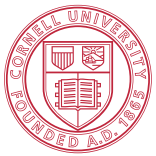
Area-Efficient Pipelining for FPGA-Targeted High-Level Synthesis

Ritchie Zhao, Mingxing Tan, Steve Dai, Zhiru Zhang

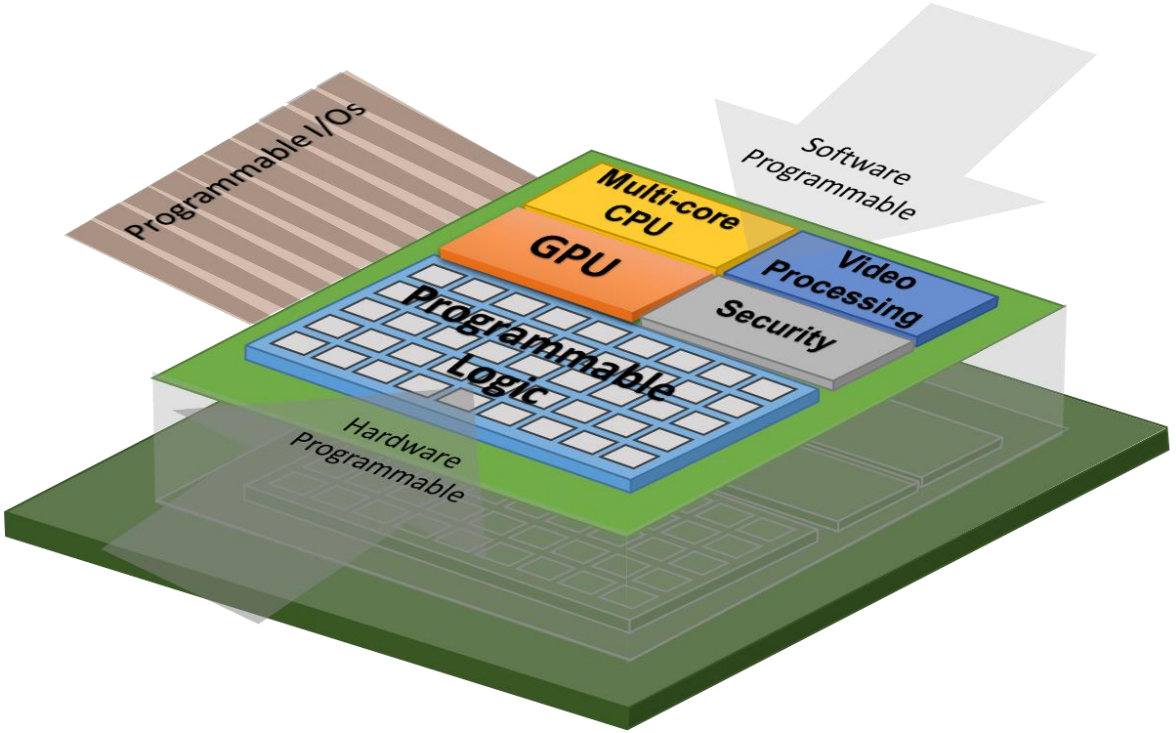
Computer Systems Lab,
Electrical and Computer Engineering,
Cornell University



Cornell University

CSL

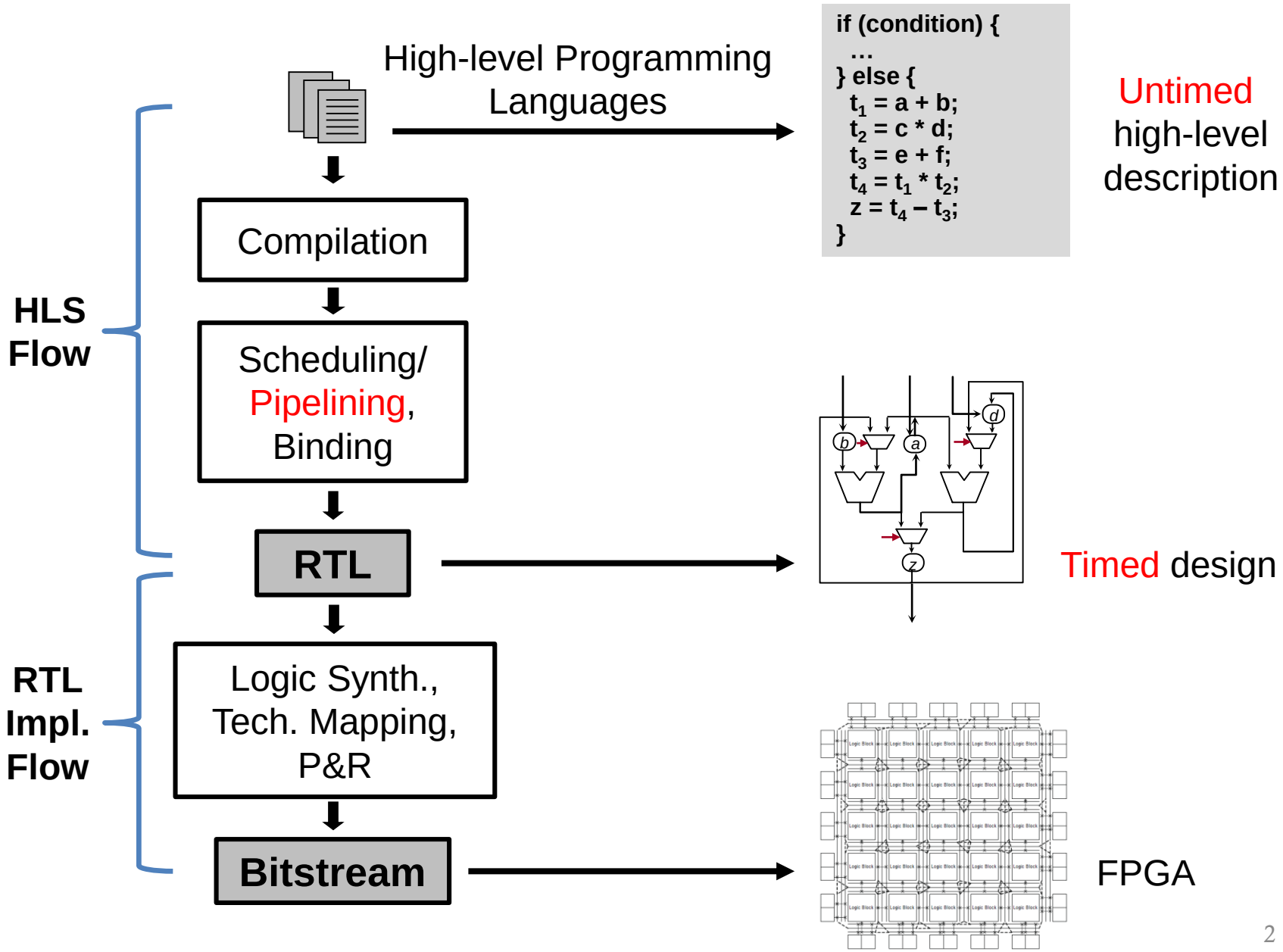
High-Level Synthesis (HLS) for FPGAs



- FPGAs are making breakthroughs into the software computing domain

- HLS is key to enabling software-programmable FPGAs

A Typical HLS Flow

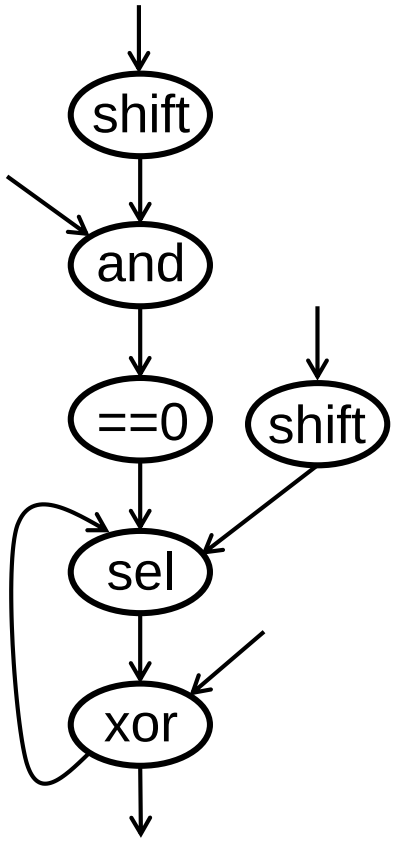


Motivating Example

Reed-Solomon Decoder

```
for (i=0; i<8; i++) {
  if (b & (1<<i))
    temp ^= (a << i);
}
```

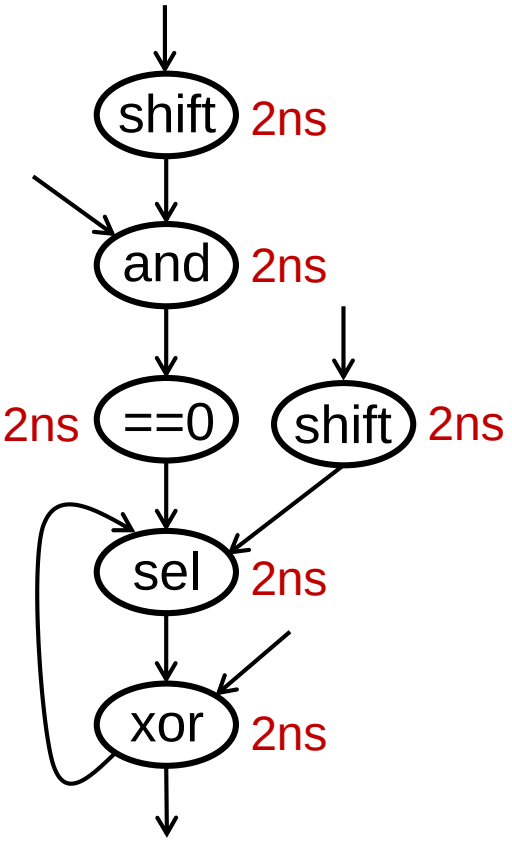
Control-Data Flow Graph



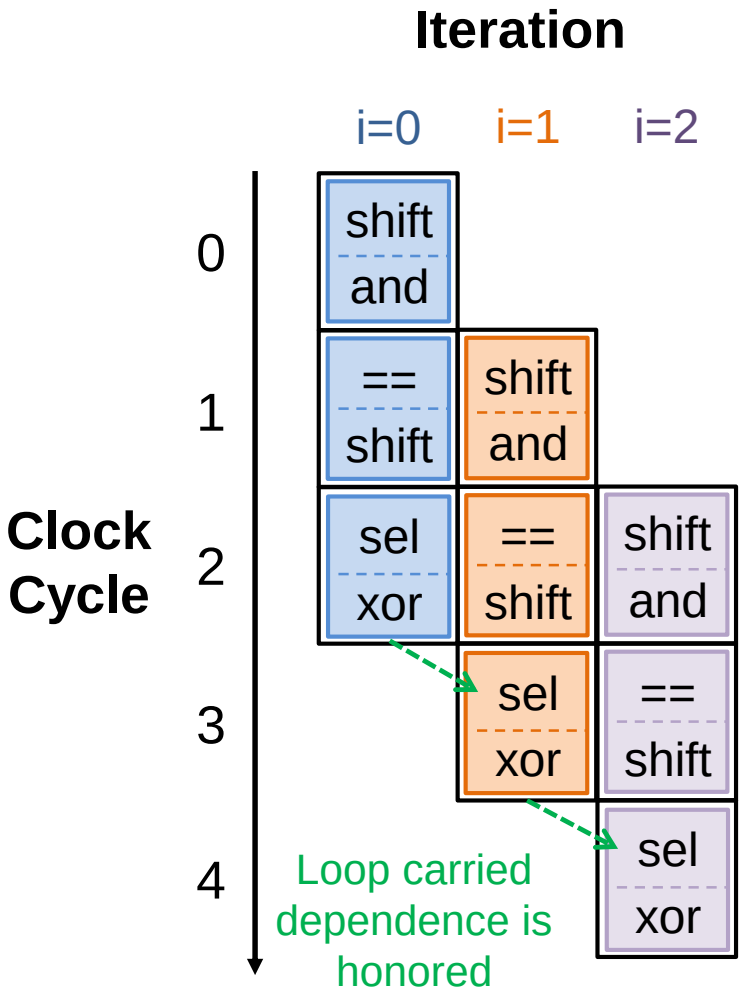
Pipelining

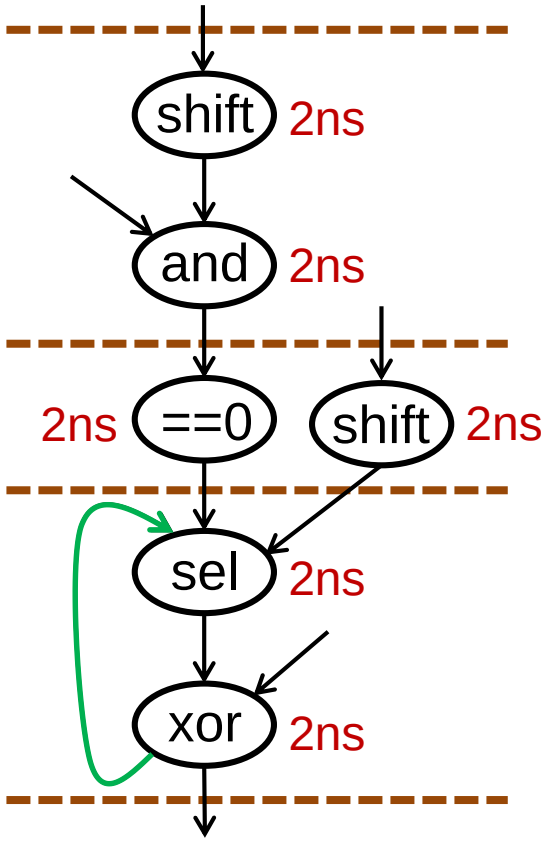
- Commonly implemented in HLS as *modulo scheduling*
- Goal: create a schedule that can be repeated every *initiation interval* (II) cycles
- Each operation has a **pre-characterized** delay

- **In this example:**
  - Operation delay = 2ns
  - Clock period = 5ns
  - Target II = 1



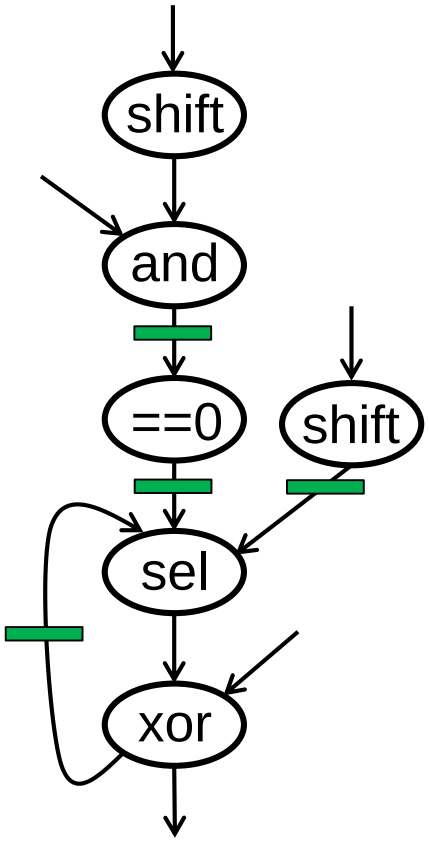
Pipelining


Iteration
i=0
i=1
i=2
Clock Cycle
0
1
2
3
4
shift
and
==
shift
shift
and
sel
xor
==
shift
shift
and
sel
xor
==
shift
sel
xor
Loop carried dependence is honored


shift
2ns
and
2ns
2ns
==0
shift
2ns
sel
2ns
xor
2ns

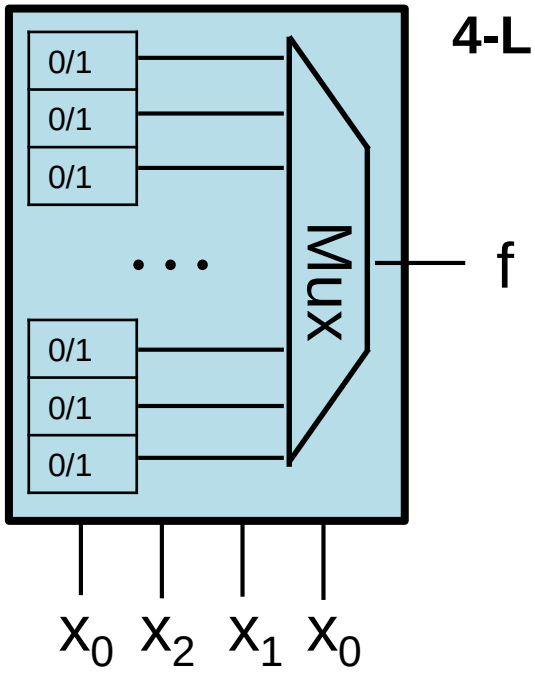
Resulting Datapath

- **Is this fully optimized for an FPGA?**
- **Problem:**
  Pre-characterized delays are acceptable for arithmetic operations, but are inaccurate for logical operations



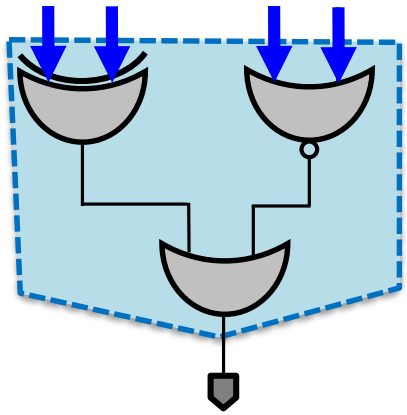
LUT Mapping

- Look-Up Tables (LUTs) are the building blocks of FPGAs
- A k-input LUT (k-LUT) can implement any k-input 1-output combinational logic network
  - A k-input 1-output subgraph is called a **k-feasible cone**



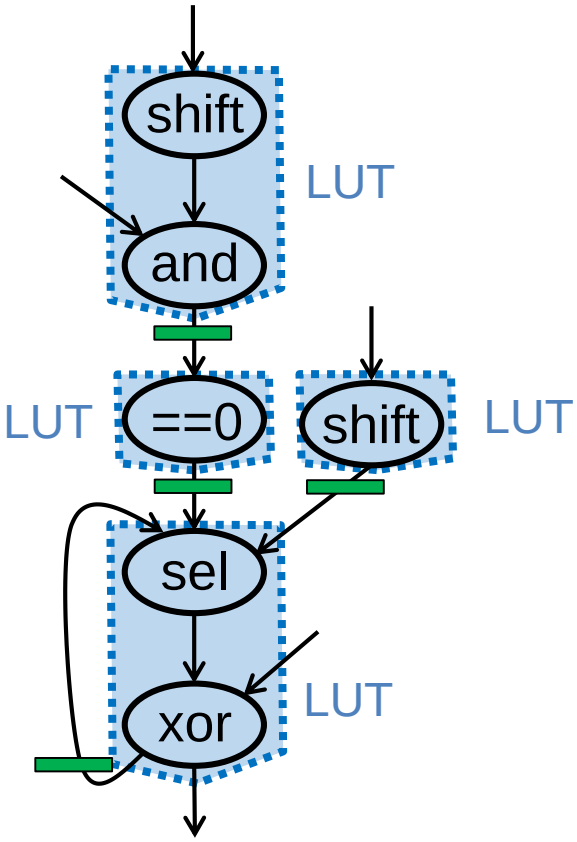
4-LUT

4-feasible cone



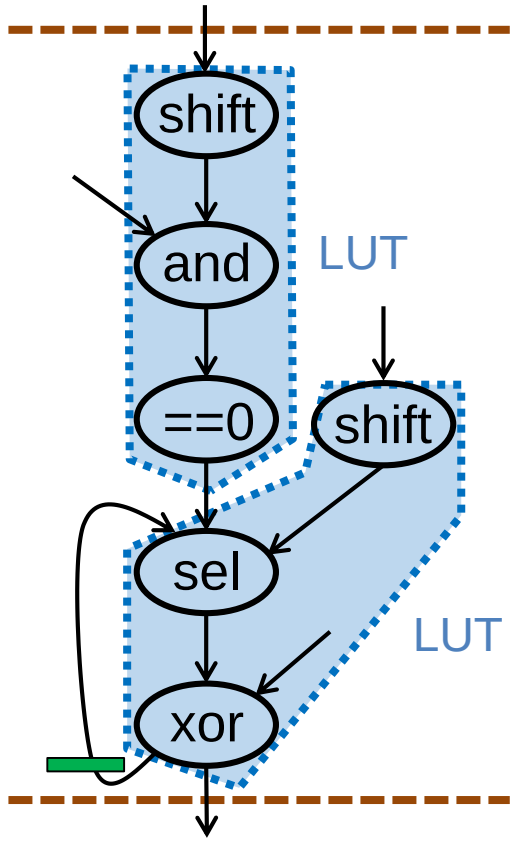
LUT Mapping

- **Assumptions**
  - 4-input LUTs
  - Single-bit operations
- LUT mapping occurs between fixed register boundaries
- Resource count:
  - 4 LUTs
  - 4 registers



A Better Solution

- **Assumptions**
  - LUTs have 2ns delay
  - Clock period = 5ns
- All operations can be mapped to 2 LUTs and 1 cycle
- **Resource count:**
  - 2 LUTs
  - 1 register

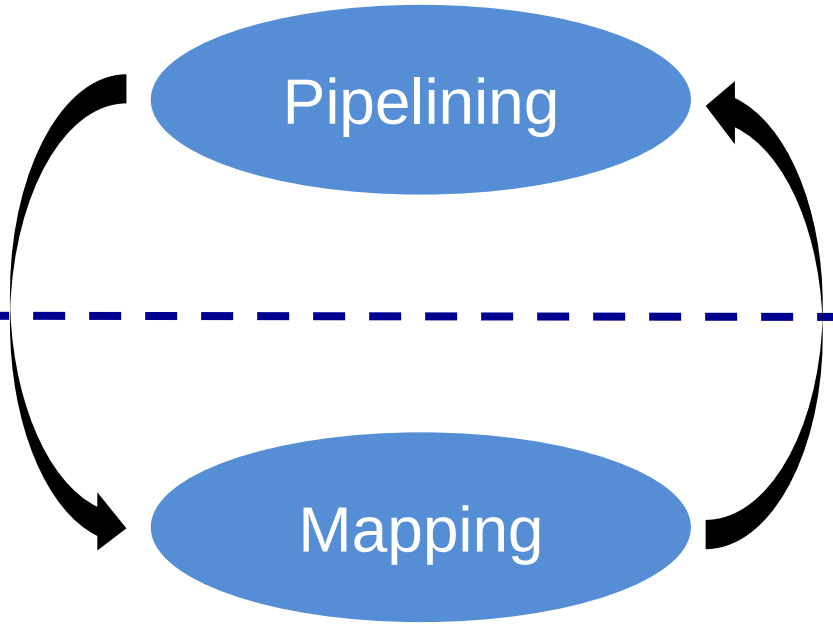


Scheduling and Mapping Interdependence

HLS

Determine register boundaries

⇒ Inaccurate delay model due to lack of mapping awareness



RTL Impl.

Determine LUT mapping

⇒ Accurate delay model, but cannot change register boundaries

Mapping-Aware Pipelining

Key Contributions

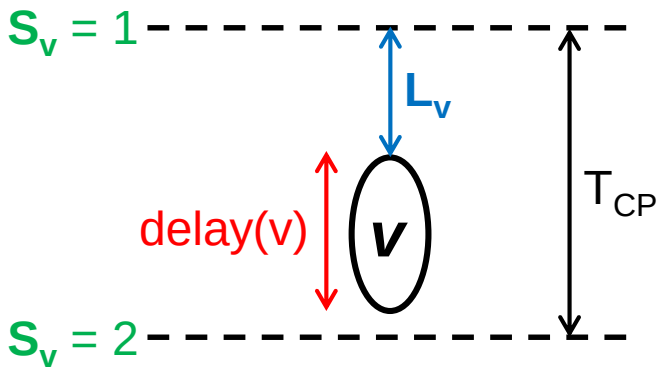
- Perform **mapping-aware pipelining** to generate schedules which are more amenable to LUT mapping optimizations
- First exact formulation of mapping-aware pipelining using a mixed integer linear program (MILP)

Results

- Significant area reduction on logic-heavy designs

MILP Formulation

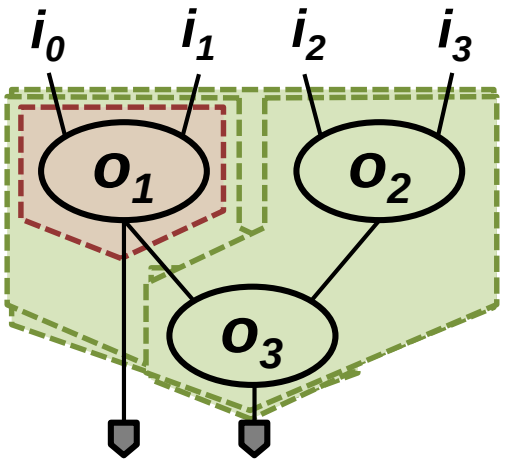
- Model both inter-cycle (S_v) and intra-cycle ($\mathbf{L_v}$) start time of each operation



- **Clock Period and Dependence Constraints:**
  - $L_v + delay(v) \leq T_{CP}$
  - $S_u - S_v - II * dist(u, v) \leq 0$
- **Hard Resource Constraints:**
  - Applies to operations mapped to DSPs, memory ports, etc.
  - Identical to canonical ILP formulation for pipelining

MILP Formulation

- **LUT Mapping Constraints:**
  - Pre-compute the feasible LUT mappings for each node
  - Ensures the MILP selects a LUT mapping for each node to cover the CDFG and generate each primary output



Constraints:

o_1 is the root of a LUT

o_3 is the root of a LUT

Feasible Mappings:

$\{o_1\}; \{o_2\}; \{o_3\}$

$\{o_1\}; \{o_2,o_3\}$

$\{o_1\}; \{o_1,o_3\}; \{o_2\}$

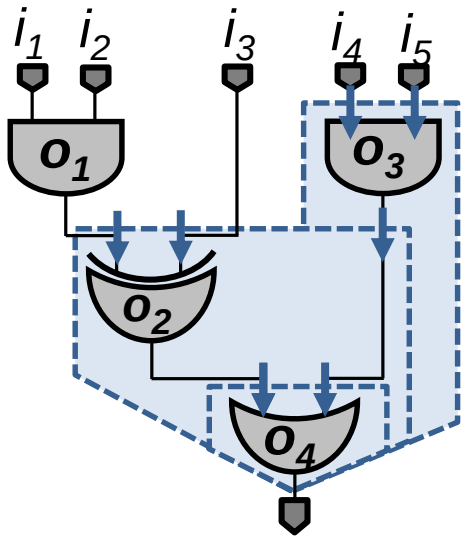
$\{o_1\}; \{o_1,o_2,o_3\}$

MILP Formulation

- **Optimization Objective:**
  - Minimize a weighted sum of LUTs, registers, and other resources
  - Possible to calculate LUTs and registers by considering mapping
- Unlike traditional formulations, we can minimize the exact physical resources used by the FPGA

Mapping Consideration

- Before setting up the MILP, we compute all k-feasible LUT mappings for each node (find the k-feasible cones)
- On a bit-level graph, a k-feasible cone is simply a cone with k or fewer inputs

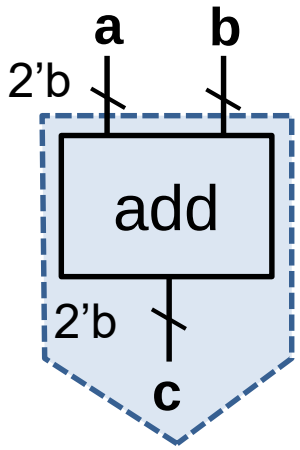


3-feasible cones for o_4 (listed by inputs):

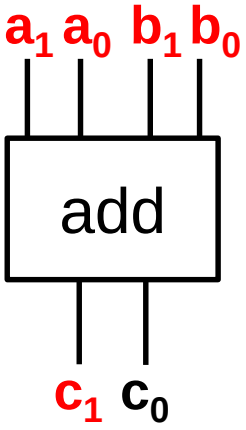
$\{o_2, o_3\}$
$\{o_1, i_3, o_3\}$
$\{o_2, i_4, i_5\}$

Mapping Consideration

- **Problem**: HLS operates on the word-level CDFG, where an output may depend on multiple bits of each input



c depends on {**a**,**b**}
2-feasible cone



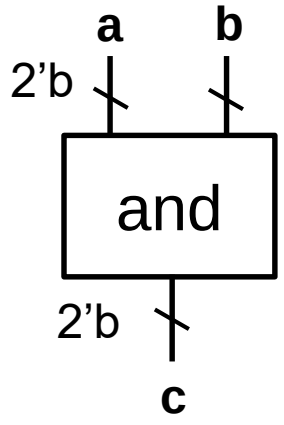
MSB $\mathbf{c_1}$ depends on
$\{\mathbf{a_1},\mathbf{a_0},\mathbf{b_1},\mathbf{b_0}\}$

We can decompose the word-level graph, but moving HLS to bit-level creates many complications

Bit-Level Dependence Tracking

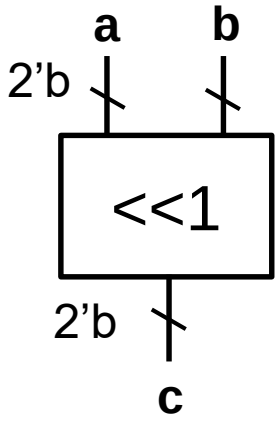
- **Solution:** Instead of just tracking inputs, track input + bit position (for example ***x*** becomes ***x*[i]**)
  - Consider three operation types

Logical



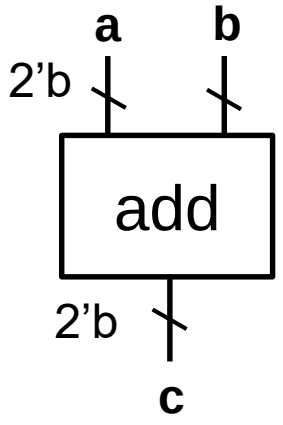
c depends on {**a**,**b**}
c[**i**] depends on
{**a**[**i**], **b**[**i**]}

Shift



c depends on {**a**,**b**}
c[**i**] depends on
{**a**[**i+1**], **b**[**i+1**]}

Arithmetic

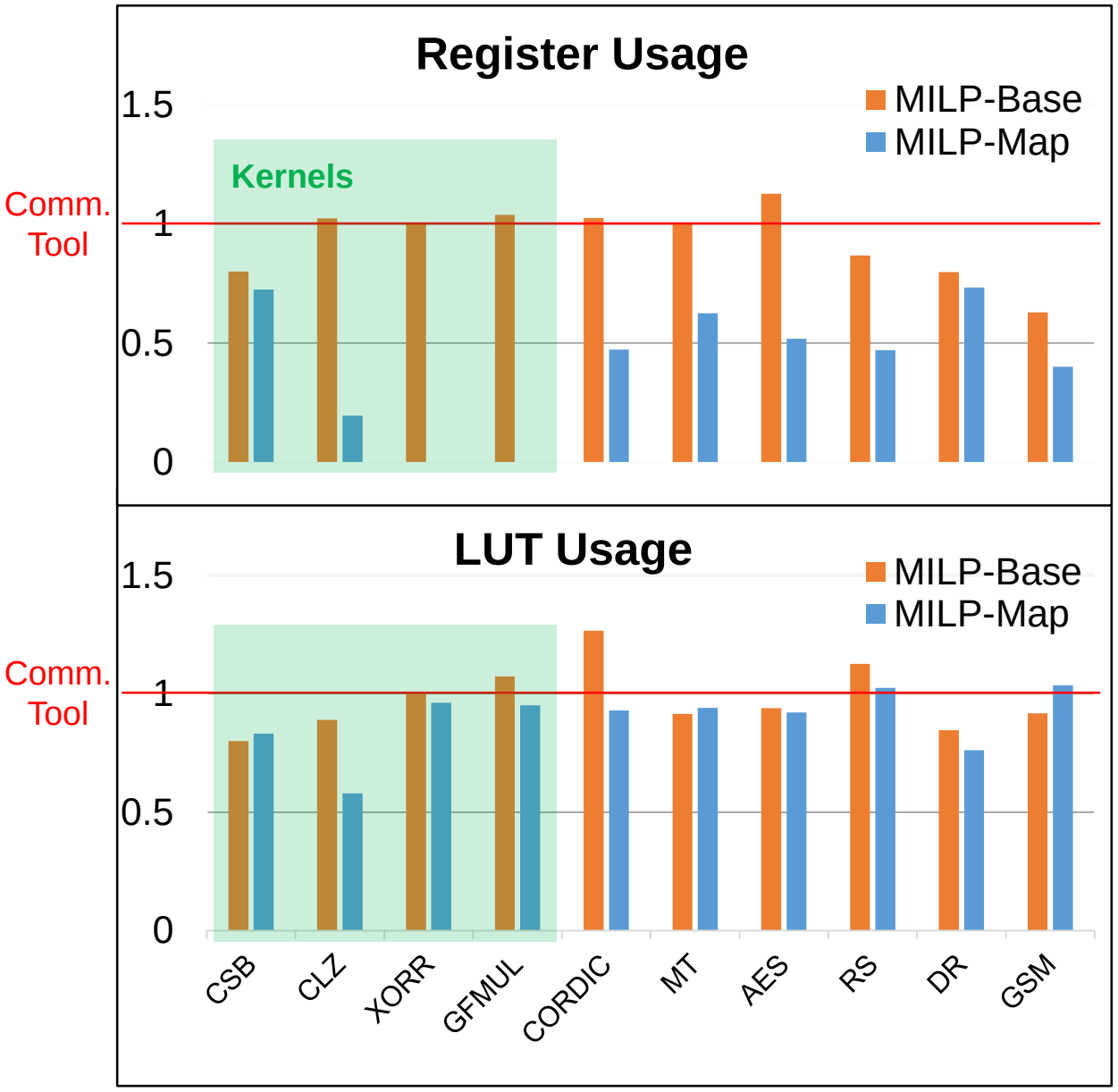


c depends on {**a**,**b**}
c[**i**] depends on
{**a**[**0**], **a**[**1**], **b**[**0**], **b**[**1**]}

Experiment Setup

- We compared our pipelining technique against that of a commercial HLS tool
- Our technique is implemented as an LLVM pass which modifies the IR to create a custom schedule
- Target device is Xilinx Virtex-7
- For fair comparison we implemented 2 versions of the MILP:
  - **MILP-Base** = does not consider mapping
  - **MILP-Map** = mapping-aware

Resource Usage Comparison



- Benchmarks:
  - 4 kernels
  - 6 applications using the kernels
- MILP-Base average:
  - 95% register usage
  - 98% LUT usage
- MILP-Map average:
  - 41% register usage
  - 89% LUT usage

MILP Runtime

Design	MILP Runtime (min)
CSB	5.3
CLZ	0.5
XORR	27.0
GFMUL	< 0.1
CORDIC	0.7
MT	**60.0**
AES	**60.0**
RS	< 0.1
DR	**60.0**
GSM	**60.0**

- MILP solver was capped at 60 minutes
- Commercial tool finishes in seconds for all designs
- Our work demonstrates a gap:
  - Significant area savings can be achieved by mapping-aware pipelining

Conclusions

- There is substantial room for improving FPGA-targeted HLS by leveraging cross-layer optimizations
- Our exact formulation of mapping-aware pipelining achieves significant resource reduction on logic-intensive designs

- **Future Work:**
  - Heuristic mapping-aware algorithm for improved runtime
  - Integration of other downstream optimizations (such as logic decomposition)

THANKS!

QUESTIONS?

Cut Enumeration

- Processes each node in topological order
- The cut-set ***CUTS*** of a node ***v*** with two inputs u_1 and u_2 can be computed from the cut-sets of u_1 and u_2:

$$CUTS(v) = \{\{v\} \cup \{C' = C_1 \cup C_2 \mid C_x \in CUTS(u_x), |C'| < K\}\}$$

v added for propagation

Union of each combination of the cuts of $\mathbf{u_1}$ and $\mathbf{u_2}$

C' must be K-feasible

- However, we cannot use cut enumeration during pipelining on a word-level graph