Serial I/O

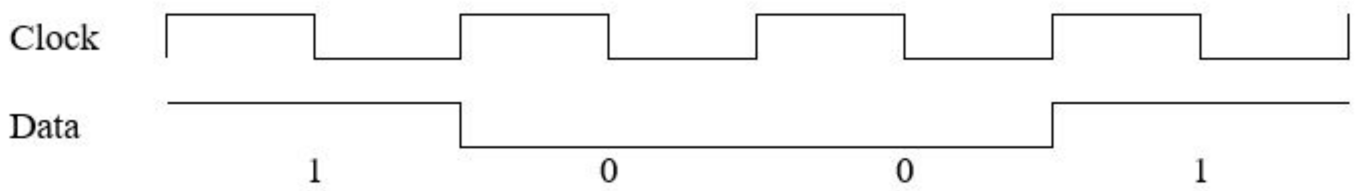
One thing after another

Overview

- An n-bit value can be transmitted serially, meaning one bit at a time (over n clock cycles).
- The alternative is to transmit in parallel (discussed and compared later).
- Serial communication is accomplished using Transmit (Tx) and Receive (Rx) shift registers.

Overview

- E.g. illustrate transmission of 4-bit value assuming a shared clock signal.



- The rate of serial transmission is measured in bps (bits per second) – and often in kbps and Mbps.

Overview

- Exercise: assuming a serial connection between a keyboard and a computer, what would be an appropriate minimum transfer rate?
- What is the average typing speed?
- Average of 50 – 70 wpm (words per minute):
- With an average o f 5 characters per word this is:
- (70 * 5) / 60 sec = 350 / 60 = 6 characters / sec
- Since each character has ~ 8 bits / character
  - this is 6 * 8 = 48 bits / sec
- See http://en.wikipedia.org/wiki/Words_per_minute

Terms

- Baud rate
  - Is a signalling rate (i.e. number of symbols transmitted per second)
  - Not the same as *data rate* (Except if the only signals are 0, and 1)
    - Baud rate is more general than data rate
- Duplex
  - Simultaneous bidirectional communication
  - Requires two data lines

Terms

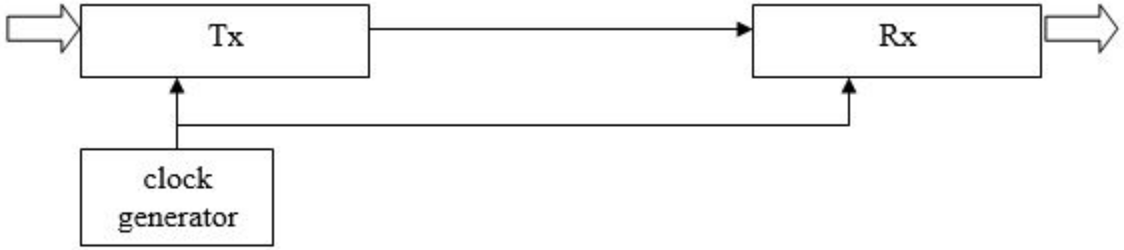
- Half-Duplex
  - Non-simultaneous bidirectional communication
  - Requires one data line
- Simplex
  - Unidirectional communication
  - Requires one data line

Synchronous Transmission

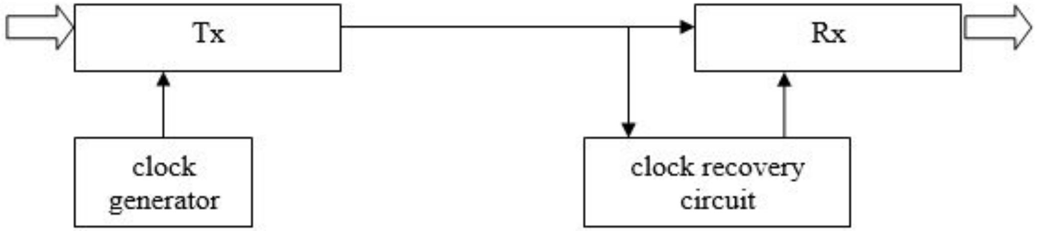
- Serial communication may be synchronous or asynchronous
- In synchronous, transmission between sender and receiver is synchronized using a common clock
- In synchronous communications, bits are sent continuously
    - If the sender has nothing meaningful to send, bits are still sent
    - There must be a communications protocol so that the receiver can distinguish meaningful data from “pad” data
    - The protocol may include transmission of synchronization or resynchronization data

Synchronous Transmission

- One possible way to synchronize clock over transmission:

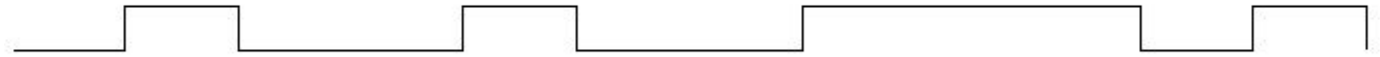


- Another possible way to synchronize clock over transmission:



Synchronous Transmission

- How can the clock signal be recovered from a data stream such as:



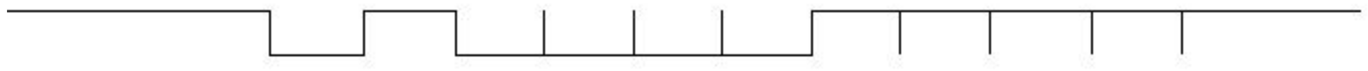
- Idea: the timing of the transitions in the data stream reveal the sender's clock signal sufficiently well
    - Assuming transitions happen often enough
- A full discussion of synchronous communication protocols is beyond the scope of this course

Asynchronous Communication

- In asynchronous serial communication, there is no common clock to synchronize sender and receiver
- Communication is not continuous, but starts and stops
- The start time of a new “frame” of data is not arranged in advance
- When data is not being transmitted, the line is left in an idle state
  - Either low or high but high is most commonly used
- The frames are normally kept short
  - 5 to 9 data bits – 8 is the most common

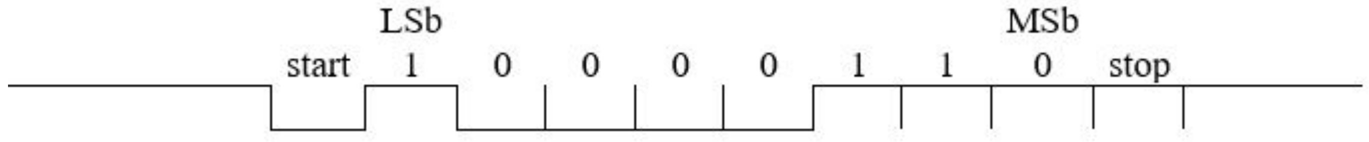
Asynchronous Communication

- Each frame has the following format:
    1. Start bit ← non-idle level to signal start of frame
    2. Data bits ← usually LSB first
    3. Optional parity bit
    4. 1, 1 ½ or 2 stop bits ← return to idle level (minimum duration)
- **Exercise:** assuming high bit for idle, 8 data bits LSB first, no parity bit and 1 stop bit (8N1), what is the value being transmitted below?



Asynchronous Communication

- Labelling the transmission allows for identification of the key elements in the frame:



- First 0, is the start bit (i.e. no longer in idle, high, transmission)
- First bit after the start bit is the LSB
- Next 8 time segments after start bit are the data
  - Data transmitted: `0110 0001 or $61 or` 97_{10} `('a')`

Error Detection - Parity

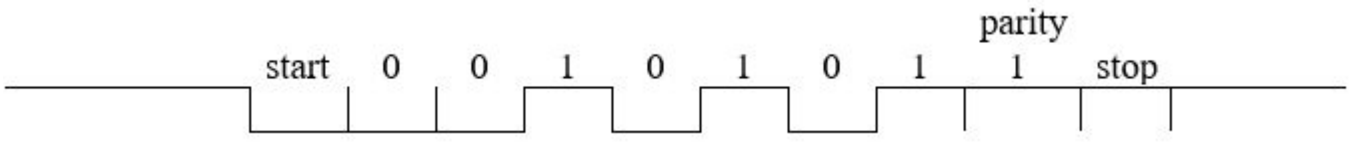
- A full discussion on error detection is transmission is beyond the scope of this course
- Parity uses the number of set bits (i.e. a value of 1) in the original value to allow the receiver to determine if any of changed
    - It is limited to showing if an odd number of bits have changed
        - It is limited so not really used now
    - Does not allow for recovering from a transmission error
    - If parity is being used the parity bit is always inserted

Error Detection - Parity

- “Odd parity” means the sum of the number of set data bits + the parity bit must be odd
    - If the number of 1’s in the data is even the parity bit will be set to 1
    - If the number of 1’s in data is odd the parity bit will be set to 0
- “Even parity” is similar, but the total number of 1’s plus the parity bit is made to be even
    - If the number of 1’s in the data is even the parity bit will be set to 0
    - If the number of 1’s in data is odd the parity bit will be set to 1
- If the receiver detects a parity mismatch, then an error occurred (e.g. due to electromagnetic interference). If not, an error might have occurred.

Error Detection - Parity

- **Exercise:** assuming 7E1, draw a diagram to represent the transmission of ASCII ‘T’
  - ‘T’ == $54 = 0101 0100 (via reference card)
  - Since 7 data bits it becomes 101 0100 as the leading 0 is dropped
  - Using even parity there needs to be an even number of 1’s in the data, including the parity bit
  - Since this data has an odd number of data bits the parity bit will be 1



Transmission Details

- Without a common clock, how does the receiver know when one bit ends and the next begins?
- The receiver uses its own high frequency clock to sample the data stream
  - e.g. at 16× the agreed-upon baud rate

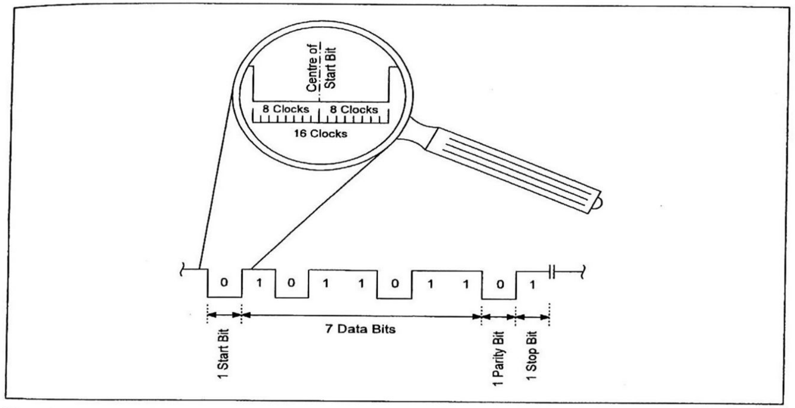


Figure 32.22: Detecting the start bit. The bits are sampled at a frequency 16 times the baud rate to enhance the reliability of data detection.

Transmission Details

- It can therefore detect the start time of a frame (idle to non-idle transition) within a margin of 1/16th the transmitter clock period
- It can then attempt to sample the value of the first data bit in the middle of it’s transmission period - 16 + 8 clock ticks later
- Subsequent data bits are sampled similarly - every 16 clock ticks
- The sender and receiver clock are not in sync
    - They will drift Error will accumulate at each sample
    - As long as frames are short, the error will not accumulate to the point where bits are being sampled outside their beginning and end times

Transmission Details

- The receiver will re-sync its clock on the start bit

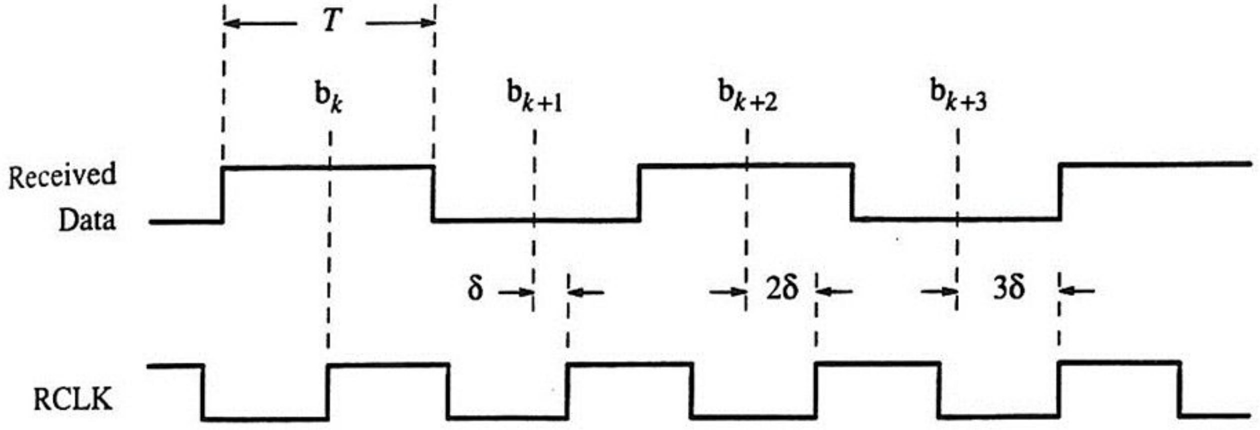


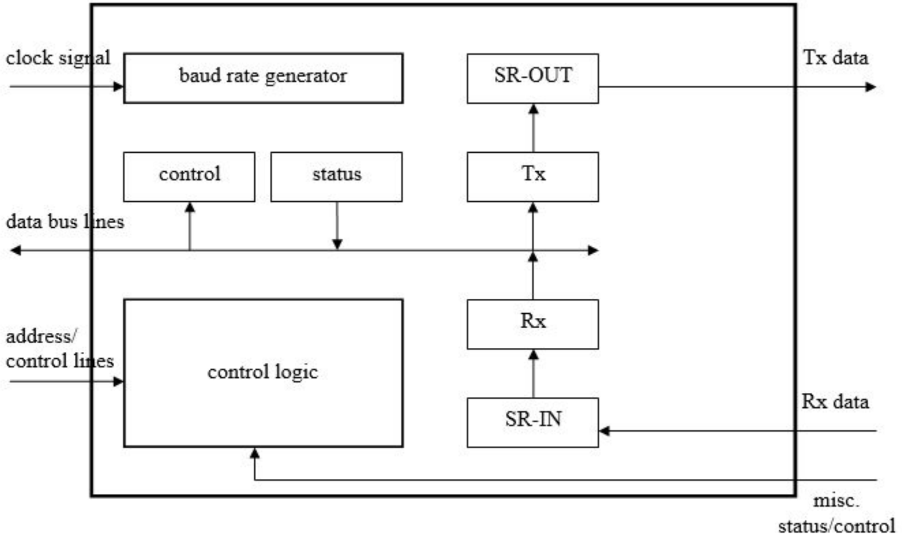
Figure 6.42 Relative timing of data and receiver clock in an asynchronous system.

Simple UART

- A UART (Universal Asynchronous Receiver Transmitter) is an I/O interface for asynchronous serial communications
- “Universal” means that the baud rate is programmable, not fixed
  - There may be a command register whose contents specify by how much to frequency-divide a master clock.
- Typical CPU-accessible registers:
  - Command (control)
  - Status
  - Tx
  - Rx

Simple UART

- Generalized organization and external signals:
  - Note the double buffering (e.g. for transmission, the Tx and SR-OUT registers)



Simple UART - Control Register Bits

- Master clock divide-down for baud rate generation
- Frame format settings (e.g. 8N1 vs. 7E2)
- Rx interrupt enable
- Tx interrupt enable

Simple UART - Status Register Bits

- Tx empty
- Rx full
- Rx overrun (error)
- Parity error
- Framing error ← received stop bit not valid
- Interrupt requested

A Simple UART

- A USART is a similar type of interface, capable of both synchronous and asynchronous operation
- ACIA (Asynchronous Communications Interface Adapter) is another name for a UART
    - The Atari ST has two MC6850 ACIAs: one for managing the serial connection to the keyboard, the other for managing the MIDI port
    - It also has a USART onboard its 68901 chip (discussed later).
- The next tutorial completes the discussion of the MC6850

Serial Line Codes

- Above, it was assumed that:
    - To transmit a 0, a low level is placed on the line
    - To transmit a 1, a high level is placed on the line
- This is the most obvious serial line code, called NRZ (Non Return to Zero)
- There are others, which are sometimes preferable – esp. for synchronous communication
- Why?

Serial Line Codes

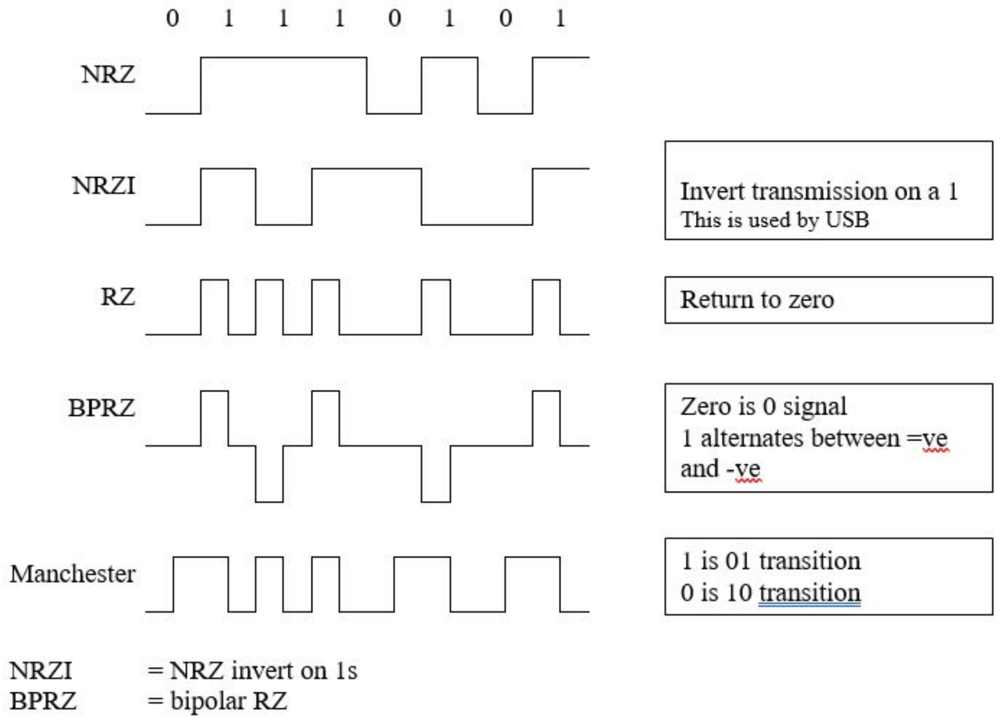
- A digital phase-locked loop (DPLL) is an analog/digital circuit that can be used to recover a clock signal from a serial data stream
    - It only works if there are enough 0-1 or 1-0 transitions in the data to reveal the clock signal
- Using NRZ, a DPLL will fail if too long a stream of 0s or 1s is transmitted
- Why would such a stream (one with a long series of one data value) occur?
    - An image file

Serial Line Codes

- Another reason: some storage media (e.g. magnetic disks) don’t store a bits as absolute 0s and 1s
  - They store the transitions between bit levels. So, NRZ is unsuitable.
- There are a few other reasons why alternatives to NRZ are sometimes used, but they are beyond the scope of this course
  - Example: media characteristics, electrical reasons – distance, repeating
- A full discussion of serial line codes is beyond our scope

Serial Line Codes

- The following chart illustrates some common ones:
  - The point here is to understand that there are many ways to transmit binary values



Serial Line Codes - Tradeoffs

- Some are better for clock recovery than others
  - More transitions
- Some are suitable for transition-sensitive media storage
- Some are “balanced”
  - There is no DC component to bias a receiving circuit
- Some require higher bandwidth due to high frequencies
  - Lots of fast transitions
- See: http://www.interfacebus.com/Definitions.html

RS-232

- RS-232 is a standard for the serial interconnection of two components:
  - a DTE (Data Terminal Equipment) e.g. a computer
  - a DCE (Data Communications Equipment) e.g. a modem
- RS-232 defines a physical interface and a set of signals.

RS-232

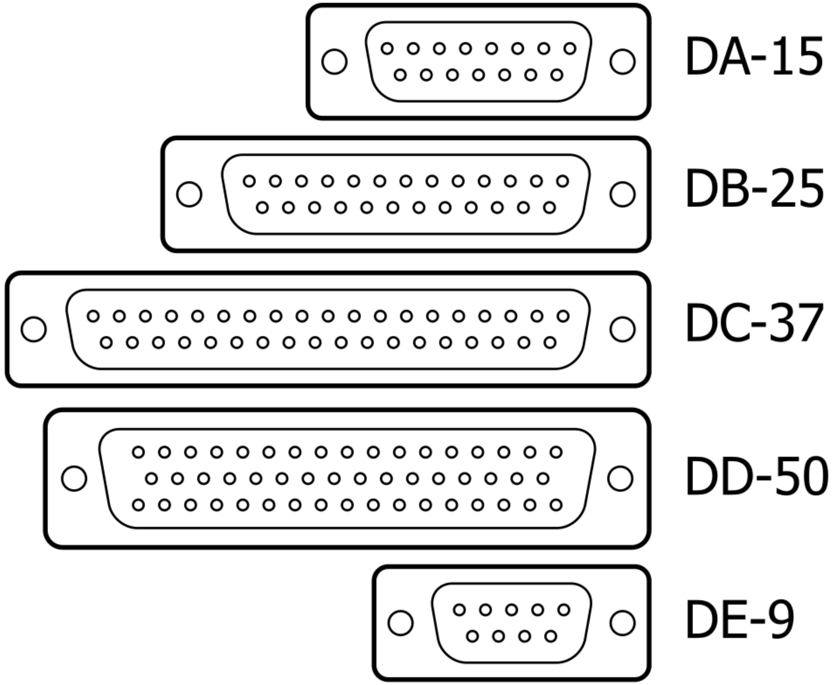
- Nine of the signals are commonly used:

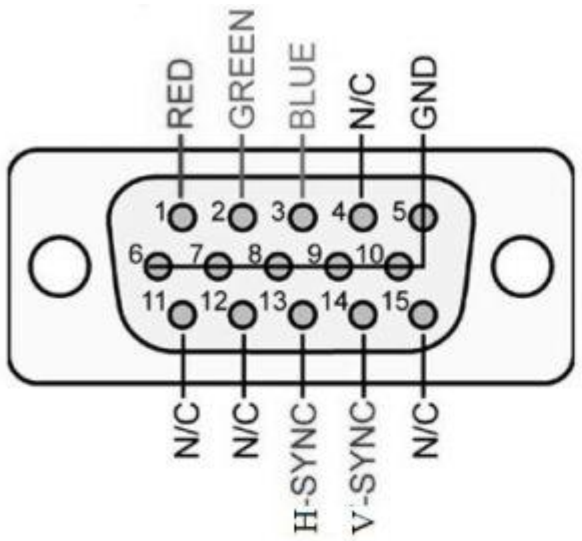
G	common Ground	
TD	Transmit Data	generated by DTE
RD	Receive Data	
DTR	Data Terminal Ready	generated by DTE
DSR	Data Set Ready	
RTS	Request To Send	generated by DTE
CTS	Clear To Send	
DCD	Data Carrier Detect	
RI	Ring Indicator	

RS-232

- A DE-9 port is common, although other variants (including a DB-25 connector) are also allowed.
- D is the shape of the metal shield and B-E are the size
- Although there are several serial communications standards, the term “serial port” usually refers to a 9- or 25-pin RS-232 port
- See http://en.wikipedia.org/wiki/RS-232
- See https://en.wikipedia.org/wiki/D-subminiature

DSubminiature Plug Ends



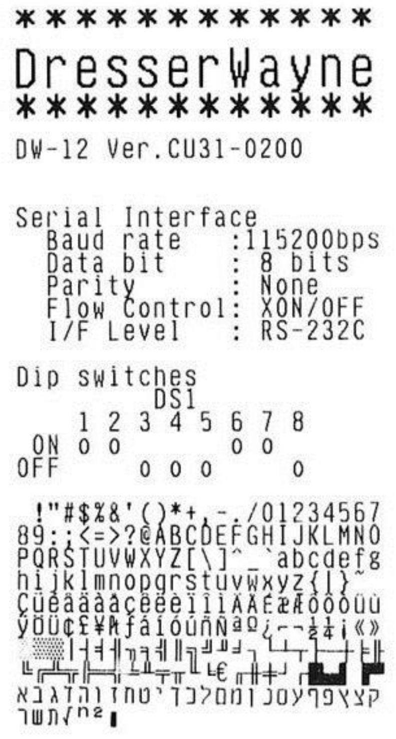


RS-232

- "Null modem" cables can be used to directly connect two computers via their serial ports
- Once very common, RS-232 ports have almost disappeared from personal computers over the past few years
    - In favour of USB, etc
- RS-232 is still common in several settings
    - Communicating with microcontrollers, etc
- UART chips are often designed to be RS-232 compatible
    - i.e. with RS-232 specific pins and functionality

RS-232

- This was “recently” found at store in Calgary when the store opened
- This was printed out as a result of turning on the checkout.
- Cash registers use an RS-232 connection



```
************
DresserWayne
************
DW-12 Ver.CU31-0200

Serial Interface
  Baud rate     :115200bps
  Data bit      : 8 bits
  Parity        : None
  Flow Control: XON/OFF
  I/F Level     : RS-232C

Dip switches
             DS1
      1  2  3  4  5  6  7  8
 ON   o  o              o  o
OFF         o  o  o        o

 !"#$%&'()*+,-./01234567
89:;<=>?@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_`abcdefg
hijklmnopqrstuvwxyz{|}~
ÇüéâäàåçêëèïîìÄÅÉæÆôöòûù
ÿÖÜ¢£¥₧ƒáíóúñÑªº¿⌐¬½¼¡«»
░▒▓│┤╡╢╖╕╣║╗╝╜╛┐└┴┬├─┼╞╟
╚╔╩╦╠═╬╧╨╤╥╙╘€╓╫╪┘┌█▄▌▐▀
אבגדהוזחטיךכלםמןנסעףפץצק
רשת√ⁿ²■
```

USB

- USB is a newer standard for connecting peripherals
    - The latest version is USB 3.1
- On PCs, USB is now used for almost everything except internal drives, graphics and specialty PCI expansion cards
- Compared to RS-232, USB is a very sophisticated standard. A complete study is beyond the scope of this course

USB - Goals of the Specification

- Flexibility and “ease of use”
  - Including multi-peripheral connections, use for user not programmer
- Low cost
- Small, robust connectors
- High speed
  - Fast enough for external storage devices and real-time audio, video and voice
- Dynamic attach / detach
- Bandwidth and latency guarantees

USB

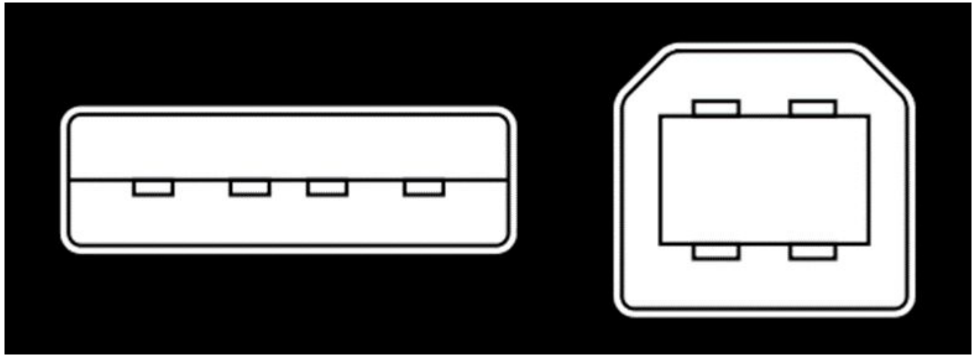
- A USB system is comprised of:
  - A USB host
  - USB devices
  - The USB interconnect
- The USB host is the host computer. The computer has a USB controller – this is the I/O interface to the USB system (the “USB bus”)
- Actually, even though several USB devices can be serviced through one USB controller, PCs often ship with 4-5 controllers

USB

- A USB device is one of:
  - A USB hub, which provides USB device connection points
  - A USB function (e.g. a keyboard, joystick, memory stick, digital camera, ...
- The USB controller has an integrated “root hub”. Therefore, a USB system is arranged as a tree (a “tiered star bus topology”)
- The USB interconnect is the manner in which USB devices are connected to and communicate with the host

USB Connectors

- USB uses 4-pin connectors
    - type A and B shown below – there are also miniature connectors
- Two of the pins carry power
- The other two carry serial data using half-duplex differential signaling
- The cable is shielded, and the data lines are twisted pair



USB

- Data is transmitted serially using NRZI (invert on 0s, with bit-stuffing after five consecutive 1s to ensure sufficient transitions).
  - i.e. communication is synchronous, with clock recovery from the data stream
- The USB bus protocol is sophisticated. A few highlights:
  - Detection of device attach and detach
  - Negotiation for power requirements
  - Negotiation for bandwidth and latency (all communication is scheduled by the host controller)
  - Error detection and self-recovery for lost packets

USB

- The bus is a “polled” bus – the USB controller schedules and initiates all data transfers (again, half-duplex).
- The USB controller repeatedly enumerates the connected devices. Devices are required to be able to identify themselves when attached
- Each device has a “device class”. The set of classes is standard. An O/S is required to provide default drivers for all classes, so that the appropriate driver can be loaded when a device is attached. There is a reserved device class for unique devices which require custom drivers

USB

- An O/S provides a USB controller driver, plus drivers for each device class
  - For ease of use, USB libraries are typically used to provide an additional layer of abstraction.
- USB 2.0 supports a data rate of 480 Mbps!
  - Note: This does not mean that the throughput of an individual device is 480 Mbps
  - The bus is shared with other devices and there is protocol overhead.
  - USB 3.0 supports 5.0Gbps, 3.1 10 Gbps, 3.2 20 Gbps USB4 40 Gbps and v2.0 80Gbps!

USB

- Specs: https://en.wikipedia.org/wiki/USB
- How USB works: https://youtu.be/wdgULBpRoXk