From Design-for-Test to Design-for-Debug-and-Test: Analysis of Requirements and Limitations for 1149.1

Gustavo R. Alves¹,² and J. M. Martins Ferreira²

¹ISEP / DEE
Rua de S. Tome
4200 Porto – PORTUGAL

²FEUP / DEEC
Rua dos Bragas
4000 Porto – PORTUGAL

Abstract

The increasing complexity of VLSI circuits and the reduced accessibility of modern packaging and mounting technologies restrict the usefulness of conventional in-circuit debugging tools, such as in-circuit emulators for microprocessors and microcontrollers. However, this same trend enables the development of more complex products, which in turn require more powerful debugging tools. These conflicting demands could be met if the standard scan test infrastructures now common in most complex components were able to match the debugging requirements of design verification and prototype validation. This paper analyses the main debug requirements in the design of microprocessor-based applications and the feasibility of their implementation using the mandatory, optional and additional operating modes of the standard IEEE 1149.1 test infrastructure.

1. Introduction

Conventional in-circuit debugging tools for microprocessor-based systems, such as In-Circuit Emulators (ICEs), ROM emulators, logic analysers and pre-processor modules are negatively affected by the reduced accessibility of modern packaging and mounting technologies [1]. However, the development of increasingly complex products and the growing concern on quality assurance, require more powerful debugging tools and methodologies for design verification and prototype validation. These conflicting demands could be met if the standard scan test infrastructures now common in most complex components were able to match the debugging requirements of design verification and prototype validation.

This paper analyses the main debug requirements in the design of microprocessor-based applications and the feasibility of their implementation using the mandatory, optional and additional operating modes of the standard Boundary Scan Test (BST) infrastructure [2]. The scope of the present analysis is restricted to applications whose functionality is defined by a program stored in memory, run by a single processor.

Section 2 describes typical debugging procedures enabled by ICEs, at the PCB level, and briefly refers the limitations due to its reduced accessibility. Section 3 contains a brief characterisation of mandatory and optional BST instructions, and a more extended description of the enhanced modes now commonly available in BST-compatible devices belonging to the SCOPE™ and SCAN™ families – with a special emphasis on the Digital Bus Monitor (DBM). Section 4 analyses the debugging procedures that can be implemented using only the mandatory BST instructions and the limitations implied by their basic operating modes. Section 5 describes how an enhanced BST infrastructure can help to implement the required debugging procedures, overcoming (part of) the previously identified limitations. Section 6 discusses the implementation in the BST infrastructure of additional features addressing the identified debugging procedures, this way justifying the expression “design-for-debug-and-test”. Section 7 concludes this paper with the final remarks and possible spin-offs.

Original contributions of this paper include:
1. Characterisation of the board-level debugging capability offered by the BST standard.
2. Identification of additional debugging procedures made possible by the high pin coverage enabled by BST.
3. Identification of additional BST features (supported by user defined instructions) that would improve its board-level debugging capability.
4. Proposed use of Texas Intruments’ DBM for debugging board-level applications.

2. Debugging procedures

The typical debugging procedures enabled by ICEs are briefly characterised as follows:
1. Displaying internal / external processor variables (memory, registers, peripherals, etc.)
2. Single stepping
3. Breakpoint insertion (by data, address, etc.)
4. Real-time sampling of signals

Internal variables correspond to the contents of registers and memories internal to the board processor. External variables correspond to the contents of memories external to the processor. Internal registers of peripherals interacting with the processor are also considered external variables. ICEs display internal / external processor variables by accessing them (requires observability) when the processor is in an ‘idle’ mode, typically after executing a single-step or a breakpoint (BP). Single stepping usually corresponds to executing the processor application instruction-by-instruction. If the ICE is able to provide its own clock, other refined levels of execution are also possible, for instance executing a single clock cycle – processors usually take several clock cycles to execute one instruction. BP insertion corresponds to executing the processor application until a certain condition is found. The condition may refer to an internal or external processor variable depending on the ICE detection capabilities. Concurrent detection requires the ICE to observe in real-time (processor running at normal operating speed), the current value of the variable in order to evaluate the BP condition. Non-concurrent detection corresponds to executing a single-step and evaluating the BP condition, in a continuous loop, until the condition is found true. Some ICEs (usually the most expensive ones) allow real-time sampling of the signals present at the processor pins. This debugging procedure is essential for detecting / diagnosing problems that only occur in real-time [3].

Limitations associated with the use of ICEs are due to its accessibility requirements – an ICE replaces the board processor by connecting a probe to a dedicated emulator socket or through the processor socket (either by replacing the processor itself or by placing a probe on the top of it). Only a small number of pins, considering the relation between the total number of pins on the board and the number of processor pins, is accessed by the ICE – thus limiting concurrent evaluation of BP conditions and real-time sampling.

3. 1149.1: basic & enhanced operating modes

The 1149.1 Std. defines three mandatory instructions: EXTEST, SAMPLE/PRELOAD, and BYPASS. EXTEST allows the state of all signals driven from the component’s output pins to be completely defined, in an intrusive mode, by the data held in the BS register, as illustrated in figure 1. The state of all signals received at the component’s input pins is loaded into the BS register on the rising edge of Test Clock (TCK). SAMPLE/PRELOAD allows the state of all signals driven or flowing through component pins (input or output) to be loaded into the BS register on the rising edge of TCK. SAMPLE allows a snapshot to be taken of the data flowing from the component pins to the on-chip logic or vice-versa, without interfering with the normal operation of the component’s function, as illustrated in fig. 2. This provides a verification mechanism of the interaction between components during normal board functional operation. PRELOAD allows an initial data pattern to be placed at the latched parallel outputs (POs) of BS register cells prior to the selection of another BS test-operation, for example before loading the EXTEST instruction. BYPASS places the 1-bit bypass register in a component’s TDI-TDO path, thus reducing the total size of the boundary scan chain where the component is inserted.

![Fig. 1: Control mode in a BS cell for EXTEST support.](image1)

![Fig. 2: Observation mode in a BS cell for SAMPLE/PRELOAD support.](image2)

The 1149.1 Std. also considers both public and private optional BST instructions. Six optional public instructions are described in [2]: INTEST, RUNBIST, CLAMP, IDCODE, USERCODE and HIGHZ. INTEST allows static (slow-speed) testing of the on-chip logic, with each test pattern and response being shifted through the BS register. This instruction requires that the on-chip logic can be operated in a single-step mode, where the circuitry moves one step forward in its operation each time shifting of the BS register is completed. RUNBIST provides the component purchaser with a means of running a user-accessible self test function within the component as a
result of a single instruction. CLAMP enables the state of the signals driven from component pins to be determined by the BS register contents, while selecting the bypass register. IDCIME and USERCODE select the component’s identification register. IDCIME provides information on the base component while USERCODE provides information on the particular programming of an off-board programmable component. HIGHZ enables all component’s output pins to be placed in an inactive-drive state (e.g., high impedance), while selecting the bypass register. During testing or emulation, this instruction is entered into the processor with the result that its outputs enter the inactive drive state. An ICE can then be enabled to drive the connections into the all system. In addition to the public instructions defined in the standard, a design may offer other public instructions to allow the device purchaser access to design-specific features. Instructions that provide access to internal scan chains or other component resources, typically fall under this designation.

Components Off-the-Shelf (COTS) with an enhanced BST infrastructure that support some of the optional operating modes previously described, along with additional debug oriented operating modes, are now widely available. For instance, the TI SCOPE™ testability IC family contains generic devices that support the instructions HIGHZ, CLAMP, IDCIME, and other public optional instructions allowing the user to perform several test and debug oriented functions [4]. These include:

- Boundary run test - the device operates in a test mode. The test operation specified in the Boundary Control Register (BCR) is executed during Run-Test/Idle.
- Boundary read - The value in the BS register remains unchanged during Capture-DR. This instruction is useful for inspecting data after certain test operations.
- Boundary self-test - the device operates in a normal mode. All BS cells capture the inverse of their current values. In this way, the contents of the shadow latches can be read out to verify the integrity of both elements of the BS register.
- access to the BCR - this operation must be performed before a boundary-run test operation to specify which test operation is to be executed. The five test operations decoded by the BCR are:
  - sample inputs/toggle outputs - Data appearing at the device I/O pins is captured in the associated BS cells on each rising edge of TCK. Data in the shift-register elements is toggled on each rising edge of TCK and applied to the associated device I/O pins on each falling edge of TCK.
  - PRPG - Pseudo-random pattern is generated in the selected BS cells on each rising edge TCK and applied to the associated device output pins on each falling edge TCK.
  - PSA - Data appearing at the selected device input-mode I/O pins is compressed into a n-bit parallel signature in the shift-register elements of the selected BS cells on each rising edge of TCK.
  - simultaneous PSA and PRPG, and
  - simultaneous PSA and binary count up – while performing PSA, an n-bit binary count-up pattern is generated in the BS cells on each rising edge of TCK and applied to the associated device I/O pins on each falling edge of TCK.

The SN74ACT8994 DBM [5] is a member of the TI SCOPE™ family, designed to monitor and/or store the values of a digital bus up to 16 bits in width. It resides in parallel with the bus being monitored, as illustrated in figure 3. The DBM can be operated in the off-line mode or the on-line mode. In the off-line mode, the device performs test operations independent of system conditions. Off-line test operations include PSA on the contents of RAM and external test. In the on-line mode, the DBM can be configured to perform test operations that are initiated based on system conditions and that operate synchronously to a logical combination of one or more system clocks. The device allows sample, storage, and/or PSA operations to be performed according to one of eight protocols. Data at the D-input pins can be stored in a scannable RAM. Compare patterns, which can be stored in the event-qualification module (EQM), allow the user to define specific values of the 16-bit bus for which the test operations are to be performed. The EQM also contains the state machines for the eight protocols that include various start/stop, start/pause/resume, and do-while algorithms. These protocols operate synchronously to the clock signal generated - from one of 32 different logical combinations of CLK1, CLK2, CLK3, and TCK - by the programmable clock interface (PCI). The user configures the PCI through the control register (CTLR). The 1024-word by 16-bit RAM and the EQM register files can be serially accessed using BST-compatible read and write instructions. However, direct memory access (DMA) instructions also are provided to speed transfer of large amounts of data to and from the RAM and EQM.

![Fig. 3: Connecting the DBM to the processor busses.](image-url)
4. Debugging through 1149.1 basic operating modes

This section considers what debugging procedures can be implemented using only the mandatory BST instructions and the limitations implied by their basic operating modes:

1. Displaying internal / external variables - displaying internal variables is considered not feasible through the basic operating modes of the BST infrastructure, since only a very restricted observability of internal variables is possible (if no instructions force a variable into the data bus, observability of its value will not be possible). Displaying external variables is considered feasible. If no lower limit exists for the processor functional clock (f_clock) frequency, any location in the memory or I/O space can be captured by using the EXTEST mode to set up the necessary conditions (in the address, data and control bus) and then capturing the contents of the data bus (EXTEST should be preceded by SAMPLE / PRELOAD to capture all input pin values, such that the same input conditions could be preserved for each vector inserted during the EXTEST mode).

2. Single step - this debugging procedure may be implemented in two ways:
   • EXTEST mode is used to substitute the following instruction by a “restart” routine. This routine could then be used for writing the values of all variables into the data bus - in a non-destructive way. In certain cases this may not be feasible, such as when the program memory corresponds to a ROM or is internal to the processor.
   • If no lower limit exists for the f_clock frequency, single-step could be accomplished by issuing the required number of f_clock cycles through the f_clock pin BS cell (in EXTEST). This action requires shifting two vectors for issuing a single clock cycle on f_clock.

3. Breakpoint insertion - this debugging procedure may be implemented in two ways:
   • Simulation determines when/where the BP condition takes place and the corresponding instruction location is filled with the beginning of a “restart” routine. In certain cases this may not be feasible, such as when the required condition depends on external events which cannot be predicted by simulation.
   • If no lower limit exists for the f_clock frequency, SAMPLE / PRELOAD mode could be used to run successive capture / shift cycles (between each two consecutive f_clock cycles), such that the detection of the required breakpoint condition could be checked at each f_clock cycle. If the required condition is not present another f Clock cycle is issued. In relation to an ICE, the BP condition would not be restricted to the processor pins, but actually extended to all pins belonging to BST-compatible devices.

4. Real-time sampling of signals - SAMPLE / PRELOAD mode could be used to run successive capture / shift cycles, between each two consecutive f_clock cycles (a mask could be used to identify what pins are not required for visualisation, in a manner similar to masking test vector response bits with unknown values). This technique would be similar to the one described in [6].

5. Optional modes: Enhanced debugging procedures

This section considers how an enhanced BST infrastructure can help to implement the required debugging procedures, overcoming (part of) the limitations identified previously. Implementation of some of the identified debugging procedures, using COTS from the TI’s SCOPE™ family, may also be found in [7]:

1. Displaying internal / external variables - If no lower limit exists for the f_clock frequency, and if the BST infrastructure allows access to internal test data registers enabling observability of any internal register or memory position, any internal variable may be observed between any two f_clock cycles. Displaying external variables is considered feasible, following the procedure described in the previous section. Variables corresponding to external memory locations (forming contiguous blocks) can be captured / shifted out using the binary count up mode of SCOPE™ devices. The BS cells connected to the address bus are first loaded with a pattern that selects a read operation on the selected memory. Every TCK cycle on the Run-Test/Idle state, the vector present on the address bus is incremented by one, and the data present on the current pointed memory location may be captured / shifted out. Savings occur because no addresses are shifted through the BS chain.

2. Single step - this debugging procedure may be implemented in three ways:
   • INTEST mode is used to single-step the processor (INTEST should be preceded by SAMPLE/PRELOAD to capture all input pin values, such that the same input conditions could be preserved for each vector inserted during the INTEST mode). If an input signal changes while a single-step operation is in progress, the new value will not be acknowledged. The values present at the processor output pins should be defined by the data held in the BS register and not placed in an inactive drive state (as allowed by the standard).
   • If no lower limit exists for the f_clock frequency, single-step could be accomplished by issuing the required number of f_clock cycles through the f_clock pin BS cell (in EXTEST). This action would now just
require applying two TCK cycles in the Run-Test/Idle state, for issuing a single clock cycle on f_clock, if this signal was provided by a device supporting the optional sample inputs/toggle outputs instruction.

- The Event Qualification Output (EQO) line of the DBM is connected to a processor “hold” line or to a “halt” line on the clock source circuit. The three clock input pins of the DBM are connected to the processor control bus, and both the PCI and EQM are configured so as to perform the following action: each time a new instruction is fetched from the (external) program memory the EQO line is activated, causing the processor to stop its activity after executing the present instruction [8].

3. Breakpoint insertion - this debugging procedure may be implemented in two ways:
   - If no lower limit exists for the f_clock frequency, the SAMPLE mode and the access to internal scan chains using the TAP could be used to run successive capture / shift cycles (between each two consecutive f_clock cycles), such that the detection of the required BP condition – now extended to internal variables - could be checked at each f_clock cycle. If the required condition is not present another f_clock cycle is issued.
   - The DBM EQO line is connected to a processor “hold” line or to a “halt” line on the clock source circuit. The three clock input pins of the DBM are connected to the processor control bus, and the 16-bit data input bus is connected to the processor address and / or data bus (if the 16-bit data is not enough for both processor busses any number of DBMs can be cascade using an external circuit to connect the several EQI / EQO pins). The PCI and EQM are configured so as to perform the following action: the BP condition (and the associated mask, if required) is loaded into one of the EQM registers. The other EQM register is loaded with a vector that selects an appropriated protocol for monitoring the values present at the DBM 16-bit data input bus. Evaluation of the BP condition is made concurrently with the processor application being executed at full speed. When the condition is evaluated as true, the EQO signal (or the several EQO signals – if several DBMs are cascaded) is activated, and the application is “halted”. However, the BP condition is now limited to lines connected to the DBM(s) data input bus(es).

4. Real-time sampling of signals – the DBM can be used to sample / store in real-time the values present at its 16-bit data input bus, according to one of the eight available protocols. The values are stored in its 1024-word memory and can be later shifted out, using DMA operations, for observing the values present on the signals. However, real-time sampling is restricted to signals connected to the DBM(s) data input bus(es).

6. Additional 1149.1 features for improved debugging

This section considers how the implementation of the debugging procedures could be enhanced / optimised if additional features were available in the BS infrastructure:

1. Displaying internal / external variables – Display of all internal variables of the processor (down to the gate level) could be enabled by an access matrix [9]. The pins used for node observation (via I/O MUXes) can not exhibit their normal values e.g., input pins are used as address lines (for the access matrix) and thus can not feed the system logic for normal operation and output pins are used as data lines (from the access matrix) and thus can not exhibit the values coming from the system logic normal operation. This limitation could be eliminated if the BS cells associated to the I/O pins were slightly modified to the structure illustrated in fig. 4, for providing access through the BST infrastructure to the access matrix, namely through a 3-input mux on the capture stage, and a two-input / two-output mux / demux on the update stage of the BS cell. This feature would enable circuit operation and node value capturing simultaneously.

![Fig. 4: Modified BS cells for devices with an access matrix.](image-url)

For an input pin, a new entry on the multiplexer feeding the capture stage, selected by a new user-defined optional instruction, would allow the BS cell (left side of figure) to capture in a non-intrusive mode the value coming from the corresponding data line. The output multiplexer on the BS cell would also contain a new entry, to enable the function performed by the I/O MUX associated with the access matrix. For an output
pin, the output multiplexer on the associated BS cell (right side of figure) would contain a new output line feeding the corresponding address line. The value present on the update stage could, this way, be fed to the system logic input or to the address line.

2. Single step - Inserting a “hold” input in the instruction decode and execution control unit, this input being accessible through the BST infrastructure. Automatically changing from functional mode to test mode by configuring the last micro-instruction (for processors designed with this technique) to prevent the processor from performing any further operation. Resuming normal function would correspond to changing from test mode to functional mode, using a new optional instruction.

3. Breakpoint insertion – this debugging procedure may be implemented in two ways:
   • Inserting a “hold” input in the instruction decode and execution control unit, this input being accessible through the BST infrastructure. Automatically changing from functional mode to test mode when a specified BP condition is detected [10]. This feature would require internal registers containing the expected data and mask information, and the possibility of associating these register bits to those nodes where the required condition is expected to take place. The output of the comparison block would then (when active) suspend the f_clock to hold the current circuit condition. Resuming normal function would correspond to changing from test mode to functional mode, using a new optional instruction.
   • Configuring the BS register to detect conditions related to vectors appearing at device I/O pins. The condition type is selected by a control register, accessible by an optional BST instruction. When a certain condition is evaluated as true, a Condition Detected Output pin (CDO) is activated. The condition may either comprehend an expected vector / mask or a limit A / limit B. These values are initially stored in the capture and update stages of the BS register. The condition is evaluated while the TAP controller is in Run-Test/Idle and a new optional instruction is active. Conditions referring to two or more devices may be evaluated by using a Condition Detected Input (CDI) pin for cascading several devices. These extra pins (CDI, CDO) are required for implementing the BP condition detection through the BS register. The BS cell architecture presented in fig. 5 supports this additional debugging mode. Signals PI, PO, SI, SO, Mode, and Shift are equal to the ones that exist on a BS cell able to implement the mandatory BST instructions. F_n evaluates the partial contribution of a single BS cell to the evaluation of a condition referring to the entire BS register. Signal “from F_{n+1}” corresponds to the F_n output from the previous BS cell. On the BS cell closest to TDI, this signal is either connected to CDI or to a permanent value. Signal “to F_{n-1}” corresponds to the F_n output that connects to the next BS cell. On the BS cell closest to TDO, this signal is connected to CDO. This same signal would then (when active) suspend the f_clock to hold the current circuit condition.

4. Real-time sampling of signals - this debugging procedure may be implemented in three ways:
   • Configuring the BS register to acquire in real-time two contiguous vectors present at the device I/O pins. The two contiguous vectors are stored in the capture and update stages of the BS cell, while the TAP controller is in Run-Test/Idle and a new optional instruction is active. The value present at PI is captured on the TCK rising edge, and the value present at the capture stage is captured by the update stage on the TCK falling edge, as illustrated in fig. 6. To shift out the stored sequence, the TAP controller is placed in Shift-DR (when moving through Capture-DR, the capture stage does not change its current value) for shifting out the vector stored in the capture stage of the BS register. To shift out the vector stored in the update stage the TAP controller is moved from Shift-DR (through Pause-DR and Exit2-DR) to Shift-DR. In Exit2-DR the capture stage captures the vector stored in the update stage. The second vector is shifted out, while the TAP controller is in Shift-DR (for the second time). The BS cell architecture presented in fig. 7 supports this additional debugging mode. Signals PI, PO, SI, SO, and Mode are equal to the ones that exist on a BS cell able to implement the mandatory BST instructions.
   • Configuring the BS register to store sequences of two contiguous vectors until a certain condition is externally detected. The system condition is monitored at CDI while the TAP controller is in Run-Test/Idle, a new optional instruction is active and an auxiliary FSM is in state Monitor_condition. While both the TAP controller and the FSM are in these states, and a logic ‘0’ is present at CDI (meaning that the condition is false or not detected), the capture stage captures the value

![Fig. 5: A modified BS cell capable of detecting conditions on device pins.](image-url)
present at the device I/O pins and the update stage captures the value present at the capture stage. When the condition becomes true (corresponding to the application of a logic '1' at CDI) the FSM enters state End_of_sequence, on the next TCK falling edge, and the capture / store activity is ceased. Fig. 8 illustrates the timing diagram associated with this additional operating mode. To shift out the stored vectors it is necessary to follow the previously described process. The BS cell architecture presented in fig. 7 also supports this additional operating mode. Signals PI, PO, SI, SO, and Mode are equal to the ones that exist on a BS cell able to implement the mandatory BST instructions.

Fig. 6: Timing diagram of capture / store two contiguous vectors in the BS register.

Fig. 7: A modified BS cell for storing two contiguous vectors, captured at the device pins.

- Configuring the BS register to store sequences of two contiguous vectors after a certain condition is internally / externally detected. The system condition is monitored while the TAP controller is in Run-Test/Idle, a new optional instruction is active and the auxiliary FSM is in Monitor_condition. The functionality described for the new optional instruction supporting the implementation of BPs is used for monitoring internal conditions. After a condition is detect, CDO is activated and the FSM enters state Store_sequence_I on the TCK falling edge. While at this state the capture stage captures the value present at the device I/O pins and the update stage captures the value present at the capture stage. On the next TCK falling edge, the FSM enters state Store_sequence_II. While at this state the capture stage captures the value present at the device I/O pins and the update stage maintains its previous value. The FSM moves into End_of_sequence on the next TCK falling edge, and the capture / store activity is ceased. Fig. 9 illustrates the timing diagram associated with this additional operating mode. To shift out the stored vectors it is necessary to follow the previously described process. The BS cell architecture presented in fig. 10 supports this optional operating mode. Signals PI, PO, SI, SO, and Mode are equal to the ones that exist on a BS cell able to implement the mandatory BST instructions.

Fig. 8: Timing diagram of capture / store two contiguous vectors until condition.

Fig. 9: Timing diagram of capture / store two contiguous vectors after condition.

Fig. 10: A modified BS cell capable of storing two contiguous vectors after a certain condition.
7. Conclusion

This paper analysed the typical debugging procedures enabled by ICEs (displaying internal / external variables, single stepping, BP insertion, and real-time sampling of signals) and the feasibility of their implementation using the mandatory, optional and additional operating modes of the BST infrastructure.

Mandatory operating modes are provided by the EXTEST, SAMPLE/ PRELOAD and BYPASS instructions. Implementation of the debugging procedures using these basic operating modes has several limitations associated: no lower limit for f_clock, forcing external variables into the processor data bus (using EXTEST); no access to internal variables; BP conditions determined by simulation or by an external control application (using SAMPLE/PRELOAD); etc.

Optional operating modes are provided by the INTEST, RUNBIST, CLAMP, IDCODE, USERCODE, and HIGHZ instructions defined in the IEEE 1149.1 Standard, and by other public optional instructions supported by COTS. Using optional and enhanced operating modes for implementing debugging procedures, eliminates part of the previously identified limitations: access to internal variables is considered feasible if an optional BST instruction providing access to internal scan chains is available; INTEST and the optional binary count up and sample inputs / toggle outputs modes available in SCOPE™ devices speed up some of the debugging procedures; etc. The guidelines to use the DBM as a debuggability building block (DBB) for BP insertion and real-time sampling of signals are important contributions to the implementation of the debugging procedures through the BST infrastructure. Limitations associated with the DBM include overhead and restricted access to only those lines that are connected to its 16-bit data input bus. The possible inclusion of DBM-like structures internal to the processor (a DBM megablock) would certainly be a powerful DBB, enabling debugging procedures even more powerful than those proposed [11].

Additional operating modes (following the expression design-for-debug-and-test) are provided by a proposed set of new user-defined optional instructions for extending BP conditions and real-time sampling of signals (without, until and after condition) to all I/O pins of devices supporting these debug-oriented operating modes. Inclusion of an access matrix, accessible through a new optional BST instruction, allows observability down to the gate-level of internal variables.

Next steps following the requirements and solutions presented in this paper include:

1. Implementation of a prototype IC supporting additional design-for-debug-and-test features accessible through the BST infrastructure (now undergoing extensive functional and timing simulation [12]) and of a low-cost debug environment supporting the implementation of the procedures described in this paper.

2. Implications of extending the application domain to other areas or levels: IC level, MCMs, system, dedicated hardware versus stored-program; multiprocessor systems; etc.

References