Measure Twice and Cut Once:
Robust Dynamic Voltage Scaling for FPGAs

Ibrahim Ahmed, Shuze Zhao, 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*—Although dynamic voltage scaling (DVS) is a popular power reduction solution that has been widely used by processors and ASICs, it is still not commercially adopted by FPGAs. A unique feature of FPGAs that leads to challenges in adopting DVS is that the critical path and hence the minimum safe V_{dd} depends on the configured application. We present a robust DVS technique that solves these challenges. For each application, we generate a calibration table (CT) that stores the actual failing points of that application on a specific FPGA, under various operating conditions. This CT is used to scale V_{dd} while the application is running to guarantee safe operation with minimal power consumption. We develop an automated tool (FRoC) that ensures a Fast-Robust-Calibration of the FPGA to any application using it. FRoC ensures that the calibration process is invisible to FPGA users and does not add any extra manual steps to the design process. We show that our proposed DVS technique achieves a 33% total power reduction on two large applications.**

1. Introduction

FPGAs' ability to be programmed allows them to implement almost any digital circuit with a low non-recurring engineering (NRE) cost and a short time to market [1]. However, their programmability comes at a cost. Studies in [2] show that FPGAs consume 7-14 times more dynamic power and 5-87 times more static power than ASICs. Although the majority of earlier academic research on FPGAs focused on optimizing the area-delay metric, e.g. [3], [4], the end of Dennard scaling has driven more research towards reducing power consumption [5], [6]. While transistors are getting smaller, supply voltage has stopped scaling due to leakage current imposed constraints on V_{th} and indeed the supply voltage of high-end FPGAs from the top two FPGA vendors has remained almost constant from 40-nm to 20-nm technology. As we continue packing ever more functionality on an FPGA with essentially constant V_{dd}, power consumption is becoming a major concern especially with the increasing demand for low-power compute chips.

One can reduce FPGA power consumption at different design levels such as device-, circuit-, CAD- or system-level. DVS is an example of a system-level power reduction scheme which scales V_{dd} to the minimum value that still guarantees successful operation at the desired speed. Since dynamic and static power are quadratically and approximately exponentially related to V_{dd}, a small reduction in V_{dd} results in a significant total power reduction. DVS offers this power reduction without any changes to existing FPGA architectures and CAD tools. It is also independent from the underlying FPGA technology and thus is a long-lived solution that can add its benefits to many architectures and process generations.

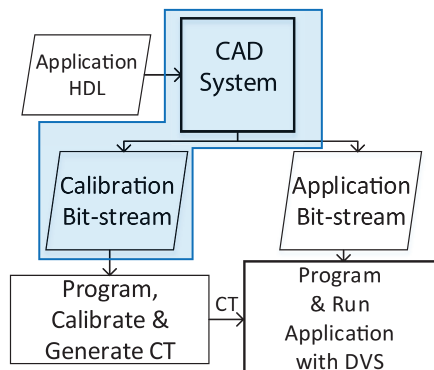


Fig. 1: Proposed DVS system overview.

In [7], we proposed a new DVS technique that is based on an off-line calibration procedure. Fig. 1 shows the system overview of the proposed technique. For each application, a calibration bit-stream is created which includes an exact replica of the application's critical paths, a test controller, on-chip heater circuits and a calibration controller to control the clock frequency and V_{dd}. The calibration bit-stream is used to test the critical paths of the application under different temperatures and V_{dd} to find the maximum operating frequency (Fmax) at every temperature-voltage pair on the *specific* FPGA running the application. On-chip heaters are implemented using the FPGA's soft logic to allow heating the die without needing any external equipment. Fmax values are stored in a calibration table (CT) and are used to set V_{dd} depending on the temperature while the application is running. In essence, we measure an application's Fmax, at different operating conditions, in two steps; an initial estimate using worst-case delays is done by a commercial static timing analyser (STA), as usual. Next, we use this estimate along with the STA-identified critical paths to carry out a second more accurate hardware measurement of Fmax on the specific target FPGA under various operating conditions.

Our work in [7] focused on designing an accurate high-frequency digital dc-dc power converter to reliably generate the variable V_{dd} along with the controller required to heat up the chip, control the clock frequency and scale V_{dd}.

To prove the concept (in [7]), we manually generated one calibration file by writing HDL code with location and routing constraints to replicate only the single most critical path of one application.

This work focuses on generating the calibration bit-stream, the part highlighted with a blue box in Fig. 1. The calibration bit-stream must capture all speed limiting paths of the application such that the generated CT is robust enough to ensure that the application is running error free. To capture within-die process variation, the calibration process should, at least, be carried out at the initial burn-in of each FPGA running the application. Ideally, to accommodate device aging effects such as bias-temperature instability [8], each FPGA should be calibrated with every power-up which means that the calibration time should be within seconds or at the maximum minutes. To this end, we:

- Develop a testing procedure to robustly measure an application's Fmax by measuring the delay of many overlapping critical paths, while only using one bit-stream and a short test time.
- Develop the FRoC (Fast Robust Calibration) tool, which automates the generation of the calibration bit-stream with the tested paths and test controller circuitry for a generic application.
- Explore the robustness of CTs generated from different calibration bit-streams, by using various coverage metrics and hardware measurements.

2. Background and Related Work

CAD tools are designed to be pessimistic since they must guarantee operation under worst-case conditions. These conditions include process variation, operating temperature and IR-drop. Process variation is steadily increasing as technology scales down and can be divided into die-to-die and within-die variation [9]. Imperfections in the fabrication process result in oxide thickness fluctuations, in-consistent dopant concentrations, and stress variation [10]. These imperfections lead to variation in transistor performance; [9] measured a 15% systematic within-die variation in a 65-nm FPGA.

Although DVS has been successfully deployed in various types of chips (notably CPUs and ASICs), it has not been commercially adopted in the FPGA industry. Applying DVS to FPGAs is inherently difficult due to the nature of the chip where the critical paths are application dependent and unknown at manufacturing time. For the same reason, FPGA vendors are forced to be conservative in their timing models and speed-bins; every resource (wire and block) must be faster than the timing-limit of a speed-bin for a chip to be placed in that bin. These factors add up, resulting in operating FPGAs with a significant timing margin, which we can convert to V_{dd} reduction. Thus, we expect that gains from applying DVS to FPGAs would be higher than other chips.

2.1. Related work

The most relevant power reduction solution available in the industry is the *SmartVID* adopted by Altera recently in their Arria 10 chips [11]. For each device, at manufacturing test time, *SmartVID* identifies the minimum V_{dd} for this specific device that still meets performance requirements. This value is then stored on-chip in non-volatile registers. Altera also provides an intellectual property (IP) block that reads these values and sends them to a voltage regulator system controller to set V_{dd}. *SmartVID* uses a closed-loop control system by having sense lines from the die power rail to compensate for voltage drops under high current load. *SmartVID* also has limited temperature adaptation; below $10\,^{\circ}$C the (higher) nominal V_{dd} is used. *SmartVID* is a step towards reducing pessimism due to global (die-to-die) variation. However, without considering the target application, pessimism due to within-die variation is still necessary. Moreover, *SmartVID* does not identify the optimum V_{dd} at different temperatures.

DVS on FPGAs has also been investigated in academic studies, and previous work can be divided into two categories: DVS using a logic delay measurement circuit (LDMC) [12] and DVS using on-line monitoring [13]–[17].

Chow *et al.* propose using an LDMC to track the delay of the application [12]. They first exercise the application at nominal conditions using a linear feedback shift register (LFSR) and store the correct output. Then, they test the application at lower V_{dd} until it fails and store the corresponding signal from the LDMC. During normal operation, they use the LDMC signal as a metric to scale V_{dd}. The main problem with such an approach is the assumption that the LFSR supplying the input vector actually exercises the critical path of the application. This is not a realistic assumption as the application gets larger with many buried states. For example, a circuit containing a simple 64-bit counter needs 2^{64} cycles just to reach the final state of the counter.

The idea presented in [13]–[17] is based on attaching shadow registers to monitor the application's high criticality (low slack) registers. The shadow registers' inputs are the same as the monitored registers, but their clock leads the monitored registers' clock by a variable phase offset. By varying this phase offset and comparing the output of the monitored registers and the corresponding shadow registers, the slack can be measured on-line. This information is used to scale V_{dd} until the minimum allowed slack is reached. A major limitation of this technique is the fact that registers in hard blocks like DSPs and BRAMs are not observable, so critical paths ending at a register inside one of the hard blocks cannot be monitored by a shadow register. On-line monitoring also adds resource and power overhead to the application due to the extra shadow registers and slack measurement circuitry added to the application. Another problem arises because the measured slack depends on the input data and the state of the application. This means that at some time the critical paths may not be exercised allowing the system to work at a lower V_{dd} which would save power. However, if suddenly the input changes and exercises the critical paths, errors could actually occur that affect the whole system. This maybe acceptable in processors where the state before the error could be retrieved, but it is quite

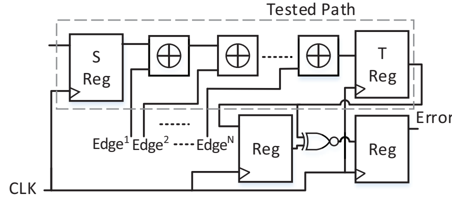


Fig. 2: Testing a single path's delay.

complex to handle errors in all applications on an FPGA. Since most industrial designs use hard blocks and have many internal states, the approaches discussed above cannot be applied to them.

In the testing community, a body of research focuses on testing delay faults on FPGAs. While not targeting DVS, it is still relevant to our proposed technique. The research can be divided into application dependent [18], [19] and application independent [20], [21] testing. Application independent testing is usually done by FPGA vendors to eliminate faulty chips and to speed-bin the remaining chips. Application dependent testing and especially the work presented by Harris *et al.* in [18] is more relevant to our proposal. They divide the paths they want to test into different groups, such that each group contains only disjoint paths. For each group, they generate a separate bit-stream with the whole application and testing logic. Then, they test the paths for delay faults and repeat this for all bit-streams. This can result in hundreds of bit-streams as paths tend to overlap quite frequently. This results in long CAD run times as each bit-stream requires a separate compile of a design, and even more importantly, large flash storage requirements and long test times as many bit-streams must be stored and configured into the device during testing. Our DVS solution cannot tolerate such a long testing time, especially since we want to measure the application's Fmax under various operating conditions which means the entire testing procedure must be performed many times.

3. Calibration Requirements and Approach

Our DVS solution requires that the calibration bit-stream can accurately measure the application's Fmax. This section presents the challenges and the testing procedure we developed to satisfy this requirement. Since we envision that the calibration process could be carried out at every power-up, it is also important that our testing procedure minimizes test time. To avoid long reconfiguration times, we designed our testing procedure to use only one calibration bit-stream.

As explained in 2.1, simply driving the primary inputs of an application with random inputs and checking the output is not a robust way to measure its Fmax. A more robust alternative is to extract the application's critical paths, sensitize them and measure their delay. To do so, we initially replicated the application's single most critical path and tested it. Measuring the delay of a single path on an FPGA can be done by converting all LUT configurations (LUT masks) on the path to perform an XOR, as the delay through a LUT is essentially independent of its function. The motivation for changing the LUT mask is to have the ability to easily sensitize the path and control the signal transition at every LUT. Fig. 2 shows an example of a single path being tested; every LUT on the tested path in the original design is transformed to an XOR while maintaining the exact original routing wires between every node. We control the transition at every LUT through the available control signals ($Edge^n$), and we check the delay by varying the clock (CLK) frequency until the path fails timing. To detect timing failures, we toggle the source (S) register's input every cycle and check the target (T) register's output. If it is not toggling every cycle, then the path has failed timing at the current clock frequency.

However, measuring an application's Fmax based only on its single most critical path can result in optimistic values due to within-die variation and thus, is not a robust technique. Moreover, FPGA components' delay have different sensitivity to V_{dd} changes. For example, an FPGA routing switch's delay reacts differently to scaling V_{dd} than a LUT's delay. FPGAs often drive the pass-transistor of routing switches with a separate, fixed voltage that is higher than V_{dd} (gate boosting) [22]. Fig. 3 shows a typical FPGA routing switch with a gate-boosted (V_{gb}) pass-transistor. When V_{dd} is lowered, V_{gb} stays the same which is different than what happens in a LUT where the gate voltage of the pass-transistor scales with V_{dd}. This difference alone implies that the delay across a LUT and a routing switch change differently when V_{dd} is scaled. Since we measure the application's Fmax at different V_{dd}, we must measure the delay of many near critical paths to capture the actual slowest path at each V_{dd}. Measuring the maximum operating frequency by measuring the delay of many paths is more difficult. When the paths are disjoint, we could replicate the structure shown in Fig. 2 for each path and test all paths simultaneously. However, typically paths are not disjoint and overlap in the resources they use.

The problem with overlapping paths is that we lose full control over LUTs shared by more than one path, which may prevent us from controlling the transition type at each LUT of every tested path. Also, timing errors could occur in overlapping paths that are eventually masked out and are not observable at target registers. To better understand the problem, we divide overlapping paths into two basic types: fan-in overlap and fan-out overlap; an example of each type is shown in Fig. 4.

Fan-out overlap occurs when multiple paths share a LUT through the same input port but fan-out to different destinations. This type of overlap has minimal effects on the testing procedure since we can still use the $Edge$ control

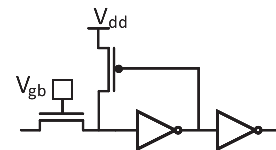
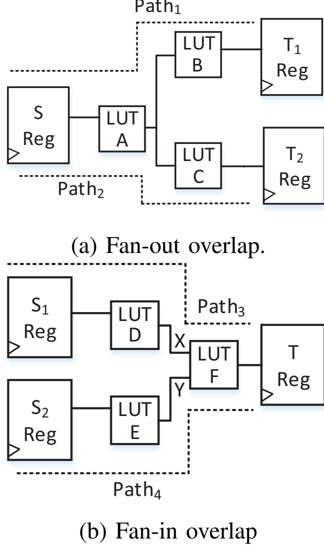


Fig. 3: A typical FPGA routing-switch with gate-boosted pass-transistor.



(a) Fan-out overlap.

(b) Fan-in overlap

Fig. 4: Examples of basic overlap types.

signal on LUTs to achieve the desired transition and there is no possibility of masking out timing errors. Fan-in overlap occurs when multiple paths use the same LUT through different input ports. This has a significant effect on the testing procedure. To guarantee that all timing errors are observable at the output registers, we have to ensure that the paths cannot interfere in a way that masks any timing errors. For example, in Fig. 4b if we test both $Path_3$ and $Path_4$ simultaneously, timing errors could be masked when both paths fail timing. To handle this problem, we propose to test fan-in overlapping paths sequentially. When testing a path, we fix (keep constant) all off-path inputs to each LUT on the currently tested path. In the absence of re-convergent fan-outs, we can fix off-path inputs by fixing all source registers that could cause them to toggle, so in Fig. 4b, when testing $Path_3$ we fix node *Y* by fixing the output of register S_2.

Re-convergent fan-out occurs when two or more paths have a fan-out overlap followed by a fan-in overlap and it presents additional test challenges. Fig. 5 shows an example of a re-convergent fan-out where $Path_1$ and $Path_2$ have a fan-out overlap at *Atom A* and then a fan-in overlap at *LUT B*. An atom could be a LUT or a register. In this case, when testing $Path_1$, if we fix node *Y* by fixing all source registers that affect it, node *X* will also be fixed making it impossible to test $Path_1$. To handle this problem, we add another control signal to each LUT that allows us to fix the LUT's output. With this control signal, we can fix all off-path inputs by fixing the output of the LUTs driving them. So in Fig. 5 when we are testing $Path_1$ we would set the control signals of the LUT feeding *Y* to fix its output at a constant value. This gives us the ability to measure the delay of many overlapping paths in the presence of re-convergent fan-outs, using only one calibration bit-stream.

After considering the modifications required to handle overlapping paths and re-convergent fan-outs, we developed a LUT mask that allows us to not only control the transition's polarity at every LUT but also lets us fix its output to a constant value. Instead of programming every tested LUT to a simple XOR (as in Fig. 2) our new function is

$$Combout = Fix \bullet (I_1 \oplus I_2 ... \oplus I_{K-2} \oplus Edge) + \overline{Fix} \quad (1)$$

where Fix and $Edge$ are control signals that fix the output and select the edge transition's type, respectively. K is the number of LUT inputs (4 to 6 for modern FPGAs) and I_i is the ith LUT input. I_1 to I_{K-2} are the *available* inputs that can be connected in the same topology as the application circuit to allow us to model and test overlapping paths, but Fix and $Edge$ cannot be connected to any tested path and are only used to control testing. Given that we fix all off-path inputs, $Edge$ gives us the flexibility to model all edge transition types between the on-path input and $Combout$. The downside of having two control signals, is that we can only test paths using the *K-2 available* input ports of a LUT.

3.1. Carry Chain Consideration

The above description assumes that an FPGA's Logic Element (LE) has one LUT and one output ($Combout$). However, almost all modern FPGAs have some form of carry-chains to speed-up addition. Since our target for this work is a Cyclone IV FPGA, we studied its carry-chain and customized our testing procedure towards its architecture. LEs in Cyclone IV include a 4-input LUT and they have a $Combout$ and a $Cout$ output [23]. $Combout$ is the standard output which can drive any wire connected to the LE. $Cout$ is a special output that computes the carry of a 3-bit addition and can only drive a special input (Cin) of the neighbouring LE. $Cout$ is only dependent on three inputs and one of them is Cin, which can't be driven from general routing wires. If we use two of these three inputs to control $Cout$ the same way as $Combout$ in Eq. 1, we would only test paths using one input port which reduces our coverage of tested paths significantly. The adder function implemented in the carry-chain has some characteristics that we can exploit: $Cout$ is negative unate in Cin and it is either positive or negative unate in all other inputs. Based on these characteristics, we can eliminate the $Edge$ control signal (for LUTs in carry-chains) that selects the type of edge transition to $Cout$ and use different LUT masks to achieve the required transition occurring in the application design. This means that from the three input ports that $Cout$ depends on, we only connect one port to a control signal (Fix) and thus, we can test paths using the remaining two ports.

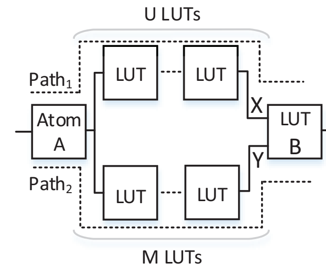


Fig. 5: Re-convergent fan-out.

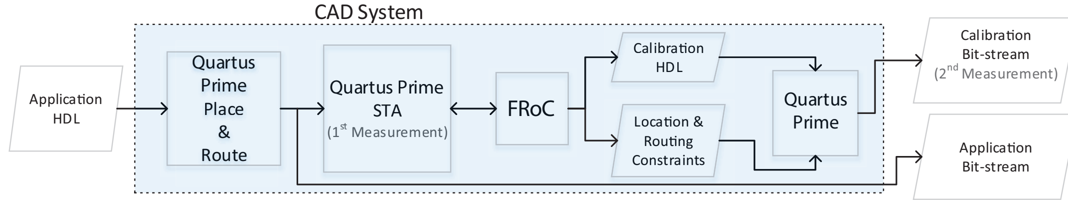


Fig. 6: Proposed design-flow with FRoC.

3.2. Local Routing Congestion Consideration

Adding two control signals (Fix and $Edge$) to each LE, could result in local routing congestion. Each cluster of LEs (LAB) has a certain number of LAB lines that connect LE inputs to the general routing [24]. Since in most designs not all LE inputs in one LAB are used or they usually share signals, FPGAs are designed such that the number of LAB lines is smaller than the sum of all LE inputs in a LAB [25]. To minimize the probability of running out of LAB inputs, we try to share as many control signals as possible and eliminate any redundant control signals. For example, we only need to fix the output of an LE when one of its fan-outs is using more than one of the *available* inputs. So in Fig. 4 we would only add the Fix control signal to *LUT D* and *LUT E*.

4. FRoC

It is not reasonable to ask users to manually replicate paths they want to test, apply all the constraints described in the previous section and add failure testing circuitry to get the CT for each application they design. We developed FRoC to perform this process automatically with no additional input beyond the design. We integrated FRoC with a commercial FPGA CAD tool (Quartus Prime). Fig. 6 shows the proposed design flow augmented with FRoC, where all test generation is invisible to the user.

First, we run placement and routing on the user's design; then we use Quartus Prime STA to analyse the design (first measurement) and generate timing reports for the top critical paths of the circuit. FRoC takes the design and the Quartus Prime STA report as inputs, and generates a new calibration design and constraints that are fed back into Quartus prime to generate the calibration bit-stream. To create this design, FRoC:

- *Extracts* candidate paths and *selects* which candidate paths will be tested.
- *Replicates* the selected paths.
- *Groups* the replicated paths into test phases and generates the test controller.

4.1. Path Extraction and Selection

FRoC runs the Quartus Prime STA and extracts the application's top critical paths (candidate paths). Ideally, we would select all candidate paths to replicate and test, but there are several cases where we must ignore (not replicate) some candidate paths in order to accommodate the control circuitry we require to test the other selected paths.

There are four cases where we are forced to ignore paths. First, we need to have two control signals ($Edge$ and Fix) for each LUT, so we remove paths to ensure that at least two LUT inputs are free for control signals. Our tool chooses which ports will be connected to control signals in a manner that ensures that the less critical paths are the ones to be ignored. For example, if the candidate paths use all of a certain LUT's inputs, our tool ranks the LUT's input ports based on the most critical path at each input; then it removes all paths using the least important port. This heuristic ensures that as we increase the number of candidate paths, we do not ignore the more critical paths.

The second reason for ignoring paths is to avoid local routing congestion. Although, as mentioned in Section 3.2, we try to minimize the number of external control signals going into a LAB, we can still face situations where the number of external signals is higher than or almost equal to the number of LAB lines. Our tool tracks the number of external signals going into a LAB and ensures that this number is at most 90% of the number of LAB lines. We leave 10% of the LAB's input signals free to ensure routability with the sparse crossbar between LAB lines and LUT inputs [26]. If the input signal count exceeds the 90% threshold for some LAB we rank the LAB's external signals based on the slack of the paths using them, and we ignore all paths using the external signal with the highest slack. We repeat this procedure for the next highest slack input until we are below the threshold.

The third reason for ignoring paths is re-convergent fan-out. Although the Fix control signal allows us to test most re-convergent fan-out paths, it is unable to break one type. In Fig. 5, if either the upper (U) or lower (M) path from *Atom A* to *LUT B* contains 0 LUTs, then it is impossible to fix the off-path input at *LUT B* while testing both paths. For example, if only M is 0, we can test $Path_2$ by fixing node X. However, we can't fix node Y and test $Path_1$. Our tool breaks this connection by removing the edge with more slack.

The last reason for ignoring paths occurs at LUTs having $Cout$ as an output. As explained earlier, only one control signal is needed for $Cout$. However, one of the inputs (Cin) controlling $Cout$ is only accessible from the neighbouring LUT. So if we have to control a LUT through Cin, we must ensure that the LUT driving this Cin is not used by any other path. In case this LUT is used by other paths, we ignore the less critical paths to handle this scenario.

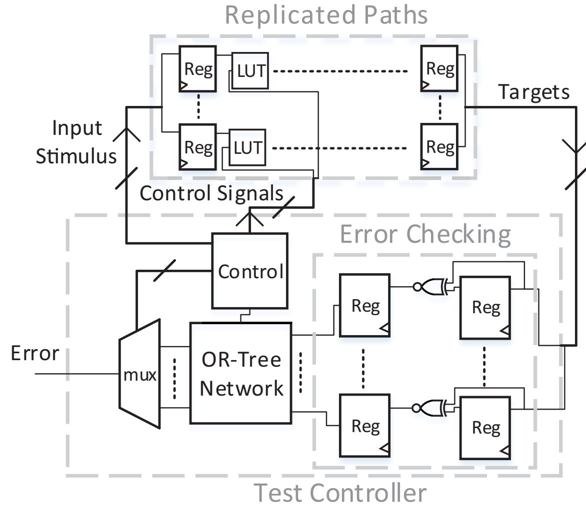


Fig. 7: Test controller block-diagram.

4.2. Path Replication

To replicate all selected paths, FRoC first creates primitives (WYSIWYGs) [27] to represent each LUT or register on the replicated paths. If the LUT in the original circuit does not use $Cout$, then it is created with a LUT mask implementing Eq. 1. If $Cout$ is used, the chosen LUT mask depends on the transition types between the LUT inputs and $Cout$ in the user design. In the calibration bit-stream, some LUT inputs come from the replicated paths while others are control signals to sensitise paths and control transition polarity. Next, FRoC fixes the location of every created LUT to match the exact location from the user design. Finally, FRoC generates a routing constraints file [27] that forces the routing of the replicated paths to match that in the user design, including both the routing wires and LUT input ports used.

4.3. Path Grouping and Test Controller

After replicating all selected paths, FRoC starts grouping paths that can be tested simultaneously into test phases by generating a path-relation graph that is similar to the graph presented in [18]. Each node in this graph represents a path and an edge between two nodes indicates that these two paths cannot be tested at the same time. To generate the graph we loop across all replicated LUTs and at each input port we add edges between paths using the current port and all paths using LUTs driving the other input ports. After creating the graph, FRoC colors the graph greedily by visiting nodes in the order of criticality. Each color represents a test phase, so nodes with the same color will be tested in parallel. We formulated this as a graph colouring problem to minimize the number of test phases and hence minimize testing time.

FRoC then generates HDL for the test controller which consecutively loops through all test phases and at each phase performs the following steps:

- Sets the appropriate control signals, generates input stimulus for the current test phase and waits for five cycles, to let the control signals settle.
- Checks for errors at the target registers of paths in the current test phase for Y cycles.

The block diagram of the test controller is shown in Fig. 7. The test controller toggles the inputs of source registers that are part of the paths being tested in the current test phase and fixes the other source inputs to a constant value. To set the appropriate control signals, the test controller sets the Fix control signal to fix the output of all LUTs driving an off-path input to the LUTs forming the paths tested at the current phase. Also, it sets the $Edge$ control signal to get the worst-case edge transition at each tested LUT. FRoC uses the user design timing report (generated by Quartus Prime STA) to identify the worst-case combination of edge transitions for every path. However, the test controller is also able to test all possible rise/fall combinations at the expense of longer test time. As shown in Fig. 7, the test controller checks for errors at all target registers. It also contains an OR-tree for each test phase that only checks for errors in target registers that are part of the paths being tested at this phase. At any point in time, the error is the output of the OR-tree that corresponds to the current test phase.

The chosen time (Y cycles) to check for errors at each test phase is affected by several factors like clock jitter and power supply ripple. We define the maximum operating frequency as the frequency that generates zero errors through the entire testing process. This is different than the definition used in [28], [29], as our goals are different. In [28], [29], testing is done to measure the delay of some resources on the FPGA, so it is important to remove clock uncertainty to report an accurate delay. However, we are trying to measure the maximum operating frequency of the application; thus, we must account for the worst-case jitter and power supply ripple. We chose a testing time that is long enough to capture high-frequency clock jitter that can affect cycle times and capture the lowest point of the power supply ripple, during each test phase. The switching frequency of our designed dc-dc converter is 500 KHz, which gives an output voltage ripple with a $2\,\mu s$ period. We test every test phase for at least two periods ($4\,\mu s$) of the power supply ripple which also covers enough cycles to capture high-frequency clock jitter at the tested frequencies.

It is critical that all parts of the test controller fail at a higher frequency than the replicated paths. To ensure this, we deeply pipeline the test controller, allowing only one stage of logic between pipeline stages in all OR-trees. We also allow the control signal to settle for five cycles before checking for errors. Moreover, we generate timing constraints that guide Quartus Prime to spend more effort minimizing the delay of the test controller and other parts of the calibration design not related to the replicated paths.

5. Experiments and Results

To evaluate our DVS technique we developed benchmarks and targeted them to a Cyclone IV EP4CE115F29C7 FPGA manufactured using TSMC 60-nm technology. The FPGA is mounted on a DE2-115 board and has a nominal voltage of 1.2V. We experimented with two different applications. Our first benchmark is based on a dual-channel

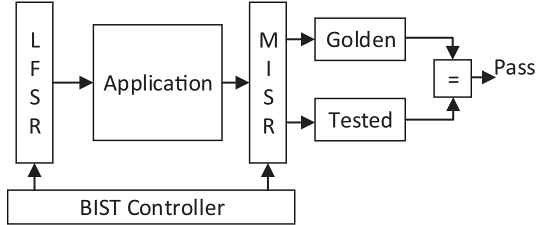


Fig. 8: Benchmarks overall structure.

51-tap low pass FIR filter generated from the Quartus IP catalogue. The second benchmark is based on a full crossbar (XBar) with 16 100-bit-wide-ports and registered I/Os. To validate and assess the robustness of the generated CT, we need to know the actual maximum operating frequencies of the benchmarks at different supply voltages. This also helps us quantify the pessimism added by Quartus Prime, in the absence of DVS. In order to do so, we wrapped our applications with a built-in-self-test (BIST) structure. Fig. 8 shows an overview of our benchmarks' general structure; both benchmarks have the same structure but use different applications (XBar or FIR). The linear feedback shift register (LFSR) feeds the circuit with pseudo-random inputs, while the multiple input signature register (MISR) analyses the output signals at each cycle. We ensured that the circuit under test fails timing before other parts. As explained in Section 2.1, this set-up is not guaranteed to exercise all critical paths of a general application, so we chose our applications to be feed-forward circuits with no buried states. We compile each benchmark through Quartus Prime with an unachievable 200 MHz timing constraint and default settings. Table 1 summarizes the resource usage and speed reported by Quartus Prime for both benchmarks.

To identify the actual Fmax, we first run the application at nominal voltage and nominal Fmax reported by Quartus Prime for 2^{32} cycles and store the output signature as the golden reference. We then vary V_{dd}, run the application again for 2^{32} cycles at different frequencies and check the result of comparing the output signature with the golden signature. We powered the FPGA using a variable output DC power supply and used a function generator to provide the clock; this allowed us to manually vary both V_{dd} and the clock frequency. We measured Fmax at different V_{dd} values for both benchmarks and plot them in Fig. 9. This graph shows that the actual Fmax is on average 50% higher than the reported nominal Fmax and that the application can successfully run at the nominal Fmax with a 20% reduction in V_{dd}, on average.

An important question to answer is how many candidate paths should FRoC extract to generate the CT. At one extreme, extracting and testing one path gives the shortest

Table 1: Benchmarks' summary

	FIR	XBar
Logic Elements	67,505 (59%)	26,579 (23%)
Quartus Prime (nominal) $Fmax$	121 MHz	115 MHz
Nominal V_{dd}	1.2 V	

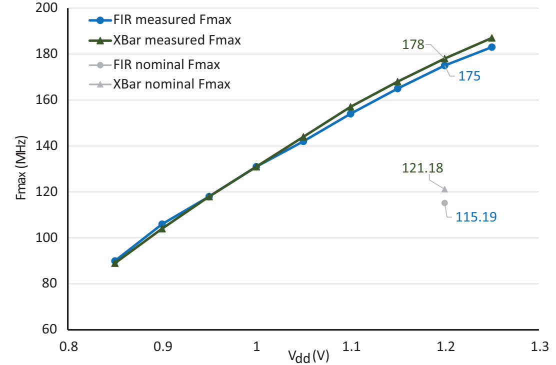


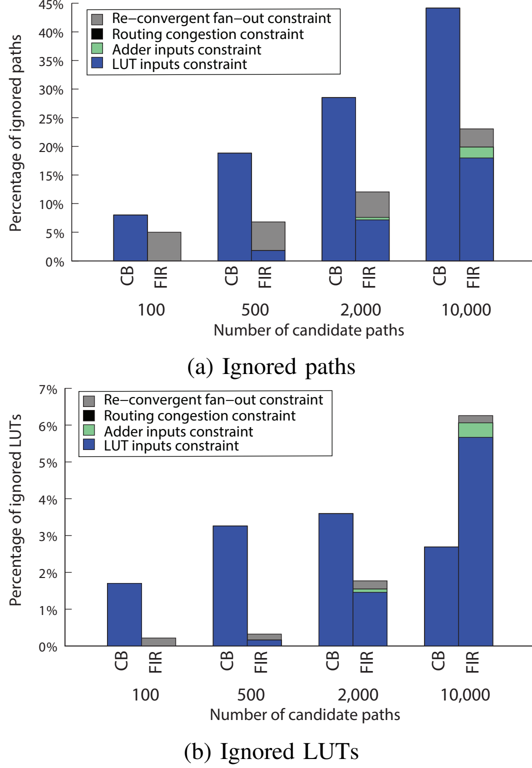
Fig. 9: Benchmarks maximum operating frequencies.

test time but is not robust, as explained earlier. At the other extreme, extracting and testing all possible paths results in high confidence in the generated CT but is not really feasible. Paths can be exponential in the number of nodes in a circuit, so to test all paths we must use many bit-streams and the test time for each bit-stream would be significantly long. Moreover, testing all paths is unlikely to be necessary as many paths have a much lower delay than the critical path and will not add any real information to the CT. We believe that a compromise should be made between test time and the number of extracted paths. To gain insight into this trade-off, we performed several experiments and we divide the results into coverage analysis and hardware measurements.

5.1. Coverage Analysis

In this section, we investigate the achieved coverage when we vary the number of candidate paths and vary their extraction procedure. Moreover, since we ignore several candidate paths to accommodate the testing of others, we also quantify the effect of the ignored paths. We define two different sets of paths: the all paths set (AP) which contains all paths of a circuit and the pairs only set (PO) which contains only the most critical path between every pair of registers. Since many paths are dominated by others, testing every path would cost us test time without much-added value to the CT. So we use PO as a heuristic representation of the important paths that actually limit the circuit speed. FRoC enabled us to run many experiments by varying the number of candidate paths it extracts and changing the set (AP or PO) from which it extracts them (candidate paths).

To quantify the effect of ignoring paths, Fig. 10 shows the normalized ignored paths and LUTs across different numbers of candidate paths extracted from the PO set. The figure also quantifies the impact of each constraint (explained in Section 4.1) that can cause us to ignore paths. As more paths are added, we are forced to ignore some paths to allow the insertion of our testing logic. The histogram shows that the dominant constraint for ignoring paths and LUTs is the number of inputs constraint, which is the sole reason for ignoring paths in the XBar benchmark. This suggests that as we move our testing procedure to high-end FPGAs with larger LUTs, significantly fewer paths will be ignored. Another important point is the fact that the amount of ignored LUTs (LUTs in the candidate paths but which



(a) Ignored paths

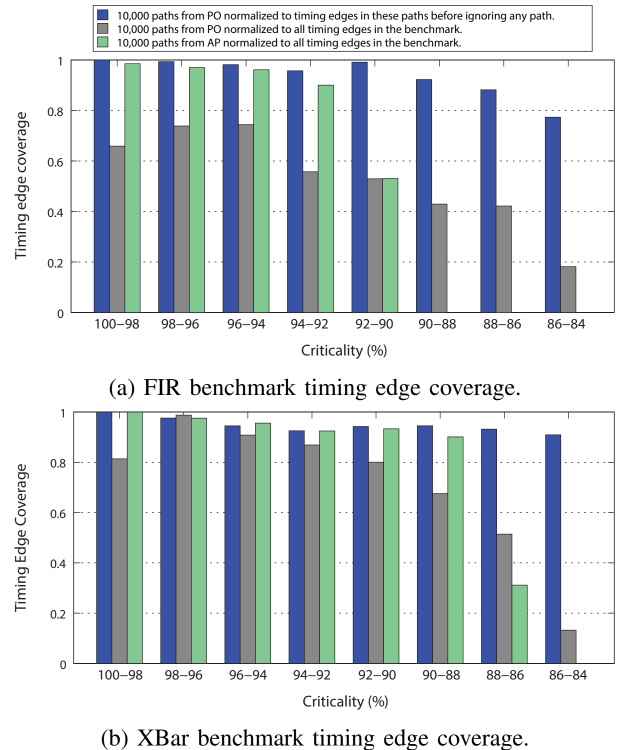
(b) Ignored LUTs

Fig. 10: The number of ignored paths and LUTs when extracting different numbers of candidate paths from PO.

are not present at all in the replicated paths) is significantly less than ignored paths. For the XBar benchmark when we ignore around 45% of the paths, we only ignore less than 3% of LUTs. This means that even though we are ignoring many paths, we are still covering most of the LUTs in the candidate paths.

Although we are testing the majority of LUTs, we must ensure that the ignored LUTs are less critical and that we are testing the remaining LUTs through the longest paths using them and not just any path. To verify this, we evaluated the timing edge coverage of the replicated paths. A timing edge is a connection between the pins of an atom. This connection could be between the input and output pin of an atom (cell delay) or the output pin of an atom and input pin of another atom (connection delay). We binned timing edges based on their criticality, which is the ratio of the longest path using this timing edge to the most critical path of the application.

Fig. 11 shows the coverage of the replicated paths' binned timing edges for both benchmarks, when FRoC extracted 10,000 candidate paths from the PO and AP set. The blue and grey bars are the coverage when the 10,000 paths are extracted from the PO set normalized to the timing edges in these 10,000 candidate paths (before ignoring paths) and normalised to all timing edges in the benchmark, respectively. Since the blue bar compares the timing edges of the replicated paths against the timing edges of the candidate paths, it is the best metric to reflect how much coverage are we losing due to ignoring paths. The figure shows that we are covering all given edges in the most critical bin and that our coverage goes below 90% only at the least significant bins. The grey bar reflects how much we are covering from the whole application. It is expected that this percentage would be smaller than the blue bar, as paths in the PO set neglect several edges since it only considers the worst path between pairs of registers. The green bar represents the coverage when we extract 10,000 candidate paths from the AP set normalized to all timing edges in the application. It is shown that our coverage is extremely high for the more critical bins. These results indicate that FRoC successfully ignores the less critical paths in favour of the more critical paths, which results in covering the more critical timing edges. Moreover, it also suggests that extracting paths from the AP set is better as we obtain higher coverage compared to extracting from the PO set. However, we still have to consider testing time which is proportional to the number of test phases. Table 2 presents the number of test phases for different numbers of candidate paths extracted from PO and AP sets. Although extracting paths from the AP set gives us more timing edge coverage, it requires more test phases as replicated paths in this case overlap more than paths from the PO set. The increase in the number of phases is application dependent and is based on how much overlap exists between paths sharing the same source and target register. For the FIR application, the number of test phases for 10,000 paths from the AP set is 5x larger than 10,000 paths from the PO set, while it is only 1.7x larger for the XBar application. Given that each test phase



(a) FIR benchmark timing edge coverage.

(b) XBar benchmark timing edge coverage.

Fig. 11: Replicated paths' timing edge coverage for both benchmarks across multiple experiments.

Table 2: Number of test phases and required test time.

# of Paths	2000	2000 (AP)	10000	10000 (AP)
FIR (Test phases)	34	78	69	349
FIR (Test time (µs))	137	315	279	1410
XBar (Test phases)	12	20	17	29
XBar (Test time (µs))	49	81	69	117

is tested for 4 µs and requires 5 cycles for the control signals to settle (as explained in 4.3), Table 2 also indicates the time required to test the replicated paths at nominal Fmax under one operating condition (temperature-voltage pair).

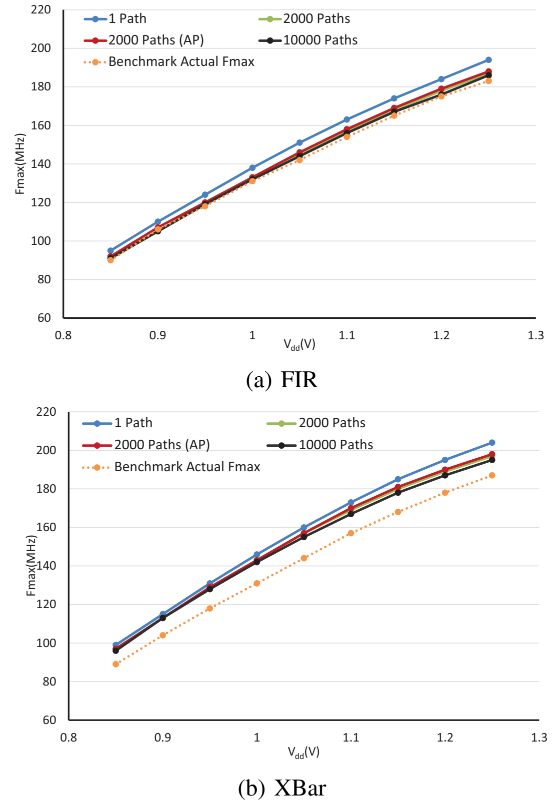
To evaluate if the higher coverage achieved when extracting paths from the AP set results in a more accurate CT, we have to perform hardware measurements and compare the generated CTs.

5.2. Hardware Measurements

5.2.1. Manual Measurements. This section shows how the CT of Fmax is impacted by the number of candidate paths and the set (AP or PO) from which they are extracted. To allow for a fair comparison with the values in Fig. 9, we used the same set-up used to measure the benchmark's actual Fmax with the addition of a heat-gun to fix the package temperature so that we can measure Fmax at different V_{dd} but constant temperature. Moreover, we connected the error signal in Fig. 7 to an oscilloscope to monitor the failing clock frequencies. Fig. 12 shows the measured Fmax for several calibration bit-streams using different numbers of candidate paths, for both benchmarks. When measuring Fmax from the calibration bit-streams, we fixed the package temperature at 58 °C and 41 °C for the FIR and XBar benchmarks respectively. These were the package temperatures when we were running the full-benchmarks at nominal conditions. We measured Fmax using various numbers of candidate paths and for both benchmarks, the values start to stabilize beyond 2000 paths, so we only plot the results for 1, 2000 and 10000 paths.

Fig. 12a shows that, for the FIR benchmark, the measured Fmax from the calibration bit-streams using 2000 candidate paths or more closely match the benchmark's actual Fmax. As expected using only 1 candidate path in the calibration bit-stream results in Fmax values that are noticeably different than the actual values. The graph also shows that the 10,000 paths curve is slightly below the 2000 paths which is also expected due to IR-drop. When more paths are tested, the FPGA draws more average current and thus the IR-drop increases resulting in this slight change in values. To ensure that the low IR-drop is not the reason for the high Fmax measured with 1 path, we added redundant logic that consumes extra current (more than the 2000 paths' current) to the calibration bit-stream with 1 path and re-measured Fmax. In this case, Fmax values were slightly less than without the redundant logic but still higher than results obtained with 2000 paths. The average decrease of Fmax values when using the extra redundant logic was 0.4% and the maximum decrease was 1%. Another interesting point is the fact that the measured Fmax values for 2000 paths extracted from PO and AP are almost identical. This means that, for the benchmarks tested in this work, the extra coverage achieved by extracting paths from the AP set is not worth the increase in test time as it does not affect the CT values.

Fig. 12b shows the measured Fmax for the XBar benchmark, and most trends match those of the FIR design. Firstly, we see that 1 path is again not sufficient and that 2000 candidate paths produces a noticeably lower calibration Fmax that better matches the benchmark circuit. Also, the results of 2000 candidate paths from AP and PO are almost identical. The main difference is that the measured Fmax values do not match the benchmark's actual Fmax, where the benchmark's actual Fmax values are on average 6% lower than the Fmax values measured from 2000 candidate paths. This discrepancy is caused by the absence of some fan-outs in the calibration bit-streams. Currently, FRoC replicates the selected paths without modelling fan-outs that are not part of any selected path. This could reduce the delay of the replicated paths compared to the actual delay in the application as LUT outputs and routing switches do have moderate delay increases when their fan-out increases. The FIR benchmark is mainly composed of adders so most of the critical paths are long carry-chains, while the XBar benchmark has no adders and most of its critical paths are formed by regular LUTs and routing elements. Carry-chains could only have fan-outs at the beginning or the end of the



(a) FIR

(b) XBar

Fig. 12: Measured Fmax when using different numbers of candidate paths.

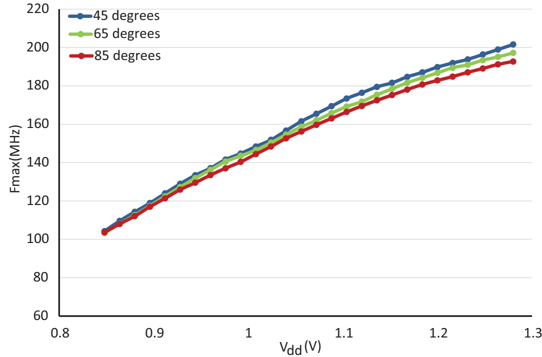


Fig. 13: Automatically measured CT for XBar benchmark with different package temperatures.

chain, so the delay is almost not affected by the absence of fan-out. However, the critical paths of the XBar have many fan-outs at almost every intermediate node in the path, so the absence of fan-outs have an effect in this case. To check that this is the reason for the discrepancy, we compared the Fmax reported by Quartus Prime for the calibration bit-stream with 1 path against the reported Fmax for the full benchmark. The FIR calibration bit-stream resulted in a similar Fmax to the benchmark's reported Fmax, while, for the XBar calibration bit-stream, Quartus Prime reported an Fmax which is almost 5% higher than the benchmark's reported Fmax. We went one step further where we manually modelled fan-outs, added it to the calibration bit-stream with 1 path and re-measured the actual Fmax for the XBar benchmark. The measured Fmax values were on average 3% lower than without fan-outs. These results show that fan-outs have a significant effect on delay and must be considered during the calibration process.

5.2.2. Automatic Measurements. This section presents results of the automatically generated CT when using on-chip heaters to heat the die. The experimental set-up is different than the one used in generating the actual benchmarks Fmax. We power the FPGA using our designed dc-dc converter and use a clock generator to provide the clock. Both are controlled from the FPGA and more details on this set-up can be found in [7]. Using this set-up, we are able to automatically generate the CT entries at different temperatures. Fig. 13 shows part of the generated CT for the XBar at 45 °C, 65 °C and 85 °C when using 2000 candidate paths from the PO set to generate the calibration bit-stream. Our CT covers the temperature range from 30 °C to 85 °C, but we only show a subset of those values. This figure shows that the effect of temperature on the measured Fmax is more significant at higher voltages. The FIR CT shows similar behaviour.

5.3. Guard-band and Power Reduction

After generating the CT we must add a guard-band to the measured Fmax values to capture various effects that are not considered in the calibration process. Both the instantaneous and the average current drawn during the calibration process are significantly different than the current drawn when the application is running. This results in different IR-drop that could affect delay. Moreover, capacitive crosstalk between simultaneously switching wires is also different when the application is running. To accommodate these variations, we add a 5% guard-band to match our experimental results. Since at this stage our tool does not model the complete fan-outs of replicated paths, we also add a guard-band to compensate for the missing fan-outs. This guard-band is calculated based on the difference between the Quartus Prime reported Fmax from the benchmark and the calibration HDL, which was around 1% for the FIR and 5% for the XBar.

Table 3: Power in nominal condition and with DVS.

	Total Power (W)	Temperature (°C)
FIR (Nominal)	2.85	58
FIR (DVS)	1.89	50
XBar (Nominal)	1.07	41
XBar (DVS)	0.72	39

After adding the guard-band, we ran the benchmarks at the reported nominal Fmax using both nominal V_{dd} and the lowest guard-banded V_{dd} from the CT. Table 3 presents the total measured power consumption and the package temperature (due to self-heating) for the nominal V_{dd} and with DVS. With our DVS approach we are able to save 33.2% total power consumption on average.

6. Conclusion and Future work

We presented a robust DVS solution based on a two-step measurement approach, that exploits FPGA reconfigurability and calibrate every FPGA to its target application. Using information generated by a standard STA, we replicate the application's speed limiting paths, measure their delay under various operating conditions on the specific FPGA being programmed, and store this information on-chip in a CT. In this paper, we presented a testing procedure that enables us to measure the delay of many overlapping paths using a single bit-stream. We developed FRoC, a CAD tool that analyses an application, replicates its most critical paths and generates testing circuitry to measure the replicated paths' delays. FRoC also tries to minimize testing time by dividing paths into groups that can be tested in parallel.

We tested FRoC and our DVS solution on two applications (FIR and XBar). The results show that after adding a small guard-band to our CT we can safely run both applications with the guard-banded values and achieve a 33.2% total power reduction, on average.

Our next step is to add fan-out modelling to FRoC which will allow us to eliminate the guard-band needed for fan-out enabling further voltage reduction and power savings. To allow FRoC to handle hard-blocks, we will develop custom tests for DSPs and BRAMs that can identify the minimum safest voltage for the specific blocks used by the user application.

Acknowledgments

We would like to thank Mark Bourgeault, Gurvinder Tiwana and Ashraf Lotfi for the insightful discussions. This work was funded by NSERC, Altera, OCE and SRC.

References

[1] I. Kuon, R. Tessier, and J. Rose, "Fpga architecture: Survey and challenges," *Found. Trends Electron. Des. Autom.*, vol. 2, no. 2, pp. 135–253, Feb. 2008. [Online]. Available: http://dx.doi.org/10.1561/1000000005

[2] I. Kuon and J. Rose, "Measuring the gap between fpgas and asics," *TCAD*, vol. 26, no. 2, pp. 203–215, Feb 2007.

[3] M. Sheng and J. Rose, "Mixing buffers and pass transistors in fpga routing architectures," in *FPGA*, 2001, pp. 75–84.

[4] V. Betz and J. Rose, "Fpga routing architecture: Segmentation and buffering to optimize speed and density," in *FPGA*, 1999, pp. 59–68.

[5] A. A. M. Bsoul and S. J. E. Wilton, "An fpga with power-gated switch blocks," in *FPT*, 2012, pp. 87–94.

[6] E. Kadric, D. Lakata, and A. DeHon, "Impact of memory architecture on fpga energy consumption," in *FPGA*, 2015, pp. 146–155.

[7] S. Zhao, I. Ahmed, C. Lamoureux, A. Lotfi, V. Betz, and O. Trescases, "A universal self-calibrating dynamic voltage and frequency scaling (dvfs) scheme with thermal compensation for energy savings in fpgas," in *APEC*, March 2016, pp. 1882–1887.

[8] A. Amouri, S. Kiamehr, and M. Tahoori, "Investigation of aging effects in different implementations and structures of programmable routing resources of fpgas," in *FPT*, 2012, pp. 215–219.

[9] T. Tuan, A. Lesea, C. Kingsley, and S. Trimberger, "Analysis of within-die process variation in 65nm fpgas," in *ISQED*, March 2011, pp. 1–5.

[10] K. Agarwal and S. Nassif, "Characterizing process variation in nanometer cmos," in *DAC*, June 2007, pp. 396–399.

[11] Altera, "An 711: Power reduction features in arria 10," 2015.

[12] C. T. Chow, L. S. M. Tsui, P. H. W. Leong, W. Luk, and S. J. E. Wilton, "Dynamic voltage scaling for commercial fpgas," in *FPT*, Dec 2005, pp. 173–180.

[13] J. M. Levine, E. Stott, G. A. Constantinides, and P. Y. K. Cheung, "Online measurement of timing in circuits: For health monitoring and dynamic voltage & frequency scaling," in *FCCM*, April 2012, pp. 109–116.

[14] ——, "Smi: Slack measurement insertion for online timing monitoring in fpgas," in *FPL*, Sept 2013, pp. 1–4.

[15] J. M. Levine, E. Stott, and P. Y. Cheung, "Dynamic voltage ; frequency scaling with online slack measurement," in *FPGA*, 2014, pp. 65–74.

[16] J. Nunez-Yanez, "Energy proportional computing in commercial fpgas with adaptive voltage scaling," in *FPGAworld*, 2013, pp. 6:1–6:5.

[17] 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, Dec. 2012.

[18] I. G. Harris, P. R. Menon, and R. Tessier, "Bist-based delay path testing in fpga architectures," in *ITC*, 2001, pp. 932–938.

[19] M. B. Tahoori and S. Mitra, "Application-dependent delay testing of fpgas," *TCAD*, vol. 26, no. 3, pp. 553–563, March 2007.

[20] E. Chmelaf, "Fpga interconnect delay fault testing," in *ITC*, Sept 2003, pp. 1239–1247.

[21] M. Abramovici and C. Stroud, "Bist-based delay-fault testing in fpgas," in *On-Line Testing Workshop.*, 2002, pp. 131–134.

[22] C. Chiasson and V. Betz, "Should fpgas abandon the pass-gate?" in *FPL*, 2013, pp. 1–8.

[23] Altera, "Cyclone iv device handbook," 2014.

[24] D. Lewis, V. Betz, D. Jefferson, A. Lee, C. Lane, P. Leventis, S. Marquardt, C. McClintock, B. Pedersen, G. Powell, S. Reddy, C. Wysocki, R. Cliff, and J. Rose, "The stratix^{TM} routing and logic architecture," in *FPGA*, 2003, pp. 12–20.

[25] E. Ahmed and J. Rose, "The effect of lut and cluster size on deep-submicron fpga performance and density," *TVLSI.*, vol. 12, no. 3, pp. 288–298, Mar. 2004.

[26] G. Lemieux and D. Lewis, "Using sparse crossbars within lut clusters," in *FPGA*, 2001, pp. 59–68.

[27] "Quartus-ii university interface program," 2009. [Online]. Available: http://www.altera.com/education/univ/research/unv-quip.html, 2009

[28] B. Gojman and A. DeHon, "Grok-int: Generating real on-chip knowledge for interconnect delays using timing extraction," in *FCCM*, May 2014, pp. 88–95.

[29] B. Gojman, S. Nalmela, N. Mehta, N. Howarth, and A. DeHon, "Grok-lab: Generating real on-chip knowledge for intra-cluster delays using timing extraction," in *FPGA*, 2013, pp. 81–90.