Mapping Multiplexers onto Hard Multipliers in FPGAs

Peter Jamieson and Jonathan Rose

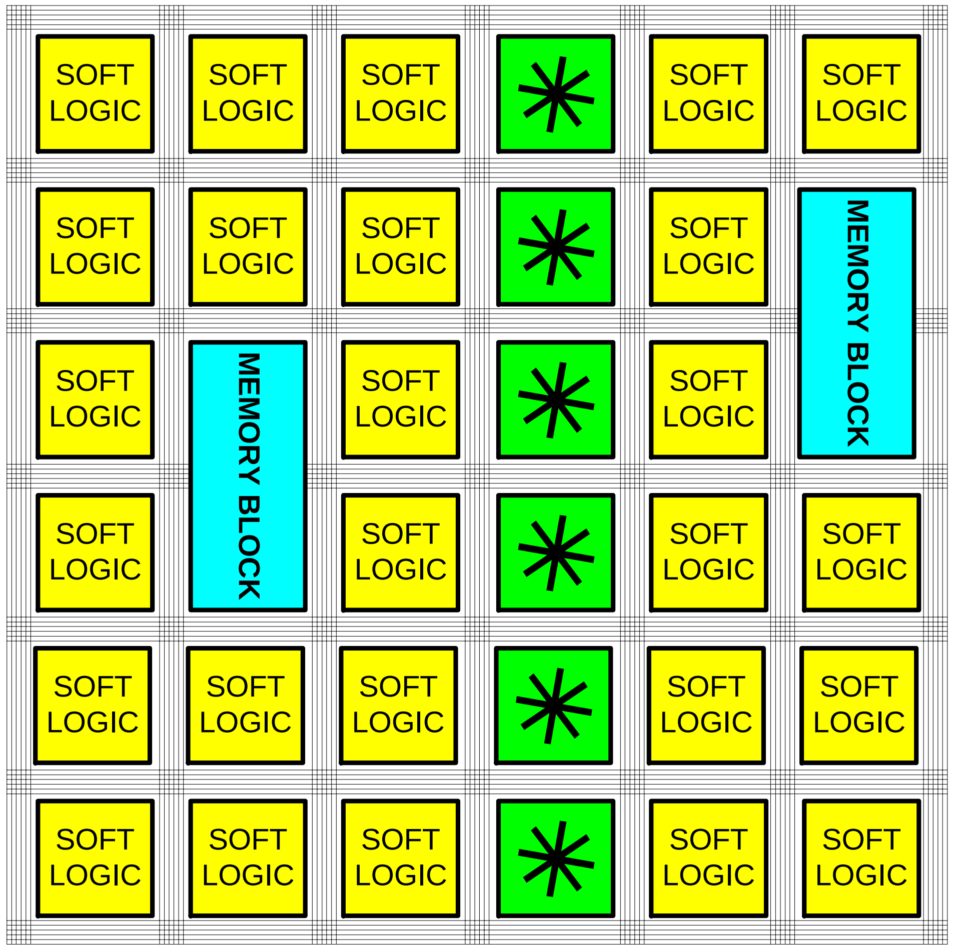
The Edward S. Rogers Sr. Department of Electrical and Computer Engineering

University of Toronto

Modern FPGAs

Consist of:

1. Programmable logic and routing
   - soft logic fabric
2. Less-programmable dedicated circuits
   - hard structures
   - e.g. multiplier
   - e.g memory block



FPGA Hard Structures

- Are very useful **when used**
  - major gain in speed, logic density
- Are very **wasteful when not used**
- The logic area goes unused, but worse…

Programmable Routing is Wasted!

- 70%-90% of FPGA area is routing!
- Everyone pays for hard structures
  - – but not everyone can use all of them

What can we do about waste?

Try to employ unused hard structures

1. Use multiplier as barrel shifter
   - [Gigliotti, 2004]
2. Use memory block as lookup table to implement combinational logic
   - [Wilton, 2002; Cong and Xu, 2000]

Our Idea

- Employ unused multipliers to implement **multiplexers**
  - Saves programmable logic
- Multiplexers are common in many applications:
  - Networking
  - Processors
  - Many kinds of datapath

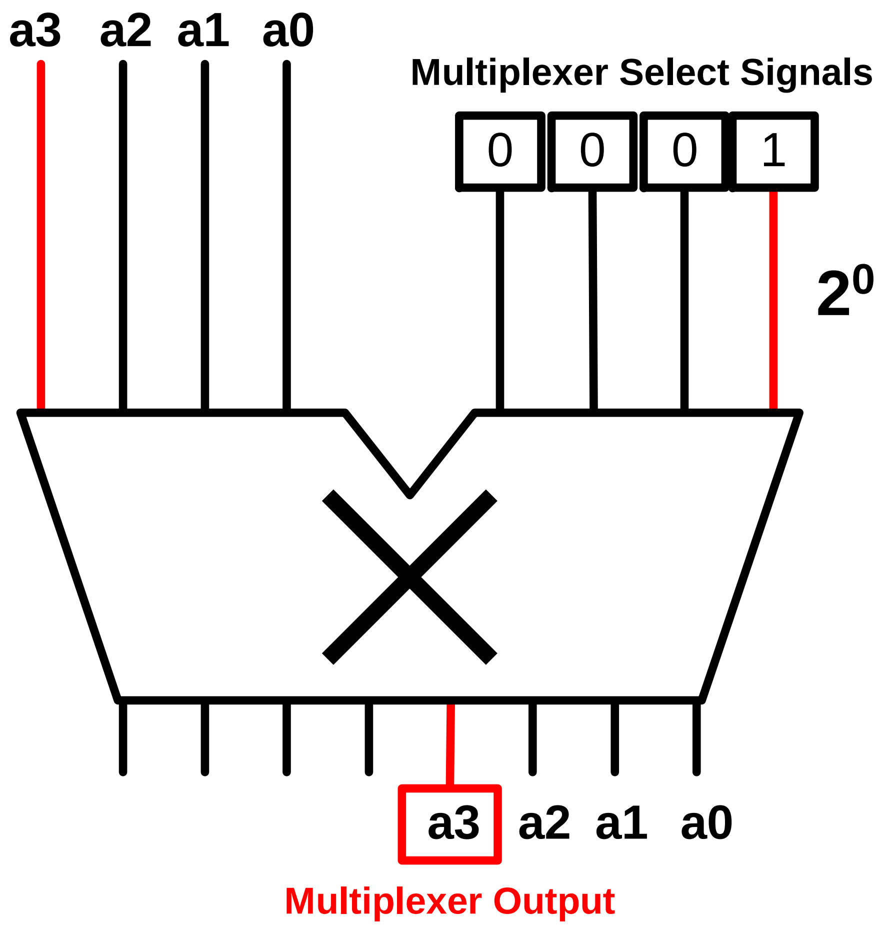
How to Make Multiplexers from Multipliers

Recall:

- multiply a number by 2^n shifts the binary number by **n** bit positions
- we're going to use it to make expensive shifters/multiplexers
  - Not actually expensive, since multiplier was going unused!

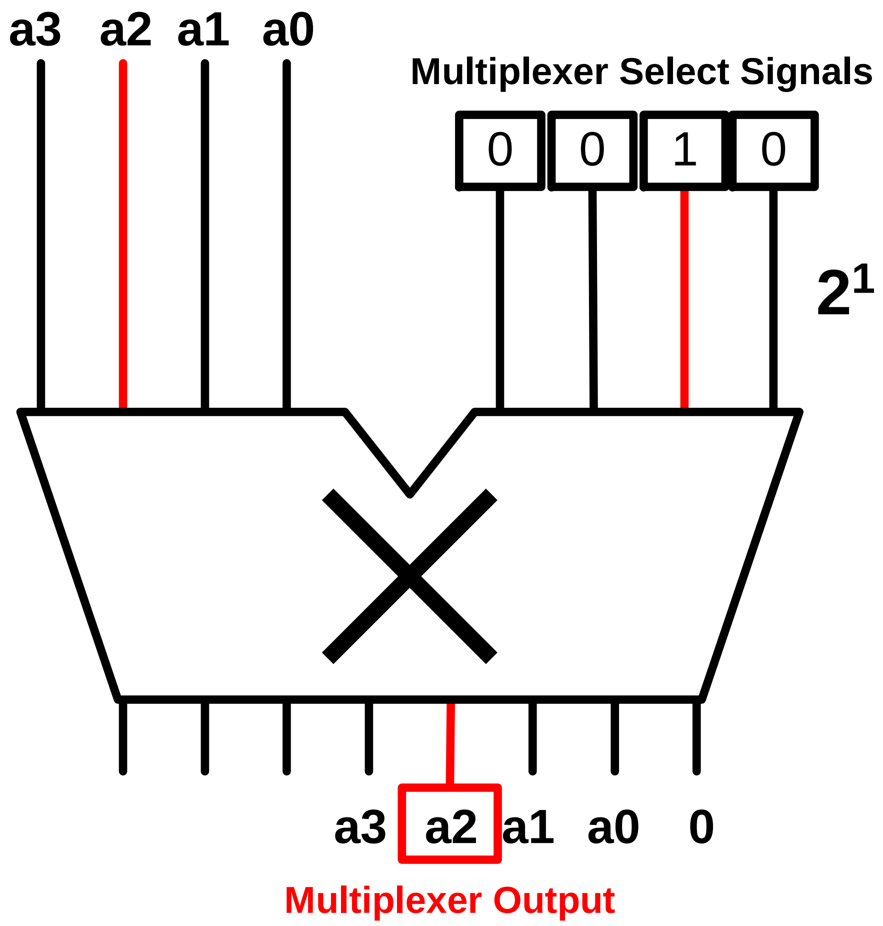
Use Multiplier to Shift/Select

- Use shift to select input to pass through



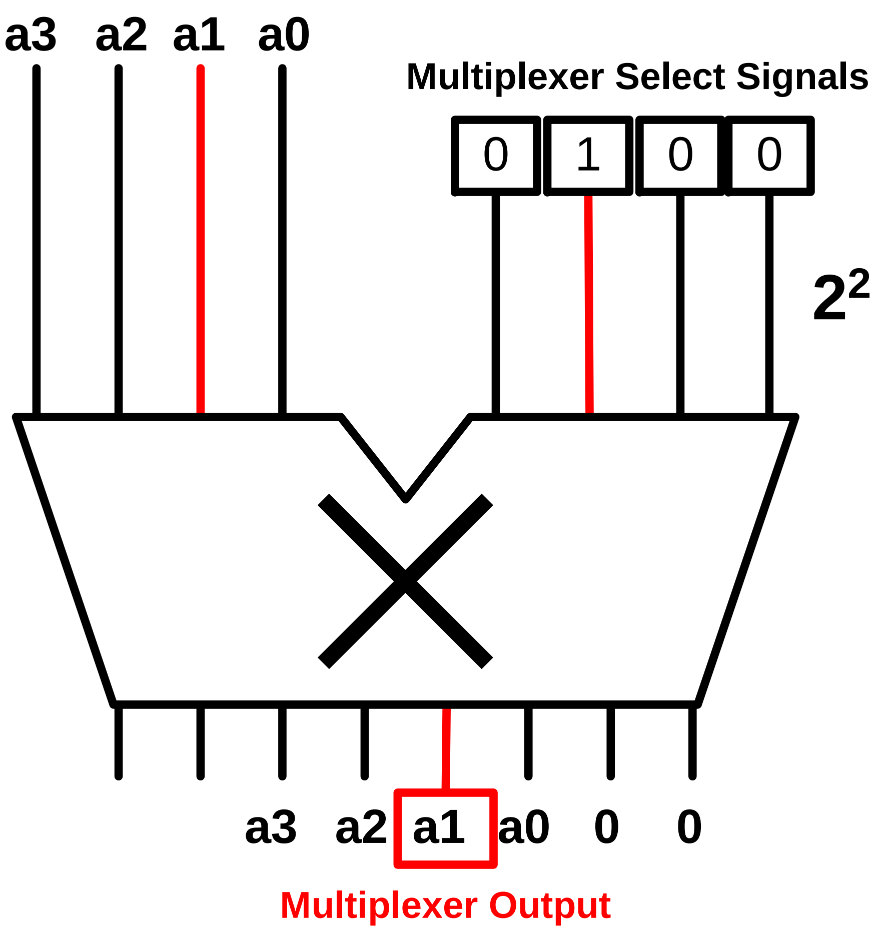
Use Multiplier to Shift/Select

- Use shift to select input to pass through



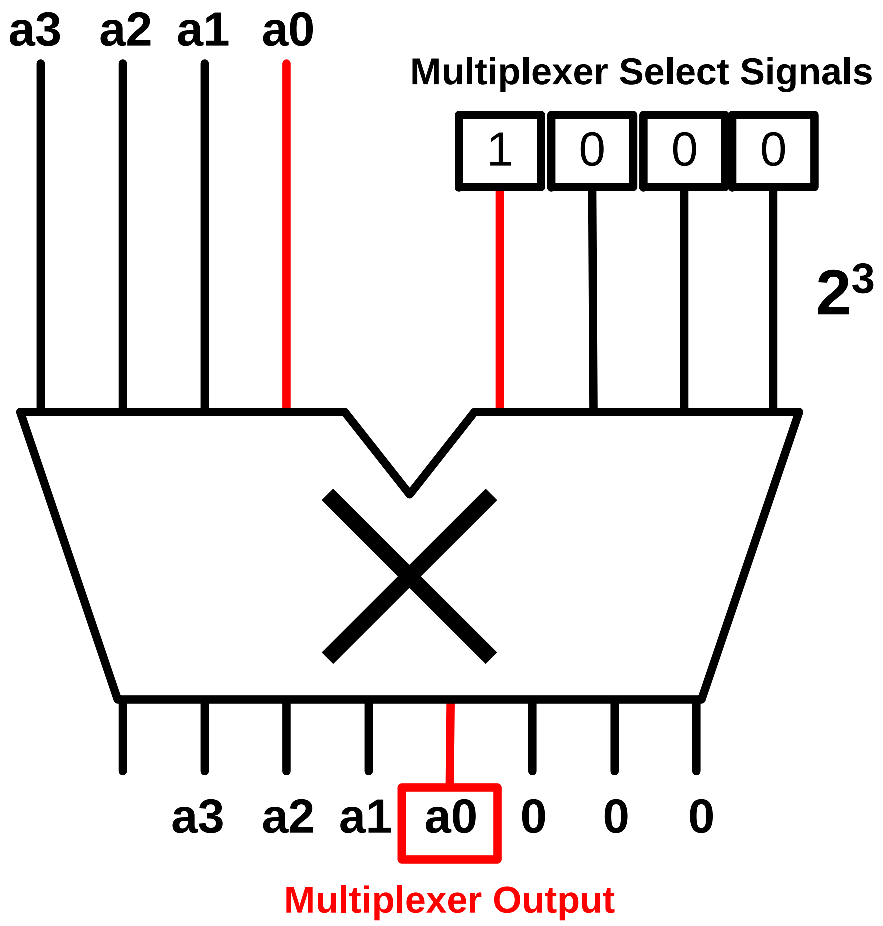
Use Multiplier to Shift/Select

- Use shift to select input to pass through



Use Multiplier to Shift/Select

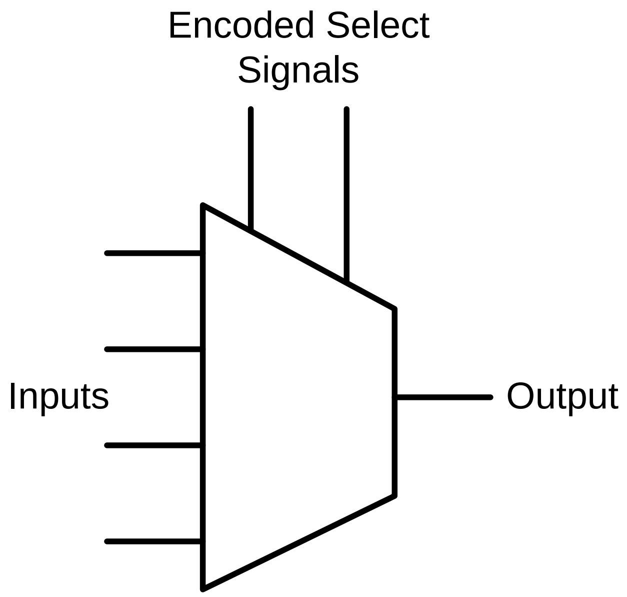
- Use shift to select input to pass through



Doesn’t look like a multiplexer

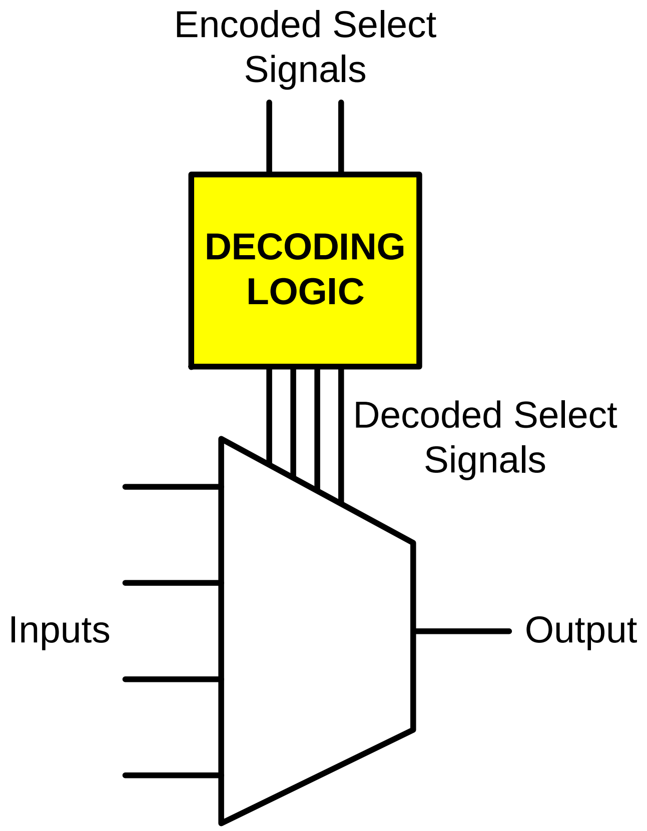
What we know:

- Multiplexer has encoded select signals



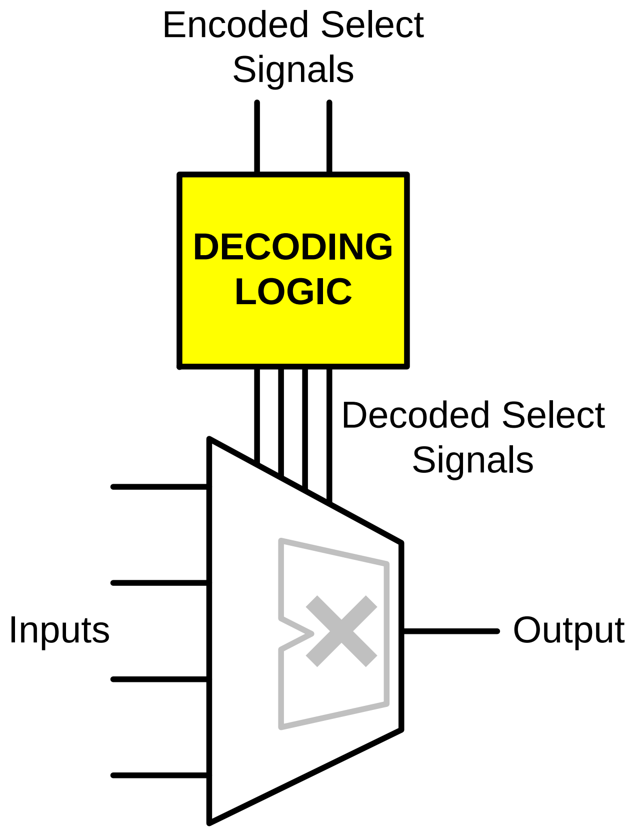
Multiplier select Requires Decoded Selects

- Multiplier is only used to select signal
- Requires that select signals are decoded in programmable logic



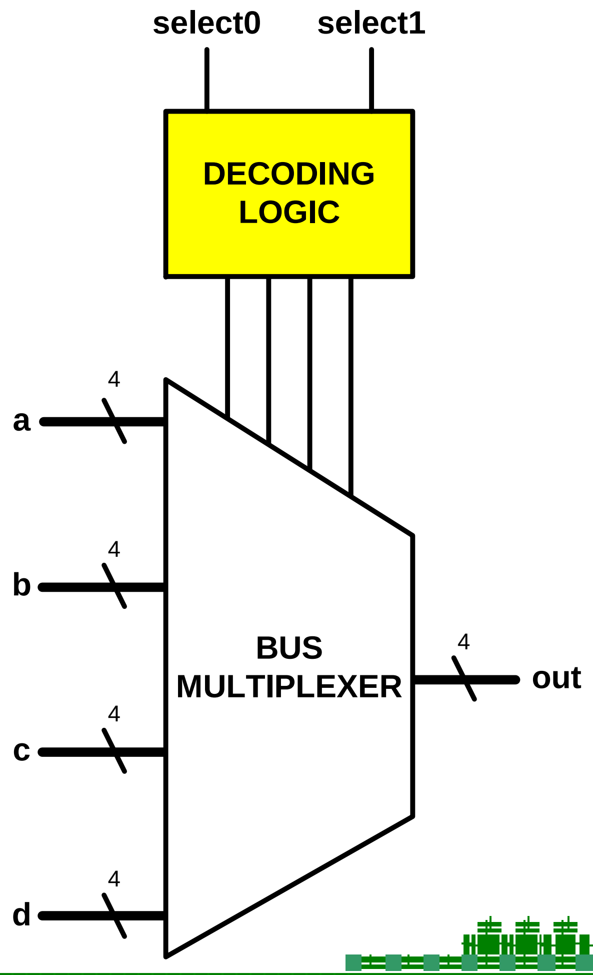
Multiplier select Requires Decoded Selects

- Multiplier is only used to select signal
- Requires that select signals are decoded in programmable logic
- Decoding costs, but …



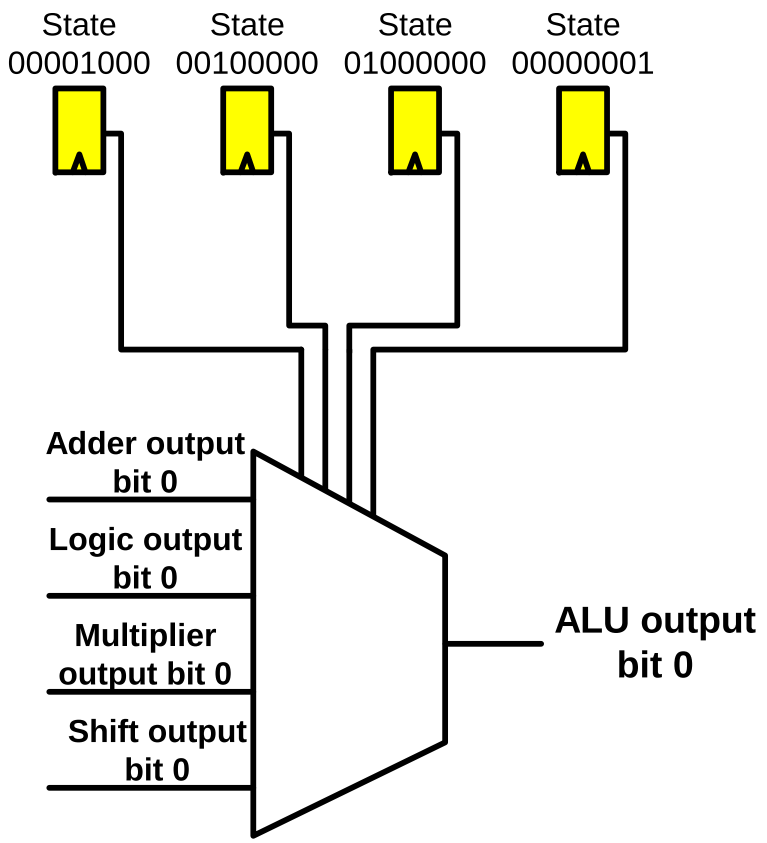
1. Can Amortize Decoding Logic

- For busses share signals to multiplex
- Only need to generate decoded selects **once**



2. Decoded Selects are Already There!

- Imagine a one-hot encoded state machine controlling the output of an ALU



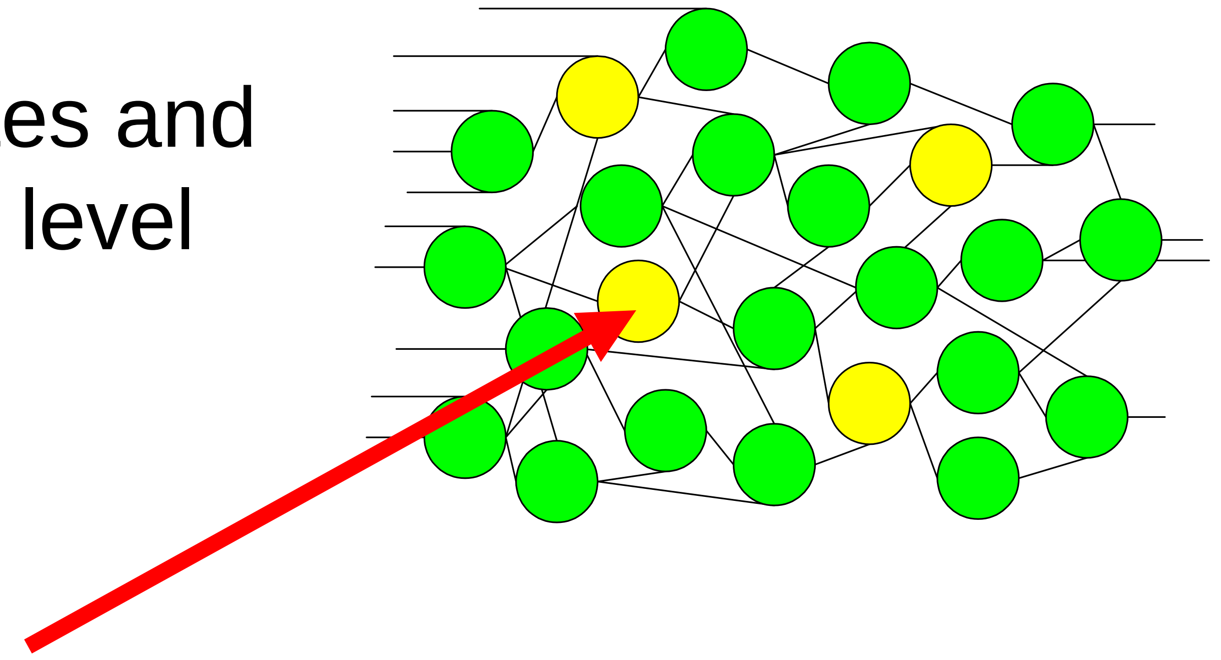
Issue: Speed

- How will mapping affect circuit speed?
  - Example: 9:1 multiplexer
    - Soft logic mux = 422 MHz;
    - Built with 9*9 hard multiplier = 307 MHz.
    - **115 MHz** slowdown!
- Problem if employed on critical path, or causes path to become critical

Mapping Algorithm

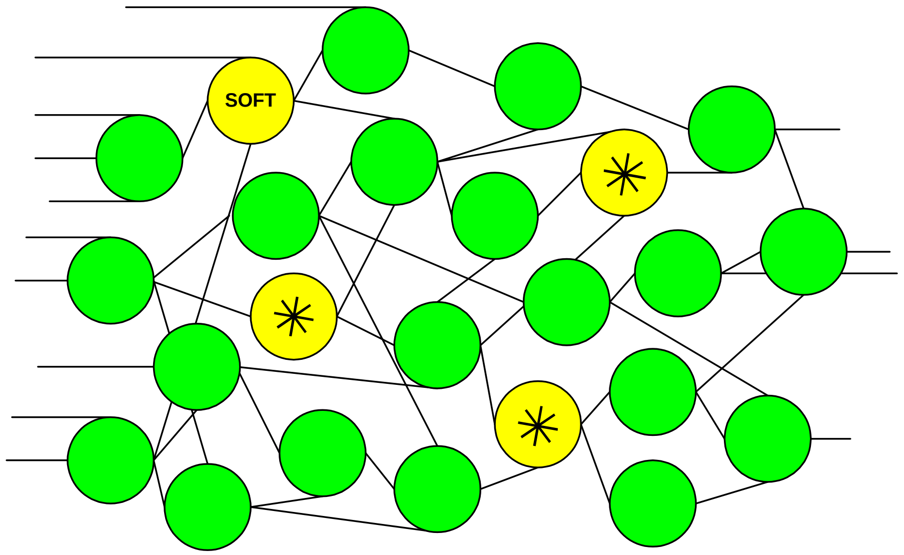
Input

- Netlist of gates and some higher level structures
  - Logic
  - Multipliers
  - Memories
  - Multiplexers



Output

- Similar netlist with multiplexers mapped to
  - – hard multipliers <u>or</u>
  - – soft logic



Goal of Algorithm

- Largest reduction in number of soft logic elements used
  - Potentially allow user to reduce part size, save money

Step 1

- Group all multiplexers that share common encoded select signals

Step 2

- For each group or multiplexer, estimate the area saving
  - Let
    - CS(size) = cost of implementing soft multiplexer
    - CD(size) = cost of decoding logic
    - |M| = number of multiplexers in a group
  - SAVINGS = (ΣCS(size) – CD(size)) / |M|

Step 3

- Sort the groups and multiplexers in the order from greatest savings to least savings

Step 4

- Remove from list multiplexers on path that includes a multiplier
  - Hard multipliers often sit on critical path
  - Heuristic avoids speed reduction

Step 5

- Greedily select most-beneficial multiplexers to map
- Go back to step 4
- Finish
  - when no unused multipliers left
  - When no multiplexers left to map

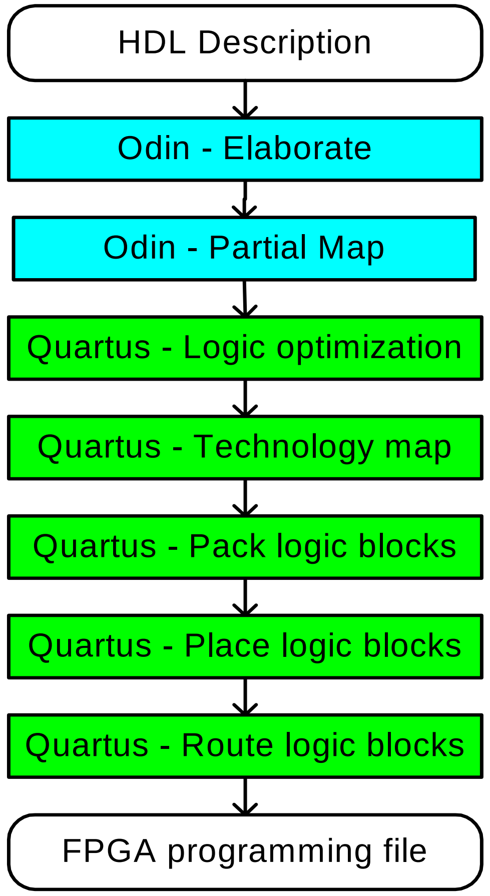
Experimental Setup

Experimental Setup

- Will compare with and without mapping
- FPGA Used: Stratix I
- CAD Flow based on Altera Quartus 4.1

CAD Flow

- New front-end synthesis
  - Odin [FPL 2005]
- Plugs into Quartus back-end from logic synthesis down


HDL Description
Odin - Elaborate
Odin - Partial Map
Quartus - Logic optimization
Quartus - Technology map
Quartus - Pack logic blocks
Quartus - Place logic blocks
Quartus - Route logic blocks
FPGA programming file

Benchmarks

- Collection of Verilog Benchmarks
  - Opencores
  - SCU-RTL
  - Texas-97
  - Benchmark Suite for Placement-2001
  - Local designs converted from VHDL
    - Raytrace
    - Molecular Dynamics
    - Stereo Vision
- Results are dependent on these benchmarks

Results

- Will present results separately for benchmarks with and without multipliers

Results – Benchmarks without Multipliers

Benchmarks	Percent reduction in LEs	Percent Speed Gain
cordic_8_8	4%	-149%
cordic_18_18	1%	-25%
MAC1	5%	-40%
MAC2	2%	-12%
des_area	18%	5%
des_perf	5%	-145%
sv_chip0	1%	-30%
sv_chip0_no_mem	1%	-46%
sv_chip3_no_mem	19%	-168%
rt_frambuf_top	9%	-58%
rt_frambuf_top_no_mem	1%	-76%
rt_boundtop	2%	-31%
rt_boundtop_no_mem	5%	-33%
Geometric Average	6%	-54%

- Average reduction in LEs: 5.8%
- Average <u>reduction</u> in speed: 54%

Results – Benchmarks with Multipliers

Benchmarks	Percent reduction in LEs	Percent Speed Gain
fft_258_6	1%	1%
iir1	0%	1%
iir2	11%	-1%
fir_3_8_8	0%	0%
fir_24_16_16	0%	0%
fir_scu_rtl	42%	-13%
diffeq_f_systemC	7%	0%
diffeq_paj_convert	12%	0%
sv_chip1	0%	-1%
rt_raygentop	5%	3%
rt_raygentop_no_mem	5%	-3%
oc45_cpu	6%	5%
reed_sol_decoder1	8%	-1%
reed_sol_decoder2	5%	1%
md	0%	0%
Geometric Average	8%	0%

- Average reduction in LEs: 7.6%
- Average <u>reduction</u> in speed: %0.5

Summary

- Technique reduces Logic Element Count by 6.8% on average across all benchmarks
- Reduction in speed varies:
  – Benchmarks that already employ multipliers almost no loss in speed
  – Benchmarks w/o multipliers: 54% reduction

Future Work

- Technique should improve if we can estimate speed of circuit paths at front-end
- Should architect a better hard structure that are more widely usable