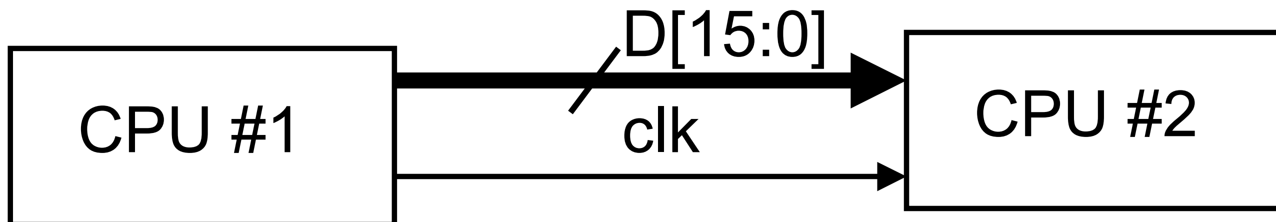
Parallel IO

Parallel IO – data sent over a group of parallel wires. Typically, a clock is used for synchronization.

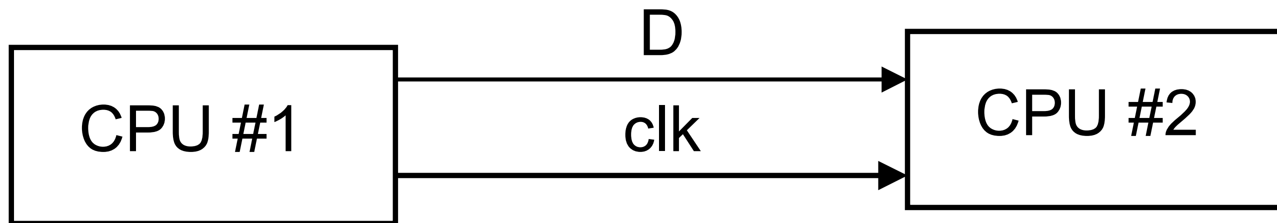


A 16-bit data channel is shown above. If data is transferred each rising clock edge, and clock rate is 300 MHz, then the **data transfer rate** **(bandwidth)** in bytes/sec is:

2 Bytes/clock period = 2 /(1/300e06)s
= 2 * 300e06/s = 600e06/s
= 600 MB/s (MB = MBytes)

Serial IO

Serial IO – data sent one bit at a time, over a single wire.
A clock may or may not be used for synchronization



Question: Assuming one bit is sent each rising clock edge, how fast does the clock have to be achieve 600 MB/s?

600 MByte/s = 600 MBytes/s * 8 bits/1Byte = 4800Mb/s

Clock period = 1/4800e06
Clock Frequence = 1/clock period = 4800e06 = 4.8e09 = 4.8GHz

Parallel vs. Serial IO

Parallel IO Pros/Cons

Pros: Speed, can increase bandwidth by either making data channel wider or increasing clock frequency

Cons: Expensive (wires cost money!). Short distance only – long parallel wire causes crosstalk, data corruption.

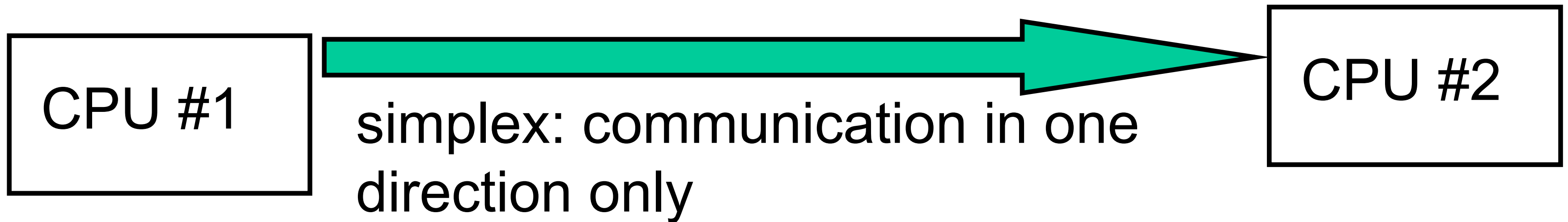
Serial IO Pros/Cons

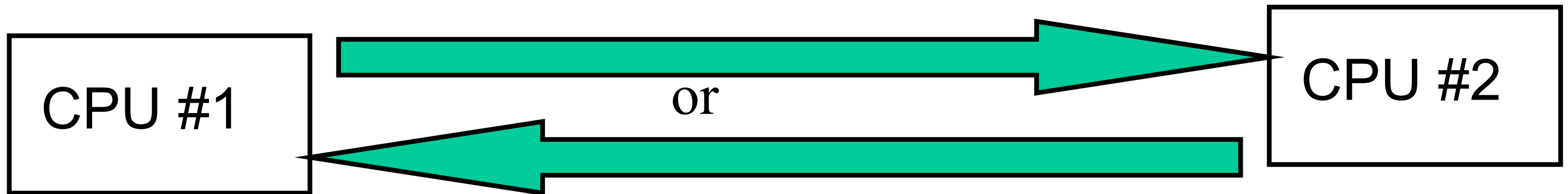
Pros: Cheap, very few wires needed. Good for long distance interconnect.

Cons: Speed; the fastest serial link will typically have lower bandwidth than the fastest parallel link. However, for long distances (meters), new fast serial IO standards (USB2, Firewire) have replaced older parallel IO standards.

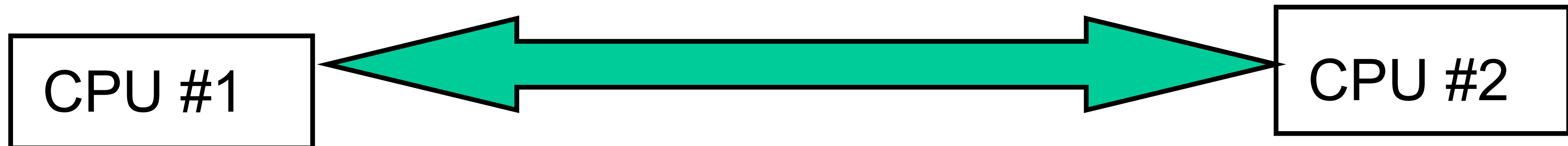
simplex vs half-duplex vs full-duplex

For communication channels





Half-duplex: communication in either direction, but only one way at a time

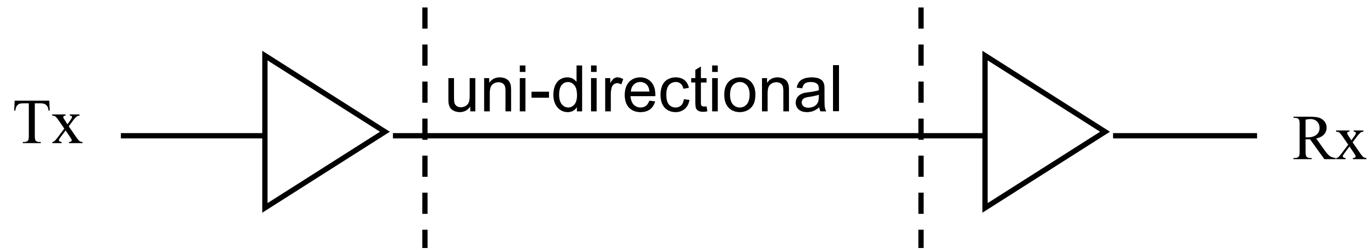


Full-duplex: communication in both directions at same time.

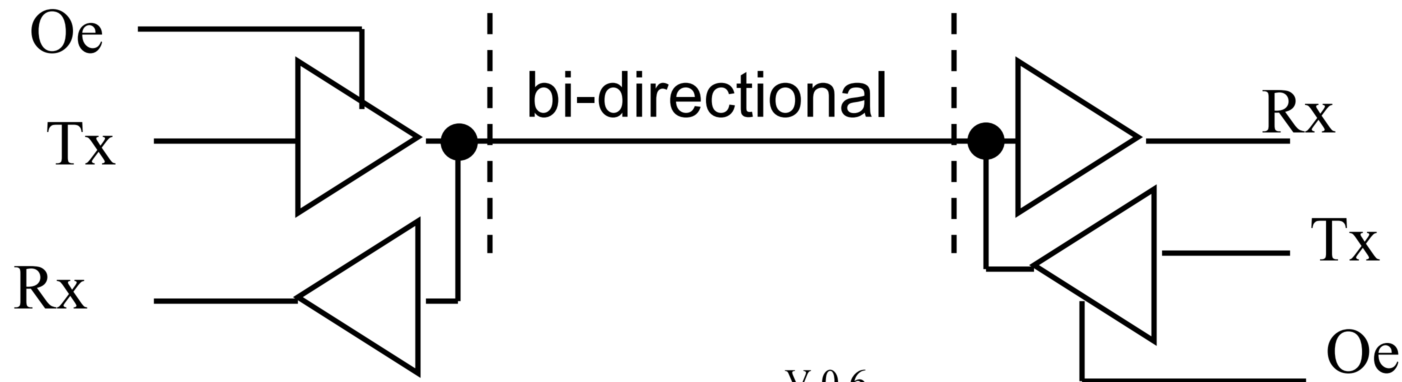
Wires: Simplex, Half-duplex

For wires:

simplex wire: communication occurs only in one direction.

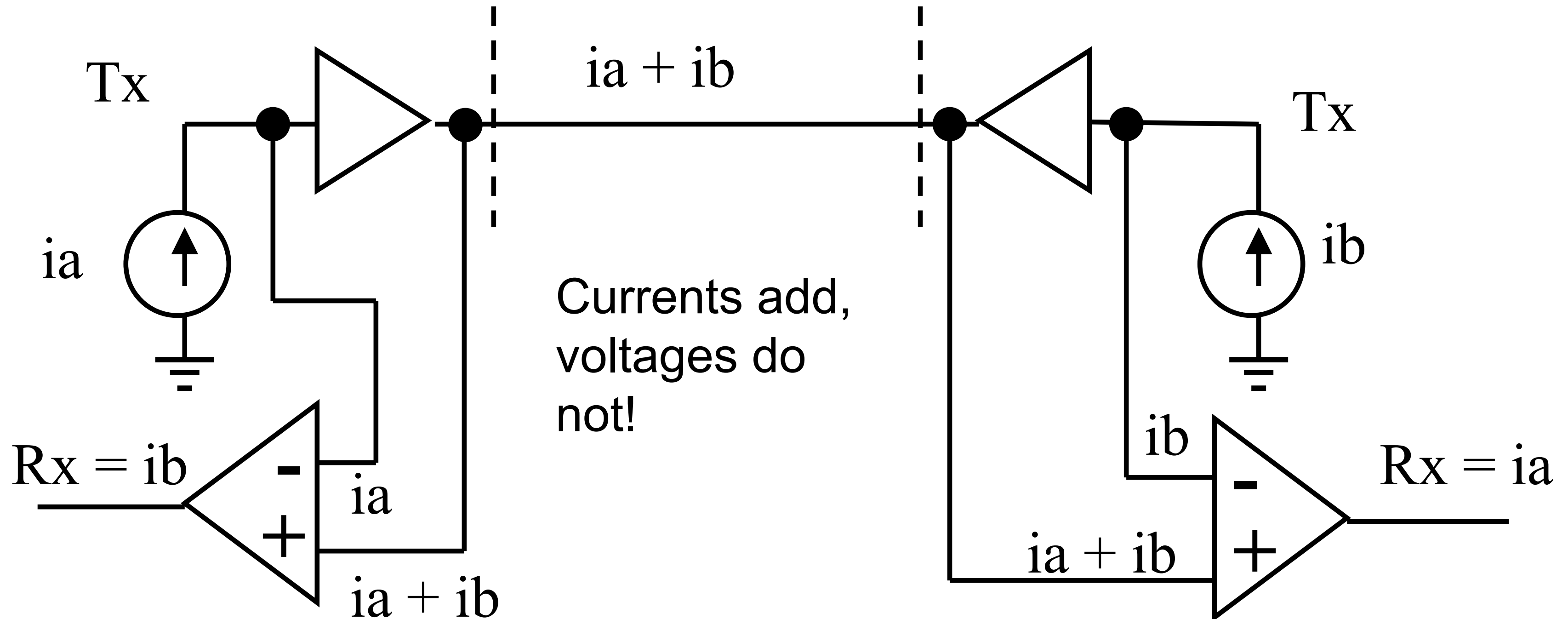


half-duplex wire: communication can occur in either direction, but with voltage signaling only one direction at a time.

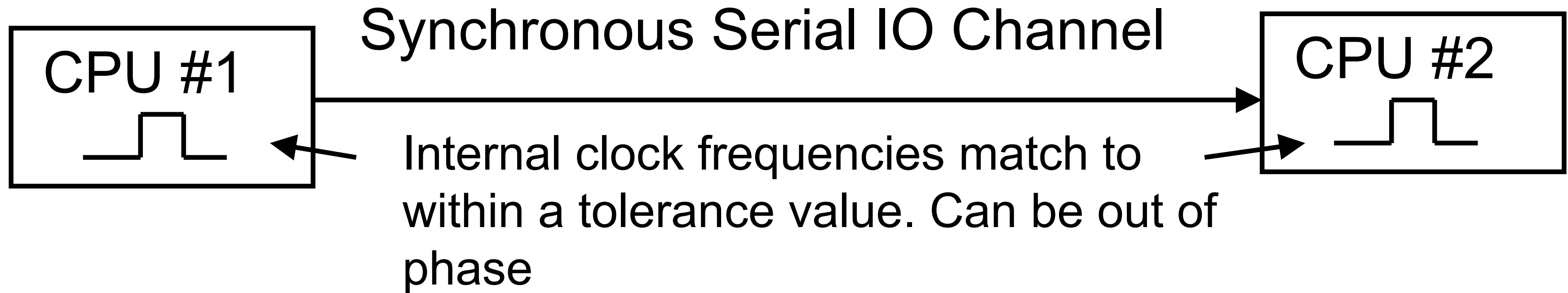


Wires: Full Duplex

Current mode signaling allows full duplex communication over a single wire. Used for communication in some advanced chipsets.



Synchronous Serial IO



Synchronous serial IO either

(a) sends the clock as a separate wire

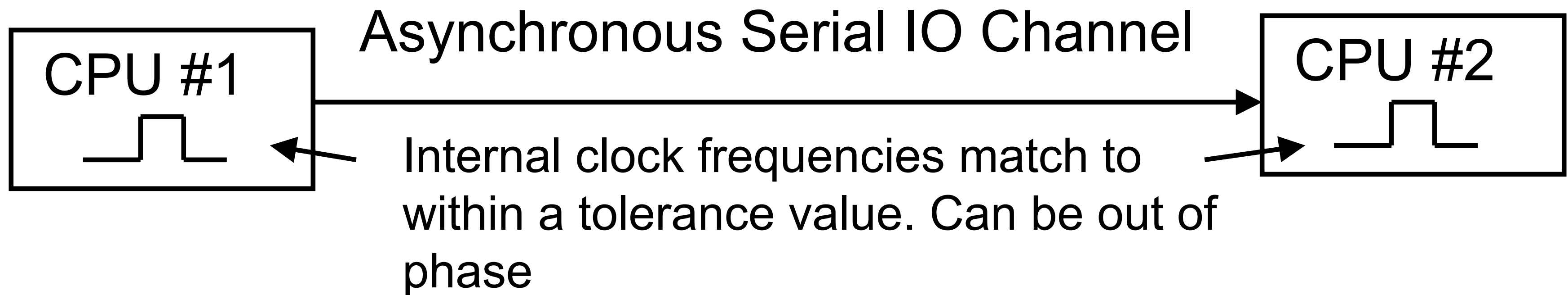
OR

(b) receiver (CPU #2) extracts clock from data stream or uses a Phase-Locked-Loop (PLL) and changes in the data stream to synchronize internal clock (phase alignment) to data stream.

For PLL synchronization, the data line must be guaranteed to have a minimum number of state changes ($0 \rightarrow 1$ or $1 \rightarrow 0$) within a particular time interval (*transition density*).

Synchronous serial IO can achieve high speeds; all new high speed serial standards are synchronous.

Asynchronous Serial IO



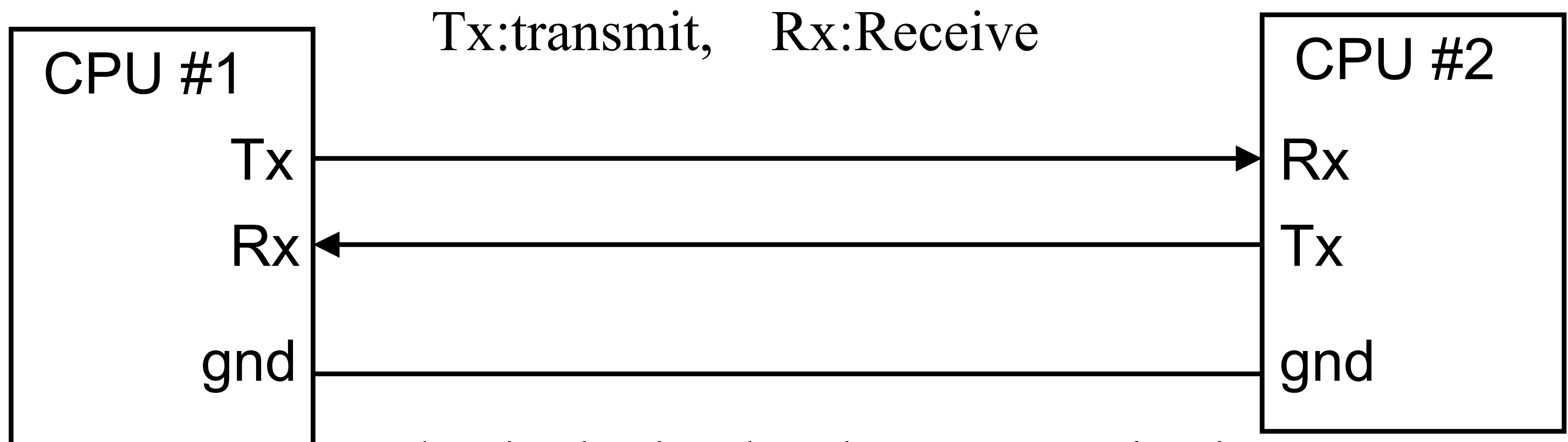
Asynchronous Serial I/O does not transmit the clock on a separate wire nor does it guarantee a particular transition density (ie., the data line could remain in the same state, either '1' or '0' for the duration of the transmission after the initial state change indicating start of transmission).

Asynchronous Serial I/O is used in older standards, is easy to implement, but is slower than synchronous serial standards.

A Three-Wire Async Serial Interface

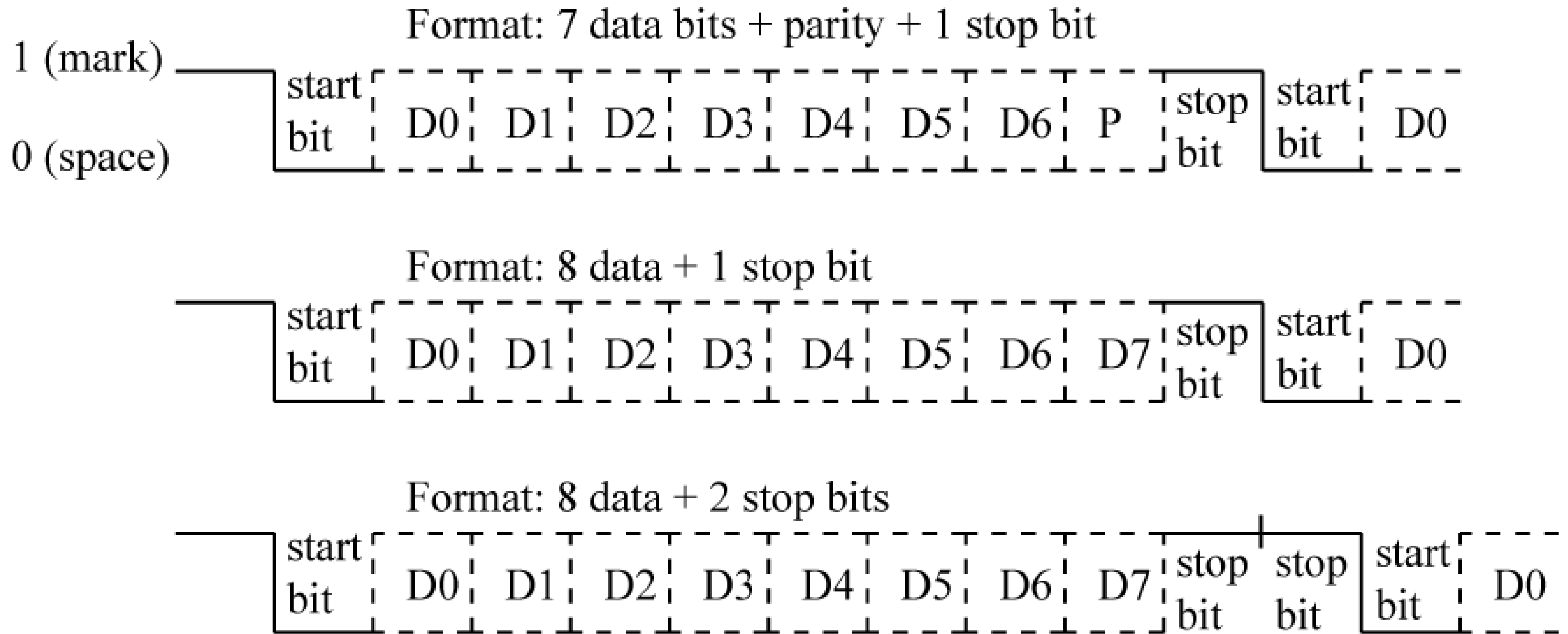
We will use a three-wire asynchronous serial interface to connect the PIC to an external PC.

This interface standard is known as RS-232 (there are more wires defined in the standard, we will only use 3 wires)



Each wire is simplex, but communication channel is full duplex

Asynchronous Serial Data Formats



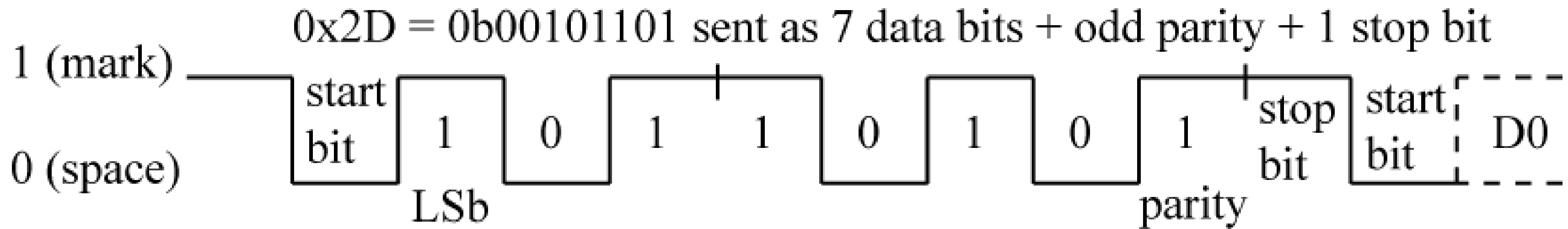
Data sent LSb to MSb.

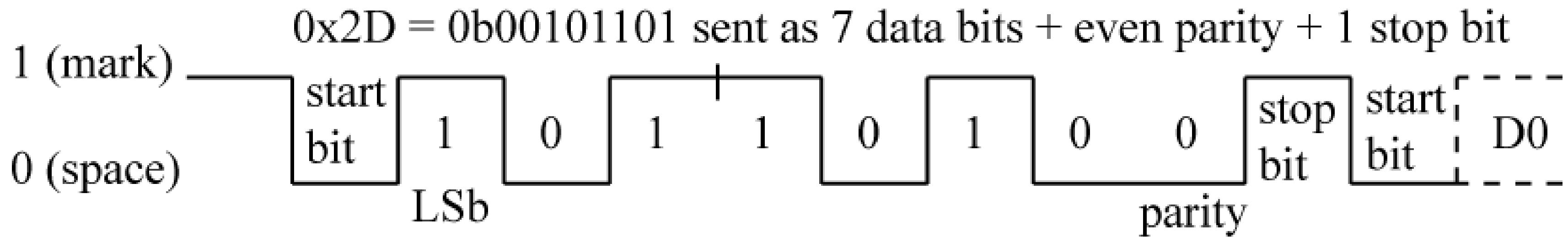
On the PIC, will use 8 data bits.

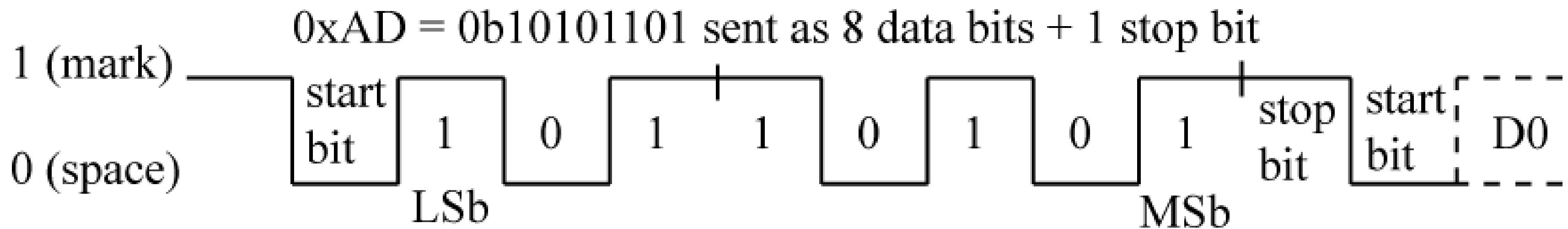
Parity

- A **parity** bit is an extra bit added to a data frame to detect a single bit error
  - A single bit error is when one bit of the frame was received incorrectly (read as '0' when should have been '1', or vice-versa).
  - Not guaranteed to detect multi-bit errors
- Odd parity – parity bit value makes the total number of '1' bits in the frame odd
  - For 7-bit data value 0x56 (1010110),
    odd parity bit = '1'
- Even parity – parity bit value makes the total number of '1' bits in the frame even
  - For 7-bit data value 0x56 (1010110),
    even parity bit = '0'

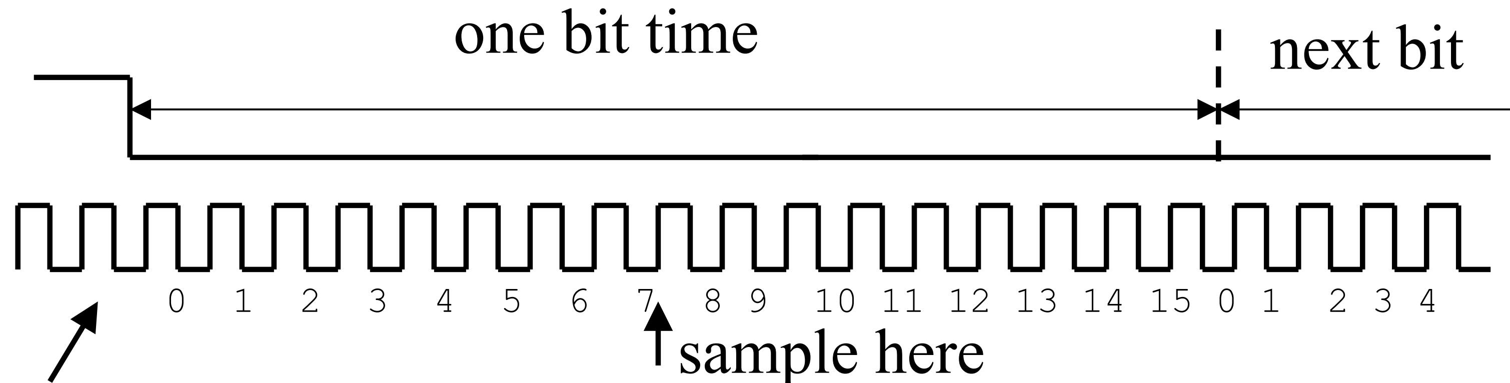
Example







Receiver Sampling



Receiver clock; period usually either 64x or 16x bit time (above is 16x).

At start bit, internal 4-bit counter set to 0. Sample at mid-point of bit time (counter value 7 or 8, some receivers sample at 7,8 and 9 and only accept bit if all values are the same – do this for glitch rejection).

Receiver/Transmitter clocks not perfectly matched. Our tolerance is ½ bit time (50%) spread over entire frame. Assuming a 10 bit frame, maximum mismatch between Rx/Tx clocks is 50%/10 = 5%,

Baud Rate vs Bits Per Second

- Baud rate is the rate at which signaling events are sent
- Bits per second (bps) is the number of bits transferred per second (any type of bits, data or overhead bits)
- If only a ‘1’ or ‘0’ is sent for each signaling event, then baud rate = bps
- However, could use a signaling protocol that transfers multiple bits per signaling event
  - i.e., use 4 different voltage levels, send two bits of data per signaling event (00 = -15v, 01= -5v, 10=+5v, 11 = 5v).
  - In this case, bit rate will be double the baud rate
- The effective data rate is the rate at which data is transferred, minus the overhead bits (ie. start and stop bits).

Common Baud Rates

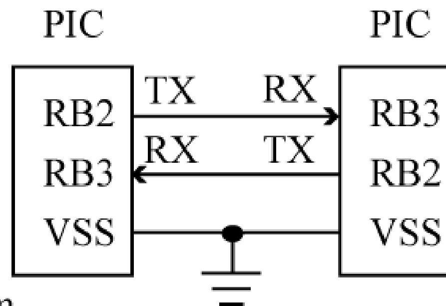
Baud Rate	Divisor for 29.4912 MHz
115200	256
57600	512
38400	768 (256 x 3)
19200	1536 (512 x 3)
9600	3072 (1024 x 3)
4800	6144 (2048 x 3)
1200	24576 (8192 x 3)

The PIC oscillator is divided down in order to provide the Tx/Rx clocks.

The divisor values on the right show that the commonly-used baud rates are even multiples of 29.4912 MHz. This means these baud rates can be accurately reproduced by the PIC using this clock frequency.

Software-driven Serial I/O

Software Asynchronous Serial Link



Not an efficient use of CPU resources

putch - send one 8-bit datum

getch - receive one 8-bit datum

Send start+8 data+ stop

```
void putch(
unsigned char c)
{
  unsigned char i;
  RB2 = 0;              } Send start bit
  delay_1bit();
  // Send 8 data bits, LSb to MSb
  for(i=0;i<8;i++) {
    if (bittst(c,0))
      RB2 = 1;          } One data bit
    else RB2 = 0;
    delay_1bit();
    c = c >> 1;  <-- Right shift for next bit
  }
  RB2 = 1;              } Send stop bit
  delay_1bit();
}
```

Receive start+8 data+ stop

```
unsigned char getch(void)
{
  unsigned char i,c;

  c = 0x00;                 Wait for start bit
  while(RB3); <--           (exit when RB3 = = 0)

  delay_onehalf_bit(); <--  Wait for middle of bit time

  for(i=0;i<8;i++) {
   delay_1bit();
   if (RB3) c = c | 0x80;   } Receive one bit
   if (i != 7) c = c >> 1;  <-- Right shift for next bit
  }
  delay_1bit();
  return(c);  <-- Receive stop bit, no error check
}
```

Copyright Thomson/Delmar Learning 2005. All Rights Reserved.

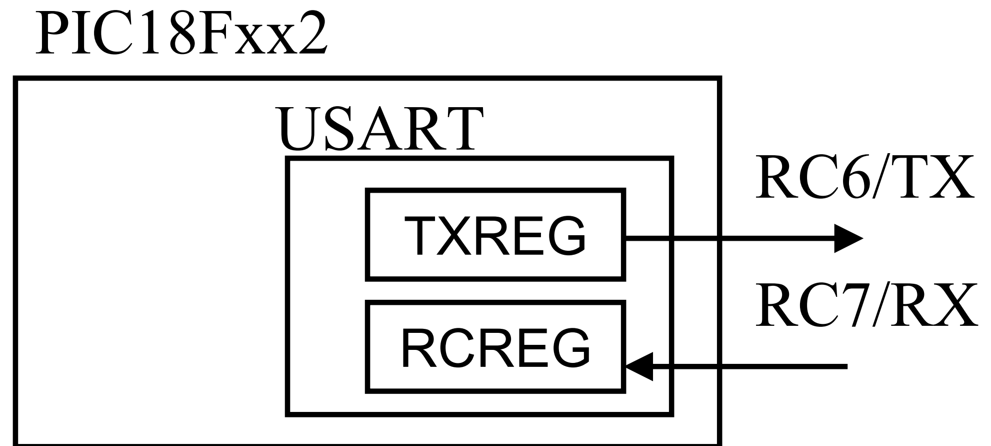
PIC18Fxx2 USART

USART → **U**niversal **S**ynchronous **A**synchronous **R**eceiver **T**ransmitter

Hardware module in PIC18Fxx2 that implements both synchronous and asynchronous serial IO. We will use asynchronous mode.

Frees the processor from having to implement software delay loops; receive/transmit done by USART while processor can do other tasks.

Will always use 8-bit, no parity for PIC18Fxx2 serial IO.



USART Registers

- RCREG – holds a received character; read this to get character
- TXREG – write to this register to send a character
- RCSTA – contains status bits for received character
- SPBRG and TXSTA control baud rate
  - TXSTA status bits also select between async/sync IO, enable TX transmission
- PIR1 register contains status bits
  - TXIF (transmit interrupt flag), '1' if TXREG is empty
  - RCIF (receive interrupt flag), '1' if RCREG is full

RCIF, TXIF Bits

TABLE 16-6: REGISTERS ASSOCIATED WITH ASYNCHRONOUS TRANSMI

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF

Will be a '1' when RCREG has a character.

Wait until RCIF=1, then read RCREG to get received character.

Will be a '0' if TXREG is full (last character written to TXREG has not been sent yet).

Wait until TXREG=1, then write character to TXREG

Copyright Microchip, from PIC18xx2 Datasheet DS39564B

Transmit Hardware

TXREG is **double-buffered**. If TSR is empty, after write to TXREG, then TXREG transferred to TSR and TREG empty again (TXIF = 1). Do not have to wait for last character to be sent before writing new character.

Data sent LSB to MSB

Output Shift register, TXREG is buffer register

Receive Hardware

RCREG is also a buffered register via 2 deep FIFO. This gives the processor more freedom in how fast it responds to received characters.

Figure redrawn by author from PIC18Fxx2 datasheet (DS39564B), Microchip Technology Inc.

CREN
OERR
FERR
MSb
LSb
STOP | 8 | 7 | • • • | 1 | 0 | START
RSR register
x64 Baud Rate Clk
SPBRG
Baud Rate Generator
÷ 64 or ÷ 16
RX9
Pin Buffer and Control
Data Recovery
RC7/RX/DT pin
SPEN
RX9D
RCREG Register
FIFO
8
RCIF
Interrupt
RCIE
Data Bus

Two-deep FIFO. Can hold 2 characters, while 3rd character is being shifted into RSR register.

getch()/putch() (USART)

```
void putch(unsigned char c)
{
  // wait until TXREG empty
  while (!TXIF){
    asm("clrwdt");
    };
  TXREG = c;
  asm("clrwdt");
}
```

```
unsigned char getch(void)
{
   // wait until data available
  while (!RCIF){
    // ok to wait forever for input
    // so clear watchdog timer
    asm("clrwdt");
  };
  return(RCREG);
}
```

These subroutines much simpler than software-based serial I/O. The *putch*/*getch* single character functions is used by the library function *printf()*.

The *asm("clrwdt")* instructions are included in case the watchdog timer is enabled.

Baud Rate Control

The baud rate is controlled by the 8-bit value in the SBPRG register and the BRGH bit (bit 2 in TXSTA register).

Baud_Rate = Fosc/ [K*(SBPRG+1)]

or

SBPRG = (Fosc/[K*Baud_Rate]) – 1

K = 16 if BRGH = 1 (high speed mode), then K = 16

K = 64 if If BRGH = 0 (low speed mode)

Baud Rate Examples

Desired baud rate of 9600, Fosc = 29.4912 MHz

What is SBPRG value for high speed mode?

SBPRG = (29.4912e06/[16*9600]) – 1
= 191

What is SBPRG value for low speed mode?

SBPRG = (29.4912e06/[64*9600]) – 1
= 47

Sample Baud Rates

FOSC = 29.4912 MHz

Baud Rate	SPBRG (Hi Speed)	Actual	%err	SPBRG (Low Speed)	Actual	%err
230400	7	230400	0.0%	1	230400	0.0%
115200	15	115200	0.0%	3	115200	0.0%
57600	31	57600	0.0%	7	57600	0.0%
38400	47	38400	0.0%	11	38400	0.0%
19200	95	19200	0.0%	23	19200	0.0%
9600	191	9600	0.0%	47	9600	0.0%
4800	383	n/a		95	4800	0.0%

FOSC = 40 MHz

Baud Rate	SPBRG (Hi Speed)	Actual	%err	SPBRG (Low Speed)	Actual	%err
230400	10	227272.7	-1.4%	2	208333	-9.6%
115200	21	113636.4	-1.4%	4	125000	8.5%
57600	42	58139.53	0.9%	10	56818.2	-1.4%
38400	64	38461.54	0.2%	15	39062.5	1.7%
19200	129	19230.77	0.2%	32	18939.4	-1.4%
9600	259	n/a		64	9615.38	0.2%
4800	520	n/a		129	4807.69	0.2%

} Error is too large

Enabling Async Serial IO

1. Configure serial port pins (RC6/TX, RC7/RX) via SPEN bit (RCSTA:7 =1). Must also set TRISC7=1, (RC7 as input), TRISC6 =0 (RC6 as output).
2. Select high or low speed baud rate via BRGH bit (bit 2) of TXSTA register
3. Select async mode SYNC bit (TXSTA:4 = 0)
4. Select 8-bit transmit via TX9 bit (TXSTA:6 = 0)
5. Select 8-bit receive via RX9 bit (RCSTA:6 = 0)
6. Enable transmit port via TXEN bit (TXSTA:5 = 1)
7. Enable receive port via CREN bit (RCSTA:4 = 1)

Example *C* code to Enable Serial I/O

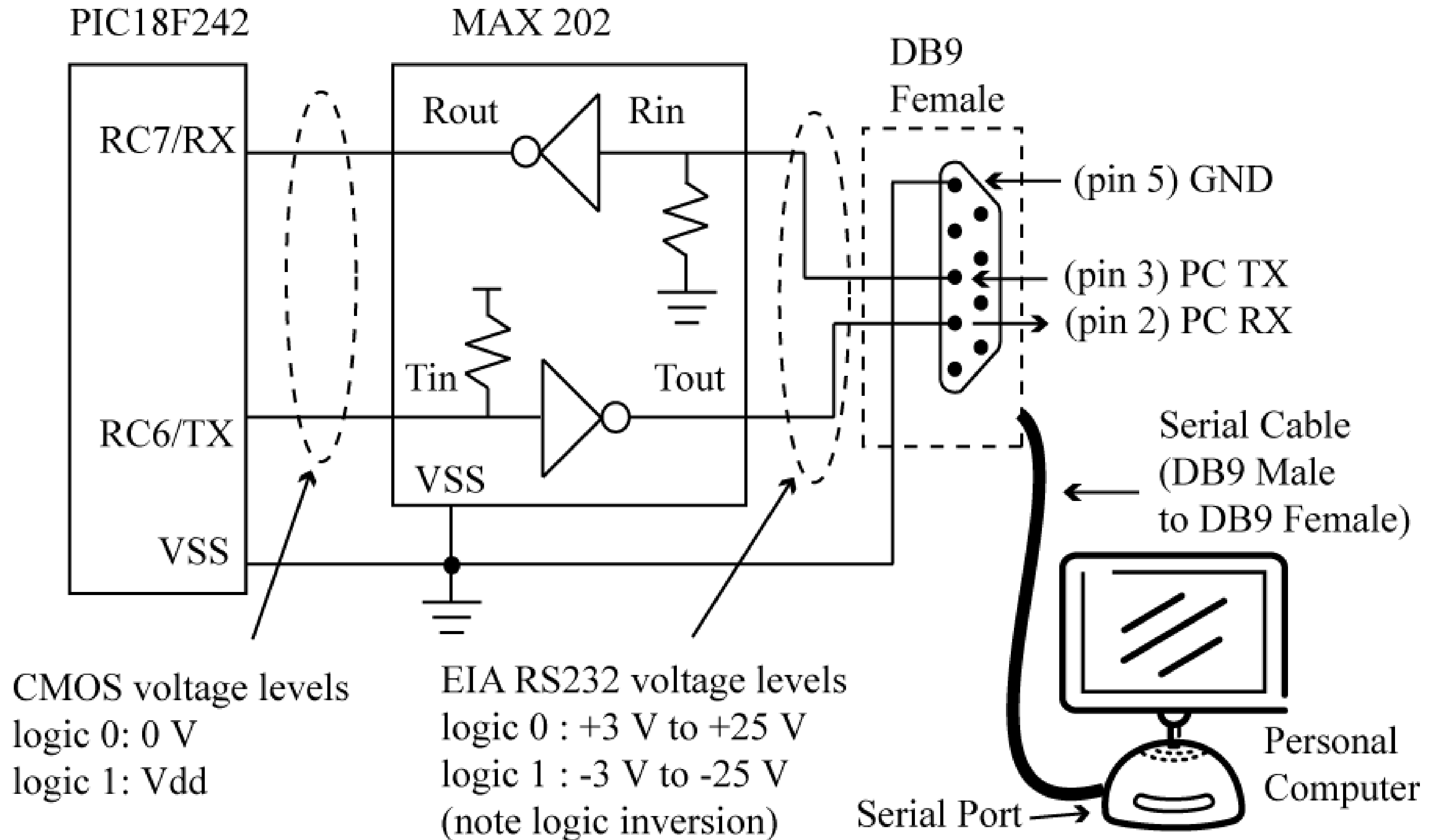
```
void serial_init(
char brg,
char hi_speed)
{
  // setup Async communication
  TX9 = 0;
  TXEN = 1; // transmit enable
  SYNC = 0; // async mode
  if (hi_speed)   BRGH = 1; // hi speed mode
  else BRGH = 0; // lo speed mode
  SPBRG = brg;
  bitset(TRISC, 7); // RC7 input
  bitclr(TRISC, 6); // RC6 output
  RX9 = 0;  // 8-bit reception
  SPEN = 1; // serial port enable
  CREN = 0; // clear enable first
  CREN = 1; // now enable first
}
```

Loaded into SPBRG reg to set baud rate. (arrow pointing to `char brg,`)

Receive Error Conditions

- FERR bit (RCSTA:2) is set when a framing error is detected
  - A framing error occurs when a STOP bit is detected as a ‘0’ value.
  - Most likely to occur when actual baud rate is slower than expected baud rate.
- OERR bit (RCSTA:1) is set when an overrun error is detected
  - Waited too long to read RCREG and FIFO fills up
  - Set when stop bit of 3rd byte is detected (2 bytes in FIFO, and 3rd byte is shifted in)
  - All receive activity is stopped; to reset, clear CREN (RCSTA:4), then set CREN.

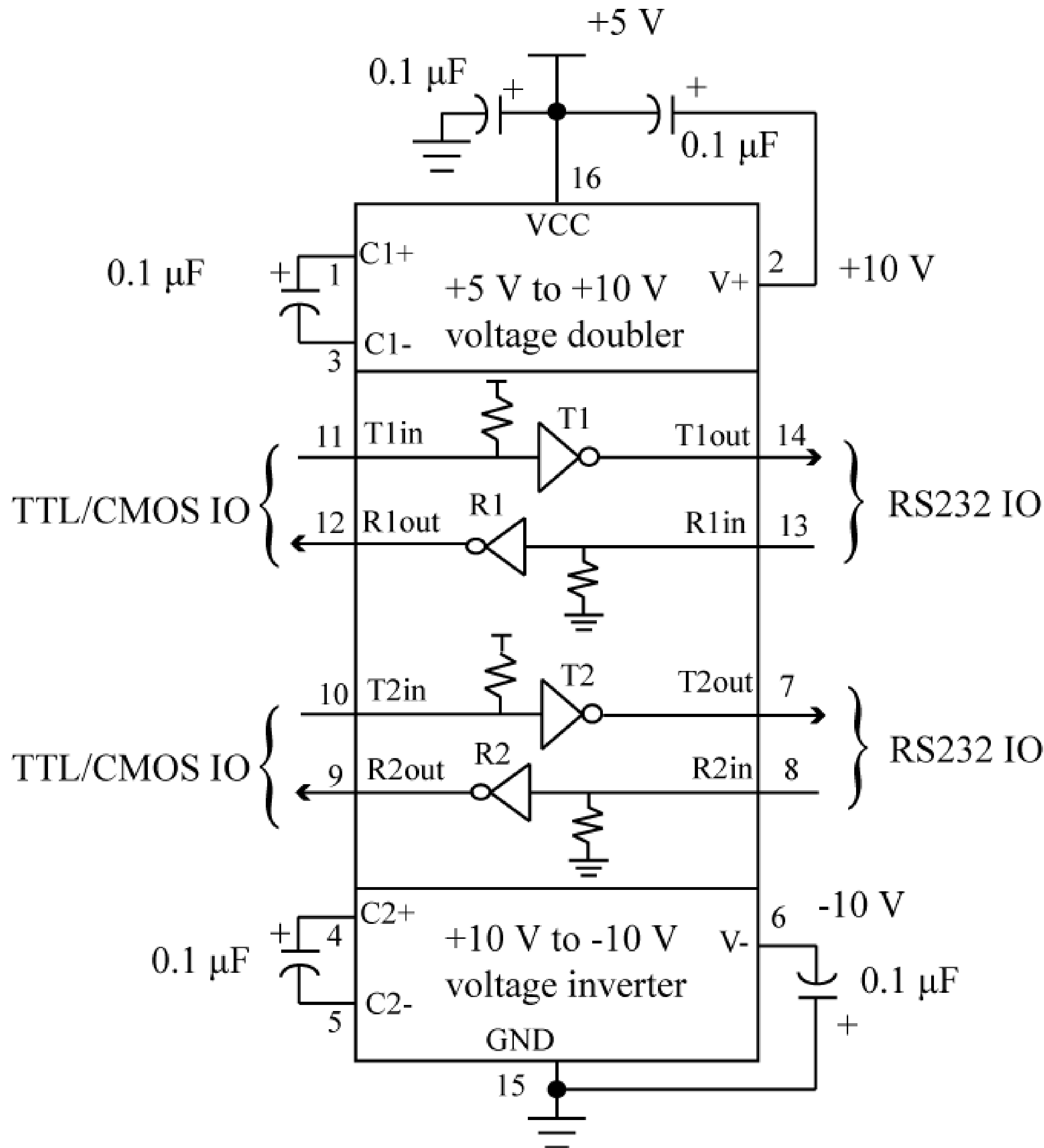
PIC18Fxx2 to PC Serial IO Connection



Copyright Thomson/Delmar Learning 2005. All Rights Reserved.

What is EIA-RS232?

- An interface standard originally used to connect PCs to modems
  - A modem is a device used to send digital data over phone lines
  - The standard defines voltage levels, cable length, connector pinouts, etc
- There are other signals in the standard beside TX, RX, Gnd
  - The other signals are used for modem control (Data Carrier Detect, Ring Indicator, etc) and flow control (flow control signals are used to determine if a device is ready to accept data or not)
  - We will not cover the other signals in the RS232 standard



MAXIM 202 RS232 driver/receiver

Converts RS232 voltage levels to digital levels and vice-versa

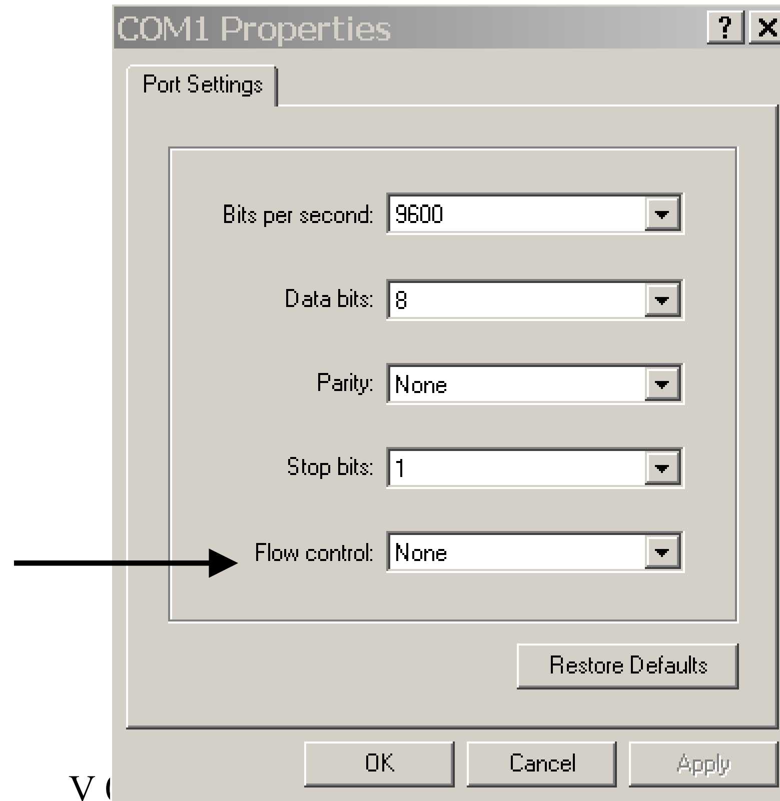
External capacitors used with internal charge pump circuit to produce +/- 10V from 5V supply

HyperTerminal

Will use HyperTerminal program on PC to communicate with PIC18F242.

Under Programs→Accessories →Communications → HyperTerminal

When configuring HyperTerminal connection, must know port number (COM1/COM2/etc), baud rate, data bits (8), parity (none), stop bits (1), and flow control(none)



Very important to set flow control to **none** since we are only using a 3-wire connection and not using the handshaking lines in the RS232 standard. If you forget this, then will not receive any characters.

echo.c – Testing the serial port

```
main(void){
    unsigned char c;
    // init serial port
    // 19200 in HSPLL mode, crystal = 7.3728 MHz
      serial_init(95,1);
      while(1) {
        c = getch();      // wait for character
        c++;              // increment character
        putch(c);         // send it back
      }
    }
```

putch(), getch(), serial_init() functions are not shown.

This reads one character from serial port, increments its value, then echos it back. Thus, ‘A’ is echoed as ‘B’; ‘B’ as ‘C’, etc.

What do you have to know?

- Difference between async/sync serial IO
- Format of async serial IO frames
- Details of PIC18Fxx2 USART operation for asynchronous IO
- Definitions of simplex, half-duplex, full-duplex
- What is meant by RS-232 and the need for voltage conversion between RS-232 and digital levels
- PIC18Fxx2 to PC serial port interfacing