Using Existing Reconfigurable Logic in 3D Die Stacks for Test

Fanchen Zhang[1], Yi Sun[1], Xi Shen[1], Kundan Nepal[2], Jennifer Dworak[1], Theodore Manikas[1], Ping Gui[1], R. Iris Bahar[3], Al Crouch[4], and John Potter[5]

[1]*Southern Methodist University, Dallas, TX*
[2]*University of St. Thomas, St. Paul, MN*
[3]*Brown University, Providence, RI*
[4]*SiliconAid, Austin, TX*
[5]*ASSET InterTech, Richardson, TX*

***Abstract*— We propose an architecture for an FPGA-based tester for a 3D stacked IC. Our design exploits the underlying structure of the FPGA, allowing it to be used to efficiently store and apply predefined test patterns at a high bandwidth, reducing the FPGA resources required and often reducing scan shift toggling. The proposed approach and its advantages can generally also be applied to 2.5D multi-die circuits containing FPGAs.**

I. Introduction

Future 3D stacked integrated circuits (ICs) may implement the functionality of an entire board through multiple bare dies stacked directly on top of each other and connected by through silicon vias (TSVs). These dies may contain intellectual property (IP) from multiple sources and may include processors, memories, ASICs (application specific ICs), analog dies, and even FPGAs (field programmable gate arrays) [1]. In addition to dramatically increasing the functionality contained within a very small form factor, 3D stacked ICs also have significant performance benefits arising from the high-bandwidth, low delay, and low-power connections that TSVs and bumped connections can provide [2].

Similar advantages are also obtained in 2.5D multi-die scenarios, in which multiple die are laid side-by-side on a silicon interposer or embedded multi-die interconnect bridge (EMIB) [3]. 2.5D systems are already being produced by multiple manufacturers, including AMD, which makes a high performance graphics card using 2.5D technology, and IBM, which is manufacturing server chips in both 2.5D and true 3D, among others [4].

FPGAs have become increasingly important in a variety of areas. As their performance and density have increased, they have become cost-effective replacements for ASICs in many applications. FPGAs are also used for performance acceleration through the on-demand instantiation of specialized hardware and for system repair. Intel has already announced that its EMIB will be used to connect its CPUs to Altera FPGAs to enhance performance and handle power issues [3].

In addition to these other applications, FPGAs have been used to aid in testing for many years, either by adding functionality to the load board when a chip is tested using ATE (automatic test equipment) at the factory or by serving as a tester for chips that are connected to it directly on a board [5]. This can allow a board to be partially tested even when all of the chips and board firmware are not yet available, for example during board development.

When used to test other chips on the board, an FPGA can serve as a generator of tests for a directly connected chip. For example, it could be programmed to contain a memory built-in-self test (MBIST) engine to send read and write commands to a directly connected memory chip. Alternatively, it may also serve as a target for functional or protocol-based tests—receiving/generating information from/to other chips based upon their functional behavior.

Just as an FPGA included on a board for other purposes can provide important test capabilities, an FPGA in a 3D stack can be repurposed, when desired, to provide critical testing functions as well. In fact, the advantages of using an FPGA as a tester on a board become magnified in the 3D IC space.

For example, one important issue in 3D is how and when to test each die in the stack. Bandwidth to upper die is likely to be limited to a few pins at the base die, and the P1838 Standard committee is currently investigating protocols and methods for the transmission of test data including, a TAP and TAP controller on every die, a serial boundary wrapper on every die interface (e.g. Upward and Downward) to conduct interconnect testing, and a parallel port to deliver high bandwidth test data. However, because the number of TSVs on a die can be much greater than the number of pins on a package, it may be possible to obtain significant additional test bandwidth by using many TSVs between an FPGA-based tester and the die under test to transmit test data.

These TSVs between the FPGA and another die may serve as functional communication buses under normal operation or could have been added for performance enhancement or repair. In either case, the high bandwidth available may allow a larger number of short chains to be accessed directly for scan-based testing—reducing the overall shift cycles and thus the energy dissipated (and heat created) during test.

Using an FPGA to test dies already in the stack will also allow for field-testing. For example, a tester could be instantiated on system boot-up—allowing errors arising from

This paper was supported in part by NSF grant CCF-1061164. .

978-1-4673-8949-5/16 $31.00 © 2016 IEEE
DOI 10.1109/NATW.2016.15

wearout, aging, warping of dies, etc. to be found. Tests could be selected or changed based upon functional behavior and history of the device. Furthermore, the test environment would be closer to the normal operating environment—helping to create more accurate tests, to provide valuable failure data for debug, and possibly even to allow field repair.

Finally, using an FPGA as a tester in a 3D stack provides significant additional security advantages over an FPGA on a board because the inter-die connections are hidden in the stack and cannot be physically probed. As a result, test data, including test patterns, may never appear outside of the stack, and side channel analysis, such as power or thermal analysis, is much less likely to be effective.

In this paper, we explore the implementation of one tester design that is intended to take advantage of the underlying FPGA structure. Specifically, we consider the case where specific ATPG patterns should be applied to the die under test and how those patterns can be efficiently stored in the lookup tables (LUTs) that form the programmable fabric of FPGAs. We explore both the FPGA resources required as well as the scan shift toggling expended. Test energy arising from scan shift toggling is especially important in 3D stack structures, where excess toggling may generate heat that is difficult to remove from the stack. Excessive toggling can also cause brownouts when the di/dt exceeds the capacity of power rails that have limited connections to the board.

The rest of this paper is organized as follows. Section II provides some more details on previous work in FPGA testers and 3D test. In Section III we discuss several reasons (other than test) for including FPGAs in a 3D stack. Section IV introduces our proposed FPGA tester design and Section V explores the resources and toggling required by our tester. Section VI concludes the paper.

II. PREVIOUS WORK

Replacing traditional test and measurement equipment with FPGAs on boards has been previously shown to help significantly reduce test costs and allows high-speed testing because FPGA-based instruments can be reconfigured as needed and have direct access to the DUT (Design Under Test) [6]. FPGAs have also been embedded into SoCs (Systems on Chips) to provide system test capabilities [7]. Using this approach, the FPGA may be reprogrammed for different functions at different times, so the FPGA may be used to add functionality to the chip, as well as being used as an embedded tester.

Various methods have also been developed for testing 3D stacks. For example, [8] discusses methods for scan-chain design and optimization for 3D ICs. They found that 3D scan-chain optimization achieves significant wire-length reduction compared to common 2D optimization approaches. The authors of [9] discuss DFT architecture and ATPG for interconnect test of 3D memory chips (DRAMs) and propose serial and parallel TAMs (Test Access Mechanisms) to communicate between dies. The serial TAM is used to transport test mode instructions and low-bandwidth test data, while the parallel TAM is used for high-bandwidth volume-production test data. There has also been significant research on the testing of TSVs [10], test scheduling [11], and the communication of test data between layers through the JTAG port [12]. However, test approaches for chip logic in 3D stacks have generally assumed that all test data will initially be provided through the bottom die by a tester (ATE).

III. FPGAS IN 3D STACKED ICS

Including FPGAs in the 3D stack can provide many advantages. In 2D, an FPGA can often provide the required performance while meeting area or power constraints. In addition, the re-programmability of FPGAs allows designs to be modified easily over a system's lifetime, as specifications or standards change, or even as design errors or enhancements are discovered. Finally, 2D versions of FPGAs have been used for performance acceleration, allowing co-processing hardware to be reconfigured "on-the-fly" when a particular portion of the code can benefit [13]. It is reasonable to expect that these advantages of FPGAs will likely carry over into the 3D IC space.

FPGA companies are already proposing, and in some cases manufacturing, 2.5D and 3D systems containing FPGAs. In recent years, Altera and Amkor have proposed a face-to-face packaging approach consisting of a mother die (FPGA) and daughter die (ASIC) [14]. Xilinx currently produces a Virtex 7 FPGA that contains four FPGA dies sitting side-by-side on a silicon interposer, aiding in prototyping and emulating large processor systems [15]. Intel is planning to ship its first server chips containing its CPUs and Altera FPGAs combined into multi-chip modules to leading edge cloud customers in the first quarter of 2016 and will place them in mass production in 2017 [16].

Our previous work has demonstrated the advantages of including FPGAs in a 3D stack for built-in self-repair [17], [18]. This paper expands upon our previous work by using FPGAs in a 3D stack for built-in self-test.

IV. FPGA-BASED TESTER ARCHITECTURE

As individual dies become more complex, the need for embedded instruments (such as sensors, hardware monitors, environment monitors, built-in-self test (BIST) engines, trace buffers, etc.) will only grow. They are likely to be needed not only for manufacturing test and failure or yield-analysis, but also to identify and address aging, wearout, and thermal issues in the field, and to verify or configure inter-die communication. An FPGA in a 3D stack may be used as a controller for these instruments or it may be used to implement some instruments, such built-in-self-test (BIST) pattern generators, itself.

One type of BIST pattern generator that may be implemented either in a die or on an FPGA is an LFSR-based LBIST (logic BIST) engine. Although adding weights and test points can increase the coverage of LBIST, top-off patterns may still be needed to achieve high coverage. Thus, in this section, we describe one possible FPGA-based tester

architecture that is capable of generating *specific* patterns to apply to a die-under-test (such as those that may be needed for top-off) while making use of the underlying FPGA architecture to reduce the resources needed for the design.

To meet these goals, our chosen FPGA-based tester stores the data to be shifted into the chains on different patterns into 1-bit LUTs on the FPGA. As an example, Fig. 1 shows how the outputs of a set of LUTs are fed into a multiplexer's data inputs. The output of the multiplexer feeds into one of the scan chains on the ASIC through a TSV (possibly via a SerDes connection.) A counter is used to cycle through all of the entries in the LUTs so that they can be shifted out one-by-one into the chain. This same architecture is repeated for all chains in the design.

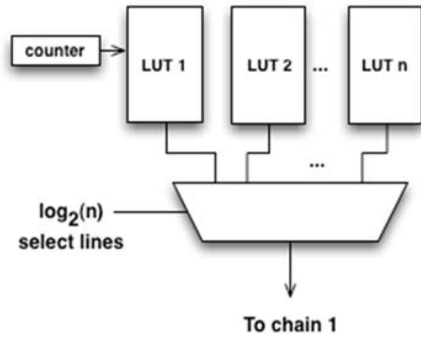

Fig. 1: Example FPGA-based implementation for storing pattern data for a single scan chain. This is repeated for multiple chains, with LUTs possibly shared among chains.

To save on FPGA resources, we can reduce the number of LUTs by merging compatible patterns into a single LUT that can be selected multiple times. Such merging may occur both among those patterns that will eventually be fed into a single chain as well as across chains, in which case a single LUT may fanout to multiple muxes.

Of course, the select line data is also needed. If the length of each chain is equal to the size of a LUT, one set of select lines must be stored per mux/chain for each pattern. For longer chains, more select line values would be needed so multiple LUTs may be unloaded in sequence during scan shift. These values may be stored in the FPGA itself, in a memory located in the stack, in a memory on the board, or they may be passed to the stack by an external tester. In our experiments, we used 5-input LUTs to store 32 bits of data to feed 32-bit chains, so each mux requires one set of select line values to be stored per pattern.

A. *Merging Algorithm*

The LUT design process starts with a synthesized Verilog circuit netlist, which undergoes scan insertion. Using the Mentor Graphics Tessent tool, an ATPG pattern set for stuck-at faults is generated using the "set_atpg_fill X" command so that don't cares (Xs) in the patterns are retained. The patterns are broken up into a set of 32-bit chains containing both input and flip-flop values. (We assume that output values will be fed into a response compactor and do not need to be merged.)

Once the patterns have been generated and divided between multiple scan chains, they must be assigned to LUTs. This process is summarized in Fig. 2. There are two constraints here—the number of LUTs needed and the number of select lines on the multiplexers (mux). To keep the size of the muxes manageable, we give preference to the reduction of select lines on muxes. For each chain, we analyze the patterns that will be applied to that chain and see if different patterns can be merged into a single LUT. We also look to see if patterns across different chains can be merged to reduce the total number of LUTs. Note that a pattern can only be merged with a member of the current global list of LUTs (which we call a LUT pool), if for all bit positions of the pattern, the bits are compatible between the pattern and the LUT. An X merged with a defined value (0 or 1) is replaced by the defined value in the merged LUT. In each case, we need to keep track of which of the muxes each LUT connects to and when that LUT should be selected (i.e., for which patterns) for each chain.

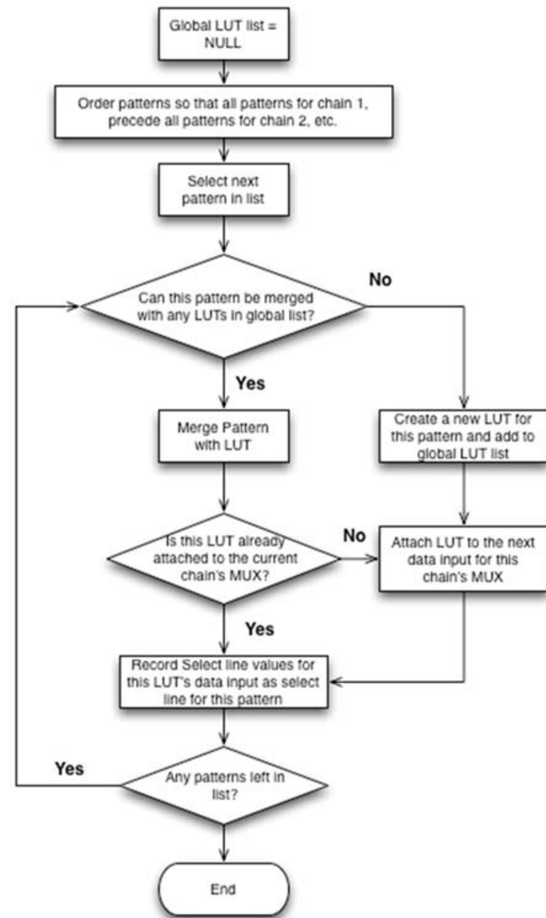

Fig. 2: Flowchart for LUT and Select Line Reduction.

B. *Example*

To help illustrate this compression methodology, consider the following example consisting of 3 chains, 4 patterns, and

5 bits per chain, with patterns shown in Table I. To reduce the LUTs and select lines required, we must merge the patterns when possible, taking the following steps:

TABLE I: EXAMPLE PATTERN DATA BEFORE MERGE

	Chain 1	Chain 2	Chain 3
Pattern 1	01XX1	100X0	XX1X1
Pattern 2	1XX11	11XX1	110XX
Pattern 3	X0XX0	1X001	1X0XX
Pattern 4	XX11X	101XX	X1XX1

1. Because the LUT pool is empty, we push the first pattern of Chain 1 (01XX1) into the LUT pool. This LUT is added to the first data input of Chain 1's mux, and the select line value for Pattern 1, Chain 1 is set to 0.

2. Pattern 2 of Chain 1:1XX11. This pattern cannot be merged with the LUT pool so we must create a new LUT. The new LUT is added to the next data input for Chain 1's mux, and the select line value 1 for the pattern is recorded.

Now *LUT pool: 01XX1, 1XX11. Chain 1's LUTs: 0,1; Chain 1's Select lines:0,1.*

3. Pattern 3 of Chain 1: X0XX0. X0XX0 cannot be merged with LUT0(01XX1) or LUT1(1XX11). Add the pattern to the pool, attach the LUT to the 3rd data input of Chain 1's Mux, and record the select line value.

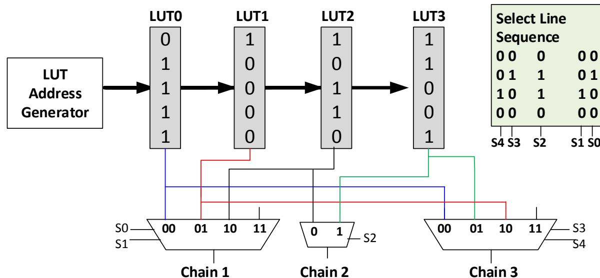

Fig. 3: Resulting implementation for patterns shown in Table I after pattern merging.

Now *LUT pool: 01XX1, 1XX11, X0XX0. Chain 1's LUTs: 0,1,2 Chain 1's Select lines:0,1,2.*

4. Pattern 4 of Chain 1: XX11X. This pattern can be merged with LUT0 (01XX1). Create merged pattern 01111 and replace LUT0 in the pool with this merged pattern. Since LUT0 exists in the LUT pool and is already attached to this chain's mux at data input 0, it does not need to be added to another data input. However, the select line value 0 must be recorded for this chain and pattern 4.

Now *LUT pool: 01111, 1XX11, X0XX0. Chain 1's LUTs: 0,1,2; Chain 1's Select line values :0,1,2,0.*

5. Pattern 1 of Chain 2: 100X0. This pattern can be merged with LUT2 (X0XX0) to create 100X0. Replace LUT2 with this new merged pattern in the pool. Add LUT2 to Chain 2's MUX 0th data input and record 0 as the select line value for Chain 2, pattern 1.

This process continues until we have attempted to merge all of the patterns. To store the final data into the LUTs, we replace any remaining don't cares (Xs) with 1s and 0s using the adjacent fill technique. This gives us our final LUT pool: 01111, 10000, 10110, 11001.

The final implementation is shown in Fig. 3. The merging process allows LUTs to be shared between chains and also allows the size of the muxes to be reduced when the same LUTs can be used multiple times for each chain.

Although this example assumed uncompressed patterns with many X's, it is still compatible with patterns with fewer X's at the cost of less merging. Eventually, if no X's are available, and if the number of repeated pattern sequences are small (as could happen with embedded deterministic test (EDT) [19]), then other variations could be needed. For example, it might become more efficient to store the patterns directly in the FPGA memory or to store some of the data, such as the select line data, off-chip.

V. IMPLEMENTATION RESULTS AND ANALYSIS

To evaluate the effectiveness of the FPGA-based tester architecture outlined in Section IV, we ran several experiments on different benchmark circuits obtained from **opencores.org**. These circuits were synthesized with Synopsys Design Compiler using a 90 nm ASIC library. We assumed that the primary inputs and primary outputs would be registered. Mentor Graphics Tessent was originally used to insert a single scan in each circuit. This scan chain was then subdivided to form multiple scan chains of length 32 containing only PIs and/or flip-flops (with the final chain possibly containing fewer scan cells when the original chain plus the PIs was not evenly divisible by 32). Stuck-at fault ATPG patterns were generated with Tessent as well. Details regarding each of the circuits studied are provided in Table II.

Note that such circuits could very easily represent a core on a chip that needs to be tested using top-off patterns after LBIST. Furthermore, although the tester design may be used to apply top-off patterns only, in these experiments we will store and apply the entire test set for each circuit.

TABLE II: OPENCORES.ORG BENCHMARKS

	PI	PO	FF	Fault	Pattern	Chain
colorconv	299	35	584	36534	82	28
des56	134	68	193	13962	113	11
fm_receive	12	13	501	17664	411	17
fpu_double	138	71	5231	264096	294	168
quadratic	36	25	120	6448	40	5

We assumed that capture values of the flip-flops and primary outputs would be scanned out and analyzed using an output signature analyzer, such as a MISR. However, because a comparison of different output compression methods is not a goal of the current research, no specific result compactor was selected or implemented. Note that no X's appeared in the scan chains' capture values during test for

any of the circuits studied. When tests do generate X's in the capture values, they must be masked if a MISR will be used.

To provide a proof-of-concept implementation of our design outlined in Section IV, we mapped the tester architecture to a Xilinx Artix-7 (XC7A200T) FPGA device using Xilinx ISE software. The Artix 7 series configurable logic block (CLB) provides real 6 and 5 input look-up tables (134,600 LUTs), distributed memory (2,888Kb), block RAM memory (13,140Kb) and shift register (1.444Kb) logic capabilities and fast wide multiplexers (16:1 MUX using 4 LUTs or 1 slice) for efficient FPGA fabric utilization [20].

These features of the FPGA are important for efficient implementation of our controller. Our controller will have a number of 5-input LUTs that each store a 32-bit pattern. These LUTs will be multiplexed with wide multiplexers. To take advantage of LUT sharing as described earlier and to reduce the total width of multiplexers as far as possible, the select lines for the multiplexers are predetermined and stored in another RAM block (implemented as either distributed RAM or block RAM on the FPGA). Fig. 4 shows a basic architecture of the test controller implemented on the FPGA. The structure consists of several modules—a LUT address generator, a LUT layer, a RAM address generator, a RAM layer, a multiplexer layer, a scan register, a signature checker, and a scan enable signal generator. The test controller has three inputs (CLK, RESET and a scan signature from the ASIC), four outputs (a scan enable signal and a reset sent to the ASIC), a registered bus feeding scan data to the ASIC via a SerDes, and an output that compares the signature received from the ASIC to indicate a test having passed or failed.

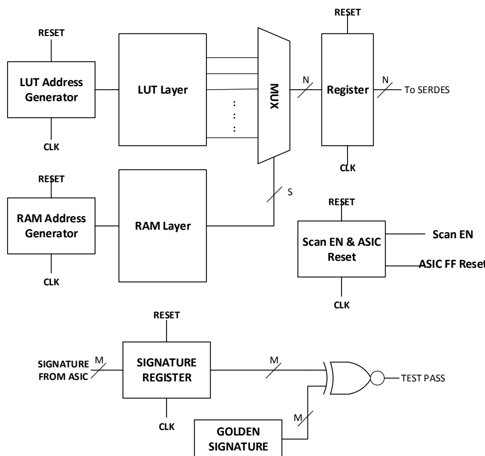

Fig. 4: FPGA-based tester block diagram

As already noted, we ran experiments on several circuits from **opencores.org** to validate the effectiveness of our approach. Two separate implementations for each circuit were generated—one where all modules were implemented as distributed RAM or slice LUTs in the FPGA and a second one where the mux select signals were all grouped into a larger Block RAM (BRAM) in the FPGA. For both experiments, we used Verilog HDL and synthesized it with Xilinx ISE 14.6 with a synthesis goal set to reduce the overall system area. Results appear in Tables III and IV.

Table III shows that the tester architecture takes up very little area on the FPGA and that the tester can be operated at a clock frequency of 163 to 257 MHz for Experiment 1. Note that the tester does not need to operate at the speed of a functional ASIC because the tester is primarily engaging in scan shift operations, which can occur at a much slower clock frequency. In fact, a slower clock frequency for scan shift is likely to be preferable to prevent thermal issues in the stack during test. The smallest circuit *quadratic* used negligible hardware resources and was the fastest while *fpu_double* used the most resources (10.3% of LUTs available) and could be run at just over 163MHz.

TABLE III: EXPERIMENT 1—ALL MODULES ARE DISTRIBUTED RAMS/SLICE LUTS

Circuits	Max Freq (MHz)	Slice LUTs	% use LUTs
color	219.4	2045	1.5%
des56	225.6	986	0.7%
fm_receive	173.9	1969	1.5%
fpu_double	163.7	13797	10.3%
quadratic	256.8	269	0.2%

TABLE IV: EXPERIMENT **2**—MUX SELECT LINES IMPLEMENTED IN BLOCK RAMS (BRAMS)

CKT	Max Freq (MHz)	Slice LUT	% use LUTs	Block RAMs	%use BRAMs
color	231.7	1760	1.3%	3	0.8%
des56	252.1	845	0.6%	1	0.3%
fm	214.9	1626	1.2%	3	0.8%
fpu	222.1	7337	5.5%	31	8.5%
quad	234.2	240	0.2%	1	0.3%

In Experiment 2, we see that the circuits use less LUTs compared to Experiment 1 because all the LUTs of Experiment 1 that were dedicated to storing the multiplexer select lines are now stored in one or more of the 365 available block RAMs (BRAMs). Keeping the select lines in BRAMs also helps increase the speed of four of the circuits. The small size of the *quadratic* circuit prevented it from really taking advantage of the BRAMs.

A. FPGA Occupancy Data

Another issue we wanted to explore was how much data reduction we were able to achieve with our current FPGA-based architecture as compared to simply storing the full test patterns in a memory—either in BRAMs on the FPGA or in another memory in the stack. Specifically, if LUTs could be used for multiple chains and/or multiple patterns, and if the number of select lines needed for each chain MUX was not too large, then the number of bits stored should be less. The data obtained for our 5 circuits is shown in Table V. (Note that this does not consider additional bits needed to implement the actual controller in the FPGA). The first column corresponds to the circuit name and the second to the original amount of test data that would need to be stored. This is simply equal to: *32×#chains×#patterns*, including padding, for chains of length 32. Column 3 corresponds to the number of bits stored for pattern pieces in the LUTs and is

equal to the number of LUTs identified with the algorithm in Section IV.B multiplied by 32. Column 4 adds the data for the select lines values on each pattern and is equal to the number of select lines needed for all chain muxes multiplied by the number of patterns.

TABLE V: DATA STORAGE REDUCTION

CKT	Original Total (bits)	LUT data (bits)	Select line (bits)	% ⇓ (LUT + sel)	% ⇓ (LUT only)	% ⇓ (sel only)
color	73472	36512	15252	29%	50%	79%
des56	39776	19072	7458	33%	52%	81%
fm	223584	35968	45210	64%	84%	80%
fpu	1580544	97728	322224	73%	94%	80%
quad	6400	5184	1200	0.3%	19%	81%

Column 5 corresponds to the percent reduction in data required when Columns 3 and 4 are added together and compared with Column 2. This assumes that the values on the select lines will be stored in the FPGA and are included in the data overhead. The percent reduction varies from a low of 0.3% for our smallest circuit to a high of 73% for our largest circuit. This is very encouraging because the percentage reduction increases significantly as the total original test data increases. This indicates that this FPGA-tester architecture appears to be fairly scalable.

Column 6 compares Column 3 and Column 2, trying to ascertain the percent reduction in data storage needed if only the data in the LUTs is considered. For example, this might be appropriate if we are worried about the occupancy of the FPGA but are obtaining the values on the select lines from an external memory. Now, the percent reduction is even larger. It varies from 19% for our smallest circuit to 94% for our largest circuit.

Finally, Column 7 compares the amount of data stored for select bits only (number of total select bits multiplied by the number of patterns) to the total number of bits in Column 2. This comparison is most appropriate from the perspective of how much data may need to be stored in an external memory for feeding to the FPGA. For all circuits, this total reduction in test data is approximately 80%. We compared this result with the reduction reported by authors of [21]. In [21], the authors performed data compression offline on a host computer and stored the compressed data on an external "slow" memory. The compressed data stored in the external memory is then decompressed by an FPGA before being presented to the DUT (Device under Test). The authors reported an average compression rate of 69.8% while the total reduction in our approach was 80%.

Thus, the selected FPGA-based tester architecture is highly effective at reducing the amount of test data that may need to be stored in an external memory or on the FPGA itself. Even more encouraging, the method appears to scale very well with increasing amounts of test data.

B. Switching Activity

Although we were able to compress our data well in the previous section, the overall compression rate is considerably less than is often achieved with on-chip decompressors alone. Of course, it is still possible, to write the decompressor's incoming channel data to LUTs or to on-chip memories in the FPGA. However, as already noted, in the presence of on-chip decompressors, the pattern sequences applied to the channels may not have any X's, making compression in the LUTs very difficult.

There are several reasons why this may not be a significant problem. First, as already noted, the patterns stored in the LUTs may correspond only to those top-off patterns that are needed to get coverage for random-pattern-resistant faults that are not covered by LBIST engines. This automatically reduces the test data volume that needs to be stored.

In addition, one of the reasons why such decompressors are needed is to reduce the test data bandwidth when the test inputs and outputs are limited to only a few pins. When an FPGA in a 3D stack is used, it may be possible to have many more chains on other dies accessed directly either through individual TSVs or through TSVs that are implementing SerDes. SerDes TSV channels are extremely efficient in 3D because of the very short distances between dies. This means that the test data bandwidth may automatically be higher in 3D between dies even without an on-chip decompressor, if we choose not to use one.

In addition, if test patterns are going to be generated or selected within the stack so that only a subset of all potential patterns in the set are applied to better match suspected defects or operating conditions, it might be necessary to set the decompressor to bypass mode and use patterns stored in the LUTs directly instead.

Finally, thermal issues during test are likely to be very problematic in 3D because it may be more difficult for heat to escape. Thus, reducing switching activity during scan shift is very important. Although low power ATPG for on-chip decompressors is possible with commercial tools, some approaches to reducing scan shift toggling, such as adjacent fill, are difficult or impossible to apply in the presence of on-chip decompressors because they depend on having a large number of X's. It may be easier to get low power test patterns from our approach if enough X's remain in the patterns to perform adjacent fill.

As a result, we investigated the difference in scan shift switching activity for both patterns shifted in as the output of a power-limited on-chip decompressor and for our original scan patterns with adjacent fill implemented after merging. Specifically the on-chip decompressor patterns were generated in Mentor Graphics Tessent with the low-power options—and we tried to limit switching activity to between 10% and 25% of the maximum possible. (Further limitations on switching lead to lower test coverage.) More than 200 test sets were created for each circuit with different threshold parameters to try to find the very lowest EDT toggling activity possible without losing significant fault coverage.

The switching activity comparisons are shown in Table VI.

TABLE VI: TOTAL TOGGLING FOR OUR METHOD VS. EDT

CKT	Patterns with Adjacent Fill		Embedded Deterministic Test (EDT)		% ⇑ with EDT
	Fault Cvrg.	# of Toggles	Fault Cvrg.	# of Toggles	
color	98.92%	974355	98.26%	1235642	26.82%
des56	99.98%	387082	99.98%	507995	31.24%
fm	99.99%	2873740	98.62%	1656506	-42.36%
fpu	99.97%	30169143	98.91%	46773663	55.04%
quad	100.00%	87596	98.65%	92799	5.94%

When collecting the data, we used scan chains of length approximately 32 for all circuits, whether or not an on-chip decompressor was used. Toggling includes both toggling due to the pattern shifted into the chains as well as toggling due to the results shifted out of the chains in each case. In each case, only toggling of flip-flops in the chains was counted. Any toggling that would have been generated in the circuit's combinational logic is not included. Thus, the total toggling that would have occurred in the circuit overall due to the toggling of these flip-flops would have been even greater.

The toggling of flip-flops during scan-shift for each circuit is shown in Columns 3 and 5. Column 6 shows the percent increase in total toggling when EDT is used instead of our patterns (that are merged into LUTs and then use adjacent fill for remaining X's). Clearly, the amount of energy dissipated into the circuit due to toggling during scan shift is usually greater when EDT is used. In the case of the largest circuit, *fpu*, there is more than a 55% increase in the total toggling during scan shift of the test set when EDT is used instead of our method. Only in the case of *fm_receiver* is the toggling for EDT less than ours. This is true even though we didn't consider any additional EDT toggling due to the fact that the channel length is longer than the chain length and the decompressor itself will toggle internally. We also merged our patterns to minimize FPGA occupancy instead of toggling. In future work, we will explore options to reduce the toggling even more. This may be easier with our method because of its inherent flexibility and amenability to making tradeoffs.

VI. CONCLUSIONS

In this paper we have explored some of the advantages of using an existing FPGA as a tester in a 3D stack. We have proposed an FPGA-based tester design for the application of specific test patterns that is well-matched to the underlying structure of the FPGA fabric. Our analysis shows that the proposed technique uses only a very small fraction of FPGA resources, and the percent reduction in overall bit storage relative to simply storing the patterns in a memory actually increases as the total amount of test data increases. Furthermore, the proposed technique can take advantage of the high TSV bandwidth that is likely possible in 3D die stacks to transmit data to multiple chains in parallel. This allows us to implement adjacent fill and can often significantly reduce the amount of scan-shift toggling that is needed when compared to patterns generated in low power mode for embedded deterministic test. In general, most of these advantages should also carry over into the 2.5D space. Future work will investigate in detail the security advantages of using an FPGA in a 3D stack as an in-system tester.

REFERENCES

[1] R. Chaware, K. Nagarajan, and S. Ramalingam, "Assembly and reliability challenges in 3D integration of 28nm FPGA die on a large high density 65nm passive interposer," in *Electronic Components and Technology Conference (ECTC), 2012 IEEE 62nd*, 2012, pp. 279 –283.

[2] M. Agrawal and K. Chakrabarty, "Test-cost optimization and test-flow selection for 3D-stacked ICs," in *IEEE VLSI Test Symp (VTS), 2013*\pp. 1–6.

[3] E. Sperling, "Thinking Outside The Chip," *Semiconductor Engineering*, 14-Jan-2016.

[4] E. Sperling, "Is The 2.5D Supply Chain Ready?," *Semiconductor Engineering*, 28-Sep-2015.

[5] A. L. Crouch, J. C. Potter, A. Khoche, and J. Dworak, "FPGA-Based Embedded Tester with a P1687 Command, Control, and Observe-System," *Des. Test IEEE*, vol. 30, no. 5, pp. 6–14, Oct. 2013.

[6] I. Aleksejev, S. Devadze, A. Jutman, and K. Shibin, "Virtual reconfigurable scan-chains on FPGAs for optimized board test," *Test Symp. LATS 2015 16th Lat.-Am.*, pp. 1–6, Mar. 2015.

[7] S. Devadze, A. Jutman, I. Aleksejev, and R. Ubar, "Fast extended test access via JTAG and FPGAs," *International Test Conf..*, pp. 1–7, Nov. 2009.

[8] X. Wu, P. Falkenstern, K. Chakrabarty, and Y. Xie, "Scan-chain Design and Optimization for Three-dimensional Integrated Circuits," *J Emerg Technol Comput Syst*, vol. 5, no. 2, pp. 9:1–9:26, Jul. 2009.

[9] S. Deutsch, B. Keller, V. Chickermane, S. Mukherjee, N. Sood, S. K. Goel, J. Chen, A. Mehta, F. Lee, and E. J. Marinissen, "DfT architecture and ATPG for Interconnect tests of JEDEC Wide-I/O memory-on-logic die stacks," in *Test Conference (ITC), 2012 IEEE International*, 2012, pp. 1–10.

[10] C. Wang, J. Zhou, R. Weerasekera, B. Zhao, X. Liu, P. Royannez, and M. Je, "BIST Methodology, Architecture and Circuits for Pre-Bond TSV Testing in 3D Stacking IC Systems," *Circuits Syst. Regul. Pap. IEEE Trans. On*, vol. 62, no. 1, pp. 139–148, Jan. 2015.

[11] S. K. Roy, P. Ghosh, H. Rahaman, and C. Giri, "Session Based Core Test Scheduling for 3D SOCs," in *VLSI (ISVLSI), 2014 IEEE Computer Society Annual Symposium on*, 2014, pp. 196–201.

[12] Y. Fkih, P. Vivet, B. Rouzeyre, M.-L. Flottes, and G. Di Natale, "A JTAG based 3D DfT architecture using automatic die detection," in (*Ph.D. Research in Microelectronics and Electronics (PRIME), 2013 9th Conference on*, 2013, pp. 341–344.

[13] C. Claus, R. Ahmed, F. Altenried, and W. Stechele, "Towards rapid dynamic partial reconfiguration in video-based driver assistance systems," *Reconfigurable Comput. Archit. Tools Appl.*, pp. 55–67, 2010.

[14] J. Xie and D. Patterson, "Realizing 3D IC Integration with Face-to-Face Stacking," *Chip Scale Rev.*, vol. 17, pp. 16–19, 2013.

[15] P. Dorsey, "Xilinx stacked silicon interconnect technology delivers breakthrough FPGA capacity, bandwidth, and power efficiency," *Xilinx White Pap. Virtex-7 FPGAs*, pp. 1–10, 2010.

[16] A. Shah, "The first fruits of Intel's biggest buy ever coming this quarter," *Computerworld*, 14-Jan-2016.

[17] K. Nepal, X. Shen, J. Dworak, T. Manikas, and R. I. Bahar, "Built-in Self-Repair in a 3D die stack using programmable logic," in *2013 26th IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, (DFTS),* pp. 243–248, October *2013, New York.*

[18] K. Nepal, S. Alhelaly, J. Dworak, R. I. Bahar, T. Manikas, and P. Gui, "Repairing a 3-D Die-Stack Using Available Programmable Logic," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 34, no. 5, pp. 849–861, May 2015.

[19] J. Rajski, J. Tyszer, M. Kassab, and N. Mukherjee, "Embedded deterministic test," *Comput.-Aided Des. Integr. Circuits Syst. IEEE Trans. On*, vol. 23, no. 5, pp. 776–792, 2004.

[20] Xilinx, "7 Series FPGAs Configurable Logic Block User Guide (UG474 (v1.7))." Xilinx, 17-Nov-2014.

[21] L. Ciganda, F. Abate, P. Bernardi, M. Bruno and M. S. Reorda, "An enhanced FPGA-based low-cost tester platform exploiting effective test data compression for SoCs," *Intl. Symp. Design and Diagnostics of Electronic Circuits & Systems, (DDECS)*, Liberec, 2009, pp. 258-263.