
STAPL

A System Level
Boundary Scan Language?

Mike Westermeier and Don Lillencrantz

Outline

- What's the problem?
- Where did STAPL come from?
- What it is?
- How do you make it work?
- Summary

System Level implementation from 1997

Conceptual

- What hardware needs to be added to the boards?
- What tools are available?
- How are we applying test vectors and/or files?**
- What are the software requirements?

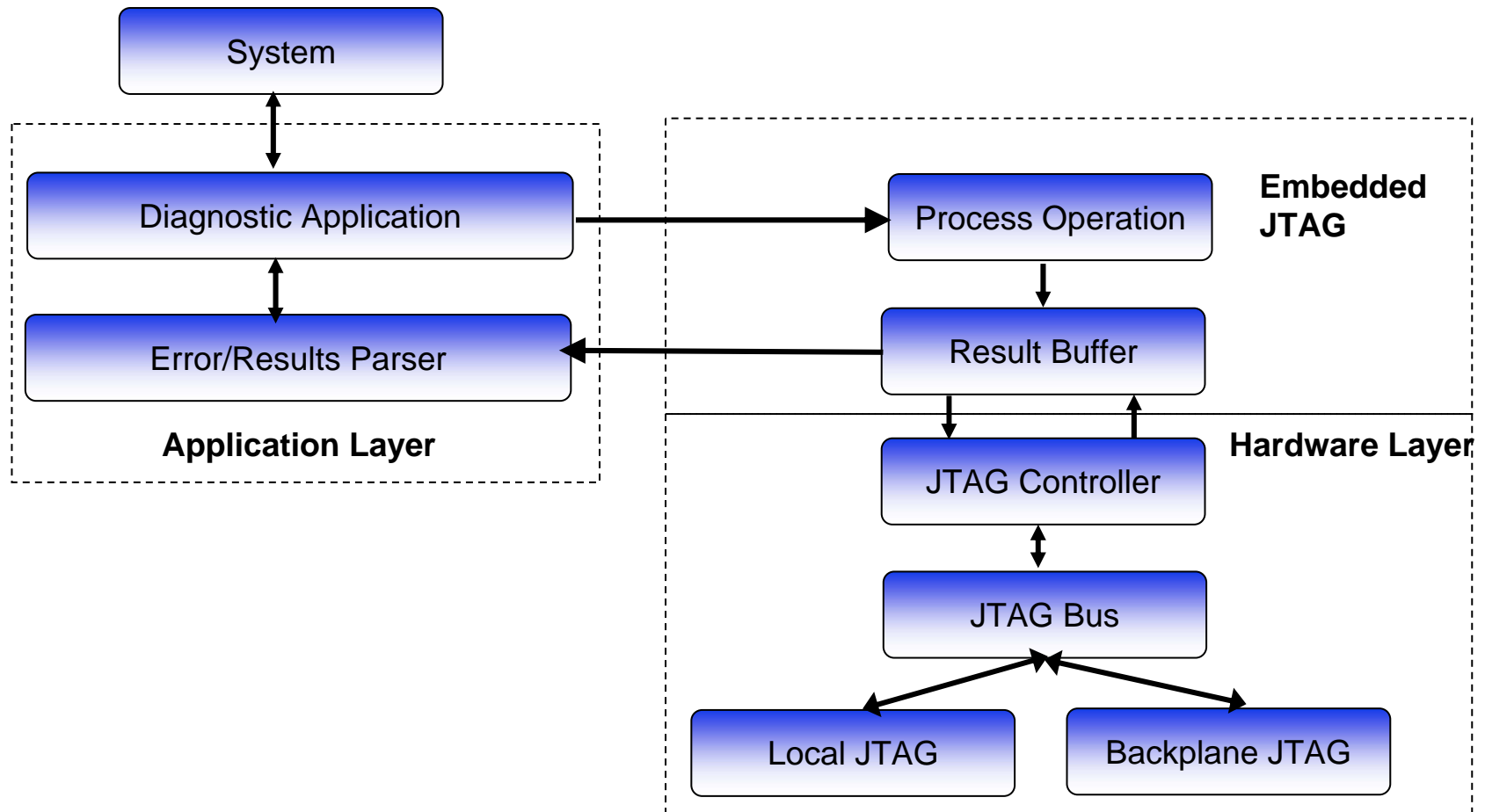
Actual Problem?

- Why are we doing system level?
 - Not for test purposes.....

What actions did we intend to run?

- PLD programming
- FPGA configuration
- Module Identification
 - Read Module information
- I²C redundant controller
 - Control I²C bus to program/verify/read serial PROM
 - Control I²C bus to read voltage and temperature sensors
- Digital IO – Reset, Initialization, etc....
- Read FPGA Done bits to isolate configuration problems.
 - Processor can not read Done bit register if PCI FPGA not available.

JTAG System Overview



JTAG Test Language Goals

- Use the same tests from Prototype through Production
 - Debug on PC and then use in System
 - Export to “commercial ATE”
- Some method for checking file integrity
- Date and time stamping for file control
- Pass arguments to support file reuse
- Export errors and results back to System
- Support for TI’s Shadow protocol and National’s ScanBridge
- Low Hardware and Software Cost Implementation

JTAG System Integration

□ Goal was to have JTAG actions be as standalone as practical

- System diagnostic infrastructure calls JTAG action(s)
- Error handling and decision making is part of JTAG action
- JTAG action provides status back
- Infrastructure decides what happens next based on status
 - Was previous result an error – stop immediately
 - **A JTAG scan path test failed**
 - Was warning – a failure that could be ignored
 - **Manufacturers part revision**

□ Reason

- Faster response times
- Easier to develop and verify

JTAG System Level Language - Choices

Create our own language

- Need to create toolkits for conversions - who?
- Development time?
- Not exportable to outside vendors, customers
- Reinventing the wheel

Serial Vector Format (SVF)

- Not a language – no decision making
- Can't handle TI ASP?
- Defacto standard but not easily usable for real applications
 - Most JTAG tools read/write SVF

Why not our own?

- Too much support, training, etc.....
 - Too risky?
- Would need to create tools for file conversion
- Would need to create our own hardware
- Insufficient problem to warrant

Why not SVF?

- ❑ FC chip with the same ID code but different chain length
 - Required two different SVF files for all actions
- ❑ Alternate ID codes
 - Required two different SVF files for Scan Path actions
- ❑ Couldn't open an ASP
- ❑ Altera had created JAM for programming (SVF had issues) – we would need a JAM to SVF converter

Then our local Altera Apps Engineer suggested we use JAM instead of SVF

Other vendor options

❑ Jdrive from Xilinx

- Required JAVA – systems weren't shipping with JAVA – licensing issues

❑ Virtual Machine from Lattice

- Proprietary code?
- SVF like?

❑ JAM from Altera

- What was it intended for?
 - Programming PLDs
- What we felt it really was?
 - A JTAG Language?

Alteras JAM as a JTAG Language?

- ❑ Combined SVF + BASIC-like instructions for a simple interpreted language
 - Incorporates conditional looping using For-Next, If-Then
 - Compressed data arrays
- ❑ Source code was freeware from Altera
- ❑ Contains CRC command to verify file is not corrupted
- ❑ Simple hardware – low level control of JTAG signals
 - Parallel port player
- ❑ ASCII files for easy reading and debug use
 - Self contained actions with date stamps
- ❑ Converters available for SVF to JAM

Issues with JAM

- ❑ No ASP support
 - Added Shadow commands to our JAM player
- ❑ No ability to single step
 - Added “Note Pause”
- ❑ Increased the bit length for DRscans and IRscans
- ❑ Could not read external files
 - Created Perl scripts to move Intel Hex data into data arrays instead of changing JAM
 - No extra overhead for JAM to access external files

Jam becomes STAPL

Serial Test and Programming Language

- ❑ JEDEC standard JESD71 – August 1999
- ❑ Based off of Altera's JAM Programming Language
- ❑ Kept most of JAM features but not all
 - Variable override was dropped
 - Used this to pass specific ASP address
 - **Default variable was set to 1 inside of JAM action**
 - **Significant nuisance – loss of flexibility**
- ❑ Has variable arguments for passing actions
 - A **single** file may contain erase, program and verify operations which may be selected through command line arguments

Things you can do with STAPL

- ❑ Write, Read and Manage test data
- ❑ Configure/Validate CPLDs and FPGAs
- ❑ Capture the result of the action and compare it
- ❑ Log test action execution details
- ❑ Send specific reports based off test data
- ❑ Using STAPL we could:
 - Program Memories – I2C, Serial Proms, Flash, etc....
 - Check oscillators
 - Make flow control decisions
 - Controlling board signals
 - Test DACs basically anything you can digitize you can test

What makes this a System Language?

Instructions

- Call, GoTo, For, IF/Then, Note, Export, Print, etc.....
- DR/IR Scans, PRE/Post IR/DR
- Data arrays

Operators

- Bitwise, Math, Comparison, etc....

Functions

- Integer and Boolean

Extensions

- Vector and VMAP

For additional or specific info please reference the standard

Vector and VMAP Example

□ STAPL ASP procedure

- Initialize the action
 - Use variable override to pass ASP address
- Use VMAP and Vector to bit fiddle JTAG signals for handling ASP protocol

```
VMAP "***TDO**", "***TDI**", "***TMS**", "***TCK**";
BOOLEAN VDir[4]; ` vector directions
BOOLEAN VIn[4]; ` driving values
BOOLEAN VOut[4]; ` captured values
VIn[0] = 1;
Vector VDir[3..0], VIn[3..0] CAPTURE VOut[3..0]; `
    TCK High
ASPOut[Vx] = VOut[3];
```

User Defined EXTENSIONS?

□ EXPORT Definition from the Standard

- The EXPORT statement exports a key string and a data value
- EXPORT <key string>, <Integer or Boolean array expression>;
 - EXPORT “Percent_Done”, (done*100)/total;
 - EXPORT “Data”, \$12AC3F74;
 - EXPORT “Usercode”, user_value[31..0];

EXPORT User Defined Extensions

❑ EXPORT “INFOPROC PROGRAM_PROCEDURE”, 0;

- A “0” is leaving, a “1” would enter

❑ EXPORT “INFOCOM Whatever”,0;

- User defined information to be sent

❑ EXPORT “FLOWPAUSE”, 0;

- Provides single step capability by stopping execution until a key is pressed.

❑ EXPORT “SLOTID”, 13;

- Provides boards position in system
 - Are we talking to who we think we are?

EXPORT User Defined Extensions

❑ EXPORT “INFOEXPECTLOW”, abc;

- Bit position abc expected a low but the scanned value was high.

❑ EXPORT “INFOEXPECTHIGH”, abc;

- Bit position abc expected a high but the scanned value was low

❑ EXPORT “INFOEXPECTINT”, abc;

- The variable defined earlier in an EXPORT command was expecting a value of abc

❑ EXPORT “INFOACTUALINT”, abc;

- The variable defined earlier in an EXPORT command found a value of abc.

Other potential user extensions

NOTE

- NOTE “MAXIMUM DR”, abc;
 - Provide to the player how long the maximum DRscan to reduce memory allocation within this action is

PRINT

- Debug purposes

System Flow

❑ Decision within the system

- Spawn appropriate STAPL action with command line arguments
- Inside of the STAPL action
 - You make action appropriate decisions
 - Determine and export response

❑ Determine next decision based off of response

- Spawn next action
 - Decision and Export

❑ Repeat as desired

System Definition

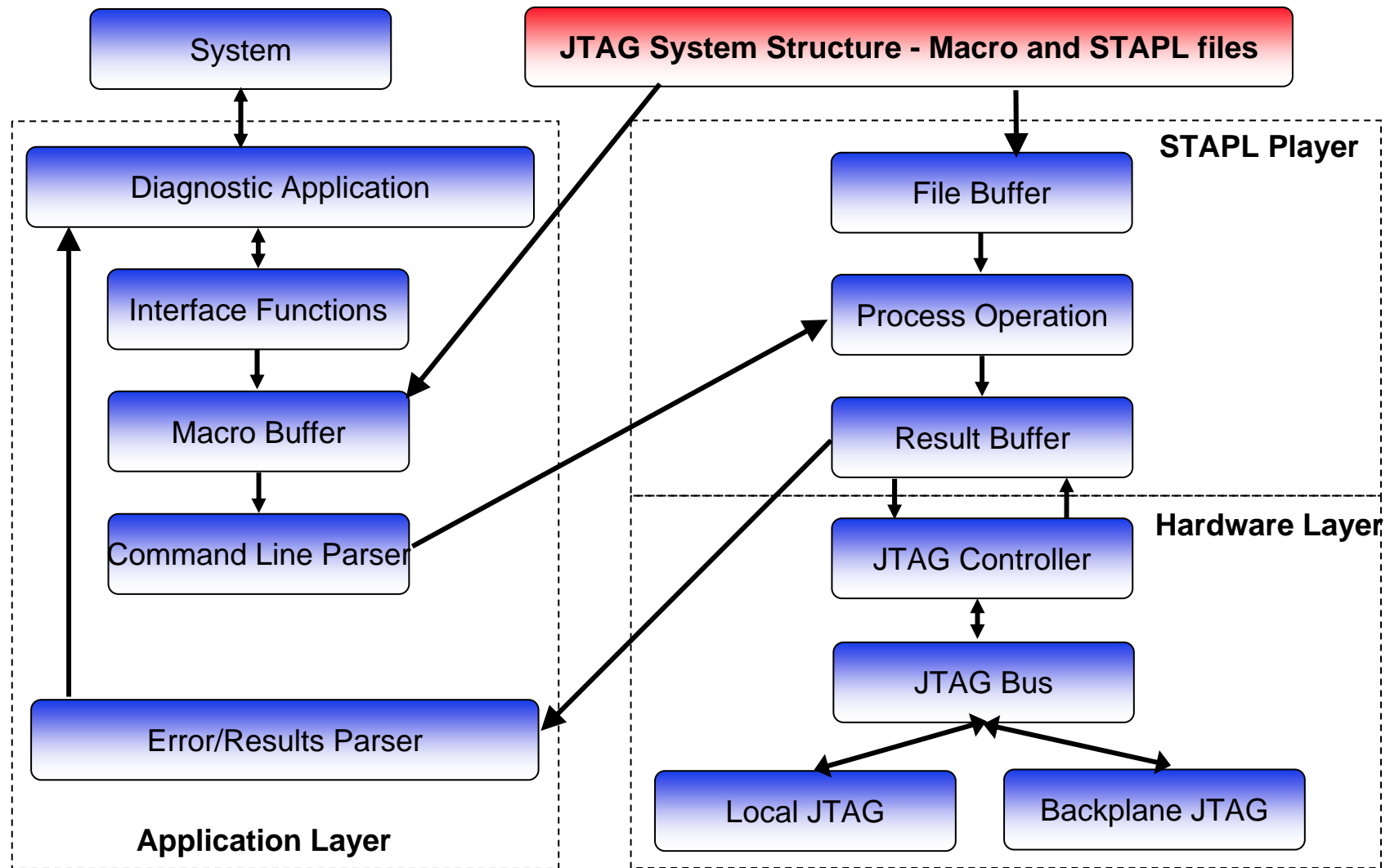
□ Diagnostic Module Layer

- JTAG was of many sub layers
 - Macros that contain JTAG sequences consisting of individual STAPL actions
 - Macros controlled the start and error handling
 - Types of sequences
 - **Update Sequence**
 - Open Slot
 - Read Slot, Module and Firmware ID
 - Update if out of rev
 - I²C backup, Cable connection, System Status, etc.....

□ JTAG application layer (STAPL player)

□ Hardware layer

JTAG System integration



Issues

- ❑ Vendor tools won't be able to process real intent of extensions on export statement
 - Make them Syntax acceptable
- ❑ You will need to modify your embedded player to process extensions
 - You will need to anyway to adapt it to your system
- ❑ Can't read external files
 - Not needed or wanted
- ❑ **The file extension for a STAPL file**
 - **.JAM or .STP or .STA or STAPL**

What makes it a SYSTEM Language?

- Supports all states of the TAP
 - Ability to re-use all actions from BSV tools
 - Ability to add your own extensions on EXPORT command
 - Ability to use the VMAP and VECTOR for complete control of JTAG signals or Digital IO
-
- STAPL as defined supports almost all required JTAG goals

What really makes it a SYSTEM Language?

- Validated in a simulation environment
- Validated at a past company in a Multi-drop Backplane system with multiple embedded masters
- Validated at a past company in a no Backplane system with embedded masters

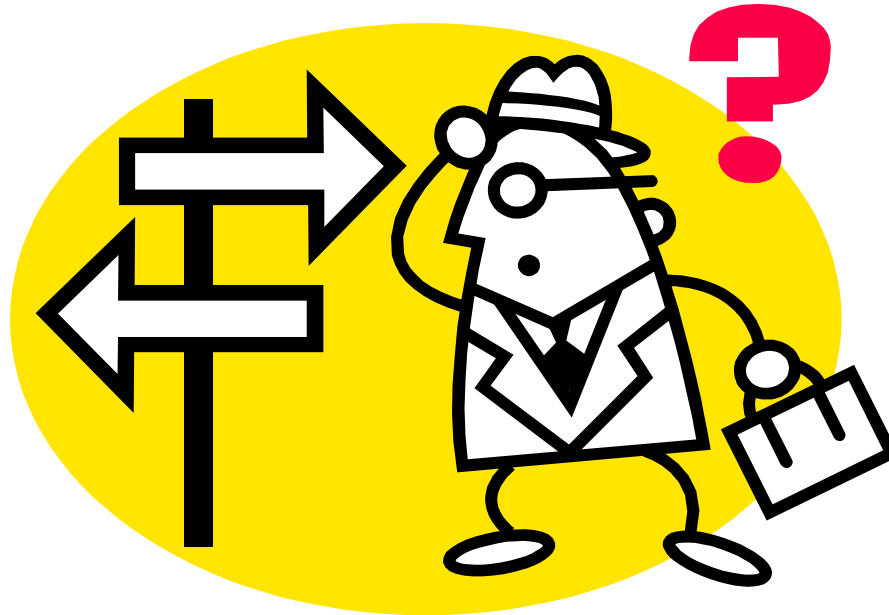
Conclusion

- STAPL works as a System JTAG language



Reference: JESD71 Standard Test and Programming Language (STAPL), August 1999

Questions



[STAPL Example](#)

[STAPL Instructions](#)