Testing HW/SW Interfaces in the As-Built System

ivv-itc@lists.nasa.gov

http://www.nasa.gov/centers/ivv/jstar/ITC.html

Presenter
Steven Seeger
sseeger@mpl.com

November 9, 2012

Agenda

- NASA's IV&V Program
- Independent Test Capability (ITC)
- ITC Simulation Architecture
- Hardware Overview
- Project Work/Issues Found
- Conclusion

Independent Verification and Validation (IV&V) Facility

- IV&V grew out of the Challenger accident in 1986
- IV&V implemented on the Space Shuttle Program
- NASA wanted to further expand IV&V
  - Technical Independence: Achieved by IV&V personnel who use their expertise to assess development processes and products independent of the developer
  - Managerial Independence: Requires responsibility for the IV&V effort to be vested in an organization separate from the organization responsible for performing the system implementation
  - Financial Independence: Requires that control of the IV&V budget be vested in an organization independent the of development organization

Independent Test Capability

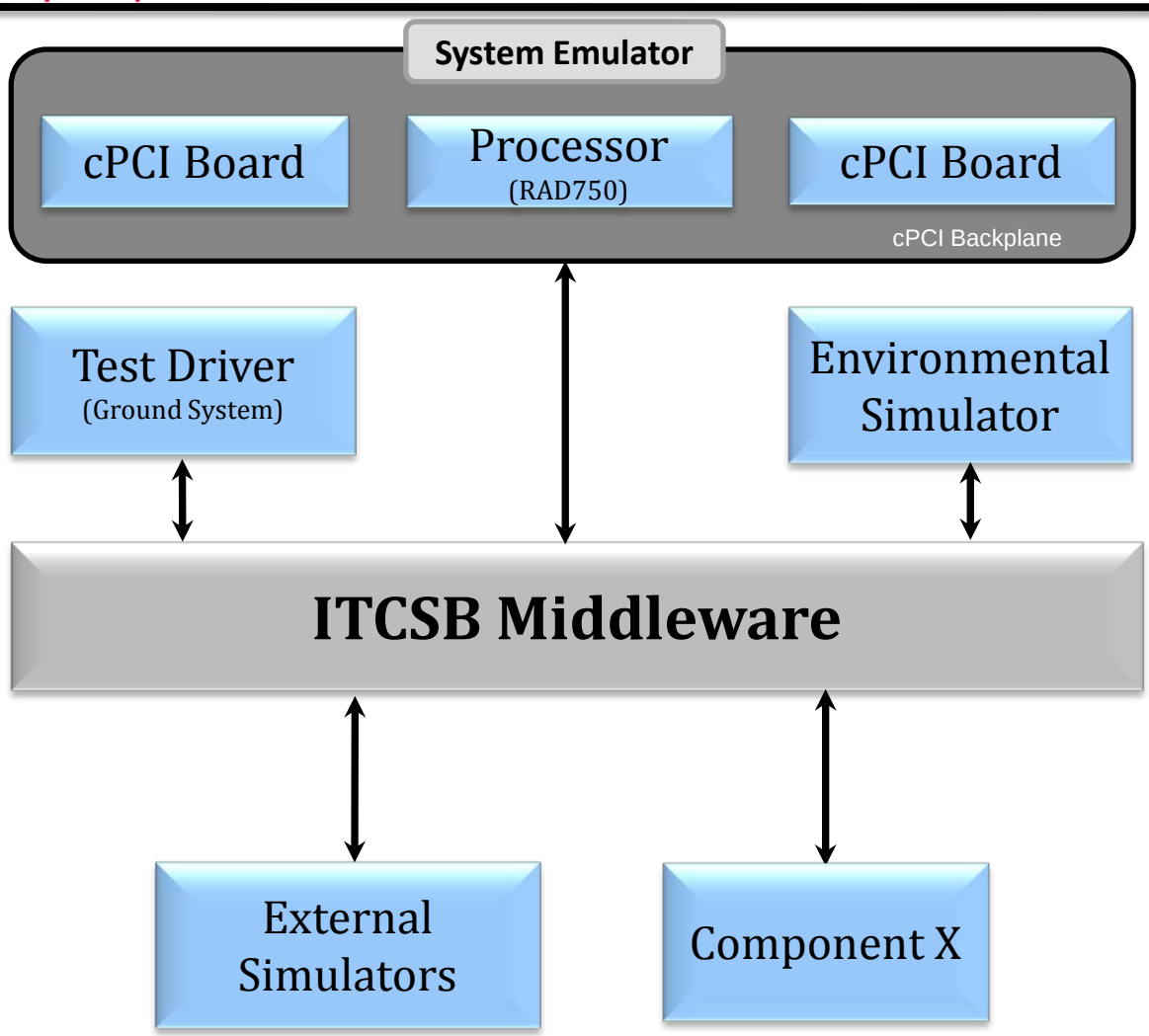
- ITC team is chartered to acquire, develop, and maintain test environments for NASA's IV&V Program to enable the dynamic analysis of software behaviors for multiple missions.

- ITC team provides tools for IV&V analysts to perform independent testing
  - Develop simulators
  - Acquire and support hardware for IV&V
  - Acquire and integrate existing simulations

Dynamic vs Static

- Static code analysis syntactically verifies code
  - Language rules verified
  - Issues identifiable at compile time are discovered
- Dynamic analysis semantically verifies code
  - Behaviors are verified
  - Issues at runtime are discovered
- Dynamic analysis yields far fewer false positives than static analysis
- Approaches complement each other

ITC Simulation Architecture



Hardware/Software Interaction

- Flight software accesses hardware
- FSW typically uses memory-mapped registers
  - Registers on external boards (cPCI, VME, etc)
  - 1, 2, or 4 byte access
- Hardware accesses can also be discrete lines
  - General Purpose Input/Output (GPIO)
  - CompactPCI (cPCI) discretes

Hardware/Software Interface Control Document (ICD)

- Access to hardware defined in a HW/SW ICD
  - Define memory map
  - Describe registers
    - Bits
    - Fields
- ICD is a hardware "user manual" for software
- Informs software engineer of the right way and sometimes the wrong way to do things

Hardware Modeling

- ITC team provides executable environment for software binaries
  - Emulated hardware written as software
  - NASA-specific hardware models created by ITC

- Re-creation of hardware interfaces from ICD
  - Memory mapped IO, registers, etc.
  - Behavior modeled from description in ICD

- **HW/SW ICD rules checked in the model at very minimal cost**
  - **Regression testing**
  - **ICD verification**

Method

- When modeling hardware, behaviors should be driven by the ICD
- Preconditions for some operation as defined in the ICD should be checked by the model
- Log any undefined behavioral states as defined by ICD
- If the ICD defines behavior outside the level of detail for the simulation, it can be ignored
  - If behavior affects pre or post-conditions it must be minimally implemented to perform these checks
  - Not all timing requirements need to be modeled, but if the ICD makes a statement like “this bit must not be set until 1 ms after that bit” then timing must be taken into account

Benefits of Approach

- Can detect correct or incorrect use of hardware as described in ICD
- Undefined behavioral states are detected
- Regression testing “for free”
- Examples
  - Use some bank of memory unless a problem detected
  - Do not access some register before hardware indicates completion of some operation
  - Only write certain bit patterns in a field
- Verifies FSW follows rules correctly

Case Study

- ITC is developing a project simulator
  - Rad750 CPU + cPCI backplane
  - Board A
  - Board B
  - Board C (x3)
- Core of work is modeling the three types of boards
- Boards created entirely from several ICDs
- During the course of development three major issues were discovered in the FSW

Issue 1 Introduction

- ICD describes interface to Summit 1553 chip
  - MIL-STD-1553 (A or B) is structured communication bus commonly used in aerospace
  - Bus consists of a single bus controller and several remote terminals
- Summit 1553 chip
  - Commonly used part
  - Has several 16-bit registers
- Issue clearing illegalization registers

Issue 1 Part 1

- Board A has two Summit 1553 chips with several 16-bit registers accessed as 32-bit accesses providing two 16-bit accesses at once

Excerpt From ICD:

> The same 2-word concatenation occurs when the IFSW uses the cPCI bus to read from (or write to) the registers that are internal to the DXE5 chips. As shown in Table 6-1 and Table 6-5, a dword read from one of the cPCI addresses used for the DXE5 internal registers returns the contents of a consecutive pair of 16-bit DXE5 registers. And a dword written to one of the cPCI addresses used for the DXE5 internal registers actually affects the contents of a consecutive pair of 16-bit DXE5 registers.

Summit1553.c:

```
/* clear all 16 16-bit illegalization regs with 8 32-bit writes */
                for(i = 0; i < 8; ++i){
                        summitRtRegs->illegalization[i*2] = 0;
                }
```

Issue 1 Part 2

Excerpt From ICD:

Table 6-1. Packed 32-bit Read-Write Accesses for RT DXE5 Internal Registers

cPCI Address	Bits [31:16]	Bits [15:0]
[redacted]	Operational Status Register 1	Control Register 0
	Interrupt Mask Register 3	Current Command Register 2
	Interrupt Log List Pointer Register 5	Pending Interrupt Register 4
	Time-Tag Register 7	Built-In Test (BIT) Word Register 6
	1553 Status Word Bits Register 9	RT Descriptor Pointer Register 8
	Register 11 is Not Used	Register 10 is Not Used
	Register 13 is Not Used	Register 12 is Not Used
	Register 15 is Not Used	Register 14 is Not Used
	Receive SA Illegalization Registers 17 and 16	
	Transmit SA Illegalization Registers 19 and 18	
	Broadcast Receive SA Illegalization Registers 21 and 20	
	Broadcast Transmit SA Illegalization Registers 23 and 22 (not used)	
	Mode Code Receive Illegalization Registers 25 and 24	
	Mode Code Transmit Illegalization Registers 27 and 26	
	Broadcast Mode Code Receive Illegalization Registers 29 and 28	
	Broadcast Mode Code Transmit Illegalization Registers 31 and 30	

Issue 1 Part 3

summit1553Registers.h:

```
typedef struct {
        SuControlStatusReg  controlStatus;
        SuCmdIntmaskReg     cmdIntmask;          /* offset 0x04 */
        SuIntLogIntPendReg  intLogPending;       /* offset 0x08 */
        SuBitTimetagReg     bitTimetag;          /* offset 0x0c */
        SuSrtDescStatusReg  srtDescStatus;       /* offset 0x10 */
        unsigned short      reserved_1[6];       /* offset 0x14 - 0x1e */
        unsigned short      illegalization[16]; /* offset 0x20 - 0x3e */
        unsigned short      reserved_2[16];      /* offset 0x40 - 0x5e */
        unsigned short      reserved_3[16]       /* offset 0x60 - 0x7e */
        unsigned short      reserved_4[16];      /* offset 0x80 - 0x9e */
        unsigned short      reserved_5[16];      /* offset 0xa0 - 0xbe */
        unsigned short      reserved_6[16];      /* offset 0xc0 - 0xde */
        unsigned short      reserved_7[16];      /* offset 0xe0 - 0xfe */
        unsigned long       cardControl;         /* offset 0x100 */
} SuRegs;
```

Issue 1 Part 4

- Unsigned short? That's 2 byte! Not 4 byte. That's a bug!
- The programmer's own comment indicates 8 4-byte accesses, but there are really 8 2-byte accesses that skip every odd-numbered register
- The actual hardware will accept the 2-byte access on PowerPC (PPC)
- The hardware model will not. It threw an error and halted
  - In order to continue the simulation work, either the hardware model must be hacked or the software binary changed to use 4-byte opcodes here instead of 2-byte opcodes

Issue 1 Part 5

- Illegalization registers allow some 1553 traffic to be ignored
- On power on, these registers are set to 0
- Flight Software (FSW) does not currently change them. In fact, the ICD says they are not to be used
- In the case of a power glitch, SEU, or access to these registers via memory load, a soft-reset will not correctly clear them

Issue 2 Introduction

- SpaceWire
  - ESA specification implemented in many NASA projects
  - SPW uses routers to send messages to where they need to go
  - Sender must know where to send messages
- Project has board with SpaceWire and a fault injection register
- An invalid value is written to the register

Issue 2 Part 1

Excerpt From ICD:

Table 4-6. cPCI Address Assignments for [redacted] (continued)

cPCI Address	Name/Description	Reference(s)
[redacted]	Start of Address Space for [redacted] Function	4212 H Table 5-1
	[redacted] Version ID Register	Table 8-3
	Interrupt Enable Register	Table 6-84
	Interrupt Cause & Event Status Register	Table 6-80
	Not Used (RESERVED)	
	[redacted] SpaceWire Link Configuration Register	Table 6-62

DaqxxxPwrCtrl.cpp:

```
brdCPciWrite32( yyyyPwrCtrl.brdCChan[brdCId].builtInTestReg,
        BRDC_LIGHTS_OFF); //Built In Test Register
```

brdCctrldef.h:

```
#define BRDC_LIGHTS_OFF 0x3F
```

Issue 2 Part 2

Independent Test Capability

brdCCtrl.cpp:

```
// XXXX Registers Offset
xxxxRegBaseAddr = brdCBaseAddr + FCSI_REG_OFFSET;

// YYYY Built-In Test (BIT) Registers
brdCChan[brdCId].builtInTestReg   = brdCBaseAddr + YYYYBIT_REG_OFFSET;

▪▪▪
//Built In Test Register
brdCChan[brdCId].builtInTestReg = xxxxRegBaseAddr + BUILTINTEST_REG_OFFSET;
```

These are in the same scope! First value is never used, so it is clobbered.
brdCBaseAddr depends on which brdC board is being worked on. (brdC_1 0xc0000000)

brdCctrldef.h:

```
#define  XXXX_REG_OFFSET         0x6000000
#define  YYYYBIT_REG_OFFSET      0x7000000
#define  BUILTINTEST_REG_OFFSET  0X0010
```

First setting is 0xc7000000. This appears to have been an address for a test port. It is not defined in the ICD!

Second setting is 0xc0000000 + 0x6000000 + 0x10…spacewire configuration link reg!

Issue 2 Part 3

Excerpt From ICD:

Table 5-5. SpaceWire Link Configuration Register (0x000010)

R/W	Bits	Description	Default
[redacted]			
W	2:0	Parity Error Enable – inverts a bit received on SpaceWire port to cause a parity error. "101" => Generate Parity Error on Link Rx Data. "110" => Generate Parity Error on Link Tx Data.	000

Writes 0x3f. That's writing reserved bits and undocumented pattern in parity error injection!

Issue 2 Part 4

- Hardware model had two errors logged
  - Writing 1 to reserved field
  - Writing unknown pattern to defined field

- Impact is that they will incorrectly inject fault into system if the register is not updated later

Issue 3 Introduction

- Summit 1553 chip uses SRAM module
- Project uses module twice the required size, providing two banks
- The software is written to use the wrong bank

Issue 3 Part 1

Excerpt From ICD

The two Bank Select bits (one for each of the two separate BC and RT memories) are both initialized to 0 at C&DH subsystem power-on, **and they should never be changed to 1 unless there has been some kind of SRAM problem.** If changing a Bank Select bit is ever required, then the IFSW must make any change at the very beginning of its two 1553B SRAM initialization sequences, and before any descriptor table entries or data values or other information have been written to the selected bank of SRAM.

Issue 3 Part 2

brdACard.h:

```
#define BRDA_1553_CFG_USE_UPPER_BANK    0x00000001
```

Summit1553.c:

```
if(summitStop(summitRtRegs) == OK
     && summitReset(summitRtRegs, BRDA_1553_CFG_USE_UPPER_BANK) == OK
        && summitClearMem((unsigned long *)(summitRtMem),(unsigned long *)
          (brdABaseAddr + BRDA_1553_RT_SRAM_ADDR_END)) == OK){

…

static STATUS summitReset(SuRegs *p, unsigned long bankSel)
```

As you can see, bank 1 is selected for this Summit chip. (Happens to be RT.)

This is a violation of the ICD!

Conclusion

- Analysis of implemented software should include its use of hardware
- If functionality is correct, and unless hardware is implemented with certain tricks the developers will never realize these violations are occurring
- Software-only simulators provide mechanisms to support vigorous IV&V activities
- Build software-only simulations with sufficient level of detail so that ICD errors like these can be found for free