# Resource Optimization Techniques for LUT-Based Devices

*Nov, 2008*

# Agenda

- Objectives
- Resource Optimization Techniques
  - Quartus II setting options
  - Design guidelines on some typical modules
- Summary

# Objectives

- Summarize all the setting options provided in Quartus II helpful to reduce resource utilization of LUT-based devices.
- Deep study the implementation mechanism of some settings which is visible to Quartus II users.
- Provide resource reduction design guidelines for some frequently-used design constructs.

# Resource Overuse Types

- I/O pin overuse (not discussed here)
- Logic resource overuse
  - LE/LAB in Cyclone series & MAX II family
  - ALUT/ALM in Stratix series & Arria GX family
  - Dedicated resources such as memory and DSP blocks (not discussed here)
- Routing resource overuse

# Quartus II Synthesis Settings for Logic Utilization Optimization

- Set “Synthesis Optimization Technique” as “balanced” or “Area”.
- Logic resources <-> Dedicated resources balancing
  - Auto DSP Block/RAM/ROM/Shift Register Replacement
  - Allow Any RAM/ROM/Shift Register Size For Recognition
- Remove redundant logic
  - Power-Up Don’t Care / Remove Duplication Registers / Remove Redundant Logic Cells / Auto Resource Sharing
- For multiplexer
  - Restructure Multiplexers
- For state machine
  - Extract Verilog/VHDL State Machines
  - Safe State Machine
  - State Machine Processing

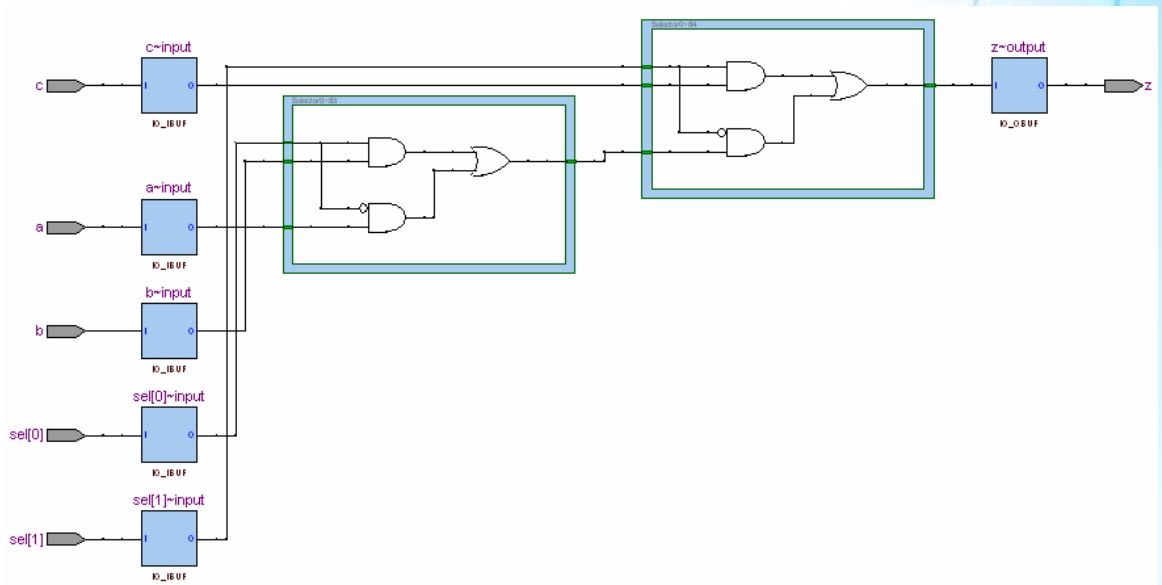
# Multiplexer

- Created from **case** statement, **if** statement or state machine in HDL code.
- Forms a large portion of logic utilization in many cases.
- Examples
  - **Case** statement
  - **If** statement

# Multiplexer

- Created from **case** statement, **if** statement or state machine in HDL code.
- Forms a large portion of logic utilization in many cases.
- Examples
  - **Case** statement
  - **If** statement

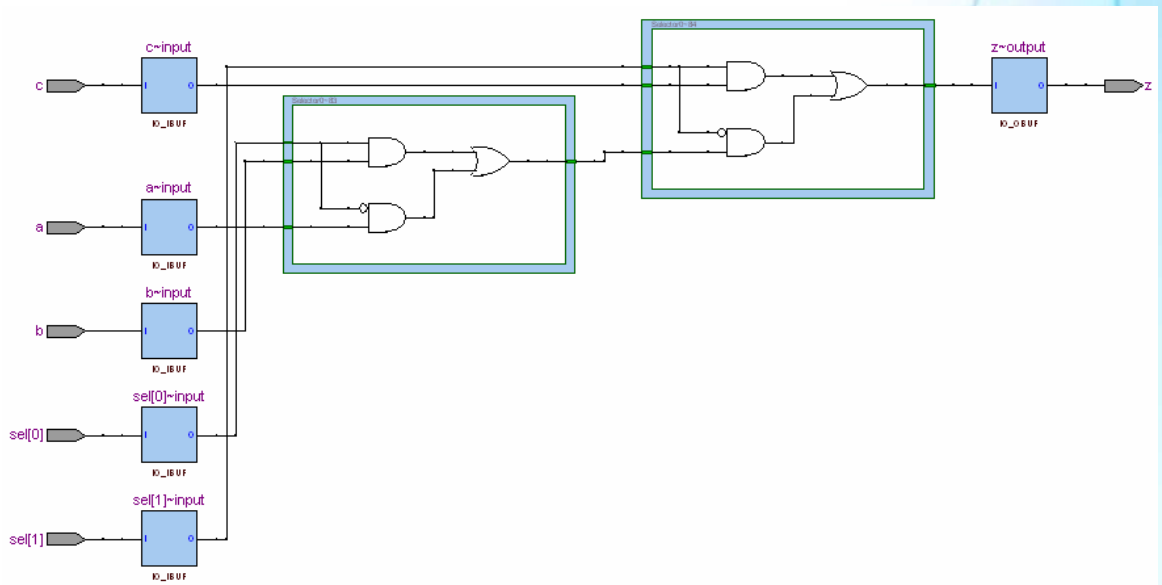
```
case (sel[1:0])
    2'b00: z = a;
    2'b01: z = b;
    default: z = c;
endcase
```



# Multiplexer

- Created from **case** statement, **if** statement or state machine in HDL code.
- Forms a large portion of logic utilization in many cases.
- Examples
  - **Case** statement
  - **If** statement

*if (sel[1:0] == 2'b00) z = a;*
*else if (sel[1:0] == 2'b01) z = b;*
*else z = c;*



# Multiplexer Design Considerations for Resource Optimization

- Handling methods of the default case items
- Synthesis option “Restructure Multiplexers”

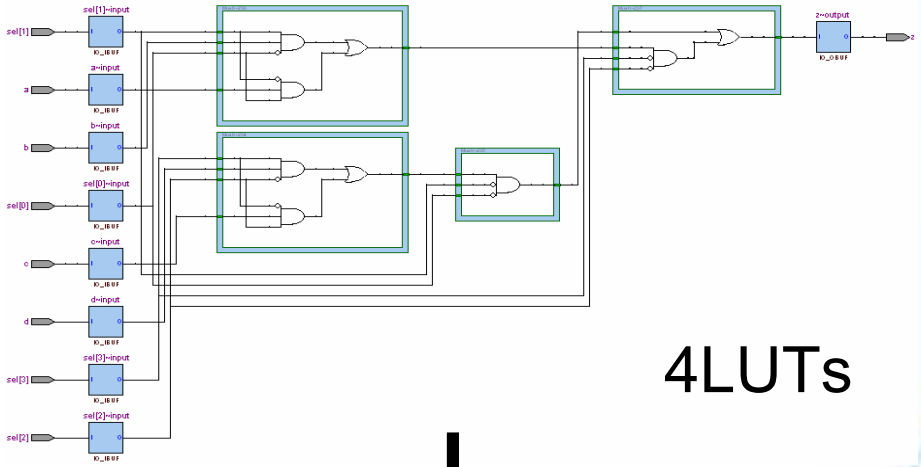
# Default Case Assignment

- To avoid latch, use default case assignment for cases not specified.
  - For **case** statement, use **default** (Verilog) or **Others** (VHDL);
  - For **if** statement, use **else**.
- To reduce area, assign **x** (don't care) value for the default case if possible.
  - In QII **x** value only works in case statement.
  - Suggest QII being improved for if statement for better synthesis results.
- If the default value is cared, consider improving possible degenerate multiplexer in your design.
  - More helpful for bus of multiplexers.

# Assign “x” Default Value

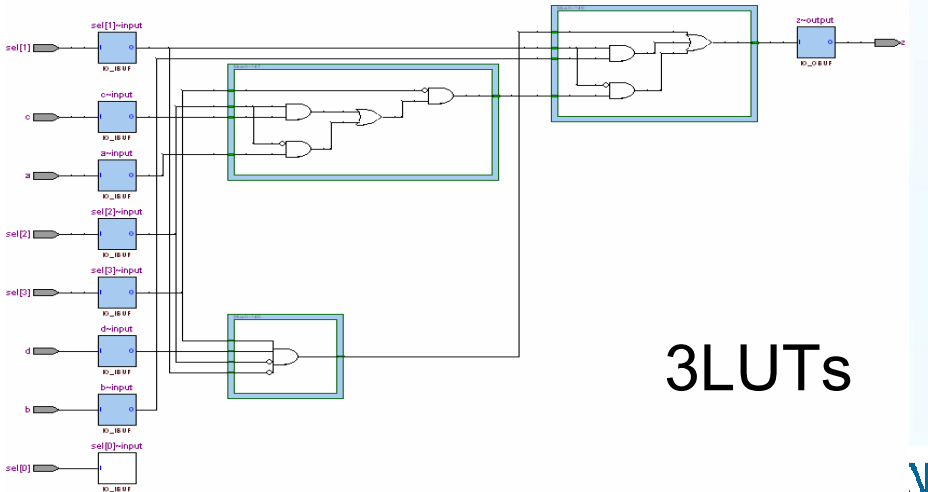
- Helpful for case statement

```
case (sel[3:0])
    4'b0001: z = a;
    4'b0010: z = b;
    4'b0100: z = c;
    4'b1000: z = d;
    default: z = 1'b0;
endcase
```



4LUTs

```
case (sel[3:0])
    4'b0001: z = a;
    4'b0010: z = b;
    4'b0100: z = c;
    4'b1000: z = d;
    default: z = 1'bx;
endcase
```



3LUTs

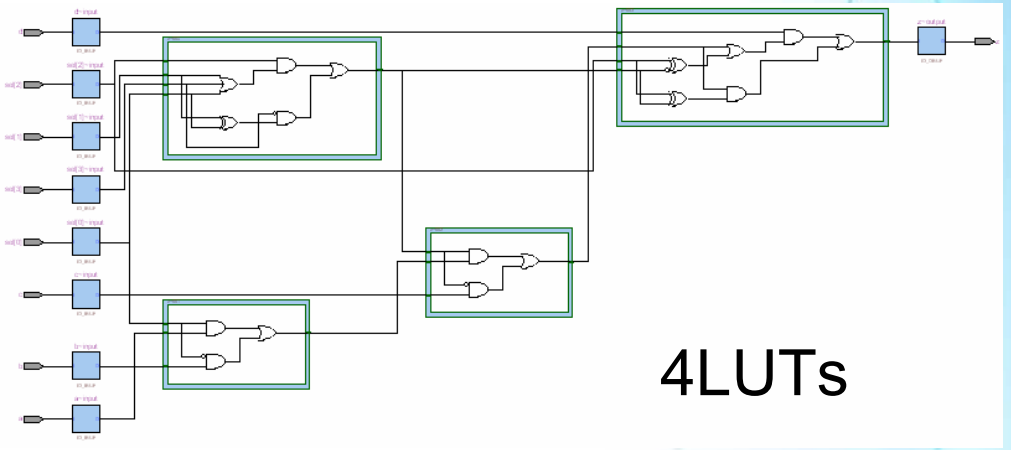
# Assign “x” Default Value (Cont.)

- Not helpful for if statement

```
if (sel[3:0] == 4'b0001) z = a;
else if (sel[3:0] == 4'b0010) z = b;
else if (sel[3:0] == 4'b0100) z = c;
else if (sel[3:0] == 4'b1000) z = d;
else z = 1'bx;
```

is treated as

```
if (sel[3:0] == 4'b0001) z = a;
else if (sel[3:0] == 4'b0010) z = b;
else if (sel[3:0] == 4'b0100) z = c;
else z = d;
```




- QII Improvement suggested
  - Synplify treats “1’bx” as arbitrary value.

# Improvement on Degenerate Multiplexer

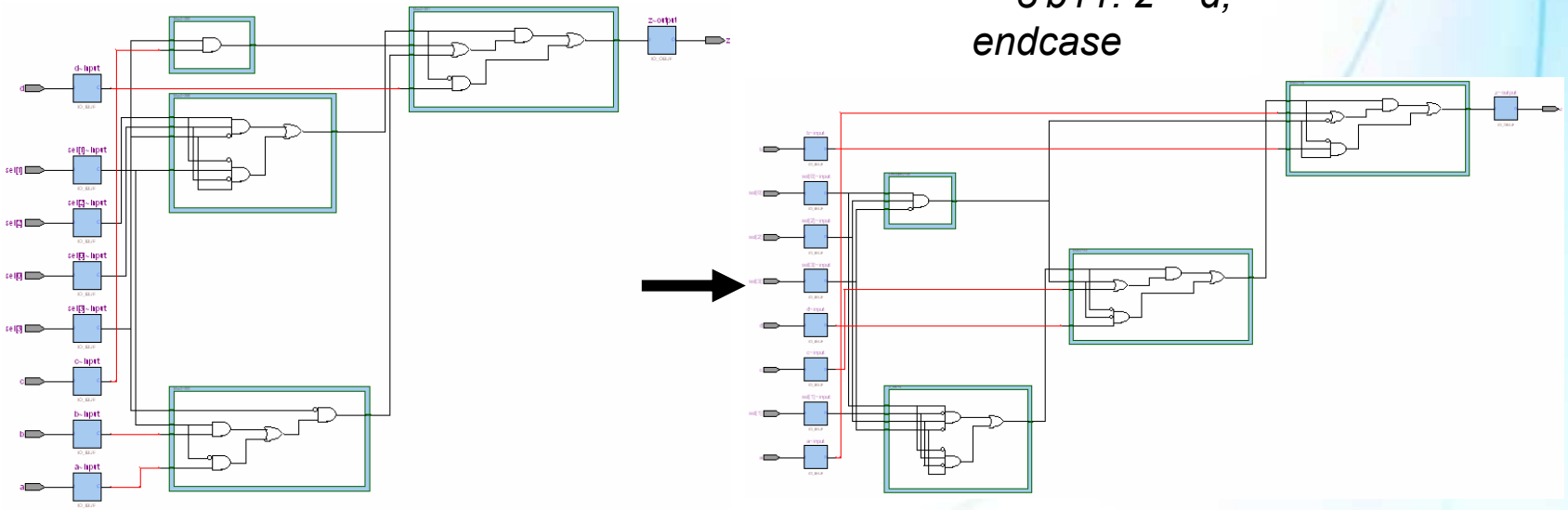
- Not all of the cases are used for unique data inputs.
- Example
  - One-bit multiplexer
  - Bus of multiplexers

# Improvement on Degenerate Multiplexer (Cont.)

- Example
  - One-bit multiplexer
  - Bus of multiplexers

```
case (sel[3:0])
    4'b0101: z = a;
    4'b0111: z = b;
    4'b1010: z = c;
    default: z = d;
endcase
```

```
case (sel[3:0])
    4'b0101: z_sel = 2'b00;
    4'b0111: z_sel = 2'b01;
    4'b1010: z_sel = 2'b10;
    default: z_sel = 2'b11;
endcase
case (z_sel[1:0])
    3'b00: z = a;
    3'b01: z = b;
    3'b10: z = c;
    3'b11: z = d;
endcase
```



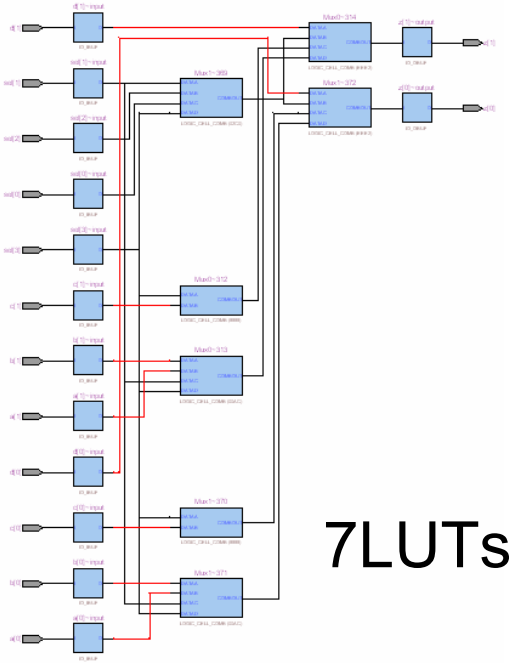
# Improvement on Degenerate Multiplexer (Cont.)

- Example
  - One-bit multiplexer
  - Bus of multiplexers

```
case (sel[3:0])
    4'b0101: z = a[1:0];
    4'b0111: z = b[1:0];
    4'b1010: z = c[1:0];
    default: z = d[1:0];
endcase
```

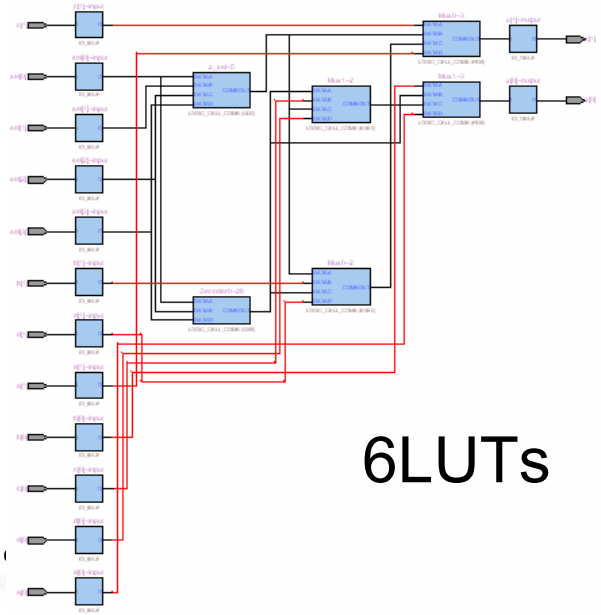
→

```
case (sel[3:0])
    4'b0101: z_sel = 2'b00;
    4'b0111: z_sel = 2'b01;
    4'b1010: z_sel = 2'b10;
    default: z_sel = 2'b11;
endcase
case (z_sel[1:0])
    3'b00: z = a[1:0];
    3'b01: z = b[1:0];
    3'b10: z = c[1:0];
    3'b11: z = d[1:0];
endcase
```



7LUTs

→



6LUTs

# “Restructure Multiplexers” Logic Option

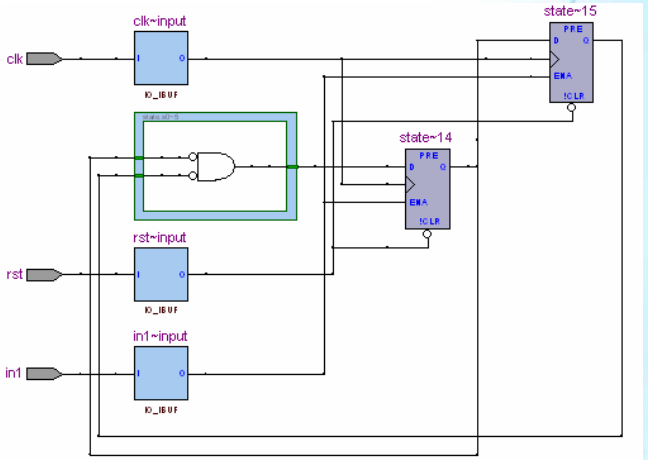
- Optimize resource usage only for buses of Multiplexers.
- Performs some recoding work without changing HDL codes.
  - Such as the improvement on degenerate multiplexer.
- The default setting is “Auto”.
  - Enabled when the **Optimization Technique** option is set to **Area** or **Balanced**;
  - Disabled when the **Optimization Technique** option is **Speed**.

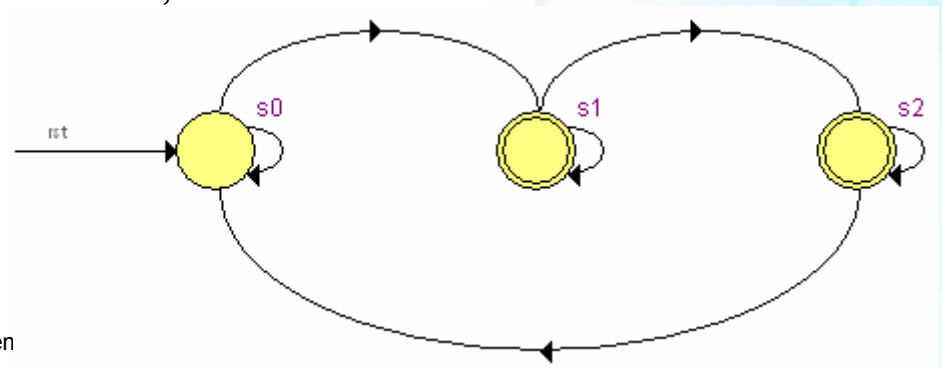
# State Machine

- Refer to "*State Machine General Coding Guidelines*" in QII HB for designing proper state machines.

```
parameter s0 = 2'b00;
parameter s1 = 2'b01;
parameter s2 = 2'b10;

always @(posedge clk or posedge rst)
   if (rst) state <= s0; else state <= next_state;

always @(state or in1)
   case (state)
      s0: if (in1) next_state <= s1; else next_state <= s0;
      s1: if (in1) next_state <= s2; else next_state <= s1;
      s2: if (in1) next_state <= s0; else next_state <= s2;
      default: next_state <= s0;
   endcase
```

***State Machine Processing** as **User-Encoded***



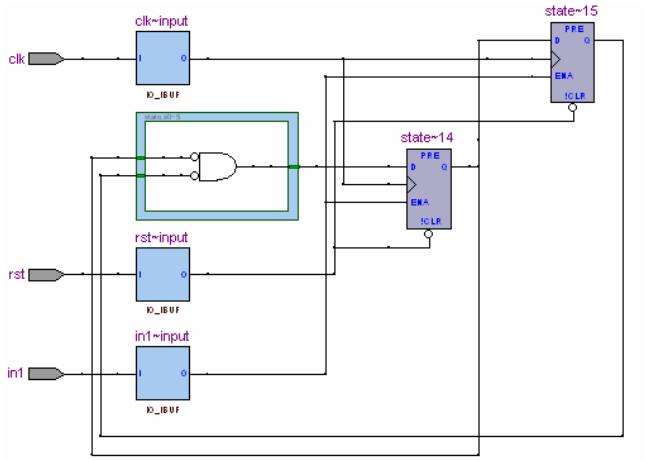
# State Machine Design Considerations for Resource Optimization

- Three synthesis options
  - Extract Verilog/VHDL State Machines
  - Safe State Machine
  - State Machine Processing

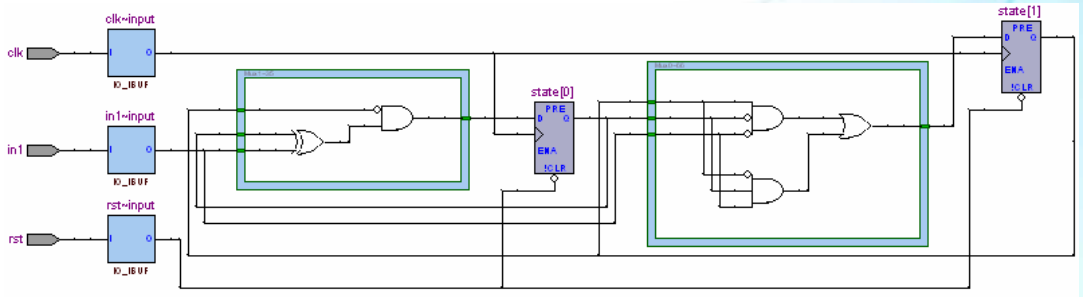
# Extract Verilog/VHDL State Machines

- Allows QII to extract state machines from HDL Design Files.
  - On by default.
  - If set as Off, QII treats state machine modules as regular logic.
  - Take the state machine on slide 17 as an example.

**On**




**Off**



# Safe State Machine

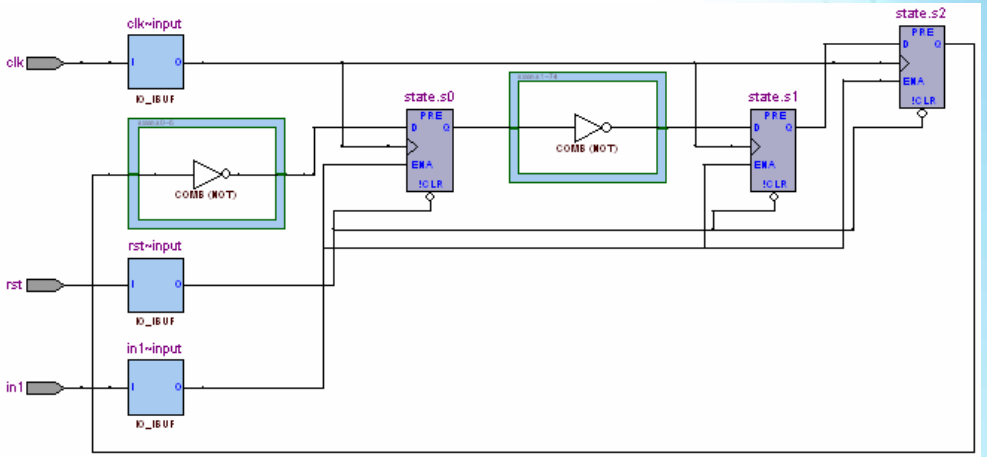
- Directs QII to implement state machines that can recover from an illegal state to the reset state.
- Add extra logic to detect and handle the illegal state.
- Off by default
- Causes of the illegal state
  - State machines have input(s) from another clock domain.
    - This option is not required generally.
    - Synchronizing the async input(s) can avoid the illegal state.
  - Device works in a hostile operating environment such as space applications.
    - This option is required.

# Safe State Machine (Cont.)

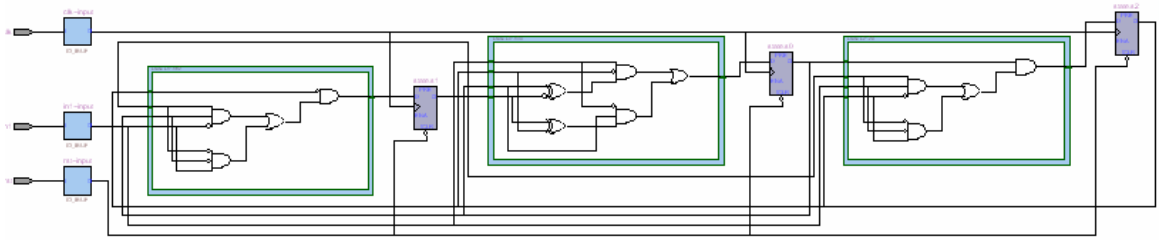
- Example

```
parameter s0 = 2'b00;
parameter s1 = 2'b01;
parameter s2 = 2'b10;

always @(posedge clk or posedge rst)
   if (rst) state <= s0; else state <= next_state;

always @(state or in1)
   case (state)
      s0: if (in1) next_state <= s1; else next_state <= s0;
      s1: if (in1) next_state <= s2; else next_state <= s1;
      s2: if (in1) next_state <= s0; else next_state <= s2;
      default: next_state <= s0;
   endcase
```

**Off**




**On**



***State Machine Processing*** *as* ***Auto***

# State Machine Processing

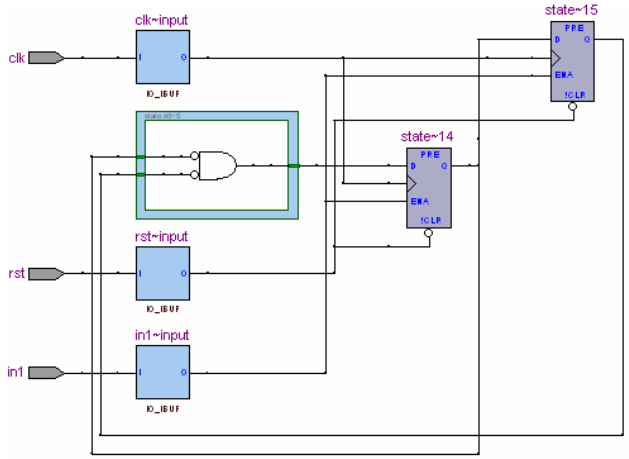
- Directs QII to implement state machines with the specified encoding style.
- Options
  - Auto (Default), User-Encoded, Minimal Bits, One-Hot, Gray, etc.
- Consider changing the state machine encoding styles to reduce resource usage of a design with state machines.
  - Select “Minimal Bits” if timing is not a concern.
  - Try “User-Encoded” to see if a better result can be got.

…

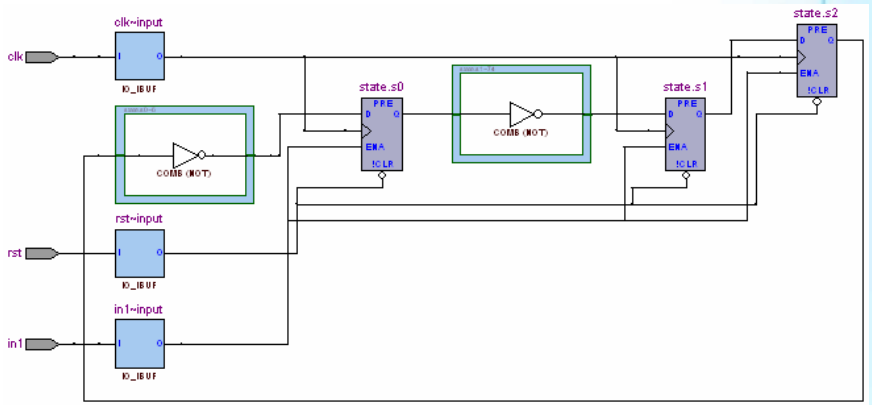
# State Machine Processing (Cont.)

- Example

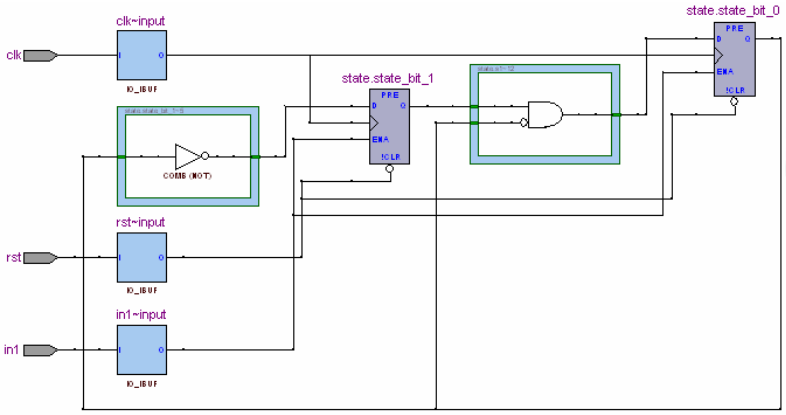
***State Machine Processing** as **User-Encoded***




***State Machine Processing** as **Auto***




***State Machine Processing** as **Gray***



# Quartus II Fitting Settings for Logic Utilization Optimization

- Auto Packed Registers
- Physical synthesis for fitting
  - Perform physical synthesis for combinational logic
    - detects and removes duplicate combinational nodes, etc.
  - Perform logic to memory mapping
    - More efficiently, make sure the source codes can implement RAM/ROM/FIFO/Shift register functions with memory resources instead of logic cells.

# Auto Packed Registers

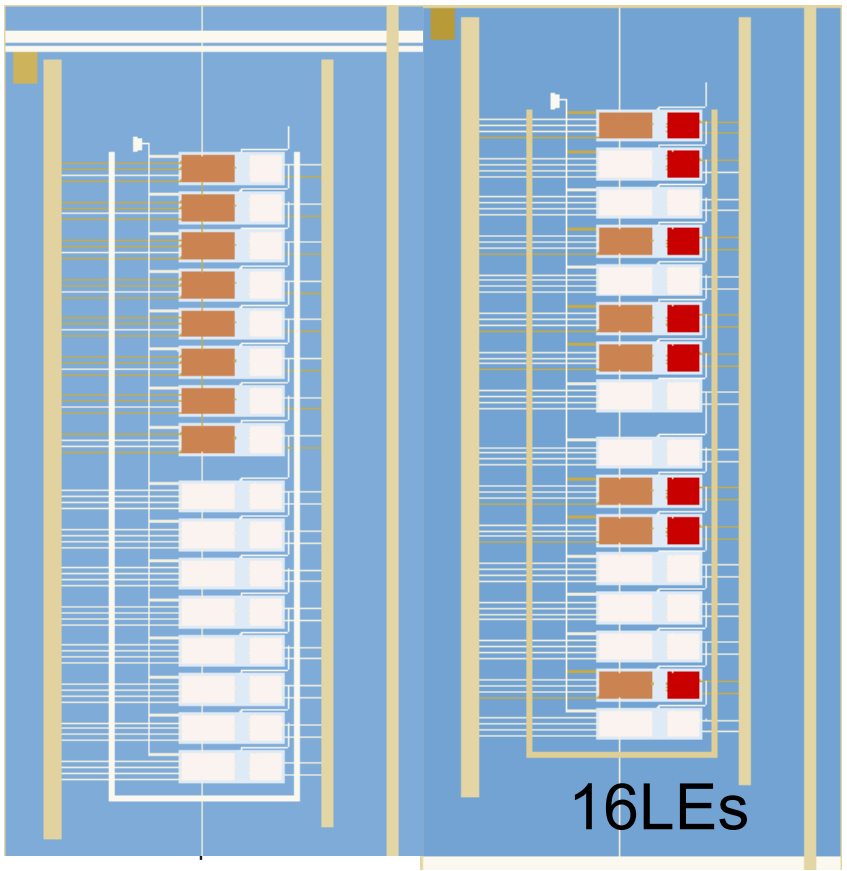
- Merges two logic cells into one by combining the register of one cell in which only the register is used with the LUT of another cell in which only the LUT is used.
- Options
  - Off, Normal, Minimize Area, Minimize Area with Chains, Auto (Default), Sparse, Sparse Auto.
- Example

```
assign c = a[7:0] + b[7:0];

always @(posedge clk or negedge rst)
   if (!rst) e <= 8'b0;
   else begin
      e[0] <= d; e[1] <= e[0]; e[2] <= e[1]; e[3] <= e[2];
      e[4] <= e[3]; e[5] <= e[4]; e[6] <= e[5]; e[7] <= e[6];
   end
```

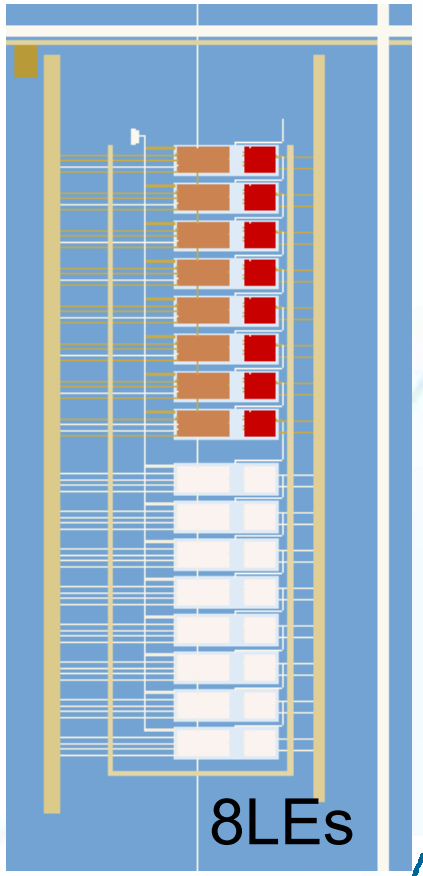
# Auto Packed Registers (Cont.)

- Example (Cont.)

***Auto Packed Registers** as **Off/Normal/Sparse/Sparse Auto/Auto***




***Auto Packed Registers** as **Minimize Area/ Minimize Area with Chains***

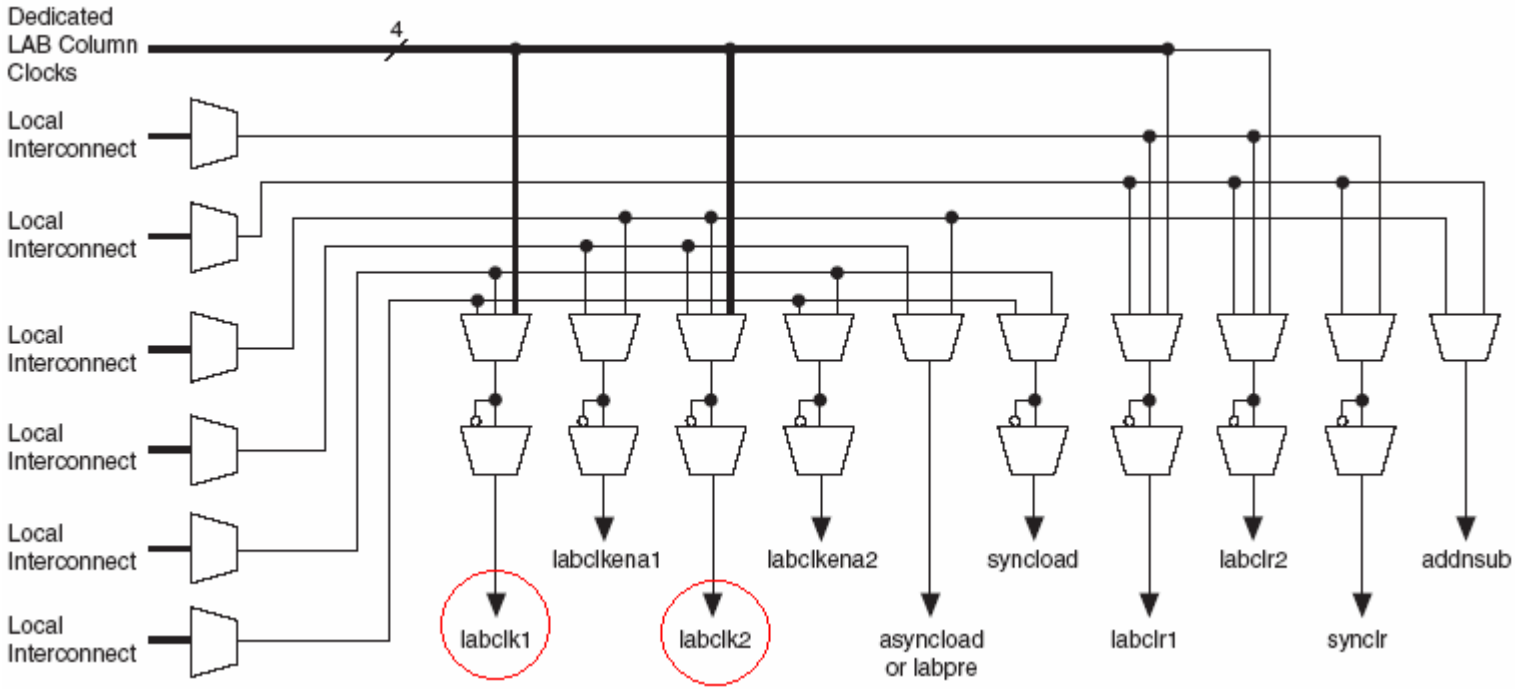


# Quartus II Settings for Routing Optimization

- Set “Synthesis Optimization Technique” as “balanced” or “Area”.
- Fitter Aggressive Routability Optimizations
- Final Placement Optimizations
- Increase Placement Effort Multiplier
- Increase Router Effort Multiplier
- Logic Duplication
  - Auto Register Duplication / Logic Cell Insertion – Logic Duplication

# A Typical Example for Routing Resource Reduction

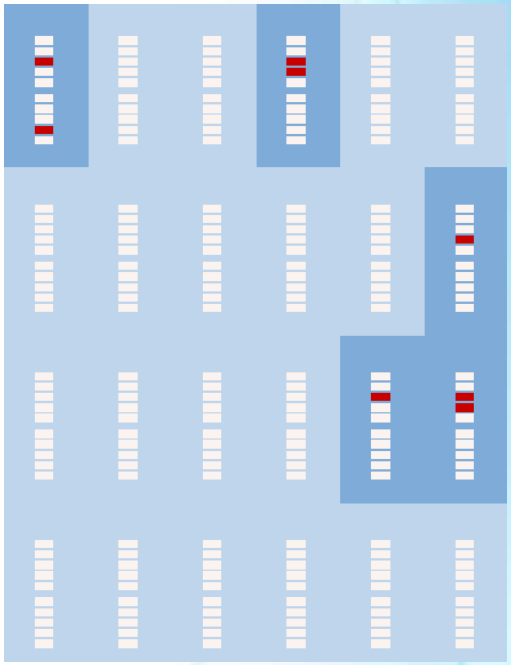
- Background
  - Each MAX II LAB has two LAB-wide clock inputs.

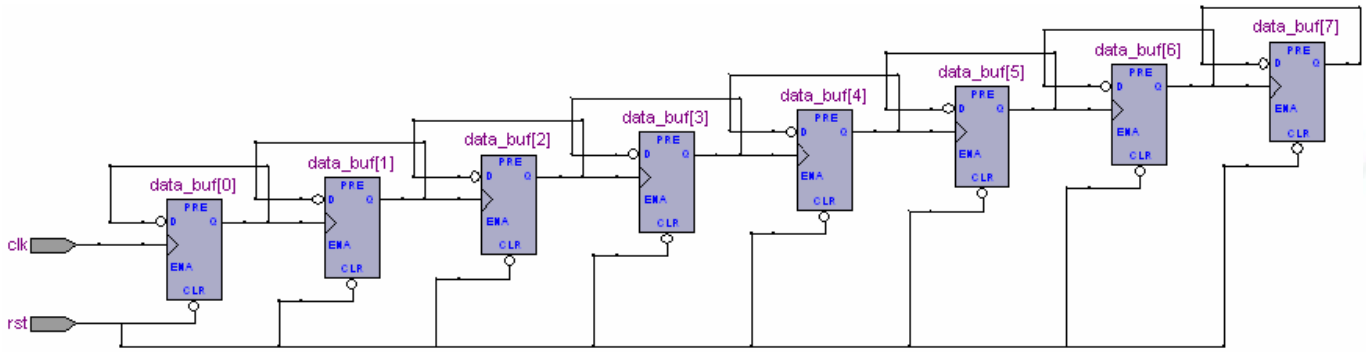


# A Typical Example for Routing Resource Reduction (Cont.)

- Frequency divider – Original design

```
generate for (i = 0; i <= 7; i = i + 1)  begin: divider
   always @ (posedge data_buf[i] or negedge rst)
      if(!rst)
         data_buf[i+1] <= 1'b0;
      else
         data_buf[i+1] <= ~data_buf[i+1];
end
endgenerate
```

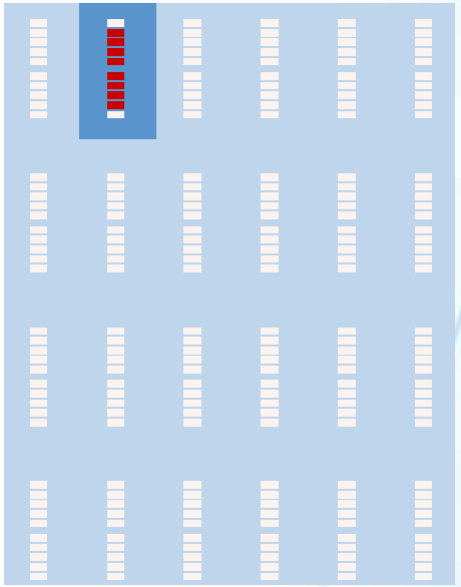


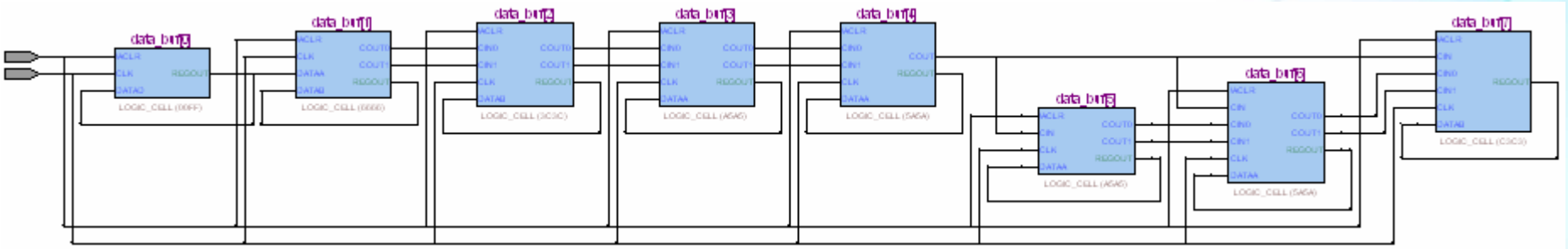


# A Typical Example for Routing Resource Reduction (Cont.)

- Frequency divider – Improved design

```
always @ (posedge clk or negedge rst)
    if(!rst)
        data_buf[7:0] <= 1'b0;
    else
        data_buf[7:0] <= data_buf[7:0] + 1'b1;
```





# Summary

- Quartus II provides many setting options in the synthesis and fitting stages helpful to reduce resource usage.
- If there is no-fit issue Quartus II software cannot fix, consider making design-specific modifications to the source codes.