IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 21, 2019, accepted November 26, 2019, date of publication November 28, 2019,
date of current version December 12, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2956679

Automated Synthesis of Energy-Efficient Reconfigurable-Precision Circuits

DANIELE JAHIER PAGLIARI[1], (Member, IEEE), ENRICO MACII[2], (Fellow, IEEE), AND MASSIMO PONCINO[1], (Fellow, IEEE)
[1]Dipartimento di Automatica e Informatica, Politecnico di Torino, 10129 Turin, Italy
[2]Dipartimento Interateneo di Scienze, Progetto e Politiche del Territorio, Politecnico di Torino, 10129 Turin, Italy
Corresponding author: Daniele Jahier Pagliari (daniele.jahier@polito.it)

ABSTRACT In recent years, designers are trying to move part of the computing tasks involved in Internet of Things applications from the cloud to the edge. This imposes increasing performance demands on edge nodes, which usually clash with their limited energy budget. An effective workaround is to leverage hardware capable of varying its computational precision at runtime, which can provide "good-enough" results while significantly reducing energy consumption. Dynamic Voltage and Accuracy Scaling (DVAS) and its variants are particularly promising methods to implement such hardware, due to their general applicability and contained overheads. However, these methods are negatively affected by the optimizations performed by commercial Electronic Design Automation (EDA) tools. As a consequence, when applied within a standard design flow, they do not yield the expected results. This paper describes a synthesis tool that solves this issue, allowing the integration of reconfigurable-precision circuits based on DVAS in standard design flows based on commercial tools. Moreover, our tool can receive information about the relevance of different precisions for the target application(s) that will use the circuit, and further optimize the design accordingly. When applied to two realistic use cases (neural network inference and image compression), our tool reduces the total energy consumption of reconfigurable-precision circuits of 20-25% compared to a straight-forward application of DVAS.

INDEX TERMS Low-power design, approximate computing, energy-quality tradeoff.

I. INTRODUCTION

Emerging mobile and Internet of Things (IoT) applications impose stringent constraints on the performance and energy consumption of edge nodes. Increasingly complex tasks (e.g. machine learning based classification, multimedia processing, etc.) execute on cost- and power-constrained embedded devices, powered by batteries with limited capacity, and which should ideally support months or years of continuous operation with a single charge cycle [1].

As the end of Dennard's constant-field scaling approaches, such tight requirements can only be met by means of domain specialization [2]. One common feature of many modern application domains is *error resilience* (or *error tolerance*), defined as the capability of tolerating relaxations in the quality (i.e. accuracy or precision) of internal computations, without a significant impact on the quality of final results [1], [3], [4]. Error resilience stems from the characteristics of applications' inputs, outputs and internal algorithms. Multimedia applications produce outputs that are meant for humans, and can therefore exploit the limited perceptual capabilities of our sense organs to relax computation quality constraints. Machine learning tasks typically receive as inputs noisy sensors data, and process them by statistical/aggregative algorithms [3], [5]; therefore, these tasks can tolerate "errors" in computations as long as they are negligible with respect to input noise or can be filtered out by the algorithm.

Error tolerance can be exploited to trade-off computational quality and other design metrics, most commonly energy efficiency. In recent years, this concept has been applied at all levels of the computing stack, from single devices to software [1], [4], [6]. Examples of optimizations that explore this trade-off in specialized computing systems are found in accelerators for deep neural networks inference [7]–[13], or video processing [14]–[16]. Tolerance to errors is however typically time-dependent, and is affected both by the operation context (e.g. battery state of charge, input noise power)

The associate editor coordinating the review of this manuscript and approving it for publication was Sofana Reka S.

and by the input data (e.g. easy/difficult input to classify) [9], [10], [12], [13]. Moreover, domain-specific accelerators might still be used to execute multiple applications from the same domain, each with its own output quality requirements, at different times. Consequently, *energy-quality scalable* designs are desirable, i.e. components and systems able to dynamically alter their energy consumption and output quality at runtime [17]–[21].

At the hardware level, one of the easiest yet most effective ways to achieve multiple energy-quality points consists in tuning the data *precision* used for internal operations. In fact, reducing data precision simultaneously improves energy efficiency in both computing and memory, by allowing the hardware to perform operations on smaller bit-widths and cutting the required memory bandwidth [10], [13], [18], [22]. Dynamic Voltage and Accuracy Scaling (DVAS) and its variants are design techniques that combine precision and supply voltage scaling to improve energy efficiency in datapath circuits for energy-quality scalable systems [17], [18]. While being general and potentially very effective, however, the straight-forward application of these techniques conflicts with the optimizations performed by Electronic Design Automation (EDA) tools, thus yielding very limited energy benefits when applied within a standard integrated circuit design flow.

In this paper, which extends the work of [21], we describe a synthesis method that enables the design of precision and voltage scalable datapaths and is fully compatible and integrated with standard EDA tools. Our flow automatically identifies the optimal supply voltage for each target precision and constrains the synthesis accordingly, overcoming the limitations deriving from a straight-forward application of DVAS-like techniques. In doing so, information on the target application is leveraged to drive the optimization, thus making our method perfectly suited for domain specific accelerators.

This paper extends the original work of [21] as follows:

- Providing a more detailed description of the proposed flow and extending its application to the case of a DVAS variant called Dynamic Voltage, Accuracy and Frequency Scaling (DVAFS) [18].
- Evaluating its effectiveness on two realistic use cases, i.e. neural network inference [23] and image compression [14] and comparing it to that of a previous technology-specific synthesis method for energy-quality scalable datapaths based on runtime back-gate biasing [20].

Through our experimental results, we show that our method achieves $>$ 25% total energy savings with respect to a standard application of DVAS, while incurring a limited area overhead.

The rest of the paper is organized as follows. Section II analyzes existing work on energy-quality scalable circuits. Section III focuses on the limitations of DVAS/DVAFS and provides the motivation for this work. In Section IV we describe the proposed synthesis method, whereas in Section V we assess its performance through experimental results. Section VI concludes the paper.

II. BACKGROUND AND RELATED WORK

A. ARCHITECTURAL-LEVEL TECHNIQUES FOR ENERGY-QUALITY SCALABLE HARDWARE DATAPATHS

Early approaches to implement datapath circuits able to dynamically reduce their output quality in exchange for energy savings are based on *architectural* modifications. Most of these works focus on adders and multipliers, due to them being the fundamental building blocks of most complex datapaths, and propose modifications to their architectures to enable multiple *modes of operation*, i.e. energy-quality points.

The majority of the proposed architectures are designed following a common paradigm, based on two main concepts. First, a part of the logic is disabled or replaced by a simpler circuit when the hardware is operating in a low-quality mode, thus reducing its dynamic power consumption and critical delay. Second, the extra slack made available by the logic simplification is exploited to apply voltage scaling and further reduce both dynamic and leakage power.

In the case of adders, one popular technique is based on enhancing an *inexact* (or *approximate*) architecture, i.e. one that is modified at *design time* to produce lower-quality outputs in exchange for energy efficiency (for example by approximating the output LSBs). The inexact design is coupled with *error recovery* circuitry, selectively activated when high-quality results are needed, in the next clock cycle after the computation of the approximate output [25]–[28]. These designs incur critical latency penalties when operating in the highest quality modes, as well as significant area and power overheads due to the additional recovery logic. Another family of adder designs is based on selective *carry-chain segmentation*, in which the adder is split in sub-blocks and the carry signal from each sub-block to the next can be selectively replaced with the output of a simpler *carry prediction* circuit [28], [31]. More recently, an energy-quality scalable adder using both carry-chain segmentation and error recovery is proposed in [29], which allows a more flexible selection of quality levels at runtime. While effective, their implementation requires multiple clock cycles for accurate computations, which complicates its integration in a larger datapth, and incurs significant area overheads. Finally, an advanced energy-quality scalable adder is proposed in [30]. This solution implements carry prediction using part of the circuitry already present in each sub-adder, in order to reduce overheads. Moreover, the carry is accurately propagated through *couples* of sub-adders to improve output accuracy. Although this work allows accurate sum computation in a single clock cycle, it still incurs a significant area overhead compared to a standard adder. Moreover, the delay of the circuit at maximum precision also increases, proportionally to the number of quality-configurations made available at design time.

In the case of multipliers, one solution consists in selectively ignoring some of the partial products, as proposed in [32]. Alternatives, all partial products can be computed, but their accumulation can be implemented by means of inexact adders. In particular, in the work of [27], the accumulation tree is built with adders able to produce an approximate sum output and a secondary *correction output*. The latter can be selectively used, depending on the operating mode, to reduce the accumulation error, at the cost of additional energy consumption. This approach is improved in [33], where the approximate accumulation is made reconfigurable so to support more quality-configurations. However, the proposed implementation does not support fully accurate computations. In [34] a methodology for designing quality-configurable multipliers using a genetic algorithm is proposed. With this method, the authors are able to generate a large number of different Pareto-optimal multipliers, which however only support two quality configurations (approximate and accurate) and incur large delay overheads when performing accurate computations.

Many advanced variants of these basic schemes have been proposed in recent years, but their detailed description is out of the scope of this work. More in-depth information can be found in [4], [6], [35].

In general, energy-quality scalable circuits based on architectural modifications tend to lack generality, as each modification is specific to a particular family of circuits. For instance, [32] and [27] are limited to array multipliers only. Additionally, these techniques either enable only few operating modes (e.g. one accurate and one approximate mode) or tend to have significant power and area overheads when working at maximum quality [25], [26], [36]. Finally, recent studies have shown that these designs tend to consume more power than the results of a simple precision scaling obtained through bit-width truncation, for the same output quality [17], [36]. The intuition behind this result is that circuits based on architectural modifications still invest a significant amount of power to compute *erroneous* information of limited usefulness. Conversely, bit truncation completely *avoids* part of the computation, thus saving more energy.

B. DYNAMIC VOLTAGE ACCURACY (AND FREQUENCY) SCALING

Dynamic Voltage and Accuracy Scaling (DVAS) [17] and its variants are alternative methods to design energy-quality scalable circuits, that take into account the aforementioned limitations of architectural solutions. These techniques realize multiple quality-modes simply by *gating* some of the input LSBs to zero, thus reducing the circuit *precision*. DVAS obtains energy reductions by a twofold mechanism. First, zeroing-out some LSBs directly reduces the switching activity and hence the dynamic power of the circuit. Second, it also *deactivates* some of the timing paths in the circuit, thus (theoretically) reducing its overall delay, as the longest timing paths within most datapath architectures are those connecting input LSBs to output MSBs. Such reduction would in turn allow one to down-scale the supply voltage below the value used at maximum precision, while maintaining the same operating frequency, thus providing much more relevant dynamic and leakage power reductions [17]. An example of using precision reduction to tune the quality of computations is reported in [37] for a multiplier.

Despite its simplicity, DVAS has several advantages compared to the methods described in Section II-A. First, bit-width gating can be realized with a 1-bit granularity, hence allowing for a large number of quality modes. Second, this solution has practically zero power overheads when the circuit works in the maximum quality mode. Third, input gating can be applied, in principle, to any arithmetic operator, making DAS and DVAS much more general than previous architecture-based solutions.

As detailed in Section III, however, the aforementioned assumption on the relation between the overall circuit delay and the number of gated inputs stops being valid when the circuit is synthesized with a standard EDA flow. This dramatically reduces the energy benefits deriving from DVAS, as the supply voltage can only be reduced slightly, even at very low precisions.

Dynamic Voltage Accuracy and Frequency Scaling (DVAFS) [18], is a variant of DVAS, stemming from the observation that input gating in DVAS leaves a part of the circuit unused. While this part does not switch and hence only consumes static power, even further benefits could be obtained by reusing it for other computations. In practice, this is achieved by means of *sub-word parallel* operations. For example, when a 32-bit multiplier is working at a reduced precision of 16-bit, the LSB-part of the circuit is used to perform *another* 16-bit multiplication in parallel, by means of some additional gating logic to decouple the two halves. In this way, the critical delay of the operator is still reduced thanks to the separation of the two sub-operations, allowing supply voltage scaling as in DVAS. Moreover, thanks to the enhanced parallelism, the clock frequency can be halved while maintaining the original throughput, thus further reducing the power consumption. This principle is applied ''recursively'' to generate more quality modes (e.g. four 8-bit multiplications and eight 4-bit multiplications in the previous example). A full chip including DVAFS-based datapaths has been demonstrated in [13].

With respect to DVAS, DVAFS clearly permits a smaller number of quality modes. Moreover, it is also less general, as it does rely on some architectural modifications and can only be applied to circuits that can be easily modified to support subword-parallel operations (e.g. arithmetic units) but not directly to more complex accelerators.

III. MOTIVATION

Despite its theoretical effectiveness, DVAS fails to yield the expected energy reductions when applied to circuits that are synthesized using a standard EDA flow [19], [20].

In fact, all synthesis and place and route (P&R) tools for ASICs optimize the longest timing paths in a circuit

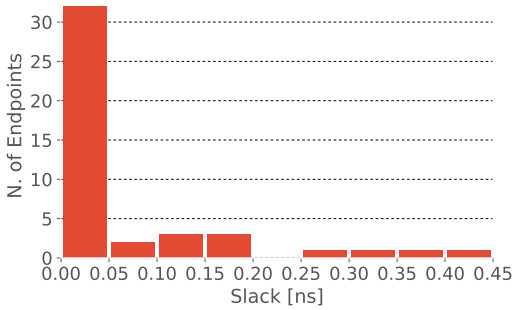

FIGURE 1. Example of wall-of-slack: endpoint slack distribution for a 16 × 16-bit MAC with 44-bit accumulator.

for performance, whereas shorter paths are exploited for area and power recovery [38], [39]. In particular, tools map gates in short timing paths to slow library cells (i.e. with small widths and high threshold voltages), since these are smaller and consume less static and dynamic power. As a result, after synthesis and P&R, the delays of the initially non-critical paths tend to increase, to the point where their slack becomes comparable to that of critical ones. From the perspective of the synthesis tools, the ideal result is obtained when all slacks are very close to zero, causing a phenomenon known as the *wall-of-slack* (WOS) [19], [20], [40].

An example is shown in Figure 1, which reports the distribution of timing slack for the different endpoints of a 16 × 16-bit Multiply and Accumulate (MAC) circuit, with 44-bit accumulator, synthesized at $f_{clk} = 1.25GHz$, $V_{DD} = 0.95V$, targeting a 28nm Fully-Depleted Silicon On Insulator (FDSOI) technology. As shown, the great majority of endpoints have a worst slack very close to 0s.

The existence of the WOS invalidates the DVAS assumption that the overall slack of the circuit should increase as an effect of gating some LSBs. Indeed, given a situation like the one of Figure 1, it is easy to see that, regardless of the number of gated input bits, the slack distribution will change only marginally. This, in turn, means that V_{DD} can barely be reduced (without incurring timing violations), even when the circuit works at very low precision.

The authors of [17] do mention this problem and propose to solve it by modifications of the EDA flow, which are however not described in their papers. A partial solution has been presented in [20], where it was shown how the slack distribution can be re-shaped when working at low precision, by means of fine-grain, adaptive back-gate biasing applied to selected parts of the circuit. However, that method is specific to FDSOI, where back-gate biasing is an extremely powerful knob for tuning the power/performance tradeoff, and would therefore not be as effective on different technologies.

In Section IV we will describe a more general solution, applicable to any technology. Our method first explores the design space to find the optimal supply voltage (V_{DD}) to use for each precision, then constrains the synthesis and place and route accordingly, thus preventing the formation of the WOS. Importantly, the initial optimization phase can also leverage application statistics to yield greater benefits, which is impossible with the method of [20].

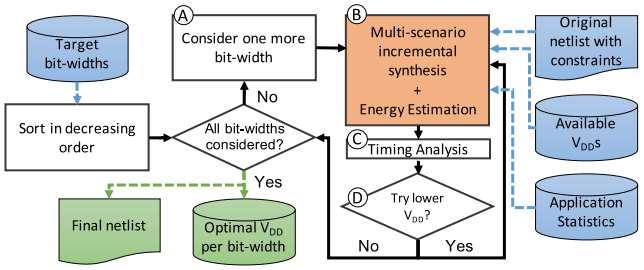

FIGURE 2. High-level view of the proposed synthesis flow.

IV. APPLICATION-DRIVEN SYNTHESIS FLOW FOR MULTIPLE PRECISION CIRCUITS

This section describes a novel tool that allows designers to integrate DVAS- and DVAFS-based energy-quality scalable circuits within a standard EDA flow, overcoming the aforementioned limitations. A high-level view of its main operations is shown in Figure 2, where inputs and outputs are colored in blue and green respectively.

The tool receives four inputs:

1) The gate-level netlist of the design with its corresponding constraints (clock frequency, boundary conditions, etc.). This is a standard implementation of the circuit, that only supports the maximum precision mode, synthesized at nominal V_{DD}.
2) The set of *target precisions* (i.e., bit-widths) that must be supported in the final energy-quality scalable version of the circuit.
3) The set of available supply voltages to be used for voltage scaling.
4) Statistics about the frequency of usage of each precision in the target application, detailed more in depth in the following. This input is optional.

Given these data, the tool iteratively re-synthesizes the circuit considering different combinations of precision (obtained through LSB-gating) and V_{DD}, using an EDA synthesis tool. Its final outputs are a list containing the *optimal* V_{DD} to use for each precision, and a modified version of the input netlist, able to support such combinations of supply voltages and bit-widths.

The most important operation of the tool is labeled step B and highlighted with an orange background in Figure 2. Sections IV-A and IV-B focus in depth on the two operations that compose this step, whereas Section IV-C describes the overall flow of the tool. These three sections consider the case of a circuit based on standard DVAS, while Section IV-D describes the differences in the flow for the case of DVAFS.

A. MULTI-SCENARIO SYNTHESIS

Our tool makes use of multi-scenario synthesis (also known as multi-corner, multi-mode, or MCMM) to simultaneously consider the operation of a circuit at different V_{DD}s

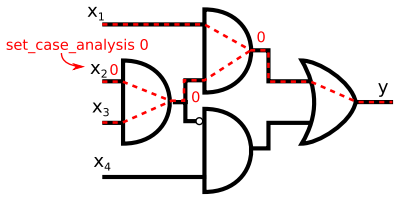

FIGURE 3. Example of case analysis constraint. Red dashed lines correspond to false paths.

and precisions. This functionality is supported by all major commercial synthesis and P&R tools [38]. When using a MCMM flow, the synthesizer is instructed to simultaneously consider multiple *operating corners* during its optimizations, each associated with its corresponding *operating mode*. A corner is simply the description of the process, supply voltage and temperature (PVT) point in which the circuit will operate, while a mode is a set of constraints (e.g. a target clock frequency). MCMM is used, among others, in variability-aware designs.

In a DVAS-based circuit, multi-scenario synthesis can be leveraged *to ensure that the circuit does not have timing violations in all target combinations of V_{DD} and precision (bit-width).*[1] This is achieved by creating a separate scenario for each bit-width. These scenarios are generated automatically starting from the nominal one corresponding to the original design, as follows:

- New corners are simply copies of the original one, in which V_{DD} is modified to refer to the correct supply voltage value.
- New modes are obtained by adding *case analysis* directives to the set of constraints of the original scenario.

Case analysis is a feature of industrial synthesis and P&R software that forces them to only consider one particular value or transition for some nets in the circuit when doing timing and power analysis [38]. When this feature is used to force a logic "0" on a net, for instance, the synthesizer will assume that such net never changes its value and consequently never incurs a transition. This means that the timing paths activated by a 0-1 or 1-0 transition on that net will be classified as *false* and ignored for timing optimization and power analysis (e.g. switching activity computation). Moreover, the EDA tool, using appropriate options, allows propagating case analysis constraints to the transitive fanout of the originating net. An example of the effects of a case analysis constraint on a gate-level netlist is shown in Figure 3, where red dashed lines correspond to disabled timing paths.

In order to inform EDA software that the circuit is working at a reduced bit-width, our tool automatically generates a set of case analysis directives imposing constant logic 0s on the input LSBs, which are appended to standard constraints in the corresponding scenario. Importantly, we always keep in the set of scenarios passed to the synthesizer the one corresponding to the *maximum precision*, i.e. the one whose constraints do not include case analysis on input LSBs. This implies that the synthesizer will ignore false paths when checking timing compliance and power consumption in reduced-precision scenarios, but will not eliminate the corresponding logic, as the same paths are not false in the maximum-precision scenario.

[1]The way to associate an appropriate V_{DD} to each precision is described in Section IV-C. This section assumes that such association has already been determined.

```
1  set target_scenarios {{16 0.95} {8 0.80} {4 0.70
       }}
2  set input_ports {{A 16} {B 16}}
3
4  set_app_var case_analysis_sequential_propagation
       always
5  set_app_var case_analysis_propagate_through_icg
       true
6
7  foreach scen ${target_scenarios} {
8      set prec [lindex $scen 0]
9      set vdd [lindex $scen 1]
10     create_scenario "${vdd}V_${prec}bit"
11 #   common constraints
12     source input_constraints.tcl
13     foreach port ${input_ports} {
14         set name [lindex $port 0]
15         set gated [expr [lindex $port 1] - $prec
       ]
16         for {set b 0} { $b < ${gated} } {incr b}
        {
17             set pin "${name}\[${b}\]"
18             set_case_analysis 0 [get_ports $pin
       ]
19         }
20     }
21     set_operating_conditions -analysis_type
       bc_wc -max "ss28_${vdd}V_125C" -min "
       ff28_1.30V_m40C"
22 #   other scenario configs
23     ...
24 }
```

FIGURE 4. TCL script for the generation of per-bit-width scenarios.

Figure 4 shows a simplified version of the TCL scripts used to generate one scenario per bit-width as just described, using Synopsys Design Compiler's jargon as an example. Similar commands can be used in tools from other vendors. The script is sourced within the synthesizer, before launching the actual synthesis (e.g. with the `compile_ultra` command). Lines 1-2 are the input variables for the procedure, which are actually set by our tool upon calling Design Compiler. The first variable describes the target scenarios, represented as a list of (precision, V_{dd}) pairs. The second variable informs the tool of the input ports whose precision should be dynamically tuned, by providing a list of (name, bit-width) pairs. This is needed because not all inputs of the circuit should be considered when gating LSBs; clock, reset and other control inputs should obviously never be gated.[2]

Lines 4-5 are tool-dependent settings that ensure that case analysis constraints are propagated through sequential cells (e.g. flip flops) and integrated clock gating (ICG) cells respectively. Next, the script iterates through the list of target

[2]In practice, the specification of input ports in our scripts is more complex, e.g. it also includes a *multiplier* for the case of ports with different LSB significance, but these technical details are neglected to avoid over-complicating the description.

scenarios (lines 7-24). The scenario is created in line 10 with a custom name; next, an external script containing common (scenario independent) constraints is sourced in line 12. This is simply the constraint file provided as input to our tool and includes the definition of boundary constraints (input/output delays, input driving cells and output loads, etc.) and of the circuit clocks (main period, uncertainty, maximum transition time, etc.). In lines 13-20, the script integrates these constraints with scenario-specific ones related to LSB gating. To this end it iterates over all ''gateable'' input ports, then computes the number of LSBs that should be gated from the target precision and port bit-width (line 15), and finally applies one case analysis constraint per each gated bit (line 18), specifying the logic value ''0''.

The rest of the script is devoted to the definition of the corner for each scenario. The most relevant operation in this section is the definition of the operating conditions libraries (line 21), which depend on the target V_{DD}. Notice that each scenario also includes a common fast corner library (''ff28_1.30V_m40C'' in this example) for hold time optimization. Following this definition, several other scenario-related operations are performed (line 23), such as reading VCD/SAIF files for switching activity annotation, setting scenario options etc. However, these are identical to the ones performed in a regular synthesis flow and are therefore not reported here.

We use the multiple scenarios constructed as shown in Figure 4 to perform an *incremental re-synthesis* on the input netlist, that is a modification of the gate-level mapping with which the EDA tool attempts to respect the newly provided constraints. Thanks to the MCMM flow, the tool will be forced to *concurrently ensure timing compliance in all V_{DD} and bit-width combinations*.

If the V_{DD} associated with one of the reduced bit-width scenarios is too low to ensure timing compliance only by exploiting the additional slack deriving from LSB gating, the EDA tool will then replace some of the logic cells belonging to the *active* paths of the circuit (i.e. those corresponding to not-gated inputs) with larger/lower threshold voltage equivalents. This will shorten the corresponding paths, thus reshaping the overall slack distribution of the circuit and contrasting the wall-of-slack. If the tool succeeds in resolving all timing violations during the re-synthesis step, the circuit will then be able to operate at the target V_{DD} for that bit-width, thus possibly consuming less power.

Clearly, the effect of incremental re-synthesis is to increase the area occupation of the circuit, as well as the power consumption in higher precision modes (especially at maximum precision), due to the insertion of larger and more consuming cells. Fortunately, in many applications, maximum precision is only seldom required [14], [18], [22], [41]; therefore, although the power consumption at maximum precision increases, this solution will be beneficial in terms of *energy*, as the least power consuming instances will be used more frequently.

B. APPLICATION-DRIVEN ENERGY ESTIMATION

After each re-synthesis step, our tool must evaluate the expected energy consumption of the resulting netlist, considering the contribution of all target scenarios. However, as anticipated, not all scenarios have the same importance for the final design, especially if the circuit is going to be included in a custom hardware accelerator. In fact, we do not aim at optimizing the power consumption of individual precision configurations, but rather the *overall energy consumption* of the circuit. In the typical case of some precision configurations being used more often than others by the target application (or set of applications), it is clear that the most frequently used precisions should be more carefully optimized.

As explained in Section IV-A the problem is not trivial, as there is typically a trade-off between reducing the consumption at one precision (by lowering the corresponding V_{DD}) and increasing the power of other configurations (due to using larger/lower threshold voltage cells). Therefore, our tool evaluates each intermediate solution during its optimization by computing a proxy of the final metric to be optimized (total energy) as follows:

$$E_{est} = \sum_{i}^{N} w_i P_i(\Psi, V_{DD,i}, b_i) \tag{1}$$

where N is the number of bit-width configurations and P_i is the power in each configuration. P_i is in general a function of the set Ψ of standard cells that compose the circuit (note that Ψ is common to all configurations), of the supply voltage applied to the circuit in that configuration ($V_{DD,i}$) and of the corresponding bit-width b_i. The latter has an influence as it determines the activation of some of the cells in Ψ. Each re-synthesis step changes both Ψ and $V_{dd,i}$, thus possibly influencing in opposite ways different precision configurations.

Weights w_i are optional, and can be set to $w_i = 1\ \forall i$ to give the same ''importance'' to the power consumption at all precisions, e.g. if the target design is a general-purpose energy-quality scalable platform. In case of application-specific hardware, however, we propose to assign to each w_i a value proportional to the frequency at which precision i is used. For example, if an application uses three precisions (e.g. 4, 8 and 16-bit) for the 50%, 30% and 20% of the time respectively, we could assign weights 0.5, 0.3 and 0.2 to Equation (1) during re-synthesis.

The statistics on the usage frequency of each precision can be gathered from software simulations. Alternatively, in some relevant domains (e.g. neural network inference), such frequencies are roughly *data independent* (e.g. for a given neural network architecture and classification task) hence the weights can be directly obtained from ''pen-and-paper'' considerations [18], [22], [41].

C. OPTIMAL V_{DD} selection

In Sections IV-A and IV-B we described how our tool optimizes an existing circuit to support multiple V_{DD} and

```
1:  procedure GREEDY SUPPLY VOLTAGE SELECTION
2:      V_start = current supply voltage, initialized to nominal V_DD
3:      s_0 = nominal scenario [V_start, b_max]
4:      S = list of considered scenarios, initialized to s_0
5:      Ψ = nominal circuit netlist, synthesized in scenario s_0
6:      for all reduced bit-widths b (in decreasing order) do
7:          s_new = [V_start, b] (same V_DD as previous scenario)
8:          E_new = "energy" of Ψ in scenarios S + s_new
9:          V_new = V_start
10:         do
11:             s_ref = s_new, E_ref = E_new, V_ref = V_new
12:             V_new = decrease V_new
13:             s_new = [V_new, b]
14:             Ψ_new = incr. synthesis of Ψ in scenarios S + s_new
15:             E_new = "energy" of Ψ_new in scenarios S + s_new
16:             Ts = worst slack of Ψ_new in scenarios S + s_new
17:         while V_new > V_min and Ts ≥ 0 and E_new < E_ref
18:         S = S + s_ref
19:         V_start = V_ref, Ψ = Ψ_new
20:     end for
21: end procedure
```

FIGURE 5. Proposed greedy algorithm for the selection of the optimal V_{DD} for each bit-width.

bit-width configurations and how it evaluates the energy consumption resulting from such optimization. This section completes the description by detailing the remaining operations shown in Figure 2 and focuses in particular on how to select the appropriate V_{DD} for each bit-width. In this work, we perform such selection using a greedy algorithm, as shown in the pseudo code of Figure 5.

In the pseudo-code, the + symbol used with scenarios (lines 8, 14, 15, 16) corresponds to the list concatenation operation. Initially (lines 2-5) the procedure considers the input circuit in its original synthesis scenario, i.e. nominal supply voltage (V_{start}) and maximum precision (b_{max}).

The core of the algorithm spans the available set of supply voltages, starting from the nominal value and progressively decreasing it. While doing so, it adds one bit-width mode at a time to the considered set, in decreasing order of precision (line 6, corresponding to step A in Figure 2).

For each new bit-width, the first step (line 8) is an assessment of the energy consumption of the circuit when the new scenario uses the *same* supply voltage as the previous one, i.e. the only difference between the two is in the number of zeroed LSBs. Energy estimation is performed with Equation (1) and accounts for all currently-considered bit-width modes with their appropriate weights. Next V_{DD} is decreased (line 12) and the circuit is re-synthesized with the new scenario, so that the EDA tool can attempt to restore timing compliance with the lower V_{DD}. Re-synthesis is performed as described in Section IV-A and accounts for all currently considered bit-widths thanks to MCMM. The energy of the resulting circuit is evaluated again using Equation (1) in line 15.

The result of re-synthesis is then also inspected with a Static Timing Analysis (STA) tool (line 16 or step C in Figure 2), to check if all timing violations have been resolved; failures may occur when the selected V_{DD} is too low. STA checks for positive setup and hold slacks in *all* considered bit-width modes. The progressive scaling of V_{DD} is continued until it results in a *reduction of total energy*, and the tool is able to avoid timing violations (line 17 or step D). The lowest V_{DD} that satisfies these two conditions is selected as final supply voltage for the considered bit-width, and the corresponding scenario is saved (line 18). Then, a new bit-width is added to the set, and the procedure is repeated starting from the last considered V_{DD}, i.e. V_{start}.

1) OPTIMALITY CONSIDERATIONS

Identifying the optimal V_{DD} for each precision requires several re-synthesis steps. In order to limit the number of re-synthesis we adopt a greedy approach, stopping the V_{DD} decrement at the first minimum of Equation 1 although in general, that point might not correspond to the global optimum.

In fact, it could happen that a particular combination of V_{DD} and bitwidth causes the value of Equation 1 to increase, while a further decrement of V_{DD} might reduce the metric. Whether this happens depends on many factors, such as: the values of weights w_i, the topology of the considered circuit, the re-synthesis algorithm used by the EDA tool (which is typically proprietary and not modifiable) and the synthesis constraints (especially f_{clk}). In practice, however, when considering the use cases detailed in Section V such situation never occurs, and the greedy and exhaustive solutions coincide, with the former requiring a smaller number of re-synthesis steps (at *maximum* equal to the number of different V_{dd}s considered in the optimization).

With the same rationale of keeping the execution time as small as possible, our optimization is performed at gate-level rather than after P&R. Although the latter solution would yield more accurate results, post layout optimizations take much longer to execute than gate-level re-synthesis. Moreover, the P&R tool also has less freedom to optimize the circuit; for instance, performing the incremental steps after placement would preclude gate resizing, and only allow routing optimizations, which are less effective in contrasting the wall-of-slack. Finally, another important reason why it is preferable to use the proposed tool at gate-level is that its output, i.e. the modified netlist and corresponding set of constraints can be freely embedded in the P&R of a larger circuit. In contrast, executing the flow at P&R level would imply that each individual reconfigurable-precision module is *separately* placed and routed, and then inserted in a larger design as a macro, causing large area and routing overheads. In summary, as long as power estimations after synthesis and P&R are correlated [38], it makes sense to *select* the appropriate V_{DD} for each precision post-synthesis, and then *enforce* these V_{DD}s during P&R. In such a way, a single (multi-scenario) P&R is performed, using the previously selected V_{DD} for each bit-width and the corresponding case analysis directives as constraints.

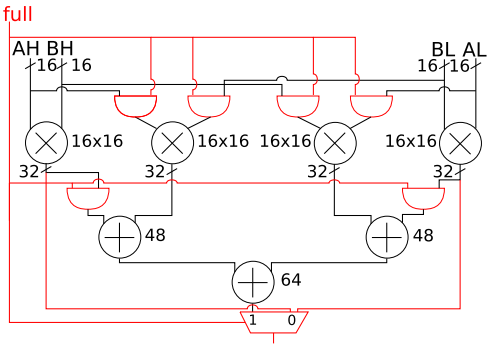

FIGURE 6. Schematic of a 32-bit multiplier able to perform two 16-bit operations in parallel.

D. SUPPORT FOR DVAFS

As detailed in Section II, DVAFS uses sub-word parallel operations to increase the throughput of a circuit operating at reduced bit-width, then restores the original throughput by appropriately dividing the clock frequency. Therefore, while DVAS does not require any architectural modification of the circuit, besides some minimal logic to allow LSB gating, DVAFS does require a customized architecture.

As an example, Figure 6 shows a basic implementation of a DVAFS-compatible multiplier, able to perform one 32×32-bit operation or two 16×16-bit operations per clock cycle. Red blocks and connectors in the figure represent the additional elements of the architecture compared to a standard multiplier. As shown, depending on the value of the additional *full* control bit, the architecture isolates the multiplication of the MSB-halves (AH and BH) and of the LSB-halves (AL and BL) of the two inputs. This is obtained through the output multiplexer, which selects either the normal 64-bit product or the concatenation of the two 32-bit half-products. Moreover, the inputs of the unused partial products ($AL \cdot BH$ and $BL \cdot AH$), as well as the inputs of the accumulation adders are zeroed-out when *full* is 0 to reduce the dynamic power. In order to support smaller precisions (e.g. four 8×8-bit operations), the same modifications can be implemented recursively to the two 16×16-bit multipliers that compute $AH \cdot BH$ and $BL \cdot AL$. Clearly, the resulting circuit will then have additional control inputs to select the operating mode.

Notice that, although there are clearly *four* 16×16 multipliers in Figure 6, the circuit only supports two 16-bit operations in parallel. The reason is twofold. First, the inputs to the four multipliers are not independent. Therefore, feeding them with eight independent inputs would require replacing the AND gate arrays with multiplexers, which are typically slower and bigger, thus increasing the overheads. Second, the number of I/Os of the circuit would increase for each precision mode, e.g. 64 inputs plus 64 outputs for one 32-bit operation versus $128 + 128$ for eight 16-bit operations, etc. The total number of connections would become very large if lower precisions (8/4-bit) are supported. Consequently, all the circuitry surrounding the multiplier (registers, control logic, etc.) would have to be sized for the maximum number of I/Os, with a dramatic impact on area occupation and power consumption (especially leakage). For these reasons, it is preferable to sacrifice the full utilization of the 16×16 multipliers and only support two parallel operations.

```
1  set target_scenarios {{32 0.95 1 {full 1} {half
       1}} {16 0.80 2 {full 0} {half 1}} {8 0.70 4
       {full 0} {half 0}}}
2
3  ...same as DVAS...
4
5  foreach scen ${target_scenarios} {
6      ...same as DVAS...
7  #   common constraints
8      source input_constraints.tcl
9
10 #   override default create_clock
11     set mult [lindex $scen 2]
12     set period [expr $mult * $default_period]
13     create_clock -period $period -name
       $clock_name
14
15     set ctrl_signals [lrange $scen 3 end]
16     foreach sig ${ctrl_signals} {
17         set name [lindex $sig 0]
18         set val [lindex $sig 1]
19         set_case_analysis $val [get_ports $name
       ]
20     }
21     ...same as DVAS...
22 }
```

FIGURE 7. TCL script for the generation of per-bit-width scenarios in the case of a DVAFS circuit.

The most significant difference for our tool when dealing with a DVAFS circuit is in the definition of case analysis constraints. In fact there are no gated LSBs in DVAFS, and what changes between two precision configurations is the value of the control inputs that enable sub-word parallelism (e.g. *full* in Figure 6). Figure 7 shows the differences in the TCL script for the automatic creation of multiple scenarios with respect to the case of standard DVAS. In the definition of the target scenarios provided to the synthesizer (line 1), after the precision and V_{DD}, each list element now contains an integer *clock period multiplier* for frequency scaling (1, 2 and 4 for the 3 scenarios in the example). Then, each element also contains a list of (name, value) pairs defining all the control signals that should be set to a fixed value when optimizing that scenario (*full* and *half* in the example).

After reading the scenario independent constraints (line 8), the script now overrides the clock period definition in each scenario, multiplying the period by the appropriate factor (lines 11-13). Finally, case analysis constraints are set by simply applying the specified values to all the control inputs defined in the `target_scenarios` (lines 15-20). In practice, our tool uses a single TCL script to generate scenarios, and discerns between DVAFS and DVAS depending on the presence/absence of clock multipliers and control inputs definitions in the input list. Besides these differences, the rest of the flow described in Section IV-C is identical to the case of DVAS.

Clearly, the wall-of-slack effect is less critical for DVAFS, as the additional slack for voltage scaling at reduced precision

mainly derives from the clock frequency reduction, rather than from disabled timing paths. However, our tool is still useful to determine the optimal V_{DD} for each operating mode of the circuit. Without this association, in fact, a designer could only synthesize the circuit without imposing multi-scenario constraints, and then identify the minimum voltage for each bit-width a posteriori through STA. While the loss in power saving opportunities would be less relevant than in the case of DVAS, it could still be sizable. In fact, especially when the usage frequency of different scenarios is non-uniform, our tool optimizes the netlist to allow lower-voltage operation for the most relevant bit-widths, finding the best balance with the power increase in other configurations due to gate remapping, all of which is not possible using a standard synthesis flow.

V. EXPERIMENTAL RESULTS

In this section, we assess the impact of our tool on two DVAS-based circuits of different sizes and complexity, which have been assumed to be part of hardware accelerators for realistic error tolerant applications, i.e. Convolutional Neural Network (CNN) inference [23] and image compression using Discrete Cosine Transform (DCT) [14]. We then show examples of the effectiveness of our tool for two other circuits, not related to a specific application, in order to show the generality of the proposed methodology: a 32-bit multiplier based on DVAFS and a 16-bit non restoring divider. Overall, these benchmarks internally include most of the components found in digital datapaths (adders, multipliers, registers, glue logic, etc.). Therefore, they serve as examples to show the effectiveness of our proposed flow on most datapaths, and on both DVAS and DVAFS. Finally, we compare the proposed synthesis modifications with the technology-specific methodology proposed in [20] to synthesize DVAS-based circuits.

A. SETUP

All experiments were performed starting from VHDL descriptions of the circuits, synthesized targeting a 28nm FDSOI technology library from STMicroelectronics. The nominal V_{DD} was set to 0.95V for all benchmarks and the clock frequency was selected so to guarantee a positive worst timing slack smaller than 100ps. Unless specified otherwise, we instructed our tool to consider supply voltages from 0.95V to 0.60V in steps of 0.05V.

For logic synthesis (both the initial run to get the nominal circuit, and the multiple re-synthesis performed by our algorithm) we used Synopsys Design Compiler L-2016.03. Place and Route was executed with Cadence Innovus 16.1, while Static Timing Analysis and power analysis were performed in Synopsys PrimeTime L-2016.06. The algorithm of Figure 5 was implemented in Python 3.5 and uses the *subprocess* package to invoke Design Compiler and PrimeTime. However, the functionality of the tool is not tied to the specific EDA tools used in this setup, and can be made compatible with other vendors simply by replacing the adapter Python methods that translate generic tool commands into TCL scripts.

TABLE 1. Number of MAC operations performed at each bit-width during the classification of one image with LeNet-5 [23] and corresponding objective function weights in the proposed synthesis tool.

Bit-Width	Number of MACs	w_i
16	$\sim 0.01 \cdot 10^6$	0.01
8	$\sim 1.6 \cdot 10^6$	1.6
4	$\sim 0.3 \cdot 10^6$	0.3

The Python code is made available in [24]. Throughout our experiments, we kept all synthesis tool options (such as the optimization effort, the use of topographical synthesis, etc.) constant, and set to the most appropriate values for the target technology. This is because our goal is to propose an optimization that is fully compatible with the pre-existing EDA flows used by a designer or company, without requiring any tweak of tool-specific options or algorithms.

B. USE CASE 1: MAC FOR CNN INFERENCE

This first use case focuses on a Multiply-and-Accumulate (MAC) circuit, i.e. the most important arithmetic component in hardware accelerators for neural networks inference [18]. We have considered a scenario in which this circuit has to be included in an accelerator for handwritten digit recognition, using a LeNet-5-like CNN architecture [23].

Previous research has shown that hardware accelerators for CNN inference can leverage reduced bit-width fixed-point MACs to gain performance and reduce energy consumption, with a small impact on classification accuracy [18], [22], [41]. The authors of [18] have also shown that a better accuracy is obtained if different bit-widths are used for the MAC operations relative to different layers of a CNN. In this experiment, we have targeted the same bit-widths proposed in [18] for the MAC operations required by the Convolutional layers of LeNet-5. We have also assumed that the Fully Connected (FC) layers of the CNN use 16-bit MACs, in accordance with [22]. Combining these assumptions with the architectural details of each layer (number of convolutional filters/FC nodes, filter sizes, etc.), we have computed the number of MACs that require a given bit-with in LeNet-5, when performing a single image classification; the results of this analysis are reported in the first two columns of Table 1. As shown, the MAC circuit must support three precision configurations, i.e. 4-bit, 8-bit and 16-bit. Notice that the impact of using these precisions for MACs on the final accuracy of the neural network has already been evaluated in [18] and is out of the scope of this work. Herein, we use their results as a starting point and focus on the design of an appropriate datapath for achieving that accuracy.

We have considered the impact of three different DVAS-based synthesis flows on the same initial VHDL description of the circuit:

- *Conventional*: a standard flow that does not include specific optimizations to facilitate DVAS, i.e. LSB gating and voltage scaling are applied a posteriori, to the outputs of a normal synthesis and P&R.

TABLE 2. Reduced bit-width V_{DD} and area overhead (normalized to the conventional version) of the three MAC versions.

MAC Version	V_{DD} [V]		Area Ovr. [%]
	8-bit	4-bit	
Conventional	0.95	0.95	-
Uniform	0.90	0.90	9
Weighted	0.75	0.70	16

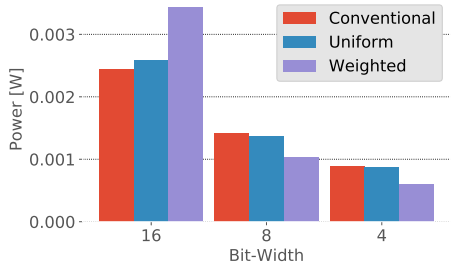

FIGURE 8. Power consumption in different bit-width modes for the three MAC versions.

- *Uniform*: an execution of the proposed flow (Figure 2) in which the power weights for different precision configurations of Equation 1 are all set to 1, i.e. the target application is not taken into account.
- *Weighted*: an execution of the proposed flow in which the power weights are set to values proportional to the usage frequency of each precision. Weight values are reported in the rightmost column of Table 1.

The MACs have been designed with a 44-bit accumulator and the target clock frequency for all three versions has been set to $f_{clk} = 1.25GHz$. For this benchmark, the complete execution of our tool on the server used for our experiments (Intel Xeon E5-2630 @ 2.40GHz, 128GB RAM, Linux OS) has taken a maximum of 30 minutes for the *Weighted* version. For comparison, a standard synthesis of the MAC takes around 4 minutes.

Table 2 shows the minimum supply voltage achievable without timing violations when the circuit is working at 8-bit and 4-bit for the three MAC versions. The V_{DD} for maximum precision (16-bit in this case) is always equal to the nominal one (0.95V) and is therefore not reported in the table. In fact, our flow never modifies the operating conditions for maximum precision while optimizing lower-precision modes.

These values have been confirmed analyzing the final P&R output in PrimeTime. For the Uniform and Weighted circuits, the V_{DD} values correspond to the optimal ones identified by our tool during synthesis. The rightmost column of Table 2 reports the area overhead of the Uniform and Weighted MACs with respect to the Conventional version. Figure 8 shows the total power consumption of the three MACs at each bit-width. Power values include both leakage and dynamic components and are estimated using the post-P&R layouts.

As shown by the data in Table 2, down-scaling V_{DD} of just $0.05V$ (from $0.95V$ to $0.90V$) causes a timing violation

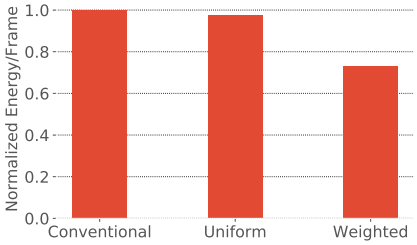

FIGURE 9. Energy due to MAC operations for classifying one image in LeNet-5.

at both 8-bit and 4-bit precision in the Conventional MAC, as an effect of the wall-of-slack (indeed, the slack distribution shown in Figure 1 corresponds exactly to this circuit). Conversely, for the Uniform MAC, our tool is able to remap the cells in the circuit in order to ensure timing compliance at $0.90V$ for both low-precision configurations. The supply voltage is not reduced further due to the high power overheads that this would cause at 16-bit. However, this intermediate solution does not take the statistics about application data into account. In particular, it neglects the fact that 16-bit precision is only required in about 0.5% of the MAC operations.

When provided with information about the usage of different configurations (Weighted MAC), our algorithm selects a much more aggressively scaled V_{DD} for both 8-bit and 4-bit precision. This increases the area overhead to 16%, but allows significant savings at reduced bit-widths (27% at 8-bit, and 31% at 4-bit).

Figure 9 reports the total energy consumed by the three MAC versions when classifying one image in LeNet-5. Results are normalized to the Conventional implementation.

While the Uniform version only consumes $\approx$ 5% less energy compared to a standard design, the Weighted solution reduces consumption by $\approx$ 27%. Importantly, these result are achieved despite the additional leakage energy caused by the area increase in the Uniform and Weighted MACs.

C. USE CASE 2: DCT

In this experiment, we have applied our tool to a DCT accelerator. As for the CNN example, we have leveraged a previous analysis on the impact of reduced-precision operations in DCTs. The authors of [14] studied the impact of using different bit-widths for the computation of different frequency components within a DCT. They used a design space exploration framework to identify the optimal combinations of bit-widths for different quality (PSNR) levels. In order to implement these combinations, a HW accelerator should support 12-bit, 9-bit, 6-bit and 4-bit operations. Table 3 reports the percentage of coefficients computed at each bit-width for the two the solutions of [14] that use all the four bidwidths (12, 9, 6, and 4 bits). Since the computation of each coefficient involves the same operations, these values also correspond to the usage frequencies of the different precisions.

We have synthesized an open source DCT accelerator available on *OpenCores* [42], targeting a clock frequency

TABLE 3. Number of DCT coefficients computed using each bit-width in the first two solutions of [14].

Bit-Width	DCT Coefficients [%]	
	Sol 1	Sol 2
12	16.66	12.50
9	41.66	25.00
6	33.33	37.50
4	8.33	25.00

TABLE 4. Reduced bit-width V_{DD} and area overhead (normalized to the conventional version) of the four DCT versions.

DCT Version	V_{DD} [V]			Area Ovr. [%]
	9-bit	6-bit	4-bit	
Conventional	0.90	0.90	0.90	-
Uniform	0.75	0.75	0.75	6
Weighted 1	0.70	0.70	0.70	16
Weighted 2	0.75	0.70	0.70	14

TABLE 5. Power and normalized energy results for the DCT use case.

Version	Power [mW]				Energy [%]	
	12-bit	9-bit	6-bit	4-bit	Sol 1	Sol 2
Conventional	11.21	9.37	8.95	8.66	100	100
Uniform	12.99	7.66	7.22	6.90	88.0	86.0
Weighted 1	15.54	6.76	6.35	6.02	**84.7**	81.2
Weighted 2	15.00	7.80	6.07	5.48	86.9	**80.5**

of $f_{clk} = 2GHz$. Although this accelerator is still internally composed of multipliers and adders (as for most arithmetic circuits) the difference with the previous MAC benchmark is the *size* of the input netlist provided to our tool (4x larger number of cells). Therefore this second use case demonstrates the effectiveness of our approach on larger designs.

We have synthesized the DCT in the same three versions described in Section V-B; however, we have implemented two versions of the "Weighted" design corresponding to the two solutions *Sol1* and *Sol2* reported in Table 3. For this use case, the execution of our tool on the same server used for the MAC has taken at most 38 minutes (for the *Sol1* run).

Tables 4 and 5 show the same results described in the previous section for this second use case. As for Table 2, the V_{DD} used at maximum precision is implicitly 0.95V for all versions of the circuit and therefore it is not reported.

The results are similar to those obtained for the MAC. Thanks to our tool, low precision V_{DD}s are significantly reduced, at the cost of an increase in area and power consumption at 12-bit. This, in turn, guarantees significant energy reductions when considering a complete DCT operation. Notice that the Weighted 1 solution reduces the V_{DD} at 9-bit of an additional $0.05V$ compared to Weighted 2, due to the higher usage frequency of this bit-width in Sol 1 (41.66% vs. 25% – Table 3). In general, however, the difference in usage between different bit-width modes is less marked in this case than it was for the MAC, and the Uniform DCT achieves similar energy reductions compared to the Weighted ones. Nonetheless, the DCT versions synthesized with the most appropriate power weights (i.e. Weighted 1 for Sol 1 and Weighted 2 for Sol 2) always achieve the smallest total energy, highlighted in boldface in Table 5.

TABLE 6. Reduced bit-width V_{DD} and area overhead (normalized to the conventional version) of the three divider versions.

Div. Version	V_{DD} [V]		Area Ovr. [%]
	8-bit	4-bit	
Conventional	0.90	0.85	-
Uniform	0.85	0.80	4
Weighted	0.75	0.75	10

TABLE 7. Power and normalized energy results for the three divider versions.

Version	Power [mW]			Norm. Energy [%]
	16-bit	8-bit	4-bit	
Conventional	0.93	0.56	0.47	100
Uniform	0.96	0.48	0.42	90.9
Weighted	0.97	0.46	0.40	67.2

D. APPLICATION TO A DIVIDER

In order to further show the generality of our method, we have applied it to a circuit that has a different internal structure from both the MAC for CNN inference and the DCT accelerator. Specifically, we have selected the 16-bit non restoring combinational divider provided as an instantiable *DesignWare* component in Synopsys Design Compiler [38]. This is a smaller circuit compared to the previous two, and does not include multipliers. For this design, we have not focused on a specific application. Rather, we have assumed a generic scenario in which the divider has to support 16-bit, 8-bit and 4-bit precision, and we have assumed that 16-bit is only used 10% of the time, whereas 8-bit and 4-bit are used 45% of the time each. These usage frequencies have been chosen at random, with the only constraint that clearly, low precision operating modes must be used frequently in order to show the advantages deriving from our method. Otherwise, our tool would produce a circuit that is very similar to a standard one, as it would not spend effort optimizing low precision modes if the latter are almost never used. For this circuit, the clock frequency has been set to $f_{clk} = 500MHz$. The execution of our tool has taken 28 minutes.

The results of this experiment are reported in Tables 6 and 7. Once again, the V_{DD} at maximum precision is implicitly 0.95V for all three versions of the divider. Energy values in this case are simply computed by assuming 100 divisions, 10 of which use 16-bit precision whereas the remaining 90 are equally split between 8-bit and 4-bit.

The interpretation of these results is identical to the previous two use cases, and their purpose is mainly to demonstrate the generality of our method. Notice in particular that, because both low-precision operating modes are very important for energy (given the weights selected for this example), the total savings are even larger than in the previous two experiments.

TABLE 8. Area, power and normalized energy results for the DVAFS-based multiplier.

Version	Area Ovr. [%]	Power [mW]			Norm. Energy [%]
		16-bit	8-bit	4-bit	
Conventional	-	1.81	0.34	0.11	100
Weighted	5	1.95	0.29	0.16	90.5

E. APPLICATION TO DVAFS-BASED CIRCUITS

To evaluate the effectiveness of our flow on DVAFS-based circuits, we have considered a simple multiplier design similar to the one depicted in Figure 6. We have synthesized a 32-bit version of this circuit able to support two 16-bit or four 8-bit operations in parallel, with $f_{clk} = 1.25GHz$. In this case, we have only used the Conventional and Weighted flows, since the Uniform case would basically coincide with the former, as explained in Section IV-D. As for the previous experiments on the divider, when setting usage frequencies, we have not focused on a specific application, but generically assumed a scenario in which 8-bit precision is used 90% of the time, whereas 16-bit and 4-bit are used 5% of the time each. This is not an unlikely scenario, as it is not so different from the results of Table 1. Our tool has completed the re-synthesis of this benchmark in approximately 45 minutes.

The post-P&R layout obtained with the traditional flow can be operated at $V_{DD} = 0.70V$ when using 8-bit precision and at $V_{DD} = 0.60V$ (i.e. the minimum available) at 4-bit. This is mostly due to using a half (for 8-bit) or a quarter (4-bit) of the original f_{clk}, which allows voltage scaling even in presence of the wall-of-slack. However, when using our tool with power weights proportional to the aforementioned frequencies, the V_{DD} at 8-bit is also reduced to $0.60V$ thanks to gates remapping. The area, power and energy results of the two multipliers are shown in Table 8. Energy values have been computed as for the previous divider example.

Similarly to the previous experiments, the power increase at 16-bit produced by our flow comes from the resizing of gates, performed in order to enable a more aggressive voltage scaling at 8-bit. Moreover, in this case, the modified circuit consumes more power than a standard one also for 4-bit operations. The reason is that the standard circuit was already able to use a supply voltage of 0.60V for that precision, thanks to the frequency scaling present in DVAFS. Since 0.60V is the minimum V_{DD} available in our library, our tool cannot reduce the voltage further. Therefore, due to the larger gates inserted in the modified circuit when optimizing the 8-bit mode, the power at 4-bit also increases compared to the original design (since both use the same V_{DD}).

However, both these overheads are acceptable given that 16-bit and 4-bit mode are used rarely in this example, and consequently contribute minimally to the total energy consumption. If that was not the case, our tool would have selected a different combination of V_{DD} and gate resizing, e.g. one yielding a smaller power reduction at 8-bit, but also a smaller overhead at 4/16-bit.

TABLE 9. Results with different clock frequency constraints for the MAC circuit described in Section V-B. For each frequency the results refer to three different MAC versions (*Ver.* column): Conventional (C), Uniform (U) and Weighted (W), where the latter uses the same weights of Section V-B.

f_{clk} [GHz]	Ver.	Prec. [bit]	V_{DD} [V]	Power [mW]	Area Ovr. [%]	Norm. Energy [%]
1.4	C	16	0.95	3.10	-	100
		8	0.95	2.07		
		4	0.95	1.48		
	U	16	0.95	3.10	0	100
		8	0.95	2.07		
		4	0.95	1.48		
	W	16	0.95	3.72	9.3	84.5
		8	0.85	1.76		
		4	0.80	1.17		
0.8	C	16	0.85	1.12	-	100
		8	0.85	0.72		
		4	0.80	0.52		
	U	16	0.85	1.57	15.0	89.7
		8	0.70	0.66		
		4	0.70	0.44		
	W	16	0.85	1.59	18.7	75.2
		8	0.65	0.54		
		4	0.65	0.39		

These results show that, even if the wall-of-slack does not prevent the application of DVAFS, our tool is still useful to avoid missing significant energy savings opportunities. In fact, rather than finding the minimum V_{DD} for 8-bit operations a posteriori, our tool enforces a low supply voltage based on power weights, and this permits a significant total energy reduction (almost 10%) if the circuit is actually used with the estimated distribution of bit-widths.

F. DEPENDENCY ON CLOCK FREQUENCY

In all previous experiments, the target clock frequency for synthesis and P&R had been set to a value that ensures a positive slack < 100 ps in nominal conditions (i.e. maximum precision, nominal supply voltage and slow process/temperature corner). However, the frequency is often determined based on system-level latency/throughput constraints and can assume different values for the same design and technology target. It is therefore interesting to discuss its impact on the effectiveness of our methodology. To this end, we have repeated the experiments of Section V-B on the MAC, considering two additional clock settings, i.e. 1.4GHz and 0.8GHz. The results of these experiments are reported in Table 9. We do not report the same results for the other benchmarks, but the trends are very similar for all designs.

For each clock frequency we have synthesized the usual three versions of the MAC, i.e. Conventional (C), Uniform (U) and Weighted (W), using the same weights of Section V-B. The rows of the table report the optimal V_{DD} determined by our flow for each target precision and the corresponding power consumption. The area overhead and normalized energy consumption for the circuit (with respect to the Conventional multiplier) are also reported.

When f_{clk} is increased to 1.4GHz and our flow is ran with Uniform weights (i.e. same importance to all precisions), it simply returns the original design (working at nominal V_{DD} for all precisions). In other words, the flow determines that resizing the gates in order to scale down V_{DD} is not beneficial for the total energy. This is because the available slack is so small (e.g. 3ps at 0.95V and 8-bit) that a significant resizing of gates is needed in order to reduce V_{DD} even just by 0.05V. This results in a very large power overhead at 16-bit, which causes our algorithm to discard the modified circuit. In contrast, when our flow is ran with application-aware power weights (W), the same overhead is accepted, thanks to the additional information available that 16-bits are seldom used. Overall, the Weighted MAC reduces the energy consumption for LeNet inference of $\approx 15\%$ compared to the Conventional version, at the cost of an area overhead of 9%.

When f_{clk} is reduced to 800MHz, the results are more similar to the ones of Section V-B. In fact, at such a low frequency, the Conventional MAC (i.e. the result of a standard synthesis flow) can be made timing compliant at 0.85V.[3] Therefore, our flow behaves exactly as described in Section V-B, just starting from a "smaller nominal voltage" and yields very similar results. In this specific case the area overhead and the total energy saving of the Uniform MAC are slightly larger than at 1.25GHz, but this is imputable to the different variety of small/large gate sizes available in the target library for performing resizing at low/high frequencies respectively.

In summary, this experiment shows that our method is equally effective over a large wide of clock frequencies (800MHz to 1.25GHz in this example) and only becomes less effective when the clock constraints are extremely tight (1.4GHz), which is not common in low-power systems. In those extreme conditions, however, the area overhead of our method also reduces proportionally to the energy savings. In the worst case, our flow simply returns the original circuit when no gate-resizing and V_{DD} scaling combination is more beneficial for total energy.

G. COMPARISON WITH ADAPTIVE BACK-BIASING ON FDSOI

Finally, we have compared our flow with the only other published method, to the best of our knowledge, for the synthesis of DVAS-based operators, presented in [20]. For this experiment we have targeted once again a multiplier, which was one of the circuits considered for the evaluation of the previous method. We have run our tool considering only the same V_{DD} values used in [20], i.e. from $1.00V$ to $0.60V$ in steps of $0.10V$, and targeted the same clock frequency, i.e. $f_{clk} = 1.25GHz$. The standard cell library is also identical.

The main difference between the flow presented in this paper and the one of [20] is that the latter does not change the synthesis process, but rather uses a technological knob (back biasing) to adapt the slack distribution on the circuit. Consequently, that method has no way to differently weigh different bit-widths. Considering this aspect, for a fair comparison we have run our flow using uniform power weights.

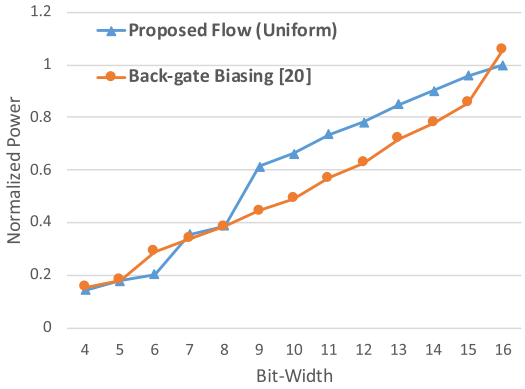

FIGURE 10. Comparison between the proposed flow and the solution of [20] for a booth multiplier.

The other consequence of this difference is that, while [20] is equally effective for *all possible* bit-widths (in steps of 1-bit), our tool only targets a set of specific precisions.[4] Starting from the original 16-bit multiplier, we instructed our tool to target 8-bit and 4-bit in this experiment.

Figure 10 shows the result of this comparison on a power versus bit-width plane. Power values are normalized to the consumption obtained with the proposed flow at maximum precision. As shown, the method based on back-biasing achieves a generally lower power consumption on those bit-widths that were not targeted by our optimization. For the three precisions considered by our tool, however, the two methods obtain almost identical results (the synthesis-based flow is slightly better at 4-bit and 16-bit). Moreover, our method also produces a smaller overhead on this benchmark, i.e. 10% vs the 14.5% of [20]. This is an important achievement as the proposed tool does not rely on technology-specific knobs. Moreover, non-uniform power weights can be used to achieve even lower consumption at selected bit-widths, as shown in previous experiments. Finally, 1-bit precision granularity is seldom used in real applications, as it significantly complicates data transfers to/from memory. So, the fact that our tool cannot optimize *all* bit-widths might be irrelevant for most use cases. In summary, by not relying on technology-specific knobs, the proposed flow has a much wider scope of applicability compared to [20] while yielding comparable results.

VI. CONCLUSION

We have described a synthesis tool for realizing energy-quality scalable circuits based on voltage and precision

[3]Not doing so would yield an unfairly favorable comparison for our flow as the Conventional MAC would consume too much at maximum precision.

[4]It is theoretically possible to include one scenario per bit-width (e.g. 16 scenarios in this example), but this would complicate the re-synthesis process and increase the runtime of our tool.

scaling. This tool solves the main practical limitations arising from the straight-forward application of DVAS within a modern EDA flow, thus permitting significantly larger energy savings at the cost of a limited area overhead. Furthermore, it can accept as input a set of weighting factors, which allow users to balance the optimization effort spent on different precisions. This makes our tool particularly suitable for the synthesis of domain-specific hardware accelerators.

The proposed method yields significant energy reductions (10-25%) on several benchmarks. Moreover, it achieves comparable power savings to those of a previous solution based on FDSOI back-biasing, despite not being limited to a particular technological node.

REFERENCES

[1] M. Alioto, "Energy-quality scalable adaptive VLSI circuits and systems beyond approximate computing," in *Proc. DATE*, Mar. 2017, pp. 127–132.

[2] J. Cong, V. Sarkar, G. Reinman, and A. Bui, "Customizable domain-specific computing," *IEEE Design Test Comput.*, vol. 28, no. 2, pp. 6–15, Mar./Apr. 2011.

[3] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *Proc. DAC*, 2013, Art. no. 113.

[4] D. J. Pagliari, M. Poncino, and E. Macii, "Energy-efficient digital processing via approximate computing," in *Smart Systems Integration and Simulation*. Cham, Switzerland: Springer, 2016, ch. 4, pp. 55–89.

[5] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006.

[6] Q. Xu, M. Todd, and S. K. Nam, "Approximate computing: A survey," *IEEE Design Test*, vol. 33, no. 1, pp. 8–22, Feb. 2016.

[7] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proc. IEEE CVPR*, Jun. 2017, pp. 6071–6079.

[8] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *Proc. ISCA*, Jun. 2016, pp. 267–278.

[9] H. Tann, S. Hashemi, R. I. Bahar, and S. Reda, "Runtime configurable deep neural networks for energy-accuracy trade-off," in *Proc. IEEE/ACM CODES*, Oct. 2016, Art. no. 34.

[10] D. J. Pagliari, E. Macii, and M. Poncino, "Dynamic bit-width reconfiguration for energy-efficient deep learning hardware," in *Proc. IEEE/ACM ISLPED*, 2018, pp. 47:1–47:6.

[11] S. Venkataramani, A. Raghunathan, J. Liu, and M. Shoaib, "Scalable-effort classifiers for energy-efficient machine learning," in *Proc. DAC*, 2015, Art. no. 67.

[12] B. Moons, B. De Brabandere, L. Van Gool, and M. Verhelst, "Energy-efficient ConvNets through approximate computing," in *Proc. IEEE WACV*, Mar. 2016, pp. 1–8.

[13] B. Moons, "Envision: A 0.26-to-10TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable Convolutional Neural Network processor in 28 nm FDSOI," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2017, pp. 246–247.

[14] J. Park, J. Choi, and K. Roy, "Dynamic bit-width adaptation in DCT: An approach to trade off image quality and computation energy," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 5, pp. 787–793, May 2010.

[15] M. Abdelrasoul, M. S. Sayed, and V. Goulart, "Scalable integer DCT architecture for HEVC encoder," in *Proc. ISVLSI*, Jul. 2016, pp. 314–318.

[16] E. Nogues, D. Menard, and M. Pelcat, "Algorithmic-level approximate computing applied to energy efficient HEVC decoding," *IEEE Trans. Emerg. Topics Comput.*, vol. 7, no. 1, pp. 5–17, Jan./Mar. 2016.

[17] B. Moons and M. Verhelst, "DVAS: Dynamic voltage accuracy scaling for increased energy-efficiency in approximate computing," in *Proc. IEEE/ACM ISLPED*, Jul. 2015, pp. 237–242.

[18] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "DVAFS: Trading computational accuracy for energy through dynamic-voltage-accuracy-frequency-scaling," in *Proc. DATE*, Mar. 2017, pp. 488–493.

[19] D. J. Pagliari, Y. Durand, D. Coriat, A. Molnos, E. Beigne, E. Macii, and M. Poncino, "A methodology for the design of dynamic accuracy operators by runtime back bias," in *Proc. DATE*, Mar. 2017, pp. 1165–1170.

[20] D. J. Pagliari, Y. Durand, D. Coriat, E. Beigne, E. Macii, and M. Poncino, "Fine-grain back biasing for the design of energy-quality scalable operators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 6, pp. 1042–1055, Jun. 2019.

[21] D. J. Pagliari and M. Poncino, "Application-driven synthesis of energy-efficient reconfigurable-precision operators," in *Proc. IEEE ISCAS*, May 2018, pp. 1–5.

[22] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," 2015, *arXiv:1502.02551*. [Online]. Available: https://arxiv.org/abs/1502.02551

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[24] *Reconfigurable Precision Synthesis*. Accessed: Oct. 15, 2019. [Online]. Available: https://github.com/danielepagliari/reconfigurable-precision-synthesis

[25] A. K. Verma, P. Brisk, and P. Ienne, "Variable latency speculative addition: A new paradigm for arithmetic circuit design," in *Proc. DATE*, Mar. 2008, pp. 1250–1255.

[26] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *Proc. DAC*, 2012, pp. 820–825.

[27] C. Liu, J. Han, and F. Lombardi, "A low-power, high-performance approximate multiplier with configurable partial error recovery," in *Proc. DATE*, Mar. 2014, pp. 95:1–95:4.

[28] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy, "Design of voltage-scalable meta-functions for approximate computing," in *Proc. DATE*, Mar. 2011, pp. 1–6.

[29] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," in *Proc. DAC*, Jun. 2015, pp. 1–6.

[30] W. Xu, S. S. Sapatnekar, and J. Hu, "A simple yet efficient accuracy-configurable adder design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 6, pp. 1112–1125, Jun. 2018.

[31] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in *Proc. IEEE/ACM ICCAD*, Nov. 2013, pp. 48–54.

[32] M. de la Guia Solaz, W. Han, and R. Conway, "A flexible low power DSP with a programmable truncated multiplier," *IEEE Trans. Circuits Syst.*, vol. 59, no. 11, pp. 2555–2568, Nov. 2012.

[33] T. Yang, T. Ukezono, and T. Sato, "A low-power high-speed accuracy-controllable approximate multiplier design," in *Proc. ASP-DAC*, Jan. 2018, pp. 605–610.

[34] V. Mrazek, Z. Vasicek, and L. Sekanina, "Design of quality-configurable approximate multipliers suitable for dynamic environment," in *Proc. AHS*, Aug. 2018, pp. 264–271.

[35] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62-1–62-33, Mar. 2016.

[36] B. Barrois, O. Sentieys, and D. Menard, "The hidden cost of functional approximation against careful data sizing: A case study," in *Proc. DATE*, Mar. 2017, pp. 181–186.

[37] R. Ramya and S. Moorthi, "Design and implementation of accuracy configurable multi-precision multiplier architecture for signal processing applications," in *Proc. IEEE RAICS*, Dec. 2018, pp. 89–93.

[38] *Synopsys SolvNet*. Accessed: Jan. 2019. [Online]. Available: https://solvnet.synopsys.com

[39] *Genus Command Reference, Product Version 16.1*, Cadence Des. Syst., San Jose, CA, USA, Oct. 2016.

[40] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Slack redistribution for graceful degradation under voltage overscaling," in *Proc. ASP-DAC*, Jan. 2010, pp. 825–831.

[41] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," 2014, *arXiv:1412.7024*. [Online]. Available: https://arxiv.org/abs/1412.7024

[42] *Opencores*. Accessed: Jan. 2019. [Online]. Available: https://opencores.org

DANIELE JAHIER PAGLIARI (M'15) received the M.Sc. and Ph.D. degrees in computer engineering from the Politecnico di Torino, Italy, in 2014 and 2018, respectively. He is currently an Assistant Professor with the Politecnico di Torino. His main research interest focuses on computer-aided design of digital systems, with particular emphasis on low-power optimization and approximate computing.

ENRICO MACII (F'05) received the Laurea degree in electrical engineering from the Politecnico di Torino, Turin, Italy, the Laurea degree in computer science from the UniversitÀ di Torino, Turin, and the Ph.D. degree in computer engineering from the Politecnico di Torino. He is currently a Full Professor of computer engineering with the Politecnico di Torino. His research interest is in the design of electronic digital circuits and systems, with a particular emphasis on low-power consumption aspects.

MASSIMO PONCINO (SM'12–F'18) received the Ph.D. degree in computer engineering and the Dr.Eng. degree in electrical engineering from the Politecnico di Torino, Turin, Italy. He is currently a Full Professor of computer engineering with the Politecnico di Torino. His current research interest includes several aspects of design automation of digital systems, with emphasis on the modeling and optimization of energy-efficient systems.