CPE 323
Intro to Embedded Computer Systems
Serial Communication (UART)

Aleksandar Milenkovic

milenka@uah.edu

Admin

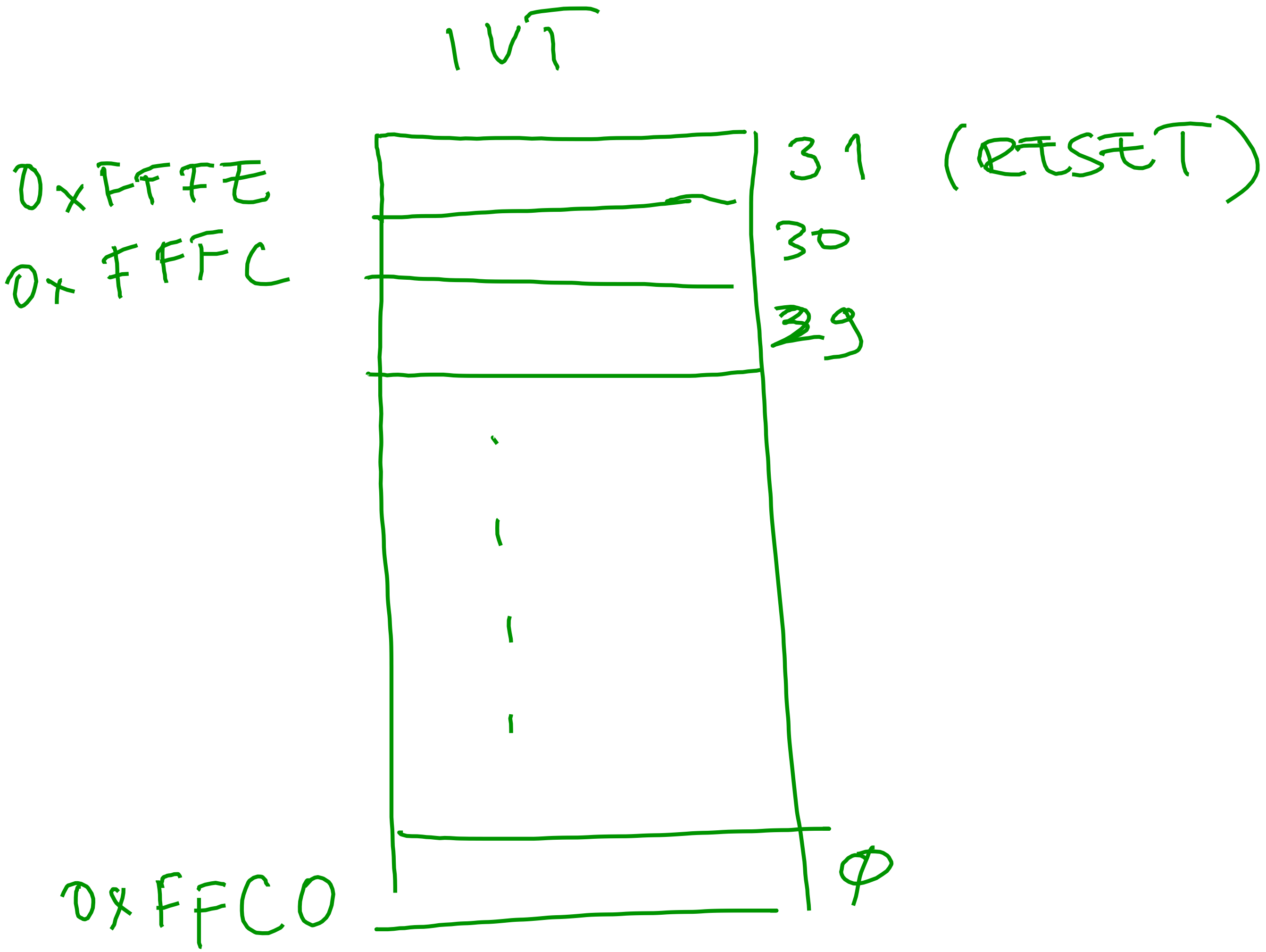
- → Quiz 05 MSP430 Interrupt quiz is tomorrow
- → Timers (WDT, Timer_A)
- → serial communication
  - UART – Universal Asynchronous Receiver/Transmitter
  - SPI – Serial Peripheral Interface
  - I^2C – Inter IC Comm.

Interrupts (Review)

1) Instruction Fetch
2) Ti – Decode
3) Operand Fetch
4) Instruction Execution
5) Store Results
6) Exception processing

1. Finish current instr.
2. Push PC
3. Push SR
4. Clear SR (GIE)
5. Select highest priority interrupt
6. Clear corresponding IFG bit (for single-sourced interrupts)
7. Read the corresponding entry from the IVT

Interrupts (Review)



1entry = 1word
= 2 bytes

Total size = 32 × 2 =
64 bytes

Question 8 — 20 pts

Stack allocation, pointers, pointer arithmetic.

Consider the following C code snippet. Assume that the register SP at the beginning points to 0x1100. Answer the following questions. Assume all variables are allocated on the stack and in the order as they appear in the program (li, ia, mc, pli, pc). ASCII code for character '0' is 48 (0x30), for 'A' is 0x41, and for 'a' is 0x61.

```
1 int main( void ) {
2     volatile long int li[ 2 ] = {3, -4};
3     volatile int ia[ 2 ] = {-4, 5};
4     volatile char mc[ 4 ] = {'2', 'A', 'a', '3'};
5     volatile char *pc = mc;
6     volatile long int *pli = li+1;
7     pc = pc + 3;              //
8     *pc += *(pc-1) - 0x30;    //
9     *pli = *pli + 4;          //
10 }
```

2x4 = 8
2x2 = 4

Answer the following questions.

A. How many bytes is allocated on the stack for the variable declared in line 2? 8 bytes (enter decimal number)

B. How many bytes is allocated on the stack for the character array in line 4? 4 bytes (enter decimal number)

C. How many bytes is allocated on the stack for all variables in lines 2-6? 20 bytes (enter decimal number)

D. What is the value of mc[1] after initialization in line 4? (enter hex value starting with 0x)

E. What is the value of li? (enter hex value starting with 0x)

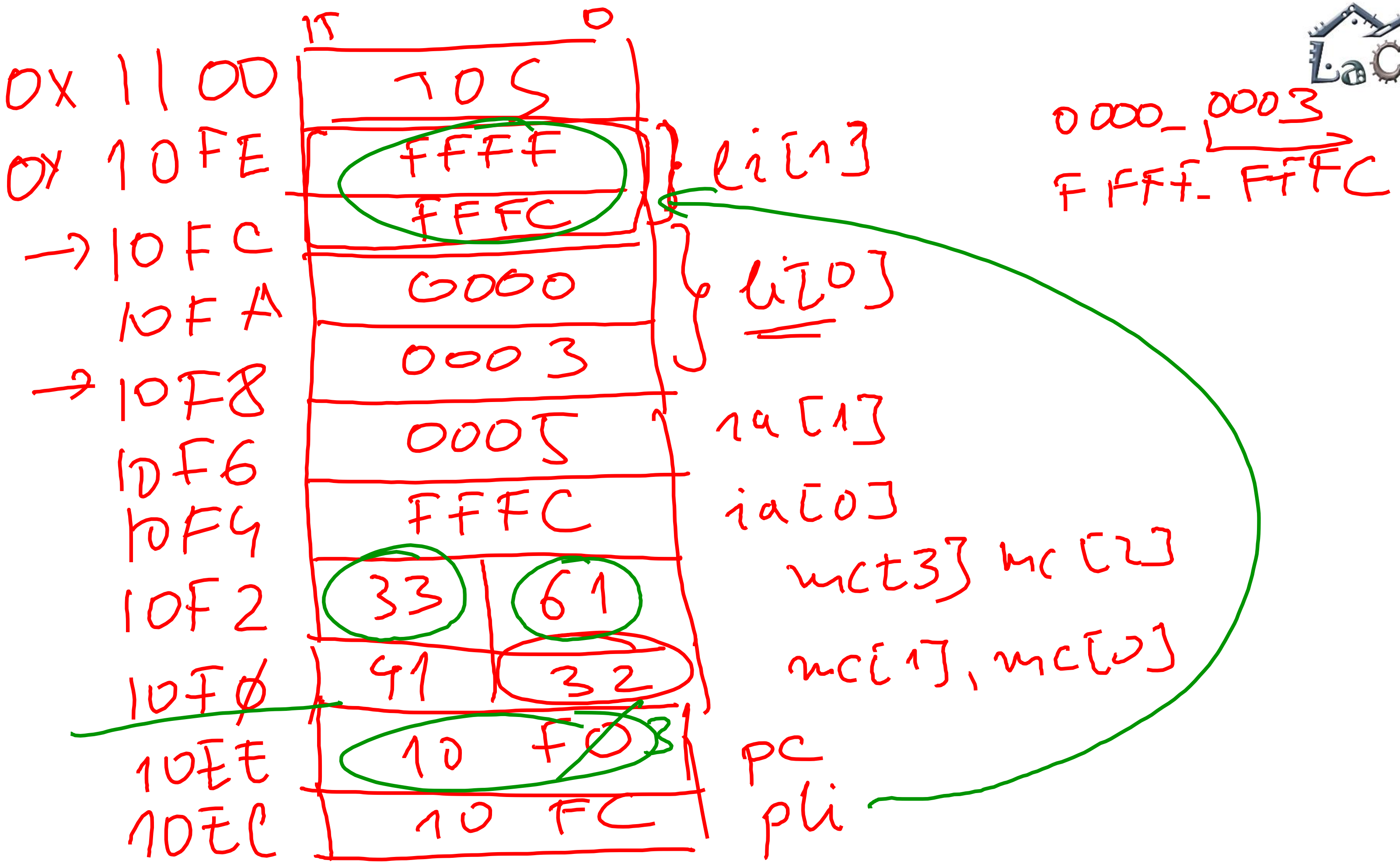
F. What is value of pc after the statement in line 5 is executed? 0x41 (enter hex value starting with 0x)

G. What is the value of pli after the statement in line 6 is executed? 10FC (enter hex value starting with 0x)

H. What is the value of pc after the statement in line 7 is executed? 10F3 (enter hex value starting with 0x)

I. What is the value of mc[3] after the statement in line 8 is executed? 0x64 (enter hex value starting with 0x)

J. What is the value of li[1] after the statement in line 9 is executed? 0x0000_0000 (enter hex value starting with 0x)



$$
\begin{array}{r}
33 \\
+\ 61 \\
-\ 30
\end{array}
\qquad
\begin{array}{r}
0x33 \\
+\ 0x31 \\
\hline
0x64
\end{array}
$$

pc = pc + 3

S: .long

Question 9

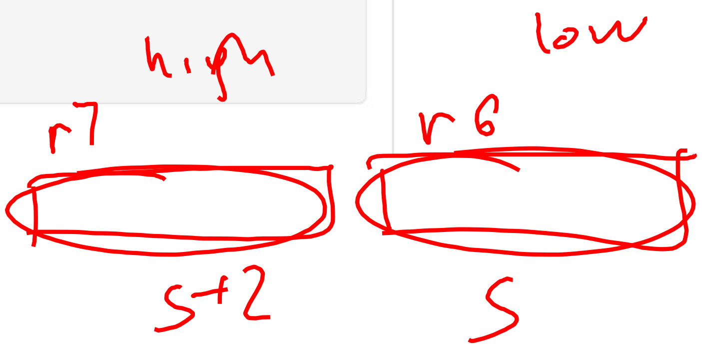
20 pts

Design and write an MSP430 assembly language subroutine AddStoLINTA with the following prototype:

void AddStoLINTA(long int myli, long int S, unsigned int size)*

The subroutine adds a scalar *S* to each element of an array *myli* with *size* elements. Both the scalar *S* and the array *myli* are long signed integers (32-bit long). Assume that a caller program pushes the input parameters onto the stack, before calling the subroutine. The inputs are pushed onto the stack in the order they are listed in the function descriptor, i.e., the main program before calling the subroutine *AddStoLINTA* will execute the following sequence of instructions:

```
PUSH.W #myli  ;
PUSH.W  S+2   ;
PUSH.W  S     ;
PUSH.W  size  ;
```




```
AddStoLINTA:
        push r5
        push r6
        push r7
        push r8
        mov.w 16(SP), r5
        mov.w 10(SP), r8
myl2:   mov.w @r5+, r6
        mov.w @r5+, r7
        add.w 12(SP), r6
        addc.w 14(SP), r7
        mov.w r6, -4(r5)
        mov.w r7, -2(r5)
        dec.w r8
        jnz myl2
        pop r8
        pop r7
        pop r6
        pop r5
        ret
```

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

LaCASA

MSP430F5529 Block Diagram

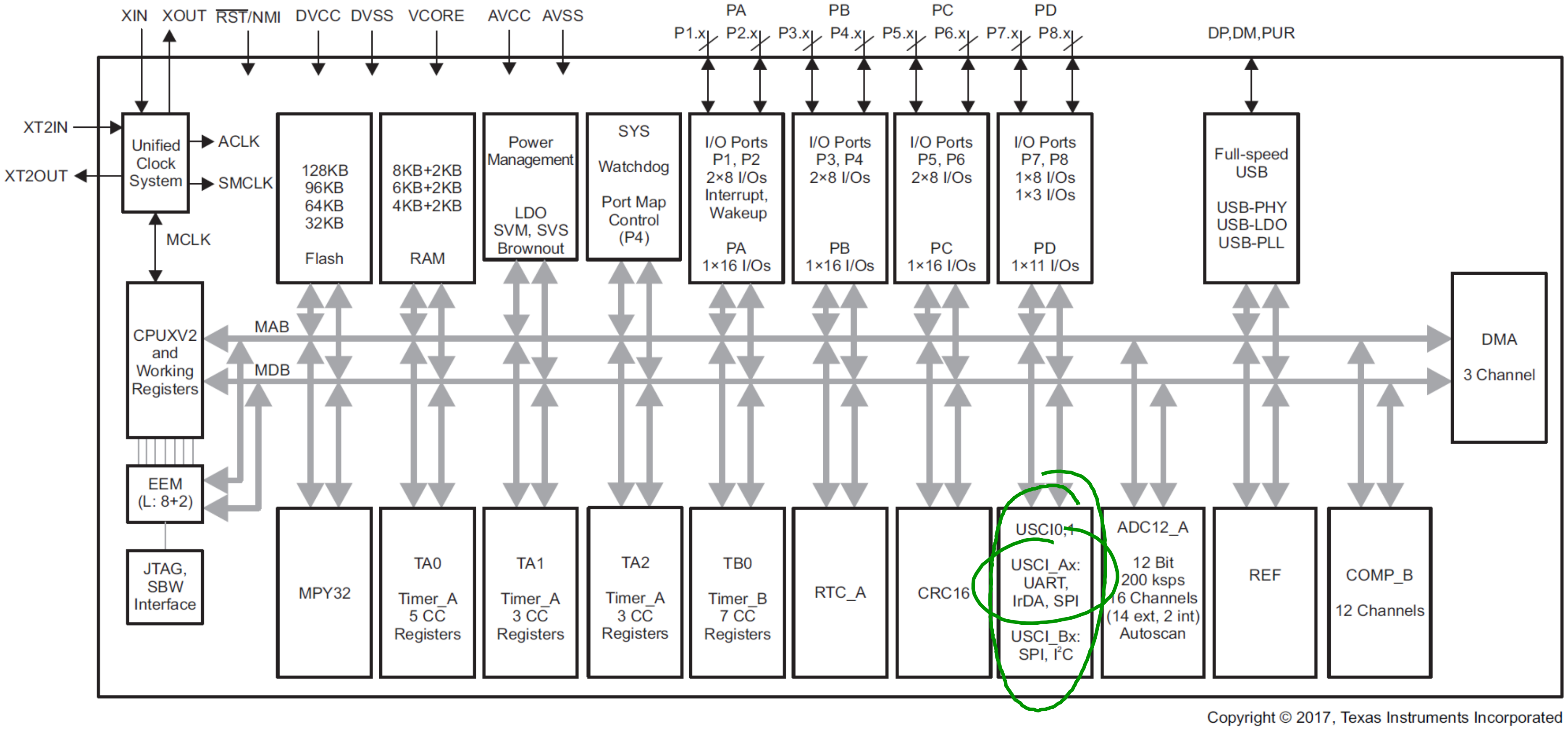


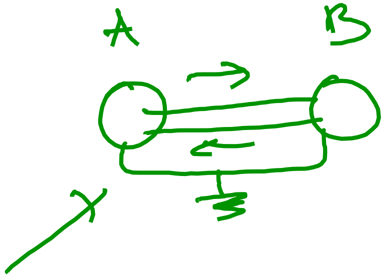
Figure 1-1. Functional Block Diagram – MSP430F5529IPN, MSP430F5527IPN, MSP430F5525IPN, MSP430F5521IPN

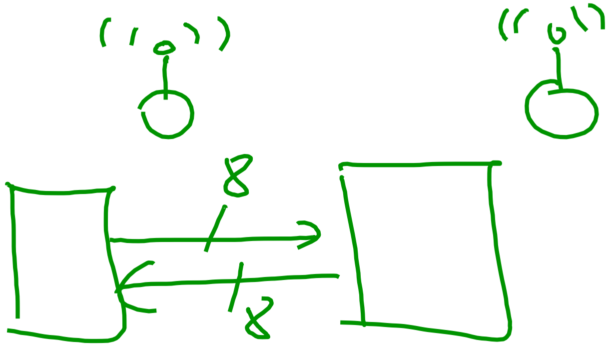
Communication

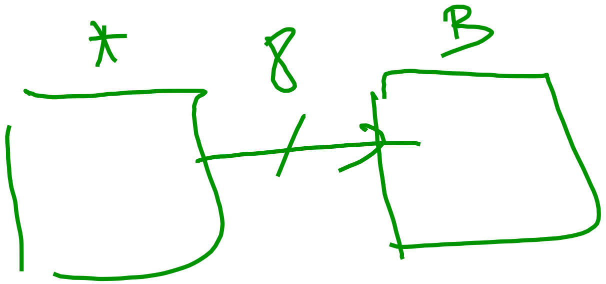
- Part of big 4
  - sense
  - process (compute)
  - store (memory)
  - communicate (UI, networks, ...)
- Communication in embedded systems
  - Between integrated circuits on PCB (e.g., μC $\leftrightarrow$ sensors)
  - Between development platform and a workstation
  - Between embedded systems

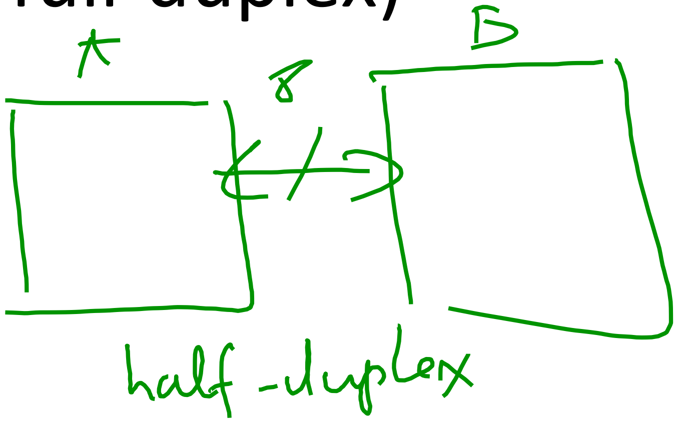
Types of Communication

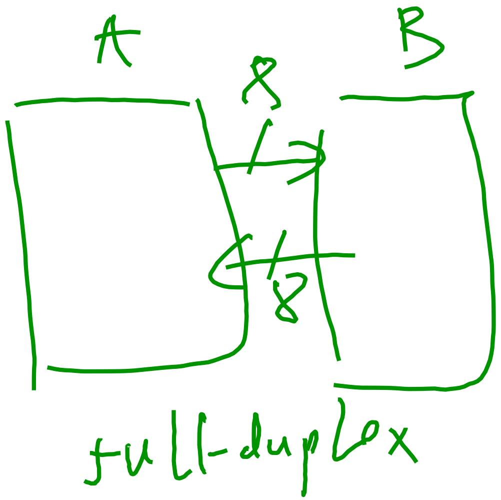
- Wired vs. wireless
- Serial vs. parallel
- Synchronous vs. asynchronous
- Unidirectional (simplex) vs. bidirectional (half-duplex and full-duplex)











Serial Communication in MSP430

- Communication protocols
  - UART (Universal Asynchronous Receiver/Transmitter)
  - SPI (Serial Parallel Interface) – Synchronous, bidirectional
  - I^2C (Inter Integrated Circuit) – Synchronous, bidirectional, bus [SDA SCL]
  - Infrared –
- Peripheral devices
  - $\boxed{\text{USCI}}$ – Universal Serial Communication Interface
  - USI – Universal Serial Interface
  - USART – Universal Synchronous/Asynchronous Receiver/Transmitter

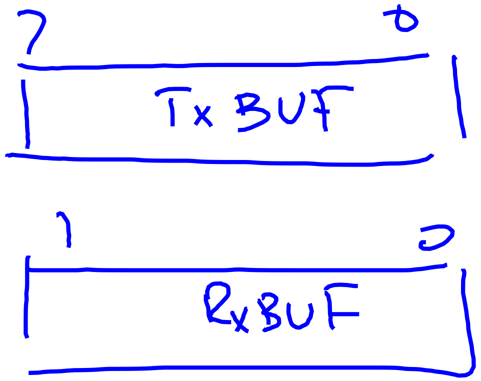
UART

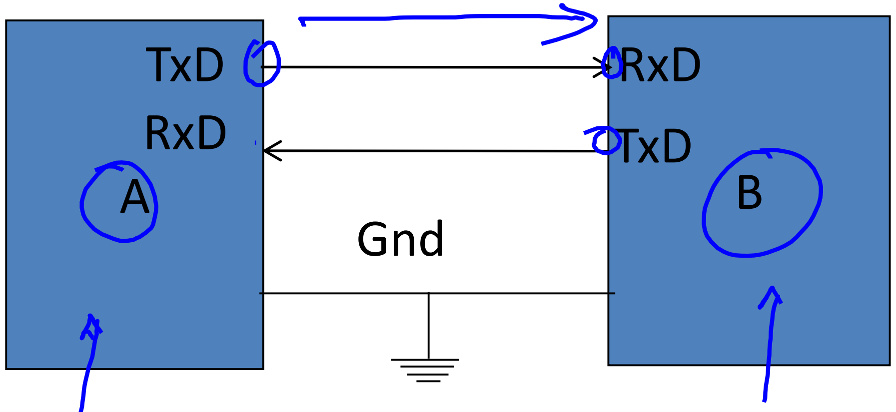
TxD - transmit data

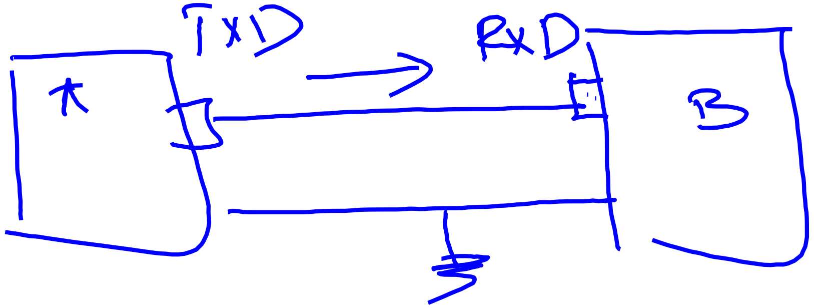
RxD - receive data

asynchronous communication

Character-oriented



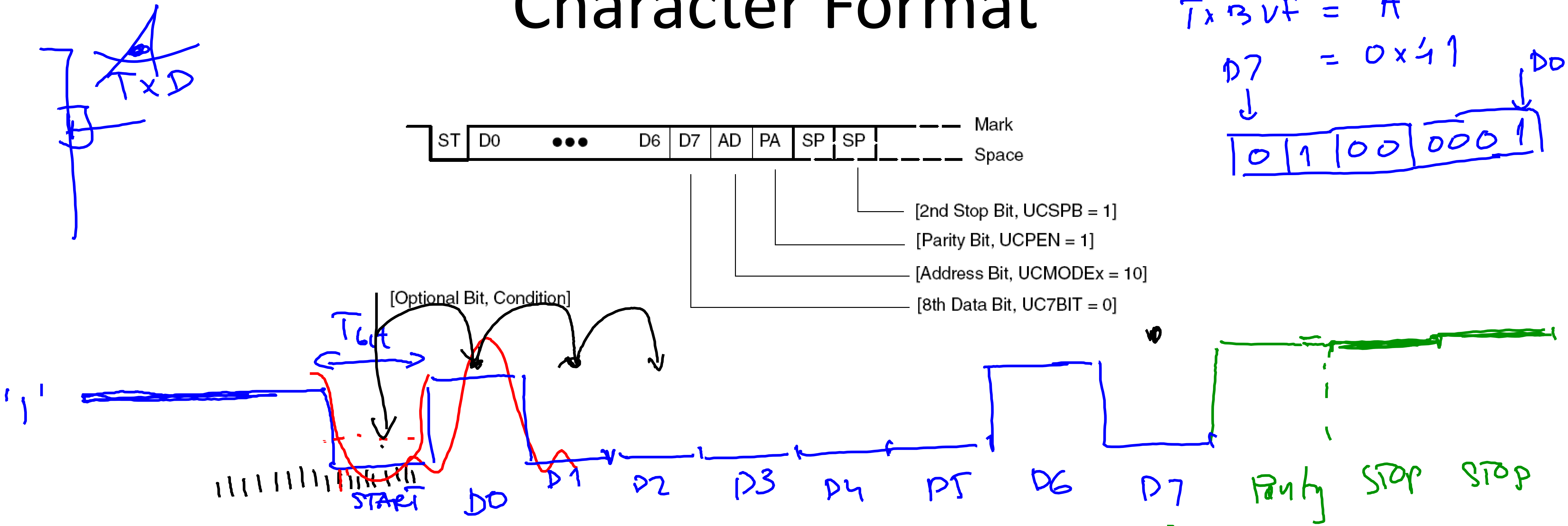




Character Format

TxBUF = 'A'
= 0x41

D7 D0
0 1 0 0 0 0 0 1



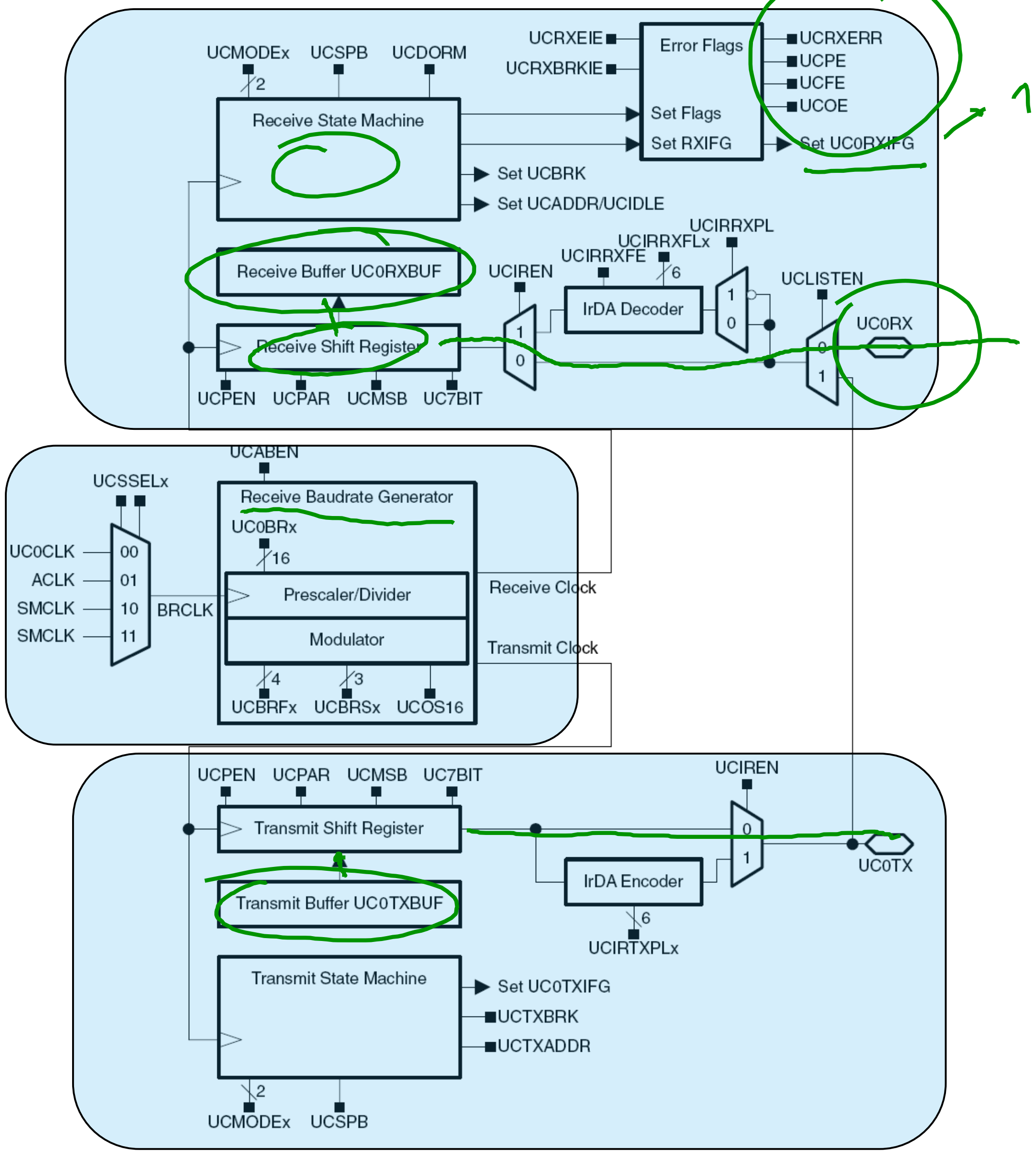
Parity - Even
- Odd

$1 + 8 + 1 + 2 = 12$ bits

start, data, parity, stop

$\boxed{12 \cdot T_{bit}}$

Figure 19–1. USCI_Ax Block Diagram: UART Mode (UCSYNC = 0)



1

Baud rate = 9,600 bps

= 14,400 bps

= 13,200

= 38,400

⋮

$T_{bit} = \frac{1}{9,600}$

USCI_A0 Registers

Table 19–6. USCI_A0 Control and Status Registers

Register	Short Form	Register Type	Address	Initial State
USCI_A0 control register 0	UCA0CTL0	Read/write	060h	Reset with PUC
USCI_A0 control register 1	UCA0CTL1	Read/write	061h	001h with PUC
USCI_A0 Baud rate control register 0	UCA0BR0	Read/write	062h	Reset with PUC
USCI_A0 Baud rate control register 1	UCA0BR1	Read/write	063h	Reset with PUC
USCI_A0 modulation control register	UCA0MCTL	Read/write	064h	Reset with PUC
USCI_A0 status register	UCA0STAT	Read/write	065h	Reset with PUC
USCI_A0 Receive buffer register	UCA0RXBUF	Read	066h	Reset with PUC
USCI_A0 Transmit buffer register	UCA0TXBUF	Read/write	067h	Reset with PUC
USCI_A0 Auto Baud control register	UCA0ABCTL	Read/write	05Dh	Reset with PUC
USCI_A0 IrDA Transmit control register	UCA0IRTCTL	Read/write	05Eh	Reset with PUC
USCI_A0 IrDA Receive control register	UCA0IRRCTL	Read/write	05Fh	Reset with PUC
SFR interrupt enable register 2	IE2	Read/write	001h	Reset with PUC
SFR interrupt flag register 2	IFG2	Read/write	003h	00Ah with PUC

Error Conditions

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 UCAxRXIFG if the break interrupt enable UCBRKIE bit is set.

Baud Rate Generation

- Definitions
  - BRCLK is input clock (ACLK, SMCLK, UCLK)
  - $F_{BITCLK}=F_{BAUD}$ is bit clocks (e.g., 38,400 bps) $T_{BITCLK} = 1/38,400$
  - $F_{BITCLK16}=16*F_{BAUD}$
- Oversampling mode (UCOS16=1)
  - BRCLK is divided to give BITCLK16, which is further divided by 16 to give BITCLK
- Low Frequency mode (UCOS16=0)
  - BRCLK is divided to give BITCLK

Baud Rate Generation

- Oversampling: f_{baud} = 9600 Hz, $f_{BRCLK} = 2^{20}$ Hz

Baud Rate Generation

- Low frequency is used when $f_{BRCLK} < 16*f_{baud}$
- f_{baud} = 9600 Hz, $f_{BRCLK} = 2^{15}$ Hz (ACLK)

Echo a character using Polling

```
/*------------------------------------------------------------------------------
 * File:         Lab8_D1.c
 *
 * Function:     Echo a received character, using polling.
 *
 * Description:  This program echos the character received from UART back to UART.
 *               Toggle LED1 with every received character.
 *               Baud rate: low-frequency (UCOS16=0);
 *               1048576/115200 = ~9.1 (0x0009|0x01)
 *
 * Clocks:       ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = default DCO
 *
 * Board:        MSP-EXP430F5529
 *
 * Instructions: Set the following parameters in putty
 * Port: COMx
 * Baud rate: 115200
 * Data bits: 8
 * Parity: None
 * Stop bits: 1
 * Flow Control: None
 *
 * Note:      If you are using Adafruit USBtoTTL cable, look for COM port
 *            in the Windows Device Manager with the following text:
 *            Silicon Labs CP210x USB to UART Bridge (COM<x>).
 *            Connecting Adafruit USB to TTL:
 *             GND - black wire - connect to the GND pin (on the board or
BoosterPack)
 *             Vcc - red wire - leave disconnected
 *             Rx    white wire (receive into USB, connect on TxD of the board P3.3)
 *             Tx -  green wire (transmit from USB, connect to RxD of the board
P3.4)
 *        MSP430F5529
 *     -----------------
 * /|\ |             XIN|-
 *  |  |                | 32kHz
 *  |--|RST         XOUT|-
 *     |                |
 *     |   P3.3/UCA0TXD|------------>
 *     |                | 115200 - 8N1
 *     |   P3.4/UCA0RXD|<------------
 *     |           P1.0|----> LED1
 *
 * Input:     None (Type characters in putty/MobaXterm/hyperterminal)
 * Output:    Character echoed at UART
 * Author:    A. Milenkovic, milenkovic@computer.org
 * Date:      October 2018, modified August 2020
 *------------------------------------------------------------------------------*/
```

```c
#include <msp430.h>

void UART_setup(void) {

    P3SEL |= BIT3 + BIT4;    // Set USCI_A0 RXD/TXD to receive/transmit data
    UCA0CTL1 |= UCSWRST;     // Set software reset during initialization
    UCA0CTL0 = 0;            // USCI_A0 control register
    UCA0CTL1 |= UCSSEL_2;    // Clock source SMCLK

    UCA0BR0 = 0x09;          // 1048576 Hz  / 115200 lower byte
    UCA0BR1 = 0x00;          // upper byte
    UCA0MCTL |= UCBRS0;      // Modulation (UCBRS0=0x01, UCOS16=0)

    UCA0CTL1 &= ~UCSWRST;    // Clear software reset to initialize USCI state machine
}

void main(void) {
    WDTCTL = WDTPW + WDTHOLD;        // Stop WDT
    P1DIR |= BIT0;                   // Set P1.0 to be output
    UART_setup();                    // Initialize UART

    while (1) {
       while(!(UCA0IFG&UCRXIFG));    // Wait for a new character
       // New character is here in UCA0RXBUF
       while(!(UCA0IFG&UCTXIFG));    // Wait until TXBUF is free
       UCA0TXBUF = UCA0RXBUF;        // TXBUF <= RXBUF (echo)
       P1OUT ^= BIT0;                // Toggle LED1
    }
}
```

Echo a character using ISR

```c
#include <msp430.h>

// Initialize USCI_A0 module to UART mode
void UART_setup(void) {

    P3SEL |= BIT3 + BIT4;    // Set USCI_A0 RXD/TXD to receive/transmit data
    UCA0CTL1 |= UCSWRST;     // Set software reset during initialization
    UCA0CTL0 = 0;            // USCI_A0 control register
    UCA0CTL1 |= UCSSEL_2;    // Clock source SMCLK

    UCA0BR0 = 0x09;          // 1048576 Hz  / 115200 lower byte
    UCA0BR1 = 0x00;          // upper byte
    UCA0MCTL |= UCBRS0;      // Modulation (UCBRS0=0x01, UCOS16=0)

    UCA0CTL1 &= ~UCSWRST;    // Clear software reset to initialize USCI state machine
    UCA0IE |= UCRXIE;        // Enable USCI_A0 RX interrupt
}

void main(void) {
    WDTCTL = WDTPW + WDTHOLD;// Stop WDT
    P1DIR |= BIT0;            // Set P1.0 to be output
    UART_setup();             // InitiAlize USCI_A0 in UART mode

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

// Echo back RXed character, confirm TX buffer is ready first
#pragma vector = USCI_A0_VECTOR
__interrupt void USCIA0RX_ISR (void) {
    while(!(UCA0IFG&UCTXIFG));  // Wait until can transmit
    UCA0TXBUF = UCA0RXBUF;      // TXBUF <-- RXBUF
    P1OUT ^= BIT0;              // Toggle LED1
}
```

Display Real-Time Clock

```
/*------------------------------------------------------------------------------
 * File:          Lab8_D3.c
 * Function:      Displays real-time clock in serial communication client.
 * Description:   This program maintains real-time clock and sends time
 *                (10 times a second) to the workstation through
 *                a serial asynchronous link (UART).
 *                The time is displayed as follows: "sssss:tsec".
 *
 *                Baud rate divider with 1048576hz = 1048576/(16*9600) = ~6.8 [16
from UCOS16]
 * Clocks:        ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = default DCO = 1048576Hz
 * Instructions:  Set the following parameters in putty/hyperterminal
 * Port: COMx
 * Baud rate: 19200
 * Data bits: 8
 * Parity: None
 * Stop bits: 1
 * Flow Control: None
 *
 *        MSP430F5529
 *     -----------------
 * /|\ |            XIN|-
 *  |  |               | 32kHz
 *  |--|RST        XOUT|-
 *     |               |
 *     |   P3.3/UCA0TXD|------------>
 *     |               | 9600 - 8N1
 *     |   P3.4/UCA0RXD|<------------
 *     |           P1.0|----> LED1
 *
 * Author:      A. Milenkovic, milenkovic@computer.org
 * Date:        October 2018
------------------------------------------------------------------------------*/
```

```
#include <msp430.h>
#include <stdio.h>

// Current time variables
unsigned int sec = 0;                // Seconds
unsigned int tsec = 0;               // 1/10 second
char Time[8];                        // String to keep current time

void UART_setup(void) {
    P3SEL = BIT3+BIT4;                          // P3.4,5 = USCI_A0 TXD/RXD
    UCA0CTL1 |= UCSWRST;                        // **Put state machine in reset**
    UCA0CTL1 |= UCSSEL_2;                       // SMCLK
    UCA0BR0 = 6;                                // 1MHz 9600 (see User's Guide)
    UCA0BR1 = 0;                                // 1MHz 9600
    UCA0MCTL = UCBRS_0 + UCBRF_13 + UCOS16;     // Mod. UCBRSx=0, UCBRFx=0,
                                                // over sampling
    UCA0CTL1 &= ~UCSWRST;                       // **Initialize USCI state machine**
}

void TimerA_setup(void) {
    TA0CTL = TASSEL_2 + MC_1 + ID_3; // Select SMCLK/8 and up mode
    TA0CCR0 = 13107;                 // 100ms interval
    TA0CCTL0 = CCIE;                 // Capture/compare interrupt enable
}
```

Display Real-Time Clock (cont'd)

```c
void UART_putCharacter(char c) {
    while (!(UCA0IFG&UCTXIFG));    // Wait for previous character to transmit
    UCA0TXBUF = c;                  // Put character into tx buffer
}

void SetTime(void) {
    tsec++;
    if (tsec == 10){
        tsec = 0;
        sec++;
        P1OUT ^= BIT0;              // Toggle LED1
    }
}

void SendTime(void) {
    int i;
    sprintf(Time, "%05d:%01d", sec, tsec);// Prints time to a string

    for (i = 0; i < sizeof(Time); i++) {  // Send character by character
        UART_putCharacter(Time[i]);
    }
    UART_putCharacter('\r');        // Carriage Return
}
```

```c
void main(void) {
    WDTCTL = WDTPW + WDTHOLD;       // Stop watchdog timer
    UART_setup();                   // Initialize UART
    TimerA_setup();                 // Initialize Timer_B
    P1DIR |= BIT0;                  // P1.0 is output;

    while (1) {
        _BIS_SR(LPM0_bits + GIE);   // Enter LPM0 w/ interrupts
        SendTime();                 // Send Time to HyperTerminal/putty
    }
}

#pragma vector = TIMER0_A0_VECTOR
__interrupt void TIMERA_ISA(void) {
    SetTime();                       // Update time
    _BIC_SR_IRQ(LPM0_bits);          // Clear LPM0 bits from 0(SR)
}
```