Communication methods for digital systems

- Communication methods
  - wires and signalling conventions used to transmit data between digital devices – we'll only deal with digital communication – other methods include radio frequency (RF), infra-red (IR), freq. modulation (FSK), optical, etc.
- Orthogonal elements of communication methods
  - bandwidth – number of wires
  - speed – bits/bytes/words per second
  - timing methodology – synchronous or asynchronous
  - number of destinations/sources
  - arbitration scheme – daisy-chain, centralized, distributed
  - protocols – provide some guarantees as to correct communication

Bandwidth

- Serial
  - single wire to trasmit information one bit at a time
  - requires synchronization between sender and receiver
  - sometimes includes extra wires for clock and/or handshaking
  - good for inexpensive connections (e.g., terminals)
  - good for long-distance connections (e.g., LANs)
  - examples: RS-232, Ethernet, Apple desktop bus (ADB), Philips inter-integrated circuit bus (I2C), USB, Firewire, IrDA
- Parallel
  - multiple wires to transmit information one byte or word at a time
  - good for high-bandwidth requirements (CPU to disk)
  - more expensive wiring/connectors/current requirements
  - examples: parallel port, SCSI, PCI bus (PC), NuBus (Mac), PCMCIA

Bandwidth

- Issues
  - encoding
  - data transfer rates
  - cost of connectors and wires
  - modularity
  - error detection and/or correction

Speed

- Serial
  - low-speed, cheap connections
    - RS-232 1K–20Kbits/sec, copper wire
  - medium-speed efficient connections
    - I2C 10K-400Kbits/sec, board traces
    - IrDA 9.6K-4Mbits/sec, line-of-sight, 0.5-6.0m
  - high-speed, expensive connections
    - Ethernet 1.5-100Mbytes/sec, twisted-pair or co-axial
- Parallel
  - low-speed, not too wide
    - SCSI bus, 10Mbytes/sec, 8 bits wide
    - NuBus, 40Mbytes/sec, 32 bits wide
    - PCI bus, 250Mbytes/sec, 32 bits wide
  - high-speed, very wide – memory systems in large multi-processors
    - 200M-2Gbytes/sec, 128-256 bits wide

Speed

- Issues
  - length of the wires (attenuation, noise, capacitance)
  - connectors (conductors and/or transducers)
  - environment (RF/IR interference, noise)
  - current switching (spikes on supply voltages)
  - number and types of wires (cost of connectors, cross-talk)
  - flow-control (if communicating device can't keep up)

Timing methodology

- Asynchronous
  - less wires (no clock)
  - no skew concerns
  - synchronization overhead
  - appropriate for loosely-coupled systems (CPU and peripherals)
  - common in serial schemes
- Synchronous
  - clock wires and skew concerns
  - no synchronization overhead
  - can be high-speed if delays are small and can be controlled
  - appropriate for tightly-couple systems (CPU and memory/disk)
  - common in parallel schemes

Timing methodology

- Issues
  - clock period and wire delay
  - synchronization and skew
  - encoding of timing and data information
  - handshaking
  - flow-control
  - power consumption

Number of devices communicating

- Single source – single destination
  - point-to-point
  - cheap connections, no tri-stating necessary
- Single source – multiple destination
  - fanout limitations
  - addressing scheme to direct data to one destination
- Multiple source – multiple destination
  - arbitration between senders
  - tri-stating capability is necessary
  - collision detection
  - addressing scheme
  - priority scheme
  - fairness considerations

Arbitration schemes

- Daisy-chain or token passing
  - devices either act or pass to next
  - fixed priority order
  - as many wires as devices
  - fairness issues
- Centralized
  - request to central arbiter
  - central arbiter implements priority scheme
  - wires from/to each device can be costly
  - can be dynamically changing priority/fairness
- Distributed
  - no central arbiter
  - common set of wires (or ether) observed by all devices
  - fixed priority/fairness scheme

Case studies (serial)

- RS-232 (IEEE standard)
  - serial protocol for point-to-point, low-cost, low-speed applications
  - commonly used to connect PCs to I/O devices
- I2C (Philips)
  - serial bus for connecting multiple components (senders and receivers)
  - commonly used in microcontroller-based systems
- Ethernet (popularized by Xerox)
  - most popular local area network protocol with distributed arbitration
  - different versions from 1.5Mbit/sec to 100Mbit/sec
- IrDA (Infrared Data Association)
  - up to 115kbps wireless serial (Fast IrDA up to 4Mbs)
  - standard on all laptops and PDAs, but also in desktop equipment
- Firewire (Apple - now IEEE1394)
  - 1.6Gbytes/sec
  - consumer electronics (video cameras, TVs, audio, etc.)

Case studies (parallel)

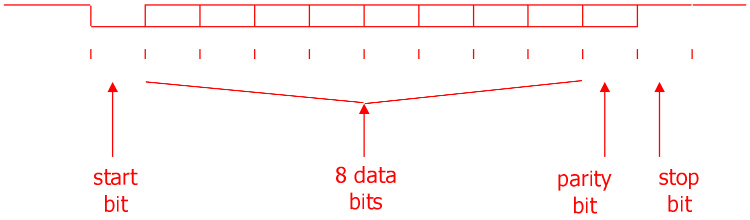
- NuBus (Texas Instruments)
  - parallel system bus used for PCs
  - backbone of Apple Macintosh
- PCI Bus (Intel)
  - parallel system bus for modern PCs
  - 66MHz with 32-bit wide data
- PCMCIA (PC Memory Card Int'l Association)
  - mostly memory-oriented bus for personal computer cards
  - supports hot insertion/removal

RS-232 (standard serial line)

- Point-to-point, full-duplex
- Synchronous or asynchronous
- Flow control
- Variable baud (bit) rates
- Cheap connections (low-quality and few wires)

Serial data format

- Variations: parity bit; 1, 1.5, or 2 stop bits



RS-232 wires

- TxD – transmit data
- TxC – transmit clock
- RTS – request to send
- CTS – clear to send

- RxD – receive data
- RxC – receive clock
- DSR – data set ready
- DTR – data terminal ready

- Ground

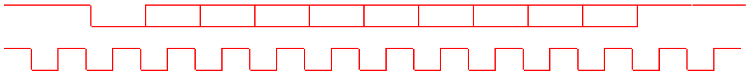
all wires active low

"0" = -12v, "1" = 12v

special driver chips that
generate ±12v from 5v

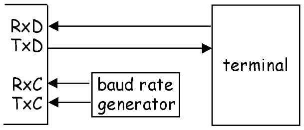
Transfer modes

- Synchronous
  - clock signal wire is used by both receiver and sender to sample data
- Asynchronous
  - no clock signal in common
  - data must be oversampled (16x is typical) to find bit boundaries
- Flow control
  - handshaking signals to control rate of transfer

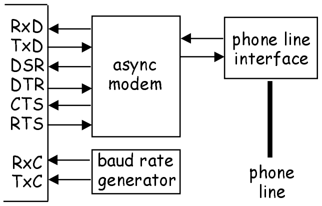


Typical connections

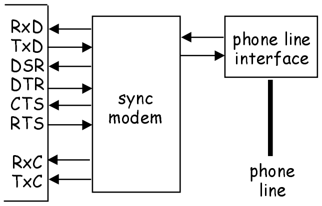
- Terminal

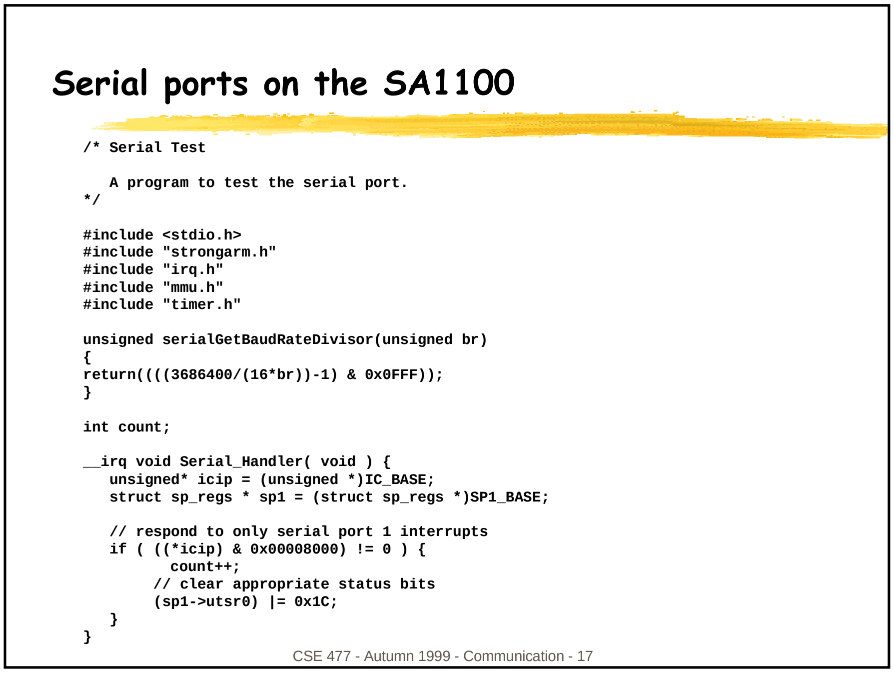


- Asynchronous modem



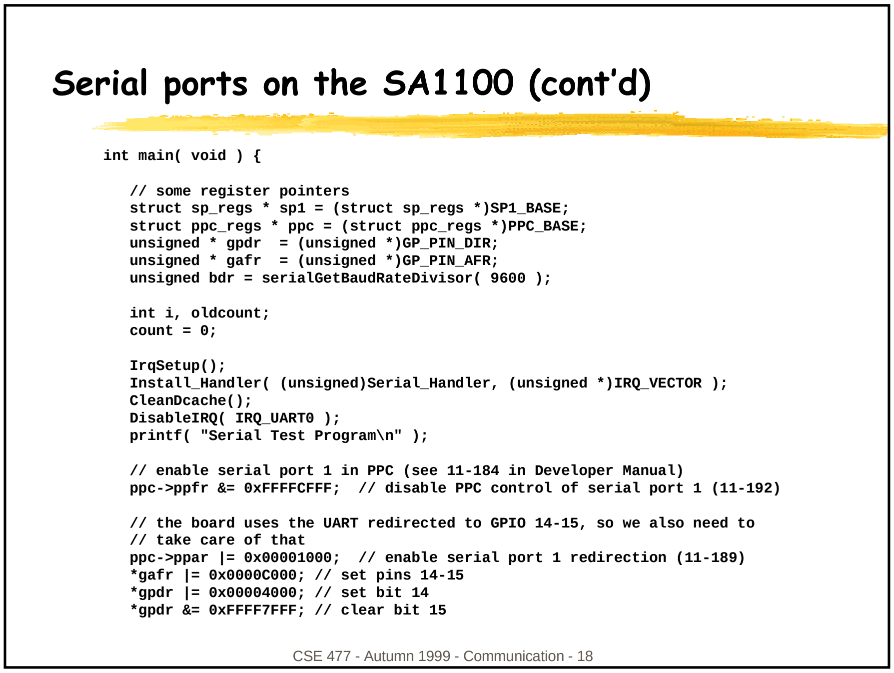
Synchronous modem





Serial ports on the SA1100

```
/* Serial Test

   A program to test the serial port.
*/

#include <stdio.h>
#include "strongarm.h"
#include "irq.h"
#include "mmu.h"
#include "timer.h"

unsigned serialGetBaudRateDivisor(unsigned br)
{
return((((3686400/(16*br))-1) & 0x0FFF));
}

int count;

__irq void Serial_Handler( void ) {
   unsigned* icip = (unsigned *)IC_BASE;
   struct sp_regs * sp1 = (struct sp_regs *)SP1_BASE;

   // respond to only serial port 1 interrupts
   if ( ((*icip) & 0x00008000) != 0 ) {
          count++;
        // clear appropriate status bits
        (sp1->utsr0) |= 0x1C;
   }
}
```



Serial ports on the SA1100 (cont'd)

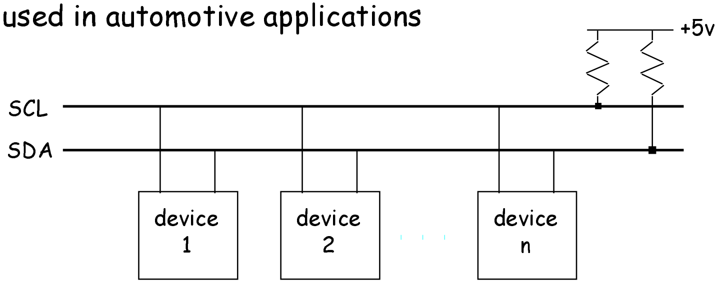
```
int main( void ) {

   // some register pointers
   struct sp_regs * sp1 = (struct sp_regs *)SP1_BASE;
   struct ppc_regs * ppc = (struct ppc_regs *)PPC_BASE;
   unsigned * gpdr  = (unsigned *)GP_PIN_DIR;
   unsigned * gafr  = (unsigned *)GP_PIN_AFR;
   unsigned bdr = serialGetBaudRateDivisor( 9600 );

   int i, oldcount;
   count = 0;

   IrqSetup();
   Install_Handler( (unsigned)Serial_Handler, (unsigned *)IRQ_VECTOR );
   CleanDcache();
   DisableIRQ( IRQ_UART0 );
   printf( "Serial Test Program\n" );

   // enable serial port 1 in PPC (see 11-184 in Developer Manual)
   ppc->ppfr &= 0xFFFFCFFF;  // disable PPC control of serial port 1 (11-192)

   // the board uses the UART redirected to GPIO 14-15, so we also need to
   // take care of that
   ppc->ppar |= 0x00001000;  // enable serial port 1 redirection (11-189)
   *gafr |= 0x0000C000; // set pins 14-15
   *gpdr |= 0x00004000; // set bit 14
   *gpdr &= 0xFFFF7FFF; // clear bit 15
```

Serial ports on the SA1100 (cont'd)

```
// setup serial port params
   sp1->utcr0 =  0x08;  // set 8-N-1, asynchronous
   sp1->utcr1 &= 0xF0;  // clear the last four bits
   sp1->utcr1 |= (bdr >> 8);  // set the baud rate
   sp1->utcr2 =  bdr;  // ...
   sp1->utcr3 =  0x0B;  // set various functions (see 11-135 in Developer Manual)

   SetIrqLevel( IRQ_UART0, FALSE );
   EnableIrq( IRQ_UART0 );

   oldcount = count;
   while(1) {
      if ( count > oldcount ) {
         printf( "received characters %c\n", sp1->utdr );
      }
   }

   return 0;
}
```

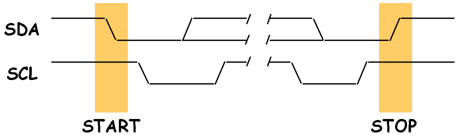
Inter-Integrated Circuit Bus (I2C)

- Modular connections on a printed circuit board
- Multi-point connections (needs addressing)
- Synchronous transfer (but adapts to slowest device)
- Similar to Controller Area Network (CAN) protocol used in automotive applications



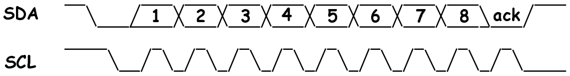
Serial data format

- SDA going low while SCL high signals start of data
- SDA going high while SCL high signals end of data
- SDA can change when SCL low
- SCL high (after start and before end) signals that a data bit can be read



Byte transfer

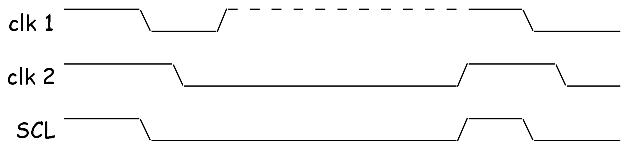
- Byte followed by a 1 bit acknowledge from receiver
- Open-collector wires
  - sender allows SDA to rise
  - receiver pulls low to acknowledge after 8 bits



- Multi-byte transfers
  - first byte contains address of receiver
  - all devices check address to determine if following data is for them
  - second byte usually contains address of sender

Clock synchronization

- Synchronous data transfer with variable speed devices
  - go as fast as the slowest device involved in transfer
- Each device looks at the SCL line as an input as well as driving it
  - if clock stays low even when being driven high then another device needs more time, so wait for it to finish before continuing
  - rising clock edges are synchronized



Arbitration

- Devices can start transmitting at any time
  - wait until lines are both high for some minimum time
  - multiple devices may start together - clocks will be synchronized
- All senders will think they are sending data
  - possibly slowed down by receiver (or another sender)
  - each sender keeps watching SDA - if ever different (driving high, but its really low) then there is another driver
  - sender that detects difference gets off the bus and aborts message
- Device priority given to devices with early 0s in their address

Inter-Integrated Circuit Bus (I2C)

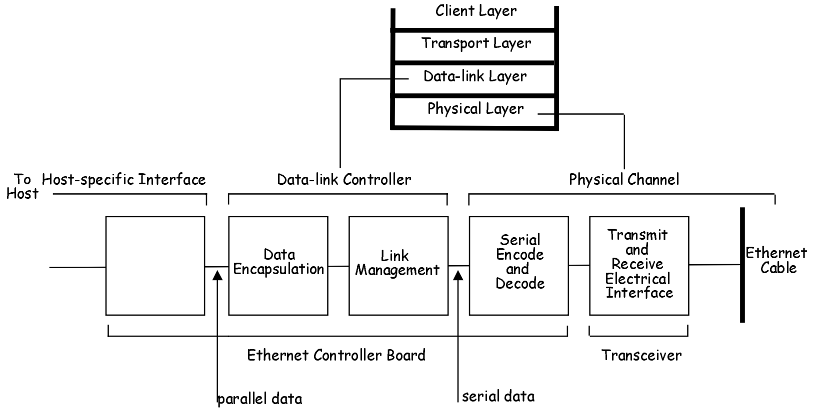
- Supports data transfers from 0 to 400KHz
- Philips (and others) provide many devices
  - microcontrollers with built-in interface
  - A/D and D/A converters
  - parallel I/O ports
  - memory modules
  - LCD drivers
  - real-time clock/calendars
  - DTMF decoders
  - frequency synthesizers
  - video/audio processors

Ethernet (Xerox local area network)

- Local area network
  - up to 1024 stations
  - up to 2.8 km distance
  - 10Mbits/sec serially on shielded co-axial cable
  - 1.5Mbits/sec on twisted pair of copper pair
- Developed by Xerox in late 70s
  - still most common LAN right now
  - being displaced by fiber-optics (can't handle video/audio rates or make required service guarantees)
- High-level protocols to ensure reliable data transmission
- CSMA-CD: carrier sense multiple access with collision detection

Ethernet layered organization

- Physical and data-link layers are our focus



Client Layer

Transport Layer

Data-link Layer

Physical Layer

To Host

Host-specific Interface

Data-link Controller

Physical Channel

Data Encapsulation

Link Management

Serial Encode and Decode

Transmit and Receive Electrical Interface

Ethernet Cable

Ethernet Controller Board

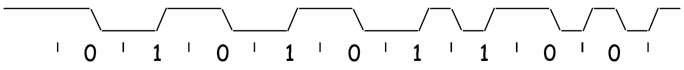
Transceiver

parallel data

serial data

Serial data format

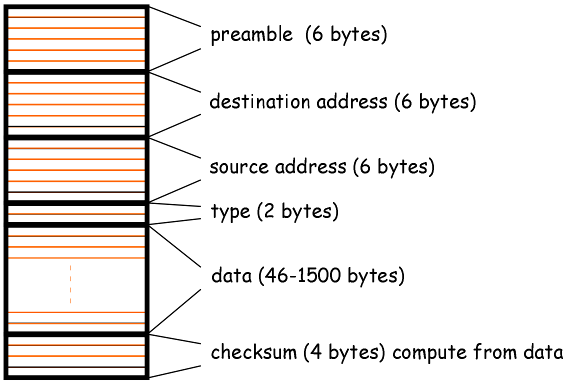
- Manchester encoding
  - signal and clock on one wire (XORed together)
  - "0" = low-going transition
  - "1" = high-going transition



- Extra transitions between 00 and 11 need to be filtered
  - preamble at beginning of data packet contains alternating 1s and 0s
  - allows receivers to get used to where important transitions should be and ignore extra ones (this is how synchronization is achieved)
  - preamble is 48 bits long: 10101. . . 01011

Ethernet packet

- Packets size: 64 to 1518 bytes + 6 bytes of preamble



preamble (6 bytes)

destination address (6 bytes)

source address (6 bytes)

type (2 bytes)

data (46-1500 bytes)

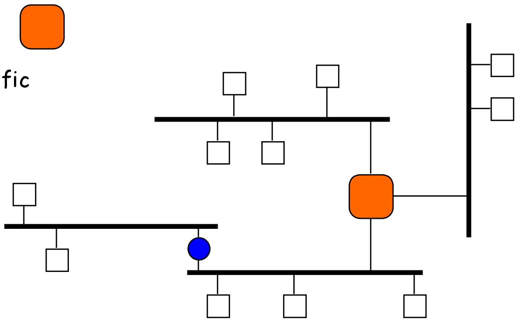
checksum (4 bytes) compute from data

Arbitration

- Wait for line to be quiet for a while then transmit
  - detect collision
  - average value on wire should be exactly between 1 and 0
  - if not, then two transmitters are trying to transmit data
- If collision, stop transmitting
  - wait a random amount of time and try again
  - if collide again, pick a random number from a larger range (2x) and try again
- Exponential backoff on collision detection
- Try up to 16 times before reporting failure

Extending Ethernet

- Segments, repeaters, and gateways
  - segment: a single cable
  - repeater: transfers all messages on one segment ● to another and vice-versa
  - gateway: selectively forwards messages 🟧 to another segment helps to isolate traffic

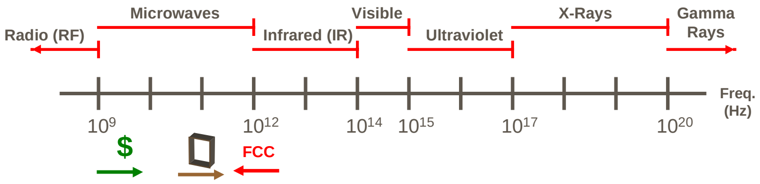


IrDA: The Infrared Data Association Standard

- Wireless communication
- IrDA goals
- IrDA protocol stack
- Extensions to the standard
- Performance issues
- Design implications

Where Infrared (IR) Fits In

- Notes:
  - Implementation costs rise significantly around 1-10 GHz. (But one important exception is IR at around 500 THz ; very inexpensive.)
  - Signals above 100 GHz cannot penetrate walls
  - Most signals below 300 GHz are regulated by the FCC

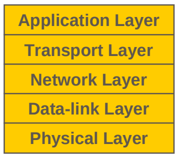


Infrared Data Association

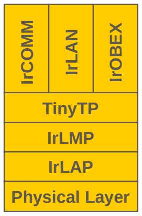
- Consortium of over 160 companies
- Goals:
  - Perceived target was the "mobile professional"
    - Short interactions with other devices (file transfer, printing)
    - Possibly using others' peripherals (visiting a customer's office)
  - Thus, wanted:
    - Suitable replacement for cables
    - Interoperability
    - Minimal cost
    - "Point-and-shoot" model (intended use and to reduce interference)
- History:
  - First standard developed in 1994
  - Revisions as recently as late 1998 (i.e., still active)

IrDA Protocol Stack

- Analogous to the standard layered network model
- Consists of both required and optional components

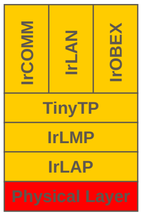


Standard Network Model



IrDA Protocol Stack

Physical Layer



- Purpose:
  - Handle the bit-level transfer of data
- Components include:
  - Transmitter (LED)
  - Receiver (photodiode)
  - “Framer” (software to handle on/off protocol)
- Note(s):
  - Exact physical-layer protocol used depends on speed of IrDA connection

Speed

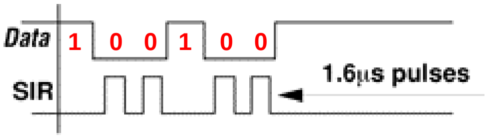
- IrDA supports wide range of speeds
  - 2400 bps to 4 Mbps.
- Recall: uses highest speed available on both devices (determined when connection is established)
- Future promises even higher speeds:
  - 16 Mbps standard is "nearly complete"
  - 50 Mbps is "technologically feasible" (but far off)
- Comparison to other technologies:
  - RF slightly slower (1 - 2 Mbps max)
  - For perspective: max modem speed is 56.6 kbps

Power Issues

- Different modulation schemes are used for different speeds
  - Reasons depend on efficiency and backward compatibility
  - Also, eye safety is a concern
- As a result, power consumption (and efficiency) depends on speed of connection
- Some interesting results, it turns out.
- To understand fully, need to briefly examine the modulation schemes….

Low-speed Modulation

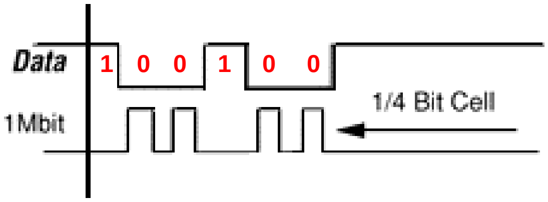
- Speed: 2400 bps - 115 kbps ("Serial Infrared", or SIR)
- How it works:
  - only 0's require pulse (and thus power) ; pulse < full bit time
  - standard UART byte framing
  - pulse is constant 1.6 µs long (so duty cycle varies with speed)
- Average Duty Cycle: ≤ 9%



Byte framing (UART):

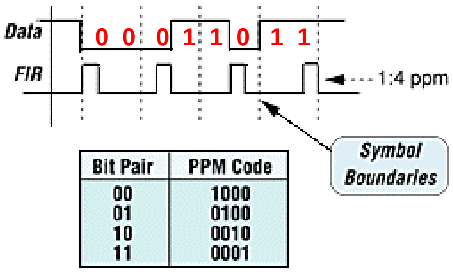
Medium-speed Modulation

- Speed: 576 kbps - 1 Mbps
- How it works:
  - similar to SIR (pulse only for 0's ; pulse < full bit time)
  - pulse lasts 1/4 of bit time (so pulse varies with speed)
- Average Duty Cycle: 12.5%



High-speed Modulation

- Speed: 4 Mbps ("Fast Infrared", or FIR)
- How it works:
  - uses four-pulse-position-modulation scheme (4PPM)
  - pulse during exactly 1/4 of each symbol boundary
  - 4PPM makes synchronization easier to maintain
- Duty Cycle:
  25% (independent
  of data)



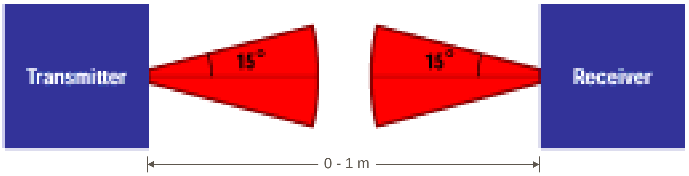
Bit Pair	PPM Code
00	1000
01	0100
10	0010
11	0001

Power Issues (cont'd)

- So what does this all mean?
- Duty cycle (and thus total power consumption) increases as speed increases
  - Somewhat expected. It's doing more, and nothing is free.
- However, interesting to note that power per bit actually decreases as speed increases
  - IrDA's higher-speed modulation schemes more efficient
- Hard Numbers:
  - Around 5 mW during operation (at 1 Mbps)
  - Can often go into shutdown mode when not in use
  - Compare: around 100 mW for typical RF

Range

- Linear:
  - IrDA standard requires 0-1 m
  - Realistically, some transceivers work at up to 10 m
- Angular:
  - Limited to a narrow cone (15° half-angle)
  - Done to help reduce interference between devices

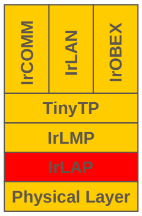


Physical Dimensions and Cost

- Physical Dimensions:
  - IrDA-compliant transceivers can be extremely small
  - Newest IBM module is only 4 mm x 5 mm x 9 mm !
  - (Assumes CPU handles protocols, etc.)
- Cost:
  - In bulk, can get IrDA transceiver for approx. $2 - $4
  - RF modules typically more expensive ($20 - $25)

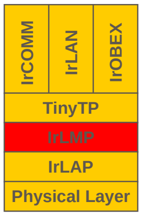
IrLAP - "Link Access Protocol"

- Purpose:
  - Handle connections/disconnections
  - Implement reliable transfer



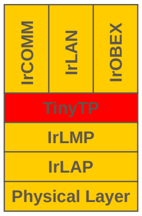
- Details:
  - Connection negotiation always begins using fixed set of parameters (9600 bps, no parity, etc) to avoid compatibility problems
  - After exchanging capabilities, speed is increased to best available
  - If connection interrupted, sends notification to higher layer
- Interesting: IrLAP based on HP calculators' IR xfer

IrLMP - "Link Management Protocol"



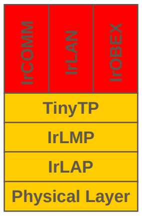
- Two main components:
- IrMUX:
  - Multiplexes several "virtual" connections on a single IrLAP connection
    - To allow this, uses Logical Service Access Points (LSAPs), which are very similar to IP ports
    - Main differences: only 256 LSAPs, and so dynamically allocated for services (instead of "well-known" ports as in IP)
- IrIAS:
  - The "yellow pages" of services available on device, and the LSAPs to which those services are currently mapped
    - List may be hard-coded in embedded system, or an API might exist that allows applications to add/remove services from list

TinyTP - Tiny Transport Protocol



- Purpose:
  - Segmentation and Re-assembly
    - Automatically break-up large packets (and put back together correctly)
  - Per-channel Flow Control
    - Uses a "credit"-based system (credits allow sender to send)
    - Is necessary in order to avoid problems where 2 IrLMP connections are on same IrLAP connection
    - To avoid deadlock, credit-only packets are not subject to flow-control
- Note: TinyTP is technically an optional protocol stack level, but IrDA strongly recommends it in all IrDA implementations

High-level Protocol Layers



- Exist in order to make life easier for developers
- Some exist mainly to help support legacy applications
- IrCOMM:
  - Serial and parallel port emulation
    - Designed to aid initial migration of serial/parallel port applications to IrDA
    - Overcomes the significant differences between the two protocols, which include (for example) the presence of only a single beam in IR, vs. multiple wires in a serial or parallel port
    - Disadvantage: some IrDA features lost when IrCOMM used (e.g., IAS and connection-speed negotiation)

High-level Protocol Layers (cont'd)

- IrLAN:
  - Makes it easy for an IrDA device to connect to a local-area network.
- IrOBEX:
  - IR "Object Exchange" ; allow transfer of abstract objects
    - Very convenient way to transfer files (most common use)
    - Some support for recognizing and handling file type automatically (similar to HTTP)
    - Interesting: was actually based on HTTP protocol
- Trend seems to be that extensions are getting very specialized...

Other Protocol Extensions

- In addition to the original high-level protocol layers, more recent extensions have been developed, including:
- IrTRAN-P:
  - handles transfer of pictures between devices (especially digital cameras)
- IrMC:
  - handles exchange of information (phonebook, calendar) among mobile communication devices; also handles real-time voice.

Other Protocol Extensions (cont'd)

- IrCONNECT:
  - Designed for communication between cordless peripherals (e.g., mice, joysticks, etc) and host devices (e.g., PCs, TV/web set-top boxes, etc.)
  - Important goals include low-latency, and compatibility with USB components (recall purpose of IrCOMM)
  - Radically different from other IrDA protocols in several ways:
    - Longer range (8 m), but slower (max 75 kbps)
    - Bi-directional
    - Allows single host to talk to multiple peripherals (up to 8) simultaneously!
- AIR - "Advanced Infrared":
  - High-speed (4 Mbps), multiple connections (up to 10), designed for network-type situations
  - Not yet standardized (expected 2Q/99)

When Space Is Tight - IrDA Lite

- What:
  - A set of IrDA implementation suggestions
    - Reduces code and RAM needed (at the expense of performance)
    - Designed for devices where RAM/ROM very limited, and performance not critical (e.g., wristwatches, etc)
  - Strategies include:
    - Limited speed (e.g., only 9600 bps)
    - Limited packet size (e.g., ≤ 64 bytes)
    - Limited window size (e.g., only 1 window slot)
  - Pick-and-choose approach
    - Can mix and match strategies based on desired performance
    - One exception: replacing IrLMP connect/disconnect with an exposed IrLAP version requires major code changes. (So basically required; has implications.)

resource needs down to around 5 KB code and 200 bytes RAM

Performance Analysis

- When deciding whether to use IrDA, want to consider several important factors
- These include:
  - Speed
  - Power Consumption
  - Range
  - Protocol Overhead
  - Physical Dimensions
  - Cost
- Convenience is also an important factor

Protocol Overhead

- Very simple model (point-to-point), so can expect reduced protocol overhead
- For layers in IrDA protocol stack, overhead per packet/frame is:
  - IrLAP = 2 bytes
  - IrLMP = 2 bytes
  - TinyTP = 1 byte

  <u>Total:</u> 5 bytes
- For perspective, compare to TCP/IP over Ethernet:
  - Ethernet = 18 bytes minimum
  - IP = 20 bytes
  - TCP = 20 bytes

  <u>Total:</u> 58 bytes (minimum)
- So IrDA takes advantage of its simpler model, and keeps protocol overhead very low.

Convenience

- Lots of things available to make use of IrDA easier:
- IrDA transceivers:
  - Ready-to-use modules available from many companies
- IrDA protocol stack:
  - Protocol stacks available in "kit" form, for use in embedded system designs
- Operating system support:
  - Many operating systems have built-in support for IrDA (including WinCE / Win98 / Win2000, GeoWorks, VxWorks, and pSOS)

Summary of IrDA

- Advantages :
  - Competitive cost
  - Lots of industry support
  - Well-understood protocol stack
  - Existing applications easy to port (due to IrCOMM, etc.)
  - Interoperability
  - Low power
- Disadvantages:
  - Growing number of specialized protocols
  - Limited range
    - "Point-and-shoot" model prohibits certain applications

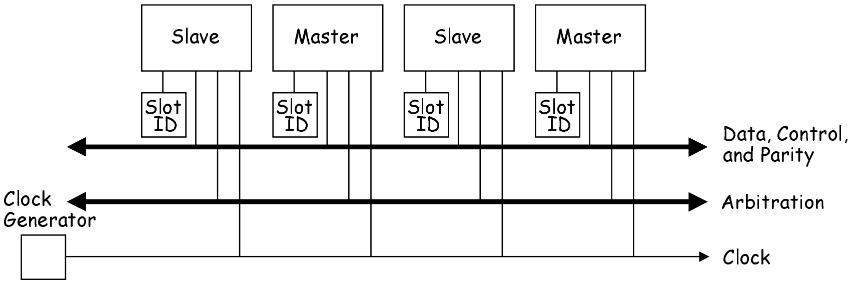
NuBus (Texas Instruments and Apple)

- Parallel system bus (within a computer)
  - used in TI and Apple computers
  - 40MBytes/sec maximum transfer rate
  - fully synchronous (data transfer and arbitration)
- Supports up to 16 masters
  - distributed arbitration
  - fairness enforced
- All operations are memory-mapped
  - read/write is everything
- Mechanical standards
  - size and shape of boards
  - position of pins
  - capacitance limits

NuBus organization

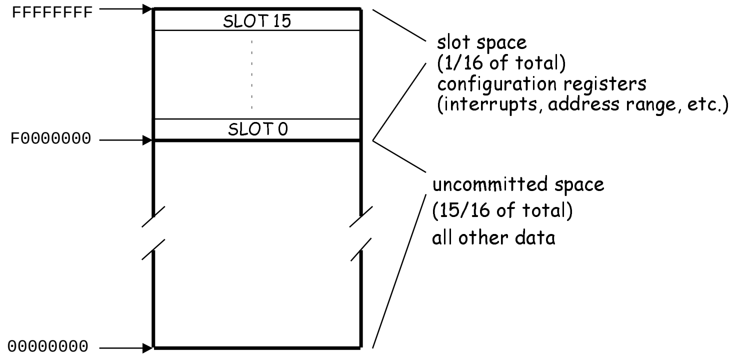
- Up to 16 elements
  - masters originate operations
  - slaves respond to requests from masters
  - slot ID provides part of addressing

Utility	reset, clock
Control	start, ack, tm0, tm1
Addr/Data	ad[31:0]
Arbitration	arb[3:0], rqst
Parity	sp, spv
Slot ID	id[3:0]
Power/Ground	+5 (11), -5 (8), +12 (2), -12v (2), gnd (23)
Total 96 pins	3 rows of 32 pins



Addressing

- Multiplexed address/data bus
- Slot ID provides second set of high-order 4 bits of address

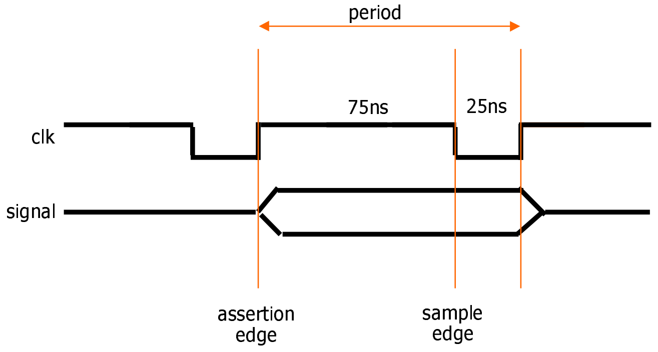


slot space
(1/16 of total)
configuration registers
(interrupts, address range, etc.)

uncommitted space
(15/16 of total)
all other data

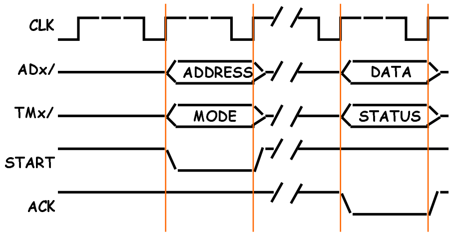
NuBus timing

- 100ns cycle time, 75% duty cycle
  - allows more time for propagation of signals, less for skew

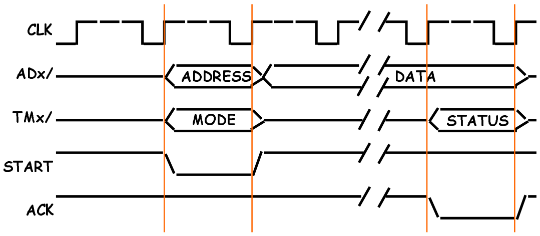


Data transactions

- Read



- Write



Transfer modes and status codes

- TM0/, TM1/, AD1/, AD0/ define transfer mode from master

TM1/	TM0	/AD1/	AD0/	Type of transfer
0	0	0	0	write byte 3
0	0	0	1	write byte 2
0	0	1	0	write byte 1
0	0	1	1	write byte 0
0	1	0	0	write halfword 1
0	1	0	1	write block (2 to 16 words specified using AD2/ to AD5/
0	1	1	0	write halfword 0
0	1	1	1	write word
1	0	0	0	read byte 3
1	0	0	1	read byte 2
1	0	1	0	read byte 1
1	0	1	1	read byte 0
1	1	0	0	read halfword 1
1	1	0	1	read block (2 to 16 words specified using AD2/ to AD5/
1	1	1	0	read halfword 0
1	1	1	1	read word

- TM0/, TM1/ define status code from slave

TM1/	TM0/	Type of acknowledge
0	0	bus transfer complete
0	1	error
1	0	bus timeout error
1	1	try again later

Arbitration

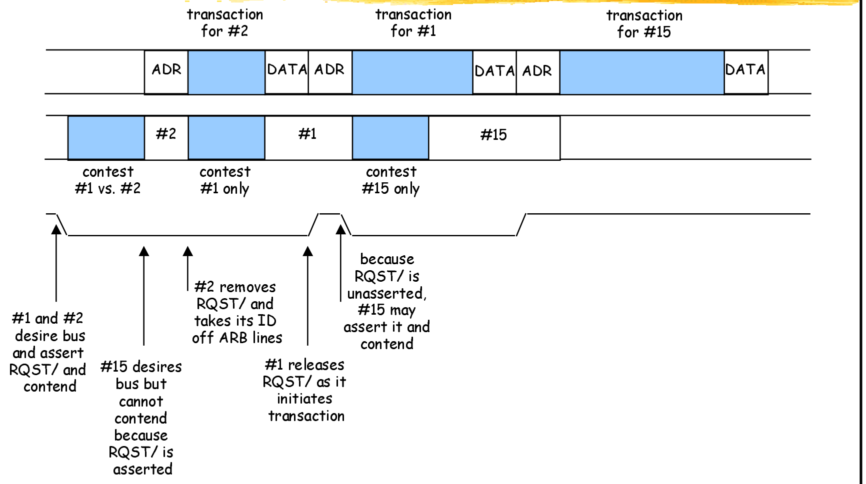
- Distributed arbitration
  - devices requesting bus place ID values on open-collector ARB lines
  - if value on ARB ≠ ID then they stop driving ARB lines
  - device with ARB = ID gets bus (must be decided in 2 bus cycles)
  - e.g., #1 vs. #2:

#1	#2	bus	
0001	0010	0000	start arbitration
0...	0...	0...	check 1st bit (both ok)
00...	00...	00...	check 2nd bit (both ok)
000...	001...	000...	check 3rd bit (#2 loses and removes itself)
0001	...	0001	check 4th bit (#1 matches and wins)

Arbitration (cont'd)

- Avoids starvation by doing arbitration in rounds
  - RQST/ must be high before device can request bus
  - all simultaneous requestors are taken care of before others can request
  - bus is quiet after ACK/ and before next START/
- Fairness
  - all requestors in round are taken care of
  - master may monopolize the bus if it doesn't raise its RQST/ and
  - releases ARB/ lines – sometimes necessary for time critical actions

Arbitration example



transaction
for #2

transaction
for #1

transaction
for #15

ADR

DATA

ADR

DATA

ADR

DATA

#2

#1

#15

contest
#1 vs. #2

contest
#1 only

contest
#15 only

#1 and #2
desire bus
and assert
RQST/ and
contend

#15 desires
bus but
cannot
contend
because
RQST/ is
asserted

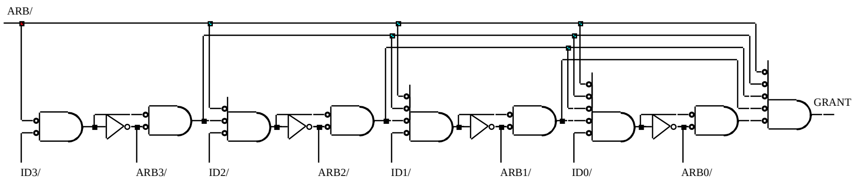
#2 removes
RQST/ and
takes its ID
off ARB lines

#1 releases
RQST/ as it
initiates
transaction

because
RQST/ is
unasserted,
#15 may
assert it and
contend

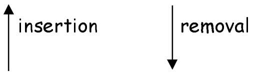
Arbitration logic

- ARB/ lines are open-collector
- ID/ values are hard-wired
  on each slot of NuBus chassis



PCMCIA/JEIDA (PC card)

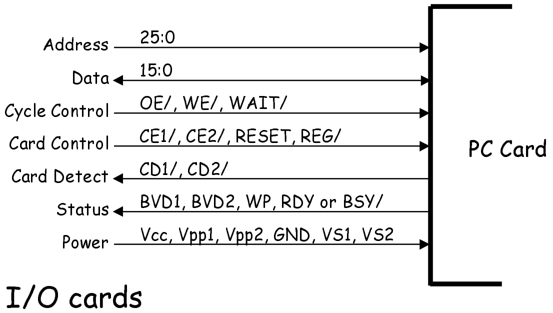
- Parallel bus for memories and I/O devices
  - under control of a single master processor (no arbitration)
  - 16-bit wide data transfers, 26-bit address space
- Designed for portable applications
  - 3.3mm thick cards with 68-pin connectors (2 rows of 34 pins)
  - 5.0v or 3.3v operation
  - supports hot-insertion/removal
    - short power pins
    - medium length signal pins
    - long ground pins



- Card types
  - PCMCIA type I: 3.3mm thick cards (requires QFP surface mount)
  - PCMCIA type II: 5mm (for cabling, e.g., FAX/modem)
  - PCMCIA type III: 10.5mm (for hard disks, wireless transceivers)

PCMCIA signals

- For memory cards (SRAM, DRAM, Flash, ROM)



- For I/O cards
(disks, FAX/modem, wireless transceiver)
  - IREQ/ replaces BSY/
  - IOR/ replaces OE/
  - IOW/ replaces WE/

PCMCIA signals

- Address/Data
  - 26 bits of address space, 16 bits parallel data (can also just use 8 bits)
- Cycle Control
  - read (OE/), write (WE/), and wait (WAIT/)
- Card Control
  - reset (RESET), access card information structure (REG/), low and high byte selects (CE1/, CE2/)
- Card Detect
  - shortest pins at either end of connector to determine proper card insertion (CD1/, CD2/)
- Status
  - card battery voltage detect and reset (BVD1, BVD2), write protected (WP), card ready (RDY or BSY/, used to slow down data transaction)
- Power
  - power to card (Vcc, GND)
  - programming voltage for flash memory (Vpp1, Vpp2)
  - card voltage requirements (VS1, VS2)

Data transactions

- Asynchronous transfer
  - pace set by system using card
  - can be slowed down by the system using WAIT/ or by the card using RDY line
- OE/ and WE/ determine read and write, respectively
- CE1/ and CE2/ select width of transfer
- REG/ is used to select reading of card information structure (CIS)

PCMCIA architecture

- Card information structure (CIS)
  - separate memory space for manufacturer information on card
  - also used as configuration space for card, e.g., modem baud rate
- Layers of software make card usable by the system

socket services:	reads CIS, configures card on behalf of card services, sends commands to card interface, forward interrupts (independent of specific cards)
card services:	operating system component, performs resource allocation for applications, keeps track of insertion and removal of cards, uses socket service to interact with card (independent of specific cards)
card drivers:	actually know how to use specific cards and use card services to execute operations