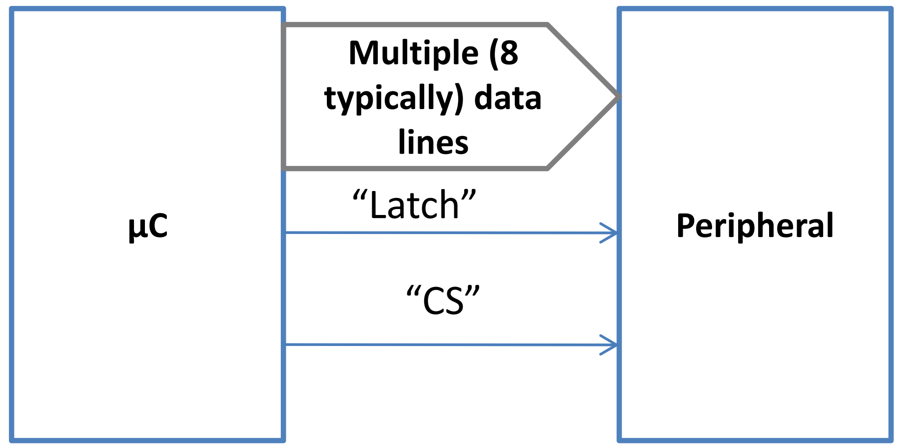
Serial Communications

(Chapter 10)

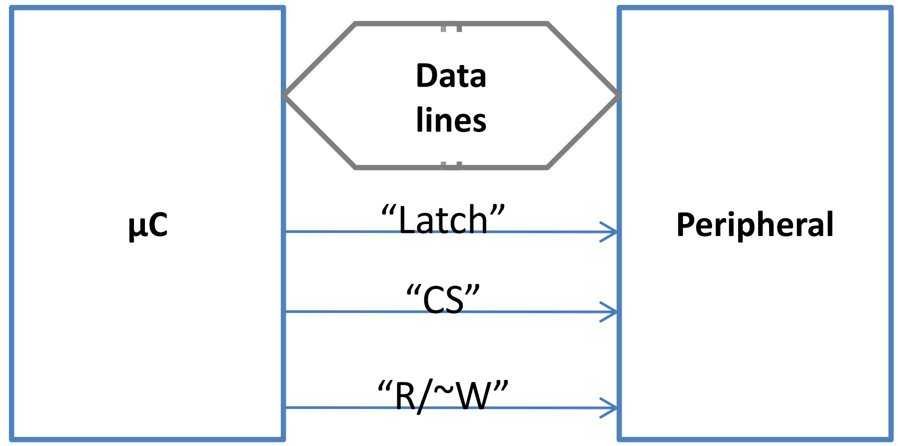
RS232, SPI, I2C

Communications

- The simplest is parallel
  - One way
    - There may be mechanism for peripheral to get attention of μC (i.e., interrupt, or poll)



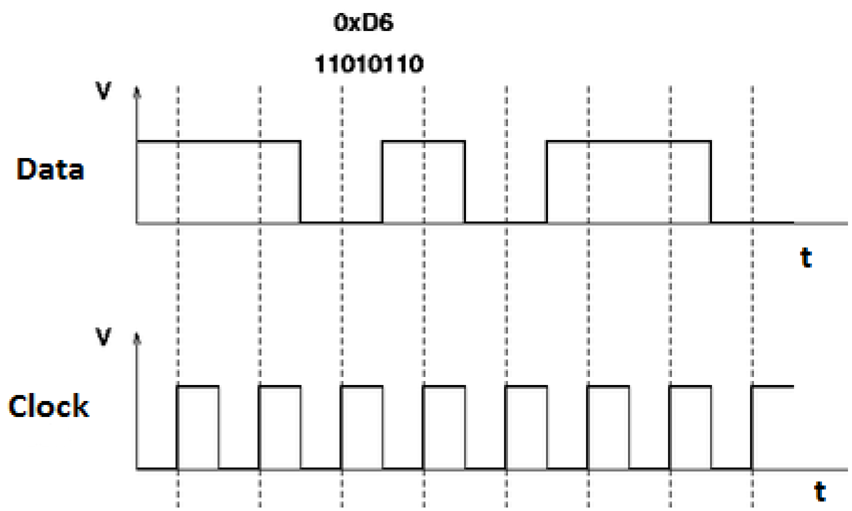
  - Two way

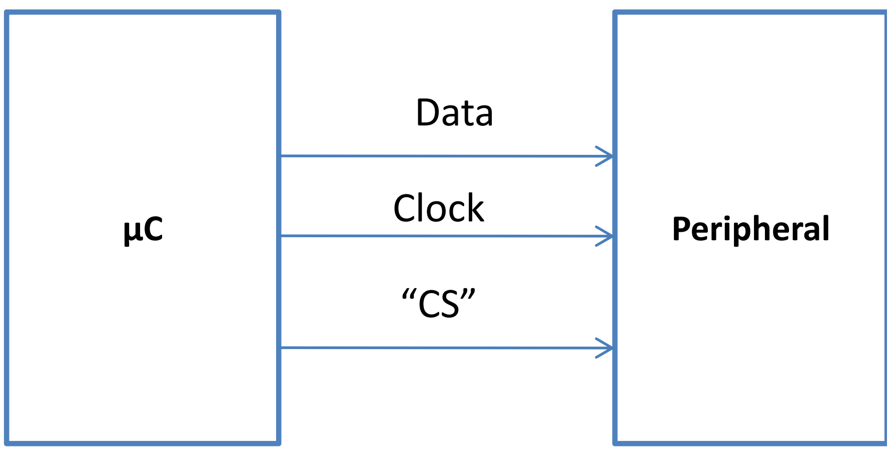


- This is resource expensive (pins, real-estate…) in terms of hardware, but easy to implement

Serial Communications

- Many fewer lines are required to transmit data. This is requires fewer pins, but adds complexity.

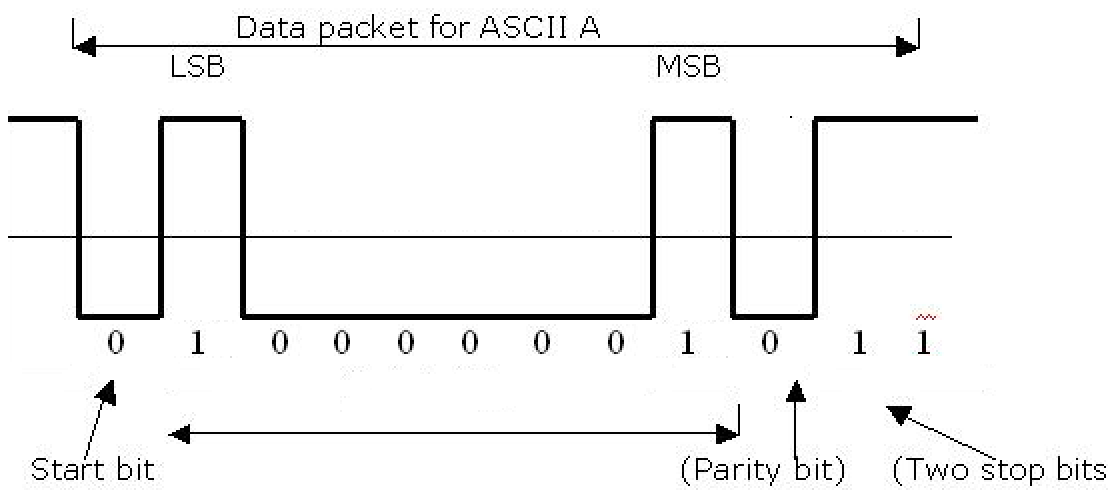




- Synchronous communications requires clock. Whoever controls the clock controls communication speed.
- Asynchronous has no clock, but speed must be agreed upon beforehand (baud rate).

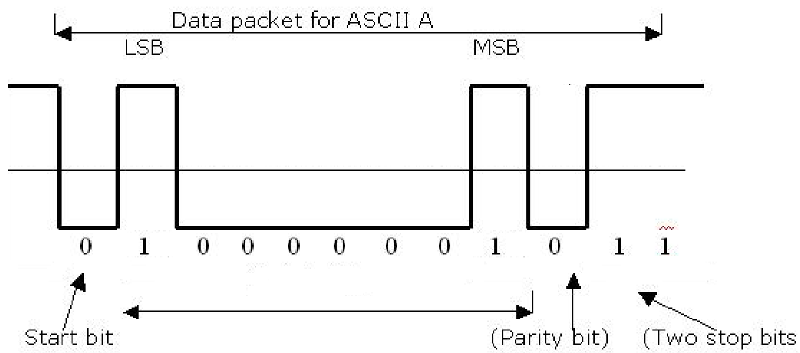
Asynchronous Serial (RS-232)

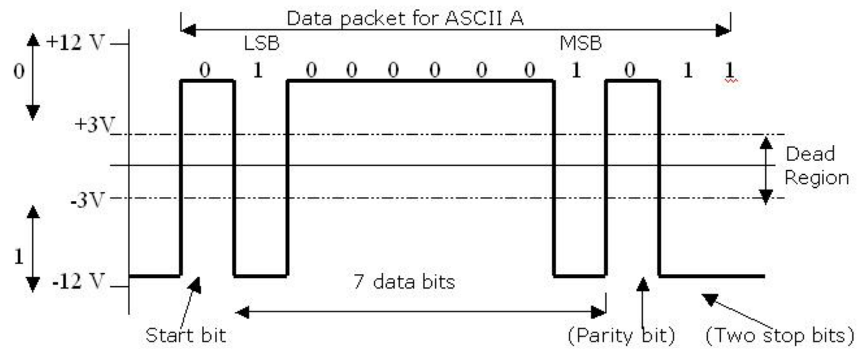
- Commonly used for one-to-one communication.
- There are many variants, the simplest uses just two lines, TX (transmit) and RX (receive).
- Transmission process (9600 baud, 1 bit=1/9600=0.104 mS)
  - Transmit idles high (when no communication).
  - It goes low for 1 bit (0.104 mS)
  - It sends out data, LSB first (7 or 8 bits)
  - There may be a parity bit (even or odd – error detection)
  - There may be a stop bit (or two)



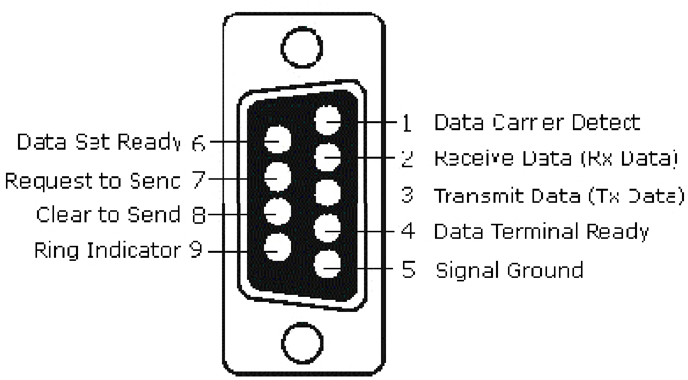
RS232 Voltage levels

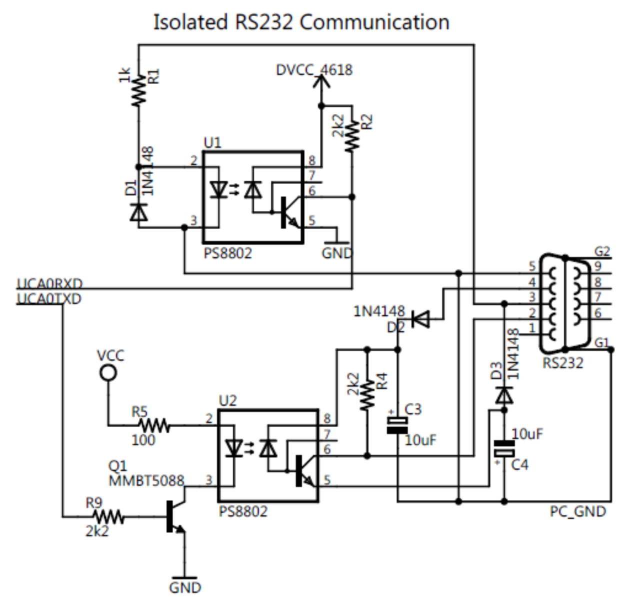
- From processor side, 0V=logic 0, 3.3V=logic 1
- In a “serial” cable +12→+3V=logic 0, -3→-12V=logic 1





- On “Experimenter’s board”
- Physical connector





RS232 – Handshaking

- Some RS232 connections using handshaking lines between DCE (Data Communications Equipment) and DTE (Data Terminal Equipment).
  - RTS (Ready To Send)
    - Sent by the DTE to signal the DCE it is Ready To Send.
  - CTS (Clear To Send)
    - Sent by the DCE to signal the DTE that it is Ready to Receive.
  - DTR (Data Terminal Ready)
    - Sent to DTE to signal the DCE that it is ready to connect
  - DSR (Data Set Read)
    - Sent to DC to signal the DTE that it is ready to connect
- In practice if these handshaking lines are used it can be difficult to set up the serial communications, but it is quite robust once working.
- There is also software handshaking (XON/XOFF)
- DTE and DCE have different connector pinouts.

MSP430 USCI in UART mode

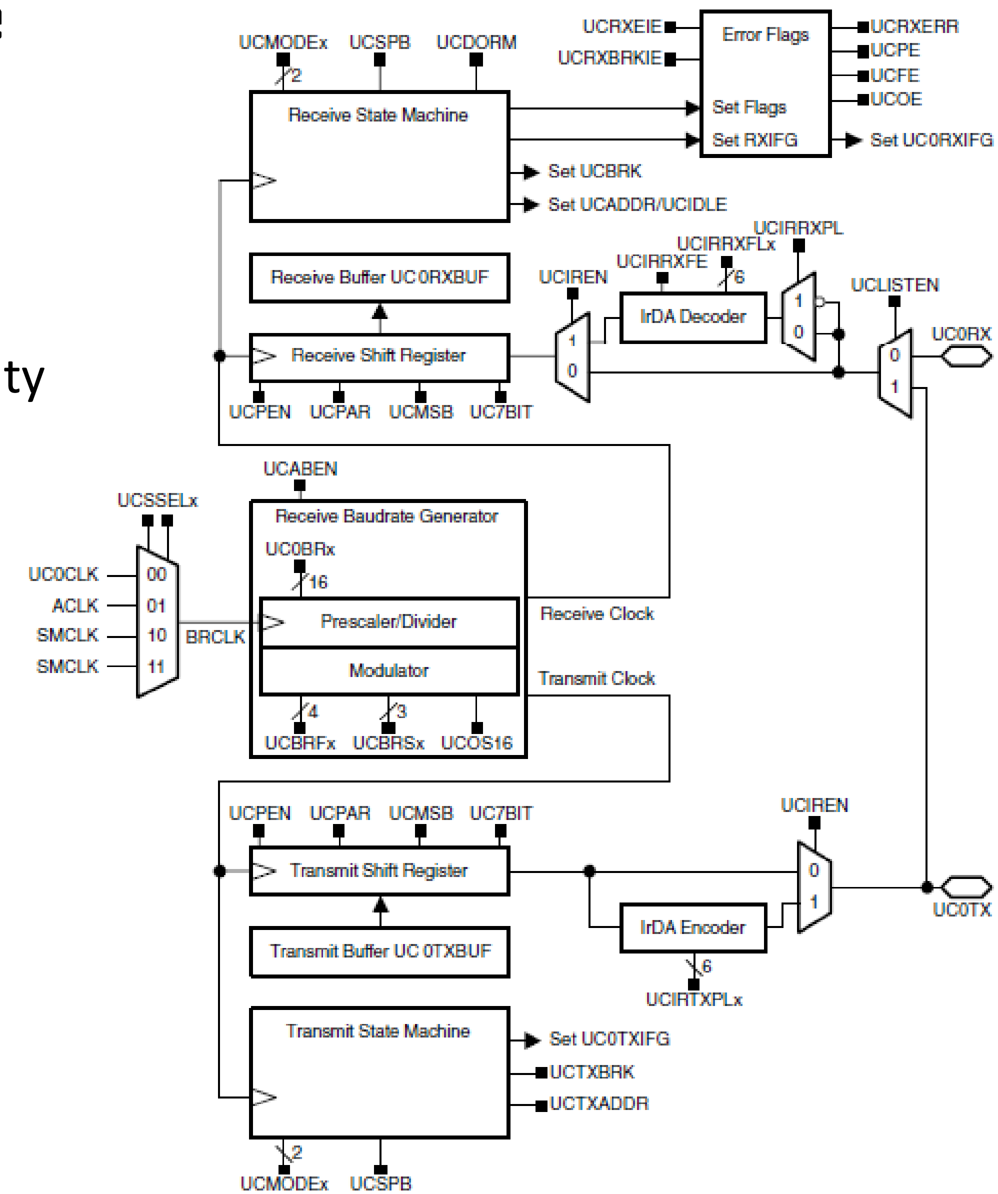
(also USART peripheral)

UART mode features include:

• 7- or 8-bit data; odd, even, or non-parity
• Independent transmit and receive
•LSB-first or MSB-first data
•Receiver start-edge detection for auto-wake up from LPMx modes
•Independent interrupt capability for receive and transmit
• Status flags for error detection and suppression

•Built-in idle-line and address-bit communication protocols for multiprocessor systems
• Status flags for address detection

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

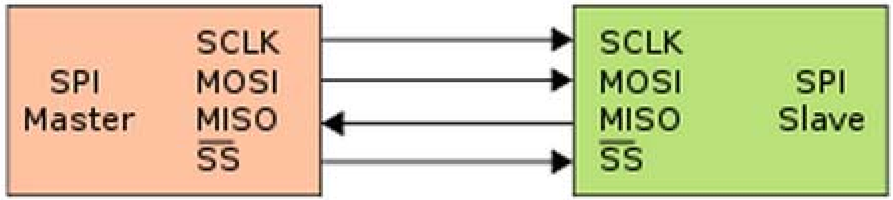


UART code

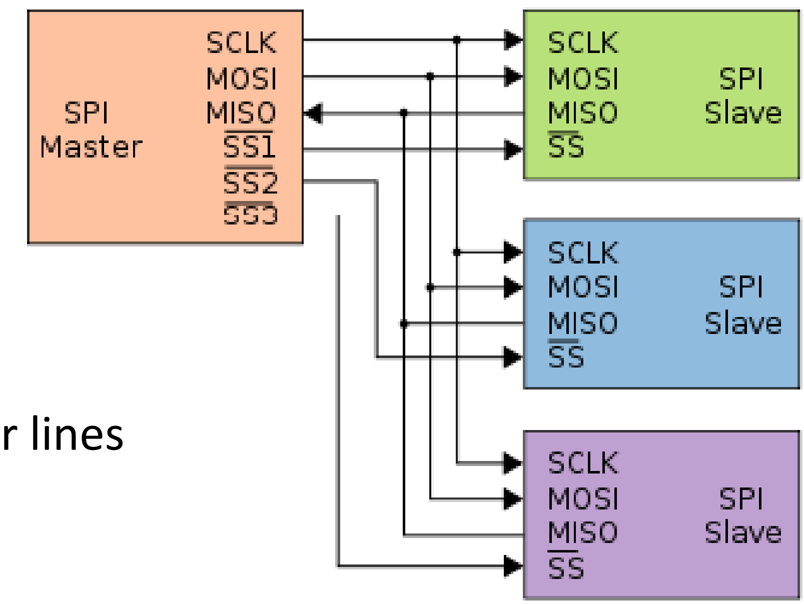
```
// Echo a received character, RX ISR used. Normal mode is LPM3,
// USCI_A0 RX interrupt triggers TX Echo.
// ACLK = BRCLK = LFXT1 = 32768, MCLK = SMCLK = DCO~1048k
// Baud divider, 32768hz XTAL @9600= 32768/9600= 3.41(0003h 03h )
//             -----------------
//         /|\|   MSP430xG461x  |-
//          | |             XIN|- 32kHz
//          --|RST          XOUT|-
//            |     P4.7/UCA0RXD|------------>
//            |                 | 9600 - 8N1
//            |     P4.6/UCA0TXD|<------------
```

```
#include  "msp430xG46x.h"

void main(void)
{
  volatile unsigned int i;

  P4SEL |= 0x0C0;                        // P4.7,6 = USCI_A0 RXD/TXD
  UCA0CTL1 |= UCSSEL_1;                  // CLK = ACLK
  UCA0BR0 = 0x03;                        // 32k/9600 - 3.41
  UCA0BR1 = 0x00;                        // User's manual has formulas for these
  UCA0MCTL = 0x06;                       // Modulation
  UCA0CTL1 &= ~UCSWRST;                  // **Initialize USCI state machine**
  IE2 |= UCA0RXIE;                       // Enable USCI_A0 RX interrupt

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

//  Echo back RXed character, confirm TX buffer is ready first
#pragma vector=USCIAB0RX_VECTOR
 _interrupt void USCIA0RX_ISR (void)
{
  while(!(IFG2&UCA0TXIFG));              // Make sure last character went out.
  UCA0TXBUF = UCA0RXBUF;                 // TX -> RXed character
}
```

SPI
(Serial Peripheral Interface - Motorola)



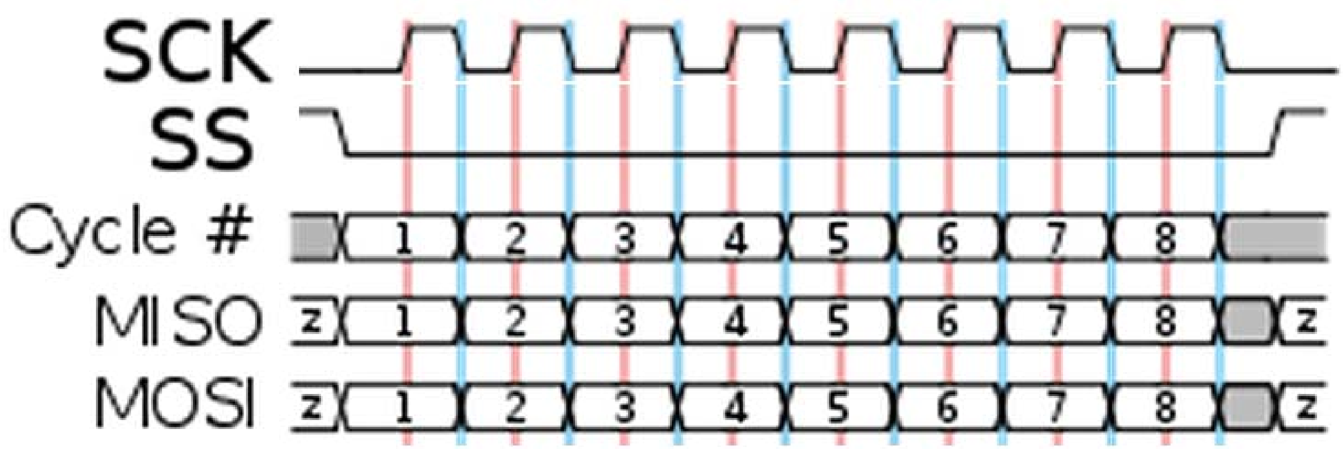
- Two types of devices, masters and slaves.
- We'll consider only one master, but multiple slaves.
- Signals
  - SCLK: Serial CLocK, set by Master
  - MOSI: Master Out, Slave In
  - MISO: Master In, Slave Out
  - ~SS: Slave Select
    - Each slave gets its own slave select (other lines are shared)
    - Pulling line low selects slave



SPI and the clock

(intro)

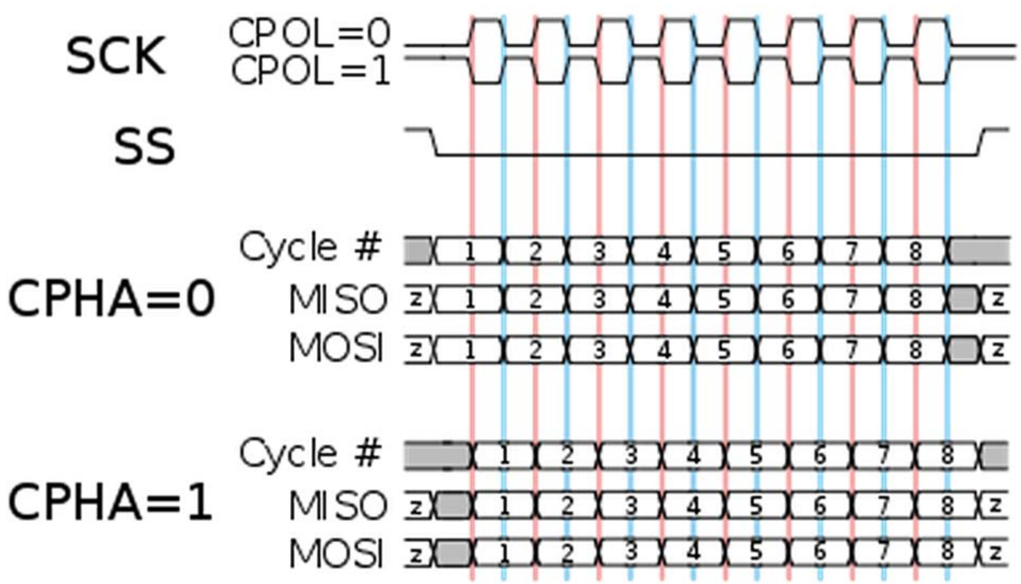
- Pull slave select line low to select device.
- First bit of data gets put on MISO and MOSI
  (so a byte goes both ways)
- Data gets shifted out (typically 8 bits, but not necessarily)
  - The data gets put on bus on falling edge of clock.
  - The data gets read on the rising edge of clock.



SPI and the clock
(the hard truth)

Unfortunately, clock can be set many ways as determined by clock polarity and phase.

- CPOL=0: Base value of the clock is 0
  - CPHA=0: Data read on rising edge, put on bus on falling edge of SCLK. (i.e., clock is low). (Case from previous slide)
  - CPHA=1: Data read on falling edge, put on bus on rising edge (i.e., clock is high).
- CPOL=1: Base value of the clock is 1
  - CPHA=0: Data read on falling edge, put on bus on rising edge (i.e., clock is high).
  - CPHA=1: Data read on rising edge, put on bus on falling edge (i.e., clock is low).

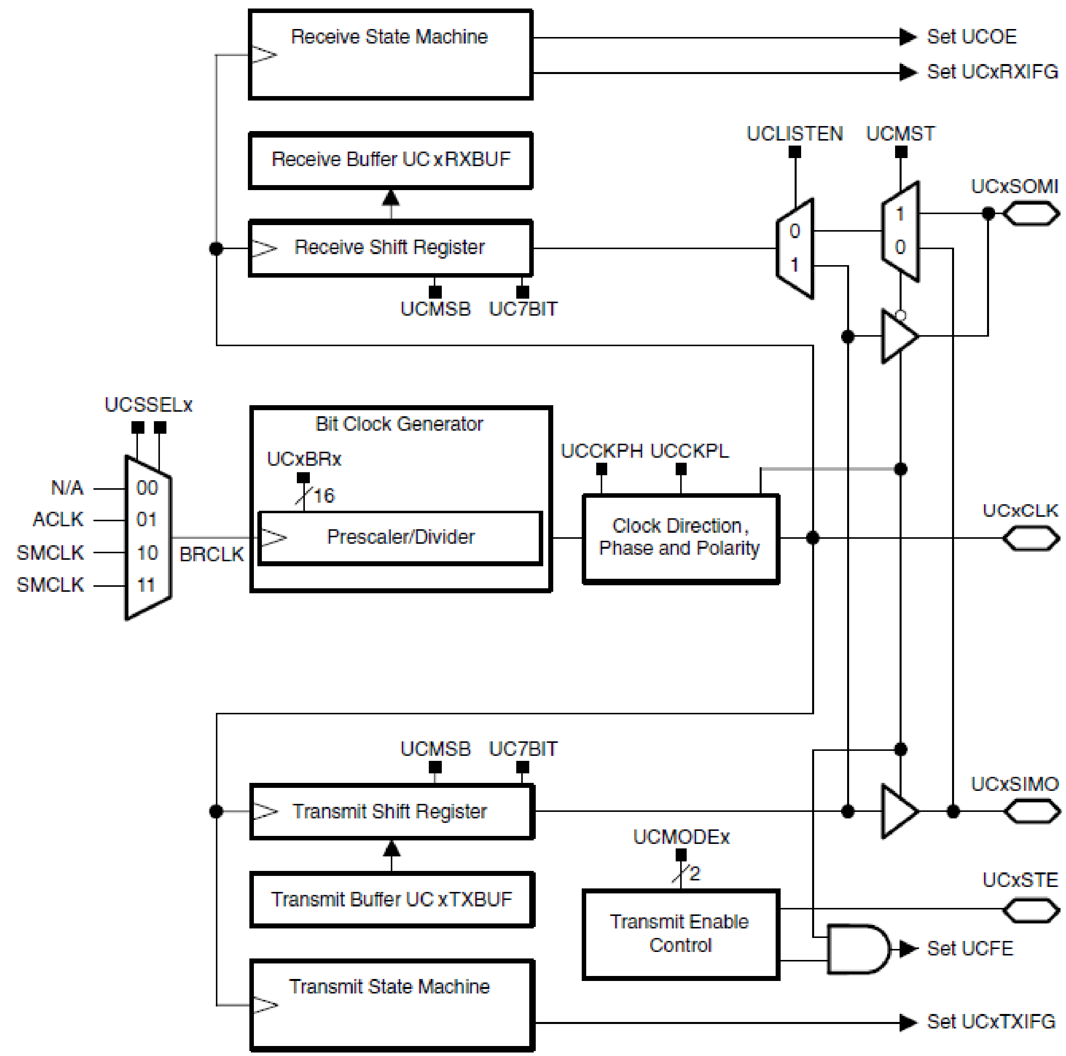


SPI and SCI

SPI mode features include:

- 7- or 8-bit data length
- LSB-first or MSB-first data
- Master or slave modes
- Selectable clock polarity and phase control
- Programmable clock frequency in master mode
- Independent transmit and receive
- Continuous transmit and receive
- Independent interrupt capability for receive and transmit
- Slave operation in LPM4

Figure 20−1. USCI Block Diagram: SPI Mode



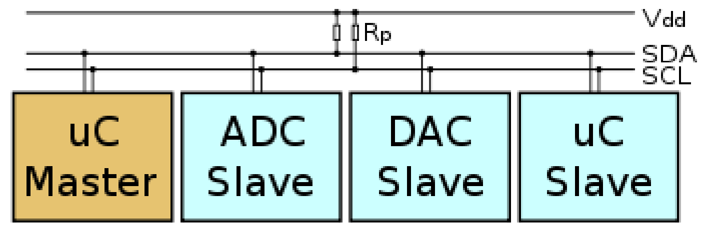
SPI Code

```
// MCLK = SMCLK = default DCO ~1048k, BRCLK = SMCLK/2
//                          -----------------------
//                      /|\|     MSP430xG461x      |
//        TLC549         | |                 XIN |-   32kHz
//     -------------     --|RST             XOUT|-
//    |            CS|<---|P3.0                  |
//    |       DATAOUT|--->|P3.2/UCB0SOMI         |
// ~>|  IN+   I/O CLK|<---|P3.3/UCB0CLK      P5.1|--> LED
```

```
#include  "msp430xG46x.h"

void main(void)
{
  volatile unsigned int i;
  char data;

  P5DIR |= 0x02;                          // P5.1 output
  P3SEL |= 0x0C;                          // P3.3,2 option select
  P3DIR |= 0x01;                          // P3.0 output direction
  UCB0CTL0 |= UCMST+UCSYNC+UCMSB;         // 8-bit SPI mstr, MSb 1st, CPOL=0, CPHS=0
  UCB0CTL1 |= UCSSEL_2;                   // SMCLK
  UCB0BR0 = 0x02;                         // Set Frequency
  UCB0BR1 = 0;
  UCB0CTL1 &= ~UCSWRST;                   // **Initialize USCI state machine**

  while(1)
  {
    P3OUT &= ~0x01;                       // Enable TLC549 (A/D) , ~CS (~SS) reset
    UCB0TXBUF = 0x00;                     // Dummy write to start SPI
    while (!(IFG2 & UCB0RXIFG));          // USCI_B0 RX buffer ready?

    data = UCB0RXBUF;                     // data = 00|DATA

    P3OUT |= 0x01;                        // Disable TLC549, ~CS (~SS) set

    if(data>=0x7F) P5OUT |= 0x02;         // data = AIN > 0.5(REF+ - REF-)?  LED On
    else P5OUT &= ~0x02;                  // LED off
   }
}
```

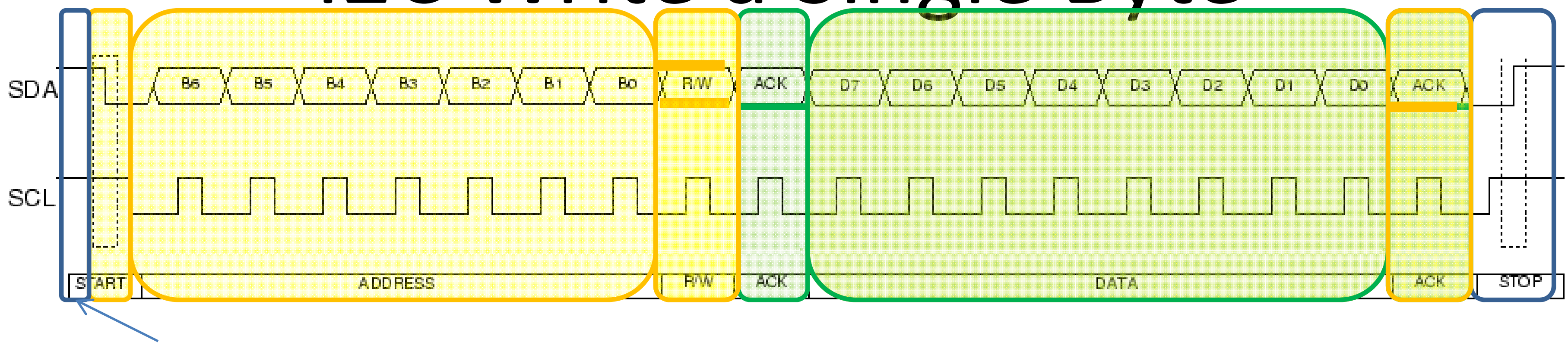
I2C or I^2C

(Inter-Integrated Circuit – Philips)



- As with SPI a master-slave system.
- Also called a 2-wire bus.
  It Has only clock and data, with pull-up resistors (Rp in diagram).
- Lines can be pulled low by any device, and are high when all devices release them.
- There are no “slave-select” lines – instead the devices have “addresses” that are sent as part of the transmission protocol.
- Four max speeds (100 kbS (*standard*), 400 kbS (*fast*), 1 MbS (*fast plus*), and 3.4 MbS (*high-speed*)

I2C Write a Single Byte

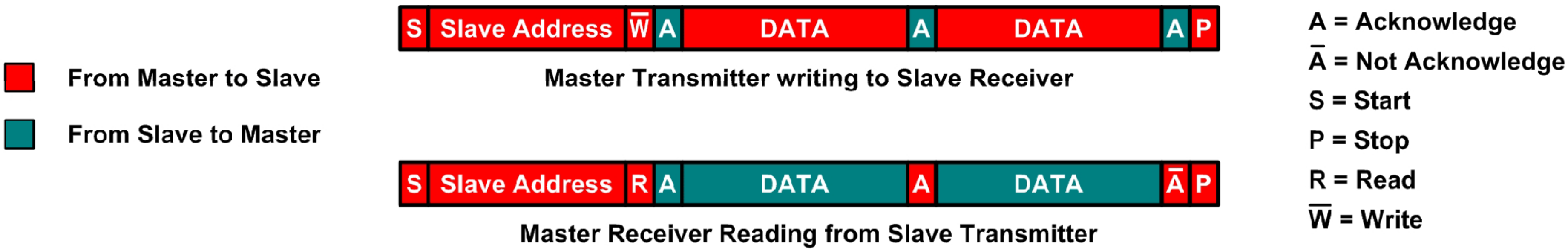


1. All: allow SDA, SCL start high
2. Master: SDA low to signal start
3. Master: Send out SCL, and 7 bit address followed by 0 (~W) on SDA
4. Slave: Pull SDA low to signify ACKnowledge
5. Master: Send out 8 data bits on SDA
6. Slave: Ack
7. All: allow SDA to go high when SCL is high (stop)

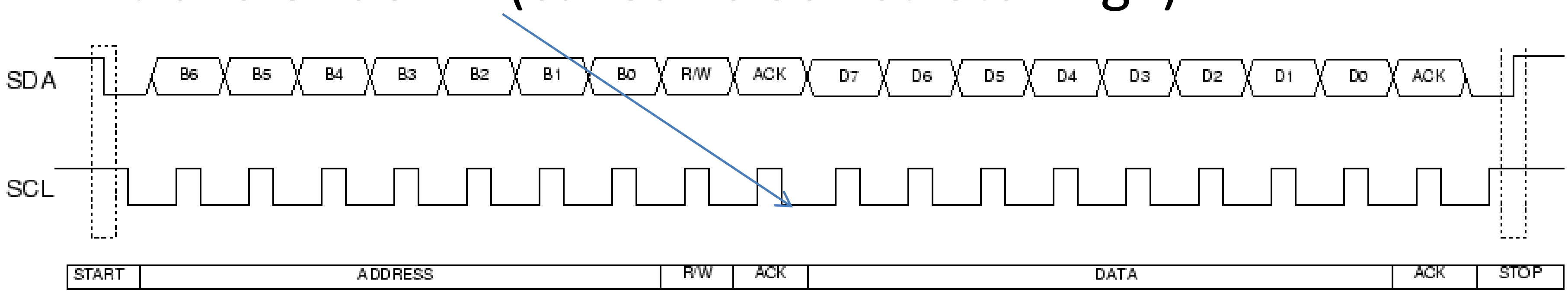
- **For “Read”,**
  3. Master: Address following by 1 (R) on SDA
  5. Slave: Send out 8 data bits on SDA
  6. Master: Ack

Other Features

- You can transfer multiple bytes in a row



- At end of transfer, slave can hold SCL low to slow transfer down (called “clock-stretching”)

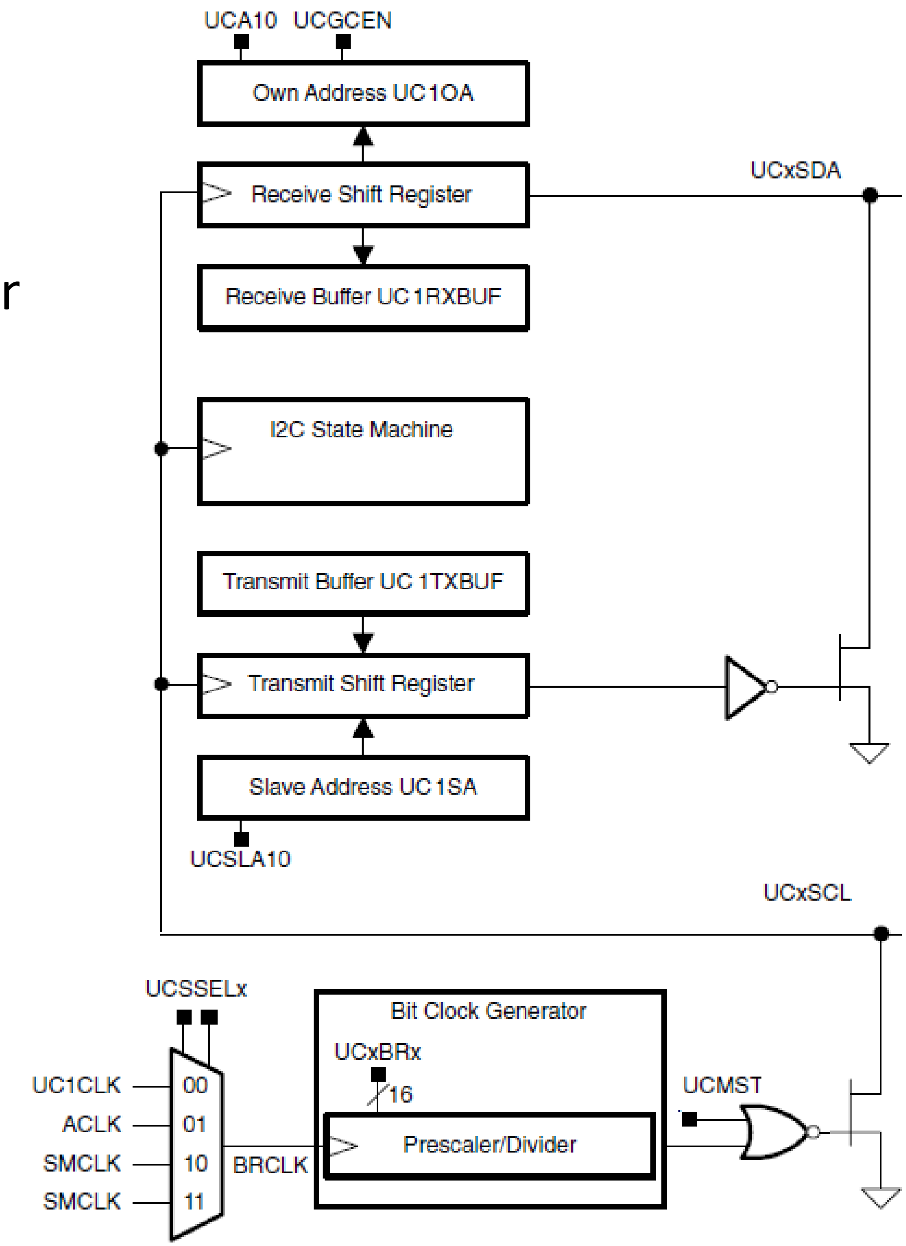


- Any device that malfunctions can disable bus.

I2C and SCI

The I2C features include:

- Compliance to Philips I2C specificatior
- Slave receiver/transmitter mode
- Standard mode up to 100 kbps and fast mode up to 400 kbps support
- Programmable UCxCLK frequency in master mode
- Designed for low power
- Slave receiver START detection for auto-wake up from LPMx modes
- Slave operation in LPM4



I2C Code

```
//   MSP430xG461x Demo - USCI_B0 I2C Master Interface to DAC8571, Write
//   Description: Using UCB0TXIE, a continuous sine wave is output to
//   external DAC using a 16-point look-up table. Only one start
//   is executed. Data is handled by the ISR and the CPU is in LPM0.
//   MCLK = SMCLK = TACLK = BRCLK = 1MHz
//   DAC8571 I2C address = 0x4C (A0 = GND)
//                MSP430xG461x                      DAC8571
//              ------------------                ------------
//            -|XIN   P3.1/UCB0SDA|<--------------->|SDA         |
//       32kHz |      P3.2/UCB0SCL|---------------->|SCL  I2C    |
//            -|XOUT              |                 |    SLAVE   |
//             |     I2C MASTER   |              GND|A0          |
```

```
void main(void) {
  WDTCTL = WDTPW + WDTHOLD;                 // Stop Watchdog Timer
  P3SEL |= 0x06;                            // Assign I2C pins to USCI_B0
  UCB0CTL1 |= UCSWRST;                      // Enable SW reset
  UCB0CTL0 = UCMST + UCMODE_3 + UCSYNC;     // I2C Master, synchronous mode
  UCB0CTL1 = UCSSEL_2 + UCSWRST;            // Use SMCLK, keep SW reset
  UCB0BR0 = 11;                             // fSCL = SMCLK/11 = 95.3kHz
  UCB0BR1 = 0;
  UCB0I2CSA = 0x4c;                         // Set slave address
  UCB0CTL1 &= ~UCSWRST;                     // Clear SW reset, resume operation
  IE2 |= UCB0TXIE;                          // Enable TX ready interrupt
  UCB0CTL1 |= UCTR + UCTXSTT;               // I2C TX, start condition
  UCB0TXBUF = 0x010;                        // Write DAC control byte
  __bis_SR_register(CPUOFF + GIE);          // Enter LPM0 w/ interrupts
}

// USCI_B0 Data ISR
#pragma vector = USCIAB0TX_VECTOR
__interrupt void USCIAB0TX_ISR(void) {
  static unsigned char ByteCtr;

  UCB0TXBUF = Sine_Tab[ByteCtr++];          // Transmit data byte
  ByteCtr &= 0x1f;                          // Do not exceed table
}
```

Wireless

- Order: Increasing complexity, power and bandwidth
  - SimpliciTI: <200 kbS
  - Zigbee (IEEE 802.15.4): 250 kbS
  - Bluetooth (IEEE 802.15.1): 1 MbS – 24 MbS
  - WiFi (IEEE 802.11): b 11 MbS; g 54 MbS; n 150 MbS

- Data rates needed
  - Voice: 4 kbS
  - Music: 700 kbS
  - Video: 3.5 MbS Standard; 40 MbS Blu-ray

References

- MSP430x4xx Family User’s Guide http://focus.ti.com/lit/ug/slau056j/slau056j.pdf
- MSP430FG4618/F2013 Experimenter’s Board User’s Guide http://focus.ti.com/lit/ug/slau213a/slau213a.pdf
- Serial Comm image http://www.ee.nmt.edu/~rison/ee308_spr99/supp/990406/sync_serial.gif
- RS-232 byte image http://www.eeherald.com/images/rs232-3.jpg
- RS-232 Connector Image http://www.bisque.com/tom/bluetooth/Images/db9.jpg
- SPI http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus
- I2C: http://en.wikipedia.org/wiki/I%C2%B2C
- I2C: http://www.best-microcontroller-projects.com/i2c-tutorial.html
- I2C: http://www.eetimes.com/design/analog-design/4010395/SIGNAL-CHAIN-BASICS-Part-32--Digital-interfaces-con-t---The-I2C-Bus