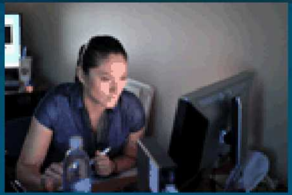
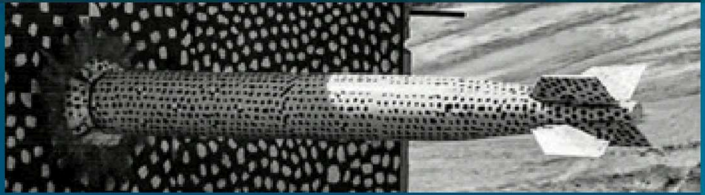
SAND2020-4970PE

SST Paths for ARIAA





Presented By:

Clay Hughes, Sandia National Laboratories

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Overview

Pursuing two paths, independent of one another

essent integration: firrtl → RTL simulation

- Emits C++ that can be compiled to make a fast simulator of the design
- Typical flow: essent to make C++ from the firrtl input → write a C++ harness for the emitted code → compile everything → run the simulation

SST dataflow component: HLL → cycle-approximate simulation

- Compile HLL to IR
- Execute host code in front-end simulator, execute device code on dataflow accelerator

SST + essent

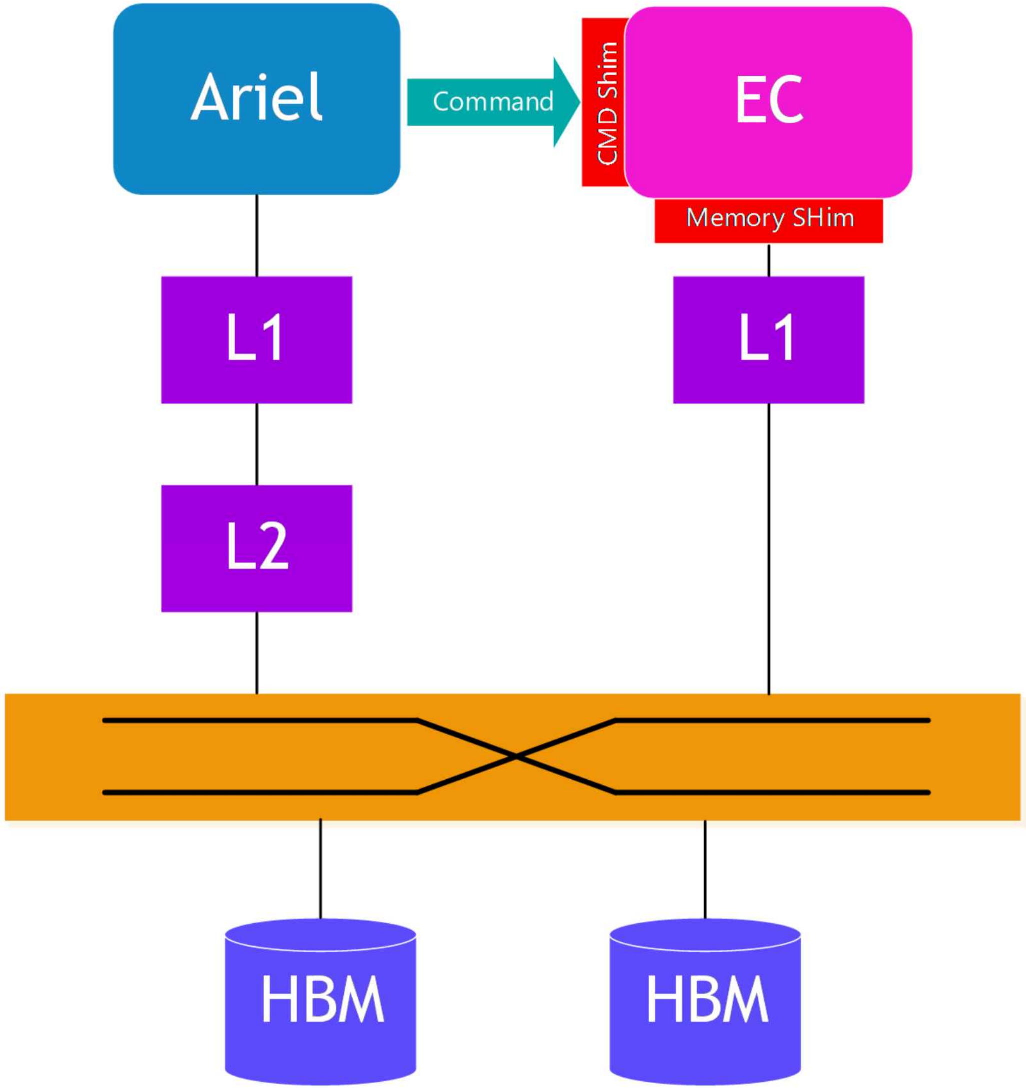
Will most likely need a to develop a set of shims to provide a common interface for other components

Pros

- Component generated directly from hardware description

Cons

- Inflexible design
  - New components require expertise hardware to generate
- Simulation is likely to be slow
- Programming model is unknown and likely to change with each component



essent SST Component Generation

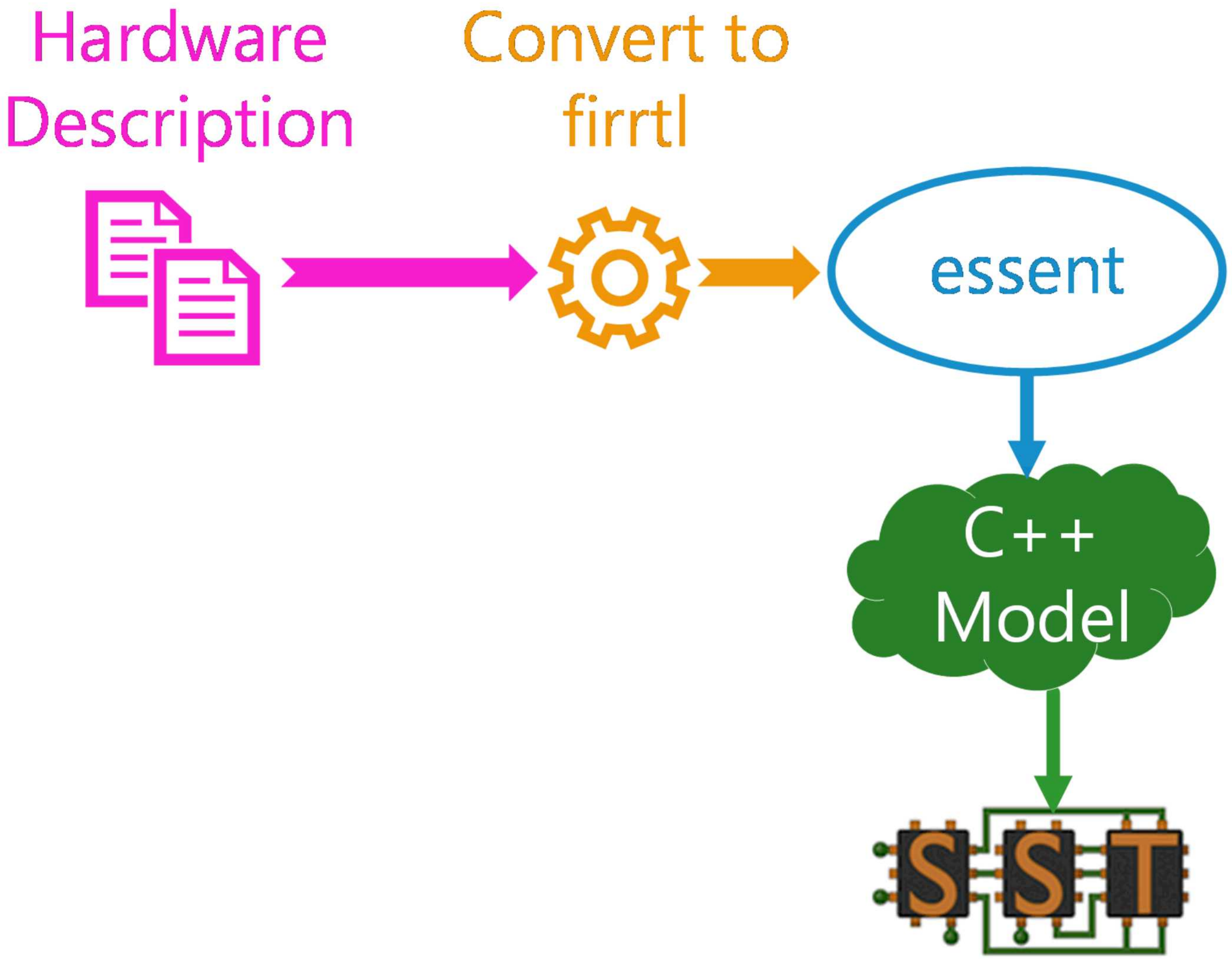
Hardware description as input

- Verilog
- Chisel model

essent ingests firrtl representation and generates a C++ model of the design

Model is integrated with SST via common shims

- Memory
- Control path



SST + Dataflow Component

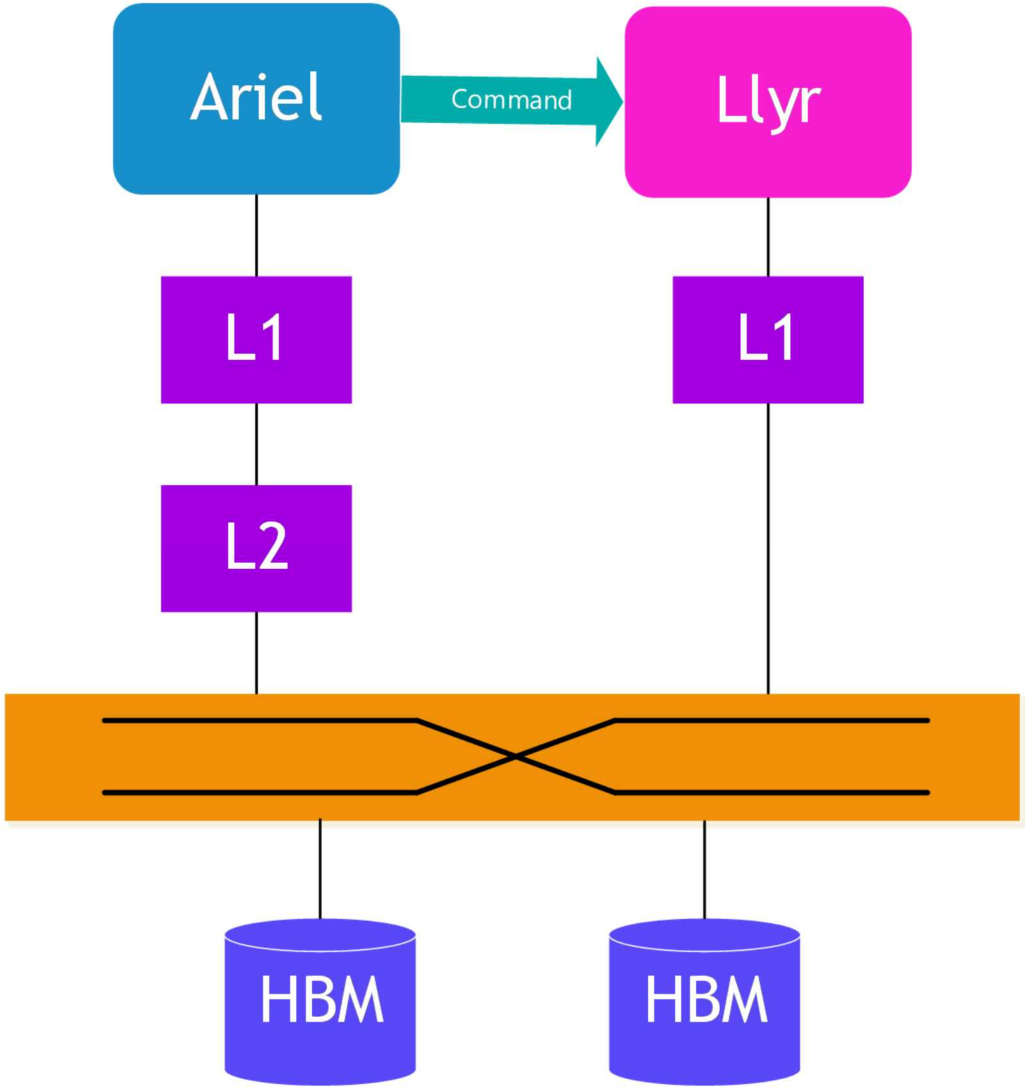
Connected to other components via common SST links

Pros

- Faster than HDL simulation
- More flexible than HDL simulation
- Easier integration with programming model

Cons

- Variable accuracy dependent on kernel complexity
- Open design space creates many possible corner cases



Integration With Programming Model

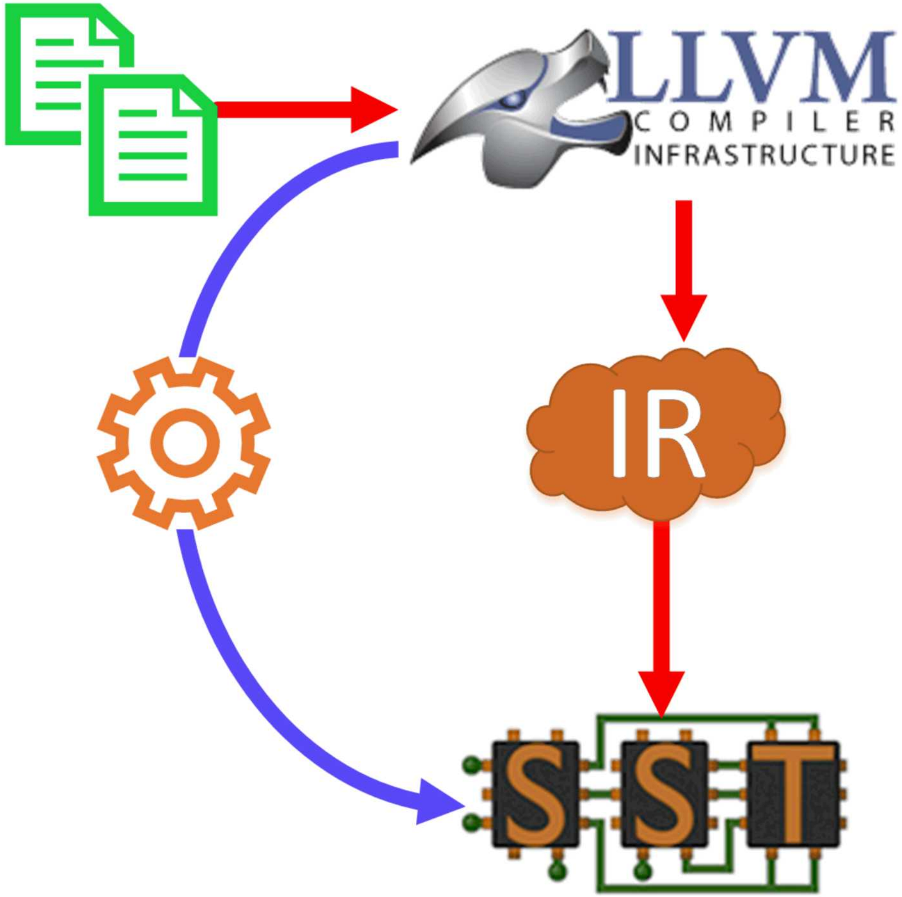
High-level algorithm (C/C++/Python?)

LLVM lowers to IR (not sure if it's LLVM IR or MLIR yet)

LLVM compiles binary

SST ingests both the binary and the IR

- Binary to frontend
- IR to dataflow component



Internal Software Description, Derived From IR

Additional compilation pass to dump IR

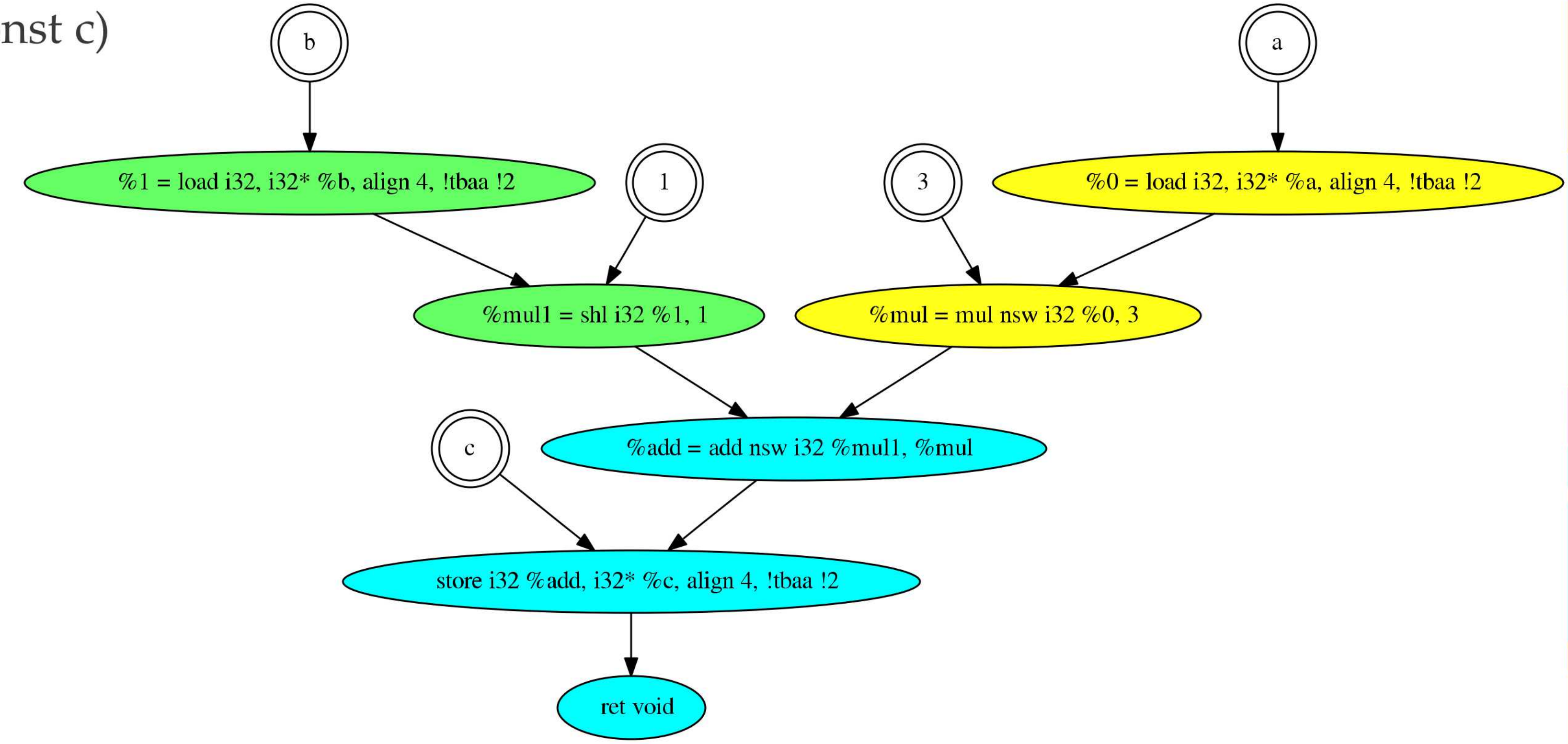
- Binary in directory containing IR files
- When frontend encounters offload regain, function name is mapped back to the correct IR
- Component is responsible for parsing the IR and building CDFG

Likely that offload targets will need a specific naming scheme or additional syntax

```
void  multiply_mod(int* a, int *b, int* const c)
{
  int d = *a;
  int e = *b;

  int f = 3 * d;
  int g = 2 * e;

  *c = f + g;
}
```

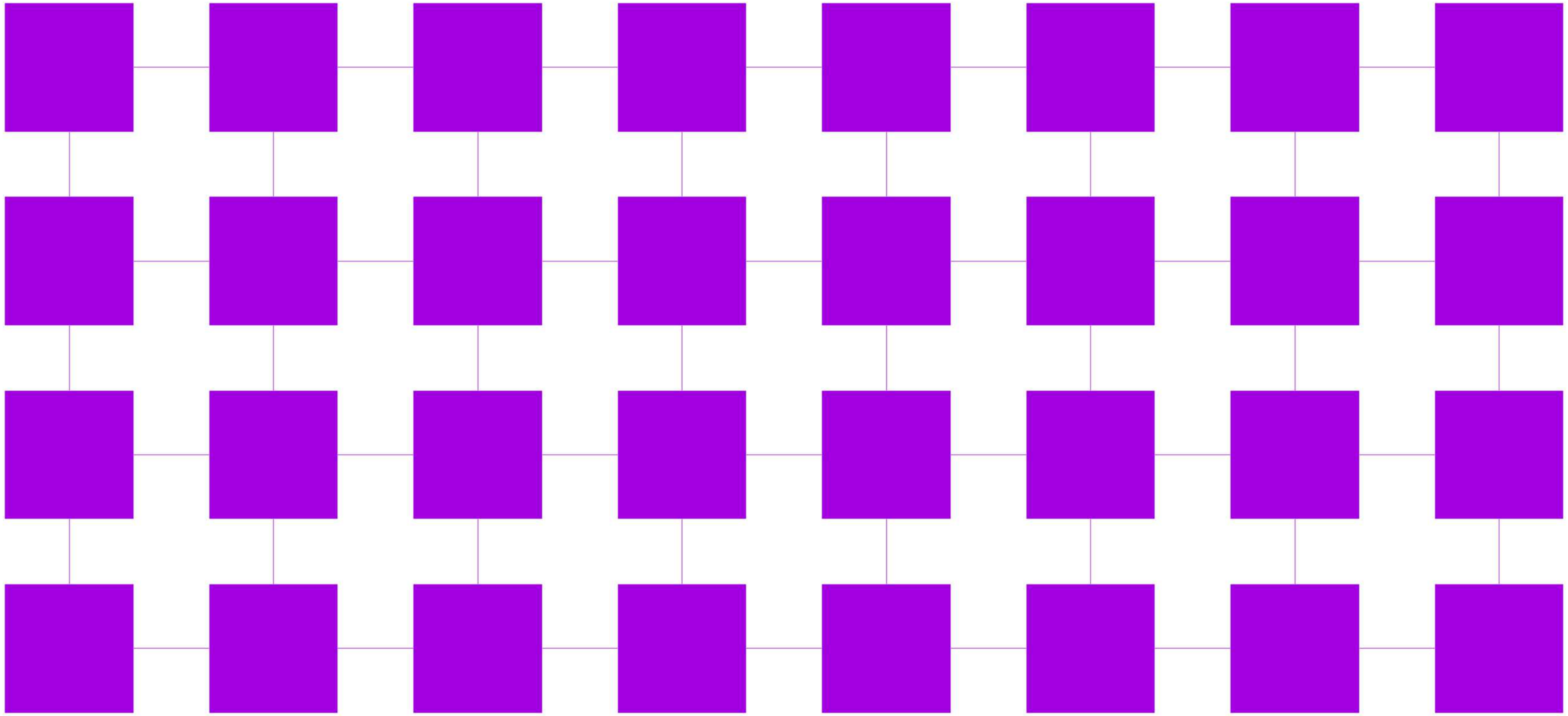

b
a
%1 = load i32, i32* %b, align 4, !tbaa !2
1
3
%0 = load i32, i32* %a, align 4, !tbaa !2
%mul1 = shl i32 %1, 1
%mul = mul nsw i32 %0, 3
c
%add = add nsw i32 %mul1, %mul
store i32 %add, i32* %c, align 4, !tbaa !2
ret void

Hardware Description

Passed as argument to component

Fixed organization and fixed function PEs initially

- Can do any basic operation – add, sub, mul, *etc.*
- More complex mappings later – connectivity, function, etc.



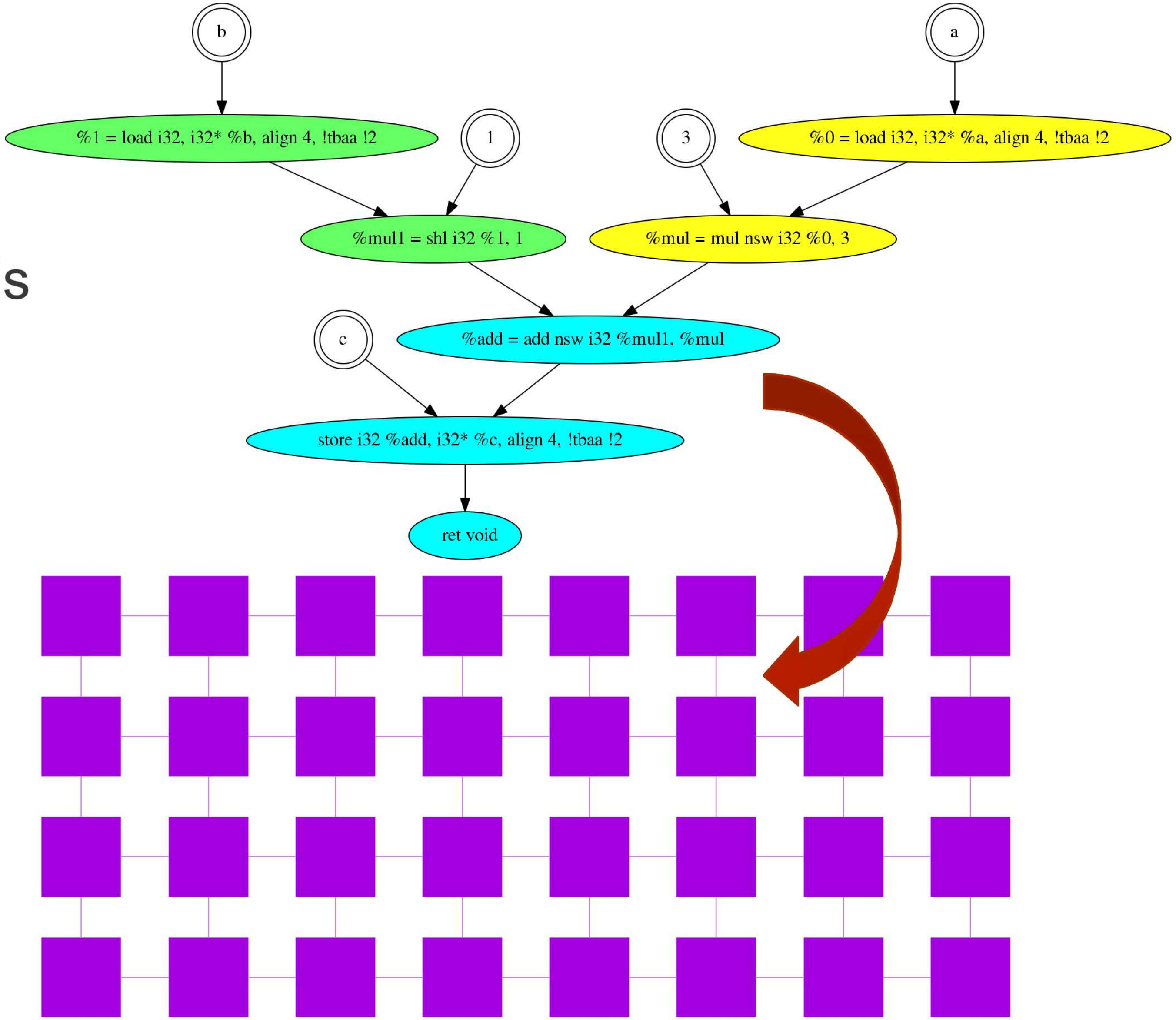
Mapping Software to Hardware

Map CDFG to hardware

Not easy to find best fit

- Essentially mapping a graph to a grid

Will simplify by assuming that all PEs can do everything


b
a
%1 = load i32, i32* %b, align 4, !tbaa !2
1
3
%0 = load i32, i32* %a, align 4, !tbaa !2
%mul1 = shl i32 %1, 1
%mul = mul nsw i32 %0, 3
c
%add = add nsw i32 %mul1, %mul
store i32 %add, i32* %c, align 4, !tbaa !2
ret void

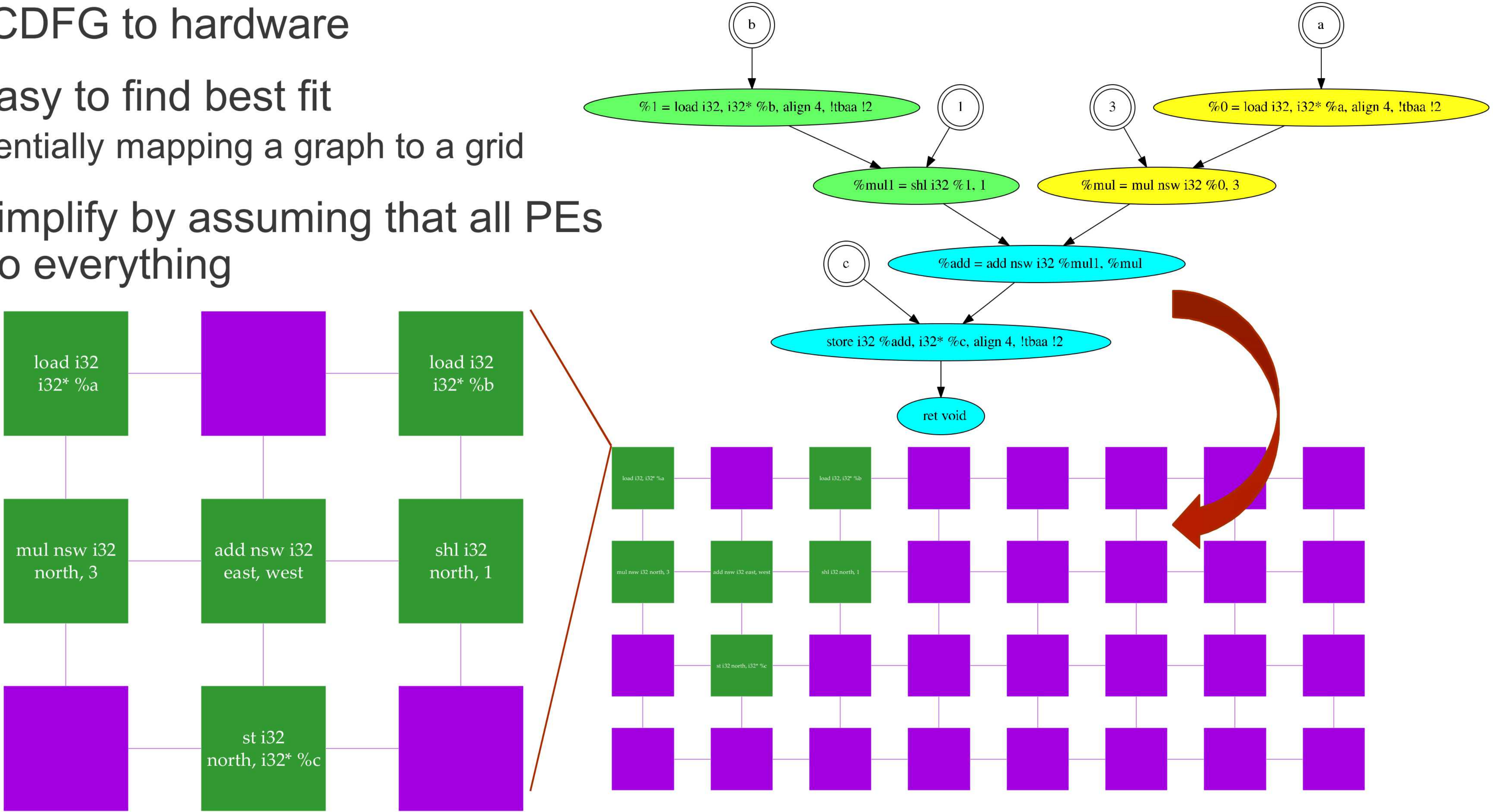
Mapping Software to Hardware

Map CDFG to hardware

Not easy to find best fit

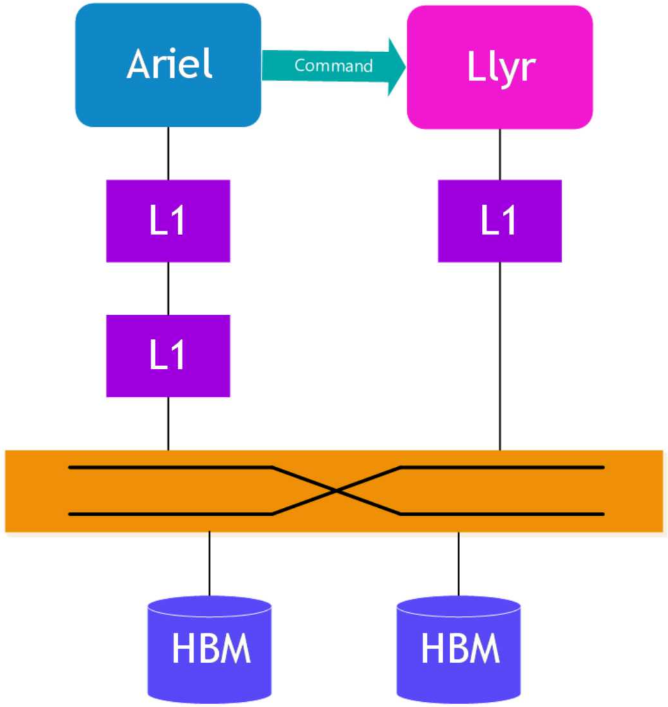
- Essentially mapping a graph to a grid

Will simplify by assuming that all PEs can do everything



Llyr Sample Configuration

```
llyr = sst.Component("dataflow0", "llyr.llyr")
llyr.addParams({
        "verbose": "1",
        "clock"  : "1GHz",
        "config" : "maeri_layout.cfg",
        "fp_lat" : "4",
        "int_lat": "1",
        "div_lat": "3",
        "mul_lat": "2",
        )}
```



Clock: Operating frequency for entire device

Config: Input hardware layout

xxx_lat: Number of cycles to complete the operation

Rough Timelines

essent

- 12-18mo

SST Dataflow Component

- SST Component 6-9mo
- Compiler Component 6-12mo
- Graph Optimization ??

Links

essent

- https://github.com/ucsc-vama/essent

Yosys Verilog-to-Firrtl

- https://github.com/YosysHQ/yosys

Chisel3 (Hardware DSL)

- https://github.com/freechipsproject/chisel3

SST

- https://github.com/sstsimulator

QUESTIONS

BACKUP SLIDES

Section Break Slide