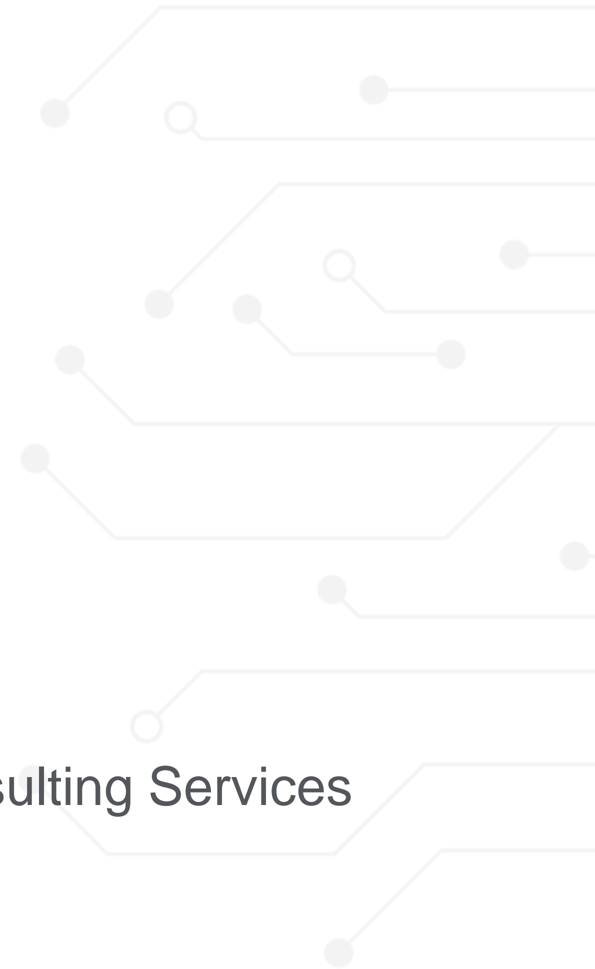
# Exploring and Comparing IEEE P1687.1 and IEEE 1687 Modeling of Non-TAP Interfaces

Special Session SP1

*Jeff Rearick* *Martin Keim* *Michele Portolan* *Brad Van Treuren* *Hans Martin von Staudt*

# Outline of the Special Session

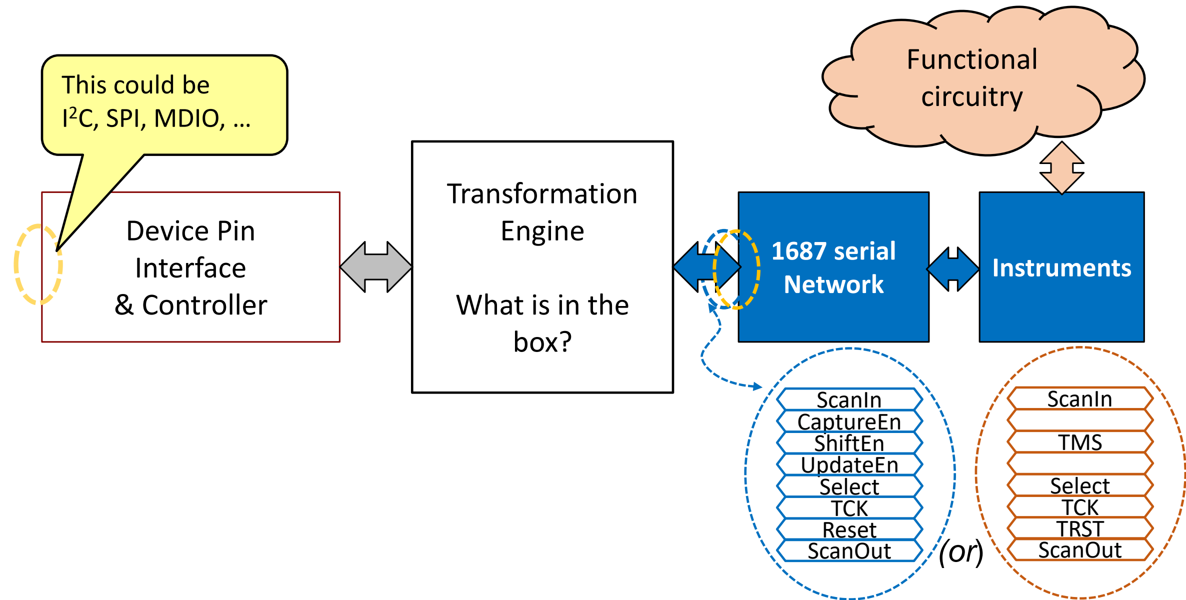
- Introduction of IEEE P1687.1
  - Martin Keim, Siemens Digital Industries Software
- I²C modeling with IEEE 1687-2014
  - Hans Martin von Staudt, Dialog Semiconductor
- Modeling non-TAP interfaces with IEEE P1687.1
  - Jeff Rearick, Advanced Micro Devices,
    With contributions from Bradford Van Treuren, VT Enterprises Consulting Services
- A working example of IEEE P1687.1 callbacks
  - Michele Portolan, Univ Grenoble Alpes CNRS



# Introduction of IEEE P1687.1

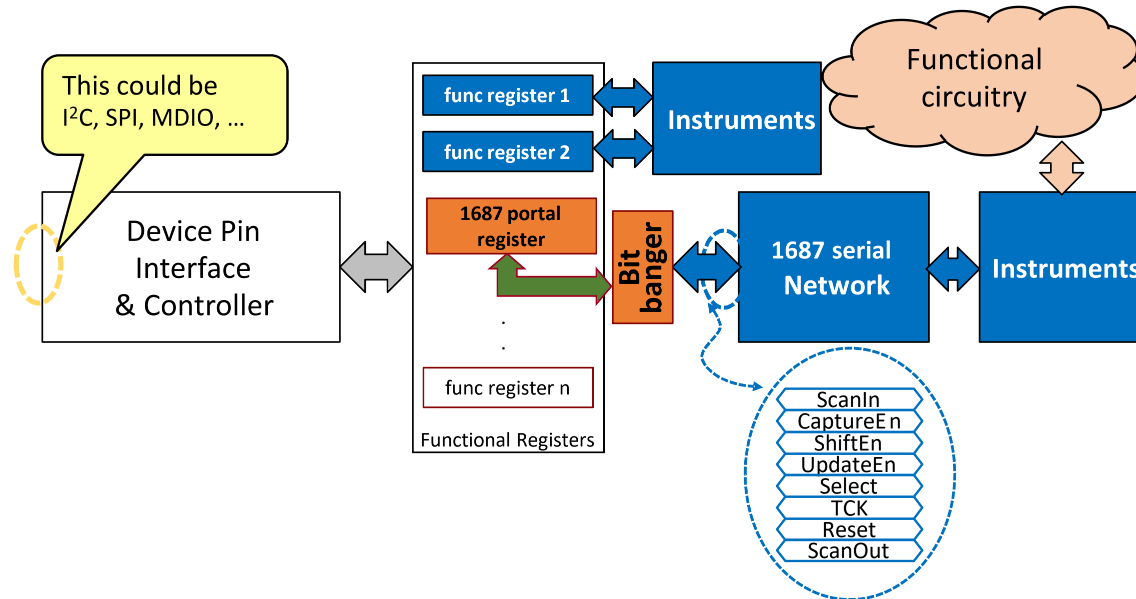
Martin Keim, Siemens Digital Industries Software

# Goal of IEEE P1687.1



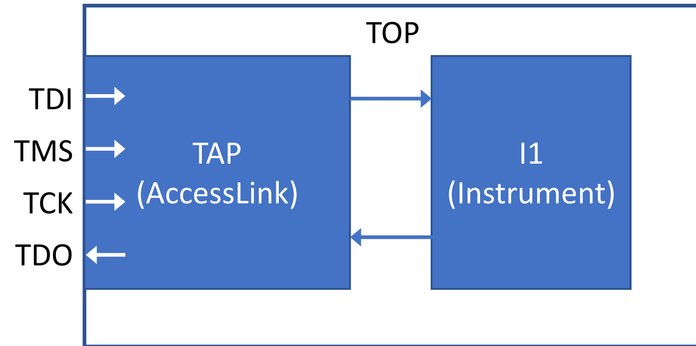

- Access & operation of an IEEE 1687 network through a non-Tap interface
- Like IEEE 1687: Descriptive, not Prescriptive
- How to describe the "Transformation Engine"?

# Early Ideas of IEEE P1687.1

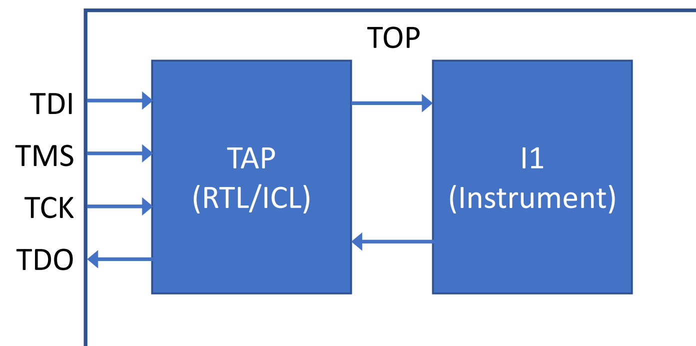



- Observation: Many interfaces have an actual or virtual 'portal register'
- Idea of expanding IEEE 1687's
  - Register callback
  - AccessLink
- Issues
  - Register call back difficult to implement
  - Bit banging

# Concerning IEEE 1687's AccessLink

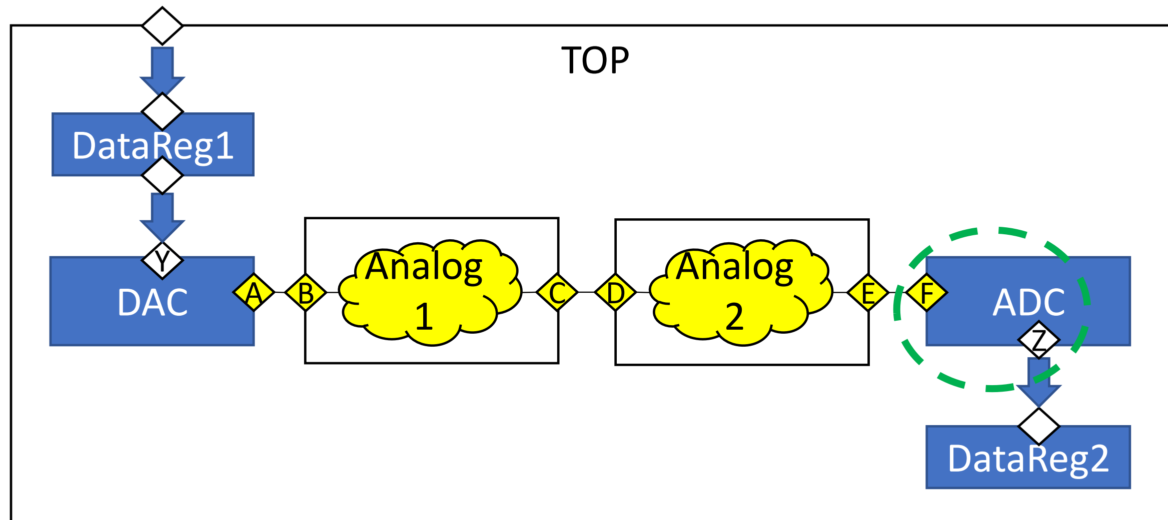



```
Module TOP {
 Instance I1 of MyInstrument { }
 AccessLink TAP of STD_1149_1_2001 {
  BSDLEntity TOP ;
  my_ijtag_en { // instruction name
   ScanInterface { I1.scan_client; }
  }
 }
}
```



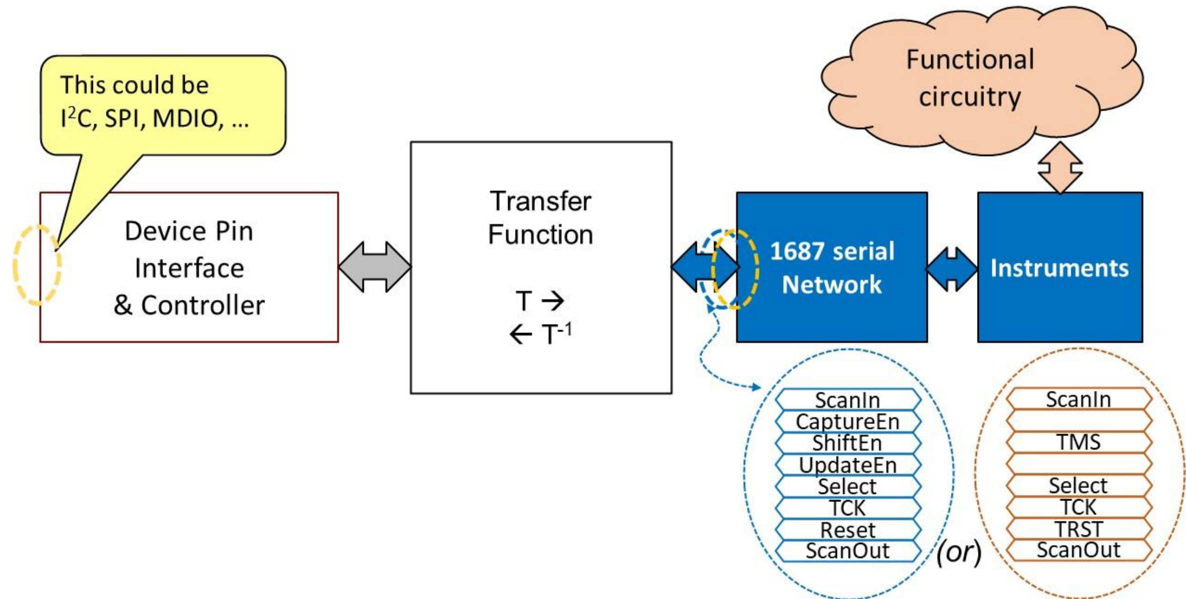

- 1687 describes 2 AccessLink types
  - For type 1149.1 (shown here)
  - Generic type (practically unusable)
- With AccessLink one can attach a protocol to an ICL instance
  - Without describing much of the body of the ICL module
- Observation
  - Loss of mapping between ICL and RTL
  - AccessLink incorporates DPIC

# IEEE P2654 and IEEE P1687.2 Crossovers




- In short
  - P2654 = scale up to board level
  - P1687.2 = incorporate analog elements
  - More details later
- Transformation Engine not at the device IO
- One transformation engine feeds into another
- Still, the same problem!
- Can we find one (1) answer for all?

# Concept of Transfer Function




- Transfer Function ≠ Call back
- A mathematical object
  - Models the device's output for each possible input
  - Models the entire module
  - No module internals needed
- Examples
  - 1149.1 TAP
    - ICL for comb. logic + FSM instance
    - FSM describable as a transfer function
  - 1149.7
    - Number of input and output cycles not necessarily the same

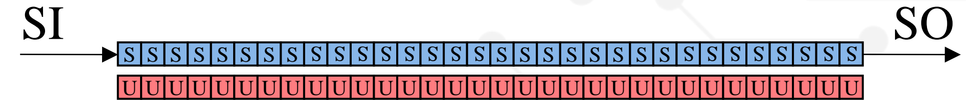
# $I^2C$ Modeling with IEEE 1687-2014

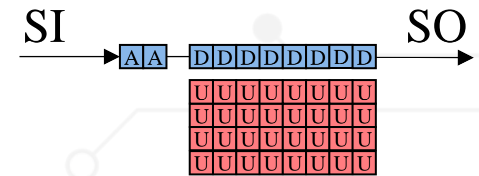
Hans Martin von Staudt, Dialog Semiconductor

# Why I²C/SPI/etc. Again and Again?

- Ubiquitous. That leads to the killer argument:
  - **If you have I²C already for mission mode, why bother about a TAP?**

- Inherently cheaper than a scan architecture
  - Scan
    - Locates a bit by position on a chain (which might reconfigurable)
    - Each data bit needs a shift bit: 100% overhead. Total bit count: 2n
  - I²C
    - Locates a bit by position in a word, which is located in an address map
    - Overhead grows logarithmically.  Total bit count → $n + w + \log_2(n/w)$

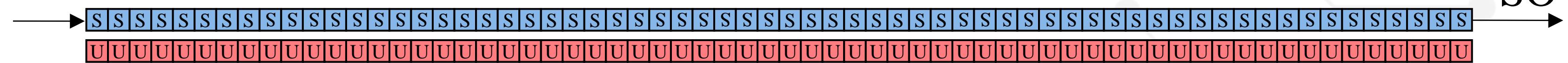


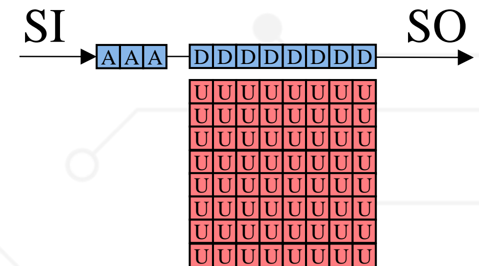
# Why I²C/SPI/etc. Again and Again?

- Ubiquitous. That leads to the killer argument:
  - **If you have I²C already for mission mode, why bother about a TAP?**

- Inherently cheaper than a scan architecture
  - Scan




    - Locates a bit by position on a chain (which might reconfigurable)
    - Each data bit needs a shift bit: 100% overhead. Total bit count: 2n
  - I²C



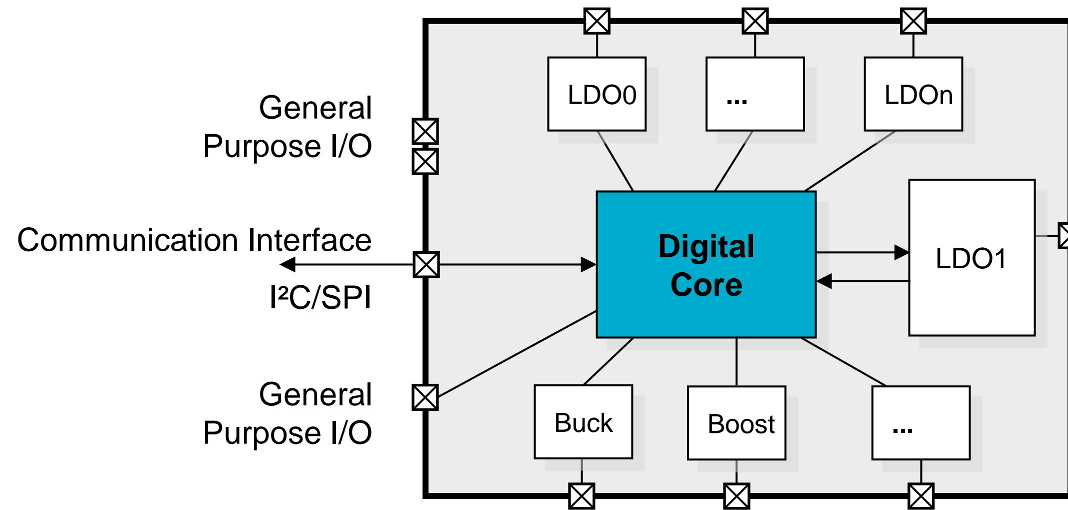

    - Locates a bit by position in a word, which is located in an address map
    - Overhead grows logarithmically. Total bit count → $n + w + \log_2(n/w)$

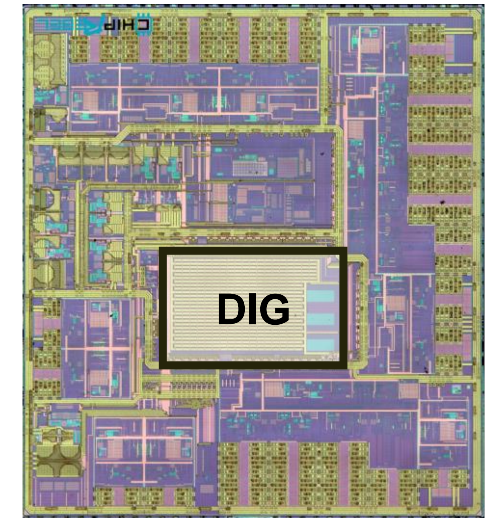
- Simpler to use
  - Computers work on words in an address map
  - Scan needs dedicated software

**How to reconcile both concepts?**

# Industrial Example: Power Management IC

- Simple architecture?
  - Big-A / little-d




- Many power management functions, multiple variants. Up to 20 LDOs, 10 switching converters, charge pumps, rail switches, charger, etc.
- Integration different every time

H. M. v. Staudt, M. A. Benhebibi, J. Rearick and M. Laisne, "Industrial Application of IJTAG Standards to the Test of Big-A/little-d devices," IEEE International Test Conference (ITC), 2020

# How to reconcile both concepts?

Two examples from the literature

A. Design IJTAG Scan Network to Match $I^2C$ transaction concept

B. Describe $I^2C$ Registers as IJTAG Callback DataRegister

# How to reconcile both concepts?

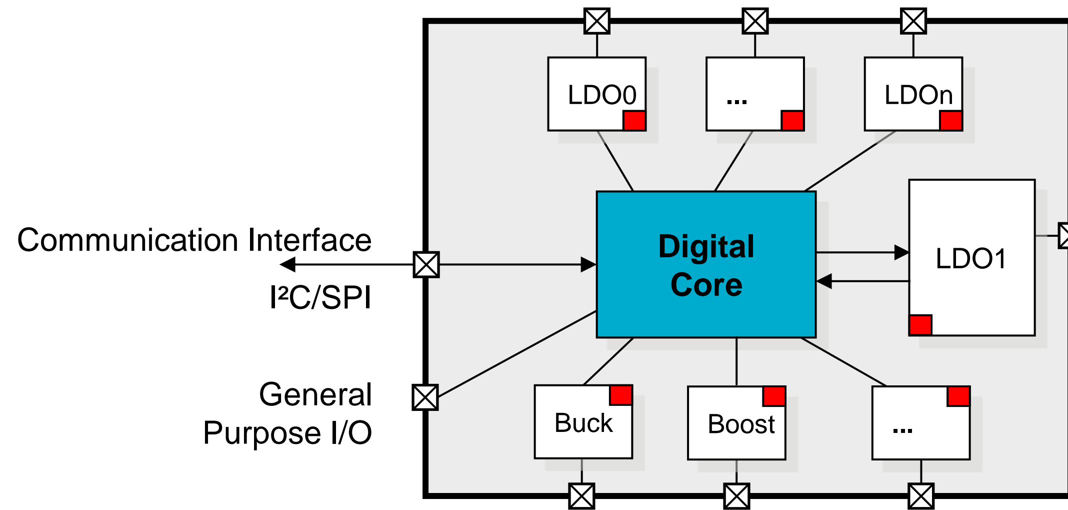
Two examples from the literature

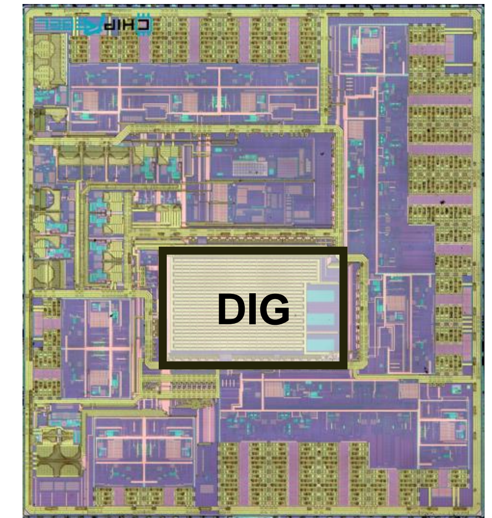
A. Design IJTAG Scan Network to Match $I^2C$ transaction concept

B. Describe $I^2C$ Registers as IJTAG Callback DataRegister

# Industrial Example: Power Management IC

- Equip every analog IP with an IJTAG 1687 trim & test island



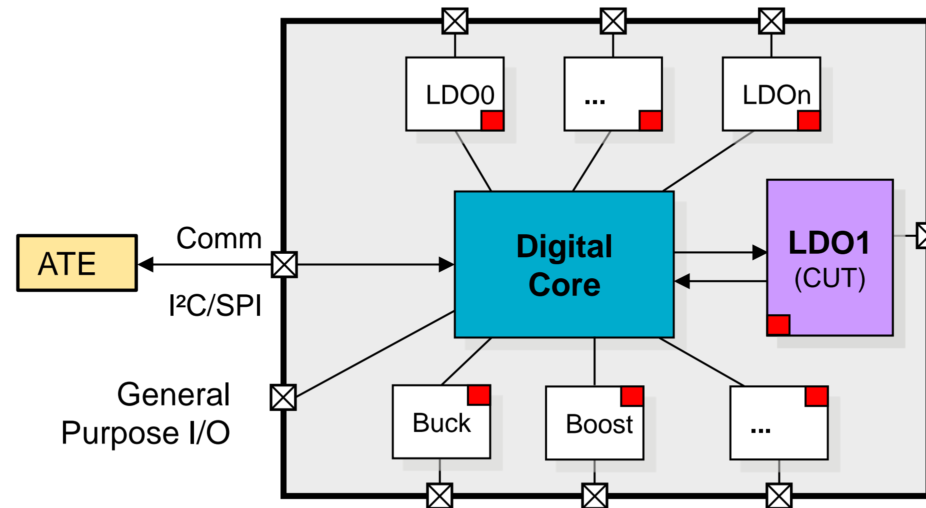



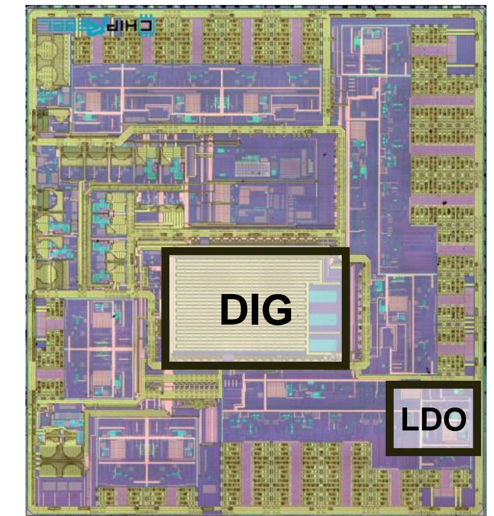

- Connect 1687 islands to the digital core. External communication via I²C

H. M. v. Staudt, M. A. Benhebibi, J. Rearick and M. Laisne, "Industrial Application of IJTAG Standards to the Test of Big-A/little-d devices," IEEE International Test Conference (ITC), 2020

# Industrial Example: Power Management IC

- Equip every analog IP with an IJTAG 1687 trim & test island



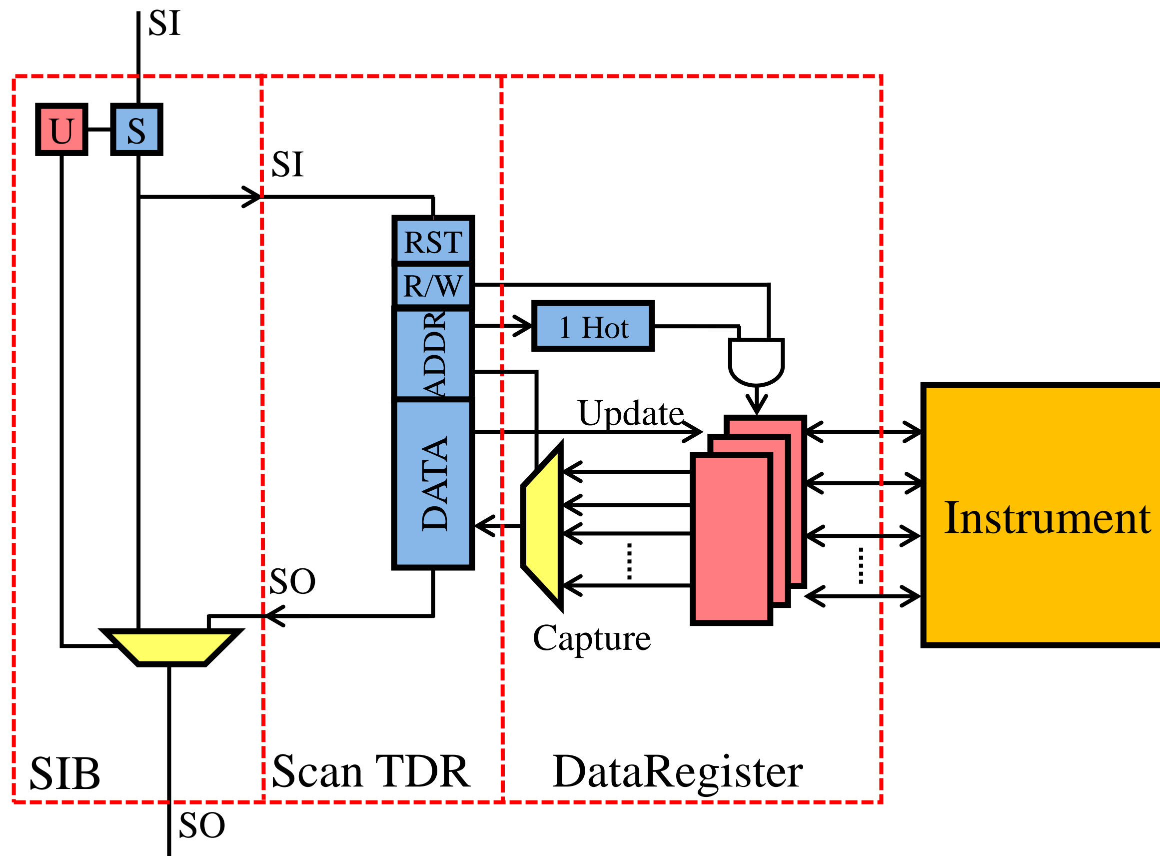




- Connect 1687 islands to the digital core. External communication via I²C
- PDL written on CUT level: LDO1. Retarget to chip top level → ATE

H. M. v. Staudt, M. A. Benhebibi, J. Rearick and M. Laisne, "Industrial Application of IJTAG Standards to the Test of Big-A/little-d devices," IEEE International Test Conference (ITC), 2020

# A) Design IJTAG Scan Network to Match I²C

## IJTAG Test and Trim Islands

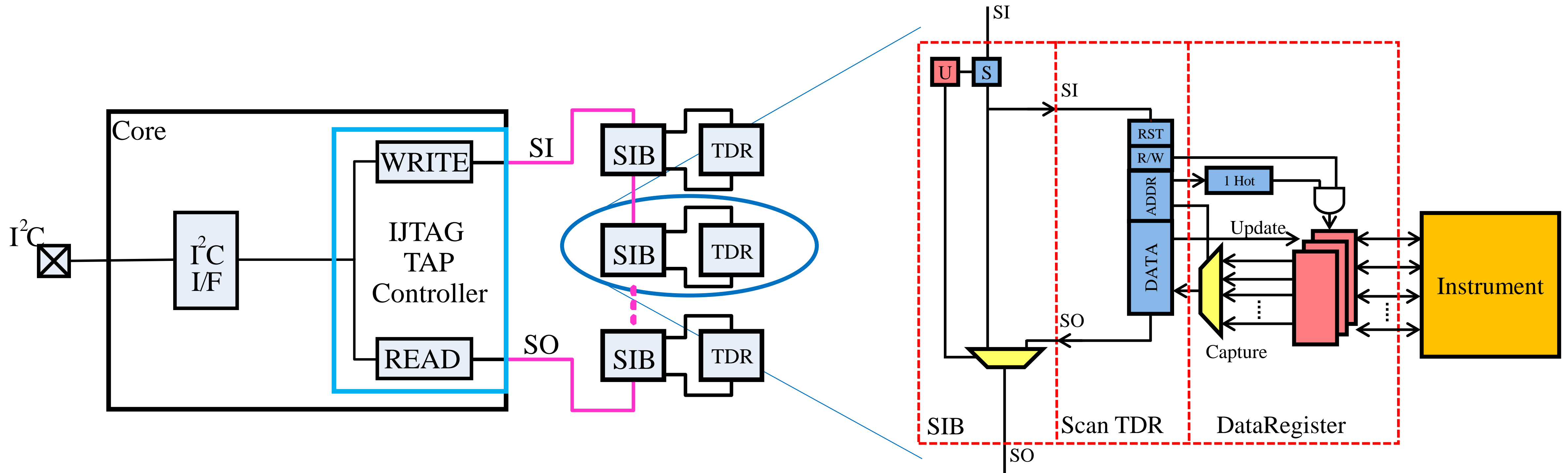



- Use <u>addressed</u> DataRegister As per 1687-2014
- Implementation:
  - SIB
  - Scan TDR
    - 1-bit Read/Write
    - 4-bit address
    - 8-bit data
  - Up to 16 bytes of DataRegister
- Scan TDR design replicates I²C transaction structure

H. M. von Staudt and A. Spyronasios, "Integration of IJTAG Test and Trim Islands in I²C Legacy Designs", International Mixed-Signals Testing Workshop (IMSTW)2015

# A) Design IJTAG Scan Network to Match I²C

## State machine TAP operating chain of identical test and trim islands



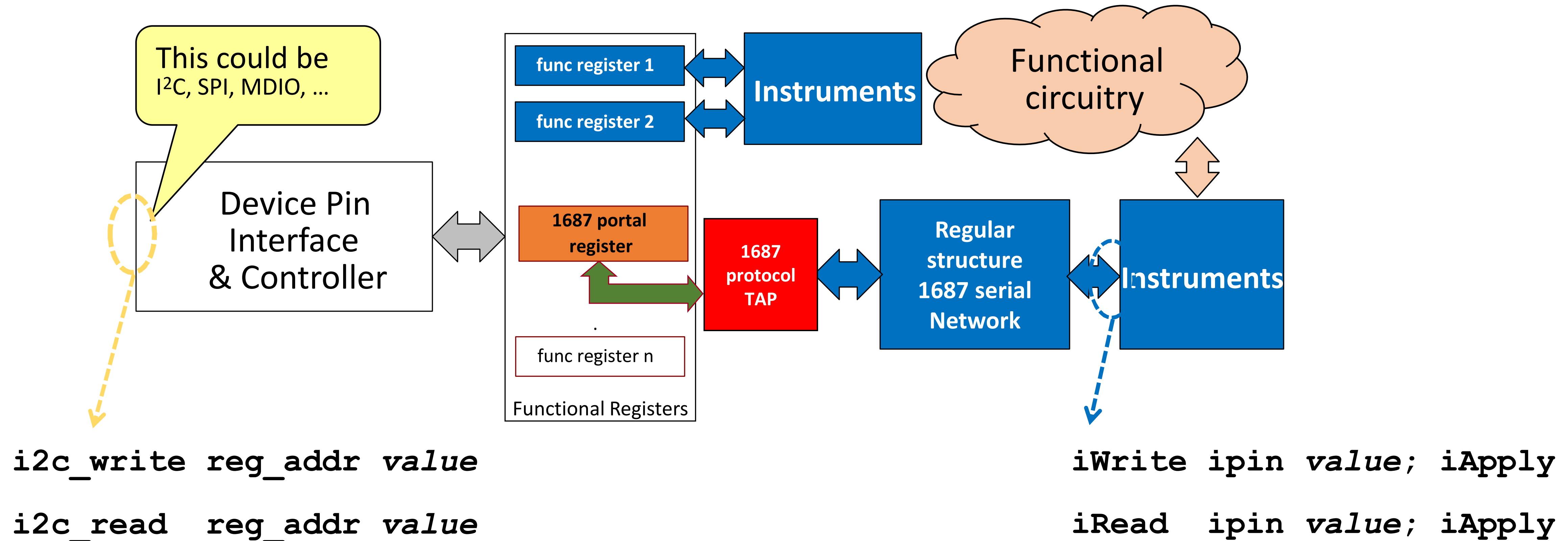

```
i2c_write reg_addr value
i2c_read  reg_addr value
```

```
iWrite ipin value; iApply
iRead  ipin value; iApply
```

"Integration of IJTAG Test and Trim Islands in I²C Legacy Designs", 2015 IEEE 20th International Mixed-Signals Testing Workshop (IMSTW), Paris, 2015

# A) Design IJTAG Scan Network to Match I²C

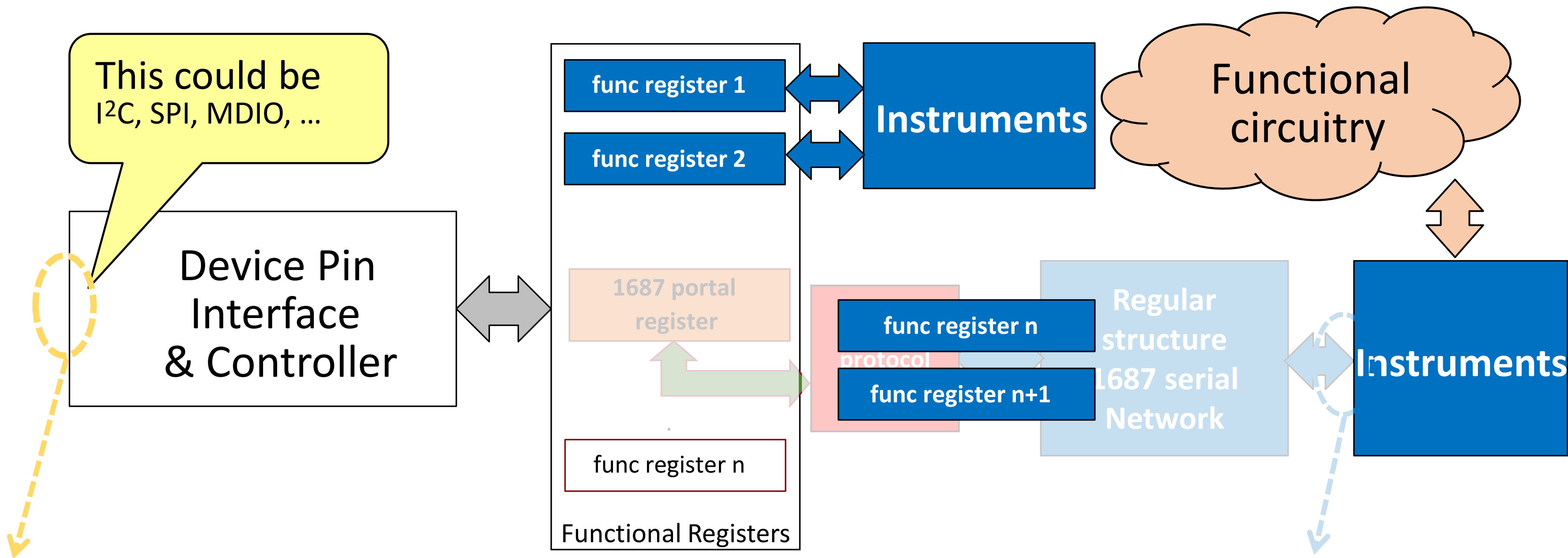
1:1 mapping of iWrite/iRead to I²C transaction



# A) Design IJTAG Scan Network to Look Like I²C

IJTAG hidden from the user.

Then, what‘s the point? Grow 1687 enabled test infrastructure!




```
i2c_write reg_addr value
i2c_read  reg_addr value
```

```
iWrite ipin value; iApply
iRead  ipin value; iApply
```

# How to reconcile both concepts?

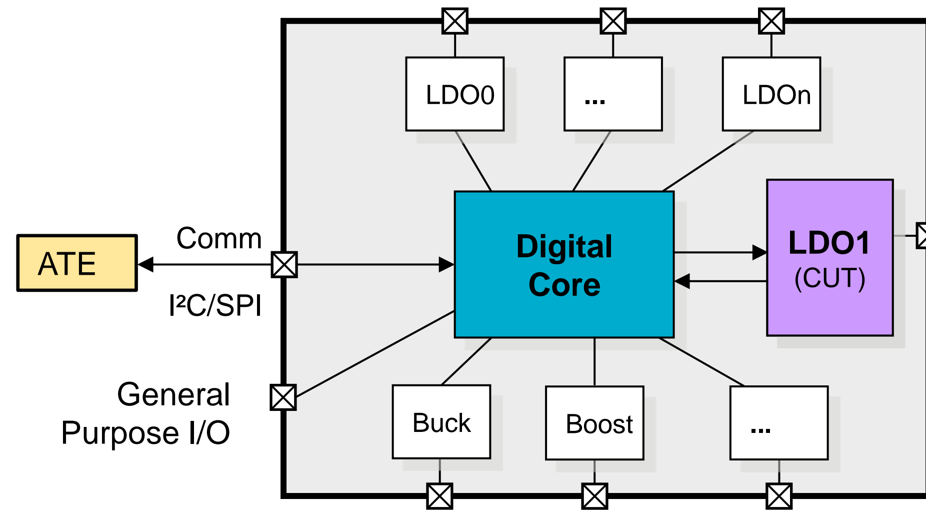
Two examples from the literature

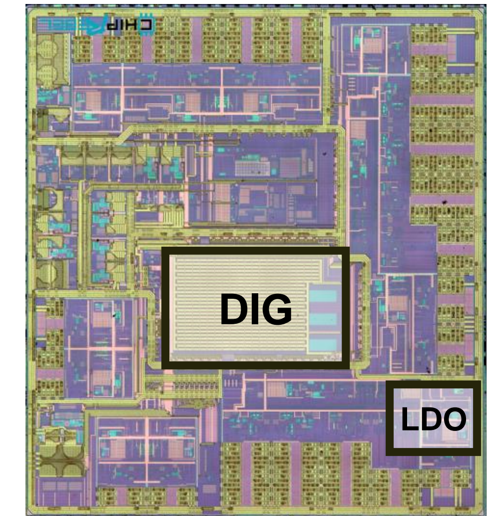
A. Design IJTAG Scan Network to Match I²C transaction concept

B. Describe I²C Registers as IJTAG Callback DataRegister

# Industrial Example: Power Managment IC

- PDL written on CUT level: LDO1



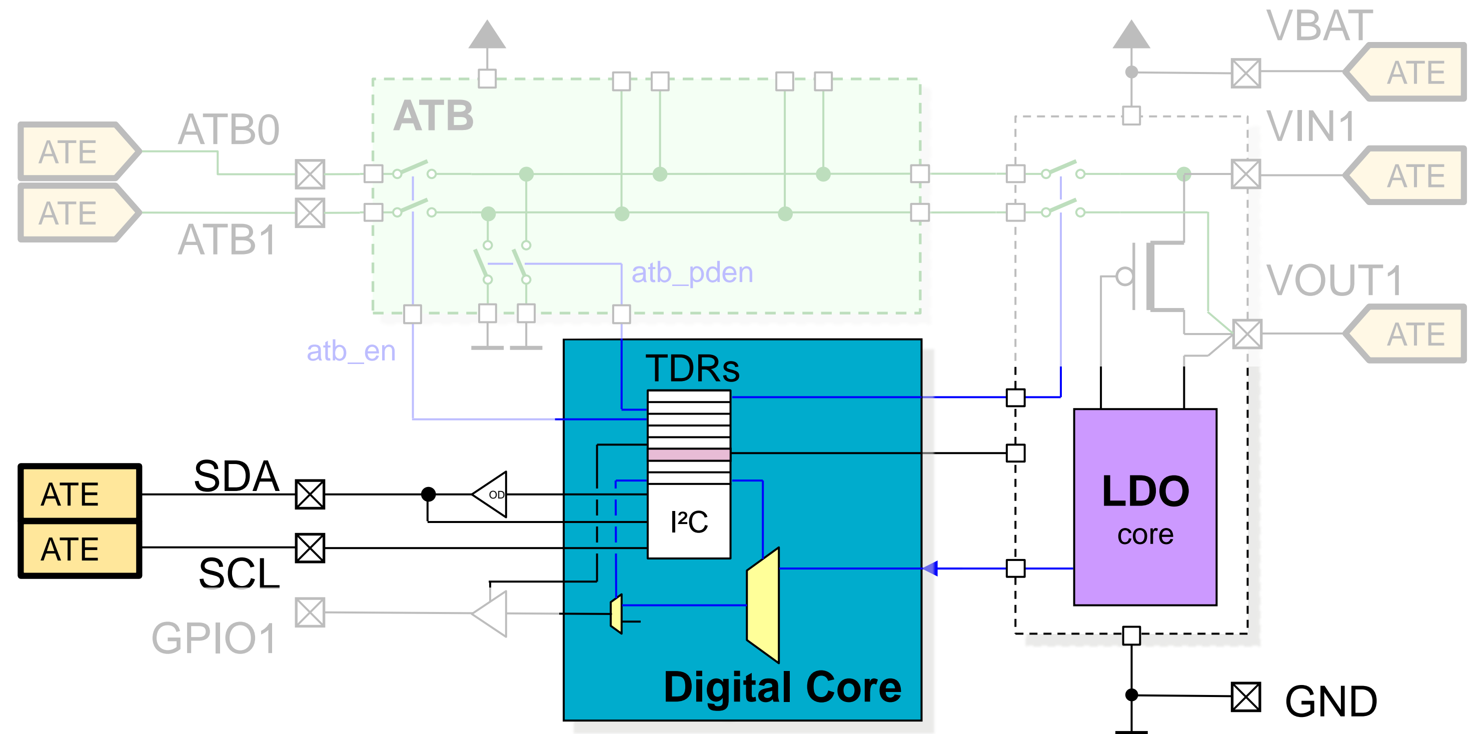




- No test and trim island.
- Control register in digital core, accessible via I²C

H. M. v. Staudt, M. A. Benhebibi, J. Rearick and M. Laisne, "Industrial Application of IJTAG Standards to the Test of Big-A/little-d devices," IEEE International Test Conference (ITC), 2020

# Zoom In on CUT and Digital Core

Ignore all analog to stay with the focus of this Special Session



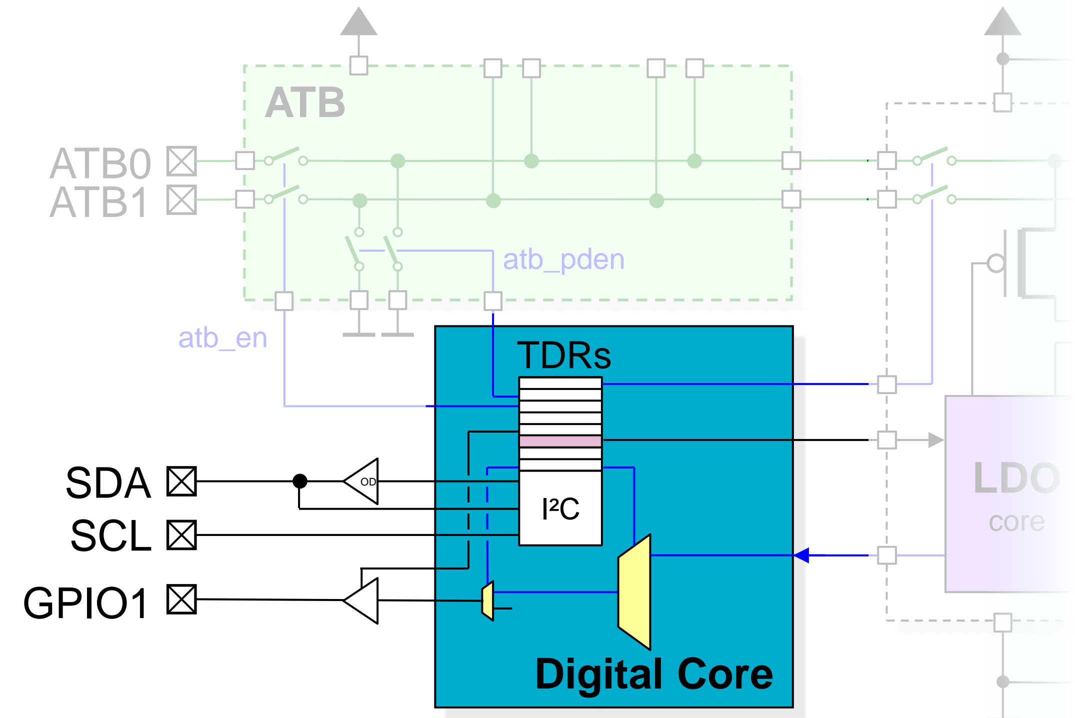

H. M. v. Staudt, M. A. Benhebibi, J. Rearick and M. Laisne, "Industrial Application of IJTAG Standards to the Test of Big-A/little-d devices," IEEE International Test Conference (ITC), 2020

# ICL for I²C Registers

- Example: LDO control register

| Register | Bit Assignments | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | *7* | *6* | *5* | *4* | *3* | *2* | *1* | *0* |
| LDO1_CTRL 0x20 | ldo1 _en | ldo1 _pden | ldo1_vset[5:0] 0x00 = 1.2V, LSB=0.05V, 0x3F = 4.35V | | | | | |

```
Module DIG_CORE {
...
    DataRegister LDO1_CTRL[7:0] {
        Attribute addr = 20;
        ResetValue 8'b01_000000; // LDO off, pulldown on
        WriteCallBack PMIC write_i2c <R> <D>;
        ReadCallBack  PMIC  read_i2c <R> <D>;
    }
    Alias ldo1_en   = LDO1_CTRL[7];
    Alias ldo1_pden = LDO1_CTRL[6];
    Alias ldo1_vset = LDO1_CTRL[5:0];
...
}
```




- Standard code as per IEEE 1687-2014
- How to describe the I²C connectivity?

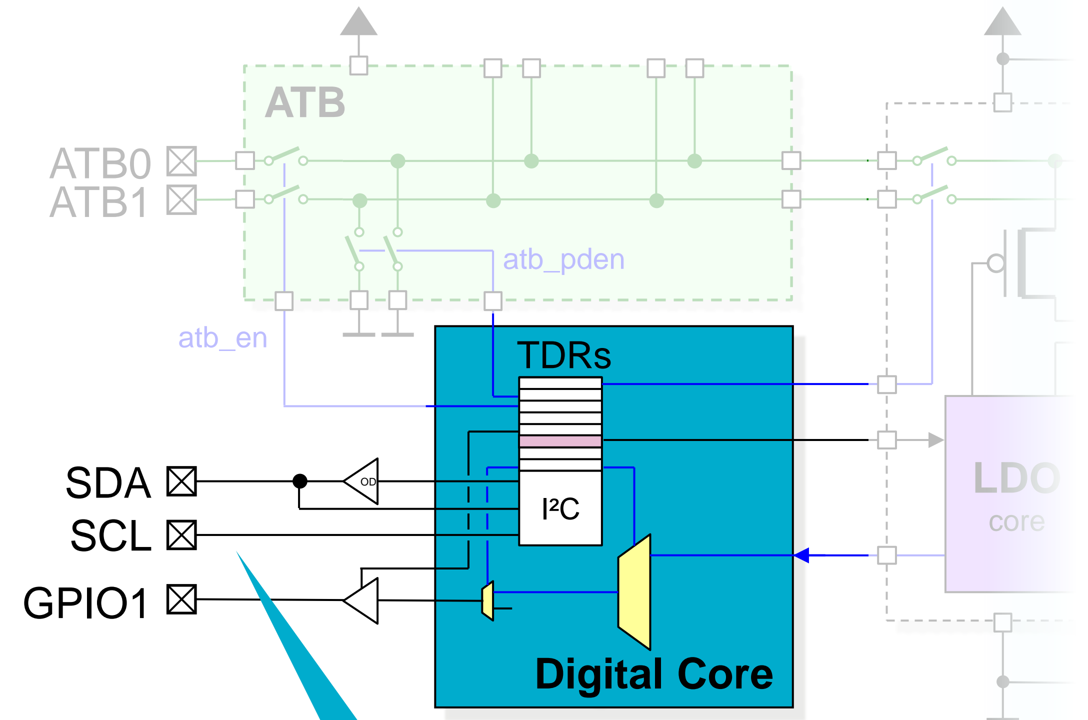
H. M. v. Staudt, M. A. Benhebibi, J. Rearick and M. Laisne, "Industrial Application of IJTAG Standards to the Test of Big-A/little-d devices," IEEE International Test Conference (ITC), 2020

# ICL for I²C Registers: Callback Access

- Example: LDO control register

| Register | Bit Assignments | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | *7* | *6* | *5* | *4* | *3* | *2* | *1* | *0* |
| LDO1_CTRL 0x20 | ldo1_en | ldo1_pden | ldo1_vset[5:0] 0x00 = 1.2V, LSB=0.05V, 0x3F = 4.35V | | | | | |

```
Module DIG_CORE {
...
    DataRegister LDO1_CTRL[7:0] {
        Attribute addr = 20;
        ResetValue 8'b01_000000; // LDO off, pulldown on
        WriteCallBack PMIC write_i2c <R> <D>;
        ReadCallBack  PMIC  read_i2c <R> <D>;
    }
    Alias ldo1_en   = LDO1_CTRL[7];
    Alias ldo1_pden = LDO1_CTRL[6];
    Alias ldo1_vset = LDO1_CTRL[5:0];
...
}
```



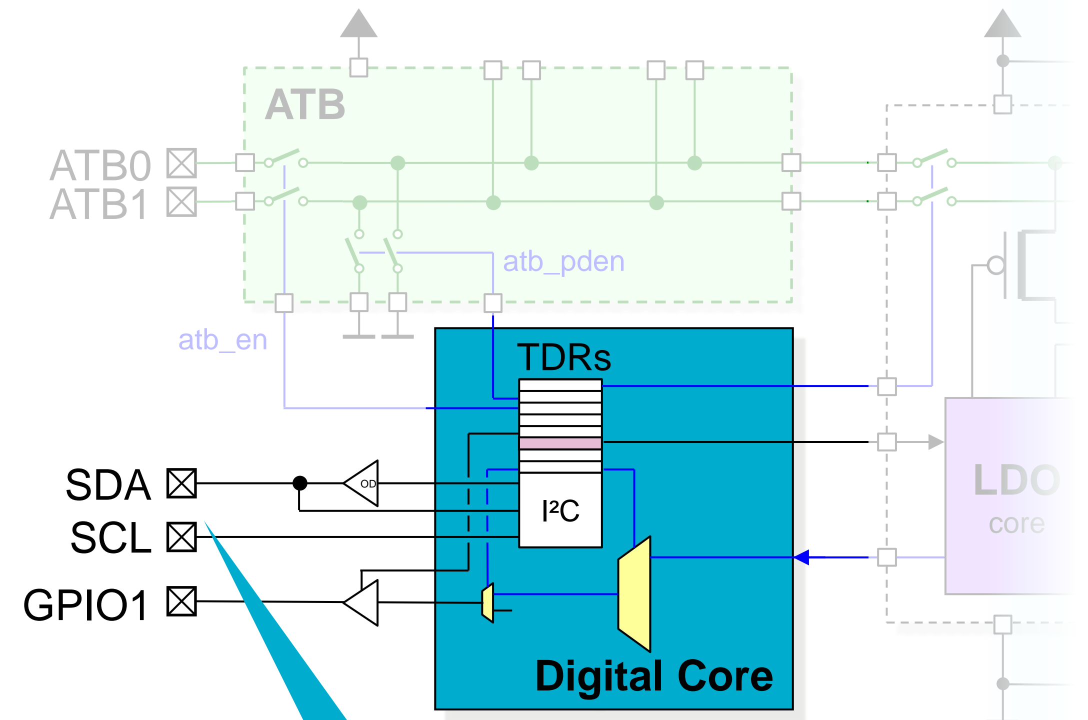

**Callback functions**
- Can operate the protocol at chip level
- No need to describe access network


H. M. v. Staudt, M. A. Benhebibi, J. Rearick and M. Laisne, "Industrial Application of IJTAG Standards to the Test of Big-A/little-d devices," IEEE International Test Conference (ITC), 2020

# ICL for I²C Registers: Callback Procedure

```
Module DIG_CORE {
...
    DataRegister LDO1_CTRL[7:0] {
        Attribute addr = 20;
        ResetValue 8'b01_000000; // LDO off, pulldown on
        WriteCallBack PMIC write_i2c <R> <D>;
        ReadCallBack  PMIC  read_i2c <R> <D>;
    }
    Alias ldo1_en   = LDO1_CTRL[7];
    Alias ldo1_pden = LDO1_CTRL[6];
    Alias ldo1_vset = LDO1_CTRL[5:0];
...
}
```

```
iProcForModule PMIC;
iProc  read_i2c {reg, data} {
  set reg_addr [iGetAttribute $reg addr]; # Fetch register address
  # ... bit stream generation
}
iProc write_i2c {reg, data} {
  set reg_addr [iGetAttribute $reg addr]; # Fetch register address
  # ... bit stream generation
}
...
```

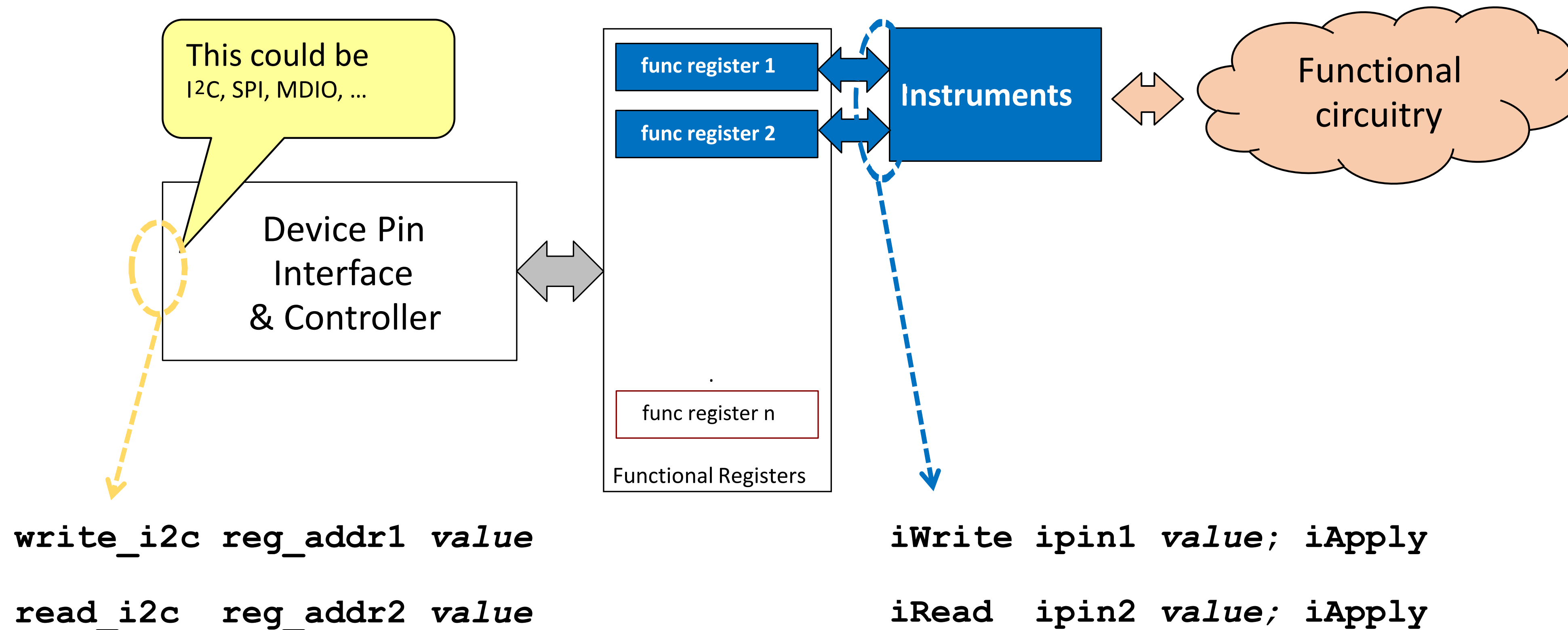



**Callback functions**

- Obtain address from register instance name
- Generate bit stream

H. M. v. Staudt, M. A. Benhebibi, J. Rearick and M. Laisne, "Industrial Application of IJTAG Standards to the Test of Big-A/little-d devices," IEEE International Test Conference (ITC), 2020

# B) Describe I²C Registers as Callback DataRegister




```
write_i2c reg_addr1 value
read_i2c  reg_addr2 value
```

```
iWrite ipin1 value; iApply
iRead  ipin2 value; iApply
```

# 2 Proposals to Reconcile Both Concepts?

Not yet the ideal solution

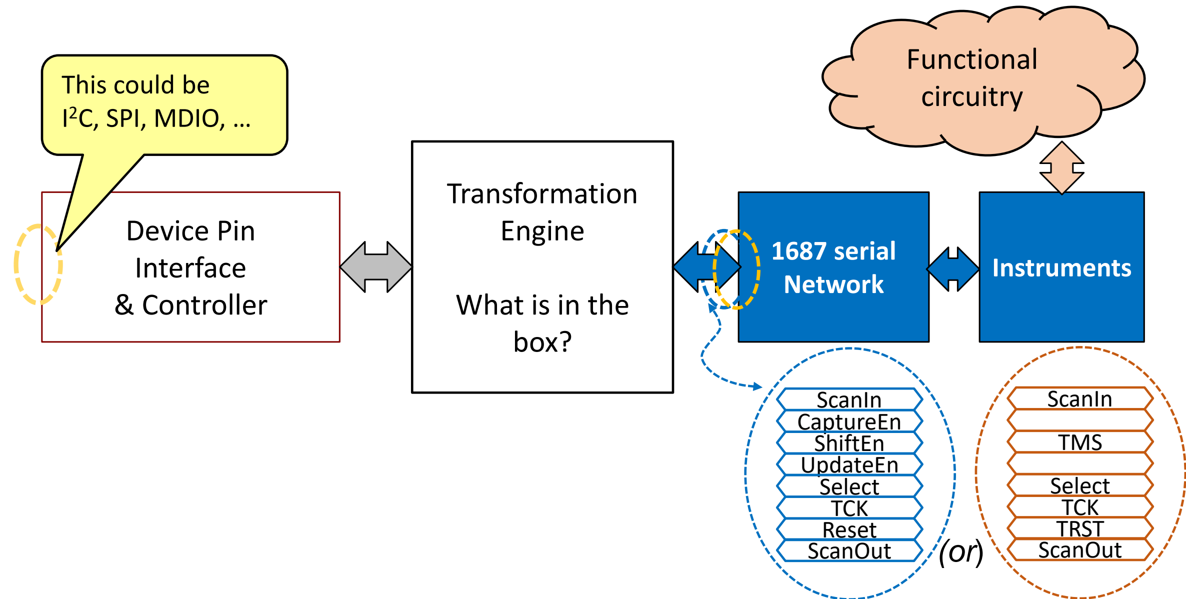
A. Design IJTAG Scan Network to Match I²C transaction concept
- Restricts topology of scan register to regular structure
- No external visibility of 1687 serial network

B. Describe I²C Registers as IJTAG Callback DataRegister
- No serial 1687 network
- Co-existence of legacy I²C with other IJTAG concepts, analog (1687.2) and any P1687.1 solution

# Modeling Non-TAP Interfaces with IEEE P1687.1

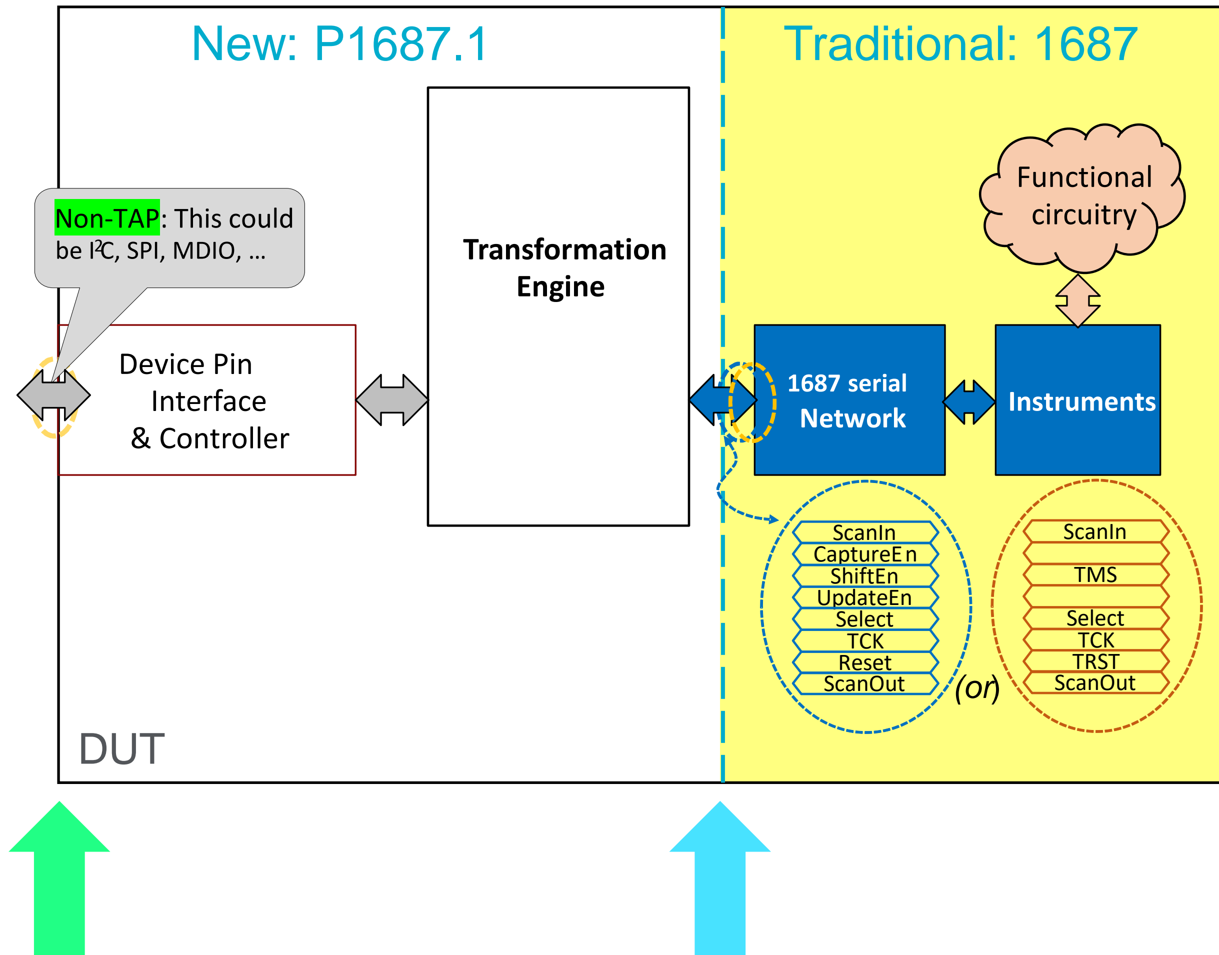
Jeff Rearick, Advanced Micro Devices *(presenter)* ->
Bradford Van Treuren, VT Enterprises Consulting Services

# Goal of IEEE P1687.1 (from Martin's intro)



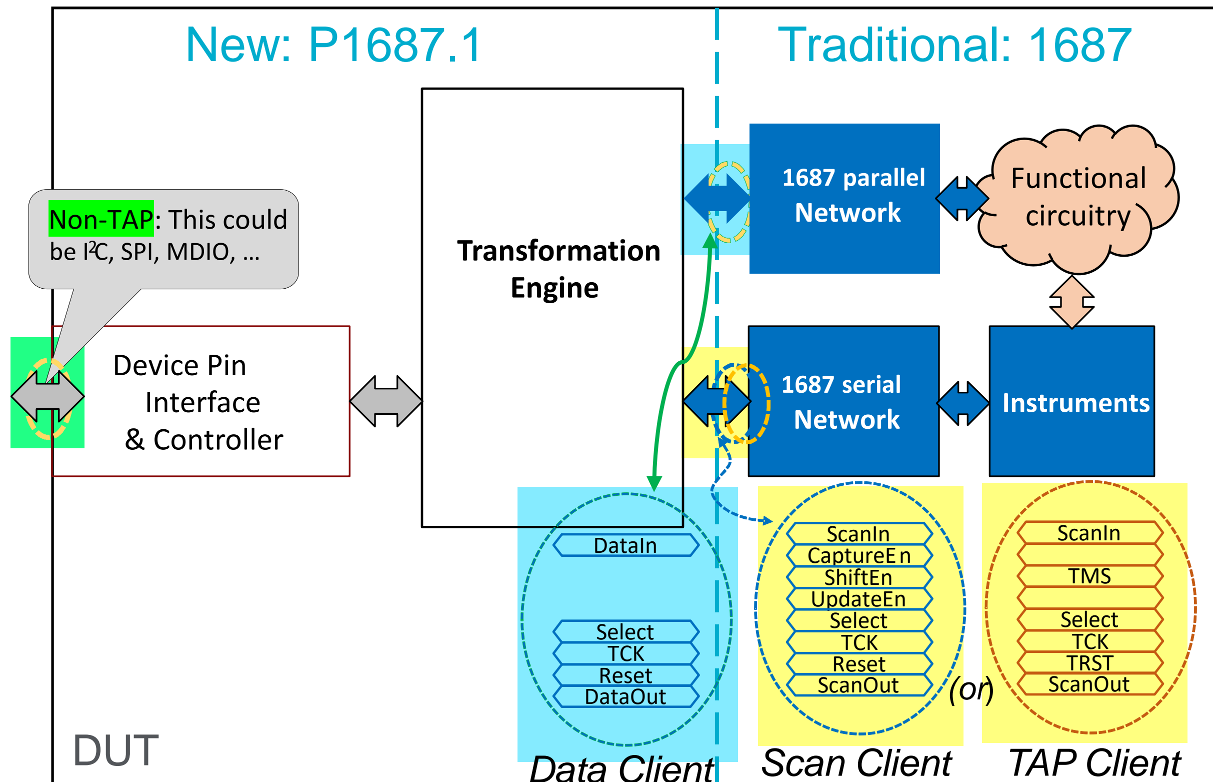

- Access & operation of an IEEE 1687 network through a non-Tap interface
- Like IEEE 1687: Descriptive, not Prescriptive
- How to describe the "Transformation Engine"?

# IEEE P1687.1 Context



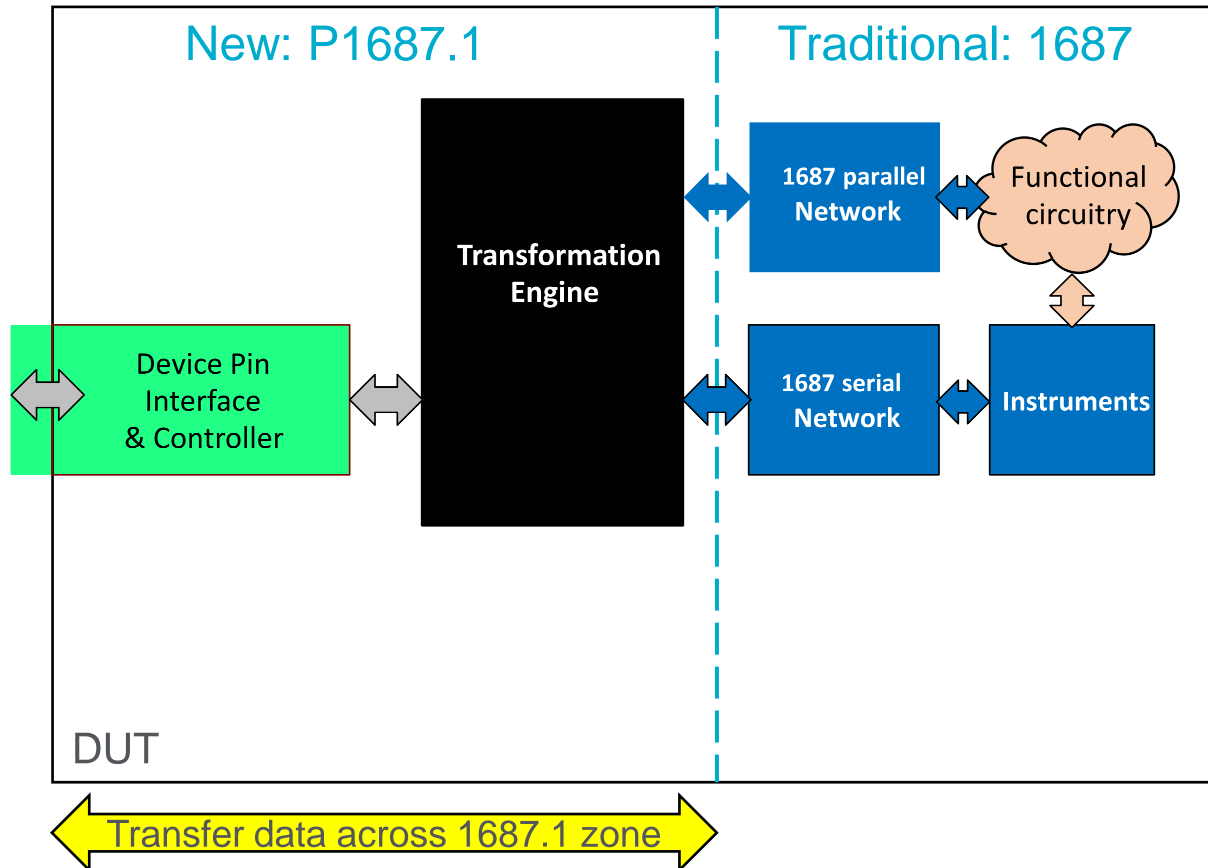

- The 1687 zone stays like it is and will be modeled in ICL
- P1687.1 is needed when the DUT uses a non-TAP interface (on the left side)
- The interface between the right side of the P1687.1 zone and the 1687 zone is a key place to start: it is actually a bit more complex than shown…

# IEEE P1687.1 Expanded Context with DataInterface



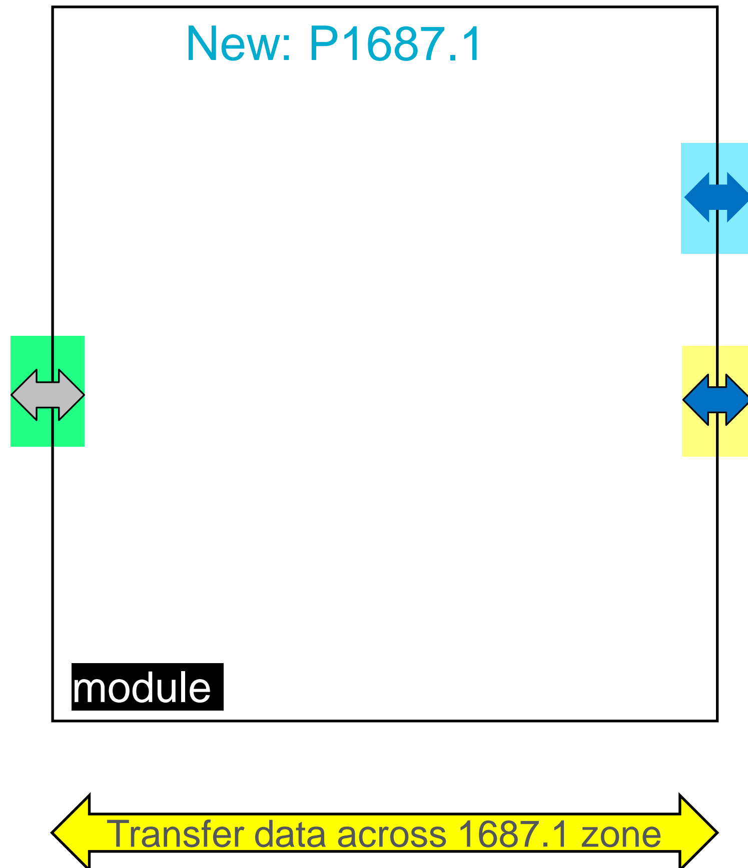

- The ScanInteface is already defined in IEEE 1687 (of type *Scan Client* or *TAP Client*)
- New! P1687.1 plans to codify a DataInterface which serves a similar purpose for parallel 1687 signals (which are also already in place, just not wrapped in an interface yet)
- The (Non-TAP) device pins could be described in ICL, perhaps as part of the AccessLink (TBD)

# IEEE P1687.1 Contents



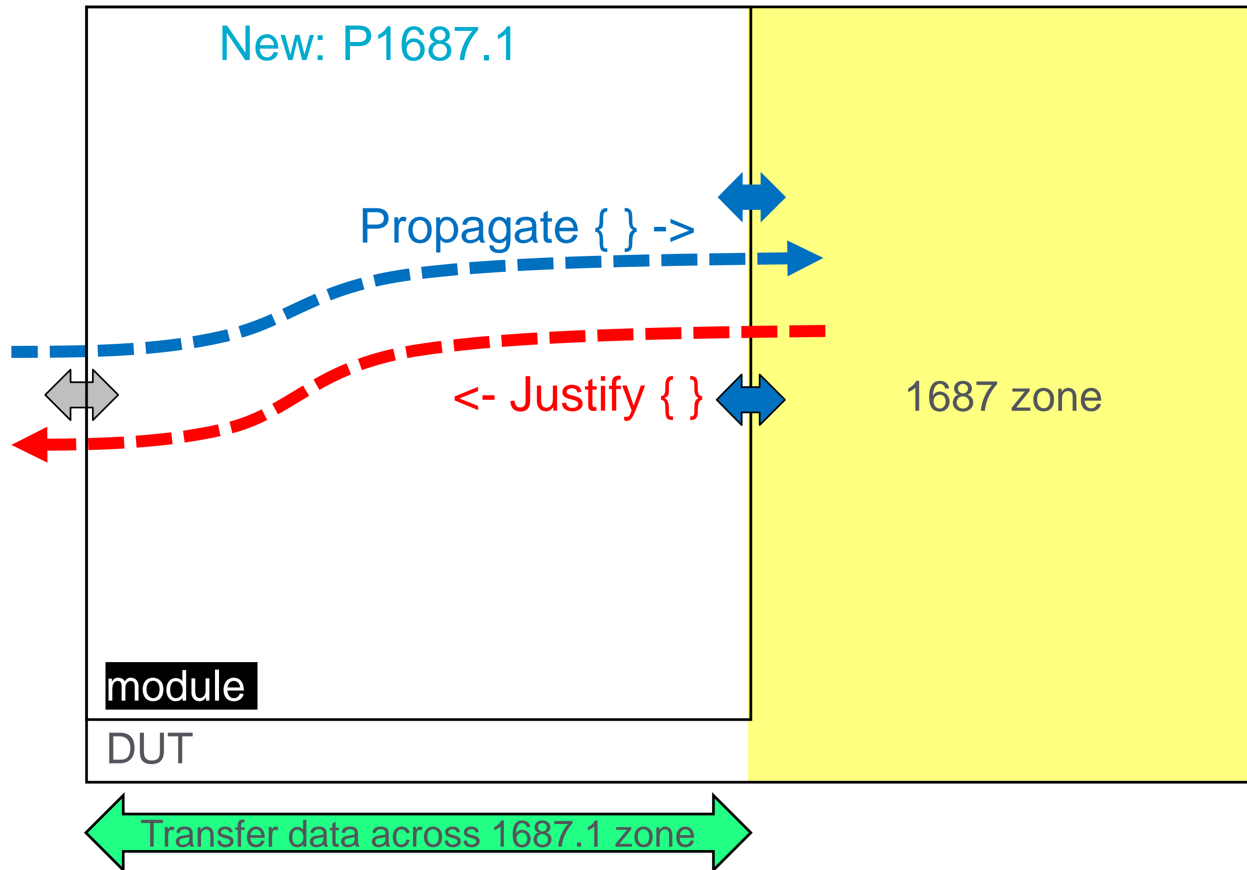

- The non-TAP interface (DPIC) is existing functional logic (like I2C or SPI or …)
- The Transformation Engine (TE) is … a black box from a *hardware description* perspective whose *behavior* is described instead
- In combination, the DPIC and the TE transfer operations across the 1687.1 zone

# IEEE P1687.1 Hardware Modelling Essentials



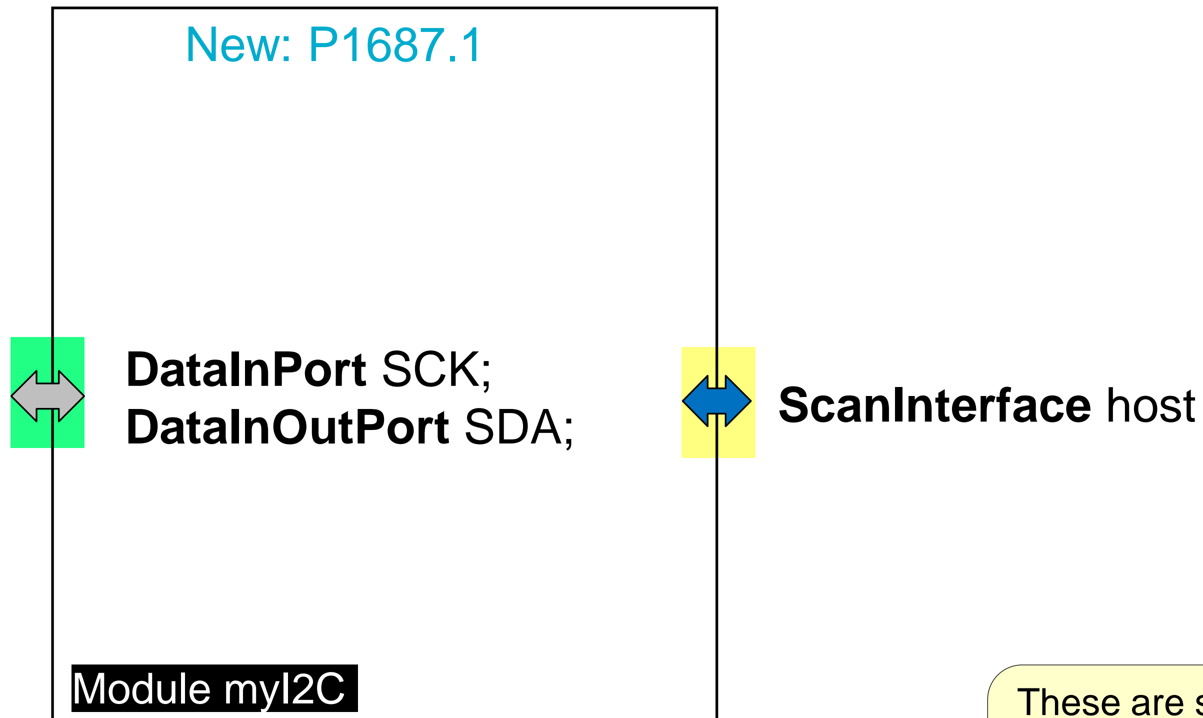

- The non-TAP interface pins
- A ScanInterface if present
- A DataInterface if present
- The module with those ports along with the Transfer Procedure prototypes:
  - Justify { }
  - Propagate { }

# IEEE P1687.1 Transfer Procedures: Two Flavors




- When a PDL test in the 1687 zone is being retargeted, the retargeter needs to Justify commands to the DUT's DPIC through the 1687.1 module
- When the DUT (or the higher-level system containing the DUT) is utilizing circuitry within the 1687 zone, we must Propagate those commands and data through the 1687.1 module

# IEEE P1687.1 Hardware Modelling **ICL** Example: I2C



**TransferProc** Justify { }
**TransferProc** Propagate { }

These are simply the "prototypes" of the procedures: no body here …

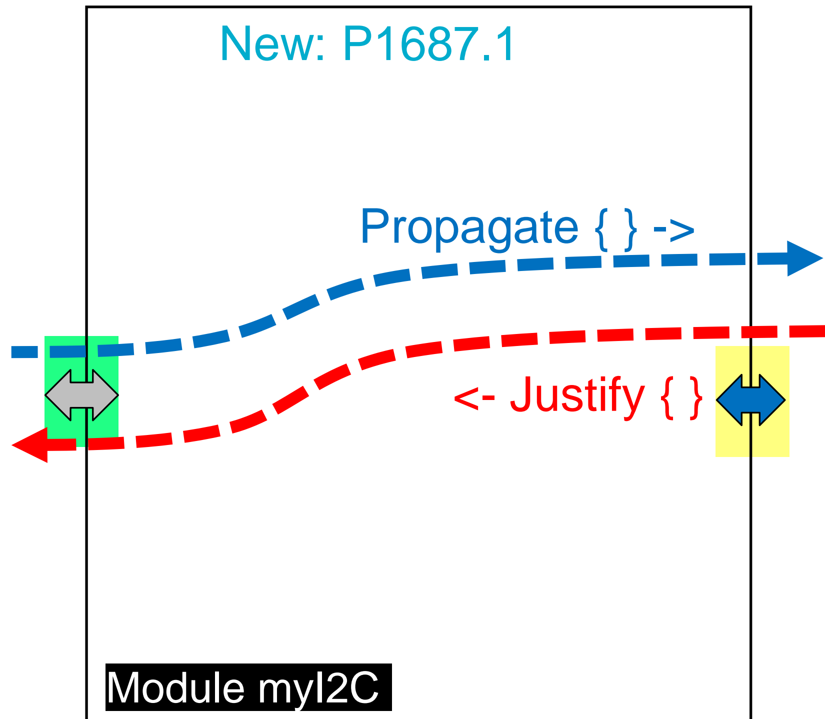
… and yes, this is all there is in the ICL: there is no structural content like registers and muxes

```
Module myI2C_SI {
    DataInPort SCK;
    DataInOutPort SDA;
    ScanOutPort toSI;
    ScanInPort fromSO;
    ToCaptureEnPort Capture;
    ToShiftEnPort Shift;
    ToUpdateEnPort Update;
    ToSelectPort Select;
    ToTCKPort TCK;
    ToResetPort Reset;

    ScanInterface host { Port toSI; Port SO; Port Capture; Port Shift; Port Update; Port Select;  Port TCK; Port Reset; }

    TransferProc Justify { }
    TransferProc Propagate { }
}
```

# IEEE P1687.1 “Hardware” Modelling **PDL** Example: I2C




**TransferProc** Justify { }
**TransferProc** Propagate { }

… and no, this is not all there is in the PDL: the P1687.1 Working Group is still formulating the details of required and allowable content

iProcsForModule myI2C;

**iProc** Justify { } {
…
}

**iProc** Propagate { } {
…
}

# IEEE P1687.1 Behavioral Modelling Essentials

**What comprises a Transfer Procedure?**

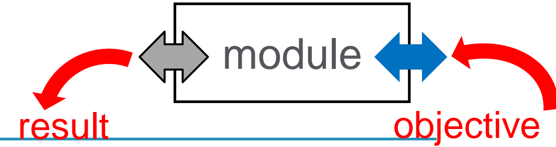
- **Input**: a desired atomic operation ("objective")
- **Output**: a sequence of one or more operations
  - These can be thought of as "new objectives" for the interface on the other side of the module
- **Action**: transform the desired objective at one interface into the sequence of operations at the other interface
- **Requirements**:
  - Operate the module to correctly move the command/data payload through it
  - Enable tracking of payload for debug/diagnosis
  - Format the result in the language which is associated with the other interface
- **Author**: a human with detailed knowledge of the module
- **Consumer**: a piece of software generating a test which needs to traverse the module

```
iProcsForModule myI2C;

iProc Justify { } {
  …
}

iProc Propagate { } {
  …
}
```

# IEEE P1687.1 Behavioral Modelling Example 1a




**What comprises a Transfer Procedure?**

- **Input**: a desired atomic operation ("objective")
- **Output**: a sequence of one or more operations
  - These can be thought of as "new objectives" for the interface on the other side of the module
- **Action**: transform the desired objective at one interface into the sequence of operations at the other interface
- **Requirements**:
  - Operate the module to correctly move the command/data payload through it
  - Format the result in the language which is associated with the other interface
  - Enable tracking of payload for debug/diagnosis
- **Author**: a human with detailed knowledge of the module
- **Consumer**: a piece of software generating a test which needs to traverse the module

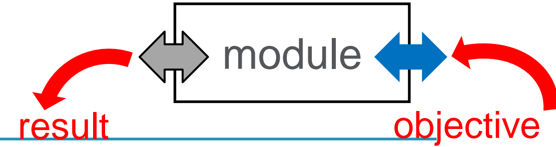
iWrite RegA 0x55AA

iWrite SCK 0b0
iWrite SDA 0b0
iWrite SCK 0b1
iWrite SCK 0b0
iWrite SDA 0b1 ...

This is formatted as bit-banging PDL, which may not be the best method

**iProc** Justify { } {
  … # implementation 1
}

# IEEE P1687.1 Behavioral Modelling Example 1b




**What comprises a Transfer Procedure?**

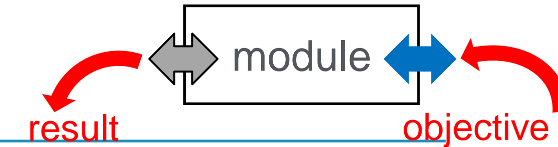
- **Input**: a desired atomic operation ("objective")
- **Output**: a sequence of one or more operations
  - These can be thought of as "new objectives" for the interface on the other side of the module
- **Action**: transform the desired objective at one interface into the sequence of operations at the other interface
- **Requirements**:
  - Operate the module to correctly move the command/data payload through it
  - Format the result in the language which is associated with the other interface
  - Enable tracking of payload for debug/diagnosis
- **Author**: a human with detailed knowledge of the module
- **Consumer**: a piece of software generating a test which needs to traverse the module

```
iWrite RegA 0x55AA
```

```
i2c_write(0x7F);   # TE addr
i2c_write(0x1E);   # write cmd
i2c_write(0x55);   # first 8 bits
i2c_write(0xAA);   # next 8 bits
i2c_write(0x01);   # execute
```

```
iProc Justify { } {
  … # implementation 1
}
```

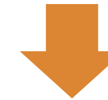
# IEEE P1687.1 Behavioral Modelling Example 2




**What comprises a Transfer Procedure?**

- **Input**: a desired atomic operation ("objective")
- **Output**: a sequence of one or more operations
  - These can be thought of as "new objectives" for the interface on the other side of the module
- **Action**: transform the desired objective at one interface into the sequence of operations at the other interface
- **Requirements**:
  - Operate the module to correctly move the command/data payload through it
  - Format the result in the language which is associated with the other interface
  - Enable tracking of payload for debug/diagnosis
- **Author**: a human with detailed knowledge of the module
- **Consumer**: a piece of software generating a test which needs to traverse the module

```
iWrite RegA 0x55AA
```



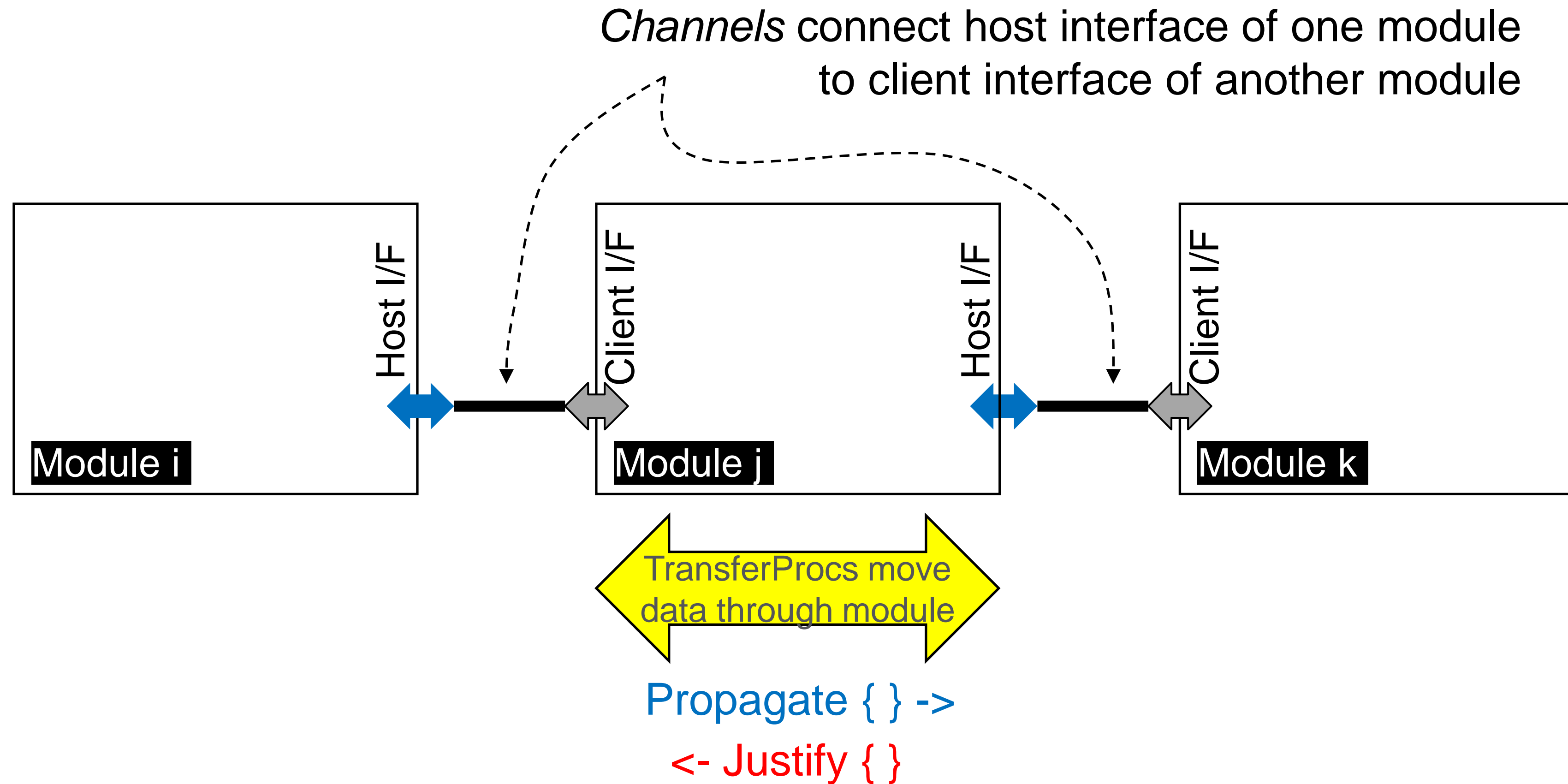
Just another of many possible implementations; P1687.1 is not prescriptive, but descriptive

```
i2c_write(0x7F);   # TE addr
i2c_write(0x10);   # write cmd
i2c_write(0x02);   # num bytes
i2c_write(0x55);   # first byte
i2c_write(0xAA);   # next byte
```

```
iProc Justify { } {
  … # implementation 2
}
```

The module designer chooses how to codify its behavior in TransferProcs

# IEEE P1687.1 Modelling Abstraction Summary




Summary: IEEE P1687.1 describes the modules, the ports comprising their host and client interfaces, and the Transfer Procedures for traversing them

- An objective originates at an *endpoint* module interface (client or host)
- The objective is passed to the next module over channel
- The objective is transformed to one or more new objectives at the other side of that module by applying the appropriate Transfer Procedure (Justify or Propagate)

# A Working Example of IEEE P1687.1 Callbacks

Michele Portolan, Univ Grenoble Alpes CNRS

# What is a Domain?

- A Set of Resources/Operations sharing the same
  - Building Blocks
  - Atomic Operations

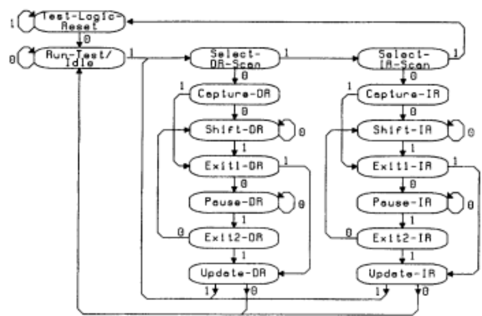
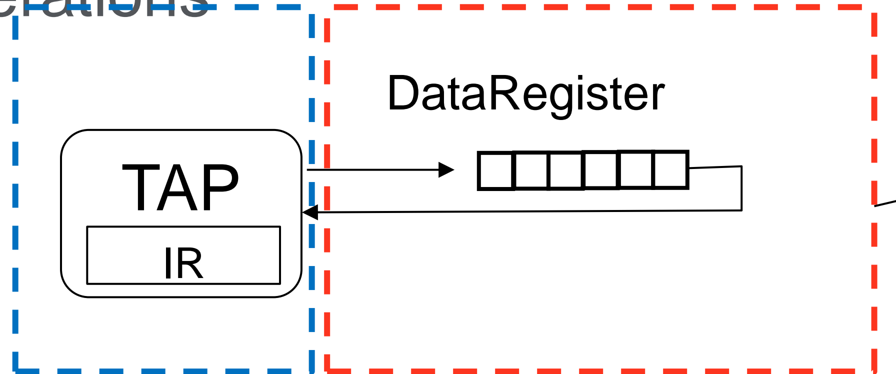



Figure 6-1—TAP controller state diagram




TMS domain: SIR, SDR, etc..

Scan domain: CSU

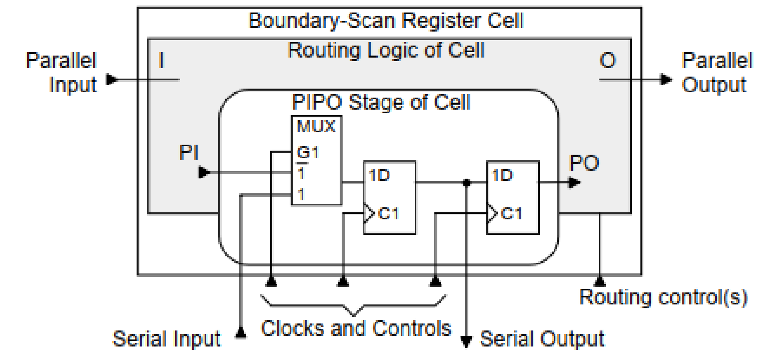



Figure 11-5—Conceptual view of a control-and-observe boundary-scan register cell

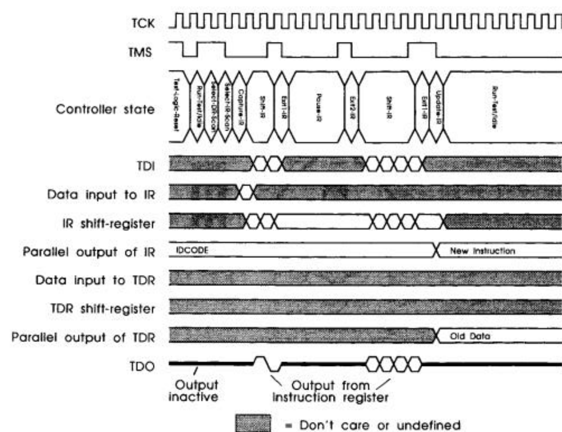



Figure 6-3—Test logic operation: instruction scan

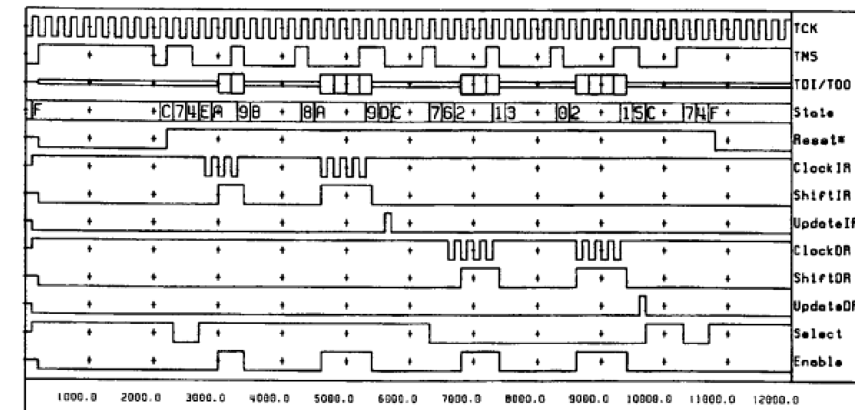
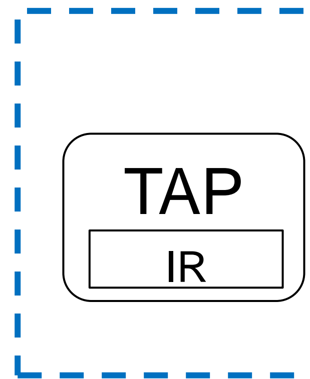


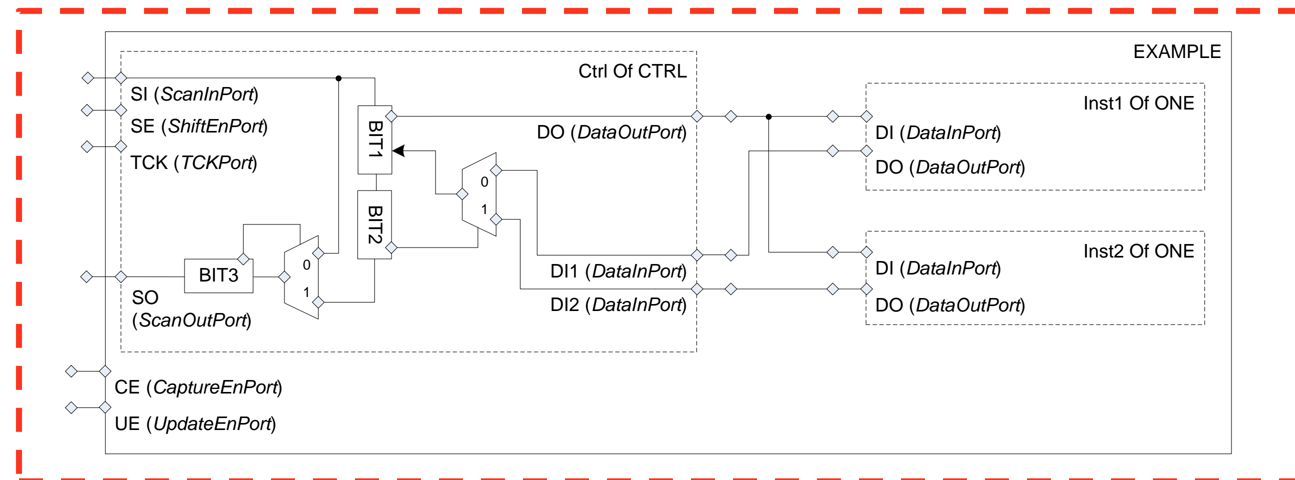

Figure 6-7—Operation of the example TAP controller

# Domain Specific Languages (DSL)

- Describe what is INSIDE a domain



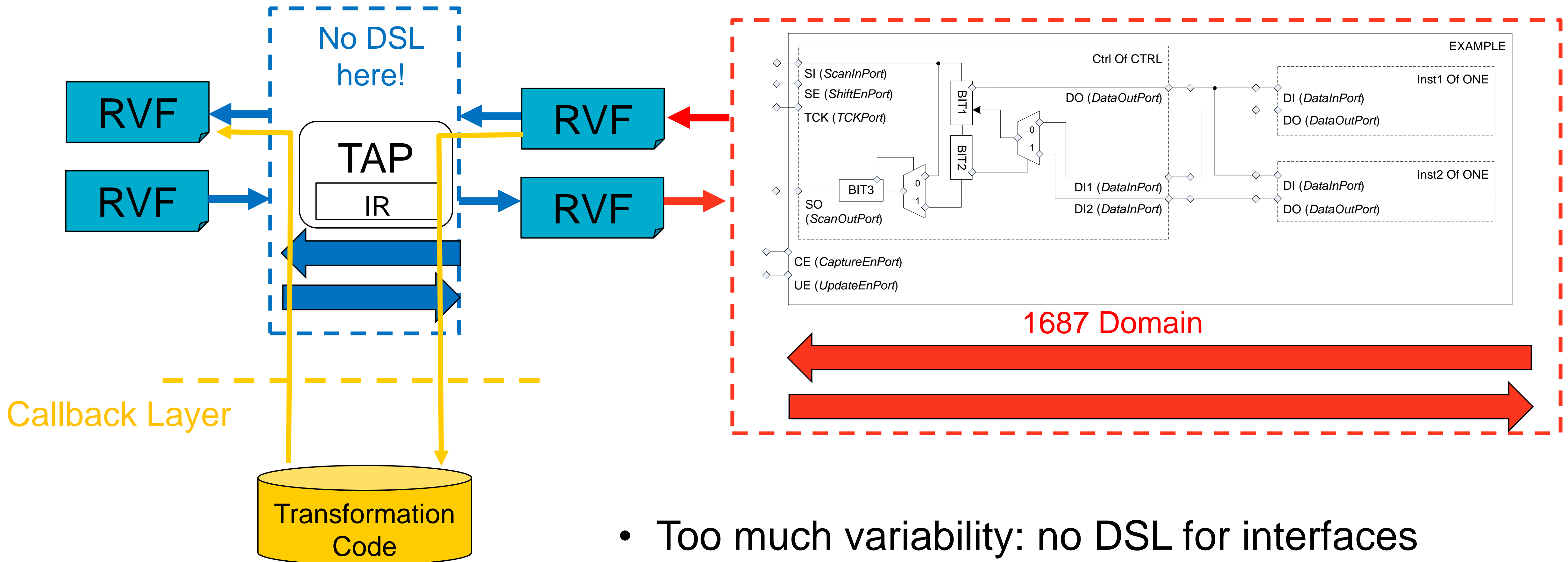

JTAG: Boundary Scan Description Language




1687: Instrument Connectivity Language

Simple, efficient …. and limited!

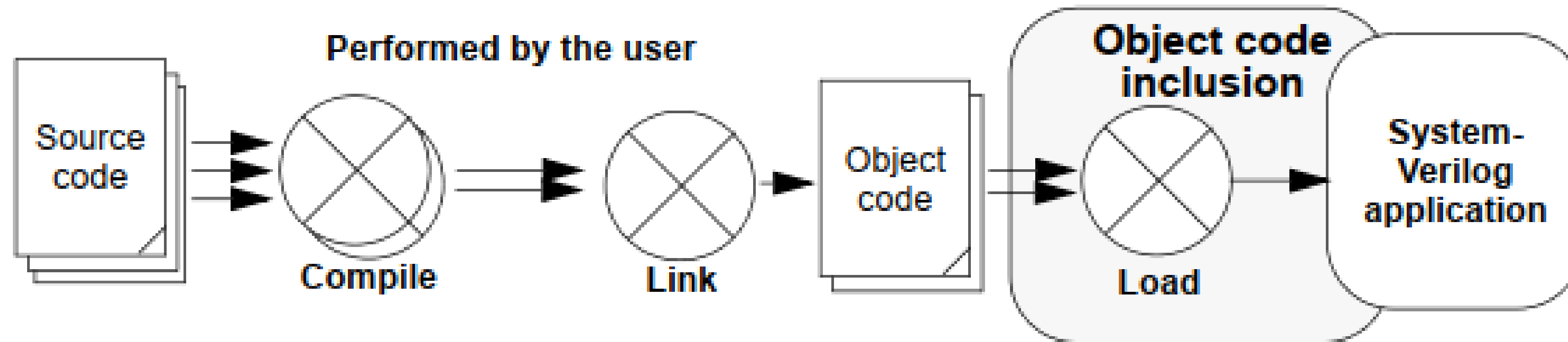
# P1687.1: get out of the Domain!




- Too much variability: no DSL for interfaces
- User provides external Callback for Transformation

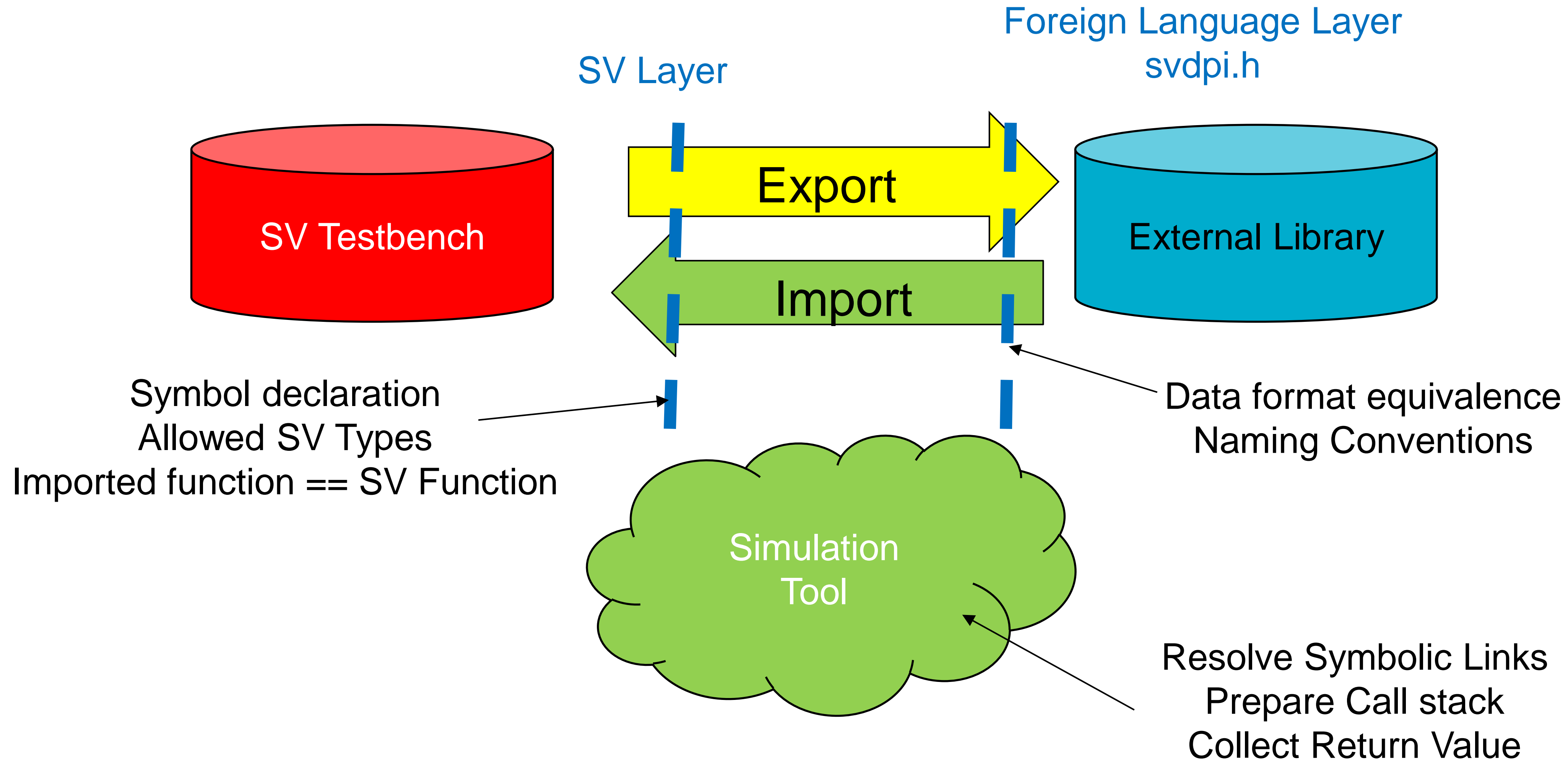
# The Example of System Verilog

- Direct Programming Interface (DPI) : execute external code

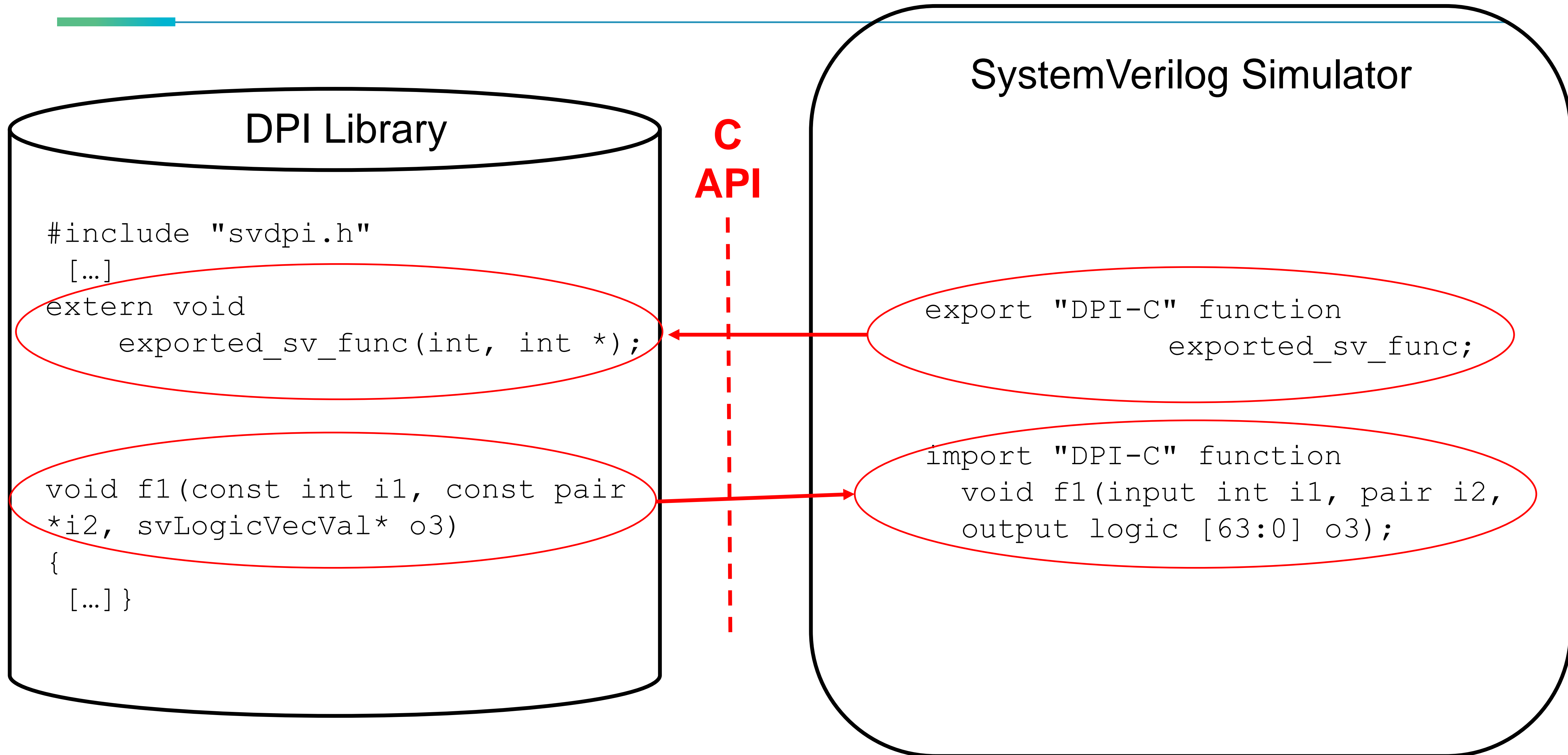



- Enhance RTL Simulators with custom features
- Not limited by SV : language of choice
- Used for complex testbenches: UVM, etc…

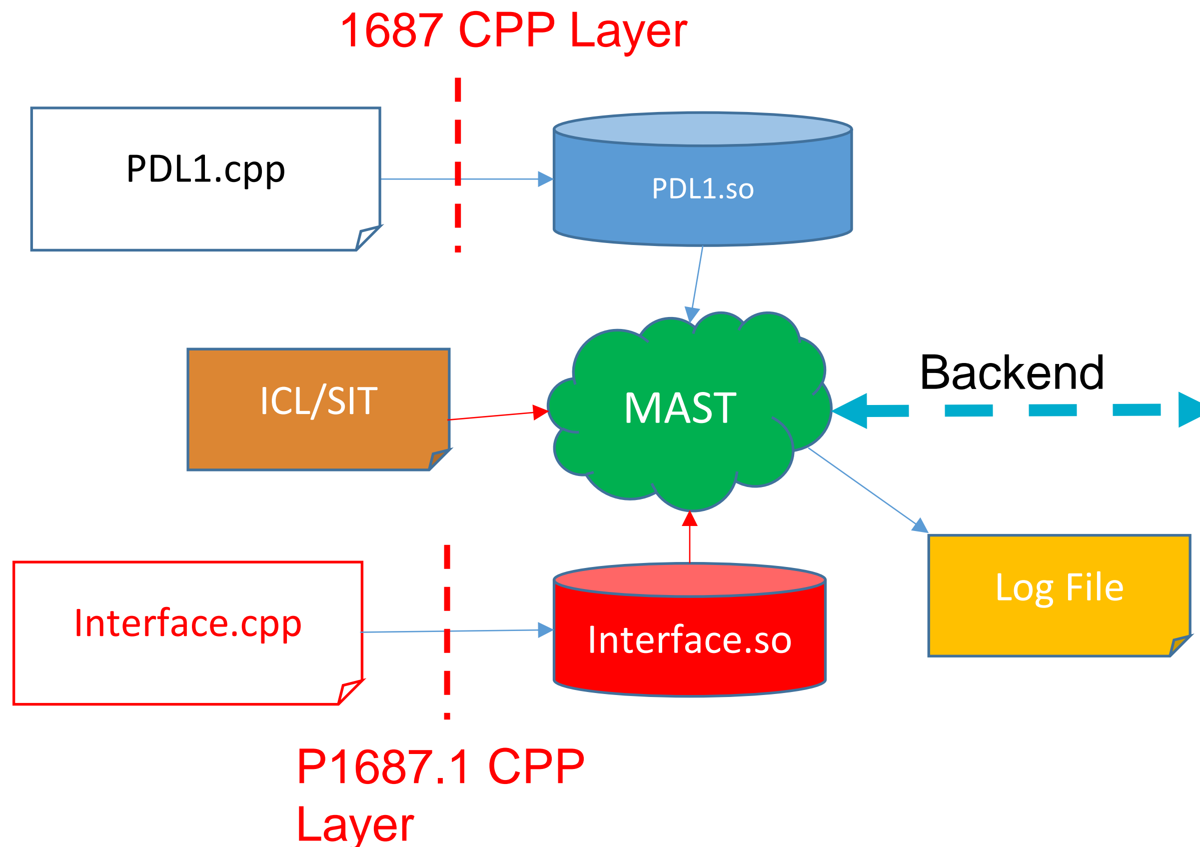
# DPI Layers



# DPI : C API for custom code

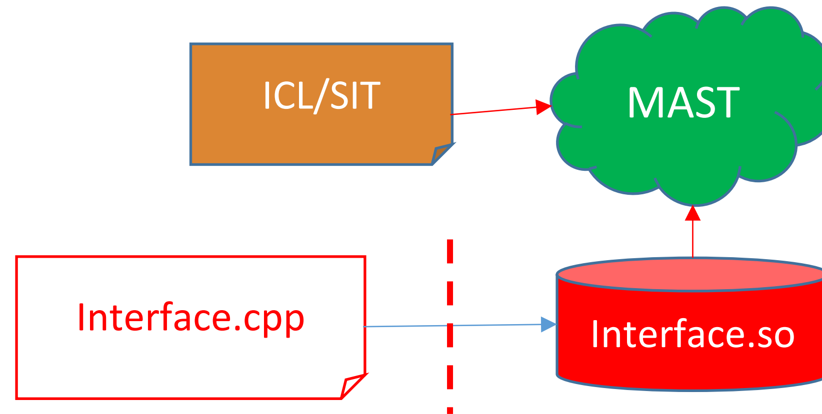


# MAST: Dynamic Interactive 1687/P1687.1 Tool



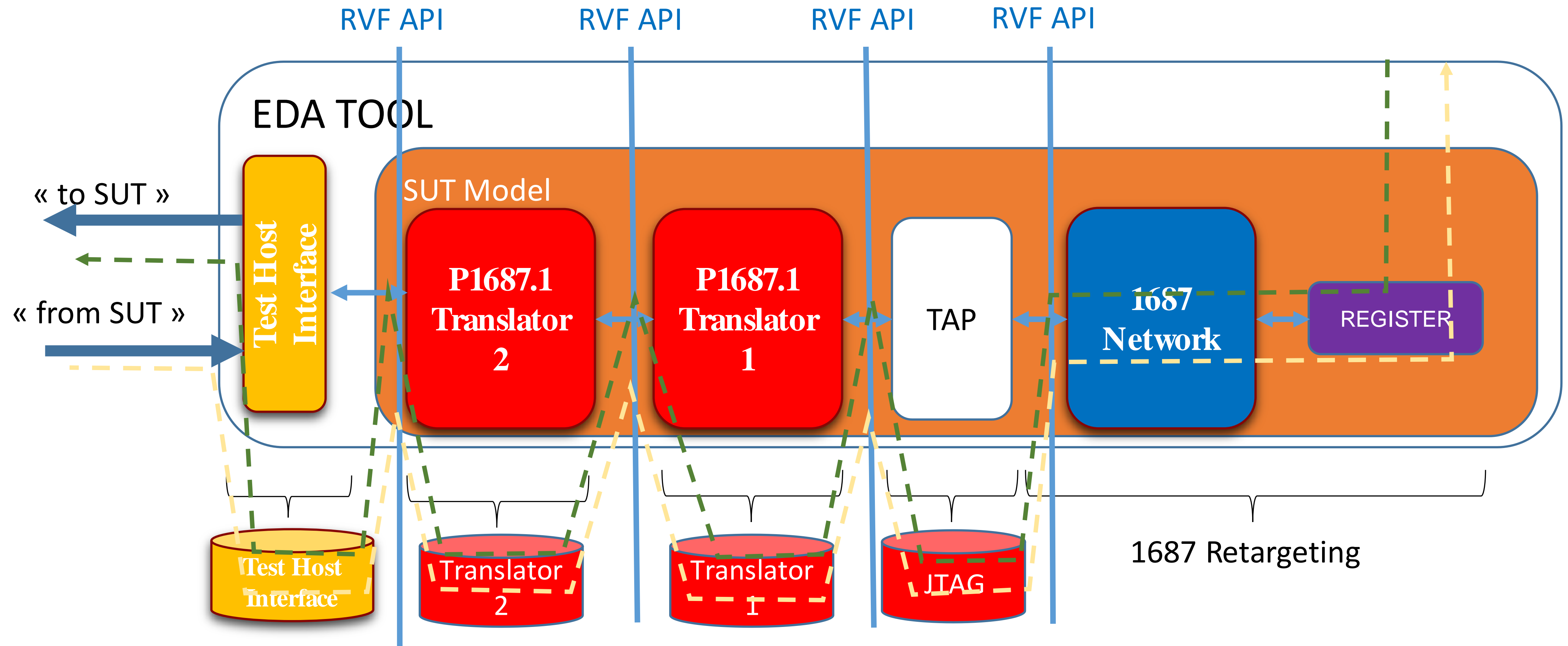

- Dynamic Interactive Functional Retargeting Engine
- Unified middleware for HW/SW interaction
- Massive PDL-1 Concurrency
- Executes **ANY** algorithms, on **ANY** topology, with **ANY** interface
- Extensively uses Callbacks

# MAST P1687.1 implementation: C++ API



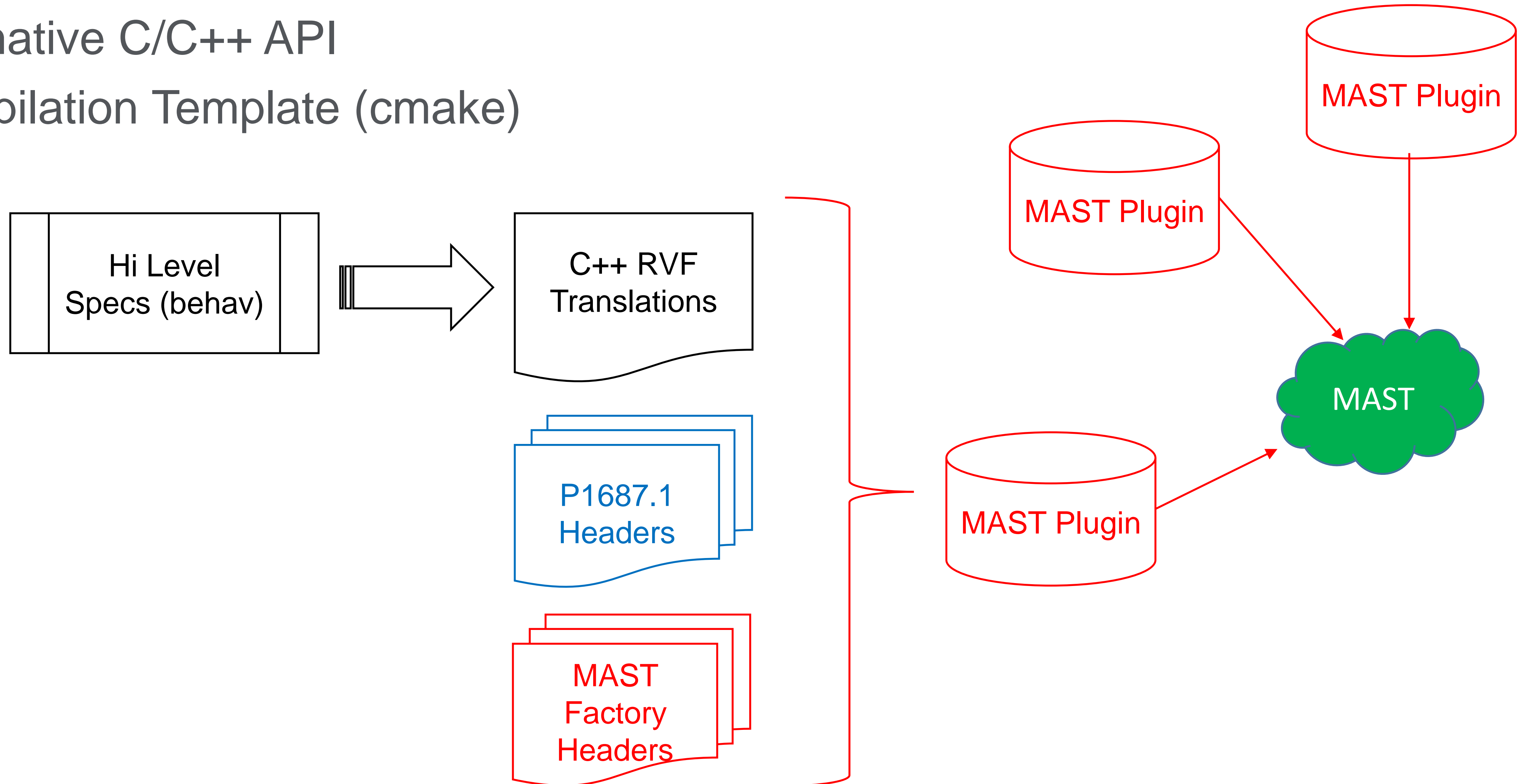

- Relocatable Vector Format (RVF) for Data Exchange
- Normative HPP Headers
- `BinaryVector.hpp`
- `CallbackRequest.hpp`
- `AccessInterfaceProtocol.hpp`
- `AccessInterfaceTranslatorProtocol.hpp`

# P1687.1 Flow: Succession of Translations/Transformations

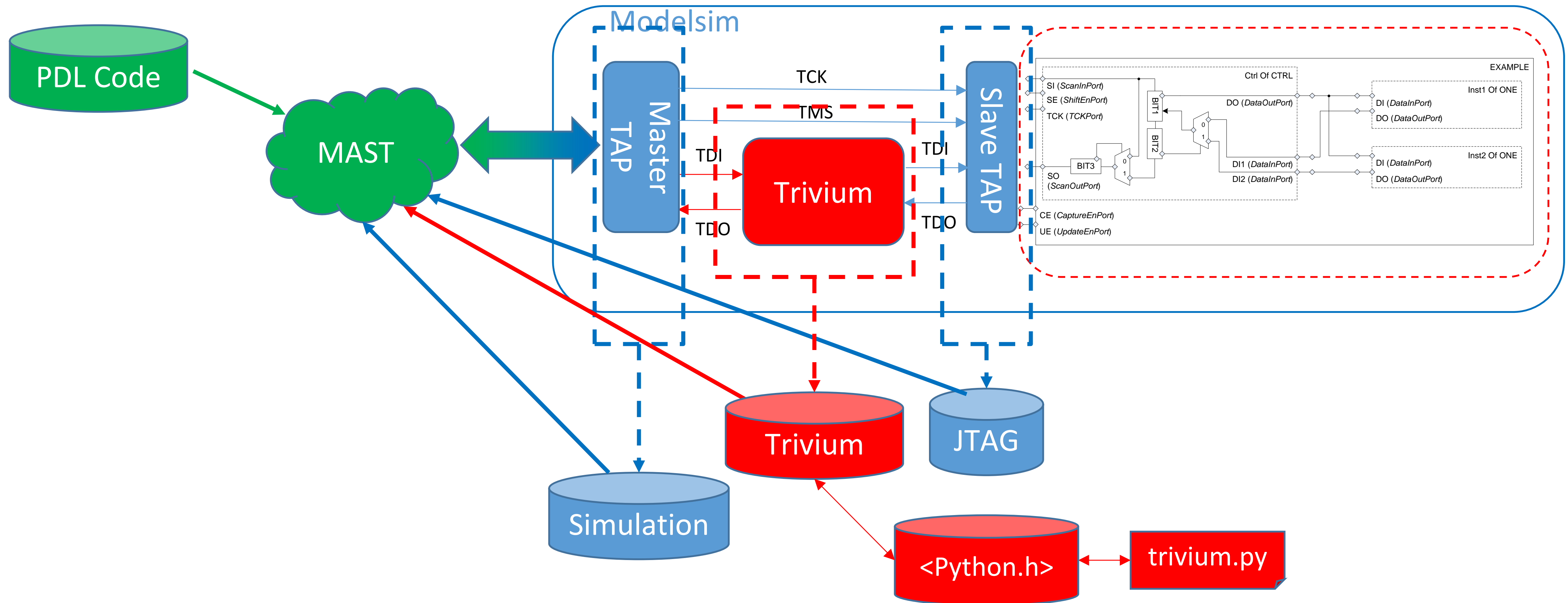


# MAST P1687.1 : Interface Provider

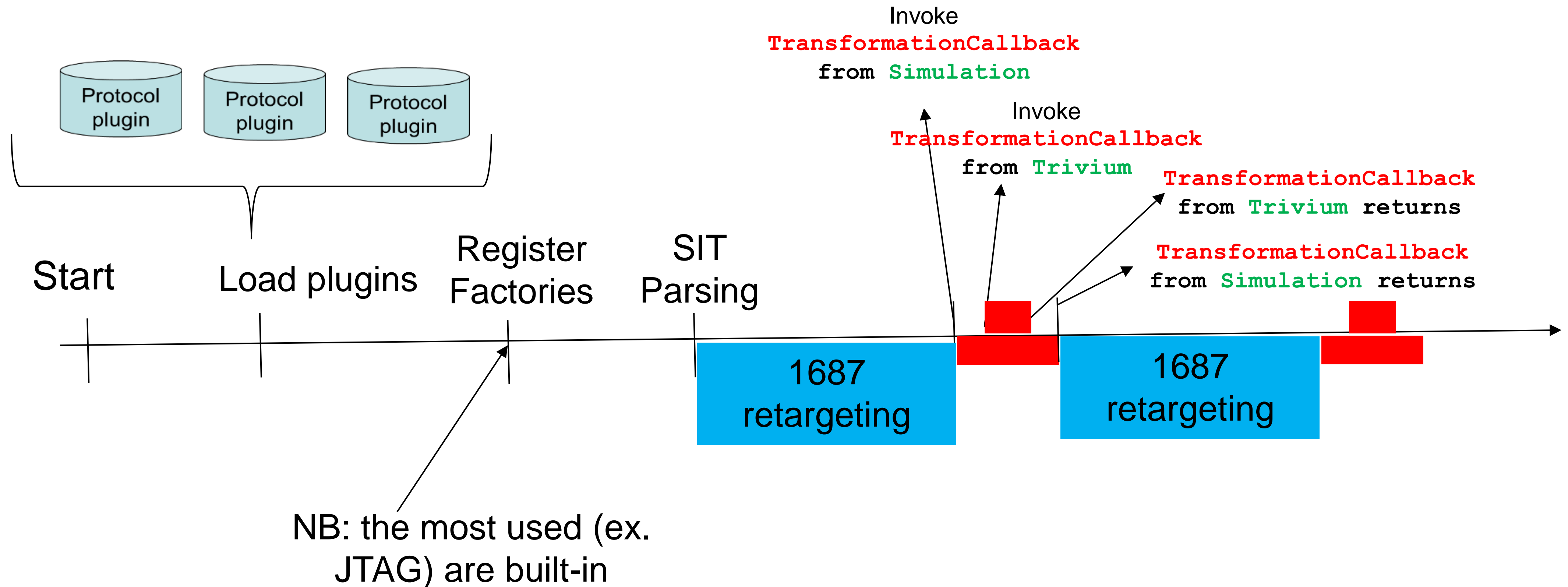
- Normative C/C++ API
- Compilation Template (cmake)



# Example: P1687.1-based Scan Encryption



# MAST P1687.1 Execution Flow



# Conclusions

- Callback Model can Support  Arbitrary Interfaces
- Working Implementation on MAST
  - C/C++ API based on Relocatable Vector Format
  - Template for 3d Party Code
  - Use case for Scan Encryption
- What remains to do for the WG?
  - Finalize RVF Format
  - Normative API : C/C++ … others (ex. gRPC)?
  - Normative Callback Set : which primitives?

# Thank You!

*Jeff Rearick* *Martin Keim* *Michele Portolan* *Brad Van Treuren* *Hans Martin von Staudt*