



SOPC
WORLD
2004

Advanced Synthesis: Multiplexer Optimization

Objectives

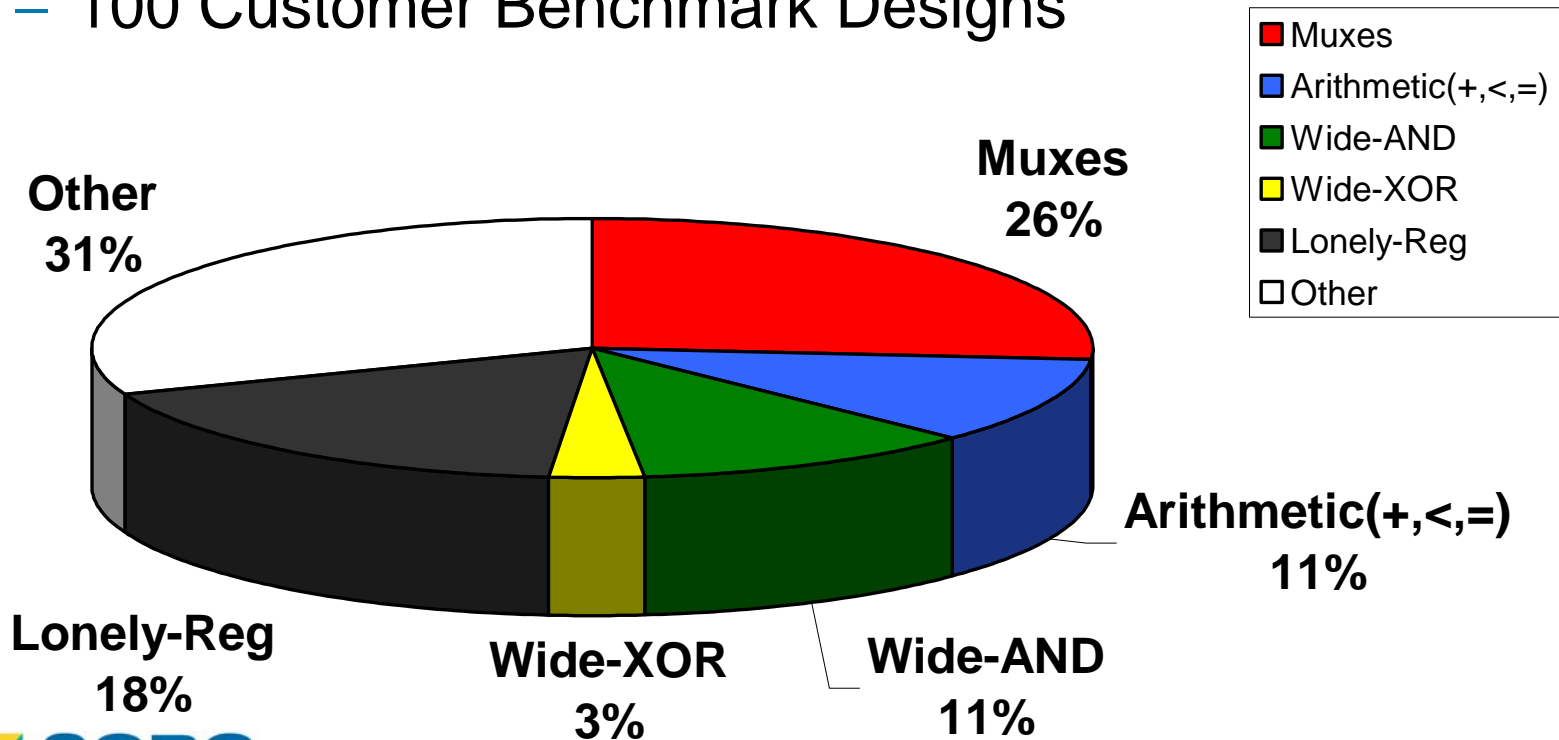
- Teach How to Get Most Out of HDL When Coding Muxes for 4-Input LUT-Based Devices
- Provide Insights Into How Synthesis Deals with Muxes
 - Research Done for Quartus II, Most Applies to Other EDA Tools Too
- Present New Quartus II Synthesis Enhancements and Future Enhancements
 - Includes Hidden INI Variable for MUX Optimization

Lessons Learned

- Typically, We Improve Synthesis/Fitting Results By Optimizing “Final” Designs
- In Reality, Many Customers Change Their Designs to Fit Synthesis!
 - Especially for New FPGA Architectures
 - Customers Don’t Know What Works Well
- Optimizations Often Require Human Knowledge of Design
 - Synthesis Can Not Always Know the Design Intent
 - Customer Is In Best Position to Improve QoR!
- Some of the Work Has to be Manual!

How Often Do Multiplexers Occur? Why Mux Optimization?

- Mean LE Usage After Synthesis
 - 100 Customer Benchmark Designs



Agenda

Background/Theoretical

Design Guidelines

Quartus II Project to Improve MUX Synthesis

Agenda - Background/Theoretical

- Where do Multiplexers come from?
- How Are Multiplexers Implemented in Synthesis (especially Quartus II)?
- An Improved Multiplexer Implementation
- Implicit Multiplexing
- Taking Advantage of Implicit Multiplexing



SOPC
WORLD
2004

Where Do Multiplexers Come From?

Types of Multiplexers

- Binary Multiplexers
 - CASE Statements
- Selector Multiplexers
 - CASE Statements
 - State Machines
 - IF Statements
- Priority Multiplexers
 - IF Statements
- The Synthesis Tool Chooses Which Kind of MUX to Implement from Your HDL!

How Are CASE Statements Synthesized?

- Selector Multiplexer Used in Quartus II:
 - If Selecting Is Based on States of a State Machine
 - If Case Statement Covers $< 1/8$ of All Possible Cases
- Binary Multiplexer Used in Quartus II:
 - In All Other Cases
- Synplicity Almost Always Uses Selector
 - Why? It's Up to the Vendor to Make the Choice



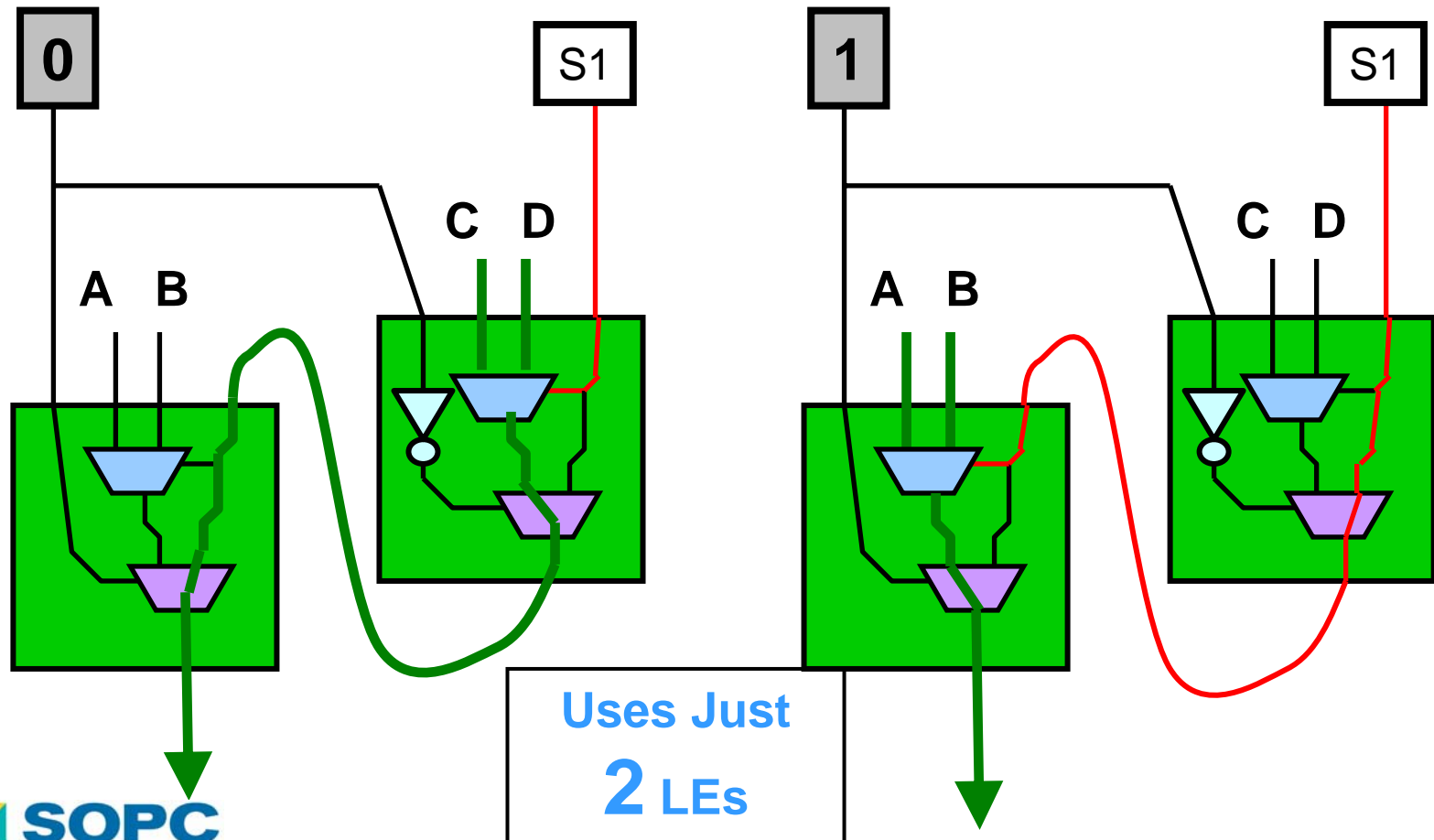
SOPC
WORLD
2004

How Are Multiplexers Implemented in Synthesis?

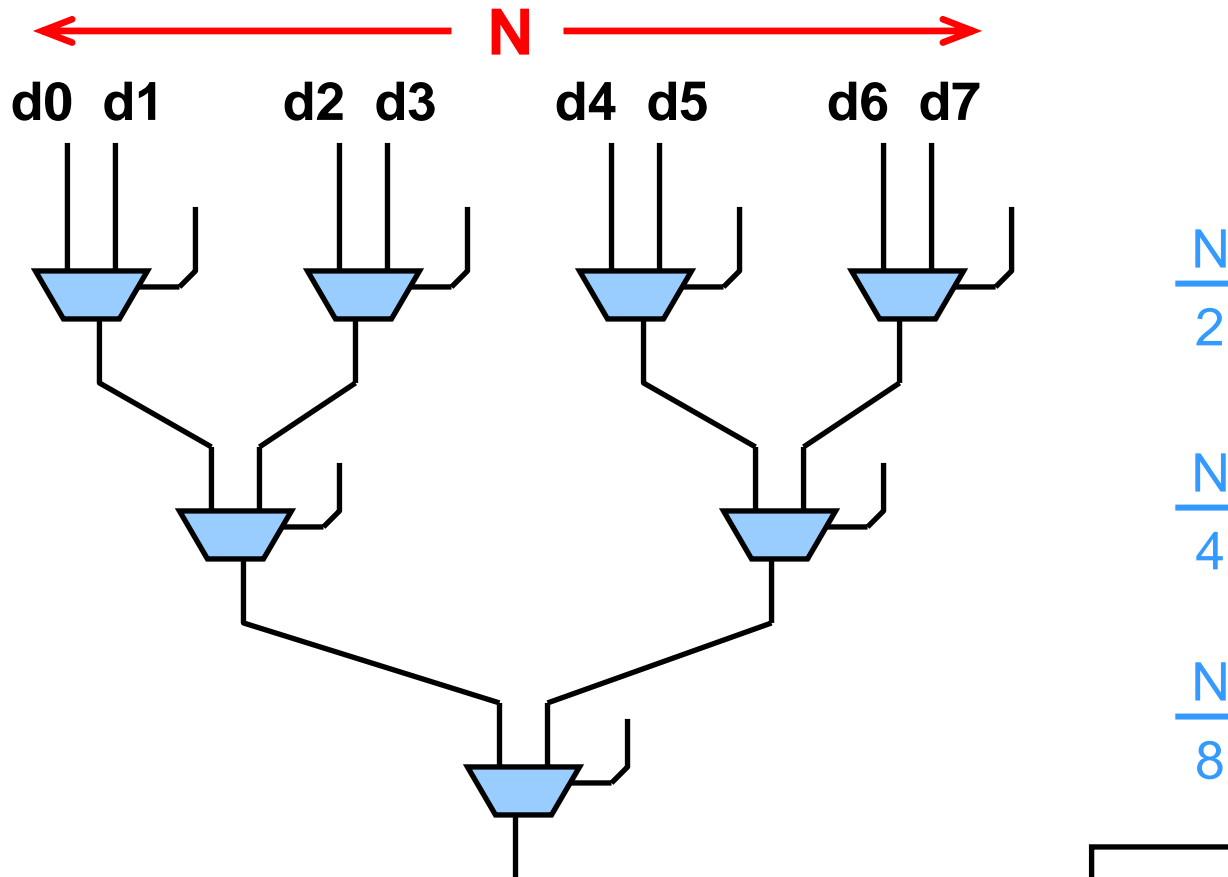
Multiplexer Implementation

- Binary Multiplexers
 - 4:1 Multiplexer
 - LPM_MUX
- Selector Multiplexers
- Priority Multiplexers

Efficient 4:1 Mux: How It Works

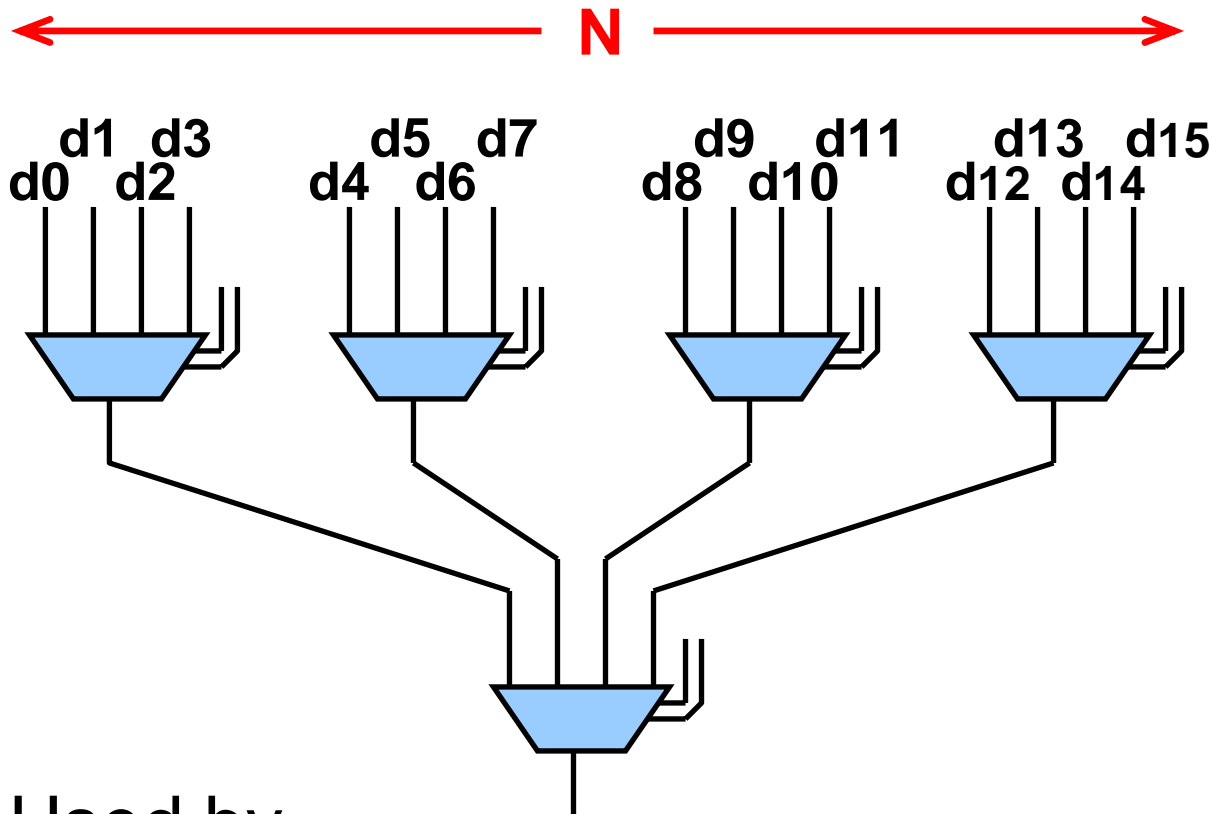


2:1 Mux Tree Forms N:1 Mux



1.00 N LEs

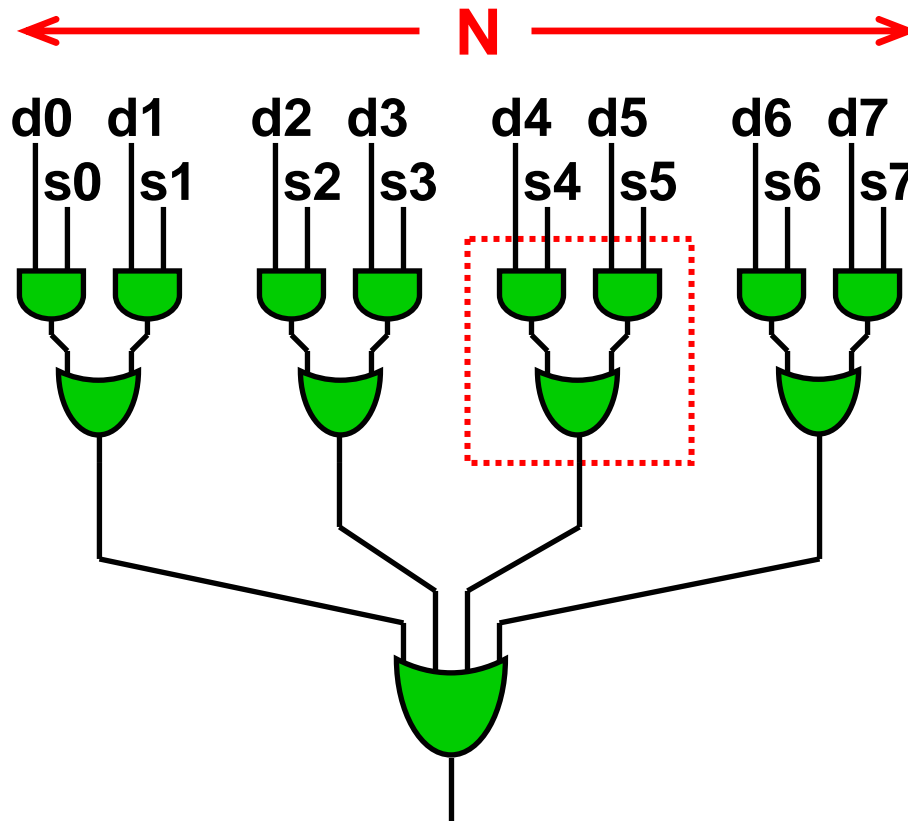
4:1 Mux Tree Forms N:1 Mux



■ Used by
LPM_MUX

0.66 N LEs

Selector Mux

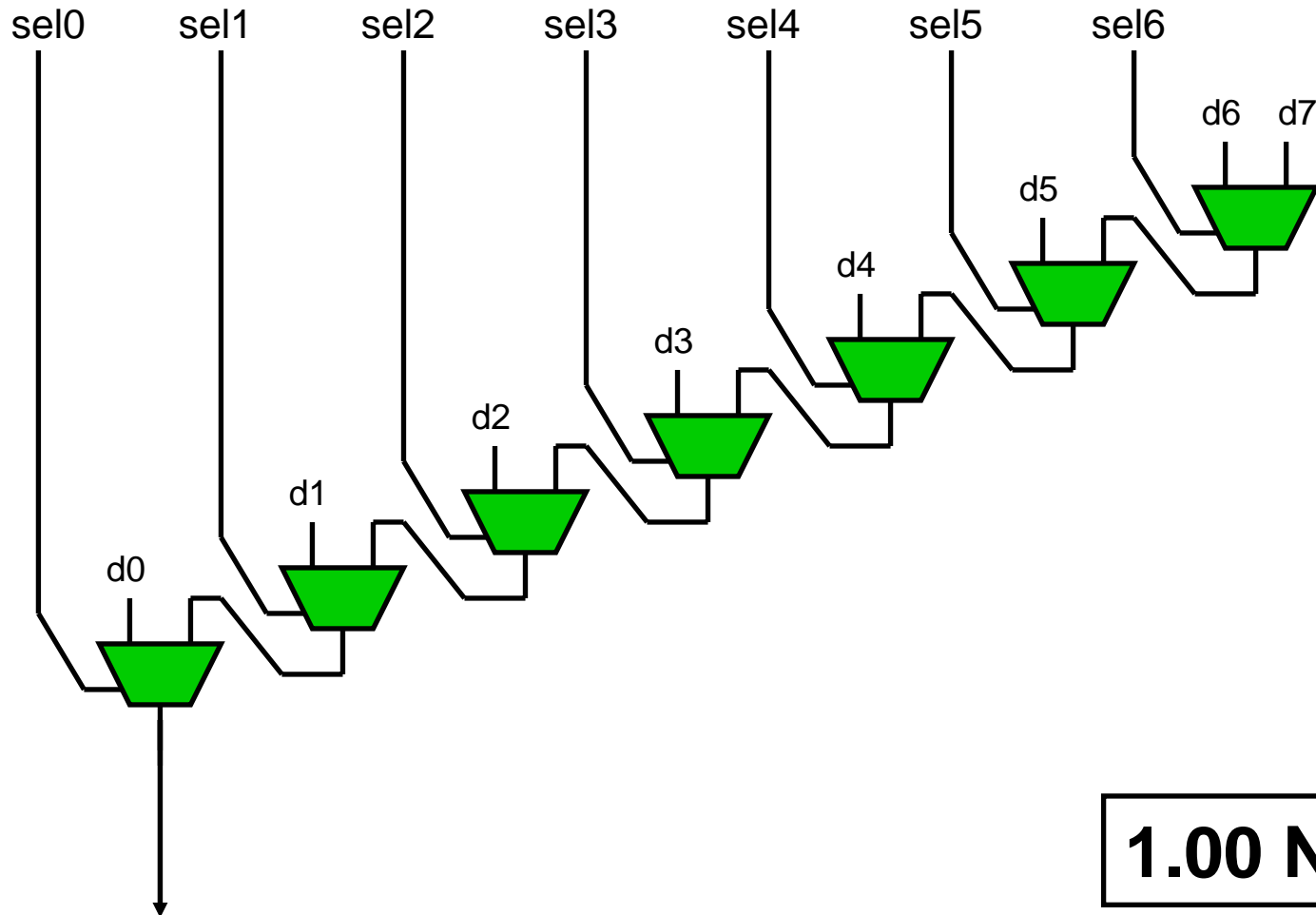


$$\frac{N}{2}$$

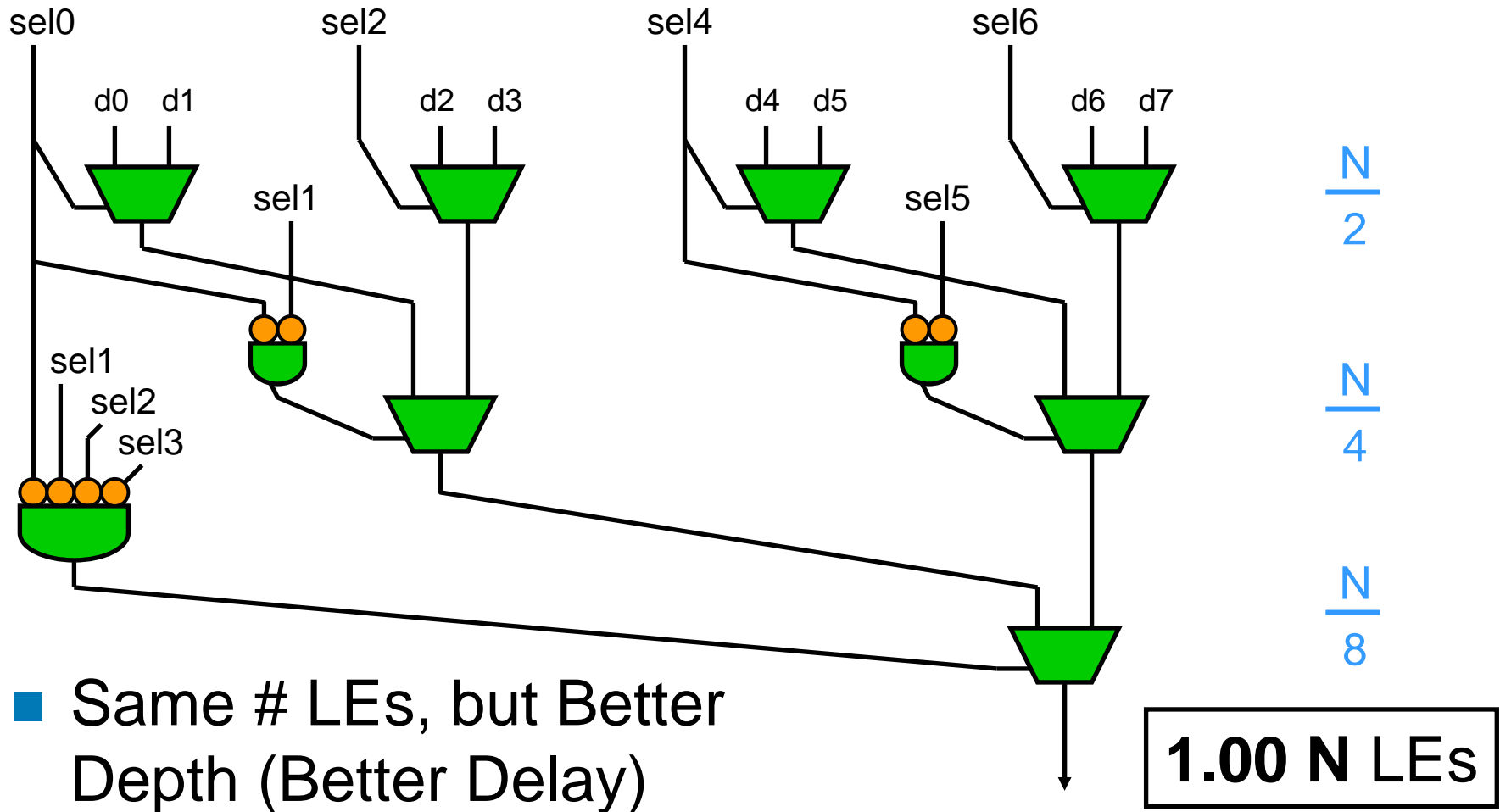
$$\frac{N}{2} \left(\frac{1}{4} + \frac{1}{16} + \dots \right)$$

$$0.66 N \text{ LEs}$$

Priority Mux



Priority Mux: Quartus II-Optimized



- Same # LEs, but Better Depth (Better Delay)



SOPC
WORLD
2004

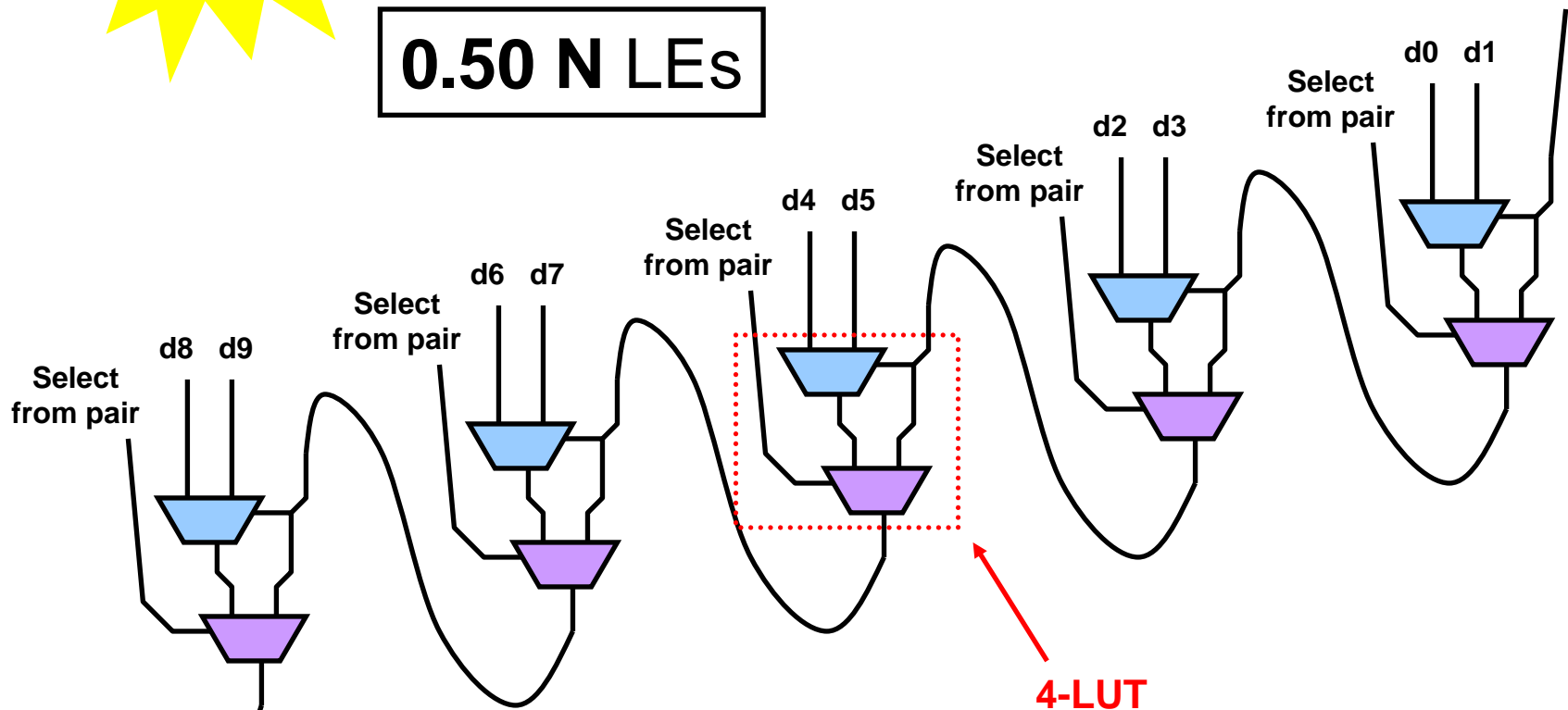
An Improved Multiplexer Implementation: Linear Mux

Linear Mux : How It Works...

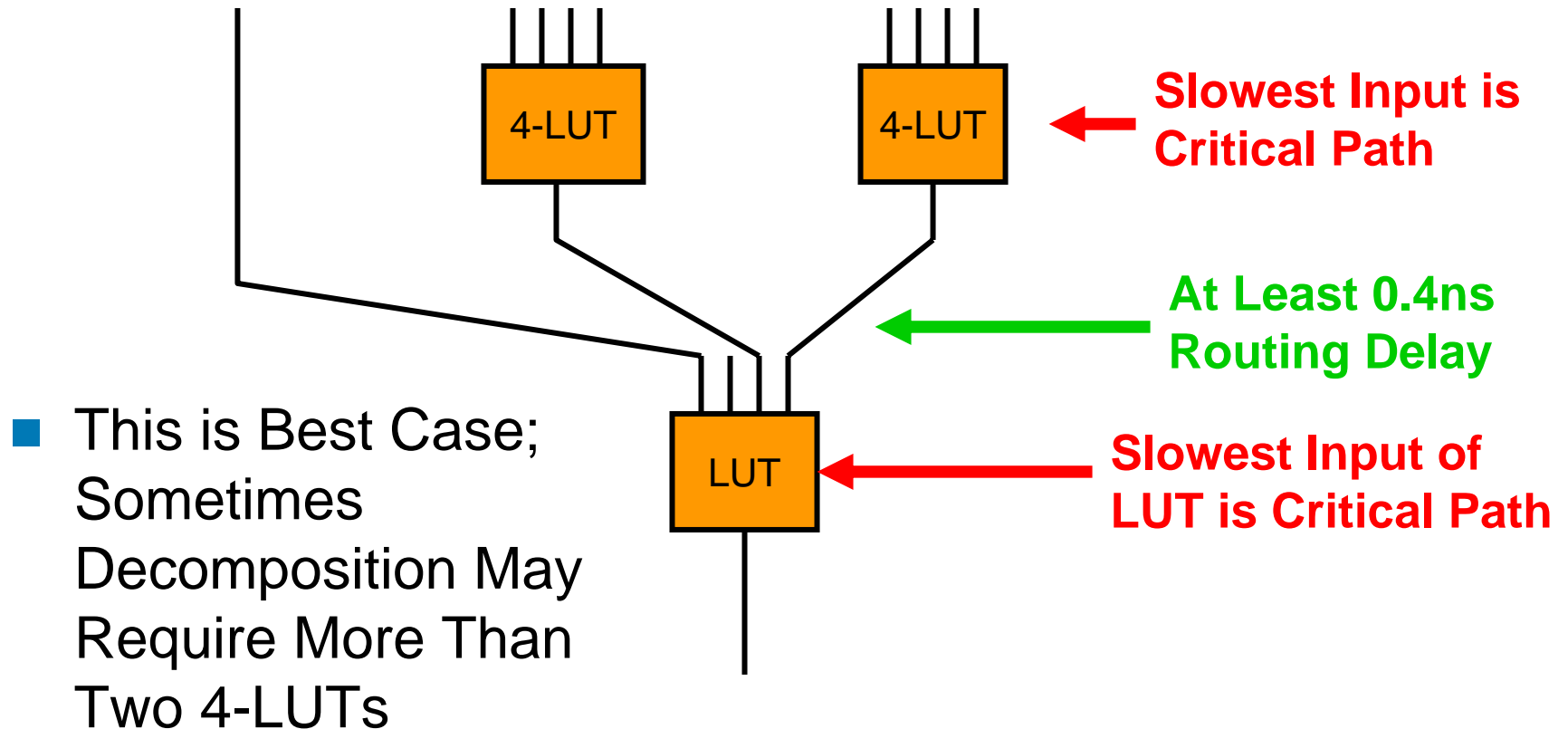
-25 %

0.50 N LEs

Select
Odd/Even



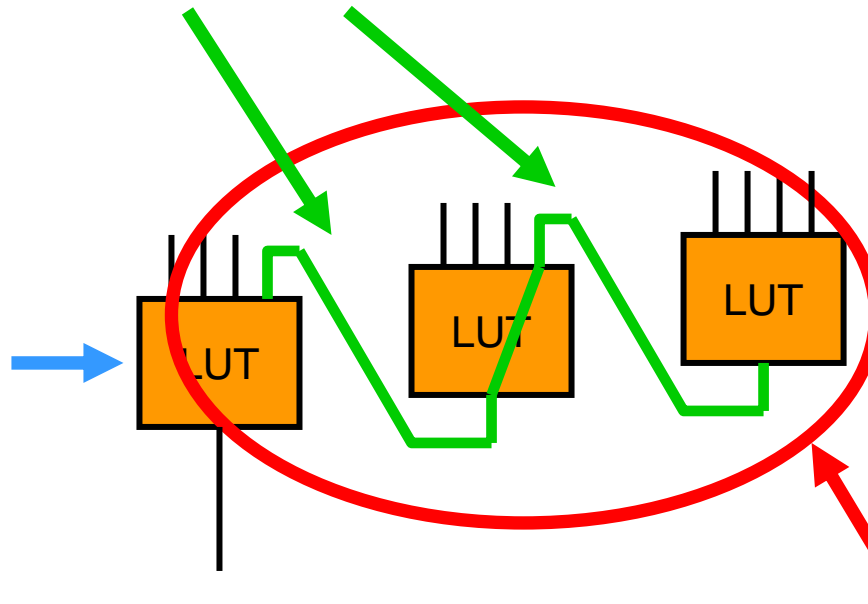
Tree of LUTs is Not Necessarily Optimal for Delay



Chained Logic Can Be Faster

Special Fast Routing Chains
on Stratix/Cyclone
(Cascade Chain)

Fastest Input
of LUT on
Critical Path



0.365ns



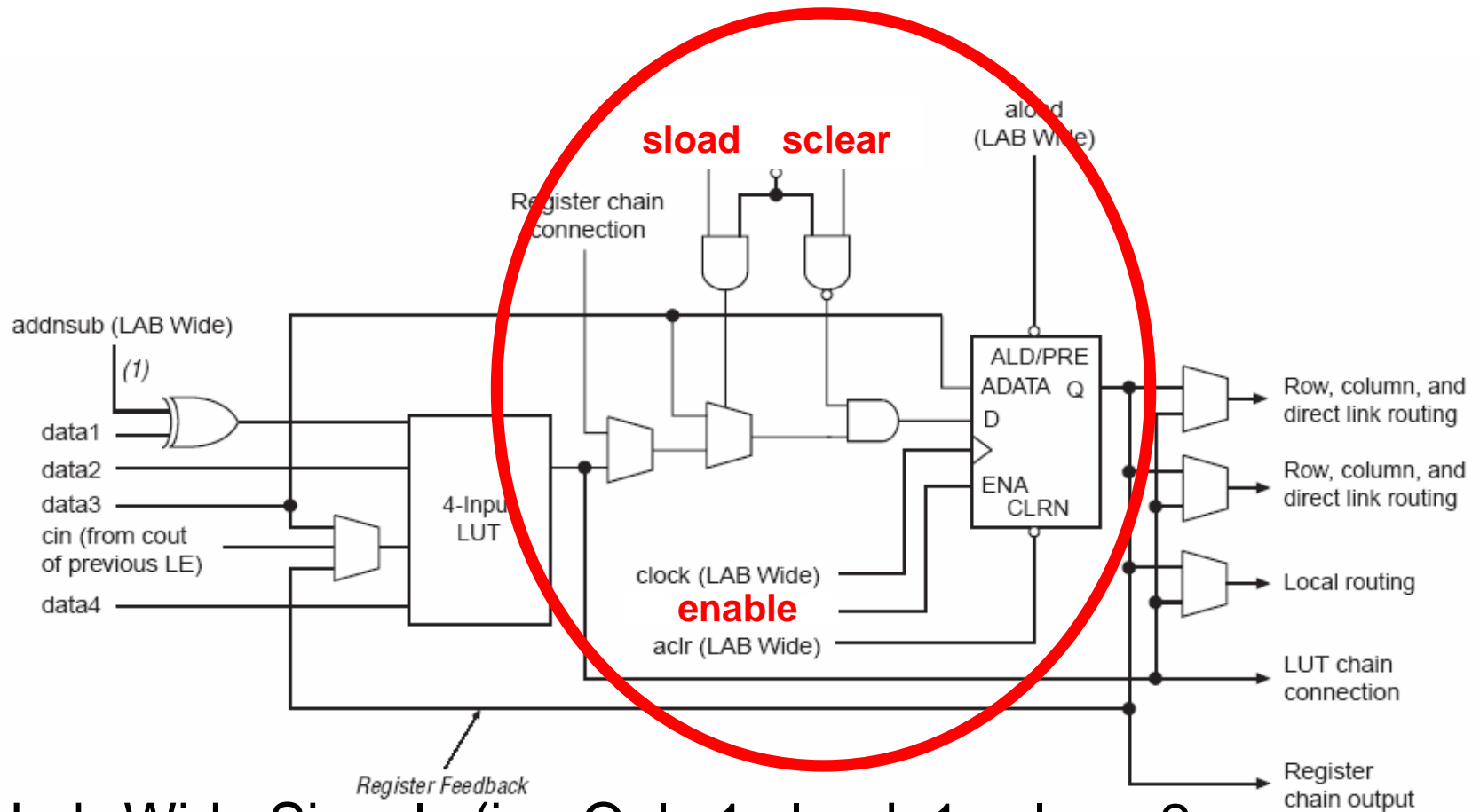
SOPC
WORLD
2004

Implicit Multiplexing

Implicit Multiplexing

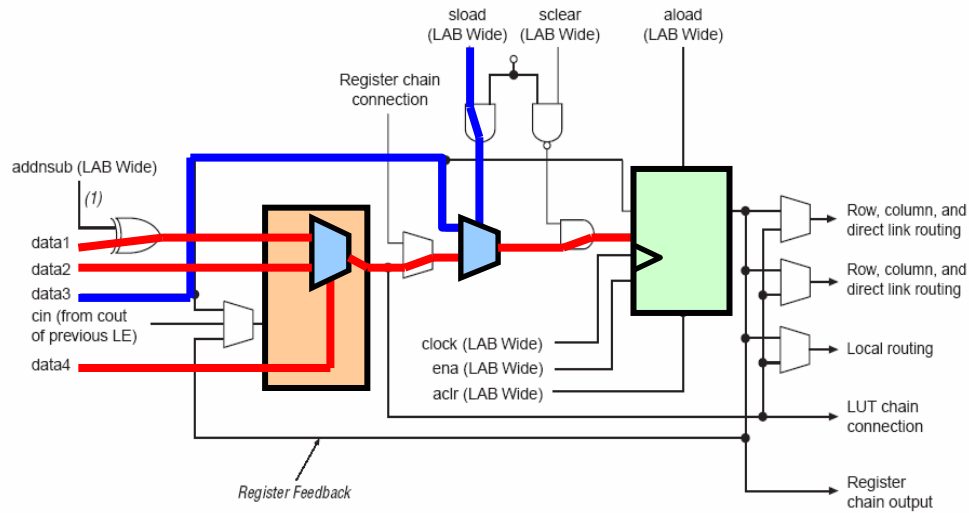
- Take Advantage of Implicit Muxes in Architecture
- Registered Multiplexers Only
 - Register adds extra functionality
- Can Also Implement Control Signals:
 - Synchronous Load
 - Synchronous Reset
 - Clock Enable

The Stratix LE



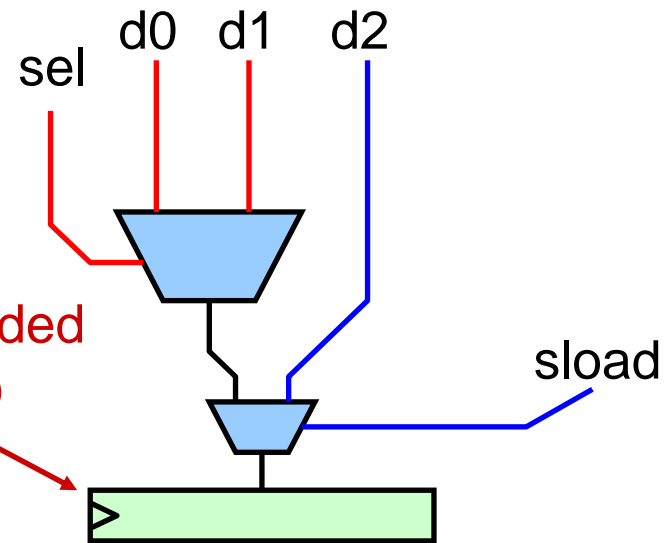
- Lab-Wide Signals (i.e. Only 1 sload, 1 sclear, 2 enables per LAB)

3:1 Mux in 1 LE



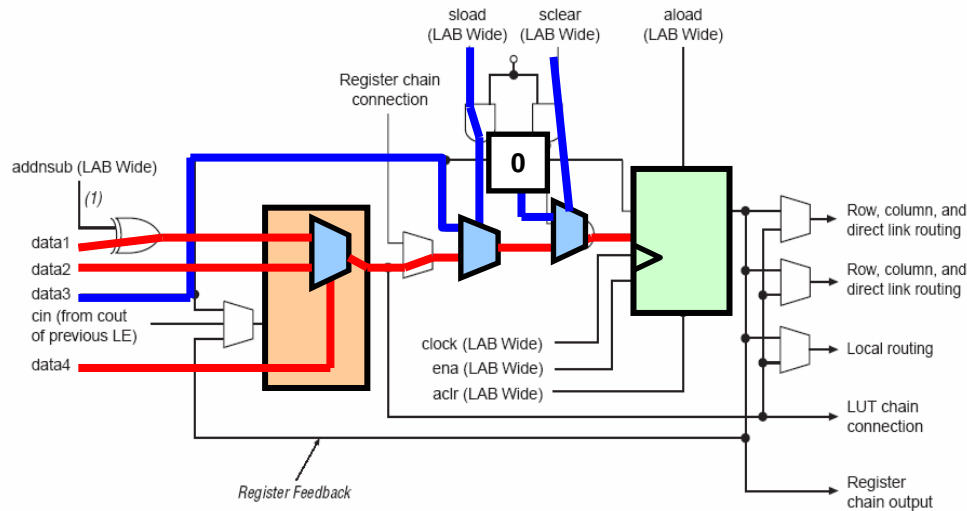
Efficiency:
3:1 Mux / LE

Register Needed
(for sload)



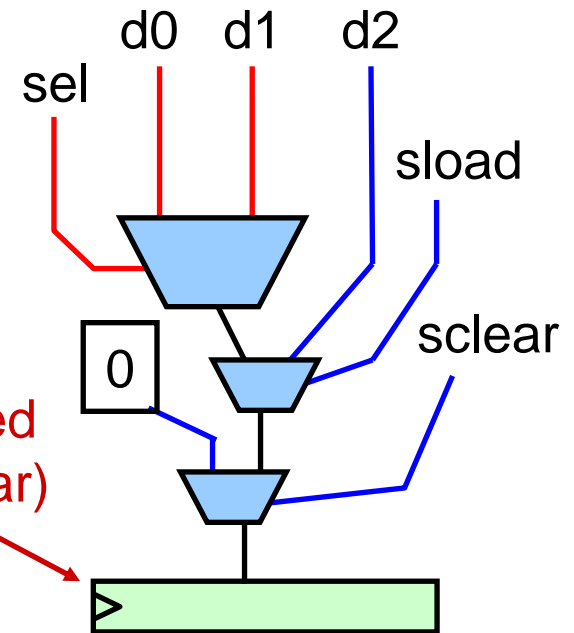
4:1 Mux in 1 LE

- One Input is 0



Efficiency:
4:1 Mux / LE

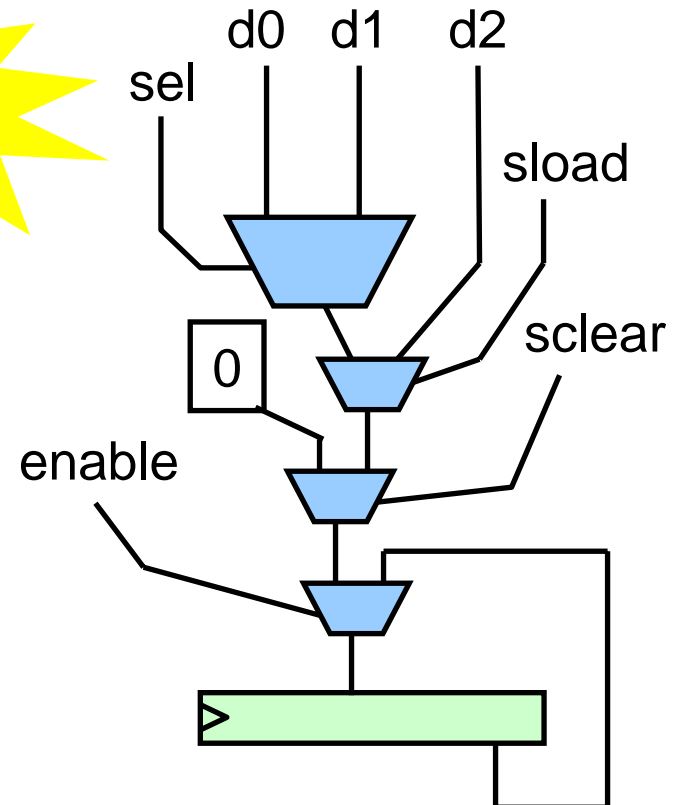
Register Needed
(for sload / sclear)



Clock Enable Is Also a Mux - 5:1 Mux in 1 LE!

New for
QII 4.0

```
PROCESS (clk, reset)
IF reset THEN
    z <= 0;
ELSIF rising_edge(clk) THEN
    IF enable THEN
        IF sclear THEN
            z <= 0;
        ELSIF sload THEN
            z <= d2;
        ELSE
            IF sel THEN
                z <= d1;
            ELSE z <= d0;
            END IF;
        END IF;
    END IF;
END IF;
```



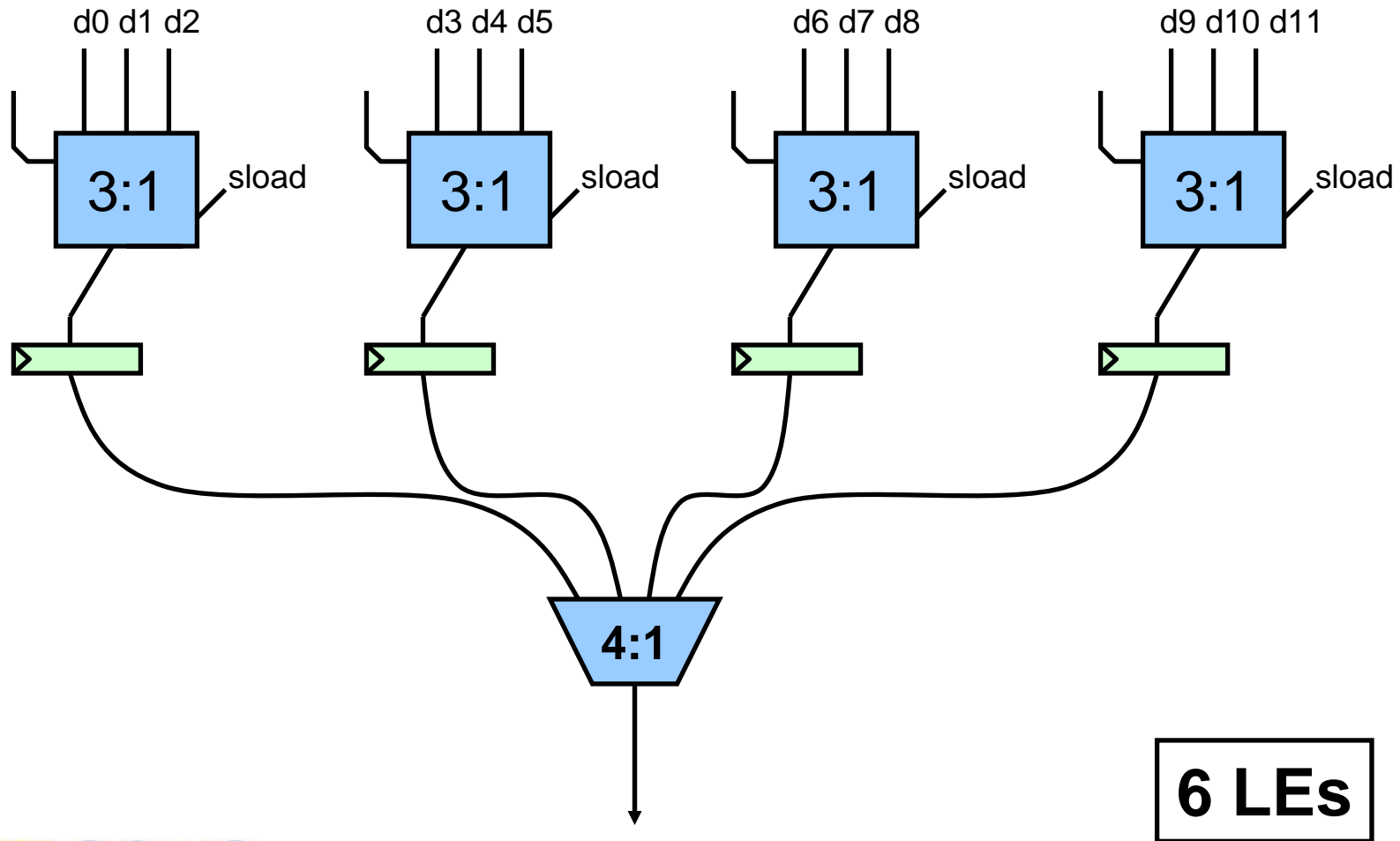
- One Input is 0
- Feed Back Value When Enable = 0



SOPC
WORLD
2004

Taking Advantage of Implicit Multiplexing

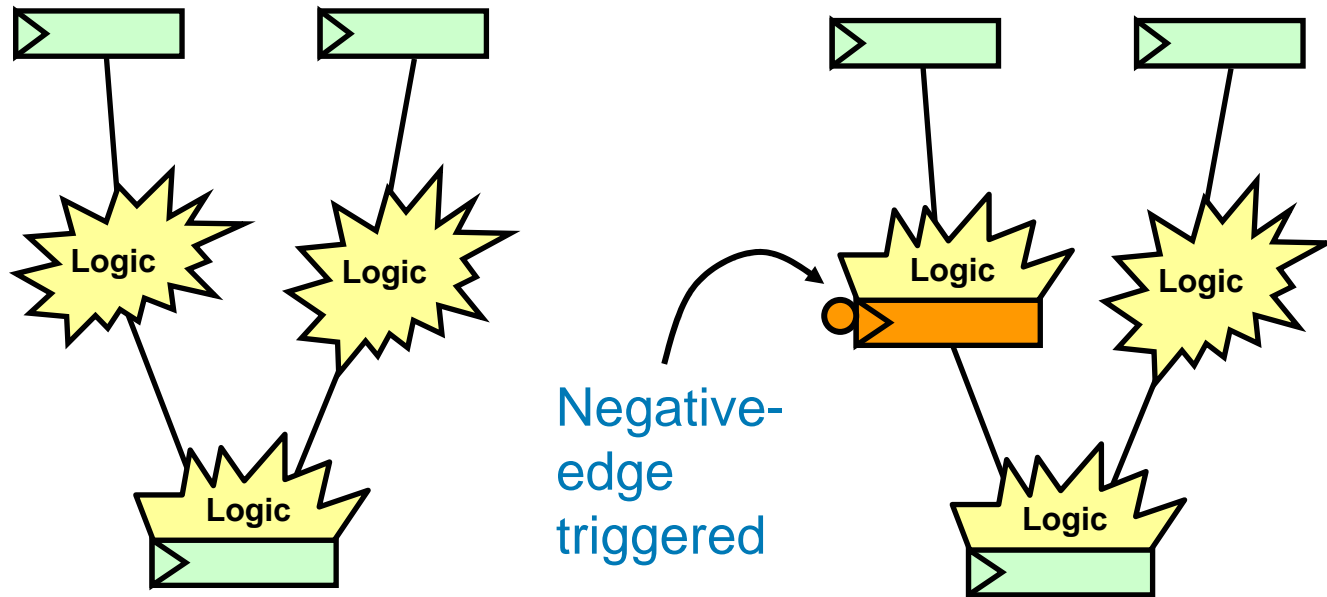
Can Build Large Muxes Using 3:1s



Making the Most of Implicit Muxing

- Implicit Muxing Relies on Register Functionality
- Good to Use When Output of Mux is Registered
- What If Mux Output Is Asynchronous?

Can Add Artificial Registers to a Design



- Registers Come for Free With Logic
- Registers Allow Additional Sload/Sclr Functionality

Summary – Background

- Where do Multiplexers come from?
 - CASE Generally Gives Selector or Binary Muxes
 - IF THEN ELSE Generally Gives Priority Muxes
- How Often Do Multiplexers Occur?
 - 26% of LEs on Average
- How Are N:1 Multiplexers Implemented in Quartus II (in 4-LUT Architectures)?
 - Binary / Selector (0.66 N LEs)
 - Priority (1.00 N LEs)
 - Linear Mux (0.50 N LEs) **NEW!**

Summary – Background

- Implicit Multiplexing
 - Extra functionality with Registers
 - Enable, Sync-Load, Sync-Clear
- Taking Advantage of Implicit Multiplexing
 - General 3:1 Mux in 1 LE!
 - Potential for 5:1 Mux in 1 LE!
 - Can Add Extra Registers to Async Logic in Some Cases

Agenda

Background/Theoretical

Design Guidelines

Quartus II Project to Improve MUX Synthesis

Agenda - Design Guidelines

Common Multiplexer Pitfalls:

- One-Hot Controls
- The ‘Others’
- Degenerate Muxes



SOPC
WORLD
2004

Common Multiplexer Pitfalls: One-Hot Controls

One Hot Encoding: Common Mistake

```
CASE sel IS
WHEN "0001" => z <= a;
WHEN "0010" => z <= b;
WHEN "0100" => z <= c;
WHEN "1000" => z <= d;
WHEN OTHERS => z <= 0;
END CASE;
```




**Synthesis Can't Know that
This Is Intended to be a
One-Hot Select**

**Quartus II Will Build a 16:1
Binary Multiplexer:
8-10 LEs**

One Hot Encoding: Solution

```
Z <= a WHEN sel[0]='1' ELSE "ZZZZ";  
Z <= b WHEN sel[1]='1' ELSE "ZZZZ";  
Z <= c WHEN sel[2]='1' ELSE "ZZZZ";  
Z <= d WHEN sel[3]='1' ELSE "ZZZZ";
```



**Synthesis Assumes
Tri-States are One-Hot**

**Quartus II Will Build a 4
Input Selector Multiplexer:
3 LEs**

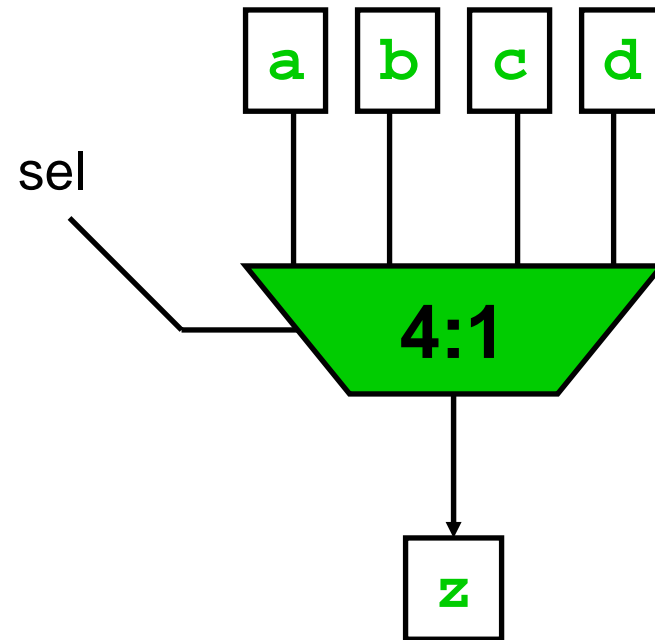


SOPC
WORLD
2004

Common Multiplexer Pitfalls: The “Others”

Efficient 4:1 Multiplexer

```
CASE sel IS  
WHEN "00" => z <= a;  
WHEN "01" => z <= b;  
WHEN "10" => z <= c;  
WHEN "11" => z <= d;  
END CASE;
```



**No OTHERS, or Default, Case.
HDL Rules Say You Should
Always Specify a Default.**

2 LEs

What Should OTHERS Be Set To?

```
CASE sel IS
WHEN "0001" => z <= a;
WHEN "0010" => z <= b;
WHEN "0100" => z <= c;
WHEN "1000" => z <= d;
WHEN OTHERS => z <= ????
END CASE;
```

Quartus II 4.1 Results (Planned)

OTHERS:	# LEs
Z <= NULL	3
Z <= Z	3
Z <= "0000"	3
Z <= "-- --"	2
Z <= d	2
Z <= c	5

Should Be the Correct Way to Make OTHERS Assignment.
Synthesis Tool Should Choose the Optimal Assignment
for "Don't Care" Value; for 4.1.

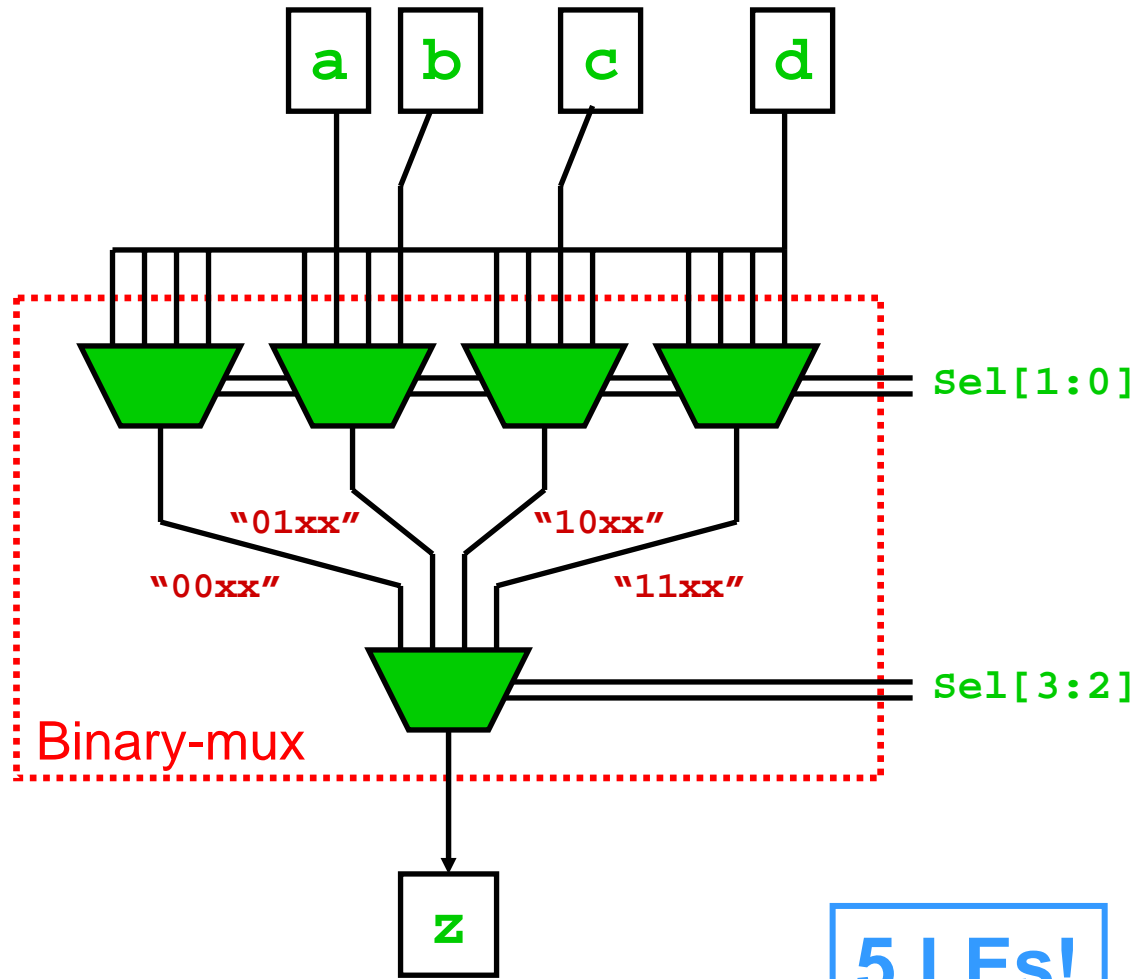


SOPC
WORLD
2004

Common Multiplexer Pitfalls: Degenerate Muxes

Problem: Degenerate 8:1 Mux

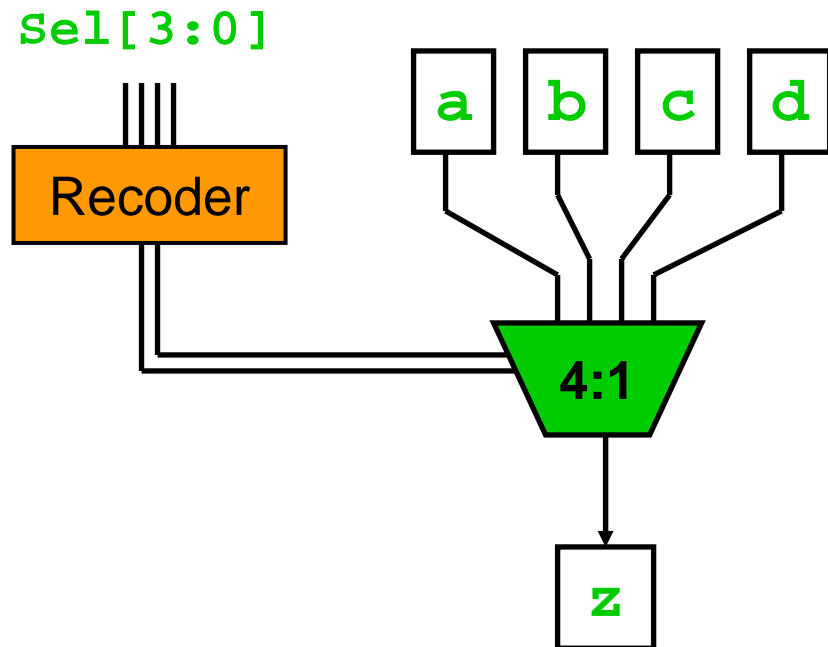
```
CASE sel[3:0] IS
WHEN "0101" =>
    z <= a;
WHEN "0111" =>
    z <= b;
WHEN "1010" =>
    z <= c;
WHEN OTHERS =>
    z <= d;
END CASE;
```



Solution: Recode Degenerate Muxes

Recoder

```
CASE sel[3:0] IS
WHEN "0101" =>
    z_sel <= "00";
WHEN "0111" =>
    z_sel <= "01";
WHEN "1010" =>
    z_sel <= "10";
WHEN OTHERS =>
    z_sel <= "11";
END CASE;
```



Recoder (Can Be
Shared): 2 LEs

2 LEs

Synthesis Does Not Extract Defaults Well

■ Possible Solutions:

- Flatten Multiplexer (Use One CASE Statement!)
- RecodE (Use 4:1 CASE Statement Method)
- Restructure Code So to Reduce Default Cases
- Question Whether Defaults Are Don't Cares
 - Promote Last ELSIF to ELSE If No Other Cases Will Happen

Summary - Design Guidelines

- Encourage Muxes to be Fully Populated, Binary Controlled
 - May Require Recoding of Control Lines
 - Avoid Degenerate Muxes
- Be Careful of Special Cases
 - One Hot Controls
 - Implicit Defaults

Agenda

Background/Theoretical

Design Guidelines

Quartus II Project to Improve MUX Synthesis



SOPC
WORLD
2004

Quartus II MUX Synthesis Improvements

Quartus II Synthesis Project

- Performs Recoding for Bus of Muxes Automatically
 - Searches for Buses of Mux Trees
 - Estimates Cost of Recoding
 - Recodes If More Efficient (in Terms of Area)
 - Exploits Duplicates & Constants In Mux Inputs
 - Uses Most Efficient Multiplexer Implementation
- Hidden INI and QSF Variable in 4.0, Feature Release Planned for 4.1
 - For Stratix only in 4.0

Enabling MUX Optimization in 4.0

- INI Variable:

`mast_extract_and_optimise_bus_muxes=on`

- QSF Variable:

`EXTRACT_AND_OPTIMIZE_BUS_MUXES`

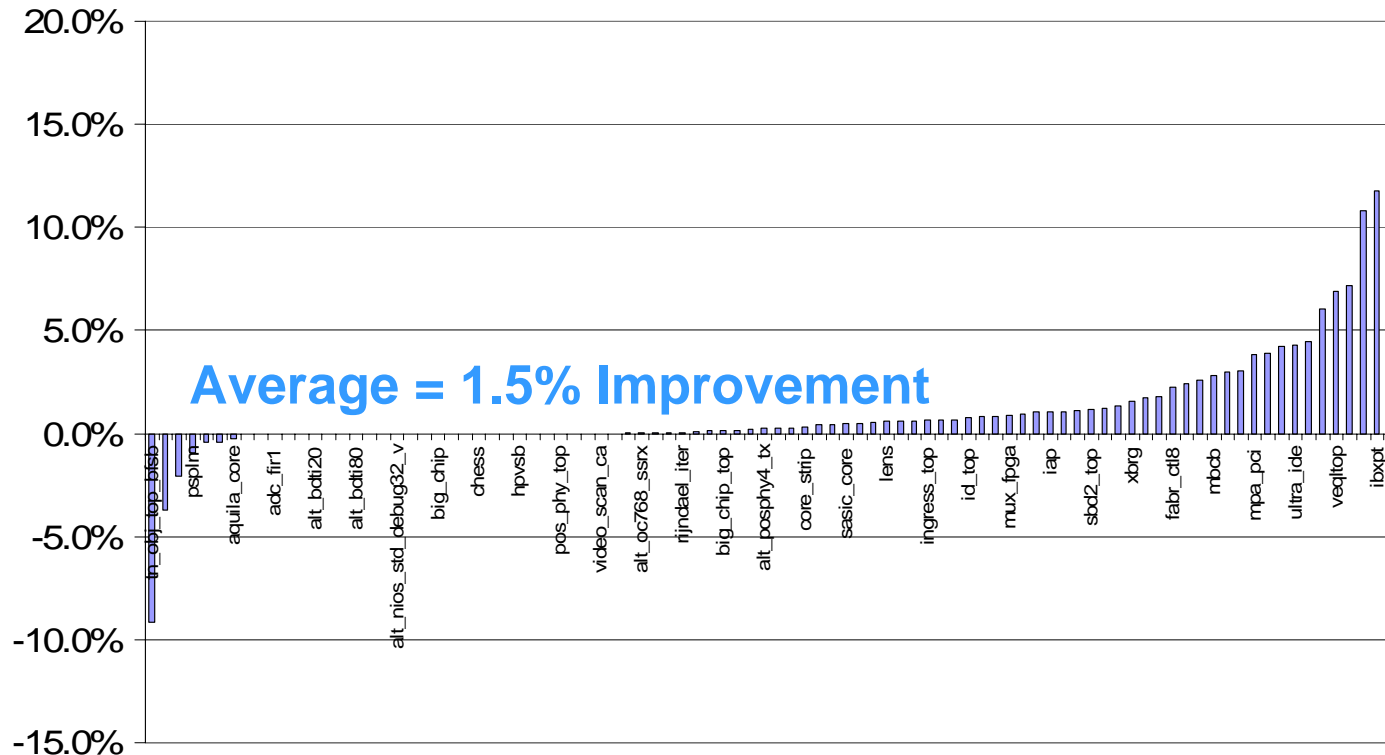
- Example Tcl/QPF Entry:

- Enables Option for All Nodes in Entity `mux_bus_alpha` of Type `mux_bus` Instantiated in `test_controller`, the Top-Level Entity:

```
set_instance_assignment -name  
EXTRACT_AND_OPTIMIZE_BUS_MUXES ON -to  
"mux_bus:mux_bus_alpha" -entity  
test_controller
```

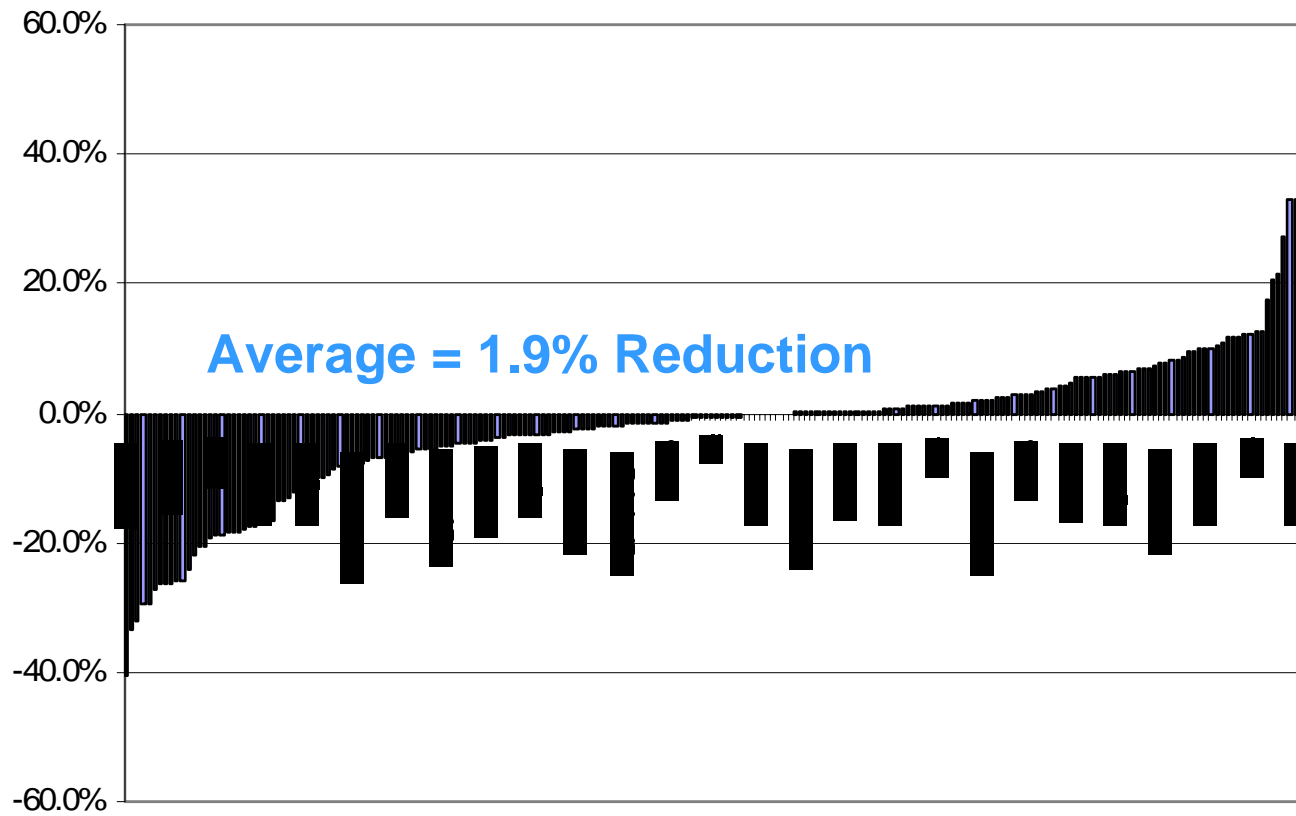
Results So Far... (Quartus II 4.0)

Percentage Improvement in Area



Results So Far... (Quartus II 4.0)

F_{max} Effect



General Summary

- Where Multiplexers Come From (CASE, IF)
- How Often Muxes Occur: Average 26% of LEs
- How N:1 Multiplexers Are Implemented in Quartus II (in 4-LUT Architectures)
- Guidelines: Encourage Muxes to be Fully Populated, Binary Controlled (Recode if Needed)
- Buses of Multiplexers Can Have Big Effect
- Try the INI for Improved MUX Optimization in Quartus II

POP Quiz

■ Where can we set enabling MUX OPTIMIZATION in QuartusII4.1?

1. Assignment>>Settings>>Analysis&Synthesis settings>>Restructure Multiplexers
2. Assignment>>Assignment Editor>>Logic options>> Restructure Multiplexers
3. Above all