Approximate Computing for Low-power: Survey and Challenges

Prof. Dr. Akash Kumar
Chair for Processor Design
(Ack: my past and current students/PostDocs)
(Some slides adapted from Anand)
Outline

- Why?
  - Motivation for Approximate Computing

- What?
  - Approximate computing: Design philosophy and approach

- How?
  - Technologies for Approximate Computing
SEVENTH HEAVEN OF APPLICATIONS

Incrased compute efficiency demands

Efficiency gap

Diminishing benefits from technology scaling

Scaling limits?

HELL OF NANOSCALE PHYSICS

Need new sources of efficiency to bridge the gap!

Ack: Hugo De Man, DATE 2002

Interconnects, closure, SI

Dennard scaling ends

Leakage, Variations

Thermal

Reliability
Efficiency Gap In Computing

- Significant gap between future requirements and projected capabilities of computing platforms.

Today’s mobile platforms:
- Snapdragon 805
- Tegra K1, A8X
- 10-20 GFLOPS/W

Scaling (~7nm), many-core, heterogeneity, near-threshold computing

~100 GOPS - TOPS/W

Real-time object recognition & tracking
Immersive VR & AR
3D imaging/sensing

Gap (need for new efficiencies)

“How do we advance computing systems without (significant) technology progress?” DARPA/ISAT workshop, March 2012
The Computational Efficiency Gap

IBM Watson playing Jeopardy, 2011
Humans Approximate

**Task:** Division

\[ \frac{923}{21} > 1.75? \]

\[ \frac{923}{21} > 45? \]

21) 923 (43 ---

\[ \begin{array}{c}
84 \\
83 \\
63
\end{array} \]

\[ \frac{923}{21} = --.--? \]

**Accuracy**

\[ \sim 1 \text{ Petaflop/W} \]

Application context dictates required accuracy of results

Effort expended increases with required accuracy
But Computers DO NOT

\[
\frac{923}{21} > 45
\]

\[
\frac{923}{21} > 1.75
\]

- **Overkill** (for many applications)
- Leads to **inefficiency**
- Can computers be more efficient by producing “just good enough” results?
Intrinsic Application Resilience: Sources

- Intrinsic application resilience: Ability to produce acceptable outputs despite underlying computations being performed in an approximate manner.

Diagram:
- Redundant Input Data
- Perceptual Limitations
- Statistical Probabilistic Computations
- Self-Healing

- ‘Noisy’ Real World Inputs
- Intrinsic Application Resilience

Process:
- Compute distances & assign points to clusters
- Update cluster means
- Repeat until convergence

© Akash Kumar
Intrinsic Resilience In RMS Applications

Recognition, Mining, Synthesis Application Suite

<table>
<thead>
<tr>
<th>Application</th>
<th>% Runtime in resilient computations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document search</td>
<td></td>
</tr>
<tr>
<td>Image search</td>
<td></td>
</tr>
<tr>
<td>Digit recognition</td>
<td></td>
</tr>
<tr>
<td>Digit model generation</td>
<td></td>
</tr>
<tr>
<td>Eye detection</td>
<td></td>
</tr>
<tr>
<td>Eye model generation</td>
<td></td>
</tr>
<tr>
<td>Image segmentation</td>
<td></td>
</tr>
<tr>
<td>Census data modeling</td>
<td></td>
</tr>
<tr>
<td>Census data classification</td>
<td></td>
</tr>
<tr>
<td>Health information analysis</td>
<td></td>
</tr>
<tr>
<td>Character recognition</td>
<td></td>
</tr>
<tr>
<td>Online data clustering</td>
<td></td>
</tr>
</tbody>
</table>

Applications have a mix of resilient and sensitive computations

83% of runtime spent in computations that can be approximated


© Akash Kumar
Its an Approximate World … At the Top

- No golden answer (multiple answers are equally acceptable)
  - Web search, recommendation systems

- Even the best algorithm cannot produce correct results all the time
  - Most recognition / machine learning problems

- Too expensive to produce fully correct or optimal results
  - Heuristic and probabilistic algorithms, relaxed consistency models, …
Outline

- Why?
  - Motivation for Approximate Computing
- What?
  - Approximate computing: Design philosophy and approach
- How?
  - Technologies for Approximate Computing
Approximate Computing: Design Philosophy

- Computing platforms that can modulate the effort expended towards quality of results
  - Higher effort → Higher quality but lower efficiency
- How do we get the best Q vs. E tradeoff?
  - Disproportionate benefit

© Akash Kumar
Approximate Computing Throughout the Stack

- No golden answer
- Perfect/correct answers not always possible
- Too expensive to produce perfect/correct answers

Programming Languages, Compilers, Runtimes

<table>
<thead>
<tr>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic</td>
</tr>
<tr>
<td>Circuits</td>
</tr>
</tbody>
</table>
Outline

- Why?
  - Motivation for Approximate Computing
- What?
  - Approximate computing: Design philosophy and approach
- How?
  - Technologies for Approximate Computing
Approximate Computing Approach

- Approximations at various layers of abstraction
- Need to ensure quality specifications are met

Application

Approximate Software
Approximate Architecture
Approximate Circuits
Approximate Layout

Quality Specifications

Approximation mechanisms

Quality Met?

Relaxed Equivalence

Approximate Implementation

© Akash Kumar
Some Early Efforts in Approx. Computing*

- Approximate signal processing (Chandrakasan et. al, 1997)
- Voltage overscaling (Shanbhag et. al, ISLPED 1999)
- Probabilistic CMOS (Palem et. al, 2003)
- Manufacturing yield enhancement (Breuer et. al, 2004-)
- Energy-efficient, variation-tolerant approximate hardware (Roy et. al, 2006-)
- Probabilistic Arithmetic / Biased voltage overscaling (Palem et. al, CASES 2006-)
- Parallel runtime framework with computation skipping, dependency relaxation (Raghunathan et. al, IPDPS 2009; IPDPS 2010)
- Error-resilient / stochastic processors (Mitra et. al, 2010; Kumar et. al, 2010)
- Cross-layer, scalable-effort approximate HW design (Chippa et. al, 2010)
- Programming support for approximate computing (Chilimbi el. al, 2010; Misailovic et. al, 2010; Sampson et. al, 2011)
- ...
- ...
- http://timor.github.io/refgraph/ Dancing authors. 😊

* Not an exhaustive list!
Approximate Software

Largely based on:
Approximate Software

- Techniques can be applied at
  - Compile-time OR
  - Run-time

- Frameworks that exploit multiple layers
  - Precision specification -> identify and specify what to approximate
  - Precision reduction implementation -> actually perform and control approximation

- Application at different layers (Better throughout!)
  - Language
  - Algorithm
  - Compiler
Compile-time vs Run-time

- Compile-time
  - Use information available before execution
  - Possibly lower execution overhead
  - Need analysis on accuracy bounds

- Run-time
  - More lenient towards incomplete accuracy analysis
  - Generally larger overhead

- Combination of the two
  - Runtime reconfigurable approximate systems
Precision Specification

1. Code annotation
2. Built-in Language support
3. Explicit algorithm techniques
4. Output quality monitoring
Precision Reduction Implementation

1. **Loop perforation** — identify loops where only a subset of iterations can be performed while maintaining acceptable accuracy
2. **Precision Scaling** — right-shift data or truncate
3. **Memoization** — use for functions with similar input/output pairs
4. **Task Skipping** — perform subset of tasks
5. **Program Selection** — select from multiple versions
6. **Approximate Storage** — allow data to degrade
7. **Neural Network Substitution**
## Overall Frameworks

1. **Green**  

2. **iACT**  
   Mishra, A. K.; Barik, R. & Paul, S. iACT: A software-hardware framework for understanding the scope of approximate computing, WACAS, 2014

3. **GRATER**  
Approximate Computing Approach

- Approximations at various layers of abstraction
- Need to ensure quality specifications are met

- Approximate Software
- Approximate Architecture
- Approximate Circuits
- Approximate Layout

Approximation mechanisms

Quality Specifications

Quality Met?

Relaxed Equivalence

Approximate Implementation

© Akash Kumar
Approximate Architecture
Approximate Architecture

Algorithm-specific accelerators

Domain-specific accelerators – image, video ...

Programmable accelerators (GPGPUs, MIC) / Vector processors

General purpose processors/ Multicores

Application specific designs

- ANT – Hedge et. al. – ISLPED 1999
- Significance driven computing – Mohapatra et.al. – ISLPED 2009
- Scalable effort hardware – Chippa et. al. – DAC 2010, DAC 2011

Cores of different reliabilities

- ERSA – Leem et. al. – DATE 2010
- Stochastic processor – Narayanan et. al. – DATE 2010

Accurate and approximate instructions

- Truffle – Esmaeilzadeh et. al.– ASPLOS 2012
- EnerJ – Sampson et. al. – PLDI 2011
Approximate Architecture

Pros:
😊 Large energy benefits
😊 Broader applicability

Challenges:
😢 Limited applicability
😢 Inherently limited energy benefit – Dominated by control front-ends that cannot be approximated
😢 Allow arbitrary errors in hardware – limits the fraction of computations that can be approximated

Algorithm-specific accelerators – Domain-specific accelerators – Programmable accelerators (GPGPUs, MIC) / Vector processors – General purpose processors/ Multicores
Approximate Architecture

Algorithm-specific accelerators

Domain-specific accelerators – image, video ...

Programmable accelerators (GPGPUs, MIC) / Vector processors

General purpose processors / Multicores

Opportunity:

😊 Wide range of applications – fine grained parallelism

😊 SIMD: Control overheads amortized over many execution units

😊 Need quality guarantees from HW

Designing Inexact Systems Efficiently using Elimination Heuristics

DATE 2015
Introduction

- Diminishing transistor sizes \(\Rightarrow\) increase in power density and errors
- Inexact computing can trade accuracy for significant gain in area/power
- Previous works:
  - Decreasing the voltage of operation significantly to reduce the power consumption albeit at the cost of reliable circuit operation [Kim, ACM JETC 2014][George, CASES 2006]
  - Reducing the number of transistors in order to save energy [Lingamneni, ACM TECS 2013]
  - Removing parts of circuit that have a lower probability of being active – probabilistic pruning [Lingamneni, DATE 2011]
- However, designing such inexact systems is expensive
- Exponential growth in search space
Current inexact systems lack

- Ability to estimate quickly the overall inexactness of a system
- Identifying the best set of inexact components to use from a given set of components

Having an overall design flow to construct such inexact systems with tunable parameters is the scope of this research
Contributions

- Algorithm to quickly estimate the inexactness of the larger components
- A design-flow that uses the above algorithm to design the entire system under the area and power constraints
- A heuristic to reduce the design-space exploration time by eliminating the non-distinct points.
- Results of the design-flow applied to an ECG application of QRS detection.
Design Flow – Inexact Components

- Inexact components considered
  - Adders
  - Multipliers
- Probabilistic pruning
Accuracy tradeoff for adder/multiplier

- As more nodes pruned, gains in area, delay and energy increase
- An order of magnitude improvement in energy-delay-area for 10% error
Design Flow – Optimization Problem

- **Given**
  - Inexact versions for each adder and multiplier

- **Objective**
  - Choose the inexact versions for all components such that we get the most significant gains in power/delay/area with the least tradeoff in accuracy
Design Flow – Optimization Problem

- Exhaustive search – exponential growth
  - For a system with 2 adders and 2 multipliers with 5 inexact versions of each – design search space is 625 points
  - For a system with 5 adders and 5 multipliers with 5 inexact versions of each – design search space is 9.7 million points
  - 37 years for simulating all options!
Design Flow — Heuristic Search

- Reduce design space exploration time
  - Order of inexact components does not matter (??!!)
  - Only designs which would result in distinct Pareto points considered
  - Design space compared to exhaustive search
    - 2 adders, 2 multipliers = 64 points (vs 625)
    - 5 adders, 5 multipliers = 16,384 points (vs 9.7mln) (Still 22 days!)
  - Orders of magnitude smaller than exhaustive search
Design Flow – Heuristic Search
Results – Accuracy of Estimation

- Estimation and simulation results in similar trend
- Estimation considers worst case scenario
QRS detection, one of the most important features of ECG considered.

Figure below shows steps required to process ECG signal before QRS can be detected.
Case Study – Exact vs Inexact design

- None of the inexact designs missed a QRS signal
- Able to achieve good output with up to 15% power

<table>
<thead>
<tr>
<th>Design</th>
<th>Power savings (%)</th>
<th>Area savings (%)</th>
<th>Relative Error (%)</th>
<th>Number of QRS signals missed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact design</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pareto optimal 1</td>
<td>12.6</td>
<td>4.5</td>
<td>0.18</td>
<td>0</td>
</tr>
<tr>
<td>Pareto optimal 2</td>
<td>14.6</td>
<td>4.9</td>
<td>0.96</td>
<td>0</td>
</tr>
<tr>
<td>Pareto optimal 3</td>
<td>12</td>
<td>5.3</td>
<td>1.92</td>
<td>0</td>
</tr>
<tr>
<td>Pareto optimal 4</td>
<td>12.8</td>
<td>5.9</td>
<td>3.03</td>
<td>0</td>
</tr>
</tbody>
</table>
Limitations and future works

- The error in estimation increases with the number of components although the trend remains the same – have better heuristics for estimation

- Heuristic for automated Pareto point selection rather than human input

- Designing co-efficient specific components for filters
Approximate Computing Approach

- **Approximations** at various layers of abstraction
- Need to ensure *quality* specifications are *met*

Diagram showing:
- Application
- Approximate Software
- Approximate Architecture
- Approximate Circuits
- Approximate Layout
- Quality Specifications
- Approximation mechanisms
- Quality Met?
- Relaxed Equivalence
- Approximate Implementation

© Akash Kumar
Approximate Circuits
Approximate Logic Synthesis: Re-visiting a Classical Problem

- **Inputs**
  - RTL specification
  - Constraint on the output quality

- **Output**
  - Functionally approximate circuit that adheres to imposed quality bounds

Error magnitude:
\[ Q = \left( \frac{\left| PO_{orig} - PO_{approx} \right|}{\leq K} \right) \]

Error rate:
\[
\frac{Total \ Inputs \ \& \ O_{orig} \neq O_{approx}}{Total \ Number \ of \ Inputs}
\]

Experimental Methodology

- **Q- Functions**

  Maximum Error: \( Q = \left( \left| PO_{\text{orig}} - PO_{\text{approx}} \right| \leq K \right) \) ? 1:0

  Relative Error: \( Q = \left( 1 - K \leq \frac{PO_{\text{approx}}}{PO_{\text{orig}}} \leq 1 + K \right) \) ? 1:0

- **Benchmark Circuits**

  - SALSA implemented using two logic synthesis tools – SIS and Synopsys Design Compiler

  - Circuits mapped to IBM 45nm technology library for iso-delay and evaluated for area, power

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Bit Width</th>
<th>I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCA</td>
<td>Ripple Carry Adder</td>
<td>32</td>
<td>64/33</td>
</tr>
<tr>
<td>KSA</td>
<td>Kogge Stone Adder</td>
<td>32</td>
<td>64/33</td>
</tr>
<tr>
<td>CLA</td>
<td>Carry Look ahead Adder</td>
<td>32</td>
<td>64/33</td>
</tr>
<tr>
<td>MUL</td>
<td>Array Multiplier</td>
<td>8</td>
<td>16/16</td>
</tr>
<tr>
<td>WTM</td>
<td>Wallace Tree Multiplier</td>
<td>8</td>
<td>16/16</td>
</tr>
<tr>
<td>MAC</td>
<td>Multiply and Accumulate</td>
<td>8</td>
<td>48/33</td>
</tr>
<tr>
<td>SAD</td>
<td>Sum of Absolute Difference</td>
<td>8</td>
<td>48/33</td>
</tr>
<tr>
<td>EU_DIST</td>
<td>Euclidean Distance</td>
<td>8</td>
<td>32/16</td>
</tr>
<tr>
<td>BUT</td>
<td>Butterfly Structure (FFT)</td>
<td>8</td>
<td>16/16</td>
</tr>
<tr>
<td>FIR</td>
<td>Finite Impulse Response Filter</td>
<td>8</td>
<td>32/16</td>
</tr>
<tr>
<td>IIR</td>
<td>Infinite Impulse Response Filter</td>
<td>8</td>
<td>56/16</td>
</tr>
<tr>
<td>DCT</td>
<td>Discrete Cosine Transform</td>
<td>8</td>
<td>64/72</td>
</tr>
</tbody>
</table>
Approximate Circuits: Other Work

- Quality-configurable circuits
  - Substitute-and-simplify
    [Venkataramani et al., DATE 2013]

- Modeling and verification of approximate circuits
  - MACACO [Rangharajan et al., ICCAD 2011]

- Approximate computing with stochastic logic
  - StoRM [Chippa et al., ISLPED 2014]
A Flexible Inexact TMR Technique for SRAM-based FPGAs

DATE 2016
Approximate TMR

- **Inexact computing** (an alternative to the TMR overhead):
  - F and H are inexact modules (approximations) of G
  - **Main objective**: reduce the size of the inexact modules and maximize fault masking
Problem Formulation

- Given a function, $G$
  - Obtain functions $F$ and $H$, such that the area overhead is minimized while maximizing the fault masking
Huge Design Space

- 78,400 design points for a small MCNC benchmark, alu4 (output 3)
- Examining all possibilities of F and H, neither practical, nor scalable
F and H are independently explored at fixed area overhead intervals.

Overall area-masking Pareto points are stored.

Only $A=0.75-1.25$ shown for clarity.
Conclusions

- Approximate Triple Module Redundancy approach, especially targeted for FPGAs
- The approach provides a trade-off between the area overhead and the masking factor
- The heuristic saves 84% of time while providing similar results
Architectural-Space Exploration of Approximate Multipliers

ICCAD 2016
Design Space of Existing Approximate Multipliers

What is Required?

Generation and exploration of multiple design alternatives of approximate multiplier

=> Enable trade-offs for area, power, and accuracy

A: Accurate Design
B: Bhardwaj’14
C1: Kulkurni’11 + Non-Config.
C2: Kulkurni’11 + Configurable

© Akash Kumar
Our Methodology

Step 1: Building Library of Elementary Approximate Modules

- Approximate Adder Modules
  - VHDL Files
  - C Behavioral Model
- Approximate Multiplier Modules
  - VHDL Files
  - C Behavioral Model

Step 2: Characterization and Early Design Space Reduction

- Synthesize (Synopsys Design Compiler)
- Validation (ModelSim)
- Power Estimation (PrimeTime)
- Area, Power, Quality Statistics
- Input Data
- Behavioral Simulation
- Design Space Reduction
- Application Program

Step 3: Building Larger Approximate Modules

Step 4: Selection Methodology

© Akash Kumar
All Tested Approximate

(a) AccuMul

(b) ApproxMul₁

(c) ApproxMul₂

(d) ApproxMul₃

(e) ApproxMul₄

(f) ApproxMul₅

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>01</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
</tr>
<tr>
<td>10</td>
<td>0000</td>
<td>0100</td>
<td>0110</td>
<td>0110</td>
</tr>
<tr>
<td>11</td>
<td>0000</td>
<td>0110</td>
<td>0110</td>
<td>1001</td>
</tr>
</tbody>
</table>

© Akash Kumar
Overview of Larger Multiplier Structure

Constructing a Multi-Bit Approximate Multiplier

I. Type of Elementary 2x2 Multiplier [Acc. ~ Appx. types]
II. Type of Elementary 1-Bit Adder [Acc. ~ Appx. types]
III. Number of Bits to be Approximated
Design Selection Methodology

- Approximate Multiplication

**Depth-First Search Algorithm**

to find near optimal configuration based on:

i. WAP Values

ii. An Error Model for Quality Evaluation
Design Selection Methodology

- Evaluate WAP for 1-bit Full Adders and 2x2 Multipliers;
- List WAP values in ascending order for multipliers and Adders
- List of LSBs to be approximated (0,2,4…etc.)
- For each multiplier alternative, all adder types are then tested for different LSB-combinations
- The first configuration that satisfies the quality criteria is selected as the solution. QCurrent is checked with Qconstraint.

Algorithm 1: Design Selection using DFS (Depth-First Search)

**INPUT:** (i) QMetric: the quality criteria, for example, the highest error magnitude; (ii) QConstraint: the condition to satisfy for the given metric (for example, QMetric < 5); (iii) $W_A$: area weight; (iv) $W_P$: power weight.

**OUTPUT:** (i) the selected configuration of 2-bit multipliers, adders, and LSBs; (ii) the associated quality metric

```
BEGIN
1. $WAP_{Add} = \text{EvalWAP for Adders}(W_A, W_P)$; //Evaluate WAP for all considered 1-bit FAs
2. AdderList = AscendingSort($WAP_{Add}$);
3. $WAP_{Mult} = \text{EvalWAP for Multipliers}(W_A, W_P)$; //Evaluate WAP for all considered 2x2 multipliers
4. MultiplierList = AscendingSort($WAP_{Mult}$);
5. LSBsValues = DescendingSort(LSBValues);
6. while (traversing multiplierTree depthFirst) do
7.   for all mult ∈ MultiplierList do
8.     for all add ∈ AdderList do
9.       for all lsb ∈ LSBValues do
10.      QCurrent = ApplyErrorModel(QMetric, mult, add, lsb);
11.      if(QCurrent meets QConstraint) then
12.       return current configuration
13.      endif
14.   end for
15. end for
16. end while
END
```
DSE Results for 8x8 Multipliers

- $3 \times 10^{12}$ Possible Different Configurations
- 19 Selected Design-Points Filtered to:
  - A, B3 and B6, C1 and C3, D6, D7, and D8
Conclusions

- Approximate Computing provides potential for significant power/area reductions and performance improvement

- Challenges:
  - Improve power/energy/area and performance efficiency with low quality degradation
  - Enabling Quality vs. Efficiency tradeoffs [Design ~ Run-Time]

- Our Approach: Cross-Layer Approximate Computing
  - Developed an approximate 2x2 multiplier and its accuracy configurable version, Implemented state-of-the-art designs
  - Building modular and configurable approximate modules
  - Exploration of the design-space of large-sized approximate multipliers

- Open-Source Libraries of Approximate Computing Modules
DeMAS: An Efficient Design Methodology for Building Approximate Adders for FPGA-Based Systems

DATE 2018
Contributions

- 8 different LUTs-based approximate adder designs
- A generic design methodology to implement approximate adders for any FPGA-based system
- Open-source library (RTL and behavioral) of approximate adders
Preliminaries for the Implementation

- Slice structure of the configurable logic block (CLB) of Xilinx 7-series FPGAs

For INIT value: 0000000000000002(hex)
Design Technique for Approximate Adders

- Utilizes the carry chain truncation technique.
- Analysis of the truth tables of 1-and 2-bit full adders,
  - Simplify and reduce them to occupy lesser number of lookup tables.

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Accurate</th>
<th>Truncated</th>
<th>Optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$S_k$</td>
<td>$C_{out}$</td>
<td>$S_k$</td>
</tr>
<tr>
<td>$A_k$</td>
<td>$B_k$</td>
<td>$C_{in}$</td>
<td>$S_k$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Proposed Approximate Adders

- Approximate Adder-1: A single LUT2 primitive to implement a half-adder circuit
- Approximate Adder-2: LUT3 primitive-based approximate FA circuit.
  - The carry computation circuit is eliminated
  - Sum-bit is modified to reduce the error magnitude
  - $C_{out}$ is equated to the input $A_0$

![Adder-1](image1)

![Adder-2](image2)
Proposed Approximate Adders

- Approximate Adder-3: A 2-bit adder circuit using LUT2 and LUT4 primitives
  - The carry is neither utilized nor computed.
- Approximate Adder-4: A 2-bit adder design implemented using LUT2 and LUT5 primitives
- Approximate Adder-5: A 2-bit approximate adder circuit
Proposed Approximate Adders

- Approximate Adder-6: The computation of both $S_k$ and $S_{k+1}$ considers the $C_{in}$
- Approximate Adder-7: Compute $S_{k+1}$ and equated to $S_k$
  - $C_{out}$ equated to $A_1$
- Approximate Adder-8: Compute both $S_{k+1}$ and $S_k$ with $S_k = S_{k+1}$
Area-Optimized Low-Latency Approximate Multipliers for FPGA-based Hardware Accelerators

DAC 2018
Motivational Case Study

- Performance Comparison of two ASIC-based approximate multipliers, "W" [1] and "K" [2], with their FPGA-based implementations.
- Using Xilinx Vivado 17.1 tool for the 7VX330T device of Virtex-7 family

Research Challenge: Define LUTs-based approximations for FPGA-based systems.

Contributions

- LUTs-based generation of accurate/approximate partial products.
- A novel approximate $4 \times 2$ multiplier as an elementary module.
- For complete utilization of 6-input LUTs.
- Utilizing FPGA-specific optimization to reduce the number of output errors and obtain an approximate $4 \times 4$ multiplier.
- Usage of accurate/approximate addition for implementing higher order multipliers.
Approximate 4×2 Multiplier

- Implementation of logic equations for the product bits of a 4×2 multiplier

  Multiplicand: $A_3A_2A_1A_0$  
  Multiplier: $B_1B_0$

Final output accuracy: 75% with maximum error magnitude of “1” for all input combinations
### Approximate 4×4 Multiplier

Approximate multiplication products along with some optimizations. Total LUTs required for 4×4 multiplier: 14.

Two LUTs may not be used by any other operation and wasted. Total LUTs: 16 LUTs.

Average relative error: 0.049

Error probability: 0.375

One LUT for accurately computing \( P_3 \), \( P_4 \) and \( P_5 \) using generate and propagate signals.

P_6 and P_7 are generated implicitly. Error is limited to \( P_3 \) only.

Fixed Error Magnitude: 8

<table>
<thead>
<tr>
<th>PP&lt;5&gt;</th>
<th>PP&lt;4&gt;</th>
<th>PP&lt;3&gt;</th>
<th>PP&lt;2&gt;</th>
<th>PP&lt;1&gt;</th>
<th>PP&lt;0&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>PP&lt;5&gt;</td>
<td>PP&lt;4&gt;</td>
<td>PP&lt;3&gt;</td>
<td>PP&lt;2&gt;</td>
<td>PP&lt;1&gt;</td>
<td>PP&lt;0&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P_7</th>
<th>P_6</th>
<th>P_5</th>
<th>P_4</th>
<th>P_3</th>
<th>P_2</th>
<th>P_1</th>
<th>P_0</th>
</tr>
</thead>
</table>

© Akash Kumar
Higher Order Multipliers

Multiplier: A \( (A_{2M-1}A_{2M-2} \ldots A_1A_0) \)

Multiplicand: B \( (B_{2M-1}B_{2M-2} \ldots B_1B_0) \)

\( A_L \) & \( B_L \): M LSBs of A & B

\( A_H \) & \( B_H \): M MSBs of A & B

Accurate/approximate addition can be used for adding sub-multipliers together.
8×8 Approximate Multiplier Ca

**Ternary Adder**

![Diagram of 8×8 Approximate Multiplier Ca with Ternary Adder](image)
8×8 Approximate Multiplier Cc

- Utilizes highly-inaccurate addition of sub-multipliers.

**PP<3><7> – PP<3><4>** are not added in Cc implementation of higher order multipliers. **PP<0><3> – PP<0><0>** are not added in any implementation (accurate/approximate) of higher order multipliers.

No carry propagation between product terms
Conclusion

- Approximate $4 \times 2$ and $4 \times 4$ multipliers have been presented as elementary blocks for designing higher order multipliers.
- An approximate adder for adding partial products is presented.
- Approximate multipliers based SUSAN image smoothing accelerator has been implemented.
SMApproxLib: Library of FPGA-based Approximate Multipliers

DAC 2018
Contributions

- Approximate multiplier designs optimized for FPGAs.
- For each $N \times N$ accurate multiplier, we provide three approximate $N \times N$ multipliers.
- A design space exploration methodology for generating approximate multipliers of arbitrary data sizes.
- To reduce the execution time of an $N \times N$ multiplier, our methodology recommends implementing it using four instances of $\frac{N}{2}$ multipliers.
- A novel approximate adder for adding the partial products.
Accurate Base Architecture

- Based on the general structure of an $n \times n$ multiplier.
- Utilizes 6-input LUTs and associated carry chains.
Approximate Multiplier Implementations

- Three approximate designs.
- Trades the accuracy for gains in critical path delay & dynamic power.
- Partial products are divided into multiple layers.
- Each layer contains four partial products for maximum utilizing 6-input LUTs.

Partial products of $8 \times 8$ Multiplier
Approximate Multiplier Design 1

- Utilizes LUTs of configuration T-1 and associated carry chains for generating and adding every two consecutive partial product rows.
- Two sum vectors are generated in each layer.
A novel approximate adder is used for adding the sum vectors in each layer.

Every LUT also considers bit values of the preceding column, for calculating the current output bit.

Final product is obtained by adding the final outputs of each layer.
Approximate Multiplier Design 2 & 3

Optimized to improve latency and energy gains

Removing the associated carry chains for partial products accumulation

Approx2 and Approx3 differ only in the computation of partial product bits enclosed by the green boxes.
Approximate Multiplier Design 2

- Optimized to improve latency and energy gains
- A second LUT is utilized for generating the next two partial product terms and adds them with the previously computed sum.
- Approx2 offers reduced latency as compared to Approx1.
Approximate Multiplier Design 3

- Approx3 considers predicting the carry out from preceding bit locations.
- A 6-input LUT is used for computing the first three partial product terms in each group.
- This LUT also implements three AND gates, shaded, to predict the carry from preceding bit locations.
- The three partial product terms and the predicted carry are added together to generate an approximate sum.
- A second LUT is used for adding the approximate sum with the fourth partial product term.
Design Space Exploration

- Utilizes the accurate multiplier, approximate adder and the three approximate multiplier designs
- Provides an open source automated tool flow for implementing approximate multipliers of arbitrary sizes, with different performance gains.
- Performs mapping of logic to LUTs at design time and allows for quick estimation of area and latency requirements of the design.
- Supports a recursive approach of designing $N \times N$ multipliers from $\frac{N}{2}$ multipliers
EvoApprox8b offers 12 non-dominated design points.

24 non-dominated points are presented by the proposed approximate multipliers.

1. M. Shafique et al. 2015. A low latency generic accuracy configurable adder. In IEEE DAC.
Summary

- Modern device/system level challenges forces us to rethink the design principles
- Approximate Computing is not new, but surely opens a new door
- Various mechanisms in various layers proposed to address the challenges and save power
- Can be applied at all levels, but the higher the layer, the bigger the gains
Questions and Answers

Email: akash.kumar@tu-dresden.de