Instructors:
Dr. Phillip Jones

Overview of the Lecture

- Concepts behind Serial Communication
- ATmega128 USART Programming Interface
- Initializing USART, transmitting and receiving data

Serial Communication

- USART = Universal Synchronous & Asynchronous Serial Receiver & Transmitter
- Asynchronous (no common clock)
- Can transmit over long link distances
- Uses start and stop to sandwich data bits
- parity bit can be used for error detection

Serial Byte Format

![Serial Byte Format Diagram]

Example: Sending byte value 01000001
### Start and Stop Bits

- **Idle period**: logic high
- **Start bit**: logic low, 1 bit
- **Stop bit**: logic high, 1 bit or 2 bits

### Parity Bit

- **Three choices**: even, odd, or none
- \[ P_{\text{even}} = d_{n-1} \oplus \cdots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0 \]
- \[ P_{\text{odd}} = d_{n-1} \oplus \cdots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1 \]

*If one bit is flipped, how to detect it?*

### Baud Rate

- **How to define communication speed?**
- **Baud rate**: Number of symbols transferred per second
  - Same as bit rate (bps) for USART
- Baud rate is **not** data rate
  - With 56,000 bps, 8-data bit frame, two stop bits and parity bit used, what is the maximum data rate?

### Programming USART

- Both sides of communication should use the same **frame format** and **baud rate**

- **Frame format**:
  - Number of data bits in the frame: 5, 6, 7, 8 or 9
  - Number of stop bits: 1 or 2
  - Parity bit: Odd, Even, or None

### USART Programming Interface

- **UCSRnA, UCSRnB, UCSRnC**: Control and Status Registers
  - Three 8-bit registers for control and status checking
  - \( n \) is either 0 or 1, e.g. UCSR0A is for USART0
  - There are two USART units, USART0 and USART1; USART1 used for communication with iRobot Create

- **UBRRnH and UBRRnL**: Baud Rate Registers
  - Two 8-bit registers used together as 16-bit register

- **UDRn**: 8-bit Register for reading and writing data
Datasheet Page Numbers

- Trying to set UCSRnA/B/C of the USART?
  - Review pages 188 to 192 of the ATmega128 User Guide

- Setting the baud rate register (UBRR0)?
  - See the table on page 196 of the ATmega128 User Guide

- Need code examples for reading / writing data?
  - Page 176 (initialization example)
  - Page 177 (transmit example)
  - Page 180 (receive example)
  - Also reproduced on upcoming slides

Serial (ATmega128)

- This time will be spent reviewing the individual bit positions inside of UCSR0A, UCSR0B, UCSR0C from pages 188 to 192 of the ATmega128 data sheet

UCSRnA: Mostly a Status Register

- Bit 7 – RXCn: UART Receive Complete
  This flag bit is set when there are unread data in the receive buffer and cleared when the receive buffer is empty

- Bit 6 – TXCn: UART Transmit Complete
  This flag bit is set when the entire frame in the Transmit Shift Register has been shifted out

- Bit 4 – FEn: Frame Error
- Bit 3 – DORn: Data OverRun
- Bit 2 – UPEn: Parity Error
- Bit 1 – U2Xn: Double the USART Transmission Speed
- Bit 0 – MPCMn: Multi-Processor Communication Mode

UCSRnB: Mostly a Control Register

- RXCIE, TXCIE, UDRIE: Receive, Transmit, UDR interrupt enable
- RXEN, TXEN: Receive, Transmit enable
- UCSZn2: To decide number of “data” bits in the frame (see also UCSZn1 and UCSZn0)
- RXB8n and TXB8n: Used in 9 data bit frame setting. Not to be used in this course

UCSRnC: A Control Register

- UMSEL: Asynchronous or Synchronous
  - 0 for Async, 1 for Sync; always use 0 only in this course
- UPM1-0: Parity mode = 00 (disabled), 10 (even), 11(odd)
- USB: stop bit select: 0 (1 stop bit), 1 (2 stop bits)
- UCSZ2-0: Number of data bits in a frame
  - 000 (5-bits), 001 (6), 010 (7), 011 (8), 111 (9)
- UCPOl: Invert the polarity (invert logic low and high)
  - Use 0 in this course

Serial (ATmega128)

- Baud rate
  - 1 baud = 1 symbol per second
  - In our case, 8 data bits are book ended by start and stop bits

- Baud rate is different from data rate
  - Baud rate includes overhead of start/stop/parity bits
Initialization

- **UBRR0 does not contain the baud rate**
  - In order to keep sameness across microprocessors, AVR uses the following formula to calculate the value stored in the baud rate register

### Example UBRR Settings

<table>
<thead>
<tr>
<th>Baud Rate (b/s)</th>
<th>UBRR0</th>
<th>Error</th>
<th>UBRR0</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>2400</td>
<td>2416</td>
<td>-0.1%</td>
<td>832</td>
<td>0.0%</td>
</tr>
<tr>
<td>4800</td>
<td>207</td>
<td>0.2%</td>
<td>416</td>
<td>-0.1%</td>
</tr>
<tr>
<td>9600</td>
<td>103</td>
<td>0.2%</td>
<td>207</td>
<td>0.2%</td>
</tr>
<tr>
<td>14.4k</td>
<td>68</td>
<td>0.6%</td>
<td>138</td>
<td>-0.1%</td>
</tr>
<tr>
<td>19.2k</td>
<td>51</td>
<td>0.2%</td>
<td>103</td>
<td>0.2%</td>
</tr>
<tr>
<td>28.8k</td>
<td>34</td>
<td>-0.8%</td>
<td>68</td>
<td>0.6%</td>
</tr>
<tr>
<td>38.4k</td>
<td>25</td>
<td>0.2%</td>
<td>51</td>
<td>0.2%</td>
</tr>
<tr>
<td>57.6k</td>
<td>16</td>
<td>2.1%</td>
<td>34</td>
<td>-0.8%</td>
</tr>
</tbody>
</table>

### Transmitting

//Transmit a piece of data
void serial_putchar(char data) {
    /* Wait for empty transmit buffer by checking the UDRE bit */
    while ((UCSR0A & 0b10000000) == 0) {
    }

    /* Put data into transmit buffer; sends the data */
    UDR0 = data;
}

### Receiving

//Receive data
char serial_getchar() {
    /* Wait for the receive complete flag (RXC) */
    while ((UCSR0A & 0b10000000) == 0); // optional: receive interrupt enable bit

    /* Reads data from the receive buffer; clears the receive buffer */
    return UDR0;
}

### Warning!

- **UDR0 is actually two registers!**
  - One for reading (the receive buffer)
  - One for writing (the transmit buffer)

- Every read of the receive buffer could get a newly received character
Interrupts

- Vector names for interrupts
  - USART0_RX_vect (Receive complete)
  - USART0_TX_vect (Transmit complete)
  - USART0_UDRE_vect (Data register empty)

// The serial receive interrupt
ISR (USART0_RX_vect) {
  char received_byte = UDR0;
}

Lab 5

- Part I. Receive and Display Text
  - Check frame format and baud rate
  - Optional: Use interrupt
- Part II. Provide Character Echo
  - Send back received characters
- Part III. Push Button Response
  - Send back special messages when a push button is pressed
- Part IV. Bluetooth (57600 baud)
  - Perform USART communication on top of Bluetooth
  - Use a different baud rate

BAM

- The datasheet on your BAM specifies the following USART settings (BAM = Bluetooth Adapter Module)
- See iRobot BAM Datasheet (10542B.pdf), on class resource page

SPI (Serial Peripheral Interface)

High speed synchronous data transfer interface (system clock divided by 4, 16, 64, 128)

- Simple 3-wire interface: clock, and two one-directional wires between the communicating parties.

---

**SPI Status Reg**

Master Shift Register

- MOSI
- MISO
- MSB
- LSB

Slave Shift Register

- MOSI
- MISO
- MSB
- LSB

Clock Logic

- 8-bit shift register
- CLK
- SCK
- MOSI
- MISO
- SPI
- SCLK
- SS
- CS

SPI Control Reg

- SPI
- SCK
- MISO
- MOSI
- SS
- CS

SPI Status Reg
Bit 7 – SPIE: SPI Interrupt Enable
This bit causes the SPI interrupt to be executed if SPIF = 1

Bit 6 – SPE: SPI Enable
When the SPE bit is written to one, the SPI is enabled.

Bit 5 – DORD: Data Order
DORD=1 (0), LSB (MSB) of the data word is transmitted first.

Bit 4 – MSTR: Master/Slave Select
1: Master SPI mode, and 0: Slave SPI mode

Bit 3 – CPOL: Clock Polarity

Bit 2 – CPHA: Clock Phase

Bits 1, 0 – SPR1, SPR0: SPI Clock Rate Select 1 and 0
0 0 f/4
0 1 f/16
1 0 f/64
1 1 f/128

Bit 7 – SPIF: SPI Interrupt Flag
When a serial transfer is complete, the SPIF flag is set. SPIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, the SPIF bit is cleared by first reading the SPI Status Register with SPIF set, then accessing the SPI Data Register (SPDR).

Bit 0 – SPI2X: Double SPI Speed Bit

The SPI Data Register is a Read/Write Register used for data transfer between the register file and the SPI Shift Register.

Writing to the register initiates data transmission.
Reading the register causes the Shift Register Receive buffer to be read.

// sends a char to the LCD
void
lcd_putchar(char ch) // the character
{
    spi_send(0x01); // command prefix
    spi_send(ch); // send the char
    lcd_strobe(); // toggle the enable bit
    _delay_us(160);
}

// initialize the SPI interface
void
spi_init(void)
{
    DDRF |= 0x08; // port F bit 3 is enable for LCD
    PORTB |= 0x00; // port B initialization for SPI
    DORB |= 0x07; // Turn on SS, MOSI, SCLK
    // Master mode, Clock=clk/2, Cycle half phase, Low polarity,
    // MSB first
    SPCR = 0x50;
    SPSR = 0x01;
}
SPI in Interrupt mode

/* SPI Serial Transfer Complete */
#define SPI_STC_vect    _VECTOR(17)
#define SIG_SPI         _VECTOR(17)

ISR (SIG_SPI) {  }