Design and DfT of a High-Speed Area-Efficient Embedded Asynchronous FIFO

Paul Wielage[1*] Erik Jan Marinissen[1] Michel Altheimer[2**] Clemens Wouters[3]

[1] NXP Semiconductors
Research - Digital Design & Test
High Tech Campus 48, M/S-02
5656AE Eindhoven
The Netherlands
paul.wielage@nxp.com
erik.jan.marinissen@nxp.com

[2] NXP Semiconductors
Digital Library Technology
505, Route des Lucioles
Sophia Antipolis, 06560 Valbonne
France

[3] NXP Semiconductors
Digital Library Technology
High Tech Campus 46, M/S-11
5656AE Eindhoven
The Netherlands
clemens.wouters@nxp.com

Abstract

Embedded First-In First-Out (FIFO) memories are increasingly used in many IC designs. We have created a new full-custom embedded ripple-through FIFO module with asynchronous read and write clocks. The implementation is based on a micropipeline architecture and is at least a factor two smaller than SRAM-based and standard-cell-based counterparts. This paper gives an overview of the most important design features of the new FIFO module and describes its test and design-for-test approach.

1 Introduction

Embedded First-In First-Out (FIFO) memories are increasingly used in IC design for intermediate storage, data rate conversion, and clock domain crossing. New design paradigms like Network-on-Chip (NOC) and Globally-Asynchronous Locally-Synchronous (GALS) use embedded FIFOs extensively [1, 2]. Despite the fact that a single individual FIFO is not very large, the number of FIFOs in a circuit can be huge, and hence their overall silicon area contribution can be significant. Consequently, it is important that they are implemented in an area-efficient way.

Conventional FIFO designs are based on random access memory for storing the data, e.g. an embedded SRAM or one made up entirely from standard-cell logic. SRAM-based FIFOs offer an area-efficient bit-cell, but small instances suffer from a relatively large SRAM periphery (such as address decoders and sense-amplifiers) in addition to the FIFO control overhead. Standard-cell-based FIFOs omit any kind of analog circuitry, but require a relatively large area to store a single data bit.

This paper describes the design of a new full-custom FIFO module which is now part of the NXP design library. It is a micropipeline, consisting of a series of asynchronously communicating stages, each implemented as a register of latches and a control cell. The FIFO provides area-efficient data storage, a small control path, low energy consumption, and high-speed operation. The latter is achieved by implementing a modified version of the conventional four-phase handshake protocol for communication between the various stages of the micropipeline.

We also describe the Design-for-Test (DfT) approach for this FIFO. The FIFO has a dedicated test, but unlike what is common for most embedded non-logic modules such as memories, we do not equip our FIFO with a full test wrapper or Built-In Self Test (BIST). Instead, dedicated DfT hardware enables the application of test stimuli to multiple (or all) FIFOs in parallel, while for further test access, the FIFOs are integrated into the on-chip scan chains.

The remainder of the paper is organized as follows. The FIFO design is described in Section 2 and its test and DfT approach is detailed in Section 3. The paper is concluded in Section 4.

2 FIFO Design

The architecture of the FIFO design is depicted in Figure 1(a). The FIFO consists of a write interface, a read interface, and a core. The core is an asynchronous micropipeline [3]. The pipeline consists of a chain of stages, where each stage can hold at most one word of data. By means of local handshake signaling between the stages data propagates through the pipeline in a self-timed fashion. The employed four-phase handshake protocol takes care of a fast and coherent movement of data from one stage to the next stage provided the receiving stage is ready to accept data. Data that arrives at the read interface of the FIFO can be read out from the pipeline. Basically, a read operation frees the last stage of the pipeline, allowing all data to shift one position towards the output. However, since the read operation creates only one free position, the whole content is not shifted instantaneously, but word by word. So it looks like the read operation creates a *'hole'* which ripples from the read side to the write side interface

* Paul Wielage is currently with NXP Semiconductors' IC Laboratory in Eindhoven, The Netherlands.
** Michel Altheimer is currently with NXP Semiconductors in Crolles, France, e-mail:michel.altheimer@nxpcrolles.st.com.

978-3-9810801-2-4/DATE07 © 2007 EDAA

in a direction opposite to the data. When the FIFO is completely full at the moment of a read operation, this hole must ripple all the way to the first stage before writing becomes possible again. In short, the handshake mechanism causes the data to line up at the read-side of the pipeline and, consequently, the empty positions at the write-side.

The read (write) interface of the FIFO module converts the internally employed asynchronous handshake protocol into a read (write) enable signal and an empty (full) status flag that are synchronous to the read (write) clock. This allows the FIFO to be smoothly integrated in a synchronously clocked context, possibly with different clock signals for the write and read interfaces.

The FIFO core is detailed in Figure 1(b). Each stage consists of a data register and a stage controller (SC). The data register is built of latches, with one latch per bit of a word. The stage controller determines whether the register is in hold (full) or transparent (empty) mode by means of signal $\mathtt{G}_i = 0$ and 1 respectively. The communication between the stages is via channels based on the concept of *bundled data* [4, 5]. In here, the data is encoded on a bus of bit signals and presence resp. acceptance of valid data on the data bus is encoded by two signals $\mathtt{R}_i$ and $\mathtt{A}_i$ according to a four-phase handshake protocol. In Figure 1(b) the data busses are depicted by the big arrows.

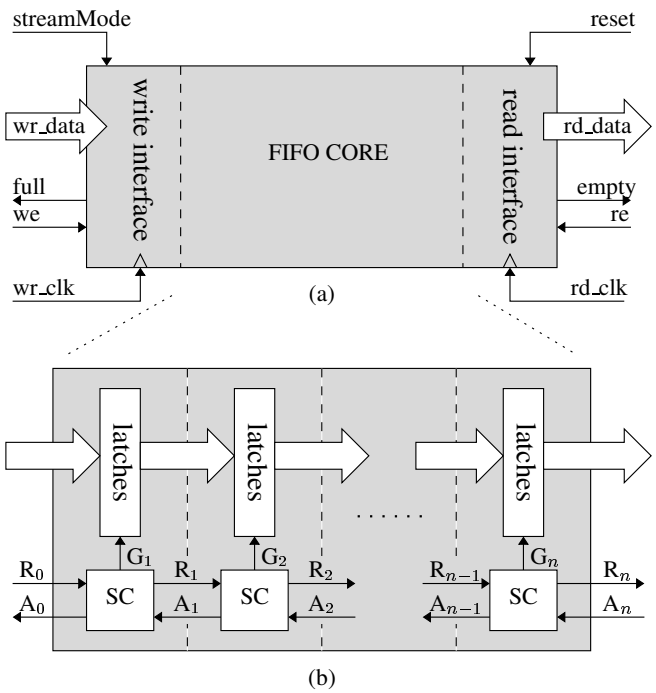

Figure 1: The FIFO module (a) and its internal asynchronous core (b).

Other micropipeline implementations which exploit bundled data are MOUSETRAP [6], IPCMOS [7], GasP [8], LP_{SR}2/1 [9], and LP_{HC} [10]. Our implementation is different in that it (1) employs an optimized data latch consisting of seven transistors and (2) provides very fast propagation of holes from the read to the write interface, which improves the slowest timing arc of a micropipeline design significantly.

The remaining subsections describe the data path, the control path, and the write and read interfaces in more detail.

2.1 Data Path

The schematic of the data path latch is given in Figure 2. Transistors `P1`, `P2`, `N1`, and `N2` form two cross-coupled inverters for keeping the state. Transistors `N3`, `N4`, and `N5` allow changing the state. The bits in the data path are dual-rail encoded, i.e., both the bit value and its inverse are stored. This allows to employ an SRAM-like write technique in the transparent mode of the latch. To that end, transistor `N5` must be turned on by a high level on `G` such that the true or the complement signal of the cell is forced to ground, depending on the data input `D` (and its complement `DN`).

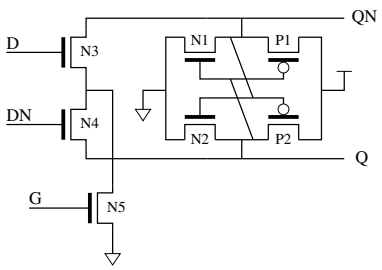

Figure 2: The static latch as used in the data path.

2.2 Control Path

Transfer of data between two successive stages in the pipeline is based on a four-phase handshake protocol. The typical way of four-phase handshaking is shown in Cycle 1 of Figure 3, including the dashed arcs. Since we choose for the early data-valid scheme, the rising edges of $\mathtt{R}_i$ and $\mathtt{A}_i$, at the end of Phases 1 and 2, indicate presence resp. acceptance of data on the channel [4]. With typical four-phase handshaking, $\mathtt{R}_i$ and $\mathtt{A}_i$ return to zero as shown in Phases 3 and 4, where the first transition is a condition for the second. This strict ordering guarantees that both the request and acknowledgement have been seen by the other side.

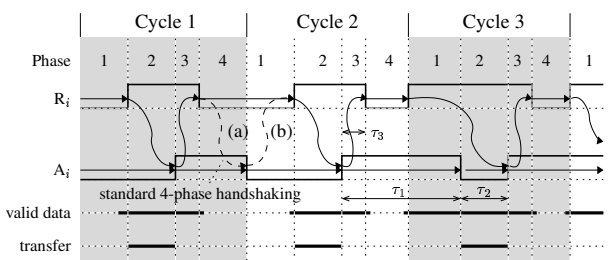

Figure 3: Three possible cycles of the modified 4-phase handshake protocol.

To improve the propagation speed of both data words and holes, we have implemented an alternative four-phase protocol, which is one without the dashed arcs. Now, consistant data transfer is guaranteed when two conditions are met. Firstly, $\tau_3 < \tau_1$ which implicitely implements arc (a) and, secondly, the stage controller is (level) sensitive to $\mathtt{R}_i$ only when $\mathtt{A}_i$ is low which makes arc (b) superfluous. The difference with standard four-phase handshaking is depicted in Cycle 3. In particular, it shows that the modified protocol allows a new request before de-assertion of a previous acknowledgement. Given this protocol, the relation between the request signal to the next stage, the acknowledgement signal to the previous stage and $\mathtt{G}_i$ becomes very

simple: $R_i = A_{i-1} = \overline{G_i}$, provided that G_i becomes low after an incoming request event and immediately high after an incomming acknowledgement event.

The danger of omitting the dashed arcs is two-fold: unintended data duplication and deadlock. The first problem is solved by the timing constraint $\tau_3 < \tau_1$. The second problem is solved by setting a lower bound on τ_2. Furthermore, the latter delay constraint also guarantees that data can be captured robustly in the latch register. By introducing these timing contraints, we effectively change the controller from speed-independent into self-timed [5].

Our modified four-phase handshake protocol is summarized in Table 1. Given the previously introduced encodings for R_i and A_i, data transfer happens during Phase 2. During this phase (1) valid data is present since G_i=0 and (2) the receiving stage is empty since G_{i+1}=1. In the first and fourth phases, the channel is waiting for these two conditions to become true. Since both events can happen in arbitrary order, Phase 1 has two possible encodings. Finally, Phase 3 is devoted to the immediate de-assertion of R_i to prevent duplication of data as explained.

Phase	R_i / A_i	Action
1	00 or 11	wait for the second data transfer condition
2	10	transfer of data
3	11	de-assert request signal
4	01	wait for a first data transfer condition

Table 1: Modified four-phase protocol with encoding.

To conclude, the proposed handshake protocol allows concurrent signaling of data presence and acceptance of data on the channels between the stages and this contributes to high-speed operation.

2.3 Read and Write Interfaces

The FIFO supports three basic operations: (1) *write*, in which data is written into the FIFO under control of `wr_clk`, `we`, and `full`, provided it is not full, (2) *read*, in which data is read out of the FIFO under control of `rd_clk`, `re`, and `empty`, provided it is not empty, and (3) *shift*, in which all data words in the FIFO are shifted one stage towards the output. While the write and read operations can be performed truly asynchronous, the shift operation, which is actually a combined write and read operation, only works if `wr_clk` and `rd_clk` are synchronous to each other. The shift operation can be used when the FIFO is embedded in a synchronous context, or when the two clock signals are explicitly made synchronous. The latter is the case in our test mode, as described in Section 3.

The write interface implements four functions: (1) conversion of the synchronous timing convention at the interface into the handshake timing convention of the FIFO core, (2) conversion of single-rail data inputs into dual-rail, (3) capturing the data bus at the rising clock-edge, such that intermediate transitions on the data bus in between write operations are blocked, and (4) enabling a shift operation even when the FIFO is full. The write interface is equipped with an additional latch register. This supports Function (3) by allowing the first register stage to effectively behave as a master/slave flip-flop. If signal STREAMMODE is asserted, the same register implements Function (4). During shift operation it can be used as an extra word location to temporary store a written word while a read-created hole is still making its way towards the write interface.

The read interface implements four similar functions: (1) conversion of the synchronous timing convention at the module interface into the handshake timing convention of the FIFO core, (2) buffering of the data output bus, (3) providing a flush mechanism to reset the FIFO to its empty state, and (4) support of the stream mode. The reset function is implemented as a sequence of asynchronous read operations for as long as the FIFO contains data and the reset is enabled. The support of the stream mode is simply a feed-through of the `re` signal to the write interface.

2.4 Layout

Three FIFO instances have been developed in a 90 nm CMOS technology. Table 2 lists the three instance sizes and corresponding silicon area, and compares that with SRAM-based and standard-cell-based alternatives.

Words × Bits	Silicon Area including DfT				
	SRAM		Standard Cell		New
	(μm^2)	(Δ%)	(μm^2)	(Δ%)	(μm^2)
16 × 19	10,869	+278	12,580	+338	2,875
32 × 37	20,043	+206	35,814	+446	6,554
64 × 37	25,421	+122	62,319	+444	11,453

Table 2: Currently available FIFO instances and their area sizes.

The layout of the 16×19-bit instance is shown in Figure 4. The largest layout block is formed by the memory-cell matrix. It contains 17 words (horizontal rows in the figure) of 19 bit cells each; 16 regular words plus the additional latch register that is enabled in the stream mode. The orientation of all memory cells is equal; hence data signal `D` of one bit is physically next to data-not `DN` of an adjacent bit in the same word. Power lines are routed left and right of the memory-cell matrix, as well as through the middle. The second-largest layout block is formed by the asynchronous stage control cells. The control cells are laid out in eight horizontal rows of two control cells each. The height of one row of control cells corresponds to the height of two rows of memory cells. A similar layout set-up works for all three FIFO instances, as they all have an even number of words. Other layout blocks implement the input and output buffers and write and read interfaces.

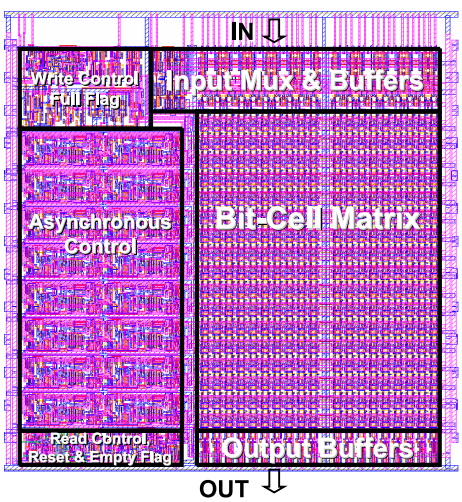

Figure 4: The layout of the 16×19-bit FIFO instance.

3 FIFO Test and DfT Approach

3.1 Prior Work in FIFO Testing

Conventional FIFO designs are based on either an SRAM or standard cells. SRAM-based FIFOs are typically tested with a dedicated BIST per FIFO as in the approaches by Barbagallo et al. [11], Van de Goor et al. [12], and Zorian et al. [13]. However, for small FIFOs, the corresponding DfT area is relatively large; a dedicated BIST would take 130% for our smallest 16 word × 19 bits FIFO. This becomes prohibitively expensive in case a large number of FIFOs are used in one IC design. In order to reduce the DfT area costs, Grecu et al. [14] propose to use one common BIST controller and stimulus generator for multiple FIFOs, while only the response evaluator is dedicated per FIFO. To be efficient, this approach requires all FIFOs to be of the same size, which unfortunately is rather uncommon. Also, the area saved by sharing controller and stimulus generator might be spent again for the wiring between the shared BIST and the various FIFOs [15].

Two different approaches exist to test standard-cell FIFOs. The first approach is to make the latches or flip-flops that constitute the FIFO memory scan-testable and integrate them into the on-chip scan chains. The associated DfT area costs are high, but the FIFO can now be tested by ATPG-generated patterns as integral part of the IC's logic. The second approach is to treat the FIFO as a separate module that requires a dedicated test. Rearick [16] follows this second approach and uses, in a pre-test-wrapper era, scan chains surrounding the FIFO to provide test access; while test generation for embedded FIFO and surrounding logic is separate, the actual execution of both tests is merged into one. A potential drawback of this approach is that there might be logic gates between the embedded FIFO and its surrounding scan chains, that will complicate test generation and access.

Shi et al. [17] describe standard-cell based asynchronous FIFOs and propose a test method based on ATPG that targets both stuck-at and delay faults in the micro-pipeline. As DfT structure, a two-input multiplexer is added at each pipeline stage. This seems an expensive solution, but area costs are not mentioned. The paper also does not discuss if embedded FIFOs are wrapped for test, and how test access to them is achieved.

Van de Goor et al. [12] discuss on a high level defects and fault models. However, actual defects such as shorts and opens are difficult to model at high level. Our test solution, as explained below and in companion paper [18], aims at identifying both hard and weak shorts and opens.

3.2 Modular Test with Partial Test Wrapper

Non-logic modules embedded into an environment consisting of digital standard-cell logic are traditionally tested as separate units by dedicated tests; this was already the case even before the terms 'SOC' and 'core-based testing' were invented [19]. Examples of such non-logic embedded modules are custom-designed blocks like memories, analog, and mixed-signal blocks. The reason for testing them as separate units is that these modules have circuit structures different from standard-cell logic, which consequently exhibit different defect mechanisms and fault behavior. Our FIFO is also a custom-designed hard macro, and hence, we adopted from the start of the project the approach to use a dedicated test.

The common design-for-test approach that enables separate testing of embedded modules is to encapsulate the module with a *test wrapper* [20]. The test wrapper switches between the functional and test connections of the module. It supports at least three modes: (1) normal mode, in which the functional connections are enabled and the test infrastructure is switched off, (2) INTEST mode, in which the module is tested internally while external influences are blocked, and (3) EXTEST mode, in which the test wrapper participates in testing the circuitry external to the module while blocking signals internal to the module. Test wrappers with this functionality are nowadays quite common [21, 22] and have been standardized as IEEE Std. 1500 [23, 24]. For embedded memories, it is quite common to extend the functionality of the test wrapper with on-chip stimulus generation and response evaluation capabilities, commonly termed *Built-In Self Test* (BIST) [15].

For our FIFO, we decided *not* to implement a full test wrapper. A first-order approximation showed that the silicon area required for implementing only the data portion of such a wrapper (i.e., excluding the Wrapper Instruction Register [23]) for the smallest 16×19-bit FIFO would in 90 nm CMOS technology take about 1,500 μm^2. Compared to the size of the FIFO, 2,500 μm^2, this implies an area cost of well over 50%! The principal reason of existence for our full-custom FIFO is its small area size, compared to implementation alternatives based on SRAM or standard cells. With a conventional IEEE Std. 1500 compliant test wrapper, the FIFO's area advantage would be completely destroyed. The *absolute* sizes of the FIFO and its wrapper are small, and hence, these area costs are bearable in case only one or few wrapped FIFOs are integrated into an SOC design. However, the intended use of the FIFO is especially for SOC designs in which several hundreds of FIFOs are integrated, and in that scenario, the area costs of equipping the FIFOs with a full-fledged test wrapper would become disproportionally expensive.

Instead of implementing a full test wrapper, we implemented only a partial wrapper [25]. We used wrapper cells like IEEE Std. 1500 type WC_SD1_COI [26, 23] to provide both controllability and observability to FIFO inputs `we`, `re`, `streamMode` and `reset` and FIFO outputs `full` and `empty`. Per IEEE Std. 1500, clock signals `wr_clk` and `rd_clk` are exempt from having wrapper cells.

The write and read data ports `wr_data` and `rd_data` are *not* equipped with wrapper cells. This cuts most of the area costs of a full wrapper and saves 38 (for the smallest FIFO instance) to 74 (for the larger FIFO instances) wrapper cells. A Wrapper Instruction Register (WIR) and associated Wrapper Serial Control (WSC) port, both mandatory per IEEE Std. 1500, are *not* supplied either.

3.3 FIFOs Integrated in Scan Chains

During its shift operation, i.e, when simultaneous write and read operations are applied, an $n \times m$-bit FIFO (i.e., a FIFO consisting of n words of m bits each) filled with i words ($1 \leq i \leq n$) actually behaves as m parallel scan chain segments of length i bits each. For test, we exploit this fact by including all FIFOs into on-chip scan chains. In this way, the FIFO's write- and read-data inputs become fully controllable and observable from chip pins, and hence are compensated for their lacking wrapper functionality.

Inclusion into the on-chip scan chains requires an $n \times m$-bit FIFO to be equipped with an m-bit wide scan-in port `si[m-1:0]`, an m-bit

wide scan-out port `so[m-1:0]` (tapped off from the `rd_data` output), and a scan enable control input `se`. Note that the FIFO requires a filling of at least one word (i.e., to be non-empty) in order to be able to shift data. As described in Section 2.3, signal `streamMode` should be asserted to be able to shift a full FIFO, i.e., a FIFO with a filling of n words.

The on-chip SOC-level TestRail architecture design [21] is not constrained by the inclusion of the FIFO scan chain segments. The SOC integrator can freely design a TestRail architecture, as long as all FIFO scan chain segments are connected into one or more scan chains. It is possible to create dedicated 'FIFO scan chains', but it is also allowed to mix and match with regular scan chain segments that run through the surrounding standard-cell logic. It is possible to concatenate the scan chain segments of the FIFO(s) into any smaller number k ($k \geq 1$). This allows our approach to be used even for SOCs which allow only very few scan chains, such as very-low pin-count SmartCard ICs.

3.4 FIFO InTest Procedure

For an $n \times m$-bit FIFO (with n even), our INTEST procedure consists of three steps, as listed in Table 3. Note that the reset, write, and shift operations are all normal functional modes of the FIFO.

Step	Operation	#ops
1	*Reset*	1
2	*Write* $(00\ldots0;11\ldots1)^{n/2};00\ldots0$	$n+1$
3	*Shift-Out* $(00\ldots0;11\ldots1)^{n/2}$	n

Table 3: FIFO INTEST procedure.

Upon start-up of the SOC, the FIFO is in an unknown state. In order to bring it to a known state, Step 1 resets the FIFO by applying `reset`$= 1$ for one clock cycle, thereby effectively flushing the FIFO's content.

Step 2 tests whether the FIFO correctly performs write operations at different levels of being filled. At every distinct fill level, a write operation is confronted with different delay paths, which need to be within certain margins for the FIFO to operate correctly. In order for Step 2 to cover all fill levels, it is important that only write operations are executed, and no read (or shift) operations. In this way, we cover all fill levels, from empty to full. For an n-word FIFO, we write $n+1$ words, in order to let the last write operation test if the FIFO indeed refuses to write additional data, once full. The alternating sequence of 00...0 and 11...1 words enables to distinguish between neighboring words of the FIFO. In the layout, each word is in its turn an alternating sequence of bit and bit-not wires, such that the test data fills the FIFO with a physical *checkerboard* pattern. Next to the delay paths at different fill levels, this test also aims to detect single-cell stuck-at and transitions faults, as well as coupling faults between neighboring cells.

In Step 3, we read out the FIFO content by shifting. Shifting a full FIFO actually exercises the worst-case timing paths in the FIFO, as for every individual shift operation a hole needs to be propagated from the read interface to the write interface through the entire FIFO. The FIFO data is shifted to the chip pins, where the ATE compares actual with expected responses. Note that depending on the actual chip-level scan chain configuration (see Section 3.3), the responses scanned out of the FIFO might require propagation through other FIFOs and/or logic scan chain segments before actually reaching the chip pins.

3.5 Low-Cost 'Filler' DfT

In Step 2 of the INTEST procedure, we cannot perform the write operations at different fill levels by simply shifting all FIFOs together as part of the SOC scan chains. Write data could be provided via the scan chains only if the subsequent FIFOs execute and complete their respective Steps 2 one by one. This would require a test control infrastructure that allows us to subsequently activate the FIFOs. This could be implemented by means of a dedicated on-chip counter per FIFO or, alternatively, a dedicated control pin per FIFO; both solutions were considered prohibitively expensive. In a very-low pin-count SOC, such as a SmartCard IC, where we need to concatenate even the scan chain segments through a single FIFO, Step 2 of our test cannot be performed at all by means of scan access, as it would require bit i of the FIFO words to shift, while bit j is only written and does *not* shift (for $1 \leq i < j \leq m$).

To overcome the above difficulties, we have implemented additional write access DfT. This so-called *Filler* is a degenerate version of BIST, which is capable only of providing test stimuli, and only in a restricted way. It consists of three input bits to the FIFO, named `fillerData[1:0]` and `fillerEnable`. When control input `fillerEnable` is asserted, the words written into the FIFO are determined by `fillerData`, such that all even bits correspond to `fillerData[0]` and all odd bits correspond to `fillerData[1]`. This Filler DfT gives us a restricted, but low-cost and efficient way to control the data written into a FIFO. Due to the restrictions imposed by the Filler, test stimuli for the FIFO can now be characterized and denoted by two bits only; one representing all even bits, and one representing all odd bits in the stimulus words.

The envisioned usage of the three additional Filler inputs at SOC integration level is that they will be connected up for all FIFOs per signal, allowing for a broadcast of the same Filler signals to all FIFOs simultaneously. Such a connection scheme would allow to carry out Step 2 of our test for all FIFOs in parallel. Steps 1 (reset) and 3 (scan-out) could already be executed in parallel for all FIFOs, such that now effectively the entire test can be carried out in parallel for all FIFOs.

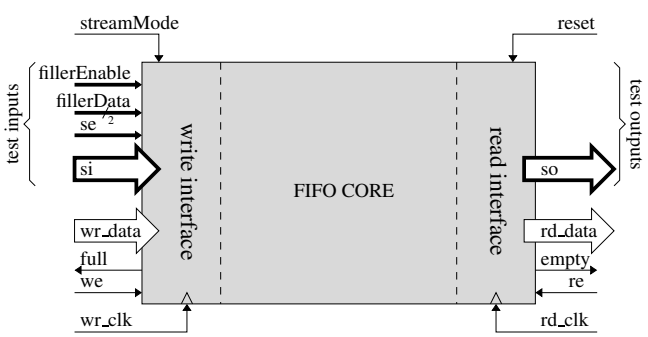

Figure 5: The FIFO module and its test-specific terminals.

Figure 5 shows the FIFO module and highlights the additional test-specific inputs and outputs. They are `si[m-1:0]`, `so[m-1:0]`, and `se` for scan access, and `fillerData[1:0]` and `fillerEnable` for Filler access. The overall DfT area costs of our solution is 15%, 7.1%, and 4.1% for the three instances respectively.

4 Conclusion

Many IC designs use numerous embedded FIFO memories for intermediate storage, data rate conversion, and clock domain crossing. The usage of embedded FIFOs is expected to grow, as new design paradigms such as Network-on-Chip (NOC) and Globally-Asynchronous Locally-Synchronous (GALS) use FIFOs extensively. We have developed a new embedded asynchronous FIFO module, based on a micropipeline architecture. Due to its full-custom design, the new FIFO is substantially smaller than SRAM-based and standard-cell-based counterparts; the savings are 3.4 mm^2 and 7.3 mm^2 in 90 nm technology for 250 FIFOs of 32 words $\times$ 37 bits, respectively.

As hard macro, the FIFO is tested with a dedicated test. The initial test procedure is described in this paper, while a companion paper describes the process of analyzing and improving the detection qualities of this test [18]. Contrary to what is common for embedded hard macros, the FIFO is *not* equipped with a full test wrapper, in order to maintain its area advantage. Instead, the FIFO has a partial test wrapper, while the data path has been made controllable and observable during testing by integrating it into the regular on-chip scan chains. A dedicated 'Filler' interface allows restricted write access, which is sufficient to test multiple FIFOs in parallel. With our approach, the overall DfT costs range between 4.1% and 15% of the bare FIFO silicon area.

Acknowledgements

The authors thank Mohamed Azimane of NXP Research in Eindhoven, The Netherlands, and Tobias Dubois and Erik Larsson of Linköpings Universitet in Linköping, Sweden for working with us on the improvement of the FIFO test procedure [18]. Furthermore, we thank Erik van Geest of NXP Semiconductors and Ananta Majhi, Bart Vermeulen, and Kees Goossens of NXP Research in Eindhoven, The Netherlands for constructive criticism on an early draft of this paper.

References

[1] Kees Goossens, John Dielissen, and Andrei Rădulescu. The Æthereal network on chip: Concepts, architectures, and implementations. *IEEE Design and Test of Computers*, 22(5):21–31, Sept-Oct 2005.

[2] Jens Muttersbach, Thomas Villiger, and Wolfgang Fichtner. Practical design of globally-asynchronous locally-synchronous systems. In *Proceedings Intnl. Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 52–59, Eilat, Israel, April 2000.

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

[4] Ad M.G. Peeters. *Single-Rail Handshake Circuits*. PhD thesis, Eindhoven University of Technology, June 1996.

[5] Jens Sparsø and Steve Furber, editors. *Principles of Asynchronous Circuit Design: A Systems Perspective*. Kluwer Academic Publishers, Dordrecht, The Netherlands, September 2001.

[6] Montek Singh and Steven M. Nowick. MOUSETRAP: Ultra-High-Speed Transition-Signaling Asynchronous Pipelines. In *Proceedings International Conference on Computer Design (ICCD)*, pages 9–17, Austin, TX, USA, September 2001.

[7] S. Schuster et al. Asynchronous Interlocked Pipelined CMOS Circuits Operating at 3.3-4.5 GHz. In *Proceedings International Solid State Circuits Conference (ISSCC)*, pages 292–293, San Francisco, CA, USA, February 2000.

[8] Ivan Sutherland and Scott Fairbanks. GasP: A Minimal FIFO Control. In *Proceedings Intnl. Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 46–53, Salt Lake City, UT, USA, March 2001.

[9] Montek Singh and Steven M. Nowick. High-Throughput Asynchronous Pipelines for Fine-Grain Dynamic Datapaths. In *Proceedings Intnl. Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 198–209, Eilat, Israel, April 2000.

[10] Montek Singh and Steven M. Nowick. Fine-Grain Pipelined Asynchronous Adders for High-Speed DSP Applications. In *Proceedings of the IEEE Computer Society Workshop on VLSI*, pages 111–118, Orlando, FL, USA, April 2000.

[11] S. Barbagallo et al. A Parametric Design of a Built-In Self Test FIFO Embedded Memory. In *Proceedings IEEE Intnl. Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, pages 221–229, Boston, MA, USA, November 1996.

[12] Ad J. van de Goor, Ivo Schanstra, and Yervant Zorian. Functional Test for Shifting-Type FIFOs. In *Proceedings European Design and Test Conference (ED&TC)*, pages 133–138, Paris, France, March 1995.

[13] Yervant Zorian, Ad J. van de Goor, and Ivo Schanstra. An Effective BIST Scheme for Ring-Address Type FIFOs. In *Proceedings IEEE International Test Conference (ITC)*, pages 378–387, Washington, DC, USA, October 1994.

[14] Cristian Grecu et al. Methodologies and Algorithms for Testing Switch-Based NoC Interconnects. In *Proceedings IEEE Intnl. Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, pages 238–246, Monterey, CA, USA, October 2005.

[15] Rob Aitken. A Modular Wrapper Enabling High Speed BIST and Repair for Small Wide Memories. In *Proceedings IEEE International Test Conference (ITC)*, pages 997–1005, Charlotte, NC, USA, October 2004.

[16] Jeff Rearick. Practical Scan Test Generation and Application for Embedded FIFOs. In *Proceedings IEEE International Test Conference (ITC)*, pages 294–300, Atlantic City, NJ, USA, September 1999.

[17] Feng Shi, Yiorgos Makris, Steven M. Nowick, and Montek Singh. Test Generation for Ultra-High-Speed Asynchronous Pipelines. In *Proceedings IEEE International Test Conference (ITC)*, pages 1009–1018, Austin, TX, USA, November 2005.

[18] Tobias Dubois, Mohamed Azimane, Erik Larsson, Erik Jan Marinissen, Paul Wielage, and Clemens Wouters. Test Quality Analysis and Improvement for an Embedded Asynchronous FIFO. In *Proceedings Design, Automation, and Test in Europe (DATE)*, Nice, France, April 2007.

[19] Frans Beenker, Karel van Eerdewijk, Robert Gerritsen, Frank Peacock, and Max van der Star. Macro Testing: Unifying IC and Board Test. *IEEE Design & Test of Computers*, 3(4):26–32, December 1986.

[20] Yervant Zorian, Erik Jan Marinissen, and Sujit Dey. Testing Embedded-Core Based System Chips. In *Proceedings IEEE International Test Conference (ITC)*, pages 130–143, Washington, DC, USA, October 1998.

[21] Erik Jan Marinissen et al. A Structured And Scalable Mechanism for Test Access to Embedded Reusable Cores. In *Proceedings IEEE International Test Conference (ITC)*, pages 284–293, Washington, DC, USA, October 1998.

[22] Teresa McLaurin and Souvik Ghosh. ETM10 Incorporates Hardware Segment of IEEE P1500. *IEEE Design & Test of Computers*, 19(3):6–11, May/June 2002.

[23] Francisco DaSilva, editor. *IEEE Std 1500™-2005, IEEE Standard Testability Method for Embedded Core-based Integrated Circuits*. IEEE Standards Association, New York, NY, USA, August 2005.

[24] Erik Jan Marinissen et al. On IEEE P1500's Standard for Embedded Core Test. *Journal of Electronic Testing: Theory and Applications (JETTA)*, 18(4/5):365–383, August 2002.

[25] Nur Touba and Bahram Pouya. Testing Embedded Cores Using Partial Isolation Rings. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 10–16, Monterey, CA, USA, April 1997.

[26] Erik Jan Marinissen, Yervant Zorian, Rohit Kapur, Tony Taylor, and Lee Whetsel. Towards a Standard for Embedded Core Test: An Example. In *Proceedings IEEE International Test Conference (ITC)*, pages 616–627, Atlantic City, NJ, USA, September 1999.