Chapter 6
Automatic Synthesis Techniques for Approximate Circuits

Ashish Ranjan, Swagath Venkataramani, Shubham Jain, Younghoon Kim, Shankar Ganesh Ramasubramanian, Arnab Raha, Kaushik Roy, and Anand Raghunathan

6.1 Introduction

Approximate circuits are the basic building blocks of approximate computing systems [28, 30, 31]. Approximate circuits realize a slightly different functionality from the required function. They are oftentimes described in conjunction with a *quality metric* that defines the degree to which the approximate circuit's output can differ from an accurate circuit implementation. Leveraging the relaxed notion of correctness, approximate circuits achieve disproportionate reduction in hardware complexity (e.g., switched capacitance, leakage, and critical path) compared to their accurate counterparts.

The first wave of efforts in the area of approximate circuits focused on the design of approximate versions of simple arithmetic units such as adders [5, 6, 12, 18, 20, 23, 24] and multipliers [7, 9, 11, 15, 21]. While these efforts demonstrated the

A. Ranjan (✉) · S. Jain · Y. Kim · K. Roy · A. Raghunathan
Purdue University, West Lafayette, IN, USA
e-mail: aranjan@purdue.edu; jain130@purdue.edu; kim1606@purdue.edu; kaushik@purdue.edu; raghunathan@purdue.edu

S. Venkataramani
IBM T.J. Watson Research Center, Yorktown Heights, NY, USA
e-mail: swagath.venkataramani@ibm.com

S. G. Ramasubramanian
Intel Corporation, Hillsboro, OR, USA
e-mail: shankar.ganesh.ramasubramanian@intel.com

A. Raha
Intel Corporation, Santa Clara, CA, USA
e-mail: arnab.raha@intel.com

S. Reda, M. Shafique (eds.), *Approximate Circuits*,
https://doi.org/10.1007/978-3-319-99322-5_6

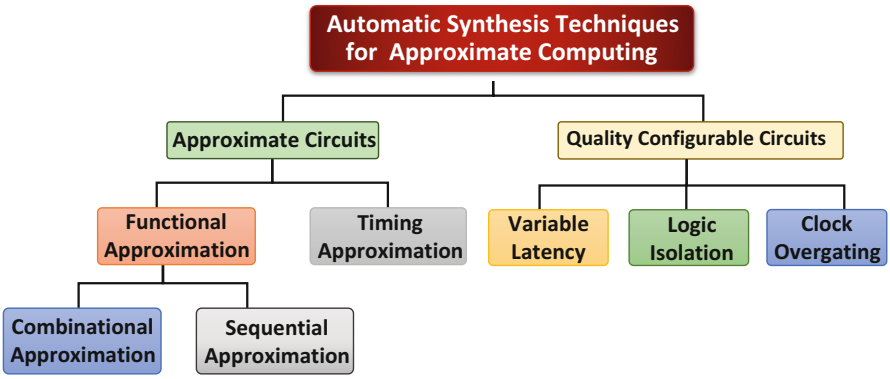

Fig. 6.1 Automatic synthesis techniques for approximate circuits: Overview

potential benefits in using approximate circuits, their broader adoption requires systematic methodologies that can create approximate implementations for any arbitrary circuit. Ideally, the tools should:

- allow designers to directly specify the circuit and the quality constraint, alleviating them from the burden of how to perform approximations;
- generate "correct-by-construction" approximate circuits that are guaranteed to conform to the imposed quality constraints;
- effectively translate the flexibility engendered by the quality constraints into energy or performance benefits.

This chapter provides an overview of the key ideas behind systematic methodologies geared towards designing approximate circuits, as summarized in Fig. 6.1. Such methodologies are applicable to any arbitrary combinational or sequential circuit, containing a mix of datapath and control logic. We categorize them into two classes:

1. **Functional approximation** techniques, which modify the logic function implemented by the circuit to improve efficiency subject to quality constraints
2. **Timing approximation** techniques that redesign the circuit to minimize the number of timing errors when its supply voltage is scaled.

An important extension of approximate circuits are *quality configurable circuits*, or circuits with the ability to reconfigure their accuracy and efficiency at runtime. They are key to approximate computing systems since in many applications, the degree of resilience often varies depending on the application context or the dataset being processed. In this chapter, we will also describe systematic methods to design quality configurable circuits.

6.2 Quality Metrics

Before describing the different approximate circuit design frameworks, we present an overview of the quality metrics that are often utilized to ascertain the degree of approximation. The quality metric bounds the type and amount of error that can be introduced in the circuit during approximation. Typically, it is a function of the original (O_{orig}) and the approximate (O_{approx}) circuit outputs. Quality metrics can be broadly classified in three categories: (i) metrics constraining error magnitude, (ii) metrics bounding error frequency, and (iii) composite metrics constraining both error magnitude and frequency. The following sections describe them in detail.

6.2.1 Metrics Constraining the Error Magnitude

The first class of metrics provide a bound on the error magnitude at the output of the circuit. The error magnitude bound can be either absolute, i.e., true for every input to the circuit or statistical over all possible circuit inputs. Some of the common quality metrics belonging to both the categories, i.e., absolute or statistical, are described below.

6.2.1.1 Maximum Error Magnitude

The maximum error magnitude ($MaxErr$), given in Eq. (6.1), constrains the absolute difference in magnitude between the outputs of the original and approximate circuits to be less than a specific threshold for each input.

$$MaxErr = MAX_{\forall inputs} |O_{orig} - O_{approx}| \tag{6.1}$$

6.2.1.2 Relative Error Magnitude

The relative error metric ($RelErr$), shown in Eq. (6.2), bounds the absolute value of the difference between 1 and the ratio of the approximate output to the original output by at most a certain margin for every input.

$$RelErr = MAX_{\forall inputs} \left| 1 - \frac{O_{approx}}{O_{orig}} \right| \tag{6.2}$$

6.2.1.3 Average Error Magnitude

The average error metric ($AveErr$), given by Eq. (6.3), bounds the absolute difference in magnitude between the approximate output and the original output, averaged over all possible circuit inputs.

$$AveErr = \frac{\sum_{\forall inputs} |O_{orig} - O_{approx}|}{Total\ number\ of\ inputs} \tag{6.3}$$

6.2.1.4 Mean Squared Error Magnitude

The mean squared error metric ($MSErr$), shown in Eq. (6.4), bounds the mean of the squared difference between the original and the approximate output across all possible inputs to be less than a given threshold.

$$MSErr = \frac{\sum_{\forall inputs} (O_{orig} - O_{approx})^2}{Total\ number\ of\ inputs} \tag{6.4}$$

6.2.1.5 Unidirectional Error Metrics

Besides constraining the absolute magnitude of error, unidirectional error metrics also restrict the direction of errors, i.e., positive or negative direction. Note that the unidirectional variants can be formulated for each of the error magnitude based quality metrics described above.

6.2.2 Metrics Bounding the Error Frequency

The second class of quality metrics constrain the frequency of error, i.e., the number of inputs for which the circuit can produce incorrect values. These metrics are especially useful when the circuit outputs do not have a numerical value. Few examples metrics are described below.

6.2.2.1 Error Probability

The error probability metric ($ErrProb$), given in Eq. (6.5), is defined as the fraction of input vectors for which the approximate circuit output differs from the original circuit output.

$$ErrProb = \frac{Total\ number\ of\ Inputs\ with\ O_{orig} \neq O_{approx}}{Total\ number\ of\ Inputs} \tag{6.5}$$

6.2.2.2 Bit Error Probability

The bit error probability metric ($BitErrProb$), shown in Eq. 6.6, is a slight variant of the error probability metric. In this case, the error probabilities of individual output bits are bounded separately.

$$BitErrProb^i = \frac{Total\ number\ of\ Inputs\ with\ O^i{}_{orig} \neq O^i{}_{approx}}{Total\ number\ of\ Inputs} \tag{6.6}$$

6.2.3 *Composite Metrics*

Composite quality metrics are a combination of both the quality metric categories described above, wherein the error magnitude as well as the error frequency is bounded.

These metrics are typically represented as an error probability distribution, an example of which is shown in Fig. 6.2. In this example, the X-axis indicates the error magnitude and the Y-axis provides the probability with which error of a given magnitude can occur in the approximate version of circuit.

6.3 Approximate Circuits

The problem of approximate circuit synthesis, as illustrated in Fig. 6.3, can be stated as follows. Given a golden specification of a circuit, and a quality constraint that bounds the degree to which the circuit can be approximated, modify the functionality realized by the circuit such that it leads to a more efficient implementation, while meeting the specified quality constraints. Broadly, approximations in circuits are introduced in two different forms: (i) *Functional approximation*, where the circuit

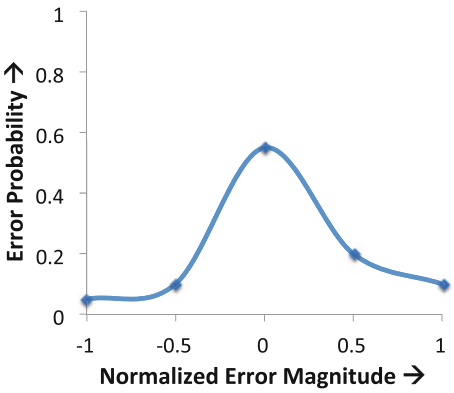

Fig. 6.2 Error probability distribution

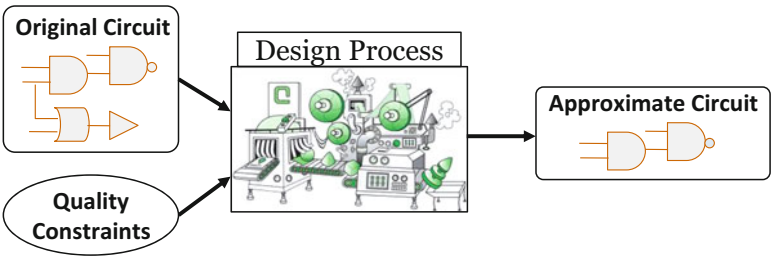

Fig. 6.3 Approximate circuit design

realizes a slightly different logic function than specified, leading to a more efficient hardware implementation [5, 6, 9, 12, 18, 23–26], and (ii) *Timing approximation*, where the circuit is designed to operate under overscaled conditions (e.g., voltage or timing), resulting in timing-induced errors [8, 14, 19]. In this section, we describe systematic approaches that enable both forms of approximations, highlighting their key principles, merits, and limitations.

6.3.1 Functional Approximation

Functional approximation involves making judicious changes to the logic implemented by the circuit thereby reducing hardware complexity without violating the quality constraints. Some of the approximations that have been explored at the logic-level include removing gates by setting some internal wires to logic 1 or 0 and propagating the redundancies [25], removing paths based on lower activation probability [16], reducing critical path length by eliminating connections between circuit nodes [18, 24, 32], identifying nodes with similar functionality and using one for the other [29], among others. However, the key challenge lies in developing a systematic method to apply these approximations to any given circuit that achieves the best improvement in performance/energy for a given quality requirement. This requires characterizing how approximating each node of the circuit affects the overall circuit's quality, and understanding how approximations at different points in the circuit interact with each other. This problem is even more challenging in the context of sequential circuits, where errors accumulate over multiple cycles of execution. We now describe two frameworks, SALSA [27] and ASLAN [22], which systematically address the above challenges and enable automatic synthesis of approximate combinational and sequential circuits, respectively.

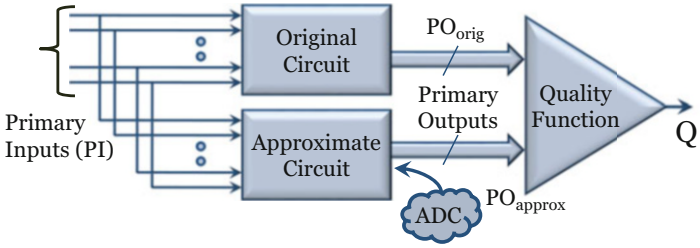

Fig. 6.4 Quality constraint circuit

6.3.1.1 Approximate Combinational Circuits

First, in the context of combinational circuits, SALSA formulates the problem of approximate synthesis into an equivalent traditional logic synthesis problem, thereby utilizing the capabilities of existing boolean optimization techniques for approximate logic synthesis. It identifies implicit don't care conditions that arise out of the relaxed quality specifications and uses them to simplify the circuit. We now describe the methodology in more detail.

Quality Constraint Circuit SALSA enforces the quality constraints during synthesis by constructing a virtual quality constraint circuit (QCC) shown in Fig. 6.4. The QCC is composed of three major blocks *viz.* the *original circuit*, the *approximate circuit*, and the *quality function* (Q-function). The inputs to the QCC are the primary inputs of the circuit considered for approximation. The Q-function is nothing but the quality constraints encoded as a Boolean function. It takes outputs from both the original (O_{orig}) and approximate (O_{approx}) circuits and produces a single bit output Q that indicates if the desired quality is met. The QCC evaluates to a tautology when the synthesized approximate circuit satisfies the quality specifications.

Approximation Don't Cares Given the QCC, SALSA utilizes the concept of observability don't cares (ODCs) to identify opportunities to approximate the original circuit. In multi-level logic synthesis, the ODCs of a node in a logic circuit are defined as the set of input values for which the primary outputs of the circuit remain insensitive to the node's output. These input combinations can then be used to simplify the node because they do not affect the primary outputs of the circuit.

Applying this concept to QCC, finding the ODCs at an output of the approximate circuit (which is an internal signal in the QCC) provides the set of primary input values for which Q remains insensitive and hence do not violate the quality constraints. These inputs are referred to as *approximation don't cares* (ADCs), an entirely new class of don't care conditions. The ADCs thus obtained are used to simplify the circuit by specifying them as external don't cares (EXDCs) to the corresponding output. We know that EXDCs of an output in a circuit are the set of primary input combinations for which that primary output is a don't care. In

this case, if the approximate circuit block is considered in isolation, the ADCs for a given bit of O_{approx} can be interpreted as the external don't cares for that output. Therefore, by setting these input combinations (ADCs) as EXDCs of an output in the approximate circuit, the framework can legally simplify or (in this context) approximate the cone of logic generating that output by using standard don't care based synthesis techniques. This process is iterated for all output bits of the circuit and after each iteration, the QCC is updated with the latest available approximate circuit. Doing so ensures that the approximations performed never violate the specified constraints. Moreover, the intermediate circuit produced after each iteration is legal and synthesis can be terminated at any point to yield a valid approximate circuit.

6.3.1.2 Approximate Sequential Circuits

We now describe ASLAN, a systematic framework which extends approximate logic synthesis into the realm of sequential circuits. The need for approximating sequential circuits arises from the fact that, in general, most designers are not directly concerned with quality at the output of a combinational circuit block at the end of every cycle; rather, output quality is naturally specified at a coarser granularity, i.e., after several cycles of a sequential computation. Thus approximate sequential synthesis frameworks relieve designers from the burden of apportioning the error resilience to each individual combinational block in the design. Approximating sequential circuits is even more challenging as it requires modeling of how "errors" due to approximations are generated and propagated in each cycle of computation. Due to the cyclic nature of sequential circuits, errors may get recirculated through the approximate circuit before the outputs are generated. From a different perspective, considering the sequential nature of circuits leads to better opportunities for approximation since different cycles or circuit blocks may not have the same significance towards the output. This spatio-temporal disparity can be exploited to approximate the circuit more aggressively in less significant blocks or cycles of operation.

Sequential Quality Constraint Circuit Inspired from SALSA, ASLAN formulates the problem of sequential logic approximation by constructing a sequential quality constraint circuit (SQCC), as shown in Fig. 6.5, that characterizes the impact of approximations on the primary outputs of the sequential circuit (generated after several cycles). SQCC is a virtual circuit composed of: (i) the original sequential circuit, (ii) the approximate sequential circuit, and (iii) the quality evaluation circuit (QEC). The inputs to the SQCC are the primary inputs of the circuit to be approximated. The primary outputs and state registers of both the circuits are provided as inputs to the QEC, which evaluates the quality constraints and generates two bits *viz.* quality (Q) and quality valid (V). QEC monitors the state registers of the approximate and original circuits, and sets the valid output V when the approximate circuit output is ready for quality evaluation. QEC then compares the

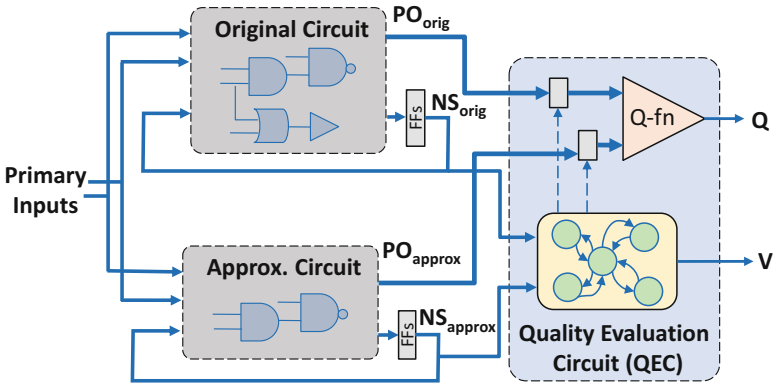

Fig. 6.5 Sequential quality constraint circuit

outputs of the approximate and the original circuit, and sets the quality output Q if the quality constraint is satisfied. The QEC is similar to a test bench used to verify the functionality of a sequential circuit with respect to golden results; in this case, the golden results are provided by the original circuit, and exact equivalence is relaxed with the quality constraint.

Formal Verification of Quality ASLAN ensures the quality constraints on the approximate circuit by formulating it as a sequential model checking problem on the SQCC. The following safety and liveness properties [4, 17] have to be satisfied for a valid approximate circuit:

1. $\Box(V \rightarrow Q)$, i.e., in all possible states of the SQCC, if V is *true*, then Q should be *true*. This ensures that the approximate circuit satisfies the quality bound when both the original and approximate circuits produce their outputs.
2. $\Diamond(V)$, i.e., V eventually becomes *true* along all possible paths through the state space of the SQCC. This property states that the original and approximate circuits should eventually produce their respective outputs.

ASLAN employs bounded model checking techniques to guarantee that the safety and liveness properties described above are preserved during synthesis.

Quality Constrained Approximation The SQCC provides ASLAN a formal method to check if the approximations introduced meet the desired quality. Given the SQCC, ASLAN adopts an iterative approach to approximate the circuit. First, it identifies combinational blocks such as arithmetic components (adders, multipliers, etc.) within the sequential circuit that are amenable to approximation. Having identified such candidates for approximation, ASLAN utilizes existing combinational approximate design techniques to generate local quality *v.s.* energy ($Q - E$) graphs for each candidate. The candidates are then approximated based on a gradient descent approach over the local $Q - E$ graphs and the SQCC formulation is utilized to verify that the global quality constraint is satisfied. The process is repeated until the circuit cannot be approximated any further without violating quality.

Thus functional approximation methodologies enable automatic design of approximate combinational and sequential circuits.

6.3.2 Timing Approximation

In timing approximation, the circuit is operated at an overscaled operating condition. For example, consider voltage overscaling [8], where the operating voltage is lowered without correspondingly decreasing the clock frequency. The lower voltage results in reduced energy consumption. However, the critical paths may exceed the clock period, resulting in timing errors at the circuit outputs. There are two key challenges that need to be addressed in overscaling based approximation. First, in most circuits, the longest paths naturally tend to lead to the most-significant bits of the output. Therefore, the errors that result from overscaling are of very large magnitude, resulting in an unfavorable quality-efficiency trade-off. Second, in timing-optimized circuits, the path-delay distribution is such that a large fraction of paths in the circuit are near-critical. This phenomenon, commonly referred as the *path wall*, results in a large number of timing errors even when the circuit is slightly overscaled. Therefore, the key challenge to timing approximation is to shape the path-delay distribution of circuits such that it ensures a more graceful degradation in quality under overscaled operation.

To this end, we present a synthesis methodology, *Relax-and-Retime*, which utilizes retiming to reshape the path-delay distribution of the circuit, achieving a more favorable energy *vs.* accuracy trade-off. Figure 6.6 illustrates this concept with an example circuit optimized for performance with a clock period of $4d$. The circuit has 3 logic cones X, Y, Z, containing paths P_x, P_y, and P_z ($P_x \ll P_y$, P_z) with delays of $4d$, $2d$, and $4d$, respectively. As shown in Fig. 6.6, it contains a significant number of paths that are critical ($P_x + P_z$) and hence even marginal voltage overscaling leads to excessive errors. After retiming (e.g., moving FF_2 forward by a delay d), the relaxed circuit possibly incurs timing errors at overscaled voltages when operated at $4d$. However, since the logic cone X has relatively fewer paths, the path wall is shifted to a lower delay ($3d$ due to logic cones Y and Z) compared to the original circuit, thereby allowing additional voltage overscaling.

Akin to functional approximation methods, Relax-and-Retime constructs a favorable path-delay distribution through retiming by mapping it to a classical minimum period retiming problem, allowing the reuse of existing synthesis tools for this purpose. The minimum period retiming algorithm is constrained by the longest path(s) in the circuit and hence does not directly target the paths with delay less than the critical delay. Relax-and-Retime relaxes this timing constraint in the circuit by eliminating selected paths that are bottlenecks to retiming. This presents the retiming tool with a relaxed version of the original problem, often providing new opportunities for retiming.

The key challenge lies in determining which paths to relax. Enumerating all bottleneck paths and setting them as false is infeasible in large designs. Relax-

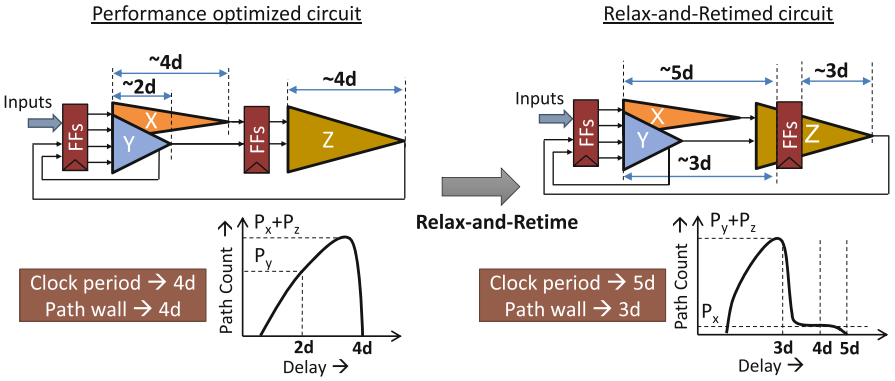

Fig. 6.6 Relax-and-Retime: approach

and-Retime addresses this issue by considering gates instead of paths. It identifies specific gates in the circuit that are bottlenecks to retiming and ignores all paths through them to relax the retiming problem. Although, this leads to a coarser selection of paths, Relax-and-Retime empirically demonstrates that relaxing at the granularity of gates is sufficient to yield significant energy benefits. To select which gates to relax in the circuit, various heuristics such as its switching activity and the slack it generates when relaxed, are utilized.

In summary, by mapping the problem of approximate circuit synthesis into a traditional logic synthesis problem and using formal methods to characterize the impact of approximations on the circuit's output quality, approximate circuit synthesis methodologies enable automatic design of "correct-by-construction" approximate versions of any given circuit for any desired quality constraint.

6.4 Quality Configurable Circuits

One of the key limitations of approximate circuits is that the degree of approximation is hardwired into the circuit implementation. However, in many applications, the degree of resilience often varies across computations depending on the application context or the dataset being processed [1–3]. For instance, consider the JPEG image compression application, shown in Fig. 6.7. In this example, each 8×8 block of image pixels is transformed to its frequency domain representation using two-dimensional discrete cosine transform (DCT). It is well known that the JPEG application output is most sensitive to the DC component at the top-left corner of the image compared to the other high frequency components. Now, if the computations associated with each component are mapped to the same underlying hardware, then it is essential to operate the hardware with different accuracies based on the significance of each component towards the eventual application output. In such

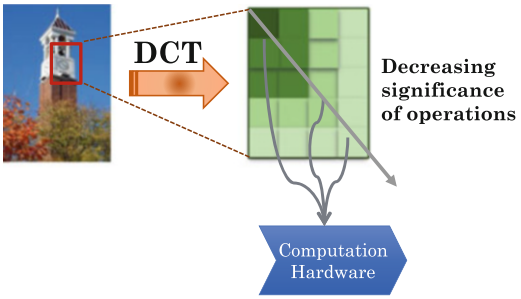

Fig. 6.7 Example JPEG compression application for quality configurable circuits

scenarios, it is desirable to design a variant of approximate circuits, referred to as *quality configurable circuits*, wherein the circuits have the ability to reconfigure their accuracy at runtime. Quality configurable circuits typically contain additional inputs that capture the current quality requirement and the circuits are equipped with the ability to dynamically modulate their accuracy and energy consumption accordingly. The quality constraint for these circuits consists of a series of quality levels that are desired during operation.

In this section, we describe the various design methodologies for automatic synthesis of quality configurable circuits. These methodologies focus on scaling the energy consumed by the circuit when the quality requirements are modulated at runtime.

6.4.1 Quality Configurability Through Variable Latency

The first methodology, Substitute-and-Simplify (SASIMI) [29], achieves quality configurable execution through variable latency operation. We illustrate the design of quality configurable circuits using Fig. 6.8, where the circuit operates in two quality modes—the accurate mode and the approximate mode. First, SASIMI introduces approximations in the circuit by taking advantage of the correlation (or similarity) that exist between nodes in any circuit. The idea is to identify near-identical signal pairs, or signal pairs that assume the same value with high probability, and substitute one for the other. For example, in Fig. 6.8, the target signal (TS) is replaced with the substitute signal (SS). These substitutions, if performed judiciously, improve power consumption, as the logic in the transitive fan-in and fan-out of TS can be downsized owing to the timing slack introduced. We note that, if the circuit were to operate only in the approximate mode, then the logic generating TS can be eliminated, saving even more power.

To enable accurate mode of operation, SASIMI operates the circuit with variable latency and *recovers* from errors due to approximations. To this end, SASIMI introduces logic to monitor the difference between TS and SS. In the accurate mode, the circuit operates in a single cycle if both TS and SS take the same

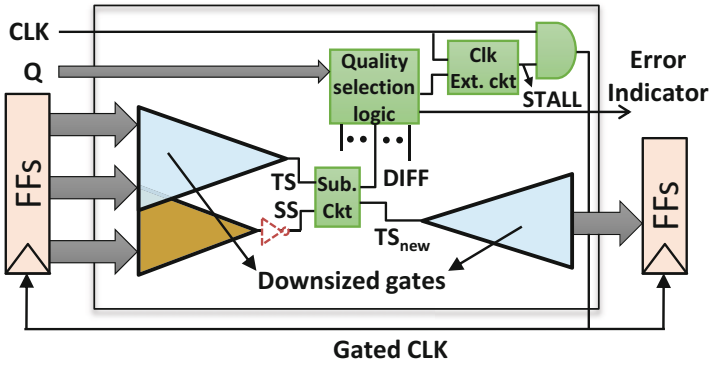

Fig. 6.8 Quality configurable circuit design using SASIMI

value. Otherwise, an additional clock cycle is utilized to re-compute the logic in the transitive fan-out of SS, thereby *correcting* the error. In the approximate mode, since the error caused by the substitution is tolerable, the difference between TS and SS is ignored and the circuit always operates in a single cycle.

In a more general scenario, where more than two quality levels are desired, the circuit recovers from a subset of substitutions that are intolerable for the desired quality level. To this end, SASIMI introduces a *quality selection logic* that utilizes the difference from all substitutions along with the input quality (Q) bits to determine the need for an additional clock cycle. Hence, starting with single cycle operation, which results in the lowest energy and quality, the quality configurable circuit progressively recovers from more and more errors (due to substitutions) as the quality constraints are tightened. Thus quality configurable execution is achieved using SASIMI.

6.4.2 *Quality Configurability through Logic Isolation*

A key limitation of achieving quality configurability through variable latency operation is that it impacts the interface timing behavior of the circuit, i.e., the resultant quality configurable circuit takes a variable number of execution cycles across inputs, even for a single quality level. Therefore, it is challenging to integrate this circuit as part of a larger system. Quality configurability through logic isolation addresses this limitation by spatially deactivating portions of logic from the circuit based on the desired quality level [10]. Figure 6.9 illustrates this concept in detail. The shaded logic portions in the circuit are identified as candidates for isolation, and the circuit is augmented with logic that suppresses them from being evaluated while forcing their outputs to predefined values. During circuit operation, the quality control unit interprets the target quality specifications, and activates a subset of

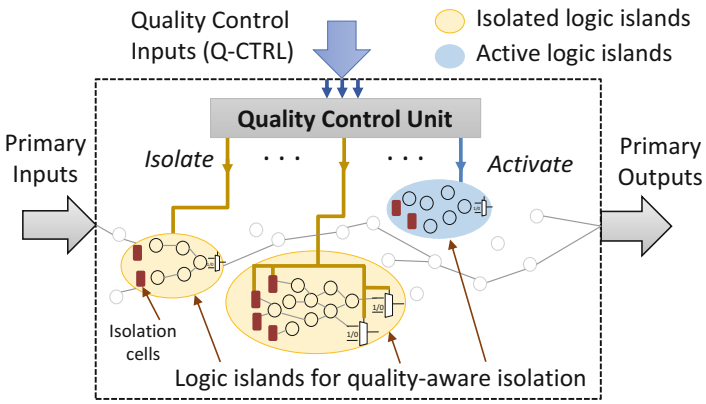

Fig. 6.9 Approximation through logic isolation: concept

these logic islands, while isolating the remaining islands. Note that when the circuit operates in accurate mode, all the logic islands are activated. As the quality constraints are progressively relaxed, increasing number of islands are isolated from the circuit. Thus, quality configurable operation is achieved at runtime.

Logic islands can be isolated in several ways. The simplest approach, *operand isolation*, uses latches or AND/OR gates at the inputs of the island to prevent switching activity within it. Similarly, muxes are inserted at the island outputs to force them to fixed logic values. An alternative approach is to *power-gate* the island by inserting isolation cells at the island outputs, and employing power gating transistors to cut off both dynamic power and leakage.

The challenge then boils down to identifying the right portions of logic to isolate for a given quality level. Clearly, considering all possible subsets of gates for isolation is not scalable. Therefore, this search space is explored in a structured manner by dividing the circuit into fan-out free cones, and using heuristics to pick the best candidate for isolation. An interesting phenomenon inherent to logic isolation based approximation is that two or more logic islands may mutually compensate their errors when isolated simultaneously, i.e., the total error at the circuit output decreases as a result of isolating both logic portions compared to isolating any one of them. This provides further room to approximate the circuit, leading to superior energy benefits for a given quality.

6.4.3 *Quality Configurability through Clock Overgating*

Both the variable latency and logic isolation methodologies focused on scaling the energy consumed by the logic when the quality constraints are relaxed. Further benefits can be achieved if the energy consumed in the clock network can also be reduced under approximate operation. This is achieved through an approach

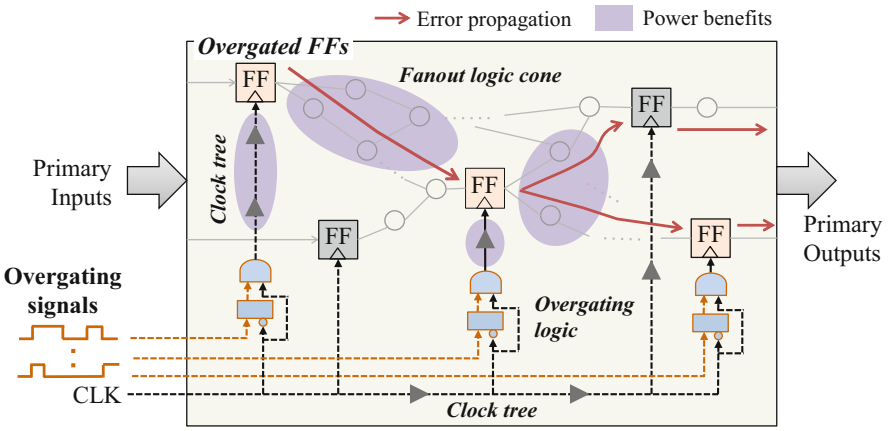

Fig. 6.10 Clock overgating: concept

called *clock overgating* [13], illustrated in Fig. 6.10, which extends the concept of clock gating to quality configurable circuit designs. Clock gating is one of the most widely adopted low-power techniques, in which clock signals to sequential elements (flip-flops or latches) in the circuit are suppressed to reduce power, provided that doing so preserves the exact functionality of the circuit. Clock overgating extends this concept to gate the clock signal to selected sequential elements, even during execution cycles when the circuit functionality is sensitive to their state. This lowers the power consumed in the clock tree, the sequential elements, and their fan-out logic cone, while introducing local errors in the circuit due to erroneous states of the constituent sequential elements. These local errors may eventually propagate to the circuit outputs, potentially degrading the output quality.

Clock overgating has the following desirable properties: (i) it is easily reconfigurable, i.e., the overgating signal to each sequential element can be regulated at runtime in a fine-grained manner to achieve different quality levels; (ii) it also preserves the structure of the circuit and hence minimally intrusive; and (iii) it exploits the existing support of clock gating in commercial off-the-shelf synthesis tools.

Given a description of a circuit, an input test bench, and a target quality constraint, a systematic methodology is utilized to identify where (in which FFs) and when (during which cycles) to perform overgating such that the energy benefits are maximized for a given quality level. The search space for overgating, defined by all possible FFs and all possible execution cycles, is extremely large (e.g., $2^{M.N}$ possible configurations for a circuit with M FFs that operate for N cycles). Naturally, a brute force search of all possible overgating configurations is infeasible for practical designs.

To navigate this large search space efficiently, clock overgating introduces two different heuristics that significantly reduce the possible overgating configurations. The first strategy utilizes internal signals already present in the circuit, e.g.,

outputs of FFs and their complements, to trigger clock overgating, thereby pruning the search space while also minimizing the logic overheads for overgating. The second heuristic groups FFs into clock overgating islands based on their functional relationship and their impact on overall application output. FFs in each overgating island are constrained to have the same overgating condition, greatly reducing the search space without drastically affecting the energy savings from overgating. Despite reducing the design space using these heuristics, exploring all possible configurations becomes computationally expensive for larger circuits. Therefore, clock overgating iteratively performs a gradient descent search to identify the set of overgating island-trigger signal combinations that maximize the improvements in energy for a target quality level.

6.5 Summary

Approximate circuits lie at the core of many approximate systems. Hence designing efficient approximate circuits is key to fully leverage the benefits of approximate computing. Approximate circuits realize a slightly different functionality from the given specifications, but they do so with disproportionately reduced hardware complexity. Approximate circuits are typically specified in conjunction with a quality metric that constrains the degree of approximation that can be introduced in the implementation. In this chapter, we outlined the key ideas behind systematic frameworks that automatically synthesize approximate versions of any given circuit and for any desired quality metric. We also described how those frameworks can be leveraged in the context of quality configurable circuits, which are provisioned to modulate their accuracy and energy at runtime. We believe approximate circuit synthesis frameworks, such as those described in this chapter, are key to the mainstream adoption of approximate computing.

References

1. Chippa V, Raghunathan A, Roy K, Chakradhar S (2011) Dynamic effort scaling: managing the quality-efficiency tradeoff. In: 2011 48th ACM/EDAC/IEEE design automation conference (DAC), pp 603–608
2. Chippa V, Chakradhar S, Roy K, Raghunathan A (2013) Analysis and characterization of inherent application resilience for approximate computing. In: 2013 50th ACM/EDAC/IEEE design automation conference (DAC), pp 1–9
3. Chippa V, Venkataramani S, Chakradhar S, Roy K, Raghunathan A (2013) Approximate computing: an integrated hardware approach. In: 2013 Asilomar conference on signals, systems and computers, pp 111–117. https://doi.org/10.1109/ACSSC.2013.6810241
4. Clarke E, Grumberg O, Peled D (1999) Model checking. MIT Press, Cambridge
5. Gupta V, Mohapatra D, Park SP, Raghunathan A, Roy K (2011) Impact: imprecise adders for low-power approximate computing. In: 2011 international symposium on low power electronics and design (ISLPED), pp 409–414. https://doi.org/10.1109/ISLPED.2011.5993675

6. Hanif MA, Hafiz R, Hasan O, Shafique M (2017) Quad: design and analysis of quality-area optimal low-latency approximate adders. In: 2017 54th ACM/EDAC/IEEE design automation conference (DAC), pp 1–6. https://doi.org/10.1145/3061639.3062306
7. Hashemi S, Bahar RI, Reda S (2015) Drum: a dynamic range unbiased multiplier for approximate applications. In: 2015 IEEE/ACM international conference on computer-aided design (ICCAD), pp 418–425. https://doi.org/10.1109/ICCAD.2015.7372600
8. Hegde R, Shanbhag N (1999) Energy-efficient signal processing via algorithmic noise-tolerance. In: 1999 international symposium on low power electronics and design, 1999. Proceedings, pp 30–35
9. Imani M, Peroni D, Rosing T (2017) Cfpu: configurable floating point multiplier for energy-efficient computing. In: 2017 54th ACM/EDAC/IEEE design automation conference (DAC), pp 1–6. https://doi.org/10.1145/3061639.3062210
10. Jain S, Venkataramani S, Raghunathan A (2016) Approximation through logic isolation for the design of quality configurable circuits. In: 2016 design, automation test in Europe conference exhibition (DATE), pp 612–617
11. Jiang H, Han J, Qiao F, Lombardi F (2016) Approximate radix-8 booth multipliers for low-power and high-performance operation. IEEE Trans Comput 65(8):2638–2644. https://doi.org/10.1109/TC.2015.2493547
12. Kahng A, Kang S (2012) Accuracy-configurable adder for approximate arithmetic designs. In: 2012 49th ACM/EDAC/IEEE design automation conference (DAC), pp 820–825
13. Kim Y, Venkataramani S, Roy K, Raghunathan A (2016) Designing approximate circuits using clock overgating. In: 2016 53nd ACM/EDAC/IEEE design automation conference (DAC), pp 1–6. https://doi.org/10.1145/2897937.2898005
14. Krause PK, Polian I (2011) Adaptive voltage over-scaling for resilient applications. In: Proc. DATE, pp 1–6
15. Kulkarni P, Gupta P, Ercegovac M (2011) Trading accuracy for power with an underdesigned multiplier architecture. In: 2011 24th international conference on VLSI design (VLSI design), pp 346–351. https://doi.org/10.1109/VLSID.2011.51
16. Lingamneni A, Enz C, Palem K, Piguet C (2013) Synthesizing parsimonious inexact circuits through probabilistic design techniques. ACM Trans Embed Comput Syst 12(2s):93:1–93:26. https://doi.org/10.1145/2465787.2465795
17. Manna Z, Pnueli A (1992) The temporal logic of reactive and concurrent systems. Springer, New York
18. Miao J, He K, Gerstlauer A, Orshansky M (2012) Modeling and synthesis of quality-energy optimal approximate adders. In: 2012 IEEE/ACM international conference on computer-aided design (ICCAD), pp 728–735
19. Mohapatra D, Chippa V, Raghunathan A, Roy K (2011) Design of voltage-scalable meta-functions for approximate computing. In: Design, automation test in Europe conference exhibition (DATE), 2011, pp 1–6. https://doi.org/10.1109/DATE.2011.5763154
20. Olivieri N, Pappalardo F, Smorfa S, Visalli G (2007) Analysis and implementation of a novel leading zero anticipation algorithm for floating-point arithmetic units. IEEE Trans Circuits Syst II: Express Briefs 54(8):685–689. https://doi.org/10.1109/TCSII.2007.896937
21. Qian L, Wang C, Liu W, Lombardi F, Han J (2016) Design and evaluation of an approximate Wallace-Booth multiplier. In: 2016 IEEE international symposium on circuits and systems (ISCAS), pp 1974–1977. https://doi.org/10.1109/ISCAS.2016.7538962
22. Ranjan A, Raha A, Venkataramani S, Roy K, Raghunathan A (2014) Aslan: synthesis of approximate sequential circuits. In: 2014 design, automation test in Europe conference exhibition (DATE), pp 1–6. https://doi.org/10.7873/DATE.2014.377
23. Shafique M, Ahmad W, Hafiz R, Henkel J (2015) A low latency generic accuracy configurable adder. In: 2015 52nd ACM/EDAC/IEEE design automation conference (DAC), pp 1–6. https://doi.org/10.1145/2744769.2744778
24. Shin D, Gupta SK (2008) A re-design technique for datapath modules in error tolerant applications. In: 2008 17th Asian test symposium, pp 431–437. https://doi.org/10.1109/ATS.2008.75

25. Shin D, Gupta S (2010) Approximate logic synthesis for error tolerant applications. In: Design, automation test in Europe conference exhibition (DATE), 2010, pp 957–960. https://doi.org/10.1109/DATE.2010.5456913
26. Shin D, Gupta SK (2011) A new circuit simplification method for error tolerant applications. In: 2011 design, automation test in Europe, pp 1–6. https://doi.org/10.1109/DATE.2011.5763248
27. Venkataramani S, Sabne A, Kozhikkottu V, Roy K, Raghunathan A (2012) Salsa: systematic logic synthesis of approximate circuits. In: Proceedings of the 49th annual design automation conference. ACM, New York, pp 796–801. https://doi.org/10.1145/2228360.2228504
28. Venkataramani S, Chippa V, Chakradhar S, Roy K, Raghunathan A (2013) Quality programmable vector processors for approximate computing. In: Proceedings of the 46th annual IEEE/ACM international symposium on microarchitecture. ACM, New York, pp 1–12. https://doi.org/10.1145/2540708.2540710
29. Venkataramani S, Roy K, Raghunathan A (2013) Substitute-and-simplify: a unified design paradigm for approximate and quality configurable circuits. In: Design, automation test in Europe conference exhibition (DATE), 2013, pp 1367–1372. https://doi.org/10.7873/DATE.2013.280
30. Venkataramani S, Ranjan A, Roy K, Raghunathan A (2014) Axnn: energy-efficient neuromorphic systems using approximate computing. In: 2014 IEEE/ACM international symposium on low power electronics and design (ISLPED), pp 27–32. https://doi.org/10.1145/2627369.2627613
31. Venkataramani S, Chakradhar S, Roy K, Raghunathan A (2015) Approximate computing and the quest for computing efficiency. In: Proceedings of the 52nd annual design automation conference. ACM, New York, pp 120:1–120:6. https://doi.org/10.1145/2744769.2751163
32. Zhu N, Goh WL, Zhang W, Yeo KS, Kong ZH (2010) Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing. IEEE Trans Very Large Scale Integr Syst 18(8):1225–1229. https://doi.org/10.1109/TVLSI.2009.2020591