THE UNIVERSITY OF TEXAS AT DALLAS

Department of Electrical and Computer Engineering

TEC | Tecnológico de Costa Rica

Approximating Hardware Accelerators through Partial Extractions onto shared Artificial Neural Networks

Prattay Chowdhury, Jorge Castro Godínez, and **Benjamin Carrion Schaefer**

Prattay.chowdhury@utallas.edu, jocastro@itcr.ac.cr, **schaferb@utdallas.edu**

ASIA SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE

28th Asia and South Pacific Design Automation Conference

ASP-DAC 2023

January 18th, 2023, Tokyo

Outline

- Introduction
  - Target HW platform: Heterogeneous SoC (CPU+HW Accelerators)
  - Heterogenous SoC low power design
- Approximate computing overview
- Contribution
  - Proposed flow
  - Experimental results
- Conclusion

Introduction

Heterogeneous SoC

- System on Chip (SoC) with dedicated HW accelerators

What is Hardware Accelerator?

- Customized circuits to do complex task power efficiently

Problem:

- Power efficient RTL design is time consuming
- RTL has limited re-usability

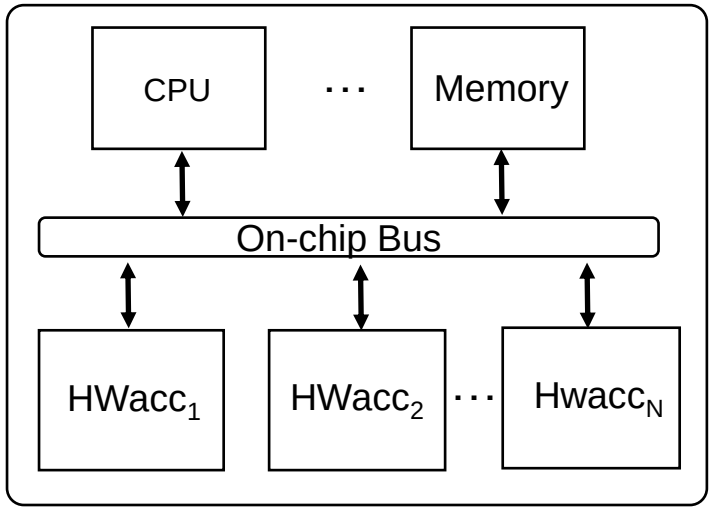
Solution:

- Raise design abstraction level through High-Level synthesis (HLS)
- HLS : “Automatically converts of behavioral, untimed descriptions into hardware that implements that behavior”

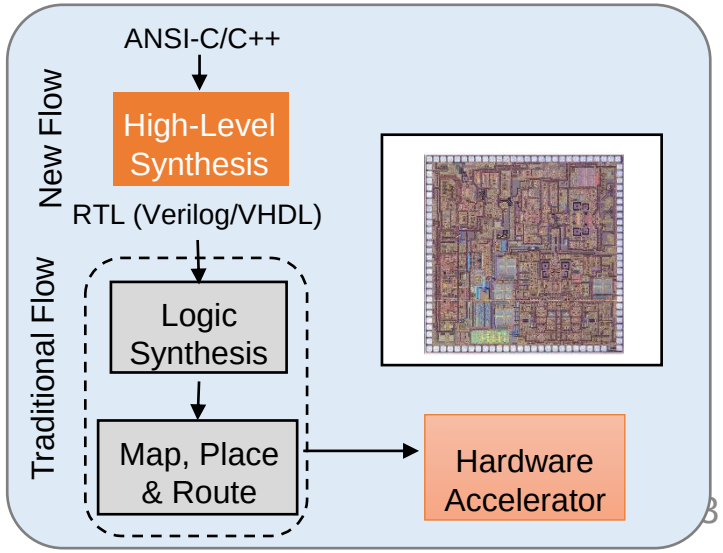
- Objectives of this work:

  Create Low-power HW Accelerators gives as behavioral descriptions for HLS

Heterogeneous System on Chip (SoC)

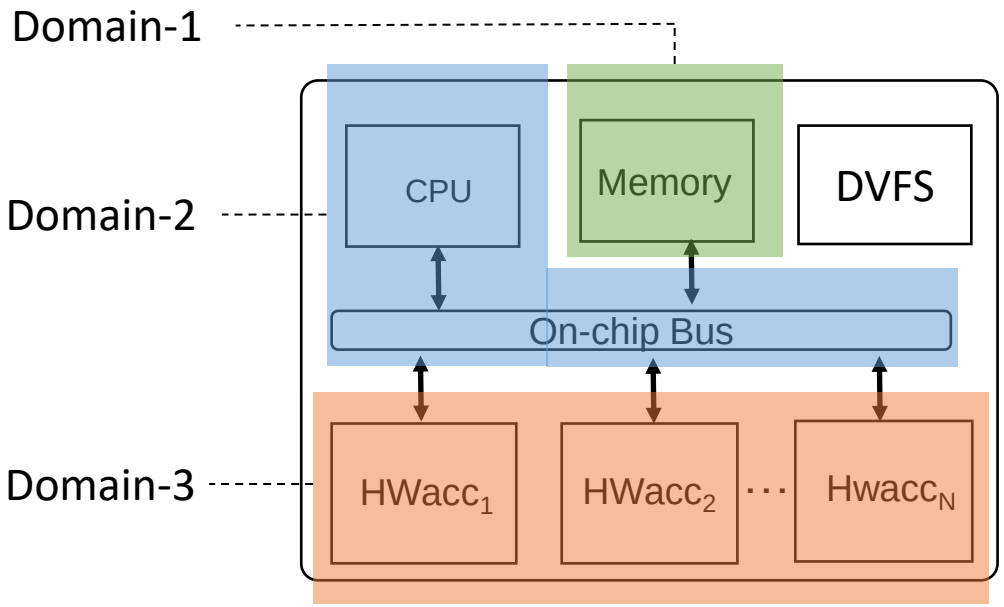


ASIC design overview with HLS



Towards Low Power Design

- Why Low Power?
  - Use of battery run embedded systems
- Common Solution:
  - Use of dynamic voltage and frequency scaling (DVFS)
  - Use of Power Domain and clock gating



➔Orthogonal approach is Approximate Computing

Approximate Computing

What is approximate computing?

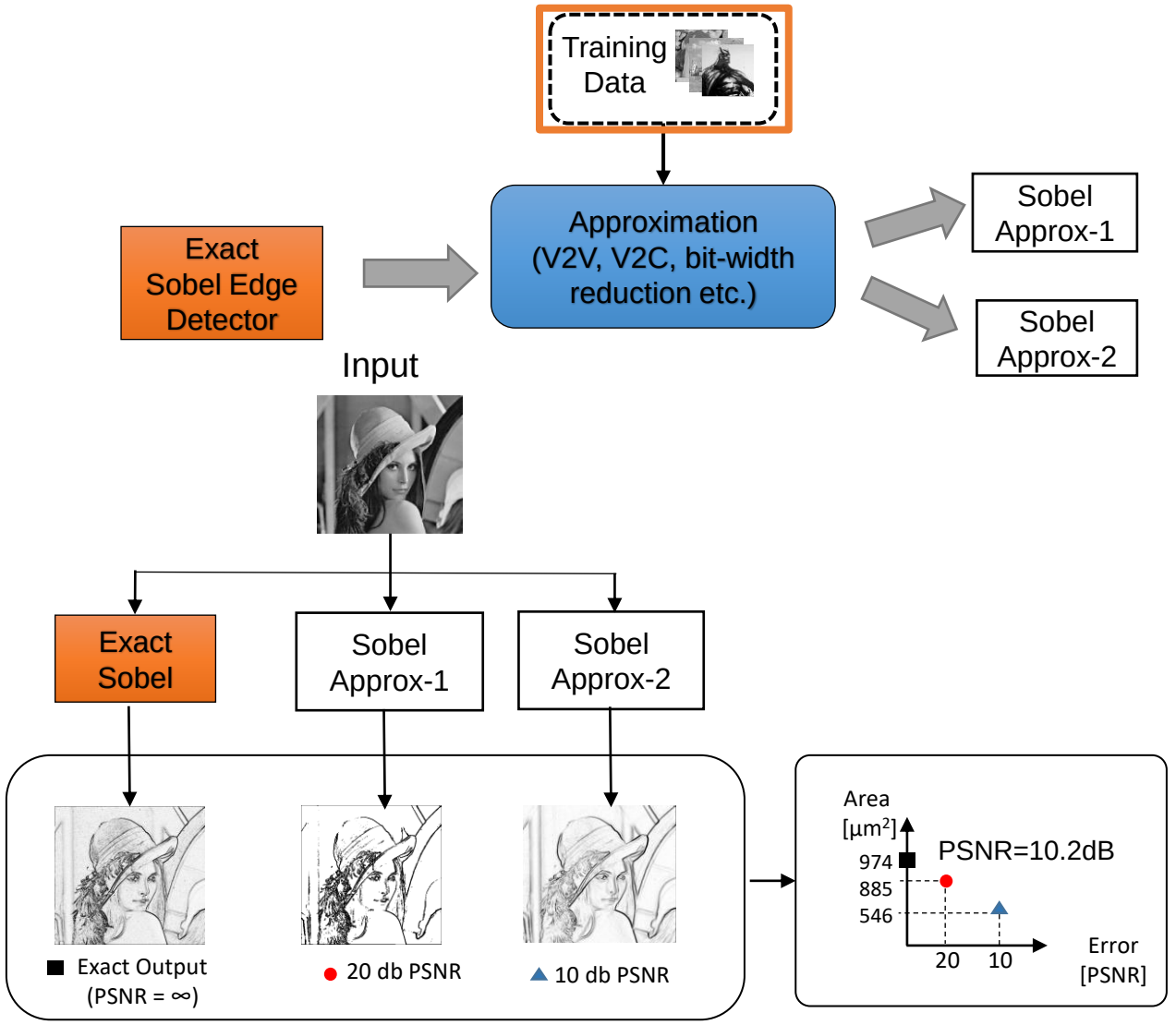
- Trading quality with complexity of a design
- It is a data dependent optimization

Where it is applicable?

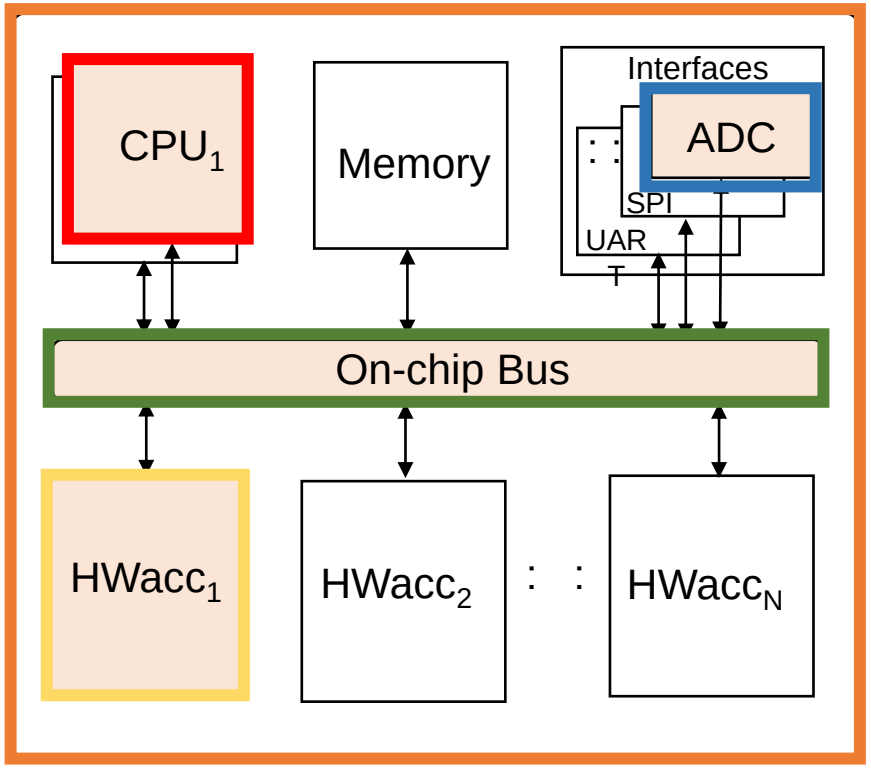
- Error tolerant applications (e.g., Image Processing, DSP) where allowing certain percentage of error leads to large area/power saving

Challenge:

- Finding the optimal design that leads to maximum power/energy saving within the given error threshold



Approximate Computing: Where to Apply?



- ADC Approximation
- Bus Approximation
- CPU Approximation
- **HW Accelerator approximation**
- System level approximation

Approximate Computing Literature Review

Single Component Approximation

Approximation Type	Proposed Method
ADC Approximation	[1],[2]
BUS Approximation	[3]-[5]
CPU Approximation	[6],[7]
HW Acc. Approximation	[8]-[11]

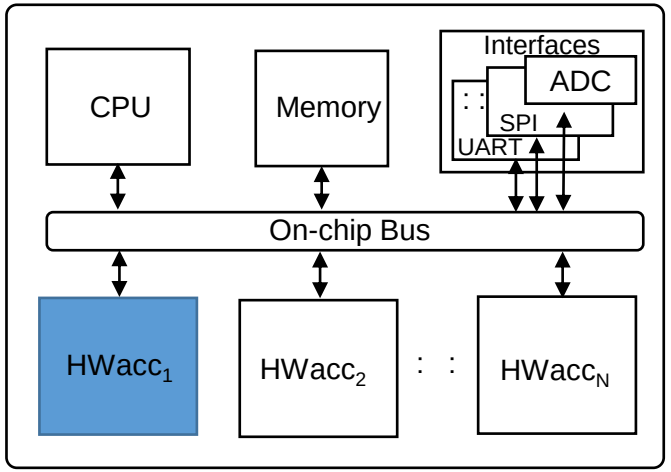
System Level Approximation

Approximation Type	Proposed Method
System Level Approximation	[12]-[14]

[1] Yang et al, "**A 1 GS/s 6 Bit 6.7 mW Successive Approximation ADC Using Asynchronous Processing**," in *IEEE Journal of Solid-State Circuits*, vol. 45, no. 8, pp. 1469-1478, Aug. 2010,
[2] B. Murmann, "**The successive approximation register ADC: a versatile building block for ultra-low- power to ultra-high-speed applications**," in *IEEE Communications Magazine,* 2016
[3] Goiri et al "**Approxhadoop: Bringing approximations to mapreduce frameworks"**, ACM SIGARCH Computer Architecture News, 2015
[4] Samadi et al, "**Sage: Self-tuning approximation for graphics engines",** IEEE/ACM International Symposium on Microarchitecture, 2013
[5] Betzel et al "**Approximate Communication: Techniques for Reducing Communication Bottlenecks in Large-Scale Parallel Systems"**, ACM Computing Surveys, Vol. 51, No. 1, 2018
[6] Sreekumar et al "**Bespoke Behavioral Processors"**, IEEE International Conference on Compute Design (ICCD), 2020
[7] Xu et al, "**Approximating Behavioral HW Accelerators through Selective Partial Extractions onto Synthesizable Predictive Models"**, IEEE/ACM International Conference on Computer Aided Design (ICCAD), pp.1.-8, 2019
[8] Gupta et al , "**Low-power digital signal processing using approximate adders**," IEEE TCAD, vol. 32, no. 1, pp. 124–137, 2013.
[9] Liu et al, "**A low-power, high-performance approximate multiplier with configurable partial error recovery**," in 2014 Design, Automation Test in Europe Conference Exhibition (DATE), 2014.
[10] Xu et al **Exposing Approximate Computing Optimizations at Different Levels: From Behavioral to Gate-Level**, IEEE Transactions on Very Large-Scale Integration (TVLSI) Systems, 2017
[11] Scarabottolo et al "**Circuit carving: A methodology for the design of approximate hardware**," in DATE, March 2018
[12] Shafique et al, "**Cross-layer approximate computing: From logic to architectures,**" in DAC, pp. 1–6, 2016.
[13] Raha et al, "**Towards full-system energy-accuracy tradeoffs: A case study of an approximate smart camera system**," in DAC, 2017
[14] Agrawal et al , "**Approximate computing: Challenges and opportunities**," in 2016 IEEE International Conference on Rebooting Computing (ICRC), pp. 1–8, 2016.

Hardware Accelerator Approximation

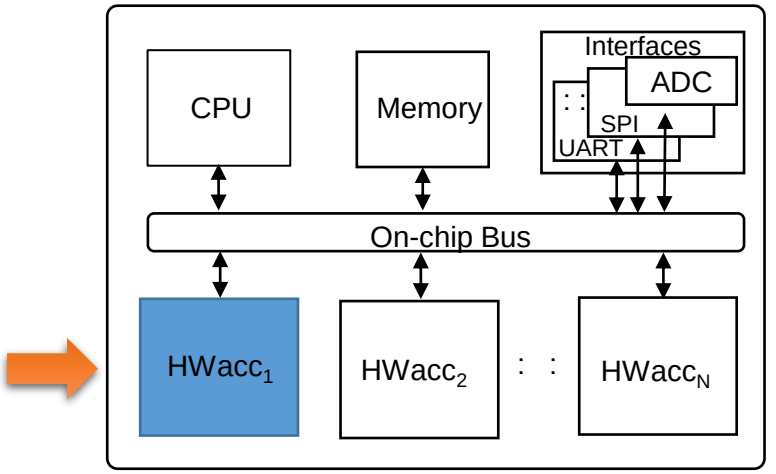
- V2V and V2C Approximations in C and RTL designs
- Approximating FU's (e.g., adders, multipliers) to reduce area/delay [8],[9]
- Reduction of the bit-width of a variable (e.g., output) in C and RTL design [10]



[8] Gupta et al , "**Low-power digital signal processing using approximate adders**," IEEE TCAD, vol. 32, no. 1, pp. 124–137, 2013.
[9] Liu et al, "**A low-power, high-performance approximate multiplier with configurable partial error recovery**," in 2014 Design, Automation Test in Europe Conference Exhibition (DATE), 2014.
[10] Xu et al **Exposing Approximate Computing Optimizations at Different Levels: From Behavioral to Gate-Level**, IEEE Transactions on Very Large-Scale Integration (TVLSI) Systems, 2017

Single Component Approximation Challenges

- One of the powerful approximations are V2V and V2C which lead to maximum energy saving
- V2V and V2C have conditions that happens rarely in regular circuit
- Other methods (FU approximation reduction) cannot unlock full potential of approximation



➔ Propose an artificial neural network (ANN) based hardware accelerator approximation method given a behavioral description for HLS

Why ANNs ?

```
// 2-stage
// Interpolation
// filter
----------------------
----------------------
// Filter 1
x1=filt1*coef1+
  filt2*coef2;
--------------------
--------------------
// Filter 2
x2=filt3*coef3+
  filt4*coef4;
--------------------
--------------------
// output
odata=x1+x2*factor;

--------------------
--------------------
--------------------
```

(a)

```
// Approximated 2-stage
// Interpolation filter
// using standard techniques
----------------------
----------------------
// Filter 1
x1=approx_mul(filt1,coef1)+
  filt2*coef2;
--------------------
--------------------
// Filter 2
x2=filt3*coef3+
 lut(filt4);
--------------------
--------------------
// output
odata=x1+x2*factor;

--------------------
--------------------
--------------------
```

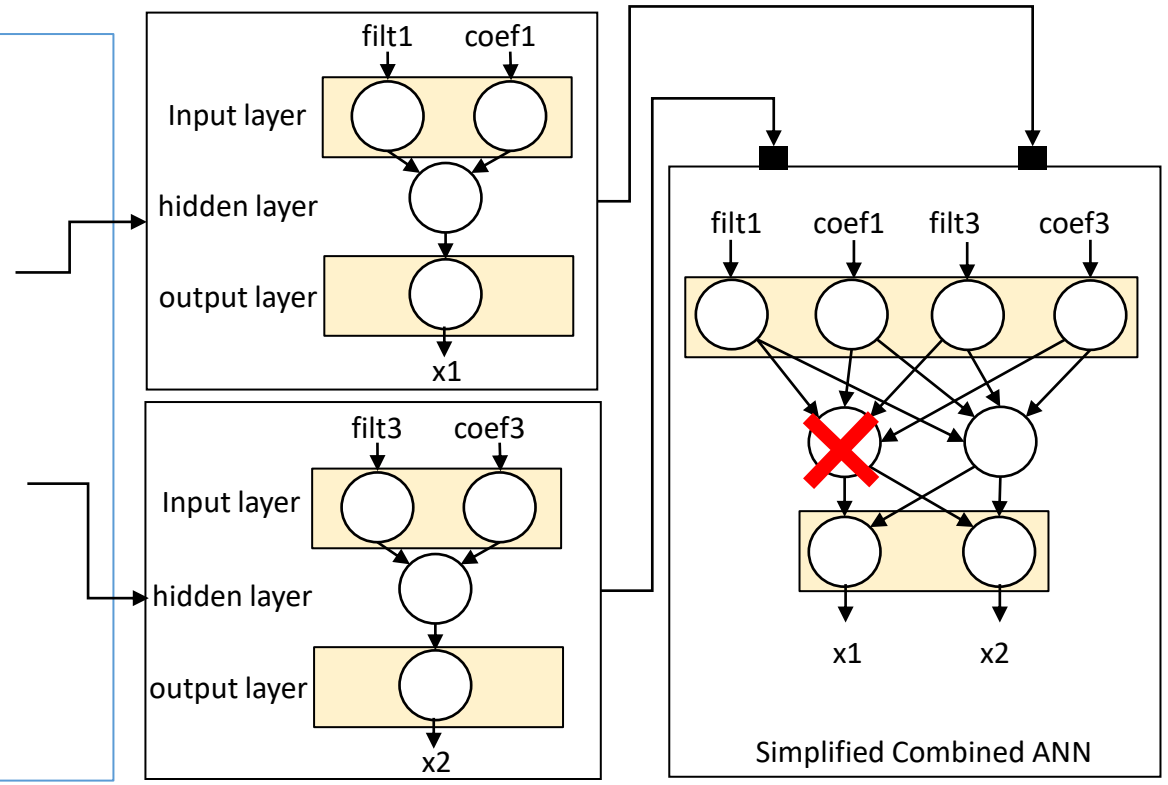
(b)

```
// Approximated 2-stage
// Interpolation filter
// using proposed ANN
----------------------
----------------------
// Filter 1
x1=ann(filt1,coef1)+
  filt2*coef2;
--------------------
--------------------
// Filter 2
x2=filt3*coef3+
 ann(filt3, coef4);


// output
odata=x1+x2*factor;

--------------------
--------------------
--------------------
```

(c)

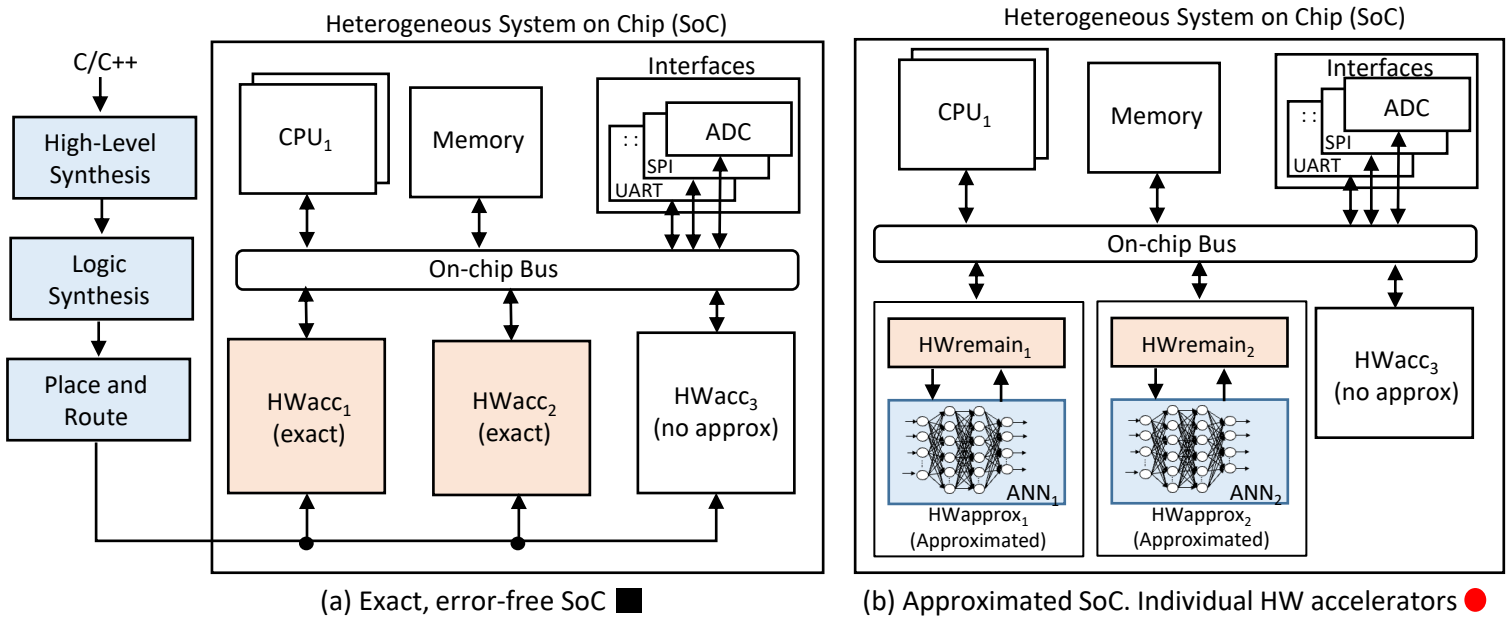


(d) (e)

Main benefit: The ability to approximate multiple portions of the design, merge it and re-optimize the ANN to maximize savings

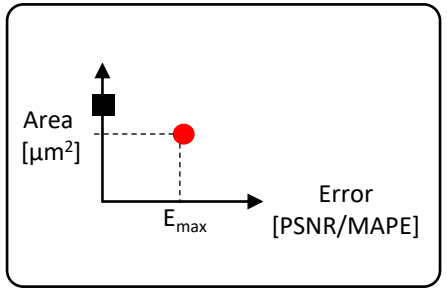
Note: ANN is trained and weights and biased fixed → Multipliers synthesized as shifts+additions

Proposed Architecture



(a) Exact, error-free SoC ■

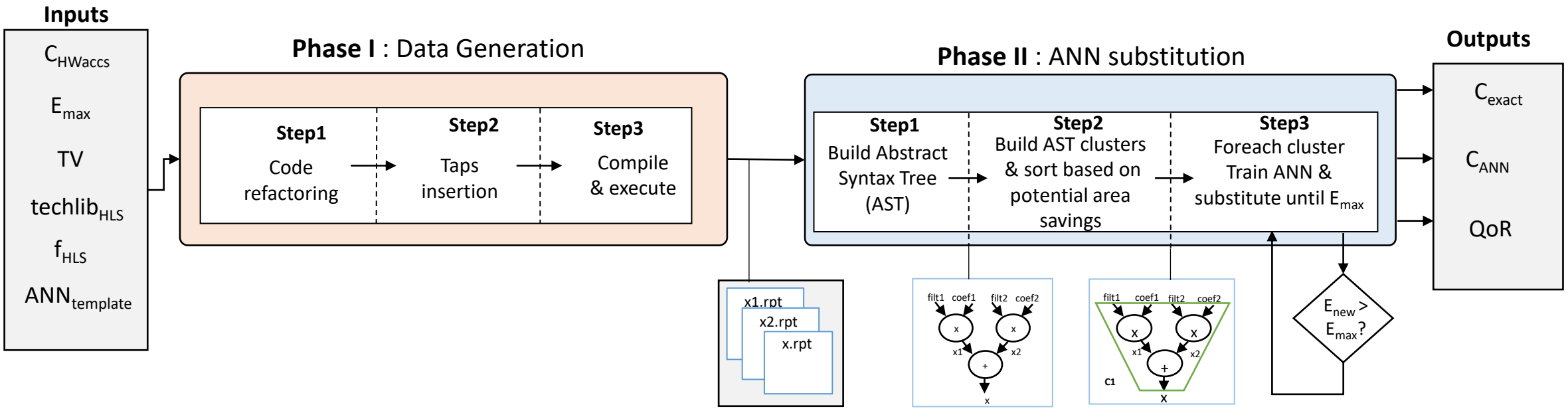
(b) Approximated SoC. Individual HW accelerators ●



Effect of approximations on SoC area given a maximum error threshold E_{max}

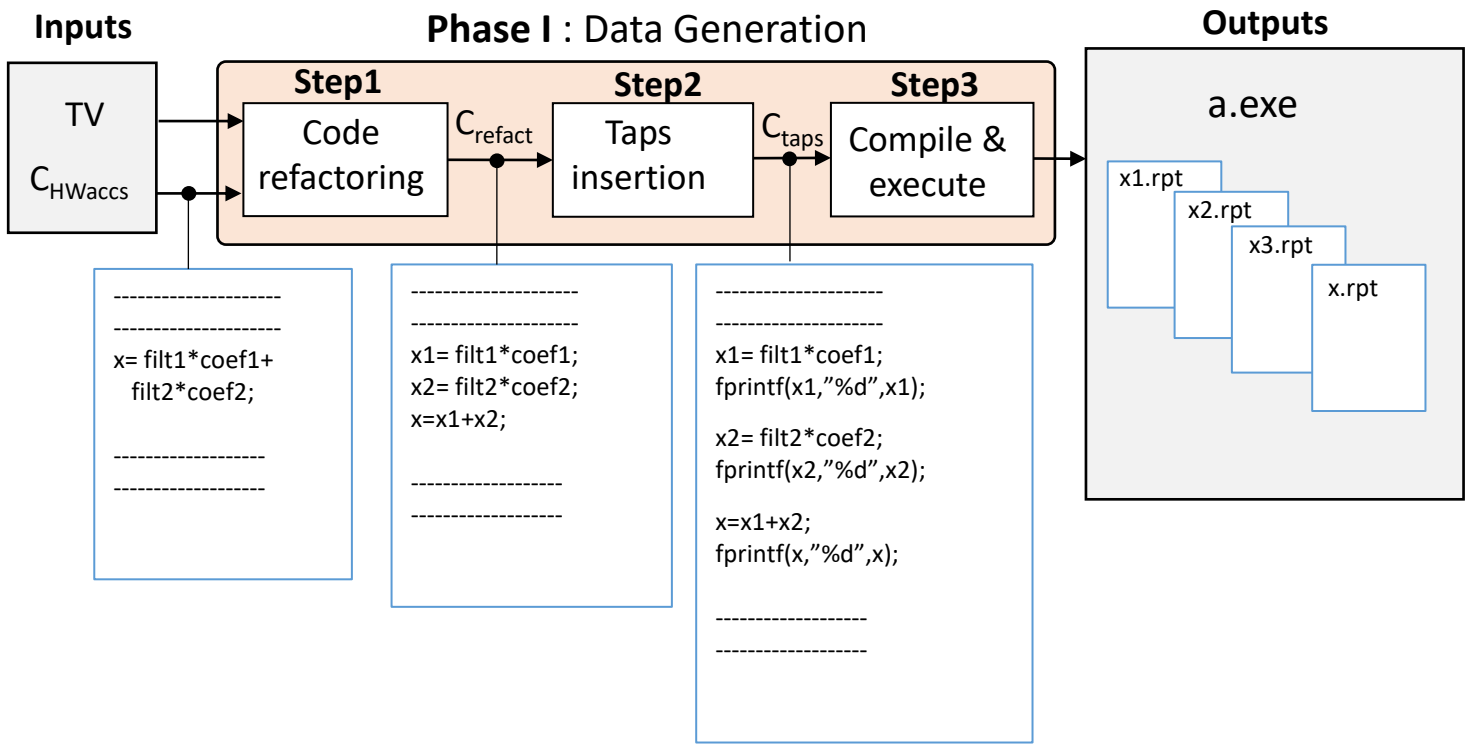
- Partition hardware accelerator into exact version + approximated
- Accelerator specified as an untimed behavioral description for HLS
- Approximated through trained ANN

Overview of Proposed Flow



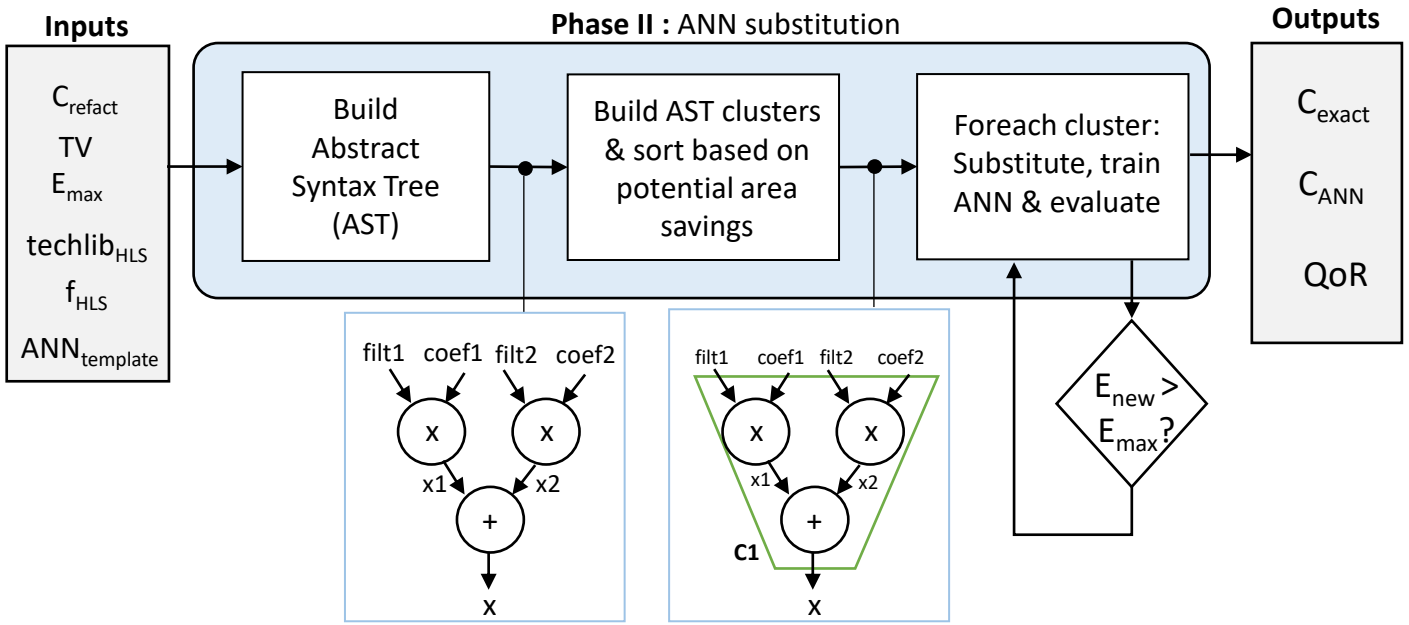
- Flow composed of two phases:
  Phase 1: Training data generation
  Phase 2: ANN substitution

Phase 1: Data Generation



- Codes are refactored (Function inlining, loop unrolling, variable renaming, expression decomposition) to expose more variables
- Taps are inserted in order to observe the values of all the internal signals
- Modified code is executed with given test data and record of all the internal variables are obtained

Phase 2 : ANN Substitution



1. Takes the refactored code, test vector and other constrains and an ANN template
2. Builds tree based on the dependency of variables
3. Builds clusters inside tree and sort them based on potential area saving starting with the highest saving
4. Continuously adds cluster to ANN+ train until E_{max} is reached
5. Merge ANNs of different HW accelerators

Outputs:

1. Remaining exact accelerator
2. Trained ANN
3. Quality report file

Error Metric for Approximation

- Peak Signal to Noise Ratio (PSNR) for image processing

$$PSNR = 20 \log \frac{255}{MSE}$$

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (GO_i - AO_i)^2$$

$$GO = Golden\ Output, AO = Actual\ Output$$

- Mean absolute percentage error (MAPE) for DSP

$$MAPE = \frac{1}{N} \left| \frac{GO_i - AO_i}{GO_i} \right| \times 100$$

- Target error is application dependent
- Research works use 10-20% MAPE and 10 to 20db PSNR for approximation

Experimental Setup (All experiments)

Tools

- ML Tool: Python scikit-learn, TensorFlow v2.6
- HLS Tools : NEC CyberWorkBench v.6.1
- Logic Synthesis tool: Synopsys Design Compiler v.0-2018.06-SP1
- Target technology: Nangate Opencell 45nm
- Target synthesis frequency: 100 MHz
- Power Simulator: Synopsys PrimePower v.P-2019.03-SP5
- RTL and GL Simulator: Synopsys VCS 0-2018.06

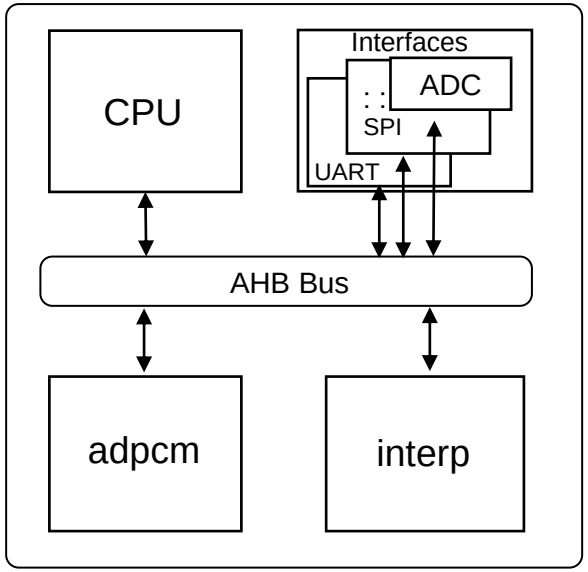
Evaluation

- Hardware Accelerator: S2CBench Benchmark suite
- Processor: 32-bit MIPS, Bus: AMBA-AHB
- Compare our work vs. state-of-the-art applying a variety of approximation primitives to the hardware accelerator (i.e., bit width reduction, V2V, V2C)

Experimental Setup – SoC Configuration

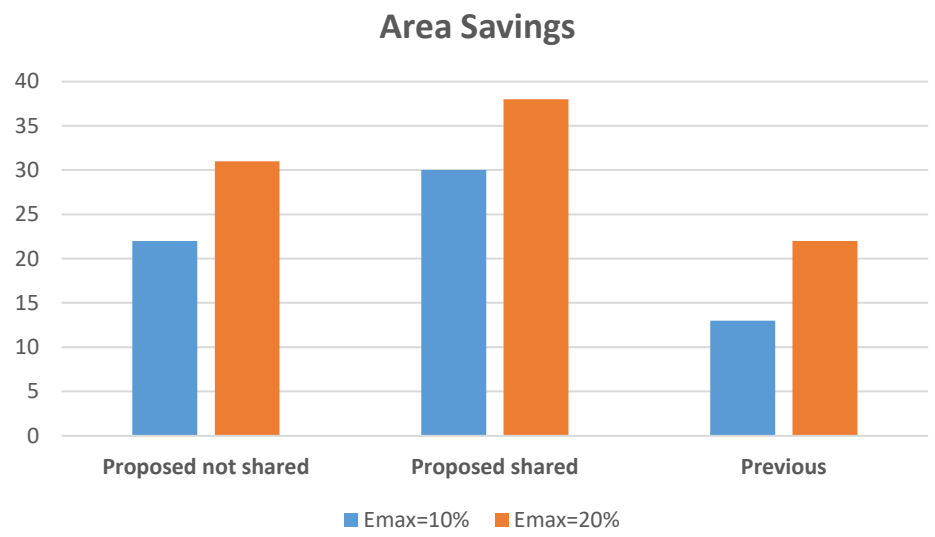
System Benchmark Configuration Overview

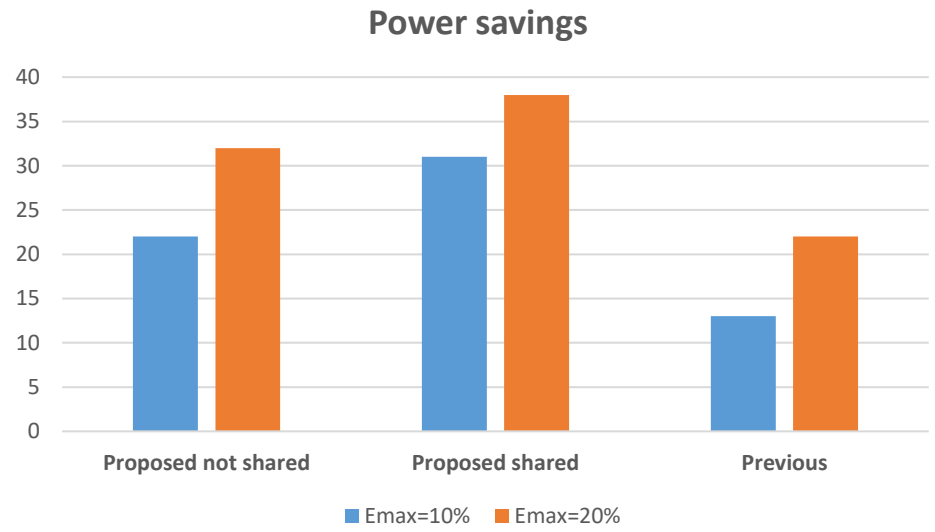
System		S1	S2	S3	S4	S5	S6
CPU		×	×	×	×	×	×
Bus (AHB)		×	×	×	×	×	×
HWacc	adpcm	×	×			×	
	fft				×		×
	interp	×		×			×
	decim		×			×	
	idct			×	×		
	jpeg					×	×



- 3 cases considered and compared against the error free hw accelerators (**exact**)
  - **Previous**: Apply library of well-known approximations (bw reduction, V2V, V2C, etc.)
  - **Proposed not shared**: ANN method individually for each hw accelerator
  - **Proposed shared**: Share ANN between different hw accelerators

Experimental Result – Area/Power Reduction

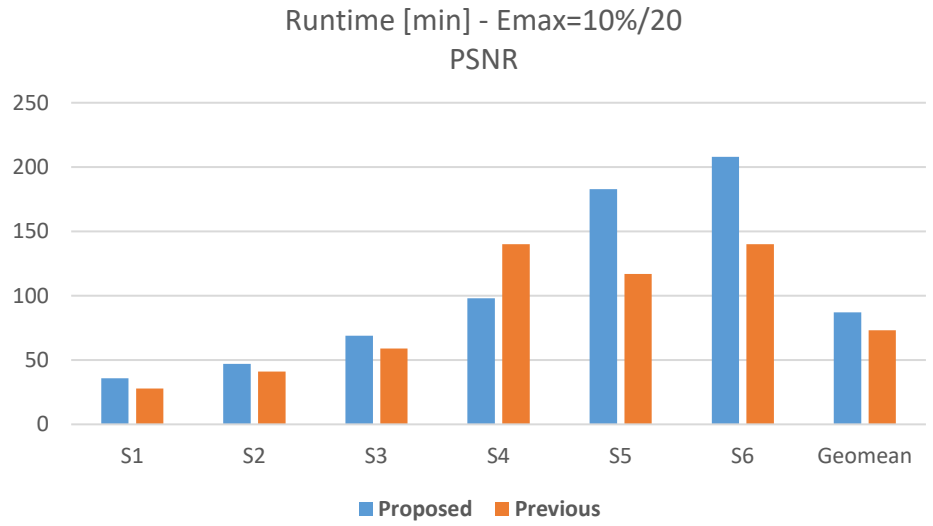


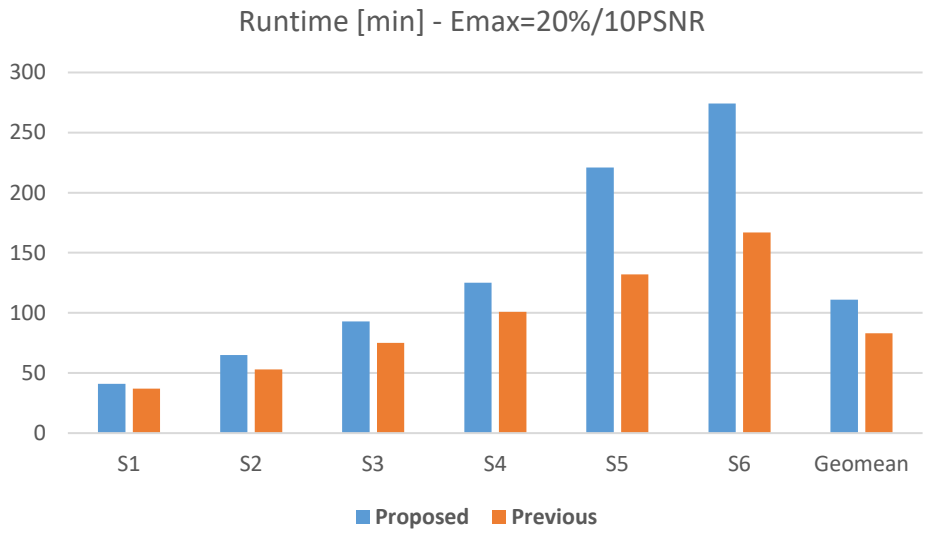


- Two error cases has been considered for experiment
  - $\mathbf{E_{max}}$**=** 10% MAPE/20db PSNR – Smaller error margin
  - $\mathbf{E_{max}}$ 20% MAPE/10db PSNR) – Relaxed error margin

Observation 1: The higher the error margin the better the results (~22% vs. 31%)
Observation 2: Our approach is on average ~10% better that the state of the art for single accelerator
Observation 3: Further sharing the ANN across accelerators further increases the savings by~8%

Experimental Result - Runtime

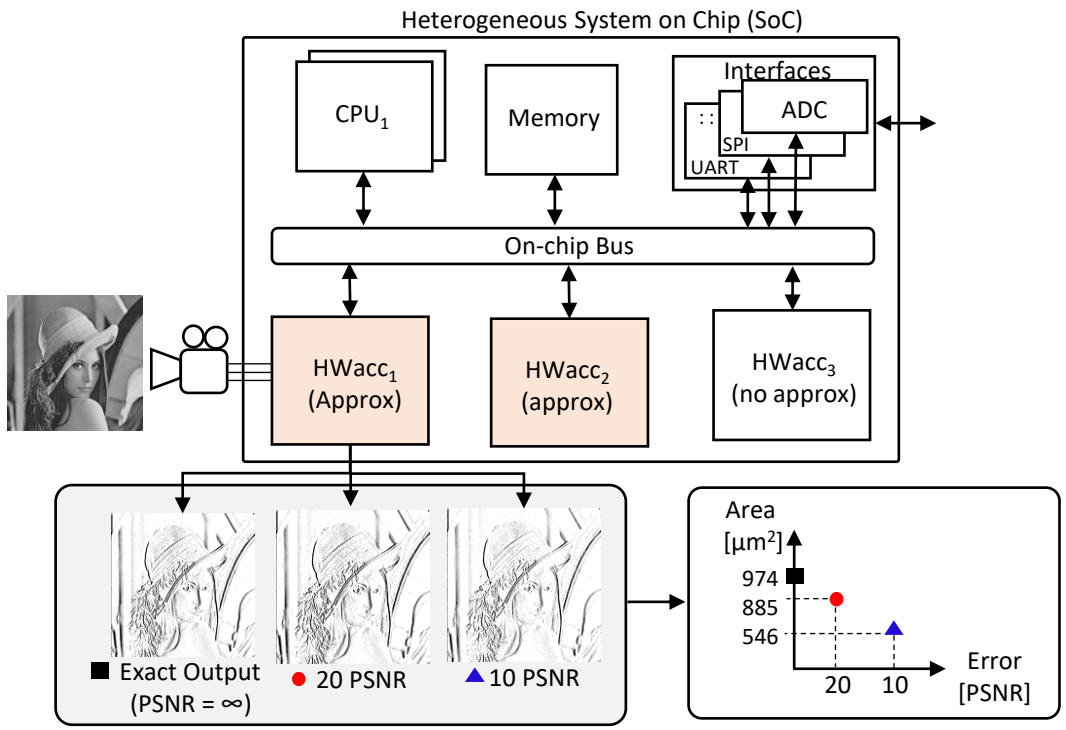




- Due to training and retraining ANN, our proposed method is slower than previous based on runtime
- Our method 1.30x and 1.34x more time than previous work for single ANNs approximations

Conclusions

- Proposed an approximation method for hardware accelerators specified as untimed behavioral descriptions for HLS
  - Method splits accelerator into exact part and ANN
  - ANN can map different portions of the accelerator
- Our proposed method saves more area and power with respect to previous state of the art using a variety of different approximation primitives



Thank You