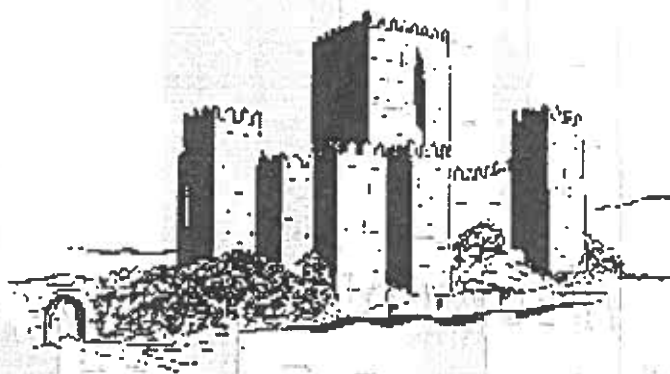


# Student Forum on Industrial Electronics

## ISIE'97



**Guimarães - Portugal**

**University of Minho**

**July 7-11, 1997**

**Advance Program**

Sponsors

**IEEE Industrial Electronics Society**

**University of Minho**



Technical Cosponsors

**IEEE Portugal Section**

**IEEE - Region 8**

# Debug and Test of Microcontroller Based Applications using the Boundary Scan Test Infrastructure

Gustavo R. Alves  
Department of Electrical Engineering  
ISEP  
Rua de S. Tomé  
4200 Porto, Portugal

José M. M. Ferreira  
Department of Electrical Engineering  
FEUP  
Rua dos Bragas  
4000 Porto, Portugal

Daniel Aga and Ovidiu Moşuc<sup>1</sup>  
Department of Electrical Engineering  
University of "Stefan Cel Mare"  
University street nr. 1  
5800 Suceava, Romania

**Abstract** – Microcontroller based applications are usually debugged with the assistance of In-circuit emulators and logic analysers. However, these traditional debug tools represent a huge investment for use in classes with several groups of students working at the same time. The development of a new low-cost debug tool that uses the Boundary Scan Test infrastructure to implement the basic functionality provided by an In-circuit emulator and a logic analyser is a possible solution to overcome this economical problem. To reduce costs the Boundary Scan Test infrastructure is controlled through the parallel port of a normal Personal Computer, now widely available in classrooms.

## I. INTRODUCTION

Boards with a microcontroller are usually debugged using an In-circuit Emulator (ICE) and a logic analyser. These tools are able to assist not only in the detection and analysis of problems faced during the prototype validation but also during the integration of software (e.g. the program run by the microcontroller) with hardware, generally considered to be the most troublesome phase. Unfortunately the proliferation of different microcontroller versions within the same family implies the acquisition of different probes, thus incurring in added costs. Also the use of Surface Mounted Devices (SMD) implies some restrictions in the placement of the logic analyser probes, thus limiting the physical access. These two factors are posing new challenges to traditional debug tools effectiveness in helping designers to validate the prototype and debug the integration of software with hardware.

The concept of Design for Testability (DfT) is being adopted by an increasing number of designers in order to solve, earlier in the design phase, the testability problems faced during the prototype validation and production test. Boundary Scan Test (BST) or IEEE 1149.1 standard [1] is also now widely used and supported by several silicon manufacturers. Although BST is primarily used for the structural test of Printed Circuit Boards (PCBs), it can be used for others applications like functional test, circuit emulation or debug [2, 3, 4, 5, 6].

This paper describes the development of a low-cost debug tool that uses the BST infrastructure and a software package for Windows 95 to implement the basic functionality provided by an ICE and a logic analyser. The BST infrastructure is controlled through the parallel port of a normal Personal Computer (PC), now widely available in classrooms. Section II of the paper identifies and defines the basic functionality provided by our debug tool. Section

III describes the circuit diagram and the debug and test infrastructure implemented. It is assumed the reader has a basic understanding of the IEEE 1149.1 standard. Section IV presents the software package and the Graphical User Interface (GUI) that allows users to select and run the available debug operations. Section V presents some results and future work guidelines. Section VI concludes this paper with closing remarks.

## II. SPECIFICATIONS AND ANALYSIS OF REQUISITES

The first task of our project was to identify the basic functionality provided by ICEs and logic analysers in order to write down the debug tool specifications. A brief study showed that when using an ICE it is generally possible to reset, single step, stop (on break point conditions) or run the microcontroller [7]. When the microcontroller is stopped it is also possible to examine and change the internal registers contents and the internal and external data memory contents. Also the microcontroller may run the program from an internal or external memory. The logic analyser provides the capabilities needed to observe the program execution flow in real time [8]. Characteristics like the maximum acquisition speed, the memory size, the number of data inputs, the number of inputs for synchronisation and the trigger options are generally taken into account when selecting the right logic analyser. The information gathered allowed us to specify four basic debug modes:

- Start mode.
- Single Step (SS) mode.
- Break Point (BP) mode.
- Real Time (RT) mode.

It is also possible to jump from one mode to any other mode, as illustrated in figure 1. A brief description of each mode now follows.

### A. Start mode.

In this mode the microcontroller is in the reset state and it is possible to download the program into the program memory. As the microcontroller is stopped it is also possible to read and change the external data memory contents. To implement these capabilities the debug and test infrastructure must:

- a) provide control to the microcontroller reset line.
- b) provide access and control to both memories.

<sup>1</sup> Presently working at ISEP with student grants from the European TEMPUS Program.

The user may jump from this mode into any other mode by selecting the appropriated command.

#### B. SS mode.

In this mode it is possible to single step the microcontroller activity. There are two options available for what is considered to be a *single step*: providing  $n$  clock cycles to the microcontroller, where  $n$  is user specified, or providing clock cycles until the microcontroller program counter changes it's current value. When the microcontroller is stopped it is possible to read and change the external data memory contents. To implement these capabilities the debug and test infrastructure must:

- provide control to the clock line and monitor the address bus in order to detect changes in the microcontroller program counter.
- provide access and control to the external data memory.

The user may jump from this mode into any other mode by selecting the appropriated command.

#### C. BP mode.

In this mode the user may specify a BP condition and run the microcontroller program until the condition becomes true. Although there are many possible BP conditions it was decide, due to time limits, to implement only the following conditions:

- BP on program address
- BP on program data
- BP on external data memory address
- BP on external data memory data

When the microcontroller is stopped it is possible to read and change the external data memory contents. To implement these capabilities the debug and test infrastructure must:

- monitor in real time the address and data bus in order to detect the BP condition.
- provide control to the clock line in order to stop the microcontroller.
- provide access and control to the external data memory.

The user may jump from this mode into any other mode by selecting the appropriated command.

#### D. RT mode.

In this mode the microcontroller is in a free-running state and it is only possible to monitor and sample the address and data bus. A survey made on several logic analysers trigger options showed that it is generally possible to define an acquisition protocol where the trigger or transition conditions are specified by the user and where the following states commonly exist: start, look for condition (or idle) and store sample. Due to time limits it was decide to just implement two already defined protocols, although there was no specific criteria for our selection. The selected protocols are described on section

III. Again it was decided to restrict the possible conditions to the ones already supported on the BP mode. To implement this capability the debug and test infrastructure must:

- monitor and sample in real time the address and data bus in order to detect the transition condition.
- store the samples in a dedicated memory later to be dumped into the PC.

The user may jump from this mode into any other mode by selecting the appropriated command.

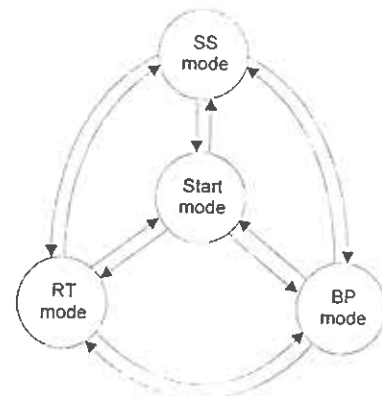


Fig. 1. Possible transitions between supported debug modes

Two restrictions were also identified. It is not possible to examine and change the microcontroller internal registers and data memory contents using an external debug and test infrastructure. To implement this capability the microcontroller should have a BST infrastructure with access to the internal registers. As it is not possible to guarantee this for all microcontrollers it was decided not to include this debug operation in our specifications. The second restriction consists in not allowing the microcontroller to run the program from an internal program memory, as it would be impossible for the debug and test infrastructure to externally monitor the program execution flow. Thus it was decided the microcontroller would always run the program from an external memory (to support the capability of downloading the program from a PC a Non-volatile RAM should be used as the program memory).

### III. THE DEBUG AND TEST INFRASTRUCTURE

Although the original circuit is based on a special version of the popular 8051 microcontroller (the 8951 from ATMEL, designed with static logic so that the microcontroller state may be preserved when the clock is stopped) the procedures and steps next described are intended to be circuit independent e.g., they may be applied to any circuit build around a microcontroller running a program from an external memory.

The requisites presented in the previous section allowed us to define the debug and test infrastructure that had to be inserted in the original circuit in order to implement the four debug modes. Summarising, the debug and test infrastructure must:

- provide control to the microcontroller clock line.
- provide control to the microcontroller reset line.

- c) provide access and control to the external program and data memory.
- d) monitor (and sample) in real time the address and data bus in order to detect: changes in the microcontroller program counter (*SS mode*), the BP condition (*BP mode*) or the transition condition (*RT mode*).
- e) store the samples in a dedicated memory.

With help from the circuit block diagram illustrated in figure 2 we will now describe the implementation of each one of debug and test infrastructure requisites mentioned.

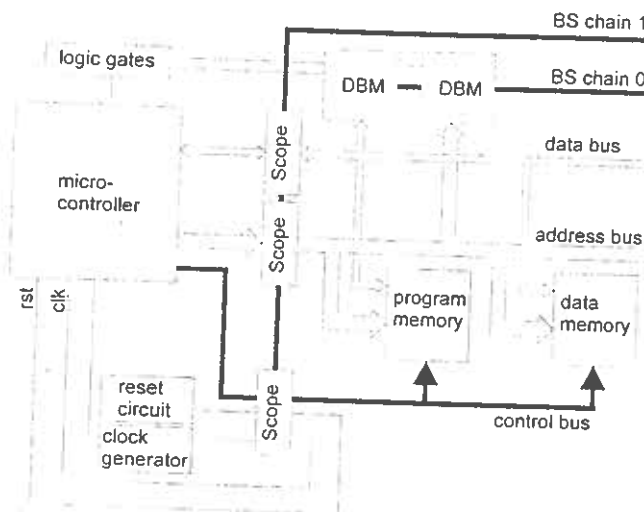


Fig. 2. Circuit block diagram.

To control the microcontroller clock line it is necessary to build an independent clock circuit generator and buffer the clock line using for instance the 74SN8244 *SCOPE Octal* [9] from Texas Instruments. The mandatory EXTEST instruction or the *SCOPE* optional TOGGLE Outputs instruction enables the provision of a certain number of clock cycles to the microcontroller. In some debug modes it is necessary to stop and resume the clock synchronously with the clock circuit generator activity, without causing any spikes that could have hazardous effects. A simple circuit based on a flip-flop and an AND gate had to be designed and placed at the clock generator circuit output, in order to solve this potential problem. The following sequence enables the provision of  $n$  clock cycles to the microcontroller:

- Shift in the SAMPLE/PRELOAD instruction.
- Shift in the vector that places a logic '1' or '0' in the BS cell associated with the SCOPE output pin driving the microcontroller clock line.
- Shift in the EXTEST instruction.
- Shift in the vector that places a logic '0' or '1' (opposite of the previous value) in the same BS cell.
- Repeat last operation 2 ( $n - 1$ ) times.

To control the microcontroller reset line it is necessary to build an independent reset circuit and buffer the reset line before connecting it to the microcontroller, using again a *SCOPE Octal*. The following sequence enables the control of the reset line logic value:

- Shift in the SAMPLE/PRELOAD instruction.

- Shift in the vector that places the desired logic value in the BS cell associated with the SCOPE output pin driving the microcontroller reset line.
- Shift in the EXTEST instruction.
- Provide 6 clock cycles so that the microcontroller may acknowledge reset (microcontroller dependent).

To download the microcontroller program into the external program memory and to view and change the external data memory contents it is necessary to buffer the address and data bus and the control lines of both memories. The following sequence enables the view of the external data memory contents:

- Shift in the SAMPLE/PRELOAD instruction to the buffers driving the address, data and control busses.
- Shift in the vector that places the address 0000H (for a 64Kbytes memory) on the associated BS cells: "don't care" values on the buffer driving the data bus and a that selects a read operation on the external data memory.
- Shift in the EXTEST instruction to the buffers driving the address and control busses.
- Shift out the vector captured on the buffer associated with the data bus and shift in the next address.
- Repeat last operation until necessary.

A light different sequence is used for downloading the program into the external program memory and for modifying the contents of a specific data memory location.

The *SCOPE Octals* illustrated in figure 2 allow the implementation of the first three requisites. These devices are controlled through the board BS chain 1.

To monitor and sample in real time the address and data bus it is necessary to add a device called Digital Bus Monitor (DBM) or 74SN8994 [9, 10]. The DBM contains a special register that allows users to scan in, through the Test Access Port (TAP), a vector and a mask to be compared against the patterns sampled at the 16 input data lines. When the comparison results true an output pin, called Event Qualification Output (EQO) is activated. This pin is connected to the clock circuit generator so that the clock may be stopped immediately after a match is found. Two DBMs are used in our board, one for monitoring the address bus (16 lines) and another for monitoring the data bus (8 lines). The following sequence enables the implementation of a BP on the program address X5AAh:

- Shift in the SCANCN instruction to the DBM connected to the address bus.
- Shift in the vector that selects the following options on the DBM Control Register (CR):
  - RUNN OPCODE = SAMPLE
  - The data inputs are compared directly with the expected data
  - RUNN Clock Signal = the microcontroller line that enables the external program memory outputs (the PSEN line for the 8051)
  - The remaining CR bits are here without significance
- Shift in the SCANEQN instruction

- Shift in the vector that selects the following options on the DBM Event Qualification Register 1 (EQR1):
  - EQO output signal = CTERM (this signal is the result of the compare between the expected data and the vector sampled at the DBM data inputs (presently connected to the address bus))
  - EQO becomes active as soon as the expected data is found
  - The remaining EQR1 bits are here without significance
- Shift in the READFILE instruction
- Shift in the vector that places the following values in the DBM Event Qualification Register 2 (EQR2):

Address	Event Counter	Expected Data	Bit Mask for Data
0	Don't care	05AAh	1111000000000000

- Shift in the RUNN instruction
- When the Expected Data is found EQO will be activated and the microcontroller clock will be stopped

The DBM supports eight different protocols for sampling and storing in an internal memory (16 by 1024 bits) the values present at the 16 data input pins. The DBM internal memory contents can be scanned out through the TAP. The two protocols referred in section II corresponds to the DBM protocols 3 and 5. Protocol 3 may be generally described in the following way:

- Start.
- After  $n$  times *start condition* store words until  $m$  times *stop condition*.
- End.

Where  $n$  and  $m$  are user specified and *start* and *stop conditions* may be any of the supported BP conditions. Protocol 5 may be generally described in the following way:

- Start.
- After  $n$  times *condition* store  $m$  words.
- End.

Where  $n$  and  $m$  are user specified and *condition* is any of the supported BP conditions. The two DBMs illustrated in figure 2 allow the implementation of the last two requisites. These devices are controlled through the board BS chain 0.

#### IV. THE SOFTWARE PACKAGE

The debug and test infrastructure formed by the SCOPE *Octals* and the DBMs is controlled by a software package developed during this project. The software package comprehends a user-friendly GUI, allowing first time users to quickly understand and experiment the supported debug operations. A main window and several secondary windows form the GUI. The nine buttons (second row) displayed in the main window up left hand corner, illustrated in figure 3, form the basic interface for selecting and setting up the conditions for each one of the supported debug modes.

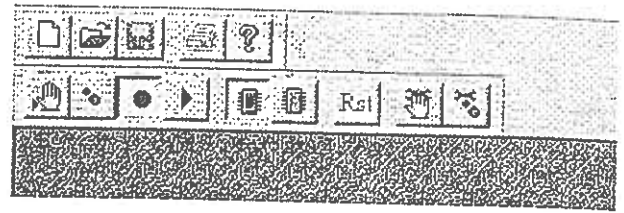


Fig. 3. Main window up left hand corner

A brief description of each button operation will now follow, starting from the left most one:

- SS setup button. When this button is pressed the secondary window illustrated in figure 4 appears. The user may then specify what is considered to be a *single step*. The available options were already presented in section II.
- BP setup button. When this button is pressed the secondary window illustrated in figure 5 appears. The user may then specify the BP condition that will be monitored on the microcontroller busses by the two DBMs. The available options were already presented in section II.

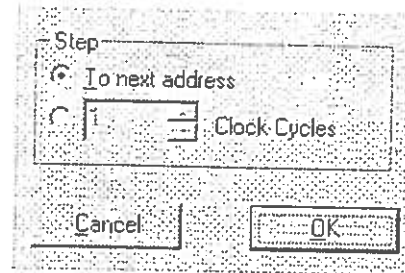


Fig. 4. Single step setup window

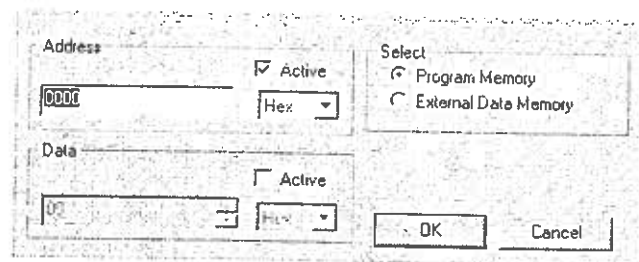


Fig. 5. Break point setup window

- Reset button. When this button is pressed the microcontroller reset line is activated and six pulses are applied at the clock line. Typically the EXTEST instruction is used for this purpose on the SCOPE *Octal* that buffers the two lines.
- Select external data memory button. When this button is pressed the secondary window illustrated in figure 6 appears. The user may then press the Upload button in order to update the displayed window contents with the external data memory contents. To change the contents of one memory location the user must place the mouse pointer on the appropriated cell and click twice on the left mouse button. The window will then change into the form illustrated in figure 7 and the user may then specify the new memory location contents on a hexadecimal, decimal or binary format.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0000	42	4D	EE	00	00	00	00	00	00	00	76	00	00	00	28	00
0001	00	00	10	00	00	00	0F	00	00	00	01	00	04	00	00	00
0002	00	00	78	00	00	00	00	00	00	00	00	00	00	00	00	00
0003	00	00	00	00	00	00	00	00	00	00	00	00	80	00	00	80
0004	00	00	00	80	80	00	80	00	00	00	80	00	80	00	80	80
0005	00	00	C0	C0	C0	00	80	80	80	00	00	00	FF	00	00	FF
0006	00	00	00	FF	FF	00	FF	00	00	00	FF	00	FF	00	FF	FF
0007	00	00	FF	FF	FF	00	77	77	00	00	07	77	77	77	77	70
0008	77	00	07	70	77	77	77	70	70	77	70	70	77	77	77	70
0009	70	77	70	70	77	77	77	70	70	77	70	70	77	77	77	70
000A	77	00	07	70	77	77	77	70	70	77	70	70	77	77	77	70

Fig. 6. External data memory window on upload state

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0000	42	4D	EE	00	00	00	00	00	00	00	76	00	00	00	28	00
0001	00	00	10	00	00	00	0F	00	00	00	01	00	04	00	00	00
0002	00	00	78	00	00	00	00	00	00	00	00	00	00	00	00	00
0003	00	00	00	00	00	00	00	00	00	00	00	00	80	00	00	80
0004	00	00	00	80	80	00	80	00	00	00	80	00	80	00	80	80
0005	00	00	C0	C0	C0	00	80	80	80	00	00	00	FF	00	00	FF
0006	00	00	00	FF	FF	00	FF	00	00	00	FF	00	FF	00	FF	FF
0007	00	00	FF	FF	FF	00	77	77	00	00	07	77	77	77	77	70
0008	77	00	07	70	77	77	77	70	70	77	70	70	77	77	77	70
0009	70	77	70	70	77	77	77	70	70	77	70	70	77	77	77	70
000A	77	00	07	70	77	77	77	70	70	77	70	70	77	77	77	70

Fig. 7. External data memory window on change contents state

- Select external program memory button. When this button is pressed the window illustrated in figure 6 appears. The user may then press the Upload button in order to update the displayed window contents with the external program memory contents. On Start mode it is possible to download a program file from the PC into the external program memory.
- Select RT mode button. When this button is pressed the secondary window illustrated in figure 8 appears. The user may then select one of the protocols presented in section III. When the Start button is pressed the microcontroller will be placed in the free-running state and the two DBMs will, according to the selected protocol, capture and store in real time the values present on the address and data bus. When the DBMs memories are filled up a warning message is displayed and the user may then view the captured values on the window illustrated in figure 9.

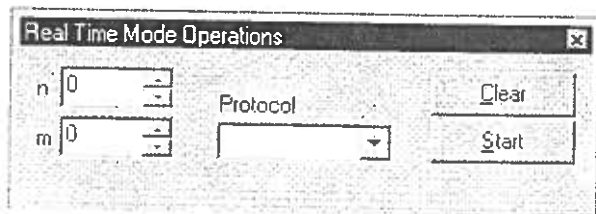


Fig. 8. Real time acquisition protocols window

- Select Start mode button. When this button is pressed the microcontroller clock is stopped and a reset command is issued (similar to press the reset button).
- Select BP mode button. When this button is pressed the BP condition defined on the BP setup window is scanned into the DBMs, which will then start monitoring the two busses. When the BP condition is found (e.g. becomes true) EQO is activated and the

clock is stopped, thus freezing the microcontroller state.

- Select SS mode button. When this button is pressed the microcontroller is single stepped as defined on the SS setup window.

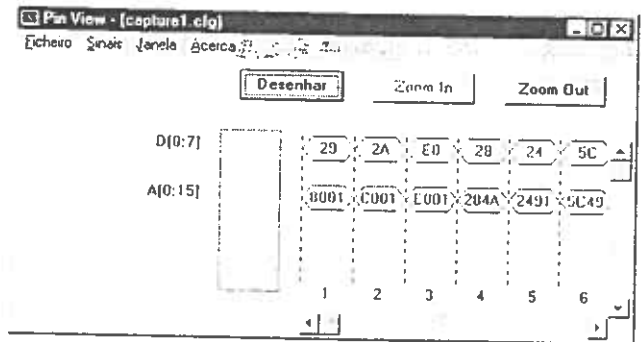


Fig. 9. Secondary window exhibiting the values captured in real-time in the address and data bus.

On the main window bottom right hand corner, illustrated in figure 10, some information regarding the current status and selected options of the debug tool is presented. This information consists of:

- the selected external memory
- the current debug mode
- the current option for the SS implementation
- the current value of the microcontroller program counter
- the current date



Fig. 10. Main window bottom right hand corner

The debug tool GUI allows a straightforward approach for selecting and running the available debug operations. It is not necessary to spend much time in training, as it is generally needed when using an ICE for the first time.

## V. ANALYSIS OF RESULTS AND FUTURE WORK

The inclusion of a debug and test infrastructure in the original circuit implies an added cost. The following item costs may be considered:

- BS components.
- Clock generator circuit, as some microcontrollers just need an external crystal and some capacitors to generate the clock.
- Board connector for the TAPs.
- Extra PCB area.

The cost of replicating and installing the software package is almost negligible. Also the PC is not considered to be a cost, as generally it is already used for developing and validating (through simulation) the microcontroller program code. The cable for connecting the board TAPs to the PC parallel port may be considered as an added cost.

All costs may be compared against the benefits gained, namely dispensing the ICE and the logic analyser. These

traditional debug tools represent a huge investment if one considers several student groups working at the same time in a classroom. Although our debug tool does not implement all the debug operations supported by an ICE and all the acquisition protocols provided by a logic analyser, it does allow to carry out the basic debug operations needed to validate the prototype and assist on the integration of software with hardware. The restrictions of running the microcontroller program from an external memory and not being able to view and change the microcontroller internal registers and data memory contents may be considered too restrictive for some applications. While this may be possible it is realistic to consider that the number of such applications being developed at the same time may be low, typically one or two at the same class. For these applications the ICE will represent the best available debug tool.

As future work it is possible to increase the number of supported BP conditions, namely combining different BP conditions with logic operators (BP on program address AND data) or extending the BP conditions to other circuit lines not directly connected to the microcontroller. This last suggestion could be achieved using the high pin coverage provided by the BST infrastructure. Also the acquisition protocols supported by our debug tool could match the acquisition protocols supported by the DBM. This would imply some modifications on the real time acquisition protocols window and new routines to translate the user options into the vectors needed to program the DBM internal registers.

## VI. CONCLUSION

In this paper we presented a new low-cost debug tool that uses the BST infrastructure to implement the basic functionality provided by an ICE and a logic analyser. The driving force was to supply every student group with a debug tool, that otherwise would not have been possible due to the high cost of ICEs and logic analysers, the traditional tools for prototype validation and verification of the integration phase of software with hardware.

## VII. ACKNOWLEDGMENT

The authors gratefully acknowledge João Paulo Baptista for providing the equipment and support that made possible a rapid project development.

## VIII. REFERENCES

- [1] IEEE Standard Test Access Port and Boundary-Scan Architecture, Oct. 1993, IEEE Std. 1149.1 (Includes IEEE Std. 1149.1a).
- [2] M. F. Lefévre, "Functional Test and Diagnosis: A Proposed JTAG Sample Mode Scan Tester," in *International Test Conference*, pp. 294-303, IEEE Computer Society Press, 1990.
- [3] Richard M. Sedmak, "Boundary-Scan: Beyond Production Test," in *International Test Conference*, pp. 415-420, IEEE Computer Society Press, 1994.
- [4] K. Sievert, Y. Manoli, A. Both and R. Lerch, "On-chip Emulation and Debugging for Embedded Microcontrollers using the IMS ScanDebugger," in *European Design & Test Conference User Forum*, pp. 229-233, IEEE Computer Society Press, 1995.
- [5] Andy Halliday, Greg Young and Al Crouch, "Prototype Testing simplified by Scannable Buffers and Latches," in *International Test Conference*, pp. 174-181, IEEE Computer Society Press, 1989.
- [6] Jerry Katz, "A Case-Study in the use of Scan in microSPARC<sup>TM</sup> testing and debug," in *International Test Conference*, pp. 70-75, IEEE Computer Society Press, 1994.
- [7] Bruce Erickson, "Selecting the Right Debugging Tool," in *Electronic Design*, pp. 83-98, Oct. 1995.
- [8] Thomas R. Blakeslee and Jan Liband, "Real-Time Debugging Techniques: Hardware-Assisted Real-Time Debugging Techniques for Embedded Systems," in *Embedded Systems Programming*, vol. 8, n° 4, 1995.
- [9] Texas Instruments, *SCOPE<sup>TM</sup> Logic Products: Application and Data Manual*, 1994.
- [10] Lee Whetsel, "An IEEE 1149.1 Based Logic/Signature Analyser in a Chip," in *International Test Conference*, pp. 869-878, IEEE Computer Society Press, 1991.

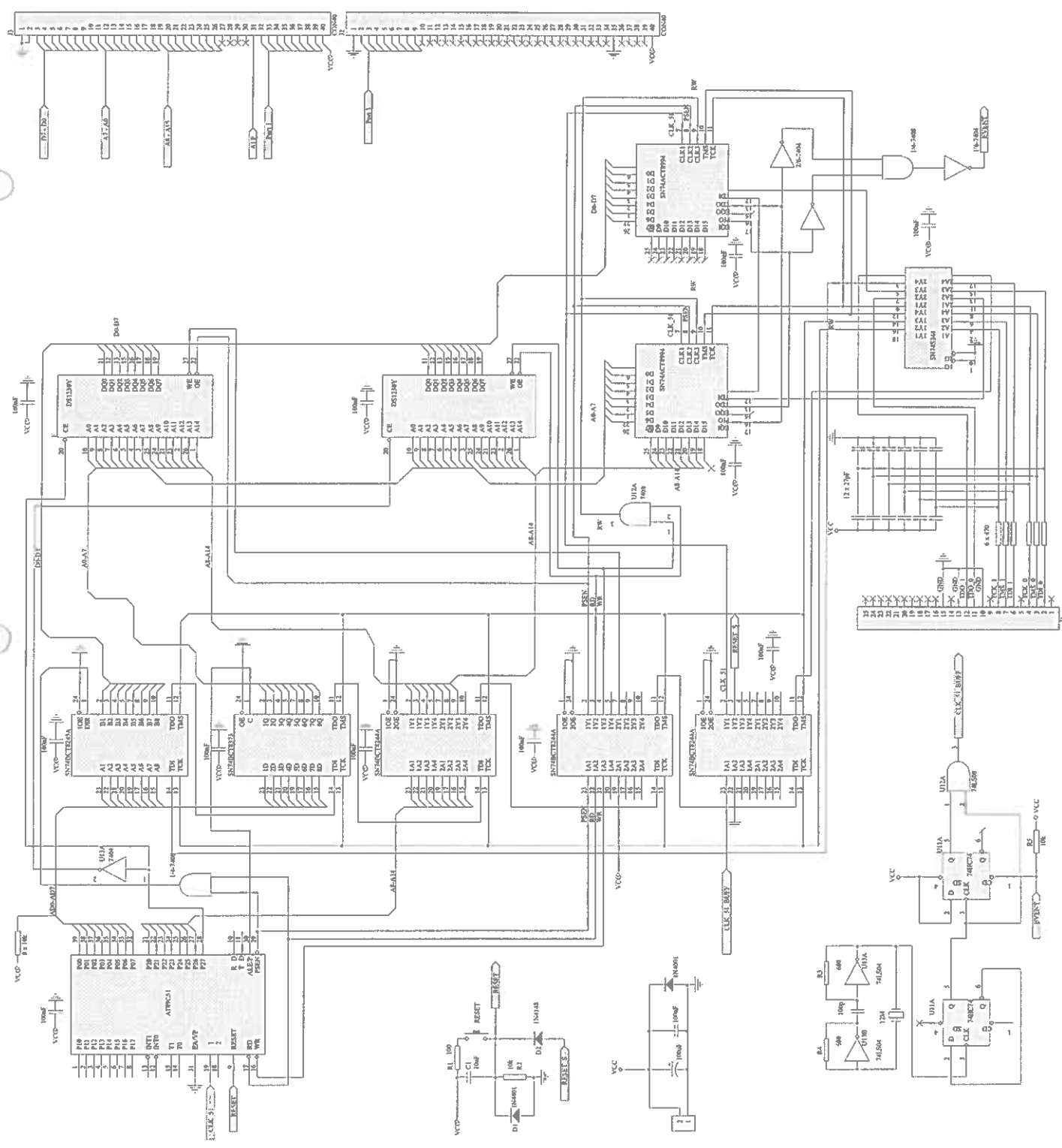


Table of pin connections for the 100-bit counter and shift register:

Pin	Signal
1	RESET
2	CLK
3	PRE
4	R
5	D
6	W
7	P00
8	P01
9	P02
10	P03
11	P04
12	P05
13	P06
14	P07
15	P08
16	P09
17	P10
18	P11
19	P12
20	P13
21	P14
22	P15
23	P16
24	P17
25	P18
26	P19
27	P20
28	P21
29	P22
30	P23
31	P24
32	P25
33	P26
34	P27
35	P28
36	P29
37	P30
38	P31
39	P32
40	P33
41	P34
42	P35
43	P36
44	P37
45	P38
46	P39
47	P40
48	P41
49	P42
50	P43
51	P44
52	P45
53	P46
54	P47
55	P48
56	P49
57	P50
58	P51
59	P52
60	P53
61	P54
62	P55
63	P56
64	P57
65	P58
66	P59
67	P60
68	P61
69	P62
70	P63
71	P64
72	P65
73	P66
74	P67
75	P68
76	P69
77	P70
78	P71
79	P72
80	P73
81	P74
82	P75
83	P76
84	P77
85	P78
86	P79
87	P80
88	P81
89	P82
90	P83
91	P84
92	P85
93	P86
94	P87
95	P88
96	P89
97	P90
98	P91
99	P92
100	P93