ECE2049: Embedded Computing in Engineering Design
C Term -- Spring 2013

Lecture #18: Introduction to Serial Interfaces

Reading for Today:	Articles, User's Manual Ch 15
Reading for Next Class:	User's Manual Ch 15, External Temp Sensor information
Lab #3 (on web):	Report due 2/19/2013
HW #5 (coming soon!):	Due Thursday 2/28/2013
Lab #4 (on web):	Due Friday 3/1/2013 (by 4 pm to box in ECE office)

Serial Interfaces – USART and SPI

Parallel Interfaces = Each bit has its own electrical connection (interconnect, trace, wire)

>> Advantages = Fast, easier to synchronize (1 clock edge transfers all bits together)

>> Used almost exclusively inside a CPU (or other) chip

>> Disadvantage = Each bit must have its own electrical connection

***Serial Interfaces* –** Bits sent one after another along a single connection

>> Used almost exclusively to make connections off-chip (and off-computer, through the Internet, out to the Mars Rover, etc)

>> Advantages = Simpler/fewer connections between CPU and peripheral (2-4 lines)

Device Select (CS)

Sychronizing CLK (SCLK)

Data Line(s) (SDI and SDO)

Common ground (GND)

>> “Connection” = PCB trace, wire, RF, acoustic, optical, etc.

>> Disadvantages = Slower, more complicated synchronization, potential timing issues

Universal Synchronous/Asynchronous Receive Transmitter (USART)

>> Basically acts as a parallel-to-serial and serial-to-parallel converter

>> Most modern microprocessors/microcontrollers will have built-in USART
  -- MSP430F449 has two!

>> Role of USART has grown with growing sophistication and speed of serial interfaces (SPI and I^{2}C to USB and others)

>> MSP430F449 USARTs have 2 modes: UART and SPI

UART Mode (User's Guide Ch 14)

>> UART mode configures basic 2-wire *asynchronous* serial comms

>> 2 external pins (URXD and UTXD)

>> Not synchronous (no shared clock) = Asynchronous

>> To use serial communications both devices must know data format and baud rate

  -- These are set using USARTs control registers

  -- Implies make data format & baud rate decisions at design time

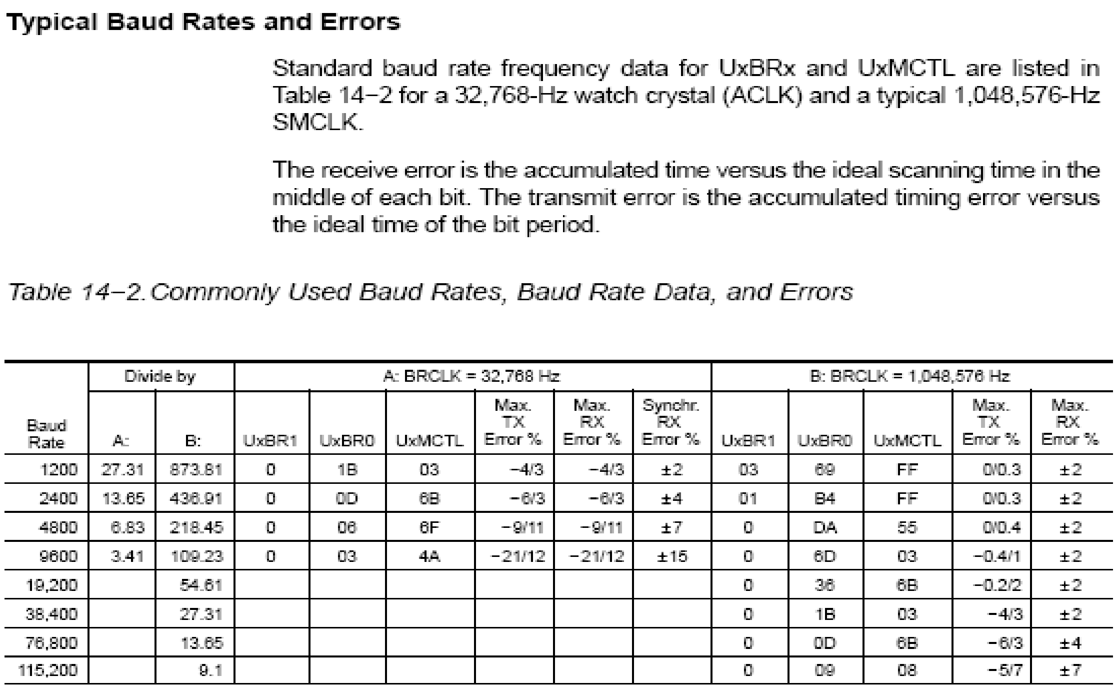
Data Formats -- Serial Communications

Start Bit = 1 bit (“low”)   Data Bits = 7 or 8 bits

Parity = Even, Odd, or None
- >> Even Parity = 1 when number of 1's including parity is even
- >> Odd Parity = 1 when number of 1's including parity is odd

Stop bit(s) = 1 or 2 bits (“high”)

RS-232 – “Old standby” for serial format → Data sent Least Significant Bit (LSB) first!



Typical Baud Rates and Errors

Standard baud rate frequency data for UxBRx and UxMCTL are listed in Table 14−2 for a 32,768-Hz watch crystal (ACLK) and a typical 1,048,576-Hz SMCLK.

The receive error is the accumulated time versus the ideal scanning time in the middle of each bit. The transmit error is the accumulated timing error versus the ideal time of the bit period.

Table 14−2. Commonly Used Baud Rates, Baud Rate Data, and Errors

Baud Rate	Divide by		A: BRCLK = 32,768 Hz						B: BRCLK = 1,048,576 Hz				
	A:	B:	UxBR1	UxBR0	UxMCTL	Max. TX Error %	Max. RX Error %	Synchr. RX Error %	UxBR1	UxBR0	UxMCTL	Max. TX Error %	Max. RX Error %
1200	27.31	873.81	0	1B	03	−4/3	−4/3	±2	03	69	FF	0/0.3	±2
2400	13.65	436.91	0	0D	6B	−6/3	−6/3	±4	01	B4	FF	0/0.3	±2
4800	6.83	218.45	0	06	6F	−9/11	−9/11	±7	0	DA	55	0/0.4	±2
9600	3.41	109.23	0	03	4A	−21/12	−21/12	±15	0	6D	03	−0.4/1	±2
19,200		54.61							0	36	6B	−0.2/2	±2
38,400		27.31							0	1B	03	−4/3	±2
76,800		13.65							0	0D	6B	−6/3	±4
115,200		9.1							0	09	08	−5/7	±7

Example: How long would it take to transmit “C TERM IS HALF OVER” at 9600 baud with 1 start bit 1 stop bit and even parity assume 8-bit ASCII encoding?

Serial Peripheral Interface Bus (SPI)

>> Used primarily for synchronous serial comms between a CPU and peripherals “within the box”

-- Synchronous = shared clock (supplied by Master)

>> Usually a 4 wire connection (some times 3 wire)

SIMO = Slave In/Master Out data line

SOMI = Slave Out/ Master In data

SCLK = Serial Clock (UCLK in MSP430 USART documentation)

CS = Chip Select

>> SPI Loose standard (close to Microwire but not quite) --> *Different* from I^2C

***How will you know what to use*?**

--> SPI, I^2C, asynchronous serial (RS-232), other?

Sensors or other peripheral devices will specify the interfaces with which they are compatible

→ MSP430F449 does not support I^2C

>> ***To use SPI the programmer must...***

1) Enable USART for SPI mode
   → Including SELecting data and clock pins for Function mode
2) Select data format
3) Setup synchronous clock

```
void setupSPI(void)
{
  ME1 |= USPIE0;       // Enable USART0 SPI mode

  U0CTL &= ~SWRST;     // Make sure SW RESET bit is off

// Bits 3, 2 & 1 are used for SPI UCLK, SOMI and SIMO
  P3SEL |= BIT3|BIT2|BIT1;   // set bits to Function mode

  U0CTL |= CHAR + SYNC + MM;   // USART0 module control
                               // CHAR = 1 => 8-bit data
                               // SYNC = 1 => SPI mode
                               // MM = 1 => master mode

  U0TCTL |= SSEL0+SSEL1+STC+CKPL;  // USART0 xmit control
                                   // SSEL0 = 1 & SSEL1 = 1
                                // (use SMCLK for baud-rate gen.)
                                // STC = 1 => 3-pin SPI mode
                                // CKPL =  1 = set Clock polarity
  U0BR0  = 0x02;    // Divide SMCLK by 4 => SCLK = 262KHz
  U0BR1  = 0x00;    // (a “reasonable” CLK value for temp sensor)

  U0MCTL = 0x00;    // Modulation control - not used.
                    // Ensure all these bits are reset to 0

  U0TXBUF = 0x00;   // Clear the transmit buffer
}
```

Then programmer must ...

4) Select peripheral using its chip select (CS)

First, what is the function of a CS?

>> Likely to be multiple peripherals using SPI bus

>> Only 1 Slave device and Master (MSP430) can use SPI bus at a time

>> CS is typically ACTIVE LOW digital signal

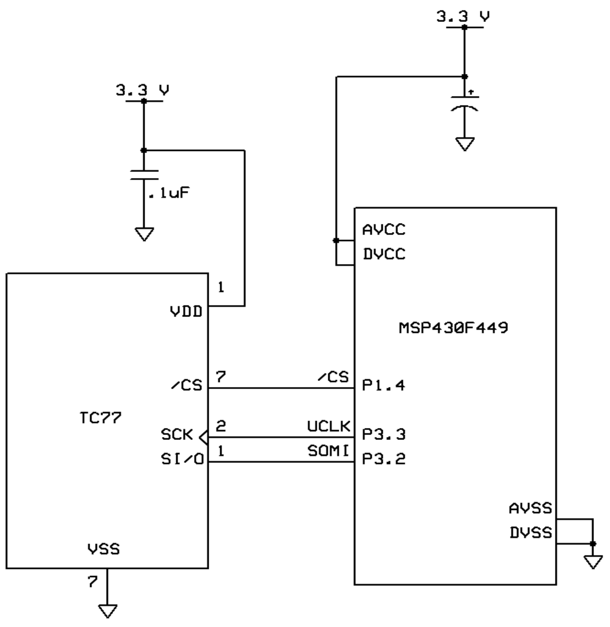
CS = 0 = Device is Enabled (will read and write to SPI data lines)

CS = 1= Device is Disabled (outputs are high impedence)

>> On our lab boards, the MSP430 Interface Board has 2 peripherals that are SPI devices – the external temperature sensor and the digital to analog converter.

Configuring the CS for the External Temperature Sensor

--> Check schematic to see how the temperature sensor's CS is connected



```
// Configure a Chip Select signal (active low) for temp sensor
#define     tempCS   0x10   // Temp sensor Chip Select on Pin 4


P1SEL &= ~(tempCS);     // Select P1.4 digital IO for TC77 CS but
                        // leave other bits alone

P1DIR |= tempCS;        // Set CS bit as an output

P1OUT |= tempCS;        // De-assert CS (set CS =1)
```