

# **Reusing and Retargeting On-Chip Instrument Access Procedures in IEEE P1687**

Farrokh Ghani Zadegan, Urban Ingelsson, Gunnar Carlsson and Erik Larsson

**Linköping University Post Print**

N.B.: When citing this work, cite the original article.

©2012 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Farrokh Ghani Zadegan, Urban Ingelsson, Gunnar Carlsson and Erik Larsson, Reusing and Retargeting On-Chip Instrument Access Procedures in IEEE P1687, 2012, IEEE Design & Test Magazine, (29), 2, 79-88.

<http://dx.doi.org/10.1109/MDT.2012.2182984>

Postprint available at: Linköping University Electronic Press

<http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-73803>

---

# REUSING AND RETARGETING ON-CHIP INSTRUMENT ACCESS PROCEDURES IN IEEE P1687

**Farrokh Ghani Zadegan**  
Linköping University

**Urban Ingelsson**  
Semcon AB

**Gunnar Carlsson**  
Ericsson

**Erik Larsson**  
Lund University

## ABSTRACT

Modern chips may contain a large number of embedded test, debugging, configuration, and monitoring features, called instruments. An instrument and its instrument access procedures may be pre-developed and reused, and each instrument—in different chips and through the life-time of a chip—may be accessed in different ways, which requires retargeting. To address reuse and retargeting of instrument access procedures, IEEE P1678 specifies a hardware architecture, a hardware description language, and an access procedure description language. In this paper, we investigate how P1687 facilitates instrument access procedure reuse and retargeting.

**Keywords:** IEEE P1687, on-chip instruments, access procedures, ICL, PDL, reuse and retargeting

## REUSE AND RETARGETING IN TEST STANDARDS

Design reuse is attractive because it reduces IC design time and efforts, allows outsourcing portions of a large design to other companies, and therefore helps to manage the complexity of large designs through modular design approaches. However, when it comes to testing and verification, the benefits of design reuse are unfortunately limited. For example, when a pre-developed test feature, for which the access procedures (or test patterns) are described at its terminals, is reused in a new IC where direct access to those terminals is in general not feasible, the access procedures must be modified such that the instrument can be accessed from the pins of the new IC. This modification is referred to as retargeting. There are tools available for retargeting the test patterns (i.e., concatenating the bit strings corresponding to the test patterns) for each discrete IC on a PCB in order to perform tests at the board-level. Such tools, however, do not yet exist for operating on on-chip test features.

In recent years, the need for reusing access procedures for embedded test, debug, and monitoring features, called on-chip instruments, has become more apparent, since modern chips may contain a large number of such embedded instruments. This abundance of on-chip instruments inside on-chip modules or IP blocks—developed by their respective designers for test and debugging purposes—has given rise to the idea of reusing that wealth of on-chip instruments. Even though designed for a particular time in the life cycle of a chip, instruments can be accessed throughout its whole life cycle. To efficiently reuse instruments, it is necessary to automatically retarget the instrument data—or the access procedures described for a given instrument at its terminals—to the higher design levels up to the chip pins through an instrument access infrastructure.

If we want to make on-chip instrument access infrastructure available for off-chip usage, the ubiquitous IEEE 1149.1 (JTAG) [1] Test Access Port (TAP) is an attractive alternative. To implement and use JTAG, a set of languages are employed: to describe the JTAG circuitry there is the Boundary-Scan Description Language (BSDL), and to describe the JTAG TAP operations there are Serial Vector Format (SVF) [2] and Standard Test and Programming

Language (STAPL) [3]. BSDL, however, is not sufficient to describe all types of instruments or the instrument access infrastructure, nor is it efficient when describing large number of instruments. As an example, there are some SerDes implementations for which the Test Data Register (TDR) length varies depending on the performed operation. A variable length TDR is not supported by BSDL and therefore, a separate JTAG instruction should be used for each operation. Many JTAG instructions will lead to a long instruction register (IR) and complex decoding logic, which may slow down the test clock (TCK). Similarly, SVF and STAPL do not lend themselves well to the design reuse practice inside the chip. For example, SVF does not support aliases or enumerations to make the code easy to reuse and to maintain. In particular, SVF and STAPL are not linked to the BSDL description of the JTAG circuitry and therefore, prove inefficient when we try to automatically retarget the access procedures, described in SVF or STAPL at the instrument's terminals, to the chip pins. As an example, currently there is no standard way for third-party vendors to provide procedures for accessing instruments inside their designed IP blocks—which are to be embedded in a larger design. Therefore, the test engineer has the responsibility of generating access procedures at the terminals of the IP block and translating those IP-level procedures to the system level. That is, even if we have the access procedures for a given instrument in the form of bit strings at its terminals, it is difficult to retarget those procedures to higher levels, since current EDA tools do the bit string concatenation only at the board level and not the chip-level. It should be noted that the Core Test Language (CTL) [4], which leverages previous standards such as Standard Test Interface Language (STIL) [5], can be used to describe the wrapped and unwrapped cores specified in IEEE 1500 [6]. CTL can be seen as a language for describing both the core (instrument) interface and the test patterns (access procedures) for that core. Although CTL provides the means to reuse the test patterns developed for a core at the next level of integration, it cannot be used to fully describe the behavior of instruments with complicated access procedures. For example, CTL lacks the flexibility of a programming language, which is required to describe the access procedures that make decisions based on responses of application of previous test patterns. Furthermore, CTL cannot be used to describe the on chip instrument access infrastructure, and there are also incompatibilities between IEEE 1500 and IEEE 1149.1 [7].

Since current standards are limited in terms of describing instruments, reuse, and retargeting, a proposal for a new standard IEEE P1687 [8] introduces two new languages, Instrument Connectivity Language (ICL) and Procedural Description Language (PDL), to standardize the access and control of on-chip instruments. In this paper, we give an introduction to P1687 and compare the ICL/PDL pair with BSDL/SVF regarding their utility in reuse and retargeting of instrument access procedures.

P1687 specifies JTAG as an off-chip to on-chip interface to the instrument access infrastructure (the P1687 *network*) and is informally called Internal JTAG (IJTAG). P1687 includes (1) specifications on the hardware that interfaces the on-chip instruments to the outside world (see Appendix), (2) ICL which describes the instrument's port functions and logical connection to other instruments and to the JTAG TAP, and (3) PDL which describes how an instrument should be operated. The idea in introducing ICL and PDL is to provide an adequate and standardized description of the P1687 network, instruments, and instrument access procedures, and to enable ICL and PDL interpreter tools to automate the retargeting of access procedures.

In comparing ICL/PDL with BSDL/SVF, note that ICL and BSDL both describe on-chip features (accessed through the JTAG TAP). However, ICL is used in conjunction with BSDL (when using a P1687 network through a JTAG TAP) and does not replace it. PDL, STAPL and SVF all describe how to use on-chip features. In this paper, we compare PDL with SVF, rather than with STAPL, because SVF is currently a de facto standard at the board level to operate the JTAG TAP and interchange test data between EDA tools. For the comparison that is presented in this paper, we start by describing a very simple scenario in which a temperature sensor is accessed through the JTAG TAP, using firstly BSDL/SVF and secondly ICL/PDL. Subsequently, we extend the scenario to multiple sensors serially connected on the JTAG scan-path, and make the scan path configurable (flexible) through the use of P1687-specific components.

## A SMALL EXAMPLE: ACCESSING A SIMPLE INSTRUMENT

In this section, we illustrate how to access an embedded instrument using BSDL/SVF and ICL/PDL. We make use of a temperature sensor (see Figure 1) which, when enabled, makes the temperature available at its terminals as a 4-bit number, after 10 system clocks.

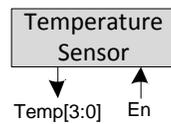


FIGURE 1 SHOWS THE TEMPERATURE SENSOR

Assume that we need to access our temperature sensor through the JTAG TAP as shown in Figure 2. One option is using BSDL to describe the JTAG circuitry, and SVF to describe how to access the sensor. P1687 provides another option which is using ICL in conjunction with BSDL to describe the network, and PDL to describe the access procedure for the sensor. We will compare the two alternatives in the following subsections.

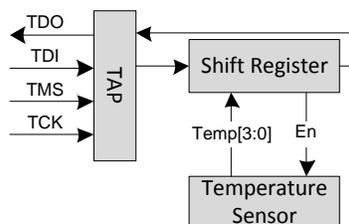


FIGURE 2 SHOWS THE CONNECTION OF THE TEMPERATURE SENSOR TO THE JTAG TAP THROUGH A SHIFT AND UPDATE REGISTER.

### HOW TO ACCESS THE SENSOR BY USING BSDL/SVF

Figure 2 shows a partial view of the JTAG circuitry that we have considered for this example. To keep the illustration easy to read, the rest of the mandatory components such as instruction register, instruction register decoder, etc. are not shown (see Appendix for a more detailed view of a JTAG circuitry). Clock and control signals (e.g., shift enable) are not shown in Figure 2. Listing 1 shows the partial BSDL description of the JTAG circuitry shown in Figure 2. In Listing 1, we have replaced some lengthy parts of the code which are irrelevant to this discussion with (. . .).

## LISTING 1 THE PARTIAL BSDL DESCRIPTION FOR THE DESIGN IN FIGURE 2

```
01  entity Chip_BSDL is
02      generic (PHYSICAL_PIN_MAP : string := "DIP22_PACKAGE");
03
04      port (...);
05
06      use STD_1149_1_1990.all;
07
08      attribute PIN_MAP of Chip_BSDL : entity is PHYSICAL_PIN_MAP;
09      constant DIP22_PACKAGE : PIN_MAP_STRING := "...";
10
11      attribute TAP_SCAN_IN   of TDI : signal is true;
12      attribute TAP_SCAN_MODE of TMS : signal is true;
13      attribute TAP_SCAN_OUT  of TDO : signal is true;
14      attribute TAP_SCAN_CLOCK of TCK : signal is (20.0e6, BOTH);
15
16      attribute INSTRUCTION_LENGTH of Chip_BSDL : entity is 4;
17      attribute INSTRUCTION_OPCODE of Chip_BSDL : entity is
18          "BYPASS (1111, 0000) , " &
19          "EXTTEST (0001, 1001) , " &
20          "SAMPLE (0010, 1010) , " &
21          "INTEST (0011, 1011) , " &
22          "HIGHZ (0100, 1100) , " &
23          "CLAMP (0101) , " &
24          "READINSTR (0111, 1000) " ;
25
26      attribute INSTRUCTION_CAPTURE of Chip_BSDL : entity is "0001";
27      attribute INSTRUCTION_DISABLE of Chip_BSDL : entity is "HIGHZ";
28      attribute INSTRUCTION_GUARD  of Chip_BSDL : entity is "CLAMP";
29
30      attribute REGISTER_ACCESS of Chip_BSDL : entity is
31          "BOUNDARY (EXTTEST, INTEST, SAMPLE)," &
32          "BYPASS (BYPASS, HIGHZ, CLAMP)," &
33          "INSTR[4] (READINSTR)" ;
34          .
35          .
36          .
37  end Chip_BSDL;
```

For our discussion, the interesting points in Listing 1 are how a JTAG instruction called READINSTR is defined (Line 24) and how the 4-bit TDR that this instruction selects is specified as INSTR[4] (Line 33). This 4-bit TDR represents the shift and update register in Figure 2. We have assumed that the same 4-bit register is used both for enabling the sensor (using the LSB) and for capturing the temperature value (all four bits).

Listing 2 shows a sample SVF script that reads our sensor according to the procedure stated above. We developed this SVF script assuming (as in Figure 2) that there are no other JTAG devices on the scan path, i.e., the TDI and TDO terminals of the TAP are directly connected to the external tester. Please note how the READINSTR instruction is loaded in Line 1 of Listing 2.

## LISTING 2 SHOWS THE SVF FILE

```
01 SIR 4 TDI (7);           ! 4-bit IR scan => Loading READINSTR ("0111")
02 SDR 4 TDI (1);         ! 4-bit DR scan => Activating the sensor ("0001")
03 STATE DRPAUSE;        ! Going to state DRPAUSE
04 RUNTEST 10 SCK ENDSTATE DRPAUSE; ! Waiting for 10 system clocks
05 SDR 4 TDI (0);         ! Shifting out the temperature
06 STATE IDLE;           ! Going to state Run-Test/Idle
```

## HOW TO ACCESS THE SENSOR BY USING ICL/PDL

Although ICL has some overlap with BSDL, it is not meant to replace it. In fact, the JTAG TAP and boundary scan related circuitry will still be in the scope of BSDL. Therefore, for our small temperature sensor, we still need to use the BSDL in Listing 1 to describe the instruction opcode used for placing the TDR (shift register in Figure 2) between the TDI and TDO terminals. Listing 3 shows the ICL code that describes a P1687 network consisting of a sensor and a shift register, as well as how this network is interfaced to the JTAG TAP. For the sake of brevity, we have not shown and port-mapped the clock and control signals—i.e., ShiftEn, CaptureEn, and UpdateEn—which are required for the operation of the shift register.

LISTING 3 SHOWS ICL DESCRIPTION OF THE P1687 NETWORK IN FIGURE 2

```

01  Module Sensor {
02      DataInPort      en;
03      DataOutPort     temp[3:0];
04  }
05
06  Module TDR {
07      ScanInPort      si;
08      ScanOutPort     so { Source SR[0];
09                      LaunchEdge Falling; }
10      DataInPort      pi[3:0];
11      DataOutPort     po { Source SR[0]; }
12
13      SelectPort      en;
14
15      ScanRegister    SR[3:0]{ ScanInSource si;
16                          CaptureSource pi; }
17  }
18
19  Module Chip_ICL {
20      Instance TDR1 Of TDR {
21          InputPort en = Tap1.en_TDR;
22          InputPort pi = Sensor1.temp;
23      }
24      Instance Sensor1 Of Sensor {
25          InputPort en = TDR1.po;
26      }
27      AccessLink Tap1 Of STD_1149_1 {
28          BSDL_Entity Chip_ICL;
29          READINSTR {
30              ScanPath { TDR1; }
31              ActiveSignals { en_TDR; }
32          }
33      }
34  }

```

Listing 3 contains three modules, one describing the interface of the sensor (Lines 1–4), one describing the shift register that connects the sensor’s terminals to the scan path (Lines 6–17), and the chip level module (Lines 19–34). The description of the sensor’s interface (i.e., the *Sensor module*) contains the I/O ports shown in Figure 2, namely the *en* input terminal and the *temp* output terminal. The description of the shift register (i.e., the *TDR module*) is more detailed since it contains both connection to the scan path (Lines 7–9) and connection to the sensor (Lines 10–11). The main component inside the TDR module is *ScanRegister* (Lines 15–16) which is among the *primitive building blocks* specified by ICL. For *ScanRegister*, the scan-in source and the parallel capture ports are described to connect it to the serial (scan) and parallel terminals of the TDR module.

The chip level module, i.e., *Chip\_ICL*, puts it all together by instantiating and port mapping the other modules (Lines 20–26), and connecting them appropriately to the BSDL description of the JTAG circuitry (Lines 27–33).

Listing 4 shows the PDL code that accesses the sensor as we had specified above. The PDL commands are categorized either as *setup* commands or as *action* commands. Setup commands configure the environment for the action commands, and action commands perform actual operations that make the setup commands take effect. For example, commands such as *iProcsForModule*, *iWrite*, *iRunLoop*, and *iRead* are setup commands, whereas *iPDLLLevel* and *iApply* are action commands which are immediately performed.

LISTING 4 SHOWS THE PDL CODE FOR READING THE TEMPERATURE SENSOR

```

01  iPDLLLevel 0;
02  iProcsForModule Sensor;
03
04  iProc Read_Temperature{} {
05      iWrite      en 1;
06      iApply;
07
08      iRunLoop    10  -sck;
09      iApply;
10
11      iRead      temp;
12      iWrite      en 0;
13      iApply;
14  }
15
16  iProcsForModule Chip_ICL;
17  iCall Chip_ICL.Sensor1.Read_Temperature();

```

We will now examine the code in Listing 4 in detail. The `iPDLLevel` command (Line 1) chooses the Level-0 flavor of PDL which can be seen as a sequential set of actions without any flow control, which is sufficient for our simple example. The `iProcsForModule` command (Line 2) specifies the ICL module (see Listing 3) for which the following commands are specified. The `iProc` command is used to specify a PDL procedure, i.e., a group of PDL commands for a given instrument, which can for example identify a certain instrument operation or a test. Using PDL procedures increases readability of the code and simplifies multiple repetitions of a group of commands. In our example, we only specify one procedure for reading the temperature from the sensor (Lines 4–14). To use the sensor, we enable the sensor (Lines 5–6), wait for 10 system clocks (Lines 8–9), and read the temperature while returning the `en` signal back to zero to make the sensor ready for the next read (Lines 11–13). It can be seen that multiple setup commands can be queued and applied concurrently with a single `iApply` command. The `iProcsForModule` command in Line 16 tells the PDL interpreter that the next command is for the `Chip_ICL` module, and the following `iCall` command (Line 17) retargets the access procedures for the temperature sensor, from its terminals (parallel access) to the boundary of the chip, i.e., the JTAG TAP (serial access).

## EXPANDING THE SMALL EXAMPLE

So far, for our small design, ICL and PDL required more lines of code compared to BSDL/SVF. However, for slightly larger designs, ICL/PDL show better maintainability and ease of use, which will be detailed in this section.

### USING MULTIPLE INSTANCES OF OUR TEMPERATURE SENSOR

The first extension to consider is to use two instances of our sensor, as would be the case when there is a need to read temperatures of different areas of a chip. Adding another sensor can be done either by adding its shift register as a separate TDR to the JTAG circuitry, or by connecting the shift registers of the two sensors in series and adding them as a single TDR to the JTAG circuitry, as is shown in Figure 3. Generally speaking, using a separate TDR for each instrument has the following drawbacks: (1) it is not possible to access the instruments concurrently, and (2) addition of each new instrument requires heavy modifications to BSDL (e.g., addition of new instructions, new registers, and possible changes in the IR length) and SVF (e.g., adding an SIR command for each new instrument and change of other SIR commands due to the changes in the IR length). Therefore, in our comparison we only consider the second alternative which is connecting the instrument shift registers in series (Figure 3). The impact of this on the BSDL code (for both BSDL/SVF and ICL/PDL scenarios) is that the length of the custom TDR `INSTR` which is described in Line 33 of Listing 1 as four, should be doubled since now there are two shift registers of length four on the scan path (that is, when the `READINSTR` JTAG command is loaded).

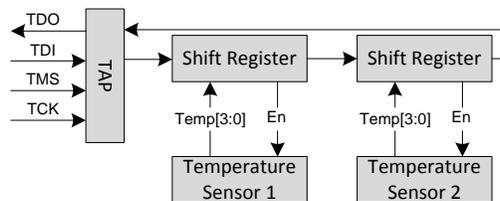


FIGURE 3: USING TWO INSTANCES OF THE SENSOR

Listing 5 shows the updated SVF code for reading the temperature from Sensor 1. Since the length of the scan path for the `INSTR` TDR has changed, any SDR command corresponding to `INSTR` operations should be updated, as can be seen in Line 2 and Line 5 of Listing 5. For reading both sensors, the length of the scan path, specified after each SDR command, does not change but the data bits that are to be scanned will change from 10 (i.e., "00010000") to 11 (i.e., "00010001") to enable both sensors. It can be seen from this example that changes in the

hardware or the access procedure can be tricky to be applied correspondingly to the SVF code, mainly because there is no link between the hardware description (BSDL) and the access procedure description (SVF).

#### LISTING 5 SHOWS THE UPDATED SVF CODE FOR FIGURE 3

```

01 SIR 4 TDI (7);           ! 4-bit IR scan => Loading READINSTR ("0111")
02 SDR 8 TDI (10);         ! 8-bit DR scan => Activating Sensor 1 ("00010000")
03 STATE DRPAUSE;         ! Going to state DRPAUSE
04 RUNTEST 10 SCK ENDSTATE DRPAUSE; ! Waiting for 10 system clocks
05 SDR 8 TDI (0);         ! Shifting out the temperature
06 STATE IDLE;           ! Going to state Run-Test/Idle

```

We will now examine the changes required for the P1687 alternative. Listing 6 shows the updates that are required in the ICL description. Since the same sensor and shift register components are used twice, the only part of code that should be modified is the chip level module that should instantiate two instances of the sensor and shift register modules (Lines 20–33), and add both shift registers to scan path (Lines 37–38).

#### LISTING 6 SHOWS THE UPDATED ICL CODE FOR FIGURE 3

```

.
.
.
19 Module Chip_ICL {
20     Instance TDR1 Of TDR {
21         InputPort en = Tap1.en_TDR;
22         InputPort pi = Sensor1.temp;
23     }
24     Instance Sensor1 Of Sensor {
25         InputPort en = TDR1.po;
26     }
27     Instance TDR2 Of TDR {
28         InputPort en = Tap1.en_TDR;
29         InputPort pi = Sensor2.temp;
30     }
31     Instance Sensor2 Of Sensor {
32         InputPort en = TDR2.po;
33     }
34     AccessLink Tap1 Of STD_1149_1 {
35         BSDL_Entity Chip_ICL;
36         READINSTR {
37             ScanPath { TDR1;
38                       TDR2; }
39             ActiveSignals { en_TDR; }
40         }
41     }
42 }

```

As for the PDL code, no changes are required if we are only interested in reading Sensor 1, otherwise, another iCall command, similar to the one in Line 17 in Listing 4, should be added for Sensor 2.

It seems to us that from a developer's point of view, the PDL code is easier to read and to maintain as the complexity of the access procedures grows. Compared to SVF, PDL features aliases and enumerations which make the PDL code more human readable. Moreover, PDL is a good means for documenting how to use an instrument.

## USING A VARIABLE LENGTH SCAN-PATH

In the previous example shown in Figure 3, the shift-registers for the instruments were always on the scan-path. Having all the instruments on the scan-path is not desirable in chips that have hundreds of instruments. In particular, if an access is made only to one or a subset of those instruments in a given access schedule, having all the instruments on the scan-path might incur a large access time overhead. Furthermore, in certain scenarios such as when instruments are located in different power islands, a single scan-path containing all the instruments will be broken when an island goes to a low-power mode [9]. Therefore, another interesting extension to our design will be to use the P1687-specified SIB module (see the Appendix on P1687 for basic information on SIBs) to add flexibility to the scan-path. Figure 4 shows how SIB components are added to the scan-path for our example.

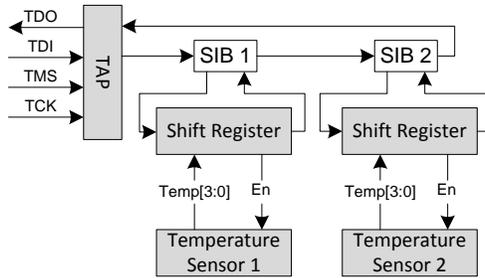


FIGURE 4: ACCESSING THE SENSORS INDIVIDUALLY ON A VARIABLE LENGTH SCAN-PATH

It is important to note that it is not possible to fully describe the new flexible scan path in BSDL, and only the first level, i.e., the two SIBs and not the shift registers, is describable. Therefore, Line 33 of the BSDL code (Listing 1) should be modified such that the length of `INSTR` is set to two, since now the initial scan path for `INSTR` consists of two SIBs. As for the SVF code for reading Sensor 1, this example requires an additional step before accessing the shift registers. In the additional step, we need to open the SIBs on the scan path. Opening the SIBs requires a separate scan sequence (i.e., `SDR` command) in the SVF code, as can be seen in Line 2 of Listing 7, which opens the appropriate SIB to access Sensor 1. The remaining `SDR` commands should also be updated due to both the change in the scan path length and the inclusion of SIB programming bits in the shifted data.

LISTING 7 SHOWS THE UPDATED SVF CODE FOR FIGURE 4

```

01  SIR 4 TDI (7);           ! 4-bit IR scan => Loading READINSTR ("0111")
02  SDR 2 TDI (2);           ! 2-bit DR scan => Opening SIB 1 by shifting "10"
03  SDR 6 TDI (6);           ! 6-bit DR scan => Activating Sensor 1 and keeping SIB 1 open by shifting "000110"
04  STATE DRPAUSE;          ! Going to state DRPAUSE
05  RUNTEST 10 SCK ENDSTATE DRPAUSE; ! Waiting for 10 system clocks
06  SDR 6 TDI (0);           ! Shifting out the temperature
07  STATE IDLE;             ! Going to state Run-Test/Idle

```

Regarding the ICL code modifications, we need to describe the SIB module and apply modifications to the chip level module to instantiate the SIB components and describe the scan path correspondingly. Listing 8 shows the partial ICL code for the design in Figure 4. The Sensor and TDR modules are not modified and therefore are not shown in Listing 8. Here again, we have not shown and port-mapped the clock and control signals for the SIB module. The `PortGroup` command used in the SIB module, guides the PDL interpreter in retargeting the access procedures. Besides instantiating and port-mapping the components required for the design in Figure 4, `ScanPath` (Line 59) is also updated to reflect that the scan-path is now through the SIBs.

Regarding the PDL code, again no change is required. It is expected in P1687 that the PDL interpreter will automatically take care of opening the SIBs on the scan path and performing the scan operations required to access Sensor1. Therefore, retargeting of the sensor access procedures (from the sensor's terminals, through the P1687 network, and to the JTAG TAP) is performed by the PDL interpreter, and a chip-level code—which could be something similar to SVF—is automatically generated. Manual generation and maintenance of SVF code, as can be seen from the above examples, becomes more difficult as the number of instruments and the complexity of the on-chip instrument access network increases. Furthermore, it was stated earlier that SVF is not linked to the BSDL description of JTAG circuitry, and that it is not possible to describe a flexible scan path in BSDL. These two issues make it difficult to automate the generation of the SVF code shown in Listing 7, and as of today there exists no tool for such automation.

## LISTING 8 SHOWS THE UPDATED ICL CODE FOR FIGURE 4

```

19     Module SIB {
20         ScanInPort    si;
21         ScanInPort    fso;
22         ScanOutPort   so { Source    sr;
23                       LaunchEdge Falling; }
24         ScanOutPort   fsi { Source si; }
25         SelectPort    en;
26         ToSelectPort  to_en;
27         ScanRegister  sr { ScanInSource    mux1;
28                           CaptureSource  1'b0;
29                           ResetValue     1'b0; }
30         ScanMux       mux1 sr {
31                           1'b0 : si;
32                           1'b1 : fso;
33                       }
34         PortGroup     tap_side {si, so, en}
35         PortGroup     instrument_side {fso, to_en}
36     }
37
38     Module Chip_ICL {
39         Instance SIB1 Of SIB {
40             InputPort en = Tap1.en_TDR;
41             InputPort fso = TDR1.so;
42         }
43         Instance TDR1 Of TDR {
44             InputPort en = SIB1.to_en;
45             InputPort si = SIB1.fsi;
46             InputPort pi = Sensor1.temp;
47         }
48         Instance Sensor1 Of Sensor {
49             InputPort en = TDR1.po;
50         }
51         /* SIB2, TDR2, and Sensor2 are
52            instantiated and port-mapped similarly
53            .
54            .
55            . */
56         AccessLink Tap1 Of STD_1149_1 {
57             BSDLEntity Chip_ICL;
58             READINSTR {
59                 ScanPath { SIB1; SIB2; }
60                 ActiveSignals { en_TDR; }
61             }
62         }
63     }

```

In the above simple examples, we only used a small subset of ICL and PDL commands. However, ICL goes well beyond BSDL's ability to describe the interface to a variety of instruments from simple ones to complex scenarios such as multiple TAPs on a chip and direct (parallel) interfaces between instruments. PDL also provides functionality of a programming language, required to describe the operation of a variety of instruments including fire-and-forget (such as memory BISTs), read-at-will (such as our sensor), slave-bus-protocol, instruments with fully asynchronous interfaces, and memory-mapped instruments.

To summarize, by using P1687, the modular design approach can be utilized for test, monitoring, and similar purposes as well by (1) reusing the access procedures for a given instrument (which can be among the deliverables for an IP block), and (2) retargeting those access procedures to any higher level of design and to any test access mechanism.

## OPEN ISSUES FOR P1687

In our discussion of accessing an embedded instrument from the JTAG TAP, we mentioned that with the help of P1687 interpreter tools, ICL and PDL facilitate reuse and retargeting of instrument access procedures. Some EDA companies have already announced availability of prototype P1687 tools which will be publicly available close to the ratification of P1687.

In addition to the EDA tools required for interpreting ICL and PDL codes, we expect an additional tool for *data handling*, which is the process of making decisions based on the output data such as test responses extracted from

TDO. If the output data does not match the expected data, different decisions might be made based on the testing scenario. For example, in a manufacturing test scenario, testing might stop on the first fail, but in a debugging scenario, pattern application might be paused to allow data extraction, and be continued afterwards. If SVF is to be used, this handling of *miscompare* data should be done with the help of higher level languages such as C or Python, whereas PDL supports this type of miscompare data handling. In this work we have only considered the Level-0 flavor of PDL which is a sequence of setup and action commands executed serially. However, Level-1 PDL is based on Tool Command Language (TCL) [10] to support looping, branching, modularity, and so. These features of a full programming language—plus some specific Level-1 PDL features such as special instructions to collect and compare output data—make it possible to both describe complex instrument behaviors and to support different testing and debugging scenarios. This data handling is an area which requires support from the EDA industry.

## BIBLIOGRAPHY

- [1] IEEE association, IEEE Std 1149.1-2001, IEEE Standard Test Access Port and Boundary-Scan Architecture, 2001.
- [2] (1999) Serial Vector Format Specification. [Online]. <http://www.asset-intertech.com/support/svf.pdf>
- [3] (1999, Aug) STANDARD TEST AND PROGRAMMING LANGUAGE (STAPL). [Online]. <http://www.jedec.org/standards-documents/results/STAPL>
- [4] IEEE association, IEEE Std 1450.6, IEEE Standard Test Interface Language (STIL) for Digital Test Vector Data-Core Test Language (CTL), 2005.
- [5] IEEE association, IEEE Std 1450-1999, IEEE Standard Test Interface Language (STIL) for Digital Test Vector Data, 1999.
- [6] IEEE association, IEEE Std 1500-2005, IEEE Standard Testability Method for Embedded Core-Based Integrated Circuits, 2005.
- [7] A.L Crouch, "IJTAG: The path to organized instrument connectivity," in *Proceedings of the International Test Conference*, 21-26 Oct. 2007, pp. 1-10.
- [8] IJTAG. [Online]. <http://grouper.ieee.org/groups/1687/>
- [9] Al Crouch. (2011) IEEE P1687 Internal JTAG (IJTAG) taps into embedded instrumentation. [Online]. [http://www.asset-intertech.com/pressroom/whitePapers/IEEE\\_P1687\\_IJTAG\\_Whitepaper.pdf](http://www.asset-intertech.com/pressroom/whitePapers/IEEE_P1687_IJTAG_Whitepaper.pdf)
- [10] John K. Ousterhout and Ken Jones, *Tcl and the Tk toolkit*, 2nd ed. Reading, MA: Addison-Wesley, 2009.

## APPENDIX: A NOTE ON P1687 HARDWARE

The on-chip P1687 network is interfaced to the JTAG TAP by using a special TDR called (Level-0) Gateway module. One interesting feature in the P1687 specification is the concept of a variable length scan-path which can be achieved by using a module called Segment Insertion Bit or SIB for short. A SIB is a 1-bit shift and update register on the scan-path which can be programmed to insert another segment of P1687 scan-path into the current (active) scan-path—hence the name Segment Insertion Bit. It is possible to build a multitude of different hierarchical P1687 networks for the same set of instruments by using SIBs. The Gateway itself can be composed of one or more SIBs. Figure 5 shows a possible SIB implementation, a small P1687 network, and its connection to the JTAG TAP. The shown network also illustrates the concept of a variable length scan-path achieved by using the SIB modules.

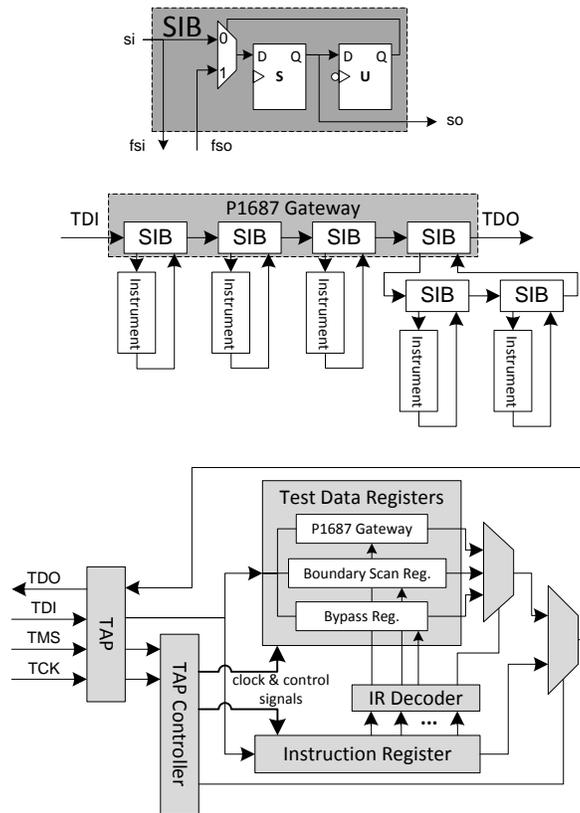


FIGURE 5 SHOWS AN EXAMPLE SIB, AN EXAMPLE P1687 NETWORK, AND HOW THE NETWORK IS INTERFACED TO THE JTAG CIRCUITRY

## AUTHOR BIOGRAPHIES

**Farrokh Ghani Zadegan** is a PhD student in Computer Systems at Embedded Systems Laboratory (ESLAB), Linköping University, Linköping, Sweden. He is interested in design-for-test and fault-tolerance for electronic systems. He is a student member of the IEEE.

*Contact Information:* farrokh.ghani.zadegan@liu.se

**Urban Ingelsson** received his MSc degree from Linköping University, Sweden, in 2005, and his PhD degree from the University of Southampton, United Kingdom, in 2009. Subsequently, he spent two years working as a postdoc at Linköping University. Presently, Ingelsson is an Embedded Systems Engineer at Semcon AB. His research interests include functional safety, requirements engineering, manufacturing test of integrated circuits, diagnosis, and fault tolerance in embedded systems. He is a member of the IEEE.

*Contact Information:* urban.ingelsson@semcon.com

**Gunnar Carlsson** is a Test and DFT strategist with Ericsson. He has 30+ years of experience in the Test and DFT areas. He has been involved in test and DFT spanning from ASIC over boards to network nodes, and over the product life cycle from design verification over production test to testing in the field. He is a member of the IEEE and the IEEE Computer Society.

*Contact Information:* gunnar.carlsson@ericsson.com

**Erik Larsson** received the MSc and the PhD degrees from Linköping University in 1994 and in 2000, respectively. He is currently Associate Professor at the Electrical and Information Technology Department at Lund University. He did his postdoc at the Computer Design and Test Laboratory at Nara Institute of Science and Technology (NAIST), Japan, and sabbatical at NXP Semiconductors, Eindhoven, The Netherlands. His current research interests include test planning for manufacturing test, test during operation (in-situ), scan-chain diagnosis, silicon debug and validation, IJTAG/SJTAG, stacked 3D chip test, fault tolerance for MPSoCs (Multiprocessor System-on-Chip), and property checking in distributed systems (MPSoCs with Network-on-Chips (NoC)). He has more than 130 publications in these areas. He is in the steering committee of Workshop on RTL ATPG & DFT (WRTL) and International Workshop on Reliability Aware System Design and Test (RASDAT), and was program chair of European Test Symposium (2011), vice program chair European Test Symposium (2010), topic chair of European Test Symposium (2007-2009, 2012-2013), Design Automation and Test in Europe (2007-2009), and Great Lake Symposium on VLSI, (2011-2012). He is technical committee member of a dozen of conferences. He is a senior member of the IEEE.

*Contact Information:* erik.larsson@eit.lth.se