Today's topics:

- Serial I/O
  - general concepts in preparation for Lab 8

Introduction to Serial I/O

Serial communication transmits one bit of information at a time.

One bit is sent, a time delay occurs, next bit is sent.

Used to interface to printers, keyboards, scanners, etc.

Universal asynchronous receiver/transmitter (UART) is the interface chip that implements the transmission.

A serial channel is collection of signals (or wires) that implement the communication.

Data terminal equipment (DTE) is the computer.

Data communication equipment (DCE) is the modem.
A Serial Channel

<table>
<thead>
<tr>
<th>True/Mark</th>
<th>CMOS Level</th>
<th>RS232 Level</th>
<th>RS422 Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>False/Space</td>
<td>+5 V</td>
<td>TxD = -12 V</td>
<td>(TxD+ - TxD-) = -3 V</td>
</tr>
<tr>
<td>False/Space</td>
<td>+0.1 V</td>
<td>TxD = +12 V</td>
<td>(TxD+ - TxD-) = +3 V</td>
</tr>
</tbody>
</table>

Definitions

A frame is a complete and nondivisible packet of bits. Includes both information (e.g. data, characters) and overhead (start bit, error checking, and stop bits).

Parity is generated at the transmitter and checked at the receiver to help detect errors in transmission.

Even parity makes number of 1s even (data+parity).

Odd parity makes number of 1s odd (data+parity).

Bit time is the time between each bit.
Bandwidth

*Baud rate* is total number bits transmitted per time.
*Information* is data user wishes to transmit:
Characters to be printed.
*Overhead* is bits added to achieve transmission:
Start bit(s), stop bit(s), parity, etc.

\[
\text{Bandwidth} = \frac{\text{information bits/frame}}{\text{total bits/frame}} \times \text{baud rate}
\]

More Basics

- **Classic UART: DTE & DCE communications**

  - UART is the port
    - for Freescale this is SCI (serial communications interface)
      - this is just one instance
      - as usual signaling levels are TTL
      - interface logic can be used to convert to RS232 levels
        - e.g. MAX232, MC145407 chips
      - parity is generated by Tx side and checked by Rx side
Half Duplex Signalling

- **Half Duplex**
  - **normal usage**
    - fixed Tx and Rx side – a.k.a. Simplex signalling
  - **expanded version**
    - send can come from either side but only in one direction at a time
    - problem = collisions
      - solution same as with Ethernet CDMA
      - transmit & receive + compare
        - if Tx & Rx values aren't the same then collision & retry

Other Issues

- **Full Duplex**
  - more wires but 2 independent communication channels
    - concurrent send and receive buffers

- **Timing**
  - send and receive baud rates must be the same

- **Asynchronous (e.g. SCI)**
  - separate send and receive clocks
    - start sequence is used to synch clocks for the frame
      - model is that drift won't be enough to cause errors intra-frame
      - in high speed signaling (e.g. HT, QPI, etc.)
        - this is a big problem and requires complex and energy hungry circuitry
        - long transmission paths also require significant pre- and post-emphasis circuits

- **Synchronous: multiple options**
  - common clock (e.g. SPI)
  - Tx side clock – source synchronous signaling
Protocols & Specifications

- There are many
- Each one has specifications
  - electrical
    - what voltage levels mean what logical value
    - current sink and source requirements
  - cables
    - often limited to some max length
  - mechanical
    - what does the connector look like & pin function

SCI & SPI Illustrated

- Computer 1
  - DTE
  - SCI
  - TxD
  - RxD
  - Clock
  - E

- Computer 2
  - DTE
  - SCI
  - TxD
  - RxD
  - Clock
  - E

- Synchronous
  - Computer 1
    - DTE
    - SPI
    - MOSI
    - MISO
    - SCK

- Computer 2
  - DTE
  - SPI
  - MOSI
  - MISO
  - SCK

- Asynchronous
2 Common Freescale Options

Simple SCI

![Simple SCI Diagram]

SCI to RS232 conversion

![SCI to RS232 Conversion Diagram]

RS232 Output Specifications

- Must withstand
  - Short to ground
  - Short to any other wire

- Operating range:
  - True: \(-15 \leq V_{out} \leq -5\) V
  - False: \(+5 \leq V_{out} \leq +15\) V

- Maximum output voltage: \(-25 \leq V_{out} \leq +25\) V

- Short circuit current: \(I_{out} \leq 0.5\) A

- Transition (\(-3\) to \(+3\)) range time \(\leq 4\) ms

- Maximum slew rate: \(dV_{out}/dt \leq 30\) V/µs

Operating Range:
- True: \(-15 \leq V_{in} \leq -3\) V
- False: \(+3 \leq V_{in} \leq +15\) V

- Input resistance: \(30\) k\(\Omega \leq R_{in} \leq 70\) k\(\Omega\)

- Input capacitance including cable: \(C_{in} \leq 5\) pF

- Input open circuit voltage: \(E_{in} \leq 2\) V
RS232 DB9 Pin Assignments

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal</th>
<th>Description</th>
<th>True</th>
<th>DTE</th>
<th>DCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DCD</td>
<td>Data Carrier Detect</td>
<td>+12</td>
<td>In</td>
<td>Out</td>
</tr>
<tr>
<td>2</td>
<td>RxD</td>
<td>Receive Data</td>
<td>-12</td>
<td>In</td>
<td>Out</td>
</tr>
<tr>
<td>3</td>
<td>TxD</td>
<td>Transmit Data</td>
<td>-12</td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>4</td>
<td>DTR</td>
<td>Data Terminal Rdy</td>
<td>+12</td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>5</td>
<td>SG</td>
<td>Signal Ground</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>DSR</td>
<td>Data Set Ready</td>
<td>+12</td>
<td>In</td>
<td>Out</td>
</tr>
<tr>
<td>7</td>
<td>RTS</td>
<td>Request to Send</td>
<td>+12</td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>8</td>
<td>CTS</td>
<td>Clear to Send</td>
<td>+12</td>
<td>In</td>
<td>Out</td>
</tr>
<tr>
<td>9</td>
<td>RI</td>
<td>Ring Indicator</td>
<td>+12</td>
<td>In</td>
<td>Out</td>
</tr>
</tbody>
</table>

A Simple Serial Network
RS422/RS423/RS485 Specifications

<table>
<thead>
<tr>
<th>Specification</th>
<th>RS232D</th>
<th>RS423A</th>
<th>RS422</th>
<th>RS485</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drivers on one line</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td>Receivers on one line</td>
<td>1</td>
<td>10</td>
<td>10</td>
<td>32</td>
</tr>
<tr>
<td>Max distance (ft)</td>
<td>50</td>
<td>4,000</td>
<td>4,000</td>
<td>4,000</td>
</tr>
<tr>
<td>Max data rate</td>
<td>20kb/s</td>
<td>100kb/s</td>
<td>10Mb/s</td>
<td>10Mb/s</td>
</tr>
<tr>
<td>Max driver output</td>
<td>±25V</td>
<td>±6V</td>
<td>-0.25/+6V</td>
<td>-7/+12V</td>
</tr>
<tr>
<td>Receiver input</td>
<td>±15V</td>
<td>±12V</td>
<td>±7V</td>
<td>-7/+12V</td>
</tr>
</tbody>
</table>

Universal Serial Bus (USB)

Single host computer controls the USB. Host controls all transactions using a token-based protocol. Uses a tiered star topology with host at the center. Up to 127 devices can be connected to one USB bus. Plug’n’play implemented with dynamically loadable drivers. Host detects new devices and loads appropriate driver. Can operate at high (480Mb/s), full (12Mb/s), or low (1.5Mb/s) speeds.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Color</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Red</td>
<td>VBUS (5V)</td>
</tr>
<tr>
<td>2</td>
<td>White</td>
<td>D-</td>
</tr>
<tr>
<td>3</td>
<td>Green</td>
<td>D+</td>
</tr>
<tr>
<td>4</td>
<td>Black</td>
<td>Ground</td>
</tr>
</tbody>
</table>
Optical SCI Channel

Where & Why would you want to do this?

SCI

Most embedded microcomputers support SCI.
Common features include:
- A baud rate control register used to select transmission rate.
- A mode bit M used to select 8-bit (M=0) or 9-bit (M=1) data frames.
- Each device can create its own serial port clock with period that is integer multiple of the E clock period.
Transmitting in Asynchronous Mode

Common features in the transmitter:
- TxD data output pin, with TTL voltage levels.
- 10- or 11-bit shift register, not directly accessible.
- Serial communications data register (SCDR), write only, separate from receive reg. though same address.
- T8 data bit for 9-bit data mode.

Control Bits for the Transmitter

Transmit Enable (TE), set to 1 to enable transmitter.
Send Break (SBK), set to 1 to send blks of 10 or 11 0s.
Transmit Interrupt Enable (TIE), set to arm TDRE flag.
Transmit Complete Enable (TCIE), set to arm TC flag.

NOTE: Tx Data Reg Empty (TDRE) flag signals that the SCDR register is empty. TDRE is cleared by reading it. Different from previous flag clearing methods where you had to write a 1 to the flag. Read of TC flag (transmit complete) similarly clears it. Then write to the SCDR.
Transmission Illustrated

Pseudo Code for Transmission Process

```
TRANSMIT
Set TxD=0
Wait 16 clock times
Wait 1 bit time
Set n=0
Bit counter

TLOOP
Set TxD=bn
Wait 16 clock times
Output data bit
Set n=n+1
Goto TLOOP if n=7
Set TxD=T8
Wait 16 clock times
Output T8 bit
Set TxD=1
Wait 16 clock times
Output a stop bit
Wait 16 clock times
Wait 1 bit time
```

Page 11
Receiving in Asynchronous Mode

Common features in the receiver:
- RxD data input pin, with TTL voltage levels.
- 10- or 11-bit shift register, not directly accessible.
- Serial communications data register (SCDR), read only, separate from transmit reg. though same address.
- R8 data bit for 9-bit data mode.

Control Bits for the Receiver

Receiver Enable (RE), set to 1 to enable receiver.
Receiver Wakeup (RWU), set to 1 to allow a receiver input to wakeup the computer.
Receiver Interrupt Enable (RIE), set to arm RDRF flag.
Idle Line Interrupt Enable (ILIE), set to arm IDLE flag.
Status Bits Generated by the Receiver

Receive Data Register Full flag (RDRF), set when new data available, clear by reading RDRF and SCDR.
Receiver Idle flag (IDLE), set when receiver line is idle, clear by reading IDLE, then reading SCDR.
Overrun flag (OR), set when input data lost because previous frame not read, clear by reading OR and SCDR.
Noise flag (NF), set when input is noisy, clear by reading NF flag, then reading SCDR.
Framing error (FE), set when stop bit is incorrect, clear by reading FE, then reading SCDR.

Receiving Illustrated

![Diagram showing the process of receiving data with status bit generation.](image-url)
Pseudo Code for Receive Process

RECEIVE Goto RECEIVE if RxD=1  Wait for start bit
   Wait 8 clock times  Wait half a bit time
   Goto RECEIVE if RxD=1  False start?
   Set n=0

RLOOP  Wait 16 clock times  Wait 1 bit time
   Set bn=RxD  Input data bit
   Set n=n+1
   Goto RLOOP if n<=7
   Wait 16 clock times  Wait 1 bit time
   Set R8=RxD  Read R8 bit
   Wait 16 clock times  Wait 1 bit time
   Set FE=1 if RxD=0  Framing error if no stop bit

9S12C32 SCI Details

One SCI port using Port S bits 1 and 0.
Least significant 13 bits of SCIBD register determine baud rate.
SCICR2 register contains control bits for SCI (see Table 7.11).
SCICR1 register contains other miscellaneous SCI control bits.

LOOPS: disconnects receiver from RxD pin.
RSRC: when LOOPS=1, RSRC=0 connects receiver to transmitter internally while RSRC=1 connects receiver to TxD.
WAKE: 0 wakeup on IDLE line, 1 wakeup on address mark.
ILT: determines if idle line count starts from start or stop bit.
SWAI: setting this bit causes SCI to shutdown and pause any communication.
PE: setting this bit enables parity checking.
PT: 0 is even parity while 1 is odd parity.
More SCI Details

Flags in SCISR1 register can be read but not modified by software.
The error conditions are also reported in the SCISR1 register
including parity flag, PF, set on parity errors.
The SCISR2 register contains two mode control and one status bit.
  BRK13: break character is 13 or 14 bits when 1 and 10 or 11
  bits when 0.
  TXDIR: specifies direction of the TxD pin in single-wire
  mode.
  RAF: 1 when frame is being received.
The SCIDRL register contains data transmitted and received.
The SCIDRH register contains the 9th data bits.

SCI I/O Interrupts

Simultaneous input and output requires two FIFOs.
RxFifo passes data from InSCI handler to main thread.
TxFifo passes data from main thread to OutSCI handler.
Since TxFifo initially empty, transmit interrupts initially disarmed.
When main thread calls OutChar, transmit interrupts armed.
Interrupt handler disarms transmit interrupts when TxFifo becomes empty.
SCI Rx, Tx, & ISR's

SCI Interface Ritual

```c
void SCI_Init(void)
{
    asm sei
    RxFifo_Init(); // empty FIFOs
    TxFifo_Init();
    SCIBD = 26;     // 9600 bits/sec
    SCICR1 = 0;     // M=0, no parity
    SCICR2 = 0x2C;  // enable, arm RDRF
    asm cli         // enable interrupts
}
```
SCI Interface ISR

// RDRF set on new receive data
// TDRE set on empty transmit register
interrupt 20 void SciHandler(void){
  char data;
  if(SCISR1 & RDRF){
    RxFifo_Put(SCIDRL); // clears RDRF
  }
  if((SCICR2&TIE)&&(SCISR1&TDRE)){
    if(TxFifo_Get(&data)){
      SCIDRL = data; // clears TDRE
    }
    else{
      SCICR2 = 0x2c; // disarm TDRE
    }
  }
}

SCI In/Out Character

// Input ASCII character from SCI
// spin if RxFifo is empty
char SCI_InChar(void){ char letter;
  while (RxFifo_Get(&letter) == 0){};
  return(letter);
}

// Output ASCII character to SCI
// spin if TxFifo is full
void SCI_OutChar(char data){
  while (TxFifo_Put(data) == 0){};
  SCICR2 = 0xAC; // arm TDRE
}
Serial Port Printer Interfaces

Printer bandwidth may be less than the maximum bandwidth supported by the serial channel.
Special characters may require more time to print.
Most printers have internal FIFOs that could get full.
The printer may be disconnected.
The printer may be deselected.
The printer power may be off.

Flow control is needed to synchronize computer with variable rate output device
DTR: data terminal ready
XON/XOFF

Note: 2 approaches
DTR is a handshake saying send me another frame
Xoff is a shut up signal – more efficient for larger buffers
but some timing complexity for on the fly & response time issues

SCI Simplex Printer Interface (w/ DTR handshake)
Serial Output w/ DTR

```c
void Printer_Init(void)
{
    asm sei
    Tx Fifo_Init(); // empty FIFOs
    SCI BRD = 52;   // 9600 bits/sec
    SCICR1 = 0;    // M=0, no parity
    SCICR2 = 0x0C; // enable disarm TDRE
    TlDS &=~0x08;  // PT3 input capture
    DORT &=~0x08;  // PT3 is input
    TSCR1 = 0x80;  // enable TCNT
    TCTL4 |= 0xCO; // both rise and fall
    TIE |= 0x08;   // Arm IC3
    TFLG1 = 0x08; // initially clear
    asm cli         // enable interrupts
}
```

DTR Handshake ISR

```c
// IC3 interrupt on any change of DTR
void interrupt 11 IC3Han(void) {
    TFLG1 = 0x08; // Ack, clear C3F
    checkIC3();   // Arm SCI if DTR=+12
}
```
Serial Output to Printer

```c
void checkIC3(void){
    if((PTT&0x08) // PT3=1 if DTR=-12
        SC1CR2 = 0x0C; // busy, so disarm
    else
        SC1CR2 = 0x8C; // not busy, so arm
}
// Output ASCII character to Printer
// spin if TxFifo is full
void Printer_OutChar(char data){
    while (Tx FIFO_Put(data) == 0){};
    checkIC3();
}
```

Using XON/XOFF (busy waiting)
Synchronous = SPI (3 options)

Two devices communicating with SCI operate at same frequency but have 2 separate (not synchronized) clocks.

Two devices communicating with SPI operate using the same (synchronized) clock.

Master device creates the clock while slave device(s) use the clock to latch data in or out.

SPI Master/Slave Example
SPI Fundamentals

Motorola SPI includes four I/O lines:
- S$ - slave select, used by master to indicate the channel is active.
- SCK - 50% duty cycle clock generated by the master.
- MOSI (master-out slave-in) - data line driven by master.
- MISO (master-in slave-out) - data line driven by slave.

Transmitting device uses one edge of clock to change data, and receiving device uses other edge to accept data.

When data transfer occurs combined 16-bit register is serially shifted eight bit positions (data exchanged).

More SPI Fundamentals

Common control features of the SPI module include:
- A baud rate control register
- A mode bit in the control register to select master versus slave, clock polarity, clock phase.
- Interrupt arm bit
- Ability to make outputs open-drain.

Common status bits for the SPI module include:
- SPIF, transmission complete
- WCOL, write collision
- MODF, mode fault

Mode fault occurs when master and slave synchronization is wrong – e.g. 2 masters
SPI Pseudo Code

TRANSMIT
Set n=7
On fall of Sck, set Data=bn
Set n=n-1
Goto TLOOP if n>=0
Set Data=1
Goto TLOOP if n>=0

RECEIVE
Set n=7
On rise of Sck, read data
Set bn=Data
Set n=n-1
Goto RLOOP if n>=0

SPI Modes

CPOL sets SCLK polarity – e.g. what is IDLE
CPHA sets even or odd clock edges for the receiver shift register
9S12C32 SPI Details (Port M)

Uses four pins, PM3 = SS, PM5 = SCLK, PM4 = MOSI, and PM2 = MISO.
If 6812 is master, set DDRM to make PM5, PM4, and PM3 outputs.
Can be in run, wait, or stop mode.

<table>
<thead>
<tr>
<th>SPR0</th>
<th>SPR1</th>
<th>Div</th>
<th>4 MHz Freq</th>
<th>4 MHz Bit Time</th>
<th>24 MHz Freq</th>
<th>24 MHz Bit Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2 MHz</td>
<td>500 ns</td>
<td>12 MHz</td>
<td>63.3 ns</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1 MHz</td>
<td>1 µs</td>
<td>6 MHz</td>
<td>166.7 ns</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td>500 kHz</td>
<td>2 µs</td>
<td>3 MHz</td>
<td>333.3 ns</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>16</td>
<td>250 kHz</td>
<td>4 µs</td>
<td>1.5 MHz</td>
<td>666.7 ns</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>32</td>
<td>125 MHz</td>
<td>8 µs</td>
<td>750 kHz</td>
<td>1.33 µs</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>64</td>
<td>62.5 kHz</td>
<td>16 µs</td>
<td>375 kHz</td>
<td>2.67 µs</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>128</td>
<td>31.25 kHz</td>
<td>32 µs</td>
<td>187.5 kHz</td>
<td>5.33 µs</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>256</td>
<td>15.625 kHz</td>
<td>64 µs</td>
<td>93.75 kHz</td>
<td>10.67 µs</td>
</tr>
</tbody>
</table>

SPIBR register

SPI Control Registers

SPIDR is 8-bit register used for both input and output.
SPICR1 register species SPI mode of operation.
- SPE: enables the SPI system.
- SPIE: arms interrupts on the SPIF flag.
- SPTIE: arms interrupts on the SPTEF flag.
- LSBF: if 1 transmits least significant bit first.

SPISR register contains flags for the SPI system.
- SPIF: indicates that new data is available to be read.
- SPTEF: indicates that SPI data register can accept new data.
- MODF: mode error interrupt status flag.
### SPI Modes

<table>
<thead>
<tr>
<th>SPI Mode Selections</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MODFEN</strong></td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pin Mode</th>
<th><strong>MSTR</strong></th>
<th><strong>SPCO</strong></th>
<th><strong>BIDIROE</strong></th>
<th><strong>MISO</strong></th>
<th><strong>MOSI</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>Master In</td>
<td>Master Out</td>
</tr>
<tr>
<td>Bidirectional</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>MISO not used</td>
<td>Master In</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>Master I/O</td>
</tr>
<tr>
<td>Normal</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>Slave Out</td>
<td>Slave In</td>
</tr>
<tr>
<td>Bidirectional</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Slave In</td>
<td>MOSI not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Slave I/O</td>
<td></td>
</tr>
</tbody>
</table>
Concluding Remarks

• Serial I/O is very common
  • USB is obviously everywhere
  • SPI & SCI are more prevalent in embedded systems
    » primarily because it’s low cost
    » most controllers support this
      • your kits support both
    » difference is synch vs. asynch

• Too much detail already
  • but advise that you take a look at the DAC application
  • we’ll go through the full SCI ritual next lecture
    » in prep for Lab 8

• SPRING BREAK
  • hope you have some fun
  • hope I can catch up