Serial Communications

Steve Gunn
Hardware Communication

- Sending data over a communication channel or bus
  - **Serial** One bit is sent at a time
  - **Parallel** Multiple \((n)\) bits are sent as a whole
- Synchronous (with clock) and Asynchronous (without)
- Historically serial only used for long distances.
- Parallel potentially \(n\) times faster.
- Parallel has no need for SerDes (Serialiser/De-serialiser).
- Serial now prevalent at shorter distances. Why?
Parallel Disadvantages

**Pin Count**

- Integrated circuits cost increases with pin count.

**Interconnect Density**

- Cable/board cost increases with conductor count.

**Clock Skew (Timing Skew)**

- Signals take time to move down wires. Arrival times may be different.

**Crosstalk**

- Signals may interact due to undesired capacitive or inductive coupling
Serpentine Routing
Examples of Serial Communication

• Morse Code
• RS-232 (UART)
• I²C
• SPI
• USB
• SATA (c.f. PATA)

• Ethernet
• JTAG
• PCI Express (c.f. PCI)
• FireWire
• Thunderbolt
• MIDI
Resources Used

- AVR Datasheet
  - Section 17: SPI
  - Section 18: U(S)ART
  - Section 20: TWI (I²C)
- AVR Libc
  ```c
  #include <avr/io.h>
  #include <util/setbaud.h>
  #include <util/twi.h>
  ```
- Il Matto Quick Reference Communication
# Pin Functions

<table>
<thead>
<tr>
<th>Port</th>
<th>Pin</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>PA7</td>
<td>ADC7</td>
<td>ADC input channel 7</td>
</tr>
<tr>
<td>A</td>
<td>PA6</td>
<td>ADCn</td>
<td>ADC input channel n</td>
</tr>
<tr>
<td>A</td>
<td>PA0</td>
<td>ADC0</td>
<td>ADC input channel 0</td>
</tr>
<tr>
<td></td>
<td>PB7</td>
<td>SCK</td>
<td>SPI Bus Master clock input</td>
</tr>
<tr>
<td></td>
<td>PB6</td>
<td>MISO</td>
<td>SPI Bus Master Input/Slave Output</td>
</tr>
<tr>
<td></td>
<td>PB5</td>
<td>MOSI</td>
<td>SPI Bus Master Output/Slave Input</td>
</tr>
<tr>
<td></td>
<td>PB4</td>
<td>SS</td>
<td>SPI Slave Select input</td>
</tr>
<tr>
<td></td>
<td>PB0</td>
<td>XCK0</td>
<td>USART0 External Clock Input/Output</td>
</tr>
<tr>
<td></td>
<td>PB1</td>
<td>INT2</td>
<td>External Interrupt 2 Input</td>
</tr>
<tr>
<td></td>
<td>PB2</td>
<td>OC0B</td>
<td>Timer/Counter 0 Output Compare Match B Output</td>
</tr>
<tr>
<td></td>
<td>PB3</td>
<td>AIN1</td>
<td>Analog Comparator Negative Input</td>
</tr>
<tr>
<td></td>
<td>PB2</td>
<td>AIN0</td>
<td>Analog Comparator Positive Input</td>
</tr>
<tr>
<td></td>
<td>PB1</td>
<td>T1</td>
<td>Timer/Counter 1 External Counter Input</td>
</tr>
<tr>
<td></td>
<td>PB0</td>
<td>T0</td>
<td>Timer/Counter 0 External Counter Input</td>
</tr>
<tr>
<td></td>
<td>PC7</td>
<td>TOSC2</td>
<td>Timer Oscillator pin 2</td>
</tr>
<tr>
<td></td>
<td>PC6</td>
<td>TOSC1</td>
<td>Timer Oscillator pin 1</td>
</tr>
<tr>
<td></td>
<td>PC5</td>
<td>TDI</td>
<td>JTAG Test Data Input</td>
</tr>
<tr>
<td></td>
<td>PC4</td>
<td>TDO</td>
<td>JTAG Test Data Output</td>
</tr>
<tr>
<td></td>
<td>PC3</td>
<td>TMS</td>
<td>JTAG Test Mode Select</td>
</tr>
<tr>
<td></td>
<td>PC2</td>
<td>TCK</td>
<td>JTAG Test Clock</td>
</tr>
<tr>
<td></td>
<td>PC1</td>
<td>SDA</td>
<td>2-wire Serial Bus Data Input/Output Line</td>
</tr>
<tr>
<td></td>
<td>PC0</td>
<td>SCL</td>
<td>2-wire Serial Bus Clock Line</td>
</tr>
<tr>
<td></td>
<td>PD7</td>
<td>OC2A</td>
<td>Timer/Counter2 Output Compare Match A Output</td>
</tr>
<tr>
<td></td>
<td>PD6</td>
<td>ICP1</td>
<td>Timer/Counter1 Input Capture Trigger</td>
</tr>
<tr>
<td></td>
<td>PD5</td>
<td>OC2B</td>
<td>Timer/Counter2 Output Compare Match B Output</td>
</tr>
<tr>
<td></td>
<td>PD4</td>
<td>OC1A</td>
<td>Timer/Counter1 Output Compare Match A Output</td>
</tr>
<tr>
<td></td>
<td>PD3</td>
<td>OC1B</td>
<td>Timer/Counter1 Output Compare Match B Output</td>
</tr>
<tr>
<td></td>
<td>PD2</td>
<td>XCK1</td>
<td>USART1 External Clock Input/Output</td>
</tr>
<tr>
<td></td>
<td>PD1</td>
<td>TXD1</td>
<td>USART1 Transmit Pin</td>
</tr>
<tr>
<td></td>
<td>PD0</td>
<td>RXD1</td>
<td>USART1 Receive Pin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INT1</td>
<td>External Interrupt 1 Input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TXD0</td>
<td>USART0 Transmit Pin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RXD0</td>
<td>USART0 Receive Pin</td>
</tr>
</tbody>
</table>

**Notes:** Each pin also has a pin change interrupt. Each pin can also be configured as GPIO.

- 32 I/O pins
- Grouped as 4 ports
- Each pin has a special function
- Or can be used as GPIO
- Sec. 13 of AVR Datasheet
UART
Universal Asynchronous Receiver/Transmitter
UART Overview

- End to end communication between two devices.
- Speeds up to \(~2\text{M baud}^1\) possible
- High speeds need accurate timing and good signal integrity.
- Two wires (TX and RX) for sending and receiving.
- Additional line (CLK) for USART
- Effectively RS-232 with logic voltage levels
- Communication modes: Simplex, Duplex, Half-duplex

---

1. The baud rate is defined to be the transfer rate in bit per second (bps)
UART connection

- Asynchronous

- Synchronous
UART Frame

Figure 18-4. Frame Formats

<table>
<thead>
<tr>
<th>St</th>
<th>Start bit, always low.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n)</td>
<td>Data bits (0 to 8).</td>
</tr>
<tr>
<td>P</td>
<td>Parity bit. Can be odd or even.</td>
</tr>
<tr>
<td>Sp</td>
<td>Stop bit, always high.</td>
</tr>
<tr>
<td>IDLE</td>
<td>No transfers on the communication line (RxDn or TxDn). An IDLE line must be high.</td>
</tr>
</tbody>
</table>
Figure 18-1. USART Block Diagram

UART

Clock Generator

UBRR[H:L]

BAUD RATE GENERATOR

SYNC LOGIC

PIN CONTROL

XCK

Transmitter

UDR (Transmit)

TRANSMIT SHIFT REGISTER

PARITY GENERATOR

TX CONTROL

TxD

Receiver

RECEIVE SHIFT REGISTER

CLOCK RECOVERY

RX CONTROL

RxD

PARITY CHECKER

PARITY RECOVERY

PIN CONTROL

UCSRA

UCSRB

UCSRC

DATA RECOVERY
UART Baud rate

Figure 18-2. Clock Generation Logic, Block Diagram
# UART Baud rate configuration

## Table 18-1. Equations for Calculating Baud Rate Register Setting

<table>
<thead>
<tr>
<th>Operating Mode</th>
<th>Equation for Calculating Baud Rate(^{11})</th>
<th>Equation for Calculating UBRR Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous Normal mode (U2Xn = 0)</td>
<td>( BAUD = \frac{f_{osc}}{16(UBRRn + 1)} )</td>
<td>( UBRRn = \frac{f_{osc}}{16BAUD} - 1 )</td>
</tr>
<tr>
<td>Asynchronous Double Speed mode (U2Xn = 1)</td>
<td>( BAUD = \frac{f_{osc}}{8(UBRRn + 1)} )</td>
<td>( UBRRn = \frac{f_{osc}}{8BAUD} - 1 )</td>
</tr>
<tr>
<td>Synchronous Master mode</td>
<td>( BAUD = \frac{f_{osc}}{2(UBRRn + 1)} )</td>
<td>( UBRRn = \frac{f_{osc}}{2BAUD} - 1 )</td>
</tr>
</tbody>
</table>

\( f_{osc} \): System Oscillator clock frequency  
\( UBRRn \): Contents of the UBRRHn and UBRRLn Registers, (0-4095)
AVR UART Communication

\[
\text{UART} \quad (n \in \{0, 1\}, \quad \text{BAUD} \in \{1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200\})
\]

```c
#define BAUD 57600
#include <util/setbaud.h>
void init_uart(void) { /* 8N1 */
    UBRRnH = UBRRH_VALUE; UBRRnL = UBRL_VALUE;
    UCSRnA = USE_2X << U2Xn;
    UCSRnB = _BV(RXENn) | _BV(TXENn);
    UCSRnC = _BV(UCSZn0) | _BV(UCSZn1);
}
void tx(uint8_t b) { while(!(UCSRnA & _BV(UDREn))); UDRn = b; }
uint8_t rx(void) { while(!(UCSRnA & _BV(RXCn))); return UDRn; }
```

- Other baud rates are possible (common ones listed).
- `#define BAUD` must precede `#include <setbaud.h>`
AVR C Example from Lab X2

* Notes: F_CPU must be defined to match the clock frequency */
#include <inttypes.h>
#include <avr/io.h>

void init_uart0(void)
{
    /* Configure 9600 baud, 8-bit, no parity and one stop bit */
    const int baud_rate = 9600;
    UBRROH = (F_CPU/(baud_rate*16L)-1) >> 8;
    UBRROL = (F_CPU/(baud_rate*16L)-1);
    UCSROB = _BV(RXEN0) | _BV(TXEN0);
    UCSROC = _BV(UCSZ00) | _BV(UCSZ01);
}

cchar get_ch(void)
{
    while(!(UCSR0A & _BV(RIC0)));
    return UDR0;
}

void put_ch(char ch)
{
    while (!(UCSR0A & _BV(UDRE0)));
    UDR0 = ch;
}
```c
void put_str(char *str)
{
    int i;
    for(i=0; str[i]; i++) put_ch(str[i]);
}

int main(void)
{
    char ch;
    init_uart0();
    put_str("Hello from Il Matto\n\r");
    /* forever loop */
    for (;;)
    {
        /* get character from UART */
        ch = get_ch();
        /* send message back to the host terminal */
        put_str("You sent the character ");
        put_ch(ch);
        put_str(" \n\r");
    }
}
```
Devices that talk UART

- C232HM
- XBee wireless
- RS232 serial port (level shifter required)
SPI
Serial Peripheral Interface (Bus)
SPI Overview

- End to end communication between two devices.
- Multiple devices allowed on the bus.
- One device operates as the Master.
- Speeds up to 100MHz possible
- High speeds need good signal integrity.
- Four wires (SCLK, MOSI, MISO, /SS) sending and receiving.
- Communication mode: Duplex
SPI connection

MOSI  Master Out, Slave In
MISO  Master In, Slave Out
/SS   Slave Select (Active Low)
SCLK  Serial Clock (Output from Master)
SPI 8-bit transfer

- Different bit lengths possible. Hardware dependent.
- AVR has internal 8-bit shift register
### Clock Phase and Polarity

<table>
<thead>
<tr>
<th>Mode</th>
<th>CPOL</th>
<th>CPHA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Slave Select

• In general if only a single slave device is used, you may connect the /SS pin on the slave to ground.

• However, this will fail with some devices:
  – some devices use the falling edge (high→low transition) of the chip select to initiate an action.
  – e.g. some ADCs, which starts conversion on high→low transition.

• There are two possible ways to attach multiple slave devices to the bus: Individual selects, Daisy chained.
Multiple SPI devices on the bus

Individual Selects

- Normal configuration.

Daisy Chained

- Not supported by all devices
AVR SPI Communication

**SPI** \( F_{SCK} = F_{CPU}/d, \ d \in \{2,4,8,16,32,64,128\} \)

```c
void init_spi_master(void) { /* out: MOSI, SCK, /SS, in: MISO */
    DDRB = _BV(PB4) | _BV(PB5) | _BV(PB7);
    SPCR = _BV(SPE) | _BV(MSTR) | _BV(SPI2X); /* F_SCK = F_CPU/2 */ }
void tx(uint8_t b) { SPDR = b; while(!(SPSR & _BV(SPIF))); }
void init_spi_slave(void) { /* out: MISO, in: MOSI, SCK, /SS */
    DDRB = _BV(PB6); SPCR = _BV(SPE); /* Enable SPI */ }
uint8_t rx(void) { while(!(SPSR & _BV(SPIF))); return SPDR; }
```

- Other pins can be used as CS lines for connecting multiple devices to the bus.
Devices that talk SPI

• Generally, used for sensors and lower speed memories:
  – SD cards (slower than 4-bit transfer mode)
  – ADC/DAC
  – Digital Potentiometer
  – Accelerometers
  – Gyros
  – Magnetometers
  – EEPROM
  – ...

I\textsuperscript{2}C Overview

- End to end communication between two devices.
- Multiple devices allowed on the bus (limited by the address space and by the bus capacitance)
- Distances limited to a few metres.
- Multiple devices can operate as the Master.
- Typical speed: 100kHz or 400kHz (up to 5MHz possible)
- Two wires (SCL, SDA) for communication.
- Communication mode: Simplex
I²C connection

128 Unique addresses
SCL Clock Line (Open Drain)
SDA Data Line (Open drain)
I²C Operation

- Four potential modes of operation for a given bus device:
  
  - \textit{master transmit} – master node sending to a slave
  
  - \textit{master receive} – master node receiving from a slave
  
  - \textit{slave transmit} – slave node is sending to the master
  
  - \textit{slave receive} – slave node is receiving from the master

- Most devices only use a single role and its two modes
I²C Operation: Master to Slave R/W

- Master is initially in master transmit mode:
  - send a start bit,
  - send the 7-bit slave address it wishes to talk to,
  - send a direction bit (write(0), read(1))
  - wait for/send an acknowledge bit
  - send/receive data byte (8 bits)
  - wait for/send an acknowledge bit
  - send the stop bit
Transmission of one data byte

Transmission of multiple data bytes

• Data Transfer from master to slave

![Diagram showing data transfer from master to slave]

• Data Transfer from slave to master

![Diagram showing data transfer from slave to master]

AVR I²C Communication

I²C (F_SCL = F_CPU/d,  \( d = 2(8+b4^p) \),  \( b \in \{0,1,...,255\} \),  \( p \in \{0,1,2,3\} \))

```c
#include <util/twi.h>
void start(void) { TWCR = _BV(TWINT) | _BV(TWSTA) | _BV(TWEN);  
    while(!(TWCR & _BV(TWINT))); }
void stop(void) { TWCR = _BV(TWINT) | _BV(TWSTO) | _BV(TWEN); }
void tx(uint8_t b) { TWDR = b; TWCR = _BV(TWINT) | _BV(TWEN);  
    while(!(TWCR & _BV(TWINT))); }
uint8_t rx(void) { TWCR = _BV(TWINT) | _BV(TWEN);  
    while(!(TWCR & _BV(TWINT))); return TWDR; }
TWBR = 0x34; TWSR = 0x00; /* F_SCL = F_CPU/120 */
start(); assert(TW_STATUS == TW_START);
 tx(SLA | TW_READ); assert(TW_STATUS == TW_MR_SLA_ACK);
 uint8_t b = rx(); assert(TW_STATUS == TW_MR_DATA_NACK);
 start(); assert(TW_STATUS == TW_REP_START);
 tx(SLA | TW_WRITE); assert(TW_STATUS == TW_MT_SLA_ACK);
 tx(b); assert(TW_STATUS == TW_MT_DATA_ACK);
 stop();
```
Devices that talk I$^2$C

- Generally, used for sensors and lower speed memories:
  - ADC/DAC
  - Digital Potentiometer
  - Accelerometers
  - Gyros
  - Magnetometer
  - EEPROM
  - ...
JTAG
Joint Test Action Group
JTAG Overview

- Similar to SPI with an additional mode pin (TMS)
- Multiple devices allowed by daisy chaining.
- Speeds up to 100MHz possible
- Four wires (TCK, TDI, TDO, TMS)
- Communication mode: Duplex
- Use #1: device programming of firmware
- Use #2: boundary scan
- Use #3: hardware debugging (breakpoints and single step)
JTAG Chain
JTAG Boundary Scan
Devices that talk JTAG

• Generally, more complex ICs with larger pin counts and internal processing.
  – Microcontrollers
  – CPLDs
  – FPGAs
  – Microprocessors

• Not usually found on simpler devices such as sensors.