Remote Boundary-Scan Testbench with Extended SFV Commands

Américo Dias *, Paulo Sousa *†, José M. M. Ferreira *‡
* Universidade do Porto - Faculdade de Engenharia, Porto, PORTUGAL
† Instituto de Engenharia Biomédica, Porto, PORTUGAL
‡ Buskerud University College, Kongsberg, NORWAY
email: {americo.dias, sousa, jmf}@fe.up.pt

***Abstract*—This paper proposes a solution for real-time monitoring of boundary-scan compatible integrated circuits. To enable the monitoring of forbidden patterns, a extended set of SFV instructions is also introduced. The solution runs in a low-cost embedded board with its own IP address, enabling remote control of the target hardware. An application example is presented, using a simple dual-port boundary-scan board.**

I. Introduction

The IEEE boundary-scan test (BST) standard was approved in 1990 [1] in response to the difficulties faced by traditional in-circuit and functional test technologies, which were increasingly unable to cope with the shrinking trends enabled by surface-mount technology, and the complexity of digital cores, resulting from the exponential growth of gate/pin ratios. The built-in BST infrastructure allocates a digital test cell to each functional pins, enabling direct control of their logic values. A standard 4-pin test access port (TAP) supports scan-in (TDI), scan-out (TDO), mode select (TMS) and clock (TCK) functions. The test programs can be generated automatically and represented in a standard test format known as SVF (Serial Vector Format) [2].

This paper proposes a remote testbench for electronics circuits equipped with IEEE 1149.1-compatible integrated circuits and an extension of the standard Serial Vector Format (SVFe) which enables the design of simple test algorithms and hardware monitoring. It can be used for educational proposes in electrical engineering courses, providing students with a tool that allows them to run tests remotely, as a complement to the test programs executed at the lab during classes. This solution is based on a hardware/software framework that was presented by the authors in a recent paper [3].

The next sections are organized as follows: Section II describes the developed hardware/software solution. Section III depicts the Serial Vector Format, the additional instructions proposed in this work, and presents some utilization examples. Finally section IV addresses the conclusions about this work.

II. System description

The system comprises a hardware / software server plus a client interface, as described in the following sections.

Fig. 1: The ICnova AP7000 Base board (9,65 x 6,1 cm)

A. Server hardware

The ICnova AP7000 Base board [4], [5] is represented in Figure 1.

Its main characteristics can be summarized as follows:

- Software development support includes a GNU C compiler (http://gcc.gnu.org/), a C library optimized for embedded systems (http://www.uclibc.org/), small executable modules containing many common UNIX utilities (http://busybox.net/), a telnet daemon for remote command line access, Dynamic Host Configuration (DHCP), HTTP server, and full control over the general purpose input / output (GPIO) pins and other devices.
- All Linux sources are open source and delivered on the accompanying CD.
- Main hardware: An AT32AP7000 microcontroller with 32-bit data bus, running at 140 MHz (max. 200 MHz). The boards also offers 64 MB SDRAM and 8 MB Flash RAM, a USB-UART, 10/100 Mbps Ethernet, 64 GPIOs, an I2C bus, and a built-in step-down voltage regulator.

The large number of GPIOs enables the implementation of parallel I/O test channels, plus one or more TAPs. In order to protect the ICnova board from undesired hazards, a simple interface board was designed. It comprises only two MAX30001 integrated circuits. The interface board also translate the Device Under Test (DUT) logic level to the same used in the ICNova board (3.3V). The complete hardware is shown in Figure 2.

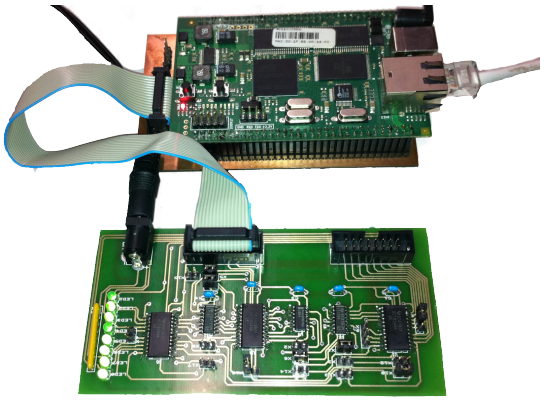

Fig. 2: Server hardware and BST demonstration board

B. Server software

Any functions implemented on the ICnova I/O pins, using the original GPIO drivers, are limited to a few hundred hertz. Due to this reason, and with a view to maximizing the performance of our test controller, new device drivers were written to enable execution of all I/O operations in kernel space. The kernel and user space organization can be represented as shown in Figure 3.

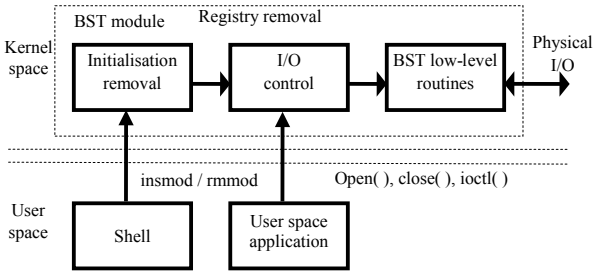

Fig. 3: Kernel and user space

The insmod instruction must be used at the command shell to load the device kernel / module dynamically (when booting the system, this operation can be executed automatically). The open(), close(), ioctl(), etc., system calls are then used to interact with the user space application, making it possible to execute high-level functions in user space, and low-level control functions in kernel space.

The low level routines available in the kernel module, through the ioctl function, are the following:

- SELTAP: TAP selection
- RESET: Puts the selected TAP controller in the RESET state
- STATE: Puts the selected TAP controller in the desired state (i.e. shift IR or shift DR)
- GET_STATE: Returns the current state of the TAP controller
- SHIFT: Shifts data or instructions to TAP controller
- RUNTEST: Forces the TAP controller to a run state for a specified number of clocks
- SET_IO: Set of GPIO pins
- GET_IO: Returns the state of GPIO pins

On the other hand, the user space application deals with the high level operations such as TCP/IP communications, file I/O, interpretation of the SVF code, and interface with the kernel module.

C. Client software

The client software was written in Visual Basic .Net 2010 and runs on Microsoft Windows Operating Systems offering an easy-to-use interface to interact with the server hardware.

The user interface of the client software, comprises various windows selectable by the corresponding tabs. The main features of the client are the SVFe source code editor and the visualization of the BST signals, generated as the test program executes. The SVFe editor, includes a set of features specifically designed to simplify and speed up input of source code, such as syntax highlighting, online syntax checking and correction. The SVFe editor, also has an "compiler" facet, when the SVFe code is converted into hexadecimal machine code, which is preferable for the operation of the remote workbench server. The editor also enables the execution of the SVFe code in debug mode (step-by-step) to facilitate the analysis of the experiment results.

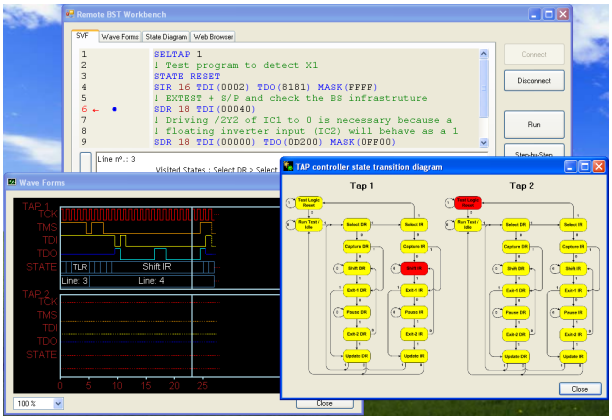

Fig. 4: Client software with detachable windows

To show the waveforms that result from the experiments, a graphic engine was developed. It supports zooming in and out, and offers a "detach window" option to enable observation together with other tab windows as shown in Figure 4. The BST workbench client interface uses the graphic engine provided, to produce a waveform display window showing the digital signals present in the two sets of TAP pins. Following the execution of SIR (Scan Instruction Register), SDR (Scan Data Register), or of any other instructions generating TAP activity, this window enables the users to see the effect of every SVF command on each pin. SVF line numbers are indicated below the sets of waveforms associated to each TAP, as illustrated in Figure 5.

The zoom function, available on the bottom left part of this window, allows the user to see longer segments, or to analyze finer details. The operation codes shifted into the devices' BST instruction registers, in combination with the state of

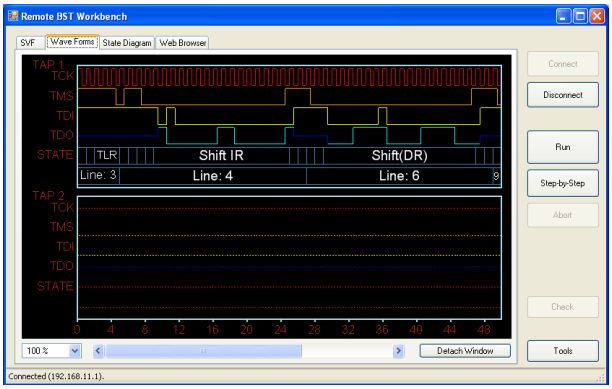

Fig. 5: Client software waveform window

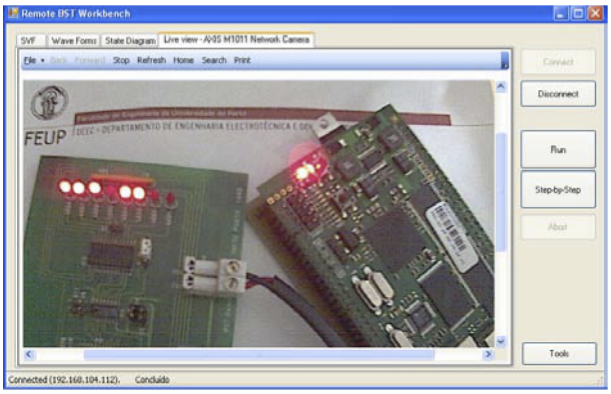

Fig. 7: Client software mini-browser window

the internal TAP controller finite state machine, dictate the operating mode of the BST test infrastructure, and in particular of the BST cells. An additional tab was therefore included to identify the current TAP controller state, as illustrated in Figure 6.

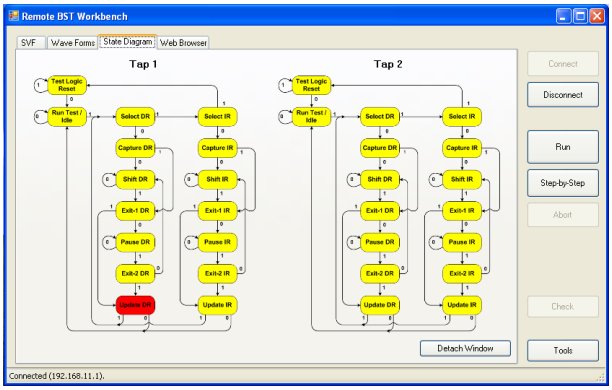

Fig. 6: Client software state diagram window

According to each specific remote experiment, further tabs can be added to the basic core, which includes a mini-browser window (to enable integrated web casting/conferencing, live image feedback from the remote workbench, etc.). The client software integrates the two TCP/IP communication channels that are also present in the server core: 1) a bidirectional synchronous channel to exchange commands and 2) an asynchronous channel for data transfer. File management functions (Open, Save, Save as) are provided on the bottom-left corner (“Tools”). When ready, the SVF code can be executed step-by-step, or completely in one run, using the corresponding button. Live video feedback from the remote workbench may or may not be necessary for BST experiments. In the case of strictly digital test experiments, the waveforms and the data shifted out of the board under test will contain all the necessary information. When live video is required, the stream produced by an IP camera can be visualized in the mini-browser, as illustrated in Figure 7.

Likewise, this mini-browser window can also be used by the lecturer to webcast a demonstration/presentation of structural test detection in distance learning scenarios.

III. Extended Serial Vector Format

The Serial Vector Format (SVF), was developed as a vendor-independent way of representing JTAG test patterns in ASCII (text) files.

SVF is an industry standard file format that is used to describe JTAG chain operations in a compact and portable fashion. SVF files are portable because complicated vendor-specific programming algorithms can be converted to sequences of SVF instructions, requiring no special knowledge of the target device [6].

SVF files consist of a list of statements and/or comments as shown in the next example:

```
SDR 8 TDI(00) TDO(FF) MASK(0F)
```

In this example: i) 8 TCK cycles are required; ii) An all-0 8-bitstream will be shifted into the selected scan chain; iii) The 8 bits shifted out are compared to their expected values XXX1111b; iv) The mask is set to ignore the more significant nibble.

The SVF specification defines the following commands [2]:

- ENDDR: Specifies default end state for DR scan operations.
- ENDIR: Specifies default end state for IR scan operations.
- FREQUENCY: Specifies maximum test clock frequency for IEEE 1149.1 bus operations.
- HDR: (Header Data Register) Specifies a header pattern that is prepended to the beginning of subsequent DR scan operations.
- HIR: (Header Instruction Register) Specifies a header pattern that is prepended to the beginning of subsequent IR scan operations.
- PIO: (Parallel Input/Output) Specifies a parallel test pattern.
- PIOMAP: (Parallel Input/Output Map) Maps PIO column positions to a logical pin.
- RUNTEST: Forces the IEEE 1149.1 bus to a run state for a specified number of clocks or a specified time period.
- SDR: (Scan Data Register) Performs an IEEE 1149.1 Data Register scan.

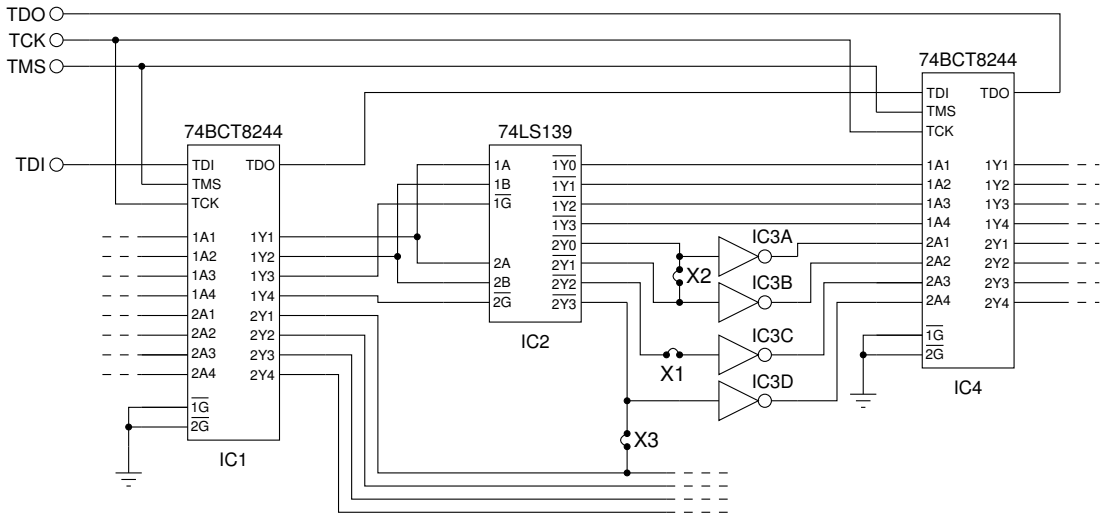

Fig. 8: A sub-circuit of the BST demonstration board

- SIR: (Scan Instruction Register) Performs an IEEE 1149.1 Instruction Register scan.
- STATE: Forces the IEEE 1149.1 bus to a specified stable state. TDR: (Trailer Data Register) Specifies a trailer pattern that is appended to the end of subsequent DR scan operations.
- TIR: (Trailer Instruction Register) Specifies a trailer pattern that is appended to the end of subsequent IR scan operations.
- TRST: (Test ReSeT) Controls the optional Test Reset line.

All commands in SFV programs are executed sequentially. Another limitation of SFV is that it can only control a single IEEE 1149.1 TAP. However, there are cases where multiple IEEE 1149.1 TAPs are present. To overcome these limitations, this work proposes an additional set of commands, to be added as an extension to SFV. The new commands enable the operation of several TAPs, and the design of basic test algorithms using simple / conditional jumps. Another feature introduced by our proposal is the possibility to read / write additional signals. This feature may be used to monitor digital pins outside the IEEE 1149.1 chain or to control additional hardware.

The extended SVF (SVFe) commands are the following:

- SELTAP: (SELect TAP) Selects a tap for operation
- JMP: (JuMP) Executes a jump to a specified label in the SVF program
- JMPE: (JuMP if Error) Jump to a specified label if an error has been detected
- RSTE: (ReSeT Errors) Clean previously detected errors
- JMPP: (JuMP if Port) Jump if a specified input is set (logic level 1)
- SETP: (Set Port) Set the specified output to logic level 1
- RSTP: (Reset Port) Set the specified output to logic level 0
- ADDCHECK: Add a specified test to the check list
- CLEARCHECK: Clear the check list
- SAFECHECK: Executes all the tests in the check list for a specified period of time. If an error is detected, jumps to a specified label.

The last commands, ADDCHECK, CLEARCHECK, and SAFECHECK are particularly useful to enable real-time monitoring of hardware malfunctions. It is possible to create a check list with several forbidden patterns and monitor the hardware for a period of time. If a forbidden pattern is detected, it is possible to immediately disable the I/O pins of the integrated circuits in the boundary scan chain, in an attempt to avoid permanent damage to the hardware.

A. Application examples

A section of our BST demonstration board is illustrated in Figure 8. This sub-circuit is comprises two 74BCT8244 octal buffers equipped with IEEE 1149.1 boundary scan, a 74LS139 dual 2-line to 4-line decoder, and a 74LS04 hex inverter. Three possible defects are shown: X1, X2 and X3. X1 is normally closed and when opened emulates an open circuit between the output $\overline{2Y2}$ of IC2 and the input of the inverter IC3C. X2 is normally open and emulates a short circuit between the outputs $\overline{2Y0}$ and $\overline{2Y1}$ of the decoder. Finally, the defect X3 is normally open and emulates a short circuit between the output 2Y1 of IC1 and the output $\overline{2Y3}$ of IC2. A simple example to detect defects X1, X2, and X3 is shown in Example 1.

```
SELTAP 1
STATE RESET
SIR 16 TDI(0002) TDO(8181) MASK(FFFF)
! X1 test
SDR 18 TDI(00040)
SDR 18 TDI(00000) TDO(0D200) MASK(0FF00)
! X2 and X3 test
SDR 18 TDI(00000) TDO(00800) MASK(00D00)
```

Example 1: Test program to detect X1, X2, and X3 defects using standard SVF

In this example, the TAP 1 is selected and its state is

set to RESET. The integrity of the boundary scan chain is then tested, shifting through the Instruction Register with the instruction SIR. At the same time, IC1 is configured to external test mode and IC4 to sample/preload. The first scan operation is intended to drive the output $\overline{2Y2}$ of IC2 to 0 because a floating inverter input will behave as a 1. Finally, the IC4 I/O values are read and compared with their expected value.

Example 2, shows how to monitor the same defects for a period of time, using the proposed set of SVFe commands. The initialization and the integrity check remain the same as in Example 1.

```
SELTAP 1
STATE RESET
SIR 16 TDI(0002) TDO(8181) MASK(FFFF)
! X1 test
ADDCHECK 18 TDI(00040) TDO(0D200) MASK(0FF00)
! X2 and X3 test
ADDCHECK 18 TDI(00000) TDO(00800) MASK(00C00)
SAFECHECK 20 Error
! No error detected.
(...)
Error:
! Error detected.
```

Example 2: Test program to monitor X1, X2, and X3 defects using SFVe

Two tests are added to the check list and the hardware is monitored for 20 seconds or until a defect is detected. If a defect is found the execution jumps immediately to the label "Error" where the user should put the hardware in a safe state, avoiding permanent damage to the DUT.

IV. Conclusion

This paper proposes a low-cost remote boundary-scan workbench with extended SFV commands. The server hardware is based on a ICNova AP7000 board, running Linux. The client software was written in Visual Basic .Net 2010 and is compatible with Microsoft Windows Operating Systems. It comprises an SVF editor and compiler, a waveform visualizer, a state diagram visualizer and a mini-browser to reproduce live video from experiments or presentations from the lecturer.

The extended SVF instructions enables the control of several IEEE 1149.1 buses, the development of simple algorithms and the monitoring of the hardware under test.

The server and client open source code sets are available from the following web addresses (Code license: GNU General Public License v3; Content license: Creative Commons 3.0 BY-SA):

- http://code.google.com/p/rbstws/
- http://code.google.com/p/rbstw-client/

References

[1] "IEEE standard test access port and boundary-scan architecture," *IEEE Std 1149.1-2001*, 2001.

[2] ASSET InterTech, Inc., "Serial Vector Format specification," pp. 1–26, Mar 1999.

[3] J. M. M. Ferreira, A. Dias, P. Sousa, Z. Nedic, J. Machotka, O. Gol, and A. Nafalski, "Low-cost workbench client / server cores for remote experiments in electronics," *7th International Conference on Remote Engineering and Virtual Instrumentation*, pp. 59–64, Jun 2010.

[4] (2011) ICnova AP7000 Base. [Online]. Available: http://www.ic-board.de/product_info.php?info=p75_ICnova-AP7000-Base.html

[5] (2011) ICnova AP7000 base. [Online]. Available: http://www.avrfreaks.net/index.php?module=Freaks%20Tools&func=viewItem&item_id=874

[6] B. Bridgford and J. Cammon, "SVF and XSVF file formats for xilinx devices," pp. 1–25, Aug 2000.