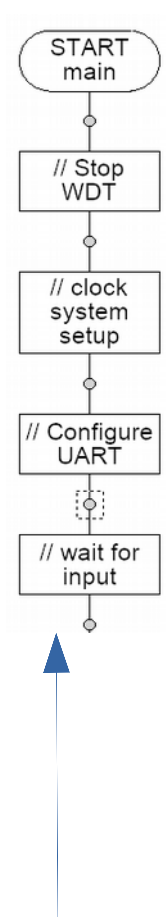
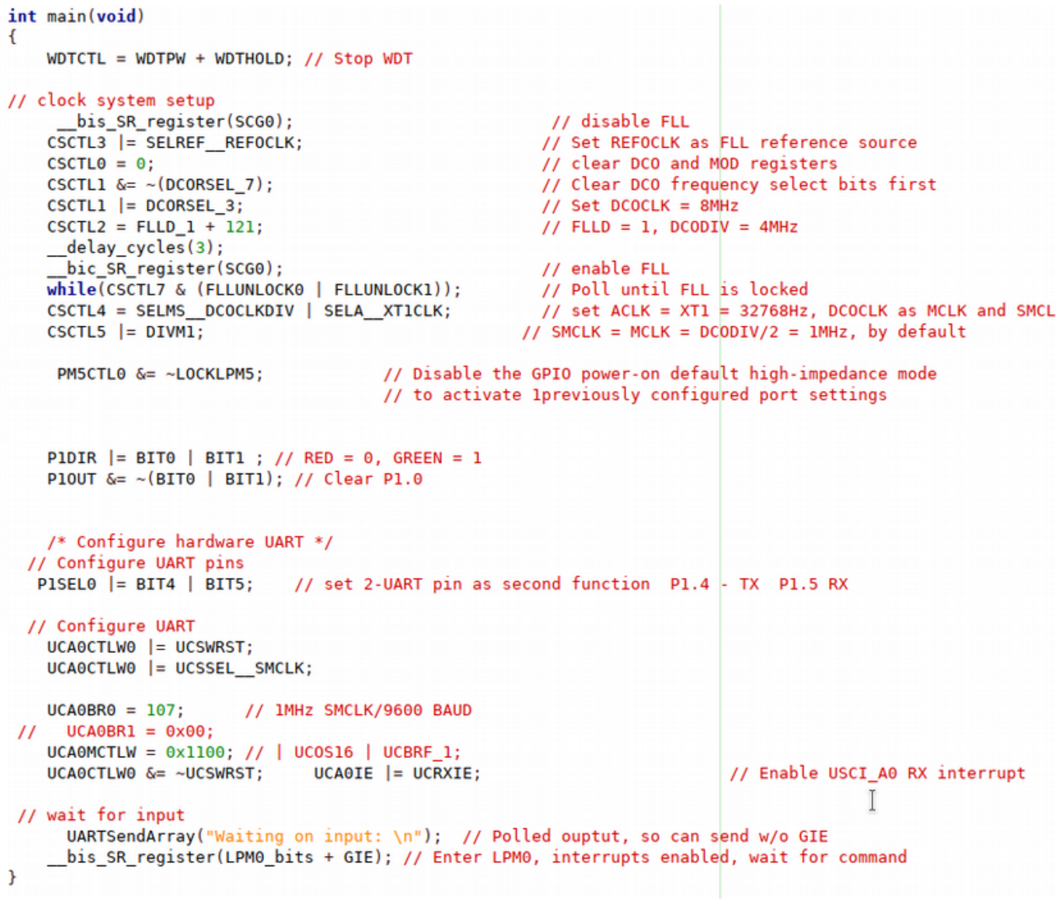
Program Flow Chart

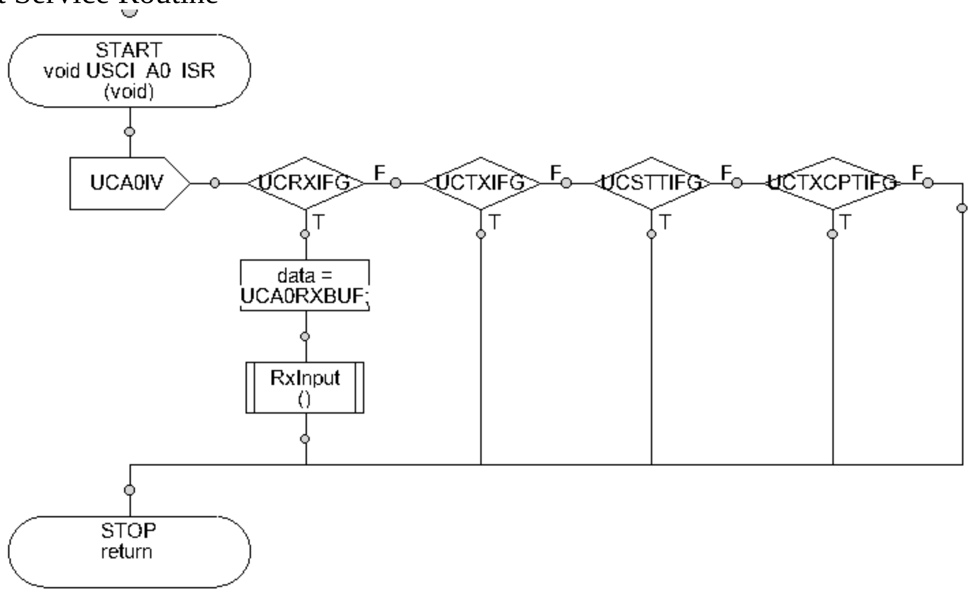


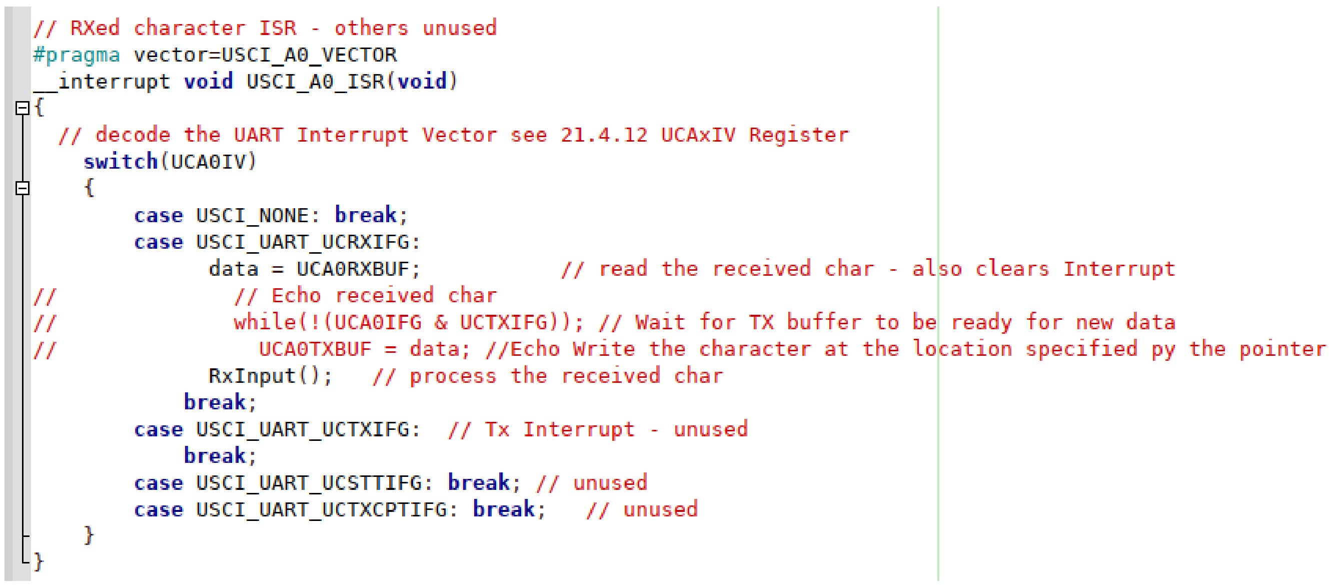


```
int main(void)
{
    WDTCTL = WDTPW + WDTHOLD; // Stop WDT

// clock system setup
     __bis_SR_register(SCG0);                           // disable FLL
    CSCTL3 |= SELREF__REFOCLK;                         // Set REFOCLK as FLL reference source
    CSCTL0 = 0;                                        // clear DCO and MOD registers
    CSCTL1 &= ~(DCORSEL_7);                            // Clear DCO frequency select bits first
    CSCTL1 |= DCORSEL_3;                               // Set DCOCLK = 8MHz
    CSCTL2 = FLLD_1 + 121;                             // FLLD = 1, DCODIV = 4MHz
    __delay_cycles(3);
    __bic_SR_register(SCG0);                           // enable FLL
    while(CSCTL7 & (FLLUNLOCK0 | FLLUNLOCK1));         // Poll until FLL is locked
    CSCTL4 = SELMS__DCOCLKDIV | SELA__XT1CLK;          // set ACLK = XT1 = 32768Hz, DCOCLK as MCLK and SMCL
    CSCTL5 |= DIVM1;                                 // SMCLK = MCLK = DCODIV/2 = 1MHz, by default

     PM5CTL0 &= ~LOCKLPM5;              // Disable the GPIO power-on default high-impedance mode
                                        // to activate 1previously configured port settings


    P1DIR |= BIT0 | BIT1 ; // RED = 0, GREEN = 1
    P1OUT &= ~(BIT0 | BIT1); // Clear P1.0


    /* Configure hardware UART */
  // Configure UART pins
   P1SEL0 |= BIT4 | BIT5;    // set 2-UART pin as second function  P1.4 - TX  P1.5 RX

  // Configure UART
    UCA0CTLW0 |= UCSWRST;
    UCA0CTLW0 |= UCSSEL__SMCLK;

    UCA0BR0 = 107;       // 1MHz SMCLK/9600 BAUD
 //   UCA0BR1 = 0x00;
    UCA0MCTLW = 0x1100; // | UCOS16 | UCBRF_1;
    UCA0CTLW0 &= ~UCSWRST;     UCA0IE |= UCRXIE;                       // Enable USCI_A0 RX interrupt

 // wait for input
      UARTSendArray("Waiting on input: \n");  // Polled ouptut, so can send w/o GIE
    __bis_SR_register(LPM0_bits + GIE); // Enter LPM0, interrupts enabled, wait for command
}
```

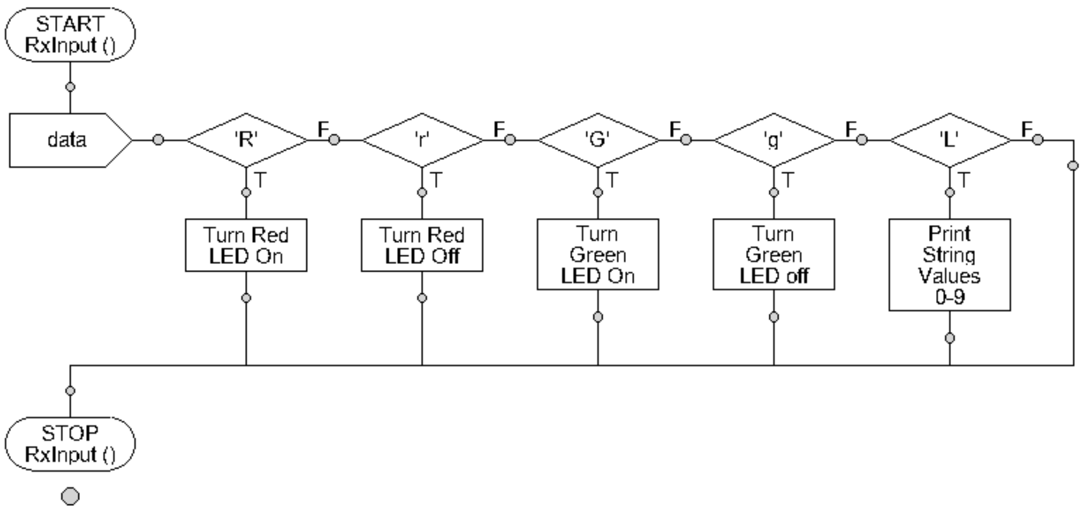
Flow chart for Program Set Up and Low Power Stand-by

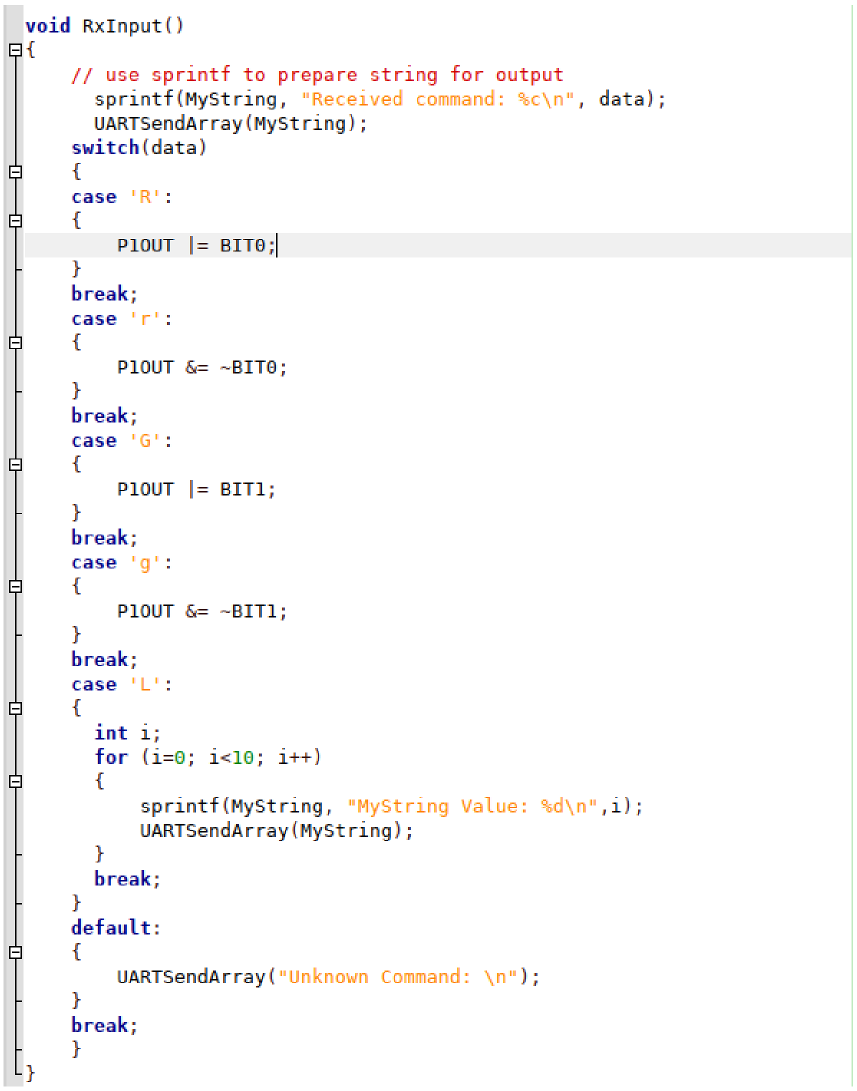
Interrupt Service Routine




```
// RXed character ISR - others unused
#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
{
  // decode the UART Interrupt Vector see 21.4.12 UCAxIV Register
    switch(UCA0IV)
    {
        case USCI_NONE: break;
        case USCI_UART_UCRXIFG:
              data = UCA0RXBUF;           // read the received char - also clears Interrupt
//              // Echo received char
//              while(!(UCA0IFG & UCTXIFG)); // Wait for TX buffer to be ready for new data
//                UCA0TXBUF = data; //Echo Write the character at the location specified py the pointer
              RxInput();   // process the received char
            break;
        case USCI_UART_UCTXIFG:  // Tx Interrupt - unused
            break;
        case USCI_UART_UCSTTIFG: break; // unused
        case USCI_UART_UCTXCPTIFG: break;   // unused
    }
}
```

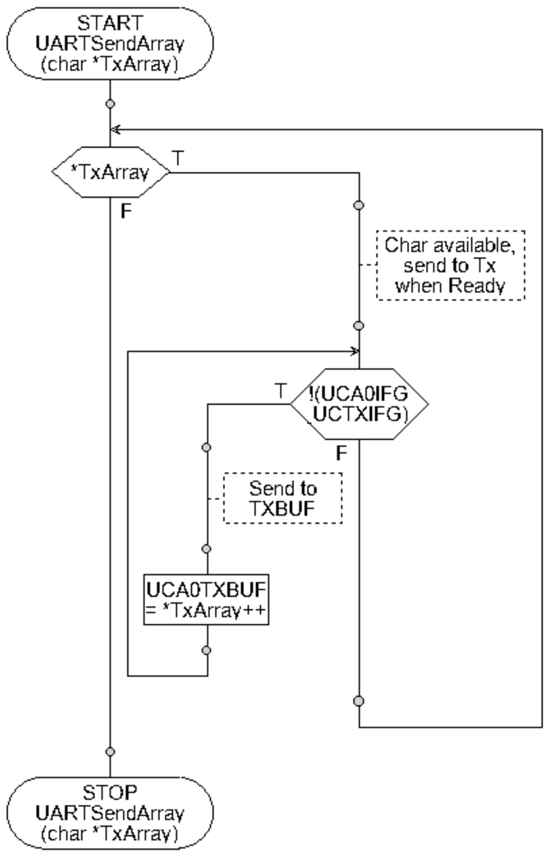
RxInput decode received commands

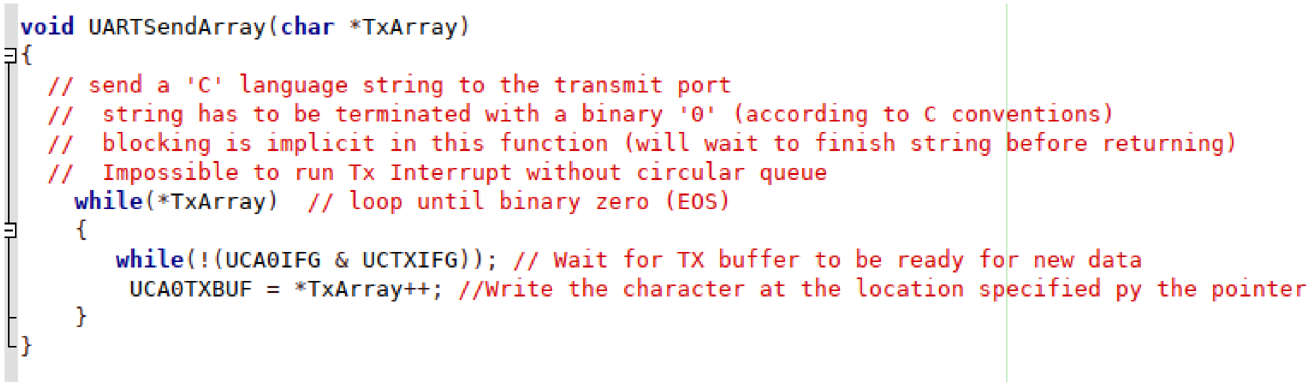




```
void RxInput()
{
    // use sprintf to prepare string for output
      sprintf(MyString, "Received command: %c\n", data);
      UARTSendArray(MyString);
    switch(data)
    {
    case 'R':
    {
        P1OUT |= BIT0;
    }
    break;
    case 'r':
    {
        P1OUT &= ~BIT0;
    }
    break;
    case 'G':
    {
        P1OUT |= BIT1;
    }
    break;
    case 'g':
    {
        P1OUT &= ~BIT1;
    }
    break;
    case 'L':
    {
      int i;
      for (i=0; i<10; i++)
      {
          sprintf(MyString, "MyString Value: %d\n",i);
          UARTSendArray(MyString);
      }
      break;
    }
    default:
    {
        UARTSendArray("Unknown Command: \n");
    }
    break;
    }
}
```

Send Array function – Transmit characters until Binary 0 (End of String) reached





```
void UARTSendArray(char *TxArray)
{
  // send a 'C' language string to the transmit port
  //  string has to be terminated with a binary '0' (according to C conventions)
  //  blocking is implicit in this function (will wait to finish string before returning)
  //  Impossible to run Tx Interrupt without circular queue
    while(*TxArray)  // loop until binary zero (EOS)
    {
       while(!(UCA0IFG & UCTXIFG)); // Wait for TX buffer to be ready for new data
        UCA0TXBUF = *TxArray++; //Write the character at the location specified py the pointer
    }
}
```

Document:

Download and run the code

Include listing of code in your document

Show the Red and Green LEDs illuminated with photo

Show Serial Monitor with listing of commands and the ‘L’ array printed

1. Question: Explain how dereferencing the TxArray pointer allows testing the value of each character in the array?
2. Question: Where does the TxArray pointer get incremented to the next characters?
3. Question: Why is TxArray a pointer instead of a character?

Video:

Turn on the Green and Red LED’s in video

Show Serial Monitor listing the ‘L’ command Array values

Be sure to say your time and date and name