

In 1970, *one byte* of RAM cost \$1.
This means that in 1970, 1MB of RAM
cost *ONE MILLION DOLLARS...*

How many MB of RAM
do *you* have in your computer?

Moore's Law states that processor
capability doubles every 18 months.
How long can this continue?



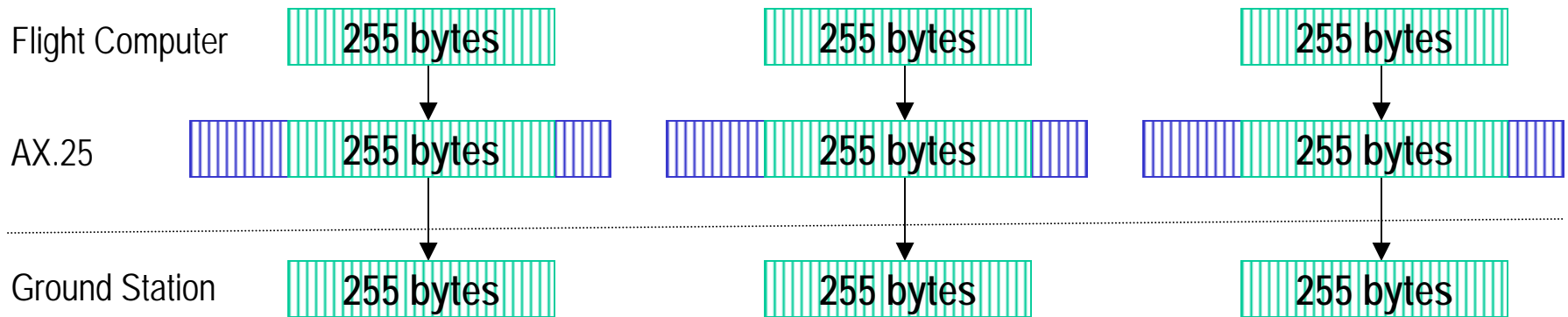
► Presentations on the web at: <http://citizen-explorer.colorado.edu/restricted/cdh>



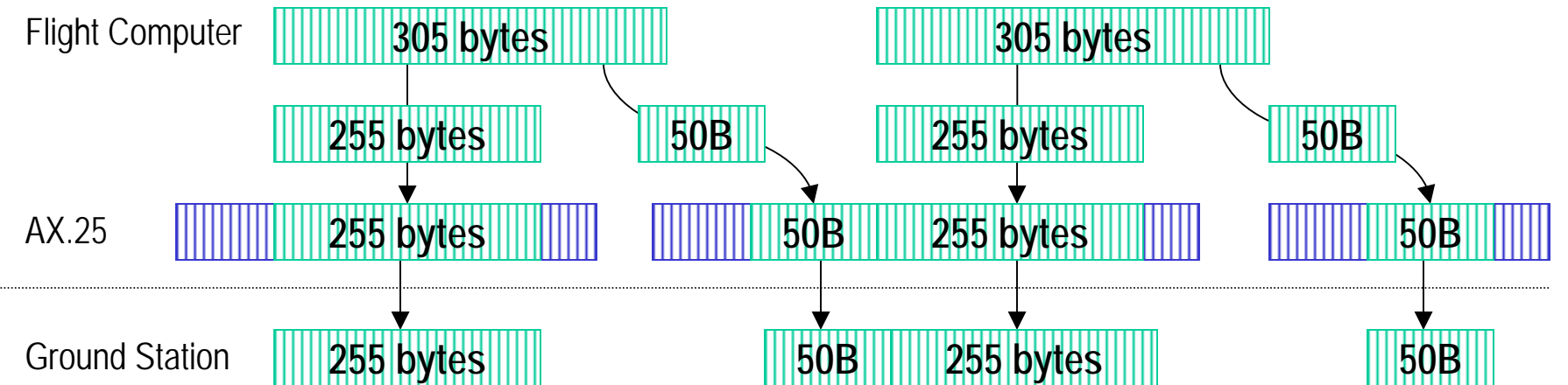
- ▶ Provide a *completely transparent* interface to the flight computer
 - ▶ Minimize additional required software
 - ▶ No AX.25, KISS, or command/response facilities running on VxWorks
 - ▶ Instead we use TCP/IP over PPP to get network connections, let the Comm Interface handle the packetizing / unpacketizing
 - ▶ Allow full visibility of Flight Computer
 - ▶ “Embedded BIOS” features attach screen/keyboard I/O to Comm serial port
 - ▶ If Comm transparently sends everything from serial port (no extra software), we can watch, affect system startup, alter BIOS settings, run DOS programs, etc.
 - ▶ Provides valuable recovery options
- ▶ Interface with SpaceQuest hardware
 - ▶ Generic (“dumb”) sync-serial interface plus nybble-wide Modem config interface (6-wire)
 - ▶ Allows complete protocol flexibility
- ▶ Read/write data compatible with the COTS Amateur Radio ground station
 - ▶ AX.25-compatible packets
 - ▶ “Unconnected transparent” mode (no AX.25 handshaking)
 - ▶ Minimizes firmware overhead on flight side
 - ▶ Deeper PPP, TCP/IP layers handle error detection, resend handshaking



Synchronized Transmission (PPP packet size = AX.25 packet size)



Unsynchronized Transmission (PPP packet size != AX.25 packet size) - a problem?



Original packet sizing shattered, but all bytes still arrive in the same order. Intrapacket timing issues?



- ▶ Dual-channel full-duplex synchronous serial ports (data + clock)
 - ▶ Clocks come from SpaceQuest modem on both transmit and receive
- ▶ Memory mapped from 0x7000 to 0x7FFF, also uses IRQ IE0
- ▶ HDLC controller - sends / receives HDLC packets (Siemens: "HSCX")
 - ▶ Underlying protocol for AX.25
 - ▶ Using "Transparent Mode 0" - chip handles start, end flags and checksum, no automatic address recognition, ACKs, or resend requests
 - ▶ We build AX.25 "Unnumbered Information" (UI) packets on top of buffer
 - ▶ Fixed header - contains callsign information for source and destination (CU / CX-1)
 - ▶ Chip performs "NRZI" encoding on data
- ▶ 32-byte double-buffers on send and receive per channel
 - ▶ read / fill one 32-byte buffer while chip is accessing other (= 37Hz, 27ms)
- ▶ parallel interrupts
 - ▶ IRQ may be caused by more than one interrupt, all sources visible as bits
 - ▶ Many commands and interrupts

Flag	Address	Control	PID	Info	FCS	Flag
01111110	112/224 Bits	8/16 Bits	8 Bits	N*8 Bits	16 Bits	01111110

Figure 3.1b. Information frame construction.

<http://citizen-explorer.colorado.edu/restricted/cdh/ax25.pdf>



- ▶ Standard UART architecture for PCs (beware lots of reversed lines!)
- ▶ Single channel full-duplex async serial port
 - ▶ External crystal for baud rate generation (firmware initializes at 9600 baud, 81N)
- ▶ Full hardware handshaking available and supported by hardware
 - ▶ Some ports renamed from chip designations - DCE instead of DTE
 - ▶ May be necessary to implement for buffer control - 9600bps to < 9600bps
- ▶ Memory mapped from 0x6000 to 0x6FFF, also uses IRQ IE1
- ▶ 16-byte FIFO buffers on send and receive
 - ▶ Read / written through single memory byte
 - ▶ Interrupts available for 1,4,8,14 bytes received, plus "stale" data (= 120Hz, 8ms)
- ▶ Serial interrupts
 - ▶ Highest pending priority (4 types) is reported
 - ▶ Level-controlled interrupt is cleared after interrupt condition is removed
 - ▶ i.e. reading characters from receive FIFO until level falls below trigger
 - ▶ Interrupt-driven "send" task may starve while "receive" takes 100% of time



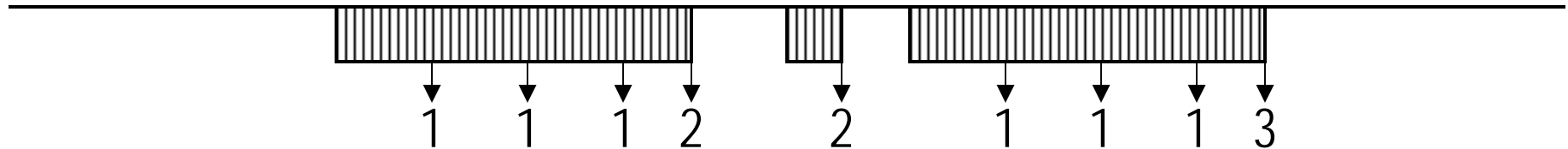
- ▶ Interrupts provide a mechanism to interrupt normal program execution to run a piece of code “right now” based on a hardware trigger
- ▶ Program control returned to original location when interrupt concludes (RETI)
 - ▶ It is the interrupt routine’s job to save the processor’s state and restore it when finished
- ▶ The 8051 provides about a dozen interrupts, some external, some internal, vectored to hardcoded locations at start of program space
 - ▶ Each location provides 8 bytes of code space, usually jump to longer routines elsewhere
 - ▶ No “software” interrupts, although any interrupt can be triggered via software
- ▶ External interrupts shared with general digital I/O lines
- ▶ Four priorities; a higher priority interrupt will break into a lower-priority interrupt
 - ▶ New lower-priority interrupts will not be queued! Workaround:
- ▶ Can be edge or level triggered
 - ▶ We’re using level triggered to avoid missing an edge while servicing another interrupt
- ▶ Problem: Interrupts must finish quickly, for they prevent other code from executing
 - ▶ Deadlines can be missed
- ▶ Interrupts can be disabled / enabled individually and globally
 - ▶ May help with synchronization issues which priority doesn’t address



- ▶ IE0 (HDLC)
 - ▶ Receive Pool Full (RPF)
 - ▶ Receive Message End (RME)
 - ▶ Transmit Pool Ready (XPR)
 - ▶ Receive Frame Overflow (RFO)
 - ▶ Transmit Data Underrun (XDU)
 - ▶ IE1 (UART)
 - ▶ Receiver Line Status (RLS)
 - ▶ Received Data Available (RDA)
 - ▶ Character Timeout (TO)
 - ▶ Transmitter Holding Register Empty (THRE)
 - ▶ MODEM Status (MS)
 - ▶ TIMER 2 (internal to 8051)
 - ▶ *n* ms "tick"
 - ▶ used for various timers (uptime, packet send time delay, transmitter auto-off, deploys), also attempted for use in packet processing and storage after packet receive (allowed IE0 to continue functioning - otherwise misses start of next packet)
 - ▶ ...May also need to use Serial Line Interrupt (micronet interface) before this is over...
- (HDLC commands)
Receive Message Complete (RMC)
Transmit Transparent Frame (XTF)
Transmit Message End (XME)
Reset HDLC Receiver (RHR)
Transmitter Reset (XRES)
- UART interrupts are cleared by resolving the offending condition



Incoming data

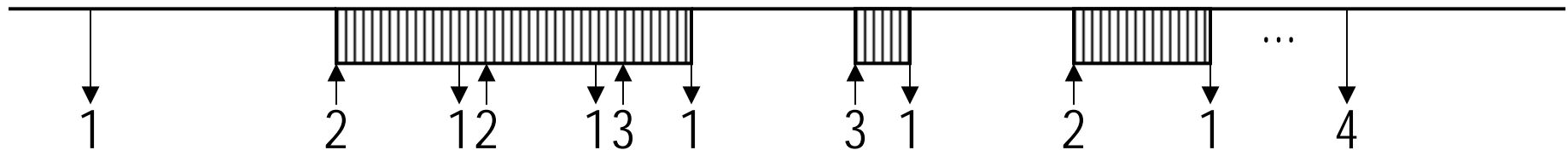


1. Chip reports that receive buffer is full, remove 32 bytes ASAP and acknowledge so that chip can use that buffer space.

2. When end of packet is reached, chip reports that ≤ 32 bytes are available. Remove data and perform any necessary packet manipulation and storage. This needs to be done fast; another incoming packet could start immediately!

3. If a bad packet is received (bad checksum or improper formatting), chip will send an error interrupt (the rest of the message should be discarded). An error will also be reported if the receive buffer isn't emptied fast enough (RX overrun).

Outgoing data



1. Chip reports that transmit buffer is empty (must remember this until data is available!)

2. When data is available, transfer next 32 bytes to chip. If less than 32 bytes available, send a short packet (#3), or wait for more data.

3. When end of packet is reached (< 32 bytes available, or $>$ packet size), signal end of packet

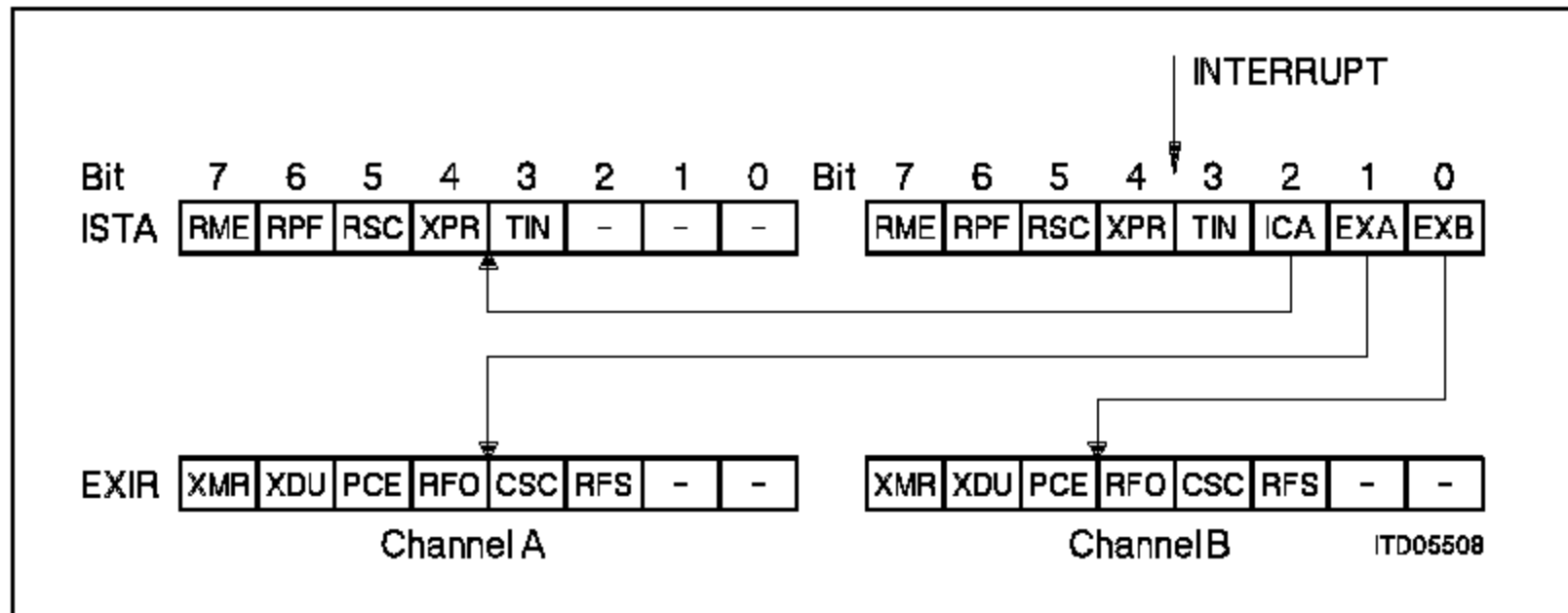
4. If in the middle of sending a packet and can't fill transmit buffer fast enough (TX underrun), chip will send an error interrupt

The Hard Part: both of these are happening at the same time, on two different chips...



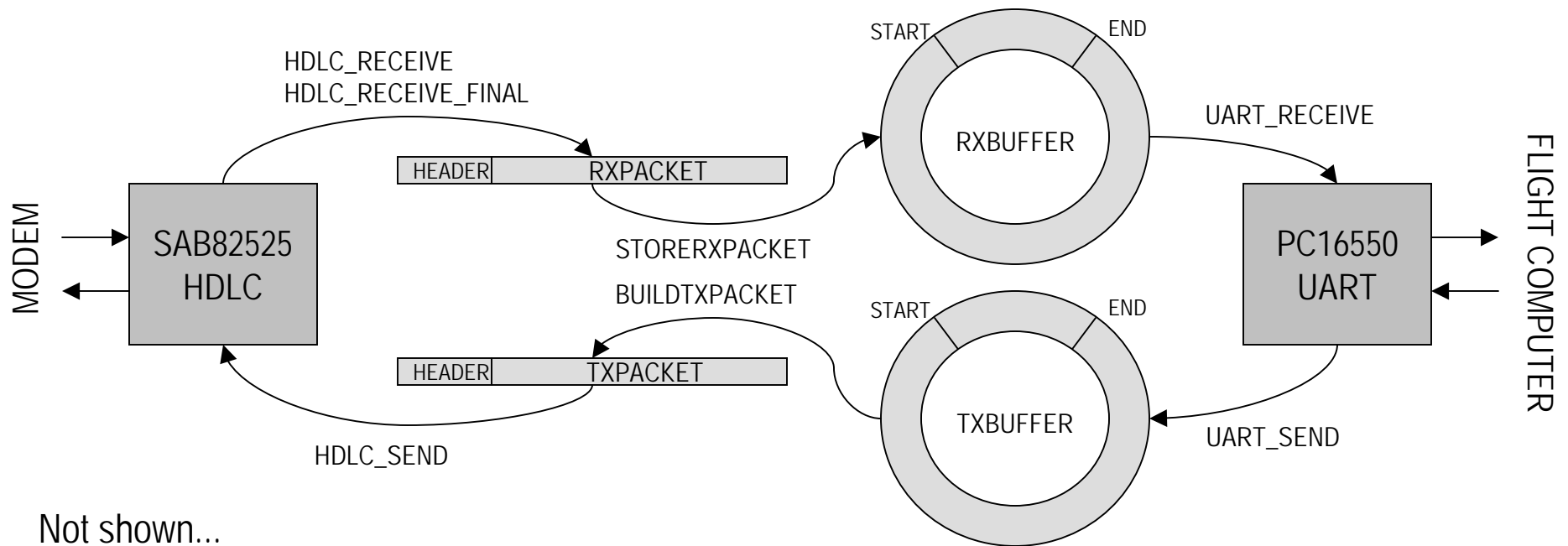
Interrupt Identification Register			Interrupt Set and Reset Functions			
Bit 2	Bit 1	Bit 0	Priority Level	Interrupt Type	Interrupt Source	Interrupt Reset Control
0	0	1	—	None	None	—
1	1	0	Highest	Receiver Line Status	Overrun Error or Parity Error or Framing Error or Break Interrupt	Reading the Line Status Register
1	0	0	Second	Received Data Available	Receiver Data Available or Trigger Level Reached	Reading the Receiver Buffer Register or the FIFO Drops Below the Trigger Level
1	0	0	Second	Character Timeout Indication	No Characters Have Been Removed From or Input to the RCVR FIFO During the Last 4 Char. Times and There Is at Least 1 Char. in It During This Time	Reading the Receiver Buffer Register
0	1	0	Third	Transmitter Holding Register Empty	Transmitter Holding Register Empty	Reading the IIR Register (if source of interrupt) or Writing into the Transmitter Holding Register
0	0	0	Fourth	MODEM Status	Clear to Send or Data Set Ready or Ring Indicator or Data Carrier Detect	Reading the MODEM Status Register





Command and Data Handling System (C&DH)

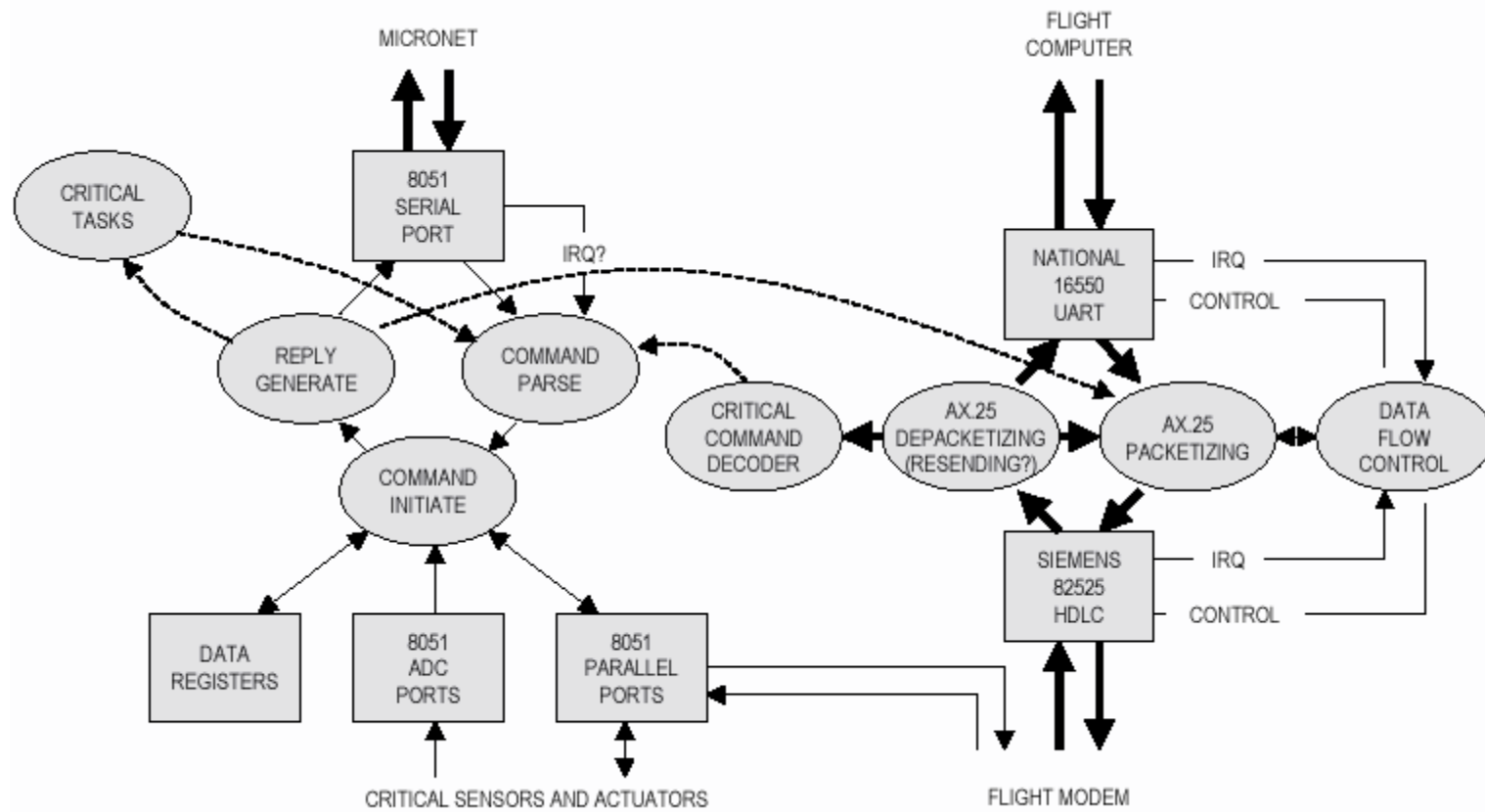
Comm Data Flow



Not shown...

- ▶ AX.25 header build and check (100%)
- ▶ Timing to send short packets before data goes stale (90%)
- ▶ Refiring of Tx_buff_empty IRQs if data is not immediately available (80%)
- ▶ Edu-channel operation (75%)
- ▶ Error detection, handling, and recording for all buffers (50%)
- ▶ Transmitter auto powerup and timeout (50%)
- ▶ Parameter control of Modem and Transmitter (30%)
- ▶ Hardware handshaking with Flight Computer ("pause" functionality, 10%)
- ▶ Detection and shunting of Critical Commands to micronet and back (10%)
- ▶ Synchronization between Comm side and Micronet side (command waiting flags, etc., 10%)
- ▶ Micronet side reorg (transmit, etc., 10%)
- ▶ Micronet UART busy-wait issues? (reprogram UART for IRQ operation? 0%)





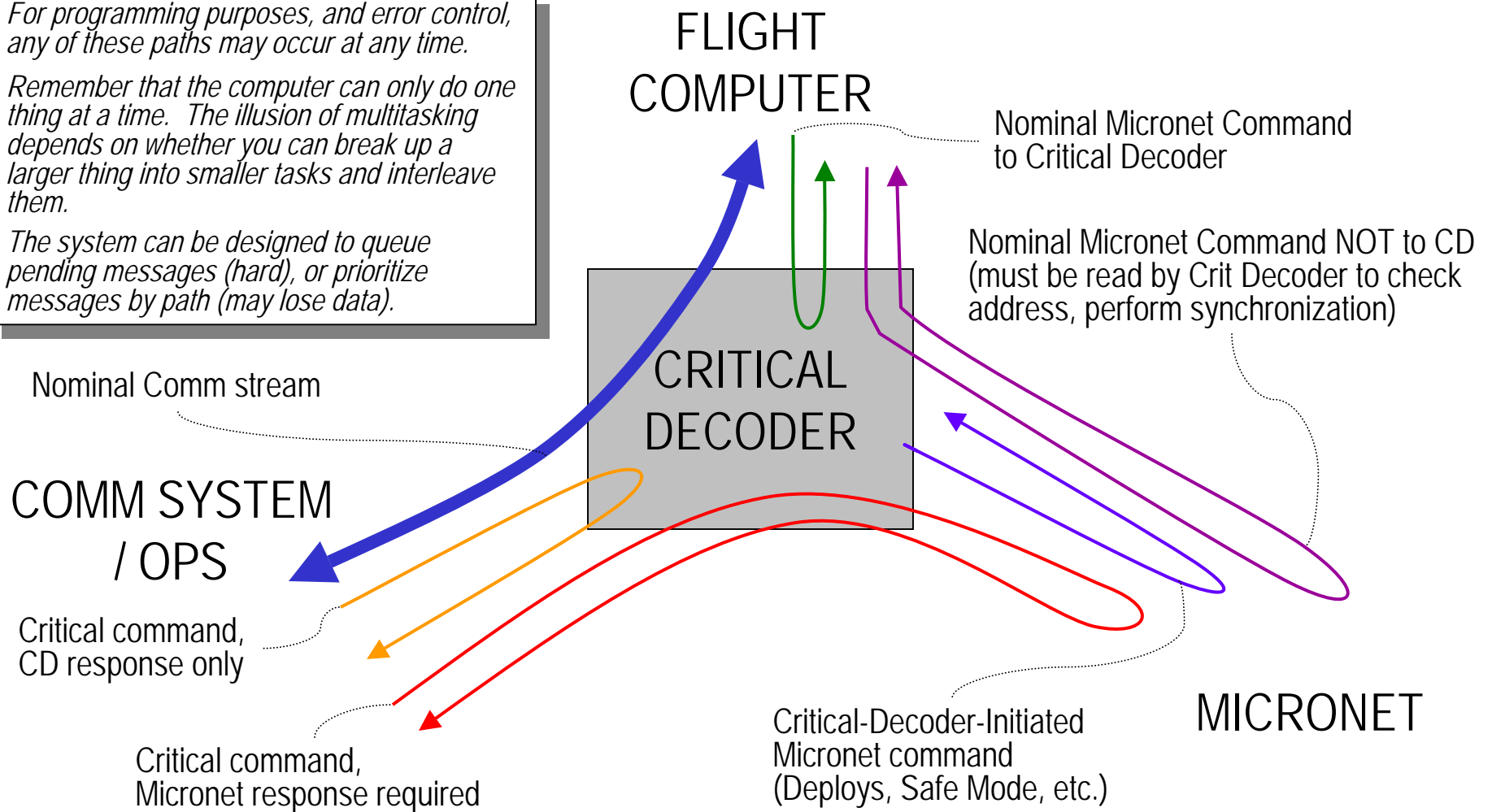
Command and Data Handling System (C&DH)

Possible Data Flows

For programming purposes, and error control, any of these paths may occur at any time.

Remember that the computer can only do one thing at a time. The illusion of multitasking depends on whether you can break up a larger thing into smaller tasks and interleave them.

The system can be designed to queue pending messages (hard), or prioritize messages by path (may lose data).



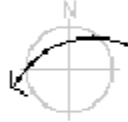

- ▶ 9600 baud problems
 - ▶ Remember that Receive interrupt on UART has higher priority than Transmit
 - ▶ If “too many” characters are coming in (from Flight Computer), Transmit will never occur
 - ▶ Hardware handshaking (“pause”) will not help, this is a UART hardware issue
 - ▶ This may be mitigated by faster code or slower data rates
- ▶ Lockup
 - ▶ After “random” amounts of time, “Protocol Error” interrupts occur on HDLC channel B
 - ▶ Code is currently not using channel B, should be disabled
 - ▶ “Protocol Error” indicates that channel B got bad data, but input lines, clock grounded
 - ▶ Such errors should be handleable, but chip remains locked
 - ▶ Even power-up reset procedures do not unlock chip
 - ▶ Is this a “real” interrupt from the HDLC device?
 - ▶ Bus contention, line noise, bad interrupt code could be “simulating” the above condition
- ▶ Solutions?
 - ▶ Faster (Assembly), cleaner code
 - ▶ More attention to interrupt architecture, priority, possible double-update issues
 - ▶ Hardware fix: split into multiple processors (synchronization issues), faster processor
 - ▶ Or fly TNC, with attendant new issues (known and unknown).




- ▶ CD132.PPT
 - ▶ NMIL-0016 Documentation
 - ▶ Siemens / Infineon 80C535
 - ▶ Micronet I/O List
 - ▶ System Schematic / EIAD
 - ▶ System Safety Diagram
 - ▶ DDS Micro C Development System Docs
 - ▶ Siemens / Infineon SAB82525 Manual
 - ▶ National PC16550 Manual
 - ▶ Micronet Design Document
 - ▶ Critical Decoder Design Document
- Crit Decoder Schematics
 - 8051 Processor Implementation Info.
 - 8051 Programming Information
 - Hardware Interfacing Details
 - Wiring Harness / Connector Details
 - Important GSE Information
 - C, Assembly Programming Environment
 - HDLC Details
 - UART Details
 - In Progress
 - In Progress



MIR approx. visual mag = -0.5

Date (mm/dd)	Rise	Set	Duration (mm:ss)	Sun Lit (mm:ss)	Max. Elevation	Aprox. Image
12/16	06:20:05am NW	E	09:02	04:14	22 NNE	
12/16	07:54:38am WNW	SSE	09:11	07:59	29 SW	

STATION approx. visual mag = -0.5

Date (mm/dd)	Rise	Set	Duration (mm:ss)	Sun Lit (mm:ss)	Max. Elevation	Aprox. Image
12/16	06:04:25pm WSW	NE	10:20	08:20	36 NNW	

Pass info generated (and emailed!) by the J-Track project at
<http://liftoff.msfc.nasa.gov/>

Image of Mir and Shuttle Atlantis taken 9/96 by [Ron Dantowitz](#) at the Boston Museum of Science using an 12" telescope and special tracking/slewing software...

