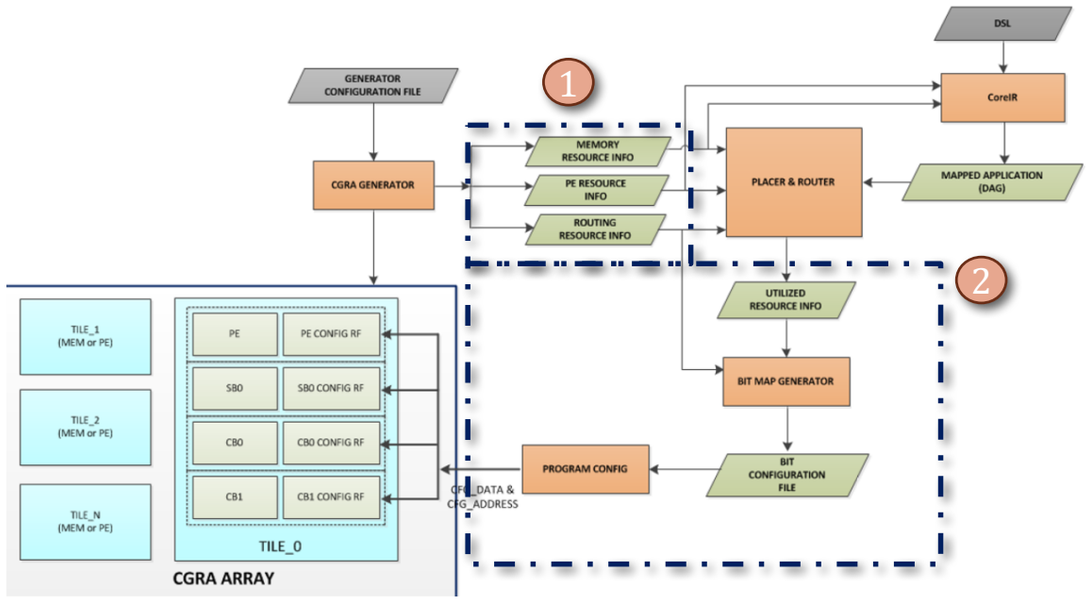
Alex Carsello

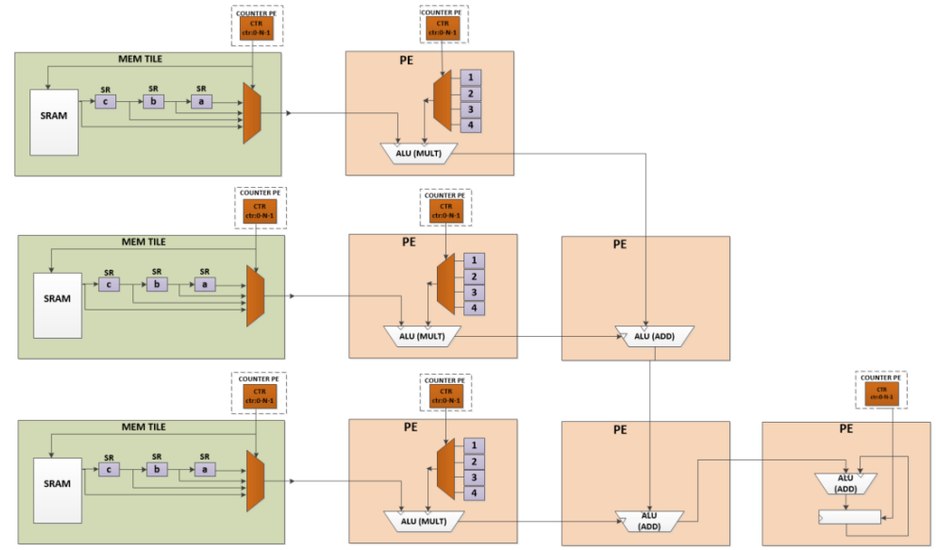
- Worked on hardware for configuration and debugging in CGRA
  - JTAG and global controller
- Post-synthesis/P&R verification
- Currently investigating ways to better leverage generators for:
  - Design space exploration
  - Guiding physical design
  - Design Reuse
- How do we structure these generators? Which “knobs” should we provide?

Ankita Nayak – Advised by Mark Horowitz



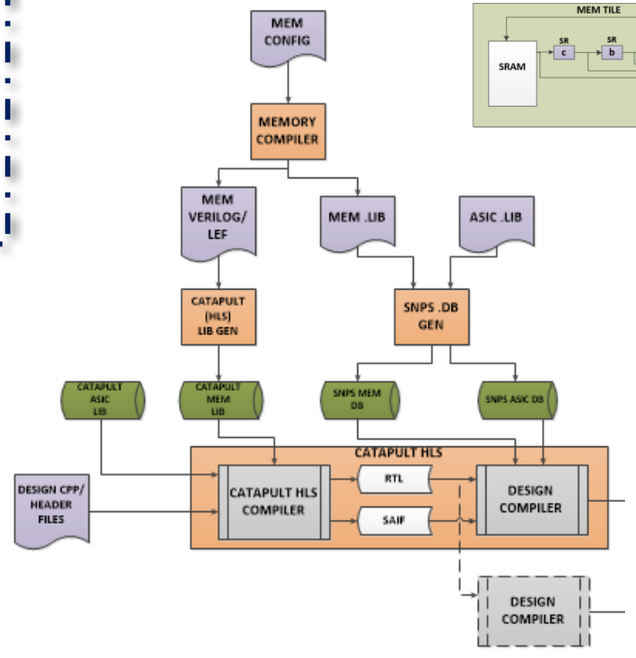
CGRA

- CGRA generator interface with CoreIR & P&R
- 1st bitstream generator for CGRA
- Genesis SB & CB coding & Tapeout Spec
- CGRA Tapeout planning
- Atrenta Spyglass based Design-Rule Lint Checks..



Hardware Virtualization

- Fitting application size > available HW resources
- Multi-cycle operation support
- Resource-constrained mapping
- FSM based design



Deep Neural Network Dataflow Evaluation

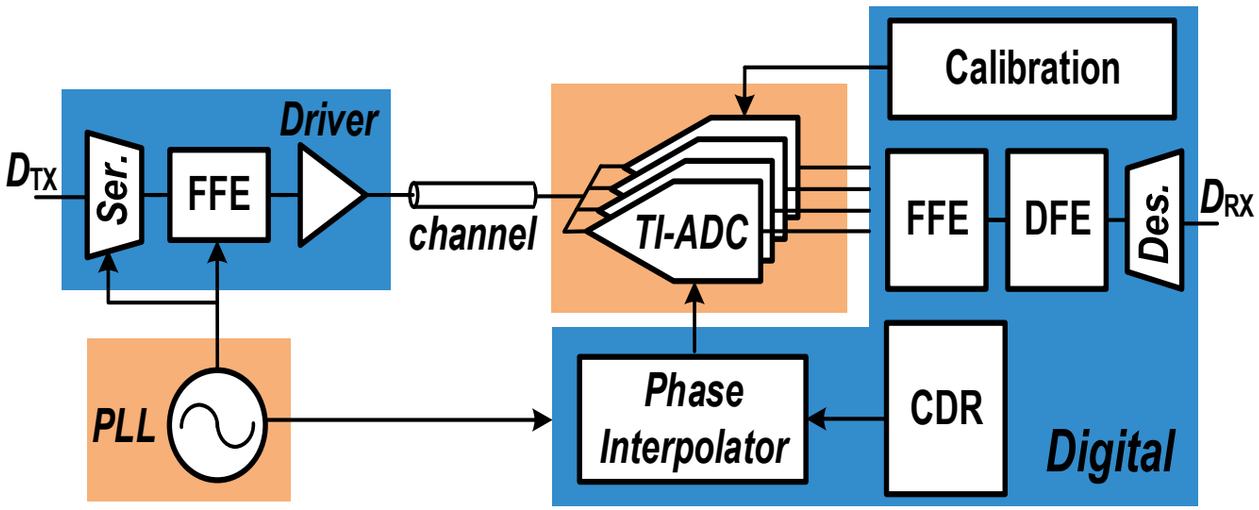
- Catapult HLS-RTL-Synthesis based ASIC design space exploration for different DNN architectures and dataflow (E.g. TPU, Eyeriss etc.)

Stanford University

Open Source PHY

Sung-Jin Kim

- **Motivation**
  - Nearly all SoCs have high-speed I/Os, but design complexity and thus NRE cost of building PHY are high.
- **Goal**
  - To build a robust, portable(***synthesizable***) PHY
  - To build an associated validation framework for quick and thorough validation of the design



<ADC based Serial Link>

- **Task 1: Framework for system-level validation of mixed-signal SoCs**
  - Create a standard analog cell template library
  - Each cell template generates both event-driven functional models and test routines

- **Task 2: Robust, portable PHY design**
  - The design is to fully leverage digital design/verification methodology
  - Our basic architecture is a baud-rate link with time-interleaved ADCs
  - ***Analog blocks are built upon digital std cells*** and use digital P&R flow to automate the layout

CGRA Energy and Area Efficiency

Energy Efficiency
pj/op

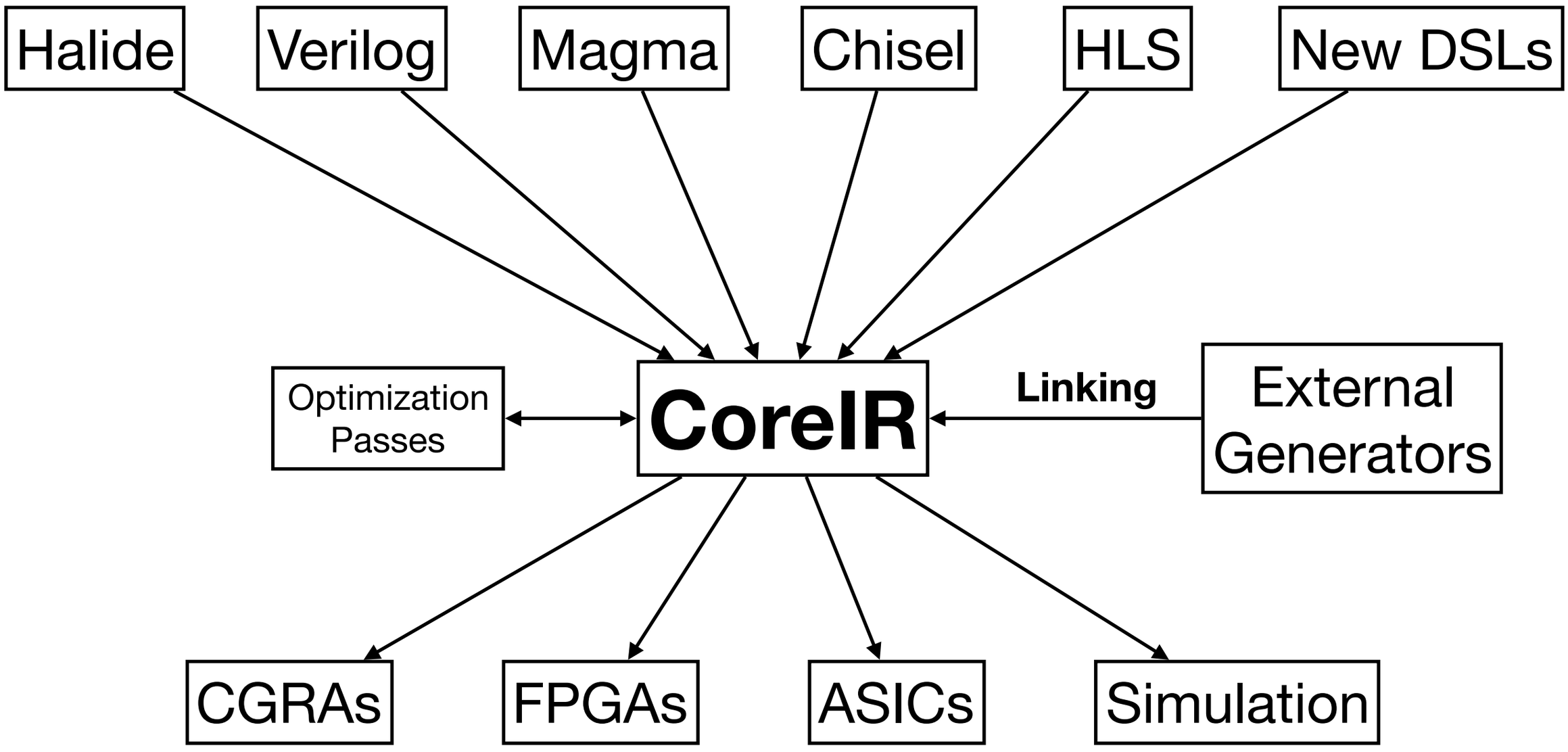
	CGRA	SIMD	FPGA+pw	ASIC
Harris	1.38	2.08	3.69	0.53
Fast	3.91	6.27	3.19	0.52
ISP	0.91	1.68	1.12	0.20
FCAM	1.04	1.41	1.42	0.15
Stereo	1.30	1.94	3.19	0.13
Average	1.71	2.68	2.35	0.31
Peak	0.38	0.76	1.35	0.03

Area Efficiency
mm^2 / GigaOp / s

	CGRA	SIMD	FPGA+pw	ASIC
Harris	.013	.013	.039	.0046
Fast	.034	.046	.032	.0038
ISP	.007	.010	.016	.0011
Fcam	.006	.009	.017	.0006
Stereo	.006	.008	.024	.0004
Average	.013	.017	.026	.0021

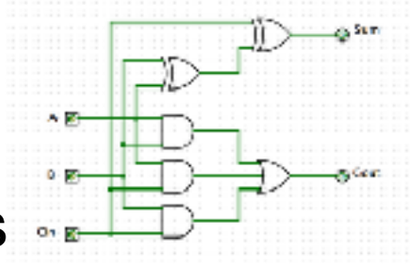
- Enhance usability by compiling down from a DSL
- Maintain energy and area efficiency
- CGRA, on average
  - 40% more energy efficient
  - 30% more area efficient
  - 5X worse than ASIC
- Heterogeneity is the key
  - Tiles of varying complexities
  - Route density can vary across fabric

CoreIR: The LLVM of Hardware



Magma: Python Ecosystem for Hardware Design

An open source suite of productivity tools focused on rapid prototyping, reuse, and flexibility.



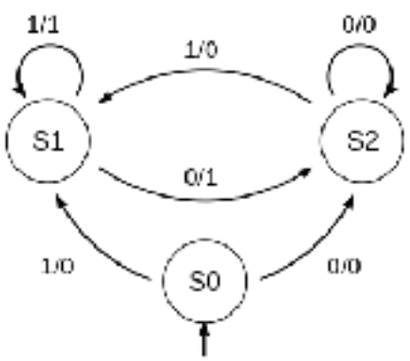
magma - Embedded HDL for constructing synthesizable circuits

mantle - Standard library of useful magma circuits

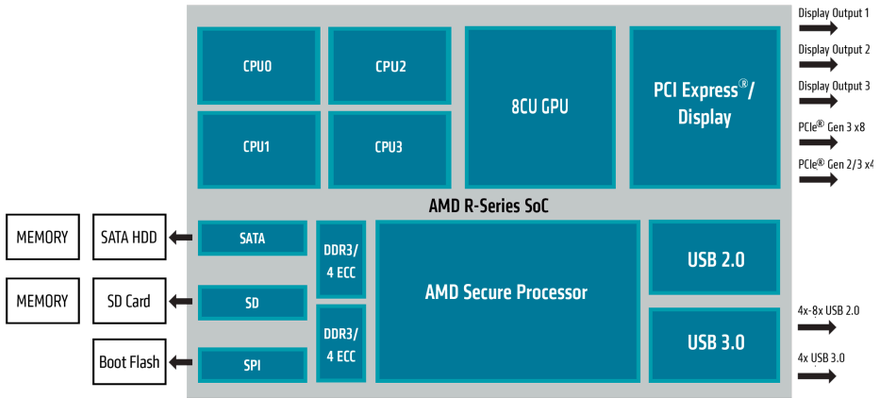
loam - Library for peripherals, parts (ICs), and boards (PCBs)

fault - Magma extensions for functional verification

silica - Magma extensions for finite state machines

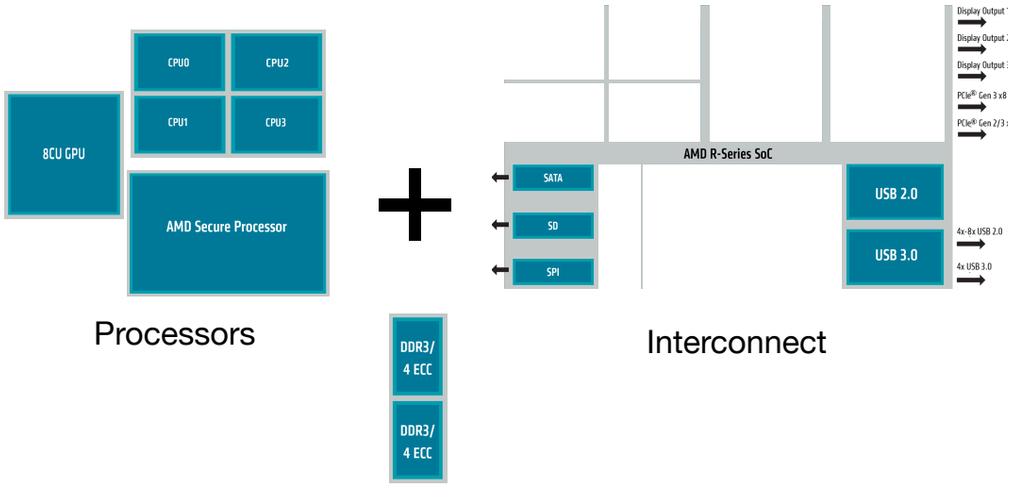


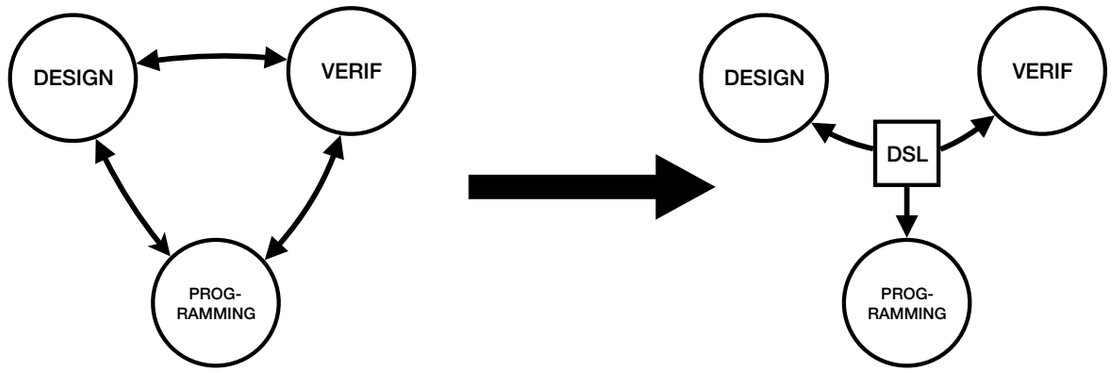
pycoreir - Python bindings for CoreIR hardware compiler infrastructure



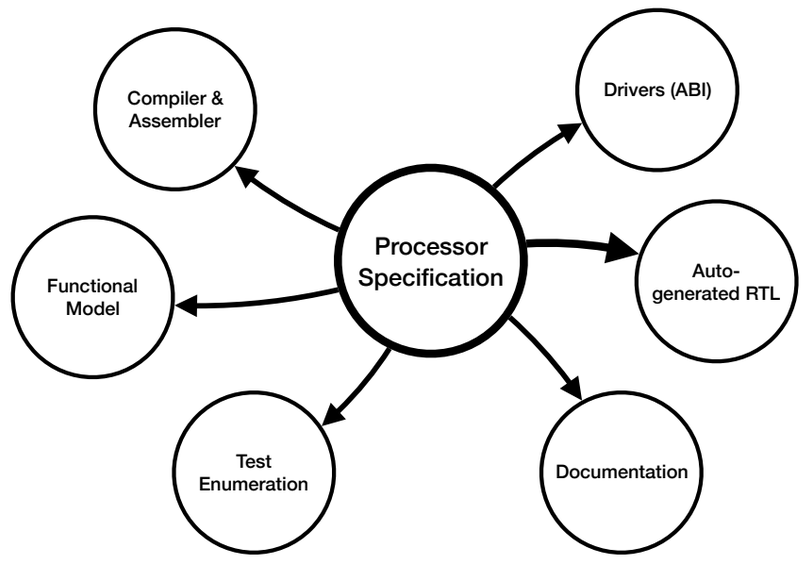
AMD SOC [https://www.amd.com/en/products/embedded-r-series-soc]

- **Decompose SOC**'s into sub-parts
- Specify each sub-part independently
- **Wire up parts** to build full SOC



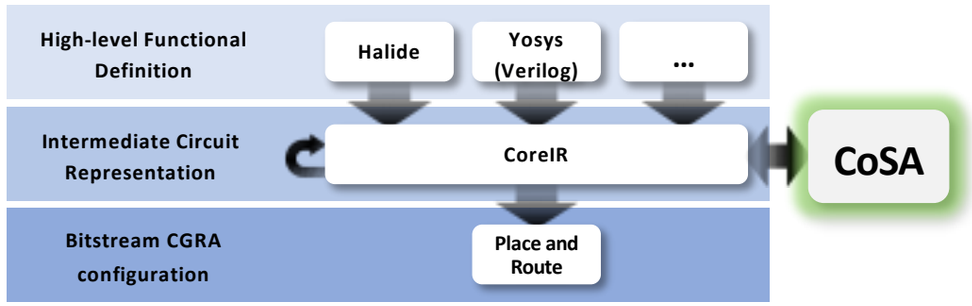


- Construct a **specification DSL** for each sub-part (e.g. PE's)
- This specification is the **ground truth** - all other collateral can be derived from this single specification



Stanford University

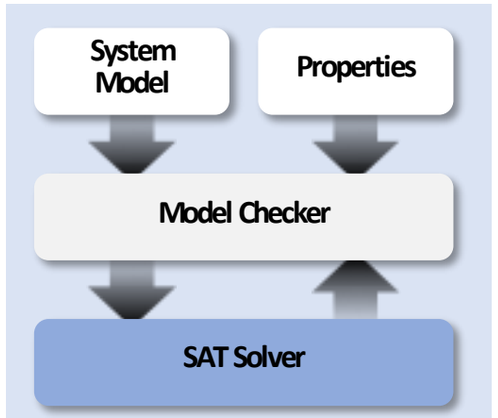
Integration with AHA flow and functionalities

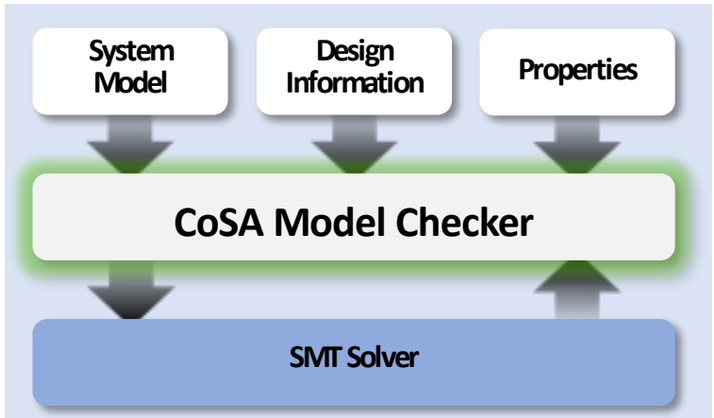


Functionalities of CoSA

- **Safety Verification:** standard invariant verification with proving capabilities
- **LTL Verification:** support for all temporal operators, with specialized algorithms
- **Equivalence Checking:** synchronous product between systems under analysis and reduction to safety verification
- **Lemma Verification:** automated check if the model satisfies the lemmas, and integration with the verification part
- **Clock Abstraction:** skips neg-edge steps in case of system with only pos-edge registers
- **Synchronous Input Models:** multiple models are combined into a synchronous product in order to simplify the analysis
- **Automated Lemma Extraction:** interaction with CoreIR in order to improve the equivalence checking performance

New approach to Model Checking





Model Checking (Commercial tools)

- Model and Properties as single entry point to the model-checker
- Bitblasting loses word-level structure
- Black-box usage of SAT solver
- Closed source

Our Approach - CoreIR Symbolic Analyzer (CoSA)

- Can make use of additional design information to optimize verification
- Maintains word-level structure by using SMT theories
- Uses customizable SMT solver
- Entire toolchain is open-source

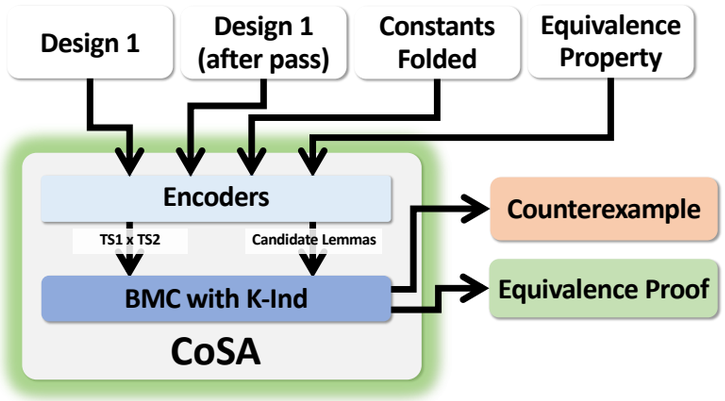
Stanford University

Verification: Global Controller

Property	Result
Always return to ready state assuming counter delay < 10	T
When not in ready state, the counter always decreases	T
No underflow in counter	F
Read signal high implies the global controller is in read state	T
Write signal high implies the global controller is in write state	F

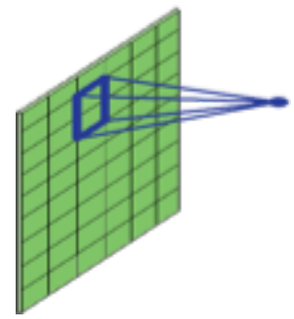
- Global controller handles configuration and debugging for a Coarse Grained Reconfigurable Array
- Implemented in Verilog
- Translated to CoreIR through the open-source VerilogToCoreIR plugin pass for Yosys

Equivalence Checking: Fold-Constants



- Comparison of a CoreIR design before and after optimization pass
- Automated lemma extraction from constants information
- Significant performance improvement: 1.3 min vs. T.O. (2 h)

Verification: 2x1 Convolution



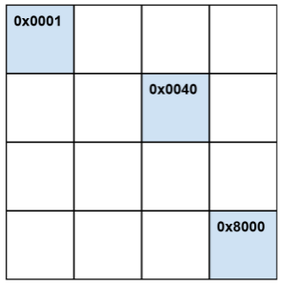
- 4x4 CGRA: 78,924 variables (including arrays and Bitvectors), and a total of 443,376 bits and 4 arrays
- Application: 2x1 Convolution, Mapped to CGRA Primitives, with bitstream produced by Place and Route
- Memories encoded using the SMT Theory of Arrays
- CGRA implements linebuffer with 2 memories and complex logic
- Load bitstream into CGRA through CoSA using an Explicit State Synchronous System
- Verify Sequential Equivalence between original CoreIR application file and the configured CGRA: Equivalent in all executions up to 20 cycles (10 cycles of valid output)
- Barriers to inductive proof: Equivalence is not inductive, and cannot strengthen property with Array equivalence because different number/size of arrays

Stanford University

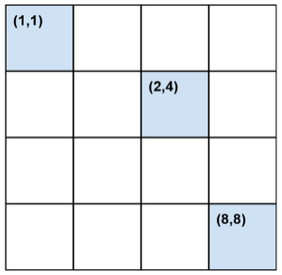
PnRDoctor

Placement constraints

- Each element is placed on fabric
- No two elements are placed in the same spot
- Connected elements are “close”
  - Nearest neighbor
  - Within neighborhood
  - Half-perimeter wirelength
- Elements are placed on matching resource



One-hot Bitvector encoding



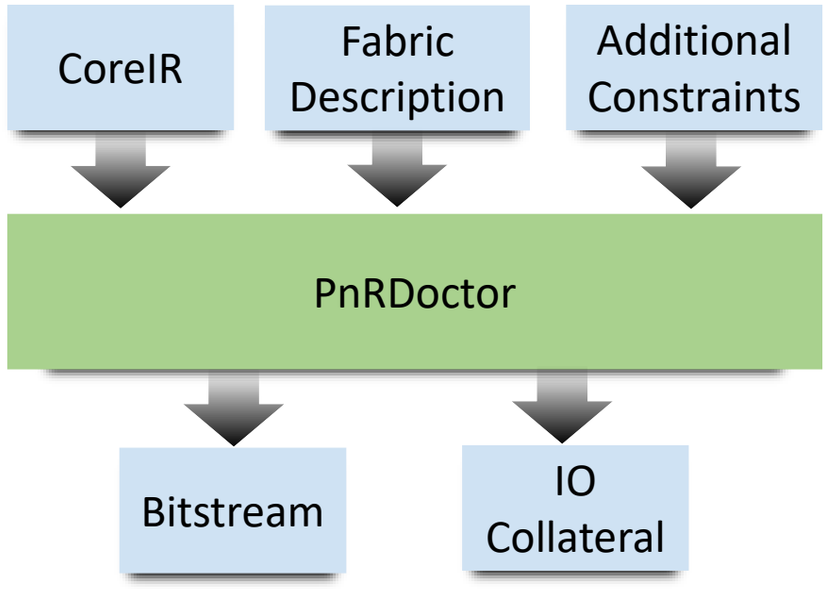
One-hot x, One-hot y

Optimizations

- Exploit column structure of resources
- Distinctness using uninterpreted functions
  - Assign each module a unique ID
  - distinctFunction(x, y) = ID

PnRDoctor

- SMT-based place-and-route tool
- Easily extended Python implementation
- Supports arbitrary heterogeneous fabrics with an XML description
- Decomposes word-level and bit-level routing
- Configuration file for pinning IOs and other additional constraints
- Produces bitstream for CoreIR application file



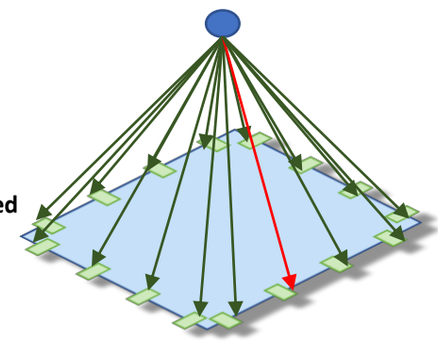
Routing constraints

- Connected components in design are connected via switch boxes on the fabric
- Each wire segment has at most one driver
- Connected elements are “close”
  - Manhattan distance
  - Bound inferred based on placement

Graph-Aware SMT Solver MonoSAT

- Combines graph algorithms with SMT solver
- Exploits *monotonicity* of graph properties
  - Reachability
  - Shortest path
  - etc…

Virtual MonoSAT node connected to all registers in a switchbox

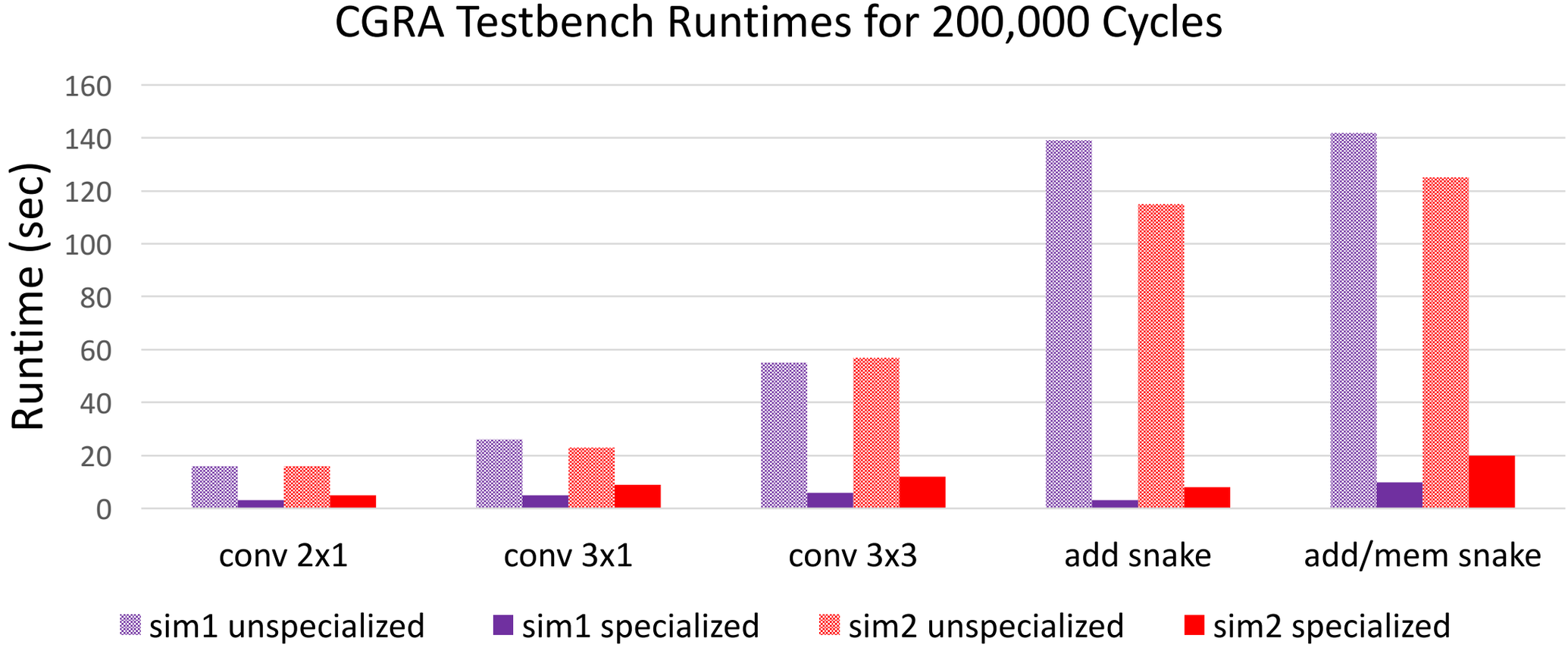


Placement as Routing

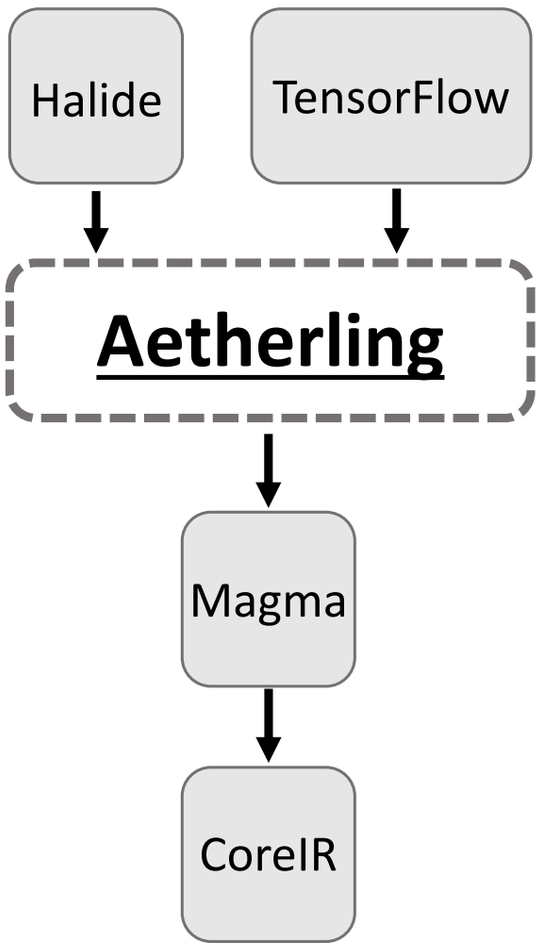
- Absorbs some placement constraints in routing
- Add virtual nodes and assert constraints on them
- Assert that only one edge is used (placed in one location)
- Currently used for placing registers

One Simple Trick to Accelerate CGRA Simulations by an Order of Magnitude

- Problem: Statically programmable datapaths are slow in conventional hardware simulators
- Solution: just in time (JIT) compilation

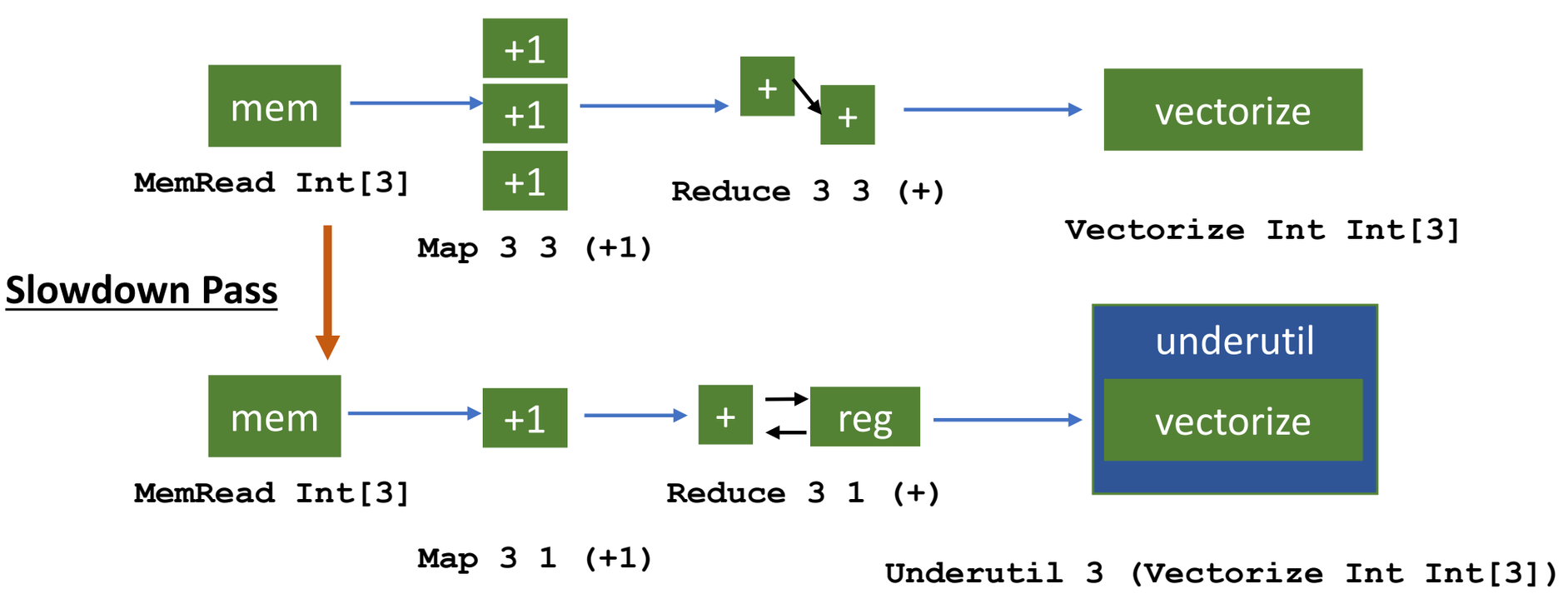


Aetherling: Defining The Space-Time Tradeoffs For Pipelines of Data Parallel Operators



Goals:

1. Collection of standard data parallel operators (map, reduce, fold) with:
   1. Hardware implementations
   2. Models of their space and time resource requirements
2. Compiler passes that tradeoff space, time, and utilization



Jeff Setter

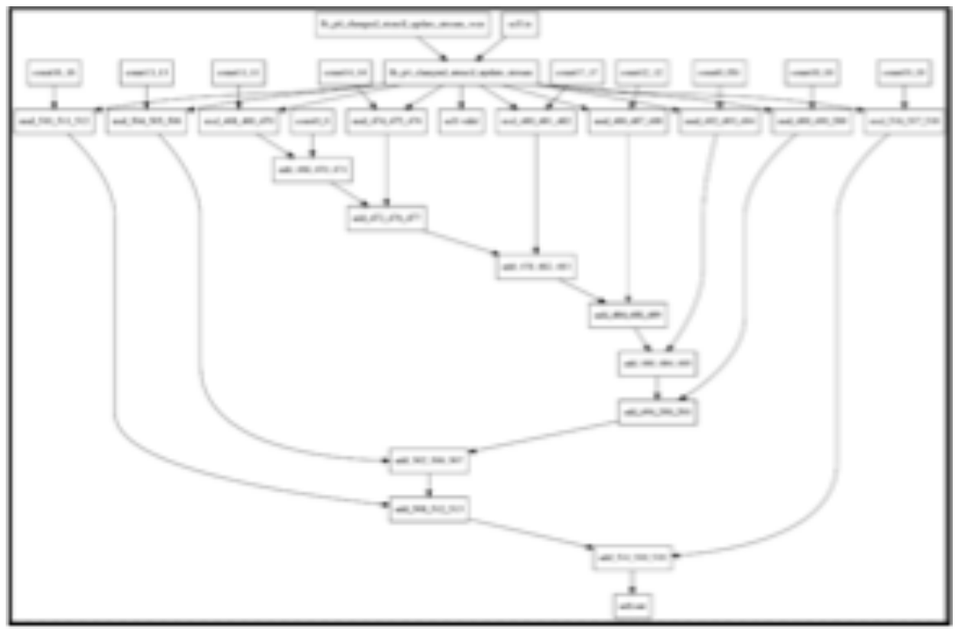
Halide Source Code

```
Var x, y, xi, yi, xo, yo; Func conv, out;
RDom win(0,3, 0,3);
kernel(x,y) = {{11,12,13},{14,15,16},{17,18,19}};

// algorithm
conv(x, y) += input(x+win.x, y+win.y) *
                kernel(win.x, win.y);
out(x, y) = conv(x, y);

// schedule
conv.update(0).unroll(win.x).unroll(win.y);
out.tile(x,y, xo,yo, xi,yi, 62,62).reorder(xi,yi, xo,yo);
conv.linebuffer();
```

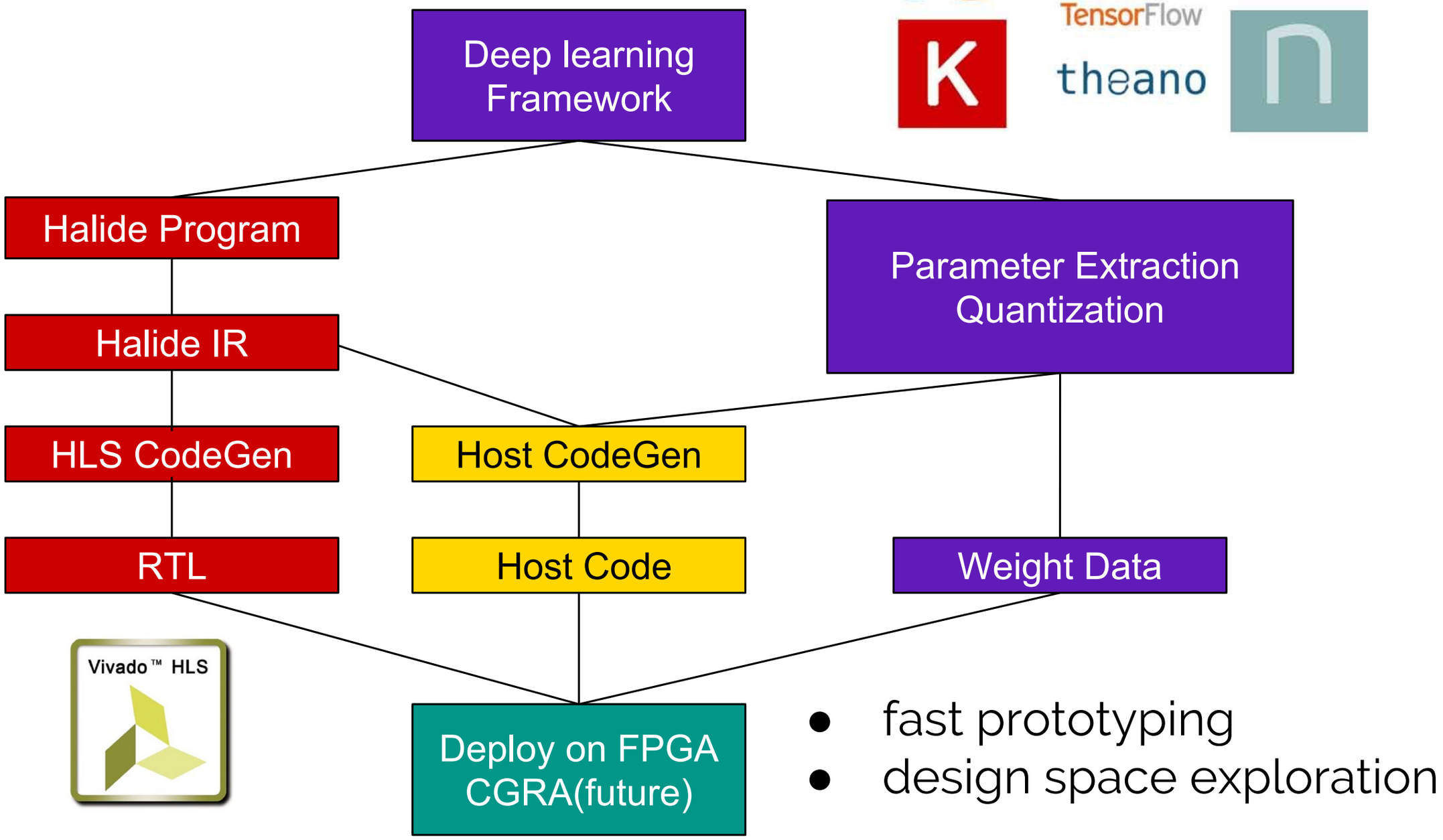
Generated CoreIR Circuit



Short Term Plan:

- Larger application suite
- Transfer Halide metadata to CoreIR

DNN HW Generation



- fast prototyping
- design space exploration

A Language and Platform for Massively Parallel Streaming on FPGAs

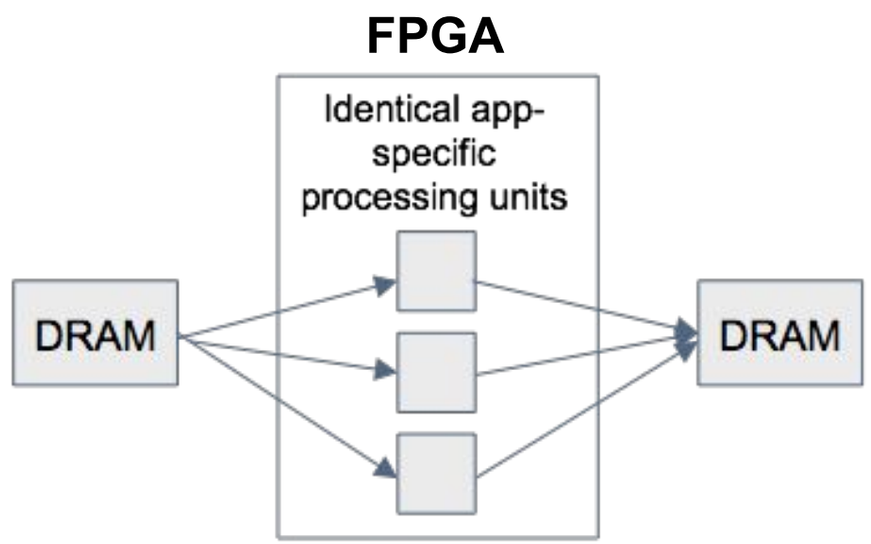
James Thomas, Matei Zaharia, Pat Hanrahan

Stream processing unit to compute histograms:

```
reg counter(bitWidth = 8, init = 0)
bram histogram(numElts = 8, bitWidth = 9)
reg hist_idx(bitWidth = 4, init = 0)

if (counter == 0) {
    pre_while (hist_idx < 8) {
      emit(histogram[hist_idx])
      histogram[hist_idx] = 0
      hist_idx += 1
    }
    hist_idx = 0
}
histogram[input] += 1
counter += 1 // wraps around
```

Compiler



~7x throughput/watt over V100 GPU for JSON parsing and integer compression

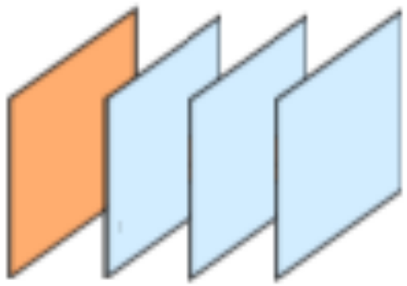
As many units as can fit on chip

Zero-cost control-flow divergence

Exploit Temporal Redundancy in Video: Efficiency? *Yes!* Accuracy? *Yes!*

Push efficiency-accuracy trade-off forward

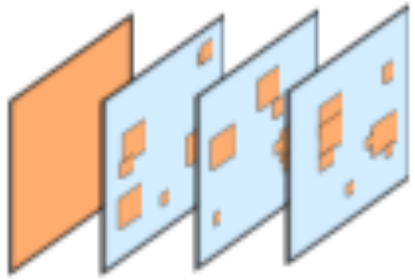
1. Temporally partial update



- Key frame scheduling
- Feature flow

+

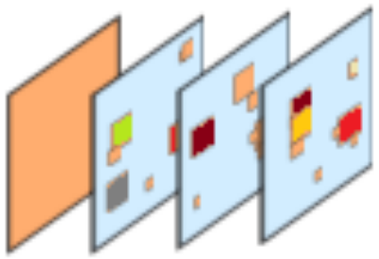
2. Spatially partial update



- Adaptively select regions
- Otherwise, feature flow

+

3. Architectural partial update



- Adaptively choose depth
- Otherwise, feature flow

Application

- Mainly for real-time, limited-power embedded system: *e.g. autonomous driving, IoT*
- But, can also be applied to all video related applications*: e.g. video action recognition, video classification*

Autoscheduling Deep Neural Networks for Hardware

- From Halide to FPGA/ASIC
- Optimizer
  - blocking
  - dataflow
  - memory hierarchy