TERMA

ALLIES IN INNOVATION

OBC INITIALISATION SEQUENCE AND ATB REMOTE ACCESS

Mariarosaria Cardone, Project Manager
Dr. Mattias Holm, Technical Project Manager
Ignacio López, System administrator
Miquel Català, Sim & validation engineer
Yi Tan, Senior engineer
Michael Larsen, Senior engineer

Outline

- Introduction to the project
- Remote Access to the Avionics Test Bench at ESTEC
- The Initialisation Sequence and the OBC Initialisation Converter
- The TSIM like interface to the ESOC Emulator 2.0
- Feedback to SAVOIR

© The Terma Group 2013

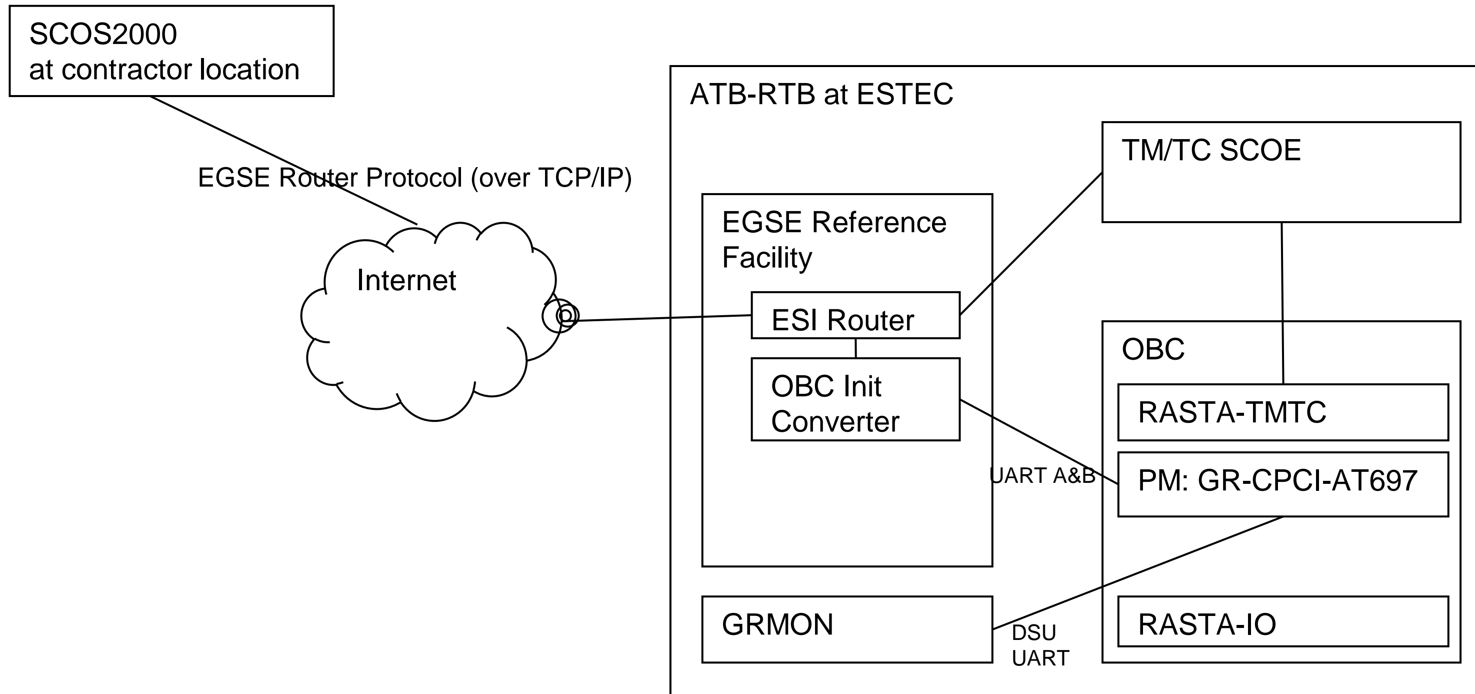
Introduction

- **SAVOIR group has been working on providing generic building blocks for reusable avionics.**
  - Flight Computer Initialisation Sequence – ESA Requirements
  - Implement Initialisation sequence for the ATB
  - Feedback to SAVIOR for improving generic initialisation sequence requirements
- **ATB is installed at ESTEC Avionics Lab**
  - Long colocations for contractors
  - Difficulties in getting support for problems with the board when e.g. the PM board is located at a contractor's site.
  - Remote access with SCOS connecting to the ATB remotely via the Internet would help.
- **ATB in its SVF configuration supports multiple emulator cores**
  - No support for running with ESOC Emulator 2.0

REMOTE ACCESS

© The Terma Group 2013

Introduction: Remote Access



- **Currently SCOS2k is in the EGSE reference facility at ESTEC.**
- **If SCOS2k would be moved outside ESTEC, situation would improve for contractors working with the system.**
- **SCOS2k users can stay at home…**

© The Terma Group 2013

Remote Access

- **Easy setup for SCOS2k to connect to remote ESI Router:**
  - Modify variables EGSE_EGW_SERVER and EGSE_EGW_SERVER_PORT on MISCconfig configuration file.
  - IP of CCS/SCOS must be resolvable through reverse DNS.
- **VPN Software provided by ESTEC.**
  - VPN Blocks all Other Network Connections
  - Run either with physical console or VM, not remote.
- **Terma used a local LAB to test the system**
  - Simulated VPN connection (i.e. firewall blocking everything but relevant ports)
  - Network reliability has been explored in local lab
- **Remote work normal scenario**
  - Remote user configure CCS system for the external ESI router and port
  - User start ESTEC VPN software
  - CCS work can be carried as normal.

© The Terma Group 2013

Remote Access

- **When using VPN, network reliability and latency could be an issue.**
- **CCS checkout system has been designed for local connectivity with the router. When CCS + VPN connection ends abnormally, human intervention may be need.**

- **Problem scenario:**
  - VPN is connected
  - CCS -> ESI connection established.
  - VPN is disconnected (ESI is not properly notified about disconnection).
  - CCS session is closed by user or unexpected failure.
  - VPN is re-established
  - New CCS -> ESI connection is tried
    - ESI router see a running connection and a trial from the same computer to reopen a running session.
- **Workaround: Script that scans ESI router log files for session lockups and tries to recover the situation without human intervention.**
  - Sends close signal, to ensure ESI router terminates the TCP connection.
  - Allows reconnection afterwards.
- Remember: With remote access the contractor does not have the ability to trouble shoot issues on the ESTEC side with e.g. the ESI router.

© The Terma Group 2013

Remote Access

- **With remote access the contractor does not have the ability to trouble shoot issues on the ESTEC side with e.g. the ESI router.**
    - Plausible problem:
        - Enable and disable commands are used in sequence
        - Disconnection occurs between
        - Disable command will either not be sent, or it will be delayed
        - Ensure the remote on-board-software have timed commands for critical functions that automatically disables themselves (e.g. fire thrusters in 20 seconds for 5 seconds).
- **VPN connection with ESTEC tested.**
    - Acceptance tests with the initialisation sequence in the loop executed successfully over the VPN connection with ESTEC.
        - Enabling the initialisation sequence standby mode
        - Usage of initialisation sequence standby mode
        - Invalid telecommands
- **Further improvements would include**
    - CCS and ESI have different timeouts that can be aligned.
    - Analyse CCS-ESI router protocols to ensure proper session handling (e.g. by providing usernames and passwords)
    - Safe-Killing switch for the remote platform to bring all hardware components on safe state.

© The Terma Group 2013

TERMA
ALLIES IN INNOVATION

THE OBC INITIALISATION SEQUENCE

© The Terma Group 2013

Introduction: GR-CPCI-AT697

- **Compact PCI board with an AT697 LEON2 chip**
  - 80 MHz
  - 128 MiB accessible SDRAM (SDRAM chips replaceable)
  - EDAC support for SDRAM
- **GRMON connects to the board via serial cable**
- **Two serial ports routed to UART A and UART B on the AT697 chip.**
- **Flash memory soldered to the board. Two 16 bit wide Intel flash chips following the JDEC CFI standard. 16 MiB flash memory total.**

© The Terma Group 2013

Introduction: Initialisation Sequence

- **Boot Software**
  - First thing that executes
  - Similar to BIOS, EFI, UBOOT, OpenFirmware, etc.
  - Common features:
    - Hardware initialisation
    - Maintenance mode (e.g. command line for EFI and OpenFirmware, menu interface in many BIOSes)
    - POST: Power-On Self-Test
    - Boot loader (at least the first stage boot loader)
- **Initialisation sequence / boot software for the ATB**
  - First thing that boots
  - Initialises the hardware
  - Maintenance mode called the “standby mode”
    - Patch, dump and check memory
    - Flash unlock, erase and unlock
  - Hardware self-tests (and diagnostics in boot report)
  - Boot loader (loads ELF files from a simple flash memory file system)

© The Terma Group 2013

Initialisation Sequence: Limitations

- **The OBC in the ATB-RTB has limitations w.r.t. a real OBC**
  - No redundancy
  - No reconfiguration module
  - No relay configuration by HPC/ground
  - No SGM
  - Boot PROM and EEPROM in single FLASH memory.
- **Consequently:**
  - Standby mode enabling using 2 conditions:
    - Preferred: warm start and relay bit set by HPC
    - Now: warm start and ‘s’ in UART B’s data register
  - SGM writes and initialisation functions implemented as dummies in the BSP. Called by the boot report functions.
  - Redundant computer (which drives the standby mode) emulated by the OBC Initialisation Converter tool.

© The Terma Group 2013

The Initialisation Sequence: Design (1/3)

- **Implementation languages**
  - Assembler
    - Trap handlers
    - CPU initialisation
    - Mode detection (nominal or standby)
    - Instruction cache test
    - Memory test
    - Stack initialisation
  - C99 (requires a working stack for SPARCv8 SysV ABI compliance)
    - Boot reporting
    - Other hardware tests (may contain inline assembler)
    - File system code
    - ELF loader/parser
  - ESOC Emulator Test Harness Scripting Language
    - White box unit test
  - UNIX shell scripts
    - Test harness (starting unit tests, handling test reports etc.)

© The Terma Group 2013

The Initialisation Sequence: Design (2/3)

Start → CPU Init → Mode Detect

Standby Mode

- Stack Init
- Mount File System
- Load Standby App

Standby Mode Application

- Wait for TC
- Handle TC

Nominal Mode

- Icache Test
- RAM Test
- Stack Init
- Boot Report Init
- Write Icache and RAM reports
- Execute Self-Tests and Device Inits
- Mount File System
- FSCK
- Load Selected ASW
- ASW Takes Over

Languages

- Asm
- C99

© The Terma Group 2013

Initialisation Sequence: Design (3/3)

- **Initialisation sequence is independent from runtimes and operating systems**
  - Own implementations of everything needed by a C-compiler (e.g. memcpy) and standard library needs (e.g. stdint.h, string.h etc.)
  - Loads ELF-files
    - Load information in the program headers of the ELF file
    - Entry point in the ELF header
- **Source organised for portability**
  - src/arch/sparc-v8 for specific common SPARC code
  - src/bsp/gr-cpci-at697 for code specific for the board.
- **Linker scripts**
  - One for including (defining locations of init sequence constants and addresses) by ASW
  - One for linking
- **Build system**
  - Plain GNU make
  - Very little to configure, just works

© The Terma Group 2013

Initialisation sequence: Self Tests

- **Power-On Self Tests for**
  - I-Cache (tags and lines)
  - RAM:
    - Address bus (often overlooked)
    - Data bus (often overlooked)
    - Content (including EDAC bits)
  - D-Cache (tags and lines)
  - Interrupt controllers (forced IRQs)
  - UARTs (loopback)
  - Timers (enable timer and check if IRQ taken)
  - PCI interrupts (force PCI interrupts)
  - Register EDAC (diagnostics)

© The Terma Group 2013

Initialisation Sequence: File System (1/3)

- **Virtual file system layer for boot software and one concrete file system implementation:**
  - Virtual file system layer mounted in stack memory when system is executing C-code.
  - Supports
    - Directories (read only)
    - Files (read only)
  - eboot_vnode_t represents anything in the file system (directories and files).
    - Pointer to virtual function table installed by the file system
    - All node operations using a common interface are dispatched to the file system implementation.
    - eboot_mount(eboot_fs_t *fs, eboot_vnode_t *root): mount file system
    - eboot_fsck(eboot_vnode_t *fsroot): File system consistency check / integrity check
  - eboot_file_t represents an open file, file operations inspired by POSIX
    - eboot_open(eboot_vnode_t *vnode, eboot_file_t *file)
    - eboot_read(eboot_file_t *file, void *dest, size_t len)
    - eboot_lseek(eboot_file_t *file, eboot_off_t offset, int whence)
    - eboot_close(eboot_file_t *file)

© The Terma Group 2013

Initialisation Sequence: File System (2/3)

- **Very simple file system in flash memory**
  - Does not support writes to files
  - Does support:
    - Multiple application binaries
    - Standby application as separate app
    - CRCs for file system consistency
  - Layout
    - Flash blocks are 256 KiB
    - Boot SW installed in flash block 0
    - File system root directory in flash block 1
      - Directory and file CRCs (including boot sw and standby app)
      - Number of files
      - File info: CRC, size, location
        - Metadata: CRCs for expanded ELF program loading segments
    - Standby application in flash block 2
    - ASW 1 in flash block [3, k]
    - ASW 2 in flash block [k+1, m]
- **Very easy to add additional file systems if needed**

Initialisation Sequence: File System (3/3)

- **Tool for generating the file system images implemented**
  - `mkfs.eepromfs boot-sw.elf standby.elf asw1.elf asw2.elf 0x00040000 [corrupt]`
  - Hex value specify the address where the root directory tables should be placed
  - Optional corrupt-command added for testing purposes, inverts all CRCs in the file system
  - Generates a binary file system image without the boot software (boot software is linked to the FS image with a file entry though)
  - Binary image is converted to SREC with objdump (automatically started by the tool)

© The Terma Group 2013

The OBC init converter

- **Connects the ESI Router to the GR-CPCI-AT697 board**
- **ESI router to OBC Init Converter connects using the EGSE libraries via TCP/IP**
- **OBC Init Converter connects to the GR CPCI board over via two serial cables connected to the LEON2 UARTs.**
- **Simple ASCII based TM/TC protocol between the board and the converter.**
  - Patch
  - Dump
  - Crc
  - Reset
  - Enable / disable standby mode
  - Flash memory lock, unlock and erase
- **Written in Python 3**
- **MIB-database updated with additional TCs to reset the board, manage standby mode and flash memories.**

© The Terma Group 2013

Initialisation Sequence: Testing and Validation

- **Unit testing boot software not practical in traditional sense**
  - White box unit testing performed using the ESOC emulator test harness (EMTH).
  - Emulates failed hardware by:
    - Injecting unexpected characters on the UARTs
    - Modifying the read values from registers and memory
- **Validation tests executed on the board (without remote access)**
  - Load custom application software that prints messages on the UARTs (e.g. dumping the boot report)
  - Start standby mode by writing ‘s’ to UART B
  - Positive cases tested (i.e. no failing hardware, no way to really inject hardware errors in a predictable way in the board).
  - Performance test:
    - Boot time without instruction caches enabled before the RAM test: ~120 seconds
    - Boot time with instruction caches enabled before the RAM test: ~25 seconds
- **Acceptance test plan based on full system**
  - SCOS in loop, using the board, using the init sequence
  - Custom application software
  - Focus on testing end user functionality:
    - Activation of standby mode
    - Dumping the boot report
    - Selecting application software

© The Terma Group 2013

Remote Access Demo Scenario

- **Based on acceptance test scenarios:**
  - Execute a maintenance cycle (with flash memory) using the initialisation sequence.
  - Using the application software from the acceptance testing.
  - Sequence:
    - Program the board's initial software
    - Start boot software
    - Startup SCOS, ESI router and converter
    - Enable standby mode
    - Unlock flash block
    - Erase flash block
    - Write patch value to flash block
    - Select ASW 2
    - Reboot
    - Inspect memory modified by ASW 2 (using patch in flash memory)
  - All steps executed from SCOS except initial programming of flash memory and starting the boot software.

© The Terma Group 2013

The Initialisation Sequence: Problems along the way (1/2)

- **UML as a tool for boot software (and operating systems)**
  - Sequence and activity diagrams are useful
  - Remaining UML is hard to map to low level mostly sequential software
  - Nothing that gains from object oriented design in boot software (except perhaps file system support)
  - Structured software engineering methods and tools for this type of software…
- **Unit testing boot software / operating system kernels**
  - Not practical in the traditional sense, too much direct access to hardware

    Solution: Unit tests are white box tests implemented as emulator scripts (Can fake hardware failures easily)
- **Flash memory on the GR-CPCI-AT697**
  - Interleaved 2x16 bit flash chips:
    - The CFI flash command 0x12:
      - Per chip: 0x1200 (SPARC is big endian)
      - Write both chips at once (is a must for erasing and writing): 0x12001200
  - No documentation available (except the individual chip docs and the CFI standard from JDEC)
- **Register EDAC selftest**
  - Implemented according to LEON2 documentation (did not work)
  - CPU errata for the AT697 and the register EDAC diagnostics system (oops!)

© The Terma Group 2013

The Initialisation Sequence: Problems along the way (2/2)

- **Bricked board**
  - At one point the board appeared to have become bricked (shortly before the CDR delivery)
  - GRMON was unable to connect
  - Problem: boot-sw was writing the MEMCFG registers early in the boot process and used invalid values

    Workaround: push reset and break buttons at the same time, board then breaks before starting to execute instructions and GRMON can connect before the MEMCFG values are written
- **GRMON does not support the uploading of blobs only ELF and SREC**
  - Used GNU objcopy to generate SRECs from the file system images.
- **LEON2 / AT697 documentation for ASI instructions**
  - Very difficult to read and understand how to address the instruction and data caches with diagnostic ASI access.
- **sparc-elf-gcc (binary releases)**
  - 3.4.4 linker does not support the EXCLUDE_FILES linker script directive and does not complain about it.
  - 4.4.2 compiler does not run on some Linux versions due to dynamic linking to absent libraries.
  - Mixed (unsupported) mode added to build system which can be enabled if needed: i.e. use 3.4.4 compiler and 4.4.2 linker (don’t use this mode for final builds)

© The Terma Group 2013

TERMA
ALLIES IN INNOVATION

THE ESOC EMU 2.0 IN THE ATB

© The Terma Group 2013

Introduction: ESOC Emulator 2.0

- **ESOC Emulator 2 is the evolution of the previous ESOC emulator.**
  - Drops the MIL-STD-1750A CPU support
  - Adds LEON2 and LEON3 support
- **High performance on the x86-64**
  - LEON2 runs in realtime
  - Previous backends still around (MIPS, Alpha)
- **The emulator is written in a CPU independent macro assembler**
  - Host specific backends generate the cores for the relevant target
- **User/integrator is responsible for tailoring the emulator**
  - Exposing APIs to the simulators
  - Defining I/O decoding mechanisms
- **Common devices (e.g. LEON2 on chip registers) are emulated in the core**
  - Memory accesses to external devices are dispatched by target specific code.

© The Terma Group 2013

ESOC Emulator in the ATB

- **Integrated by implementing the TSIM API**
- **ATB/SVF is not 64 bit clean**
  - One crashing issue
  - At least one silent problem that resulted in failures for the EagleEye integration tests
- **ESOC Emulator 2.0 is very flexible**
  - Supporting the TSIM API (a foreign emulator's API) is straight forward
  - The other way around would be very hard
- **TSIM API relies on string parsing**
  - Room for improvement…
  - Easy to integrate new emulators by wrapping the API but one may loose functionality or performance as the native emulator APIs may have other assumptions
    - E.g. the ESOC emulator have facilities for integrating devices and address decoding of them in the core.

© The Terma Group 2013

SAVOIR Feedback

- **Very clean set of requirements**
  - Programmers with the right knowledge can go from baseline requirements to an implementation directly
- **Possible improvements**
  - Instruction caches should be tested and enabled before the RAM test
    - Code execution from PROM / Flash is slow.
    - Speeds up boot times around 10 times, from 200 to 25 seconds
  - Testing the instruction cache first appears to violate requirements with respect to boot reporting
  - Additional remarks in report
    - OBC Initialisation Sequence and ATB Remote Access: Final Report (terma/sdp/63/final/rep/1001)

Delivered Items

- **User manuals:**
  - Remote Access
  - EMU 2 TSIM API
  - Initialisation sequence (and converter)
- **UML model for boot software**
- **Software**
  - Boot Software Code
    - mkfs tool code
    - Unit tests
  - Tailored ESOC emulator code
  - TSIM API code for tailored emulator
  - Converter tool code
- **Updated MIB database**
- **Other documents**
  - Testing plans and procedures
  - Testing reports
- **ECSS-E-ST-40 quality documents**
  - TERMA C99 coding standard

© The Terma Group 2013

Meet us at…

www.terma.com

www.terma.dk/press/newsletter

www.linkedin.com/company/terma-a-s

www.twitter.com/terma_global

www.youtube.com/user/TermaTV

TERMA

ALLIES IN INNOVATION