Automatic BRAM Testing for Robust Dynamic Voltage Scaling for FPGAs

Ibrahim Ahmed, Shuze Zhao, James Meijers, Olivier Trescases and Vaughn Betz
Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada
Email:{ibrahim, szhao, ot, vaughn}@ece.utoronto.ca

***Abstract*—Recently FPGA researchers have proposed different approaches to enable dynamic voltage scaling (DVS) for FPGAs. While the proposed approaches have shown that DVS is able to significantly reduce FPGA power consumption, most of these solutions were developed only for the soft fabric of the FPGA and hence cannot be deployed for applications that use the FPGA hard blocks such as block RAMs (BRAMs). In this work, we extend a previously proposed offline calibration-based DVS approach to enable DVS for FPGAs with BRAMs; we build testing circuitry to ensure that all used BRAM cells operate safely while scaling V_{dd}, and we develop testing procedures that are able to measure the delay of timing paths that start or end at BRAMs. We extend the CAD tool FRoC to automatically generate calibration designs with BRAM testers along with soft fabric testers to measure the actual F_{max} of each application on any chip under different operating conditions; this information is stored in a calibration table that is then used when the application is running to scale V_{dd} to the minimum value that guarantees safe operation at the desired speed. Using our proposed solution, we show that we can run a discrete Fourier transform core with 32% and 46% power reduction compared to the conventional fixed-voltage operation at the reported F_{max} and at a lower clock frequency, respectively.**

I. Introduction

Moore's law [1] and Dennard scaling [2] have enabled the semiconductor industry to continue shrinking the size of transistors while making them faster and more power efficient. Dennard scaling reduces the supply voltage (V_{dd}) with the transistor size such that the electric field remains constant. This has allowed the industry to increase the density of transistors, reduce gate delays and maintain a constant switching power density with each successive technology node. Unfortunately, below 90 nm, V_{dd} scaling has significantly slowed [3] due to leakage-current-imposed constraints on the transistor threshold voltage (V_{th}). Fig. 1 plots the supply voltage and the number of equivalent logic elements (LE) per chip for FPGAs manufactured by Altera/Intel using different technology nodes. The figure shows that while the number of LEs has been rapidly growing, the supply voltage has not been scaling in recent technology nodes. These two trends add significant challenges for designers to meet power budgets.

Dynamic voltage scaling (DVS) has been recently proposed as a solution to the power consumption challenges facing FPGAs [4]–[7]. With DVS, the supply voltage of each FPGA chip is scaled, during online operation, to the minimum voltage that guarantees safe operation at the desired speed (clock frequency). Since dynamic power is proportional to V_{dd}^2 and static power drops even faster with V_{dd} [8], a small reduction in V_{dd} results in significant power savings. Other compute chips such as commercial CPUs [9], [10] have been using DVS, and the work presented in [9] shows that running an IBM POWER7 processor with DVS results in a 24% total power reduction compared to operating at a fixed voltage. Unfortunately, while

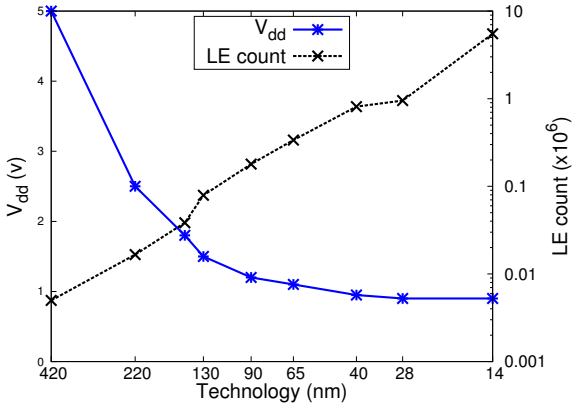

Fig. 1. Nominal supply voltage and equivalent LE count of FPGAs with different technology nodes.

academic works on DVS for FPGAs have shown significant power savings of around 30% total power reduction compared to fixed-voltage operation [4]–[6], commercial FPGA systems have not deployed DVS yet. One of the main barriers to industry adoption of the academically proposed DVS solutions is that most of these solutions were developed for circuits that only use the FPGA soft fabric. Since nearly all industrial designs use the FPGA hard blocks such as the block random access memories (BRAMs), a commercial DVS solution must take them into consideration.

Any DVS solution must be able to identify the minimum supply voltage that guarantees an error-free operation at the requested clock frequency. Ahmed *et al.* [11] proposed exploiting the programmable nature of FPGAs by using an offline design-specific per-chip calibration approach to identify this voltage. They developed a CAD tool (FRoC) that uses a commercial CAD tool (Quartus) to identify timing critical paths of an application; FRoC then generates a calibration bitstream that contains these timing critical paths, testing circuitry and heaters to test the timing critical paths for delay faults at different temperatures. The output of the calibration process is a calibration table that stores the minimum safe voltage at different speeds (clock frequencies) and temperatures. The information stored in the calibration table is used in a closed-loop feedback system to scale the supply voltage while the application is running according to the chip temperature and desired speed. However, their testing procedure used to generate the calibration table assumes that the source and sink registers of all timing paths have observable inputs and outputs. This assumption is true for registers in the soft fabric of an FPGA but it does not hold for registers in hard blocks (DSPs and BRAMs). In this work, we build upon their approach to achieve a DVS solution that can support any application using BRAMs. We focus on BRAMs as they are different than

other components on the FPGA due to their analog nature and so scaling V_{dd} may result in various types of failures, such as parametric failures, in weaker SRAM cells within the BRAM. Moreover, several previous studies [12]–[14] and our own experiments have shown that in many applications the most timing critical paths either start or end at BRAMs and hence it is critical to measure the delay of these paths. Our new DVS solution not only ensures that all cells in all BRAMs used by the application do not suffer from any parametric failures during voltage scaling but also robustly and automatically tests for delay faults in the BRAM interface logic. In this work we:

- design and implement automated testers that check for parametric failures in FPGA BRAMs,
- measure FPGA BRAM performance across different modes and chips as we scale V_{dd},
- automatically test delay faults in paths that are part of the BRAM interface logic, and
- integrate BRAM testing with logic testing to automatically generate calibration bitstreams that enable DVS for applications using BRAMs.

II. Background

A. DVS for FPGAs

As mentioned earlier, a fundamental task for any DVS solution is mapping the requested clock frequency to the minimum supply voltage that guarantees safe operation at this frequency on a specific chip. Accordingly, previous research dealing with DVS for FPGA can be divided into two groups: the first group [6], [15] identifies this safe voltage during online operation, and the second group [7], [11] performs this task offline and stores information to scale V_{dd} accordingly.

The ideas presented in [6], [15] are based on attaching shadow registers to the inputs of timing critical registers in the application. In [6], the shadow registers are clocked with a different clock than the application registers such that a delay fault due to a low V_{dd} would occur at the shadow registers before the application registers. In [15], the shadow registers and application registers share the same clock but the shadow registers are placed such that the delay to the input of the shadow registers is longer than the delay to the application register; this also ensures that a delay fault occurs at the shadow registers before the application registers. In both solutions, V_{dd} is scaled down until some shadow register reports a failure, indicating that the timing slack is getting too low so the system stops reducing V_{dd}. One downside of online monitoring is that it adds area and power overhead while the application is running. A major limitation of these online solutions is that they assume that the inputs and outputs of the timing critical registers are observable; this assumption is not valid for hard blocks and so these solutions cannot monitor the registers buried inside BRAMs or DSPs.

In [16], Nunez *et al.* presented an interesting solution to monitor registers that are buried in hard blocks. They create a register (main register) in the soft fabric for each timing critical register in a hard block. This main register acts as a copy of the buried register and is clocked by a different clock ($clk2$) than the buried register clock ($clk1$). $clk2$ leads $clk1$ by T_x seconds, where T_x is calculated as the difference in delay between the path ending at the main register and the path ending at the buried register. Ideally, this phase shift would compensate for the delay difference between the path ending at the buried register and the path ending at the main register and thus allows the main register to act as a proxy for the buried register. However, there are no guarantees on how well this phase offset matches the delay difference between the paths as V_{dd} is scaled, so we cannot ensure that the timing path to the main register will always fail before the timing path to the buried register. Moreover, this solution cannot handle timing paths that start and end in the same hard block with no connection to the soft fabric of the FPGA, such as paths that start at the address port of a BRAM and end at the output registers of the same BRAM.

The FRoC CAD tool presented in [11], instead takes an offline calibration-based approach. It identifies the minimum safe V_{dd} at each frequency by measuring the delay of the most timing-critical paths on each chip the application runs on. To measure the delay of a path, FRoC replicates all the resources used by this path, changes the LUT masks of all tested LUTs, and adds control signals to sensitize these paths so they can be tested for delay faults. To minimize calibration time, FRoC also groups paths that can be tested in parallel into a test phase, and it generates a main test controller that sequentially loops over the different test phases. At each test phase, the controller provides input stimulus, sets the control signals, and checks the output registers of all currently tested paths. The controller toggles the input of the source register of each tested path and checks that in the next clock cycle the output of the sink register of this path also toggles, indicating that this path can safely operate at the corresponding V_{dd} and clock frequency. The controller repeats this testing at different V_{dd} values and clock frequencies, and stores the minimum safe voltage for each clock frequency in a calibration table that is later used to guide DVS while the application is running. This testing approach works well for timing paths that only use the soft fabric of the FPGA, but it cannot test paths that use hard blocks such as BRAMs as they have buried registers that cannot be observed from the soft fabric.

B. BRAM Potential Failures with DVS

To realize a DVS solution that is able to handle applications that use BRAMs, we first need to analyze the effects of scaling V_{dd} on BRAMs. We divide the BRAM potential failures with DVS into (1) parametric failures and (2) timing failures in paths that start and/or end at a BRAM. This section explains the two different types of failures.

Transistors in an SRAM cell are sized to allow for successful read and write operations on all SRAM cells [17]. However, a mismatch in the strength of these transistors could result in parametric failures [18]. A Parametric failure can manifest as [19]:

- hold retention failure: flipping of the contents of an SRAM cell during standby due to a low V_{dd},
- destructive read failure: flipping of the contents of an SRAM cell during a read operation (occurs when the noise from the capacitive bitline is higher than the tripping point of the SRAM cell inverter),
- read access time failure: inability to read the contents of an SRAM cell (occurs when the voltage difference between the two bitlines stays lower than the sense amplifier threshold during a read operation)
- write access time failure: inability to write to an SRAM cell (occurs when the 1 value stored in an SRAM cell cannot be discharged to reach the tripping point of the inverter during the write operation)

These failures are caused by variations in the device (PVT) parameters and are thus named parametric failures [19]. DVS in

commercial CPUs is limited to a minimum voltage V_{min} [20] that is usually set by parametric failures in the SRAM-based caches. Since FPGA BRAMs are also SRAM-based, it is important to check for these failures in any DVS solution that targets applications with BRAMs. To our knowledge, no prior proposed DVS solution for FPGAs has accounted for SRAM parametric failures.

The other type of BRAM failure that can occur when scaling V_{dd} is a timing failure in paths that are part of the memory interface logic; these paths have either the source or the sink register inside a BRAM. To support DVS on applications using BRAMs, our DVS solution needs to measure the delay of paths in the memory interface logic to ensure that we do not scale V_{dd} to a value that results in timing failures in these paths. In the FPGA academic community, there have been interesting works on delay measurement and delay fault testing [21]–[24]. However, to the best of our knowledge no prior work attempted to test delay faults in the BRAM interface logic, and testing these faults presents unique challenges due to limited observability.

III. Testing BRAM Parametric Failures

A. Testing Methodology

The fundamental operations required to test parametric failures include: writing arbitrary values to the memory, reading data from the memory, and checking if the read data matches the expected value. Parametric failures will occur first in the weakest cell of the BRAM, so it is important to test all cells of a BRAM to identify all possible parametric failures. We also need to check for parametric failures at different V_{dd} to ensure the correct operation of all used BRAMs during voltage scaling. Since this testing should be performed automatically on each FPGA by the DVS calibration, we need the testing circuitry to test all BRAMs in a small number of calibration bitstreams (ideally one), and to not use external test equipment.

To test all BRAMs in a single bitstream, we designed a light-weight memory tester that can be attached to each BRAM to perform the basic write, read and check operations. These memory testers are also connected to a central controller that orchestrates the entire testing operation. Fig. 2 shows the overall testing circuitry along with a reconfigurable PLL that is used to sweep the frequency when testing for access failures. Each memory tester accepts a read or write (r/w) command from the central controller. When the write command signal is asserted, the tester writes the given data input to all even memory addresses and the inverse of the given data input to odd addresses, and when the read command is asserted, the tester reads the data in all addresses and checks that it matches the written data. Each tester asserts an error signal for every word that does not match the expected value. This error signal is connected to the central controller through an SR-latch. The error signal sets the output of this latch and it is only reset from the central controller. An SR-latch is used here as the memory tester runs at a high and variable frequency while the central controller runs at a fixed low clock frequency. Dividing the testing circuitry into memory testers and a central controller allowed us to easily test different types of failures by only changing the central controller. Moreover, it allowed us to build smaller memory testers by refactoring common tasks from the memory testers into the central controller; these tasks include sweeping the clock frequency and waiting between write/read operations to/from the BRAM.

To check for hold retention failures, the central controller:

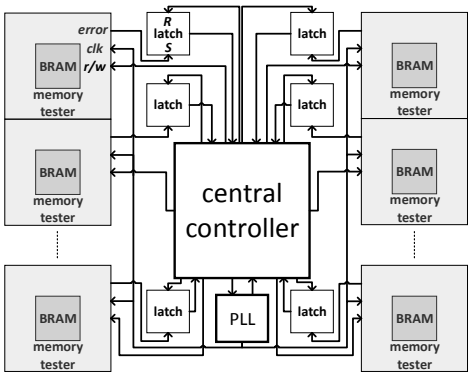

Fig. 2. Overview of the testing circuitry designed to test parametric failures.

issues a write command to all BRAMs (at nominal V_{dd}), reduces V_{dd} to a lower voltage (V_{low}), waits for a certain time, brings V_{dd} back to nominal value, issues a read command to all BRAMs and monitors the error signal from all memory testers. These steps are repeated with a lower V_{low} until the minimum V_{low} that avoids hold retention failures is identified.

To check for destructive read failures, the central controller performs the same steps as described for the hold retention failure but instead of being idle at the low V_{dd}, the central controller issues multiple read commands at the low V_{dd}. Read operations at the low voltage stress the BRAM to identify the minimum voltage that does not show any destructive read failures.

Tests for read and write access time failures are more complex as they require sweeping the clock frequency to identify the maximum clock frequency for safe read (read F_{max}) and write operations (write F_{max}) at each V_{dd}. To check for read access time failures, the central controller:

1) reconfigures the PLL to generate a safe low clock frequency;
2) issues a write command to all memory testers and waits for all writes to finish;
3) reconfigures the PLL to increment the clock frequency;
4) issues a read command and monitors the error latches from all memory testers;
5) resets the error latches, and either stops if BRAMs failed, or goes to step 3 if BRAMs haven't failed.

Steps 1 and 2 ensure that the initial write operations to all BRAMs are safe as they are done at a low clock frequency. After the write operations are completed, read commands are issued at higher frequencies to detect the read F_{max} of each BRAM that does not result in any read access time failures. This procedure is repeated at different V_{dd} to identify the read F_{max} at each V_{dd}.

The steps for testing write access time failures are similar to those for read access time failures; however, write access testing includes a read operation that needs to be performed under safe conditions to avoid missing a write access failure due to the presence of a read access failure. To check for write access failures, the central controller:

1) reconfigures the PLL to generate a write clock frequency (F_{write}), initially set to a safe low value;
2) issues a command to all memory testers to write a specific data pattern (D) and waits for all writes to finish;
3) reconfigures the PLL to generate a safe low read clock frequency (F_{read});

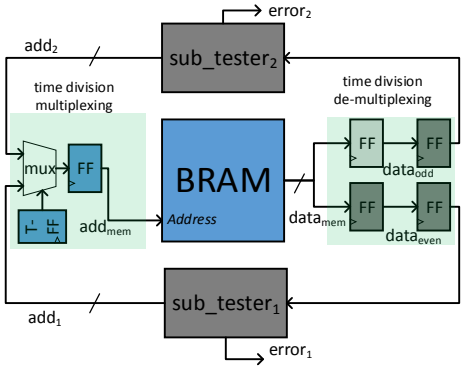

Fig. 3. The internal structure of the memory tester.

4) issues a read command and monitors the error latches from all memory testers;
5) If the BRAMs failed then it stops. Otherwise it resets the error latches, inverts the data pattern D ($D = \sim D$), reconfigures the PLL to increment the write clock frequency (F_{write}) and goes to step 2.

To guarantee that all read operations are safe, the central controller lowers the clock frequency before issuing the read command. The central controller inverts the data pattern written to the memory at each clock frequency to ensure that each successful write operation inverts the previously stored value, which is important to capture any write access failures. Similarly to the read access testing, this procedure allows us to identify the write F_{max} of each BRAM at different supply voltages.

B. At-speed Testing Challenges

The testing methodology explained earlier assumes that the memory testers fail at a higher clock frequency than the BRAM. To be able to perform at-speed testing of the FPGA BRAMs, we need to ensure that as we are sweeping the clock frequency no delay faults occur in the testing circuitry. Since the FPGA soft fabric is much slower than the hard BRAM blocks, building memory testers in the soft logic that are able to run at a higher clock frequency than the BRAMs is a challenging task. To achieve this goal, we built parallel testers that multi-pump the BRAM and we explored different compilation techniques to generate a placed and routed circuit that meets our timing requirements.

1) Parallel Testers: We divided our memory tester into two sub-testers that run at half the clock frequency of the BRAM and together they feed the BRAM with the inputs required for testing access failures. Fig. 3 shows how the two sub-testers are connected to the BRAM, where blocks colored in blue run at the same fast clock frequency as the BRAM (clk_{fast}), blocks in dark grey run at half the fast clock frequency ($clk_{slowEven}$), and blocks in light grey also run at half the fast clock frequency ($clk_{slowOdd}$) with a 180 degrees phase shift relative to $clk_{slowEven}$. All clocks are generated from the same PLL shown in Fig. 2. Fig. 4 shows the timing diagram of these clocks and the signals connecting the sub-testers to the BRAM. add_1 and add_2 are the address signals from sub_tester_1 and sub_tester_2, respectively. Since the sub-testers run at half the clock frequency, each sub-tester only needs to provide half the addresses as shown in the waveforms of add_1 and add_2. The multiplexer (mux), toggle flip-flop (T-FF) and the flip-flop (FF) running at the fast clock frequency form a time division multiplexing circuit that provides the BRAM with all addresses (add_{mem}) at the fast clock frequency. The output

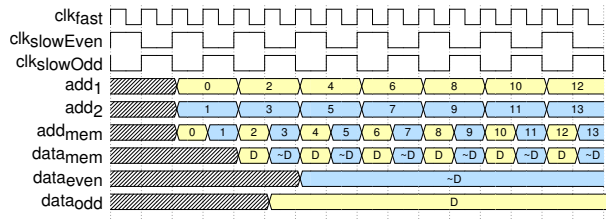

Fig. 4. Timing diagram of the memory tester internal signals.

data from the memory ($data_{mem}$) is time de-multiplexed into two signals $data_{odd}$ and $data_{even}$ as shown in Fig. 4. The wave diagram in Fig. 4 assumes that the tester successfully wrote a specific word (D) in even addresses and the inverse of D in the odd addresses. Each de-multiplexed data signal is fed to a different tester that checks if the read data matches the previously written data.

2) Iterative Compilation: Dividing the memory tester into parallel sub-testers allows us to clock the sub-testers with half the BRAM clock frequency while providing at-speed testing to the BRAMs. However, there are still timing paths inside the time division multiplexing and de-multiplexing circuitry that need to run at the same clock frequency as the BRAM. To safely test the BRAMs, we must ensure that these timing paths fail at a higher frequency than the BRAM. When we instantiate the designed memory tester to test a single BRAM, the CAD tool (Quartus) is able to place and route the design such that all paths within the time division multiplexing and de-multiplexing circuitry fail at a higher frequency than the BRAM. However, when we instantiated many testers to test all BRAMs, the placed and routed circuit clock frequency was limited by the tester and so it was not safe to test BRAM access time failures. Motivated by the fact that Quartus was able to generate fast time division multiplexing and de-multiplexing circuits when only one tester was instantiated, we explored different compilation techniques to achieve similar results with many testers.

Fig. 5 shows the Quartus-reported tester F_{max} and logic utilization for the circuit with a single tester, the circuit with all testers required to test all BRAMs using the different compilation techniques, and the Quartus-reported BRAM F_{max}. We report the soft logic flipflop, look-up table (LUT) and logic element (LE) utilization. These results were obtained when targeting a Cyclone IV EP4CE115F29C7 FPGA that has 432 BRAMs. One BRAM was used to reconfigure the PLL and another one was used to send testing results to a host PC, so we instantiated 430 testers to test all the remaining BRAMs. Fig. 5 shows that for the single-tester case, the reported F_{max} (594 MHz) is much higher than the BRAM F_{max} (314 MHz), but it drops when we instantiate all the required testers. We first performed a conventional flat compilation in which Quartus flattens the entire circuit into a single netlist and implements it. Since the algorithms used by Quartus are heuristics, as the design gets larger the quality of results can become sub-optimal as shown in Fig. 5, where the tester F_{max} dropped by more than 50% for the 430-tester case compared to the single-tester case. We noticed that Quartus was trying to save area by sharing logic across different testers, which resulted in the huge frequency drop. To prevent Quartus from performing any circuit optimizations across the different tester instances, we partition our design such that each tester and BRAM instance are in a design partition. This resulted in a small increase in logic utilization because Quartus now cannot share logic between the different testers. However, it also increased

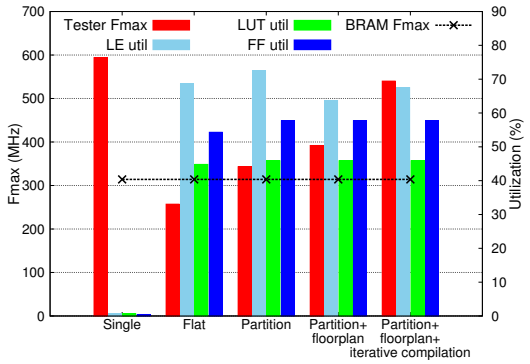

Fig. 5. F_{max} and logic utilization of the testing circuitry using different compilation techniques.

the testers' F_{max} to 343 MHz compared to 257 MHz for the flat compilation. To achieve better results, we manually floorplanned our design. In our floorplan, we assign a fixed-size region around each BRAM that is exclusively assigned to this BRAM's tester so no other module can be placed in this region. After floorplanning, the flipflop and LUT utilization remained the same but LE count decreased from 72% to 63%; Quartus needs to place each partition in its assigned region, so it packs the circuit more densely. With partitioning and floorplanning, we were able to achieve a tester F_{max} of 392 MHz, which is still much worse than the single-tester case. By inspecting the detailed timing report, we notice that testers in some partitions are almost as fast as the single-tester case while others are much slower. We decided to perform iterative compilation to let Quartus focus only on optimizing the slow partitions instead of the whole design. Our automated iterative compilation flow:

1) compiles all partitions,
2) sets all partitions to use the post-fit netlist,
3) sets the slowest N partitions that did not meet our timing constraints to use the post-synthesis netlist,
4) if all partitions meet timing constraints then stop, else go to step 1.

Setting a partition to post-fit means that the placement and routing of this partition will be saved and used again in the next compilation, while setting it to post-synthesis means that the compiler can change its placement and routing when it is recompiled. By setting all partitions except the worst N that do not meet timing to post fit, we guide the compiler to optimize these N partitions more and hence we reach a higher F_{max} on these partitions. Once all partitions meet the timing constraints we stop the iterative compilation. We experimented with different values for N and 10 gave a good compromise between runtime and quality of results. With the iterative compilation flow for the 430-testers case (utilizing 64% of the FPGA), the worst case tester F_{max} reached $\sim$ 540 MHz, only 9% lower than the single-tester case that utilized less than 1% of the FPGA. This allowed us to perform at-speed testing for access failures of all BRAMs using a single bitstream. While we have developed the iterative compilation flow to generate faster memory testers, we believe that the it can be generalized and used to close timing in other large modular designs.

C. Parametric Failure Results

We used our proposed testing circuitry to test for BRAM parametric failures on seven different DE2-115 boards equipped with a Cyclone IV EP4CE115F29C7 FPGA that has a 1.2 V nominal supply voltage; we also modified the board to power the FPGA using a variable voltage power supply. Our testing circuitry communicates with a PC over the RS-232 serial port available on the DE2-115 board. Using this setup we tested for all four types of parametric failures across a voltage range from 0.8 V to 1.2 V in 0.1 V steps. Below 0.8 V the FPGA would reset itself and so we couldn't perform any testing at a lower V_{dd}. In our experiments, we tested the BRAMs in different modes (single port and dual port); in different depths and widths; and we also tested them by writing and reading different data patterns. We also repeated each test multiple times to validate that the results are reproducible.

Interestingly, on all seven FPGAs we did not encounter any hold retention or destructive read failures across the tested V_{dd} range. While this observation means that for these seven FPGAs we can lower V_{dd} without corrupting the contents of the BRAMs, we still should (and do) test for these failures as part of our DVS calibration to ensure that this is the case for the specific chip that the application is running on.

Testing for access time failures requires sweeping the clock frequency. To achieve this without using any external equipment, our testing circuitry dynamically reconfigures the on-chip PLL to sweep the clock frequency from 125 MHz to 625 MHz in 1.25 MHz steps. It should be noted that to prevent the FPGA from drawing a high current we do not test all BRAMs in parallel, but rather our central controller tests each BRAM in isolation and then moves to the next BRAM. With this testing technique, the FPGA does not draw high current and thus IR-drop effects on the measurement results are reduced. Fig. 6 shows the average measured read and write F_{max} for all BRAMs. Results shown in this figure are based on BRAMs used in single port mode with 1024 8-bit words. The figure shows that read and write F_{max} have a linear trend with V_{dd}, and they also have a wide range across V_{dd} where the average read and write F_{max} at 1.2 V are almost three times higher than at 0.8 V. The figure also shows how pessimistic the Quartus-reported F_{max} is; we can run the BRAM at the Quartus-reported F_{max} with a 1 V supply instead of the nominal 1.2 V. At 1.2 V the difference between the fastest and the slowest BRAM write F_{max} is 15% and at 0.8 V the difference is 20%. These results show that at lower V_{dd} the effects of PVT variation are higher. While these results are for single port BRAMs configured with 1024 8-bit words, different widths and depths show similar trends, with narrower BRAMs having a slightly higher read and write F_{max} on average. We have also tested BRAMs in simple and true dual port modes and we have observed trends similar to the results of Fig. 6 .

The BRAMs on the chips we tested are organized into six columns spread over the FPGA. Fig. 7 shows the heat map of the measured read F_{max} at nominal supply voltage for all BRAMs from seven different FPGAs (F1-F7). Instead of showing results on separate figures we shifted the results of the n^{th} FPGA n columns to the right, so each group of seven adjacent coloured columns represents the results of the same column of BRAMs but from different FPGAs. White columns represent the soft fabric and DSP columns, and the white rows in the first and third group of coloured columns are the BRAMs used for reconfiguring the PLL and communicating with the PC. Fig. 7 shows that there is a die-to-die variation component as evidenced by the fact that F1 BRAMs (left most column) are consistently faster than all other FPGAs. A random variation component can also be seen along with a spatial correlation trend of slower BRAMs in the center of the FPGA. Analyzing

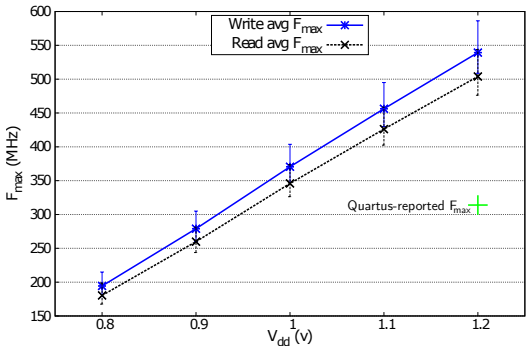

Fig. 6. Measured arithmetic average read and write F_{max} with error bars showing the minimum and maximum F_{max} for 430 BRAMs x 7 chips.

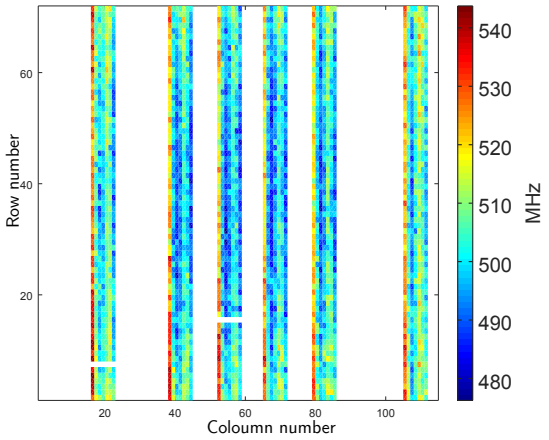

Fig. 7. Heatmap of the measured read F_{max} of all BRAMs on 7 different chips running at nominal voltage.

the measured F_{max} of all BRAMs, we found that the read F_{max} σ/μ does not vary much under different V_{dd} and is approximately 2.3%. However, the write F_{max} σ/μ increases from 1.9% at 1.2 V to 2.9% at 0.8 V.

To the authors' knowledge, this is the first work that presents testing circuitry that is able to measure FPGA BRAM performance without external equipment. We are using these automatically created testers to enable safe DVS, but they can also be of assistance to other research. They can be used to measure BRAM performance on more chips and, using the measurement results, a detailed variation model can be built, or the results can be used to support component specific mapping explorations [25].

IV. Testing BRAM Interface Logic

In our calibration procedure, after checking for parametric failures, we need to ensure that paths starting or ending at BRAMs do not suffer from any timing failures during V_{dd} scaling; measuring the delay of these paths is different than measuring the delay of paths that start and terminate in the soft fabric as we do not have observability or direct control of registers buried in BRAMs. The main test controller in the calibration bitstreams generated by FRoC in [11] can set the control signals of LUTs to ensure that the tested path is sensitized, but if the path starts at a BRAM, it cannot exercise it, and if the path ends at a BRAM, it cannot check if a delay fault occurred or not. In this work, we extend FRoC to generate different memory controllers that can perform these tasks and integrate these memory controllers with the main test controller to produce calibration bitstreams that can test both logic paths and paths in the BRAM interface logic. FPGA BRAMs only support synchronous read and write operations which means that their input ports are internally registered, but registering BRAM outputs is optional. A timing path can start at a BRAM from the output data port (q) or it can end at a BRAM at different ports such as: write enable (we), input data (d) and address port (a).

The main steps performed by all our memory controllers are: initializing the BRAM to a known state, activating the delay fault of the tested path, and propagating the fault to the main test controller. Since different BRAM ports have different functionality we designed different memory controllers to perform these steps for each port. We illustrate these steps by describing the different operations performed by the memory controller responsible for testing timing paths ending at a *write-only* address port. A *write-only* address port cannot read the contents of the BRAM, but can only be used to write data to it. Write-only ports are present in BRAMs configured as simple dual port, where one port is used for writing and the other port is used for reading; this configuration mode is used in many circuits as it implements FIFOs.

To test a timing path ending at a write-only address port (a_x) we need to perform a write operation at a specific address, use the other BRAM port to read the content of this address, and check that it matches the written data. To perform these operations, our memory controller needs to control the write enable (we_x) input of the tested port, the input data (d_x) of the tested port, and the address (a_y) of the other port that can read. Fig. 8 shows an example of the generated test circuitry that tests paths ending at a write-only port, with the paths under test highlighted in red. The paths under test are replicated from the application and so they have a fixed placement and routing that exactly matches the application. However, the remaining parts can be placed and routed using any resources. We ensure that all paths that are part of the test or memory controllers have significantly more timing margin than all tested paths, using timing constraints that guide Quartus to spend more effort on optimizing these paths. Similarly to [11], we use the main test controller to control the LUTs along the tested path to guarantee that all off-path inputs are fixed and that the tested path is sensitized from the source register to the BRAM. The main test controller also ensures that only one bit of the address port is tested at a time, so when we are testing a path that ends at a_{x0} all other bits of a_x are kept constant; this is important to guarantee that any timing failure is captured. When it is time to test the path ending at a_{x0}, the main test controller sends a signal to the memory controller to initiate the testing sequence. Fig. 9 shows the timing diagram of the inputs and outputs of the BRAM when testing a path that ends at a_{x0}. The main test controller sets the appropriate control signals to sensitize the tested path and sets the input stimulus to toggle a_{x0} which results in changing a_x from A to B as shown in the figure. Note that since we are testing a path that is ending at a_{x0}, all bits in addresses A and B are the same except for the 0^{th} bit.

The first stage of the test sequence is initializing the BRAM to a known state. To achieve this, the memory controller asserts we_x and sets the input data (d_x) such that it writes a specific value (D) to address A and the inverse of this value ($\sim D$) to address B. The initialization step is safe with no timing failures as the write operation is multi-cycled over 3 clock cycles.

In the second testing stage, the delay fault in the tested path is activated. Since the tested path ends at a write-only

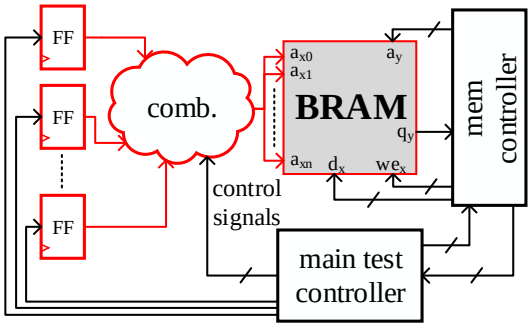

Fig. 8. Test circuitry for testing delay faults in paths ending at a write-only address port.

address port, if the corresponding write enable signal is low, then the presence or absence of a timing failure will have no effect on the BRAM. To activate the timing fault, the memory controller asserts the write enable signal for only one clock cycle at the cycle in which the new value of a_{x0} should be captured. In the first cycle of the *Activate fault* section in Fig. 9, a_{x0} toggles resulting in changing the address from B to A and the memory controller asserts we_x for this cycle only. In the absence of a delay fault in the tested path, the address registers of the BRAM should capture the new address A and since the corresponding write enable signal is high, the BRAM should write $\sim D$ at address A. Three cycles later when the address changes from A to B, the memory controller repeats the same test but writes D at address B; this ensures we test both the falling and rising transition along the tested path.

The final testing stage requires propagating the delay fault. In this stage, our memory controller sets the address of the read port of the BRAM (a_y) to read the contents of addresses A and B. In the absence of a delay fault, both write operations in the fault activation stage should have been successful, so we should read $\sim D$ and D from addresses A and B, respectively. If a delay fault is present then either one or both write operations would have failed and the output of the read operations would have been different. If the read data does not match the expected output, the memory controller asserts an error signal to the main test controller indicating that it is not safe to power the FPGA with the current V_{dd} at the tested clock frequency. The memory controller repeats the fault activation and propagation multiple times to test for delay faults in the presence of clock jitter.

While we have only explained the memory controller used to test paths ending at a write-only address port, we have designed multiple different memory controllers to test paths ending at different port types and we have also created a memory controller for testing paths that start at a BRAM. We extended FRoC to select which memory controller should be used based on the tested port. Our extended FRoC also automatically replicates the tested BRAM in the same read/write port configuration and width/depth mode used in the application.

V. Results

To evaluate our proposed DVS solution, we built a suite of ten benchmarks that use BRAMs and compiled them using Quartus while targeting a Cyclone IV EP4CE115F29C7 FPGA. Our benchmark suite includes: a subset of the CHStone [26] and the Titan [27] benchmarks; a discrete Fourier transform (DFT) core [28]; a sorting network (SN) core [29]; a turbo decoder [30] and two variations of a convolutional neural

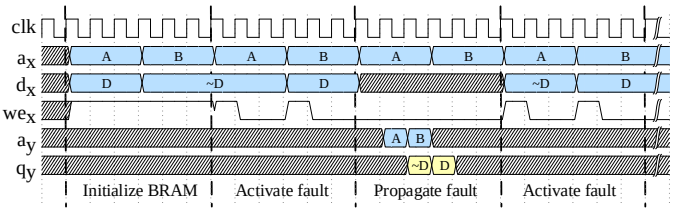

Fig. 9. Timing diagram of the BRAM inputs and outputs when testing a path ending at a write-only address port.

network inference core (CNN). Previous works [21], [31] have shown that measuring the delay of timing paths with reported delay more than 90% of the reported worst critical path (path with least slack) delay is sufficient to identify the failing clock frequency of the application; we define paths that meet this criterion as the top critical paths. By analyzing top critical paths in our benchmarks, we observed that on average 62% of these paths either start or end in a BRAM, which means that it is not safe to deploy a DVS solution that does not account for BRAMs on these circuits. Table I shows detailed statistics for our benchmarks, where P_{crit} is the number of top critical paths and P_{BRAM} is the number of top critical paths that start or end at a BRAM. We ran our benchmarks through the extended FRoC and Table I shows how many calibration bitstreams (N) FRoC generated to test these paths. Since FRoC adds control signals to tested LUTs, some paths cannot be tested in the same calibration bitsream, so sometimes multiple calibration bitstreams are required to test all paths [11]. In all benchmarks, our extended FRoC could test all used BRAMs for parametric failures and all top timing paths in the design; the number of bitstreams required for calibration varied from 1 to 17 with a geometric average of 6 bitstreams.

To validate the calibration table (CT) values generated from the calibration procedure, we designed a built-in-self-test (BIST) system around the DFT benchmark. This system consists of a linear feedback shift register (LFSR) that feeds inputs to the benchmark and a multiple input signature register (MISR) that generates a signature from the output of the benchmark over a large number of cycles. While this kind of testing cannot guarantee that timing critical paths of the application are exercised, we can still use it in feed-forward applications to validate our CT values. We tested the DFT benchmark at nominal conditions to generate a golden signature from the MISR, then we swept the supply voltage and clock frequency to measure the maximum clock frequency that does not result in any errors at each V_{dd}. The solid lines in Fig. 10 represent the measured F_{max} from the BIST system for two different FPGAs. Next, we performed our calibration process on these two chips for this benchmark. Since FRoC does not replicate the fanouts of the tested paths,

TABLE I. Benchmark statistics.

Circuit	LE	BRAM	DSP	F_{max}	P_{crit}	P_{BRAM}	N
DFT	15%	20%	27%	157 MHz	75	75	1
sha	2%	5%	0	113 MHz	374	369	3
CNN1	33%	88%	84%	171 MHz	1561	158	8
aes	9%	8%	0	89 MHz	733	733	6
blowfish	4%	8%	0	94 MHz	212	212	4
SN	12%	24%	0	154 MHz	308	299	5
sparcT1	76%	30%	3%	46 MHz	11528	127	15
turbo decoder	2%	10%	0	130 MHz	905	111	17
motion	1%	1%	0	82 MHz	395	395	10
CNN2	34%	88%	84%	168 MHz	818	46	5

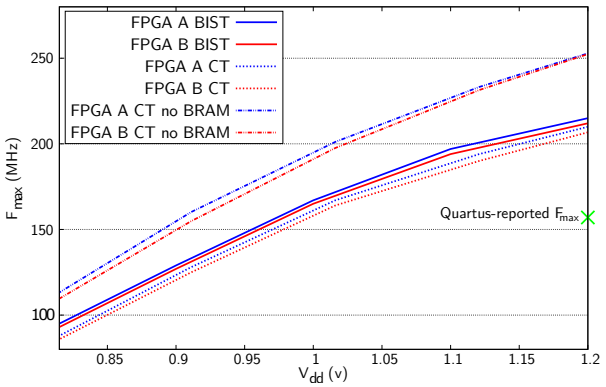

Fig. 10. Measured F_{max} for the DFT core on 2 different FPGAs using the BIST system and the proposed DVS solution.

it adds a fanout compensation factor to the CT values. This factor is computed by comparing the Quartus-reported delay of the tested paths in the calibration design to the same paths in the application. For the DFT core, this factor was 7%. Similarly to [32], our DVS solution also handles the difference between the IR drop during calibration and during online operation by monitoring the current consumed by the application and adjusting V_{dd} accordingly. Fig. 10 shows the operating points of the DFT core with our proposed approach. This graph shows that our calibration procedure is able to track the speed of each chip and is able to generate V_{dd} values for safe but power-efficient operation at various frequencies. We also ran another calibration procedure that ignored paths starting/ending at BRAMs and tested the slowest paths that use the soft logic only. The top two lines in Fig. 10 show the DFT operating points computed by this calibration procedure that ignores BRAMs. As shown in the figure, these operating points are unsafe and result in running the application at a much higher frequency than the maximum frequency measured using the BIST system. This demonstrates the importance of testing paths that start or end at BRAMs.

To evaluate the power savings, we ran the DFT benchmark with fixed V_{dd} and with our proposed DVS solution at the reported F_{max} (157 MHz). Since many designs are clocked with a frequency that gives some timing margin over the reported F_{max}, we also measured the power consumption at a lower clock frequency (100 MHz). Table II shows the power consumed at both these frequencies. At nominal V_{dd}, both FPGA chips consumed 1.4 and 0.9 Watts at 157 and 100 MHz, respectively. However, using our DVS solution, the chips consumed 0.95 and 0.49 W resulting in a total power reduction of 32% and 46% when the application is running at the reported F_{max} and at 100 MHz, respectively.

VI. Conclusion

While nearly all industrial applications use FPGA BRAMs to some extent, no prior DVS approach could fully guarantee safe BRAM operation. In this work, we have developed a DVS solution that automatically generates testers that are able to test for parametric failures in BRAM cells and test for timing failures in the BRAM interface logic. Our solution is fully automated and does not add any manual steps for FPGA users.

TABLE II. DFT power consumption with DVS and fixed-voltage operation.

	DVS	Fixed V_{dd}	Power reduction
Power consumption at 157 MHz	0.95 W	1.4 W	32%
Power consumption at 100 MHz	0.49 W	0.9 W	46%

We showed that our proposed DVS solution is able to save 32% power for a DFT core running with no timing margin and save 46% power at a lower operating frequency.

References

[1] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, April 1965.

[2] D. J. F. et al., "Device scaling limits of Si MOSFETs and their application dependencies," *Proceedings of the IEEE*, vol. 89, no. 3, pp. 259–288, Mar 2001.

[3] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. USA: Addison-Wesley Publishing Company, 2010.

[4] J. Nunez-Yanez, "Energy Proportional Computing in Commercial FPGAs with Adaptive Voltage Scaling," in *FPGAworld*, 2013.

[5] S. Zhao *et al.*, "A universal self-calibrating Dynamic Voltage and Frequency Scaling (DVFS) scheme with thermal compensation for energy savings in FPGAs," in *APEC*, 2016.

[6] J. M. Levine *et al.*, "Dynamic Voltage & Frequency Scaling with Online Slack Measurement," in *FPGA*, 2014.

[7] C. T. Chow *et al.*, "Dynamic voltage scaling for commercial FPGAs," in *FPT*, 2005.

[8] T. Tuan *et al.*, "A 90nm Low-power FPGA for Battery-powered Applications," in *FPGA*, 2006.

[9] C. R. Lefurgy *et al.*, "Active Management of Timing Guardband to Save Energy in POWER7," in *MICRO*, 2011.

[10] B. Bowhill *et al.*, "4.5 The Xeon processor E5-2600 v3: A 22nm 18-core product family," in *ISSCC*, 2015.

[11] I. Ahmed *et al.*, "Measure twice and cut once: Robust dynamic voltage scaling for FPGAs," in *FPL*, 2016.

[12] V. R. Devanathan *et al.*, "Towards effective and compression-friendly test of memory interface logic," in *ITC*, 2010.

[13] L. C. Chen *et al.*, "Transition Test on UltraSPARC- T2 Microprocessor," in *ITC*, 2008.

[14] J. Zeng *et al.*, "On correlating structural tests with functional tests for speed binning of high performance design," in *MTV*, 2004.

[15] A. Nabina and J. L. Nunez-Yanez, "Adaptive Voltage Scaling in a Dynamically Reconfigurable FPGA-Based Platform," *TRETS*, vol. 5, no. 4, pp. 20:1–20:22.

[16] J. Nunez-Yanez, "Adaptive voltage scaling in a heterogeneous FPGA device with memory and logic in-situ detectors," *Microprocessors and Microsystems*, vol. 51, pp. 227 – 238, 2017.

[17] S. Yazdanshenas *et al.*, "Don't Forget the Memory: Automatic Block RAM Modelling, Optimization, and Architecture Exploration," in *FPGA*, 2017.

[18] S. Mukhopadhyay *et al.*, "Statistical Design and Optimization of SRAM Cell for Yield Enhancement," in *ICCAD*, 2004.

[19] ——, "Modeling of failure probability and statistical design of SRAM array for yield enhancement in nanoscaled CMOS," *TCAD*, vol. 24, no. 12, pp. 1859–1880, Dec 2005.

[20] A. R. Alameldeen *et al.*, "Adaptive Cache Design to Enable Reliable Low-Voltage Operation," *IEEE Transactions on Computers*, vol. 60, no. 1, pp. 50–63, Jan 2011.

[21] E. Stott *et al.*, "Timing Fault Detection in FPGA-Based Circuits," in *FCCM*, 2014.

[22] M. B. Tahoori and S. Mitra, "Application-Dependent Delay Testing of FPGAs," *TCAD*, vol. 26, no. 3, pp. 553–563, March 2007.

[23] P. R. Menon *et al.*, "Design-specific path delay testing in lookup-table-based FPGAs," *TCAD*, vol. 25, no. 5, pp. 867–877, May 2006.

[24] J. S. J. Wong *et al.*, "Self-Measurement of Combinatorial Circuit Delays in FPGAs," *TRETS*, vol. 2, no. 2, pp. 10:1–10:22, Jun. 2009.

[25] N. Mehta *et al.*, "Limit Study of Energy & Delay Benefits of Component-specific Routing," in *FPGA*, 2012.

[26] Y. Hara *et al.*, "CHStone: A benchmark program suite for practical C-based high-level synthesis," in *ISCAS*, 2008.

[27] K. E. Murray *et al.*, "Titan: Enabling large and complex benchmarks in academic CAD," in *FPL*, 2013.

[28] P. Milder *et al.*, "Computer Generation of Hardware for Linear Digital Signal Processing Transforms," *TODAES*, vol. 17, no. 2, Apr. 2012.

[29] M. Zuluaga *et al.*, "Computer Generation of Streaming Sorting Networks," in *DAC*, 2012.

[30] J. L. andothers, "A dynamically-reconfigurable, power-efficient turbo decoder," in *FCCM*, 2004.

[31] I. Ahmed *et al.*, "Find the Real Speed Limit: FPGA CAD for Chip-Specific Application Delay Measurement," in *FPL*, 2017.

[32] S. Zhao *et al.*, "A robust dynamic voltage scaling scheme for FPGAs with IR drop compensation," in *APEC*, 2017.