ECE 6745 Complex Digital ASIC Design Course Overview

Christopher Batten

School of Electrical and Computer Engineering Cornell University

http://www.csl.cornell.edu/courses/ece6745



Complex Digital ASIC Design

- Course goal, motivation, structure
 - What is the goal of the course?
 - Why should students want to take this course?
 - How is the course structured?
- Activity: Evaluation of Integer Multiplier
- ASIC Design Case Studies
 - Example design-space exploration
 - Example real ASIC chips



ASIC Design Case Studies

The Computer Systems Stack

Application
Algorithm
Programming Language
Operating System
Compiler
Instruction Set Architecture
Microarchitecture
Register-Transfer Level
Gate Level
Circuits
Devices
Technology

In its broadest definition, computer architecture is the design of the abstraction/implementation layers that allow us to execute information processing applications efficiently using available manufacturing technologies

ASIC Design Case Studies

What is Computer Architecture?



Application Requirements

- Provide motivation for building system
- SW/HW interface expressive yet productive

Computer architects provides feedback to guide application and technology research directions

Technology Constraints

- Restrict what can be done efficiently
- New technologies make new arch possible

In its broadest definition, computer architecture is the design of the abstraction/implementation layers that allow us to execute information processing applications efficiently using available manufacturing technologies



Key Metrics in Computer Architecture



- Primary Metrics
 - Execution time (cycles/task)
 - Energy (Joules/task)
 - Cycle time (ns/cycle)
 - ⊳ Area (µm²)
- Secondary Metrics
 - Performance (ns/task)
 - Average power (Watts)
 - Peak power (Watts)
 - ▷ Cost (\$)
 - Design complexity
 - Reliability
 - Flexibility

Discuss qualitative first-order analysis from ECE 4750 on board

Unanswered Questions from ECE 4750



- How can we quantitatively evaluate area, cycle time, and energy?
- How do we actually implement processors, memories, and networks in a real chip?
- How should we implement/analyze application-specific accelerators?
 - Very loosely coupled memory-mapped accelerators
 - More tightly coupled co-processor accelerators
 - Specialized instructions and functional units

ASIC: Application-Specific Integrated Circuit



ASIC Design Case Studies

ASIC: Application-Specific Integrated Circuit



Goal for ECE 6745 is to answer these questions!



- How can we quantitatively evaluate area, cycle time, and energy?
- How do we actually implement processors, memories, and networks in a real chip?
- How should we implement/analyze application-specific accelerators?
 - Very loosely coupled memory-mapped accelerators
 - More tightly coupled co-processor accelerators
 - Specialized instructions and functional units

Full Custom Design vs. Standard-Cell Design



Full-custom layout in 1.0µm w/ 2 metal layers

Full-Custom Design (ECE 4740)

- Designer is free to do anything, anywhere; though team usually imposes some design discipline
- Most time consuming design style; reserved for very high performance or very high volume chips (Intel microprocessors, RF power amps for cellphones)
- Standard-Cell Design (ECE 6745)
 - Fixed library of "standard cells" and SRAM memory generators
 - Register-transfer-level description is automatically mapped to this library of standard cells, then these cells are placed and routed automatically
 - Enables agile hardware design methodology

ASIC Design Case Studies

Standard-Cell Design Methodology



ASIC Design Case Studies

Standard-Cell Design Methodology



ASIC Design Case Studies

Example Standard-Cell Chip Plot



Control Processor	8.1%
Vector Register File	56.9%
Vector Integer ALUs	9.7%
Vector FPUs	9.4%
Vector Memory Units	7.6%
Other	8.3%

810 µm

What is Complex Digital ASIC Design?



Complex digital ASIC design is the process of

quantitatively exploring the area, cycle time, execution time, and energy trade-offs

of various

application-specific accelerators (and general-purpose proc+mem+net)

using

automated standard-cell CAD tools

and then to transform the most promising design to

layout ready for fabrication



Complex Digital ASIC Design

- Course goal, motivation, structure
 - What is the goal of the course?
 - Why should students want to take this course?
 - > How is the course structured?
- Activity: Evaluation of Integer Multiplier
- ASIC Design Case Studies
 - Example design-space exploration
 - Example real ASIC chips



Motivation for Taking Course

- Trends in Artificial Intelligence
- 2023/2024 Sabbatical
 - Fall 2023: Slice Lab @ UC Berkeley
 - Spring 2024: NVIDIA Research @ Santa Clara, CA

Al for Image Recognition

Rate

Error

S

Top



Christopher Batten



Software: Deep Neural Network

AI in Computer Engineering • ECE Retreat



Hardware: Graphics Processing Units



AI for Protein Folding

Amino Acid Sequence

APRKFFVGGNWKMNGDKKSLGELIHTLNGAKL SADTEVVCGAPSIYLDFARQKLDAKIGVAAQN CYKVPKGAFTGEISPAMIKDIGAAWVILGHSE RRHVFGESDELIGQKVAHALAEGLGVIACIGE KLDEREAGITEKVVFEQTKAIADNVKDWSKVV LAYEPVWAIGTGKTATPQQAQEVHEKLRGWLK THVSDAVAQSTRIIYGGSVTGGNCKELASQHD VDGFLVGGASLKPEFVDIINAKH

Tertiary Protein Structure



Christopher Batten



- Trained using 128 Tensor Processing Units (TPUv3) for 1.5 weeks
- Inference for large proteins requires hours on 4 NVIDIA V100 GPUs

Al for Natural Language Q&A

Top-performing open and closed AI models on Massive Multitask Language Understanding benchmark which includes 16,000 multiple-choice questions spanning 57 subject areas



Christopher Batten

Al for Natural Language Q&A

GPQA: Dataset of 448 multiple-choice questions written by experts in biology, physics, and chemistry

Molecular Biology

A scientist studies the stress response of barley to increased temperatures and finds a protein which contributes to heat tolerance through the stabilisation of cell membrane. The scientist is very happy and wants to create a heat-tolerant cultivar of diploid wheat. Using databases, they find a heat tolerance protein homologue and start analysing its accumulation under heat stress. Soon enough, the scientist discovers this protein is not synthesised in the wheat cultivar they study. There are many possible reasons for such behaviour, including:

A) A miRNA targets the protein, which makes exonucleases cut it immediately after the end of translation and before processing in ER

B) Trimethylation of lysine of H3 histone in position 27 at the promoter of the gene encoding the target protein

C) A stop-codon occurs in the 5'-UTR region of the gene encoding the target protein

D) The proteolysis process disrupts a quaternary structure of the protein, preserving only a tertiary structure

Astrophysics

Astronomers are studying a star with a Teff of approximately 6000 K. They are interested in spectroscopically determining the surface gravity of the star using spectral lines (EW < 100 mA) of two chemical elements, El1 and El2. Given the atmospheric temperature of the star, El1 is mostly in the neutral phase, while El2 is mostly ionized. Which lines are the most sensitive to surface gravity for the astronomers to consider?

A) El2 I (neutral)

B) El1 II (singly ionized)

C) El2 II (singly ionized)

D) El1 I (neutral)

Quantum Mechanics

Suppose we have a depolarizing channel operation given by $E(\rho)$. The probability, p, of the depolarization state represents the strength of the noise. If the Kraus operators of the given state are $A_0 = \sqrt{1 - \frac{3p}{4}}, A_1 = \sqrt{\frac{p}{4}}X, A_2 = \sqrt{\frac{p}{4}}Y$, and

 $A_3 = \sqrt{\frac{p}{4}Z}$. What could be the correct Kraus Representation of the state $E(\rho)$?

A) $E(\rho) = (1 - p)\rho + \frac{p}{3}X\rho X + \frac{p}{3}Y\rho Y + \frac{p}{3}Z\rho Z$ B) $E(\rho) = (1 - p)\rho + \frac{p}{3}X\rho^2 X + \frac{p}{3}Y\rho^2 Y + \frac{p}{3}Z\rho^2 Z$ C) $E(\rho) = (1 - p)\rho + \frac{p}{4}X\rho X + \frac{p}{4}Y\rho Y + \frac{p}{4}Z\rho Z$ D) $E(\rho) = (1 - p)\rho^2 + \frac{p}{3}X\rho^2 X + \frac{p}{3}Y\rho^2 Y + \frac{p}{3}Z\rho^2 Z$



https://epoch.ai/data/ai-benchmarking-dashboard

Christopher Batten

Al in Computer Engineering • ECE Retreat

Anthropic

Al Training in the Cloud



- Exponential growth in training requirements at a rate of ~4x/year
 - 2×10²² FLOPs on a rack of AMD EPYC CPUS (each @ 2 TFLOPS) would take several years to train average model
 - 2×10²² FLOPs on the latest NVDIA Al supercomputer (Blackwell NVL72
 @ 720 PFLOPS) would take less than a day to train average model



https://epoch.ai/blog/training-compute-of-frontier-ai-models-grows-by-4-5x-per-year

Christopher Batten

Al Inference in the Cloud



Christopher Batten



Client & Edge Al Hardware



Al hardware is critical to enabling new Al capabilities, so any discussion about Al without talking about Al hardware is meaningless



Motivation for Taking Course

- Trends in Artificial Intelligence
- 2023/2024 Sabbatical
 - Fall 2023: Slice Lab @ UC Berkeley
 - Spring 2024: NVIDIA Research @ Santa Clara, CA

Computer Architecture Research in ML/AI

- Tremendous excitement in the computer architecture community over the past 10 years around hardware accelerators for ML and AI
- Much of this work focuses on an accelerator in isolation and/or with small ML/Al microbenchmarks
- Goal for Fall 2023 sabbatical was to learn more about full-system Al accelerators running complete Al software stacks



Fall 2023: Sabbatical @ UC Berkeley

- The UC Berkeley Slice lab seeks to comporatize the design, no gran miles and integration of domain specific computing at scale
- Leverages significant prior investment in systembuilding infrastructure





Hammer

ristoph there is a chitecture Rage to mputer Engineering • ECE Retreat

Gemmini Deep-Learning Accelerator

- Systolic array for dense matrix and vector operations
- Integrated with two Linux-capable cores using DMA and hardware cache coherence
- Runs completed end-to-end machine-learning software stacks
- Enables studying context switches, page table evictions, multiprogrammed workloads
- Includes a "generator" to enable broad system-level design-space exploration



2023/2024 Sabbatical Key Takeaways





Al for Chip Design



Christopher Batten

Spring 2024: Sabbatical @ NVIDIA



Exploring large-language models for chip design in Brucek Khailany's Accelerator & VLSI Research Group





Christopher Batten

PyMTL: A Python Framework for Hardware Modeling, Generation, Simulation, & Verification



Christopher Batten

Embedded DSLs & LLMs

A hardware design framework embedded in the most popular generalpurpose programming language can potentially make **design engineers** more effective

Hypothesis

A hardware design framework embedded in the most popular generalpurpose programming language can potentially make LLMs more effective


PyHDL-Eval Framework



- Each problem includes prompt, Verilog reference, Verilog test bench, and Python test script
- **Generate** script supports different ICL examples, models, model hosting services, and languages
- Verilog reference can be tested against itself using both Icarus Verilog and Python via PyMTL3/Verilator
- LLM-generated Verilog RTL is tested against Verilog reference using Icarus Verilog
- **Analysis scripts** enable automatic failure classification
- LLM-generated Pythonembedded DSL RTL is tested against Verilog reference using Python test scripts
- Includes workflow management scripts

Christopher Batten

Al in Computer Engineering • ECE Retreat

PyHDL-Eval Problems

Pro	Problem Category Examples of Problems in Category					
	legory			11003		
1	comb	const	constant zero, one, lohi, 32-bit value	4		
2	comb	wires	pass through, split, bit/byte reverse, sign extend	8		
3	comb	gates	single logic gate, multiple logic gates, natural language	13		
4	comb	bool	logic equations, truth table, K-map, waveform, natural language	15		
5	comb	mux	various bitwidths and number of inputs, mux/demux	9		
6	comb	codes	encoder, decoder, priority encoder, graycode, BCD, ASCII, parity	14		
7	comb	arith	add/sub, popcount, multiplication, shift/rotate, comparison, ALU, sorter	17		
8	comb	fsm	FSM tables and diagrams for next state and output logic	9		
9	comb	param	parameterized: gates, mux, encoder, decoder, priority encoder, rotator	8		
10	seq	gates	single gate, multiple gates, natural language	9		
11	seq	bool	truth table, waveform	5		
12	seq	sreg	shift registers: SISO, PISO, SIPO, universal, bidir, LFSR	7		
13	seq	count	counters: binary, up/down, saturating, variable, decade, timer, clock	9		
14	seq	edge	edge detect, edge capture, edge counters, max switching	5		
15	seq	arb	arbiters: priority, rotational oblivious, round robin, grant-hold, weighted	5		
16	seq	fsm	FSM tables and diagrams, natural language	13		
17	seq	mem	1r1w, 2r1w register file (forwarding, reg zero, byte enables), 1s1w CAM	6		
18	seq	arith	bit serial incrementer, bit serial adder, byte-serial adder, accumulator	4		
19	seq	pipe	1/2/3-stage delay, 1/2-stage 2/3/4-input adder, 2-stage minmax	8		

- New evaluation benchmark with 168 specification-to-RTL problems developed from scratch
- Careful **ontological approach** with problems categorized into classes and subclasses
 - 97 combinational problems
 - 71 sequential problems
 - 19 subclasses

AI in Computer Engineering • ECE Retreat

PyHDL-Eval Problem 9.8

Pro Cat	blem tegory	lem gory Examples of Problems in Category			
1	comb	const	constant zero, one, lohi, 32-bit value	4	
2	comb	wires	pass through, split, bit/byte reverse, sign extend	8	
3	comb	gates	single logic gate, multiple logic gates, natural language	13	
4	comb	bool	logic equations, truth table, K-map, waveform, natural language	15	
5	comb	mux	various bitwidths and number of inputs, mux/demux	9	
6	comb	codes	encoder, decoder, priority encoder, graycode, BCD, ASCII, parity	14	
7	comb	arith	add/sub, popcount, multiplication, shift/rotate, comparison, ALU, sorter	17	
8	comb	fsm	FSM tables and diagrams for next state and output logic	9	
9	comb	param	parameterized: gates, mux, encoder, decoder, priority encoder, rotator	8	
10	seq	gates	single gate, multiple gates, natural language	9	
11	seq	bool	truth table, waveform	5	
12	seq	sreg	shift registers: SISO, PISO, SIPO, universal, bidir, LFSR	7	
13	seq	count	counters: binary, up/down, saturating, variable, decade, timer, clock	9	
14	seq	edge	edge detect, edge capture, edge counters, max switching	5	
15	seq	arb	arbiters: priority, rotational oblivious, round robin, grant-hold, weighted	5	
16	seq	fsm	FSM tables and diagrams, natural language	13	
17	seq	mem	1r1w, 2r1w register file (forwarding, reg zero, byte enables), 1s1w CAM	6	
18	seq	arith	bit serial incrementer, bit serial adder, byte-serial adder, accumulator	4	
19	seq	pipe	1/2/3-stage delay, 1/2-stage 2/3/4-input adder, 2-stage minmax	8	

Implement a hardware module named TopModule with the following interface. All input and output ports are one bit unless otherwise specified.

- parameter nbits
- input in_ (nbits)
- input amt (log2(nbits))
- input op
- output out (nbits)

- 0 : rotate left

- 1 : rotate right

The module should implement a variable rotator which takes as input a value to rotate (in_) and the rotation amount (amt) and writes the rotated result to the output (out). The module should be parameterized by the bitwidth (nbits) of the input and output ports; nbits can be assumed to be a power of two. The input op specifies what kind of rotate to perform using the following encoding:

```
module RefModule
#(
  parameter nbits
)(
 input logic [
                       nbits-1:0] in_,
 input logic [$clog2(nbits)-1:0] amt,
 input logic
                                   op,
 output logic [
                       nbits-1:0] out
);
  logic [(2*nbits)-1:0] temp;
 always @(*) begin
  if ( op == 1'd0 ) begin
    temp = { in_, in_ } << amt;
    out = temp[(2*nbits)-1:nbits];
  end
  else begin
    temp = { in_, in_ } >> amt;
    out = temp[nbits-1:0];
  end
  end
```

endmodule

Christopher Batten

Al in Computer Engineering • ECE Retreat

PyHDL-Eval Problem 19.6

Pro Cat	blem tegory		Examples of Problems in Category	
1	comb	const	constant zero, one, lohi, 32-bit value	4
2	comb	wires	pass through, split, bit/byte reverse, sign extend	8
3	comb	gates	single logic gate, multiple logic gates, natural language	13
4	comb	bool	logic equations, truth table, K-map, waveform, natural language	15
5	comb	mux	various bitwidths and number of inputs, mux/demux	9
6	comb	codes	encoder, decoder, priority encoder, graycode, BCD, ASCII, parity	14
7	comb	arith	add/sub, popcount, multiplication, shift/rotate, comparison, ALU, sorter	17
8	comb	fsm	FSM tables and diagrams for next state and output logic	9
9	comb	param	parameterized: gates, mux, encoder, decoder, priority encoder, rotator	8
10	seq	gates	single gate, multiple gates, natural language	9
11	seq	bool	truth table, waveform	5
12	seq	sreg	shift registers: SISO, PISO, SIPO, universal, bidir, LFSR	7
13	seq	count	counters: binary, up/down, saturating, variable, decade, timer, clock	9
14	seq	edge	edge detect, edge capture, edge counters, max switching	5
15	seq	arb	arbiters: priority, rotational oblivious, round robin, grant-hold, weighted	5
16	seq	fsm	FSM tables and diagrams, natural language	13
17	seq	mem	1r1w, 2r1w register file (forwarding, reg zero, byte enables), 1s1w CAM	6
18	seq	arith	bit serial incrementer, bit serial adder, byte-serial adder, accumulator	4
19	sea	pipe	1/2/3-stage delay, 1/2-stage 2/3/4-input adder, 2-stage minmax	8

Implement a hardware module named TopModule with the following interface. All input and output ports are one bit unless otherwise specified.

-	input	clk

	input	in0	(8)	bits)
	input	in1	(8)	bits)
ł	input	in2	(8)	bits)
		01	10	1-2

- output out01 (8 bits)
- output out (8 bits)

The module should implement a two-stage pipelined three-input adder. Just the first addition of in0 and in1 should occur during the first stage, and then the second addition of this first sum and input in2 should occur during the second stage. The final result should be written to the output out at the end of the second stage. The module should write output out01 with the result of the first addition at the end of the first stage.

All transactions should have a two cycle latency (i.e., inputs on cycle i will produce the corresponding sum on cycle i+2). The module should be able to achieve full throughput, i.e., it should be able to accept new inputs every cycle and produce a valid output every cycle. Here is an example execution trace. An X indicates that the output is undefined because there is no reset signal for the pipeline registers.

cycle	ļ	in0	in1	in2	out01	out
0	1	00	00	00	XX	XX
1	İ.	01	02	04	00	XX
2	Î.	02	03	04	03	00
3	1	03	04	05	05	07
4	1	00	00	00	07	09
5	1	00	00	00	00	0c
6	1	00	00	00	00	00

Assume all sequential logic is triggered on the positive edge of the clock.

AI in Computer Engineering • ECE Retreat

PyHDL-Eval Results: ICL for Verilog & PyMTL3

	Num	Code				
	ICL	Gemma	Liama3	Llama3		GP1
DSL	Ex	7B	8B	70B	GPT4	Turk
Verilog	0	14.9%	21.5%	57.5%	60.2%	71.0
Verilog	1	26.6%	33.0%	56.7%	62.4%	70.1
Verilog	2	32.2%	34.7%	62.4%	69.3%	72.7
Verilog	3	32.7%	36.6%	63.2%	69.9%	72.2
PyMTL3	0	0.0%	0.0%	0.6%	13.2%	2.0
PyMTL3	1	20.0%	15.7%	27.3%	33.1%	33.2
PyMTL3	2	23.5%	14.3%	31.9%	43.5%	39.2
PyMTL3	3	23.9%	15.9%	33.0%	47.0%	43.6

- Up to three ICL examples
 - 8-bit combinational incrementer
 - 8-bit registered incrementer
 - 8-bit parameterized registered increi
- ICL for Verilog
 - ICL examples help smaller models r
 - ICL examples also help LLaMA3 70
 - ICL examples do not significantly he

Al in Corr

10%

Christopher Batten



10%





Code Gemma 7B 🔺 LLaMA3 8B 📮 LLaMA3 70B 🎽

🔶 GPT4 🏾 🖕 GPT4 Turbo





 LLMs have more success on specification-to-RTL tasks targeting MyHDL vs other Pythonembedded DSLs



Ultimately, our findings do not support the original hypothesis that LLMs would be naturally more effective targeting Python-embedded DSLs

Christopher Batten

AI in Computer Engineering • ECE Retreat



PyHDL-Eval: An LLM Evaluation Framework for Hardware Design Using Python-Embedded DSLs ristopher Batten* Nathaniel Pinckney Mingjie Liu

Christopher Batten* cbatten@cornell.edu Cornell University Ithaca, NY, USA

npinckney@nvidia.com NVIDIA Corporation Santa Clara, CA, USA Mingjie Liu mingjiel@nvidia.com NVIDIA Corporation Santa Clara, CA, USA

Haoxing Ren haoxingr@nvidia.com NVIDIA Corporation Santa Clara, CA, USA Brucek Khailany bkhailany@nvidia.com NVIDIA Corporation Santa Clara, CA, USA

Abstract

Embedding hardware design frameworks within Python is a promising technique to improve the productivity of hardware engineers. At the same time, there is significant interest in using large-language models (LLMs) to improve key chip design tasks. This paper describes PvHDL-Eval, a new framework for evaluating LLMs on specification-to-RTL tasks in the context of Python-embedded domainspecific languages (DSLs). The framework includes 168 problems, Verilog reference solutions, Verilog test benches, Python test scripts, and workflow orchestration scripts. We use the framework to conduct a detailed case study comparing five LLMs (CodeGemma 7B. Llama3 8B/70B, GPT4, and GPT4 Turbo) targeting Verilog and five Python-embedded DSLs (PyMTL3, PyRTL, MyHDL, Migen, and Amaranth). Our results demonstrate the promise of in-context learning when applied to smaller models (e.g., pass rate for CodeGemma 7B improves from 14.9% to 32.7% on Verilog) and Python-embedded DSLs (e.g., pass rate for LLama3 70B improves from 0.6% to 33.0% on PyMTL3). We find LLMs perform better when targeting Verilog as compared to Python-embedded DSLs (e.g., pass rate for GPT4 Turbo is 72.2% on Verilog and 29.8-62.0% on the Python-embedded DSLs) despite using a popular general-purpose host language. PyHDL-Eval will serve as a useful framework for future research at the intersection of Python-embedded DSLs and LLMs.

CCS Concepts

• Hardware \rightarrow Hardware description languages and compilation; • Computing methodologies \rightarrow Machine learning.

Keywords

hardware description languages, Python-embedded domain-specific languages, large language models

*Also with NVIDIA Corporation.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the fall citation on the first page. Copyrights for components of this work owned by others than the author(i) must be honored. Abstrating with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permission@acm.org. MLCAD '24, September 9-11, 2024, Salt Lake City, UT, USA © 2024 Copyright held by the owner/author(i). Publication rights licensed to ACM. ACM ISBN 979-8-4007-609-8/24(09) https://doi.org/10.1145/3507447.365948 n mingjiel NVIDIA Santa Cl Brucek Khailany

ACM Reference Format:

Christopher Batten, Nathaniel Pinckney, Mingjie Liu, Haoxing Ren, and Brucek Khailany. 2024. PyHDL-Eval: An LLM Evaluation Framework for Hardware Design Using Python-Embedded DSLs. In 2024 ACM/IEEE International Symposium on Machine Learning for CAD (MLCAD '24), September 9–11, 2024, Salt Labe City, UT, USA ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3670473.3655948

1 Introduction

Novel domain-specific languages (DSLs) can improve hardware designer productivity by introducing new execution semantics, type systems, and/or custom compilation flows [14, 27, 28, 38]. Alternatively, embedded DSLs for hardware augment a software host language with a carefully designed library to provide the illusion of a new DSL while maintaining all of the features of the host language [3-6, 32–34]. Python-embedded DSLs for hardware design and verification capitalize on Python's popularity but also its unique mix of language features (e.g. lightweight syntax, dynamic typing, reflection, metaprogramming) and ecosystem (e.g., third-party libraries, package management) [2, 11–13, 17–19, 23, 38]. Trend #1: Python-embedded DSLs are increasingly being used to improve the productivity of hardware design and verification.

Large-language models (LLMs) can also improve hardware designer productivity by enabling engineers to express their intent as a natural-language prompt and using an LLM to generate low-level design or verification code [42]. Early work identified the key challenges and opportunities in using general-purpose LLMs for chip design [7, 10]. More recently, there has been work on new LLM benchmarks and fine-tuned LLMs for Verilog RTL code completion and specification-to-RTL tasks (e.g., VerilogEval [21], RTLLM [24], VeriGen [35, 36], RTLCoder [22], RTL-Repo [1]). Agent-based approaches use this recent work as one step in an automated and iterative workflow (e.g., RTLFixer [39], AutoChip [37]). LLMs are also being used more broadly to target other HDLs [40], generate hardware accelerators [15], generate dynamic test stimulus [41], generate formal assertions [31], and serve as a general chip-design AI assistant [20]. Trend #2: LLMs are increasingly being used to improve the productivity of hardware design and verification.

This paper describes PyHDL-Eval¹, a framework designed to enable research at the intersection of these two trends. Section 2 briefly reviews the Python-embedded DSLs that are considered in

¹PyHDL-Eval is available at https://github.com/cornell-brg/pyhdl-eval

PyHDL-Eval Verilog Results on Newer Models

					\frown			
DSL	Num ICL Ex	Code Gemma 7B	Llama3 8B	Llama3 70B	Llama3.2 90B	GPT4	GPT4 Turbo	GPT4o Mini
Verilog	0	14.9%	21.5%	57.5%	76.1%	60.2%	71.0%	83.6%
Verilog	1	26.6%	33.0%	56.7%	82.4%	62.4%	70.1%	84.7%
Verilog	2	32.2%	34.7%	62.4%	84.0%	69.3%	72.7%	86.4%
Verilog	3	32.7%	36.6%	63.2%	85.0%	69.9%	72.2%	87.2%

Both open and closed large-language models are continuing to make impressive progress

because of AI hardware

Christopher Batten

Al in Computer Engineering • ECE Retreat

2023/2024 Sabbatical Key Takeaways







Al in Computer Engineering • ECE Retreat



Motivation for Taking Course

- Trends in Artificial Intelligence
- 2023/2024 Sabbatical
 - Fall 2023: Slice Lab @ UC Berkeley
 - Spring 2024: NVIDIA Research @ Santa Clara, CA

Incredibly exciting time since

hardware systems are critical to enabling AI capabilities

and

AI will transform the way we build hardware systems Hardware Systems Artificial Intelligence

Activity



Complex Digital ASIC Design

- Course goal, motivation, structure
 - What is the goal of the course?
 - Why should students want to take this course?
 - > How is the course structured?
- Activity: Evaluation of Integer Multiplier
- ASIC Design Case Studies
 - Example design-space exploration
 - Example real ASIC chips



Course Goal, Motivation, Structure

Activity

ASIC Design Case Studies

Part 1: ASIC Design Overview





Activity

ASIC Design Case Studies

20 / 46

Part 2: Digital CMOS Circuits



Course Goal, Motivation, Structure

Activity

Part 3: CAD Algorithms





Activity

ASIC Design Case Studies

Five-Week Design Project





Devices

Technology

Complex Digital ASIC Design

- Course goal, motivation, structure
 - What is the goal of the course?
 - Why should students want to take this course?
 - How is the course structured?
- Activity: Evaluation of Integer Multiplier
- ASIC Design Case Studies
 - Example design-space exploration
 - Example real ASIC chips

Fixed-Latency Iterative Multiplier Datapath





Complex Digital ASIC Design

- Course goal, motivation, structure
 - What is the goal of the course?
 - Why should students want to take this course?
 - How is the course structured?
- Activity: Evaluation of Integer Multiplier
- ASIC Design Case Studies
 - > Example design-space exploration
 - > Example real ASIC chips



Figure 1: High-level diagram of data transfer with RSA

ASIC Design Case Studies

Algorithm Design-Space Exploration

```
def mod_exp( base, exponent, modulus ):
    result = 1
    base = base % modulus
    while( exponent > 0 ):
        if( ( exponent % 2 ) == 1 ):
            result = ( result * base ) % modulus
        exponent = exponent >> 1
        base = ( base * base ) % modulus
    return result
```

```
def mont_mod_exp( base, exponent, modulus ):
```

result = 1
base = base % modulus

```
# Set up Montgomery multiplier
MontMult = MontMultiplier( modulus, ( 1 << 32 ) )</pre>
```

```
# Convert into N-residue format
result = MontMult.convert_in( result )
base = MontMult.convert_in( base )
```

while(exponent > 0):

```
if( ( exponent % 2 ) == 1 ):
    result = MontMult.multiply( result, base )
```

exponent = exponent >> 1
base = MontMult.multiply(base, base)

```
# Convert out of N-residue format
result = MontMult.convert_out( result )
```

return result

Baseline Software Implementation

```
unsigned int mont_mod_exp( unsigned int base, unsigned int exponent, unsigned int modulus )
{
    unsigned int curr_base = base % modulus;
    unsigned int curr_result = 1;
    montmult_t multiplier;
    montmult_construct( &multiplier, ( unsigned long long ) modulus, 0x100000000 );
    // Convert into N-residue format
    curr_base = montmult_convert_in( &multiplier, curr_base );
    curr_result = montmult_convert_in( &multiplier, curr_result );
    // Perform iterative exponentiation
    while( exponent > 0 )
    ł
        if( ( exponent & 1 ) == 1 ) // LSB is 1
            curr_result = montmult_multiply( &multiplier, curr_result, curr_base );
        exponent = exponent >> 1;
        curr_base = montmult_multiply( &multiplier, curr_base, curr_base );
    // Convert out of N-residue format
    curr_result = montmult_convert_out ( &multiplier, curr_result );
    return ( unsigned int ) curr_result;
}
```

ASIC Design Case Studies

Baseline Hardware Implementation



Figure 4: Modified Processor Datapath to support divu, remu, mulhu, and lbu,

Course Goal, Motivation, Structure

ASIC Design Case Studies

Alternative Naive Design



Course Goal, Motivation, Structure

ASIC Design Case Studies

Alternative Optimized Design



ECE 6745

Course Overview

ASIC Design Case Studies

ASIC Implementations



(a) Amoeba plot of our ProcXcelNaive system



(b) Amoeba plot of our ProcXcelMont system

Area vs. Performance Design-Space Exploration



- Optimized accelerator improves performance by over 20× but increases area by 4.4×
- Optimized algorithm for encryption *does not* improve performance of software baseline, but *does* improve performance of hardware accelerator

Energy vs. Performance Design-Space Exploration



Accelerators have higher power because they do more work per cycle, but they also finish sooner such that the overall energy/task is less



Complex Digital ASIC Design

- Course goal, motivation, structure
 - What is the goal of the course?
 - Why should students want to take this course?
 - How is the course structured?
- Activity: Evaluation of Integer Multiplier
- ASIC Design Case Studies
 - Example design-space exploration
 - Example real ASIC chips

BRG Contributions to Various Chip Tapeouts



Course Goal, Motivation, Structure

ASIC Design Case Studies

BRG Test Chip 1 (2016)



Post-Silicon Evaluation Strategy

The testing platform enables running small test programs on BRGTC1 to compare the performance and energy of pure-software kernels versus the HLS-generated sorting accelerator



Taped-out Layout for BRGTC1

2x2mm 1.3M transistors in IBM 130nm RISC processor, 16KB SRAM HLS-generated accelerators Static Timing Analysis Freq. @ 246 MHz

Celerity w/ UCSD, UW, Michigan (2017)

Target Workload: High-Performance Embedded Computing

- 5×5 mm in TSMC 16 nm FFC
- 385 million transistors
- 511 RISC-V cores
 - 5 Linux-capable Rocket cores
 - 496-core tiled manycore
 - 10-core low-voltage array
- 1 BNN accelerator
- 1 synthesizable PLL
- 1 synthesizable LDO Vreg
- 3 clock domains
- 672-pin flip chip BGA package
- 9-months from PDK access to tape-out



BRG Test Chip 2 (2018)



Block Diagram 4xRV32IMAF cores with "smart" sharing L1\$/LLFU, synthesizable PLL



Taped-out Layout for BRGTC2

2x2mm, 1.2M-trans, IBM 130nm Static Timing Analysis Freq. @ 500MHz

ASIC Design Case Studies

CIFER w/ Princeton (2020)



ASIC Design Case Studies

BRG Test Chip 3/4 (2020/2021)



Course Goal, Motivation, Structure

Activity

ASIC Design Case Studies

BRG Test Chip #5 (2022)





ECE 6745

Course Overview

 $(\cap \cap A$

42 / 46

SPI minion for config; SPI master and GP I/O for peripherals

BRG Test Chip #5 (2022)


Course Goal, Motivation, Structure

ASIC Design Case Studies

BRG Test Chip #5 (2022)



Simulated and Measured Energy per Instruction at 66 MHz and 3.3 V Core Voltage



Fall 2022

PIPES EIC w/ Columbia (2023)



Fig. 1. Approaches to Tightly Integrated Optical Interconnects



Fig. 2. Hybrid 2.5D/3D Packaged System





Circuits

Devices

Technology

Take-Away Points

- ASIC design is process of quantitatively exploring the area, cycle time, execution time, and energy trade-offs of general-purpose and application-specific designs using automated standard-cell CAD tools and then to transform the most promising design to layout ready for fabrication
- Incredibly exciting time since hardware systems are critical to enabling AI capabilities, and AI will transform the way we build hardware systems
- Course provides an excellent foundation for students interested in pursuing a industry career in ASIC development or research in acceleratorcentric system design