Generating Hardware Description with Target-Independent Code Generator

Zheng, Hongbin

etherzhhb@gmail.com

Outline

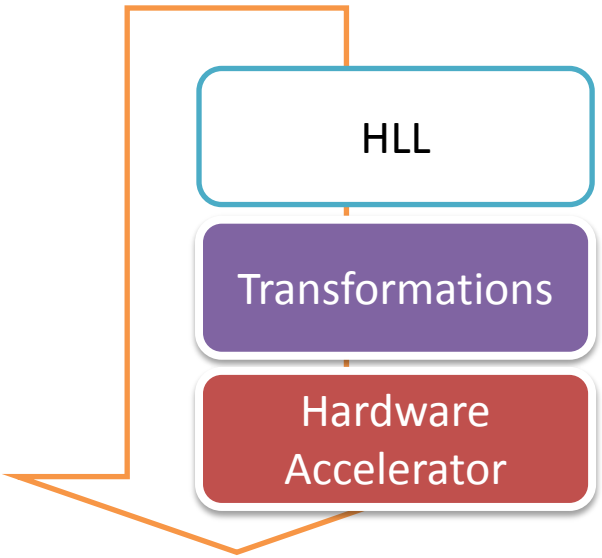
- Introduction
- Overview of the Shang HLS framework
- The Cross-level Engine
- Kernel-only Software Pipelining
- Experimental Results and Further Work

Outline

- Introduction
- Overview of the Shang HLS framework
- The Cross-level Engine
- Kernel-only Software Pipelining
- Experimental Results and Further Work

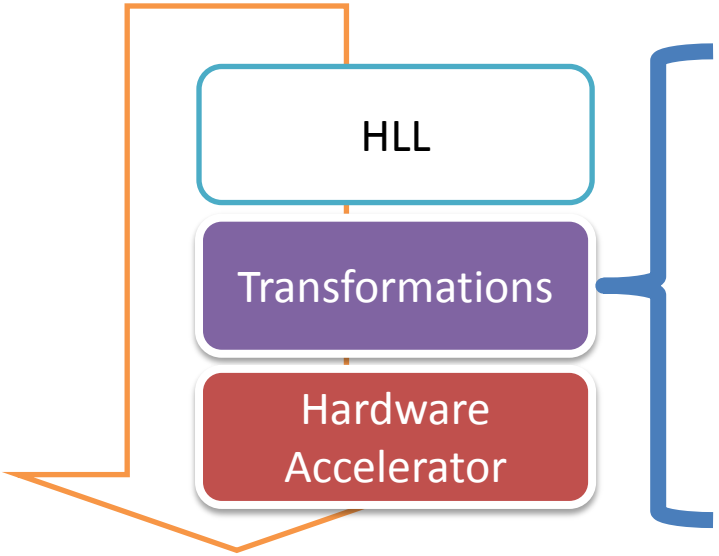
High-level Synthesis

- What is High-level Synthesis (HLS)?
  - Generate Hardware description from High-level Languages (HLL), automatically.



High-level Synthesis

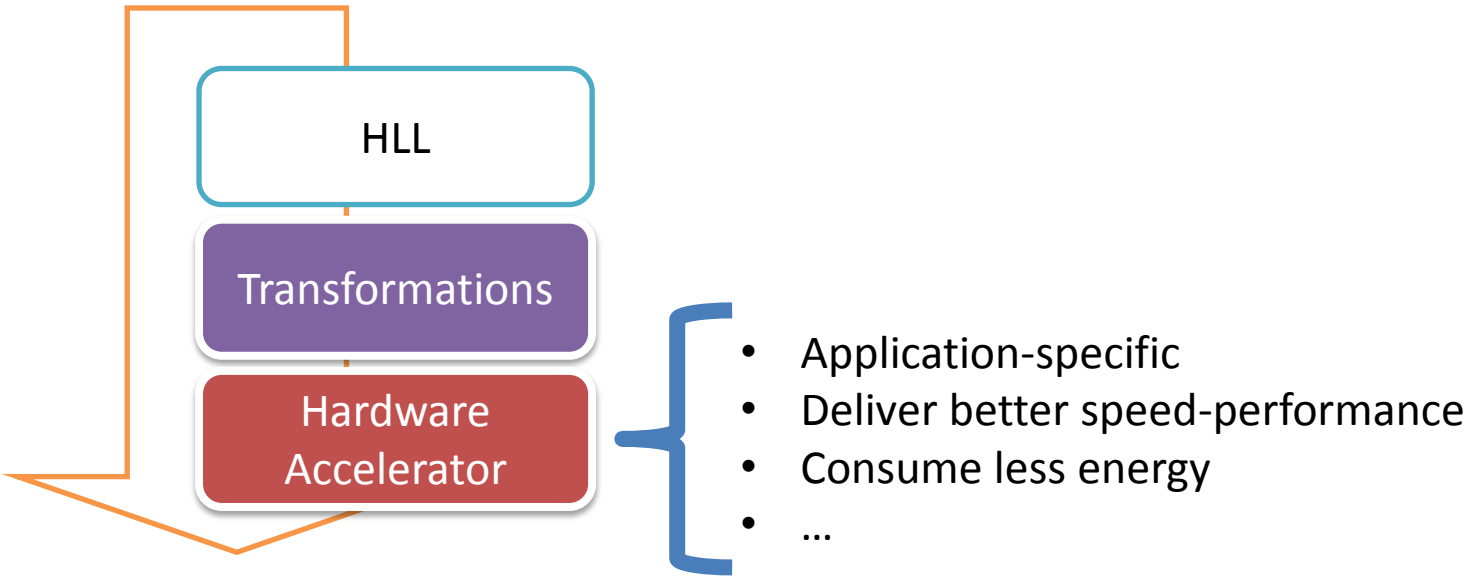
- What is High-level Synthesis (HLS)?
  - Generate Hardware description from High-level Languages (HLL), automatically.



* Scheduling:
-Assign the timing to each operation.
-Determine the speed of the accelerator.
* Functional Units/Registers Allocation and binding:
-Allocate hardware resources,
-And assign the operations to the allocated resources.
-Determine the resource requirement.
* …

High-level Synthesis

- What is High-level Synthesis (HLS)?
  - Generate Hardware description from High-level Languages (HLL), automatically.

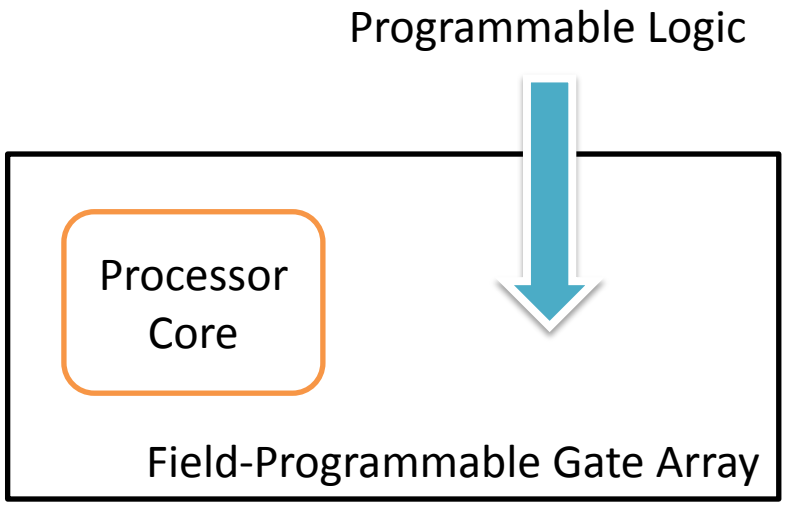


High-level Synthesis

- What is High-level Synthesis (HLS)?
  - Generate Hardware description from High-level Languages (HLL), automatically.
- Why HLS?
  - Better performance, lesser power consumption …
  - Achieve good quality without hardware expert…
  - Shorter development cycle ...
  - …

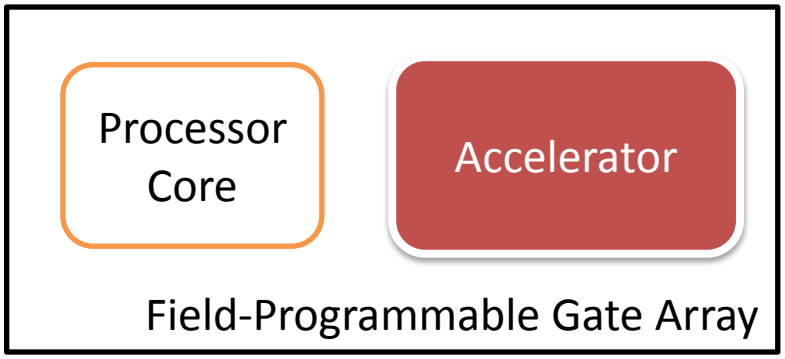
HLS Example

- There exist some FPGAs with ARM core:
  - Arria V SoC FPGA by Altera
  - Zynq by Altera Xilinx

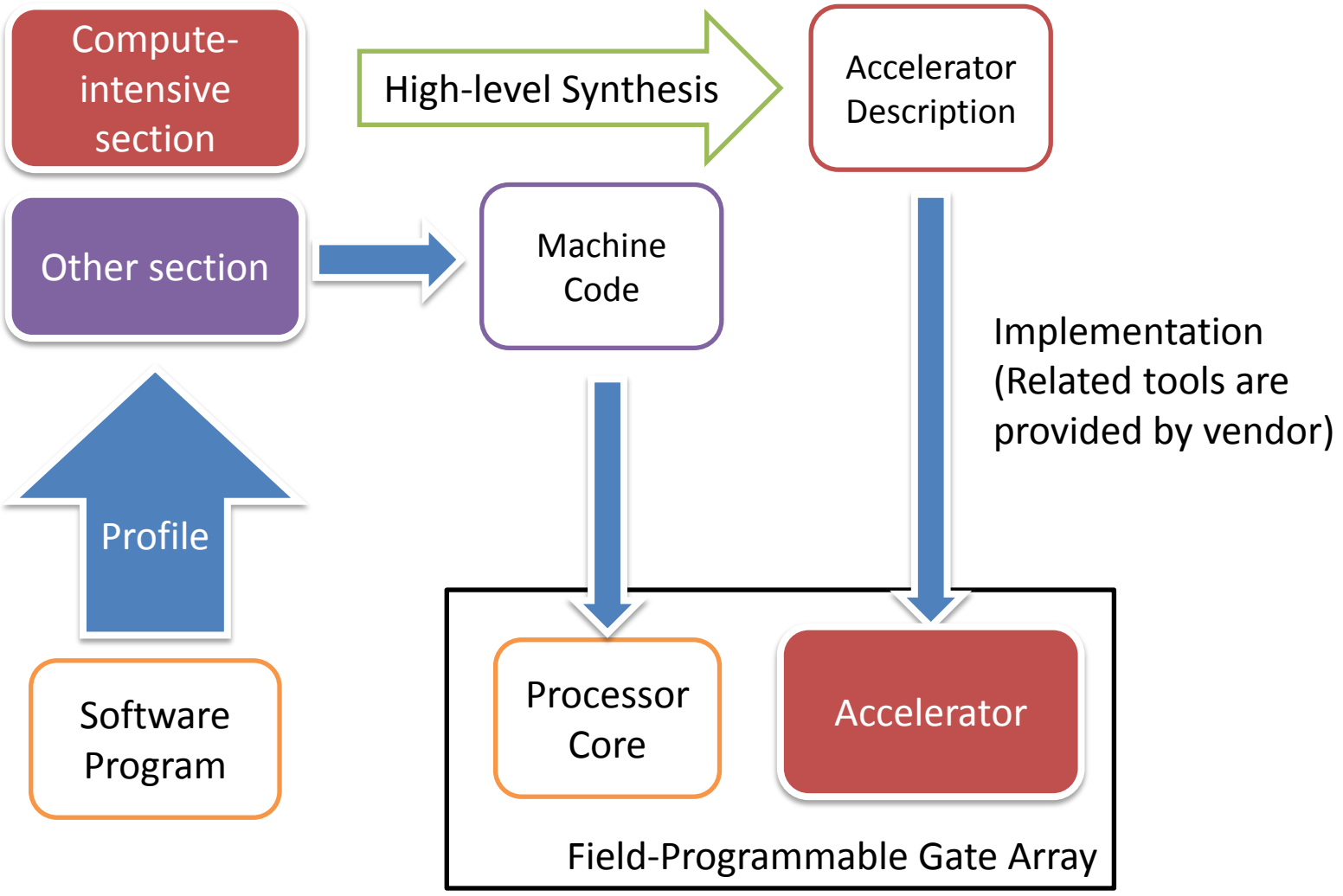


HLS Example

- There exist some FPGAs with ARM core:
  - Arria V SoC FPGA by Altera
  - Zynq by Altera Xilinx
- The programmable logic can implement the hardware accelerator.



HLS Example


Compute-intensive section
High-level Synthesis
Accelerator Description
Other section
Machine Code
Implementation (Related tools are provided by vendor)
Profile
Software Program
Processor Core
Accelerator
Field-Programmable Gate Array

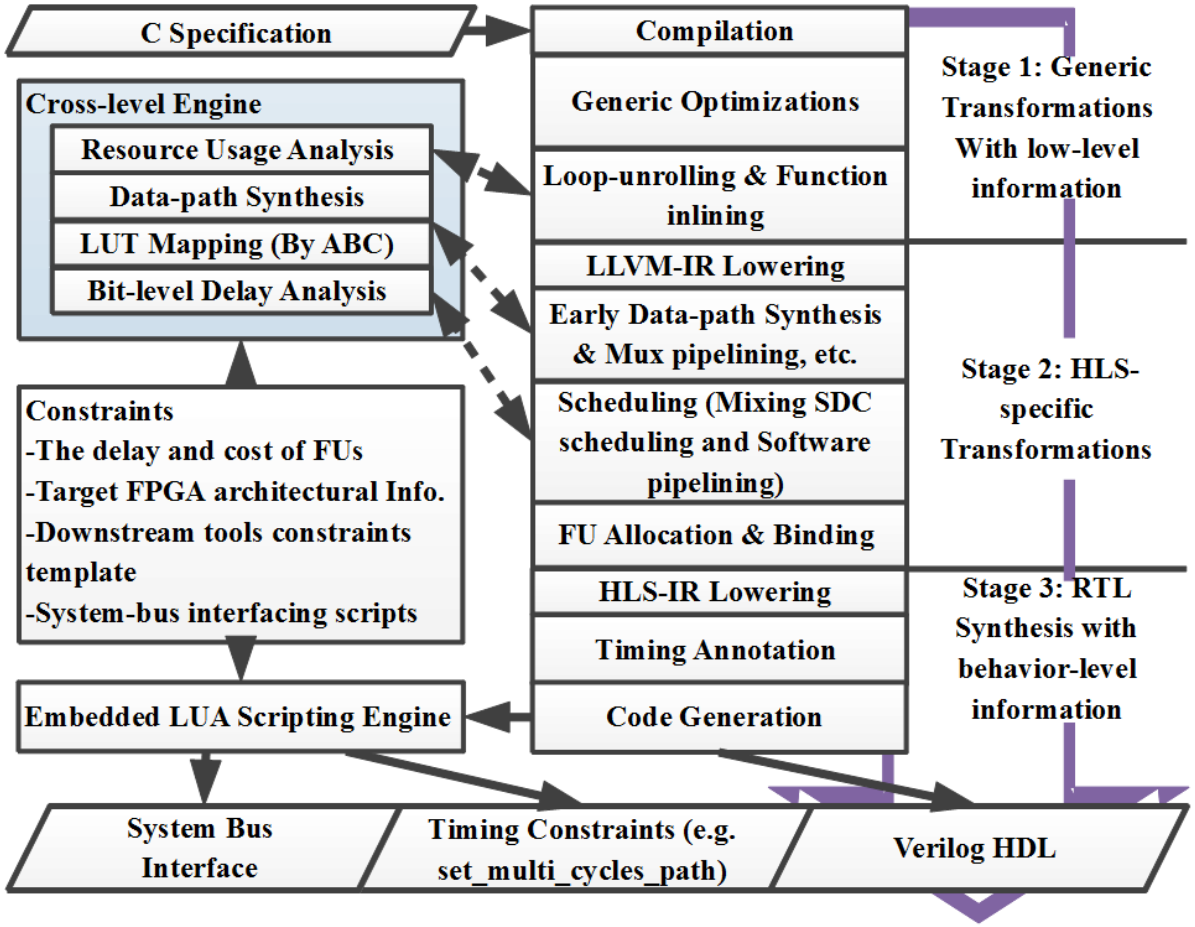
Outline

- Introduction
- Overview of the Shang HLS framework
- The Cross-level Engine
- Kernel-only Software Pipelining
- Experimental Results and Further Work

The Shang HLS framework

- Automatically translate C to Hardware.
- Advance hardware-specific optimizations:
  - Subword-level and Bit-level optimizations.
  - Cross BB parallelism exploitation.
  - … all based on the Target-Independent CodeGen.
- Platform Independent.
- And open source.
  - https://github.com/SysuEDA/Shang

Flow Overview

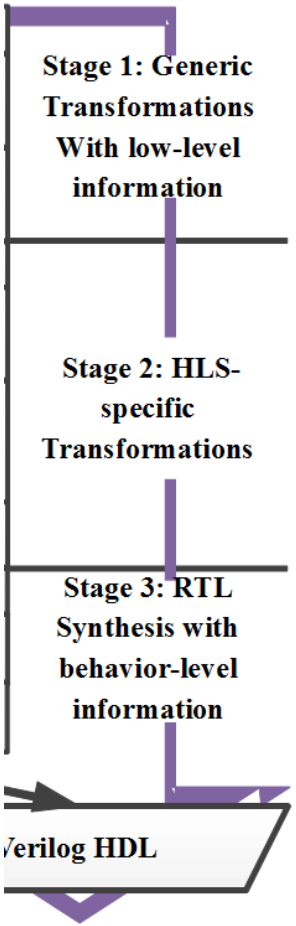


Intermediate Representations

LLVM IR

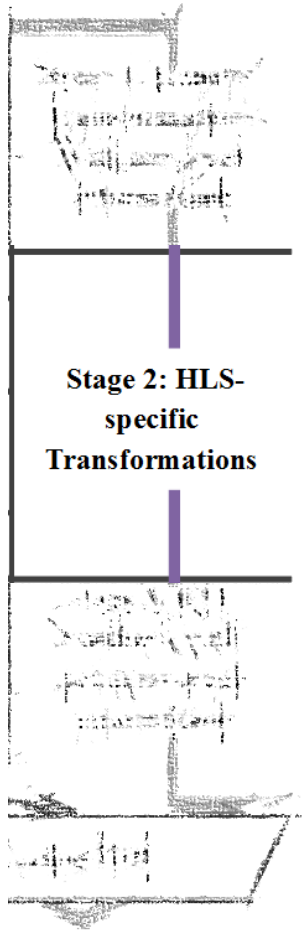
Machine Code

RTL Netlist



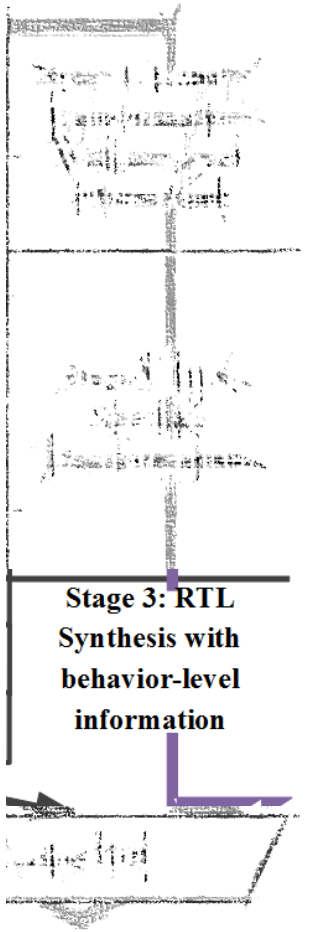
HLS-specific Machine Code

- A virtual instruction set targeting FPGA.
- Contains special instructions:
  - Bit concatenation, subword extraction, look-up table …
- All instructions are predicated.
  - Enable some advance control construct.
- Represents the behavior of the design.
  - In more details than LLVM IR.

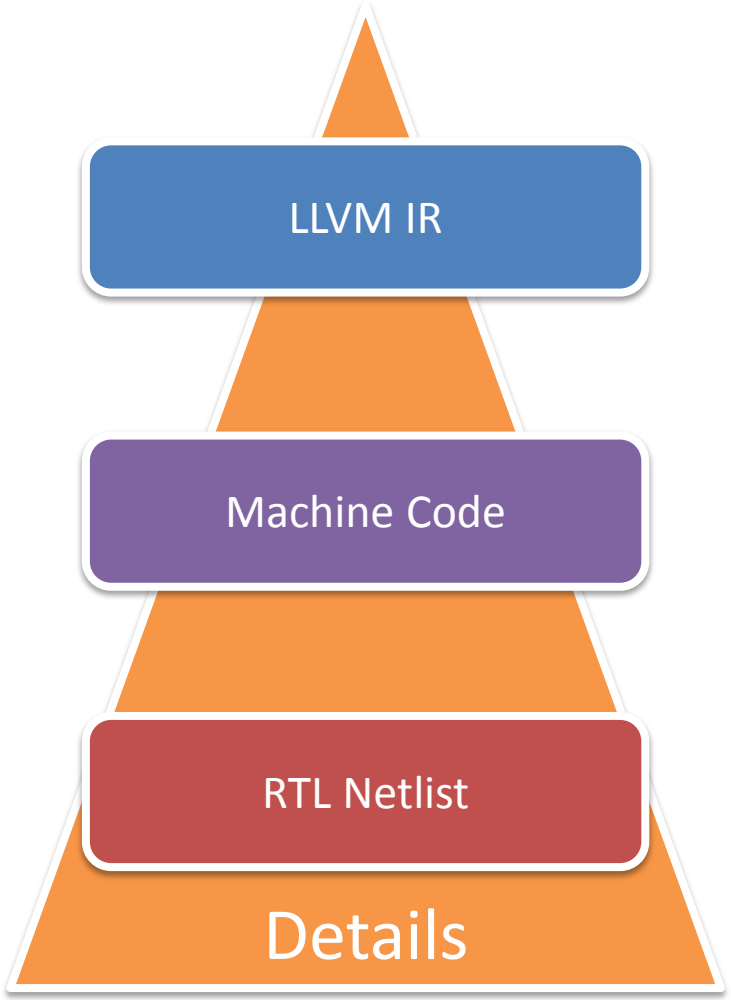


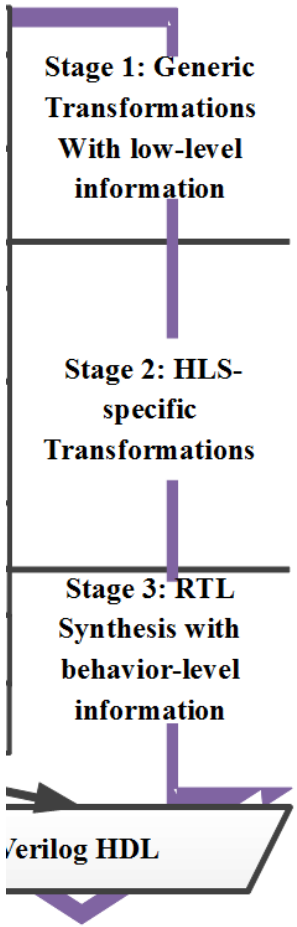
RTL netlist

- Derived from the scheduled and bound Machine Code.
- Explicitly describe the HW:
  - The Functional Units,
  - And how they are connected.
- The data transactions on the registers:
  - At which cycle?
  - Under what condition?

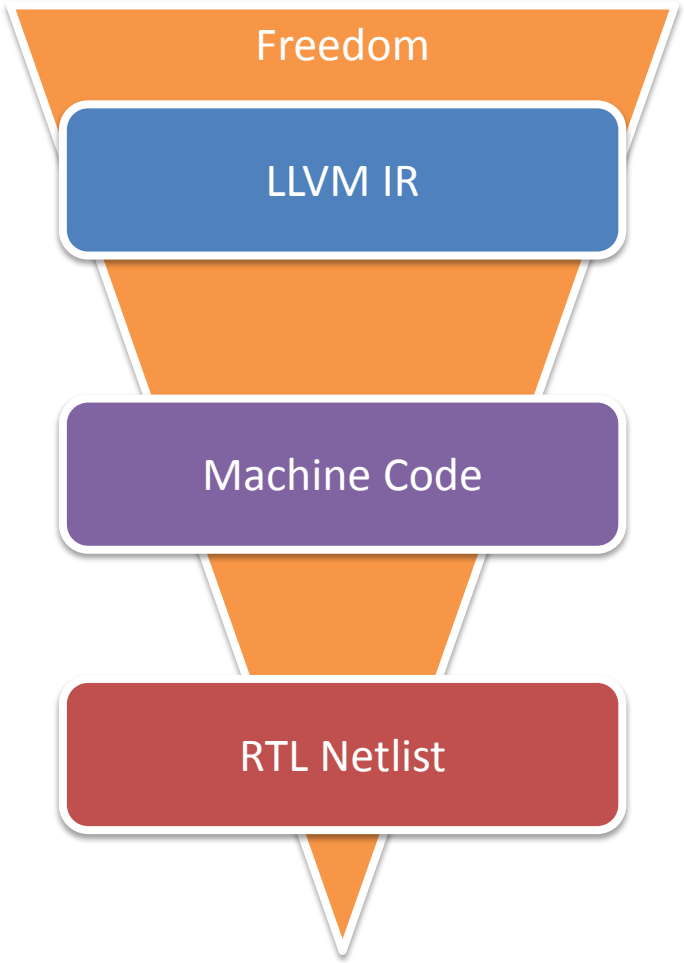


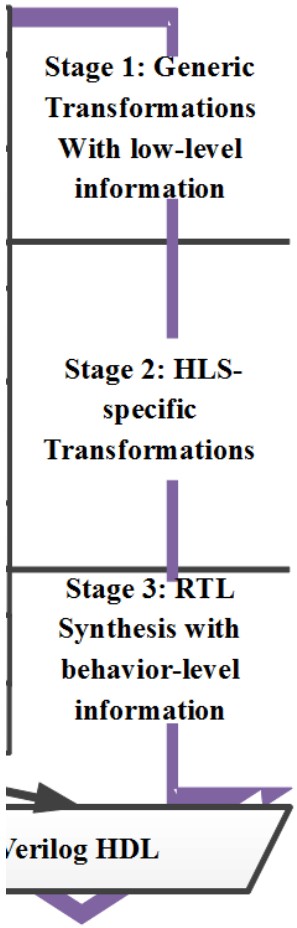
Intermediate Representations



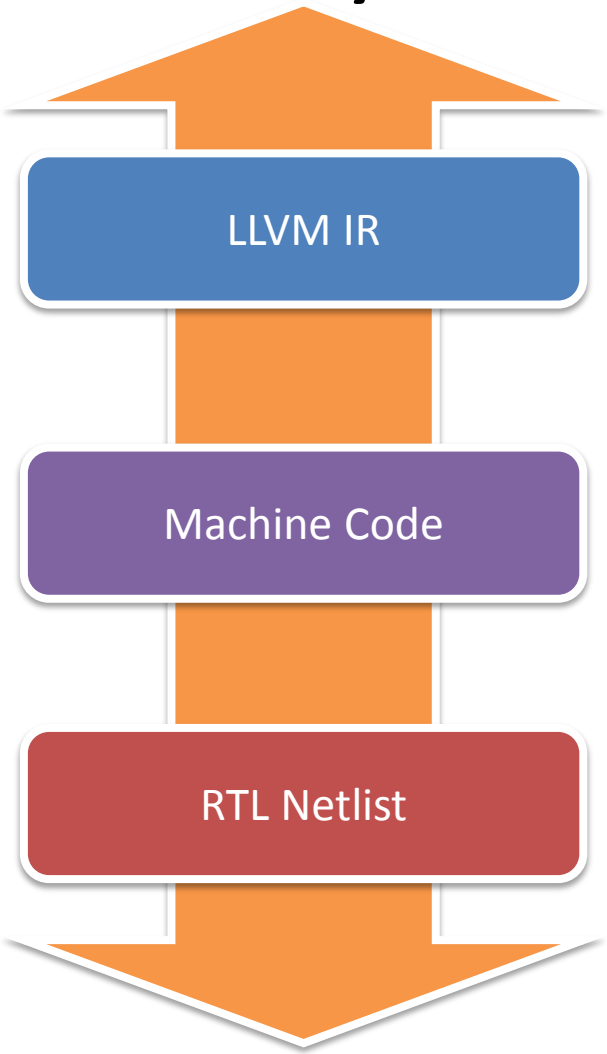


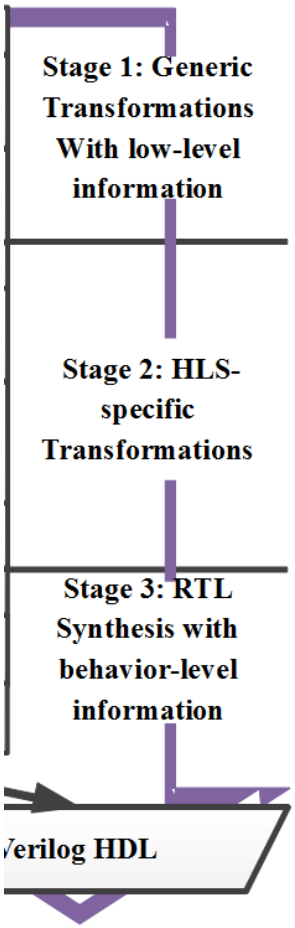
Intermediate Representations



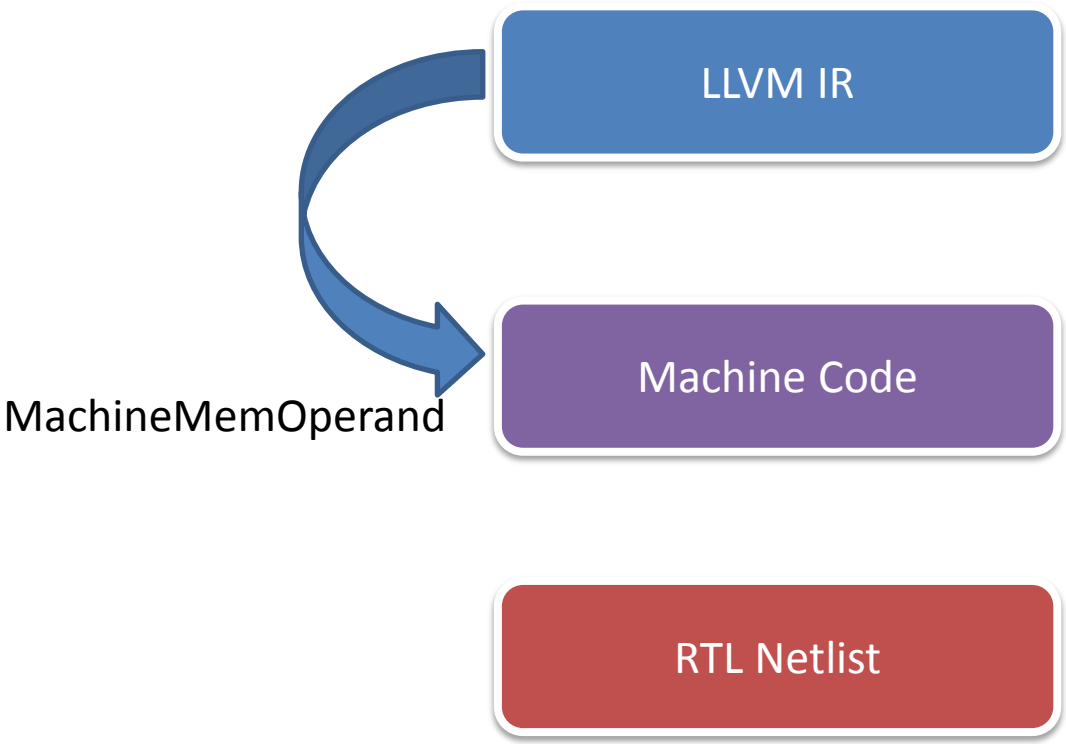


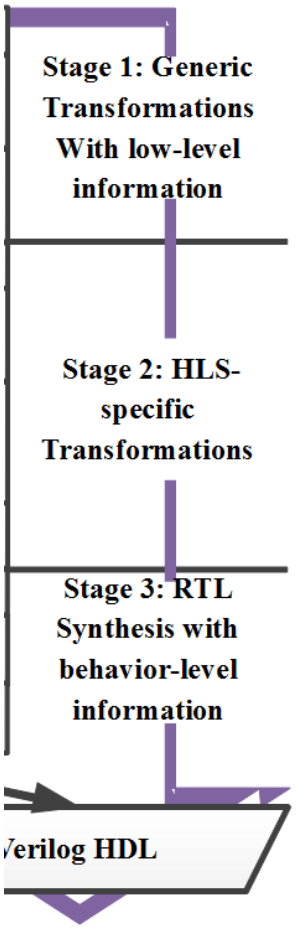
Cross-level Analyses/Optimizations



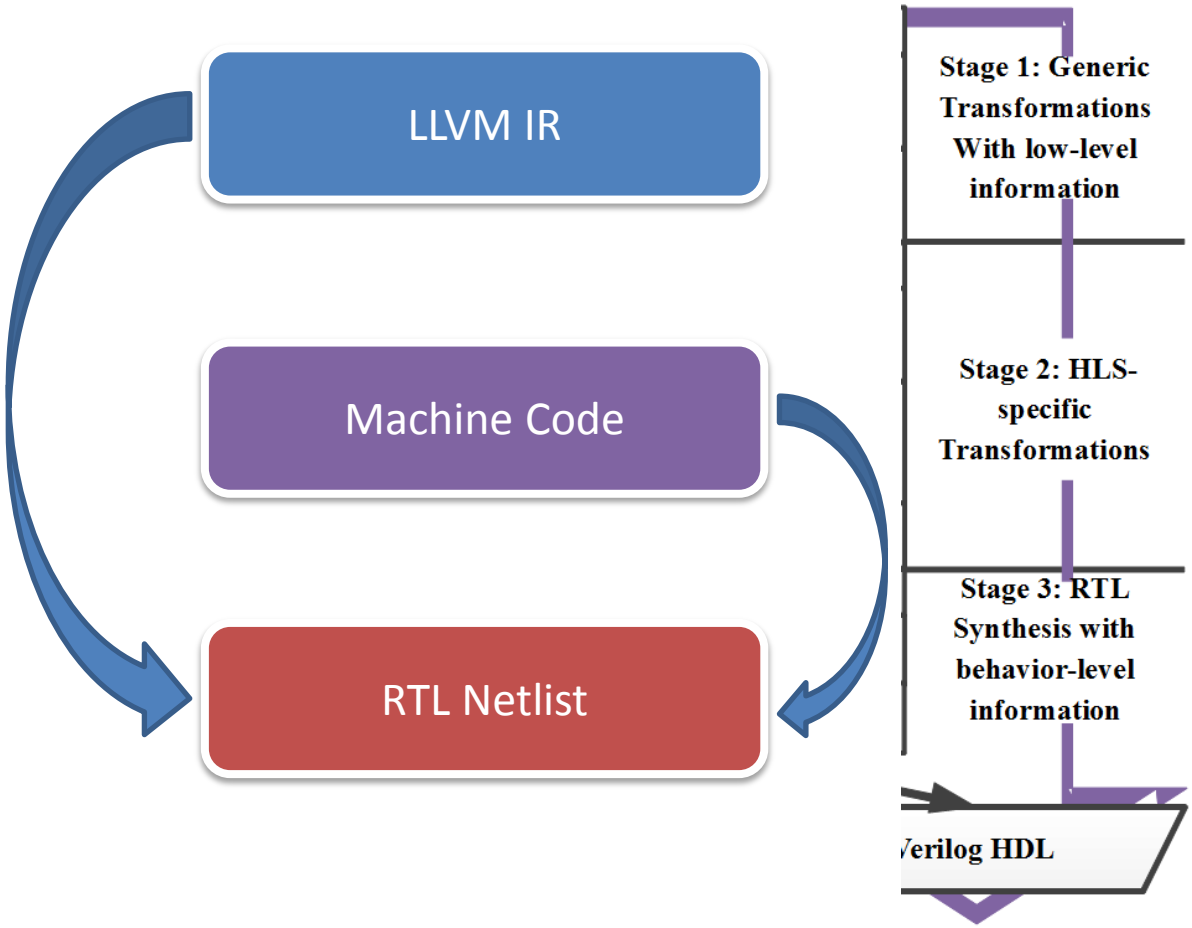


Memory Dependency

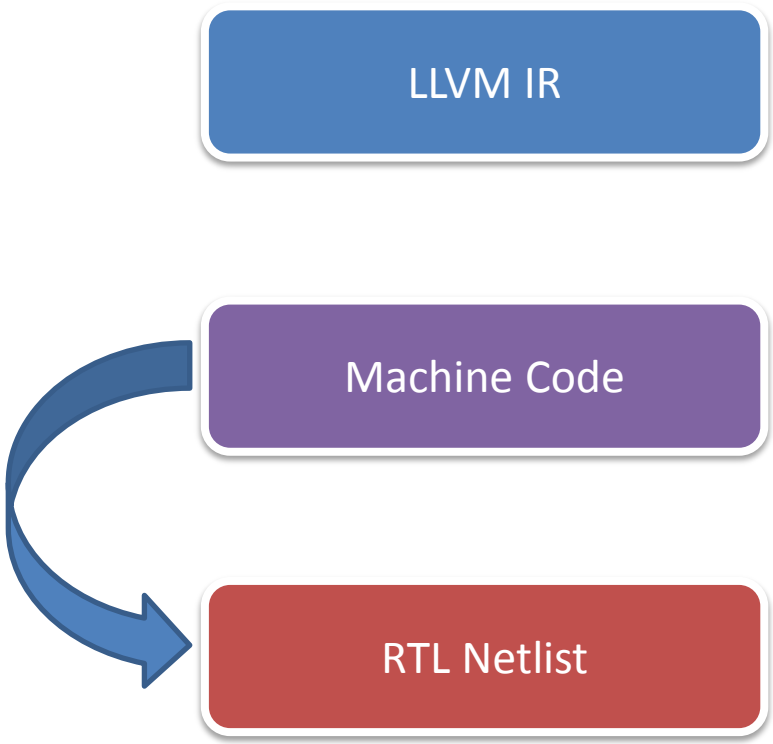


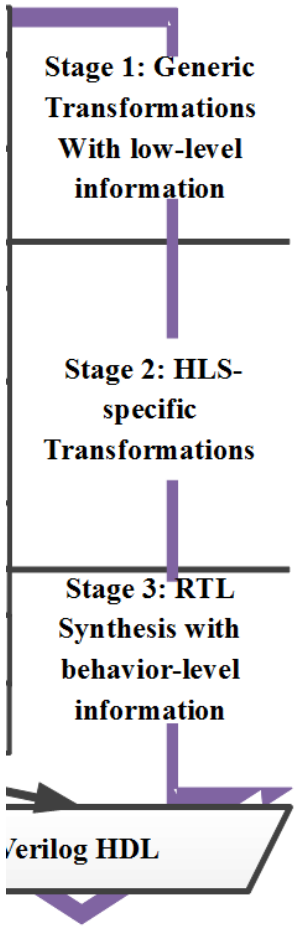


Delay/Cost Estimation



Timing annotation

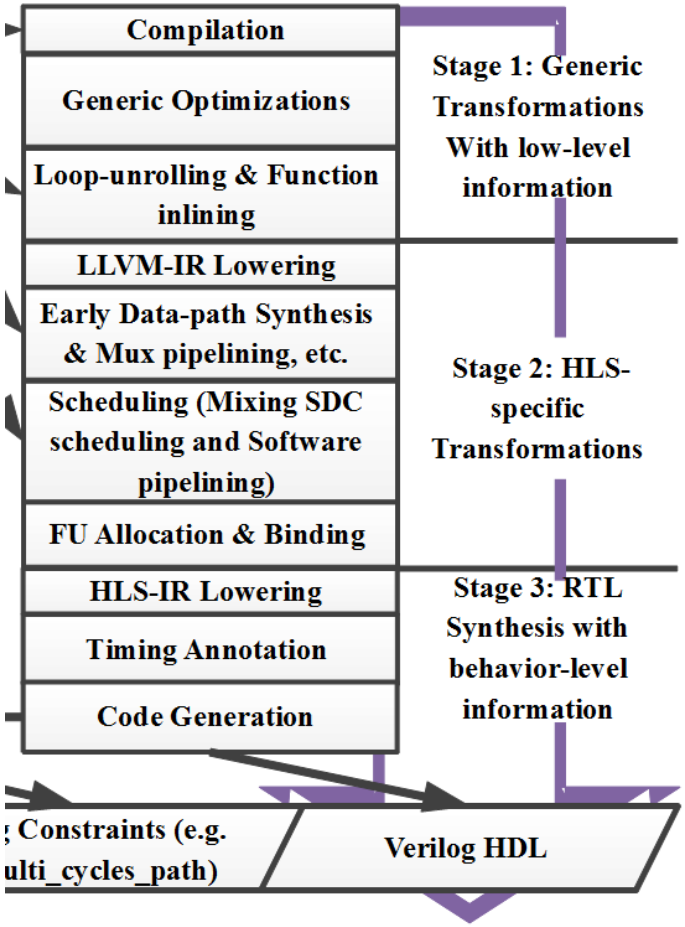




Flow implementation

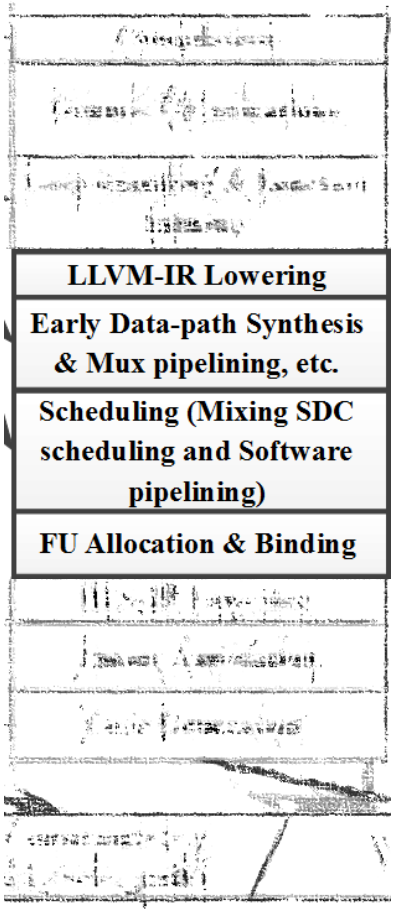
LLVM IR Passes

MachineFunction Passes,
Based on a HLS-specific virtual instruction set.



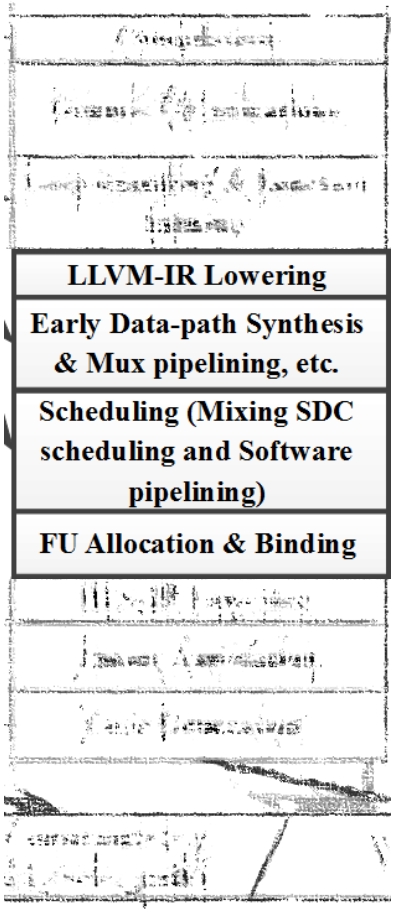
Fitting HLS flow into CodeGen

- Scheduling
  - Build the Scheduling Graph based on pre-register-allocation Machine code.



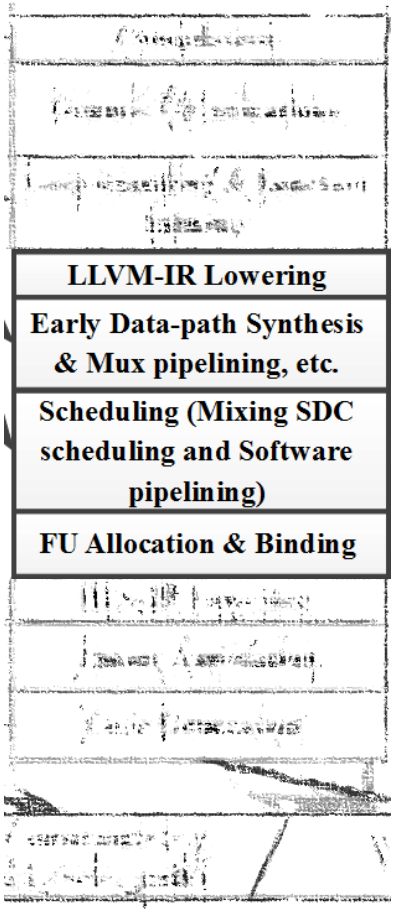
Fitting HLS flow into CodeGen

- Scheduling
  - Build the Scheduling Graph based on pre-register-allocation Machine code.
  - Pack instructions into bundles according to the scheduling results.



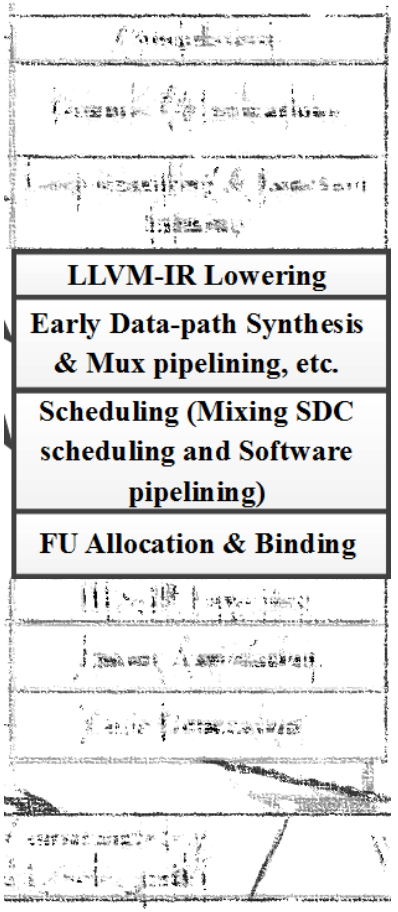
Fitting HLS flow into CodeGen

- Binding
  - Model all resource with physical registers.



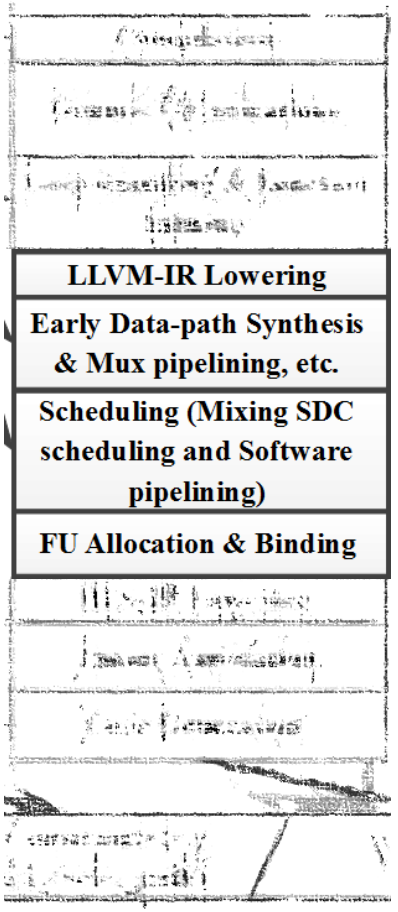
Fitting HLS flow into CodeGen

- Binding
  - Model all resource with physical registers.
  - LLVM helps to eliminate the PHIs.



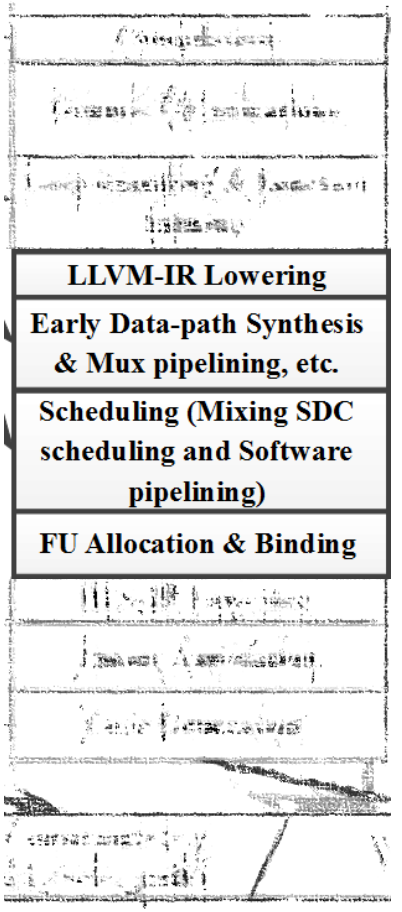
Fitting HLS flow into CodeGen

- Binding
  - Model all resource with physical registers.
  - LLVM helps to eliminate the PHIs.
  - LLVM helps to build the live-interval.



Fitting HLS flow into CodeGen

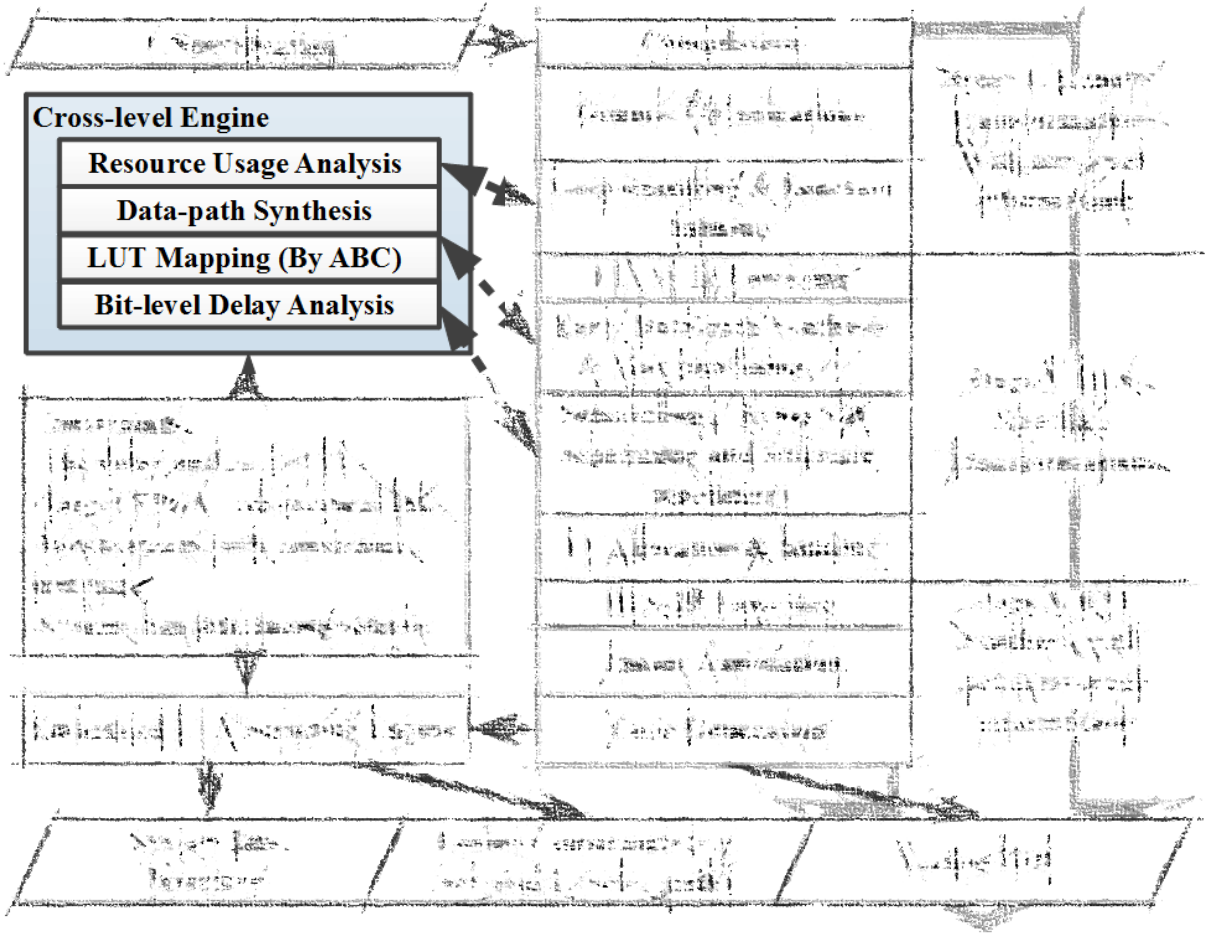
- Binding
  - Model all resource with physical registers.
  - LLVM helps to eliminate the PHIs.
  - LLVM helps to build the live-interval.
  - Perform LLVM physical register binding to solve the problem.



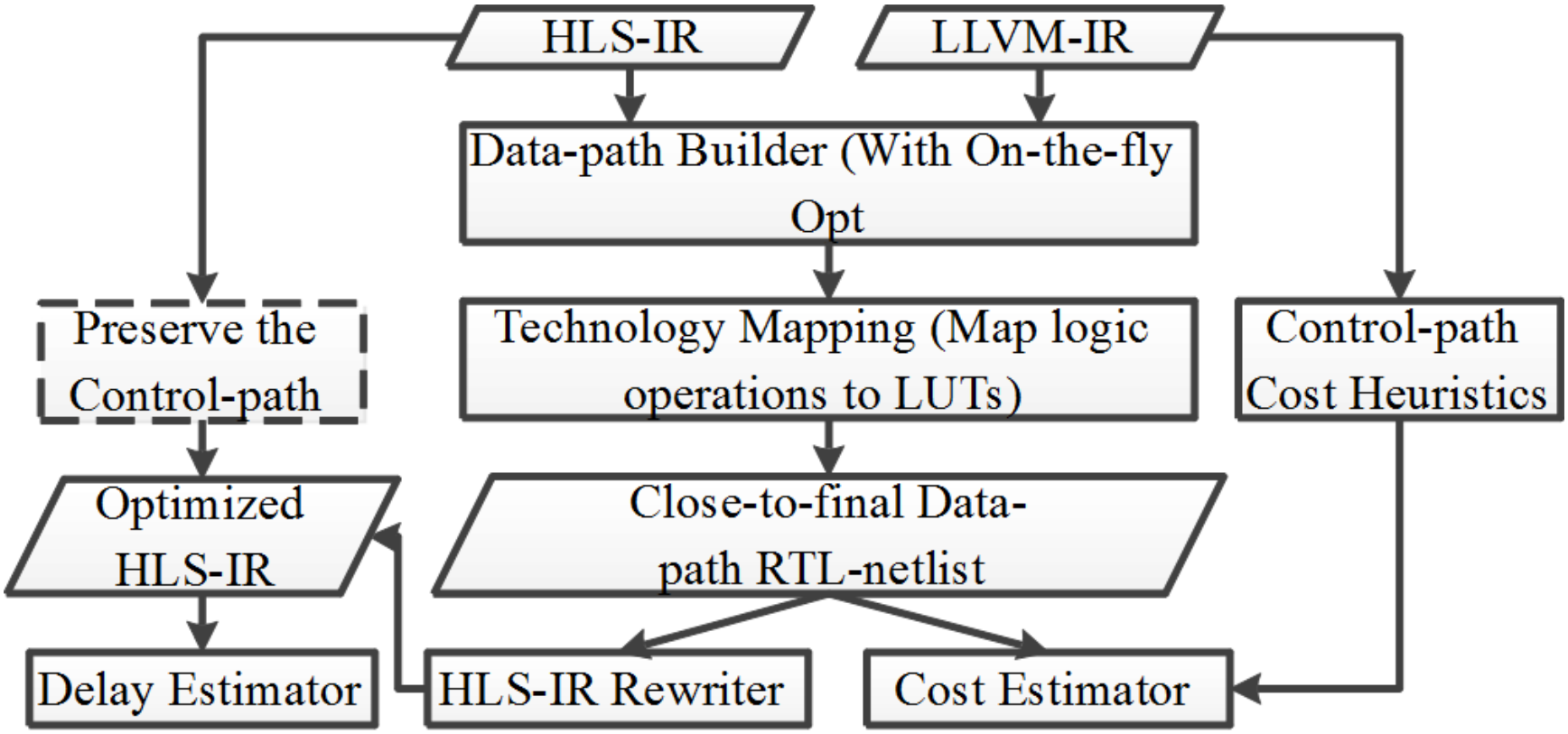
Outline

- Introduction
- Overview of the Shang HLS framework
- The Cross-level Engine
- Kernel-only Software Pipelining
- Experimental Results and Further Work

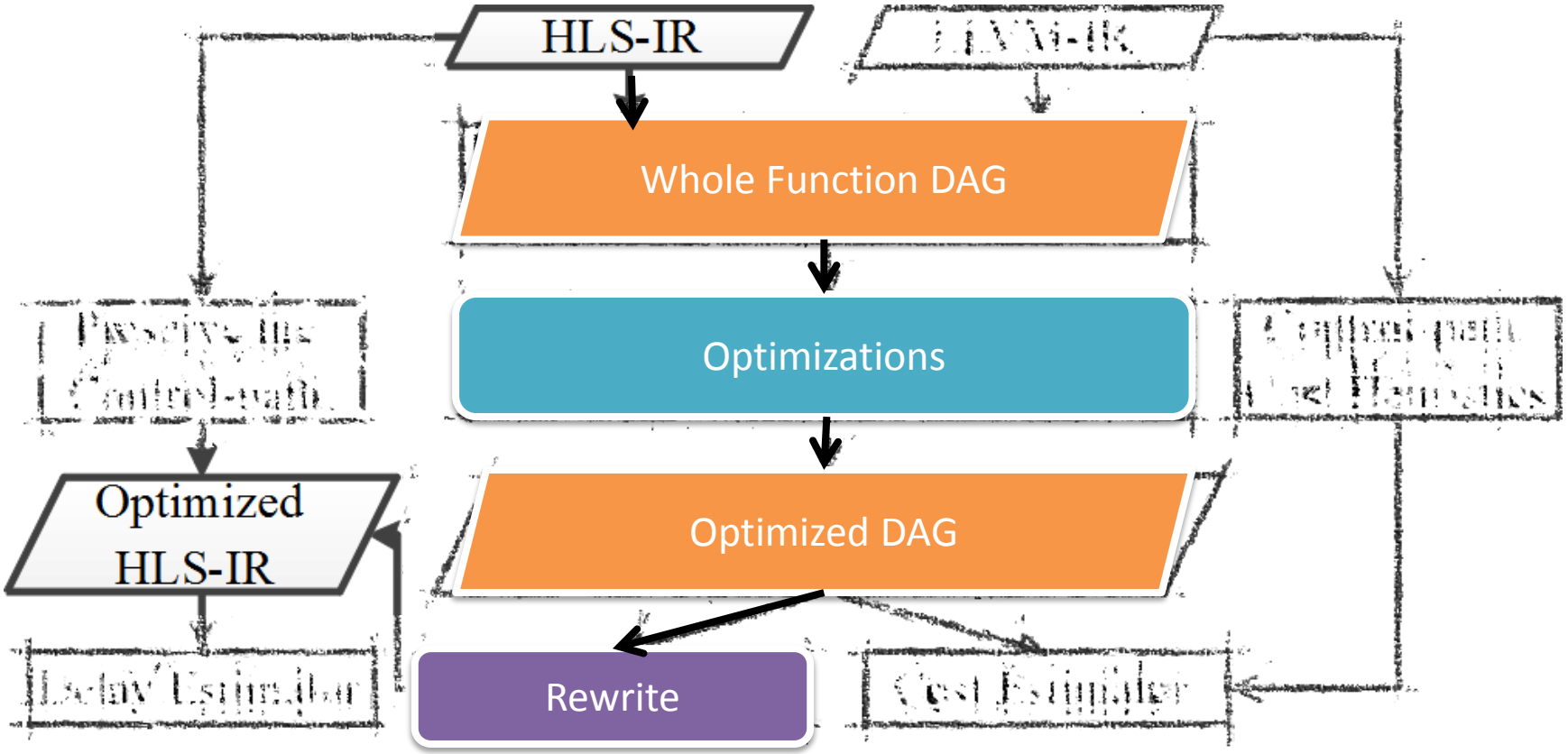
Cross-level Engine



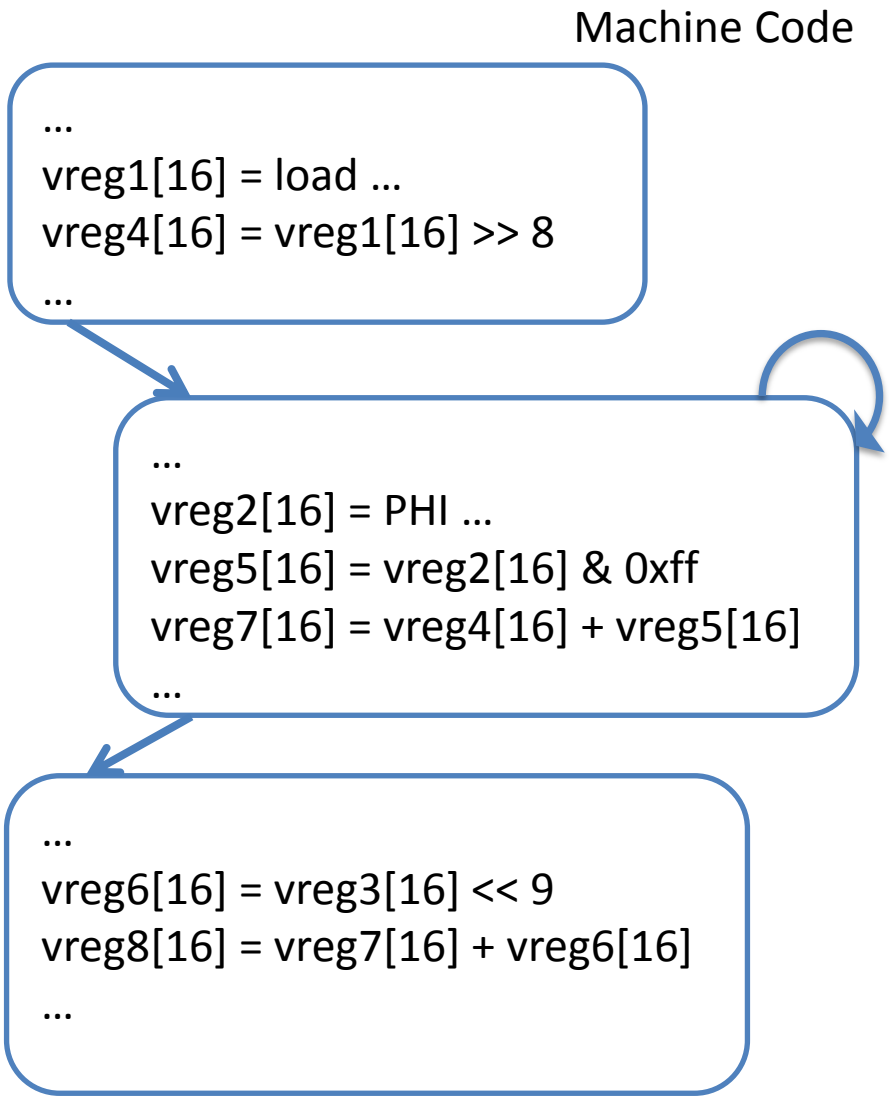
CLE Internal



Early Data-path Synthesis



Early Data-path Synthesis - Example



Early Data-path Synthesis - Example

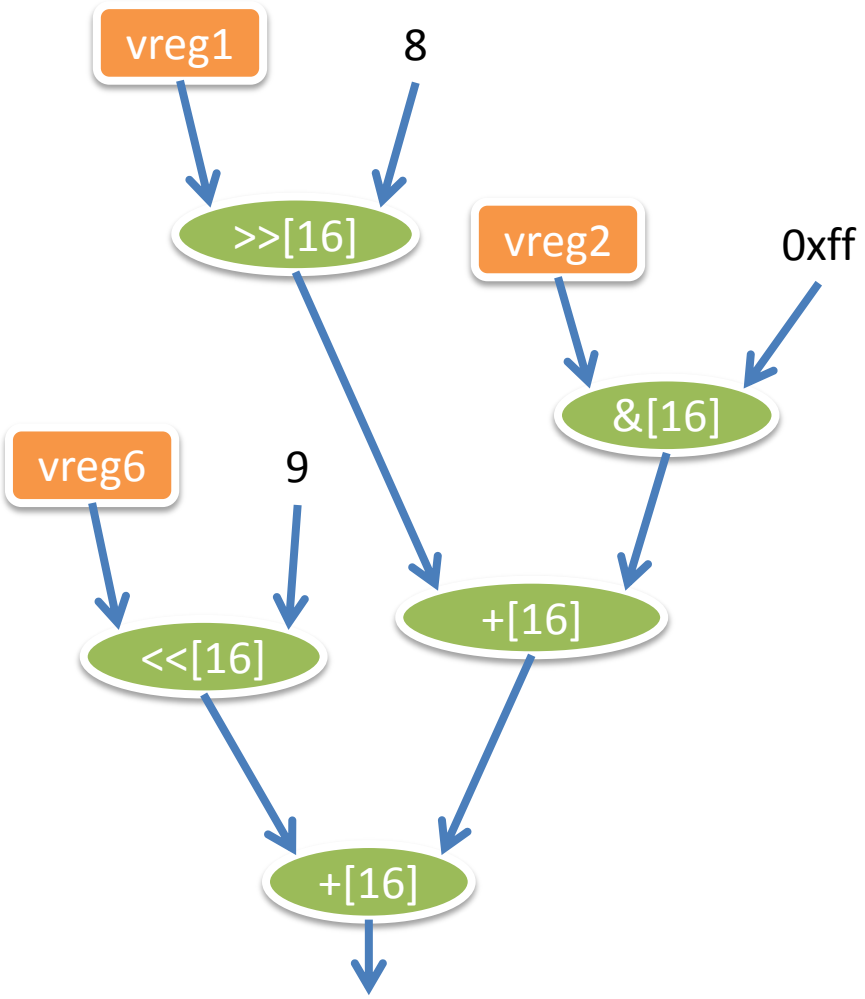
Machine Code

```
…
vreg1[16] = load …
vreg4[16] = vreg1[16] >> 8
…
```

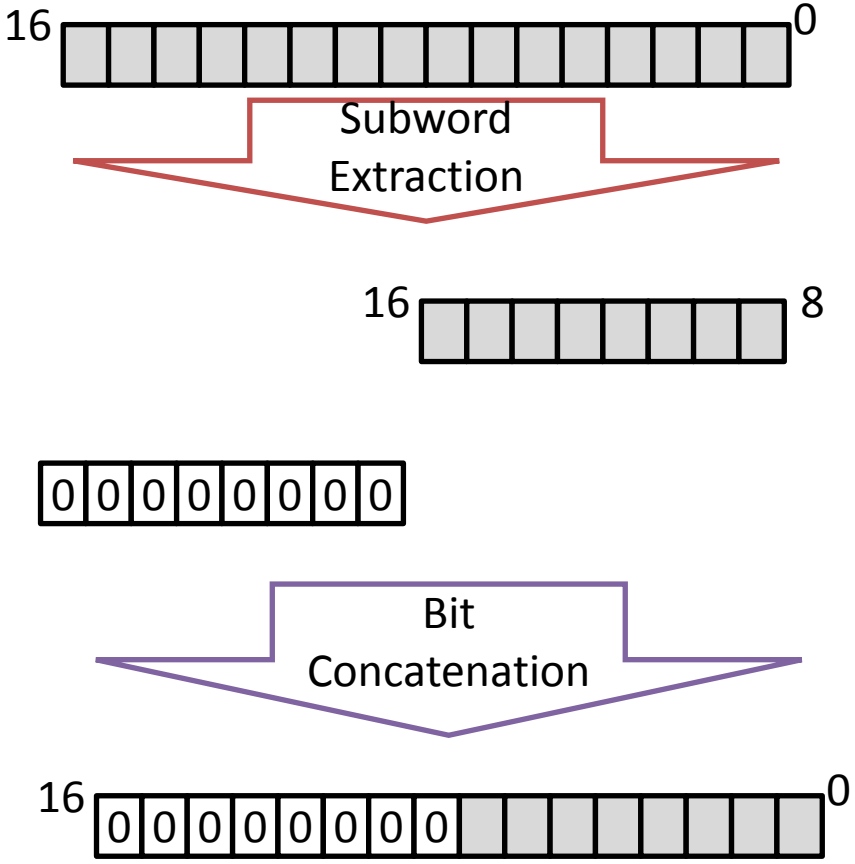
```
…
vreg2[16] = PHI …
vreg5[16] = vreg2[16] & 0xff
vreg7[16] = vreg4[16] + vreg5[16]
…
```

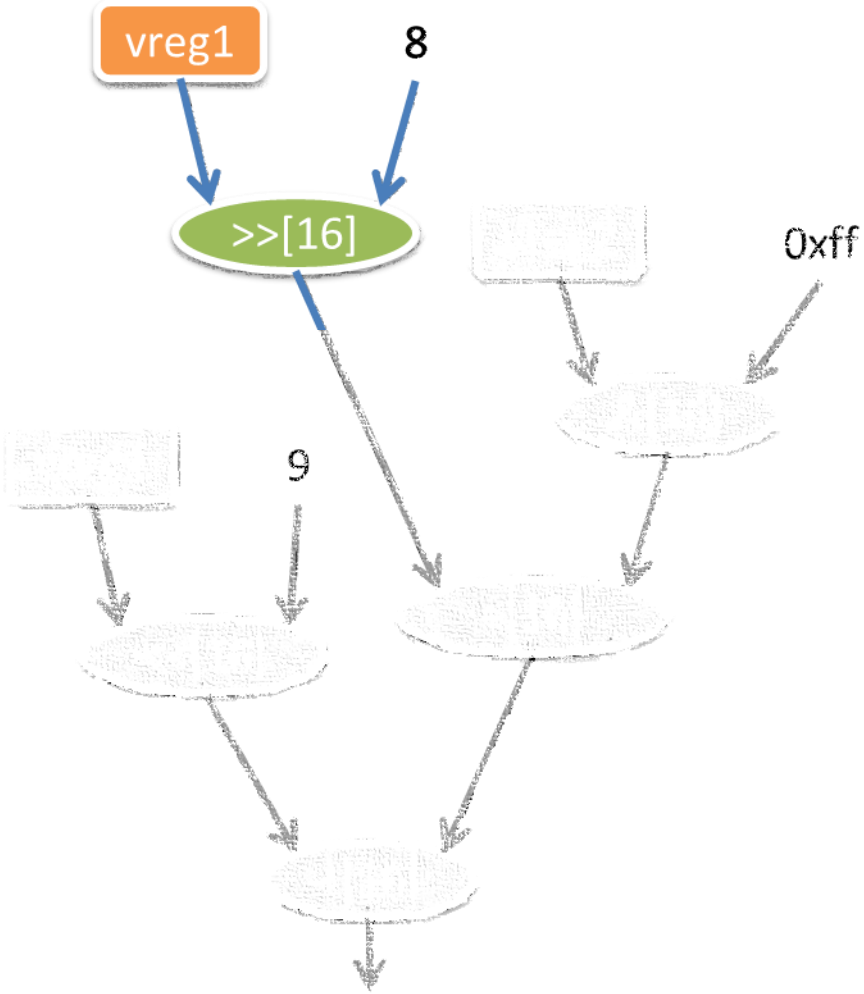
```
…
vreg6[16] = vreg3[16] << 9
vreg8[16] = vreg7[16] + vreg6[16]
…
```

Data-path of RTL netlist

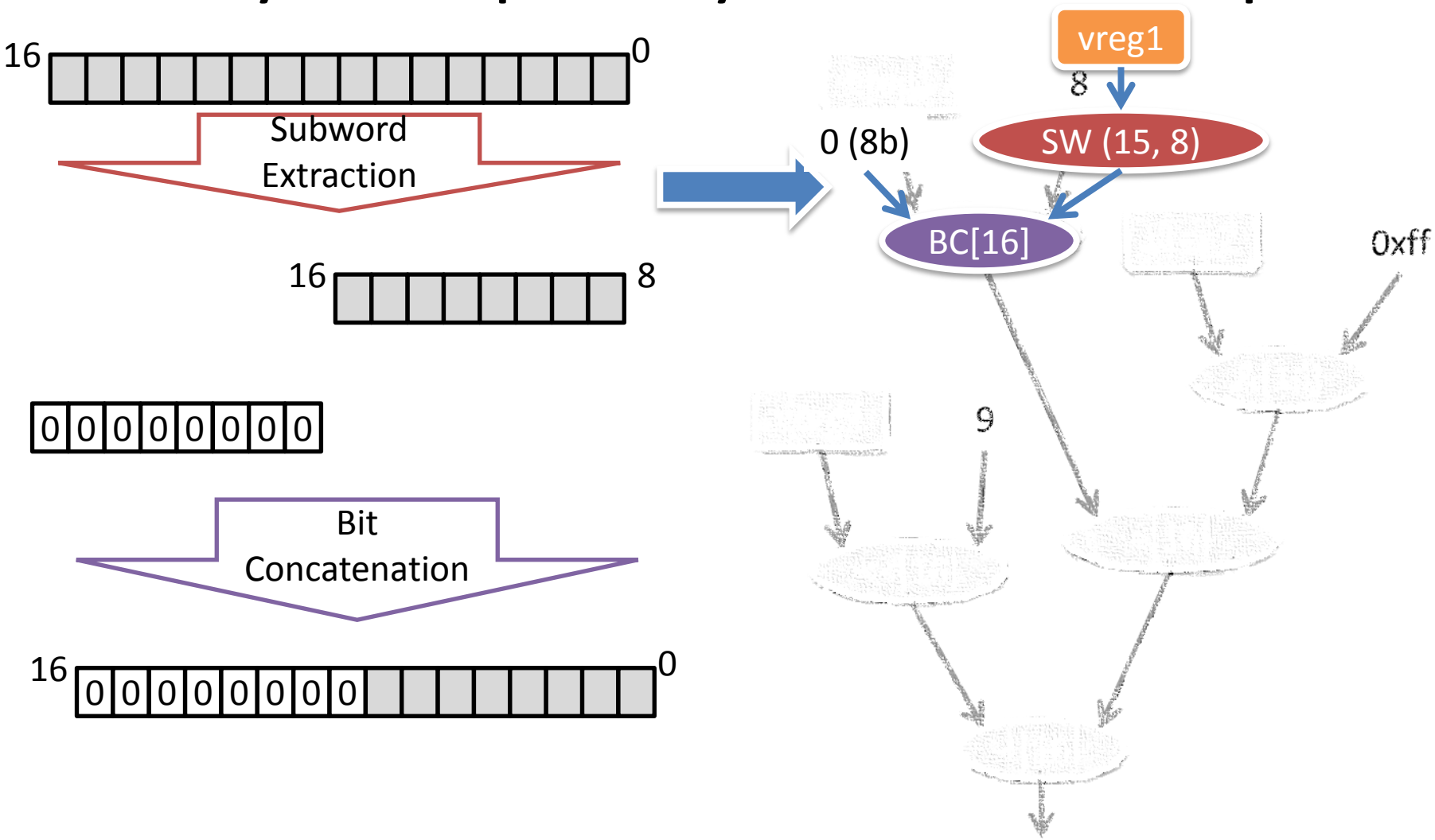


Early Data-path Synthesis - Example

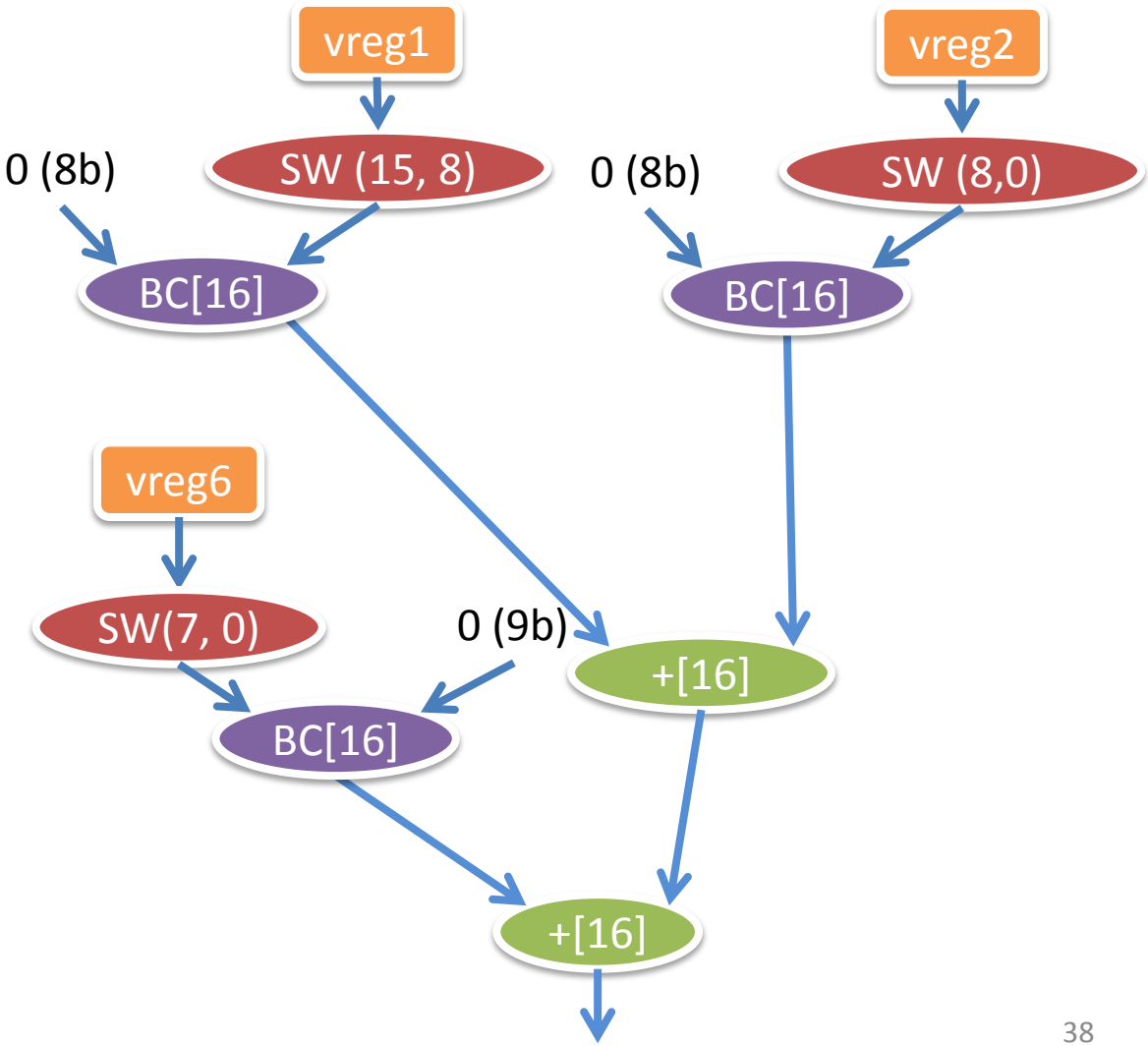




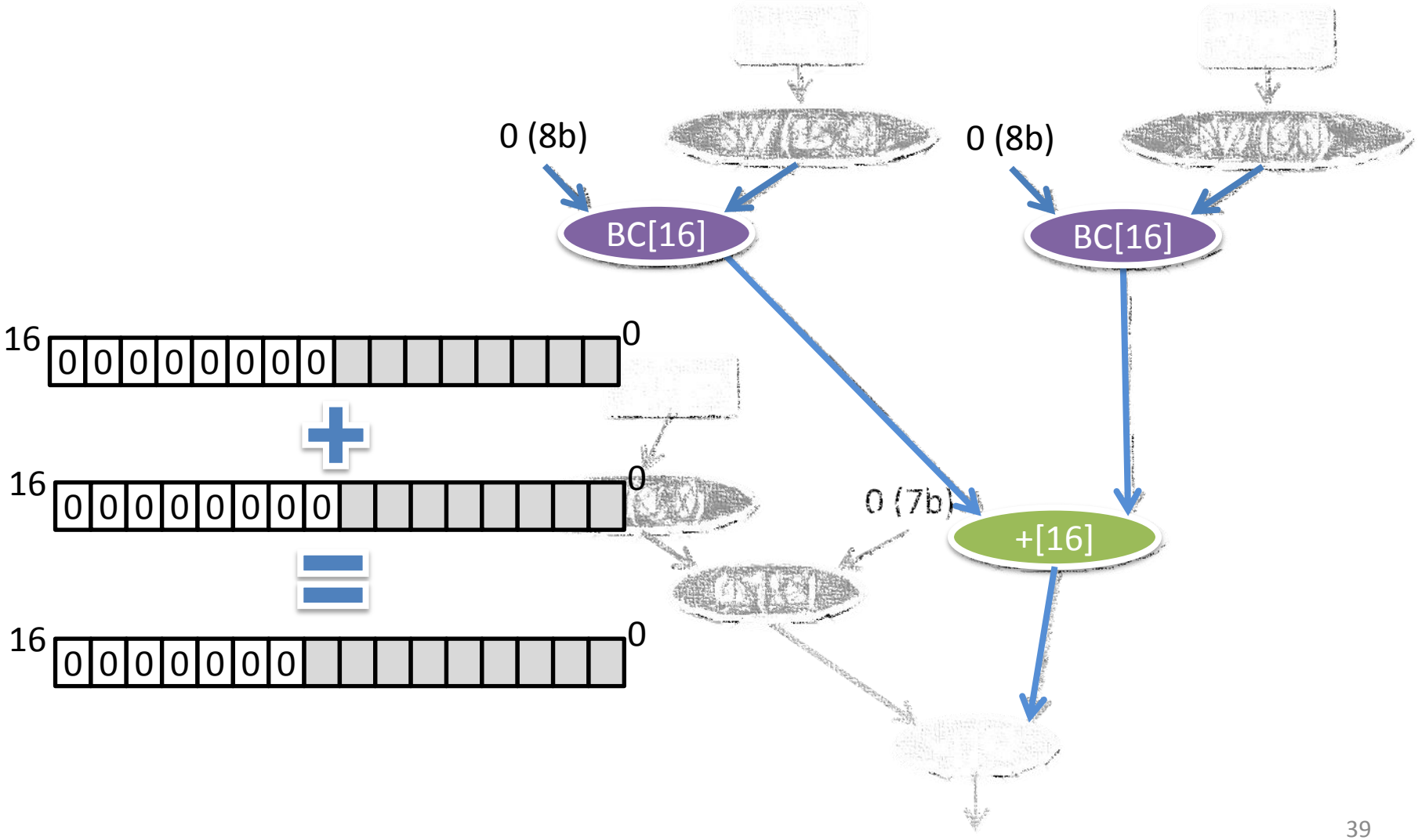
Early Data-path Synthesis - Example



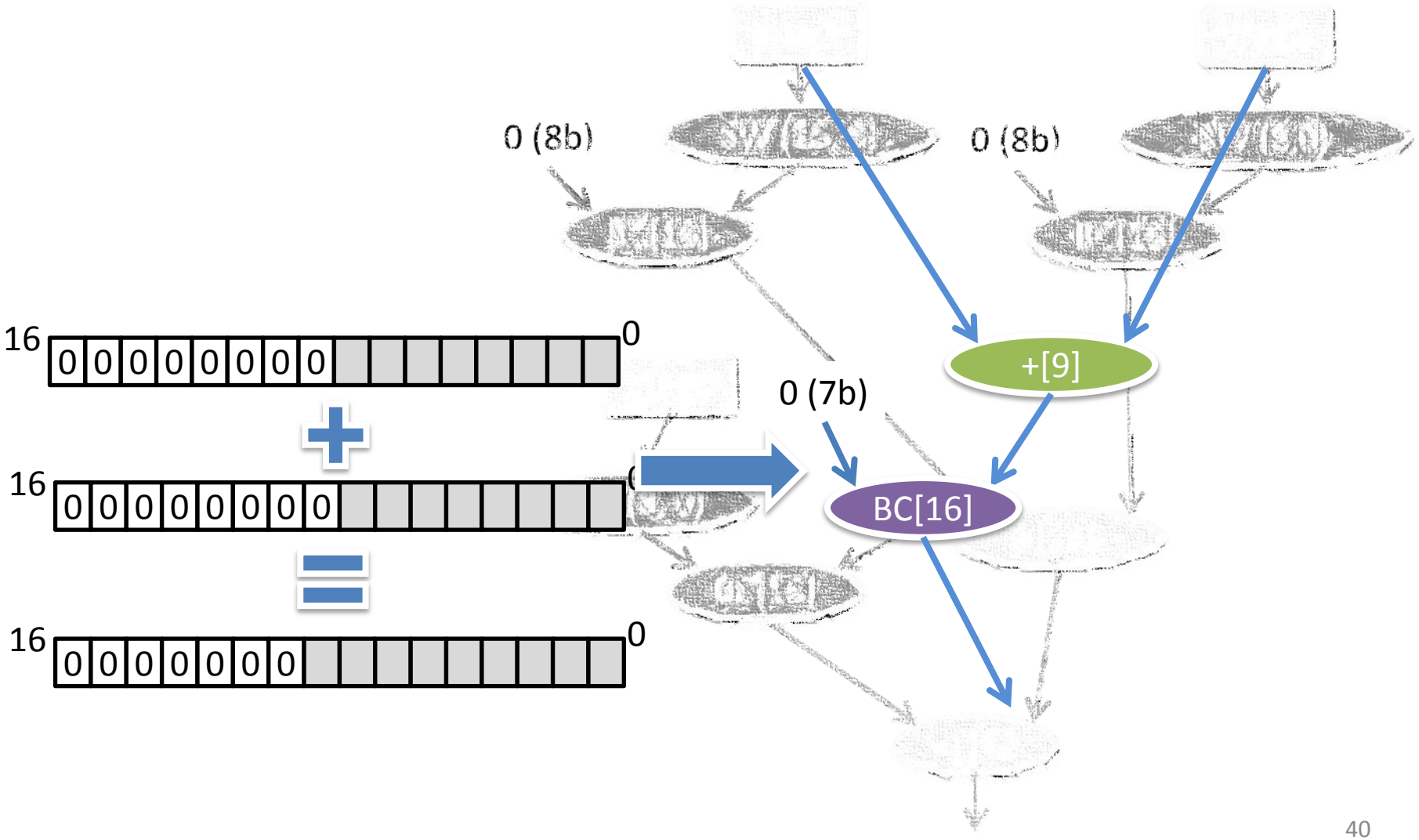
Early Data-path Synthesis - Example



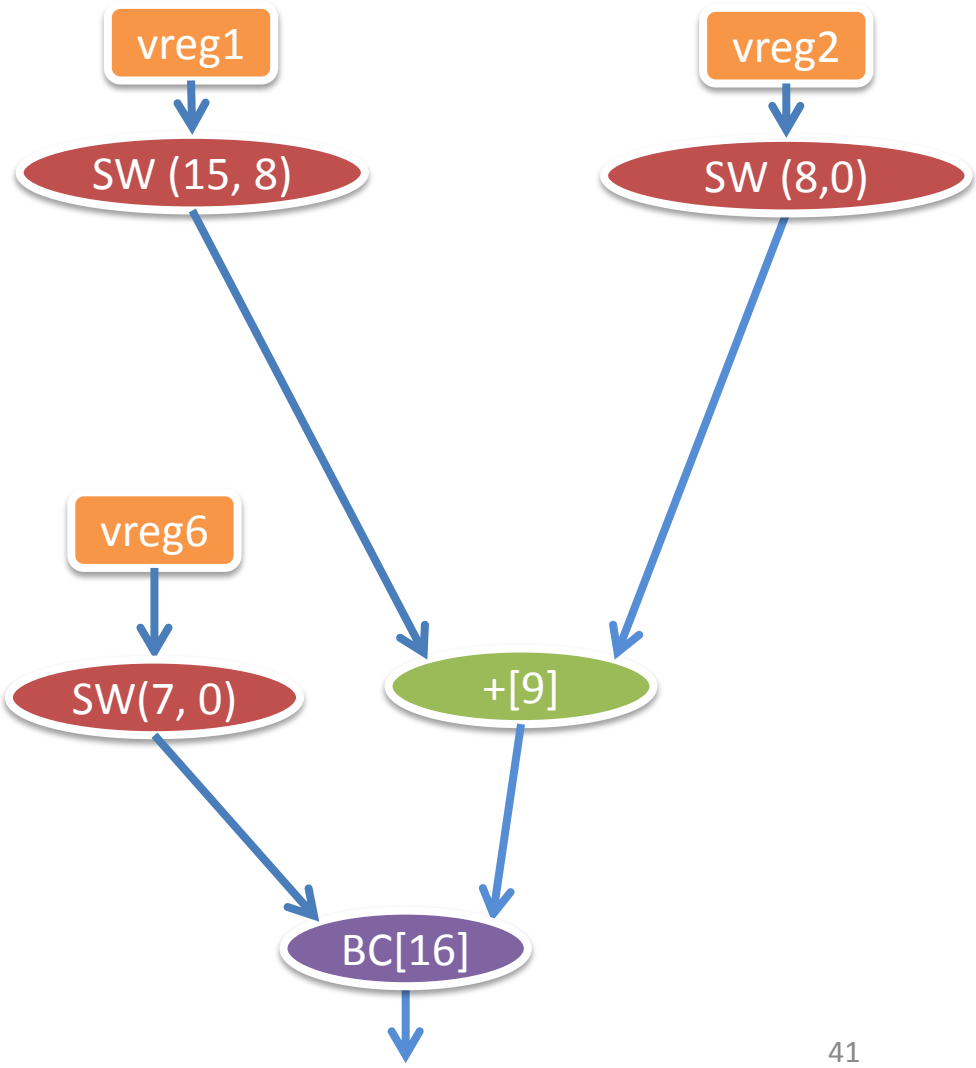
Early Data-path Synthesis - Example



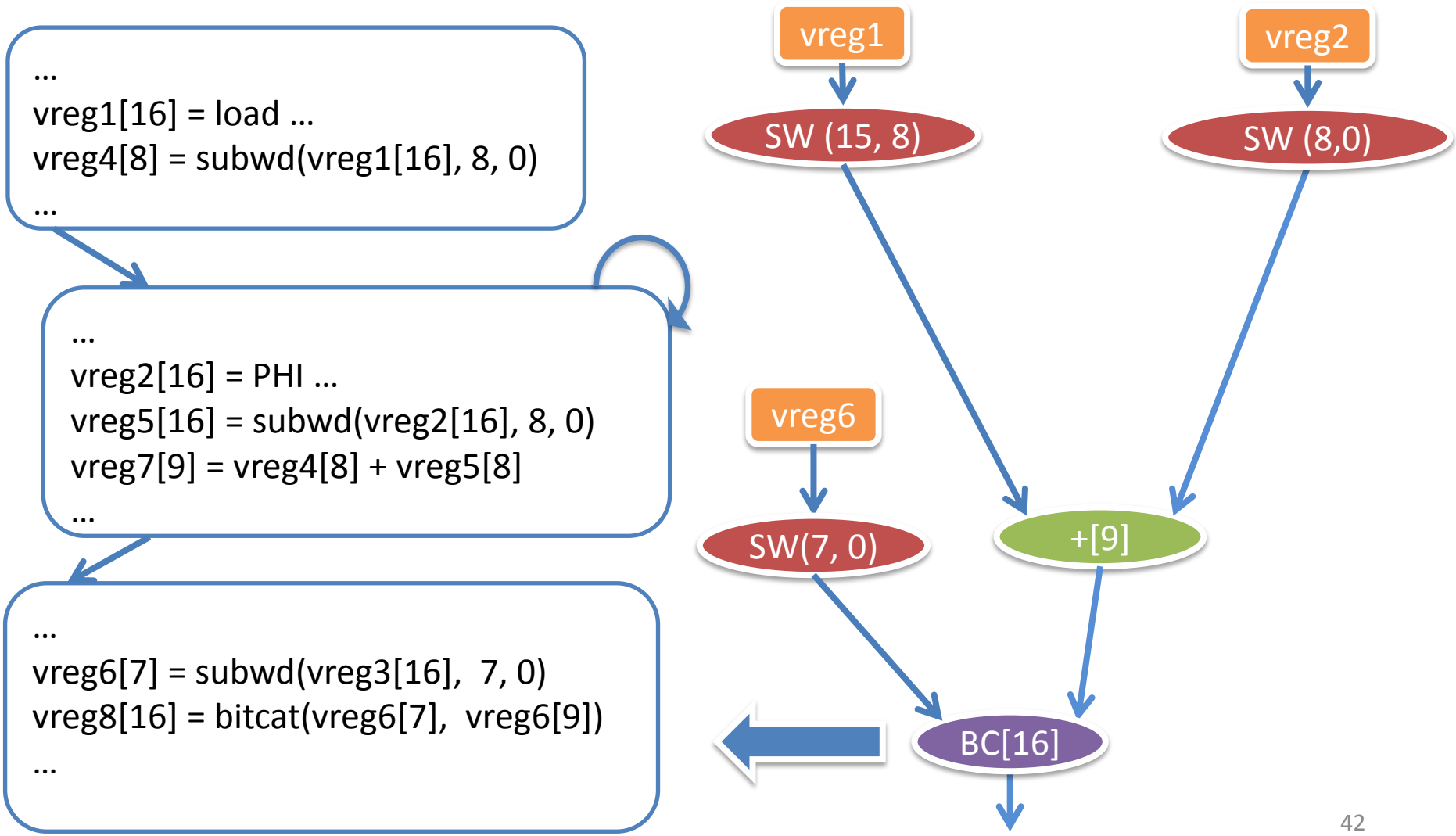
Early Data-path Synthesis - Example



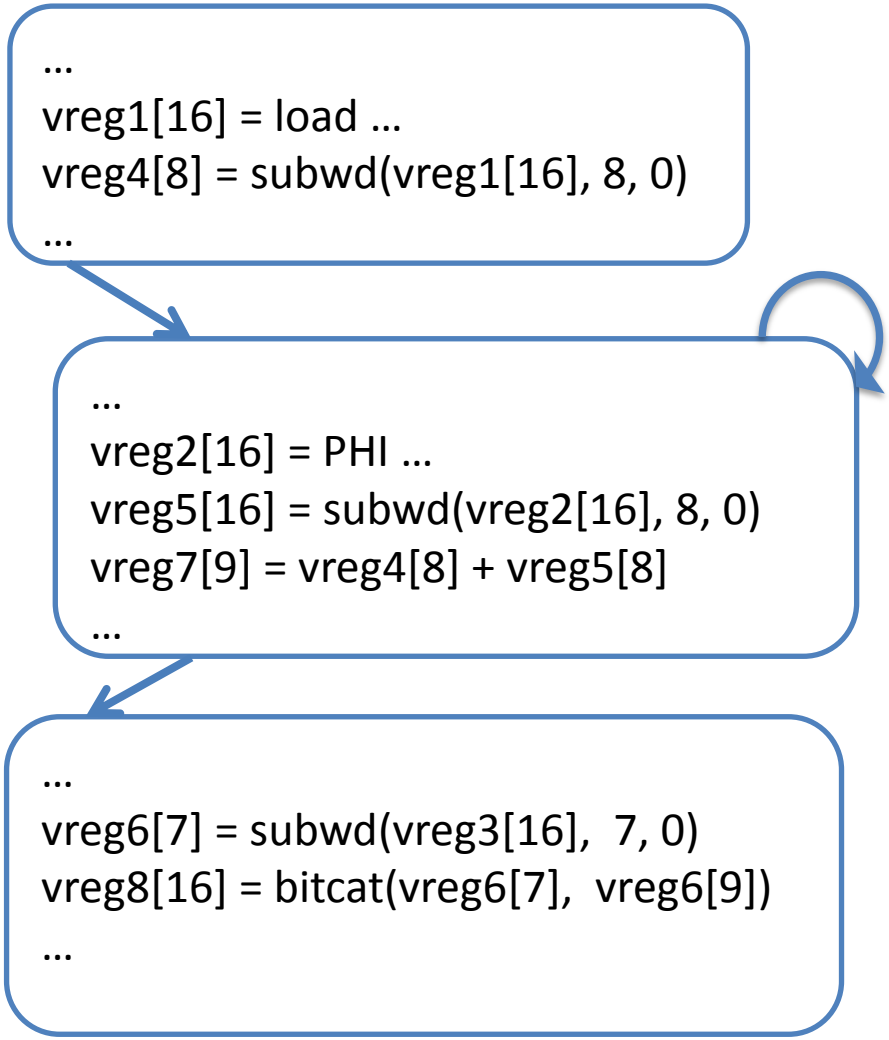
Early Data-path Synthesis - Example



Early Data-path Synthesis - Example

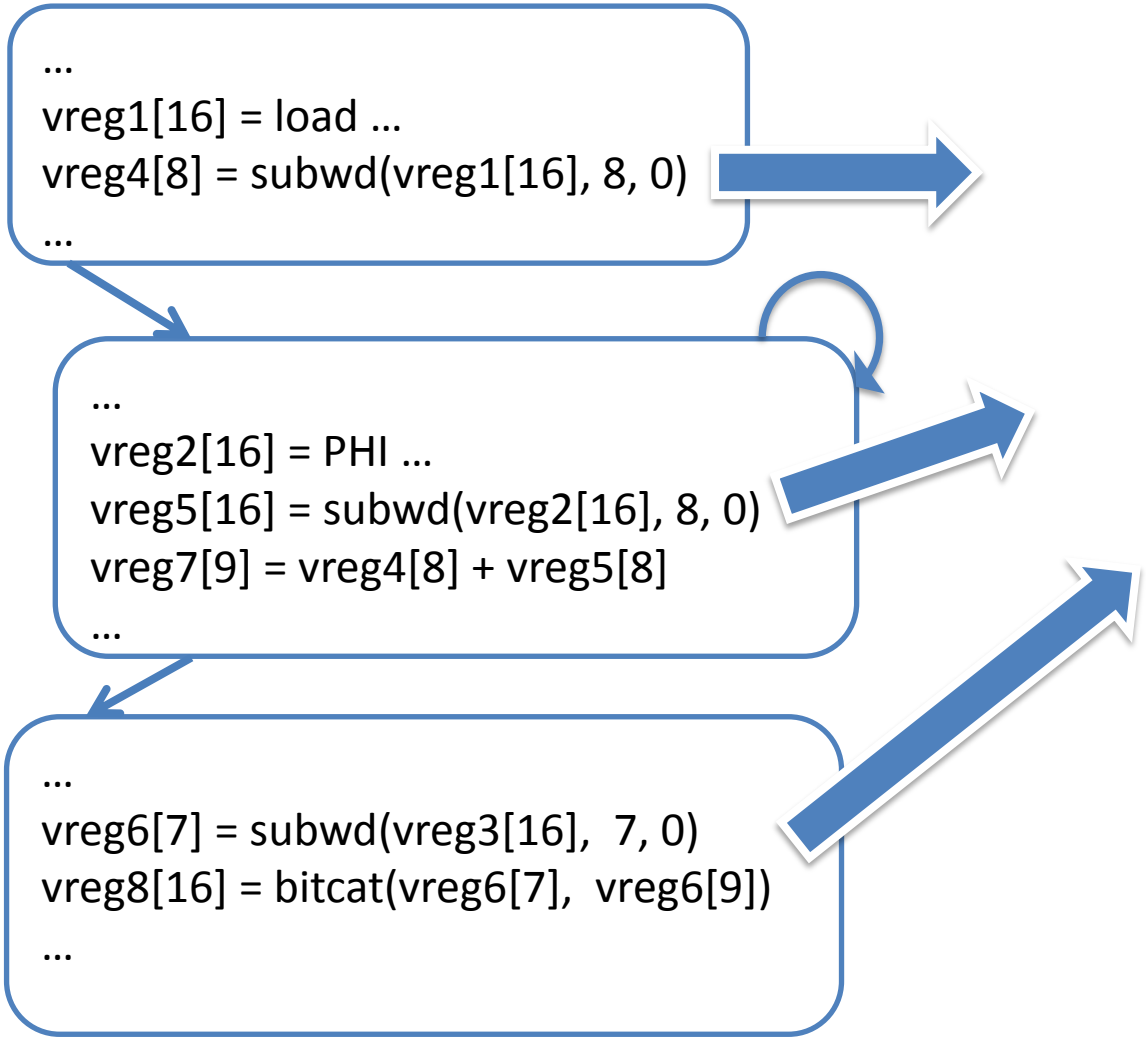


Early Data-path Synthesis - Example



- Original Resource usage:
  - 2x 16-bit adder
- Optimized:
  - 1x 9-bit adder
- Original Critical-Path delay:
  - 16-bit adder + 16-bit adder
- Optimized:
  - 9-bit adder

Early Data-path Synthesis - Example



Convenience provided by CodeGen:
- Directly represent the custom instructions

Early Data-path Synthesis

- Apply bit-level and subword-level optimizations, and LUT mapping (by ABC).
  - In fact, these optimizations are available in the implementation tools.

Early Data-path Synthesis

- Apply bit-level and subword-level optimizations, and LUT mapping (by ABC).
  - In fact, these optimizations are available in the vendor implementation tools.
- Transform the design representation toward final form.
  - Provide better delay estimation to the scheduler.
  - Provide better cost estimation to the binder.

Outline

- Introduction
- Overview of the Shang HLS framework
- The Cross-level Engine
- Kernel-only Software Pipelining
- Experimental Results and Further Work

Kernel-only Software Pipelining

- Do not need to generate the Prologue and Epilogue.

Kernel-only Software Pipelining

- Do not need to generate the Prologue and Epilogue.
- This reduce “the size of the code”.
  - Reduce the pressure to binding algorithm.

Kernel-only Software Pipelining

- Do not need to generate the Prologue and Epilogue.
- This reduce “the size of the code”.
  - Reduce the pressure to binding algorithm.
- Based on predicated execution.
  - All instructions are predicated in our VISA.

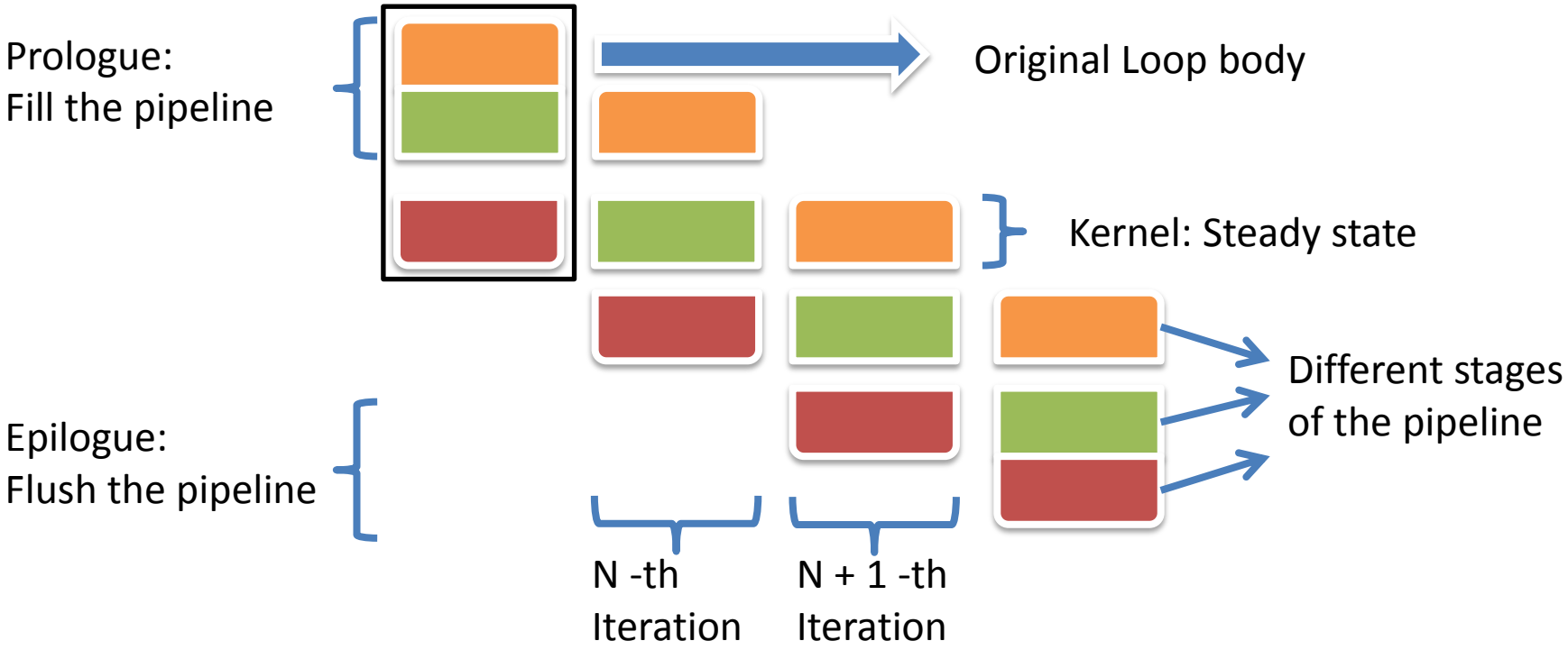
Kernel-only Software Pipelining

- Do not need to generate the Prologue and Epilogue.
- This reduce “the size of the code”.
  - Reduce the pressure to binding algorithm.
- Based on predicated execution.
  - All instructions are predicated in our V-ISA.

Convenience provided by CodeGen

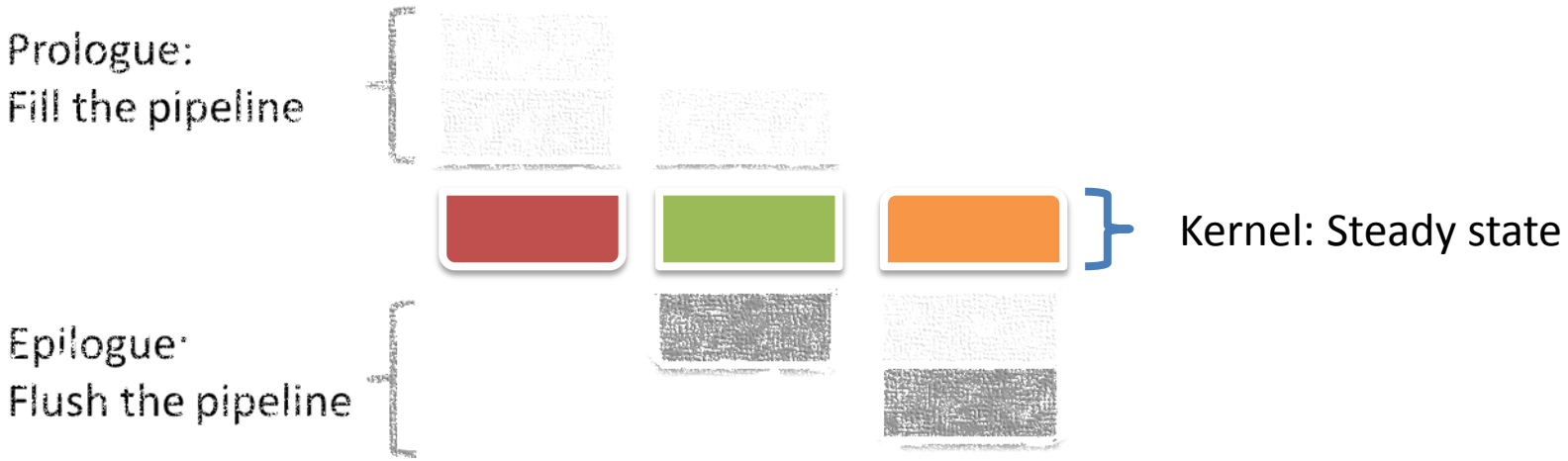
Software Pipelining

- Software pipelined loop execution:



Software Pipelining

- The kernel is already contains all stages, why we need to duplicate them in the Prologue and Epilogue?

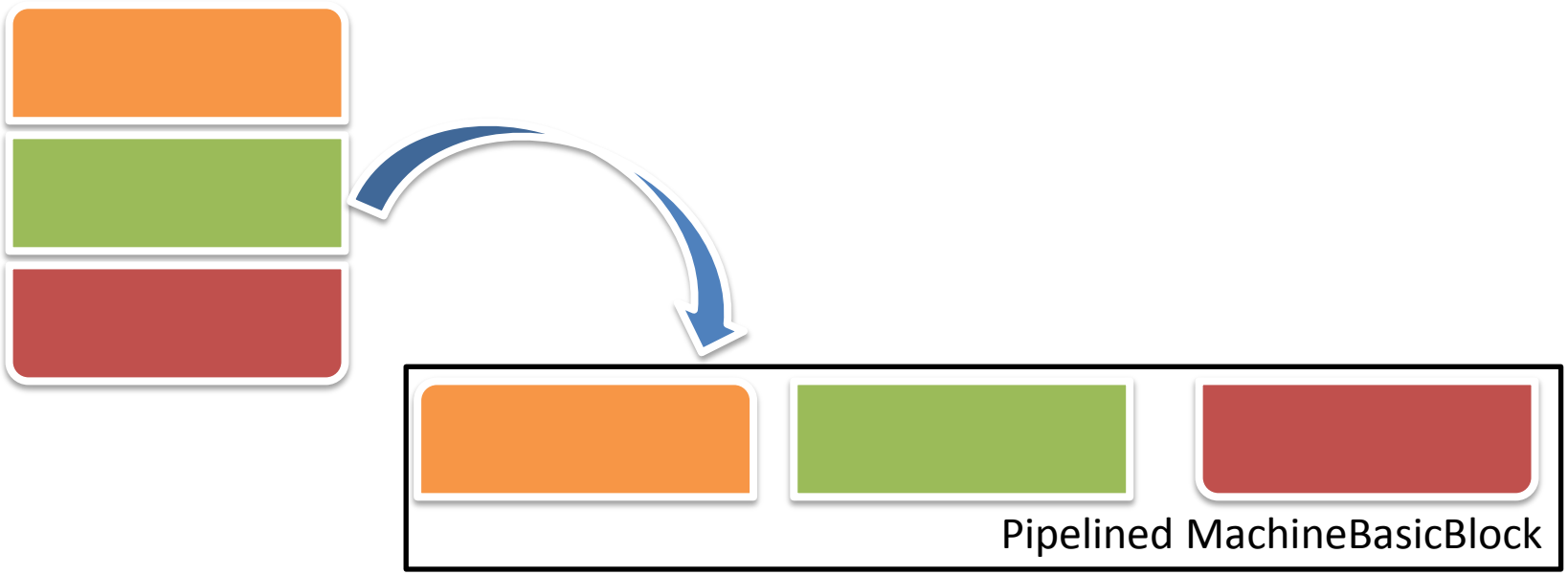


Loop Body Folding

- Loop body scheduled by Modulo Scheduling:

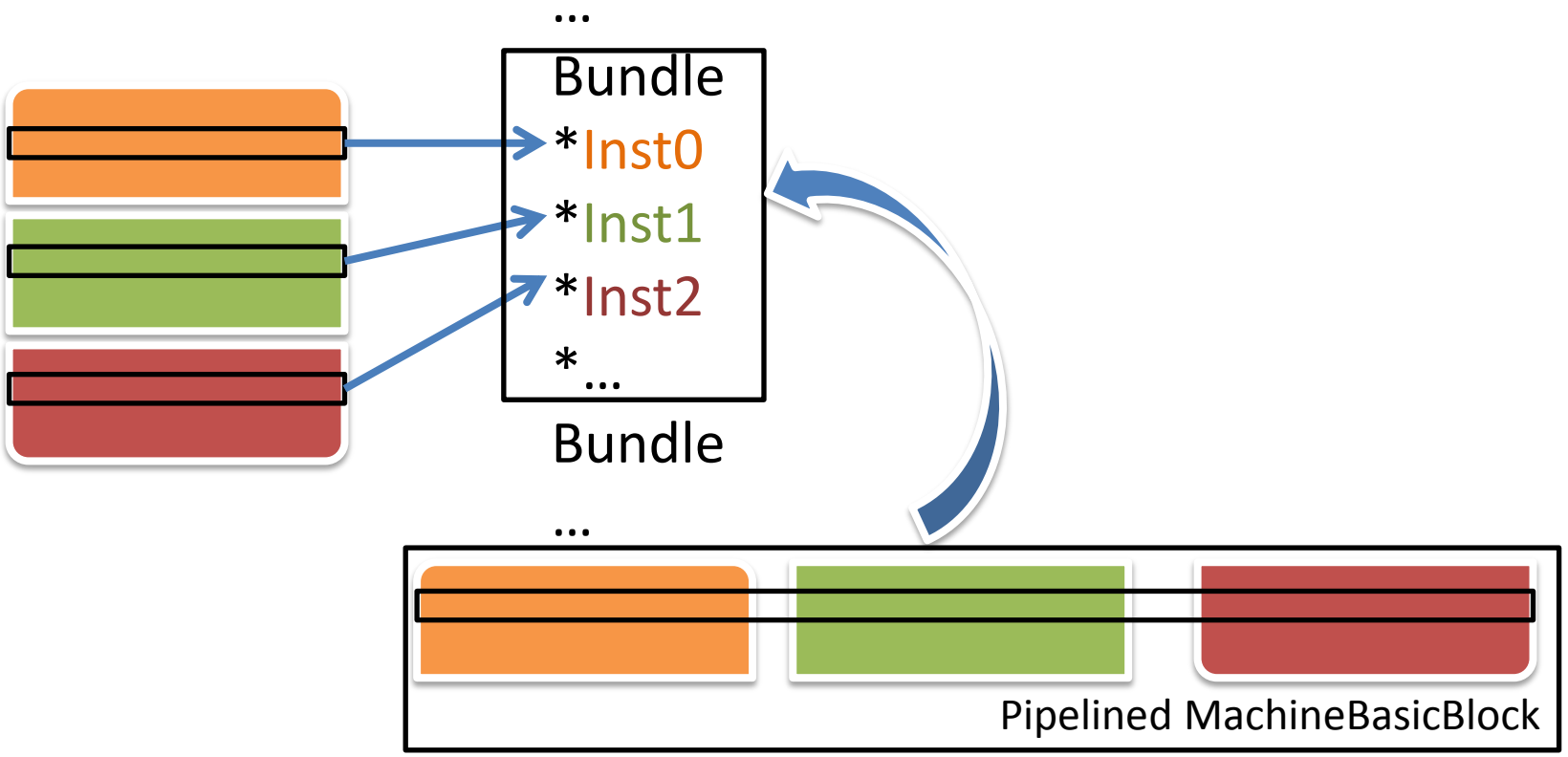
Loop Body Folding

- Fold the loop body to reflect the fact that the stages are executed in parallel.



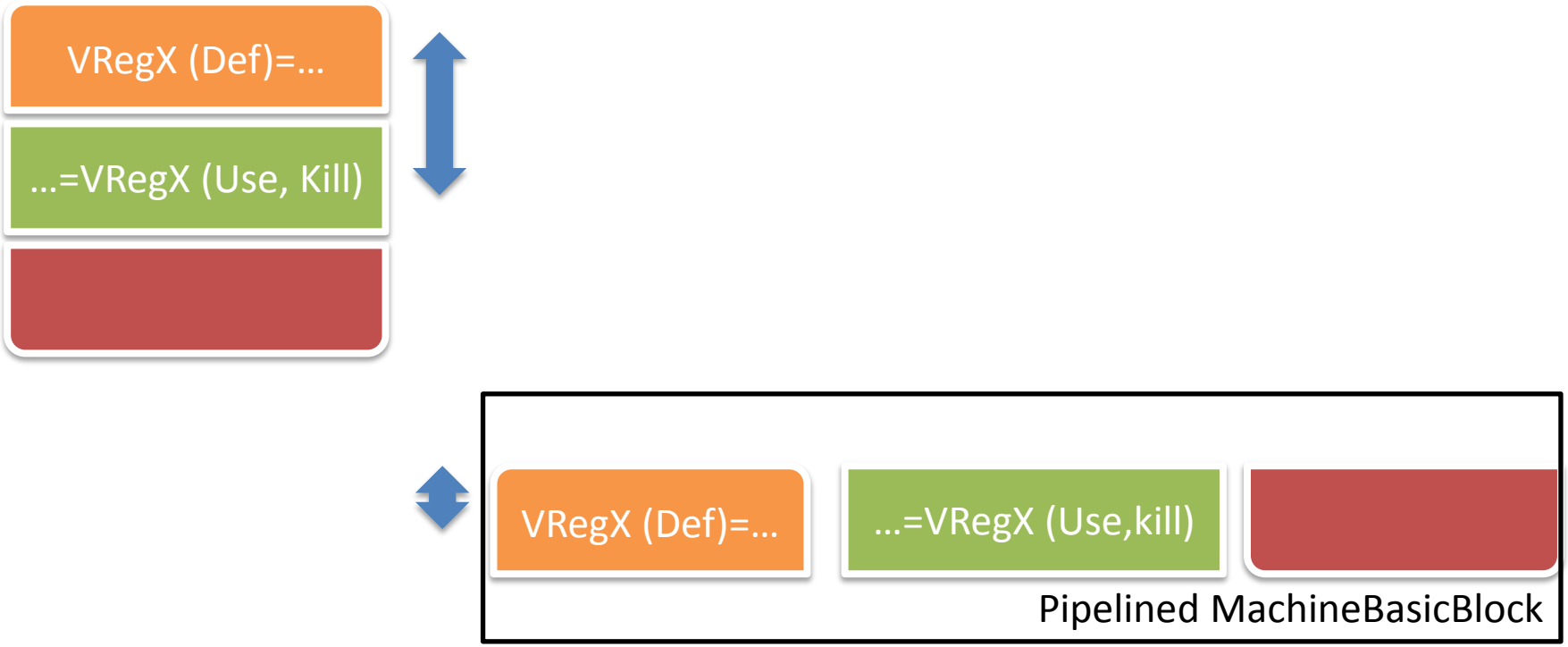
Loop Body Folding

- The Instructions in different stages are packed into the same bundle.



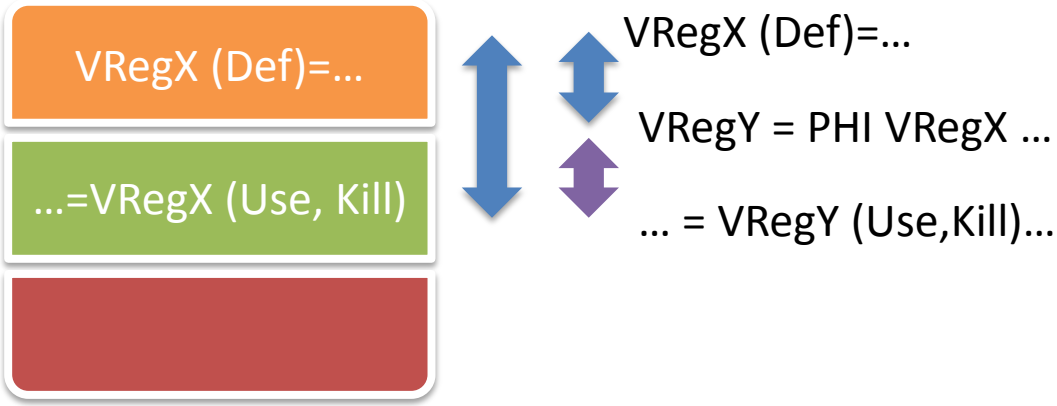
Loop Body Folding

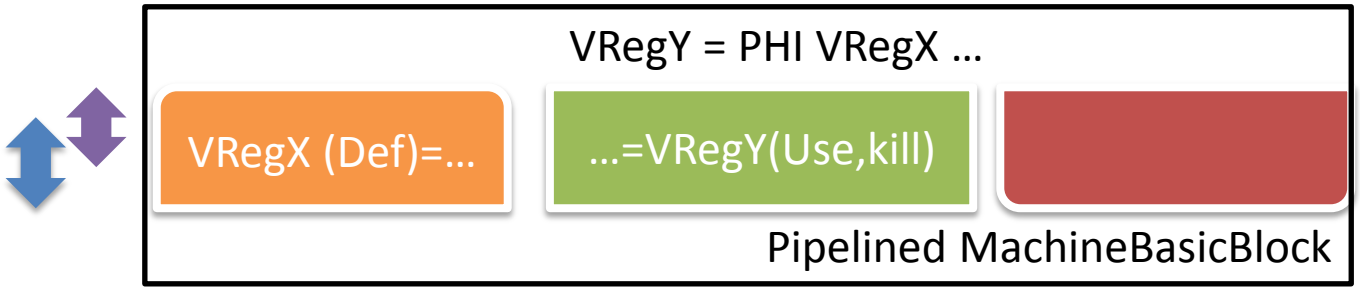
- VReg Def-Use chain across the pipeline stages:



Loop Body Folding

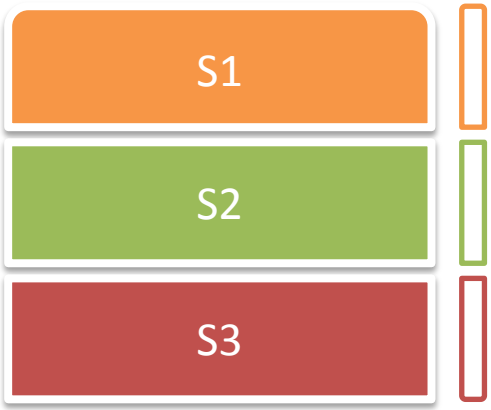
- PHIs are inserted to preserve SSA-form.





Loop Body Folding

- Predicate each stage:



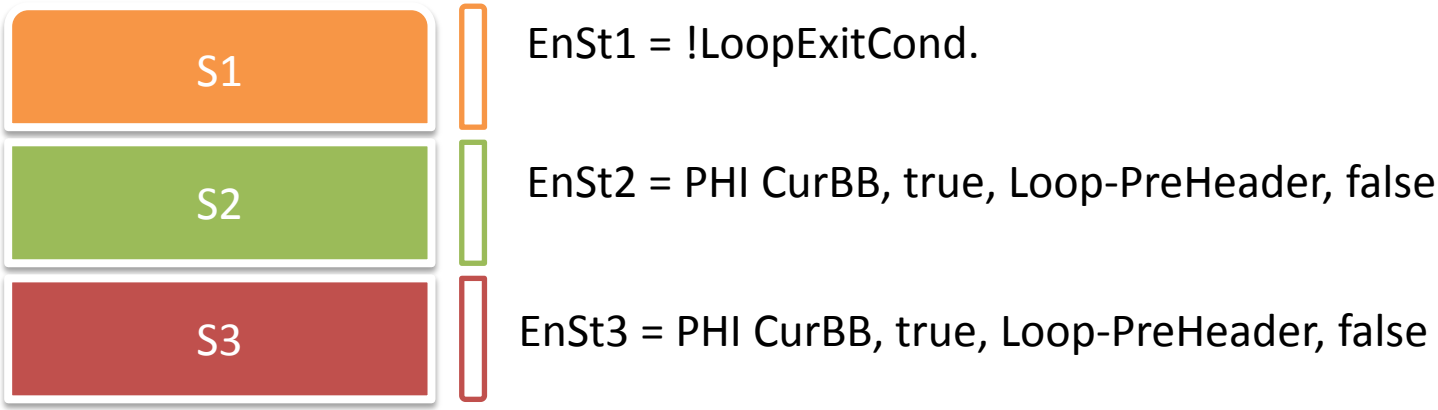
EnSt1 = !LoopExitCond.

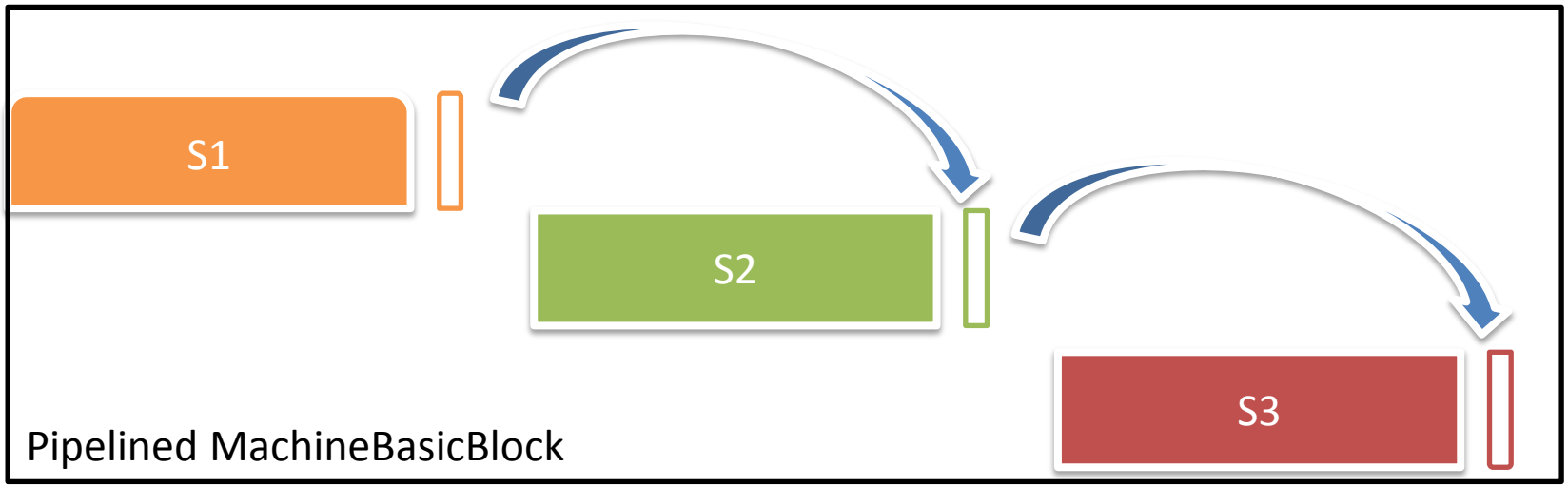
EnSt2 = PHI CurBB, true, Loop-PreHeader, false

EnSt3 = PHI CurBB, true, Loop-PreHeader, false

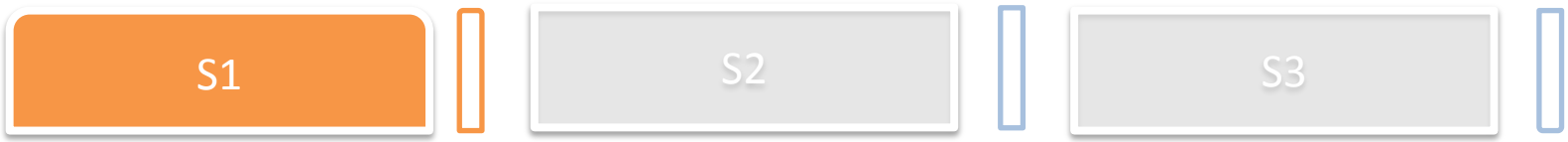
Loop Body Folding

- Predicate each stage:

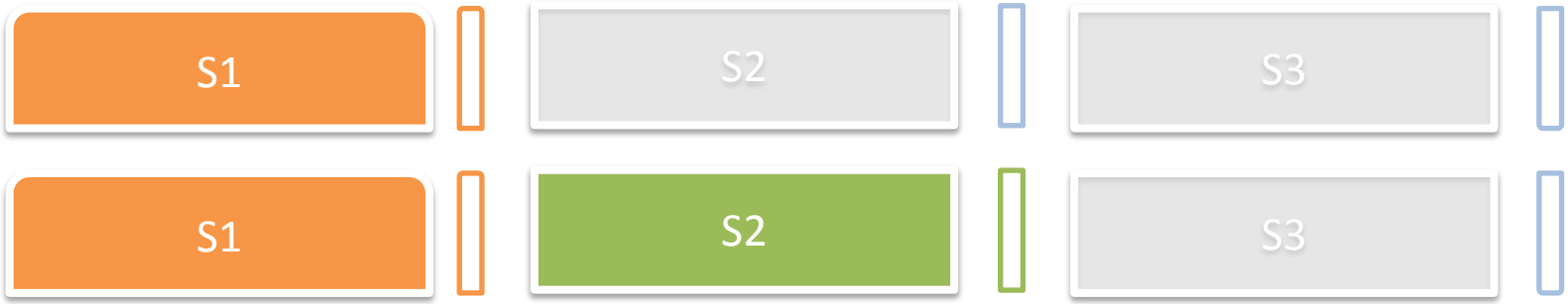




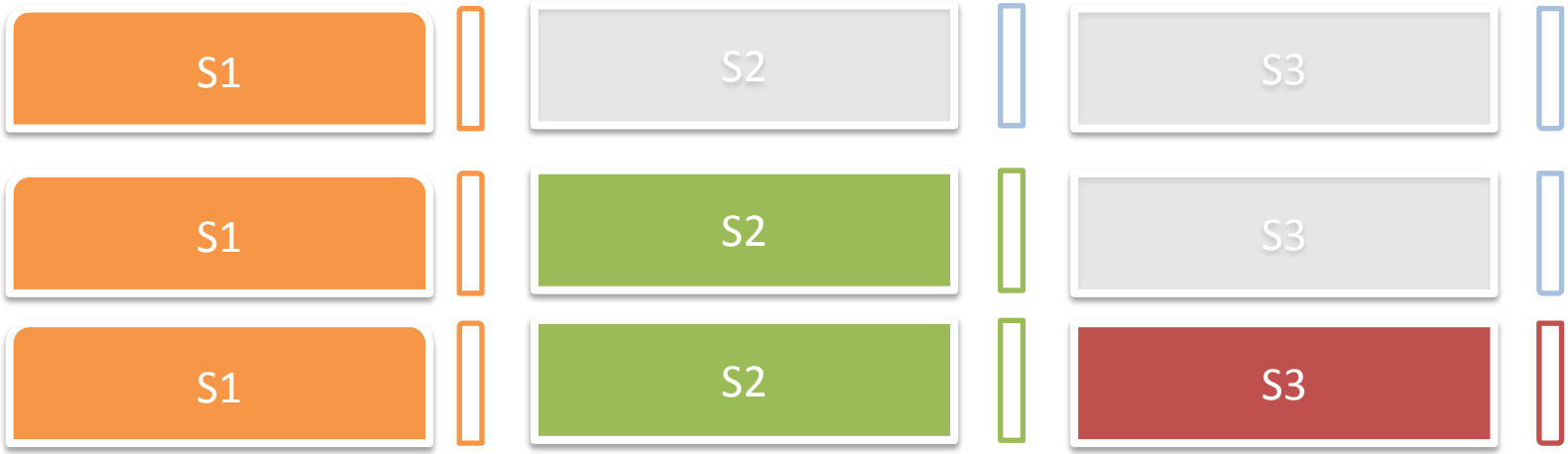
Execution of the Folded Loop Body



Execution of the Folded Loop Body



Execution of the Folded Loop Body



Execution of the Folded Loop Body

Execution of the Folded Loop Body

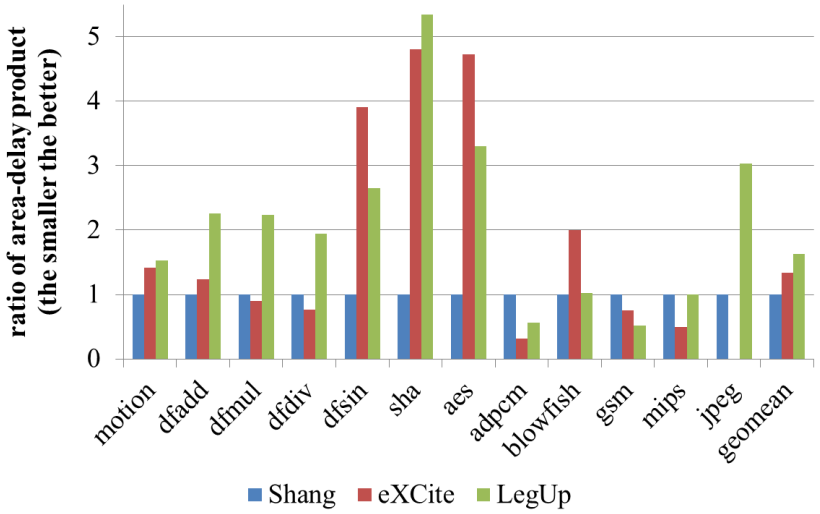
Execution of the Folded Loop Body

Outline

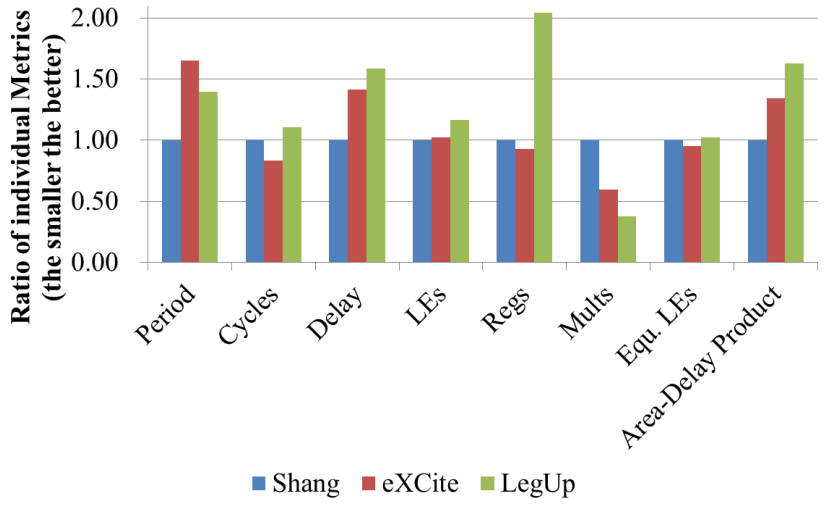
- Introduction
- Overview of the Shang HLS framework
- The Cross-level Engine
- Kernel-only Software Pipelining
- Experimental Results and Further Work

Experimental Results

For each programs in CHStone



For each metrics of the HW



Delay (us) = Period (ns) * Cycles / 1000
Equ. Les = LEs + Mults * 115
Where 115 is the LEs required to implement
A 9x9 mult.

Wish List and Further work

- Access more than one MachineFunction at a time.
- Interprocedural Analyses/Optimizations with Polly.
- Compile flow for heterogeneous architecture.
- HLS-specific IR passes, e.g. do not duplicate function body when inlining.

Acknowledgements

- Thanks the people invoked in this project.
  - Qingrui Liu, Junyi Le, Yuelai Yuan, …
- Thanks the LLVM community to provide the compiler infrastructure.
- Thanks SYSU to support this work.
- Thanks ADSC to pay my salary.
  - So that I can buy the flight ticket.

THANKS AND QUESTIONS?