Repairing a 3-D Die-Stack Using Available Programmable Logic

Kundan Nepal, *Senior Member, IEEE*, Soha Alhelaly, Jennifer Dworak, *Member, IEEE*, R. Iris Bahar, *Senior Member, IEEE*, Theodore Manikas, *Senior Member, IEEE*, and Ping Gui, *Senior Member, IEEE*

***Abstract*—3-D die-stacks hold great promise for increasing system performance, but difficulties in testing dies and assembling a 3-D stack are leading to yield issues and slowing the large scale manufacturing of these devices. In many cases, a single defective die will kill the entire stack. To help mitigate this issue, we explore the possibility of repairing a stack that contains a defective die by utilizing an field programmable gate array (FPGA) that has already been included in the stack for other purposes, such as performance enhancement. Specifically, we propose bypassing the defective portion of a nonprogrammable die by replacing the defective functionality with functionality on the FPGA. In this paper, we discuss what additional logic must be added to an Application-Specific Integrated Circuit (ASIC) die to allow such a bypass to occur. We then show through detailed simulation of a 2.5-D Xilinx FPGA how bypassing of logic can be achieved and throughput maintained even when the two different dies involved operate at different frequencies. Finally, we explore the performance of this technique in a superscalar, out-of-order processor, where different functional units are marked for replacement. Our simulation results show that not only can we salvage a device that would otherwise have to be discarded, but creating multiple copies of the defective partition in the FPGA can allow us to regain performance even when the latency of the units in the FPGA is longer than that of the original defective copy.**

***Index Terms*—3-D, built-in-self-repair (BISR), defects, fault tolerance, field programmable gate array (FPGA), programmable logic.**

I. Introduction

As CIRCUITS approach the limits of Moore's law, and as power considerations have placed a limit on increases in clock frequency, stacking bare dies to form a 3-D die-stack has been proposed as one method that will allow significant increases in system performance to continue. Performance gains are expected to arise primarily from the fact that the routes between dies in a stack are much shorter than routes from chip to chip across a board or routes from one end of a chip to another. Unfortunately, high volume manufacturing has proven difficult. One of the main problems preventing the large-scale manufacturing of 3-D integrated circuits is the difficulty in testing dies and obtaining high yields [1]. The insertion of through-silicon vias (TSVs) may damage the die during the "drill and fill" process or when the silicon is ground away to expose the TSV so that it can be "micro-bumped." The TSVs themselves are difficult to probe without damaging them—making testing of individual dies difficult as well [2]. Furthermore, dies in the stack may warp due to mechanical forces during assembly or during normal operation, thereby exposing the chip to an additional source of potential errors [2].

Operating conditions can also cause errors. Hot spots in the stack are easier to create and harder to dissipate than in 2-D, and they may affect several dies. Such hot spots can cause either temporary incorrect operation or permanent damage. If we add these issues to standard problems that arise from imperfect manufacturing test, including test escapes, and latent defects, it becomes apparent that there is great potential for a defective die to find its way into the stack—either during assembly or later in the field due to wearout.

If a chip on a board is found to be defective, it is generally possible to de-solder the chip from the board and replace it with a working chip. Unfortunately, this strategy will not work when a defective chip is present in a stack because it is impossible to remove a chip from the stack once it has been assembled. Instead, the entire stack is often "killed" by a single defective chip. The greater the number of chips that are present in the stack, the more likely this is to occur, and the greater the financial cost of throwing away the stack. As a result, the ability to repair the stack could provide a significant advantage for increasing yields and salvaging a stack—even if there is some performance degradation.

Fortunately, a 3-D stack also provides new opportunities for repair. Specifically, if a die containing programmable logic is included in the stack, it may be harnessed to bypass defective components of other dies. This has the potential to be much more powerful than simply using an field programmable gate array (FPGA) on a board for repair. On a board, connections to an FPGA may be limited to a maximum of 2000 pins, and routes across the board are long. In a 3-D stack, routing between layers is short and 10 000 TSVs may be present in a square millimeter.

In this paper, we continue our work begun in [3] on using an FPGA in the stack as a resource for replacing

Manuscript received September 12, 2014; revised December 11, 2014; accepted January 21, 2015. Date of publication February 4, 2015; date of current version April 17, 2015. This work was supported by the National Science Foundation under Grant CCF-1061164. This paper was recommended by Associate Editor C.-W. Wu.

K. Nepal is with the University of St. Thomas, Saint Paul, MN 66105 USA (e-mail: kundan.nepal@gmail.com).

S. Alhelaly, J. Dworak, T. Manikas, and P. Gui are with the Southern Methodist University, Dallas, TX 75206 USA.

R. I. Bahar is with the Brown University, Providence, RI 02912 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2015.2399441

defective functionality in the stack with working functionality. Preferentially, the FPGA will already be included in the stack for an alternative purpose, such as performance acceleration, and harnessed for repair only when necessary. Many levels of granularity for repair are possible—from replacing the functionality of an entire die to replacing a single pipeline stage or functional unit. Repair is particularly well-matched to the repair of functional units in out-of-order processors because such processors are already designed to naturally handle multiple functional units with different latencies. In some cases, repair is mandatory when the only copy of a critical component is found to be defective. However, even when the lack of a defective component only causes performance degradation, replacement of the defective functionality may still be desirable.

In this paper, we describe the following investigations.

1) We design and analyze the circuitry needed to communicate between dies for the repair of a pipeline stage in a multiplier. This circuitry is mapped to two separate dies operating at different frequencies in a Xilinx 2.5-D Virtex-7 FPGA using the Xilinx Vivado software suite, and the timing and synchronization of the resulting designs are verified through simulation.
2) We show that when different dies in a Xilinx 2.5-D FPGA are run at different frequencies, it is possible to use multiple copies of the pipeline stage under repair to maintain the throughput of the pipeline when the pipeline does not contain feedback. In particular, we show that it is possible to maintain the multiplier's pipeline throughput when the FPGA die used for repair is operated at half the frequency of the "original" when two copies of the faulty pipeline stage are used.
3) We show how the timing overhead of our scheme on the original application-specified integrated circuit (ASIC) can be minimized by placing the multiplexers needed to return data from the FPGA in the scan path instead of the original functional path for full-scan designs implemented with Multiplexer (MUX)-D flip-flops.
4) We investigate the area overhead of our repair scheme on the ASIC layer by using commercial tools from Synopsys and Cadence to perform layout and TSV insertion for both a repairable and nonrepairable version of the multiplier design.
5) We describe the impact of replacing defective functional units in a 4-wide superscalar processor. We consider both the case where the only copy of a functional unit present in the original machine becomes defective and must be replaced as well as the case where one of multiple copies becomes defective, reducing performance if it is not repaired. We will show that even when latencies of the repair units are longer, multiple copies can be used to reduce or eliminate the performance overhead of repair.

The rest of this paper is organized as follows. Section II describes previous work in built-in-self-repair (BISR) and provides information on 3-D die-stacks. Section III gives a conceptual overview of our repair methodology. Section IV describes our investigation of TSV timing and design synchronization using simulation of a Xilinx 2.5-D Virtex-7 FPGA. Section V extends the analysis of the pipelined multiplier to the ASIC layer to determine the area overhead of the additional logic and TSVs that are needed for repair. Section VI uses architectural simulation to explore how the proposed approach will map to the repair of functional units in an out-of-order processor. Finally, Section VIII describes the conclusion.

II. Background

A. Related Work

A wealth of work has previously been done in fault tolerance and repair of circuits and systems. For example, BISR is a fault tolerance technique against permanent faults, where in addition to operational components, a set of hard-coded, permanent spare components is provided. If a faulty component is detected, it is replaced with a spare component [4]. Other fault tolerance approaches try to detect and correct errors in real-time, using fully-replicated shadow pipelines [5], simple in-order pipelines that check the results of a complex out-of-order pipeline [6] or specialized checkers [7]. A common application of BISR is in 2-D RAMs, where spare memory units [8] or redundant row/columns [9] are implemented. This approach has been expanded to 3-D stacked memories, where spare resources are borrowed from adjacent dies [10], [11], and spare TSVs are used to replace defective TSVs [12], [13]. When not enough spares are available, configurable fault-tolerant serial links have been proposed for TSV repair [14].

BISR with reconfigurable logic has also been applied to functional modules implemented in 2-D circuits. One approach has been to use reconfigurable modules in FPGAs, such as logic blocks or routing resources, to replace the defective modules [15]. Another approach is to use spare functional units on FPGAs [16], such as spare arithmetic logic units (ALUs) [17]. These approaches implement both the original and spare modules on a single FPGA. FPGAs and ASIC hardware may also be implemented on the same die in an SoC, to provide capabilities for modifying the design later when design errors are present or specifications (such as communication standards) change [18].

This paper extends the concept of BISR to the digital logic in 3-D stacks. We utilize two separate dies in the 3-D stack: the original circuit implemented in an ASIC process and a separate FPGA die that can be programmed when needed to create spare functional modules. This approach harnesses the advantages provided by 3-D architectures—including potentially large numbers of TSV connections and short distances between dies—to increase the flexibility and improve the performance of repair. To the best of our knowledge, we are the first to propose such an approach.

B. What Kinds of Dies are Present in 3-D Stack?

A 3-D die stack may take one of several forms depending upon the functionality of the stack and how many companies are involved in designing the different dies. At one extreme, a homogeneous stack may contain identical copies of the same die stacked on top of each other. One example would be a memory cube, in which almost the entire stack consists of identical dynamic random-access memory (DRAM) chips placed above a logic base [19]. At the other extreme,

a stack may one day be composed of different dies manufactured by different companies for a "competitive socket" approach [1], [20]. In this case, the stack could consist of different types of dies—processors, memories, DSPs, ASICs, FPGAs, and analog dies [21], [22]. In addition, silicon interposers may be placed between dies to allow appropriate TSV connections. In this approach, a 3-D stack serves as a board compressed into a single package. In between these two extremes, a single company may design most or all of the component dies in the stack. This provides significant flexibility and allows the designers to optimally layout a single design across multiple vertical layers to improve performance or reduce problems such as heat transfer. However, even here, some portions of the stack could be obtained from other companies. This paper focuses primarily on stacks that correspond to one of the last two cases.

C. FPGAs in 3-D Stack

Advances in FPGA design and manufacturing have made FPGAs an increasingly viable alternative to ASICs for many designs in 2-D. Unless the product will have very high volume or require high performance or area resources, an FPGA often provides the required functionality at a much more acceptable cost/die. Furthermore, because an FPGAs programming can be changed, it is especially useful for telecommunications or other applications where standards may change over a system's lifetime. Finally, 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 from it [23]. These advantages of FPGAs are likely to carry over into the 3-D IC space, and thus, there are many reasons other than repair that an FPGA may be present in the stack. In recent years, Altera and Ankor have proposed a face-to-face packaging approach dubbed the POSSUM package configuration consisting of a mother (FPGA die) and daughter die (ASIC) [24].

FPGA companies are also already manufacturing FPGAs containing TSVs. For example, Xilinx is currently selling a 2.5-D version of the Virtex-7 FPGA that contains four FPGA dies sitting side-by-side on a silicon interposer [25]. As noted by Xilinx, this makes the resulting hardware ideal for prototyping and emulating large processor systems. For example, one problem with prototyping large systems with multiple FPGAs is the need to partition the design so that relatively few connections are needed between the partitions due to limited pin count. This is much less of an issue in 3-D. Furthermore, the delay and drive strength needed to drive TSVs is much less than that needed to drive the I/O buffers on a traditional FPGA. The distances between layers of a 3-D stack are also much shorter than the routes on a board.

D. Clock Synchronization in 3-D Stack

As mentioned earlier, an advantage of using 3-D stacks over standard 2-D integrated circuits is that the TSVs that connect the stacks have much smaller delay than traditional bonding wires. For reliable operation, the clock distribution on the stacked dies often must operate synchronously, with minimal skew across the die stack [26]. However, the propagation delays through different TSVs can often vary due to process and/or temperature effects, which will affect clock synchronization [27].

Clock synchronization and clock deskewing across multiple dies in a 3-D stacked IC is an active area of research [26]–[29]. A common approach to address the clock synchronization problem across multiple dies in a 3-D stack is to use delay-locked loops (DLLs) to drive the TSVs to help ensure that the local clock networks on each die are synchronized. Chuang *et al*. [28] proposed a dual-delay-locked loop (D-DLL) for die-to-die clock deskew applications, allowing for a clock synchronization within 2 ps for 550 MHz–1.5 GHz clock signals. Ke *et al*. [27] proposed an approach based on dual-locking DLL that allows for clock synchronization without the need to replicate TSV delays. Once the DLL is locked, their approach continues to fine-tune the two DLLs in an interleaved manner to maintain the phase alignment. The clock synchronization circuitry has a very small area footprint (0.0044 mm^2 and consumes only 1.8 mW of power. Recently, an all-digital delay-locked loop (ADDLL) for 3-D-IC die-to-die clock synchronization with TSVs was presented in [29]. The authors proposed ADDLLs with two high resolution delay lines with digital controlled varactors to compensate for the delay variations of two TSVs. Two digitally controlled delay lines can then be used to eliminate clock skew between the clock signals across two dies.

Recently, researchers at Samsung electronics [30] have proposed an on-chip self-calibration scheme to remove signal conflict and reduce signal skews among stacked dies in a 3-D DRAM with TSVs. The calibration method finds the slowest die and compensates the data output time mismatch among the stacked dies. Using four stacked 16 G DDR4 DRAM designed in 25 nm CMOS technology and operational at a supply voltage of 1.2 V, the authors were able to measure and compensate mismatch allowing the memory to operate at speeds of over 2400 Mb/s.

In the case of reconfigurable architectures, Xilinx has also developed very effective techniques that minimize or eliminate skew across dies in their 2.5-D FPGAs [31], [32]. The Virtex 7 family, consists of a mixed-mode clock manager module and PLLs that are used to compensate for the clock network delays in the stack.

In this paper, we assume that a de-skewing approach such as those described above is available and focus our efforts on the architecture and tool flow for repair.

III. Methodology

To enable repair of an ASIC or processor using an FPGA in a 3-D stack, advance planning is needed to ensure that the required support structure is available for the desired repair options. Which portions of the stack may be repaired (and at what level of granularity) should be decided *a priori*.

A. Granularity of Replacement

Repair of a defective 3-D stack may occur at different levels of granularity. At the highest level, the functionality of an entire die could theoretically be placed into an FPGA "as is."

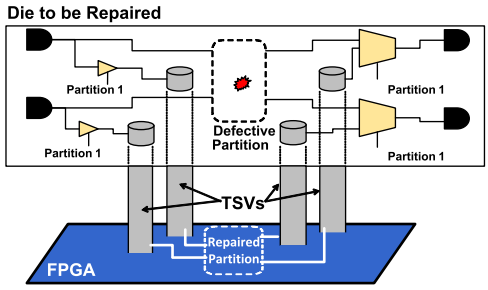

Fig. 1. Repair of a partition on the ASIC using the FPGA layer.

However, in many cases this may not be possible, especially when the total functionality of the defective die will not fit into the space available on the FPGA. Alternatively, only part of a die's functionality may be replaced. Depending on the underlying structure of the defective die, replacement could occur on the level of a pipeline stage, a functional unit, or any other module or submodule.

B. Basic Architecture of Replacement

Each partition that may be bypassed and replaced by FPGA functionality must have appropriate connections to TSVs that are ultimately connected to the FPGA. Note that such TSVs do not need to be used solely for repair. For example, when the repairable partition corresponds to a functional unit in an out-of-order processor, the same TSVs could be multiplexed so that they can also be used for performance acceleration. Specifically, the TSVs can be used to allow the implementation of new custom instructions on the FPGA. Effectively, this would allow a hardware implementation of functionality that ordinarily would take multiple instructions if it were implemented with the original instruction set.

An example of our repair architecture is shown in Fig. 1. Each of the inputs to the partition fans out not only to the bypassable partition, but also to a driving buffer or series of buffers that is capable of driving the TSV. Note that in this figure a single buffer is used and is tri-stated. The driving buffers should be sized so as to minimize the load and delay seen by the circuit. Each of these TSVs is also an input to the FPGA. The FPGA itself will need to be programmed to realize the functionality of the partition using those inputs. The outputs of the FPGA-implemented module will then travel through other predefined TSVs until they reach the level of the ASIC being repaired. Multiplexers select the values sent by the FPGA if the circuit is in repair mode, as indicated by the value of the select line on the MUX.

C. Reducing the Impact of Inserted Multiplexers on Circuit Delay

The extra delay inserted by the multiplexers during normal operation (i.e., when repair is not needed) may be minimized by placing the additional MUXs in the scan path usually used for test instead of the normal functional path, as shown in Fig. 2(b). Fig. 2(a) shows a schematic of a normal MUX-D flip-flop that is the standard flip-flop used in full-scan designs. Specifically, to enhance the testability of circuits by making the flip-flops in the design easily controllable and observable during test, each flip-flop in the design is concatenated into one or more shift registers, called scan chains, that allow values to be shifted directly into and out of the flip-flops during test.

The select signal for the MUX in a MUX-D flip-flop determines whether the scan chain is in shift mode or capture mode. For example, for the MUX-D flip-flop in Fig. 2(a), when the select line is equal to 0, the MUX will capture the value generated by the circuit's functional logic. Thus, the select line should be equal to 0 during normal functional operation and when results are being captured by the circuit's logic during test. In contrast, when the select line is equal to a logic 1, the data from the "scan in" input is clocked into the flip-flop. This allows the flip-flop to take on the value of the previous flip-flop in the chain. Because scan chains are so effective at making test effective and efficient, the implementation in Fig. 2(a) is standard in most of today's digital designs.

Fig. 2(b) shows the MUX that provides data from the FPGA in the scan path of the design. No additional circuitry is added to the normal functional path. Instead, the additional circuitry is added to the scan path. Because scan shift generally occurs at a very slow speed (10–40 MHz is not uncommon), this additional logic in the scan shift path should not hinder scan shift and may even help to prevent hold-time violations. The appropriate values on the select lines of both MUXes simply need to be set correctly when the circuit is in repair mode.

D. CAD Design Flow

In this paper, we choose to motivate our programmable repair approach using both a pipelined multiplier and a super-scalar processor as examples. However, the approach is generally applicable to other types of circuits at various user-defined levels of granularity. The basic flow of our approach is shown in Fig. 3 and includes the following steps.

1) Provide a Verilog netlist of an ASIC die to the synthesis software. For this paper, we used Synopsys design compiler.
2) Identify potential partitions for repair. As mentioned earlier in Section III-A, the repair of the defective die can occur at any user-defined level of granularity. For the multiplier example, a partition will correspond to the combinational logic in a pipeline stage; for the superscalar processor, the partitions will correspond to functional units in the execute stage of the processor. In general, a large circuit could be partitioned into smaller sub-circuits using a variety of tools and techniques. For example, a state-of-the-art min-cut partitioning tool such as hMETIS [33] could be used.
3) Once the circuit partitions have been identified, the ASIC flow continues with the placement of the synthesized cells, and the insertion and placement of the TSVs, drivers and MUXes needed to transfer data between the ASIC level and the FPGA during repair. Cadence EDI was used for the designs presented in this paper.

While the ASIC design flow completes, the FPGA design flow will start. The purpose of the FPGA flow is to create a variety of bitstream files that can be used to implement the

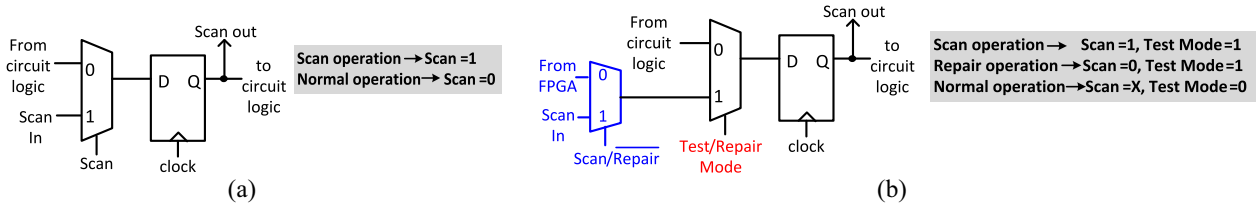

Fig. 2. (a) Standard MUX-D flip-flop (b) MUX placement in standard scan path for repair.

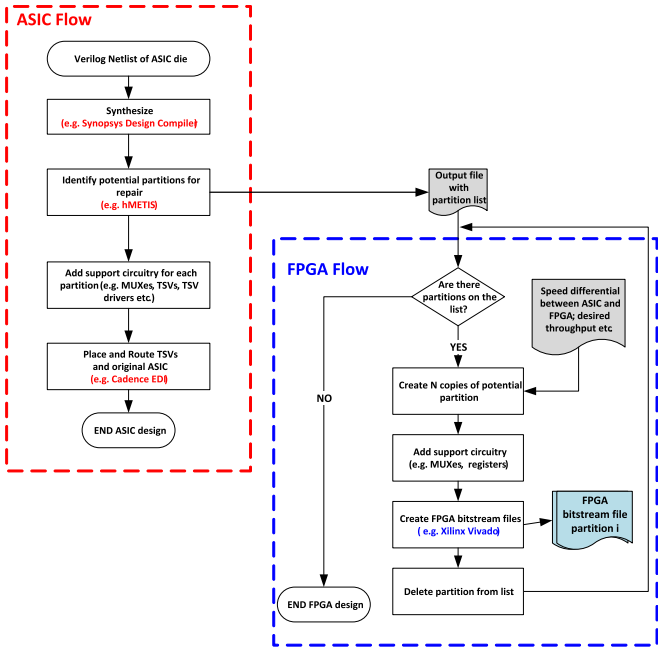

Fig. 3. Design flow for repair.

repair of one of the predefined partitions when necessary. The FPGA flow consists of the following steps.

1) A repairable partition is selected from the list, and a new circuit netlist with multiple copies of the partition is created. In the simplest implementation, the number of copies that are needed is determined by the speed differential between the FPGA and ASIC clock signals of the design; however, other aspects, such as the utilization of the component for a typical application can also affect the ideal number of copies.
2) Additional support registers, multiplexers, etc., are added to the FPGA design to allow data from the ASIC layer to multiplex to the appropriate copy and to return data to the ASIC layer when execution completes.
3) The FPGA design is synthesized and a bit-stream is generated that can be loaded when repair of the desired partition is necessary.
4) Steps 1)–3) are repeated until a bitstream for repair of each of the partitions that are identified as potential candidates for repair is generated.

IV. Investigation of Architecture and Timing Characteristics of Repair in 2.5-D FPGA

Several practical issues must be addressed when using an FPGA for BISR. Among others, this includes the timing

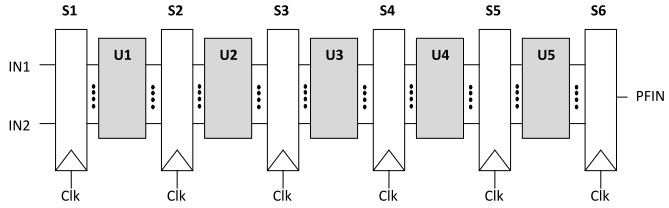

Fig. 4. Pipelined architecture for a multiplier.

and clock synchronization of the design through the TSVs. Although the operating frequency of FPGAs has increased dramatically, allowing them to replace ASICs in some designs, FPGAs are generally still slower than ASICs. Allowing some ASIC functionality to be replaced with functionality on the FPGA could require the ASIC speed to be reduced to the FPGAs clock frequency. In some applications, the corresponding reduction of performance could still be preferable to a totally nonfunctioning part.

However, for other designs, this reduction in ASIC clock frequency may not be required. In particular, for some pipelined designs, we may be able to handle the additional latency of computation while maintaining the overall throughput by instantiating multiple correct copies of the defective partition in the FPGA and multiplexing among them.

Such an approach is particularly suitable for pipelines without feedback or for functional units in an out-of-order execution machine. We will show that, in the case of pipelined circuitry, where partitions for replacement are defined as pipeline stages, the lack of feedback from later in the pipeline means that an increase in latency of the instruction will not cause stalls as long as that pipeline stage is multiplexed between multiple correct copies on the FPGA. A new instruction (or data) can still enter the pipeline on each cycle. Modifications to data forwarding hardware are also not necessary.

In the case of an out-of-order execution machine, control logic to handle execution units with different latencies and allow results to pass between them is already present. Thus, adding the equivalent of new functional units with slightly longer latencies does not require an unreasonable number of changes to the control logic.

A. Mapping the Design to Xilinx 2.5-D Virtex-7 2000T FPGA

We are not aware of any commercial tools that would allow us to model both an ASIC and an FPGA in a 3-D die-stack. However, tools do exist for modeling multiple FPGA dies connected by TSVs. Specifically, Xilinx has recently begun

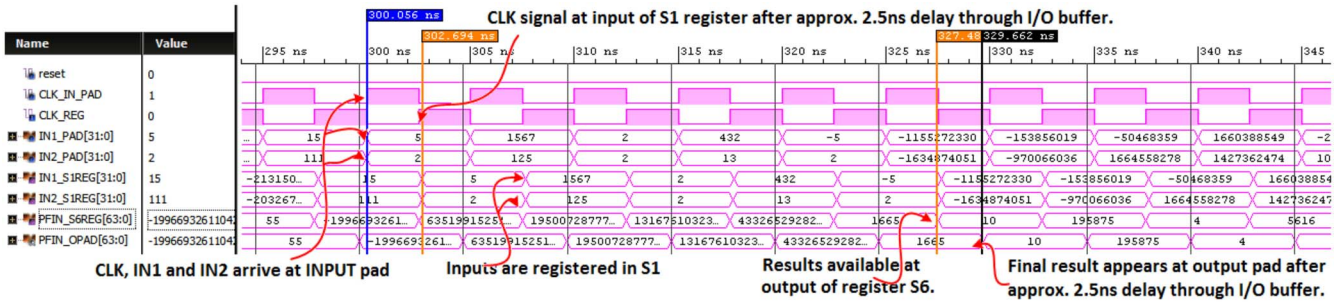

Fig. 5. Simulation result of the pipelined multiplier implemented in SLR1.

making 2.5-D Virtex-7 2000T series chips, which consist of four FPGA dies [also known as super logic regions (SLRs)] placed next to each other on a single silicon interposer and connected to each other through TSVs [25]. Although we do not currently have physical access to a Virtex-7 2000T, the Vivado software suite provided by Xilinx is still capable of providing important simulation-based post-implementation timing information.

One important aspect of these chips is that they allow circuitry on different SLRs to be run at different clock frequencies. Thus, for our investigation, we allowed one SLR to contain the logic that would be present on the ASIC while a second SLR contained the logic for repair of the defective partition. The second SLR (representing the FPGA) is run at half the frequency of the first (representing the ASIC).

To provide a proof-of-concept implementation of a pipelined design, and to investigate the timing and synchronization issues of passing data between an ASIC and FPGA layer through TSVs, we investigated repair of a simple pipelined multiplier with five pipeline stages, as shown in Fig. 4. The multiplier was generated using the ARITH project website and pipelined into five stages [34]. No feedback is present in the pipeline. All communication between the two SLRs occurs through TSVs. A goal of this portion of the research is to show that using two copies of the defective partition in the second SLR can indeed maintain the overall throughput of the pipelined multiplier when the clock of the second SLR is run at half the frequency of the first.

B. Timing of the Multiplier Without Repair

The high-level structure of the multiplier is shown in Fig. 4. The multiplier takes two 32-bit signed numbers as inputs: IN1 and IN2. The signed output is 64-bits and is named PFIN (for final product). The multiplier contains five pipeline stages: U1 through U5. Each pair of stages is separated by a pipeline register (S2 through S5). In addition, both the inputs and outputs of the multiplier are registered with pipeline registers S1 and S6.

Our first task was to determine the maximum clock frequency of the multiplier when the entire multiplier was placed on a single SLR. This is the best case scenario because no repair is necessary. To achieve this, the multiplier was first synthesized and mapped to a 2.5-D Xilinx Virtex-7 (xc7v2000tfhg1761) device using Xilinx Vivado software. All logic, including all stage registers, combinational logic, and all inputs/output were implemented on a single SLR (SLR1) out of the four available.

TABLE I
PIPELINE TIME-STAMPS FOR DATA BEING CLOCKED INTO EACH STAGE REGISTER OF ARCHITECTURE SHOWN IN FIG. 4

Stage	Input pad	S1	S2	S3	S4	S5	S6	Output Pad
time(ns)	300	302.5	307.5	312.5	317.5	322.5	327.5	330

Once the design was placed and routed, vivado post-implementation timing simulation was used to verify circuit operation using a standard delay format file with annotated part/net delays. The minimum clock period of the design was calculated to be 5 ns.

Fig. 5 shows a snapshot of the multiplication timing waveforms. The 32-bit signals IN1_PAD and IN2_PAD are the values at the input pads of the system. Markers are added to the figure to help point to important aspects of the timing simulation. The first marker is at 300 ns and shows the values of IN1 and IN2 taking on the values of 5 and 2, respectively at the INPUT pads of the device. The I/O pad buffers on the Virtex-7 have a delay of approximately 2.5 ns, and the inputs are clocked into stage register S1 at approximately 302.5 ns, as shown by the second marker.

Based on the pipeline shown in Fig. 4, the output PFIN should be ready five clock cycles after the input values, IN1 and IN2, are clocked into the S1 pipeline register. With a 5 ns clock period, this corresponds to 25 clock cycles. Accordingly, the next marker shows that the value of PFIN_O_S6REG, which is the output of register S6, becomes equal to the correct value of $10(5 * 2 = 10)$ at approximately 327.5 ns.

The output of the S6 stage register then goes through an output pad buffer with a delay of another 2.5 ns. The additional delay through I/O buffers for both the input and output signals adds approximately 5 ns to the overall computation time. Thus, the total time between the input pad and output pad of the multiplier is approximately 30 ns. For clarity, we summarize the time-stamps for data being clocked into each stage register in Table I.

C. Architecture and Multiplier Repair

Next, we assume that one of the stages in the pipelined multiplier is flagged by error monitoring circuitry. The error monitoring circuit could be based on a number of runtime error

TABLE II
PIPELINE STAGE TIME-STAMPS FOR REPAIR WITH TWO COPIES ON THE FPGA. THE FPGA CLOCK IS RUNNING AT HALF THE SPEED OF THE ASIC CLOCK

	In Pad	S1	S2	S3	S3F1	S3F2	S4F1	S4F2	S4	S5	S6	Out Pad
Mult1	295	297.5	302.5	307.5	312.5	X	322.5	X	327.5	332.5	337.5	340
Mult2	300	302.5	307.5	312.5	X	317.5	X	327.5	332.5	337.5	342.5	345
Mult3	305	307.5	312.5	317.5	322.5		332.5	X	337.5	342.5	347.5	350
Mult4	310	312.5	317.5	322.5	X	327.5	X	337.5	342.5	347.5	352.5	355

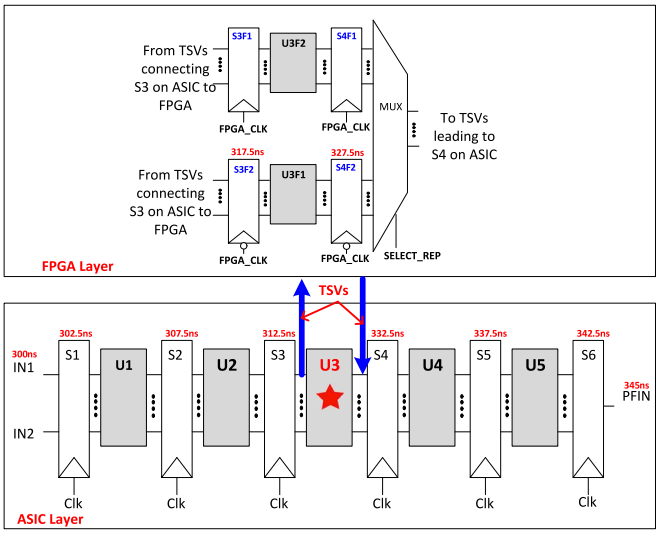

Fig. 6. Repair of stage U3 done using two copies on the FPGA. The time stamps at each input, output, and register reflects the time-stamp of Mult2 instruction shown in Table II.

detection approaches (e.g., logic implications [35]–[37], parity check codes [38], [39], and selective duplication [40], etc.). For this example, we assume that the error monitoring circuit is in place in the original ASIC layer and is able to trigger the dynamic reconfiguration of the FPGA layer for proper repair. Alternatively, the defective portion could be found through test either at manufacturing or in the field on startup. As mentioned earlier, which portion(s) of the circuit can be repaired must be determined *a priori*. Let us assume that multiplier stage U3 is faulty and is in need of repair. For repair, we dynamically reconfigure the FPGA layer with a copy of the U3 stage. Recall that the original design was all in SLR1 layer of the Virtex-7; in this experiment, the repair will all be done on SLR2 layer using the TSVs and silicon interposers present in the Virtex-7 for communication between the two dies.

A number of studies have shown that a design implemented on an FPGA runs between two to four times slower than a design running on a dedicated ASIC [41], [42]. If we consider the delay on the FPGA fabric to be two times the delay of the ASIC, then a design implemented on the FPGA with a single replacement unit would require that we slow down the speed of the ASIC pipeline to one half because the FPGA unit would not be capable of accepting a new set of pipeline inputs from ASIC stage register S3 on every ASIC clock cycle.

To maintain throughput, at least two copies on the FPGA would be needed as shown in Fig. 6, one which captures data on the rising edge of the FPGA clock and one which captures data on the falling edge. Note that the ASIC and FPGA clocks should be appropriately synchronized as mentioned earlier in Section II-D. This is done automatically by the Vivado software during clock generation and logic placement and routing.

In our experiment, the ASIC (represented by SLR1) is run with a clock period of 5 ns, and the clock period of the FPGA layer (represented by SLR2) is 10 ns. In this case, all odd instructions are processed by the rising edge triggered FPGA module (register S3F1, unit U3F1, and register S4F1 in Fig. 6) while all even instructions are processed by the falling edge triggered FPGA module (register S3F2, unit U3F2, and register S4F2). A multiplexer is used to transfer the result from S4F1 or S4F2 register to the ASIC via a single set of TSVs.

When there were no faulty pipeline stages, the processing time for data from register S3 to register S4 in Fig. 4 was only 5 ns. With the repair architecture shown in Fig. 6, it takes 5 ns for data in S3 in the ASIC (represented by SLR1) to transit through the TSVs and eventually be latched by register S3F1/S3F2 on the FPGA (SLR2). It takes 10 ns for the data to be processed in one of the replicated FPGA modules and latched in register in S4F1/S4F2 (in SLR2). Finally, it takes another 5 ns to transit back to SLR1 (representing the ASIC) via the multiplexer and TSVs and latch the resulting values in the S4 register. Thus, repair results in an additional 15 ns total (three ASIC clock cycles) of latency for each multiplication operation. However, the pipeline will still be able to accept new instructions at the original clock rate.

Table II shows the time-stamps (in nanoseconds) extracted from our Vivado simulation for four different multiply instructions under the repair scenario illustrated in Fig. 6. We see that with the repair done using two FPGA units, a new multiply instruction can appear at the input pad every 5 ns; similarly, the outputs for a new multiply instruction are also ready every 5 ns. Although the latency of each multiplication instruction has increased by three ASIC clock cycles, the rate at which data enters and leaves the pipeline is maintained at the original speed—helping to hide the additional latency by maintaining the average throughput. A post-implementation timing simulation of the repair is shown in Fig. 7.

D. Extension to Slower FPGAs

If we were to assume that the FPGA runs at a speed four times slower than the ASIC, then to maintain throughput, we would need four correct copies of the defective partition to be implemented on the FPGA. The four copies would need to be run by two separate FPGA clocks with one-fourth the frequency of the ASIC clock. The two FPGA clocks should be skewed by 90° in relation to each other so that exactly one of the two FPGA clocks has either a rising edge or falling edge on each rising edge of the ASIC clock. This will ensure

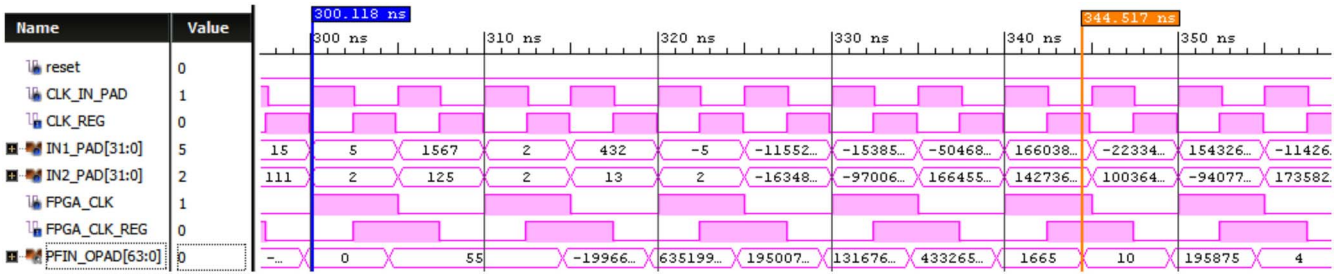

Fig. 7. Post implementation timing simulation of stage U3 being repaired using two copies of the FPGA. The original pipeline stages are implemented in SLR1 region of the Virtex-7 while the repair stage is implemented in SLR2 region. Inputs arrive at input pad at 300 ns and the product is available at the output pad 345 ns. Throughput is maintained.

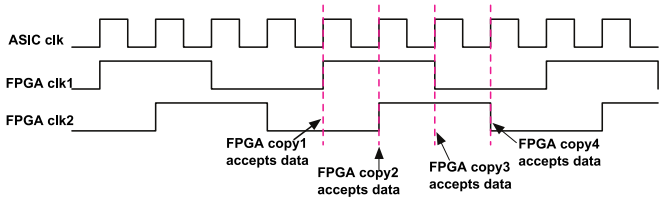

Fig. 8. Two quarter speed clocks skewed by 90° allow data to be captured by one of four partition copies on the FPGA on each rising edge of the ASIC clock.

that the data being passed from register S3 in the ASIC gets registered for processing in exactly one of the four good copies on the FPGA.

Fig. 8 shows how these two quarter-speed clocks may be used to pass data to each of the four good copies of the defective partition in turn. Specifically, the first copy on the FPGA registers its input data on the rising edge of the first FPGA clock. The second copy registers its input data on the rising edge of the second FPGA clock. Copies 3 and 4 register their input data on the falling edges of FPGA clocks 1 and 2, respectively. As shown in the figure, this allows a new set of input data to be passed to the next FPGA copy of the defective partition in sequence on each rising edge of the ASIC clock—allowing the ASIC pipeline to accept new data on each clock cycle without stalls. Theoretically, such an implementation would allow the ASIC to continue operating at the original clock frequency. However, as more copies are used on the FPGA, care must be taken to ensure that the routing delay to the extra copies does not exceed the time available. If the routing delay gets too large, either additional pipeline stages may be needed on the FPGA or the ASIC clock frequency may need to be reduced.

V. 3-D Layout Results

To determine the overhead of the required TSVs on the ASIC layer of our design, we first synthesized both a "repairable" and "nonrepairable" version of the multiplier using Synopsys design compiler [43]. We then performed 3-D layout using Cadence EDI [44]. Our circuits were mapped to the saed90nm_typ standard cell library files from the Synopsys 90 nm generic library [45].

The nonrepairable version of the multiplier consisted of the simple pipelined multiplier implemented on one ASIC layer with no logic added for repair. In contrast, the repairable version was implemented on two layers. The main layer (representing the ASIC) contained all of the original multiplier circuitry, including the MUXes, TSVs, and TSV drivers needed for communication with the second layer (which represents the FPGA). The second layer contained the required repair logic (i.e., extra registers, copies of the repaired partition, etc.). A total of 191 signals were routed between layers arranged in a 14 × 14 TSV array.

Note that driving buffers for the TSVs were inserted automatically during TSV insertion with Cadence EDI and are not tri-stated as shown in Fig. 1. Also, scan insertion was not performed, so the added MUXes were placed before simple D-flip-flops instead of MUX-D scan flip-flops, such as those shown in Fig. 2.

The total area for the nonrepairable version of the ASIC after layout was 108 041 μm^2. In contrast, the total area of the primary die that represented the repairable ASIC, including the TSVs that allowed connection to the second layer was 142 543 μm^2. In general, the cost of the extra TSVs is much larger than the cost of the added logic, and these TSVs add an approximately 32% area overhead.

A typical TSV area overhead range for 3-D stacks is 15–30%, depending on the size of the circuit and number of TSVs used in the stack [46]. Our circuit is much smaller than a typical 3-D die, so 32% area overhead is not unreasonable under these circumstances. It is also important to remember that these TSV connections can be shared for the repair of other portions of the ASIC die (e.g., other pipeline stages or functional units). Furthermore, these TSV connections can also serve a dual purpose of providing performance enhancement when no-repair is necessary. Thus, the cost of the TSVs can be amortized across multiple possible repairs and enhancements.

VI. Repair of Out-of-Order Processor Functional Units

In the previous section, we showed in detail how the proposed approach can be implemented for a defective partition corresponding to a single pipeline stage in a pipeline without feedback. However, one place where the proposed method is likely to be especially well-suited is the replacement of defective functional units in an out-of-order execution microprocessor. Such processors already contain the control logic necessary to handle functional units of varying latencies and pass results between instructions that start and complete their execution out-of-order. Thus, only minor modifications to this

TABLE III
BASELINE PROCESSOR CONFIGURATION

Parameters	Baseline Configuration
Decode/Issue/Commit width	4 (inst/cycle)
Fetch Queue (IFQ) size	16
Register Update Unit (RUU) size	64
Load/Store Queue Size	16
Functional Units (Integer)	1 or 3 ALUs 1 Multiplier/Divider
Functional Units (Floating Pt)	2 ALU 1 Multiplier/Divider

logic should be necessary to allow repairs to be implemented. Furthermore, it is in the execution stage that performance enhancement is likely to be performed for particular applications through the temporary implementation of new specialized functional units on the FPGA. Such performance enhancement will already require TSV connections to the FPGA, which can be repurposed for repair when necessary.

Unfortunately, in the case of repair, we still face the prospect that the time required to communicate data to/from the FPGA and process that data in the FPGA may be longer than that required for a nondefective functional unit in the ASIC layer to complete its computation. Thus, in this portion of the investigation, we studied the potential performance loss and gains when one or more copies of a functional unit that operate at a longer latency is used to replace an original, faster functional unit. As already stated, we study both the case where repair is necessary, because only a single copy of the defective unit exists in the original processor, and the case where repair is optional, because at least one good copy of the functional unit is still available.

A. Experimental Setup

All these experiments analyzed the effect of repair on the performance of applications from the SPEC95 [47] integer benchmark suite using the SimpleScalar [48] simulator. The SimpleScalar out-of-order processor model (sim-outorder) is an execution-driven simulation engine that reproduces the super-scalar processor's internal operations and provides a detailed micro-architectural timing model. For our experiments, we chose a 4-wide processor as the baseline processor configuration. The details of the configuration are reported in Table III. Note that depending on the particular experiment, the number of integer ALUs in the nondefective machine may be equal to 1 or 3. We explored two scenarios for repair of a 4-wide out-of-order processor.

1) A critical component is damaged and cannot be bypassed in software, making the entire stack unusable if a hardware-based approach is not available. We investigated the case where the only multiplier or the only integer ALU present in the processor must be repaired.
2) A component is damaged but the original processor contains multiple copies (1–3 in our experiments). The defective component may cause a loss of performance if not repaired. We quantify the performance loss and explore ways to mitigate performance loss through the use of spare components implemented in the FPGA.

B. Performance Impact When Repair is Necessary

First, we consider the case where the single integer ALU in the original processor is defective. The performance results after repair are shown in the top portion of Table IV. The top part of Table IV shows the percentage change in instructions per cycle (IPC) that occurs when the only integer ALU in the ASIC is defective and is replaced with 1, 2, 3, or 4 unpipelined integer ALUs in the FPGA. The ALUs in the FPGA were simulated for latencies varying between 2 and 4 times the ASIC ALU latency. In this case, the latency is assumed to correspond to the entire time required to send data to the FPGA, calculate the result, and return the data so that it can be latched in the ASIC layer. We also varied the number of copies of the FPGA functional unit from 1 to 4 in an attempt to hide the additional latency while maintaining throughput, as described in Section IV.

Clearly, replacing the single ALU in the machine with a single slower ALU in the FPGA can have a significant negative impact on performance. For example, the third column shows the case where the ALU operations take twice as long when sent to the FPGA (latency = 2) and only one copy of the ALU is instantiated in the FPGA. In that case the performance for the programs drops by 44.7% on average. Note that the alternative to the reduced performance is a completely nonfunctioning chip. However, if we increase the number of FPGA ALUs appropriately, we can entirely negate the performance loss. As shown in the next column, if we increase the number of FPGA-based ALUs to 2 (where each of the ALUs has a latency of 2), the performance drop is now only 1.4% on average. Furthermore, if we continue to increase the number of ALUs in the FPGA, we actually get performance gains. Similar results occur for FPGA copies with latencies of 3 or 4. Unsurprisingly, the number of copies we need in the FPGA increases as the latency increases.

We next performed similar experiments using the integer multiplier. The instruction profile in Table V shows that, of the seven benchmarks tested, only two benchmarks (ijpeg and vortex) made any use of the built-in integer multiplier unit. The performance results for those two benchmarks are shown in the bottom portion of Table IV. (We verified that there was no effect on the other benchmarks.) Our experiments show that even if the FPGA multiplier's latency is three times the latency of the original ASIC multiplier, having two FPGA multipliers will restore the performance to the fault-free case. These experiments indicate that the amount of effort placed toward repair should be decided in the context of the criticality of the unit in question. Although some repair is necessary for these instructions to operate correctly, the fact that they are used so rarely indicates that fewer resources (i.e., fewer copies) should generally be used for repair.

C. Performance Impact When Repair is Optional

Next, we considered the case where there are three integer ALUs in the ASIC, one of which is defective. If no repair of the bad ALU is performed, the system has only two usable integer ALUs, resulting in a loss of performance. Fig. 9 shows the percent performance degradation (measured in IPC).

TABLE IV
PERCENTAGE CHANGE IN PERFORMANCE FOR UNPIPELINED FPGA ALU/MULT REPLACING THE ONLY ASIC INTEGER ALU/MULT UNIT

Functional Unit	Benchmark	# of FPGA ALU Latency 2				# of FPGA ALU Latency 3				# of FPGA ALU Latency 4			
		1	2	3	4	1	2	3	4	1	2	3	4
ALU	compress	-41.7	-2.2	25.7	39.8	-58.7	-28.1	-1.3	17.1	-68.1	-42.9	-20.3	-1.6
	gcc	-45.5	-0.6	32.8	57.8	-62.3	-29.0	-0.4	23.4	-71.8	-45.7	-21.7	-0.4
	go	-38.4	-2.4	18.6	29.8	-55.0	-23.1	-1.8	13.0	-66.5	-38.7	-17.4	-1.6
	ijpeg	-48.9	-0.1	44.2	83.4	-65.5	-32.2	-0.4	29.8	-74.2	-48.8	-24.5	-0.5
	li	-46.6	-0.8	33.2	57.1	-62.3	-28.3	0.1	23.4	-72.3	-45.6	-21.9	0.0
	perl	-47.1	-2.7	31.9	50.6	-62.5	-29.1	-2.3	23.4	-72.4	-47.8	-25.7	-3.9
	vortex	-44.6	-1.3	32.1	52.3	-61.3	-28.2	-0.4	22.3	-70.9	-44.7	-21.1	-0.7
	ALU Avg.	-44.7	-1.4	31.2	53.0	-61.1	-28.3	-0.9	21.8	-70.9	-44.9	-21.8	-1.3
MULT	ijpeg	-0.7	0.1	0.2	0.2	-5.1	0.0	0.1	0.2	-8.9	-0.3	0.0	0.2
	vortex	-0.2	0.1	0.1	0.1	-0.4	0.1	0.1	0.1	-0.6	0.1	0.1	0.1
	MULT Avg.	-0.4	0.1	0.1	0.1	-2.7	0.0	0.1	0.1	-4.7	-0.1	0.1	0.1

TABLE V
PERCENTAGES OF INSTRUCTIONS THAT USE THE INTEGER ALU AND THE INTEGER MULTIPLIER

benchmark	% ALU inst	% multiplier inst
compress	61.5	0.0
gcc	79.1	0.0
go	69.1	0.0
ijpeg	80.9	1.2
li	68.4	0.0
perl	71.9	0.0
vortex	80.0	0.2

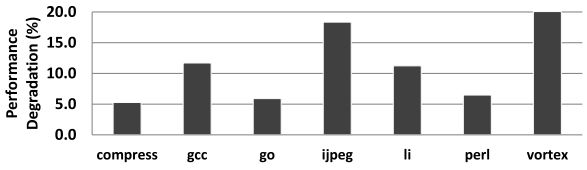

Fig. 9. Performance degradation with the loss of 1 integer ALU when no repair is performed.

The performance degradation varies between 5.3% for the compress benchmark to as high as 20% for vortex; on average performance decreases 11.3% across the benchmarks.

We then created a new pipelined version of the ALU with increased latency to represent ALUs that would be implemented on the FPGA. Because they are pipelined, these ALUs may accept a new calculation on every cycle even though they may take multiple cycles to complete. This may occur when the ASIC and FPGA can execute at the same internal clock frequency, but the realization of the ALU in the FPGA is such that we need to pipeline it to utilize that frequency and/or handle the propagation time through the TSVs. We measured the performance gains over the nonrepair case of the processor when different numbers of FPGAs are used for repair.

The results are shown in Table VI. On average, we see performance improvement of approximately 8.9% over the no-repair case when the single FPGA ALU has a latency of 2. The performance improvement falls to 7.6% and 3.3% when the FPGA ALU latencies increase. Once again, this indicates that when the increased latency of the FPGA ALU is too high, a single copy on the FPGA for replacement is not enough. In fact, benchmarks compress and perl show a slight performance degradation of 0.5% and 2.8%, respectively when the FPGA ALU has a latency of 4. This is because once an instruction is issued to an ALU, it is tied-up within that ALU until it finishes. No dependent instructions can execute in the meantime. When the latency of an ALU becomes very long, in some cases it may actually be better to wait for a shorter latency ALU to become available. This also appears to be related to what is happening to the vortex benchmark. That benchmark has a higher degree of instruction-level parallelism than others, and its performance appears to be very dependent on the order in which instructions are issued to the functional units. When these instructions are issued in a different order due to changes in instruction latencies, it has a significant effect.

From the results of the pipelined experiments shown in Table VI, we see that adding two FPGA ALUs provides additional performance improvement. However, adding more FPGA ALUs (either 3 or 4) produces no additional benefit. This is to be expected. Our baseline processor is a 4-wide superscalar. Because there are two working integer ALUs in the processor, adding two more on the FPGA allows up to four integer instructions to be issued on any given cycle. Each of those two FPGA ALUs can accept a new instruction each cycle because they are pipelined. However, because only four instructions can be issued on each clock cycle in a 4-wide machine, adding additional ALUs on the FPGA adds no improvement. Overall, we can conclude that repair with two pipelined FPGA ALUs appears to be ideal, provided that the latency of the FPGA ALUs is not too long.

We also ran the same experiments where the added FPGA ALUs were not pipelined. The addition of unpipelined FPGA ALUs still results in a performance improvement over the unrepaired case, as shown in Table VII. However, since the FPGA ALU is marked busy and unavailable for a certain number of cycles when it is in use, the performance improvement for 1 additional ALU is smaller than in the corresponding pipelined case–6.1% instead of 8.9% for latency of 2; 3.6% instead of 7.6% for latency 3 and 1.2% instead of 3.3% for latency 4. However, unlike the pipelined FPGA ALU case, adding 3 and 4 FPGA ALUs can positively impact performance in the nonpipelined case. The processor can take advantage of a fifth or a sixth ALU (which might not be busy) and issue a ready instruction present in the issue queue. The gains from the availability of these additional ALUs makes the unpipelined case approximately match the peak

TABLE VI
PERCENTAGE IMPROVEMENT (OVER THE NO REPAIR CASE) IN PERFORMANCE WHEN THE BAD INTEGER ALU IS REPAIRED WITH 1–4 PIPELINED INTEGER ALU OF LATENCY 2–4 ON THE FPGA

Benchmark	# of FPGA ALU Latency 2				# of FPGA ALU Latency 3				# of FPGA ALU Latency 4			
	1	2	3	4	1	2	3	4	1	2	3	4
compress	3.3	3.3	3.3	3.3	1.8	1.7	1.7	1.7	-0.5	-1.1	-1.1	-1.1
gcc	10.9	12.3	12.3	12.3	7.6	9.3	9.3	9.3	3.3	3.7	3.7	3.7
go	5.1	5.8	5.8	5.8	3.5	3.9	3.9	3.9	1.8	1.7	1.7	1.7
ijpeg	20.9	28.7	28.7	28.7	18.8	24.3	24.3	24.3	16.0	19.6	19.6	19.6
li	10.4	12.7	12.7	12.7	6.8	8.6	8.6	8.6	3.1	4.2	4.2	4.2
perl	5.3	5.2	5.2	5.2	2.6	0.9	0.9	0.9	-2.9	-2.4	-2.4	-2.4
vortex	6.7	7.7	7.7	7.7	12.2	10.4	10.4	10.4	2.5	3.0	3.0	3.0
AVERAGE	8.9	10.8	10.8	10.8	7.6	8.4	8.4	8.4	3.3	4.1	4.1	4.1

TABLE VII
PERCENTAGE IMPROVEMENT (OVER THE NO REPAIR CASE) IN PERFORMANCE WHEN THE BAD INTEGER ALU IS REPAIRED WITH 1–4 UNPIPELINED INTEGER ALU OF LATENCY 2–4 ON THE FPGA

Benchmark	# of FPGA ALU Latency 2				# of FPGA ALU Latency 3				# of FPGA ALU Latency 4			
	1	2	3	4	1	2	3	4	1	2	3	4
compress	2.9	3.3	3.3	3.3	0.1	1.7	1.7	1.7	-1.1	-0.7	-1.1	-1.1
gcc	7.6	11.1	11.9	12.3	3.3	6.2	8.3	9.0	0.0	1.6	2.7	3.4
go	3.6	5.2	5.7	5.8	1.9	3.0	3.5	3.8	0.5	1.0	1.4	1.5
ijpeg	12.9	21.0	26.0	28.7	8.6	14.7	19.1	21.7	6.4	11.1	14.7	16.6
li	7.3	11.0	12.6	12.7	3.0	6.6	8.2	8.6	2.1	2.9	3.5	4.1
perl	3.9	5.4	5.1	5.2	1.4	1.1	1.7	1.2	-0.2	-1.4	-1.8	-3.2
vortex	4.2	6.9	7.7	7.7	7.2	10.5	8.9	10.1	0.6	1.5	2.4	2.8
AVERAGE	6.1	9.1	10.3	10.8	3.6	6.2	7.4	8.0	1.2	2.3	3.1	3.4

performance gain of the pipelined case. For most benchmarks, the final improvement of the repaired version (when repair is not mandatory) allows it to closely reach the performance (within 2%) of the nondefective version if enough FPGA ALUs (2 pipelined FPGA ALUs or 4 nonpipelined FPGA ALUs of latency 2) are added.

VII. DIAGNOSIS AND TEST OF FAILING PARTITIONS

Another aspect of 3-D IC repair is the requirement to diagnose which portion of a chip is defective so that the right partition can be bypassed and replaced. In the results presented in Section VI, we have assumed that some sort of diagnosis approach exists. Note that diagnosis for repair is easier than diagnosis for yield learning. Only the defective partition must be identified. There is no need to identify the exact fault or defective site.

While, the exact nature of the diagnosis approach in a 3-D IC stack is outside the scope of this paper and a subject of our future investigations, [35]–[37] on online error detection in a 2-D IC could be adapted to identify the location of some defective behavior in a 3-D stack. For example, we have previously demonstrated that when logic implications are used for online-error detection, determining which implication was violated provides automatic diagnostic data because each implication can only protect a well-defined subset of all possible circuit sites [37]. If all the suspected circuit sites are located within a single partition, the exact partition to be repaired can be identified automatically. If the suspect sites are scattered through multiple partitions, then a short test set applied by the FPGA operating as a tester or another core in the chip could potentially be used to provide more diagnostic resolution. Other online error detection schemes may provide similar information. For example, if a particular functional unit is protected by parity, then any errors in the parity signature may automatically be associated with that functional unit. As a part of future work, we are considering various online-error detections schemes, how their diagnostic resolution relates to the partitions we are creating for repairability on the FPGA, and what kinds of test sets or on-chip data collection may be needed to increase that resolution.

VIII. CONCLUSION

In this paper, we have described an architecture that will allow a 3-D stacked IC to be repaired in the presence of a defective ASIC through the repurposing of programmable logic already included in the stack for other purposes, such as performance enhancement. We have investigated bypassing both a single defective pipeline stage of a multiplier with a correct copy of the stage on an FPGA as well as bypassing and replacing a defective functional unit in an out-of-order microprocessor.

FPGAs often operate at a slower clock speed than ASICs; however, this paper has shown that the flexibility introduced by the use of an FPGA can make the performance drop introduced by repair minimal or nonexistent. The key to handling the clock speed differential is to implement multiple copies of the defective partition on the FPGA. In particular, even when the FPGA clock period is one half of the ASIC clock period, a new instruction can be sent to the FPGA on every clock cycle—requiring no reduction in the ASIC clock frequency when two copies of the defective partition are implemented on the FPGA. Our simulations regarding the repair of functional units in an out-of-order microprocessor using SimpleScalar show that performance of the repaired version may even exceed the original performance in some cases if a sufficient number of copies of a nonpipelined defective functional unit are added for repair.

We have also mapped our proposed architecture to a 3-D stacked IC architecture using commercial tools from

Synopsys and Cadence. The most significant overhead occurs due to the need for TSVs to communicate between the ASIC and FPGA layers. However, our data shows that the TSV area requirements are reasonable, especially when it is considered that these same TSVs may be harnessed for multiple purposes. In particular, the same set of TSVs may be multiplexed and used to repair multiple potentially defective partitions. The overhead of the TSVs may even be nonexistent if they were already included in the design to allow new functional units to be temporarily added to the machine for performance enhancement. In addition, the timing overhead seen by the original ASIC due to the introduction of multiplexers used to return data from the FPGA can be hidden by adding those multiplexers to the scan-in pin of a MUX-D flip-flop in a standard full-scan design.

Our future work will expand the proposed solution to additional designs and repair scenarios. We will also further investigate additional shared uses for the FPGA in the stack, including using the FPGA for built-in-self-test and diagnosis of stack components.

REFERENCES

[1] A. Crouch and J. Dworak, "What is 3-D test and how do IEEE standards help?," *Electr. Device Fail. Anal.*, vol. 13, no. 4, pp. 4–13, Nov. 2011.

[2] E. Marinissen and Y. Zorian, "Testing 3-D chips containing through-silicon vias," in *Proc. Int. Test Conf. (ITC)*, Austin, TX, USA, Nov. 2009, pp. 1–11.

[3] K. Nepal, X. Shen, J. Dworak, T. Manikas, and R. Bahar, "Built-in self-repair in a 3-D die stack using programmable logic," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, New York, NY, USA, Oct. 2013, pp. 243–248.

[4] M. Choi, N. Park, F. Lombardi, and V. Piuri, "Quality enhancement of reconfigurable multichip module systems by redundancy utilization," *IEEE Trans. Instrum. Meas.*, vol. 51, no. 4, pp. 740–749, Aug. 2002.

[5] T. Miller, N. Surapaneni, and R. Teodorescu, "Flexible error protection for energy efficient reliable architectures," in *Proc. 22nd Int. Symp. Comput. Architect. High Perform. Comput. (SBAC-PAD)*, Petrópolis, Brazil, Oct. 2010, pp. 1–8.

[6] T. Austin, "DIVA: A reliable substrate for deep submicron microarchitecture design," in *Proc. 32nd Annu. Int. Symp. Microarchitect. (MICRO-32)*, Haifa, Israel, 1999, pp. 196–207.

[7] R. Iyer, N. Nakka, Z. Kalbarczyk, and S. Mitra, "Recent advances and new avenues in hardware-level reliability support," *IEEE Micro*, vol. 25, no. 6, pp. 18–29, Nov./Dec. 2005.

[8] C.-S. Hou, J.-F. Li, and T.-W. Tseng, "Memory built-in self-repair planning framework for RAMs in SOCs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 11, pp. 1731–1743, Nov. 2011.

[9] S.-K. Lu, Z.-Y. Wang, Y.-M. Tsai, and J.-L. Chen, "Efficient built-in self-repair techniques for multiple repairable embedded RAMs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 4, pp. 620–629, Apr. 2012.

[10] C.-W. Chou, Y.-J. Huang, and J.-F. Li, "A built-in self-repair scheme for 3-D RAMs with interdie redundancy," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 4, pp. 572–583, Apr. 2013.

[11] C. Lee, W. Kang, D. Cho, and S. Kang, "A new fuse architecture and a new post-share redundancy scheme for yield enhancement in 3-D-stacked memories," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 33, no. 5, pp. 786–797, May 2014.

[12] L. Jiang, Q. Xu, and B. Eklow, "On effective through-silicon via repair for 3-D-stacked ICs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 4, pp. 559–571, Apr. 2013.

[13] Y.-J. Huang and J.-F. Li, "Built-in self-repair scheme for the TSVs in 3-D ICs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 10, pp. 1600–1613, Oct. 2012.

[14] V. Pasca, L. Anghel, M. Nicolaidis, and M. Benabdenbi, "CSL: Configurable fault tolerant serial links for inter-die communication in 3-D systems," *J. Electr. Test. Theory Appl.*, vol. 28, no. 1, pp. 137–150, Feb. 2012.

[15] T.-H. Wu *et al*. "A memory yield improvement scheme combining built-in self-repair and error correction codes," in *Proc. IEEE Int. Test Conf. (ITC)*, Anaheim, CA, USA, Nov. 2012, pp. 1–9.

[16] T. Koal and H. Vierhaus, "Optimal spare utilization for reliability and mean lifetime improvement of logic built-in self-repair," in *Proc. IEEE Int. Symp. Design Diagn. Electr. Circuits Syst. (DDECS)*, Cottbus, Germany, Apr. 2011, pp. 219–224.

[17] S. Di Carlo, A. Miele, P. Prinetto, and A. Trapanese, "Microprocessor fault-tolerance via on-the-fly partial reconfiguration," in *Proc. 15th IEEE Eur. Test Symp. (ETS)*, Prague, Czech Republic, May 2010, pp. 201–206.

[18] J. Nunes, "Improving the dependability of FPGA-based real-time embedded systems with partial dynamic reconfiguration," in *Proc. IEEE/IFIP Conf. Depend. Syst. Netw. Workshop (DSN-W)*, Budapest, Hungary, Jun. 2013, pp. 1–4.

[19] J. Jeddeloh and B. Keeth, "Hybrid memory cube new DRAM architecture increases density and performance," in *Proc. Symp. VLSI Technol. (VLSIT)*. Honolulu, HI, USA, Jun. 2012, pp. 87–88.

[20] A. Crouch, "3-D test: You tell me and we'll both know," in *Proc. BiTS Workshop*, Mar. 2011, pp. 1–17.

[21] R. Chaware, K. Nagarajan, and S. Ramalingam, "Assembly and reliability challenges in 3-D integration of 28nm FPGA die on a large high density 65nm passive interposer," in *Proc. IEEE 62nd Electr. Compon. Technol. Conf. (ECTC)*, San Diego, CA, USA, Jun. 2012, pp. 279–283.

[22] L. Madden *et al*., "Advancing high performance heterogeneous integration through die stacking," in *Proc. Eur. Solid-State Device Res. Conf. (ESSDERC)*, Bordeaux, France, Sep. 2012, pp. 18–24.

[23] C. Claus, R. Ahmed, F. Altenried, and W. Stechele, "Towards rapid dynamic partial reconfiguration in video-based driver assistance systems," in *Proc. 6th Int. Conf. Reconfigurable Comput. Architect. Tools Appl.*, 2010, pp. 55–67.

[24] J. Xie and D. Patterson, "Realizing 3-D IC integration with face-to-face stacking," *Chip Scale Rev.*, vol. 17, no. 3, pp. 16–19, May/Jun. 2013.

[25] P. Dorsey, "Xilinx stacked silicon interconnect technology delivers breakthrough fpga capacity, bandwidth, and power efficiency," in *Proc. Xilinx White Paper Virtex-7 FPGAs*, Taipei, Taiwan, Oct. 2010, pp. 1–10.

[26] M. Buttrick and S. Kundu, "On testing prebond dies with incomplete clock networks in a 3-D IC using DLLs," *J. Electr. Test.*, vol. 28, no. 1, pp. 93–101, 2012.

[27] J.-W. Ke, S.-Y. Huang, C.-W. Tzeng, D.-M. Kwai, and Y.-F. Chou, "Die-to-die clock synchronization for 3-D IC using dual locking mechanism," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 60, no. 4, pp. 908–917, Apr. 2013.

[28] A.-J. Chuang, Y. Lee, and C.-Y. Yang, "A chip-to-chip clock-deskewing circuit for 3-D ICs," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Seoul, Korea, May 2012, pp. 1652–1655.

[29] C.-C. Chung and C.-Y. Hou, "All-digital delay-locked loop for 3-D-IC die-to-die clock synchronization," in *Proc. Int. Symp. VLSI Design Autom. Test (VLSI-DAT)*, Hsinchu, Taiwan, Apr. 2014, pp. 1–4.

[30] R. Oh, J. Jang, J. Kim, and M. Y. Sung, "Simultaneous process self-calibration method using TDC for 3-D DDR4 DRAM," *Electr. Lett.*, vol. 50, no. 22, pp. 1579–1581, Oct. 2014.

[31] (Aug. 1, 2014). *Xilinx Virtex-7 Data Sheet*. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds183_Virtex_7_Data_Sheet.pdf

[32] (Aug. 1, 2014). *7 Series FPGAs Clocking Resources User Guide*. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf

[33] G. Karypis and V. Kumar, "hMETIS," *A Hypergraph Partitioning Package (Version 1.5. 3)*, Dept. Comput. Sci. Eng., Univ. Minnesota, Minneapolis, MN, USA, 2007.

[34] (Aug. 1, 2014). *ARITH Project: High-Level Design Methodology for Integer/Galois-Field Arithmetic Circuits for Embedded Systems*. [Online]. Available: http://www.aoki.ecei.tohoku.ac.jp/arith/index.html

[35] K. Nepal, N. Alves, J. Dworak, and R. I. Bahar, "Using implications for online error detection," in *Proc. IEEE Int. Test Conf. (ITC)*, Santa Clara, CA, USA, Oct. 2008, pp. 1–10.

[36] N. Alves, A. Buben, K. Nepal, J. Dworak, and R. Bahar, "A cost effective approach for online error detection using invariant relationships," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 5, pp. 788–801, May 2010.

[37] J. Dworak *et al*., "Using implications to choose tests through suspect fault identification," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 18, no. 1, pp. 14:1–14:19, Jan. 2013.

[38] S. Ghosh, N. Touba, and S. Basu, "Synthesis of low power CED circuits based on parity codes," in *Proc. 23rd VLSI Test Symp.*, May 2005, pp. 315–320.

[39] N. Touba and E. McCluskey, "Logic synthesis of multilevel circuits with concurrent error detection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 16, no. 7, pp. 783–789, Jul. 1997.
[40] P. Drineas and Y. Makris, "Concurrent fault detection in random combinational logic," in *Proc. 4th Int. Symp. Qual. Electr. Design*, Mar. 2003, pp. 425–430.
[41] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 2, pp. 203–215, Feb. 2007.
[42] M. Genovese and E. Napoli, "ASIC and FPGA implementation of the Gaussian mixture model algorithm for real-time segmentation of high definition video," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 3, pp. 537–547, Mar. 2014.
[43] (Aug. 1, 2014). *Synopsys Design Compiler*. [Online]. Available: http://www.synopsys.com/tools/implementation/rtlsynthesis/Pages/default.aspx
[44] (Aug. 1, 2014). *Cadence EDI System*. [Online]. Available: http://www.cadence.com/rl/ resources/datasheets/edi_system_ds.pdf
[45] (Aug. 1, 2014). *Synopsys 90nm Generic Library*. [Online]. Available: http://www.synopsys.com/Community/UniversityProgram/Pages/Library.aspx
[46] D. H. Kim, K. Athikulwongse, and S. K. Lim, "A study of through-silicon-via impact on the 3-D stacked IC layout," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCCAD)*, San Jose, CA, USA, Nov. 2009, pp. 674–680.
[47] (Aug. 1, 2014). *SPEC95 Benchmark Suite*. [Online]. Availble: http://www.spec.org/ cpu95
[48] T. Austin, E. Larson, and D. Ernst, "Simplescalar: An infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, Feb. 2002.

Kundan Nepal (S'04–M'07–SM'12) 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 Engineering with the University of St. Thomas, Saint Paul, MN, USA. His current research interests include defect/fault tolerant circuits and systems, nanometer digital VLSI system design, and reconfigurable computing.

Soha Alhelaly received the B.S. degree in computer science from Umm al-Qura University, Mecca, Saudi Arabia, and the master's degree in computer information system from St. Mary's University, San Antonio, TX, USA, in 2011. She is currently pursuing the Ph.D. degree in computer science and engineering from Southern Methodist University, Dallas, TX, USA.

Her current research interests include security and trust in 3-D integrated circuits and VLSI design algorithms.

Jennifer Dworak (S'96–M'05) received the B.S.E.E. (*summa cum laude*), M.S.E.E., and Ph.D. degrees in electrical engineering from Texas A&M University, College Station, TX, USA, in 1998, 2000, and 2004, respectively.

She joined the faculty at Brown University, Providence, RI, USA. She is currently a Faculty Member with Southern Methodist University, Dallas, TX, USA. Her current research interests include digital circuit testing and automatic test pattern generation, online error detection and built-in-self repair, 3-D integration, secure test, and trust in integrated circuits.

R. Iris Bahar (S'93–M'96–SM'12) 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 1986, 1987, and 1995, respectively.

From 1987 to 1992, she was 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 a Professor of Engineering. Her current research interests include computer architecture, computer-aided design for synthesis, verification, and low-power applications, and design, test, and reliability issues for nanoscale systems.

Dr. Bahar currently serves as an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS and the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS.

Theodore Manikas (S'83–M'84–SM'06) received the B.S. degree from Michigan State University, East Lansing, MI, USA, the M.S. degree from Washington University in St. Louis, St. Louis, MO, USA, and the Ph.D. degree from the University of Pittsburgh, Pittsburgh, PA, USA, all in electrical engineering.

Since 2009, he has been with the Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX, USA. His current research interests include disaster-tolerant system design, computer architecture, and physical design automation.

Prof. Manikas is a Licensed Professional Engineer in Texas and Oklahoma.

Ping Gui (S'03–M'04–SM'09) received the Ph.D. degree in electrical and computer engineering from the University of Delaware, Newark, DE, USA.

She is currently an Associate Professor of Electrical Engineering with the Lyle School of Engineering, Southern Methodist University, Dallas, TX, USA, and joined the Faculty of Lyle School of Engineering, Southern Methodist University, in 2004. Her current research interests include digital, analog, mixed-signal, and RF IC design for a variety of applications including high-speed wireless and wireline communications, SoC design and test, and circuits and systems for harsh environments.

Dr. Gui was the recipient of the CERN Scientific Associate Award and the IEEE Dallas Section Outstanding Service Award. Since 2007, she has been the Technical Chair of the IEEE Solid-States Circuits Society Dallas Chapter and is currently serving on the Technical Program Committee of the IEEE Radio-Frequency Integrated Circuits Symposium.