Using hierarchy in macro cell test assembly

Jens Leenstra and Lambert Spaanenburg*

Institute for Microelectronics Stuttgart, Allmandring 30a, 7000 Stuttgart 80 (B.R.D.)

Abstract
Test generation & assembly is investigated for hierarchical VLSI designs of modules with testable macro cells and an associated local controller. To create a testable (a)synchronous controller structure, the use of state cells is proposed. It is shown that state cell controllers can be tested by reconfiguration into a token scanpath and permits, for reason of the direct correspondence between implementation and function, accurate functional fault modelling and test pattern generation. Moreover such controllers ease the assembly of the module test from the macro tests.

1. Introduction
In recent years, design for testability has received increasing attention. To address the testing problems, a variety of techniques known as design for testability have been proposed. Structured techniques such as scanpath [1] assume, that test generation of combinational networks is possible. Therefore in test mode, all memory elements are lined up into one or more shift registers through which the test vectors are shifted in and out.

Macro testing [2] has been proposed to solve a number of problems with scan design, which became apparent during the growth towards VLSI. Various types of logic structures [3], such as ROMs, RAMs, PLAs, multipliers, and ALUs are often better tested through the use of dedicated fault modeling and test procedures. Macro testing partitions the design into test-related macro's, providing the ability to adapt a test method to each type of logic.

Constructing VLSI systems in a structured and modular fashion by the use of functional building blocks or modules can be very attractive for design and test. Modules in our approach, comprised of a set of datapath elements of medium sized complexity (macro's) together with a local controller, which will sequence the operations of the datapath elements.

(*) Part of this research was performed, while the authors were at Twente University (The Netherlands).
design process, for example due to the addition of test structures. The use of modules, assembled as outlined above, can reduce the amount of time-consuming redesign significantly, since design changes inside a module only require the modification of the local controller and do not have impact on other blocks.

![Diagram of structural hierarchy]

The structural hierarchy of a module-based system is illustrated in figure 1. In the proposed design methodology a system is described as a couple of modules and a system controller, scheduling the module operations. Test structures are added, if needed, to increase the system testability by gaining access to the modules I/O's. On the next level the components of a module are shown. A module consists of datapath macro's, a local controller and test structures.

To create a testable system, at least the basic components, of which the system is built, need to be testable. The main difficulty in designing testable macro cells is caused by the fact, that there are many different structures for these kinds of cell with very different test characteristics. Almost for every logic structure there is an optimal, yet different design-for-test technique. Therefore the knowledge to create testable controller and datapath macro's will be hard-coded in the macro cell generator or later on added through an expert (system).

After test vectors have been generated, done for each individual macro by a macro related test algorithm, the test vectors have to be merged into a system test program. A straightforward way to merge the test vectors is based on partitioning during system test into a set of independently testable, pre-characterized macro's. The Macro Testing approach creates this division by adding test interface elements at the outputs of each macro. The chip test vector set is then a simple combination of macro test vectors.

However, when the chip is constructed with many small macro blocks (as in our case), the number of Test Interface Elements can be rather high, resulting in severe delay and area penalties. Therefore other ways to propagate the test vectors to and from the macro to the primary input/outputs have to be found.

Another approach to merge the test patterns of macro's is described in [8]. By symbolic execution of a high-level description of the system, a path tree is constructed describing the set of functions, that the (sequential) system is able to perform. To test a macro of the system, first a path in the path tree has to be found, corresponding to the travel of the system through a number of states, in which data will propagate from the primary inputs to the macro inputs; next the macro is activated to perform the function and finally the macro response is propagated to the primary outputs. A primary disadvantage of such an approach is that, when the number of states is large (resulting in a large path tree) and when the tests demand long propagating sequences, it may require a considerable amount of time to find the appropriate path in the path tree (if such a path exists).

Basically there are two possibilities to reduce the complexity of the test assembly (generation) algorithm:
1. Reduce the number of states to deal with.
2. Shorten the path length of the propagating sequences.

In the module based design approach, a reduction of the number of states the test assembly algorithm has to deal with can be obtained by assembling the macro test vectors according to the structural hierarchy of the system (shown in figure 1). Using this hierarchy, first a module test program will be constructed from macro test vectors based on the functional description of the module: a path tree is constructed for each module, instead for the complete system, thereby reducing the size of the path tree which has to be searched. In a following step the system test can be constructed by merging the module test programs.

To shorten the length of the input sequence, use can be made of test structures like testpoints, scan latches and so on. In the macro testing approach mentioned above, the test sequence is reduced to one by adding test interface elements to all macro outputs. Simply adding interface elements to all outputs without first ascertaining the difficulty of the problem of assembling the macro tests into a module test program could unduly raise area and performance costs. The approach described in the next section will improve the macro test assembly in two ways:
- An efficient inherently testable controller implementation technique will be shown, which can be very easily set into a specific state by the use of a token scanpath. The direct setting of the state can reduce the test length significantly, since thus it is possible to start the test propagation from every point in the path tree instead of always starting from the root.
- A design representation is presented, which permits the composition of a module test program from the macro test specifications without constructing a path tree.
3. Gaining access to the datapath macro's

3.1. Functionality of the module

The outline of a module is shown in figure 2. Externally the module will be described by a functional, a timing and a test specification. Inside the module a distinction is made between the macro's implementing the local controller and the datapath macro's. Control and datapath macro's are handled differently, since the controller always plays a crucial role inside a system. The controller determines the operations performed by the datapath blocks by steering the (de)multiplexers in the datapath and selecting the operation of the general datapath macro's like ALU's.

Therefore when the module has to perform an operation the associated start event will activate the controller, who (a) decodes the opcode, to determine which operations have to be done, and (b) monitors and controls the datapath such that the datapath macro's perform the requested operations. When all requested operations have been performed, the control part will signal completion by sending a "ready" signal.

The composition of a module is to some extent comparable to the composition of a procedure in software. Externally the procedure is described by the header, which corresponds to the external test, functional and timing views. The program control is implemented by programming statements like "while", "if", which will sequence the datapath manipulations done by operators and assignments. This provides an easy interface to hardware specification through specialized or general description languages, such as ELLA or C++.

The crucial role of the controller has an important impact on the assembling of macro test vectors. By first testing the controller, it can be used later on to control the propagation of test data to and from the datapath macro's. This can be done by (a) setting the controller into a particular state and (b) letting it travel through a number of states, independent of the outcome of the datapath operations. The latter condition is needed to interrupt the data dependent operation of the controller and can be realized by setting the input signals of the controller to a fixed value. For the propagation of the test data to/from the datapath macro from/to the primary inputs/outputs the test data propagation path has to remain the same, independent of the actual value of the test data. So to implement this test scheme an efficient controller implementation structure is needed, which is easy testable and of which the state of the controller can be easily set.

3.2. Testable Controller design

Traditionally, controller circuits are designed using the Huffman Finite State Machine model. This model can be used for the implementation of synchronous and asynchronous FSM's. A PLA implementation based on this circuit model is shown in figure 3.

![Fig. 3. Module with a PLA based FSM controller](image)

For a synchronous local controller embedded within a module, the scan approach could be used to test the FSM. To this purpose all output registers are series-chained during test, thus providing serial access. Test interface elements are added to control the controller inputs which are connected to the datapath outputs. Later on the same scanpath can be used to steer the state of the multiplexers in the datapath through which test data can flow to/from the datapath macro-under-test. However each time the data path has to perform another operation, in order to propagate the test data, an update of the scanpath will be required.

A way to circumvent this update, as discussed in the foregoing section, is the use of the functionality of the controller. After setting the initial state of the controller and the controller inputs, the controller will operate in normal operation (with fixed inputs) in order to propagate the datapath test data. This mode of propagation will be interrupted, when the inputs of the controller need to be changed or when the test result is available at a datapath output.

The main drawback of using the scanpath approach to test small controller macro's is the relatively high overhead. Improvement can be reached by judiciously adding logic to the PLA. However the overhead of these techniques can still be enormous when used for small controller structures[9]. Another disadvantage is, that to set the new initial state of the controller (to start the test propagation from that state) it requires the use of the scanpath. For a
single small controller this is not a problem. However when all macro scanpaths are lined up into a single system wide controller scanpath, the time needed to set a new controller state may be rather long, which can cause to change the state of the datapath when dynamic memory elements are in use.

3.3. State diagram implementation

State diagrams are a popular means to specify control. A technique for the direct mapping of asynchronous and synchronous state diagrams onto silicon is based on one-hot encoding[10]. This technique will be explained using the very simple state diagram shown in figure 4. Initially the

![State Diagram](image)

Fig. 4. Example (a) of state diagram and its realization (b) 'wait' state output is active ('1') and the other two state outputs 'exec' and 'ready' will be inactive ('0'). When the start event occurs (start='1'), the 'exec' state will become active thereby resetting the 'wait' state. Next the 'ok' input line will become '1' and the token will shift from the 'exec' state to the 'ready' state, so the ready output will become '1' and so on. Every state can be directly replaced by a hardware equivalent called state cell and the arcs in the state diagram are implemented by arc cells (figure 4b). A special power-on cell is needed to implement states, which have to be active after a general reset. In figure 5 a CMOS realization of a state and an arc cell are shown.

![Arc and State Cell](image)

Fig. 5. Partial implementation of a state diagram through state/arc-cells.

This scheme offers several advantages compared to the PLA realization:
- Basically a state cell controller is nothing else than a special type of shift register, in which the memory elements (state cells) are separated by arc cells. The active state cell (i.e. output='1') can be seen as to carry a token. The token shift between memory elements is thereby controlled by the arc cell condition input (cond). Therefore such a controller can easy be reconfigured into a token scanpath by switching in test mode the arc cells into "through connection". In the next sections it will be shown that through this token scanpath the controller can be tested and easily set into a particular state, thereby circumventing the need to add scan registers;
- The scanpath approach was only appropriate for synchronous controllers; the state cell scheme implements asynchronously operating controllers without input and delay restrictions such as fundamental mode operation and therefore leads to testable realizations of asynchronous control structures (one may alternatively view the circuit as just an asynchronous register);
- Circuitry can be designed with greater ease and leads to automated, yet efficient realizations through regular layout structures [10];
- Modifications in the controller specification do not require a new state assignment and the consequent total redesign of the circuit;
- Correspondence between implementation and functional description provides a way to describe faulty behaviour based on the structural faults in state and arc cells, which makes accurate functional test pattern generation possible.

3.4. Token scanpath

Testing the datapath cells requires, that the controller can be set in a specific state. Two alternatives to bring the controller in a particular state are shown in figure 6 and 7.

![Controller Diagram](image)

Fig. 6. External reconfiguration to set the state machine

In figure 6 the state of the controller is set by the use of the controller inputs and the clock lines ("tstclk1" and "tstclk2" in figure 5). First the inputs controlling the arc cells are set into the required state. Next clock pulses will shift the token among the state cells. If the arc cell inputs need to be changed, these two steps will be repeated till the required state is reached. Otherwise only the last step has to be repeated.
In figure 7 the state control setting is realized through the introduction of an extra test mode, which affects the arc cell circuitry [10]. If the signal "test" is low, it behaves as an ordinary arc cell. However if the signal test is low, the arc cell degenerates to a mere through-connection. Another type of arc cell will be needed, when more than one state cell can be activated. Then one of the arc cell connections between state cells has to be blocked by adding a "test" transistor in series with the "cond" pass transistor and an alternative path to the blocked state has to be created when this state is not included in the token scan path through another arc cell connection. The advantage of this scheme is, that it eliminates the need to add scan latches. So for relative simple controllers as usually appear within modules, the use a direct implementation of the state diagram may increase the area consumption of the pure controller implementation; however it definitely reduces the overall area by eliminating scan registers at the inputs and outputs.

The same scheme for forcing the controller into a particular state can be used for an asynchronous state cell controller. In this case the asynchronous operation will be interrupted for synchronizing the token flow by means of explicit scan clocks. So for testability reasons, also asynchronous controllers will be partly synchronously implemented.

3.5. Functional fault model

Test generation can be based on a geometrical, a structural, a behavioural description or a combination of these. In all these approaches the problem arises: which fault or error hypothesis has to be used? Using a structural type description, either the functional primitives themselves are not tested or one has to describe all the faulty functioning of these primitives. Using only a behavioral description the accuracy of the error modeling is often doubtful.

In the presented controller implementation technique, there is however a direct relation between the state diagram and the way this diagram is implemented. Therefore once the layout of the state cells is known, a fault model can be determined based on the circuit and layout properties to model the effect of these faults in the state diagram.

Application of the single stuck-at fault model to the structural description of the state/arc cells has led to the following fault phenomena in relation to the state diagram description:
- state outputs stuck-at-one, stuck-at-zero;
- unconditional token shift, when "cond" input of arc cells are stuck-at-one/zero;
- no next state setting, when the token transfer condition becomes true.
- no state cell reset from the foregoing cell.

These faults have to be detected from either externally applied or on-chip produced test patterns.

3.6. Controller test

To detect the above described faults, the controller test is performed in 2 steps.

Step 1. "state outputs are stuck-at 0/1", "not resetting the foregoing state cell" and "does not set following state" fault detection.

To detect this class of faults, the output of the power-on state cell will be made easy observable. This can be done in several ways. For example, this output can be observed at the output pins by connecting it to a scanpath.

Test Procedure:
- Reset the controller and verify the effect of the reset request through the observable output;
- Activate the test mode if the internal reconfiguration scheme of figure 7 is in use. Otherwise set the values of the controller inputs to enable a token shift as in figure 6;
- Token shift. The token will be shifted by activating the controller clock and after each shift command the value of the selected output will be observed. If the external activation scheme is in use the values of the controller inputs will be rendered, if needed, to enable the token shift;

If one of the outputs of the state-cell is stuck-at-one or stuck-at-zero, the output value of the observable output will not correspond with the expected response. A stuck-at-one fault will increase the number of tokens in the token scanpath, as can be seen in the circuit diagram in figure 5. A stuck-at-zero fault will halt the token passing. When the token passing will not reset the foregoing cell, also the number of tokens will be increased.

So these types of faults can be detected by just shifting the token and observing a single output of the state cell controller. If the external reconfiguration scheme is in use, all faults concerning the set and reset of other state cells can thus be detected. In the internal reconfiguration scheme, a number of arc cells are blocked in the test mode. These faults will be detected by first setting the state of the con-
troller; then the shift of the token in the normal mode is controlled by setting the corresponding input of the arc to the requested value. Next the controller is switched to the test mode to observe the response.

Step 2. "unconditional token shift" fault.

Here we have to detect, whether a token is shifted to the next state while the condition of the arc cell was false. This can be done easily by propagating the token from the faulty state to the observable output. If the fault exists, the observed output will be '1', otherwise it will be '0'.

4. Testing a module

4.1. Test view of datapath macro's

The test properties of every datapath cell are described in the test view. A distinction is made between (a) control lines, determining the operation to be performed, (b) clock lines & module start selection and (c) data lines.

The test view of this model will contain:
- a list of control line values such that the data on the input lines is reproduced without logic transformation at the outputs (l path[3]);
- a list of control line values that provide a 1-to-1 mapping between data inputs and data outputs (F-path [11]);
- a test program that describes the input patterns together with the expected output response.

The I-paths and F-paths constitute connections, that can be used to propagate test vectors from input ports to the datapath cell inputs and to propagate the test response to an output port.

The module test can now be assembled from the test vectors for the local controller and the test set for the datapath cells, as described above. A design representation is needed to merge the tests for the datapath cells in an efficient way. This requires an explicit distinction between control and operation, as provided in the module hardware description in C.

4.2. Design representation of the module

The representation of a module design will be illustrated in a discussion about the circuit shown in figure 8. Functional notation is given in the programming language C. For every time slot (cs1 and cs2), it is exactly specified which operations have to performed by the datapath cells.

The datapath is modeled as a digraph, where the nodes represent the datapath cells and the arcs are used to model the connections between them. These nodes may have different degrees of complexity. For example a node may represent datapath cells such as a register, ALU or ROM. Every connection has a label to describe the control states using this connections.

The control part, which has to produce a sequence of control signals to determine the operations performed by the overall data part, is modeled by the state diagram. For every state the corresponding datapath cell action is specified. Basically this representation has a large degree of correspondence with the timed petri net modeling used in some silicon compiler systems [7]. Especially in case of asynchronous state diagrams, it is almost identical. Since petri net models are mathematically well-defined, a number of properties can be checked such as "liveliness" and "dead-lock", which can help the system designer to formally verify the system.

At this stage only the controller test program is known, as discussed in section 3.6. The design representation is used to find the test sets for the individual datapath cells. These can be merged to assemble a complete module test program description.
4.3. Datapath cell test procedure

To test the datapath cells, all inputs to the datapath cells need to be controllable and the test response must be propagated from the datapath cell to the module outputs, where it can be observed.

A distinction is made between datapath inputs and control inputs. Datapath inputs are controllable from the module boundary. Control inputs are steered by the local controller. To change the state of the control lines, the token will be shifted to the corresponding state. In the presented example it is assumed, that the control outputs of the datapath are made observable by the addition of test interface elements (see figure 6).

Test procedure:
1. Set the controller into the required state by the use of the controller clock, which shifts the token into the required position;
2. Feed the test vector to the module data inputs;
3. Propagate the test vector to the datapath cell under test. This will be performed under normal operation of module. The test vectors will be propagated to the datapath cell by using the I- and/or F-path's;
4. Perform the test (normal operation, one clock cycle);
5. Propagate the test response to an observable module output. This is done under normal operation of module (with fixed controller inputs). The test vectors will propagate to an observable output by using I and/or F path's.

Step 1 to 5 are repeated till all test vectors specified in the datapath cell view have been fed to the datapath input.

4.4. Datapath test composition

The datapath test vectors have to be propagated from the datapath inputs to the datapath cells, which have to be tested, and thereafter the response has to be propagated to the module outputs. To propagate the fault effects, I- and F-path's can be used.

When the controller travels through a distinct number of states, the function can be expressed by an iterative array structure. This transforms the sequential circuit into a combinational iterative array[12] as shown in figure 9.

This transformation can help to find I- and F-path's from and to the datapath cell under test. Basically the I- and F-path models the D-drive in the usual D-algorithm. Therefore the classical approach to test (path sensitization) is applicable to some extent. However where the normal D-algorithm will always find a path when the circuit is not redundant, here the algorithm may fail when no I- or F-path is specified for the datapath cell functions.

With reference to the classical approach, two similar steps have to be performed: consistency check and forward propagation. But before starting the algorithm, the initial state of the controller has to be determined. Because the controller can be directly forced into the required state, the determination of the homing sequence provides no problems.

Initial state selection: select a controller state, where the datapath cell under test is updated. The set of states, from which can be chosen, is represented by the labels on the input arcs of the datapath cell in the corresponding representation (figure 9), since these labels represent the states wherein the datapath cell is used or updated.

Backward propagation: Determine the combinational circuit corresponding to this state. Determine if an I-path or F-path can be found from primary inputs to the datapath cell inputs.

If the above described step fails, try to find vector propagating path's using another initial state. When all these attempts fail, an iterative array is constructed. Add a copy of the datapath cell, steered by a state from which a token can flow directly to a potential initial state. Next the backward propagation step is performed for the newly constructed circuit and so on.

Only when all the above attempts fail, hardware has to be added (such as multiplexers or scan registers) to ease the propagation of a test vector to the particular datapath cell.

Forward propagation: The propagation of the test response to an observable output starts with the datapath circuit corresponding to the initial state selected above. Next fault propagating path's from the datapath cell outputs to the module outputs are determined. If this attempt fails, an iterative array is constructed by adding a copy of the next state circuit and so on. When no fault propagation path's
can be found, hardware will be added to make the test response observable.

The addition of extra test hardware may also alter the controller. The introduction of a test mode signal can ease the propagation by steering extra multiplexers used during test. This is often more economical than the addition of scan registers. For reason of the flexible controller implementation using state cells, this modification is very easy to execute.

5. Discussion & conclusions

The module concept as outlined above has been evaluated over the years from its first application in microprocessor design [13] to real-time ASIC design for telecommunication purposes. The latest example is the LSID904: a 50k transistor single-chip implementation of an IBM-327X/325X protocol handler (figure 10), consisting of:
- the physical interfaces MMU & CMU to respectively the local memory and the IBM network.
- the 2-level programmable program sequencer PCS, that translates IBM-327X/325X instructions and evaluates priority conditions;
- the actual protocol interpretation hardware ISU, DMU & IBU.

The total comprises of 16 modules. Although some variety exists on the architectural level, the format remains as depicted in figures 1 and 2.

By a strict standardization of the way, in which datapath and controlpath macro's are connected, a degree of universality in test can be achieved. Therefore, attempts are presently under way to construct a separate test processor for generating the universal tests, required for the modules. Because of the large degree of similarity it is expected, that such a test processor can be achieved at a negligible overhead.

Acknowledgments

The research reported here is presently funded from the EC program PROMETHEUS, subprogram PRO-CHIP, as part of project 23132 "Parallel Architectures for Computer Vision".

References