# BIST FOR 1149.1-COMPATIBLE BOARDS:

## A LOW-COST AND MAXIMUM-FLEXIBILITY SOLUTION

José M. M. Ferreira[1,2], Manuel G. Gericota[2], José L. Ramalho[2], Gustavo R. Alves[2]

[1] Faculdade de Engenharia (DEEC)
University of Porto
Rua dos Bragas
4000 Porto - PORTUGAL

[2] INESC
Largo Mompilher, 22
4000 Porto - PORTUGAL

### Abstract

*A set of board-level testability blocks is proposed in this paper, with the aim of improving the fault coverage achievable on boards restricted to commercially available BST components. It is shown that a high flexibility and low-cost solution to board-level BIST is possible by combining an HDL-based implementation with the wide availability of medium-complexity PLDs.*

# 1. INTRODUCTION

The development of highly complex printed circuit boards (PCBs), raising extreme testability requirements, is enabled by the availability of advanced packaging and mounting technologies, and by integration levels producing components with thousands of gates per pin. Boundary scan design and test (BST) was developed in response to this challenge [1,2] and is now largely recognized as being able to effectively improve previous test technologies, essentially in two main areas: lowering the physical accessibility requirements and improving the controllability and observability (C&O) levels of internal PCB nodes. Accepted as an IEEE standard in 1990, BST is now supported by several ATE manufacturers, semiconductor manufacturers and CAD systems.

The number of off-the-shelf BST components available to board-level designers is however still limited, which contributes to the fact that significant non-BST clusters are normally present. It should still be referred that restrictions are present even when the designers are not limited to use commercial components, since including a BST infrastructure into an ASIC will require additional silicon area and pins (eventually requiring a different package). Considering that the BST technology is essentially aimed at board-level test, it is therefore of interest to analyse what are the main board-level testability requirements, and to provide a range of *testability building blocks* (TBBs) able to meet these requirements. Board-level designers will additionally benefit if the specification of these TBBs is sufficiently flexible as to enable changes according to the requirements of each specific application, and if their implementation is based on widely available technology. Although some proposed solutions in this domain have already been described [3:9], a much larger offer is still required.

The solution proposed in this paper is based on three main types of TBBs: a dedicated test processor providing board-level built-in self-test (BIST) capability, interfaces to non-BST digital I/O nodes and an interface to analog I/O nodes. Section two deals with the identification of the main board-level testability requirements to be met and section three presents the TBBs proposed. The fourth section presents implementation details, showing that a low-cost and high-flexibility solution to board-level BIST is possible by using a simple hardware description language (HDL) to specify all TBBs, which are then implemented in widely available medium-complexity programmable logic devices (PLDs). The fifth section presents an example to illustrate the practical application of part of the TBBs proposed.

# 2. MAIN BOARD-LEVEL TESTABILITY REQUIREMENTS

While the general board-level testability requirements address the major goal of improving the C&O of internal nodes, specific requirements have to be considered if board-level BIST is to be achieved. Both types will be considered in this section, since they will lead to a set of TBBs which can then be employed according to the requirements to be met.

A first requirement in order to allow board-level BIST is the availability of an on-board test controller (BIST processor) which will run a BIST program stored in local memory. This program will have embedded the complete set of test vectors to be applied, together with the fault-free responses and mask information, and will address the following main steps: test of the BST infrastructure, interconnect test (including those interconnects buried into non-BST clusters) and component test (mainly by executing component-level BIST functions) [10].

The test program run by the BIST processor will generate a sequence of low-level BST test access port



0-7803-1429-8/93 $3.00 

(TAP) operations, where the following three main types of operations can be identified:

- State transition, where the TMS (test mode select) value present at the rising edge of TCK (test clock) defines the next state of the internal TAP controller in each BST component. A single TCK cycle is therefore required to complete this operation.
- Shifting data through the selected registers, which takes place by keeping TMS at "0", except on the last bit to be shifted (in order to step from the *Shift* state to the *Exit1* state). The number of TCK cycles required to complete this operation will be given by the length of the selected scan path.
- Application of TCK cycles while TMS is kept at "0", in order to execute available component-level BIST functions (the TAP controllers in these components will previously have been taken to the *Run Test / Idle* state). The number of TCK cycles required to complete this operation will be imposed by the component requiring the largest number of TCK cycles to complete BIST execution (around 1.2 million for the i486, for example).

These "standard" low-level TAP operations can then be used to define the core of the dedicated BIST processor. Previous work done by the authors in this field has already been presented in [4], where a 68-pin ASIC device implementing such a BIST processor is described. Further work has however produced improvements in the corresponding architecture and implementation, which will be described in section 3.1.

Further board-level testability requirements are essentially related to the C&O levels of internal nodes. Interconnects associated to BST pins exhibit excellent levels of C&O, which allow straightforward procedures for structural fault detection and diagnosis. However, complex digital non-BST clusters will raise testability problems, both because of the number of test vectors which may be required and because of the need to serialise these test vectors [11,12]. Board-level designers should therefore be able to use solutions providing BST access to non-BST nodes, these being either located within non-BST clusters or in edge connectors. BST components are already available, which provide local test pattern generation and response evaluation through an enhanced BST infrastructure [8]. Concerning primary I/O pins, the concept of an "active connector" [13] is able to provide BST access to these pins as well, such as illustrated in figure 1.

Mixed signal designs where analog clusters are also present on the board will generally present low-levels of C&O, which bring up the final main requirement considered for board-level testability: the designer should be able to guarantee BST access to nodes within or surrounding these clusters, even if only for basic C&O operations.

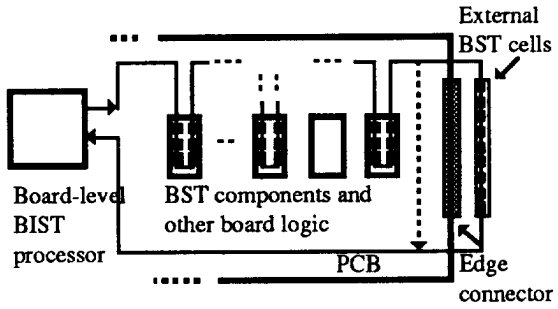


**Fig. 1:** BST access to primary I/O pins.

In summary, the main board-level testability requirements identified may be stated as follows:

- Board-level BIST should be supported by a dedicated test processor, with an instruction set designed to optimise the three main types of low-level TAP operations identified. An automatic test program generation (ATPG) tool must be available to assist the task of generating the test code to be executed. Although a significant user intervention will still be required, it is nevertheless possible to automate important parts of this task.
- The designer should guarantee BST access to non-BST digital nodes, both to primary I/O pins and to pins located within complex non-BST clusters. Simple access (EXTEST mandatory operating mode) will be sufficient for primary I/O pins, but local test pattern generation and response evaluation should be made available for complex non-BST clusters.
- When analog clusters are present, the designer should guarantee that at least two basic operations are possible in a subset of these nodes: capturing the analog values present on the nodes to be observed and forcing the required analog values on the nodes to be controlled.

The next section will present a set of TBBs proposed with the objective of meeting these requirements.

# 3. THE TBBs PROPOSED

The board-level BIST processor will be presented first, followed by the TBBs addressing the requirements identified for non-BST digital and analog nodes. The BIST processor interfaces directly to all the remaining TBBs, which can therefore be seen as peripherals to be used according to the testability requirements to be met.

## 3.1. The board-level BIST processor

The BIST processor architecture is illustrated in figure 2 and supports an optimised instruction set allowing a

straightforward implementation of all the procedures required to test the board, according to the three main types of low-level operations identified in the previous section.

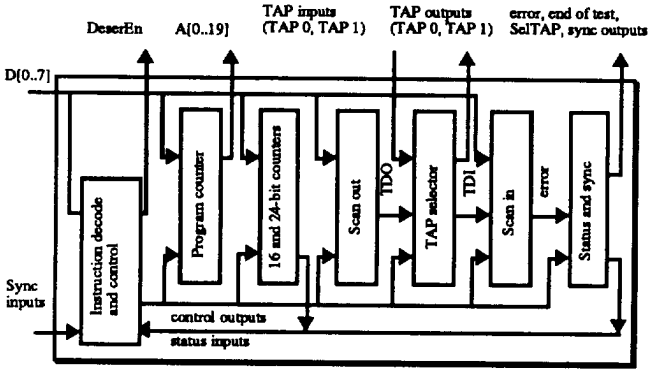


**Fig. 2:** Block diagram of the board-level BIST processor.

Notice that a TAP selector block allows the internal processor resources to be multiplexed by two board BST chains, and that a 20-bit program counter is able to address test programs up to 1 Mbyte. Status information concerning error detection, end-of-test condition and the indication of which board BST chain is being controlled at each moment, are kept internally on the *Status and sync* register and made available externally on three output pins. Two additional bits of this register are also available externally on dedicated output pins, which can be used to synchronise BIST program execution with external events, such as the handshake procedures required for successive approximation A/D converters.

Since this same processor may eventually be used as an external test controller in low-cost equipment, an output pin (*DeserEn*, in figure 2) is provided to indicate when valid test responses are being shifted out of the board BST chain. A deserialiser peripheral can then be used to convert this data into 8-bit wide words to be stored in memory and later accessed by a diagnostic tool.

The instruction set of the BIST processor is presented in table 1, including those instructions which do not directly represent TAP operations.

As an example, and if we consider that the TAP controller of each BST component is in the *Exit1-IR* state, following an instruction load operation which left either the RUNBIST or BYPASS opcodes in each instruction register, then the BIST processor may order component-level BIST execution through the following sequence of instructions:

```
TMS1                ; go to Update IR
TMS0                ; go to Run Test / Idle
LD C24, num_cycles  ; load the 24-bit counter with the
                    ; required number of clock cycles
NTCK                ; execute BIST in each component
```

A Windows 3.1 based ATPG tool was also developed, which partially automates the task of generating the test program for the BIST processor. This tool reads a set of input files containing board-level structural information, a description of the BST infrastructure present in each component, and a description of existing non-BST clusters (including the identification of the surrounding BST cells and of the test vectors to be used). The test code is then generated, addressing the three main steps already referred: checking the integrity of the board-level chains, testing the interconnects and testing the components.

| TAP OPERATIONS | |
|---|---|
| SELTAP0, SELTAP1 | Selects the BST chain to be controlled by the following instructions. |
| TRST | Forces an asynchronous reset through the /TRST output of the selected BST chain. |
| NSHF | N bits will be shifted into the BST chain. Bits shifted out of the BST chain are not compared. N represents the contents of the internal 16 bit counter. |
| NSHFCP | N bits will be shifted into the BST chain. Bits shifted out of the BST chain are compared with their expected value. Mask bits are used to discard don't care bits. N represents the contents of the internal 16 bit counter. |
| TMS0, TMS1 | Forces a state transition in the internal BST logic of each component, in the selected BST chain. |
| NTCK | Applies N test clock cycles, while keeping TMS at "0". N represents the contents of the internal 24 bit counter. |
| **INTERNAL CONTROL AND SYNCHRONISATION** | |
| LD C16, N | Loads the internal 16-bit counter with the number of test clock (TCK) cycles to be applied. |
| LD C24, N | Loads the internal 24-bit counter with the number of test clock (TCK) cycles to be applied. |
| JPE Address JPNE Address | Conditional jumps based on the state of the internal error flag. |
| SS0, SS1 | Forces a logical value (0,1) on the synchronism output. |
| WS0, WS1 | Waits for a logical value (0,1) on the synchronism input. |
| HALT | Terminates test program execution. |

**Table 1:** Instruction set supported by the board-level BIST processor.

## 3.2. The interface to non-BST digital I/O nodes

Non-BST digital I/O nodes belonging to edge connectors (primary I/O pins) may be tested through test channels synchronised with the on-board BST chains, but the simple solution illustrated in figure 1 may also be used. Although requiring that an "active socket" is placed in each connector, and that the BIST processor program is generated for this special test set-up, this approach presents the advantage of integrating the test of primary I/O interconnects through the BST infrastructure. The set of external BST cells can be integrated into one TBB which is a simple no-core logic BST component containing the standard BST infrastructure shown in figure 3.

The boundary scan register of the component shown consists of 10 input pins and 26 bidirectional pins with individual tristate control, and the instructions supported

conform to the IEEE 1149.1 standard requirements (EXTEST, SAMPLE / PRELOAD and BYPASS). Different numbers of input or bidirectional pins are also possible (such as will be explained in section 4), and complete C&O over the primary I/O pins may be achieved by simply cascading the required number of these components. A similar solution might be achieved by using commercially available BST octals, but these components would not allow individual tristate control of each output, which may be required.

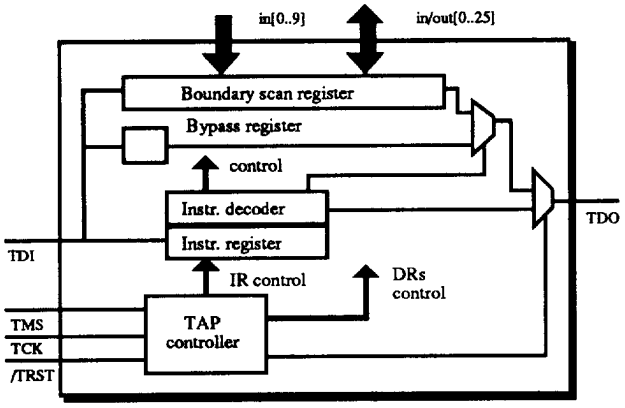


**Fig. 3:** Block diagram of the primary I/O pin TBB.

Complex non-BST clusters may have to be tested by in-circuit test equipment, although there are cases where the surrounding BST infrastructure may instead be used (virtual cluster testing). An example may be found on combinational clusters, which may efficiently be tested by pseudo-random pattern generation (PRPG) and signature analysis (SA) techniques.

In these cases, individual tristate control or programmable-length linear feedback shift registers (LFSRs) might be useful, features which are not supported by commercially available components. Programmable-length LFSRs might specially be of use when guarding values are to be applied through those bits not used for PRPG. These requirements led to the development of a programmable-length LFSR TBB providing both PRPG and SA, with individual tristate control in each output, and where those bits which are not used for PRPG will keep their initial value. A total of 20 input pins can be used to perform response compression through SA. An internal control register may be selected by a dedicated instruction, and loaded with a 4-bit word which defines the number of bits required to be non-PRPG outputs with individual tristate control (between 0 and 15). Since 20 output pins are provided, the length of the LFSR will be given by 20-(CR), where (CR) represents the value loaded into the control register. A 3-bit instruction register supports the instructions described in table 2.

| Opcode | Instruction | Selected data register |
|---|---|---|
| 000 | Extest | Boundary scan |
| 001 | Sample / Preload | Boundary scan |
| 010 | Control register scan | Control register |
| 011 | Read SA | Boundary scan |
| 100 | Boundary PRPG/SA | Bypass |
| remaining | Bypass | Bypass |

**Table 2:** BST instructions supported by the PRPG/SA TBB.

The block diagram of this TBB is identical to the one shown in figure 3, with the exceptions that an additional data register exists (the control register) and that an enhanced boundary scan register supports PRPG and SA operating modes.

### 3.3. The interface to analog I/O nodes

The main goal behind the development of the BST technology was to provide structural testing of high-complexity digital boards. Functional or analog test are consequently areas where the BST technology faces important limitations [14]. An interface to analog I/O nodes may therefore be very useful, even if restricted to simple low-speed test operations [15].

In order not to cause delays, distortion, or frequency response limitations on the analog signals, no analog multiplexers should be inserted into the signal flow path. Capture operations face smaller restrictions, but the analog nodes to be controlled would in this case consist only of primary input nodes, through a set-up similar to the one shown in figure 1. However, if it is acceptable to insert analog multiplexers, the solution illustrated in figure 4 may be implemented.

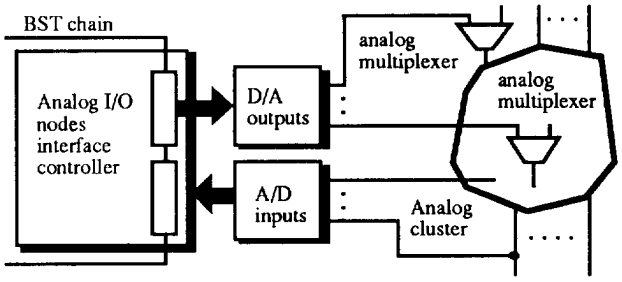


**Fig. 4:** The interface to analog I/O nodes.

The analog I/O nodes interface controller allows BST access to 16 analog inputs and 16 analog outputs. The block diagram of this interface controller is also identical to the one shown in figure 3, but now with the exceptions that the boundary scan register is divided as shown in figure 5 and that a dedicated block interfaces the A/D and D/A converters to the instruction register of this TBB.

Capture and compare operations on analog I/O nodes

assume that there is an interval on which the captured analog value is considered correct, meaning that some type of mask must be used to specify the acceptable range at the output of the A/D converter. However, and since adjacent binary codes may exhibit changes on as many as *every* bit (consider for example the codes corresponding to decimals 127 and 128), a code conversion operation must be performed, so that a mask can be used to specify the accepted deviation.

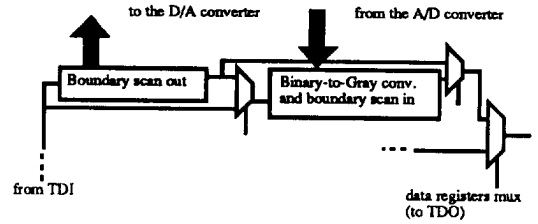


**Fig. 5:** Boundary scan register of the analog I/O nodes interface controller.

A simple solution consists of performing a binary to Gray code conversion, which guarantees that successive codes do not differ in more than one bit position. Allowing a four-code acceptable range may then be accomplished by using a mask generated by ex-noring the two codes adjacent to the expected value. As an example, and if the expected Gray code word is 00011100, the mask word is /(00011101⊕00010100) = 11110110. Use of this mask for comparing the Gray code equivalent of the A/D converter output will correspond to an acceptable range defined by 1111X11X (X = don't care). Since the binary to Gray code conversion is achieved by ex-oring each bit to its left neighbour (binary $I_n$, $I_{n-1}$, ..., $I_0$ corresponds to Gray $I_n$, $I_n \oplus I_{n-1}$, ..., $I_1 \oplus I_0$), this operation is implemented by simply adding an ex-or to the serial input of the BST cells connected to the A/D converter output.

| Opcode (G defines the group) | Instruction | Selected data register |
|---|---|---|
| XXXX XXXX 000 | Extest | Boundary scan |
| XXXX XXXX 001 | Sample / Preload | Boundary scan |
| XXXX G000 010 | A/D conversion (channel 0) | Boundary scan in |
| ... | A/D conversion (channels 1 to 6) | Boundary scan in |
| XXXX G111 010 | A/D conversion (channel 7) | Boundary scan in |
| G000 XXXX 011 | D/A conversion (channel 0) | Boundary scan out |
| ... | D/A conversion (channels 1 to 6) | Boundary scan out |
| G111 XXXX 011 | D/A conversion (channel 7) | Boundary scan out |
| XXXX XXXX 1XX | Bypass | Bypass |

**Table 3:** BST instructions supported by the analog I/O TBB.

An 11-bit instruction register allows A/D or D/A conversions to take place individually on the selected channels, such as described in table 3 (the 16 inputs and outputs are divided in two groups of 8, selected according to instruction register bit G).

The A/D converters used are assumed to be of the successive-approximation type, and the end of conversion state may be checked by examining the bits shifted out of the instruction register following a Capture-IR operation (bits 2 and 3 capture the end-of-conversion signal from each 8-channel A/D converter).

# 4. TBB IMPLEMENTATION

With the exception of the analog I/O nodes interface controller, each TBB presented in the previous section was implemented on one 68-pin medium-complexity PLD (Altera 5128, 128 macrocells). The analog I/O nodes interface controller shown in figure 4 was implemented on a smaller 44-pin medium-complexity PLD (Altera 5064, 64 macrocells).

Every component was specified using the Altera HDL (AHDL), which allowed fast specification and debugging. Careful design rules had however to be observed, since usage of the internal resources is very close to the maximum complexity allowed, as can be seen in table 4.

| | Dedicated input pins used | I/O pins used | Macrocells used | Expanders used |
|---|---|---|---|---|
| Prim. I/O pin | 100 % | 75 % | 100 % | 9 % |
| PRPG and SA | 100 % | 71 % | 99 % | 92 % |
| Analog I/O int. | 100 % | 96 % | 95 % | 28 % |
| BIST processor | 100 % | 78 % | 99 % | 82 % |

**Table 4:** Resource usage in each PLD (TBB).

Combining an HDL specification with an implementation based on PLDs guarantees the required high flexibility, since it becomes easy to adapt the functionality of each TBB to the specific requirements of each application. The new TBBs can then be made available by simply programming new PLDs.

The use of a simple HDL makes specifications readable even by non-HDL users, as can be illustrated by the following AHDL specification of a 16-bit program counter (an 8-bit latch is used to temporarily store the most significant byte, since an 8-bit data bus is assumed):

```
if (ld_pc_msb) then
    latch[7..0] = data[7..0];
    pc[15..0] = pc[15..0];
elsif (ld_pc_lsb) then
    pc[15..8] = latch[7..0];
    pc[7..0] = data[7..0];
elsif (incr_pc) then
```

```
    pc[15..0] = pc[15..0] + 1;
else
    pc[15..0] = pc[15..0];
end;
```

The HDL/PLD based approach to design and implement each TBB makes it possible to produce a library of TBBs, where the basic four types proposed are available on different versions (different ratios of input/output pins for the primary I/O pins TBB, a smaller package board-level BIST processor with a smaller address space and supporting only one BST chain, etc.). Reusability is also an important issue, since the specification of several blocks can be used in different TBBs. This can be illustrated by considering the BST infrastructure existing in the TBBs described in sections 3.2 and 3.3, which is largely based on a common AHDL specification.

# 5. AN APPLICATION EXAMPLE

Several application circuits were used to validate the set of TBBs proposed, most of them during a larger project where a low-cost modular test equipment was developed. The example shown in figure 6 consists of a small board with two BST chains and two simple non-BST digital clusters.

The test set-up for this board included one BIST processor responsible for controlling both BST chains, while the primary I/O pins were interfaced according to the scheme illustrated in figure 1. This test set-up included three primary I/O pins TBBs placed in BST chain number 1. Since this chain contains only one BST octal (18 BS cells) and two TBBs were placed in bypass mode (one is enough to provide the required number of external BST cells), the number of TCK cycles required to shift one vector into / out of BST chain 1 is 82 (18+2+62).

The fault model assumed by the ATPG tool generating the BIST processor program considers open and short interconnect faults. A repetitive behaviour is assumed for open faults, but the value captured on a floating input is considered unknown (the test vectors generated to detect open faults force both "0" and "1" values through each net driver in turn). A slightly modified version of the self-diagnosis algorithm [16] is used to guarantee 100% short fault detection (no dominant value is assumed for short faults, but it is considered that all inputs in presence capture the same value).

The complete test program for this example occupies approximately 1 Kbyte, including all test vectors, fault-free responses and mask information (the test vectors for the two non-BST clusters were generated by an external ATPG tool). This test program may be illustrated by the following segment, which addresses the detection of open faults.

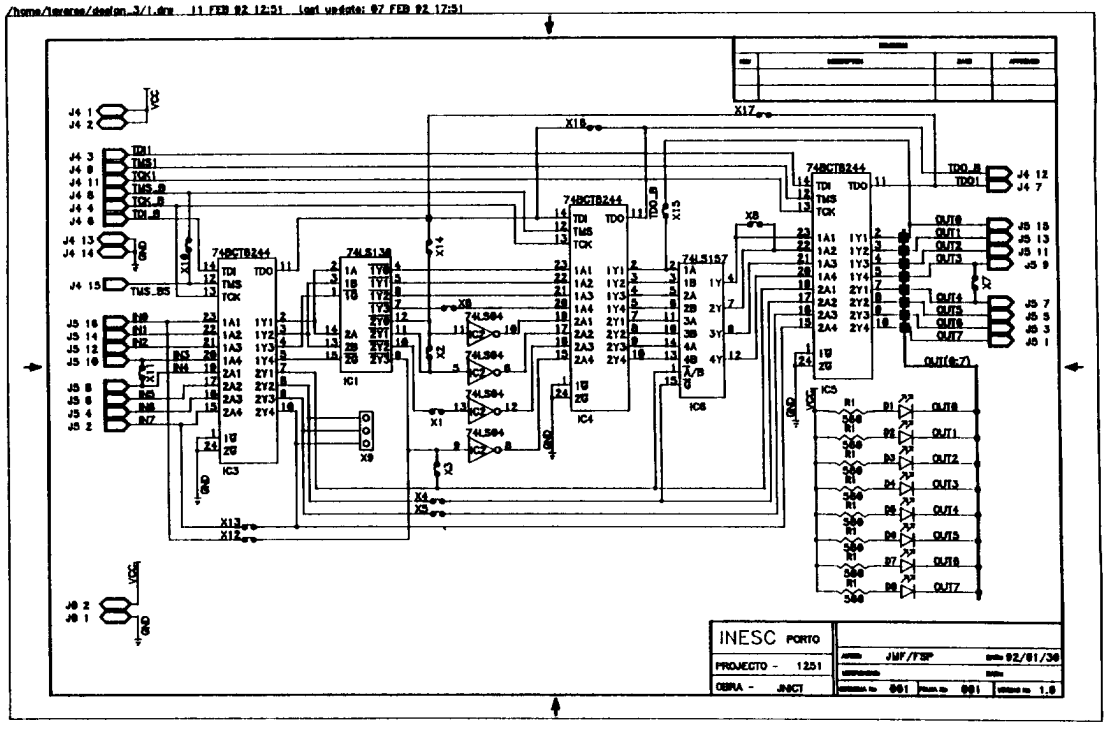


**Fig. 6:** An example with two BST chains and small non-BST clusters.

```
0120   00069 02 00 24      ld    c16,36   ; Length of BS chain 0
0121   0006C 04            nshf           ; Shift in vector # 1
0122   0006D 00            .db   $00
0123   0006E 00            .db   $00
0124   0006F 00            .db   $00
0125   00070 00            .db   $00
0126   00071 00            .db   $00
0127   00072 01            tms1           ; Go to Update DR
0128   00073
0129   00073 1B            seltap1        ; Switch to TAP 1
0130   00074
0131   00074 01            tms1           ; Go to Select DR Scan
0132   00075 00            tms0           ; Go to Capture DR
0133   00076 00            tms0           ; Go to Shift DR
0134   00077
0135   00077 02 00 52      ld    c16,82   ; Length of BS chain 1
0136   0007A 04            nshf           ; Shift in vector # 1
0137   0007B FE            .db   $fe
0138   0007C 01            .db   $01
0139   0007D 00            .db   $00
0140   0007E 00            .db   $00
0141   0007F 00            .db   $00
0142   00080 00            .db   $00
0143   00081 00            .db   $00
0144   00082 00            .db   $00
0145   00083 00            .db   $00
0146   00084 00            .db   $00
0147   00085 00            .db   $00
0148   00086 01            tms1           ; Go to Update DR
0149   00087
0150   00087 01            tms1           ; Go to Select DR Scan
0151   00088 00            tms0           ; Go to Capture DR
0152   00089 00            tms0           ; Go to Shift DR
0153   0008A
0154   0008A 1A            seltap0        ; Switch to TAP 0
0155   0008B
0156   0008B 01            tms1           ; Go to Select DR Scan
0157   0008C 00            tms0           ; Go to Capture DR
0158   0008D 00            tms0           ; Go to Shift DR
0159   0008E
0160   0008E 02 00 24      ld    c16,36   ; Length of BS chain 0
0161   00091 05            nshfcp         ; Shift in vector # 2
0162   00092 00 FF 00      .db   $00,$ff,$00
0163   00095 00 FF 00      .db   $00,$ff,$00
0164   00098 0C FC 03      .db   $0c,$fc,$03
0165   0009B 00 03 FC      .db   $00,$03,$fc
0166   0009E 00 00 0F      .db   $00,$00,$0f
0167   000A1 06 00 05 A7   jpe   theend   ; stop if faulty
0168   000A5 01            tms1           ; Go to Update DR
0169   000A6
0170   000A6 1B            seltap1        ; Switch to TAP 1
0171   000A7
0172   000A7 02 00 52      ld    c16,82   ; Length of BS chain 1
0173   000AA 05            nshfcp         ; Shift in vector # 2
0174   000AB FE FF 00      .db   $fe,$ff,$00
0175   000AE 01 FF 00      .db   $01,$ff,$00
0176   000B1 00 FF 00      .db   $00,$ff,$00
0177   000B4 F8 FF 00      .db   $f8,$ff,$00
0178   000B7 07 FF 00      .db   $07,$ff,$00
0179   000BA 00 FF 00      .db   $00,$ff,$00
0180   000BD 00 7F 80      .db   $00,$7f,$80
0181   000C0 00 80 7F      .db   $00,$80,$7f
0182   000C3 FF FF 00      .db   $ff,$ff,$00
0183   000C6 00 FC 03      .db   $00,$fc,$03
0184   000C9 00 00 03      .db   $00,$00,$03
0185   000CC 06 00 05 A7   jpe   theend   ; stop if faulty
0186   000D0 01            tms1           ; Go to Update DR
0187   000D1
0188   000D1 01            tms1           ; Go to Select DR Scan
0189   000D2 00            tms0           ; Go to Capture DR
0190   000D3 00            tms0           ; Go to Shift DR
0191   000D4
0192   000D4 1A            seltap0        ; Switch to TAP 0
```

Notice that the operand of the NSHFCP (multiple shift and compare) instruction contains the data to be shifted into the selected BST chain, the fault-free responses and the mask information. These three types of data are byte-interleaved in memory and each next three-byte block is loaded into the BIST processor while the board-level shift operation is in progress, which maximises the test execution speed.

# 6. CONCLUSION

A set of testability building blocks (TBBs) has been proposed with the objective of meeting the main testability requirements identified for 1149.1-compatible boards. All TBBs were specified using a simple hardware description language (HDL) and were implemented in medium complexity programmable logic devices (PLDs). This HDL/PLD based design and implementation approach makes it possible to develop a library of highly flexible TBBs, which can be rapidly adapted to the specific requirements of each application (any changes can be made by simply editing the corresponding text file and re-compiling the design). Fast prototyping is also a keypoint for flexibility, since an unrestricted number of different versions can be programmed and tested as many times as required. Finally, if small volume productions are required, it is possible to combine reduced time-to-market with the lower price of pre-programmed parts.

The set of TBBs proposed can also be used for non-BIST applications, such as the development of low-cost 1149.1 test equipment, where the so-called BIST processor is used as the test controller. Additional peripherals to the BIST processor (such as the deserialiser peripheral referred in section 3.1) have already been developed and have been put to use in this type of applications.

The complete set of specification files for the four types of TBBs developed has been made available by public domain ftp. All that is required is to connect to ftp.inescn.pt (use anonymous as username, and your e-mail address as password), and move to a directory called pub/doc/dftplds. Any questions or comments can be addressed to any of the authors at e-mail address jmferreira@porto.inescn.pt.

# REFERENCES


[1] IEEE Std 1149.1, *IEEE Standard Test Access Port and Boundary Scan Architecture*. IEEE Standards Board, May 1990.

[2] C. Maunder and R. Tulloss, "An Introduction to the Boundary Scan Standard: ANSI/IEEE Std 1149.1," *Journal of Electronic Testing: Theory and Applications*, March 1991, Vol.2, Nº 1, pp. 27-42.

[3] W. Bruce, M. Gallup, G. Giles and T. Munns, "Implementing 1149.1 on CMOS Microprocessors," *IEEE International Test Conference Proceedings*, 1991, pp. 879-886.

[4] J. Matos, F. S. Pinto and J. M. Ferreira, "A Boundary Scan Test Controller for Hierarchical BIST," *IEEE International Test Conference Proceedings*, 1992, pp. 217-223.

[5] N. Jarwala and C. Yau, "The Boundary-Scan Master: Architecture and Implementation,"

*European Test Conference Proceedings*, 1991, pp. 1-10.

[6] S. Kritter and T. Rahaga, "Boundary Scan and BIST Compatible IEEE 1149.1: VHDL & Autosynthesis of a SRAM Tester Macrocell and Chip," *European Test Conference Proceedings*, 1991, pp. 17-25.

[7] P. Raghavachari, "Circuit Pack BIST from System to Factory - The MCERT Chip," *IEEE International Test Conference Proceedings*, 1991, pp. 641-648.

[8] L. Whetsel, "An IEEE 1149.1 Based Logic / Signature Analyzer in a Chip," *IEEE International Test Conference Proceedings*, 1991, pp. 869-878.

[9] L. Whetsel, "A Proposed Method of Accessing 1149.1 in a Backplane Environment," *IEEE International Test Conference Proceedings*, 1992, pp. 206-216.

[10] R. Tulloss and C. Yau, "BIST & Boundary-Scan for Board Level Test: Test Program Pseudocode," *European Test Conference Proceedings*, 1989, pp. 106-111.

[11] P. Hansen, "Taking Advantage of Boundary-Scan in Loaded-Board Testing," in C. Maunder et al., *The Test Access Port and Boundary Scan Architecture*. The IEEE Computer Society Press. ISBN 0-8186-9070-4, pp. 81-96.

[12] P. Hansen, "Assessing Fault Coverage in Virtual In-Circuit Testing of Partial Boundary-Scan Boards," *European Test Conference Proceedings*, 1991, pp. 393-396.

[13] C. Maunder and R. Tulloss, "Testability on TAP," *IEEE Spectrum*, February 1992, pp. 34-37.

[14] P. Fleming, "Applications of the IEEE Std 1149.1: An Overview," in C. Maunder et al., *The Test Access Port and Boundary Scan Architecture*. The IEEE Computer Society Press. ISBN 0-8186-9070-4, pp. 129-140.

[15] J. Hirzer, "Testing Mixed Analog/Digital ICs," in C. Maunder et al., *The Test Access Port and Boundary Scan Architecture*. The IEEE Computer Society Press. ISBN 0-8186-9070-4, pp. 199-204.

[16] W. Cheng, J. Lewandowski and E. Wu, "Diagnosis for Wiring Interconnects," *IEEE International Test Conference Proceedings*, 1990, pp. 565-571.