Journal of Electronic Testing (2019) 35:887–900
https://doi.org/10.1007/s10836-019-05845-5

Repurposing FPGAs for Tester Design to Enhance Field-Testing in a 3D Stack

Yi Sun[1] · Fanchen Zhang[1] · Hui Jiang[1] · Kundan Nepal[2] · Jennifer Dworak[1] · Theodore Manikas[1] · R. Iris Bahar[3]

Received: 13 September 2019 / Accepted: 3 December 2019 / Published online: 30 December 2019

Abstract
We propose an architecture for a Field Programmable Gate Array (FPGA) based tester for a 3D stacked integrated circuit (IC). Due to the very short distances between dies in a stack that can make SerDes connections very efficient and the high density of through silicon vias (TSVs) that may be available, it is possible to connect the FPGA to the die under test through a very high bandwidth connection that can feed multiple short scan chains. We propose and evaluate two designs that exploit the underlying structure of the FPGA, allowing it to be used to efficiently store and apply predefined test patterns, reducing the FPGA resources required and the switching activity in the circuit under test when compared to a more traditional on-chip decompressor implemented to feed short scan chains. For the largest circuit we studied, the switching activity was reduced about 80% and the test time by 90%.

Keywords Design for testabiliy (DFT) · Low power test · On-chip decompressor

1 Introduction

Stacked integrated circuits (ICs) have the potential to implement the functionality of an entire board as a single packaged chip. The dies are stacked on top of each other and connected by through silicon vias (TSVs). The intellectual property of the dies may be from different companies and may include processors, memory, ASICs (application specific ICs), and even FPGAs (field programmable gate arrays) [6]. In addition to dramatically increasing the functionality contained within a very small form factor, 3D stacked ICs also have significant performance benefits arising from high-bandwidth, low delay, and low-power connections that TSVs and bumped connections can provide [1].

Similar advantages are also obtained in 2.5D multi-die scenarios, in which multiple dies are laid side-by-side on a silicon interposer or embedded multi-die interconnect bridge (EMIB) [21]. 2.5D ICs can provide a scalable interconnection for core-core, core-memory and core-accelerator devices [16]. 2.5D systems have already been 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 [20].

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 [7]. It is reasonable to expect that these advantages of FPGAs will likely carry over into the 3D IC space.

An FPGA can be placed in one layer or multiple layers in a 3D stack and has the potential to serve different purposes in a variety of applications. Intel has already created an embedded multi-die interconnect bridge (EMIB) used to connect its CPUs to Altera FPGAs to enhance

Responsible Editor: T. Xia

This work was supported in part by NSF grant CCF-1814928 and CCF-1812777.

✉ Yi Sun
yis@mail.smu.edu

1 Southern Methodist University, Dallas, TX, USA
2 University of St. Thomas, Saint Paul, MN, USA
3 Brown University, Providence, RI, USA

performance and handle power issues [21]. Altera and Amkor have proposed a face-to-face packaging approach consisting of a mother die (FPGA) and daughter die (ASIC) [24]. Xilinx currently produces FPGAs that contains multiple FPGAs and other dies sitting side-by-side on a silicon interposer, aiding in prototyping and emulating large processor systems [12, 25].

FPGAs have been used to aid in testing for many years. Boards may be partially tested (even when not all the chips and firmware are available yet), by adding more functionality to the load board at the factory or by connecting chips directly on the board [8]. 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 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, an important issue in 3D is how and when to test each die in the stack. Bandwidth to upper dies 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 delivering high-bandwidth test data. These include a test access port (TAP) and TAP controller on every die, a serial boundary wrapper on every die interface to conduct interconnect testing, and a parallel port. 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 as well as the power dissipated during test.

In our previous work [27], a tester design that was intended to take advantage of the underlying FPGA structure was introduced. Specifically, we considered the case where specific ATPG (automatic test pattern generation) patterns should be applied to the die under test and how those patterns could be efficiently stored in the lookup tables (LUTs) that form the programmable fabric of FPGAs. We explored both the FPGA resources required as well as the the amount of scan flip-flop toggling expended during scan shift. Power dissipation 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. Reducing the power consumption also increases the allowable thermal budgets in a stack, allowing more ICs to pass thermal tests, increasing the number of the chips that can be stacked together, and allowing the integration of more functionality in a single stacked IC [14].

While the work presented in [27] served as a good initial exploration of the proposed approach, multiple issues remained to be explored. In this article we expand the work of [27] in several ways to better demonstrate the benefits of our approach. More specifically, we make the following contributions:

- We explicitly extract the interior circuit toggling during shift to better estimate dynamic power and test time using our approach.
- We investigate the tradeoff between reducing power dissipation and more efficiently storing patterns in the FPGA's lookup tables by more efficiently dealing with don't cares. In particular, we consider adjacent fill merging (ADJCOM) and an "X"-retained merging algorithm (XRET) in our analysis.
- We explicitly investigate the use of multiple-input signature registers (MISRs) for capturing test responses and obtain data for MISR overhead along with the effect of aliasing on fault coverage.

The rest of this paper is organized as follows. Section 2 provides additional details on previous work in FPGA testers and 3D test. Section 3 introduces the implementation for our testers using the generic fabric of the FPGA. Section 4 introduces the adjacent fill merging algorithm (ADJCOM) to prioritize the reduction of scan shift toggling and accompanying reduction in shift power dissipation while Section 5 introduces the "X"-retained merging algorithm (XRET) to prioritize the more efficient use of FPGA resources. Section 6 analyzes MISR aliasing and test time reduction. Section 7 concludes the paper.

2 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) [2]. FPGAs have also been embedded into SoCs

(System on Chips) to provide system test capabilities [11]. 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. In the case of a 3D stacked IC, because the dies may come from different companies, dedicated embedded tester logic may be provided by each IP provider. In addition to providing a means of testing the stack, this approach may help protect the intellectual property (IP) among the different companies with IP in the stack. Furthermore, 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.

Various methods have been developed for testing 3D stacks. For example, [23] 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 [10] 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 [22], test scheduling [19], and the communication of test data between layers through the JTAG port [13]. 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).

3 Exploiting the FPGA's Generic 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.[1] 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.

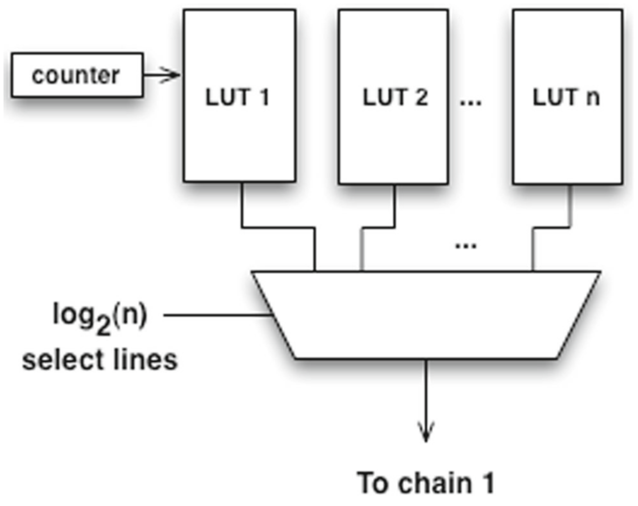

Fig. 1 Example FPGA-based implementation for storing pattern data for a single scan chain. This is repeated for multiple chains, with LUTs possibly shared among chains

To 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.

To save on FPGA resources, we can reduce the number of LUTs by merging the same pattern slice 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. Each scan chain would require one set of select lines for its MUX as shown in Fig. 1.

Note that to maximize the efficiency of mapping scan data to LUTs, ideally, the scan chain length will match the number of bits available in the LUT. For example, in our experiments, we mapped our tester design to an FPGA with

[1]In order to achieve high test coverage, the number of top off patterns could be quite significant. Exploring how to decrease the top-off patterns is left for future work.

5-input LUTs containing 32 bits. As a result, we used scan chains of length 32 for each of the circuits when collecting data for this paper.

Of course, the select line data are 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.

4 Using Adjacent Fill in Merging Algorithm to Reduce Power during Test (ADJCOM)

Different approaches may be taken to merge patterns into LUTs. In particular, how Xs are filled before and/or after a merge can influence the scan shift power and the LUT overhead. Because we are interested in reducing the power during scan shift, we first look at an approach which uses adjacent fill [5] to minimize switching activity.

4.1 Adjacent Fill Merging Algorithm

The LUT design process starts with a synthesized Verilog circuit netlist, which undergoes scan insertion with a scan chain size ideally equal to the size of the FPGA's lookup tables. (This will be 32 bits long in our later experiments). An ATPG pattern set for stuck-at faults is generated in such a way that any remaining X's in the patterns are not automatically filled by the tool but are kept in the pattern set. ATPG options (such as dynamic compaction) are used to create the initial compact test set, but on-chip decompressors are not.

Table 1 shows an example of a pattern set divided between three chains. These generated patterns are analyzed and any don't care "X" is filled with the value of an adjacent bit. Consider chain 1 pattern 1 (01**XX**1) shown in Table 1; we fill the X with 1 (the same value as the second bit) to decrease the switching activity yielding 01**11**1. This is called *adjacent fill*, and the remaining patterns are filled in a similar manner. These adjacently filled patterns are shown in Table 2. Our experiments show that the time required for adjacent fill grows linearly with the number of flip-flops in the circuit. After adjacent fill, these patterns no longer have any don't care bits left and can now be directly assigned to LUTs.

Table 1 Example original pattern set

	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

Table 2 Example pattern data after adjacent fill

	Chain 1	Chain 2	Chain 3
Pattern 1	01111	10000	11111
Pattern 2	11111	11111	11000
Pattern 3	00000	11001	11000
Pattern 4	11111	10111	11111

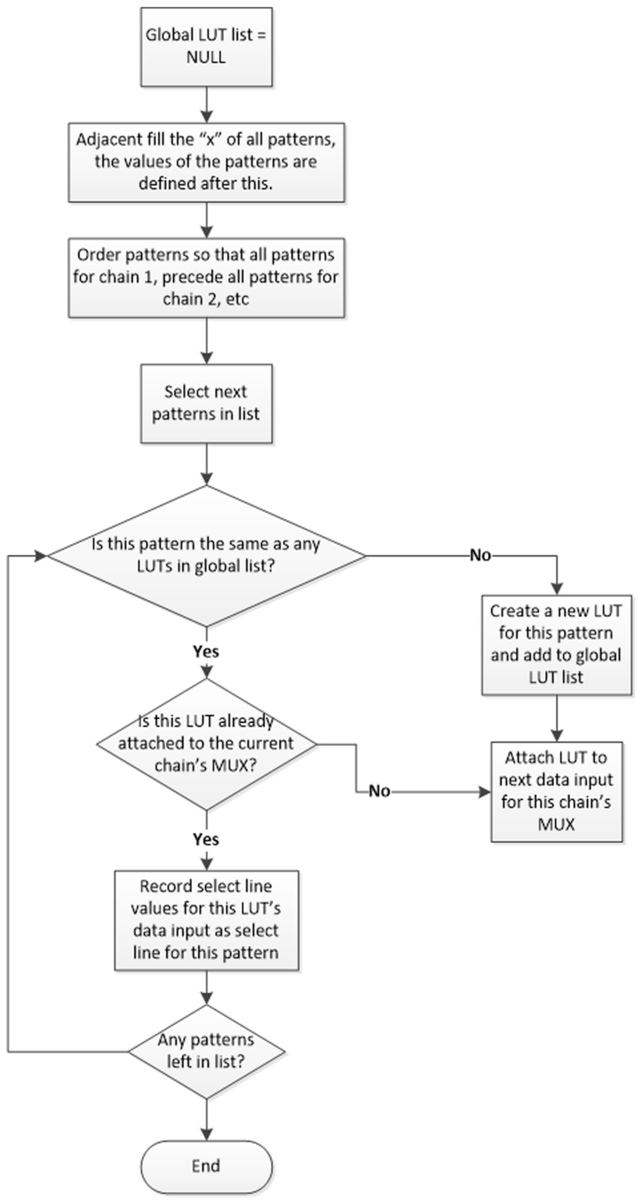
Global LUT list = NULL
Adjacent fill the "x" of all patterns, the values of the patterns are defined after this.
Order patterns so that all patterns for chain 1, precede all patterns for chain 2, etc
Select next patterns in list
Is this pattern the same as any LUTs in global list?
No
Create a new LUT for this pattern and add to global LUT list
Yes
Is this LUT already attached to the current chain's MUX?
No
Attach LUT to next data input for this chain's MUX
Yes
Record select line values for this LUT's data input as select line for this pattern
Any patterns left in list?
End

Fig. 2 Flowchart for LUT and Select Line Reduction using ADJCOM

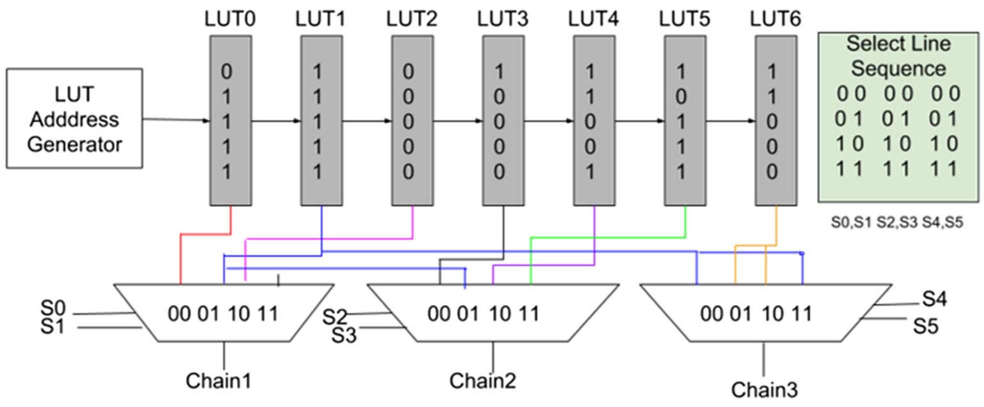

Fig. 3 Resulting implementation for patterns shown in Table 2 after pattern merging

However, direct assignment of these patterns to LUTs can be wasteful. There might be cases where after adjacent fill, there are identical patterns within the same chain or across different chains (e.g. pattern 11111 in Table 2). Instead of storing 11111 multiple times we compress the data that needs to be stored in the LUTs using our Adjacent-Fill and Compress (ADJCOM) algorithm illustrated in Fig 2.

For each chain, after we do adjacent fill, we determine what LUTs and mux connections will be needed. Specifically, after ordering the patterns in chain order, we start selecting patterns one-by-one. If the current pattern is identical to one already contained in a LUT, no new LUT needs to be allocated, and a new mux connection may be made, if required. If they are not the same, we create a new LUT, attach it to the LUT pool, and attach it to this chain's mux. In both cases, we record the appropriate select line index for this pattern so that the correct mux input (and LUT) will be selected for this pattern during test. We then check if there are any remaining patterns left. In summary, for each iteration, we need to keep track of the muxes where each LUT connects and also when that LUT should be selected (i.e., for which patterns) for each chain.

To help illustrate this ADJCOM compression methodology, consider the following example consisting of 3 chains, 4 patterns, and 5 bits per chain, with patterns shown in Table 2. To reduce the LUTs and select lines required, we must merge the patterns when possible, taking the following steps:

1. Because the LUT pool is empty, we push the first pattern of Chain 1 (01111) into the LUT pool. This LUT is added to the first data input of the mux for Chain 1, and the select line value for Pattern 1, Chain 1 is set to 0.
2. Pattern 2 of Chain 1 (11111). 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 the mux of Chain 1, and the select line value 1 for the pattern is recorded.

   Now *LUT pool: 01111, 11111. Chain 1 LUTs: 0,1; Chain 1 Select lines:0,1.*
3. Pattern 3 of Chain 1: 00000. 00000 cannot be merged with LUT0 (01111) or ADJCOM (11111). Add the pattern to the pool, attach the LUT to the third data input of the mux of Chain 1, and record the select line value.

   Now*LUT pool: 01111, 11111, 00000. Chain 1 LUTs: 0,1,2 Chain 1 Select lines:0,1,2.*
4. Pattern 4 of Chain 1: 11111. This pattern can be merged with ADJCOM (11111). Since ADJCOM exists in the LUT pool and is already attached to this chain's mux at data input 1, it does not need to be added to another data input. However, the select line value 1 must be recorded for this chain and pattern 4.

   Now *LUT pool: 01111, 11111, 00000. Chain 1 LUTs: 0,1,2; Chain 1 Select line values :0,1,2,1.*
5. Pattern 1 of Chain 2: 10000. This pattern cannot be merged with the existing LUT pool so we must create a new LUT. The new LUT is added to the next data input for the mux of Chain 2 mux, and the select line value 0 for the pattern is recorded.

   Now *LUT pool: 01111, 11111, 00000, 10000. Chain 1 LUTs: 0,1,2; Chain 1 Select line values :0,1,2,1. Chain 2 LUTs: 3; Chain 2 Select line values: 0.*

This process continues until we have attempted to merge all of the patterns. The final result is shown in Fig. 3. The merging process allows LUTs to be shared among 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 values, it is still compatible with patterns with fewer don't cares at the cost of less merging.[2] Eventually, if no X values are available, and if the number of repeated pattern sequences are small (as could happen with embedded deterministic test (on-chip decompressor) [18]),

[2] By uncompressed, we mean the patterns are generated without an on-chip decompressor, but still after the dynamic pattern compression from the ATPG tool.

Table 3 Characteristics of our opencores.org benchmark circuits

	PIs	POs	FFs	Faults	Patterns	Chains
Quad	36	25	184	7132	40	6
Color	297	34	858	36534	91	29
des56	132	67	193	16050	120	14
fm	10	12	521	23408	365	18
fpu	72	70	5493	276930	254	172

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 (the select line data) off-chip.

4.2 FPGA Implementation Results and Analysis

To evaluate the effectiveness of our algorithm, 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. When generating the test patterns for the circuits, both inputs and outputs of the circuits were registered.

An ATPG tool was used to insert multiple scan chains of length 32 (to match the size of the LUTs in our target FPGA) in each circuit. The scan chains contain only PIs, POs 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 as well. Details regarding each of the circuits studied are provided in Table 3. The table lists the number of Primary Inputs (PIs), Primary outputs (POs), and Flip flops (FFs) present in the original circuit as well as the number of test patterns generated with a target test coverage at 100% and the number of scan chains used.

Note that these 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.

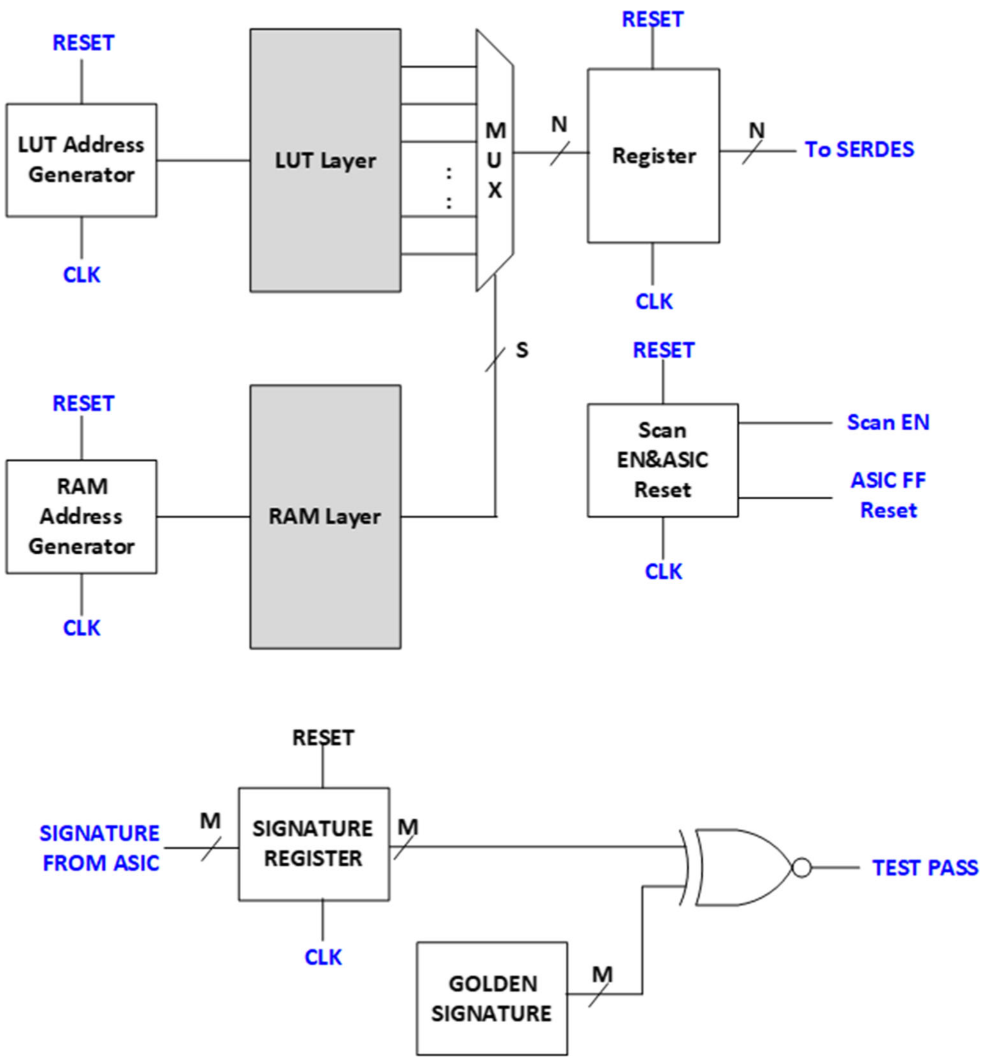

Fig. 4 FPGA-based tester block diagram

After a test is applied, the capture values of the flip-flops and primary outputs are shifted out to a MISR (multi-input signature register). In our experiments, we fill all the values in the PIs and scan chains with known values and there are no uninitialized memories or other sources of X's, so we can use the MISR as our result compactor to store the test response.

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 Vivado 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), shift register (1.444Kb) logic capabilities, and fast wide multiplexers (16:1 MUX using 4 LUTs or 1 slice) for efficient FPGA fabric utilization [26].

These features of the FPGA are important for efficient implementation of our controller. Figure 4 shows a basic architecture of the test controller that tries to harness existing FPGA resources. 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 register (MISR), and a scan enable signal generator. Our controller will have a fixed set 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 much 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). The test controller has three inputs (CLK, RESET and a scan signature from the ASIC), four outputs (a scan enable signal, a reset sent to the ASIC, a registered bus feeding scan data to the ASIC via a SerDes connection, and an output that indicates the test having passed or failed.)

As noted earlier, 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 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 Vivado 17.2 with a synthesis goal set to reduce the overall system area. Results of these two experiments are shown in Tables 4 and 5. Note that the area results include all the structures shown in Fig. 4 except the signature register, golden signature, and XNOR comparison logic. This signature logic is negligible compared to the rest of the FPGA-based tester block design.

Table 4 Experiment 1: All modules are distributed RAMs/Slice LUTs

Circuits	Max Freq (MHz)	Slice LUTs	% use LUTs
Quad	254.5	207	0.15%
Color	210.7	2107	1.5%
des56	233.1	797	0.6%
fm	180.4	2515	1.8%
fpu	164.3	11571	9.8%

Table 5 Experiment 2: Mux select lines implemented in block RAMs (BRAMs)

CKT	Max Freq (MHz)	Slice LUT	% use LUTs	Block RAMs	%use BRAMs
Quad	212.7	160	0.1%	1	0.3%
Color	220.9	974	0.7%	3	0.8%
des56	232.1	452	0.3%	1	0.3%
fm	221.5	1725	1.2%	3	0.8%
fpu	200.3	1687	1.1%	30	8.2%

Table 4 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 164.3 to 254.5 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 *quad* used negligible hardware resources and was the fastest while *fpu* used the most resources (9.8% of LUTs available) and could be run at just over 164.3 MHz. In experiment 1, the speed of the circuits goes up if less LUTs are used, because more LUTs means more time need to be used to program the LUTs. Because the more the LUTs, the more routing it takes to access to the corresponding LUTs.

Table 5 shows the results for Experiment 2, where we store the patterns and select lines in LUT and BRAMs respectively. This results in fewer LUTs compared to Experiment 1 because all the LUTs of Experiment 1 that were dedicated to storing the multiplexer select line values are no longer needed. The corresponding data are now stored in one or more of the 365 available block RAMs (BRAMs) instead. Keeping the select lines in BRAMs also helps to increase the tester speed of three of the circuits. However, the small size of the *quad* circuit prevented it from taking advantage of the BRAMs.

4.3 Data Reduction Using ADJCOM

Another issue we wanted to explore was how much data reduction we were able to achieve with our current

FPGA-based architecture. We took a preference in reducing the number of LUTs, storing only the compressed scan chain pieces in the LUTs. As a result, the number of bits in the LUTs should be much less than original data bits. If the number of select lines needed for each chain MUX was not too large, then even storing pattern bits and select line bits should be less.

The data obtained for our 5 circuits is shown in Table 6. (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:

$$32 \times \# \, of \;\; chains \times \# \, of \;\; patterns \qquad (1)$$

including padding, for chains of length 32 bits. 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 4.1 multiplied by 32 (LUT size). Column 4 adds the data for the select line values on each pattern and is equal to the number of select lines needed for all chain muxes multiplied by the number of patterns. Column 5 corresponds to the percent reduction in data required when Columns 3 and 4 are added together and compared with the original test data shown in Column 2. We see larger percentage reduction in the number of bits needed to store LUTs and select lines as the total original test data increases. Specifically, we see reduction of 51% for our largest circuit. This is encouraging. Column 6 compares Column 2 and 3 to determine the percent reduction in data storage needed if only the data in the LUTs is considered. This might be significant if we are worried about the occupancy of the FPGA but are obtaining the values on the select lines from an external memory.

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 four of the five circuits tested we see this reduction in test data to be over 75%.

Table 6 Data storage reduction

CKT	Original Total (bits)	LUT Data (bits)	Select Line (bits)	%↓ (LUT +sel)	%↓ (LUT only)	%↓ (sel only)
Quad	7680	6624	1260	−2.7%	14%	83%
Color	84448	67424	70034	−63%	20%	17%
des56	53760	25504	9960	34%	53%	81%
fm	210240	80480	50370	38%	62%	76%
fpu	1398016	370272	308610	51%	73%	76%

We thus see that 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.

4.4 Switching Activities

Although we were able to compress our data well enough 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.Even if the number of top-off patterns required is relatively large, as we showed in Table 6, there are multiple approaches to storing the test data depending on the size and available resources in the FPGA and off-chip memories that can help ameliorate the issue.

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, even with new materials proposed to enhance heat dissipation [15]. 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

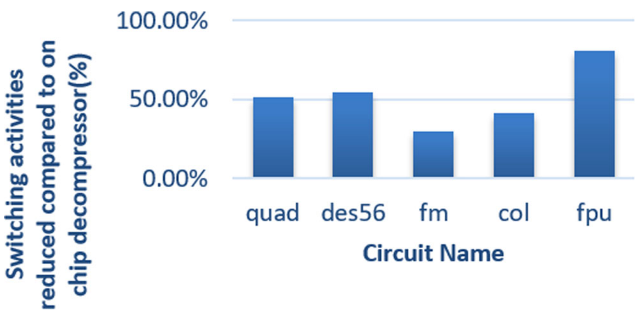

Fig. 5 Switching activities for our method vs. on-chip decompressor

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.

To investigate this possibility, we collected data regarding the difference in switching activity obtained both for patterns shifted in as the output of a power-limited on-chip decompressor as well as for our original scan patterns with adjacent fill implemented after merging.

For these experiments we used scan chains of length 32 bits for all the circuits whether generating patterns with or without an on-chip decompressor. To try to make the switching activity comparison as fair as possible, more than 200 test sets were created for each circuit with different low-power parameters when patterns were generated in the presence of the on-chip decompressor. The pattern set with the lowest toggling activity that did not lead to a significant reduction in test coverage was selected for comparison against our approach. We also used low-power options to generate patterns for our approach and allowed X's to remain in the test set for merging and adjacent fill.

To collect the switching activities, we apply our test set to each circuit, simulate the circuit, and extract total switching activity from every node in the circuit using VCD (value change dump) files. In each case, the switching activities of flip-flops in the chains as well as any switching activities that would have been generated in the circuit's combinational logic is also included during both shift and capture. The switching activity reduction achieved using our ADJCOM approach over the on-chip decompressor during scan test for each circuit is shown in Fig. 5. The switching activity reduction is computed using:

$$\%reduction = (1 - \frac{activity_{\text{ADJCOM}}}{activity_{\text{on-chip decompressor}}}) \times 100\% \quad (2)$$

We see that each circuit shows significant reduction in total switching activity, with our largest circuit *fpu* showing an 81% reduction. One possible reason for this is that, when an on-chip decompressor is used, the ATPG tool tends to create more patterns than the original pattern set.[3] Higher pattern count translates to more switching activity. Our approach requires fewer patterns. This, coupled with our use of adjacent fill, results in lower power consumption for ADJCOM. It is also encouraging that the associated tester architecture works better for the largest circuit with increasing amounts of test data: *fpu* (Fig. 6).

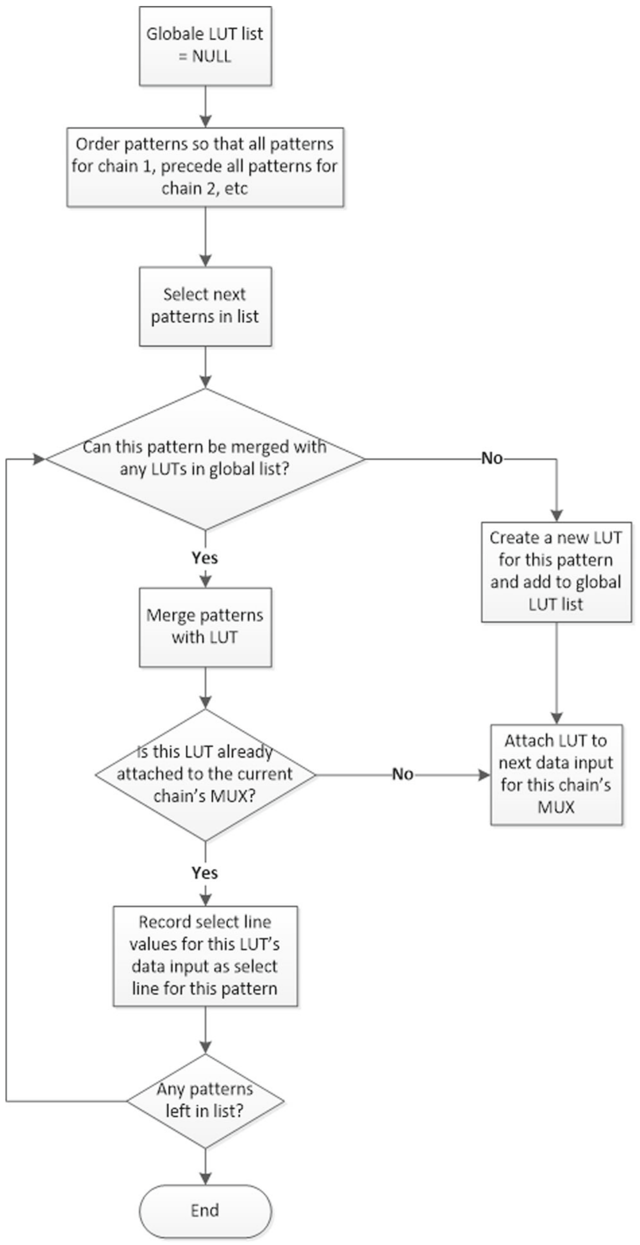
Globale LUT list = NULL
Order patterns so that all patterns for chain 1, precede all patterns for chain 2, etc
Select next patterns in list
Can this pattern be merged with any LUTs in global list?
No
Create a new LUT for this pattern and add to global LUT list
Yes
Merge patterns with LUT
Is this LUT already attached to the current chain's MUX?
No
Attach LUT to next data input for this chain's MUX
Yes
Record select line values for this LUT's data input as select line for this pattern
Any patterns left in list?
End

Fig. 6 Flowchart for LUT and Select Line Reduction using XRET

[3]Note that the length of the scan chain can have a bearing on the number of patterns produced from an on-chip decompressor [4] with larger chains leading to fewer patterns.

5 X Retained Merging Algorithm (XRET)

The data storage experiments of Section 4.3 showed that some circuits did not see as much of a benefit (e.g. circuit *color* showed a 63% increase in LUT and Select line data over the original case). In this section we propose a new algorithm called X retained merging algorithm (XRET) to efficiently deal with don't cares to improve the storage of LUTs data and select lines on an FPGA.

5.1 Merging with "X" in Pattern Set Retained

In this approach, the don't care bits in the original pattern set are retained in order to achieve the maximum pattern reduction. The way the patterns are merged is different, such that it significantly improves the results regarding the number of LUTs (and therefore the area overhead and data compaction). 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 (i.e., the 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 the muxes that each LUT connects to and when that LUT should be selected (i.e., for which patterns) for each chain.

To help illustrate this compression methodology, consider the same patterns used in Table 1. To reduce the LUTs and select lines required, we must merge the patterns when possible, taking the following steps:

- 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.

Table 7 Experiment 1—All modules are distributed RAMs/Slice LUTs

CKT	Max Freq (MHz)	Slice LUTs	% use LUTs
Quad	270.2	175	0.12%
Color	232.4	1182	0.8%
des56	240.3	578	0.4%
fm	168.2	2074	1.5%
fpu	167.6	2996	2.4%

- 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.*
- Pattern 3 of Chain 1: X0XX0. X0XX0 cannot be merged with LUT0 (01XX1) or ADJCOM(1XX11). Add the pattern to the pool, attach the LUT to the third data input of Chain 1's mux, and record the select line value. Now *LUT pool: 01XX1, 1XX11, X0XX0. Chain 1's LUTs: 0,1,2 Chain 1's Select lines:0,1,2.*
- 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.*
- Pattern 1 of Chain 2: 100X0. This pattern can be merged with XRET (X0XX0) to create 100X0. Replace XRET with this new merged pattern in the pool. Add XRET to

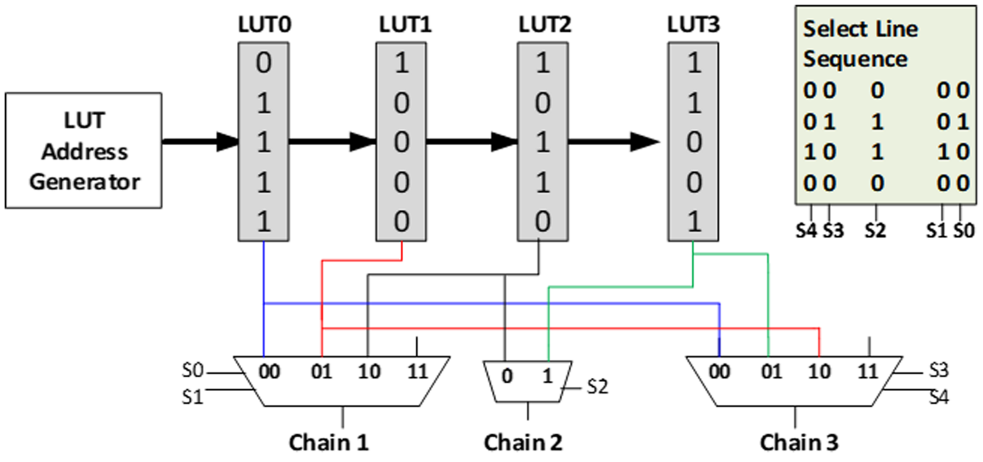

Fig. 7 Resulting implementation for patterns shown in Table 1 after pattern merging

Table 8 Experiment 2— Mux select lines implemented in block RAMs (BRAMs)

CKT	Max Freq (MHz)	Slice LUT	% use LUTs	Block RAMs	%use BRAMs
Quad	229.1	124	0.08%	1	0.3%
Color	251.4	847	0.6%	3	0.8%
des56	273.1	397	0.6%	1	0.3%
fm	247.8	796	0.6%	3	0.8%
fpu	223.6	1387	1.2%	30	8.2%

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 with 1s and 0s using the adjacent fill technique.

This gives us our final LUT pool: 01111, 10000, 10110, 11001. The resulting implementation is illustrated in Fig. 7. For this example, we see both a reduction in the LUT bits as well as the select line bits compared to the implementation after ADJCOM (Fig. 3).

Table 7 shows that this new merging algorithm takes up less area on the FPGA compared to the adjacent fill merging algorithm and that the tester can be operated at a clock frequency of 167.6 to 270.2 MHz for Experiment 1 where all modules were stored in distributed RAMs or slice LUTs. Again, the tester does not need to operate at the speed of a functional ASIC. The circuits used negligible hardware resources. Our largest circuit *fpu* showed the most compression in LUT use (down from 11571 in Table 4) to 2996). Other conclusions that were drawn from Table 4 can also be applied here.

We also see similar pattern as Table 5 when we run the XRET algorithm and store the Mux select lines in block RAMS instead of distributed RAMs. The results are shown in Table 8. We see across all five circuits a slight improvement in the max clock frequency and a slight increase in the slice LUT count.

Table 9 Data storage reduction

CKT	Original Total (bits)	LUT data (bits)	Select Line (bits)	%↓ (LUT +sel)	%↓ (LUT only)	%↓ (sel only)
Quad	7680	5600	1224	11%	27%	84%
Color	84448	37824	68951	−26%	55%	18%
des56	53760	18496	9480	48%	66%	82%
fm	210240	66368	49275	45%	68%	77%
fpu	1398016	44384	284988	76%	96.8%	79.6%

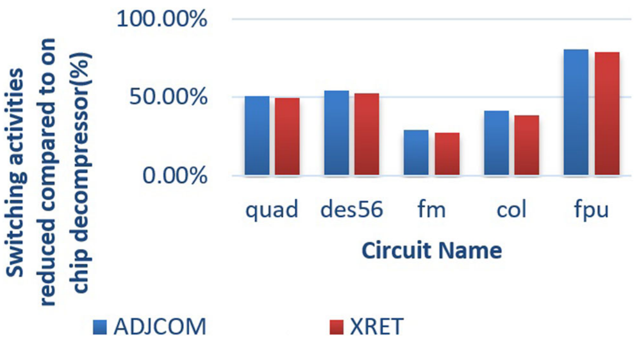

Fig. 8 Switching Activities for our methods vs. on-chip decompressor

5.2 Data Reduction Using XRET

Using this new XRET algorithm, we repeat the data storage experiments described earlier in Section 4.3 for the benchmark circuits. The data storage reduction results for the circuits is shown in Table 9.

Comparing the results seen in Table 9 with the ones obtained using ADJCOM (Table 6), we can see that XRET has a much higher compression rate when the "Xs" are retained during the merging for all of our circuits across all three storage scenarios (LUT+slect line bits, LUT only, and select lines only).

5.3 Switching Activities

The switching activities reduction during scan test for each circuit is shown in Fig. 8. The blue bars (ADJCOM) and red bars (XRET) show the percentage of power reduction compared to an on-chip decompressor. Both approaches achieve a good amount power reduction. We see that ADJCOM achieved a slightly higher power reduction compared to XRET. However, the XRET approach can still achieve almost the same amount of power reduction as ADJCOM and while achieving a much better compression rate. As a result, XRET is a very good option when the tester needs to achieve good compression rates while still prioritizing thermal issues during test. Furthermore, we consider XRET our best option unless power reduction is the main concern and there are plenty of FPGA resources available.

In order to obtain a more direct visualization of the power dissipation, we used Cadence Encounter Test to automatically generate the layout for all five circuits and mapped the switching to the location where the cell is located. Figure 9 shows an IC floorplan with total switching activity during test for *fpu*. Note that the two 2D subplots are divided into 100×100 squares, where each square could have one or even hundreds of cells. Red squares correspond to areas of high switching activity while purple/blue squares correspond to low switching activity areas. The IC floorplan

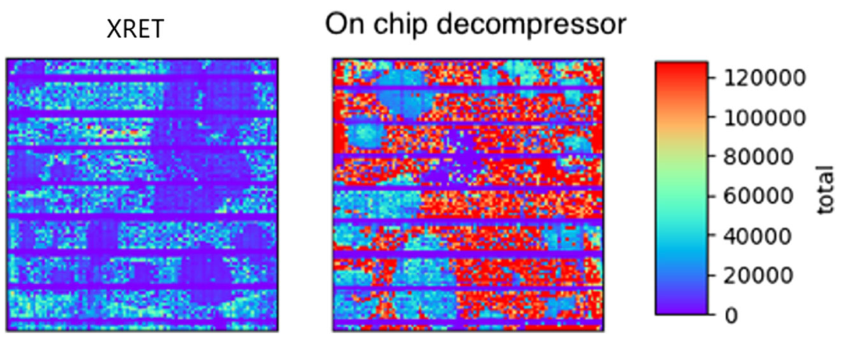

Fig. 9 Switching activities in IC floorplan for *fpu* using XRET versus an on-chip decompressor. The legend shows switching activity scale – red corresponds to high switching and purple/blue low switching activity

shows that the switching activity is not only low in the case of XRET but areas of red spots are almost non-existent compared to the on-chip decompressor case.

6 Test Response and Test Time

The test architecture shown in Fig. 4 contains the LUTs as well as the select lines for the muxes generated using either the ADJCOM or XRET algorithms. These are used to apply the compressed test patterns to the scan chains. The test response is then captured as a signature using a multiple-input signature register (MISR). The use of a MISR is common in DFT architectures to compress the test responses. Their primary downside arises when unknown values may be present in the test responses due to partial scan designs, memory elements, etc., that lead to unknown values. In our case, no unknowns will be present in any of our benchmark circuits; however, if they were present in a design, previously proposed approaches, such as X-compact [17] could be used to help prevent unknown values from propagating into and corrupting the signature.

Unfortunately, MISRs are also known to cause some loss of coverage due to aliasing. In this section, we investigate the use of a MISR and its effect on the test coverage for the benchmark circuits and patterns studied. We will show that the coverage loss is reasonable and is in line with normal coverage loss ($\leq 5\%$) due to a MISR [9].

For our experiments, one MISR per circuit is being used. The MISR length is dependent on the number of chains in the ciruit and is computed as:

$$\text{MISR Length} = \#\text{of chains} + 5. \tag{3}$$

For example, consider circuit *quad*. Table 3 earlier showed that it had 6 scan chains; the number of bits in the MISR is 11 for this circuit. Each MISR is created using taps for XNOR gates using a characteristic polynomial and tap points based on [3]. Table 10 reports the size of the MISR as well as the test coverage achieved before and after using the MISR. We see an average coverage reduction of only 1.4% across the five circuits with coverage reduction ranging from 0.16% for *fm* to 3.2% for *color*.

Our method also uses less test time for the benchmark circuits than using an on-chip decompressor for 32-bit scan chains. Test time is especially important in field testing because a device must be taken offline to perform the test. Because the FPGA programming to implement the tester can be done without the circuit-under-test being taken offline, we do not include the time required to program the FPGA in our analysis. Figure 10 shows the percentage of the test time used by the on-chip decompressor that is needed by our FPGA–based approach. Compared to an on-chip decompressor for the same scan chain length, we can reduce the test time by 38% to 90%.

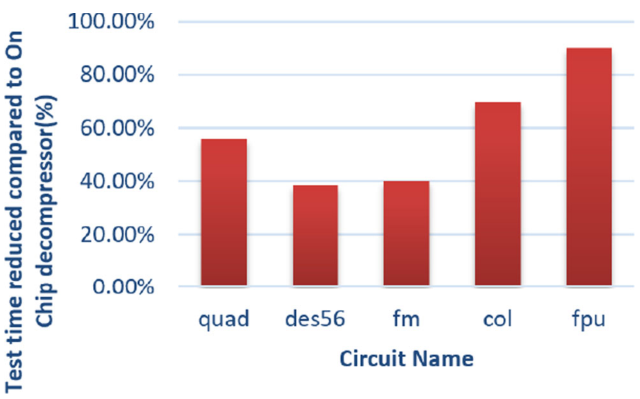

Fig. 10 Test time reduction:our Method vs. On-Chip Decompressor

Table 10 Test coverage before and after using MISR

Circuit	Test Coverage w/o MISR	Test Coverage with MISR	MISR Length	MISR XNOR Tap Points
Quad	99.57%	98.21%	11	9,11
Color	100.00%	96.80%	34	1,2,27,34
des56	99.98%	98.87%	19	1,2,6,19
fm	99.93%	99.77%	23	18,23
fpu	99.18%	97.82%	177	172,174,175,177

7 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 implemented two different merging algorithms (ADJCOM and XRET) for an FPGA-based tester design. The two methods require a very small fraction of FPGA resources for the circuits studied. We also see a reduction in the switching activity as well as test time when compared to on-chip decompressors for both methods.

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. In general, most of these advantages should also carry over into the 2.5D space.

In the future, we plan to more thoroughly explore how the proposed methods can be implemented in conjunction with other types of test time and power reduction approaches. For example, we could consider how the proposed approach can be combined with other previously proposed techniques to reduce shift and capture power during test. We could also explore the ability of the proposed method to select and possibly even generate new patterns to be applied for additional coverage after the use of standard LBIST test sessions.

References

1. Agrawal M, Chakrabarty K (2013) Test-cost optimization and test-flow selection for 3d-stacked ics. In: Proc IEEE 31st VLSI Test Symposium (VTS), pp 1–6. https://doi.org/10.1109/VTS.2013.6548941
2. Aleksejev I, Devadze S, Jutman A, Shibin K (2015) Virtual reconfigurable scan-chains on fpgas for optimized board test. In: Proc 16th Latin-american Test Symposium (LATS), pp 1–6. https://doi.org/10.1109/LATW.2015.7102411
3. Alfke P (1996) Efficient shift registers, LFSR counters, and long pseudorandom sequence generators. Tech. rep., Xilinx. https://www.xilinx.com/support/documentation/application_notes/xapp052.pdf/
4. Chakravadhanula K, Chickermane V, Cunningham P, Foutz B, Meehl D, Milano L, Papameletis C, Scott D, Wilcox S (2017) Advancing test compression to the physical dimension. In: Proc IEEE International Test Conference (ITC), pp 1–10. https://doi.org/10.1109/TEST.2017.8242035
5. Chandra A, Kapur R (2008) Bounded adjacent fill for low capture power scan testing. In: Proc 26th IEEE VLSI Test Symposium (VTS), pp 131–138. https://doi.org/10.1109/VTS.2008.47
6. Chaware R, Nagarajan K, Ramalingam S (2012) Assembly and reliability challenges in 3D integration of 28nm FPGA die on a large high density 65nm passive interposer. In: Proc IEEE 62nd Electronic Components and Technology Conference, pp 279–283. https://doi.org/10.1109/ECTC.2012.6248841
7. Claus C, Ahmed R, Altenried F, Stechele W (2010) Towards rapid dynamic partial reconfiguration in video-based driver assistance systems. In: Sirisuk P, Morgan F, El-Ghazawi T, Amano H (eds) Reconfigurable Computing: Architectures, Tools and Applications. Springer, Berlin, pp 55–67
8. Crouch AL, Potter JC, Khoche A, Dworak J (2013) Fpga-based embedded tester with a p1687 command, control, and observe-system. IEEE Des Test 30(5):6–14. https://doi.org/10.1109/MDAT.2013.2278531
9. Debany WH, Gorniak MJ, Daskiewich DE, Macera AR, Kwiat KA, Dussault HB (1992) Empirical bounds on fault coverage loss due to LFSR aliasing. In: Proc IEEE VLSI Test Symposium, pp 143–148. https://doi.org/10.1109/VTEST.1992.232739
10. Deutsch S, Keller B, Chickermane V, Mukherjee S, Sood N, Goel SK, Chen J, Mehta A, Lee F, Marinissen EJ (2012) Dft architecture and ATPG for interconnect tests of JEDEC wide-I/O memory-on-logic die stacks. In: Proc IEEE International Test Conference (ITC), pp 1–10. https://doi.org/10.1109/TEST.2012.6401569
11. Devadze S, Jutman A, Aleksejev I, Ubar R (2009) Fast extended test access via JTAG and FPGAs. In: Proc International Test Conference, pp 1–7. https://doi.org/10.1109/TEST.2009.5355668
12. Dorsey P (2010) White paper: Xilinx stacked silicon interconnect technology delivers breakthrough fpga capacity, bandwidth, and power efficiency. Tech. rep., Xilinx
13. Fkih Y, Vivet P, Rouzeyre B, Flottes M, Di Natale G (2013) A JTAG based 3D DfT architecture using automatic die detection. In: Proc 9th Conference on ph.d. Research in Microelectronics and Electronics (PRIME), pp 341–344. https://doi.org/10.1109/PRIME.2013.6603184
14. Lau JH, Yue TG (2009) Thermal management of 3D IC integration with TSV (through silicon via). In: Proc 59th Electronic Components and Technology Conference, pp 635–640. https://doi.org/10.1109/ECTC.2009.5074080
15. Loeblein M, Tsang SH, Han Y, Zhang X, Teo EHT (2016) Heat dissipation enhancement of 2.5D package with 3D graphene and 3D boron nitride networks as thermal interface material (TIM). In: Proc 2016 IEEE 66th Electronic Components and Technology Conference (ECTC), pp 707–713. https://doi.org/10.1109/ECTC.2016.85
16. Manoj PDS, Lin J, Zhu S, Yin Y, Liu X, Huang X, Song C, Zhang W, Yan M, Yu Z, Yu H (2017) A scalable network-on-chip microprocessor with 2.5d integrated memory and accelerator. IEEE Trans Circuits Syst I, Reg Papers 64(6):1432–1443. https://doi.org/10.1109/TCSI.2016.2647322
17. Mitra S, Kim KS (2002) X-compact: an efficient response compaction technique for test cost reduction. In: Proc International Test Conference (ITC), pp 311–320. https://doi.org/10.1109/TEST.2002.1041774
18. Rajski J, Tyszer J, Kassab M, Mukherjee N, Thompson R, Tsai K-h, Hertwig A, Tamarapalli N, Mrugalski G, Eide G, Qian J (2002) Embedded deterministic test for low cost manufacturing test. In: Proc International Test Conference (ITC), pp 301–310. https://doi.org/10.1109/TEST.2002.1041773
19. Roy SK, Ghosh P, Rahaman H, Giri C (2014) Session based core test scheduling for 3D SOCs. In: Proc IEEE Computer Society Annual Symposium on VLSI, pp 196–201. https://doi.org/10.1109/ISVLSI.2014.61
20. Sperling E Is the 2.5D supply chain ready? Semiconductor Engineering https://semiengineering.com/is-the-stacked-die-supply-chain-ready/ Accessed: 2019-11-26
21. Sperling E Thinking outside the chip. Semiconductor Engineering https://semiengineering.com/thinking-outside-the-chip/ Accessed: 2019-11-26
22. Wang C, Zhou J, Weerasekera R, Zhao B, Liu X, Royannez P, Je M (2015) Bist methodology, architecture and circuits for pre-bond tsv testing in 3d stacking ic systems. IEEE Trans Circuits Syst I, Reg Papers 62(1):139–148. https://doi.org/10.1109/TCSI.2014.2354752

23. Wu X, Falkenstern P, Chakrabarty K, Xie Y (2009) Scan-chain design and optimization for three-dimensional integrated circuits. J Emerg Technol Comput Syst 5(2):9:1–9:26. https://doi.org/10.1145/1543438.1543442
24. Xie J, Patterson D (2013) Realizing 3D IC integration with face-to-face stacking. Chip Scale Review 17(3):16–19
25. Xilinx: 3D ICs. https://www.xilinx.com/products/silicon-devices/3dic.html. Accessed: 2019-07-27
26. Xilinx: UG 474: 7 series FPGAs configurable logic block user guide. https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf. Accessed: 2019-06-17
27. Zhang F, Sun Y, Shen X, Nepal K, Dworak J, Manikas T, Gui P, Bahar RI, Crouch A, Potter J (2016) Using existing reconfigurable logic in 3D die stacks for test. In: Proc IEEE 25th North Atlantic Test Workshop (NATW), pp 46–52. https://doi.org/10.1109/NATW.2016.15

Yi Sun received the B.S. degree in electrical engineering from Chongqing University, Chongqing, China, the M.S.E.E. degree from Southern Methodist University, Dallas, TX, USA, 2013 and 2016 respectively. She is currently working as a Research Assistant in her PhD program in Southern Methodist University.Her current research interests include manufacturing and in-field testing and hard-ware security.

Fanchen Zhang received his B.S. degree in Telecommunication Engineering from Harbin Institute of Technology in Harbin, Heilongjiang province, China, M.S. degree in electrical engineering, and PhD degree in computer engineering from Southern Methodist University in Dallas, TX, USA in 2001, 2012 and 2018. He is currently a Hardware Engineer at Cisco Systems, Inc. in San Jose, CA, USA. His work includes semiconductor Design-for-Test (DFT), pre-silicon verification tests, post-silicon validation and debug.

Hui Jiang received her B.S. degree in electrical engineering from Shandong University, Weihai, Shandong, China in 2014 and M.S. degree in computer engineering from Southern Methodist University, in Dallas, TX, USA in 2016. Now, she is in her third year pursuing her Ph.D. degree in computer engineering in Southern Methodist University. Her current research area is Design for Test and hardware security.

Kundan Nepal received the B.S. degree in electrical engineering from Trinity College, Hartford, CT, USA, the M.S.E.E. degree from the University of Southern California, Los Angeles, CA, USA, and the Ph.D. degree in electrical and computer engineering from Brown University, Providence, RI, USA, in 2002, 2003, and 2007, respectively. He is currently an Associate Professor of Electrical & Computer Engineering at the University of St. Thomas, St Paul, MN, USA. His current research interests include embedded systems, defect/fault tolerant circuits and systems, nanometer digital VLSI system design, and reconfigurable computing.

Jennifer Dworak received her B.S., M.S., and PhD degrees in electrical engineering from Texas A&M University, College Station, TX, USA in 1998, 2000, and 2004 respectively. She is currently an Associate Professor of Electrical & Computer Engineering at Southern Methodist University in Dallas, TX, USA. Her current research interests include manufacturing and in-field testing, reliable systems, and hardware security.

Theodore Manikas received the B.S. degree in electrical engineering from Michigan State University, the M.S. degree in electrical engineering from Washington University (St. Louis) and the Ph.D. degree in electrical engineering from the University of Pittsburgh. He has been with Southern Methodist University since 2009 and is a Clinical Professor in the Department of Computer Science. His current research interests include system security and testing. He is a Licensed Professional Engineer in Texas and Oklahoma.

R. Iris Bahar received the B.S. and M.S. degrees in computer engineering from the University of Illinois at Urbana-Champaign, Urbana, IL, USA, and the Ph.D. degree in electrical and computer engineering from the University of Colorado, Boulder, Boulder, CO, USA. In between her M.S. and Ph.D. studies, she spent 5 years with Digital Equipment Corporation, Hudson, MA, USA, researching on microprocessor hardware design. Since 1996, she has been with the School of Engineering, Brown University, Providence, RI, USA, where she is currently a Professor of Engineering and Computer Science. Her research interests include computer architecture, computer-aided design for synthesis, verification, and low-power applications, and design, test, and reliability issues for nanoscale systems.