Range and Bitmask Analysis for Hardware Optimization in High-Level Synthesis

Jan 25, 2013
ASP-DAC

Marcel Gort
Jason Anderson

UNIVERSITY OF TORONTO

Motivation

- Software programs mostly use standard 32 and 64 bit datatypes to represent variables.
  - However, don't need 32 bits for a loop counter that only counts to 100!
  - Software is over-engineered, which is fine because processor datapaths are fixed-width.

LegUp

- LegUp is an open-source high level synthesis framework built within the llvm compiler framework.
  - C to Verilog (supports CHStone benchmarks).
  - Targets pure HW or processor/accelerator system.
  - Automated verification.
- Developed at the University of Toronto.
- Freely downloadable at legup.eecg.utoronto.ca

Motivation

- High-level-synthesis (HLS) generates hardware from software program.
- Unlike with software, efficiency of that hardware is dependent on bit-level representation of variables.
- Need bitwidth analysis in HLS to generate minimum bit-level representation for each variable.

This work

- Created a new bitmask analysis approach and combined it with existing variable range analysis techniques.
- Built bitwidth analysis pass into LegUp HLS.

Bitwidth Minimization

- Minimize variable bitwidths by propagating constants through the program instructions.
- Variables represented in one of two ways:

Bitwidth Minimization

- Minimize variable bitwidths by propagating constants through the program instructions.
- Variables represented in one of two ways:
    1) As a min/max value – e.g. -2 -> 2

Bitwidth Minimization

- Minimize variable bitwidths by propagating constants through the program instructions.
- Variables represented in one of two ways:
    1) As a min/max value – e.g. -2 -> 2
    2) As a bitmask of known values (0 or 1), unknowns (?), or sign-extended bits (S).
       e.g. “S?10”

Bitwidth Minimization

- Minimize variable bitwidths by propagating constants through the program instructions.
- Variables represented in one of two ways:
  1) As a min/max value – e.g. -2 -> 2
  2) As a bitmask of known values (0 or 1), unknowns (?), or sign-extended bits (S).
     e.g. "S?10"

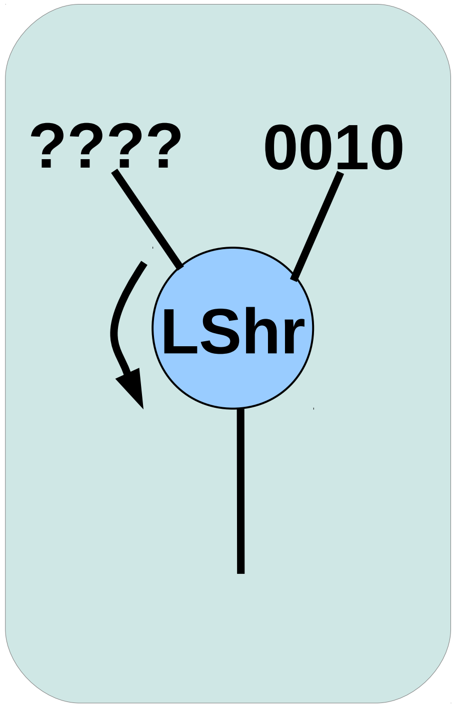
$\uparrow$

Focus of our work

Bitwidth Minimization

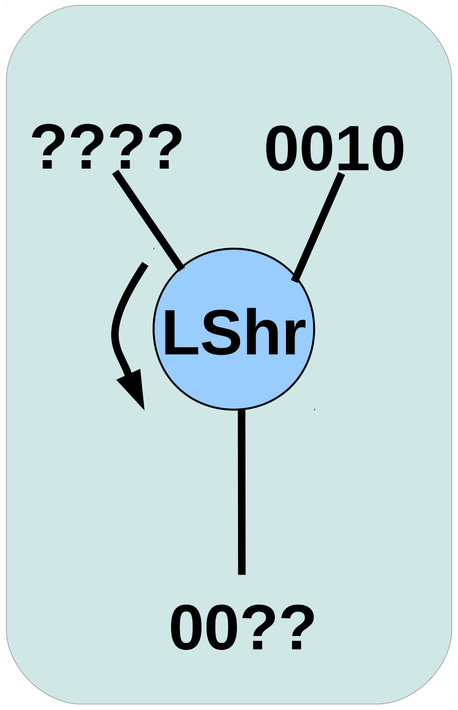
- Software program represented as a control dataflow graph (CDFG) of llvm operators.
- Traverse CDFG in forward and backward directions, propagating bitwidths through operators.
- For each llvm operator, we created forward and backward transit functions.
  - e.g Xor, Shl, Ashr, Mul, Div, etc.

Examples



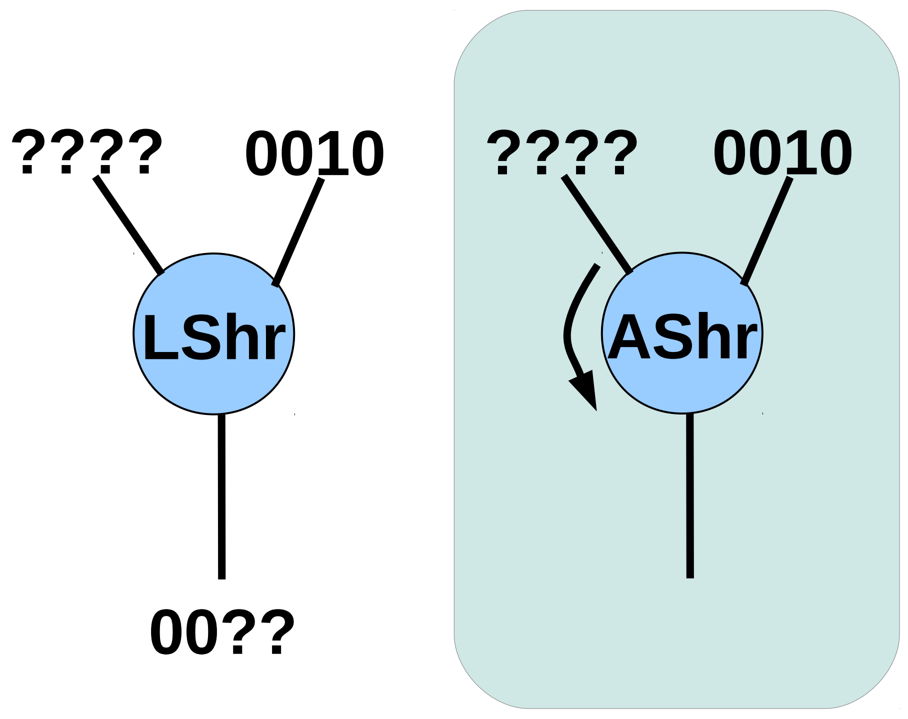
Forward

Examples



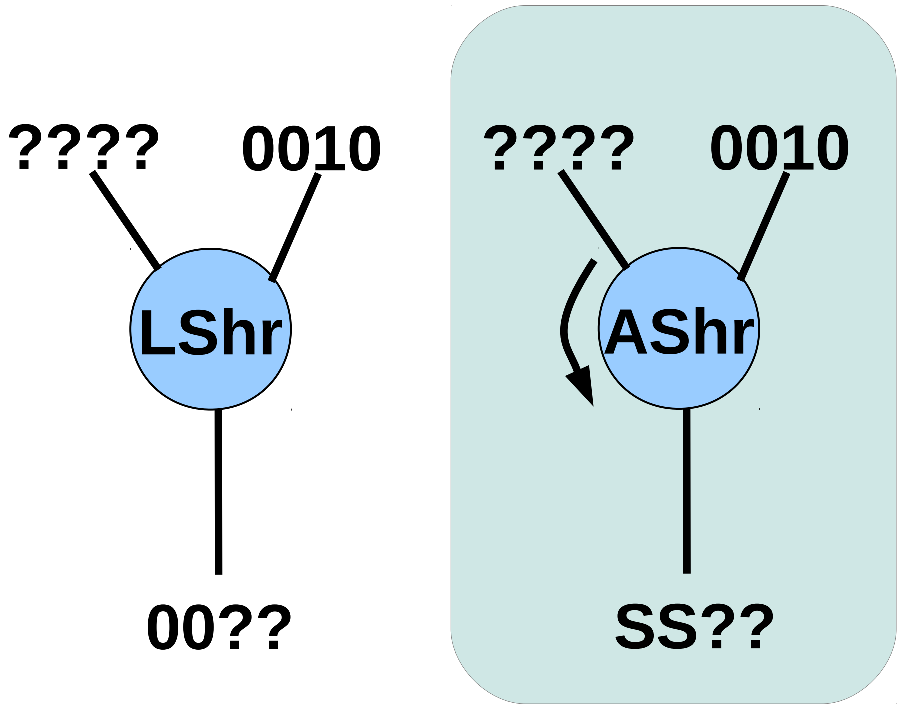
Forward

Examples



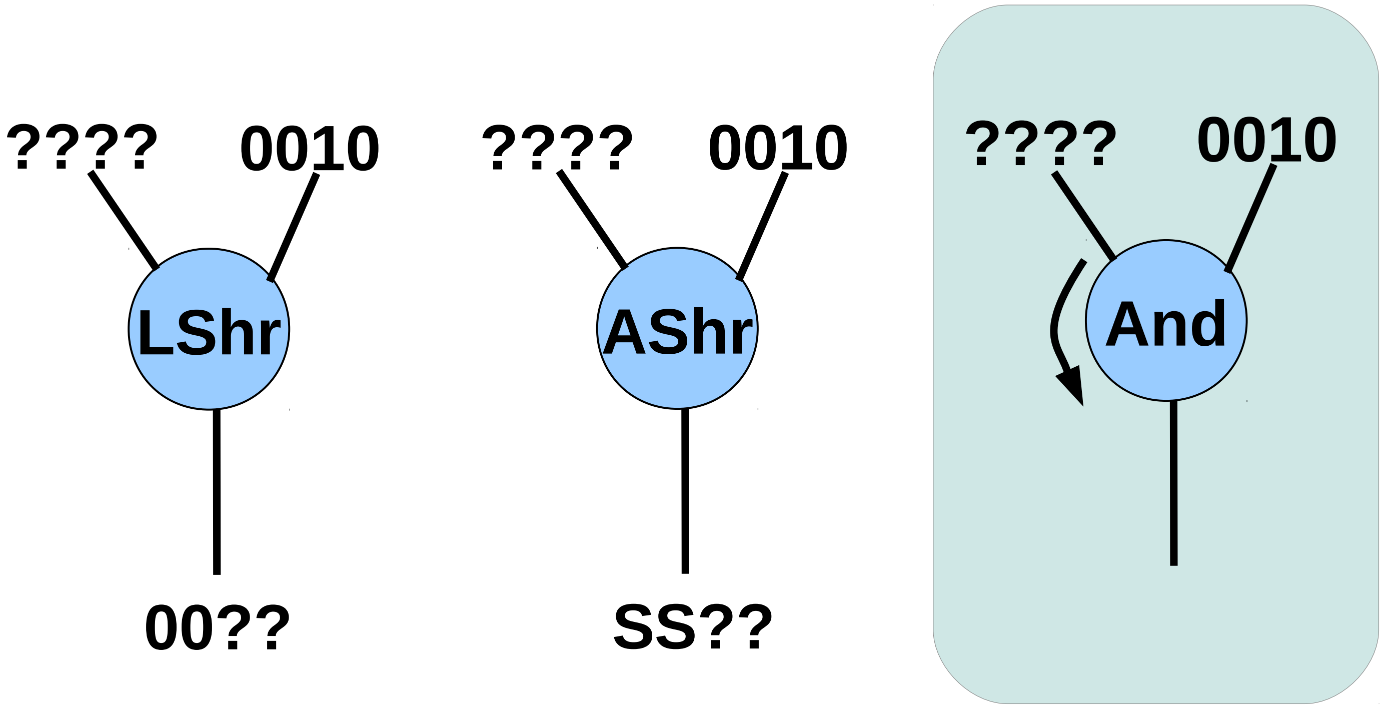
Forward

Examples



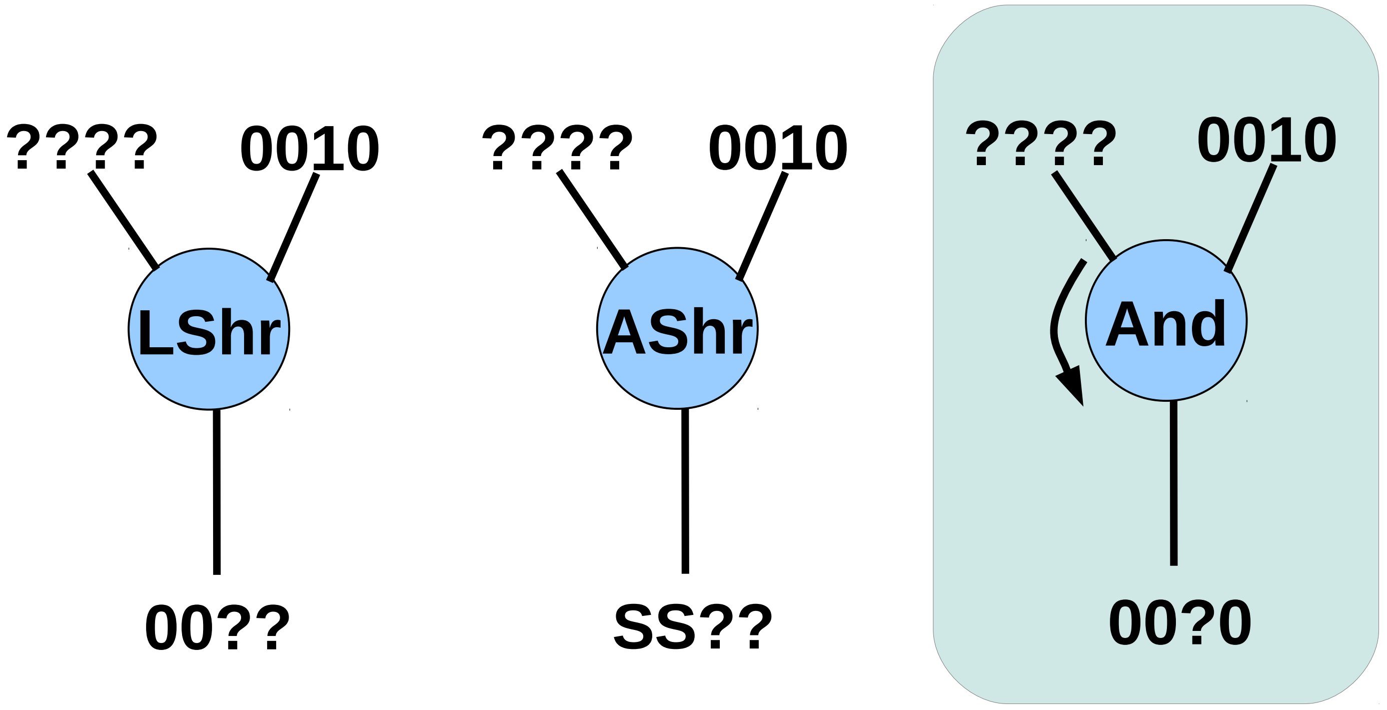
Forward

Examples



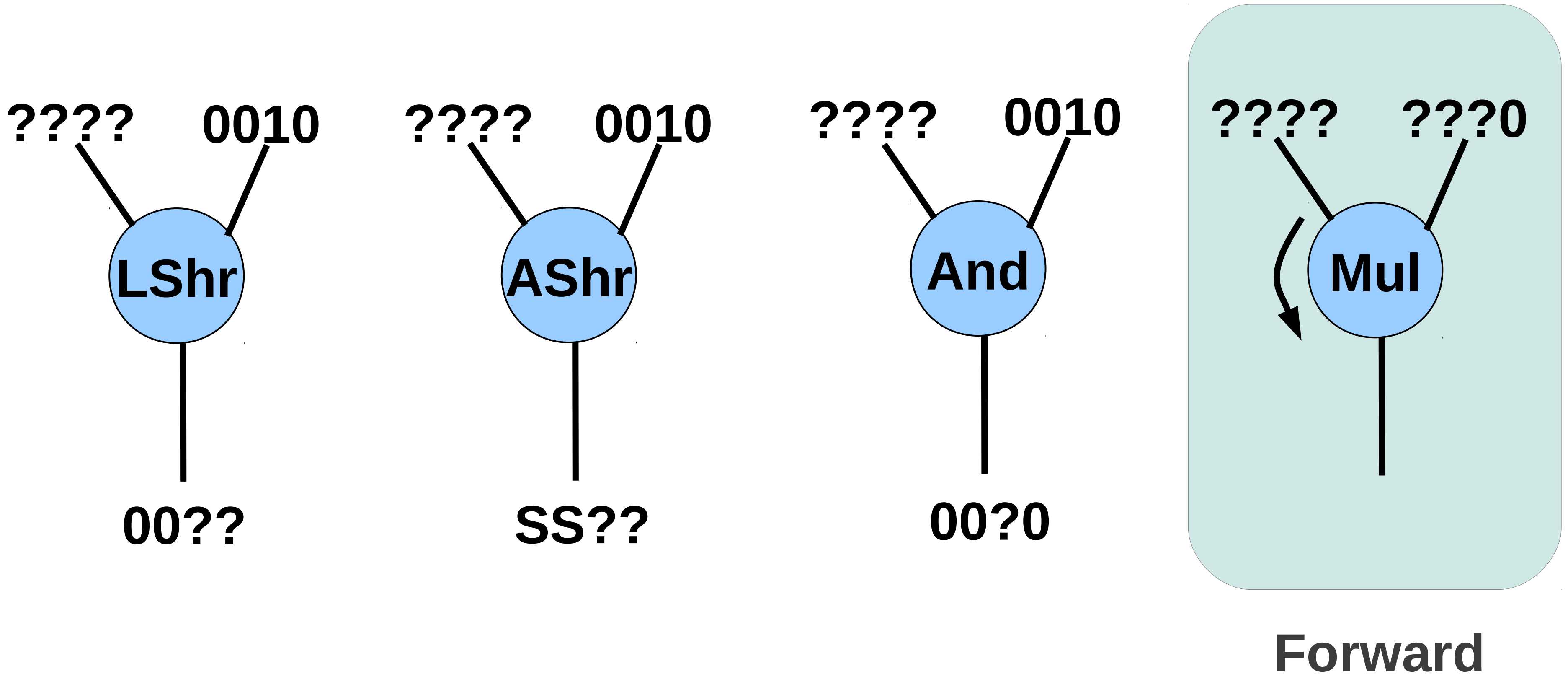
Forward

Examples

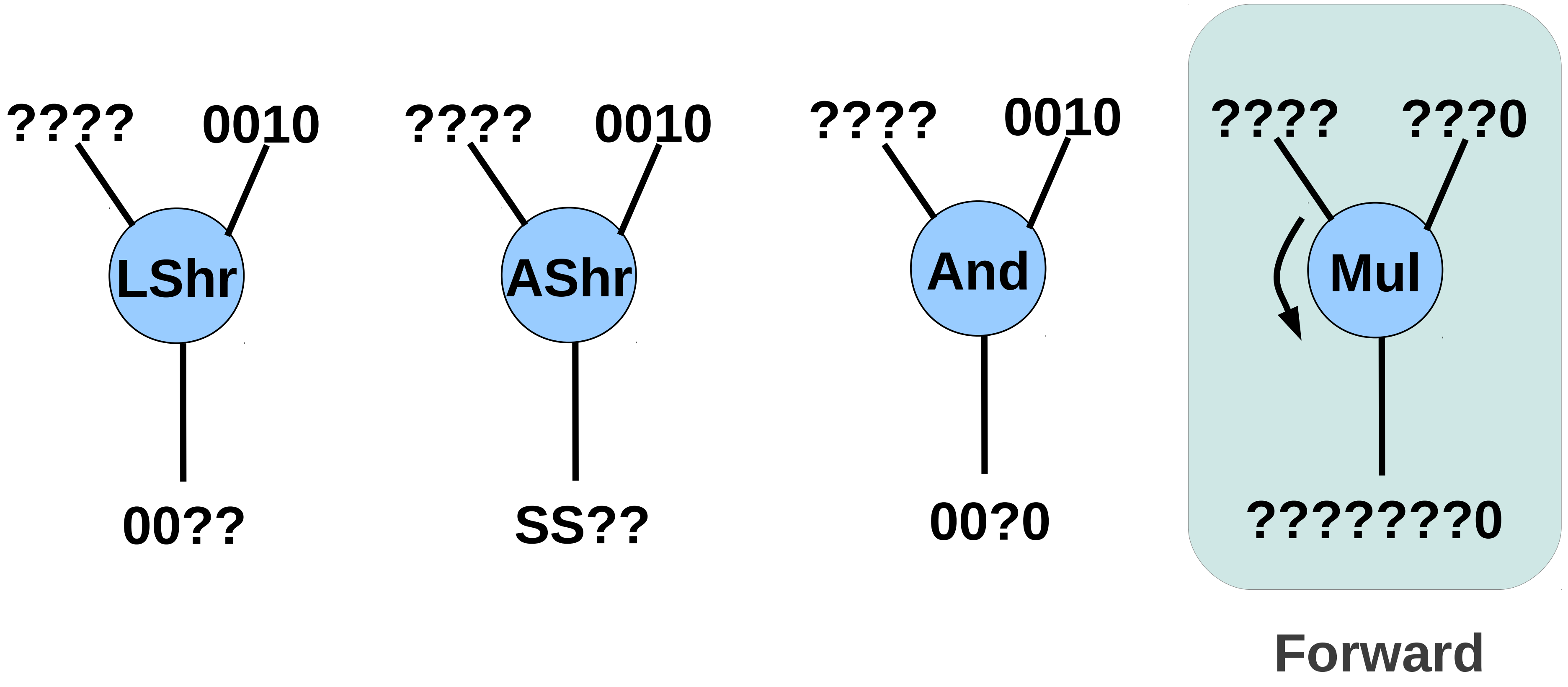


Forward

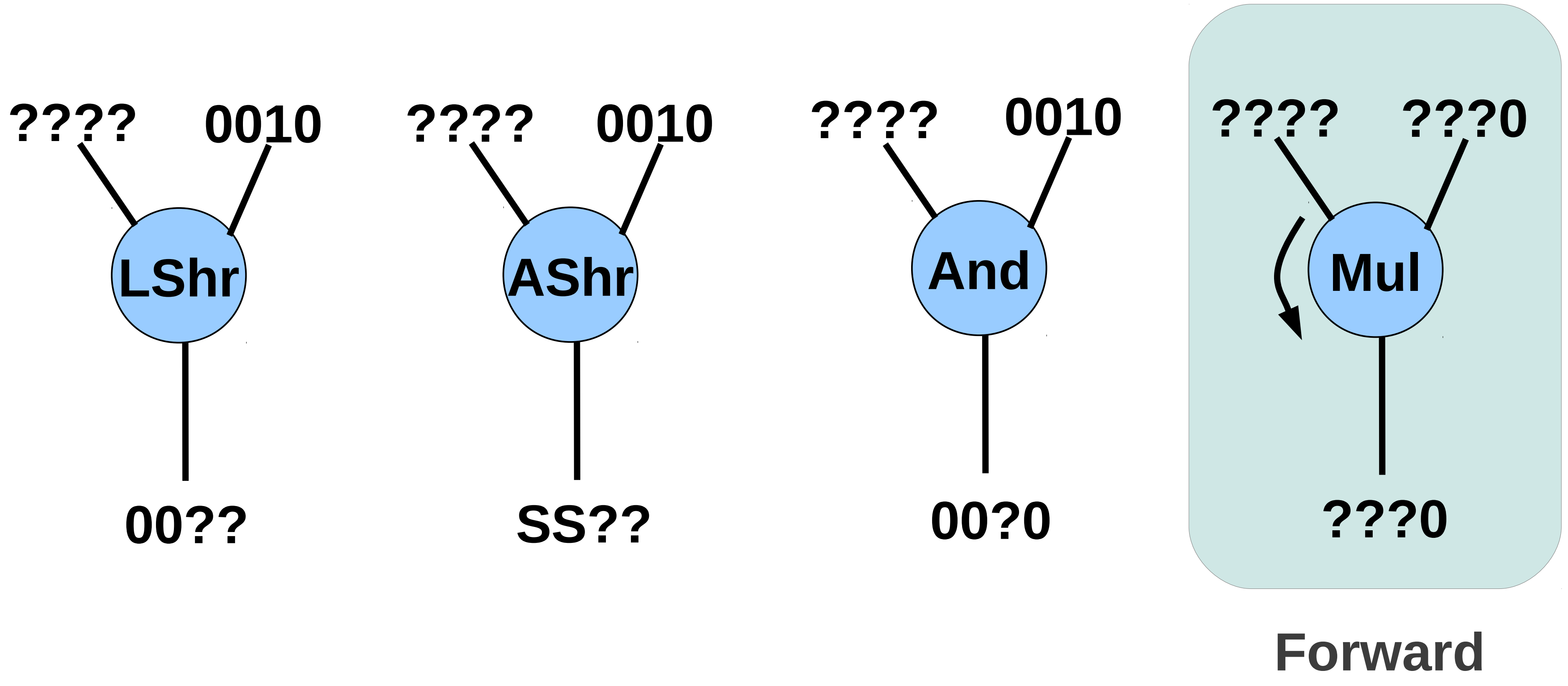
Examples



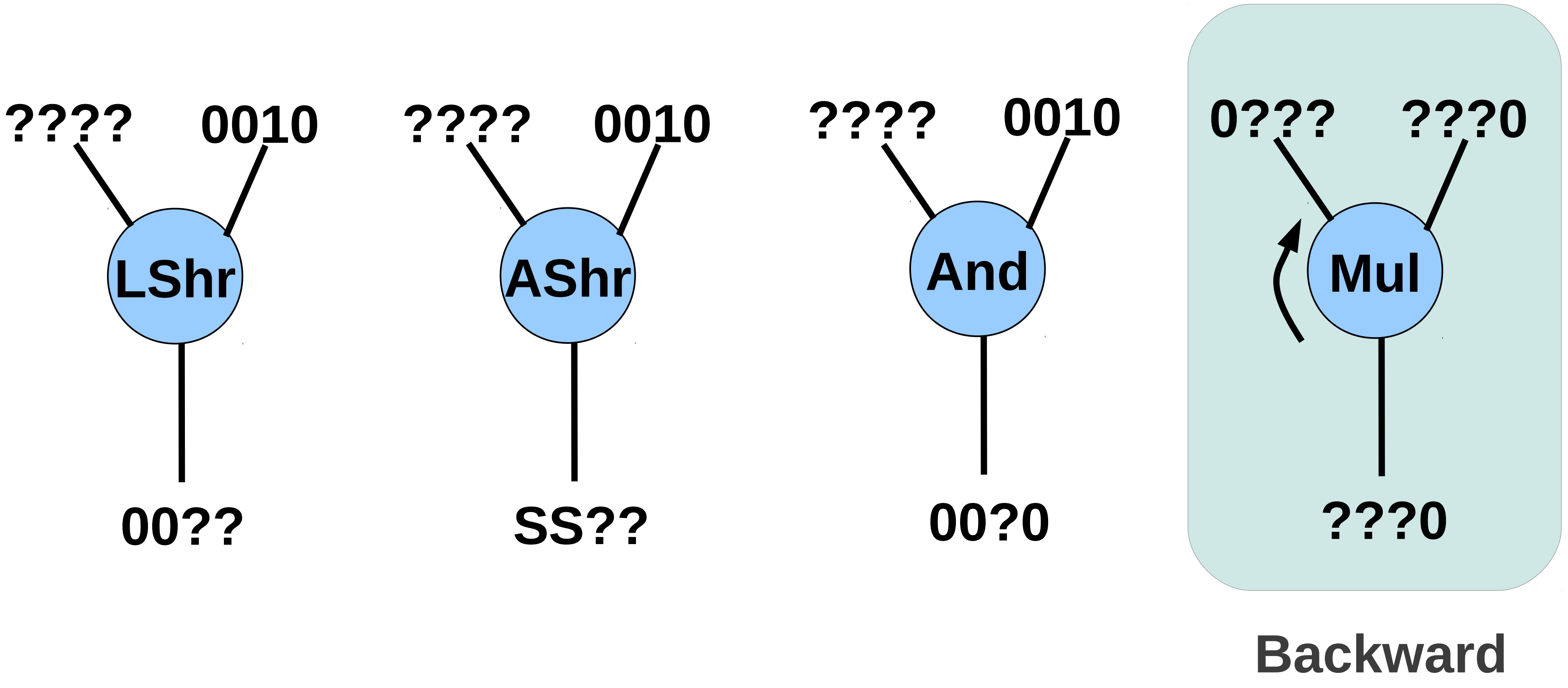
Examples



Examples

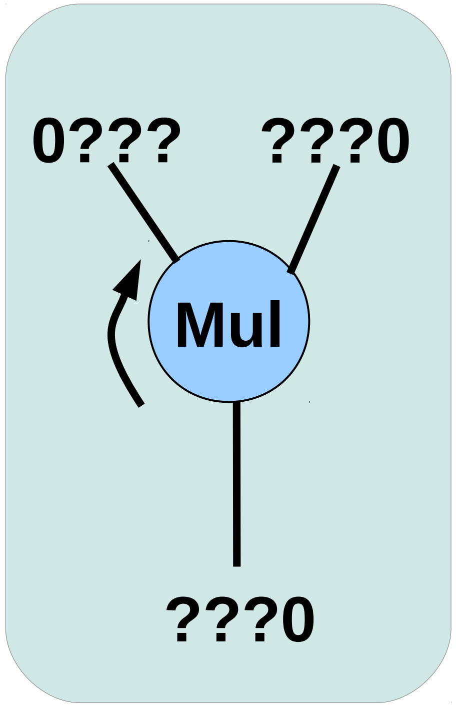


Examples



Examples

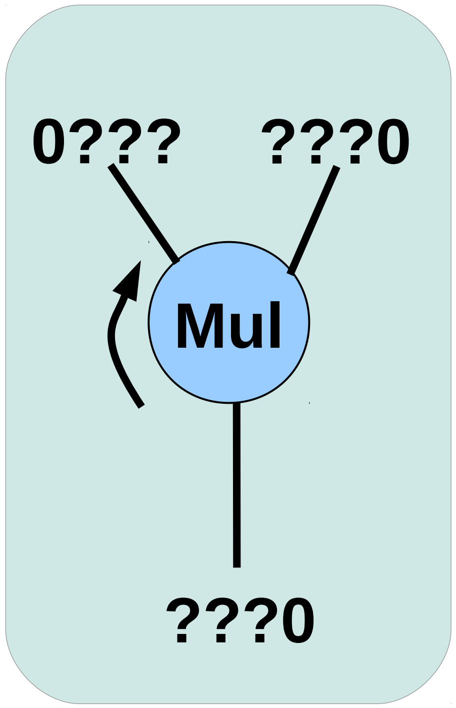
```
   ???0
x  ????
   ????
   ???0
```



Backward

Examples

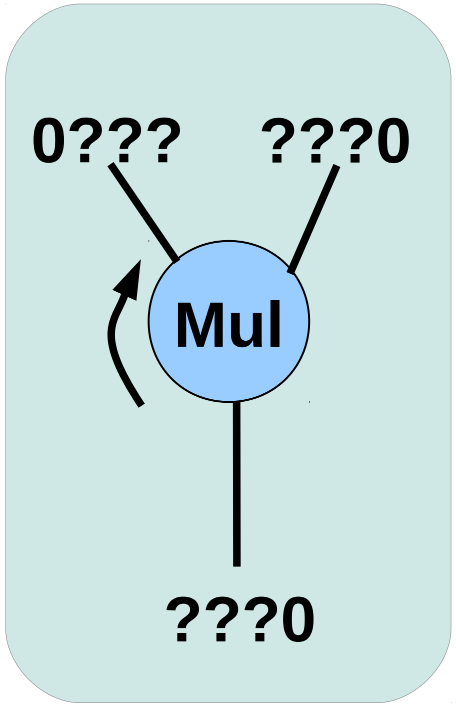
```
   ???0
x  ????
-------
   ???0
  ???00
```



Backward

Examples

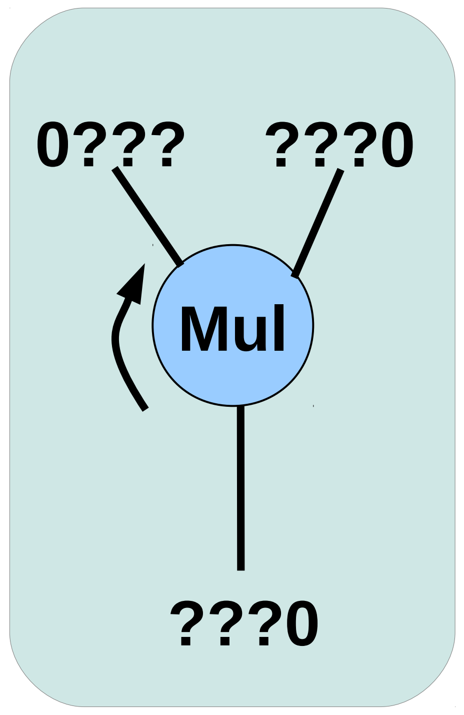
```
    ???0
x  ????
   ____
    ???0
   ??00
  ???000
```



Backward

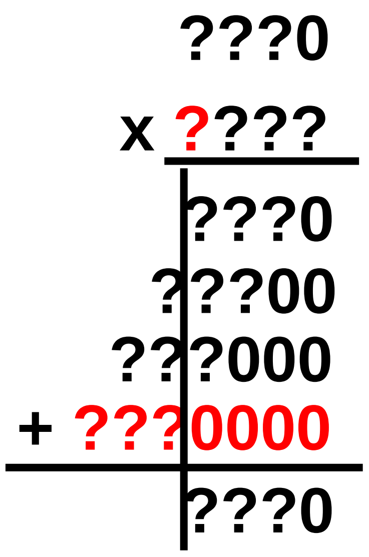
Examples

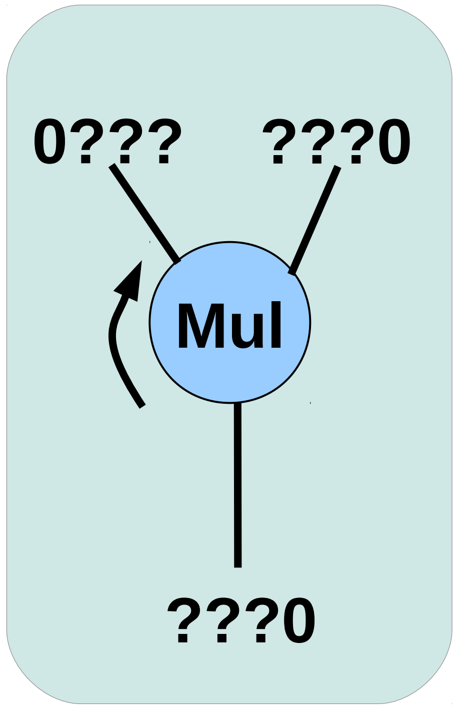
```
     ???0
 x  ????
 --------
     ???0
    ??00
   ??000
+ ???0000
```



Backward

Examples

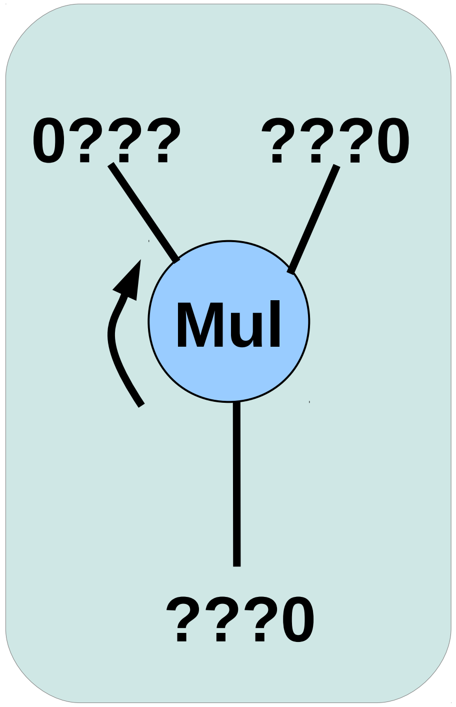




Backward

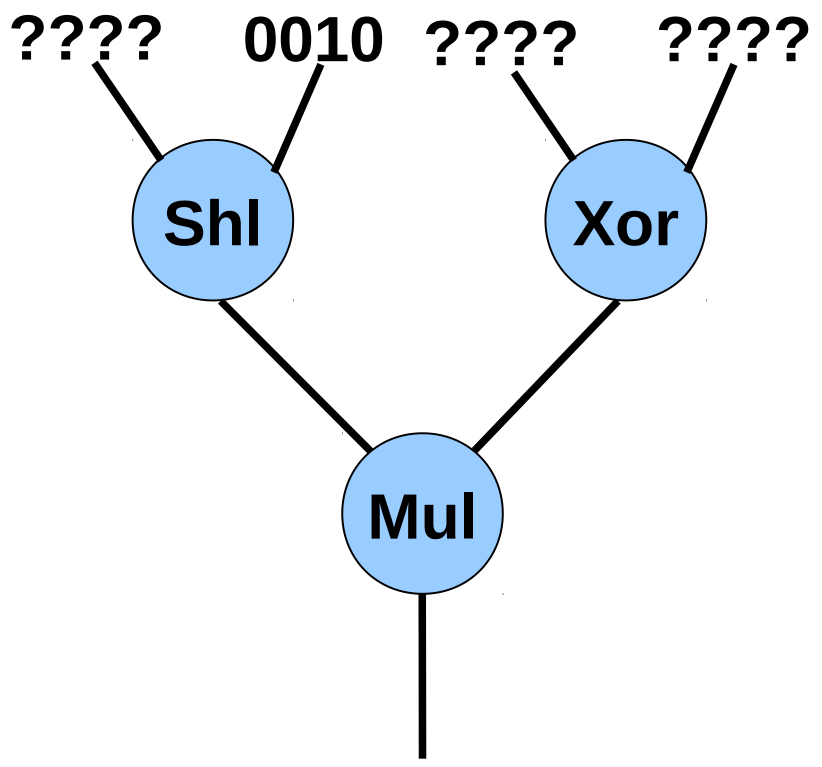
Examples

```
       ????0
     x 0????
       ????0
      ???00
     ???000
  + 0000000
       ????0
```

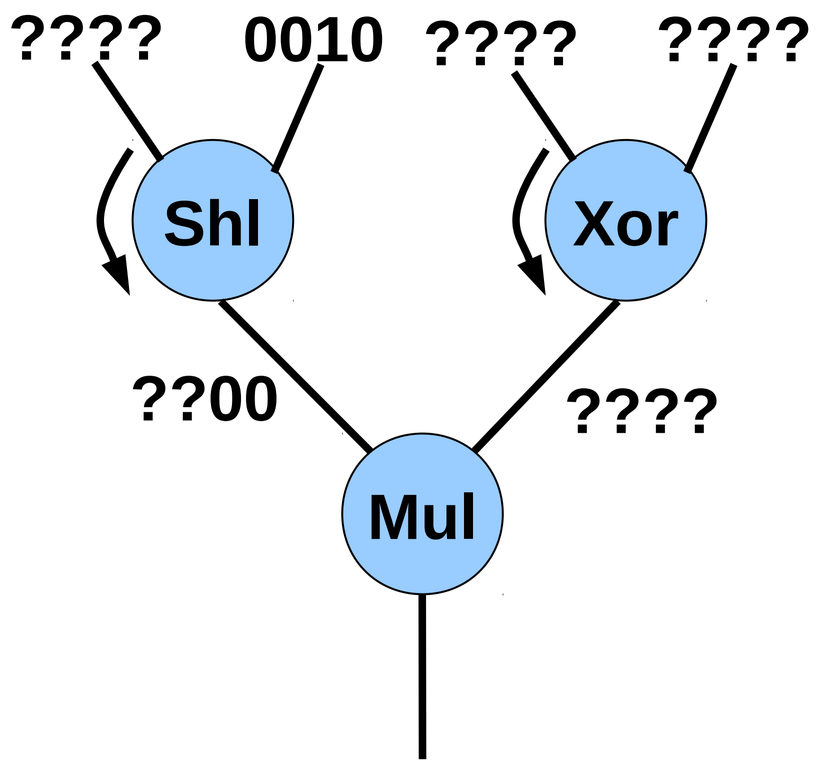


Backward

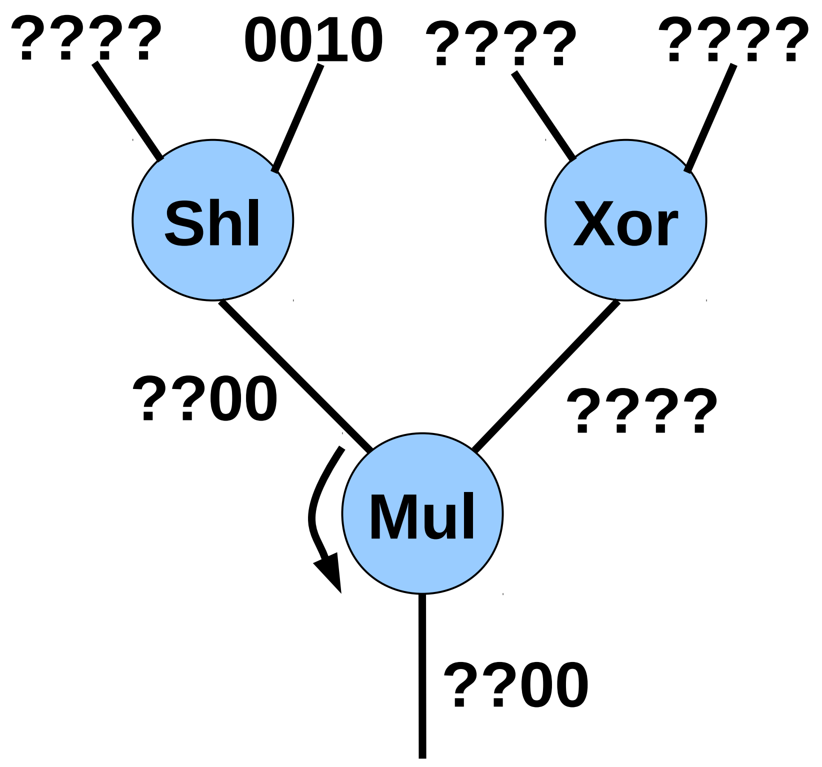
Examples



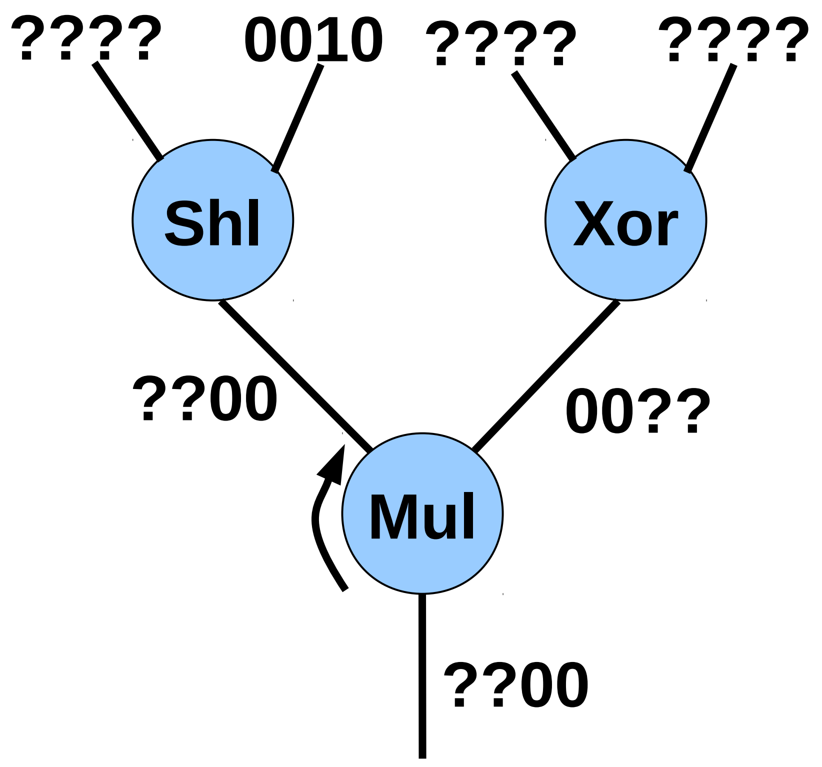
Examples



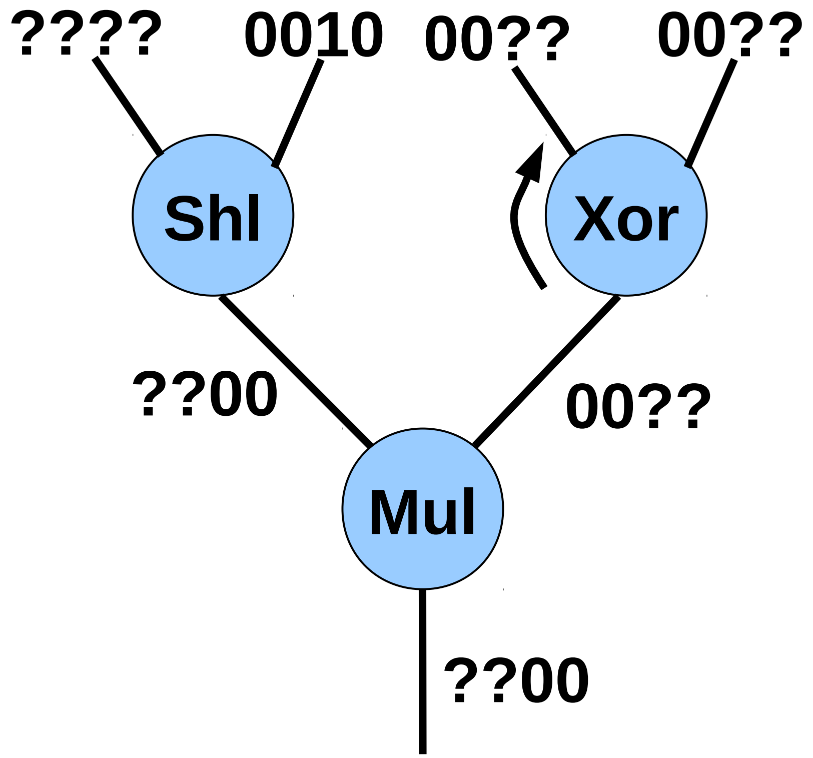
Examples



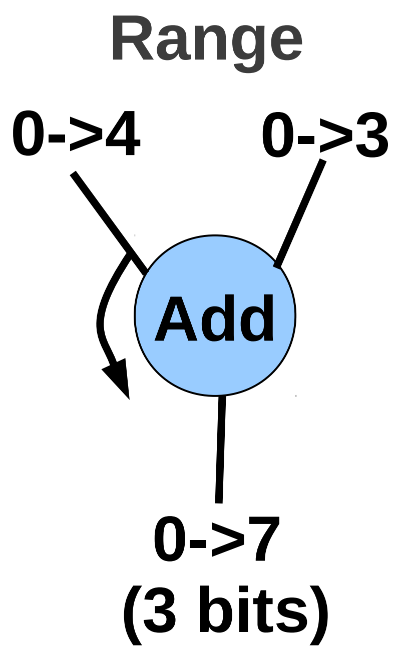
Examples



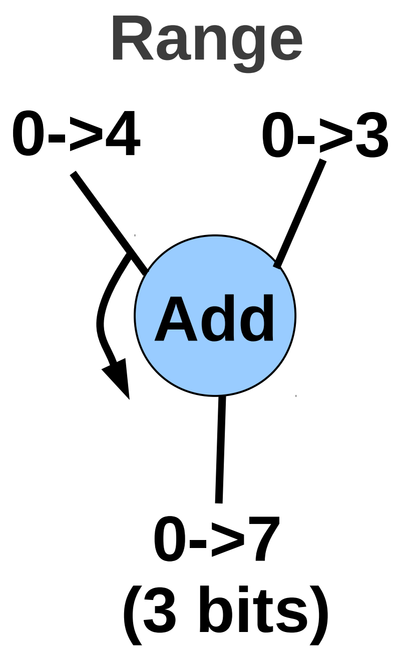
Examples

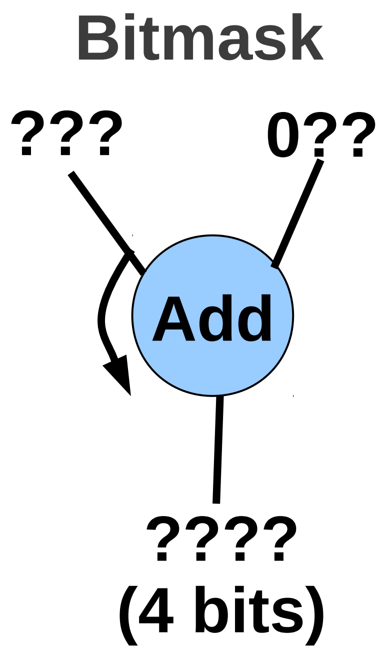


Range vs. Bitmask analysis

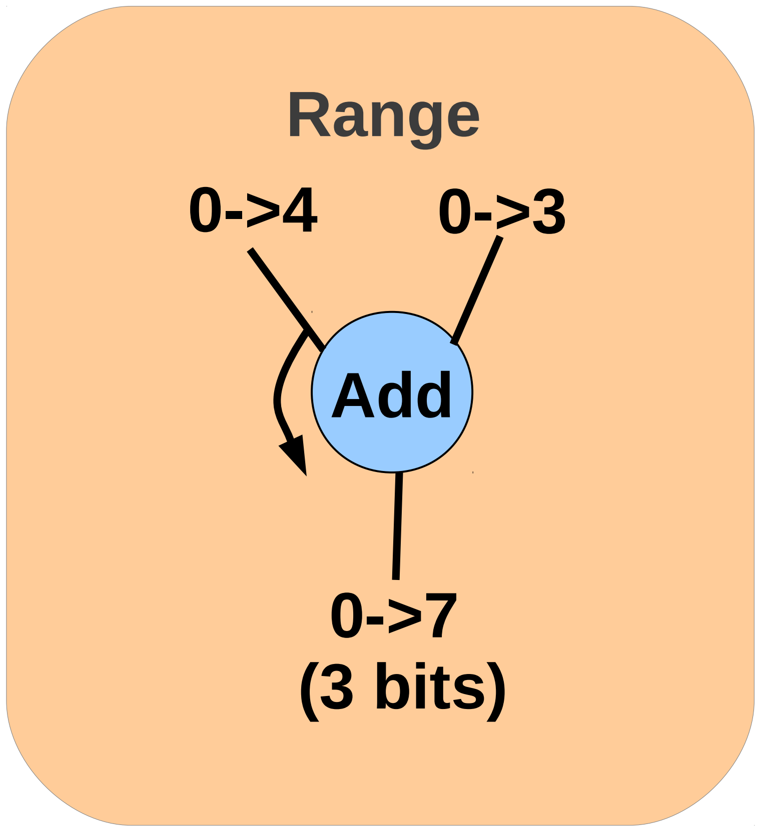


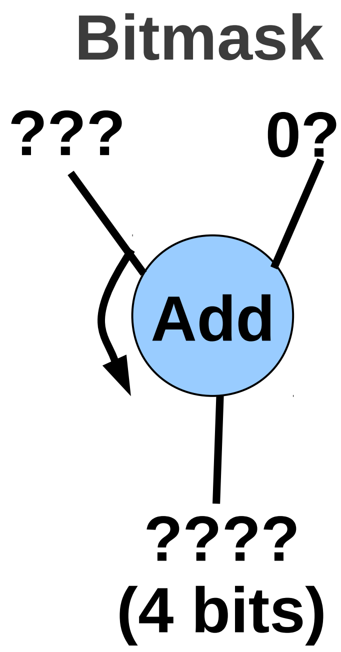
Range vs. Bitmask analysis





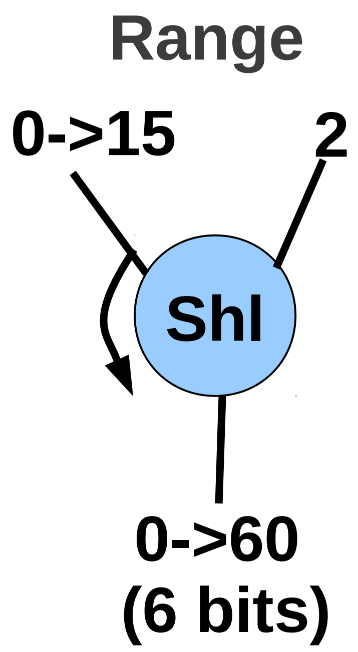
Range vs. Bitmask analysis



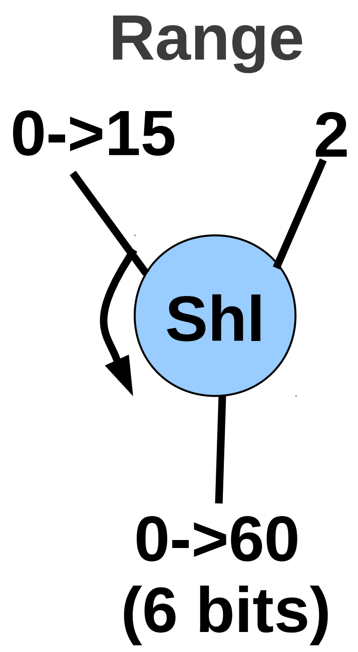


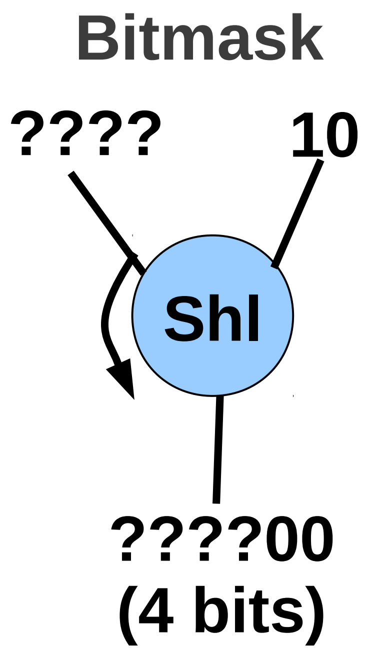
WINNER!

Range vs. Bitmask analysis

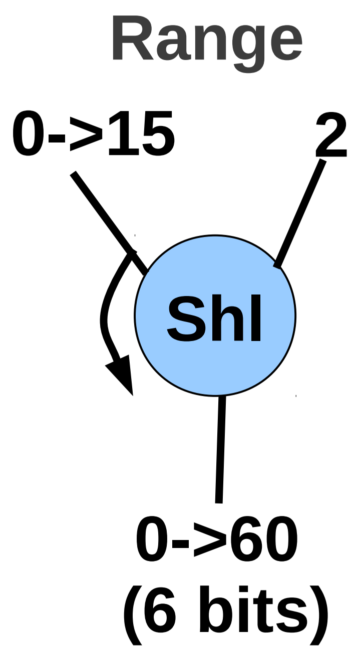


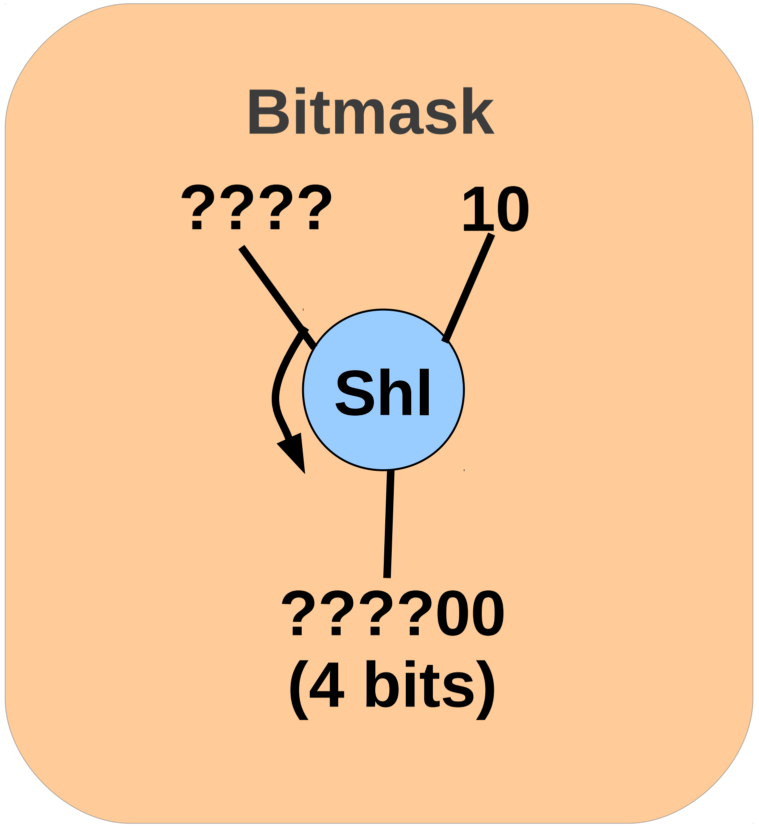
Range vs. Bitmask analysis





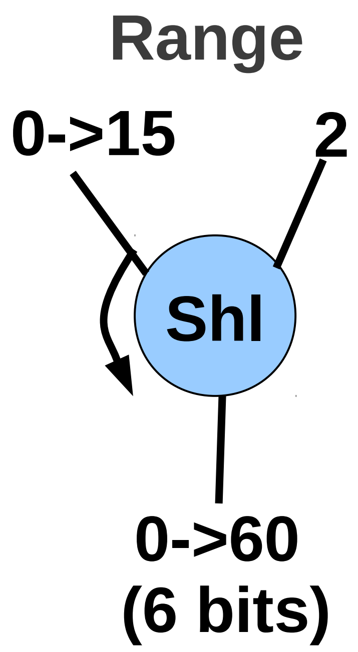
Range vs. Bitmask analysis

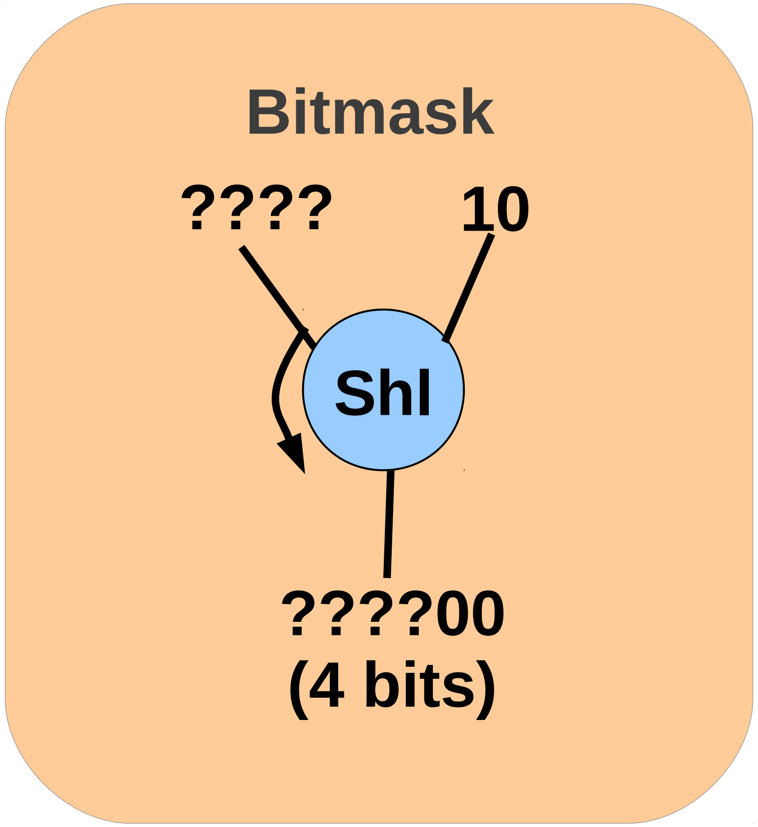




WINNER!

Range vs. Bitmask analysis





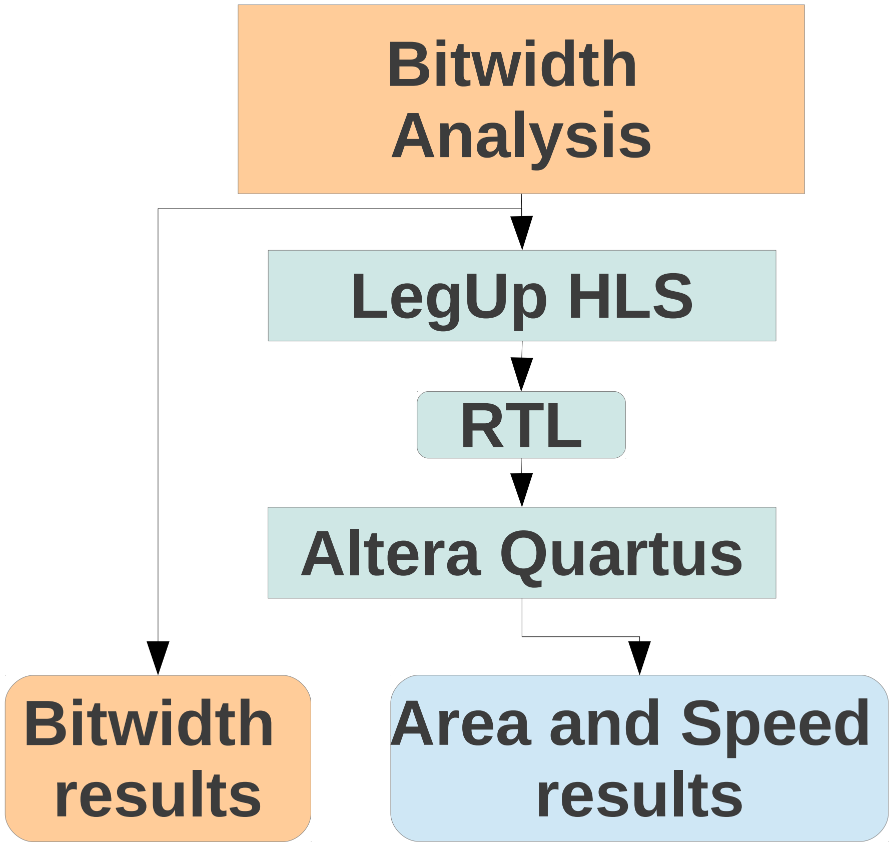
Range and bitmask analyses are complementary

Experimental Methodology

- Target Altera Cyclone II FPGAs.
- Used 10 CHStone benchmarks
  - All circuits were simulated after bitwidth reduction using ModelSim and golden inputs provided with CHStone to verify correct functionality.

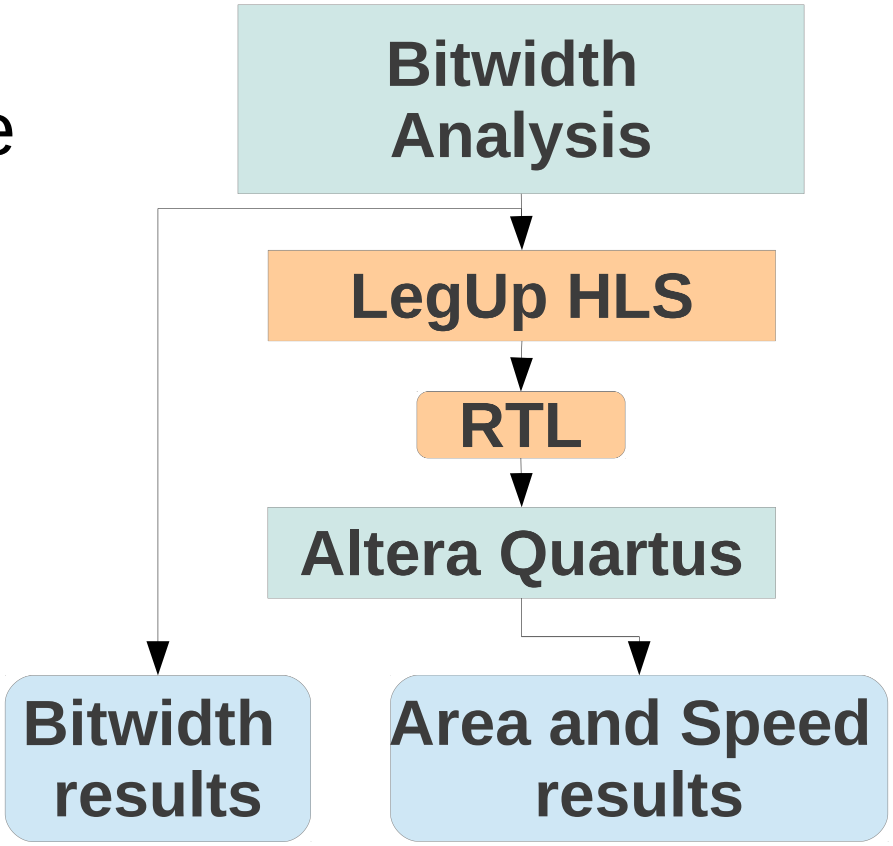
Experimental Methodology

- Bitwidth analysis llvm pass.
  - Result: Sum of instruction widths.



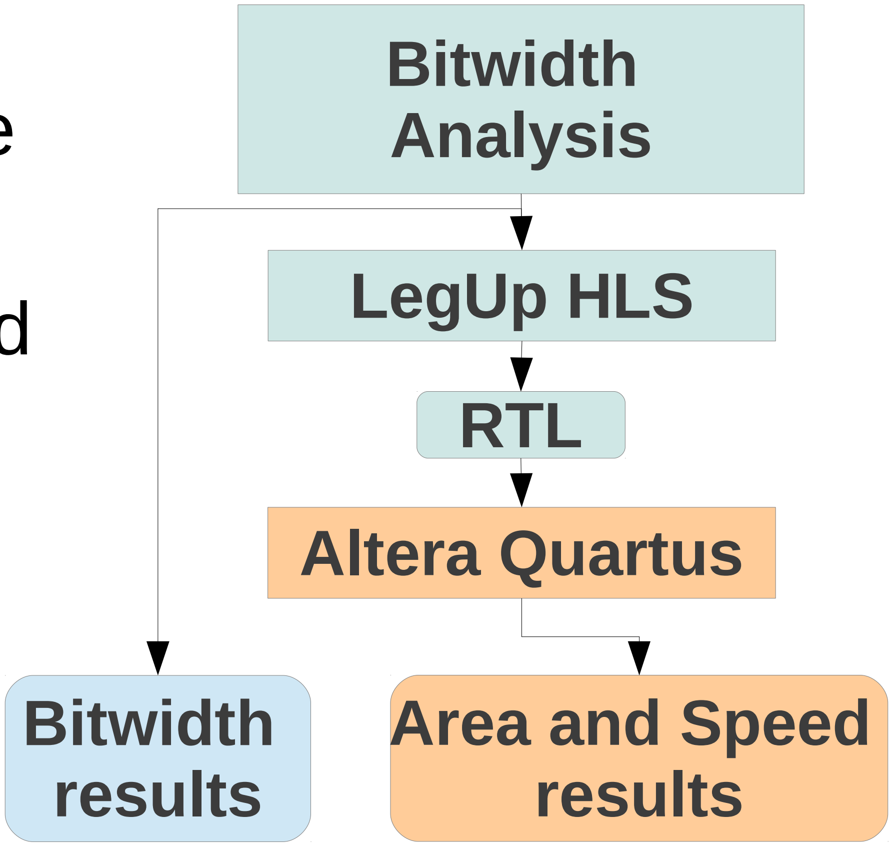
Experimental Methodology

- Bitwidth analysis llvm pass.
  - Result: Sum of instruction widths.
- LegUp HLS llvm pass uses bitwidth analysis to generate minimized RTL.



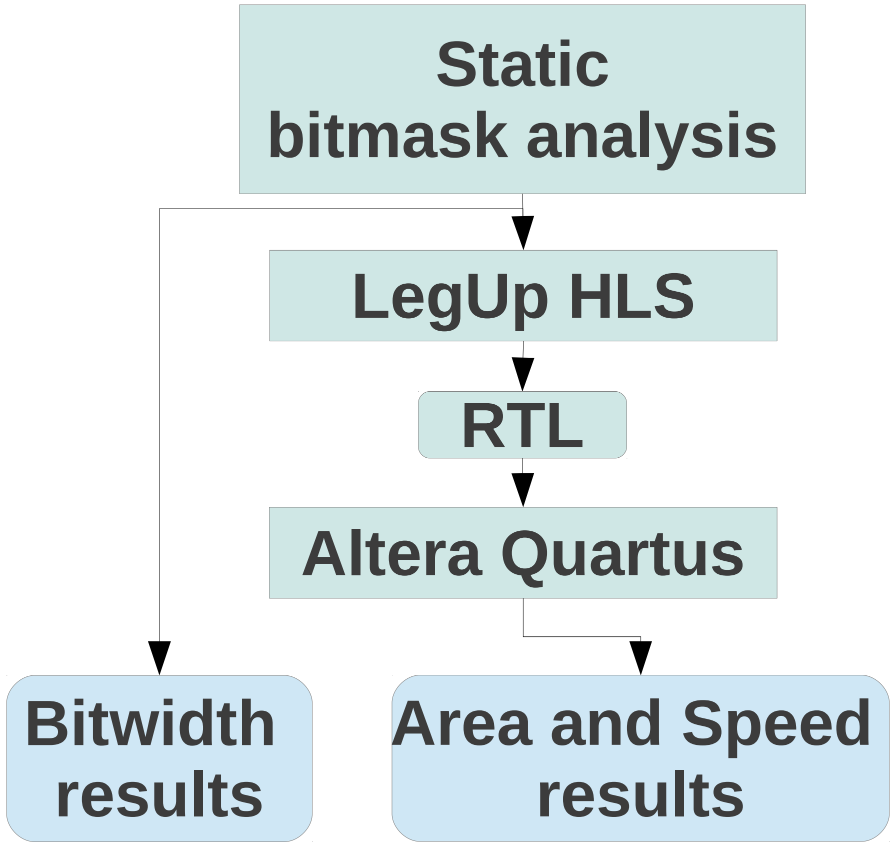
Experimental Methodology

- Bitwidth analysis llvm pass.
  - Result: Sum of instruction widths.
- LegUp HLS llvm pass uses bitwidth analysis to generate minimized RTL.
- Quartus generates optimized FPGA implementation.
  - It also minimizes bitwidth!
  - Results: Area in LUTs and registers, speed in Fmax.



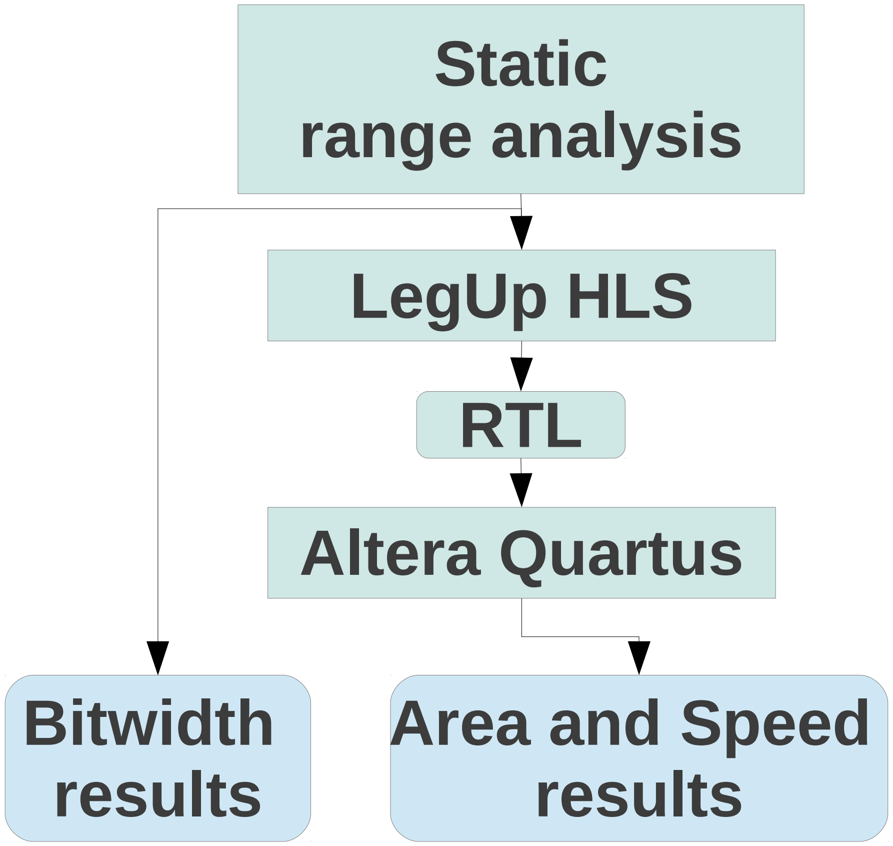
Experimental Methodology

- 5 flows
  - **Bitmask analysis by itself**



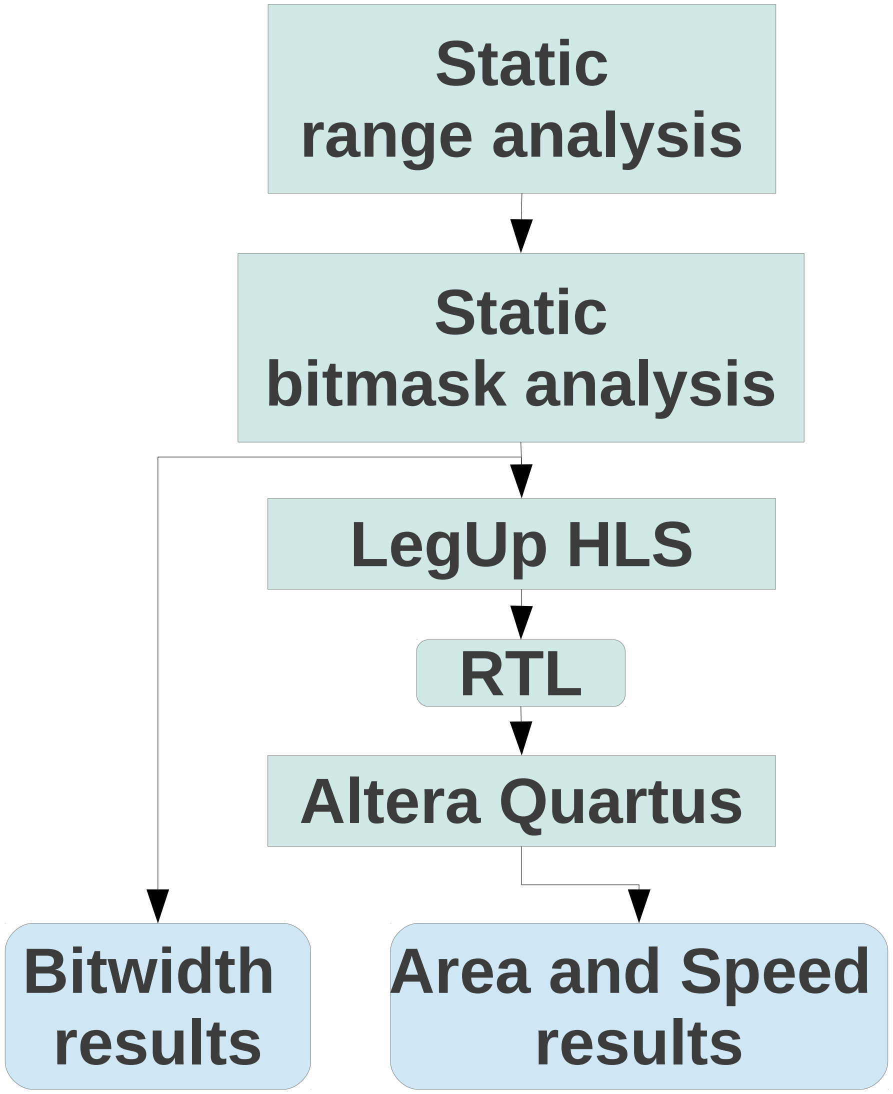
Experimental Methodology

- 5 flows
  - Bitmask analysis by itself
  - **Range analysis by itself (Campos et. al 2012)**



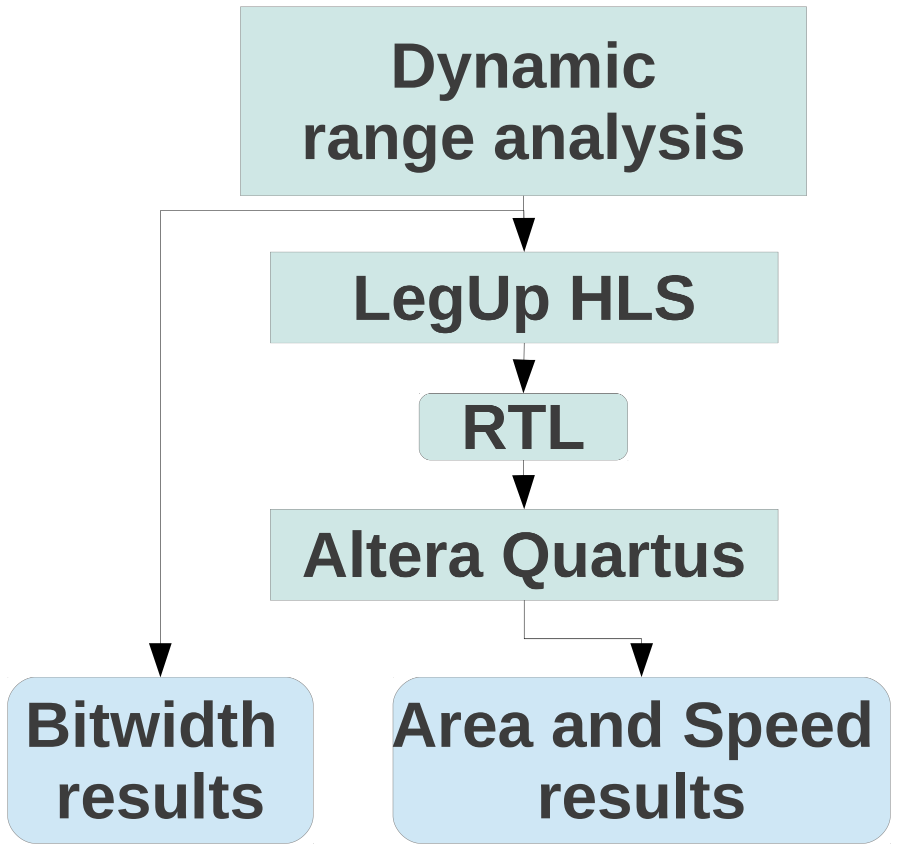
Experimental Methodology

- 5 flows
  - Bitmask analysis by itself
  - Range analysis by itself (Campos et. al 2012)
  - **Range & bitmask analysis**



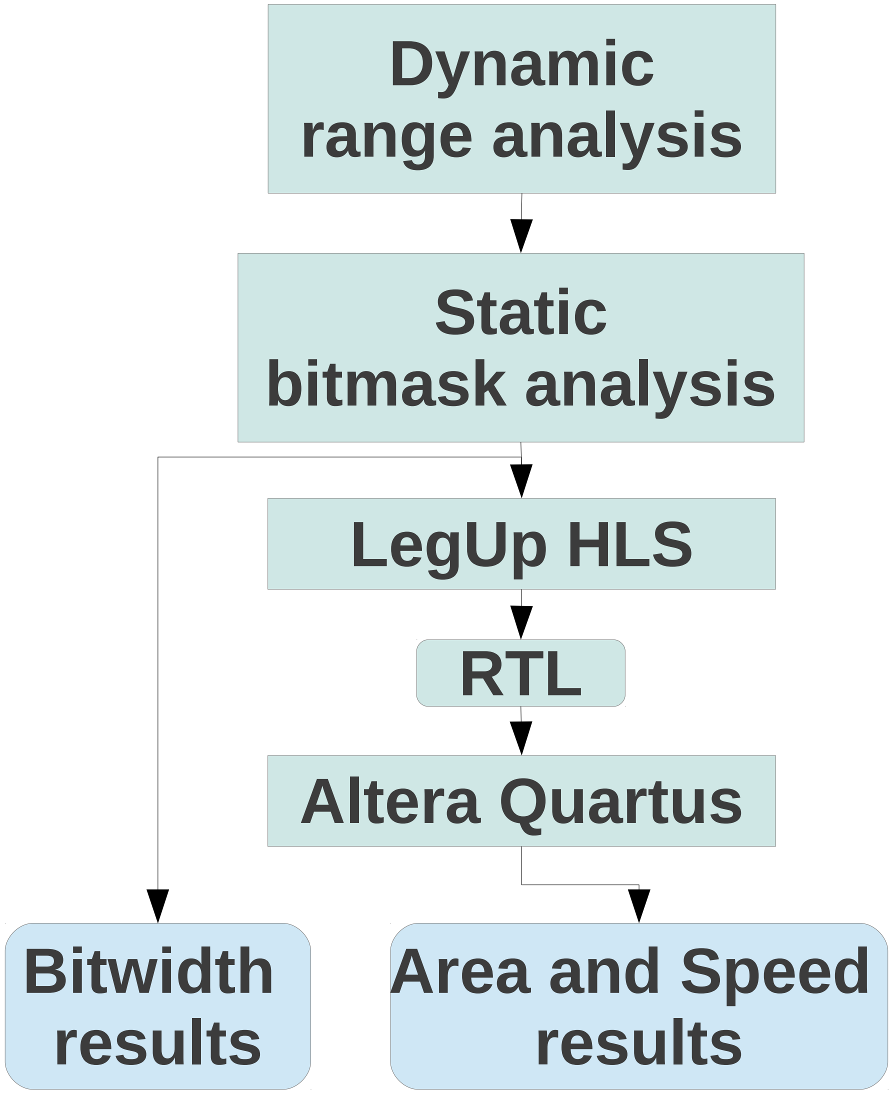
Experimental Methodology

- 5 flows
  - Bitmask analysis by itself
  - Range analysis by itself (Campos et. al 2012)
  - Range & Bitmask analysis
  - **Profiling-based dynamic range analysis**

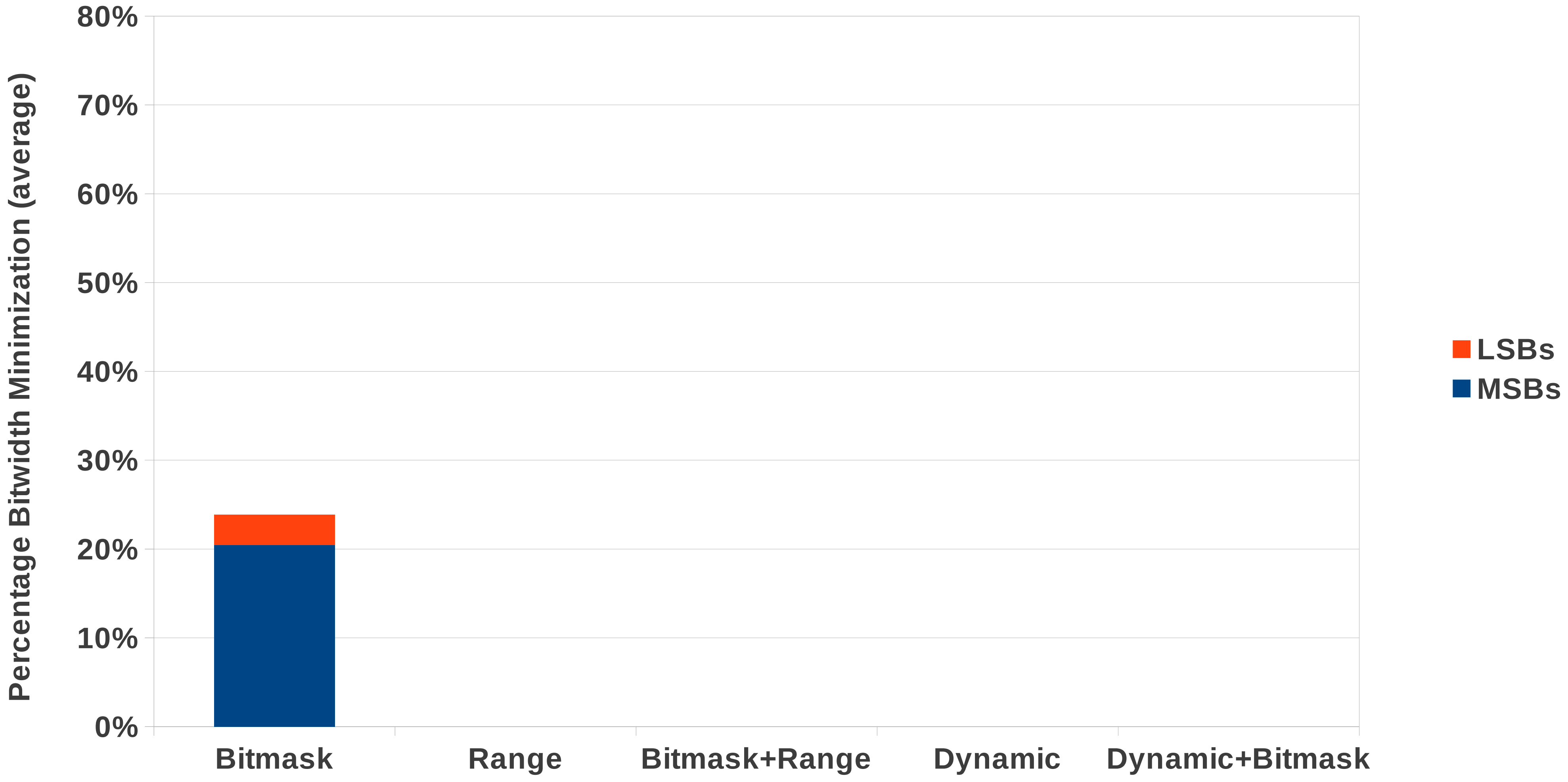


Experimental Methodology

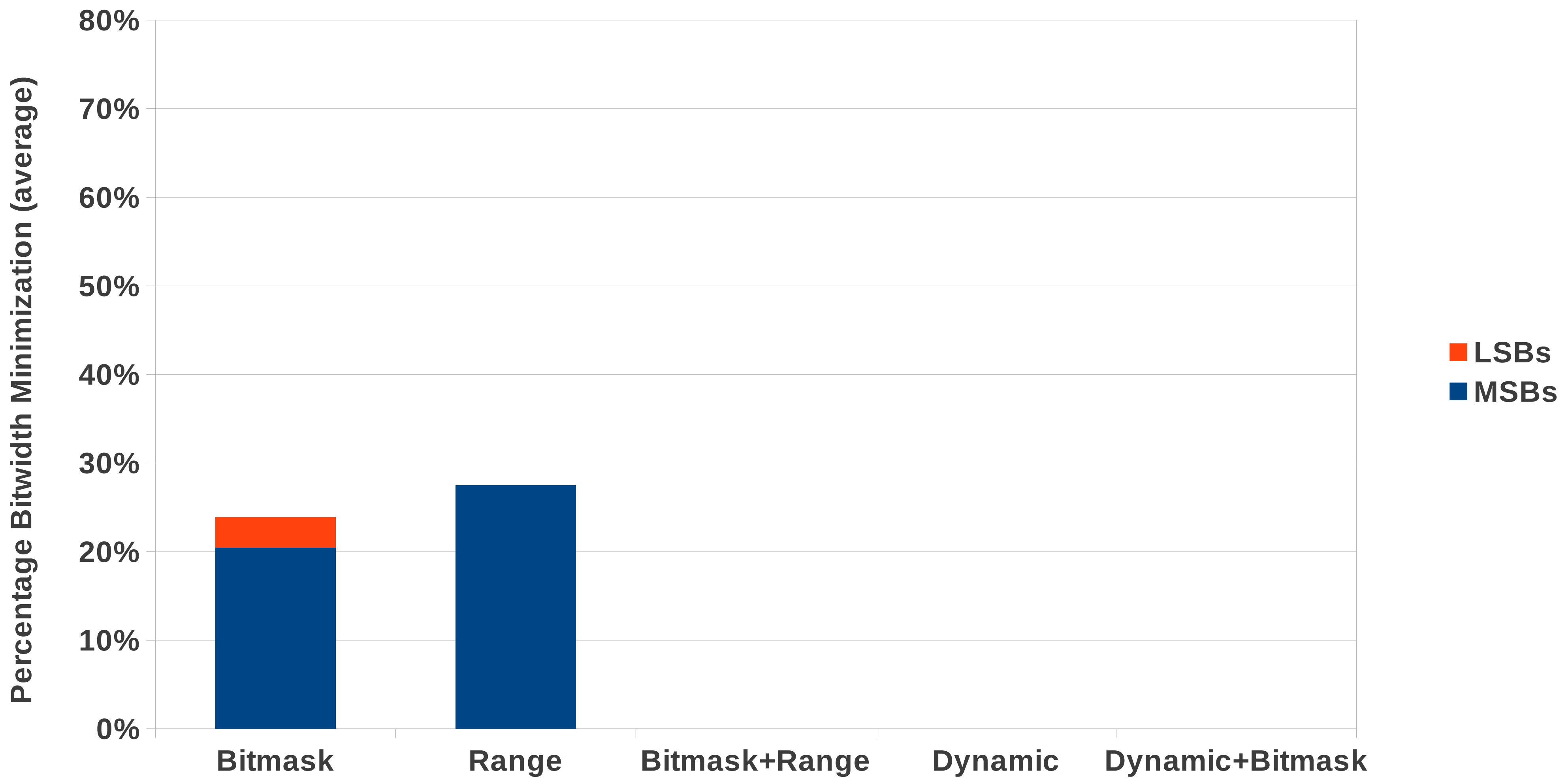
- 5 flows
  - Bitmask analysis by itself
  - Range analysis by itself (Campos et. al 2012)
  - Range & Bitmask analysis
  - Profiling-based dynamic range analysis
  - **Profiling-based dynamic range analysis & bitmask analysis**



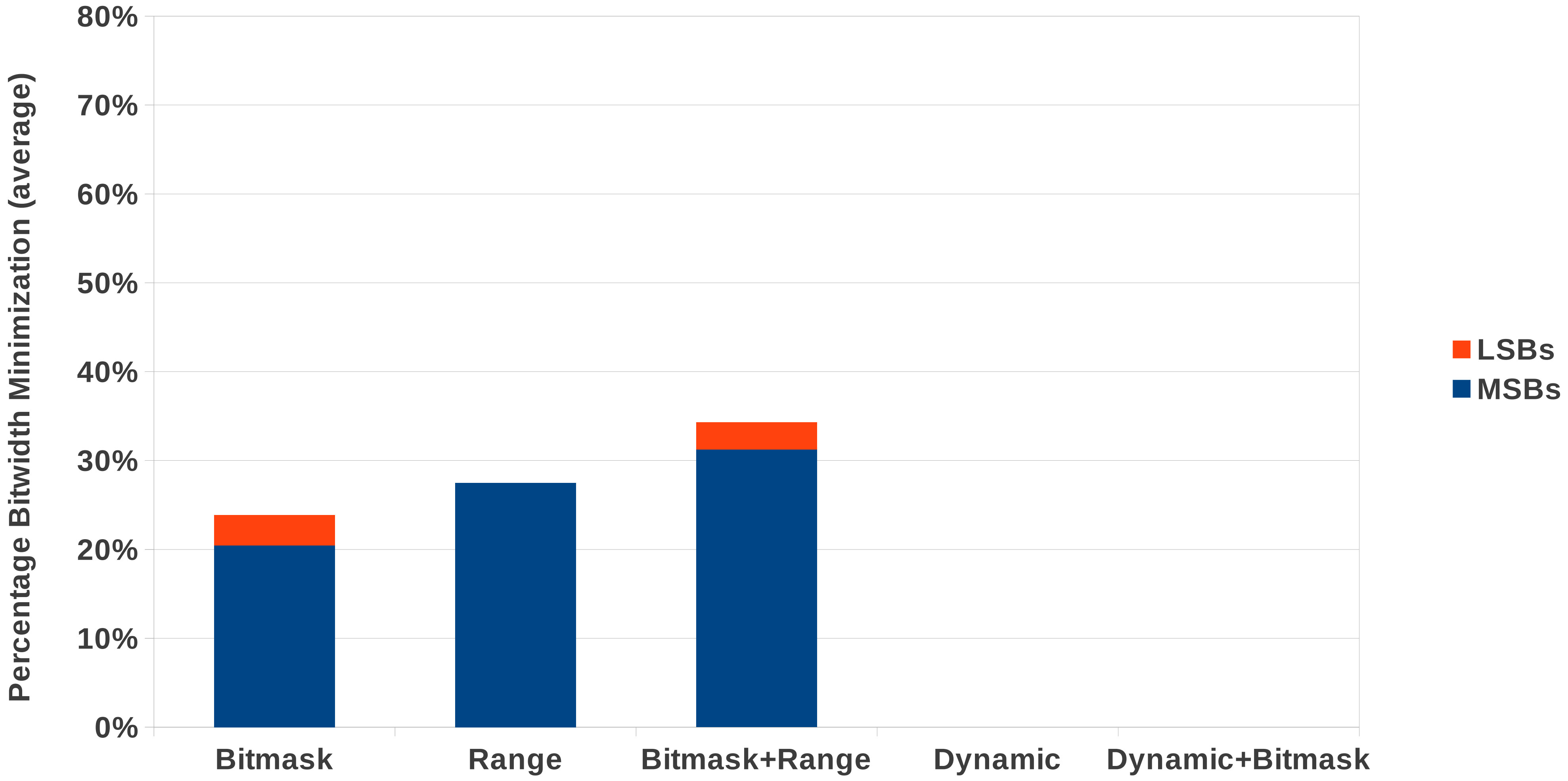
Bitwidth Reduction Results



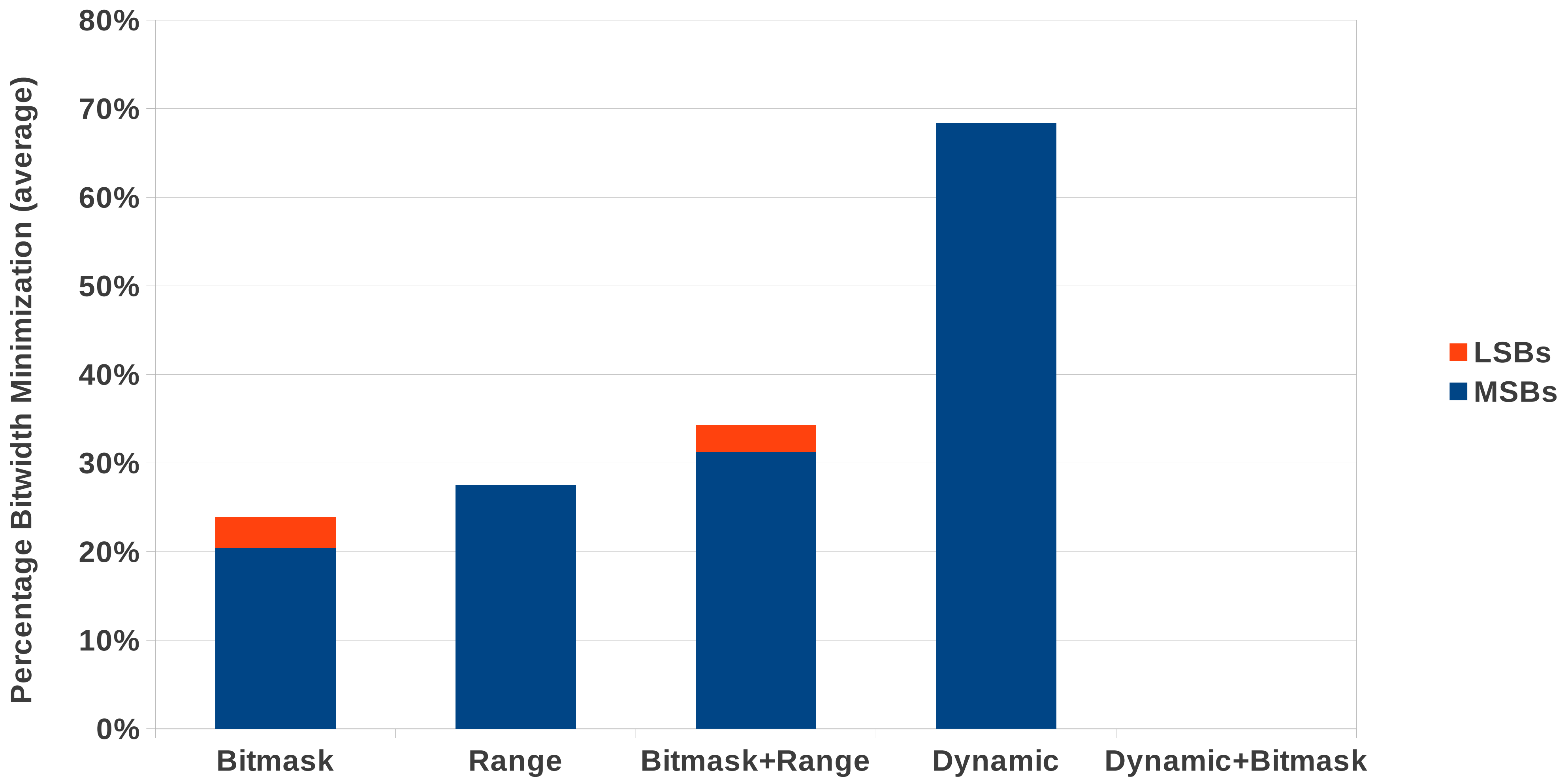
Bitwidth Reduction Results



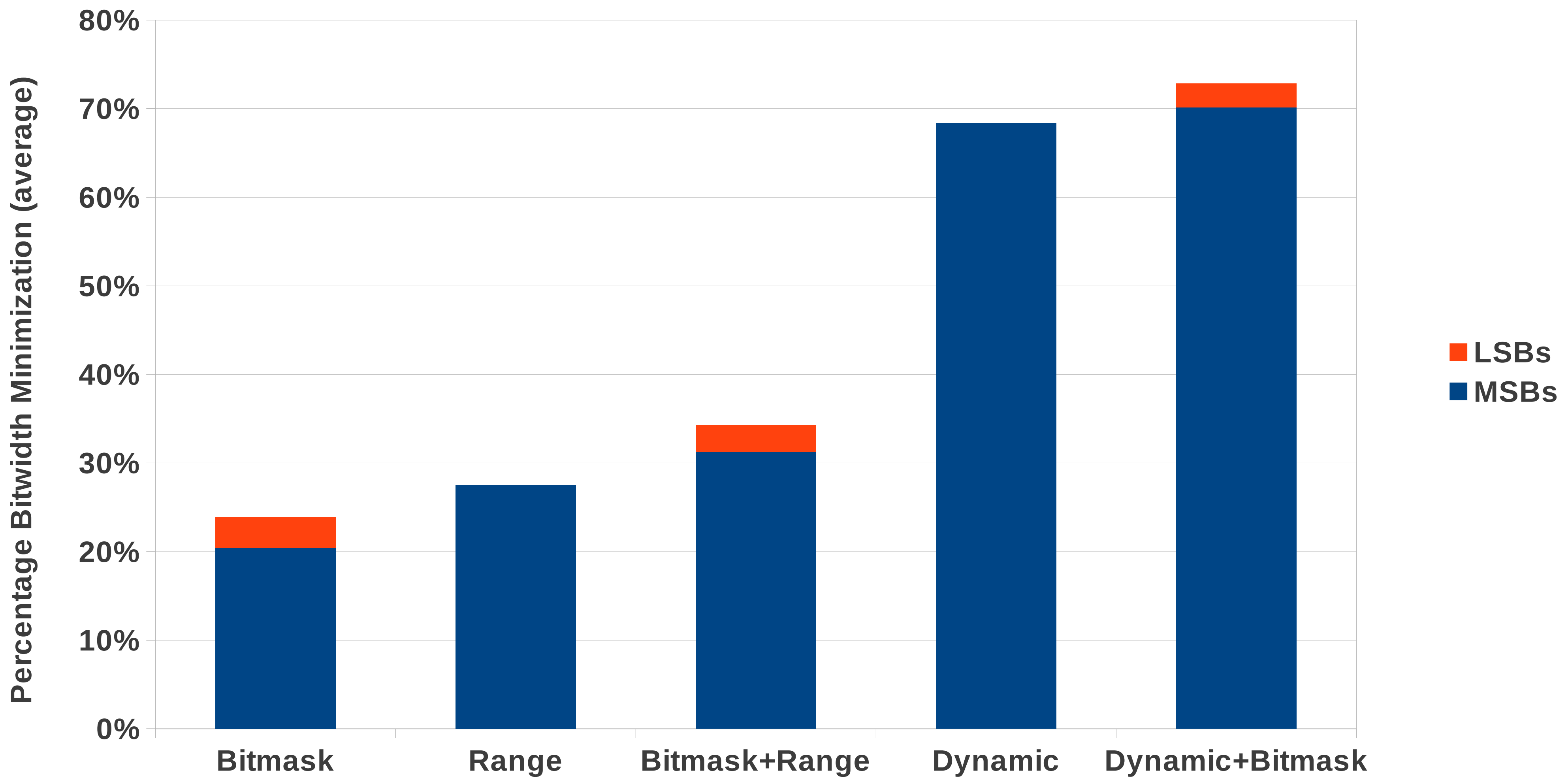
Bitwidth Reduction Results



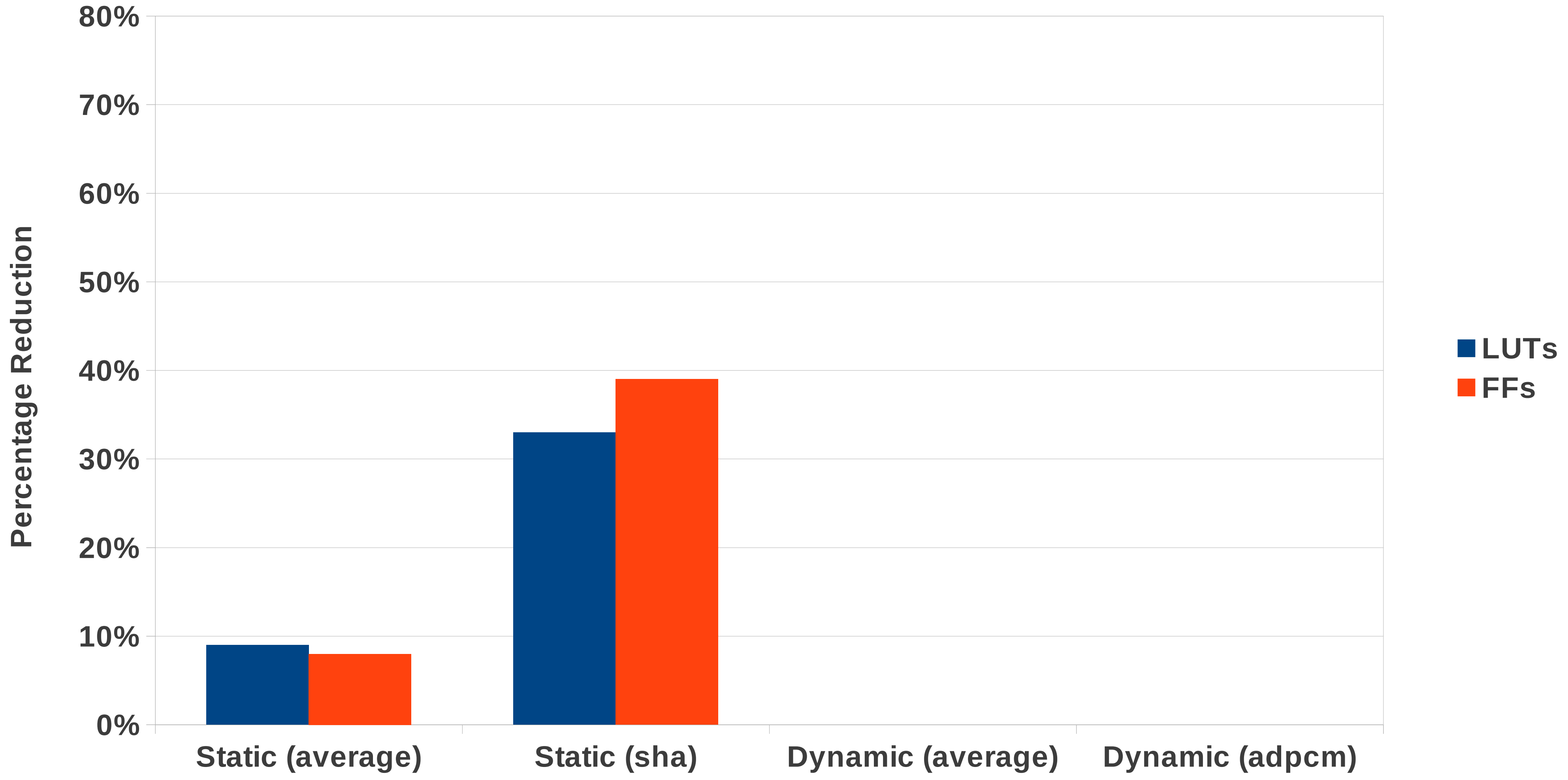
Bitwidth Reduction Results



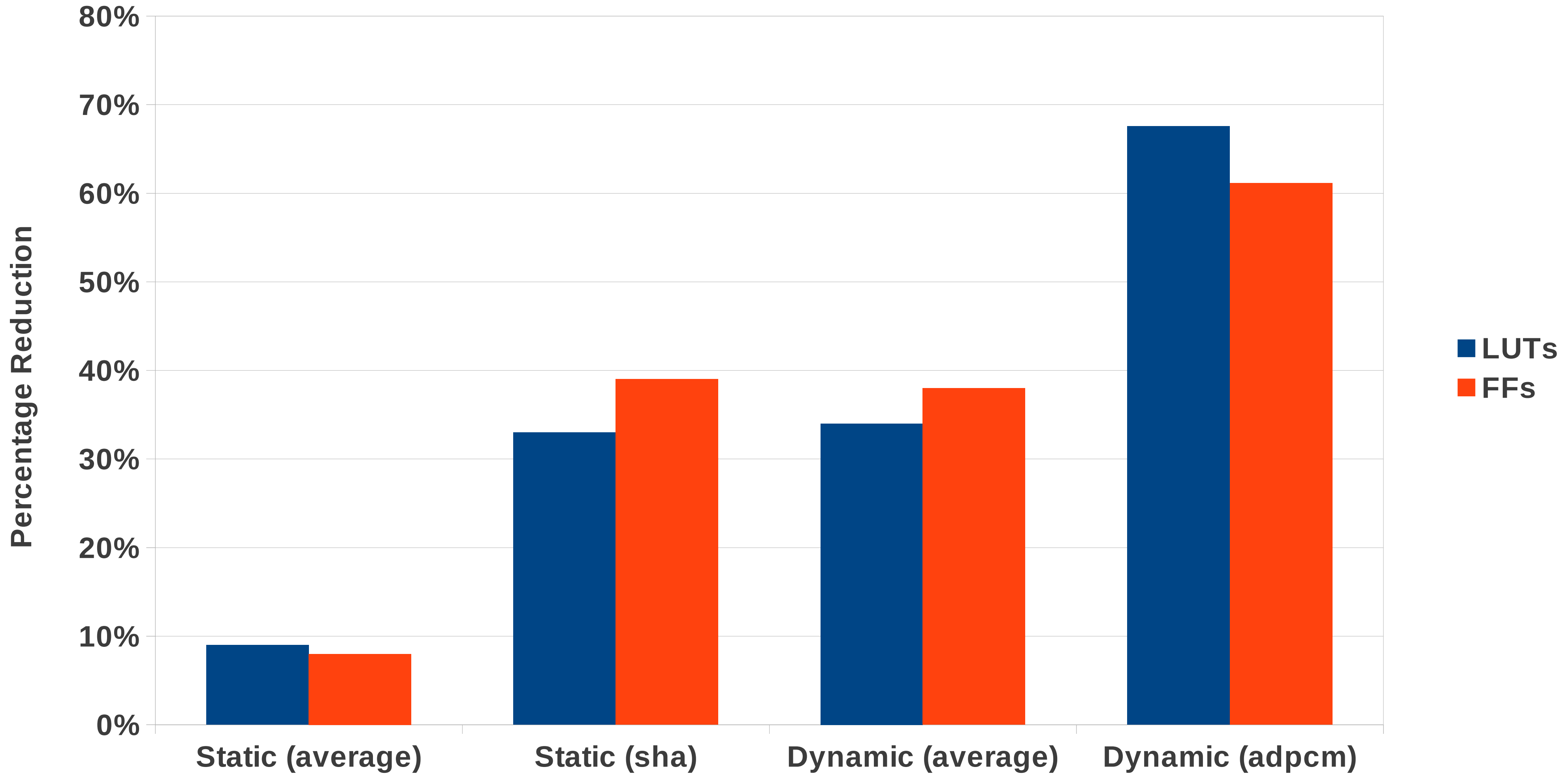
Bitwidth Reduction Results



Area Results



Area Results



Conclusions and Future work

- Opportunities exist to optimize instruction bitwidths in HLS that are not present in RTL synthesis.
  - 9% area improvement over Quartus.
- Using range and bitmask analysis approaches together yields better results than using either in isolation.
- Excellent dynamic range-analysis results show that program information can be used to further reduce area.
  - In hybrid system, minimized HW with SW fallback.
  - User hints for variable use.

Bitwidth minimization will be part of the LegUp 3.0 release.
Soon to be available at:

http://legup.eecg.utoronto.ca