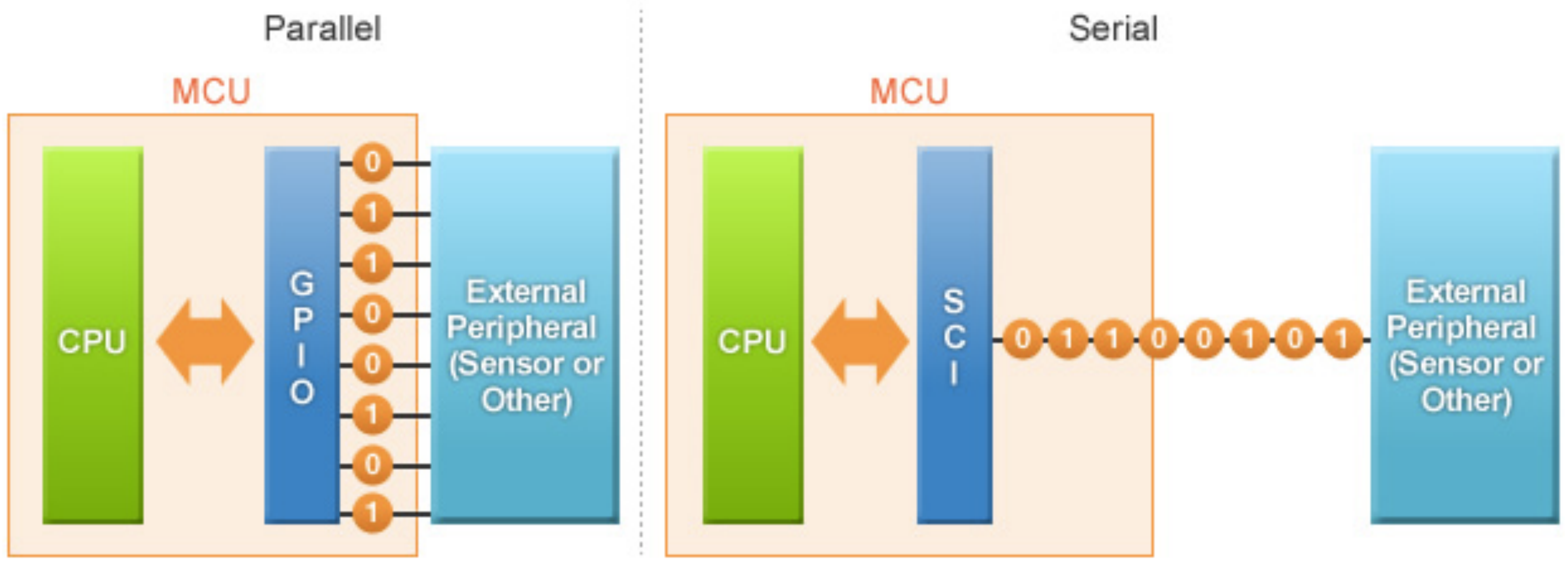
Serial I/O in Embedded Systems

Baris Aksanli
Department of Electrical and Computer Engineering
San Diego State University

Outline

- Serial data communication
- Data format/example
- Serial signals/ports/control status
- Serial interfaces
- Example: Using serial interface in AVR
- Serial peripheral interface (SPI)
- I^{2}C (Inter-Integrated Circuit) interface

Serial Communication



Parallel
MCU
CPU
GPIO
0 1 1 0 0 1 0 1
External Peripheral (Sensor or Other)
Serial
MCU
CPU
SCI
0 1 1 0 0 1 0 1
External Peripheral (Sensor or Other)

- Multiple data bits are transmitted/received at a time
- Faster
- Higher number of cables required

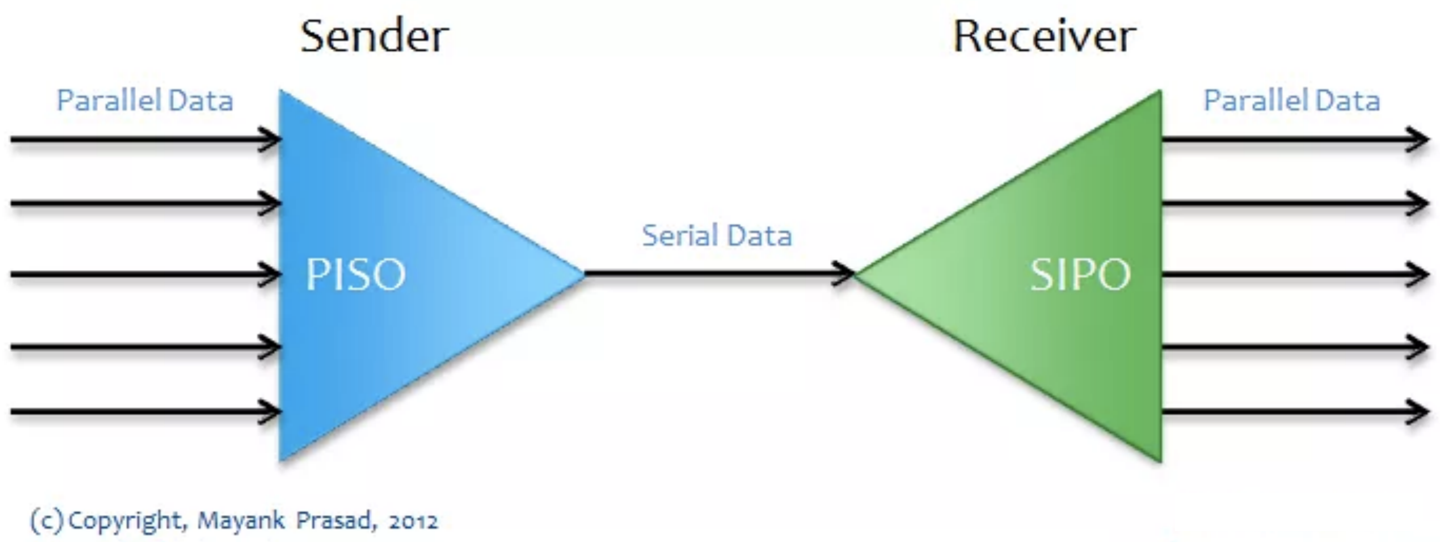
- One data bit is transmitted/ received at a time
- Slower
- Less number of cables required

Parallel vs. Serial Communication

- Limiting factors for parallel communication
  - Complexity
  - Cannot achieve ideal speed
  - More cables required
- Advantages of serial over parallel communication
  - Fewer interconnection cables
  - Simpler design

How is data sent serially?

- The sender converts parallel data to serial data (PISO) (Parallel In Serial Out), MSB (Most Significant Bit) first or LSB (Least Significant Bit) first as according to the protocol
- When this data is received by another microcontroller in its receiver buffer, the receiver buffer converts it back into parallel data (SIPO) (Serial In Parallel Out) for further processing



Serial Data Communication

- Asynchronous :
  - Data bits are not “synchronized” with a clock line
  - UART: Universal Async Receiver Transmitter
  - Data signal only
  - Independent clock in receiver
- Synchronous (clocked):
  - Separate data and clock signals
  - Also I^{2}C™, SPI™, Microwire™, CAN bus, …
  - Other non-TM names: SM bus, TWI, …
- USART: Sync AND Async Receiver Transmitter

Serial Communication Definitions

- **MSB/LSB:** Most Significant Bit (or Least Significant Bit)
- **Simplex Communication:** Data can only be transferred from transmitter to receiver and not vice versa.
- **Half Duplex Communication:** Data transmission can occur in only one direction at a time, i.e. either from master to slave, or slave to master, but not both.
- **Full Duplex Communication:** Data can be transmitted from the master to the slave, and from slave to the master as the same time.
- **Baud Rate:** It is the unit of symbol rate, also known as baud or modulation rate. For our case, baud refers to bits per second.

Transmission Types

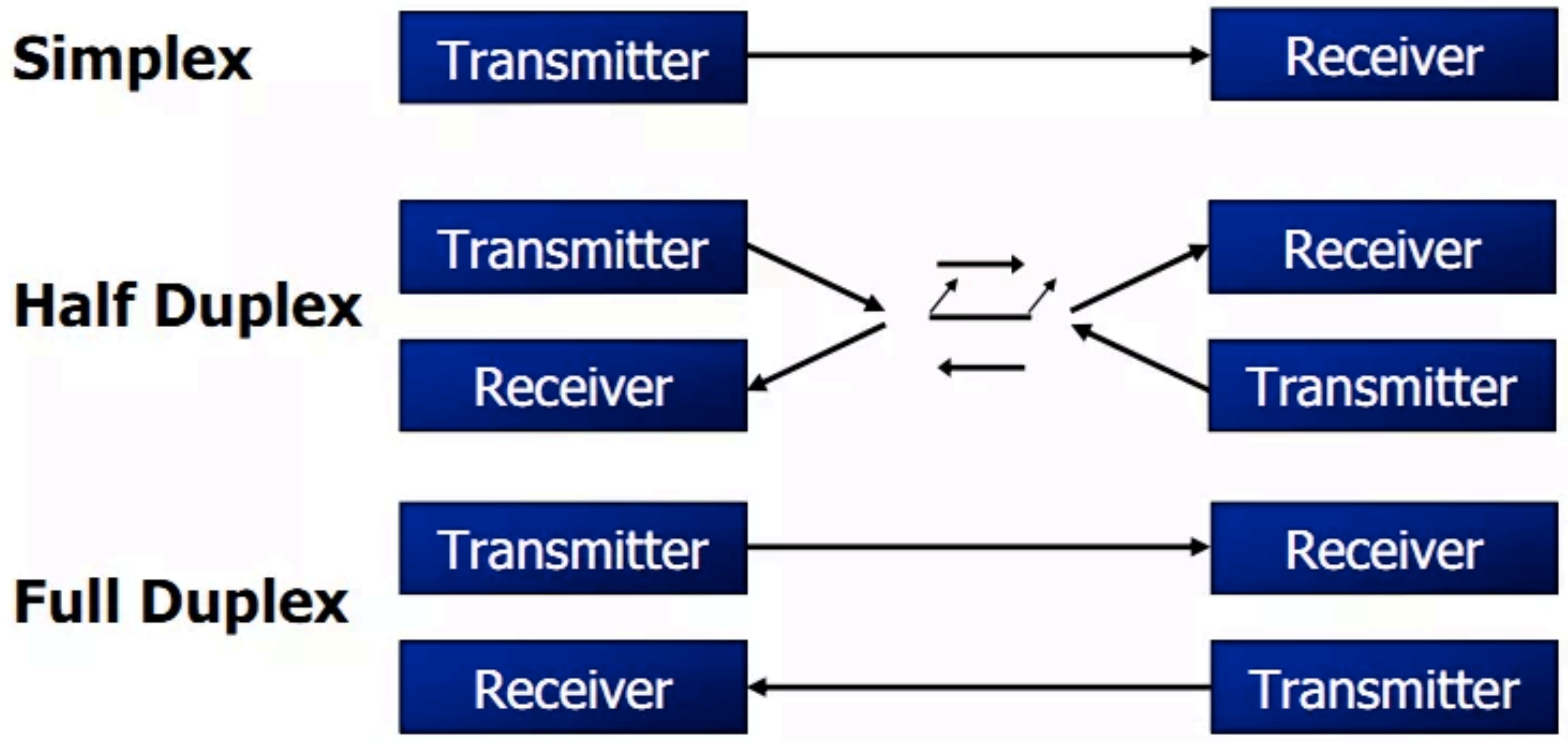
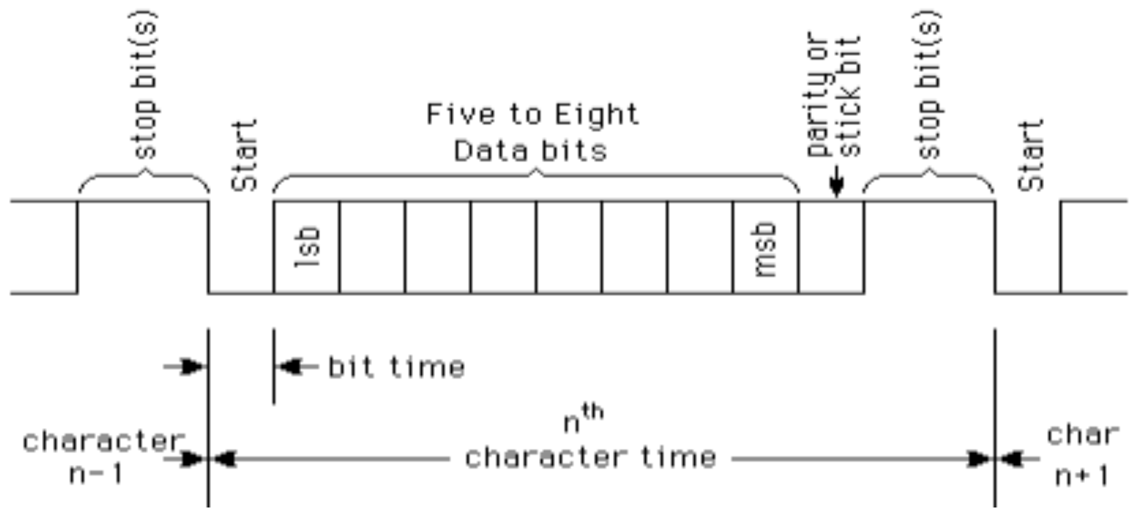


Image Courtesy
The 8051 Microcontroller and Embedded Systems by Mazidi and Mazidi

Important Notes

- Two serially communicating microcontrollers should have the same baud rate
- This is because when you set a baud rate, you direct the microcontroller to transmit/receive the data at that particular rate.
- So if you set different baud rates, then the receiver might miss out the bits the transmitter is sending
- Different baud rates are available for use. The most common ones are 2400, 4800, 9600, 19200, 38400 etc.
  - You cannot choose any arbitrary baud rate
  - The unit of baud rate is bps (bits per second)

Async Serial Data Format

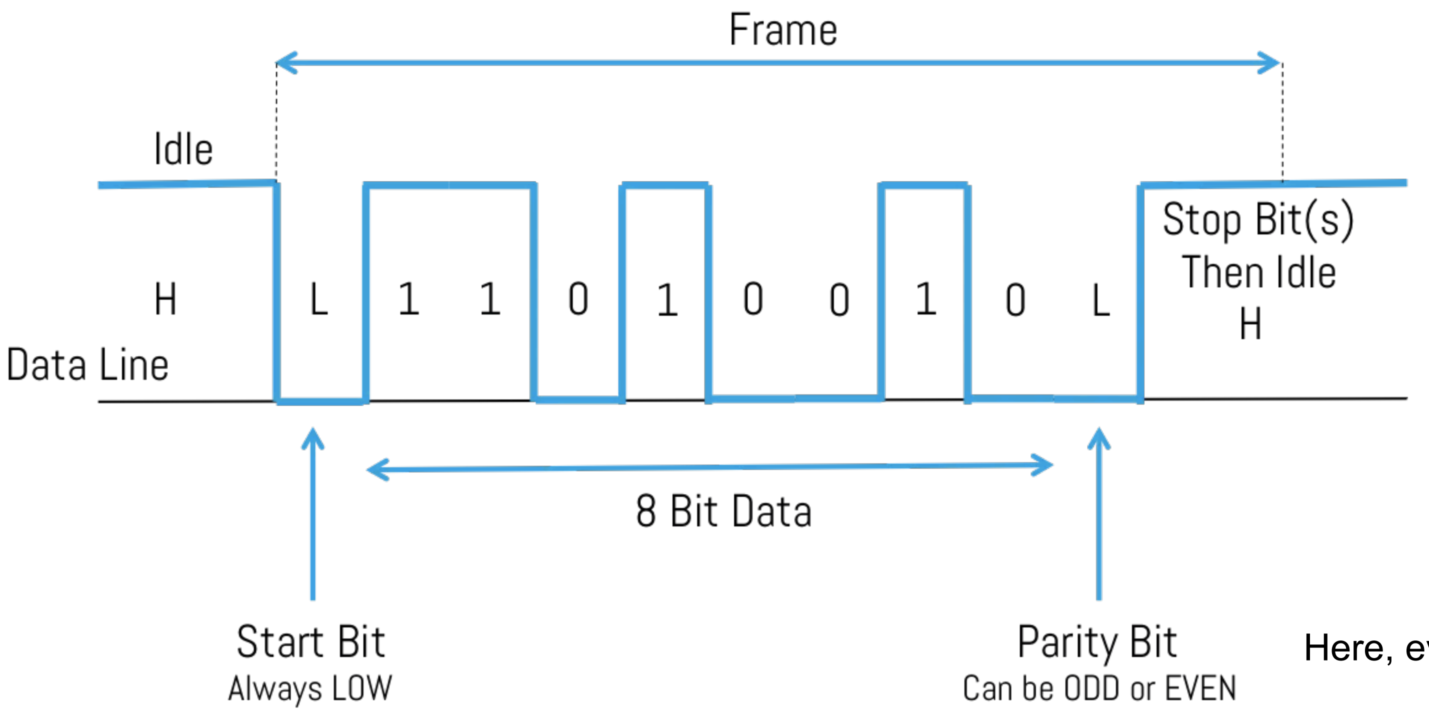


- Every frame will have at least
  - 1 start bit (always logic 0)
  - (5, 6, 7, 8, or 9) data bits
  - 1 or 2 stop bits (always logic 1)
- Parity bit is optional

Serial Data Example

Serial data 4BH: ASCII code for upper case 'K'

Binary 0100 1011 - LSB is sent first



Here, even parity

Common Async Formats

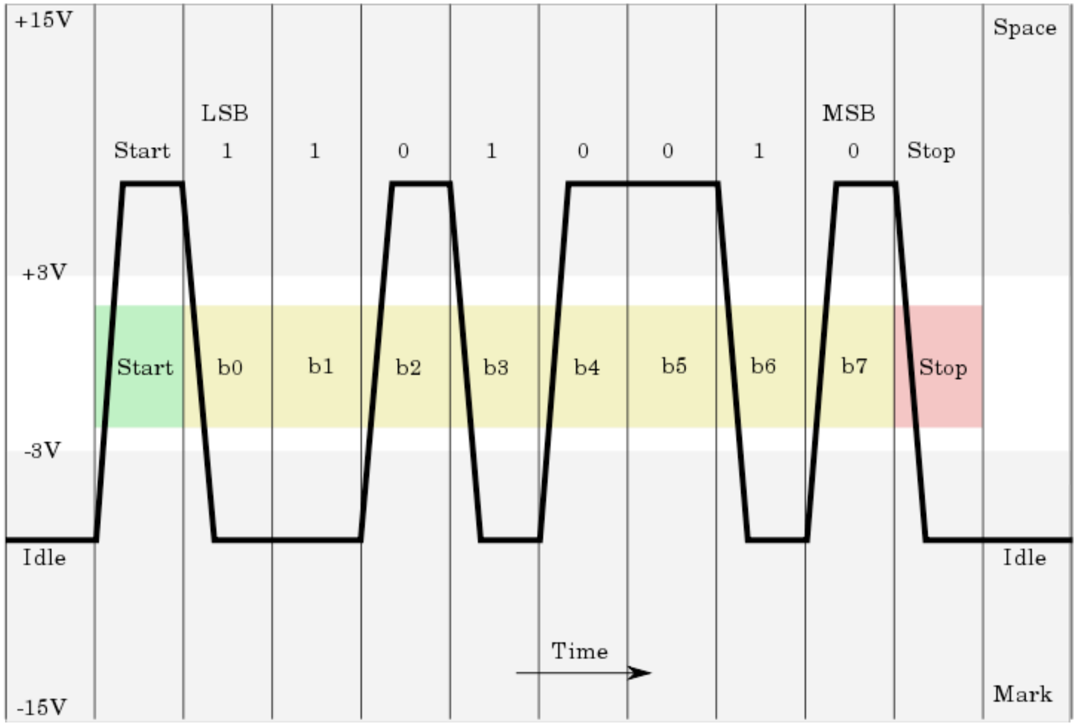
- Most common format for embedded:
  - 1 Start bit, 8 Data Bits, No Parity Bit, 1 Stop bit
  - = 10 bits per character transmitted
  - Abbreviated as 8,n,1 from old PC notation
- Data rates usually multiples of 300:
  - 300, 1200, 9600, 19200, 38400, 119200…
  - Three wires minimum: Tx, Rx, Gnd
  - Optional handshake signal wires: DTR, RTS…

Serial Signal Levels

- EIA Standard RS-232:
  - Approximately +/-3 to +/-12Volts
  - -12V = logic 1 = "mark" and
  - +12V = logic 0 = "space"
- EIA Standard RS-422, -423, -485:
  - Similar logical data formats
  - Different electrical signaling
- 3V CMOS levels – common for embedded
  - 3V = logic 1 = "mark" and 0V = logic 0 = "space"
- Most CMOS uC power supply = Vdd <= 3.3V

Serial Interfaces

- The old but persistent RS-232 standard supports asynchronous serial connections (no common clock)



Many but not all uses of RS-232 are being replaced by USB, which is electrically simpler but with a more complex protocol

RS-232 data is INVERTED
Ex: Uppercase ASCII "K" character (0x4b) with 1 start bit, 8 data bits, 1 stop bit. *Image license: Creative Commons*Image license: Creative Commons *ShareAlike 1.0* License

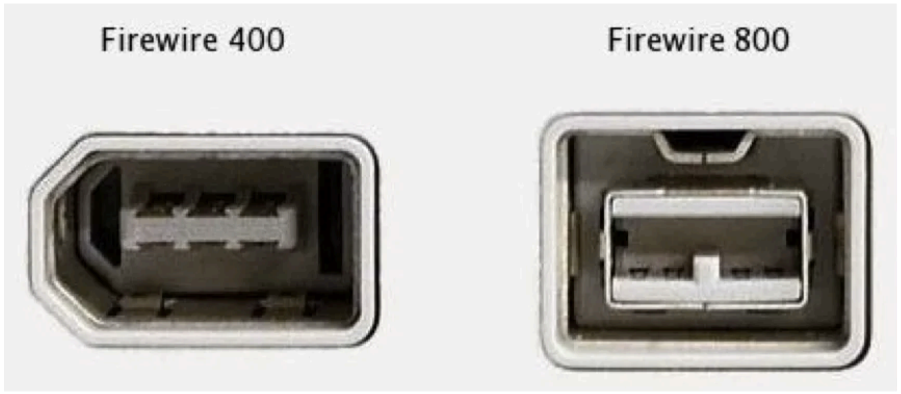
Serial Communication Protocols

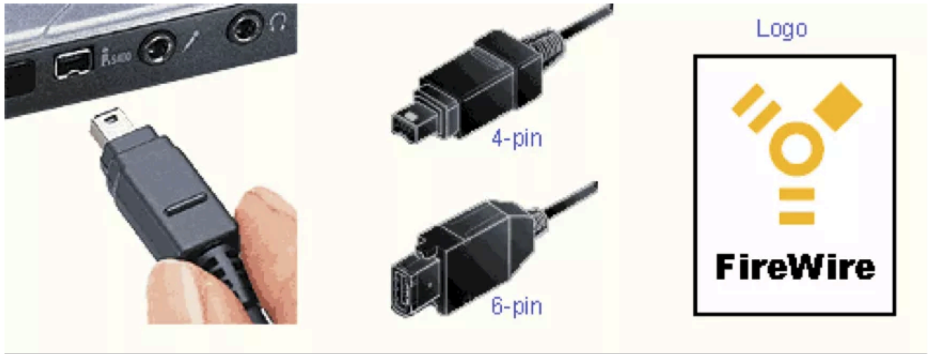
- **SPI – Serial Peripheral Interface**
  - It is a three-wire based communication system
  - One wire each for Master to slave and Vice-versa, and one for clock pulses
  - There is an additional SS (Slave Select) line, which is mostly used when we want to send/receive data between multiple ICs
- **I2C – Inter-Integrated Circuit**
  - This is an advanced form of USART
  - The transmission speeds can be as high as 400KHz
  - The I2C bus has two wires – one for clock, and the other is the data line, which is bi-directional – this being the reason it is also sometimes (not always – there are a few conditions) called Two Wire Interface (TWI)
  - It is new technology invented by Philips

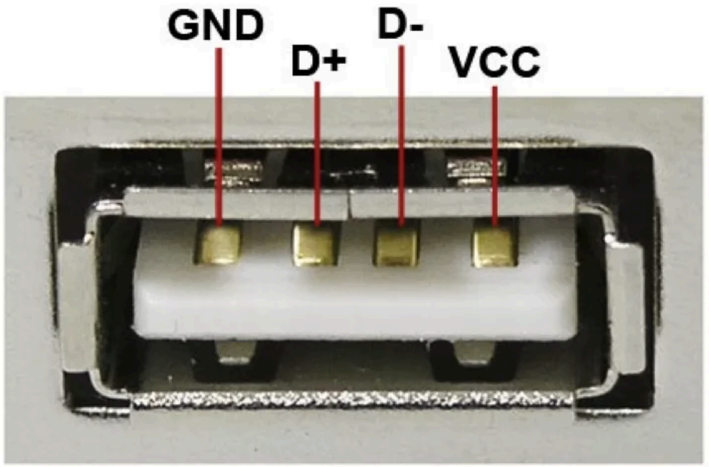
Serial Communication Protocols

- **FireWire**
  - Developed by Apple, they are high-speed buses capable of audio/video transmission
  - The bus contains a number of wires depending upon the port, which can be either a 4-pin one, or a 6-pin one, or an 8-pin one
- **Ethernet**
  - Used mostly in LAN connections, the bus consists of 8 lines, or 4 Tx/Rx pairs
- **Universal serial bus (USB)**
  - This is the most popular of all. Is used for virtually all type of connections. The bus has 4 lines: VCC, Ground, Data+, and Data-

Serial Communication Protocols





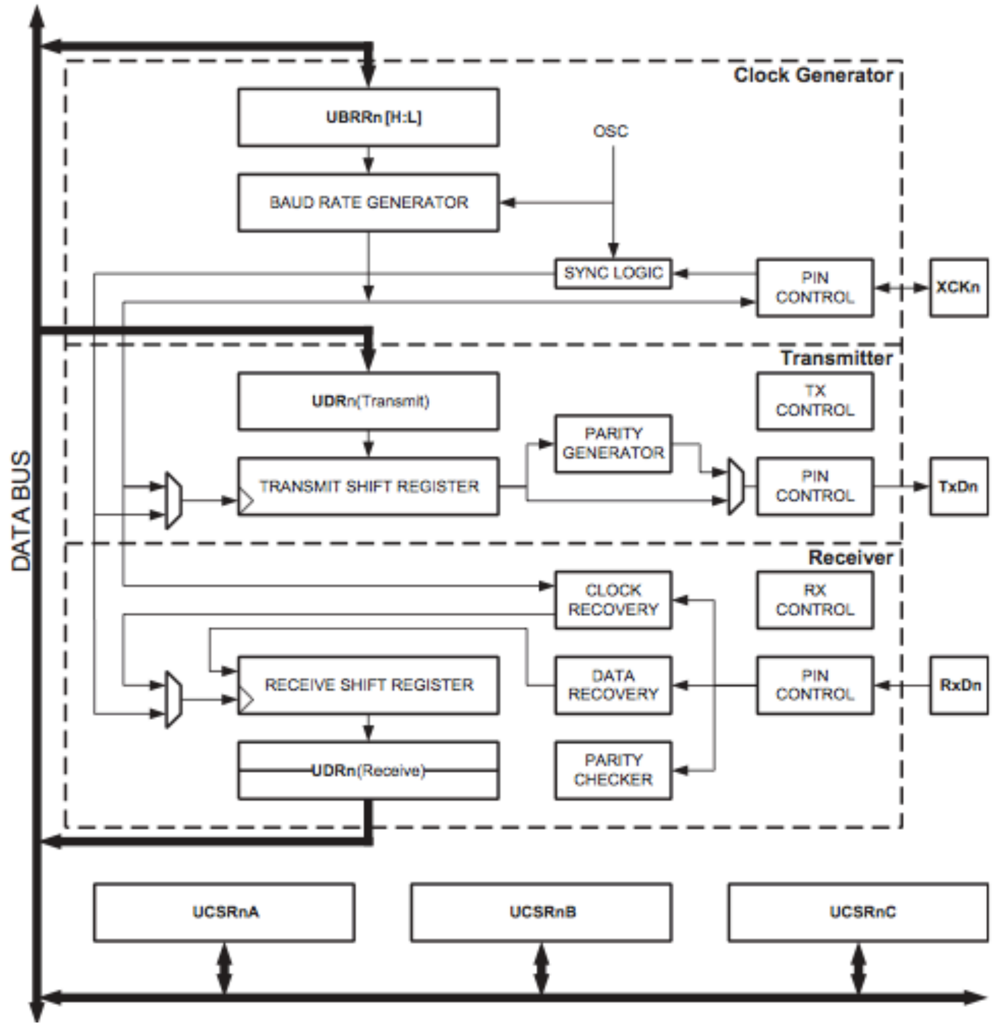


USB pins

UART: Universal Asynchronous Receiver-Transmitter

- Convert serial data to parallel data, and vice versa
- Uses shift registers to load store data
- Can raise interrupt when data is ready
- Commonly used with RS-232 interface

USART on AVR →



AVR USART Features

- Full duplex operation (Independent Serial Receive and Transmit Registers)
- Asynchronous or synchronous operation
- Master or slave clocked synchronous operation
- High resolution baud rate generator
- Supports serial frames with 5, 6, 7, 8, or 9 data bits and 1 or 2 stop bits

How to set up USART on AVR?

- AVR USART is fully compatible with the AVR UART in terms of register bit locations, baud rate generation, transmitter/receiver operations and buffer functionality
- In UART, there is no master or slave since master is defined by the microcontroller, which is responsible for clock pulse generation. Hence Master and Slave terms occur only in the case of USART.

1. The first step is to set the baud rate in both, the master and the slave. The baud rate has to be the same for both – master and slave.
2. Set the number of data bits, which needs to be sent.
3. Then enable the transmitter/receiver according to the desired usage.
4. Get the buffer ready! In case of transmission (from AVR to some other device), load it up with the data to be sent, whereas in case of reception, save the previous data so that the new received data can be overwritten onto it.

AVR USART Pin Configuration

- RxD: USART Receiver Pin
- TxD: USART Transmit Pin
- XCK: USART Clock Pin

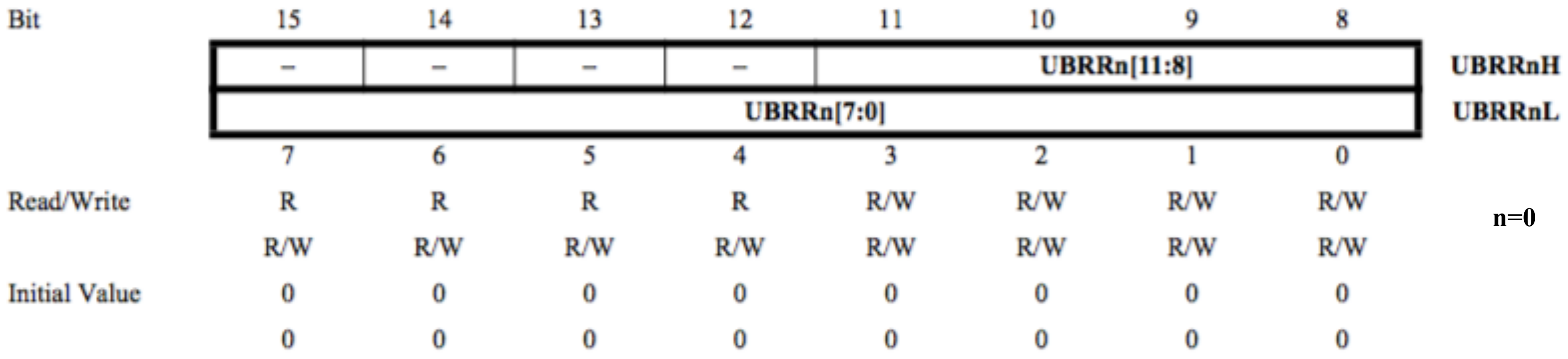
- See where they are located on your device
  - Listed in the manual

Modes of Operation

- **Asynchronous Normal Mode:** The data is transmitted/ received asynchronously (no need for the XCK pin).
  - The data is transferred at the BAUD rate we set in the UBBR register. This is similar to the UART operation.
- **Asynchronous Double Speed Mode:** We set the baud rates and other initializations similar to Normal Mode.
  - The difference is that data is transferred at double the baud we set in the UBBR Register.
- **Synchronous Mode:** This is the USART operation of AVR.
  - When Synchronous Mode is used (UMSEL = 1 in UCSRC register), the XCK pin will be used as either clock input (Slave) or clock output (Master).

Baud Rate Generation

- The baud rate of UART/USART is set using the 16-bit wide UBRR0 register (see below)
- The 16-bit UBRR register is comprised of two 8-bit registers – UBRR0H (high) and UBRR0L (low)
- Since there can be only specific baud rate values, there can be specific values for UBRR0, which when converted to binary will not exceed 12 bits. Hence there are only 12 bits reserved for UBRR0[11:0]

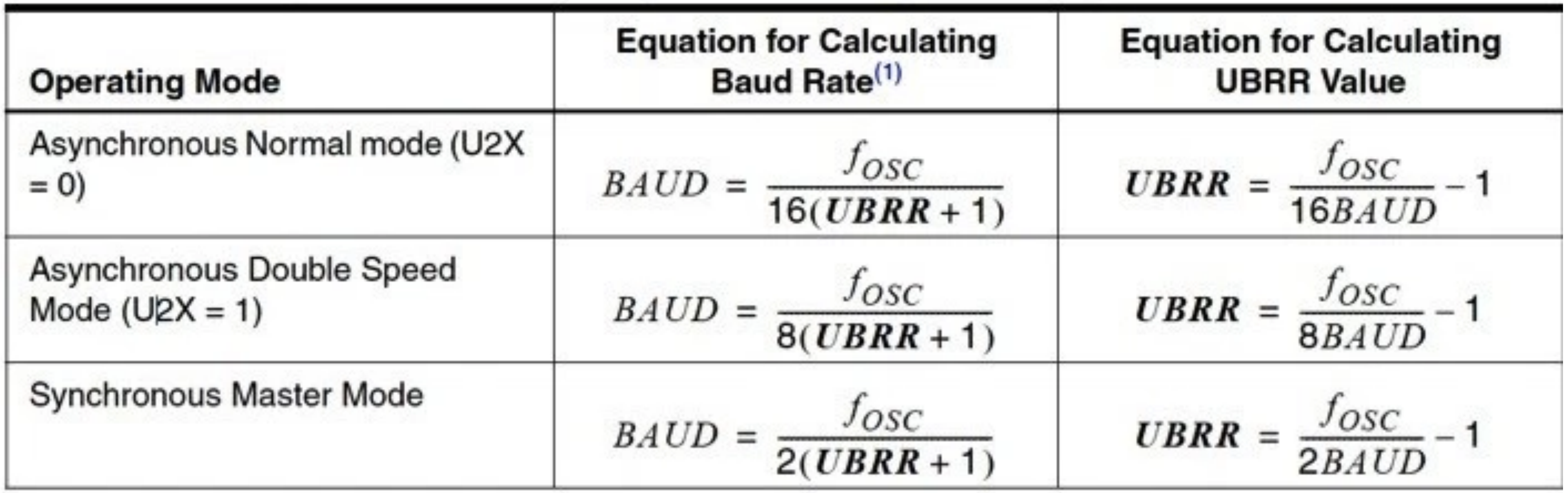


Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRRn[11:8]				UBRRnH
	UBRRn[7:0]								UBRRnL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	n=0
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

USART Baud Rate Register (UBRR0)

- UBRR and the down-counter connected to it functions as a programmable pre-scaler or baud rate generator
- The down-counter, running at system clock (FOSC), is loaded with the UBRR0 value each time the counter has counted down to zero or when the UBRR0L Register is written
- A clock is generated each time the counter reaches zero
- This clock is the baud rate generator clock output (= FOSC/(UBRR0+1)). The transmitter divides the baud rate generator clock output by 2, 8, or 16 depending on mode. The baud rate generator output is used directly by the receiver's clock and data recovery units.

Baud rate vs. UBRR

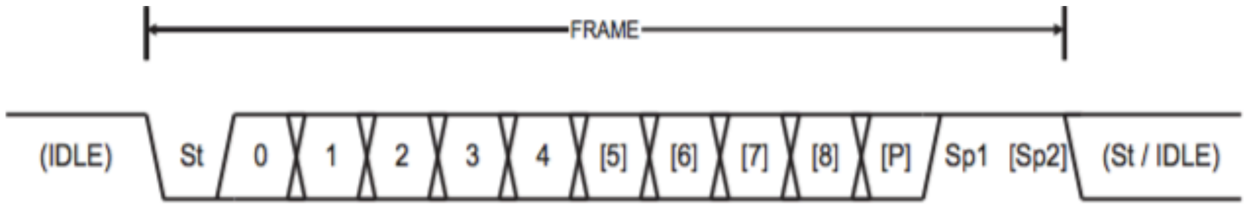


Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR + 1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR + 1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{OSC}}{2(UBRR + 1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

- Baud = bits per second
- f_OSC = System clock frequency
- UBRR = contents of UBRRL and UBRRH registers

Setting Data Frame Format

- Remember start/data/parity/stop bits
- A typical frame for USART/RS232 is usually 10 bits long: 1 start bit, 8 data bits, and a stop bit.
    - Many other combinations exist (30 in total – how?)

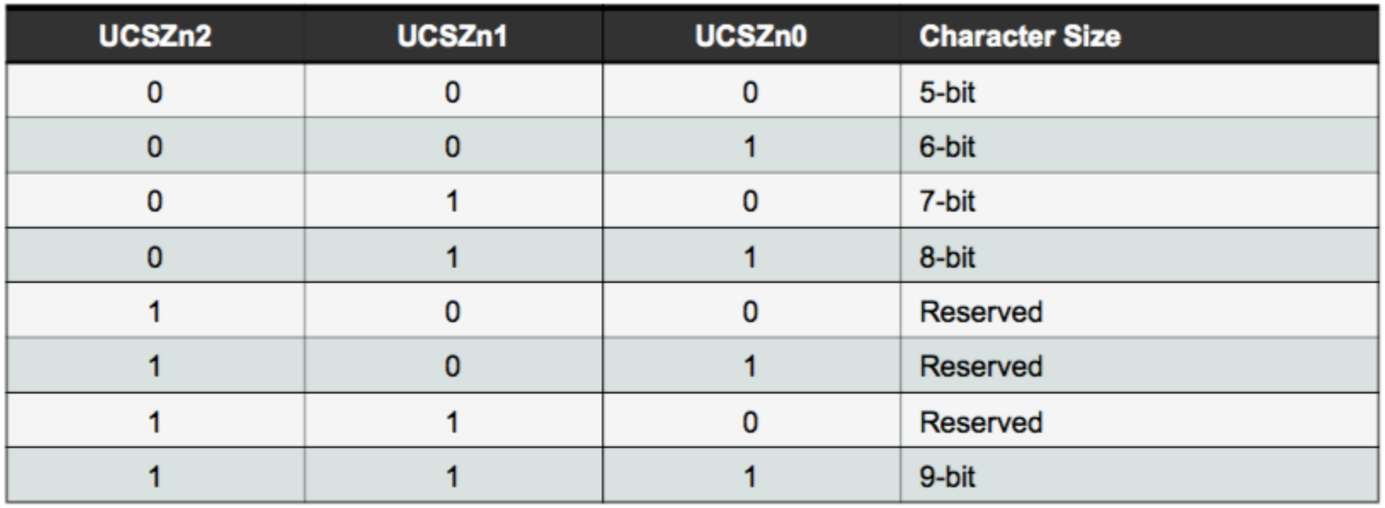


St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxDn or TxDn). An IDLE line must be high.

Setting the Number of Data Bits

- The data size used by the USART is set by the UCSZ2:0, bits in UCSRC Register
  - The Receiver and Transmitter use the same setting
- Note: Do not change these settings during communication as it might corrupt the data.
  - Make sure that you configure the same settings for both transmitter and receiver.

Table 20-11. UCSZn Bits Settings



UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

Setting the Number of Stop Bits

- This bit selects the number of stop bits to be inserted by the transmitter
- The Receiver ignores this setting
- The USBS bit is available in the UCSRC Register

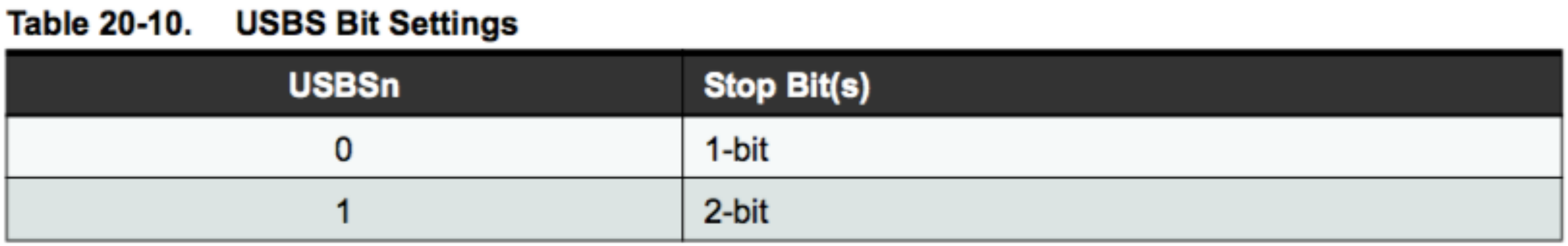


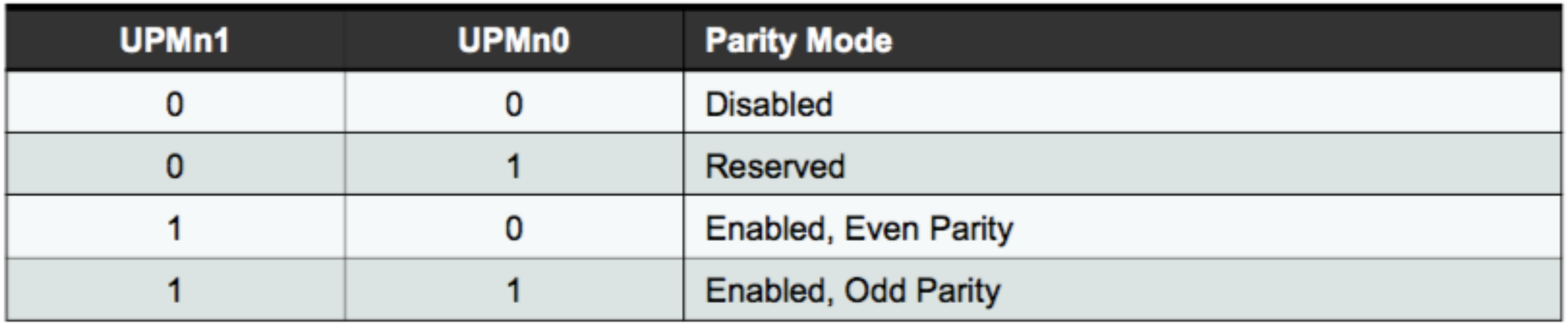
Table 20-10. USBS Bit Settings

USBSn	Stop Bit(s)
0	1-bit
1	2-bit

Setting the Parity Bit

- The Parity Generator calculates the parity bit for the serial frame data
- When parity bit is enabled (UPM1 = 1), the Transmitter control logic inserts the parity bit between the last data bit and the first stop bit of the frame that is sent
- The parity setting bits are available in the UPM1:0 bits in the UCSRC Register

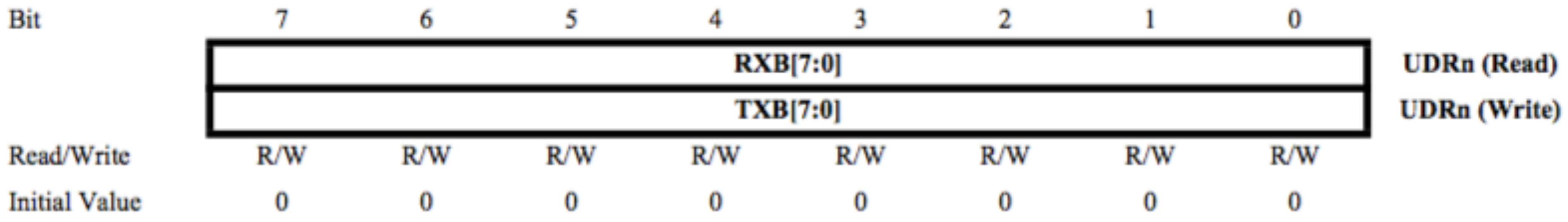
Table 20-9. UPMn Bits Settings



UPMn1	UPMn0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

UDR0: USART Data Register (16-bit)

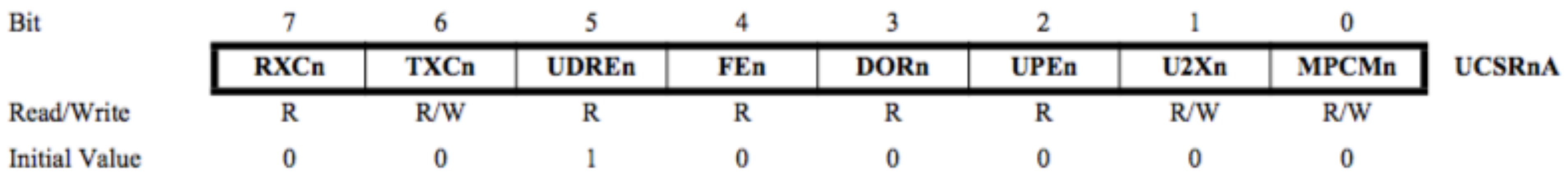
- The USART Transmit Data Buffer Register and USART Receive Data Buffer Registers share the same I/O address referred to as USART Data Register or UDR
- The Transmit Data Buffer Register (TXB) will be the destination for data written to the UDR Register location
- Reading the UDR Register location will return the contents of the Receive Data Buffer Register (RXB)
- For 5-, 6-, or 7-bit characters the upper unused bits will be ignored by the Transmitter and set to zero by the Receiver



Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDRn (Read)
	TXB[7:0]								UDRn (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

UCSR0A: USART Control and Status Register A (8-bit)

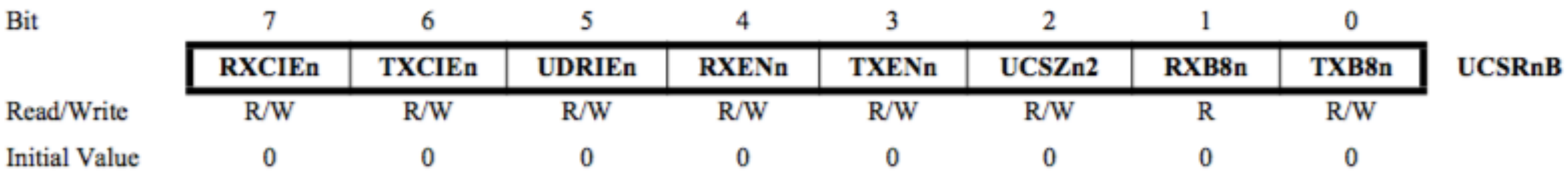
- Bit 7: RxC – USART Receive Complete Flag
- Bit 6: TxC – USART Transmit Complete Flag
- Bit 5: UDRE – USART Data Register Empty
- Bit 4: FE – Frame Error
- Bit 3: DOR – Data Overrun Error
- Bit 2: UPE – Parity Error
- Bit 1: U2X – Double Transmission Speed
- Bit 0: MPCM – Multi-Processor Communication Mode



Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

UCSR0B: USART Control and Status Register B (8-bit)

- Bit 7: RXCIE – RX Complete Interrupt Enable
- Bit 6: TXCIE – TX Complete Interrupt Enable
- Bit 5: UDRIE – USART Data Register Empty Interrupt Enable
- Bit 4: RXEN – Receiver Enable
- Bit 3: TXEN – Transmitter Enable
- Bit 2: UCSZ2 – Character Size
- Bit 1: RXB8 – Receive Data Bit 8
- Bit 0: TXB8 – Transmit Data Bit 8



Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

UCSR0C: USART Control and Status Register C (8-bit)

- Bit 7:6: UMSEL1:0 – USART Mode Select
- Bit 5:4: UPM1:0 – Parity
- Bit 3: USBS – Stop Bit Select
- Bit 2:1: UCSZ1:0 – Character Size
- Bit 0: UCPOL – Clock Polarity

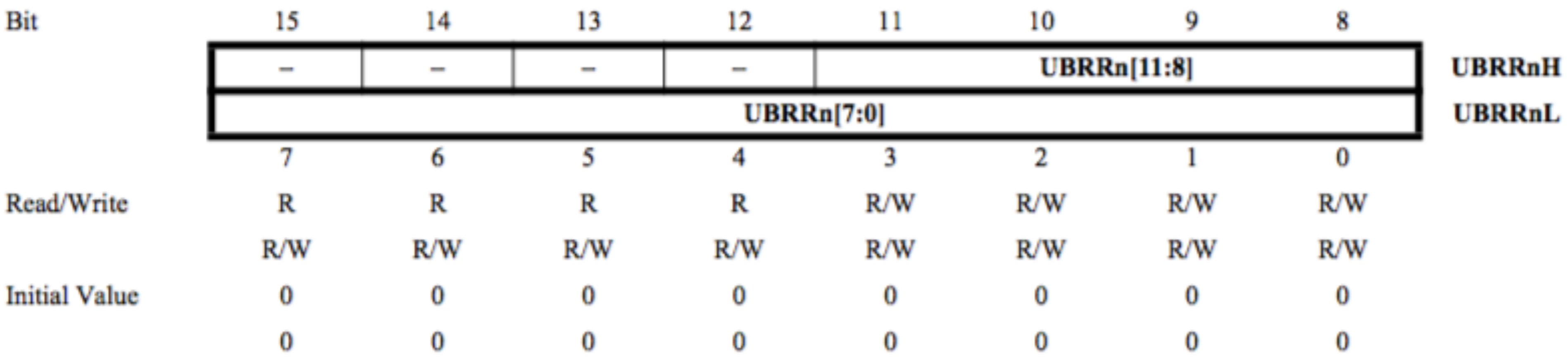
Table 20-8. UMSELn Bits Settings

UMSELn1	UMSELn0	Mode
0	0	Asynchronous USART
0	1	Synchronous USART
1	0	(Reserved)
1	1	Master SPI (MSPIM)[1]

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

UBRR0: USART Baud Rate Register

- 16-bit



Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRRn[11:8]				UBRRnH
	UBRRn[7:0]								UBRRnL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Coding example: Initialize UARt

- Initializing UART involves the following steps
  - Setting the Baud Rate
  - Setting Data Size (5/6/7/8/9 bits)
  - Enabling Reception/ Transmission (in the TXEN and RXEN bits in UCSRB)
  - Setting parity, and number of Stop Bits

```
// define some macros
#define BAUD 9600                                          // define baud
#define BAUDRATE ((F_CPU)/(BAUD*16UL)-1)                   // set baud rate
value for UBRR

// function to initialize UART
void uart_init (void)
{
    UBRR0H = (BAUDRATE>>8);         // shift the register right by 8 bits
    UBRR0L = BAUDRATE;              // set baud rate
    UCSR0B|= (1<<TXEN0)|(1<<RXEN0);   // enable receiver and transmitter
    UCSR0C|= (1<<UCSZ00)|(1<<UCSZ01);   // 8bit data format
}
```

Coding example: Transmission/Reception

- Transmission

```
// function to send data void
uart_transmit (unsigned char data) {
    while (!( UCSR0A & (1<<UDRE0))); // wait while register is free
    UDR0 = data;                     // load data in the register
}
```

- Reception

```
// function to receive data
unsigned char uart_recieve (void)
{
    while(!(UCSR0A) & (1<<RXC0)); // wait while data is being received
    return UDR0;                  // return 8-bit data
}
```

Send a Sequence of Bytes

```
for(i = 0; i < 8; i++) {
      while(!(UCSR0A & UDRE0));
      UDR0 = data[i];
}
```

- How many cycles does the while loop take to execute? Assume:
  - Processor operates at 18 MHz
  - 57600 baud serial speed
  - 8/57600 seconds = 139 microseconds
- Neglecting loop/assignment overhead, each while loop will consume 2500 cycles

AVR ATmega Peripherals

- 23 General Purpose IO Bits
- Two 8 bit & one 16 bit timer/counters
- Real time counter with separate oscillator
- 6 PWM Channels
- 6 or 8 ADC channels (depends on package)
- Serial USART
- SPI & I^2C Serial Interfaces
- Analog comparator
- Programmable watchdog timer

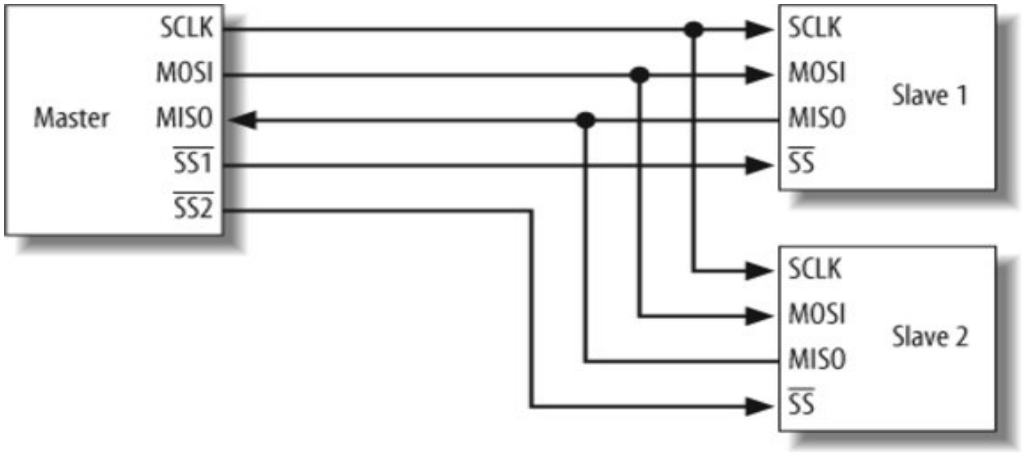
Additional Slides on SPI and I²C

Serial Peripheral Interface (SPI)

- Industry standard serial protocol for communication between local devices
- Master/Slave protocol
- 3 Wire interface
- Slaves addressed via Slave Select (SS) inputs

SPI Bus – Signal Descriptions

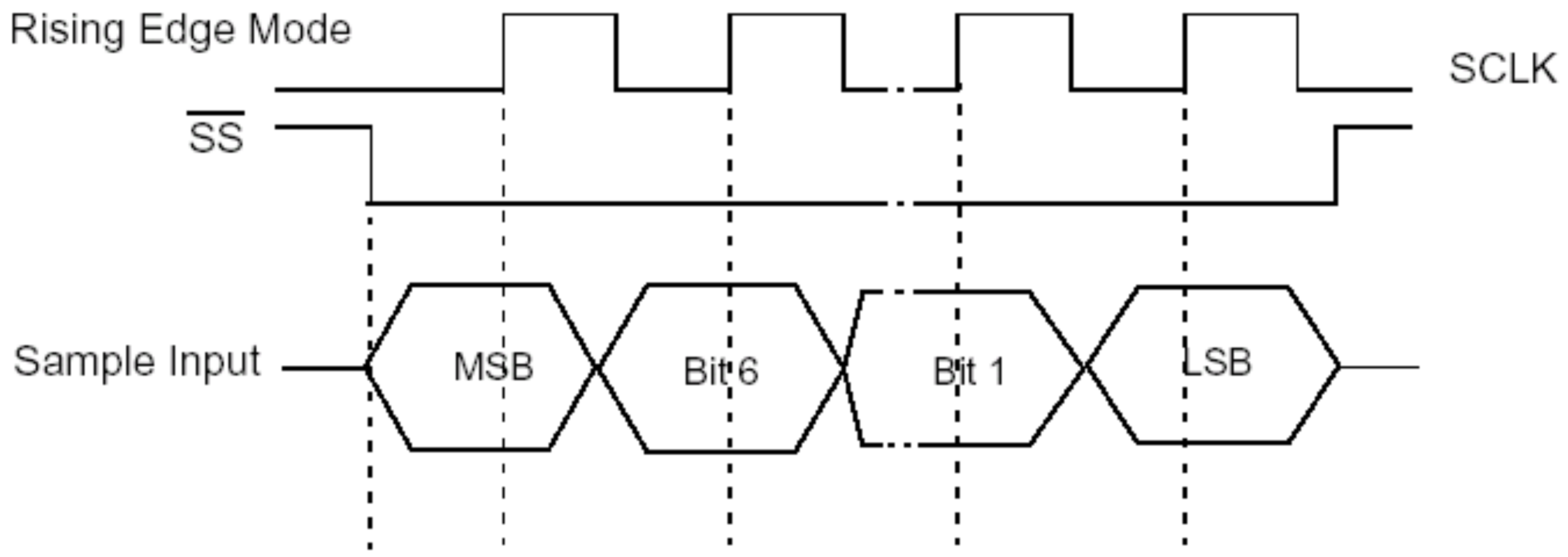
SCLK	Serial Clock
MOSI	Master Out Slave In
MISO	Master In Slave Out
SS	Slave Select



SPI: Benefits

- Simple, easy to implement interface
- Software can “bit-bang” the bus
- Large support base
- Only requires 2-4 pins:
  - SCLK: Serial Clock Line
  - MOSI: Master Data Out/Slave Data In
  - MISO: Master Data In/Slave Data Out
  - SS: Slave Select (active low)
- Good for data streaming devices
  - Full duplex interface

SPI: Data Transfer



SPI: Example

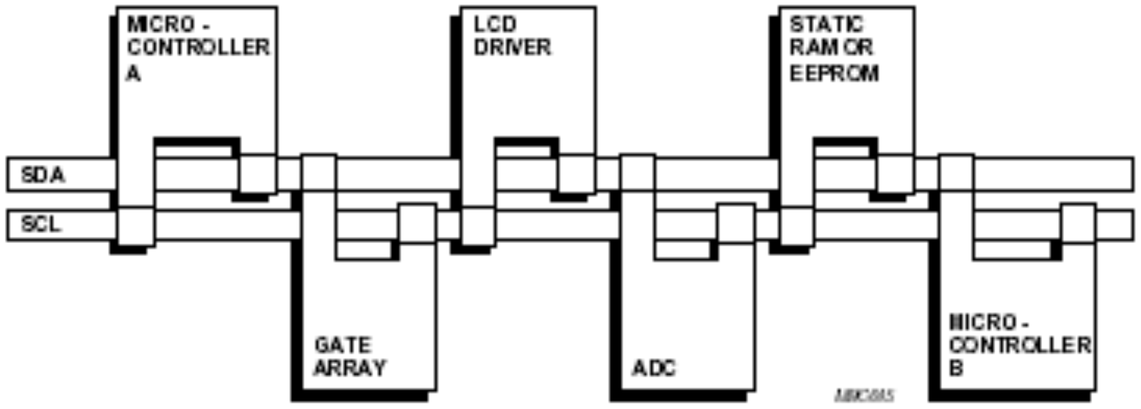
```
unsigned char adcget( void )
{
   unsigned char result ;
   unsigned char adcbit ;
   result = 0;          // initialize result
   adcselect = 0;       // start with chip selected and then

   for ( adcbit=0 ; adcbit < 8 ; adcbit++ )
   {
        result = ( result << 1 ) | adcdata ; // shift A/D bit in
        adcclock = 1; adcclock = 0; // pulse clock pin once
   }
   adcselect = 1;       // deselect chip
   return (result);
}
```

I²C Bus Interface

- Industry standard serial protocol for communication between local devices
- Master/Slave protocol
- 2 Wire interface
- Byte oriented messages
- Slave address embedded in command

I²C Bus Signal Descriptions

SDA	Serial Data
SCL	Serial Clock



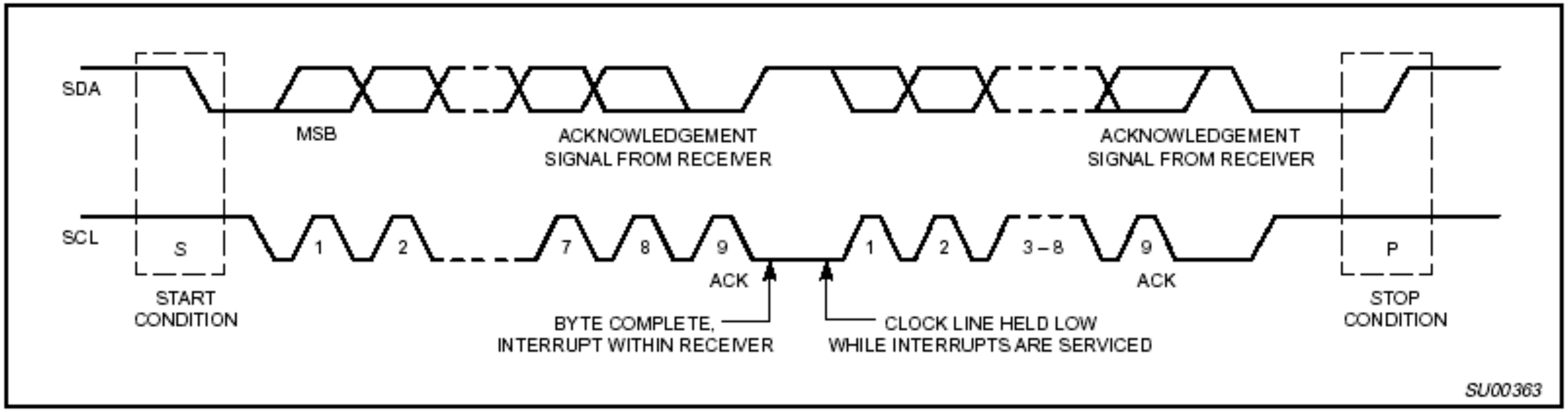
I²C: Benefits

- Simple, easy to implement interface.
- Software can "bit-bang" the bus.
- Large support base.
- Only requires 2 pins:
  - SDA: Serial Data Line
  - SCL: Serial Clock Line
- Multi-Master:
  - Arbitration & Clock generation rules

I²C: Data Transfer

- Start (1 bit)
- Address (7 bits)
- Read/Not_Write (1 bit)
- ACK (1 bit)
- Data (8 bits)
- ACK (1 bit)
- [etc]
- Stop (1 bit)

I²C: Data Transfer


SDA
SCL
S
START CONDITION
MSB
1
2
7
8
9
ACK
ACKNOWLEDGEMENT SIGNAL FROM RECEIVER
BYTE COMPLETE, INTERRUPT WITHIN RECEIVER
CLOCK LINE HELD LOW WHILE INTERRUPTS ARE SERVICED
1
2
3 – 8
9
ACK
ACKNOWLEDGEMENT SIGNAL FROM RECEIVER
P
STOP CONDITION
SU00363

I²C: Extensions

- Fast mode: 400 kbit/s
- High-speed modes: 3.4 Mbit/s & higher
- 10-bit addressing
- Multi-Master

Typical SPI & I²C Devices

- EEPROM
- IO Expanders
- Real Time Clocks
- ADC & DAC
- Temperature sensors
- Ultrasonic range finders
- Compass
- Servo / Motor Controller
- LED Display