A STUDY OF INSTRUMENT REUSE AND RETARGETING IN P1687

Farrokh Ghani Zadegan, Urban Ingelsson, Erik Larsson
Linköping University

Gunnar Carlsson
Ericsson

ABSTRACT

Modern chips may contain a large number of embedded test, debug, configuration, and monitoring features, called instruments. An instrument and its instrument data, instrument access procedures, may be pre-developed and reused and instruments may be accessed in different ways through the life-time of the chip, which requires retargeting. To address instruments reuse and retargeting, 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.

REUSE AND RETARGETING IN TEST STANDARDS

Design reuse is attractive because it reduces the 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 cannot be completely utilized. For example, when a pre-developed test feature is reused and embedded inside another block of logic or another IC, the test patterns developed for that test feature—or in general the access procedures that describe how to use an on-chip feature—must be redeveloped. There are tools available to concatenate bit strings corresponding to the test patterns for each discrete IC on a PCB to perform tests at board-level. Such tools do not yet exist for operating on on-chip test features.

In recent years, the need for reusing the 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 debug 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 Definition 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 neither efficient nor sufficient to describe all types of instruments or the instrument access infrastructure. 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 prove inefficient when we try to retarget the access procedures at the instrument's terminals to the chip terminals. 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. Another

alternative to describe both the instruments and the access procedures is to use the Core Test Language (CTL) of IEEE 1500. However, IEEE 1500 does not describe the chip-level network, and retargeting in IEEE 1500 is provided only from core terminals to core wrapper terminals.

Since current standards are limited in terms of describing instruments, reuse, and retargeting, IEEE P1687 [4] introduces two new languages, Instrument Connectivity Language (ICL) and Pattern Description Language (PDL), to standardize the access and control of on-chip instruments. In this paper, we introduce 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 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 the 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, 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. We make the comparison 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 this comparison, 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 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.

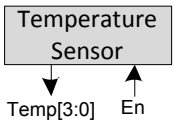

FIGURE 1 SHOWS THE TEMPERATURE SENSOR

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.

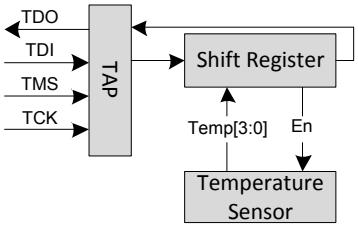

FIGURE 2 SHOWS THE CONNECTION OF THE TEMPERATURE SENSOR TO THE JTAG TAP THROUGH A SHIFT AND UPDATE REGISTER.

HOW IT IS DONE 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 the 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 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
```

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

```
01 entity SingleInstrumentChip 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 SingleInstrumentChip : 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 SingleInstrumentChip : entity is 4;
17 attribute INSTRUCTION_OPCODE of SingleInstrumentChip : entity is
18                         "BYPASS       (1111, 0000)  , " &
19                         "EXTEST       (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 SingleInstrumentChip : entity is "0001";
27 attribute INSTRUCTION_DISABLE of SingleInstrumentChip : entity is "HIGHZ";
28 attribute INSTRUCTION_GUARD   of SingleInstrumentChip : entity is "CLAMP";
29
30 attribute REGISTER ACCESS of SingleInstrumentChip : entity is
31                         "BOUNDARY (EXTEST, INTEST, SAMPLE)," &
32                         "BYPASS   (BYPASS, HIGHZ, CLAMP)," &
33                         "INSTR[4]    (READINSTR)" ;
34                                        .
35                                        .
36                                        .
37 end SingleInstrumentChip;
```

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 accesses is specified as `INSTR[4]` (Line 33). This 4-bit TDR represents the shift and update register in Figure 2.

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.

How It Is Done by Using ICL/PDL

Although ICL has some overlap with BSDL, it is not supposed to replace it. In fact, the JTAG TAP and boundary scan related circuitry will still be in the scope of JTAG. 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 the 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 TDR_Sensor module.

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_Sensor {
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 SingleInstrumentChip_ICL {
20          Instance TDR_Sensor_1 Of TDR_Sensor {
21                InputPort en = Tap1.en_TDR_Sensor;
22                InputPort pi = Sensor1.temp;
23                }
24          Instance Sensor1 Of Sensor {
25                InputPort en = TDR_Sensor_1.SR[0];
26                }
27          AccessLink Tap1 Of STD_1149_1 {
28                BSDL_Entity SingleInstrumentChip_ICL;
29                READINSTR {
30                      ScanPath {  TDR_Sensor_1; }
31                      ActiveSignals { en_TDR_Sensor ;}
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 contains the I/O ports shown in Figure 2, i.e. the *en* input terminal and the *temp* output terminal. The description of the shift register is more detailed since it contains both connection to the scan path (Lines 7-9) and connection to the sensor (Lines 10-12). The main component inside the TDR_Sensor 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_Sensor module.

The chip level module, i.e. SingleInstrumentChip_ICL, puts it all together by instantiating 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 iTarget, iWrite, iRunLoop, and iRead are setup commands, whereas iPDLLevel and iApply are action commands which are immediately performed.

LISTING 4 SHOWS THE PDL CODE FOR READING THE TEMPERATURE SENSOR

```
01    iPDLLevel 0;
02    iTarget 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    iTarget     SingleInstrumentChip_ICL;
17    iCall SingleInstrumentChip_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 iTarget 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 series of PDL commands for a given instrument as a group, which simplifies multiple repetitions of those commands and makes it possible to call them at higher levels. 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 iTarget command in Line 16 tells the PDL interpreter that the next command is for the SingleInstrumentChip_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 (sequential 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. 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).

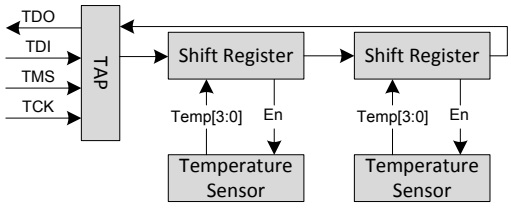

FIGURE 3: USING TWO INSTANCES OF THE SENSOR

Listing 5 shows the updated SVF code for reading the temperature from Sensor1. Since the length of 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 Sensor1 ("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 PDL CODE FOR FIGURE 3

```
                .
                .
                .
19   Module SingleInstrumentChip_ICL {
20         Instance TDR_Sensor_1 Of TDR_Sensor {
21               InputPort en = Tap1.en_TDR_Sensor;
22               InputPort pi = Sensor1.temp;
23               }
24         Instance Sensor1 Of Sensor {
25               InputPort en = TDR_Sensor_1.SR[0];
26               }
27         Instance TDR_Sensor_2 Of TDR_Sensor {
28               InputPort en = Tap1.en_TDR_Sensor;
29               InputPort pi = Sensor2.temp;
30               }
31         Instance Sensor2 Of Sensor {
32               InputPort en = TDR_Sensor_2.SR[0];
33               }
34               AccessLink Tap1 Of STD_1149_1 {
35               BSDL_Entity SingleInstrumentChip_ICL;
36               READINSTR {
37                     ScanPath {  TDR_Sensor_1;
38                                 TDR_Sensor_2; }
39                     ActiveSignals { en_TDR_Sensor ;}
40                     }
41               }
42         }
```

As for the PDL code, no changes are required if we are only interested in reading Sensor1, otherwise, another iCall command, similar to the one in Line 17 in Listing 4, should be added for Sensor2.

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 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 which is not desirable in chips that have hundreds or more instruments, in particular if an access is made only to one or a subset of those instruments in a given access schedule. In such cases having all the instruments on the scan-path might unnecessarily 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 [5]. 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.

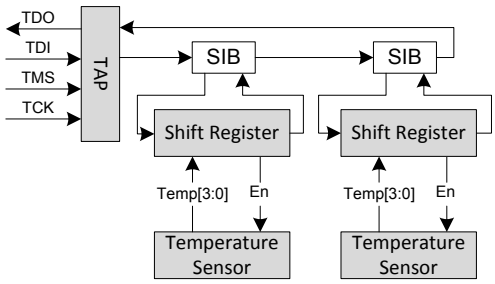

FIGURE 4: ACCESSING THE SENSORS INDIVIDUALLY

Again, 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 Sensor1, 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 Listing 7, which opens the appropriate SIB to access Sensor1.

Regarding the ICL code modifications, we need to describe the SIB module and apply modifications to

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 SIB1 by shifting "10"
03    SDR 6 TDI (34);                   ! 6-bit DR scan => Activating Sensor1 by shifting "100010"
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
```

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_Sensor 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.

LISTING 8 SHOWS THE UPDATED ICL CODE FOR FIGURE 4

```
                        .
                        .
                        .
19  Module SIB {
20        ScanInPort        si;
21        ScanInPort        fso;
22        ScanOutPort       so {  Source      SIB;
23                                LaunchEdge Falling; }
24        SelectPort        en;
25        ToSelectPort      to_en;
26
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 SingleInstrumentChip_ICL {
39        Instance sib1 Of SIB {
40              InputPort   en = Tap1.en_TDR_Sensor;
41              InputPort   fso = TDR_Sensor_1.so;
42              }
43        Instance TDR_Sensor_1 Of TDR_Sensor {
44              InputPort en = sib1.to_en;
45              InputPort si = sib1.so;
46              InputPort pi = Sensor1.temp;
47              }
48        Instance Sensor1 Of Sensor {
49              InputPort en = TDR_Sensor_1.SR[0];
50              }
51        /* sib2, TDR_Sensor_2, and Sensor2 are
52           instantiated and port-mapped as above
53                                  .
54                                  .
55                                  .          */
56        AccessLink Tap1 Of STD_1149_1 {
57              BSDL_Entity SingleInstrumentChip_ICL;
58              READINSTR {
59                    ScanPath {  sib1; sib2; }
60                    ActiveSignals { en_TDR_Sensor ;}
61                    }
62              }
63        }
```

Regarding the PDL code, again no change is required. It is expected in P1687 that the PDL interpreter will take care of opening the SIBs on the scan path and performing the scan operations required to access Sensor1, and therefore, retargeting of the access procedure of the sensors (from the sensor's terminals, through the P1687 network, and to the JTAG TAP) is automatically performed.

From the above examples, it is becoming evident that without the help of a procedural description language such as P1687's PDL, developing access procedures for on-chip instruments from the JTAG TAP (i.e. from an outside point of view) by using SVF is becoming difficult to maintain as the complexity and size of the on-chip instrument network increases.

In the above simple examples, we only used a small subset of ICL commands. However, ICL goes well beyond BSDL's ability to describe the interface to a variety of instruments from simple ones to complex instruments such as MBIST engines, memories, and SERDESs, as well as scenarios such as multiple TAPs on a chip and direct (parallel) interfaces between instruments.

To summarize, by using P1687, the modular design approach can be utilized for test, monitoring, and etc. 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.

CHALLENGES IN ADOPTING P1687

In our discussion of accessing an embedded instrument from the JTAG TAP, we mentioned that ICL and PDL facilitate reuse and retargeting of instrument access procedures. In addition to EDA tools required for interpreting the ICL and PDL codes, we expect two additional tools, (1) a network construction tool for automatic generation of optimized ICL code which will address potential overheads, and (2) a SIB handling tool [5] for optimized operation of SIBs used in a variable length scan path.

As for the P1687 network construction tool, it should be noted that one should take into account the

instrument access time overhead, which might be considerable for large designs. In [6] the access time overhead is discussed for the scenario where the JTAG TAP is used to access the on-chip instruments. For example, chaining all instrument shift registers into a long scan-path, is not the best P1687 network design practice due to the prohibitively high access time overhead [6]. Moreover, such a long scan path is vulnerable to manufacturing defects such as stuck at faults in the chain that could potentially render the whole chain useless. P1687 specifies the SIB module to be used to form a variable length scan path so that it becomes possible to include/exclude instruments as needed. However, a long flat [6] scan-path might also prove inefficient when some instruments are to be accessed more frequently than the others—a situation that can be handled, again by using SIBs to add hierarchical levels to the P1687 network [7]. Hierarchical network design may also solve the issue with the instruments being in different power islands [5]. Such a network construction tool is not yet available from the industry.

Regarding the SIB handling tool, given a well designed P1687 network with a variable length scan path, it is still required that based on the power constraints, resource conflicts, etc., SIBs are operated (i.e. opened and closed) in a way that the constraints are satisfied and the access time overhead is minimized. This smart handling [5] of SIBs is another area which requires support from the EDA industry.

BIBLIOGRAPHY

[1] 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] IJTAG. [Online]. http://grouper.ieee.org/groups/1687/

[5] 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

[6] F. Ghani Zadegan, U. Ingelsson, G. Carlsson, and E. Larsson, Access Time Analysis for IEEE P1687, doi: 10.1109/TC.2011.155, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5999657&isnumber=4358213.

[7] F.G. Zadegan, U. Ingelsson, G. Carlsson, and E. Larsson, "Design automation for IEEE P1687," in *proc. Design, Automation & Test in Europe Conference & Exhibition (DATE) 2011*, Grenoble, 14-18 March 2011.

APPENDIX

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 by using SIBs. The Gateway itself can be composed of one or more SIBs. Figure 5 shows 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.

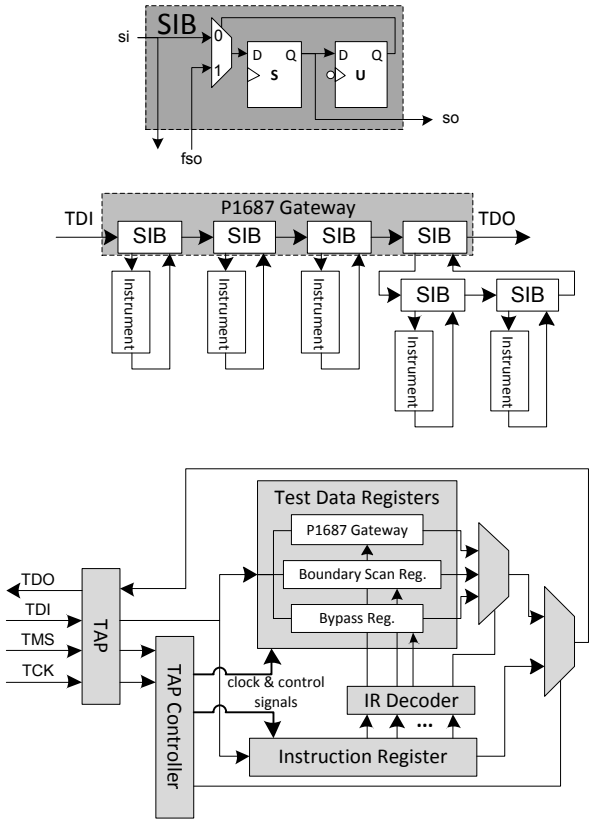

FIGURE 5 SHOWS A SAMPLE JTAG CIRCUITRY, A SAMPLE P1687 NETWORK, AND HOW THEY ARE INTERFACED