BASICS of PDL and how it works

CJ Clark, Intellitech Corp.

Business-wise some have chosen to focus on EXTEST or the boundary register
- that is fine, it doesn't change the original scope of 1149.1

1. Overview

1.1 Scope

This standard defines test logic that can be included in an integrated circuit to provide standardized
approaches to
— testing the interconnections between integrated circuits once they have been assembled onto a printed circuit board or other substrate;
— testing the integrated circuit itself; and
— observing or modifying circuit activity during the component's normal operation.
The test logic consists of a boundary-scan register and other building blocks and is accessed through a Test Access Port (TAP).

Some parts of 1149.1 have not addressed complexity of IC design well to accomplish a) and c)

- i.e. Runbist

1.2.2 The use of IEEE Std 1149.1 to test an assembled product

This subclause outlines the use of the boundary-scan circuitry defined by this standard during the process of testing an assembled product such as a printed circuit board.

The test problem for any product constructed from a collection of components can be decomposed into three goals:

a) To confirm that each component performs its required function;

b) To confirm that the components are interconnected in the correct manner; and

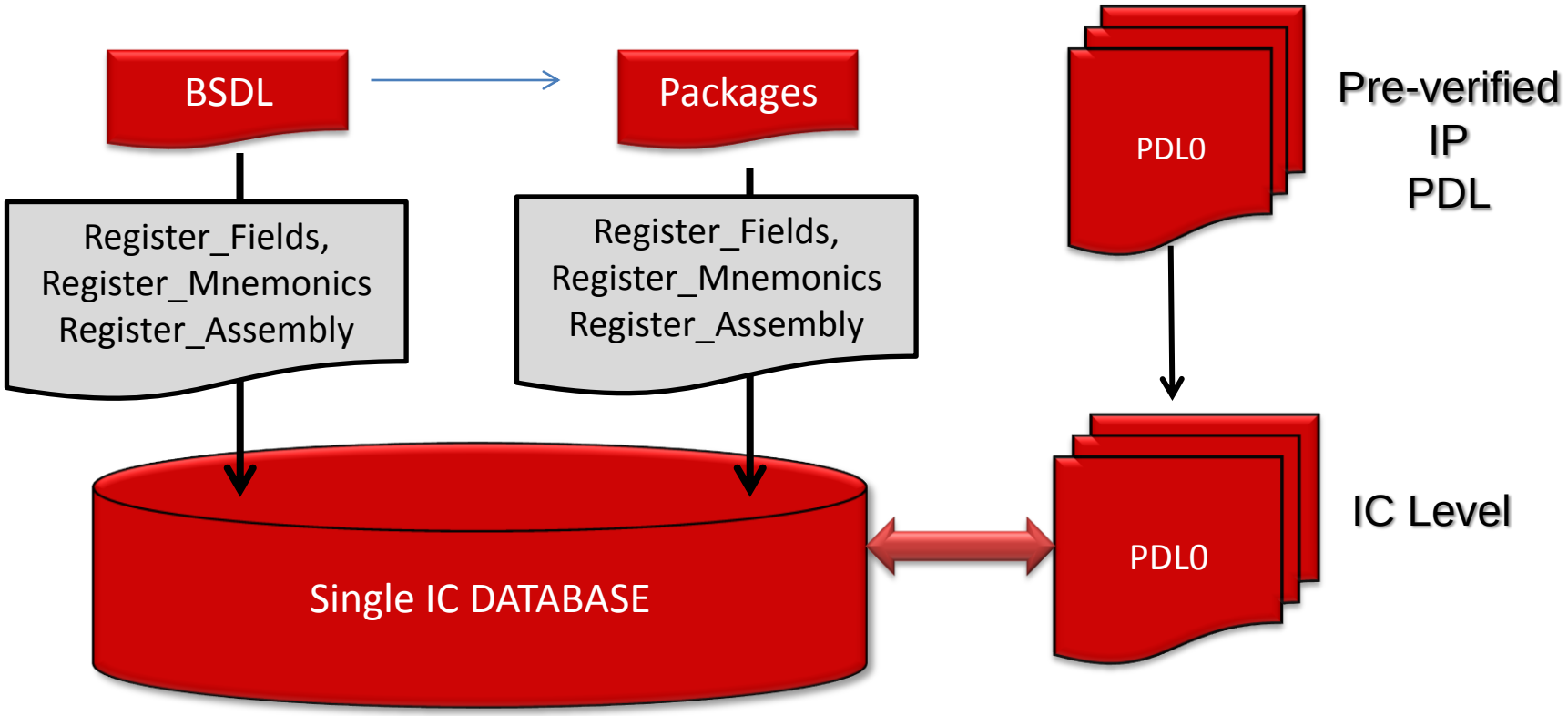
c) To confirm that the components in the product interact correctly and that the product performs its intended function.

While respect is provided to those who want to use just the EXTEST part of the standard, the converse also must adhere for those who have used the standard as it was originally described.

How PDL works:

Basic concept is that BSDL and package files are combined to make a single IC database which is operated on by PDL

- Tool vendors combine single IC databases into board scan chains
- Tool vendors have had proprietary macros/languages before BSDL this is not new or unproven territory



A look at how PDL operates on scan database

	iWrite (in)	iRead (expected)	TDO (OUT)
init-data	UU	XX	XX

CAPTURES fills in iExpected
RESETVALs on cells without PI fill in iExpected

Without any scans you may have expected data

	iWrite (in)	iRead (expected)	TDO (OUT)
init-data	UU	0x11	XX

Tool vendors choice as to what to put in when nothing in field

DEFAULT and SAFE values can initialize iWrite field

	iWrite (in)	iRead (expected)	TDO (OUT)
init-data	0x01	0x11	XX

after iWrite init-data 0x55, database holds 0x55

	iWrite (in)	iRead (expected)	TDO (OUT)
init-data	0x55	0x11	XX

Tool provider is responsible for 'checks'/warnings if new iWrite data overwrites a SAFE value
(Default typically would not be a warning, but tool provider choice

after iRead init-data(1) 1, database holds 0x13

	iWrite (in)	iRead (expected)	TDO (OUT)
init-data	0x55	0x13	XX

During Pattern Generation TDO is not Looked at As the UUT Is not Connected.

Register_fields are just pointers to bits. iWrite mysinglebit 0

	iWrite (in)	iRead (expected)	TDO (OUT)
init-data	0x54	0x13	XX

Mnemonics of course can be used as well

	iWrite (in)	iRead (expected)	TDO (OUT)
init-data	PCIe	Pass	XX

iProc init_setup is used to configure instance specific features of an IC.

Some init_setup steps are consistent from one instance to another.

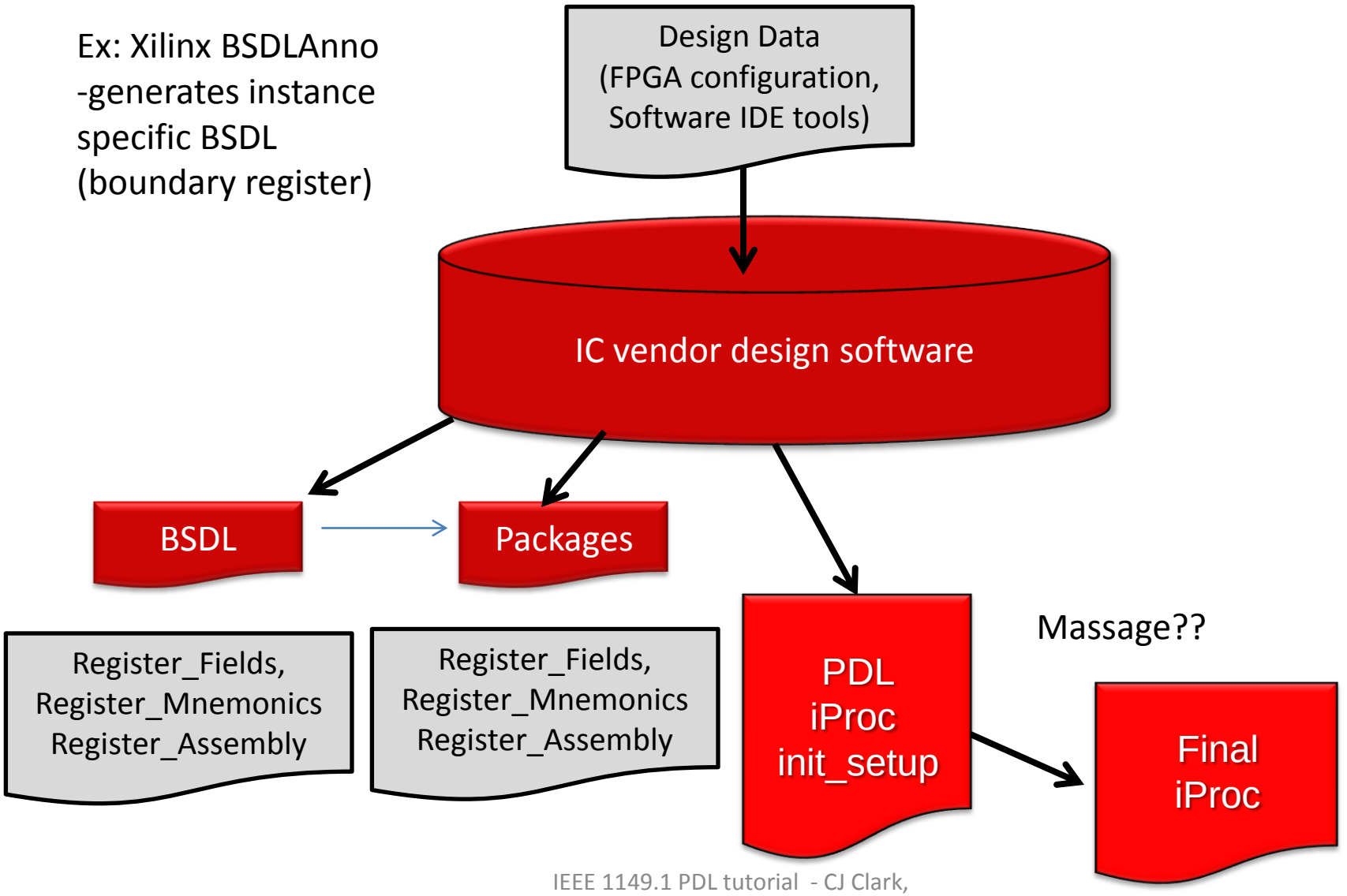
- Ex: Turning PLLs off

Most likely all init_run iProcs are exactly the same from one instance to another. Polling on a status bit for example

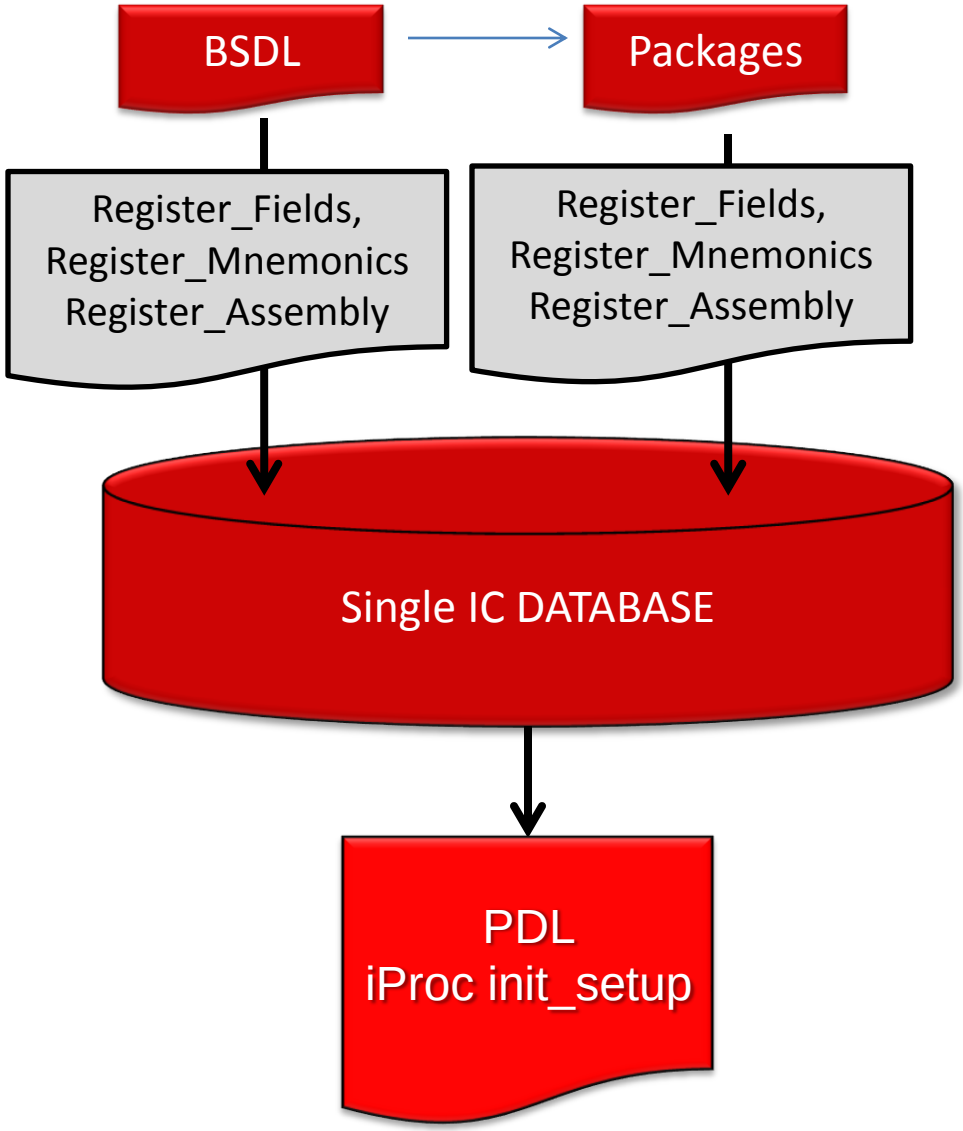
Several methods can exist to enable init_setup which are shown in the following slides.

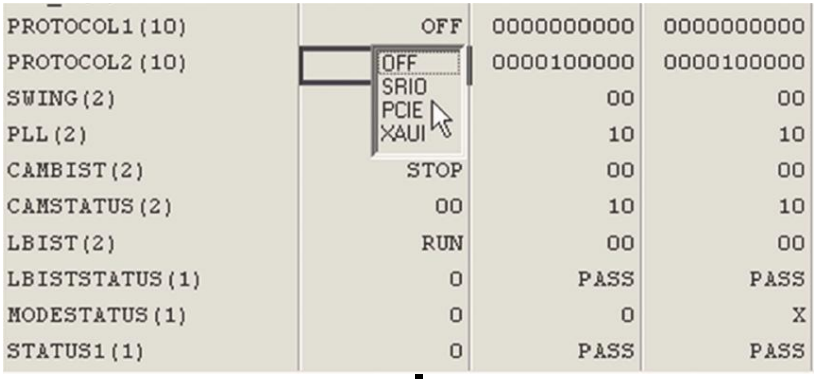
Design Engineer generates iProc from design environment and passes custom BSDL, package files and PDL to test engineer - this may not happen for years though

Ex: Xilinx BSDLAnno -generates instance specific BSDL (boundary register)

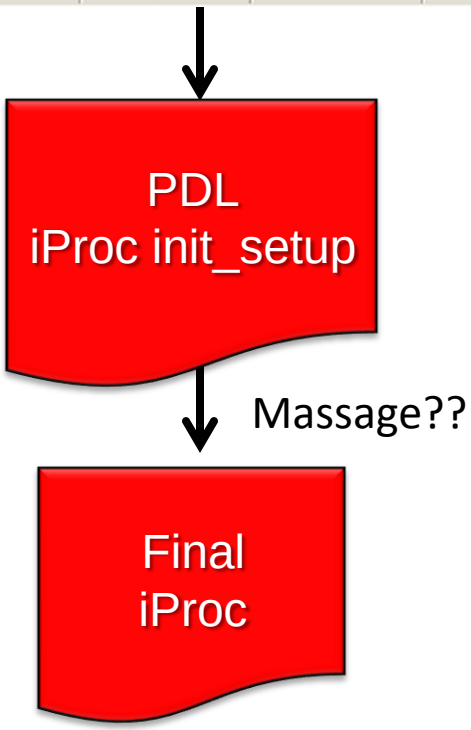


Test Engineer generates settings from BSDL aware GUI





PROTOCOL1(10)	OFF	0000000000	0000000000
PROTOCOL2(10)	OFF	0000100000	0000100000
SWING(2)	SRIO	00	00
PLL(2)	PCIE	10	10
CAMBIST(2)	XAUI	00	00
CAMSTATUS(2)	00	10	10
LBIST(2)	RUN	00	00
LBISTSTATUS(1)	0	PASS	PASS
MODESTATUS(1)	0	0	X
STATUS1(1)	0	PASS	PASS



No netlist says “PCIe” in it. However, netlist can be used to optimize building of custom init_setup for the test engineer. Standard does not specify this

IP provided init_setup for IO. Parameters passed in
- not practical for 1000 I/O with four parameters each

```
iProc  init_setup  {   Proto Swing} {

iWrite  Proto   $Proto
iWrite  Swing   $Swing
iApply
iMatchLoop -begin
iRead Status Ready
iApply
iMatchLoop -end
}
```

For optimization some merging would help
when this init_setup is called at a higher level

IP or IC provided init_setup for IO. Parameters passed in for all I/O???
- not practical for 1000 I/O with multiple parameters each

```
iProc init_setup { IO1_Proto IO1_Swing IO2_Proto IO2_Swing IO3_Proto IO3_Swing IO4_Proto IO4_Swing \
 IO5_Proto IO5_Swing IO6_Proto IO6_Swing IO7_Proto IO7_Swing IO8_Proto IO8_Swing \
....
 IO998_Proto IO998_Swing IO999_Proto IO999_Swing IO1000_Proto IO1000_Swing } {

iWrite IO1_Proto $IO1_Proto
iWrite IO1_Swing $IO1_Swing
iWrite IO2_Proto $IO2_Proto ;# PCIe , SATA, SRIO
iWrite IO2_Swing $IO2_Swing ;# 200mv 300mv 500mv 800mv

iWrite IO3_Proto $IO3_Proto
iWrite IO3_Swing $IO3_Swing
iWrite IO4_Proto $IO4_Proto
iWrite IO4_Swing $IO4_Swing

iWrite IO5_Proto $IO5_Proto
iWrite IO5_Swing $IO5_Swing
iWrite IO6_Proto $IO6_Proto
iWrite IO6_Swing $IO6_Swing

iWrite IO7_Proto $IO7_Proto
iWrite IO7_Swing $IO7_Swing
iWrite IO8_Proto $IO8_Proto
iWrite IO8_Swing $IO8_Swing
.
. <many pages later>
.
iWrite IO999_Proto $IO999_Proto
iWrite IO999_Swing $IO999_Swing
iWrite IO1000_Proto $IO1000_Proto
iWrite IO100_Swing $IO1000_Swing

iApply
iMatchLoop -begin
iRead Status Ready
iApply
iMatchLoop -end
}
```

Large ICs can benefit from PDL1

- description is compact, still recorded out as flat binary tester commands
- IC vendor supplied init_setup
- what are the reasons we would not enable this?

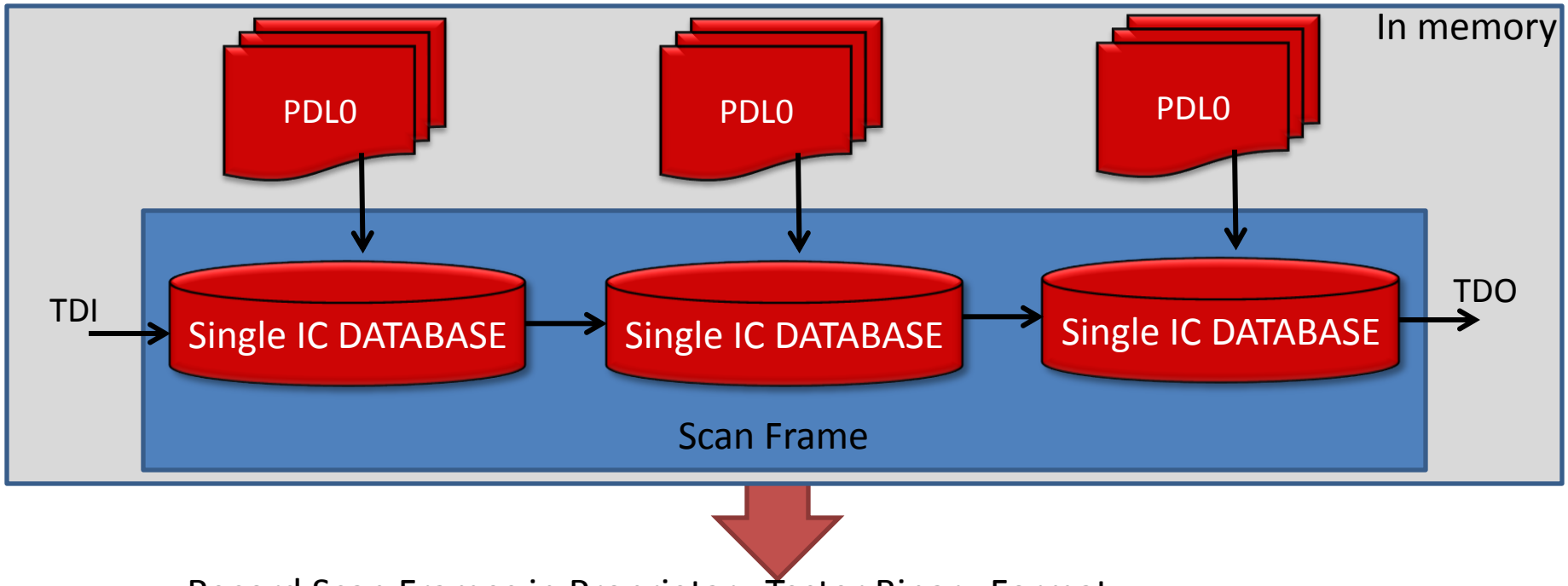
```
iProc  init_setup  { }  {
set i 0   ;# variable
while { $i < 988 } {

iWrite  IO($i).Pullup     ON   ; # register_field
iWrite  IO($i).ACMODE   OFF  ;#  use DC I/O only
iWrite  IO($i).WEN   ON
incr i
}

iApply
}
```

PULLUP, ACMODE and WEN are registers
Register_assembly supports arrays of I/O.

Each PDL routine may be executed sequentially

- must stop at iApply (leave in interpreter)
- execute next PDL
- At 'last' PDL perform iApply
- continue with first PDL

Various optimizations
Available to vendor
Such as slicing board
Scan frame, then piecing
In tester format



Record Scan Frames in Proprietary Tester Binary Format
(record separately register names/info tags as you wish for 'simple' diagnostics)

If one PDL ends before the others, then the scan frame data repeats
-For discussion: 'merging' is not 'on' by default – hence the iMerge command
Consider for init_setup or init_run iProc some may want it to be the default

Use Models

PDL0 iProcs

Designed for use with tester hardware
- memory behind pin, sequencer type with stim/exp/mask

“load and go” type operation

For use in Production or Field where database of BSDL info is not coupled to Test application.

High-speed - hardware comparison - TDO data is compared with expected and masked by X bits

iProcs init_setup, init_run could be defined as PDL0 only (potentially)
(some may view this as challenging and would like
the ability to use PDL1 to describe init_setup)

```
# A PDL File for IC vendor XYZ's ABCIC would be as follows :

iPDLLevel 0 –version IEEE1149_1_2012  ; # level-0 PDL only
iProcGroup ABCIC   ;# entity or package name  -iProcTarget

iProc init_setup {  param } {

iWrite Core1_PLL1  $param   ;# REGISTER_FIELD
iWrite Core1_PLL2  $param
iWrite Core2_PLL1  $param
iWrite Core2_PLL2  $param
iWrite Reg  ON  ;# triggers just from a DR scan
iWrite RegEN ON ;#
iApply
iWrite RegEN OFF ;# required to leave init_setup in non-triggered mode
iApply
}
# this is the same for all ABCIC's
iProc  init_run {}  {
  iRunLoop 10000    ; # 10,000 TCK cycle delay
  iRead   init_status(1)  Pass
  iApply

}

# this is the same for all ABCIC's
iProc  main {}  {
.
.
}
iProc userdefined {} {
…
}
# EOF
```

In memory:

ABCIC.init_setup
ABCIC.init_run
ABCIC.main
ABCIC.userdefined

P1687 and 1149.1

iCall U3.init_setup

(look up u3 what it is (ABCIC)
And then call the iProc Associated with it
In the context of U3.)
iRead U3.init_status(1)
iWrite U3.Core1_PLL1

In memory:

ABCIC.main
ABCIC.userdefined
ABCIC.init_setup
ABCIC.init_run

Tool is seeing:

```
iWrite U3.Core1_PLL1  OFF
iWrite U3.Core1_PLL2  OFF
iWrite U3.Core2_PLL1  OFF
iWrite U3.Core2_PLL2  OFF
```

Strip proc name off, look up instance path type <entity or package file>
Pass the <instance path> to the proc formed by <entity or package file>.proc

```
iPDLLevel 0 –version STD_IEEE_1149_1_2012
iProcGroup U3 ; # Associate the following procs with U3

# this procedure becomes U3.init_setup internally to the PDL
interpreter
iProc  init_setup { }  {
  U3.init_setup           ;# call IC vendor init setup
  iWrite  Clock       125Mhz ; # use of BSDL mnemonics
  iWrite  Voltage     0x40           ; # use of hex values
  iWrite  Protocol    PCIe           ; # use of BSDL mnemonics
  iApply

}
iProc main {}  {
U3.main                   ;# call on chip tests xyz
#U3.membist               ;# tool generates this commented out
                          ;# test engineers enable
}
#end of file
```

Probably no way to automate everything. Test engineers will still have jobs

Also In memory:

ABCIC.init_setup
ABCIC.init_run
ABCIC.main
ABCIC.userdefined

In memory: U3.init_setup
U3.main

1149.1 Same. Operation. (Drop iTarget). However
- note loss of knowing type
- solution? Easier to use instance names has been suggested

```
iCall U3.main                ;# call instance U3 or ABCIC main??????
iCall U3.init_setup          ;# call instance U3 or ABCIC init_setup??????
```

Why -direct here?
(parameters come after iProc)

```
iCall –direct U3.main        ;# call it directly no lookup
iCall U3.main
iCall U3.init_setup          ;# call prebuilt init_setup for ABCIC
iCall  -direct U3.init_setup ; # call instance specific init_setup
```

In memory: U3.init_setup
U3.main

Also In memory:

ABCIC.init_setup
ABCIC.init_run
ABCIC.main
ABCIC.userdefined

1149.1 Are init_setup by default Merged?
up to tool vendor to merge or not merge init_setup?

init_run MUST be merged in WG current collective thinking

```
iCall U3.init_setup              ;# call prebuilt init_setup for ABCIC
iCall  -direct U3.init_setup     ; # call instance specific init_setup
```

Can this be done to reduce?

```
iMerge –begin                    ; # reduce iApply
iCall U3.init_setup
iCall –direct U3.init_setup
iMerge -end
```

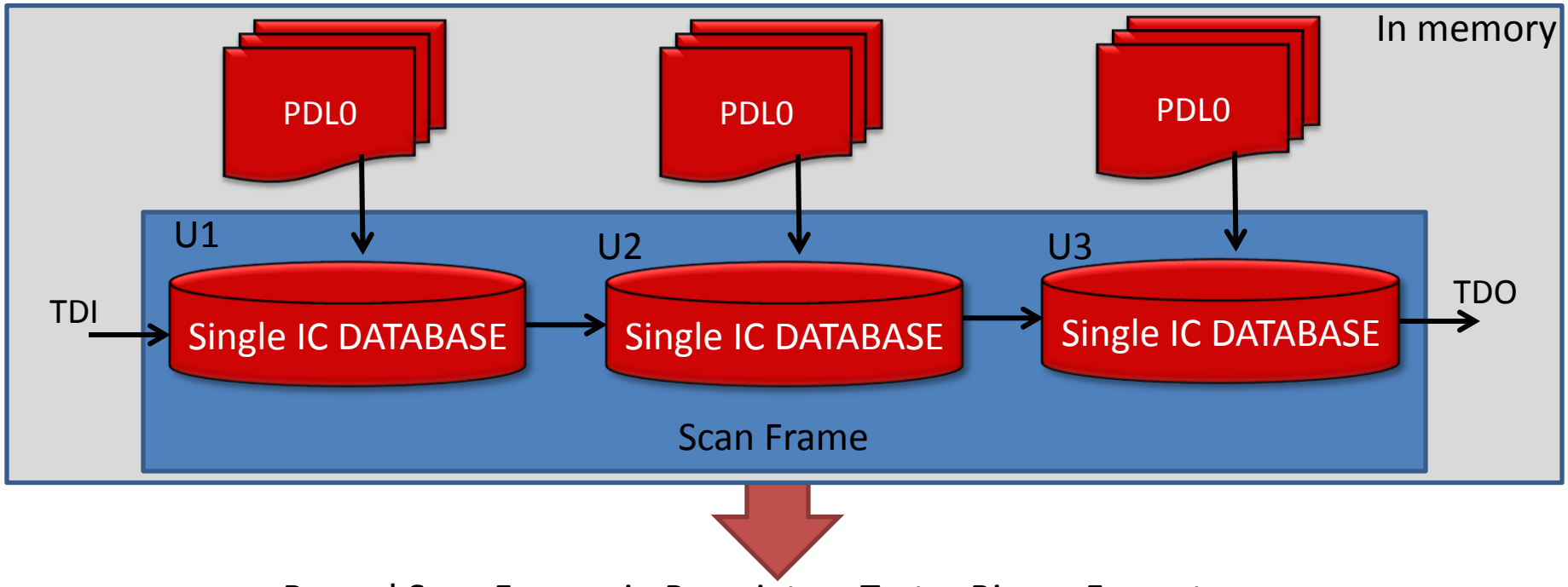
In memory: U3.init_setup
U3.main

Also In memory:

ABCIC.init_setup
ABCIC.init_run
ABCIC.main
ABCIC.userdefined

iLoop/iUntil Loop in an iProc for U2?

- record the loop start in tester binary format, max count
- process U1 and U3 as usual
  - continue to next iApply (other choices don't seem to work
  - load same frame data repeatedly, clear expected data (XXX) on U1/U3
  - scan frame)
  - record same frame data only on PDL which ends for U1 and U3
- any PULSE1/PULSE0 cells which take on 1 after 0 must be set back after iApply
- record end in tester binary format
- resume recording round-robin on each PDL/iProc
-

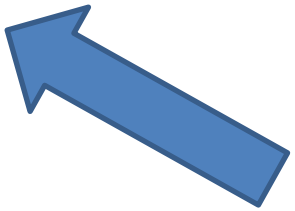


Record Scan Frames in Proprietary Tester Binary Format

iLoop/iUntil

- two examples in draft
- used for ‘rdy’ or ‘bsy’ type polling
- tools can merge or not merge as they see fit (not possible for init_run)
- iLoop can 'block' and other procs would finish before loop section

```
iProc XYZ_EXIO {} {
# external voltage may be coming up or non-stable get
# 10 good readings before proceeding otherwise remaining tests may
# have failures due to instability
iLoop -begin  ;# repeat
iWrite ADDR   VREFADDR
iWrite WE  0
iApply
iWrite WE  1
iApply                  ;# register dump from OS
iRead  VREF-VOLTAGE 0xC2  ;#  Loop to make sure VREF is stable
                          ;#  Error if any reading is incorrect
iApply  -nofail
iUntil -match
#; VREF stable
}
```

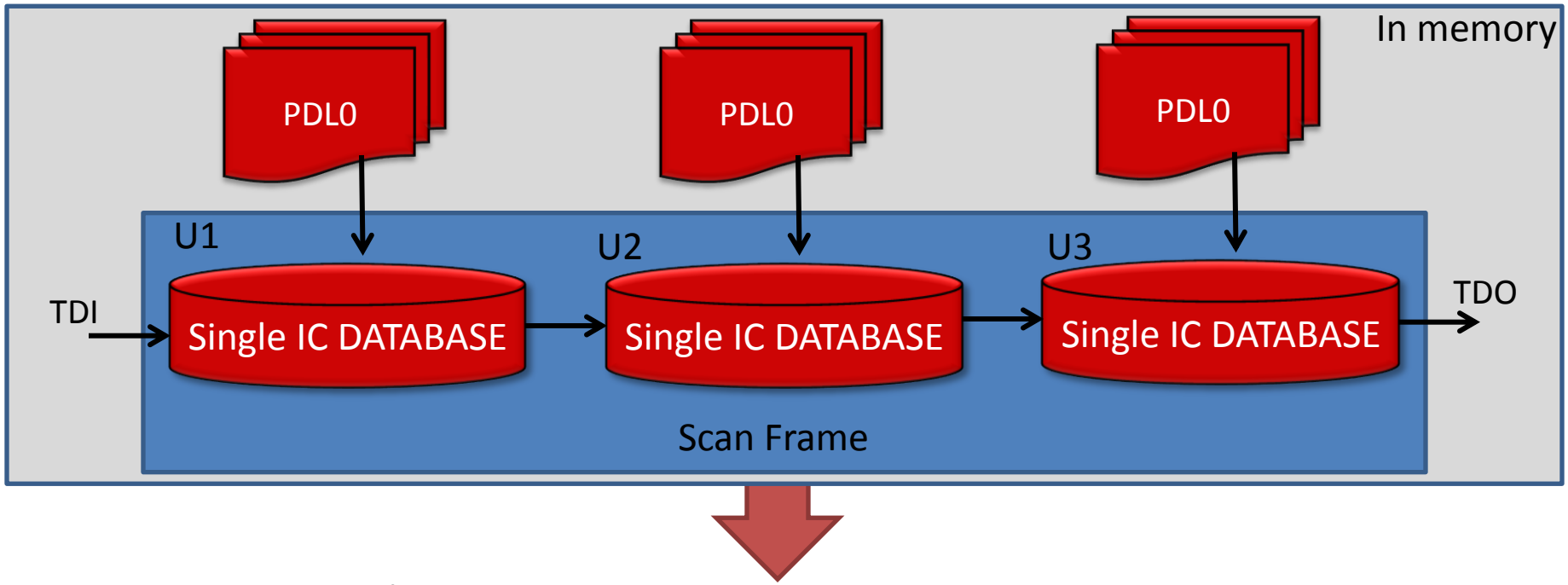


This is currently an 'implied -nofail'
(All iApply inside) suggest
iApply -nofail on appropriate ones

```
iWrite data  0x01
iWrite WE   1 ;# PULSE1
iApply
iRead  Status 0xFF
iWrite data  0x02
iWrite WE   1
iApply
iRead  Status 0xFE
iWrite data  0x03
iWrite WE   1
iApply

…
```

```
iMisMatchLoop 3 -begin
iWrite ADDR   VREFADDR
iWrite WE  0
iApply
iWrite WE  1
iApply
iRead  VREF-VOLTAGE 0xC2
iApply -nofail
iMisMatchLoop -end
```

```
iWrite data  0x01
iWrite WE   1 ;# PULSE1
iApply
iRead  Status 0xFF
iApply
iWrite data  0x02
iWrite WE   1 ;# PULSE1
iApply
iRead  Status 0xFE
iApply
iWrite data  0x03
iWrite WE   1 ;# PULSE1
iApply
iRead  Status 0xFD
iApply
```



Record Scan Frames in Proprietary Tester Binary Format

iMatchLoop/iMisMatchLoop
-Tool responsible for optimization (if any)
- Tool can simply stop at iM/iMM commands and finish other iProcs
- Tool can continue to merge (standard is silent on this)
-Tool responsible for practical limits (maxcnt = 999999?)
-Note in U3, PULSE1 being reset to 0 to match cell return to 0
-PDL writers always need to verify/don't end in triggering states
-Tool can simply take diminutive case of maxcnt = 1
-Tool responsible for exiting on failures/register dumps etc as desired

U1			U2			U3		
Data	Status	WE	ADDR	WE	VREF-VOLTAGE	Data	Status	WE
0x01	XX	1	VREFADDR	0	XX	0x01	XX	1
0x02	0xFF	1	VREFADDR	1	XX	0x01	0xFF	0
0x03	0xFE	1	VREFADDR	1	0XC2	0x02	XX	1
0x04	0xFD	1	VREFADDR	0	XX	0x02	0XFE	0
0x05	0xFC	1	VREFADDR	1	XX	0x03	XX	1
0x06	0xFB	1	VREFADDR	1	0XC2	0x03	0XFD	0
0x07	0XFA	1	VREFADDR	0	XX	0x04	XX	1
0x08	0XF9	1	VREFADDR	1	XX	0x04	0XFC	0
0x09	0xF8	1	VREFADDR	1	0XC2	0x05	XX	1

automatic return to 0 for WE on iApply without iWrite WE

explicit enable for WE on each iApply

PDL rolled out into tester binary format

ifTrue/ifFalse - High-speed flow control

-Branching based on miscompare/compare of expected data

-Carol's pins need a check and exit (tester stop-on-fail is not guaranteed to be on)

- simple example, ignore argument on how many different package files and PDL can be delivered as an alternative that is not scalable and requires more integration at top level

- Multiple file solution doesn't work around problem that production may have three or four versions of IP in production line or in field across multiple ICs. We'd like the IP to deal with its own init_setup variations to lower the downstream costs

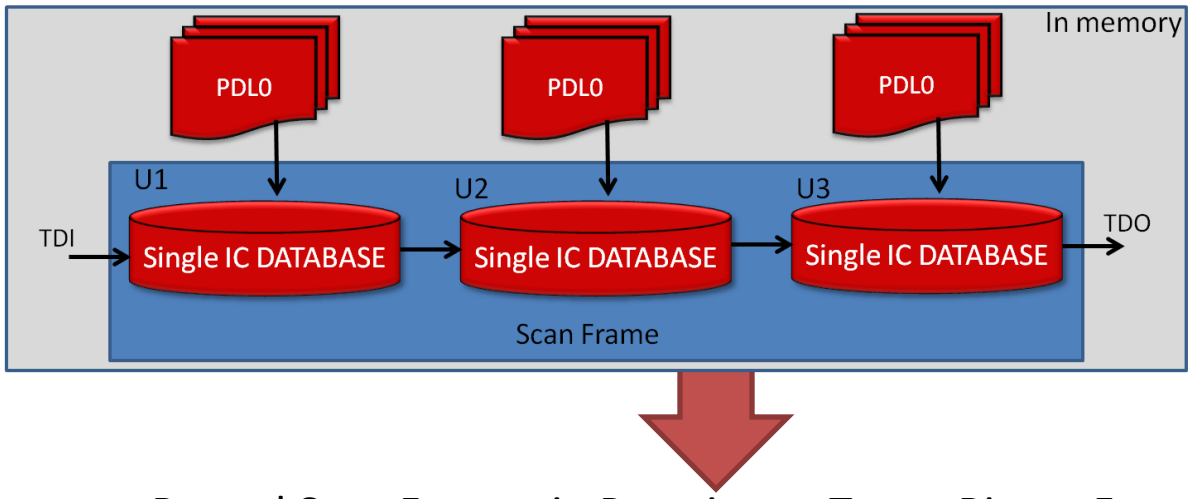
```
iProc INIT_SETUP {} {
iRead VERSION 0b01
iApply -nofail                ;# tell tester OS not to dump register

ifTrue                        ;# The first version uses TERM1
   iWrite SWING 800mv
   iWrite CMMV  Test_cm
   iWrite UPD      ON
   iWrite TERM1 Test
   iApply
ifFalse                       ;#  Version 2 the bits are swizzled
   iWrite SWING 600mv         ;# max swing with this rev
   iWrite CMMV  Test_cm
   iWrite UPD      ON
   iWrite TERM2 Test  ; # set the bits differently on this rev
   iApply
ifEnd


iWrite UPD     OFF  ;# prevents further updates
}
```

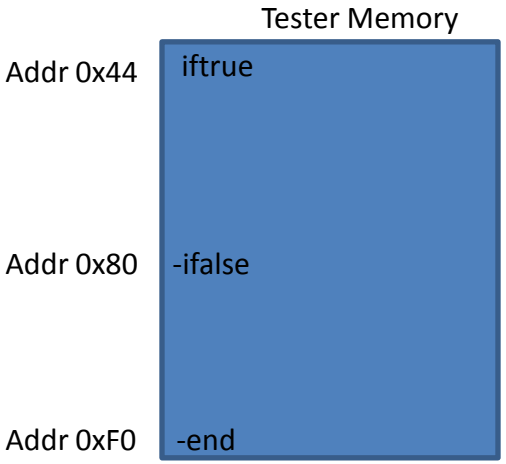
ifTrue/ifFalse - High-speed flow control
- need to set failure in some cases

```
iProc INIT_SETUP {} {
iRead Observe_IO_VSEL  0x13 ;# observe Freescale's strapping pins (5 bit value)
iApply -nofail

ifFalse             ;# catastrophic

iWrite myreg -safe  ;# example of setting value before failing/exiting

iApply
iSetFail  -quit    ; #all bets off, we need to tell tester to exit

ifTrue

iWrite xxxx
iApply
iRead  xxxx
iApply

ifEnd

}
```

ifTrue/ifFalse/ifEnd

- Record both ifTrue and ifFalse
- record command in tester format to "-nofail" on first iApply
- record iApply
- One option: Stop processing U1/U3 (there are some optimizations available)
  choices: load same frame data on U2's iApply and optionally
  clear expected data (XXX) on U1/U3 scan frame
  a) PULSE1/PULSE0 cells which take on 1 after 0 must be set
- record tester's binary command for branch-on-compare/miscompare
  (branch address set when else and end command encountered)
- record scan frames for If, record tester binary command for else compare
- record scan frames for else
- record/set –end location + 1 for branch operations





Record Scan Frames in Proprietary Tester Binary Format

Piece binary iApply together for U1/U3 for both ifTrue/ifFalse - nofail means no register dump

this will be dumped

```
iWrite data  0x01
iWrite WE   1 ;# PULSE1
iApply
iRead  Status 0xFF
iWrite data  0x02
iWrite WE   1
iApply
iRead  Status 0xFE
iWrite data  0x03
iWrite WE   1
iApply

...
```

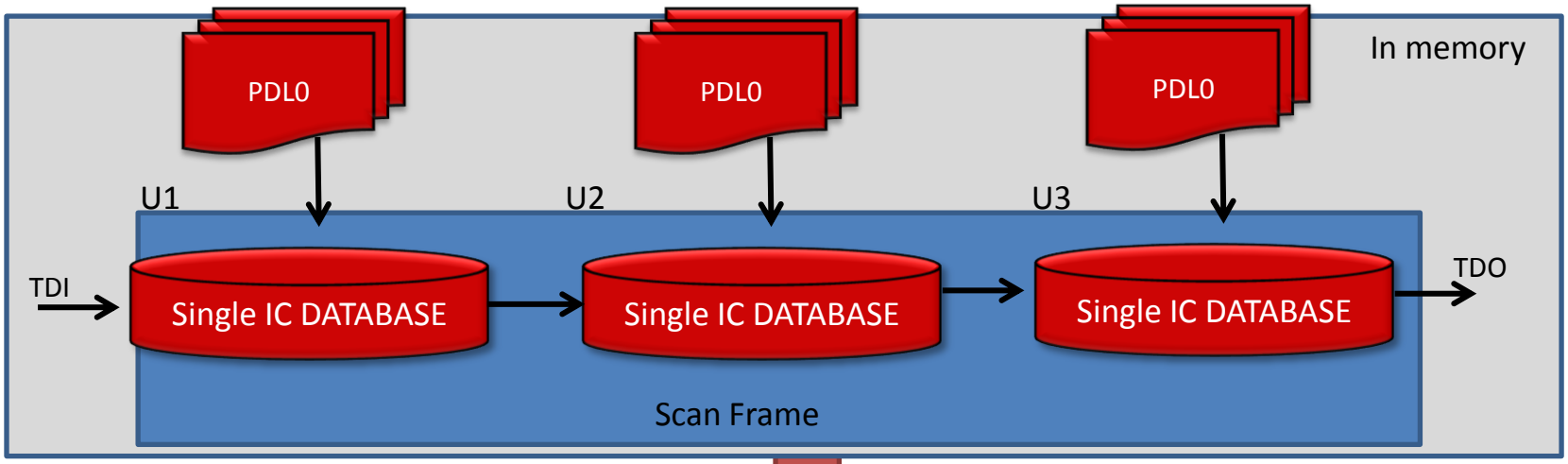
record iApply in other PDLs to use with 'else' if present

need same iApplys here + any additional

```
iRead VERSION 0b01   -nofail??
iRead  otherreg 0x02
iApply  -nofail???

ifTrue
   iWrite SWING  800mv
   iWrite CMMV  Test_cm
   iWrite UPD      ON
   iWrite TERM1 Test
   iApply
ifFalse
   iWrite SWING  600mv
   iWrite CMMV  Test_cm
   iWrite UPD      ON
   iWrite TERM2 Test
   iApply
ifEnd

iWrite UPD     OFF
iWrite SWING   -safe
iWrite TERM1    -safe
iWrite TERM2    -safe
```

one iApply

```
iWrite data  0x01
iWrite WE   1 ;# PULSE1
iApply
iRead  Status 0xFF
iApply
iWrite data  0x02
iWrite WE   1 ;# PULSE1
iApply
iRead  Status 0xFE
iApply
iWrite data  0x03
iWrite WE   1 ;# PULSE1
iApply
iRead  Status 0xFD
iApply
```



Record Scan Frames in Proprietary Tester Binary Format

Recorded test binary format for both the ifTrue and the ifFalse

- tester has to have command to branch on miscompare with addr
- tester needs Jump command at ifFalse to jump over ifFalse
- when iftrue/iffalse/ifend is non-symmetric, tool needs to align

			Data	Status	WE	VERSION	SWING	CMMV	UPD	Term1	Term2	Data	Status	WE
		1	0x01	XX	1	XX	safe	safe	OFF	safe	safe	0x01	XX	1
check	BNE 4	2	0x02	0xFF	1	0b01	safe	safe	OFF	safe	safe	0x01	0xFF	0
TRUE	JMP 5	3	0x03	0xFE	1	XX	800mv	test_cm	ON	test	(swizzled)	0x02	XX	1
FALSE		4	0x03	0xFE	1	XX	600mv	test_cm	ON	(swizzled)	test	0x02	XX	1
end		5	0x04	0xFD	1	XX	safe	test_cm	OFF	safe	safe	0x02	0XFE	0
		6	0x05	0xFC	1	XX	safe	test_cm	OFF	safe	safe	0x03	XX	1
		7	0x06	0xFB	1	XX	safe	test_cm	OFF	safe	safe	0x03	0XFD	0
		8	0x07	0XFA	1	XX	safe	test_cm	OFF	safe	safe	0x04	XX	1
		9	0x08	0XF9	1	XX	safe	test_cm	OFF	safe	safe	0x04	0XFC	0
		10	0x09	0xF8	1	XX	safe	test_cm	OFF	safe	safe	0x05	XX	1

User developed INIT_SETUP iProc for an instance U3
-No IP provided init_setup iProc
-Since init_setup/init_run are pre-defined possible to have them 'iExported' by default

```
iPDLLevel 0 –version STD_IEEE_1149_1_2012  ; # level-0 PDL only
iProcTarget U3   ; # Associate the following procs with the instance of U3

iExport          ; # indicate to user/tools these procs available

# this procedure becomes U3.init_setup internally to the PDL interpreter
iProc  init_setup {}  {
  iPrefix  i1
  iWrite  Clock       125Mhz          ; # use of BSDL mnemonics
  iWrite  Voltage     0x40            ; # use of hex values
  iWrite  Protocol    PCIe            ; # use of BSDL mnemonics
  iApply
}
# this procedure becomes U3.init_setup internally to the PDL interpreter
iProc  init_run {}  {
  iRunLoop 10000 ;                        # 10,000 TCK cycle delay
  iPrefix i1
  iRead   init_status(1)   Pass;  # use of single register bit
  iApply
}
#end of file
```

Other use models of 1149.1

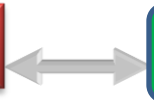
- 1149.1 is used in non-production segments of the industry

Interactive mode:

IC Characterization, Debug, lab bring up, system analysis, IC to IC SERDES testing, IC to DDR memory testing, Working With mixed signal devices (DACs and ADCs), voltage reading, temperature reading, read-write-modify registers.

This mode requires use of iGET on -SO (return) data.

Protocol Aware Testers



Interactive PC
Testers and analyzers
USB based JTAG pods > 50K units

1. Overview

1.1 Scope

This standard defines test logic that can be included in an integrated circuit to provide standardized approaches to

— testing the interconnections between integrated circuits once they have been assembled onto a printed circuit board or other substrate;

— testing the integrated circuit itself; and

— observing or modifying circuit activity during the component's normal operation.

What is PDL1?

PDL1 is these two commands (iGet and iGetStatus)
- note addition of -FAIL from Friday's meeting
+ All PDL0 commands
+ TCL (Tool Command Language)

TCL has been around since late 1990s. Used in nearly all EDA tools, both major FPGA vendors use TCL.

13175

Table C-3—PDL Level-1 Commands

Command	Parameters	Purpose
iGet	<register> [-IN | -OUT | -EXPECT | -FAIL] [-HEX | -BIN | -DEC | -MNEM]	Return a TCL string representing the value associated with a register in the specified format.
iGetStatus	[-clear]	Get the pass/fail status since the last time it was cleared.

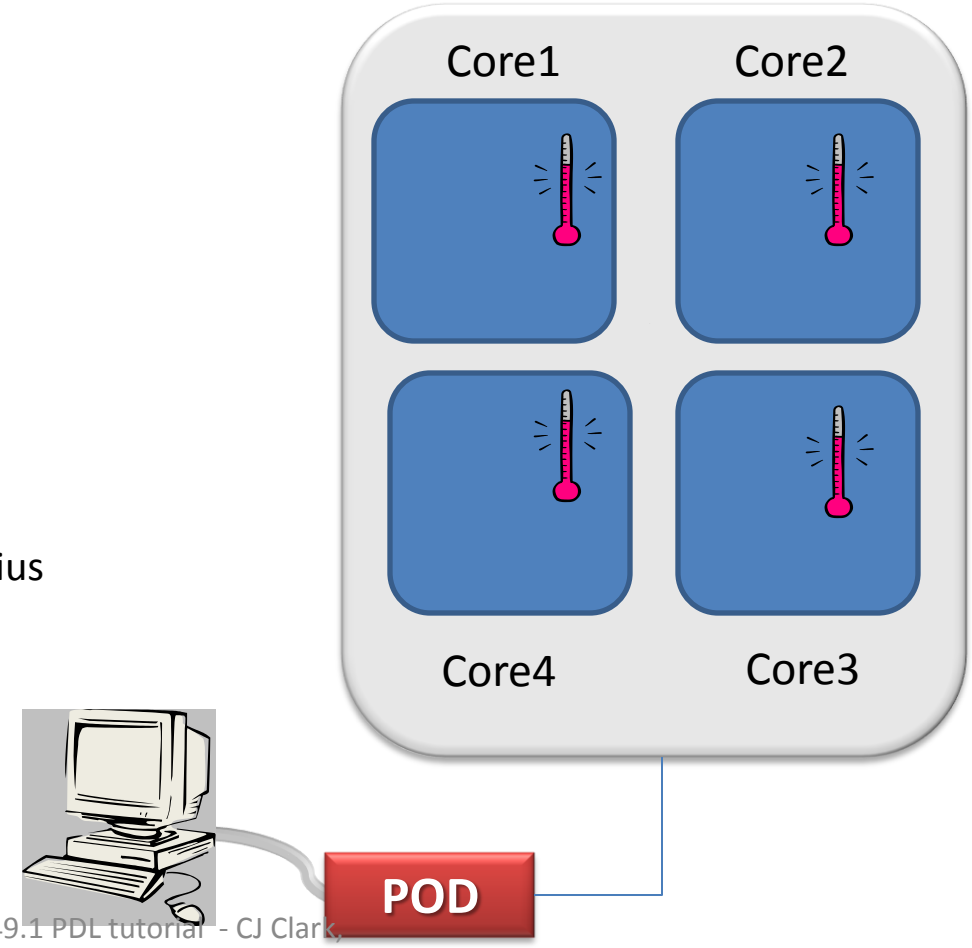
13176

Safe and cool - how do we know?

PDL0 only checks EXPECTED values. Not suitable for setting a range or < less than or > greater than as all those values are not the expected

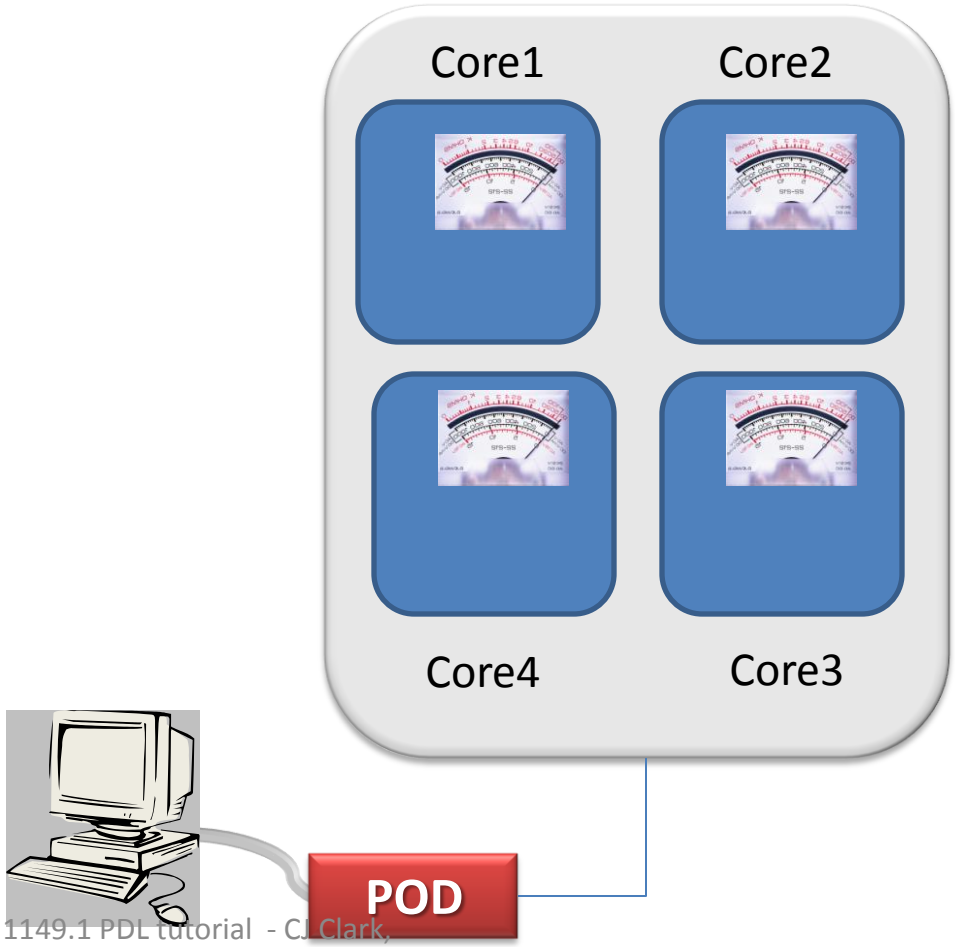
- can be used in 3D-SIC stacks as well.

```
# vendor supplied reg to temp conversion
proc  Reg2Temp { $regval $CorF } {
…
…
}

iExport -begin
# this proc returns a temperature  and
# high level warnings could be specified
iProc  init-setup-temp-check { } {

iRead tempreg
iApply
set val  [iGet tempreg]
# convert reg value to temperature in celsius
set  temp [Reg2Temp $val  CEL]
#if  {temp > 70}  {
#puts "Temperature is excessive $temp"
#}
return temp
}
```



AVS - automatic voltage scaling. Monitor mission mode operation during characterization. While CPU can access voltage monitors, if software is not prepared, 1149.1 is a convenient access mechanism to monitor AVS during bring-up, functional test (idea from Dharma)

```
# this proc returns a temperature  and
# high level warnings could be specified
proc  Reg2voltage {} {

}

iExport -begin
iProc  read-voltage{ } {
iRead  voltagereg
iApply
set val  [iGet  voltagreg]
# convert reg value to  voltage
set  volts [Reg2voltage $val]

return volts
}
```



Counterfeit parts continue to be a very well known problem

- ECID may not be just a 'value' to program in or to read out

recent letter from DoD contractors to suppliers

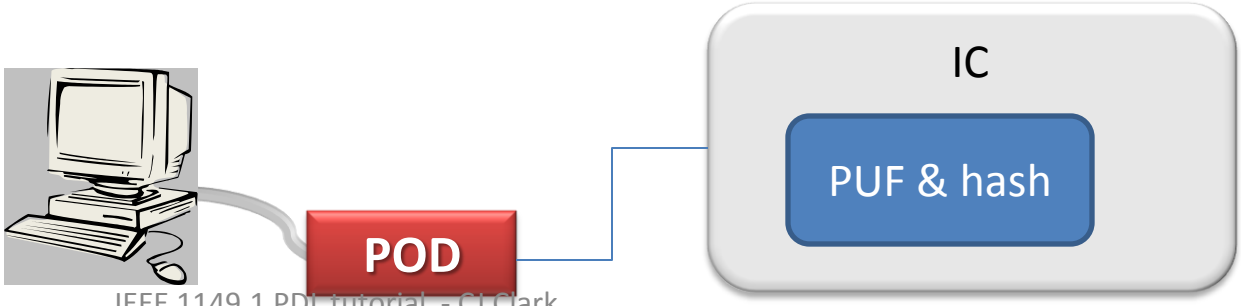
Subject: Global IC and Hong Dark Suspect Counterfeit Electronic Parts

The attached correspondence from the United States Air Force indicates that Hong Dark Electronic Trade Company a/k/a Hong Dark Electronic Co., Ltd. a/k/a Hong Dark Electronics Co., Ltd. a/k/a Hongdark Electronic Co., Ltd. a/k/a Hongdark Electronics Co., Ltd. a/k/a Shenzhen Hongdark Electronics Co., Ltd. a/k/a Hongdark Electronic a/k/a Hongdark Technology Co., Limited a/k/a Hongdark a/k/a Hong Xing Da Technology Co., Ltd. (collectively Hong Dark), Global IC Trading Group, Inc., and Global IC Trading Group, LLC (collectively Global IC) have been suspended from government contracting and from directly or indirectly receiving the benefits of federal assistance programs. In order for us to assess the impact of any potential counterfeit parts in our products,

1149.1 ECID can help!

ECID

PUF = physically unclonable function (layout dependent)
SHA256 = Secure Hash Algorithm



Counterfeit parts continue to be a very well known problem - ECID may not be just a 'value' to read out.


```
iPDLLevel 1 -version STD_IEEE_1149_1_2012
iProcGroup MNO_ECID_V1 ; # Associate the following procs MNO's ECID

# this is a SHA256 on the ECID read value with
iProc SHA256Calc { data } {

...   ;# too many details to show but needs PDL1
            ;# perhaps even external program call
}

# this procedure reads MNO company's ECID IP
iProc ECID {}   {}
iRead ECIDREADDONE 1
iRead ECIDPUF
iRead ECID_Hash
#iRead ECID_WaferNum
#iRead ECID_DieNum
#iRead ECID_ManuLocation
iApply

set data [iGet ECIDPUF]
set ECIDHash [iGet ECID_HASH]

set hashval [SHA256Calc $data]

if {$hashval != $ECIDHash} {
  puts "ERROR: Read Hash Does Not Match Expected - Exp: ECIDHash, Act:$hashval\n"
}

#this information may or may not get released
# they are just numbers, so without a decoder ring, the data is
# meaningless to end users
set WaferNum     [iGet ECID_WaferNum D]
set DieNum       [iGet ECID_DieNum D]
set ManuLocation [iGet ECID_ManuLocation]
}
```

IC vendor may not want to describe these registers (optional)

iProc allow additional routines from IC vendor for checking
- PDL1 used as a description language for constraints

```
iPDLLevel 1
iExport -begin
iProc check-values {} {
set val1 [iGet -IN -MNEM swing]     ; # 200mv, 400mv 800mv
set val2 [iGet -IN -MNEM  protocol]  ; # PCIe, SATA
if { $val1 == "200mv" && $val2 == "SRIO" } {
        puts "The I/O can not be set to 200mv in SRIO mode"
        return FALSE;   # instruct tool that check failed
}

}
```

1.1 Scope

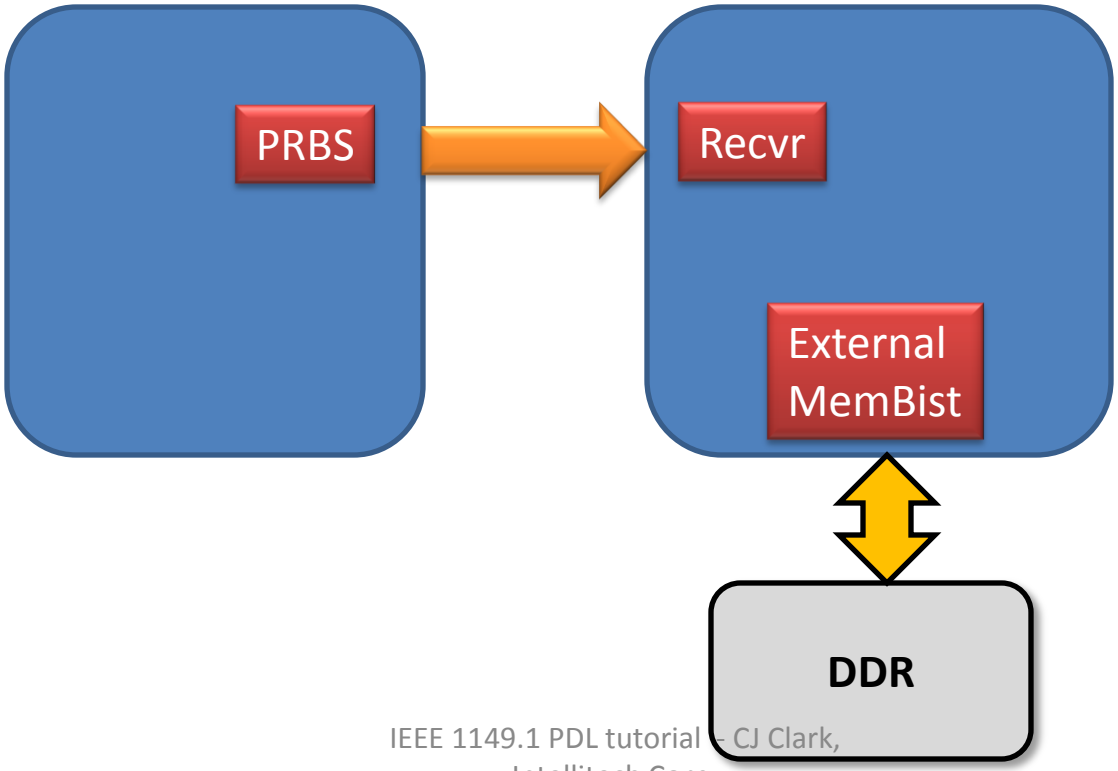
This standard defines test logic that can be included in an integrated circuit to provide standardized approaches to

- testing the interconnections between integrated circuits once they have been assembled onto a printed circuit board or other substrate;

Some tests cannot easily be described in PDL0

- some failing bits in SERDES test are tolerated

DDR memory tests, while slow speed tests can be done through EXTEST, less faults can be detected and test time can be unacceptable



PDL0 Diagnostics:

- Register: DATA expected 0x55 received 0x40

IP Provider

I have diagnostic information, how do I communicate that to my customer?

IC Provider

I have diagnostic information and checks, on-chip tests to exonerate my IC how do I communicate this?

Without a common language we have classic tower of Babel

IEEE 1801 extends TCL with commands for describing power intent
- very much in alignment with this proposal
-P1687 potentially will include some form of PDL1

6.5.2 errorInfo

See the *Tcl command reference* [B5].

6.6 add_domain_elements

Purpose	Add design elements to a power domain	
Syntax	**add_domain_elements** *domain_name* **-elements** *element_list*	
Arguments	*domain_name*	The power domain to modify.
	-elements *element_list*	The list of design elements to add. The elements shall be referenced relative to the active scope and are the descendents of the scope of the specified power domain.
Return value	Return a 1 if successful or raise a TCL_ERROR if not.	

The **add_domain_elements** command provides the ability to separate the creation of a power domain from the specification of the elements contained within it. This is similar to only specifying the elements using the **-elements** option in the **create_power_domain** command (see 6.19). The effect of **add_domain_elements** is additive, i.e., a power domain consists of any elements specified in the **create_power_domain** command and those elements specified in any **add_domain_elements** commands.

It shall be an error if *domain_name* does not indicate a previously created power domain.

This command is semantically equivalent to

```
proc add_domain_elements {dn elements el} {
   if { string equal $elements "-elements" }{
      create_power_domain $dn -update -elements $el
      return 1
   } else {
       return -code TCL_ERROR \
           -errorcode $ecode \
           -errorinfo $einfo \
           $resulttext
   }
}
```

where any *italicized* arguments are implementation-defined.

Possible syntax errors
did not prevent the adoption
of TCL in that standard
- TCL like BSDL (or verilog)
has to be verified

IEEE 1801's entire syntax reference points to TCL links in an informative annex bibliography

- precedence that we don't have to include all syntax in 1149.1, just the syntax of 1149.1 commands

Annex A

(informative)

Bibliography

[B1] IEEE 100, *The Authoritative Dictionary of IEEE Standards Terms,* Seventh Edition. New York: Institute of Electrical and Electronics Engineers, Inc.

[B2] IEEE Std 1364™, IEEE Standard for Verilog Hardware Description Language.[7]

[B3] ISO/IEC 8859-1, Information technology—8-bit single-byte coded graphic character sets—Part 1: Latin Alphabet No. 1.[8]

[B4] For a summary of Tcl language syntax, see the following Internet location: http://www.tcl.tk/man/tcl8.4/TclCmd.

[B5] For more details on using the Tcl language, see the following Internet location: http://sourceforge.net/projects/tcl/.

[B6] For more details on using the Liberty library format, see the following Internet location: http://synopsys.com/cgi-bin/tapin/login1.cgi.

[B7] Coding examples are available from the UPF *WG* World Wide Web site http://www.accellera.org/upf/references.html.

iGet command is liked so much it was "brought" to P1687 before 1149.1 has voted

-----Original Message-----
From: Ted (Theodore) Eaton-SSI [mailto:t.eaton@ssi.samsung.com]
Sent: Wednesday, February 08, 2012 10:37 PM
To: Rearick, Jeff; hugh_wallace@agilent.com; acrouch@asset-intertech.com; bill.bruce@siliconaid.com; Brian Turmelle; hojun@cisco.com; Doege, Jason; jeffrey_wilkerson@non.agilent.com; JF_Cote@mentor.com; Jpotter@asset-intertech.com; Martin_Keim@mentor.com; mcoldewey@asset-intertech.com; mlaisne@QUALCOMM.COM; szuo@QUALCOMM.COM; teaton@ieee.org; CJ Clark; carl.barnhart@SILICONAID.COM
Cc: ken.posse@avagotech.com
Subject: RE: Draft Chapter 8
All,
Here is the iGet command definition from the current 1149.1 draft for consideration.
iGet
<register> [IN | OUT | EXPECT] [HEX | BIN | DEC | MNEM]
Return a TCL string representing the value associated with a register in the specified radix.

_____

My Thoughts :
1. The default behavior of the PDL1 commands seems to be targeted for a specific tester type. From my standpoint, most environments and PDL will be most interested in seeing the results of the previous iApply rather than a history of the last N iApply operations. It seems to me that it would be better for the iGet<*> commands to default to :
a. Capture Active (the data of the last operation is available without the use of iCaptureData).
a. iGetReadValues returns the result of the last iApply for a register. (does these need to be the last iRead-iApply sequence, not sure).
b. iGetWriteValues returns that value of the last iWrite command to that register
c. iGetExpectValues returns the value of the last iRead command to that register

i.

If the last iRead did not have an expected value an X value is returned

ii.

If an iApply has been performed after the iRead, what is returned . Expect data is not sticky so I would assume the current state of the registers expect data is X
b. Capture depth is 1 by default.
2. The Radix of the return value is not defined here. It seems that we should have some switch/parameter available to allow the user to select a Radix (BIN/HEX/mnemonic/INT?)
3. 1149.1 has a similar process defined that seems more compact and may be a good place that we can consolidate (I will provide the exact command specification when I get access)
4. We can leave the iCaptureData function available for memory behind pin or other testers that can make use of a pattern buffer history, but the default behavior would be defined as above.

Register_fields are just pointers to bits. iWrite mysinglebit 0

	iWrite (in)	iRead (expected)	TDO (OUT)
init-data	0x54	0x13	XX

Mnemonics of course can be used as well

	iWrite (in)	iRead (expected)	TDO (OUT)
init-data	PCIe	Pass	XX

iGet returns data from each of the three fields.

```
set val [iGet -IN -MNEM init-data]
set val2 [iGet -EXP -MNEM init-data]
set val3 [iGet -OUT -HEX init-data]
set val4 [iGet -FAIL -BIN  init-data ]
puts "$val  $val2  $val3 $val4"
            - output:  PCIe  Pass  0xXX  0bXXXXXXXX
```

These operate on database only

This type of operation would require access To UUT data

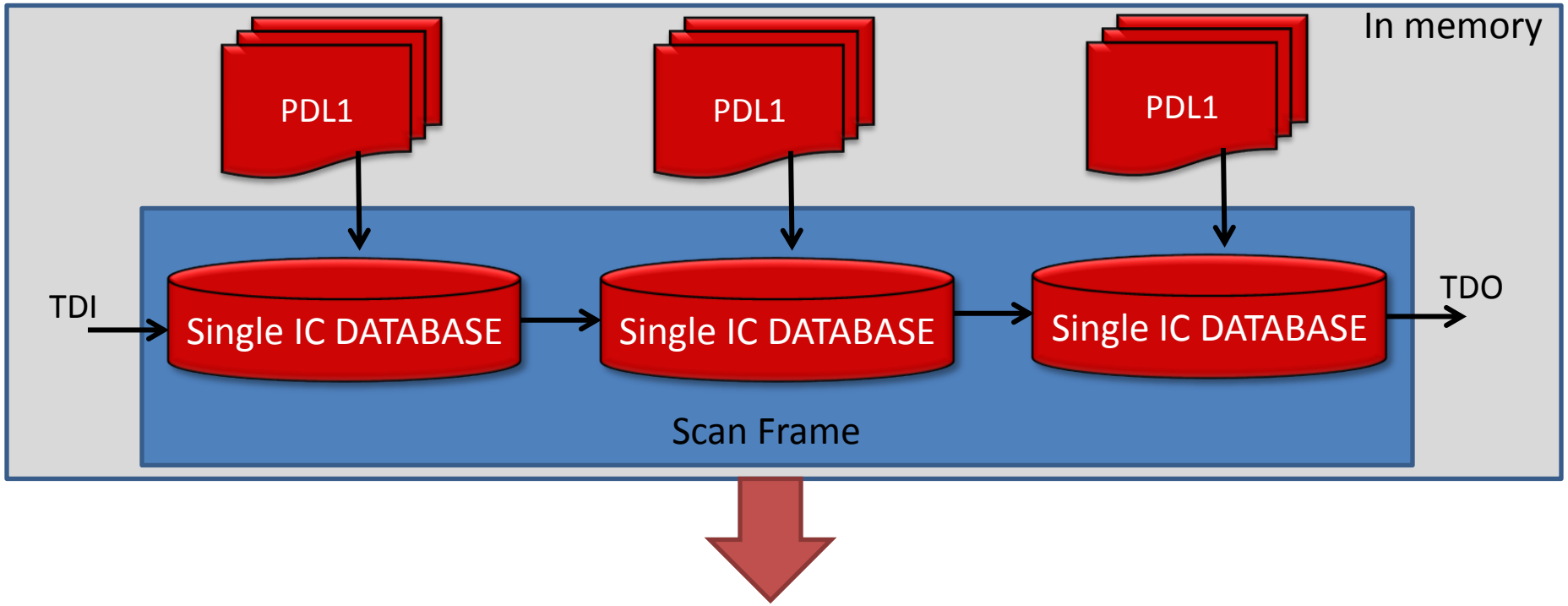
For discussion: PDL1 ends up not being much different

- Consider iGet -OUT <regname> as non-merge-able
- iGet -IN and -EXP data *is* merge-able

-Requires use of TCL interpreter

-Instead of private compiler of PDL0

-math, expressions/branching all resolve into an iApply on a populated scan frame



For consideration: Constraints can be described directly in PDL1

```
# default value is 'off' for param check

iProc  Constraints {  { check OFF } }  {

if { $check == ON } {

set val1 [iGet -IN -MNEM DOMSELA]     ; # DOM A ON
set val2 [iGet -IN -MNEM DOMSELB]     ; # DOM B ON

if { $val1 == "ON" && $val2 == "ON" } {
         puts "ERROR Domain A cannot be turned on when Domain B is on"
         iWrite DOMSELA OFF
         return FALSE  ;#  instruct tool init_setup failed
}
}

return TRUE;
}
```

It does appear 'sequential' or 'executable' but the approach is essentially a More robust form of the BSDL constraint attribute. Without the need for The WG to create a language inside of BSDL

Describing constraints/conditions

One approach which has been used in the past

attribute REGISTER_CONSTRAINTS of mychip : entity is
"(DOMAINA == ON && DOMAINB == ON)";

Need a language for operators: X % ! + - /
And order precedence ( )
- note we have just two registers, what about 3 or 4 or 10?

What about?

attribute REGISTER_CONSTRAINTS of mychip : entity is
"(REGA + 1 && REGB)"; ;# math required

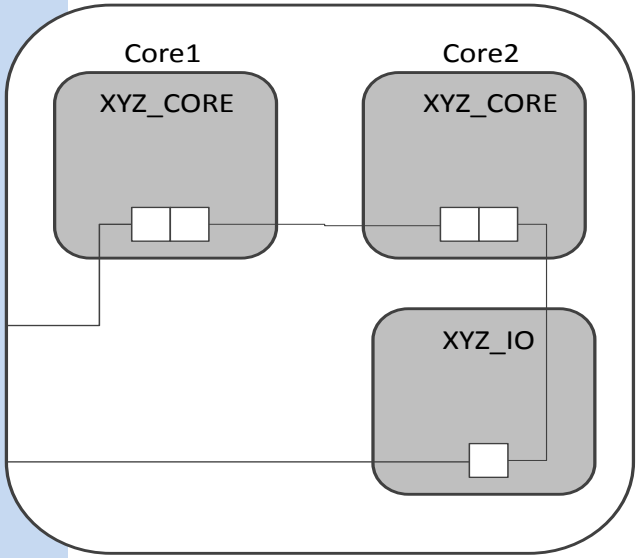
Sequential constraints REGA can't be a 1 after being a 0 and REGB is a 0? BSDL is a difficult place to describe these relationships

What happens when constraints not met? What is the error message?

Merging with iMerge

Merging – used for reducing scan operations

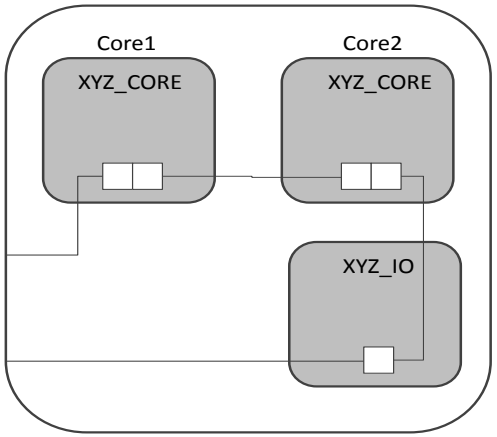
```
 PLL
# provided by IP provider, ideally it would be
beneficial to have these optimized for INIT_SETUP
iPDLLevel 0 -version IEEE1149_1_2012
iProcTarget XYZ_CORE

iProc setpll { val } {
# PLL-WE is 1 bit and PLLREG is 1 bit
iWrite PLL-WE  0
iWrite PLLREG $val
iApply
iWrite PLL-WE  1
iApply
}
#end file
```




```
iPDLLevel 0 -version IEEE1149_1_2012  ; # level-0 PDL only
iProcTarget XYZ_IO
iProc init_setup { val } {
iWrite AC-MODE $val
iApply
}
#end file
```

iMerge

Discussion: Are iCalls merged automatically in init_setup or init_run?

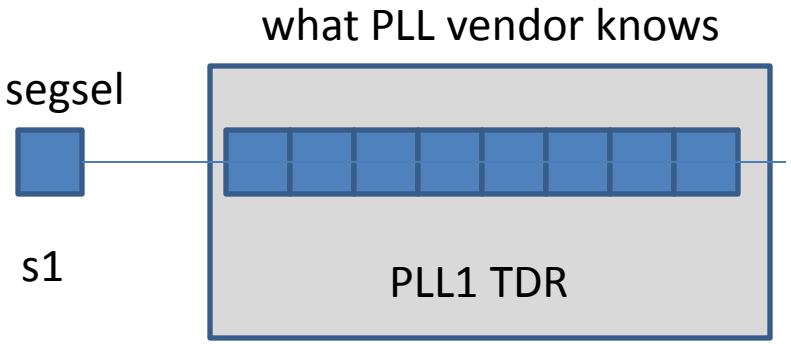



```
iPDLLevel 0 -version IEEE1149_1_2012
iSource XYZ_CORE.PDL
iSource XYZ_IO.PDL
iProcTarget XYZOxygen

iProc init_setup { } {
iMerge -begin
iCall U1.Core1.setpll OFF  ;# XYZ_CORE  two   iApply
iCall U1.Core2.setpll OFF  ;# XYZ_CORE  two   iApply
iCall U1.i1.init_setup OFF ;# XYZ_IO    one   iApply
iMerge -end
}
#end of file
```

reduces to two scan operations

	U1.Core1.PLLREG	U1.Core1.PLL-WE	U1.Core2.PLLREG	U1.Core2.PLL-WE	iWrite U1.i1.AC-MODE
iApply	OFF	0	OFF	0	OFF
iApply	OFF	1	OFF	1	OFF

PLL vendor does not know where PLL is in final IC



This fails

IC level init_setup
iProc init_setup {} {
iCall Pll1.init_setup
iCall IO.init_setup
}

Like board level Scan Path Linkers, tool is responsible for opening SEGSELs, turning on DOMCTRL

Checking for OO on Power pins and SEGSEL captures are catastrophic events which a tool can know during tester binary format recording and insert proper tester command

```
# IP
iProc init_setup {} {
iWrite PLL OFF
iApply
}
```

This adds to work for end user:

```
iWrite  s1 ON
iApply
iCall PLL1.init_setup
```

Tool manages access to TDR
Potentially in one iApply this is done
iApply is :

IR scan if required
DR to turn on Domains (if any)
observe DOM_EXT SEGSELs
DR - to open/capture power
on internal power domains
DR - to PLL