# High Level Behavioural Modelling of Boundary Scan Architecture

S. S. A. Medhat
Bournemouth University

PhD 1993

# High Level Behavioural Modelling of Boundary Scan Architecture

Sa'ad Sabih Ahmed Medhat

Submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy under the conditions for the award of higher degrees of the Bournemouth University.

APRIL 1993

Bournemouth University

In Collaboration with:

Siemens Plessey Systems, Christchurch, Dorset

## ACKNOWLEDGEMENTS

The author wishes to express his thanks and gratitude to a number of people, who have given their time and resources to support the research in this thesis.

To Mr Alan Potton, Director of Studies, of the Bournemouth University, for his aid in academic matters relating to content presentation of this thesis.

To Professor Ted Pritchard, of Huddersfield University and to Mr Brian Wilkins of Southampton University, both for their invaluable contributions and suggestions through the research.

To Martin Linda and Toby Williams, of Siemens Plessey Systems, and Ian Black of Mentor Graphics, for their industrial advice and guidance.

To David Lincoln and Maurice Downing, Apollo Systems Group at Bournemouth University for trouble-shooting workstation problems.

The author's greatest thanks go, of course, to his family for their support, encouragement and understanding.

# INDEX


| | |
|---|---|
| SUMMARY | I |
| RATIONALE | II |
| INTRODUCTION | IV |
| AIM | VI |
| OBJECTIVES | VI |
| THESIS TAXONOMY | VII |

***CHAPTER 1***
***OVERVIEW OF TEST TECHNOLOGY PRIOR TO JTAG***

| | | |
|---|---|---|
| 1.0 | INTRODUCTION | 1-1 |
| 1.1 | DEFINITIONS | 1-2 |
| 1.2 | BACKGROUND | 1-2 |
| 1.3 | TEST TECHNOLOGY PRIOR TO IEEE 1149.1 STANDARD | 1-7 |
| 1.4 | COST OF TEST | 1-9 |

***CHAPTER 2***
***DESIGN FOR TEST AND TEST TECHNIQUES***

| | | |
|---|---|---|
| 2.0 | INTRODUCTION | 2-1 |
| 2.1 | VLSI TEST PROCESS | 2-2 |
| 2.2 | VLSI TEST PHASES | 2-2 |
| 2.3 | THE DEFINITION OF A TESTABLE BLOCK | 2-3 |
| 2.4 | DESIGN TECHNIQUES FOR TESTABILITY | 2-4 |
| 2.5 | AD-HOC DESIGN TECHNIQUES FOR TESTABILITY | 2-5 |
| 2.5.1 | INITIALISATION | 2-7 |
| 2.5.2 | A METHOD OF TESTING COUNTERS | 2-7 |
| 2.5.3 | REDUCING THE TEST PIN COUNT | 2-9 |
| 2.6 | AD-HOC TECHNIQUES - FOR AND AGAINST | 2-10 |
| 2.7 | STRUCTURED DESIGN TECHNIQUES FOR TESTABILITY | 2-11 |
| 2.7.1 | PRINCIPLES OF INTERNAL SCAN PATH DESIGN | 2-13 |
| 2.8 | SCAN PATH TESTING - FOR AND AGAINST | 2-15 |
| 2.9 | BUILT-IN SELF-TEST AND STRUCTURED DESIGN FOR TEST | 2-16 |
| 2.10 | BUILT-IN SELF-TEST - FOR AND AGAINST | 2-18 |
| 2.11 | COMBINATION OF SCAN PATH AND BUILT-IN-SELF-TEST TECHNIQUES | 2-20 |
| 2.12 | HYBRID DESIGN FOR TEST - FOR AND AGAINST | 2-20 |
| 2.13 | TEST ACCESS AND BOUNDARY SCAN | 2-22 |
| 2.14 | BOUNDARY SCAN AND IN-SYSTEM TESTING | 2-27 |
| 2.15 | BOUNDARY SCAN AND SYSTEM TEST INTERFACE | 2-27 |
| 2.16 | BOUNDARY SCAN AND SOFTWARE ELEMENTS | 2-28 |
| 2.17 | ADVANTAGES OF BOUNDARY SCAN PATH | 2-29 |
| 2.18 | CONCLUSIONS | 2-29 |

***CHAPTER 3***
***REVIEW OF BOUNDARY SCAN DESIGN***


| | | |
|---|---|---|
| 3.0 | INTRODUCTION | 3-1 |
| 3.1 | TRENDS IN THE ELECTRONICS INDUSTRY | 3-2 |
| 3.2 | THE STANDARDISATION EFFORT | 3-3 |
| 3.3 | IMPLEMENTATION OF BOUNDARY SCAN ARCHITECTURE INTO INTEGRATED CIRCUITS | 3-3 |
| 3.4 | IEEE 1149.1 CONFORMANCE TESTING | 3-7 |
| 3.5 | BOUNDARY SCAN & BUILT IN SELF TEST | 3-8 |
| 3.6 | ADAPTATION TO CAE TOOLS | 3-9 |
| 3.7 | BSA AND TEST PATTERN GENERATION | 3-9 |
| 3.8 | BSA AND ATE | 3-10 |
| 3.9 | BSA AND POWER SUPPLY TERMINALS | 3-14 |
| 3.10 | BSA AND ANALOG COMPONENTS | 3-15 |
| 3.11 | SYSTEM COMPANY APPLICATIONS OF BSA | 3-15 |
| 3.12 | BSA AND RELATED DEVELOPMENTS | 3-16 |
| 3.13 | VHDL FOR CAD/CAE/CAT | 3-16 |
| 3.14 | VHDL FEATURES FOR CAD/CAE/CAT | 3-21 |
| 3.15 | IN CONCLUSION - THE PROPOSED PROJECT | 3-22 |


***CHAPTER 4***
***VHDL DESIGN AND MODELLING TECHNIQUES***


| | | |
|---|---|---|
| 4.0 | INTRODUCTION TO VHDL | 4-1 |
| 4.1 | DEFINITION OF VHDL | 4-2 |
| 4.2 | OPEN - SYSTEM DESIGN AUTOMATION ARCHITECTURE | 4-2 |
| 4.3 | VHDL AND THE ASIC DESIGN PROCESS | 4-2 |
| 4.4 | VHDL DESIGN HIERARCHY | 4-5 |
| 4.5 | VHDL MODES OF OPERATION | 4-9 |
| 4.6 | VHDL STRUCTURES | 4-10 |
| 4.7 | VHDL DESIGN HIERACHY AND DATA BASE | 4-10 |
| 4.8 | DESIGN DESCRIPTION METHODS | 4-12 |
| 4.8.1 | STRUCTURAL DESCRIPTION | 4-13 |
| 4.8.2 | BEHAVIORAL DESCRIPTION | 4-16 |
| 4.8.3 | DATA FLOW DESCRIPTION | 4-19 |
| 4.9 | BEHAVIORAL MODELLING OF 4 BIT MULTIPLIER | 4-20 |
| 4.9.1 | VHDL BEHAVIOURAL MODEL OF MULTIPLIER | 4-22 |
| 4.9.2 | SIMULATION AND TEST VECTORS | 4-22 |
| 4.10 | CONCLUSIONS | 4-23 |

***CHAPTER 5***
***THE IEEE 1149.1 BOUNDARY SCAN ARCHITECTURE***


| | | |
|---|---|---|
| 5.0 | INTRODUCTION | 5-1 |
| 5.1 | JTAG BOUNDARY SCAN ARCHITECTURE TEST MODES | 5-2 |
| 5.2 | JTAG BOUNDARY SCAN ARCHITECTURE MAIN COMPONENTS | 5-5 |
| 5.3 | THE TAP | 5-7 |
| 5.4 | BOUNDARY SCAN | 5-11 |
| 5.4.1 | BOUNDARY LOOP 0 RULES | 5-14 |
| 5.4.2 | THE TEST REGISTER TEST MODES | 5-15 |
| 5.4.2.1 | RUN | 5-16 |
| 5.4.2.2 | HOLD DATA | 5-17 |
| 5.4.2.3 | SHIFT DATA | 5-17 |
| 5.4.2.4 | TEST PSEUDO-RANDOM GENERATE | 5-18 |
| 5.4.2.5 | TEST SIGNATURE-ANALYSE | 5-18 |
| 5.4.2.6 | TEST SLIDE | 5-18 |
| 5.4.2.7 | TEST HOLD | 5-18 |
| 5.5 | INSTRUCTION REGISTER | 5-20 |
| 5.5.1 | JTAP INSTRUCTION REGISTER | 5-20 |
| 5.5.2 | RESERVED JTAG INSTRUCTIONS | 5-23 |
| 5.6 | BYPASS REGISTER | 5-24 |
| 5.7 | DEVICE IDENTIFICATION REGISTER | 5-25 |
| 5.8 | TEST ACCESS PORT (TAP) | 5-26 |
| 5.8.1 | JTAG SIGNALS (JCK,TMS,TDI,TDO,TRST) | 5-27 |
| 5.9 | RECOMMENDATIONS | 5-32 |
| 5.9.1 | HOW TO LAY OUT THE TEST REGISTER | 5-32 |
| 5.9.2 | PSEUDO RANDOM TEST | 5-33 |
| 5.9.3 | AUTONOMOUS SELF-TEST (RUNBIST) | 5-34 |
| 5.10 | CONCLUSIONS | 5-34 |


***CHAPTER 6***
***HIGH LEVEL VHDL MODELLING OF BOUNDARY SCAN ARCHITECTURE***


| | | |
|---|---|---|
| 6.0 | INRODUCTION | 6-1 |
| 6.1 | BACKGROUND | 6-2 |
| 6.2 | MODELLING OF THE JTAG ARCHITECHTURE | 6-3 |
| 6.3 | JTAG BOUNDARY SCAN ARCHITECTURE TIMMING AND PERFORMANCE ISSUES | 6-4 |
| 6.3.1 | CLOCK OPERATION | 6-4 |
| 6.3.2 | SET-UP AND HOLD TIMES | 6-4 |
| 6.3.3. | PROPAGATION DELAY OF SIGNALS | 6-5 |
| 6.3.4 | TDO-TDI INTERFACE | 6-7 |
| 6.4 | VHDL MODELS FOR BOUNDARY SCAN ARCHITECTURE | 6-7 |
| 6.5 | TAP CONTROLLER MODEL | 6-8 |
| 6.6 | INSTRUCTION REGISTER MODEL | 6-13 |
| 6.7 | INSTRUCTIOM DECODER MODEL | 6-16 |
| 6.8 | BYPASS REGISTER MODEL | 6-20 |
| 6.9 | BOUNDARY SCAN REGISTER MODEL | 6-22 |
| 6.10 | IDENTIFICATION REGISTER MODEL | 6-27 |
| 6.11 | MULTIPLEXER MUX_1 MODEL | 6-29 |
| 6.12 | MULTIPLEXER MUX_2 MODEL | 6-31 |
| 6.13 | SERIAL OUTPUT BUFFER MODEL (TDO) | 6-32 |
| 6.14 | CONCLUSIONS | 6-34 |

## *CHAPTER 7*
## *THE PARSING AND INSERTION ALGORITHM*


| | | |
|---|---|---|
| 7.0 | INTRODUCTION | 7-1 |
| 7.1 | AN OVERVIEW OF THE HIGH LEVEL PARSING AND INSERTION ALGORITHM | 7-2 |
| 7.2 | THE VCP PARSER OPERATION | 7-3 |
| 7.3 | THE DEVELOPMENT OF VCP-VERSION 1 | 7-4 |
| 7.4 | EXAMPLE OF THE OPERATION OF VCP-VERSION 1 | 7-6 |
| 7.5 | LIMITATION OF VCP-VERSION 1 | 7-9 |
| 7.6 | THE DEVELOPMENT OF VCP-VERSION 2 | 7-10 |
| 7.7 | EXAMPLE OF THE OPERATION OF VCP-VERSION 2 | 7-15 |
| 7.8 | LIMITATION OF VCP-VERSION 2 | 7-16 |
| 7.9 | THE INSERTION ALGORITHM | 7-17 |
| 7.10 | DEMONSTRATION OF THE INSERTION ALGORITHM | 7-22 |
| 7.11 | CONCLUSIONS | 7-25 |


## *CHAPTER 8*
## *COST IMPLICATIONS*


| | | |
|---|---|---|
| 8.0 | COST OF JTAG | 8-1 |
| 8.1 | AREA COST | 8-2 |
| 8.2 | PIN, CELL CONNECTIONS AND POWER COSTS | 8-3 |
| 8.3 | DELAY COST | 8-4 |
| 8.4 | DESIGN TIME COST AND COST BENEFIT OF THE TOOL | 8-4 |


## *CHAPTER 9*
## *OVERALL CONCLUSIONS*


| | | |
|---|---|---|
| 9.0 | THE NOVEL APPROACH OF THE PROJECT | 9-1 |
| 9.1 | ACHIEVEMENT OF AIM | 9-3 |
| 9.2 | ACHIEVEMENT OF OBJECTIVES | 9-4 |


## *CHAPTER 10*
## *FUTURE WORK*


| | | |
|---|---|---|
| 10.0 | INTRODUCTION | 10-1 |
| 10.1 | GRAPHICAL REPRESENTATION OF VHDL | 10-2 |
| 10.1.1 | GRAPHICS HARDWARE DESCRIPTION LANGUAGES GHDLS | 10-5 |
| 10.1.2 | GRAPHICS TO VHDL AUTOMATIC CONVERTER | 10-6 |
| 10.2 | ANALOGUE IMPLEMENTATION | 10-8 |
| 10.2.1 | ANALOGUE TEST APPROACH USING 1149.3 AND 1149.4 SUBSETS | 10-8 |
| 10.3 | LINKING THE ENVIRONMENT TO SYSNTHESIS TOOLS | 10-9 |
| 10.4 | THE ARTIFICIAL INTELLIGENCE ROLE | 10-11 |


## *APPENDICES*

# SUMMARY


This project involves the development of a software tool which enables the integration of the IEEE 1149.1/JTAG Boundary Scan Test Architecture automatically into an ASIC (Application Specific Integrated Circuit) design.

The tool requires the original design (the ASIC) to be described in VHDL-IEEE 1076 Hardware Description Language.

The tool consists of the two major elements:

i) A parsing and insertion algorithm developed and implemented in 'C';

ii) A high level model of the Boundary Scan Test Architecture implemented in 'VHDL'.

The parsing and insertion algorithm is developed to deal with identifying the design Input/Output (I/O) terminals, their types and the order they appear in the ASIC design. It then attaches suitable Boundary Scan Cells to each I/O, except power and ground and inserts the high level models of the full Boundary Scan Architecture into the ASIC without altering the design core structure.

## RATIONALE

The concepts of Boundary Scan, and its embodiment in the IEEE standard 1149.1 (JTAG), are now a fundamental part of Design-for-Test. The success of the Boundary Scan Architecture (BSA) ultimately depends on its popularity with Integrated Circuit (IC) users, rather than semiconductor manufacturers. However, chip and board manufacturers wanting to use JTAG are confronted with the lack of software tools to support them. In addition, the unavailability of generic models of the boundary scan test architecture has often hindered and dissuaded designers from developing BSA in accordance with the IEEE standard, and then including it into their designs. It is also worth noting that although some ASIC vendors have started to provide a set of boundary scan test cells in their libraries, they have failed in providing the associated test vectors. Furthermore, the designer is required to be able to understand the IC manufacturer configuration of the standard, and then include the necessary test cells in accordance with the guidelines outlined in the specifications of the IEEE standard. The other factor that ASIC/System Designers require when contemplating a new design is the ability to try out ideas at the behavioural/algorithmic level, and to explore a particular test strategy.

It is therefore suggested that there is a need for a facility to make JTAG architecture available in a behavioural format which is both easily applied and economical in simulation time.

The difficulty that arises in developing high-level models is that the testability features such as internal scan, and BIST (Built-in-self-test) are normally confined to the structural level. This is when internal state information is encoded into individual bits as represented by the flip-flops on the internal scan path(s), and the test signals have precise structural destinations inside the circuit.

Thus, the question that arises is how to back-annotate the high-level models developed presumably in the earlier steps of the top-down process with the test information that becomes known at the lower levels. Once such accurate high-level models are available, the testability features intended for structural testing could be used in the development of simplified functional tests of the system to a specific internal state. In the normal (mission) mode, the system could then be taken through a number of state transitions to verify a particular functional feature (eg asynchronous coupling to verify correct time / synchronisataion), and the resulting state could then be scanned out for verification.

This research project involves the development of a parameterised behavioural model of testability features as specified by the proposed IEEE 1149.1 standard. The IEEE Hardware Description Language standard (VHDL 1076-1987) will be used to describe the different models. The project will examine the use of a suitable VHDL behavioural modelling style that will allow a consistent integration of testability features to be extracted from the structural level, and subsequently included in the behavioural models of the components. It will also develop an algorithm which enables the integration of the BSA models into an ASIC design.

## INTRODUCTION

As the density of integrated circuits mounted on a printed circuit board increases, the problem of testing these boards and systems increases too. Manufacturers typically use in-circuit and functional board test systems to detect defects in their products.

An in-circuit test was originally designed to verify the goodness of connections and the operation of components on a board. Conversely, a functional tester verifies the function of the entire board/system through its edge connectors. The board is thus tested as close to the normal operating conditions as possible.

In-circuit test techniques however, are faced with increasing difficulties due to surface-mount technology and multi-layer boards. Whilst the functional test technique is better in coping with these problems, it carries a penalty of requiring the generation of comprehensive and complex test programs.

Despite the above difficulties, the majority of components available on the open market offer only Ad-hoc facilities to ease the testing problem. The JTAG group (Joint Test Action Group) initially provided standard and structured testability features on integrated circuits to simplify testing of boards and systems. The JTAG effort was continued by the IEEE standardisation committee, which resulted in the proposed testability standard 1149. It covers different types of circuits and test techniques.

This research project concentrates on the 1149.1 portion of the proposed standard, which is the outgrowth of the JTAG work and relates to the testing of synchronous digital circuits.

The aim of the standard is to allow different manufacturers to provide testability features on their integrated circuits in such a way that boards constructed using chips from different sources could form the necessary scan paths and communicate test control information on the board. It is foreseen that eventually the majority of the off-the-shelf components would contain such features.

Another standardisation effort is being expended by IEEE to provide common means for describing the behaviour/structure of integrated circuits, boards and systems using the IEEE 1076 VHDL language.

It is possible that components available on the market could be supplied with their models written in VHDL to aid in the development of models of systems using the components. This would not only reduce the duplication of work in developing models, but also help with the evolution of the product and its maintenance.

The models supplied with such components would most likely be behavioural rather than structural, so that the proprietary nature of designs would be protected. In addition, only such high-level models would be useful in modelling complex systems consisting of the interconnects of the components, as otherwise the verification of these systems by simulation would become too slow for practical purposes. It requires the accurate high-level models to be developed for components, with all of the behavioural features that these components exhibit. Therefore, all testability features that are provided inside the components must be included.

This research project takes advantage of the two IEEE standards and develops the required parameterised VHDL models of JTAG together with a mechanism for inserting the BSA (Boundary Scan Architecture) into an ASIC design in a semi-automatic way.

## AIM

To design an HDL modelling tool that hides structural details of JTAG 1149.1 from the ASIC designer, and enables the insertion of the Boundary-Scan Architecture in a semi-automatic way.

## OBJECTIVES

a) To create a parameterised behavioural model of the IEEE 1149.1 Boundary Scan Architecture using VHDL IEEE 1076 standard.

b) To integrate the Boundary Scan Architecture onto an ASIC design in a semi-automatic way.

c) To devise a strategy for mapping between levels of description of a design.

## THESIS TAXONOMY

The thesis consist of 10 chapters organised as follows:

***CHAPTER 1: Overview of Test Technology Prior to JTAG***

It provides an overview of the test technology prior to IEEE 1149.1. The background to the problem of test is briefly described. It explains the non-viability of applying exhaustive testing to VLSI systems. The commercial and engineering considerations associated with design for test at various levels along the system's integration path are also highlighted. In addition, the need for Accessibility to enable system testing is identified.

***CHAPTER 2: Design for Test and Test Techniques***

It reviews some of the ideas behind VLSI circuit testing and highlights the problems of testing both combinational and sequential logic. The aim of circuit testing is identified including both functional and post fabrication testing. Both the Ad-Hoc and the Structured Approaches to Design for Testability are described. The advantages and disadvantages of both techniques are outlined.

***CHAPTER 3: Review of Boundary Scan Design***

It reviews the recent literature on Boundary Scan Design. A historical background to the development of JTAG and the IEEE 1149.1. Boundary Scan Architecture is provided. In addition, this chapter examines the suitability of VHDL for use in the Computer Aided Design and Test environment. It therefore identifies the motivating factors behind the implementation of this project. In doing so it also identifies the originality of this work and where this proposed research relates to recent development in the field.

***CHAPTER 4: VHDL Design and Modelling Techniques***

It reviews the main features of VHDL Hardware Description Language. It identifies where VHDL could be used at the various stages of the system development cycle. VHDL design hierarchy and data base structure is discussed. VHDL Description styles are also highlighted. A full design example and simulation using a 4 bit serial multiplier is fully described.

***CHAPTER 5: The IEEE 1149.1 Boundary Scan Test Architecture***

This chapter describes the main operation of the main components of the IEEE JTAG standard in a structural way. The architecture is then tested with a simple application logic. Full simulation results are included.

***CHAPTER 6: High Level VHDL Modeling of Boundary Scan Architecture***

This chapter describes the use of Hardware Description Language VHDL, to describe behaviourally the IEEE 1149.1 Boundary Scan Test Architecture.

High level VHDL models of the JTAG Boundary Scan Architecture (BSA) are developed and tested, using both Mentor Graphics - workstation-based environment and View Logic's PC-based environment.

The use of VHDL facilities such as 'PACKAGE and 'LIBRARY' for declaring timing elements associated with each model are highlighted.

***CHAPTER 7: The Parsing and Insertion Algorithm***

This chapter describes the design, development and operation of a high level *parser/insertion* algorithm developed in "C". The parsing phase deals with identifying the mode of the input/output terminals as defined in the entity description of the application logic. The insertion phase deals with attaching the appropriate Boundary Scan Cells to the ASIC.

The chapter also integrates the rest of the pre-processed generic Boundary Scan Architecture- the TAP, Instruction Register and Decoder, Bypass Register, Identification Register (Optional) and the Multiplexers- into the ASIC design in a semi-automatic way. Examples demonstrating the operation of the algorithm are given.

***CHAPTER 8: Cost Implications***

Cost and performance implications resulting from the inclusion of JTAG into an ASIC design are identified and discussed.

***CHAPTER 9: Overall Conclusions***

Concluding discussion relating to the achievement of aims and objectives is presented. This chapter also highlights the novel outcome of this project.

***CHPATER 10: Future Work***

This chapter examines the potential for future development on the work which has been carried out.

# CHAPTER 1
# OVERVIEW OF TEST TECHNOLOGY PRIOR TO JTAG

## 1.0 INTRODUCTION

Integrated Circuit technology is now moving from Very Large Scale Integration (VLSI) to Ultra Large Scale Integration (ULSI). This major increase in gate count has brought about a decrease in gate costs along with improvements in performance. All of these attributes of ULSI are welcomed by industry. A problem which has never been adequately solved however, is that of determining in a cost effective way whether a component, a module or a board has been manufactured correctly.

The ability to test is a fundamental problem when designing complex systems, in particular VLSI and ULSI circuits. It is a requirement which must be integrated at the earliest stages of the design cycle from the specification level through functional design to the structural circuit design.

This chapter provides an overview of the test technology prior to the IEEE 1149.1.[JTAG 90] It briefly describes the background to the problem of test and explains the non-viability of applying exhaustive testing to VLSI systems. It also highlights the commercial and engineering considerations associated with design for test at various levels along the systems' integration path. In addition, it identifies the need for accessibility to enable system testing.

## 1.1 DEFINITIONS

This section clarifies some of the terms associated with test. [WILL 83],[GREE 86], [AGRA 82],GRAS 80]

**TESTING** *is the Process of Determining the Absence or Presence and in some cases the Location, of One or More Design Flaws, Manufacturing Defects, or Field Defects in a Chip, Board or System.*

**TESTABILITY** *is a Design Characteristic which allows the Status (Operable, Inoperable, or Degraded) of an Item to be Determined and the Isolation of Faults within the Item to be Performed in a Timely Manner, so as to Reduce Both Test Time and Cost.*

**DESIGN FOR TESTABILITY (DFT)** *is a Deliberate design Effort(s) Expended to Ensure that Unit is Testable.*

## 1.2 BACKGROUND

In the last few years CMOS technology has become increasingly dominant for realizing Ultra Large Scale Integrated Circuits. The popularity of this technology is due to its high density and low power requirement. The ability to realise very complex circuits on a single chip has brought about a revolution in the world of electronics and computers. [MAUN 84], [McAN 87], [PARK 86].

Testing has become a very time consuming process. The "Problem" is considered to be the resultant of many of the factors listed below:

a. Increasing Circuit Complexity.

Increasing IC Complexity results from increasing IC circuit densities and consequential gate to pin ratios. In addition, packaging can complicate the testing problem as a result of new packaging technology such as Surface Mount, Double Sided Boards, Multi-layer Boards, Conformal Coating and Multi-Chip Modules.

b. Increasing Test Generation and Fault Simulation Costs.

The problem with test pattern generation and fault simulation costs is related to the running time which is likely to be an exponential function of the number of gates. [TRIS 84] [BREU 80]

c. Expensive Automatic Test Equipment.

The disparity between the costs of the tester and the UUT (Unit Under Test) is substantial! Increasing circuit densities have helped and hindered. Using VLSI in ATE has lowered the tester cost. However, using VLSI in the UUT has increased the tester costs.

Other Contributors to the Problem of Testing include:

d. Poor design and test organisational interfaces.
e. Lack of design for testability.
f. Incompatibility within and between design and test CAE tools.
g. Lack of test requirements.
h. Lack of logic and fault simulation.
i. Test Engineer's lack of access to simulation or diagnostic vectors.
j. Schedule pressures.
k. Real estate constraints.
l. Performance penalties.

Logic designers as far back as the early 1960's were confronted with testing fairly complicated systems on PCBs. They did not have the sophisticated software/hardware we have today to aid them in this task. They had to resort to primitive methods such as exhaustive testing and/or littering their PCB's with test points, neither of which can be applied today to our more complex systems integrated on single chips.

Exhaustive testing can easily be seen to be non-viable if you consider some basic examples. For example, in a combinational circuit with n inputs there are $2^n$ possible input patterns necessary to step through the truth table. Thus: for n=4, we have 16 steps; n=8, 256 steps; n=16, 65536 steps and so on. To simply test the circuits functionally, the number of steps is seen to rise rapidly. Another more interesting example is to consider a circuit with 100 nodes each node of which may take on one of three states: good, stuck-at-0, stuck-at-1. There are therefore $3^{100}$ possible outcomes. If we arranged to test for all these, taking only 1μsec per test, then the complete test would take $3^{100} = 10^{47}$μsec = 3.2 x $10^{33}$ years, somewhat longer than the expected life of the earth!

For sequential circuits the problem grows even more. If m stored state devices (flip-flops) are present then there are $2^m$ possible internal states and thus with n inputs we would have to step through $2^{(n+m)}$ input patterns to do a full functional test. [GRAS 80], [GOEL 80]

The foregoing discussion has only considered logical function testing. Even if the transient behaviour of a circuit is controlled and separated from the functional testing, stepping through the truth tables is often not sufficient as some faults are pattern sequence dependent. If this is the case then strictly all possible transitions of input patterns should be covered.

As an example consider a 2-input circuit (such as a half-adder) where there are $2^2$ possible input patterns. Each of these 4 patterns can be followed by any one of the three remaining patterns and thus there are 4x3 transitions to investigate, three times the length of a simple truth table functional test. For a 3-input circuit (such as a full-adder) there are 8x7 transitions; for a 4-input circuit 16x15 transitions and so on.

The difficulties with exhaustive testing are insurmountable and techniques have been developed to reduce the task of testing to manageable proportions. [FUN 78]

A major contributor to testability is the *Accessibility* of internal circuit nodes via the Tester. Accessibility is impacted by such factors as :

- Whether the circuit contains storage elements.
- How many inverting levels of logic there are between I/O pins.
- The fan-in and fan-out within the circuit.
- The number of pins and test points.

Accessibility is comprised of two components: *CONTROLLABILITY* and *OBSERVABILITY.* [WILL 83][TRIS 84][AGRA 82]

**CONTROLLABILITY** : is the ability to establish a specific signal value at each node in a circuit by setting values on the circuits inputs.

**OBSERVABILITY** : is the ability to determine the signal value at any node in a circuit by controlling the circuit's inputs and observing its outputs.

The Characteristics of circuits with poor controllability include circuits that require a unique input pattern or a lengthy, complex pattern sequence to establish the state of each node. In addition, many types of circuits are inherently uncontrollable such as:

Decoders and selectors,
Circuits with feedback,
Serial sequential circuits,
Oscillators and clock generators,
Discriminators,
Electro-mechanical Transducer sensors, and
Regulators.

The types of circuits with poor observability include circuits that require a unique input pattern or a lengthy, complex sequence of input patterns to propagate the state of each node to the outputs of the circuit.

Many circuits are inherently unobservable, such as:

Sequential circuits,
Circuits with global feedback,
Embedded RAMs, ROMs, PLAs,
Concurrent error checking circuits, and
Circuits with redundant nodes.

## 1.3 TEST TECHNOLOGY PRIOR TO IEEE 1149.1 STANDARD

Over the years, automatic test equipment (ATE) used to test electronic products has evolved to cope with continued increases both in the number of integrated circuit packages used on as a single board and in the complexity of the integrated circuits (ICs) themselves. [MAUN 84][GREE 88] Typically, manufacturers of loaded boards will use high pin count in-circuit and functional board test systems, either separately or in sequence, to detect defects and to enable high quality levels in shipped products.

Using in-circuit test technique, tests are applied directly to individual components by back-driving their connections from other devices in the product. The objective is to apply the appropriate test sequence for the component regardless of the environment in which it is used.

Direct access is made to the components outputs to monitor the test results, enabling the function of each component in the circuit and interconnections between the various components to be checked. [BARD 82][BUDD 88] This method reduces the expense of test development for each circuit since the same test can be applied irrespective of where the IC is used. This is the case as long as an ICs functionality is not modified by externally wired connections (eg. by direct connection to power or ground). Clearly, the process requires extensive access to the circuit, because every connection must be driven and monitored directly to apply the test to individual components. This access is provided through a bed-of-nails interface in which spring-loaded probes are used to make contact with the interconnections on the PWB as shown in figure 1.1.

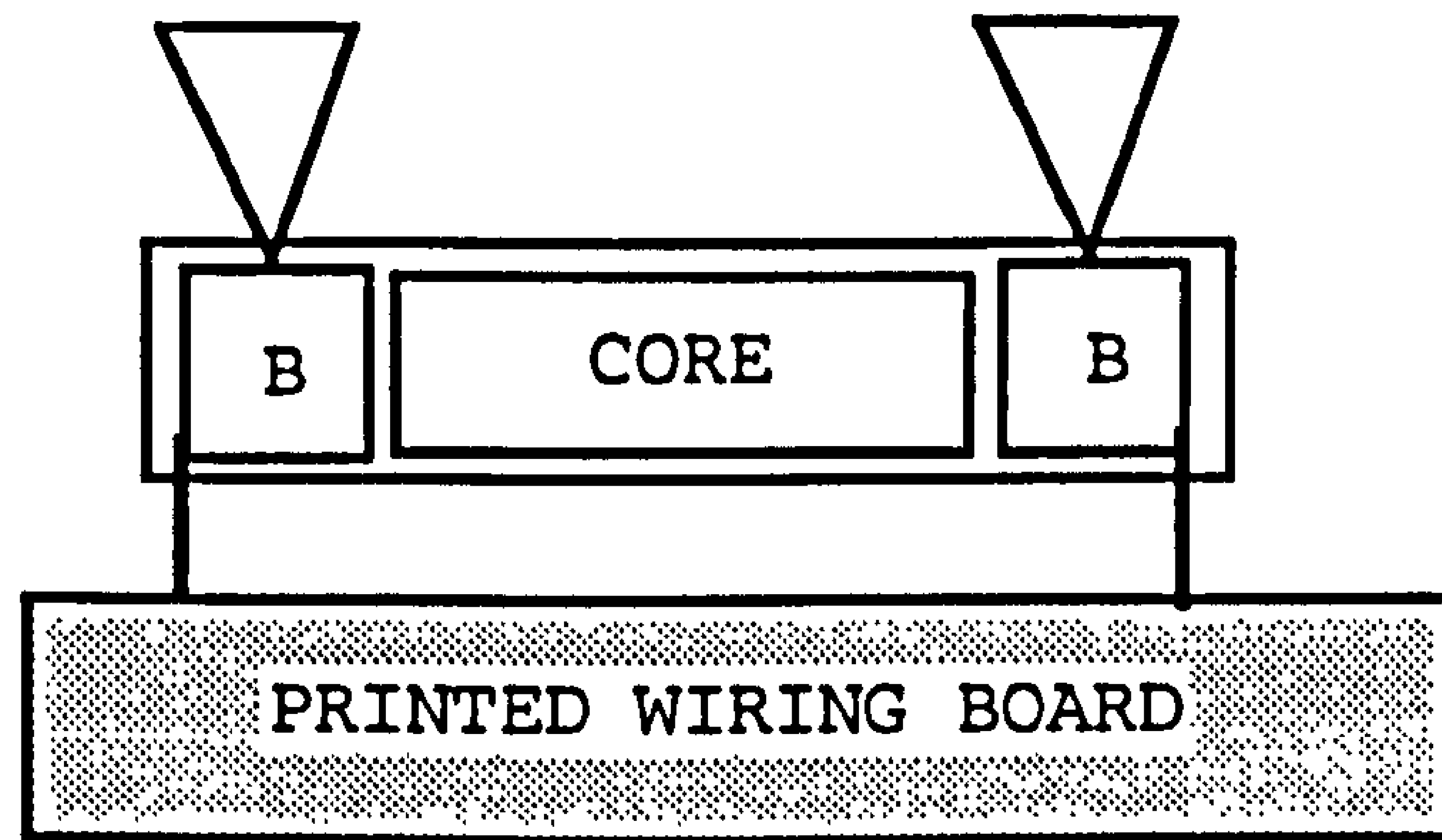



**Figure 1.1 In Circuit Test**

The principal interface in the functional test technique is used to apply test stimuli and to observe the responses which are read at normal terminations such as the edge connector. [WILL 83] [PARK 86] Access may also be made to connections internal to the loaded board, but this is more limited to monitoring, rather than to driving, the connection. In contrast to in-circuit testing, the functional test technique is able to confirm that the various components used to construct the product interact correctly, that the overall required function is achieved in process, and the correctness of both the components in the circuit and their interconnections is verified.

The achievement of a thorough test is however a difficult task since tests must be generated separately for each board. This task can be both time-consuming and extremely expensive, sometimes prohibitively so.[GOEL 80]

Due to the differences in operation and failure detection capability between in-circuit and functional test techniques, a common approach is to use the two techniques in sequence to achieve high quality test. Initial product screening is performed by using an in-circuit test system since this is able to rapidly detect and diagnose the most common failures in newly assembled boards. For example, those errors that have resulted from soldering mistakes and incorrect or wrongly inserted components can be detected in this way.

Once a loaded board has passed the screening test, it is passed forward to a functional test system where checks are made for more complex (and less frequent) failures caused by faulty interaction between components.

To allow the mix between the two test techniques to be more easily optimised for a given product, test equipment that supports both techniques within a single system has become more available recently.

## 1.4 COST OF TEST

There are commercial and engineering considerations when designing for test. [GOEL80][WILL 83] It is a general principle of electronics systems design that the further along the system integration path, the more expensive it is to replace a faulty component. Order of magnitude comparisons given in the following table and figure 1.2:

| Stage at which fault is detected | Cost of replacement |
|---|---|
| Die | £0.10 |
| Packaged chip | £1.00 |
| PCB | £10.00 |
| System at factory | £100.00 |
| System in operation | £1000.00 |

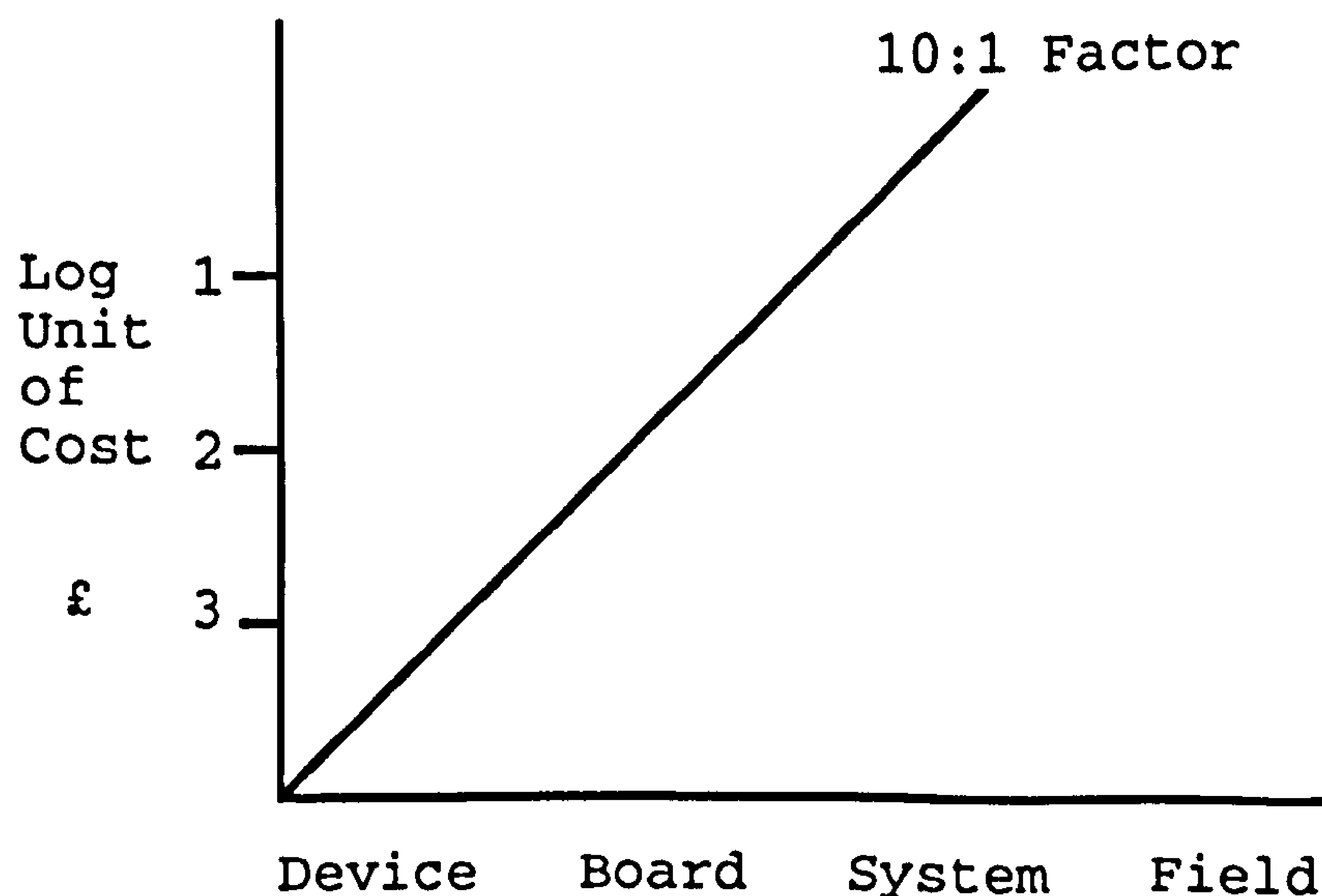



**Figure 1.2 Cost of Test**

The consequential costs of a chip failure during system operation, for example in a critical aircraft control system, can be orders of magnitude higher than these.

A set of test vectors (diagnostic patterns of 1s and 0s) which identify virtually all faults at the die stage provide great savings, since the chance of a post-fabrication fault developing is much lower than that of a fault being introduced during fabrication.

The strategy for test vector development needs to take a number of cost factors into account, including those of design, computer resources, fabrication, wafer test, packaging, component test, board test, system test, and field test. All of these need to be balanced in relation to the size of the production run and the figures in the table 1.2 of this section.[WILL 83][AGRA 82]

## REFERENCES


[JTAG 90] *IEEE Standard 1149.1-1990* "Test Access Port and Boundary Scan Architecture.

[WILL 83] T W Williams, et K P Parker, "Design for Testability ... A Survey?" IEEE Proc. of IEEE Vol 71, no 1 January 1983, pp 998-112.

[TRIS 84] E Trischler "An Integrated Design for Testability & Automatic Test Pattern Generation System: an Overview" the 21st Design Automation Conference 1984 IEEE pp 209-215.

[AGRA 82] V D Agrawal, M R Mercer "Testability Measures ...What do they Tell us?" Proc. of 1982 Test Conference 1982, pp 391-396.

[ARAB 86] J Arabian, "Test equipment-boards and Board Level Testing" Advanced in CAD for VLSI, Vol 5 VLSI Testing, edited by TW Williams, North Holland, pp 239-275, 1986.

[BARD 82] P H Bardell, W H McAnney, "Self-Testing of Multi-chip logic Modules", the IEEE Design Automation Conference, 1982 pp 309-313.

[BREU 80] M A Breuer and A D Friedman, "Functional Level Primitives in test Generation," IEEE Transactions on Computers Vol c-29, no 3, March 1980 pp 223-235.

[BUDD 88] W.O.Budde, "Modular Test processor for VLSI Chips and High Density PC Boards". IEEE Transactions on Computer Aided Design. Vol 7 No & 10 October 1988, pp 1118-1124.

[FUN 78] S Funastu,W Wakatsuki and A Yamada, "Designing digital Circuits with easily testable consideration, " Proc. of 15th ACM/IEEE Design Automation Conference 1978 pp 231-235.

[GOEL 80] P Goel, "Test Generation Cost Analysis and Projections" Proc of 17th Design Automation Conference, 1980 pp 77-84

[GRAS 80] J Grason, et A W Nagle "Digital Test Generation and Design for Testability", Proc of the 17th Design Automation Conference, 1980 pp 175-189.

[GREE 86] D Green, "Modern Logic Design', Electronic System Engineering Series, Addison - Wesley Publishing Co. In 1986.

[GRUE 88] M Gruetzner "Design for Testability for Wafer Scale Integration interconnect Systems, Design & Test Methodology" Int'l Test Conference 1988, IEE pp 146-152.

[MAUN 84] C Maunder, D Roberts et N Sinnadural, "Chip carrier based systems and their testability", Hybrid Circuits, No 5 Autumn 1984, pp 29-36

[McAN 87] W McAnney, P Bardell and J Savir, "Built-in Test for VLSI: pseudo random techniques" John Wiley & Sons, New York 1987.

[PARK 86] K P Parker, Hewlett-Packard, "Testability: Barriers to acceptance", IEEE Design & Test, October 1986 pp 11-15

# CHAPTER 2
# DESIGN FOR TEST AND TEST TECHNIQUES

## 2.0 INTRODUCTION

This chapter reviews some of the ideas behind VLSI circuit testing and highlights the problems of testing both combinational and sequential logic. The aim of circuit testing for both functional and post-fabrication is identified. The requirements for a testable block is also described. In addition, the chapter discusses some of the techniques for Design for Testability including Ad-Hoc and Structured approaches. It also identifies the advantages and disadvantages of the discussed techniques.

Test methods such as pseudo-random testing and deterministic pattern testing are described. The scan test method both internal and external is described in some detail. The chapter then identifies the benefits offered by IEEE 1149.1 Standard, when implementing a standard test interface to the unit under test.

## 2.1 VLSI TEST PROCESS

The process of VLSI circuit testing has two major facets:

a. test generation
b. test verification

Test generation is the process of enumerating stimuli for a circuit which will demonstrate its correct operation. Test verification is the process of proving that a set of tests are effective. To date, formal proof has been impossible in practice. Fault Simulation has been the best alternative, yielding a quantitative measure of test effectiveness. With the vast increase in circuit density, the ability to generate test patterns automatically and conduct fault simulation with these patterns has drastically waned.

As a result, some manufacturers are foregoing these more rigorous approaches and are accepting the risks of shipping a defective product. One alternative approach to addressing this problem is embodied in a collection of techniques known as "Design for Testability". Design for Testability, in the context of VLSI is gaining great importance.

## 2.2 VLSI TEST PHASES

In the complete Application Specific Integrated Circuit (ASIC) design and fabrication cycle, there are two test phases, as shown in figure 2.1:

- functional testing, to check whether the circuit conforms to its original specification.

- post-fabrication testing, to ensure that each die has been fabricated without any faults. These tests are used to check for differences between the operation of each individual die and an ideal chip.

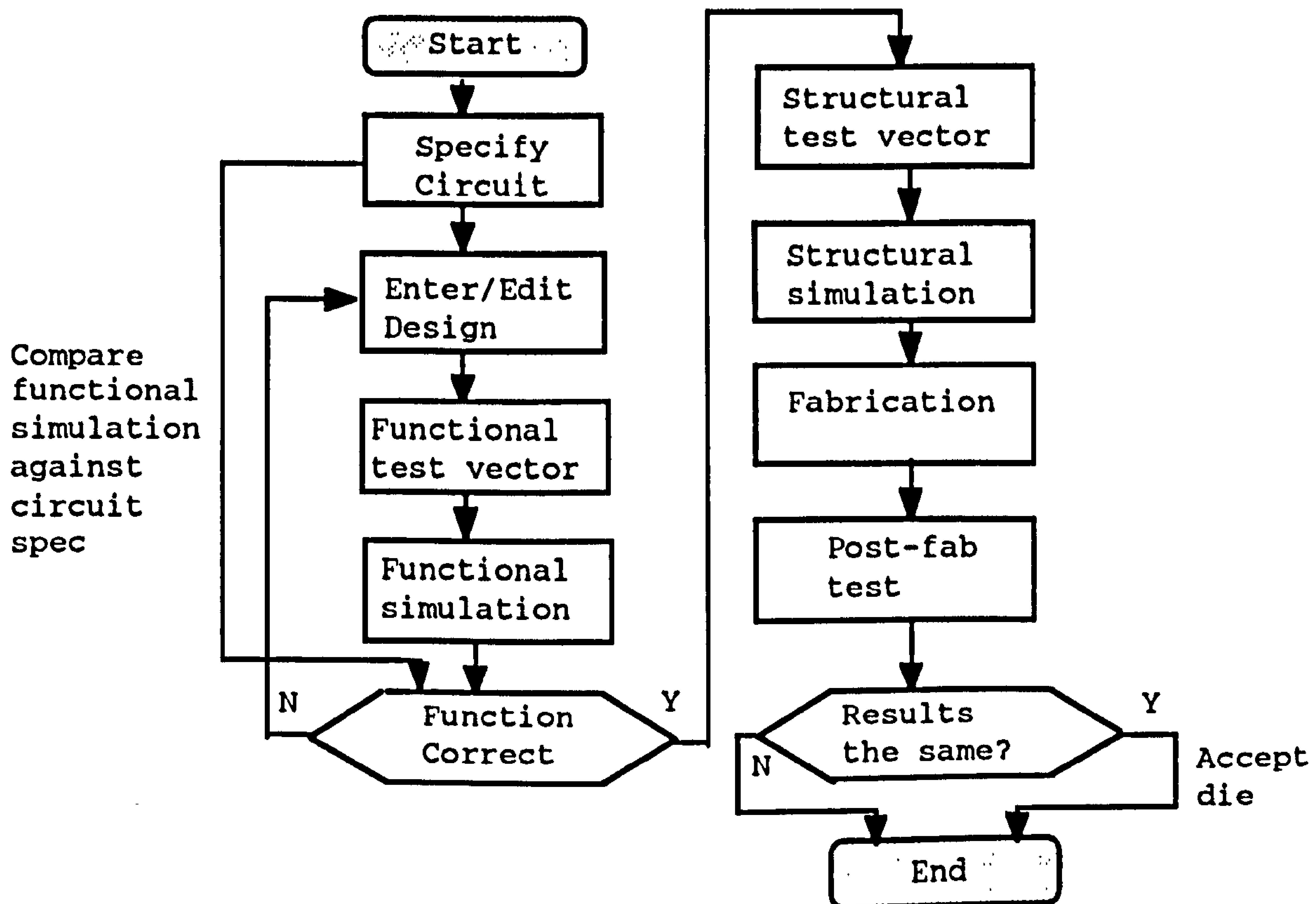



**Figure 2.1: Flow Chart Illustration of Various Test Stages**

## 2.3 THE DEFINITION OF A TESTABLE BLOCK

Before discussing the various techniques to achieve Design for Test, it is important to define what is a *testable block* in VLSI design.

A testable block is a section of a chip which can be isolated from the rest of the chip. It has a definable boundary between it and the rest of the chip. It also has a manageable amount of circuitry to test as a single section. In many instances the testable blocks will match the natural partitions and hierarchy of the design which is normally encouraged as a good design practice. A testable block many be any of :

- Combinational logic with no internal feedback,
- A single clocked pipeline state of circuitry,
- A paracell such as RAM and ROM, and
- Multiple pipeline stages.

It is possible to have several of the above sections connected together with test registers only at the inputs and outputs. The fewer the number of test registers used in a design, the more complex the test sequence that has to be used in the test registers in order to perform the complete section test. A sensible compromise has to be reached in order to work out how much circuitry can be placed in a testable block.

In non-test mode the test registers can be used as normal clocked registers. Thus when partitioning the design into testable blocks, it can be more economic to choose existing clocked registers in the design and replace them with clocked test registers. This dual function of the test register effectively reduces the amount of test circuitry overhead than might otherwise be required.

Test registers can be placed around the testable block, and a series of test sequences postulated that give adequate fault coverage. If the resulting test sequence is too lengthy and complicated, it may be necessary to split the testable block further into smaller blocks, to reduce this problem.

## 2.4 DESIGN TECHNIQUES FOR TESTABILITY

Design for Testability (DFT) techniques are divided into two categories. The first category is that of the 'ad hoc' techniques for solving the test problem. These techniques are used to solve a problem for a given design and are not generally applicable to all designs.

The second category is that of the 'structured' approaches. The 'structured' techniques are more generally applicable and usually involve a set of design rules by which the designs are implemented. The objective of a structured approach is to reduce the sequential complexity of a network to aid test generation and test verification. Figure 2.2 compares the two techniques in term of life cycle test cost.

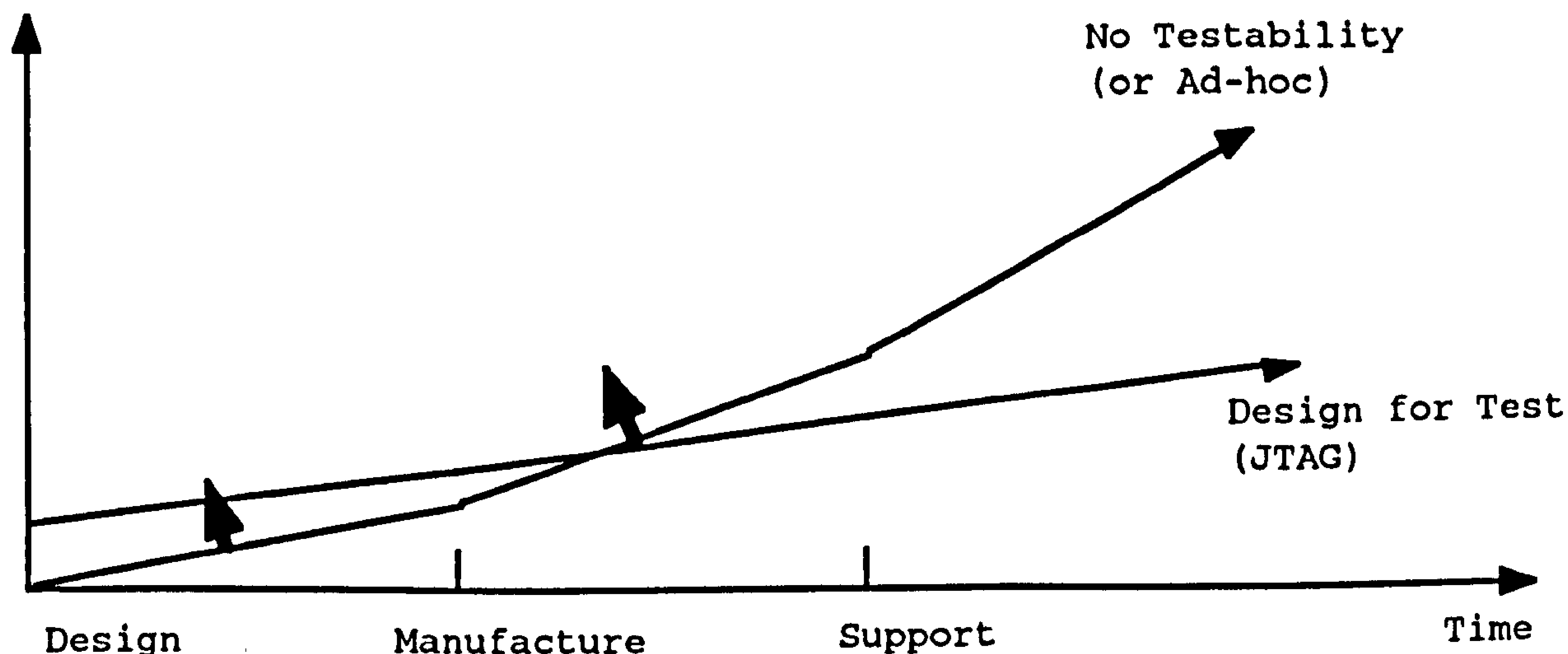



**Figure 2.2 Ad-hoc Vs Structured Test**

## 2.5 AD-HOC DESIGN TECHNIQUES FOR TESTABILITY

The first ad hoc approach is partitioning. Partitioning is the ability to disconnect one portion of a network from another portion of a network in order to make testing easier.

The second approach is that of adding extra test points and is used at the board level.

The third ad hoc approach is that of Bus Architecture Systems. This is similar to the partitioning approach and allows one to 'divide and conquer' - that is, to be able to reduce the network to smaller sub-networks which are much more manageable. These sub-networks are not necessarily designed with any design for testability in mind.

The fourth technique which bridges both the structured approach and the ad hoc approach is that of 'Signature Analysis'. Signature Analysis requires some design rules at the board level, but is not directed at the same objective as structure approaches are - that is, the ability to observe and control the state variables of a sequential machine.

Signature analysis mode can be used to collect a signature from circuitry driven by a pseudo-random sequence or slide test data. The output from a PR test is collected on a test register configured in TEST SIGNATURE ANALYSE mode. See fig 2.3.

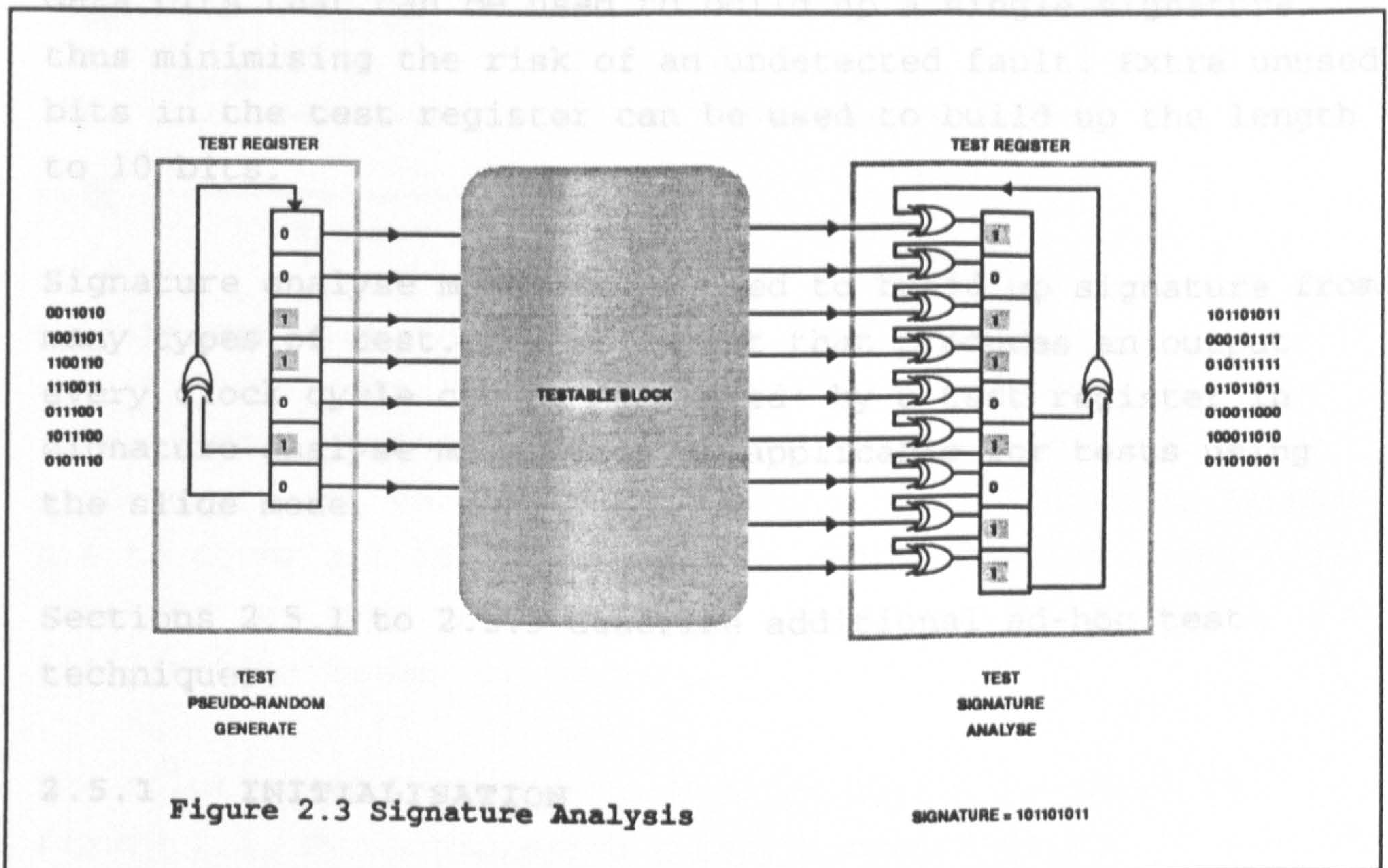



**Figure 2.3 Signature Analysis**

A faulty circuit produces a different signature. This is because the circuit behaves differently under the same pseudo-random input sequence, which is collected from the circuit outputs onto the test register as a corrupted signature.

The length of a test register determines the length of the signature. An N bit test register produces an N bit signature, this means that there are $2^N$ possible signature combinations. A fault in the circuit will produce a corrupted signature which can be any one of the $2^N$ possible values. It is possible for a faulty circuit to produce a corrupted signature which is exactly the same as the fault free signature and this would go undetected (fault aliasing). The probability of this happening is in a random circuit can be shown to be $1/(2^N)$.

Therefore it is strongly recommended that if a test register is going to be used in TEST SIGNATURE ANALYSE mode, that the length of the test register be made greater than or equal to 10 bits (ie = <1/1024). The larger the test register the more data bits that can be used to build up a single signature, thus minimising the risk of an undetected fault. Extra unused bits in the test register can be used to build up the length to 10 bits.

Signature analyse mode can be used to build up signature from many types of test. Any self-test that produces an output every clock cycle can be 'captured' by a test register in signature analyse mode. This is applicable for tests using the slide mode.

Sections 2.5.1 to 2.5.3 describe additional ad-hoc test techniques.

### 2.5.1 INITIALISATION

In order to be able to test the state of any circuit element, it must be initialised to a known value at the start of the test. This is one of the reasons for including a reset in all the DFT elements. The initial state of an element is immaterial for operational purposes but it must be reset before a structured test commences.

### 2.5.2 A METHOD OF TESTING COUNTERS

Consider the following example shown in figure 2.4. For an n-bit counter, the final state is only attained after the application of $2^n$ clock cycles. If this output feeds another m-bit counter, making it increment only once every $2^n$ clock cycles, the final state of the second counter is only attained after $2^{(n+m)}$ clock cycles. In such a situation, the first counter is generally a frequency divider and the second is a state counter.

For typical values of n(10) and m(4), the final state is only reached every $2^{14}$ (or 16384) clock cycles.

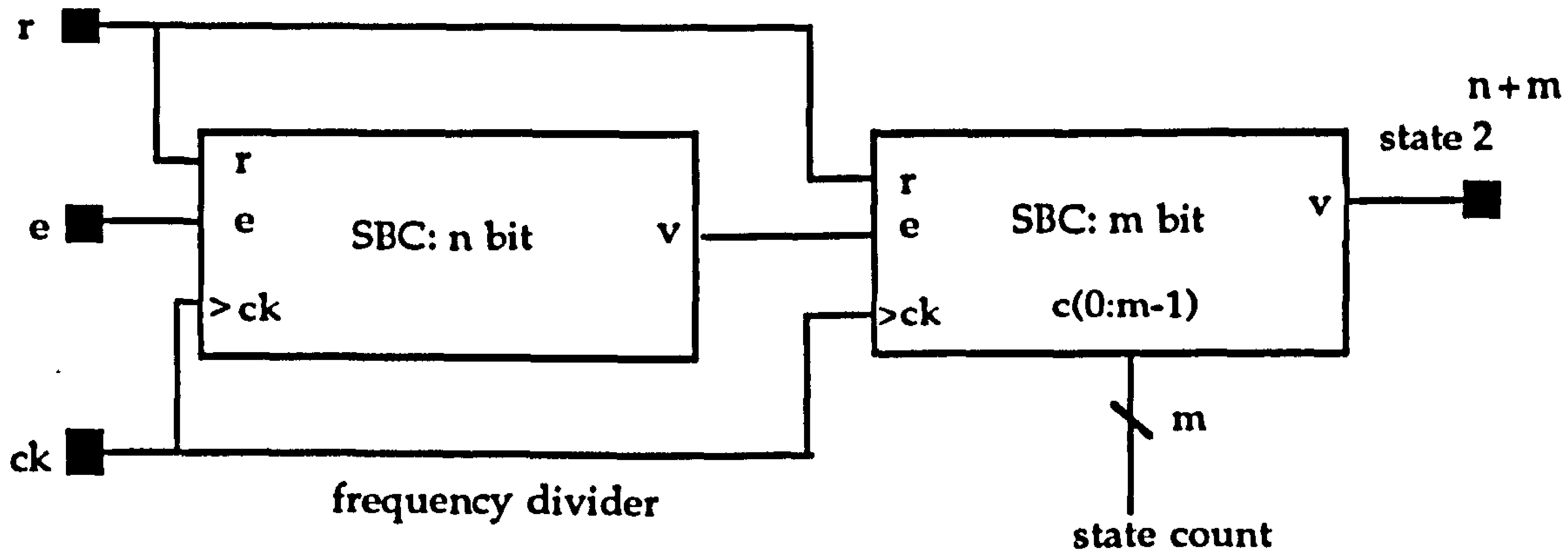



**Figure 2.4 Counter Chain**

It is almost impossible to generate an acceptable set of structural test vectors for a circuit of this nature. If they are to cover all the states of the second counter, they will take far too long to run through the simulator, or may exceed its limiting number of vectors.

The solution is to break the counter chain, as shown in figure 2.5, by multiplexing a test signal into the line which joins the two counters. The output from the first counter is also carried to a test output. This is the most general solution to the problem and creates three additional pads.

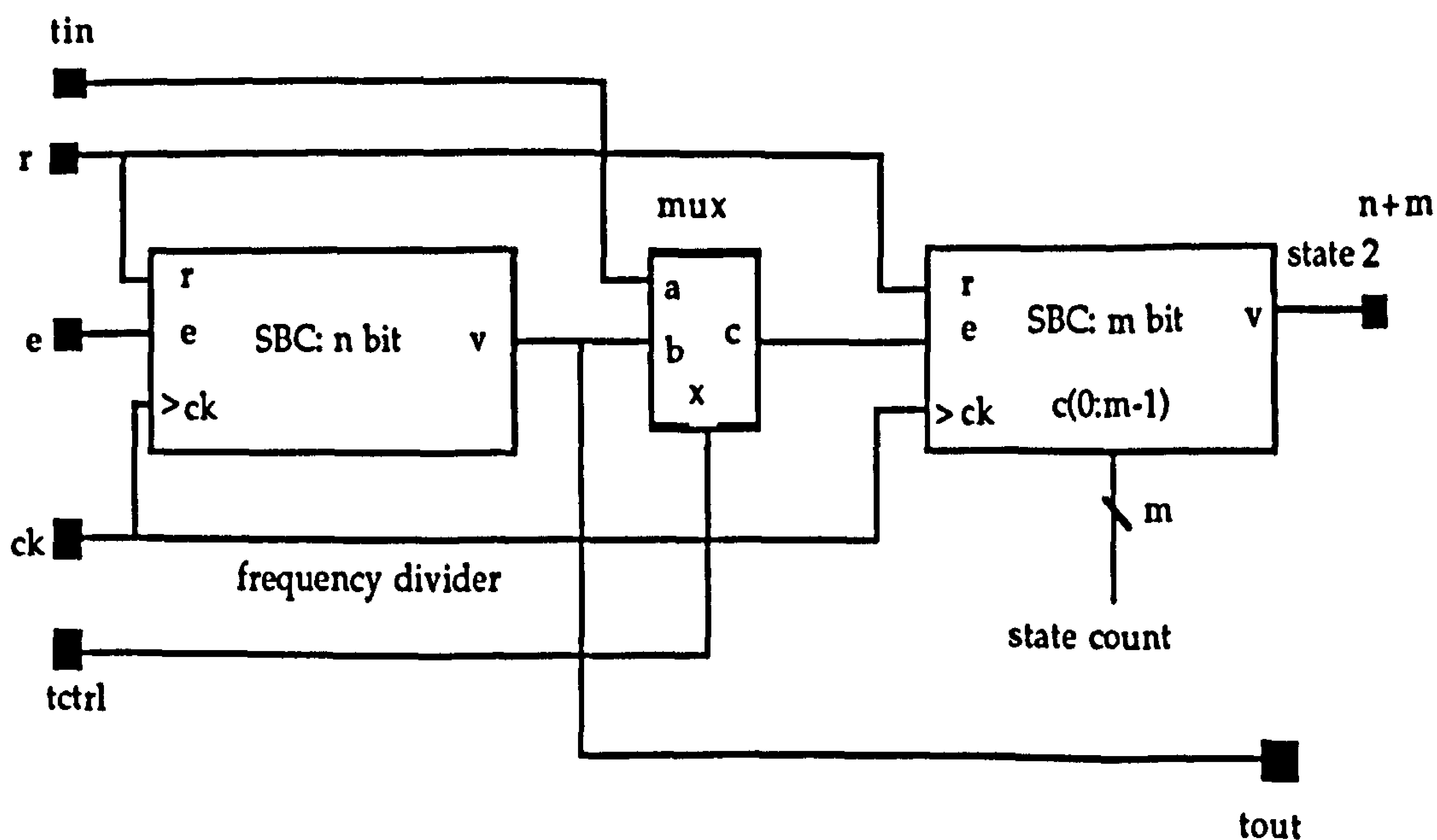



**Figure 2.5 Breaking a Counter Chain with Test Signals**

### 2.5.3 REDUCING THE TEST PIN COUNT

It is important to plan the DFT technique in such a way to minimise the additional number of input/output signals. A reduction in the number of the pads and the number of gates can sometimes be obtained by combining the test control and test data signals. Consider the following example in figure 2.6. When a counter chain is broken, the test link must eventually be driven to 1 when in test mode. If this is the case, the multiplexer, test control and test data lines can be replaced by an OR gate and a single test line. The test line is low in operational mode and high in test mode.

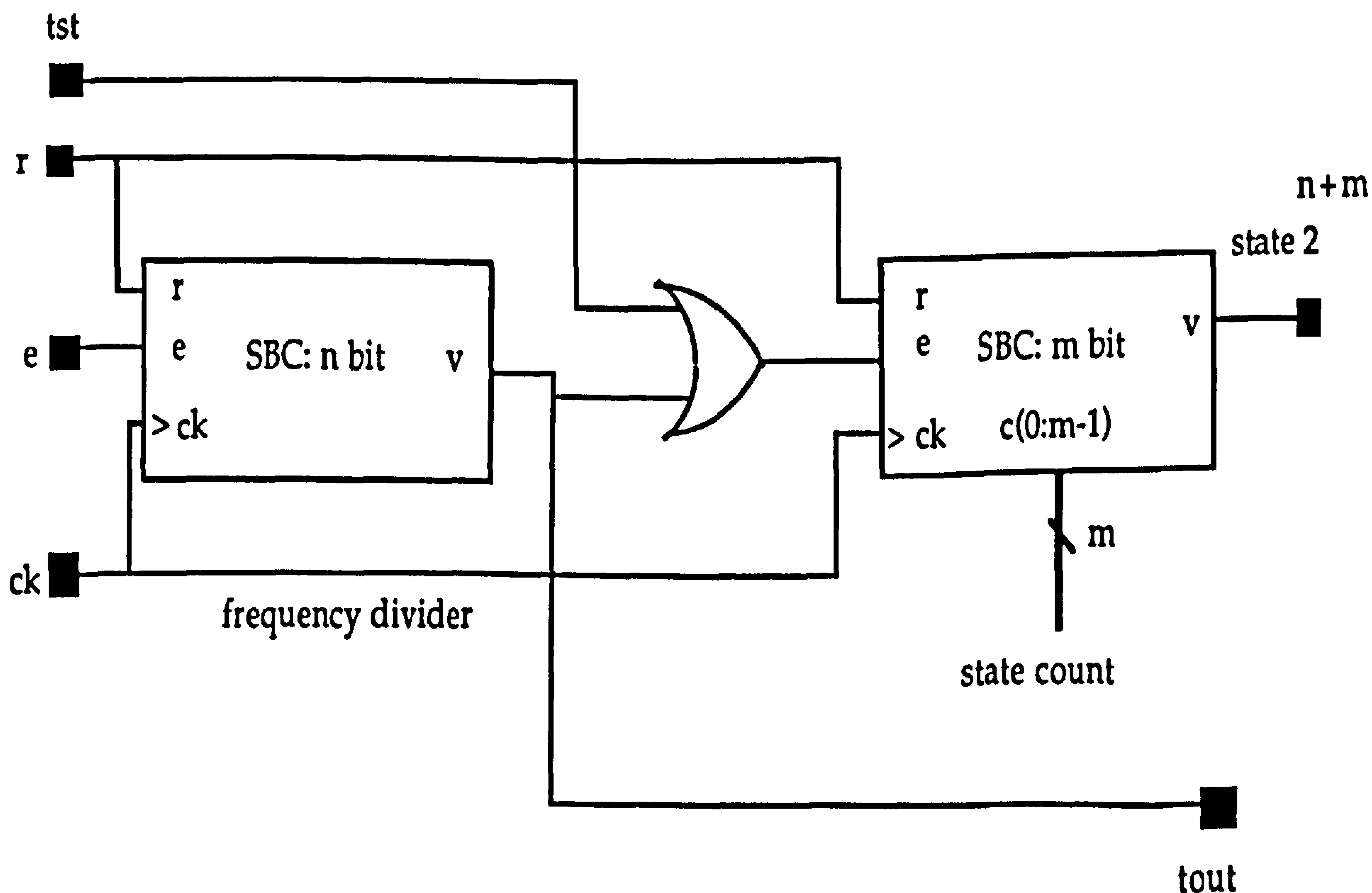



**Figure 2.6 Reducing the number of Test Signals**

Alternatively, test data can be brought in and out by replacing input and output data pads by bi-directional pads controlled by tri-states, as shown in figure 2.7.

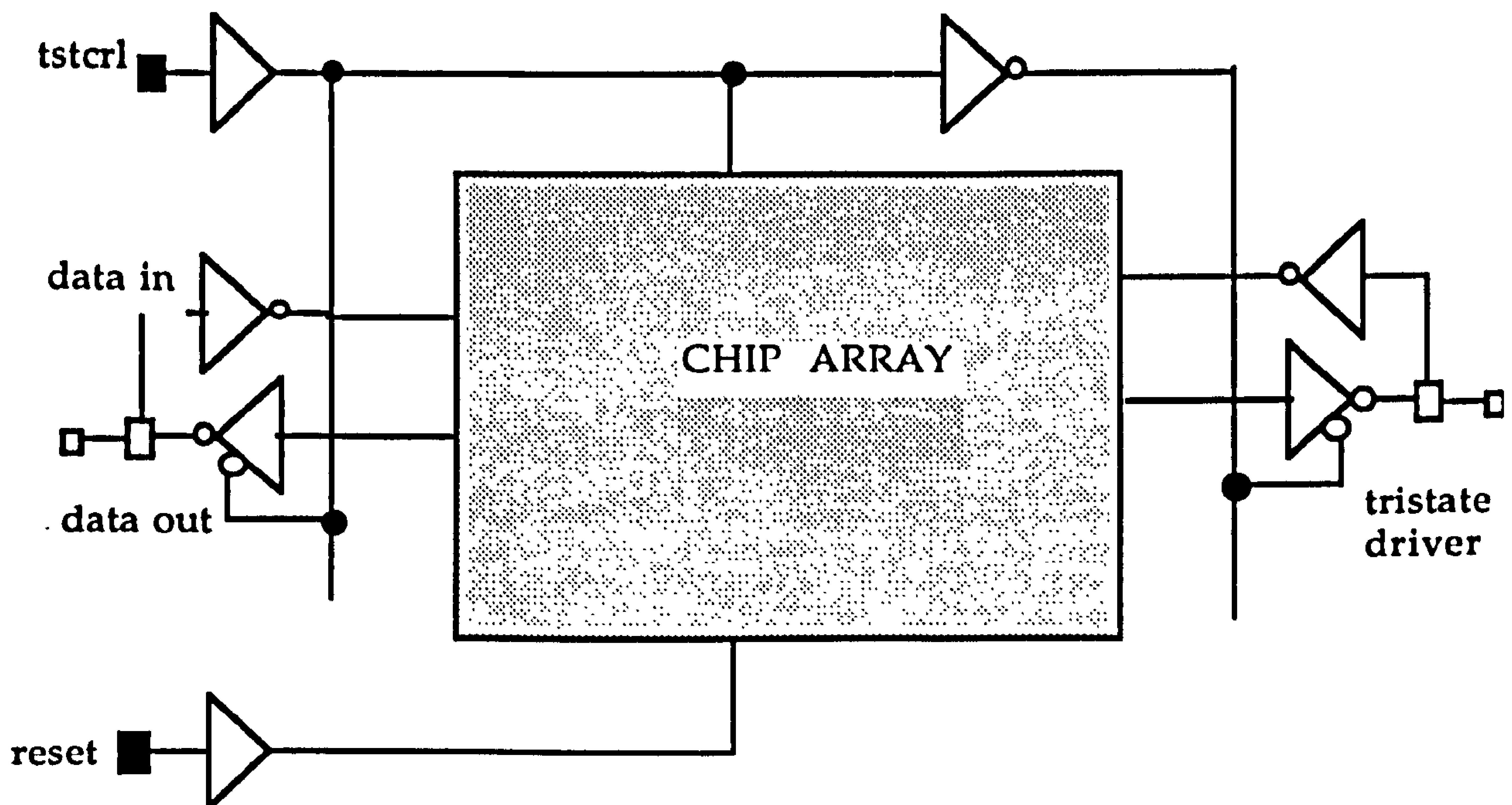



**Figure 2.7 Bi-directional Pads used for Test Input/output**

In operational mode, these pads operate in the conventional direction. In test mode they work in the opposite sense and test data goes in via output pads, out via input pads. The only additional pad is the test control pad, which also controls the direction of the tri-state drivers.

## 2.6 AD-HOC TECHNIQUES - FOR AND AGAINST

There are a number of advantages in adopting the ad-hoc approach to design for test. These include:

a. The functional circuit forms the basis of the additional test circuitry.

b. Both functional and structural tests are simplified.

c. Test vector generation time and cost can be significantly reduced.

d. Advantage can be taken of the pins in the chip package which would otherwise be unused.

There are disadvantages in these techniques, including the following:

a. There is an increase in chip area and pin count, although the latter can be minimised.

b. The operational speed of the circuit is reduced due to additional multiplexers in critical signal lines.

c. The test components are additional potential sources of fabrication faults.

d. A testability analyser can indicate the best position for test points, but the process of inserting them and generating test vectors cannot, in general, be automated.

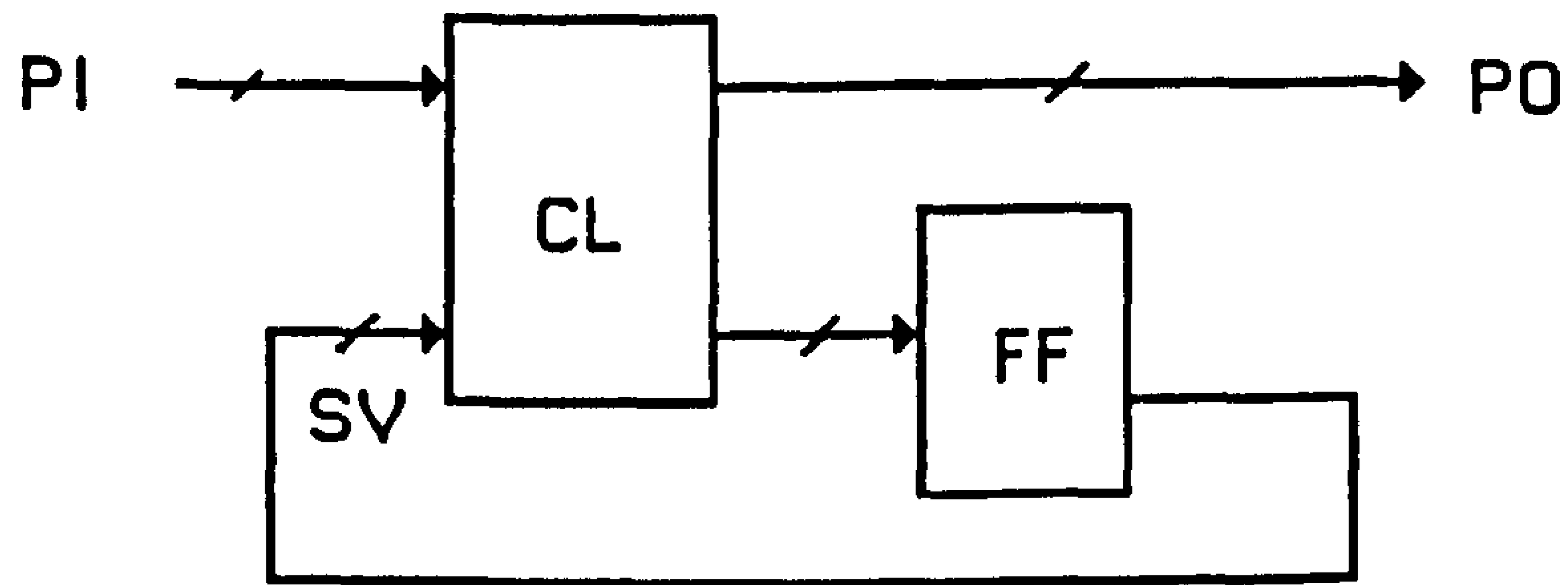
## 2.7 STRUCTURED DESIGN TECHNIQUES FOR TESTABILITY

Structured techniques allow the test generation problem to be reduced to one of generating tests for combinational logic.

One of the structured testability approaches is that of Scan-Set Logic. In this approach shift registers are used to load and unload data and are not part of the system data path. Not all system latches are necessarily controllable and observable via the shift register.

Another approach which is that of Built-In Logic Block Observation (BILBO). This technique has attributes of both the LSSD (Level Sensitive Scan Design) network and the Scan Path network. It has the ability to separate the network into combinational and sequential parts, and has the attribute of Signature Analysis - that is, employing linear feed back shift registers.

All the structured approaches allow controllability and observability of the state variables in the sequential machine. The test generation and fault simulation can therefore be directed more at a combinational network, rather than at a sequential network. In general a synchronous sequential circuit can be represented in block diagram as shown in figure 2.8. [WILL 83][WILL 86]




SV = Secondary Variables
CL = Combinational Logic

**Figure 2.8 A Synchronous Sequential Circuit**

PI are the primary inputs, PO are the primary outputs, SV are the state/secondary variables, CL is the combinational logic, and FF are the system flip-flops. The CL block will be made up of an input block generating the excitation logic to the flip-flops and an output logic block generating the primary outputs. As both these sub-blocks are driven by the same inputs, ie the PI's and the SV's, they can be lumped together as one CL block.

To test the CL block we need to have direct control over all of its inputs, of which only the PI's are so available. We also need direct observability of its O/P's, but again only the PO's are available. Testing the FF block is equally difficult as the SV's can neither be controlled nor observed directly.

The most widely used of ways that have emerged to overcome these difficulties are all based on a structured design approach known as scan-path.

Scan design reduces the complexity of the test generation problem for circuits with stored-state devices (flip-flops) and global feedback by using a 'divide and conquer' philosophy. This benefit is at the expense of increased circuit complexity to enable the necessary partitioning ('silicon overhead') which in turn can introduce unwanted further system delays.

### 2.7.1 PRINCIPLES OF INTERNAL SCAN PATH DESIGN

The principles of internal scan-path design can be summarised as follows:-

(a) the stored state devices can be tested in isolation

(b) the future-state of the secondary state variables can be set independently

(c) the outputs of the combinational logic block can be observed directly.

To do this it is important to establish a scan-path through the stored state devices by the addition of multiplexers as shown in figure 2.9.

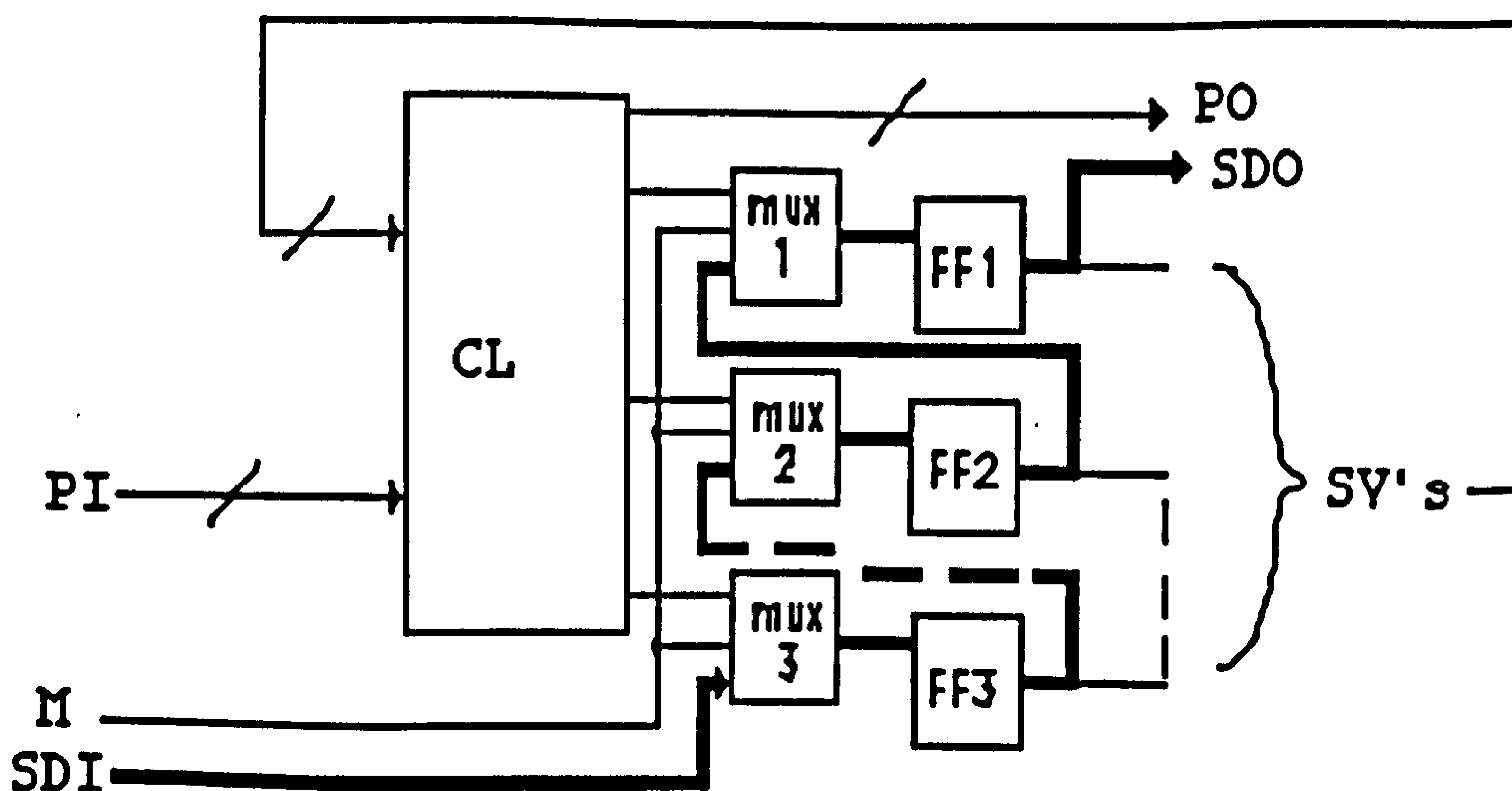



**Figure 2.9 A Scan-Path**

Three extra connections are shown: two inputs SDI and M (scan-data-in and mode respectively) and one output SDO, scan-data-out. With M set to divert the flip-flop inputs away from the CL block as scan-in/scan-out mode is entered, the flip-flops are connected as one long shift register with SDI as the serial input and SDO the serial output. This data path is known as the scan-path. Normal operating mode is re-entered with M set to connect the flip-flop inputs directly to the CL block.

The steps of a suitable test strategy could be as follows:-

(1) Select the scan-path and test scan the shift register by:-

(a) Flush testing a '1' through a background of '0's and a '0' through a background of '1's. This tests the ability of each flip-flop to assume either state.

(b) Shift testing using the sequence 00110011.... This exercises each flip-flop through all possible transitions.

(2) Determine a set of tests for the CL block assuming:-

(a) Total control of all inputs (PI's & SV's).

(b) Direct observability of all the CL block outputs (PO's and those to the flip-flop inputs).

(3) Apply each test as follows:-

(a) Select scan-path mode. Pre-load the flip-flops with test input values and establish test input values on the PI's.

(b) Select normal mode. The steady-state output response of the CL block can now be clocked into the flip-flops.

(c) Return to scan-path mode and clock out the contents of the flip-flops. Compare these values plus the PO values with the expected fault-free response.

It can be seen that the 'divide and conquer' approach has reduced the testing of the flip-flops to a standard procedure and the only test-generation problem that remains is that of generating a compact test set for the combinational block.

Scan path circuit elements are based on a storage element such as a D-type flip-flop, with a multiplexer to choose between the scan path input and operational data input. These are built up into synchronously reset enable (E-type) and toggle (T-type) flip flops.

## 2.8 SCAN PATH TESTING - FOR AND AGAINST

Structural design for test, of which scan path testing is an implementation, has a number of benefits. These include the following:

a. The test circuitry is an integral part of the circuit design. Provided that the circuit has a hierarchical structure of storage elements and combinational blocks, the scan path is easily incorporated.

b. The scan path breaks the circuit into separate combinational blocks, which are generally small and easily tested modules. Feedback loops are disabled in the process. All storage elements are directly controllable and observable, and the controllability/observability path to or from each combinational element links to the nearest storage element.

c. Scan path testing can be combined with path sensitization and justification techniques to develop test vectors, or with random test sequence generation techniques which can be incorporated on- chip.

d. A scan path, by breaking the circuit into a number of separate combinational blocks enables automatic test pattern generation software to be applied to the circuit.

Weighted against these significant benefits are the disadvantages of additional circuitry:

a. The additional gating in the scan path elements and the scan path linkage itself can give rise to a significant increase in chip area, with consequent cost increases.

b. Three additional pads are required: scan control, scan in and scan out. However, it is, possible to multiplex the latter two with data input and output pads, as these are not used during scan path testing.

c. The multiplexers in the scan path elements cause additional delay.

d. Scan path testing is not possible for level-sensitive elements such as RAM and ROM as it is an edge triggered technique.

## 2.9 BUILT-IN SELF-TEST AND STRUCTURED DESIGN FOR TEST

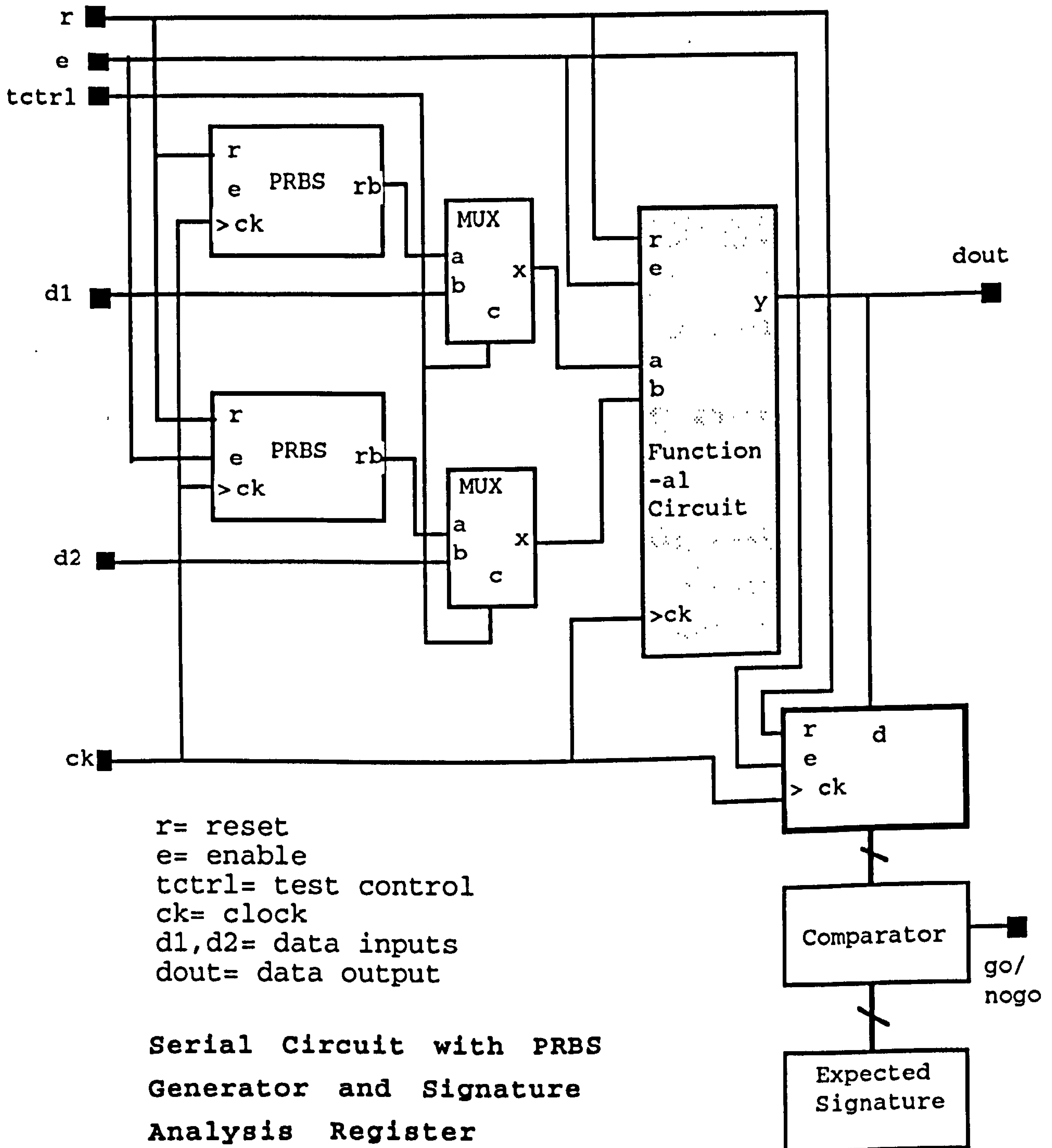
The method of path sensitization and justification, when used on its own or in conjunction with a technique such as scan path testing, is a deterministic method of generating test vectors. The return for an extensive outlay of time to develop test vectors is a set of structural test vectors which may achieve the ideal of full fault coverage, but in practice, falls somewhat short of this ideal.

The alternative non-deterministic approach is to generate a large set of random test data automatically. If the set is large enough (and sufficiently random) it should give adequate fault coverage. The random test vectors may be generated by circuits included on-chip, hence the name given to this technique built-in self-test (BIST). Not only does BIST remove the need for test vector generation altogether (all the circuit requires is a clock sequence of the appropriate length while in test mode), but also it means a circuit may be tested at any time while it is in operation.

Figure 2.10 demonstrates an example circuit with PRBS generator and signature analysis register. The random test data are generated by one or more pseudo random binary sequence (PRBS) generator modules (also known as linear feedback shift registers (LFSR), and multiplexed into the input data stream when the circuit is in test mode. A PRBS generator is based on an n-bit shift register which produces a random stream of $2^{n-1}$ bits. Although the bit stream is statistically random, the same stream is produced each time the PRBS is reset.

The output is shifted into a signature analysis register, which produces a distinctive pattern depending on the entire data stream it receives. An n-bit signature analysis register will produce a different signature for a single-bit variation in an input of $2^{n-1}$.

The signature may be compared with the fault-free signature by a comparator circuit, and a single go/no go output produced.




r= reset
e= enable
tctrl= test control
ck= clock
d1,d2= data inputs
dout= data output

**Serial Circuit with PRBS Generator and Signature Analysis Register**

**Figure 2.10**

## 2.10 BUILT-IN SELF-TEST - FOR AND AGAINST

Built-in self-test has a number of advantages, in particular:

a. There is no need to generate any test vectors - they are produced automatically on-chip. This saves a great deal of time and cost in circuit development.

b. A number of theoretical studies have shown that non-deterministic test vectors give approximately the same fault coverage as deterministic vectors for typical circuits. [TRIS 84]

c. Field tests can be carried out on the chip.

The disadvantages of the technique are as follows:

a. Without a full fault simulation run (which is not generally possible) there is no certainty that the random test data stream is covering all the possible 'stuck-at' faults in the circuit. If the circuit contains a large number of components with high fan-in such as 4-input AND gates (for which the probability of getting a 1 output is only 1/16) then the chances of putting all the nets into all states are reduced.

b. The PRBS generator(s) and signature analysis registers create an overhead in silicon area, and can themselves be source of faults. In addition they require a control test pad, as well as a go/no go pad, although the latter may be multiplexed with a data output pad. However, existing D-types can be used to form PRBS generators and signature analysis registers.

## 2.11 COMBINATION OF SCAN PATH AND BUILT-IN SELF-TEST TECHNIQUES

A number of the DFT techniques outlined previously may be used in on a single design. In particular:

a. Testability analysis is used to identify obscure nets for attention by ad-hoc DFT techniques.

b. Path sensitization and justification can be used on the combinational blocks of circuits which includes a scan path, in order to generate test vectors to be applied via the scan path.

c. A PRBS generator may be used to provide the input into the scan path, and a signature analysis register to analyse the scan path output.

d. The PRBS generator, signature analysis generator, scan path and operational register can be amalgamated into a single Built-in Logic Block Observer (BILBO), which is used for all registers in the circuit.

This technique is known as hybrid design for test which is of particular significance.

## 2.12 HYBRID DESIGN FOR TEST - FOR AND AGAINST

Hybrid design for test techniques are the most powerful available for many circuits. Their benefits include the following:

a. There is no need to develop the structural test vectors - these are generated on-chip when chip is in test mode.

b. The chip may be tested at any time while it is in operation.

c. The use of the scan path means that the combinational depth of the circuit is reduced, and chances of obscure nets not being tested are reduced.

d. The test circuitry is an integral part of the circuit, and requires only the minimum of additional design effort.

The disadvantages of the technique is the silicon overhead as follows:

a. Hybrid design for test creates a significant silicon overhead in the PRBS generator, signature analysis register and scan path register, or the large BILBO registers. However, this overhead can be reduced by checking whether a BILBO implementation is smaller than one with a separate PRBS generator and signature analysis register. If BILBO implementation is chosen, registers in the middle of the scan path can be scan path versions, rather than full BILBOs.

b. The additional circuitry is in itself an additional source of fabrication faults and circuit delay.

## 2.13 TEST ACCESS AND BOUNDARY SCAN

The 'Bed of nails' technique is becoming increasingly impractical, both because of lengthening back-drive times. In the face of miniaturisation, this technique is unsuited to todays boards with surface mount and Pin Grid Array devices, hybrid substrates, mother-daughter board assemblies and boards with components on both sides. Conformal coatings are damaged by probing and need repair afterwards.

The best solution to the test access problem is to move the ATE's (Automatic Test Equipment) test pattern registers directly into the boundary pins of the components to be tested. This overcomes the need for both probes and for logic-family-programmable buffers.

The increased program execution time due to serialisation is offset by configuring test registers as pseudo random pattern generators and signature compressors. (Attempts to do this in ATE systems were frustrated by the difficulty of mapping ATE bits into relevant register groups). Resulting test programs can be short enough to fit in a small on-board memory.

A standard for the serial bus link to these on-board test registers has emerged. The Boundary Scan proposal by JTAG (Joint Test Action Group) was ratified in Autumn 1989.

Boundary scan is a method of gaining test access to components within a circuit board, without using bed of nails or expensive Automatic Test Equipment. It is especially necessary for densely packed assemblies such as surface mount, double sided boards, hierarchical assemblies and silicon hybrids.

Test registers, embedded in the boundaries of components and functional blocks, can interrupt the normal flow of signals under the control of a global 'Test Mode Select' signal. Test data are shifted serially in and out of these registers, which are connected in one or more 'Boundary Scan Chains' as shown in figure 2.11. Tests are performed both on the interconnections and on the components themselves. Data transfer is minimised where possible by configuring registers as pseudo random generators and signature gatherers. With improved goods-in testing of new components, design errors are minimised using improved design methods and simulation.

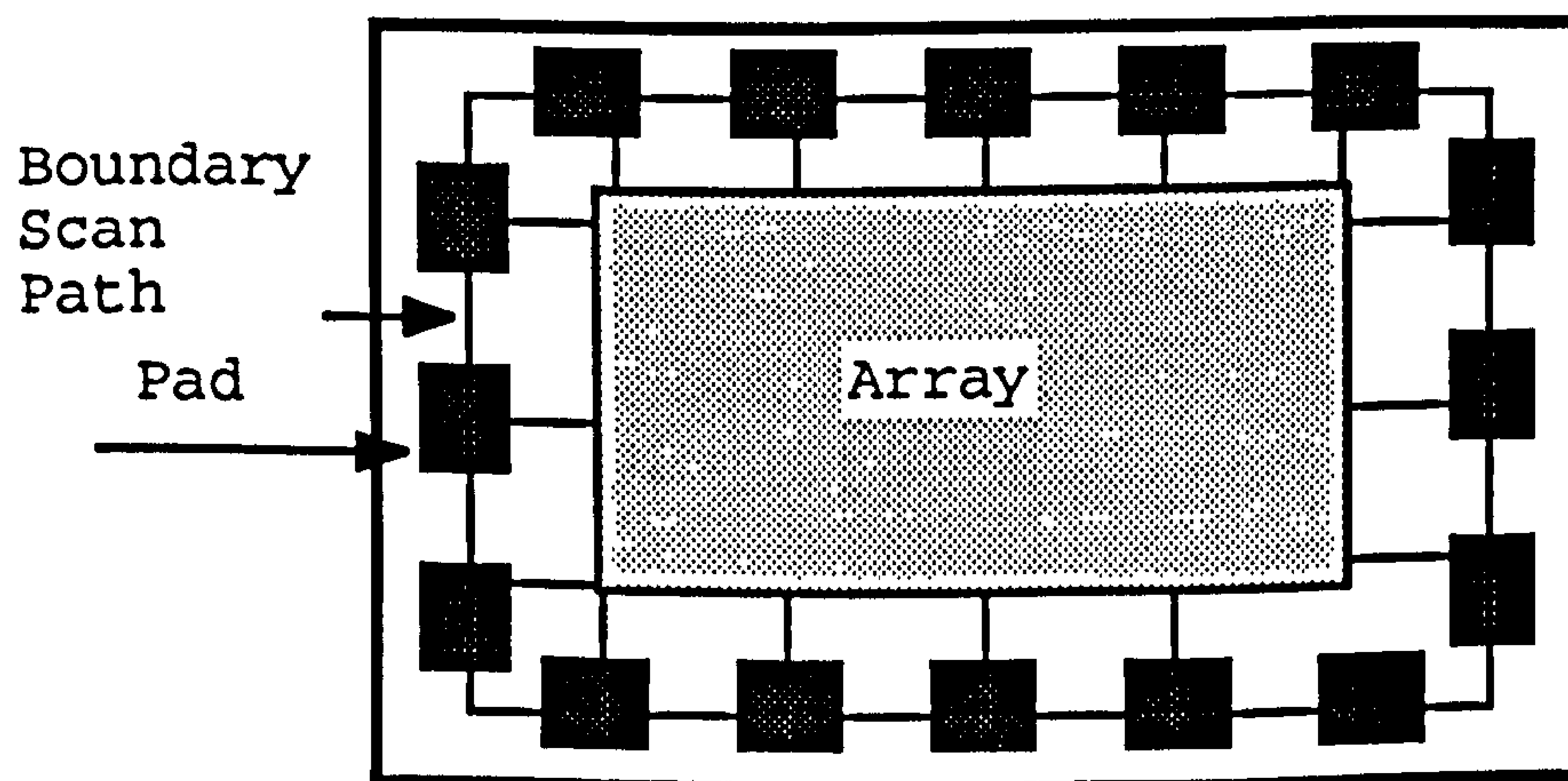



**Figure 2.11 Boundary Scan Path**

The IEEE Std 1149.1 - 1990, IEEE Standard Test Access Port JTAG [JTAG 90], defines circuitry that may be built into an Integrated Circuit to assist in the test, maintenance and support of assembled printed circuit boards.

The circuitry includes a standard interface through which instructions and test data are communicated. A set of test features is defined, including a boundary scan register, such that the component is able to respond to a minimum set of instructions designed to assist with the testing of assembled printed circuit boards.

This standard defines test logic that can be included in an integrated circuit to provide standardised approaches to:

- testing the interconnections between integrated circuits once they have been assembled into a printed circuit board or other substrates;
- testing the integrated circuit itself; and
- observing and modifying circuit activity during the component's normal operation.

JTAG involves inserting special Test Register cells into the chip design. The Test Registers are strategically placed to intercept the signals that pass between blocks of logic. The Test Registers are able to perform a number of different test functions which are used to test the blocks of logic.

When the circuit is not being tested, the test registers act as normal clocked or direct signal buffers and do not interfere with the normal operation of the chip. See fig 2.12a.

However during testing, the test registers are re-configured to form shift registers. Test Registers are connected together serially to form a 'Scan Loop', where serial data can be shifted in from a test input pin (TDI) and serial data can be shifted out via a test output pin (TDO) see fig 2.12b.

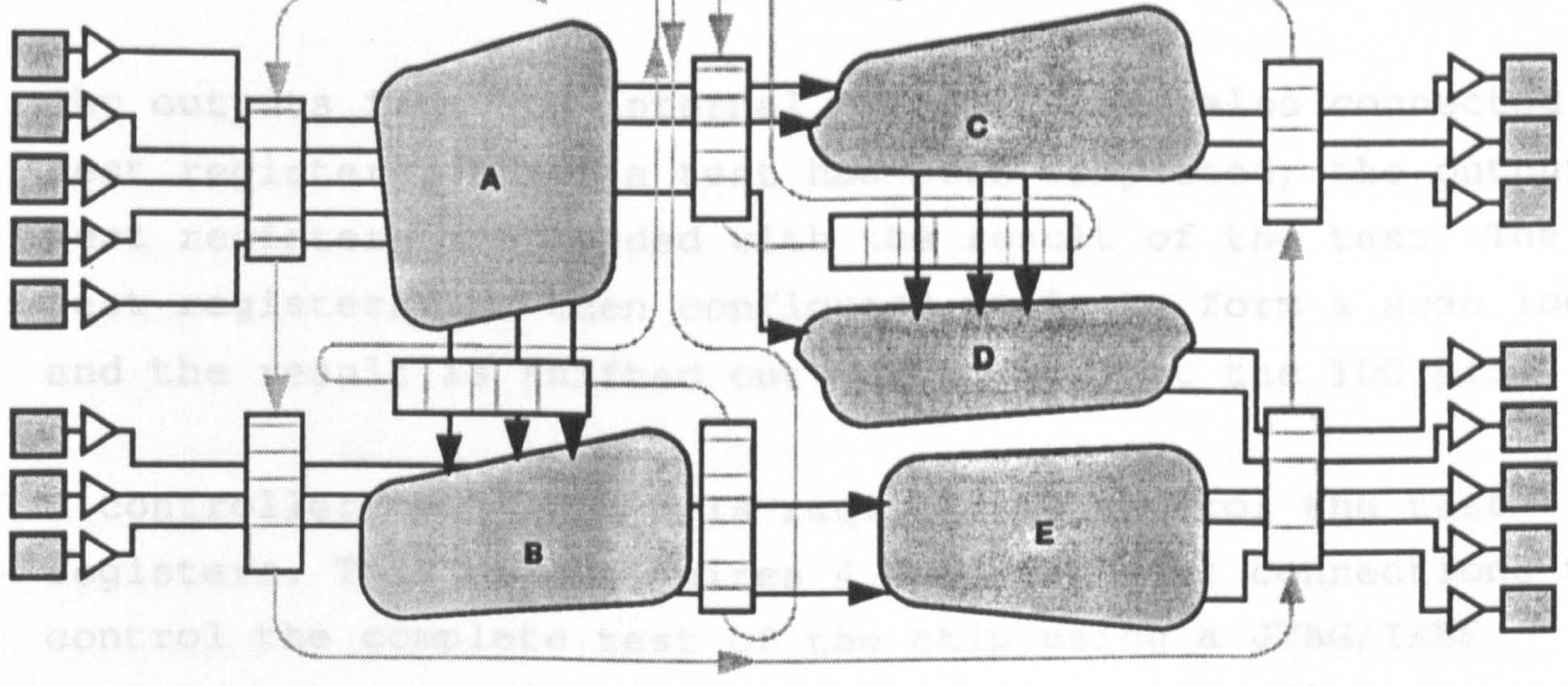



**Figure 2.12a Chip in NON-Test Mode**

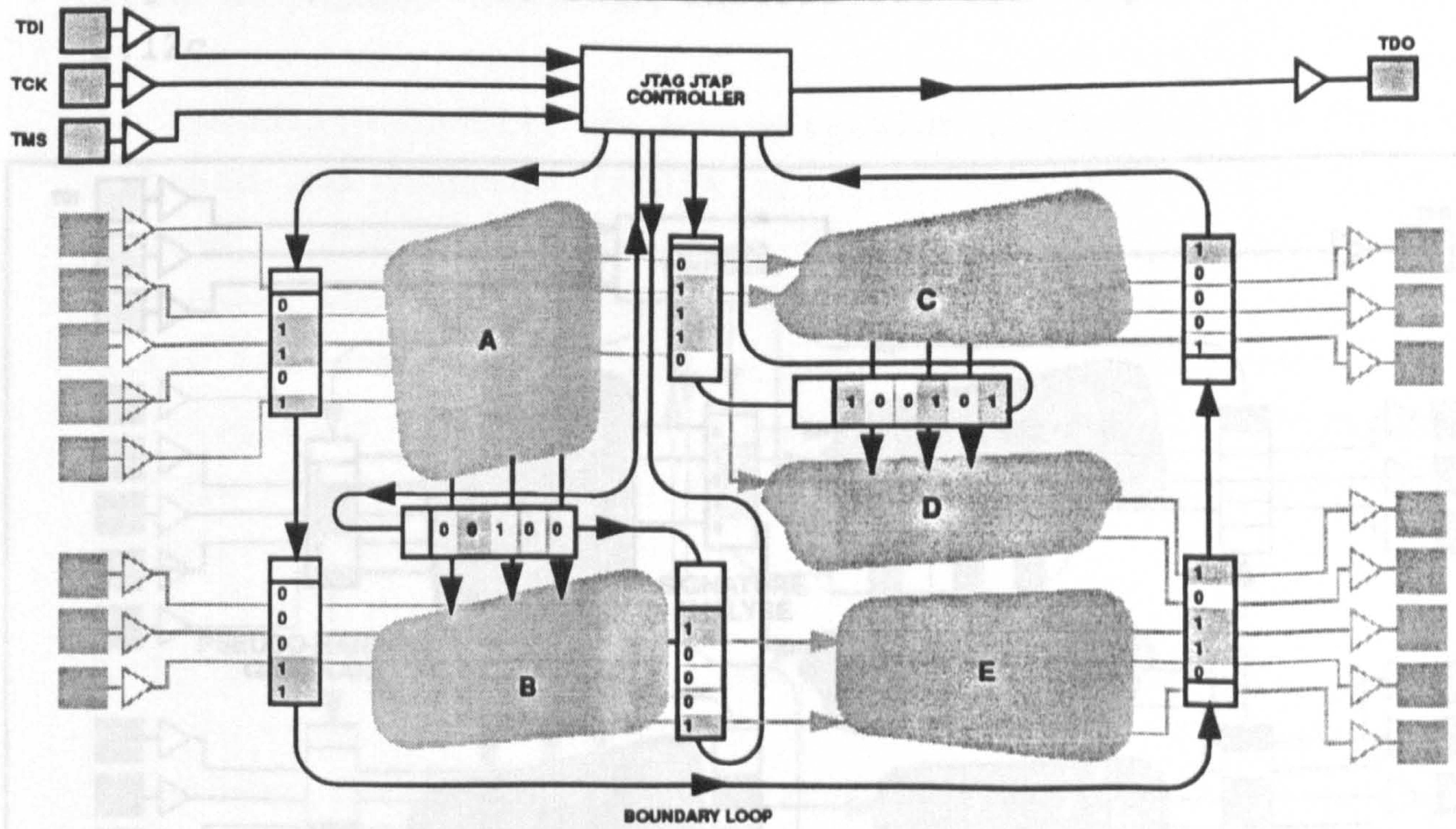



**Figure 2.12b Chip in Shift Mode**

Thus it is possible to load into the circuit a predetermined set of values which are applied to the inputs of the internal circuitry. Once loaded, the test registers can apply the data directly to the internal circuitry.

The outputs from the internal circuitry are also connected to test registers. After a test has been completed, the output test registers are loaded with the result of the test. The test registers are then configured again to form a scan loop and the result is shifted out and appears at the TDO pin.

A controller cell (JTAP) is required to control the test registers. This cell requires 4 external test connections to control the complete test of the chip using a JTAG/IEEE 1149.1 protocol which is explained in more detail later.

The test registers can be configured into pseudo-random pattern generators, to exhaustively test a block of internal circuitry. Test registers on the outputs build up a signature, which is then shifted out for comparison, see fig 2.12c.

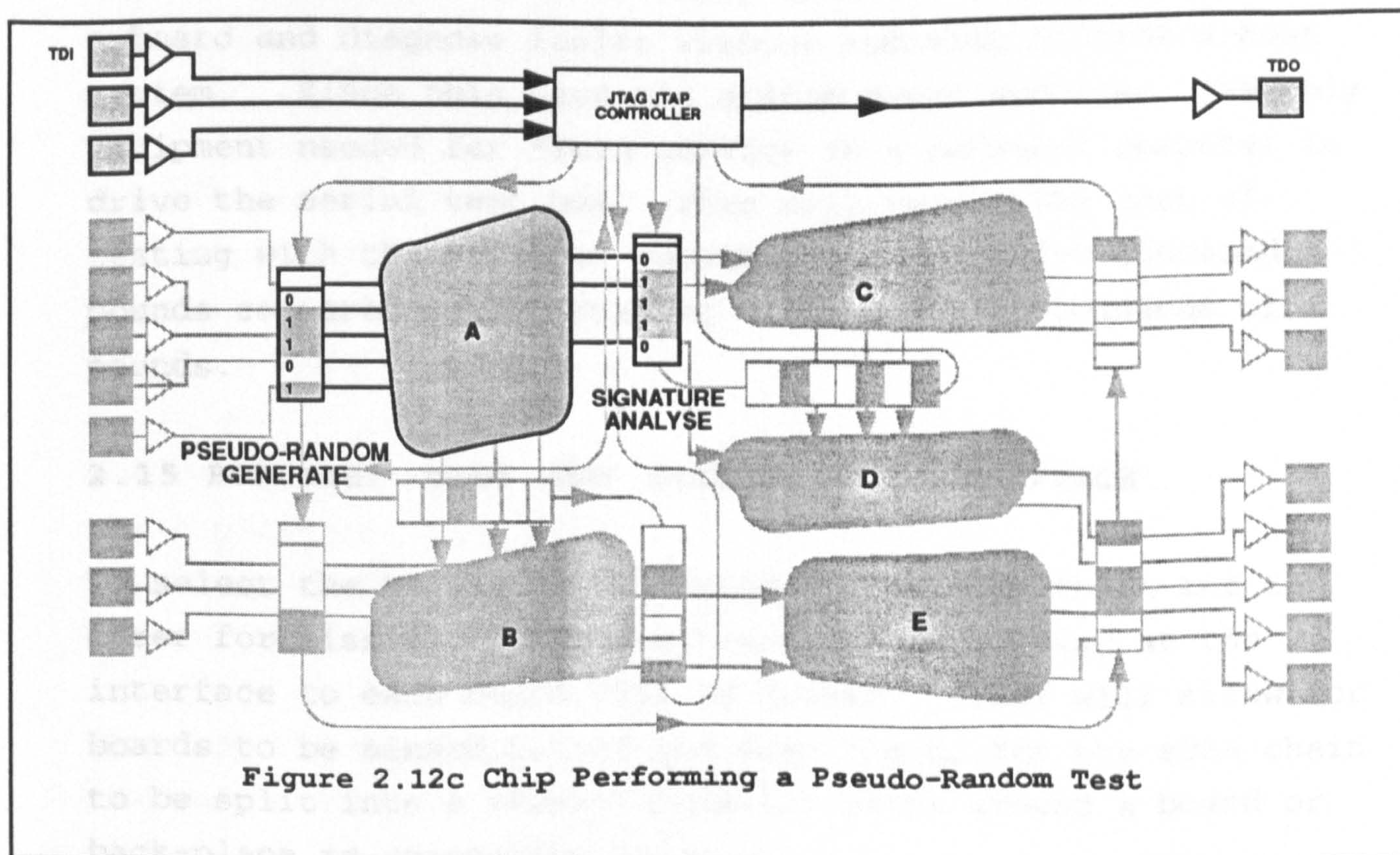



**Figure 2.12c Chip Performing a Pseudo-Random Test**

Although a single scan loop is all that is required, to reduce data shifting times, the test registers can be connected to form several parallel scan loops.

The test program primarily drives the JTAP (JTAG Test Access Port) controller on the chip. The complete BIST test program is made up from a series of individual shift and pseudo-random self-test operations which are controlled from the JTAG test interface. These test operations test out the individual blocks of the chip. The test program will contain all the serial data to load into the scan loops that is to be applied to the internal blocks of logic, plus the seeds and expected signatures for the pseudo-random tests. The JTAG hardware on a chip can be extended to include autonomous self-test. This is where the complete self-test of the chip is run automatically without the need to shift in data from the external JTAG interface.

## 2.14 BOUNDARY SCAN AND IN-SYSTEM TESTING

An immediate benefit of Boundary Scan is the ability to test a board and diagnose faults without removing it from a host system. Since this uses the system power supplies, the only equipment needed for field service is a personal computer to drive the serial test bus. This will reduce the cost of testing with the personal computer costing a few thousand pounds compared to ATE costing a few tens of thousands of pounds.

## 2.15 BOUNDARY SCAN AND SYSTEM TEST INTERFACE

To select the boards to be tested by the scan chain and to cater for missing boards, a hierarchical gateway at the interface to each board will be needed. This will allow for boards to be missed out of the scan chain, for the scan chain to be split into a several parallel paths around a board or back-plane re-converging later on.

Each board can have its own on-board scan chain test program and interpreter. This can be run as an execution of the JTAG 'run self test' instruction, clocked by the back-plane test bus clock.

The gateway and on-board program interpreter should be embodied in a test interface chip which could be obtained from IC manufacturers such as Texas Instruments.

It is expected that, in many applications, the gateway and other features will be embodied in larger VLSI devices performing other primary functions on the board.

## 2.16 BOUNDARY SCAN AND SOFTWARE ELEMENTS

Development of software tools to support the inclusion of Boundary Scan Testing from back-plane level down to the ASIC level are being carried out by many companies.

For board level testing, the tool that currently available for use is the Interactive Diagnostics Workstation running on a personal computer (PC). The boundary scan is normally driven from the PC via a software interface to a parallel port.

When the user chooses the route of the hardware description files, the Scan Chain Map Compiler reads the files and builds a map of the Boundary Scan Chain. This will be complex since it has a hierarchical shape.

The resultant presentation will normally display the scan chain graphically, and the user is able to choose the board, component to be displayed. Elements that have failed a test are subsequently highlighted.

The user can apply new test patterns, either graphically, textually, or from a file (test program).

Test patterns are intended to be derivable from design simulations. At the highest design level, patterns are written in terms of the primary inputs and outputs of functional blocks or components. The scan Chain Instruction Compiler cross references these to the scan chain positions and calculates the necessary routing instructions for the hierarchical scan path.

## 2.17 ADVANTAGES OF BOUNDARY SCAN PATH

The inclusion of serial scan paths in digital circuits has long offered an attractive route to increasing the controllability and observability of an ASIC implementation or a circuit panel for test purposes. For example:-

a. The inclusion of boundary scan paths 'around' components offers an alternative implementation to in-circuit probing to detect and diagnose open and short circuits. It avoids the need to physically probe the fragile surface mount devices which can affect the failure condition.

b. The monitoring of system performance. Whilst it is unlikely that embedded scan paths can be used to define 'at speed' testing of a system, the opportunity exists to use them to 'capture' data during real time operation of the system.

## 2.18 CONCLUSIONS

Typical fault classes today are *production faults* including shorts, opens, components inserted wrongly and wrong components inserted. These production faults comprise 99.5% of all faults. The other 0.5% of faults are from faulty components. The *maintenance faults* occur from open circuits and faulty components

This chapter has discussed both the Ad Hoc and the Structured approaches of testability. In summary, the *Ad Hoc Testability* techniques are applied on top of existing designs. They are specific to the logic under test. The *Structured Testability* techniques are designed-in up front, which complies with the Concurrent Engineering concept. The techniques form a consistent approach and are typically scan-based.

The availability of a 'standard' such as the IEEE 1149.1 to define an implementation of scan path method is considered to have immense benefits. The definition of a consistent implementation method facilitates:

a. A standard electrical interface to the UUT. Consequently, it allows the use of standard 'external' physical test equipment. The test software must still be customized for the application, but suitable design tools should be readily available.

b. The design of the scan paths to be embodied in individual sub-circuits, of either an ASIC or a circuit board, is such that they can be efficiently combined to implement the overall system test.

c. The availability of merchant parts and ASIC cells to implement scan paths efficiently. This significantly reduces the 'overhead', in terms of unit price and unit volume, which is associated with designing and developing scan path circuits.

d. The opportunity for the test vectors employed in the simulation of an ASIC to be 'directly' employed to validate the components themselves when they are mounted on a circuit panel, thereby achieving a consistency of test conditions.

e. The opportunity to combine the test vectors designed for individual components, to define part of a board test sequence,ie., implementing the system test strategy in a hierarchical fashion.

f. The development of Computer Aided Engineering (CAE) 'tools' to automate the placement of scan paths, the generation of test stimulus data and response data, the translation of component level 'files' to compile system level tests.

## REFERENCES

[PARK 86] K P Parker, Hewlett-Packard, "Testability: Barriers to acceptance", IEEE Design & Test, October 1986 pp 11-15.

[JTAG 90] IEEE Standard 1149.1-1990 "Test Access Port and Boundary Scan Architecture.

[PARK 87] K P Parker, "Integrating Design and Test: Using CAE tools for ATE Programming", Published by Computer Society Press of the IEEE, 1987.

[SCHW 87] A F Schwartz, "Computer Aided Design of Microelectronic Circuits and Systems, Fundamentals, Methods and Tools" vol 2, Digital Circuit Aspects and State of the Art, Academic Press 1987.

[TRIS 84] E Trischler "An Integrated Design for Testability & Automatic Test Pattern Generation System: an Overview" the 21st Design Automation Conference 1984 IEEE pp 209-215.

[WILL 86] T W Williams, "Design for Testability", Advances in CAD for VLSI Testing, edited by TW Williams, North Holland Publishing 1986, pp 95-160.

[WILL 83] T W Williams, et K P Parker, "Design for Testability ... A Survey?" IEEE Proc. of IEEE Vol 71, no 1 January 1983, pp 998-112.

# CHAPTER 3
# REVIEW OF BOUNDARY SCAN ARCHITECTURE

## 3.0 INTRODUCTION

The Joint Test Action Group (JTAG) was founded in 1985 as a result of an initiative within the Philips Group of companies to develop and promote structured design-for-test techniques. It has been able to create a broad consensus for Boundary Scan Test Architecture. The JTAG standard was the first formal effort by users to identify their testability requirements to Semiconductor manufactures.

Originally, the Boundary Scan Architecture (BSA) test defined by the JTAG was a simple scan interface to support board level test. As the JTAG grew to include more companies, other needs had to be considered and as a result, the original BSA architecture was expanded to satisfy these needs. The JTAG proposal version 2.0 was published in March 1988 and formed the basis of the actual standardisation architecture of P1149.1 IEEE Committee.

This chapter reviews the recent literature on Boundary Scan Design. It examines the history of VHDL development and summarises VHDL related technologies, including Computer Aided Design, Engineering and Test (CAD,CAE,CAT). It then identifies both the motivating factors behind the implementation of this research project and the development frame work.

## 3.1 TRENDS IN THE ELECTRONICS INDUSTRY

The driving forces behind the JTAG initiative were two fold; the increasing use of Surface Mount packaged devices, and Application Specific Integrated Circuits (ASICs) on printed circuit boards (PCBs). These trends are continuing in the 90's. In January 1989, Pound reported that 48% of electronic system companies had adopted surface mount technology, and that this percentage was rising [POUND 89]. This view is echoed by Cole, who also gave the prediction that the ASIC market would rise from its 1988 figure of 15% to 21% by 1993 [COLE 89].

Bursky also predicts the increasing use of ASICs, (and what he terms "user-specific ICs"), with sizes reaching to 30M transistors at geometries of 0.25 micron by the year 2000 [BURS 89]. He comments that the design time for such devices will reduce from 2 years to 9 months. The dominant packaging styles will be flip-chip and tape-automated bonding. Bursky's final comment was that testing problems will eventually be solved only by the adoption of both internal (device core logic) and external (device boundary) scan architectures.

Bassett et al. discussed the trends in ASIC usage within IBM. Up to 500 signal pins and 100K gates is now common, with size forecasts to increase to 300K gates [BASS 89]. IBM has the ability to include internal scan paths (based on Level-Sensitive Scan Design -LSSD- architecture) in their devices, and the authors state that the design philosophy has now been extended to include input/output boundary scan to reduce the number of driver/sensor pins required on the tester. The motivation for this is to reduce the overall testing. IBM's proprietary version of boundary scan is described in [DUPT 84] and more recently in [BASS 90].

## 3.2 THE STANDARDISATION EFFORT

JTAG was merged into the IEEE P1149 Testability Standards Committee (TBSC) in 1988, following exposure of JTAG's status at the 1987 International Test Conference [BEEN 87]. The TBSC operated under the section of the IEEE Computer Society's Test Technology Technical Committee [IEEE 90]. At that time, P1149 test group was developing a standardized test bus to provide access to test support functions built into the circuit board. An agreement between the P1149 group and the Joint Test Action Group (JTAG) permitted the incorporation of JTAG's effort into the 1149.1 minimum serial subset of the standard. The agreement proved JTAG's development of the proposal document and prototyped the proposed test access port (TAP) and boundary scan architecture. It also resulted in the first of a set of new standards - IEEE Standard 1149.1 -1990.

Other specifications are still in preparation: P1149.2, the Extended Serial Test Bus, P1149.3, the Real Time Test Bus; and P1149.5 the Back Plane Test and Maintenance Bus (based on the earlier VHSIC TM Bus [TBUS 87],[AVRA 87]. Each of these standards will be free standing when they finally emerge. The overall P1149 document will provide a guide to the overall structure of the set of 1149.n series of documents.

## 3.3 IMPLEMENTATION OF BOUNDARY SCAN ARCHITECTURE INTO INTEGRATED CIRCUITS

The response of semiconductor manufacturers to the promotion effort has been very encouraging, with companies such as Plessey, National Semi, VLSI Technology Inc, Motorola, AT &T, Texas Instruments, LSI Logic, NCR, NEC, LSTI, SGS Thomson and Philips Components introducing BSA features into their ASIC libraries.

In 1990, Philips Components responded to the new standard by providing their ASIC engineers with a tool kit containing a library of soft macros from which an optimised BSA circuit can be assembled. In addition, Philips have recently developed a Testability IMprover (TIM) Software system, which integrates their soft macros into their ASIC technology. This effort was initially to exploit the use of BSA for their in-house ASIC design activity. The use of this tool has now been extended to work from a Mentor Graphics CAE System [PHIL 90], as most of their designs are implemented using the Mentor system.

An interesting factor about TIM is the concept of the "Single Transport Chain" [MAUN 90]. Consider the various parallel registers in the 1149.1 architecture: Instruction, Bypass, Identification, Boundary Scan, and User-Specific. In general, a register element is capable of three fundamental actions: capture, shift and update. "Capture" and "Up-Date" mean to transfer data to or from a parallel input to outputs of the register. "Shift" means to transfer register data serially through elements of the register. Not all register elements contain all these features, for example, the identification register does not contain an update function. But all register elements contain a core shift function. Furthermore, only one register is connected from TDI to TDO at any one time. As a result, the shift elements can be considered to be a shared resource, for example a single transport chain, as shown in figure 3.1, thereby reducing the final number of memory elements needed to implement the various registers.

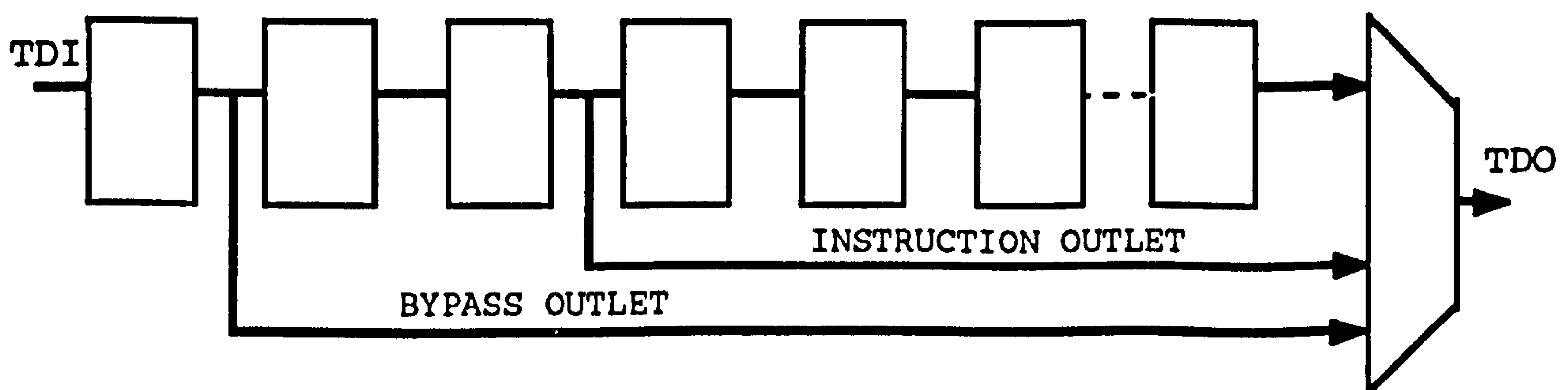



Figure 3.1 Single Transport Chain with Various Outlets
SOURCE: JOURNAL OF ELECTRONIC TESTING: THEORY AND APPLICATIONS,2, 11-25 (1991), KLUWER ACADEMIC PUBLISHERS, BENNETTS R AND OSSEYRAN A.

Texas Instruments have led the way in introducing BSA features into both commercial and ASIC devices [MCLE 89]. The first devices to emerge were modified forms of their register and latch "octals". Code named 74BCT8244/8245/8373/8374, these devices are replacements for the standard 244/245/... devices, although not directly pin-for-pin compatible because of the extra pin-outs required for the boundary scan features. The BSA octals contain boundary scan cells on the inputs and the outputs, a full TAP controller, and extra facilities to configure output cells into a pseudo-random number generator (to generate test stimulus to other devices). They also include input cells into a parallel signature analyser to capture and compress the responses from other devices. On bus-orientated designs, these octals provide a means of isolating and controlling major system components (e.g. the memory and processor) and areas of glue logic.[PERR 89]

Texas Instruments have also announced a new range of devices to be used at board level for controlling the test of other on-board devices compatible with 1149.1. This range includes: 74ACT8990 Test Bus Controller: 74ACT8997/8999 Scan Path Selectors; 74ACT8994 16-bit Digital Bus Monitor. These devices are part of TI's Systems Controllability Observability Partitioning Environment (SCOPE), which is supported by the Advanced Support System for Emulation and Test (ASSET).

The article by McLean [MCLE 89] contains details of the implementation of boundary scan cells in TI's 1.0 micron CMOS TSC500 standard cell library. This library contains fourteen different cells, including a 1149.1 TAP controller and various forms of boundary scan cells. The impact on propagation delay is quoted as being 0.5ns per scan cell. In terms of additional gates, each scan cell requires around 15 gates to implement, and each bit of the instruction register requires 11 gates. The TAP Controller requires less than 200 gates. TI has also included 1149.1 features in their 0.8 micron 100K BiCMOS gate array library.

Finally, McLean describes TI's 32 bit TMS320C30 Digital Signal Processor. This device makes extensive use of both internal and boundary scan paths and, although not strictly compatible with 1149.1, the facilities go along way to providing the features of the standard. The reason for the slight incompatibility is that the design of this chip started before the standard was finalised. As a result, TI had to make implementation decisions before the final version of the standard. The newer TMS320C5X is fully compatible with the standard.

Perry states in his article in IEEE Spectrum on the Intel i860 microprocessor, the successor to Intels 80X86 microprocessor family, that the new microprocessor will incorporate BSA features [PERR 89]. It also appeared that Motorola had incorporated a TAP interface MC68040 microprocessor [TIME 89]. Other companies started to introduce 1149.1 features into their standard products. These companies included Lattice Semiconductors (PLD family) and AT & T (digital signal processors).

An interesting BSA IC-implementation was presented by van Rissen at the 1989 European Test Conference [VAN 89], describing a hierarchical test architecture based on the JTAG proposal. The architecture was designed to ensure testability at board and chip levels and to aid self-test of macros in the chip.

Board level controllers for the application of boundary scan tests are emerging. Vining of Texas Instruments [VINI 89], describes the design of trade versions of an interface chip, called the Scan Bus Master which interfaces between a 16-bit processor bus and a TAP. Ballew and Streb of AT & T [BALL 89] describe a device called PROBE ASIC.

An earlier paper by Lien and Bruer [LIEN 88] describes a Module Test and Maintenance Controller for use in a hierarchical test and maintenance environment. The controller for a 1149.1 board is detailed. The design is similar to both TI and AT & T designs, but is based on an earlier provisional version of the 1149.1 standard.

Halliday [HALI 89] also describes the use of BSA devices (for example TI octals) for increasing controllability and observability during prototype testing.

### 3.4 IEEE 1149.1 CONFORMANCE TESTING

The paper by Dahbura, Uyar and Yau [DAHB 89] describes a method for generating conformance tests for an 1149.1 TAP controller and for the instruction register. The method is based on the theory of checking experiments as applied to finite state machines, but uses a de-generate form called a Unique Input/Output (UIO) sequence. A UIO sequence is an input/output sequence of minimum length starting from one of the states, Si, that could not be produced starting from any other state in the circuit. The TAP controller is a classical Moore machine, and from a test point of view, it is only necessary to demonstrate the existence of each of its sixteen states and each of the specified transitions. It is not necessary to locate the cause of failure if failure occurs. If it were, then the more complete form of checking experiment would have to be used.

The sequences produced use a form of flush test (110011 sequences) to check the behaviour of the instruction register, bypass register, boundary scan register and identification register (if it exists). The length of the sequence is dependant upon the length of the boundary scan path, N(bs) and the length of the instruction register N (ir), and is given by:

589 + 12 N(bs) + 23N(ir)  (no identification register)
or
1658 + 12N(bs) + 44N(ir)  (with 32 bit Identification register)

The techniques described in the paper have formed the basis for a recent commercial offering from AT & T called TAPDANCE [TAPD 90].

### 3.5 BOUNDARY SCAN & BUILT IN SELF TEST

BSA provides access to device Built In Self Test (BIST), although its primary benefit is for board testing. This allows device self test routines to be triggered as part of the functional board test, or even as part of a start up procedure.

There has therefore been a significant amount of development work in this field in the last two years. In general BSA cells developed by Plessey, Siemens, SGS Thomson, Motorola, Texas Instruments and AT & T support BIST and can be configured as pseudo-random test vector generators and signature analysers [DETT 89],[MCLE 89].

AT & T Bell Labs have devised an architecture for combining BSA and BIST techniques [SCHO 88] and outlined a complete test program for testing boards equipped with BSA and BIST [TULL 89]. Signetics Corp. described a systematic test process that combines BSA and BIST [USZY 89]. Philips presented the specifications and design of a self test mechanism for static RAMs compatible with the BSA [DEKK 89]. IBM [BLANC 84] made use of SRLs on primary inputs and primary outputs in their LSSD On Chip Self Test (LOCST) architecture. Other articles dealing with boundary scan and BIST are [VAN 89], [GLOS 89], [BRGL 89], [WANG 89],[HUDS 89].

### 3.6 ADAPTATION TO CAE TOOLS

The major concern of designers and test engineers when they decide to adopt the BSA is their lack of experience, the scarcity of support tools and the impact on their development time schedule. Use of BSA will only be widespread when the tools required to implement and support this test technique are available and when the full implications of BSA implementation are well known.

Recent publicity and promotion of Design for Test (DFT) techniques have been noted by vendors of CAD tools [GUNN 90]. Chip and Board manufacturers wanting to use these techniques were confronted with the lack of software tools to support them. However in 1989, Gateway Automation (now part of Cadence), HHB Systems (now part of Racal Redac), and Teradyne began to offer design verification packages for scan path techniques.

VLSI Technology Inc, has announced their Test Engineers Assistant (TEA) [HALL 89], a CAD environment with tools to support the design of testable devices. TEA interfaces with ADAS, VHDL, and TISSS and with commercially available tools to allow design capture, functional verification, design for testability, fault simulation, and test program generation for particular ATE.

### 3.7 BSA AND TEST PATTERN GENERATION

Because most of todays electronic circuits are dense and complex, the electronics industry is eager to develop methods for automatic generation of test programs with high fault coverage. BSA simplifies the test generation problem by partitioning the overall circuit into a collection of smaller circuits.

As early as 1982, Goel and McMahon [GOEL 87] presented a technique for assembling a test for interconnect failures assuming the existence of boundary scan cells in the devices. This work has been extended by Wagner [WAGN 87], Hassan et al [HASS 88] and Jarwala and Yau [JRAW 89].

Generally, the interconnect pattern generation algorithms are either one step algorithms or adaptive algorithms. One step algorithms rely on a sufficient set of patterns to first detect the fault and then provide enough information to allow accurate analyses of the cause of the fault. Adaptive algorithms contain a fixed sequence section (to detect condition) followed by an adaptive section (to enable fault location with the minimum of additional sequences). The first algorithm of this type was reported by Goel and McMahon [GOEL 87] and subsequent algorithms (one test and optimal C Test) by Jarwala and Yau [JARW 89].

## 3.8 BSA AND ATE

Both chip and board manufacturers are beginning to use scan path techniques for design verification and/or production tests. The major problem is the lack of ATE hardware and software tools to support this. In 1988, Gateway Design Automation HHB and Teradyne (among others), offered limited design verification packages and Gillytron Inc (now called Brothers Electronics) announced a device tester tailored to verify the design of ICs equipped with scan circuit [BST 89, issue 1, P.4].

The Gillytron scan tester is based on the VXI backplane and controlled by a PC. Two other PC-based testers for scan based boards are the ASSETT system developed at Texas Instruments and Marconi's Checkmate/Midata 510 low cost ATE, equipped with boundary scan card and ATPG software package [BST 89, issue 2, p.5).

IMS have announced support for scan, including boundary scan, on their XL device verification tester; similarly Semiconductor Test Systems on their STS8500 device tester [MCLE 89].

Teradynes' response to the impending need for board testing tools to support BSA boards was the introduction of the L300 board test family "representing a new definition of combinational testing, including concepts of BSA, LSSD and BIST" [VERE 89]. The L300 board tester automatically generates patterns and provides the algorithms necessary to diagnose the failed nets on boards equipped with boundary scan. The implemented BSA strategy is described in [HAN1 89].

Another important development was the joint TYPHOON project between Motorola and Schlumberger Technology's ATE division to develop a device tester for the next generation of Motorola devices supporting the 1149.1 standard [TIME 89]. The specification of the ATE was impressive; dynamic tests on 1024 pin CMOS, bipolar or BiCMOS chips at frequencies up to 80MHz, and all this for just $1,500 per pin.

Clearly, it is not sufficient just to provide hardware facilities on the testers. There must also be some overall test strategy, taking into account the fact that most boards in the future will probably contain a mix of devices - some with BSA features, and others without. Those devices with BSA features provide access to the non-BSA devices through the boundary scan cells and the interconnects between the two types of device. Hansen called this form of access "virtual ATE channels" [ROSE 89], [HAN2 89], [HAN3 89].

Halliday et al of Texas Instruments [HALI 89] also discussed the use of the TI's octals to enhance both controllability and observability of non-boundary scan devices on a printed circuit board. This approach enables a form of cluster testing, that is, the non-BSA devices are treated as a single entity (cluster) and tested from the periphery.

In this respect, cluster testing is similar in application to the more familiar form of cluster testing through a bed of nails.

Given that the boards will contain a mix of BSA and non-BSA devices , and that there will be limited in-circuit access through the bed-of-nails fixtures, a possible strategy is as follows:

| | | |
|---|---|---|
| Power Off Tests: | nail contact integrity<br>short tests | |
| Power On tests: | TRST test<br>daisy chain test<br>BYPASS test (*)<br>BOUNDARY SCAN test (*)<br>Chip identification test | BSA Infrastructure Tests |
| | board interconnect tests<br>analog device tests<br>digital device tests, including BIST cluster tests | |
| Performance test | | |

(*) *Note these two test are optional . They are only performed if the BYPASS and the BOUNDARY SCAN paths were not tested at chip test.*

The daisy chain test checks the chip-to-chip TDO and TDI interconnects through the instruction registers, by loading the hard-wired 01 sequences. These values provide the basis for the distributed checkerboard test. An all-1s sequence is fed in from the TDI edge connector, leaving the instruction register holding the mandatory BYPASS instruction. A further checkerboard is then passed through all the bypass register bits to check their behaviour. Finally, the instruction register is loaded with a mandatory SAMPLE/PRELOAD instruction and a further checkerboard is used to check out the boundary scan registers.

The chip identification test interrogates the contents of all chips containing an Identification register. Devices without an Identification register are automatically configured into their BYPASS function. Capturing the identification values into the Identification register places a 1 in the least-significant position of the 32 bit Identification register. Capturing the BYPASS register places a 0 in the BYPASS register. This difference allows "blind" interrogation of a board containing a collection of 1149.1 devices - some with Identification register and some without [MAUN 90]. This has value on boards containing different variations of programmable devices such as EPLDs and EPROMs.

Once it is known that the BSA infrastructure is working, then the remaining tests can be applied followed by analog and digital device tests where necessary. In addition, cluster tests using the boundary scan cells can be used as virtual ATE channels.

Full exploitation of the advantages of the BSA technique in the field of fault diagnosis will require a close partnership between the CAE and ATE vendors. The ultimate goal is to create an integrated environment for the incorporation and the use of BSA for design, test and diagnosis purposes and supporting international standard formats. An important step in this direction has been taken by Hewlett-Packard. Parker and Oresjo outlines a "strawman" draft of a language called Boundary Scan Description Language (BSDL) to describe the features of a device conforming to the 1149.1 standard. [PARK 90]

The BSDL language allows the description of information about the testability features in devices that are compliant with IEEE Standard 1149.1 - 1990. This information is needed by tools which will make use of these testability features. Such tools include design verification, testability analysis, test generation and diagnosis.

The syntax of the language was developed as a VHDL subset [PARK 90]. The language describes the basic boundary scan information fundamental to an application, but it is not rich enough to support advanced capabilities. It does not give the flexibility to describe logic functions that may be incorporated in the boundary register itself, such as Texas Instruments have provided in their current set of devices [TI 1988]. Extensions to the language are being developed and the language has recently been ratified by the JTAG Committee.

### 3.9 BSA AND POWER SUPPLY TERMINALS

In a novel paper, van de Lagemaat [LAGE 89] described an application of boundary scan to the on-board testing of multiple power supply terminals into a single device. Many complex devices require multiple supply voltage and multiple ground connections for reasons to do with noise immunity. The problem is - how to detect open circuits in the internal bond wires of these groups of connections?

Consider multiple supply voltage terminals. If all the terminals are correctly connected, then the voltage difference between them is of the order of 10mV. If one or more of the connection is open circuit , then the voltage difference is more like 100mV. This gives a basis for carrying out an open circuit test - measure the voltage between the connections inside the device.

The technique described in the paper advocates the use of a circuit similar to a RAM sense amplifier to sense the difference in the analog voltage and convert it into a corresponding digital voltage. The output of the sense amplifier is then captured by an observe only boundary scan cell, and thus made observable through the boundary scan chain.

## 3.10 BSA AND ANALOG COMPONENTS

Very little has been done yet on the application of BSA to analog or mixed signal devices. Fasang of National Semiconductor has described a multiplexer/de-multiplexer technique for partitioning the analog and digital sections of a mixed signal device [RASA 89]. Output boundary scan cells are used to observe the digitised output of the analog part of the circuit, and input boundary scan cells are used to control the input to the digital part of the circuit. The boundary scan cells used for these purposes are not part of the normal input/output pins of the device.

## 3.11 SYSTEM APPLICATIONS OF BSA

It is too early to report on system company applications of boundary scan, with one notable exception - that of the Apollo Company (now part of Hewlett Packard). The design of the Apollo DN10000 workstation included the use of boundary scan features [DEVI 88] in the gate arrays. These scan cells were not compatible with the 1149.1 since the Standard was still in its earlier JTAG form when the DN10000 design decisions were made. Nevertheless, all gate arrays on the DN10000 board included observe only boundary scan cells on the pin-out of the chip, linked to form an "external ring" . The cells were then used to act as an observer of the results on tests applied internally to the devices and to the interconnects. In principle therefore, the external ring followed the spirit of 1149.1.

## 3.12 BSA AND RELATED DEVELOPMENTS

Among other papers related to boundary scan, Landis [LAND 89] describes the use of 1149.1 as a standard interface for wafer testing to reduce test complexity and costs. Bassett et al [BASS 89] discusses the architecture of a logic device tester for components equipped with BSA and array BIST. Hassan et al [HASS 89] proposes a test scheme for NON-BST glue logic interconnects using modified BSA cells for fast test-vector application and evaluation. The paper is an extension of the earlier paper from Hassan [HASS 88].

Levitt and Abraham [LEVI 89] and Dislis et al [DISL 89] deal with the economics of DFT measure like scan, BIST and BSA. Levitt and Abraham consider the general problem of cost justifying the inclusion of scan (any form) into the design of an integrated circuit. Factors include: gate count increase, effect on yield, number of die per wafer, and cost per die. The effect on profitability is then considered. Profitability has been defined in terms of "time-to-market slippage" and is related to the ease or difficulty of creating test programs. This is a simplified approach but future version of the analysis will include more complex factors such as the effect of pattern generation and fault simulation costs.

Finally, Swan et al [SWAN 89] describes a modified gate array using the 1149.1 TAP to increase controllability and observability inside the chip. Advantage can then be taken of the increased testability at chip and board level.

## 3.13 VHDL FOR CAD/CAE/CAT

The life cycle of a typical digital system starts with system engineering and specification, which then proceeds into hardware and (software) design and development, manufacturing and field service and completes the cycle with re-procurement engineering.

The life cycle of a commercial digital system ranges from 6 to 12 years and is very sensitive to competition and timeliness in introducing a new system to the market. Therefore, one of the major concerns is the efficiency and compatibility of the CAD/CAE/CAT tool set to reduce the time from requirement identification to system delivery. The life span of a military digital system may on the other hand range from 15 to 25 years so the components may become obsolete before a system is ready for use.

Additional problems of logistic support and re-procurement engineering must be taken into account. Military systems are also on average more complex and involve more organisations than their commercial counterparts. During the product life cycle , companies which have developed the technology or inserted the technology in a hardware design, may not be involved in full production and logistic support.

In addition, there are a large variety of incompatible CAD/CAE/CAT tools each of which may require a special language and a specific interface , and data and control format. For the most part, these languages are proprietary, having evolved from old designs which do not have the features of modern languages. Some of these languages are difficult for designers to read, understand or use. Each of the tools and its supporting language is designed to support a limited number of abstractions or design methodologies. This makes sharing of design among different organisations or across levels of design difficult. To cross tool or vendor boundaries, interface software or translators are often required, which are prone to additional errors. Design languages and test vector languages are not compatible causing inefficiency and additional errors.

Test vectors used at chip level are not correlated with those for the board level testing. Up to 50% of the ASICs designs which pass the initial chip level test do not pass the board

level test because of design problems or test compatibility problems.

VHDL was developed by the DoD VHSIC program as part of the Integrated Design Automation System (IDAS). The history of its development which has been well-reported, will not be repeated here. VHDL did not evolve from an existing hardware description language, it was developed specifically to meet well-defined requirements.

Basically, VHDL describes a hardware component or building block (module, chip, device) as a design entity. A design entity consists of two parts: an interface description and a body description. The interface descriptions define the input/output ports through which the design entity communicates with the outside worlds. The body description defines the internal operation or organisation of the entity. In VHDL, the entity declaration is used to define the interface between a given design entity and the environment in which it is used. The architecture body defines the body of the design entity. It specifies the relationship between the input and the output of a design, and may be expressed in terms of behaviour, data flow, structure or any combination of these.

The behavioural model is represented by a data transformation and a timing relation in response to a data transformation at the input. The basic concept of a structural model is represented by the component, the port and the signal. The component corresponds to a hardware building block. A port is the point of connection to their components and the point through which data flows in or out of the components. A signal is defined as a path from one component to another component along which data flows.

In the VHDL behavioural model, the concept is very similar to those used in functional simulations using High-Order Language (HOL).

Algorithms or functions are used to describe the behaviour of the system. Output responses are calculated from the corresponding inputs. The basic statement for a behavioural description is the process statement. Each process statement defines a specific action to be performed which models the specific behaviour. The action is triggered when the value of one of its sensitivity signals (signals specified by the designer) changes. In the sequential model, the action is defined by the sequential order of the statements in the process.

VHDL provides sequential statements such as IF, ELSE, CASE, LOOP, WAIT, PROCEDURE CALL, to define the algorithms or functions which model the behaviour of the system. Concurrent statements are also provided to simplify description of processes which have the same behaviour. A behavioural description provides the information on how a building block behaves without the details of how it is constructed. This type of description is useful in top-down design and functional decomposition, in communication design among different design groups and in describing a device without disclosing proprietary details of its design.

A VHDL description can also be expressed in data flow format. VHDL provides several concurrent signal assignment statements which are triggered by changes in value of the input to a signal assignment statement and executed asynchronously with respect to one another. The execution of the concurrent assignment signal involves computing new output values after a specified or a default delay to the output signals of the statements.

The structural description in VHDL parallels the physical system closely. The basic elements of structural description are ports and their connections. In VHDL port declaration, the number and mode of ports, the direction of the data flow and the type of data are described.

Structural descriptions define components and their interconnects using signal constructs to define paths among components. The component instantiation statement specifies an instance of a component and which port or signal is to be connected to. A generate statement provides a means for iterative or conditional elaboration of a portion of the description for components which exhibit some degree of regularity. Generic statements provide a means for instantiating components to pass values to another instantiated component. The configuration specification specifies the section of entity declaration and architecture body for each component instance.

VHDL also provides the capability to mix various description models. This allows the design of different portions of the system to proceed without having all the structure details available. In addition it enables use of behavioural simulation for portions of the design which have been simulated structurally. Therefore in this way simulation of a larger piece of design can be carried out more rapidly.

There are many ways to integrate VHDL into an EDA environment. Each CAE vendor uses its own approach to implement VHDL. Some vendors implement a VHDL subset to facilitate interfacing with their existing environments and tool sets, to allow transition of the current customer base to full VHDL implementation at a later date. Others elect to implement the full capability as specified in the language reference manual (LRM). Some vendors use an intermediate language, others directly compile from VHDL source code to simulation language. Some vendors have bridged VHDL to other tools such as editors, debuggers, logic synthesis or even RTL-level synthesis. Still others have interfaced the VHDL based simulator to their automatic test generation tools. DoD have developed a tester-independent test support software (TISSS) environment, and defined a test vector generation language (TVL), which is a subset of VHDL. All these provide a link between design and test.

## 3.14 VHDL FEATURES FOR CAD/CAE/CAT

VHDL's capability, support, interface and potential benefits during the life cycle in general, and for design and test specifically, are summarised below:

<u>Benefits over the entire Life Span of a Hardware System</u>

- Formal precise, human-readable and easily understood description for use in all phases of the life cycle
- Machine processable and simulatable
- Technology independent
- Tool set, data type and environment-independent
- Design modularity and re-usability
- Overall hardware system acquisition process
- Acquisition and logistic support
- Efficient management of the design and the design data, library, configuration control
- Interface with High Order Language (HOL)
- Compatibility with other standards

<u>Benefits in Systems Engineering, Hardware Design and Development</u>

- Various combinations of abstractions - behavioural, data flow, structural
- Partition of functions and structures
- Sequential and concurrent processes
- Multiple design methodologies; top-down, bottom-up or mixed
- Various digital modeling techniques: Boolean, finite state, algorithmic and functional
- Both synchronous and asynchronous designs
- Various design methodologies: custom standard and macro cells, ASIC, gate array, off-the-shelf components or any combination.

- Different styles of description and documentation; behavioural, structural, data flow, procedural or various combinations.

Benefits in Engineering, Manufacturing, Operational and Test

- Design for testability
- Compatibility with test vector language
- Test independent support software environment
- Compatible data type between design modelling and test
- Boundary scan test concept at board and module levels

Benefits in Re-procurement Engineering

- Technology independent design descriptions
- Test independent test descriptions
- Simulator independent modeling

## 3.15 IN CONCLUSION - THE PROPOSED PROJECT

A need for implementing JTAG in designing ASICs was recognised in 1989. A mechanism that will shield the ASIC designer from becoming involved in interpreting and designing the JTAG 1149.1 Standard is needed. This could take the form of automatic or semi-automatic injection of JTAG into an ASIC, in a format which is both generic and technology non-specific.

The emergence of VHDL as the IEEE 1076 Standard Hardware Description Language has resolved the problem of portability and independency. This enforced the choice of VHDL as the modeling tool for this project.

In addition, VHDL's ability to model digital systems at behavioural and structural levels, has enabled the development of JTAG high level behavioural models but preserved the features that are normally exhibited at the structural level.

This research encompasses the development of a software tool that will enable the ASIC designer to include the full 1149.1 BSA into the design in a semi-automatic way.

The work models the BSA both structurally and behaviourally using VHDL. It also develops a parser which identifies and extracts a list of the input/output terminals of the ASIC design. It then inserts the full BSA high level parameterised models into the design, without altering the structure or the order of I/O terminals of the original design.

The parsing insertion environment has to be simple, portable and independent of any CAE environment. Therefore, it needs to be developed in a high level programming language such as "C". Unlike the Hewlett-Packard BSDL language, the work therefore concentrates on developing high level testability models of JTAG using VHDL. This takes advantage of the language facilities provided by VHDL in dealing with the problem of annotating internal state information from the structural or RTL level to a much higher level.

The tool is not intended to be an aid to synthesis. It does not deal with performance, speed and area issues. It is mainly a tool that will assist the designer to include JTAG into his/her design with ease. The design (ASIC) has to be either described in VHDL or has been converted to a VHDL description. The test features could be included in the design right from the behavioural description level.
The test vectors for testing JTAG are developed and fault graded.

The environment has been tested with simple examples using a suitable CAE system in order to simulate the design with the JTAG models included.

**REFERENCES**

[PARK 89] K Parker, "The impact of Boundary Scan on Board Test", *IEEE Design and Test of Computers,* August 1989 pp 18-31.

[JTAG 90] *IEEE Standard 1149.1-1990* "Test Access Port and Boundary Scan Architecture.

[ROBI 90] G D Robinson "Boundary Scan Impact on Board Test Strategies" *Prod.Electo.* May 1990, pp 1-8.

[EVAN 89] S Evanczuk, "IEEE 1149.1: a designer's reference" *High Performance Systems* August 1989 pp 52 - 60.

[POUND 89] R Pound "The Technological State of the Industry" *Electronic Packaging and Products,* January 1989 pp 68-71.

[COLE 89] H C Cole, "Making the right moves in ASICs", *Electronics* November 1989, pp 56-59.

[BAS1 89] R W Bassett et al., "Low cost testing of High Density Logic Components", *Proc IEEE Inter. Test Conf.* 1989 pp 550-557.

[BURS 89] D Bursky, "Digital ICs in the 1990s: Nearly unlimited on-chip resources", *Electronic Design* 11 January 1989 pp 79-89.

[MAUN 90] C M Maunder and R E Tulloss, "The Test Access Port and Boundary Scan Architecture: Tutorial of IEEE 1149.1 and its applications" *IEEE Computer Society Press* 1990.

[BEEN 87] C M Maunder and F M Beenker, "Boundary Scan: A framework for Structured Design for Test", *Proc. IEEE Inter. Test Conf.* 1987 pp 714-723.

[IEEE 90] IEEE Testability Bus Standards Committee, details from the Chairman, Gordon Robinson, GenRad Inc, Concord, MA USA.

[TBUS 87] "VHSIC phase 2 interoperability standards: TM-Bus Specification", version 30, November 9, 1987. Available from J P Letellier, Naval Research Lab, Code 5305, Washington DC 20375, USA.

[AVRA 87] L Avra, "A VHSIC ETM-Bus compatible Test and Maintenance Interface", *Proc. IEEE Inter. Test Confer.* 1987 pp 964-971.

[DETT 89] R Dettmer "JTAG setting the Standard for Boundary Scan Testing", *IEE Review*, February 1989, pp49-52.

[BST 89] BST News, a quarterly publication of ESPRIT-2 Project 2478 on Boundary Scan Test. Issue 1 was published in March 1989. Details available from BST Project Secretariat, HJK p818, CFT Automation, Philips NPB, Box 218, 5600 MD Eindhoven, Holland.

[DUPT 84] S das Dupta et al. "Chip Partitioning Aid: A Design Technique for Portability and Testability in VLSI" *Proc ACM/IEEE Design Automation Conf*, 1984 pp 203-208.

[PHIL 90] "Boundary Scan Test: The Structural Design for Test Standard" 1990. A brochure available from any Philips Components ASIC Design Centre.

[MCLE 89] D McLean et al.,"Bringing 1149.1 into the real world" *High Performance Syst.*, August 1989 pp 61-70.

[WILS 89] R Wilson,"TI takes the lead in JTAG support", *Compu.Design New Ed.*, January 1989.

[PERR 89] T S Perry "Intel's Secret is out", *IEEE Spectrum*, April; 1989.

[VAN 89] R P van Rissen et al., "Design and Implementation of Hierarchical Testable Architecture using Boundary Scan Standard", *Proc. European Test Confer.*, 1989 pp 112-126.

[VINI 89] S Vining, "Trade-off decisions made for a P1149.1 controller design" *Proc IEEE Inter. Test Confer.*, 1989 pp 47-54.

[BALL 89] W D Ballew and L M Streb, "Board Level Boundary Scan: Regaining Observability with an Additional IC", *Proc., IEEE Inter. Test Confer*. 1989 pp 182-189.

[LIEN 88] J C Lien and M A Bruer, "A Universal Test and Maintenance Controller for Modules and Boards," *IEEE Trans. Indus. Electron.*, 36(2): 231-240, 1988.

[DAHB 89] A T Dahbura, M Umit Uyar and C W Yau, "An Optimal test sequence for the JTAG/IEEE P1149.1 Test Access Port Controller," *Proc. IEEE Inter. Test Confer.*, 1989 pp 55-62.

[SCHO 88] H N Scholz et al, "ASIC Implementation of Boundary Scan and BIST," *Proc. Inter Custom Microelectronic Confer.*, November 1988, London Heathrow.

[TULL 89] R E Tulloss et al., "BIST and Boundary Scan for Board Level Test: Test Program Pseudocode," *Proc. European Test. Confer.*, 1989 pp 106-111.

[USZY 89] P A Uszynski et al, "Hybrid Global test Strategy", *High Perform. Syst.*, January 1989 pp 68-74.

[DEKK 89] R Dekker et al, "Realistic Built in Self Test for Static RAMs" *IEEE Design Test Comput.*, February 1989 pp 27-34.

[BLANC 84] J J le Blanc, "LOCST: A Built in Self Test Technique", *IEEE Design Test Comput.*, November 1984, pp 45-52.

[GLOS 89] C S Gloster and F Brglez, "Boundary Scan with Built-in Self Test," *IEEE Design Test Comput.*, February 1989 pp 36-44.

[BRGL 89] F Brglez, C Gloster and G Kedem, "Hardware based weighted random pattern generation for boundary scan," *Proc IEEE Inter. Test Confer.*, 1989 pp 264-274.

[WANG 89] L T Wang et al, "A Self test and self diagnosis architecture for boards using boundary scan," *Proc. European Test Confer.*, 1989 pp 119-126.

[GUNN 90] L Gunn, "CAE in the 1990's: Will users want performance or integration?" *Electronics Design*, 11 January 1990 pp 53-63.

[HALL 89] J J Hallenbeck et al, "The Test Engineers Assistant: A support environment for hardware design for testability', *IEEE Computer*, April 1989 pp 59-68.

[SAMA 89] A Samad and M Bell, "Automating ASIC Design-for-Testability: The VLSI Test Assistant", *IEEE Inter. Test Conf.* 1989 pp 819-828.

[MCLE 89] J McLeod, "Bringing down the cost of ATE", *Electronics* November 1989 pp 76-79.

[FLUK 89] J M Fluke Jr,"Standardizing instruments will increase competitiveness," *IEEE Spectrum* January 1989 pp 50-52.

[GOEL 87] P Goel and M T McMahon, "Electronic chip-in-place test," *Proc. IEEE Inter. Test Conf.*, 1987 pp 83-90.

[WAGN 87] P T Wagner, "interconnect testing with boundary scan", *Proc. IEEE Inter. Test Conf.*, 1987 pp 52-57.

[HASS 88] A Hassan et al, "Testing and diagnosis of interconnects using boundary scan architecture," *Proc. IEEE Inter. Test Conf.*1988 pp 126-137.

[JARW 89] N Jarwala and C W Yau, "A new framework for analysing test generation and diagnosis algorithms for wiring interconnects," *Proc. IEEE Inter. Test Conf.*1989 pp 63-70.

[HAN1 89] P Hansen et al. "Tough Board Test Problems solved with boundary scan", *Electronics Test*, June 1989 pp 34-40.

[VERE 89] L Vereen, "Board Tester supports Boundary Scan", *Electronics Test*, June 1989 pp 30-31.

[TIME 89] *Electronics Times*, 29th June p 12.

[ROSE 89] P Hansen and C Roseblatt, "handling the transition to boundary scan for boards," *High Perform. Syst.*, August 1989 pp 74-81.

[HAN2 89] P Hansen, "Testing conventional logic and memory cluster using boundary scan devices as virtual ATE channels," *Proc. IEEE Inter. Test Conf.*1989 pp 166-173.

[HAN3 89] P Hansen, "Strategies for testing VLSI boards using boundary scan," *Electronic Engineering*, November 1989, pp 103-111.

[HALI 89] A Halliday, G Young and A Crouch, "Prototype testing simplified by scannable buffers and latches," *Proc. IEEE Inter. Test Conf.*1989 pp 174-181.

[PARK 90] K Parker and S Oresjo, :A language for describing boundary-scan devices,' to be presented at the 1990 *IEEE Inter. Test Conf.*

[LAGE 89] D van de Lagemaat, "testing multiple power connections with boundary scan,' *Proc. European Test Conf.*, 1989 pp 127-130.

[RASA 89] P Rasang, "Boundary scan and its application to analog-digital ASICs testing in a board/system environment," *Proc. IEEE 1989 Custom IC Conf.*, pp 22.4.1-22.4.4.

[DEVI 88] B I Dervisoglu, "Using scan technology for debug and diagnostics in a workstation environment", *Proc. IEEE Inter. Test Conf.*1988 pp 976-986.

[LAND 89] D L Landis, "A self-test system architecture for re-configurable WSI," *Proc. IEEE Inter. Test Conf.*1989 pp 275-282.

[HASS 89] A Hassan et al., "Testing of glue logic interconnects using boundary scan architecture", *Proc. IEEE Inter. Test Conf.*1989 pp 700-711.

[LEVI 89] M E Levitt and J A Abraham, "The economies of scan design,", *Proc. IEEE Inter. Test Conf.* 1989 pp 869-874.

[DISL 89] C Dislis et al., "Cost analysis of test method environments," *Proc. IEEE Inter. Test Conf.* 1989 pp 875-883.

[SAWN 89] G Swan, Y Trivedi and D Wharton, "CrossCheck - a practical solution for ASIC testability, *Proc. IEEE Inter. Test Conf.*1989 pp 903-908.

[HUDS 88] C L Hudson, "integrating BIST and boundary scan on a board," *Proc. Natl. Commun. Forum*, 3-5 October 1988 pp 1796-1800.

[TAPD 90] TAPDANCE, details available from AT &T, P.O.Box 4911, Room 112-3A33, Warren, NJ 07059-0911 USA.

[BASS 90] R W Bassett et al, "Boundary scan design principle for efficient LSSD ASIC testing", *IBM J Res. Devel.*, 34(2/3):339-354, 1990.

[TI 1988] Texas Instruments, Data Sheet (Preliminary) SN54BCT8374, SN74BCT8374 Boundary Scan Device with Octal D-Type Flip-Flop, TI Inc., Dallas TX, 1988.

# CHAPTER 4
# VHDL DESIGN AND MODELLING TECHNIQUES

## 4.0 INTRODUCTION TO VHDL

Hardware Description languages (HDL) provide a way to textually represent physical electronic systems. [Wax 1], [Wax 2]. They are used for the description, documentation and communication of digital electronic designs. More recently, they have been used for design verification, simulation and synthesis.

VHDL - 1076 (VHSIC (Very High Speed Integrated Circuits) Hardware Description Language) is an IEEE Standard since 1987 [stan 1076]. "It is a formal notation intended for use in all phases of the creation of electronic systems. It supports the development, verification, synthesis and testing of hardware designs, the communication of hardware design data....." [Preface to the IEEE Standard VHDL Language Reference Manual] and especially the SIMULATION of hardware descriptions. Note that this chapter is not meant as an introduction to VHDL; that is beyond the scope of this thesis. The complete VHDL specification is documented in the current IEEE standard manual [stan 1076].

This chapter gives an overview of the VHDL language. It identifies the reasons for choosing VHDL as the modelling language for modelling the Boundary Scan Architecture. It also highlights the main ingredients of the modelling environment. In doing so, it describes the various description styles and abstraction levels available for the designer. In addition, this chapter describes the role of VHDL in the design cycle of digital systems and identifies where VHDL fits within the CAD environment.

## 4.1 DEFINITION OF VHDL

VHDL is a language which can be interpreted by humans and machines. It is used to define the behaviour of an electronic circuit or system in an unambiguous way. In conjunction with appropriate CAE software tools (such as Mentor Graphic's "QUICKSIM"), it can also define and implement simulation of a system or circuit.

The language allows simulation to take place from the SYSTEM level down to the GATE level. The language can be used to define the behaviour of entire systems, single boards, chips, single gates or an intermediate structure.

The most immediate and widespread use of VHDL is in the design and simulation of integrated circuits.

## 4.2 OPEN - SYSTEM DESIGN AUTOMATION ARCHITECTURE

Utilising VHDL as the HDL language for implementing ASIC designs will enable the use of CAE environments which support an open-system architecture (such as Mentor Graphics), with the well defined ability to interpret standards to ensure effective interfaces for the designer and his/her tool.

## 4.3 VHDL AND THE ASIC DESIGN PROCESS

As Integrated Circuit complexity increases, circuits become more specialised and their broad applicability decreases. There are several reasons why a standard hardware description language like VHDL is important to the ASIC (Application Specific Integrated Circuit) design process.

1) The ASIC design approach requires advanced design automation systems with clearly defined interfaces between the customer and the vendor. This can shorten the development cycle thereby reducing the associated costs.

   The three issues critical to the ASIC market that need to be dealt with are the design interface, true second sourcing and high performance fabrication processes. These issues are the driving forces behind the new approach. Since CAD advancements have happened so quickly, conventional Integrated Circuit manufacturers have been unable to catch up with CAD interface technology.

2) VHDL provides user documentation of the ASIC design and facilitates design second sourcing.

   In an ASIC design service, vendors usually cannot supply detailed documentation such as manuals, books and application notes typically associated with standard components because every device is different and may or may not be vendor designed.

3) VHDL enables the ASIC design service vendors to protect proprietary design system elements, giving them a competitive advantage.

Figure 4.1 demonstrates where VHDL is used in relation to the system design process.

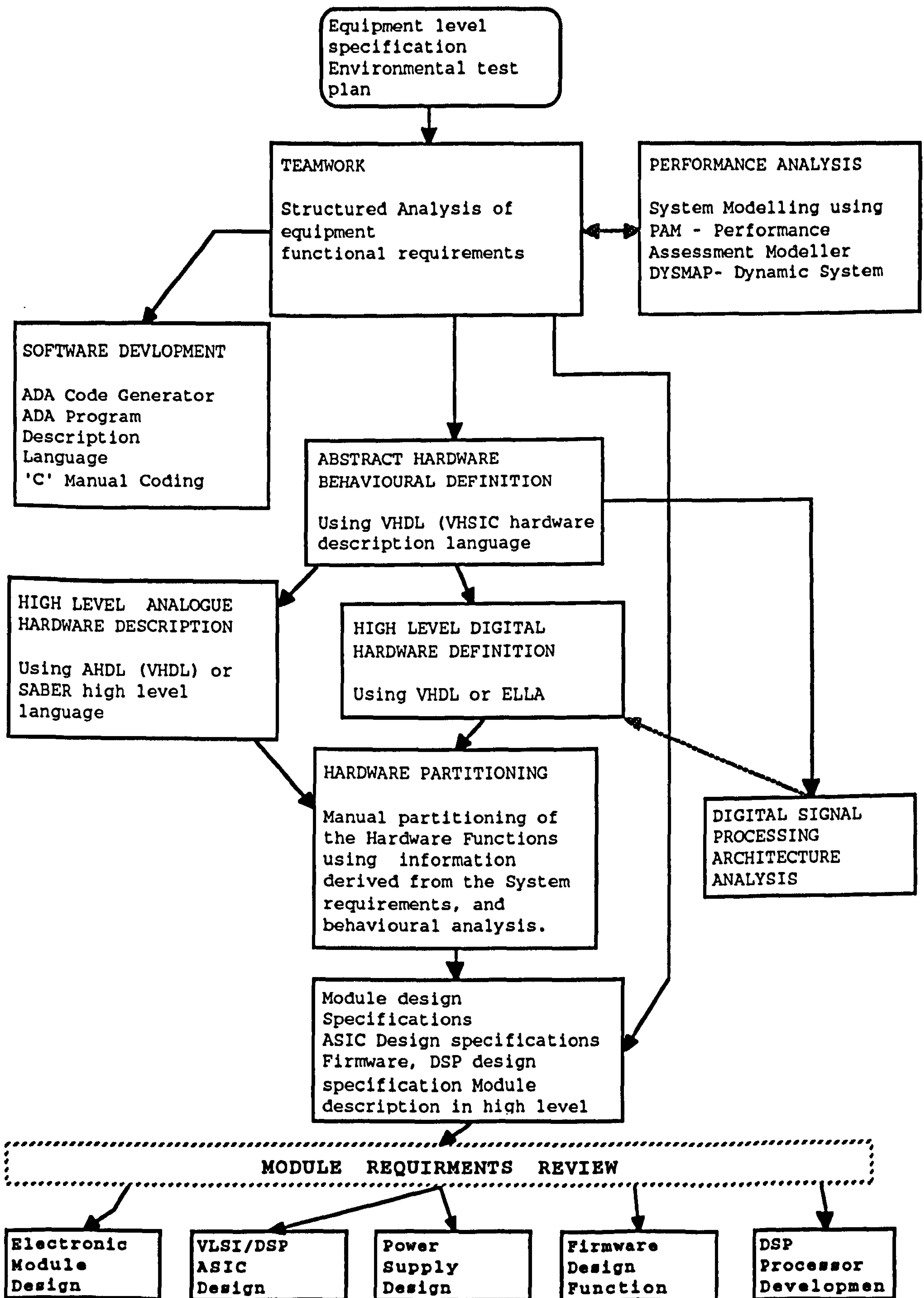



FIGURE 4.1 VHDL AND SYSTEMS DESIGN

VHDL is currently used after the hardware functions derived from the system analysis stage have been partitioned.
It is at this point in the design cycle that the use of VHDL is most appropriate. An important advantage of VHDL description at the system hardware level is that, for the first time, simulation becomes a realistic possibility.

Features such as unambiguous hardware description and reusable hardware sub-systems are obvious and highly desirable objectives, allowing alternative designs to be simulated and design trade-offs to be investigated.

## 4.4 VHDL DESIGN HIERARCHY

Three main views of design hierarchy are behavioural, structural and physical. The more established HDL's tend to cover just one of these three views and have a dedicated application.

The behavioural model describes the relationships between the ports of a node without referring to the internal logic or the physical structure.

VHDL serves as a vehicle for investigating new approaches to design techniques, models and automation in areas such as test, synthesis and simulation. Knowledge about hardware properties and characteristics, applicable to design is the very essence of language constructs comprising VHDL.

The various representations call for different areas of expertise, e.g. at the functional level one needs skills in the fields of architecture and machine design. At the structural level a good background in logic design and simulation would be essential.

VHDL offers a framework for more behaviour-orientated design, in which the behavioural axis has Boolean expression at the lower levels. Algorithms are the next level up. The highest design level is the system input/output specification.

The ability to consider electronic system software and hardware jointly has tremendous potential in improving software/hardware performance trade-off analysis and system synthesis.

The structural model describes a node in terms of its sub-components and their inter-connections. When more than just structural information is needed, such as size or position of ports around the perimeter, then it is integrated with the physical model.

To cover all aspects of design a HDL really needs to be able to have all of these aspects, but at different points in time. For example, detailed structural information would not be wanted in the first stages when the design is more abstract. The four uses of VHDL are:-

- Specification of the functionality and performance of a board or system with a view to simulation and verification of the design.

- As a source level description of hardware with a view to using logic synthesis tools to generate the physical implementation of a design.

- The modelling of components for inclusion in simulations at a detailed level.

- As a design transfer language.

Simulation - VHDL was designed to be used for a specific application or device and therefore, it cannot be synthesized easily.

Modelling - VHDL enables abstract models to be defined. These models can not, however, be mapped directly to the physical realisation. Indirect methods are available to enable this mapping using alternative design tools e.g. ELLA.

Transfer - VHDL can describe hardware at any level to cope with all types of designs and eventualities.

VHDL is mainly used for digital design and, although at the most abstract level analogue components can be described in a software like language, it cannot be used for the detailed design. Analogue HDL tools are therefore used for these parts.

Although there were many HDL's before VHDL it was the first to be standardised across the industry, making it the first truly standard HDL. VHDL has freed HDL's from vendors, enabling portability of designs, and of standard models of devices. One risk of this is that vendors will produce sub-sets and Vendor specific naming which will reduce this portability and be a great pity.

VHDL is maintaining the principles of Systems Analysis by enabling members of a team to work concurrently on different points of the overall design. The use of a common language enables the communication process which is essential if the design team is to function effectively.

When VHDL is employed, there can be many individuals working on different aspects of the design: system engineers developing global models and running system simulation; Hardware designers working on functional block diagrams; Test engineers designing appropriate test strategies. VHDL was designed to help the engineer to organise the design process.

The basis of abstraction is a design '**entity**', which gives the designer the '**black-box**' model. It is essentially the interface to the outside world, input/ output ports, and an '**Architectural Body**' which describes the structural and behavioural functions of the entity. There can be multiple architectural bodies for one design entity all giving different views of the same object

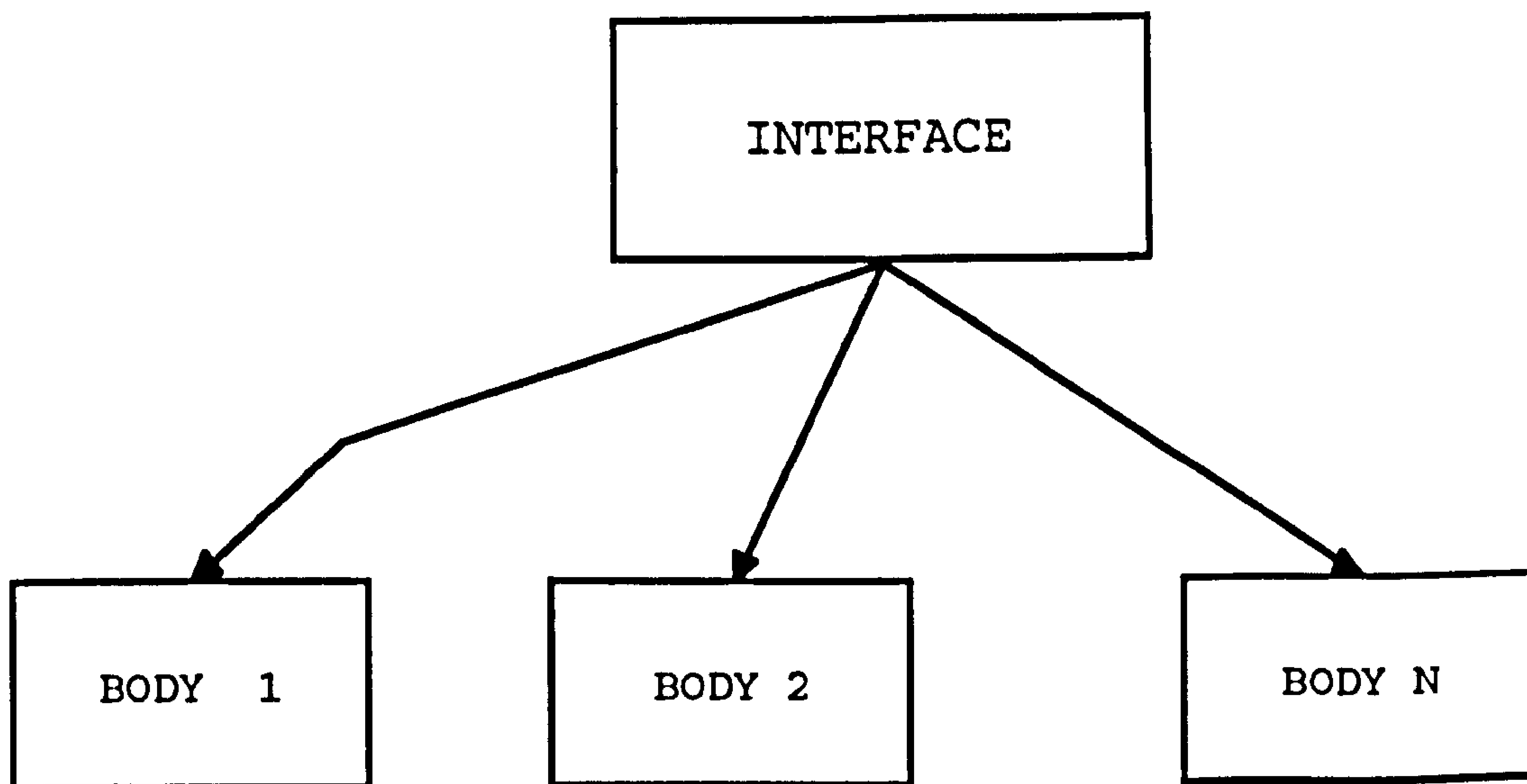



**FIGURE 4.2 A VHDL DESIGN ENTITY**

VHDL supports libraries from which designers can select pre-compiled designs and reference them in their design.

VHDL's real strength is in its ability to model a system over a wide range of abstractions and thus there is no ambiguity concerning movement from one level to another.

The area of design that VHDL is covering is from the systems analysis and division down to the gate level. VHDL cannot deal with switch or circuit level descriptions. It therefore needs to be supported by CAE tools to transfer the description into their schematic and simulation tools.

To summarise the attributes of VHDL, it:-

- Supports the design phase of development
- Interacts directly with humans and is immediately understandable
- Helps in the management of a design
- Communicates design information between people
- Supports the whole spectrum of design and test from system to chip.

## 4.5 VHDL MODES OF OPERATION

VHDL has two modes of operation:

1. Concurrent
2. Sequential

**Concurrent** -In the concurrent mode ALL statements that have been queued for evaluation are evaluated at the same time. This closely matches the action of the hardware devices in real life.

**Sequential** - In this mode of operation all statements are evaluated in strict sequence. No simulation time passes during execution unless explicitly stated in the code.

In order to simulate electronic hardware designs, the following information is required:

- Structural descriptions of the design (netlist or schematic)
- Behavioural model for each device in the design (VHDL source or model library)
- Stimulus for the design (test vectors)
- Design configuration information (specify which version of each device model to use during simulation).

## 4.6 VHDL STRUCTURES

VHDL models basically consists of two basic structures:

1. Entities
2. Architectures

**Entities.** These consist of the external worlds view of the model. They consist of port declarations of the inputs and outputs of the circuit and a number of common parameters such as rise and fall times.

**Architectures.** This section consists of the VHDL structural information. The circuit components, port to circuit connections and internal signals are all declared in this section. There are three basic levels of abstraction, these being the behavioural, data flow, and structural.

## 4.7 VHDL DESIGN HIERACHY AND DATA BASE

The basic structure within a VHDL design is the design entity. A single design consists of (potentially) many design entities, each entity describes the inputs and outputs of the design.

The entity declaration does not describe how the design entity functions, it simply defines the inputs and the outputs, thus providing an external view of the entity. Each design entity has at least one, but perhaps, more architectures associated with the entity declaration. Each architecture provides one possible way of describing the functionality of the design entity or one possible implementation of the design entity. Design entities may be hierarchically nested. This means that if one design entity is used in more than one part of the design, then it only needs to be defined once, but may be referenced (or instantiated) multiple times.VHDL gives us the flexibility to describe a design while maintaining any type of hierarchical partitioning or nesting.

Architectures describe how a particular implementation of a design entity should function. Architectures consist of two parts: the architecture declaration and the architecture body. The architecture declaration describes the various items used within the architecture body. These items include signals (which can be considered the same as wires) and references to other design entities which are nested inside the architecture. Within an architecture body may appear two types of statements, concurrent and sequential. These statements are used to describe the functionality of the architecture. VHDL's concurrent and sequential statements are an attempt to describe circuits which exhibit parallel and serial behaviour respectively. The basic data construct within both sequential and concurrent code is the assignment construct.

The main factors which make VHDL a better hardware description language to use are:

a) It has evolved from a set of user requirements
b) It includes Timing thus avoiding dependence on the simulation environment.
c) It can produce many different views and abstractions of the same model
d) It helps in the Organisation of a design.

### 4.8 DESIGN DESCRIPTION METHODS

System-1076 provides a textual method of describing a hardware design rather than a schematic representation. The following are various System-1076 methods for describing hardware architectures:

- Structural description method - expresses the design as an arrangement of interconnected components.

- Behavioural description method - describes the functional behaviour of a hardware design in terms of circuit and signal response to various stimuli. The hardware behaviour is described algorithmically without showing how it is structurally implemented.

- Data-flow description method - similar to a register-transfer language. This method describes the function of a design by defining the flow of information from one input or register to another register or output.

All three methods of describing the hardware architecture can be intermixed in a single design description.

### 4.8.1 STRUCTURAL DESCRIPTION

This subsection identifies some of the language constructs that are in a System-1076 structural description, by using an example of a two-input multiplexer. The description provides an overview, but not a complete representation, of all the language building blocks found in a structural description. A System-1076 structural description of a hardware design is similar to a schematic representation because the interconnectivity of the components is shown. This is illustrated in this subsection with a comparison of a simple NETED™ (from Mentor Graphics) design to a System-1076 structural description of the same circuit.

Figures 4.3 and 4.4 show the symbol-schematic representation of the two-input multiplexer. Note the pin names on the inside of the MUX symbol in Figure 4.3 match the net names of the inputs and output of the schematic in figure 4.4. This ensures connectivity between the two NETED sheets. The input and output ports (DO_IN, D1_IN, SEL_IN, and Q_OUT) are unrelated to the underlying sheet. They are used during simulation to access the inputs and output.

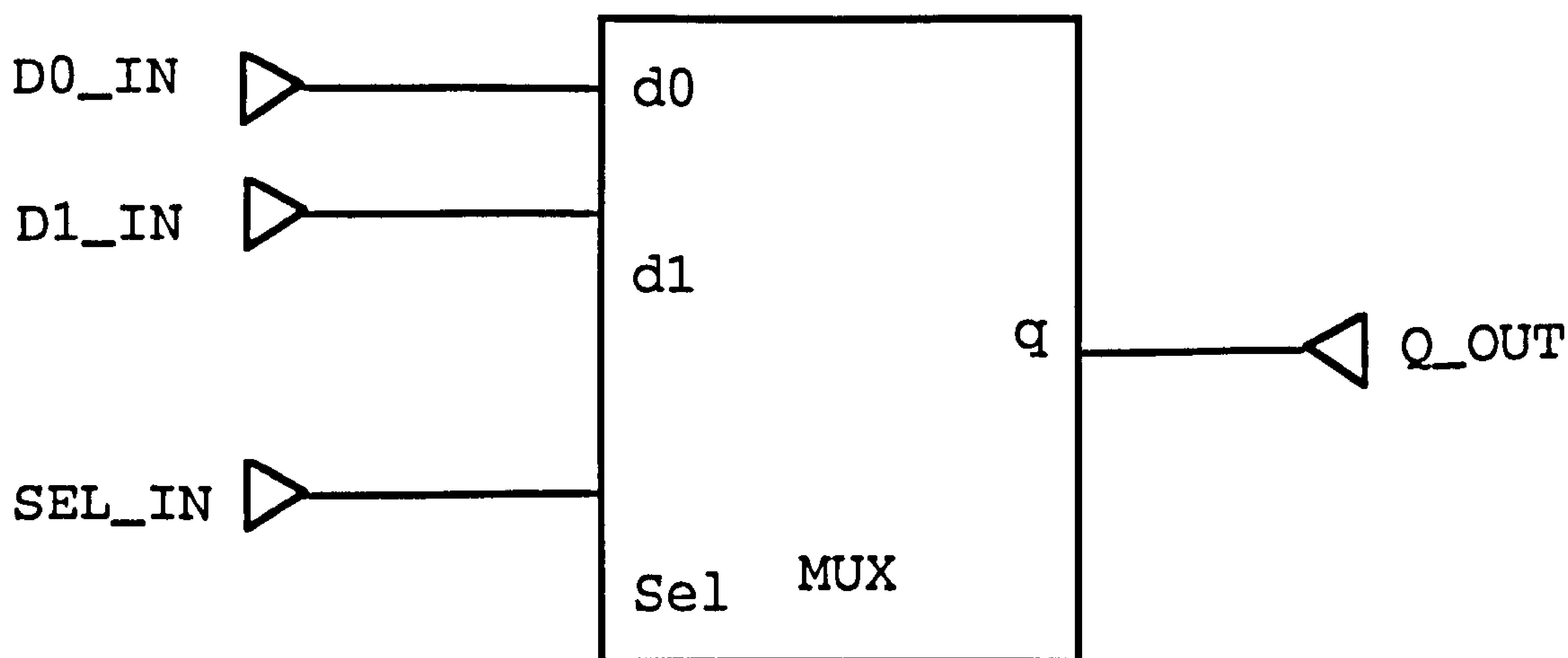



**Figure 4.3 Symbol Representation of Two-Input Multiplexer**

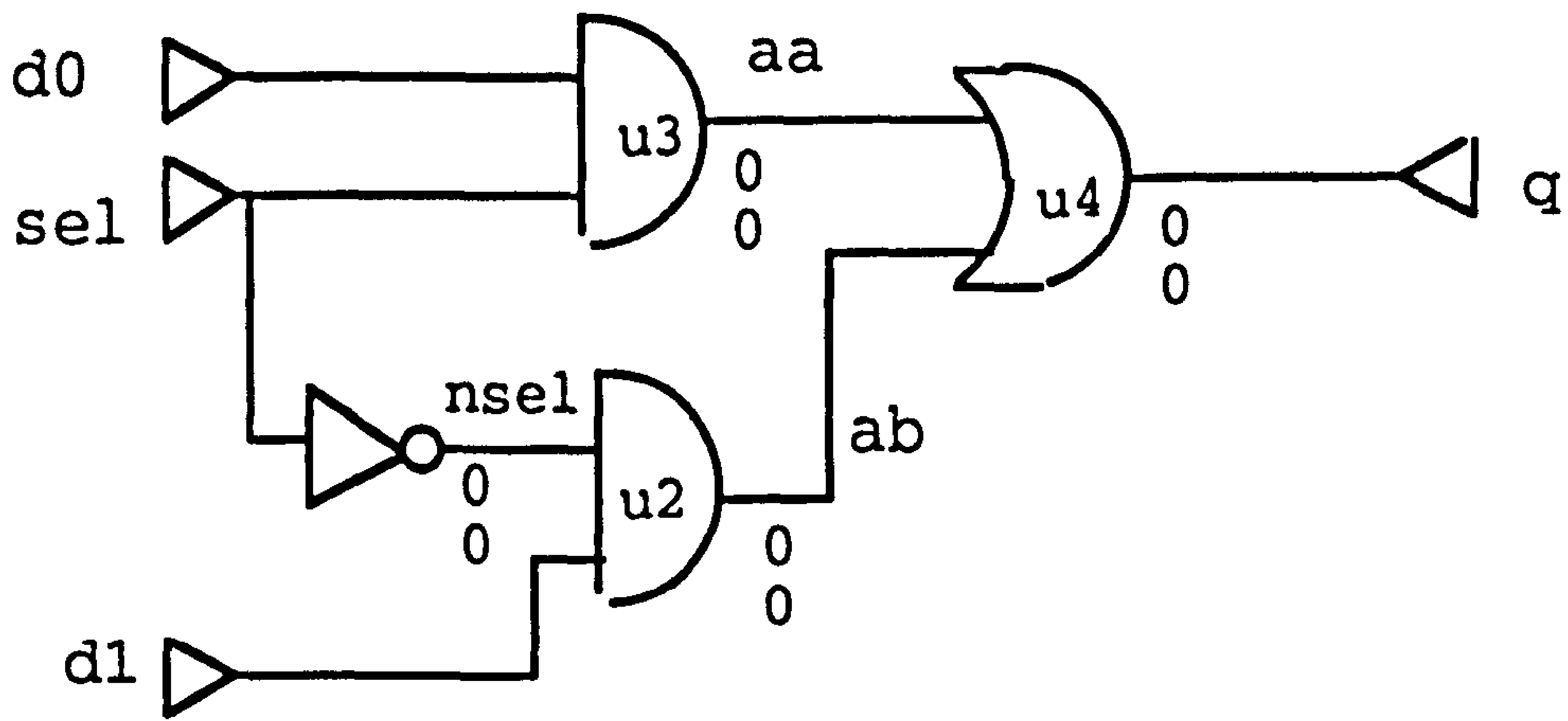



**Figure 4.4 Schematic Representation of Two-Input Multiplexer**

Figure 4.5 shows a System-1076 structural description of the two-input multiplexer. The System-1076 code contains comments that are set off with a double dash (--) symbol. Any text appearing between the double dash and the end of a line is ignored by the compiler. (See lines 1, 3, 6, 8, 18 20 through 22, and 25 in figure 4.5) Descriptive comments make the code easier to read.

```
1     -- entity declaration
2     ENTITY Mux IS
3     PORT (d0,d1,sel:IN Bit; q: OUT bit); --port  clause
4     END mux;
5
6     --  architecture body
7     ARCHITECTURE structure OF mux IS
8         COMPONENT and2 -- architecture declarative part
9         PORT (a, b: IN  bit;  z: OUT bit);
10        END COMPONENT;
11        COMPONENT  or2
12        PORT (a, b: IN bit;  z: OUT bit);
13        END COMPONENT;
14        COMPONENT  inv
15        PORT  (i: IN bit;  z: OUT bit);
16        END COMPONENT;
17
18        SIGNAL  aa, ab, nsel : bit; --signal declaration
19
20        FOR ul : inv   USE ENTITY invrt (behav);
                                           -- configuration
21        FOR u2,u3 : and2  USE ENTITY and_gt (dflw);
                                           -- specifications
22        FOR u4  :or2   USE ENTITY or_gt(archl);
23
```

```
24   BEGIN
25       u1:inv        PORT MAP (sel, nsel);
                                    -- architecture statement
26       u2:and2       PORT MAP (nsel,d1,ab);
27       u3:and2       PORT MAP (d0, sel,aa);
28       u4:or2        PORT MAP (aa, ab, q);
29   END structure;
```

**Figure 4.5: Code Example of Structural Description for a Multiplexer**

The two-input MUX represented by figure 4.5 is a basic design unit. The entity declaration at the top of figure 4.5 (lines 2 through 4) defines the interface between the entity and the environment outside of the entity.

This entity declaration contains a port clause that provides input channels (signals d0, d1 and sel in figure 4.5 line 3) and an output channel (signal q in figure 4.5, line 3). The signals are of a pre-defined type called bit which is declared elsewhere to describe all possible values (0 or 1) for each signal.

The architecture body (lines 7 through 29) describes the relationships between the entity inputs and outputs in a structural way.

The various components (and2, or2, and inv) that form the mux entity in figure 4.5 are declared in the architecture declarative part (lines 9 through 16). Signals (aa,ab, and nsel) are also declared in the architecture body (line 18) to represent the output of the two AND gates (u2 and u3) and the inverter (u1).

The configuration specifications in lines 20 through 22 bind each component instance to a specific entity which describes how each component operates.

As an example, the component u1 used in line 25 is bound to an architecture called behav for an entity called inverter.

The architecture statement part (lines 25 through 28) describes the connections between the components within the entity. It is in this part that the declared components are instantiated.

### 4.8.2 BEHAVIORAL DESCRIPTION

The following identifies some of the major language constructs found in a behavioural description using the previous MUX example and a four-bit example. Structural description method can now be compared with the behavioural description method described in this section.

A System-1076 behavioural description represents the function of a design in terms of circuit and signal response to various stimulus.

In the design shown in figures 4.5 through 4.9, the behaviour of the MUX was determined by the connections between the inverter, and AND gates, and the OR gate. The function of these gates is generally understood.

In a more complex design, the components U1 through U4 in figure 4.5 could represent entities that have complicated functions such as a central processing unit or a bus controller. When function and not structure is most important, each component can employ a corresponding behavioural description.

Figure 4.6 shows the System-1076 code which defines the MUX behaviour.

The behavioural description in figure 4.6 and the structural description in figure 4.5 both contain an entity declaration and an architecture body. In practice you would not have both the behavioural and structural architecture body shown in figures 4.5 and 4.6 in one source file. The designer can first write the entity declaration in one design file, the behavioural architecture in another design file and the structural architecture in still another design file. In an actual design after the entity declaration is written and compiled, one might write a behavioural architecture next, to allow testing of the overall circuit functions.

```
1     -- entity declaration
2     ENTITY  Mux  IS
3     PORT  (d0, d1, sel: IN Bit; q: OUT  bit); --port clause
4     END mux;
5
6     --  architecture body
7     ARCHITECTURE  behavioural  OF  mux  IS
8     BEGIN
9         f1: -- process statement
10        PROCESS  (d0, d1, sel) -- sensitivity list
11        BEGIN
12           IF sel = '0'  THEN  -- process statement part
13                q <= d1;
14           ELSE
15                q <= d0;
16            END  IF;
17        END PROCESS f1;
18    END behavioural;
```

**Figure 4.6 Code Example of Behavioural Description for a Multiplexer**

A behavioural description model is also useful to stimulate inputs of other System-1076 models during simulation.

The major difference between the structural and behavioural descriptions of the MUX is that the architecture body in figure 4.6 contains a process statement.

The process statement describes a single, independent process that defines the behaviour of a hardware design or design portion. The basic format process statement is as follows:

```
process statement ............  label:
      process (sensitivity_list)
      process_declarative_part
      begin
         process_statement_part
      end process label;
```

The process statement in figure 4.6 begins with the process label f1 followed by a colon (line 9). The process label is optional but is useful to help differentiate this process from other processes in a larger design.

Following the reserved word process is an optional sensitivity list (located between the parenthesis). The sensitivity list in figure 4.6 (line 10) consists of the signal names d0, d1, and sel. During simulation, whenever a signal in the sensitivity list changes state, that process is executed.

In the MUX example, whenever, d0, d1, or sel change state, process f1 is executed and the state of the output signal is changed accordingly. Each process in a System-1076 design description is executed once during initialisation of the System-1076 hardware model.

The heart of the process statement in figure 4.6 is the 'if' statement contained in the process statement part. The basic format of an 'if' statement is as follows:

```
if statement ..............    if  condition then
                 sequence_of_statements
           else if  condition then
                      sequence_of_statements
           else
                      sequence_of_statements
           end if ;
```

### 4.8.3 DATA FLOW DESCRIPTION

The following identifies some of the major language constructs found in a data-flow description using the previous MUX example. The VHDL data-flow description and a register-transfer language description are similar in that they describe the function of a design by defining the flow of information from one input or register to another register or output.

The data-flow and behavioural descriptions are similar in that both use a process to describe the functionality of a circuit- see figure 4.7. A behavioural description explicitly calls a single, independent process with process statement by using one or more of the following concurrent statements for each implied process:

- Block statement
- Concurrent procedure call
- Concurrent assertion statement
- Concurrent signal assignment

In addition to these language constructs, the Block statement and Component Instantiation statement are also concurrent statements but are not found in a data-flow description. Concurrent statements define interconnected processes and blocks that together describe the overall behaviour or structure of a design.

A concurrent statement executes asynchronously with respect to other concurrent statements.

```
1     -- entity declaration
2     ENTITY  Mux  IS
3     PORT  (d0, d1, sel: IN Bit; q: OUT  bit); --port clause
4     END  mux;
5
6     --  architecture body
7     ARCHITECTURE  data_flow  OF  mux  IS
8     BEGIN
9      cs1 :     -- concurrent sig assignment statement
10     q <= d1 WHEN sel = "0" ELSE  -- conditional sig. ass.
11        d0 WHEN sel = "1";
12    END data_flow
```

**FIGURE 4.7 Code Example of Data-Flow Description for a Multiplexer**

## 4.9 BEHAVIORAL MODELLING OF 4 BIT MULTIPLIER

In this section a behavioural model of a 4 bit serial multiplier will be designed. This circuit will be used as an example to demonstrate this modelling style of VHDL.

The design consists of series of successive shift and add routines, which work in basically the same way as multiplying two numbers on paper. The main difference is that on paper the shifting for each number is done first and then the whole lot added together to determine the result. This is different to the hardware design which keeps account of the last shift sum and adds it to the subtotal. In this way only one accumulation register is needed, so saving on gates and resulting in smaller die size at less expense. The other difference, mainly related to binary multiplication, is that no calculation is performed to the multiplicand if the next bit to be multiplied is a zero.

The multiplication process implemented contains three registers as can be seen from the top level multiplier schematic.

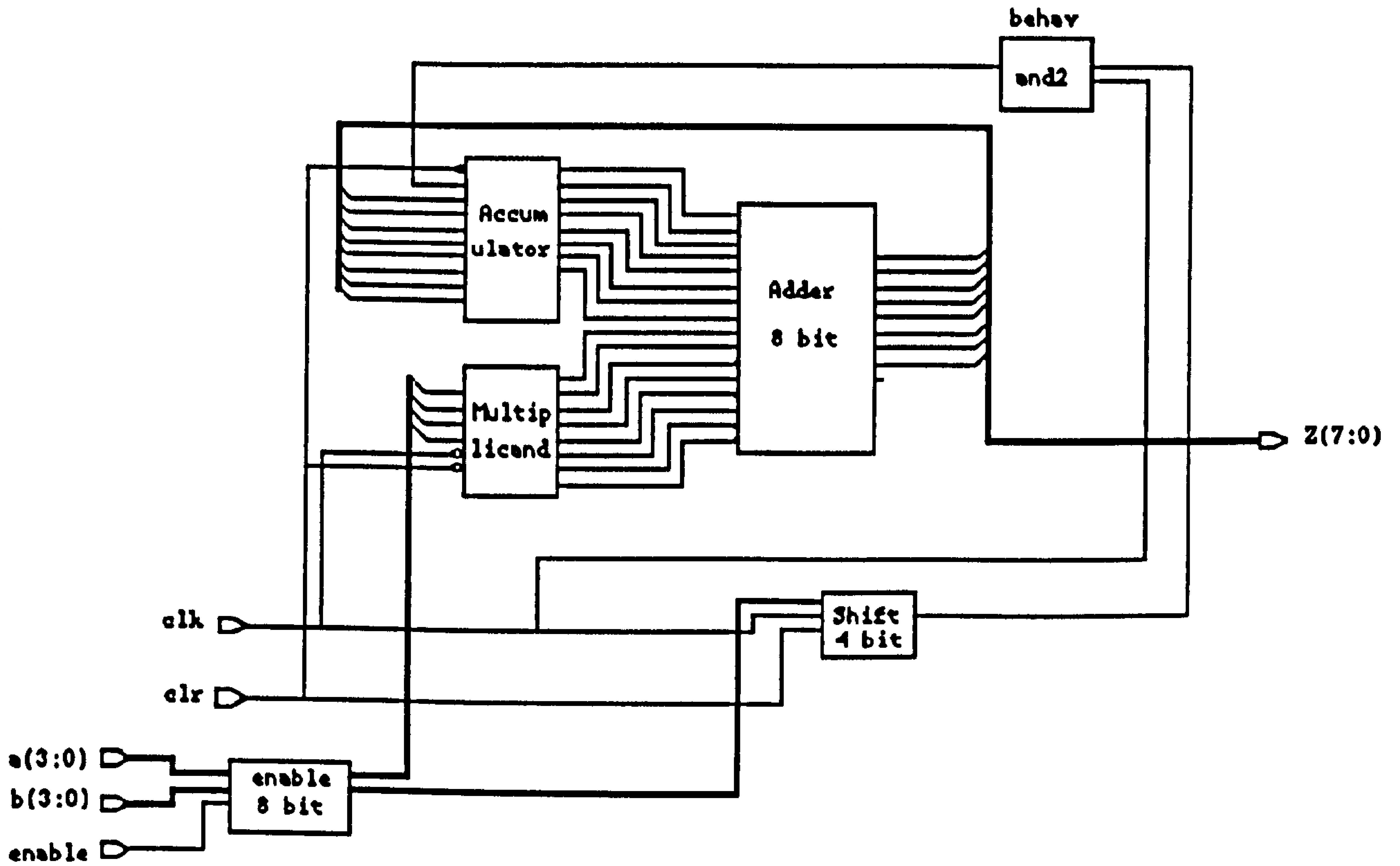



**Figure 4.8: Structural Description of 4 bit Multiplier**

One register (the four 4 bit multiplier register) is used to store one of the two four bit numbers to be multiplied together, the other 4 bit number is loaded into the eight bit multiplicand register to be shifted through. Loading is accomplished by previously resetting the registers to zero via the parallel clear and loading the values on to the preset. Any 'one' present in the word set, the appropriate cell will change to a high state.

The output is achieved by successive shifting and add instructions via the 8 bit accumulator. Note that if a zero is multiplied the accumulator latch is inhibited and the multiplicand is shifted left one cycle.

### 4.9.1 VHDL BEHAVIOURAL MODEL OF MULTIPLIER

This section describes the behavioural description of a 4 bit multiplier in VHDL.

```
ENTITY mult1 IS
  PORT (a :IN vlbit_1d(0 TO 3);
        b :IN vlbit_1d(0 TO 3);
        q :OUT vlbit_1d(0 TO 7);
        clock,start :IN vlbit);
END mult1;

ARCHITECTURE behav OF mult1 IS

BEGIN

calculate :PROCESS (clock)

VARIABLE cycle : INTEGER := 0;

BEGIN
  IF (clock='1') AND (start='1') THEN
    IF cycle=4 THEN
      q <= mulum(a,b);
      cycle := 0;
    ELSE
      cycle := cycle + 1;
    END IF;
  END IF;

END PROCESS calculate;

END behav;
```

**Figure 4.9: Behavioural Description of Multiplier**

### 4.9.2 SIMULATION AND TEST VECTORS

Test Vectors are used to verify the functionality of the device after manufacture. The test vectors generated for the multiplier are shown in the simulation wave forms in the appendix. The inputs (in decimal) loaded to test the device are as follows: 5 * 3 = 15

This resulted in a ripple through output of '0F' which tested that the first four pins could be driven high, whilst the next four bits remained low. Other tests that were used are:

15 * 15 = 225 which gives the maximum count available.
0 * 8 = 0 which gives low level output.

This does not provide a full test of all circuit nodes with all possible combinations, but proves the viability of the device to perform the mathematical computations required.

## 4.10 CONCLUSIONS

VHDL was chosen as the Behavioural Description Language for modelling the Boundary Scan Architecture in this project for several reasons. It is a well documented standard, it is gaining popular acceptance, it supports both abstraction hierarchy and design hierarchy (with its structural and procedural constructs) and it is not tied to any one vendor's design system.

The different implementations of the multiplier have demonstrated the design approaches that are becoming available in todays market. VHDL thus offers a number of benefits over other hardware description languages. These include its availability as a public standard, its ability to support different design methodologies and design technologies, its independence of both technology and process, and its capability to support a wide range of hardware descriptions of a digital system, from a behavioural level to a gate level. These benefits have therefore affirmed the decision to use VHDL in this project, to model the Boundary Scan Test Architecture.

**REFERENCES**


[Wax1] R. Waxman, "The VHSIC Hardware Description Language- A Glimpse of the Future," IEEE Design and Test of Computers (April 1986)

[Wax2] R.Waxman, "Hardware Design Languages for Computer Design and Test", IEEE Computer Volume 19 (April 1986)

[Coel1] David R. Coelho. The VHDL Handbook, Kluwer Academic Publishers.

[Ceeda91] S Medhat, J McGinty, N Sheridan. The Use of VHDL in a VLSI Matrix Manipulation Processor Design. CEEDA 91, pp 320-324 Bournemouth, UK March 1991.

[stan 1076] IEEE Press, IEEE Standard VHDL Reference Manual, IEEE 1076 - 1987, Montvale, NJ, 1986.

# CHAPTER 5

# THE 1EEE 1149.1 BOUNDARY SCAN ARCHITECTURE

## 5.0 INTRODUCTION

Up to now, the BIST test methods and general test principles have been described in the previous chapters. This chapter will describe the operation of the main components of the IEEE-JTAG standard in a structural way. Initially, the full architecture will be designed, simulated and tested without an application logic using the Mentor Graphics Computer Aided Engineering system. The Boundary Scan Architecture will then be added to a 2-bit adder design example (An Application Logic ASIC) to give an insight to the test harness operation as described in Appendix 5.

Figure 5.1 shows a block diagram of the general JTAG circuit used on a chip. It includes all the signals needed to integrate an on-chip internal scan.

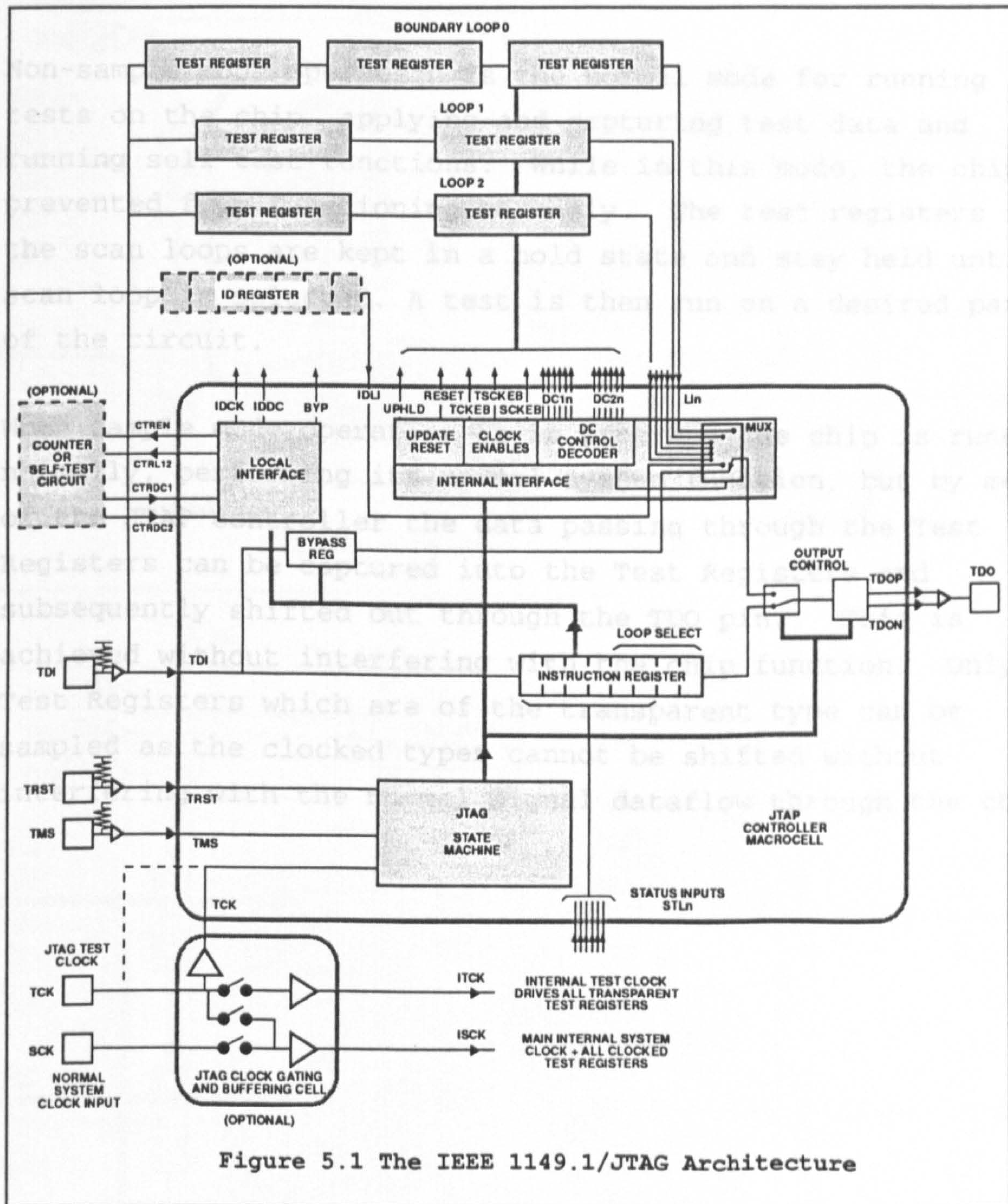



**Figure 5.1 The IEEE 1149.1/JTAG Architecture**

## 5.1 JTAG BOUNDARY SCAN ARCHITECTURE TEST MODES

There are two main modes of test operation that are supported by JTAG - 'Sample' mode and 'Non-sample' mode, see fig 5.2 a,b.

Non-sample mode operation is the normal mode for running tests on the chip, applying and capturing test data and running self-test functions. While in this mode, the chip is prevented from functioning normally. The test registers in the scan loops are kept in a hold state and stay held until a scan loop is selected. A test is then run on a desired part of the circuit.

When Sample mode operation is in progress the chip is running normally, performing its normal system function, but by means of the JTAP controller the data passing through the Test Registers can be captured into the Test Registers and subsequently shifted out through the TDO pin. This is achieved without interfering with the chip function. Only Test Registers which are of the transparent type can be sampled as the clocked types cannot be shifted without interfering with the normal signal dataflow through the chip.

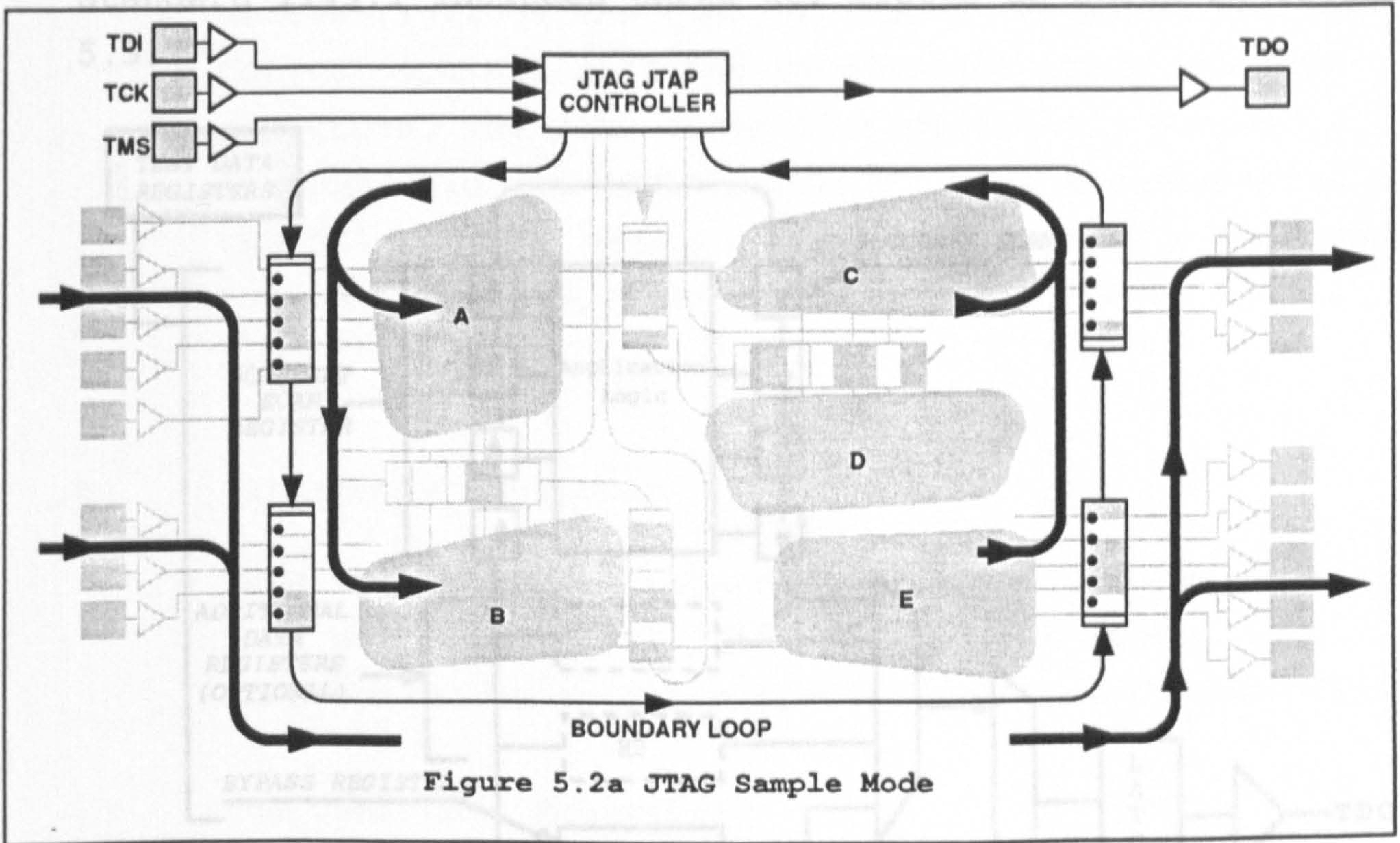


Figure 5.2a JTAG Sample Mode

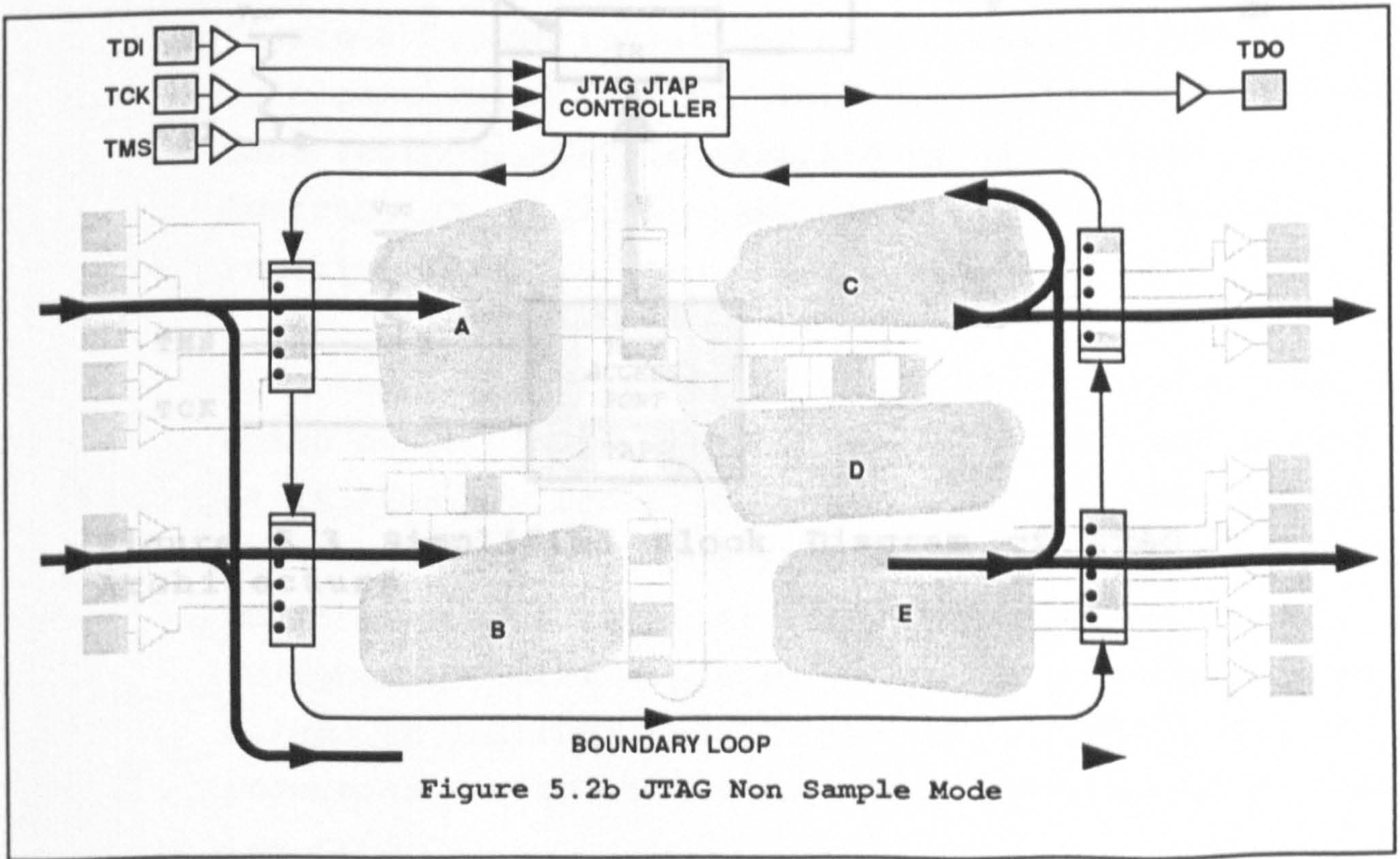


Figure 5.2b JTAG Non Sample Mode

## 5.2 JTAG BOUNDARY SCAN ARCHITECTURE MAIN COMPONENTS

The top-level schematic of the test logic defined by IEEE Standard 1149.1 includes three key blocks as shown in figure 5.3:

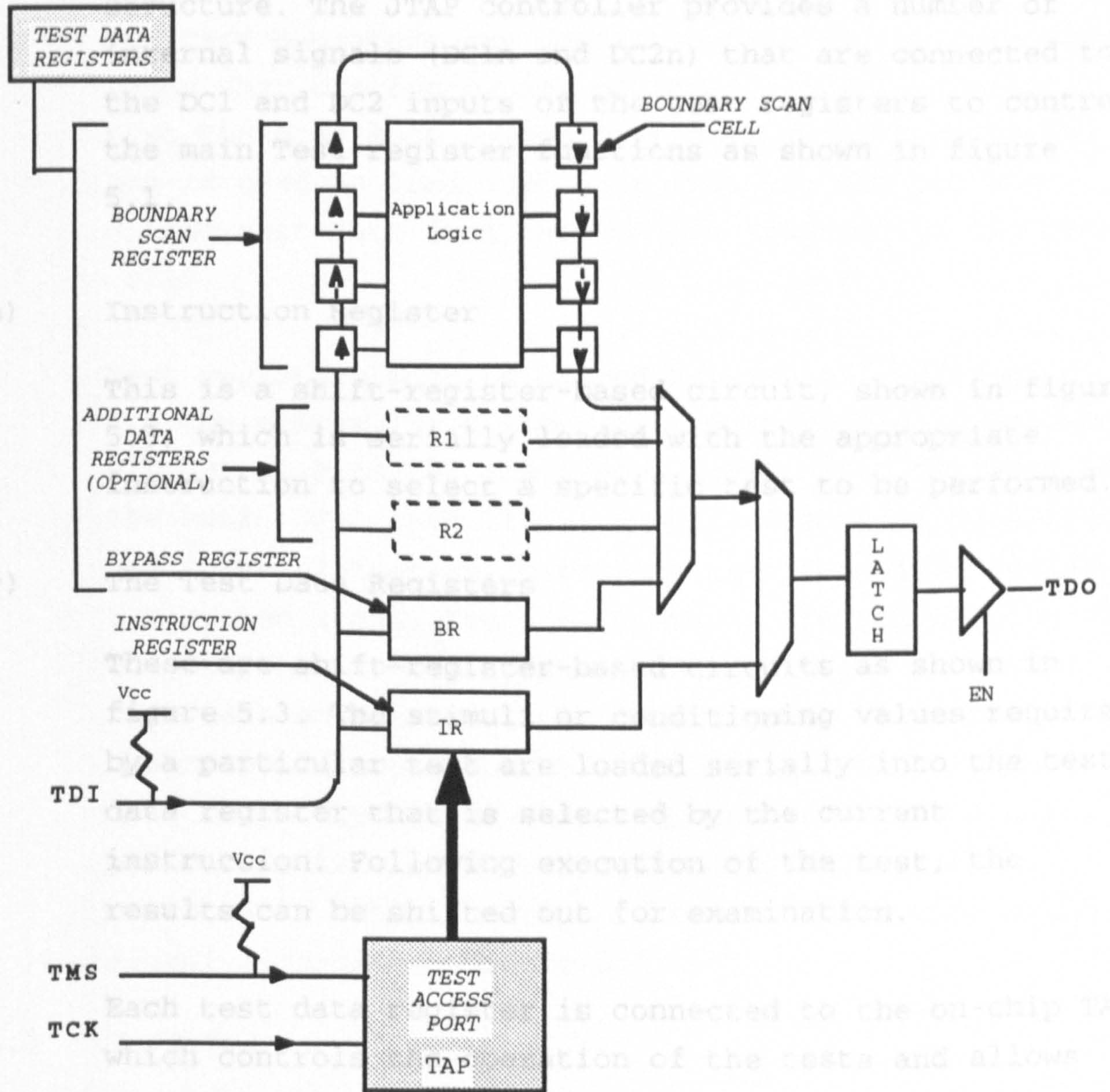



**Figure 5.3 Simplified Block Diagram of JTAG Architecture**

a) The TAP Controller

This circuit responds to the control sequences supplied through the Test Access Port and generates the necessary clock and control signals required for the correct operation of the application logic and test structure. The JTAP controller provides a number of internal signals (DC1n and DC2n) that are connected to the DC1 and DC2 inputs of the test registers to control the main Test register functions as shown in figure 5.1.

b) Instruction Register

This is a shift-register-based circuit, shown in figure 5.3, which is serially loaded with the appropriate instruction to select a specific test to be performed.

c) The Test Data Registers

These are shift-register-based circuits as shown in figure 5.3. The stimuli or conditioning values required by a particular test are loaded serially into the test data register that is selected by the current instruction. Following execution of the test, the results can be shifted out for examination.

Each test data register is connected to the on-chip TAP which controls the operation of the tests and allows serial loading and unloading of instructions and test data to take place. The role of the embedded TAP within an integrated circuit is directly analogous to the "diagnostic" socket provided on many automobiles. It allows an external test processor to control and communicate with the various test features built into the product.

In addition, the test data registers can be connected to the various modules of the application logic within the chip or to the pins that are connected to the application logic, to allow tests to be performed on the application logic.

The Test registers are connected together to form scan loops that are fed by serial input pin TDI. During test the scan loop is configured as a shift register and is used to load test data from the TDI pin and unload test data from the TDO pin (routed via the JTAP cell).

Test registers must be placed between all input pins and the system logic. They also must be placed between all output pins and the system logic. This conforms to the basic JTAG/IEEE 1149-1 specifications of self-test and internal test which require test registers inside the system logic. Test registers on the output pins of the chip have extra update holding latches to prevent test data appearing at the chip outputs. This prevents the random signals on the output Test registers from upsetting other chips that are connected to the outputs pins. This is a requirement for JTAG/IEEE 1149-1 conformance. This function is controlled by the UPHLD signal as shown in figure 5.1.

## 5.3 THE TAP

The TAP contains four or optionally, five pins [MAUND 87]. These are:

- *The test clock input (TCK):* This is an independent clock of the chip system clock(s), so that test operations can be synchronised between the various chips on a printed wiring board.

Both the rising and falling edges of the clock are significant: the rising edge is used to load signals applied at the TAP input pins TMS (test mode select) and TDI (test data input), while the falling edge is used to clock signals out through the TAP test data output (TDO) pin (figure 5.4). As will be discussed later, the boundary-scan register as defined by the standard is controlled such that data is loaded from the system input pins on the rising edge of TCK while data is driven through system output pins on the falling edge.

- *The Test Mode Select Input (TMS)*: The operation of the test logic is controlled by the sequence of 1s and 0s applied at this input, with the signal value typically changing on the falling edge of TCK. This sequence is fed to the TAP controller (which samples the value at TMS on each rising edge of TCK) by the other test logic blocks. TMS is either equipped with a pull-up resistor or otherwise is designed such that when it is not driven from an external source, the test logic perceives a logic 1.

- *The Test Data Input (TDI)*: Data applied at this serial input are fed either into the instruction register or into a test data register, depending on the sequence previously applied at TMS.

  Typically, the signal applied at TDI will be controlled to change state following the falling edge of TCK, while the registers shift in the value received on the rising edge.

  Like TMS, TDI is either equipped with a pull-up resistor or otherwise is designed such that, when it is not driven from an external source, the test logic perceives a logic 1.

- *The Test Data Output (TDO)* : This serial output from the test logic is fed either from the instruction register or from a test data register depending on the sequence previously applied at the TMS. During shifting, data applied at TDI will appear at TDO after a number of cycles of TCK determined by the length of the register included in the serial path. The signal driven through TDO changes state following the falling edge of TCK. When data are not being shifted through the chip, TDO is set to an inactive drive state (eg., high-impedance).

- *The Optional Test Reset Input (TRST)* : The need to be able to initialise a circuit to a known starting state (the reset state) is crucial in testing . As will be discussed later, the TAP controller is designed so that this state can be quickly entered under control of TCK and TMS.

  The standard also requires that the test logic can be initialised on power-up independently of TCK and TMS. This can be achieved either by building features into the test logic itself (eg. a power-up reset circuit) or by adding the optional TRST signal to the TAP. Application of a 0 at TRST asynchronously forces the test logic into its reset state. Note that, in this state, the test logic cannot interfere with the operation of the on-chip system logic, so TRST can also be viewed as a "test mode enable" input [BEEN 85].

  By loading the signals applied to the test logic through chip input pins (eg., through TMS and TDI) on the rising edge of TCK, while using the falling edge to clock signals out through chip output pins (such as TDO),the operation of the IEEE Std 1149.1 test logic can be made race-free.

For example, when chips compatible with the standard are serially connected, data is applied to TDO by the first chip on half cycle of TCK prior to the time when they are loaded from the TDI input of the second. This allows time to account for delays in the serial path, skew between the clock fed to the neighbouring ICs, and other factors.

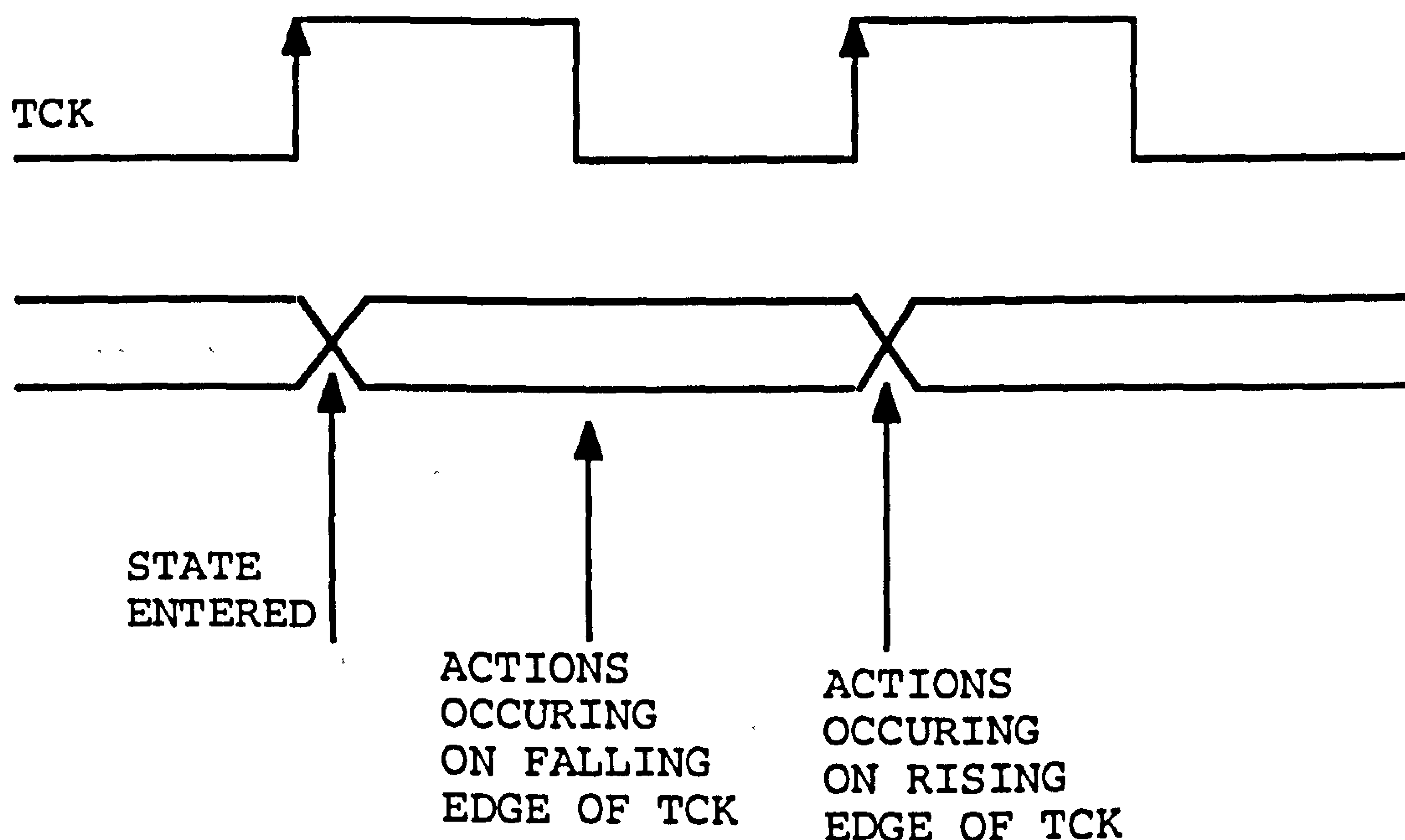



**Figure 5.4 Control Pipelinning**

Since TDO is set to an inactive drive state when no data is being shifted, the TAPs of individual chips can, if required, be connected to give parallel serial path at the board level. In such cases, a different TMS signal is required for each serial path. These signals should be controlled such that no two paths attempt to shift data simultaneously [WHET 88].

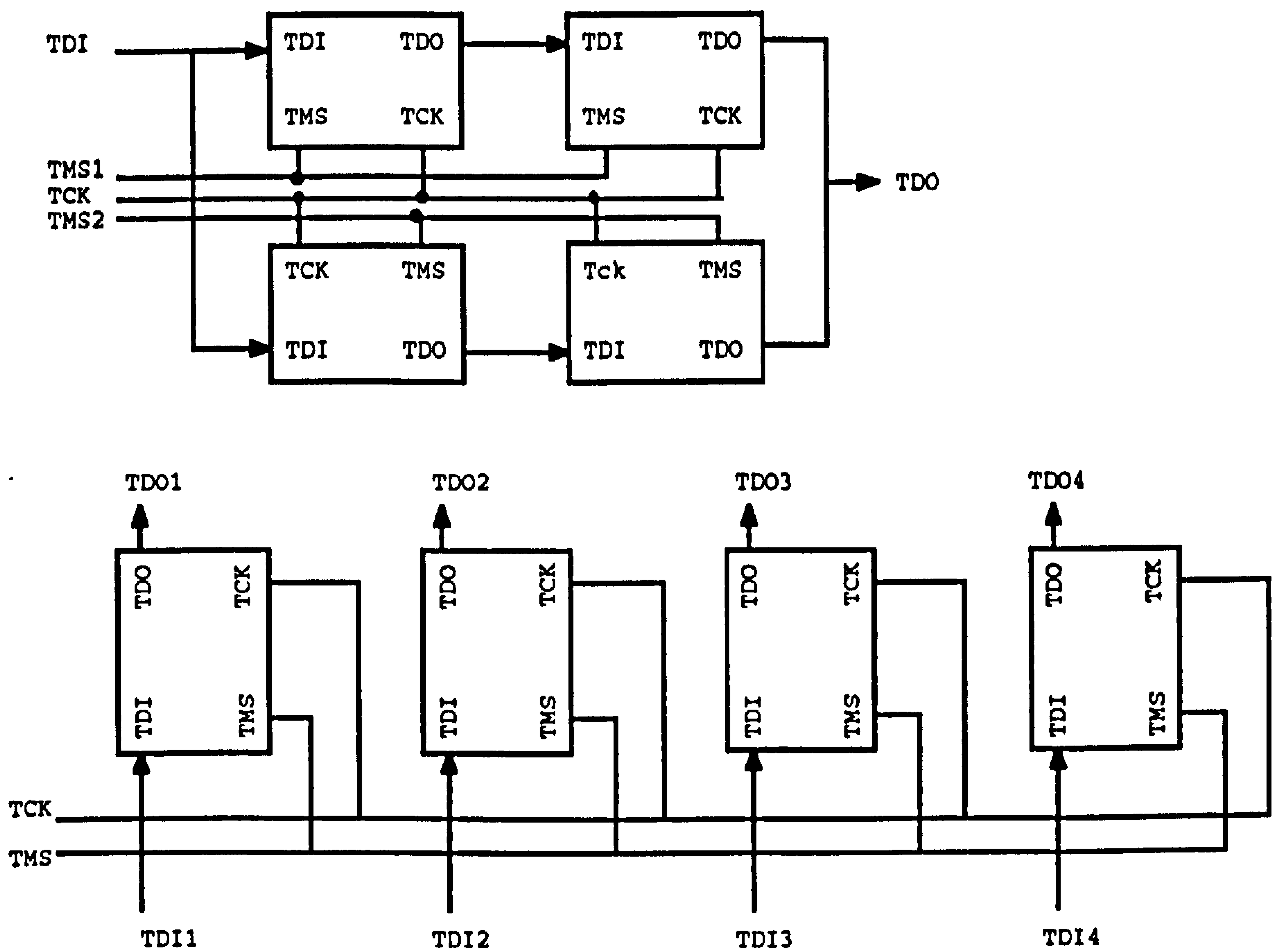



**Figure 5.5 Different Configurations of the Boundary Scan Register**

At the board level, the test signals can be controlled either by external automatic test equipment (ATE) or by an on-board bus-master chip. In the latter case, the bus-master chip might provide an interface between the interface defined by the IEEE Std 1149.1 TAP and some higher level test messaging system.

## 5.4 BOUNDARY SCAN

The boundary scan cell shown in figure 5.6 connects the external input pins to the internal functioning connections of the application logic (e.g. 2-bit adder). One cell for each independent connection is required.

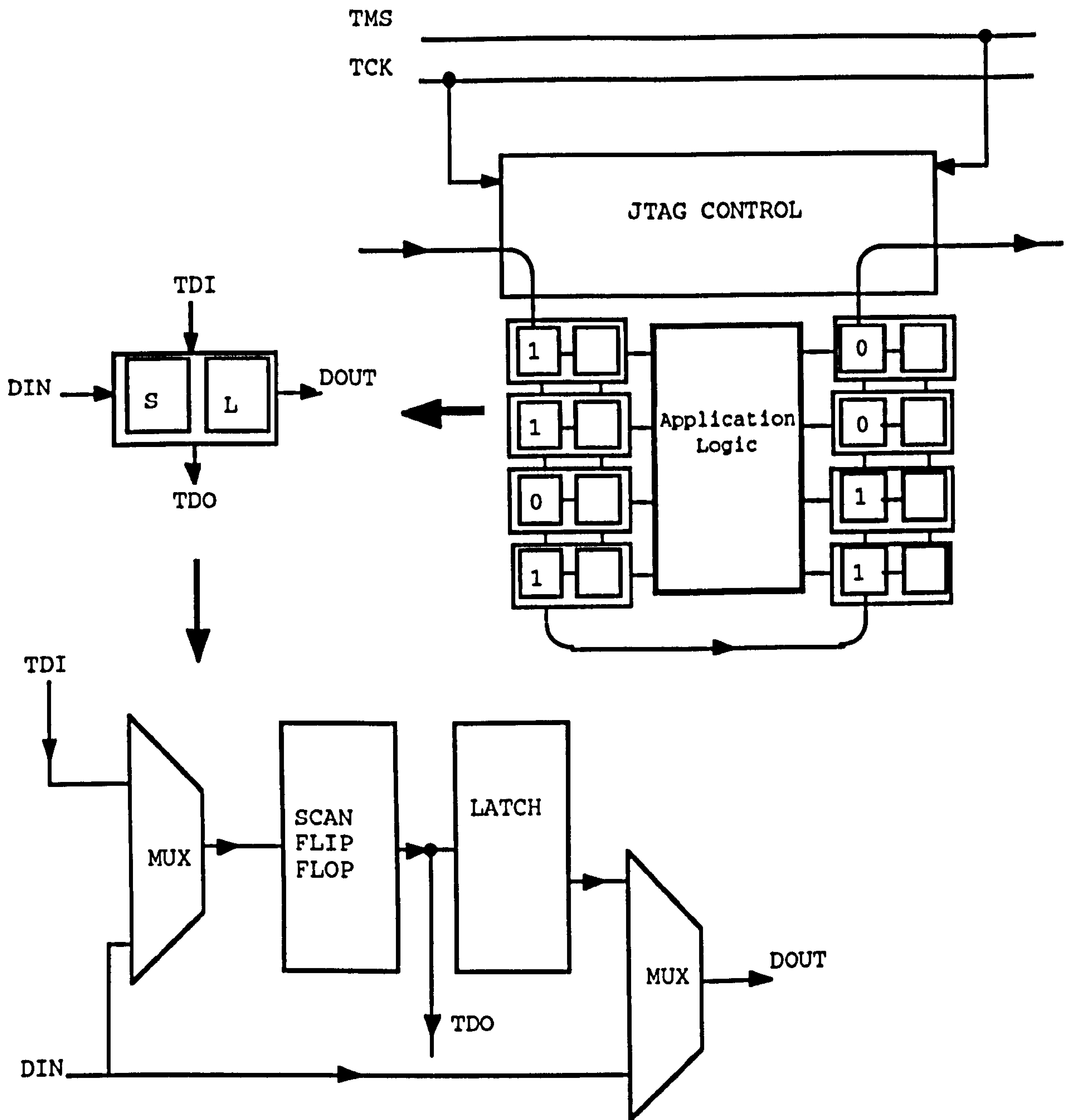



**Figure 5.6 Boundary Scan Details**

Boundary scan involves the inclusion of a controlled shift register which is connected from the input/output of the on-chip application logic to the device pins. The boundary scan cells have been designed so that they either accept serial input (test data or test instruction) or data from the standard input pins depending on the mode of operation.

These boundary cells act as 3-way latching switch which can either:

1. Let data pass straight through giving standard operation.
2. Isolate the external input, allowing data to be received from the serial input and passing it through to the application logic (the adder).
3. Sending serial data directly through the serial output bypassing the adder completely.

The cells connected on the output act in a similar way to those on the input, isolating the adder output connections from the external output pins when certain instructions are loaded [ICCD 87].

The three functions have been implemented to conform to the IEEE regulations of the standard. Other optional instructions can be added depending on the device to be tested and the complexity of the design.

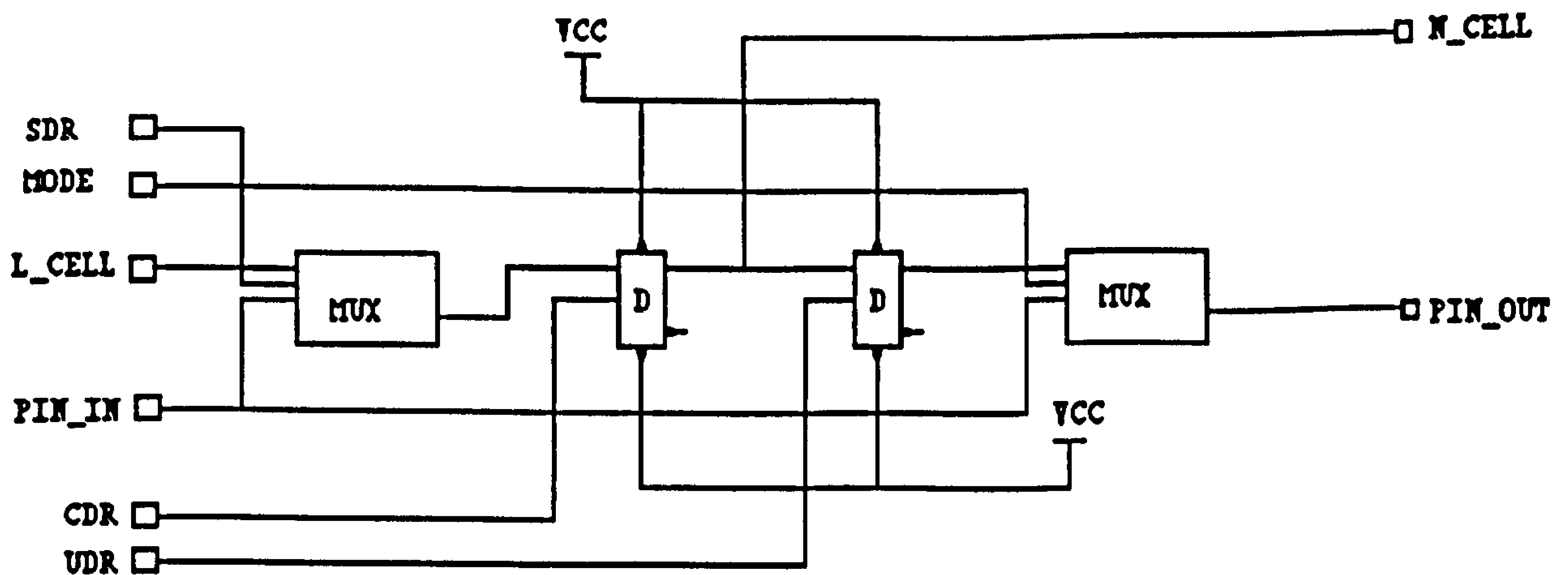



Figure 5.7-a Boundary Scan Cell (Input or Output)

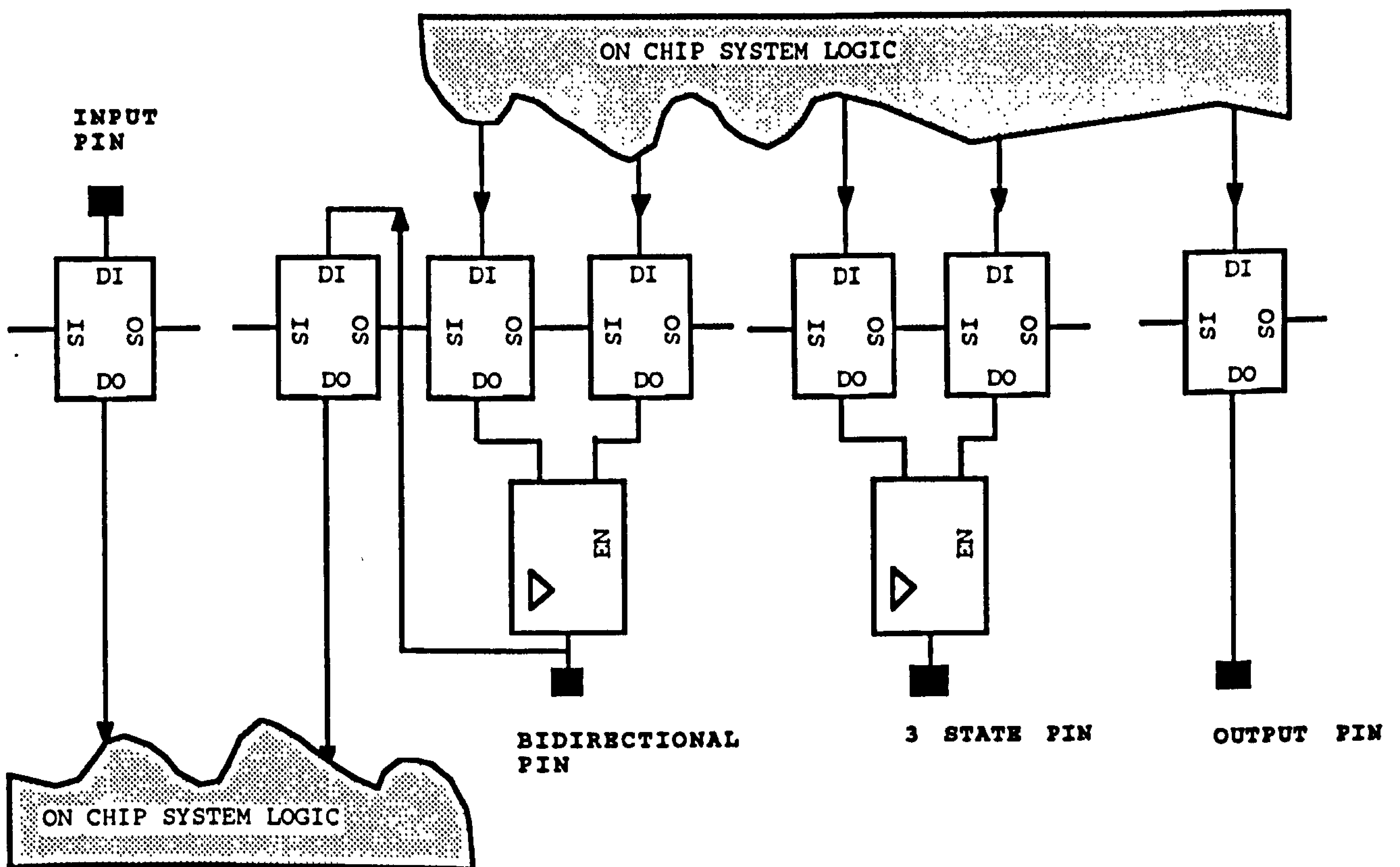



Figure 5.7-b Different Input/Output Types of BSC

### 5.4.1 BOUNDARY LOOP 0 RULES

The JTAG/IEEE 1149-1 spec defines that test registers are placed on the periphery of the chip and are connected together to form a boundary loop (loop 0).

In order for JTAG/IEEE 1149-1 spec conformance the following rules must be obeyed when connecting up the test registers on the boundary loop:-

a) All system input and output pins must pass through Test registers (including the system clock).

b) For tri-state and bi-directional pins, the tri-state enable connection must also pass into the test register. (Where tri-state and bi-directional buses are used internally in a design, it is recommended that the internal tri-state enable should be passed into the test registers).

c) Test registers on the output pins must be equipped with Update latches. Those output pins which are used for tri-state or bi-directional outputs can use a single data bit together with an update latch on the tri-state control line.

d) All Test registers connecting on the input and output pins must be connected together to form a scan loop, and should be connected to JTAP controller cell loop input L0(loop 0). The order of the test registers in the scan-path is not important.

e) For SAMPLE mode operation the test registers in boundary loop 0 must be made exclusively from transparent registers.

### 5.4.2 THE TEST REGISTER TEST MODES

In order to implement the different functions with the minimum of external control connections, each test register has two external control connections (DC1 and DC2) plus an update latch control (UPHLD). These are used to select which of the main modes in which the test register is to operate.

| DC1 | DC2 | Mode Name | Description |
|---|---|---|---|
| 0 | 0 | TEST | Performs Test selected by control bits |
| 0 | 1 | SHIFT | Serial data shifted through scan loop |
| 1 | 0 | RUN | Non-test mode, Data passes from D to F |
| 1 | 1 | HOLD | Signal data held on register |

Additionally there are two test control register bits that are shifted in series with the data bits. These control bits control which type of test sub-function is to be performed by the test register when the TEST mode is selected by DC1=0 and DC2=0 signals.

| DC1 | DC2 | TC1 | TC2 | Mode Name |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | TEST HOLD |
| 0 | 0 | 0 | 1 | TEST SIGNATURE-ANALYSE |
| 0 | 0 | 1 | 0 | TEST SLIDE |
| 0 | 0 | 1 | 1 | TEST PSEUDO-RAN GENERATE |

TC1 and TC2 are in series with the test register serial scan path, and are loaded during the shift operation.

The DC1 and DC2 signals are normally supplied from the JTAP controller.

#### 5.4.2.1 RUN

RUN mode is the state that the Test register is in during normal non-test chip operation, where data flows through the test register uninterrupted.  The RUN mode is also used during test to apply and capture data simple tests.

Data passes from the D databit input pins to the F output pins. If the databits used are the transparent type then data is propagated immediately to the output, while if clocked data bits are used then F changes on the +ve edge of the clock CLK driving the Test register. If clocked databits are used then the EN Test register input is used to hold the input if EN = 0. Fig 5.8.1 shows diagrammatically the Test register in RUN mode.

#### 5.4.2.2 HOLD DATA

Data is held on the test register. It is used either as an intermediate state whilst the JTAP controller is going through the state diagram in preparation for a test, or data is held stable on a Test register while other parts of the circuit is being shifted. Fig 5.8.2 shows the Test register in HOLD mode.

#### 5.4.2.3 SHIFT DATA

Data is shifted into the test register (on pin SI) and out through the last data bit, see fig 5.8.3. The data shifted in can be used to set up the data presented to the inputs of the circuit being tested while the data shifted out can be the results of the data test. At the same time, the test register control bits TC1 and TC2 are also shifted and set up.

#### 5.4.2.4 TEST PSEUDO-RANDOM GENERATE

The test register is configured into a linear-feedback shift register which generates a maximal-length pseudo-random sequence at the test register data outputs. This sequence is presented to the inputs of the block of circuit being tested, and may be used to exhaustively test the circuit. See fig 5.8.4.

#### 5.4.2.5 TEST SIGNATURE-ANALYSE

The test register is configured into a linear-feedback shift register and the data inputs are EXclusive ORed into the register, causing a signature to be generated at the end of the test see fig 5.8.5. It used to build up a signature from circuit outputs from circuit whilst the inputs are being tested with TEST PSEUDO-RANDOM-GENERATE or TEST SLIDE test modes.

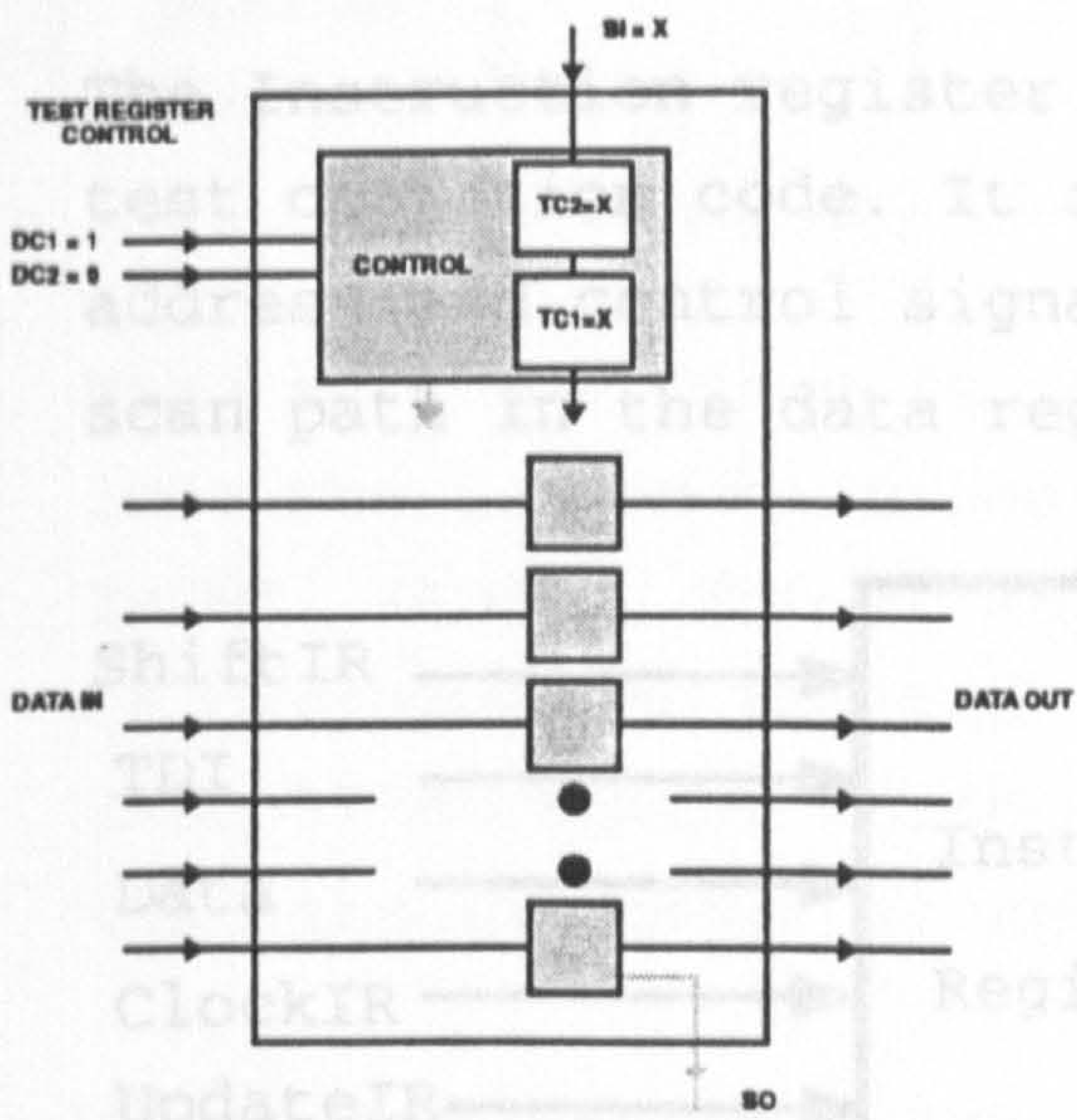
#### 5.4.2.6 TEST SLIDE

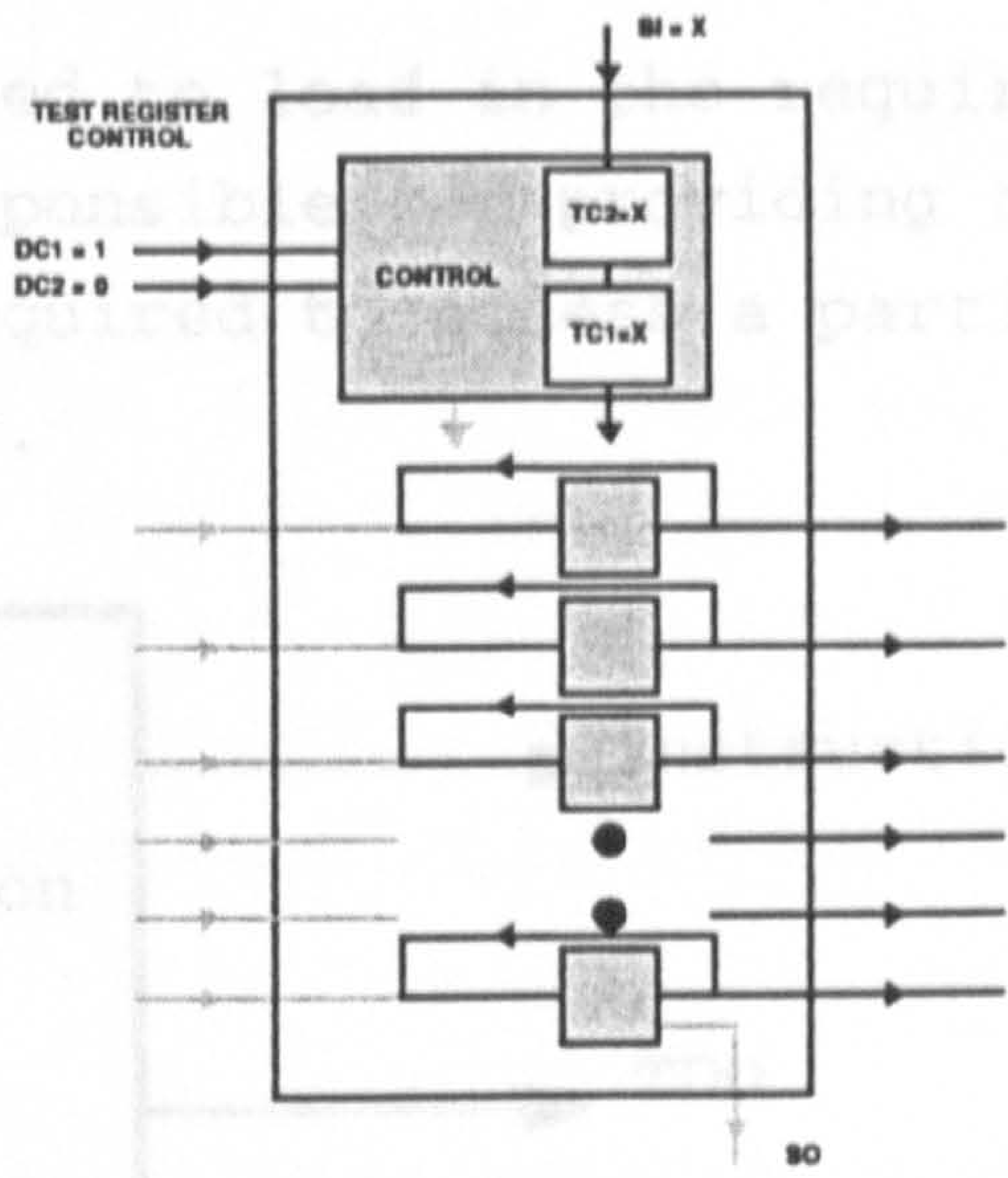
The test register data bits are configured into a shift register whose last bit output is fed back to the first bit input, thus recirculating the data as shown in fig 5.8.6. This test configuration is used to 'walk' patterns along the circuit inputs. An output signature can then be collected with the output test register set to TEST SIGNATURE-ANALYSE.

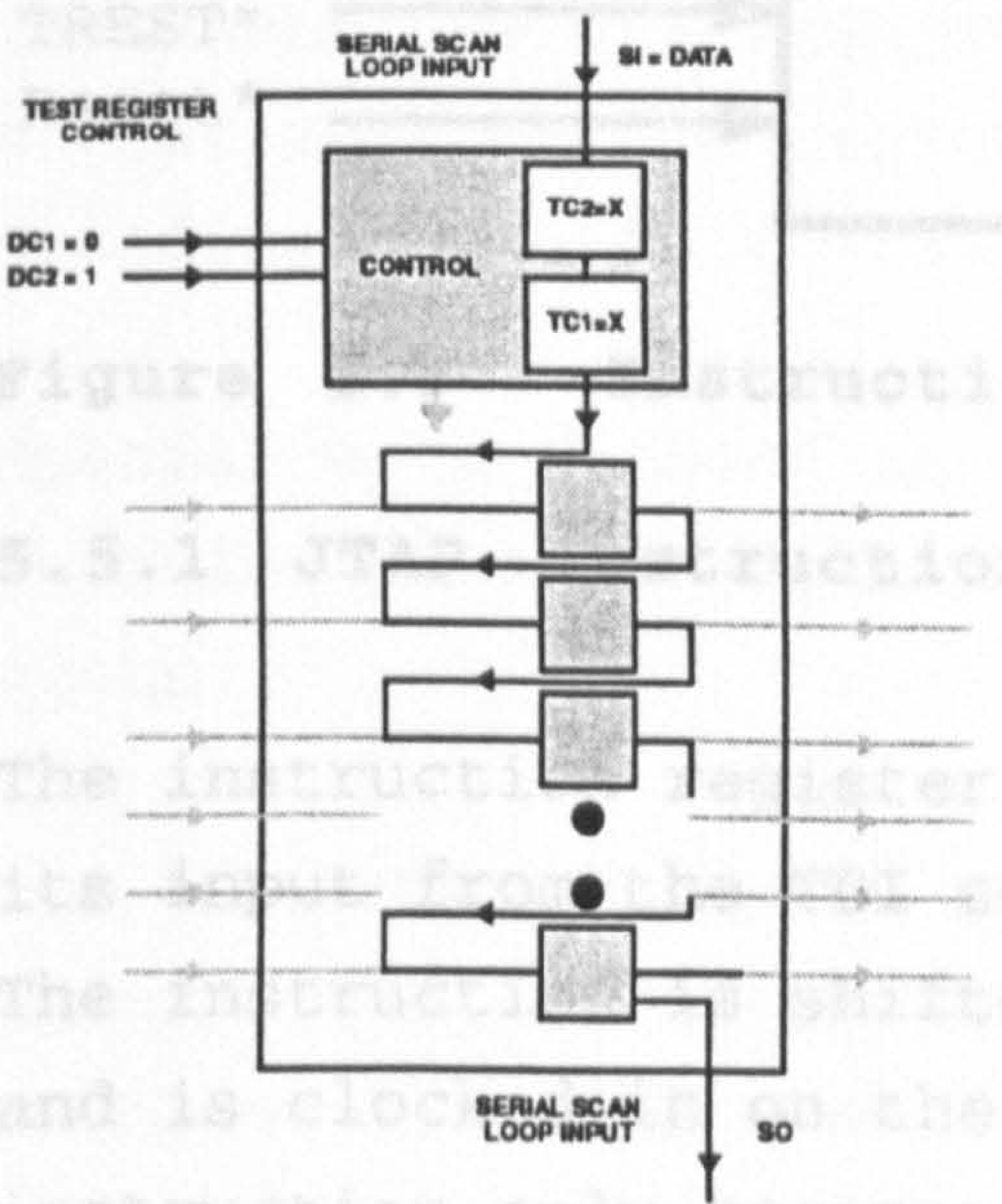
#### 5.4.2.7 TEST HOLD

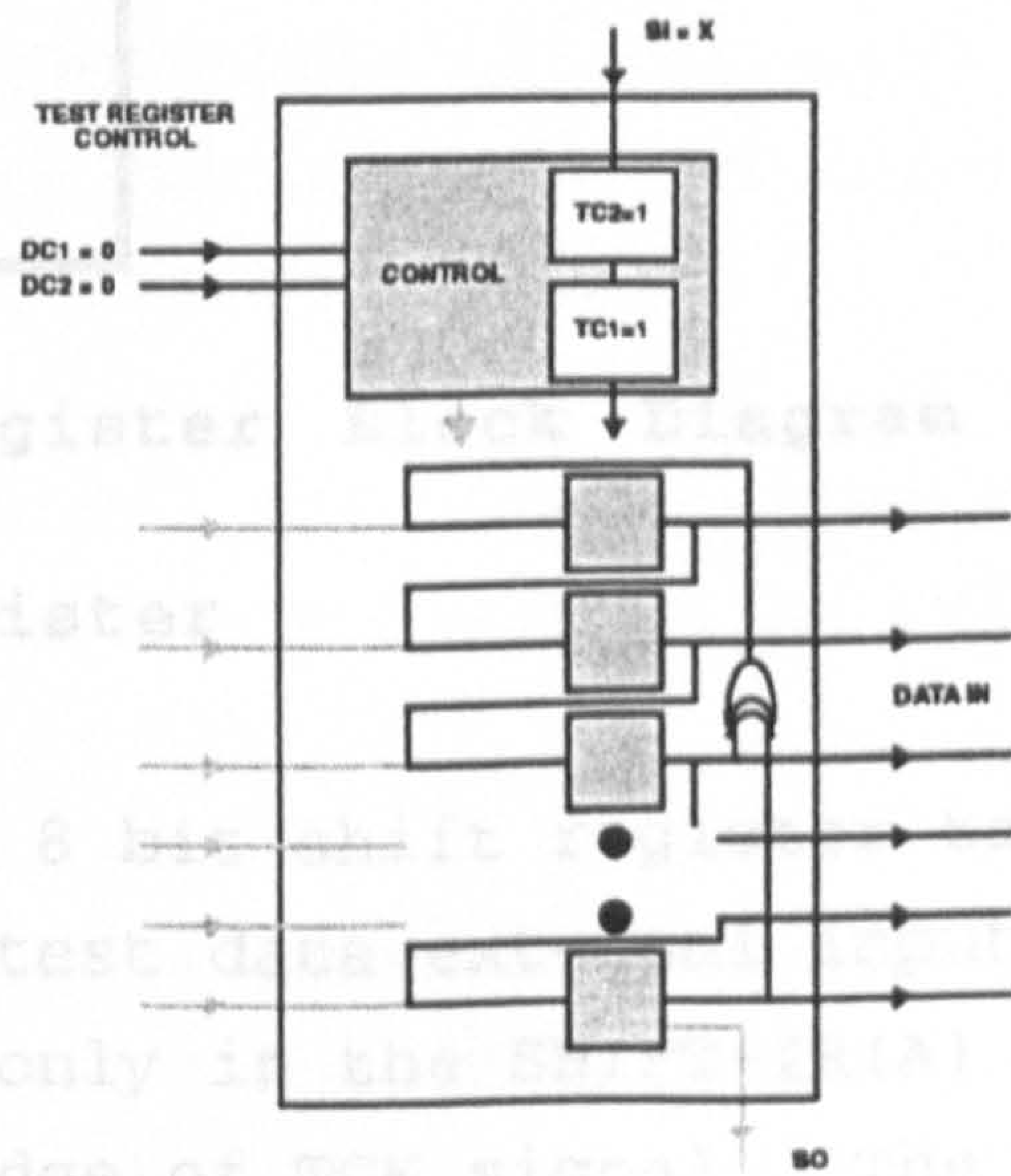
Data is held in the test register. In many tests it is required to hold data stable for the duration of the test while test operations are in progress in another part of the chip.

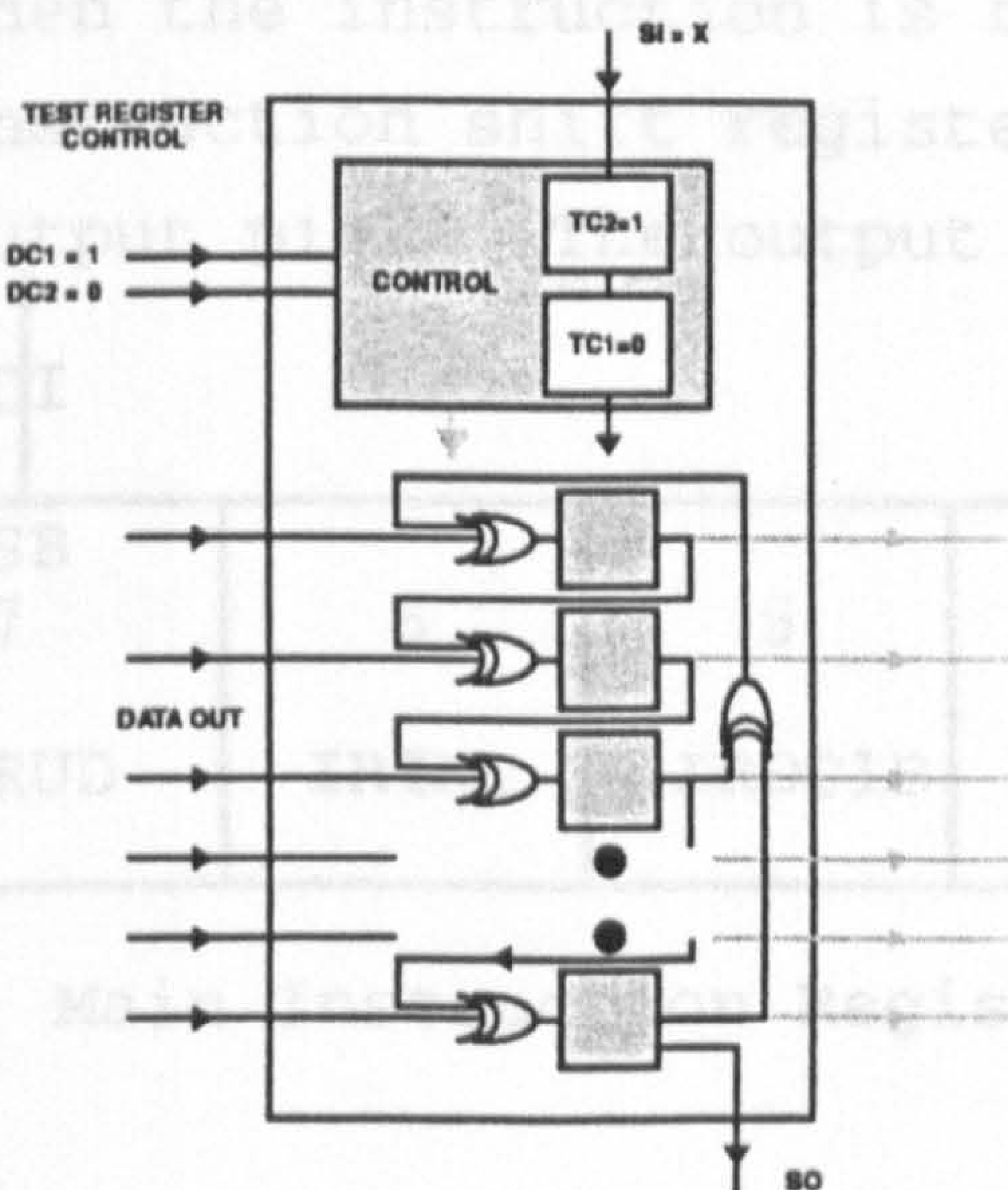
**Figures 5.8.1,2,3,4,5,6**

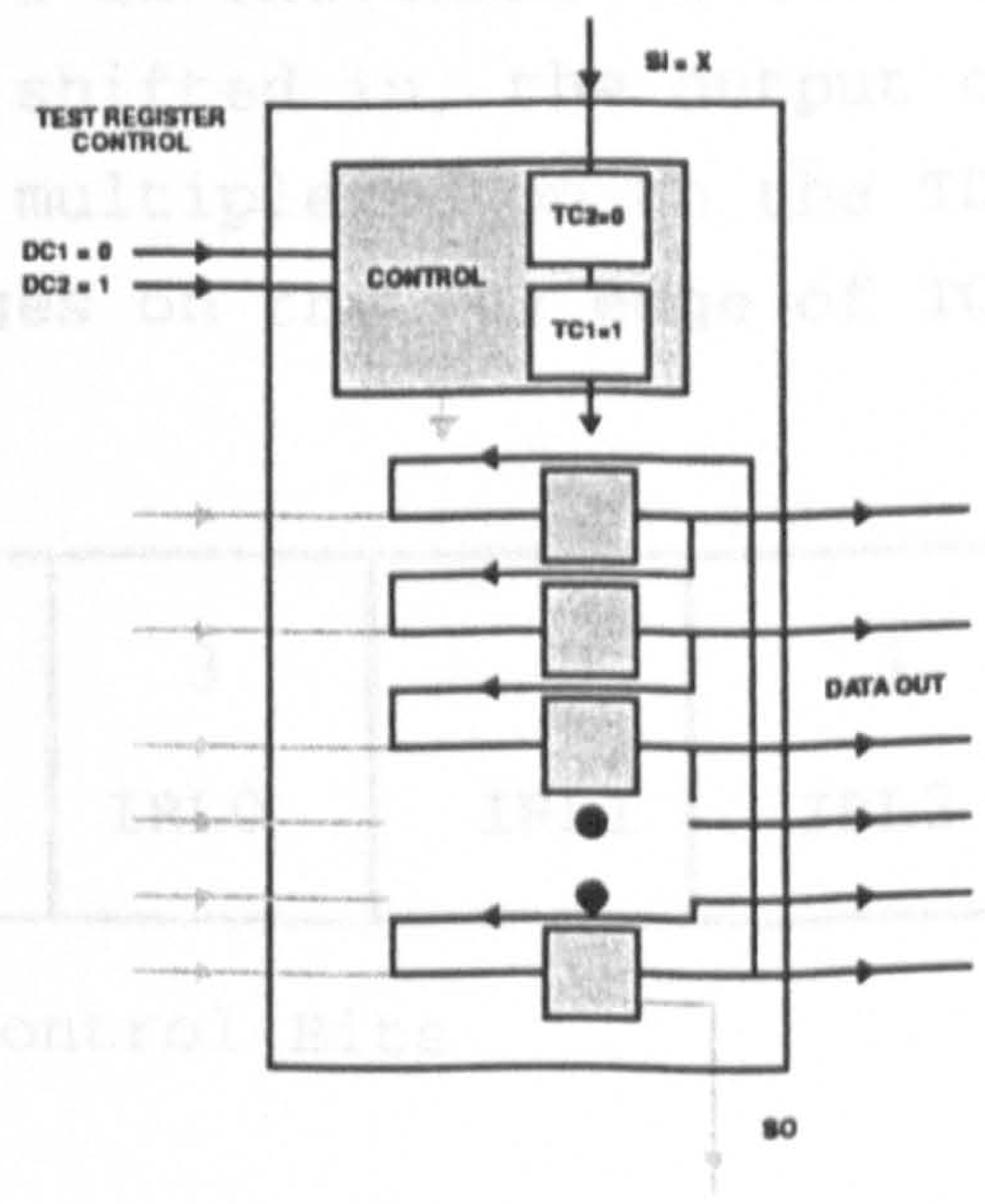


## 5.5 INSTRUCTION REGISTER

The Instruction register is used to load in the required test operation code. It is responsible for providing the address and control signals required to access a particular scan path in the data register.

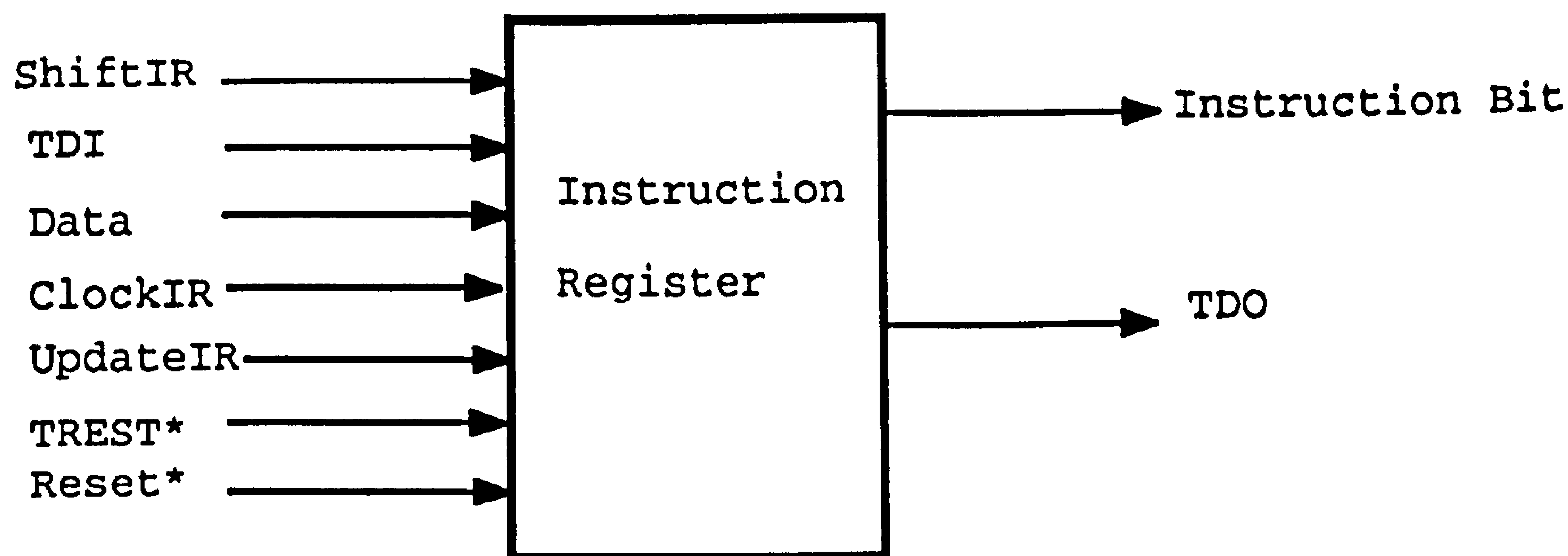
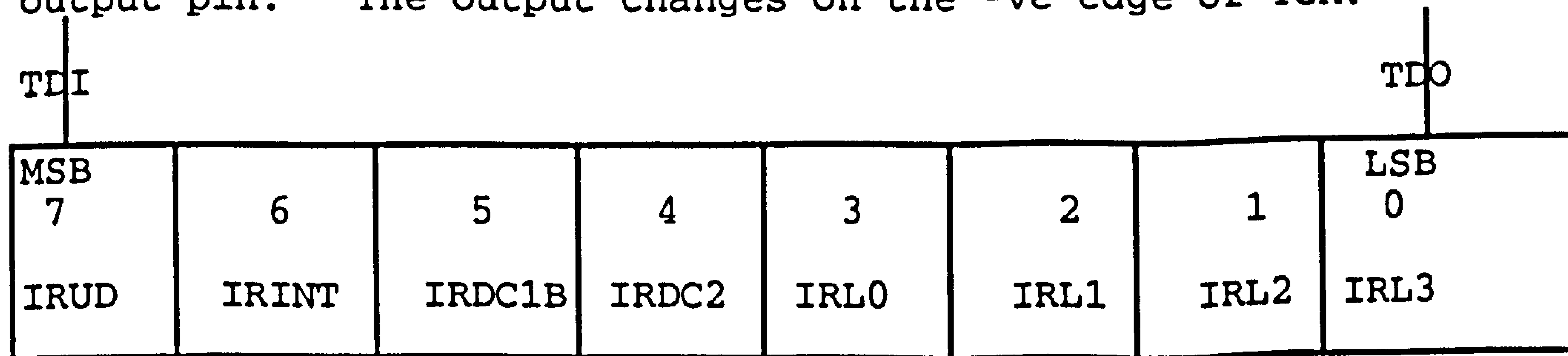



**Figure 5.9 Instruction Register Block Diagram**

### 5.5.1 JTAP Instruction Register

The instruction register is an 8 bit shift register taking its input from the TDI serial test data external input. The instruction is shifted in only in the SHIFT-IR(A) state and is clocked in on the +ve edge of TCK signal. The instruction only becomes active in the UPDATE-IR(D) state. When the instruction is being shifted in, the output of the instruction shift register is multiplexed on to the TDO output pin. The output changes on the -ve edge of TCK.




Main Instruction Register Control Bits

| IR SAMPLE | IR TSCKE | IR RESET |
|---|---|---|

Supplementary Control Bits

When an instruction is shifted in, 8 bits of data are loaded into the instruction register. Bits 7-4 (labelled IRUD, IRINT, IRDC1B, IRDC2) determine which test operation is to be performed on the scan path loop, while Bits 3-0 (labelled IRL0-3) determine which scan path loop is to be tested (note that the bit order is reversed).

Three Supplementary Control Bits provide additional control (IRSAMPLE,IRTSCKE,IRRESET). These bits are not part of the serial instruction register path but are set and reset by decoding the 8 main instruction register bits

Special escape codes on bits IRL0-3 are used to give extra functions not possible using the 8 bit instruction register alone.

Six bits ST10-5 can be used to give general purpose status information. The status inputs are loaded into instruction register bits 0-5 when in the CAPTURE-IR state and are shifted out to the TDO pin when a new instruction is shifted in.

<table>
<tr><th>Bit</th><th>Name</th><th colspan="2">Description</th></tr>
<tr><td>MSB 7</td><td>IRUD</td><td colspan="2">Update Disable and Sample mode reset. Control the updating of the test registers via the UPHLD signal<br>0 = Update in UPDATE-DR(5) state + Sample mode reset<br>1 = Don't Update during UPDATE-DR(5) but hold old value<br>When TRST=0 or when in TEST-LOGIC-RESET (F) state IRUD=1</td></tr>
<tr><td>6</td><td>IRINT</td><td colspan="2">Internal/External test select. Selects whether on Boundary loop 0, the input test registers hold data, or whether the output test registers hold data.<br><br>0 = External Test, outputs hold whilst inputs Capture.<br><br>1 = Internal Test, inputs hold whilst outputs Capture. Valid only when RUN is loaded into the Instruction register<br><br>When TRST=0 or in TEST-LOGIC-RESET(F) IRINT is set to 1</td></tr>
<tr><td>5</td><td>IRDC1B</td><td colspan="2" rowspan="2">DC Test select signals. Test control signals are applied to the selected loop during the RUN-TEST-IDLE(C) and CAPTURE-DR(6) states only.<br>Note the IRDC1B is inverted to maintain compatibility with the reserved instructions.</td></tr>
<tr><td>4</td><td>IRDC2</td></tr>
<tr><td>3</td><td>IRL0</td><td rowspan="4">SCAN<br><br>LOOP<br><br>SELECT</td><td rowspan="4">Selects the loop to be operated on. Valid for SHIFT-DR(2) RUN-TEST-IDLE(C) CAPTURE-DR(6) states only<br><br>Escape codes 1111 and 0001 are used for special functions.</td></tr>
<tr><td>2</td><td>IRL1</td></tr>
<tr><td>1</td><td>IRL2</td></tr>
<tr><td>LSB 0</td><td>IRL3</td></tr>
</table>

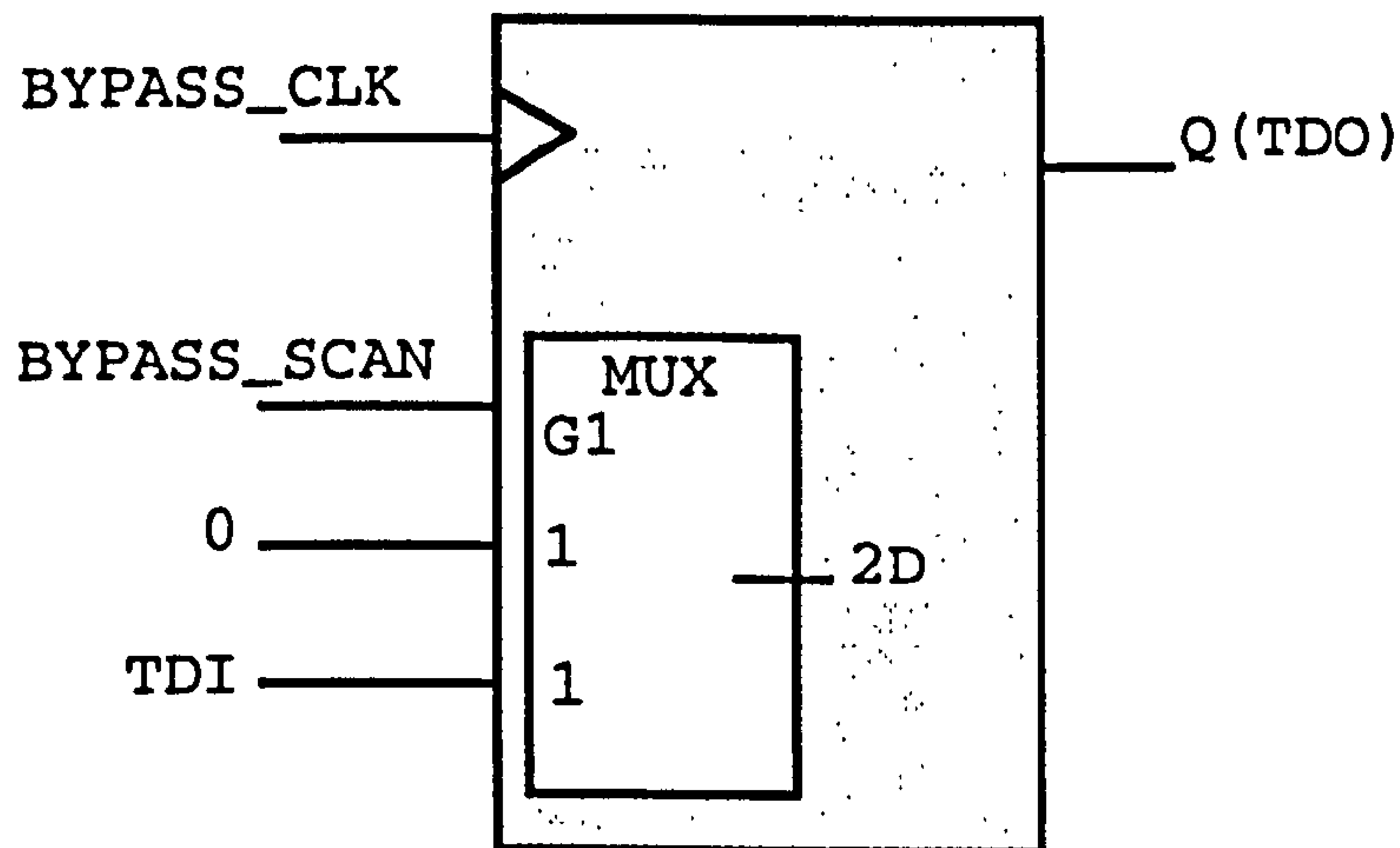
## 5.5.2 RESERVED JTAG INSTRUCTIONS

The JTAG/IEEE 1149-1 spec defines a number of specific instructions which are described below.
Some of these instructions must be implemented to be JTAG compatible and some are optional.

| Instruction Register Bit<br>7 6 5 4 3 2 1 0 | Name Comments |
|---|---|
| 1 1 1 1 1 1 1 1 | BYPASS Bypass register routed to TDO, sample mode operation enabled. Chip operates as normal. |
| 0 0 0 0 0 0 0 0 | EXTEST Boundary Loop 0 is selected. Output pins are updated in UPDATE-DR(5) state. Data is captured from the input pins in the CAPTURE-DR(6) state. Normal Chip operation is interrupted |
| 1 1 0 0 0 0 0 0 | INTEST Boundary Loop 0 is selected Output pins are not updated. Test Data is applied to the input of the chip and chip output data is captured into the boundary loop 0. Normal Chip operation is interrupted |
| 1 X 0 0 1 0 1 1 | SAMPLE Sample mode operation mode enabled on Boundary loop 0 and chip operates normally. Can only be used if Transparent databits are used on Boundary Loop 0. |
| X X 1 0 0 0 1 1 | RUNBIST Can be used to invoke an autonomous self-test in RUN-TEST-IDLE (C) state but only if the extra self-test circuitry fitted to CTRDC1, 1 JTAP input pins.<br>The implementation of this instruction is up to the designer. |
| 0 0 0 0 0 0 0 1 | IDCODE If optional ID Register is used then this instruction will shift out the chip ID code. Normal chip operation continues. |

## 5.6 BYPASS REGISTER

The Bypass register is a single cell register that is used to serially shift test data from TDI (Test Data Input) to TDO when testing of a particular device is not currently required. The data transmitted should be fed from TDI to TDO unaltered but will have a lag of one clock cycle.



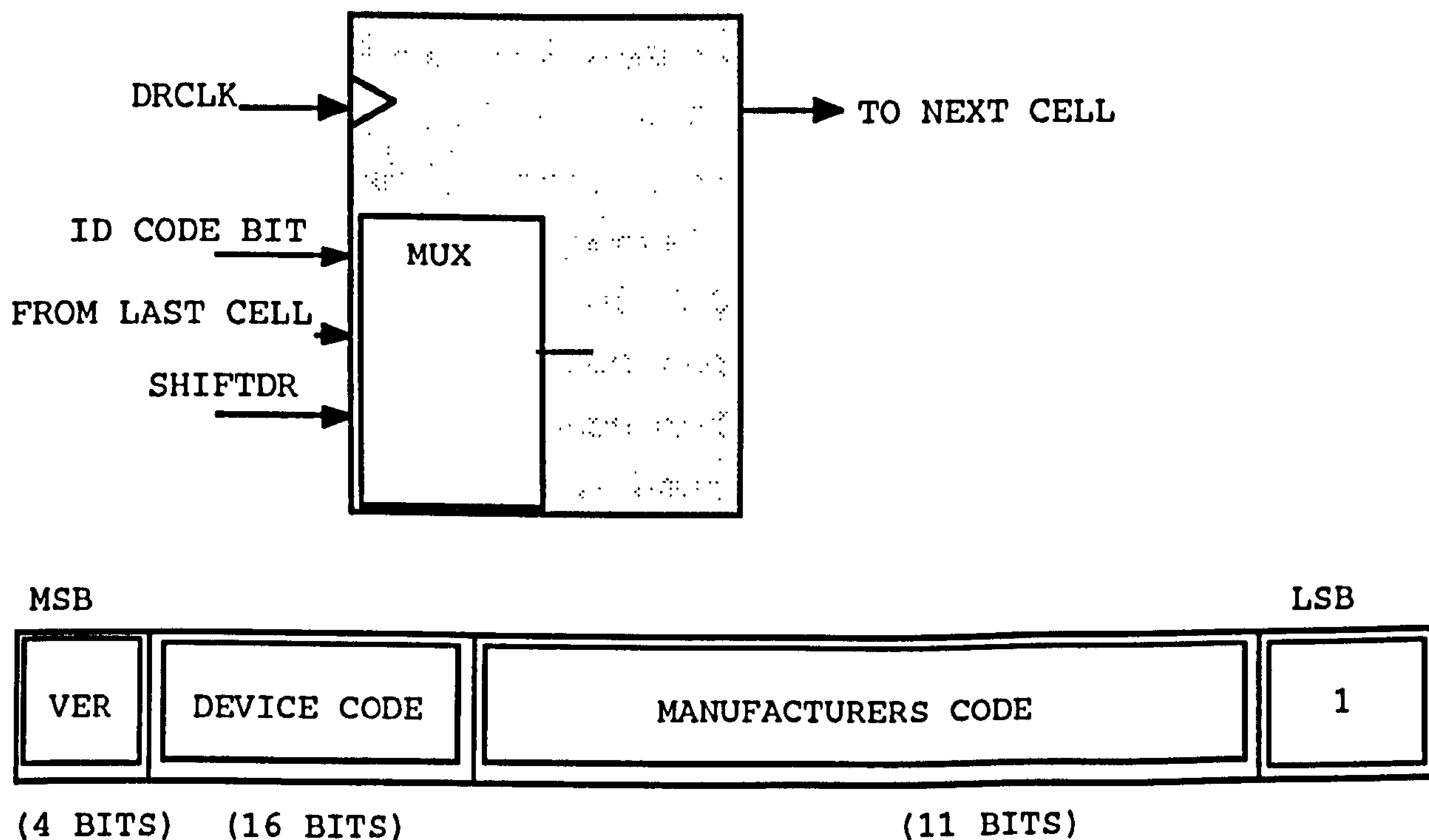

BYPASS_CLK IN PHASE WITH TCK
BYPASS_SCAN CHANGES ON NEGATIVE EDGE TCK

| BYPASS_CLK | BYPASS_SCAN | Q | |
|---|---|---|---|
| ↑ | L | 0 | *CAPTURE* |
| ↑ | H | TDI | *SHIFT* |
| L | X | Q | *HOLD* |

Figure 5.10 Bypass Register Block Diagram

## 5.7 DEVICE IDENTIFICATION REGISTER

The Device Identification Register is an optional feature [WHET 87] of the Standard. When it is included in the test logic, it allows a binary data pattern to be read from the chip identifying the manufacturer, the part number and the variant. During testing this information might be used, for example, to verify that the correct IC has been mounted in each board location.




**5.11 Device Identification Register Block Diagram**

If the designer chooses not to implement the ID register, it is necessary to connect the unused ID register input on the JTAP controller to the bypass register output. This ensures compatibility with the JTAG/IEEE 1149-1 spec in that when the ID register instruction is selected, the bypass register path is selected.

## 5.8 TEST ACCESS PORT (TAP)

The purpose of the JTAP cell is to convert the external chip JTAG test control signals to internal signals used to control the test registers when testing takes place. The JTAP cell has an external interface which connects to I/O pins of the chip, together with an internal interface which connects to the internal test circuit.

The block diagram for JTAP is shown in figures 5.12 and 5.13. The state machine controls the main operation of the JTAP controller as shown in figure 5.14. It controls the operation of the internal testing. It also controls the shifting in and out of serial data from the scan loops and the shifting in and out of data from the instruction register. The Instruction register selects which scan loop is to be shifted or tested and the test to be performed. Under control of the JTAP state machine, instructions are shifted serially into the Instruction register via the TDI serial data pin. The loaded instruction then controls the tests to be performed. All serial data and instructions are shifted in via the TDI pin and, depending on which register is selected, the serial output is shifted out to the TDO pin. A single register Bypass path from TDI to TDO is provided to give a direct path from TDI to TDO. This gives shorter access to chips further along the serial data path. The JTAP controller provides connections to an optional ID register, whose purpose is to output serially an identity code that can be passed out to the tester. This register is optional and when not used, the Bypass register is connected in its place. The JTAP controller provides the signals required to control each scan path loop independently.

The connections to the JTAP controller can be divided into three categories:-

- External Interface Signals - JTAG signals to the external chip pins.
- Internal Interface Signals - Signals to control the internal test circuitry.
- Local Interface Signals - Signals used to control a local ID register and unused scan path loops.

### 5.8.1 JTAG SIGNALS (TCK,TMS,TDI,TDO,TRST)

Access to the boundary scan and the data transmitted is via 4 or 5 buffered I/O test pins called a test access port or TAP.

There are four mandatory pins and one optional pin as listed below:

| | |
|---|---|
| TCK | - test clock |
| TMS | - test mode select |
| TDI | - test data in |
| TDO | - test data out |

The optional pin:

| | |
|---|---|
| TRST * | - test reset input pin |
| | (* - active low) |

The TRST* pin need only be added if, as in this case, there is no 'reset on power up' present. It provides asynchronous initialisation of the TAP controller by returning it to the (Test-Logic-Reset) state.

**TCK**

TCK is an external test clock used for shifting data through the cells. Note that it is totally independent of the system clock which means that data can be loaded without altering the current state of the device.

**TMS**

TMS is used to change the current status of the TAP controller. If the TMS line is not driven, internal circuitry should send the controller into the 'Test-Logic-Reset' condition on five successive cycles of the TCK. It will remain in this state until the TMS line changes.

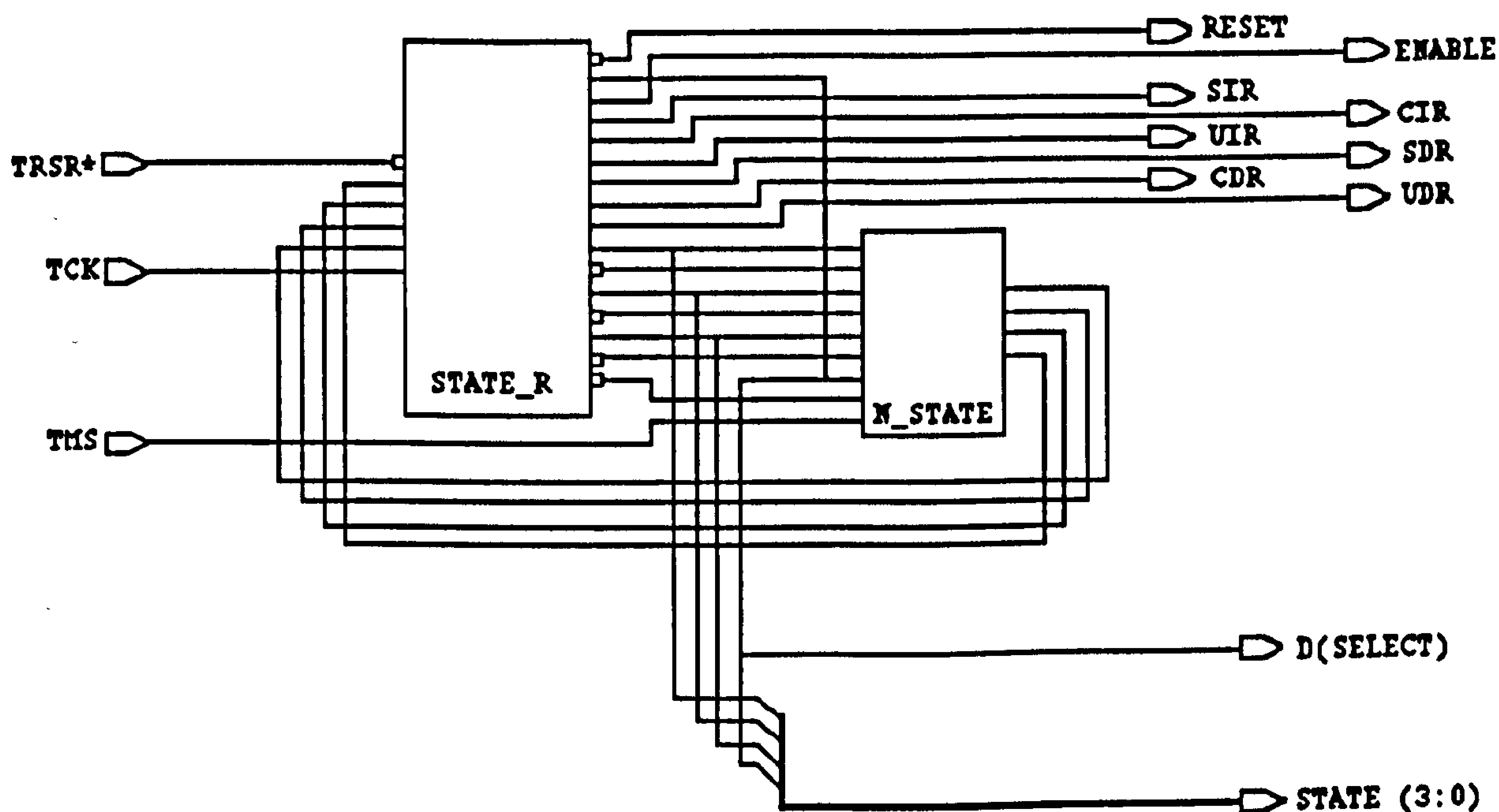



**Figure 5.12 TAP State Machine Block Diagram**

Note that the (Test-Logic-Reset) is the default condition in which normal operation of the device is obtained. This means that should the TMS line fail or be unused the device can still function normally.

**TDI**

TDI is used to load test data serially into the device via the Test Access Port to the first boundary cell. With the correct instruction this will be shifted left through all the cells on the rising edge of TCK until the controller status changes.

If a requirement exists where the test operation is to be repeated but with fresh data, the input to the TDI can be reloaded whilst the old data is being clocked out.

**TDO**

TDO is used to accept serial data that has been transferred through the device. Unlike the TDI, the test output data shifts on the falling edge of the clock to prevent race hazards across the register.

**TAP Controller**

The operation of the boundary scan cells and the conditioning of other inputs is controlled by the TAP controller. This is a state machine, which sets all the control signals required for the operation of JTAG.

States are assigned a Hex number as shown in figure 5.14. The state machine is controlled solely by external input pin TMS (Test Mode Select) and clocked on the +ve edge of TCK (Test Clock). The TEST-LOGIC-RESET (F) state is the non-test state in which the chip is running normally. An asynchronous reset signal (TRST) is required to put the state machine into TEST-LOGIC-RESET (F) state and is normally present as a power-on-reset signal. Together with the contents of the instruction register, the state machine controls the timing of the shifting of serial data through the scan loops and the capturing and application of self-test functions. Two main paths in the state diagram determine whether the instruction register (IR) is shifted and loaded, or whether the scan paths (DR) are shifted or loaded. Self-Test is performed in the RUN-TEST-IDLE (C) and CAPTURE-DR (6) states. The instruction register's contents determine which type of test should be run during these states.

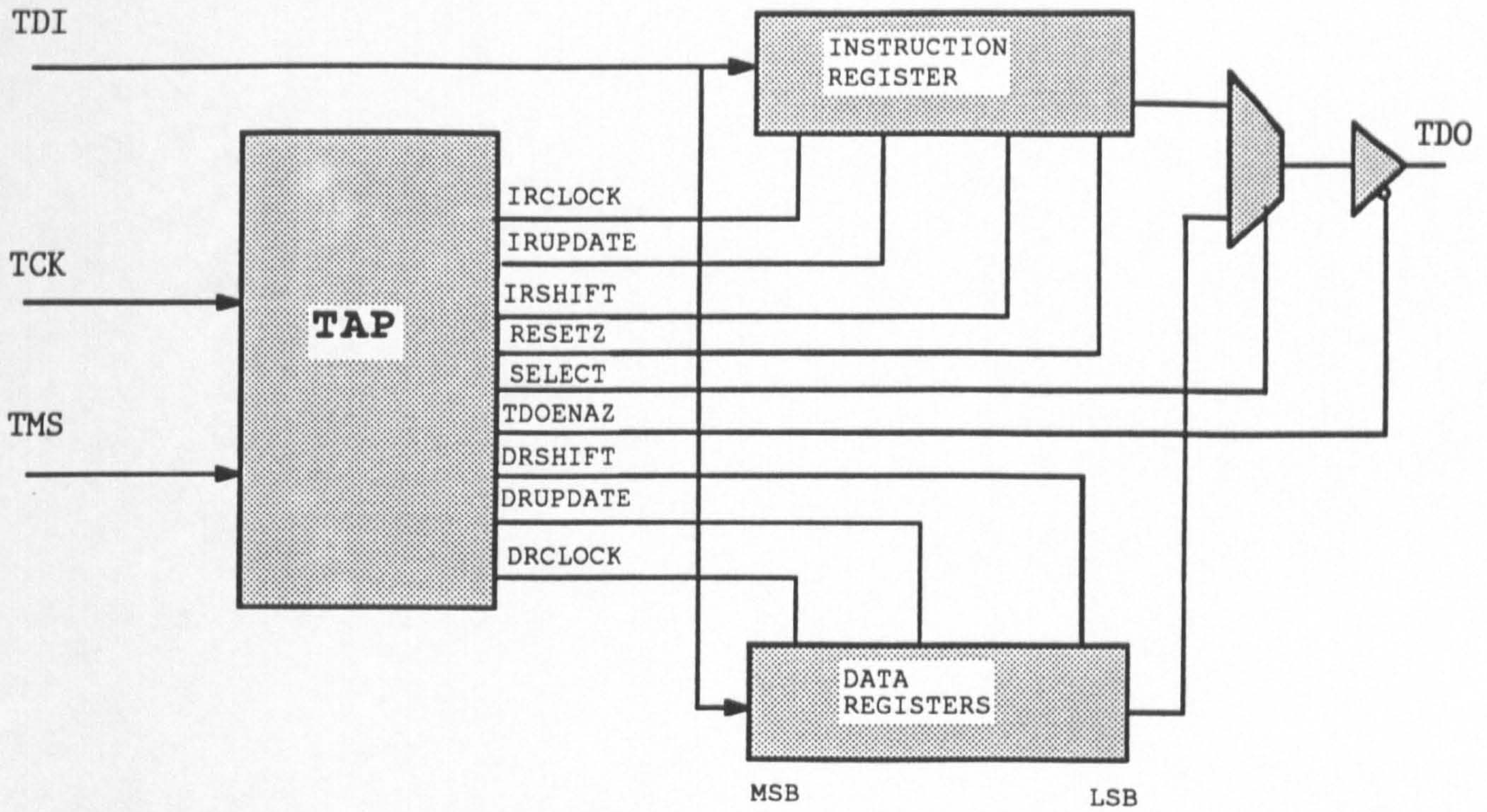



**Figure 5.13-a TAP Controlling Instruction and Data Registers**

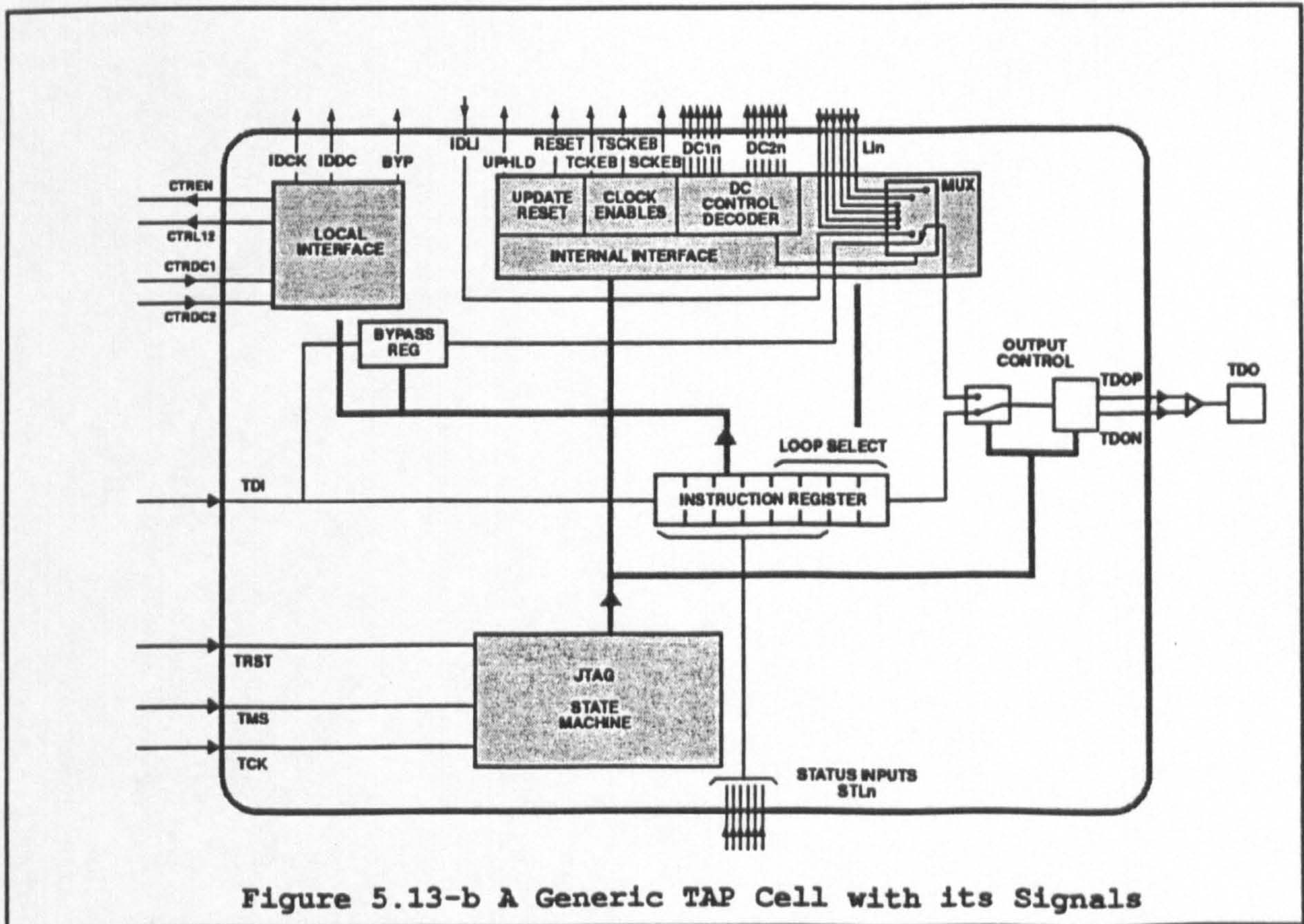



**Figure 5.13-b A Generic TAP Cell with its Signals**

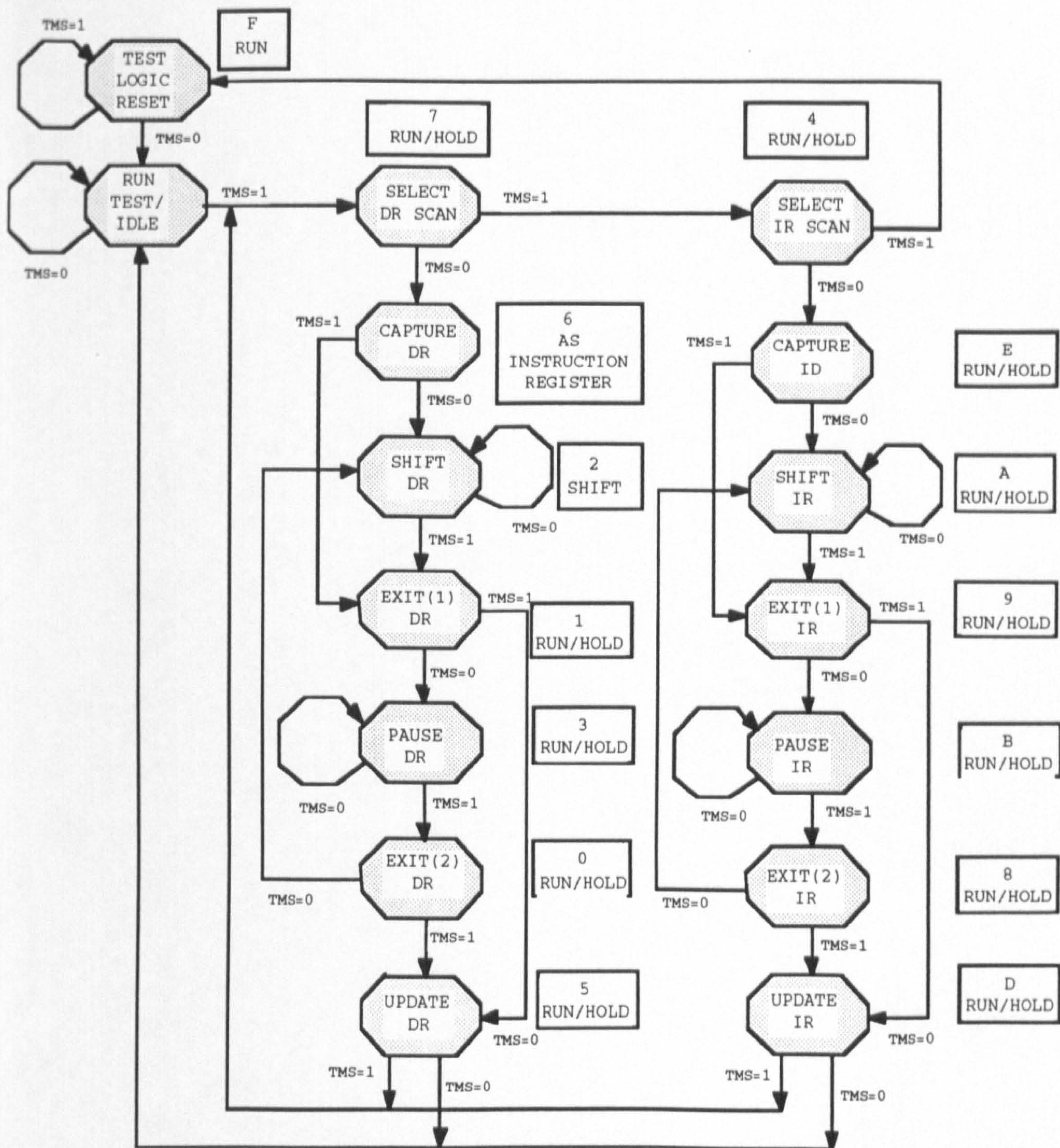



**FIGURE 5.14 TAP Controller State Diagram**

Depending on the position and the current status of the machine 16 possible states are possible, these are listed below:

| | | | | | |
|---|---|---|---|---|---|
| Exit2-DR | 0 | Update-DR | 5 | Pause-IR | B |
| Exit1-DR | 1 | Capture-DR | 6 | Run-Rest/Idle | C |
| Shift-DR | 2 | Select-DR-SCan | 7 | Update-IR | D |
| Pause-DR | 3 | Exit2-IR | 8 | Capture-IR | E |
| Select-IR-Scan | 4 | Exit1-IR | 9 | Test-Logic-Reset | F |
| | | Shift-IR | A | | |

## 5.9 RECOMMENDATIONS

### 5.9.1 HOW TO LAY OUT THE TEST REGISTER

The layout of the test register should be considered carefully in order to produce an efficient chip layout. The test registers should be arranged such that the pitch of the databits matches the pitch of the data lines coming from and going to the main logic. This ensures that data flow does not have to 'dog-leg' through the chip taking up silicon area as demonstrated in fig 5.15 a,b.

The Tall databits have a width of 2 minor cells, while the Fat databits have a height of 2 minor cells as shown below. Therefore Test registers can be made to match data busses with any pitch down to 2 minor cells horizontally or vertically (whole cell numbers only!). If a finer pitch is required it is possible to place two Test registers next to each other running in parallel thus giving an effective pitch of one minor cell. Routing Transparency through the databits ensures that signals can pass through the databits.

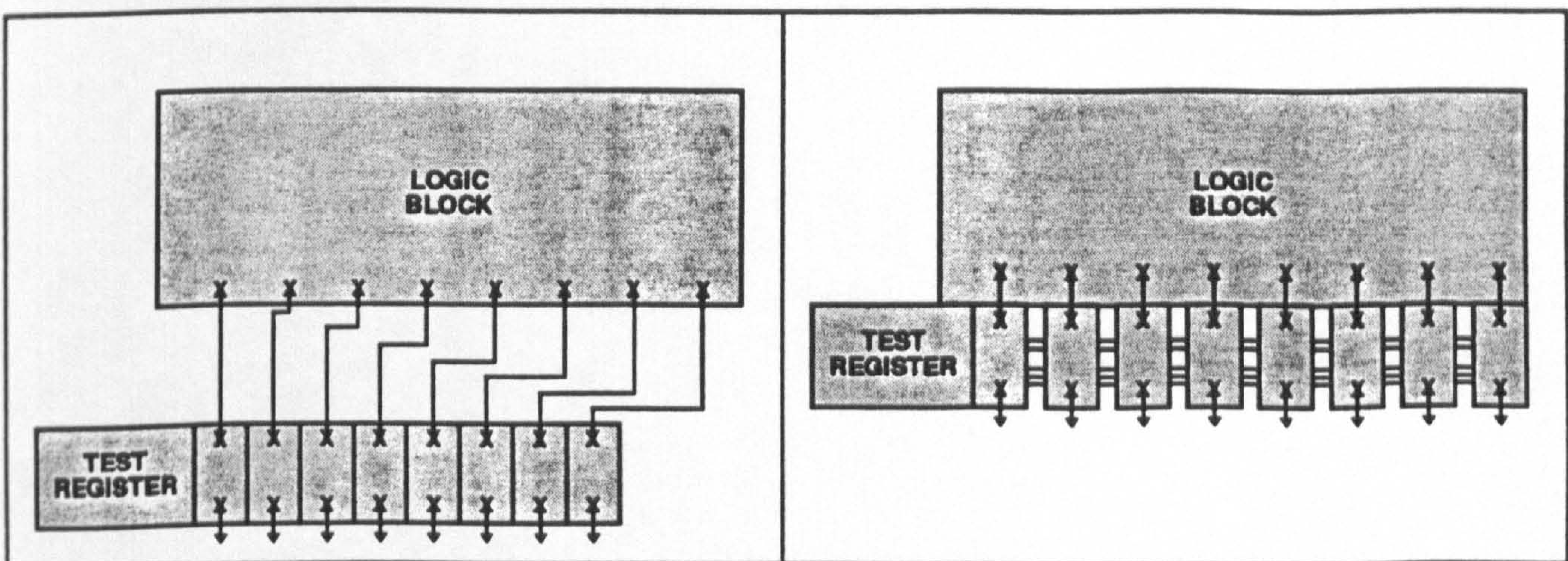



**Figure 5.15 a,b Test Register Layout Techniques**

Where the horizontal pitch exceeds 5 or 6 minor cells for a horizontal Test register, the wasted gaps between the databits make the Test register inefficient. In this case instead of using the Tall databits, it may be more appropriate to use Fat databits which are 5 or 6 cells wide but only 2 cells high. Obviously the connections between the Fat databits will not be optimum and external routing is required. However, the saved area is considerable. Likewise when vertical databit pitch is greater than 5 or 6 minor cells then Tall databits may be used.

Wherever possible, use should be made of the butting cells to reduce the routing of the Test register.

### 5.9.2 PSEUDO RANDOM TEST

The JTAG/IEEE 1149-1 spec requires that a pseudo-random test is run in the RUN-TEST-IDLE(C) state and that the resulting signature should be independent of the number of clock cycles in the RUN-TEST-IDLE(C) state (above a minimum number). There is therefore a requirement to be able to freeze the signature after a given number of clock cycles when in the RUN-TEST-IDLE(C) state.

Having this requirement means that blocks of logic can have their signatures predicted for a fixed number of clock cycles. As signature prediction for a pseudo-random test on a block of logic can be very CPU expensive, the ability to freeze the signature after a given number of clock cycles means that a signature prediction need only be determined once for a block of logic. It is also independent of the length of test. It is therefore recommended that a counter should be included which can be used to hold the result of the test after a given count.

### 5.9.3 AUTONOMOUS SELF-TEST (RUNBIST)

It is possible to design into the test circuit an autonomous self-test function that can invoke a complete self test without the need to shift in complicated serial test data to run the self test. The intention is that the self-test is completely self running and can be invoked by a single RUNBIST instruction. Boundary loop 0 can be selected from the instruction register by more than one loop number.

Boundary loop 0 serial data input passes through a multiplexer which provides an alternative source of serial data into the boundary loop from a ROM. The ROM contains the control bits and seed values that are needed for a pseudo-random test.

## 5.10 CONCLUSIONS

The IEEE 1049.1 JTAG structure has been described. The test architecture was successfully simulated at the register transfer level. The full simulation results are shown in Appendix 5A which are based initially on 4 test instructions including NOP, SAMPLE, EXTEST and BYPASS.

The 'macro function' language facility (similar to 'C') was used to describe the stimuli needed to simulate the architecture fully. An application logic, a 2 bit adder circuit, was then included. Both logic simulation and fault simulation was successfully carried out based on 6 test instructions which have been generated using macro-functions.

The development, implementation and simulation of the JTAG architecture provided an understanding of the operation needed for developing the architecture behaviourally using VHDL.

**REFERENCES**


[IEEE 1149] IEEE Standard 1149.1, 'A Standard Test Access Port and Boundary Scan Architecture' May 1990.

[MAUND 87] Maunder C and Beenker F 'Boundary Scan: A Framework for Structured Design For Test' Proceedings IEEE International Test Conference, 1987,pp.714-123.

[BEEN 85] Beenker, F 'Systematic and Structured Methods for Digital Board Testing, 'Proceedings IEEE International Test Conference, 1985, pp. 380-385.

[WHET 88] Whetsel, L 'A View of the JTAG Port and Architecture', ATE & Instrumentation Conference West, January 1988, pp 385-401.

[ICCD 87] Pradham, M. M., Tulloss R.E., Beenker, F.P.M., and Bleeker, H., 'Developing a Standard for Boundary Scan Implementation,'Proc. International Conference on Computer Design, Rye, New York, October 5-8, 1987, pp.426-466.

[WHET 87] Whetsel, L ' A proposed Standard Test Bus and Boundary Scan Architecture', Application Paper, Texas Instruments Inc, Test Automation Dept, USA.

# CHAPTER 6
# HIGH LEVEL VHDL MODELLING OF BOUNDARY SCAN ARCHITECTURE

## 6.0 INRODUCTION

This Chapter presents VHDL Models developed for the different components of the JTAG Boundary Scan Architecture (BSA). It describes the techniques used for accurate, high level modelling of the BSA.

Both Mentor Graphic's -System 1076 version 7 running on an Apollo 4500 workstation and View Logic's VHDL subset running on a PC-386 were used to develop, compile and validate the behavioural models of BSA.

The chapter highlights the use of some of the features provided in VHDL, such as the Package procedural facility, when developing the BSA models.

## 6.1 BACKGROUND

Functional level test generation continues to take an increasingly significant role in addressing the test requirements of digital circuits. A functional test examines the correctness of a given circuit's performance in its various modes of operation. Therefore, when functional-level test generation techniques are developed and applied to digital networks, determining the correctness of a logical operation, independent of specific implementation, is a primary objective.

What is gained by using a high-level approach in applying Design-for-Test generation is freedom from the need for implementation details of circuits to be tested. Herein lies a marked advantage of a high-level methodology over many classical test generation strategies relying to a greater or lesser degree on specific structural information. Structural information is not readily available in many situations, thus negating the applicability of many test generation strategies relying on such information.

For example, structural information is not always available in circuits designed with the aid of a silicon compiler or by using a Third Party Cell library. Field Programmable Logic Devices also do not provide structural information for the user. In all cases however, functional descriptions of digital circuits are available and serve as a starting point for a high level approach.

For a high-level approach to be successful, the functional description and logical behaviour of a digital device under test must be represented in a comprehensive and concise manner.

The approach taken by M A Brewer and A D Friedman [BREU 80] is based on Functional Level Primitives to describe high-level representation.

Another approach suggested by U J Dave and J H Patel [DAVE 89] is based on two-level representations. This is a scheme used for representing the input-output behaviour of combinational circuits which rely solely on their corresponding functional description.

Other schemes such as the use of binary decision diagram for describing high level logical behaviour of the circuit under test have been suggested by Akers, Abadir and others [Abad 85], [Aker 78].

The approach described in this chapter is based on the use of Hardware Description Language VHDL, to describe behaviourally the IEEE 1149.1 Boundary Scan Test Architecture [JTAG 90].

## 6.2 MODELLING OF THE JTAG ARCHITECHTURE

When a function rather than a structure is most important, it is possible to describe each component with a corresponding behavioural description. VHDL behavioural description is therefore used to represent the function of JTAG in terms of circuit and signal response to various stimuli. The VHDL models are also described using a schematic editor with an added attribute which links the circuit schematic to the behaviour of the component during the circuit simulation.

After the successful simulation and refinement of the functional model of JTAG with the application circuit, it is then possible to substitute the behaviour with a vendor specific structural architecture. In this way the intellectual property rights of the component models supplied by the IC vendor are protected.

The timing elements associated with each model of the BSA have been defined in a generic form. These declarations are made visible to all entities of JTAG through the use of the PACKAGE facility provided in VHDL. The *'declaration package'*

contains all the necessary delay parameters. It gives the user the ability to define the desired delays easily.

In addition the full JTAG architecture is described within the *'work package'* later in this chapter, in order that the desired delays can be defined easily. [IEEE 88] [COEL 89] [LISP 89]

## 6.3 JTAG BOUNDARY SCAN ARCHITECTURE TIMMING AND PERFORMANCE ISSUES

### 6.3.1 CLOCK OPERATION

The test logic must be capable of operating with TCK in the frequency range identified by the JTAG IEEE standard. The maximum frequency of TCK must therefore be clearly specified. This rule is introduced in order to ensure a minimum acceptable level of performance for the test logic. The minimum performance for a data path on the test card will be dictated by the slowest model, with respect to the clock speed. The time taken to shift the data along the test data path on the card increases if the maximum clock frequency expected is reduced.

A VHDL "TCK_GENERATOR" model was written for the TCK clock generator. This produces a clock with a 50% duty cycle and a PER period of 200 nanoseconds (which is the maximum period). The generation of the clock will only take place after a transition of '0' to '1' on the Run input. On the other hand, a transition of '1' to '0' on Run causes the clock generator to stop. In this way the TCK clock can be controlled.

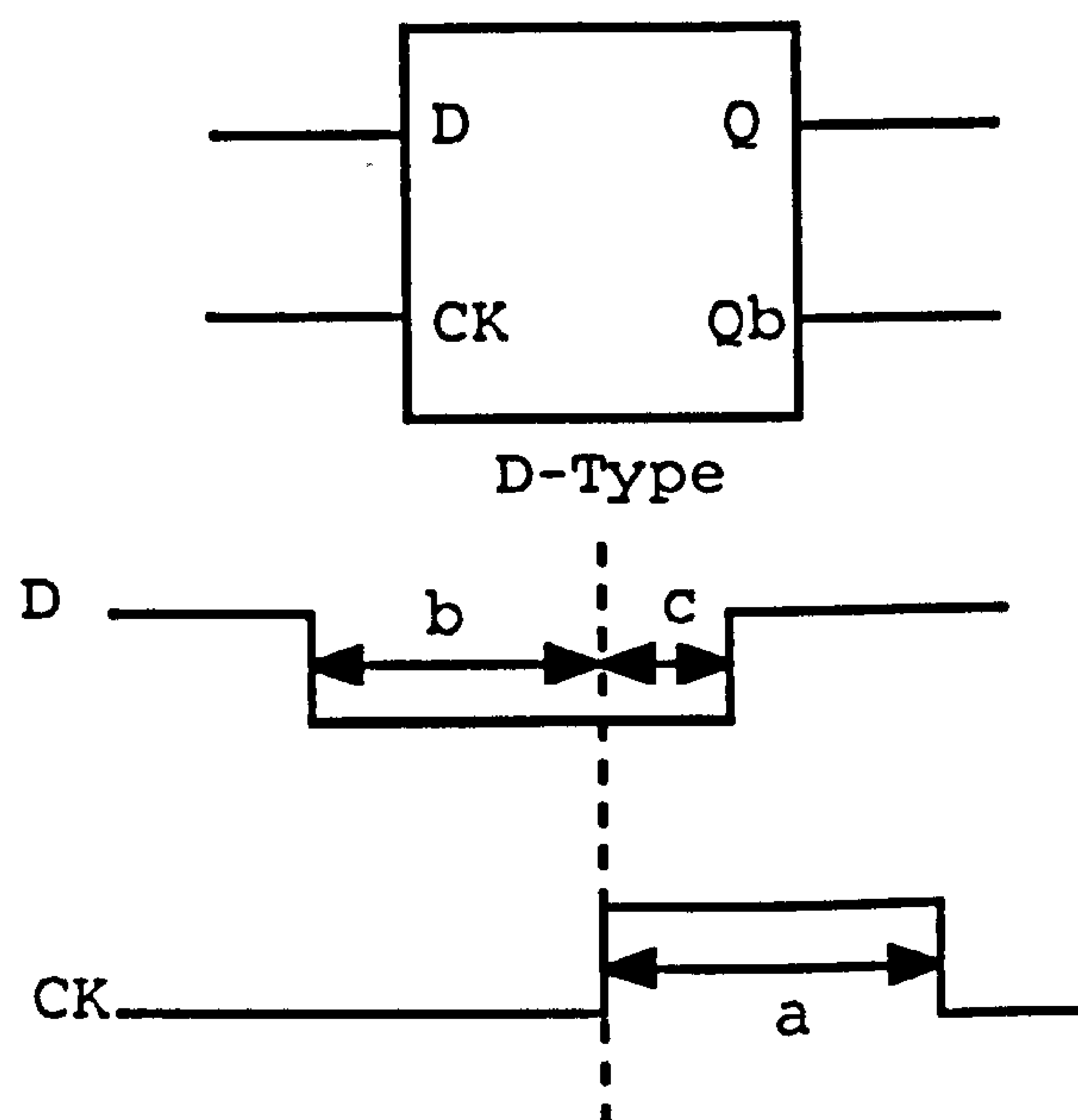
### 6.3.2 SET-UP AND HOLD TIMES

Definition 1: The set-up time is the interval of time between the application of a signal at a data input terminal and the following active transition on a clock input terminal.

A value of b in figure 6.1. is an example of the set-up time for the data input terminal D.

Definition 2: The hold time is the time interval during which a signal is saved at an input terminal after an active transition has taken place on a clock terminal.

A value of c in figure 6.1. is an example for the hold time for the data input terminal D.




Two wave forms for input signals

a : duration of the clock pulse.

**Figure 6.1 Set-up and Hold Times**

### 6.3.3. PROPAGATION DELAY OF SIGNALS

The propagation of a signal through a circuit is delayed by a time which is equal to the minimum at $tp_{min}$ (Figure 6.2). This delay can be prolonged by factors external to the circuit, such as the input or mains voltage (the lowest voltage permitted must be considered). The other factor is temperature as the performance of a circuit deteriorates at very high temperatures. Figure 6.2 shows the minimum propagation time which is calculated by taking account of the worst parameters, where $tp_{min}$ represents their best case.

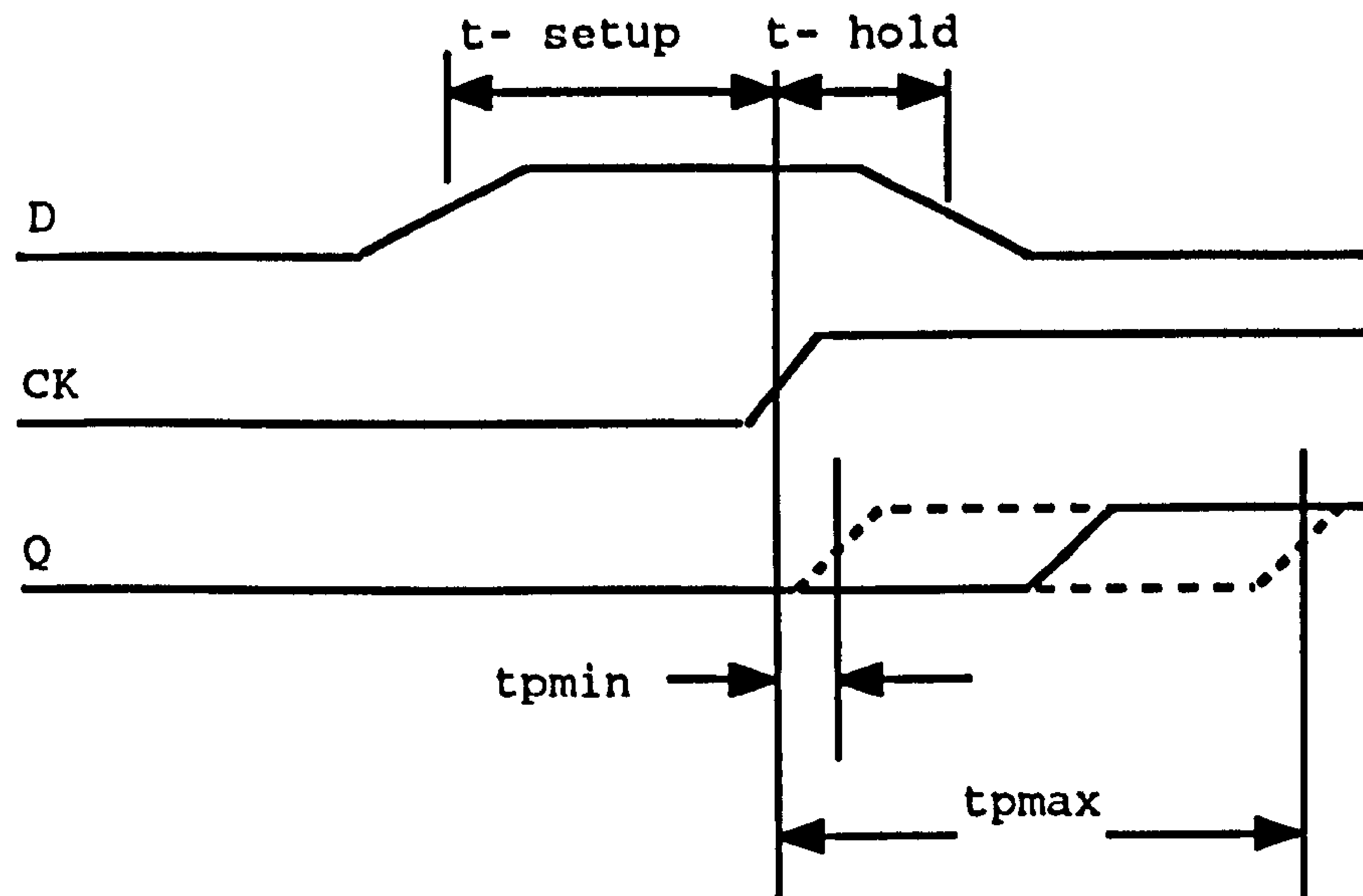



**Figure 6.2. Maximum and Minimum Propagation Delays**

There are several ways of modelling set-up and hold times with VHDL. Three methods are described below:

- When there is a violation of time the simulator outputs a message identifying the time problem as well as the component which contains the violation. The model will propagate a state as if there had been no time violation and the simulation therefore continues.

- Simulation continues and the models show an unknown state at the output when there has been a violation of time.

- Other practices imply the use of minimal time with precision during the simulation of the logic function. Simulation models do not verify the violations and propagate a state without taking into account the possible time violations.

The first method was chosen to model set-up and hold times of circuits. The VHDL language facilitates the modelling of propagation delays. VHDL also allows the description of two types of signal delays. The first type is *inertial* delay where a signal propagation is only carried out after the set-up time at a given level for a specified moment of time and given by the clause "after".

The second type is the *transport* delay where all the changes on an input propagate to the output without reference to the duration of the set-up time. In the course of modelling this facility offered by VHDL was used and the delays have been symbolised by n and u mnemonics whose values[1] are developed by the *generic* clause.

### 6.3.4 TDO-TDI INTERFACE

The TDO pin must be able to control at least two TAP data input pins (TDI) described in the same technology. The logical levels used at the TAP pins should match those used in other pins in the assembly of the module. It is possible for a TDO to be connected to other test data paths on a circuit card. As a result, TDO is declared as a 3-state pin during modelling.

## 6.4 VHDL MODELS FOR BOUNDARY SCAN ARCHITECTURE

The Boundary Scan Architecture is broken up into several components which are easily found from integrated circuit suppliers. Each BSA component is described in terms of behaviour by a VHDL model. This makes it possible for the models to be extended for other test support and future enhancements. These models reside in a VHDL package and they are interconnected structurally in order to describe the target architecture.

---

[1] During simulation of the models, arbitrary delay values were given to the symbols to represent different propagation delays of signals.

## 6.5 TAP CONTROLLER MODEL

The black box model of the TAP Controller is described below. The VHDL model describes the Controller as a finite state machine (16 states) that responds to changes of TMS and TCK signals.

Figure 6.3 identifies the interface input/output requirement for the TAP Controller.

The "TAP_CONTROLLER" is the name of a model entity of VHDL which describes the behaviour of the TAP controller.

The functioning of the circuit is described by the architecture "tap_controller_behaviour", at a high level of abstraction and is independent of the particular structure. It is therefore only necessary to describe the required behaviour without presenting its characteristics. As the initial state of the controller is undetermined (during power on) it was necessary to add the state UNDEFINED in order to facilitate initialisation.

The interface represents all the signals generated by this model, after a delay ODEL, in order to control the behaviour of all the registers and multiplexers. Therefore, it must assign specific values to a signal during each state reached by its particular function. Thus, if there is an event "0" to "1" on TCK and TMS was initialised, then the present state of the component will be changed to the following state if TMS is maintained at this logical level during a time equals to the Hold_Time or longer. This assumes TCK remains at a value of "1" during a minimum pulse duration Min_Pulse_Width_1, and TMS has a logic value which can alter the state according to the transition diagram after a time equals to the Setup_Time or less.

If this is not the case, an error message of violation will be sent and the simulation will continue with the expected change as if there had been no violation.

The Reset port is used by the instruction register and the boundary scan register cells which represent the control signals for the Tri-state and bidirectional pins. The Selectt port is used to select a serial data test port (either a serial instruction register output or a selected test data register). The Enablee port is used to validate the test data output buffer since it is a three state port. The three ports: ShiftIR, ClockIR and UpdateIR are used in the design of the instruction register. The ShiftDR, ClockDR and UpdateDR ports are used in the design of the test data registers.

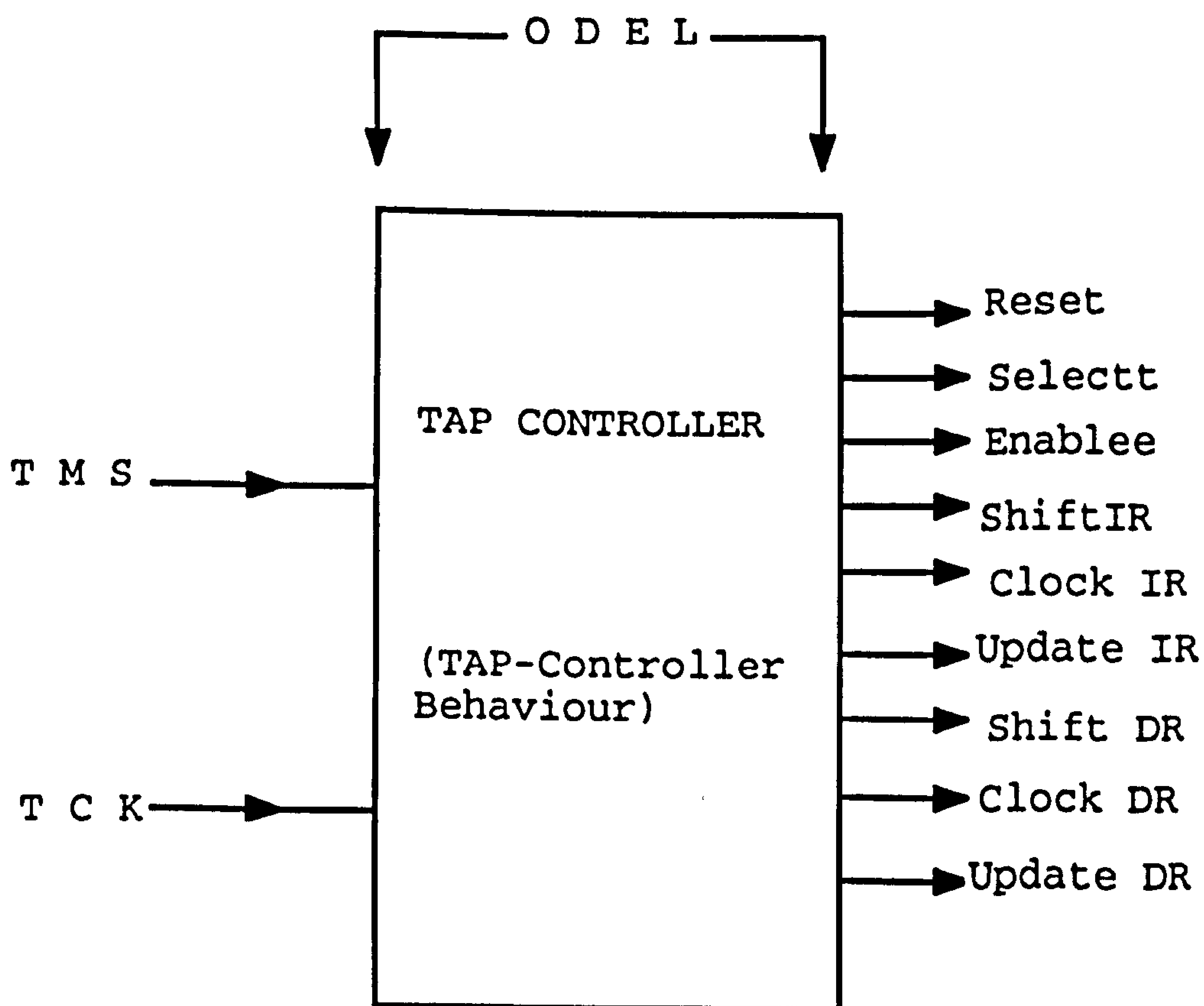



**Figure 6.3 The TAP Controller**

```
library std,work;
use std.standard.all;
use work.declaration.all;
```

```
ENTITY tap_controller IS
        generic  (Setup_Time, Hold_time,Min_Pulse_Width_1,
                  Min_Pulse_Width_0, S_Odel, Odel:Time);
        PORT (
        TMS,
        TCK: in bit;
        Reset,
        Selectt,
        Enablee,
        ShiftIR,
        ClockIR,
        UpdateIR,
        ShiftDR,
        ClockDR,
        UpdateDR : out bit
    );

END tap_controller;


ARCHITECTURE tap_controller_behav of tap_controller is

      signal state: state_tap := undefined ;

BEGIN

state_diagram:process (tck)

      begin

      if tck = '1' then

      case state is

      when Test_logic_reset=> if TMS='0' then
                  state <=Run_state_idle after S_Odel;
      end if;

      when Run_Test_Idle => if TMS ='1' then
                  state <= select_DR_Scan after S_Odel;
                  end if;

      when Select_DR_Scan => if TMS ='0' then
                  state <= capture_DR after S_Odel;
                  else
                  state <= select_IR_scan after S_Odel;
                  end if;

      when Capture_DR => if TMS = '0' then
           state <= shift_DR after S_Odel;
                   else
                   state <= Exit_1_DR after S_Odel;
                   end if;

      when Shift_DR => if TMS ='1' then
                   state <= Exit_1_DR after S_Odel;
                   end if;
```

```
when Exit_1_DR => if TMS ='0' then
                state <= Pause_DR after S_Odel;
     else
     state <= Update_DR after S_Odel;
     end if;

when Pause_DR => if TMS ='1' then
                state <= Exit_2_DR after S_Odel;
     end if;

when Exit_2_DR => if TMS ='0' then
     state <= shift_DR after S_Odel;
     else
     state <= Update_DR after S_Odel;
     end if;

when Update_DR => if TMS ='0' then
     state <= capture_IR after S_Odel;
     else
     state <= Select_DR_Scan after S_Odel;
     end if;

when Select_IR_Scan => if TMS ='0' then
     state <= Capture_IR after S_Odel;
     else
     state <= Test_Logic_Reset after S_Odel;
     end if;

when Shift_IR => if TMS ='1' then
     state <= Exit_1_IR after S_Odel;
     end if;

when Exit_1_IR => TMS ='0' then
     state <= Pause_IR after S_Odel;
     else
     state <= Update_IR after S_Odel;
     end if;

when Pause_IR => if TMS ='1' then
     state <= Exit_2_IR after S_Odel;
     end if;

when Exit_2_IR => if TMS ='0' then
     state <= Shift_IR after S_Odel;
     else
     state <= Update_IR after S_Odel;
     end if;

When Update_IR => if TMS ='0' then
     state <= Run_Test_Idle after S_Odel;
     else
     state <= select_DR_SCAN after S_Odel;
     end if;

when undefined => state <= Test_Logic_Reset
                            after S_Odel;
```

```
        end case;

        end if;
        end process STATE_DIAGRAM;

        SIG_GEN_1 : block(not TCK'stable and TCK='0'
                          and state /=Undefined)
        begin

        Selectt   <=guarded '1' after Odel when
                  (state=exit_2_IR) or
                  (state= Exit_1_IR) or (state= shift_IR) or
                  (state= Pause_IR) or (state= Run_Test_Idle) or
                  (state= Update_IR) or (state= Capture_IR) or
                  (state= Test_Logic_Reset) else '0' after Odel;
        ClockIR   <=guarded '0' after Odel when tck ='0' and
                  (state= shift_IR or state= Capture_IR) else'1'
                  after Odel;
        UpdateIR <=guarded '0' after Odel when tck ='0' and
                  (state= Update_IR) else'1' after Odel;
        ClockDR   <=guarded '0' after Odel when tck ='0' and
                  (state= Shift_DR or Capture_DR) else'1'
                  after Odel;
        UpdateDR <=guarded '0' after Odel when Tck='0' and
                  (state= Update_DR else '1' after Odel;

        end block SIG_GEN_1;
END tap_controller_behav;
```

## 6.6 INSTRUCTION REGISTER MODEL

The instruction register is represented by the VHDL model whose entity name is "REG_INSTRUCTION". The architecture of this entity "reg_instruction_behaviour" describes the behaviour of the register. Figure 6.4. illustrates the propagation delay in the path and the VHDL model used for defining the instruction register. In addition, it identifies the interface input/output requirement for the Instruction Register.

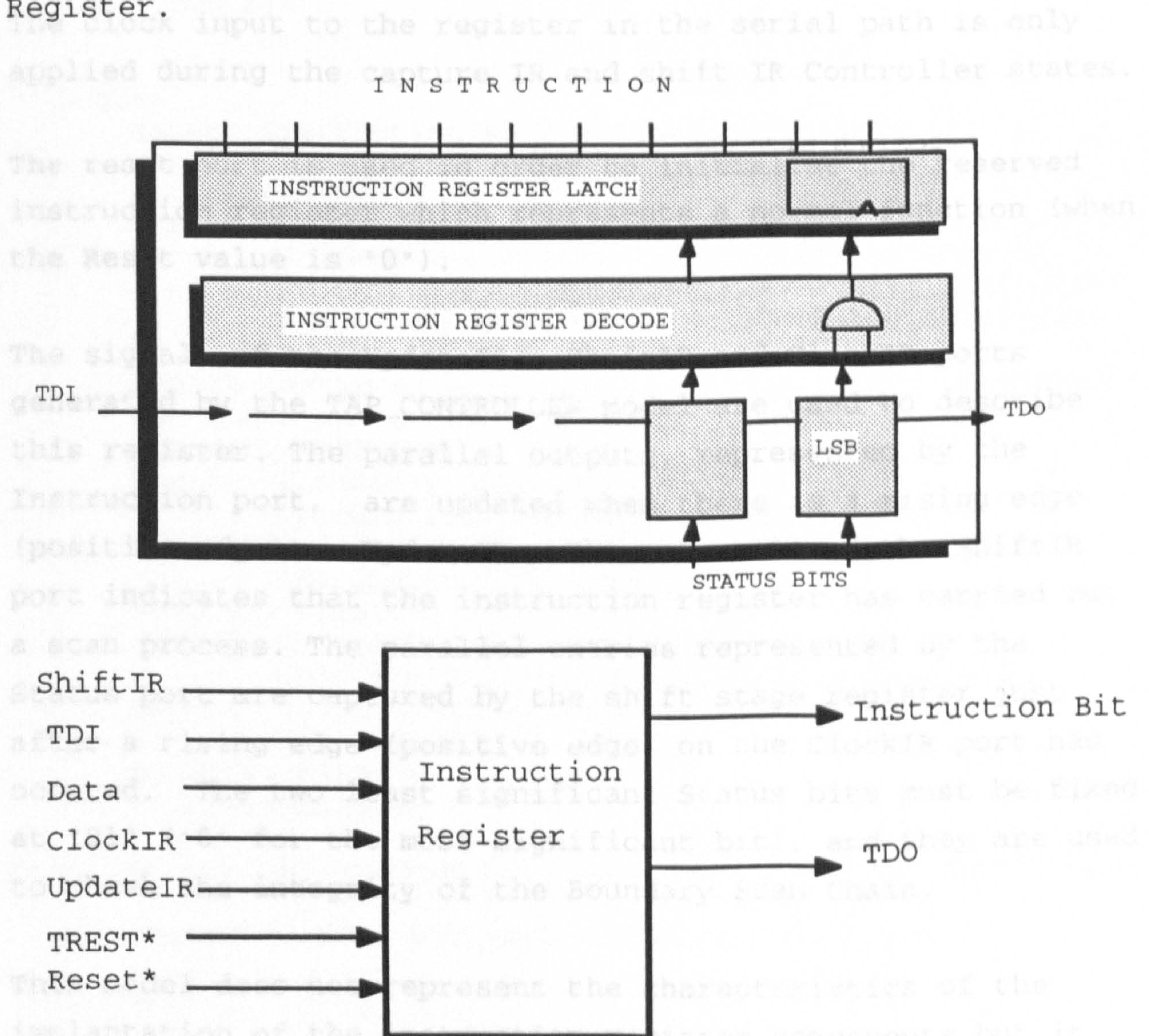



**Figure 6.4 The Instruction Register**

The Instruction Register allows instructions to be entered serially into the test logic during an instruction register scan cycle. The user must enter the instruction through the serial test input (port TDI). The logical value of the serial output for the test data (port TDO) must be equal to the most significant instruction bit (S(S'RIGHT)).

The Instruction bit output is updated at the end of the instruction scan cycle during the update IR Controller state.

The clock input to the register in the serial path is only applied during the capture IR and shift IR Controller states.

The reset port is used in order to initialise the reserved instruction register which represents a normal function (when the Reset value is "0").

The signals of the UpdateIR, ShiftIR and ClockIR ports generated by the TAP_CONTROLLER model are used to describe this register. The parallel outputs, represented by the Instruction port, are updated when there is a rising edge (positive edge) on UpdateIR. The value '1' of the ShiftIR port indicates that the instruction register has carried out a scan process. The parallel entries represented by the Status port are captured by the shift stage register just after a rising edge (positive edge) on the ClockIR port has occured. The two least significant Status bits must be fixed at "01" ('0' for the most significant bit), and they are used to check the integrity of the Boundary Scan Chain.

This model does not represent the characteristics of the implantation of the instruction register components but it provides an accurate functional description.

The timing elements (for shifting, updating and max delay etc) are taken into account by the various parameters shown in figure 6.5 and in VHDL code.

```
entity REG_INSTRUCTION is
   generic (Setup_Shift_Time, Hold_Shift_Time,
            MPluse_Width_Shift, Setup_Update_Time,
            Hold_Update_Time, MPulse_Width_Update,
            Mux_Del, Stage_Shift_Del, Stage_Update_Del:
            Time);
   port(
   Reset,
   ClockIR,
   UpdateIR,
   ShiftIR,
   TDI :in Bit;
   Status : in Bit_vector;
   Instruction : out Bit_vector;
   TDO : out Bit
   );

   begin
       assert Instruction'Right> Instruction'Left
       report "Error in the range of Instruction"
       severity ERROR;
       assert (not (ClockIR'Delayed(Hold_Shift_Time) ='1')) or
              ClockIR'Delayed(Hold_Shift_Time)'Stable or
              (TDI'Stable(Setup_Shift_Time + Hold_Shift_Time
              + Mux_Del) and Status'Stable(Setup_Shift_Time
              + Hold_Shift_Time + Mux_Del))
       report "Setup or Hold Time Violation on TDI or Status
       Terminals in" & "REG_INSTRUCTION entity"
       severity WARNING;
       assert  ClockIR'Stable or ClockIR='1' or
               ClockIR'Delayed(MPulse_Width_Shift)='1'
       report "Pulse Width Failure on ClockIR in
               REG_INSTRUCTION entity"
       severity WARNING;
       assert  UpdateIR'Stable or ClockIR='1' or
               UpdateIR'Delayed(MPulse_Width_Update)='1'
       report "Pulse Width Failure on UpdateIR in
               REG_INSTRUCTION entity"
       severity WARNING;
   end REG_INSTRUCTION;

architecture reg_instruction_behaviour of REG_INSTRUCTION is

       signal S, SP : Bit_vector(Instruction'Range);

   begin

       SHIFT_STAGE : process (ClockIR)
           variable I : integer;
       begin
       if ClockIR='1' then
       if ShiftIR = '0' then
       S <=Status after Stage_Shift_Del+Mux_Del;
       else
       S <= TDI & S(0 to Instruction"Right -1)
           after Stage_Shift_Del + Mux_Del;
       end if;
```

```
        end if;

        end process SHIFT_STAGE;

        assert (not (UpdateIR'Delayed(Hold_Update_Time)='1')) or
                UPdateIR'Delayed(Hold_Update_Time)'Stable or
                S'Stable(Hold_Update_Time)
        report "Setup or Hold Time Failure in the
        REG_INSTRUCTION architecture"
        severity WARNING;

        UPDATE_STAGE : process(UpdateIR, Reset)
        begin

            if Reset ='0' then
            for I in Instruction'Range loop
            SP(I) <= '1' after Stage_Update_Del;
            end loop;
            else
                if UpdateIR ='1' then
                SP <= S after Stage_Update_Del;
                end if;
            end if;
        end process UPDATE_STAGE;
        Instruction <= SP; -- Delta Delay
end reg_instruction_behaviour;
```

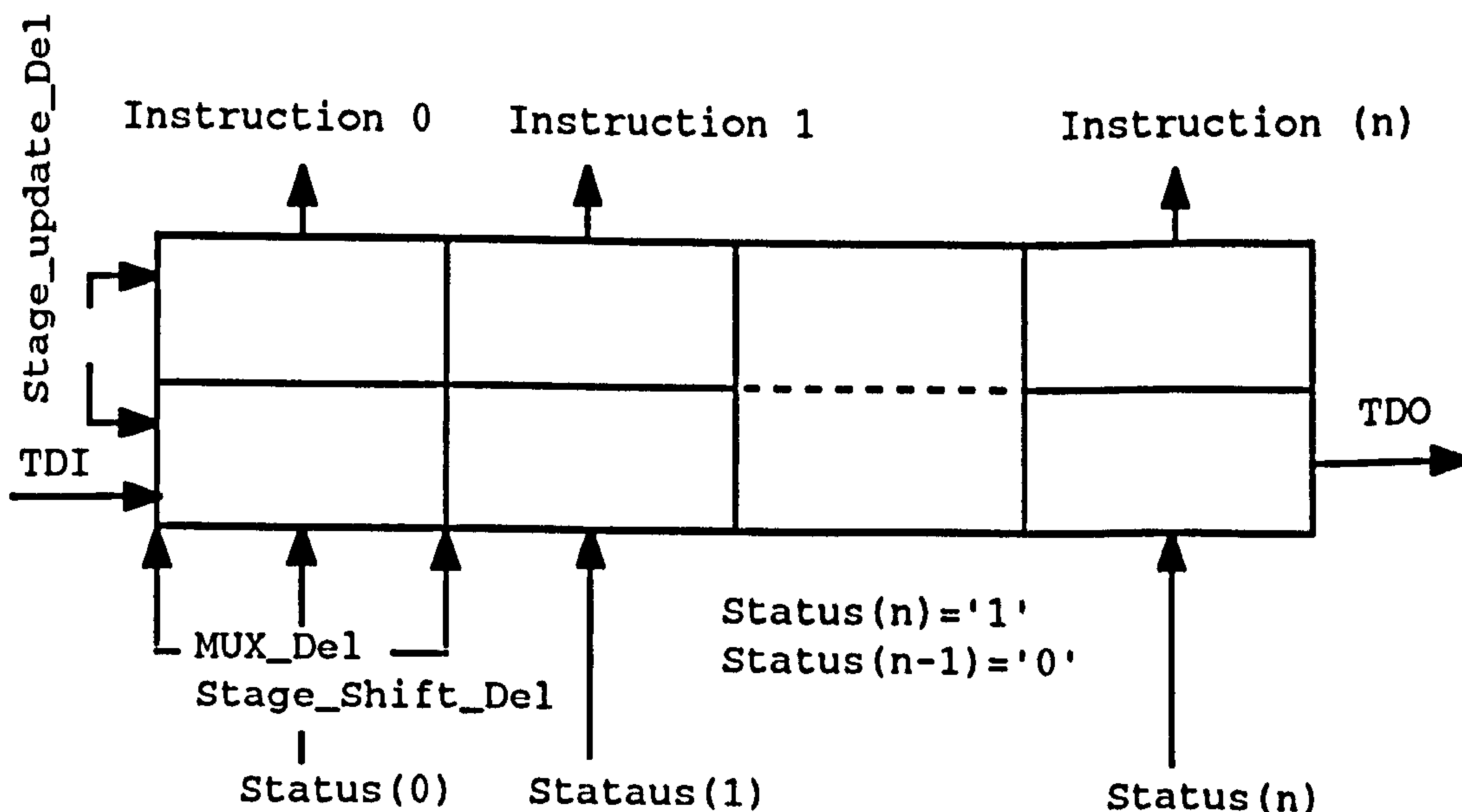



**Figure 6.5 Instruction Register with Timing Parameters**

## 6.7 INSTRUCTION DECODER MODEL

The Black Box model for the Instruction Decoder is described below. Figure 6.6 shows the interface signals to this module. The timing element of the decoder is taken into account by the DEC_DEL parameter.

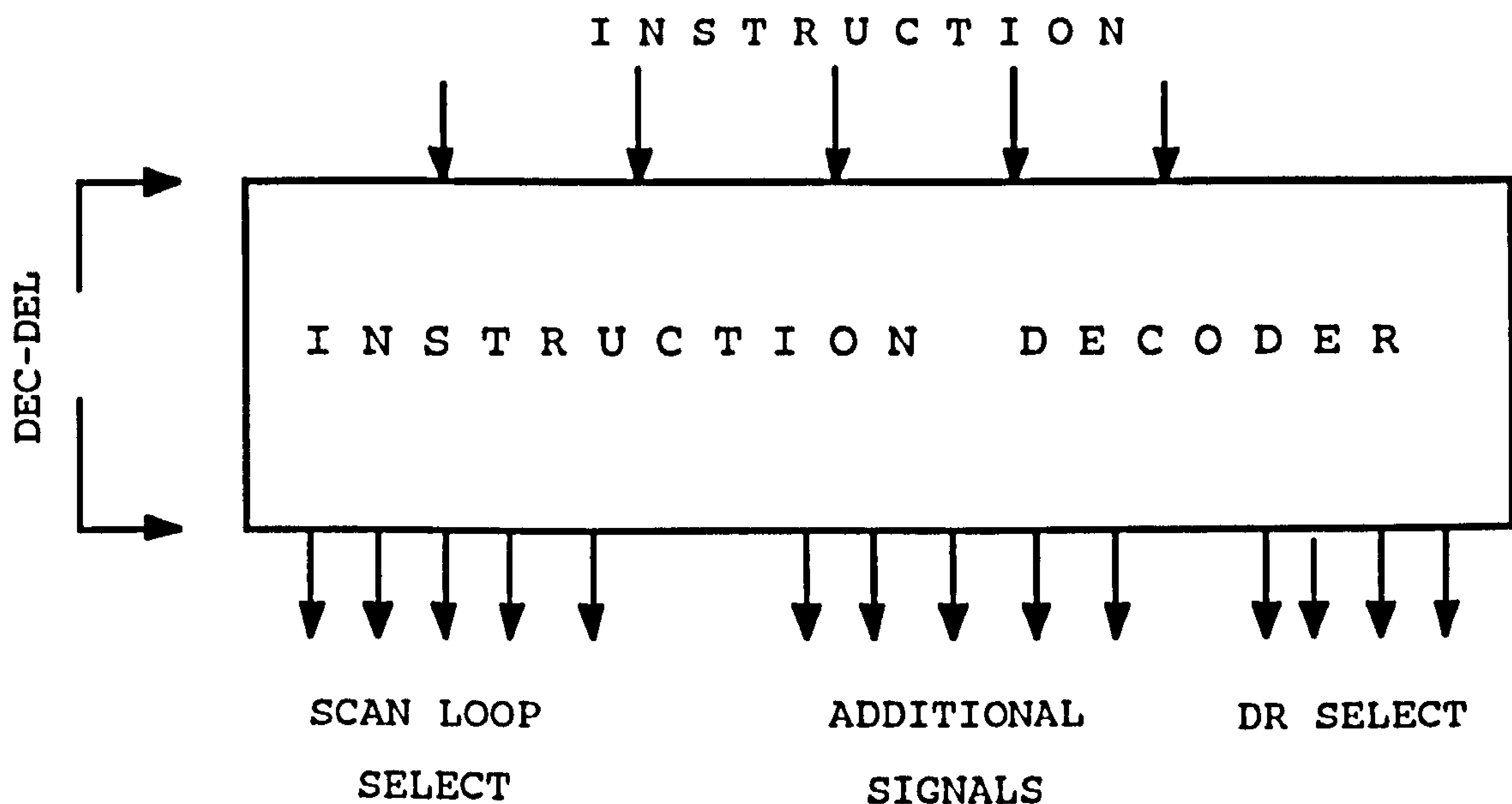



**Figure 6.6 Instruction Decoder**

The decoding logic of instruction register depends essentially on the objective of the test logic. This means that the design of an instruction decoder changes from one chip to another according to the test logic built into that chip. For example, if we have a self-testable integrated circuit, the decoder must generate an additional signal in order to control the triggering of its test, when the appropriate instruction takes place. A flexible way of describing its behaviour is therefore necessary. However, the instruction decoder must have the instruction for decoding as its input, and the test mode together with the selected test data register signals as a minimum at its output (Figure 6.6).

The entity called Instruction_Decoder represents the interface of the VHDL model which describes the function of the decoding logic of the instructions. This is a black box model which receives the input and provides the output to its architecture (*instruction_decoder_ behaviour*). Its behaviour is described at a high level of abstraction and is based on the *generic* type parameters provided by the user.

With this strategy one can avoid the problem of relying on the use of a particular structure. This model is described as follows:

```
entity INSTRUCTION_DECODER is
   generic (Instruction_Set: Bit_Vector;
           DR_Select_Set, Test_Mode_set,
           Additional_Signals_set: Bit_vector;
           open_check : Boolean;
           DEC_DEL : Time);
   port(
       Instruction : in Bit_Vector;
       DR_Select : out Bit_vector;
       Test_Mode : out Bit_vector;
       Additional_Signals :out Bit_vector
       );
end INSTRUCTION_DECODER;

architecture instruction_decoder_behaviour of
INSTRUCTION_DECODER is

begin

   DECODER : process (instruction)
       variable Number_of_instruction : integer :=
               2**Instruction'Length;
       variable Count:integer :=0;

   begin

   count := 0;
   while count < Number_of_Instructions loop
   if instruction = instruction_set(Instruction'Length * count
       to Instruction'Length * (count+1)-1) then
       DR_select <=DR_Select_Set(DR_select'Length * count to
       DR_select'Length * (count+1)-1) after DEC_DEL;
       Test_Mode <= Test_Mode_Set(Test_Mode'Length * count to
       Test_Mode'Length * (count+1)-1) after DEC_DEL;
       if Open_Check then
           Additional_Signals <=
           Additional_Signals_Set(Additional_Signals'Length *
           count to Additional_Signals'Length * (count+1)-1)
           after DEC_DEL;
       end if;
       exit;
   end if;
       count := count+1;
   end loop;
   assert (count < Number_of_Instructions)
   report "Illegal Instruction"
   severity WARNING;

   end process DECODER;
end instruction_decoder_behaviour;
```

The instruction port represents the current instruction to be decoded. The value of this port must be an element of all the instructions supplied by the user using the parameter Instruction_Set (the number of instructions making up this parameter must be equal to 2 $^{Instruction'Length}$ combinations and each one is of a length equal to Instruction'Length bits).

There are three types of output port for this decoding whose logical values are generated after a delay of DEC_DEL. The port DR_Select represents all the signals for the selection of test data registers designed to support a part of the test logic. The number of these signals is equal to the number of the registers. The corresponding information on the test mode must be propagated using all the signals represented by the Test_Mode port.

In the same way the number of these signals must be equal to the number of test types defined in the design. However, the Additional_Signals port represents all the additional signals, and its aim is to propagate the necessary information for the inclusion of additional test facilities. (for example a self testable circuit) built-in-test in integrated circuits.

The necessary values for these three ports are supplied through the intermediary of the three parameters of a generic type which are respectively DR_Select_Set, Test_Mode_set and Additional_Signals_Set. However, the third port is not generally used. As a result, during the configuration of this component in VHDL, it is possible to use an open port using the "open" clause. Alternatively, a logic '1', Open_Check of the *generic* type, indicates the use of this port for a built-in test support.

## 6.8 BYPASS REGISTER MODEL

The reg_bypass_behaviour is the name of the architecture for the entity REG-BYPASS of the VHDL model which represents a description, at a high level of abstraction, of the function of this register (Figure 6.7).

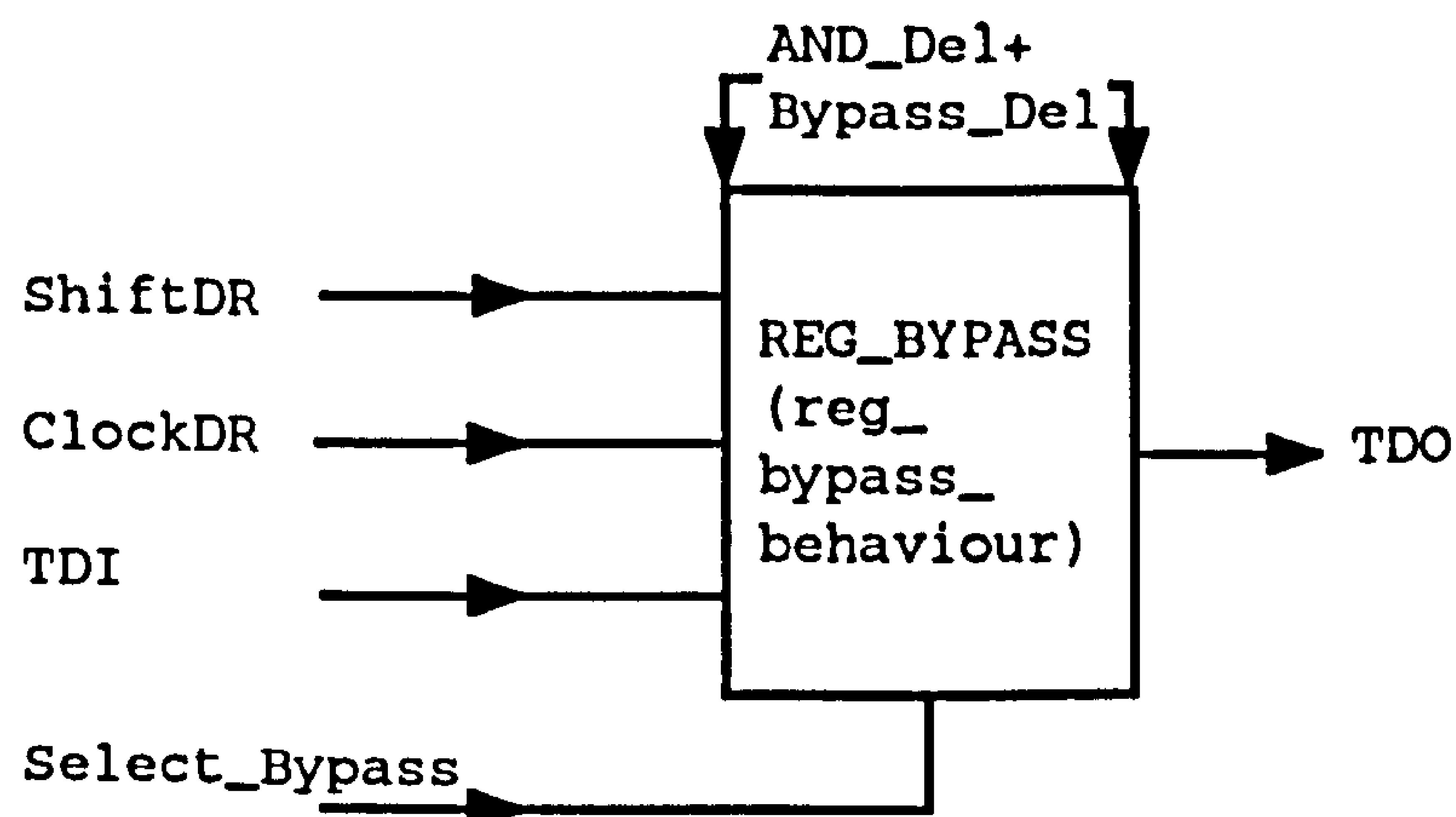


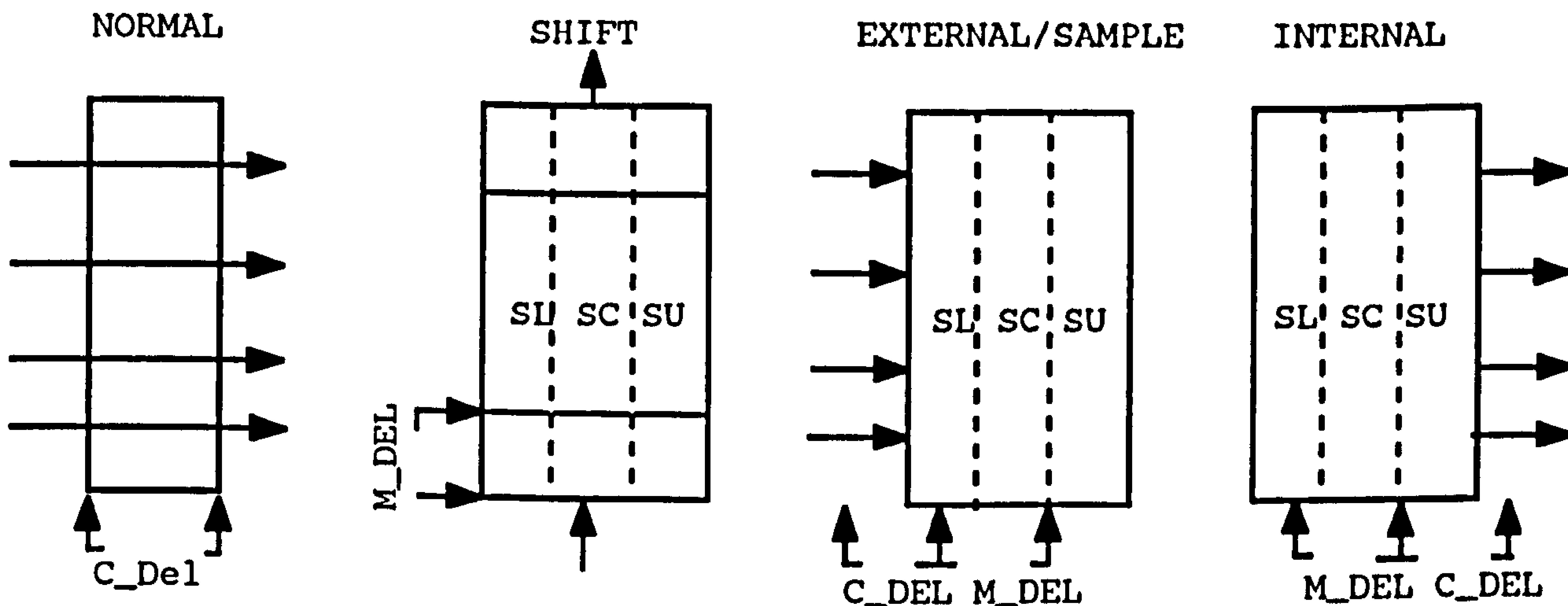

**Figure Bypass Register**

The modelling of the bypass register is simple. Its only function is to short circuit the test data inside an integrated circuit with the boundary scan architecture. When it is selected by the current instruction the value of the Select_Bypass port is equal to '1' or '0'. The value of this signal is generated by the instruction decoder. The *bypass* register must load a logical '0' in its single shift register if a rising edge (positive edge) on the ClockDR takes place and the value of the port ShiftDR (which is generated by "TAP_CONTROLLER") is fixed at '0'. The two ports TDI and TDO act respectively as serial input and output for this register. As a result, the test data is short circuited by these two ports with a delay equals to And_Del + Bypass_Del where And_Del is the delay of the logic port AND.

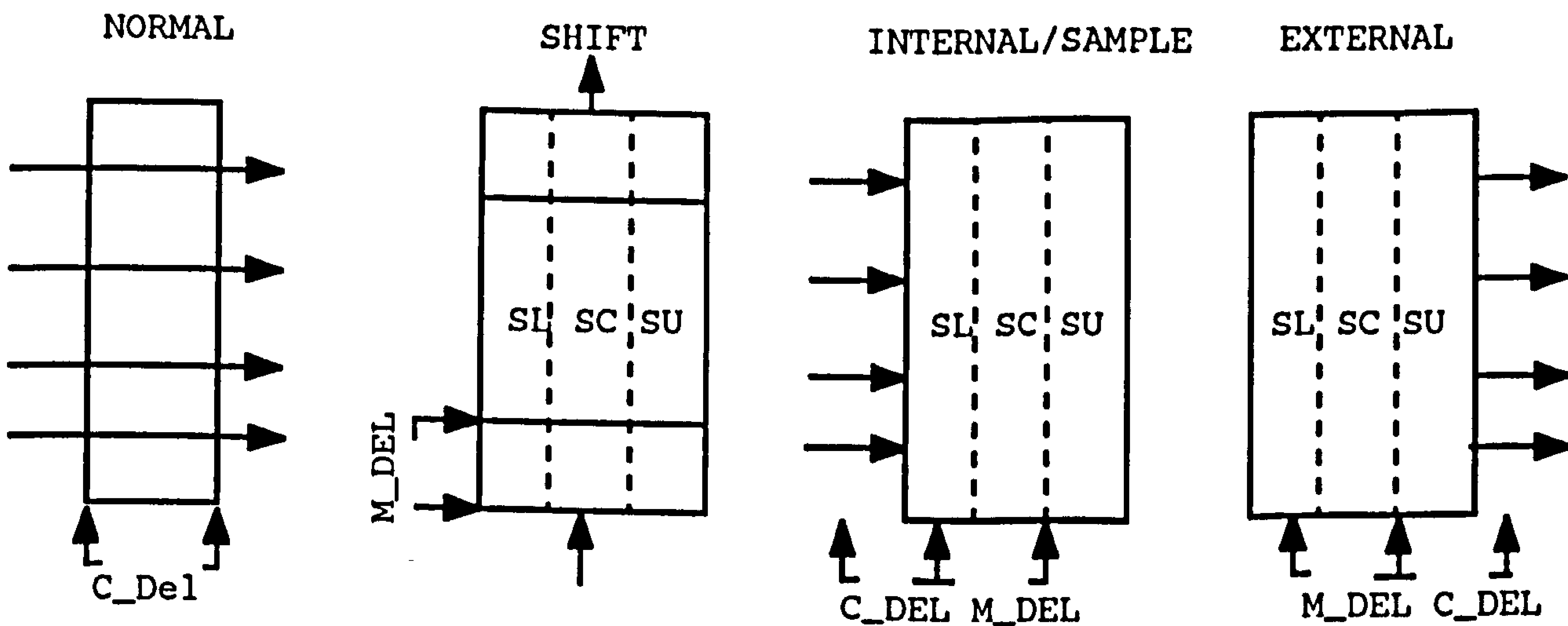
```
entity REG_BYPASS is
      generic (Setup_Time,
      Hold_Time,
      MPulse_Width,
      AND_Del,
      Bypass_Del:Time);
      port(
      Select_Bypass,
      ShiftDR,
      ClockDR,
      TDI: in Bit;
      TDO : out Bit
      );
   begin
      assert Setup_Time <=15ns and Hold_Time <=15ns
      report "15ns is recommended by JTAG as a maximum"
      severity WARNING;
      assert ClockDR'Stable or ClockDR='1' or
          ClockDR'Delayed(MPulse_Width)='1'
      report "Pulse width Time Failure on ClockDR"
      severity WARNING;
end REG_BYPASS;

architecture reg_bypass_behaviour of REG_BYPASS is

begin

   main : block(Select_Bypass='1')

       signal S:Bit;

   begin

       S <=guardrd TDI and ShiftDR after AND_Del;

       assert (not (ClockDR'Delayed(Hold_Time)='1')) or
           ClockDR'Delayed(Hold_Time)'Stable or
       report "Setup or Hold Time Failure on REG_BYPASS
       architecture"
       severity WARNING;

       BYPASS : process (ClockDR)
       begin

       if ClockDR='1' then
           TDO <=S after Bypass_Del;
       end if;

       end process BYPASS;

   end block main;
end reg_bypass_behaviour;
```

## 6.9 BOUNDARY SCAN REGISTER MODEL

This register is composed of cells whose design is based on the test mode of the application logic. Figure 6.8. illustrates the complete interface and the different configurations according to the mode of test.




(a) The input boundary scan register (the cells of the unidirectional and bidirectional input pins and the clock).




(b) The output boundary scan register (the cells of the unidirectional and bidirectional output with three control states).

C_Del : the delay taken by the multiplexer of the test mode control.
M_Del : the delay taken by the multiplexer of capture/shift mode.

**Figure 6.7 Different Configurations Examples of the Boundary Shift Scan Register and Propagation Delays**

The Parallel_Input port represents the parallel inputs of the boundary scan register. The signals at this port are set by connecting the output signals of an integrated circuit to the external logic of the cells of the register, or connecting the signals entering the logic of the integrated circuit via other cells. On the other hand, the Parallel_Output port represents the parallel outputs of this register. The signals at this port are generated directly by the logic of the register cells themselves. Each bidirectional pin is represented by a Parallel_Input signal (the input direction) and a Parallel_Output signal (output direction). Either during the "Test-Logic-Reset" state of the controller, or when the mode controls are set for an internal test, the cells associated with the validation of signals must be initialised to the state which will force the system pins to a high impedance state. The Reset port is therefore included to invalidate the control signals when Reset is equal to logic "0" and the mode controls are set for an internal test.

The signals at the ports UpdateDR, ClockDR and ShiftDR generated by the TAP_CONTROLLER component are used in the design of the logic of the boundary scan register. The UpdateDR port is used to update the boundary scan register stage. The ClockDR is the clock of the shift register (shift and capture the result). The ShiftDR port provides a control signal which indicates whether the process carried out by the boundary scan register is a shift cycle or not. The serial data input and output are represented respectively by the ports TDI and TDO. The Test_MODE port presents the control signals of the boundary scan mode. The value on this port is generated by the instruction decoder.

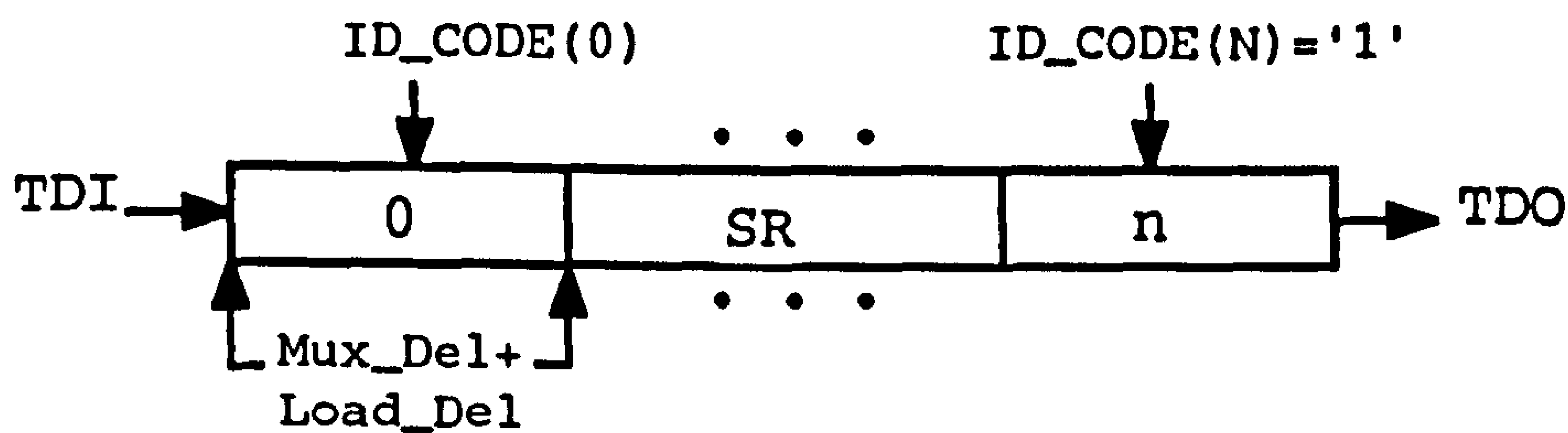
```
Use Work.Declaration.all;
entity REG_BSCN is
          generic (System_Pin_Types : String_vector;
          SUT_Capture, HT_Capture, MDEL, SUT_Update,
          HT_Update, C_DEL, Cap_Del, Upd_Del :Time);
          port (
          Select_Bscan,
          Reset,
          ShiftDR
          ClockDR,
          UpdateDR,
          TDI : in Bit;
          Parallel_Input : in Bit_vector;
          Test_Mode : in Bit_vector(0 to 1);
          Parallel_Output : out Bit_vector;
          TDO : out Bit
          );
  begin

      assert  (not(ClockDR'Delayed(HT_Capture)='1')) or
              ClockDR'Delayed(HT_Capture)'Stable or
              TDI'Stable(SUT_Capture + HT_Capture +MDEL)
      report "Setup or Hold Time Failure on TDI terminal in
      REG_BSCAN entity"
      severity WARNING;
  end REG_BSCAN;

architecture reg_bscan_behaviour of REG_BSCAN is

      signal SL, SC, SU : Bit_vector(Parallel_Input'Range);

  begin

      assert  (not(ClockDR'Delayed(HT_Capture)='1')) or
              ClockDR'Delayed(HT_Capture)'Stable or
              SL'Stable(SUT_Capture + HT_Capture +MDEL)
      report "Setup or Hold Time Failure on SL signal in" &
      "REG_BSCAN entity"
      severity WARNING;

      CAPTURE_SHIFT: process (ClockDR)

      begin

          if Select_Bscan='1' and ClockDR ='1' then

              case ShiftDR is

          when '0' => SC <= TDI & SC(0 to
          Parallel_Input'Right -1) after Cap_Del;

              end case;
          end if;

      end process CAPTURE_SHIFT;
```

```
SER_OUT : TDO <= SC(SC'Right); --Delta Delay

assert  (not(ClockDR'Delayed(HT_Capture)='1')) or
        ClockDR'Delayed(HT_Capture)'Stable or
        SC'Stable(SUT_Capture + HT_Update)
report "Setup or Hold Time Violation on SC signal in
REG_BSCAN entity" & "architecture"
severity WARNING;

UPDATE : process (UpdateDR)
begin

if Select_Bscan='1' and UpdateDR ='1' then

    for I in Parallel_Input'Range loop

        case System_Pin_Types(I) is

    when 'K' => null; -- the shift cell
    when 'C' => if Reset = '0' then
                    SU(I) <='0' after Upd_Del;
                else
                    SU(I) <= SC(I) after Upd_Del;
                end if;
    when others => SU(I) <= SC(I) after Upd_Del;
        end case;
    end loop;
end if;
end process UPDATE;

OUTPUT_MUX : process (SL, SU, Parallel_Input,
Test_Mode)

begin
    if Select_Bscan ='1' then
    assert Test_Mode /= "11"
    report "Undefined Test Mode"
    severity WARNING;

    for I in Parallel_Input'Range loop
        case System_Pin_Types(I) is
    when 'K' => SL(I) <= Parallel_Input(I);
    when 'I' | 'B' =>  case test mode is
          when "00" | "01" =>SL(I)<=parallel_Input(I)
                             after C_DEL;
          when "10" =>SL(I)<= SU(I) after C_DEL;
          When "11" => null;
                             end case;
    when others=>            case Test_Mode is
          when "00" | "10" =>SL(I)<=parallel_Input(I)
                             after C_DEL;
          when "01" =>SL(I)<= SU(I) after C_DEL;
          When "11" => null;
                             end case;
        end case;
    end loop;
```

```
            case Test_Mode is
        when "00" => null;
        when "01" => for I in Parallel_Input'Range loop
                            case System_Pin_Types(I) is
            when 'O'| 'P'| 'T'| "C" => Parallel_Output(I)
            <= SL(I);
            when others => null;
                            end case;
               end loop;
        when "10" => for I in Parallel_Input'Range loop
               case System_Pin_Types(I) is
            when 'I' |'B'|'K' => Parallel_Output(I)
            <= SL(I);
            when others =>null;
               end case;
        end loop;
        when "11"=> null;
            end case
        else -- normal
        for I in Parallel_Input'Range loop
            if System_Pin_Types(I) ='K' then
               Parallel_Output(I) <=Paralle_Input(I);
            else
               Parallel_Output(I) <=Paralle_Input(I) after
               C_DEL;
            end if;
        end loop;
    end if;

  end process OUTPUT_MUX;

end reg_bscan_behaviour;
```

## 6.10 IDENTIFICATION REGISTER MODEL

The REG_IDENT_BEHAVIOUR is the name of the architecture for the entity REG_IDENT of the VHDL model which represents a description for the function of this register at a high level of abstraction. Figure 6.9 shows the interface signals to this module. The timing elements of the decoder are taken into account by the Mux_del and Load_Del delay parameters.




Mux_Del : The propagation delay taken by the Multiplexer.

Load_Del: The propagation delay taken by the shift register.

**Figure 6.9 The Identification Register Model**

The ID_code port represents the parallel input to this register which makes it possible to load an identification code. This register has a single path based on a shift register which does not have parallel outputs and as a result the interface of this entity does not contain the port UpdateDR. The two other ports ShiftDR and ClockDR, whose signals are generated by the component TAP_CONTROLLER, are used for the UpdateDR port design. The rising edge on ClockDR makes it possible to capture the identification code of the module. When the logical value of ShiftDR is '1' the register is in a scan mode. All the operations of this register are controlled by the value of the Select_Ident port which indicates whether this register is selected by the current instruction. The TDI and TDO ports are the serial input and the output ports.

```
entity REG_IDENT is
   generic (Setup_Time, Hold_Time,MPulse_Width, Mux_Del,
   Load_Del: Time);
   port (
   Select_Ident,
   ShiftDR,
   ClockDR,
   TDI: in Bit;
    ID_Code: in Bit_vector;
   TDO: out Bit
   );

begin
   assert ClockDR'Stable or ClockDR='1' or
       ClockDR'Delayed(MPulse_Width)='1'
   report "Pulse width Failure on ClockDR in REG_IDENT entity"
   sevirity WARNING;
   assert Setup_Time <=15ns and Hold_Time <=15ns
   report "these two times are recommended by JTAG to be less
       than 15ns"
   severity WARNING;
   assert (not(ClockDR'Delayed(Hold_Time)='1')) or
       ClockDR'Delayed(Hold_Time)'Stable or
       TDI'Stable(Setup(Setup_Time+Hold_Time+Mux_Del)
   report "Setup or Hold Time Violation on TDI terminal"
   severity WARNING;

end REG_IDENT;

architecture reg_ident_behaviour of REG_IDENT is
       signal SR: Bit_vector(ID_Code'Range);

   begin

       LOAD_SHIFT: process (ClockDR)
           variable I: integer;

       begin

       if Select_Ident='1' and ClockDR= '1' then
           if ShiftDR='0' then
            SR <=ID_Code after Mux_Del=Load Del; -- load
           else
            SR <=TDI & SR(0 to ID_CODE'Right - 1)
                 after Mux_Del + Load_Del; -- Shift
           end if;
       end if;

   end process LOAD_SHIFT;

   TDO <=SR(SR'Right); -- Delta Delay

end reg_ident_behaviour;
```

## 6.11 MULTIPLEXER MUX_1 MODEL

The behaviour of the multiplexer MUX_1 is described by the VHDL model whose entity is called MUX_1 and the architecture by mux_1_behaviour. Figure 6.10 illustrates the design of this multiplexer. Mux_Del represents the delay at port TDO_Test_Data_Registers.

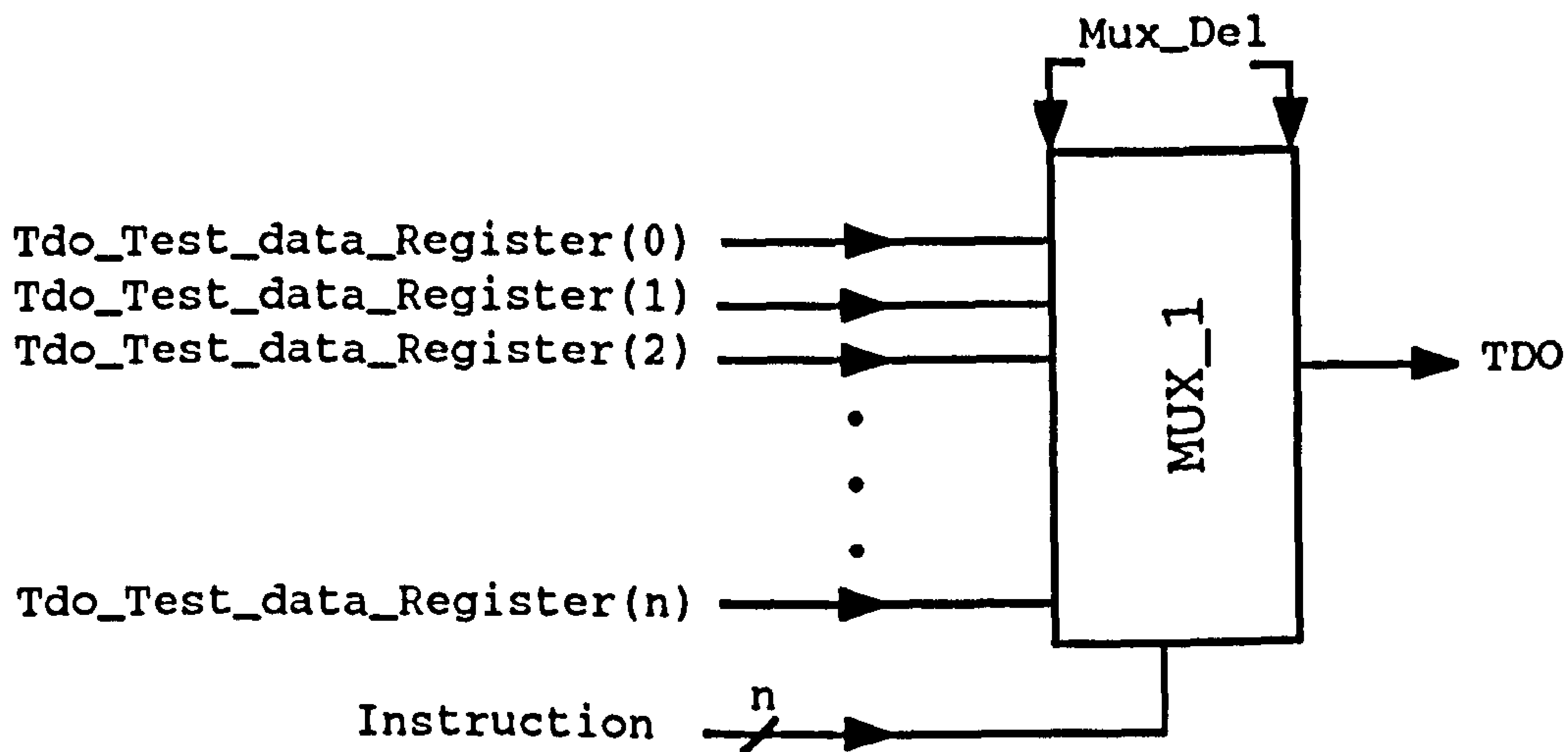



**Figure 6.10 The MUX_1 Multiplexer Model**

```
use work.declaration.all;

entity mux_1 IS

        generic (
        instruction_set :bit_vector;
        tdo_test_data_registers_set : integer;
        mux_del :time);

        port (
        tdo_test_data_registers: in bit_vector;
        instruction : in bit_vector;
        tdo: out bit);

end mux_1;

architecture mux_1_behav of mux_1 is

         signal ind:integer:=0;
         begin
                multiplexer: process (instruction,
tdo_test_data_registers,ind)
                variable number_of_instructions:integer
                :=2**instruction'length;
                variable count:integer :=0;
```

```
                begin

                if instruction'Stable then
                tdo <=transport tdo_test_data_registers(ind)
                after 5ns;
                else
                count :=0;
                while count<number_of_instructions loop
                    if instruction =
                     instruction_set(instruction'length
                     *count to instruction'length*
                     (count+1)-1) then
                    ind <=tdo_test_data_registers_set(count);
                          tdo <=tdo_test_data_registers(ind)
                          after 5ns;
                          exit;
                    end if;
                          count:= count+1;
                    end loop;
                        assert count < number_of_instructions
                        report "illegale instruction"
                        severity WARNING;
                        end if;

               end process multiplexer;

   end mux_1_behav;
```

In order not to impose a particular instruction, the parameters of *generic* type are used to pass the necessary information. It covers all combinations of the $2^{Instruction'Length}$ (the Instruction_Set parameter) and the TDO_Test_Data_Registers_Set which represents all the corresponding serial outputs contained in Instruction_Set.

## 6.12 MULTIPLEXER MUX_2 MODEL

Figure 6.11 illustrates a design for this multiplexer. If the value of the Selectt port is equal to '1' the serial output of the instruction register (TDO_Instruction port) is multiplexed and its value is assigned to the TDO port with a delay of Mux_Del. Otherwise the value of the Selectt port is that of the multiplexed serial output (the Tdo_Test_Data_Registers port) and the current instruction including the test data registers (MUX_1).

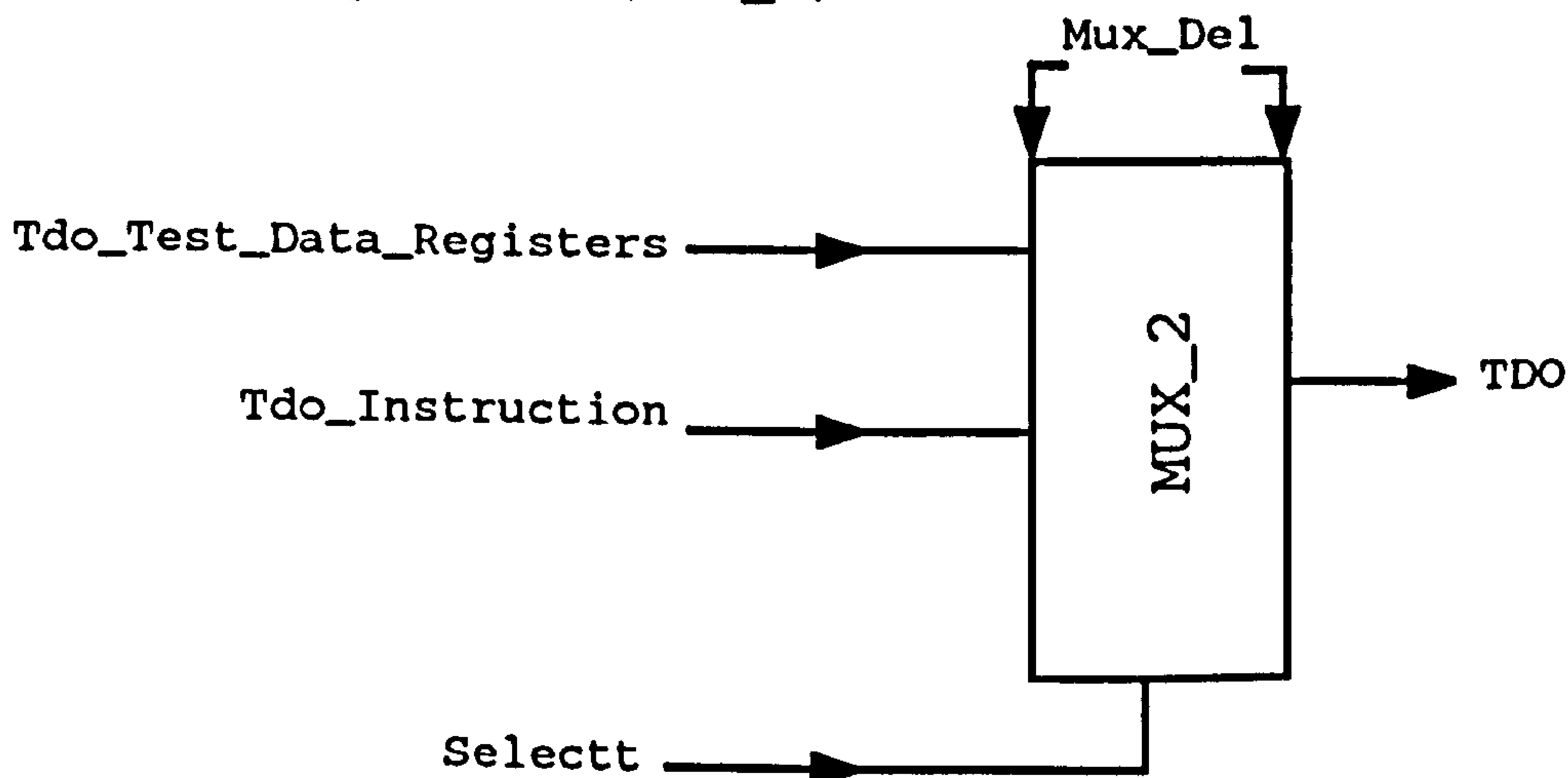



**Figure 6.11 MUX_2 Model**

The entity of this model is called MUX_2 and its behaviour is described by the mux_2_behaviour architecture.

```
entity mux_2 is
        generic (mux_del : time);
        port (
        tdo_test_data_registers,
        tdo_instruction,
        selectt : in bit;
        tdo : out bit
        );
end mux_2;

architecture mux_2_behav of mux_2 is
        begin
                multiplexer :
                process (selectt,
                tdo_test_data_registers,tdo_instruction)
                begin
                        case selectt is
                        when '0' =>tdo<=transport
                        tdo_test_data_registers
                        after mux_del;
                        when '1' =>tdo<=transport
```

```
                    tdo_instruction
                    after mux_del;
                    when others =>putline ("non test
                                              mode");
                    end case;
            end process multiplexer;
    end mux_2_behav;
```

## 6.13 SERIAL OUTPUT BUFFER MODEL (TDO)

The changes on the serial output represented by the TDO port, are delayed by the inclusion of a D type flip flop. This flip flop that is synchronised to the falling edge (negative edge) of TCK in the buffer of the TDO output. The TDO port is validated by the signal of the Enablee port when it is equal to '1'. The Input port is the input for the D flip flop (Figure 6.12).

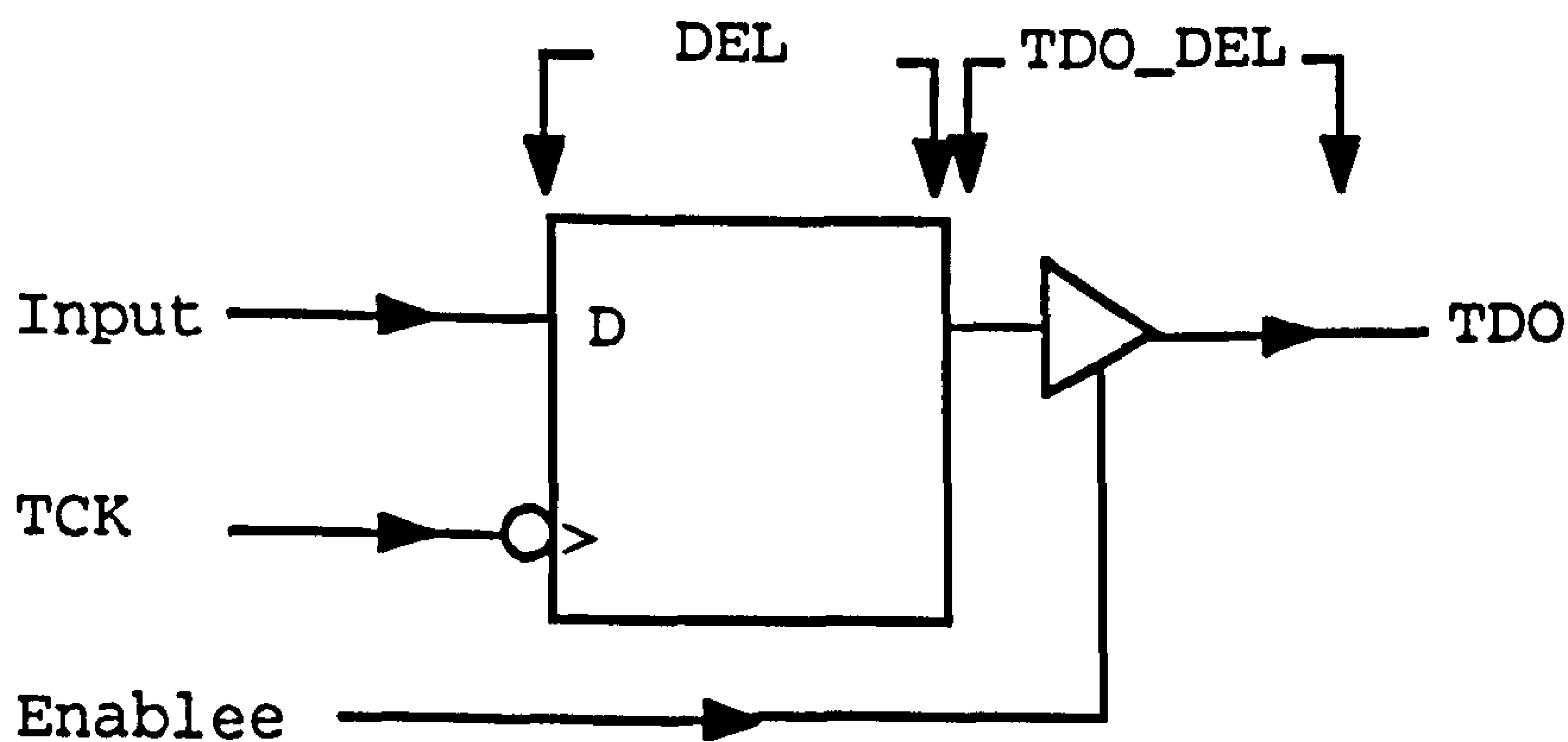



**Figure 6.12 Serial Output Buffer**

```
use Work.Declaration.all;
entity TDO_BUFFER is
      generic (Setup_Time, Hold_Time, Min_Pulse_Width,
      Del, TDO_Del : Time);
      port (
      TCK,
      Enablee,
      Input : in Bit;
      TDO : out Tristate
      );
  begin
      assert (not(TCK'Delayed(Hold_Time)='0')) or
          TCK'Delayed(Hold_Time)'Stable or
          Input'Stable(Setup_Time + Hold_Time)
      report "Setup or Hold Time Violation on Input terminal
      in" & "TDO_BUFFER entity"
      severity WARNING;
  end TDO_BUFFER;
```

```
architecture tdo_buffer_behaviour of TDO_BUFFER is

  begin

      OUTPUT_BUFFER : process (TCK)

      begin

      if TCK ='0' then
          if enablee ='1' then
              case Input is
          when '0'=> TDO <= '0' after DEL + TDO_DEL;
          when '1'=> TDO <= '1' after DEL + TDO_DEL;
              end case;
          else
          TDO <='Z' after TDO_DEL;
          end if;

      end if;

      end process OUTPUT_BUFFER;

end tdo_buffer_behaviour;
```

## 6.14 CONCLUSIONS

All the components of the JTAG Boundary Scan Architecture were successfully modelled in VHDL.

The *package* JTAG_STANDARD, was developed to provide the necessary declarations for the JTAG components separately from that of the application circuits in VHDL, as shown in Appendix 6A. This *package* will normally reside in the library of the design environment. The entity of a new design must therefore include this package in its context of analysis by calling it up, using the clauses "library" and "use". The information contained in this *package* is in fact transparent to the user who can use it according to the need without knowing the intrinsic structure. The user can add new declarations of defined components to support their test development.

Appendix 6B describes the package 'declaration' which contains all the timing elements used by the JTAG architecture.

The VHDL high level model of the JTAG architecture was simulated and its results are shown in Appendix 6C. The architecture model was then tested with an application logic, a 2 bit adder, to demonstrate the connectivity and the validity of its operation as shown in Appendix 6D. The simulation results are also listed in Appendix 6D.

**REFRENCES**

[BREU 80] M.A. Breuer and A.D. Friedman, Functional Level Primitives in Test Generation, IEEE Trans. on Computers, Volume C29, No.3, pp 223-235, March 1980.

[DAVE 89] U.J. Dave and J.H. Patel, A Functional-Level Test Generation Methodology using Two-Level Representations. 26th ACM/IEEE Design Automation Conference. pp 722-724 USA, Jun 1989.

[ABAD 85] M.S. Abadir and H.K. Reghbati, Functional Specification and Testing of Logic Circuits. Comp & Maths with Appls, Volume C-11, No.12, pp 1143-1153, 1985.

[AKER 78] A.B. Akers. Functional Testing with Binary Decision Diagrams. Proceedings of the 8th International Conference on Fault Tolerant Computing, pp.82-92, June 1978.

[MITCH 84] Steinberg & Mitchell. A knowledge based approach to VLSI CAD Redesign System. Proceedings ACM IEEE 21 Design Automation Conference 1984.

[WILL 90] John T Willey & Gary F Gordon. The VHDL Language for Design Automation. Electronic Product Design, February 1990.

[DETT 89] Dettmer R. JTAG Setting the Standard for Boundary-Scan Testing. IEEE Review. February 1989.

[COEL 89] David R. Coelho. The VHDL Handbook, Kluwer Academic Publishers 1989.

[LIPS 89] VHDL : Hardware Description and Design, Kluwer Academic Publishers 1989.

[IEEE 88] IEEE Standard VHDL Language Reference Manual - STD 1076, 1987. (New York: IEEE:1988).

[JTAG 90] IEEE Standard 1149.1, 'A Standard Test Access Port and Boundary Scan Architecture' May 1990.

# CHAPTER 7
# THE PARSING AND INSERTION ALGORITHM

## 7.0 INTRODUCTION

It is becoming increasingly common for electronic circuits to be designed, at least partially, by computer aided synthesis or by using semi-automated insertion tools. This naturally has an impact on the design cycle time .

This chapter describes the design, development and operation of a high level parsing and insertion algorithm that will enable the integration of a Boundary Scan Test Architecture into an ASIC design in a semi-automatic way.

The methodology applied for developing the algorithm is described. Examples demonstrating the operation of the proposed algorithm are given. The main features and limitations of the proposed tool are discussed.

## 7.1 AN OVERVIEW OF THE HIGH LEVEL PARSING AND INSERTION ALGORITHM

Once the application logic (the ASIC) has been described in VHDL (or has been converted into a VHDL source code or a VHDL netlist), the next step is then to add the parameterised Boundary Scan Test Architecture to the design.

However, in order to add the correct type of Boundary Scan Cells to the design, it is important to identify the type of the input/output terminal cells that have been used in the application logic. It is also important to note that whatever the design style of the application logic, in terms of behavioural, structural or data flow, the 'entity' definition of the design is always the same in VHDL. This is one of the attractive features of the VHDL Language.[IEEE 88]

A high-level procedural algorithm has been developed in a standard high level language 'C'.[SCHI 88] It consists of two phases: a *Parsing Phase*, and an *Insertion Phase.*

The parsing phase deals with identifying the mode of the input/output terminals as defined in the entity description of the application logic.

Once the parser is executed, an output file *<design.pin>* will be generated in ASCII format. It contains a list of the input/output names, types and modes. This file will form one of the required inputs before the insertion phase. The parser will be referred to as the VCP parser (VHDL- C- Parser) in the subsequent sections.

The insertion phase deals with identifying the modes of the input/output terminals in the <design.pin> file. It then attaches the appropriate Boundary Scan Cells to the ASIC to form Boundary Loop 0 which is the Boundary Scan Register around the periphery of the ASIC. The next step the algorithm performs is to integrate the rest of the preprocessed generic Boundary Scan Architecture- the TAP, Instruction Register and decoder, Bypass register, Identification Register (Optional) and the Multiplexers- into the ASIC design. The newly created entity will then be contained in the <name_jtag> output file.

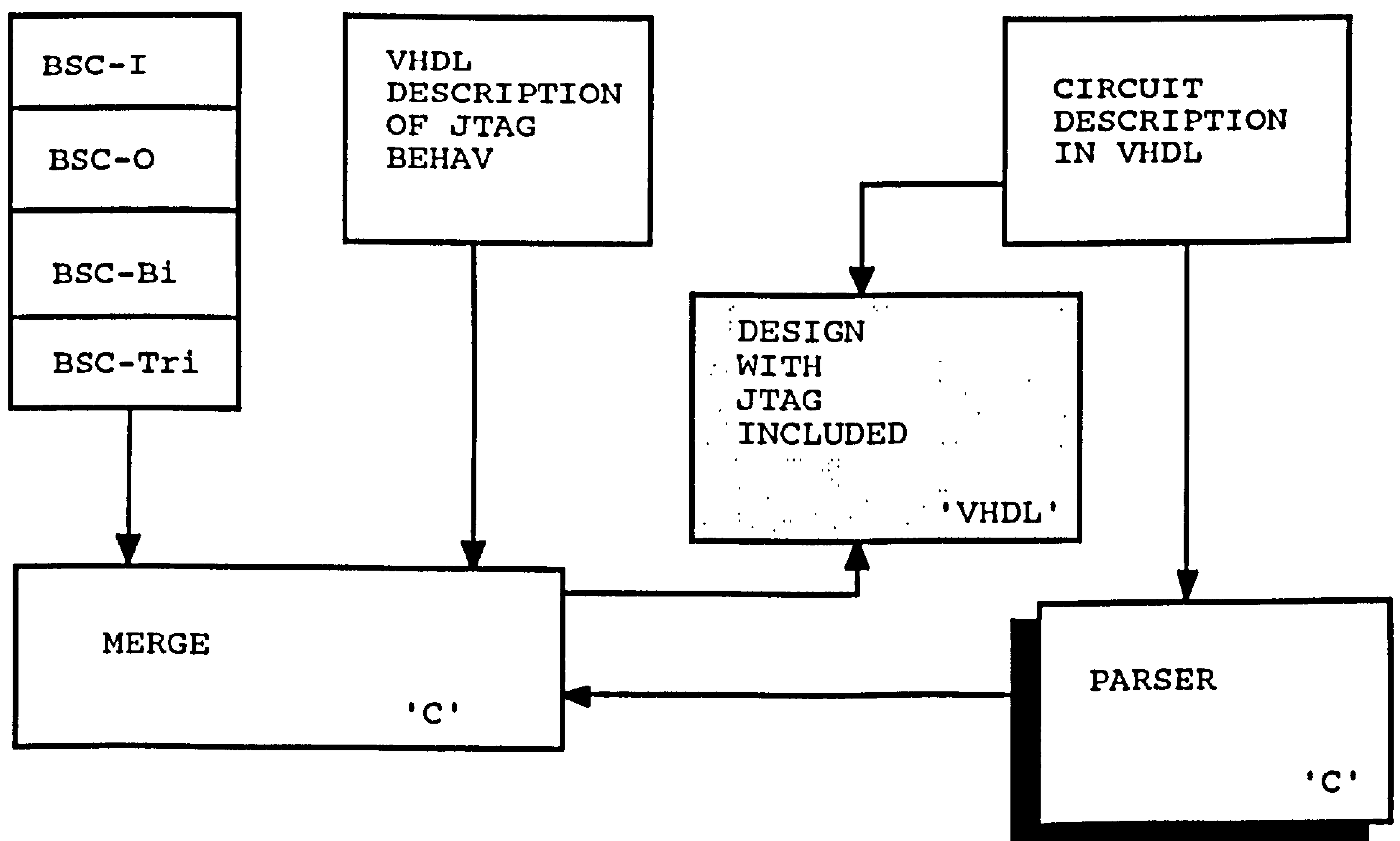



Figure 7.1 A Generic Block Diagram of The VCP Parser

## 7.2 THE VCP PARSER OPERATION

The VCP was developed using the 'C' high level language. The main role of this parser is to search for the word 'PORT' as a keyword within the VHDL entity description of the design. It identifies the input/output list including name, mode, type and the order in which they appear within the entity description.

The parser then outputs an ASCII file, which contains a list of the port names in the same order they appeared in the VHDL description of the design, together with the port types.

## 7.3 THE DEVELOPMENT OF VCP-VERSION 1

An initial development of the parsing algorithm has resulted in the first version of VCP. The VCP basic algorithm is described below:

a. Search for the Key Word PORT within the Entity description of the design.

b. Identify the port list in the entity structure of the VHDL file (the design).

c. Identify the modes of the Input/Output terminals used (IN, OUT, BIDIRECTIONAL, TRISTATE)

d. Identify the order in which the I/Os appear in port list.

e. Output an ASCII file which contains a list of the I/Os *<vhdl_port_file>* presented in the same order they appeared in the source file of the design.

The basic operation of the proposed tool is described below using data flow diagrams.[WARD 86] The Context diagram represents the main process (the parent level) and identifies the primary inputs and outputs of the algorithm.

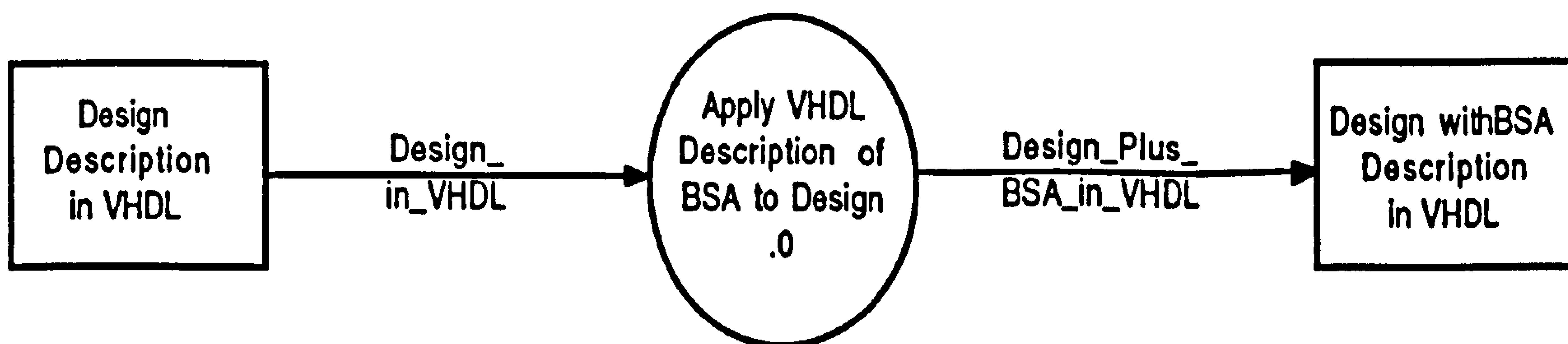



**Figure 7.2 C/DFD Context Diagram of VCP-Version 1**

Level 0 (the first child) is represented by 7 processes and a store. The analysis was carried out using the MCASE™ tools from Mentor Graphics running on an Apollo workstation.[MCAS7]

**Figure 7.3 Data Flow Diagram of VCP-Version 1**

## 7.4 EXAMPLE OF THE OPERATION OF VCP-VERSION 1

This version was tried out with a number of simple examples. The parser can be invoked from the shell on a UNIX based workstation or from the operating prompt in DOS on a PC by entering

parser <file.hdl>

where <file.hdl> is the design file to be parsed.

The parser starts by looking for the **ENTITY** clause and once it has found it, it will then start looking for the **PORT** clause. [LIPS89] As soon as the PORT clause is found, the parser lists the inputs as

**-<i>-** and the outputs as **+<o>+**

where -<i>- represents the input terminal name and +<o>+ represents the output terminal name.

The parser then writes the list of ports to a *<vhdl_port_file>*.

EXAMPLE 1

The parser is operating with a 2 input AND gate VHDL file described using the ViewLogic Workview™ VHDL environment running on a PC-386 :

parser and.hdl

```
entity and_gate is
 port (a,b : in vlbit;
         c : out vlbit);
end and_gate;

Architecture behav of and_gate is
 begin
 c <= a and b;
 end behav;
```

The listing of the vhdl_port_file is as follows:

```
LIST
INPUTS:
a
b
OUTPUTS:
c
```

The results are correct when compared with the source file.

EXAMPLE 2

The parser is operating with a D type flip flop VHDL file described using the Mentor Graphics System 1076™ Version 7 VHDL environment running on an Apollo/HP 400:

parser DFF.hdl

```
entity DFF is
 port  (
      d , clk  :    in bit;
      q        :    out bit
      );
end DFF;

Architecture behav of DFF is
 begin
    if clk='1' then
       q <= d after 1ns;
    end if;
 end;
end behav;
```

The listing of the vhdl_port_file is as follows:

```
LIST
INPUTS:
d
clk
OUTPUTS:
q
```

The results are correct when compared with the source file.

EXAMPLE 3

Consider the case where the DFF.HDL file described previously was hypothetically modified to represent an electronic circuit called test, and execute the parser as follows:

parser test.hdl

```
entity test is

 port  (
       d        :     inout bit;
       clk      :     in bit_vector  (2  to  23);
       q        :     out bit;
       );
constant  Tp_Q_test  :  time:=  3.1ns;

end test;

Architecture behav of test is
 begin
    if clk='1' then
       q <= d after Tp_Q_test;
    end if;
 end;
end behav;
```

The listing of the vhdl_port_file is as follows:

```
LIST
INPUTS:
d
OUTPUTS:
clk
```

As it can be seen from the above listing file the results are incorrect, as the d signal was listed as an input and the clk signal was listed as an output. In addition, the parser was not able to deal with the inout mode and the bit_vector type of entity declarations. The above example has highlighted some of the limitations of this version of VCP. The full limiting factors are described in the following section.

## 7.5 LIMITATION OF VCP-VERSION 1

There are a number of operational limitations which have led to the development of version-2. These limitations are confined to the entity declaration of the VHDL source code of the design and include:

1) The parser was not able to deal with separating characters such as ',' or '-'.
2) The parser was not able to deal with bus structures such as bit_vector.
3) There was no Syntax Checking facility. The intelligence of the parser is limited. It reads every thing after the key word 'port' from the beginning of the bracket till the end of the bracket.
4) It was very strict in terms of the order the input/output terminal modes are defined within the entity declarative part.Therefore a mixture of port declaration modes is not allowed. For example, defining an input mode followed by an output and another input mode is not possible.

```
port    (
        a       :       in bit;
        q       :       out bit;
        b       :       in bit;
        );
```

5) It did not output any information associated with the port list other than the names, and the order they appeared in the design.
6) It did not support port modes **inout** and **buffer.**

7) It did not have a facility to handle comments within the source file.

However, the development of this initial version has provided an insight into the development version 2.

## 7.6 THE DEVELOPMENT OF VCP-VERSION 2

The VCP version 2 was developed to overcome the shortcomings of the 1st version. In addition to the main requirements identified earlier, the additional features that the new version of VCP should process, include the following:

1) It must be portable, and operate in both PC and workstation environments.

2) It must operate with both the IEEE 1076 standard and with subsets of the VHDL implementation offered by the major CAE vendors.

3) It must have a built in syntax checker in order to handle the randomness of the way the port modes are defined, and to extract the input/output information only.

4) It must be able to handle comments defined in the source file.

5) It must handle all port modes including inout and buffer.

6) It must be fast to extract the input/output list with a compilation time of less than 10 seconds running on a PC-386 and extracting a nominal number of input/outputs of 256.

The algorithm reads a VHDL source file (the design) and extracts the top level input/output terminals from the design. It then generates an output file called **<file_name.PIN>**. The <file_name.PIN> file contains a listing of all the I/O names, modes, types, and buses including their widths.
A bubble diagram structure (similar to a state machine) is shown below.

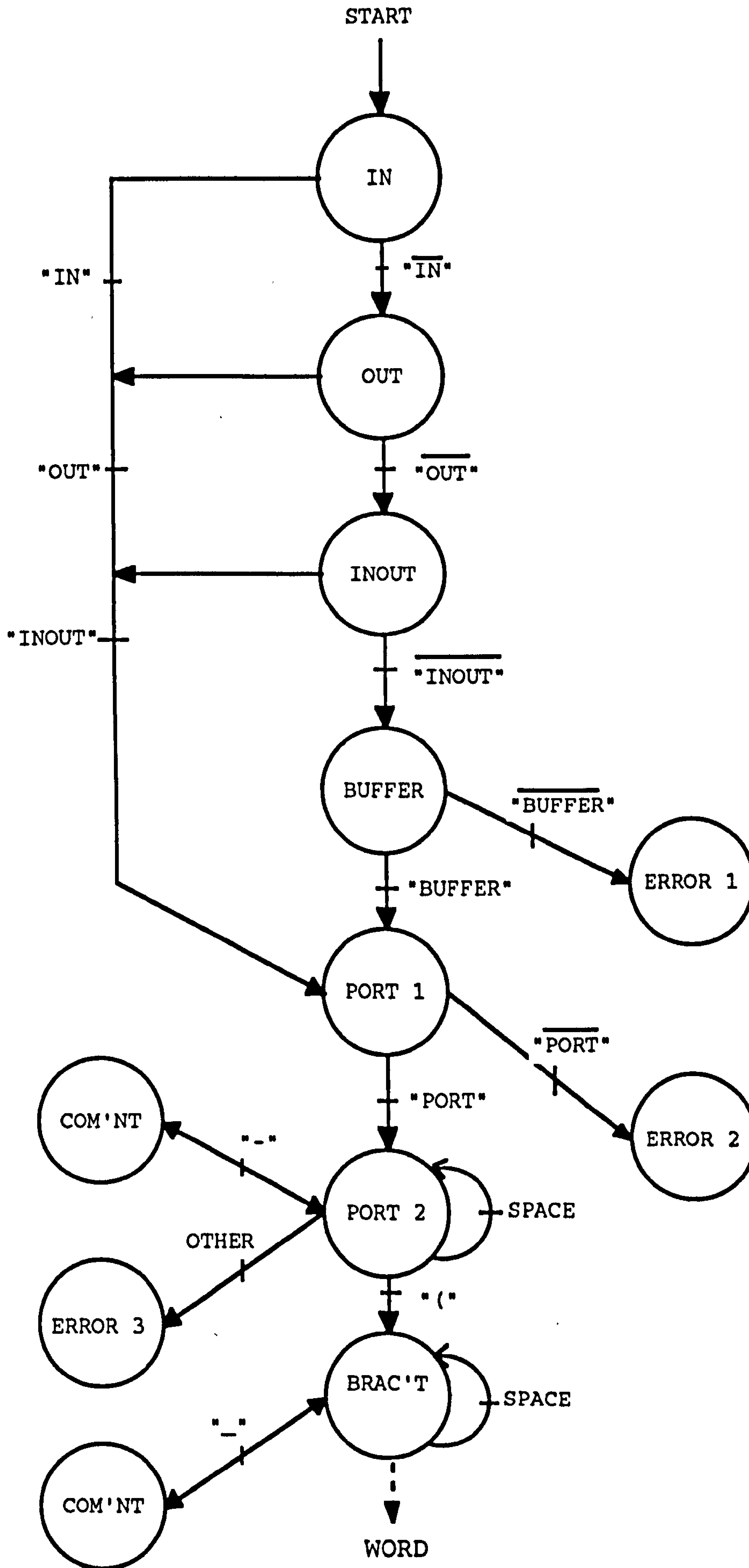



Figure 7.4 a Bubble Diagram of VCP-Version 2

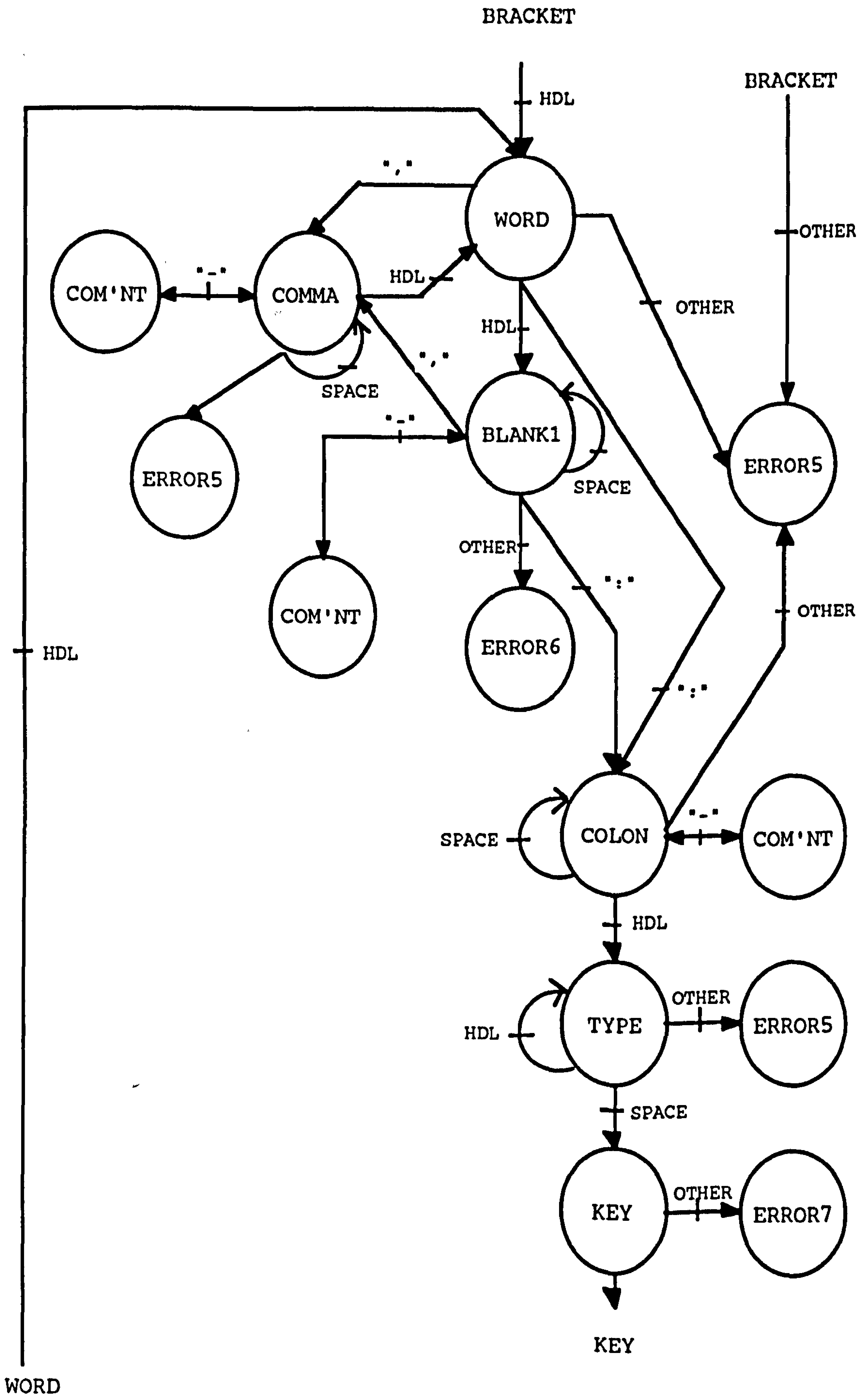



Figure 7.4 b Bubble Diagram of VCP-Version 2

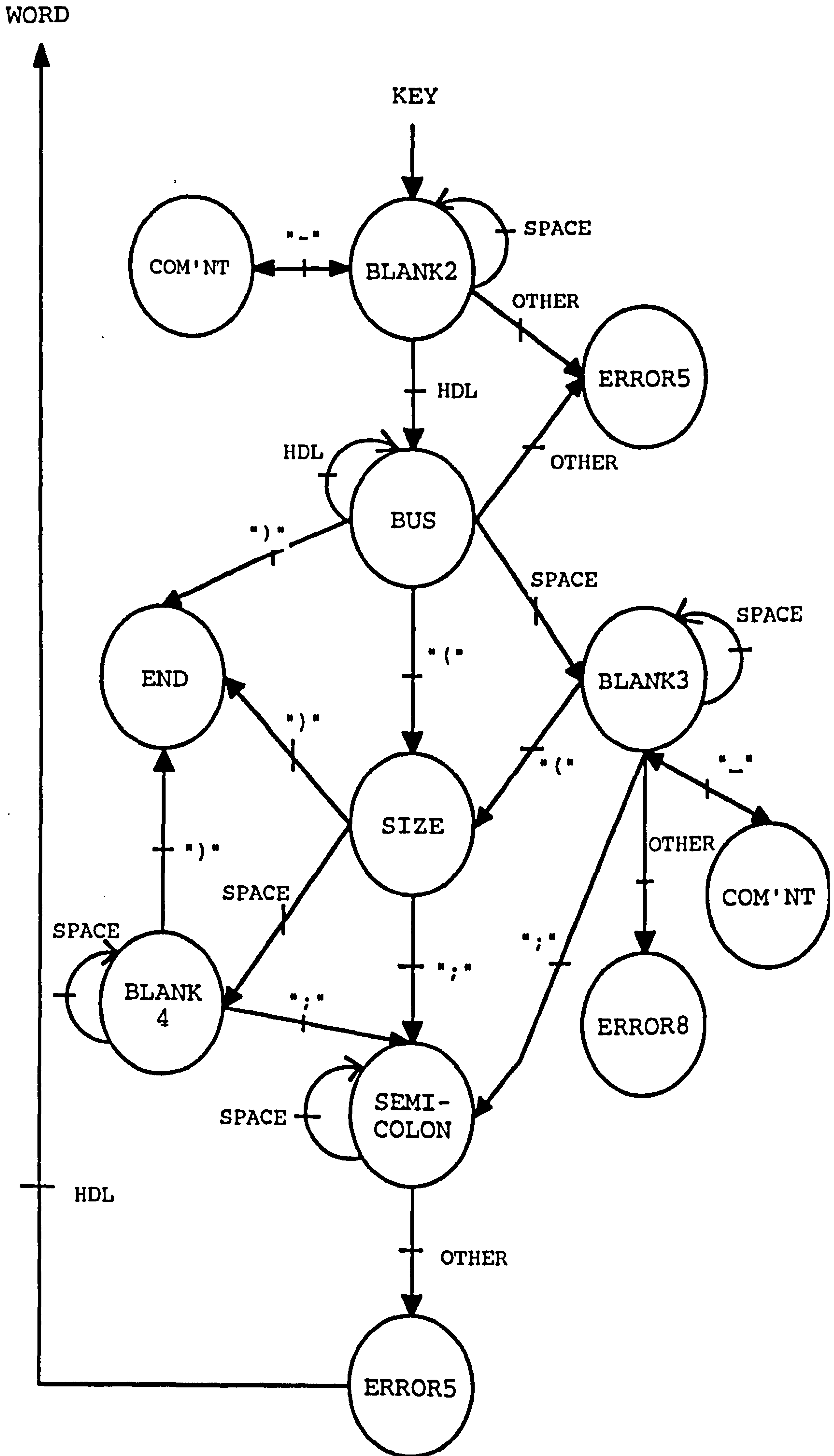



Figure 7.4 c Bubble Diagram of VCP-Version 2

The algorithm was implemented using a high level 'C' program. [TURB 88] The architecture of the EXTRACT.C program was based on a state machine's structural behaviour, which consisted of 21 states, where all declarations such as in, out, inout, buffer, bus and comma were defined. It also included error messages if the VHDL syntax of the entity description was not correct. Functions were declared in the program to handle formats of the output data and also the comments that were included in the VHDL design file. In addition, A number of global variables were declared to include, for example, a maximum bus size of 256 bits, a maximum number of inputs/outputs of 1000 and a restriction on the number of characters in a file name, bus name and wire name.

The main body of the program dealt with generating the output file <file.PIN> and the parsing process of the main key words such as Entity and Port. The extraction process was based on 15 main cases as follows:

CASE 1 : deals with the number and type of IN, OUT, INOUT and BUFFER.
CASE 2 : deals with the PORT declaration.
CASE 3 : deals with the first left bracket in the entity.
CASE 4 : deals with comments in the entity.
CASE 5 : deals with alphanumeric characters.
CASE 6 : deals with commas and colons.
CASE 7 : deals with unknown port_type.
CASE 8 : deals with semi-colon and bracket.
CASE 9 : deals with bus size.
CASE 10 : deals with right bracket in the entity.
CASE 11 : deals with blanks.
CASE 12 : deals with errors.
CASE 13 : deals with End Of File (EOF) being reached.
CASE 14 : deals with I/O conflict on the output file.
CASE 15 : deals with I/O conflict on the input file.

The program was successfully compiled using a standard Turbo C environment running on a PC-386.[TURB 88] It was then ported to the workstation environment and was re-compiled.

## 7.7 EXAMPLE OF THE OPERATION OF VCP-VERSION 2

This version was successfully tried out with a number of examples. The parser could still be invoked from either a UNIX or a DOS based environment. The EXTRACT call is:

extract <design_file_name>.[hdl]

EXAMPLE 1

Extract is invoked with the 2 input AND gate VHDL file described earlier:

**extract and**

Messages will be printed on the display stating:

*<Extract/Note>:Extracting File "and.hdl"*
*<Extract/Note>:Writing "and.pin"*

The generated listing of the **and.pin** file is described below using the following data output format. The row width is 76 characters.

| # | PIN NAME | PIN MODE | PIN TYPE | BUS WIDTH | STARTING |
|---|---|---|---|---|---|
| 0 | A | IN | vlbit | 1 | 0 |
| 1 | B | IN | vlbit | 1 | 0 |
| 2 | C | OUT | vlbit | 1 | 0 |

The results are correct when compared to the source file.

EXAMPLE 2

The parser is operating with the D type flip flop VHDL file described earlier:

**extract DFF**

Messages will be printed on the display stating:
*<Extract/Note>:Extracting File "DFF.hdl"*
*<Extract/Note>:Writing "DFF.pin"*

The generated listing of the **DFF.pin** file is described below:

| # | PIN NAME | PIN MODE | PIN TYPE | BUS WIDTH | STARTING |
|---|---|---|---|---|---|
| 0 | D | IN | Bit | 1 | 0 |
| 1 | CLK | IN | Bit | 1 | 0 |
| 2 | Q | OUT | Bit | 1 | 0 |

The results are correct when compared to the source file.

EXAMPLE 3

The parser is operating with the hypothetically modified circuit- test:

**extract test**

Messages will be printed on the display stating:

*<Extract/Note>:Extracting File "test.hdl"*
*<Extract/Note>:Writing "test.pin"*

The generated listing of the **test.pin** file is shown below:

| # | PIN NAME | PIN MODE | PIN TYPE | BUS WIDTH | STARTING |
|---|---|---|---|---|---|
| 0 | D | INOUT | Bit | 1 | 0 |
| 1 | CLK | IN | Bit_VECTOR | 21 | 2 |
| 2 | Q | OUT | Bit | 1 | 0 |

As can be seen from the above listing file the results are now correct when compared to the 'test.vhdl' source file.

## 7.8 LIMITATION OF VCP-VERSION 2

The primary limitation of this version is that it can not verify port modes as to whether they are defined in a pre-processed package declaration, or are self defined within the main entity declaration of the design file. However, this is not a major limitation and it was felt important to keep the tool simple.

## 7.9 THE INSERTION ALGORITHM

The main function of this algorithm is to attach the necessary I/O cells to the application logic. The algorithm uses the <name.pin> file as one of its inputs to provide the basis for adding the JTAG to the VHDL design description. The <name.pin> file is effectively an encription of the original design entity. Four additional input files are also required by the INSERT algorithm. The input files include JTAG.COMP, JTAG.INSTS, JTAG.SIGNALS and JTAG.USE. These four files contain components, instances, signals and VHDL libraries needed for defining the new design file as will be described later. A new entity is then generated as a result of the insertion process and is stored in a file called <name_jtag>. This file contains the final design with the Boundary Scan Architecture attached to it.

The basic steps of the insertion algorithm operation are as follows:

a. Read the output ASCII file <name.pin> generated by the parsing algorithm.

b. Rename the entity and gives it a JTAG extension.

c. Add JTAG standard signals to the design entity.

d. Add all JTAG components and the design entity.

e. Add the standard instantiation of the JTAG components.

f. Chains all the Boundary Scan Cells in the correct way to form Boundary Loop 0.

g. Generate a new file which contains the newly defined design entity called <name_jtag>.

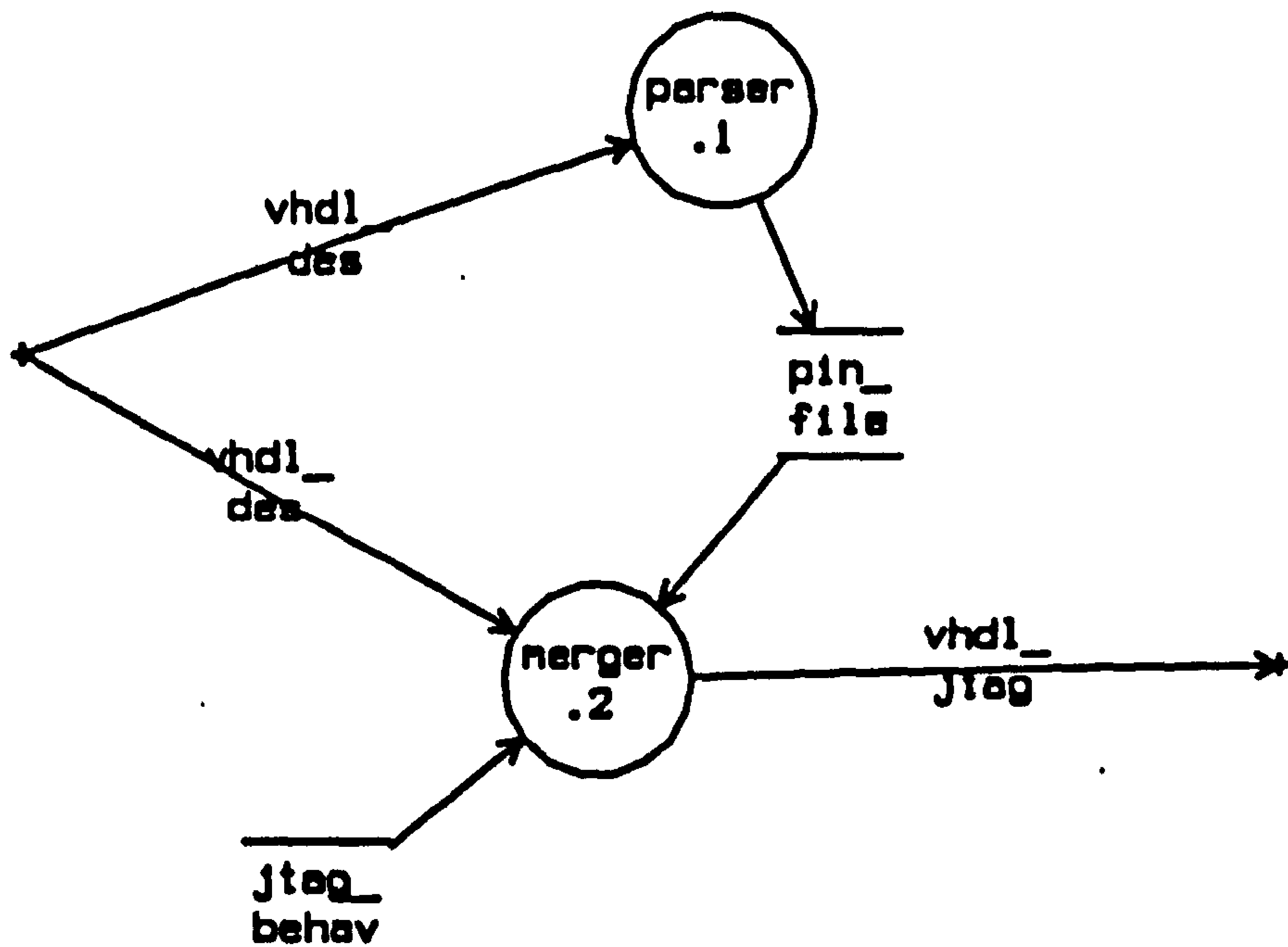
The following context diagram in figure 7.5 describes the basic aim of the insertion algorithm. The data flow diagram C/DFD0 demonstrates the relationship between the two processes, 'parser.1' and 'merger.2'.[WARD 86] The hierarchical decomposition of the parser and merger DFD operations are also described in figure 7.6.

The 'C' program to implement the insertion algorithm starts with checking the existence of the <name.pin> file. It defines the types and functions together with the structure used to store the port data. The total number of signals associated with the design inputs and outputs is then identified. The next step begins by adding the following four declaration files:

| | |
|---|---|
| ***JTAG.COMP*** | *Contains all components of the JTAG Architecture* |
| ***JTAG.INSTS*** | *Instantiates the JTAG components* |
| ***JTAG.SIGNALS*** | *Contains the standard JTAG signals* |
| ***JTAG.USE*** | *Encloses the required VHDL Libraries such as Vdeg_portable (VHDL Design Libraries)* |

The program then inserts the original component definition. It converts the pin file into an internal structure and starts reading the first line of the pin file. This includes port names, port modes, port types, and the size of bus arrays. The signals between the bscan cells and the component are declared. The bscan cells are then attached to the port. A 'generate' function is subsequently used to identify and resolve bus arrays into the required port/signal structure, maintaining both the order and the bus width. A final Boundary Scan Loop 0 is generated for the application logic and is stored in the <name_jtag> file.

An overview of the new VHDL entity architecture is given in figure 7.7.



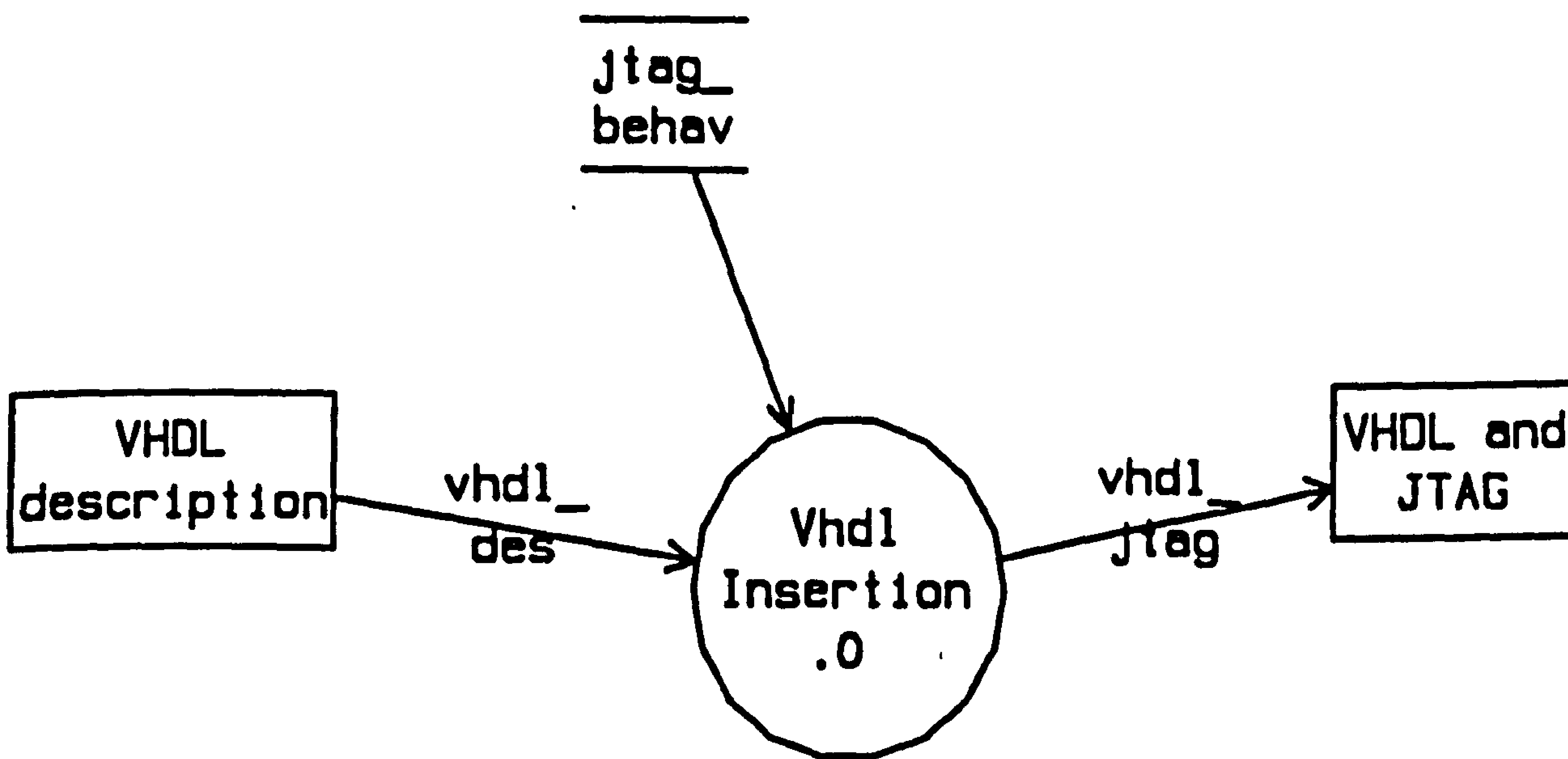




Figure 7.5 Context Diagram of Parser-Insertion Operation

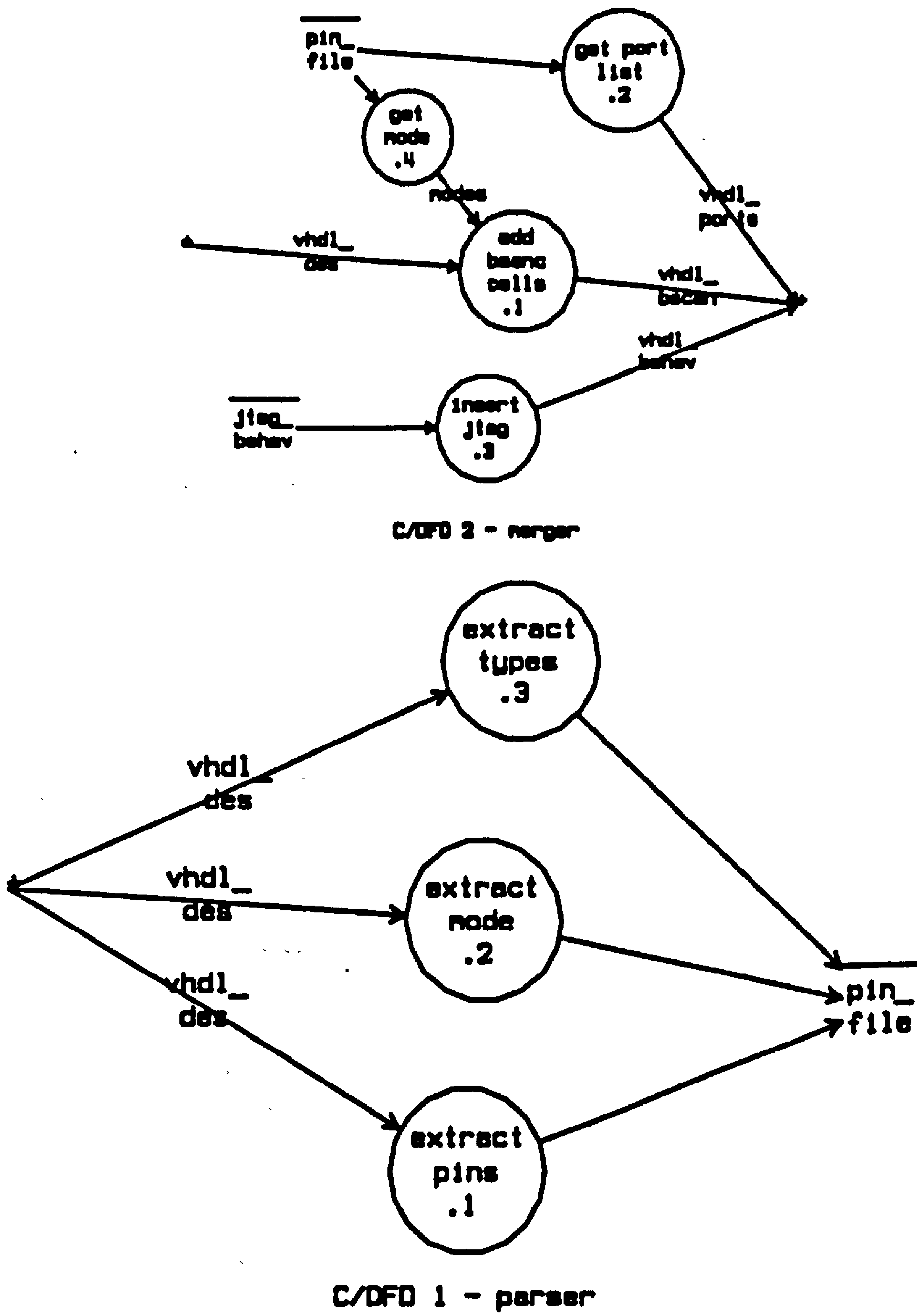



Figure 7.6 Data Flow Diagrams of Parser-Insertion Operation

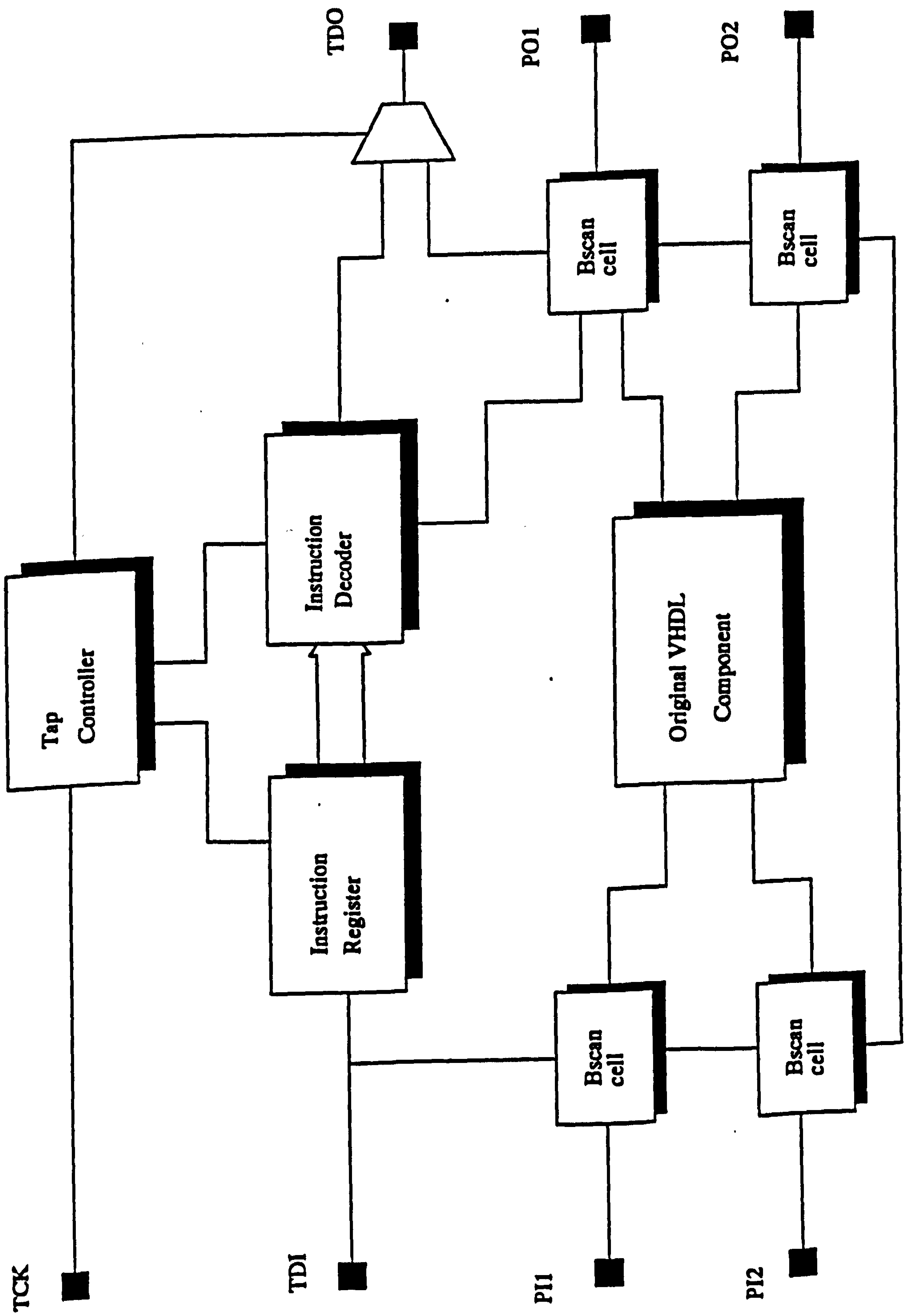



Figure 7.7 Overview of VHDL Architecture with JTAG

## 7.10 DEMONSTRATION OF THE INSERTION ALGORITHM

The insertion algorithm was successfully tested with a number of examples. The insert program could still be invoked from either a UNIX or a DOS based environment. The INSERT call is:

insert <design_file_name>.[PIN]

EXAMPLE 1

This shows the insert program operating with an example circuit of DFF, after successfully completing the parsing/extraction stage and generating a DFF.PIN file. A VHDL description of the DFF entity is listed below:

```
entity DFF is
       port  (
             d , clk  :    in bit;
             q        :    out bit
             );
```

The insert program is now invoked with the dff.pin port listing file:

**insert dff.pin**

This will generate a new file which integrates the behavioural description of JTAG Architecture with the DFF design. The name of the entity description of DFF will change to DFF_jtag and the resulting file name will also be called DFF_jtag. Extracts of this file is shown below:

```
library vdeg_portable;
use vdeg_portable.types.all;
use work.declar.all;

entity dff_jtag is
   port  ( TCK, TMS, TDI : in Bit;
           TDO : out logic4;
           D : INOUT Bit;
           Clk : IN Bit;
           Q : OUT Bit);
end dff_Jtag;
```

The architectural description of the dff component will also change to include all the signals needed for loops 0.[JTAG 90]

```
component dff
port (
        D : INOUT Bit;
        Clk : IN Bit;
        Q : OUT Bit);
end component;

constant one_high : bit_vector(0 to 7) := "00000001";

signal temp : bit_vector(0 to 7) := one_high;
signal shiftir, clockir, updateir : bit;
signal shiftdr, clockdr, updatedr : bit;
signal mode, inst_tdo, indent_tdo, enablee, reset: bit;
signal data_tdo, selectt, select_id, buff_tdo, bypass_tdo :
bit;
signal dr_select, instruction, test_mode, test_regs :
bit_vector(0 to 7);
signal int_D : Bit;
signal int_Clk : Bit;
signal int_Q : Bit;
signal nextt : bit_vector(0 to 3);

begin
   test_regs(0) <= bypass_tdo;
   test_regs(1) <= indent_tdo;
   mode <= test_mode(0);    select_id <= test_mode(1);
   nextt(0) <= TDI;
```

The TAP controller is then connected to the rest of the design as shown below:

```
  tap : tap_c port map
(tms,tck,reset,selectt,enablee,shiftir,

clockir,updateir,shiftdr,updatedr,clockdr);
   bypass : bypass_reg port map
(shiftdr,clockdr,TDI,bypass_tdo);
   instruct : reg_inst port
map(reset,clockir,updateir,shiftir,TDI,temp,instruction,
inst_tdo);
   ident : ident_reg port map(select_id, shiftdr, clockdr,
tdi, indent_tdo);
   decoder : inst_decode port map(instruction, dr_select,
test_mode, open);
   mux2 : mux_2 port map(data_tdo, inst_tdo, selectt,
buff_tdo);
   mux1 : mux_1 port map(test_regs, dr_select, data_tdo);
   tdo_buff : tdo_buffer port map(tck, enablee, buff_tdo,
tdo);
```

The nextt signal shown below is used to describe a bundle of signal structures which forms Boundary Loop 0. This is only used if there are no I/O buses in the design entity.

```
   bscan0 : bscan port map(shiftdr, clockdr, updatedr,
nextt(0), D,
   mode, nextt(1), int_D);
   bscan1 : bscan port map(shiftdr, clockdr, updatedr,
nextt(1), Clk, mode, nextt(2), int_Clk);
   bscan2 : bscan port map(shiftdr, clockdr, updatedr,
nextt(2), int_Q, mode, nextt(3), Q);

  test_regs(2) <= nextt(3);
end behav_jtag;
```

Appendix 7D describes the full DFF_JTAG file.

It is worth noting that the ***GENERATE*** concurrent VHDL [LIPS89] statement is used in the INSERT program for generating regular bus structures. The general form of the generate statement is:

*label_identifier : generation_scheme* **generate**
  *concurrent statements*
**end generate** *identifier;*

There are two kinds of generation schemes: the *if_scheme* and the *for_scheme*. Depending on the kind of generation scheme, the generate statement specifies a repetitive or conditional creation of the set of concurrent statements it contains. In this case the for_scheme is used to generate the bus for loop 0. The for_scheme declares a generate parameter and a discrete range defining the values that the generate parameter will take on. It connects boundary loop 0 which connects all the signals as a bus that contains a combination of the main signals. This will then prevent the connection of the secondary individual signals to the boundary loop.

The parser/insert algorithm was successfully tested with a number of design examples. Appendix 7E demonstrates the operation of EXTRACT/INSERT with a CPU VHDL design where the generate statement was implemented to form boundary Scan loop 0.

## 7.11 CONCLUSIONS

The Parsing/Extract algorithm was successfully designed and implemented. The EXTRACT program was tested with a range of design examples of various complexities, each with a successful outcome.

The developed software could be considered as a complementary tool to an existing CAE based systems. It can also operate as a symantics checker for Entity descriptions. [Auli 89]

Although many CAE vendors provide more than one tool to express a design as shown in figure 7.8, the EXTRACT program could still function as part of a CAE system. However, this requires the design expression to be converted into a VHDL source code or a VHDL netlist.

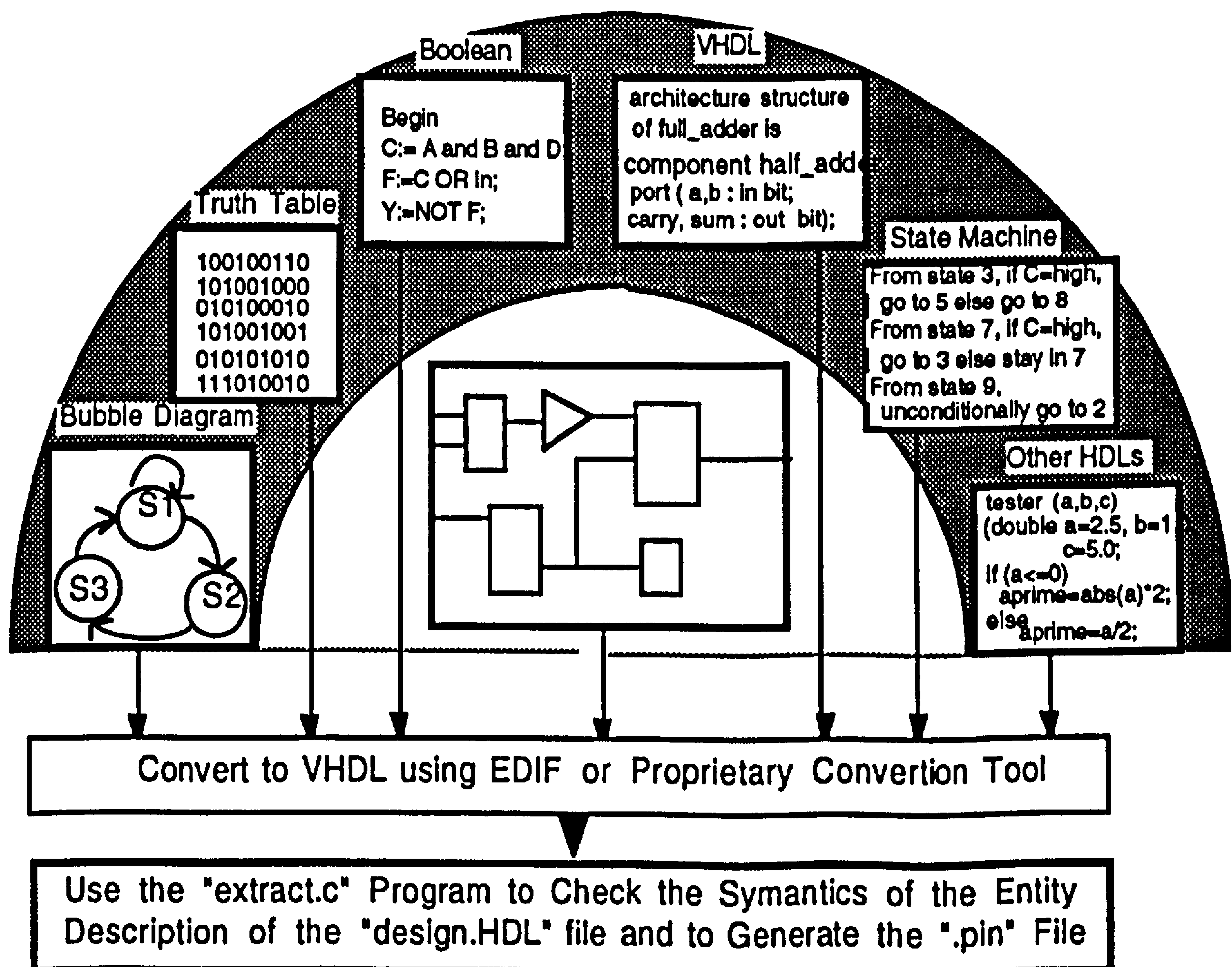



Figure 7.8 Design Expressions and VCP

The Insertion algorithm was also successfully designed and implemented. The INSERT program is currently limited to a maximum of 256 I/Os. However, this can easily be modified to handle larger number of I/Os.

It is worth noting that although the *buffer* port mode used to describe the bi-directional I/Os in VHDL [LIPS89] could be extracted and inserted automatically in the design, the designer is still required to control the direction of the signal as to whether it is performing input or output operation.

The Extract/Insert environment was successfully tested on both PC and workstation type computers and has a fast operational speed.

**REFERENCES**


[COEL 89] David R. Coelho. The VHDL Handbook, Kluwer Academic Publishers 1989.

[LIPS 89] VHDL : Hardware Description and Design, Kluwer Academic Publishers 1989.

[IEEE 88] IEEE Standard VHDL Language Reference Manual -STD 1076, 1987. (New York: IEEE:1988).

[JTAG 90] IEEE Standard 1149.1, 'A Standard Test Access Port and Boundary Scan Architecture' May 1990.

[SCHI 88] Herbert Schildt. Using TURBO C, Programming Series. Borland Osborne/McGraw Hill 1988.

[TURB 88] Borland Turbo C User Guide Version 2, 1988.

[WARD 86] P.T.Ward, S.J.Mellor, Structured Development for Real Time Systems, Vol 1-3. Yourdon Press, NY. 1986.

[MCAS 7] MCASE™ Structured Analysis Manual from Mentor Graphics Version 7.

[Auli 89] Auli Reinikka, Antti Auer, Ari Okkonen, Automatic Synthesis of Structural HDL Description from Graphic Specification of Embedded ASICs, Microprocessing and Microprogramming 27, 1989 pp. 473-478.

# CHAPTER 8
# COST IMPLICATIONS

## 8.0 COST OF JTAG

Any change made to an IC to increase testability inevitably incurs costs. Two main factors which affect cost implications are the additional physical requirement and the performance degradation.

Adopting the JTAG architecture will require additional design time, cells, pads and four extra pins per device. This additional requirement will be heavily dependent upon the ASIC's function and processing technology, together with metal dimensions. It grows linearly with the number of boundary scan inputs and outputs.

It is worth noting that the rate of growth is different for that of a soft VHDL developed macro, compared with that of a hard macro approach. Therefore, a trade-off between the development time and the chip size has to be considered for critically dense devices.

There will also be some reduction in chip performance. A greater impact on the circuit's performance is likely to be that of the additional circuitry, which is required immediately before the output buffers on a design and which would typically add two gate delays to signals leaving the device. Input signals would be similarly delayed. Although the importance of these delays can be minimised by careful design, there are always likely to be performance critical applications in which they will remain unacceptable. It is also recommended to have additional power and ground rails to the ASIC specifically for the JTAG circuit.

## 8.1 AREA COST

The total gate count of the TAP controller, the instruction register and the instruction decoder is 243 gates. Even with the optional identification register, the total gate count for the core area is only 659 gates. For all but the very small arrays, this is insignificant. The real area penalty comes from the boundary scan chain itself, which is formed as a "hard" ring around the core area. Since the size of this ring is fixed for a specific ASIC family, the number and the types of I/Os used in a particular design are not significant - unless the design is I/O limited rather than gate limited.

Once a particular die size is chosen based on the design complexity, the boundary scan chain overhead is thus fixed. Hence, it does not therefore matter how many of the available I/O pads are used. The I/O pad's overhead will vary as it represents a portion of the total die size. For a large die of 100K gates, it represents 7% of the gates that would be available to a normal, non-JTAG design. For an average die of 25K gates, the portion represents 10%-13%. However, for a small die of 10K gates, the boundary scan I/O cells would represent 22% of the gates that would be available for a non-JTAG design.

For an even smaller die of 3K gates, the overhead for the "hard" boundary scan ring is approximately 40%, leaving an estimated usable 1400 gates in the core - but 243 (or 659 if the identification register is included) are needed for the TAP controller, making the total overhead nearer 50%.

Thus for designs below approximately 10K gates in complexity, the benefits of the boundary scan need to be weighed very carefully against the large penalty. For larger designs, which are always going to include some design-for-testability logic, the area of implementing boundary scan can be almost negligible.

## 8.2 PIN, CELL CONNECTIONS AND POWER COSTS

In addition to the four (optionally 5) pins required to implement the IEEE 1149.1 standard, the following connections must be considered:

- DIN and DOUT. These signals connect the scan cell with the pad and core of the ASIC. Since each cell can be personalised as input and output, each line must be available on either side. Thus, two tracks are needed.

- TDI and TDO. The scan data pass along these lines. Since the signals are daisy chained, a single track is required.

- Global control signals. Four signals are necessary to control the operation of the latches and multiplexers in the scan cell. Each line must pass through the cells and therefore four tracks are necessary.

- Power Supply. To avoid influences from other portions of the circuit, the boundary scan logic should be supplied by a separate power bus, which requires at least two tracks.

With the hard ring of boundary scan I/O cells, some of the corner pads on the ASIC master slice can not be used for boundary I/Os. However, these pads can be used for power supply or for additional test pin inputs.

## 8.3 DELAY COST

When the chip is operating in its normal mode, the inclusion of JTAG circuitry means that there is an additional 2-to-1 multiplexer delay when going in to the device and a similar 2-to-1 multiplexer delay when coming out.

The delay through the multiplexer is dependent upon many factors:

- The Voltage
- The Temperature of chip operation
- The Processing factor of the silicon
- The Technology and Die Size in which the design has been developed
- The Fanout from the multiplexer

For example, using the LSI's CMOS LCA (Logic Cell Array) 10K, the additional delay incurred from including JTAG is 1.5ns and it is 1.0ns when using the LSI LCA 100K. These are worst case commercial figures. Therefore, the total impact on system performance is 2-3ns, approximately, on paths through the chip. This may be significant in some very critical cases, but for most designs it will represent no problem at all.

## 8.4 DESIGN TIME COST AND COST BENEFIT OF THE TOOL

Often, the decision to use new methodology such as JTAG, hinges on how it will affect the time taken to complete the design and to commence prototype manufacture. The impact of JTAG in this respect is very difficult to quantify. However, adding JTAG to an ASIC design whether translated from the IEEE 1149.1 specifications, or using a specific IC library is normally a time consuming process.

Integrating the JTAG circuitry into the design by using the parsing/insertion tool is a relatively simple operation. Therefore, the primary cost reduction that can be attained from using the tool developed in this project, is that of reducing the time of the development phase. The tool enables the designer to provide a validated description of BSA including its test vectors automatically. Thus, the life cycle costing of the ASIC will be reduced, including not only in the design phase, but also in the structural testing phase of the chips after fabrication.

**REFERENCES**


[EM&T 90] Electronics Manufacture & Test Magazine 1990 Vol. 9, Part 3

[IEE 89] IEE Review February 1989,"Printed Circuit Boards are Becoming Harder to Test. The JTAG could have the Answer."

[SCIC 91] Gartner p, Buchner T, Roos G, and Schwerderski T, "Boundary Scan and its Application to IMS Gate Forest" Institute for Microelectronics Stuttgart, Allmandring 30a, D-7000 Stuttgart 80, Germany. Journal of Semiconductor ICs, Vol. 9, No. 2, Elsevier Science Publishers 1991, England.

[D&TC 90] Van Rissen R P, Kerkhoff H G and Kloppenburg A, "Designing and Implementing an Architecture with Boundary Scan". IEEE Design & Test of Computers, February 1990.

[HITA 90] D'Souza D, "JTAG and Hitachi's Auto Diagnosis", Hitachi America Ltd, 2000 Sierra Point Parkway, Brisbane, CA 94005-1819 USA. 1990

# CHAPTER 9
# OVERALL CONCLUSIONS

## 9.0 THE NOVEL APPROACH OF THE PROJECT

The research work presented in this thesis has identified a new approach to integrating Boundary Scan Test Architecture automatically into an ASIC design. This has involved the development of an automated environment based on the creation and successful implementation of two main components:

1) a parameterised behavioural model of BSA IEEE 1149.1 standard using the IEEE 1076 Hardware Description Language VHDL.

2) a new algorithm for developing a parsing and insertion tool to integrate the behavioural description of a BSA into an ASIC design.

The novelty of this tool is that it provides the designer with a simple, yet a powerful, environment to include BSA into his/her ASIC design, with the minimum of effort.

This approach is different to that of Hewlett Packard's BSDL language, referred to in chapter 3, in that the BSDL was developed as an extension to VHDL. BSDL allows the designer to describe BSA into his/her design manually using specific constructs provided by the language. Unlike the BSDL environment, the new tool described in this thesis enables the designer to use a pre-developed and tested high level VHDL model of BSA, which can be used at the system level of the ASIC development cycle.

The tool is also different to the Testability Improver (TIM) system developed by Philips, in that it is independent of any IC manufacturer and CAE vendor. It is aimed at the system / behavioural level of the ASIC design, where the TIM software can only be employed after the design has been created and converted into the register transfer level.

When compared to other design automation test utilities such as TEA, ADAS, and TISSS described earlier in Chapter 3, the tool can be used as a facility that augments these vendor specific tools. For example, the EXTRACT program developed in this thesis can be used as a checker for VHDL semantics for any entity description of various design complexities.

The VHDL model of BSA offers the designer complete freedom to change the generic parameters, such as propagation delays, of the BSA model to suite his/her particular application. In addition, the model could be used as a pre-developed and tested, standard library part which could be made available on any CAE data capture system.

The tool can benefit the test engineer, by using one key information source for all boundary-scan characteristics, which reduces the possibility of error. It can also serve as a partial compliance check, as design errors may surface in the implementation of the test standard at the gate level using a specific IC library.

The functional test vectors developed for the BSA could also be communicated from the designer to the target verification system or ATE, with little involvement from the test engineer.

The environment developed here has the advantage of being portable and can easily be incorporated into an existing CAE system. It has been tested with a number of design examples, illustrating its operational advantages.

A number of CAE developers such as DAZIX and Innovation Research have expressed an interest in utilising the tool as an additional utility within their systems.

## 9.1 ACHIEVEMENT OF AIM

Boundary Scan is rapidly becoming a necessity. The implementation of boundary scan across the industry will solve numerous board test reliability problems and promises to save time while keeping cost down. The implementation requires widespread utilisation of IEEE Standard 1149.1.

The work presented in this thesis provides a methodological framework, for integrating a high level VHDL behavioural model of the IEEE 1149.1 Boundary Scan Test Architecture into an ASIC design automatically, using a design automation tool.

The tool developed as part of this programme of research provides the designer with the ability to explore his/her design with BSA in a significantly reduced design time, since it removes the need to know the BSA's structural characteristics. The new tool therefore, encourages the designer to consider a test strategy from the initial stages of the ASIC development, rather than including testability features as an afterthought.

The tool is based on a high level intelligent parsing and insertion algorithm which has been successfully implemented in 'C'. The parsing phase can be operated on an ASIC's VHDL description to initially check the correctness of the VHDL syntax, and to generate a list of the design I/O terminals.

This data together with the VHDL behavioural description of BSA are then used by the insertion phase of the algorithm to create a new design with the BSA attached.

## 9.2 ACHIEVEMENT OF OBJECTIVES

The work developed in this thesis satisfies the two main objectives. The first, concentrates on translating the specifications of the IEEE 1149.1 BSA into parameterised behavioural models using the VHDL description language. The second, focuses on developing a parsing and insertion algorithm which enables the designer to integrate the behavioural model of BSA into her/his ASIC design.

The BSA was initially developed and simulated structurally to conform with the IEEE 1149.1 standard. It was then tested with adder and 4-bit multiplier circuits to validate its operation.

The VHDL IEEE 1076 standard was chosen to develop the necessary models of the BSA components. Its capability to describe digital systems at various abstraction levels and in three styles (Behavioural, Structural and Data Flow) has enforced the choice of this language for developing the BSA model.

In addition, VHDL's ability to link design entities to their behavioural and structural description, resulted in defining an accurate BSA behavioural model with embedded structural properties including timing and control.

The VHDL models of the BSA were developed using a combination of ECAD tools including Mentor Graphic's 1076 VHDL Environment (version 7), and the View Logic Version 4.1. Full simulation was carried out and consisted of the necessary test instructions including, NOP, SAMPLE, EXTEST and BYPASS. The timing elements associated with the modules were defined in a generic form within a VHDL package, so that the desired delays could easily be modified by the designer.

The architecture was modelled with an application logic and 6 instruction tests were carried out in conformance with the IEEE standard. Fault simulation was also carried out to evaluate the quality and efficiency of the test vectors.

The test vectors were developed using the 'C' based 'macro function' language to provide an easy integration path with back-end test verification tools and ATE.

A full parameterised behavioural model was successfully developed, exhibiting the necessary test features which are normally confined to the structural level.

An algorithm was then developed to enable the inclusion of BSA into the ASIC design. The first phase dealt with identifying where the Boundary Scan Cells were to be added, the order they were to appear in the design and their types, in order to form a scan loop 0. In doing so, the algorithm concentrates on the "Entity" part of the VHDL description of the design. It is capable of handling all I/O terminal types including inputs, outputs, bi-directional, tri-state and Bus types. Although the added architecture will primarily form the hardware infrastructure required for Boundary Scan Test, the TAP model was defined to cope with internal scan and other BIST requirements.

The second phase of the algorithm dealt with the insertion of the TAP controller, the Instruction Register, the Instruction Decoder, the Bypass Register, the Identification Register (which is described generically) and the Test Register to form loop 0 into the ASIC design.

The algorithm's specifications were described and analysed using a structured design approach including data flow and entity relation diagrams. It was then implemented in 'C' and was compiled on both PC-AT (DOS based) and workstation (Unix based) environments.

A number of examples such as a VHDL CPU core were successfully tested to verify the validity of the tool's operation. The tool compiles very quickly and requires no significant user time. Both the tool and the BSA model are easily maintainable as they are designed in a modular and hierarchical format.

# CHAPTER 10
# FUTURE WORK

## 10.0 INTRODUCTION

This chapter examines the potential for further developments on the work which has been carried out. There are 4 possible extensions to this work. The first examines the benefits of developing a graphical environment, similar to that of Data Flow diagrams, for defining VHDL constructs and therefore the BSA model.

The second examines the possibility of extending the modeling environment of the BSA described in this project, to include mixed analogue and digital signals for testing analogue design parts such as Analogue to Digital Convertors and Comparators. It therefore examines the IEEE subsets of 1149 standard.

The third considers the potential for linking the proposed environment developed by this project to a particular testability synthesis system, such as the one provided by DAZIX and Mentor Graphics.

The final subject examines the potential role of Artificial Intelligence for developing a theory of reasoning that exploits the knowledge of structure and behaviour of a digital system.

## 10.1 GRAPHICAL REPRESENTATION OF VHDL

A facility for graphically representating VHDL constructs will provide the designer with an alternative method of describing the VHDL representation of Boundary Scan Architecture into his/her design. With the emerging IEEE 1076 VHDL standard, VLSI systems are increasingly being designed using VHDL Hardware Description Language. This has often added to the burden that faces VLSI designers in learning a new language and method of description. In addition, hardware designers often favour graphical entry methods for VLSI system's design to semantical hardware descriptions. It is suggested that a graphical medium can often assist the designer in a better understanding of the behaviour and structure of a particular algorithm, and its implementation. The BSA components will have a unique graphical representation and can be added from a pre-determined library. It is therefore, envisaged that developing a graphical facility, coupled with the parsing/insertion algorithm presented earlier in this thesis, will further encourage designers to include BSA in their designs.

There are three common methods for describing structured diagrams which include:

1) A structured diagram which shows the implementation of a block in terms of sub-blocks and their interconnects.

2) A Petri-Net which is used to describe the control behaviour of the system.

3) A Gantt Chart which illustrates the results of implementing a particular block in terms of silicon area occupied and the time required to compute the function.

It is important to concentrate on Hierarchical Abstract Descriptions of system design with embedded JTAG in order to limit the scope of the system being modelled to a degree which can be managed at one time.

A number of papers describing the graphical methods for user interface, have been published. D.Morris, [MORI 88] for example, describes a methodology for formalising the use of diagrams in the design of microelectronic systems. He uses a top-down hierarchical graphical method for system design, with a high level of abstraction. It shows major modules and data paths, ending up the decomposition with discrete primitives at the bottom (level zero).

C A Kuszynski, [KUSZ 90] describes a compiler which produces a graphical representation of the HDL STRICT.

STRICT describes blocks in terms of their behaviour and structure. The Behaviour uses WHENEVER to sense lines and SET to assert them. The behavioural description is mandatory and describes what the block is supposed to do. The structure of the block describes how the behaviour is to be implemented using primitive components. The graphical representation is a hierarchical view of the text. At the top level, a simple rectangle is drawn with the name of the block. Petri nets can then be used to describe some of the behaviour but only as a set of token passing operations.

J.Bain, [BAIN 88] describes the STELLA Schematic Capture tool for ELLA hardware description language. STELLA gives a hierarchical graphical representation of both behavioural and structural description styles. It allows designs to be entered as either text or schematics.

A. Reinikka et al, [REIN 89] describes an automatic synthesis of structural HDL descriptions from a graphical specification of an embedded ASIC. He describes a method using real time structured analysis to design and implement the ASIC. His reasons for using structured analysis is that, it has a simple graphical interface consisting of different levels of abstraction, together with a good representation of concurrency. [AUER 88], [OKKO 89], [LEPP 89].

A I Wasserman, [WASS 90] describes a method of representing object oriented designs using structured analysis techniques. The method builds on the Structure Chart notation.

The Structure Chart represents a design as a number of communicating models. It shows functional calls in addition to the parameters which are sent and returned. A Class is represented as a rectangle. Operations which can be performed on the Class are described as overlaying boxes as shown below:

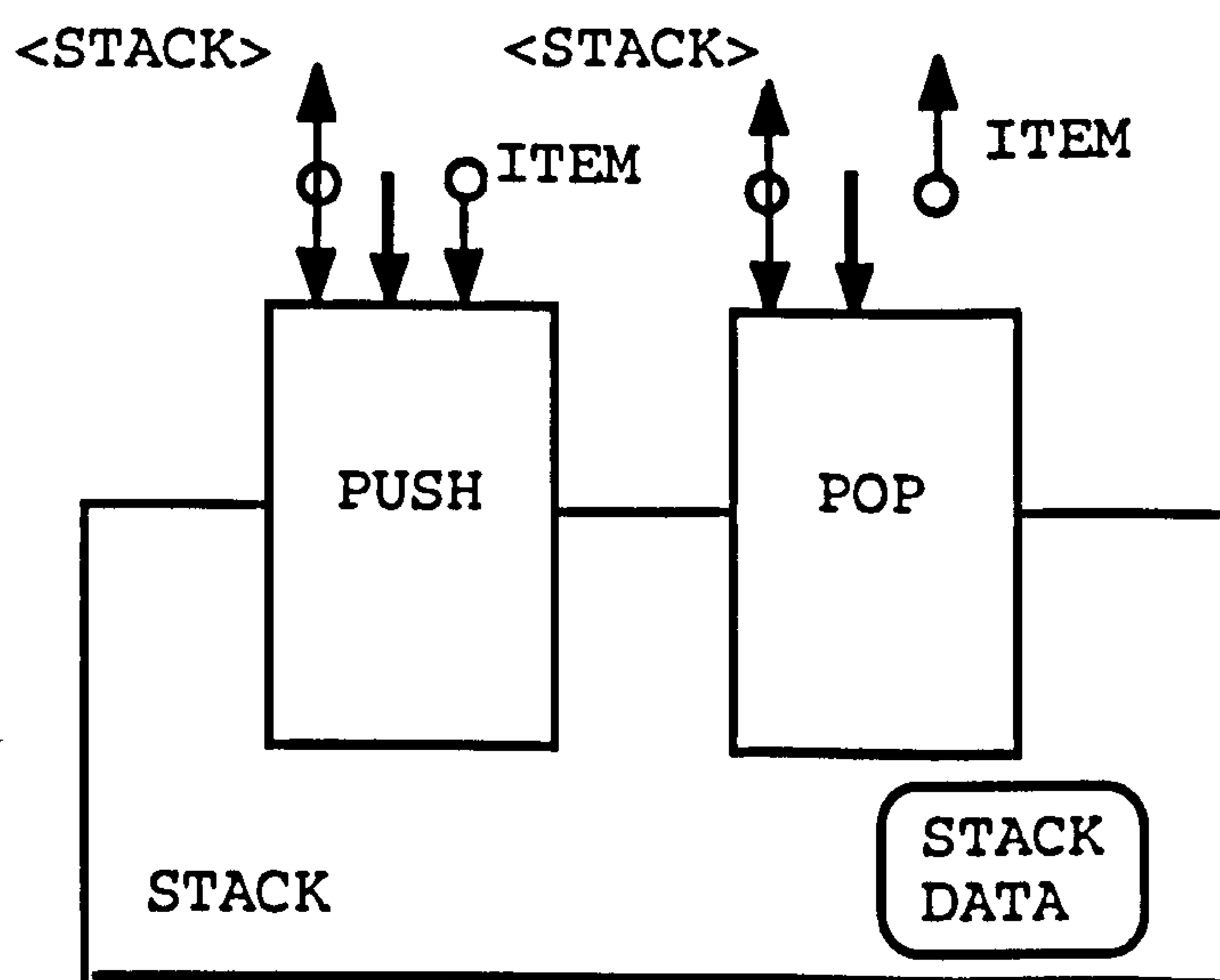



**Figure 10.1 Graphical Representation of a Class**

Object oriented languages allow generic classes to be created. These are classes which are not complete until the run time is executed and when the parameters are supplied, such as record length. [ACKR 91]

Only the notation of an object oriented structured design shows the class interface. It does not show the behaviour or what the operations of a class actually do. It gives an overview of the system and shows how everything fits together. In this way it is very much like a Structured Chart. Structured Charts can be derived from data flow diagrams. [WARD 86]

### 10.1.1 GRAPHICS HARDWARE DESCRIPTION LANGUAGES (GHDLs)

The purpose of GHDLs is to provide an alternative graphical companion notation to HDL. They are analogous to logic diagrams which serve as a companion to Boolean equations. [AUER 88], [OKKO 89], [LEPP 89]

The GHDL uses a hierarchy of blocks and a number of levels of abstraction. The register transfer level is the highest level of abstraction. The individual blocks are connected together, as far as possible, by abutment. This reduces the visual complexity. A 2 input multiplexer is shown below:

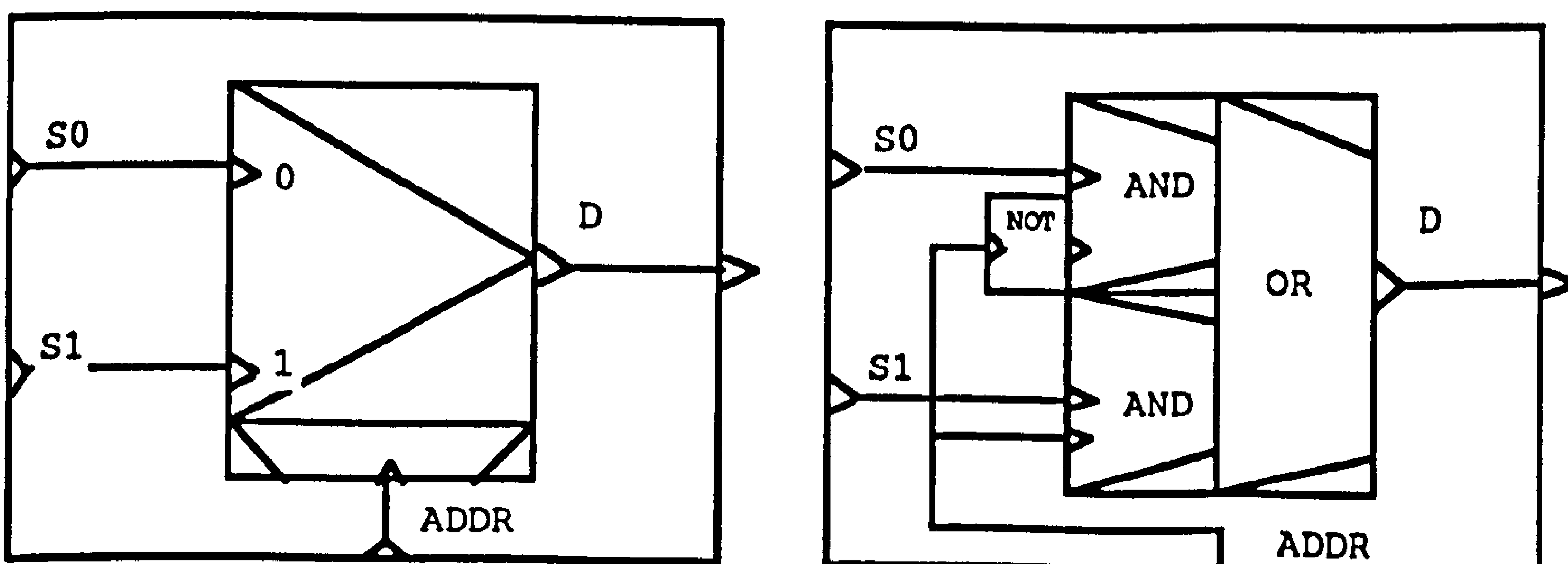



**Figure 10.2 GHDL Representations**

The diagram on the left shows the register transfer version of the multiplexer, and the diagram on the right is its logic representation.
Other GHDLs are described but most of them only support the register transfer, logic and switch levels of abstraction. Those GHDLs that support behavioural constructs are very minimal.

### 10.1.2 GRAPHICS TO VHDL AUTOMATIC CONVERTER

Products currently available on the market include the following:

Express VHDL from i-Logix allows designers to create VHDL code using graphics rather than text. The tool has two high level modeling techniques. The first technique is to define the behaviour of a system using State Charts, in that it defines the system's functional blocks and what they are supposed to do. The second technique is to describe the data flow within the system using Activity Charts which define when each function is to be used.

Stateview from Isdata allows state machines to be represented graphically and edited using a graphics editor. Both Mealy and Moore models can be represented. Arithmetic operations can be used to simplify complex branch equations. The representation is similar to algorithmic state machines. Complex conditions can be hidden in tables to reduce the complexity and the size of diagrams. The output can be in the form of VHDL code or other data formats form implementation in Programmable Logic Devices and Programmable Logic Arrays.

The model used to represent the system in this case could be based on data flow diagrams and algorithmic state machines. The data flow diagrams could give a high level abstracted view of the system. The data flows could provide the abstract data types, such as enumerated data. A data dictionary could also be used to define these data types.

A process could normally be created for events in the system's environment and a response in the form of a first-cut diagram could then be generated (i.e. drawn).

This process could upwardly be refined until one process could then be used to represent the entire system.

The first-cut diagram could therefore be refined until each process can be represented as a high level algorithmic state machine (ASM).
A possible example of an ASM of a simple ALU is shown below [GREE 86] :

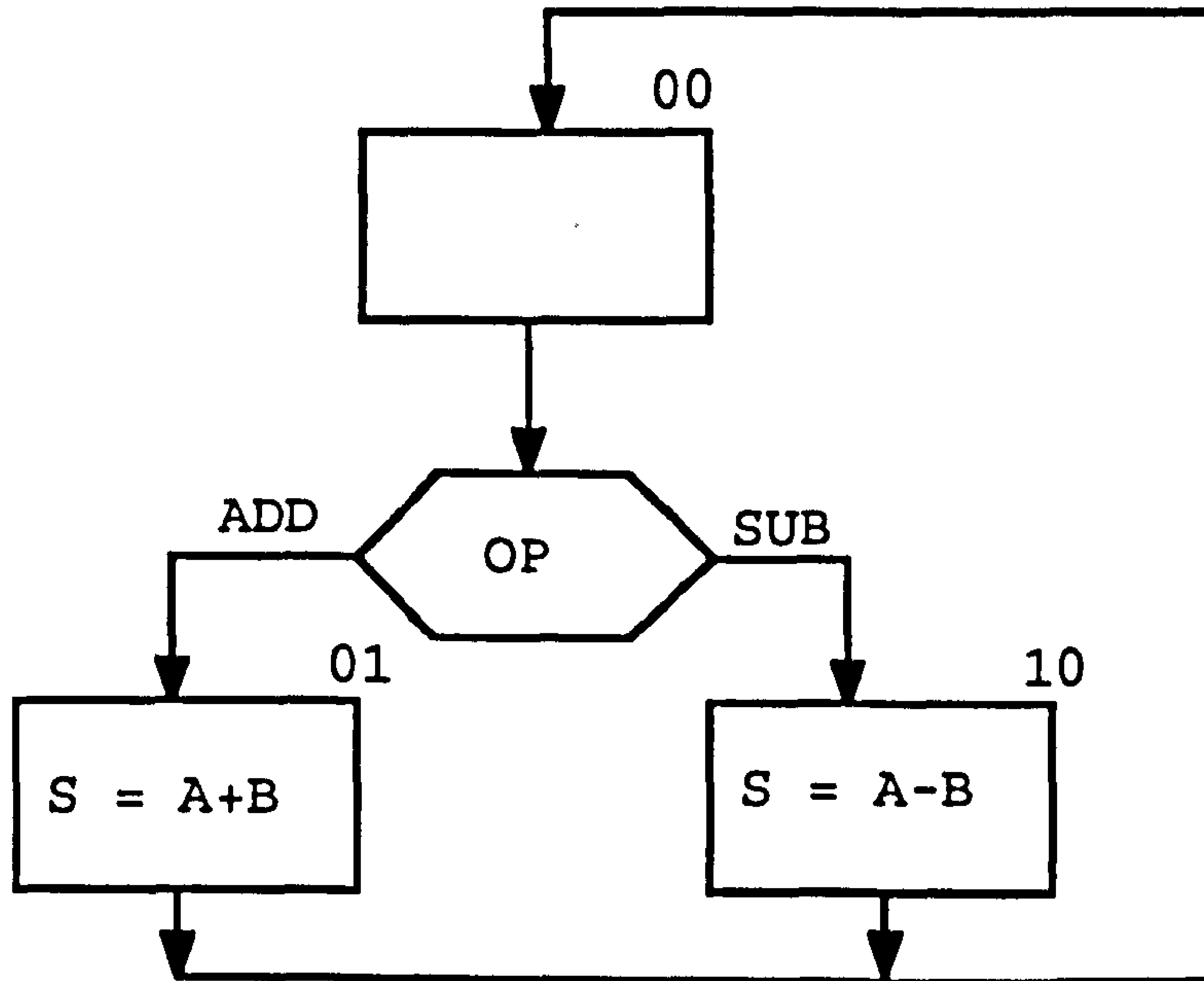



**Figure 10.3 Example of Algorithmic State Machine**

This is a three state system with signals A, B and S. OP is a signal of an enumerated type which has two possible values ADD and SUB. The other three signal types can not be determined from the diagram so the data dictionary would have to be consulted. In each state not only can outputs be assigned values, but internal variables and signals can be set. In this way, loops and counters can be represented.

The drawback of this is that although any construct can be represented, including those such as "GOTO" statements, VHDL does not support such statements and it is a bad practice to use them. This means that the representation must be limited to structured design techniques such as "FOR" and "WHILE" loops.

All data flow transforms will be assumed to work concurrently. A controller state machine can be defined to enable the other transforms in the design.

When translating the graphical representation to VHDL, the individual ASMs will be converted into processes and procedures depending upon whether they are running concurrently, or whether they are being called by other state machines. The loops in the charts must be found and converted to "While" and "For" constructs. Single and multiple decisions will be converted to "If" and "Case" statements respectively.

The variables and data types in the data dictionary will have to be declared and initialized. The data type definitions are placed in a package so that they can be used by all entities. The resulting VHDL will have to comply with IEEE 1076 standard and will enable the integration of the JTAG insertion routine graphically.

## 10.2 ANALOGUE IMPLEMENTATION

The 1149 rules create functional test access to analog circuits as part of the integrated design. Internal test architectures designed with testable interfaces provide accurate, repeatable and cost effective test solutions, by eliminating the need for expensive test instrumentation, maintenance and calibration. In addition, the required test execution time for on-board test is much less than that of traditional ATE which typically operates over the IEEE 488 bus. Test access through a standard test bus, combined with flexible signal processing techniques create a powerful analog test capability to meet challenges such as that of Surface Mount Technology.

### 10.2.1 ANALOGUE TEST APPROACH USING 1149.3 AND 1149.4 SUBSETS

The testability bus 1149 option, Real Time Analog Subset, allows a standard interface to the analog circuits either internal to the design, or externally with the use of ATE. [ARME 89] This option is used when it is not feasible to incorporate analog to digital (ADC) or digital to Analog (DAC) conversions within the module.

This subset uses many of the 1149, Real Time Testability Bus, signal lines providing necessary control in a mixed signal environment. The signals necessary to define the 1149.4 subset and some of the 1149.3 subset signals are listed in Figure 10.4.

| SIGNAL NAME | SIGNAL DESCRIPTION |
|---|---|
| RESET | Initialise all testability circuitry |
| ENABLE | Enable all testability circuitry |
| ASIN | Analog real time test signal input |
| ASOUT | Analog real time test signal onput |
| ASINEN | Analog test signal input enable |
| RTOUTEN | Enable for D/A real time output lines |
| TPA0-n | Test point address lines for digital and analog real time data and signal inputs and outputs. |

**Figure 10.4 1149.3 & 1149.4 signals**

Most analog circuits require a test stimulus in order to measure the resulting response. For a typical ATE environment this is accomplished with the use of several test points accessed by a bed-of-nails fixture via spring loaded test probes. Signals that are routed to and from the tester, often require the use of special buffering or signal conditioning, before evaluation by the test program software. This signal routing can be simplified with the use of the 1149.3 and 1149.4 testability bus subsets.

A standard test interface is being developed by the IEEE to control and process the analog test signal by using the internal test architecture. The architecture is mainly based on analog multiplexers and analog de-multiplexers.

## 10.3 LINKING THE ENVIRONMENT TO SYNTHESIS TOOLS

Many CAE systems are capable of expressing designs using more than one method. In addition, most CAE vendors have now included logic synthesis tools as part of their design suite.

VHDL has provided a neutral environment in which the design can ultimately be converted into. This has facilitated the recent advancement in the synthesis technology.

One of the potential applications of the tool developed in this project is to integrate it into a test synthesis system of a particular CAE vendor. For example, the tool developed in this thesis could conceivably be linked to the DAZIX Test Synthesis application (TESTSYN). TESTSYN is a tool which is primarily used for including internal scan in the ASIC design. It works on the principle of converting, where possible, the design flip-flop cells into scannable form by re-configuring the design and generating a new netlist. The VHDL parser and the boundary scan insertion algorithm can therefore, provide a complete Design For Test Environment.

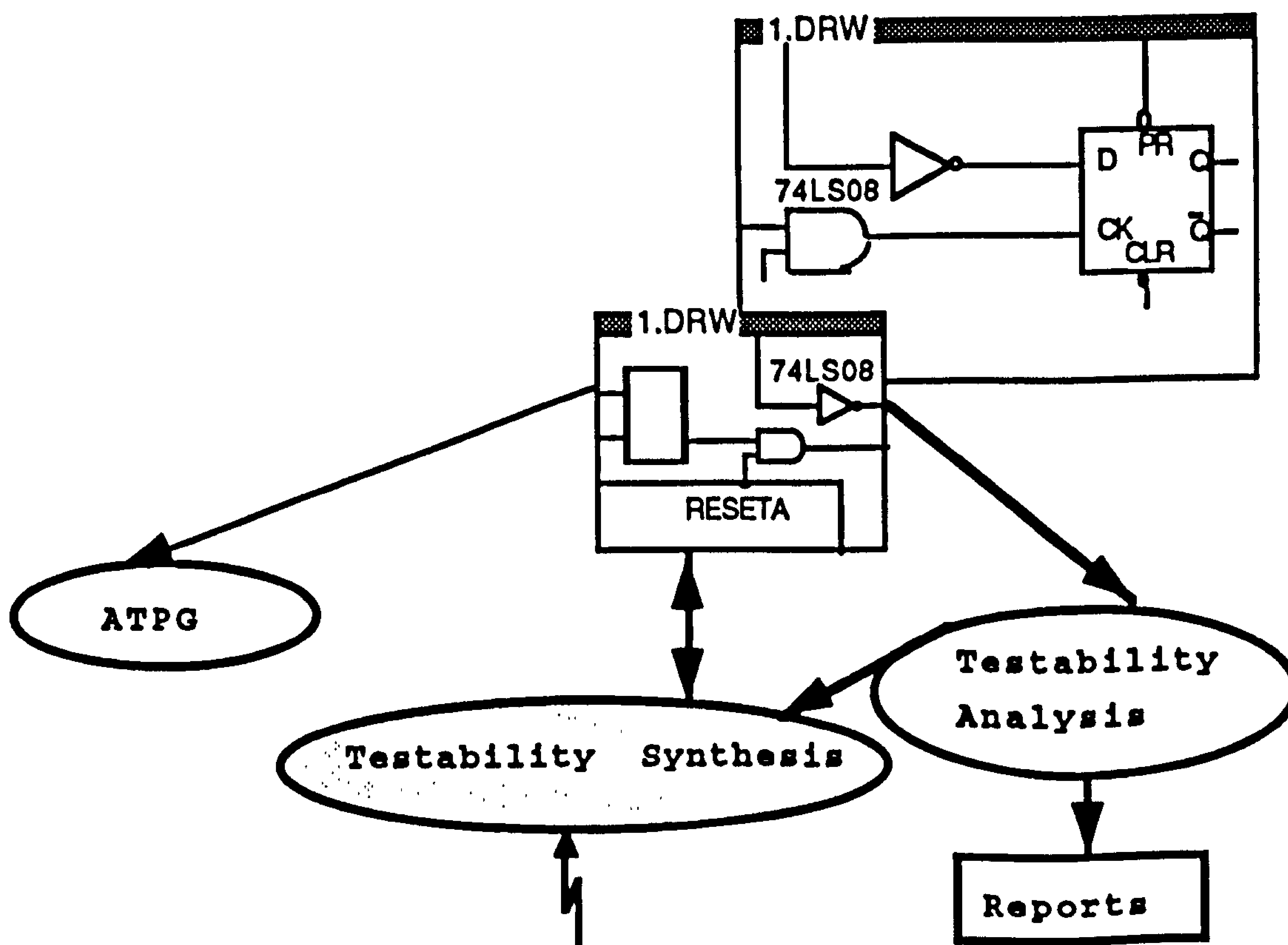



Figure 10.5 VCP and Synthesis

## 10.4 THE ARTIFICIAL INTELLIGENCE ROLE

A theory for exploiting the knowledge of the structure and the behaviour of a digital system using first principles has recently been developed by Digital Equipment Corp., the Defense Advanced Research Projects Agency (DARPA) and MIT in the United States.

Randall Davis of MIT [DAVI 89] has examined this theory in trouble-shooting digital electronic hardware. Davis claims that, a system based on reasoning from first principles is easier to construct because there is a way of systematically enumerating the required knowledge, the structure and behaviour of the device. The reasoning theory could be tested to deal with the difficulty that has arisen in developing high level test models, from a behavioural level to structural level, and not the other way around. Since this work is still in it's formative stages, there are still a number of questions about how accurate and successful this approach is.

**REFERENCES**


[Mori 88] Morris, C.J.Theaker, P.B.Whitehead, Structured Abstract Schematics, The Computer Journal Vol. 31, No. 3, 1988 pp 193 - 200.

[KUSZ 90] C.A.Kuszynski, T.Busfield, A.M Koelmans, M.R.McLauchlan, D.J. Kinniment, Graphical Representation of a Hardware Description Language, IEE proceedings, Vol 137, Pt.E, No 6,Nov 1990 pp. 462-468.

[Bain 88] J.Bain, STELLA, A Schematic Capture Tool for ELLA, Proceedings of the International Custom Microelectronics Conference,Nov 1988, Paper 36, pp.36.0-36.7.

[Rein 89] Auli Reinikka, Antti Auer, Ari Okkonen, Automatic Synthesis of Structural HDL Description from Graphic Specification of Embedded ASICs, Microprocessing and Microprogramming 27, 1989 pp. 473-478.

[AUER 88] A.Auer, P.Kemppainen, A.Okkonen, V.Seppanen, Automatic Code Generation of Embedded Real Time Systems, Microprocessing and Microprogramming 24, 1988, pp.51-56.

[OKKO 89] A.Okkonen, A.Auer, SOKRATES-SA: A Formal Specification Method for Real Time Systems. Microprocessing and Microprogramming 27, 1989.

[LEPP 89] T.Leppanen, Automatic Transformation from Structured Analysis to Hardware Description Language, MSc Thesis. University of Oulu 1989 (in Finnish).

[WASS 90] A.I.Wasserman, P.A.Pircher, R.J Muller, The Object Oriented Structured Design Notation for Software Design Representation, Computer Vol 3, March 1990.

[ACKR 91] M.Ackroyd, D.Daum, Graphical Notation for Object Oriented Design and Programming, Journal of Object-Oriented Programming, Vol 1, January 1991.

[WARD 86] P.T.Ward, S.J.Mellor, Structured Development for Real Time Systems, Vol 1-3. Yourdon Press, NY. 1986.

[GREE 86] D.Green, Modern Logic Design, Addison-Wesley, 1986.

[ARME 89] Arment E.L., and Coomb W.D., Application of JTAG for Digital and Analog SMT. Test 1989, ATE and Instrumental Conference. C.A. USA 1989.

# APPENDICES

## APPENDIX 4A

Simulation results of the VHDL 4-bit Multiplier described behaviourally.

VHDL LISTING OF 4 BIT SERIAL MULTIPLIER MODEL

```
VHDL Analyzer - V4.05; Workview 4.0, 3000 Series
Copyright (c) 1990 by Viewlogic Systems, Inc.
Analyzing MULT1.vhd; Making mult1.vsm mult1.vli
mult1.lis.
```

---

```
         1: ENTITY mult1 IS
         2:   PORT (a :IN vlbit_1d(0 TO 3);
         3:         b :IN vlbit_1d(0 TO 3);
         4:         q :OUT vlbit_1d(0 TO 7);
         5:         clock,start :IN vlbit);
         6: END mult1;
         7:
         8: ARCHITECTURE behav OF mult1 IS
         9:
        10: BEGIN
        11:
        12: calculate :PROCESS (clock)
        13:
        14: VARIABLE cycle : INTEGER := 0;
        15:
        16: BEGIN
        17:
        18:   IF (clock='1') AND (start='1') THEN
        19:     IF cycle=4 THEN
        20:       q <= mulum(a,b);
        21:       cycle := 0;
        22:     ELSE
        23:       cycle := cycle + 1;
        24:     END IF;
        25:   END IF;
        26:
        27: END PROCESS calculate;
        28:
        29: END behav;
```

```
0 errors; 0 warnings; 0 extensions; 0 notes.

VHDL Analysis of MULT1.vhd completed.
```

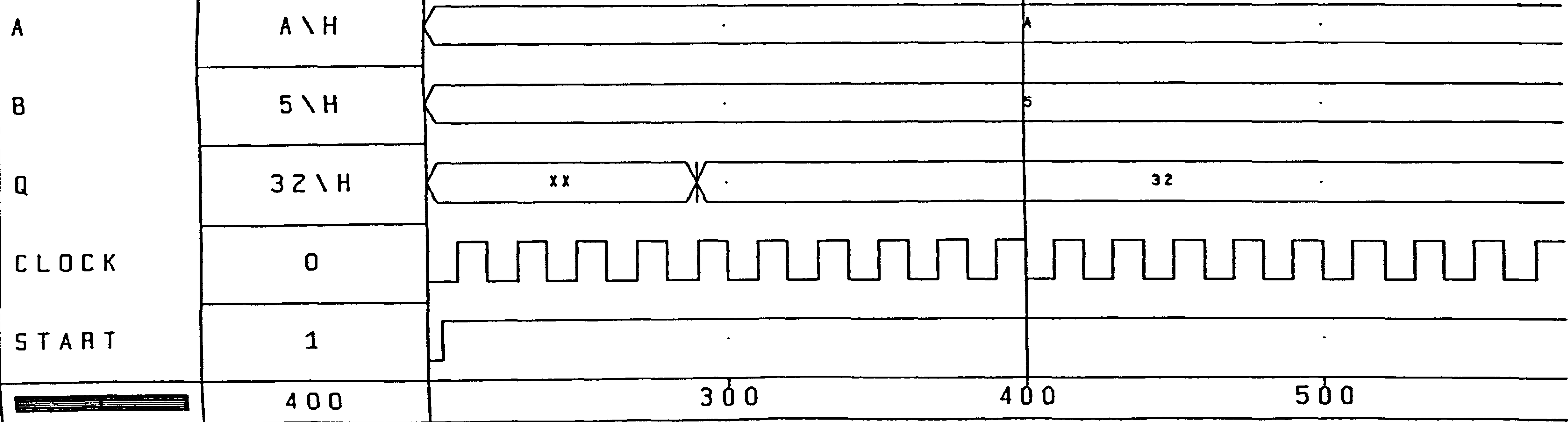
SIMULATION COMMAND FILE FOR THE MULTIPLIER - MULT1.CMD

---

```
vector a a[0:3]
vector b b[0:3]
vector q q[0:7]
watch a b q clock start
radix hex a b q
wfm a 0=0\h 100=A\h
wfm b 0=f\h 100=5\h
stepsize 100
clock clock 0 1
cycle 10
wfm start 0=0 50=1
wfm q
t a b q clock start
wave mult1.wfm a b q clock start
run 20
exit
```

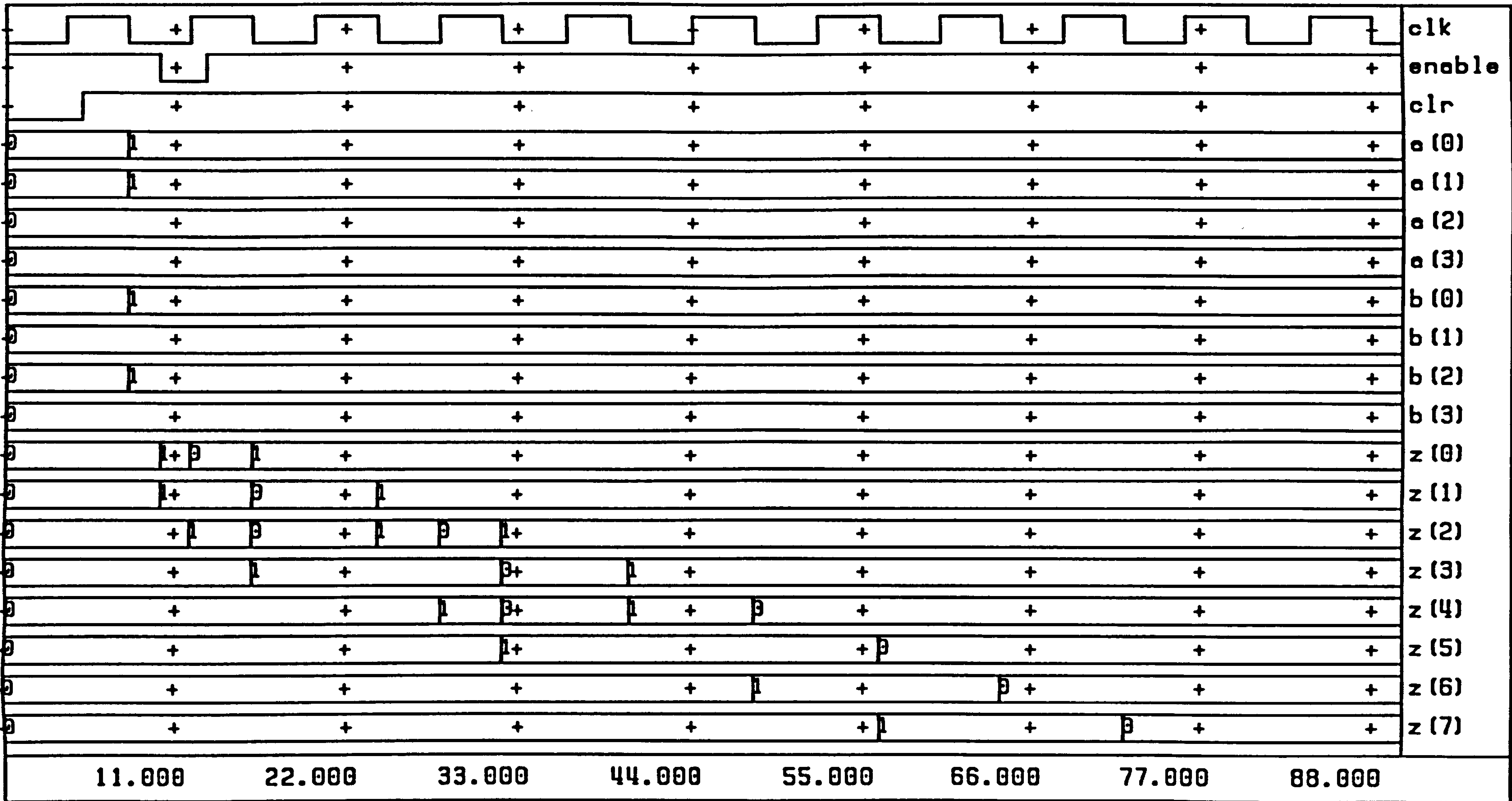
```
 Typical delays in use.
 All delays scaled by 1.
 Reading VHDL entity  file mult1.vli ...
 Reading VHDL package file
C:\workview\standard\vhdllibs\std\STANDARD.VLI ...
 Loading VHDL entity MULT1 ...
 Total of 1 digital modules were processed.
mult1
 time =    20.0ns  A=A\H B=5\H Q=XX\H CLOCK=1 START=X
 time =    40.0ns  A=A\H B=5\H Q=XX\H CLOCK=1 START=X
 time =    60.0ns  A=A\H B=5\H Q=XX\H CLOCK=1 START=X
 time =    80.0ns  A=A\H B=5\H Q=XX\H CLOCK=1 START=X
 time =   100.0ns  A=A\H B=5\H Q=XX\H CLOCK=1 START=X
 time =   120.0ns  A=A\H B=5\H Q=XX\H CLOCK=1 START=X
 time =   140.0ns  A=A\H B=5\H Q=XX\H CLOCK=1 START=X
 time =   160.0ns  A=A\H B=5\H Q=XX\H CLOCK=1 START=X
 time =   180.0ns  A=A\H B=5\H Q=XX\H CLOCK=1 START=X
 time =   200.0ns  A=A\H B=5\H Q=XX\H CLOCK=1 START=X
 Simulation stopped at 200.0ns.
 (mult1.cmd,12): There are no waveforms on 'q' to
delete!
 [time =     200.0ns]  CLOCK      1 -> 0
 [time =     200.0ns]  START      X -> 0
 [time =     205.0ns]  START      0 -> 1
 [time =     210.0ns]  CLOCK      0 -> 1
 time =   220.0ns  A=A\H B=5\H Q=XX\H CLOCK=1 START=1
 [time =     220.0ns]  CLOCK      1 -> 0
 [time =     230.0ns]  CLOCK      0 -> 1
 time =   240.0ns  A=A\H B=5\H Q=XX\H CLOCK=1 START=1
 [time =     240.0ns]  CLOCK      1 -> 0
 [time =     250.0ns]  CLOCK      0 -> 1
 time =   260.0ns  A=A\H B=5\H Q=XX\H CLOCK=1 START=1
 [time =     260.0ns]  CLOCK      1 -> 0
 [time =     270.0ns]  CLOCK      0 -> 1
 time =   280.0ns  A=A\H B=5\H Q=XX\H CLOCK=1 START=1
 [time =     280.0ns]  CLOCK      1 -> 0
 [time =     290.0ns]  CLOCK      0 -> 1
 [time =     290.0ns]  Q          XX\H -> XX\H
 [time =     290.0ns]  Q          XX\H -> XX\H
 [time =     290.0ns]  Q          XX\H -> XX\H
 [time =     290.0ns]  Q          XX\H -> 3X\H
 [time =     290.0ns]  Q          3X\H -> 3X\H
 [time =     290.0ns]  Q          3X\H -> 3X\H
 [time =     290.0ns]  Q          3X\H -> 3X\H
 [time =     290.0ns]  Q          3X\H -> 32\H
 time =   300.0ns  A=A\H B=5\H Q=32\H CLOCK=1 START=1
 [time =     300.0ns]  CLOCK      1 -> 0
 [time =     310.0ns]  CLOCK      0 -> 1
 time =   320.0ns  A=A\H B=5\H Q=32\H CLOCK=1 START=1
 [time =     320.0ns]  CLOCK      1 -> 0
 [time =     330.0ns]  CLOCK      0 -> 1
 time =   340.0ns  A=A\H B=5\H Q=32\H CLOCK=1 START=1
 [time =     340.0ns]  CLOCK      1 -> 0
 [time =     350.0ns]  CLOCK      0 -> 1
 time =   360.0ns  A=A\H B=5\H Q=32\H CLOCK=1 START=1
 [time =     360.0ns]  CLOCK      1 -> 0
 [time =     370.0ns]  CLOCK      0 -> 1
 time =   380.0ns  A=A\H B=5\H Q=32\H CLOCK=1 START=1
```

```
 [time =     380.0ns]  CLOCK      1 -> 0
 [time =     390.0ns]  CLOCK      0 -> 1
 time =   400.0ns  A=A\H B=5\H Q=32\H CLOCK=1 START=1
 [time =     400.0ns]  CLOCK      1 -> 0
 [time =     410.0ns]  CLOCK      0 -> 1
 time =   420.0ns  A=A\H B=5\H Q=32\H CLOCK=1 START=1
 [time =     420.0ns]  CLOCK      1 -> 0
 [time =     430.0ns]  CLOCK      0 -> 1
 time =   440.0ns  A=A\H B=5\H Q=32\H CLOCK=1 START=1
 [time =     440.0ns]  CLOCK      1 -> 0
 [time =     450.0ns]  CLOCK      0 -> 1
 time =   460.0ns  A=A\H B=5\H Q=32\H CLOCK=1 START=1
 [time =     460.0ns]  CLOCK      1 -> 0
 [time =     470.0ns]  CLOCK      0 -> 1
 time =   480.0ns  A=A\H B=5\H Q=32\H CLOCK=1 START=1
 [time =     480.0ns]  CLOCK      1 -> 0
 [time =     490.0ns]  CLOCK      0 -> 1
 time =   500.0ns  A=A\H B=5\H Q=32\H CLOCK=1 START=1
 [time =     500.0ns]  CLOCK      1 -> 0
 [time =     510.0ns]  CLOCK      0 -> 1
 time =   520.0ns  A=A\H B=5\H Q=32\H CLOCK=1 START=1
 [time =     520.0ns]  CLOCK      1 -> 0
 [time =     530.0ns]  CLOCK      0 -> 1
 time =   540.0ns  A=A\H B=5\H Q=32\H CLOCK=1 START=1
 [time =     540.0ns]  CLOCK      1 -> 0
 [time =     550.0ns]  CLOCK      0 -> 1
 time =   560.0ns  A=A\H B=5\H Q=32\H CLOCK=1 START=1
 [time =     560.0ns]  CLOCK      1 -> 0
 [time =     570.0ns]  CLOCK      0 -> 1
 time =   580.0ns  A=A\H B=5\H Q=32\H CLOCK=1 START=1
 [time =     580.0ns]  CLOCK      1 -> 0
 [time =     590.0ns]  CLOCK      0 -> 1
 time =   600.0ns  A=A\H B=5\H Q=32\H CLOCK=1 START=1
 Simulation stopped at 600.0ns.
run
 [time =     600.0ns]  CLOCK      1 -> 0
 [time =     610.0ns]  CLOCK      0 -> 1
 time =   620.0ns  A=A\H B=5\H Q=32\H CLOCK=1 START=1
 Simulation stopped at 620.0ns.
```


SIMULATION RESULTS OF THE 4 BIT MULTIPLIER
USING VHDL BEHAVIOURAL MODEL
(VIEW LOGIC SYSTEM RUNNING ON PC386).
A
A\H
B
5\H
Q
32\H
CLOCK
0
START
1
400
XX
32
300
400
500

## APPENDIX 4B

Simulation results of the 4-bit Multiplier described structurally




MULTIPLIER WAVE PLOT - Structural Simulation

NUMBERS LOADED 3 x 5 = 15

21 April 1991

**APPENDIX 5A**

This appendix includes the design and simulation of the Boundary Scan Architecture. 4 test instructions have been used in this case and include the following:

-NOP
-SAMPLE
-EXTEST
-BYPASS

The instructions have been generated using Macro-Functions available on the Mentor CAE system to allow ease of simulation.The Macro-Functions can easily be re-complied in 'C' to enable transportability of stimuli.

The Macro-Functions include the following:

- START — The macro START has to be executed at the beginning of a simulation run. It sets up the TAP-Controller in the SELECT-DR-SCAN state.

- LDI_NOP — Loads the instruction NOP into the instruction register.

- LDI_SAMPLE — Loads the Instruction SAMPLE into the instruction register.

- LDI_EXTEST — Loads the instruction EXTEST into the instruction register.

- LDI_BYPASS — Loads the instruction BYPASS into the instruction register.

- INTEST [nb] — Loads the instruction INTEST into the instruction register and executes INTEST. The nb argument is

used to determine the number of clock cycles to wait in the RUN/IDLE-TEST state. The default value of nb is 1.

- RUNBIST [nb] Loads the instruction RUNBIST into the instruction register and executes RUNBIST . The nb argument is used to determine the number of clock cycles to wait in the RUN/IDLE-TEST state. The default value of nb is 1.

- LD_DATA data [nb]

  Loads DATA on the TDI. The data argument is preceded by $ or % in order to specify whether the data format is Hex($) or Binary(%). The optional nb argument limits the amount of data to be entered in serially on TDI. By default nb is equal to the length of the data argument.

  e.g.

  do ld_data.do %10010 *means load TDI with 5 data values. The first is 1.*

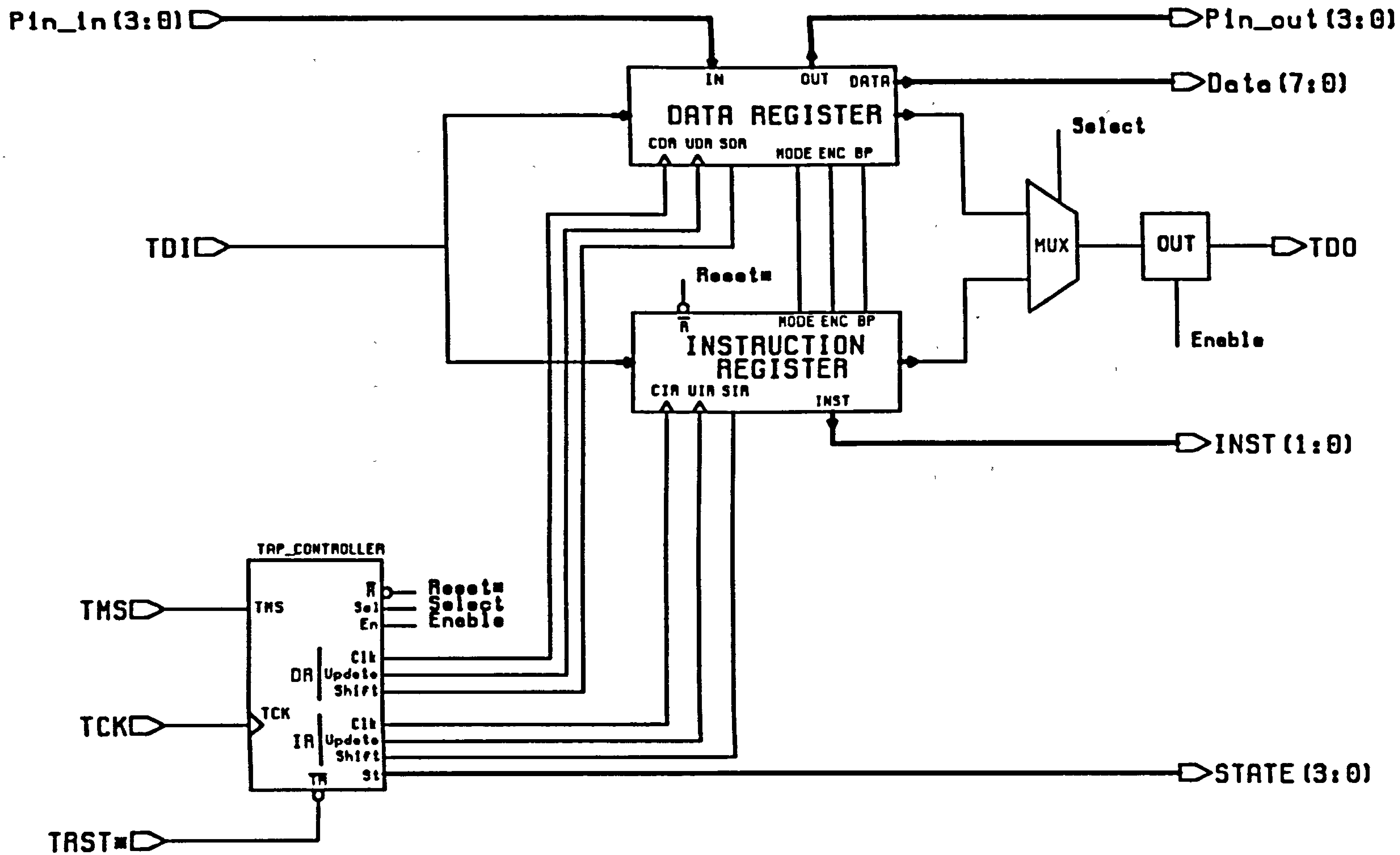
  do ld_data.do $0 equals do ld_data.do %0000
  do ld_data.do $0 2 equals do ld_data.do %00

```
# LoaD data into the the Data register

# Processing and checks
if ^($arg_1='') then
   write line EXIT : missing the number to enter -space
   exit macro
end if
assign nb ^$arg_1
assign nb_lengt ^$size(nb)
loop i 1 ^nb_lengt
     assign nb[i] ^$toupper(nb[i])
end loop
if ^(nb[1]='%') then
   assign data_lengt ^(nb_lengt-1)
   dim data ^(data_lengt)
   loop i 2 ^nb_lengt
        if ^((nb[i]='1') or (nb[i]='0')) then
           assign data[i-1] ^nb[i]
           else
           write line EXIT : data error in argument_1 -space
           exit macro
        end if
   end loop
   elseif ^(nb[1]='$') then
           assign user ^$arg_2
           if ^(user<>'') then
              assign user_lengt ^$size(user)
              loop i 1 ^user_lengt
                   if ^(user[i]<'0' or user[i]>'9') then
                      write line EXIT : data error in argument_2
                      exit macro
                   end if
              end loop
              assign data_lengt ^user
              else
              assign data_lengt ^(4*(nb_lengt-1))
          end if
          dim data ^(4*(nb_lengt-1))
          dim nb_bin 4
          loop i 2 ^nb_lengt
               if ^((nb[i]>='0' and nb[i]<='9') or (nb[i]>='A' and nb[i]<
                  assign nb_bin ^$convert_radix(nb[i],16,2)
                  assign nb_bin_lengt ^$size(nb_bin)
                  assign offset ^(4-nb_bin_lengt)
                  loop j 1 ^nb_bin_lengt
                       assign row ^((i-2)*4+j+offset)
                       assign data[row] ^nb_bin[j]
                  end loop
                  if ^(offset<>0) then
                     loop j 1 ^offset
                          assign row ^((i-2)*4+j)
                          assign data[row] 0
                     end loop
                  end if
```

```
                    else
                    write line EXIT : data error in argument_1 -space
                    exit macro
                 end if
            end loop
            else
            write line EXIT : the argument_1 must be binary(%) or hexa($) -
            exit macro
end if

# Sending of Stimuli
    FORCe tms 0      # 7
    RUN 10
    RUN 10          # 6
    loop i 1 ^(data_lengt-1)
         FORCe TDI ^data[i]    #2
         RUN 10
    end loop
    FORCe tms 1      # 2
    FORCe TDI ^data[data_lengt]
    RUN 10
    FORCe TDI 0      # 1
    RUN 10
    RUN 10           # 5
                     # 7
```

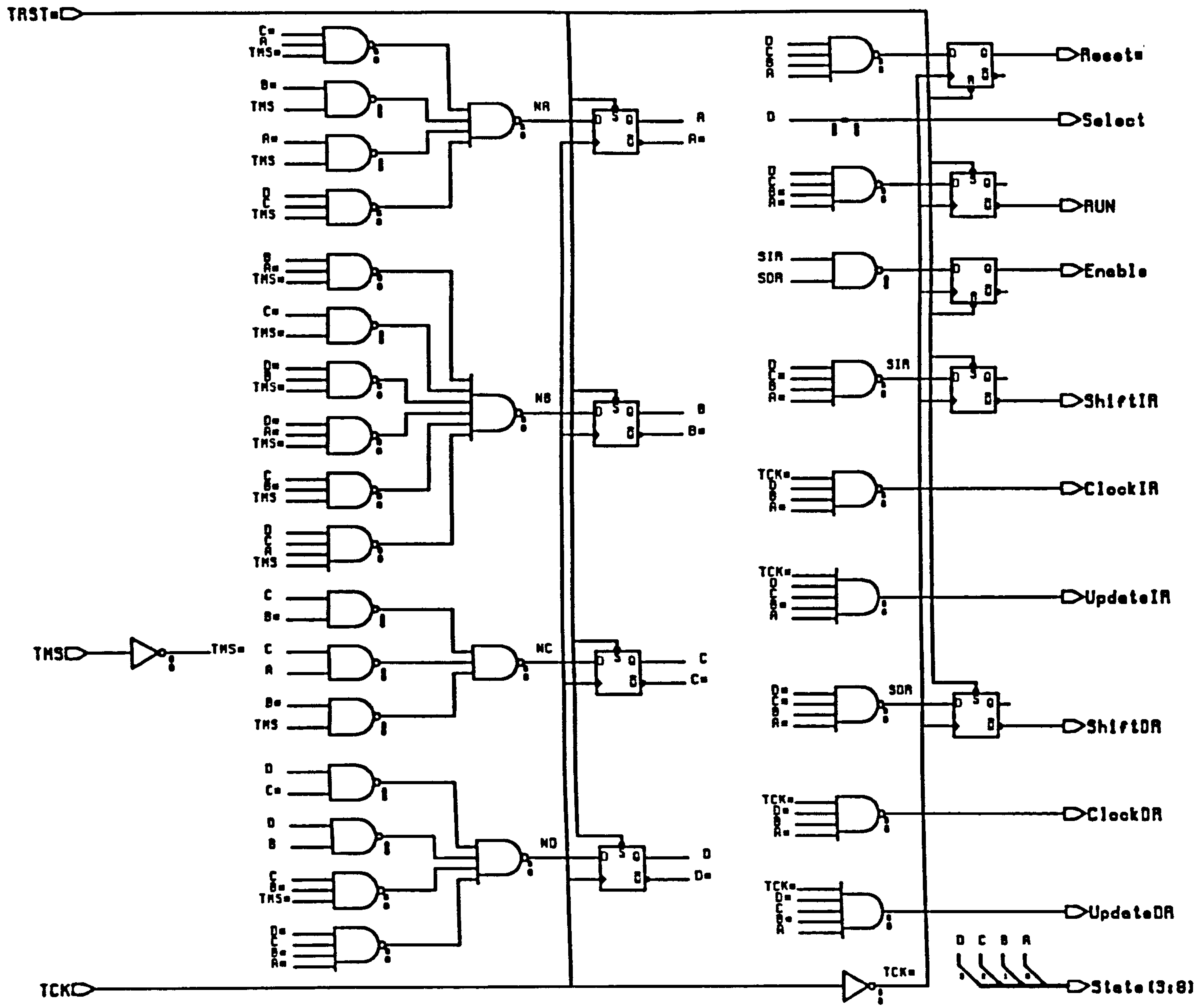
```
# LoaD Instruction RUNBIST into the instruction register

# Processing and checks
assign tempo ^$arg_1
if ^(tempo<>'' and tempo<>'0') then
   assign tempo_lengt ^$size(tempo)
   loop i 1 ^tempo_lengt
        if ^(tempo[i]<'0' or tempo[i]>'9') then
           write line EXIT : data error in argument_1 -space
           exit macro
        end if
   end loop
else
   assign tempo 1
end if

# Sending of Stimuli
    FORCe tms 1      # 7
    RUN 10
    FORCe tms 0      # 4
    RUN 10
    RUN 10           # E
    FORCe TDI 1      # A
    RUN 10
    RUN 10           # A
    FORCe tms 1      # A
    FORCe TDI 0
    RUN 10
    RUN 10           # 9
    FORCe tms 0      # D
    RUN ^(10*tempo-1) # C
    FORCe tms 1
    RUN 10
                    # 7
```

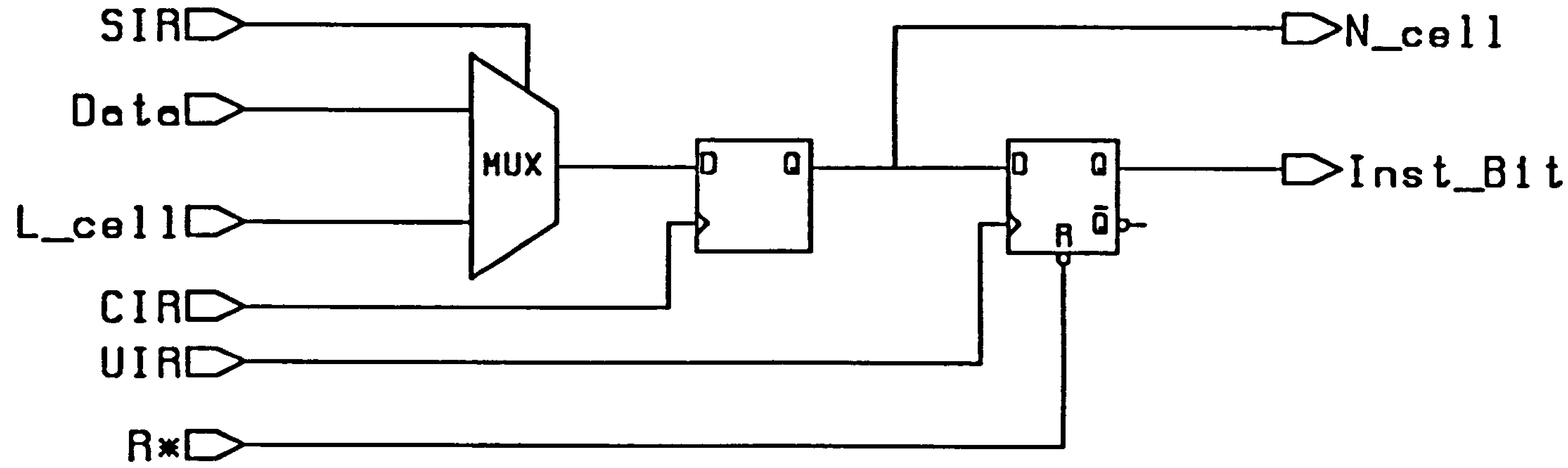
```
# LoaD Instruction INTEST into the instruction register

# Processing and checks
assign tempo ^$arg_1
if ^(tempo<>'' and tempo<>'0') then
   assign tempo_lengt ^$size(tempo)
   loop i 1 ^tempo_lengt
        if ^(tempo[i]<'0' or tempo[i]>'9') then
           write line EXIT : data error in argument_1 -space
           exit macro
        end if
   end loop
else
   assign tempo 1
end if

# Sending of Stimuli
    FORCe tms 1      # 7
    RUN 10
    FORCe tms 0      # 4
    RUN 10
    RUN 10           # E
    FORCe TDI 0      # A
    RUN 10
    FORCe TDI 1      # A
    RUN 10
    FORCe tms 1      # A
    FORCe TDI 0
    RUN 10
    RUN 10           # 9
    FORCe tms 0      # D
    RUN ^(10*tempo-1) # C
    FORCe tms 1
    RUN 10
    FORCe tms 0      # 7
    RUN 10
    FORCe tms 1      # 6
    RUN 10
    RUN 10           # 1
    RUN 10           # 5
                     # 7
```



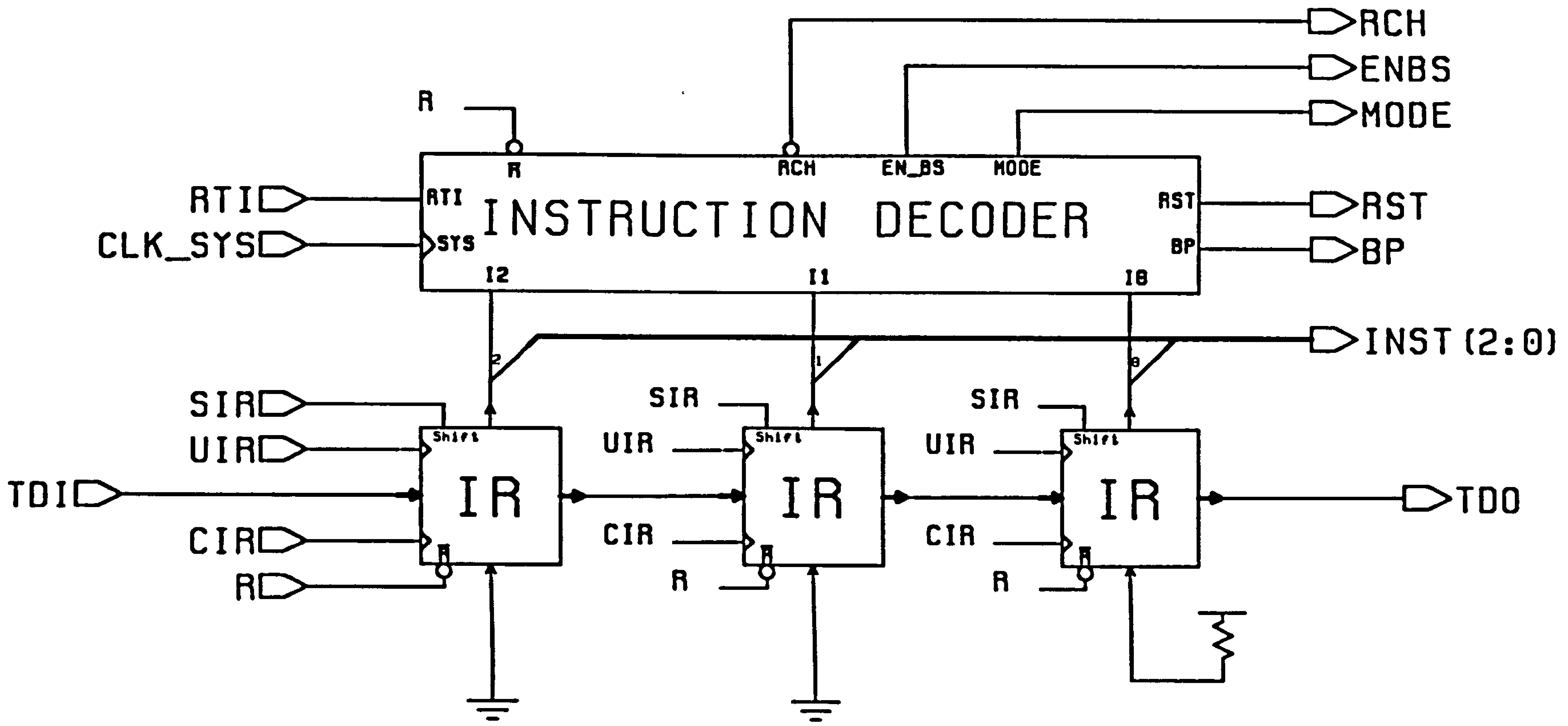

The JTAG boundary-scan architecture
for the test of functionality
(version whith 4 instructions)

# TAP-Controller

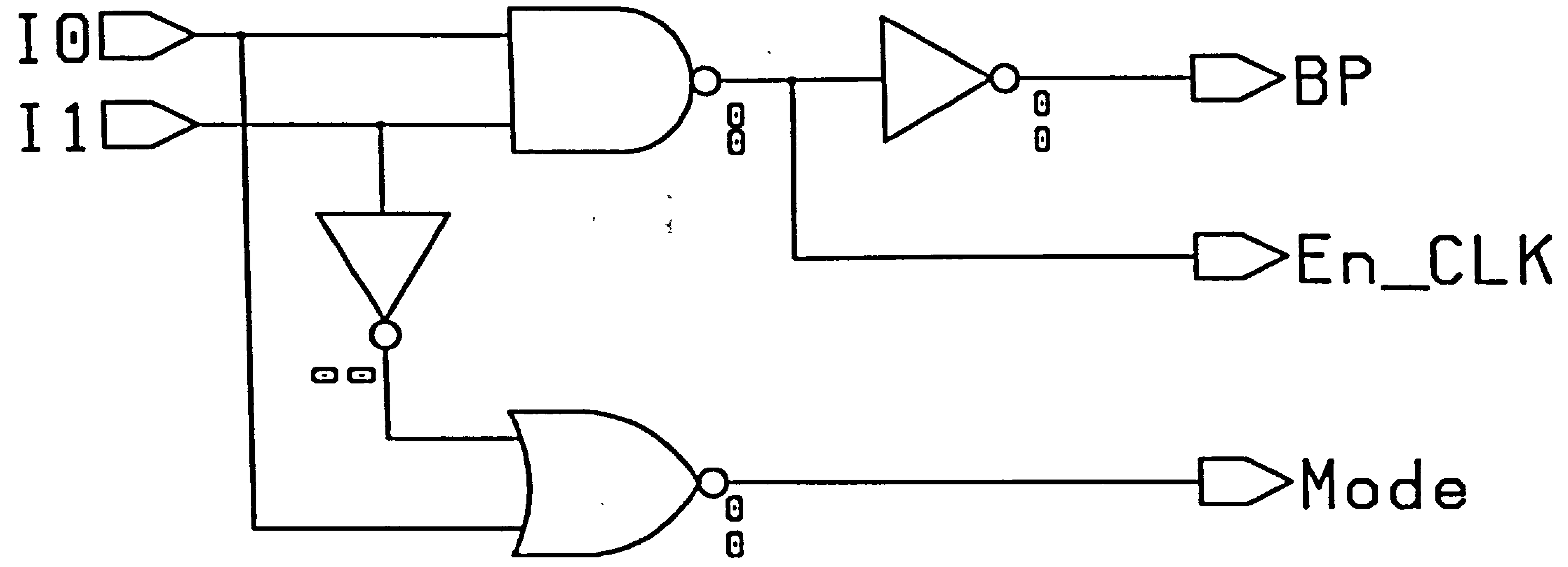




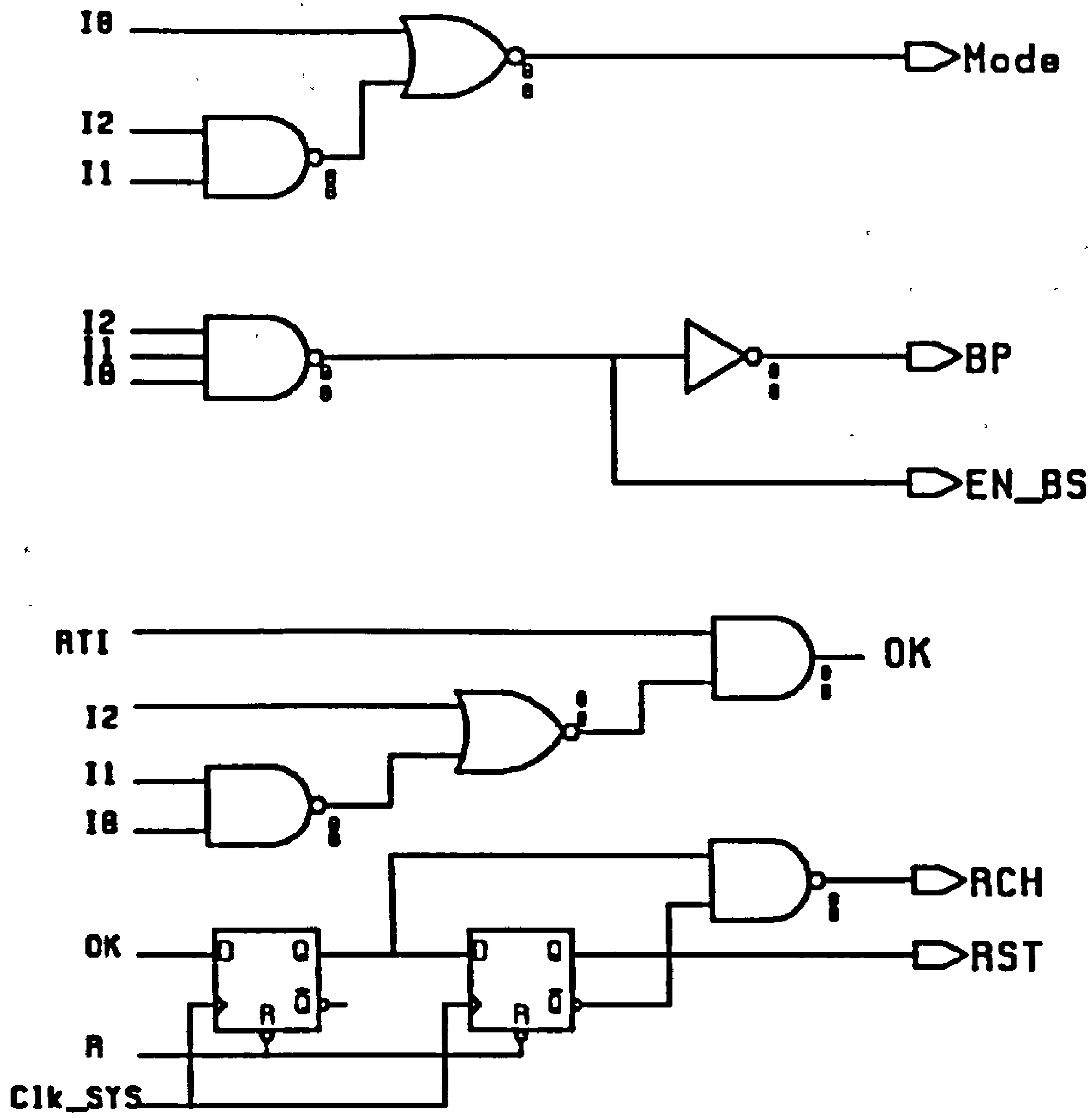

Instruction register cell



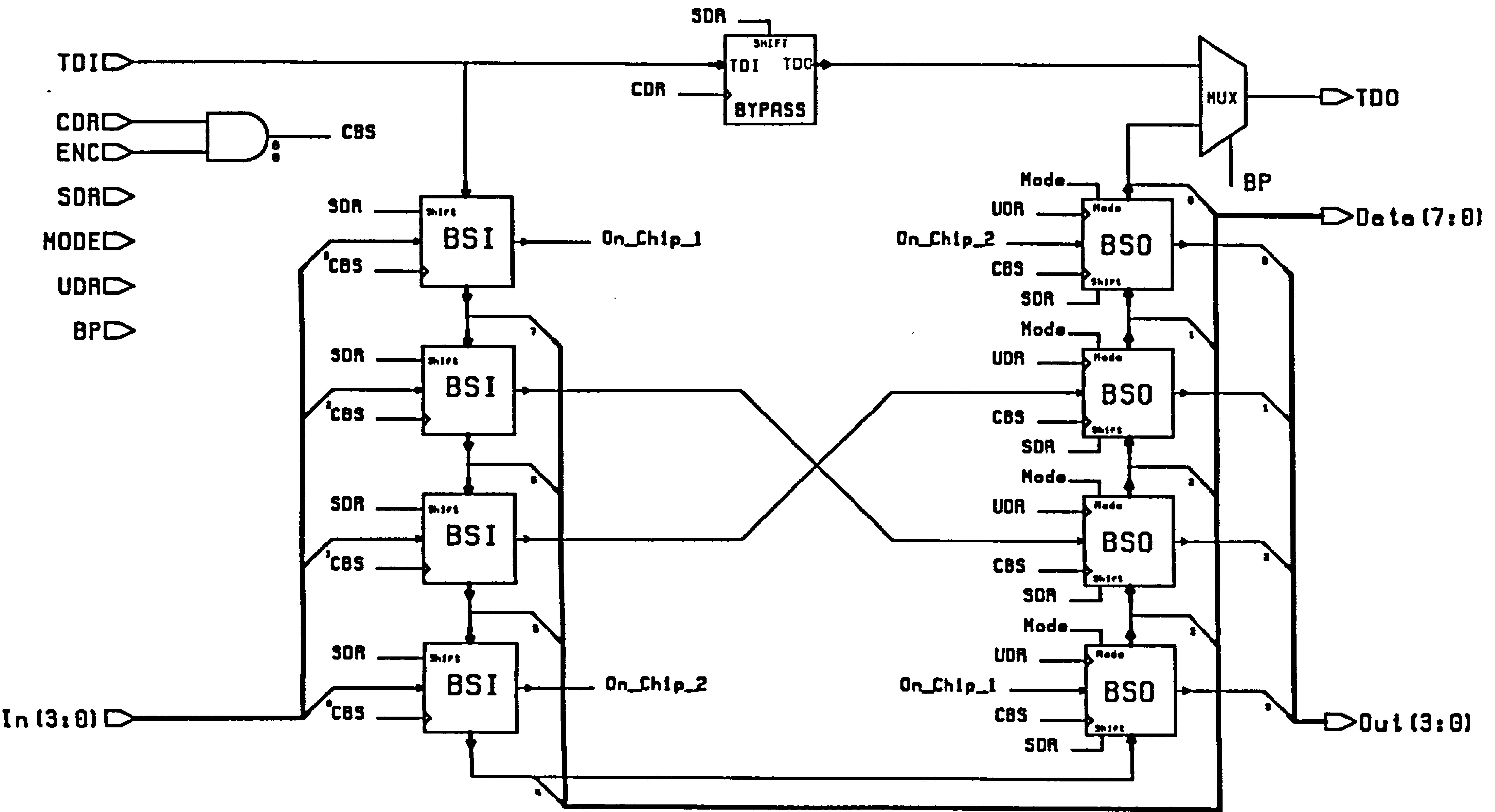

Instruction register



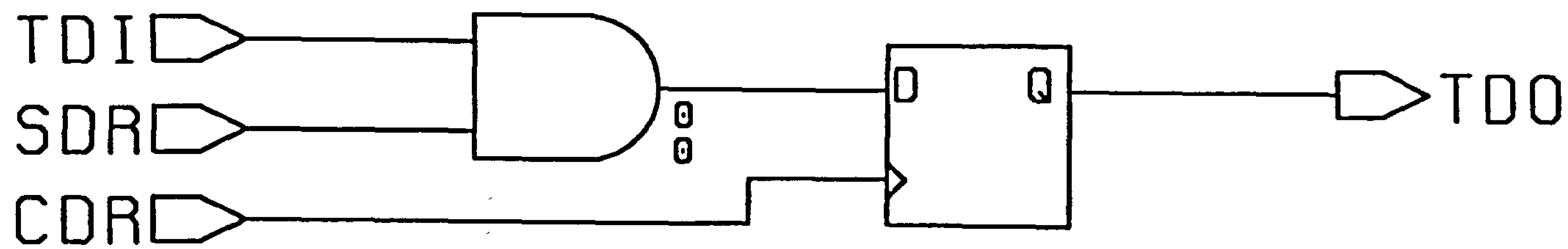

Instruction decoder



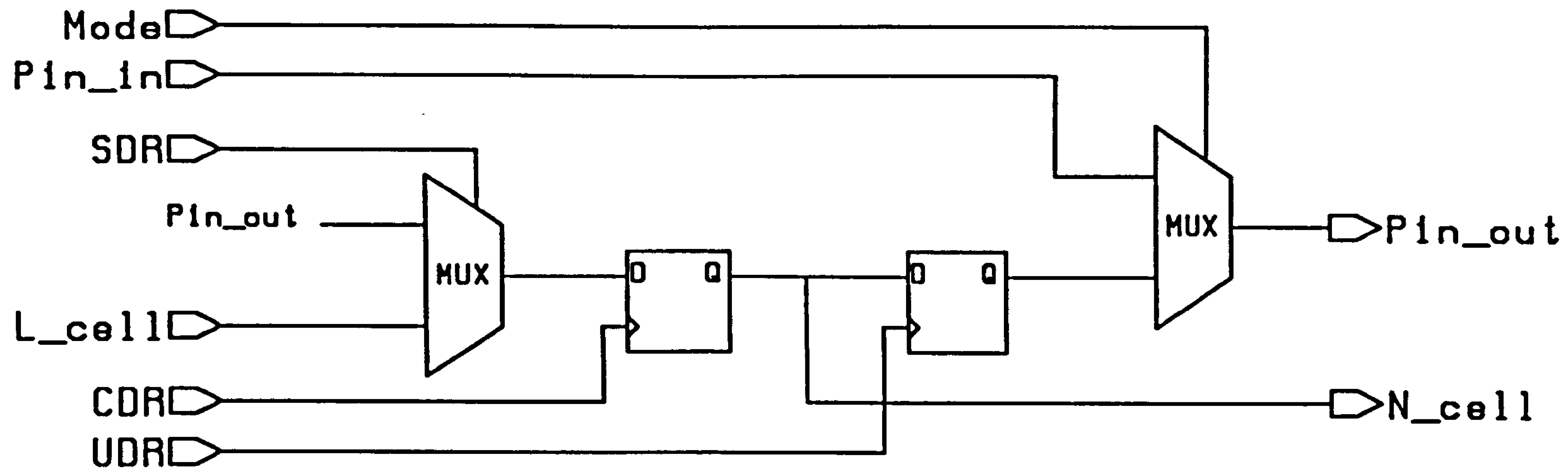

Instruction decoder



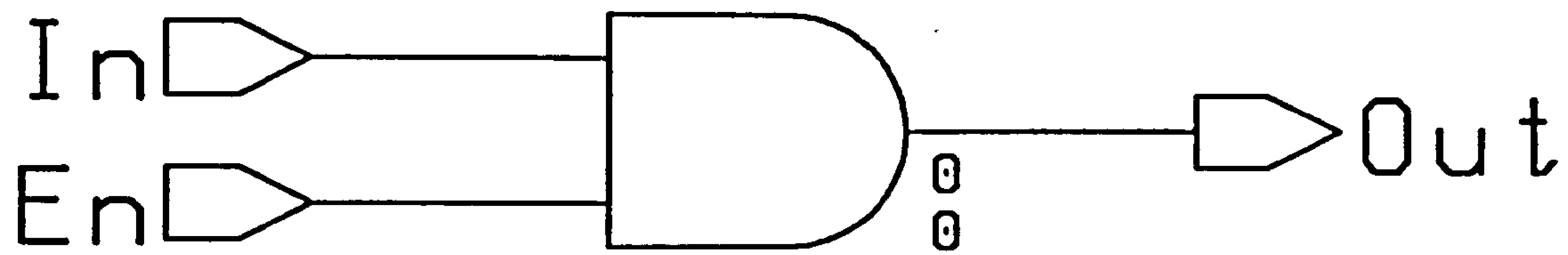

Data Register 2 states (4 in, 4 out)




Bypass register cell




Boundary-scan cell for an output pin




Output Buffer

```
# Functionnal test of the BYPASS basic cell
    RADix Binary
    ASSIgn hi_$list_radix Binary
    ASSIgn hi_$monitor_radix Binary
    TRAce CDR
    LISt -Change Binary CDR
    MONitor Binary CDR
    TRAce TDI
    LISt -Change Binary TDI
    MONitor Binary TDI
    TRAce SDR
    LISt -Change Binary SDR
    MONitor Binary SDR
    TRAce TDO
    LISt -Change Binary TDO
    MONitor Binary TDO
    SCAle USer Time 1
    SCAle TRace Time 100
    INItialize XR
    VIEw Sheet sheet1 /
    PERiod List 0
    PERiod Trace 0
    CLOck Period 50
    FORCe CDR 0 0 -Repeat
    FORCe cdr 1 25 -Repeat
    FORCe tdi 1
    FORCe sdr 0
    RUN 50
    FORCe sdr 1
    RUN 50
    FORCe tdi 0
    RUN 50
    FORCe tdi 1
    RUN 25
    RUN 25
    FORCe sdr 0
    RUN 25
    RUN 25
```

```
# Functionnal test of the boundary-Scan cell for an input pin
    RADix Binary
    ASSIgn hi_$list_radix Binary
    ASSIgn hi_$monitor_radix Binary
    TRAce CDR
    LISt -Change Binary CDR
    MONitor Binary CDR
    TRAce SDR
    LISt -Change Binary SDR
    MONitor Binary SDR
    TRAce Pin_in
    LISt -Change Binary Pin_in
    MONitor Binary Pin_in
    TRAce L_Cell
```

```
LISt -Change Binary L_Cell
MONitor Binary L_Cell
TRAce Pin_out
LISt -Change Binary Pin_out
MONitor Binary Pin_out
TRAce N_Cell
LISt -Change Binary N_Cell
MONitor Binary N_Cell
SCAle USer Time 1
SCAle TRace Time 50
INItialize XR
PERiod List 0
PERiod Trace 0
CLOck Period 50
FORCe cdr 0 0 -Repeat
FORCe cdr 1 25 -Repeat
FORCe sdr 0
FORCe l_cell 1
FORCe pin_in 0
RUN 50
FORCe pin_in 0s
RUN 10
FORCe sdr 1
RUN 40
FORCe pin_in 1
RUN 10
FORCe l_cell 0
RUN 10
RUN 30
FORCe pin_in 1
RUN 10
FORCe sdr 0
RUN 40


# Functionnal test of the boundary-Scan cell for an output pin
RADix Binary
ASSIgn hi_$list_radix Binary
ASSIgn hi_$monitor_radix Binary
TRAce CDR
LISt -Change Binary CDR
MONitor Binary CDR
TRAce UDR
LISt -Change Binary UDR
MONitor Binary UDR
TRAce Mode
LISt -Change Binary Mode
MONitor Binary Mode
TRAce SDR
LISt -Change Binary SDR
MONitor Binary SDR
TRAce L_Cell
LISt -Change Binary L_Cell
MONitor Binary L_Cell
```

```
TRAce N_cell
LISt -Change Binary N_cell
MONitor Binary N_cell
TRAce Pin_in
LISt -Change Binary Pin_in
MONitor Binary Pin_in
TRAce Pin_out
LISt -Change Binary Pin_out
MONitor Binary Pin_out
SCAle USer Time 1
SCAle TRace Time 100
INItialize XR
VIEw Sheet sheet1 /
PERiod List 0
PERiod Trace 0
CLOck Period 100
FORCe cdr 0 0 -Repeat
FORCe cdr 1 80 -Repeat
FORCe udr 0
FORCe mode 1
FORCe sdr 0
FORCe l_cell 1
FORCe pin_in 0
RUN 100
SCAle TRace Time 100
FORCe mode 0
RUN 20
FORCe pin_in 1
RUN 20
FORCe pin_in 0
RUN 20
RUN 40
FORCe pin_in 1
RUN 20
FORCe mode 1
RUN 20
FORCe udr 1
RUN 20
FORCe l_cell 0
RUN 20
FORCe sdr 1
FORCe l_cell 1
RUN 20
RUN 100
FORCe mode 0
RUN 20
FORCe udr 0
FORCe mode 1
RUN 20
FORCe udr 1
RUN 20
FORCe l_cell 0
RUN 40
FORCe udr 0
```

```
RUN 20
FORCe pin_in 0
RUN 20
FORCe mode 0
RUN 20
FORCe l_cell 1
RUN 40
FORCe sdr 0
RUN 100
```

```
# Functionnal test of the Instruction register basic cell
    RADix Binary
    ASSIgn hi_$list_radix Binary
    ASSIgn hi_$monitor_radix Binary
    TRAce R*
    LISt -Change Binary R*
    MONitor Binary R*
    TRAce CIR
    LISt -Change Binary CIR
    MONitor Binary CIR
    TRAce SIR
    LISt -Change Binary SIR
    MONitor Binary SIR
    TRAce UIR
    LISt -Change Binary UIR
    MONitor Binary UIR
    TRAce Data
    LISt -Change Binary Data
    MONitor Binary Data
    TRAce L_Cell
    LISt -Change Binary L_Cell
    MONitor Binary L_Cell
    TRAce N_Cell
    LISt -Change Binary N_Cell
    MONitor Binary N_Cell
    TRAce Inst_Bit
    LISt -Change Binary Inst_Bit
    MONitor Binary Inst_Bit
    SCAle USer Time 1
    SCAle TRace Time 50
    INItialize XR
    VIEw Sheet sheet1 /
    PERiod List 0
    PERiod Trace 0
    CLOck Period 50
    FORCe cir 0 0 -Repeat
    FORCe cir 1 25 -Repeat
    FORCe r* 1
    FORCe sir 0
    FORCe uir 0
    FORCe data 1
    FORCe l_cell 0
    RUN 10
```

```
FORCe r* 0
RUN 10
FORCe r* 1
RUN 30
FORCe uir 1
RUN 10
FORCe sir 1
RUN 40
FORCe uir 0
RUN 10
FORCe uir 1
RUN 10
FORCe l_cell 1
RUN 10
FORCe uir 0
RUN 10
FORCe uir 1
RUN 10
FORCe sir 0
RUN 10
RUN 20
FORCe data 0
RUN 50
FORCe uir 0
RUN 10
FORCe uir 1
RUN 10
```

```
# Functional Test of the TAP_CONTROLLER
# Setup of the windows
    RADix Hex
    ASSIgn hi_$list_radix Hex
    ASSIgn hi_$monitor_radix Hex
    TRAce TRST*
    LISt -Change Hex TRST*
    MONitor Hex TRST*
    TRAce TCK
    LISt -Change Hex TCK
    MONitor Hex TCK
    TRAce TMS
    LISt -Change Hex TMS
    MONitor Hex TMS
    TRAce STATE
    LISt -Change Hex STATE
    MONitor Hex STATE
    TRAce RESET*
    LISt -Change Hex RESET*
    MONitor Hex RESET*
    TRAce SELECT
    LISt -Change Hex SELECT
    MONitor Hex SELECT
    TRAce ENABLE
    LISt -Change Hex ENABLE
    MONitor Hex ENABLE
    TRAce SHIFTIR
    LISt -Change Hex SHIFTIR
    MONitor Hex SHIFTIR
    TRAce CLOCKIR
    LISt -Change Hex CLOCKIR
    MONitor Hex CLOCKIR
    TRAce UPDATEIR
    LISt -Change Hex UPDATEIR
    MONitor Hex UPDATEIR
    TRAce SHIFTDR
    LISt -Change Hex SHIFTDR
    MONitor Hex SHIFTDR
    TRAce CLOCKDR
    LISt -Change Hex CLOCKDR
    MONitor Hex CLOCKDR
    TRAce UPDATEDR
    LISt -Change Hex UPDATEDR
    MONitor Hex UPDATEDR
    HIStory 10000
#   HISTORY Threshold = 0.0 (History 10000.0 -NOABS).  Current time = 0.0
    SCAle USer Time 1
    SCAle TRace Time 200
    INItialize XR
    VIEw Sheet sheet1 /
    PERiod List 0
    PERiod Trace 0
# Stimulis' Setup
   CLOck Period 50
```

```
    FORCe tck 0 0 -Repeat
    FORCe tck 1 25 -Repeat
    FORCe trst* 0
    FORCe tms 0
# Initialisation on the F state
    RUN 50
    RUN 50
    FORCe tms 1
    RUN 50
    FORCe trst* 1
    RUN 50
    RUN 50

# main loop' check then return in F
    FORCe tms 0
    RUN 50
    RUN 50
    RUN 50
    FORCe tms 1
    RUN 50
    RUN 50
    RUN 50
    RUN 50
    RUN 50
    FORCe tms 0
    RUN 50
    FORCe tms 1
    RUN 50

# DR loop' check ( exhaustif check) then return in C
    FORCe tms 0
    RUN 50
    RUN 50
    RUN 50
    RUN 50
    FORCe tms 1
    RUN 50
    FORCe tms 0
    RUN 50
    RUN 50
    FORCe tms 1
    RUN 50
    FORCe tms 0
    RUN 50
    FORCe tms 1
    RUN 50
    FORCe tms 0
    RUN 50
    FORCe tms 1
    RUN 50
    RUN 50
    FORCe tms 0
    RUN 50
    FORCe tms 1
```

```
RUN 50

# DR loop' check (short check) then return in 7
FORCe tms 0
RUN 50
FORCe tms 1
RUN 50
RUN 50
RUN 50
RUN 50

# IR loop' check ( exhaustif check) then return in C
FORCe tms 0
RUN 50
RUN 50
RUN 50
RUN 50
FORCe tms 1
RUN 50
FORCe tms 0
RUN 50
RUN 50
FORCe tms 1
RUN 50
FORCe tms 0
RUN 50
FORCe tms 1
RUN 50
FORCe tms 0
RUN 50
FORCe tms 1
RUN 50
RUN 50
FORCe tms 0
RUN 50
FORCe tms 1
RUN 50
RUN 50

# IR loop' check (short check) then return in 7
FORCe tms 0
RUN 50
FORCe tms 1
RUN 50
RUN 50
RUN 50
RUN 50
RUN 50
RUN 50
MARK -2006.0,-59.0,Trace
VIEw ALL
```

```
# LoaD data into the the Data register
   TRANSCRIPTing OFF
   assi prompt1 'Which level would you like for TDI (1/0/return) '
   FORCe tms 0     # 7
   RUN 10
   RUN 10          # 6
   input ^prompt1 level  # 2
   if ^((level='1') or (level='0')) then
       FORCe TDI ^level
   end if
   loop
     input ^prompt1 level # 2
     if ^((level<>'1') and (level<>'0')) then
            exit loop
        else
            RUN 10
            FORCe TDI ^level
     end if
   end loop
   FORCe tms 1      # 2
   RUN 10
   FORCe TDI 0      # 1
   RUN 10
   RUN 10           # 5
                    # 7
```

```
# Setup Window File
  VIEw Sheet sheet1 /
  MARK -2.5,0.7,View
  DO /idea/sys/hi/macro/analysis/view_down I$243
  MARK -Rectangle -3.0,1.9,View
  VIEw ARea -0.8,0.2,View
  PROBe toto_1 -1.1,1.2,View
  PROBe toto_2 -0.1,0.2,View
  PROBe toto_3 -0.1,-0.8,View
  PROBe toto_4 -1.1,-1.8,View
  DO /idea/sys/hi/macro/analysis/view_up
  DEFine Bus on_chip /I$243/toto_4 /I$243/toto_3 /I$243/toto_2 /I$243/tc
  PROBe Reset_CH 0.6,2.2,View
  PROBe Run_S_T 0.4,1.7,View
  PROBe mode 0.8,1.7,View
  PROBe RTI 0.7,-1.3,View
  ASSIgn hi_$list_radix Hex
  ASSIgn hi_$monitor_radix Hex
  TRAce TRST*
  LISt -Change Hex TRST*
  MONitor Hex TRST*
  TRAce TCK
  LISt Hex TCK
  MONitor Hex TCK
  TRAce TMS
  LISt Hex TMS
  MONitor Hex TMS
  TRAce TDI
  LISt -Change Hex TDI
  MONitor Hex TDI
  TRAce Pin_In
  LISt -Change Hex Pin_In
  MONitor Hex Pin_In
  TRAce State
  LISt -Change Hex State
  MONitor Hex State
  TRAce Inst
  LISt -Change Hex Inst
  MONitor Hex Inst
  TRAce Data
  LISt -Change Hex Data
  MONitor Hex Data
  TRAce TDO
  LISt -Change Hex TDO
  MONitor Hex TDO
  TRAce Pin_Out
  LISt -Change Hex Pin_Out
  MONitor Hex Pin_Out
  TRAce On_Chip
  LISt -Change Hex On_Chip
  MONitor Hex On_Chip
  TRAce Reset_CH
  LISt -Change Hex Reset_CH
  MONitor Hex Reset_CH
```

```
TRAce Run_S_T
LISt -Change Hex Run_S_T
MONitor Hex Run_S_T
TRAce MODE
LISt -Change Hex MODE
MONitor Hex MODE
TRAce RTI
LISt -Change Hex RTI
MONitor Hex RTI
SCAle USer Time 1
SCAle TRace Time 50
INItialize XR
PERiod List 0
PERiod Trace 0
CLOck Period 14
FORCe clk_sys 0 0 -Repeat
FORCe clk_sys 1 5 -Repeat
CLOck Period 10
FORCe tck 0 0 -Repeat
FORCe tck 1 5 -Repeat
FORCe trst* 0
FORCe pin_in 0
RUN 6
FORCe trst* 1
RUN 4
```

```
# Functionnal test of the jtag_1 architecture (6 instructions)
# the path is /users/research/vhdlresearch/jtag_mentor/jtag_1/chip1
# this simulation file calls funct.do

    DO chip1/begin.do
    FORCe pin_in b
    DO start.do
    FORCe pin_in 3
    DO ldi_nop.do
    DO ld_data.do $fedc
    DO ld_data.do $4b01
    FORCe pin_in 9
    DO ld_data.do $ae
    FORCe pin_in 8
    DO ldi_sample.do
    FORCe pin_in f
    DO ldi_sample.do
    FORCe pin_in 0
    DO ldi_sample.do
    FORCe pin_in f
    DO ldi_sample.do
    FORCe pin_in 9
    DO ld_data.do $fedc
    FORCe pin_in 7
    DO ld_data.do $00dc
    FORCe pin_in 9
    DO ld_data.do $00dc
    FORCe pin_in 0
    DO ld_data.do $af3f
    DO ldi_extest.do
    FORCe pin_in f
    DO ld_data.do $01234b
    DO ldi_extest.do
    FORCe pin_in 5
    DO ld_data.do $0
    FORCe pin_in a
    DO ld_data.do $7
    DO ld_data.do $f
    FORCe pin_in 0
    DO ld_data.do $2
    FORCe pin_in 6
    DO ld_data.do $21
    FORCe pin_in e
    DO ldi_bypass.do
    FORCe pin_in 0
    DO ld_data.do %11110111101
    DO ld_data.do %110001011010101001101
    force pin_in b
    DO intest.do 4
    force pin_in f
    do runbist.do
    do ldi_nop.do
    force pin_in 4
    do runbist.do 5
```

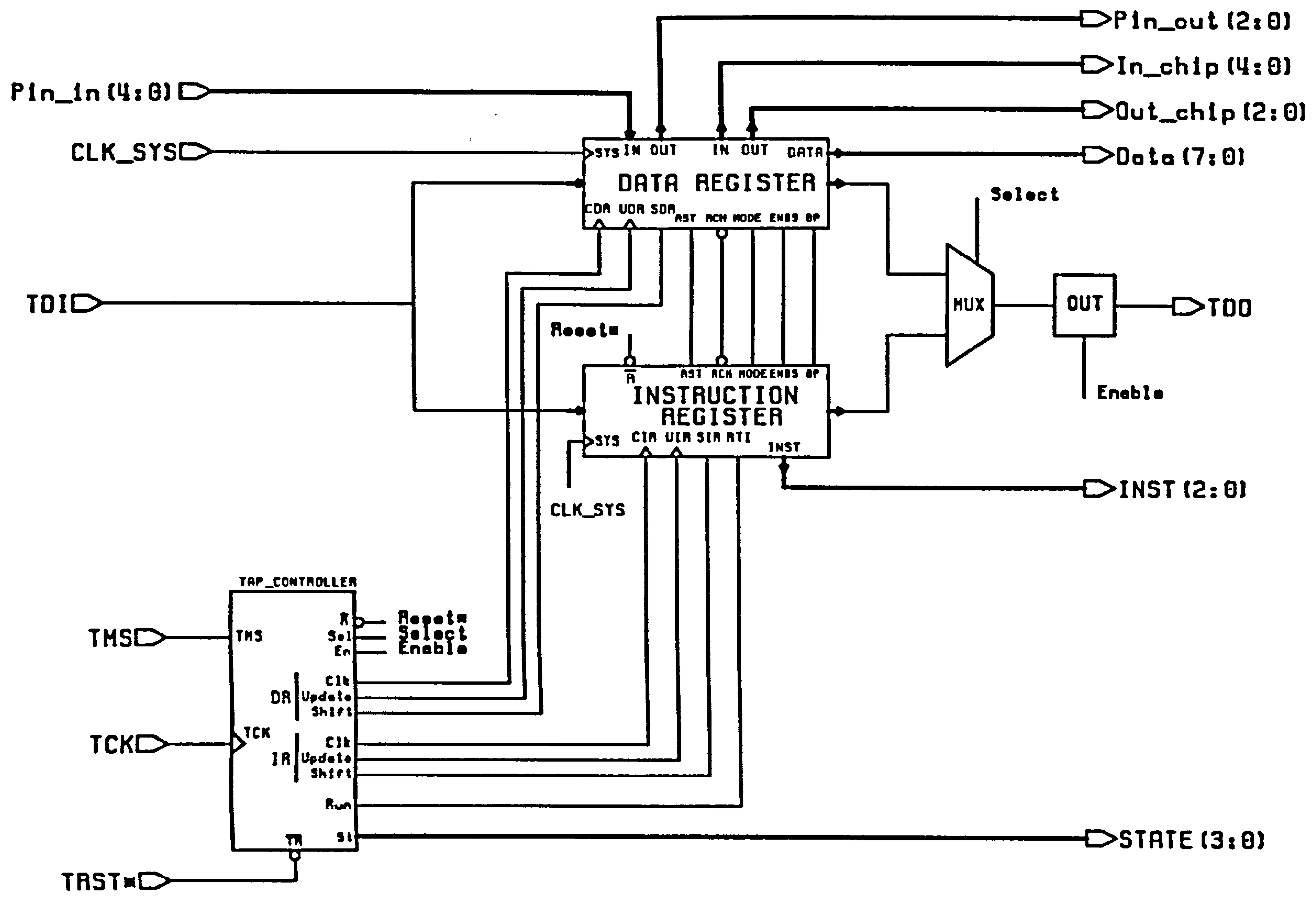
```
force pin_in 3
do ld_data.do $390a
force pin_in d
do intest.do 3
force pin_in 8
do ld_data.do $f1
```

FULL SIMULATION OF JTAG WITHOUT THE APPLICATION LOGIC

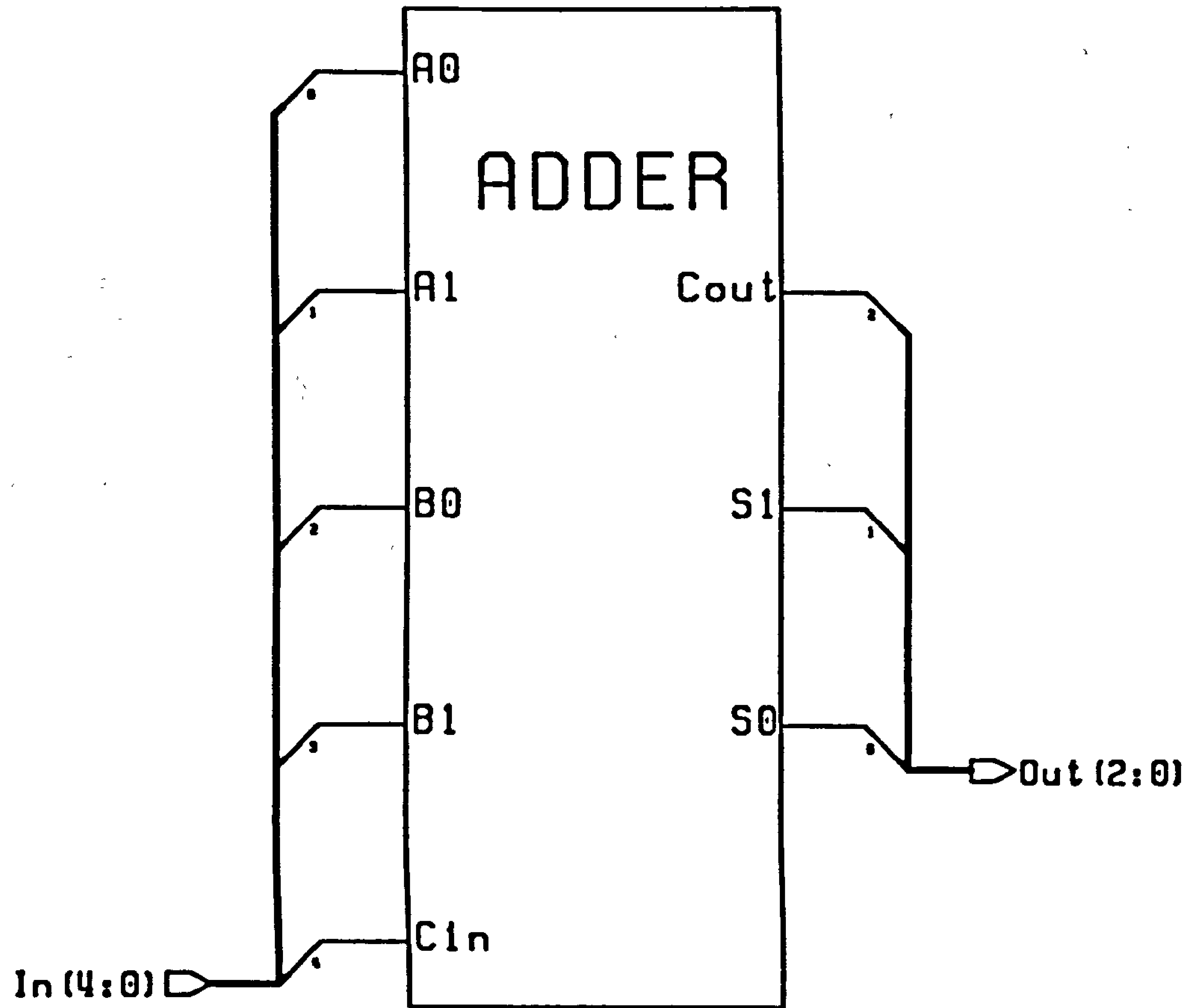
/JTAG_Mentor/JTAG0

## APPENDIX 5B

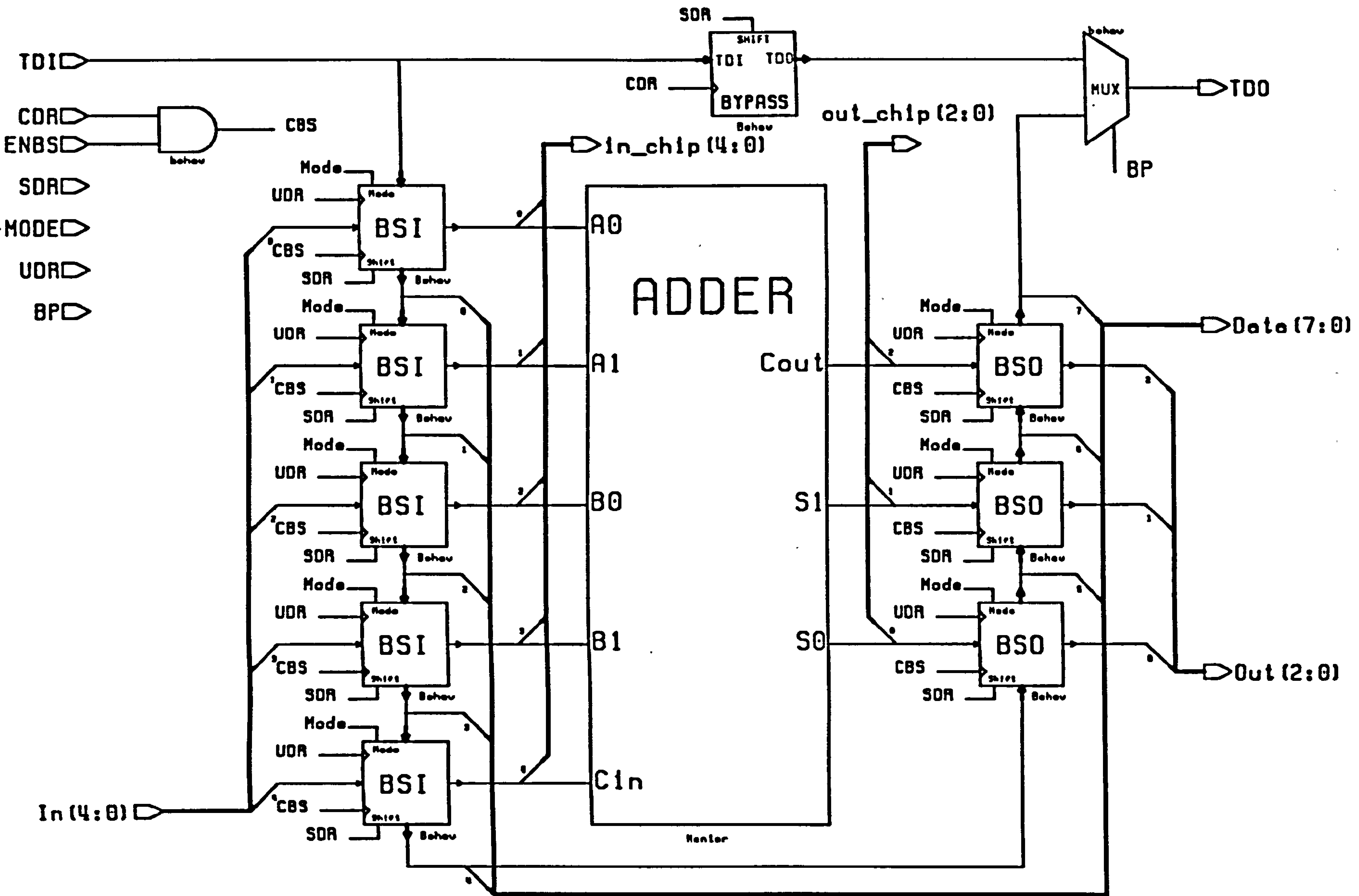
This appendix includes the design and simulation of a 2 bit Adder with JTAG included. Full logic simulation and fault simulation have been carried out. 6 test instructions have been generated using Macro-Functions available on the Mentor CAE system to allow ease of simulation.



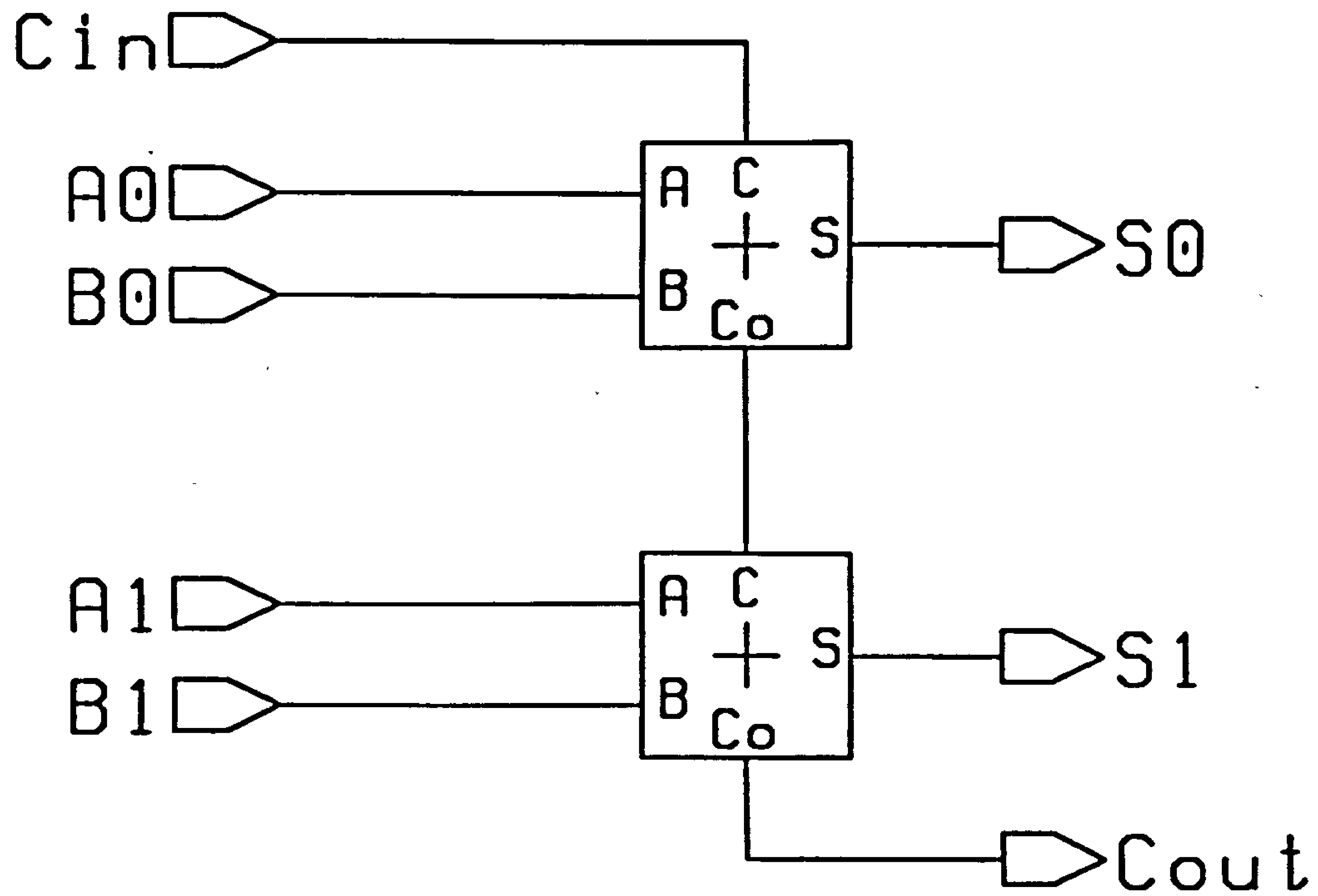

The JTAG boundary-scan architecture
for the test of an adder 2 bits



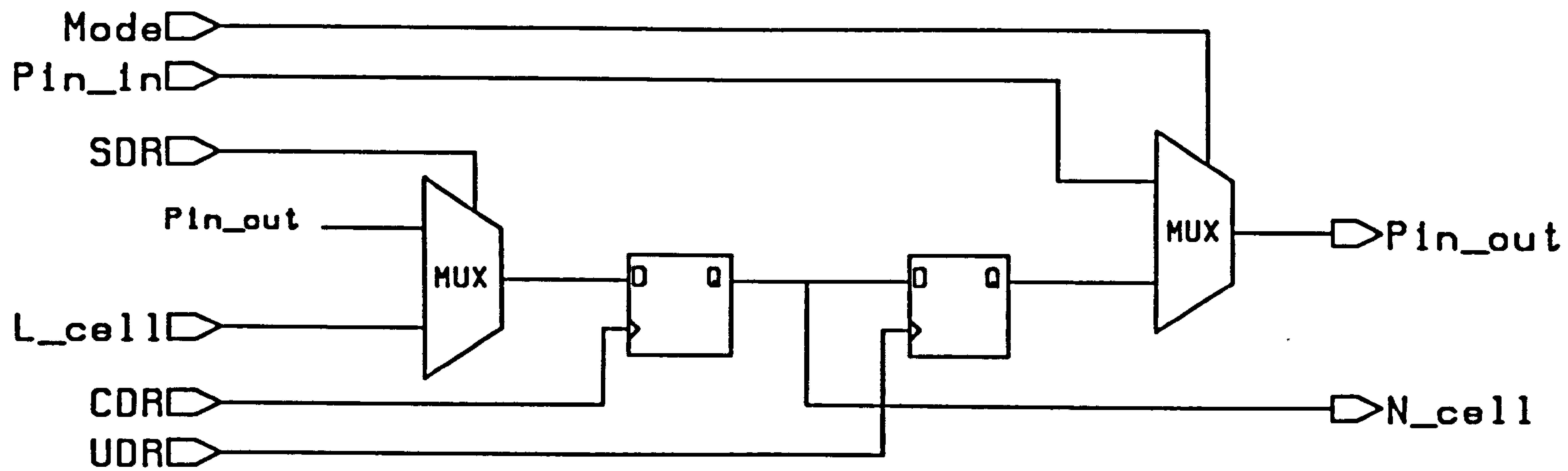

This schematic was designed to be easily tested using QUICKFAULT



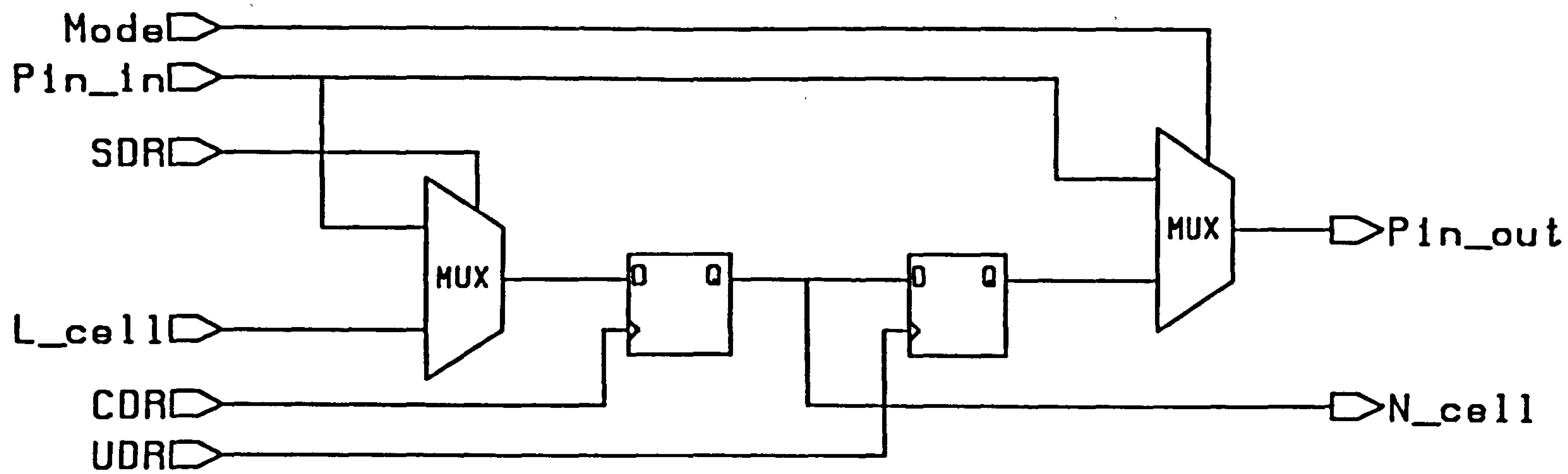

Data Register 2 states (5 in, 3 out)
Test of a 2 bits adder



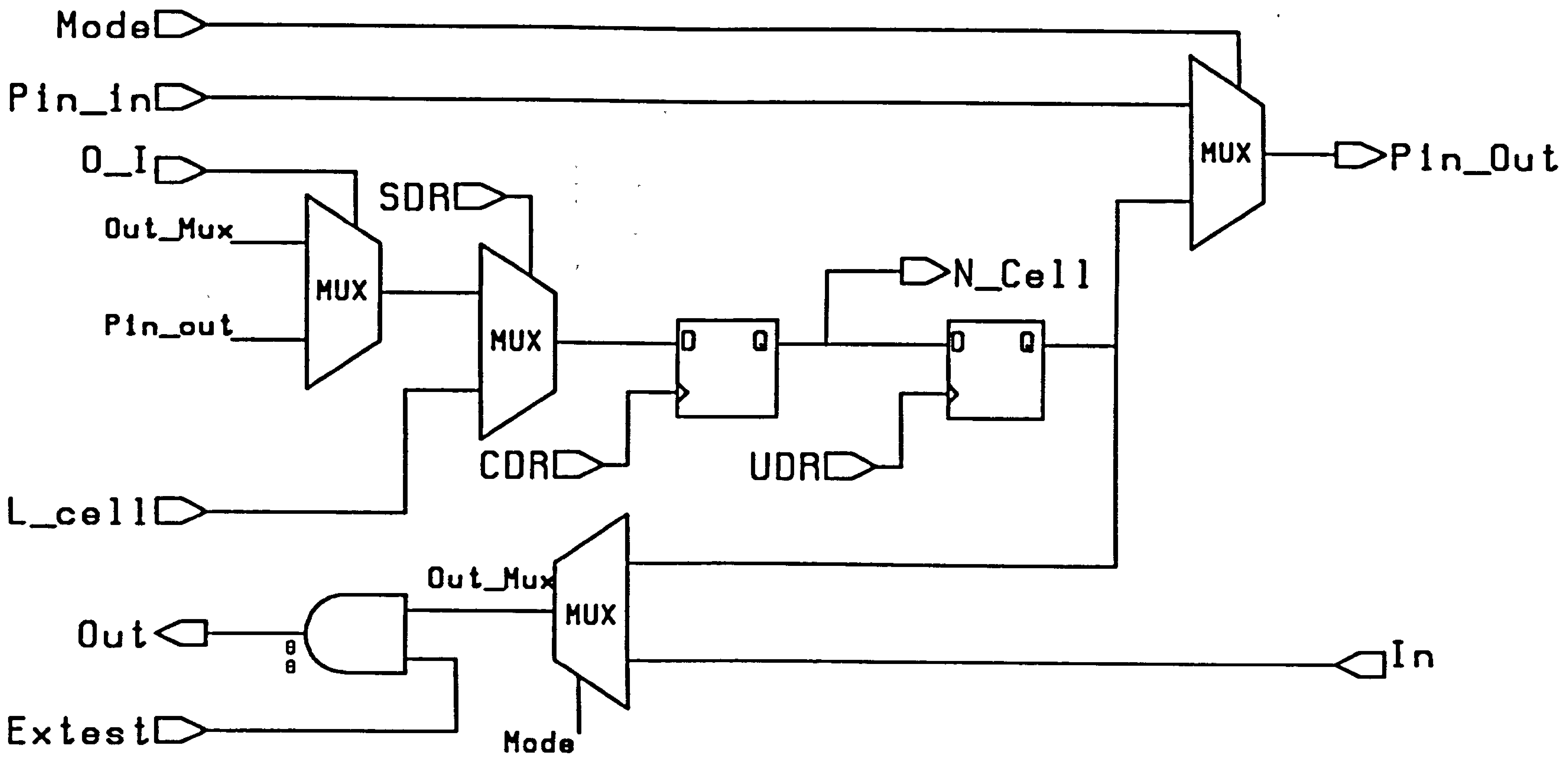

Adder 2 bits



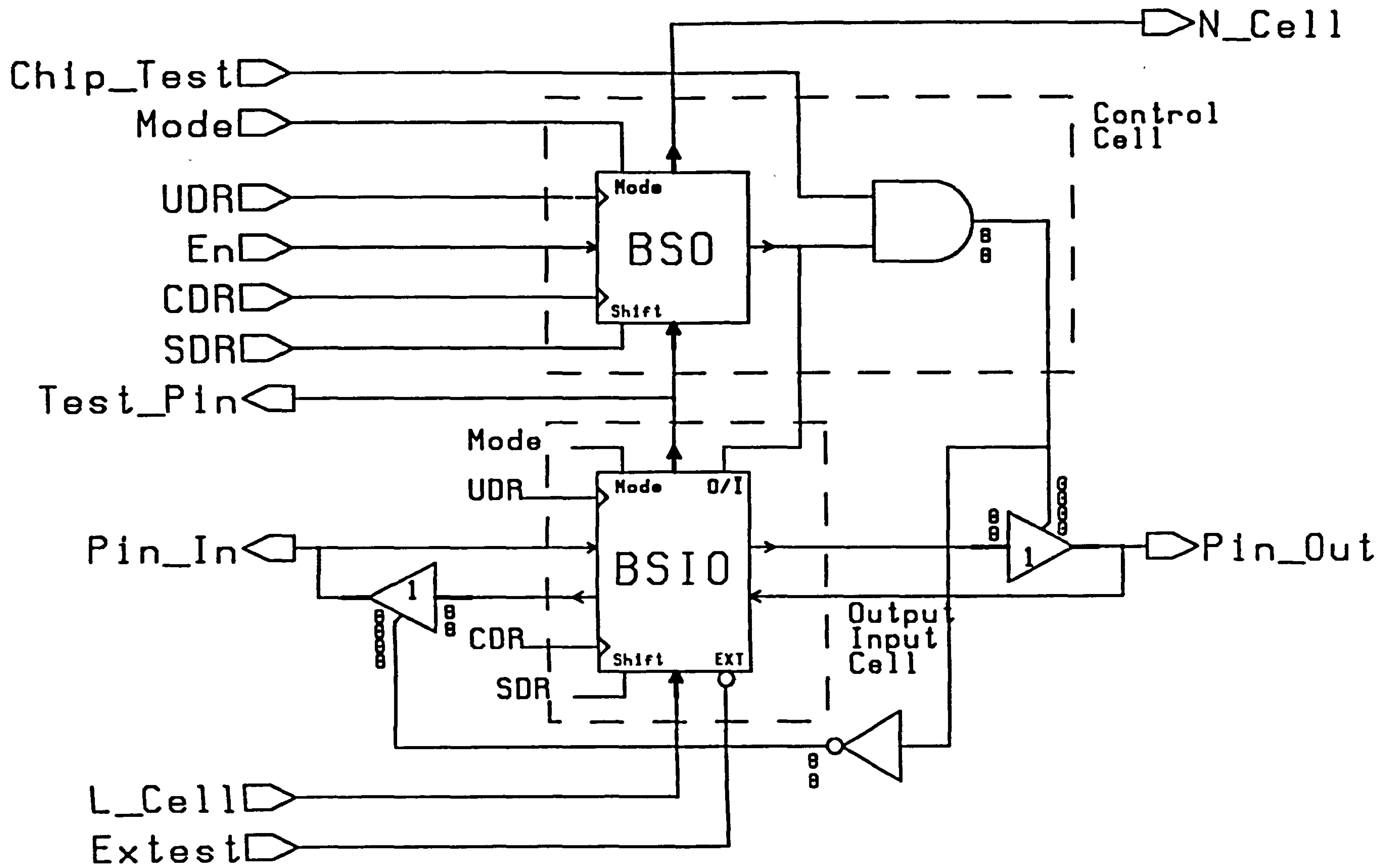

Boundary-scan cell for an input pin



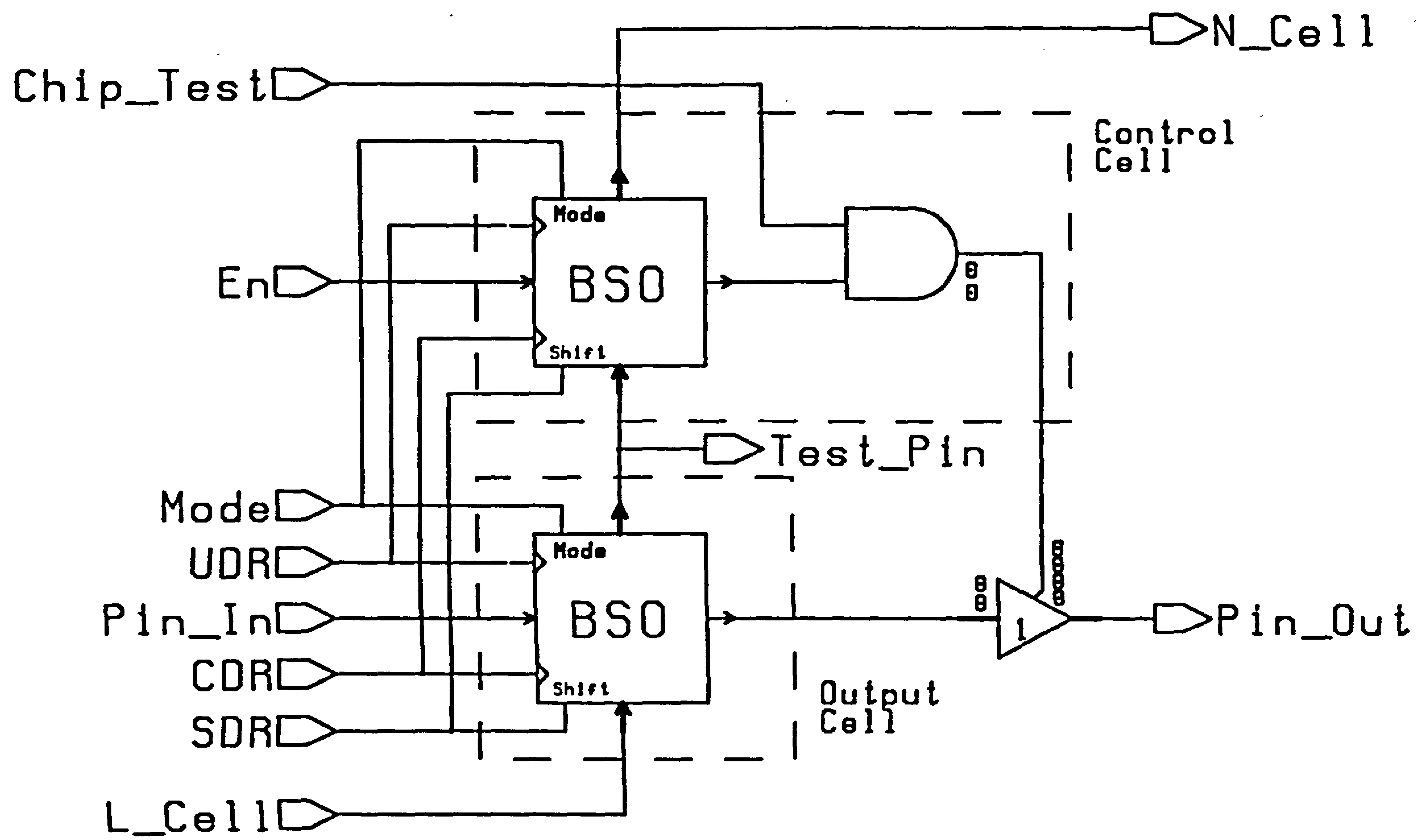

Boundary-scan cell for an output pin




Basic boundary-scan cell for an input/output pin

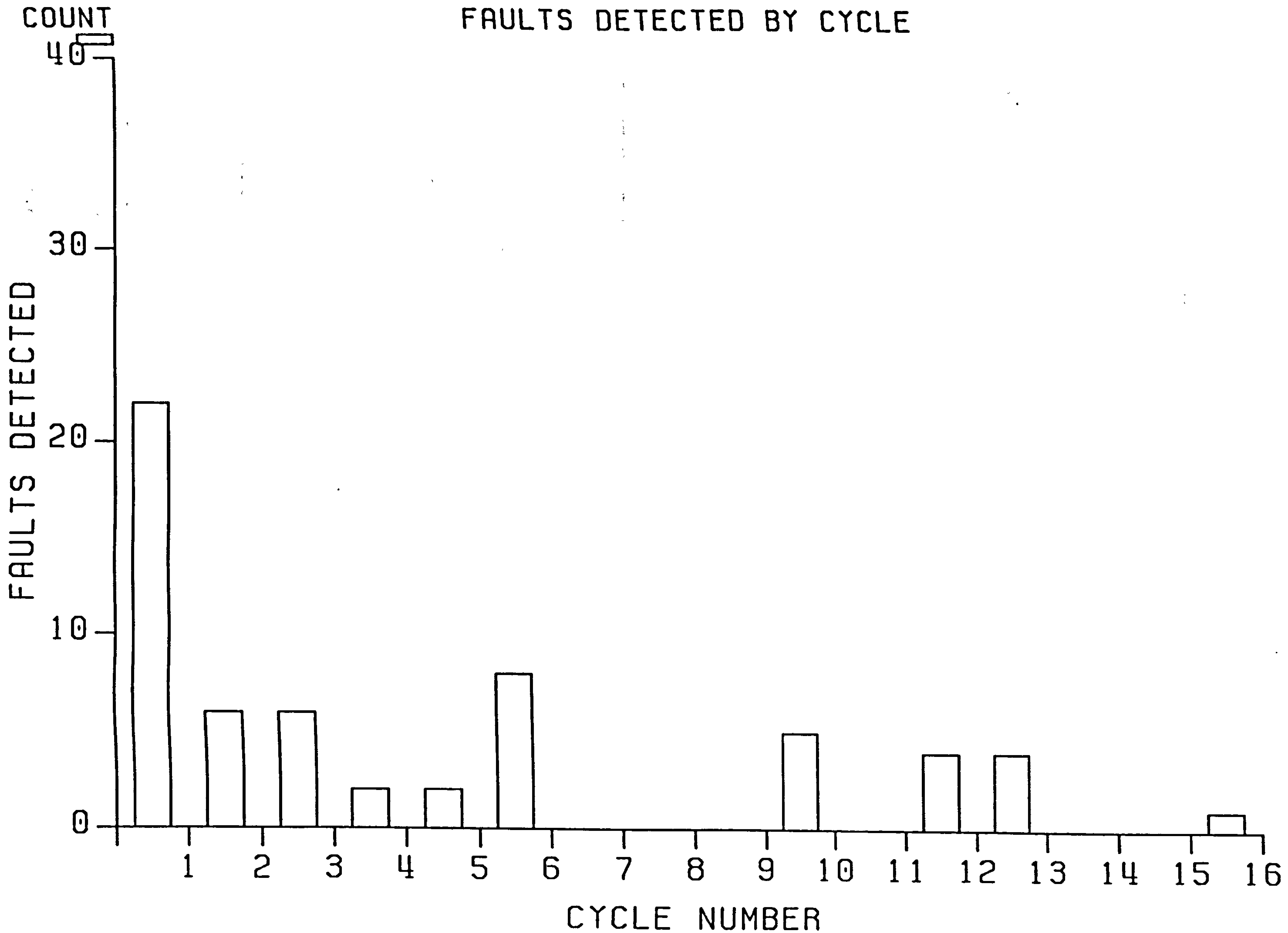


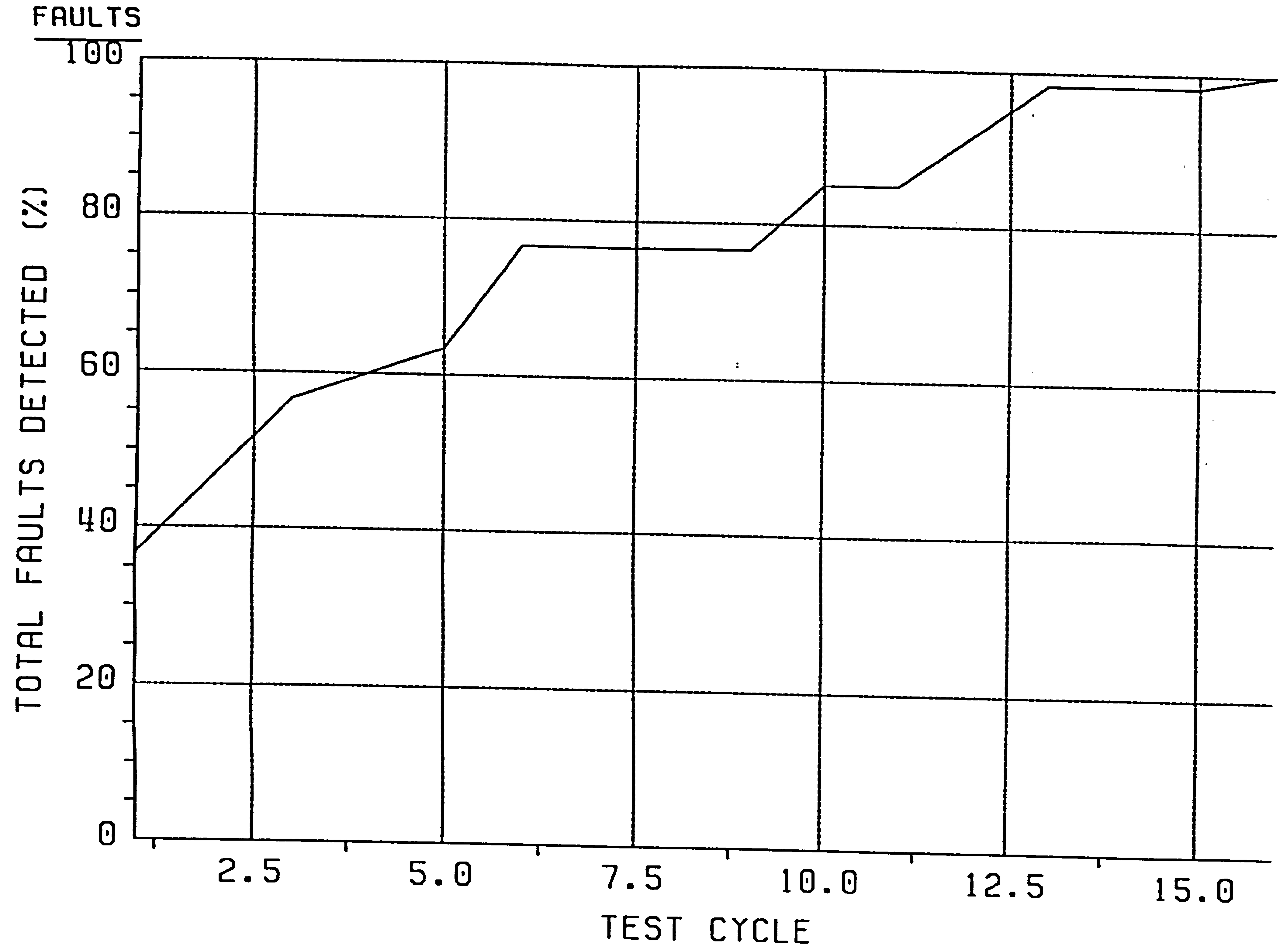

Basic boundary-scan cell for a bidirectional pin



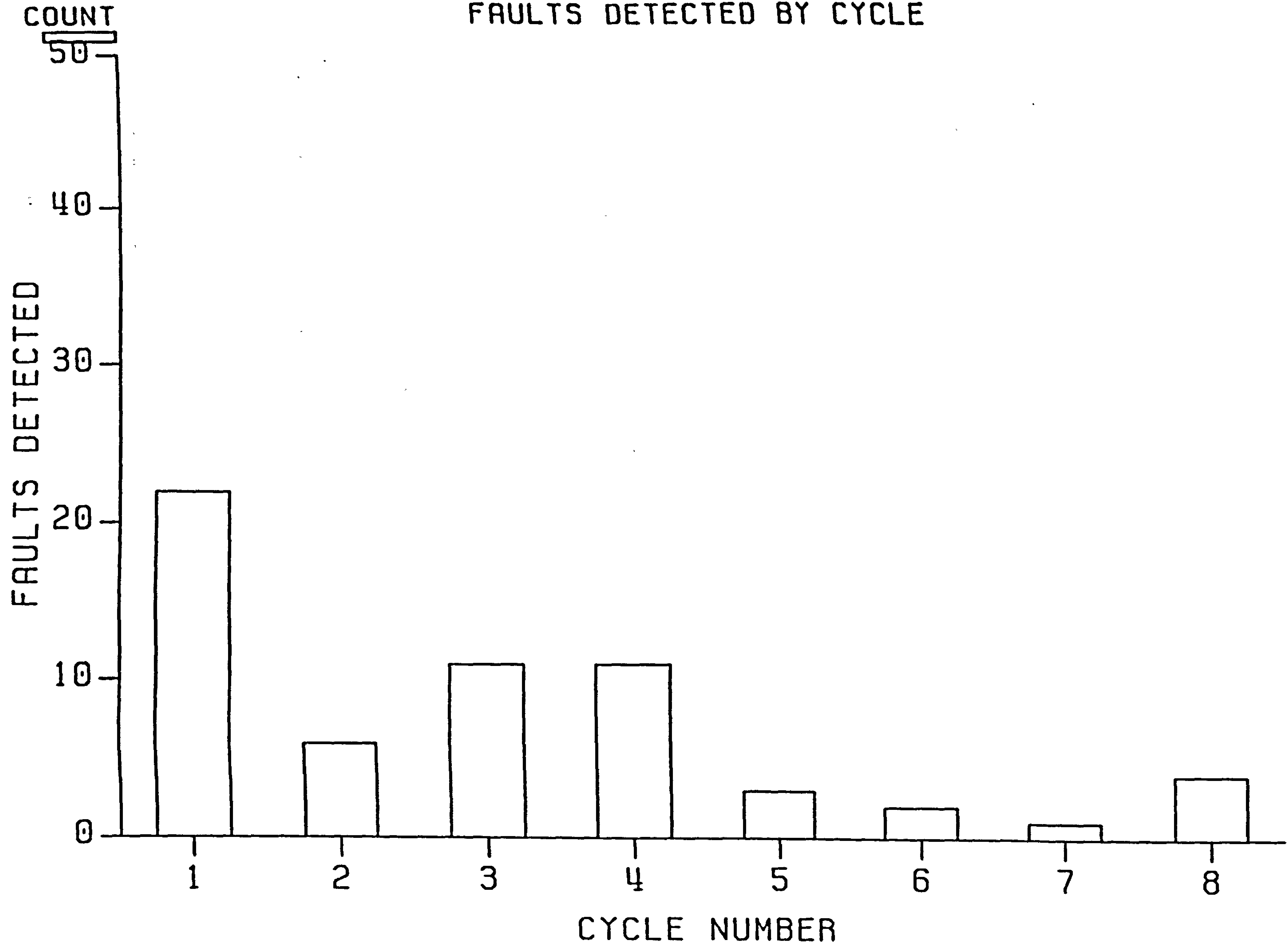

Basic boundary-scan cell for a 3-state output pin

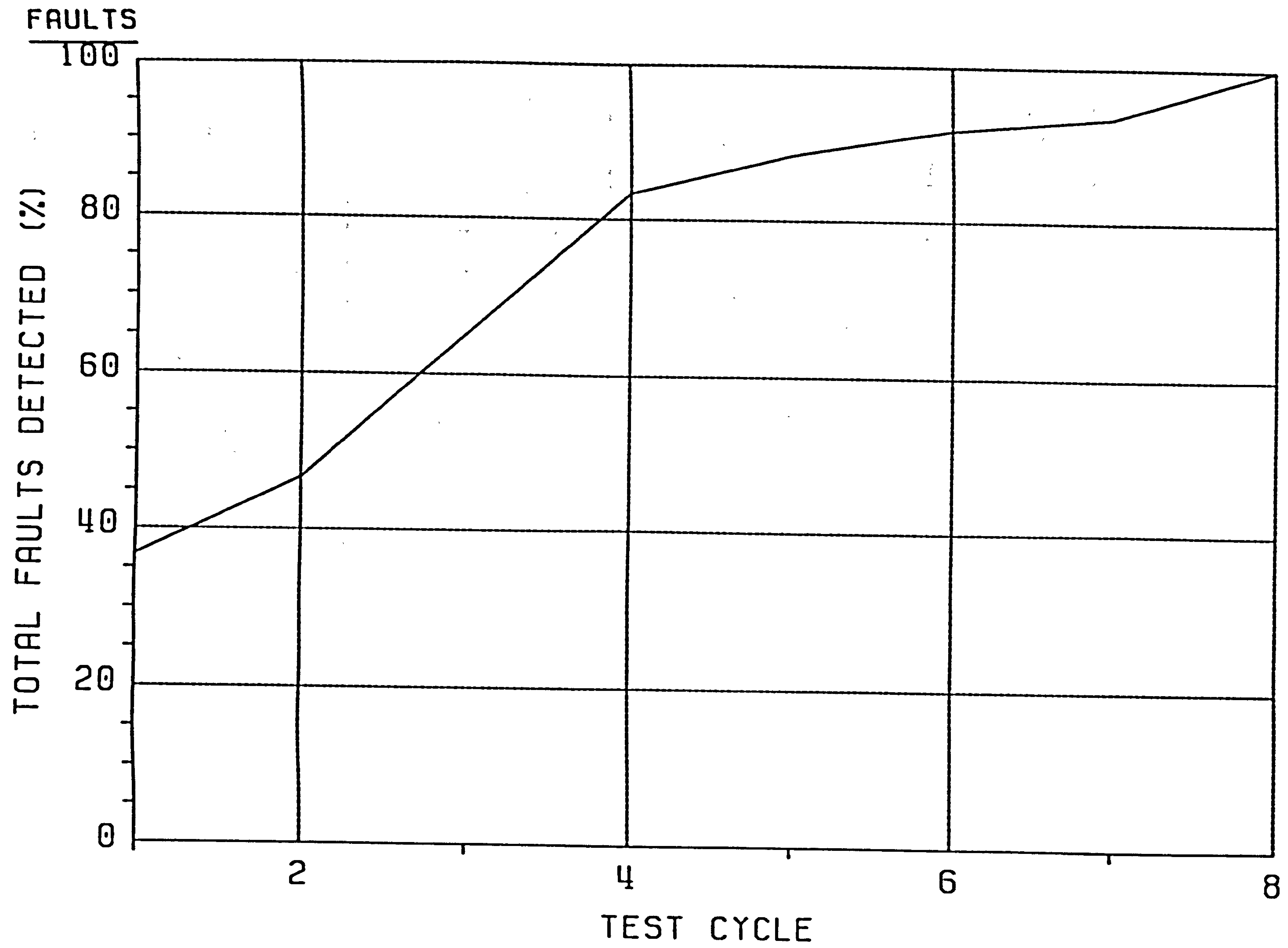
```
# Test faults simulation file for ADDER_2BITS

# PATH jtag_mentor/jtag_1/chip_to_check/test_adder/qfault_old.do
# This 1st version contains 35 test-patterns. With QFAULT, we
# expect diminish the number of test_patterns.

   CYCLe 10 5
   FORCe in  0
   FORCe in  1  10
   FORCe in  2  20
   FORCe in  4  30
   FORCe in  8  40
   FORCe in 11  50
   FORCe in  3  60
   FORCe in  6  70
   FORCe in  C  80
   FORCe in 19  90
   FORCe in 13 100
   FORCe in  7 110
   FORCe in  F 120
   FORCe in 1E 130
   FORCe in 1C 140
   FORCe in 18 150
   FORCe in 10 160
   FORCe in  1 170
   FORCe in  2 180
   FORCe in  5 190
   FORCe in  9 200
   FORCe in 15 210
   FORCe in  B 220
   FORCe in 16 230
   FORCe in  D 240
   FORCe in 1A 250
   FORCe in 14 260
   FORCe in 1D 270
   FORCe in 1B 280
   FORCe in 17 290
   FORCe in  F 300
   FORCe in 1F 310
   FORCe in 12 320
   FORCe in  E 330
   FORCe in  9 340
   RUN 350
```


FAULTS DETECTED BY CYCLE
COUNT
40
30
20
10
0
FAULTS DETECTED
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
CYCLE NUMBER


FAULTS
100
80
60
40
20
0
TOTAL FAULTS DETECTED (%)
2.5
5.0
7.5
10.0
12.5
15.0
TEST CYCLE

```
# Test faults simulation file for ADDER_2BITS

# PATH jtag_mentor/jtag_1/chip_to_check/test_adder/qfault.do
# This is the final list of the  8  patterns to send
# in order to obtain 100%

  CYCLe 10 5                 # Clock = 10 ns
  FAUlt DICTionary  -r       # Create  FAULT.DICT (faults dictonary)
  FORCe in  0                # pattern 1
  FORCe in  2   10           # pattern 2
  FORCe in 11   20           # pattern 3
  FORCe in  F   30           # pattern 4
  FORCe in 18   40           # pattern 5
  FORCe in  4   50           # pattern 6
  FORCe in  1   60           # pattern 7
  FORCe in  7   70           # pattern 8
  RUN 80
```


FAULTS DETECTED BY CYCLE
COUNT
50
40
30
20
10
0
FAULTS DETECTED
1
2
3
4
5
6
7
8
CYCLE NUMBER


FAULTS
100
TOTAL FAULTS DETECTED (%)
100
80
60
40
20
0
2
4
6
8
TEST CYCLE

```
==========================================================================

SENSOR PINS
     1 : OUT(0)        (left-most bit)
     2 : OUT(1)
     3 : OUT(2)        (right-most bit)


--------------------------------------------------------------------------
STEP 1
    GOOD VECTOR  : 000
    FAULT VECTOR : --1
                   { /I$1/I$2/I$3/OUT/1, /I$1/I$2/I$3/I0/1,
                   /I$1/I$2/I$2/I$2/OUT/1, /I$1/I$2/I$3/I1/1, /I$1/I$2/I$1
    FAULT VECTOR : -1-
                   { /I$1/I$1/I$3/OUT/1, /I$1/I$1/I$3/I0/1,
                   /I$1/I$1/I$2/I$2/OUT/1, /I$1/I$1/I$3/I1/1, /I$1/I$1/I$1
                   { /I$1/I$2/I$2/I$1/I0/1 } { /I$1/I$2/I$2/I$1/I1/1 }
                   { /I$1/I$2/I$2/I$1/OUT/1 } { /I$1/I$2/I$1/I$1/I0/1 }
                   { /I$1/I$2/I$1/I$1/I1/1 } { /I$1/I$2/I$1/I$1/OUT/1 }
    FAULT VECTOR : 1--
                   { /I$1/I$1/I$2/I$1/I0/1 } { /I$1/I$1/I$2/I$1/I1/1 }
                   { /I$1/I$1/I$2/I$1/OUT/1 } { /I$1/I$1/I$1/I$1/I0/1 }
                   { /I$1/I$1/I$1/I$1/I1/1 } { /I$1/I$1/I$1/I$1/OUT/1 }

--------------------------------------------------------------------------
STEP 2
    GOOD VECTOR  : 010
    FAULT VECTOR : --1
                   { /I$1/I$2/I$2/I$2/I0/1 } { /I$1/I$2/I$1/I$2/I1/1 }
    FAULT VECTOR : -0-
                   { /I$1/I$2/I$2/I$1/I1/0 } { /I$1/I$2/I$2/I$1/OUT/0 }
                   { /I$1/I$2/I$1/I$1/I0/0 } { /I$1/I$2/I$1/I$1/OUT/0 }

--------------------------------------------------------------------------
STEP 3
    GOOD VECTOR  : 010
    FAULT VECTOR : --1
                   { /I$1/I$2/I$2/I$2/I1/1 }
    FAULT VECTOR : -0-
                   { /I$1/I$1/I$3/I0/0, /I$1/I$1/I$2/I$2/OUT/0,
                   /I$1/I$1/I$2/I$2/I0/0, /I$1/I$1/I$2/I$2/I1/0 }
                   { /I$1/I$1/I$3/OUT/0 } { /I$1/I$2/I$2/I$1/I0/0 }
    FAULT VECTOR : 1--
                   { /I$1/I$1/I$2/I$1/I0/0 } { /I$1/I$1/I$2/I$1/I1/0 }
    FAULT VECTOR : 10-
                   { /I$1/I$1/I$1/I$1/I0/0 } { /I$1/I$1/I$1/I$1/OUT/0 }

--------------------------------------------------------------------------
STEP 4
    GOOD VECTOR  : 011
    FAULT VECTOR : --0
```

```
                { /I$1/I$2/I$3/I1/0, /I$1/I$2/I$1/I$2/OUT/0,
                /I$1/I$2/I$1/I$2/I0/0, /I$1/I$2/I$1/I$2/I1/0 }
                { /I$1/I$2/I$3/OUT/0 }
FAULT VECTOR : -0-
                { /I$1/I$1/I$3/I1/0, /I$1/I$1/I$1/I$2/OUT/0,
                /I$1/I$1/I$1/I$2/I0/0, /I$1/I$1/I$1/I$2/I1/0 }
                { /I$1/I$2/I$1/I$1/I1/0 }
FAULT VECTOR : 1--
                { /I$1/I$1/I$1/I$1/I1/0 }
```

```
------------------------------------------------------------------------
STEP 5
    GOOD VECTOR  : 110
    FAULT VECTOR : --1
                   { /I$1/I$2/I$1/I$2/I0/1 }
    FAULT VECTOR : -01
                   { /I$1/I$1/I$2/I$2/I1/1 }
    FAULT VECTOR : 0--
                   { /I$1/I$1/I$2/I$1/OUT/0 }

------------------------------------------------------------------------
STEP 6
    GOOD VECTOR  : 100
    FAULT VECTOR : -1-
                   { /I$1/I$1/I$2/I$2/I0/1 } { /I$1/I$1/I$1/I$2/I0/1 }

------------------------------------------------------------------------
STEP 7
    GOOD VECTOR  : 100
    FAULT VECTOR : -1-
                   { /I$1/I$1/I$1/I$2/I1/1 }

------------------------------------------------------------------------
STEP 8
    GOOD VECTOR  : 001
    FAULT VECTOR : --0
                   { /I$1/I$2/I$3/I0/0, /I$1/I$2/I$2/I$2/OUT/0,
                   /I$1/I$2/I$2/I$2/I0/0, /I$1/I$2/I$2/I$2/I1/0 }
```

```
# Simulation of real generation of test patterns on design 'Chip2'
# PATH jtag_mentor/jtag_1/chip2/funct.do
# 8 patterns optomised by QUICKFAULT are sent in and
# the results are collected at TDO

    DO chip2/begin.do
    DO start.do
    DO ld_data.do %00000  # $00
    DO ld_data.do %010    # $02
    DO ld_data.do %001    # $11
    DO ld_data.do %111    # $0F
    DO ld_data.do %000    # $18
    DO ld_data.do %100    # $04
    DO ld_data.do %001    # $01
    DO ld_data.do %00111  # $07
    DO ld_data.do %000    # Output of previous result
```

BYPASS INSTRUCTION

TEXT BOUND INTO

THE SPINE

INTEST INSTRUCTION
(OPTIONAL)

APRIL 1001

SAMPLE & PRELOAD INSTRUCTION

(MULTIPLYING 6 x A = 3C

EXTEST INSTRUCTION

JTAG_MENTOR/JTAG0/TAP_CONTROLLER/TRACE_SIM.PIC

SIMULATION RESULTS OF JTAP

APRIL 1991

/JTAG_MENTOR/JTAG_1/CHIP2/TRACE_SIM

FULL SIMULATION OF JTAG WITH THE APPLICATION LOGIC

**Appendix 6A**

The following software was written to describe the VHDL package of JTAG_STANDARD. It contains the high level models of the JTAG full architecture. It also demonstrates the power of using facilities such as 'PACKAGE' or 'LIBRARY' in designing a system.

```
use work.declar.all;
package JTAG_STANDARD is

component TAP_CONTROLLER
        generic (Setup_Time, Hold_time,Min_Pulse_Width_1,
        Min_Pulse_Width_0, S_Odel, Odel: Time);
        PORT (
        TMS,
        TCK: in bit;
        Reset,
        Selectt,
        Enable,
        ShiftIR,
        ClockIR,
        UpdateIR,
        ShiftDR,
        UpdateDR,
        ClockDR : out bit
        );
end component;

component REG_INSTRUCTION
        generic
        (Setup_Shift_Time,Hold_Shift_Time,MPulse_Width_Shift,

        Setup_Update_Time,Hold_Update_Time,MPulse_Width_Update,
                Mux_Del,Stage_Shift_Del,Stage_Update_Del:time);

        port(
        Reset,
        ClockIR,
        UpdateIR,
        ShiftIR,
        TDI:in bit;
        Status:in bit_vector;
        Instruction: out bit_vector;
        TDO: out bit);
end component;


component INSTRUCTION_DECODER
        generic (Instruction_set:bit_vector;

        DR_select_Set,Test_Mode_Set,Additional_Signals_Set:
                bit_vector;
                Open_Check:Boolean;
                DEC_DEL:Time);
        port (
        Instruction:in bit_vector(0 to 7);
        DR_Select: out bit_vector(0 to 7);
        Test_Mode: out bit_vector(0 to 7);
        Additional_Signals : out Bit_vector
        );
end component;
```

```
component BYPASS_REG
        generic (Setup_Time,
                Hold_Time,
                MPulse_width,
                AND_Del,
                Bypass_Del :Time);
        Port(
        ShiftDR, ClockDR,
        TDI: in bit;
        TDO: out bit);

component end;


component REG_BSCAN
        generic (System_pin_type_types: String_vector;
                SUT_Capture,HT_Capture, MDEL,SUT_Update,HT_Update,
        port(
        Select_Bscan,
        Reset,
        ShiftDR,
        ClockDR,
        UdateDR,
        TDI: in Bit;
        Parallel_Input: in Bit_vector;
        Test_Mode:in Bit_vector(0 to 1);
        Parallel_Output: out Bit_vector;
        TDO: out Bit
        );

component end;


entity IDENT_REG
        generic (Setup_time,Hold_Time,MPulse_Width,Mux_Del:Time);
        port(
        Select_Ident,
        ShiftDR,
        ClockDR,
        TDI:in Bit;
        ID_Code:in Bit_vector;
        TDO: out Bit
        );
end component;


component MUX_1
        generic (Instruction_Set :Bit_vector;
                TDO_Test_Data_Registers_Set : integer_vector;
                Mux_Del :Time);
         port (
         TDO_Test_Data_Registers: in Bit_vector;
         Instruction : in Bit_vector;
         TDO: out Bit);
end component;
```

```
component mux_2
        generic (Mux_DEL : Time);
        port (
        Tdo_Test_Data_Registers,
        Tdo_Instruction,
        Selectt : in Bit;
        TDO : out Bit
        );
end component;


entity TDO_Buffer is
        generic (setup_time, hold_time, Min_pulse_width, Del,
                Tdo_del:Time);
        port(
        TCK,
        Enablee,
        Input:in Bit;
        TDO: out Tristate
        );
end component;

end JTAG_STANDARD;
```

**Appendix 6B**

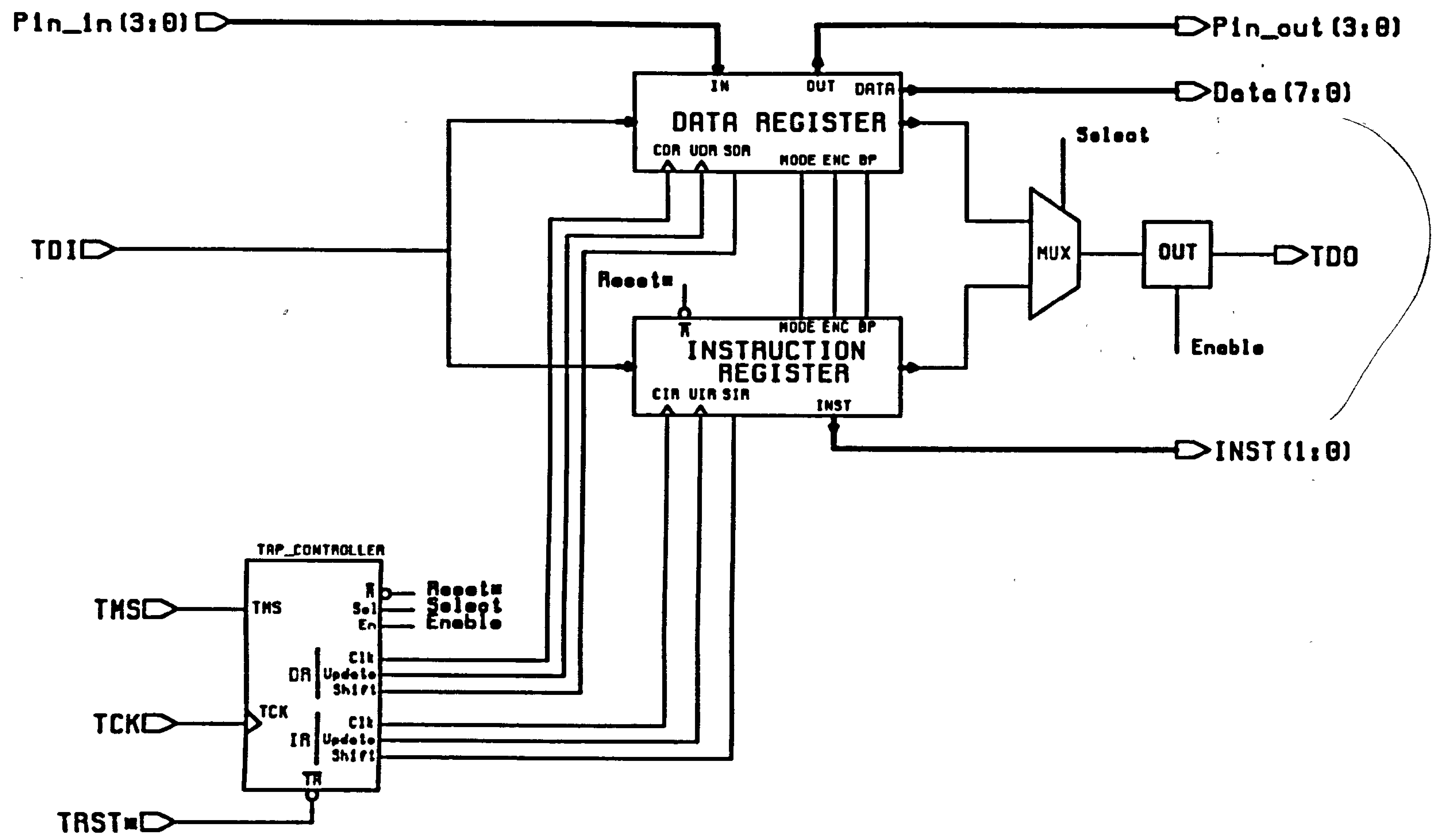
The following software was written to describe the VHDL package of Declaration. It contains all the timing elements used in JTAG.

```
library std,work;
use std.standard.all;
PACKAGE Declaration IS

        TYPE integer_vector is array (integer  range <>) of
integer;

        TYPE tristate IS
                (
                'Z',   -- high impedance
                '0',   -- logic zero
                '1'    -- logic one
                );
        TYPE state_tap IS
                (
                Test_Logic_Reset,
                Run_Test_Idle,
                Select_DR_Scan,
                Capture_DR,
                Shift_DR,
                Exit_1_DR,
                Pause_DR,
                Exit_2_DR,
                Update_DR,
                Select_IR_Scan,
                Capture_IR,
                Shift_IR,
                Exit_1_IR,
                Update_IR,
                Undefined
                );

        TYPE System_pins IS

            (   'K',               --clock,
                'I',               --Input,
                'B',               --Bidirectional_Input,
                'O',               --Output,
                'P',               --Bidirectional_Output
                'T',               --Tri_state,
                'C'                --Control
            )   ;

        TYPE String_vector is array (integer range <>) of
System_pins;

END declaration;
```

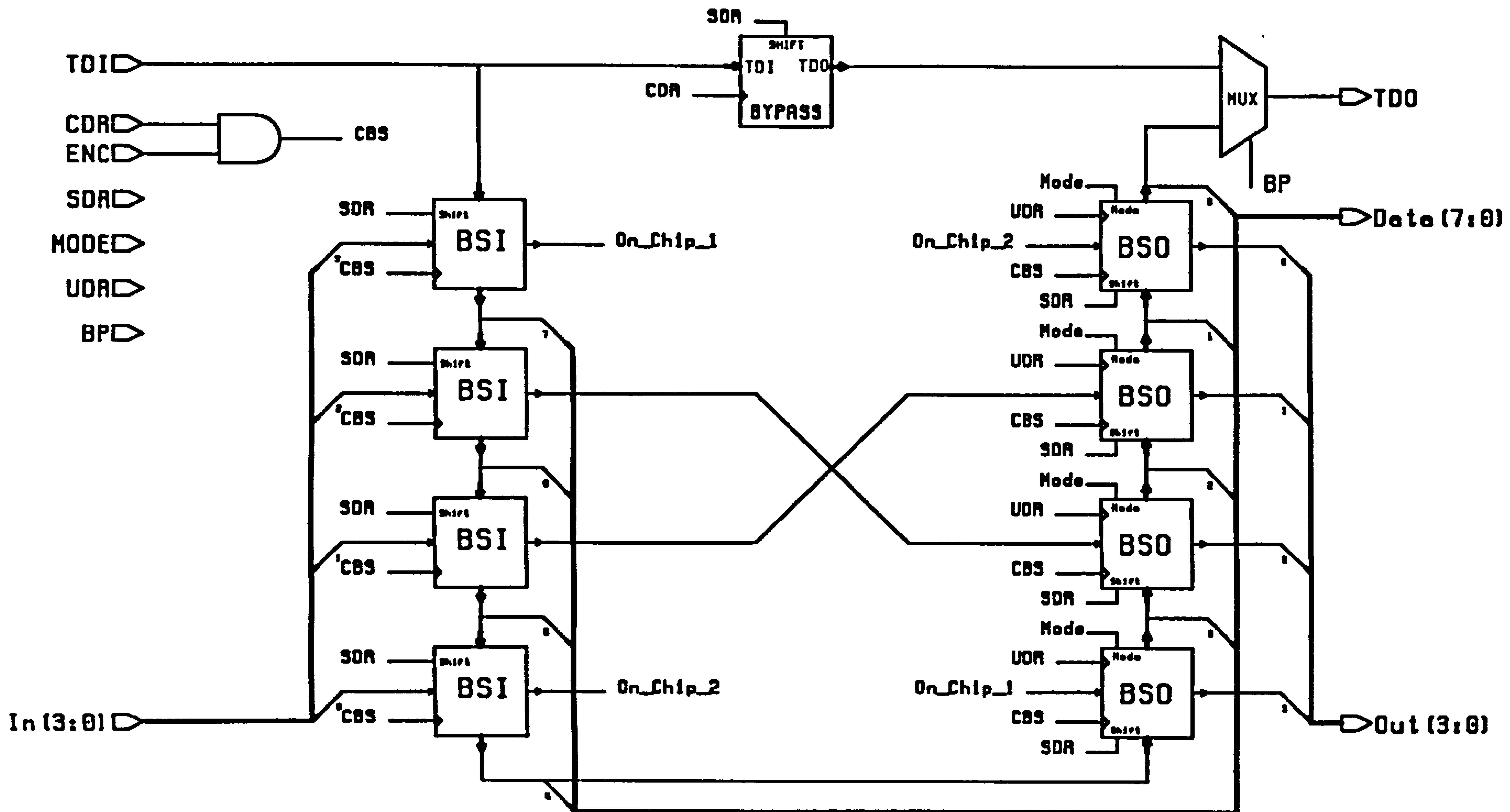
**Appendix 6C**

This appendix describes the connectivity of the VHDL high level model of the JTAG architecture and its simulation results.

PATH: /USER/RESEARCH/VHDLRESEARCH/JTAG_VHDL/JTAG_0/**CHIP1**




The JTAG boundary-scan architecture for the test of functionality




Data Register 2 states (4 in, 4 out)

SIMULATION OF JTAP USING VHDL BEHAVIOURAL DESCRIPTION

/JTAG_VHDL/JTAG_0/CHIP1/TRACE_SIM1&2

-->
a

TRST*
TCK
TMS
TDI
Pin_In
Inst
Data
TDO
Pin_Ou

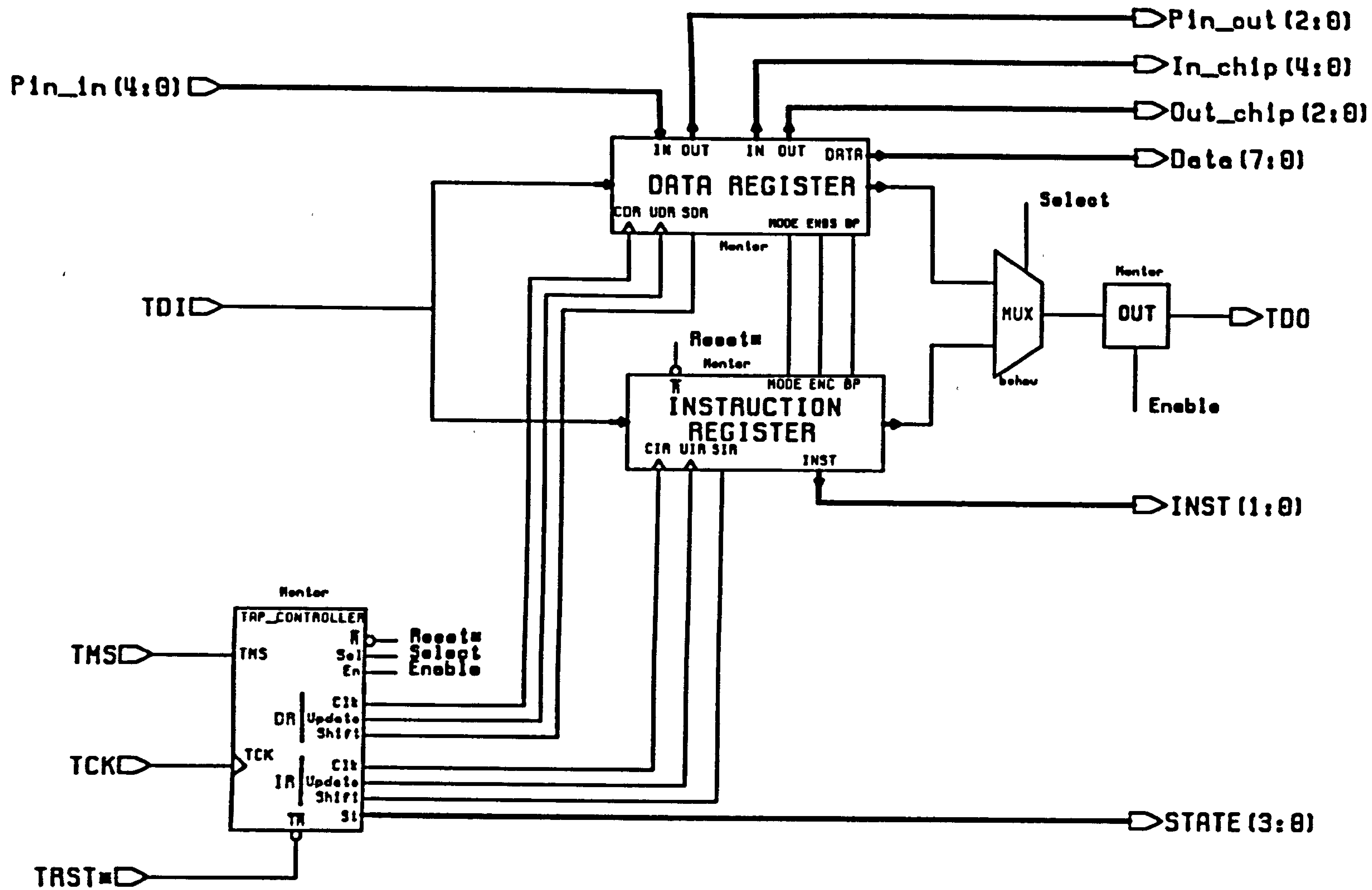
3500. 3750.0 4000.0 4250.0 4500.0 4750.0 5000.0 5250.0 5500.0 5750.0 6000.0 6250.0 6500.0 6750.0

**Appendix 6D**

This appendix describes the connectivity of the VHDL high level model of the JTAG architecture with an application logic, a 2 bit adder. It demonstrates the connectivity and the validity of its operation together with the simulation results.

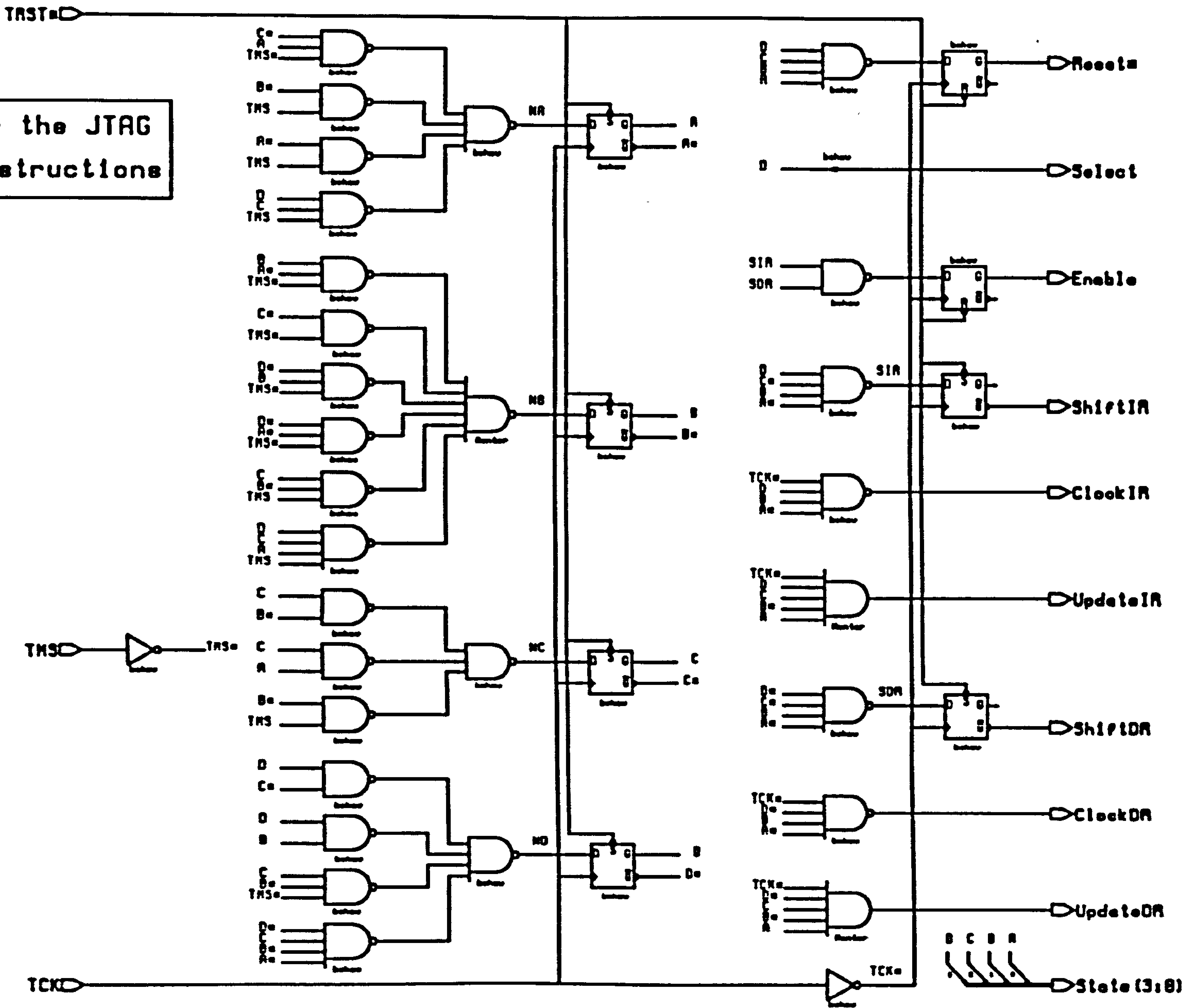
PATH: /USER/RESEARCH/VHDLRESEARCH/JTAG_VHDL/JTAG_0/**CHIP2**

/JTAG_VHDL/JTAG_0/CHIP2

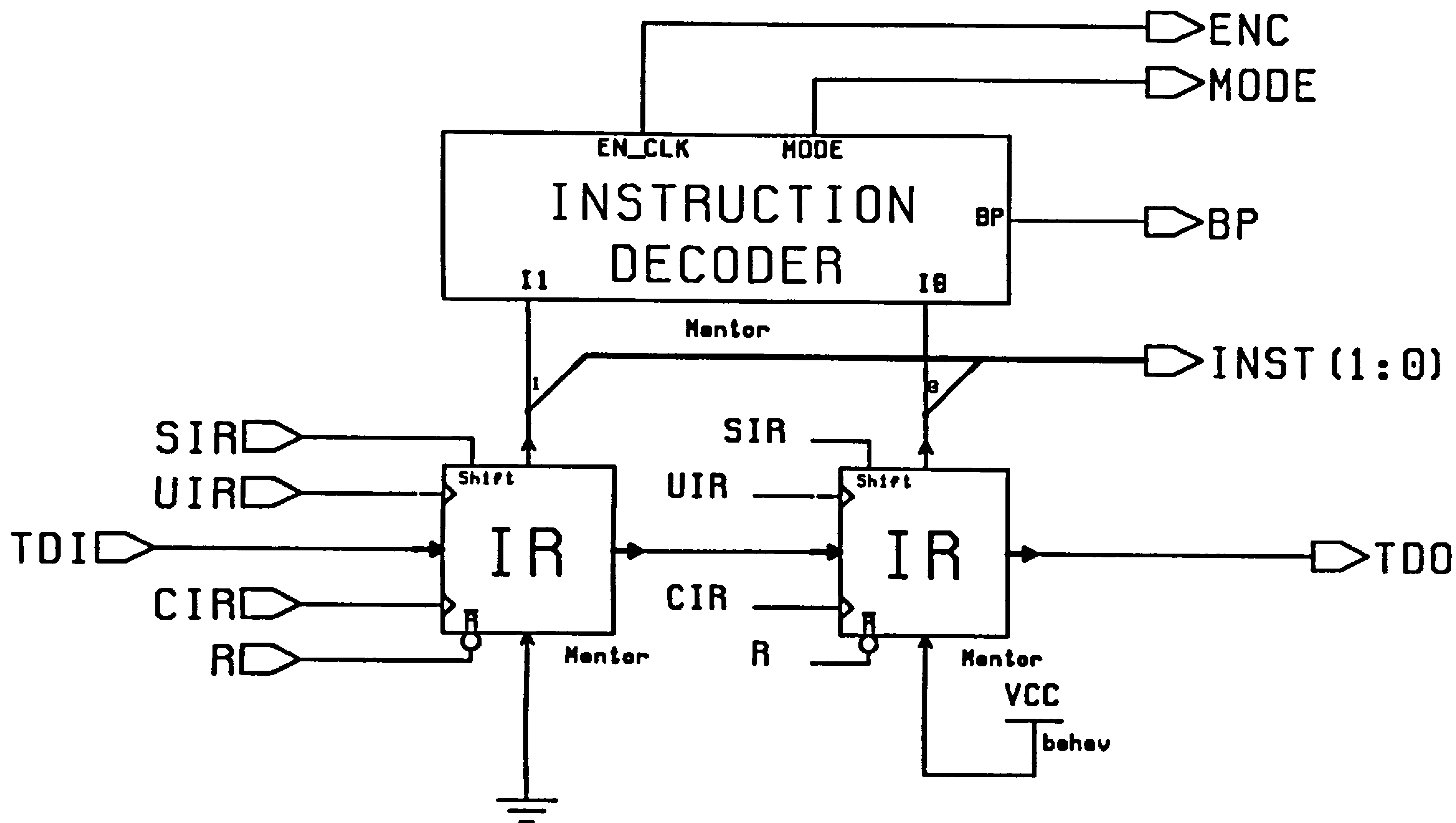



The JTAG boundary-scan architecture
for the test of an adder 2 bits
(version with 4 instructions)

/JTAG_VHDL/JTAG_0/TAP_CONTROLLER

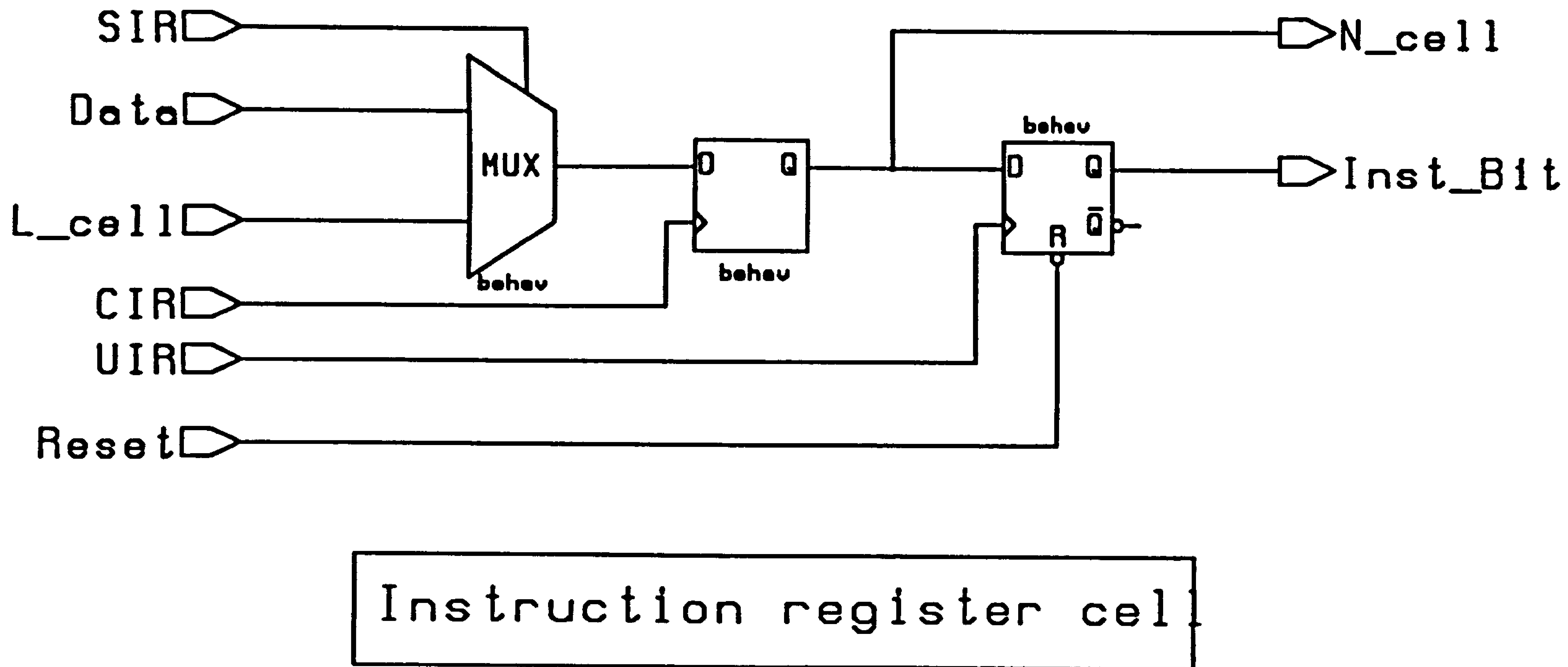


/JTAG_VHDL/JTAG_0/INST_REGISTER

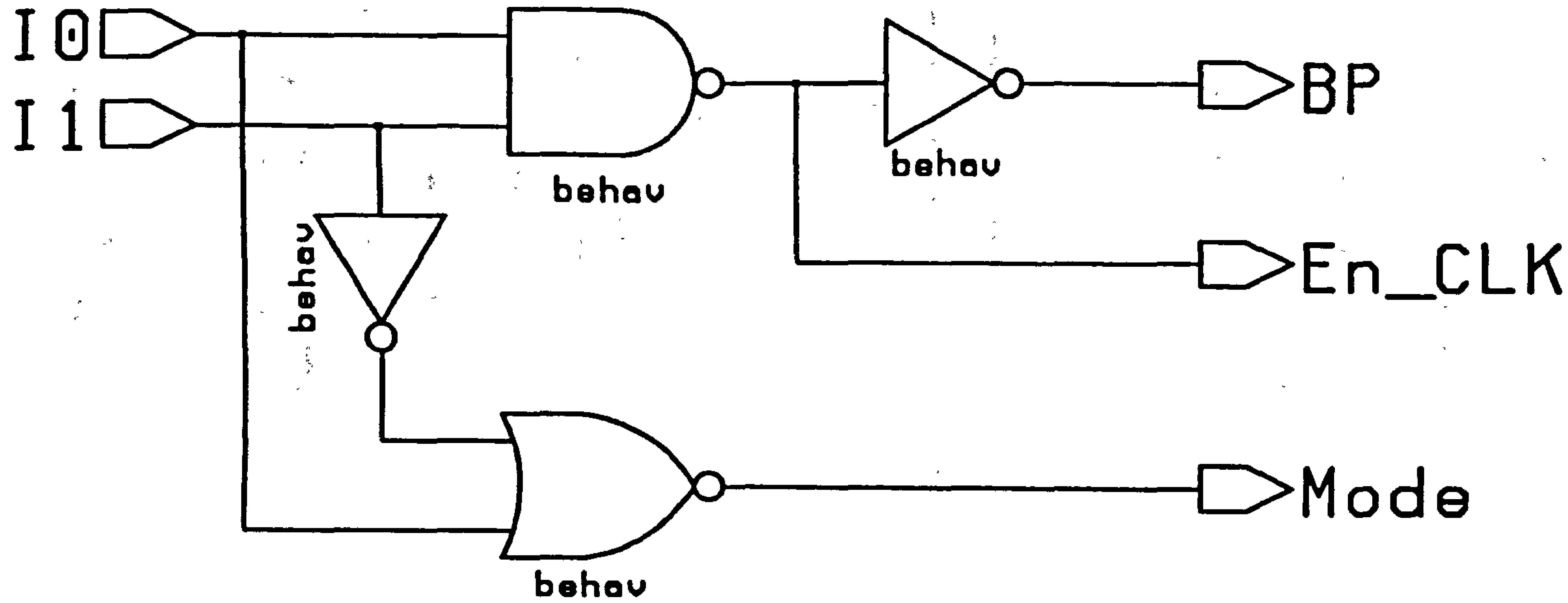



Instruction register for the version with 4 instructions

/JTAG_VHDL/JTAG_0/INST_REG_CELL

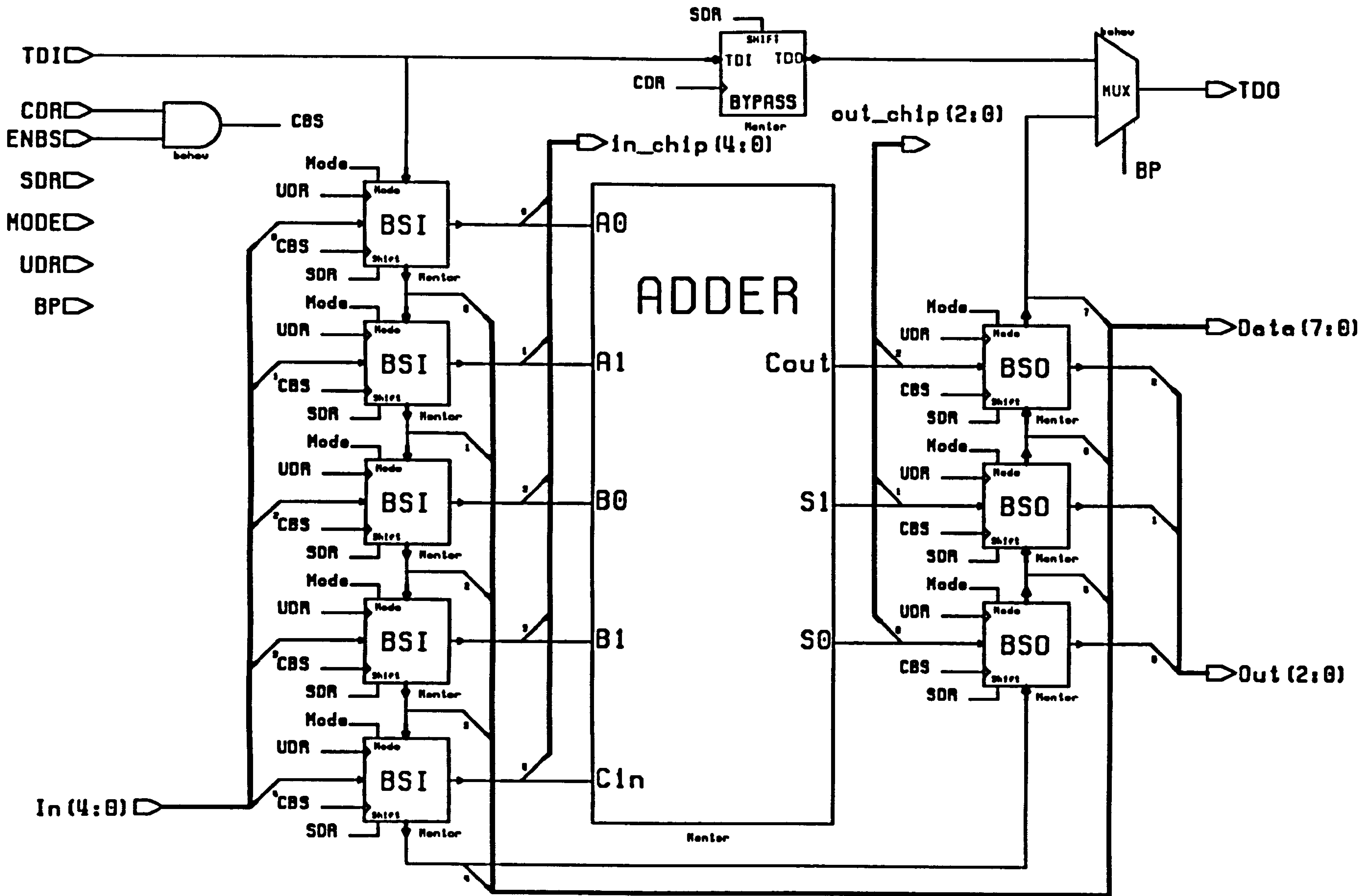


/JTAG_VHDL/JTAG_0/INST_DECODER



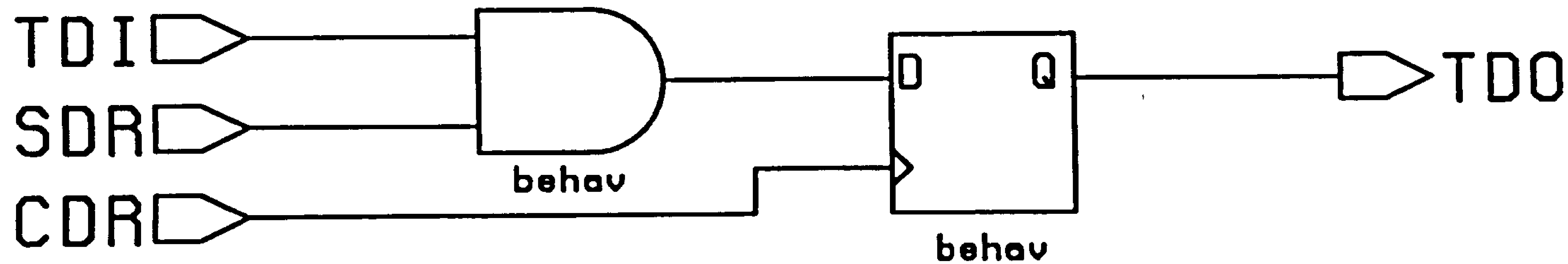

Instruction decoder for the
version with 4 instructions

/JTAG_VHDL/JTAG_0/DATA_REGISTER/VERSION_1

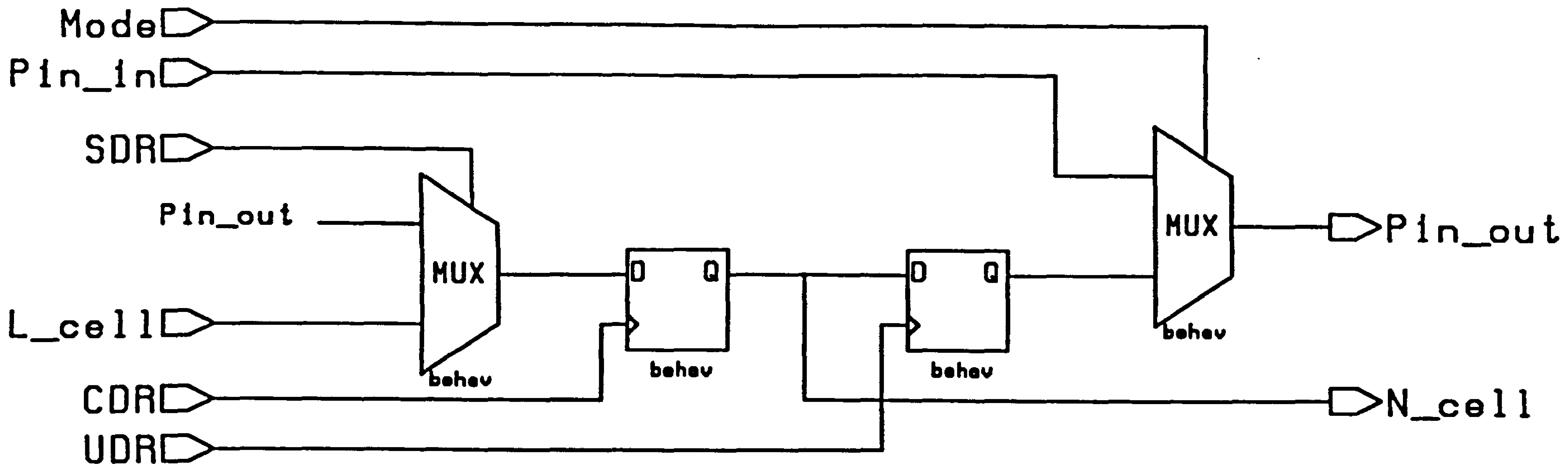



Data Register 2 states (5 in, 3 out)
Test of a 2 bits adder

/JTAG_VHDL/JTAG0/BYPASS_CELL

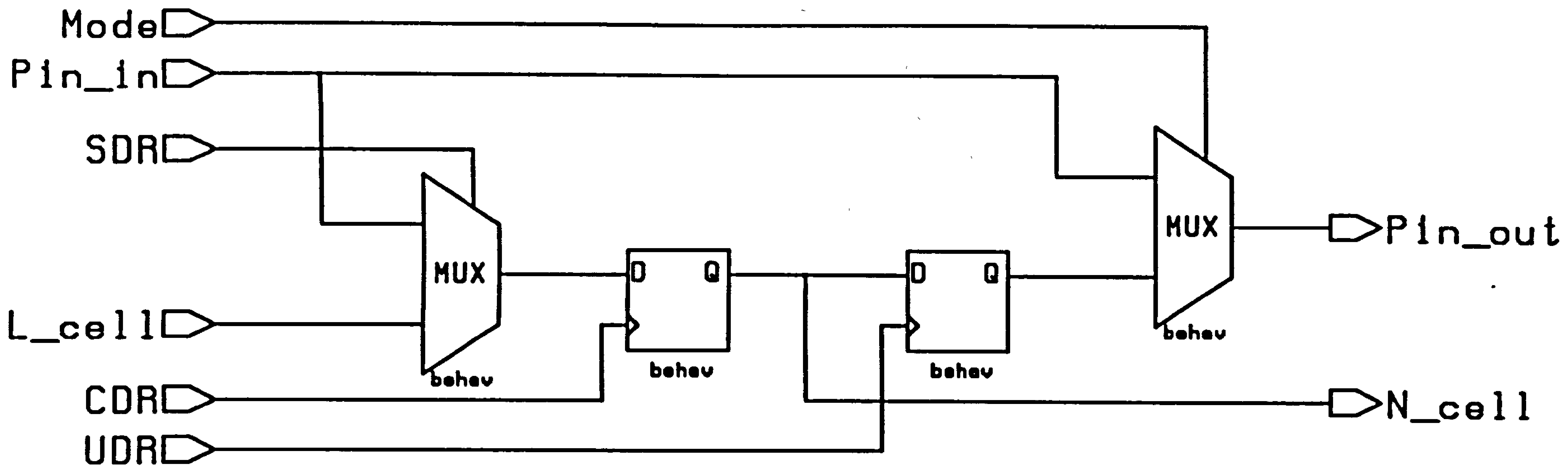


/JTAG_VHDL/JTAG0/BSCANI_CELL/VERSION_1



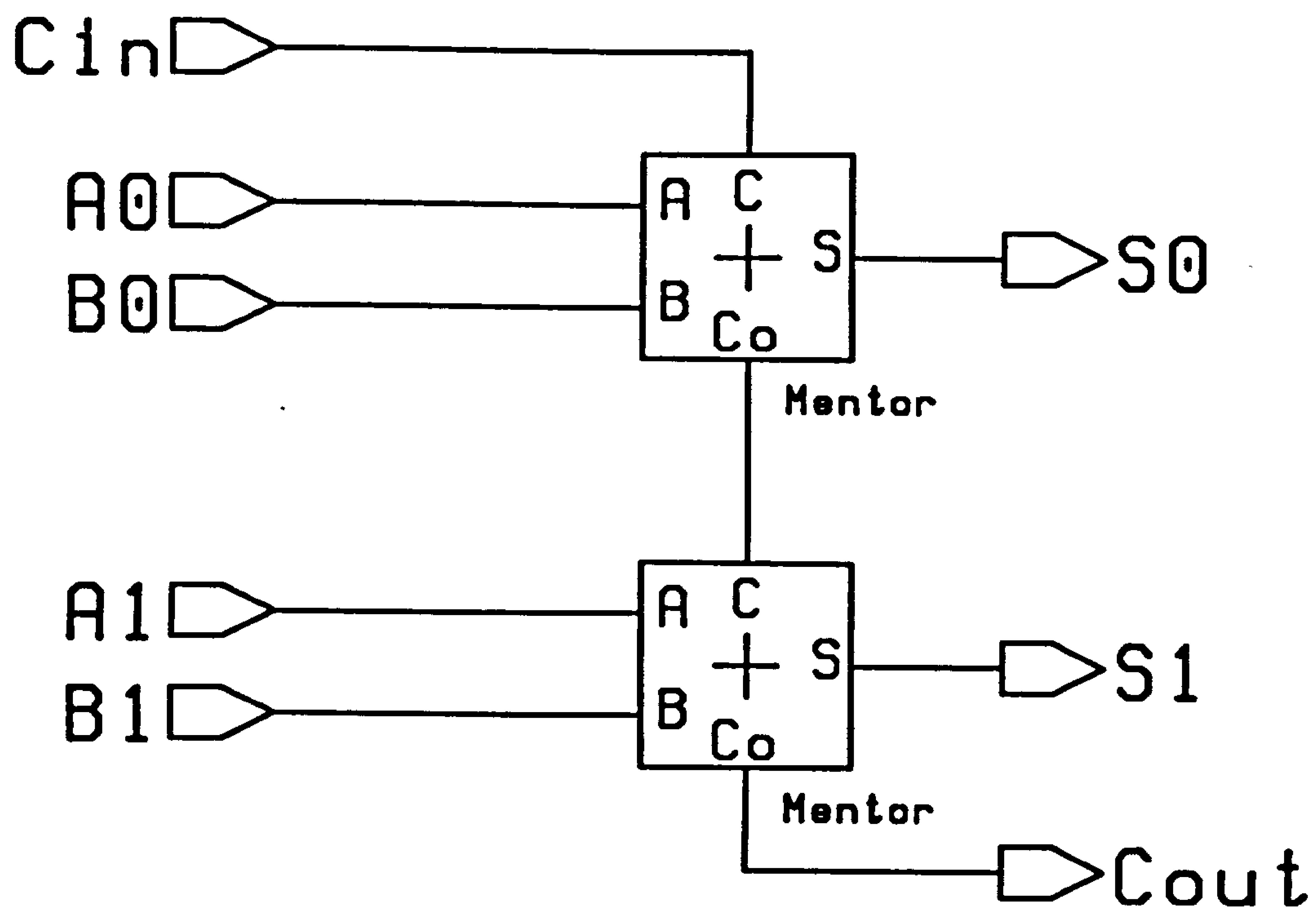

Enhanced boundary-scan cell for an input pin

/JTAG_VHDL/JTAG_0/BSCANO_CELL/VERSION_1



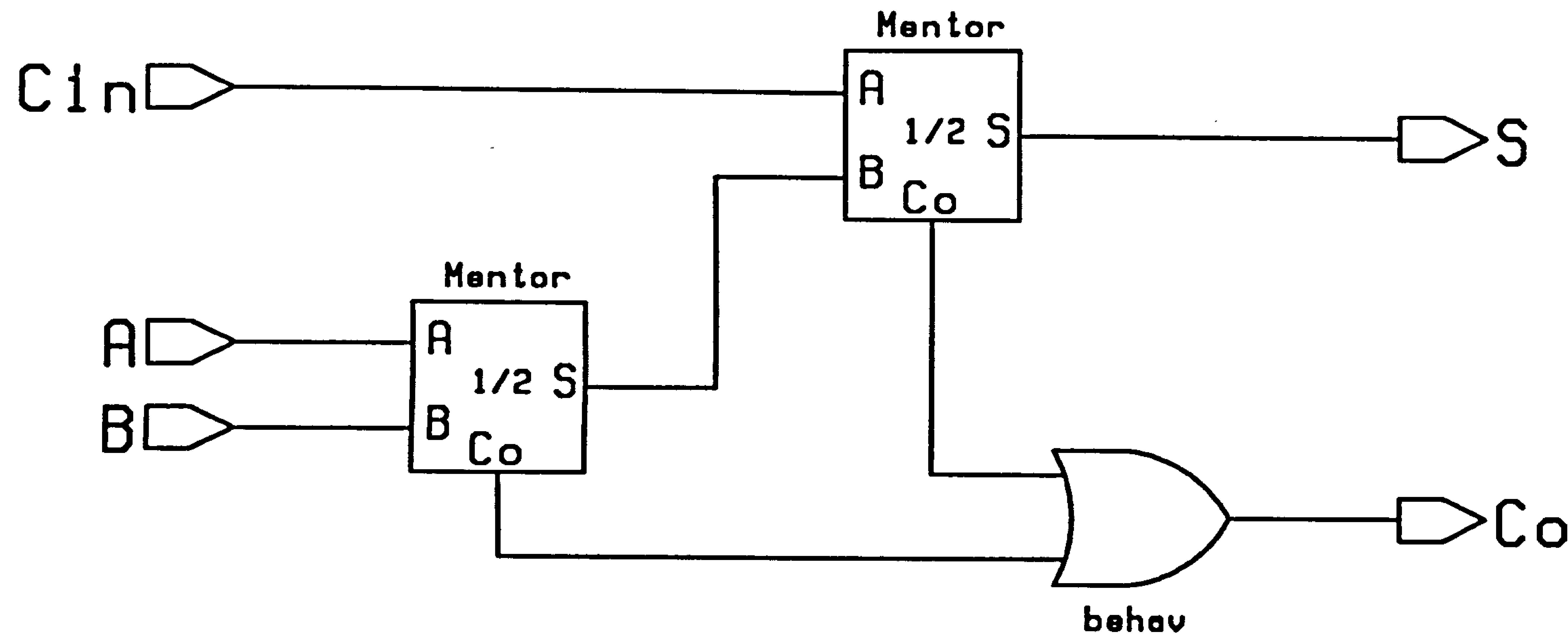

Enhanced boundary-scan cell for an output pin

/JTAG_VHDL/JTAG_0/ADDER_2BITS



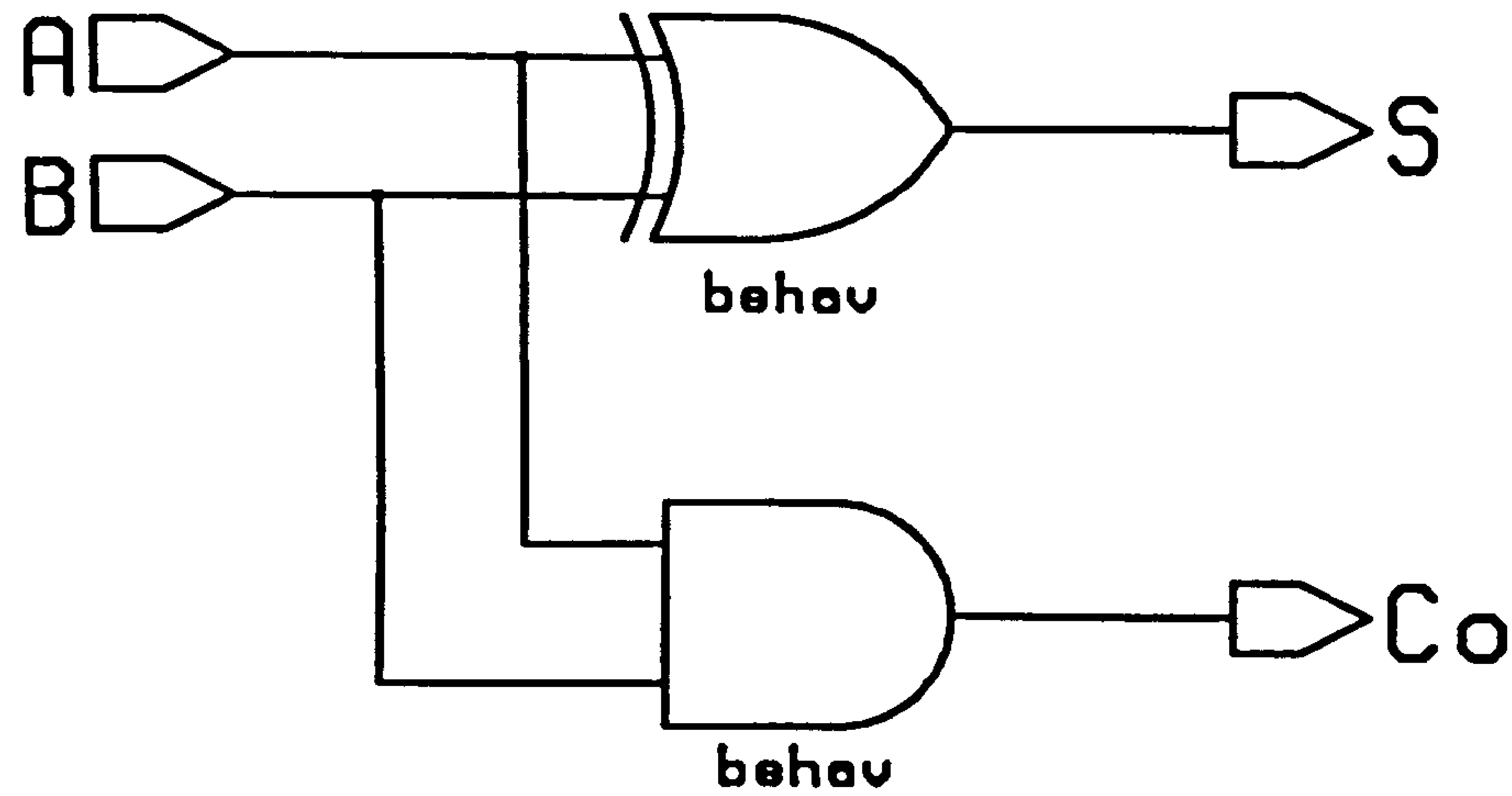

Adder 2 bits

/JTAG_VHDL/JTAG_0/ADDER_CELL




Full Adder Cell

/JTAG_VHDL/JTAG_0/HALF_ADDER_CELL




Half Adder cell

/JTAG_VHDL/JTAG0/CHIP2/TRACE_SIM



BEHAVIOURAL SIMULATION OF JTAG WITH APPLICATION LOGIC USING VHDL

August 1991

## Appendix 7A

'C' SOURCE CODE FOR VCP VERSION 1

```
/* Status VCP - version 1, final, 3rd August 1992*/

/* This program is used to extract top level I/O's from a
VHDL source file. The Output is a file containing a port
list and port type which is currently restricted to be of
type IN or OUT.*/

#include <stdio.h>
#include <ctype.h>
#include <string.h>

#define found 1
#define notfound 0
#define false (1==2)
#define true  (1==1)
#define MAX_KEYWORD_LENGTH 255
#define MAX_FILENAME_LENGTH 255
#define MAX_NO_INPUTS 100
#define MAX_NO_OUTPUTS 100
#define NAME_LENGTH 255

char filename[MAX_FILENAME_LENGTH];
FILE *vhdl_source;
FILE *port_names;
char array_of_inputs[MAX_NO_INPUTS][NAME_LENGTH];
char array_of_outputs[MAX_NO_OUTPUTS][NAME_LENGTH];
char keyword[MAX_KEYWORD_LENGTH];
int find_keyword();
int no_of_outputs;
int no_of_inputs;

main( int argc, char *argv[] )
{

 int array_index = 0;

 if ( argc != 2 )
 {
 printf(" \n You must supply a VHDL source code filename
as an argument to this program e.g. parse_vhdl <filename>
\n");
 exit();
 }

 strcpy(filename,argv[1]);
 if ( (vhdl_source = fopen(filename,"r")) == NULL )
 {
  printf("\n The selected file does not exist in current
working directory. \n");
  exit();
 }

 printf("\nParsing file %s \n \n",filename);
```

```
 printf("Looking for ENTITY clause \n \n");
 printf("<1>");
 strcpy(keyword,"ENTITY");
 printf("<2>");
 if ( find_keyword("ENTITY") )
 printf("Found ENTITY clause \n \n");
 else
 {
 printf("ERROR ENTITY clause not found in VHDL source \n
\n");
 exit();
 }
 printf("<3>");
 printf("Looking for PORT clause \n \n");
 printf("<4>");
 strcpy(keyword,"PORT");
 printf("5");
 if ( find_keyword("PORT") )
 printf("FOUND PORT clause \n \n");
 else
 {
  printf("ERROR PORT clause not found in VHDL source \n
\n");
  exit();
 }

/* Get Port data */
 printf("<6>");
 make_port_array();

 printf("Writing list of ports to vhdl_port_file \n");

 port_names = fopen("vhdl_port_file","w");

 fputs("INPUTS: \n",port_names);

 for(array_index = 0; array_index != no_of_inputs;
array_index++)
 {
/*  printf("Input %d is %s
\n",array_index,array_of_inputs[array_index]);*/
  fputs(array_of_inputs[array_index],port_names);
  fputs("\n",port_names);
 }

 fputs("OUTPUTS: \n",port_names);

 for(array_index = 0; array_index != no_of_outputs;
array_index++)
 {
 /* printf("Output %d is %s
\n",array_index,array_of_outputs[array_index]); */
  fputs(array_of_outputs[array_index],port_names);
  fputs("\n",port_names);
 }
```

```
 fclose(vhdl_source);
 fclose(port_names);

 exit(); /*temp*/

}


/*************************************************************
*************************************************************/
/* function to test for existaence of keyword in a VHDL
file */

find_keyword( keyword )

char keyword[MAX_KEYWORD_LENGTH];
{

 int pos;
 int word= false;
 char pres_string[MAX_KEYWORD_LENGTH];
 char pres_char;
 printf("<Key 1>");
 printf("<Key 2>");
 while ((pres_char=toupper(getc(vhdl_source)))!= EOF)
   {
   if ((pres_char!=' ') && isalpha(pres_char))
     {
     if (!word)
       {
       pos=0;
       word=true;
       }
     pres_string[pos++]=pres_char;
     printf(".");
     }
   else
     if (word)
       {
       pres_string[pos]=NULL;
       printf(" <%s>\n",pres_string);
       if (strcmp(keyword,pres_string)==0)
      return(found);
       else
      word=false;
       }
     else
      printf("+");
   }

   return(notfound);
   }
/*************************************************************
*************************************************************/
```

```
/**********************************************************
*********************************************************/
/* This function generates IN and OUT arrays to hold the
port names */
/* were at the end of port and scanning for the port
names */

make_port_array()
{
 char port_char;
 int order;
 int char_index;

  order = 0;
  char_index = 0;
  printf("<11>");
  port_char = getc(vhdl_source);
  printf("<12>");
  while (!isalnum(port_char))   /* find start of inputs
ASSUMES inputs start with an alphabetic character ??? */
  {
   port_char = getc(vhdl_source);
  }
  printf("<13>\n");
  while ( port_char != ':')      /* build list of input
ports */
  {

   while(isalnum(port_char))
    {
     array_of_inputs[order][char_index++] = port_char;
/* need string array to hold names of max len */
     port_char = getc(vhdl_source);
    }
   array_of_inputs[order][char_index] = NULL;
   printf("-<%s>-\n",array_of_inputs[order]);
   order++;                     /* next port */
   char_index = 0;
   no_of_inputs = order;
   while ((!isalnum(port_char)) && (port_char!=':'))
     port_char = getc(vhdl_source);
  }

  while ( port_char != ';' )   /* find end of input
declaration */
  {
   port_char = getc(vhdl_source);
  }

  while ( !isalnum(port_char) )        /* find start of
output list */
  {
    port_char = getc(vhdl_source);
  }

  order = 0;
  char_index = 0;
```

```
  while ( port_char != ':')       /* build list of output
ports */
  {
    while ( isalnum(port_char)  )
    {
      array_of_outputs[order][char_index++] = port_char;
/* need string array to hold names of max len */
      port_char = getc(vhdl_source);
    }

   array_of_outputs[order][char_index] = NULL;
   printf("+<%s>+\n",array_of_outputs[order]);
   order++;                        /* next port */
   char_index = 0;
   no_of_outputs = order - 1;
   while ((!isalnum(port_char)) && (port_char!=':'))
     port_char = getc(vhdl_source);
  }
}*
```

## Appendix 7B

'C' SOURCE CODE FOR VCP VERSION 2

```
/*
  ***********************************************************
  *                                                         *
  *  Extract top level I/Os from a VHDL source.             *
  *                                                         *
  *  EXTRACT.C                                              *
  *                                                         *
  *  BY S.MEDHAT  Version 1.3                               *
  *  Bournemouth University (U.K) 26/09/92                  *
  *                                                         *
  *  This software program reads a VHDL design source       *
  *  file and extracts the top level I/Os from its          *
  *  entity description generating an output file           *
  *  called filemane.PIN where all I/O  names, modes,       *
  *  types and bus width are listed.                        *
  *                                                         *
  ***********************************************************

  The  architecture of  EXTRACT.C is based upon a state
  machine  structure behaviour  which  consists  of 21
  states.

  The prototype of EXTRACT call is :

          extract <design_file_name>[.hdl]


***********************************************************/

#include "lib.h"  /* Include string macro-functions */


/************** Declaration of the 21 states ************/

typedef enum {
               in,
               out,
               inout,
               buffer,
               port1,
               port2,
               bracket,
               word,
               blank1,
               coma,
               colon,
               type,
               key,
               blank2,
               bus,
               blank3,
               size,
               blank4,
               semi_colon,
               end,
               errors
               } states_t;
```

```
/********** Declaration of function prototypes *********/

void format_data(); /* Format output data */
void comments();    /* Handle comments */


/********** Declaration of global variables ************/

int  top_size;
int  bottom_size;
int  bus_size;
int  pin_no;
int  error_no;

char *ptr_word;
char *ptr_type;
char *ptr_bus;
char in_filename[50]; /* Name of the input file  */
char out_filename[50];/* Name of the output file */
char type_name[20];   /* Type name storage */
char bottom[5];       /* Low index bus     */
char top[5];          /* High index bus    */
char bus_name[20];    /* Bus name storage  */
char base_mem[1000];  /* Data processing memory */

FILE *infile;  /* Stream of input resource  */
FILE *outfile; /* Stream of output resource */

boolean stop;  /* Exit program */
boolean error; /* Error detected */

enum states_t token; /* Token of the pseudo-state machine */


/*
  *******************************************************
  *                                                     *
  * EXTRACT program body                                *
  *                                                     *
  *******************************************************
*/


main(int argc, char *argv[])
 {

  /*** Usage errors detection ***/

  if (argc!=2)
   {
      printf(" Usage: extract
<design_file_name>[.hdl]\n");
    exit();
     }
```

```
/*** Filename errors detection ***/

strcpy(in_filename,argv[1]);
strcpy(out_filename,argv[1]);
strcat(in_filename,".hdl");
strcat(out_filename,".pin");
if ((infile=fopen(in_filename,"r"))==NULL)
 {
    printf("\n<Extract/Error>: \"%s\"
           does not exist\n",in_filename);
  exit();
 }

if ((outfile=fopen(out_filename,"w"))==NULL)
 {
    printf("\n<Extract/System>:Cannot open output
           file (%s)\n",out_filename);
  exit();
   }


/*** Insert the output file header ***/

fprintf(outfile,"#    PIN NAME              PIN MODE
        PIN TYPE          BUS WIDTH    STARTING\n");
fprintf(outfile,"----------------------------------
                 -----------------------------------------
\n");


/*** Send a message on the screen (Start processing) ***/

printf("\n<Extract/Note>:Extracting File \"%s\"\n"
       ,in_filename);


/*** Initialisation of variables ***/

ptr_word=base_mem;
ptr_type=type_name;
ptr_bus=bus_name;
stop=error=FALSE;
token=in;            /* Token in state "IN" */


/**************** Start data processing ***************/

while (!stop && !error)
{
switch (token)
 {

    /* State IN : Look for a keyword "IN" */
    /* if found --> state PORT1           */
    /* else     --> state OUT             */
```

```
case in       :
 {
  if (search_key(infile,"IN")) token=port1;
  else token=out;
  break;
   }


  /* State OUT : Look for a keyword "OUT" */
  /* if found --> state PORT1            */
  /* else     --> state INOUT            */

case out      :
 {
  if ((infile=fopen(in_filename,"r"))==NULL)
   {token=errors; error_no=15;}
  if (search_key(infile,"OUT")) token=port1;
  else token=inout;
  break;
 }


  /* State INOUT : Look for a keyword "INOUT" */
  /* if found --> state PORT1                */
  /* else     --> state BUFFER               */

case inout    :
 {
  if ((infile=fopen(in_filename,"r"))==NULL)
   {token=errors; error_no=15;}
  if (search_key(infile,"INOUT")) token=port1;
  else token=buffer;
  break;
 }


  /* State BUFFER : Look for a keyword "BUFFER" */
  /* if found --> state PORT1                  */
  /* else     --> state ERROR(1)               */

case buffer    :
 {
  if ((infile=fopen(in_filename,"r"))==NULL)
   {token=errors; error_no=15;}
  if (search_key(infile,"BUFFER")) token=port1;
  else {token=errors; error_no=1;}
  break;
 }


  /* State PORT1 : Look for the keyword "PORT" */
  /* if found --> state PORT2                 */
  /* else     --> state ERROR(2)              */

case port1    :
 {
  if ((infile=fopen(in_filename,"r"))==NULL)
```

```
  {token=errors; error_no=15;}
 if (search_key(infile,"PORT")) token=port2;
 else {token=errors; error_no=2;}
 break;
}


 /* State PORT2 : Look for the next left-bracket */
 /* if found   --> state BRACKET                 */
 /* if comment --> state COMMENTS                */
 /* if space   --> state PORT2                   */
 /* if other   --> state ERROR(3)                */

case port2   :
 {
     if (isspace(next_char))
       {token=port2; next_char=fgetc(infile);}
 else if (next_char=='(') token=bracket;
   else if (next_char=='-') comments();
     else {token=errors; error_no=3;}
 break;
 }


 /* State BRACKET : Look for next word */
 /* if found   --> state WORD          */
 /* if comment --> state COMMENTS      */
 /* if space   --> state BRACKET       */
 /* if other   --> state ERROR(5)      */

case bracket :
 {
 if (!next_char_separ(infile,VHDL)) token=word;
 else if (isspace(next_char)) token=bracket;
   else if (next_char=='-') comments();
     else {token=errors; error_no=5;}
 break;
 }


 /* State WORD:Read the next char of the port name*/
 /* if coma       --> state COMA                  */
 /* if colon      --> state COLON                 */
 /* if VHDL Ascii --> state WORD                  */
 /* if other      --> state ERROR(5)              */

case word    :
 {
 *ptr_word++=toupper(next_char);
 while (!next_char_separ(infile,VHDL))
*ptr_word++=next_char;
 *ptr_word++='\0';
 if (isspace(next_char)) token=blank1;
 else if (next_char==':') token=colon;
   else if (next_char==',') token=coma;
     else {token=errors; error_no=5;}
 break;
 }
```

```
 }

  /* State BLANK1 : Ignore all blanks char */
  /* if coma        --> state COMA         */
  /* if colon       --> state COLON        */
  /* if space       --> state BLANK1       */
  /* if comment     --> state COMMENTS     */
  /* if other       --> state ERROR(6)     */

case blank1  :
 {
  if (isspace(next_char=fgetc(infile))) token=blank1;
  else if (next_char=='-') comments();
    else if (next_char==':') token=colon;
      else if (next_char==',') token=coma;
           else {token=errors; error_no=6;}
  break;
 }

  /* State COMA : Look for the next port name */
  /* if VHDL Ascii --> state WORD             */
  /* if space      --> state BLANK1           */
  /* if comment    --> state COMMENTS         */
  /* if other      --> state ERROR(5)         */

case coma    :
 {
  if (!next_char_separ(infile,VHDL)) token=word;
  else if (next_char=='-') comments();
      else if (!isspace(next_char))
        {token=errors; error_no=5;}
  break;
 }

  /* State COLON : Ignore all blanks char */
  /* if space      --> state COLON        */
  /* if VHDL Ascii --> state TYPE         */
  /* if comment    --> state COMMENTS     */
  /* if other      --> state ERROR(5)     */

case colon   :
 {
  if (!next_char_separ(infile,VHDL))
   { token=type; *ptr_type++=toupper(next_char);}
  else if (next_char=='-') comments();
        else if (!isspace(next_char))
          {token=errors; error_no=5;}
  break;
 }

  /* State TYPE : Look for the next port type */
  /* if VHDL Ascii --> state TYPE             */
  /* if space      --> state KEY              */
  /* if other      --> state ERROR(5)         */
```

```
    case type     :
     {
      if (!next_char_separ(infile,VHDL))
       {token=type; *ptr_type++=toupper(next_char);}
      else
       {
           *ptr_type='\0';
           ptr_type=type_name;
           if (isspace(next_char)) token=key;
           else {token=errors; error_no=5;}
       }
      break;
     }


/* State KEY : Check the validity of the port type */
/* if OK --> state BLANK2                          */
/* else  --> state ERROR(7)                        */

    case key     :
     {
      token=blank2;
      if ((strcmp(type_name,"IN"))!=0)
          if ((strcmp(type_name,"OUT"))!=0)
            if ((strcmp(type_name,"INOUT"))!=0)
              if ((strcmp(type_name,"BUFFER"))!=0)
                {token=errors; error_no=7;}
      break;
     }


      /* State BLANK2 : Ignore all blanks char */
      /* if space      --> state BLANK2        */
      /* if VHDL Ascii --> state BUS           */
      /* if comment    --> state COMMENTS      */
      /* if other      --> state ERROR(5)      */

    case blank2  :
     {
      bus_size=1;
      bottom_size=0;
      if (!next_char_separ(infile,VHDL))
       {token=bus; *ptr_bus++=toupper(next_char);}
      else if (isspace(next_char)) token=blank2;
          else if (next_char=='-') comments();
              else {token=errors; error_no=5;}
      break;
     }


/* State BUS : Read the next char of the bus name */
/* if left-bracket  --> state SIZE                */
/* if right-bracket --> state END                 */
/* if space         --> state BLANK3              */
/* if VHDL Ascii    --> state BUS                 */
/* if other         --> state ERROR(5)            */
```

```
case bus      :
 {
  if (!next_char_separ(infile,VHDL))
      *ptr_bus++=tolower(next_char);
  else
   {
      *ptr_bus++='\0';
      ptr_bus=bus_name;
      if (isspace(next_char)) token=blank3;
        else if (next_char==';')
          {token=semi_colon; format_data();}
        else if (next_char=='(') token=size;
            else if(next_char==')') token=end;
              else {token=errors; error_no=5;}
   }
  break;
 }


  /* State BLANK3 : Ignore all blanks char */
  /* if left-bracket --> state SIZE        */
  /* if semi-colon   --> state SEMI_COLON  */
  /* if space        --> state BLANK3      */
  /* if comment      --> state COMMENTS    */
  /* if other        --> state ERROR(8)    */

case blank3  :
 {
  if (isspace(next_char=fgetc(infile))) token=blank3;
      else if (next_char==';')
        {token=semi_colon; format_data();}
      else if (next_char=='(') token=size;
          else if(next_char==')') token=end;
            else if (next_char=='-') comments();
                else {token=errors; error_no=5;}
  break;
 }


  /* State SIZE : Look for the next separater  */
  /* if right-bracket --> state END            */
  /* if semi-colon    --> state SEMI_COLON     */
  /* if space         --> state BLANK4         */
  /* if other         --> state ERROR(9,10,11) */

case size    :
 {

bottom_size=atoi(*bottom=take_next_word(infile,KEY));
  if ((strcmp(take_next_word(infile,KEY),"TO"))!=0)
      {token=errors; error_no=11;}
  else
   {
      top_size=atoi(*top=take_next_word(infile,KEY));
        if ((bus_size=top_size-bottom_size)<2)
           {token=errors; error_no=9;}
      else if (next_char==')') token=blank4;
        else {token=errors; error_no=10;}
```

```
      }
     break;
    }


     /* State BLANK4 : Ignore all blanks char */
     /* if right-bracket --> state END         */
     /* if semi-colon   --> state SEMI_COLON   */
     /* if space        --> state BLANK4       */
     /* if comment      --> state COMMENTS     */
     /* if other        --> state ERROR(12)    */

   case blank4   :
    {
         do next_char=fgetc(infile);
         while (isspace(next_char));
     if(next_char==')') token=end;
         else if (next_char==';')
          {token=semi_colon; format_data();}
         else if (next_char=='-') comments();
             else {token=errors; error_no=12;}
     break;
    }


 /* State SEMI_COLON : Look for the next port name */
 /* if VHDL Ascii --> state WORD                   */
 /* if space      --> state SEMI_COLON             */
 /* if comment    --> state COMMENTS               */
 /* if other      --> state ERROR(5)               */

   case semi_colon :
    {
     if (!next_char_separ(infile,VHDL)) token=word;
     else if (isspace(next_char)) token=semi_colon;
       else if (next_char=='-') comments();
         else {token=errors; error_no=5;}
     break;
    }


   /* State END : Terminal state */

   case end      :
    {
     *ptr_word='\0';
     format_data();
         printf("<Extract/Note>:Writing \"%s\"\n",
                out_filename);
     stop=TRUE;
     break;
    }
```

```
/* State ERRORS : Print a error message on screen */

case errors  :
 {
  switch (error_no)
   }
   {
         case  1 : {printf("<Extract/Error>:No IN, OUT,
          INOUT or BUFFER in this design\n"); break;}
         case  2 : {printf("<Extract/Error>:Cannot find the
          PORT declaration statement\n"); break;}
         case  3 : {printf("<Extract/Error>:<Left-bracket>
          expected but '%c' found\n",next_char); break;}
         case  4 : {printf("<Extract/Error>:Comment expected
          but '%c' found\n",next_char); break;}
         case  5 : {printf("<Extract/Error>:Alphanum
            character expected but '%c' found\n",next_char);
          break;}
         case  6 : {printf("<Extract/Error>:<Coma> or
            <Colon> expected but '%c'found\n",next_char);
          break;}
         case  7 : {printf("<Extract/Error>: '%s' unknown
            port-type <IN,OUT,INOUT,BUFFER>\n",type_name);
          break;}
         case  8 : {printf("<Extract/Error>:<Semi-colon> or
            <bracket> expected but '%c' found\n",next_char);
            break;}
         case  9 : {printf("<Extract/Error>:Erroneous bus
          size\n"); break;}
         case 10 : {printf("<Extract/Error>:<Right-bracket>
          expected but '%c' found\n",next_char); break;}
         case 11 : {printf("<Extract/Error>:<TO> expected
          but '%c' found\n",next_char); break;}
         case 12 : {printf("<Extract/Error>:<Semi-colon> or
            <Right-bracket> expected but '%c' found\n",
            next_char); break;}
         case 13 : {printf("<Extract/Error>:<EOF> reached
          but data extracting unfinished\n"); break;}
         case 14 : {printf("<Extract/System>:I/O conflict
          on the output file (%s)\n",out_filename); break;}
         case 15 : {printf("<Extract/System>:I/O conflict
          on the input file (%s)\n",in_filename); break;}

      /** Insert the last data line in the output file **/

  if (error_no!=14) format_data();
  error=TRUE;
  break;
 }

 }
}

/*** Close all the file and quit program ***/
if ((fclose(infile))!=NULL)
    printf("<Extract/System>:Cannot close properly the
          file %s\n",in_filename);
if ((fclose(outfile))!=NULL)
```

```
    printf("<Extract/System>:Cannot close properly the
           file %s\n",out_filename);
 }


/*
 ********************************************************
 *                                                      *
 * Function FORMAT_DATA                                 *
 *                                                      *
 ********************************************************

 This function write data processed into the output
 file with a pre-defined format.

*/

void format_data()
 {
  *ptr_word='\0';
  ptr_word=base_mem;
  while (*ptr_word)
   {
     fprintf(outfile,"%-3d %-20s %-9s  %-20s  %2d
       %2d\n",pin_no,ptr_word,type_name,bus_name,
       bus_size,bottom_size);
    while (*ptr_word++);
    pin_no++;
   }
  ptr_word=base_mem;
  *ptr_word='\0';
 }


/*
 ********************************************************
 *                                                      *
 * Function COMMENT                                     *
 *                                                      *
 ********************************************************

 This function ignores all characters included in a
 comment  procedure then  comes back at the current
 state.

*/

void comments()
 {
  if ((next_char=fgetc(infile))=='-')
   {
     while (((next_char=fgetc(infile))!='\0') &&
      (next_char!=EOF));
    if (next_char==EOF) {token=errors; error_no=13;}
   }
  else {token=errors; error_no=4;}
 }
```

```
/* Library of file functions for the VHDL parser */

#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define TRUE  (1==1)
#define FALSE (1==0)
#define FOUND    TRUE
#define NOTFOUND FALSE
#define VHDL " !'#$%`(){}+,-&.\/:;<=>?[]\\^\n\t|\"~\0"

typedef unsigned char boolean;

typedef enum {NAME,KEY} format;

boolean next_char_separ(FILE *infile,char *separ_list);
int     find_next_separ(FILE *infile,char *separ_list);
char    *take_next_word(FILE *infile,enum format
out_fmt);
int     search_key(FILE *infile,char *keyword);

char next_char;


/***********************
*                     *
*   NEXT_CHAR_SEPAR   *
*                     *
***********************/
boolean next_char_separ(infile,separ_list)
FILE *infile;     /* Point on the input file */
char *separ_list; /* List of characters of separation */
{
  boolean cont;

  next_char=fgetc(infile);
  if (next_char!=EOF)
    if (*separ_list)
      {
       while ((*separ_list) &&
(cont=(next_char!=*separ_list)))
      separ_list++;
       return(!cont);
      }
    else
      return (!isalnum(next_char));
  else
    return(TRUE);
}
```

```
/***********************
*                      *
*   FIND_NEXT_SEPAR    *
*                      *
***********************/
int find_next_separ(infile,separ_list)
FILE *infile;     /* Point on the input file */
char *separ_list; /* List of characters of separation */
{

 while(!next_char_separ(infile,separ_list));
   return(next_char);
}


/**********************
*                     *
*   TAKE_NEXT_WORD    *
*                     *
**********************/
char *take_next_word(infile,out_fmt)
FILE *infile;         /* Point on the input file */
enum format out_fmt;  /* Filtering of output flow */
{
 char word[30],*ptr;
 int (*filter1)(),(*filter2)();

 ptr=word;
 if (out_fmt==KEY) filter1=filter2=toupper;
 else
  {
   filter1=toupper;
   filter2=tolower;
  }
 while ((next_char_separ(infile,VHDL)) &&
(next_char!=EOF));
 if (next_char!=EOF)
  {
   *ptr++=(*filter1)(next_char);
   while (!next_char_separ(infile,VHDL))
     *ptr++=(*filter2)(next_char);
   *ptr='\0';
   return(&word);
  }
 else
   return(NULL);
}
```

```
/********************
*                   *
*    SEARCH_KEY     *
*                   *
********************/
int search_key(infile,keyword)
FILE *infile;  /* Point on the input file */
char *keyword; /* Point the reference Keyword */
{
 char *ptr;

 do
  {
   ptr=take_next_word(infile,KEY);
   if (strcmp(ptr,keyword)==0) return(FOUND);
  }
 while (*ptr!=NULL);
 return(NOTFOUND);
}
*
```

**Appendix 7C**

'C' SOURCE CODE FOR THE INSERTION PROGRAM

```
/***************************************************************/
/* JTAG insertion program. Takes the file "name".pin          */
/* from extract and inserts the jtag acrchitecture into      */
/* the vhdl decription.                                       */
/* The new entity becomes "name"_jtag.                        */
/* This is also the name of the                               */
/* resulting file.                                            */
/*                                                            */
/***************************************************************/

#include <stdio.h>
#include <string.h>
#include "lib.h"                /* defines types and
functions */

  port_struct port_list[200];   /* structure used to store
the
                                   port data */

  int num_ports;
  int num_int_sigs;            /* total number of signals */

  FILE *infile,*outfile;
  void insert_file();
  void insert_compdef();
  char get_space();
  void get_org_port_list();
  void put_int_signals();
  void put_bscan_cells();

main(argc, argv)
   int argc;
   char *argv[];                         /* contains file
name */
 {
  char in_filename[50],out_filename[50];
  char ent_name[50];
  int n;

  if (argc!=2)
   {
    printf(" Usage: insert <pin_file_name[.pin]>\n");
    exit();
   }
  strcpy(ent_name,argv[1]);
  strcpy(in_filename,argv[1]);
  strcpy(out_filename,argv[1]);
  strcat(in_filename,".pin");
  strcat(out_filename,"_jtag.vhdl");
  if ((infile=fopen(in_filename,"r"))==NULL)

   {
    printf("\n<Insert/Error>: \"%s\" does not
exist\n",in_filename);
    exit();
   }
```

```
  if ((outfile=fopen(out_filename,"w"))==NULL)  /* check
that file exists. */
   {
    printf("\n<Insert/System>:Cannot open output file
(%s)\n",out_filename);
    exit();
   }
   insert_file("jtag.use",outfile);
/* add library declarations */
   fprintf(outfile,"\nentity %s_jtag is\n",ent_name);
   fprintf(outfile,"   port(TCK, TMS, TDI : in bit;\n");
/* put port declarations */
   fprintf(outfile,"        TDO : out logic4;\n");
   get_org_port_list(in_filename);
/* fill port structure */
   for (n = 0; n < num_ports; n++) {
       fprintf(outfile,"        %s : %s
%s",port_list[n].name,port_list[n].mode,port_list[n].type);
       if (port_list[n].size > 1)
           fprintf(outfile,"(%d to
%d)",port_list[n].min,port_list[n].max);
       if (n == num_ports-1)
           fprintf(outfile,");\n");
       else
           fprintf(outfile,";\n");
   }
   fprintf(outfile,"end %s_jtag;\n",ent_name);
   fprintf(outfile,"\narchitecture behav_jtag of %s_jtag
is\n\n",ent_name);
   insert_file("jtag.comps",outfile);                    /*
add component declarations */
   insert_compdef(ent_name,outfile);
   fprintf(outfile,"\n");
   insert_file("jtag.signals",outfile);                  /*
add standard signals */
   put_int_signals(outfile);
   fprintf(outfile,"\nbegin\n");
   insert_file("jtag.insts",outfile);    /* add standard
instances */
   put_bscan_cells(outfile);
   fprintf(outfile,"end behav_jtag;\n");
}

void insert_file(in_name,out_file)        /* routine for
inserting ascii files to a stream */
  char *in_name;
  FILE *out_file;
{



   char c;
   FILE *in_file;
   if ((in_file=fopen(in_name,"r"))==NULL)
    {
     printf("\n<Insert/Error>: \"%s\" does not
exist\n",in_name);
     exit();
```

```
  }
  while((c = fgetc(in_file)) != EOF)
     fputc(c,out_file);
  }

void insert_compdef(ent_name,file)       /* inserts the
original component definition */
  char *ent_name;
  FILE *file;
{
  int n;

  fprintf(file,"component %s\n",ent_name);
  fprintf(file,"port (\n");
  for (n = 0; n < num_ports; n++) {
      fprintf(file,"        %s : %s
%s",port_list[n].name,port_list[n].mode,port_list[n].type);
      if (port_list[n].size > 1)
          fprintf(file,"(%d to
%d)",port_list[n].min,port_list[n].max);
      if (n == num_ports-1)
          fprintf(file,");\n");
      else
          fprintf(file,";\n");
  }

  fprintf(file,"end component;\n");
}

char get_space(file)                  /* read blank space from
input stream */
   FILE *file;
{
   char c;

  while ((c = fgetc(file)) != EOF) {
     if (!isspace(c))
        break;
  }
  return(c);
}


void get_org_port_list(pin_file_name)        /* converts pin
file into an internal structure */
  char *pin_file_name;



  {
  FILE *pin_file;
  char *ptr_name;
  char *ptr_type;
  char *ptr_mode;
  port_struct *ptr_struct;
  char c;
  int width,bottom,top;
```

```
int n;

pin_file = fopen(pin_file_name,"r");
ptr_struct = port_list;
num_ports = -1;

do {
   ptr_name = ptr_struct->name;
   ptr_type = ptr_struct->type;
   ptr_mode = ptr_struct->mode;

   do {
     if (isdigit(c))                    /* find first line
of pin file */
          break;
   }
   while ((c = fgetc(pin_file)) != EOF);

   while ((c = fgetc(pin_file)) != EOF) {
     if (isspace(c))
       break;
     }

   c = get_space(pin_file);
   *ptr_name++ = c;
   while ((c = fgetc(pin_file)) != EOF) {
     if (isspace(c))
       break;
     else
       *ptr_name++ = c;                 /* get port name */
     }
   *ptr_name = '\0';
   c = get_space(pin_file);
   *ptr_mode++ = c;
   while ((c = fgetc(pin_file)) != EOF) {
     if (isspace(c))
       break;
     else
       *ptr_mode++ = c;                 /* get port mode */
    }
   *ptr_mode = '\0';
   c = get_space(pin_file);
   *ptr_type++ = c;
   while ((c = fgetc(pin_file)) != EOF) {
      if (isspace(c))
         break;
      else
         *ptr_type++ = c;               /* get port type */
    }
    *ptr_type = '\0';
    c = get_space(pin_file);
    ungetc(c,pin_file);
    fscanf(pin_file,"%d",&width);
    c = get_space(pin_file);
    ungetc(c,pin_file);
    fscanf(pin_file,"%d",&bottom);
    top = bottom + width;
    ptr_struct->max = top;
```

```
      ptr_struct->min = bottom;           /* get size of
arrays */
      ptr_struct->size = width;

      c = fgetc(pin_file);
      ptr_struct++;
      num_ports++;
      } while (c != EOF);

}

void put_int_signals(file)    /* declare the signals between
the bscan cells and the component */
   FILE *file;
{
   int n;

   num_int_sigs = 0;
   for (n = 0; n < num_ports; n++)
      if (port_list[n].size == 1) {
         fprintf(file,"signal int_%s :
%s;\n",port_list[n].name,port_list[n].type);
         num_int_sigs++;
      }
      else {
         fprintf(file,"signal int_%s : %s(%d to
%d);\n",port_list[n].name,port_list[n].type,port_list[n].min
,port_list[n].max);
         num_int_sigs+= port_list[n].size;
      }
   fprintf(file,"signal nextt : bit_vector(0 to
%d);\n",num_int_sigs);
}

void put_bscan_cells(file)          /* add the bscan cells
to the file */
   FILE *file;
{
   int n,a;
   int gen = 0;
   int sig_num=0;

   for (n = 0;n < num_ports;n++) {
     if (strncmp(port_list[n].mode,"IN",2) == 0 &&
port_list[n].size == 1) {
        fprintf(file,"   bscan%d : bscan port map(shiftdr,
clockdr, updatedr, nextt(%d), %s, mode, nextt(%d),
int_%s);\n",n,sig_num,port_list[n].name,sig_num+1,port_list[
n].name);
        sig_num++;
     }
     else if (port_list[n].size == 1) {
        fprintf(file,"   bscan%d : bscan port map(shiftdr,
clockdr, updatedr, nextt(%d), int_%s, mode, nextt(%d),
%s);\n",n,sig_num,port_list[n].name,sig_num+1,port_list[n].n
ame);
        sig_num++;
     }
```

```
     else if (strncmp(port_list[n].mode,"IN",2) == 0) {
/* use generate for buses */
        fprintf(file,"   G%d : for I in 0 to %d
generate\n",gen,port_list[n].size-1);
        fprintf(file,"      bscan_gen%d : bscan port
map(shiftdr, clockdr, updatedr, nextt(%d + I), %s(I+%d),
mode ,nextt(%d+I+1),
int_%s(%d+I));\n",gen,sig_num,port_list[n].name,port_list[n]
.min,sig_num, port_list[n].name,port_list[n].min);
        fprintf(file,"   end generate;\n");
        sig_num+= port_list[n].size;
        gen++;
     }
     else {
        fprintf(file,"   G%d : for I in 0 to %d
generate\n",gen,port_list[n].size-1);
        fprintf(file,"      bscan_gen%d : bscan port
map(shiftdr, clockdr, updatedr, nextt(%d + I), int_%s(I+%d),
mode ,nextt(%d+I+1),
%s(%d+I));\n",gen,sig_num,port_list[n].name,port_list[n].min
,sig_num, port_list[n].name,port_list[n].min);
        fprintf(file,"   end generate;\n");
        sig_num+= port_list[n].size;
        gen++;
     }
   }
   fprintf(file,"\n   test_regs(2) <=
nextt(%d);\n",num_int_sigs);
}
```

**Appendix 7D**

EXTRACT/INSERT environment used with a DFF design example 1.

```
library vdeg_portable;
use vdeg_portable.types.all;
use work.declar.all;

entity dff_jtag is
   port(TCK, TMS, TDI : in bit;
        TDO : out logic4;
        D : INOUT Bit;
        Clk : IN Bit;
        Q : OUT Bit);
end dff_jtag;

architecture behav_jtag of dff_jtag is

component bscan
        generic (Setup_Time,
        Hold_Time,
        Mux_Del,
        Dtype_Del :Time := 3 ns);
        Port (ShiftDR, ClockDR, UpdateDR,
              Last_Cell, S_In, Mode:in bit;
              Next_Cell: inout bit;
              S_Out: out bit);

end component;

component tdo_buffer
 generic (setup_time, hold_time, Min_pulse_width, Del,
Tdo_del:Time := 3 ns);
 port(
 TCK,
 Enablee,
 Input:in Bit;
 TDO: out Logic4
 );
end component;

component bypass_reg
        generic (Setup_Time,
        Hold_Time,
        MPulse_width,
        AND_Del,
        Bypass_Del :Time := 3 ns);
        Port (ShiftDR, ClockDR,
              TDI: in bit;
              TDO: out bit);

end component;

component inst_decode
       generic (Instruction_set:bit_vector := "00000000";

DR_select_Set,Test_Mode_Set,Additional_Signals_Set
               :bit_vector := "00000000";
               Open_Check:Boolean := false;
               Dec_Del:time := 3 ns);
       port ( Instruction:in bit_vector(0 to 7);
              DR_Select: out bit_vector(0 to 7);
```

```
              Test_Mode: out bit_vector(0 to 7);
              Additional_Signals : out bit_vector(0 to
7));
end component;

component ident_reg
       generic
(setup_time,Hold_Time,MPulse_Width,Mux_Del:Time := 3 ns;
                 ID_code : bit_vector :=
"0000000000000001");
       port(
       Select_Ident,
       ShiftDR,
       ClockDR,
       TDI:in Bit;
       TDO: out Bit
       );
end component;

component reg_inst
   generic
(Setup_Shift_Time,Hold_Shift_Time,MPulse_Width_Shift,

Setup_Update_Time,Hold_Update_Time,MPulse_Width_Update,
            Mux_Del,Stage_Shift_Del,Stage_Update_Del:time
:= 3 ns);

   port(Reset,
        ClockIR,
        UpdateIR,
        ShiftIR,
        TDI:in bit;
        Status:in bit_vector(0 TO 7);
        Instruction: out bit_vector(0 TO 7);
        TDO: out bit);
end component;

component mux_1
        generic (
        instruction_set :bit_vector(0 to 7) :=
"00000000";
        tdo_test_data_registers_set : integer_vector :=
(0,0,0,0,0,0,0,0);
        mux_del :time := 3 ns);
        port (
        tdo_test_data_registers: in bit_vector(0 to 7);
        instruction : in bit_vector(0 to 7);
        tdo: out bit);
end component;

component mux_2
        generic (mux_del : time := 3 ns);
        port (
        tdo_test_data_registers,
        tdo_instruction,
        selectt : in bit;
        tdo : out bit);
end component;
```

```
component tap_c
        generic (Setup_Time, Hold_time,Min_Pulse_Width_1,
             Min_Pulse_Width_0, S_Odel, Odel: Time := 3
ns);
        PORT (
        TMS,
        TCK: in bit;
        Reset,
        Selectt,
        Enable,
        ShiftIR,
        ClockIR,
        UpdateIR,
        ShiftDR,
        UpdateDR,
        ClockDR : out bit
    );
end component;


component dff
port (
        D : INOUT Bit;
        Clk : IN Bit;
        Q : OUT Bit);
end component;

constant one_high : bit_vector(0 to 7) := "00000001";

signal temp : bit_vector(0 to 7) := one_high;
signal shiftir, clockir, updateir : bit;
signal shiftdr, clockdr, updatedr : bit;
signal mode, inst_tdo, indent_tdo, enablee, reset: bit;
signal data_tdo, selectt, select_id, buff_tdo, bypass_tdo
: bit;
signal dr_select, instruction, test_mode, test_regs :
bit_vector(0 to 7);
signal int_D : Bit;
signal int_Clk : Bit;
signal int_Q : Bit;
signal nextt : bit_vector(0 to 3);

begin
   test_regs(0) <= bypass_tdo;
   test_regs(1) <= indent_tdo;
   mode <= test_mode(0);
   select_id <= test_mode(1);
   nextt(0) <= TDI;

   tap : tap_c port map
(tms,tck,reset,selectt,enablee,shiftir,

clockir,updateir,shiftdr,updatedr,clockdr);
   bypass : bypass_reg port map
(shiftdr,clockdr,TDI,bypass_tdo);
```

```
   instruct : reg_inst port
map(reset,clockir,updateir,shiftir,TDI,temp,instruction,i
nst_tdo);
   ident : ident_reg port map(select_id, shiftdr,
clockdr, tdi, indent_tdo);
   decoder : inst_decode port map(instruction, dr_select,
test_mode, open);
   mux2 : mux_2 port map(data_tdo, inst_tdo, selectt,
buff_tdo);
   mux1 : mux_1 port map(test_regs, dr_select, data_tdo);
   tdo_buff : tdo_buffer port map(tck, enablee, buff_tdo,
tdo);
   bscan0 : bscan port map(shiftdr, clockdr, updatedr,
nextt(0), D, mode, nextt(1), int_D);
   bscan1 : bscan port map(shiftdr, clockdr, updatedr,
nextt(1), Clk, mode, nextt(2), int_Clk);
   bscan2 : bscan port map(shiftdr, clockdr, updatedr,
nextt(2), int_Q, mode, nextt(3), Q);

   test_regs(2) <= nextt(3);
end behav_jtag;
```

## Appendix 7E

EXTRACT/INSERT environment used with a CPU design example 2.

```
-- ibrary vdeg_portable; --
use vdeg_portable.types.all;
entity cpu is
port (
       clock,sel : in logic4;
       enable : in logic4;
       data : inout BusAnd(1 to 8);
       addr : inout BusAnd(1 to 8);
       zero : out logic4;
       rd : inout logic4;
       rd1 : inout logic4;
       wr1 : inout logic4;
       wr : inout logic4
      );

end cpu;
```

| # | PIN NAME | PIN MODE | PIN TYPE | BUS WIDTH | STARTING |
|---|---|---|---|---|---|
| 0 | Clock | IN | Logic4 | 1 | 0 |
| 1 | Sel | IN | Logic4 | 1 | 0 |
| 2 | Enable | IN | Logic4 | 1 | 0 |
| 3 | Data | INOUT | Busand | 7 | 1 |
| 4 | Addr | OUT | Busand | 7 | 1 |
| 5 | Zero | OUT | Logic4 | 1 | 0 |
| 6 | Rd | INOUT | Logic4 | 1 | 0 |
| 7 | Rd1 | INOUT | Logic4 | 1 | 0 |
| 8 | Wr1 | INOUT | Logic4 | 1 | 0 |
| 9 | Wr | INOUT | Logic4 | 1 | 0 |

```
component cpu is
port (
        Clock : IN Logic4;
        Sel : IN Logic4;
        Enable : IN Logic4;
        Data : INOUT Busand(1 to 8);
        Addr : OUT Busand(1 to 8);
        Zero : OUT Logic4;
        Rd : INOUT Logic4;
        Rd1 : INOUT Logic4;
        Wr1 : INOUT Logic4;
        Wr : INOUT Logic4);
end component;

constant one_high : bit_vector(0 to 7) := "00000001";

signal shiftir, clockir, updateir : bit;
signal shiftdr, clockdr, updatedr : bit;
signal mode, inst_tdo, indent_tdo, enablee, reset: bit;
signal data_tdo, selectt, select_id, buff_tdo, bypass_tdo : bit;
signal dr_select, instruction, test_mode, test_regs : bit_vector(0 to 7);
signal int_Clock : Logic4;
signal int_Sel : Logic4;
signal int_Enable : Logic4;
signal int_Data : Busand(1 to 8);
signal int_Addr : Busand(1 to 8);
signal int_Zero : Logic4;
signal int_Rd : Logic4;
signal int_Rd1 : Logic4;
signal int_Wr1 : Logic4;
signal int_Wr : Logic4;
signal next : bit_vector(0 to 22);

begin
   test_regs(0) <= bypass_tdo;
   test_regs(1) <= indent_tdo;
   mode <= test_mode(0);
   select_id <= test_mode(1);

   tap : tap_controller port map (tms,tck,reset,selectt,enablee,shiftir,
                               clockir,updateir,shiftdr,updatedr,clockdr);
   bypass : bypss_reg port map (shiftdr,clockdr,TDI,bypass_tdo);
   instruct : reg_instruction port map(reset,clockir,updateir,shiftir,TDI,one_high,instruc
   ident : indent_reg port map(select_id, shiftdr, clockdr, tdi, indent_tdo);
   decoder : instruction_decoder port map(instruction, dr_select, test_mode, open);
   mux2 : mux2 port map(data_tdo, inst_tdo, selectt, buff_tdo);
   mux1 : mux1 port map(test_regs, dr_select, data_tdo);
   tdo_buff : tdo_buffer port map(tck, enablee, buff_tdo, tdo);
   bscan0 : bscan port map(shiftdr, clockdr, updatedr, next(0), Clock, mode, next(1), int_
   bscan1 : bscan port map(shiftdr, clockdr, updatedr, next(1), Sel, mode, next(2), int_Se
   bscan2 : bscan port map(shiftdr, clockdr, updatedr, next(2), Enable, mode, next(3), int
   G0 : for I in 0 to 6 generate
      bscan_gen0 : bscan port map(shiftdr, clockdr, updatedr, next(3 + I), Data(I+1), mode
   end generate;
   G1 : for I in 0 to 6 generate
      bscan_gen1 : bscan port map(shiftdr, clockdr, updatedr, next(10 + I), int_Addr(I+1),
   end generate;
   bscan5 : bscan port map(shiftdr, clockdr, updatedr, next(17), int_Zero, mode, next(18),
   bscan6 : bscan port map(shiftdr, clockdr, updatedr, next(18), Rd, mode, next(19), int_R
   bscan7 : bscan port map(shiftdr, clockdr, updatedr, next(19), Rd1, mode, next(20), int_
   bscan8 : bscan port map(shiftdr, clockdr, updatedr, next(20), Wr1, mode, next(21), int_
   bscan9 : bscan port map(shiftdr, clockdr, updatedr, next(21), Wr, mode, next(22), int_W

   test_regs(2) <= next(22);
end behav_jtag;
```

```
component REG_INSTRUCTION is
  generic (Setup_Shift_Time,Hold_Shift_Time,MPulse_Width_Shift,
           Setup_Update_Time,Hold_Update_Time,MPulse_Width_Update,
           Mux_Del,Stage_Shift_Del,Stage_Update_Del:time := 3 ns);

  port(Reset,
       ClockIR,
       UpdateIR,
       ShiftIR,
       TDI:in bit;
       Status:in bit_vector(0 TO 7);
       Instruction: out bit_vector(0 TO 7);
       TDO: out bit);
end component;

component mux_1 IS
       generic (
       instruction_set :bit_vector(0 to 7);
       tdo_test_data_registers_set : integer;
       mux_del :time := 3 ns);
       port (
       tdo_test_data_registers: in bit_vector(0 to 7);
       instruction : in bit_vector(0 to 7);
       tdo: out vlbit);
end component;

component mux_2 is
       generic (mux_del : time := 3 ns);
       port (
       tdo_test_data_registers,
       tdo_instruction,
       selectt : in bit;
       tdo : out bit);
end component;

component REG_BSCAN is
  generic (System_pin_type_types: String_vector;
  SUT_Capture,HT_Capture, MDEL,SUT_Update,HT_Update : time := 3 ns);
 port(
 Select_Bscan,
 Reset,
 ShiftDR,
 ClockDR,
 UdateDR,
 TDI: in Bit;
 Parallel_Input: in Bit_vector;
 Test_Mode:in Bit_vector(0 to 1);
 Parallel_Output: out Bit_vector;
 TDO: out Bit
 );
end component;

component tap_controller IS
       generic (Setup_Time, Hold_time,Min_Pulse_Width_1,
            Min_Pulse_Width_0, S_Odel, Odel: Time := 3 ns);
       PORT (
       TMS,
       TCK: in bit;
       Reset,
       Selectt,
       Enable,
       ShiftIR,
       ClockIR,
       UpdateIR,
       ShiftDR,
       UpdateDR,
       ClockDR : out bit
   );
end component;
```

```
entity cpu_jtag is
   port(TCK, TMS, TDI, TDO : in bit;
        Clock : IN Logic4;
        Sel : IN Logic4;
        Enable : IN Logic4;
        Data : INOUT Busand(1 to 8);
        Addr : OUT Busand(1 to 8);
        Zero : OUT Logic4;
        Rd : INOUT Logic4;
        Rd1 : INOUT Logic4;
        Wr1 : INOUT Logic4;
        Wr : INOUT Logic4);
end cpu_jtag;

architecture behav_jtag of cpu_jtag is

component BSCAN is
        generic (Setup_Time,
        Hold_Time,
        Mux_Del,
        Dtype_Del :Time := 3 ns);
        Port (ShiftDR, ClockDR, UpdateDR,
              Last_Cell, S_In, Mode:in bit;
              Next_Cell: buffer bit;
              S_Out: out bit);

end component;

component TDO_Buffer is
 generic (setup_time, hold_time, Min_pulse_width, Del,  Tdo_del:Time := 3 ns);
 port(
 TCK,
 Enablee,
 Input:in Bit;
 TDO: out Tristate
 );
end component;

component BYPASS_REG is
        generic (Setup_Time,
        Hold_Time,
        MPulse_width,
        AND_Del,
        Bypass_Del :Time := 3 ns);
        Port (ShiftDR, ClockDR,
              TDI: in bit;
              TDO: out bit);

end component;

component INSTRUCTION_DECODER is
       generic (Instruction_set:bit_vector;
                DR_select_Set,Test_Mode_Set,Additional_Signals_Set
                :bit_vector,
                Open_Check:Boolean;
                Dec_Del:time := 3 ns);
       port ( Instruction:in bit_vector(0 to 7);
              DR_Select: out bit_vector(0 to 7);
              Test_Mode: out bit_vector(0 to 7);
              Additional_Signals : out bit_vector(0 to 7));
end component;

component IDENT_REG is
       generic (setup_time,Hold_Time,MPulse_Width,Mux_Del:Time := 3 ns);
       port(
       Select_Ident,
       ShiftDR,
       ClockDR,
       TDI:in Bit_vector;
       TDO: out Bit
       );
end component;
```

```
-- ibrary vdeg_portable; --
use vdeg_portable.types.all;
entity cpu is
port (
       clock,sel : in logic4;
       enable : in logic4;
       data : inout BusAnd(1 to 8);
       addr : inout BusAnd(1 to 8);
       zero : out logic4;
       rd : inout logic4;
       rd1 : inout logic4;
       wr1 : inout logic4;
       wr : inout logic4
      );

end cpu;
```

| # | PIN NAME | PIN MODE | PIN TYPE | BUS WIDTH | STARTING |
|---|---|---|---|---|---|
| 0 | Clock | IN | Logic4 | 1 | 0 |
| 1 | Sel | IN | Logic4 | 1 | 0 |
| 2 | Enable | IN | Logic4 | 1 | 0 |
| 3 | Data | INOUT | Busand | 7 | 1 |
| 4 | Addr | OUT | Busand | 7 | 1 |
| 5 | Zero | OUT | Logic4 | 1 | 0 |
| 6 | Rd | INOUT | Logic4 | 1 | 0 |
| 7 | Rd1 | INOUT | Logic4 | 1 | 0 |
| 8 | Wr1 | INOUT | Logic4 | 1 | 0 |
| 9 | Wr | INOUT | Logic4 | 1 | 0 |

```
entity cpu_jtag is
   port(TCK, TMS, TDI, TDO : in bit;
        Clock : IN Logic4;
        Sel : IN Logic4;
        Enable : IN Logic4;
        Data : INOUT Busand(1 to 8);
        Addr : OUT Busand(1 to 8);
        Zero : OUT Logic4;
        Rd : INOUT Logic4;
        Rd1 : INOUT Logic4;
        Wr1 : INOUT Logic4;
        Wr : INOUT Logic4);
end cpu_jtag;

architecture behav_jtag of cpu_jtag is

component BSCAN is
        generic (Setup_Time,
        Hold_Time,
        Mux_Del,
        Dtype_Del :Time := 3 ns);
        Port (ShiftDR, ClockDR, UpdateDR,
              Last_Cell, S_In, Mode:in bit;
              Next_Cell: buffer bit;
              S_Out: out bit);

end component;

component TDO_Buffer is
 generic (setup_time, hold_time, Min_pulse_width, Del,  Tdo_del:Time := 3 ns);
 port(
 TCK,
 Enablee,
 Input:in Bit;
 TDO: out Tristate
 );
end component;

component BYPASS_REG is
        generic (Setup_Time,
        Hold_Time,
        MPulse_width,
        AND_Del,
        Bypass_Del :Time := 3 ns);
        Port (ShiftDR, ClockDR,
              TDI: in bit;
              TDO: out bit);

end component;

component INSTRUCTION_DECODER is
       generic (Instruction_set:bit_vector;
                DR_select_Set,Test_Mode_Set,Additional_Signals_Set
                :bit_vector,
                Open_Check:Boolean;
                Dec_Del:time := 3 ns);
       port ( Instruction:in bit_vector(0 to 7);
              DR_Select: out bit_vector(0 to 7);
              Test_Mode: out bit_vector(0 to 7);
              Additional_Signals : out bit_vector(0 to 7));
end component;

component IDENT_REG is
       generic (setup_time,Hold_Time,MPulse_Width,Mux_Del:Time := 3 ns);
       port(
       Select_Ident,
       ShiftDR,
       ClockDR,
       TDI:in Bit_vector;
       TDO: out Bit
       );
end component;
```

```
component cpu is
port (
        Clock : IN Logic4;
        Sel : IN Logic4;
        Enable : IN Logic4;
        Data : INOUT Busand(1 to 8);
        Addr : OUT Busand(1 to 8);
        Zero : OUT Logic4;
        Rd : INOUT Logic4;
        Rd1 : INOUT Logic4;
        Wr1 : INOUT Logic4;
        Wr : INOUT Logic4);
end component;

constant one_high : bit_vector(0 to 7) := "00000001";

signal shiftir, clockir, updateir : bit;
signal shiftdr, clockdr, updatedr : bit;
signal mode, inst_tdo, indent_tdo, enablee, reset: bit;
signal data_tdo, selectt, select_id, buff_tdo, bypass_tdo : bit;
signal dr_select, instruction, test_mode, test_regs : bit_vector(0 to 7);
signal int_Clock : Logic4;
signal int_Sel : Logic4;
signal int_Enable : Logic4;
signal int_Data : Busand(1 to 8);
signal int_Addr : Busand(1 to 8);
signal int_Zero : Logic4;
signal int_Rd : Logic4;
signal int_Rd1 : Logic4;
signal int_Wr1 : Logic4;
signal int_Wr : Logic4;
signal next : bit_vector(0 to 22);

begin
   test_regs(0) <= bypass_tdo;
   test_regs(1) <= indent_tdo;
   mode <= test_mode(0);
   select_id <= test_mode(1);

   tap : tap_controller port map (tms,tck,reset,selectt,enablee,shiftir,
                                  clockir,updateir,shiftdr,updatedr,clockdr);
   bypass : bypss_reg port map (shiftdr,clockdr,TDI,bypass_tdo);
   instruct : reg_instruction port map(reset,clockir,updateir,shiftir,TDI,one_high,instruc
   ident : indent_reg port map(select_id, shiftdr, clockdr, tdi, indent_tdo);
   decoder : instruction_decoder port map(instruction, dr_select, test_mode, open);
   mux2 : mux2 port map(data_tdo, inst_tdo, selectt, buff_tdo);
   mux1 : mux1 port map(test_regs, dr_select, data_tdo);
   tdo_buff : tdo_buffer port map(tck, enablee, buff_tdo, tdo);
   bscan0 : bscan port map(shiftdr, clockdr, updatedr, next(0), Clock, mode, next(1), int_
   bscan1 : bscan port map(shiftdr, clockdr, updatedr, next(1), Sel, mode, next(2), int_Se
   bscan2 : bscan port map(shiftdr, clockdr, updatedr, next(2), Enable, mode, next(3), int
   G0 : for I in 0 to 6 generate
      bscan_gen0 : bscan port map(shiftdr, clockdr, updatedr, next(3 + I), Data(I+1), mode
   end generate;
   G1 : for I in 0 to 6 generate
      bscan_gen1 : bscan port map(shiftdr, clockdr, updatedr, next(10 + I), int_Addr(I+1),
   end generate;
   bscan5 : bscan port map(shiftdr, clockdr, updatedr, next(17), int_Zero, mode, next(18),
   bscan6 : bscan port map(shiftdr, clockdr, updatedr, next(18), Rd, mode, next(19), int_R
   bscan7 : bscan port map(shiftdr, clockdr, updatedr, next(19), Rd1, mode, next(20), int_
   bscan8 : bscan port map(shiftdr, clockdr, updatedr, next(20), Wr1, mode, next(21), int_
   bscan9 : bscan port map(shiftdr, clockdr, updatedr, next(21), Wr, mode, next(22), int_W

   test_regs(2) <= next(22);
end behav_jtag;
```

**APPENDIX F - JTAG.COMP FILE**

**This file contains all components of the JTAG Architecture**

```
component bscan
        generic (Setup_Time,
        Hold_Time,
        Mux_Del,
        Dtype_Del :Time := 3 ns);
        Port (ShiftDR, ClockDR, UpdateDR,
              Last_Cell, S_In, Mode:in bit;
              Next_Cell: inout bit;
              S_Out: out bit);

end component;

component tdo_buffer
 generic (setup_time, hold_time, Min_pulse_width, Del,
Tdo_del:Time := 3 ns);
 port(
 TCK,
 Enablee,
 Input:in Bit;
 TDO: out Logic4
 );
end component;

component bypass_reg
        generic (Setup_Time,
        Hold_Time,
        MPulse_width,
        AND_Del,
        Bypass_Del :Time := 3 ns);
        Port (ShiftDR, ClockDR,
              TDI: in bit;
              TDO: out bit);

end component;

component inst_decode
       generic (Instruction_set:bit_vector := "00000000";

DR_select_Set,Test_Mode_Set,Additional_Signals_Set
               :bit_vector := "00000000";
                Open_Check:Boolean := false;
                Dec_Del:time := 3 ns);
       port ( Instruction:in bit_vector(0 to 7);
              DR_Select: out bit_vector(0 to 7);
              Test_Mode: out bit_vector(0 to 7);
              Additional_Signals : out bit_vector(0 to
7));
end component;
```

```
component ident_reg
       generic
(setup_time,Hold_Time,MPulse_Width,Mux_Del:Time := 3 ns;
                 ID_code : bit_vector :=
"0000000000000001");
       port(
       Select_Ident,
       ShiftDR,
       ClockDR,
       TDI:in Bit;
       TDO: out Bit
       );
end component;

component reg_inst
   generic
(Setup_Shift_Time,Hold_Shift_Time,MPulse_Width_Shift,

Setup_Update_Time,Hold_Update_Time,MPulse_Width_Update,
            Mux_Del,Stage_Shift_Del,Stage_Update_Del:time
:= 3 ns);

   port(Reset,
        ClockIR,
        UpdateIR,
        ShiftIR,
        TDI:in bit;
        Status:in bit_vector(0 TO 7);
        Instruction: out bit_vector(0 TO 7);
        TDO: out bit);
end component;

component mux_1
        generic (
        instruction_set :bit_vector(0 to 7) :=
"00000000";
        tdo_test_data_registers_set : integer_vector :=
(0,0,0,0,0,0,0,0);
        mux_del :time := 3 ns);
        port (
        tdo_test_data_registers: in bit_vector(0 to 7);
        instruction : in bit_vector(0 to 7);
        tdo: out bit);
end component;

component mux_2
        generic (mux_del : time := 3 ns);
        port (
        tdo_test_data_registers,
        tdo_instruction,
        selectt : in bit;
        tdo : out bit);
end component;
```

```
component tap_c
        generic (Setup_Time, Hold_time,Min_Pulse_Width_1,
             Min_Pulse_Width_0, S_Odel, Odel: Time := 3
ns);
        PORT (
        TMS,
        TCK: in bit;
        Reset,
        Selectt,
        Enable,
        ShiftIR,
        ClockIR,
        UpdateIR,
        ShiftDR,
        UpdateDR,
        ClockDR : out bit
    );
end component;
```

**APPENDIX F - JTAG.INS FILE**

This file instantiates the JTAG Components

```
   test_regs(0) <= bypass_tdo;
   test_regs(1) <= indent_tdo;
   mode <= test_mode(0);
   select_id <= test_mode(1);
   nextt(0) <= TDI;

   tap : tap_c port map
(tms,tck,reset,selectt,enablee,shiftir,

clockir,updateir,shiftdr,updatedr,clockdr);
   bypass : bypass_reg port map
(shiftdr,clockdr,TDI,bypass_tdo);
   instruct : reg_inst port
map(reset,clockir,updateir,shiftir,TDI,temp,instruction,i
nst_tdo);
   ident : ident_reg port map(select_id, shiftdr,
clockdr, tdi, indent_tdo);
   decoder : inst_decode port map(instruction, dr_select,
test_mode, open);
   mux2 : mux_2 port map(data_tdo, inst_tdo, selectt,
buff_tdo);
   mux1 : mux_1 port map(test_regs, dr_select, data_tdo);
   tdo_buff : tdo_buffer port map(tck, enablee, buff_tdo,
tdo);
```

**APPENDIX F - JTAG.SIG FILE**

This file contains the standard JTAG signals

```
constant one_high : bit_vector(0 to 7) := "00000001";

signal temp : bit_vector(0 to 7) := one_high;
signal shiftir, clockir, updateir : bit;
signal shiftdr, clockdr, updatedr : bit;
signal mode, inst_tdo, indent_tdo, enablee, reset: bit;
signal data_tdo, selectt, select_id, buff_tdo, bypass_tdo
: bit;
signal dr_select, instruction, test_mode, test_regs :
bit_vector(0 to 7);
```

**APPENDIX F - JTAG.USE FILE**

This file encloses the required VHDL Libraries

```
library vdeg_portable;
use vdeg_portable.types.all;
use work.declar.all;
```

## PAPERS PUBLISHED

MSC 92, 23 Annual Modelling and Simulation Conference. Pittsburgh, USA. 30 April - 1 May 1992. Sponsored by IEEE. Paper Title: **'Automatic Generation and Insertion of Boundary Scan Architecture'.**

EMS 92, European Modelling and Simulation Conference 1992, York, UK. 1-3 June 1992. Sponsored by SCS. Paper Title: **'Modelling and Automatic Generation of Boundary Scan Architecture'.**

**Poster Session Paper:**

7th UK EDA Workshop, Newcastle Upon Tyne, UK. 6-9 December 1992. Paper Title: **'Automatic Insertion of Boundary Scan Architecture'.**