Example of how IEEE P1687 (IJTAG) Creates IP Test Standardization and Portability

Introduction

IEEE 1687: Internal JTAG

This emerging IEEE standard focuses on test **access** and standardization for the exploding world of embedded ASIC instruments (or IP). As the variety and number of embedded IPs have increased, so has the variety of ways to **test and debug** them. This places a burden on IP vendors, ASIC designers, ASIC test engineers and ASIC consumers by requiring extensive learning and setup for each new IP, as well as **integration** into the latest ASIC and latest process node!

Eliminate the test learning curve

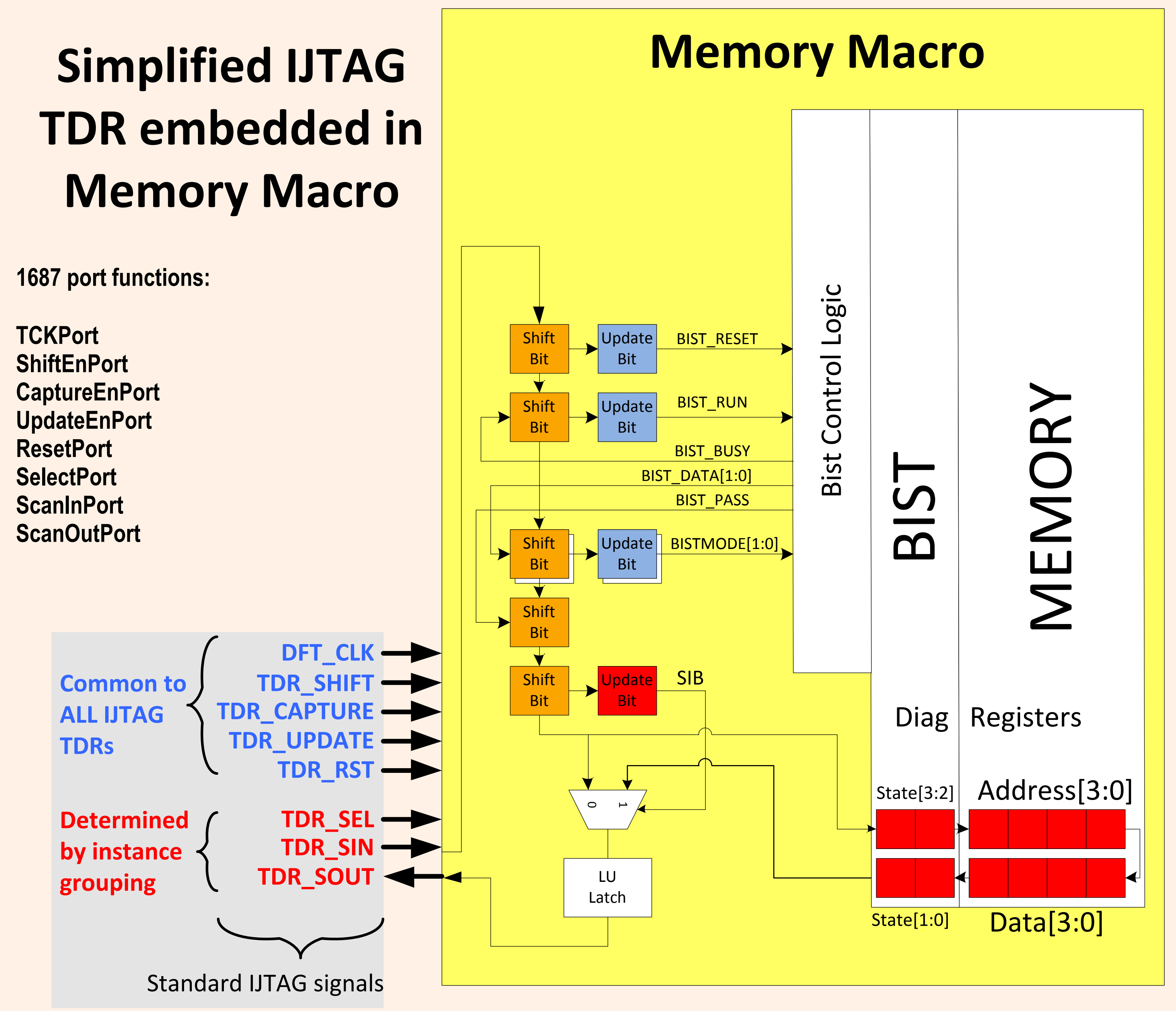
In the past, IP test integration was a long, complicated process.

- The IP provider had to teach the ASIC test designer how to integrate each IP, and teach test procedures.
- That knowledge then had to be translated to the ASIC's top level pins and how to access IP test from the outside world (ATE test).
- Yet another translation of this information has to occur to enable the ASIC consumer for board test and system level test.
- At each level, the test access looks different and is essentially custom.

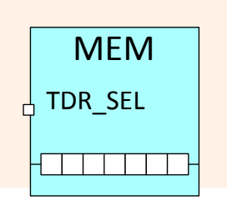
IJTAG supplies a standard eight-port interface, a standard test hardware description, and a standard test procedure language for each IP. The same descriptions are applied at each point in the design/test process from IP integration to board test and in-system test.

Avago IJTAG Implementation At The IP Level

Simplified IJTAG TDR embedded in Memory Macro



The figure to the right is meant to represent the above Macro in the top level diagram to the right



IJTAG Architectural Components

(Color of oval under each subhead ties to the color of the blocks in the figures below)

Eight Standard Interface signals

These eight signals form the basis for serial communication to all IJTAG-compliant IP. They are very similar to the IEEE P1500 interface, but do not have distinction between Instruction and Data registers. The reset has to be asynchronous because the chains must reset without a running clock. The control lines allow for capture, shift and update functions. The select line must be common to all IP daisy-chained together using the scan in/out ports.

Scan Bits

The scan bits must be able to toggle without affecting the IP. This enables access to any instrument while not disturbing other instruments. Configuring the IJTAG control as a shift chain keeps the port count low, no matter how complex the control structure. Bits may be added or deleted late in the design cycle with out changing the ports on the block. Shift bits can also **Capture** important data before starting to shift out.

Update Bits

The update bits provide stable information to the instrument, even while the chain shifts. These bits must reset to a known state asynchronously. These are the "named" bits used in IJTAG software. These bits change state only when UpdateEn is active.

The Segment Insertion Bit, or SIB

This is the heart of what is different about the IJTAG standard: reconfigurable scan chains. A SIB opens or closes access to a hidden scan chain depending upon the value contained in the SIB. There are no limits to number or "depth" of SIBs in a network. In the case of this Avago memory example, the SIB opens up access to the diagnostic data and functions that do not always need to be available.

"Scanned IP Macro"

This example shows how Avago has built the IJTAG control chain and BIST into our memory macro. This makes test insertion easy, because it becomes merely a matter of connecting to the standard interface. The P1687 standard also allows for the IJTAG control chain to be created as a wrapper around the IP for maximum flexibility.

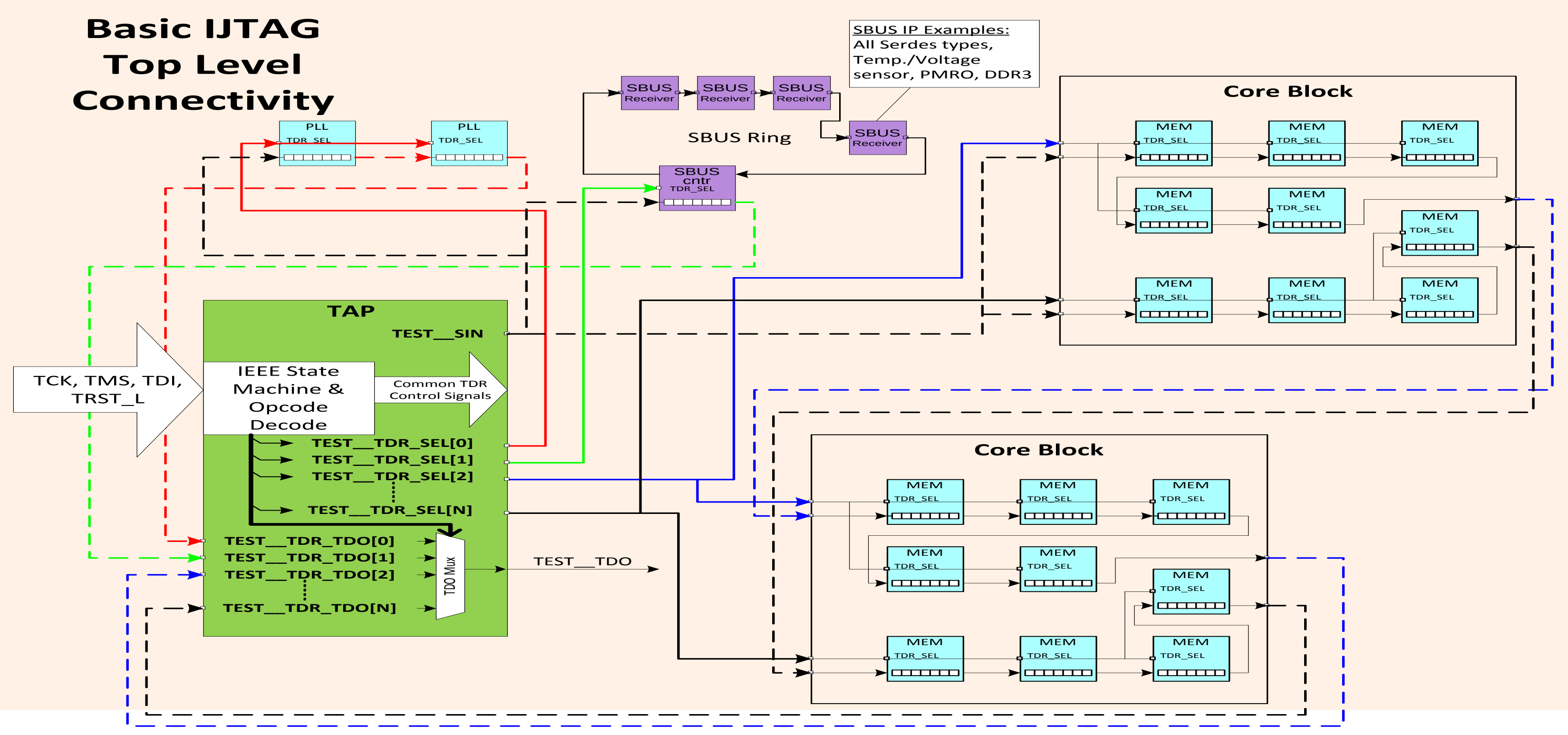
Avago TAP

Avago generates a custom TAP for each ASIC. Even though all IP could be on one IJTAG chain, Avago's TAP allows IP to be segregated on different IJTAG chains to enable efficient production test /debug methods.

Avago's SBus, and IJTAG "Super Instrument"

Avago's SBus controller enables full control of the highly complex BIST features built into our SerDes, DDR3, and more. Literally hundreds of these embedded IP can be controlled by a single SBus controller, and yet, all of this control is activated by a simple 127-bit IJTAG chain. Multiple SBus controllers can be implemented on-chip for organization and/or other architectural reasons.

Avago IJTAG Architecture At The Chip Level



The Languages of IJTAG: ICL, PDL

Standardized test hardware access requires a standardized description

To accomplish the goal of standardization and portability, the IEEE P1687 Working Group has created two description languages: the Instrument Connectivity Language (ICL) and the Procedure Description Language (PDL). The combination of these two languages allow an IP's test description to be written once at the IP level, and then be applicable no matter where the IP is instantiated or how many times.

Instrument Connectivity Language (ICL)

The goal of ICL is to describe connections between the IJAG scan circuitry and the instrument ports, but not instrument functionality. The standard is flexible and complete enough to describe such things as indirect addressing schemes, addressable registers, and basic logic/muxing functions. This permits a broad range of IP test descriptions. ICL also extends up the hierarchy to describe the connectivity of the IP as they are stitched together on IJTAG chains, including stand-alone SIBs. So a combination of IP-level ICL descriptions and chip-level IP connectivity ICL descriptions provides everything needed to find every IJTAG control/data bit on the chip.

Procedure Description Language (PDL)

PDL supplies the information needed to "RUN" the tests on the targeted IP. The tests are described at the IP level, without concern for where the IP is used in the chip's hierarchy or floorplan. The tests perform write and read operations on the registers or pins as defined in the ICL. The important element contained in PDL is a sense of time. Instructions in PDL show order, i.e., accomplishing one task before moving to another. In this way, a sequence of events can be described to accomplish a specific test goal. A set of named test procedures can be delivered with each IP to perform each of the necessary test functions, including debug.

Translation To Simulation And Tester Environments

The IJTAG standard does not provide the software to turn these configuration files into actual simulation test benches and tester vectors. That is best left to the industry. Avago and other device makers have produced internal tools, and several EDA companies have announced or are working on products, which will make available multiple solutions.

ICL Example

```
Module SBUSDef  {
    ScanInPort  sSIN ;
    ScanOutPort  sSout {Source SBUS_cntl[0];}
    SelectPort  sSEL;
    ResetPort  sRST;
    ShiftEnPort  sSEN;
    CaptureEnPort  sCEN;
    UpdateEnPort  sUPD;

    ScanRegister SBUS_cntl[126:0] {ScanInSource sSIN; ResetValue 127'b0; }
    // Use aliases to identify the fields of the 127-bit SBUS_cntl register
    Alias  sExtraIn[21:0] = SBUS_cntl[126:105];
    Alias sInstAddr[7:0] = SBUS_cntl[104:97];
    Alias  sRegAddr[7:0] = SBUS_cntl[96:89];
    Alias  sCMD[7:0] = SBUS_cntl[88:81];
    Alias sDataIn[31:0] = SBUS_cntl[80:49];
    Alias  sTapResetSel = SBUS_cntl[48];
    Alias  sMisc1[1:0] = SBUS_cntl[47:46];
    Alias   sRESET= SBUS_cntl[45];
    Alias   sOverride= SBUS_cntl[44];
    Alias   sExecute= SBUS_cntl[43];
    Alias sMisc2[10:0] = SBUS_cntl[42:32];
    Alias   sDataOut[31:0] = SBUS_cntl[31:0];
}
```

PDL Example

```
iProc PMRO_Test  {

## ------------------------------------------------------------------------------------------------- ##
##    This tests the PMRO in full test mode, using primarily defaults.                  ##
##    The SBUS drives the PMRO, so the commands load up the SBUS and     ##
##     then send the data to the PMRO.  A wait is performed for the PMRO to    ##
##    accumulate data and then it is transferred first to the SBUS then to the      ##
##    tester.                                                                                              ##
## -------------------------------------------------------------------------------------------------##

iWrite sOverrride          b1;                   ## Puts the Sbus in TAP control
iWrite sCMD                0x01;                 ## SBUS Write Command
iWrite sInstAddr           0x00;                 ## PMRO Address
iWrite sRegAddr            0x01;                 ## PMRO Cmd Reg
iWrite sDataIn             0x00000001;           ## PMRO Start Command
iApply;

iWrite sExecute            b1;                   ## Send Cmd to PMRO
iApply;

iWrite sExecute            b0;
iApply;

iRunLoop 500;                                    ## Wait 500 TCK cycles


iWrite sCMD                0x02;                 ## SBUS READ
iWrite sExecute            b1;
iApply;                                          ## Read Config Reg

iWrite sExecute            b0;
iRead sDataOut             0x00000000;           ## No operations in process
iApply;

iWrite sRegAddr            0x05;                 ## PMRO Count Reg
iWrite sExecute            b1;
iApply;

iWrite sExecute            b0;
iRead sDataOut             0x00000000;           ## Will contain the PMRO count
iApply;

}
```