Creating An Agile Hardware Flow

Stanford AHA! Agile Hardware Center

Stanford | ENGINEERING

August 18, 2019

How hardware design is done today

Study APP and DSE

Design hardware

Write software

How hardware design is done today

Verilog, VHDL, SystemVerilog...

How hardware design is done today

Study APP and DSE

Design hardware

Write software

This is a waterfall approach to design

We have a linear staged design flow:

1. create a specification,
2. hardware design,
3. software design.

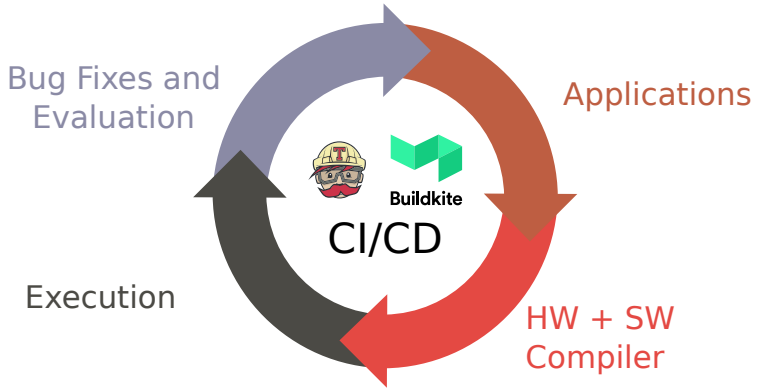
Problems:

- Change of application requirements.
- Incomplete knowledge/understanding of the problem.

Is there an agile approach to hardware design?

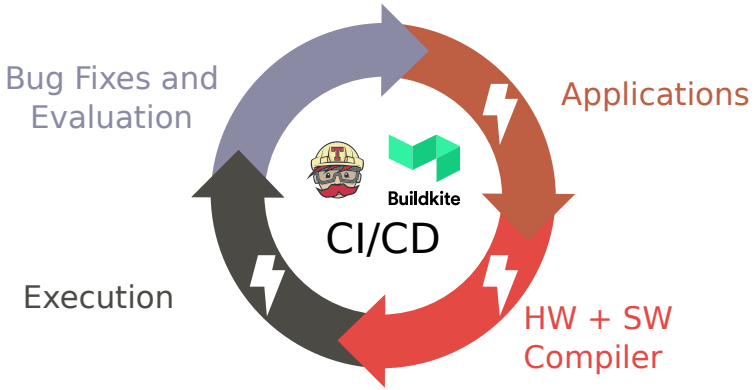
Requirements of an agile hardware/software design approach

- An end-to-end system supporting continuous integration/deployment
  - From applications to a running hardware/software system.
- Evolve that system to make it more efficient.



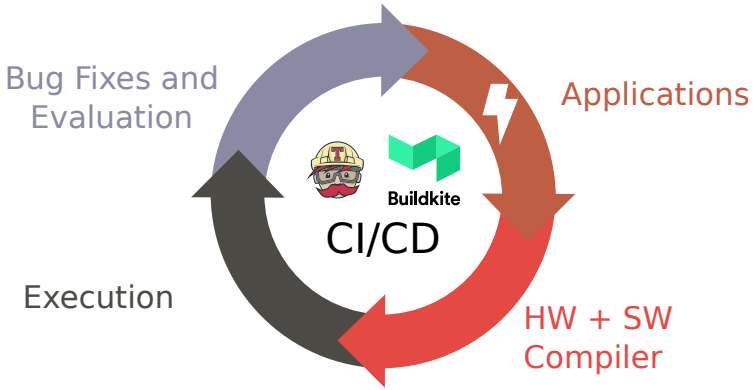
Requirements of an agile hardware/software design approach

- An end-to-end system supporting continuous integration/deployment
  - From applications to a running hardware/software system.
- Evolve that system to make it more efficient.



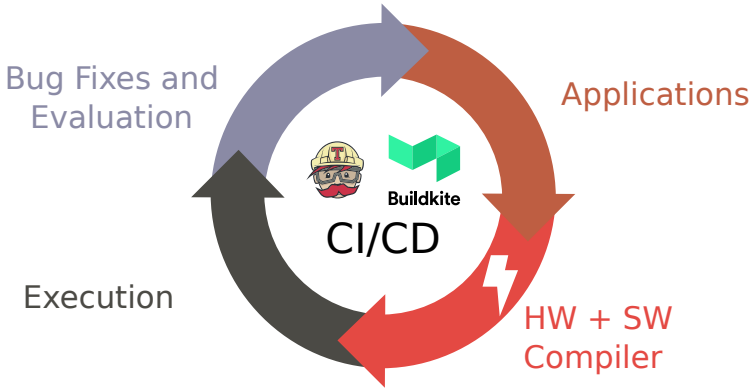
Requirements of an agile hardware/software design approach

- An end-to-end system supporting continuous integration/deployment
  - From applications to a running hardware/software system.
- Evolve that system to make it more efficient.



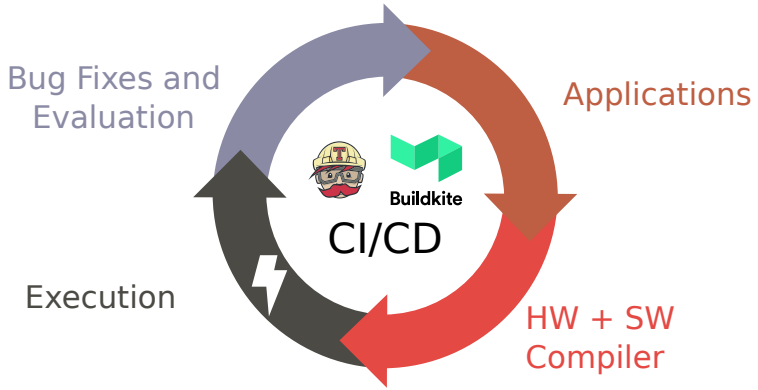
Requirements of an agile hardware/software design approach

- An end-to-end system supporting continuous integration/deployment
  - From applications to a running hardware/software system.
- Evolve that system to make it more efficient.



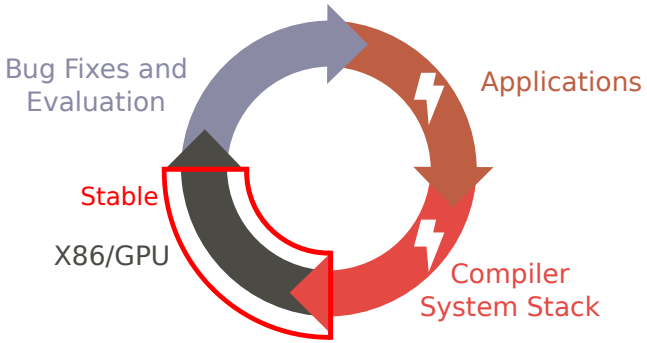
Requirements of an agile hardware/software design approach

- An end-to-end system supporting continuous integration/deployment
  - From applications to a running hardware/software system.
- Evolve that system to make it more efficient.



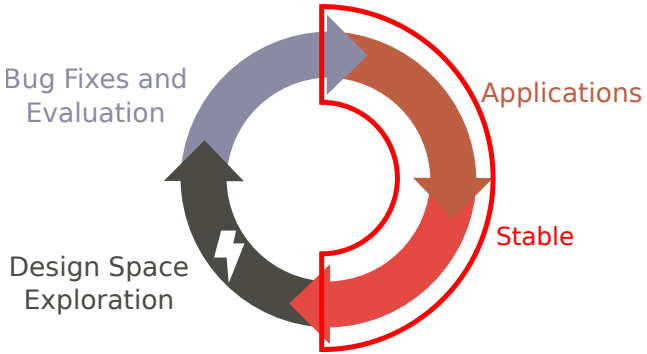
Evolving software applications

- With a fixed hardware, we can continuously optimize the applications
- And the compiler.
- Problem: we don't have fixed hardware when designing accelerators in an agile approach.

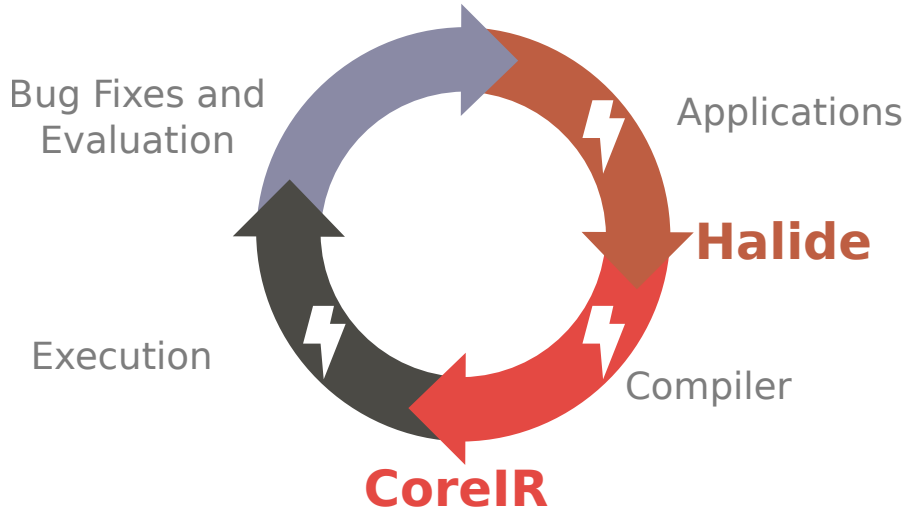


Design space exploration

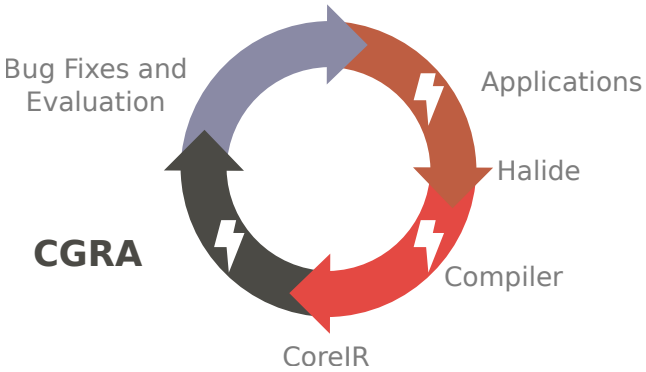
- If we have a fixed set of applications.
- Extract key application parameters.
- Problem: applications and compilers are always changing to be more performant.

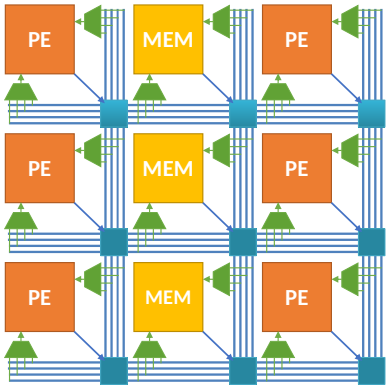


A new agile hardware/software design: stable interface



A new agile hardware/software design: CGRA

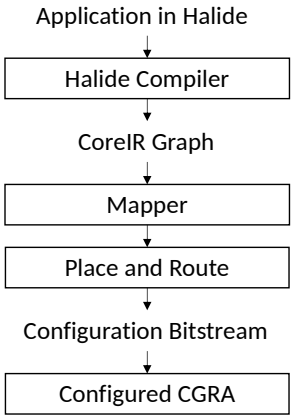




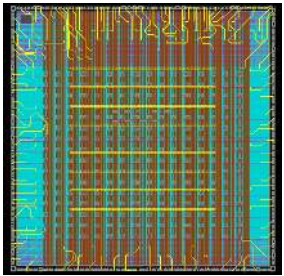
Coarse-Grain Reconfigurable Architecture (CGRA).

Jade: Our first generation of CGRA

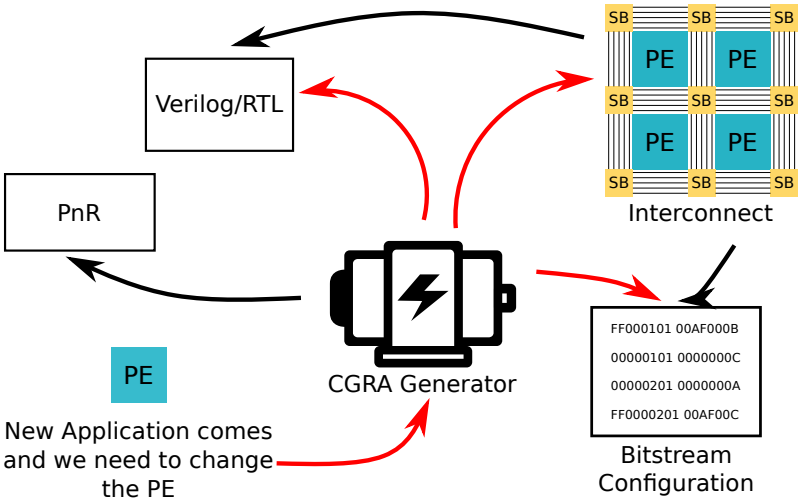
CGRA Programming Toolchain



1. Finished in a year with a group of grad students.
2. Built with **Genesis2**, a hardware generation framework that uses Perl to meta-program hardware modules written in SystemVerilog.
3. Taped out in Summer 2018 and we received packaged parts in January.



Problem: changes affect many tools



Maintaining the flow and collateral generation becomes more difficult

```
assign mode = config_mem[1:0];
assign tile_en = config_mem[2];
assign depth = config_mem[15:3];
assign almost_count = config_mem[19:16];
assign enable_chain = config_mem[19];

// ... 216 lines later ...

//;my $filename = "MEM".$self->mname();
//;open(MEMINFO, ">$filename") or die "Couldn't open file $filename, $!";
//;print MEMINFO "        <mode bith='1' bitl='0'>00</mode>\n";
//;print MEMINFO "        <tile_en bith='2' bitl='2'>0</tile_en>\n";
//;print MEMINFO "        <depth bith='16' bitl='3'>0</depth>\n";
//;print MEMINFO "        <almost_count bith='19' bitl='16'>0</almost_count>\n";
//;print MEMINFO "        <chain_enable bith='20' bitl='20'>0</chain_enable>\n";
//;close MEMINFO;
```

memory_core.vp

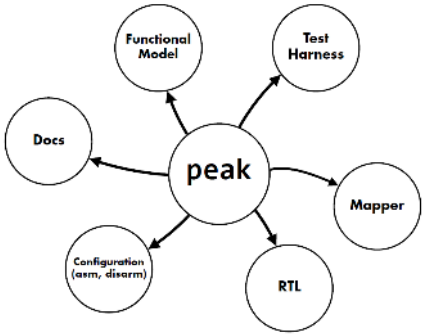
Domain-specific languages (DSLs) to the rescue

Why choose DSLs

- We want generators to be the spec of a design that produce both RTL and a consistent API for downstream tools to query. This is the single source of truth.
- Generating arbitrary hardware from arbitrary higher-level specifications is an extremely difficult problem.
- Dividing the problem into generating specific types of hardware, such as ALU and interconnect, makes the problem more tractable.

PEak: DSL for Processing Elements (PEs)

- Python-embedded DSL for describing the ISA and functional model of a PE
- Has precise formal semantics



Multiple interpretations of the same **PE**ak object

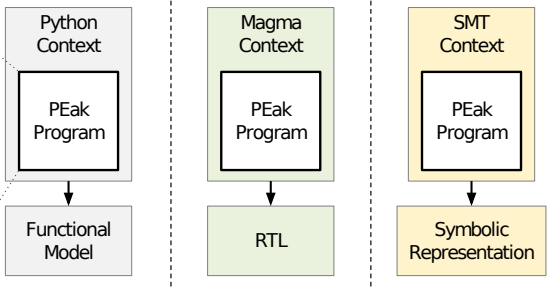
isa.py

```
class ALUOP(Enum):
    Add = 0
    Or  = 1
    And = 2
    XOr = 3
    #etc...

class Instr(Product):
    alu_op : ALUOP
    invert_in0 : Bit
    invert_in1 : Bit
    set_carry : Bit
```

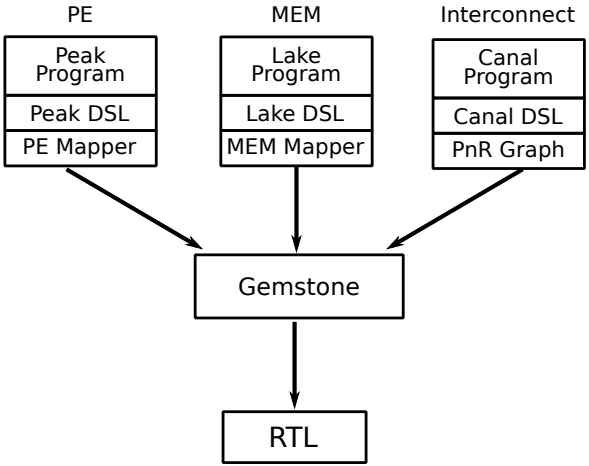
semantics.py

```
ABV = hwtypes.AbstractBitVector
class SimplePE(Peak):
    def __init__(self):
        #Declare all the state

    def __call__(self, instr : Instr,
                       in0 : ABV[16],
                       in1 : ABV[16]):
        if instr.invert_in0:
            in0 = ~in0
        if instr.invert_in1:
            in1 = ~in1

        carry = instr.set_carry
        if instr.op == Op.Add:
            res = in0 + in1 + carry
        elif instr.op == OP.And:
            res = in0 & in1
        #etc…
        return res
```

PEak Program



Overview of DSL designs in our new tool chain



Separation of concerns and stages

Naming convention

Next-generation generators

Physical implementation issues

Due to the requirement of physical design, sometimes we have to change the logical implementation.

Common solution:

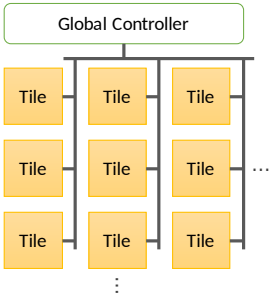
- Change how the generator produces RTL.

Result:

- The generator becomes brittle and dependent on a particular technology.
- Changes not re-usable.

Lessons learned: Generate design in stages

- Logical design goes through many "passes" to create the final design.
- Re-usable passes for other designs.

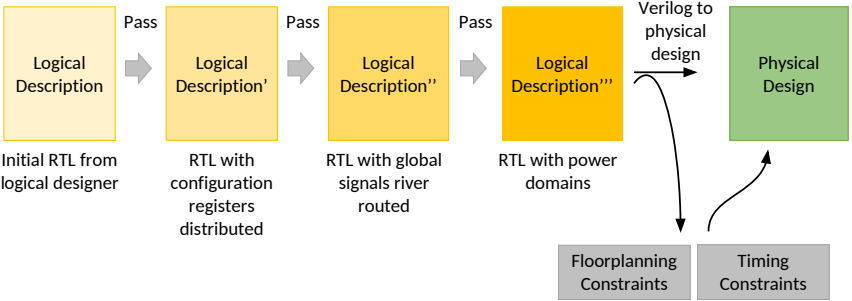


What if we want to change how the global signals are wired during physical design?

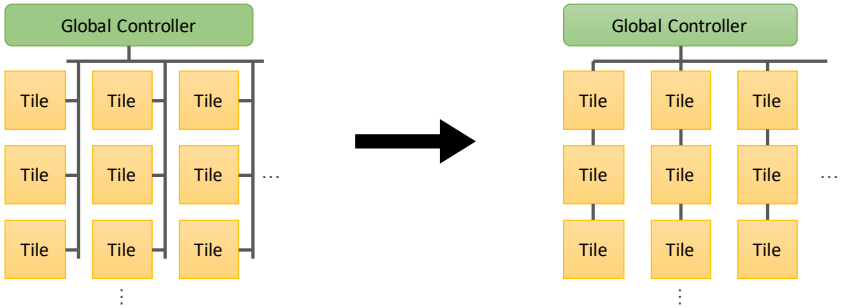
Gemstone: A staged generator

Gemstone is a structural staged generator that allows multiple passes to change the RTL design.

- Well-defined primitives on circuit objects, such as add/remove ports and instantiate generators/circuits.
- Allows multiple passes to change the RTL and generate non-RTL collateral



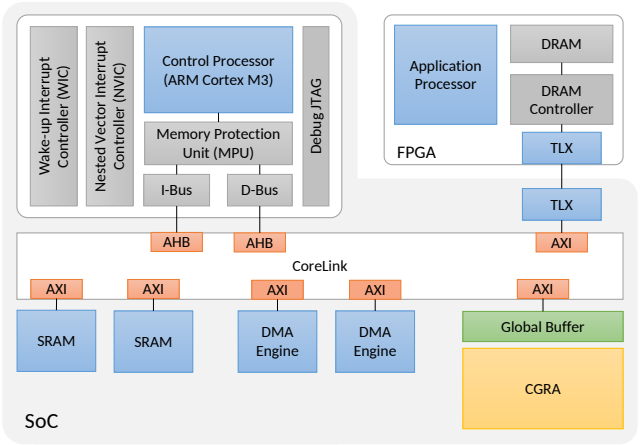
Passes to modify **Gemstone** generator objects



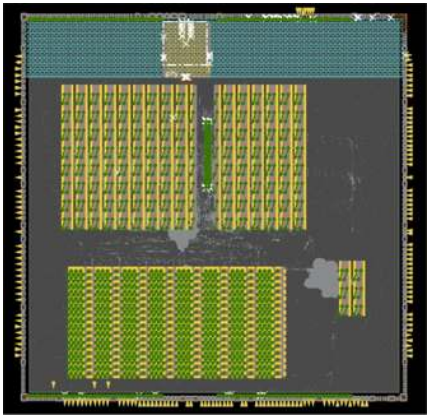
An example pass that changes the fanout global signals into a daisy chain.

Architecture diagram for **Garnet**

Garnet is more complex and has more component dependencies than the first generation **Jade** chip.



Putting everything together



Garnet SoC layout diagram

- Re-implemented the entire tool-chain from scratch in 7 months.
- Produced a tape-out ready design.
- Full-sized SoC working in RTL simulation.

Chip Spec:

- 32x16 CGRA Array
- 16-bit integer and BFloat PEs
- 4 MB second-level memory
- ARM Cortex M3 with CoreLink

Future plan

- Use DSE feedback to automatically optimize the design.
- Continuously harden the CGRA design to make it more area and power efficient.
- A Linux-capable SoC design.
- More polished tool-chain flow that tests and evaluates the system all the way through to physical design.

Agile hardware design is possible!

We produced 2 chips with complete software stacks in 2 years and we continue to evolve the hardware and software stack.

1. Clean interface between software and hardware allows both of them to be improved together.
2. DSLs for hardware generators make hardware design more tractable and easier to change.
3. Staged generators allow for a separation of concerns between logical description and physical implementation, making the system more robust to changes.

Stanford AHA! Agile Hardware Center

Contributors: Rick Bahr, Clark Barrett, Nikhil Bhagdikar, Alex Carsello, Nate Chizgi, Ross G Daly, Caleb Donovick, David Durst, Kayvon Fatahalian, Kathleen Feng, Pat Hanrahan, Teguh Hofstee, Mark Horowitz, Dillon Huff, Taeyoung Kong, Zheng Liang, Qiaoyi Liu, Makai Mann, Zachary Alexander Myers, Ankita Nayak, Aina Niemetz, Gedeon Nyengele, Priyanka Raina, Stephen Richardson, Raj Setaluri, Jeff Setter, Daniel Stanley, Maxwell Strange, Charles Tsao, James Thomas, Leonard Truong, Xuan Yang, Keyi Zhang

Special thanks to our sponsors: DARPA DSSoC program, and Intel, Facebook, Google, Amazon, and NVIDIA.

All our tools are open sourced at https://github.com/StanfordAHA and we would be happy to collaborate.