International Journal of Engineering Research and Technology. ISSN 0974-3154, Volume 14, Number 3 (2021), pp. 278-288
© International Research Publication House. http://www.irphouse.com

Low Power Field - Programmable Gate Arrays Using Approximate Computing

Murali Dova[1], Anuradha M Sandi[2]

[1]*Research Scholar, Guru Nanak Dev Engineering College, Bidar, Karnataka, India*

[2]*Associate Professor, Department of ECE, Guru Nanak Dev Engineering College, Bidar, Karnataka, India*

Abstract

In the computing arena, Integrated Circuits (ICs) are created with Very Large Scale Integration (VLSI) technology since 1970s. Right from its inception, VLSI witnessed rapid growth in terms of its usage in the industry. However, it needed appropriate customization with programmable logic components. Field-Programmable Gate Array (FPGA) with its Configurable Logic Blocks (CLBs), I/O pads and routing channels is an ideal design style of VLSI. VLSI is also known for its energy efficient design. Moreover, FPGA design can be leveraged with approximate computing phenomenon in embedded computing devices resulting computing efficiency and conservation of energy besides reducing overhead. Approximate computing, unlike exact computing, can significantly reduce computational complexity and exhibits acceptable error tolerance. Thus FPGA – VLSI design method for ICs is improved further with approximation computing techniques. Keeping the pivotal role of these techniques in mind, they have attracted attention of researchers and academia. In this paper we focus on different approximate computing methods, their pros and cons. It provides valuable insights pertaining to FPGAs using approximate computing besides finding research gaps in this area.

Keywords – VLSI technology, FPGA, ICs, approximate computing, design styles

1. INTRODUCTION

Modern applications associated with financial analysis, social media, scientific computing and big data analytics demand highly sophisticated computing and storage facilities. In is understood by researchers that data centres need to handle 50 times more information in the next decade and there is need for 10% increase in the number of processers. It shows clearly the rise in performance demands which will reflect in resource allocations and budgets and overprovisioning should not be an ideal solution. In its large spectrum, this problem, can have a promising solution in the form of a computing paradigm known as Approximate Computing (AC).

Approximate Computing (AC) is a technique of computation which may return a probabilistic (approximate) value rather than very accurate result. Such computing is best used in applications that expect a value approximately. An approximate value is sufficient for the given purpose in those applications. Interestingly AC is based on the observation that exact computation demands consumption of more resources. Thus selective approximation can help in achieving performance while satisfying needs of certain applications. AC is devised to be an alternative to the computing phenomenon known as exact computing [1]. AC often trades off effort needed with quality of computing to meet demands in terms of performance and constrained budgets. In this sense, AC has become not only attracting but also imperative. There are various processing units, memory technologies and components where AC can be used effectively. The processing units include Central Processing Unit (CPU), Graphics Processing Unit (GPU) and FPGA.

There are plenty of applications in the real world that do not need exact computing. For instance, neural approximation, machine learning algorithms and other host of methods in Artificial Intelligence (AI) can tolerate errors or inaccuracy without compromising the end result that is arriving at correct decision. In this sense, AC can leverage the presence of code regions that are error-tolerant. It will help in achieving trade-offs between accuracy and gains in energy and performance benefits. There is gap between level of accuracy given by the computing system and the expectation of an application. This gap is where AC comes into picture and exploits it for its benefits. Nevertheless, AC needs to be used appropriately. It needs judicious decision making otherwise it leads to mediocre performance. The full potential of AC in the contemporary world can be achieved by addressing certain issues. Many techniques came into existence of late for maximizing benefits of AC.

Table 1: Shows abbreviations used in this paper

Abbreviation	Description
AC	Approximate Computing
ACT	Approximate Computing Technique
AI	Artificial Intelligence
BW	Bandwidth
CLB	Configurable Logic Blocks
CPU	Central Processing Unit

Abbreviation	Description
FP	Floating Point
FPGA	Field-Programmable Gate Array
GPU	Graphics Processing Unit
IC	Integrated Circuit
IO	Input Output
LSTM	Long Short-Term Memory
LVA	Load value approximation
MCMA	Multiple-Classifiers and Multiple-Approximators
MFU	Memory Fence Unit
MIS	Macro Instruction Sequencer
NN	Neural Networks
NPU	Neural Processing Unit
PDPE	Peak Dynamic Power Estimation
QoR	Quality of Requirements
QoS	Quality of Service
RAM	Random Access Memory
RRAM	Resistive Random Access Memory
SW	Software
TC	Texture Cache
TCAM	Ternary Content Addressable Memory
UART	Universal Asynchronous Receiver and Transmitter
VLSI	Very Large Scale Integration

Various abbreviations used in this paper are provided in Table 1. The contributions in this paper include review of literature on FPGA designs and AC techniques for leveraging the circuit development. It provides valuable insights on different aspects. The remainder of the paper is structured as follows. Section 2 presents possible opportunities and challenges of using approximate computing in large scale applications. Different approximation strategies are explored in Section 3. Section 4 provides approximate computing techniques used with FPGA. Section 5 provides summary of findings while section 6 concludes the paper and also gives directions for future work.

2. OPPURTUNITIES AND CHALLENGES OF APPROXIMATE COMPUTING

Approximate computing provides plenty of opportunities to the world of computing. The computational phenomena without AC results in overheads that may lead to severe limitations in machine critical applications. AC can reduce power consumption in embedded systems where it is more than required. The applications that have tolerance to errors to some extent can gain maximum benefits with approximate computing. In this area, cloud data centres and computing equipment can gain from this by saving computational effort and energy consumption. In many areas AC creates opportunities. They include big data analytics, search operations, machine learning and multimedia processing to mention few. AC can save power from 5% to 40% [1]. The saving of power depends on kind of application and the level of error-tolerance capability of the application. AC can accelerate RMS (Recognition, Mining and Search) applications. The following are specific challenges of AC.

2.1 Domain Limitations

Some application domains are not suitable for AC. For instance, cryptography is having a class of applications that cannot use AC. There are some applications where approximation is allowed to some range of values. For instance, sin function approximation and approximate inverse operation have limited possibilities. Moreover, the gains associated with AC are bounded. For instance, inexact storage will not be able to leverage AC much.

2.2 Issues Related to Correctness

In presence of aggressive technologies related to AC, there are some correctness issues. They may prevent termination of program due to unsolvable problems, may result as corrupt input that cannot be validated by quality metrics. Or the result does not meet the quality needs of an application. AC used in compression technique may provide corrupt output. It also may interfere with operations like memory ordering and synchronization thus resulting into a non-deterministic output that does not help in debugging correctly.

2.3 Need Strategies Specific to Applications

There are some strategies related to approximation that are to be used based on applications in hand. For instance, uniform approximation may not be suitable for all kinds of applications. There are many strategies of AC such as memorization, precision scaling and so on. As one size does not fit all, there is no AC technique that can be universally applied. Therefore, AC strategies need to be application dependent.

2.4 Scalability and Overhead Issues

There are many overhead issues associated with AC that may limit scalability as well. For instance, voltage scaling implementation causes much overhead as it needs voltage shifters that move data between domains. In the same fashion, NN implementations involve signal conversions between analog and digital domains. There are some applications where developers need to write many versions of AC code. This may not be able to scale well. There are some applications that need ISA extensions leading to overhead.

2.5 Achieving Configurability and High Quality

As AC techniques need to consider desired level of QoR and ensure configurable codes there might be trade-offs between efficiency and quality. If QoR is less than a given threshold, then precise computations are necessary which leads to costs in design and verification. Sometimes, it may defeat the purpose of the AC usage itself. Therefore, it is essential to monitor and adapt to the situations at runtime. This may lead to infeasibility of AC technologies.

3. APPROXIMATION STRATEGIES

This section provides many strategies of AC that can help in improving performance of many real world applications.

3.1 Precision Scaling

There are many ACTs that try to reduce storage and computational power by adjusting precision needed or by changing bit-width. Yeh *et al.* [20] proposed a precision scaling method which is dynamic in nature. It could perform profiling process at the time of design. Experiments are made with runtime observations in terms of energy consumption difference among the steps in order to find possible instabilities. When an instability s found, precision is restored with certain increase in it. Later on it is reduced progressively based on the runtime experiences of the program. They found that there are three optimization possibilities with precision reduction. First, it converts FP operation into simple operation so that it does not required FPU. Second, localization is improved in similar scenarios which will improve coverage with respect to memorization technique. Third, it results in faster and smaller FPUs for various computational tasks. Moreover, it helps in having levels of FPUs like Level 1 and Level 2catering to different purposes. Based on precision different levels of FPUs are executed leading to optimizations.

Tian *et al.* [21] explored a precision scaling technique associated with off-chip with respect to data access in order to reduce energy consumption. They employed the techniques to solve problems of clustering with a mixed model where large volumes of data of off-chip is needed. Based on the intensity of operations, it is possible to reduce precision and increase performance. As clustering process is iterative in nature labelling can be done in a lazy approach and thus correctness is achieved besides leveraging performance and flexibility. In each iteration precision is chosen based on the runtime manifestations and based on the tolerance of application to errors. The bit-width is adjusted in memory intensive applications.

3.2 Loop Perforation Technique

There are ACTs that exploit loop perforation. It is used to skip some of the iterations and thus reduce overhead on computational costs. In this regard, Sidiroglou *et al.* [22] found many computational patterns at global level that help in loop perforation and thus the result is to reduce the search space in many algorithms. Thus it exploits the trade-off between accuracy and performance. Two algorithms are investigated to know the efficiency of loop perforation approach. When the application is error-tolerant, this technique is useful in choosing optimal performance and accuracy based on Pareto-optimality concept. A heuristic is used in order to perform prioritization. There is performance and accuracy bounding that could help in improving performance and also see that the result is acceptable.

3.3 Load Value Approximation

When there is an event like load miss in memory cache, the data needs to be obtained from the other cache or RAM. It causes more latency that deteriorates performance. In this regard, (LVA) exploits the nature of systems to find the load values and make necessary steps related to processor without causing stoppage of processing. Thus it can effectively hide the latency caused by cache miss. There are many LVA based AC techniques.

Miguel *et al.* [23] studied LVA approaches for multimedia applications. When they compared the LVA of traditional predictors, they could fetch blocks needed in order to have training for approximator. This will get rid of the need for fetching the data block when there is corresponding cache miss. Thus it can reduce the usage of memory considerably. As multimedia applications have capability of tolerating errors, AC values can be used well without the need for rollback in case of small differences in the results. Confidence estimation issued by the technique to ensure AC even when expected accuracy is relatively high. The degradation of quality of output (negligible though), the technique improves performance of the system besides saving energy.

Yazdanbakhsh *et al.* [24] proposed AC technology which allows setting of BW constraints and latency with respect to computational designs such as CPUs and GPUs. Based on the annotations used in the AC coding, memory access and flow of control operations are identified and performance is increased. The loads that really cause large number of misses are chosen in order to have performance improvement purposefully. As the loads are involved in subjective approximation, the quality may be deteriorated due to selection of approximation. However, it is essential to improve performance and ensure acceptable accuracy. It is also possible to reduce BW bottlenecks with this strategy which is very useful in case of GPUs. Some part of cross misses is also skipped in order to reduce BW bottleneck. When cache misses are removed, it can lead to avoiding long memory stall and the drop rate can be effectively controlled. Thus efficiency and quality are balanced effectively.

With respect to GPU, every request related to data an SIMD load is essential to satisfy multiple threads. Each thread’s value is predicted which led to increase in overhead. Thus value of similarity is considered across data access scenarios in different threads in order to have a predictor for multi-value items. Specialized indicators are used in parallel. Parallel predictors with their specializations can lead to efficiency. Miguel *et al.* [23] showed proof of energy efficiency in both CPU and GPU with bounded loss of QoR. Sutherland *et al.* [25] on the other hand used LVA approach in order to optimize GPUs with memory stalls reduction. They used an ACT that exploits inexact values that are sufficient for processing. Texture Cache (TC) and threads with associated texture fetch units, to exhibit capabilities related to interpolation between cache units in similar neighbourhood. It will improve performance in processing FP data with TC and its indices. Thus spatial and temporal correlation has its impact on the performance. Thus AC has its values obtained from the TC. As some approximations are pre-loaded in TC, it will help in speedy processing. They also proved that their approach could boost in performance besides reduction in quality loss and that is useful to many applications that are data-intensive.

3.4 Memoization Techniques

Memoization is an important approach that allows saving results of procedures to use them later time for similar functions. This has dramatic effect in improving the performance of operations. This way the scope of memoization will be enhanced in case of similar inputs and similar functions. This is made possible with approximation. Many ACTs follows this kind of approach. Rahimi *et al*. [26] specified that architecture related to SIMD allows parallel programming besides exploiting value locality. Thus the ACT can reuse as much of data as possible leading to reduction in overhead and errors. Unlike temporal memoization, spatial memoization is used by them for error free running of the program on the data and reuse as much as possible. With masking of inputs, it is possible to use ACT while precise bit by bit matching will not allow usage of ACT.

Reuse of instructions and simplifying FP operations, it is possible to avoid errors and keep the quality loss in acceptable bounds. Keramidas *et al*. [27] opined that modern applications that involve in graphics extensively depend on high precision. However, such applications cannot exploit ACTs unless there are flexible means used in order to use AC techniques. By enabling them reuse of approximate values, they can be proved to be successful in saving resources. In order to do so, they use a value cache concept. The reduction of accuracy is possible and that is acceptable in many applications.

With respect to fragment shaders, the approximation is possible in order to gain final colour value associated with a pixel using math operations that depend on texture fetches. By relaxing the precision, it is possible to have speedy operations with negligible compromise on the quality of outputs. Thus more aggressive relaxation of precision in math operations can be done to enhance texture related graphics operations. For instance, 12 bits can be changed to 4 bits' precision. This will maximize the value reuse. When there is less error rate and also speed is sufficient, it is also possible to increase precision dynamically. This will help in flexible operations where resource optimization and accuracy are balanced.

3.5 Skipping Memory Access and Tasks

There are many ACTs that follow the strategy of skipping memory references to see the performance improvement that is bounded to QoR loss. For instance, Samadi *et al*. [28] proposed ACT based on SW that finds similar patterns for approximation of each pattern. In case of both GPU and CPU architectures, this kind of approach yields good performance. They also used memoization in case of memory access patterns. With respect to prediction patterns, the concept of sampling is employed in order to reduce computational complexity. In case of scan patterns, the scan is minimized to only subset of arrays and the rest of the things are predicted to reduce time taken and complexity in math operations. With partition and stencil patterns, neighbours having same values is used for approximation. OpenCL and CUDA platforms are well used for this kind of approximations. In every aspect there is trade-off between performance and quality of outputs that are balanced. Therefore, their approach to have programs that can be executed in many HW platforms to have automatic optimizations. When compared to the executions with precision, their AC approach could gain many performance benefits besides achieving acceptable quality loss.

Goiri *et al*. [29] explored distributed environments like MapReduce to achieve approximation. They used strategies that can be employed in such frameworks. For instance, they used input data sampling and also task dropping besides using approximation code for tasks. Statistical theories are employed to take approximation to the next level and reduce space and time complexity in Map and Reduce operations. They used Hadoop for the implementation and ensured job executions to be more efficient. They also focused on the trading of accuracy for energy efficiency and performance.

3.6 Multiple Versions of AC Code

ACTs may be available in multiple versions. Each version code can have different capacity in trading off overhead and accuracy. As explored in Samadi *et al*. [30] ACT in case of GPUs have lot of potential. They used a two-phase technique. It has a static compiler association with CUDA with multiple versions for offline communication. For the purpose of runtime, they used a greedy algorithm that is able to adjust parameters and achieve approximation effectively. They also used GPU specific architectures to explore approximation. Further optimization is achieved with thread computations that reuse threads in tune with AC strategies. Approximate kernels are employed with GPUs to achieve optimal benefits besides having desired quality in outcomes. Baek and Chilimbi [31] proposed a model for programming in order to have approximation specific to QoS targets. Approximate versions of functions are used to get this done. Loss in QoS is bounded and thus performance is achieved. Two phases are considered such as training and operations. In the former phase AC technique generates a knowledge model and in the operations phase, it uses the model for appropriate approximation without quality loss.

In this aspect, design of inexact circuits and then using ACTs can help in approximation at architecture level. Kahn and Kang [32] made experiments with an exact adder. When there is exactness, the performance went down and when there is relaxation of exactness, it led to improving performance. With optimization parameter value of k, the outcomes became correct even when approximation is used. However, it needed bounding to ensure balance. A novel design is presented by Kulkarni *et al*. [2011] for inexact 2x2 multiplier. It represents a match operation with 1112 which is equal to 112*112 rather than 10012. It is able to use three bits instead of 4 bits saving one bit. There are many input combinations that will make use of this approach and save resources. There is trade-off between power saving and error rate. It is suitable for error-tolerant applications [33].

Venkataramani *et al*. [34] used a method for creating inexact circuits that are based on the specification named Register Transfer Level (RTL). It has associated QoR metric and also circuit in hand. According to this there is insensitive quality function applied to input values in order to see that output is there with approximation. The application doesn't worry about approximation as it gets outputs with reasonable quality for decision making. The dynamic approach followed in programming to exploit inexact approaches led to further enhancement in performance. They achieved both multipliers and adders with approximation and reduce complexity besides minimizing energy consumption.

Ganapathy *et al*. [35] provided a method for reducing errors as much as possible. They used unreliable memories that is not similar to ECC approach which takes care of correcting errors. With every write operation, there is shifting of data-word that leverages performance and improves reduction of faults. Thus errors are skewed towards a side and reduce error rate as well. It also strikes balance between quality and errors by adapting bit-shifting. The performance is improved when compared with the traditional zero-failure approaches.

Yet another approach made by Yeti *et al*. [36] is that it involves approximate executions with a faulty processor. Macro Instruction Sequencer (MIS)is employed along with Memory Fence Unit (MFU) with I/O streaming in order to have dividends in performance. The observations with threading and coarse-grained chunks of functions, MIS flows are found to be limiting the operations needed and thus complexity is reduced. MFU is capable of skipping references related to memory or use a dummy location in order to suppress faults. MFU communicates with the MIS to see that certain faults are recovered. In case of bounding of I/O, they found that I/O intensive applications needed AC techniques to have optimized performance.

3.7 Voltage Scaling

This is a concept in which energy consumption is reduced when circuits are used by allowing errors to some extent according to Mittal and Vetter [37]. As an example, reduction in SRAM supply voltage can save energy consumption and improve possibility of write failure with bounding anyway as opined by Sampson *et al*. [38]. Many ACTs are using voltage scaling concept and have bounding for trade-off. For instance, multiple level approximation is used by Chippa *et al*. [39] with an ACT technique. Their strategy, at architecture level, exploits intermediate values to have approximation and improve scalability. It led to saving energy and at circuit level, it improved performance with smaller bit-width. It helps in controlling errors and low-cost correction circuits are used to get it balanced. Achieving approximation at different levels and ensuring less QoR loss providers improvements in any design such as GPU, CPU and FPGA.

Rahimi *et al*. [40] proposed an ACT to work with GPU designs. It targeted to save energy consumed by GPUs. With FPU and Ternary Content Addressable Memory (TCAM) it is possible to have controlled scaling and avoid over scaling as well. Thus it is made possible to design error-resilient GPUs.

Grigorian and Reinman [41] proposed a method to overcome branch divergence problem in the architectures of SIMD. Data dependent flow of control is characterized and the kernels are used in order to improve performance and reduce losses due to the divergence issue. NN based approximation kernels are employed in order to achieve this. When NNs are injected into code, the divergence issues are removed but at the cost of loss in quality. Satori and Kumar [42] proposed two techniques called branch herding and data herding. These are meant for reduction in memory usage and also control divergence to have GPU applications to be error-resilient. In case of GPUs, there is load instruction that witnessed many requests that may ultimately lead to memory divergence. To overcome this problem, memory coalescing is used to control such loads and achieved desired outcomes.

3.8 Usage of Accelerators Based on Neural Networks

Acceleration is essential in circuit designs. NN techniques support this kind of acceptation with parallel approaches. There are ACTs that support mapping of NNs to approximate code regions. Esmaeilzadeh *et al*. [43] focused on the functioning of approximable codes. They exploited programmatic approach with low-power NPUs. They also used ISA extensions in order to have better configuration of NPUs. As NNs can be trained automatically after discovery, effective approximation is achieved. Olukotun [44] provides a SW based approach that can accelerate different applications with NNs. Ansel *et al*. [45] on the other hand produced an application structure with different matrices and mapped to task graphs for flexible resolution levels. Granularity of approximation is thus reduced and performance is improved besides witnessing minimal errors.

Eldridge *et al*. [46] proposed an approach based on MLP and NN based for approximation of FP operations. They include pow, log, exp, sin and cos. They employed a neuron pipeline to have better processing. They also used SW library in order to exploit accelerators with usage of benchmark suite named PARSEC. Anolog approaches may lead to noise, limited accuracy and circuit imprecision. Different approaches are employed to overcome this problem ad see that the energy efficiency is improved. Li *et al*. [47] proposed a framework known as ReRM-based AC unit to improve approximate computations. It has 3 layers with NN and designed with HW. Matrix vectorization is performed with NN approximator to achieve approximation with acceptable quality loss.

Approximation of NNs is carried by Zhang *et al*. [48]. The notion of candidates for approximation is used. Neurons are exploited to have approximation with given quality range.

With an iterative approach optimization is increased and at the same time the bounding of quality outcome is revised. Du *et al.* [49] designed HW NN accelerator which inexact design. It used feed-forward NN with 2-layers with approximate computations for reducing resource consumption. There were improvements with respect to energy, delay and area.

3.9 Summary of Approximation Strategies

This section provides summary of findings on the approximation strategies found in the literature.

Table 2: Presents summary of approximation strategies

AC Strategy	References	Key Benefits	Limitations
Precision scaling	[20], [21]	Improves performance with QoS binding	-
Loop perforation and load value approximation	[22], [23], [24], [25]	Improves flexibility and causes faster processing	-
Memoization	[26], [27]	Significant dynamic power saving, reduction in area and delay	-
Task skipping	[28], [29]	Reduces processing time and resources needed	-
Memory access skipping	[28], [29]	Reduces memory I/O operations and improve speed and reduce resource utilization.	-
Data sampling	[28], [29]	Reduces complexity of processing.	-
Versions of different accuracy	[30], [31], [32], [33], [34], [35], [36]	Suitable for different approaches, flexible	-
Usage of faulty HW	[23]	Reduces cost and improves performance	
Voltage Scaling	[37], [38], [39], [40], [41], [42]	Addresses problem of power saving	-
Reduction of refresh rate	[19]	Refresh rate reduction improves optimization	-
Inexact reads/writes	[23], [24], [24]	Saves memory, time and processing power.	-
Divergence reduction in GPUs	[41], [42], [43]	Optimizes GPU performance	-
Lossy compression	[41], [42], [43]	Improves performance, reduces area	-
Use of neural network	[48], [49], [50]	Better approximation and performance	-

As presented in Table 2, there are many approximation strategies used in different contexts. It is important to understand the suitability of a particular AC technique based on FPGA design targeted to particular architecture.

4. APPROXIMATE COMPUTING TECHNIQUES FOR FPGA

Moreauetal. [50] focused on the approximation code that can be programmable with a chip known as Sock with respect to FPGA. It is useful in order to offload some codes using compiler to FPG and also see that programmers will have fine-grained control over the programmable approach with FPGA. When it tends to face challenges with respect to FPGA and CPU, the differences in speeds there still possibility to increase or accelerate by invoking NNs appropriately. At the same time, the accelerators do not see that the program is blocked from execution due to many invocations of accelerators. This technique helps in implementing efficient and off the shelf FPGAs even without having close integration with the computing design like CPU. Thus it is possible to eliminate changes made to ISA of the processor in question to enable neural acceleration in the available devices.

Their technique also helps in configuring NN topologies and compute weights in order to have optimizations. Thus even in higher level tools of synthesis, the approach can help in accelerating different ranges of applications. They could observe energy saving besides speed in the process that is bounded to reduction of QoR. Lopes *et al.* [51] identified that it is possible to use different iterative solvers in order to ensure desired level of QoR by reducing precision of computations that take place as intermediately. Unlike direct solvers, the intermediate operations with less precision can reduce complexity and improve performance. With respect to FPGA, lowering precision can help increase parallel operations and enhance its performance. Thus an ACT is proposed by them towards accelerating solution where mathematical linear equations are employed dynamically on FPGA. The convergence is found to be good. From the results it is evident that the computation time is reduced besides achieving desired QoR. There is thus balance between iteration count and operation precision adjustment in case of large applications that need optimal performance with double-precision implementations meeting desired level of QoR.

Sinha *et al*. [1] used memorization technique in order to have better design of low power FPGA. It was targeted towards optimizations on FPGA with different parameters related to architectural design. High level of synthesis is enabled with the proposed memorization based solution. It could reduce up to 20% power saving and achieve less overhead that is lower than 5%. Memoization is also studied by Echavarria *et al*. [3] to understand the reasons for reducing power consumption with AC. They observed that memorization techniques do consume less power and the same is proved with a simulation study. Rizakis *et al*. [2] on the other hand employed AC with FPGA using deep learning technique known as Long Short-Term Memory

(LSTM). Their LSTM based FPGA design with AC showed 6.5 times better speed and 25 times higher accuracy of the system in spite of time constraints in computations. Pandey and Pattanaik [4] witnessed implementations of FPGA for realizing CPU design with low power settings and that is clock gating aware. Thus VLSI design could gain benefits of power optimizations. Such designs include intelligent CG, flip-flop based CG, latch-based CG and latch-free CG. Efficient designs at circuit level with FPGA led to energy saving.

A system identification approach to have an FPGA design is explored by Hung *et al*. [5]. It could achieve per module saving of power with the proposed approach. System wide saving of power reached to 8%. Programmable optimization with FPGA is explored by Lazorenko and Chemeris [6]. They opined that more power consumption is made by memory. This is reduced by their design based on low power loop optimization technique for realizing FPGA design. Loop fusion concept is used for optimization of program. Gaillardon *et al*. [7] employed Resistive Random Access Memory (RRAM) integration for realizing an ultra-low power FPGA which improves performance without significant compromises. Low power FPGA architecture CLBs is described well in [8] while FPGA mapped designs are explored in [9] for peak dynamic power estimation in FPGA designs. Tang *et al*. [10] achieved two objectives such as delay and energy saving with RRAM-based FPGA architecture.

Verma *et al*. [11] focused on low power techniques towards designing digital systems. They looked at clock design considerations, RTL coding styles, flattening and factorization, operand isolation, pre-computation and ACPI module. Singh *et al*. [12] explored on FPGA based design of Universal Asynchronous Receiver and Transmitter (UART) and used Verilog for simulating the same. He *et al*. [13] invented many FPGA circuits and approaches for low-power consumption. They could produce a patentable work. Winzker [14] contributed towards the clear description of overcoming low power electronics with FPGA designs while Rashidi *et al*. [15] achieved FPGA design based on digital FIR filter in a synthesized fashion. AbuShanab *et al*. [16] simplified the concepts of FPGA design. Chouhan *et al*. [17] threw light into low power FIR filter designs. RAM implementations can also be made with FPGA. It is evident in the work of Singh *et al*. [18] where they implemented a circuit related to RAM with FPGA with IO standards. Song *et al*. [19] employed neural approximate computing to realize designs based on Multiple-Classifiers and Multiple-Approximators (MCMA) with a Neural Processing Unit (NPU) chip. All FPGA based designs are found to be highly flexible and reduce power consumption and increase performance when they are coupled with AC technologies described in Section 3.

5. SUMMARY OF FINDINGS

This section presents the summary of findings with respect to AC techniques and the usage with FPGA based designs.

Table 3: Shows summary of findings related AC techniques used for FPGA designs

Authors & Year	Techniques Used	Merits	Limitations	Remarks
[1] Sharad Sinha and Wei Zhang (2016)	Memoization based AC	Dynamic power saving, less area overhead, better power to signal noise ratio	Automatic search for approximate design is still desired.	Used in low-power FPGA design
[2] Michalis Rizakis, Stylianos, Venieris , Alexandros Kouris and Christos-Savvas Bouganis (2018)	LSTM based approximation	Time complexity reduced, high accuracy	Retraining is needed for further optimization.	LSTM based FPGA design
[3] Echavarria, J., Schutz, K., Becher, A., Wildermann, S., &	Approximate computing case	Power consumption is reduced on FPGAs	Board level measurements are	AC with FPGA designs

Teich, J. (2018)	studies		recommended.	
[4] Bishwajeet Pandey and Manisha Pattanaik. (2013)	Clock Gating Aware approach	Clock power reduction	Reduction of leakage power is still desired.	FPGA based ALU design
[5] Eddie Hung, James J. Davis, Joshua M. Levine, Edward A. Stott, Peter Y. K. Cheung and George A. Constantinides (2016)	System identification approach.	Efficient mapping of tasks, reduces system wide power consumption	Different signal selection methods are to be explored	Power estimation in FPGA design
[6] D.I. Lazorenko and A.A. Chemeris. (2015)	Loop optimization	Behavioural description, low power consumption	-	Low power FPGA design
[7] Pierre-Emmanuel Gaillardon, Xifan Tang, Jury Sandrini, Maxime Thammasack, Somayyeh Rahimian Omam, Davide Sacchetto, Yusuf Leblebici and Giovanni De Micheli. (2015)	Monolithically integrated RRAMs	Reduction in area, delay and power consumption	-	Low power FPGA design
[8] Abhijeet Khandale and Dr. H R Bhagyalakshmi. (2015)	CLB with clock gating	Reduction in routing congestion and dynamic power, improve timing of design	-	Low power FPGA design
[9] Anonymous (2016)	PDPE problem solver	Power consumption is reduced	-	FPGA-mapped designs
[10] Xifan Tang, Pierre-Emmanuel Gaillardon and Giovanni De Micheli. (2014)	RRAM based FPGA	Reduction in area, delay and power consumption	-	RRAM-based FPGA
[11] Gaurav Verma, Manish Kumar and Vijay Khare. (2015)	Pow power techniques reviewed	Power management features	-	Digital system designs with FPGA
[12] Sunny Singh, Abhishek Jain, Amanpreet Kaur and Bishwajeet Pandey. (2014)	Green computations	Reduction in area and power consumption	-	UART design with FPGA
[13] Lei He. (2013)	FPGA circuits and methods, patented work	Reduction in delay and power.	-	FPGA circuits
[14] Marco Winzker. (2014)	Low power designs review	Power reduction and improving performance.	-	FPGA based digital systems
[15] Bahram Rashidi, Farshad Mirzaei, Bahman Rashidi and Majid Pourormazd. (2013)	FIR filter	Reduce dynamic power consumption.	-	FPGA designs with FIR filter
[16] Shatha AbuShanab, Marco Winzker and Rainer Brück (2015)	Knowledge based tutorial	Technical knowhow on energy efficiency	Enhancements in FPGA technologies	Low power FPGA designs
[17] Sarita Chouhan and Yogesh Kumar. (2012)	FIR filters	Low power consumption	-	FIR filters design for FPGA
[18] Deepa Singh, Bishwajeet	IO standard based	Low power	Further exploration on	RAM design based on

Pandey and Manisha Pattanaik. (2013)	RAM	consumption, reduction in delay	IO standards is desired	FPGA
[19] Haiyue Song, Chengwen Xu, Qiang Xu, Zhuoran Song, Naifeng Jing, Xiaoyao Liang and Li Jiang. (2018)	Neural AC	Energy efficiency	-	Approximators designed with NN Multi-class based.

As shown in Table 3, the techniques used for approximation and corresponding FPGA designs used for realizing different circuit based products like RAM, CPU, GPU etc. The summary also reveals that there are plenty of designs and also approximation approaches. It is essential to know the insights and then choose the ideal ones for better performance based on the design in hand. Nevertheless, there is some need for further research on defining more useful approximate computing technique for FPGA designs.

6. CONCLUSION AND FUTURE WORK

In this paper, we have covered various techniques pertaining to AC. It includes the potential opportunities of AC and its challenges. We also threw light on the LPGA designs using AC. It is understood from the literature that AC is essential in the contemporary computing world. There are many application areas that do not need exact computing. AC can leverage speed, performance besides reducing cost, effort and also energy consumption. ACTs are found to be useful in many real world applications where low-power FPGA designs. There is scope of using AC for large spectrum of application in the real world. It is understood that the existing applications developed with conventional languages may not be able to scale well with AC. There is need for identifying powerful platform for writing code in order to have maximum benefits of AC. There is still significant research desired on the low-power FPGA designs that exploit AC. When AC is used at saturation level in all possible real applications, there will be many benefits in terms of performance, resource optimization and saving time, cost and effort. Particularly usage of AC reduces power consumption significantly in embedded systems. ACTs are needed in several systems including dynamic voltage system, compression techniques and so on. Smooth integration of AC into commercial applications is essential to reap benefits of it. There is trade-off between resource constrainedness and performance requirements. This gap can be exploited well with purposefully designed application specific ACTs. From the literature we believed that AC is an attractive and useful research area that will have maximum imapact on its stakeholders across the globe. In the presence of CPU, GPU and FPGA designs, AC plays crucial role in leveraging their performance by eliminating precise computations in error-tolerant applications. It is our future endeavour to focus on designing a novel ACT and integrate it with low power FPGA to accelerate growth.

REFERENCES

[1] Sharad Sinha and Wei Zhang (2016). *Low-Power FPGA Design Using Memoization-Based Approximate Computing. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 24(8),* p2665*–2678.*

[2] Michalis Rizakis, Stylianos, Venieris, Alexandros Kouris and Christos-Savvas Bouganis (2018). *Approximate FPGA-Based LSTMs Under Computation Time Constraints. Lecture Notes in Computer Science,* p3*–15.*

[3] Echavarria, J., Schutz, K., Becher, A., Wildermann, S., & Teich, J. (2018). Can Approximate Computing Reduce Power Consumption on FPGAs? 2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS). P1-4.

[4] Bishwajeet Pandey and Manisha Pattanaik. (2013). Clock Gating Aware Low Power ALU Design and Implementation on FPGA. *International Journal of Future Computer and Communication*. 2 (5), p1-5.

[5] Eddie Hung, James J. Davis, Joshua M. Levine, Edward A. Stott, Peter Y. K. Cheung and George A. Constantinides (2016). *KAPow: A System Identification Approach to Online Per-Module Power Estimation in FPGA Designs. 2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM).* P1-8.

[6] D.I. Lazorenko and A.A. Chemeris. (2015). Program Optimization for Low Power FPGA Design. *Department of Mathematical modelling and simulation*, p1-4.

[7] Pierre-Emmanuel Gaillardon, Xifan Tang, Jury Sandrini, Maxime Thammasack, Somayyeh Rahimian Omam, Davide Sacchetto, Yusuf Leblebici and Giovanni De Micheli. (2015). An Ultra-Low-Power FPGA Based on Monolithically Integrated RRAMs, p1-6.

[8] Abhijeet Khandale and Dr. H R Bhagyalakshmi. (2015). Low Power FPGA Architecture. *International Journal of Science and Research*. 4 (6), p1-4.

[9] (2016). Peak Dynamic Power Estimation of FPGA-mapped Digital Designs, p1-6.

[10] Xifan Tang, Pierre-Emmanuel Gaillardon and Giovanni De Micheli. (2014). A High-Performance Low-Power Near-Vt RRAM-based FPGA, p1-8.

[11] Gaurav Verma, Manish Kumar and Vijay Khare. (2015). Low Power Techniques for Digital System Design. *Indian Journal of Science and Technology*. 8 (17), p1-7.

[12] Sunny Singh, Abhishek Jain, Amanpreet Kaur and Bishwajeet Pandey. (2014). Thermal Aware Low Power Universal Asynchronous Receiver Transmitter Design on FPGA, p1-4.

[13] Lei He. (2013). LOW-POWER FPGA CIRCUITS AND

METHODS. *United States Patent*, p1-82.

[14] Marco Winzker. (2014). Addressing Low-Power Electronics in a Digital System and FPGA Design Course. *iJEP*. 4 (4), p1-6.

[15] Bahram Rashidi, Farshad Mirzaei, Bahman Rashidi and Majid Pourormazd. (2013). Low Power FPGA Implementation of Digital FIR Filter Based on Low Power Multiplexer Base Shift/Add Multiplier. *International Journal of Computer Theory and Engineering*. 5 (2), p1-5.

[16] Shatha AbuShanab, Marco Winzker and Rainer Brück (2015). *Teaching low-power design with an FPGA-based hands-on and remote lab. 2015 IEEE Global Engineering Education Conference (EDUCON).* P1-9.

[17] Sarita Chouhan and Yogesh Kumar. (2012). LOW POWER DESIGNING OF FIR FILTERS. *International Journal of Advanced Technology & Engineering Research*. 2 (2), p1-9.

[18] Deepa Singh, Bishwajeet Pandey and Manisha Pattanaik. (2013). IO Standard Based Low Power Design of RAM and Implementation on FPGA. *Journal of Automation and Control Engineering*. 1 (4), p1-5.

[19] Haiyue Song, Chengwen Xu, Qiang Xu, Zhuoran Song, Naifeng Jing, Xiaoyao Liang and Li Jiang. (2018). Invocation-driven Neural Approximate Computing with a Multiclass-Classifier and Multiple Approximators. *ACM*, p1-9.

[20] ThomasYYeh, PetrosFaloutsos, MilosErcegovac, SanjayJPatel, andGlennReinman. 2007. The art of deception: Adaptive precision reduction for area efficient physics acceleration. In International Symposium on Microarchitecture.394–406.

[21] Qian Zhang, Ting Wang, Ye Tian, Feng Yuan, and Qiang Xu. 2015. ApproxANN: an approximatecomputingframeworkforartiftcialneuralnetwork.InDesign,Automa- tion&Test in Europe.701–706.

[22] Stelios Sidiroglou, SasaMisailovic, Henry Hoffmann, and Martin Rinard. 2011. Man- aging performance vs. accuracy trade-offs with loop perforation. In ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineer- ing.124–134.

[23] Joshua San Miguel, Mario Badr, and Enright Natalie Jerger. 2014. Load Value Approximation. MICRO (2014).

[24] Amir Yazdanbakhsh, Gennady Pekhimenko, Bradley Thwaites, HadiEsmaeilzadeh, Taesoo Kim, OnurMutlu, and Todd C Mowry. 2015b. RFVP: Rollback-Free Value Prediction with Safe-to-Approximate Loads. Technical Report. Georgia Institute of Technology.

[25] Mark Sutherland, Joshua San Miguel, and Natalie Enright Jerger. 2015. Texture Cache Approximation on GPUs. Workshop on Approximate Computing Across the Stack (2015).

[26] Azar Rahimi, Luca Benini, and Rajesh K Gupta. 2013. Spatial memoization: Concur- rent instruction reuse to correct timing errors in SIMD architectures. IEEE Trans- actions on Circuits and Systems II: Express Briefs 60, 12 (2013), 847–851.

[27] Georgios Keramidas, ChrysaKokkala, and IakovosStamoulis. 2015. Clumsy Value Cache:AnApproximateMemoizationTechniqueforMobileGPUFragmentShaders. Workshop On Approximate Computing (WAPCO) (2015).

[28] MehrzadSamadi, DavoudAnousheJamshidi, Janghaeng Lee, and Scott Mahlke. 2014.Paraprox: Pattern-basedapproximationfordataparallelapplications.InACM SIGARCH Computer Architecture News, Vol. 42.35–50.

[29] ´In˜igoGoiri, RicardoBianchini, SantoshNagarakatte, andThuDNguyen. 2015.ApproxHadoop: BringingApproximationstoMapReduceFrameworks.InInternationalConferenceonArchitecturalSupportforProgrammingLanguagesandOperating ACM Comput. Surv., Vol. a, No. b, Article 1, Publicati. date:2015.

[30] MehrzadSamadi, Janghaeng Lee, D AnousheJamshidi, Amir Hormati, and Scott Mahlke. 2013. SAGE: Self-tuning approximation for graphics engines. In International Symposium on Microarchitecture. 13–24.

[31] WoongkiBaek and Trishul M Chilimbi. 2010. Green: a framework for supporting energy-conscious programming using controlled approximation. In ACM Sigplan Notices, Vol. 45. 198–209.

[32] Andrew B Kahn and Seokhyeong Kang. 2012. Accuracy-configurable adder for approximate arithmetic designs. In Design Automation Conference. 820–825.

[33] ParagKulkarni, PuneetGupta, andMilosErcegovac. 2011.Tradingaccuracyforpower withanunderdesignedmultiplierarchitecture. InInternationalConferenceonVLSI Design (VLSI Design).346–351.

[34] SwagathVenkataramani, Amit Sabne, VivekKozhikkottu, Kaushik Roy, and AnandRaghunathan. 2012. SALSA: systematic logic synthesis of approximate circuits. In Design Automation Conference. 796–801.

[35] ShrikanthGanapathy, Georgios Karakonstantis, Adam Shmuel Teman, and An- dreas Peter Burg. 2015. Mitigating the Impact of Faults in Unreliable Memories for Error-Resilient Applications. In Design Automation Conference.

[36] YavuzYetim, Margaret Martonosi, and Sharad Malik. 2013. Extracting usefulcompu- tationfromerror-proneprocessorsforstreamingapplications.InDesign,Automation &Test in Europe Conference & Exhibition (DATE).202–207.

[37] Sparsh Mittal and Jeffrey Vetter. 2015. A Survey of Techniques for Modeling and Improving Reliability of Computing Systems. IEEE Transactions on Parallel and Distributed Systems (TPDS) (2015).

[38] Adrian Sampson, Werner Dietl, Emily Fortuna, DanushenGnanapragasam, Luis Ceze, andDanGrossman. 2011.EnerJ: Approximatedatatypesforsafeandgeneral low-power computation. In ACM SIGPLAN Notices, Vol. 46.164–174.

[39] Vinay K Chippa, DebabrataMohapatra, Kaushik Roy, Srimat T Chakradhar, and AnandRaghunathan. 2014. Scalable effort hardware design. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 22, 9 (2014),2004–2016.

[40] Abbas Rahimi, AmiraliGhofrani, Kwang-Ting Cheng, Luca Benini, and Rajesh K Gupta. 2015. Approximate associative memristive memory for energy-efficient GPUs. In Design, Automation & Test in Europe. 1497–1502.

[41] BeaynaGrigorian, NazaninFarahpour, andGlennReinman. 2015.BRAINIAC: Bring-ingreliableaccuracyintoneurally-implementedapproximatecomputing. In International Symposium on High Performance Computer Architecture (HPCA).615–626.

[42] John Satori and Ravindra Kumar. 2013. Branch and data herding: Reducing control and memory divergence for error-tolerant GPU applications. IEEE Transactions on Multimedia 15, 2 (2013), 279–290.

[43] HadiEsmaeilzadeh, AdrianSampson, LuisCeze, andDougBurger. 2012b.Neuralac- celebration for general-purpose approximate programs. In IEEE/ACM International Symposium on Microarchitecture.449–460.

[44] Lawrence McAfee and KunleOlukotun. 2015. EMEURO: a framework for generating multi-purpose accelerators via deep learning. In International Symposium on Code Generation and Optimization. 125–135.

[45] Jason Ansel, Yee Lok Wong, Cy Chan, Marek Olszewski, Alan Edelman, and SamanAmarasinghe. 2011. Language and compiler support for auto-tuning variable- accuracy algorithms. In International Symposium on Code Generation and Optimization. 85–96.

[46] SchuylerEldridge, FlorianRaudies, DavidZou, andAjayJoshi.2014. Neural network- based accelerators for transcendental function approximation. In Great Lakes Symposium on VLSI.169–174.

[47] B. Li, P. Gu, Y. Shan, Y. Wang, Y. Chen, and H. Yang. 2015. RRAM-based Analog Ap- proximate Computing. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2015).

[48] YeTian, QianZhang, TingWang, FengYuan, andQiangXu. 2015.ApproxMA: Approximate Memory Access for Dynamic Precision Scaling. In ACM Great Lakes Symposium on VLSI.337–342.

[49] Kyungsang Cho, YongjunLee, Young H Oh, Gyoo-cheol Hwang, and Jae W Lee.2014. eDRAM-based tiered-reliability memory with applications to low-power framebuffers. In International symposium on Low power electronics and design. 333–338.

[50] ThierryMoreau, MarkWyse, JacobNelson, AdrianSampson, HadiEsmaeilzadeh, LuisCeze, andMarkOskin. 2015.SNNAP: ApproximatecomputingonprogrammableSoCsvianeurala cceleration.InInternationalSymposiumonHighPerforman ce Computer Architecture (HPCA). 603–614.

[51] Antonio Roldao Lopes, Amir Shahzad, George Constantinides, Eric C Kerrigan, and others. 2009. More flops or more precision? accuracy parameterizable linear equationsolversformodelpredictivecontrol.InSymposium onFieldProgrammableCus- tom Computing Machines (FCCM).209–216.