Full scan testing of handshake circuits

Frank J. te Beest

2003

Ph.D. thesis
University of Twente

Twente University **Press**

Also available in print:
http://www.tup.utwente.nl/

Full scan testing of handshake circuits

This research was supported by the Technology Foundation STW, applied science division of NWO and the technology programme of the Ministry of Economic Affairs in the Netherlands.

Twente University **Press**

Publisher: Twente University Press,
P.O. Box 217, 7500 AE Enschede, the Netherlands,
www.tup.utwente.nl

Cover design: Jo Molenaar, [deel4 ontwerpers], Enschede
Print: Océ Facility Services, Enschede

ISBN 9036519098

FULL SCAN TESTING OF HANDSHAKE CIRCUITS

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof.dr. F.A. van Vught,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op woensdag 21 mei 2003 om 15.00 uur

door

Frank Johan te Beest
geboren op 19 juni 1973
te Dinxperlo

Dit proefschrift is goedgekeurd door de promotoren
prof.dr.ir. T. Krol
prof.dr.ir. C.H. van Berkel

en de assistent-promotor
dr.ir. H.G. Kerkhoff

Contents

Chapter 1

Introduction

The drive for more power-efficient circuits has been one of the main reasons to search for alternatives for the conventional synchronous circuit design style. One of the main obstacles to reduce the power consumption of synchronous circuits is the global clock that generates continuous activity in the circuit, even when (part of) the circuit is not doing any useful work. In an asynchronous circuit, this global clock is replaced by a local synchronization mechanism that is only active where and when it is required.

Handshake circuits are a sub-class of asynchronous circuits. They follow a set of design rules that make them easier to design and verify than general asynchronous circuits. The introduction of high-level synthesis tools for handshake circuits has led to a rapid increase in size and complexity of these circuits during the last ten years. The handshake circuit technology matured sufficiently to be implemented in various commercial products, like a family of pager chips and a family of smartcards [32, 33]. These products use the handshake technology to exploit one or more of the benefits offered by handshake circuits. Some of the most important benefits are [10]:

- Low power consumption
- Low electro-magnetic emission
- Robustness, with regard to power-supply voltage, process and temperature fluctuations

Although the technology has led to several commercial products, up to now the widespread industrial uptake of handshake circuits has been minimal. This is partly caused by the unfamiliarity of the technology. The other main reason has been the lack of a mature design flow covering all the steps from specification to layout. Despite the existence of a powerful synthesis tool [11], one critical step in the design flow has so far been missing. This step is an easy-to-use test method that can achieve test quality equal to conventional synchronous test quality and that offers an automated test flow.

This thesis describes the development and implementation of such a complete and automated test flow for handshake circuits. The work is primarily targeted at handshake circuits designed using the Tangram tools that were developed at the Philips Research Laboratories [11]. The Tangram tools were developed in a project starting in the late 1980's and the initial focus was VLSI programming. This is the usage of a high-level programming language to design digital VLSI circuits. The research gradually evolved into the most powerful toolkit currently available for the design of handshake circuits.

1.1 Handshake circuits

Handshake circuits are a member of the larger family of asynchronous circuits. In handshake circuits, the clock that normally synchronizes activity in a circuit is replaced by handshake channels. Two components in the circuit that need to synchronize, for example to transfer data, do this by signaling over a dedicated handshake channel. Handshake circuits consist of a number of handshake components, connected by handshake channels. Within the Tangram tools, a number of predefined handshake components are available that are combined into a circuit following a description in a high-level language. A detailed overview of the design and implementation of handshake circuits can be found in [45, 63, 8].

1.1.1 Handshake channels

Handshake channels can be implemented in many different ways, but their functionality is always the same: to synchronize two handshake components [57]. The ports at the ends of the channel can be either active or passive. A channel is activated by the handshake component connected to the active port of the channel and it uses its passive port to activate the handshake component connected to that port. A channel can be used to transfer data between a sending component and a receiving component, but it can also be used purely as a synchronization event. The concepts of sender - receiver and active - passive are orthogonal. Depending on the application, a sender can be either active or passive.

Within the Tangram tools, three types of channels are used. They are shown in Figure 1.1. The first channel is only used for synchronization, there is no data transfer. This type is called a *nonput* channel. Since there is no data transfer, there is also no sending and receiving side, only an active and a passive port. The active

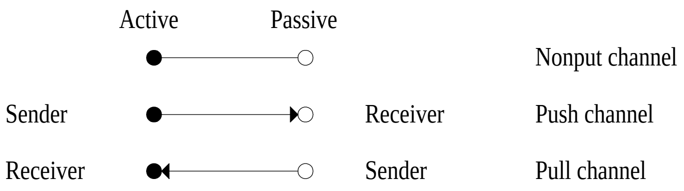


Figure 1.1: Handshake channel types used in Tangram.

port is represented by a filled circle, the passive port by an open circle. The channel is implemented with two signals, a request and an acknowledge.

The other two types of channels are used to transfer data. The first of these two channels is used to transfer data from the active port to the passive port. The direction of the data is indicated by the arrow drawn in the channel. The active port essentially pushes data to the passive port and is therefore called a *push* channel. In the last channel, the active port requests data form the passive port and is called a *pull* channel. This terminology was first introduced in [45].

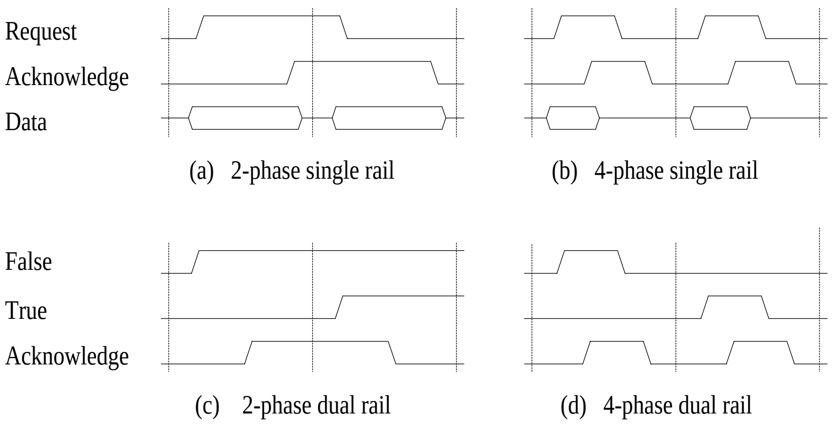


Figure 1.2: Examples of handshake protocols.

Channels can be implemented in various ways. The first choice is between 2-phase signaling and 4-phase signaling. The 2-phase signaling, shown in Figure 1.2(a) and Figure 1.2(c), uses fewer signal transitions for the request and acknowledge signals, which reduces the power consumption in the channel. However, the handshake components connected to these channels need to respond in the same way for both up and down transitions. In general this requires a more complex implementation of the handshake components, removing much of the benefits of the 2-phase signaling protocol. For this reason, Tangram based handshake circuits use a 4-phase signaling protocol, shown in Figure 1.2(b) and Figure 1.2(d), which allows for a simpler implementation of the handshake components.

A second implementation option, in case of a pull or a push channel, is the encoding of the data. An elegant encoding is the dual-rail encoding, which uses two data wires for every bit, a true wire and a false wire: {true, false}. The request information is encoded in these data wires and not available as a separate signal. A zero is transmitted by raising the false-wire {true, false} = {0, 1} and a one is transmitted by raising the true-wire {true, false} = {1, 0}. The state {1, 1} signals invalid data; if this state is detected, an error has occurred in the circuit. Data is transmitted over the channel by interleaving a data item (either true or false) with a null (or empty) {0, 0} signal. This enables the receiver to determine whether or not data on the channel is valid. This ensures that the channel is totally independent of wire delays. The obvious disadvantage of dual-rail signaling is the increased complexity of the required logic and the overhead of the additional wires.

A second data encoding option for the channel is to use normal Boolean encoding, also called single-rail encoding. The advantage is that it allows the use of conventional Boolean logic, which is smaller and can be optimized with standard logic optimizers. The disadvantage is that the receiver can no longer determine when the data on the channel is valid, just by looking at the data signals. In single-rail encoding, the request and acknowledge signals have to be delayed sufficiently long to allow the data to become valid. For this reason, delays are added to the request and acknowledge signals that will delay the handshake until the data on the receiving side of the channel is guaranteed to be valid. Example protocols of all four combinations, single rail versus dual rail and 2-phase versus 4-phase are shown in Figure 1.2. In handshake circuits generated by Tangram, the 4-phase handshake signaling protocol is used in combination with single-rail data encoding [45], corresponding to Figure 1.2(b).

1.1.2 Handshake components

Handshake channels are used to connect handshake components together to form a handshake circuit. Handshake components implement commonly used predefined behavior. About 40 different types of handshake components exist, some of which have parameters, for example to specify the number of ports. All handshake components have ports that connect to handshake channels. Like the ports of a handshake channel, these ports can be either active or passive and have optional data inputs or outputs. Every component corresponds to a different Tangram language construct. The most common handshake components are shown in Figure 1.3. The four components at the top of the figure only have nonput ports, and are used to control the flow of activity in a circuit.

- The *repeater* is activated once via its passive port and will then generate a continuous stream of handshakes at its active port.

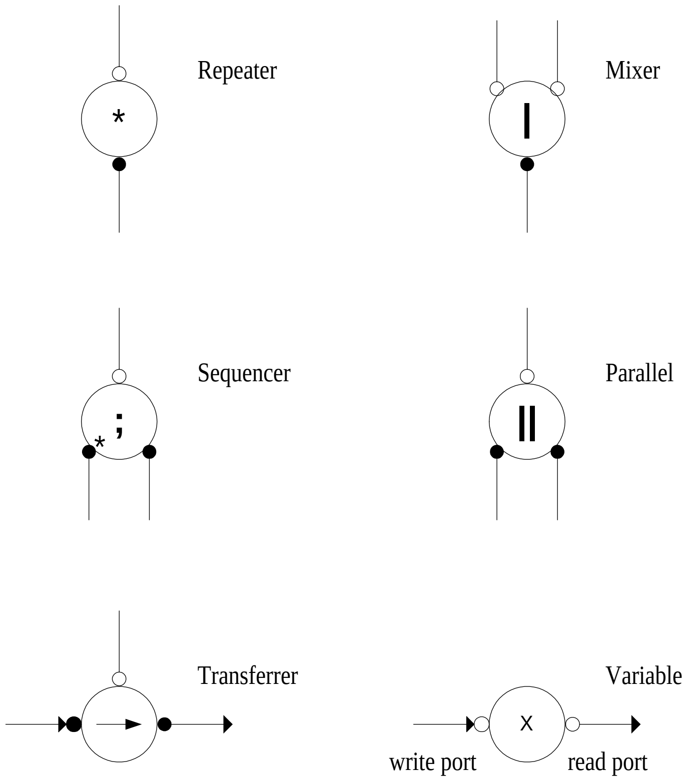


Figure 1.3: Some common handshake components.

- The *mixer* is able to accept handshakes via its two passive ports (these handshakes must be non-overlapping) and passes them on to its active port.
- When activated via their passive ports, the *sequencer* and the *parallel* components activate both of their active ports. The sequencer, however, activates them sequentially, starting with the port labelled with the $*$. After the handshake at that port has been completed, a handshake is started at its other active port. The parallel component activates both its active ports in parallel, but waits until both handshakes are completed before the handshake at the passive port is completed.

The other two components shown in Figure 1.3 have push and pull ports, indicated by the arrows, and can therefore operate on data.

- The *transferrer* is activated via a nonput channel on its passive port and will first initiate a handshake on its left active port. This port is connected to a pull channel and will retrieve a data value from the component connected to that channel. The received data value is then sent to the active port on the right. This port is connected to a push channel and sends the data value to a component connected to this channel. In Tangram, the components connected to the left channel are usually binary data operations or read ports of variable components. The right channel is usually connected to the write port of a variable.

- The *variable* is used to store a data value. Both its ports are passive. The left port is used as a write port; a handshake at this push channel delivers a new data value that the variable stores in its internal register. The right port is used as a read port; a handshake on this port reads the data value stored in the variable. A variable component can have more than one read port. Variables are connected to transferrers, usually with various data operations in between. The internal register of the variable can be based either on latches or flip-flops and can be of arbitrary word width.

Handshake components are always activated via their passive port(s) and can activate other handshake components via their active port(s) [8]. This leads to a layered structure in which a handshake component on one layer activates components at a lower layer. This is repeated until at the lowest level passive components are reached. The passive port of the handshake component in the top-layer is connected to an external channel of the circuit. This channel is referred to as the start-up channel of the circuit. By initiating a handshake on this channel, the circuit is started. Since a normal circuit will keep running, the handshake on this channel is usually never acknowledged. In case this would happen, it indicates that the circuit has stopped executing. By keeping the request signal of the start-up channel at logic zero, the circuit is initialized. The initialization process starts at the components at the top layer and gradually propagates through the other layers of the circuit until all components are initialized.

1.1.3 Tangram

The Tangram design tools [11] form a powerful way to design handshake circuits. Circuits are described in a high-level programming language. The language constructs directly correspond to handshake components. For example, infinite repetition offered by the “`Forever Do` $\cdots$ `Od`” construct is implemented with a repeater component. This results in a *transparency* property of the compiler, which allows the designer to reason about circuit properties such as area, power and performance at the programming level. This can be used for rapid evaluation of design choices. The used language is similar to a conventional programming language like C or Pascal, but there are additional language constructs available to describe parallelism and channel communication.

The Tangram design flow is shown in Figure 1.4. After the design engineer has written a Tangram program, the program is compiled into a handshake circuit descrip-

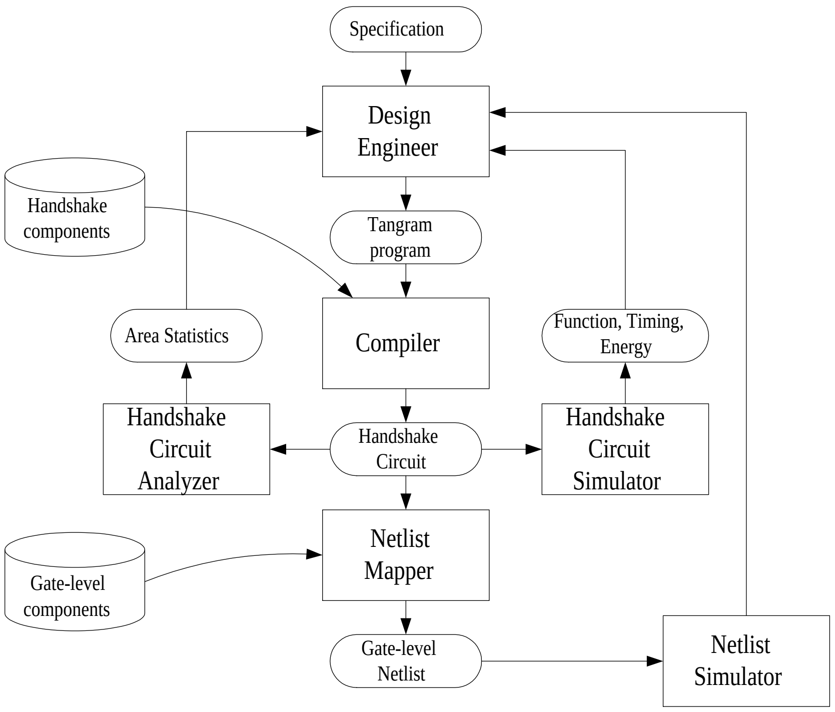


Figure 1.4: Tangram design flow. Tangram uses a two step synthesis process, first the Tangram program is compiled into a handshake circuit, second the handshake circuit is mapped onto a gate-level netlist.

tion, using a library of handshake components. The handshake circuit description is simulated and analyzed to get a rough estimate of various properties, like functionality, timing and size of the circuit. The simulation can be very fast, since it uses only the abstract handshake components and no detailed low-level information.

Handshake circuits can be mapped onto several alternative fabrication technologies and circuit styles. The process uses a mapping library that contains gate-level implementations of all handshake components in the target technology. Once every handshake component is mapped to a gate-level netlist, the resulting netlist is optimized for area and simulated again, now at gate-level. The gate-level netlist forms the input for the remaining tools that will ultimately lead to a layout for the circuit. These tools are conventional tools that are normally used for synchronous circuits.

1.2 Production testing

Production testing is a vital step to ensure that any defective ICs are removed from the production line and not shipped to customers. During the production of an IC, a range of possible defects can occur that may render the IC unusable or limit its lifetime. Some of the most common defect types are [30]:

- Opens, or highly resistive signal lines
- Shorts, or low resistive bridges between signal lines
- Parameter variations, such as threshold-voltage variations

The actual physical effect of a defect is difficult to quantify; it can range from resistance variations to parasitic transistors. For this reason, abstract fault models are being developed. In these models, faults are used that offer a simplified view of how a defect changes the behavior of a circuit. Many fault models exist, all targeting a set of potential defects. An overview of these fault models is given in [13]. The most popular fault model still in use today is the stuck-at fault model [25], even though it is over 40 years old. This model uses faults that keep an input of a logic gate or the output of a logic gate at a constant (one or zero) value. Although it cannot detect certain types of defects, it is popular because of its simplicity and high-quality tool support. Two derivative models of the stuck-at fault model are the stuck-at input model and the stuck-at output model. These models only contain the stuck-at faults at the inputs (stuck-at-input) or at the outputs (stuck-at-output) of logic gates.

During the testing process, stimuli are applied to the circuit and the response of the circuit with regard to these stimuli is observed. The responses are compared to the expected responses and if there is a mismatch, the circuit is considered to be faulty. This process is repeated many times until all testable faults in the used fault model are tested.

Two parameters are commonly used to express the testability of circuits: the controllability and the observability. The controllability defines the ability to set a signal in the circuit to a specified value. The observability defines the ability to observe the value of a signal in the circuit. External inputs to the circuit, called primary inputs, can always be directly controlled, their controllability is defined to be one. External outputs of the circuit, called primary outputs, can always be directly observed and their observability is defined to be one. The controllability and observability of other nodes in the circuit can range form zero to one. A value of zero means that the node is untestable. It is usually not possible to apply the stimuli for an internal node directly at the primary inputs of a circuit. It might require a sequence of patterns or the help of special on-chip test circuitry to reach the target node. By adding more special test structures on-chip, the testability of the chip can be increased [69].

Iddq testing

Faults can also be detected by observing the supply current of the circuit, a process known as Iddq testing [14, 27, 37]. A characteristic of advanced CMOS logic is the very low standby current. Many defects cause this standby current to increase, which can be measured. Since only the standby current has to be measured, the circuit has to be in a quiescent state to avoid any dynamic currents. In synchronous circuits this is simply accomplished by halting the clock. Although observing the fault effects is easy, the faults still have to be exercised. Therefore the controllability problem is similar to that of normal logic testing. Iddq testing can detect many defects that are not covered by the stuck-at fault model, like resistive bridging faults [56] and delay faults [61, 62].

Unfortunately, for new process generations, Iddq testing is becoming increasingly limited in practical use. This is because the standby current of these new process generations is increasing compared to the other contributions to the power consumption. This makes it more difficult to differentiate between good and bad circuits and reduces the sensitivity of the method [21]. Another drawback of Iddq testing is the relatively long time it takes to carry out a measurement.

1.2.1 Functional versus structural testing

Test methods can be divided into functional and structural methods. Functional testing exercises the functions of a circuit, whereas structural testing is based on the circuit structure. The distinction between functional and structural testing was first made in [20].

Functional testing

In functional testing, the circuit is tested in the normal operating mode. There are no special circuit modifications added to the circuit to support the test. The advantage is that no additional area is required for test structures and that the function of the circuit is tested in the same way as it is used in normal operational mode. The tests can be carried out at-speed, ensuring that critical timing problems can be detected. The major problems with functional testing are the difficult and labor intensive (manual) test-generation process and the often limited fault coverage.

Unless something is known of the underlying logic, the only way to fully test a circuit is to exhaustively exercise its functions. In reality this would lead to impractically long test times. Therefore a significantly shorter test has to be applied, implying a reduced fault coverage. In certain application areas, functional testing is still an important part of the overall test strategy. For example in a microprocessor it is used to test the IC at-speed and the result can be used for speed binning to grade the circuits performance [44].

Structural testing

The main characteristic of a structural testing is that faults from a selected fault model are tested in conjunction with the structure (gates and interconnect) of the circuit.

The goal is to detect all faults in the model, using the minimal time and cost. The test patterns that are used for this are usually generated by an automatic test-pattern generator (ATPG) tool.

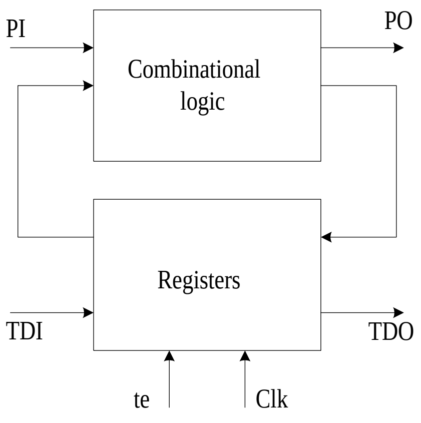
The ATPG tool uses a fault list to keep track of the obtained fault-coverage. The first step is to combine faults that have the same effect on the circuit, the so called equivalent-fault removal step, to reduce the number of faults in the fault list. After this step, the patterns are generated. Every time after a new test pattern is generated, the pattern is simulated to determine which other faults it can detect. These faults are subsequently removed from the fault list. This process continues until the fault list is empty or only contains untestable faults, which can for instance be caused by redundancy in the circuit. Many algorithms have been proposed, most of them targeting combinational circuits. The first and best-known combinational ATPG algorithms are the D-algorithm [54], PODEM [26] and FAN [22]. Current state of the art algorithms can achieve an estimated speed up of over 25000 times over the early D-algorithm [65].

1.2.2 Full-scan testing

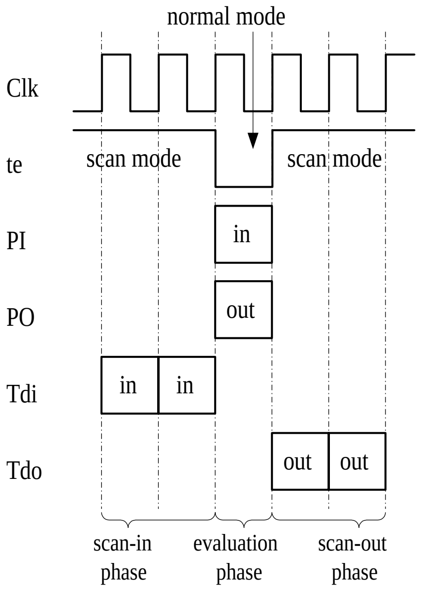
Design for Testability (DfT) refers to using a design style in which hardware modifications are included in a circuit to make the circuit better testable. Controlling primary inputs and observing primary outputs is always possible. However, for internal nodes of sequential circuits it can be very difficult to control and observe the value at the node. It can take an arbitrary long sequence of inputs to set the state of the internal node. In addition it is difficult to compute such a sequence and it can take a long time to actually apply the sequence to the circuit. In [69] test points are inserted in a circuit. A test point makes a point fully controllable and observable. Full-scan test systematically adds test points to the circuit. The effect is that the test problem is reduced to the well-known problem of testing a combinational circuit. The modification consists of adding a multiplexer to every register in the circuit. These multiplexers are used to implement are serial shift register through all the registers in the circuit. The shift register can be used to shift data in and out of the circuit. The multiplexers are controlled by a new control signal called the test enable te.

The full-scan test principle is shown in Figure 1.5(a). The shift register makes it possible to control the outputs of all registers and observe the inputs of all registers. The outputs of the registers operate as pseudo inputs for the combinational logic. Together with the primary inputs, all inputs of the combinational logic can now be controlled. The inputs of the registers operate as pseudo outputs. With the pseudo outputs and the primary outputs all outputs of the combinational logic block can be observed.

A scan-testable circuit has two modes of operation, controlled by the test enable (te) signal. The first mode is the scan-mode, used to serially shift data in and out of the registers. The second mode is the normal-mode, in which the circuit operates



(a) Scan test circuit



(b) Scan test protocol

Figure 1.5: Full-scan test principle. (a) The circuit is modified to include a scan input (DTI) a scan output (TDO) and a test enable signal. (b) Scan test protocol showing when scan signals are valid.

normally. A scan test is executed following a test protocol, shown in Figure 1.5(b). It consists of three phases. In the scan-in phase, the registers are loaded with a test pattern via the external scan-in pin TDI. The second phase is the evaluation phase, which consists of one clock cycle executed with the circuit in normal-mode. During this phase, the primary inputs (PI) and outputs (PO) are also active. After this phase the registers contain the response of the circuit to the test pattern. The third and last phase, the scan-out phase, again uses the scan-mode to shift the response out via the external scan-out pin TDO. Scan testing (of a synchronous circuit) requires investing in circuit area in the form of a multiplexer for every register, wiring for the test-enable signal to control the multiplexers and wiring to connect the registers to each other.

Any circuit that obeys certain rules can be made scan testable. The following scan test rules are valid for synchronous circuits based on flip-flop registers:

- Only D-type master-slave flip-flops can be used as register elements.
- At least one primary input and output pin have to be available for test. These will be used as the scan-in and scan-out pin.
- All flip-flop clocks must be controllable from a primary input.

- Clocks must not be connected to the data input of flip-flops.

An alternative scan method uses latch-based registers. This method is called level-sensitive scan design (LSSD) [19]. Every LSSD scan register consists of two latches: a master latch L1 and a slave latch L2 as shown in Figure 1.6(a). The master latch has two enable signals, clk_1 and Te, the first to capture data at the normal data input d, and the second to capture data at the scan-data input ti. The slave latch is clocked with a third enable signal clk_2. The three enable signal may not be active simultaneously.

1.2.3 Other scan methods

In this thesis two scan methods are mentioned that are derived from the full-scan test method. These are the L_1L_2* scan and partial scan methods.

L_1L_2* scan

L_1L_2* scan , described in [16], is an optimization of the LSSD scan method. A large part of the area overhead in LSSD scan is caused by the slave latches that are required for every master latch. In the LSSD method, master latches are referred to as L1 latches and the slave latches as L2 latches.

The principle of the L_1L_2* scan method is shown in Figure 1.6(b). The slave latch is now also scannable. It is called a L_2* latch to distinguish it from a non-scan normal slave latch. A portion of the logic is placed between master L_1 and slave L_2* latch, the other portion remains at the normal position between the two latches. L_1 and L_2* latches have independent test enable signals, to allow one to stay in scan mode while the other can be in normal mode. The advantage of the L_1L_2* scan method is that in the ideal case no slave latches are required anymore. This results in a lower area overhead, lower power consumption and less impact on circuit performance as compared to the LSSD scan-test method. In a real circuit it is generally not possible to find an ideal partitioning of the circuit. In that case some master latches still require a dedicated (non-scan) slave latch.

Partial scan

In the partial-scan technique , only a subset of the total number of registers is scanned. This is used to find a balance between the complexity of test-pattern generation and the area overhead. By only scanning a part of the registers, the area overhead decreases, while the test generation becomes more complex. Usually in partial scan, scan is applied to break at least all large cycles in a circuit [15]. In order to find such a partitioning, a number of heuristics exist [3]. The elements that are not scanned are selected such that it is still possible to test them via the primary inputs and the pseudo inputs, formed by the outputs of the registers that are still being scanned. Since the

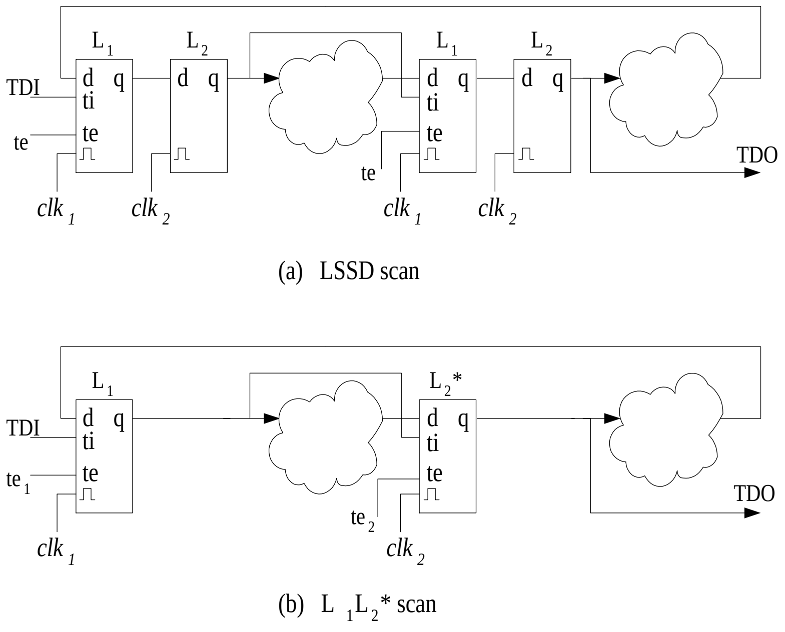


Figure 1.6: The LSSD (a) and L_1L_2* (b) scan-test principles. In LSSD scan every scan element consists of a master and a slave latch. In L_1L_2* scan, scan elements only consists of a master element.

resulting circuit is no longer combinational during scan test, in general *sequential* ATPG is required to generate the test patterns.

If special restrictions are placed on the selection of the non-scan elements, it is still possible to use combinational ATPG to generate the test patterns. An example of such an approach is the "SmartScan" method described in [40]. The resulting sections that are not scanned are pipeline structures. The internal registers are considered to be transparent during test-pattern generation. During test execution, the generated test pattern needs to be kept valid for a number of cycles, until the values reach the end of the pipeline. The method is less suitable for circuits in which large pipeline structures are not common; this for example holds for the control part of a handshake circuit.

1.2.4 Economy of testing

The cost of testing is becoming an increasing part of the total manufacturing cost of an IC [58]. Intel reported that the major part of capital cost for a new fab is spend on verification and manufacturing testing equipment [60]. There are many factors that determine the total cost for test. Unfortunately many factors are mutually dependent and differ from one product to the other. This makes it difficult to minimize the cost, leading to research to analyze the various contributions to the test cost [1, 18]. An economic model is described in [66] that allows the evaluation of alternative test strategies. The main criteria that are evaluated are:

- Test quality
- DfT cost
- Test-development effort
- Test-equipment cost

With respect to these criteria, handshake circuits are no different than conventional synchronous circuits. Therefore these criteria are directly and equally applicable for handshake circuits. The mentioned test criteria are further discussed in the following sections.

Test quality

Obtaining a high test quality is an economic necessity to be profitable. Often this is explained with the “Rule of 10”. If a defective chip is not discarded, it usually costs 10 times as much to discard (or repair) the board containing the defective chip and again 10 times more to discard (or repair) a system containing a defective board.

Production processes will produce a mix of good and faulty dies, the fraction of good dies out of the total number of dies is called the yield. The testing process must remove the faulty dies from the production line, leaving only the good dies. Additionally the test must not remove any of the good dies. The final quality is often expressed in the number of defect parts per million (ppm) of total shipped parts. The final goal is to reach a single digit ppm. By analysis of the defective ICs, useful information can be obtained that can be used to improve the fabrication process, the design rules used to design a IC in that process and the test vectors that are used to test the IC. This will gradually improve the yield of a given process.

DfT costs

The logic required to implement the DfT features in a circuit, increases the total silicon area of the circuit. Except for the direct cost of the additional area, this also has a negative impact on the yield of the chip. The yield depends on the sensitive area of a chip, the part of the chip were a defect can result in a faulty chip. Even if the added DfT circuitry uses previously unused space, the silicon area might not increase but the sensitive area will. This can therefore still have an impact on the yield.

Besides additional area, the modifications will also have a negative impact on the circuit performance and power consumption. The multiplexers in the scan chain increase the critical-path delay. The power increases even when the multiplexing logic is not used because the additional logic increases the capacitive load on other signals.

Special pins might be required to control the DfT logic. Most pins will be multiplexed onto existing pins, but a few additional control pins are usually added. Adding pins can be costly, as they increase the die size by requiring additional pads. If the pins also need to be available in the final packaged chip, then the additional pins also have to be available in the package and bonded to the package.

Test-development effort

The test-development effort represents the work that needs to be done by the designers to make a circuit testable and to generate the test patterns and the work done by the test engineer to debug the test program and get it running on a tester. The amount of work that this requires can directly increase the time-to-market. Although some tasks can be done in parallel with the completion of the chip, the generation of functional patterns for example can still be the critical path to get the product shipped. The amount of required work is becoming an increasingly important criterion for test methods. Especially in those cases were the time-to-market might otherwise increase, since that would directly limit the maximum market revenue [18].

The best way to minimize the test-development work, is to use automated and standardized test tools. In practice this means a form of DfT insertion combined with ATPG. Ideally this would be a push-button method, but in most cases at least some manual interaction is required, for example to integrate the test with other blocks (like memories or analog circuits) at the top-level of the IC.

Test-equipment costs

Another part of the cost is the required expensive test equipment and the cost to operate this equipment. To reduce these costs, test times spend on this equipment needs to be minimal and the available test equipment should be maximally reused. There are four main contributions to the equipment cost:

- Purchase of the equipment; the price for example depends on the required speed, number of pins and amount of memory per pin.
- Depreciation
- Labor cost and environment cost (eg. cooling) to operate the equipment
- Utilization factor

As mentioned before, the equipment costs represent a major part of the total cost. It is therefore important to design test methods that limit the requirements of the equipment and if equipment is already available, the test of new ICs should preferably run on that equipment to improve the utilization factor.

1.3 Motivation

The output of the Tangram design flow is a gate-level netlist. For the remaining tasks like timing analysis, logic optimization and layout, commercial tools are used. This part of the design process is increasing in importance in practice, since circuit performance in new technologies becomes more and more dependent on actual layout and wire length and less on the gates in a design. One of the tasks that has to be preformed during this phase is the development of a production test strategy. This consists of test-pattern generation and possible hardware modifications to ease the test-pattern generation. Neither of those is supported by the Tangram toolkit. There are also no commercial tools that are able to provide these functions for handshake circuits.

For an economical test, a balance between all the criteria in the last section has to be found. Previous attempts to develop a test method for handshake circuits, focused primarily on the minimization of the additional area required for test modifications. This led to methods that require a large amount of manual test-development work and that had problems in obtaining a sufficiently high fault coverage. For current products, the reduction of test development work and the increase in fault coverage are becoming more important than the increased circuit area. Therefore a new test method should focus on reducing the test-development work and on increasing the test quality, even if this increases the cost for additional test circuitry.

The need for a structural test method is increasing further, because the number and diversity of the circuits designed with the Tangram toolkit are increasing. Many of the initial Tangram designs had little need for an automated structural test method. Most of these designs were built around an 80c51 micro-controller. For this controller a functional test program was available that could achieve adequate fault coverage. Newer designs, however, become increasingly larger and more complex. Furthermore they start to span a wider application area, leading to less commonality between the designs. These developments make manual test development impractical. For a continued and successful widespread application of the Tangram design technology, an automated structural test method is essential. The lack of an effective test method for handshake circuits is beginning to hold back the uptake of the technology. Whenever the technology is evaluated for potential use in a new application area, the testability of the circuits is a major requirement.

If handshake circuits are used in a new application area, they usually replace the existing design style. The transition has to be as smooth as possible. This not only holds for the knowledge of the designers but also for the existing infrastructure. Furthermore, conventional synchronous circuit and handshake circuits can share a lot of the facilities used for production and test. In order to economically test handshake circuits, the test method has to work on the same equipment that is used for synchronous circuits. If the asynchronous test strategy requires additional features of equipment, the cost of testing can rise significantly.

1.4 Objectives

The objective of this work is to develop an automated structural test method for handshake circuits. To be industrially acceptable, there are several properties that the test method must fulfill. These properties can be divided into essential requirements and optimizations to reduce cost. The requirements for the test method are the following:

- High (structural) fault coverage. Initially, the stuck-at fault model is used and the fault coverage should be at least conform the industry standards. In this thesis full (100%) coverage is targeted. If this coverage is not obtained, the analysis of untestable faults can lead to a deeper understanding of the specific test problems of handshake circuits and improvement of the test method, as is shown in Section 4.3.5.

- Automated test flow. This enables a high productivity and consequently reduces the time-to-market. It also produces a test solution with constant cost and quality, enabling more accurate planning of projects. This in turn again leads to better design decisions and a higher productivity.

- Compatibility. To keep the cost down and improve the acceptance of the method by designers, the method has to be compatible as much as possible with existing test tools and practices and with existing test equipment.

These requirements are essential for a successful test method, but in addition it is also very important to reduce the cost of the test method. As discussed in Section 1.2.4 there are many contributions to the test cost. The contributions that are most important for this work and should be minimized are the following:

- Area overhead. Additional silicon area is required to implement the test hardware. This includes area for gates, wiring and pins.

- Impact on performance. The additional test circuitry can have a negative influence on the performance of the circuit. This might have an impact on the targeted application area.

- Power consumption. Besides an impact on performance, the additional test circuitry can also increase the power consumption. Since many applications use handshake circuits because of their low-power consumption, an increase in power consumption can reduce the possible application areas of handshake circuits.

These objectives have led to an approach in which a clock is inserted in a handshake circuit that is used to add a synchronous mode of operation to the circuit. This allows the application of scan techniques to simplify the test problem into the well-known combinational test problem.

The method requires a modification of all sequential elements in the circuit. For those that are not found in synchronous circuits, like C-elements [63], new scan elements have been designed. The scan-test method is implemented in such a way, that not only makes it possible to test all faults in the modified circuit, but also guarantees an unmodified functional *asynchronous* mode of operation. During the normal asynchronous operation, however, the performance and power consumption of the circuit could be negatively influenced.

1.5 Original contributions

The original contributions made in this thesis are:

- The development of a full-scan test method that allows the testing of large handshake circuits with high quality and automatic test-pattern generation. In addition the method is compatible to existing tools and standards, which allows simple integration with other blocks on the chip.
- Allow the testing of locally generated clock signals, by adopting an approach in which the control block and data path are tested separately.
- Introduction of new scan cells that are used to significantly reduce the area overhead of the test modifications.
- Special modifications to test specific asynchronous structures, like the mutex elements. Also in some cases the netlist generator has been modified to generate better testable circuits.
- Implementation of a new test tool TgScan, which is used as a scan insertion tool and that generates files to interface with *existing* test tools that are used to generate the test patterns and produce the final test vectors.

1.6 Thesis outline

The structure of this thesis is as follows:

- Chapter 2: Introduction in the testing of handshake circuits. The specific test problems associated with asynchronous circuits are discussed and a number of previously proposed solutions are evaluated.
- Chapter 3: Scan test is applied to handshake control circuits. This requires the design of scannable C-elements. Several alternative implementations for these elements are given and it is shown how to incorporate them in a circuit.

- Chapter 4: Scan is applied to the complete handshake circuit. A method is presented that can test any handshake circuit designed with the Tangram toolkit. Both the required circuit modifications and ATPG approach are discussed.

- Chapter 5: To be able to apply the test method on a number of benchmark circuits, a new tool called TgScan is presented. This tool modified the original handshake circuit into a scan testable circuit. Also presented are the scannable C-elements that are required to execute the experiments. Subsequently, these are used to make several benchmark circuits scan testable and the results of these experiments are given. Finally some work is presented on an industrial demonstrator circuit.

- Chapter 6: Conclusions are given and recommendations presented for future work.

- Appendix A: The Philips CAT tools are applied to scan testable handshake circuits. The tools use a number of control files to correctly recognize handshake circuits.

Chapter 2

Testing of handshake circuits

Like all other circuits, handshake circuits have to be tested for manufacturing faults after coming out of the production line. This has to be carried out fast and with high quality in order to keep the cost low and deliver functioning products to the customers. Design and test need to be considered together to meet the cost and quality requirements. Designing a circuit without accounting for test will usually result in an untestable circuit, likewise testing a circuit without looking at its design is likely to result in an expensive test solution.

Several test methods have been proposed that analyze the testability of the circuit at the handshake level [53, 68, 17], which is easier to analyze than a gate-level circuit. The conventional stuck-at fault model however is modelled at gate level and existing the test-pattern generation tools for this fault model work with gate-level circuits. For this reason, in this thesis testing is only discussed at gate-level.

In this chapter the gate-level implementation of a handshake circuit is reviewed first. This is followed by a set of properties that handshake circuits exhibit and that need to be considered during the design of the test method. These properties are then used to evaluate some test methods previously proposed in literature. Finally this leads to a discussion of the test approach followed in this thesis.

2.1 Gate-level implementations of handshake circuits

Gate-level implementations are derived from handshake circuits by substituting every handshake component in the circuit by a corresponding gate-level implementation of the component. Handshake components are designed to be generic and provide well-defined interfaces to other components. Hence, for every handshake component a

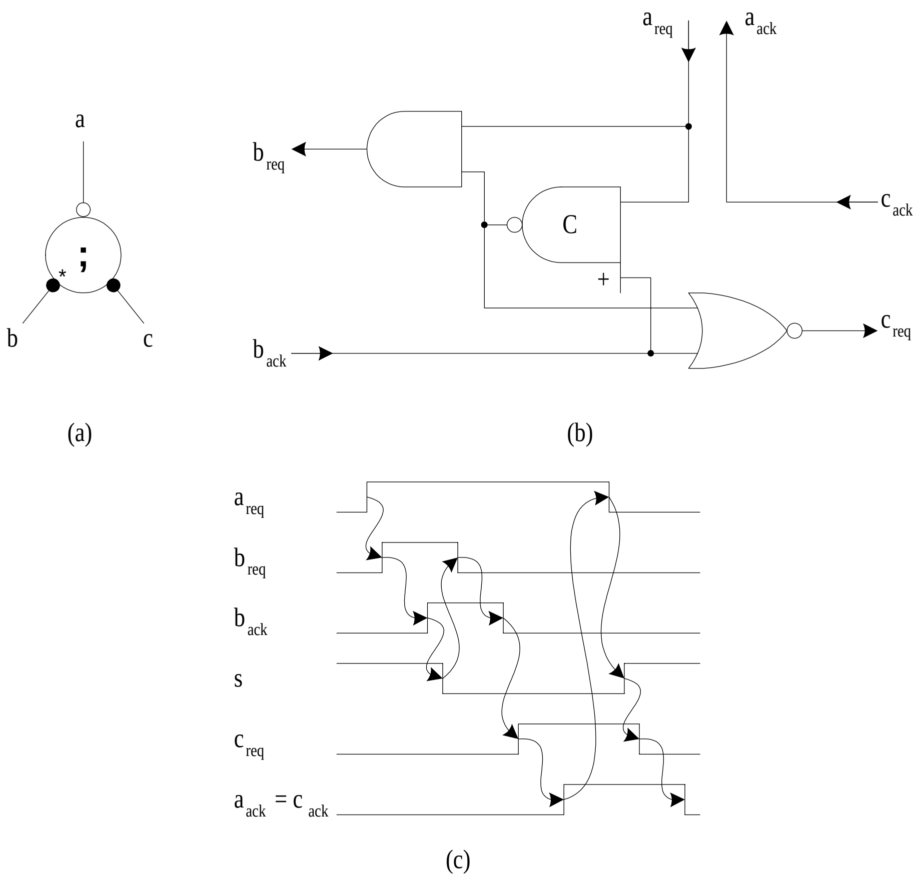


Figure 2.1: Symbol of a sequencer component (a) its gate level implementation (b) and a simulation of one complete handshake (c).

gate-level version can be designed and implemented independently. The technique used for this implementation can in principle be chosen from literature, but the components used here have been mostly designed using handshake expansion [42].

Figure 2.1(a) and (b) show the symbol and gate-level implementation of the sequencer component that was introduced in Chapter 1. The simulation shown in Figure 2.1(c) shows how after starting a handshake on channel a by raising a_{req}, first a complete handshake is preformed on channel b, followed by a handshake on channel c. The arrows show the sequence of events during the simulation. The implementation requires three gates, two of which are normal combinational gates. The other gate is a so-called C-element, which is a sequential gate similar to a set-reset latch. C-elements are commonly used in asynchronous circuit design, usually in several variants. The output of the C-element in Figure 2.1 becomes one if a_{req} is zero and it becomes zero if both a_{req} and b_{ack} are one. Since the function for setting and resetting are different, this type of C-element is an asymmetric C-element. Other types C-elements are also used, for example those with symmetric set and reset functions, therefore called symmetric C-elements. The function and implementation C-elements are further explained in Section 3.2.4. The symbol used for a symmetric C-element is an AND-gate symbol with a "C" written in it. In case of the asymmetric C-element in Figure 2.1, the symbol is modified with a $+$ to designate the input that only helps to make the C-element to evaluate to one internally.

Gate-level implementations of handshake circuits are generated by replacing all individual components with their gate-level implementations. An example of a handshake circuit implementation is shown in Figure 2.2. The figure shows a one-place

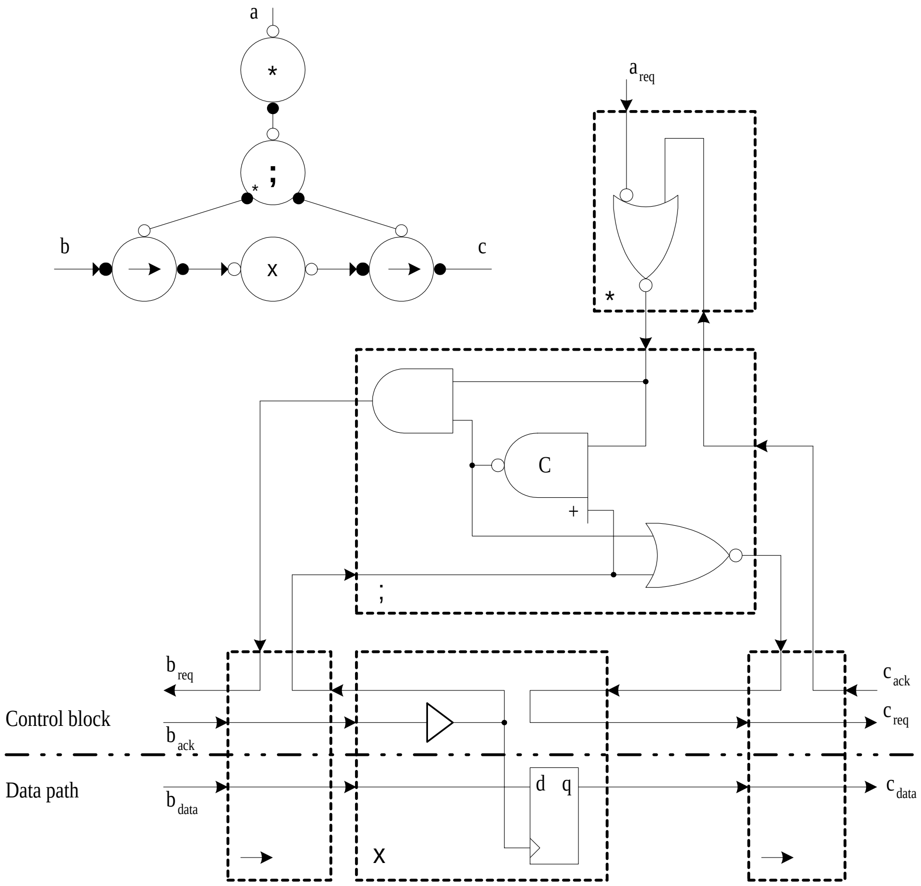


Figure 2.2: Example of a handshake circuit consisting of five handshake components and its gate-level implementation. In the implementation every dotted box contains the implementation of a handshake component.

buffer, which is a circuit that repeatedly copies the value read at input b to output c. The circuit consists of a repeater, a sequencer, a variable and two transferrer components. The repeater generates a continues stream of handshakes. The sequencer sends these handshakes first to the left transferrer, to load a new value from the input b into the variable and then to the right transferrer, to copy the value in the variable to the output c. In the circuit diagram, the logic corresponding to the individual components is identified by the dashed boxes.

The implementation of the sequencer has already been shown in Figure 2.1. The repeater contains only one gate. When activated by making a_{req} high, it operates as an inverter between the acknowledge signal coming from the sequencer and the request signal going to the sequencer. This will automatically generate a new handshake to the sequencer, when the previous handshake finishes. Note that the repeater does not have a top-level acknowledge signal, since the program never ends and a top-level acknowledge signal would therefore never become high. The variable consists of either latches or flip-flops and some control circuitry to generate a local clock signal for these latches or flip-flops. The transferrers are implemented with wires only.

In the remainder of this thesis, the difference between control logic and data path logic plays a central role. In handshake circuits, there is a fundamental difference between the two. Control logic operates on handshake signals, data path logic works on Boolean signals. The difference between control and data is already evident at the handshake level. Both handshake channels and handshake components can be divided into a control part and a data part. For nonput channels and handshake components that only have nonput ports, the data part is empty. Consequently these channels and components only consist of a control part. In Figure 2.2 the transferrer components, the variable component and the channels in between these components can be split up into a control part and a data part. At circuit level this leads to a control block that contains the control parts of all handshake channels and components and a data path that contains the data part of all handshake channels and components. In Figure 2.2 the dotted line shows the partitioning in control block and data path.

The resulting partitioning is shown schematically in Figure 2.3. The interface between the control block and the data path consist of the following three types of signals:

Conditions The control block uses the condition signals in combination with its internal state to determine the next action.

Parameters Parameters are used by the control block to control the Boolean logic in the data path. The most common use is to set multiplexers in the correct state.

Local clock signals The local clock signals are used by the control block to enable a data-path register to capture a new data value.

The control block determines the operation and activation of the rest of the circuit. Once activated via the start-up channel labelled "Start", it will start to interpret

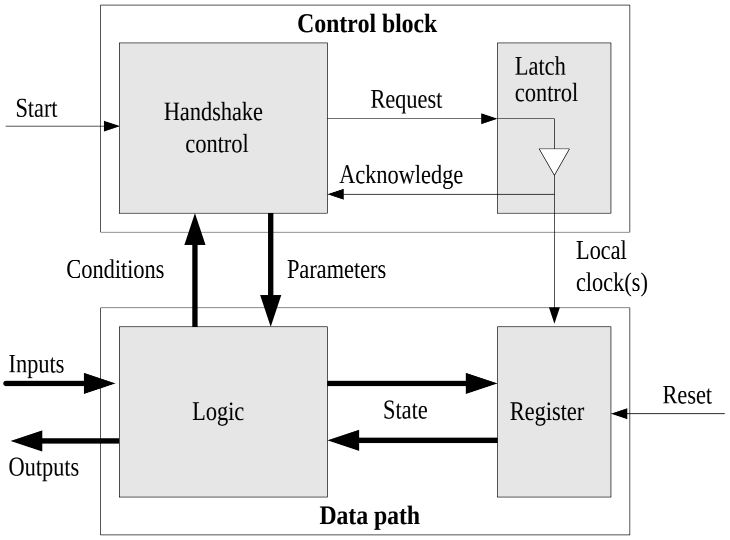


Figure 2.3: The gate-level implementation of a handshake circuit can be partitioned into a control block and a data path, both of which have a specific structure and properties.

conditions from the data path, set the parameters to the data path and send clock signals to the registers by activating the appropriate latch controllers. The data path in a handshake circuit is very similar to that of a conventional synchronous circuit. The only difference is that the registers are clocked by clock signals generated by the control block.

In the partitioning, the variable components have a special place. They are implemented by the latch control and the register blocks that are shown separately in Figure 2.3. The latch control block contains the control logic of a variable that is used to increase the drive strength of the clock signal to match the load of the register elements connected to it in the data path. The register block only contains the register elements of the variable.

2.2 Test properties of handshake circuits

Handshake circuits have some test properties that are not found in synchronous circuits. Unfortunately most of these properties, which are discussed in this section, complicate the use of standard synchronous testing methods like full scan. Only one property, the acknowledgement property, offers new possibilities for testing although only for specific circuit styles and fault models. The other four properties in this section complicate or even prevent the use of the standard test methods.

2.2.1 Acknowledgement property

The acknowledgement property [43] states that if every signal transition in a circuit is acknowledged by another signal transition, then the resulting circuit will operate correctly regardless of the delay in the wires. This property is useful for testing, because any stuck-at fault that is present will prevent a signal transition and this halts the circuit operation. In most cases this inactivity will quickly propagate throughout the circuit, leading to a circuit deadlock. Unfortunately the acknowledgement property only holds for the class of delay-insensitive circuits. As proven in [41], circuits in this class can only be constructed out of symmetric C-elements and inverters, which limits the type and efficiency of the possible circuits.

In handshake circuits a derivative circuit design style is used that allows the use of isochronic forks [7, 42]. In an isochronic fork, a transition is send to multiple receivers but only one of these receivers responds by sending back an acknowledge. The presence of isochronic forks leads to more efficient circuits but these are no longer completely delay insensitive. The design style is therefore called quasi delay insensitive (QDI) [42].

In an isochronic fork, a transition is not acknowledged by both outputs of the fork but it is assumed that if one output is acknowledged the other output is implicitly also acknowledged. As a result these forks do not have the acknowledge property. Likewise, circuits that contain these forks only partially have the acknowledge property. In these circuits, a fault will not automatically lead to a deadlock situation but can instead also lead to a misbehaving circuit. In such a circuit a gate with a fault may change its value before it is supposed to do so. This situation is called a premature firing of the gate [29].

In Figure 2.4 the difference between an isochronic fork and a non-isochronic fork

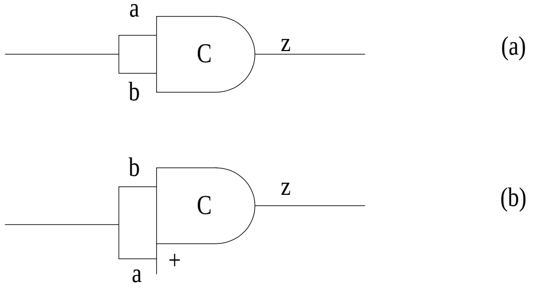


Figure 2.4: Forks connected to a symmetric C-element (a) and an asymmetric C-element (b). A stuck-at fault on the fork before gate (a) prevents any output transition, this fork is non-isochronic. A stuck-at-1 fault on input a of the fork before gate (b) results in a modified behavior, this fork is isochronic.

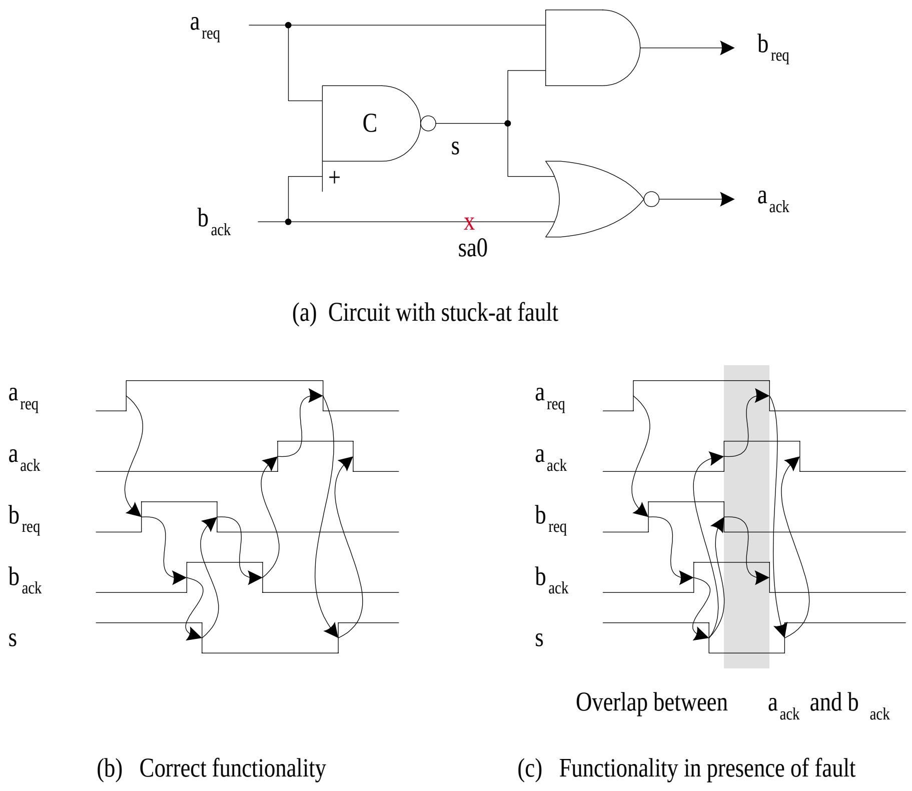

a req
b req
C
s
+
a ack
b ack
x
sa0
(a) Circuit with stuck-at fault
a req
a ack
b req
b ack
s
Overlap between a ack and b ack
(b) Correct functionality
(c) Functionality in presence of fault

Figure 2.5: Modified circuit behavior because of a fault, (a) example circuit, (b) correct behavior, (c) faulty behavior.

is illustrated. Figure 2.4(a) shows a fork that connects to both inputs of a symmetric C-element. This type of C-elements switches when both inputs have the same value. When a stuck-at fault is present on one of the inputs, this input can never switch to the opposite value and therefore the output can also not switch to this value. A transition is prevented and this will eventually lead to a circuit deadlock. Since any fault on the inputs leads to a deadlock, the fork connected to the inputs of the C-element is non-isochronic. An example of an isochronic fork is shown in Figure 2.4(b), connected to an asymmetric C-element. To switch this type of C-element to zero, only the b input has to be zero and the a input can be undefined. If there is a stuck-at-1 fault on the a input of this C-element, the output of the gate can still switch. The difference with respect to a correct C-element is that this C-element switches to one when b is one, while the correct C-element only switches when both a and b are one. The stuck-at fault does not lead to a deadlock, the fork connected to the inputs of the C-element is therefore isochronic.

A larger example of a circuit that changes its behavior in the presence of a fault is shown in Figure 2.5(a). This figure shows a handshake circuit in which the fork in the b_{ack} line is an isochronic fork; the effect of b_{ack} rising is only passed on via s to b_{req} and not to a_{ack}. In the example, a stuck-at-0 fault is present at the output of the isochronic fork leading to the NOR gate. The simulation traces show the behavior of the circuit without the presence of the fault (Figure 2.5(b)) and with the fault present (Figure 2.5(c)). In the first case, the handshake on channel b is completed before the handshake on channel a is finished. In the second simulation, the falling transition of s causes two parallel events: a rising edge on a_{ack} and a falling edge on b_{req}. This causes the handshake on channel a to overlap with the handshake on channel b, leading to unpredictable results in other parts the circuit.

Faults that do not cause a deadlock can still be detected if their effect can somehow be propagated to an observable output. The fault in this example can be detected if both outputs can be directly observed. The fault leads to a different transient behavior and an event-driven evaluation is required to capture this behavior.

The presence of isochronic forks limits the usability of the acknowledgement property for QDI handshake circuits to the testing of stuck-at output faults. These faults will always prevent the output of a gate from switching and therefore lead to a deadlock situation. In practice the use of stuck-at output fault is not sufficient since there are many defects that are not represented by this model. A more detailed fault model has been proposed [53], called the isochronic transition fault model. This model also takes stuck-at input faults into account on non-isochronic forks but not on isochronic forks. This however still does not provide a sufficient fault coverage. To achieve this, the stuck-at input faults at isochronic forks also need to be tested.

2.2.2 Autonomous behavior

By definition, asynchronous circuits have no clock to synchronize the circuit operation. Instead the circuits behave highly autonomous. In handshake circuits, correct internal timing is achieved by explicit acknowledgement of the handshake signals and dimensioning of internal delays to satisfy relative timing assumptions inside the handshake components. The level of external control is very limited. Only when the circuit communicates to the external world, some influence can be exerted on the circuit. It is not possible to stop the circuit in every state and it is also not possible to single step the circuit from one state into the other. In this sense, handshake circuits do not have a global state; instead they have many local states that are only synchronized when required. The total state of a handshake circuit can only be determined by examining the state of all handshake components and handshake channels together. For example, a sequencer contains one internal state bit which, in combination with the state of the handshake channels, is used to determine the next action of the component. This happens independently of other handshake components or global signals.

The autonomous behavior of handshake circuits presents a problem for the full-scan test method since this method is cycle based. Inputs for the circuit are applied once per clock cycle and after each clock cycle the expected response is calculated and compared to the observed outputs of the circuit. If no clock is present, it is generally more time consuming to calculate how the circuit will respond to input stimuli. In addition, event-driven testers may be required to capture these responses because the signals can change at any time not just once every cycle. Since these testers are dedicated and hence expensive and in addition require complicated test programs they are not an option to test a low-cost product.

2.2.3 Sequential behavior

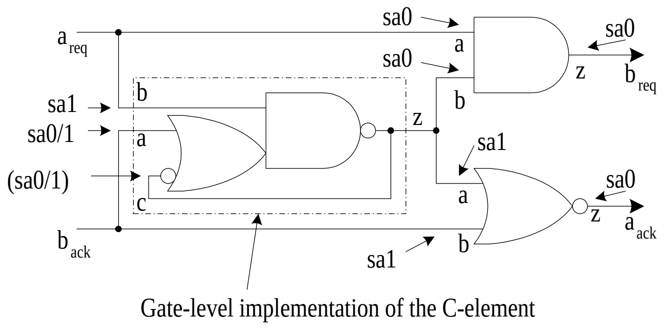
Sequential behavior is also present in synchronous circuits, but in a handshake circuit the problem is more severe. This is related to the autonomous behavior explained in the last sections, but also caused by the fact the handshake circuits use more sequential elements than the average synchronous circuit.

The control logic used a handshake circuit makes frequent use of C-elements. A C-element is a sequential element; its output is a function of both the inputs and the internal state of the element. For some input combinations the output of a C-element is determined only by the inputs. For other input combinations the output is determined by the internal state. Figure 2.6(a) shows the truth table for an asymmetric C-element. This element has one state holding state: $\{0,1\}$.

C-element

a	b	z
0	0	1
0	1	z^{-1}
1	0	1
1	1	0

(a)



(b)

Figure 2.6: Single pattern fault coverage of a circuit containing a C-element, (a) truth table of the C-element, (b) test circuit showing the untestable faults.

For testing this means that whenever a C-element is present in a circuit, this circuit can no longer be fully tested with single-pattern tests. Single-pattern tests cannot independently control the primary inputs of the circuit and setup the correct internal state. To solve this, a sequence of patterns is required where the first patterns

Table 2.1: Single-pattern fault coverage of the circuit in Figure 2.6(b), leaving many untestable faults.

		C-element				AND			NOR		
a_{req}	b_{ack}	a	b	c	z	a	b	z	a	b	z
0	0				sa0	sa1		sa1	sa0		sa1
0	1					sa1		sa1			sa1
1	0										
1	1		sa0		sa1		sa1	sa1		sa0	sa1
Untestable		sa0/1	sa1	sa0/1		sa0	sa0	sa0	sa1	sa1	sa0

are used to setup the correct internal state for the test via the primary inputs and the last pattern is used to control the value of the primary inputs during the test.

If only single-pattern tests are used, the presence of C-elements results in a limited fault coverage. This is illustrated in Figure 2.6(b), which shows the same circuit that was also used in Figure 2.5. The circuit in this example contains one C-element and therefore one internal state bit. To determine how many faults can be detected in this circuit by single-pattern tests, the C-element is first replaced by its gate-level representation, which reveals the internal feedback loop. This representation leads to less pessimistic results, since it can be seen that for three of the four input combinations of the C-element the output is a combinational function of the primary inputs only and does not depend on the internal state of the C-element.

Table 2.1 shows the four possible test patterns and the faults each pattern detects. The bottom line shows the undetectable faults. Including the faults on the feedback line of the C-element, 11 out of the total number of 20 stuck-at faults are not testable, 5 of which are located in the C-element. This results in a fault coverage of 45%. When the C-element is treated as a single gate the faults on the internal feedback line are not visible, resulting in an 50% stuck-at fault coverage. The fault coverage that is obtainable with single-pattern testing reduces further if the sequential depth, that is, the number of sequential elements in series, increases. If tests with two patterns are used, then all faults in Figure 2.6(b) are testable. This is shown in Table 2.2. The first pattern is used to set the feedback input c of the C-element. This signal is then used as a third input to the circuit.

Sequential behavior can also occur if gates are connected in a loop. As will be shown in Chapter 4, at least two situations exist in Tangram handshake circuits where such a loop might occur. When present, a circuit loop leads to the same problems as shown in the previous example. Testing a circuit with this kind of sequential behavior requires either complex multiple-pattern test generation or a hardware modification

Table 2.2: Two-pattern fault coverage of the circuit in Figure 2.6(b), all faults are now testable.

			C-element				AND			NOR		
a_{req}	b_{ack}	c	a	b	c	z	a	b	z	a	b	z
0	0	0		sa1		sa0	sa1		sa1	sa0		sa1
0	0	1				sa0	sa1		sa1	sa0		sa1
0	1	0					sa1		sa1			sa1
0	1	1					sa1		sa1			sa1
1	0	0			sa1			sa1	sa1	sa1	sa1	sa0
1	0	1	sa1		sa0		sa0	sa0	sa0	sa0		sa1
1	1	0		sa0		sa1		sa1	sa1		sa0	sa1
1	1	1	sa0	sa0		sa1		sa1	sa1		sa0	sa1

that can break all these loops temporarily during test, of course without influencing the normal asynchronous behavior.

2.2.4 Redundancy

Redundancy in a circuit reduces the number of testable faults in that circuit. In [31] it is even stated that a circuit is redundant if it contains an undetectable stuck-at fault. Therefore, for testing the definition is used that a circuit is redundant if it is not possible to obtain a 100% stuck-at fault coverage. In this sense a structure made of for example two inverters in series is not considered to be redundant for testing, since this structure is 100% stuck-at testable. When analyzed at logic level, without looking at the dynamic behavior, redundancy can always be removed by either removing a gate from the circuit or by removing an input of a gate.

In asynchronous circuits also the dynamic behavior of the circuit can be important. When the circuit switches, hazards and races can potentially change the state of the circuit [63]. Preventing these problems can lead to implementations where additional gates are added to the circuit, that would be redundant if analyzed at logic level. Such an example is shown in Figure 2.7.

In this circuit the term $a{\cdot}b$ has been added to prevent a hazard if c is changed while a and b are true. The result is that the stuck-at-0 faults at the inputs a and b of the top AND-gate in Figure 2.7 are untestable. The redundancy cannot be removed without altering the safe operation of the circuit. Hazards can cause multiple transitions on the outputs of a combinational circuit before they settle in the final state. In asynchronous

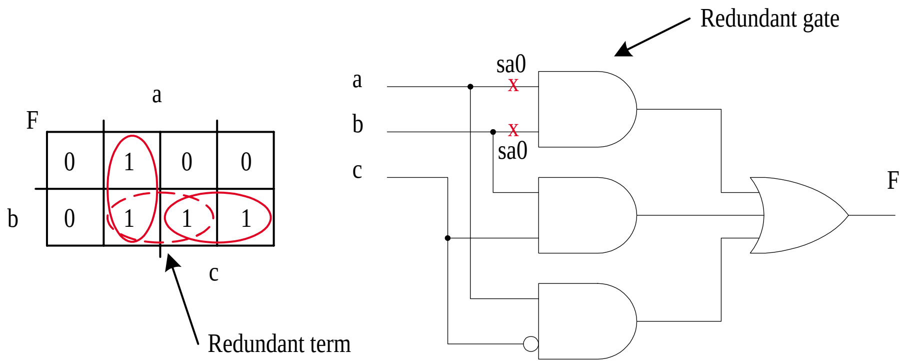


Figure 2.7: Redundancy added to prevent hazards in the circuit.

circuits these intermediate results can be falsely interpreted by the following circuits and lead to wrong results.

When testing a circuit that contains redundancy to prevent against hazards and races, no differentiation can be made between correct hazard-free circuits and circuits that have a fault present but are still functionally correct only no longer hazard free. One method to test these faults is by using variable phase splitting [36]. With this method, normal and negated versions of the same input of a combinational circuit are controlled independently during test. This increases the control over the circuit and makes it possible to also test those gates that are redundant in normal mode. The disadvantage is that it requires special and expensive scan cells that can independently control two signals instead of the normal one.

Handshake circuits are constructed by connecting handshake components chosen from a relatively small set of different types. This makes it possible to analyze these components thoroughly and to choose implementations that are free of redundancy. As will be shown later in Section 4.3.5, it is still possible that redundancy occurs in more complex handshake circuits.

2.2.5 Arbitration leading to non-determinism

In an asynchronous circuit it is possible that a single resource has to be shared by two or more parties. It is important to always guarantee that only one party is given access. The second has to wait until the first one is finished. The problems requires the synchronization of two unrelated signals, which can result in the problem of metastability [35]. Metastability results in an undefined state at a signal line and it can take arbitrary time for the signal to go back to a defined (one or zero) state.

In asynchronous circuits, a mutex is used to prevent such an undefined signal from entering the circuit. A mutex can be implemented in a number of ways, but in Tangram the implementation shown in Figure 2.8 is used. The heart of the circuit

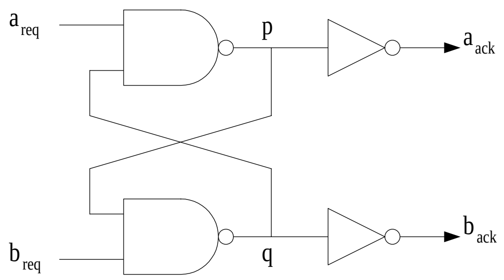


Figure 2.8: A possible gate-level mutex implementation.

is a pair of cross-coupled NAND gates. When both inputs rise simultaneously, the output of the NAND gates remains above the threshold voltage of the output inverter structure until the metastability is resolved.

Figure 2.9 shows a simulation in which the two inputs change simultaneously from zero to one. This leads to an undefined state of the internal signals p and q. The outputs a_{ack} and b_{ack} however, remain zero until internally the metastability has been resolved and it is known which of the two outputs must change to one.

The mutex has several testing problems. First of all, the pair of cross-coupled NAND gates makes it a sequential element. More fundamentally however, the mutex

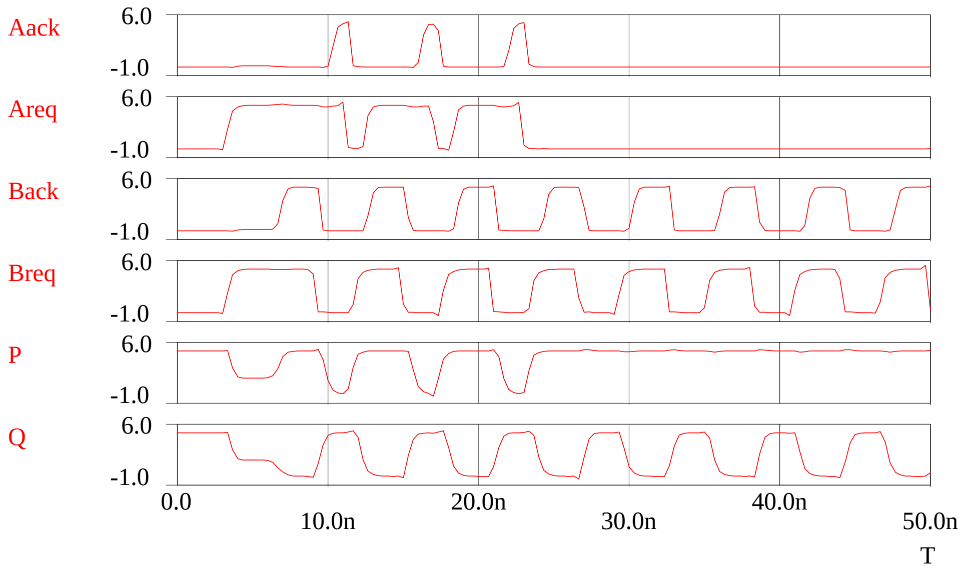


Figure 2.9: Simulation in which metastability occurs on the two internal nodes P and Q between 4 and 6ns. The outputs remain unchanged until the metastability is resolved.

leads to non-deterministic behavior during test. Whenever the pattern $\{a_{req}, b_{req}\}$ = $\{1,1\}$ is applied to the inputs of the mutex, the output cannot be predicted. It can go to either the $\{0,1\}$ state or the $\{1,0\}$ state. This makes it impossible for a test-pattern generation tool to calculate the expected response and it is therefore impossible to verify to correctness of the responses of the circuit. If a test is developed manually, the $\{1,1\}$ combination can be avoided to prevent this problem. It is possible that this leads to a reduction of the maximum fault coverage. With automatic test-pattern generation this is more difficult and all input combinations most likely will occur in the test, for example to test the environment of the mutex.

A final problem is associated with one of the possible implementations for the output inverters of the element. To function correctly, the threshold voltage of these inverters has to be above the metastable voltage at the internal nodes. Unless specially designed skewed inverters can be used that have a higher threshold voltage, the implementation uses NOR gates with the inputs connected together. This will behave similar to the skewed inverters. The structure is, however, fundamentally not fully stuck-at testable, since it is inherently redundant. A similar structure is found in the delay elements that are used in the latch controllers. Those delay chains are made out of alternating NOR and NAND gates, because this way the delay reacts the same as the logic on temperature and supply voltage variations. The only way to improve the stuck-at testability of this type of elements is to implement them as primitive library cells. How faults are counted for these redundant structures depends on the used test standard. In some cases these faults are allowed and do not influence the final fault coverage.

2.3 Handshake circuit test methods

Many methods have already been proposed to simplify and enhance the testing of handshake circuits [68, 49, 52]. All methods that have been considered in the Tangram project to test handshake circuits are based on functional test principles. As such, the methods are primarily focused on finding test solutions with a minimal area overhead, at the expense of the other cost factors described in Section 1.2.4. As a result this has led to methods that proved cumbersome in practical use and limited in their obtainable fault coverage.

2.3.1 Functional testing

Most of the designs made with Tangram are tested functionally. Creating these functional tests is a very labor-intensive task and the tests typically offer only a limited fault coverage. To assist in the development of functional tests, a test-evaluation tool was developed, described in [68]. This tool evaluates the *structural* test coverage (controllability and observability) of *functional* traces. The tool works with handshake level simulations, making the simulation very fast. It also provides direct feedback to the Tangram source code, highlighting code that has been insufficiently tested.

Another thing that eased the problem of testability of many Tangram based ICs is the fact that many of these ICs are build around the 80c51 micro-controller [24]. These ICs could all be tested with the same functional test that had been developed with the help of the test-evaluation tool. Still, it required a lot of work to generate these patterns and the final stuck-at fault coverage was estimated to be only 90% [63]. For many new products, the functional tests will be more complex and therefore it is not feasible anymore to develop functional tests for these products.

2.3.2 Programming scan in the Tangram program

Some circuits contain structures that are very difficult or time-consuming to test with functional tests. Especially, large counters require excessive test times since often exhaustive testing is required. For example a 16 bit counter that is exhaustively tested requires $2^{16} = 65536$ patterns. A method that can provide direct access from the primary inputs to such structures can significantly reduce the test time.

This type of modification to the Tangram source code makes it easier to design a functional test for those parts of the circuit that are otherwise very difficu In [49], an address generator is made testable by directly programming a scan access mechanism in the Tangram source code. The example used in the paper is shown in Figure 2.10. It is a test developed for a DCC error decoder [9] that contains an embedded address generator. The address generator is made accessible by adding the code shown in Figure 2.10 to the Tangram program. It defines a new function `scanin()` that is called whenever a new address has to be loaded into the address register. The function uses a “`for`” loop to shift 14 bits from the input $scanbit$ into the address register $Address$.

lt to test effectively. However, the test remains a functional test with the associated labor-intensive development and low fault coverage. Both adding the test modifications and the test generation have to be carried out by hand. So while making it easier to generate a more effective test, it retains the disadvantages of functional testing.

2.3.3 The hold method

A significant step towards a structural test method was introduced with the hold method, described in [50, 52]. The method exploits the fact that the activity in a handshake circuit always starts at the startup channel and ends at a passive component, such as a variable or a passivator, as explained in Section 1.1.2.

One of the problems associated with handshake circuits, is the autonomous operation of the circuits, which limits its controllability and observability. The hold

```
scanin : proc(scanchan?(0..1))

begin scanbit : var(0..1)
| for 14 do
    scanchan?scanbit;
    Address := <scanbit,Address.0,...,Address.12>
  od
end
```

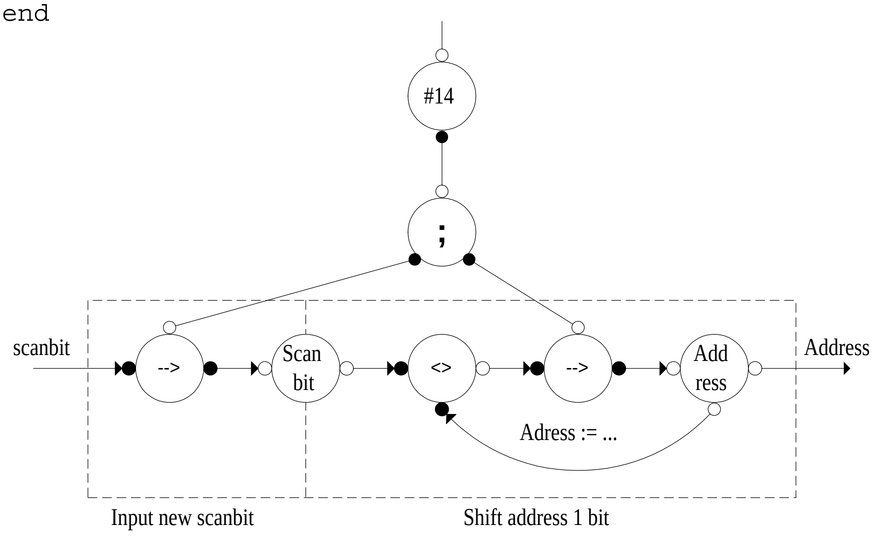


Figure 2.10: Scan function programmed in Tangram. The function scanin is called whenever a new address has to be loaded into the address register.

method is based on inserting hold components in the channel before every passive component. A hold component can stop the handshake in a channel in both the rising and the falling phase. By stopping the handshake in a channel, all connected channels are also stopped, ultimately resulting in a totally quiescent circuit. This quiescent state can be used for testing, both with Iddq and voltage-based tests.

Figure 2.11 shows a hold component inserted in a handshake circuit. The hold component is controlled by three global control signals $h1 - h3$. With these signals it is possible to stop (hold) the handshakes over the channel in both phases. In addition, the component can be used to directly clock the variable x via the $h1$ signal. The approach uses three methods to detect faults in the circuit:

- Deadlock detection: detects faults that cause a handshake to deadlock. The faults are detected if an expected response does not occur before a timeout.

- Data testing: detects faults that generate incorrect clock signals for the data path resulting in faulty data in the data path. This method uses scan testing of the data path.

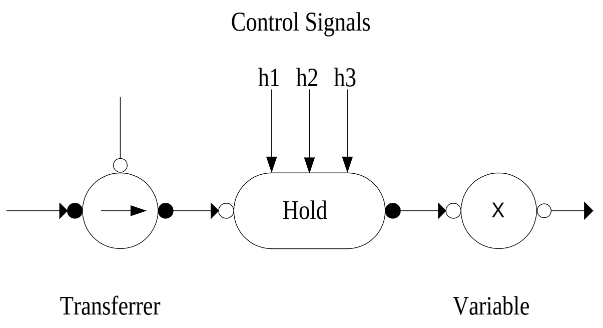


Figure 2.11: Hold component inserted in handshake circuit between a transferrer and a variable.

- Iddq testing: this is the main test to detect faults in the control logic.

Because deadlock detection and data testing cannot detect all faults in the control logic, the Iddq measurements are essential to obtain a high fault coverage. The Iddq measurements are carried out in both the rising and falling phase every time a hold component is activated. This results in a high number of Iddq tests that have to be carried out, thereby greatly increasing the test time. One advantage of the Iddq tests is that they are very effective at capturing bridging faults between request and acknowledge lines.

The gate-level implementation of the hold component is shown in Figure 2.12. In normal asynchronous operation, the element is transparent by setting the signals $h1$

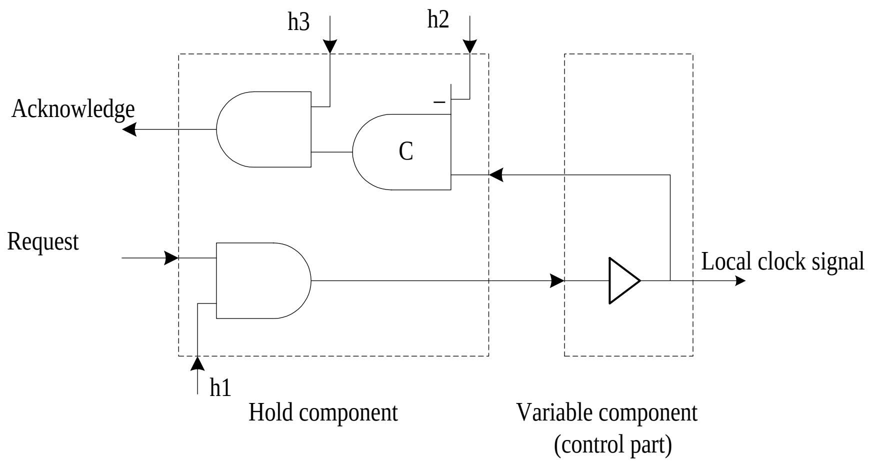


Figure 2.12: Implementation of the Hold element connected to the control part of a variable component to show how the local clock signal can be controlled by signal $h1$.

and $h3$ high and $h2$ low. In test mode, initially all the three control signals ($h1 - h3$) are low. If the component is activated through the request line the activity stops at the first AND gate. The circuit will now enter a quiescent state in which an Iddq test can be carried out. By toggling control line h1, the local clock for the data path can be operated. This is used to facilitate a scan test for the data path. After the data path test, the other control lines (h2 and h3) are raised which changes the acknowledge signal to logic one. During the "down" phase, all three control signals are lowered, starting with $h1$. In this state again an Iddq test is executed. Finally $h2$ and $h3$ are lowered to complete the handshake.

The data path is tested with a scan method as described in [51]. The method exploits the fact that the hold method allows the individual variables to be clocked separately. Only during the scan-in and scan-out phases a global clock is applied. In the evaluation phase only an individual variable is clocked using the hold elements. This allows pipelining of patterns in the scan chain, which reduces the test time. Additionally this removes the need for slave latches resulting in a latch based scan method based on L_1L_2* scan [16].

The hold method can result in a high test quality, but depends on a large number of time-consuming Iddq tests to obtain this. Additionally the test patterns for the control block are functional patterns that need to be provided by the designer. This therefore does little to reduce the manual test-development work. Only the data path test, which is scan based, can be largely automated. To conclude, the hold method uses little area for DfT and can achieve better fault coverage than functional testing. However, this is achieved with only a limited reduction in the test-pattern generation effort.

2.4 Scan test in asynchronous circuits

The scan test method is the default method to test synchronous circuits. One reason for this is the relatively inexpensive implementation. More importantly, scan test simplifies the complex sequential ATPG problem into the well-known combinational ATPG problem. For this problem many algorithms have been proposed [22, 26, 54], which have led to powerful ATPG tools.

The successful application in synchronous circuit has motivated a number of applications of scan test in asynchronous circuit. Applying scan in asynchronous circuits is more problematic for two main reasons. First, asynchronous circuits tend to use more state-holding elements than synchronous circuits. Second simply adding a scan multiplexer to every register is not sufficient. In addition a control mechanism is required as an alternative for the clock that is used to control the scan elements in a synchronous circuit. There are two options for the control mechanism: asynchronous control and synchronous control. In the next sections the two control mechanisms are discussed and applied in several applications. Because of the larger area requirements of an asynchronous scan method, scan has mostly been limited to those parts of the asynchronous circuit that have the most similarity with synchronous circuits. This means that the proposed methods focus on the data path or on the exploitation of a specific feature of a specific design style.

Asynchronous control

In an asynchronously controlled scan chain every data transfer from scan-register to scan-register requires a handshake in the control logic. The control logic is equal to the control logic of a FIFO buffer. This makes the implementation best suited for a circuit that already contains many FIFO buffers or pipeline stages. If this is not the case large and often complex control circuitry is required. The control circuitry itself is only partially tested during the scan test and additional functional tests are required to obtain full fault coverage.

This type of scan test control has been applied several times to the micro-pipeline [64] design style. Since the control logic used in this style is already very similar to the logic required for FIFO operation, the required modifications are limited. Several methods have been proposed to apply scan testing with asynchronous control to the micro-pipeline style of circuits [34, 47, 46, 55]. All methods are designed to support the test of the logic located between the pipeline registers. The registers themselves and the remaining control logic has to be tested implicitly.

Synchronous control

With a synchronous control mechanism, the scan chain is controlled by a clock signal. Since a clock is not present in an asynchronous circuit, a clock signal has to be added to the circuit and the sequential elements have to be modified to make them react on the clock. The main advantage is that the circuit during test behaves the same as a synchronous circuit. This can be exploited by using a large part of the existing test tools and test infrastructure that is in place for synchronous circuits.

The synchronous full-scan concept was applied to an asynchronous circuit design style that uses SR-latches as its sequential elements [59, 67]. The SR-latches were modified to operate as LSSD style registers. The method uses four global control signals, one of which can be used to force the $\{1,1\}$ state on the S and R inputs of the elements. The other three signals are uses to clock the LSSD registers. The work focuses on proving that a scan method can reliably detect faults in the circuit, even when the faults cause hazards making them very difficult to detect otherwise.

2.5 Summary

Handshake circuits have many characteristics that make them difficult to test. They contain virtually all constructions that are prohibited in synchronous circuits because they are difficult to test. For this reason alone, testing of a handshake circuit will probably never become as efficient as testing of a synchronous circuit.

None of the test methods for handshake circuits described so far have supported automatic test generation, neither by existing tools nor by making test generation simpler to make the development of a new tool feasible. The maximum that is available is automatic test generation for those parts of the circuit that have been made scannable.

The challenge addressed in this thesis is to develop a test method that covers the entire circuit and offers automatic test pattern generation. The quality should be equal to the quality offered by synchronous test methods. A lot of research has been put into ATPG tools, leading to complex but powerful tools. Because of their complexity it is impractical to develop new ATPG tools, except for extremely simple ones. Therefore one of the goals is to reuse or adapt existing ATPG and other tools were possible instead of developing new ones.

Current commercial ATPG tools are heavily oriented towards full-scan test. Therefore a scan-based method will have the best support for test generation. To keep the cost down and support simple integration with other modules on the chip, a synchronous clocked scan chain seems to be the best option. Even though the area overhead can be expected to be high, the other factors like simple ATPG, system integration and test development make scan-testing a promising test solution for handshake circuits. In the following chapters a complete scan test method for handshake circuits is introduced and its costs and benefits are evaluated.

Chapter 3

Scan testing of handshake control circuits

Chapter 2 ended with the conclusion that a synchronously controlled scan test method offers the best solution to meet the objectives described in Chapter 1. This chapter investigates the implementation options and applies these to the control block of a handshake circuit. Later, in Chapter 4 the entire handshake circuit is made scannable.

3.1 Synchronous scan

In a scan method all the sequential elements in the circuit are modified into scan elements that can capture data on a second input. This is schematically shown in Figure 3.1. Internally every scan element contains two sequential elements: the master and the slave.

The master element can capture a data value on either one of its two input ports, the normal data input d or the scan data input TDI. The slave element keeps the output of the scan element stable while the master element captures new data. The operation of the scan element is controlled by a set of control signals. These control signals control both normal circuit operation as well as the operation during scan test.

In a synchronous scan method, the control signals are controlled by global signals that are a combination of select signals and clock signals. These signals are controlled by primary inputs, either directly or via some control logic. Different encodings of the control signals are possible, leading to different scan approaches. Scan elements that use different control signal encodings can be used together, however this can lead to timing constraints that should be respected for safe operation. These timing conditions are discussed in the last section of this chapter.

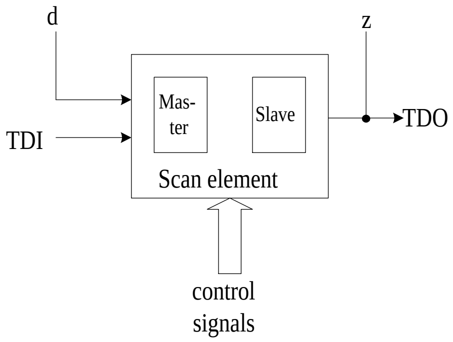


Figure 3.1: General structure of a scan element, consisting of a master and a slave element controlled by some control signals.

3.1.1 Scan approaches

There are two characteristics that can be used to distinguish alternative scan approaches. Every alternative requires its own style of scan elements.

The first characteristic is the choice of clocking strategy, there are two options:

Edge triggered clocking A single clock signal is distributed over the circuit. The scan element will internally generate two separate clock signals for its master and slave elements. The effect is that the output of the element changes on an edge in the clock signal. Usually the element reacts to the rising edge of the clock and a clock period is defined to last form one rising edge to the next.

Level sensitive clocking Two clock signals are distributed, one is exclusively used to clock the master elements, the other exclusively to clock the slave elements. The use of two independent clock signals, gives more control over the opening and closing of the internal sequential elements. Besides the normal behavior where one element is closed and one open, it also allows for both element to be closed or opened simultaneously. The first can be used to make data transfer safe, the second can be used to make the element transparent. To obtain the same behavior as an positive edge-triggered clock, the master clock should be inverted with respect to the edge-triggered clock. The slave clock has the same polarity as the edge-triggered clock.

The second characteristic is the choice of multiplexer implementation used to select between normal data input and scan data input. Again there are two options:

Data multiplexing The input is selected by a normal multiplexer, controlled by a select signal. The select signal is a data input with is stable for a complete clock cycle.

Clock multiplexing Both the test data input and the normal data input have a separate clock to control them. By clocking on one of these clocks, data from the corresponding input is captured.

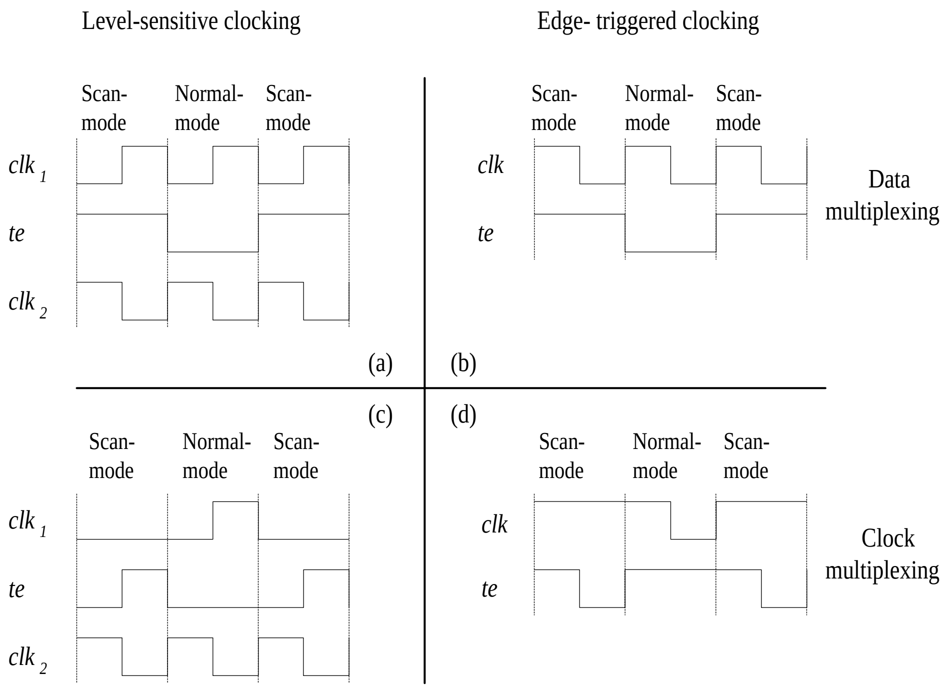


Figure 3.2: Control signal timing for the four alternative control signal encodings: level sensitive clocking (a, c), edge triggered clocking (b, d), combined with data multiplexing (a, b) and clock multiplexing (c, d).

The two characteristics lead to four alternative scan approaches. The timing of the control signals used for these options is shown in Figure 3.2. Two of the four options are well known and often used. These are the combination of level sensitive clocking with clock multiplexing, resulting in the LSSD scan method. The other frequently used scan method is the edge-triggered Mux-D variant, using edge-triggered clocks and data multiplexing. Both of these will be used in this thesis. The variant of level sensitive clocking with data multiplexing will also be used. Only the combination of edge triggered clocking with clock multiplexing is not used in this thesis. As will be explained in the next section, all new scan elements will use level-sensitive clocking and existing edge-triggered element all use data multiplexing. Details of an edge-triggered clock-multiplexing strategy can be found in [23].

3.1.2 Synchronous scan for handshake control circuits

Two fundamental test problems of handshake circuits are the autonomous and sequential behavior as described in Section 2.2.2 and Section 2.2.3. These two related problems make it especially difficult to test the handshake control block of a handshake circuit. Both the autonomous and sequential behavior can be removed during test by implementing full scan in the control block.

Handshake control circuits can be viewed as combinational logic circuits with feedback loops. These feedback loops introduce sequential behavior in the circuit. To apply scan to these circuits, all sequential behavior has to be removed by breaking the feedback loops with scan elements. Two types of loops can be identified. The first type is internal to a C-element and provides the C-element with its internal state. The second type of loop can span multiple gates. This makes the set of wires that have to be broken by scan not unique. A loop can span multiple wires and only one of them has to be broken. In those situations, the location to break the loop can be chosen freely.

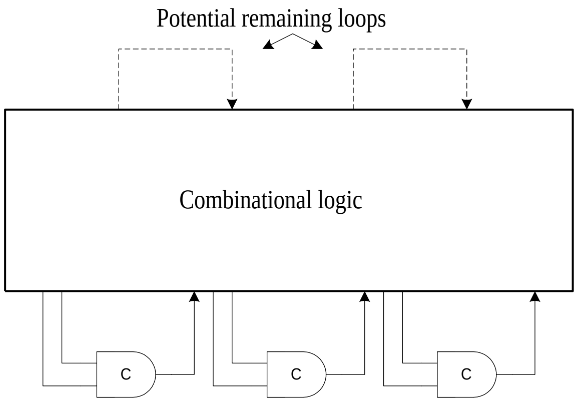


Figure 3.3: Handshake control with the C-elements and possible remaining loops shown separated from the combinational logic.

Because every C-element has an internal feedback loop, a minimum requirement is that C-elements are modified such that these internal loops are broken. A consequence of this is that all loops that contain a C-element are also broken. The remaining circuit now may or may not have any additional loops that still need to be broken. This situation is shown in Figure 3.3. It shows the combinational logic block with the C-elements pulled out at the bottom and possible remaining loops pulled out at the top. The initial test approach is to only break all the loops inside the C-elements. Any remaining loop will be broken with an additional scan elements, as will be described in Chapter 4. These remaining loops also include the loops internal to mutex elements, that can be either disabled, thereby reducing the fault coverage or scanned, similar to the approach used for C-elements.

The loops inside the C-elements could be removed by inserting transparent scan flip-flop in the loop. In this chapter, a more efficient solution is introduced in the form of clockable and scannable C-elements. The scan function that is added to the C-elements needs to support a transparent mode, in which the C-elements behaves as an original unmodified element. This can be implemented with an edge-triggered element but that would require an additional multiplexer to put it in a transparent mode. This multiplexer itself is not test during the scan test. Therefore all scan C-elements shown in this chapter use level-sensitive clocking. This clocking strategies requires two separate clocks. In order to be able to use these clocks without requiring additional external pins, an on-chip clock generator is shown in Section 3.4.4 that is able to generate all the internal clocks from a single external reference clock.

3.2 Logic gates

In this section an overview is given of the logic gates that are used in handshake circuits. Gates can be either primitive gates or composite gates. Primitive gates are those gates that are directly available in a chosen technology library and are implemented at transistor level. Composite gates are not available in the chosen library have to be constructed out of the primitive gates that are available.

The gates are specified in two ways. First by so called "production rules" [42]. Second, by Boolean logic equations that are derived from the production rules. These Boolean logic equations describe a gate in terms of simple logic functions, which can be used to implemented the gate as a composite gate. For a number of common gates, primitive implementations at the transistor-level are also given.

3.2.1 Production rules

A logic gate can be specified elegantly by a pair of production rules. Production rules consist of a set of guarded commands of the form:

$$\begin{aligned} U &\rightarrow z\uparrow \\ D &\rightarrow z\downarrow \end{aligned} \tag{3.1}$$

This specifies a gate with output z. The inputs of the gate are used to form two Boolean expressions U and D, that function as the up and down guards of this output. If the up-guard U is true, then the output will become true and if the down-guard D is true, then the output will become false. The two guards may not be true at the same time.

The production rule specification can be directly used to derive a Boolean equation by substituting the up and down guard in Equation 3.2

$$z := U + z \cdot \overline{D} \tag{3.2}$$

Production rules can be used to describe both combinational and sequential gates. If an input combination is possible where neither of the guards is true, the specification is sequential, otherwise it is combinational.

3.2.2 Combinational gates

For a gate to be combinational, for all input combinations one of the guards has to be true. This leads to the requirement that $U = \overline{D}$, or whenever one guard is true the other is false and vice-versa. The Boolean specification is now given by:

$$z := U + z \cdot \overline{D} = U \tag{3.3}$$

For example an AND gate is specified by:

$$\begin{array}{rcl} a \cdot b & \rightarrow & z\uparrow \\ \overline{a} + \overline{b} & \rightarrow & z\downarrow \end{array} \tag{3.4}$$

This corresponding equation, as expected, is the AND function:

$$z := a \cdot b + z \cdot \overline{(\overline{a} + \overline{b})} = a \cdot b \tag{3.5}$$

Most technology libraries contain a large number of primitive combinational gates. Often all of the gates with up to four inputs are present, along with many of gates that have more than four inputs. Efficient logic optimizers are commercially available and can be used to map larger combinational functions on to the primitive gates in the most cost-effective way.

3.2.3 Sequential gates

All other gates that can be specified with production rules are sequential, meaning that for at least some input combinations the output value is not determined by the inputs. Solving Equation 3.2 for a sequential specification results in a logic function F, which is a function of both its inputs and its current output value. Every sequential gate can be realized by a combinational function F with its output fed back to one of the inputs [12], as illustrated in Figure 3.4. For correct operation, it is required that the delay of the feedback loop is smaller than the delay of any other loop from the output of the gate back to an input. This type of requirement is an asymmetric isochronic fork requirement, labelled with a $<$ symbol in Figure 3.4.

A well known example in this category is the D-latch specified by:

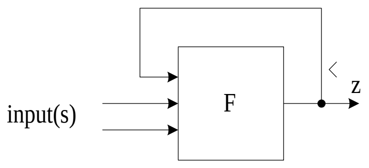
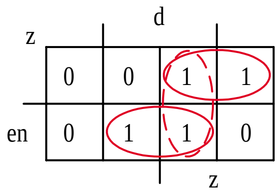


Figure 3.4: Generic implementation of a sequential gate.

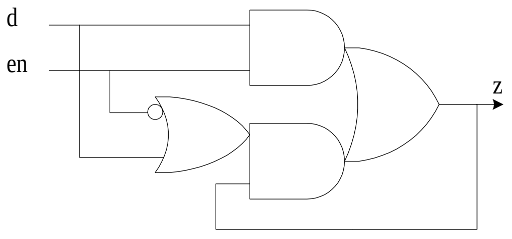
$$\begin{aligned} d \cdot en &\rightarrow z\uparrow \\ \overline{d} \cdot en &\rightarrow z\downarrow \end{aligned} \tag{3.6}$$

$$z := d \cdot en + z \cdot \overline{(\overline{d} \cdot en)} = d \cdot en + z \cdot d + z \cdot \overline{en}$$

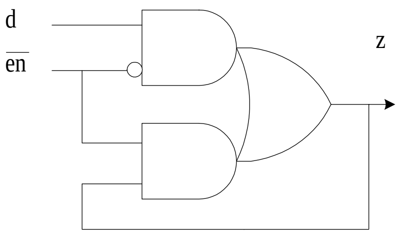
The Karnaugh diagram for this latch is shown in Figure 3.5(a), and its composite implementation is shown in Figure 3.5(b). This latch is known as the hazard-free polarity-hold latch [19].



(a)



(b)



(c)

Figure 3.5: Gate-level implementation of a latch, (a) Karnaugh diagram, (b) Hazard-free implementation, (c) Optimized implementation with inverted enable input.

The term $z \cdot d$ is logically redundant in this implementation, but can be used to guarantee hazard-free operation. The potential hazard can occur if both d and z are one and the enable signal en changes. This can potentially lead to the destruction of the internal state of the latch if en changes from one to zero. If this falling transition disables the top AND gate before the bottom AND gate is enabled, a zero can appear on the output of the latch which can feedback into the bottom AND gate. The hazard can be avoided if an implementation is used that first enables the bottom AND gate before disabling the top AND gate. In case of the latch is can be done by using the inverse of the enable signal to control the latch. This means that the potential hazard cannot occur and that the specification can be further optimized into Equation 3.7. The implementation of this latch is shown in Figure 3.5(c). An alternative implementations is to use a separate gate for the top AND and integrate the bottom AND with the NOR gate, since this also ensures that the bottom gate is faster.

$$z := d \cdot en + z \cdot \overline{en} \tag{3.7}$$

Both latch implementations shown are composite-gate implementations. Latches are very common in cell libraries, and will almost always be present in the form of primitive gates.

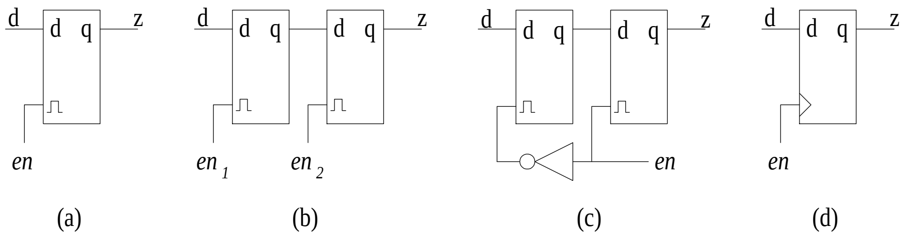


Figure 3.6: D-type sequential gates, (a) Latch, (b) Level-sensitive flip-flop, (c) Edge triggered flip-flop and (d) its symbol.

Two other important sequential gates are directly derived from the latch. These are the level-sensitive flip-flop and the edge-triggered flip-flop. The symbols of all these gates are shown in Figure 3.6. A single latch is shown Figure 3.6(a) and a level-sensitive flip-flop using two basic latches and driven by two independent enable signals is shown in Figure 3.6(b). As will be explained later in Section 3.4.1 this allows a very robust timing. An edge-triggered flip-flop, shown in Figure 3.6(c) and Figure 3.6(d), uses an internal inverter to generate the second enable signal from the external enable signal. This has the benefit of using only one global signal. This clocking method is the subject of Section 3.4.2.

3.2.4 C-elements

A C-element is a generic form of a set-reset latch, its most distinctive feature being the lack of an enable signal. The basic form has two inputs a and b that both need to have the same value in order for the output z to change to that same value, as specified in Equation 3.8. Since the function is symmetric in both inputs, this version is called a symmetric C-element. Figure 3.7 shows its symbol (a) and a possible composite gate implementation (b).

$$\begin{aligned} a \cdot b &\rightarrow z\uparrow \\ \overline{a} \cdot \overline{b} &\rightarrow z\downarrow \end{aligned} \tag{3.8}$$
$$z := a \cdot b + z \cdot (a + b)$$

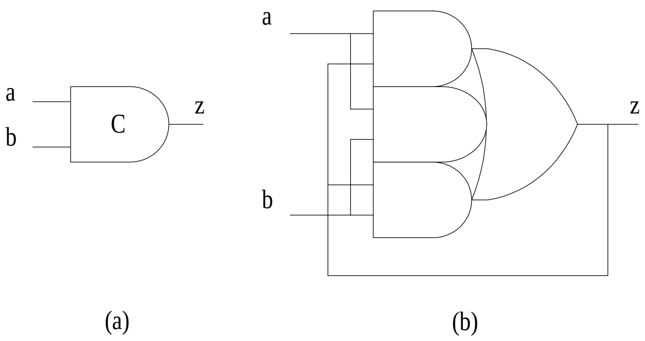


Figure 3.7: Example of a symmetric C-element, (a) symbol and (b) implementation.

From the basic symmetric C-element, a family of gates can be derived. Variations exist with more than two inputs, inverted outputs and asymmetric guard functions. An example of this last variation is given below, in Equation 3.9. In this case the up-guard is b, therefore whenever b is high, the output also becomes high. The down-guard in this example remains the same as the reset guard in the symmetric C-element. For the output to go back to zero, both inputs have to be zero. This type of C-element is called an asymmetric C-element.

$$\begin{aligned} b &\rightarrow z\uparrow \\ \overline{a} \cdot \overline{b} &\rightarrow z\downarrow \end{aligned} \tag{3.9}$$
$$z := b + z \cdot (a + b) = b + z \cdot a$$

Figure 3.8 shows the symbol (a) and a possible implementation (b) of this element. In the symbol, input a is labelled with a minus sign, this means that this input is only part of the down-guard and not of the up-guard.

Besides the composite implementations shown so far, many other implementations exist. Figure 3.9 shows possible primitive transistor-level implementations of the two example C-elements. Figure 3.9(a) shows the symmetric C-element. This element functions as an AND if z is zero and as a NOR if z is one, both functions can

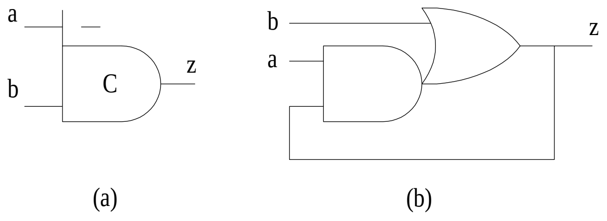


Figure 3.8: Example of an asymmetrical C-element, (a) symbol and (b) implementation.

be recognized in the implementation. Figure 3.9(b) shows the asymmetric C-element. The three transistors on the left implement the up and down guard of the C-element, whereas the transistors on the right keep the internal state valid when neither of the guards is true.

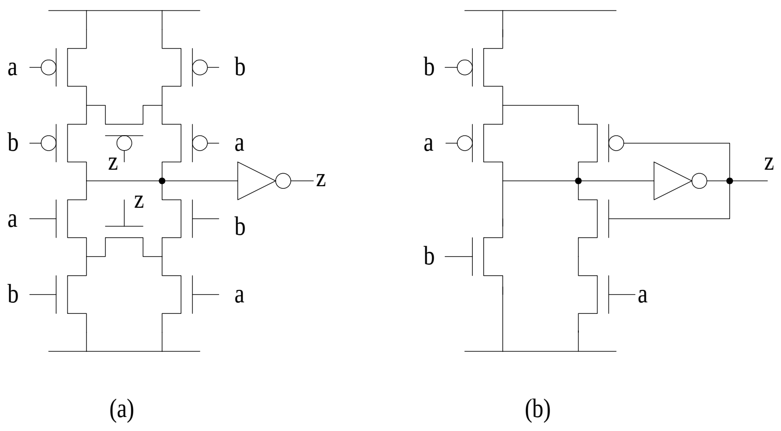


Figure 3.9: Symmetric (a) and asymmetric (b) primitive C-element implementations.

3.2.5 Enabled logic gates

Any logic gate, either combinational or sequential, can be equipped with an enable input. Such an input can be used to enable or disable the gate. When a gate is disabled, it will not respond to changes on its data inputs. The enable signal is added to a gate by taking the original production rules of a gate and add the enable signal en in both the up and down guard as shown in Equation 3.10. The resulting gate is always sequential.

$$\begin{array}{rcl} U \cdot en & \rightarrow & z\uparrow \\ D \cdot en & \rightarrow & z\downarrow \end{array} \tag{3.10}$$

The enable function is symbolized by a line drawn across the original symbol with a name beside it indicating the enable signal. Figure 3.10 shows the symbol used for a function F and for the enabled version of F.

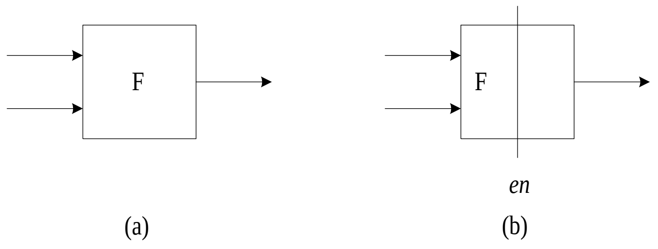


Figure 3.10: Symbols representing the function F (a) and the enabled variant of F (b).

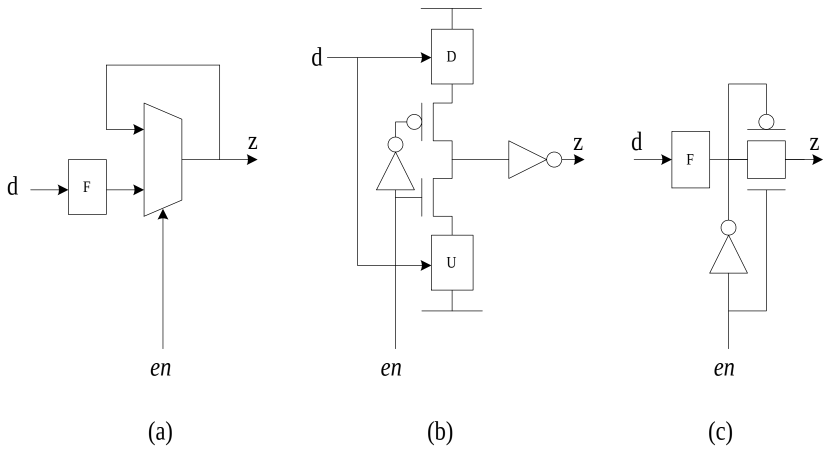


Figure 3.11: Three implementations to add an enable signal to a logic gates, (a) a multiplexer, (b) tristate transistors, (c) a transmission gate.

The enable function can be implemented in various ways. Figure 3.11 shows three (out of many) alternatives. In Figure 3.11(a), the enabling is implemented with a multiplexer. This results in a fully static circuit, equal to the circuit that would be obtained by deriving an equation from the production rules. Function F can be any function derived from production rules. The implementation in Figure 3.11(b) uses tristate transistors to implement the enable function and the implementation in Figure 3.11(c) uses a transmission gate. Both of these implementations are dynamic, since a floating node is used to store the state. This has the advantage of being more area efficient, but the resulting circuits are at the same time less resistant to noise and cannot hold their state for a long time because of leakage currents.

3.3 Scannable logic gates

The scan method for handshake control circuits relies on the use of clockable and scannable C-elements. For both the clocking and scanning functions, several implementation alternatives exist. The next section introduces multiplexing that together with the enabling of gates from the ingredients that are used to create clocked scan gates. This is followed by the implementation of scan C-elements in both composite and primitive variations.

3.3.1 Multiplexing the scan input

In a scan element a scan input has to be multiplexed with the normal data input. The active input is selected by a test enable signal. Instead of adding a separate multiplexer to the circuit for every sequential element, the multiplexer is usually integrated into the sequential element. In this way, the impact on speed and area is minimized. Scan versions of the latch and the flip-flop implemented in this way are well known and available in almost all standard cell libraries. There are two possible implementations for the multiplexers, mentioned before in Section 3.1.1, in this thesis referred to as data multiplexing and and clock multiplexing.

- Data multiplexing

$$\begin{aligned} (d \cdot \overline{te} + ti \cdot te) \cdot en &\rightarrow z\uparrow \\ (\overline{d} \cdot \overline{te} + \overline{ti} \cdot te) \cdot en &\rightarrow z\downarrow \end{aligned} \tag{3.11}$$

- Clock multiplexing

$$\begin{aligned} d \cdot en + ti \cdot te &\rightarrow z\uparrow \\ \overline{d} \cdot en + \overline{ti} \cdot te &\rightarrow z\downarrow \end{aligned} \tag{3.12}$$

Equation 3.11 and Equation 3.12 show the production rule specification of both multiplexing alternatives applied to a latch. In both specifications a new scan input ti is created, that is used to form the scan chain. The multiplexer is controlled by the test enable signal te. With data multiplexing, a normal multiplexer is added in front of the latch function [2], as shown in implementation in Figure 3.12(a). Clock multiplexing is implemented with two tristate latches. It uses the latch enable signals to select which latch is active. The implementation is shown in Figure 3.12(b).

The operation of the two types of multiplexing is illustrated in the timing diagrams included in Figure 3.12. In the data multiplexing variant, test enable signal te switches at the beginning of a clock period and keeps its value during the entire clock period. In the clock multiplexing variant the te signal is operated as a clock signal. During scan-mode the circuit is clocked with the te signal and during normal-mode the circuit is clocked with the normal enable signal en. The control signals required

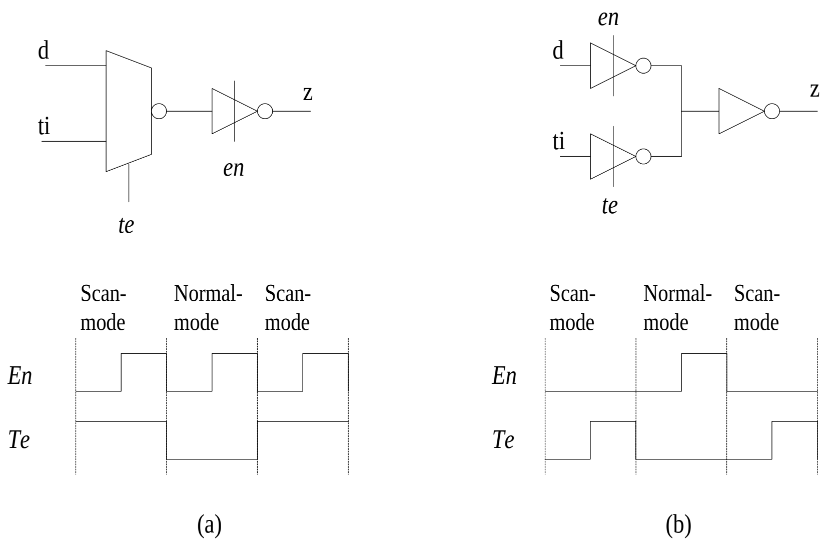


Figure 3.12: Multiplexing in a scan element and scan timing diagrams for (a) data multiplexing and, (b) clock multiplexing.

for the clock multiplexing variant can be easily generated from the signal used for data multiplexing with a single de-multiplexer, as will be shown in Chapter 4.

3.3.2 D-type scan elements

The scan version of the D-type scan element follows directly from the previous section. For edge-triggered flip-flops the data multiplexing version is used. Main reason is that this type of element is available in every technology library and usually in multiple versions.

For the level-sensitive flip-flop both data multiplexing and clock multiplexing are an option. Current implementation use data multiplexing, since these are built using latches in the technology library that use data multiplexing. The level-sensitive flip-flops themselves are not common in technology libraries and are constructed out of a scan latch and a normal latch. Another promising optimization is a primitive version of an level-sensitive flip-flop. As shown in Figure 3.6(b), the level-sensitive flip-flop consists of two back-to-back latches.

Only the master latch is used during normal asynchronous mode. This latch must be implemented in fully static logic. The slave latch is only used during scan. This slave latch can be a dynamic latch, resulting in the implementation shown in Figure 3.13.

3.3.3 Composite scan C-elements

The design of a scan C-element uses the enabled C-element as a basis. Like for latches, the scan multiplexing functionality can be added in two different ways: one leading to a data multiplexing implementation and one leading to a clock multiplexing implementation. Because scan C-element require a transparent mode, only level-sensitive clocked versions are presented. The scan C-elements use a special master latch element that is later combined with a slave latch to form a flip-flop.

Scan C-element master latch

The choice of the multiplexing method lead to the following two specifications for a scannable C-element master latch:

- Data multiplexing

$$\begin{aligned} (b \cdot \overline{te} + ti \cdot te) \cdot en &\rightarrow z\uparrow \\ (\overline{a} \cdot \overline{b} \cdot \overline{te} + \overline{ti} \cdot te) \cdot en &\rightarrow z\downarrow \end{aligned} \tag{3.13}$$

- Clock multiplexing

$$\begin{aligned} b \cdot en + ti \cdot te &\rightarrow z\uparrow \\ \overline{a} \cdot \overline{b} \cdot en + \overline{ti} \cdot te &\rightarrow z\downarrow \end{aligned} \tag{3.14}$$

The specifications are comparable to that of a latch, only the d input is replaced by the a and b inputs that implement the original function of the C-element. In the equations the asymmetric C-element is used that was specified in Equation 3.9.

For both multiplexing versions, many implementations are possible. However the clock multiplexing version leads to a more efficient implementations in terms of silicon area. By using the standard method of deriving a Boolean equation from a

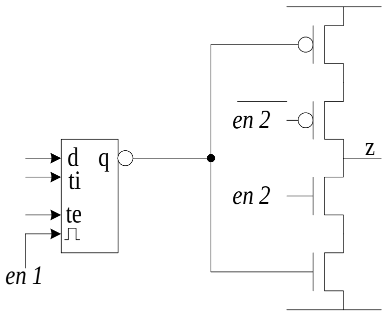


Figure 3.13: Level-sensitive flip-flop implementation with tristate slave latch.

set of production rules, the following equation is derived from the clock multiplexing style specification in Equation 3.14:

$$\begin{aligned} z &:= b \cdot en + ti \cdot te + z \cdot \overline{(\overline{a} \cdot \overline{b} \cdot en + \overline{ti} \cdot te)} \\ &= b \cdot en + ti \cdot te + z \cdot (a + b + \overline{en}) \cdot (ti + \overline{te}) \end{aligned} \tag{3.15}$$

Scan input ti and scan enable te together form a D-latch structure, similar to the latch shown in Section 3.2.3. The same optimization can be performed. During scan operation, when this latch is used, enable signal en is always zero. The logic containing this signal merely acts as an additional delay during scan mode. The optimization removes the ti term from the $(ti + \overline{te})$ factor. Like was the case for the latch, the loop delay has to be kept minimal. In the final circuit this is implemented by allowing the feedback loop to loop over only one gate. For the example, the optimization results in the following equation:

$$z := b \cdot en + ti \cdot te + z \cdot \overline{te} \cdot (a + b + \overline{en}) \tag{3.16}$$

The equation can be further optimized as shown in Equation 3.17.

$$\begin{aligned} z &:= b \cdot en + ti \cdot te + z \cdot \overline{te} \cdot (a + b + \overline{en}) \\ &\quad \left\{ \begin{array}{l} b \cdot en + x \cdot (b + \overline{en}) \equiv \\ b \cdot en + x \cdot (b \cdot en + \overline{en}) \end{array} \right\} \\ &= b \cdot en + ti \cdot te + z \cdot \overline{te} \cdot (a + b \cdot en + \overline{en}) \\ &\quad \left\{ \begin{array}{l} b \cdot en + x \cdot (b \cdot en) \equiv \\ b \cdot en \end{array} \right\} \\ &= b \cdot en + ti \cdot te + z \cdot \overline{te} \cdot (a + \overline{en}) \end{aligned} \tag{3.17}$$

The next step is to map this equation onto library cells. During this step the timing constraints that were added during the optimization have to be fulfilled. This means that the connection of the feedback loop z and the term $\overline{te}$ should be connected as close to the output of the gate as possible. This is implemented in the mapping in Equation 3.18, in which the previous mentioned signals are directly connected to the gate producing the output z. The remaining logic is grouped into a minimal number of additional gates and connected to the output gate.

$$\begin{aligned} F &:= b \cdot en + ti \cdot te \\ G &:= a + \overline{en} \\ z &:= F + z \cdot \overline{te} \cdot G \end{aligned} \tag{3.18}$$

Figure 3.14 shows the implementation of this specification. In the library used for this implementation, the functions z and G could be implemented with one cell, leading to a total of two library cells. The logic corresponding to the original C-element is shown with bold lines.

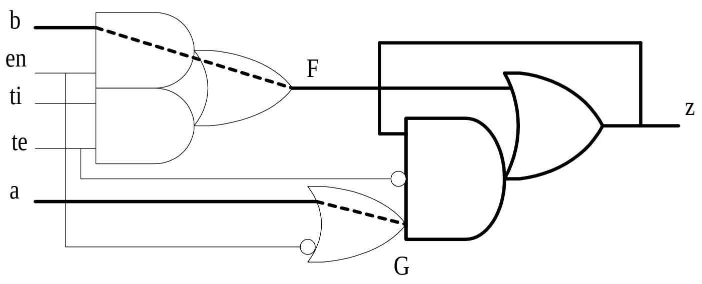


Figure 3.14: Standard-cell scan C-element implementation.

This mapping can be further optimized by changing the position of the inverters. Another consideration is the drive strength of the last gate. In the mapping shown, this gate is a complex gate that is already loaded by the internal feedback loop. An alternative would be to create a mapping that uses a separate output inverter. This decouples the feedback loop from the output and increases the drive strength. This mapping is shown in Equation 3.19 and the implementation is shown in Figure 3.15.

$$\begin{aligned} F &:= \overline{b \cdot en + ti \cdot te} \\ G &:= \overline{a + \overline{en}} \\ y &:= F \cdot (y + te + G) \\ z &:= \overline{y} \end{aligned} \tag{3.19}$$

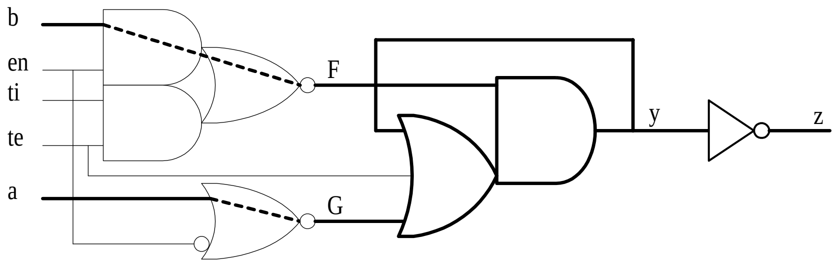


Figure 3.15: Alternative standard-cell scan C-element implementation with dedicated output inverter.

For some C-elements this optimization will introduce an inverter for the scan input signal. This inverter can be removed by treating the element as an inverting scan cell during test generation.

Scan C-element flip-flop

In scan mode, the scan C-element is operated as an LSSD type master slave flip-flop. This is implemented by adding a normal latch to the scan C-element. The C-element functions as the master latch and the additional latch as the slave latch. As can be expected, the new scan C-elements are significantly larger and slower than the original elements. The area of the asymmetric scan C-element shown in the example, including the additional slave latch, is about 5 times larger than the original C-element. For other C-elements, the area overhead is comparable or somewhat lower. The average area increase of a mix of C-elements that is typical for a Tangram design is about a factor 4.5. The scan C-element is also about 2 to 3 times slower, counted by the number of inversions between input and output. In the original C-element there are 2 inversions. In the scan C-element there are 6, including the 2 for the slave latch. These numbers can vary slightly depending on the type of C-element.

3.3.4 Primitive scan C-elements

The scannable C-elements represent by far the largest contribution to area increase and performance degradation in the final scan testable netlist. This makes it desirable to design new primitive cells for these scan C-elements. The design of these cells has been introduced in [12].

LSSD latch version

The most efficient implementation is derived from the clock multiplexing structure shown in Figure 3.12(b). The master latch is replaced by a C-element function. This

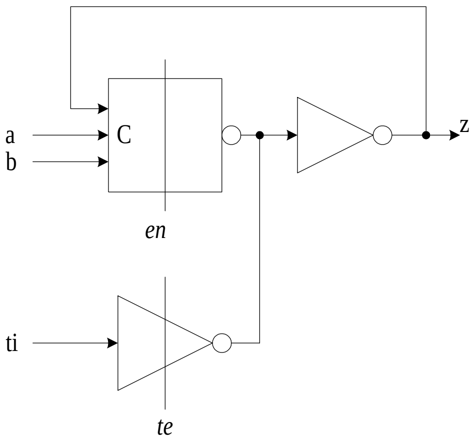


Figure 3.16: Custom scan C-element latch principle.

leads to the overall structure shown in Figure 3.16. An enable signal is added to the original C-element and this C-element forms an LSSD multiplexer with a tristate inverter.

LSSD flip-flop version

The flip-flop version of a primitive scan C-element is created by replacing the inverter in Figure 3.16 with a tristate inverter. The resulting structure is shown in Figure 3.17.

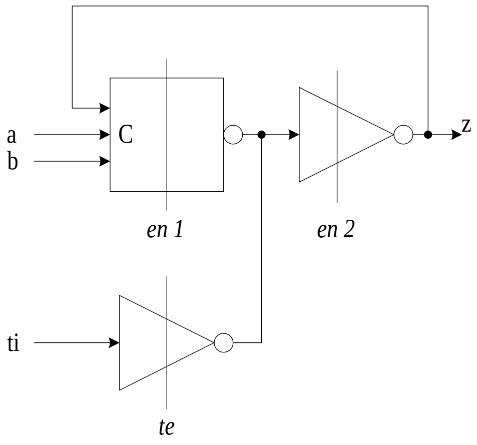


Figure 3.17: Custom scan C-element flip-flop principle.

A transistor-level implementation of this structure implementing the same type of asymmetric C-element used in the previous section is shown in Figure 3.18. The implementation of the symmetric version is shown Figure 3.19.

These primitive C-elements use about twice the area of the original non-scan C-elements, which is less than half of the area used by the composite scan C-element implementations. The speed of the elements is also improved. The number of inversions is reduced back to two. However the height of the transistor stack increased from two to three because of the tristate transistors. Therefore the gates are still somewhat slower than the original non-scan C-elements. It is also important to note that these C-elements use dynamic logic during the scan test, but that in the normal asynchronous mode of operation the behavior is completely static. The state in that case is not stored on an internal node but in the conventional way with a feedback loop.

With this type of scan C-elements, a new cell has to be added to the library for every type of C-element that is used in the design. An alternative solution is the design of one generic transparent scan flip-flop that can be customized with additional gates to form the entire family of scan C-elements. The benefit of using such a generic element is a reduction of the development and characterization work for the new cells.

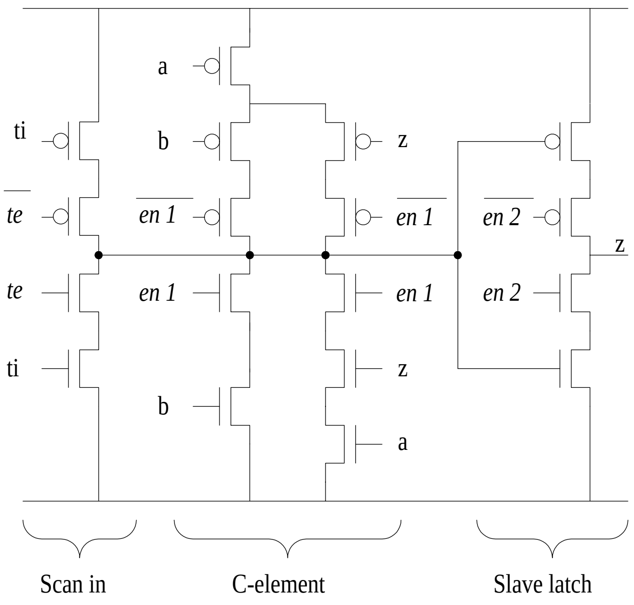


Figure 3.18: Transistor-level asymmetric scan C-element implementation
.

The area requirement and power consumption of C-elements based on a generic cell will be in between those of the composite and primitive gates shown before.

3.4 Clocking strategies

In order to clock the new scan elements, global clock signals have to be added to the circuit. These clock signals are only used during scan test and are idle during normal circuit operation. This changes the requirements of the clocks compared to synchronous clocking. The most important observation is that the speed of the clock is not important. The only requirements are the reliable operation during scan test, by preventing skew problems [28] and the minimization of the area that is used for buffers and wiring in the implementation.

Two types of clocking strategies are used: level-sensitive clocking for latches and C-element and edge-triggered clocking for flip-flops. In handshake circuits, most of the scan elements are either latches or C-elements but scan flip-flops can also occur in the data path. Therefore both types of clocking strategies are important and their properties are discussed in Section 3.4.1 and Section 3.4.2. Furthermore, the interaction between the two clocking strategies and the on-chip generation of the clock signals are the subjects of Section 3.4.3 and Section 3.4.4.

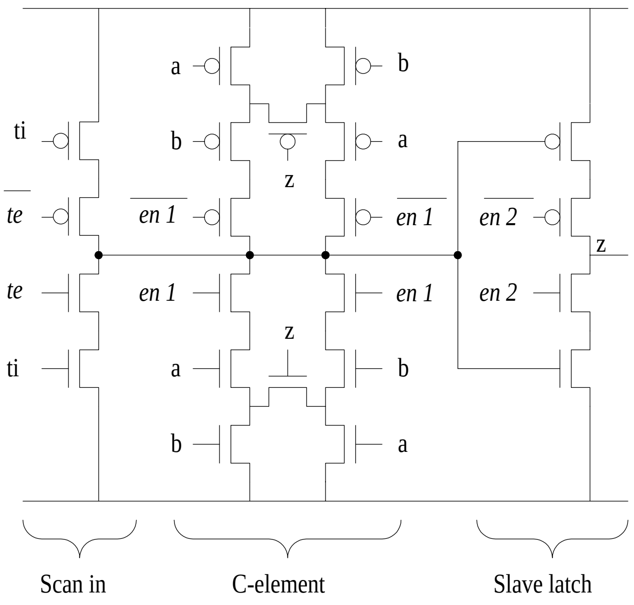


Figure 3.19: Transistor-level symmetric scan C-element implementation.

3.4.1 Level-sensitive clocking

The level-sensitive latches and C-elements in the scan circuit are clocked with a two-phase level-sensitive clock. Master and slave latches are clocked with independent clock signals. This allows external control over all clock phases, which makes it possible to avoid any timing problems such as skew. The negative aspect of this is that it requires more global wiring since two clock signals need to be distributed over the chip.

Figure 3.20 shows master and slave latch together with a timing diagram of a two-phase level-sensitive clock. The timing requirements of the latches can be expressed in two time intervals:

- Stability interval (s): the time during which the data input of the latch has to be stable.
- Validity interval (v): the time during which the output of the latch is valid.

To ensure save data transfer between two latches, the stability interval of the receiving latch has to fit within the validity interval of the sending latch. The top timing diagram of Figure 3.20 shows which circuit properties determine the stability and validity intervals of a latch. The stability interval begins a setup time t_{setup} before the falling edge of the clock. It lasts until a hold time t_{hold} after the falling edge that is delayed with the clock skew t_{skew}. The validity interval starts a data-to-output latch

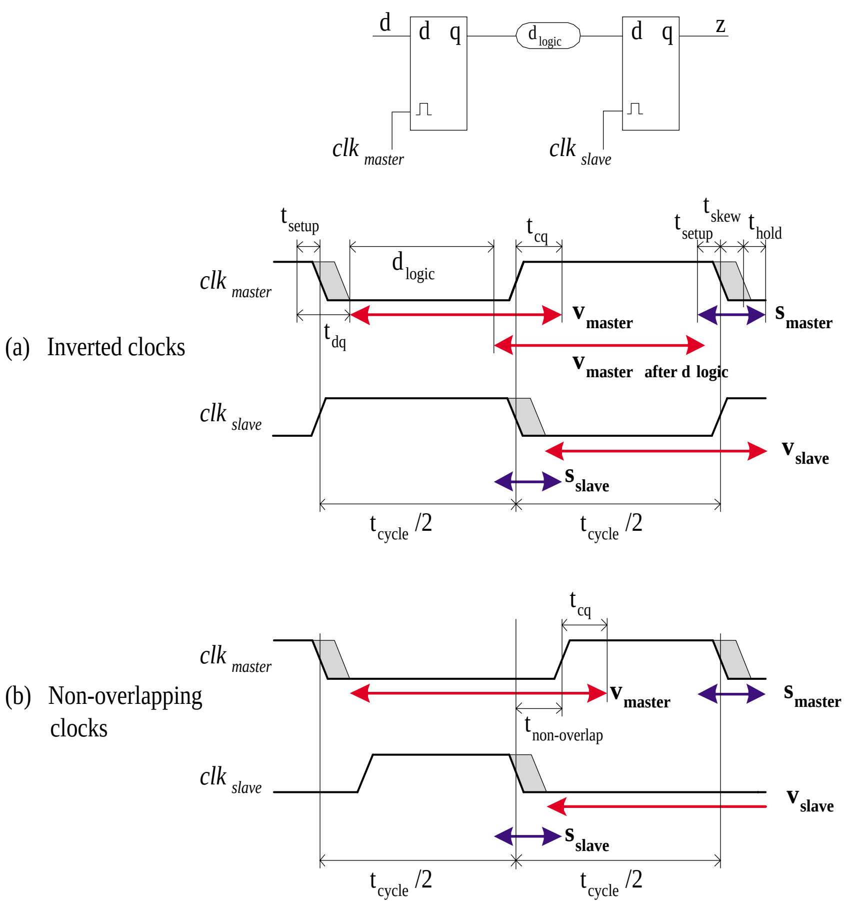


Figure 3.20: Level-sensitive clocking, (a) inverted clocks, (b) non-overlapping clocks

delay d_{dq} after the last moment that the data at the input of the latch could change, which is a setup time t_{setup} before the falling clock edge. The validity interval lasts until a clock-to-output latch delay d_{cq} after the rising edge of the clock. In those cases where logic is present between the latches, the validity period is delayed by the logic delay d_{logic}.

For the stability interval to fit inside the validity interval two conditions have to be valid, as shown in the next two expressions for the master to slave transfer. The first states that the validity period has to begin before the stability period, the second states that the stability period has to stop before the end of the validity period.

$$\begin{array}{rcll} s_{slave}\{begin\} & > & v_{master}\{begin\} & \Rightarrow \\ \frac{1}{2}t_{cycle} - t_{setup} & > & -t_{setup} + t_{dq} + d_{logic} & \Rightarrow \\ t_{cycle} & > & 2 \cdot (t_{dq} + d_{logic}) & \end{array} \tag{3.20}$$

$$\begin{aligned} s_{slave}\{end\} &< v_{master}\{end\} && \Rightarrow \\ \tfrac{1}{2}t_{cycle} + t_{skew} + t_{hold} &< \tfrac{1}{2}t_{cycle} + t_{cq} + d_{logic} && \Rightarrow \\ t_{skew} &< t_{cq} + d_{logic} - t_{hold} \end{aligned} \tag{3.21}$$

Equal relations hold for the slave to master transfer. The first condition is known as the max-delay problem, it defines the minimum required clock period. Alternatively it specifies what the maximum logic delay d_{logic} is for a given clock period. This condition can always be satisfied by choosing the clock frequency sufficiently low. The second condition is known as the min-delay problem, which defines the maximum clock skew the circuit can tolerate. The extend of the min-delay problem depends on both gate and circuit properties and is most severe if the logic delay d_{logic} is small. Additional margin can be added to remove this problem by making the two clocks non-overlapping. The non-overlap time $t_{non-overlap}$ is the time between the falling edge of one clock and the rising edge of the other clock. The non-overlap time increases the validity period of the latch and results in the modified min-delay condition:

$$t_{skew} < t_{cq} + d_{logic} - t_{hold} + t_{non-overlap} \tag{3.22}$$

By increasing the non-overlap time, all problems with clock skew can be avoided. By in addition increasing the clock cycle period, all other problems can also be avoided. With non-overlapping two-phase clocks it is therefore always possible to safely clock the circuit.

During scan, all latches are placed serially in a chain. This reduces the logic delay d_{logic} essentially to zero, leading to the worst-case situation with respect to the min-delay problem. Secondly, since the clocks are only used during scan, the signals can be minimally buffered to reduce the required area; this however increases the skew. For these reasons a clocking scheme is used that uses a large non-overlap period to decrease the sensitivity of the circuit to clock skew.

3.4.2 Edge-triggered clocking

Flip-flops can also be used in handshake circuits. They react on a rising (or falling) edge of a single distributed clock signal. The flip-flops include circuitry to produce an internal two-phase signal. This provides fewer possibilities to avoid timing problems by changing the clock parameters and therefore requires more thorough analysis of the circuit to prevent possible problems during the operation of the circuit.

The timing of flip-flops is shown in Figure 3.21. Again stability and validity intervals can be defined. Main difference with the latch is the start of the validity interval. Data on the input of a flip-flop has to wait for the clock. This means that the output only becomes valid after the clock-to-output delay d_{cq} corrected for clock

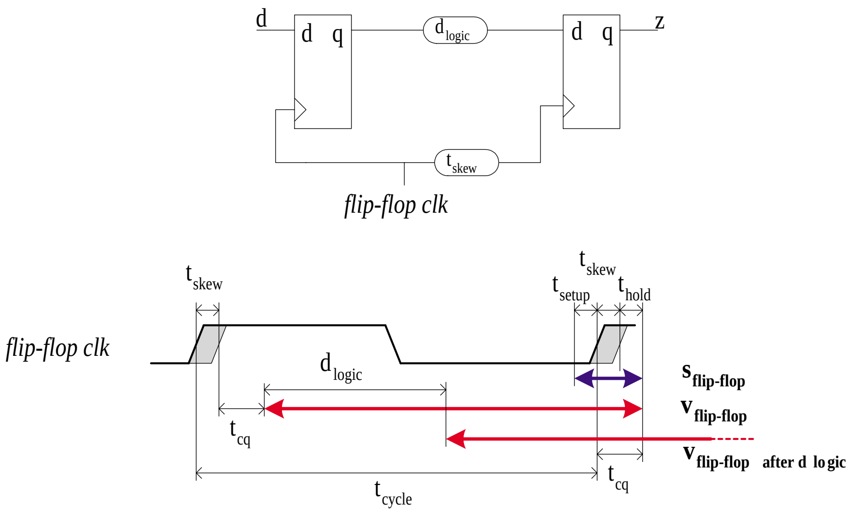


Figure 3.21: Edge triggered clocking.

skew t_{skew}. If logic is present between the flip-flops, the interval is delayed by the logic delay d_{logic}. For correct operation, the same two conditions have to hold for the stability and validity intervals, only the implications are now different:

$$\begin{aligned} s_{flip-flop}\{begin\} &> v_{flip-flop}\{begin\} & \Rightarrow \\ t_{cycle} - t_{setup} &> t_{skew} + t_{cq} + d_{logic} & \Rightarrow \\ t_{cycle} &> t_{skew} + t_{cq} + d_{logic} + t_{setup} \end{aligned} \tag{3.23}$$

$$\begin{aligned} s_{flip-flop}\{end\} &< v_{flip-flop}\{end\} & \Rightarrow \\ t_{cycle} + t_{skew} + t_{hold} &< t_{cycle} + t_{cq} + d_{logic} & \Rightarrow \\ t_{skew} &< t_{cq} + d_{logic} - t_{hold} \end{aligned} \tag{3.24}$$

The cycle time of an edge-triggered circuit also has to incorporate the clock skew t_{skew} and setup time t_{setup}, resulting in somewhat larger clock periods. The min-delay issue is a more serious problem with edge-triggered circuits. Unlike the case of level-sensitive clocks, the problem cannot be removed simply by changing the timing parameters of the clock. Therefore, circuit design methods have to be used that ensure that the clock skew is kept to a minimum and that the logic delay d_{logic} is large enough to avoid the problem, This can be accomplished by e.g. by adding buffers to delay the data inputs to a flip-flop.

For this reason, whenever flip-flops are used, some form of clock-tree balancing is required. Fortunately the number of flip-flops in a handshake circuit is usually very limited; the majority of the scan elements are level-sensitive, making the problem relatively easy to solve.

3.4.3 Combining edge-triggered and level-sensitive clocking

In handshake circuits, latches and flip-flops can be used together in a circuit. This means that it should be possible to safely transfer data between these two types of elements. This can be both in normal mode as well as in the scan chain, although in the later case can be avoided by making separate scan chains for each type of element. Figure 3.22 shows a possible situation in which a master-slave latch is connected to a flip-flop, which is again connected to a master-slave latch. There are two potentially unsafe data transfers:

- From the slave latch to the flip-flop
- From the flip-flop to the master latch.

Both conditions can be expressed with stability and validity periods. For the slave latch to flip-flop transfer, the stability period of the flip-flop has to end before the validity period of the slave latch ends.

$$\begin{array}{rcll} s_{flip-flop}\{end\} & < & v_{slave}\{end\} & \Rightarrow \\ t_{skew} & < & t_{cq(slave)} - t_{hold(ff)} + d_{logic} & \end{array} \tag{3.25}$$

For the flip-flop to master transfer, the stability period of the master latch has to end before the validity period of the flip-flop ends.

$$\begin{array}{rcll} s_{master}\{end\} & < & v_{flip-flop}\{end\} & \Rightarrow \\ t_{skew} & < & t_{cq(ff)} - t_{hold(master)} + d_{logic} & \end{array} \tag{3.26}$$

The two conditions indicate possible timing problems if the clock skew t_{skew} between the clocks is large. Similar to two-phase level-sensitive clocking, the clock skew problem can be removed by inserting non-overlap times between the flip-flop clock and the two latch clocks. Two non-overlap times are defined and shown in Figure 3.22: The master to flip-flop interval $t_{master\ to\ ff}$ and the flip-flop to slave interval $t_{ff\ to\ slave}$. With these intervals, the timing relations become:

$$\begin{array}{ll} \text{slave to flip-flop} & \Rightarrow t_{skew} < t_{cq(slave)} - t_{hold(ff)} + d_{logic} + t_{ff\ to\ slave} \\ \text{flip-flop to master} & \Rightarrow t_{skew} < t_{cq(ff)} - t_{hold(master)} + d_{logic} + t_{master\ to\ ff} \end{array} \tag{3.27}$$

The timing diagrams a and b in Figure 3.22 show a two-phase non-overlapping clock signal pair for the master and slave latches. These can be combined with a flip-flop clock signal in several ways. The first approach is to generate a new flip-flop clock signal with sufficient non-overlap intervals before and after the rising edge, such that all timing problems are prevented. This is the signal shown in timing diagram c of Figure 3.22. The disadvantage is that it requires the addition of a new

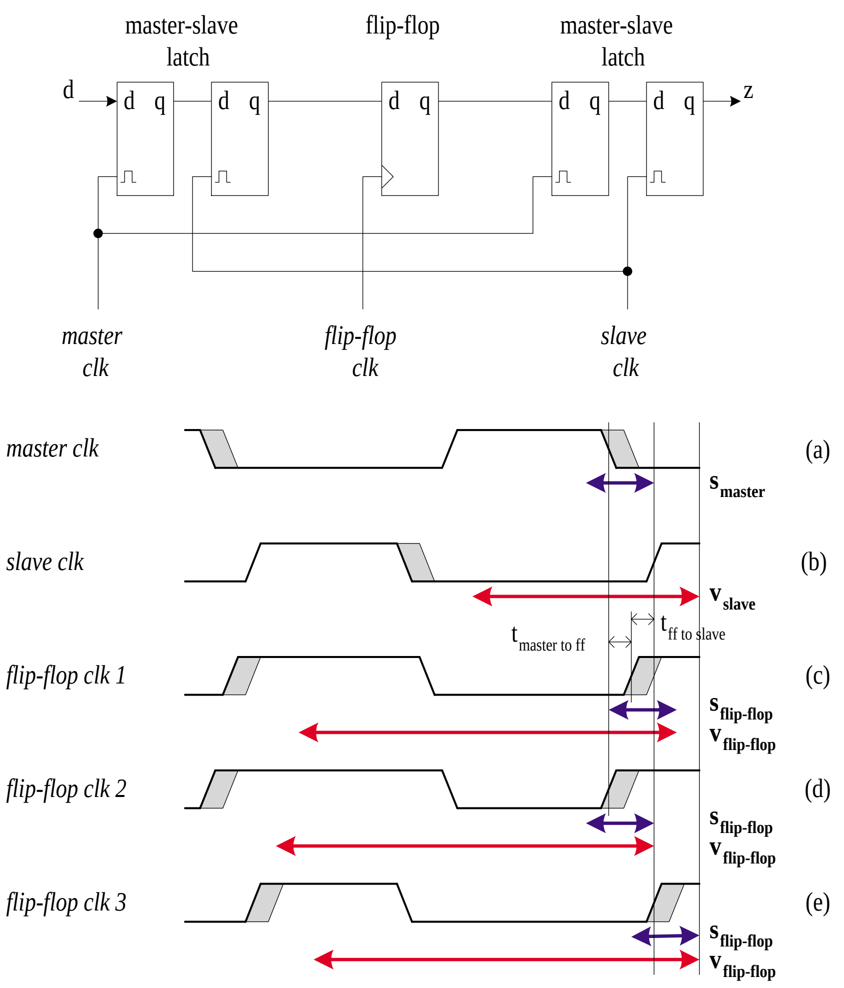


Figure 3.22: Timing conditions for data transfers between level sensitive latches and edge triggered flip-flops.

independent clock signal. To reduce cost, the flip-flop clock can also be connected to either the inverted master clock (Figure 3.22(d)) or the slave clock (Figure 3.22(e)). The first option removes the control over $t_{master\ to\ ff}$, the second over $t_{ff\ to\ slave}$. As shown in the timing diagrams, this brings the stability and validity periods close together and increases the vulnerability to skew. Whenever this leads to a timing violation, an anti-skew latch must be inserted to remove the problem.

The problem will be most profound in the scan chain, since in this case d_{logic} is considered to be zero. Therefore, the skew problem is minimized by reducing the number of transfers between latches and flip-flops. This is accomplished by first creating two separate groups, one with latches and one with flip-flops. If these two groups are connected, at most one anti-skew latch is required at the interface. During normal mode operation, d_{logic} will in general be non-zero (or a delay can be added), and hence anti-skew latches are not required for the non-scan inputs.

3.4.4 On-chip clock generation

In order to operate the scan chain, two (or three) clocks are required. Although it is more flexible if these clocks are connected to external pins on the chip, this is not always possible because of cost reasons.

In this section therefore an on-chip clock generator is shown that can produce all required internal clocks from a single external reference clock. Figure 3.23(a) shows how to generate a two-phase non-overlapping clock with a pair of cross-coupled NOR gates [57]. The non-overlap time can be tuned, independently for rising and falling edges, by modifying the delays $d1$ and $d2$ in the feedback loops.

By adding a third delay $d3$, it is possible to also generate an independent flip-flop clock with this circuit. This can be done in two ways, either by deriving the flip-flop clock from the inverted master clock (as shown in Figure 3.23(b)) or by deriving it from the slave clock (as shown in Figure 3.23(c)). In both cases, the rising edge of the clock falls after the falling edge of the master clock and before the rising edge of the slave clock, thereby satisfying the constraints in the previous section. The main difference between the last two methods is the timing of the falling edge of the flip-flop clock. This timing is not critical for normal circuit operation, but as will be shown in the next chapter, it does have an influence on the timing of the latch controllers.

For the use in handshake circuits, the clock generator also needs to be able to set the scan elements in transparent mode to support the asynchronous mode of operation. This requires both the master latches and the slave latches to be open. With the the implementations shown Figure 3.23 this is not possible. In Section 4.4.1 a clock generator is shown that is able to put the circuit in transparent mode, by using a separate test-mode signal.

3.5 Summary

In this chapter, the basic ingredients of a scan solution were presented that can be used to make handshake control circuits scan testable. In this way two of the testability problems identified in Chapter 2 could be solved: the autonomous and the sequential nature of the control circuits. The solutions for the remaining problems will be presented in Chapter 4.

The basic ingredients of the scan method are the scan C-elements. Several implementations have been presented. First, composite scan C-elements were introduced that are large and slow but can be implemented in any technology library. These

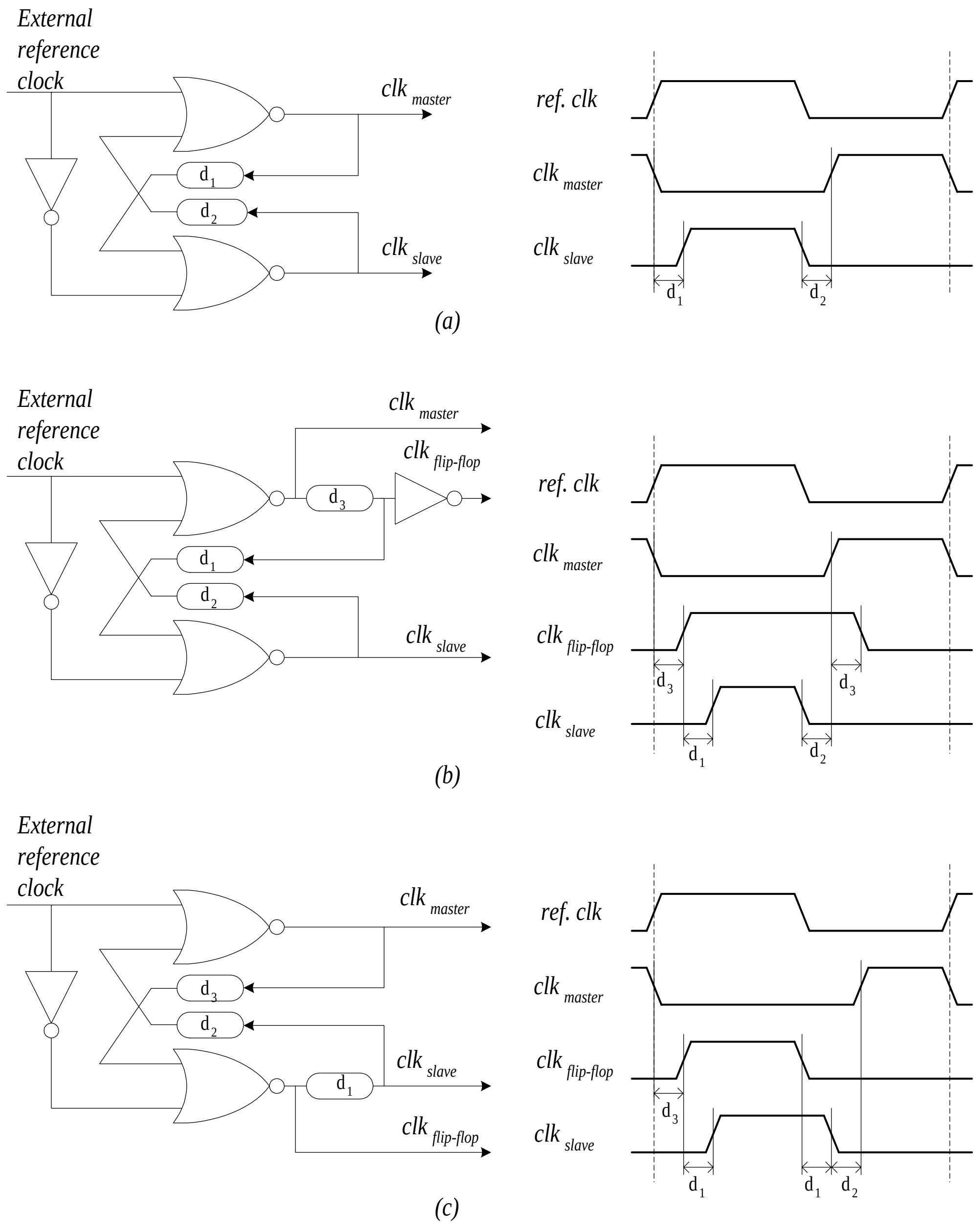


Figure 3.23: On chip clock generation, (a) two-phase non-overlapping clock, (b) with additional flip-flop clock derived from the master clock, (c) with additional flip-flop clock derived from the slave clock.

were followed by primitive scan C-elements that are smaller and faster but require an extension to the standard-cell library.

The scan C-elements are clocked with two-phase non-overlapping clocks. This makes it possible to reliably operate the scan chain, while being able to tolerate clock skew. Combined with the low target speed, this means that area can be saved by reducing the clock buffering circuitry that is normally required to generate a well defined clock signal. Another advantage is that it simplifies the design flow, by removing many of the steps that are normally used to insert and balance the clock.

In the data path flip-flops can still occur. The flip-flop clock still has to be optimized for minimal skew. Timing problems between the latches and flip-flop parts can be avoided by inserting non-overlap times between the clocks. However if the clock skew is not a problem, the flip-flop clock can be shared with one of the latch clocks to reduce the wiring area. Further reduction can be achieved by generating all of these clocks on-chip from a single external reference source. The main benefit of this scheme is that it only requires one external clock pin.

Chapter 4

Full scan test for handshake circuits

In this chapter, full-scan testing is applied to the complete handshake circuit. The test method is based on the use of scan C-elements in the control block combined with conventional scan latches and scan flip-flops in the data path. A new latch controller is introduced to allow the local clocks of the latches and flip-flops to be controlled by a global clock during the scan test [6].

Test vector generation for the modified circuit can be carried out with existing test tools. These tools were not specifically designed to be used with handshake circuits, but by using techniques described in this chapter they can be applied nevertheless.

In the last part of this chapter, a number of additional circuit modifications is given to remove a number of problems associated with certain circuit structures. Among these are the testability problems that were introduced in Chapter 2 and that are not solved by the scan modifications introduced sofar. A second class of modifications is targeted at improving the results of the method.

4.1 Design for testability

A scan testable circuit is created by replacing all sequential elements with the scannable versions of these elements. In the control block this means replacing the C-elements with scan C-elements. New global signals are added to the control block and are connected to the enable and scan-mode inputs of the scan C-elements. These signals are used to control the circuit during the scan test.

In the data path, each latch is replaced by a master scan latch and a slave latch. The flip-flops are replaced by scan flip-flops. Like in the control block, global control signals are connected to the scan-enable inputs of the master latches and to the enable inputs of the slave latches. The enable inputs of the master latches and of the scan flip-flops cannot be directly connected to a global control signal. These inputs are already connected to the local clock signals generated by the latch controllers. The latch controllers therefore need to be modified to allow the multiplexing of a global clock signal onto the internal local clock signals that are connected to the enable inputs of the latches and flip-flops.

4.1.1 Latch controller modification

For the implementation of the scan method, a new latch controller is required that can be used to control the local clock signals with a global clock signal.

Latch controllers translate the handshake signals that are used in the control block into suitable local clock signals for the data path. Every register in the data path, that can consist of either latches or flip-flops, has its own latch controller. The control block will activate a latch controller whenever the register it controls needs to capture a new data value. For historical reasons, a latch controller that clocks a flip-flop register will still be referred to as a latch controller.

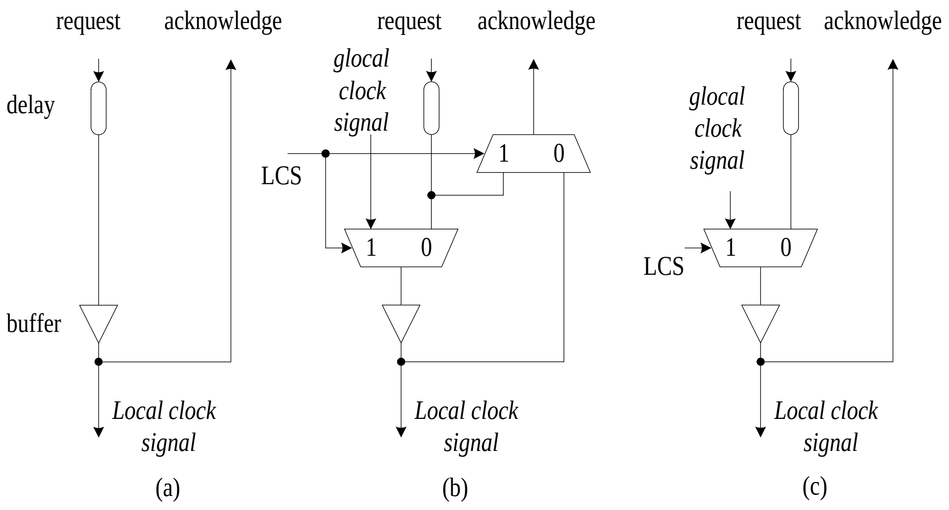


Figure 4.1: Latch controller implementations (a) Original version (b) Initial scan version (c) Optimized scan version.

The latch controller that is used in the original non-scan circuit is shown in Figure 4.1(a). It consists of only two components, a delay element and a buffer. Each of these helps to condition one of two properties of the enable signal generated by the latch controller. The two properties of the clock signal are the timing and the drive strength. The timing of the clock signals is set by the delay element. The clock signal is delayed such that the data inputs of the register are guaranteed to be valid when the register is clocked. The buffer is used to increase the drive strength of the clock signal to match it to the load on the signal.

The clock signal is also used as the acknowledge signal of the latch controller to the control block. It signals to the control block when the register has completed the capture of new data. It is of course important that this does not happen before the registers are actually closed. In the original latch controller this is done by directly connecting the acknowledge signal to the clock signal. In this way when the clock signal is delayed because it has to drive a large load, the acknowledge signal is also delayed. An implementation that decouples the acknowledge signal from the clock signal is less safe and should not be used.

These requirements equally hold for a latch controller that can be used for scan test. The multiplexing function that is required to multiplex the global clock on to the local clock is therefore not allowed to cause a decoupling of the acknowledge signal from the local clock signal. One implementation that fulfills this requirement is shown in Figure 4.1(b).

The circuit uses two multiplexers to completely separate the control block and the data path during test. The latch-control select signal (LCS) is high during the entire test. In the implementation, the acknowledge signal and the local clock signal are still coupled during normal-mode, when LCS is low. When the local clock signal is delayed, the acknowledge is also delayed. The multiplexer in the acknowledge signal will only further delay the signal. The obvious disadvantage of this latch controller implementation is that it is not fully testable. Most importantly it does not test the asynchronous-mode behavior of operating the latch via the request signal. Additionally the connection from the local clock signal to the acknowledge signal is not tested.

To test these two connections, the circuit has to be operated in normal-mode ($LCS = 0$) during the evaluation-phase of the test. This would enable the test of the connection from the request signal via the buffer to the acknowledge signal. On the other hand this would prevent the register connected to the latch controller from being clocked during the evaluation-mode. Even worse, the local clock signal is controlled by the test pattern. For some patterns the clock will be driven high, thereby making the data registers transparent and destroy the information inside the registers.

For this reason it is not possible to use the normal-mode for test evaluation when at the same time the registers are used to store data. The solution to this problem is to use two separate tests. The first test can use the normal-mode for test evaluations but cannot test the registers. The second test uses the test-mode for test evaluation and can make use of the registers. Together the two tests provide test coverage of the entire circuit. To implement this scheme, the circuit is split into a control block and a data path. The control block test uses the normal-mode during the evaluation phase and the data path test uses the test-mode during the evaluation phase.

With this test approach, the multiplexer that drives the acknowledge signal in the latch controller show in Figure 4.1(b) is no longer required. The acknowledge signal is only observed during the evaluation phase, which for the test of the latch controller is executed in normal-mode. The test-mode is only used to scan data in and out of the scan chain, during which the acknowledge signal is not observed. The final implementation of the latch controller is shown in Figure 4.1(c). The multiplexer will increase the delay of the latch controller and therefore slow down the circuit. This can be corrected by reducing the delay of the delay element in the modified latch controller by the delay represented by the test modification.

4.1.2 Design for testability overview

A schematic representation of the final scan-testable circuit is shown in Figure 4.2. The implementation is split in a control block and a data path. This is required because the two blocks will be tested separately, as addressed above.

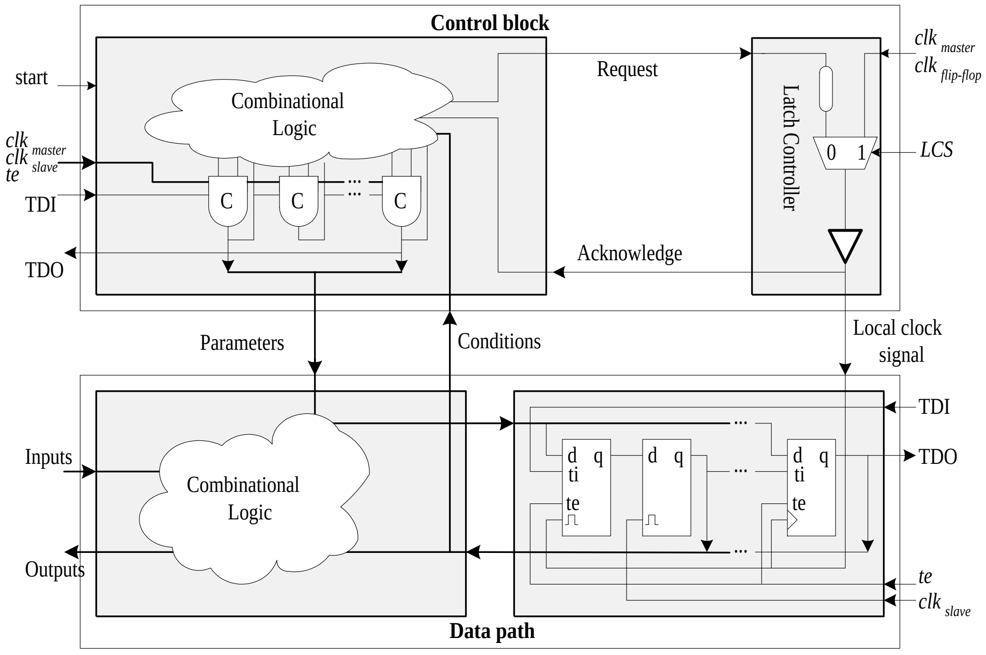


Figure 4.2: Gate-level partitioning of the scan-testable circuit. Both the control block and the data path are scannable. The original latch controller is replaced by the version in Figure 4.1(c).

In the control block all the C-elements are scannable. There are three global control signals used to control the C-element: the master clock clk_{master}, the slave clock clk_{slave} and the test enable te. The latch controllers use the implementation shown in Figure 4.1(c). This requires more control signals: the latch-control select signal LCS and the clock signal for the registers. The clock signal can be either the master clock clk_{master} or the flip-flop clock $clk_{flip-flop}$ or both, depending on the type of registers (latch or flip-flop) that are used. For clarity only one latch controller is shown, in a real circuit of course many are present, each driving its own register via a dedicated local clock signal.

In the data-path, scan versions are used for the latches and the flip-flops, equal to conventional synchronous scan implementations. The master clock for the registers is already supplied by the latch controllers. Two additional control signals are required: the test enable and the slave clock. The latter is of course only required if latches are used in the data-path.

The structure shows separate scan chains for the control block and for the data path. Alternatively these two scan chains can be combined into one scan chain or further split up into more scan chains. In Figure 4.2 separate control signals are shown for the control block and the data path. Some of these signals can be shared, for example the test enable and the slave clock. This is shown in Section 4.4.

4.1.3 Latch timing constraints

The critical components with respect to scan test signal timing are the latch controllers that generate the enable signals for the latches and flip-flops in the data path. They form a connection between clock signals and data signals. Such a connection violates the scan rules that are used to guarantee correct operation of the scan test. For this reason the timing of the latch controllers need to be analyzed to be able to guarantee correct scan test operations with these elements present. The analysis is split in two part, in this section latch controllers are analyzed that drive latches in the data path. The next section analyzes the timing of the latch controllers that drive flip-flops.

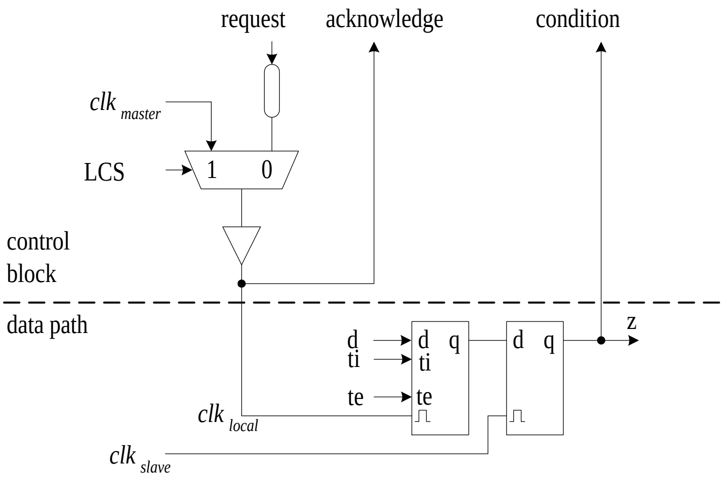


Figure 4.3: Latch controller driving a latch.

Figure 4.3 shows a latch controller that drives a latch, including all relevant control signals. Clk_{master} and clk_{slave} are used to clock the latch. The LCS signal controls the multiplexer in the latch controller. In the next two sections, the circuit in Figure 4.3 is analyzed for timing constraints. First the timing is analyzed during the data path test, afterwards it is analyzed during the control block test.

Data-path test timing

During the data-path test, the circuit is kept constantly in test-mode, by keeping LCS high. In this mode, the local clock clk_{local} is equal (however slightly delayed) to clk_{master}. The clock timing in this case corresponds to the default situation as shown in Section 3.4.1, provided a large enough non-overlap time is present to prevent problems with skew and the delay of the latch controller.

Control-block test timing

During the control-block test, the circuit has to switch between test-mode and normal-mode. The timing of the control signals is shown in Figure 4.4. The test-mode is used during scan shift cycles, the normal-mode is used during the evaluation cycle. As explained in the previous section, the change in test mode can cause the local clock driving a latch to change. This can destroy the state of the latch. In most cases this is not important since these registers are not used during the control block test. There are however two situations in which the registers are used to support the control block test. The first situation is during scan shifts where the registers can be part of the scan chain. This is executed in test mode and therefore the registers are clocked normally and can be used without problems. The second usage of a register during control block test is when the output of the register forms an input for the control block. This situation is shown in Figure 4.2 by the condition signals going from the data path to the control block. It is vital that condition signals remain stable until the control block has captured them.

The latch situation is shown in Figure 4.3. The condition signal is driven by the slave latch. It becomes valid after the falling edge of the slave clock clk_{slave}, the corresponding validity interval v_{slave} is shown in Figure 4.4. The condition signal has to remain valid until the data has been captured in the control block. This occurs at the falling edge of the master clock clk_{master}, that clocks the scan C-elements in the control block, at the end of the evaluation cycle. The required stability interval s_{master} is also shown in Figure 4.4. The timing intervals show that the slave latch to control block interface is always safe in terms of timing.

Besides this data transfer, the master latch to slave latch transfer is also important. The slave latch captures the data from the master latch during the first half of the evaluation cycle. For this transfer, the validity interval of the master latch v_{master} has to overlap with the stability interval of the slave latch s_{slave}. During the evaluation

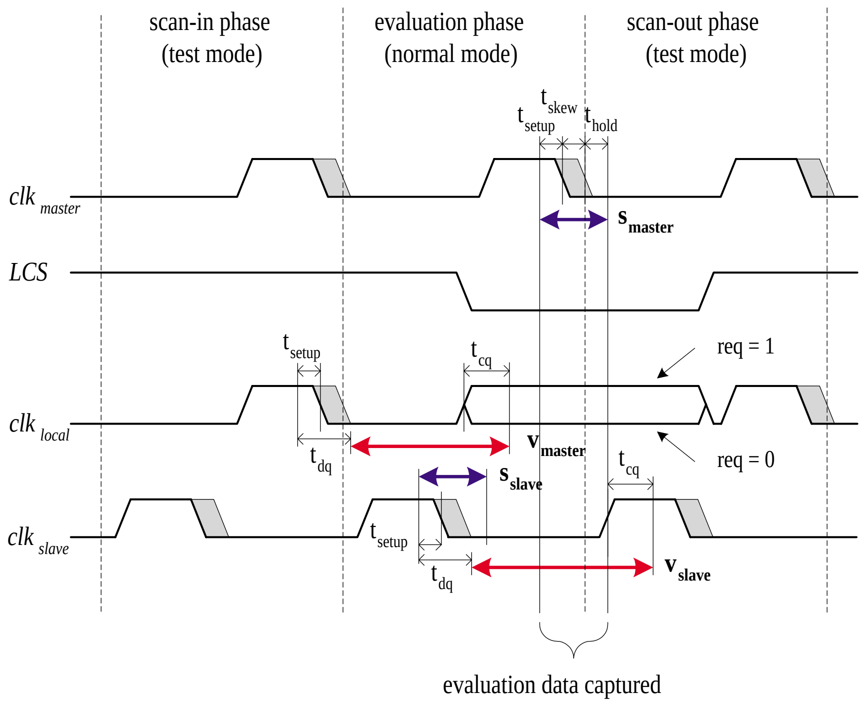


Figure 4.4: Latch timing during control-block evaluation.

cycle, the latch-control select signal LCS has to be switched to normal mode. This switch can change the state of the local clock signal and thereby ending the validity interval of the master latch. To guarantee that the validity interval of the master latch is sufficiently long, clk_{local} has to be kept low as long as clk_{slave} is still high. This implies that switching the LCS signal must be delayed until after the falling edge of clk_{slave}.

This condition will guarantee the correct operation with latches. In Figure 4.4, the LCS signal is set up to switch at 50% of the clock cycle, when both master and slave clock are low. Since on a tester the test-mode signal is defined by the same timing parameters as the combinational inputs to the circuit, other inputs of the circuit also will only become valid at 50% of the cycle. However, since the scan test is executed at low speed, the data inputs still have sufficient time to propagate to the scan elements. In Section 4.4 test control logic is presented that supports the generation of the LCS signal from input signals that switch normally at the beginning of the clock cycle.

4.1.4 Flip-flop timing constraints

In this section the timing of a latch controller that drives a flip-flop register is analyzed. The circuit is shown in Figure 4.5. One of the differences compared to the latch circuit are the additional inverters. These are required because a flip-flop is clocked on the inverse polarity of a master latch clock. The flip-flop react to the rising edge of the clock, as explained in Chapter 3, this means that the clock signal is sensitive for skew and clock tree balancing might be required.

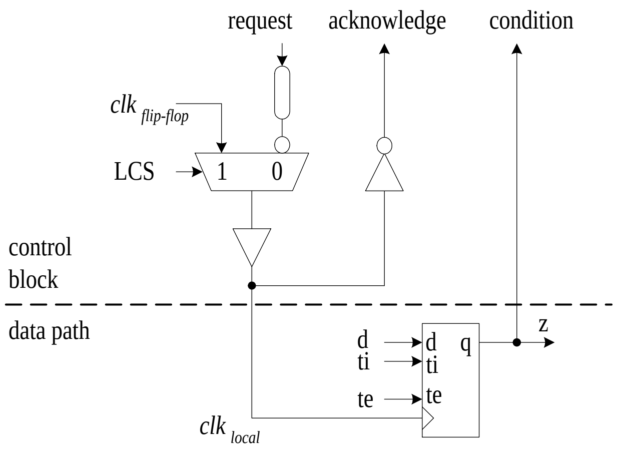


Figure 4.5: Latch controller driving a flip-flop.

Data path test timing

As said before, during the data path test the circuit is kept constantly in test-mode, by keeping LCS high. The local clock clk_{local} for the flip-flops is therefore equal to $clk_{flip-flop}$. The clock timing in this case corresponds to the default situation as shown in Section 3.4.2. To prevent problems with skew, it might be necessary to insert buffers in the clock to balance it.

Control block test timing

If flip-flops are used as registers, the situation is a little more complicated. Now only one clock signal is available to stop the propagation of incorrect values. The flip-flop controller is shown in Figure 4.5 and the timing of the control signals in Figure 4.6. The flip-flop output becomes valid at the rising edge of the local clock clk_{local} in the beginning of the evaluation cycle and has to remain valid until the falling edge of clock clk_{master}, which is used to capture the data in the control block. During this interval no rising edge may occur on clk_{local}, a falling edge however may occur since this does not change the output of a flip-flop.

There are two requirements to ensure that no rising edge occurs on the clock of the flip-flop before the falling edge of clk_{master}. The first requirement is that the request signal going into the multiplexer is stable between the time the multiplexer is

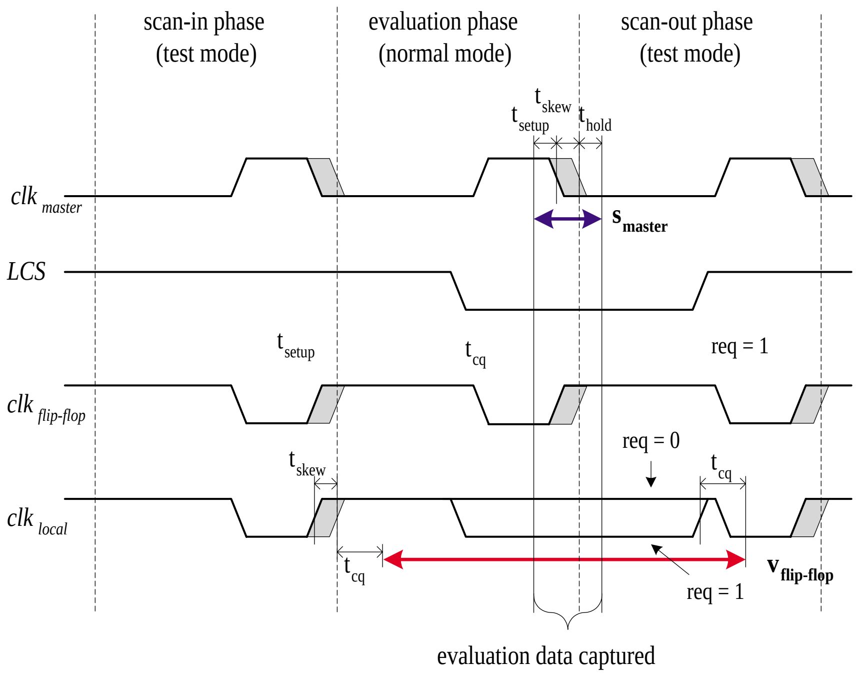


Figure 4.6: Flip-flop timing during control-block evaluation.

switched into normal mode and the falling edge of clk_{master}. The second requirement is that clock $clk_{flip-flop}$ has to remain high until after the multiplexer in the latch controller is switched into normal mode.

The requirement that the request signal has to be stable leads to two possible valid situations: the request signal is constant 0 or the request signal is constant 1. When the request signal is constant 0, the local clock remains constant at 1. In this situation, the acknowledge signal is constant 0 and output z is stable. If the request signal is constant 1, then there is a falling transition on the local clock when LCS switches to zero. This leads to a rising transition on the acknowledge signal. The transition has no influence on the output z, which remains stable. The last problem is to guarantee that the request is stable between the falling edge of LCS and the falling edge of clk_{master}. The request signal can originate from a slave latch, a primary input or from another latch controller, further elaborated below.

- Slave latch: The majority of the request signals will originate from a slave latch. These request signals will be stable since the slave latches, clocked by clk_{slave}, are closed before the falling edge of LCS.

- Primary input: Primary inputs normally use the same timing parameters as the LCS signal. This can potentially lead to a race between the input and the

LCS signal. In situations where this could occur, the timing of the primary inputs should be changed to switch earlier in the clock cycle. Alternatively the LCS signal could be slightly delayed. Most situations where the request signal is originating from a primary input, the register is used to capture an external data input. By avoiding flip-flops to capture external inputs, potential problems can be avoided.

- Latch controller: The acknowledge signals of the latch controller can have a rising transition when LCS is switched into normal mode. The effect of such a rising edge on an acknowledge signal is to lower the corresponding request signal. Such a path always contains a scan C-element or another scan element that stops the transition from propagating. If the scan element would not be present, a combinational loop would exist and as is explained in Section 4.3.2, a combinational loop always will be broken by inserting a scan element in it.

In Figure 4.6, clk_{local} shows the timing of the flip-flop clock signal. The falling edge of the clock is delayed until after LCS has switched to zero. This is implemented by using the inverse of the master clock clk_{master}, as the flip-flop clock. Unlike normal clocks for flip-flops, the location of the falling edge of the clock is important for correct operation with latch controllers. When using one of the on-chip clock generators shown in Figure 3.23, therefore always the variant shown in Figure 3.23(b) should be used. This ensures correct timing since the falling edge of the flip-flop clock is delayed until after the falling edge of LCS.

4.2 Test generation

The circuit modifications described in the previous section introduce a hierarchical structure in the circuit. Two lower level blocks for the control block and data path are combined into a top level block. The two lower level blocks are tested separately. This requires the use of a test method that supports hierarchy. Well-known examples of such test methods are the core-based test proposal (P1500) [38] and the macro test method [39]. Both of these methods use the concepts of a test-pattern and a test-protocol that together define a complete test for a circuit.

Test pattern A test pattern contains a set of stimulus and response values. A test pattern is part of a test-pattern set. Such a set contains all test patterns required to test a circuit with a certain fault coverage.

Test protocol A test protocol defines how a test pattern should be applied to the circuit. The initial test protocol describes this information at the lowest (block) level. An expanded protocol describes the information at a higher (or top) level.

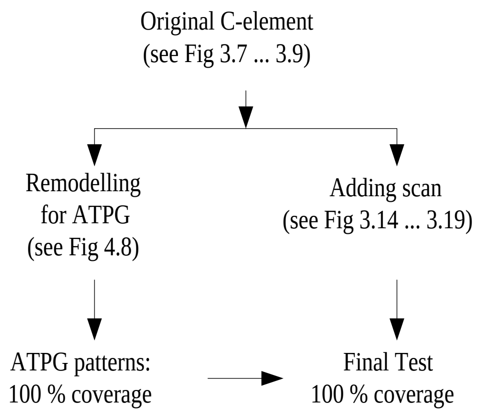


Figure 4.7: Remodelling principle.

The test-generation process consists of two steps. First, patterns and initial protocols are generated, both for the control block and for the data path. Second, these initial test protocols are expanded to top-level. The pattern generation and protocol expansion steps are not completely independent from each other, as will be discussed in the next section. For successful test-protocol expansion some constraints have to be placed on the test pattern generation process.

4.2.1 Test-pattern generation

Although the new scan C-elements and the scan latches are functionally correct scan elements, existing ATPG tools do not necessarily recognize them as such. These tools typically only recognize flip-flops as valid scan elements. The latch function of a C-element is implemented by a feedback loop. ATPG tools typically mark such a loop as invalid and then try to generate test vectors that test as much as possible of the remaining parts of the circuit. Another problem is that C-elements have two or more normal-mode inputs, which by some internal function determine the next state of the element.

Both problems can be solved by using a remodelled circuit that has the same *test* properties as the original circuit. Since the remodelled circuit is only used for test-pattern generation, it does not have to be able to operate in the normal asynchronous mode. The basic principle of remodelling is shown in Figure 4.7. From the original netlist with C-elements, two other netlists are derived. In one, the C-elements are replaced by scan C-elements; this is the final scan testable netlist. The other, in which C-elements are replaced by remodelled C-elements, is the netlist to be used exclusively for test-pattern generation.

For remodelling to work, two requirements have to be fulfilled:

- The test patterns generated with the remodelled elements have to be *valid* for the real circuit. This means that the test responses from the remodelled circuit are equal to the test responses of the real circuit.

- The test patterns that are generated have to be *necessary* and *sufficient*. This is to guarantee that all faults in the real circuit are covered by faults in the remodelled circuit and that all patterns that are generated increase the fault coverage of the real circuit.

If this is the case, then the test patterns calculated for the remodelled circuit are able to test the real circuit with 100% fault coverage, of course provided the ATPG tool could find a test with 100% coverage for the remodelled circuit. Faults that are untestable in the remodel circuit directly lead to a reduced fault coverage in the real circuit.

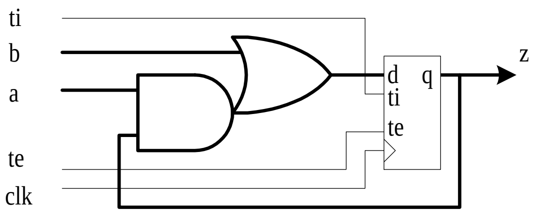


Figure 4.8: Remodelled C-element.

The circuit used to remodel a scan C-element is constructed by taking the original C-element and inserting a scan flip-flop in the feedback loop. In Figure 4.8, an example is given for an asymmetric version of a C-element. Every type of C-element requires its own remodelled version. The key to this structure is that the function located at the data-input of the flip-flop is the same function that is used in the original C-element to calculate the next state function. This means that whenever the functional inputs of the two elements are the same, the next state will also be the same. Furthermore, the logic before the flip-flop is part of the combinational circuit for which the ATPG tool will generate test patterns. This ensures that by using these patterns on the scan C-elements all the stuck-at faults in the logic part of the scan C-elements are tested. This corresponds to the bold parts in Figure 3.14 and Figure 3.15. The remaining faults in the scan C-elements are tested with a scan continuity test. Remodelling of latches is carried out by replacing them with flip-flops.

4.2.2 Test-protocol expansion

During test-protocol expansion, the initial protocols that were created by the ATPG tool are translated into top-level protocols. The tool used to do this was developed

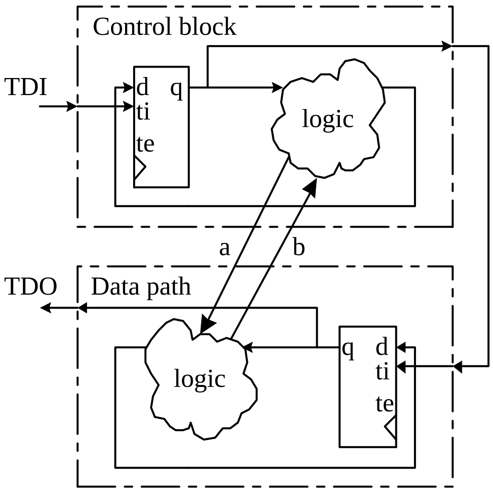


Figure 4.9: Protocol expansion in the initial circuit.

for the macro test flow [39]. It traces signal paths in the circuit to find a path from a pin of a lower level block to a top-level pin or to the scan chain.

The problem is illustrated in Figure 4.9, which shows an example of the test protocol expansion problem. Test patterns and initial protocols are available for both sub-blocks. During the generation of these files, no information was used about the interconnection between the two blocks. This means that the ATPG tool has assumed complete controllability and observability of the interface signals, which might not be the case in the real circuit.

In order to find paths to observe and control the pins of the interface signals (a and b in Figure 4.9), the test-protocol expansion tool needs to trace the interface signals back to observable and controllable locations. This is complicated by the fact that tracing these signals sometimes requires that the surrounding logic is in a state in which the signals can propagate through the logic to a scan cell or an input.

The application of this tool for an asynchronous circuit leads to the problem that some circuit structures cannot be correctly traced. For example in the latch controllers it is possible that the test-expansion tool traces a data signal from a handshake data input to the clock signal. This will result in an error and hence the protocol cannot be expanded to top-level.

The solution to this problem consists of two parts. First, some restrictions are put on the interface between the two sub blocks and second, an abstract description of the scan chain is used to hide the internal asynchronous logic of the block. This removes the need to trace interface signals, thereby avoiding problems with latch controllers and in addition improving the speed to the tool.

The restriction put on the interface between the sub-blocks is that all interface signals have to (seem to) originate directly from a scan cell. In this way, all interface

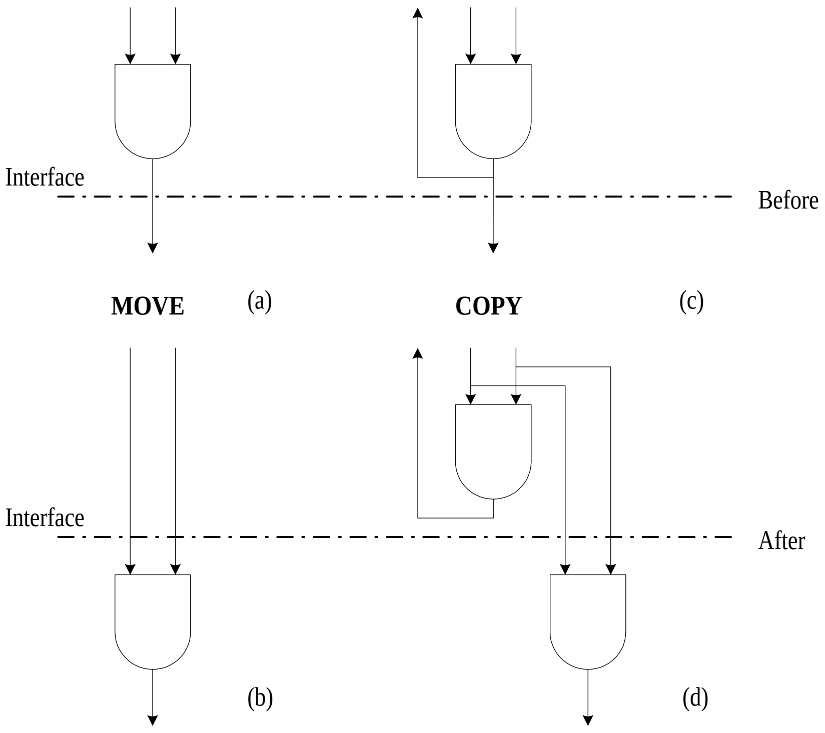


Figure 4.10: Move and copy operations to enable protocol expansion.

signals can be directly controlled from the scan chain. To fulfill this restriction, logic gates that are located between the scan cell and the interface pin are moved or copied between the sub-blocks as shown in Figure 4.10. In Figure 4.10(a), a logic gate is shown that only drives an interface pin and no other gates in its own block. This gate is moved to the other block, as shown in Figure 4.10(b). When the gate also drives other gates in its own block (Figure 4.10(c)), it cannot be moved and it is copied instead (Figure 4.10(d)). The block is only copied in the remodel netlist; it does not introduce additional gates in the real netlist. In rare cases where a conflict with a latch controller occurs, this requires an additional scan element in the signal or a redesign of some handshake components. If all interface signals originate from a scan cell, the circuit has a structure as depicted in Figure 4.11. All interface signals can be directly controlled from the scan chain.

To complete the test-protocol expansion the interface signals also need to be observed during scan. This is required because the ATPG tool includes the expected responses of the interface signals in the pattern file. However, observing these signals does not result in a higher fault coverage. Any fault on one of the interface signals is already detected if the signal is used to control a pin. Furthermore, the output value of the scan element from which the interface pin originates can be observed

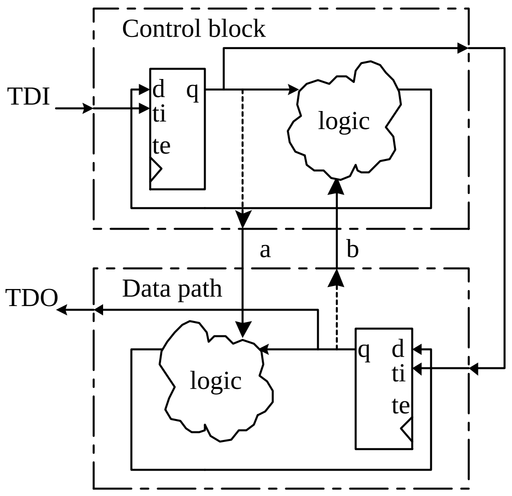


Figure 4.11: Protocol expansion in the modified circuit.

by scanning out the value via the scan chain. As a result the interface signals do not need to be observed during test.

To implement the previous approach, the ATPG remodelling has to be changed in order to avoid the generation of observability requirements for the interface lines. This is accomplished by removing the outgoing interface signals from the remodelled sub-blocks, indicated by the dotted lines in Figure 4.11. It is not required that these signals are observed and they are not traced during protocol expansion.

Normally, the tracing of signals provides the information on how to control a signal. Since the output interface signals are now disconnected, it is no longer possible to trace the input interface signals. This is solved by adding this information to the abstract description of the scan chain that is used for the protocol expansion. The description is automatically generated by the test tools described in the appendix A. For each interface signal it is specified which scan element is connected to it. Since all interface signals are connected to scan elements this is always possible.

The test-protocol expansion tool is only given the abstract descriptions of the scan chains in the sub-blocks and a top-level netlist containing empty shells of these blocks and the interconnections between the blocks. This is sufficient information to create valid expanded top-level protocols.

4.3 Additional modifications

The circuits that were described so far did not include some structures that can occur in real handshake circuits. These structures make testing more complicated, but do not change the principles of the test method previously described. The subjects that complicate testing are:

- Initialization
- Combinational loops
- Mutex elements
- Multiplexers
- Redundancy

These problems require additional test modifications, such as more scan elements. In some cases the problems can be avoided by redesigning certain handshake components. In that sense, testing can be considered to be a new dimension to evaluate the design of a handshake component.

4.3.1 Initialization

The first issue that has not yet been addressed is that of reset and initialization. In handshake circuit this requires two things:

- Initialization of the control block
- (P-)Reset of sequential elements in the circuit

The initialization of the control block is based on the initialization property of handshake circuits [8]. Handshake components are initialized by lowering the request signals of its passive ports. The component will subsequently lower the request signal of all its active ports, thereby initializing the components connected to it. This continues until passive components are reached that do not have active ports. Initialization is then completed in the reverse direction, by lowering the acknowledge signal of the active ports. The top-level component is connected to the start-up channel. Therefore the start-up channel is used for initialization by making it low. After the circuit is initialized, the start-up signal is raised, which functions as the initial handshake from which all other activity follows.

The data-path registers and a few of the handshake components in the control block cannot be initialized in this way. These handshake components have to use (re-)setable memory elements that are directly connected to a reset signal. Since the function of this reset signal is the same as the function of the request of the start-up channel, in a handshake circuit these two signals are combined into one signal.

If the circuit is viewed from a scan-test perspective, then the function of the start-up channel is no longer the same as the function of the reset signal. For scan testing, the start-up signal is just a normal data input to the circuit. As a result the start-up signal can no longer be shared with the reset signal and two separate signal are required.

Besides the use of a global reset, it is also possible to reset the variables of a function every time the function is called. In the circuit this leads to locally generated reset signals. To make these signals controllable during the scan-test, these reset signals have to be reconnected to the global reset signal. This can be implemented with a multiplexer that is controlled by a dedicated control signal. This control signal has to be active during the entire test. Unfortunately this reduces the test coverage of the circuitry that generates the local reset signal. To prevent this, additional scan elements can be inserted. A better solution is to avoid the situation altogether by not programming functions with initialized variables in Tangram. Instead, if a variable needs to be initialized it can be assigned an initial value at the beginning of a function. Both adding scan elements and redesigning the circuits increases the circuit area.

4.3.2 Combinational loops

In Tangram it is possible to create a program that leads to a combinational loop in the handshake control circuit. An example is shown in Listing 4.1, which results in a feedback loop around a NOR gate. Although this program is not practical, since it does not do anything useful, it illustrates that combinational loops can be programmed in Tangram. Combinational loops are not common, but when they occur they would have a serious impact on the achievable fault coverage. The ATPG tool will mark the loop as untestable. In addition it is likely that a number of faults in the surrounding logic can also not be tested.

Listing 4.1 Tangram program that leads to a combinational loop

```
forever do
  skip
od
```

To remove combinational loops from the circuit, the circuit is analyzed and any loop that is found is broken by inserting a scan element in the loop. Most loops occur in known constructs that are easily recognized and for which a pre-designed circuit modification exists that removes the loop. The circuit modification consists of adding a new scan cell somewhere in the loop. If the type of loop is not recognized, a transparent scan flip-flop is inserted in the loop.

4.3.3 Mutex

As explained in Section 2.2.5, the function of a mutual exclusion (mutex) element is to prevent that both its outputs are active at the same time. If both request inputs be-

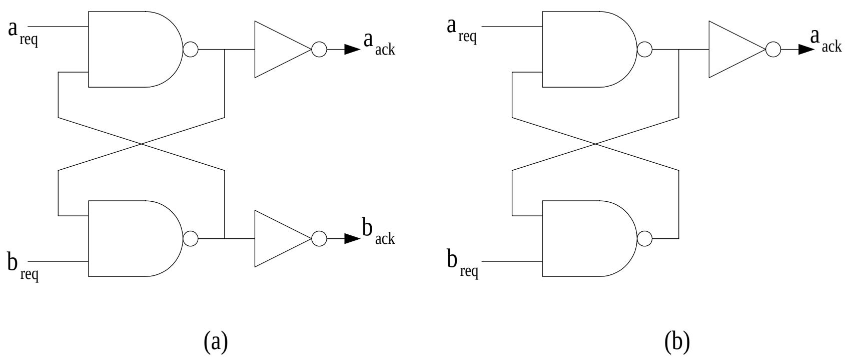


Figure 4.12: Original mutex elements, including in (b) a version with only one output.

come high simultaneously, the mutex must arbitrate between them and only activate one of its outputs. A possible metastable situation that can result has to be kept inside the element until it is resolved. The implementation of the mutex is shown again in Figure 4.12(a). Figure 4.12(b) shows an alternative with only an a_{ack} output that is also used in handshake circuits.

For testing the major problem with mutex elements is the undeterministic behavior during test (Section 2.2.5). A possible solution to this problem is to modify the element in such a way that it becomes deterministic in test-mode. One way to accomplish this is by disabling the feedback loops in test-mode, as shown in Figure 4.13(a). In test mode ($tm = 1$) this circuit behaves as two independent buffers. During test generation it is remodelled with two buffers as shown in Figure 4.13(b). This modification has an impact on both the fault coverage and the performance of the element. Because the test-mode signal has to be active during the entire test, the gates driven

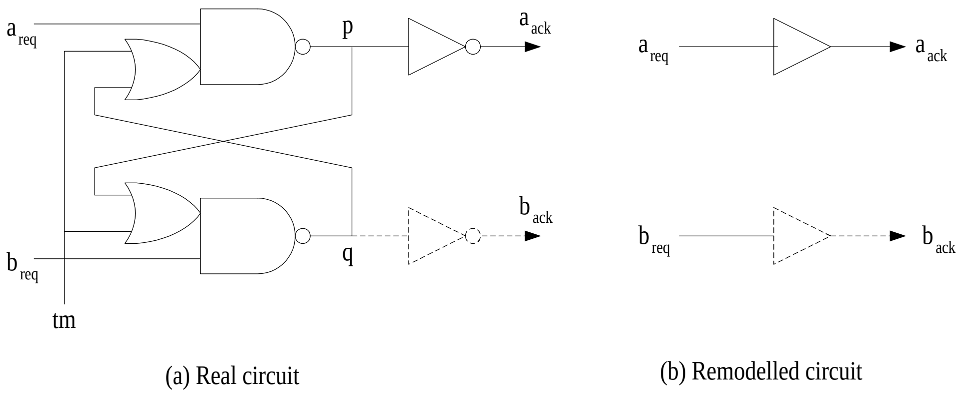


Figure 4.13: Test Mutex in which both NAND gates are disabled during test, (a) circuit, (b) remodel circuit.

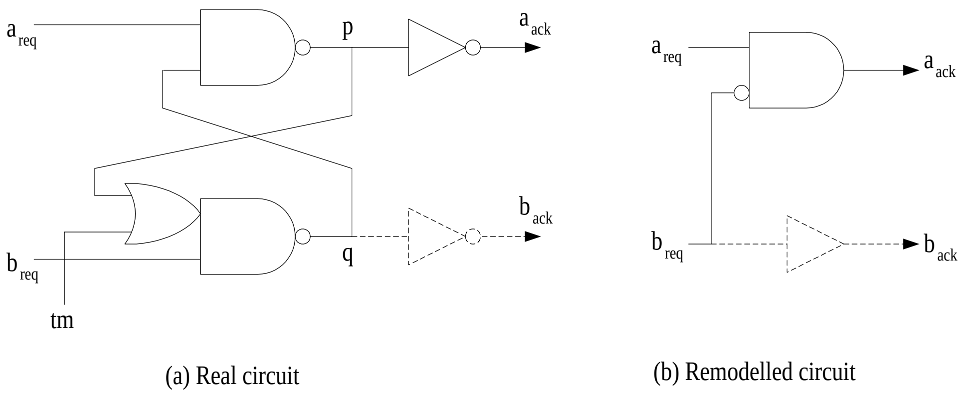


Figure 4.14: Test Mutex in which only the bottom NAND is disabled during test resulting in higher fault coverage, (a) circuit, (b) remodel circuit.

by this signal are not tested. In the case of the mutual exclusion element this is also the case for the feedback loops in the circuit. The increased logic depth may also influence the performance of the element, both in terms of overall speed and the time required to resolve metastability.

For the mutex alternative with only the a_{ack} output, the modification in Figure 4.13 results in further reduced fault coverage. The b_{ack} output cannot be observed and therefore the logic controlling b_{req} cannot be tested.

By only disabling one of the feedback signals this problem can be avoided since now output a_{ack} is a function of both inputs. This structure is shown in Figure 4.14.

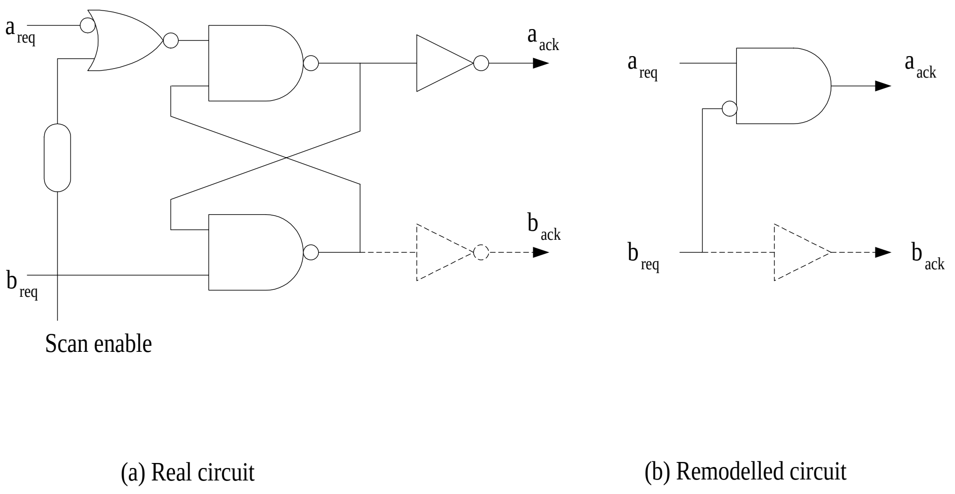


Figure 4.15: Delayed evaluation mutex, (a) circuit, (b) remodel circuit.

A potential problem of this solution is that the cross-coupled gates are no longer identical. The implementation in Figure 4.15 uses a different method to obtain a deterministic response. The main advantage is that it uses the original NAND-based mutex. During scan shifts the top NAND input is kept at zero which will force the mutex in a deterministic initial state. During the evaluation cycle, the a_{req} input is released, however only after a delay to make sure that the circuit inputs have stabilized and no hazards can occur that might change the internal state of the mutex.

4.3.4 Multiplexers

Multiplexers are used in case a variable handshake component has multiple write sources. An implementation for a two channel multiplexer is given in Figure 4.16. This can be generalized for multiple channels depending on the number of locations that need to write into the variable.

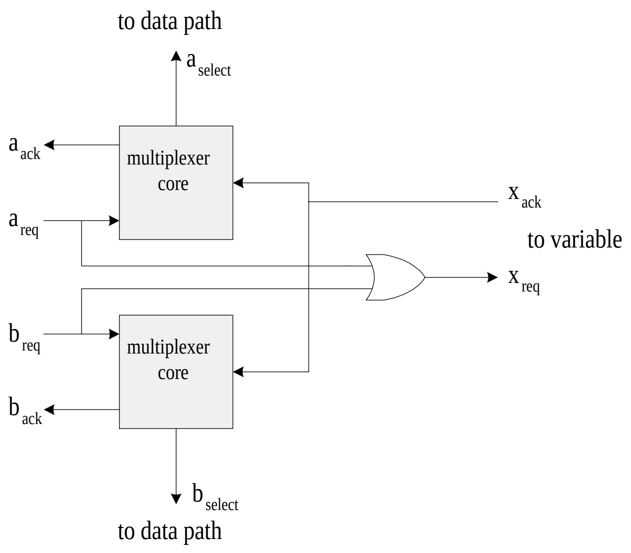


Figure 4.16: Implementation of the multiplexer component, using one multiplexer core per input channel.

Multiplexers are activated by one of the passive channels, channel a or b in Figure 4.16. If a passive channel is activated, the corresponding select signal is raised and a request is passed on to the variable via channel x. The surrounding circuitry has to ensure that only one passive channel of the multiplexer is activated at once. Every passive channel is connected to a multiplexer core that will generate the select signal and the acknowledge for that channel. The multiplexer cores can be implemented in a number of alternative ways, as shown in Figure 4.17.

The select signals generated by the multiplexer cores are connected to the data

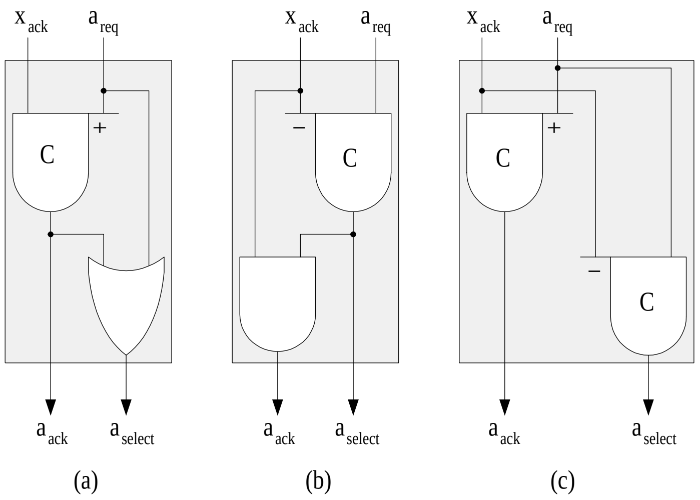


Figure 4.17: Alternative implementations for the multiplexer core, (a) original version, (b) version modified for protocol expansion, (c) version modified for protocol expansion and the prevention of global loops.

path. They are used as parameters for the data-path logic that calculates the new value for the variable. Since the select signals form an interface signal between the control block and the data path, they are analyzed during test generation to determine whether or not they originate from a scan element, as explained in Section 4.2.2.

In the implementation of the multiplexer core shown in Figure 4.17(a) this is not the case. The select signal comes from an OR gate, and a back-trace procedure is started to find out which gates need to be moved or copy to the data path. Since the select signals of the multiplexer are the most common data-path parameters, this leads to many and often large back-trace procedures to find scan elements to use in the test-protocol expansion. In fact, this often result in a failure if a signal is traced back to a latch controller.

To solve this problem the multiplexer core implementation shown in Figure 4.17(b) can be used. In this implementation the select signal is generated in a scan element and no back-trace is required. However the circuit can potentially result a combinational loop in the control block. For example, a loop can start from the request a_{req}, go via channel x to the latch controller and arrive in the multiplexer core via the x_{ack} input. Then it is passed on to a_{ack} and from there it is sometimes possible to find a combinational path back to a_{req} to complete a loop. If such a loop is identified, by the analysis described in Section 4.3.2, the third alternative implementation shown in Figure 4.17(c) can be used. This implementation is based on two scan C-elements and therefore more expensive to use, but since it is only required if there is a combinational loop, the additional scan element would be required anyway.

4.3.5 Redundancy

During some initial experiments with the scan method, it was found that the achievable fault coverage was below expectations. Analysis showed that this was due to redundant logic in the control block. The number of untestable gates because of redundancy is typically smaller than 1%. Although the individual handshake components are not redundant, redundancy can occur if two or more handshake components are connected together. At the interface between the components some gates may become redundant. The reason for this is that the combination of the two components can restrict the freedom of the individual components. The components in isolation are designed to handle situations that can no longer occur if components are connected together.

There are two ways to remove the redundancy from the circuit. The first is to choose an alternative implementation for (one of) the components. The second is to analyze the redundant components together, to find new optimization rules. These optimization rules are then added to the Tangram compiler, thereby stepwise improving the compiler. A small disadvantage of this approach is that the link between the original Tangram source code and the final circuit netlist is becoming less obvious. This complicates the identification of specific signals during simulation and debug.

In the next section, two examples are given in which redundancy is present in a handshake circuit. In the first example the components are analyzed together and this leads to a circuit optimization. In the second example, a handshake component is redesigned.

Interfaces between a passivator and a sequencer

The combination of a passivator component and a sequencer component is used in for example the implementation of FIFO buffers. The redundant part of the implementation is shown in Figure 4.18. Whenever the b_{req} signal is lowered, the b_{ack} signal will follow after a short delay. This leads to an untestable stuck-at-0 fault at the input of the NOR gate in the sequencer. To test this fault the output of the C-element has to be zero and b_{ack} has to be one. However by making the C-element zero, b_{req} becomes low and in turn also b_{ack} becomes low. This leads to a contradiction and as a result to an untestable fault.

Handshake-level analysis of the two components combined showed that it is safe to resolve this redundancy by replacing the NOR gate with the untestable fault by an inverter. The resulting circuit will already activate channel c during the return-to-zero phase of channel b. This particular optimization was known to exist [48], but the testing result showed that the optimization is mandatory to eliminate redundancy from the circuit.

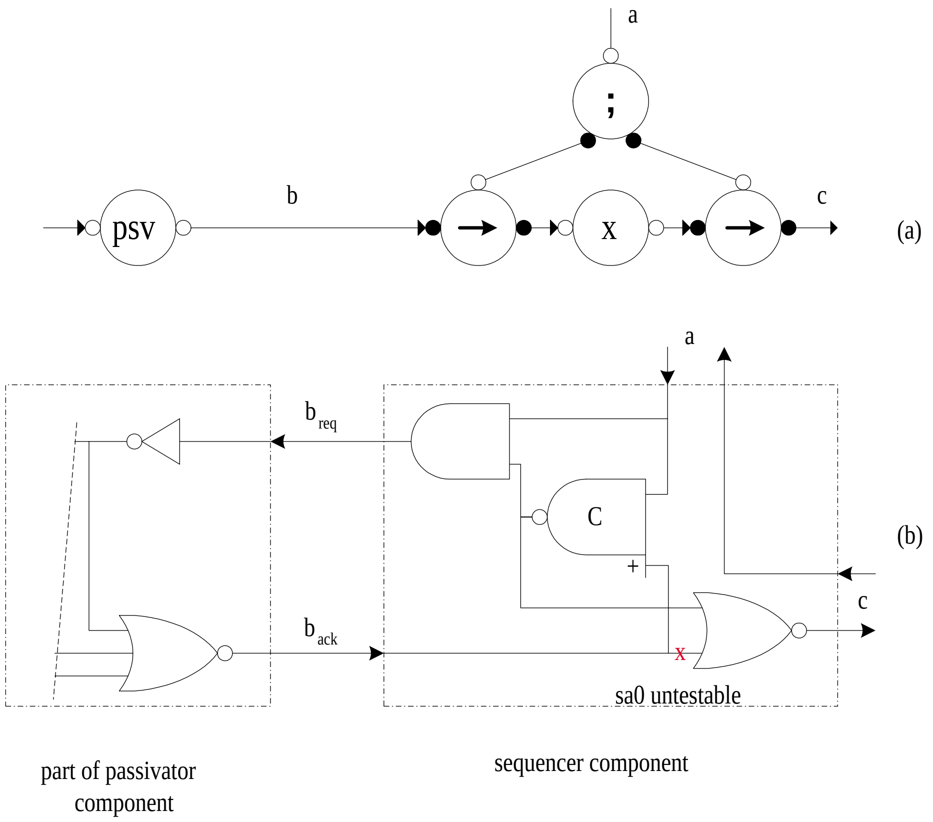


Figure 4.18: Section of FIFO circuit containing redundancy (a) handshake circuit, (b) gate level implementation.

State encoding in a multi channel sequencer

One reason for the small amount of redundancy in Tangram based handshake circuits is the sparse state encoding used in Tangram. All the state information of a Tangram circuit is stored locally in the handshake components and channels. Because every component is designed as a separate entity, this state variable can be deeply embedded in the design. The disadvantage of this method is of course that it requires many state variables. In order to reduce the cost of state variables, several large multi-channel components have been designed. Most significant of these is a multi-channel sequencer [4] that uses an internal Gray counter that contains halve the number of state variables. An implementation of a four channel sequencer is shown in Figure 4.19.

Although this circuit is smaller than a tree of two-channel sequencers, it can result in redundant circuitry whenever a combinational path exists from the request of a channel to the acknowledge of the next channel. In the figure, one such a combi-

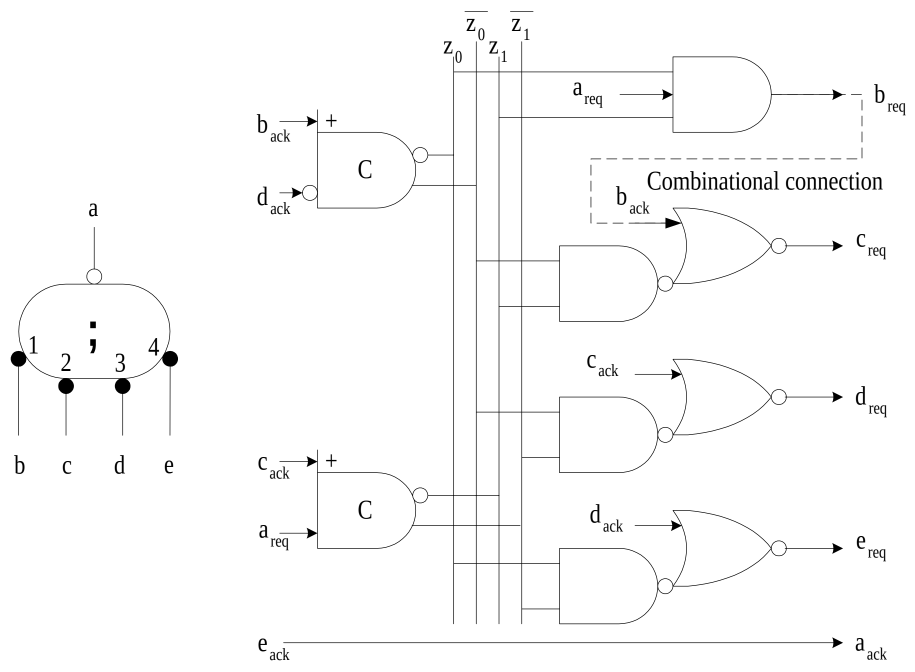


Figure 4.19: Implementation of a multi channel sequencer (a) component symbol, (b) gate-level implementation.

national path is shown, indicated by the dotted line. In this example the b_{ack} input of the NOR gate driving c_{req} is redundant, as shown in Equation 4.1:

$$c_{req} = \overline{b_{ack} + \overline{\overline{z0} \cdot z1}} = \overline{z0 \cdot a_{req} \cdot z1 + z0 + \overline{z1}} = \overline{z0 + \overline{z1}} = \overline{z0} \cdot z1 \qquad (4.1)$$

The redundant logic can be removed, if it can be proven that the resulting circuit is still functionally correct. Unlike in the previous example this is not directly clear since there are more possibilities in which this type of redundancy can occur. Instead, to remove this redundancy original two channel sequencers are used in multiple levels. Since the multi channel sequencer is only redundant when connected to a combinational path, more accurate circuit analysis can be used to identify these channels. Depending on the analysis either the redundancy can be removed or a two-channel sequencers is used for these channels. For the other channels the multi channel sequencer can then be used without problems.

4.4 Test control logic

The scan testable handshake circuit can contain four types of scan elements:

1. Scan C-elements
2. Scan latches
3. Scan flip-flops
4. Transparent scan flip-flops

The clock and test-enable inputs of these elements have to be connected to a minimal number of global control signals. Figure 4.20 shows how all scan elements can be controlled by using five top-level signals. The scan C-elements use two LSSD style non-overlapping clocks that are generated from clk_{master} and te with a demultiplexer. The clock signals for the latches and flip-flops in the data path are connected through the latch controllers.

The circuit can operate in four modes. The signal state of the four test signals during the four possible operational modes is given in Table 4.1. In the asynchronous mode, the circuit operates as an unmodified asynchronous circuit. All top-level signals have a fixed value. The other three modes are used for testing. In these modes the clocks are active, indicated by the c (for clock) in the table. In each of these modes the circuit will go through several states, governed by the clocks, to either capture evaluation data or perform scan shifts.

Table 4.1: Test-signal definitions for the various operational modes.

Mode	clocks (3x)	tm	te	LCS
Asynchronous	1	0	0	0
Scan shift	c	1	1	1
Evaluation Control	c	1	0	0
Evaluation Data	c	1	0	1

Every latch controller drives several latches or flip-flops and therefore the number of latch controllers is small compared to the number of latches and flip-flops. Hence their impact on connections to the test signals is also small. The scan enable signal is shown bold in Figure 4.20; the wiring required for this signal is the same as would be required in a synchronous circuit. The wiring for the scan-chain routing is also the same and for clarity is not shown in the figure.

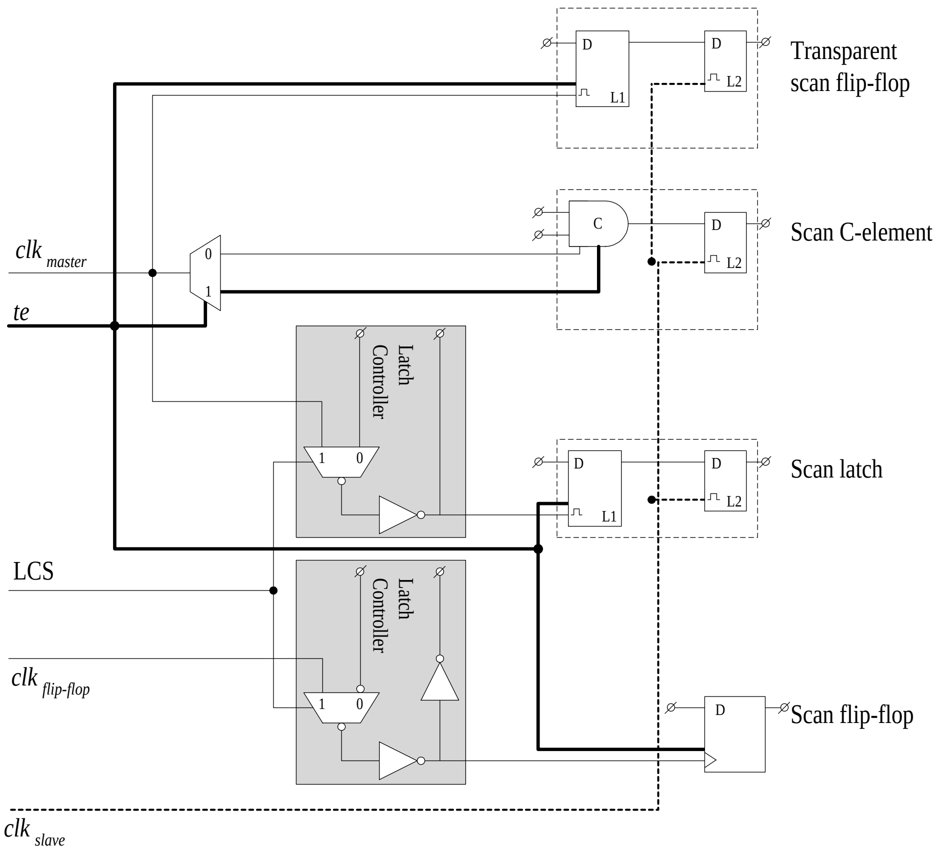


Figure 4.20: Test control logic, showing the four types of scan elements and their global connections. In total five top-level test signals are used to control all types of scan elements.

If primitive LSSD flip-flops and L_1L_2* testing are not used, it is more efficient to design circuits that use flip-flops instead of latches. A flip-flop is smaller in area than the combination of a master latch and a slave latch. By only using flip-flops also the amount of wiring can be reduced since in that case many connections to the slave clock are removed.

Finally test mutexes and asynchronously re(setable) elements can be present. For the re(setable) elements a reset signal has to be added and for the mutex a test-mode signal is required. If all possible elements are present, seven test signals are required to control the circuit during test. In the next sections optimizations are introduced that will reduce this number of signals.

4.4.1 Test control optimization

In this section three optimizations are shown that can be used to reduce the number of top-level test signals required for the test method. First the optimizations are discussed and afterwards all optimizations are combined with the testable handshake circuit to show the reduction in the number of test signals.

Clock generator

The largest reduction in top-level pins can be obtained by using an on-chip clock generator as was shown in Figure 3.23. From the timing analysis of the latch controllers followed that the option in Figure 3.23(b) has to be used for safe circuit operation. The circuit shown there, however, does not support the required asynchronous mode in which both the master and the slave clock have to be constant one. In Figure 4.21 this is solved by using the test-mode signal tm to make the slave clock one. The master clock can be kept at one by making the reference clock zero. Alternatively, the master clock can also be controlled by the test mode signal just like the slave clock. In that situation the reference clock can be undefined during asynchronous mode. The flip-flop clock may also be undefined, since it is only connected to the latch controllers.

If the on-chip clock generator is used, it becomes difficult to ensure that the latch control select signals LCS switches at the correct time, which is when both the master clock and the slave clock are zero. A solution is to derive the signal LCS_{int}

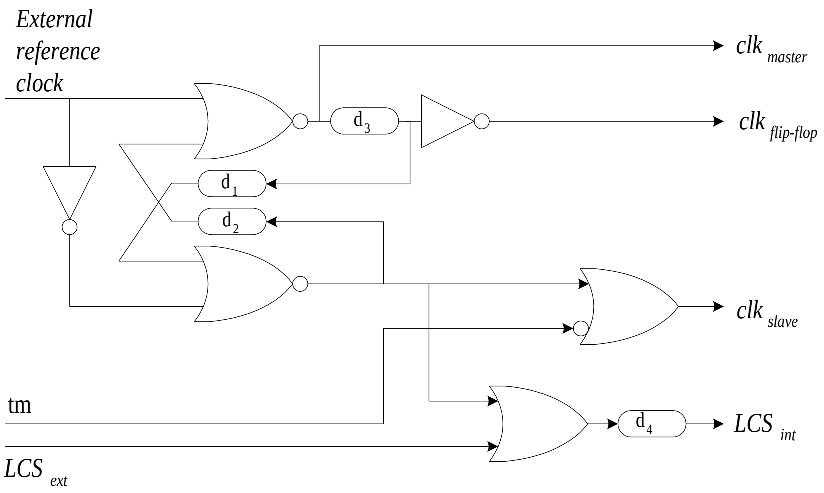


Figure 4.21: On-chip clock generator, able to generate three clock signals from the external reference clock. In addition the circuit can be used to generate a LCS signal with the correct timing parameters.

from the slave clock and an external select signal LCS_{ext}. This solution is shown in Figure 4.21. It ensures that the LCS_{int} signal cannot switch to zero before the slave clock has. Delay $d4$ in the clock generator should be about half of the total non-overlap delay $d2$, after which the master clock will become one. The location of rising edge of LCS_{int} in the clock cycle is not important as long as it occurs after the falling edge of the master clock.

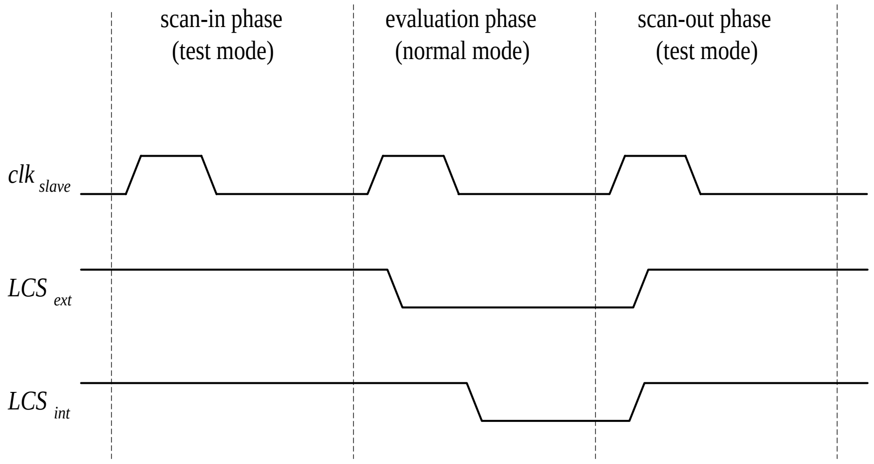


Figure 4.22: Generating the latch control select (LCS) signal with the on-chip clock generator from conventionally timed input signals.

A major benefit of using the clock generator to generate the LCS signal is that the timing of the external select signal LCS_{ext} is less stringent. The original LCS had to switch at around 50 % of the clock cycle, which is a non-typical setting and can complicate the integration with test for other blocks that use the default timing parameters, in which signals change at the beginning of the clock cycle. In the new situation the test for the handshake block can also use the default timing parameters. The only minor restriction is that the falling edge of LCS_{ext} has to occur after the rising edge of the slave clock. The timing diagram of this situation is shown in Figure 4.22. The falling edge of the LCS_{int} signal still occurs at around 50 % of the clock cycle, but it is derived from input signals that have the default timing. Another benefit is that this allows all other data and select input signals to also switch at the beginning of the cycle. This means that there is more time for these signals to propagate through the circuit, which in turn can be used to increase the test clock speed.

Reset

In Section 4.3.1 it was discussed how a new reset signal is added to the circuit if there are re-(setable) elements present. The circuit shown in Figure 4.23 can be used

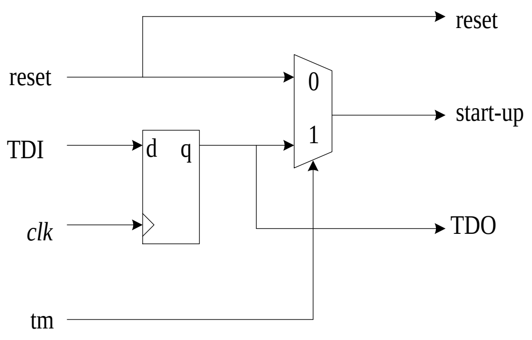


Figure 4.23: Driving the start-up signal with a scan element during scan mode. Reducing the number of external signals by one.

to reduce these two signals back into one top-level reset signal. In asynchronous mode both the start-up signal and the internal reset signal are directly controlled by the external reset signal. During test, the start-up signal is a normal data input of the handshake circuit and the data for this signal is provided by a scan element. The signals used to control the scan element are already present for the test of the handshake circuit itself and no new test signals are added.

Test control block

Large chips often have a Test Control Block (TCB), which can be programmed by the tester to put the chip in different test modes. When such a TCB is already present on a chip it can be used to save an additional top-level pin. Figure 4.24 shows how one bit in the TCB can be used to generate the both the LCS and the internal test

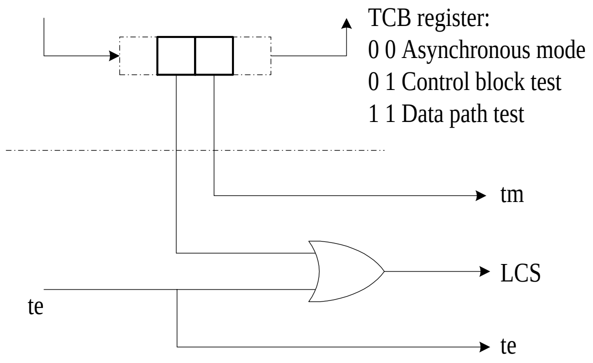


Figure 4.24: Test control block, in which one bit is used to select between control block test and data path test and a second bit is used to control the test mode signal.

enable te signal from a single external test enable te signal. A second bit in the TCB can be used to put the chip in test mode. If a TCB is used for the LCS signal it must be reprogrammed to switch between control block test and data path test. As an alternative for the TCB, the LCS signal can also be generated from the OR function of a scan element and the test enable signal.

Global test control

The three test control optimizations are shown combined in Figure 4.25. When all possible optimizations are used only two new top-level signals are required, the reference clock and the test enable, both of which use the default timing parameters. The reset signals is also shown at top-level, but this is not a new pin since it replaces the original reset pin. Also the scan-in pin (TDI) and scan-out pin (TDO), both not shown in the figure, can be multiplexed onto existing data pins.

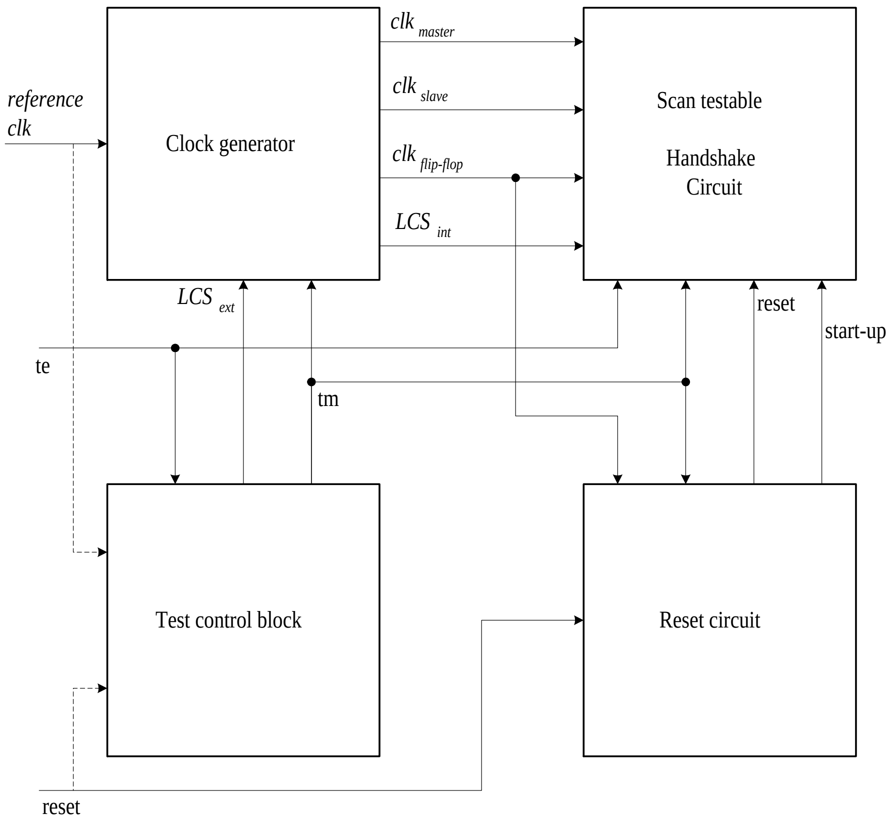


Figure 4.25: The total number of global control signals can be reduced from 8 down to 3 by using the proposed optimizations.

Logic optimization

The resulting test control logic can be further optimized with simple logic optimizations. One useful optimization is the partial sharing of the latch controller multiplexers, which is shown in Figure 4.26. Since one of the inputs to all multiplexers is always the same, this gate can be shared. A further optimization is to change the polarity of the LCS signal which results in an implementation that uses less inverters.

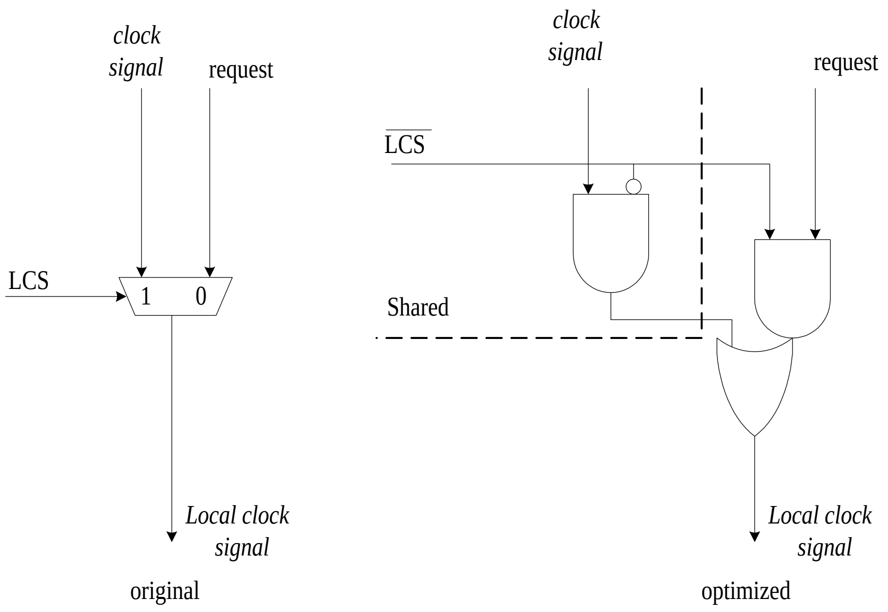


Figure 4.26: Logic optimization of the multiplexers in the latch controllers, an AND gate is shared by all multiplexers and LCS is inverted to reduce the number of inverters.

4.5 Implementation of the scan-test flow

One of the objectives for the test method was that it could be supported by an automated test flow. This means that for the scan-test method to be of practical use, the test method has to be automated. The design and test engineers should spend minimal time in creating and debugging the test. The circuit and test modifications are too complex and error prone to be done manually. During the development of the test-flow, several important guidelines and requirements were followed:

Short development time. By focussing on the specific modifications required for the testing of handshake circuits and using existing tools for all other functions, the development time for the entire flow is minimized.

High ATPG quality. Primary concern is the obtainable fault coverage, but also important is the minimization of the test time by generating a minimal number of test patterns. Both can be achieved by using existing and proven ATPG tools.

Compatibility with the test infrastructure. The test has to be executed on standard and available test equipment and should not require modification of this equipment.

Compatibility with other IP blocks on chip. Handshake circuits are often embedded in larger SoC designs. This means that the test of the handshake circuit should be compatible with the overall system test. This can be accomplished, by using the same interfaces and having the same test equipment requirements and the same test-signal timing definitions.

4.5.1 Test-flow overview

The full-scan test approach leads to the test-flow shown in Figure 4.27. It shows a newly developed tool: *TgScan*, which performs all DfT modifications and creates input files for the other tools in the flow. These other tools are all existing tools used within Philips and known as Computer Aided Test (CAT) tools. These tools have been developed and widely used for the test generation of synchronous circuits. Three existing tools will be used, namely an automatic test-pattern generation (ATPG) tool, a protocol-expansion tool, and a vector-generation tool. Two ATPG runs are carried out, one for the control block and one for the data path. The resulting two initial protocols are expanded to create top-level protocols and finally combined with the test patterns in the vector-generation phase. The resulting vectors can be simulated. Simulation can be done with a functional simulator to assess the functional correctness of the circuit. Alternatively, the simulation can be performed with a fault simulator to determine the fault-coverage achieved by the test vectors on the real scan-testable netlist. The usage of the CAT tools is discussed in detail in appendix A.

4.5.2 Compiler modifications

As discussed in Section 4.1.3, the test approach is based on two separate tests, one for the control block and one for the data path. The original netlist generator of Tangram, however, creates a single flat netlist. It would be possible to have the test tools identify the control block and data path, but a more elegant solution is to modify the compiler to generate a hierarchical netlist in which the control block and data path are located in separate hierarchical levels. This was implemented by redesigning the handshake components to include special transfer gates (consisting of a wire), which are used to identify the transfers between the control block and the data path. The compiler recognizes these transfer gates and uses them to separate the control block from the data path.

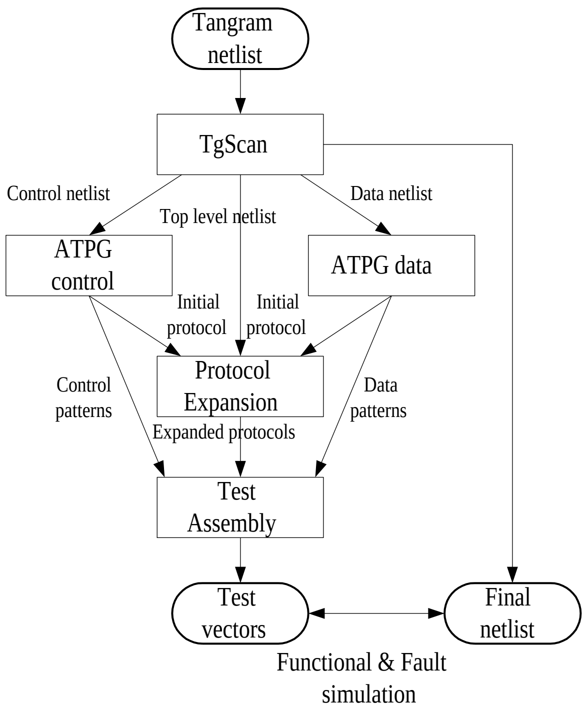


Figure 4.27: Test flow for handshake circuits, consisting of the new tool TgScan and a number of existing CAT tools.

The compiler was further optimized to remove many of the problems with redundant circuitry, as identified in Section 4.3.5. New implementations were added, for example for the multiplexer and the multi-channel sequencer components. Several new optimizations have been implemented to remove the redundancy caused by connecting these handshake components. Finally, the fan-out optimization is postponed until after the scan modifications are added, since the scan modifications will change the load of the signals.

4.5.3 TgScan

The central tool in the test flow is the newly developed TgScan. This is the only newly developed tool and it forms the link between the handshake circuit tools and the existing CAT tools; hiding the asynchronous internals from the CAT tools. TgScan achieves two things: it creates a scan netlist and it creates remodelled netlists and control files for the CAT tools.

Figure 4.28 shows the main tasks that are performed by TgScan. After the first four common tasks, the activity in the tool splits into a number of tasks that result in the scan netlist and a number of tasks that results in the CAT files.

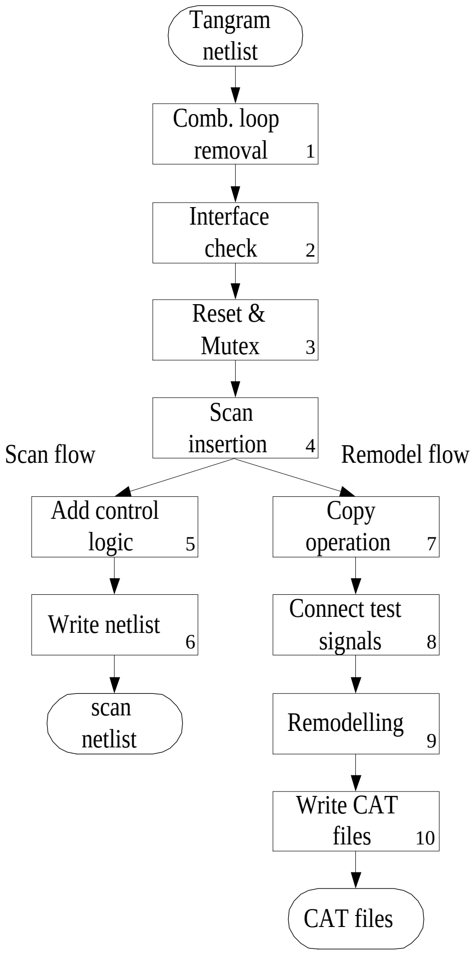


Figure 4.28: Flow diagram of TgScan. Input: Tangram netlist, Outputs: Scan netlist and CAT files.

1. **Combinational loop removal.** The first task is the identification and removal of combinational loops. Once a combinational loop is identified, it is determined if the loop is of a known type. Currently two types are recognized: the loops caused by multiplexers and loops that can occur in combination with a mutex element. The removal of the loops caused by the multiplexers has been discussed in Section 4.3.4. For the other types, a transparent scan flip-flop is inserted in the loop.

2. **Interface check.** For a successful expansion of the test-protocols of the control block and the data path, all interface signals between the control block and the data path need to originate from a scannable element. As described in

Section 4.2.2, this requires moving and/or copying gates from the control block to the data path and vice-versa. The implementation of this function uses a list of interconnection signals. Signals are removed from this list if they originate from a scan element. If a logic gate is found that has to be moved, then it is moved to the other block and the interface signals are adjusted accordingly. If a logic gate is found that has to be copied, it is labelled as such but not yet copied since that should only be done in the remodelled netlist and not in the real netlist. Again the interface list is updated, the inputs of the gate are added to the list of interface signals, the output of the gate is no longer an interface signal and is removed from this list. In case no more blocks can be moved or copied, the result is a netlist in which all interface signals either come from a scan element or from a logic gate that is labelled to be copied later. In some cases it is also possible that a signal is traced to a primary input; in that case the primary input can be used to control the signal directly.

3. **Reset & mutex.** The scan elements that have a set or reset input and the mutex elements have to be modified for test. The set or reset signals are connected to a new independent global reset signal. Some scan elements might be connected to an internally generated reset signal. In these cases, a second test signal is used to control a multiplexer that disconnects the internal reset signal from the scan element and reconnects it to the new global reset signal. Mutex elements are replaced by test versions and connected to the test mode signal as explained in Section 4.3.3.

4. **Scan insertion.** All state-holding cells are replaced by their scannable equivalents. The scan cells are connected together to form a scan chain. Compared to other scan insertion tools, the functionality to tune the scan insertion is still very limited. There is neither support for shift register recognition nor for the balancing of scan chains. Future versions of TgScan will have to incorporate these functions or use an existing scan-insertion tool for this part of the flow.

After the first set of tasks, the activity is split into two separate flows. The first is the scan flow that produces the final scan-testable netlist. The second is the remodel flow, resulting in the remodelled netlist and CAT control files used for test generation.

Scan flow

The data structure representing the netlist already contains an almost complete scan-testable circuit. There are two remaining tasks:

5. **Add control logic.** The global control signals and logic are added to the circuit and connected to the scan elements.

6. **Write netlist.** The scan netlist is completed and written to the scan-netlist file.

Remodel flow

For the remodel flow, TgScan still has to perform a number of tasks:

7. **Copy operation.** In the interface check block, a number of gates might have been labelled as copy blocks. They could not be copied before since in that case they would also be copied in the real scan netlist and cause redundancy and additional area overhead. The copy operation uses a similar mechanism as the move operation, only the original cells are not removed. For the cells that are copied, test patterns will be generated in the data-path test as well as in the control-block test. This means that the corresponding cells in the real circuit will be tested twice, which can potentially cause a small increase the number of test patterns. This is not a fundamental problem and can be solved by improving the tools used to generate the patterns.

8. **Test-signal connection.** The remodelled circuit requires different test signals than the real circuit. In the remodelled circuit, the registers are all connected to a global clock. The clock signals are disconnected from the signals coming from the latch controllers and are reconnected to a global clock. The interface pins that were used by the local clocks are also removed. They have no function during test generation. Their correct operation is tested implicitly during the execution of the test. If a fault is present that causes the connections to fail, the scan chain cannot be operated and hence, the scan-continuity test will fail.

   The interface signals between the control block and the data path are all coming from a scan element. As described in Section 4.2.2, these signals are observed implicitly and the ATPG step should not put observability requirements on these signals. This is accomplished by disconnecting the output of the scan element from the interface pin, creating a dangling pin. The ATPG tool will ignore these pins and not generate response values for them. This approach cannot be followed for the signal connecting the scan chain between the data path and the control block. This signal cannot be removed and needs to be observed during both the normal-mode and scan-mode. Currently this is solved by inserting a separate transparent scan flip-flop in the scan chain between the control block and the data path. Both the normal data input and the scan data input are connected to the scan-out pin of the data path. A second function of the transparent scan flip-flop is to act as an anti-skew latch to remove any potential skew problems between the control block and data path.

9. **Remodelling.** For cells that need to be remodelled for test-pattern generation, a remodel cell is defined in the library. In the remodelling step, original cells are replaced by their remodelled equivalents. Two types of cells are remodelled. The first type are scan elements consisting of a C-element or a latch. Such a scan element is remodelled because the ATPG tool does not recognize

the original scan elements. The second type of cells that are remodelled are cells that use redundant inputs. An example of this type of cells are the output buffers of the mutex elements. These cells are remodelled to get a more accurate calculation of the fault coverage of the generated patterns. If these cells were not remodelled, the reported fault coverage would be overly pessimistic.

10. **Write CAT files.** The remodelled netlist is now completed and from this netlist the data-path and control-block netlists are written to file. Finally the CAT control files and the top-level netlist are generated and also written to file.

The operations performed by TgScan are not computationally complex. For the circuits processed until now, containing up to 6k gates, computation time remained well under one minute on a HP workstation. This included the time spend on reading and writing the netlists and other files.

4.6 Summary

The scan method introduced in this chapter, can be used to make any handshake circuit designed with the Tangram toolkit scan testable. The main characteristic is that the control block and the data path are tested separately. This prevents problems with the control-data interface and allows full testability of the interface signals.

Besides the main scan modification that results in the tests for control block and data path, additional modifications are required in the control block to increase its achievable fault coverage. Some of these modifications are implemented in the Tangram compiler to produce alternative designs that are better suited to the scan test method. Other modifications are carried out on the final circuit. These involve, for example, the mutex elements. Mutex elements are essential for the asynchronous operation but require test modifications to be tested with the scan test method.

The scan test method was designed to work with existing test tools for test-pattern generation. To make ATPG work with the circuit, extensive use is made of remodelling. The disadvantage of this approach is that the link between the actual circuit and the generated tests is reduced, which can complicate debugging of the test and the circuit. The final test is generated by a test-protocol expansion tool, already available in the test tools that were used. To make this tool work properly, the handshake circuits are modified to make all interface signals between the control block and data path originate from a scan cell. This hides the internal asynchronous structures from the protocol expansion tool.

The result is a complete scan-test method that includes the automatic insertion of scan elements into a handshake circuit and the automatic generation of a test for that circuit.

Chapter 5

Results

To assess the costs and benefits of out scan-test method, a number of experiments have been carried out. We start with the evaluation of individual scan C-elements, that form the essential basic components for the test method. This is followed by the presentation of the scan test modifications of several complete handshake circuits. Finally some work is presented on a demonstrator circuit that has been developed using the scan test method.

5.1 Scan C-elements

The scan C-elements are the elements that have the most impact on the final properties of the scan-testable circuit. The reason for this is the extensive modification that is required to make a C-element scan-testable, as described in Section 3.3.3. Following these design principles, a family of composite scan C-elements has been developed. During the development of the scan method, the composite scan C-elements were used to test the control block. This was used to demonstrate the feasibility of using a scan method to test handshake circuits. A library with primitive cells was developed in a later stage by Philips Semiconductors. This library contains primitive versions of seven different C-elements both in a latch and a flip-flop version. These seven C-elements represent over 95% of all C-elements that are present in a typical handshake circuit design. The remaining C-elements are implemented as composite cells, either solely based on logic gates or on one of the primitive scan C-elements customized with logic gates. The two most important properties of the scan C-elements are the cell area and the performance degradation, to be discussed in the next sections.

The primitive scan C-elements were only implemented in one process technology, which is a special-purpose BiCMOS process. All the shown numbers for both primitive and composite cells are for this technology. The process is not targeted at high-speed digital logic. This can be seen in the propagation delay of the gates, which as such can not be directly compared to that of a pure CMOS process.

5.1.1 Cell area

The cell area of the various scan C-elements is shown in Table 5.1. It lists the various C-elements that are used in handshake circuits, identified by the name used in Tangram. The second column gives the area of the original non-scan C-elements. The area is given in the number of grids points used. For reference: a NAND gate uses four grids points in this technology. The next columns give the area of the composite (see Section 3.3.3) and primitive (see Section 3.3.4) scan C-element implementations. Both are available as latch and as flip-flop versions.

The area of the latch versions of the composite cells is dependent on the type and number of combinational logic cells that are available in the used cell library.

Table 5.1: Cell area of scan C-elements in number of grid points and the average area weighted by the relative occurrence in a typical design.

C-element	Non scan	Composite scan		Primitive scan		Relative
name		latch	flip-flop	latch	flip-flop	occurrence
ACZ	7	21	33	13	15	33 %
ACY	8	20	32	14	16	31 %
ACYZ	10	23	35	16	18	0 %
BCZ	7	18	30	13	15	16 %
BCY	7	21	33	16	18	0 %
BCYZ	9	21	33	16	18	3 %
C2	9	26	38	14	16	8 %
C3	16	33	45	16	18	2 %
BCINIT	9	27	39	15	17	2 %
AC56	9	25	40	17	19	5 %
Weighted area	7.9	21.2	33.3	13.8	15.8	
Weighted overhead		170 %	324 %	76 %	101 %	

If a large variety of cells is present, the C-element functions can be more efficiently mapped on these cells. In the current library, some C-elements can be mapped more efficiently than others. This explains the sometimes large difference in area between relatively similar C-elements. The composite flip-flop versions are made from the latch versions using an additional latch. This latch requires an additional 12 grids of area. The difference in area between the primitive latch and flip-flop versions is only two grids. This is because for these elements the slave latch is a dynamic latch, presented in Chapter 3, and because it is integrated with the master latch in a single primitive cell. The last column of Table 5.1 shows the relative occurrence of C-elements in the current circuits. The ACZ and ACY type C-elements together form about 66% of all C-elements. The bottom two rows show the average area and overhead of the C-elements. Both numbers are weighted using the relative occurrence of the C-elements.

In the full-scan method discussed sofar, only the flip-flop versions of the C-element are used. For the composite scan C-element, the slave latch accounts for about half of the total area overhead. The area overhead of the primitive C-elements is about three times smaller than that of the composite C-elements. The difference between latch and flip-flop versions is small for the primitive C-elements, since the slave latch is implemented with a tristate inverter.

If the composite C-elements have to be used (for instance, because the primitive versions are not available), then the L_1L_2* optimization (Section 1.2.3) can save a significant amount of area. For circuits containing primitive scan C-elements, L_1L_2* optimization does not provide such a large area reduction since the difference between the two is only two grids. The potential reduction is about 15%, which is still significant.

Table 5.2: Cell area and area overhead (%) of the latch and the flip-flop.

			Level-sensitive flip-flop	
Element	Non scan	Scan	Composite	Primitive
Latch	12	18 (50 %)	30 (150 %)	20 (66 %)
Flip-flop	17	22 (29 %)		

As a reference, the area of the non-scan and scan versions of the latch and the flip-flop are shown in Table 5.2. Included in this table is the primitive version of the level-sensitive flip-flop as shown in Figure 3.13. This element has not been implemented, its size is estimated from the size of the normal scan latch and the difference between latch and flip-flop versions of the primitive scan C-elements. The estimated area shows that a primitive level-sensitive flip-flop is smaller than a conventional flip-flop. Therefore if this element is present, more efficient circuits can be designed by replacing flip-flops with latches. As can be seen from the table, the area overhead of the latch and flip-flop is smaller than that of the C-element. The main reason for this is the small original size of the C-elements and not the difference in size of the scan elements.

5.1.2 Performance degradation

The performance (speed) of the scan elements is also an important parameter. In Figure 5.1, a simulation trace is shown of an ACZ type C-element, which is one of the most commonly used C-element. Only the normal functional behavior is of importance. Therefore the scan C-elements are put in asynchronous mode by making both master and slave enables high ($en1$ and $en2$ in Figure 3.17) and the test enable (te in Figure 3.17) low. The enable signals are not shown in the simulation. The top two signals show the a and b inputs of the C-elements. The third signal shows the output response of a normal non-scan asymmetric C-element. The next two signals show the output response of the latch and flip-flop version of the composite scan C-elements. Finally the last two signals show the response of the latch and flip-flop version of the primitive scan C-element implementations.

In the simulation, two up and two down transitions are shown. The average delay of these four transitions is given in Table 5.3, together with the relative increase as compared to the non-scan C-element. The numbers give an indication of the real

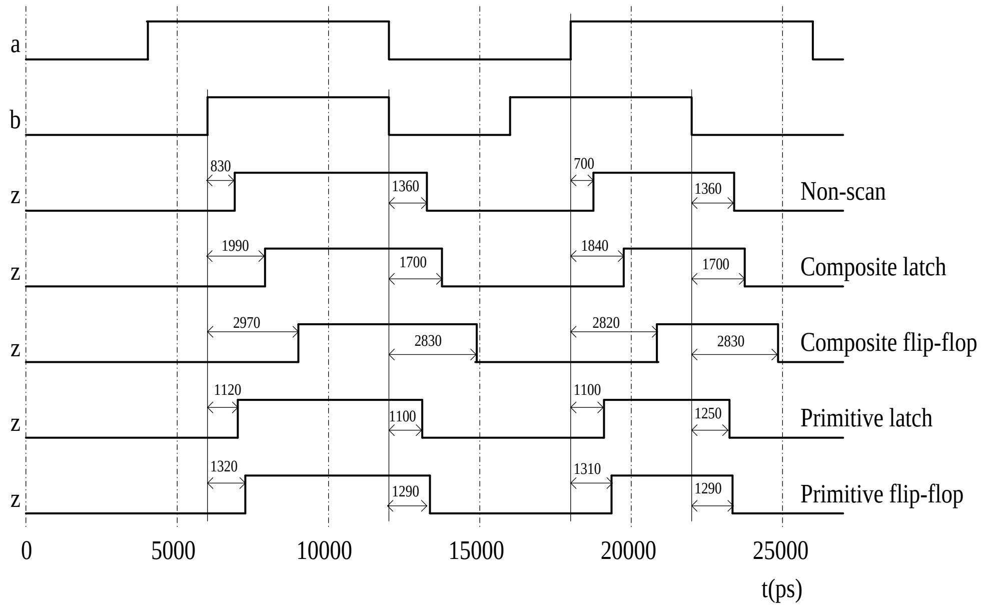


Figure 5.1: Simulation of C-elements, showing increased delay for the scan versions.

Table 5.3: Average delay of the various C-elements.

C-element	Delay (ps)	Delay increase (%)
Non-scan	1060	
Composite latch	1810	70%
Composite flip-flop	2860	170%
Primitive latch	1140	8%
Primitive flip-flop	1300	20%

delay a C-element would have if it is embedded in a circuit.

The slave latch of the composite flip-flop version causes the largest additional delay. As could be expected, the primitive versions are the fastest scan elements. In the falling transitions these C-elements are even faster than the original non-scan C-element. This is likely because larger transistors are used to fill any leftover area inside the designated area for the C-element.

5.2 Full scan

The full-scan method has been applied to a number of industrial designs. These circuits were selected from an existing set of Tangram designs. Among the designs were the DCC error decoder [9] and the 80c51 micro-controller [24]. Together, the examples cover virtually the entire Tangram syntax and are representative of typical Tangram application domains.

The circuits are listed in Table 5.4. Their names indicate the complexity of the circuits expressed in the total number of scan elements. In addition, Table 5.4 provides the number of C-elements, latches, flip-flops and other gates in the circuit. One may observe that the ratio between the various types of gates shows a large spread, reflecting the design decisions made for the benchmark circuits. With Tangram it is possible to actively influence these ratios by changing the Tangram program. The ratio between the types of gates has a significant influence on the cost of the test method for a circuit. In particular, the area overhead is very dependent on these ratios. Three of the circuits: tg60, tg303 and tg839 were designed specifically to make their test more efficient. In the target technology, the primitive implementation of the level-sensitive flip-flop was not available. This is reflected in the absence of latches, since without the primitive level-sensitive flip-flop, latches represent a higher test overhead than flip-flops. Circuit tg164 was developed for minimal circuit area. This leads to an implementation that uses the minimal number of C-elements. A positive side effect

Table 5.4: Benchmark circuits and their complexity in terms of number of gates.

Design	# C-elements	# Latches	# FFs	# Other	# Total
tg60	42	0	18	207	267
tg164	83	53	28	799	963
tg303	217	0	86	928	1231
tg432	113	258	61	1186	1618
tg610	188	324	98	1542	2152
tg839	547	0	292	2408	3247
tg980	578	402	0	1383	2363
tg1163	925	206	32	2424	3587
tg1548	566	940	42	3547	5095

of this is that this circuit can be efficiently scan tested. Circuit tg980, on the contrary, was developed when flip-flops were not yet available in Tangram; it therefore only uses latches. The result is that this circuit can be expected to have a relatively high area overhead.

5.2.1 Fault coverage

Test patterns were generated for the control block and the data path for all the benchmark circuits. The resulting pattern count and fault coverages are shown in Table 5.5. All circuits are relatively simple and the number of generated test patterns is therefore small. In the data path, the fault coverage is between 99.7 and 100%. This indicates that a small number of redundant gates is still present in the data path. The fault coverage of the control block is somewhat lower than that of the data path. There are still some circuits for which the fault coverage in the control block remains under the 98%. There are three main reasons for this:

Redundancy. The current circuits still contain some redundancy. The process of identifying redundancy and implementing solutions to remove it is ongoing. Newer versions of the Tangram compiler will generate circuits that contain less redundancy.

Mutex in test mode. The current treatment of mutex elements, which disables the element during test mode, results in untestable circuitry in and around the mutex elements. To reduce this problem either mutexes have to be tested in

Table 5.5: Fault coverage of the benchmark circuits.

	control block		data path	
Circuit	patterns	coverage (%)	patterns	coverage (%)
tg60	18	92.84	9	100
tg164	54	95.75	102	99.74
tg303	32	93.58	28	99.93
tg432	18	99.24	63	99.98
tg610	16	97.87	43	99.96
tg839	35	95.79	107	99.87
tg980	27	99.13	27	99.91
tg1163	146	98.83	83	99.94
tg1548	40	98.96	80	99.88

their normal mode of operation or mutexes have to be developed that are better testable. The first option requires a complicated ATPG process to avoid problems with non-deterministic behaviour. The second option requires the design of new testable (possibly scannable) mutexes and will require extensive simulations to verify that they still operate correctly in asynchronous mode.

Combinational loops. In the current implementation it is still possible that some combinational loops are not broken by scan. These are typically loops that span more than 10 cells and are therefore very rare. These loops can be removed by improving the TgScan tool.

The impact on the fault coverage of all these reasons listed above can be reduced by further improvements of the Tangram tools. In the circuits tg60 and tg303, a relatively large number of mutex elements is present that are disabled during test as was shown in Figure 4.13. This explains the lower fault coverage in the control blocks of these circuits.

5.2.2 Area overhead

The primary objective of the scan-test approach has been to design a test method for handshake circuits that is able to obtain high fault coverage with a minimal amount of test-development effort. After these objectives have been achieved, it is important to minimize the other costs of the test method. The most important contribution to the cost is the silicon area overhead of the DfT circuitry. As was shown before the available library largely determines the final area overhead. This can also be seen in the following results, in which the scan method was applied to the benchmark circuits using three different cell libraries:

Composite scan. In this case only composite scan elements are used. This is typically used when Tangram is first applied to a new fabrication technology since it can be implemented in any standard cell library and does not require any special additions to it.

Primitive scan C-elements. The first optimization is the addition of primitive scan C-elements. The primitive elements are usually only added if first tests are successful and commercial products are being developed.

Primitive level-sensitive flip-flop. The library may be further extended with a primitive version of an level-sensitive flip-flop. This cell has not yet been added to a library. The resulting figures are calculated using the cell size given in Table 5.2.

The resulting area overhead of the benchmark circuits for the three different cell-libraries are given in Table 5.6. The same results are shown graphically in Figure 5.2,

Table 5.6: Area in NAND equivalents and area overhead (%) of the benchmark circuits for the three alternative cell libraries.

	Non scan	Composite scan C-elements		Primitive scan C-elements		Primitive level-sensitive flip-flops	
Circuit	Area	Area	(%)	Area	(%)	Area	(%)
tg60	425.3	807.3	89.8	632.8	48.8	632.8	48.8
tg164	1649.3	2507.3	52.0	2142.0	29.8	2009.5	21.8
tg303	2071.5	3747.5	80.9	2808.0	35.6	2808.0	35.6
tg432	3054.8	5051.3	65.3	4561.0	49.3	3916.0	28.2
tg610	4559.8	7451.3	63.4	6644.0	45.7	5834.0	27.9
tg839	5550.5	9982.5	79.8	7628.0	37.4	7628.0	37.4
tg980	4475.8	10050.3	124.5	7483.3	67.2	6478.3	44.7
tg1163	5829.0	12843.3	120.3	8821.5	51.3	8306.5	42.5
tg1548	9892.0	17849.1	80.4	15432.8	56.0	13082.8	32.3
average			84		47		36

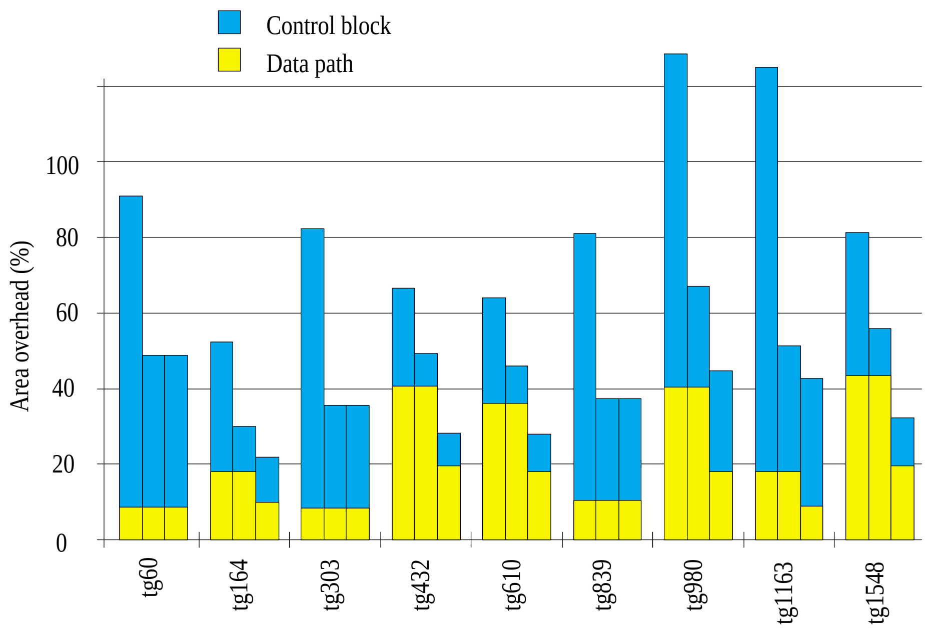


Figure 5.2: Area overhead of the benchmark circuits: first columns for composite scan, second columns for primitive scan C-elements, third columns for primitive scan C-elements and level-sensitive flip-flops.

with the addition of showing the contributions of the control block and the data path to the area overhead separately.

The composite scan overhead ranges from 52% for circuit tg164 to 124% for circuit tg980. This difference between the circuits shows the large influence of circuit design on the area overhead. By taking test into account when choosing alternative implementations, the area overhead can be reduced by half. Of course this is not possible for every circuit; sometimes the function cannot be further optimized. The average area overhead of the circuits using the composite library is 84%. As can be seen in Figure 5.2, most of this overhead is caused by the control block. The overhead of the method when composite scan C-element are used, is so large that it makes it virtually impractical for commercial use.

The addition of the primitive scan C-elements to the library results in a significant reduction of the area overhead. The average overhead is reduced to 47%. For most circuits, the data-path now causes the largest contribution to the area overhead. This is mainly a result of the slave latches that are used to make the latches scan testable. Only in those circuits that use flip-flops in the data path, the control block still causes the largest overhead.

Further addition of the primitive level-sensitive flip-flop results in a decrease of the average area overhead to 36%. This is getting sufficiently low to be of interest for certain commercial applications. Especially those in which the properties of the handshake circuits are important or those in which handshake circuits are combined with other types of circuits.

5.2.3 Performance degradation

The final performance degradation that is caused by the scan method can only be accurately determined after layout. It depends on the design of the scan C-elements, other scan elements, and increased wire capacity.

To get a rough feeling for the expected degradation, circuit tg980 has been functionally simulated at gate level. The functional test exercises the circuit with a set of stimuli, the speed at with the circuit responds to these is determined by the design of the circuit. This simulation only accounts for the gate delays and increased fan-out of some signals. The simulation was done with composite flip-flop versions for the scan C-element, resulting in a worst case situation.

The the time required by the scan version to complete the test was 77% higher that the time required by the original circuit. Compared to the increase of a separate C-element, which was 170% for this type of implementation, the increase was less than half. Based on this, the performance impact on a circuit that uses primitive scan C-elements is expected be around 10%.

5.3 Demonstrator

The scan test method has been applied to a demonstrator IC developed by Philips Semiconductors. The IC contains four digital handshake circuit blocks and a number of analog circuits. The goal was to demonstrate the validity of the scan test method on an industrial design and in particular to:

- Demonstrate the automatic tool flow;
- Demonstrate the possibility to integrate the tests of multiple Tangram handshake blocks;
- Combine these in a design that also contains analog functions.

5.3.1 Design

The design contains three main digital blocks. Two of these control an analog front-end. The third is the system controller used to control the overall functions of the chip. One other smaller digital block supports the system controller. The three main blocks were based on circuits already presented in the previous sections, being the designs: tg60, tg303 and tg839. In Figure 5.3, the block diagram of the circuit is shown. Besides the above mentioned blocks, the circuit also contains a clock generator and a test control block(TCB). The clock generator can be bypassed to allow direct external access to the clock signals. This can be used to test the circuit in case the clock generator does not operate correctly. The TCB in the circuit can only be programmed directly after reset. Since this TCB is used to select between control-block test and data-path test, the circuit has to be reset to switch between these two types of tests. During test generation, all data-path tests will be grouped into one test and all control-block tests will be grouped in a second test.

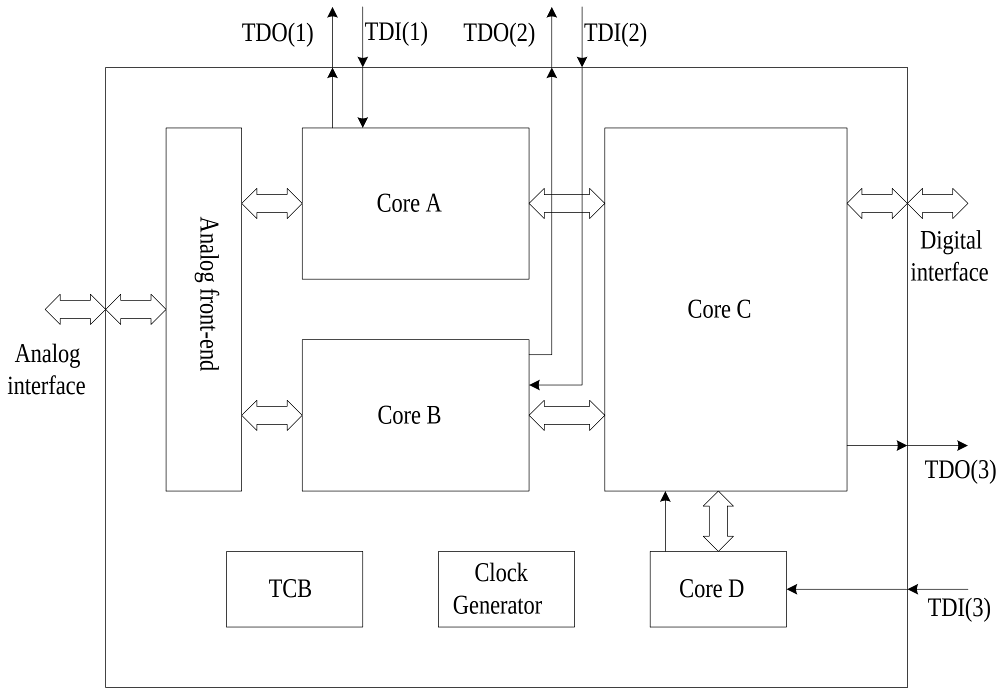


Figure 5.3: Block diagram of the design.

5.3.2 Test generation

Each of the four digital blocks has been designed separately with Tangram. Also the scan insertion and test-pattern generation was done separately for each block, resulting in a total of eight test-pattern sets, four for the data-paths and four for the control blocks. At circuit level, the scan chains of the cores C and D are serially connected. The scan chains of cores A and B are directly connected to the external interface. All scan inputs and outputs are multiplexed onto existing signals in the digital interface.

Test generation is first performed for the four sub-blocks. This uses the approach described in Appendix A, with one exception: during the test-protocol expansion phase no scan-continuity test is generated. This is postponed until the protocols are further expanded to the top-level of the chip. Furthermore, the test protocol expansion does not generate input files for the vector generation tool; instead it generates files that can be used to expand the protocols further.

Top level integration

In total there are eight pattern sets and corresponding initial protocols that need to be expanded. The first step is to expand the control block test and the data path test for each of the four blocks to block level. After this step is performed for all blocks, the protocols are further expanded to top-level. During the top-level expansion it is also necessary to program the TCB.

The TCB that is included can only be written directly after reset; therefore the chip is reset between the three control-block tests and the three data-path tests. The programming of the TCB can be included in a test protocol by adding an initialization section. The commands listed in such a section are executed once before the normal protocol expansion is started.

The tests for cores A, B, and the combination of C and D, can be executed in parallel, since they all use a different scan chain. This means that the test time is reduced to the test time of the core with the maximum test time, defined by the product of the scan chain length with the the number of test patterns.

5.3.3 Results

Initially the design was implemented using composite scan C-elements. These were in a second generation replaced by scan C-element that are build around a single primitive transparent flip-flop. In this way it was possible to achieve a large reduction in area overhead by only designing one new primitive cell. This netlist was used for the demonstrator IC. In a later stage a complete set of primitive scan C-elements was designed, which resulted in a further reduction of the area overhead and this also improved the speed of the circuit. A demonstrator IC using the primitive cells is, however, not yet available at the time of writing.

The cell area of the individual blocks was already given in the previous section. The cell area overhead of the blocks combined is given in Table 5.7, for the three alternative library implementations. The designs avoided the use of latches in the data path, which kept the contribution of the data path to the area overhead minimal. This also means that there is no difference between having a composite or a primitive level-sensitive flip-flop. The demonstrator IC that was made used the C-elements based on transparent a flip-flop, which resulted in a cell area overhead of 60 %.

Table 5.7: Area overhead.

C-element implementation style	Overhead
Composite	80 %
Based on transparent flip-flop	60 %
Primitive	37 %

The demonstrator IC also offers the possibility to evaluate the area after layout. This area has to account for all the additional wiring that is required to connect the scan elements. It was found that the total area after layout is only 5% higher than the cell area alone. This was obtained for the circuit using scan C-elements build with transparent flip-flops. This indicates that the overhead caused by the wires used for the test-control logic is not that significant. A non-scan handshake circuit has about the same overhead after layout. For a synchronous circuit the overhead is usually even larger because more area is used to create a balanced clock tree.

The scan test of the demonstrator IC has been successfully executed, however, detailed results about the tests are not yet available.

5.4 Summary

To use the test flow it was also necessary to design a library of scan C-elements. Initially a library containing composite cells was designed. These were build out of existing cells and could therefore be used immediately. Later, a more efficient primitive cell library was developed to significantly reduce the area used by the scan C-elements.

The method has been applied to a number of benchmark circuits. The results showed a high fault coverage, although for some circuits, the reset and mutex elements caused a lower coverage in the control block. The area overhead caused by the full scan test method depends highly on the available primitive cells in a library. Without special primitive cells for scan C-elements and level-sensitive flip-flops, the average overhead is 80%. Adding the primitive cells significantly reduces this. The current average area overhead if all primitive cells are available is around 35 %. In most cases the final chip area overhead will be even smaller, depending of what other types blocks (like memories) are present on the chip.

Chapter 6

Conclusion

In this thesis, a scan-test method has been introduced to test handshake circuits designed with the Tangram toolkit. A synchronous scan mode of operation is added to the handshake circuit. In this mode, the circuit is operated as a normal synchronous circuit. Because the test is based on synchronous scan testing, the test patterns can be generated with existing and unmodified test tools. In the following sections the results of the test method are summarized and reviewed, followed by some options for further optimizations to improve on the current results.

6.1 Conclusions

The objective of this work has been to develop a test method for handshake circuits. In the introduction, three main requirements for the test method were listed: a high fault coverage, an automated test flow and compatibility to existing test tools and practices. Besides these requirements, a number of other important properties were defined that needed to be optimized to result in a cost effective test method. The most important of these properties is the additional area that is required for the DfT logic. The trend in testing is that the cost associated with fault coverage and manual development effort is rising in comparison with the cost of silicon area. Therefore, it was concluded that the test method for handshake circuits should primarily target a high fault coverage and an automated flow, even if this would result in a higher area overhead.

The method best suited to meet the test requirements is a scan test method, controlled by a synchronous clock. The full-scan method that has been developed, supports a fully automated test flow, both for the DfT modifications and for the test generation. For the DfT modifications the new tool *TgScan* has been developed, that forms the interface between the handshake circuit world and the synchronous test world. TgScan inserts a scan chain in the handshake circuit and generates several output files that are subsequently used to control the application of existing test tools. By using these files, the test tools are able to generate test patterns that can be executed on the scannable handshake circuit and that result in a high structural fault coverage. ATPG is carried out with an existing (Philips proprietary) ATPG tool, ensuring both high-quality test patterns and a minimum number of patterns. The obtainable fault coverage depends on the presence of redundancy and the number of gates that have to be disabled by the test-mode signal, like the mutex elements. Further optimizations have to be added to the Tangram compiler to avoid these structures or reduce their number.

The current fault coverage for the control block as reported by the ATPG tool varies between 92 and 99%. After the optimizations are included, the fault coverage will increase further, reaching a level equal to the data-path fault coverage, which is over 99%. The overall fault coverage is comparable to that of a conventional scan tested synchronous circuit, and higher than that achieved by other the test proposals for handshake circuits discussed in Section 2.3.

An important practical advantage of a test method based on synchronous scan is that the method is compatible with many existing test standards and tools. Especially if the test-control optimizations, described in Section 4.4.1, are applied, the external interface of a scan testable handshake circuit is equal to that of a synchronous circuit. This holds for both the number and type of the interface signals as well as for their timing parameters. The result is that the scan test method is compatible with test-integration standards that are used to integrate the test for several blocks on an IC. Test integration is important because situations in which a chip consists of only a single handshake circuit are rare. In most cases the handshake circuit is combined with memories, analog circuits or other digital logic blocks, either synchronous or asynchronous. Most important and likely to be used for test integration are the macro test flow and the core based testing proposal (P1500).

The large area overhead required for the suggested DfT modifications is the main remaining issue. Initially when the composite cell library was used, the overhead was about 80%. The inclusion of primitive scan C-elements in the library reduced the area overhead from around 80% down to around 35%. This, however, might still not acceptable for all applications. Therefore, most of the future research will focus on a further reduction of the area overhead. Within the full scan method still some small improvements are possible, as will be discussed in Section 6.2. Two significant improvements that go beyond full-scan are L_1L_2* scan and partial scan, which are briefly discussed in Section 6.3. The optimizations are primarily targeted at reducing the area overhead, but in doing so they also reduce the power consumption and improve the circuit performance.

Besides the area overhead, cost is also associated with the number of additional test pins. The number of new test pins that are required depends on the design of the IC. If the handshake circuit is part of a larger IC, most or all of the test pins will already be present to support the test of the other parts of the IC. If this is not the case, a number of external pins will have to be added.

6.2 Improving full-scan

Synchronous scan-insertion tools offer a number of options for optimizations that are not yet applied to handshake circuits. The preferred way to add these features is to use (parts of) the synchronous tools as a pre (or post) processor and not to re-implement them in TgScan. The optimizations that are identified for incorporation in TgScan are discussed in the following sections.

6.2.1 Shift-register recognition

If shift registers are present in the circuit, no scan multiplexers are required for these registers, provided the proper scan-ordering is chosen. This avoids the additional area for the multiplexers and the test enable signal to control it. In handshake circuits, shift registers can only occur in the data path. In the control block, C-elements are used and since these have at least two data inputs, it is not directly possible to remove the scan multiplexer. However, it is possible to optimize the scan multiplexer if it is known which data input is also used as scan input. In those cases, the techniques described in Section 3.3.3 can be used to derive optimized designs or alternatively, new optimizations can be defined in the compiler. Shift-register optimization in the control block requires many new scan C-elements, all combinations of C-element type and scan-input pin have to be available. The primitive scan C-elements can also be optimized, but the result is largely dependent on the type of C-element and which data input is also used as the scan-in input.

6.2.2 User-defined scan-chain architecture

The current implementation of TgScan has no interface to customize the scan chain it generates. TgScan creates a single scan chain and adds new external pins to the design for the scan-in and scan-out signals. To make this more flexible, it should become possible to specify the number of scan chains and which pins should be used for scan-in and scan-out. These functions are all supported in the standard scan insertion tool that is used for synchronous circuits. The best solution would be to use (part of) this tool for these functions.

6.2.3 Layout-based scan routing

Layout-based scan routing is standard practice in synchronous scan test. It minimizes the wiring required to connect the scan elements by changing the order of the elements in the chain to reflect their physical location in the layout. For application in a handshake circuit, the tools need to know how to recognize scan C-elements and latches. Additionally, special conditions such as making separate groups for latches and flip-flops might be required.

6.3 Beyond full-scan

Next to the full scan optimizations, two optimizations exist that are not common in synchronous circuits and go beyond the full-scan test method. These optimizations are L_1L_2* scan and partial scan. Finally it is possible to combine these two optimizations. All three options are briefly discussed in the following sections.

6.3.1 L_1L_2* scan

The full-scan method as described in Chapter 4, requires a slave latch to be inserted for every latch and every C-element in the circuit. If primitive versions for the scan C-elements and level-sensitive flip-flops are available, then the overhead associated with these slave latches is limited. However, if primitive cells are not available, the overhead caused by these slave latches is substantial. In both cases it is desirable to remove the slave latches from the circuit. Besides the reduced cell area, also other properties of the circuit will improve if the slave latches are removed. Without slave latches there are also no global signals that have to be connected to these latches, reducing the wiring required for global control signals. Furthermore, without slave latches, the circuit is faster and consumes less power.

Following the L_1L_2* scan principles, introduced in [16] and briefly discussed in Section 1.2.3, some experiments have been carried out to determine the improvements that can be expected from this optimization. Some initial work on this subject has been reported in [5] and a partitioning algorithm has been developed. The potential improvement of the L_1L_2* scan method in terms of how many slave latches can be removed is highly dependent on the structure of the circuit. For a typical handshake circuit around 50 to 60% of the slave latches can be removed, resulting in an overall circuit area reduction of around 5 to 8%.

6.3.2 Partial scan

Another promising optimization is partial scan, introduced in Section 1.2.3. In the data path, a method like SmartScan [40] seems to be the best option. It allows a reasonable reduction of the number of scan elements, while it remains possible to use combinational ATPG. In the control block this approach might not be so successful, because the control block does not contain large pipeline structures. Instead, the control block contains many small loops, mostly spanning only a few gates. A conventional partial scan solution is also not desirable since that requires complex sequential ATPG which can lead to long test patterns.

A potential solution for the control block is to limit the length of a test pattern to only two sequential patterns. This creates a situation in which non-scan elements are completely surrounded by scan elements. Therefore all input pins of the non-scan elements are controllable and their output pins are observable. In this situation, all internal faults in the non-scan element can be tested with a two-pattern test, as shown in Section 2.2.3. Selecting the non-scan elements is equivalent to the "independent set" problem from graph theory. Some initial experiments have shown that such an approach can achieve a reduction of the number of scan elements in the control block of around 30%. On the overall circuit area this results in an area reduction of about 5%.

6.3.3 L_1L_2* scan combined with partial scan

It is also possible to combine the usage of L_1L_2* scan with partial scan. The minimal scan requirements are that every loop in the circuit needs to be broken by an L_1 latch and an L_2 latch. For loops that span three or more latches, the combination offers a higher potential area reduction than the optimizations individually. The effect in a real circuit will of course be smaller, since loops are often intersected with other loops and cannot be analyzed individually. The potential result on real circuits is not know yet, but is a promising area for research.

6.4 Implications for handshake circuits

In the previous sections, a number of optimizations have been described to reduce the cost of the scan test method. Figure 6.1 shows an estimation of the average area overhead that can be achieved by implementing these optimizations. As can be seen in this figure, an average area overhead of around 25% to 30% is expected. At the moment this seems to be the best obtainable by using a (full) scan method on the current type of handshake circuits.

If the area has to be reduced even further, this will start to influence the way handshake circuits are designed and tested. First it is important to avoid the programming constructs that have been identified in this thesis as difficult to test. The two most important design choices in this respect are: avoiding the use locally generated reset signals and only allow the used of latches if either primitive level-sensitive scan flip-flops or L_1L_2* scan are available. Both of these do not require a modification of the tools.

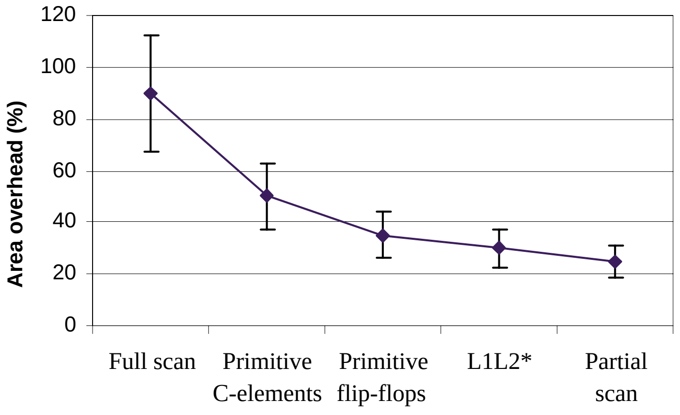


Figure 6.1: Reduction of the area overhead first by using primitive cells and then by applying the suggested optimizations

A further step is to specifically redesign some handshake components to make them less expensive to test. The current components were designed without scan test in mind. In that situation a C-element is not much more expensive to use than a logic gate. However, when using a scan method, this situation changes in that the C-elements now become much more expensive. The use of scan testing for handshake circuits changes the cost functions of the handshake components. To take this into account requires a redesign of the handshake components to reduce the number of C-elements. Handshake components that are designed to use fewer C-elements are better when scan test is used, even though they may be larger otherwise.

Bibliography

[1] M. Abadir and A. P. Ambler, editors. *Economics of electronic design, manufacture and test*. Kluwer Academic Publishers, 1994.

[2] M. Abramovici, M. A. Breuer, and A. D. Friedman. *Digital systems testing and testable design*. Freeman, 1990.

[3] V. D. Agrawal, editor. *Special Issue on Partial Scan Methods*, volume 7 of *Journal of Electronic Testing: Theory and Applications*. Kluwer Academic Publishers, Augustus/October 1995.

[4] A. Bailey and M. Josephs. Sequencer circuits for VLSI programming. In *Asynchronous Design Methodologies*, pages 82–90. IEEE Computer Society Press, May 1995.

[5] F. te Beest, A. Peeters, C. H. van Berkel, and H. G. Kerkhoff. Synchronous full-scan for asynchronous handshake circuits. In *IEEE European Test Workshop (ETW02)*, pages 381–387, May 2002.

[6] F. te Beest, A. Peeters, M. Verra, C. H. van Berkel, and H. G. Kerkhoff. Automatic scan insertion and test generation for asynchronous circuits. In *Proc. International Test Conference*, October 2002.

[7] C. H. van Berkel. Beware the isochronic fork. *Integration, the VLSI journal*, 13(2):103–128, June 1992.

[8] C. H. van Berkel. *Handshake Circuits: an Asynchronous Architecture for VLSI Programming*, volume 5 of *International Series on Parallel Computation*. Cambridge University Press, 1993.

[9] C. H. van Berkel, R. Burgess, J. Kessels, A. Peeters, M. Roncken, and F. Schalij. A fully-asynchronous low-power error corrector for the DCC player. In *International Solid State Circuits Conference*, pages 88–89, February 1994.

[10] C. H. van Berkel, M. B. Josephs, and S. M. Nowick. Scanning the technology: Applications of asynchronous circuits. *Proceedings of the IEEE*, 87(2):223–233, February 1999.

[11] C. H. van Berkel, J. Kessels, M. Roncken, R. Saeijs, and F. Schalij. The VLSI-programming language Tangram and its translation into handshake circuits. In *Proc. European Conference on Design Automation (EDAC)*, pages 384–389, 1991.

[12] C. H. van Berkel, A. Peeters, and F. te Beest. Adding synchronous and LSSD modes to asynchronous circuits. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 161–170, April 2002.

[13] M. L. Bushnell and V. D. Agrawal. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Kluwer Academic Publishers, 2000.

[14] S. Chakravarty and P. J. Thadikaran. *Introduction to IDDQ testing*. Kluwer Academic Publishers, 1997.

[15] K. T. Cheng and V. D. Agrawal. A partial scan method for sequential circuit with feedback. *IEEE Transactions on Computers*, 39(4):544–548, April 1990.

[16] S. DasGupta, P. Goel, R. G. Walther, and T. W. Williams. A variation of LSSD and its implications on design and test pattern generation in VLSI. In *IEEE Test Conference*, 1982.

[17] I. David, R. Ginosar, and M. Yoeli. Self-timed is self-checking. *Journal of Electronic Testing: Theory and Applications*, 6(2):219–228, April 1995.

[18] B. Davis. *The economics of automated testing*. McGraw-Hill, 1994.

[19] E. B. Eichelberger and T. W. Williams. A logic design structure for LSI testability. In *IEEE Transactions on Computers*, pages 462–468, 1978.

[20] R. D. Eldred. Test routines based on symbolic logical statements. *Journal of the ACM*, 6(1):33–36, January 1959.

[21] A. Ferre and J. Figueras. On estimating bounds of the quiesent current for IDDQ testing. In *Proc. of the 14th VLSI test symp.*, pages 106–111, May 1996.

[22] F. Fujiwara and T. Shimono. On the acceleration of test generation algorithms. In *IEEE Trans. Comput.*, volume C-32, pages 1137–1144, 1983.

[23] S. Funatsu, N. Wakatsuki, and T. Arima. Test generation systems in Japan. In *Proceedings 12th Design Automation Symposium*, pages 114–122, June 1975.

[24] H. van Gageldonk. *An Asynchronous Low-Power 80C51 Microcontroller*. PhD thesis, Dept. of Math. and C.S., Eindhoven Univ. of Technology, September 1998.

[25] J. M. Galey, R. E. Norby, and J. P. Roth. Techniques for the diagnosis of switching circuit failures. In *R. S. Ledley, editor, Proc. of the second annual symp. on switching circuit theory and logical design*, pages 152–160, October 1961.

[26] P. Goel. An implicit enumeration algorithm to generate tests for combinational logic circuits. In *IEEE Trans. Comput.*, volume C-30, pages 215–222, 1981.

[27] R. K. Gulati and C. F. Hawkins, editors. *IDDQ testing of VLSI circuits*. Kluwer Academic Publishers, 1993.

[28] D. Harris. *Skew-tolerant circuit design*. Morgan Kaufmann Publishers, 2001.

[29] P. J. Hazewindus. *Testing Delay-Insensitive Circuits*. PhD thesis, California Institute of Technology, 1992.

[30] M. J. Howes and D. V. Morgan, editors. *Reliability and degradation - semiconductor devices and circuits*. Wiley-Interscience, 1981.

[31] H. Hulgaard, S. M. Burns, and G. Borriello. Testing asynchronous circuits: A survey. *Integration, the VLSI journal*, 19(3):111–131, November 1995.

[32] J. Kessels, T. Kramer, G. den Besten, A. Peeters, and V. Timm. Applying asynchronous circuits in contactless smart cards. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 36–44. IEEE Computer Society Press, April 2000.

[33] J. Kessels, T. Kramer, A. Peeters, and V. Timm. DESCALE: a design experiment for a smart card application consuming low energy. In Rene van Leuken, Reinder Nouta, and Alexander de Graaf, editors, *European Low Power Initiative for Electronic System Design*, pages 247–262. Delft Institute of Microelectronics and Submicron Technology, July 2000.

[34] A. Khoche and E. Brunvand. Testing micropipelines. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 239–246, November 1994.

[35] L. Kleeman and A. Cantoni. On the unavoidability of metastable behavior in digital systems. *IEEE Transactions on Computers*, C-36(1):109–112, January 1987.

[36] L. Lavagno, M. Kishinevsky, and A. Lioy. Testing redundant asynchronous circuits by variable phase splitting. In *Proc. European Design Automation Conference (EURO-DAC)*, pages 328–333. IEEE Computer Society Press, September 1994.

[37] Y. K. Malaiya and R. Rajsuman, editors. *Bridging faults and IDDQ testing*. IEEE Computer Society Press, 1992.

[38] E. J. Marinissen, R. Kapur, and Y. Zorian. On using IEEE P1500 SECT for test plug-n-play. In *Proc. International Test Conference*, pages 770–777, October 2000.

[39] E. J. Marinissen and M. Lousberg. The role of test protocols in testing embedded-core-based system ICs. In *Proceedings IEEE European Test Workshop*, pages 70–75, Konstanz, Germany, May 1999.

[40] E. J. Marinissen and M. Muijen. Smartscan: Parial scan with full scan benefits. In *4th IEEE International Test Synthesis Workshop*, May 1997.

[41] A. J. Martin. The limitations to delay-insensitivity in asynchronous circuits. In William J. Dally, editor, *Advanced Research in VLSI*, pages 263–278. MIT Press, 1990.

[42] A. J. Martin. Programming in VLSI: From communicating processes to delay-insensitive circuits. In C. A. R. Hoare, editor, *Developments in Concurrency and Communication*, UT Year of Programming Series, pages 1–64. Addison-Wesley, 1990.

[43] A. J. Martin and P. J. Hazewindus. Testing delay-insensitive circuits. In Carlo H. Séquin, editor, *Advanced Research in VLSI*, pages 118–132. MIT Press, 1991.

[44] P. Maxwell, I. Hartanto, and L. Bentz. Comparing functional and structural test. In *Proc. International Test Conference*, pages 400–407, 2000.

[45] A. Peeters. *Single-Rail Handshake Circuits*. PhD thesis, Eindhoven University of Technology, June 1996.

[46] O. A. Petlin. *Design for Testability of Asychronous VLSI Circuits*. PhD thesis, Department of Computer Science, University of Manchester, 1996.

[47] O. A. Petlin and S. B. Furber. Scan testing of micropipelines. In *Proc. IEEE VLSI Test Symposium*, pages 296–301, May 1995.

[48] L. A. Plana and S. M. Nowick. Architectural optimization for low-power non-pipelined asynchronous systems. *IEEE Transactions on VLSI Systems*, 6(1):56–65, March 1998.

[49] M. Roncken. Partial scan test for asynchronous circuits illustrated on a DCC error corrector. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 247–256, November 1994.

[50] M. Roncken. Defect-oriented testability for asynchronous IC's. *Proceedings of the IEEE*, 87(2):363–375, February 1999.

[51] M. Roncken, E. Aarts, and W. Verhaegh. Optimal scan for pipelined testing: An asynchronous foundation. In *Proc. International Test Conference*, pages 215–224, October 1996.

[52] M. Roncken and E. Bruls. Test quality of asynchronous circuits: A defect-oriented evaluation. In *Proc. International Test Conference*, pages 205–214, October 1996.

[53] M. Roncken and R. Saeijs. Linear test times for delay-insensitive circuits: a compilation strategy. In S. Furber and M. Edwards, editors, *Asynchronous Design Methodologies*, volume A-28 of *IFIP Transactions*, pages 13–27. Elsevier Science Publishers, 1993.

[54] J. P. Roth, W. G. Bouricious, and P. R. Schneider. Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits. In *IEEE Trans. Electronic Comput.*, volume EC-16, pages 567–579, 1967.

[55] V. Schöber and T. Kiel. An asynchronous scan path concept for micropipelines using the bundled data convention. In *Proc. International Test Conference*, October 1996.

[56] J. Segura and A. Rubio. A detailed analysis of CMOS SRAMs with gate oxide short defects. *IEEE Journal of solid-state circuits*, 32(10):1543–1550, October 1997.

[57] C. L. Seitz. System timing. In Carver A. Mead and Lynn A. Conway, editors, *Introduction to VLSI Systems*, chapter 7. Addison-Wesley, 1980.

[58] International Sematech. International technology roadmap for semiconductors (ITRS), 2001.

[59] M.-D. Shieh, C.-L. Wey, and P. D. Fisher. A scan design for asynchronous sequential logic circuits using SR-latches. In *Proc. of the Midwest Symposium on Circuits and Systems*, pages 1300–1303, 1993.

[60] G. Singer. Current trends and future directions in test and DfT, keynote address. In *Proc. of the 15th VLSI test symp.*, May 1997.

[61] J. M. Soden, R. R. Fritzemeier, and C. F. Hawkins. Zero-defect or zero stuck-at faults - CMOS IC process improvements with iddq. In *Proc. International Test Conference*, pages 240–245, September 1990.

[62] J. M. Soden, C. F. Hawkins, R. K. Gulati, and W. Mao. IDDQ testing: A review. *Journal of Electronic Testing: Theory and Applications*, 3(4):5–17, December 1992.

[63] J. Sparsø and S. Furber, editors. *Principles of Asynchronous Circuit Design: A Systems Perspective*. Kluwer Academic Publishers, 2001.

[64] I. E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, June 1989.

[65] P. Tafertshofer, A. Ganz, and M. Henftling. A SAT-based implication engine for efficient ATPG, equivalence checking and optimization of netlists. In *Proc. of the International Conference on Computer-aided design*, pages 648–655, November 1997.

[66] E. H. Volkering, A. Khoche, L. A. Kamas, J. Rivior, and H. G. Kerkhoff. Tackling test trade-offs from design, manufacturing to market using economic modeling. In *Proc. International Test Conference*, pages 1098–1107, October 2001.

[67] C.-L. Wey, M.-D. Shieh, and P. D. Fisher. ASCLScan: a scan design for asynchronous sequential logic circuits. In *Proc. International Conf. Computer Design (ICCD)*. IEEE Computer Society Press, 1993.

[68] R. van de Wiel. High-level test evaluation of asynchronous circuits. In *Asynchronous Design Methodologies*, pages 63–71. IEEE Computer Society Press, May 1995.

[69] M. J. Y. Williams and J. B. Angell. Enhancing testability of large scale intergrated circuits via test points and additional logic. In *IEEE Transactions on Computers*, volume C-22, 1973.

Appendix A

Using the CAT tools

Besides the newly developed tool TgScan, all remaining functions used in the test flow use tools available in the standard in-house Computer Aided Test (CAT) toolkit at used Philips. In most cases, special settings have been used for these tools. To correctly introduce these settings, control files for the CAT tools are automatically generated by TgScan. In addition, TgScan generates a test-data file. This file contains an abstract description of scan chains and test modes in the netlist.

The test tools do not operate on the real scan netlist, rather a significant part of the processing is done using the remodelled files. The remaining part operates on an abstract view of the circuit, referred to as the CAT view. The information available in the CAT view is shown in for an example in Figure A.1. This example will be used in the following sections to explain the operation of the CAT tools. Figure A.1 contains a top-level block and two lower-level blocks for the control block and the data path. In the scan chain between the data path and the control block an additional scan element, labelled *transparent FF*, is present. This is a transparent scan flip-flop that connects the scan chains and in addition functions as an anti-skew latch. The information that is available with regard to the blocks is the number and type of input and output pins, and an abstract description of the scan chain inside the blocks. This description includes the length of the chain, the scan-in and scan-out pin, the interface signals between the control block and data path, and possible inversions in the scan chain.

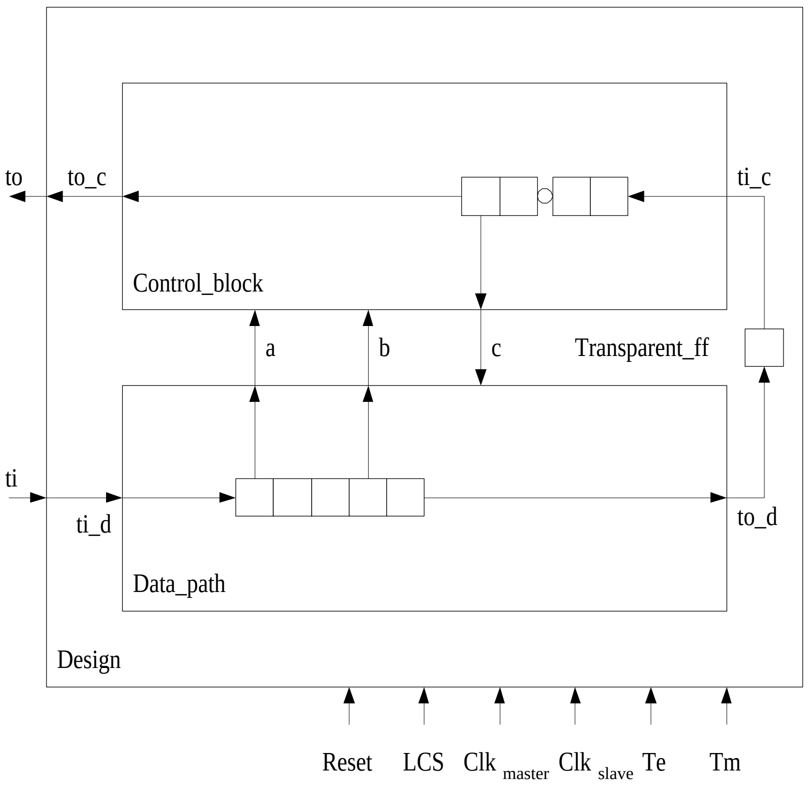


Figure A.1: CAT view of the design.

A.1 Test-pattern generation

Test patterns are generated separately for the control block and the data path. For both blocks, a remodelled netlist and a control file are available. In the remodelled netlist some input pins are not connected. In the real circuit, these are used for either clock signals or control signals. For example, the slave clock clk_{slave} is not connected because in the remodel file no slave latches are present. The reason for this is that latches have all been remodelled by flip-flops. All scan elements in the remodelled netlist are connected to clk_{master}, as this clock is defined as the system clock for the CAT tools. If an on-chip clock generator is used, the other clock pins do not have to be described. If a separate input pin is used for the slave clock, this pin has to be defined as a clock signal in the control file.

The same holds for the latch-control select signal LCS that is used to control the multiplexer in the latch controller. In the remodel file, this multiplexer is not present and the request and acknowledge signals of the latch controller are directly connected to each other. During the execution of the test, these input pins need to be correctly controlled. This is accomplished by specifying the behaviour of these pins in the control file.

The number of input signals that have to be defined in the control file depends on the circuit. Not all circuits require all possible control inputs and some inputs can be generated on-chip from other signals as was shown in Section 4.4.1. Besides the clock signals, at most four test-control signals have to be defined: LCS, Te, Tm and $Reset$, all shown in Figure A.1. The behaviour of these signals is specified during shift-mode and normal-mode. The ATPG tool will derive the function of the remaining signals like scan-in and scan-out from the remodelled netlist. Besides the input signal definition, the control file is also used to specify the technology libraries that should be used and the parameters for the ATPG algorithm. These can for example be used to tune the algorithm, exclude certain input combinations and the limit the computational time.

The ATPG tool will generate a test-pattern file. This file contains the generated test patterns and the initial test protocol that describes how to apply the patterns. Listing A.1 shows a number of possible patterns generated for the control block. There are six stimulus values, corresponding to four scan elements and two primary inputs. In the example, no primary outputs are present and therefore the only response values are the four corresponding to the scan elements.

Listing A.1 Pattern set for the control block test of the circuit in Figure A.1

```
Pattern   Stimulus   Response
          012345     0123

1:        001101     0110
2:        101000     1101
3:        ......     ....
```

The test protocol specifies which value in the table corresponds to which signal in the circuit. The initial protocol of the example is given in Listing A.2. The signal names in the protocol correspond to the signal names in Figure A.1. The protocol contains three main sections: condition, stimulus and response. Every section contains signal definitions and the times for which those definitions hold which range from $< -x$ to $y >$. The normal cycle is defined at time zero. The scan-in phase uses negative numbers, starting at time $-x$. Time y specifies the end of the scan-out phase; both x and y are defined by the scan-chain architecture. The condition section specifies the behavior of the test-control signals LCS and Te during the test. Te is also specified during the scan-out phase, LCS is undefined during this phase. In the example, Tm and $Reset$ are not used. The stimulus and response sections relate the circuit inputs and outputs with the values in the pattern file. For example, the statement `a = S[4]`, relates input a of the circuit to the fourth column in the test-pattern file. The output c is not specified because this signal is not used for the scan test of the control block, as explained in Section 4.2.2.

Listing A.2 Initial protocol of control block test of the circuit in Figure A.1

```
Cell Control_block {
 Protocol at Control_block {
  Patternfile = "Control_block.pat";
  Condition {
   LCS<-4:-1> = [0];
   LCS<0> = [1];
   Te<-4:-1> = [1];
   Te<0> = [0];
   Te<1:4> = [1];
  }
  Stimulus {
   ti_c<-4:-1> = S[0:3]
   a = S[4];
   b = S[5];
  }
  Response {
   to_c<1:4> = R[0:3]
}}}
```

A.2 Protocol expansion

During protocol expansion, the two initial protocols that were generated by the ATPG tool for the control block and data path are expanded to top-level. This involves tracing all pins of a block to either the corresponding pin at top-level or to a pin that is controlled by the scan chain.

The protocol expansion tool starts by loading all the data required to expand the protocols. First loaded is the top-level netlist of the design. This top-level netlist only contains the top-level block and the interface definitions for the control block and data path. The structure of top-level netlist for the example is shown in Listing A.3, for clarity the signal names are not shown. Besides the modules for the control block and the data-path, a new module for the transparent scan flip-flop is defined.

Listing A.3 The top-level netlist in Verilog format

```
module Data_path( ... );

endmodule
```

```
module Control_block( ... );

endmodule

module Transparent_ff( ... );

  TransFF inst ( ... );
endmodule

module Design( ... );

  Data_path data_inst( ... );
  Control_block ctrl_inst( ... );
  Transparent_ff trans_inst( ... );
endmodule
```

The netlist does not contain information about the internal scan chains. This information is loaded from a test-data file. The test data for the example is given in Listing A.4. This file specifies the scan chain for every sub-block. The properties that are defined are the scan input, the scan output, the scan-enable signal and the length of the scan chain. The “sffoutput” property specifies intermediate outputs and the number of the scan element the output signal originates from. This defines the connections between the control block and the data path and is used to determine which scan element drives which pin. In the control block, inverting scan elements can be used, as explained in Section 3.3.3. During ATPG, this inversion is correctly modelled in the remodel file. For the protocol expansion this information is also required in order to correctly expand the protocol for the data-path test. If inversions are not accounted for, the intermediate signals from control block to the data path might be inverted; also the scan-out pin might be inverted. Information about inversions in the control block scan chain is included in the test-data file and is specified by the property “InversionBetween”.

Listing A.4 Test data file

```
cell data_inst {
  chain d_chain {
    scanenable = Te;
    scaninput = ti_d;
    scanoutput = to_d;
    length = 5;
    sffinput {}
    sffoutput {a, 1; b, 4;}
  }
}
cell ctrl_inst {
  chain c_chain {
    scanenable = Te;
    scaninput = ti_c;
    scanoutput = to_c;
    length = 4;
    sffinput {}
    sffoutput {c, 4;}
    InversionBetween {2, 3;}
  }
}
cell trans_inst {
  chain t_chain {
    scanenable = Te;
    scaninput = to_d;
    scanoutput = ti_c;
    length = 1;
    sffinput {to_d, 1;}
    sffoutput {ti_c, 1;}
  }
}
```

The final information loaded by the protocol expansion tool are the two initial protocols. This completes the data required to expand the protocols. In the next step, three top-level test protocols are generated: one is a scan continuity test, and the other two are the expanded versions of the control-block test and the data-path test.

The expanded protocol of the control-block test of the example circuit is shown in Listing A.5. Compared to the initial protocol in Listing A.2, the protocol uses more cycles. The reason for this is that the values for the scan elements first have to be shifted through the scan elements in the data path before they reach their intended location. For this reason, the protocol now starts six cycles earlier. The second main difference is that the stimulus data for the primary inputs a and b of the control block is now applied via a scan element in the data-path. The line `a = S[4]` has been replaced by `ti_c<-4> = S[4]` meaning that the data for input a has to be present four cycles before the normal-mode cycle at the scan input.

Listing A.5 Test protocol of control block at top-level

```
Cell Control_block {
 Protocol at Design {
  Patternfile = "Control_block.pat";
  Condition {
   Tm<-10:-1> = [0];
```

```
    Tm<0> = [1];
    Te<-10:-1> = [1];
    Te<0> = [0];
    Te<1:4> = [1];
   }
   Stimulus {
    ti_c<-10:-7> = S[0:3]
    ti_c<-4> = S[4];
    ti_c<-1> = S[5];
   }
   Response {
    to_c = R[0:3]
 }}}
```

A.3 Vector generation

The final step in the test flow is the generation of the test vectors. This is accomplished by a vector-generation tool that is part of the existing CAT toolkit. The tool combines the test protocols and the test patterns into the final test vectors that are used to test the circuit. The output of the tool is either a set of test vectors for a specific tester or a simulation file that can be used to simulate the test vectors together with the scan-testable circuit. Default, a Verilog simulation file is created that can be used for logical simulation of the test vectors. An option is to include information in the simulation file to allow for fault simulation. Fault simulation is a tool that can be used to determine the fault coverage of the test with regard to the real scan-testable circuit, which includes all asynchronous structures. Unfortunately, this requires excessive run times and is therefore only feasible for small designs. If the simulation of the patterns is successful, vectors can be generated for a selected tester.

Summary

In this thesis a full-scan test method is introduced that can be used to test handshake circuits. Handshake circuits are a class of asynchronous circuits that are designed according to set of rules that guarantee correct operating circuits. The circuits used in this work are designed using the Tangram toolkit that was developed at Philips Research.

Handshake circuits are more difficult to test than conventional synchronous circuits, because of a number of reasons. Most importantly is the fact that the circuits behave autonomously. Internal operations can occur independently of external input signals, which limits the outside control that can be exerted on the circuit. Other testing problems associated with handshake circuits are the higher number of sequential elements in the circuits, as compared to synchronous circuits, and to a lesser extend problems with initialization and non-deterministic behavior.

For a test method to be cost effective, a number of properties are important. Traditionally minimization of the area overhead has been crucial, but the possibility of offering an automated flow and guaranteeing high fault coverage are becoming increasingly important. The full-scan test method was chosen because this method offers an automated flow and high fault coverage, even though it requires a relatively high area overhead.

With a full-scan test method every sequential element in the circuit is modified into a scan element. Scan elements have a multiplexer on the data input that makes a second data input available that is used to connect all scan elements serially in a shift register. All scan elements are controlled by a global clock signal that is added to the circuit to support the scan test. Clocking is based on a pair of two-phase non-overlapping clock signals, which apart from safe timing also support a transparent mode in which the circuit behaves functionally as an unmodified asynchronous circuit.

Many of the sequential elements used in handshake circuits consist of C-elements. A C-element is a generic form of a set-reset latch. Several different types exist; each with alternative set and reset functions. In total about a dozen variations are commonly used. For use in a scan test method, the C-elements have to be modified into scan testable C-elements. Two modifications are required: the addition of a scan data input and the addition of clock input to control the element. These modifications increase the size of a C-element, which combined with the fact that a typical handshake circuit contains a large number of C-elements will result in a high overall area increase of the circuit. To counteract this, smaller scannable C-elements are designed at transistor level and circuits are optimized to reduce the number of C-elements.

Handshake circuits can also contain conventional latches and flip-flops. These are used in the data path and are clocked with local clock signals generated by the control block. For the scan test method, a global clock has to be multiplexed onto these local clock signals. This is implemented by inserting a multiplexer in the local clock signal, however, to be able to test the clock multiplexer itself, separate tests are used for the data path and the control block. Each of these two tests will cover a part of the faults in the multiplexer, and together they cover all faults in the multiplexer.

One of the main reasons to choose the full scan test method, is the availability of a large number of existing tools that, without modification, can be used to generate the test patterns. In addition these test tools support hierarchical test generation. This allows the tests for the control block and for the data path to be generated separately first and later combined into a top-level test. During test pattern generation, C-elements and latches are remodelled by flip-flops. This is required to make the test pattern generation tool to recognize them as valid scan elements.

The method has been applied to several benchmark circuits, which were processed fully automatically. The results show that a high fault coverage of over 99% can be achieved, equal to that of a synchronous circuit. The required area overhead is however much larger than that of a scannable synchronous circuit. Using gate-level scan C-element implementations results in an average overhead of around 80%. By using an optimized transistor level library, this reduces to around 35%.

Samenvatting

In dit proefschrift wordt een full-scan test methode geïntroduceerd die gebruikt kan worden voor het testen van handshake schakelingen. Handshake schakelingen zijn een klasse van asynchrone schakelingen die worden ontworpen op basis van een aantal regels die garanderen dat de schakeling goed functioneert. De schakelingen die in dit proefschrift worden gebruikt zijn ontworpen met behulp van de Tangram gereedschappen die ontwikkeld zijn bij Philips Research.

Door een aantal redenen, zijn handshake schakelingen moeilijker te testen dan gewone synchrone schakelingen. Het belangrijkste is het feit dat de schakelingen zich autonoom gedragen. Interne handelingen kunnen onafhankelijk van externe ingang signalen gebeuren, hetgeen de controle beperkt die vanaf de buitenkant op de schakeling kan worden uitgeoefend. Andere test problemen van handshake schakelingen zijn het grotere aantal geheugen elementen, vergeleken met synchrone schakelingen, en in mindere mate problemen met initialisatie en niet deterministisch gedrag.

Om een test methode kosten effectief te laten zijn, zijn een aantal eigenschappen belangrijk. Traditioneel is de minimalisering van de extra oppervlakte belangrijk, maar in toenemende mate worden het aanbieden van een geautomatiseerde methode en de fouten dekkingsgraad belangrijk. De full-scan test methode is gekozen omdat deze geautomatiseerd is en een hoge dekkingsgraad bied, ook al gaat dit gepaard met een relatief hoge extra oppervlakte.

Met een full-scan test methode wordt elk geheugen element in de schakeling gemodificeerd in een scan element. Scan elementen hebben een multiplexer voor de data ingang, die gebruikt wordt voor het maken van een tweede data ingang. Deze wordt vervolgens gebruikt om alle scan elementen serieel te verbinden in een scan keten. Alle scan elementen worden bestuurd door een globaal klok signaal die hiervoor wordt toegevoegd aan de schakeling. De elementen worden geklokt met een paar tweefase niet overlappende klokken, die naast het zorgen voor een veilig gedrag in de schakeling ook een transparante mode ondersteunen waarin de schakeling zich als een niet gemodificeerde asynchrone schakeling gedraagt.

Veel van de geheugen elementen die gebruikt worden in handshake schakelingen zijn de C-elementen. Een C-element is een generieke vorm van een set-reset latch. Er bestaan verschillende types; elk met een alternatieve set en reset functie. In totaal worden er ongeveer 12 variaties gebruikt. Om ze te kunnen gebruiken in een scan test methode, moeten de C-elementen worden gemodificeerd tot scan testbare C-elementen. Twee modificaties zijn nodig: het toevoegen van de scan data ingang en het toevoegen van de klok ingangen om het element te kunnen besturen. De twee modificaties zorgen voor een toename van de grote van het C-element, gecombineerd met het feit dat handshake schakeling typisch veel van deze elementen bevat, zorgt dit voor een grote toename van het oppervlak van de schakeling. Om dit tegen te gaan zijn er ook kleinere scan C-elementen ontworpen op transistor niveau en worden de handshake schakelingen geoptimaliseerd om het aantal C-elementen te verminderen.

Handshake schakelingen kunnen ook conventionele latches en flip-flops bevatten. Deze worden gebruikt in het datapad en worden geklokt met klok signalen die lokaal in een controle blok worden gegenereerd. Om ze te gebruiken in de scan test methode, moet er een globaal klok signaal worden gemultiplext op de lokale klok signalen. Dit is geïmplementeerd door in elk lokale klok signaal een multiplexer toe te voegen, echter om deze multiplexer zelf te testen, zijn aparte tests nodig voor het datapad en het controle blok. Elk van deze testen dekt maar een gedeelte af van de fouten in de multiplexer, maar samen dekken ze alle fouten in de multiplexer af.

Een van de belangrijkste redenen om een full-scan test methode te gebruiken is de aanwezigheid van een groot aantal programma's die zonder wijziging gebruikt kunnen worden om test patronen te genereren. Daarnaast ondersteunen deze programma's hiërarchische test generatie, hetgeen gebruikt wordt om eerst aparte testen te genereren voor het datapad en het controle blok en deze later samen te voegen in één test. Gedurende test patroon generatie worden C-elementen en latches gehermodelleerd door flip-flops. Dit is nodig om het test patroon generatie programma deze elementen te laten herkennen als geldige scan elementen.

De methode is toegepast op een aantal referentie schakelingen, die volledig automatisch zijn aangepast. De resultaten laten zien dat een fouten dekking van meer dan 99% kan worden gehaald, hetgeen vergelijkbaar is met de dekking van een synchrone schakeling. De benodigde extra oppervlakte is echter veel groter dan wat nodig is in een synchrone schakeling. Wanneer gate-level scanbare C-elementen worden gebruikt, is er ongeveer 80% extra oppervlakte nodig. Door transistor niveau geoptimaliseerde C-elementen te gebruiken kan dit worden gereduceerd tot ongeveer 35%.

Acknowledgments

It all started after I finished my masters thesis back in 1998. My supervisor Ronald Tangelder asked if I would be interested in doing a Ph.D. thesis. The answer was: no, unless you can offer me (1) an exciting subject, (2) industrial cooperation and (3) more than the at that time unconvincing salary. A little to my surprise he was able to offer all three of them and now over four years later it has resulted in this thesis.

It has been a time with numerous pleasant moments and new experiences made possible by all my friends and colleges. I therefore would like to thank everybody who has supported me over the last four years. A special word of thanks is in place for a number of people.

- My promotor prof. Thijs Krol and my second promotor prof. Kees van Berkel for their support and critical review of this work.
- Hans Kerkhoff, my assistant promotor and daily supervisor in Twente.
- Ad Peeters, my daily supervisor at Philips for his detailed knowledge about everything Tangram and everything else.
- I would like to thank Ronald Tangelder for getting me to start this work and for being my supervisor during the first two years.
- Marc Verra for his support with implementing TgScan, without him it would have taken a lot longer to develop a working prototype.
- Marc Mutsaers and Marjolein Koopman for providing me a place to stay whenever I was in Enschede.
- Vladimir Zivkovic and Octavian Petre for being my roommates and all the other (former) Ph.D. candidates in Twente: Milan Stancic, Liquan Fang, Arun

Anthony Joseph, Herman Vermaak and Nur Engin. Milan and Octavian are specially acknowledged for the great time we spend scootering around Corfu.

- All the support form the secretariat, system administration, financial administration and technical support. Without you it would now have been possible to complete this work.

- Pieter van der Horn and Hajo Broersma for working on the L_1L_2* algorithm, unfortunately time was to short to fully implement it.

Finally, I would like to thank my family who have supported me and gave me always the opportunity to get away from my daily work and to do some refreshing outdoor work on my parents farm.

Eindhoven
April, 2003.

Frank te Beest

About the author

Frank te Beest was born on the 19th of June 1973 in the town of Dinxperlo in the Netherlands. He started his study Electrical Engineering at the University of Twente in Enschede in 1993. In 1998 he graduated on an M.Sc. thesis on a behavioral model of an AD converter.

Subsequently in 1999 he started to work towards his Ph.D. on the testability of asynchronous circuits. This work was carried out as a cooperation between the University of Twente and the Philips Research Laboratories, supported by the Dutch Technology Foundation STW. The work led to a number of publications at journals, conferences, workshops and finally to this thesis. His research interests are the design and test of asynchronous circuits, including scan test, ATPG and core based test.

Frank te Beest is currently working at the Philips Research Laboratories in Eindhoven, the Netherlands.

Current address

Philips Research Laboratories
Prof. Holstlaan 4
5656 AA Eindhoven
The Netherlands

E-mail: frank.te.beest@philips.com

Index