Introduction to Serial Communication

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

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

SCI versus SPI

- **Asynchronous**
  - Computer 1: SCI, DTE
  - Computer 2: SCI, DTE

- **Synchronous**
  - Computer 1: SPI, DTE
  - Computer 2: SPI, DTE
Various Serial Channels

A Desktop Network

RS232 Cables

RS232 DB9 Pin Assignments

RS232 Interface

More RS232 Interface Chips
RS232 Specifications

- 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/unloadable 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 Channel

- Fiber optics
- Two optic cables
- Light
- Sensor
- Ground
- Not connected

Modem Serial Interface

- Logic Originate Answer
- True 1270 Hz 2225 Hz
- False 1070 Hz 2025 Hz

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/unloadable drivers.
- Host detects new devices and loads appropriate driver.
- Can operate at high (480Mb/s), full (12Mb/s), or low (1.5Mb/s) speeds.

Current Loop Serial Channel

- Current loop
- LED/LST
- Not connected
- Ground
**Digital Logic Channel**

![Diagram of Digital Logic Channel]

**Serial Communication Interface**

- 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.

**Status Bits Generated by the Transmitter**

- Transmit Data Register Empty flag (TDRE), set when SCDR empty, clear by reading TDRE and writing SCDR.
- Transmit Complete flag (TC), set when transmit shift register done shifting, cleared by reading TC flag then writing SCDR.

**Figures for Transmission**

![Figures for Transmission]
### Pseudo Code for Transmission Process

**TRANSMIT**
- Set TxD=0: Output start bit
- Wait 16 clock times: Wait 1 bit time
- Set n=0: Bit counter

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

---

### Receiving in Asynchronous Mode

- **Common features in the receiver:**
  - Rx2: 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 wake up 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.

---

### Figures for Receiving

- **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.

---

### Pseudo Code for Receive Process

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

**RLOOP**
- Wait 16 clock times: Wait 1 bit time
- Set bn=Rx: Input data bit
- Set n=n+1
- Goto RLOOP if n<=7
- Wait 16 clock times: Wait 1 bit time
- Set R8=Rx: Read R8 bit
- Wait 16 clock times: Wait 1 bit time
- Set FE=1 if Rx=0: Framing error if no stop bit
MC9S12C32 SCI Details

- One SCI port using Port S bits 1 and 0.
- Least significant 13 bits of SCIBD register determine baud rate.

\[
\text{SCIBaudRate} = \frac{E}{16 \times \text{SCIBD}}
\]

- 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.

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.

Input and Output Interrupts on the SCI

- 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 Software

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 Ritual

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&TXE) & (SCISR1&TDRE))
  {
    if(TxFifo_Get(&data))
    {
      SCIDRL = data; // clears TDRE
    }
    else{
      SCICR2 = 0x2C; // disarm 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 (ex. DTR or XON/XOFF).

Serial Output Using DTR Synchronization Ritual

```c
void Printer_Init(void){
    asm sei
    TxFifo_Init(); // empty FIFOs
    SCIBD = 52; // 9600 bits/sec
    SCICR1 = 0; // M=0, no parity
    SCICR2 = 0x0C; // enable disarm TDRE
    TIOS &=~0x08; // PT3 input capture
    DDRT &=~0x08; // PT3 is input
    TSCR1 = 0x80; // enable TCNT
    TCTL4 |= 0xC0; // both rise and fall
    TIE |= 0x08; // Arm IC3
    TFLG1 = 0x08; // initially clear
    asm cli // enable interrupts
}
```

Serial Output Using DTR Synchronization ISRs

```c
void checkIC3(void){
    if(PTT&0x08) // PT3=1 if DTR=-12
        SCICR2 = 0x0C; // busy, so disarm
    else
        SCICR2 = 0x8C; // not busy, so arm
}
```

 SCI In/Out Character

```c
// Input ASCII character from SCI
char SCI_InChar(void){ char letter;
while (RxFifo_Get(&letter) == 0){}
return(letter);
}
```

 SCI Simplex Interface with DTR Handshaking

![Serial Port Printer Interfaces Diagram](image1)

Serial Output Using DTR Synchronization Ritual

```c
void Printer_Init(void){
    asm sei
    TxFifo_Init(); // empty FIFOs
    SCIBD = 52; // 9600 bits/sec
    SCICR1 = 0; // M=0, no parity
    SCICR2 = 0x0C; // enable disarm TDRE
    TIOS &=~0x08; // PT3 input capture
    DDRT &=~0x08; // PT3 is input
    TSCR1 = 0x80; // enable TCNT
    TCTL4 |= 0xC0; // both rise and fall
    TIE |= 0x08; // Arm IC3
    TFLG1 = 0x08; // initially clear
    asm cli // enable interrupts
}
```
Use of XON/XOFF to Interface a Printer

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.

Synchronous Communication Using the SPI

A Synchronous Serial Interface

Motorola SPI includes four I/O lines:
1. SS - slave select, used by master to indicate the channel is active.
2. SCK - 50% duty cycle clock generated by the master.
3. MOSI (master-out slave-in) - data line driven by master.
4. 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).

SPI Fundamentals

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

Common status bits for the SPI module include:
1. SPIF, transmission complete
2. WCOL, write collision
3. MODF, mode fault

Pseudo Code for SPI Communication

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

RECEIVE
Set n=7
RLOOP
On rise of Sck, read data
Input bit
Set bn=Data
Set n=n-1
Goto RLOOP if n>=0
**Synchronous Serial Modes**

- Uses four pins, \( PM_3 = SS, PM_5 = SCLK, PM_4 = MOSI, \) and \( PM_2 = MISO. \)
- If 6812 is master, set \( DDRM \) to make \( PM_5, PM_4, \) and \( PM_3 \) outputs.
- Can be in **run**, **wait**, or **stop** mode.

<table>
<thead>
<tr>
<th>SPR2</th>
<th>SPR1</th>
<th>SPR0</th>
<th>Div</th>
<th>Freq</th>
<th>Bit Time</th>
<th>Freq</th>
<th>Bit Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2 MHz</td>
<td>500 ns</td>
<td>12 MHz</td>
<td>83.3 ns</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>8</td>
<td>500 kHz</td>
<td>2 ( \mu )s</td>
<td>3 MHz</td>
<td>333.3 ns</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>16</td>
<td>250 kHz</td>
<td>4 ( \mu )s</td>
<td>1.5 MHz</td>
<td>666.7 ns</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>32</td>
<td>125 MHz</td>
<td>8 ( \mu )s</td>
<td>750 kHz</td>
<td>1.33 ( \mu )s</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>64</td>
<td>62.5 kHz</td>
<td>16 ( \mu )s</td>
<td>375 kHz</td>
<td>2.67 ( \mu )s</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>128</td>
<td>31.25 kHz</td>
<td>32 ( \mu )s</td>
<td>187.5 kHz</td>
<td>5.33 ( \mu )s</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>256</td>
<td>15.625kHz</td>
<td>64 ( \mu )s</td>
<td>93.75 kHz</td>
<td>10.67 ( \mu )s</td>
</tr>
</tbody>
</table>

**MC9S12C32 SPI Details**

- **SPIDR** is 8-bit register used for both input and output.
- **SPICR1** register specifies 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 new data is available to be read.
  - **SPTEF**: indicates SPI data register can accept new data.
  - **MODF**: mode error interrupt status flag.

**Mode Selections**

- **MODFEN**
  - **SSOE**
    - **Master Mode** (MSTR=1)
      - Slave Mode (MSTR=0)
    - 0 0: PM3 not used with SPI
    - 0 1: PM3 not used with SPI
    - 1 0: PM3 is SS input w/MODF
    - 1 1: PM3 is SS output
- **Pin Mode**
  - **MSTR**
  - **SPCO**
  - **BIDIROE**
  - **MISO**
  - **MOSI**

**Digital-to-Analog Converter**

- Embedded microcontroller
- SPI connection
- DCO43
- +12V
- VREF
- Q0
- 15.83 kHz
- MOSI
- SRI
- **Digital-to Analog Converter**
SPI Interfacing Issues

- **Word size** - need to transmit 12 bits to DAC, send 16-bits, first 4 bits will be ignored.
- **Bit order** - DAC8043 requires most significant bit first.
- **Clock phase/polarity** - DAC8043 samples on rising edge, so SPI must change data on falling edge.
- **Bandwidth** - DAC8043 has minimum clock low width of 120ns, so shortest SPI period is 250ns.

# Initialization for D/A Interface Using the SPI

```c
void DAC_Init(void){ // PM3=LD=1
  DDRM |= 0x38; // PM5=CLK=SPI clock out
  PTM |= 0x08; // PM4=SRI=SPI master out
  /* bit SPICR1
  7 SPIE = 0 no interrupts
  6 SPE = 1 enable SPI
  5 SPTIE= 0 no interrupts
  4 MSTR = 1 master
  3 CPOL = 0 output changes on fall, 2 CPHA = 0 clock normally low
  1 SSOE = 0 PM3 regular output, LD
  0 LSBF = 0 most sign bit first */
  SPICR1 = 0x50;
  SPICR2 = 0x00; // normal mode
  SPIBR = 0x00; // 2MHz}
```

D/A Interface Using the SPI

```c
#define SPIF 0x80
void DAC_out(unsigned short code){
  // better implementation using SPTEF
  unsigned char dummy;
  while((SPISR&SPTEF)==0); // wait for transmit empty
  SPIDR = (code>>8); // mbyte
  dummy = SPIDR; // clear SPIF
  while((SPISR&SPTEF)==0); // wait for transmit empty
  SPIDR = code; // lbyte
  dummy = SPIDR; // clear SPIF
  Timer_Wait(10); // wait for SPI output completion
  PTM |= ~0x08; // PM3=LD=0
  PTM |= 0x08; // PM3=LD=1
}
```

Analog-to-Digital Converter

Initialization for ADC Interface Using the SPI

```c
void ADC_Init(void){ // PM3=CS=1
  DDRM |= 0x38; // PM5=SCLK=SPI clock out
  DDRM &~0x04; // PM2=DOUT=SPI master in
  PTM |= 0x08; // PM4=DIN=SPI master out
  /* bit SPICR1
  7 SPIE = 0 no interrupts
  6 SPE = 1 enable SPI
  5 SPTIE= 0 no interrupts
  4 MSTR = 1 master
  3 CPOL = 0 output changes on fall,
  2 CPHA = 0 clock normally low
  1 SSOE = 0 PM3 regular output, LD
  0 LSBF = 0 most sign bit first */
  SPICR1 = 0x50;
  SPICR2 = 0x00; // normal mode
  SPIBR = 0x00; // 2MHz
```

SPI Interfacing Issues

- **Word size** - need to transmit 8 bits to ADC, then receive 12 bits back. MAX1247 sends it as 2 8-bit transmissions.
- **Bit order** - MAX1247 requires most significant bit first.
- **Clock phase/polarity** - MAX1247 samples on rising edge, so SPI must change data on falling edge.
- **Bandwidth** - MAX1247 has maximum SCLK frequency of 2 MHz, so shortest SPI period is 500ns.
ADC Interface Using the SPI

```
unsigned short ADC_in(unsigned char code){
unsigned short data; unsigned char dummy;
PTM &= ~0x08; // PM3=CS=0
while((SPISR&SPTEF)==0); // wait for transmit empty
SPIDR = code; // set channel,mode
while((SPISR&SPIF)==0); // gadfly wait
dummy = SPIDR; // clear SPIF
SPIDR = 0; // start SPI
while((SPISR&SPIF)==0); // gadfly wait
data = SPIDR<<8; // msbyte of ADC
SPIDR = 0; // start SPI
while((SPISR&SPIF)==0); // gadfly wait
data += SPIDR; // lsbyte of ADC
PTM |= 0x08; // PM3=CS=1
return data>>3; // right justify
}
```
short in9(void){
    short result;
    DDRM &=~0x10; // PM4=DQ input
    while((SPISR&SPTEF)==0); // wait for transmit empty
    SPIDR = 0; // start shift register
    while((SPISR&SPIF)==0); // wait for SPIF
    result = SPIDR; // get LS data, clear SPIF
    while((SPISR&SPTEF)==0); // wait for transmit empty
    SPIDR = 0; // start shift register
    while((SPISR&SPIF)==0); // wait for SPIF
    if(SPIDR&0x01) // get MS data, clear SPIF
        result |= 0xFF00; // negative
    else
        result &=~0xFF00; // positive
    DDRM |= 0x10; // PM4=DQ output
    return result;
}