Design of Fine-grained Sequential Approximate Circuits using Probability-aware Fault Emulation

David May, Walter Stechele
Technische Universität München, Germany
{david.may, walter.stechele}@tum.de

Abstract— Approximate Computing has recently drawn interest due to its promise to substantially decrease the power consumption of integrated circuits. By tolerating a certain imprecision at a circuit output, the circuit can be operated at a more resource-saving state. For instance, parts of the circuit could be switched off or driven at sub-threshold voltage. Clearly, not all applications are suitable for this approach. Especially applications from the signal and image processing domain are applicable, due to their intrinsic tolerance to imprecision. But even for these circuits, one has to be very careful where to approximate a circuit and to what extent, in order not to fall below a minimum required QoS. In this paper we are presenting an approach to generate approximate circuits from existing deterministic implementations. The flow reaches from application-driven QoS definition down to approximated RTL. We are employing FPGA-based fault emulation of the circuit in order to find out how faults, i.e. imprecisions in the circuit, affect the overall circuit behavior. Due to the FPGA-based emulation, our approach is very fast and accurate. And furthermore, it allows us to fine-granular tune the resulting precision to the required QoS, in order to get the most out of the approximation.

Keywords— Approximate computing, low power, fault emulation, FPGA

I. Introduction

Due to the increasing demand for low-power applications during the last decade, energy-efficiency has become a major concern in embedded systems design. However, voltage-scaling, the most efficient technique to save power, cannot be driven further in conventional systems. Apart from a severe performance loss, the aggressive scaling of MOSFET feature-sizes, makes circuits prone to even small voltage variations, when operating in the near-threshold area and hence susceptible to bit-flips within the circuit.

Approximate Computing

One emerging approach to tackle this problem is captioned by the term *Approximate Computing*. Approximate computing tries to increase the energy-efficiency by tolerating a certain uncertainty at the circuit outputs, i.e. the result of a operation. The term covers a variety of abstraction levels, from programming level to transistor level, as well as techniques on how to reduce the power consumption. Power can be saved for instance by calculating with a *reduced precision*, i.e. removing parts of the circuit, a method clearly related to fixed-point arithmetic. However, approximate computing introduces dynamics to this field. It proposes to switch off and on the precision depending on the actual demand of the application. Another way to save power can be to *over-scale the voltage*, and hence accept timing violations or noise-based faults. Clearly, approximate computing is not applicable for all kind of applications. Especially applications from the signal processing domain are suitable. These applications usually have an inherent imprecision tolerance. They have to deal with the human (imperfect) perception or have redundant input data in order to deal with noisy data. For a comprehensive coverage of the topic, we refer to the work of Han et al. [1].

Problem Description

Even if the applications mentioned above seem to be applicable for approximate computing, parts of the circuit cannot be easily switched off and faults cannot be tolerated everywhere in the circuit. It is crucial to know which parts of the circuit influence the functionality in which way. Ideally, we want to gain this knowledge for each gate in the circuit, to get a very detailed overview about the possible approximations. To simplify the problem we are analyzing circuits at register-transfer level, which gives a good trade-off between the precision of gate-level and the abstraction of algorithmic level [2]. However, even there the number of possible states of the circuit corresponds to 2^N, where N is the number of registers in the circuit. Hence, the difficulty of realizing approximate computing on circuit level is to efficiently and reliably determine which parts of the circuit can be approximated and to what extent.

Related Work

Approximate computing is a research field that has to be considered at multiple levels of abstraction. Furthermore, it can be applied to CPU-based applications, as well as dedicated hardware accelerators. In the former case usually the source code is annotated defining which operations can be executed on approximate hardware, e.g. as proposed in [3]. In the latter case, most research effort has been spent on building approximate standard building blocks, like adders and multipliers [4], [5]. However, most methods presented are based on manual interaction. Only a few present an automated approach, like Nepal et al. [6] did, who proposed an automated method, based on the behavioral description. Some works focus on approximation of circuits at synthesis time [7], [8]. However, only very few explicitly focus on the approximation of sequential circuits. Ranjan et al. [9] use formal verification techniques to identify the impact of approximate parts of a sequential circuit on the global output.

This seems to be a very promising approach. However, it only offers the ability to generate fixed approximated circuits. In order to save power e.g. by using dynamic voltage over-scaling or by dynamically enabling precision, an even more fine-grained analysis is required, to model varying approximation operating points.

Contribution of this Work

In order to solve these limitations we are introducing bit-flip probabilities to the analysis. By assigning bit-flip probabilities to each register in a circuit, we are able to model the various effects responsible for the approximation, from timing violations to circuit pruning. We are using FPGA-based emulation to actually flip the bits. This makes the analysis very fast and allows us to cover a large state space, so that the results are reliable. In this work we are presenting and evaluating a methodology in order to find out which parts of a sequential circuit can be approximated and to what degree. The approach is automated, wherever it is possible. Given a netlist and a set of quality constraints our tool automatically generates the information of what bit-flip probability is allowed at which register, without violating the output constraint. The approach is fine-granular. I.e., it does not only model the "integer" pruning of circuit elements, but also model a bit-flip probability. Integer approximations, i.e. if parts of the circuit can be pruned, can then be implemented directly. The non-integer information can be used to generate approximate combinational logic, e.g. based on aggressive power-scaling, using solutions on a lower abstraction level.

The remainder of this paper is organized as follows: In **Section II** we are explaining our methodology in detail. In **Section III** we will evaluate our approach with various realistic benchmark circuits. And finally, in **Section IV** we will conclude our work.

II. Methodology

A. Probability-aware Fault Emulation

In our previous work [10] we have developed an FPGA-based fault emulation system. In this work, we are using this system to detect which parts of a circuit can be approximated. The circuit we want to approximate is mapped on an FPGA, independent of the target architecture. To make the emulation as realistic as possible, the circuits have to be emulated after being synthesized for the specific target technology. In case of an ASIC target, primitives of the ASIC library have to be replaced by corresponding FPGA primitives in order to map the circuit onto the emulating FPGA. In case of an FPGA target, usually no adaptions have to be made due to the interoperability of third party synthesis tools concerning FPGA vendors and their generations. Faults, hence bit-flips, are injected at register level. By doing so we are emulating faults in the combinational logic before that register. This gives us a reasonable trade-off between complexity and abstraction. The impact of the faults on the behavior of the circuit is analyzed by comparing the output pins with those of a reference circuit running on the same FPGA. In order to perform the injection the netlist of a synthesized circuit has to be modified so that specific nets can be manipulated. We developed a tool that automatically modifies Verilog netlists. The emulation is probability-aware. Instead of injecting single faults into registers and trying to observe their effects on the output, we inject faults at all registers at once by assigning probabilities. Compared to other implementations our injection is hardware-assisted. Each register is connected to a *probabilistic bit-error generator*. The purpose of such a generator is to flip a register based on a assigned probability. Clearly, this hardware assistance results in a large hardware overhead. However, it allows us to run the emulation at real-time, while maintaining the flexibility to assign one specific error-probability to each register in the circuit. Emulation speed is an essential factor, especially as we are injecting faults based on (usually small) probabilities, in order to cover a large state-space in a reasonable amount of time. By assigning error probabilities to the individual registers we are able to model the two main approximation methods. **Voltage Over-scaling** results in an increased probability of timing violations. The effects as well as the probability of the violation are dependent on the path through the combinational logic. Hence, each combinational logic has to be modeled by a different error probability at the register input. **Circuit Pruning**, i.e. reducing the precision of a circuit by removing parts of it, can also be easily modeled. Assuming a switching activity of $\alpha_{01} = 0.5$ we can assign an error probability $p_e = 0.5$ to model the pruning of that register. Even if the switching activity does not equal 0.5, which is usually the case, we can determine the correct α_{01} of that path using simulations and assign the corresponding $p_e = \alpha_{01}$ to it.

B. Application-reasoned Approximation

Clearly, it is necessary to reason the required precision of a circuit, i.e. the maximum tolerable error probability of the output pins, from the application that utilizes that particular circuit. The first exemplary application is a generic MIMO wireless communication system. The application exists as a MATLAB model. The model generates random data, transmits it over a noisy communication channel using 8 antennas and a 16 QAM modulation. The signal is received again using 8 antennas, and the receiver tries to recover the original data using a simple *zero-forcing equalizer* by estimating the communication channel. In order to do so, the equalizer has to compute the inverse of a 8×8 channel matrix. One efficient method to do this involves a *QR-Decomposition*. In this first example we want to approximate a hardware implementation of a QR-Decomposition. The goal is to find a set of operating points for different channels (SNR [dB] $= \{30; 40; 50\}$) while maintaining a target bit-error rate at the receiver of BER $= 0.01$. The second application is a simple *Sobel* image filter, which exists as a software implementation in C. This example differs from the previous one in two major points. First, in this example the input quality is

not varying. In this example we want to generate approximations for different (in-)accuracies of the filter operation (PSNR [dB] = $\{30; 40; 50\}$). And second, the target application is CPU based. In this application we focus on floating-point operations of a FPU, hence, we want to find out what error probability can be allowed at the outputs of a FPU, depending on the operation and the target quality of the sobel filter. Similarly to the algorithm we will introduce in **Subsection II-C**, we propose a two-step approach in order to speed up the simulation. In the first step the integer approximations are determined. Usually this means, that one wants to find out which bits can be ignored. This is actually a typical fixed-point arithmetic problem [11]. In our two examples, where we only have 2, respectively 3, dimensions this is a simple operation. Starting with the least-significant bit we successively prune the bits of the result, and observe whether the quality is within the required range. The second step that we propose is a fine-granular approximation. For the remaining bits, those that cannot be pruned completely, we want to find out what error probability still can be tolerated. In order to do so, we subsequently, register by register, increase the error probability of the output bits. If the quality constraint is satisfied we continue, if not, we stop increasing the bit-error rate of that specific output.

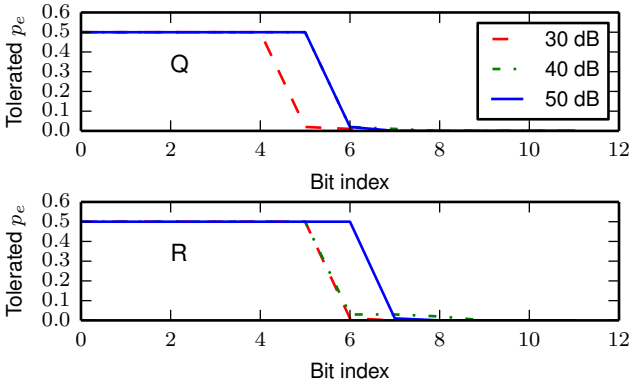

Fig. 1. Tolerable imprecision of a QR decomposition, part of a 8x8 MIMO zero-forcing equalizer, for different signal qualities and a target BER=0.01

The result of this approximation at algorithmic level can be seen in **Figure 1** and **2**. In case of the zero-forcing algorithm, we can see that both for the Q and the R part of the decomposition up to 7 bits can be pruned (visible as $p_e = 0.5$). This large number is clearly dependent on the input data and the type of application and might be less for other applications. However, what can also be seen is that there is a one bit dynamics in both parts, depending on the signal quality. This seems less in the first place, but on hardware level this multiplies as we will see later. Furthermore, we can also see that, for bits that cannot be pruned completely, a certain error probability can be tolerated. For the second application, the *sobel* filter, we can see similar results. The output bits correspond to a 32-bit IEEE 754 floating-point number. Many bits of the

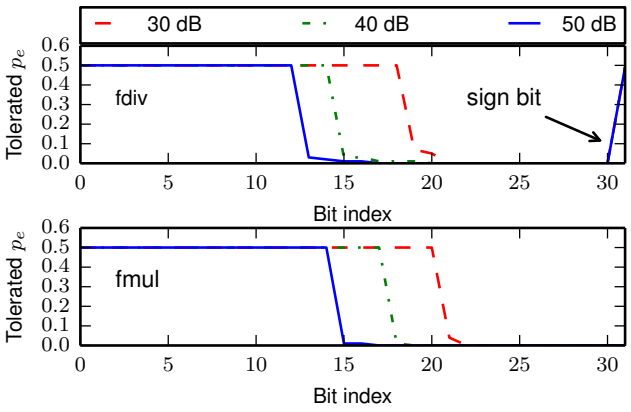

Fig. 2. Tolerable imprecision of floating-point operations in a *sobel* filter for different target qualities (PSNR)

mantissa can be ignored, which is again dependent on that specific application. Exponent bits can never be pruned. In case of *fdiv* and *fmul* the dynamics are very large, which will result in many approximations at hardware level. In case of *fsqrt*, the possible approximations are independent on the target QoS, which is why results are not shown in the Figure.

C. Approximation of Sequential Circuits

With the information generated at the algorithm level, one can now start with the actual hardware approximation. In this step we want to find out which registers in the circuit can be approximated, without violating the previously determined constraint. Hence, we are going one step down in the hierarchy. At first, it seemed that the difficulty of performing this step is to find out the one approximation combination that saves most power from a large set of possible approximations. This is why we started solving this problem using evolutionary algorithms, as it can be formulated as a global optimization problem. However, it pointed out not to be the best way. First of all it pointed out to be very unreliable due to the stochastic nature of our problem, as the same set of applied error probabilities leads to different results. Even though they are small, these differences sometimes pushed the algorithms in the wrong direction. And second, the run-time was very long, despite the FPGA-accelerated emulation. The methodology we then came up with consists of 4 steps. With each step we are eliminating approximation candidates. This makes the final, fine-grained approximation step manageable concerning its complexity.

C.1 Data-path Separation

The first step can be seen as a separation of data- and control-path. The idea is to simply exclude all registers from our potential candidates, that have an influence on outputs, where no errors can be tolerated at all. Influencing means that if there is any path from a register to an output, a fault at that register can also be visible at that output. As these outputs are usually control outputs, like address pins, we call this operation data-path separation. This

step has to be performed very careful, as every error in the control-path of a circuit, even if it is very unlikely, can have disastrous effects on the functionality of a circuit. The identification of control path-registers is pretty straightforward. We subsequently inject into each register an error probability of $p_e = 0.5$ but only at one at a time, and observe the outputs. If faults can be observed at least at one unwanted output, this register is discarded from the candidates. In order to make this approach prone to false decisions, it ideally has to be performed several times, with a large set of test-vectors [12].

C.2 High Variance Registers

The second step pointed out to be in particular necessary in case of the approximation of test circuit *QR*. We observed that the same set of applied error probabilities resulted in a highly varying output error probability. Although the errors could only be observed at data-path outputs, this behavior makes it impossible to reliably approximate a circuit. In order to trust the approximation, the results have to be reproducible, which is why these elements have to be excluded from the list of candidates as well. We are doing this using the same method as for the data-path separation, but this time we are calculating the variance over multiple runs. If the variance is above a certain threshold ϵ the register will be discarded. The threshold ϵ has to be chosen according to the requirements of the user, as it has a direct influence on the stability of the outputs of the approximated circuit.

C.3 Coarse-grained Approximation

The third step is the first approximation step. In this step we are identifying those registers that can be pruned completely. As a consequence, all elements prior to that, that only influence that one register can be pruned as well. Hence, this is the step that will save most power in the end. As we have excluded all unsuitable elements in the previous steps, this step is straightforward, as shown in **Algorithm 1**. The subset of candidates is denoted as the vector $\mathbf{c} = \{\mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_o}\}$. We are simply "switching off", i.e. injecting faults with an error probability of $p_e = 0.5$, one register by another and check whether the output error constraint is violated or not. If it is, we switch back on that register, and continue with the next one. Due to the large error probability of the injected faults, this step can usually be performed very fast. Usually only a few million cycles have to be emulated in order to get a reliable result.

Algorithm 1 Coarse Approximation

```
1:  procedure CoarseApprox.(c, p_e,max)
2:      for i ← 1, o do
3:          p_e(i) ← 0.0
4:      end for
5:      for i ← 1, o do
6:          p_e(c_i) ← 0.5
7:          p_e,outputs = injectFaults(p_e)
8:          if p_e,outputs > p_e,max then
9:              p_e(c_i) ← 0.0
10:         else
11:             AppendToArray(c_coarse, i)
12:         end if
13:     end for
14:     return c_coarse
15: end procedure
```

C.4 Fine-grained Approximation

Finally, the last step is the fine-grained approximation. In this step we want to find out which error probability, other than 0.5, can be tolerated at the circuit registers. We are step-wise raising the applied error probabilities to see if the required constraint is still satisfied. There are two main alternatives in approaching the approximation limit. One is to make the approximation at individual registers as high as possible. The other is to approximate evenly over the registers. If a register is holding data from a large combinational logic block it might be beneficial to tolerate as much approximation as possible. However, we think that tolerating a small approximation at many locations is favorable. Previous work has shown that by tolerating a small error probability, large power savings can be made in logic gates, due to an exponential correlation [13], which is a clear argument for the second approach. Furthermore, in the former case, usually the whole "imprecision budget" is consumed by very few registers, which would limit the significance of this step. At this point it would be possible to use complex optimization algorithms to find the optimal distribution of register precision. However, as mentioned earlier, for us it pointed out to bring no further improvement. Additionally, due to the first three steps our search-space is already highly sorted-out, so that we can simply iterate over the set of candidates, as shown in **Algorithm 2**. The injected error probability of each candidate gets increased step by step by a factor of δ. If the increase led to a constraint violation this particular register is removed from the candidates. This procedure is contin-

Algorithm 2 Fine Approximation

```
1:  procedure FineApprox.(c, c_coarse, p_e,max, δ)
2:      c_fine = c \ c_coarse                ▷ relative complement
3:      p_e(c) ← 0.0
4:      p_e(c_coarse) ← 0.5
5:      while (!isEmpty(c_fine)) do
6:          for i ← 1, numel(c_fine) do
7:              p_e(c_fine,i) ← p_e(c_fine,i) + δ
8:              p_e,outputs = injectFaults(p_e)
9:              if p_e,outputs > p_e,max then
10:                 p_e(c_fine,i) ← p_e(c_fine,i) − δ
11:                 removeFromArray(c_fine, i)
12:             end if
13:         end for
14:     end while
15:     return p_e,approximation = p_e
16: end procedure
```

ued until the set is empty. The result of this operation is a vector containing the maximum error probability for each register in the circuit, where a value of 0.5 corresponds to coarse-grained approximated registers and other values to fine-grained approximated registers.

III. Experimental Results

In this section we present the experimental results of our proposed methodology. We are introducing a third benchmark, a *viterbi decoder*. The precision dynamics do come in this case again due to varying signal qualities. However, the major difference to, for instance, the MIMO (QR) benchmark is, that the output of the decoder practically has to be error free. For instance in case of digital audio broadcast (DAB), a frame would be discarded if the checksum of the viterbi output is wrong, resulting in an immediate sound drop. A summary of the circuits is shown in **Table I**.

TABLE I
THE BENCHMARK CIRCUITS USED FOR THIS EVALUATION

Name	Description	Flip-flops	Technology
fpu100 [14]	32-bit fpu	2030	Synops. 90nm
QR [15]	QR decomp.	414	Virtex 6
vitdec [16]	Viterbi dec.	1297	Virtex 6

A. Approximation Evaluation

We start by evaluating the approximation itself. We want to verify if the methodology allows to efficiently approximate circuits. **Figure 3** and **Table II** are showing the approximation for benchmark circuit *QR*. One can see

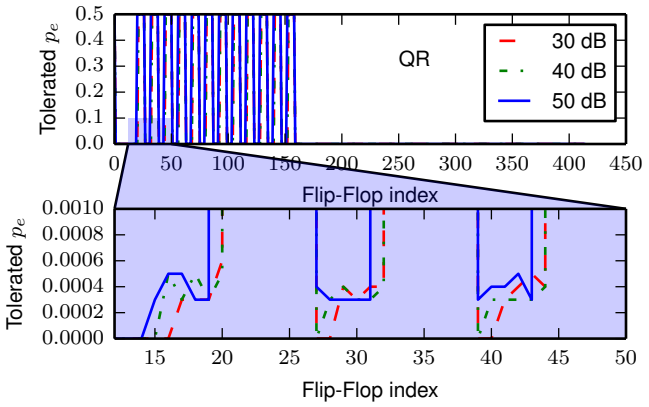

Fig. 3. Approximation result for benchmark circuit *QR*, showing the maximum tolerable error probability at each register

that only a subset of the registers was qualified for approximation. Actually only the R part could be approximated. Although the algorithm was able to detect registers influencing the Q part, all have been filtered out in the step described in **Section II-C.2**. Only those registers influencing the R part, have a predictable coherence between error probability at the register and the measured error probabilities at the outputs. However for the remaining elements an approximation was possible. One can see that the one bit reduction of precision, for a signal quality of 50dB, at application level, spreads over multiple registers at register level. Furthermore, one can see that the fine-grained approximation is also working well. The better the signal quality, the more faults can be tolerated. The results of benchmark circuit *fpu100* are similar. The sum of tolerated error probabilities over all registers is shown in **Table II**. This time the emulated target technology is an ASIC implemented in Synopsys 90nm technology. As we have already seen at application level, one can see that the possible approximations vary a lot depending on the target quality. This will also have an impact on the power consumption, as we will see in the **Subsection III-B**.

TABLE II
POSSIBLE APPROXIMATION FOR BENCHMARK CIRCUIT *QR* AND *fpu100* FOR DIFFERENT QUALITY GOALS

QR		fpu100	
Signal Qual.	$\sum p_{e,i,\max}$	Target Qual.	$\sum p_{e,i,\max}$
30 dB	36.5183	50 dB	214.5132
40 dB	36.5190	30 dB	207.0054
50 dB	42.5191	20 dB	187.0082

And finally, **Figure 4** is showing the results of the approximation of the viterbi decoder. The Figure is showing the sum of tolerable error probabilities over all registers depending on the signal quality. This circuit offers no dynamics in case of coarse-grained approximations. I.e. no parts of the circuit can be switched off and on depending on the signal quality. The large base-level of $\sum p_e = 245$ is coming from our testbench not triggering all registers (traceback length, etc.). However, in case of the fine-grained approximation the algorithm was able to find a corresponding working point for each signal quality. Hence, this circuit is in particular useful for approximate computing through voltage over-scaling.

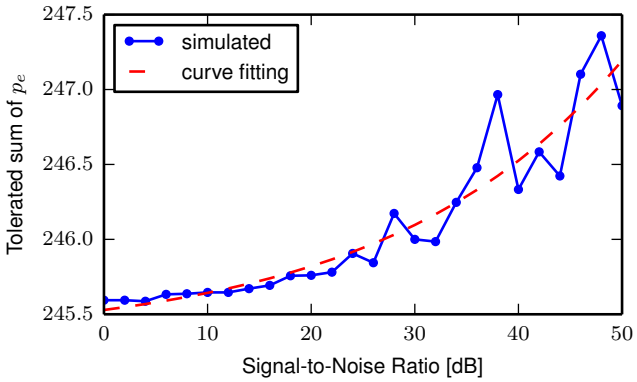

Fig. 4. Possible approximation, in terms of tolerated error probabilities, for benchmark circuit *viterbi* for varying signal qualities and a target BER=0.0

B. Power Evaluation

In this section we are evaluating the power savings enabled by the approximation presented in the previous Sections. We are able to estimate the power savings either due to circuit pruning or by clock gating the corresponding registers. However, the differences in the power consumption are only marginal. In order to estimate the power savings enabled by voltage scaling one would need an estimate of the relation between supply voltage and error probability due to timing violations. This is a complex topic on its own, hence the power evaluation of the approximated *viterbi decoder* is subject of future work. **Figure 5** is showing the estimated dynamic power consumption of test circuit *QR*. The power estimations have been made

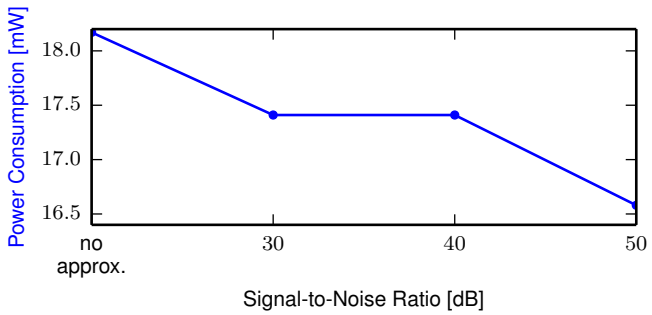

Fig. 5. Dynamic power consumption of benchmark circuit *QR* for different approximations based on varying signal qualities, when performing 8x8 ZF Equalization

for a Virtex 6 FPGA running at 100MHz. One can see that, when applying approximate pruning, compared to the deterministic original circuit, up to 1.6 mW (−8.7%) can be saved depending on the signal quality. An increase of the signal quality from 30 dB to 50 dB can save 0.83 mW of power. The power consumption of benchmark circuit *fpu100* is estimated using the Synopsys 90nm generic library again running at 100 MHz. When applying dynamic approximation, compared to the deterministic version, up to 1.79 mW (−43.3%) can be saved. Accepting a reduction of the target quality from 50 to 30dB, results in a power saving of 0.05 mW (−2.1%). One can see that the initial approximation saves most of the power, while the savings based on dynamic approximations are small in comparison.

IV. Conclusion

In this work we have presented a methodology to generate approximate circuits based on existing deterministic implementations. The information about the required quality is defined at an abstract level and propagated down to the circuit level. The approach uses FPGA-based fault emulation in order to determine the possible imprecision on circuit level. The approximation compromise four steps. The first two steps reduce the search space of potential approximation candidates in order to make the actual approximation fast and accurate. The third step approximates the circuit on a coarse-grained level, detecting which parts of the circuit can be pruned or dynamically switched off. The last step determines to what degree the remaining elements can be approximated, which can be used in future work to apply aggressive voltage scaling. Power estimations have shown that we could save up to ∼ 43% of the power consumption by applying our proposed methodology to exemplary test applications.

Acknowledgment

This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre Invasive Computing (SFB/TR 89) and by the German Federal Ministry of Education and Research (BMBF) under grant number 16M3091F.

References

[1] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Test Symposium (ETS), 2013 18th IEEE European*, May 2013, pp. 1–6.

[2] A. Herkersdorf, H. Aliee, M. Engel *et al.*, "Resilience articulation point (rap): Cross-layer dependability modeling for nanometer system-on-chip resilience," *Microelectronics Reliability*, vol. 54, no. 6, pp. 1066–1074, 2014.

[3] A. Sampson, W. Dietl, E. Fortuna *et al.*, "EnerJ: Approximate data types for safe and general low-power computation," in *Proc. of the 32Nd ACM SIGPLAN Conf. on Programming Language Design and Implementation*, ser. PLDI '11. ACM, 2011, pp. 164–174.

[4] J. Huang and J. Lach, "Exploring the fidelity-efficiency design space using imprecise arithmetic," in *Proc. of the 16th Asia and South Pacific Design Automation Conf.*, ser. ASPDAC '11. IEEE Press, 2011, pp. 579–584.

[5] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *Proc. of the 49th Annual Design Automation Conf.*, ser. DAC '12. ACM, 2012, pp. 820–825.

[6] K. Nepal, Y. Li, R. Bahar, and S. Reda, "ABACUS: A technique for automated behavioral synthesis of approximate computing circuits," in *Design, Automation and Test in Europe Conf. and Exhibition (DATE), 2014*, Mar. 2014, pp. 1–6.

[7] J. Miao, A. Gerstlauer, and M. Orshansky, "Approximate logic synthesis under general error magnitude and frequency constraints," in *2013 IEEE/ACM International Conf. on Computer-Aided Design (ICCAD)*, Nov. 2013, pp. 779–786.

[8] D. Shin and S. K. Gupta, "Approximate logic synthesis for error tolerant applications," in *Proc. of the Conf. on Design, Automation and Test in Europe*. European Design and Automation Association, 2010, pp. 957–960.

[9] A. Ranjan, A. Raha, S. Venkataramani *et al.*, "ASLAN: Synthesis of approximate sequential circuits," in *Proc. of the Conf. on Design, Automation & Test in Europe*, ser. DATE '14. European Design and Automation Association, 2014, pp. 364:1–364:6.

[10] D. May and W. Stechele, "An fpga-based probability-aware fault simulator," in *Embedded Computer Systems (SAMOS), 2012 International Conference on*. IEEE, 2012, pp. 302–309.

[11] R. Yates, "Fixed-point arithmetic: An introduction," *Digital Signal Labs*, vol. 81, no. 83, p. 198, 2009.

[12] D. May and W. Stechele, "Improving the significance of probabilistic circuit fault emulations," in *On-Line Testing Symposium (IOLTS), 2014 IEEE 20th International*. IEEE, 2014, pp. 128–133.

[13] P. Korkmaz, B. Akgul, L. Chakrapani, and K. Palem, "Advocating noise as an agent for ultra low-energy computing: Probabilistic cmos devices and their characteristics," *Japanese Journal of Applied Physics*, vol. 45, no. 4B, pp. 3307–3316, 2006.

[14] J. Al-Eryani. (2007) fpu100 at opencores.org. [Online]. Available: http://opencores.org/project,fpu100

[15] C. Gimmler-Dumont, P. Schlafer, and N. Wehn, "ASIC implementation of a modified QR decomposition for tree search based MIMO detection," in *2013 IEEE Fourth Latin American Symposium on Circuits and Systems (LASCAS)*, Feb. 2013, pp. 1–4.

[16] M. Fehrenz. (2014) Viterbi decoder at opencores.org. [Online]. Available: http://opencores.org/project,viterbi_decoder_axi4s