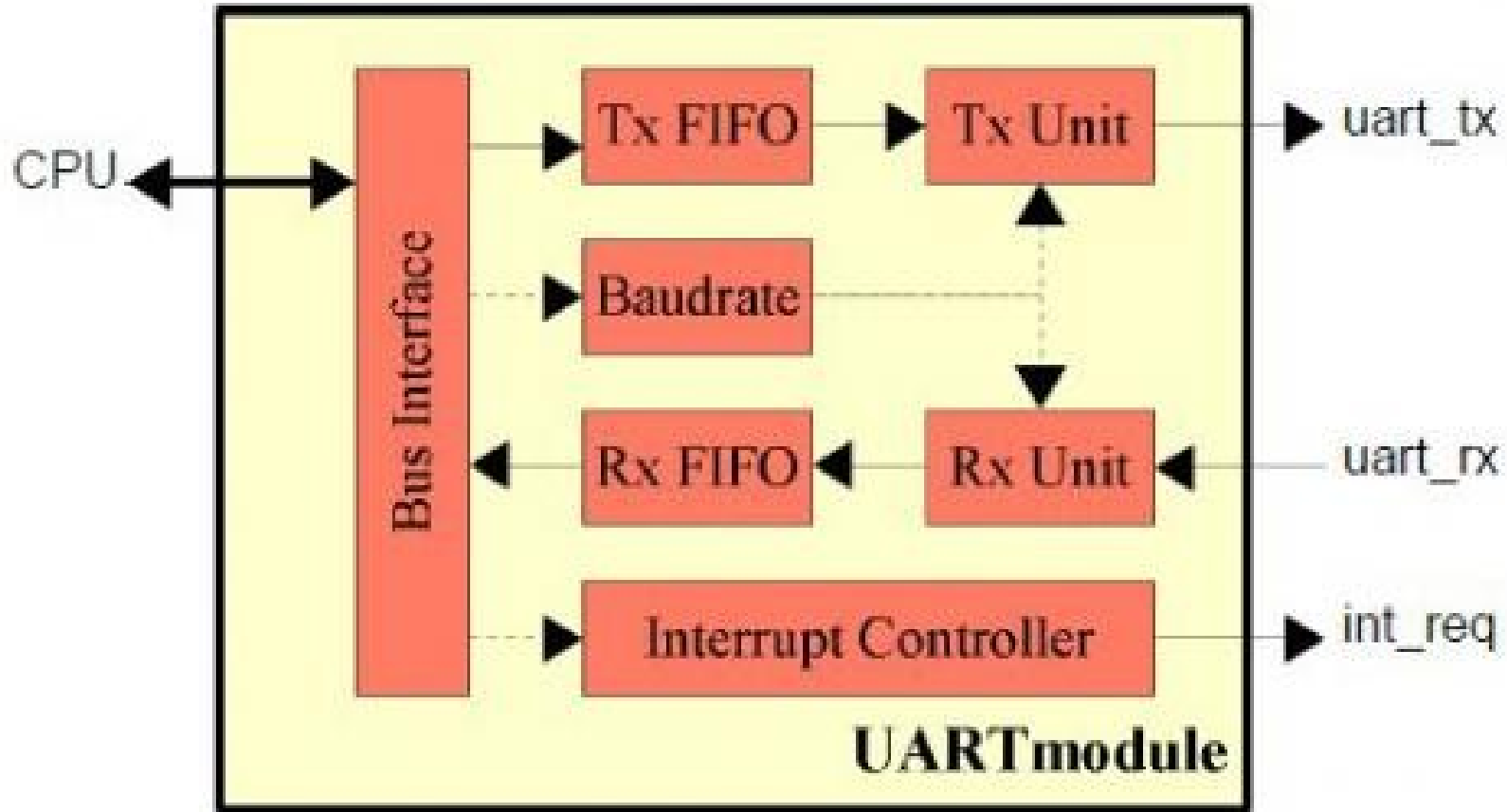


**UART – Serial Communications**

**MSP430FR2433 eUSCI interface**

**Universal Asynchronous Receive and Transmit**



**UART – Serial Communication - Block Diagram**

Serial communication stands for the process of sending data one bit at a time, sequentially, through the bus or communication channel.

- Shift registers are a fundamental method of conversion between serial and parallel forms of data, and therefore are a part of every UART.
- Serial communication has two primary forms, (synchronous and asynchronous) there are also two forms of UART, known as:
  - **UART - Universal Asynchronous Receiver/Transmitter**
  - **USART - Universal Synchronous/Asynchronous Receiver/Transmitter**
- The asynchronous type of transmitter generates a data clock internally depending on the MCU clock cycles.
- There is no incoming clock signal associated with the data. To achieve proper communication between two modules, both of them have to have the same clock cycle length (rate), which means that they must work on the same BAUD rate.
- **BAUD:** a unit used to measure the speed of signaling or data transfer, equal to the number of pulses or bits per second: baud rate.

The serial communication goes through independent ends of a line :

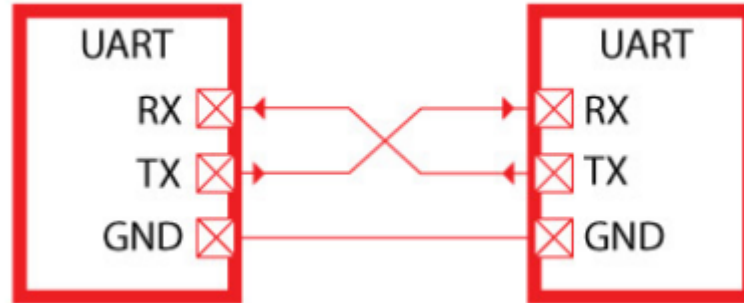
- TX (transmission) and
- RX (reception).

Communication can be :

- Simplex - One direction only, transmitter to receiver
- Half Duplex – Devices take turns transmitting and receiving
- Full Duplex - Devices can send and receive at the same time

From the transmitter the TX line is the same as the RX line from the receiver. The data stream always has the same direction through every line.

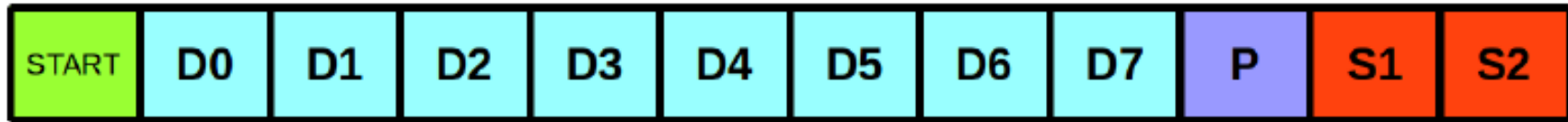
In the idle states lines are pulled high. This allows recognition by the receiver side that there is a transmitter on the other side which is connected to the bus. (Stop condition or bits)

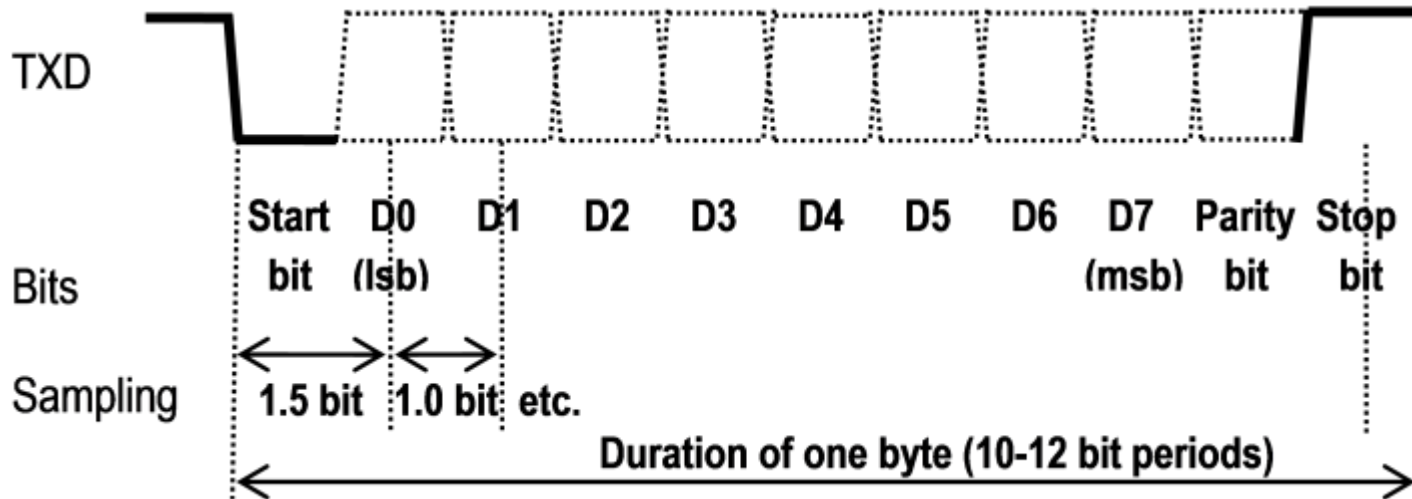


Data are transmitted in 8 bit bytes (characters) using the serial protocol **RS232**  
Com ports are used on personal computer for serial communications

Frame starts with "**Start Bit**" which is logic low and is used to signal the receiver that a new frame is coming. Next **5-9** bits carry the data and parity bit. The part of the frame is configurable depending on the code set employed. After the data, is the **parity bit**, if the data length is not 9 bits. The parity bit is not used very often, but in the case of noisy buses, parity bits can be a good method to avoid reception of the wrong data packets. It represents the sum of the data bits. Even parity means when the sum is even this bit will be 1, and in the case of odd this bit will be 0. If odd parity is used, the bit values will be reversed, (even sum is 0, and odd is 1 ).

The end of the frame can have one or two **stop bits**. Stop bits are always logical high. The difference in the logic between start and stop bits allows break detection on the bus. These bits are also called synchronization bits because they mark the beginning and the end of the packet.





The falling edge of the start bit enables the baud rate generator and the UART state machine checks for a valid start bit.

If no valid start bit is detected the UART state machine returns to its idle state and the baud rate generator is turned off again.

If a valid start bit is detected, a full character is received by the Rx.

# Terminology:

The baud rate specifies how fast data is sent over the bus and it is specified in bits per second. It is allowed to choose any speed for the baud rate but actually there are values that are used as a standard. The most common and standardized value is **9600**. Other standard baud rates are : 1200, 2400, 4800, 19200, 38400, 57600 and 115200. Baud rates higher than 115200 can be used but usually that causes a lot of errors in transmission.

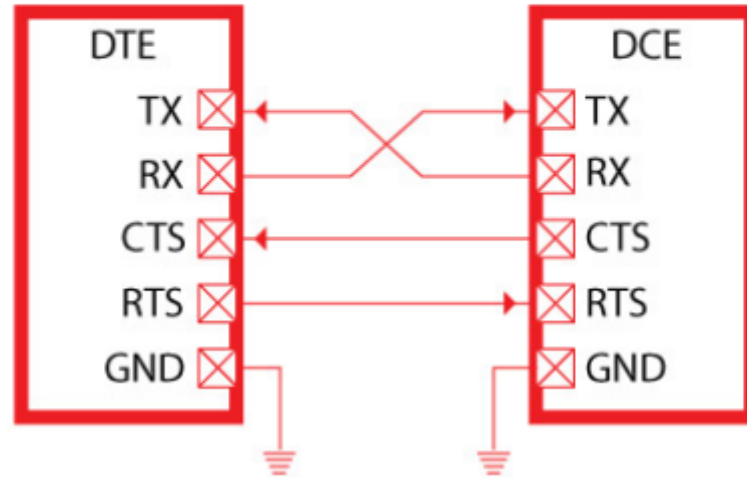
The shortcuts used for describing the UART bus configuration are usually in form :

<DATA\_BITS><STOP\_BITS>

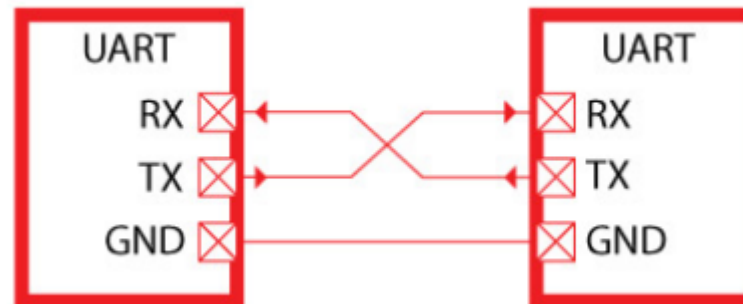
Examples :

- **9600 8N1**      9600 baud rate, 8 bits of data, No parity, 1 stop bit
- **115200 8E2**    115200 baud rate, 8 bits of data, Even parity, 2 stop bits

Data flow control in RS-232 introduced the usage of two additional lines known as **RTS** and **CTS**. These two lines from the view of an embedded developer are nothing more than simple GPIO pins, but usage of them from the view of a software developer can bring a lot of benefits - we will discuss this later. To properly understand what data flow control is we have to know what **DTE** and **DCE** are.



With Data Flow Control



Without Data Flow Control





# eUSCI Interrupts in MSP430

## IE2

UCA0TXIE	Used to enable/disable the transmit interrupt.
UCA0RXIE	Used to enable/disable the receive interrupt.

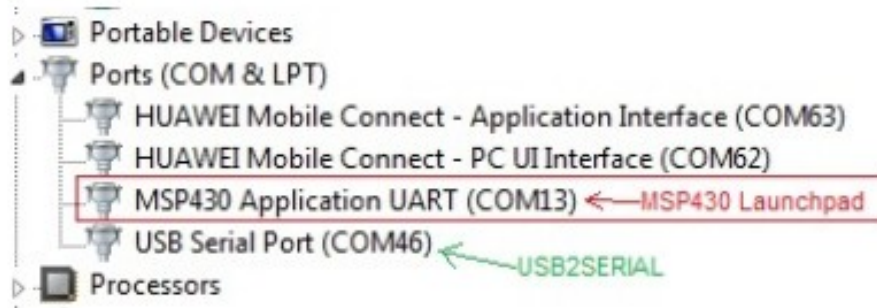
## IFG2

UCA0TXIFG	USCI_A0 transmit interrupt flag is set when UCA0TXBUF is empty
UCA0RXIFG	USCI_A0 receive interrupt flag is set when UCA0RXBUF have received a complete character.you should <b>remember to clear the flag</b> inside the interrupt service routine manually.

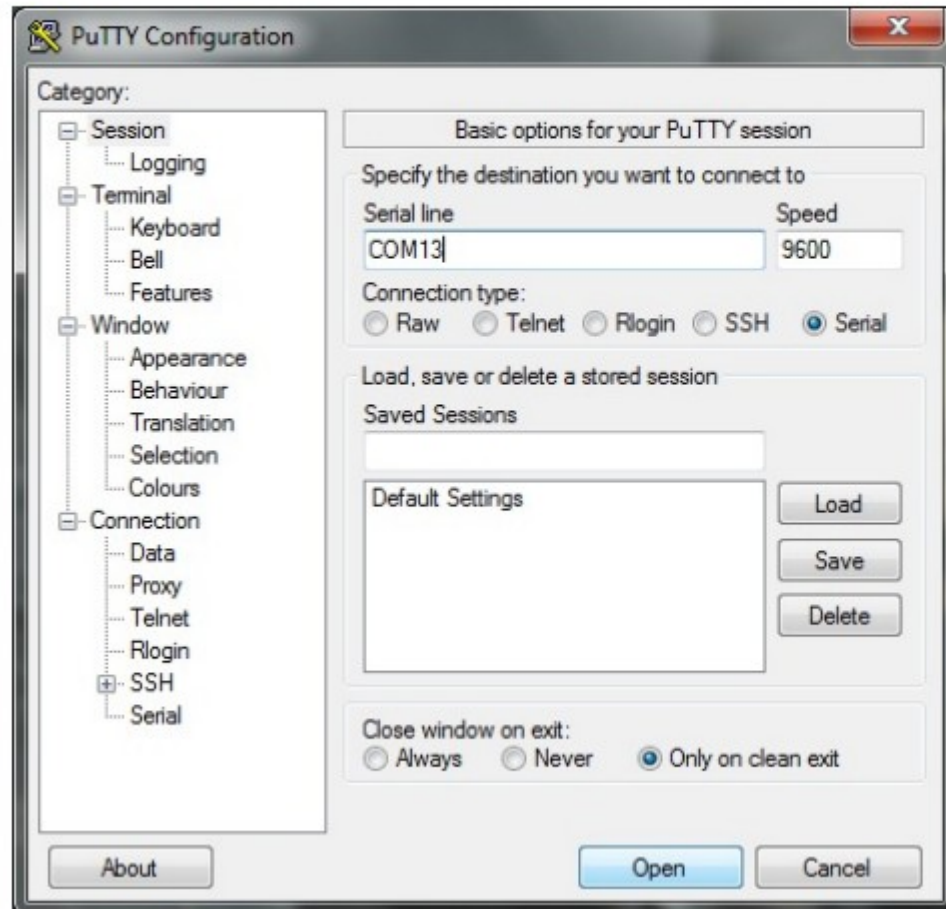
# Device Manager – Windows 10

## Using MSP430 Launchpad Serial Port

In Launchpad, the TUSB3410 chip( inside the emulator section) can act as a USB to Serial converter too.If you check under "**Device Manager**" you can see that as "**MSP430 Application UART COMxx**",highlighted in red (here COM13).The one below that is the COM port number corresponding to the USBto Serial Converter ([USB2SERIAL](#)).



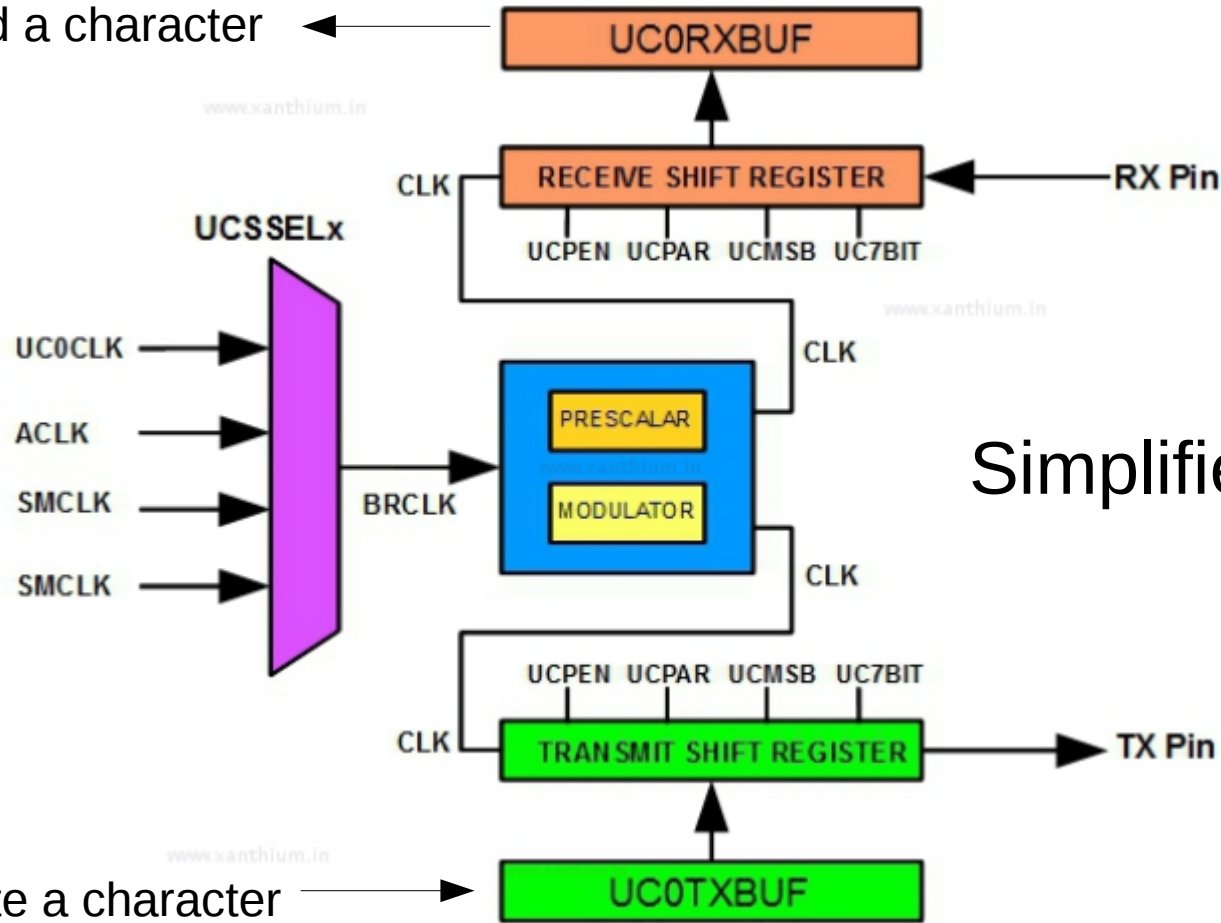
After you have configured the jumpers in the board ,Launch PuTTY or any other terminal program of your choice.Put the COM port number corresponding to your launchpad serial port (here COM13) ,select the baudrate (here 9600) and open the connection.



PIN NAME (P1.x)	x	FUNCTION	CONTROL BITS AND SIGNALS <sup>(1)</sup>		
			P1DIR.x	P1SELx	ADCPCTLx <sup>(2)</sup>
P1.4/UCA0TXD/ UCA0SIMO/TA1.2/TCK/ A4 /VREF+	4	P1.4 (I/O)	I: 0; O: 1	00	0
		UCA0TXD/UCA0SIMO	X	01	0
		TA1.CCI2A	0	10	0
		TA1.2	1		
		A4, VREF+	X	X	1 (x = 4)
		JTAG TCK	X	X	X
P1.5/UCA0RXD/ UCA0SOMI/TA1.1/TMS/ A5	5	P1.5 (I/O)	I: 0; O: 1	00	0
		UCA0RXD/UCA0SOMI	X	01	0
		TA1.CCI1A	0	10	0
		TA1.1	1		
		A5	X	X	1 (x = 5)
		JTAG TMS	X	X	X

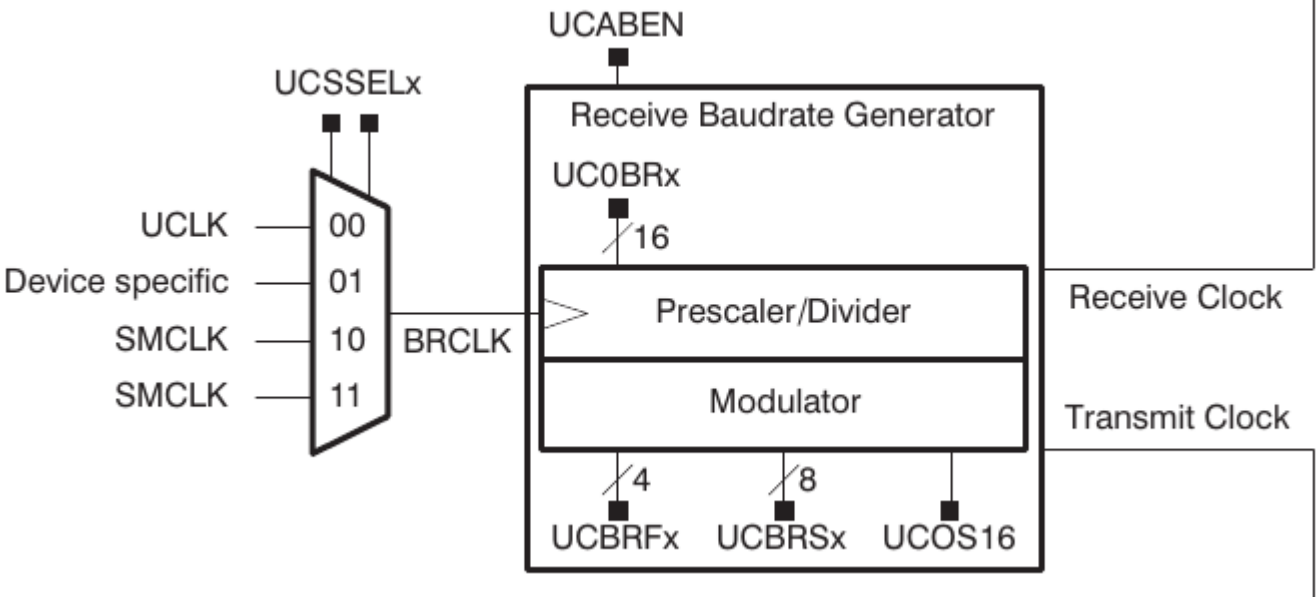
In MSP430FR2433, P1.4 and P1.5 are configured as UCA0TXD and UCA0RXD using their respective PXSEL registers.

Read a character



Simplified Block Diagram

# Real functional diagram in 3 parts – Baud Rate Generator



## mcp430fr243x\_euscia0\_uart\_01.c

```
58 __bis_SR_register(SCG0); // disable FLL
59 CSCTL3 |= SELREF__REFOCLK; // Set REF0 as FLL reference source
60 CSCTL0 = 0; // clear DCO and MOD registers
61 CSCTL1 &= ~(DCORSEL_7); // Clear DCO frequency select bits first
62 CSCTL1 |= DCORSEL_3; // Set DCO = 8MHz
63 CSCTL2 = FLLD_0 + 243; // DCODIV = 8MHz ←
64 __delay_cycles(3);
65 __bic_SR_register(SCG0); // enable FLL
66 while(CSCTL7 & (FLLUNLOCK0 | FLLUNLOCK1)); // Poll until FLL is locked
67 CSCTL4 = SELMS__DCOCLKDIV | SELA__REFOCLK; // set default REF0(~32768Hz) as ACLK source, ACLK = 32768Hz
68 // default DCODIV as MCLK and SMCLK source
69
```

## Clock System Setup

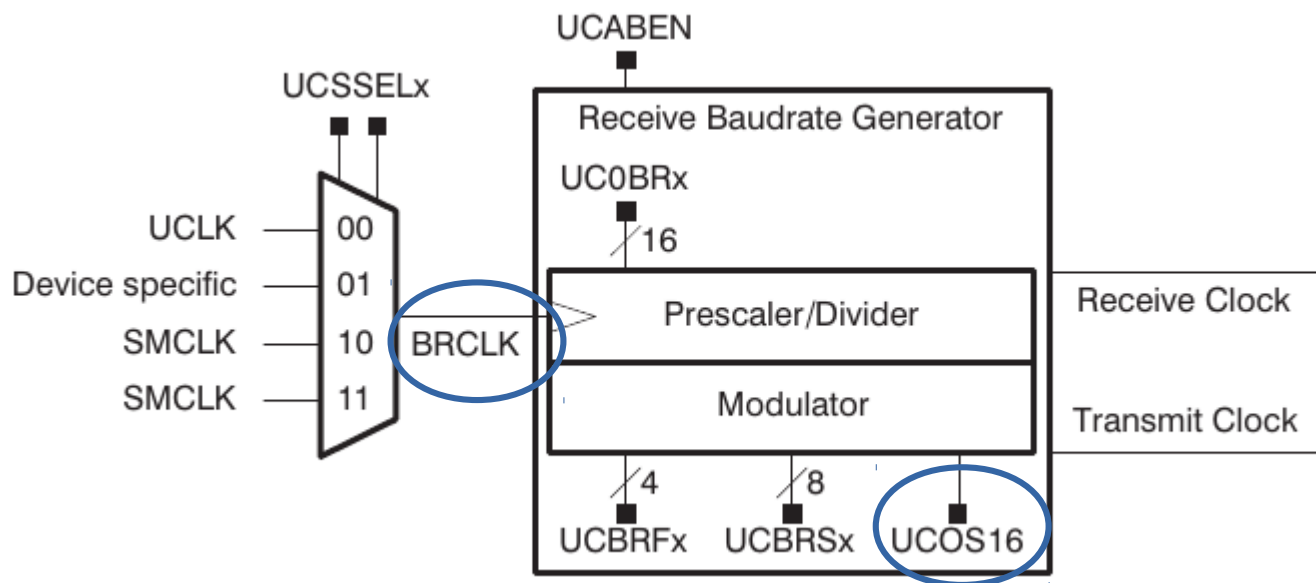
DCOCLK = 8MHz

Therefore

MCLK and SMCLK = 8MHz

ACLK = 32768 Hz





## SET BAUD RATE

NOTE:

Baud Rate settings quick set up

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

1. Calculate  $N = f_{BRCLK} / \text{Baud Rate}$   
[if  $N > 16$  continue with step 3, otherwise with step 2]
2. **OS16 = 0, UCBR0 = INT(N)**
3. **OS16 = 1, UCBR0 = INT(N/16)**
4. UCBR1 second byte of N - leave 0

```

72
73 // Configure UART
74 UCA0CTLW0 |= UCSWRST;
75 UCA0CTLW0 |= UCSSEL_SMCLK;
76 // Baud Rate calculation
77 // 8000000/(16*9600) = 52.083
78 // Fractional portion = 0.083
79 UCA0BR0 = 52; // 8000000/16/9600
80 UCA0MCTLW = 0x4900 | UCOS16 | UCBRF_1; // Remainder - 0.083
81 // see Table 21.4 & 21.5
82 UCA0CTLW0 &= ~UCSWRST; // Initialize eUSCI
83 UCA0IE |= UCRXIE; // Enable USCI_A0 RX interrupt
84

```

## Interrupt Vectors – All combinations for eUSCI

```
#pragma vector = USCI_A0_VECTOR __interrupt void USCI_A0_ISR(void) {  
switch(UCA0IV) {  
case 0x00:  
    // Vector 0: No interrupts  
    break;  
case 0x02: ... // Vector 2: UCRXIFG - Received Char Interrupt  
    break;  
case 0x04: ... // Vector 4: UCTXIFG - Transmit BufferEmpty Interrupt  
    break;  
case 0x06: ... // Vector 6: UCSTTIFG - START byte received  
    break;  
case 0x08: ... // Vector 8: UCTXIPTIFG - All transmit complete  
    break;  
default: break;  
}  
}
```

Interrupt Flag	Interrupt Condition
UCSTTIFG	START byte received interrupt. This flag is set when the UART module receives a START byte.
UCTXCPTIFG	Transmit complete interrupt. This flag is set, after the complete UART byte in the internal shift register including STOP bit got shifted out and UCAXTXBUF is empty.


The UCTXIFG interrupt flag is set by the transmitter to indicate that UCAXTXBUF is ready to accept another character. An interrupt request is generated if UCTXIE and GIE are also set. UCTXIFG is automatically reset if a character is written to UCAXTXBUF.

The UCRXIFG interrupt flag is set each time a character is received and loaded into UCAXRXBUF. An interrupt request is generated if UCRXIE and GIE are also set. UCRXIFG and UCRXIE are reset by a system reset PUC signal or when UCSWRST = 1. UCRXIFG is automatically reset when UCAXRXBUF is read.

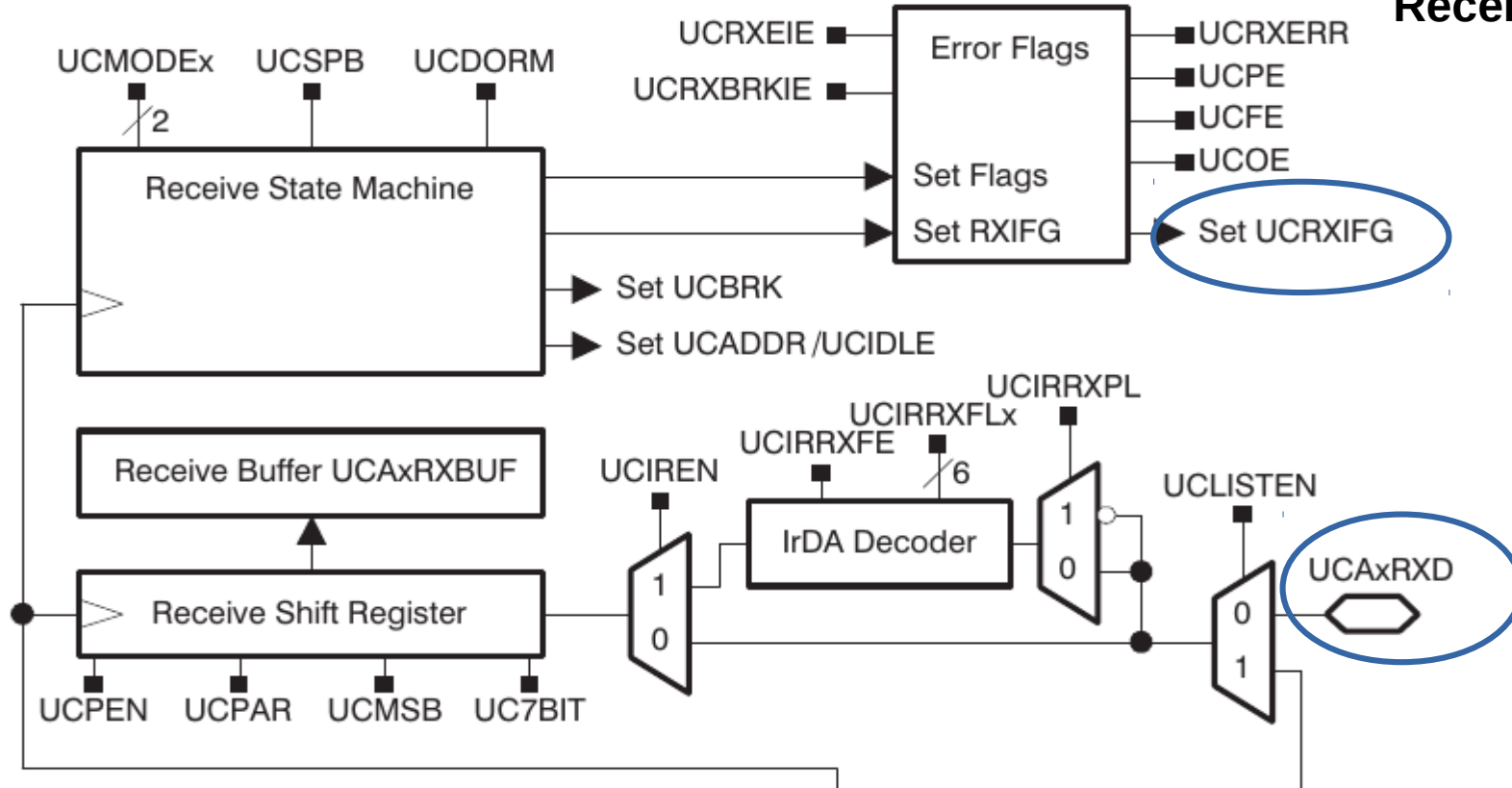
## Actual code from example 1

```
90 |
91 | #pragma vector=USCI_A0_VECTOR
92 | __interrupt void USCI_A0_ISR(void)
93 | {
94 |     switch(UCA0IV)
95 |     {
96 |     case USCI_NONE: break;
97 |     case USCI_UART_UCRXIFG:
98 |         while(!(UCA0IFG&UCTXIFG));
99 |         UCA0TXBUF = UCA0RXBUF;
100 |         __no_operation();
101 |         break;
102 |     case USCI_UART_UCTXIFG: break;
103 |     case USCI_UART_UCSTTIFG: break;
104 |     case USCI_UART_UCTXCPTIFG: break;
105 |     default: break;
106 |     }
107 | }
108 |
```

Echo received character  
(same as putchar(TxByte))  
Polled output

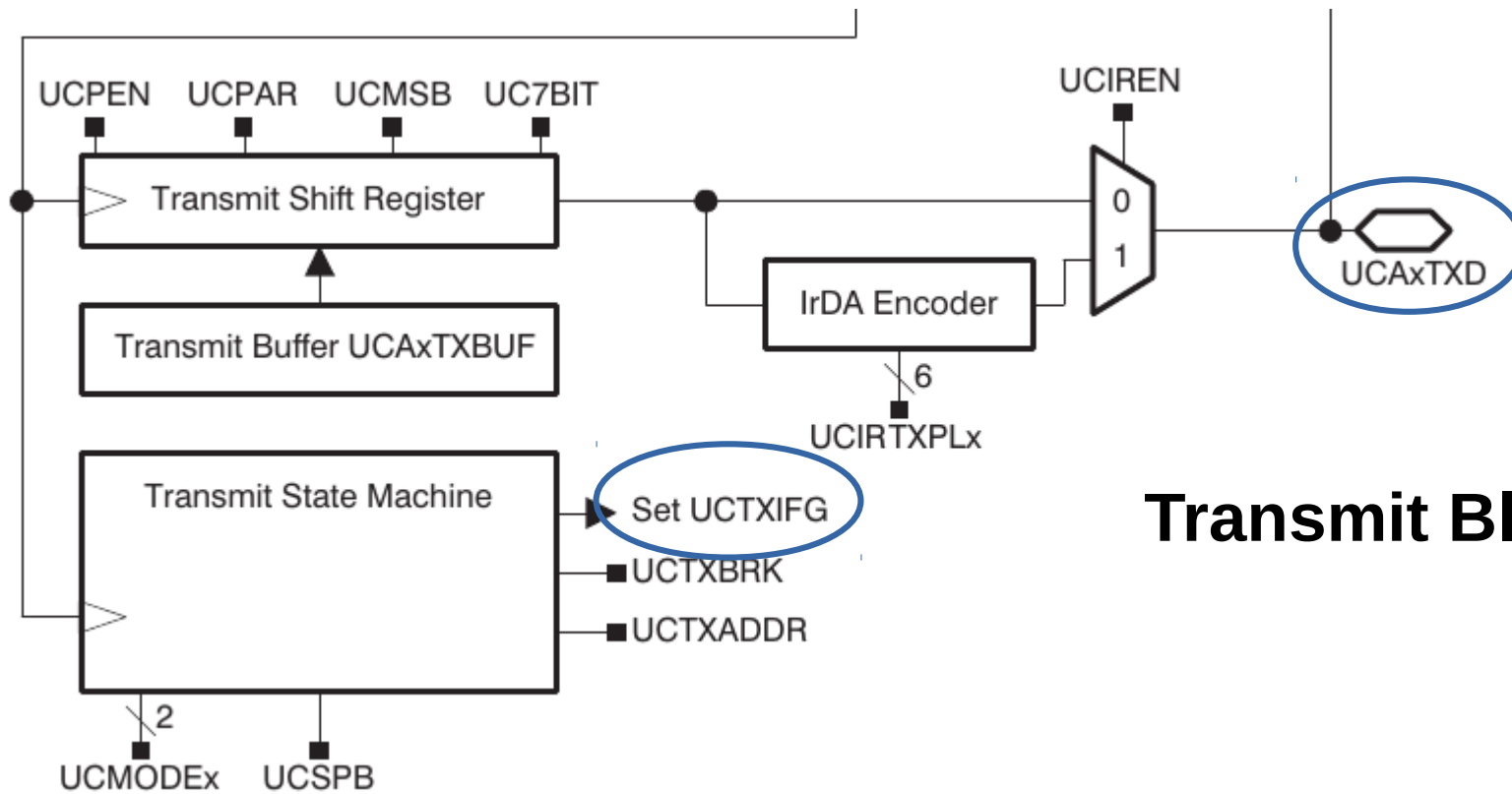


## Receive Block Diagram



The **UCRXIFG** interrupt flag is set each time a character is received and loaded into **UCAxRXBUF**. An interrupt request is generated if  $UCRXIE$  and  $GIE$  are also set.  $UCRXIFG$  and  $UCRXIE$  are reset by a system reset PUC signal or when  $UCSWRST = 1$ .

**UCRXIFG** is automatically reset when **UCAxRXBUF** is read.



## Transmit Block Diagram

### 21.3.15.1 eUSCI\_A Transmit Interrupt Operation

The **UCTXIFG** interrupt flag is set by the transmitter to indicate that UCAxTXBUF is ready to accept another character. An interrupt request is generated if UCTXIE and GIE are also set.

UCTXIFG is automatically reset if a character is written to UCAxTXBUF.

Show Example 1 Running

Example 2 – R/G LED 115200 BAUD

Example 3 – Transmit BUFFER with ISR  
9600 BAUD, ACLK LPM3