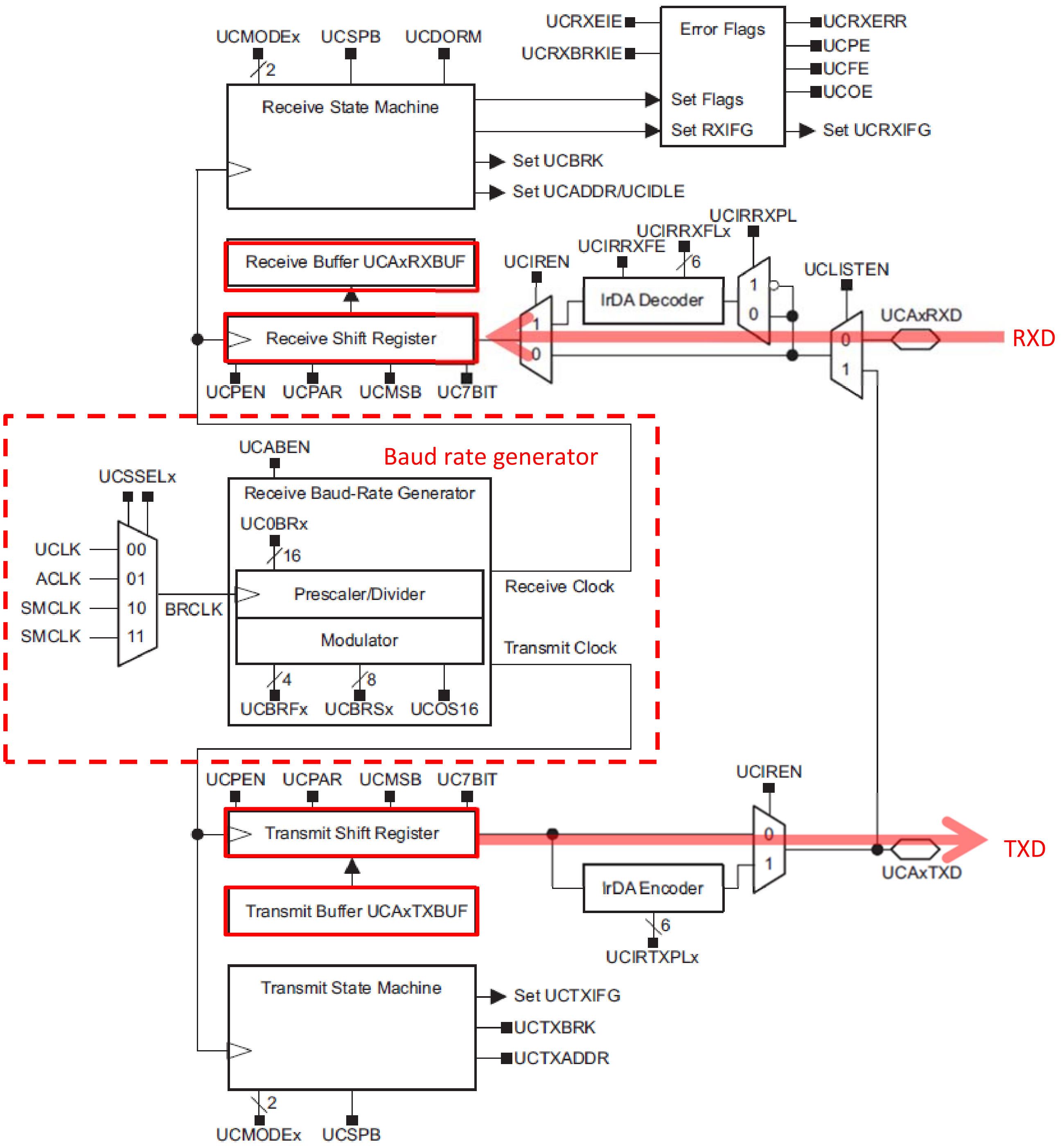


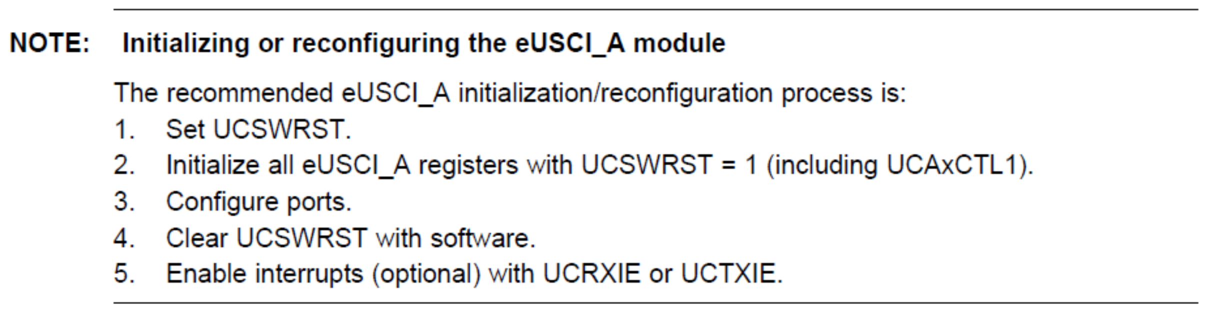
Figure 10.2 – eUSCI_A UART Mode Block Diagram

eUSCI_A offers many more functions beyond the basic serial communication protocol. Some of these functions are described below. We will not use any of these functions in the lab.

- Address bit - An Address bit (AD) can be added to the data frame after the data bits. This address bit is for multiple devices communicating on the same UART bus. This bit is used to indicate that the data within the frame is an address for a device on the bus.
- Automatic Baud-Rate Detection - MSP432 supports automatic baud-rate detection in UART mode. This allows multiple devices transmitting and receiving at different Baud-rate.
- IrDA **-** IrDA is a standardized, wireless, infrared data communications method that uses light to transmit data. MSP432 has an IrDA encoder/decoder that works alongside of UART.
- Glitch Suppression - A glitch suppression function prevents the eUSCI_A form being accidentally started due to mistaken a short glitch on the data line as the Start bit.

10.3 UART Operation

UART Initialization



NOTE: Initializing or reconfiguring the eUSCI_A module

The recommended eUSCI_A initialization/reconfiguration process is:

1. Set UCSWRST.
2. Initialize all eUSCI_A registers with UCSWRST = 1 (including UCAxCTL1).
3. Configure ports.
4. Clear UCSWRST with software.
5. Enable interrupts (optional) with UCRXIE or UCTXIE.

Figure 10. 3– UART Initialization

Example sequence for UART initialization is listed below:

1. Set UCSWRST bit
2. Configure options such as parity, MSB/LSB first, character length, and stop bit select
3. Configure communication method of eUSCI_A module as UART and asynchronous
4. Select desired clock source for the eUSCI_A module
5. Set up baud rate by configuring MCTLW and BRW registers according to Figure 10.5
6. Clear UCSWRST bit
7. Enable interrupts, if program uses interrupt method

Transmit operation

The eUSCI_A module is enabled by clearing the UCSWRST bit (in UCAxCTLW0 Register) and the transmitter is ready and in an idle state. A transmission is initiated by writing data to UCAxTXBUF (transmit buffer). The data is automatically moved to the transmit shift register and on the next BITCLK. At this point, Interrupt Flag UCTXIFG is set indicating transmit buffer UCAxTXBUF is empty. The next data byte can then be written into the buffer. The flag is automatically reset when a write to UCAxTXBUF occurs. This operation can be carried out by using interrupt or polling. In either case, Transmit Interrupt Flag (UCTXIFG) should be checked and make sure that it is set before writing data to the transmit buffer. In the case of using interrupt, checking UCTXIFG flag is necessary since there is only one interrupt request from the UART to NVIC. The request could be triggered by other UART condition such as an error condition.

Receive operation

The eUSCI_A module is enabled by clearing the UCSWRST bit and the receiver is ready and in an idle state. The UCRXIFG interrupt flag is set when a character is received and loaded into UCAxRXBUF. The flag is reset when UCAxRXBUF is read. Similar to transmit mode, this receiving operation can be done by interrupt or by polling. In either case, UCRXIFG flag should be checked before reading the receive buffer.

Example codes

The following code sends 'HELLO' to a remove device and stores received byte to a display buffer array. An infinite loop is used to display the received data on 4 7-segment displays. This code uses polling. This is not a complete program; only the essential instructions are shown.

```
char outmsg[5] = { 'H', 'E', 'L', 'L', 'O' };
char display[4] = { 0, 0, 0, 0};

simple_clock_init();    //initialize clock
configure_display_pins();  // configure the GPIO pins for display
configure_uart_pins();     // configure the GPIO pins for UART
configure_UCA2();         // configure the UART
EUSCI_A2->CTLW0 &= ~BIT0;   //enable UART operation

while (1)      // loop for displaying the received data on 4 7-seg display
 {
  if((EUSCI_A2->IFG & BIT1) && (Tx_count >5))  //Tx_buff empty & more data
       { EUSCI_A2->TXBUF = outmsg[Tx_count]; //also reset the Tx buffer flag
         Tx_count++;
      }

  if(EUSCI_A2->IFG & BIT0)  //Rx_buffer full
       { display[Rx_count] = (EUSCI_A2->RXBUF & 0x0F);//save 4 LSB, reset flag
         Rx_count++;
         if (Rx_count >3) {Rx_count=0;}
      }

  if (LED_number == 0) {LED_number = 3};
  else   {LED_number--};

  P4->OUT = 0xff;  // Blank all digits
  P8->OUT = ~(0b00000100<< LED_number);
  P4->OUT=sseg_table[display[LED_number]];
 }
```

The following code sends 'HELLO' to a remove device and stores received byte to a display buffer array. An infinite loop is used to display the received data on 4 7-segment displays. This code uses interrupt. This is not a complete program; only the essential instructions are shown.

```
char outmsg[5] = { 'H', 'E', 'L', 'L', 'O' };
char display[4] = { 0, 0, 0, 0};


simple_clock_init();    //initialize clock
configure_NVIC();       // initialize interrupt controller
configure_display_pins();  // configure the GPIO pins for display
configure_uart_pins();     // configure the GPIO pins for UART
configure_UCA2();         // configure the UART
EUSCI_A2->CTLW0 &= ~BIT0;   //enable UART operation

EUSCI_A2->TXBUF = outmsg[0];   // send the first byte to Tx buffer register
Tx_count =1;                   // initialize the byte counter.
                                 //The rest of the message is sent in ISR.

while (1)                 // Infinite loop for displaying the received data
   {
       if (LED_number == 0) LED_number = 3;
      else   LED_number--;
       P4->OUT = 0xff;  // Blank all digits
       P8->OUT = ~ (0b00000100<< LED_number);
       P4->OUT=sseg_table[display[LED_number]];
   }


void EUSCIA2_IRQHandler(void)
{
    uint8_t iv_val = EUSCI_A2->IV; //read IV register

    if (iv_val & 0x4) //if transmit buffer empty
    {
        if(Tx_count < 5)                // if more to send.
          { EUSCI_A2->TXBUF = outmsg[Tx_count];
            Tx_count++;
         }

    if (iv_val & 0x2) //if receive buffer full
    {
        display[Rx_count] = (EUSCI_A2->RXBUF & 0x0F); //save to display array
        Rx_count++;
         if (Rx_count >3) {Rx_count=0;}
     }
}
```

10.4 UART Baud-Rate Generation and Configuration

As shown in Figure 10.2, two shift-registers are used: one for transmitting data and one for receiving data. The clock signals for these shift-registers determine the speed at which the date bits are shifted into the shift-registers and hence frequency of these clocks (BITCLK) are the bit rate or baud rate. The circuit that generate these clocks are called 'baud rate generator'. The input clock to the baud rate generator (BRCLK, see Figure 10.2) must be at least several times higher than the baud rate in order to achieve the following two objectives:

(1) The BRLCK should be able to be divided down to match the desired baud rate. For example, if the desired baud rate is 9600Hz and BRCLK is only 32768Hz. The best the baud rate generator can do is divide 32768 by 3 and obtain a bit rate of 10923Hz which is substantially different from the desired 9600Hz. If BRCL is 4MHz, the generator can divide it by 417 and obtain 9592Hz which is much closer to 9600Hz.

(2) This serial communication is asynchronous. Namely, the transmitting device and the receiving devices do not use a common clock signal. So, it is important that the receiver has a much higher frequency clock (BRCLK) than the incoming data rate so that the receiver can operate at a higher 'time resolution' in order to synchronize its receiving operation with the input bit stream. Figure 10.4 shows the case where three BRCLK cases are considered. As shown, a higher receiver clock frequency allows for a better alignment of the sampling action (must takes place at an edge of the clock signal) with the incoming data frame.

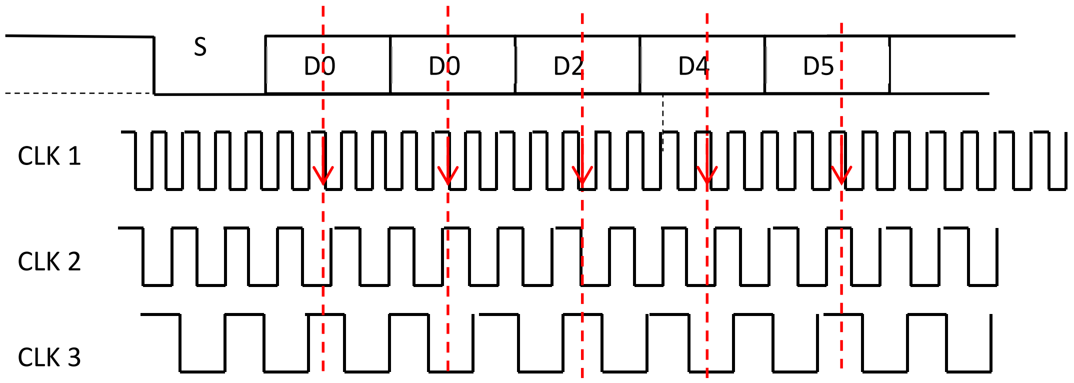


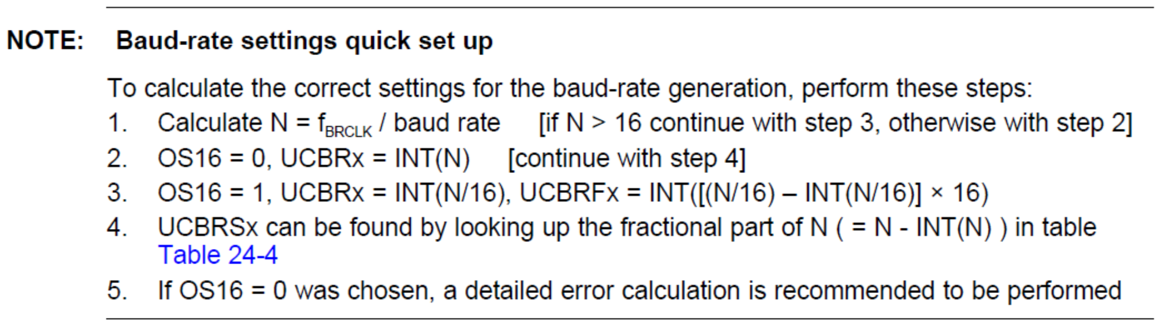
Figure 10.4

The generator's clock BRCLK is selected from UCLK, ACLK, or SMCLK. Once BRCLK is determined, there are four parameters that need to be set to configure the generator to generate the desired baud rate: UCBRx, UCBRFx, UCBRSx, and UCOS16. All these parameters are in two registers: UCAxBRW and UCAxMCTLW (see Section 10.5).

Standard baud rates include 110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200, 128000 and 256000 bits per second. In eUSCI_A, there are two ways of generating baud-rate: low- frequency and oversampling depending on the ratio (N) between BRCLK and the desired baud rate.

$$N = f_{BRCLK} / \text{baud rate}$$

If N is less than 16, the generator should be set to operate in the low-frequency mode (setting UCOS16 to 0). Otherwise, it should be set to oversampling mode (UCOS16=1). Figure 10.5 shows the steps in determining the values for UCBRx, UCBRFx, and UCBRSx.



NOTE: Baud-rate settings quick set up

To calculate the correct settings for the baud-rate generation, perform these steps:

1. Calculate N = f_{BRCLK} / baud rate [if N > 16 continue with step 3, otherwise with step 2]
2. OS16 = 0, UCBRx = INT(N) [continue with step 4]
3. OS16 = 1, UCBRx = INT(N/16), UCBRFx = INT([(N/16) – INT(N/16)] × 16)
4. UCBRSx can be found by looking up the fractional part of N (= N - INT(N)) in table Table 24-4
5. If OS16 = 0 was chosen, a detailed error calculation is recommended to be performed

Figure 10.5– Setting Up Baud Rate

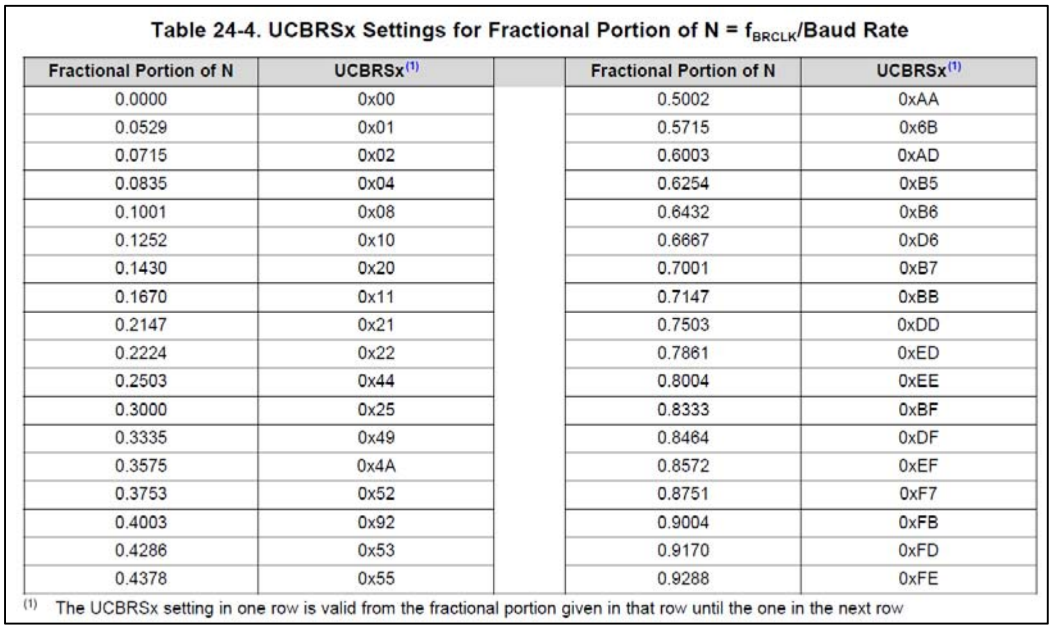


Table 24-4. UCBRSx Settings for Fractional Portion of N = f_{BRCLK}/Baud Rate

Fractional Portion of N	UCBRSx[1]		Fractional Portion of N	UCBRSx[1]
0.0000	0x00		0.5002	0xAA
0.0529	0x01		0.5715	0x6B
0.0715	0x02		0.6003	0xAD
0.0835	0x04		0.6254	0xB5
0.1001	0x08		0.6432	0xB6
0.1252	0x10		0.6667	0xD6
0.1430	0x20		0.7001	0xB7
0.1670	0x11		0.7147	0xBB
0.2147	0x21		0.7503	0xDD
0.2224	0x22		0.7861	0xED
0.2503	0x44		0.8004	0xEE
0.3000	0x25		0.8333	0xBF
0.3335	0x49		0.8464	0xDF
0.3575	0x4A		0.8572	0xEF
0.3753	0x52		0.8751	0xF7
0.4003	0x92		0.9004	0xFB
0.4286	0x53		0.9170	0xFD
0.4378	0x55		0.9288	0xFE

(1) The UCBRSx setting in one row is valid from the fractional portion given in that row until the one in the next row

Figure 10.6 – UCBRSx Settings for Fractional Portion of N

Example: If the frequency of BRCLK is 12 MHz and the desired baud rate is 9600:

1. N = 12MHz / 9600 = 1250.0. Because N > 16, we go to step 3
2. Skipped
3. OS16 = 1, UCBRx = int (1250 / 16) = int(78.125)=78. UCBRFx = int([78.125 -78] * 16) = 2
4. UCBRSx = 0x0, because by looking at the table, since N=1250.0, the fractional part of it is 0.
5.

Example: If the frequency of BRCLK is 120kHz and the desired baud rate is 9600:

1. N = 120000 / 9600 = 10.5. Because N < 16, we go to step 2
2. OS16 = 0, UCBRx = int (10.5) =10.
3. Skipped
4. Since N=10.5, the fractional part of it is 0.5, UCBRSx = 0xAA from the table,.

The receiver samples the input line (RXD) at the middle of the bit time period to determine the state (0 or 1) of the bit. eUSCI samples the input line 3 times at or near the center of the bit period and, if three samples don't agree, the majority state is taken. This operation is transparent to software. Figure 10.7 shows the timing relationship between BRCLK and the BITCLK (the clock of the shift-register, i.e., the baud rate frequency). This shows the importance of having a BRCLK that has a much higher frequency than the baud rate, the BITCLK.

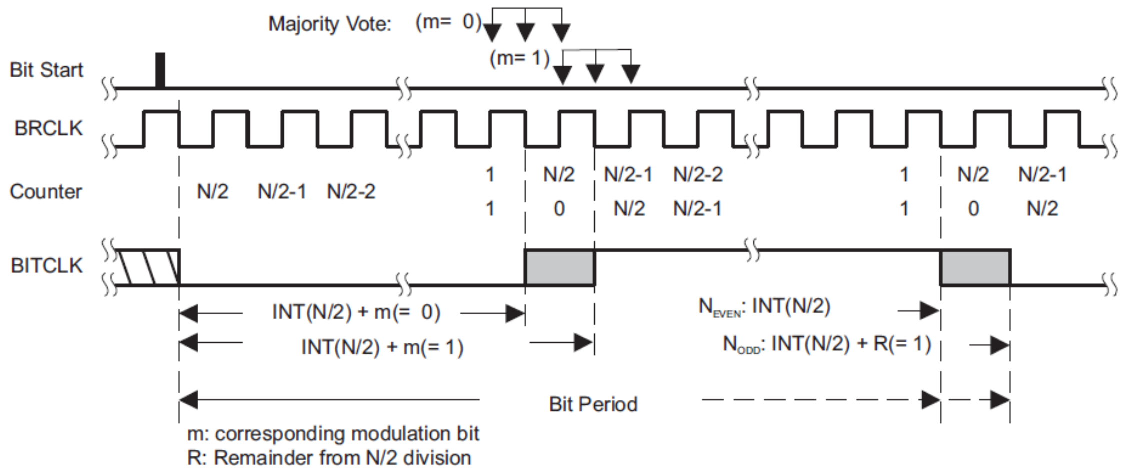


Figure 10.7 – BITCLK Baud-Rate Timing

10.5 UART Registers

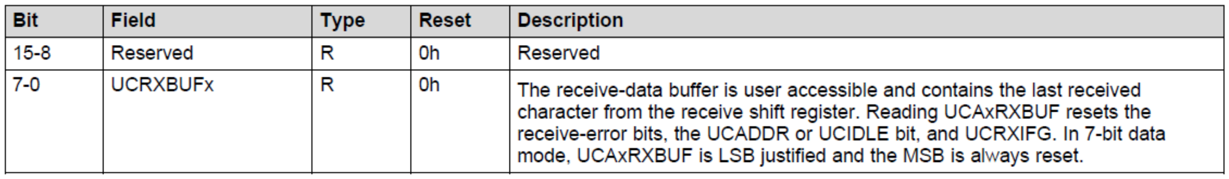
Figure 10.8 shows all registers in a eUSCI module.

Offset	Acronym	Register Name	Section
00h	UCAxCTLW0	eUSCI_Ax Control Word 0	
00h	UCAxCTL1	eUSCI_Ax Control 1	General Configuration Register
01h	UCAxCTL0	eUSCI_Ax Control 0	
02h	UCAxCTLW1	eUSCI_Ax Control Word 1	
06h	UCAxBRW	eUSCI_Ax Baud Rate Control Word	
06h	UCAxBR0	eUSCI_Ax Baud Rate Control 0	Baud Rate Configuration Registers.
07h	UCAxBR1	eUSCI_Ax Baud Rate Control 1	
08h	UCAxMCTLW	eUSCI_Ax Modulation Control Word	
0Ah	UCAxSTATW	eUSCI_Ax Status	Status Register
0Ch	UCAxRXBUF	eUSCI_Ax Receive Buffer	Transmit & Receive Buffer Registers
0Eh	UCAxTXBUF	eUSCI_Ax Transmit Buffer	
10h	UCAxABCTL	eUSCI_Ax Auto Baud Rate Control	
12h	UCAxIRCTL	eUSCI_Ax IrDA Control	Special function register
12h	UCAxIRTCTL	eUSCI_Ax IrDA Transmit Control	
13h	UCAxIRRCTL	eUSCI_Ax IrDA Receive Control	
1Ah	UCAxIE	eUSCI_Ax Interrupt Enable	Interrupt Registers
1Ch	UCAxIFG	eUSCI_Ax Interrupt Flag	
1Eh	UCAxIV	eUSCI_Ax Interrupt Vector	

Figure 10.8 – eUSCI Registers

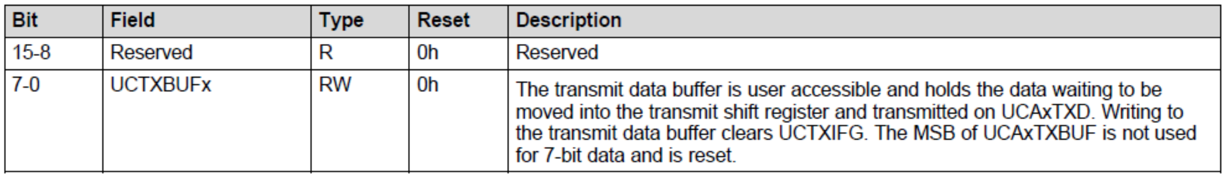
Receive Buffer Register (UCAxRXBUF) and Transmit Buffer Register (UCAxTXBUF)

These two registers are shift-registers for receiving (UCAxRXBUF) or transmitting (UCAxTXBUF) data purpose.



Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved
7-0	UCRXBUFx	R	0h	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 UCRXIFG. In 7-bit data mode, UCAxRXBUF is LSB justified and the MSB is always reset.

Figure 10.9 – UCAxRXBUF Register



Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved
7-0	UCTXBUFx	RW	0h	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 UCTXIFG. The MSB of UCAxTXBUF is not used for 7-bit data and is reset.

Figure 10.10 – UCAxTXBUF Register

Control Word Registers (UCAxCTLW0/ UCAxCTLW1)

There are two 16-bit control registers, named UCAxCTLW0 and UCAxCTLW1. The important bits in the UCAxCTLW0 registers (see Figure 10.11) for our purpose are:

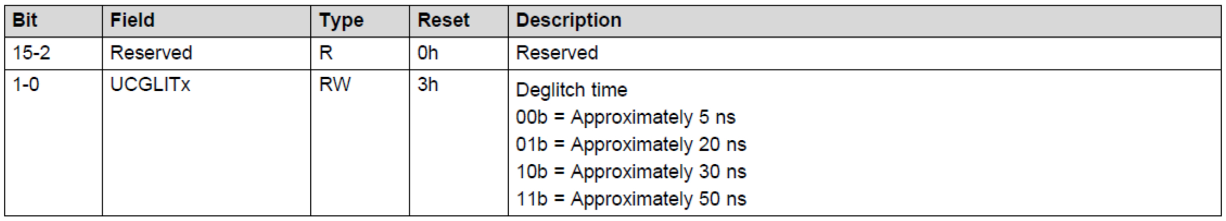
- Parity enable (**UCPEN**) and parity select (**UCPAR**) – The parity enable bit enables or disables the parity bit in the UART message frame, and parity select bit configures the parity as even or odd.
- MSB first select (**UCMSB**) – This bit controls the direction of the receive and transmit shift register as either MSB first or LSB first. LSB first is the standard.
- Character length select (**UC7BIT**) – This bit configures data length as either 7-bit or 8-bit.
- Stop bit select (**UCSPB**) – This bit configures the number of stop bits as either 1 or 2.
- eUSCI_A mode select (**UCMODEx**) – Set these 2 bits to 00b for UART operation. Other options involve auto baud rate detection, or multiprocessor operation.
- Synchronous mode enable (**UCSYNC**) – This bit selects between synchronous or asynchronous mode. Since UART is asynchronous, this bit should be set to 0.
- eUSCI_A clock source select (**UCSSELx**) - These bits select the BRCLK source clock.
- Software reset (**UCSWRST**) – When enabled, the eUSCI_A module is reset. This bit should be set at the start of initialization and cleared at the end of initialization, after all the registers are already configured.

Bit	Field	Type	Reset	Description
15	UCPEN	RW	0h	Parity enable 0b = Parity disabled 1b = Parity enabled. Parity bit is generated (UCAxTXD) and expected (UCAxRXD). In address-bit multiprocessor mode, the address bit is included in the parity calculation.
14	UCPAR	RW	0h	Parity select. UCPAR is not used when parity is disabled. 0b = Odd parity 1b = Even parity
13	UCMSB	RW	0h	MSB first select. Controls the direction of the receive and transmit shift register. 0b = LSB first 1b = MSB first
12	UC7BIT	RW	0h	Character length. Selects 7-bit or 8-bit character length. 0b = 8-bit data 1b = 7-bit data
11	UCSPB	RW	0h	Stop bit select. Number of stop bits. 0b = One stop bit 1b = Two stop bits
10-9	UCMODEx	RW	0h	eUSCI_A mode. The UCMODEx bits select the asynchronous mode when UCSYNC = 0. 00b = UART mode 01b = Idle-line multiprocessor mode 10b = Address-bit multiprocessor mode 11b = UART mode with automatic baud-rate detection
8	UCSYNC	RW	0h	Synchronous mode enable 0b = Asynchronous mode 1b = Synchronous mode
7-6	UCSSELx	RW	0h	eUSCI_A clock source select. These bits select the BRCLK source clock. 00b = UCLK 01b = ACLK 10b = SMCLK 11b = SMCLK
5	UCRXEIE	RW	0h	Receive erroneous-character interrupt enable 0b = Erroneous characters rejected and UCRXIFG is not set. 1b = Erroneous characters received set UCRXIFG.
4	UCBRKIE	RW	0h	Receive break character interrupt enable 0b = Received break characters do not set UCRXIFG. 1b = Received break characters set UCRXIFG.
3	UCDORM	RW	0h	Dormant. Puts eUSCI_A into sleep mode. 0b = Not dormant. All received characters set UCRXIFG. 1b = Dormant. Only characters that are preceded by an idle-line or with address bit set UCRXIFG. In UART mode with automatic baud-rate detection, only the combination of a break and synch field sets UCRXIFG.
2	UCTXADDR	RW	0h	Transmit address. Next frame to be transmitted is marked as address, depending on the selected multiprocessor mode. 0b = Next frame transmitted is data. 1b = Next frame transmitted is an address.
1	UCTXBRK	RW	0h	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. 0b = Next frame transmitted is not a break. 1b = Next frame transmitted is a break or a break/synch.
0	UCSWRST	RW	1h	Software reset enable 0b = Disabled. eUSCI_A reset released for operation. 1b = Enabled. eUSCI_A logic held in reset state.

Figure 10.11 – UCAxCTLW0 Register

Control Word Register 1 (UCAxCTLW1)

This register configures deglitch time. As mentioned earlier in the document, any pulse shorter than the deglitch time is ignored.

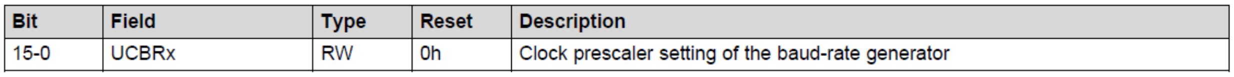


Bit	Field	Type	Reset	Description
15-2	Reserved	R	0h	Reserved
1-0	UCGLITx	RW	3h	Deglitch time 00b = Approximately 5 ns 01b = Approximately 20 ns 10b = Approximately 30 ns 11b = Approximately 50 ns

Figure 10.12– UCAxCTLW1 Register

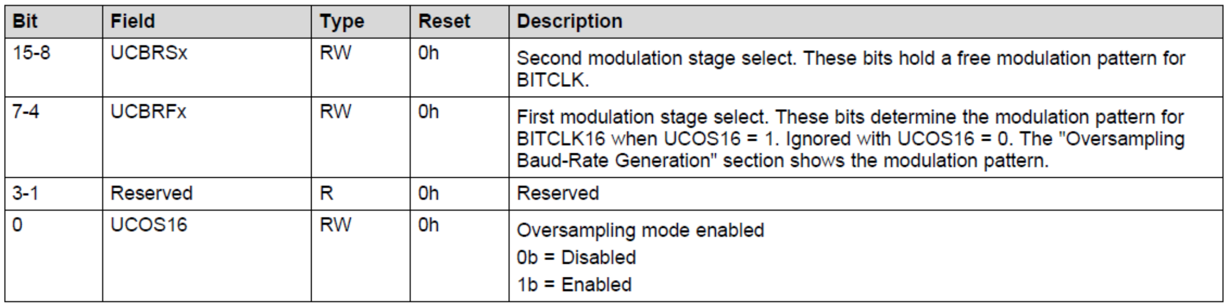
Baud Rate Control and Modulation Control Registers (UCAxBRW and UCAxMCTLW)

These two registers (Figure 10.13 and Figure 10.14) together configure UART baud rate according to Figure 10.5. Note that UCBRx can be modified only when UCSWRST=1 and UCBRFx is ignored if UCOS16=0, i.e., when N<16.



Bit	Field	Type	Reset	Description
15-0	UCBRx	RW	0h	Clock prescaler setting of the baud-rate generator

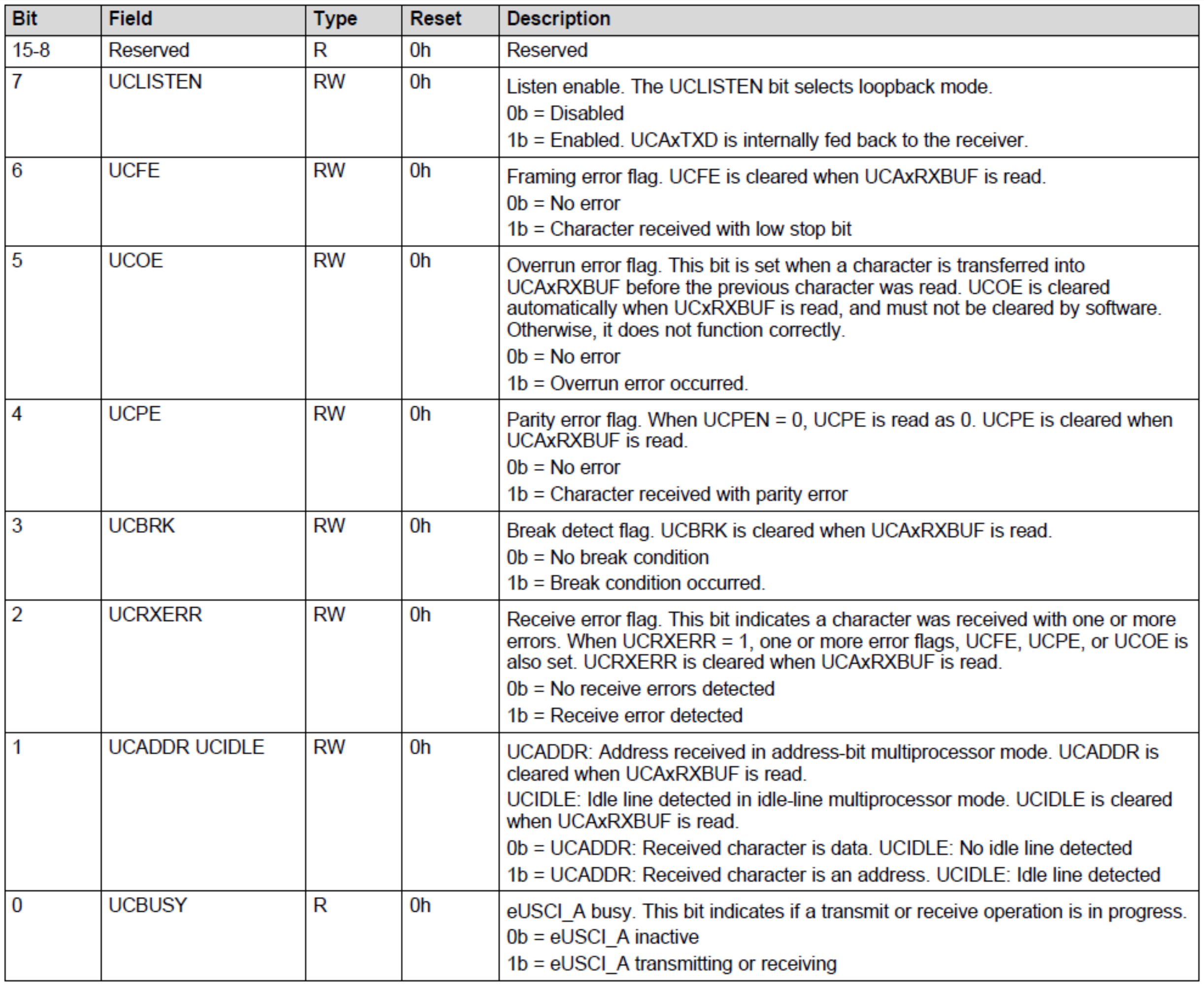
Figure 10.13 UCAxBRW register (modify only when UCSWRST=1)



Bit	Field	Type	Reset	Description
15-8	UCBRSx	RW	0h	Second modulation stage select. These bits hold a free modulation pattern for BITCLK.
7-4	UCBRFx	RW	0h	First modulation stage select. These bits determine the modulation pattern for BITCLK16 when UCOS16 = 1. Ignored with UCOS16 = 0. The "Oversampling Baud-Rate Generation" section shows the modulation pattern.
3-1	Reserved	R	0h	Reserved
0	UCOS16	RW	0h	Oversampling mode enabled 0b = Disabled 1b = Enabled

Figure 10.14 – UCAxMCTLW (bottom) Register

Status Register (UCAxSTATW)

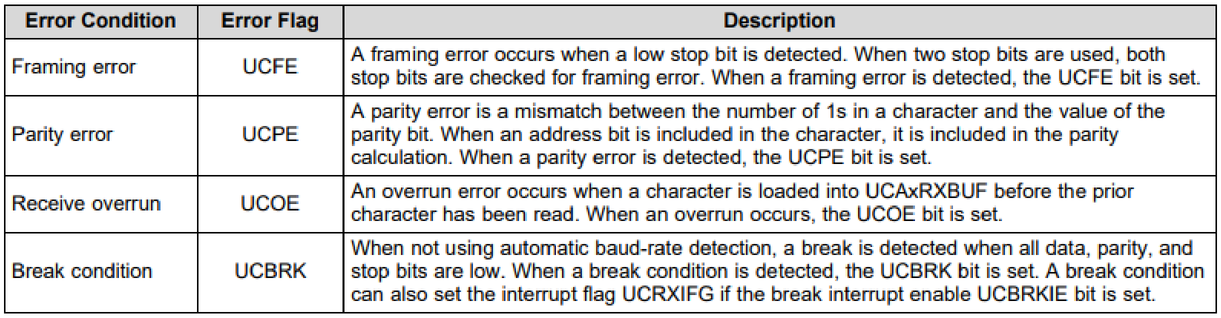


Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved
7	UCLISTEN	RW	0h	Listen enable. The UCLISTEN bit selects loopback mode. 0b = Disabled 1b = Enabled. UCAxTXD is internally fed back to the receiver.
6	UCFE	RW	0h	Framing error flag. UCFE is cleared when UCAxRXBUF is read. 0b = No error 1b = Character received with low stop bit
5	UCOE	RW	0h	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 UCxRXBUF is read, and must not be cleared by software. Otherwise, it does not function correctly. 0b = No error 1b = Overrun error occurred.
4	UCPE	RW	0h	Parity error flag. When UCPEN = 0, UCPE is read as 0. UCPE is cleared when UCAxRXBUF is read. 0b = No error 1b = Character received with parity error
3	UCBRK	RW	0h	Break detect flag. UCBRK is cleared when UCAxRXBUF is read. 0b = No break condition 1b = Break condition occurred.
2	UCRXERR	RW	0h	Receive error flag. This bit indicates a character was received with one or more errors. When UCRXERR = 1, one or more error flags, UCFE, UCPE, or UCOE is also set. UCRXERR is cleared when UCAxRXBUF is read. 0b = No receive errors detected 1b = Receive error detected
1	UCADDR UCIDLE	RW	0h	UCADDR: Address received in address-bit multiprocessor mode. UCADDR is cleared when UCAxRXBUF is read. UCIDLE: Idle line detected in idle-line multiprocessor mode. UCIDLE is cleared when UCAxRXBUF is read. 0b = UCADDR: Received character is data. UCIDLE: No idle line detected 1b = UCADDR: Received character is an address. UCIDLE: Idle line detected
0	UCBUSY	R	0h	eUSCI_A busy. This bit indicates if a transmit or receive operation is in progress. 0b = eUSCI_A inactive 1b = eUSCI_A transmitting or receiving

Figure 10.15 – UCAxSTATW Register

The important bits in this register are:

- Receive errors (**UCFE**, **UCOE**, **UCPE**, and **UCBRK**). All of these errors are cleared when UCAxRXBUF is read. See Figure 10.16 for descriptions of these errors.



Error Condition	Error Flag	Description
Framing error	UCFE	A framing error occurs when a low stop bit is detected. When two stop bits are used, both stop bits are checked for framing error. When a framing error is detected, the UCFE bit is set.
Parity error	UCPE	A parity error is a mismatch between the number of 1s in a character and the value of the parity bit. When an address bit is included in the character, it is included in the parity calculation. When a parity error is detected, the UCPE bit is set.
Receive overrun	UCOE	An overrun error occurs when a character is loaded into UCAxRXBUF before the prior character has been read. When an overrun occurs, the UCOE bit is set.
Break condition	UCBRK	When not using automatic baud-rate detection, a break is detected when all data, parity, and stop bits are low. When a break condition is detected, the UCBRK bit is set. A break condition can also set the interrupt flag UCRXIFG if the break interrupt enable UCBRKIE bit is set.

Figure 10.16– Error Flags

- Receive error flag (**UCRXERR**) - This bit indicates a character was received with one or more errors. If this flag is implemented, this bit should be read prior to reading the RXBUF, because as soon as RXBUF is read this flag will reset automatically.
- eUSCI_A busy bit (**UCBUSY**) – This bit can be polled to see if UART module is busy.

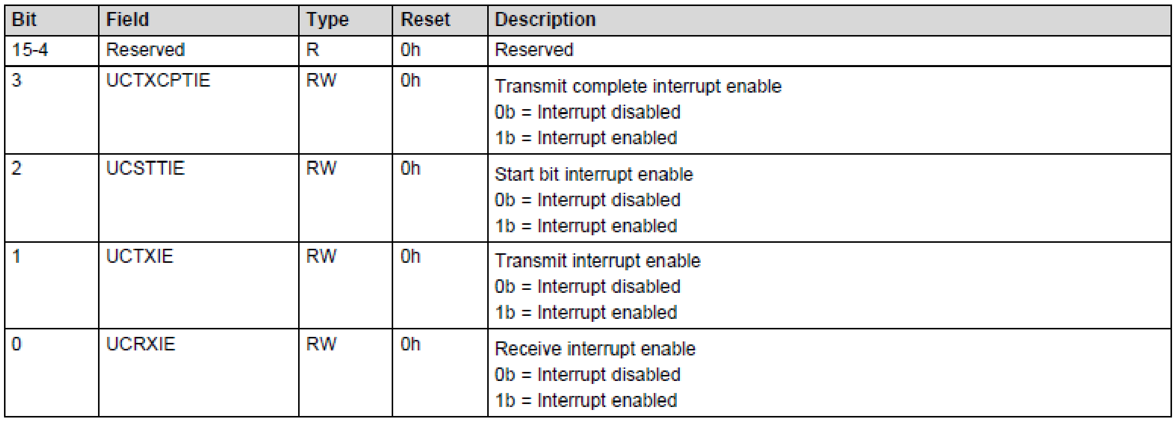
Auto Baud Rate Control Register (UCAxABCTL) and IrDA Control Word Register (UCAxIRCTL)

Auto baud rate and IrDA are not standard UART function and will not be discussed in this class.

10.6 Interrupt Registers and Operation

There are three Interrupt registers in the eUSCI_A UART module: Interrupt Enable Register (UCAxIE), Interrupt Flag Register (UCAxIFG), Interrupt Vector Register (UCAxIV). They are described below.

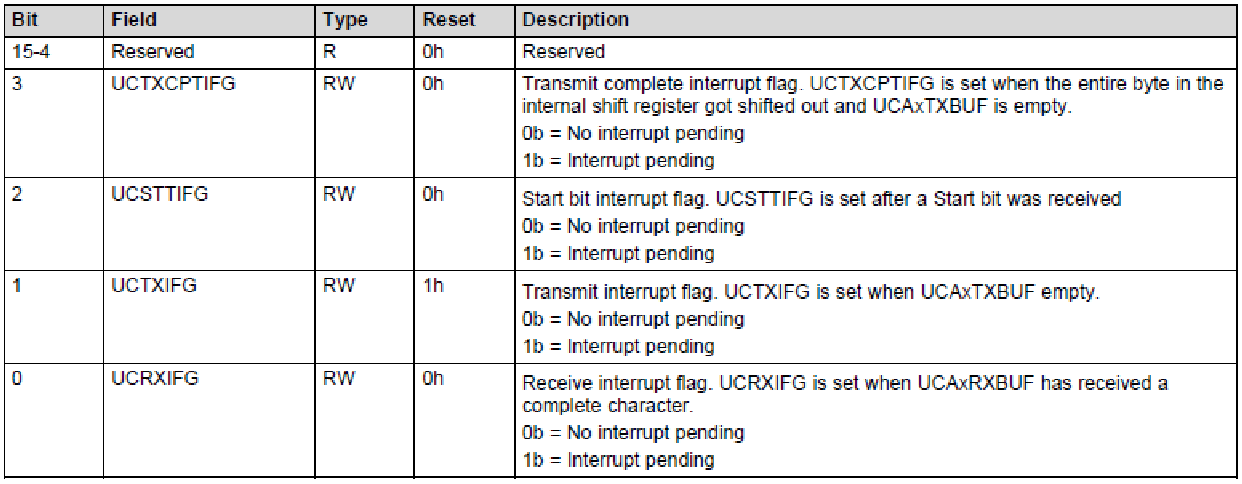
Interrupt Enable Register (UCAxIE) – similar to all other interrupt enable registers, this register enables and disables interrupts on its corresponding bits. An interrupt has to be enabled before it is used.



Bit	Field	Type	Reset	Description
15-4	Reserved	R	0h	Reserved
3	UCTXCPTIE	RW	0h	Transmit complete interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
2	UCSTTIE	RW	0h	Start bit interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
1	UCTXIE	RW	0h	Transmit interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
0	UCRXIE	RW	0h	Receive interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled

Figure 10. 17– Interrupt Enable Register (UCAxIE)

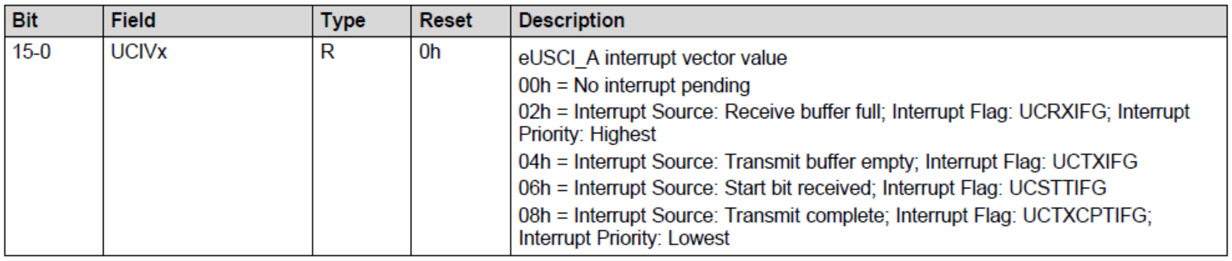
Interrupt Flag Register (UCAxIFG) – The register with actual interrupts. Reading the corresponding bit in this register will tell the user if the flag is pending or not. Writing a "0" to the corresponding bit in this register clears the flag. These flags are useful even if interrupt is not used. For example, UCTXIFG is also known as 'transmit buffer empty' flag and 'UCRXIFG' is also known as 'Receive buffer full' flag. User program can poll these flags and send/receive data at the proper time. This is explained in Section 10.8.



Bit	Field	Type	Reset	Description
15-4	Reserved	R	0h	Reserved
3	UCTXCPTIFG	RW	0h	Transmit complete interrupt flag. UCTXCPTIFG is set when the entire byte in the internal shift register got shifted out and UCAxTXBUF is empty. 0b = No interrupt pending 1b = Interrupt pending
2	UCSTTIFG	RW	0h	Start bit interrupt flag. UCSTTIFG is set after a Start bit was received 0b = No interrupt pending 1b = Interrupt pending
1	UCTXIFG	RW	1h	Transmit interrupt flag. UCTXIFG is set when UCAxTXBUF empty. 0b = No interrupt pending 1b = Interrupt pending
0	UCRXIFG	RW	0h	Receive interrupt flag. UCRXIFG is set when UCAxRXBUF has received a complete character. 0b = No interrupt pending 1b = Interrupt pending

Figure 10. 18– Interrupt Flag Register (UCAxIFG)

Interrupt Vector Register (UCAxIV) - This interrupt Vector register works in exactly same way as the interrupt vector register in GPIO port or ADC14 – multiple interrupt requests in eUSCI_A are OR'ed and a single interrupt request is sent to NVIC. The priority and the automatic reset mechanism is the same as that for GPIO port or ADC 14.



Bit	Field	Type	Reset	Description
15-0	UCIVx	R	0h	eUSCI_A interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: Receive buffer full; Interrupt Flag: UCRXIFG; Interrupt Priority: Highest 04h = Interrupt Source: Transmit buffer empty; Interrupt Flag: UCTXIFG 06h = Interrupt Source: Start bit received; Interrupt Flag: UCSTTIFG 08h = Interrupt Source: Transmit complete; Interrupt Flag: UCTXCPTIFG; Interrupt Priority: Lowest

Figure 10.19 – UCAxIV Register