COMMAND AND DATA HANDLING AND SOFTWARE DEVELOPMENT

Brian Sanders
Colorado Space Grant Consortium

brian.sanders@colorado.edu
May 31, 2013

C&DH System from 3CornerSat circa 2000
Presentation Purpose

• Cover the terms and concepts for a practical design of Command and Data Handling and the compliment of Software Development
• Identify common tools and practices
• Based largely on C&DH and Software Development efforts seen in student satellite programs
• Focus is to help a largely non-embedded team understand and plan for the system typically forgotten
• Not all of what is listed here is needed for every mission. Scope appropriately. Does it trace to a higher level requirement?
• USER WARNING
  – Software can have some of the largest mission scope growths of a satellite system development because it can be largely unseen. The software requirements at CDR represent ~50% of the needed functionality if not adequately defined.
  – Software requirements should be written so that a programmer with no “space” experience can deliver the correct functionality.
ISS C&DH Architecture

C&DH MIL-STD-1553 Architecture
- POST Assembly Complete (Visiting Vehicles)

For Reference Only

Notes:
1. There are two ICAS inside one ICAS GRU. Each internal ICAS connects to two buses, bus connections depend on CEV's docking section, PMAP or PMA-3.
2. Node #1 MDM is CEV GNC-1/2 and LE SYS-LAB-1/2 bus controller in Mighty Mouse Scenario.
3. The Orbital Science Vehicle may be attached to the ISS CAS sites or the integrated h i s a ssembly. M-STD-1553 connection provides via LB PL-1 for the backboard CAS sites and LB PL-2 for the Port CAS sites.
The ISS CDH system was never tested all at one time. Some modules were being designed while others were active on orbit. The architecture was based on Shuttle and its power is in the scalability, designed lifetime/reliability, and providing processing power where needed.

<table>
<thead>
<tr>
<th>Key Subcomponent</th>
<th>Key Function</th>
<th>Standard</th>
<th>Enhanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>80386</td>
<td>Microprocessor</td>
<td>12 MHZ</td>
<td>16 MHZ</td>
</tr>
<tr>
<td>EEPROM</td>
<td>Nonvolatile Storage, MDM boot up software storage</td>
<td>1 MB</td>
<td>1 MB</td>
</tr>
<tr>
<td>DRAM</td>
<td>Volatile Storage, CVT and software applications</td>
<td>2 MB</td>
<td>8 MB</td>
</tr>
<tr>
<td>A/D converter</td>
<td>Converts analog signals to digital ones</td>
<td>Present</td>
<td>Present, used as a temp sensor</td>
</tr>
<tr>
<td>BIA</td>
<td>Provides a 1553B bus connection</td>
<td>Present</td>
<td>Present</td>
</tr>
<tr>
<td>Math Coprocessor</td>
<td>Assists the CPU to increase processor speed</td>
<td>Not Present</td>
<td>Present</td>
</tr>
</tbody>
</table>
Why Apollo 11 Landing almost didn’t happen

The final minutes of the Apollo 11 Landing, Alarms 1201 and 1202 were noted due a rarely seen in testing interference in decent hardware sensors. This allowed too much radar data to be processed by the Apollo Guidance Computer (AGC). The computer reset 3 times during one 40 second period. The last reset was within 770 feet of touch down.

102:38:26 Armstrong: (With the slightest touch of urgency) Program Alarm.
102:38:30 Armstrong: (To Houston) It's a 1202.
102:38:32 Aldrin: 1202. (Pause)
102:38:53 Duke: Roger. We got you...(With some urgency in his voice) We're Go on that alarm.

…

102:42:19 Aldrin: Program Alarm. (Pause) 1201
102:42:25 Duke: Roger. 1201 alarm. (Pause) We're Go. Same type. We're Go.

…

102:45:40 Aldrin: Contact Light.

http://www.doneyles.com/LM/Tales.html

Great Reason why C&DH and software development and full system testing is important
Main C&DH and Software Requirements

• What should C&DH do
  – C&DH and Software is a Utility, similar to EPS and COM. It serves the satellite.
  – Monitor the satellite at some level of autonomy to insure unsafe conditions do not persist.
  – Collect data and store satellite data
  – It does not serve the existence of itself or to demonstrate novel software features unless there is a specific mission requirement. This is where requirement creep enters into the picture.
  – Manage Mission Phases
  – Control sub-systems
  – Manage ground communication
Data Budgets

• Data Storage
  – How much data will be collected on orbit

• Engineering Data
  – Utility systems such as COM, EPS, ADCS, THM, etc
  – Min, Max, Average data points for a period might be a good way to reduce data size depending on mission phase

• Science mission Data
  – Maximizing this data is your goal
  – Different Data rates corresponding to mission phases

• Keep data rates up-to-date
• Can drive communication system requirements if H&S, Science, and Overhead data rates are not understood.
  – Since storage is cheap, consider logging data on board and downloading higher resolution data as needed to help debug systems.

• Compression
  – Don’t over estimate compression rates – use real data to baseline. JPEG is lossy.
  – Bzip2 and tarring small files will help storage efficiency and possibly help downlink rates.
Data Storage

- All Memory is not created equally
  - RAM or Memory is fast
    - Volatile Memory used as a temporary scratch pad
  - EEPROM or Flash is slower
    - Non-Volatile Memory used for flight code or program
  - Data Storage: SD Card, Flash or USB stick equivalent.
    - Store mission data before downlinking

- How to protect your flight code from Bit flips?
  - Radiation will fill a transistor well and cause a 0 to 1 or 1 to 0 state change
  - Multiple copies of Flight Code and compare?
    - Two copies will show an inconstancy but three copies could be compared and voted to show a likely truth.
    - Make the system tolerant
      - Checksum and memory scrubbing
      - Is this too complex for our missions? Likely, yes
Resource Utilization

• **CPU Utilization**
  – Amount of time that the processor is doing something
  – Watch this as code is developed
  – Rule of thumb, shoot for 65% or less

• **Bus Utilization, Data Throughput between devices**
  – I2C, SPI, Ethernet
  – Devices with unique addresses
  – Producer/Consumer
    • Example: If C&DH can read at 9,600bps, ACS produces data at 4800bps, and EPS at 4800bps and Science produces at 4800bps there is a problem
      – Don’t use “Bandwidth” to mean data throughput.
Supporting Operations

- **Configuration Files**
  - Gives the team a way to turn the knobs of the satellite during on-orbit operations
  - Pre positioned loads or Configuration Files (XML type for example)
  - Allow a standard PPL configuration for testing and for flight

  - Your “Remove Before Flight” equivalent for software
    - Don’t want to deploy your antenna accidently during testing
  - As opposed to changing source code, for instance variable, this allows for software changes during DITL, testing, and on-orbit without loading new binaries.

```xml
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE dcol-config SYSTEM "dcol.dtd">
<configuration>
  <logfile>/data/log/dcol.log</logfile>
  <loglevel>1</loglevel>
  <logsize>25</logsize>
  <logmult>SINGLEFILE</logmult>
  <monitor_sleeptime>5</monitor_sleeptime>
  <outdir>/data/collected/done</outdir>
  <outdirtempdir>/data/collected/parts</outdirtempdir>
  <epochfile>/home/modes/C/epoch</epochfile>
  <sensorvalspath>/home/modes/C/sensorvals.xml</sensorvalspath>
  <numcols>20</numcols>
  <collector>4 0 2 5000 1 600 eps_0 1 16</collector>
  <collector>4 1 2 5000 1 600 eps_1 1 17</collector>
  <collector>4 2 2 5000 1 600 eps_2 1 18</collector>
  <collector>4 3 2 5000 1 300 eps_3 1 19</collector>
  <collector>4 4 2 5000 1 600 eps_4 1 20</collector>
</configuration>
```

- **Documentation**
  - subsystem is the number (in decimal) of the subsystem to query
  - sensornum is the number (in decimal) of the subsystem’s sensor to query
  - vilen (value vector length) is the number of elements returned within each datapoint from this
  - bpv (bytes per value) is the number of elements returned within each datapoint from this
  - period is the query period, in milliseconds
  - numdptoreq is number of samples to request per period (should be size of subsys buffer)
  - numdptosave is the number of samples to save per file
  - filestr is the file name prefix for sample files
  - beacon_fwd is sending a copy of the data to com_cdh, yes = 1, no = 0
  - beacon_fwd_cnt is the number of data sets between each copy sent to com_cdh
Supporting Operations

- **Beacon Status information**
  - Telemetry/High level status of satellite

- **STK is a great tool to determine pass times, Com mode duration**

- **Confirming the software load**
  - Version ID, or checksum in telemetry

- **Command Counters**
  - Can help verify that uplinked commands are received
  - Can help detect malicious attempts
  - Command Validation – insure the uplinked command is reasonable

<table>
<thead>
<tr>
<th>General</th>
<th>CDH</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CDH Reset</td>
</tr>
<tr>
<td>2</td>
<td>COM Reset</td>
</tr>
<tr>
<td>3</td>
<td>ACS Reset</td>
</tr>
<tr>
<td>4</td>
<td>GPS Reset</td>
</tr>
<tr>
<td>5</td>
<td>Payload Reset</td>
</tr>
<tr>
<td>6</td>
<td>EPS Reset</td>
</tr>
<tr>
<td>7</td>
<td>Undefined Error</td>
</tr>
</tbody>
</table>
Software Life Cycle

• Develop the storyboard first
  – Supports all the mission phases
  • Pre Launch - Development, Software Testing, Satellite Hardware Testing, and Environmental Testing
  • Post Launch - Activation, Checkout, Safe Mode, Science Mode, etc
• Develop the Software FBD
• Break down in to units
• Test each unit called Unit Tests
  – Use completion of each Unit Test to measure real progress
  – Source Lines of Code (SLoC) estimates give a ROM of size but not complexity or functionality.
• Integrate one unit at a time
• Integrate one sub-system at a time
• Build full FlatSat
• Each Sub-System should develop their end to end test.
Software Development

- Software is always the last to be defined
- Tendency is to “fix it in software” or “fix it in operations” during late cycle development
  - Available time can help define critical and non-critical software features
  - Scope by requirements
  - Build the Functional Block Diagram
CDH Topology

- Driving decisions
  - Volume, mass, power
  - Integration and testing
  - Processing and local needs
- Distributed
  - Each team can own their hardware and system. ADCS, EPS, etc
  - Basic functionality common from CDH
  - Communication interfaces
  - Analog or sensor data collection
- Single processor
  - CPU utilization
  - I2C response times can be limiting
  - Sensors on a SPI or I2C bus
  - Threading all subsystem needs onto one device may be challenging
  - ADCS is usually a resource hog
Major CDH components

CDH

- Configuration Memory
  - MR2A16A
  - 4 Mb

- Data Memory
  - SD Cards (2 x 268B)

- GPS Interface Processor
  - ATXmega256A2

- Microcontroller
  - AT32UC3A3256S

- Payload ADC
  - ADS8332

- Payload Digital Sensors
- Payload Analog Sensors

- SD Bus
- SPI

- Power Switches
  - Switched 12V
  - Switched Battery
  - Switched 3.3V
  - Battery

Electrical Interfaces

15 mA (max)
28 mA (avg)
165 mA (max)
10 mA (avg)
100 mA (max)

Processing Solutions – Order of Magnitude

- **CPU – Central Processing Unit**
  - Usually board level system consisting of RAM, ROM, and an Operating System. Single board computer.

- **FPGA – Field Programmable Gate Arrays**
  - Logic gates, AND, OR, NOR, etc
  - Logic Blocks: Black Box functionality such as compression or encryption
  - Programming with Hardware Description Language (HDL)
  - Fast, low power, usually capable of a dedicated purpose

- **CPLD – Complex Programmable Logic Devices**
  - Similar functionality to FPGA
  - Typically less complex, lower functionality, but higher timing capabilities

- **Microcontroller – think Arduinos, PICs, AVR, etc.** Smaller horsepower mini-computers
  - Processing, memory, non-volatile storage
  - Great for collection of data in distribution

- **Allow requirements to drive decision**
  - Power, complexity, personnel software familiarity, utilization
Hardware Selection

- Build your own or Buy and modify
  - Early version of DANDE CDH for testing
- Mission life time and radiation hardness
- SEU – Single Event Upset
  - Soft error, bit flip
- SEL – Single Event Latchup
  - Burning out a gate or other damage
- What device drivers are supported?
- Most missions can’t afford Rad hard
- Older technology = wider features, more radiation tolerant

http://www.esa.int/Our_Activities/Space_Engineering/Radiation_sattellites_unseen_enemy
Operating System and Software Selection

- **Real Time Operating Systems**
  - Important tasks are given a priority level and then scheduled CPU time in relation to their deadline.
  - Meet real time goals such as computing an attitude before it is needed
  - Anything can run in any order – Many execution paths
  - Typically more capable

- **Deterministic Linear Code**
  - Programming a microcontroller
  - C/C++
  - Easier to test due to fewer execution paths

- **Software Language selection**
  - Java, C/C++, Python, Perl, an endless list
  - Something that is taught as part of your institutions curriculum as a sustainable language that future students will know.
Failure, Detection, Isolation and Recovery

• Watchdogs
  – Needs to be serviced, or pat the dog, on a pre-determined interval to insure the system is operating well
  – Hardware
    • Off chip/CPU dedicated device which needs to be serviced. If not, then power is removed or reset line to processor is activated.
  – Software
    • High priority task running on the CPU. This method doesn’t protect against total processor latch-up.
    – Finding the right level of Monitoring is hard.
    – Reset just the CPU, entirety of C&DH, or the entire satellite?
• Other internal software monitoring which might monitor satellite component temperatures, CPU loading, etc and force the satellite to a “Safe” mode to conserve power or protect hardware.
Common Software Development Solutions

• Software Development is usually the last system to start and last to finish
  – The software development team will want to start writing code immediately. Resist this temptation.
  – Build the FBD, software requirements, prototype low level hardware interfaces and start by building the core software infrastructure.
  – Build confidence with hardware interfaces by using off the shelf systems such as an Arduino to prototype interfaces.
  – Develop small functionality before integrating into the subsystem.
  – By working the larger FBD, multiple parties can develop simultaneously by respecting the ICD.
  – Rank software features in order of necessity before implementation.
FlatSat Overview

- 3 electronics board levels
  - **Brass:** For sandbox / first run testing. Keep many versions as they are great for testing.
  - **SILVER:** Identical to flight boards
  - **GOLD:** Flight electronics boards. Conformal coated
    - **GOLD SPARE:** Identical to Gold boards. Flight Spares
- Silver Flat-Sat to be used for initial system integration and testing
  - Identical to flight boards, except for conformal coating
  - Arranged on static mats, no subsystem boxes
  - Allows for rapid testing and debugging of full system for initial integration and testing
  - Lessons learned and problems fixed with silver flat-sat will speed up flight system integration
  - Great or mission training and dry run during operations
The EGSE is the often forgotten system

- Typically providing power and safety interfaces to the satellite (fused power).
- Fundamental Computer interface
- Support Radio Com
Testing Scripts

• Build scripts to test each subsystem
  – Quick and easy way to insure subsystems haven’t broken
  – EGSE commanding CDH, which would send commands or read from subsystems, stimulus and thereby confirm proper operation
  – Checking against acceptable range
• Build a larger test script which test the entire software suite
  – At right is the DANDE aliveness check
  – Note that 189 of 189 steps passed
    • This rarely happens
• Plan for Day in the Life and plenty of “Burn In Time”.
• Great way to find memory leaks

Clearing data buffer, sleeping 5 seconds...
Data buffer 7 length - E: Length 36 A: 000A008008070100015F015D015DFF Pass
clbm 80 GETDATA 7 1 - A: 015F015D015D
DP0 - Lrange: -160 <= Val: 351 <= Urange: 800 Pass
DP1 - Lrange: -160 <= Val: 349 <= Urange: 800 Pass
DP2 - Lrange: -160 <= Val: 349 <= Urange: 800 Pass

Clearing data buffer, sleeping 5 seconds...
Data buffer 8 length - E: Length 36 A: 000A008008080100339C0001E001A00190FF Pass
clbm 80 GETDATA 8 1 - A: 01E001A00190
DP0 - Lrange: -640 <= Val: 480 <= Urange: 800 Pass
DP1 - Lrange: -640 <= Val: 416 <= Urange: 800 Pass
DP2 - Lrange: -640 <= Val: 400 <= Urange: 600 Pass

Clearing data buffer, sleeping 5 seconds...
Data buffer 9 length - E: Length 36 A: 000B00800809010039DD00015CFF Pass
clbm 80 GETDATA 9 1 - A: 015C
DP0 - Lrange: -160 <= Val: 348 <= Urange: 800 Pass

Subsystem Aliveness Verification Complete! Press [ENTER] to display the results.

*************  Results: *************
189 total tests were run.
0 of 189 tests failed.
189 of 189 tests passed.
*************  Results: *************
Software Tools

• Content Management System
  – Subversion (SVN) is a great tool
  – Keeps track of code changes
  – Manages releases
  – Keeps configuration files
  – Check in CDH hardware or all system schematics


```plaintext
dande - Revision 3556: /src

- .
- ACC/
- ADC/
- CDH/
- CDH_releases/
- COM/
- EPS/
- NMS/
- PCTest/
- README.txt
- THM/
- conf/
- db/
- format
- gui/
- hooks/
- locks/
- subsystem/

Powered by Subversion version 1.6.6 (r40053).```
TRAC ticket management

- Software Configuration Management and Project Management
- Open Tickets for team members to do work
- Wiki based with hooks into SVN
- TRAC management is appropriate during testing
Ticket #1 (closed defect: fixed)

**com_cdh comes out of beacon silence early after being killed by pdog**

- **Reported by:**
- **Priority:**
- **Component:**
- **Keywords:**
- **Owned by:**
- **Milestone:**
- **Version:**
- **CC:**

Opened 17 months ago
Last modified 17 months ago

**Description (last modified by rowbrma) (diff)**

Discovered during the DITL 1/8/2012 dry run.

To reproduce:
1. Flash the satellite and boot it.
2. Check the process list on the satellite and verify that there is exactly one thread of com_cdh running.
3. Wait 30 minutes.

Expected response:
1. Satellite beacons within 32 minutes of power on (2 minutes for com_cdh startup and first beacon).
2. Check the process list on the satellite, 7 threads of com_cdh should be visible.

Actual response:
1. Satellite beacons 3 minutes of power on
2. There are 7 threads of com_cdh when there should only be one.
Questions?
Additional References