Lecture #11

Serial Ports

18-348 Embedded System Engineering

Philip Koopman

Wednesday, 17-February-2016

Electrical & Computer ENGINEERING

Carnegie Mellon

High Tech Hospital Beds

- **Typical features:**
  - Move from flat bed to sitting for meals
  - In-bed scale
  - Massage capability for bed sores
  - Inflatable bladder for bed sores
  - Power+network for equipment attached to bed
  - Battery backup for patient transport with equipment attached

[http://www.bedtechs.com/pdf/H.R.TotalCare.pdf]

- **Technology inside the bed:**
  - Serial data transmission
  - Controller Area Network (CAN) via a 16-bit microcontroller
  - Link from bed to nurse station (wired; wireless)

Where Are We Now?

- **Where we've been:**
  - Memory bus (back to hardware for a lecture)
  - Economics / general optimization
- **Where we're going today:**
  - Serial ports
- **Where we're going next:**
  - Exam #1 Wed 24-Feb-2016
    - See course web page for material included
    - Bring a single two-sided letter size notes sheet in your own handwriting
    - NO calculators
    - We will provide the HC12 reference guide at the test (the "short version" of instruction descriptions, XB encoding table, etc.)
      - All 32 pages -- please do not mark on it since we re-use from year to year
  - Second half of course: timers, interrupts, real time operation, I/O, …

Preview

- **Sending digital data**
  - How bits go on a wire
  - RS-232 serial communications
- **Getting serial devices to talk**
  - RS-232 signal and control lines
  - SCI control and data registers
  - Some other serial protocols (RS-485, I^2C, SPI, USB)
- **Error detection codes**
  - Data on wires is subject to corruption due to noise
  - It is very common for designers to get this stuff wrong, or grossly suboptimal

How Do You Send Digital Data?

- **Bit Serial Communication**
  - To send N bits of data, perform N sequential one-bit data transfers
  - Alternative is “parallel” – send multiple bits at a time
    - Printers used to send 8 bits at a time (“parallel printer port”)…
    - …but with USB, even they are bit serial now
- **One wire for data bits costs less than multiple wires**
  - Less cost for materials (copper); thinner; lighter
  - Only need one copy of high-speed bit handling electronics, not 8 (or more)
  - Minimizes problems with bit skew
    - If you have 8 data lines, data value edges arrive at slightly different times
    - If you need to leave extra time for edges to settle, it slows things down

Bit Serial Communication Used on Different Scales

- **Desktop systems – bit serial communication via Ethernet, wireless, etc.**
- **Multi-processor embedded systems:**
  - Special real-time communication networks between processors (e.g., CAN bus)
  - Extensive look at this in 18-649
- **Single-processor embedded systems:**
  - Communicating with outside world (e.g., “diagnostic” or “service” port)
  - Communicating with some peripherals (e.g., LCD, keyboard, mouse, modem)
  - Communicating with mass storage (e.g., flash memory)
- **We’re going to look at a basic bit serial protocol – RS-232**
  - RS-232C Standard from 1969 – some desktop PCs still have a serial port today!
    - They are prevalent in embedded systems, and won’t go away any time soon
  - Gets the job done reliably and at low cost
    - Once you understand this, most serial transfer schemes are not all that different
  - Fancier stuff can be found in 18-649

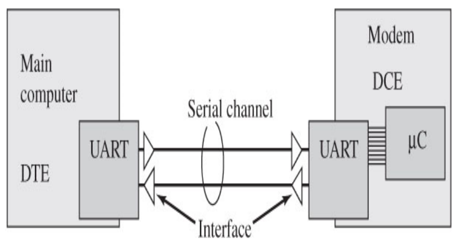
Serial Communication Terminology (RS-232)

- **UART does the serial communication in hardware**
  - Universal Asynchronous Receiver/Transmitter
    - a.k.a. ACIA (Asynch. Communications Interface Adapter)
    - a.k.a. SCI (Serial Communications Interface)
- **From the days of teletypes & computer "terminals"**
  - DTE – Data Terminal Equipment (a terminal)
  - DCE – Data Communication Equipment (a phone modem)

Teletype model 33-ASR

[wikipedia]

Figure 7.1
A serial channel connects a DTE to a DCE.

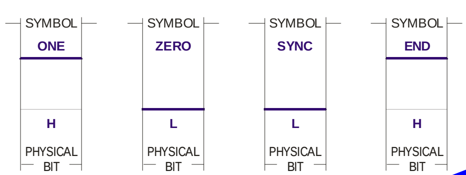


[Valvano]

Non-Return to Zero (NRZ) Encoding

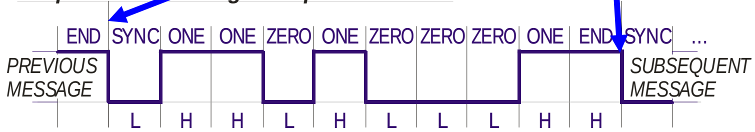
- **Example: Send a Zero as LO; send One as HI**
  - Worst case can have all zero or all one in a message – no edges in data
  - Simplest solution is to limit data length to perhaps 8 bits
    - SYNC and END are opposite values, guaranteeing two edges per message
    - This is the technique commonly used on computer serial ports / UARTs
  - Bandwidth is one edge per bit

Simple NRZ Bit Encoding



Guaranteed falling edge at start of message (high END to low SYNC)

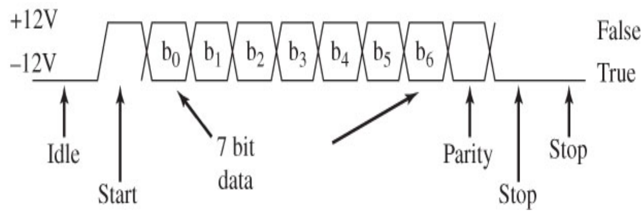
Simple NRZ Encoding Example: 1101 0001



RS-232 Signals

- **NRZ bits**
  - Note: typically +/- 12V, not 5V! – requires level shifting interface chip
    - (5V is acceptable within the standard, but is not the default value)
  - That's a main reason why there are 12V outputs on PC-104 bus!
  - Mapping to data is a little strange: -12V is "true=1" +12V is "false=0"

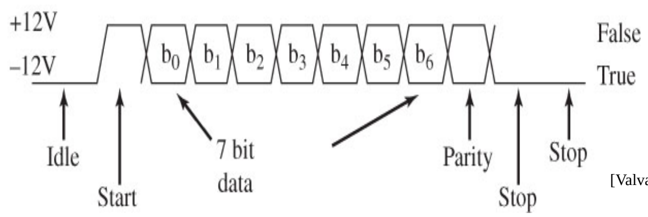
Figure 7.2
A RS232 frame showing one start, seven data, one parity, and two stop bits. [Valvano]



- **Start – "This is the start of a message"**
  - Always +12V ("0")
  - Always one bit from either idle or stop
  - Rising edge of start bit provides timing point for subsequent bits

RS-232 Signals – Continued

Figure 7.2
A RS232 frame showing one start, seven data, one parity, and two stop bits.



[Valvano]

- **Stop – End of Message**
  - Always -12V ("1")
  - One or more "stop bits" to give processing time between bytes
    - For mechanical systems, gives time to actually print a character on paper
  - No real difference between "idle" and "stop" other than how long they last
    - Except that there is a guaranteed minimum number of stop bits after each character sent

- **Data – The Actual Bits**
  - Either high or low depending on value
  - Can be 5, 6, 7, 8, or 9 bits
    - (5 bits for very old printers that only used capital letters – such as some teletypes)

RS-232 Signals – Continued

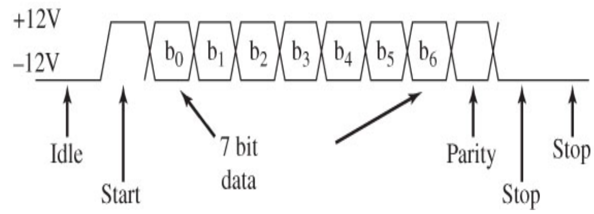


Figure 7.2
A RS232 frame showing one start, seven data, one parity, and two stop bits.

[Valvano]

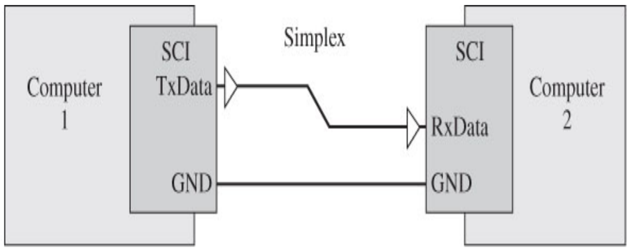
- **Parity**
  - Simple error detection
  - "Even Parity" – parity bit is 0 if parity of data is 0 (=xor of data bits)
  - "Odd Parity" – parity bit is 1 if parity of data is 0 (=inverse of xor of data bits)

- **Today, values are almost always:**
  - 1 start bit
  - 8 data bits
  - 1 stop bit
  - no parity (use CRC on message, not per-byte parity)
  - Both sender and receiver usually know the settings in advance

What Wires Are Involved?

- **Simplex – One direction of transmission (either input OR output)**

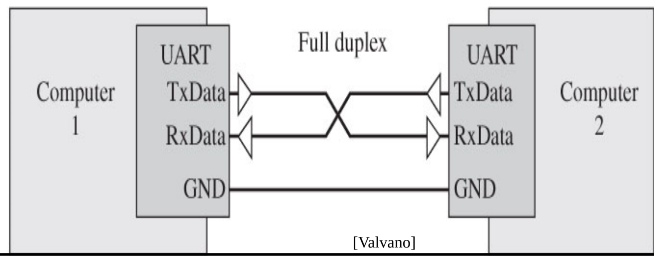
Figure 7.7
A simplex serial channel between two computers.



[Valvano]

- **Full Duplex – Simultaneous two-direction transmission**

Figure 7.3
A full-duplex serial channel connects two DTEs (computers).

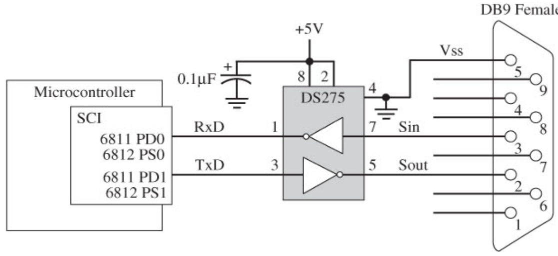


[Valvano]

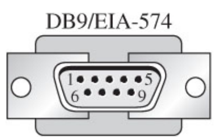
9-Pin Serial Connector (DB9)

- **For pin numbers, always check if the numbering is:**
  - For male or female
  - For front (connector side) or back (solder side)

Figure 7.12
RS232 interface to Freescale microcomputers.



[Valvano]



[Valvano]

A male DE-9 serial port on a PC style computer.

[wikipedia]

How Many Bits Per Second

- **Often bit time is power of two times 300 bits per second:**
  - 300 bps   (teletype)
  - 600 bps
  - 1200 bps   (first generation “fast” modem)
  - … 9600   (common default serial port speed on PCs)
  - ... 57,600 …   (if you are lucky via a telephone phone modem)
  - Set using a frequency divider from the CPU’s crystal oscillator

- **These “bits” include start bit, stop bit, parity, etc. => raw data rate**
  - Actual data rate is slower (e.g., 8 data bits per 10 raw bits)

- **Receiver and transmitter have to have the same oscillator speed**
  - AND have to be set at the same baud rate (e.g., 1200 bps)
  - AND have same start, stop, parity bit settings

- **Sometimes you hear “56K baud” or “9600 baud” etc.**
  - Baud is “symbols per second”
  - For RS-232, bps and baud happen to be the same number
  - For other methods, bits/sec might be faster or slower than symbols/second

Bit Timing – Transmit

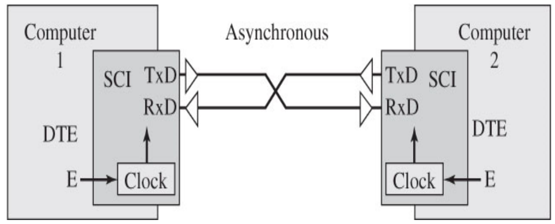


Figure 7.8
Nodes on an asynchronous channel run at the same frequency but have separate clocks.

[Valvano]

- **Separate Transmit and Receive clocks determine bit length**
  - This is "asynchronous" – no clock signal on the communication line!
  - Clock runs 16 times faster than bit rate
  - Every 16 TxClk cycles, move to the next bit being transmitted

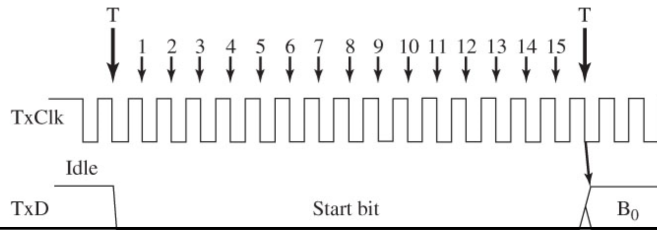


Figure 7.28
Start bit timing during a transmit data frame.

[Valvano]

Bit Timing – Receive

- **Receiver doesn't "know" when the bits start**
  - There isn't a clock signal on the lines
  - Must recover bit edge information from the bits themselves
  - Approach: "Start" of first bit is falling edge of Start Bit
  - Measure other bits 8 clocks into their assumed bit time (every 16 clocks)
  - Hope that the RxClk *doesn't drift too much* compared to TxClk

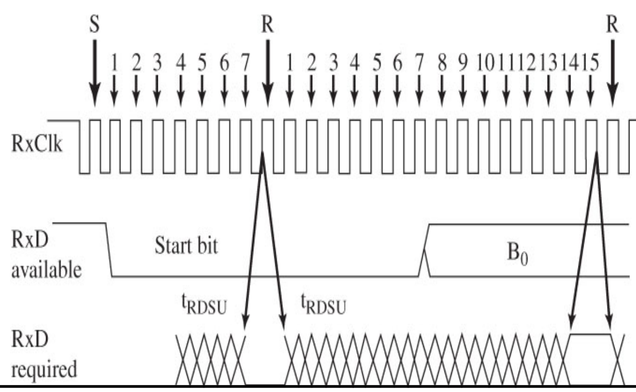


Figure 7.31
Start bit timing during a receive data frame.

[Valvano]

Control Flow

- **How do you know the receiver is ready?**
  - Simplest option: blast bits full speed and hope nothing gets dropped
  - This can (sometimes) work at 300 bps; less reliable at high bit speeds
- **Hardware flow control – byte at a time**
  - "RTS" – I'm ready to send bits. Please let me know when you're ready to received
  - "CTS" – OK, I'm ready to receive bits – send them!
  - CTS stays active as long as the receiver is OK to go…
    … or, CTS goes high after every byte, then goes low again for the next byte
  - Optionally used to make sure CPU can get byte out of input buffer in time
    - Most useful for very fast data being received by very slow device
- **Software flow control – message at a time**
  - "XON" – ($11) OK, I'm ready to receive the next message
  - "XOFF" – ($13) Wait; I can't receive any messages for a while
  - Optionally used to make sure CPU empties message buffer in time

The Rest Of The Pins

- **Remember, this was originally for modems and terminals!**
  - "Data terminal" is the embedded computer (the "teletype")
  - "Data Set" is the device you are controlling (the "modem")
  - Usually the only other control signals are "RTS" and "CTS"
    - (see next slide)
  - Note: 25 pin serial connector is obsolete; 9-pin connector still in wide use

9-pin	25-pin	pin definition
1	8	DCD (Data Carrier Detect)<PC-input>
2	3	RX (Receive Data)<PC-input>
3	2	TX (Transmit Data)<PC-output>
4	20	DTR (Data Terminal Ready)<PC-output>
5	7	GND (Signal Ground)<RefZeroVolts>
6	6	DSR (Data Set Ready)<PC-input>
7	4	RTS (Request To Send)<PC-output>
8	5	CTS (Clear To Send))<PC-input>
9	22	RI (Ring Indicator)<PC-input>

See:
http://en.wikibooks.org/wiki/Serial_Programming:RS-232_Connections#Wiring_Pins_Explained

Cabling

- **Connecting two computers**
  - A Modem (DCE) knows that the "transmit" pin is incoming data
    - Similarly, RTS/CTS are backward on the DCE side
  - But, both computers think "transmit" is outgoing!

A null modem adapter

[wikipedia]

  - Solution: "null modem" or use a crossover cable
    - Crosses over TD and RD
    - Crosses over RTS and CTS
    - (These are the four important signals I expect you to know!)

Signal Name	DB-25 Pin	DB-9 Pin		DB-9 Pin	DB-25 Pin	
FG (Frame Ground)	1	-	X	-	1	FG
TD (Transmit Data)	2	3	-	2	3	RD
RD (Receive Data)	3	2	-	3	2	TD
RTS (Request To Send)	4	7	-	8	5	CTS
CTS (Clear To Send)	5	8	-	7	4	RTS
SG (Signal Ground)	7	5	-	5	7	SG
DSR (Data Set Ready)	6	6	-	4	20	DTR
CD (Carrier Detect)	8	1	-	4	20	DTR
DTR (Data Terminal Ready)	20	4	-	1	8	CD
DTR (Data Terminal Ready)	20	4	-	6	6	DSR

[wikipedia]

- **Faking Out RTS/CTS**
  - Connect RTS to CTS at the connector
  - Hardware at other end had better be ready!

SCI – Serial Communication Interface

- **The SCI has a memory-mapped interface**
  - Control information AND
  - Actual data being read/written

[Freescale]

Table 1-1. Device Register Map Overview

Address	Module	Size
0x0000–0x0017	Core (ports A, B, E, modes, inits, test)	24
0x00A0–0x00C7	Reserved	40
0x00C8–0x00CF	Serial communications interface (SCI)	8
0x00D0–0x00D7	Reserved	8

  - Addresses below are offsets from base address (i.e., 0x00C8.. 0x00CF)
    - Why this address range (what's special about addresses with top 8 bits = 0?)
  - See chapter 13 of MC9S12 data sheet for details

	Address	Name		Bit 7	6	5	4	3	2	1	Bit 0
0x00C8	0x0000	SCIBDH	R	0	0	0	SBR12	SBR11	SBR10	SBR9	SBR8
			W								
0x00C9	0x0001	SCIBDL	R/W	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
0x00CA	0x0002	SCICR1	R/W	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT
0x00CB	0x0003	SCICR2	R/W	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
0x00CC	0x0004	SCISR1	R	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
			W								
0x00CD	0x0005	SCISR2	R	0	0	0	0	0	BRK13	TXDIR	RAF
			W								
0x00CE	0x0006	SCIDRH	R	R8	T8	0	0	0	0	0	0
			W								
0x00CF	0x0007	SCIDRL	R	R7	R6	R5	R4	R3	R2	R1	R0
			W	T7	T6	T5	T4	T3	T2	T1	T0

= Unimplemented or Reserved

Figure 13-2. SCI Register Summary

[Freescale]

Setting Baud Rate

- **SBR – Select Baud Rate (13 bit *integer* value)**
  - Sets clock divider to change bit rate (divides from module clock)
  - Receiver clock is 16x Transmitter Clock
    - Receiver clock cycles 16 times per bit – looks at multiple samples per bit
    - Transmitter clock cycles 1 time per bit (just need clock at each bit edge)
  - example: SBR value of 326 sends at ~4800 Hz
    - Caution – table below at 25 MHz. Course module will be running at 8 MHz
      - » (Note: runs at 2 MHz out of the box, but we're providing code to increase to 8 MHz)

SCI baud rate = SCI module clock / (16 * SCIBR[12:0])

Table 13-10. Baud Rates (Example: Module Clock = 25 MHz)

Bits SBR[12-0]	Receiver Clock (Hz)	Transmitter Clock (Hz)	Target Baud Rate	Error (%)
41	609,756.1	38,109.8	38,400	.76
81	308,642.0	19,290.1	19,200	.47
163	153,374.2	9585.9	9600	.16
326	76,687.1	4792.9	4800	.15
651	38,402.5	2400.2	2400	.01
1302	19,201.2	1200.1	1200	.01
2604	9600.6	600.0	600	.00
5208	4800.0	300.0	300	.00

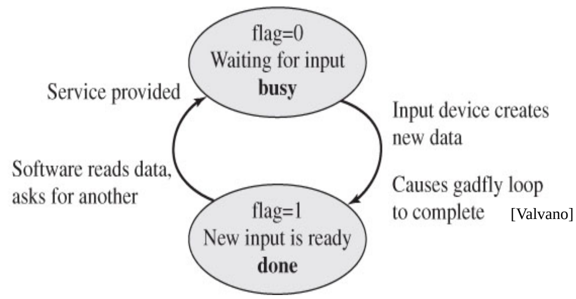
[Freescale]

Other Control & Data Registers

- **SCI Control Registers (SCICR1; SCICR2)**
  - Set start, stop, data bit configuration
  - Set parity configuration
  - Enable transmit and receive
- **SCI Status Registers (SCISR1; SCISR2)**
  - Has data been received?
  - Has an error occurred (e.g., parity error)
  - **RDRF** = "Receive Data Register Full" → A data byte has arrived
  - **TDRE** = "Transmit Data Register Empty" → Ready for the next byte to write
- **Data Registers (SCIDRL)**
  - Read to receive a byte
  - Write to send a byte
- **Software reads/writes registers as if they were memory locations**
  - What C keyword is important to make sure optimizer doesn't omit reads or writes?

Polled (“gadfly”) Data Reading

Figure 3.1
The input device sets a flag when it has new data.



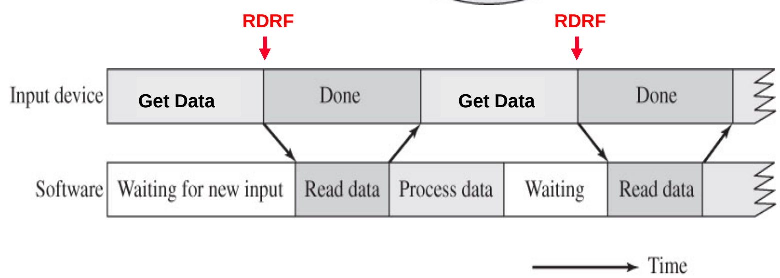
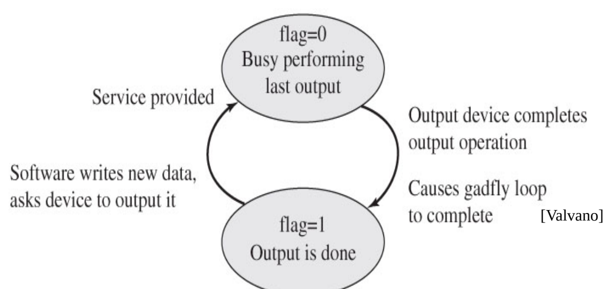


Figure 3.2
The software must wait for the input device to be ready.

Polled (“gadfly”) Data Writing

Figure 3.3
The output device sets a flag when it has finished outputting the last data.



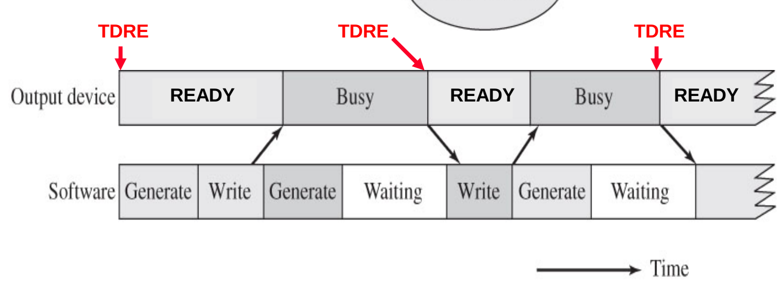
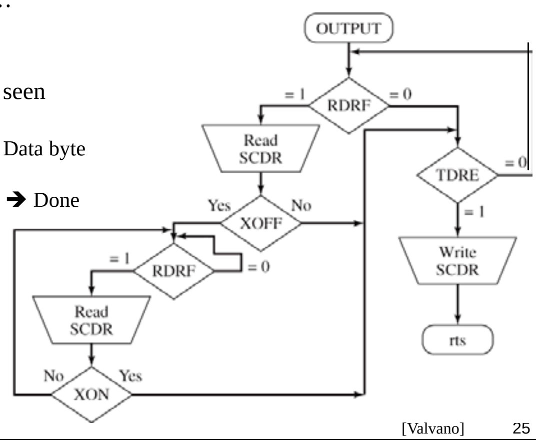


Figure 3.4
The software must wait for the output device to finish the previous operation.

Polled SCI operation

- **Simplest way to do serial data communication**
  - Use a loop to transmit bytes as soon as they can be sent
  - Use a loop to receive bytes, waiting for the next one
  - Combined loop below:
    - Receives a byte if ready…
      else transmits a byte if it can…
      else goes back to trying
      to receive
    - Inhibits transmit when XOFF seen

RDRF = "Receive Data Register Full" ➔ Data byte arrived
TDRE = "Transmit Data Register Empty" ➔ Done sending
SCDR = "Serial Comms. Data Register"
XON/XOR ➔ Flow Control



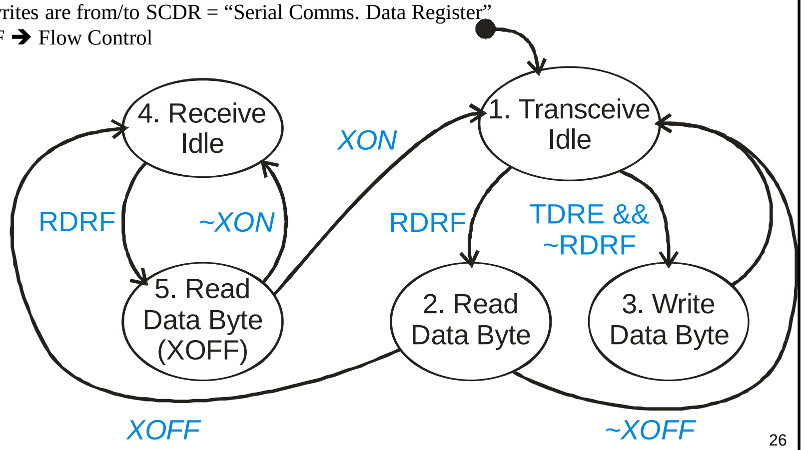
Is it easier to understand this flowchart or statechart on next page?

[Valvano]

Polled SCI Operation

- **Assumes infinite amount of data to be written**
  - Implements XON/XOFF – State 4 inhibits transmit until XON received
  - When in Transceive Idle state, gives priority to reading

RDRF = "Receive Data Register Full" ➔ Data byte arrived
TDRE = "Transmit Data Register Empty" ➔ Done sending
Data read/writes are from/to SCDR = "Serial Comms. Data Register"
XON/XOFF ➔ Flow Control



Framing Messages

- **How do you know how many bytes to receive?**
  - Similar problem to string handling
    - C solves with a null byte termination
    - Other languages solve with a count before the string
    - Sometimes all strings in system are exactly the same length to make it simple
    - Both approaches have strengths and weaknesses

- **Usual serial message components**
  - Header info – what type of message is this?
  - [optional] – count of how many bytes to expect
  - Payload – the actual data you care about
  - Error detection – something beyond parity to detect corrupted bytes

  - Each message might also be sandwiched between an XON and XOFF

Buffering Messages

- **For XON/XOFF to work, you need a message buffer**
  - Most messages are more than one byte
  - Receive entire message, then pass to application software

  - General idea:

```
// receive a message
char ibuf[80];  // input buffer
uint8 rcv_count = 0;
Transmit XON; // Ready to receive a buffer full of data
while ( still bytes remaining in message )
{   wait for input byte to be ready;
      ibuf[rcv_count++] = input_byte;
     …handle case that rcv_count overflows ibuf size;
}
// result is in ibuf, and rcv_count says how many bytes
Transmit XOFF;  // Hold off any more incoming data
```

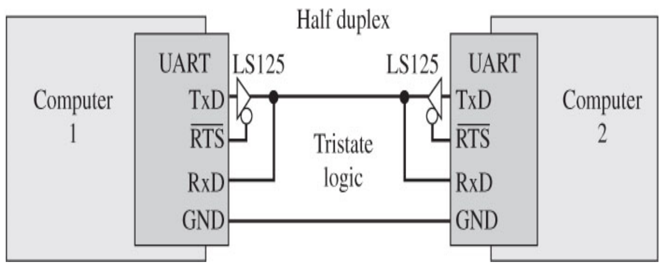
Multi-Drop Serial Connections

- **What if you want to connect 3 or more points to form a network?**
  - Usually don't want N data wires for N points – want to share a single data cable
  - Start with N=2; "half duplex"
  - Then add better physical layer (next slide), then combine ideas (coming up soon)

- **Half duplex RS-232: only one side can transmit at a time**
  - A single data line (reduces wiring cost – 2 wires instead of 3)
  - Tristate drivers to avoid conflicts
  - Software must keep straight who is the transmitter

Figure 7.4
A half-duplex serial channel can be implemented with tristate logic.

[Valvano]

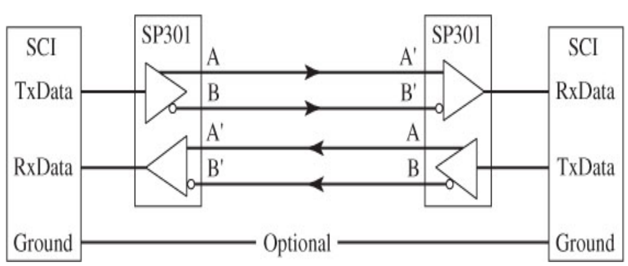


RS-422 Differential Data Transmission

- **Differential drivers (RS-422 serial channel)**
  - Transmit both data.H and data.L at same time
  - Receiver looks at difference, not absolute voltage
  - Gives common mode noise rejection
  - Higher bit rates (up to 10 Mbits/sec)
  - Typically 5V operation, not 12 V

Figure 7.16
RS422 serial channel.

[Valvano]



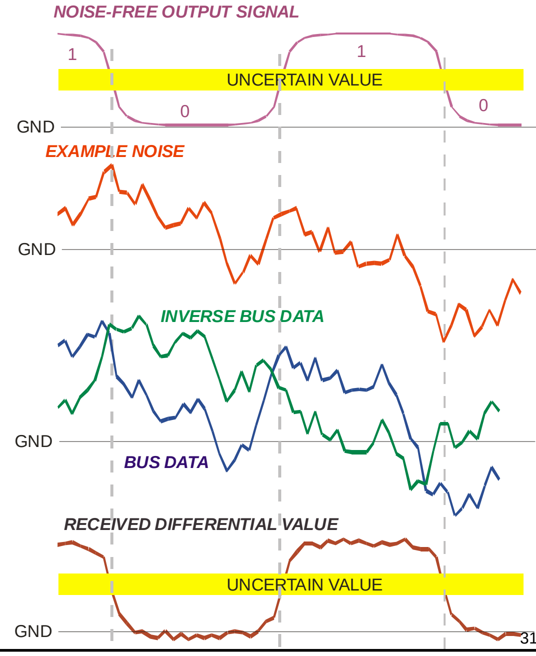
Differential Drivers Suppress Noise

- **Send both Data and Inverse Data values on a 2-wire bus**
  - Example:

    DATA   HI = 5 volts
            LO = 0 volts

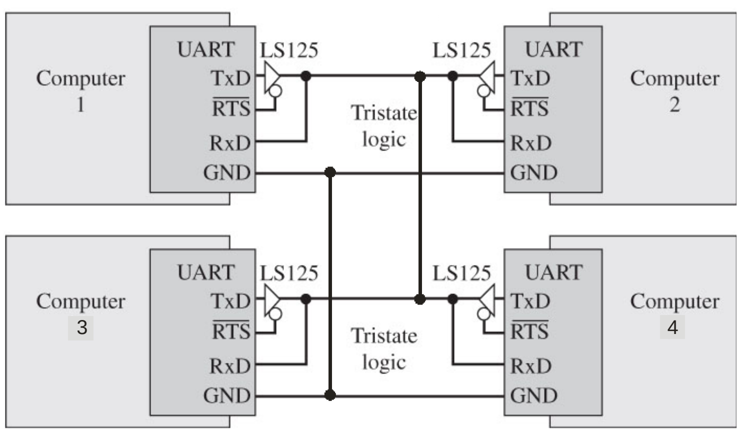
    Inverse DATA
            HI = 0 volts
            LO = 5 volts

  - Receiver subtracts two voltages
    - Eliminates common mode voltage bias
    - Leaves any noise that affects lines differently



Multi-Drop Serial Transmission

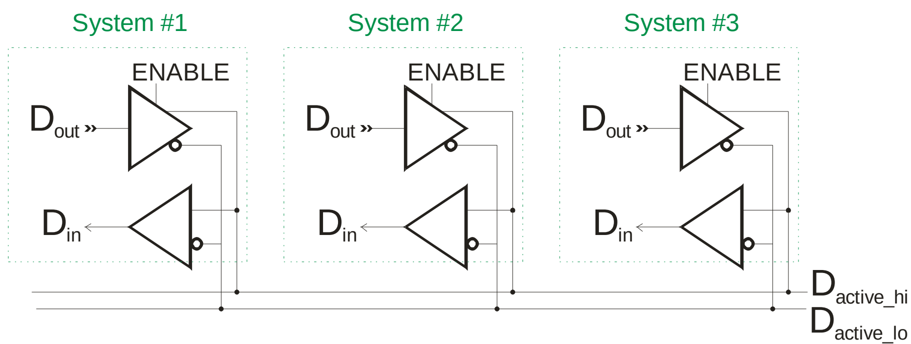
- **Let's go back to RS-232 half duplex**
  - You could hook up as many nodes as you want
  - Just make sure only one node transmits at a time



[Valvano]++

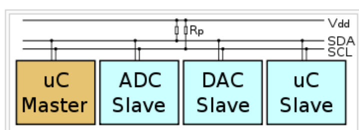
RS-485 Is A Common Multi-Master Bus

- **Used in industrial control networks (e.g., Modbus; Profibus)**
  - RS-422 differential drivers; high speed + good range (10 Mb/s @ 12 meters)
  - Multi-drop approach like RS-232 on previous slide
  - Add terminators to reduce noise
  - Make sure that exactly one system has its output enabled at a time!
    - How exactly you do this is covered in 18-649
    - Often it is “master/slave” – one system tells each other system when its turn comes



I²C Bus – (Inter Integrated Circuit Bus)

- **Multi-master serial bus for short distances**
  - Typically on the <u>same circuit board</u>
    - SMBus is a subset of I²C for interoperability
  - Often runs 10K bps to 100K bps; 3.3-5V DC
  - SDA – Serial Data
  - SCL – Serial Clock (gives clock edges for data)
    - Simplifies receiver; extra wire is almost “free” on a circuit board



A sample schematic with one master (a microcontroller), three slave nodes (an ADC, a DAC, and a microcontroller), and pull-up resistors R_p

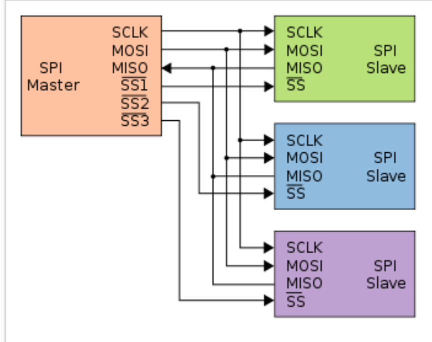
[Wikipedia]

- **Each master node can run the bus (one at a time!)**
  - Master sends data to slave
  - Slave potentially sends data back to master

- **Master/slave polling:**
  - Master sends start bit + 7-bit address + read/write
  - Slave either listens (write) to data from master or sends (read) to master
  - When bus is idle, a different master can take over transmission
    - If they collide, they arbitrate on slave address (lowest address gets to send)
    - Often high bits of slave address pre-set by device type; low bits via input pins

SPI – Serial Peripheral Interface Bus

- **Higher speed short range bus**
  - Higher speed than I^{2}C – 8 MHz+
  - Typically connects devices on same circuit board
  - Simple slave hardware interface



Typical SPI bus: master and three independent slaves

[Wikipedia]

- **Single Master design**
  - Four wires: clock, data in, data out, slave select (slave enable lines)
  - Master device initiates reads or writes to one or more slave devices
  - Full duplex (input and output can run concurrently)
  - Synchronous bus – separate clock line rather than self-clocking data

USB – Universal Serial Bus

- **Very high speed medium range bus**
  - Originally to connect PC peripherals
  - Typically 3-10 foot cables, Half-duplex differential signals
  - 0V / 3.5V for low speed (1.5 Mbit/s) and full speed (12 Mbit/sec)
    - High speed of 480 Mbit/sec for USB 2.0
  - Cables can connect via hubs
  - Can supply 5V power to peripheral
    (500 mA in USB-2 ➔ which might not be enough for your proto-board!)

- **Single Master design**
  - Data in packets with PID (Packet Identifier) to determine type of packet
  - Versions 1 & 2 were master/slave polling
  - Much more complex protocol than others described…
    … so complicated that Wikipedia doesn't have a simple picture for it!
    … so complicated that to implement it you pretty much dedicated a small CPU
  - Example: SMSC USB3300-EZK USB 2.0 controller
    - $1.28 apiece in 500 quantity from Digi-Key as of 2012

Many Other More Complex Protocols

- **CAN – Control Area Network**
  - Main high speed data bus on cars and many other systems
  - Optimized for short real-time control messages (8-byte payload)
  - Up to 1 Mbps on truck-size vehicles
  - We'll talk about that in a later lecture

- **FlexRay**
  - Next-generation automotive network
  - Optimized for safety-critical high speed control
  - Up to 10 Mbps on vehicles
  - Fault tolerant and guaranteed real-time features

- **"Fieldbus" networks**
  - This is a generic term for embedded networks of many different types
  - Often *not* based on Ethernet due to cost and real time concerns
  - Much more in 18-649

What About Error Coding?

- **Noise on serial buses is a fact of life**
  - In embedded systems, can easily be one bit error per 10^5 (or 10^6) bits
    - Does that matter?
  - At 9600 bps x 24 hours
    - 86,400 seconds/day; 829,440,000 bits per day ➔ ~8300 errors per day
  - CAN (serial network in cars) might run at 1Mbps ➔ ~ 1 million errors/day
    - Many will be single-bit errors, but many others will be multi-bit errors.

- **Is parity enough?**
  - Detects all odd number of bit errors
  - Parity on 8 bits is good at catching single bit upsets…
  - BUT, it costs too much (~10% bandwidth penalty)
  - AND, it is only a 50/50 shot to catch multi-bit upsets and bursts of noise

- **Want a more general approach**
  - In case a noise burst creates multiple bit errors close together
  - In case network has periods of high noise, or otherwise sees many errors
  - For example …. checksums (remember that?)
    - But can do even better using more sophisticated error detection codes .. CRCs

Review

- **Sending digital data**
  - How do bits go on a wire?
    - NRZ, start, stop, parity, idle, receive clock
- **Getting serial devices to talk**
  - RS-232 serial communications
    - Data pins, types of control flow, RTS/CTS, why a crossover cable
    - BUT NOT memorizing pin numbers; not obscure control pins
  - From lab:
    - SCI control and data registers, by general name
    - “What does RDRF do?” BUT NOT “What does bit 3 of SCISR1 do?”
  - General understanding of other multi-master buses discussed
    - E.g., differences among RS-232, RS-422, RS-485

Lab Skills

- **Get a serial port to operated**
  - Send data to a test program on a PC
  - Received data from a test program on a PC