The following is an older presentation given to the P1687/IJTAG working group in 2008. After three years the WG had not set a direction on how to describe instruments. CJ Clark presented his vision of how P1687 should work drawing on past use of controlling on-chip DFT via 1149.1 and TCL.

This set the direction and TCL has been incorporated in P1687 Level 1 PDL.

For the latest information and access to P1687 tools please go to:

http://www.intellitech.com/ijtag

Intellitech

IJTAG Language Position

CJ Clark is the President and CEO of Intellitech Corporation.

He was the elected chairperson of the IEEE 1149.1 JTAG working group from 1996 to present. He has been active in other IEEE working groups and has presented at International Test Conference, TECS (Testing Embedded Cores-Based Systems) Workshop, the Board Test Workshop, Ottawa Test Workshop and VLSI Test Symposium.

CJ serves on the University of New Hampshire College of Engineering and Physical Science (CEPS) Advisory Board. He also serves on the UNH Department of Electrical Engineering Advisory Board. He is co-inventor on four US patent related to scan-based test, two Canadian, two European, two Taiwanese patents with others pending world-wide. His first job in test was in 1978 with Plantronics/Wilcom.

Seeking "Nemawashi"

"Digging around the roots of a tree"

Japanese culture of peaceful process called Nemawashi. Each root a stakeholder in the process – ensuring buy-in of each one – before decision is announced

Instrument Architectures exist

Instrument Providers deliver .VIs

.VIs contain *executable code* not language to interpret

Labview Software calls 'callbacks' in .VI instruments

```
init_gui(init = 0x101);
```

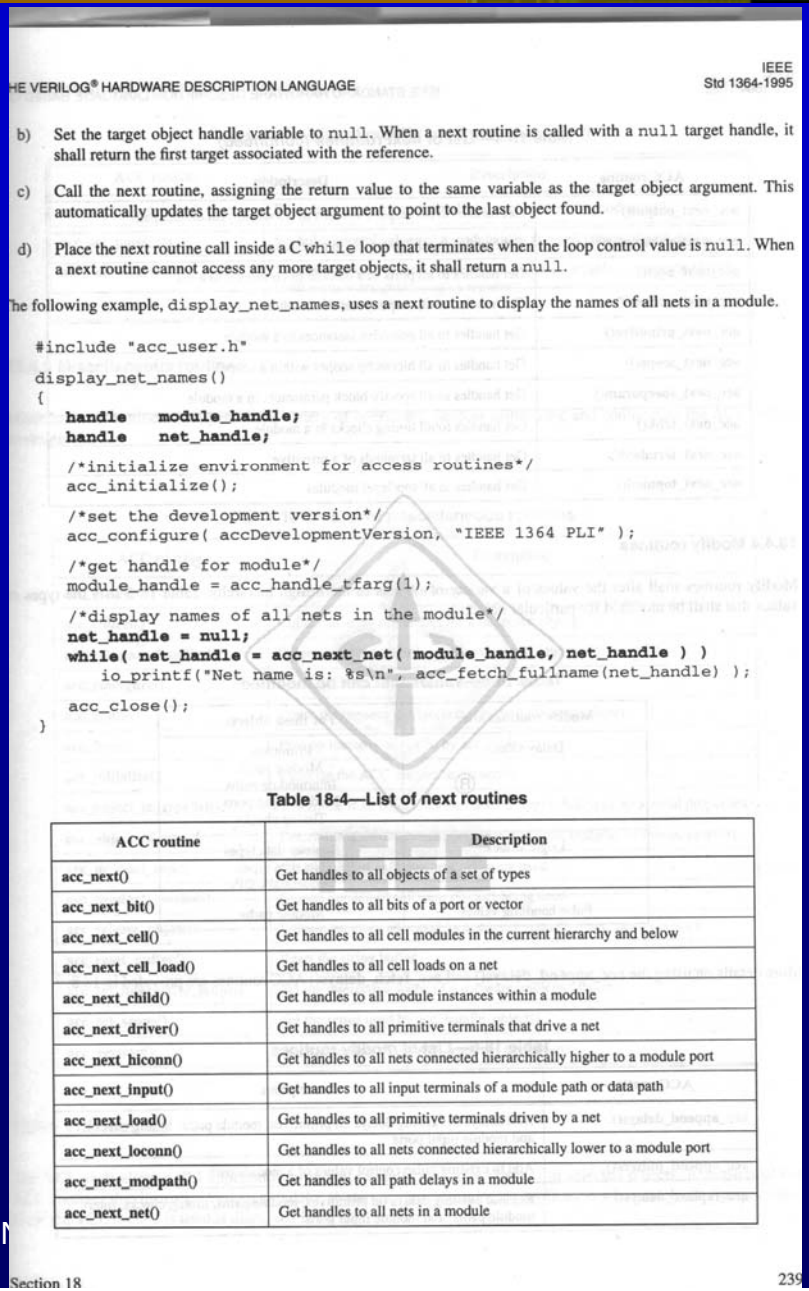
IJTAG needs a similar framework
- not proprietary
- open for all vendors
- Instruments are/will be more complex than BIST

IEEE & Existing languages

Verilog IEEE Std. 1394

-Majority of standard is PLI definitions
-C language
-Definition of mandatory routines

Standards can pick a An existing language



HE VERILOG® HARDWARE DESCRIPTION LANGUAGE IEEE Std 1364-1995

b) Set the target object handle variable to `null`. When a next routine is called with a `null` target handle, it shall return the first target associated with the reference.

c) Call the next routine, assigning the return value to the same variable as the target object argument. This automatically updates the target object argument to point to the last object found.

d) Place the next routine call inside a C `while` loop that terminates when the loop control value is `null`. When a next routine cannot access any more target objects, it shall return a `null`.

he following example, `display_net_names`, uses a next routine to display the names of all nets in a module.

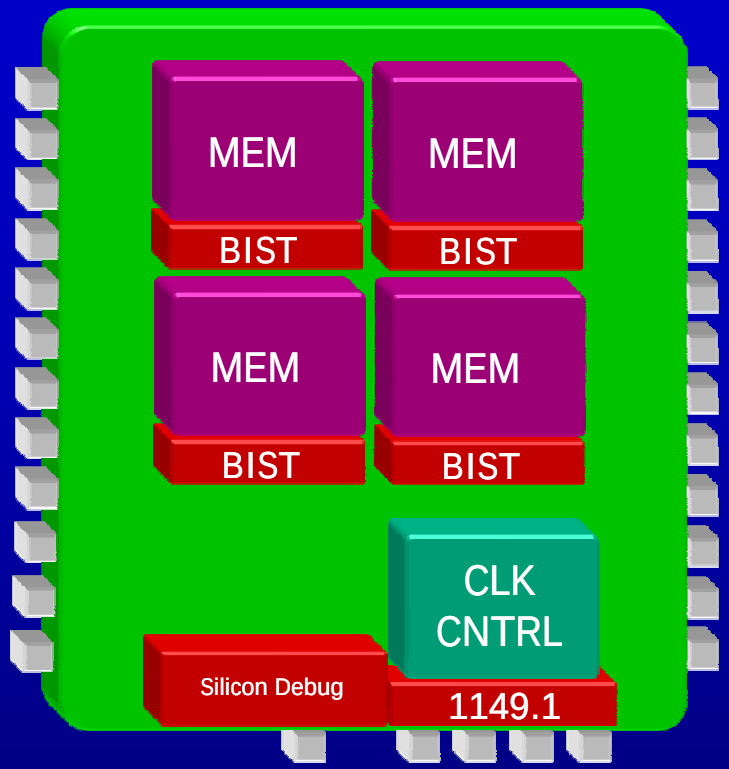
```
#include "acc_user.h"

display_net_names()
{
   handle   module_handle;
   handle   net_handle;

   /*initialize environment for access routines*/
   acc_initialize();

   /*set the development version*/
   acc_configure( accDevelopmentVersion, "IEEE 1364 PLI" );

   /*get handle for module*/
   module_handle = acc_handle_tfarg(1);

   /*display names of all nets in the module*/
   net_handle = null;
   while( net_handle = acc_next_net( module_handle, net_handle ) )
      io_printf("Net name is: %s\n", acc_fetch_fullname(net_handle) );

   acc_close();
}
```

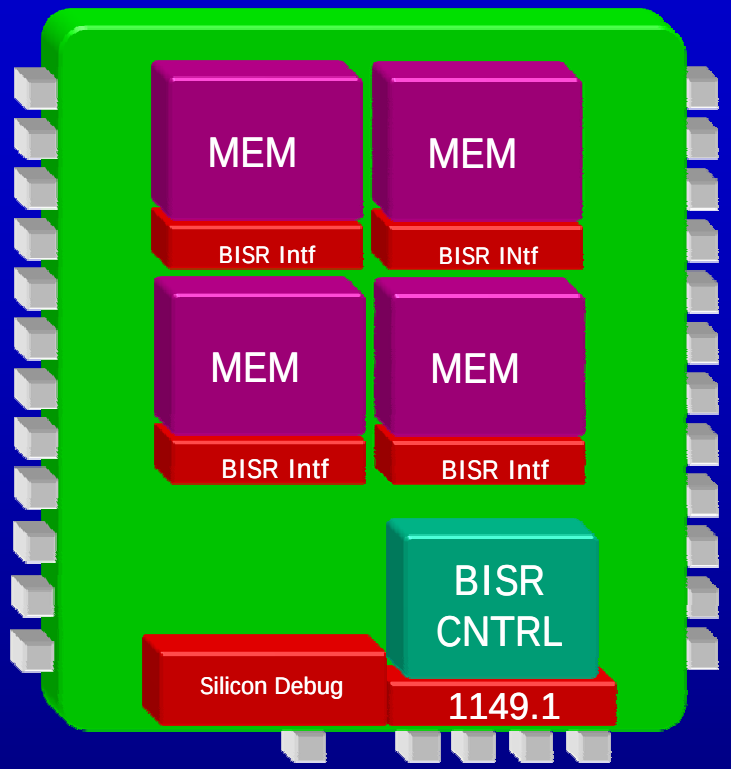
Table 18-4—List of next routines

ACC routine	Description
acc_next()	Get handles to all objects of a set of types
acc_next_bit()	Get handles to all bits of a port or vector
acc_next_cell()	Get handles to all cell modules in the current hierarchy and below
acc_next_cell_load()	Get handles to all cell loads on a net
acc_next_child()	Get handles to all module instances within a module
acc_next_driver()	Get handles to all primitive terminals that drive a net
acc_next_hiconn()	Get handles to all nets connected hierarchically higher to a module port
acc_next_input()	Get handles to all input terminals of a module path or data path
acc_next_load()	Get handles to all primitive terminals driven by a net
acc_next_loconn()	Get handles to all nets connected hierarchically lower to a module port
acc_next_modpath()	Get handles to all path delays in a module
acc_next_net()	Get handles to all nets in a module

Section 18 239

On-chip only Instruments

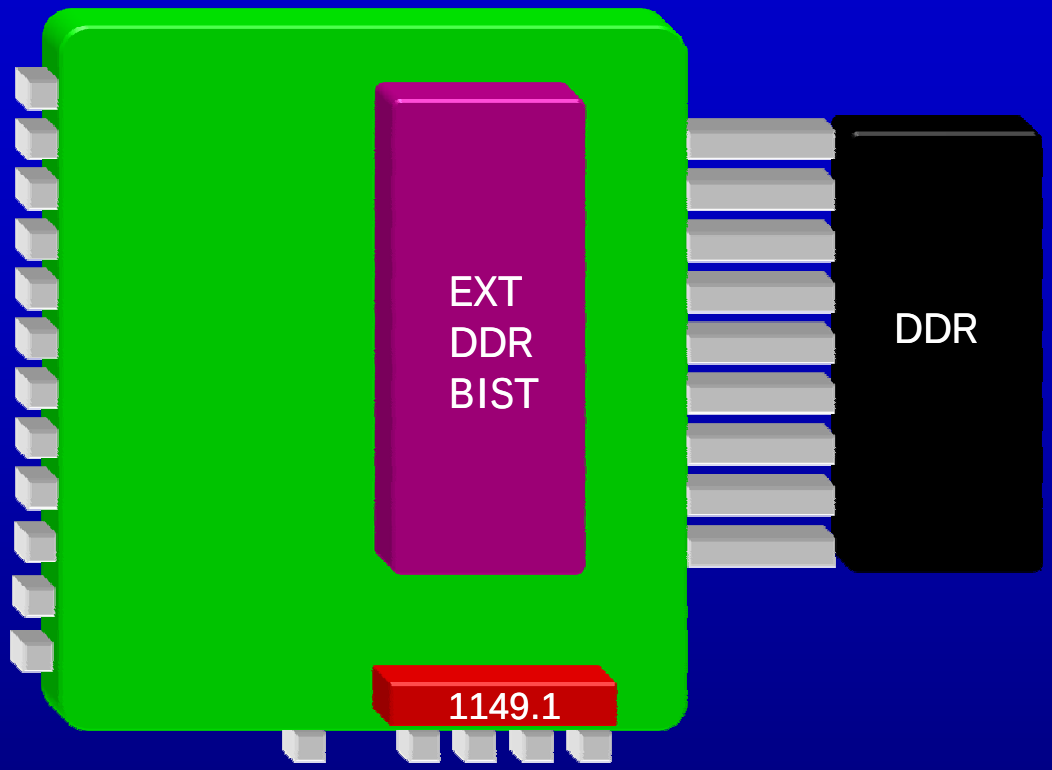




HDL needs dependency attribute
- for insertion
- for test generation

HDL needs attribute for concurrency
- BSDL extension to tell PCB level tools what to execute
  - attribute tying HDL with a particular IC instance

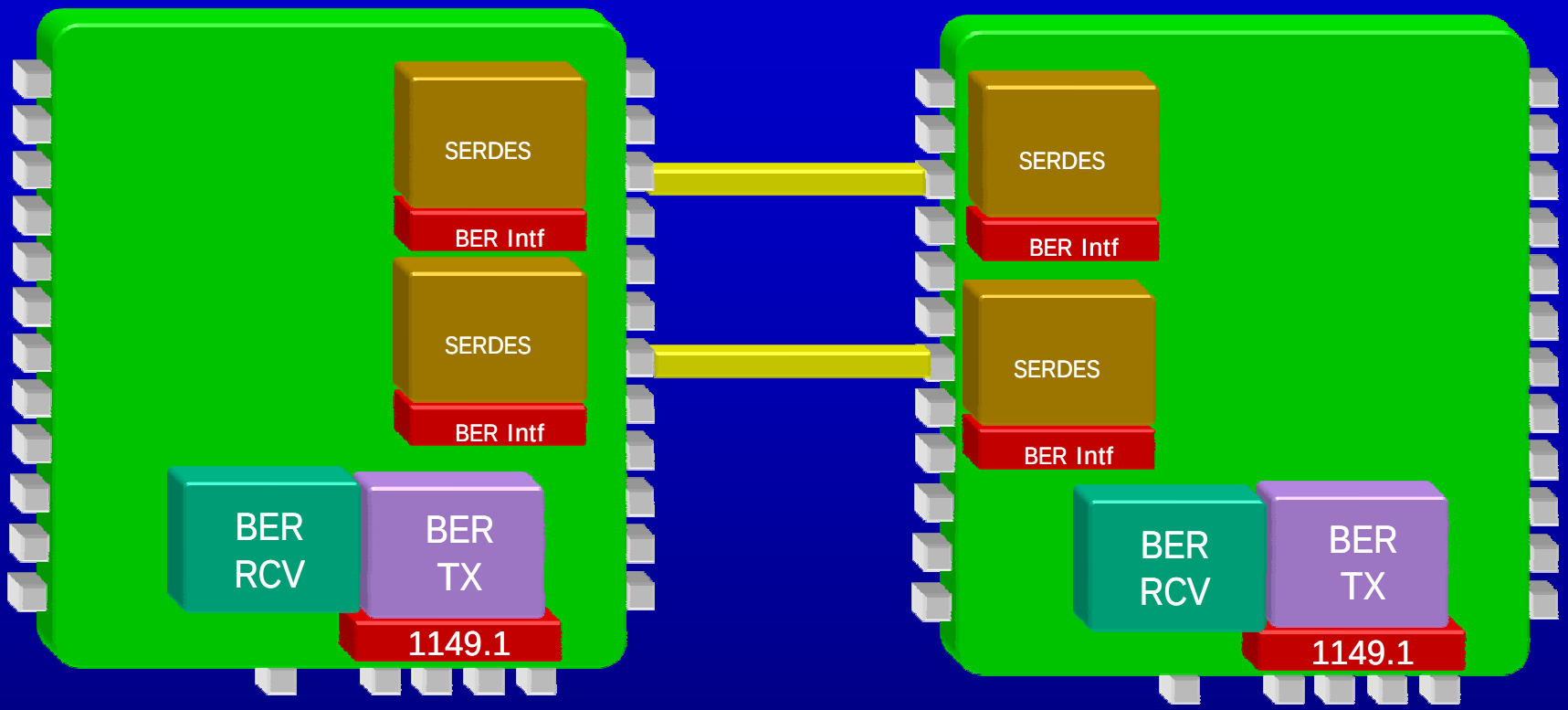
On-chip to non-IJTAG device or instrument



DDR
FLASH
DACs to DMM
On-chip LA

BSDL extension needed for
- Board DFT tools to indicate coverage
- ATPG tools to identify and include test

Chip-2-chip Instruments

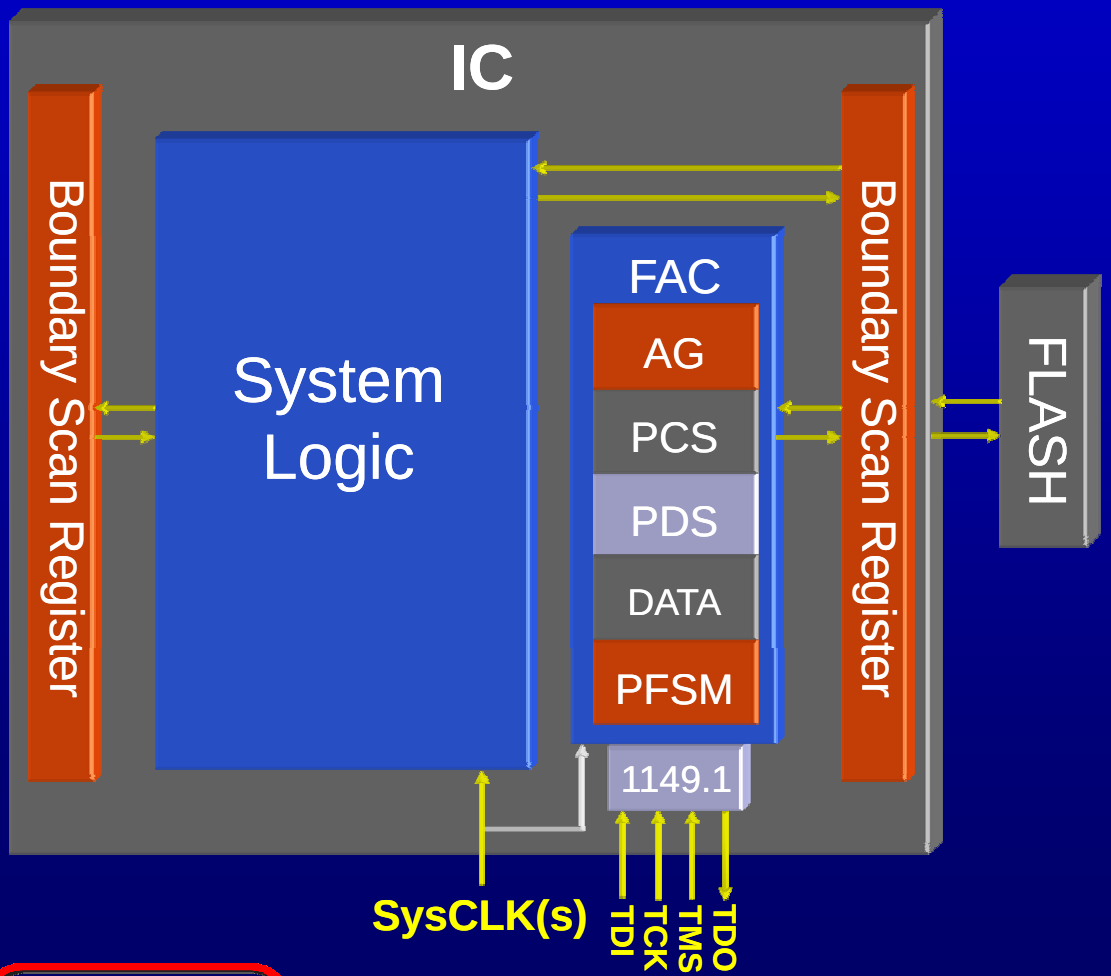


Even with HDL + PDL (adequate?)
BSDL extensions needed

- Attribute for instrument pins connection/capability
- Attribute to tell PCB level DFT tools pins can be tested
- Attribute to tell ATPG tool how to execute PCB level tests

Complex Instruments

Off CPU FLASH programmer

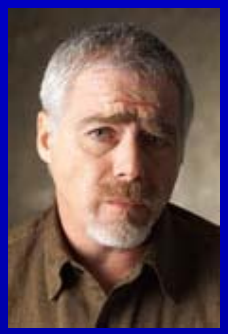


AG: Address Generator
PCS: Programmable Control Sequencer
PDS: Programmable Data Sequencer
DATA: Data Register
PFSM: Programmable Finite State Machine

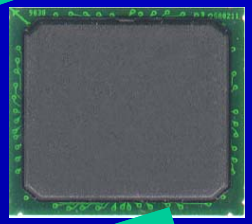
Need Plug ‘n Play with GUI

Instrument Providers

How many Tool providers do I support?

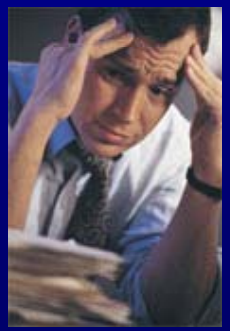


Validated Just on IC tester Or didn’t (Recall BSDL problems)



How many Instruments must I validate? (recall CPU Emulation History)

IC on PCB First Time



Must be tool problem!

Test Engineer What reg bit means what? What is it supposed to do? Something is wrong

What exists today in 1149.1


```
Use intellitechextension.all;        --Get Intellitech BSDL Extensions

--- cut ------


  attribute REGISTER_ACCESS of ttl74bct8374 : entity is
       "BOUNDARY (READBN, READBT, CELLTST),"   &
       "BYPASS (TOPHIP, RUNT),"   &
       “INTERN_SCAN[33010] (INTSCAN),” &
       "BCR[2] (SCANCN, SCANCT)";   -- 2-bit Boundary Control Register

----  cut -----------
attribute REGISTER_NAME of ttl74bct8374 : entity is
       "Duplicate_D_Bus[8] (BOUNDARY[8,15]),"&
       "BCR_bus[2] (BCR[1,0]),"&
       "Partial_D_Bus_2[4] (D[5], D[3], D[7], D[6]),"&
       "Partial_D_Bus[4] (D[1], D[3], D[5], D[7]),"&
       “Bits53[3] (D[5,3])";
```

REGISTER_NAME like HDL Alias

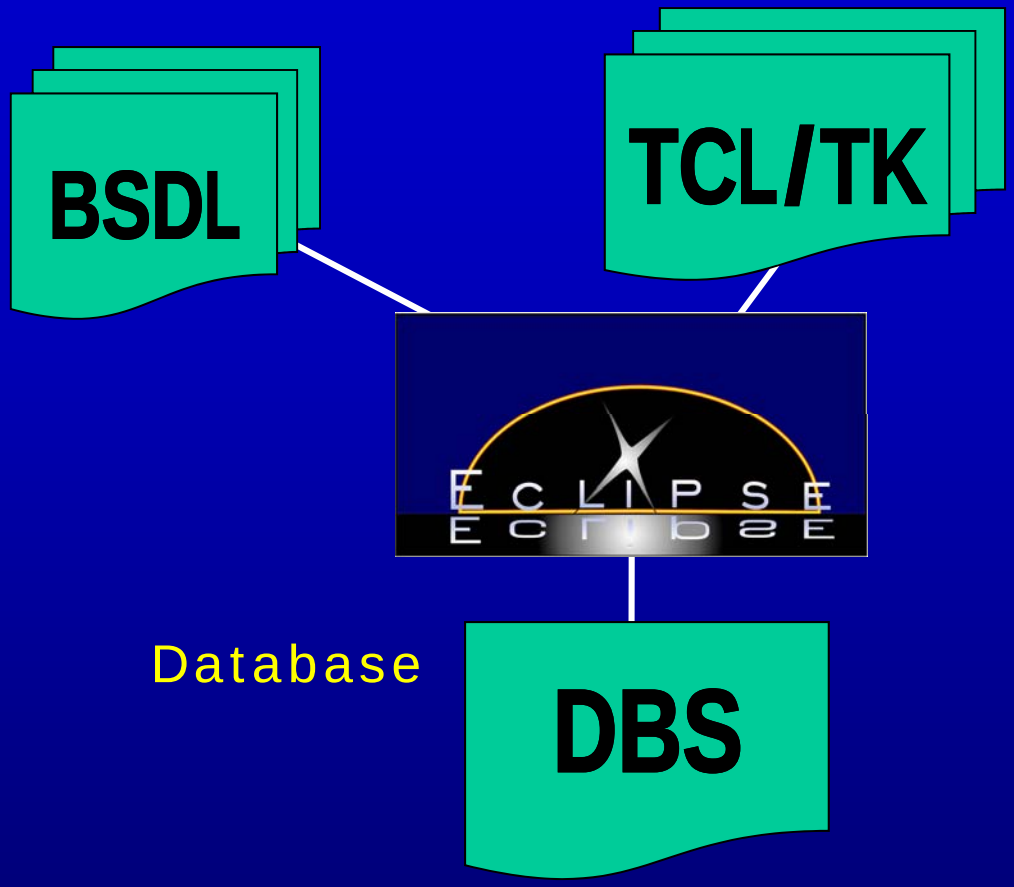
IJTAG-like internal scan access


```
"fs[4](intern_scan[21119,21122])"&
"Hierarchy_Level 2 i_grp_sysadrl;"&
"Hierarchy_Level 3 i_sysadr__;"&
"Hierarchy_Level 4 oeff;"&
"Hierarchy_Level 5 i0;"&
"fs[20](intern_scan[21123],intern_scan[21126],intern_scan[21129],intern_scan[21132],"&
"intern_scan[21135],intern_scan[21138],intern_scan[21141],intern_scan[21144],intern_scan[21147],intern_scan[21150],"&
"intern_scan[21153],intern_scan[21156],intern_scan[21159],intern_scan[21162],intern_scan[21165],intern_scan[21168],"&
"intern_scan[21171],intern_scan[21174],intern_scan[21177],intern_scan[21180])"&
```

Like ‘Alias’ and ‘symbol’

```
--  format register/bus:MNEMONIC (bit pattern, data, data)

attribute MNEMONIC of ttl74bct8374 : entity is
      "D:Both_Zero (00000000)";


attribute MNEMONIC of ttl74bct8374 : entity is
      "BCR_bus:BCR_HIGH (11)";

attribute MNEMONIC of ttl74bct8374 : entity is
      "BCR:BCR_HIGH (11)";

attribute TDI_MNEMONIC of ttl74bct8374 : entity is
      "Q:HexAA (10101010),"   &
      "Q:Hex55 (01010101),"  &

-- extra data fields for future mnemonic support
      "Q:LDA   (00000010, XXXXXXXX),"  &
      "Q:JMP   (00000011, XXXXXXXX, XXXXXXXX)";


attribute TDO_MNEMONIC of ttl74bct8374 : entity is
      "D:HexAA (10101010),"   &
      "D:Hex55 (01010101)";

end ttl74bct8374;
```

BSDL Extentions



Database

Registers and Mnemonics Available in GUI and scripting

```
setUUT BCR_bus BCR_HIGH
setUUT Q       HexAA
setUUT Fs.i_grp_sysadrl.i_sysadr__.oeff.i0  0x12345
```

PDL / TCL comparison

PDL similar to what we've had in 1149.1 TCL/TK for 10+ years

```
iwrite clk_div 3'b111
iwrite free_run 1'b1
iapply
iwrite background 8'h55555555
iwrite low_addr 1'h4
iwrite high_addr 1'h7
iwrite alg 1'b11
iwrite run 1'b1
iread reg_id
iapply
```

```
Set hierarchy brd.U22
setUUT clk_div b111
setUUT free_run 1
DRScan
setUUT background 0x55555555
setUUT low_addr 4
setUUT high_addr 7
setUUT alg b11
setUUT run 1
DRScan
Set regval [getUUT reg_id h]
Puts "idcode = $regval
```

Relatively easy to add IJTAG extensions to TCL through SWIG
- could provide source to WG

TCL/TK community

Altera – Quartus FPGA development Environment
e
Cadence – NC-Sim
Mentor Graphics – ModelSIM
Intellitech – Eclipse & ScanExecutive
Synopsys – Prime Time, Formality and Design Compiler
(Synopsys moved from DC_SHELL to TCL),
Synplicity – Symplify
Xilinx ISE – Integrated Software Environment

-The death of TCL/TK has greatly been exaggerated
- check www.tcl.tk 15th annual meeting was Oct 28th
-Language popular in IC/FPGA design industry
-Easy for IC designers to write without worrying about object oriented programming
- Can be expanded to include object oriented programming

Why TCL/TK

My requirements:
Need functional description which is executed
not another language to parse with ambiguity
Need fast execution – interpreted but byte compiled
Need for single-step/robust dev env – translation from lang to lang prevents this
No compiler/linker (runs within jtag tool)
Need GUI
Need built-in event driven (for GUI)
Easy for non-software engineers to develop
Embeddable/Cross platform (even a tcl engine in VHDL!)
Open source – community can have access for
tool development + instrument development
Extendable (add GPIB, object oriented incr tcl, etc)
Solid history

Library Element Scanfpga.tcl

```
Proc pcb_test  { refdes {tap 0}  }
{
# Execute an external MEMORY BIST by loading a 1149.1 instruction
StatusMessage “Executing Memory BIST for $refdes"
# Reset Scan-Board
RST $tap
# Put ASIC into RAM BIST mode
setUUT $refdes.INSTRUCTION RAMBST
IRScan $tap
runTest TCK 44000 IDLE IDLE 0    # Send 44000 TCK signals
DRScan $tap # Get the result
set result [getUUT $refdes.BISTREG d]
# BISTREG is 4 bits, we want just the upper 2 bits
set result [expr 12 & 0x$result] # perform math to extract
if {$result == 4} {# Tell the user the results of the test
        StatusMessage “$refdes RAM BIST PASSED.”
        return PASSED
}
if {$result != 4} {
        StatusMessage “$refdes BIST FAILED.”
        return FAILED
}}
```

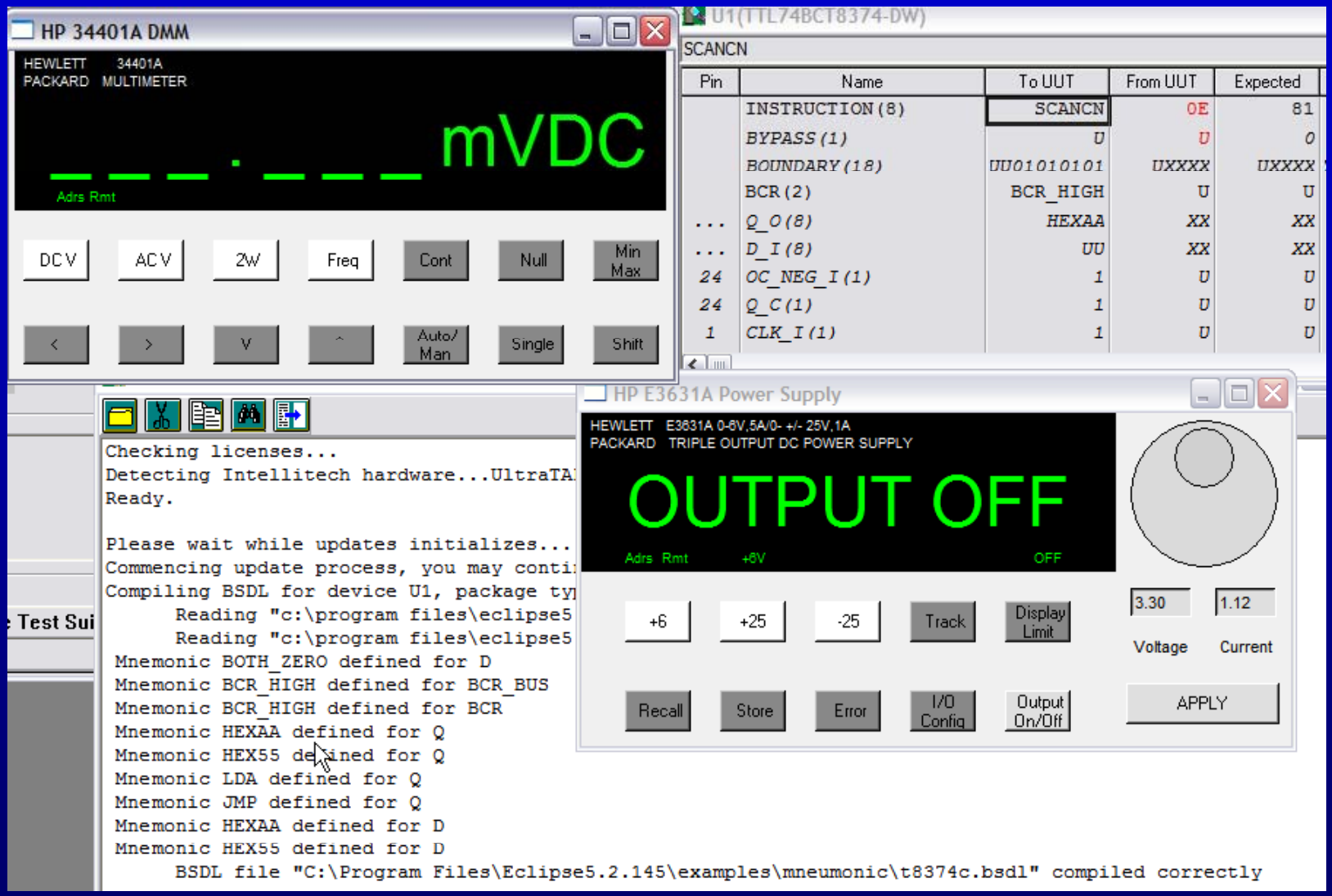
PCB level tools find U22
of type scanfpga in netlist
-Looks for scanfpga.tcl in lib
-Calls pcb_test

-All libs have pcb_test
Or stub. Additional params
Also can be included

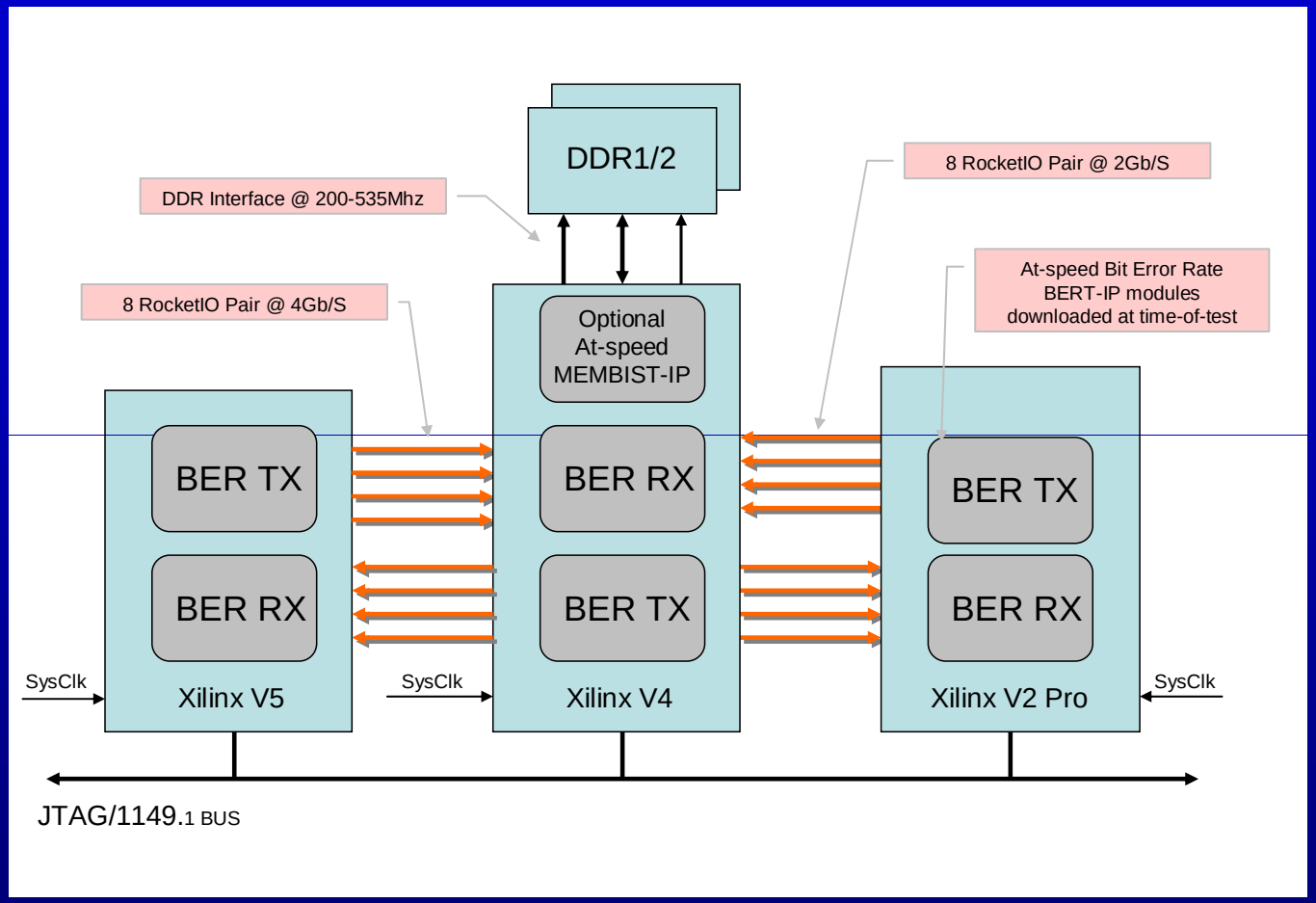
More TCL extensions

```
set wglFile "alarm_clock.wgl"
set dbFile "alarm_clock.dbs"
# Apply WGL with diagnostics
setTapVoltage 1.8v
setTAPFrequency 20mhz
set result [applyFile $wglFile "TetraMAX"]


# call external ‘protected’ binary
set EXEC_FILE  "C:/myprogram.exe"
set EXEC_SCRIPT  “Param1 Param2 Param3"
catch {exec $EXEC_FILE $EXEC_SCRIPT &} result

# & means run in background
```

TCL/TK Instruments & 1149.1



PCB Level BER with BERT-IP

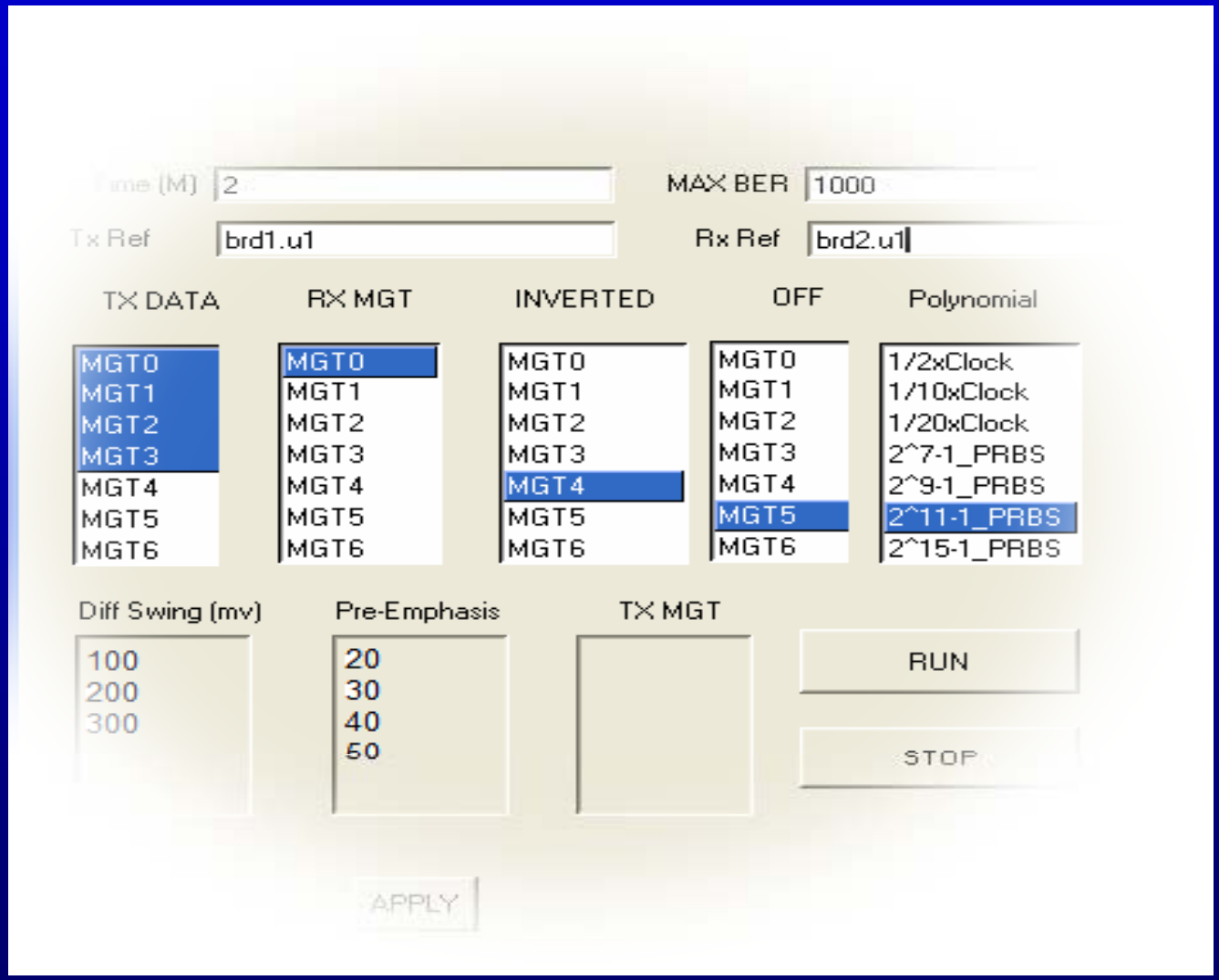


Need to identify what connections
Present and what capability on each pin
- DFT Analysis + PCB Test Generation

BERT-IP GUI in TCL/TK

Interactive
Or
Select and
Generate

Lots of
Registers
To be set
-Need
Plug'nPlay



Like functional tests?

Non-BIST instruments may require inline diagnostics like a functional test

```
## check for tx and rx lock on mgt's
set txlock1_check [txlock_check $board_ref89 $mgt_to_test 0]
if {$txlock1_check == 0 } {
 errorMessage 0 "Not able to acquire txlock from $mgt_name at $ref89\n"
                set txlock_fail  1
                set passed1 0
              } else {
                set txlock_fail 0
                set passed1 1
              }
```

PDL / TCL comparison

BSDL extensions (attributes)
+
HDL (register names + access)
+
TCL/TK with iwrite/iread

Is PDL stand-alone needed?

We have SVF, STIL, WGL already which define scan ins/outs

TCL/TK supports Python

TclPython package embeds a python interpreter.

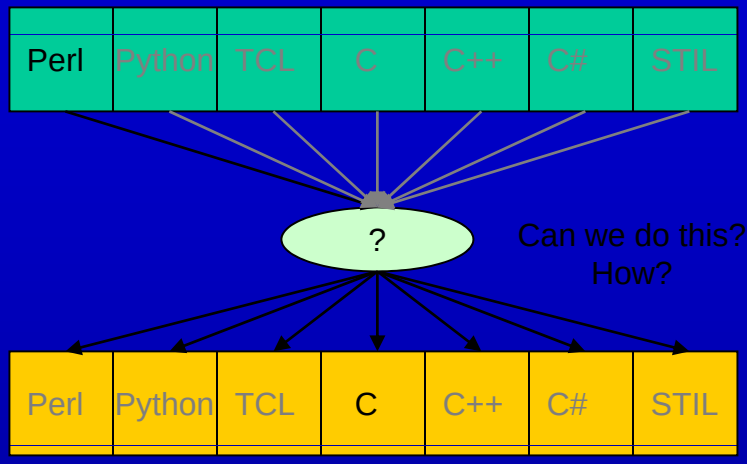
```
package require tclpython
set py [python::interp new]

# $a and $b are the text of the two revisions -- inject them into
the python interpreter
$py exec "
from difflib import HtmlDiff
a = \"\"\"$a\"\"\".split('\\n')
b = \"\"\"$b\"\"\".split('\\n')
hd = HtmlDiff(wrapcolumn=80)
"
ns_return 200 text/html [$py eval "hd.make_file(b, a,
context=True)"]
```

IJTAG basic or complete?

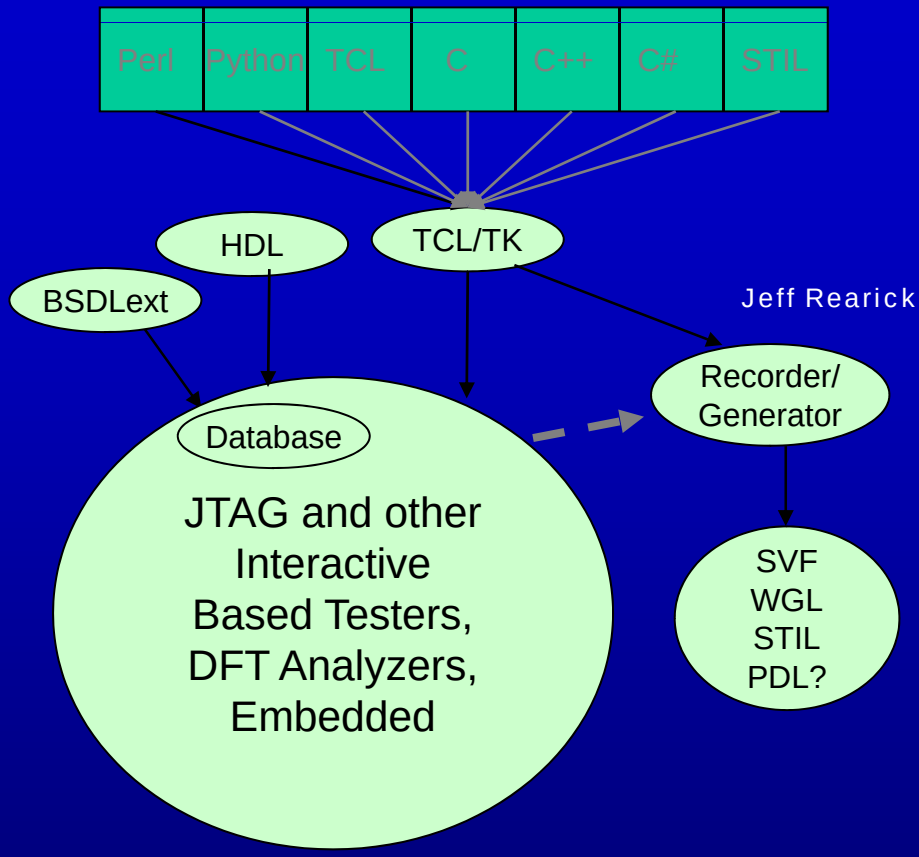
Without standardization of instruments
with plug and play “execute in place” language
with defined procedures for chip, PCB and interactive GUI;
we have nothing more than what we had with JTAG

IJTAG HDL+PDL only
- better insertion
- adds new access mechanism in addition to 1500 and JTAG
- requires users to read data sheets and write access or requires a given tool vendor to support writing interfaces in a proprietary language

TCL/TK + HDL + BSDL ext



Stylianos Diamantidis



Example proc for a .inst library

```
# tcl supports variable length arguments
Proc Info     { instance1 {tap 0}  }

Proc IC_test  { instance1 {tap 0}  }
Proc IC_require  { instance1 {tap 0}  }
Proc IC_GUI    { instance1 {tap 0}  }
Proc IC_concurrent { instance1 {tap 0}  }
Proc IC_userdefined { instance1 {tap 0}  }

Proc pcb_test  { refdes {tap 0}  }
Proc pcb_dft   { refdes {tap 0}  }
Proc pcb_GUI   { refdes {tap 0}  }
Proc pcb_userdefined  { refdes {tap 0}  }
Proc pcb_disable { refdes {tap 0}  }
```

Not a complete list, but a start. Ideally, would Like structured routines in the standard that support

Automation the way we have EXTEST automated