Trading Off Temperature Guardbands via Adaptive Approximations

Behzad Boroujerdian*, Hussam Amrouch†, Jörg Henkel†, Andreas Gerstlauer*
*Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX, USA
†Karlsruhe Institute of Technology, Chair for Embedded Systems (CES), Karlsruhe, Germany
Email: behzadboro@utexas.edu, amrouch@kit.edu, henkel@kit.edu, gerstl@utexas.edu

***Abstract*—Runtime circuit delay variations due to degradation effects like temperature are traditionally protected against using worst-case timing guardbands. Such an approach leads to a permanent performance overhead even though effects may only be transient. Recently, approximate computing has been proposed as a technique to trade off quality for various metrics. Existing approaches, however, do not target reductions in circuit delays and guardbands, or have only been applied statically. In this paper, we propose a novel design paradigm in which adaptive approximations are employed to *dynamically* trade off transient, degradation-induced variations in circuit delays and associated worst-case timing guardbands for permanent performance improvements with minimal quality loss. A key challenge is to design circuits that exhibit a significant delay profile across approximation levels while maintaining a high base performance. To achieve that, we introduce and implement two approaches for synthesizing arbitrary dynamic quality-versus delay-configurable circuits at fine temporal and spatial granularities while exploring associated area, speed and quality trade-offs. We apply our approach specifically to temperature variations and guardbands. Results for an IDCT image decoding example show up to 21% speedup with less than 2% area and energy impact compared to traditional guardbanding while maintaining a worst-case transient PSNR of at least 39 dB.**

I. Introduction

Designers traditionally employ worst-case timing guardbands to overcome runtime degradations, e.g. due to temperature. Such degradations increase circuit delays. To compensate, the design's clock period T_{clk} is typically extended by a timing guardband T_{GB} that accommodates for the maximum delay increase that can be induced by degradations [8]:

$$T_{clk} = T_{CP}(\alpha_{max}) = T_{CP}(\alpha_{nom}) + T_{GB}, \qquad (1)$$

where T_{CP} denotes the critical path delay under worst-case or nominal degradation conditions α_{max} and α_{nom}, respectively. Guardbands lead to a reduced frequency and hence performance loss. Many approaches to narrow guardbands have been introduced in the past, but they typically come at the cost of other circuit overheads, such as transistor size and area.

For inherently error-tolerant applications, such as in image processing or machine learning, approximate computing has emerged as a paradigm to trade off quality losses for various metrics, such as power, performance or temperature [5]. More recently, approximate computing principles have also been applied to target critical path delay and timing guardband reductions in the specific context of aging [3]. A simple approach to do so is to run the design at a faster than worst-case clock $T_{clk} < T_{CP}(\alpha_{max})$ while accepting resulting timing errors. Although this method enables performance improvements, timing errors due to insufficient guardbands have been shown to result in non-deterministic and unacceptable quality losses already for small guardband reductions [3].

To reduce clocks and trade off guardbands for deterministic and controlled quality losses, dedicated logic approximations can instead be applied. Lowering quality q, e.g. through precision scaling in arithmetic units can be used to reduce logic complexity and critical path delays, which can in turn be exploited to run the design at a faster clock

$$T_{clk} = T_{CP}(\alpha_{max}, q) < T_{CP}(\alpha_{max}, q_{exact}) \qquad (2)$$

compared to an exact realization at full precision q_{exact}. At the same time, the absolute delay increase due to degradations is smaller and remaining guardbands are reduced. Such an approach thus trades off guardbands for quality losses. However, existing work applied such tradeoffs only statically at design time [3]. Although providing a speedup, the permanent quality loss associated with a static approach has several disadvantages: (1) it sacrifices quality even with no degradations present, and (2) it is limited to small degradations requiring only small and thus permanently tolerable quality losses, as is the case for aging as demonstrated in their work [3].

In this paper, we instead propose to *dynamically apply approximations* and *adaptively lower quality* in order to *compensate for transient degradation-induced critical path delay increases under permanently reduced timing guardbands*. In our approach, such occasional lowering of quality will be imposed only as necessary, and limited only to periods in which larger disturbances are experienced at runtime in return for a permanent speedup. This makes our approach applicable to a wider range of potentially significant disturbances. In this paper, we specifically target temperature-induced degradations. For typical technology parameters in which reverse conditions do not occur [17], temperature variations can result in slowdowns of up to 42% (at 75 °C compared to 25 °C) [2]. At the same time, as opposed to the permanent aging, their transient nature allows for a temperature-aware runtime with the capability to adaptively compensate for such slowdowns through equivalent quality (and hence delay) reductions.

Approximations have been dynamically and adaptively applied in the past to target metrics including power and temperature [13], [12]. However, instead of lowering quality to

reduce temperature, our aim is to adapt to temperature-induced degradations in circuit delays by dynamically lowering quality in exchange for reduced critical path delays. This can in turn also help to limit further temperature increases and degradations similar to prior work. Achieving such dynamic delay vs. quality configurable designs, however, is more challenging than targeting other metrics: (1) Datapaths need to be designed in a way that makes dynamic delay scaling a possibility. Lowering of quality must result in a reduced *dynamic* worst-case path delay of the circuit that is able to compensate for the increase in circuit delays due to degradations; (2) The delay overhead associated with dynamic configurability must not exceed the delay reductions achieved due to quality scaling. Specifically, while delay configurability is easy to achieve in simple circuit structures, such as ripple-carry adders, base delays of configurable designs must be competitive in terms of achievable clock rates; and (3) Area and power overheads associated with such a design need to be justifiable.

We make the following contributions towards these goals: (1) We propose and implement two approaches to synthesize quality- and delay-configurable variants of arbitrary basic datapath units at a range of different speed, area and energy trade-offs: (i) a duplication approach that is general in supported approximation techniques and combines instantiation of datapath units at different quality levels with appropriate muxing, control and glue logic, and (ii) an approach for synthesizing standalone datapath units that share logic but provide dedicated delay profiles across different quality modes.

(2) Using such dynamically quality- and delay-configurable components, we propose a methodology to synthesize complete RTL designs that support dynamic adaptation at different quality, delay and degradation levels.

The remainder of this paper is organized as follows: Section II first presents an overview of the related work. Sections III and IV then describe our duplication and standalone approaches for component and entire register-transfer level (RTL) designs, respectively. Section V applies the provided techniques to a real life IDCT design, and further conducts a comparative analysis of the two approaches on this design. Finally, Section VI lists our conclusions and future plans.

II. Related Work

Various approaches have been proposed to reduce guardbands [8]. Our work is orthogonal to existing design time solutions that loose the potential for runtime adaptivity. Several approaches compensate for degradations by dynamically adjusting frequency or voltage. For example, in [2], temperature-aware cell libraries are created and employed to perform static timing analysis at different circuit temperatures. Such analysis is used later to optimize and adapt the timing guardband dynamically. However, switching frequency at runtime can take several cycles and still incurs performance overheads.

The idea to apply approximations in the context of guardbanding was recently introduced in [3]. As mentioned before, however, their approach replaces guardbands with equivalent permanent quality reductions statically at design time. By contrast, our approach is aimed at minimizing quality losses by applying quality scaling dynamically and adaptively in the presence of transient and temporary circuit degradations.

Approximate computing has been employed in many forms at the hardware level [5]. A wide range of approaches have been proposed to improve performance or reduce energy through approximations in basic arithmetic units, such as adders and multipliers [6]. However, in most existing approaches, the accuracy is set at design time and runtime configurability is lacking. The work in [14], [10], [11] proposes general accuracy-configurable adder and multiplier designs, but they remain focused on static design-time exploration for specific component types.

Ye *et al.* [18] and Kahng *et al.* [7] proposed dynamically accuracy-configurable adders. This, however, requires redundant units or is limited to specific architectures in addition to the overhead for configuration. Furthermore, their work only targets adders. By contrast, our approach is general and can be applied to any arithmetic unit. The authors in [13], [16], [15], [9] propose approaches to arrive at quality configurable units for general designs. However, their work targets energy, not delay. These approaches incur significant delay penalties or do not provide delay benefits under approximations, which makes them unsuitable for guardband trade-offs. The work in [12] uses input-dependent approximations to reduce temperature under quality goals. It similarly does not target circuit-level delay reductions in reaction to temperature changes.

III. Quality-Configurable Component Synthesis

In this section, we explain two different approaches used to arrive at quality-configurable arithmetic units (with deterministic precisions) as the building blocks of our systems. We aim to design components and datapath stages that can operate at M quality levels q_m with different worst-case dynamic path delays T_{DP} experienced at runtime. The clock that each component can be operated at will then be determined as:

$$T_{clk} = \max_m T_{DP}(\alpha_m, q_m). \tag{3}$$

Here, α_m denotes the degradation point at which the design will switch from quality level q_m to the next lower one q_{m-1}. In other words, α_m specifies the maximal amount of degradation that is tolerated and experienced by the design for it to stay in quality level q_m, i.e. until it switches to the next lower quality. Note that at the lowest quality (q_0), the design needs to be able to sustain worst-case degradations, i.e. $\alpha_0 = \alpha_{max}$. Without loss of generality, we simplify and limit discussions to designs with two quality levels for the rest of the paper. Using only exact (E) and approximate (A) modes, Eq. (3) becomes:

$$T_{clk} = \max(T_{DP}(\alpha_s, q_E), T_{DP}(\alpha_{max}, q_A)), \tag{4}$$

where α_s defines the switching point between modes.

To maximize achieved speedup, we aim to minimize T_{clk}. We can mainly focus on the delay in approximate mode as the primary determiner of the clock in Eq. (4). This is because its worst-case is determined by α_{max}, which is an external

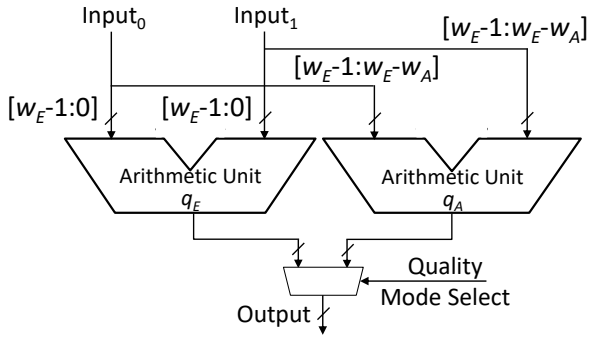


Figure 1: Duplication example with $q_A < q_E$. w_E and w_A denote exact and approximate unit bitwidths, respectively.

constraint on the design. By contrast, we can freely set α_s to guarantee that the delay in exact mode will never exceed the approximate mode delay. In practice, we want to maximize quality and α_s, i.e. stay in exact mode as long as possible. Hence, we set α_s such that $T_{DP}(\alpha_s, q_E) = T_{DP}(\alpha_{max}, q_A)$.

In the following, we describe two approaches for synthesizing components with different quality modes. An approach using duplicate but optimal component instances for each mode will result in the best delay trade-offs, but suffers from energy and area overheads. We propose an alternative approach for synthesizing a standalone component instance with different delay and quality modes. This allows for circuit sharing across quality levels resulting in lower area and energy overheads but also worse delay profiles compared to duplication. In all cases, we compare against a traditional design that is aggressively synthesized and optimized for best delay and minimal guardbands.

A. Duplicate Design

Duplication is capable of generating design points at different quality and delay trade-offs allowing for configurability at runtime. In such an approach, multiple units of different quality levels are muxed together as shown in Fig. 1, where the subunit selected by the mux determines overall output quality and delay. This approach is general in approximation techniques for each subunit. Without loss of generality, we use truncation of least significant bits (LSBs) for this example. Duplication and muxing-based approaches have been applied for quality configuration in the past [13], [15], [18]. However, existing work does not target configurability for delay, where the challenge is in achieving both a fast base delay and a significant delay profile between the different units and modes.

A simple setup instantiates pre-synthesized subunits connected by a mux. This ignores opportunities for the synthesis tool to optimize across subunit and mux boundaries. An approach in which the whole design is synthesized together is desired instead. To arrive at a design with different delays across modes, each subunit needs to be constrained separately. Since external input nets are in general shared, we need to apply constraints at the level of internally separated subunit inputs. We force the synthesis tool to not flatten the design, i.e. keep subunits decoupled, and then apply constraints to each subunit separately. Note that this will also prevent the synthesis algorithms from sharing logic across units, which allows minimizing delays associated with each unit and mode separately. Due to interferences, sharing of logic otherwise has a negative impact on delay trade-offs between modes. Further details on optimization of units are in Section III-B.

Algorithm 1 Dual Mode Synthesis

```
1: function ITERATIVE_SEARCH(RTL, C_{E,U},C_{E,L}, C_{A,U},C_{A,L})
2:     for C_E := C_{E,L} to C_{E,U}  step : sweep_step_size do
3:         Net_C ← Ext_BS(RTL, C_{A,L}, C_{A,U}, C_E)
4:         Net_bests.append(Net_C)
5:     end for
6:     Net_pareto ← get_pareto(Net_bests)
7: end function
8:
9: function EXT_BS (RTL, C_{A,L}, C_{A,U}, C_E)
10:    Net_C ← synthesize(RTL, C_E)
11:    Net_best ← Net_C
12:    while C_{A,L} ≤ C_{A,U} do          ▷ iterate while bounds not equal
13:        C_A ← avg(C_{A,L}, C_{A,U})
14:        for counter ∈ {0, ..., R − 1} do
15:            Net_C ← resynthesize(Net_C, C_E, C_A)
16:            T_A ← report_timing(Net_C)
17:            if T_A ≤ T_{best_A} then
18:                Net_best ← Net_C
19:            end if
20:            if T_A ≤ C_A then     ▷ if timing met,update upper bound
21:                C_{A,U} ← T_A
22:                break
23:            end if
24:        end for
25:        if T_A>C_A then ▷ if timing didn't meet, update lower bound
26:            C_{A,L} ← T_A
27:        end if
28:        Net_c ← Net_best
29:    end while
30: end function
```

1) Duplicate Synthesis

We apply an iterative approach to synthesize a duplicate design while minimizing delays in each mode. Assuming degradations affect circuit delays monotonically [2], we minimize delays under nominal conditions α_{nom}, which minimizes delays at any α. We denote the delay in approximate and exact mode under nominal degradations as $T_A = T_{DP}(\alpha_{nom}, q_A)$ and $T_E = T_{DP}(\alpha_{nom}, q_E)$, respectively. To find the actual T_{clk} and α_s associated with a design, delays are later re-characterized under actual degradations.

Algorithm 1 shows the pseudo code of the iterative search used to explore the timing constraint space across both modes. The *Iterative_Search()* function takes as its inputs upper (U) and lower (L) search bounds denoted as $C_{E,U}$ and $C_{E,L}$ for exact and $C_{A,U}$ and $C_{A,L}$ for approximate modes. We find these bounds using a preprocessing step. To determine lower bounds $C_{E,L}$ and $C_{A,L}$, we use the lowest delay that we can find when synthesizing a standalone subunit at the desired level of accuracy[1]. We set $C_{A,U} = C_{E,L}$ because a

[1]A binary search similar to the one presented later is used to find designs.

Table I: Traditional guardbanding vs. duplicate approach.

Metric		Adder		Multiplier	
		Trad.	Duplicate	Trad.	Duplicate
T_{DP} [ps]	T_A	149	167 (12.1%)	519	485 (-6.6%)
	T_E		175 (17.4%)		553 (6.6%)
T_{clk} [ps]		211	237 (12%)	737	690 (-6%)
Energy [pJ]	q_A	0.25	0.51 (103%)	11.2	14.3 (27%)
	q_E		0.56 (123%)		17.3 (54%)
Area [um^2]		532	1,080 (103%)	9,038	14,203 (57%)

quality-configurable design whose approximate delay exceeds the delay of a standalone exact design is not worth using (since the latter would then win both in quality and delay at all times). For $C_{E,U}$, we use the exact mode's best delay at α_{max} since otherwise the speedup would be less than 1.

Exact Constraint Exploration: The top-level *Iterative_Search()* function sweeps the timing constraint space associated with the exact mode, invoking an extended binary search (*Ext_BS()*) per iteration to find a design with the lowest delay associated with both modes. The *sweep_step_size* is thereby chosen according to the number of exact mode constraints to be explored.

Approximate Constraint Exploration: The *Ext_BS()* function finds the lowest delay associated with the approximate mode given a constraint on the exact mode. Approximate mode bounds passed to the top-level search determine the search space. A binary search is used to quickly narrow down upper and lower constraint bounds. Within each iteration, an inner loop is used to iteratively re-synthesize the gate-level netlist R times to arrive at an optimal design for the given constraints.

Logic Synthesis and Static Timing Analysis: In each iteration of the inner loop, the current gate-level netlist (Net_C) is re-synthesized given a set of constraints and the delay information for the generated design is reported. As a starting point, the unit's RTL is first synthesized using only its exact mode constraints (line 10). Within the loop, the *resynthesize()* step (line 15) then uses the previously generated gate-level netlist, imposes a separate constraint on each subunit, and resynthesizes the design. Finally, a *report_timing()* function returns the delay associated with the approximate mode in the generated netlist of the design to feed back into the binary search.

The complexity of the synthesis algorithm is $O(kRn \log m)$, where n is the number of exact mode constraint steps explored in the outer *Iterative_Search()* ($n = \lfloor \frac{C_{E,U} - C_{E,L}}{sweep\_step\_size} \rfloor$), m is the number of approximate mode constraints with distinct delay results in the inner binary search, and k depends on the complexity of the logic synthesis and timing analysis step as a function of the design size.

2) Duplicate Results

Table I shows results targeting temperature degradations of up to $\alpha_{max} = 75\,^{\circ}\mathrm{C}$ for duplicate multiplier and adder designs as compared to a traditional guardbanding approach (Trd) synthesized for best $C_{E,L}$. The designs use 32x32 and 24x24 bits in exact and approximate mode, respectively. Energy and worst-case critical/dynamic path delays T_{DP} are reported for reference at a nominal $\alpha_{nom} = 25\,^{\circ}\mathrm{C}$. By contrast, T_{clk} is the clock period that guarantees operation across the whole temperature range according to Eqn. (1) or (4). While the nominal delay in exact mode is larger than in traditional designs, the delay in approximate mode at high temperatures should be lower than a corresponding guardbanding. Using the duplicate approach, we achieve such a 6% speedup for the multiplier. In case of the adder, however, no speedup is possible. The mux and glue logic overheads exceed any delay reductions in the approximate subunit. Furthermore, the duplication of subunits incurs energy and area overhead. At the same time, the duplication approach allows for independent and arbitrary designs and approximation techniques to be employed in exact versus approximate subunits. Energy in approximate mode (q_A) is generally smaller than in exact mode (q_E) since we assume that the LSB inputs are approximated to zero in this case and do not change.

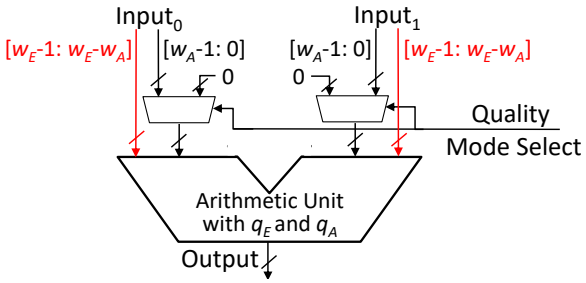


Figure 2: Standalone quality-configurable design approach.

B. Standalone Design

Reducing the redundancy and the overhead associated with the duplicate approach requires sharing part of the data path across different accuracy levels. This is possible for arithmetic units, such as adders and multipliers, in which one can embed an entire level of accuracy within another given the right precision scaling technique, such as LSB truncation or rounding. For example, a 32x32 multiplier can be turned into an approximated 24x24 multiplier by simply muxing the 8 input LSBs to zero and only using the upper 48 output MSBs (Fig. 2). In practice, we replace muxes, reduce overhead and provide equivalent functionality by raising (and maintaining) the reset signal of the LSB bits of the connected input registers. In theory, this should also reduce the delay since critical paths in arithmetic circuits start from LSBs, and such paths are disabled when hardwired[2]. In reality, however, synthesis tools will normally strive to balance all paths and use available slack to optimize the gate netlist for area and energy such that all paths have similar delays [4]. As such, no delay gain is obtained when muxing input LSBs to zero without changes to the synthesis flow.

1) Standalone Synthesis

Similar to the duplicate case, we therefore propose an approach in which standalone units are synthesized under different constraints for each mode. We again combine an iterative search space exploration with logic synthesis using

[2]Note that in the cycle switching to approximate mode, the worst-case will be determined by all paths until changes have rippled through the muxes.

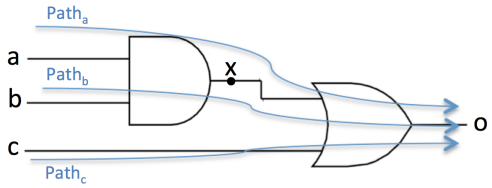


Figure 3: Example of gate-level logic with path sharing.

industry-standard tools. The search algorithm is the same as presented in Algorithm 1. However, the complexity of imposing two constraints on a shared design requires an extended and more complicated logic synthesis step.

Logic Synthesis and Static Timing Analysis: As before, the purpose of this step is to generate a gate-level netlist given a set of constraints and report delay information for the generated design. The main difference is in finding the correct paths corresponding to the approximate mode in order to impose a tighter constraint on them. In the duplicate approach, paths were confined to the subunits for each mode and thus easily separated. By contrast, in a shared design, approximate paths are entirely embedded within the exact ones. Since switching to a lower precision means forcing LSBs to zero, only paths starting from non-zeroed inputs (shown in red in Fig. 2) can possibly be on the critical approximate paths once muxes have settled. Since such paths can be easily identified, we can impose the tighter approximate constraint on them externally while treating the design as a black box.

However, some of these paths can be further excluded from being optimized for approximate mode to release pressure during synthesis. As is well known and shown for the simple example in Fig. 3, paths emanating from different inputs can internally re-converge and partially share the same logic. If input a in this example is externally muxed to zero, output x of the AND gate will be forced to zero, the path from input b will be disabled, and only paths through other inputs (e.g. c) will contribute to delays. We provide a method to precisely guide the synthesis tool to find only those paths contributing to the approximate mode while discarding any such paths that are internally masked.

We first derive a version of the design in which the approximate mode is emulated. To do so, approximated LSBs are hardwired to zero directly in the gate-level netlist of the design. This involves parsing the netlist and introducing *assign to zero* statements appropriately. Our algorithm then uses the static timing analysis of the synthesis tool to request the delay associated with the paths going through each cell in the design. The synthesis tool will in turn report any cells whose output ends up being hardwired due to the input zeroing. We modify Algorithm 1 to consider any paths going through such cells as masked and discard them. Hardwiring of LSB inputs is removed and tighter delay constraints are only imposed on the undiscarded paths (i.e., paths going through cells whose output is changing) when the design is synthesized. We perform this selection for a gate-level netlist Net_C as part of the *resynthesize()* step in Algorithm 1.

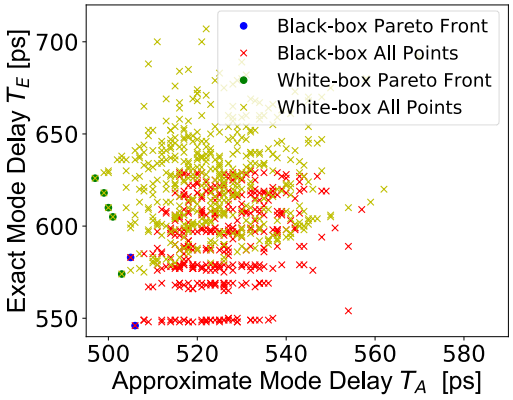


Figure 4: Standalone multiplier delay trade-offs.

2) Standalone Results

Fig. 4 shows the trade-offs between exact and approximate mode delays obtained for the example of a 32x32/24x24 multiplier using our approach. We plot the design points and the resulting Pareto fronts found by our iterative search with (white-box) and without (black-box) discarding internally masked paths. In case of standalone units, there is a clear Pareto trade-off between approximate and exact delays. Due to remaining logic sharing between unmasked approximate and exact paths, optimizing paths associated with the lower precision mode generally increases delays on exact paths, and vice versa. Note that with inputs and subunits separated in the duplicate case, there is no logic sharing and such trade-offs do not exist[3]. The figure shows that using a white-box approach reveals extra points on the Pareto front not found by a pure black-box approach. This is because the white-box method allows for more paths to be discarded, and thus more opportunities for approximate mode optimization.

Table II and Table III summarize results for standalone 32x32/24x24 adder and multiplier designs targeting temperature degradations of up to 75 °C as compared to a traditional guardbanding approach. For each standalone design, we report selected points I through IV on the Pareto front. We are able to achieve a 8% and 4% speedup for the adder and multiplier, respectively which is less than that of the duplicate approach in case of the multiplier. Note that the standalone approach still incurs area and energy overhead to support dynamic configurability, but overheads are significantly smaller than in the duplicate approach. At the same time, in case of the multiplier, significantly reduced switching activity from LSB gating results in overall energy savings in approximate mode. Final energy consumption will be determined by the fraction of time the system spends in exact versus approximate mode.

Since approximate delays determine the clock (and hence speedup) while exact delays determine switching points, the trade-off between delays translates into a trade-off between speedup and switching point. These trade-offs are shown for

[3]A duplicate black-box approach that only constrains external inputs would synthesize the subunits under both constraints resulting in similar trade-offs.

Table II: Traditional vs. standalone approach (adder).

Metric		Trad.	Standalone Configurable			
			I	II	III	IV
T_{DP} $[ps]$	T_A	149	134 (-10%)	135 (-9.4%)	137 (-8.1%)	141 (-5.4%)
	T_E		180 (20.8%)	176 (18.1%)	170 (14.1%)	164 (10.1%)
T_{clk} $[ps]$		211	194 (-8%)	195 (-7.5%)	196 (-7%)	202 (-4%)
Energy $[pJ]$	q_A	0.25	0.285 (13%)	0.277 (9%)	0.296 (17%)	0.293 (16%)
	q_E		0.336 (33%)	0.320 (26%)	0.347 (37%)	0.354 (40%)
Area $[um^2]$		532	688 (29%)	650 (22%)	697 (31%)	711 (34%)

Table III: Traditional vs. standalone approach (multiplier).

Metric		Trad.	Standalone Configurable			
			I	II	III	IV
T_{DP} $[ps]$	T_A	519	497 (-4.2%)	499 (-3.9%)	503 (-3.1%)	506 (-2.5%)
	T_E		626 (21%)	618 (19.1%)	574 (10.6%)	546 (5.2%)
T_{clk} $[ps]$		737	707 (-4%)	708 (-4%)	714 (-3%)	720 (-2%)
Energy $[pJ]$	q_A	11.2	10.944 (-3%)	10.471 (-7%)	10.689 (-5%)	8.85 (-21%)
	q_E		15.83 (41%)	14.931 (26%)	15.109 (35%)	11.79 (5%)
Area $[um^2]$		9038	13982 (55%)	12784 (41%)	13278 (47%)	9589 (6%)

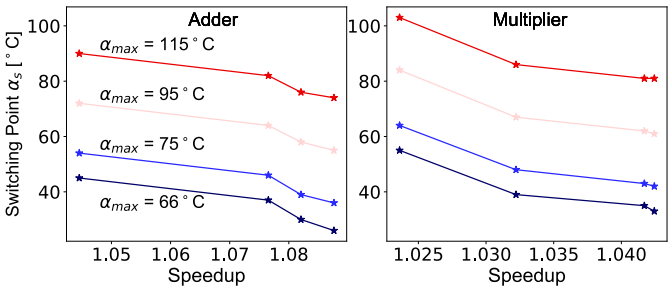


Figure 5: Speedup vs. switching temperature trade-offs.

the adder and multiplier in Fig. 5. As can be observed, the higher the speedup, the lower the temperature at which the design must switch to approximate mode. Note that in this figure, each line represents a different maximum temperature that the design is expected to tolerate.

Overall, in comparison to duplication, the standalone approach achieves comparable or better speedups with lower area overhead and potential energy savings. However, it does not support independent unit designs for each mode and is limited to LSB truncation or rounding as approximation technique.

IV. Degradation-Adaptive RTL Design

In the following, we explain how configurable units synthesized using one of the approaches in Section III can be employed as building blocks of the entire RTL designs that can adapt to degradations dynamically. One can either apply the iterative search described in Algorithm 1 on the entire design or synthesize each configurable unit in isolation and then introduce them into the design. Although the former approach is simple and can be applied with only minor changes to the algorithm, it requires repeated resynthesis of the complete design and the result will only have as many modes as provided by each configurable unit. By contrast, an isolated approach, although more involved, can provide a finer granularity of up to P^2 quality levels and switching points for a design with P units each with 2 modes. Furthermore, each iteration of synthesizing an isolated unit has significantly reduced synthesis complexity. However, since the total number of combined iterations is larger, an overall runtime comparison depends on the specific design space.

A. Synthesis of the Entire Design

This approach applies the *Iterative_Search()* from Algorithm 1 to the entire design. This means that the combined RTL of the design is explored and repeatedly resynthesized. Doing so requires minor changes to be applied to the logic synthesis and timing analysis step described in Section III. The *resynthesize()* step needs to be modified to apply the constraints to every datapath stage whether that stage contains a quality-configurable unit or not. Non-configurable stages thereby need to be synthesized to meet the tightest constraints. This means applying the approximate constraint (C_A) on quality-configurable units as well as all non-configurable stages, while applying the exact constraint (C_E) only to the configurable units. The *report_timing()* step also needs to be adjusted to report the wort-case delay among all the non-configurable stages and all the configurable stages in approximate mode. Since the same constraints are applied to all configurable units, they will end up with similar delays, clocks T_{clk} and switching points α_s, where the worst case among all components will determine the clock and switching point for the whole design. In other words, individual units will all switch at the same time and α, i.e. for a design with two modes, all quality-configurable units will either be in exact or approximate mode at any given point in time.

B. Synthesis in Isolation

This approach allows for the design of systems supporting a combination of modes across different units, i.e. systems in which units can switch modes independently. This requires exploring the design space for each configurable unit in isolation following one of the approaches in Section III. Units are then combined together in an optimal fashion and embedded back into the design to arrive at a complete configurable system as Fig. 6 demonstrates.

Preprocessing: A preprocessing step similar to the one in Section III-B is used to find the bounds $C_{A,U,i}$, $C_{A,L,i}$, $C_{E,U,i}$ and $C_{E,L,i}$, associated with each configurable unit i.

Finding Pareto Fronts: This step iterates over the configurable units in the design and generates Pareto fronts PF_i (similar to Fig. 4) associated with each unit i. This is achieved by calling the *Iterative_Search()* in Algorithm 1 with bounds obtained from the preprocessing step and the unit's RTL_i.

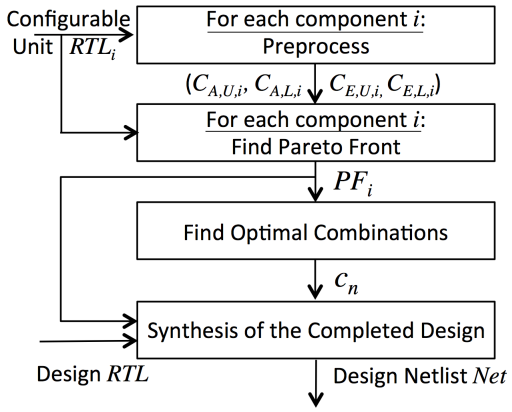


Figure 6: Synthesis in isolation.

As a result, each unit's Pareto front PF_i contains a list of design points j associated with an optimal netlist $Net_{i,j}$ and its approximate delay $T_{A,i,j}$.

Finding Optimal Combinations: We then find optimal combinations of individual unit design points to arrive at a list of optimal configurations for the configurable portion of the design. Each configuration c_n is given as a mapping $c_n(i)$ that selects a particular design point j_n for each unit i.

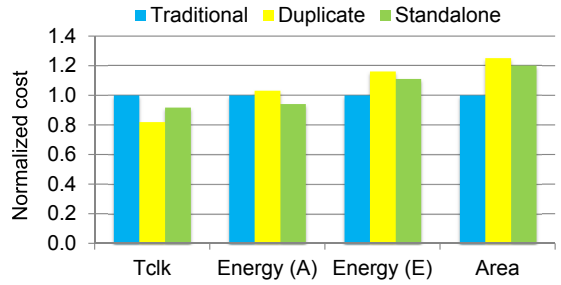
Finding possible c_n involves iterating over each point j of every unit i. In each iteration $n = (i, j)$, we use the j^{th} design point of unit i as the seed for a new configuration c_n in which $c_n(i) := j$. We then find design points $c_n(k)$ for every other unit $k \neq i$ such that their approximate delay $T_{A,k,c_n(k)}$ is closest to the seed unit's (i) approximate delay $T_{A,i,c_n(i)}$ without being larger. We discard configuration c_n if such a point can not be found for any of the other units.

Note that the reason behind selecting the closest $T_{A,k}$ to the seed's $T_{A,i}$ is that for two design points associated with a unit, the one whose T_A is the smallest (and hence further away from the T_A of interest) has the largest T_E and hence less tolerance for disturbance, rendering it less optimal.

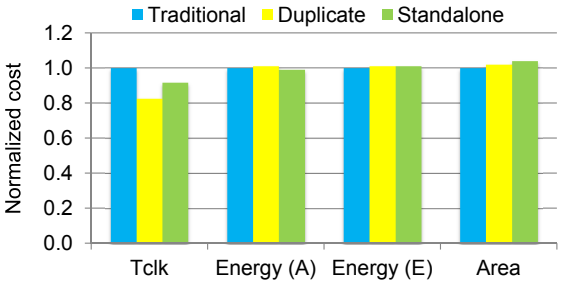
Synthesis of the Completed Design: This final step completes the synthesis of optimal configurations for the entire design. For each mapping c_n found in the previous step, the selected netlists $Net_{i,c_n(i)}$ for each unit i are embedded back into the design, and the resulting design is synthesized using the worst-case $\max_i(T_{A,i,c_n(i)})$ as the constraint with configurable units in approximate mode. The final clock of the design is determined based on synthesis results, and a post-processing step is used to find the switching points $\alpha_{s,i}$ for each configurable unit i within the design.

V. Experiments and Results

We apply our approach to the design of an IDCT block for JPEG imaging applications under temperature degradations. We use a fixed-point IDCT with 15-bit coefficients and a 26x15 multiply-accumulate (MAC) unit in exact mode. We allow for up to 8 bit of LSB truncation while maintaining



(a) Multiplier



(b) Whole IDCT

Figure 7: Our IDCT compared to traditional guardbanding in terms of achieved clock (T_{clk}), energy in exact (E) and approximate (A) mode, and area.

acceptable image quality. As such, the precision of the MAC unit is lowered to 18x7 bits in approximate mode. The MAC unit is pipelined and only the multiplier stage is synthesized to be configurable. The adder stage is not on the critical path and can remain in exact mode at any temperature. Configuration of the MAC unit and IDCT block is assumed to be controlled by existing power and temperature monitoring and management circuits in the system [2].

We apply an approach synthesizing either duplicate or standalone units in isolation as described in Section IV. We again compare results against a traditional design aggressively optimized for minimal delay and guardbands. We use Synopsys DesignCompiler with the "ultra compile" option as synthesis and static timing analysis backend, We let DesignCompiler synthesize fast tree-based adders and multipliers from behavioral "+"/"*" operators, where traditional and our designs are constrained to achieve the best $C_{E,L}$ and performance. We use the temperature-aware cell libraries from [1], [2] for synthesis and timing characterization under different temperature conditions. The libraries are based on the 45nm NanGate open cell library and contain delay and power information of every cell under different temperatures. We target a range of $\alpha_{nom} = 25\,°\text{C}$ to $\alpha_{max} = 75\,°\text{C}$, for which a traditional IDCT design requires a $T_{clk} = 833\,\text{ps}$ including a minimal 240 ps guardband.

Fig. 7 shows results for the 26x15 multiplier used in the IDCT including surrounding registers (Fig. 7a) and the complete IDCT design (Fig. 7b). Using a duplicate approach for the multiplier provides a speedup of 22%. By contrast,

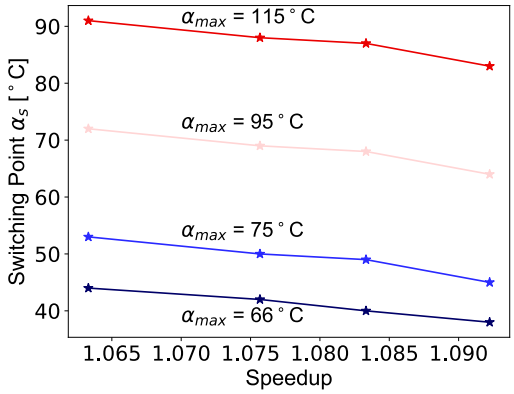


Figure 8: Standalone IDCT speedup vs. switching temperature.

the speedup using a standalone multiplier is 9%. However, duplication incurs a higher energy and area overhead at the component level, where a standalone multiplier provides 6% energy savings in approximate mode.

When incorporated into the complete IDCT design, similar speedups of 21% and 8.5% are maintained corresponding to T_{clk} of 687 ps and 767 ps, respectively. They are slightly lower due to the impact on register loads not considered during isolated synthesis. At the same time, the impact of component overheads to overall IDCT area and energy become negligible in all cases. Energy differences are less than 1%, and area overheads for the duplicate and standalone IDCT variants are 2% and 4%, respectively.

The speedup vs. switching temperature trade-offs using a standalone multiplier are shown in Fig. 8. By contrast, with a duplicate multiplier, the design requires switching from exact into approximate mode at $\alpha_s = 36\,°\text{C}$.

Finally, Fig. 9 compares the output of our IDCT design using a duplicate multiplier against a traditional design. Our design maintains a high image quality of 39 dB PSNR at maximal degradations with a 18% shorter cycle time and thus faster clock compared to a traditional guardbanding approach. By contrast, timing errors for the traditional design using the same T_{clk} as in our approach lead to catastrophic image failures. Note that the standalone approach provides the same image quality results with a T_{clk} of 767 ps.

VI. Summary and Conclusions

Designers traditionally use worse-case guardbanding to battle run time variations. This approach leads to a permanent performance overhead. In this work, we propose a new design paradigm to dynamically and adaptively trade-off timing guardbands for quality losses. Such a paradigm requires runtime configurable units capable of switching to a lower precision and latency. We introduced two approaches to implement such units integrated into complete RTL designs. Results for an IDCT example under temperature degradations demonstrate more than 20% speedup for a worst-case quality loss of 20%. In future work, we intend to apply our approach to other designs with more than two levels of configurability.

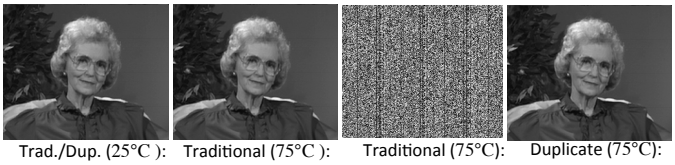


Figure 9: Quality vs. cycle time for different approaches.

Acknowledgments

This work is partially supported by National Science Foundation (NSF) Grant CCF-1337393, a Humboldt Research Fellowship and the German Research Foundation (DFG) priority program "Dependable Embedded Systems" (SPP 1500). We thank Ku He for the original IDCT example and the anonymous reviewers for their comments to improve the paper.

References

[1] Temperature-aware cell libraries. http://ces.itec.kit.edu/dependable-hardware.php.
[2] H. Amrouch et al. Optimizing temperature guardbands. In *Design, Automation and Test in Europe (DATE)*, 2017.
[3] H. Amrouch et al. Towards aging-induced approximations. In *Design Automation Conference (DAC)*, 2017.
[4] S. Ghosh et al. CRISTA: A new paradigm for low-power, variation-tolerant, and adaptive circuit synthesis using critical path isolation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 26(11):1947–1956, 2007.
[5] J. Han et al. Approximate computing: An emerging paradigm for energy-efficient design. In *European Test Symposium (ETS)*, 2013.
[6] H. Jiang et al. A review, classification, and comparative evaluation of approximate arithmetic circuits. *ACM Journal on. Emerging Technologies in Computing (JETC)*, 13(4):60:1–60:34, 2017.
[7] A. Kahng et al. Accuracy-configurable adder for approximate arithmetic designs. In *Design Automation Conference (DAC)*, 2012.
[8] J. Keane et al. An odomoeter for CPUs. *IEEE Spectrum*, 48(5), 2011.
[9] S. Lee et al. Fine grain word length optimization for dynamic precision scaling in DSP systems. In *International Conference on Very Large Scale Integration (VLSI-SoC)*, 2013.
[10] C. Liu et al. A low-power, high-performance approximate multiplier with configurable partial error recovery. In *Design, Automation and Test in Europe (DATE)*, 2014.
[11] J. Miao et al. Modeling and synthesis of quality-energy optimal approximate adders. In *International Conference on Computer-Aided Design (ICCAD)*, 2011.
[12] D. Palomino et al. Thermal optimization using adaptive approximate computing for video coding. In *Design, Automation and Test in Europe (DATE)*, 2016.
[13] A. Raha et al. Input-based dynamic reconfiguration of approximate arithmetic units for video encoding. *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, 24(3):846 – 857, 2015.
[14] M. Shafique et al. A low latency generic accuracy configurable adder. In *Design Automation Conference (DAC)*, 2015.
[15] S. Venkataramani et al. Quality programmable vector processors for approximate computing. In *International Symposium on Microarchitecture (MICRO)*, 2013.
[16] S. Venkataramani et al. Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits. In *Design, Automation and Test in Europe (DATE)*, 2013.
[17] D. Wolpert and P. Ampadu. Temperature effects in semiconductors. In *Managing Temperature Effects in Nanoscale Adaptive Systems*. 2012.
[18] R. Ye et al. On reconfiguration-oriented approximate adder design and its application. In *International Conference on Computer-Aided Design (ICCAD)*, 2013.