The Role of Test Languages for Embedded System JTAG Testing

Bradford G. Van Treuren

Lucent Technologies, Inc.

vantreuren@lucent.com

Boundary-Scan and Test Strategies Group

One Language Does Not Fit All

Board Test Action Group

Lucent Technologies
Bell Labs Innovations

Outline

- Test Language Theory
- System Architectural Impact on Test Languages
- System Operational Impact on Test Languages
- System Complexity Impact on Test Languages
- Conclusion

Test Language Roles

- We use specialized languages for many test activities
- Each language has a specific purpose for existence
- The purpose defines the capability/scope of the language
- Languages provide opportunity for portability
- Languages provide a standardization of process and data to ensure repeatability

Purpose of Languages

- Represent high level perspective of the problem domain
- Language commands capture common process steps
- Language commands capture many process actions in a single statement
- Simplify ability to produce a repeatable process
- Separation of data from the process
- Capture/Preserve state information

Test Language Scopes

- Ordered Pattern Representation Languages (e.g., Lucent Intermediate Test Language (ITL))
- Static Representation and Application Languages (e.g., SVF – programming open-loop)
- Dynamic Representation and Application Languages (e.g., STAPL – programming closed-loop)
- Application Programming Interfaces (API's)
- Flow Control Languages (e.g., IEEE 716-1995 Standard Test Language for All Systems--Common/Abbreviated Test Language for All Systems (C/ATLAS), Lucent TFCL™)

Current State of the Art

Vector-Based Languages

Free vector language players available for major JTAG vector languages (SVF, STAPL, 1532)

- Distributed as source code
- Players require upper level software written in C/C++ to manage and coordinate tests
- Players only provide application of vector languages

Current State of the Art

Vector-Based Languages

Vector Player Features

- Some players use a translated/compiled form of the original language (e.g., byte code, virtual machine code) to support more than one vector language (e.g., ispVM: SVF, STAPL, 1532)
- Difficult to recreate/follow original form of the vector language based on the Translated/Compiled code
- Other players use a binary version of the original vector language file which preserves line information and improves performance by eliminating run-time semantic and syntax checking

➢Line information is important for single stepping through a test using original source or in accurately tracking the progress of the application

Current State of the Art

Model-Based Languages

- High level dynamic modeled vector languages
  - ASSET InterTech Macro Language[1]
  - Goepel CASLAN™[2]
  - Easily able to target a single device in the chain
- General purpose language vector and modeling API's supporting:
  - C/C++
  - Tcl/Tk
  - VBScript
  - LabView
  - Etc.

➢ These languages are unable to be embedded in-system because of their model overhead so we need to rely on header and trailer information as a workaround

1. http://asset-intertech.com/jtag_scanworks_testautomation.html#Macro
2. http://www.huntron.com/Products/Boundary/cascon.htm

Outline

- Test Language Theory
- System Architectural Impact on Test Languages
- System Operational Impact on Test Languages
- System Complexity Impact on Test Languages
- Conclusion

Star and Multi-drop Architecture

Intra-Board Testing Language Issues

- Management of vectors/tests with boards
- Identification of board types per slot
- Board selection vs. vector application
  - Need common design solution for both architectures
- Current support status by standard vector languages
  - Chain selection support for 1149.1 compliant protocols
  - Extended 1149.1 support for SCAN Bridge selection protocol for Multi-drop
  - No support for ASP/LASP selection protocol for Multi-drop
  - No support for ad hoc JTAG Star topology

Star and Multi-drop Architecture

Inter-Board Testing Language Issues

- Inter-board testing requires coordinated application of one vector at a time to each board
- Current vector languages assume a batch flow of the vector file (all or nothing)
  - Current languages are unable to provide sequencing across boards designed without 1149.1 compliant gateway devices (e.g., SCAN Bridge interface)
  - Many have augmented the vector language to support selection commands for ASP/LASP as part of the language to provide the sequencing

➢ SCAN Bridge becomes part of the board chain after selection so the parking and waiting part of the protocol must be embedded in the test vectors to preserve the board signal state during these operations

Star and Multi-drop Architecture

Inter-Board Testing Observations

- Each board instance of a given type will eventually apply the same vector patterns to test the edge connector
  – Each pin must go through a 0-1-0 transition
- Inter-board testing requires vector reuse to reduce the memory volume required for vector storage

➢ Integration of the chain selection protocol in the vector language eliminates the possibility for vector reuse across board instances unless the language supports modular programming features like procedure calls or dynamic importing/linking

Outline

- Test Language Theory
- System Architectural Impact on Test Languages
- System Operational Impact on Test Languages
- System Complexity Impact on Test Languages
- Conclusion

Star and Multi-drop Architecture

Operational Issues

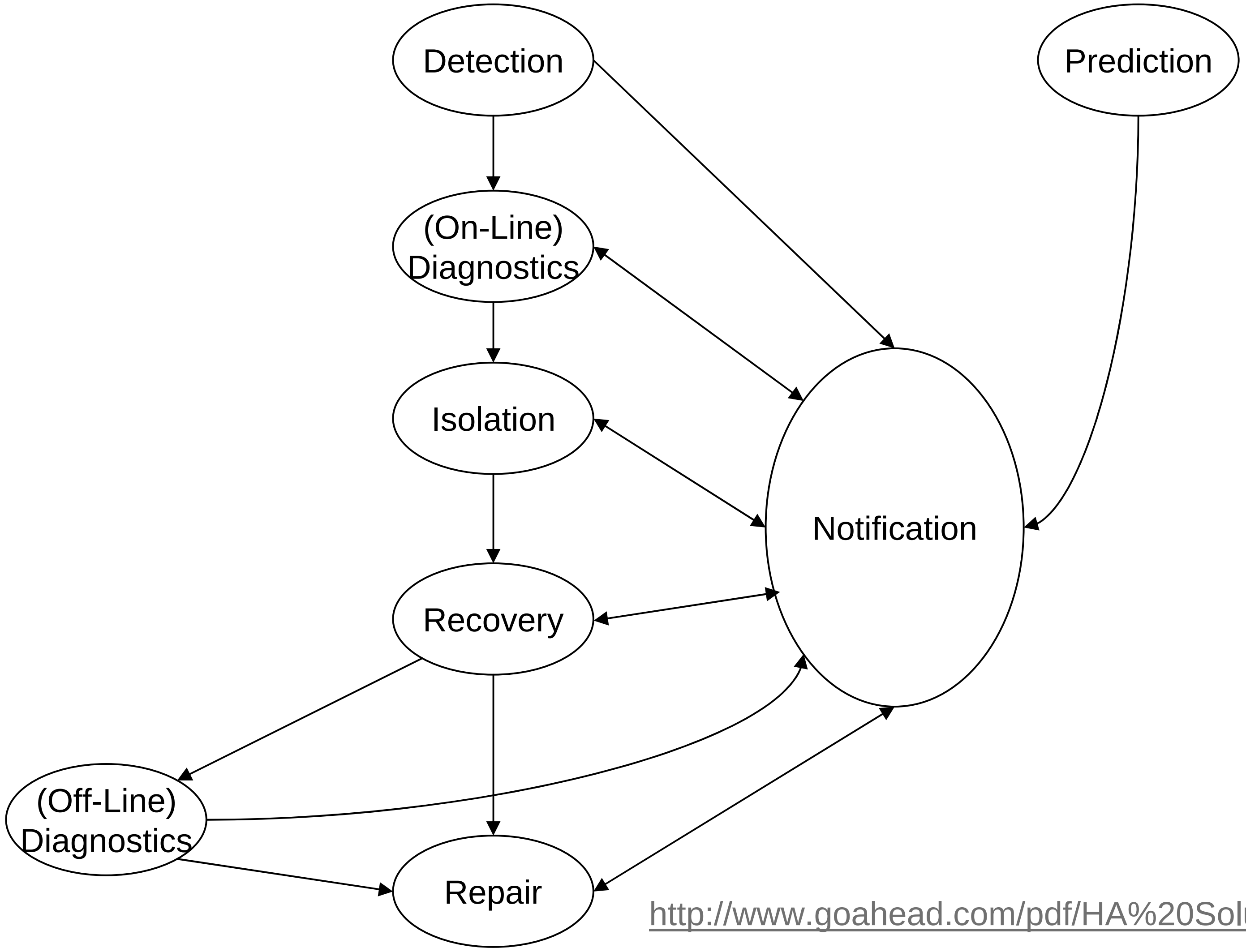
- Most system software for functional test delegates the diagnostics and reporting to a board-centric architecture
- A Multi-drop architecture does not integrate well with board-centric diagnostic reporting methods
- Multi-drop/Star board testing also requires rethinking of vector storage/application in an embedded implementation vs. on-board testing

System JTAG Integration Role

- Embedded Boundary-Scan interface should be able to leverage the current system infrastructure used for functional test
- System software should be responsible for ensuring the UUT is in the proper state for the operation prior to calling the JTAG test
  - On-line
  - Stand-by
  - Off-line
  - Out of service

Fault Management System

Fault Management Flow Chart

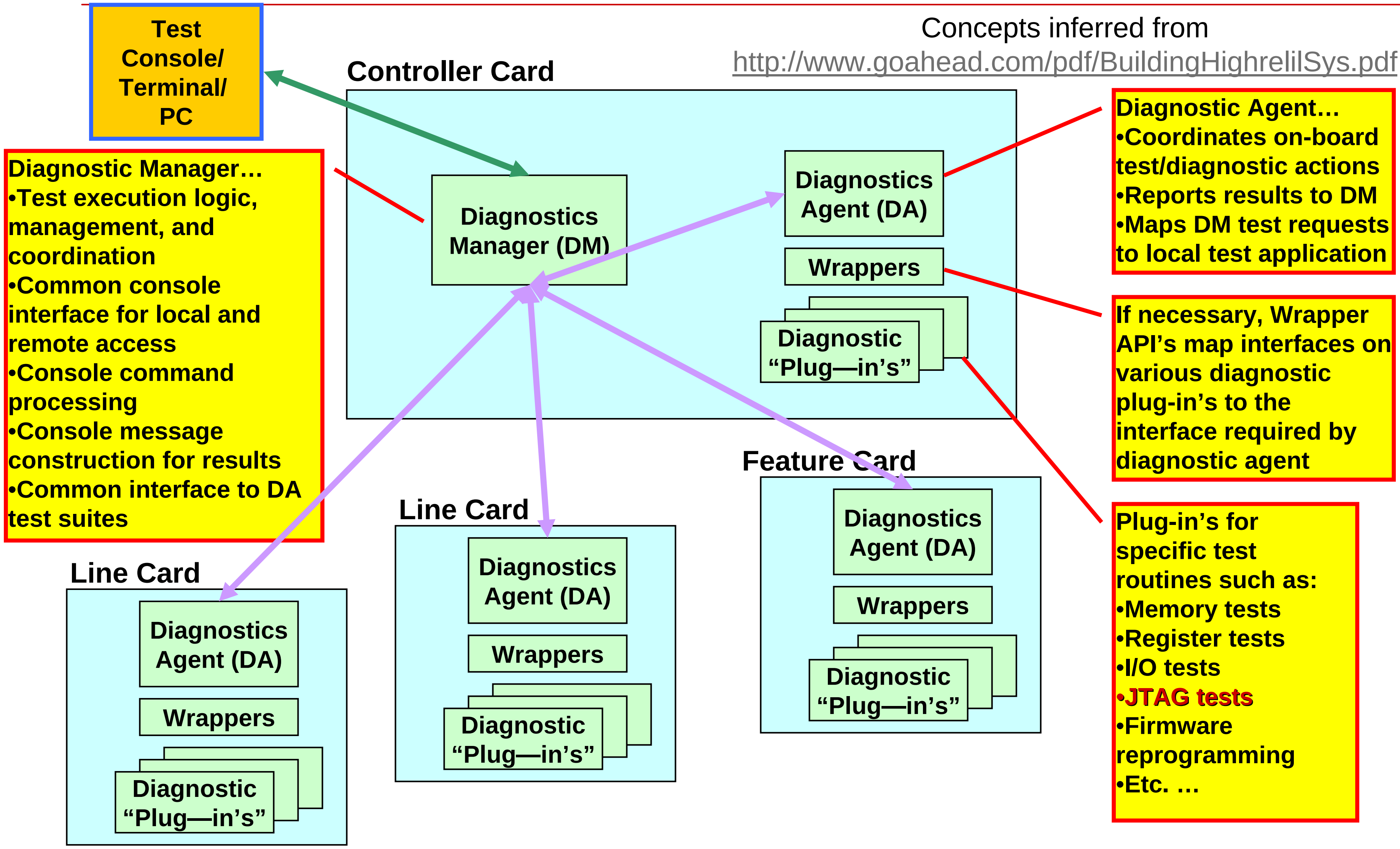


http://www.goahead.com/pdf/HA%20SolutionsWP.pdf

Product Application Software

(Ideal System Diagnostics)

Concepts inferred from http://www.goahead.com/pdf/BuildingHighrelilSys.pdf



Diagnostic Manager…
- **Test execution logic, management, and coordination**
- **Common console interface for local and remote access**
- **Console command processing**
- **Console message construction for results**
- **Common interface to DA test suites**

Diagnostic Agent…
- **Coordinates on-board test/diagnostic actions**
- **Reports results to DM**
- **Maps DM test requests to local test application**

If necessary, Wrapper API's map interfaces on various diagnostic plug-in's to the interface required by diagnostic agent

Plug-in's for specific test routines such as:
- **Memory tests**
- **Register tests**
- **I/O tests**
- **JTAG tests**
- **Firmware reprogramming**
- **Etc. …**

System JTAG Integration Role

- ➢ Just as there are diagnostic “plug-ins” for system software there should be language “plug-ins” to support the various forms of JTAG testing/enabling
- Propose: Test Coordination Language Layer needed between vector players and system level software
  - Allows software engineers to focus on system state management, error handling and system reporting (what they are good at)
  - Allows test engineers to focus on the test application and coordination (what they are good at)
- Treat JTAG vector language players as “Language Plug-ins” or small “Test Steps” in the overall coordinated test operation

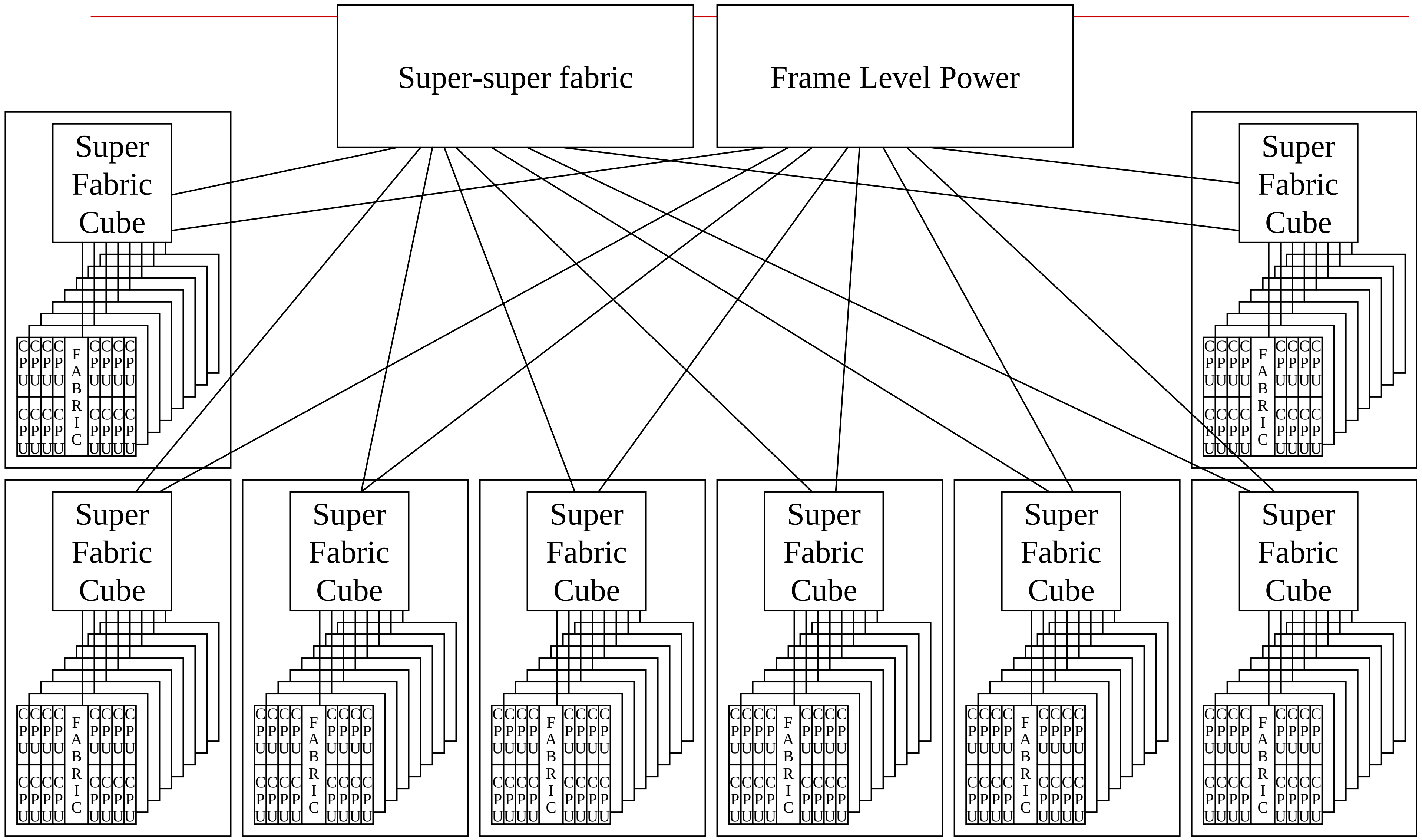
System JTAG Integration Role

- History has shown there will always be a new language on the horizon to perform a new JTAG based operation (e.g., SVF->STAPL->1532)
- The Test Coordination Language needs to be able to isolate the changes in the way we do JTAG operations in the system from the system diagnostics interface
- The Test Coordination Language needs to be able to treat all the operations required for performing a test (e.g., gateway access, chain selection, application of test vectors, programming) as a single autonomous entity to the system diagnostics software – A JTAG Application/Test

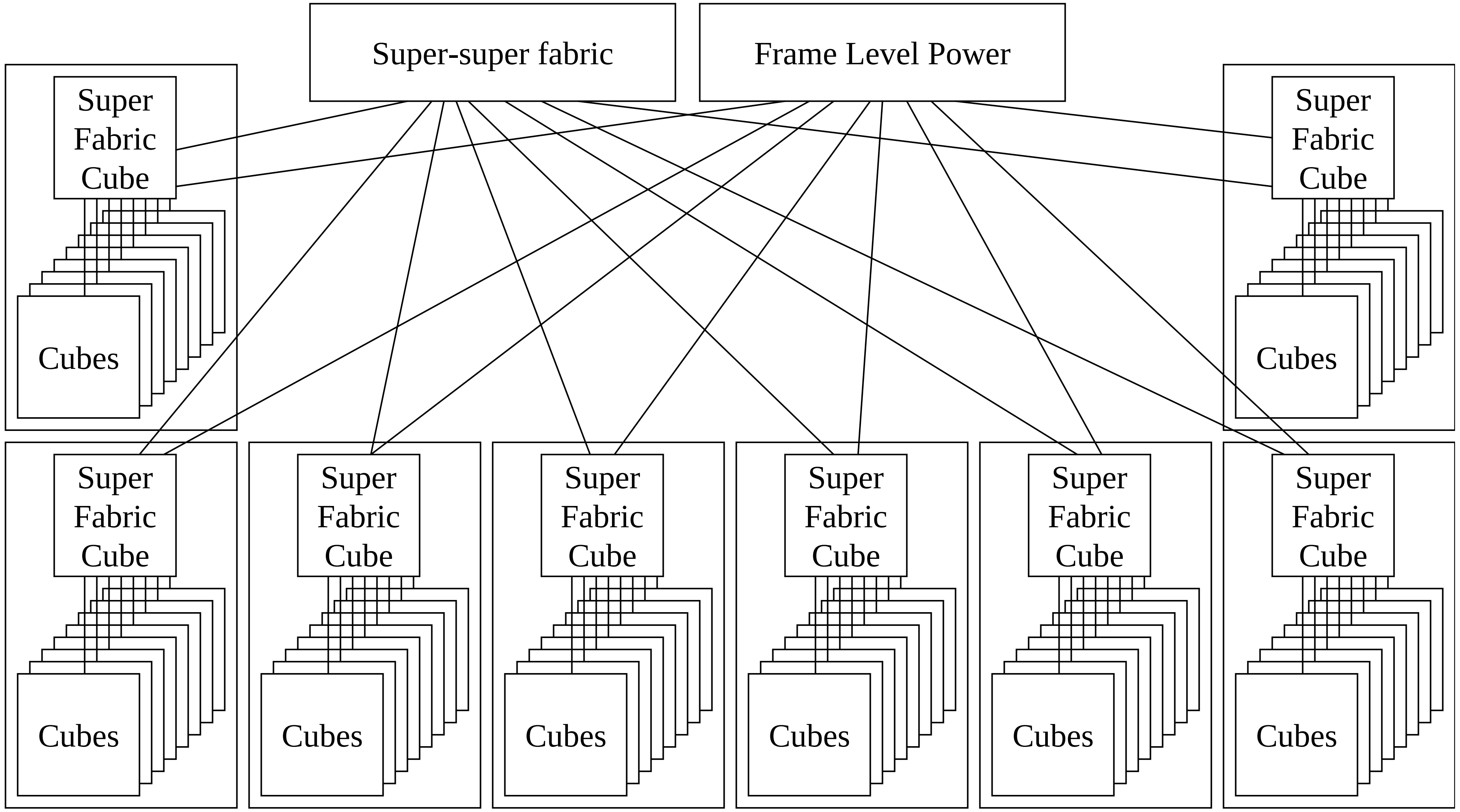
Outline

- Test Language Theory
- System Architectural Impact on Test Languages
- System Operational Impact on Test Languages
- System Complexity Impact on Test Languages
- Conclusion

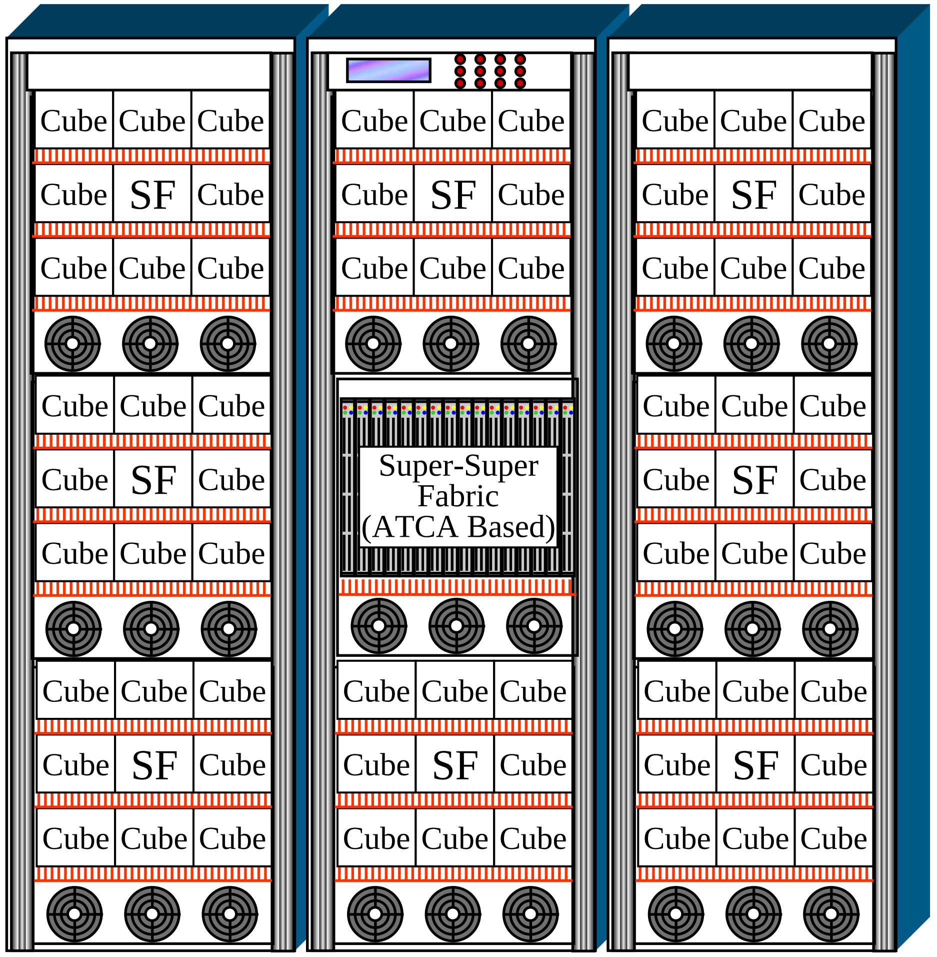
Application Study: Supercomputer



System Block Diagram



System Mechanical Design



Testability of Complex Systems

Test Factors

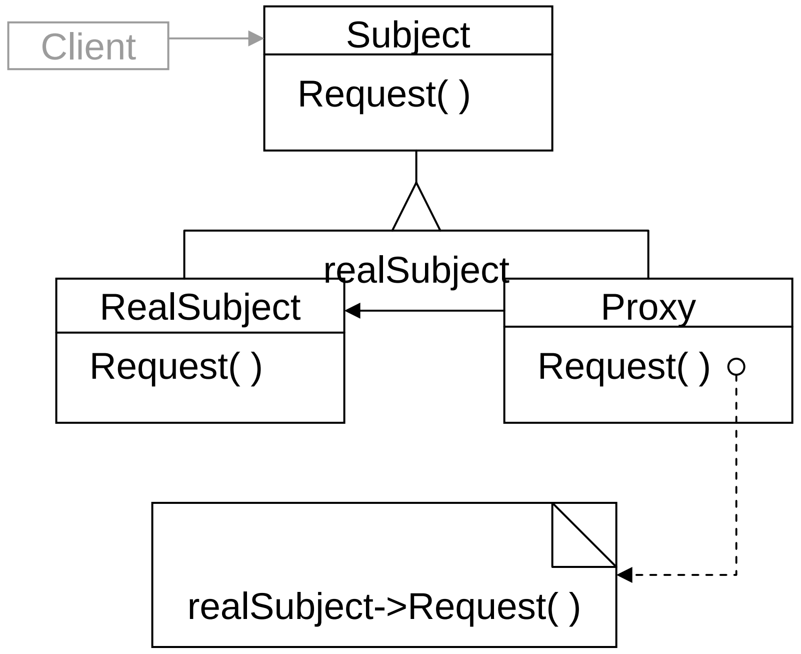
- Interconnections between fabrics and frames are accomplished using a “Cable Backplane”
  - Hundreds of cables are manually installed
  - Likelihood of incorrect or missing placement is high
  - Difficult to isolate failing cable using functional test on an N-wide channel (Reason for using 1149.6 or IBIST® in the Cube)
- Different system configuration for each customer base
- Traditional functional loop-back tests take too long to perform on all interconnections

Testability of Complex Systems

- The interface between cubes and super-fabric cubes and between super-fabric cubes and super-super-fabric cubes use the same SERDES or fiber technology used at the board and backplane level in the cube
- What prevents us from using the same tools and techniques to test these fabric interfaces? Connectivity!
- There is no physical unified JTAG interface between these elements to allow for the use of 1149.6 or IBIST to perform the coordinated testing

Scan Interface Proxy Architecture

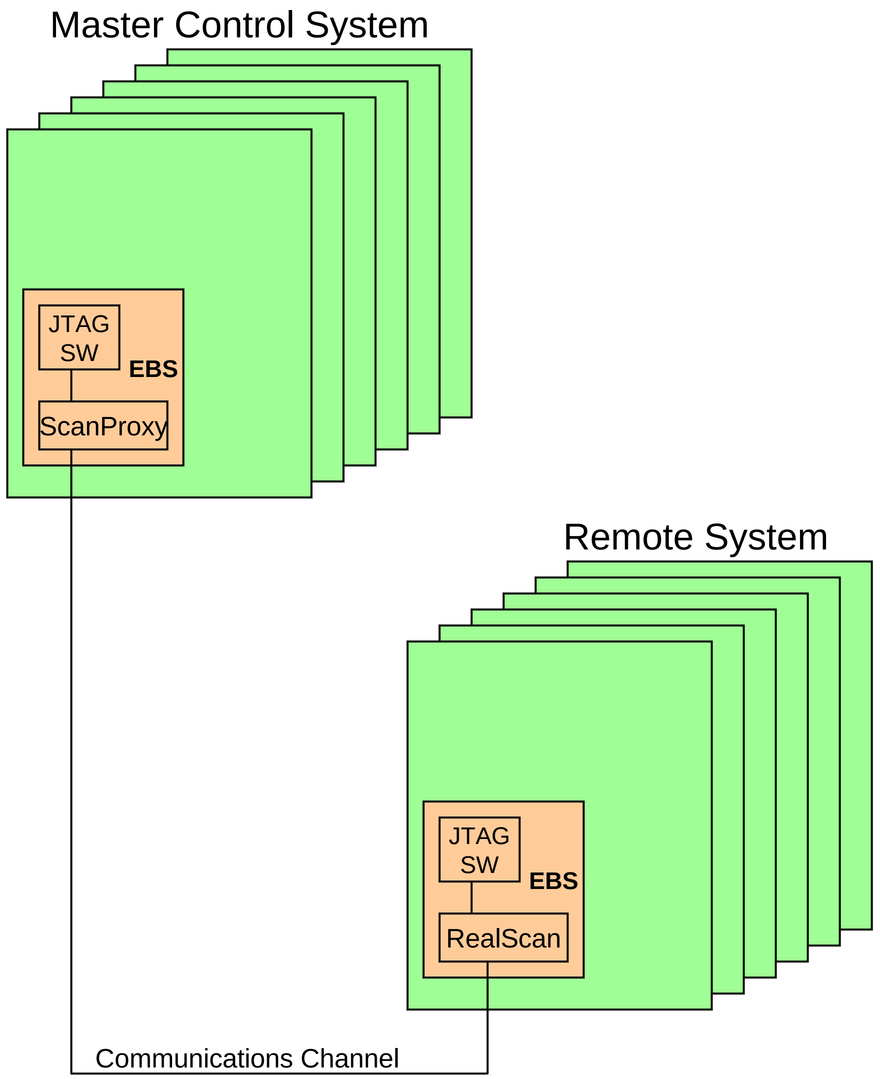
- Based on Proxy Design Pattern for Software
- A Remote Proxy provides a reference to an object located in a different address space on the same or different machine
- Scan Interface Proxy is used to extend the JTAG scan interface to a remote system to provide the same behavior on the remote system as if it were connected locally



Scan Interface Proxy Architecture

- Proxy interface to multiple remote JTAG Software interfaces provides a coordinated virtual Hierarchical Star or Multi-drop architecture over an established communications channel
- Requires the communications channel to be tested independently of the JTAG testing facility
- The communications channel may be a single pair of SERDES operating at relatively slow speeds (Mbps)

➢Yet another chain selection interface definition for a test language to support



Outline

- Test Language Theory
- System Architectural Impact on Test Languages
- System Operational Impact on Test Languages
- System Complexity Impact on Test Languages
- Conclusion

Conclusion

- Each language used for JTAG has a special purpose
- Selection of board/system gateway chains should be done at a level outside the vector language since there are many methods used for gateway/chain selection operations
- For 1149.1 compliant gateway protocols, separate vector files or procedures should be used for selection and parking operations to ensure ability to reuse vectors for multiple instances of the board in a system
- Non-1149.1 gateway protocols/methods should be treated as separate test operations from the vector language – Otherwise, the vector language will always be changing as new methods appear

Conclusion

- JTAG applications must resemble a functional test to be able to leverage existing system diagnostic facilities already required in a system
- The details of what is done during a JTAG application must be abstracted away from the system diagnostic software to ease integration and provide test flexibility
- A middleware language layer performing Test Coordination provides the abstraction and test flexibility required
- Each vector language should be implemented as a “Language Plug-in” to the Test Coordination language
- The Scan Interface Proxy provides a close approximation to the Star and Multi-drop architectures to support JTAG operations for isolated JTAG implementations