MSP430: Serial Communication: UART

MSP430 serial communication is handled by an on chip peripheral called **USCI** (**U**niversal **S**erial **C**ommunications**I**nterface).The peripheral is designed in such a way that it can handle multiple serial communication formats ,synchronous as well as asynchronous like **SPI**, **I2C**, IrDA,**UART** etc.**MSP430** has two USCI modules names as **USCI_A0** and **USCI_B0** for handling multiple communication formats. USCI_A0 can be configured to handle LIN, IrDA, SPI and asynchronous serial communication (UART) while USCI_B0 can handle SPI and I2C.

UART mode

The UART mode uses two pins to transmit (**UCA0TXD**) and receive data(**UCA0RXD**) . Asynchronous serial communication is widely used for communicating with external devices like GSM modems, GPS modules etc. Simplified block diagram of USCI_A0 in UART mode is given below.

Registers Required for UART

Configuration of MSP430 for UART operation means to write 0 or 1 bits to configuration registers of USCI module. USCI module has various 8-bit configuration registers to accomplish initialization and configuring USCI in UART mode. The steps for configuration are:

1. Set UCSWRST; disable UART module by setting the BIT
2. Initialize all USCI registers with UCSWRST = 1 (including UCAxCTL1).
3. Configure ports.
4. Clear UCSWRST; enable UART module for operation by clearing the BIT Enable interrupts (optional) via UCRXIE and/or UCTXIE.

The MSP430x5xx series device USCI_Ax module consists of two control registers for UART configuration as shown below:

UCAxCTL0 Register abbreviated as USCI_Ax ControlRegister0 UCAxCTL1 Register abbreviated as USCI_AxControlRegister1

The UART mode of the USCI_Ax module can be enabled by setting the UCSYNC bit of the UCAxCTL0 Register. Prior to the setting, we need to enable the UCSWRST bit of UCAxCTL1Register.

```
UCA0CTL1 |= UCSWRST;
```

1) UCAxCTL0 (USCI_Ax Control register0) here we are dealing with **USC1_A0** so **UCA0CTL0** -This register controls the settings for Parity selection, direction of data transmission(LSB or MSB first),character length, no of stop bits, modes of serial transmission.

UCA0CTL1

UCAxTXBUF (UCA0TXBUF) 8 bit data register for holding the byte to be transmitted by the MSP430 UART.

UCAxRXBUF (UCA0RXBUF) 8 bit data register that stores the received byte.

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 **remember to clear the flag** inside the interrupt service routine manually.

Baud rate configuration

BRCLK is the main clock source for generating baud rate for UART module. The external clock UCAxCLK, or the internal clocks ACLK or SMCLK depending on the UCSSELx settings can source the clock frequency for BRCLK (Refer UCAxCTL1Register). For a given BRCLK, the baud rate can be calculated if division factor N is known.

N= fBRCLK/Baudrate

The UART module supports two baud-rate generation mechanisms:
- Low frequency baud-rate generation
- Oversampling baud-rate generation

For UCAx, module values of UCBRx can be represented by two 8 bit registers UCA0BR0 which contains LSBs and UCA0BR1 which contains MSBs. In case of 1 MHZ BRCLK frequency, to set 9600, value of N is 104. This value is less than decimal 255. The code to set baud rate can be written as:

The USCI module registers need to be configured as:

UCA0CTL0: USCI_A0 control register 0, configure the register in this manner: No parity, LSB first, 8-bit data, 1 stop bit, UART, Asynchronous.

UCA0CTL1: USCI_A0 control register 1, use SMCLK as USCI clock source and enable software interrupts.

UCA0CTL1 |= UCSSEL_2

UCA0BR0 & UCA0BR1: USCI_A0 Baud rate control register 0 and 1, configure these 16-bit register to have 1MHz frequency and desired baud-rate

UCA0MCTL: USCI_A0 modulation control register, configure this register : second Stage modulation = 1, Oversampling off.

IE2: Interrupt Enable Register 2, Enable interrupt.

IFG2: Interrupt Flag Register 2, clear the flags.

Algorithm to transfer data serially using UART module

The algorithm to transfer character ‘A’ serially and toggle LED connected at P1.0 when data is transmitted.

Main program :

- • Configure watchdog timer
- • Configure P1.0 as output pin
- • Configure P3.3 as transmitter and P3.4 as receiver.
- • Set clock source for BRCLK , Set UCSWRST to enable USCI_A0
- • Configure baud rate of USCI module for UART
- • Set modulation by configuring modulation control register UCAxMCTL depending upon UCSO16 bit
- • Clear UCSWRST to enable USCI_A0
- • Enable Tx interrupt of USCI_Ax module
- • Store ASCII value of character A in transmit buffer register.
- • Enable Low power mode LPM1 with interrupt enable.

ISR_routine:

- • Initialize the interrupt address
- • Togel LED connected at pin P1.0.

Program:

```
#include <msp430.h>
#define TXnotRX  0  //transmitter = 1, receiver = 0
#define loopBack

char myString[] =  {"hello I m Redouane"};
char *txData = myString;
char rxData[64];
char offsetTab = 0;

int main(void)
{
  WDTCTL = WDTPW | WDTHOLD;                  // Stop WDT

  P3SEL |= BIT3 | BIT4;                     // Assign P3.3 to UCA0TXD and...
  P3DIR |= BIT3;                            // P3.4 to UCA0RXD

  UCA0CTL1 |= UCSWRST;                       // **Put state machine in reset**

  UCA0CTL1 |= UCSSEL_1;                      // CLK = ACLK
  UCA0BR0 |= 0x03;                            // 0x0332kHz/9600=3.41 (see User's
Guide)
  UCA0BR1 = 0x00;                            //
  UCA0MCTL = UCBRS_3|UCBRF_0;                // Modulation UCBRSx=3, UCBRFx=0

  UCA0CTL1 &= ~UCSWRST;                      // **Initialize USCI state machine**
  UCA0IE |= UCRXIE;                          // Enable USCI_A0 RX interrupt

  _enable_interrupt();

  while (1){


 UCA0TXBUF = 0xCA;


  }

}

// UART ISR
#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
```

```
{
  switch(__even_in_range(UCA0IV, 4))
  {
  case 0:break;                               // Vector 0 - no interrupt
  case 2:                                     // Vector 2 - RXIFG
    while (!(UCA0IFG & UCTXIFG));               // USCI A0 TX buffer ready?
    rxData[offsetTab++] = UCA0RXBUF;
#ifdef loopBack
    // loop back
    UCA0TXBUF = UCA0RXBUF;                    // TX -> RXed character
#endif

    break;
  case 4:break;                               // Vector 4 - TXIFG
  default: break;
  }
}
```