Built-in Self-Repair in a 3D Die Stack Using Programmable Logic

Kundan Nepal*, Xi Shen†, Jennifer Dworak†, Theodore Manikas†, and R. Iris Bahar‡
*School of Engineering, University of St Thomas, St Paul, MN
†Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX
‡School of Engineering, Brown University, Providence, RI

***Abstract*—3D stacked integrated circuits hold great promise for increasing system performance, but difficulties in testing dies and assembling a 3D stack are leading to yield issues and slowing the large scale manufacture of these devices. We propose helping to mitigate these issues by *repairing* the stack with programmable logic in FPGAs that have already been included in the stack for other purposes. Specifically, we propose bypassing the defective portion of a die by replacing the defective functionality with functionality on the FPGA. In this paper, we focus on the replacement of selected defective functional units in an out-of-order microprocessor. 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.**

I. Introduction

As circuits approach the limits of Moore's law, and as power considerations have placed a limit on increases in clock frequency, the stacking of bare dies to form a 3D stack has been proposed as one approach that may allow significant increases in the performance of integrated circuits and systems. Performance gains are expected to arise primarily from the fact that the routes between dies in a stack, which may be as short as 30 microns, 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 3D integrated circuits is the difficulty in testing dies and obtaining high yields [1]. The insertion of through-silicon vias (TSVs) into a die 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—causing additional potential error sources [2]. Hot spots in the stack may affect several dies and cause either temporary incorrect operation or permanent damage. If we add these issues to standard problems of 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 3D 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 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 3D stack, routing between layers is short (on the order of 30 microns), and 10,000 TSVs (Through-Silicon-Vias) may be present in a square millimeter [2].

In this paper, we propose the use of an FPGA in the stack as a resource for replacing defective functionality with working functionality in the stack. 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. 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 will investigate the impact of replacing defective functional units in a 4-wide superscalar processor. We will 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 were one of multiple copies becomes defective, reducing performance. Our simulation results show that even when the functional units implemented in the FPGA are slower than the original defective copy, the performance loss can often be almost entirely masked with reasonable overhead.

978-1-4799-1585-9/13/$31.00

II. Background

A wealth of work has previously been done in fault tolerance and repair of circuits and systems. For example, Built-In-Self-Repair (BISR) is a fault tolerance technique that addresses permanent faults in both memories and logic. Generally, in addition to core operational components, a set of spare components is provided. If a faulty core component is detected, it is replaced with a spare component [3]. For example, a common application of BISR is in 2D RAM's, where spare memory units [4] or redundant row/columns [5] are implemented. This approach has also been expanded to 3D stacked memories, where spare resources are borrowed from adjacent dies [6] and spare TSVs are used to replace defective TSVs [7]. When not enough spares are available, configurable fault-tolerant Serial Links (CSLs) have been proposed for TSV repair [8].

BISR has also been applied to functional modules implemented in 2D circuits. One approach has been to use reconfigurable modules in FPGAs, such as logic blocks or routing resources, to replace the defective modules [9], [10]. Another approach is to use spare functional units on FPGAs [11], such as spare ALUs [12]. These approaches implement both the original and spare modules on a single FPGA. In addition, 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 [13].

FPGA companies are already manufacturing FPGAs containing TSVs. For example, Xilinx is currently selling a 2.5D version of the Virtex 7 FPGA that contains four FPGAs sitting side-by-side on a silicon interposer [14]. 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 3D. 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 3D stack are also much shorter than the routes on a board. Although some FPGAs, such as the Virtex 7, are currently very expensive, other high performance, low cost FPGAs are available. For example, new 22nm FPGAs are coming to the market that are intended to approach the performance of a Virtex at much lower cost and at low power [15]. Similar FPGAs would be a reasonable choice for inclusion in a 3D stack in the future.

Our paper extends the concept of built-in-self-repair to the digital logic in 3D stacks. We consider two separate dies in the 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 3D architectures—including potentially large numbers of TSV connections and short distances between dies—to increase the flexibility and improve the performance of repair. We will also show how our implementation reduces the effect of the possible speed differential between the ASIC and FPGA. To the best of our knowledge, our work is the first to propose this type of repair in 3D.

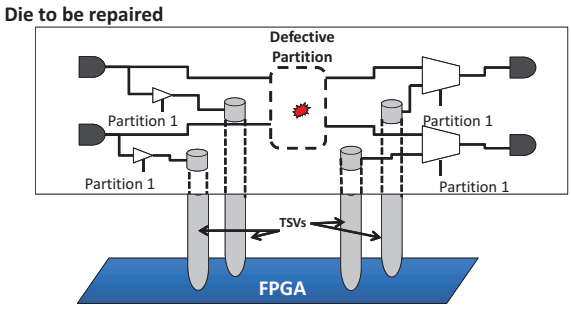

Fig. 1. Repair of a partition on the ASIC using the FPGA layer.

III. Methodology

To enable repair of an ASIC or processor using an FPGA in a 3D stack, advance planning is needed to ensure that the required support structure is available for the desired repair options. Specifically, which portions of the stack may be repaired (and at what level of granularity) should be decided *a priori*. Such decisions may be based on the criticality of the partition that could be repaired, the likelihood of failure, and tradeoffs involving the overall cost of the repair.

A. Basic Architecture of Replacement

Each partition that may be bypassed and replaced by FPGA functionality must have appropriate connections to a set of TSVs that are ultimately connected to the FPGA. Specifically, once a partition has been identified, driving buffers must be inserted in the original design to allow each input to the partition to be alternatively connected to the FPGA. Similarly, multiplexers must be inserted between the outputs of the partition being repaired and the downstream part of the circuit to allow the rest of the circuitry to be driven by the FPGA instead of by the defective partition. An example is shown in Figure 1. In this figure, each of the inputs to the partition fan out not only to the bypassable partition, but to a tri-stated buffer (or possibly a series of buffers) that is capable of driving the TSV as well. The driving buffers should be sized so as to minimize the load and delay seen by the circuit. Each of these TSVs is connected to the FPGA such that it becomes 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 pre-defined TSVs until they reach the level of the ASIC being repaired. Muxes are used to 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.

B. Tradeoffs and Overhead

1) Reducing TSV Overhead: One of the primary measures of overhead with this method lies in the need to allocate TSVs to allow the appropriate signals to be connected between the ASIC under repair and the FPGA implementing the bypass

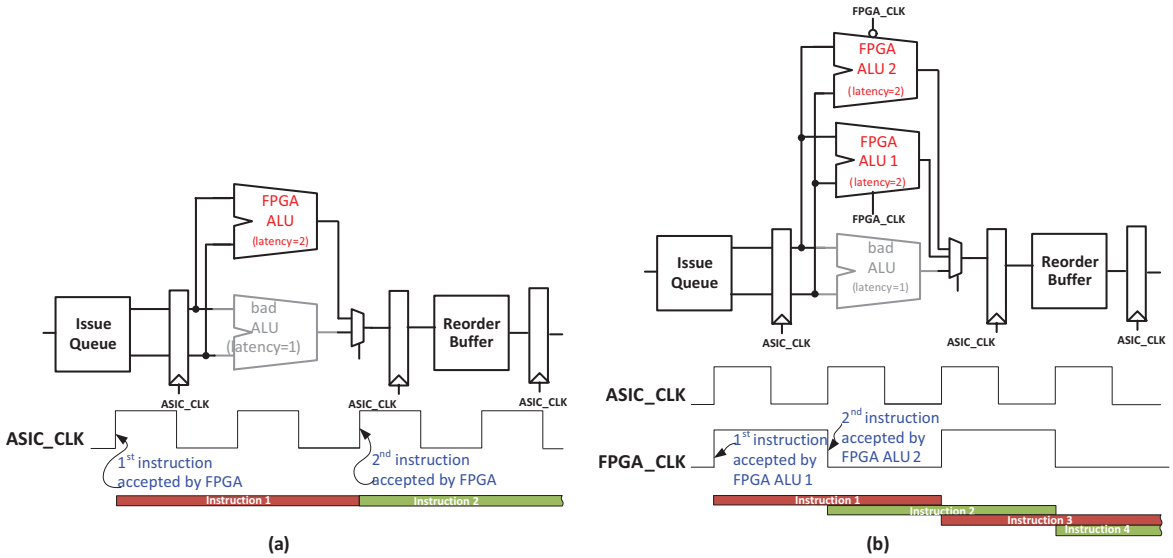

Fig. 2. Replacement of bad ALU with (a) single FPGA ALU (b) two FPGA ALUs triggered by rising and falling clock edges respectively.

logic. However, it is important to note that a single set of TSVs may be shared and used to replace any of several pre-defined partitions—while possibly restricting the total number of partitions that can be bypassed at any one time. Partitions may also be pre-determined in such a way as to reduce the number of inputs and outputs to/from each partition—further reducing the number of TSVs needed.

2) Hiding the Speed Differential between the ASIC and FPGA: In general, FPGAs are often assumed to be slower than an ASIC. However, in recent years, the speed of FPGAs has increased dramatically—putting them much closer to ASIC speeds—especially if an ASIC does not need to be ultra-high performance or implemented in the latest technology. Under these conditions, it may be possible to match the clock frequency in both the ASIC and the FPGA—making the performance cost of replacement very small. However, even if the FPGA is inherently slower than the ASIC hardware it is replacing, we may still be able to hide some of the additional latency while maintaining throughput by providing *multiple copies* of the hardware to be replaced in the FPGA. For example, in Figure 2, we are assuming that the speed of the FPGA is approximately one half the speed of the ASIC under repair. The portion under repair corresponds to a pipeline stage in a design that does not include feedback to earlier stages of the pipeline. This pipeline stage takes twice as long to execute in the FPGA, and thus it has double the latency. If only one copy is placed in the FPGA, as shown in part (a), we must slow the speed of the ASIC pipeline by one half because the FPGA is incapable of accepting a new set of pipeline inputs on every ASIC clock cycle. However, in part (b), we have implemented two copies on the FPGA—one of which captures data on the rising edge of the FPGA clock and one which captures data on the falling edge. (Note that the clocks should be appropriately synchronized.) In this case, all odd instructions are processed by the rising edge triggered FPGA module while all even instructions are processed by the falling edge triggered FPGA module. Although the latency of each instruction has increased by an extra ASIC clock cycle, the rate at which data enters and leaves the pipeline is maintained at the original speed—hiding the additional latency while maintaining the same throughput.

This approach is obviously extendable to additional copies in the FPGA. It is especially useful for repairing functional units in processor architectures that issue instructions to multiple functional units of different latencies—allowing them to complete out-of-order and resolving ordering issues with the standard Reorder Buffer and Commit logic. In that case, the additional latency is handled automatically by the underlying issue and commit hardware.

IV. Experimental Results

If the only copy of a critical component is damaged and cannot be bypassed in software, the stack becomes unusable if a hardware-based approach is not available. However, in other cases, a defective component may cause a loss of performance that could potentially be mitigated through the use of spare components implemented in an FPGA. In this paper we investigate both types of situations under different FPGA latencies. First, we investigate the case where the only multiplier or the only integer ALU present in the processor must be repaired. We follow that by the case where only one of several ALUs has been damaged.

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

TABLE I
Baseline processor configuration.

All our experiments analyzed the effect of repair on the performance of programs from the SPEC95 [16] integer benchmark suite using the SimpleScalar [17] simulator. The SimpleScalar out-of-order processor model (sim-outorder) is an execution-driven simulation engine that reproduces the

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 AVERAGE	-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 AVERAGE	-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 II
UNPIPELINED FPGA ALUS (OR MULTIPLIER) REPLACING ONLY ASIC INTEGER ALU (OR MULTIPLIER)

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 I. Note that depending on the particular experiment, the number of integer ALUs in the non-defective machine may be equal to 1 or 3.

In each of our experiments, we allowed the speed discrepancy between the original functional unit and the functional unit in the FPGA to vary between 2 and 4. For example, in experiments where the latency of the FPGA is 4, it is assumed to take four times as many clock cycles for the calculation to complete and be made available when the calculation is performed in the FPGA instead of the ASIC. We also varied the number of copies of the defective functional unit from 1 to 4 in an attempt to hide the additional latency while maintaining throughput, as described in Section III-B2.

A. 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 are shown in the top portion of Table II. This table shows the percentage change in IPC (Instructions Per Cycle) 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 2-4 times the ASIC ALU latency.

As is clear from the table, 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 in the FPGA take twice as long as the original (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. However, note that the alternative to the reduced performance is a completely non-functioning chip. Now, if we increase the number of FPGA ALUs appropriately, we can entirely negate the performance loss. For example, 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 FPGAs with latencies of 3 or 4. Unsurprisingly, the number of copies we need in the FPGA increases as the latency increases.

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

TABLE III
PERCENTAGES OF INSTRUCTIONS THAT USE THE INTEGER ALU AND THE INTEGER MULTIPLIER.

We next performed similar experiments using the integer multiplier. The instruction profile in Table III 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 II. (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.

B. Performance impact when repair is optional

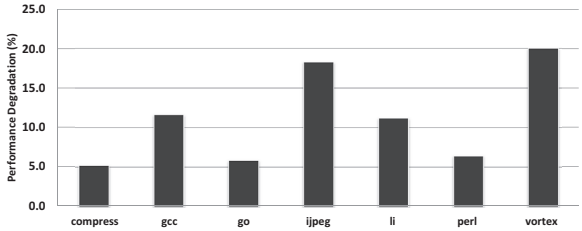

Fig. 3. Performance degradation with the loss of 1 integer ALU when no repair is performed.

Next, we consider the case where there are 3 integer ALUs in the ASIC, one of which is defective. Figure 3 shows the percent performance degradation (measured in Instructions per cycle (IPC)). Because no repair of the bad ALU is

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 IV
PERCENTAGE IMPROVEMENT IN PERFORMANCE WHEN THE BAD INTEGER ALU IS REPAIRED WITH 1–4 PIPELINED INTEGER ALU OF LATENCY 2–4 ON THE FPGA. THE IMPROVEMENT IS OVER THE *No Repair* CASE.

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

TABLE V
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.

performed, the system has only two usable integer ALUs. 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.) We measured the performance gains of the processor when different numbers of FPGAs are used for repair *over the non-repair case*. The results are shown in Table IV.

On average, we see performance improvement of approximately 8.9% when the the FPGA ALU has a latency of 2 *over the no-repair case*. 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 is too high, a single 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 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 large effect.

From the results of the pipelined experiments shown in Table IV, 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 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 V. Because 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, we see that adding 3 and 4 FPGAs can have a positive impact on the performance. The processor can now take advantage of the fifth or the sixth additional ALU (which might not be busy) and issue a ready instruction present on the issue queue. The gains from the availability of these additional ALUs makes the unpipelined case approximately match the peak performance gain of the pipelined case. In many cases, the final improvement of the repaired version allows it to closely reach the performance of the non-defective version if enough FPGA ALUs are added.

C. Overhead

We also wanted to estimate the TSV overhead required for replacement of an ALU. The addition of ALUs in the FPGA

TSV parameters	2011–2014	2015–2018
Min. diameter(μm)	4–8	2–4
Min. pitch(μm)	8–16	4–8
Min. depth(μm)	20–50	20–50
Max. aspect ratio	5:1–10:1	10:1–20:1
# of tiers	2–3	2–4

TABLE VI
GLOBAL 3D INTERCONNECT ROADMAP [18].

for repair requires that the output port of the issue queue be connected to the inputs of the ALU on the FPGA layer via a TSV to transfer the operands, operation identifier etc. TSVs also are needed to return the operation results and flags from the FPGA ALU back to the reorder buffer on the ASIC for an in-order commit. We estimated that a single issue/commit of an instruction to/from an ALU will need: 32 bits for operand A, 32 bits for operand B, 16 bits for operation identifier and other control signals to the ALU *(we believe this is an overestimate)*, 32 bits for the operation result, 8 bits for the operation and function-unit availability flags and 1 bit for the clock signal. This brings the total number of signals to be transferred between the ASIC and FPGA layer to 121. Assuming that the TSVs are laid out on a NxN array bundle, adding 1 FPGA ALU would require that we have 121 TSVs arranged in an 11x11 array. Similarly, for adding 4 FPGA ALUs the number of TSVs is 484 (requiring an array of 22x22 TSVs). The ITRS roadmap for TSVs is shown in Table VI. Using the diameter, d, and the pitch, p, of TSVs for the near term (2011–2014) from the table, we estimate the area for the different TSV arrays and present the results in Figure 4. For a 22x22 TSV array resulting from the addition of 4 FPGA ALUs, we estimate the area to be 69,696μm^2 (about 0.28% of a 25mm^2 die size) using the lower range of diameter and pitch and 278,784μm^2 (about 1.12% of a 25mm^2 die size) using the upper range. Based on the ITRS roadmap for year 2018, this area overhead would be as little as 0.07% of a 25mm^2 die size. The area overhead shown in Figure 4 does not include the area of the buffers and additional circuitry needed for TSVs. The added overhead associated with the buffers would be significantly lower compared to the overhead of the TSV bundle as the TSV cut size is about 5–10 times the height of standard cells used for logic in 32nm technology [19].

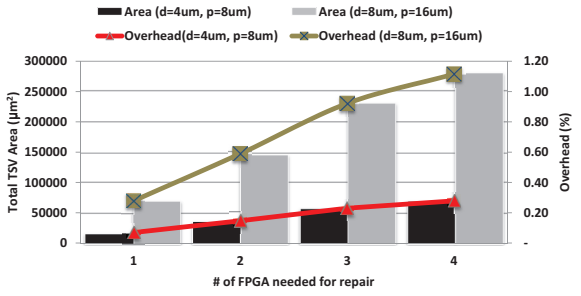

Fig. 4. Estimate of the TSV area needed to carry signals from the ASIC to the FPGA layer.

V. CONCLUSION

We propose harnessing resources within an FPGA already present in a 3D stack for bypass and replacement of defective portions of the logic in other dies in the stack. We have investigated cases where repair was mandatory and when repair can mitigate the corresponding performance loss. Maximizing this mitigation depends on the speed differential between the ASIC and the FPGA, whether the units are pipelined or not, how many copies of the faulty hardware are realized in the FPGA, and how often the affected functional unit will be used. However, with appropriate choices, we can often hide almost the entire performance hit due to a defective component. Future work will include a more extensive analysis of the complexity and the cost of implementation with the FPGA.

ACKNOWLEDGMENT

This work was supported in part by NSF under Grants CCF-0915302, CCF-1110290, CCF-1061164, and CCF-1205176.

REFERENCES

[1] A. Crouch and J. Dworak, "What is 3-D test and how do IEEE standards help?" *Electronic Device Failure Analysis*, vol. 13, no. 4, pp. 4–13, 2011.
[2] E. Marinissen and Y. Zorian, "Testing 3D chips containing through-silicon vias," in *Int. Test Conference (ITC)*. IEEE, 2009, pp. 1–11.
[3] M. Potkonjak, L. Guerra, and J. Rabaey, "Heterogeneous BISR techniques for yield and reliability enhancement using high level synthesis transformations," in *Intl. Conf. on Application-Specific Array Processors*, Oct 1993, pp. 454 –465.
[4] C.-S. Hou, J.-F. Li, and T.-W. Tseng, "Memory built-in self-repair planning framework for rams in socs," *IEEE Tran. Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 11, pp. 1731 –1743, nov. 2011.
[5] S.-K. Lu, C.-L. Yang, Y.-C. Hsiao, and C.-W. Wu, "Efficient BISR techniques for embedded memories considering cluster faults," *IEEE Tran. Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 2, pp. 184 –193, feb. 2010.
[6] L. Jiang, R. Ye, and Q. Xu, "Yield enhancement for 3d-stacked memory by redundancy sharing across dies," in *IEEE/ACM Int. Conf. on Computer-Aided Design*, nov. 2010, pp. 230 –234.
[7] L. Jiang, Q. Xu, and B. Eklow, "On effective TSV repair for 3D-stacked ICs," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2012, pp. 793–798.
[8] V. Pasca, L. Anghel, M. Nicolaidis, and M. Benabdenbi, "Csl: Configurable fault tolerant serial links for inter-die communication in 3d systems," *Journal of Electronic Testing*, pp. 1–14, 2012.
[9] S. Mitra, W.-J. Huang, N. Saxena, S.-Y. Yu, and E. McCluskey, "Reconfigurable architecture for autonomous self-repair," *Design Test of Computers, IEEE*, vol. 21, no. 3, pp. 228 – 240, may-june 2004.
[10] M. Psarakis and A. Apostolakis, "Fault tolerant FPGA processor based on runtime reconfigurable modules," in *IEEE European Test Symp.*, May 2012, pp. 1 –6.
[11] J. Emmert, C. Stroud, and M. Abramovici, "Online fault tolerance for FPGA logic blocks," *IEEE Tran. Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 2, pp. 216 –226, feb. 2007.
[12] S. Di Carlo, A. Miele, P. Prinetto, and A. Trapanese, "Microprocessor fault-tolerance via on-the-fly partial reconfiguration," in *IEEE European Test Symposium (ETS)*, may 2010, pp. 201 –206.
[13] K. Schleupen, S. Lelaich, R. Mannion, Z. Guo, W. Najjar, and F. Vahid, "Dynamic partial fpga reconfiguration in a prototype microprocessor system," in *Int. Conf. on Field Programmable Logic and Applications (FPL)*, aug. 2007, pp. 533 –536.
[14] P. Dorsey, "Xilinx stacked silicon interconnect technology delivers breakthrough fpga capacity, bandwidth, and power efficiency," *Xilinx White Paper: Virtex-7 FPGAs*, pp. 1–10, 2010.
[15] P. Clark, "Achronix reveals 22-nm FPGAs, courtesy of Intel," *EE Times*, April 2012. [Online]. Available: http://www.eetimes.com/electronics-news/4371563/Achronix-22-nm-FPGAs-Intel-process
[16] "SPEC95 benchmark suite http://www.spec.org/cpu95."
[17] T. Austin, E. Larson, and D. Ernst, "Simplescalar: An infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, Feb. 2002.
[18] "The International Technology Roadmap for Semiconductors: 2011," http://www.itrs.net.
[19] V. Gerousis, "Physical design implementation for 3D IC: methodology and tools," in *Intl. symp. on Physical design (ISPD)*, 2010, pp. 57–57.