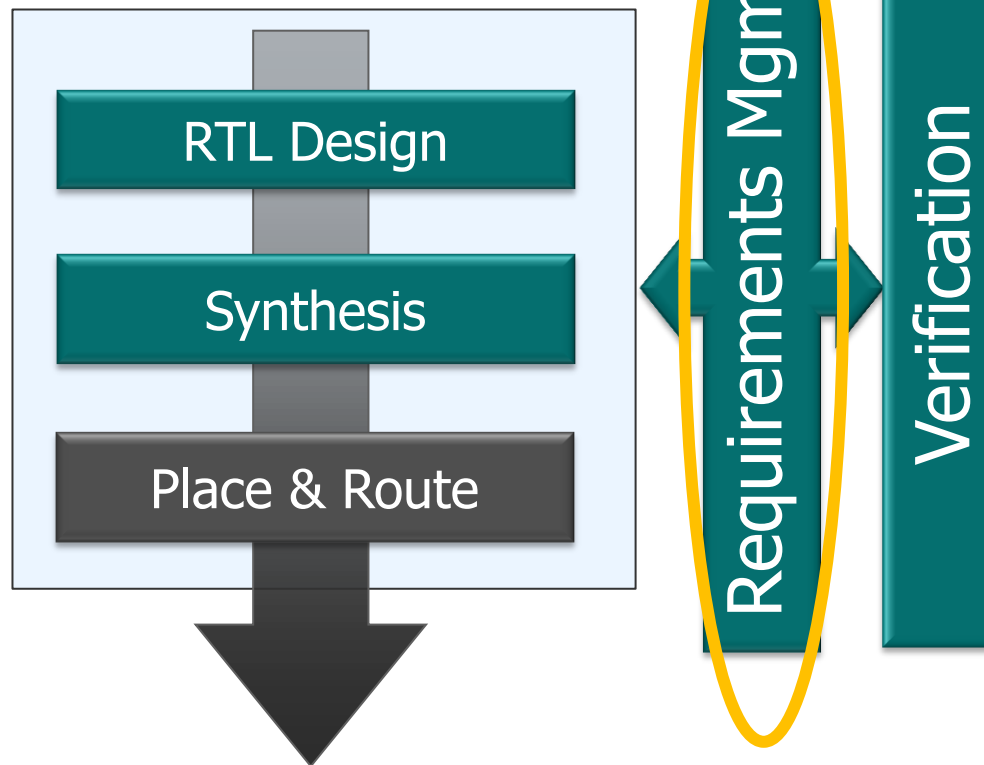


Functional Design

Functional

RTL → Implementation



Safety, Mission, & Security Critical Design: Impact of Product Failure

- Safety – Lives Placed at Risk
 - Fatalities
 - Injuries
 - Jeopardized security
- Business – Negative Financial Impact
 - Missed market window, Late to market
 - Failure in the field
 - Product recall
 - Mission fails
 - Damaged reputation



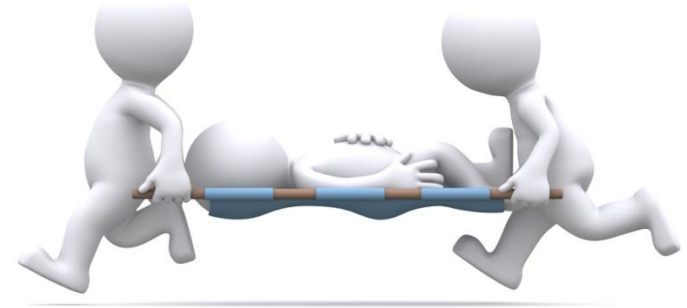
How to ensure that these devices perform as expected & under any foreseeable condition?



Designing Medical Electronics: Can You Afford Product Failure?

■ Safety – Lives Placed at Risk

- Fatalities
- Injuries
- Misdiagnosis



■ Business – Negative Financial Impact

- Missed market window
- Late to market
- Failure in the field
- Product/device recall
- Damaged reputation



General Design Considerations:

Can You Afford Product or Project Failure?

If You Ignored Good Design Practice...You Could Face

■ Negative Financial / Business Impact

- Missed market window, late to market
- Failure in the field
- Product recall
- Mission fails
- Damaged reputation



■ Lives Placed at Risk / Safety Impact


- Fatalities
- Injuries
- Jeopardized security



Need Regulatory Compliance?

■ Following a Prescribed Flow to Demonstrate that a Device Performs its Intended Function

- Managing & tracing requirements
- Appropriate verification at every step of flow
- Documenting the design standards
- Showing compliance to the standards
- Having sufficient monitoring of data & process
- Documenting everything!



Proactively
building a
quality focus
into the process

Establishing a Requirements-Driven Design Flow
is Key to “Design Assurance”

Safety, Mission, & Security Critical Design: Regulatory Standards

- DO-254 for Commercial Aircraft Electronic Hardware

"The equipment, systems...must be designed to ensure that they perform their intended functions under any foreseeable operating condition."

- ISO-26262 for Automotive Electronic Hardware
- SAE ARP 4754a, ARP 4761, EWIS & Other Similar Standards for Aircraft & Automotive Components

Designing Medical Electronics: Regulatory Standards

- Ambiguous Design Control Regulations
 - Current regulations are on SW & Systems
 - No regulations on HW
- So...Leverage Design Assurance Process from Other Safety-critical Industries
 - Avionics: DO-254
 - Automotive: ISO-26262
- Design Assurance Process Is
 - Controlled
 - Auditable
 - Specific to requirements of HW engineering
 - Repeatable



Complex Devices Must Meet Their Requirements &
Perform Well Under All Foreseeable Conditions

General Design Considerations: Regulatory Compliance?

- Probably not mandated
- However, follow a Prescribed Flow to Demonstrate that a Device Performs its Intended Function
 - Proactively build a quality focus into the process



How Do You Know When You are Finished Verifying?

It's an Age Old Question

- *How do I know that I've really finished verifying my design before I commit it to silicon?*

And the Answer is

- *When I have proven that all of my chip requirements are implemented & successfully pass verification!*



Requires a New Design Approach That Will...

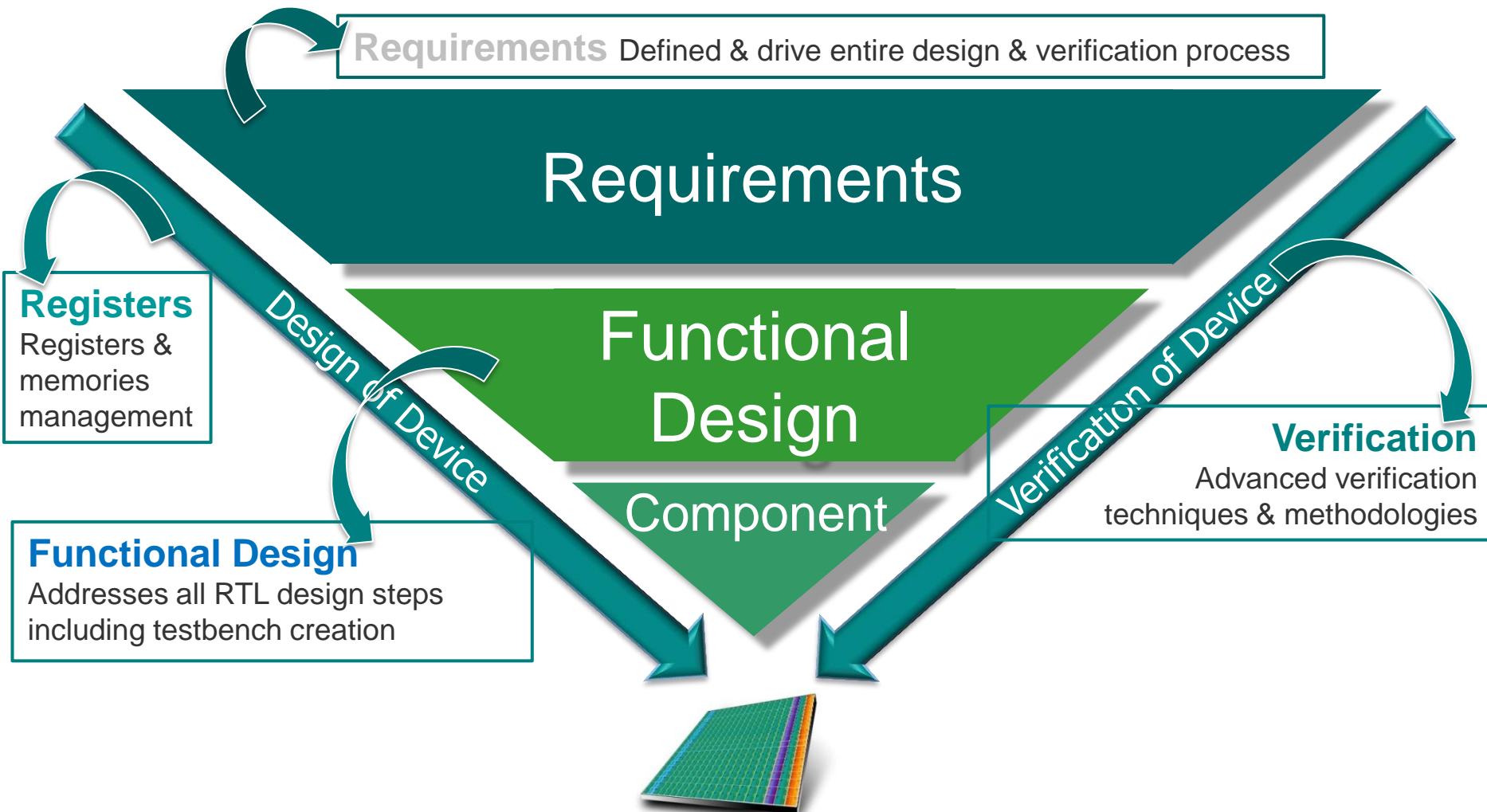
↓ Design Development Cycle

↑ Quality of Final Product

↓ Project Risk



Requirements-Driven Design Process For Today's FPGAs & ASICs



Requirements-Driven Design is Good Design Assurance Practice

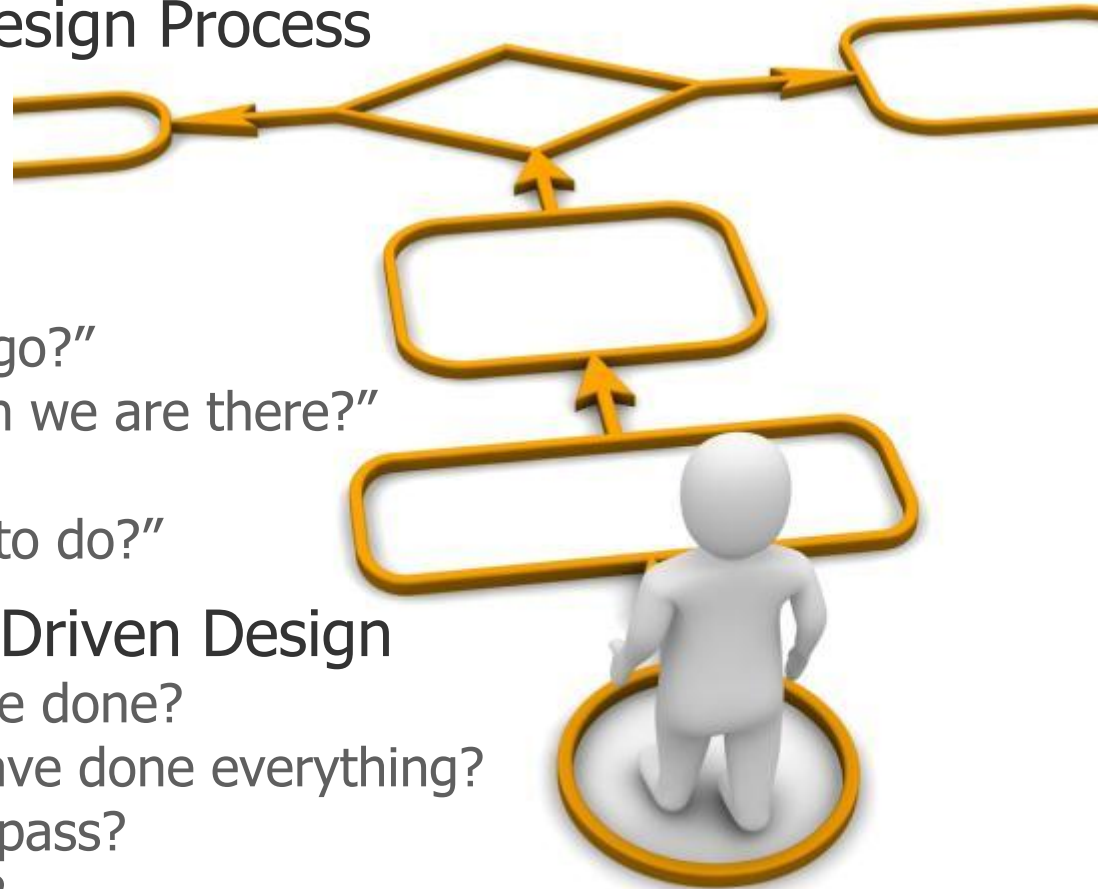
A Requirements-Driven Design Process Instills Structure to the Design Flow

■ States Specifically

- “Where do we need to go?”
- “How do we know when we are there?”
- “Is everything done?”
- “Is there anything else to do?”

■ Without Requirements-Driven Design

- How do we know we are done?
- How do we know we have done everything?
- Will this chip work first pass?
- Will meet our schedule?
- Will we beat the competition?
- Will we achieve success?



Benefits of a Requirements-Driven Design Process

↑ Efficiency

- Design & validate to requirements
- Enhance processes to make more efficient

↑ Productivity

- New techniques

↑ Predictability

- Full project visibility = knowledge

↑ Quality

- Device fully verified against requirements

↓ Risk

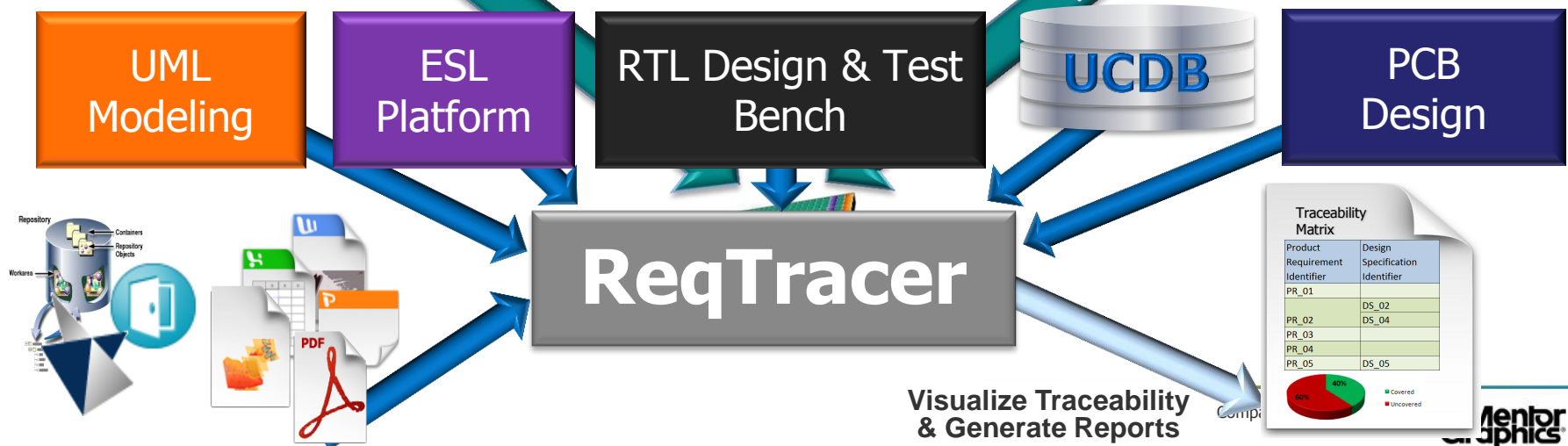
- Structured, repeatable design flow



Requirements Tracking & Management

Requirements

- Manage a Requirements-Driven Design Process
- Trace Requirements Throughout all Design Phases
- Improve Quality & Predictability
- Understand the Impact of Requirement Changes



Requirements Capture

Gathering Requirements from Numerous Sources

Design Specification

Identifier

Attributes

References

Description

DS_02

Priority : HIGH

Covers : PR_02

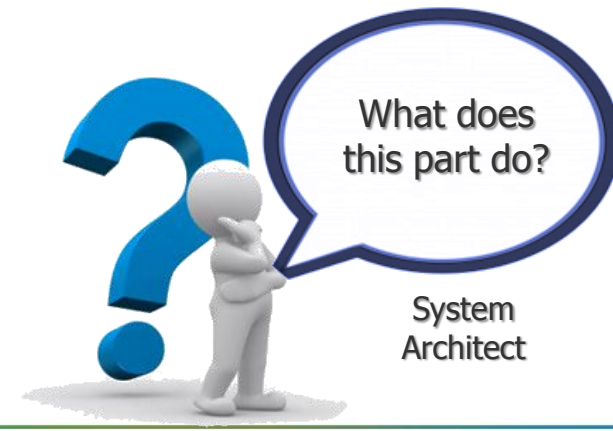
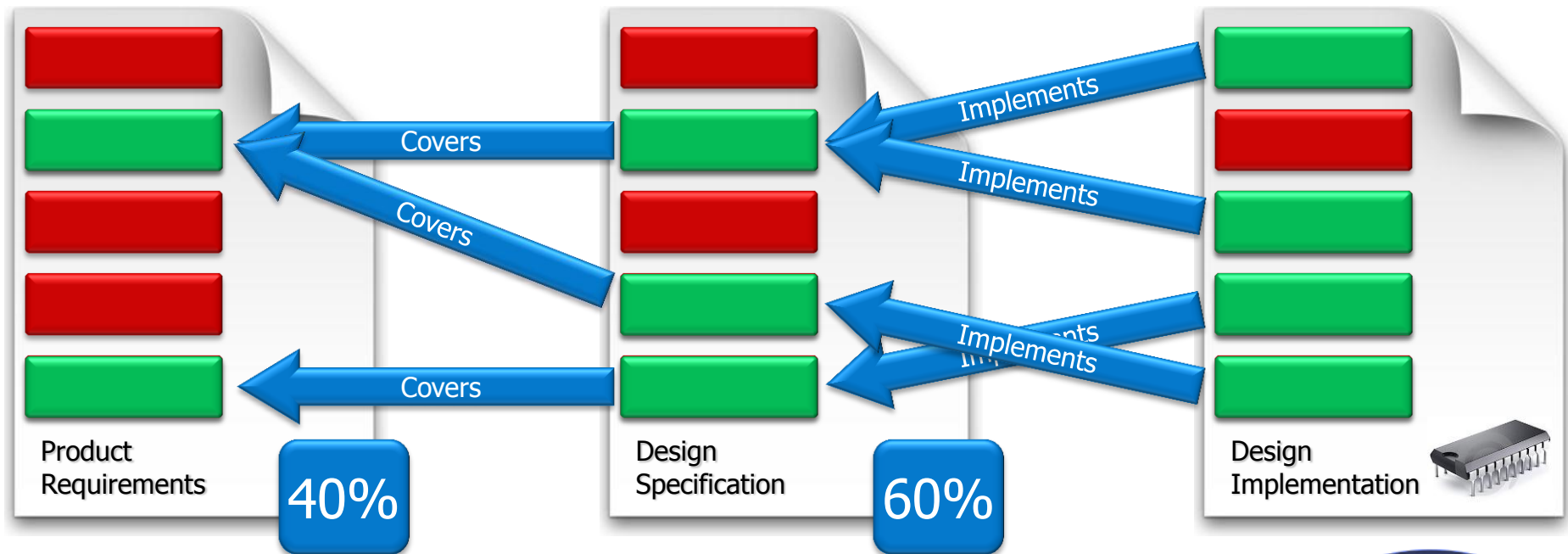
The design shall be power efficient

- Analyze Requirement Sources in Place
 - Another database is not needed!
- Requirements Extracted From
 - DOORS
 - Requisite Pro
 - Many other tools & databases
- Tagger Facility Captures Requirements within Existing Word & PDF Documents
 - Leverage what you are already doing

Each requirement consists of a unique identifier, defined attributes, upstream references & it's textual description

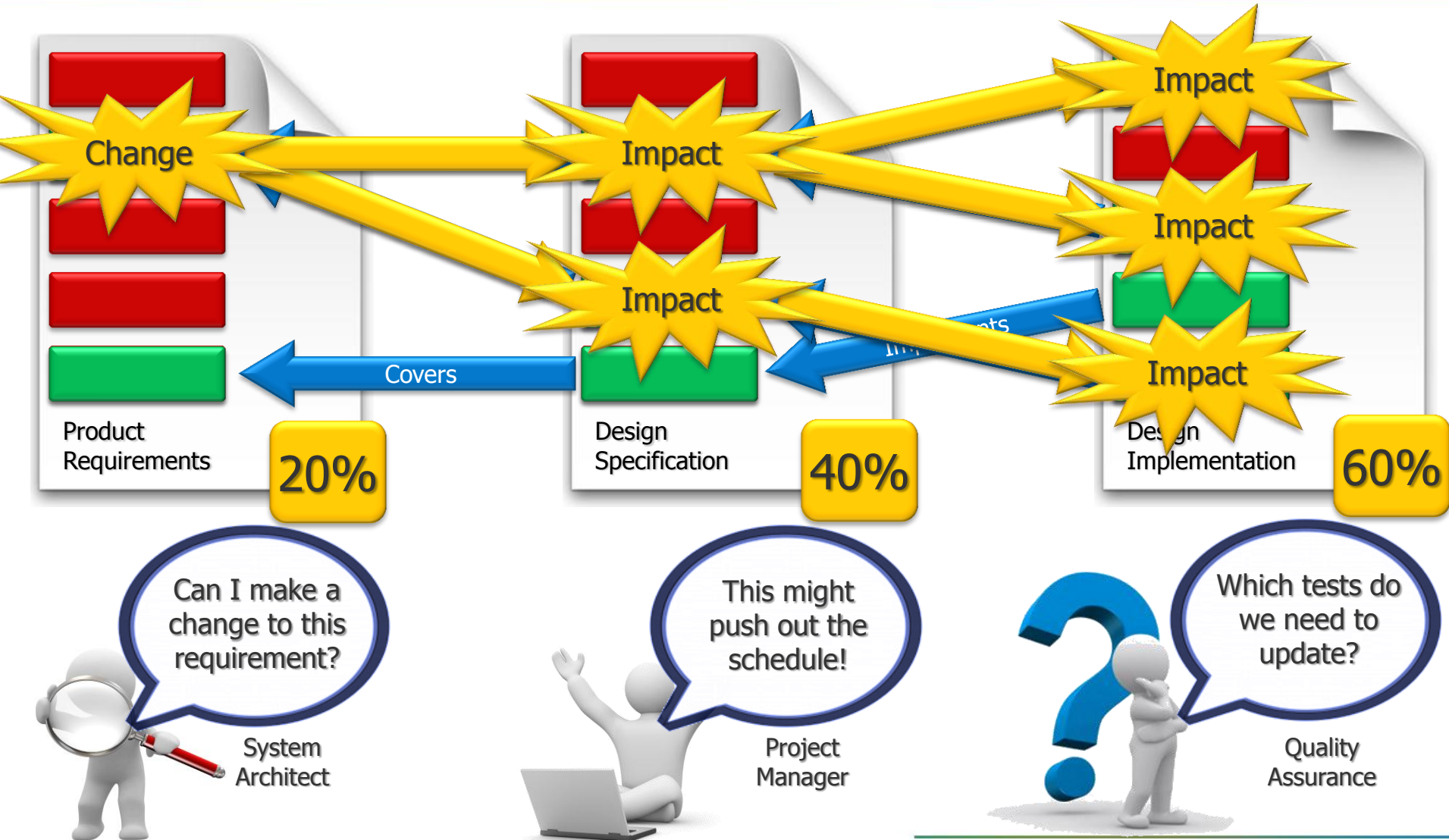
Requirements Tracing

Throughout the Development Stages



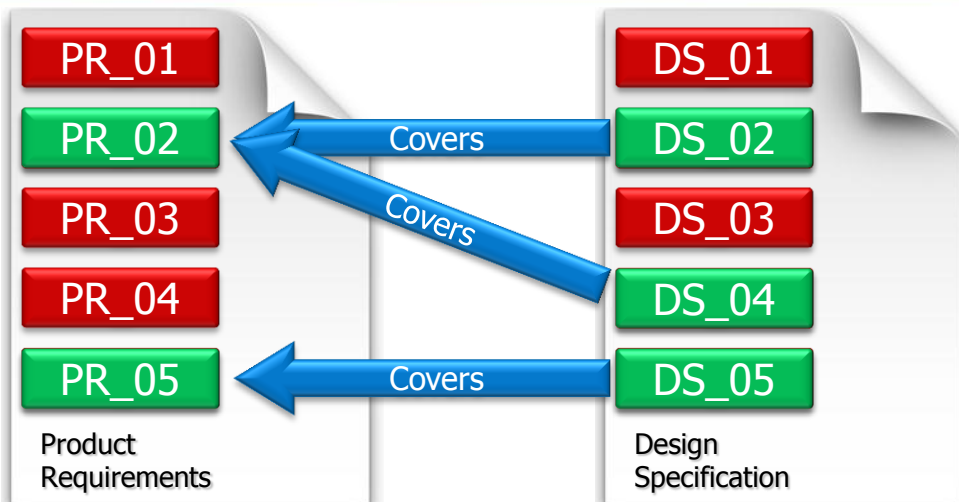
Impact Analysis

Understand the Effect of an Engineering Change Order (ECO)



Report Generation

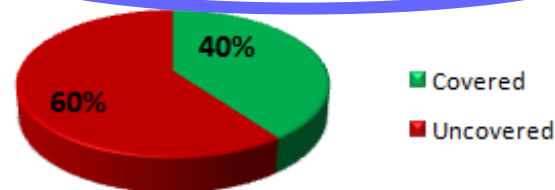
Automatic & Customizable Reporting



- Common Report Generation
 - Traceability matrix
 - Impact analysis
- Fully Customizable
 - Content, style, output format
- Accurately Reflects Current State of the Live Project Data
- Generate Artifacts for Safety Compliance Audits or Design Reviews

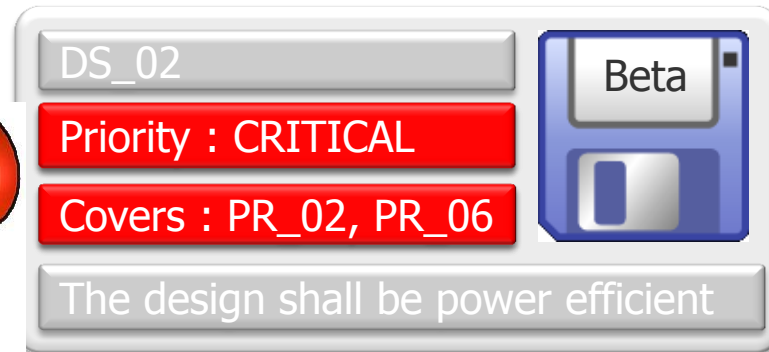
Traceability Matrix Report

Product Requirement Identifier	Design Specification Identifier
PR_01	
PR_02	DS_02
PR_03	DS_04
PR_04	
PR_05	DS_05



Managing Change

Snapshot & Compare Project Status



- Ability to Compare the Requirement Data at Points Throughout the Project Life Cycle
 - Requirements definition
 - Traceability information
- Generate Impact Analysis Reports
 - Based upon the changes that have occurred
- Compare Between Different Snapshots
 - Or against the current analysis of the live data
- Automate Communication Between Team Members
 - When changes occur

Track Verification of Requirements

- Links Verification Results into Requirements-aware Relationships
 - Associate verification results with verification requirements
 - Test Plan
 - Verification Results
 - Analyze direct correlation between requirements & verification
- Imports Results Directly from the ModelSim/Questa Unified Coverage Database (UCDB)
- Confirms all Design Requirements Have Been Designed In & Fully Tested
 - Verification process fully tests all the design requirements
- Reports Coverage Levels Achieved & Pass/Fail Status

Verification Results		My_Questa
0 - testplan	(85.64%)	
3 - Cache Functionality	(85.64%)	
3.1 - CPU Read Hit	(100%)	
3.2 - CPU Read Miss	(100%)	
3.3 - CPU Write	(100%)	
3.4 - Memory Read	(48.24%)	
3.5 - Memory Write	(89.06%)	
3.6 - Reset	(100%)	
3.7 - Random Testing	(62.23%)	



Requirement Coverage

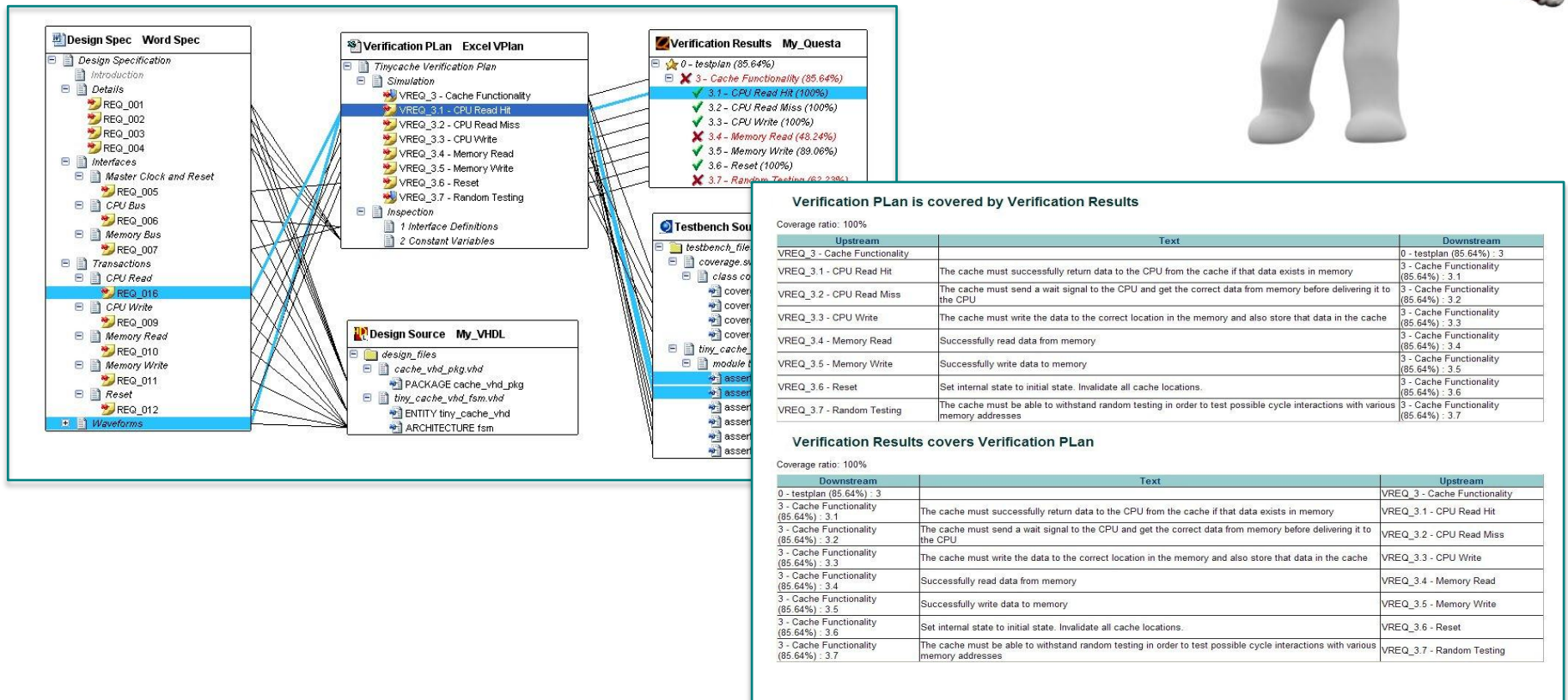
Throughout The Entire Design

- Automated Report Generation
 - Up to the minute status reports
 - Detail of overall project status
 - Keeps project on schedule
- Interactive Visibility of the Requirement Coverage Throughout All Design Data
 - Analyze potential problems & bottlenecks
- Traceability can Span Multiple Design Domains
 - ESL, RTL, PCB, etc.



At every stage of the design

■ Requirements tracking & status reports



Requirements Reviews & Audits

- All Requirements, Including Derived Requirements, Must be Validated
- Must Show Tracking Review Status, Action Items & Owners
- Generate Review Status Reports



Certification Authorities will
audit your DO-254 project!

You may also have internal
process reviews



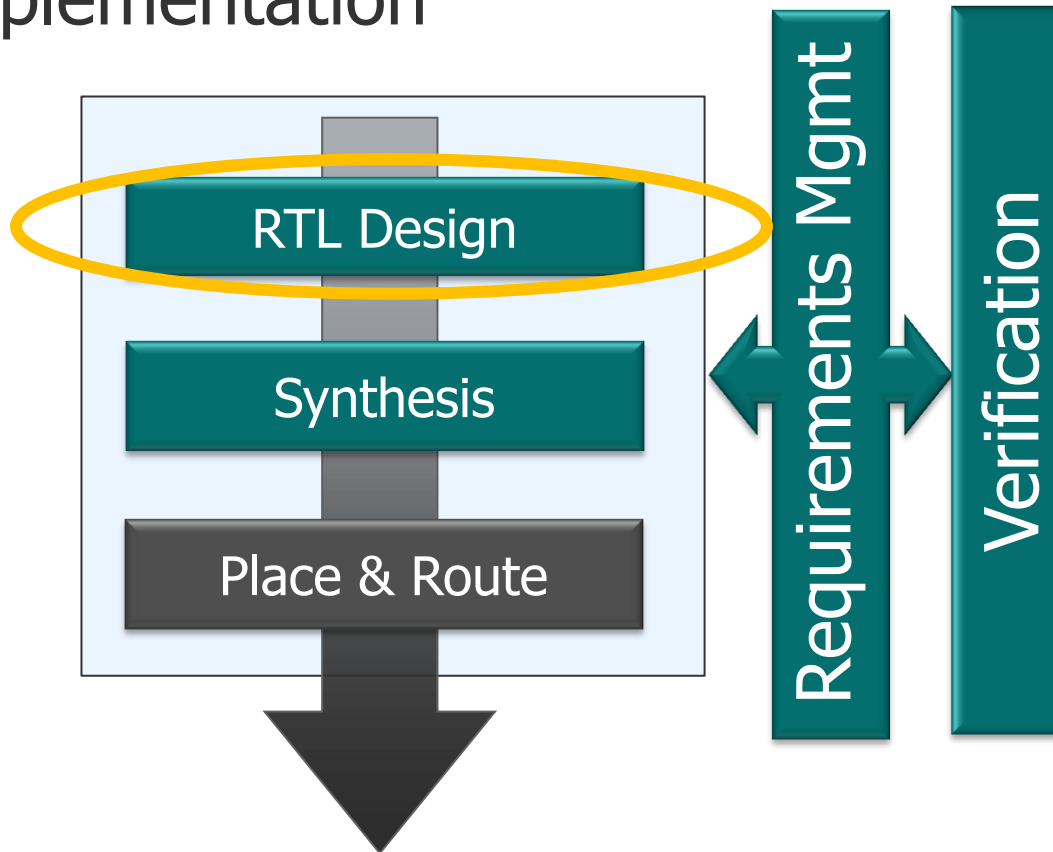
Questions?



Functional Design

Functional

RTL → Implementation



Good RTL Design Practice

- Enforce a Structured Flow
- Create New RTL Code According to Requirements
- Tag Code to Requirements
- Define & Check Against Coding Rule Standards
- Reuse Existing RTL Code
- Document Project
- Review Code & Project
- Keep Code & Project Under Version Management



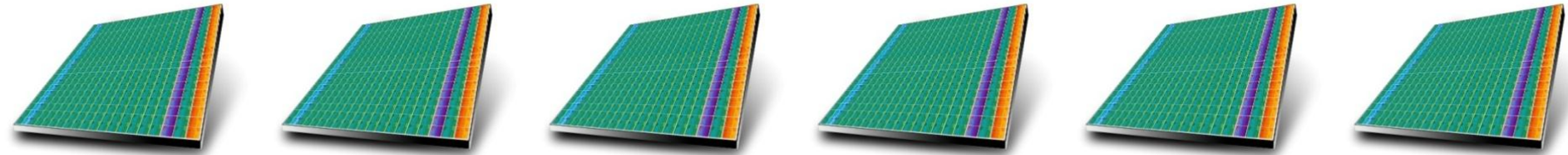
These Are All Features of HDL Designer

Structured Flow

- **Ensure Consistency of Tool Versions Used Across Team**
 - Of HDL Designer & all downstream tools that are part of the flow
 - Simulation, synthesis, P&R, Formal, etc.
 - Tools in “Built-in”
 - ModelSim/Questa
 - Precision Synthesis
 - FPGA Vendor Tools & P&R
 - 3rd Party tools
- **Coding Templates**
- **Team Preferences**
- **Tasks to Automate Repetitive Flows**
 - Quickly integrate your own tools
 - Build your own custom design flows
 - Consistency of tool versions in Tasks



Repeatable Design & Design Process



- Must be Able to Reproduce the Design & Design Flow
 - Overall helps improve quality & productivity
 - Needed in all design assurance projects
 - Required by DO-254
- Requires
 - Consistency of
 - Produced results
 - Tools used by all team members
 - Flows followed
 - Tasks applied (ex. Code checking rules)
 - Version control of design project

Creating New RTL Code

Templates

Language Templates

- Module
- Primitive
- Declarations
- Statements
- Instantiations
- Compiler Directive
- Blocks
- Always
- Initial
- System Tasks and Func

Drag'n'Drop

Component Browser

- moduleware
 - Arithmetic
 - Bit Manipulation
 - Combinatorial
 - Constants
 - Logic
 - Memory
 - Primitives
 - Registers
 - Sequential
 - Stimulus
- Sequencer_vhd
 - accumulator
 - fibgen
 - fibgen_tb

Component: Sequencer_vhd.accumulator (VHDL)

To instance a component, drag onto a diagram and Paste.

0: F:\TOOLS\hds2003.1rc/examples/uart_txt/hdl/uart_top_rtl.vhd (read-only)

File Edit Search View Document Options Window Help

Find Block:

Code Browser

Parser Level: Full

uart_top_rtl.vhd

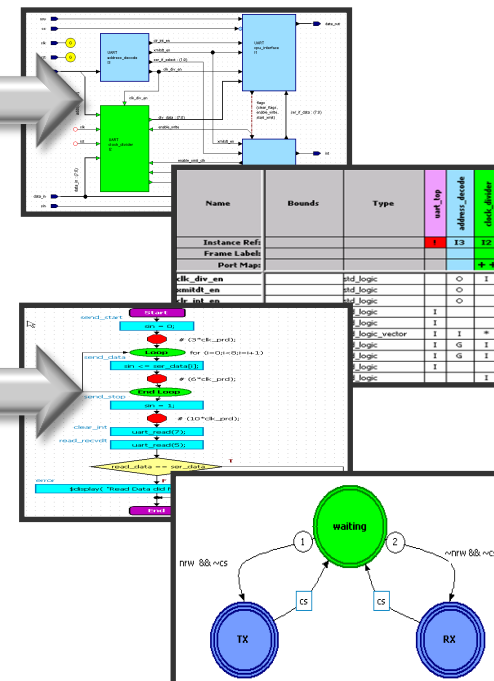
```

36 ARCHITECTURE rtl OF uart_top IS
37
38   -- Internal signal declarations
39   SIGNAL clear_flags      : std_logic;
40   SIGNAL clk_div_en       : std_logic;
41   SIGNAL clr_int_en       : std_logic;
42   SIGNAL div_data         : std_logic_vector(7 DOWNTO 0);
43   SIGNAL enable_rcv_clk   : std_logic;
44   SIGNAL enable_write     : std_logic;
45   SIGNAL enable_xmit_clk  : std_logic;
46   SIGNAL sample           : std_logic;
47   SIGNAL ser_if_data      : std_logic_vector(7 DOWNTO 0);
48   SIGNAL ser_if_select    : std_logic_vector(1 DOWNTO 0);
49   SIGNAL start_xmit       : std_logic;
50   SIGNAL xmitdt_en        : std_logic;
51
52   -- Component Declarations
53   COMPONENT address_decode
54   PORT (
55     addr      : IN std_logic_vector(12 DOWNTO 0);

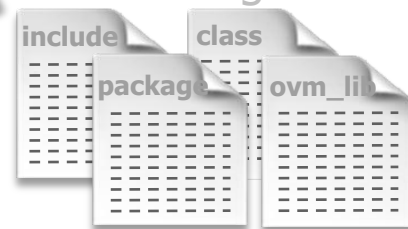
```

- HDL Aware**
 - Full featured text editor
 - Syntax checks as you edit
- Language Support**
 - VHDL, Verilog, SystemVerilog, PSL, C/C++, Tcl, XML
- Language Templates**
 - Accelerate design entry
- Drag & Drop**
 - Automatically declares signals for instance ports & maps them
 - Automatically handles package & library references
- Cross Reference**
 - To errors & graphical views
- Fast Design Navigation Features**
- Emacs & vi Emulation Modes**

Visualization

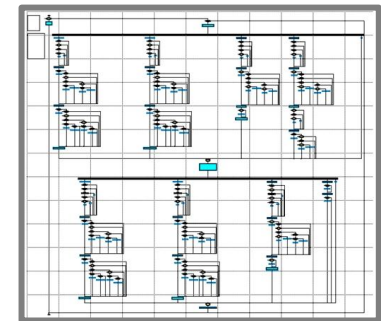
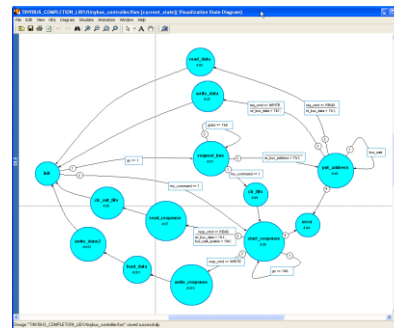
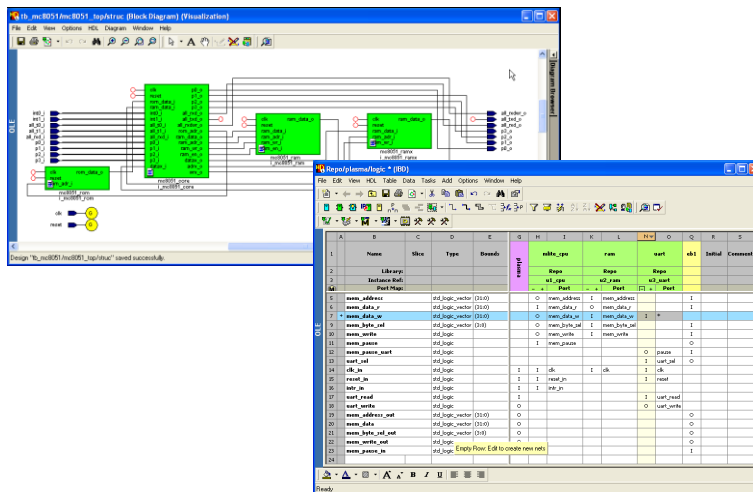


Text Navigation

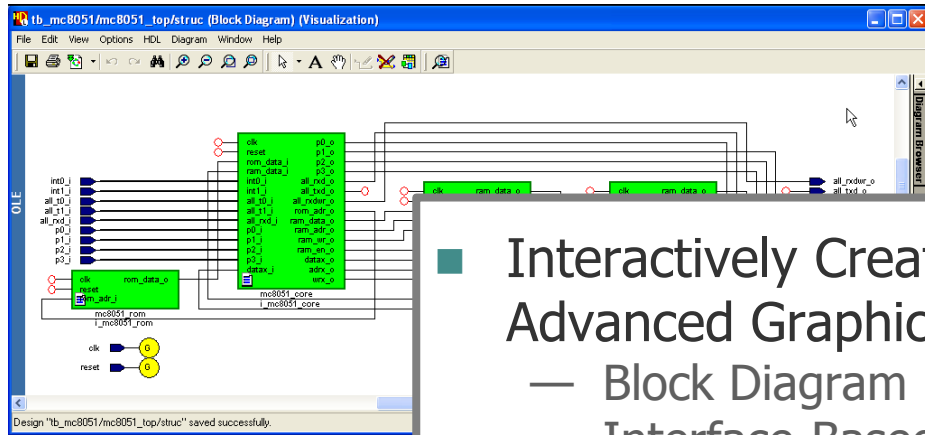


Code Visualization

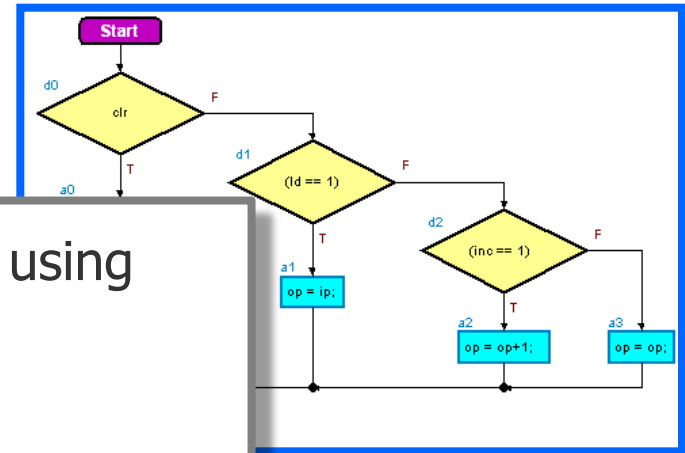
- Automated Visualization of RTL code
- Visually Navigate Design for Understanding Purposes
- Documentation-ready Results
 - Structural RTL as Block Diagrams and/or Spreadsheets
 - Control Logic as State Machines
 - Sequential Code as Flow Charts



Also RTL Design Graphical Entry



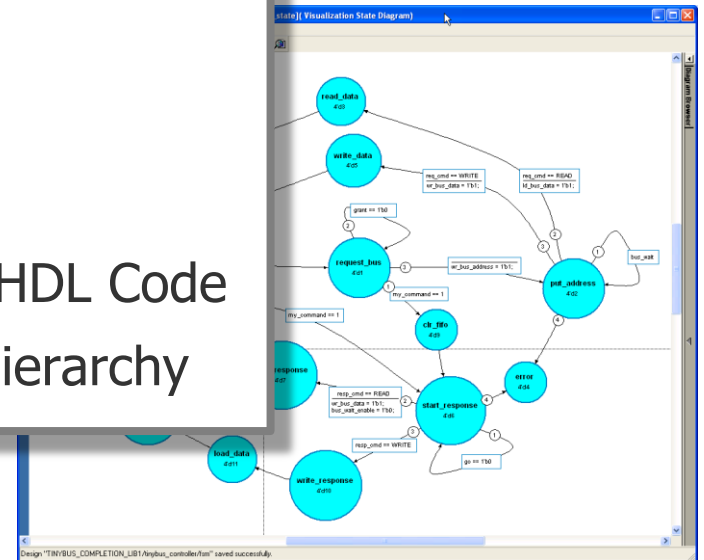
- Interactively Create Design using Advanced Graphical Editors
 - Block Diagram
 - Interface-Based Design
 - intelligent spreadsheet
 - State Machine
 - Flow Chart
- Automatically Generate “Correct-by-Construction” HDL Code
- Visually Navigate Design Hierarchy



Repo/plasma/logic * (IBD)

Name	Slice	Type	Bounds
mem_address	std_logic_vector (31:0)		
mem_data_r	std_logic_vector (31:0)		
mem_data_w	std_logic_vector (31:0)		
mem_byte_sel	std_logic_vector (3:0)		
mem_write	std_logic		
mem_pause	std_logic		
mem_pause_uart	std_logic		
uart_sel	std_logic		
clk_in	std_logic		
reset_in	std_logic		
intr_in	std_logic		
uart_read	std_logic		
uart_write	std_logic		
mem_address_out	std_logic_vector (31:0)		
mem_data	std_logic_vector (31:0)		
mem_byte_sel_out	std_logic_vector (3:0)		
mem_write_out	std_logic		
mem_pause_in	std_logic		

Ready



Convert to Editable Graphics

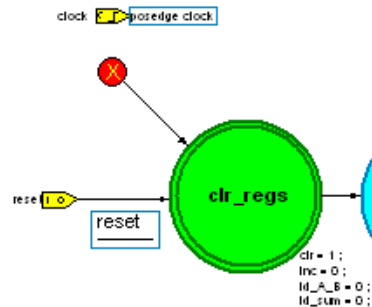
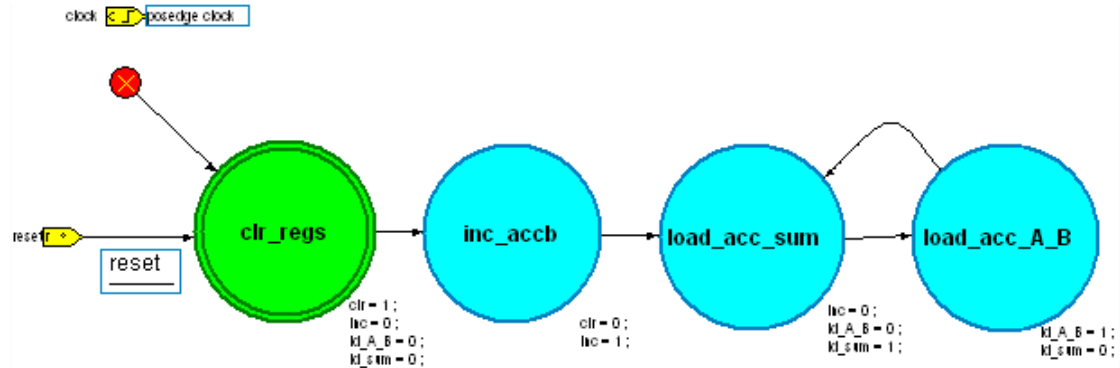
```

ARCHITECTURE struct OF struct IS
-- Architecture declarations
-- Internal signals declarations
SIGNAL mem_address : std_logic_vector ( 31 DOWNTO 0 );
SIGNAL mem_type_sel : std_logic_vector ( 3 DOWNTO 0 );
SIGNAL mem_data_w : std_logic_vector ( 31 DOWNTO 0 );
SIGNAL mem_data_r : std_logic_vector ( 31 DOWNTO 0 );
SIGNAL mem_status : std_logic;
SIGNAL mem_wait : std_logic;
SIGNAL mem_sel : std_logic;

-- Component declarations
COMPONENT struct_mem
PORT (
    clk_in : IN std_logic;
    mem_address : IN std_logic_vector ( 31 DOWNTO 0 );
    mem_type_sel : IN std_logic_vector ( 3 DOWNTO 0 );
    mem_data_w : IN std_logic_vector ( 31 DOWNTO 0 );
    mem_data_r : OUT std_logic_vector ( 31 DOWNTO 0 );
    mem_status : OUT std_logic;
    mem_wait : OUT std_logic;
    mem_sel : OUT std_logic;
);
END COMPONENT;

-- Component declarations
COMPONENT struct_mem
PORT (
    clk_in : IN std_logic;
    mem_address : IN std_logic_vector ( 31 DOWNTO 0 );
    mem_type_sel : IN std_logic_vector ( 3 DOWNTO 0 );
    mem_data_w : IN std_logic_vector ( 31 DOWNTO 0 );
    mem_data_r : OUT std_logic_vector ( 31 DOWNTO 0 );
    mem_status : OUT std_logic;
    mem_wait : OUT std_logic;
    mem_sel : OUT std_logic;
);
END COMPONENT;
    
```

1. Convert Code



2. Make a Change

3. Generate Code

```

ARCHITECTURE struct OF struct IS
-- Architecture declarations
-- Internal signals declarations
SIGNAL mem_address : std_logic_vector ( 31 DOWNTO 0 );
SIGNAL mem_type_sel : std_logic_vector ( 3 DOWNTO 0 );
SIGNAL mem_data_w : std_logic_vector ( 31 DOWNTO 0 );
SIGNAL mem_data_r : std_logic_vector ( 31 DOWNTO 0 );
SIGNAL mem_status : std_logic;
SIGNAL mem_wait : std_logic;
SIGNAL mem_sel : std_logic;

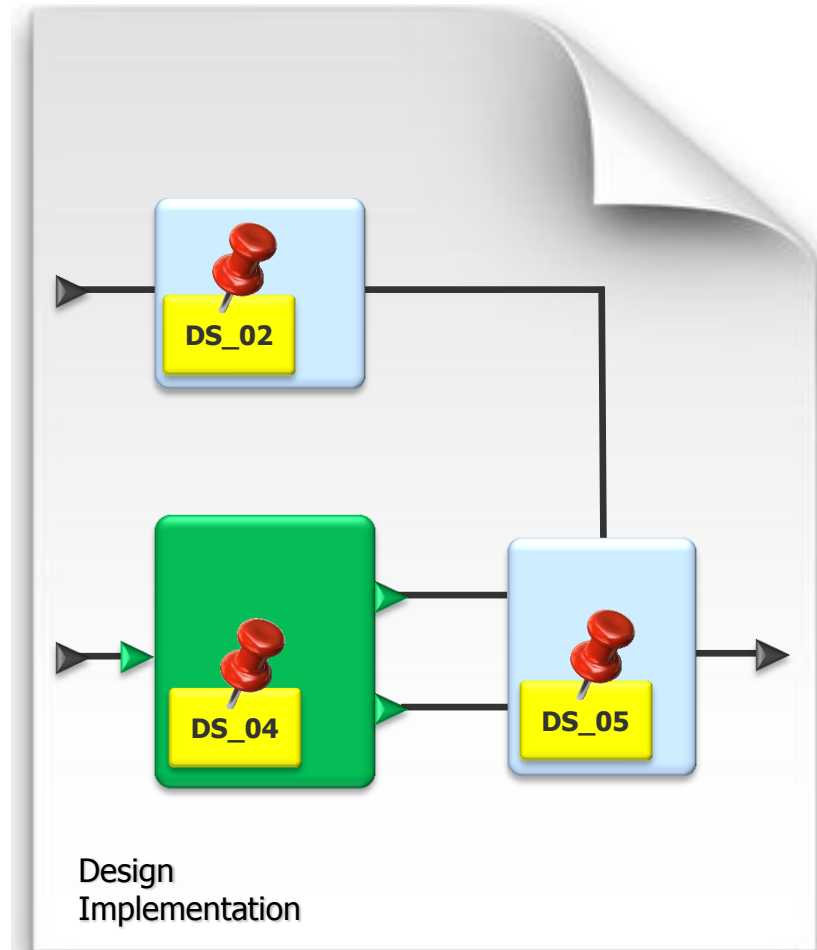
-- Component declarations
COMPONENT struct_mem
PORT (
    clk_in : IN std_logic;
    mem_address : IN std_logic_vector ( 31 DOWNTO 0 );
    mem_type_sel : IN std_logic_vector ( 3 DOWNTO 0 );
    mem_data_w : IN std_logic_vector ( 31 DOWNTO 0 );
    mem_data_r : OUT std_logic_vector ( 31 DOWNTO 0 );
    mem_status : OUT std_logic;
    mem_wait : OUT std_logic;
    mem_sel : OUT std_logic;
);
END COMPONENT;

-- Component declarations
COMPONENT struct_mem
PORT (
    clk_in : IN std_logic;
    mem_address : IN std_logic_vector ( 31 DOWNTO 0 );
    mem_type_sel : IN std_logic_vector ( 3 DOWNTO 0 );
    mem_data_w : IN std_logic_vector ( 31 DOWNTO 0 );
    mem_data_r : OUT std_logic_vector ( 31 DOWNTO 0 );
    mem_status : OUT std_logic;
    mem_wait : OUT std_logic;
    mem_sel : OUT std_logic;
);
END COMPONENT;
    
```


Tag RTL Code to Requirements

From within the Design Coding

- Paste Requirement References onto Design Objects within HDL Designer
 - Enhances documentation
 - Textual RTL code view
 - Graphical design view
 - Design Manager view
- Navigate Easily Between Requirement Definitions & Design
 - Cross-referenced
- Generated RTL Contains the References
 - Guaranteeing traceability down to the source code



Design Code Checking: Key to Successful Design

- What is Design Checking?
 - Coding guidelines to teach/enforce design engineers to write design code effectively & efficiently
- Design Checking Ensures
 - Catching potential design problems in RTL code
 - Avoids potentially dangerous coding styles
 - Without checking, may not be detected until downstream design or production usage
 - Higher cost to fix downstream
 - Cheaper & easier to correct in the coding phase
 - Consistent style, reusability & reliability practices
 - Improves code comprehension, portability & reviews



Historically Has Been Applied to ASICs,
but Rapidly Growing Usage on FPGA Designs

Automatic Rule Checking

- Empowers Each Designer to Code According to Guidelines
- Promotes Regular RTL Design Checking
 - Throughout the design development process
 - Prior to & during simulation & synthesis
- Prevents Human Errors
 - Too easy to miss violations when reviewing thousands of lines of code
- Reduces Manual Code Review Effort
 - Most rules can be automated
- Speeds Design Review Processes
 - Rapidly performs checks & summarizes results
- Improves Design Reuse
 - Saves costs on future projects



3 Types of HDL Designer Design Checking Rules

■ Style Rules

- Naming & coding
- Format & readability
- Reporting


■ Synthesis Rules

- Identify synthesis & in-field reliability problems
- Examples:
Latch inference, Unreachable blocks, Register race, Gated clocks, Combinatorial feedback

■ Design-wide Rules

- Applied across the whole design
- Examples:
Snake paths, Clock used as data, Shared clock & reset signal, Un-driven logic, Unused logic

7 Pre-configured, Parameterizable Rulesets

- Essentials Ruleset  Excellent Starting Point for Ruleset Creation
 - Core rules the design community considers essential to identify code that would fail in downstream tools
- RMM Ruleset
 - Pre-configured implementation of the RTL coding guidelines described in the third edition of the Reuse Methodology Manual co-developed by Mentor Graphics & Synopsys
- Safety-Critical Ruleset
 - High-impact checks for safe coding practices to avoid typical design errors & to provide safe synthesis
- DO-254 Ruleset
 - Extends the Safety-Critical ruleset to add code checks that ensure efficient code reviews
- Checklist Ruleset
 - Rules previously performed by 0-In[®] Checklist
- Altera Ruleset
Xilinx Ruleset
 - Pre-configured to be compatible with the design code implemented using Altera & Xilinx tools

Dramatic ROI

- Checking Time ↓ While Quality of Code ↑

Manual Checking	10,000 Lines@200/hr → 50 hrs
Automatic Checking	10,000 Lines → 2mins 40 sec

- Automatic Design Checking Significantly Reduces Project Costs
 - Greatly speeds checking
 - Repeated numerous times during design
 - Catches violations before simulation, synthesis & production



All the Creation Features Help Improve Design Reuse

Create Highly Reusable, Guidelines-Compliant RTL Code & Leverage Previously Written HDL



1. Automatically identify missing files & syntactical problems
 - *Do I have all the files?*
 - *Are they syntactically correct?*
2. Use design checking to assess quality of code against corporate coding standards & scoring criteria
 - *How well is the code written?*
 - *Is it better to create it new, from scratch?*
3. Visualize code for enhanced design comprehension & improved documentation
 - *Is there documentation?*
 - *Can I rapidly understand the code?*

Design Documentation

■ Many Different Types of Documentation Files

- Source data files
- Requirements
- Testbench
- Stimulus files
- Verification results
- Constraints
- Functional description documentation

■ Documentation Can Also Include

- Code comments
- Coding standards
- Quality metrics
- Design visualizations
- Tool scripts



Good Design Practice Requires Thorough Documentation

Simplifying Design Reviews

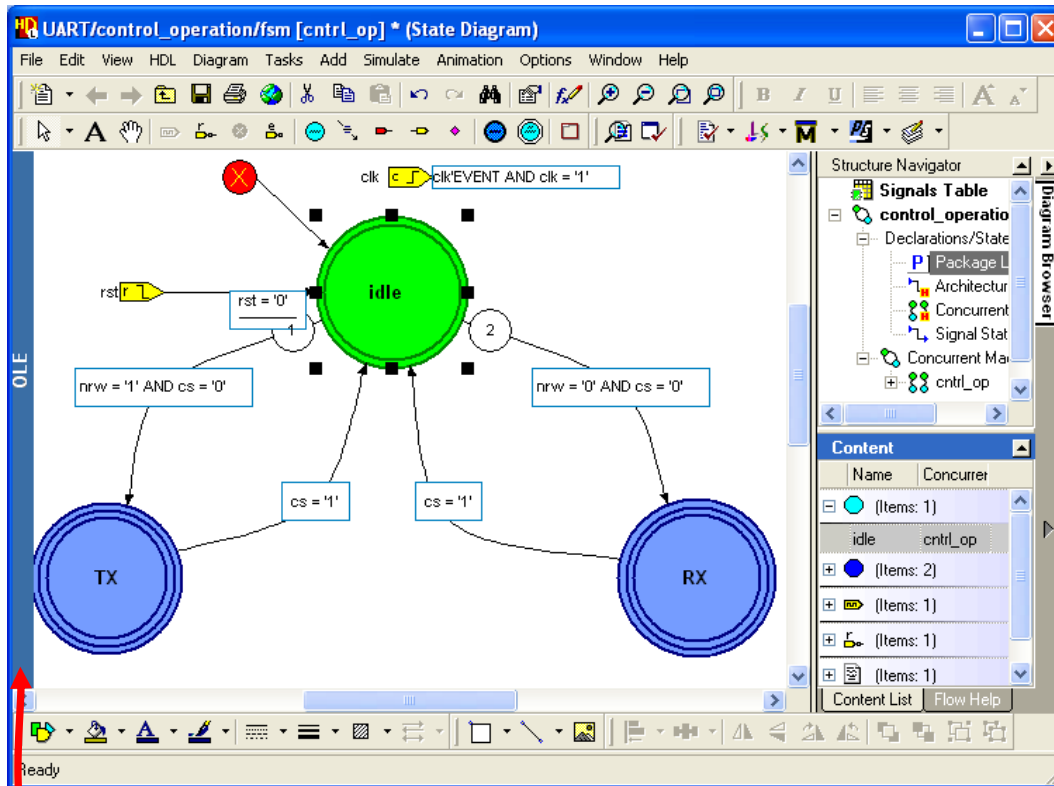
Many Automated Documentation Capabilities in HDL Designer

- Export into design documents
- OLE drag'n'drop into Microsoft applications
- Export to Visio
- Import & export spreadsheets
- Export to HTML

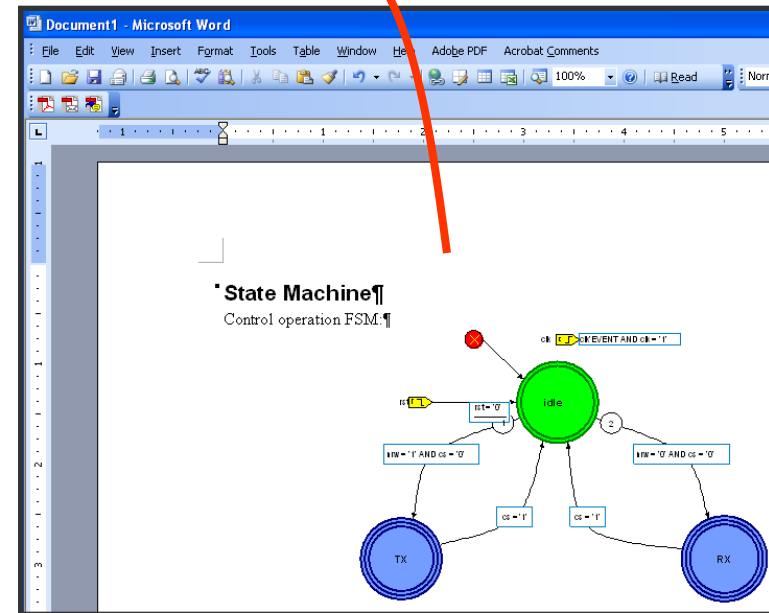


Static OLE Drag & Drop

Drag & drop from any HDL Designer view



Click to Edit



Word®

OLE Bar

Efficient Design Reviews w/HTML

Design Explorer [Using viewpoint : Design_Hierarchy]

Design Unit | File Name

- CM_uart_v_lib
 - address_decode address_decode_tbl.v
 - clock_divider clock_divider_flow.v
 - control_operation control_operation_fsm.v
 - cpu_interface cpu_interface_intconx.v
 - serial_interface serial_interface_struct.v
 - status_registers status_registers.v
 - tester
 - uart_tb
 - uart_top
 - xmit_rcv

HDL Designer Web Export - Windows Internet Explorer

C:\Tools\hdocs\HDSData\CM_uart_v_lib\uart_tb\HTML\Index.htm

File Edit View Favorites Tools Help

Go [Google] [HDS IP REPOSITORY] [RTL Repository] [CM_uart_v_lib\uart_top\str...] [HDL Designer Web Export X] Home Feeds (J) Print Page

Diagram Information Side Data HDL

Ctrl + Left click to zoom in
Ctrl + Left drag to zoom area
Ctrl + Shift + Left click to zoom out
Alt + Left drag to pan
Right click on diagram for find

Click Net below to Highlight

- 0: addr
- 1: bundle_U_1_U_4_0
- 2: bundle_U_3_U_1_0
- 3: bundle_U_3_U_4_0

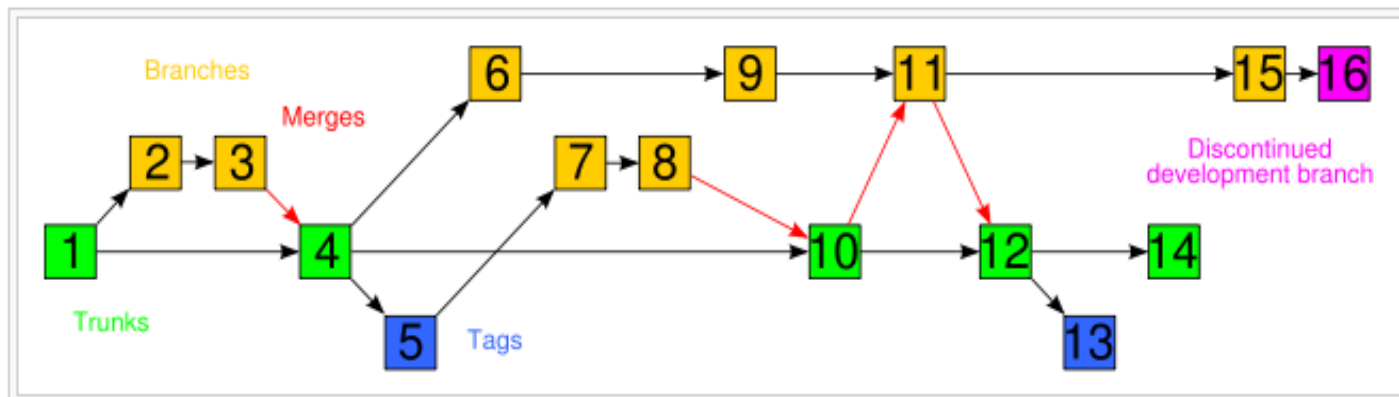
Generated on: 07/ 3/2008 at: 4:56 PM

HDL Designer



Keep Project Under Version Management

- Version Control of the Design Project
 - Of all design/project related data
 - Sandboxes & Team Workspace
 - Archiving of milestones & final design
- HDL Designer Version Management Built-in Interfaces
 - Open source RCS & CVS included
 - Microsoft Visual Source Safe (VSS)
 - Synchronicity (Dassault/Enovia) DesignSync
 - Subversion
 - Rational (IBM) Clearcase
 - ClioSoft SOS
 - Serena Software PVCS



Full Project Visibility

At every stage of the design

■ Code quality metrics



Summary

Settings

Design Quality: 156/198 (79%)

Quality Score: 79%

Score/Total Possible Score: 156/198 Excludes 0 Disabled Rules

Ruleset Hierarchy Report:

Ruleset	Score	%	Error	Warning	Note	Disabled
My_Essentials_Policy	156/198	79%	3	6	0	0
Essentials	156/198	79%	3	6	0	0
Coding Practices	56/72	78%	1	3	0	0
Downstream Checks	64/90	71%	2	3	0	0
Code Reuse	36/36	100%	0	0	0	0

Violations: 45

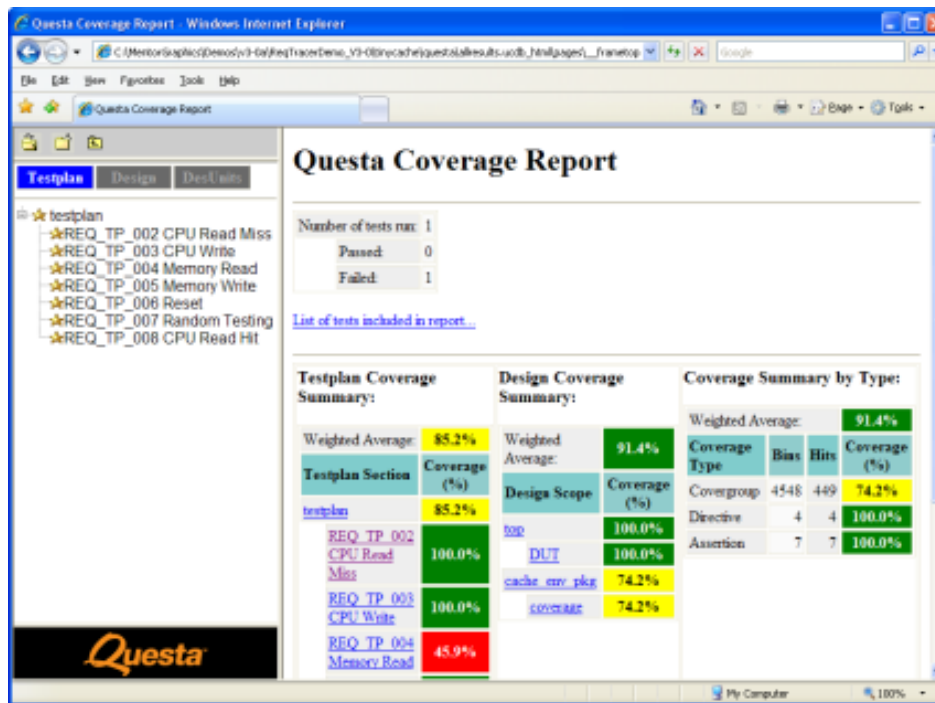
Rules: (Using policy My_Essentials_Policy)

Design Units:

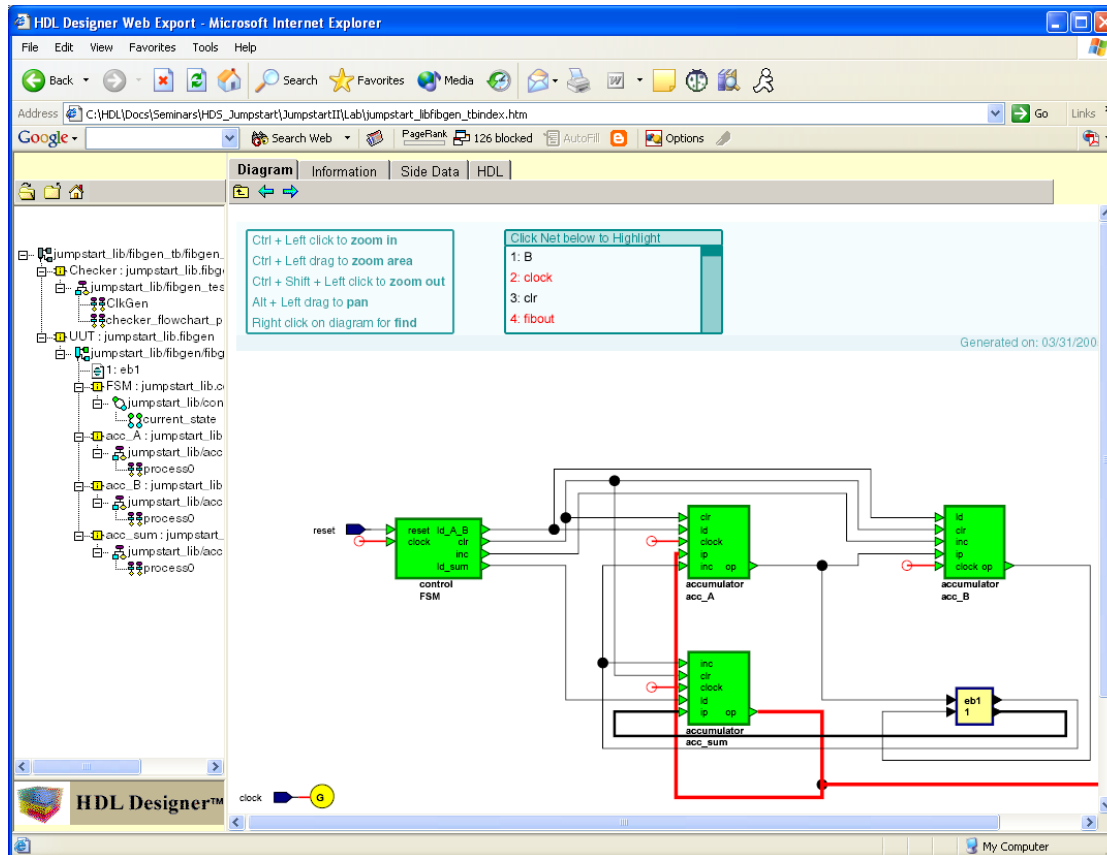
Full Project Visibility

At every stage of the design

- Measurable verification reports



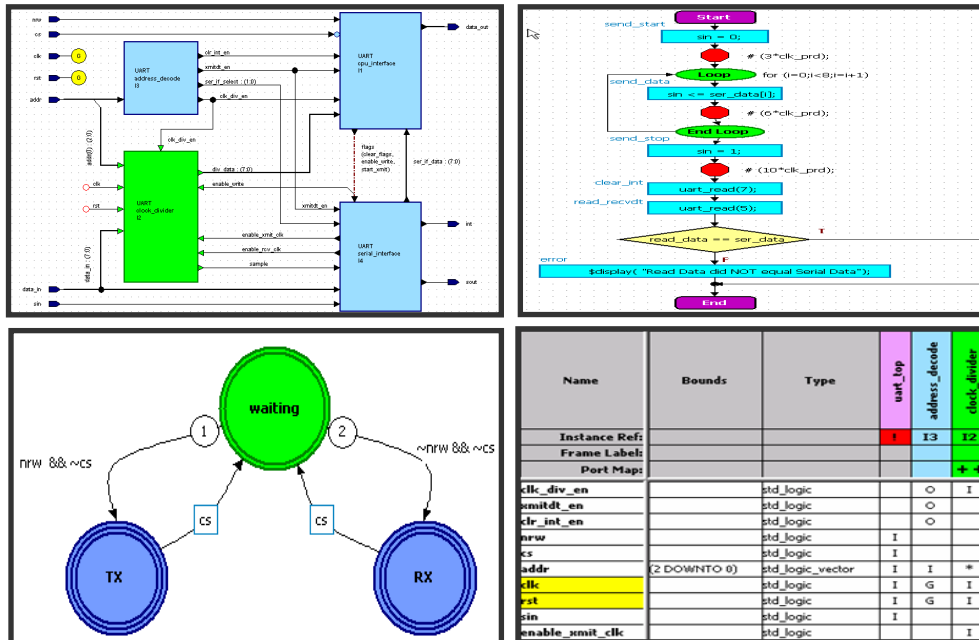
- HTML for team/customer design reviews



Full Project Visibility

At every stage of the design

- Automated visualization & documentation of design



ReqTracer & HDL Designer for a Requirements-Driven Design Process

- ReqTracer
 - Provides a requirements-oriented project management environment from concept to implementation
- HDL Designer
 - Provides the design flow to ensure the device is developed in a structured, repeatable & controlled environment
- ReqTracer + HDL Designer Together Create a Requirements-Driven Good FPGA/ASIC Design Flow
 - Requirements validation
 - Requirements traceability
 - ECO change impact analysis
 - Requirements management
 - Version Management
 - Documentation
 - Automated design rule checking
 - Verification of requirements

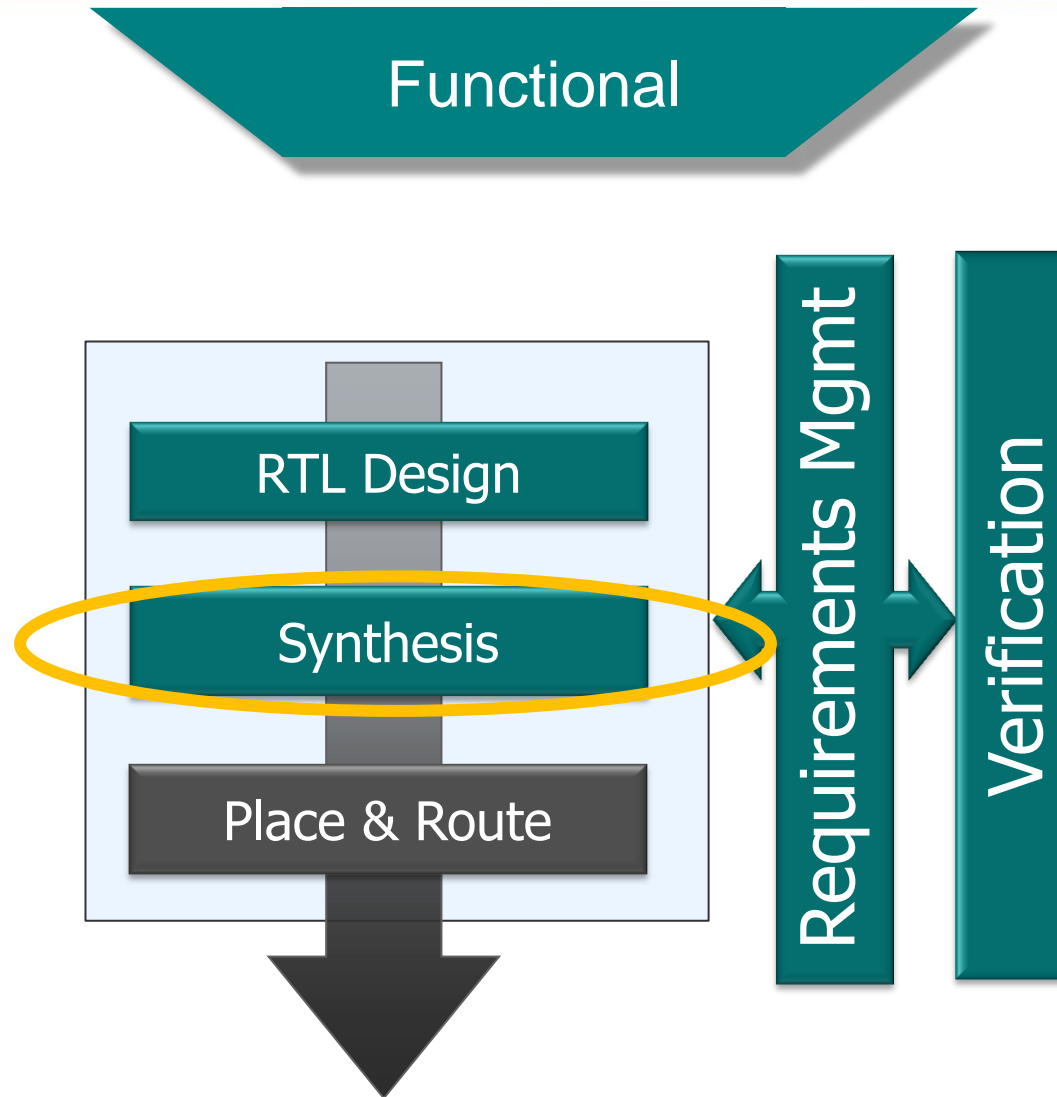
**Good Design Practice: Improved Development Cycle,
Improved Product Quality, Reduced Project Risk**



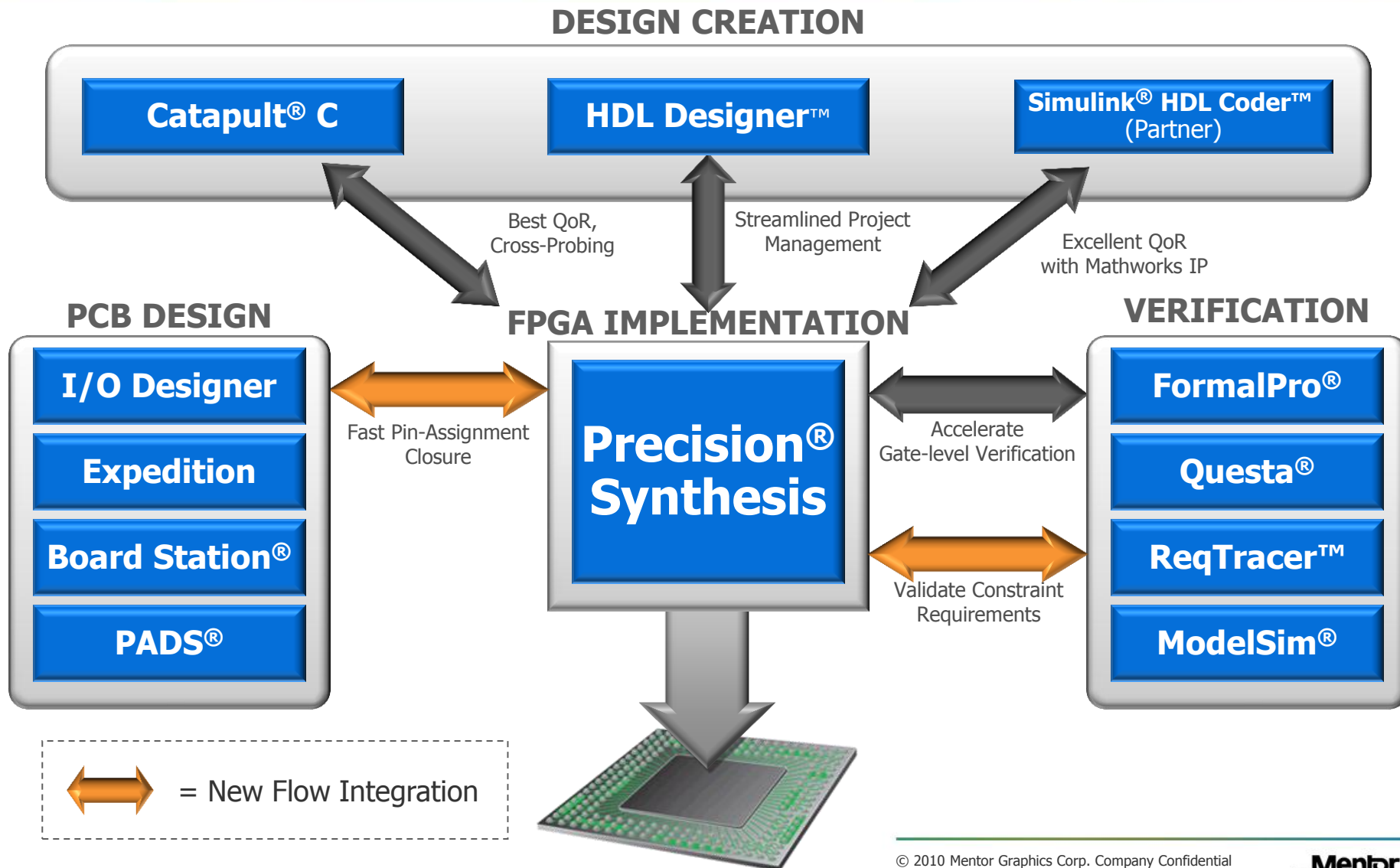
Questions?



Functional Design



Mentor's Complete FPGA Vendor Independent Solution



Integration Across the Flow (for Synthesis)

Precision ↔ Catapult C

Accelerate design creation with C/C++ & RTL Synthesis

- Best of both worlds
- Easier to learn
- Higher productivity

Precision ↔ HDL Designer

- Setup design once for HDL Designer and Precision

RTL re-use, project management from design to implementation

Precision ↔ MS/Quarta Compatibility

- Cross-platform for development and debugging
- Consistent interpretation of HDL

Consistent language interpretation, IP encryption, and synchronized roadmap

Precision ↔ FormalPro

- Synchronized on IP Encryption
- Fast gate-level verification
- High confidence of coverage
- Automated

Post synthesis equivalence checking is important for DO-254, Mil-Aero & Safety-Critical

Precision ↔ I/O Designer

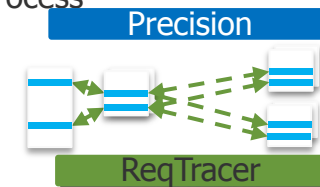
- Reduce system design time
- Optimize overall performance

PCB 

Integration between synthesis and PCB tools enables co-development of FPGAs and board → significantly reduces schedule

Precision ↔ ReqTracer

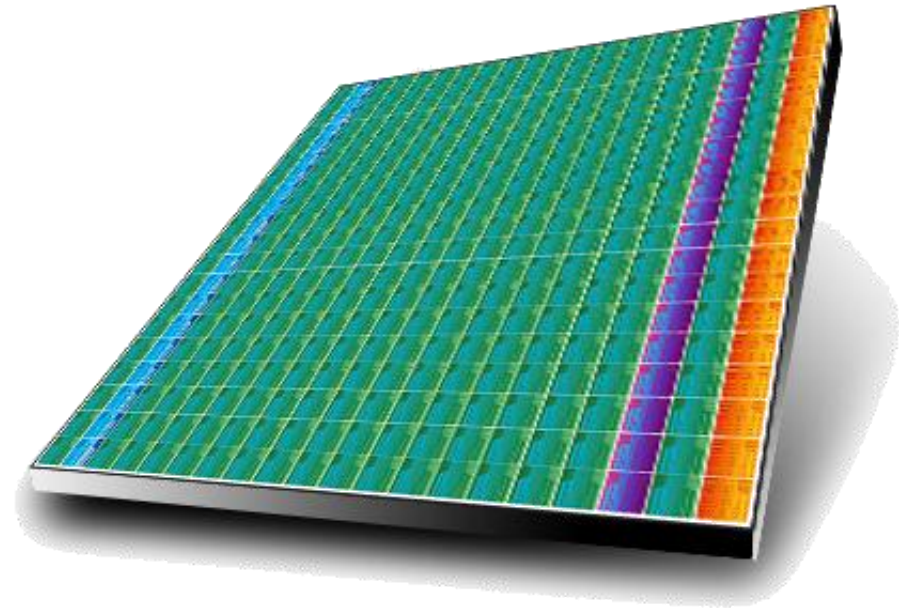
- Assurance that design meets requirements
- Automated process



Tracking that requirements are met by synthesis and place-and-route is critical for DO-254

Precision Synthesis Flow

- Design Import
 - Vendor Independence
 - Standard language support
 - Precise-IP Platform
- Synthesize & Optimize
 - Out-of-the-box Quality of Results
 - Physical synthesis
 - ASIC Prototyping Optimizations
- Design Closure
 - Award-winning analysis & debug
 - Incremental flows
- Industry Specific Capabilities
 - Low Power
 - Mil-Aero & Safety Critical
 - Rad-Tolerant



Vendor Independent Synthesis

- **Minimize cost of** maintenance & training
 - One tool to learn and maintain
 - Easy-to-use GUI and project manager
 - Generate command script to reproduce results from scratch
- **Select best FPGA** for current & future projects
 - Quickly re-target design to different FPGA
 - Objective device selection, regardless of tool expertise
 - Reduce production cost: select cheapest device, negotiation leverage with vendors
- **Mitigate risk** with an extra synthesis tool
 - FPGA vendor synthesis still available as back-up
 - Minimizes the risk of “getting stuck”



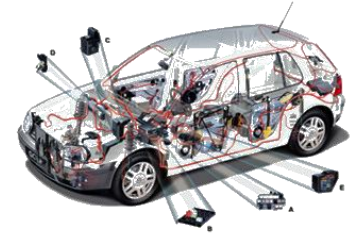
**FPGA Vendor Lock-In = missed opportunity to use
reduce cost, select best device, reduce risk**

Vendor Independence Success Story

High-End Automotive System Company

Requirements

- Primarily using Xilinx devices
- Desire flexibility to switch if needed in the future
- Avoid risk of having to change methodology and flow



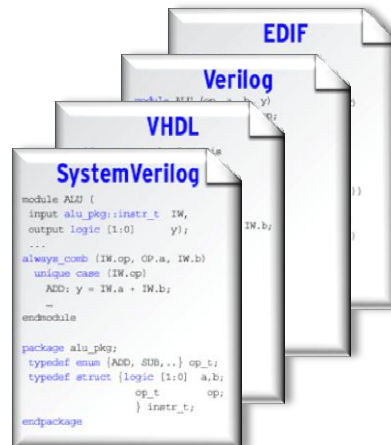
Solution

- Current vendor-independent (Precision) methodology is reliable
- Reduces the impact of changing vendors if the need arises
- Excellent Technical Support

“We want to minimize the impact to our methodology in case of an FPGA vendor change”

Standard Language Support

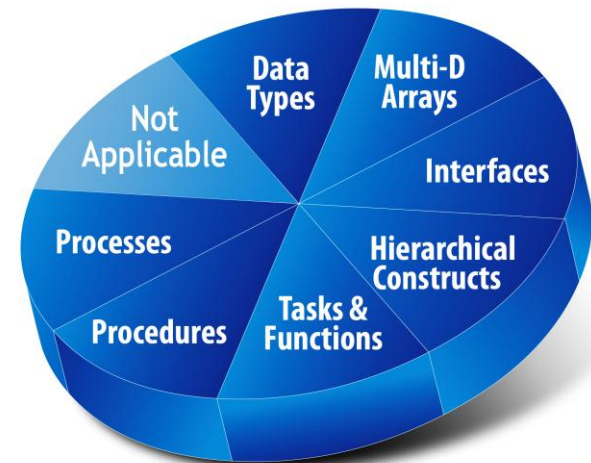
- Mixed Language Support Leadership
 - Verilog, SystemVerilog, VHDL 2008, EDIF
 - Support for mixed language design input
- Synopsys Design Constraint (SDC) Support
 - ASIC Industry standard for timing constraint



Precision SystemVerilog Coverage

SystemVerilog Category	Precision
Literal Values	✓
Data Types	✓
Arrays constructs	✓
Data declaration	✓
Attributes	✓
Operators & Expressions	✓
Procedural Statements, Control Flow	✓
Process	✓
Tasks/functions	✓
Hierarchical constructs	✓
Interfaces	✓
Parameter Declaration Syntax	✓
System Tasks, System Functions	✓

Precision SystemVerilog Support



Precision is the Industry leader in SystemVerilog Support

SystemVerilog Success Story

Video-Processing IC Company

Requirements

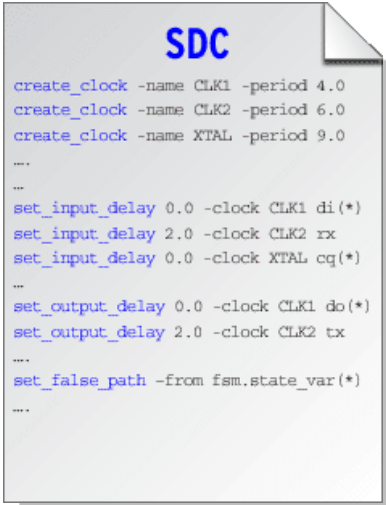
- Tool that supports mixed SystemVerilog, VHDL, Verilog
- ASIC Prototyping Support
- Meet QoR goals for video processing IC (HDTV application)
- SDC Constraint Support

Results

- Precision showed better SystemVerilog support than competition
- Met performance and area goals
- Successfully mapped next generation design with Precision

Standard Constraints Input to Precision

- Synopsys® Design Constraints (SDC)
 - Industry standard
 - Common across ASIC & FPGA synthesis flows
- Precision's interaction with SDC
 - Import existing SDC files
 - Enter constraints through GUI → generate SDC file to be used in future runs
 - Precision automatically generates FPGA constraint file for place-and-route tool

A screenshot of an SDC file with a blue title 'SDC' and a folded corner. The file contains several lines of text in a monospaced font, with some lines highlighted in blue. The text includes commands for creating clocks and setting delays.

```
SDC
create_clock -name CLK1 -period 4.0
create_clock -name CLK2 -period 6.0
create_clock -name XTAL -period 9.0
...
set_input_delay 0.0 -clock CLK1 di(*)
set_input_delay 2.0 -clock CLK2 rx
set_input_delay 0.0 -clock XTAL cq(*)
...
set_output_delay 0.0 -clock CLK1 do(*)
set_output_delay 2.0 -clock CLK2 tx
...
set_false_path -from fsm.state_var(*)
...
```

Precise-IP™, IP Encryption: Vendor Independent IP Platform

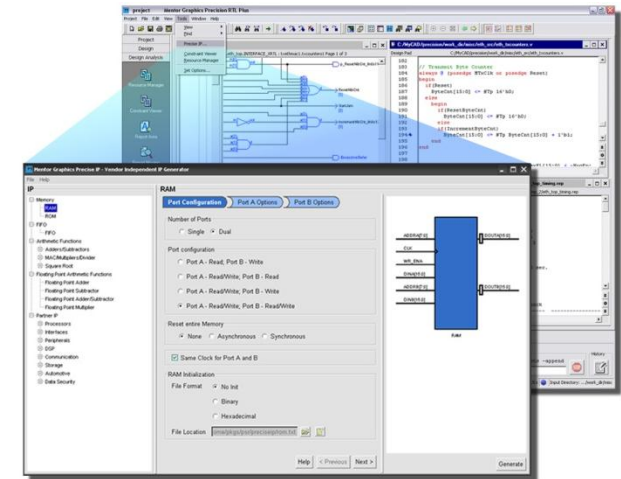
■ Precise-IP

- Vendor Independent IP Platform
- Expanded library of configurable cores
- Re-target IP to different FPGAs
- Out-of-the-box quality-of-results
- Over 70 certified cores from leading IP vendors

■ Precise-Encrypt

- Vendor independent encryption
- Synthesize encrypted IP (IEEE P1735 draft)
- Share IP securely with teams, vendors, partners

Precise-IP™ Platform

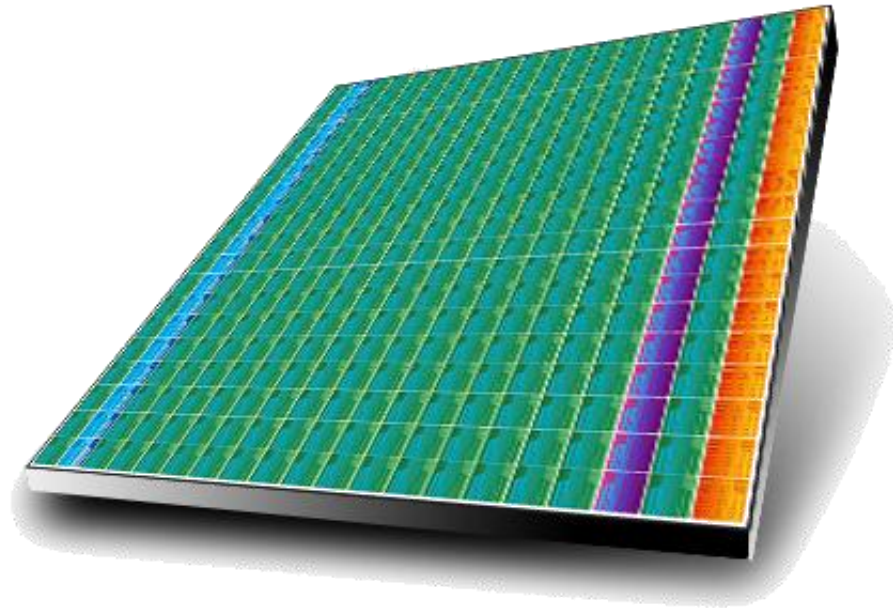


IP Vendors



Precision RTL Plus Synthesis Flow

- Design Import
 - Standard language support
 - Vendor Independence
 - Precise-IP
- Synthesize & Optimize
 - Out-of-the-box Quality of Results
 - Physical synthesis
 - ASIC Prototyping Optimizations
- Design Closure
 - Award-winning analysis & debug
 - Incremental flows
- Industry Specific Capabilities
 - Low Power
 - Mil-Aero & Safety Critical
 - Rad-Tolerant



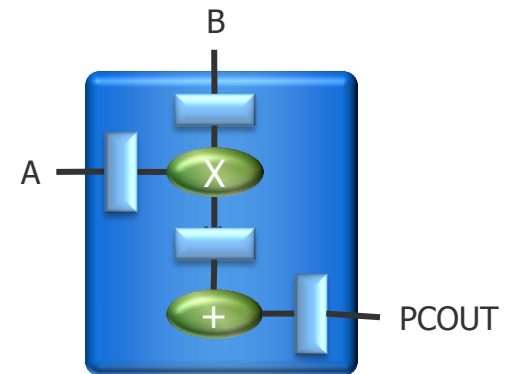
Out-of-the-Box Quality of Results (RTL Synthesis)

- Advanced technology-independent inference
 - Memories, DSP elements, operators, shifters
- Advanced optimizations
 - Resource sharing
 - Retiming
 - FSM encoding
 - I/O mapping

Incremental Flows

DSP

```
always @(posedge clk)
begin
    mult0_result <= A * B;
    PCOUT <= mult0_result +
    PCIN;
end
```

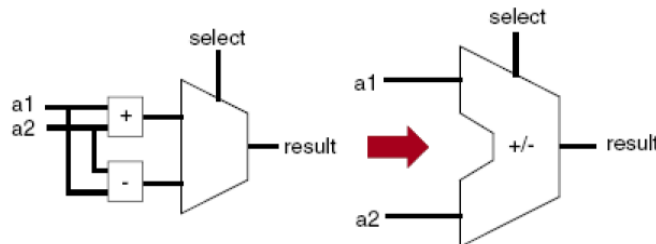


DSP48 SLICE

Technology Independent Inference

```
If (select = '1') THEN
    result <= a1 + a2;
ELSE
    result <= a1 - a2;
END IF;
```

VHDL Code



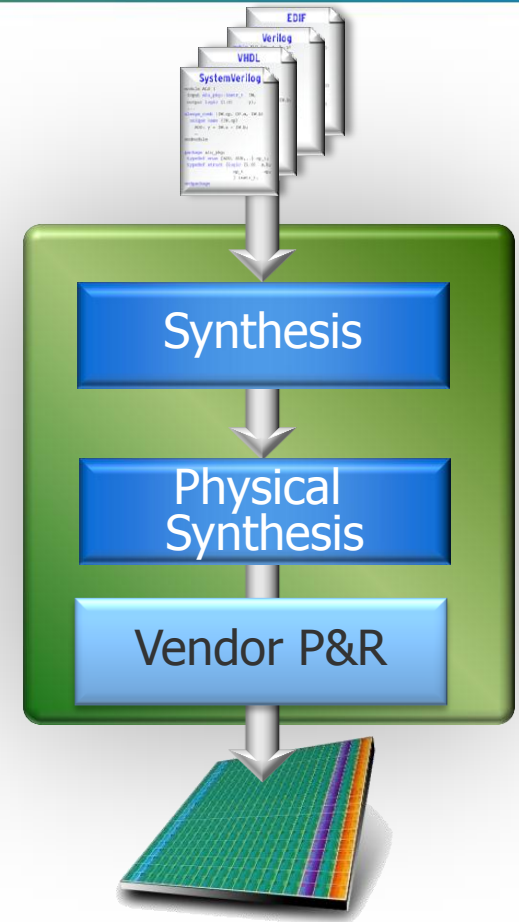
Before Resource Sharing

After Resource Sharing

Resource Sharing

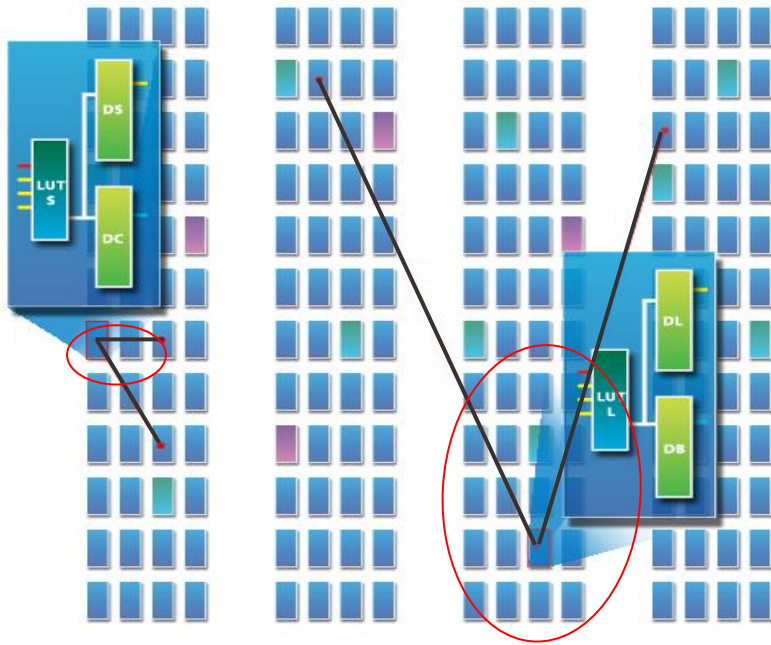
Physical Synthesis

- Pre-place-and-route physical synthesis
- Average 10% performance improvement
 - Typically ranges from 5% to 40%
- Industry's only multi-vendor physical synthesis
 - 26 FPGA device families supported (2010a)
 - 3x the nearest competitor
 - Actel, Altera, Lattice, Xilinx
- Push-button so every designer benefits
- Pre-P&R physical creates optimized netlist
 - Fewer P&R iterations
 - Improved netlist → faster placement & routing



Reach design goals faster with fewer iterations

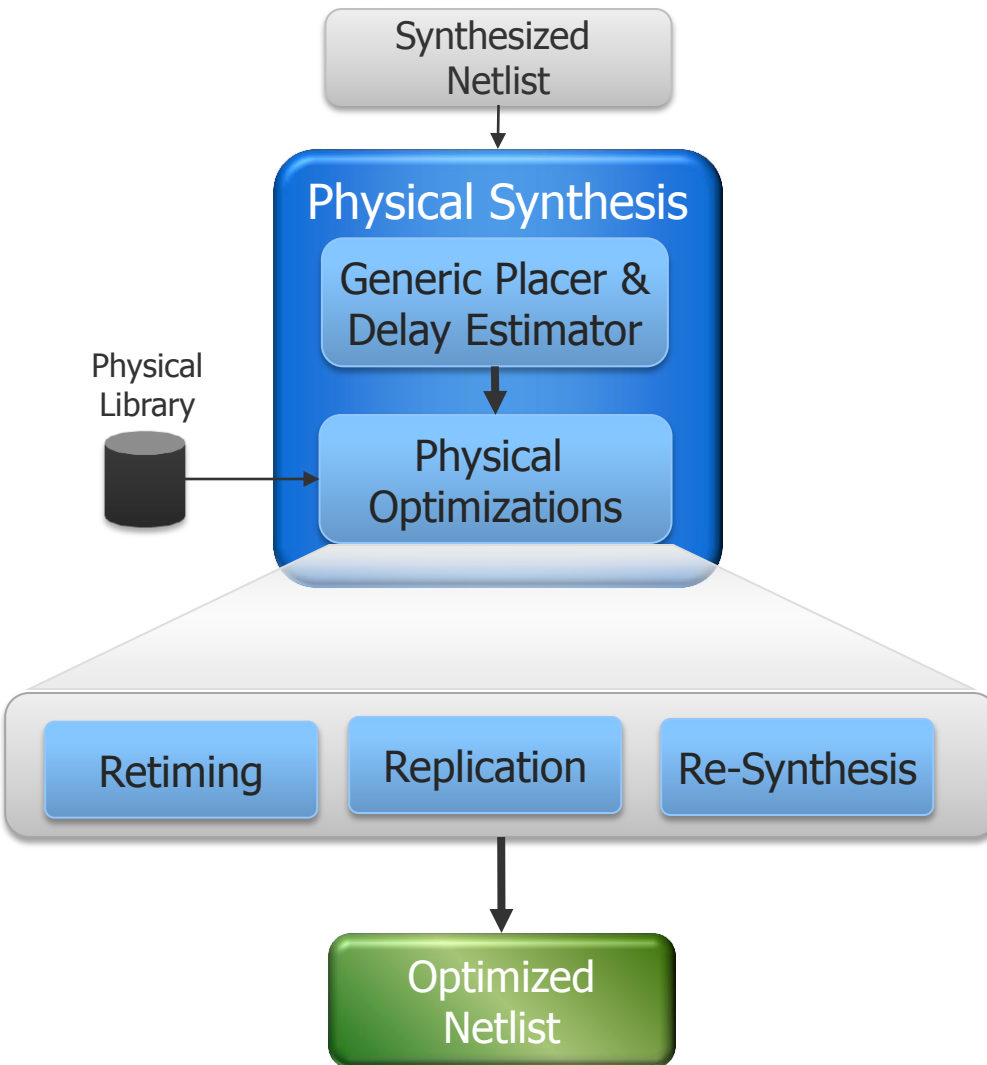
Limitations of RTL Synthesis



Traditional wire load delay models treat these interconnect delays equally

- Standard RTL synthesis bases interconnect delay on vendor-provided wire-load-models
- As technology node increases, interconnect delay dominates total path delay
- Calculations can vary widely from actual physical delays (due to final placement)

How Physical Synthesis Works



- Advanced Delay Estimation
 - Estimates location
 - Estimates routing resources
 - More accurate net delays
 - Identifies Critical Paths
- Netlist Optimization
 - Retiming, Replication, Re-synthesis
- No placement sent to P&R
 - No DRC/packing violations
 - Maximum flexibility for P&R

Precision Physical Synthesis Complements Vendor Physical Synthesis

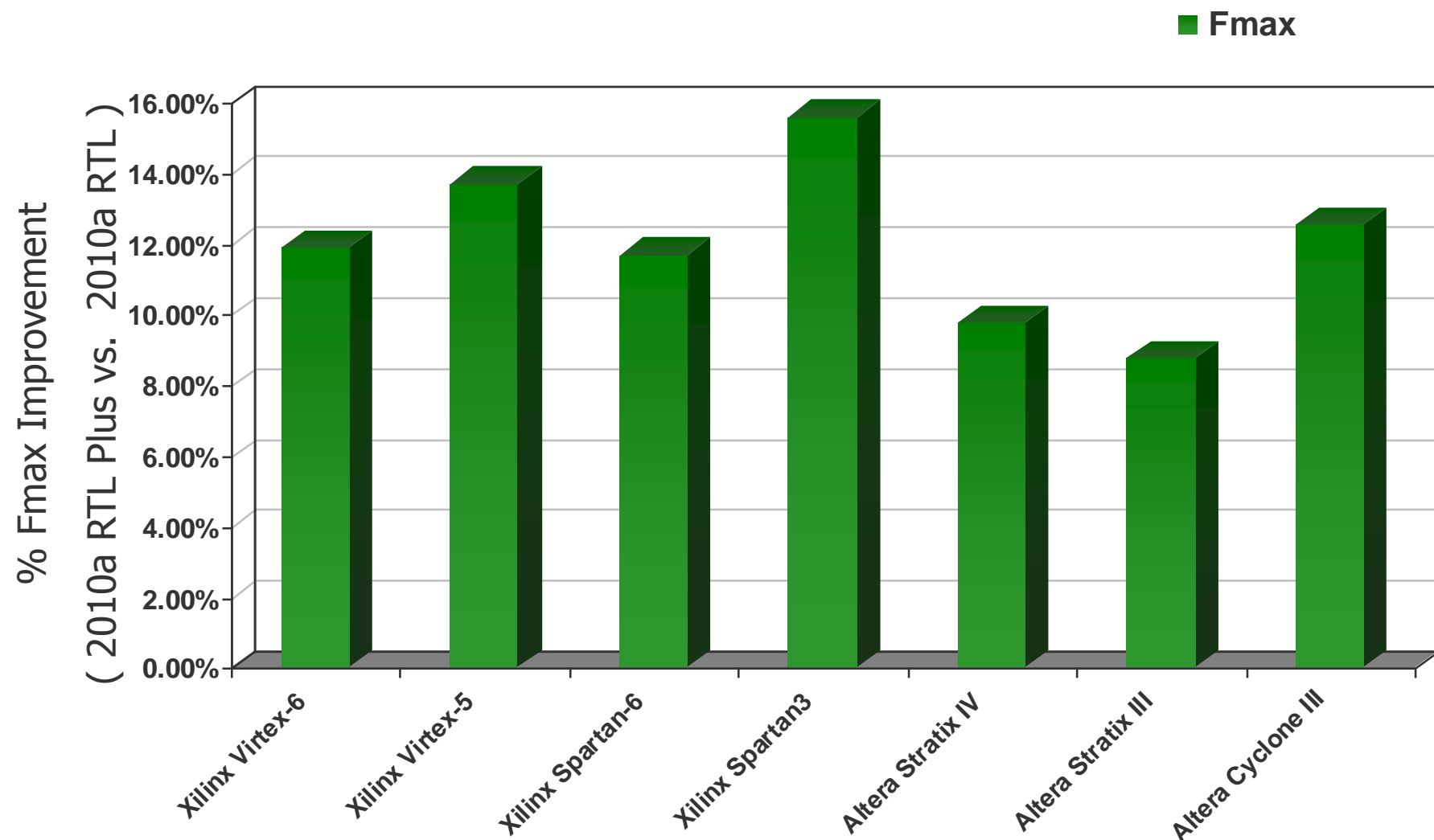
- Precision physical synthesis creates optimized netlist
 - Do not restrict vendor P&R with **locked placement**
- Vendor physical synthesis and P&R are unrestricted to produce best netlist
 - Free to perform additional physical optimizations
- Precision physical synthesis followed by vendor physical synthesis produces superior results



Physical Synthesis Observed Data

Application	Target Device	RTL Fmax (MHz)	RTL Plus Fmax (MHz)	Improvement
Ethernet	Virtex-6	195	249	27%
DSP	Virtex-6	130	192	47%
Microprocessor	Virtex-6	122	150	23%
Microprocessor	Virtex-5	223	329	47%
Communication	Virtex-5	83	93	13%
Serial Interface	Virtex-5	204	279	36%
Data Compression	Virtex-5	380	438	15%
Consumer	Virtex-4	56	62	9%
Consumer	Virtex-4	54	61	11%
DSP	Spartan-6	56	74	33%
Microprocessor	Spartan-6	64	78	21%
DSP	Spartan-3	52	56	7%
Serial Interface	Stratix-III	154	211	37%
DSP	Stratix-III	98	128	26%
Consumer	Cyclone-III	138	160	16%
DSP	Cyclone-III	182	250	38%

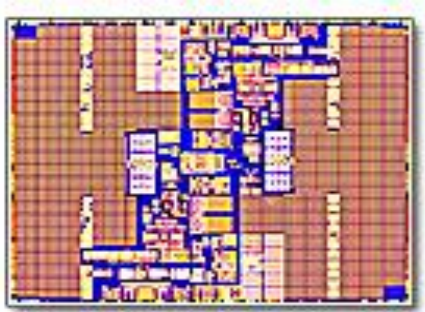
Physical Synthesis Improvement over RTL Synthesis



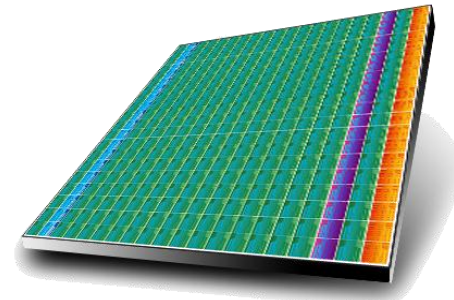
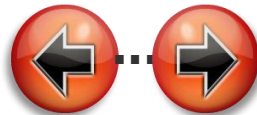
* Based on Mentor Test Suite (100+ Customer Designs), RTL Plus physical synthesis may cause 2-5% area degradation

ASIC Prototyping Support

- Precision eases the transition from ASIC to FPGA
 - Use the same HDL and constraint syntax to target ASIC and FPGA
- Synopsys Design Constraints – ASIC industry standard
- Advanced gated clock conversion
 - Automatically converted during synthesis
- Synopsys Designware Component Support
 - Optimal synthesis implementation of DesignWare components



ASIC to FPGA



Precision Gated Clock Conversion

■ Benefits

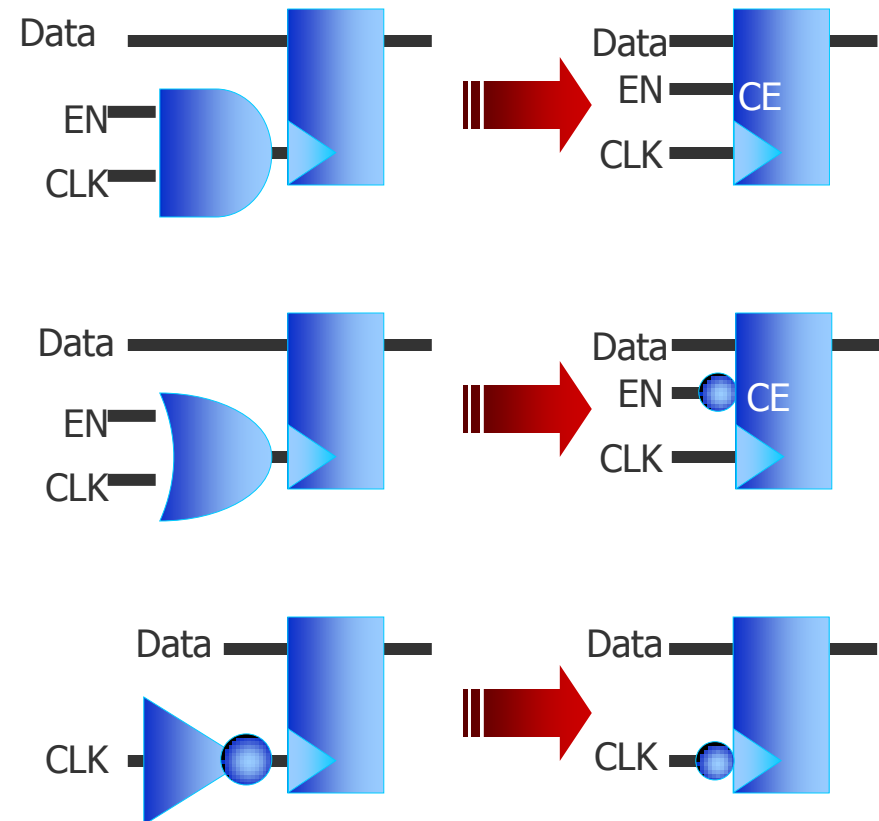
- Minimize clock skews in FPGAs
- Use same RTL ASIC source files

■ Automatic Conversion

- Re-route to clock enables (when applicable)
- Re-implement with functional equivalence

■ Support

- Flip-Flops, latches
- Counters, multipliers, DSPs RAM
- Generated clocks (clock dividers)
- Multiplexed clocks
- Inverter-gated clocks



Precision DesignWare® Support

```
//VERILOG
// No includes needed

DW01_add #(32) myadder (.A (a), .B
(b), .CI (c_in), .SUM (sum), CO
(c_out));

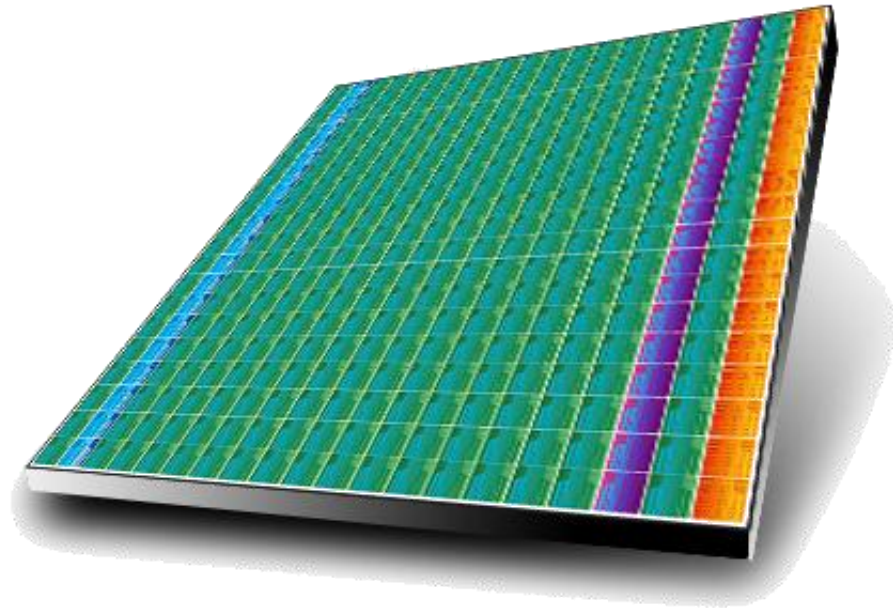
-- VHDL
-- LIBRARY DECLARATION
LIBRARY DW01;
USE DW01.DW01_COMPONENTS.ALL;

component DW01_add
-- Description
end component;
```

- DW support enables HDL reuse
- Minimal user intervention
 - Transparent flow
 - Exchanges with optimal and compatible implementations
 - Black boxes unsupported DW
- Verilog Design Flow
 - No need to include components
- VHDL Design Flow
 - Include DW components in library declarations

Precision RTL Plus Synthesis Flow

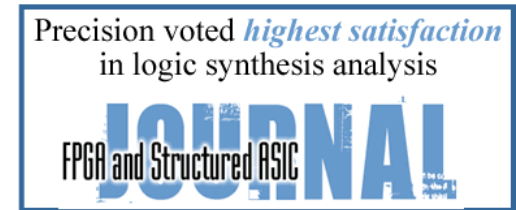
- Design Import
 - Standard language support
 - Vendor Independence
- Synthesize & Optimize
 - Out-of-the-box Quality of Results
 - Physical synthesis
 - ASIC Prototyping Optimizations
- Design Closure
 - Award-winning analysis & debug
 - Incremental flows
- Industry Specific Capabilities
 - Low Power
 - Mil-Aero & Safety Critical
 - Rad-Tolerant



Award-winning Analysis and Debug

■ Design Analysis

- Comprehensive messaging and warnings
- Cross-probing between HDL & Schematics

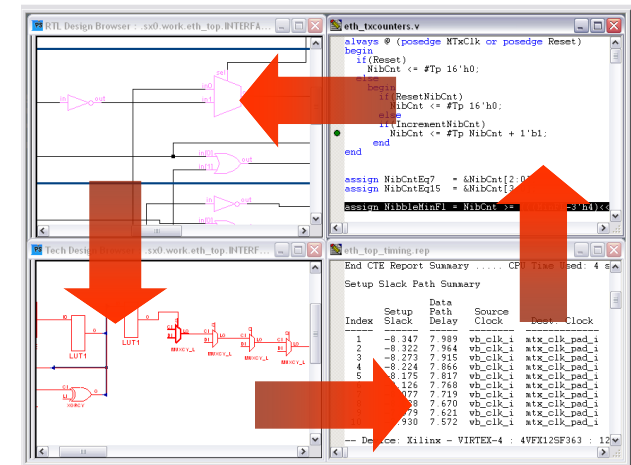


■ Constraint Analysis

- Missing Constraint report
- Clock Domain Crossing report
- What if analysis without re-synthesis

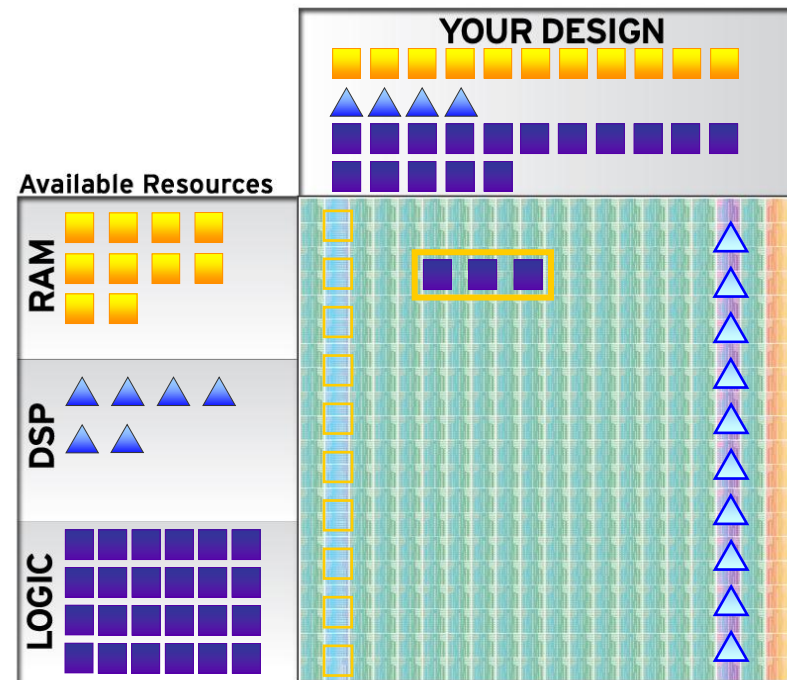
■ QoR Analysis

- Critical-path viewing for timing analysis
- Graphical resource analysis and control*



Resource Analysis & Control

- Make efficient use of FPGA embedded resources
- Identify available resources
- Identify required resources for design
- Control mapping for Area or Timing
- Cross-probe to HDL and schematics
- Budget resources for team-based design
- Intuitive, easy-to-use interface



Control resources to meet timing & area goals efficiently

Resource Manager User Interface

The screenshot shows the 'Resource Manager' window with tabs for BLOCKRAM, DSP, and LUT. The DSP tab is active, showing summary statistics and a table of resource instances.

Resource Types: Points to the tabs (BLOCKRAM, DSP, LUT).

Total Available: Points to the 'Total Resources' field, which shows 32.

Operator Instances: Points to the 'Instance' column in the table.

Operator Functions: Points to the 'Operator Type' column in the table.

Resource Choices: Points to the 'User Assignment' dropdown menu in the table.

Current Assignment: Points to the 'Current Assignment' column in the table.

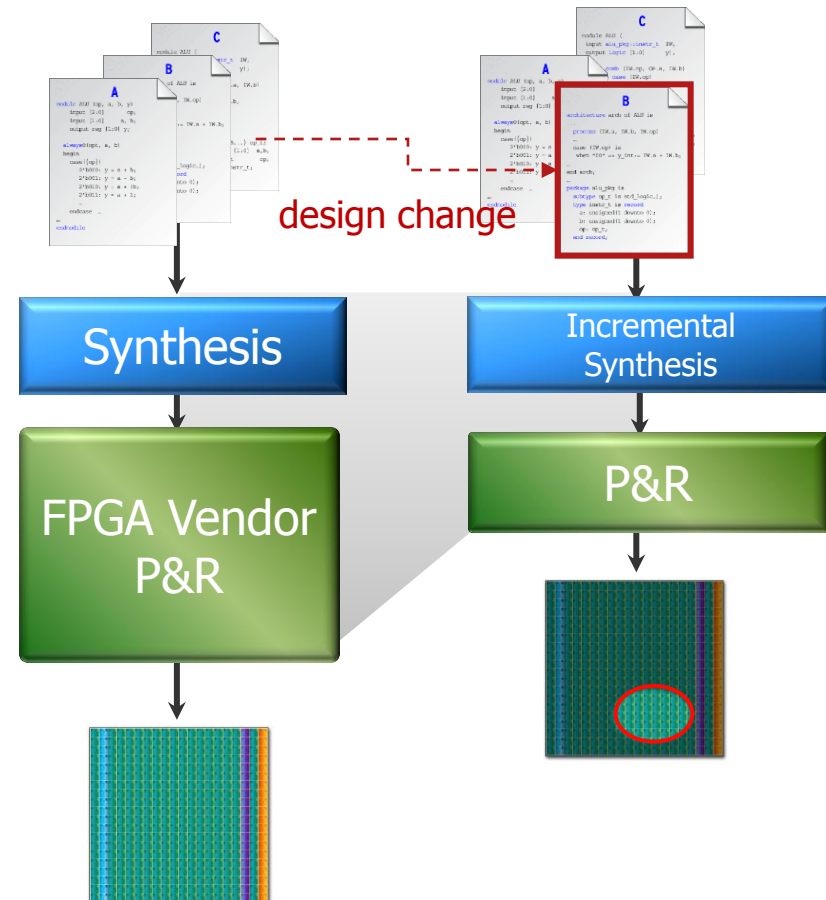
Instance	Operator Type	Size	User Assignment	Current Assignment	Resource Estimate
z_add34_Oi5	add	34	<Auto-Selected>	DSP	1
z1_add33_Oi4	add	33	<Auto-Selected>	LUT: Carry chain	---
z_add32_Oi3	add	32	DSP	DSP	1
z1_add32_Oi2	add	32	DSP	LUT: Carry chain	---
z0_madd25_1	modgen_multadd	25	LUT	DSP	1
p0_madd23_0	modgen_multadd	23	<Auto-Selected>	DSP	1
ix5	modgen_multadd	25	<Auto-Selected>	DSP	1
z0_madd24_Oi1	modgen_multadd	24	<Auto-Selected>	DSP	1

"... This tool would be tremendously helpful in allocating resources ... For a designer, he could quickly check on the critical parts of his design to verify that it was being implemented in the best manner and to have the ability to change if necessary."

—Leading Mobile Chipset Company

Automatic Incremental Synthesis

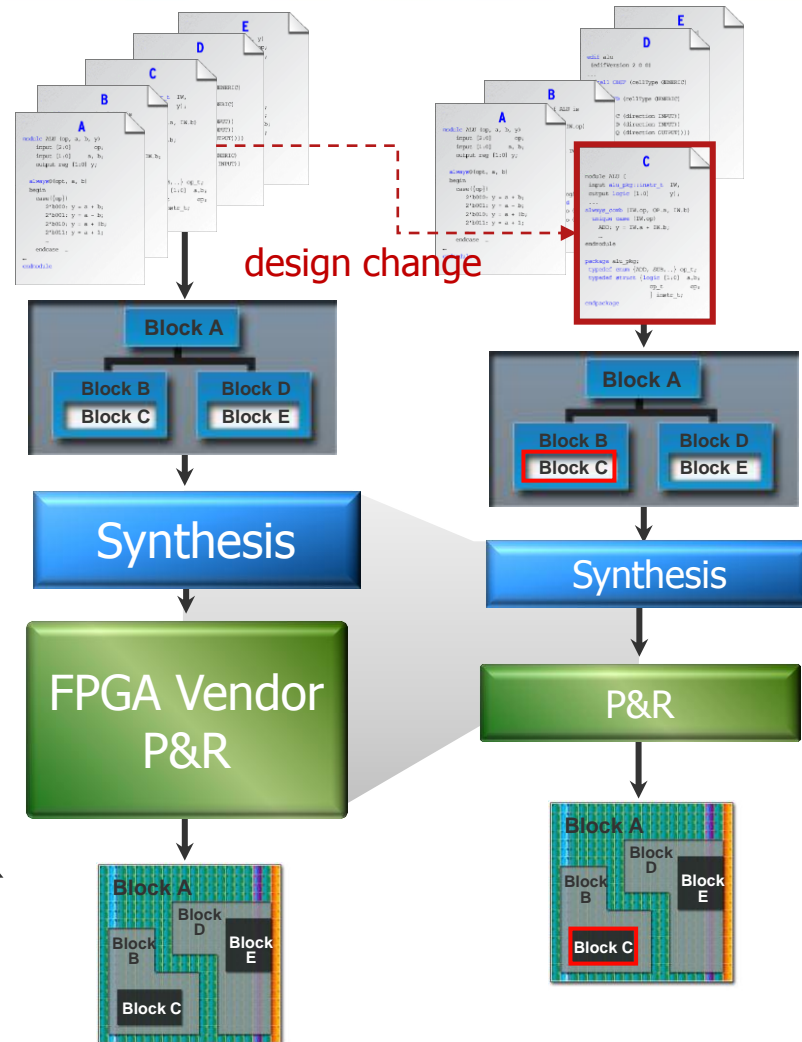
- Up to 60% runtime savings
 - Design, change dependent
- Industry's only automatic incremental synthesis
 - No partitioning or prior planning
 - Maintains QoR with cross-hierarchy optimizations
 - Based on real changes in parse tree
- All FPGA families supported



"Precision RTL Plus is the first tool to support a truly **push-button FPGA incremental synthesis** enabling RTL designers to use top down approach without sacrificing quality of results and runtime."
- Leading Satellite Communications Company

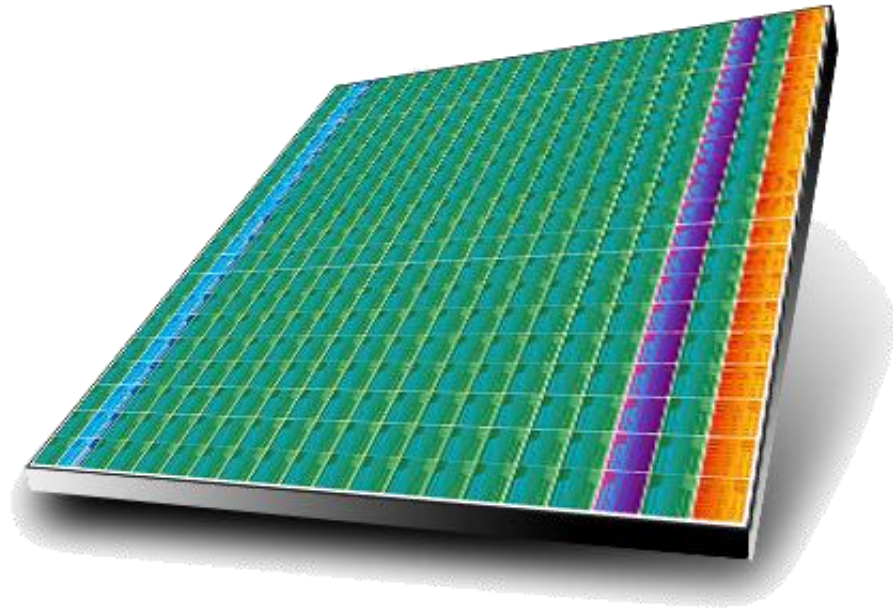
Partition-Based Incremental Synthesis & P&R

- Up to 6x synthesis runtime savings
 - 100% predictability for unchanged blocks
 - Design, change dependent
- Divide and conquer for team-based design
 - Localized effects of changes
 - Preserves unaffected partitions
 - Automatic incremental synthesis within changed partitions
- Supports Altera & Xilinx incremental P&R



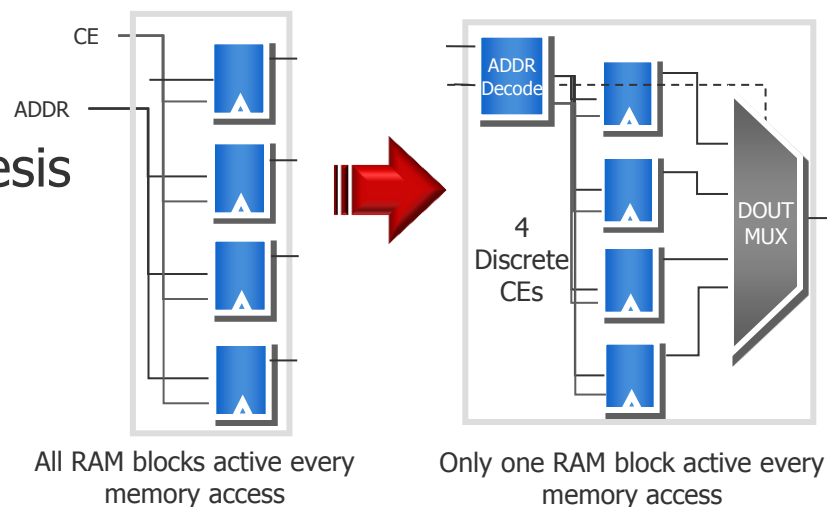
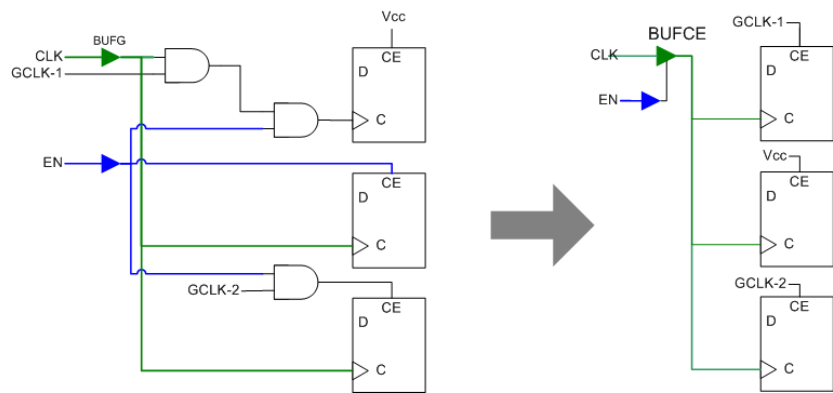
Precision RTL Plus Synthesis Flow

- Design Import
 - Standard language support
 - Vendor Independence
- Synthesize & Optimize
 - Out-of-the-box Quality of Results
 - Physical synthesis
 - ASIC Prototyping Optimizations
- Achieve Design Closure
 - Award-winning analysis & debug
 - Incremental flows
- Industry Specific Capabilities
 - Low Power
 - Mil-Aero & Safety Critical
 - Rad-Tolerant



Low Power Synthesis

- Mapping to enabled clock buffers
 - Minimizes power consumed by clock networks
- Map large RAMs based on address
 - Reduces power per memory access
- Low power register retiming
 - Reduce area & switching activity
- The only vendor independent synthesis offering low power optimizations



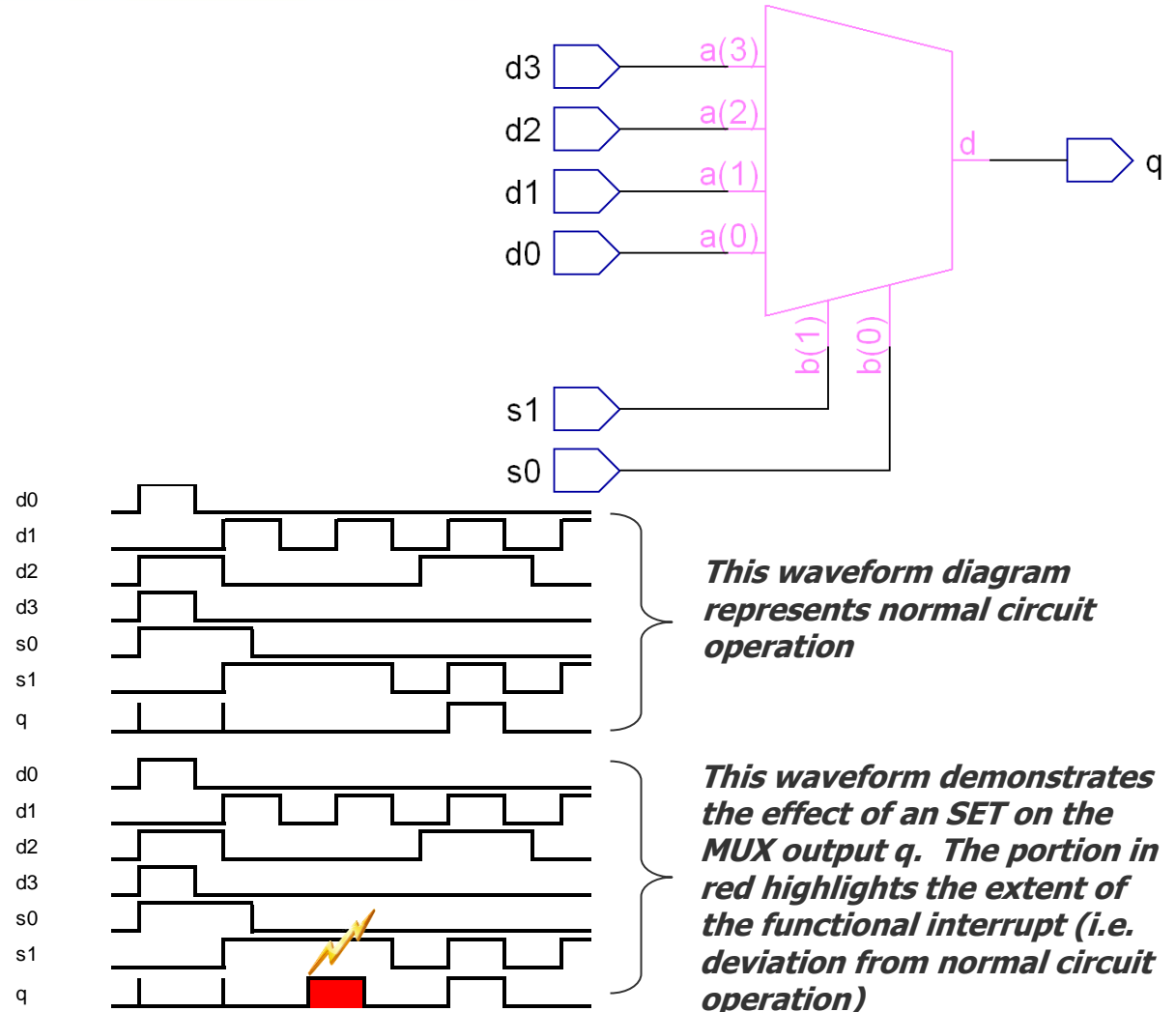
Precision Synthesis for Mil-Aero & Safety Critical Applications

Feature	Benefit
Assured synthesis mode	Enables options needed for easier verification
Logic Equivalence Checking	Integrated flow ensures RTL = implementation
ReqTracer Integration Flow	Requirements traceability
Repeatability	Extensive internal testing to ensure repeatability



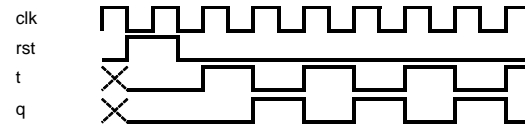
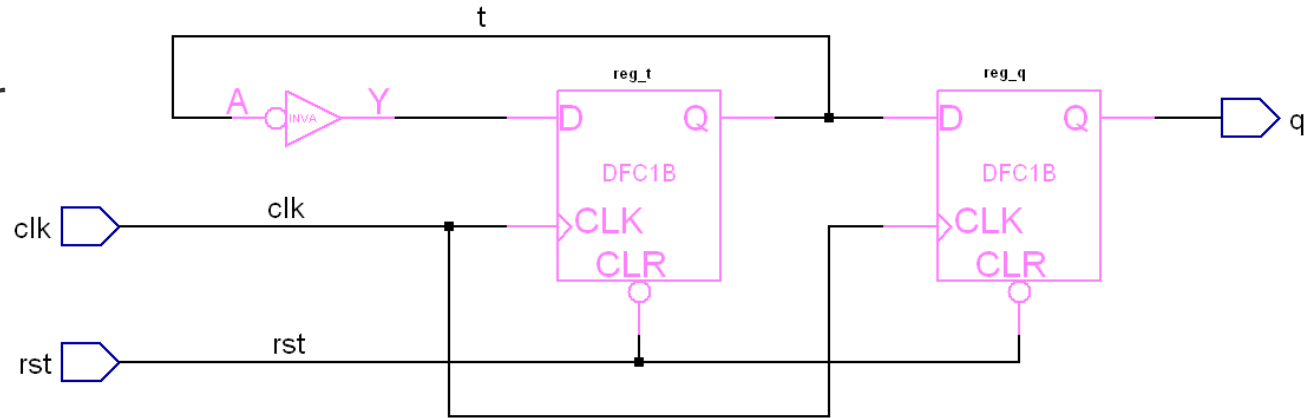
Single Event Transient

- Temporary change in output of combinatorial logic caused by ions or electro-magnetic radiation
- Corrects itself within several nanoseconds
- Far more problematic at higher operating frequencies
 - Higher probability of the SET's error window spanning a rising clock edge

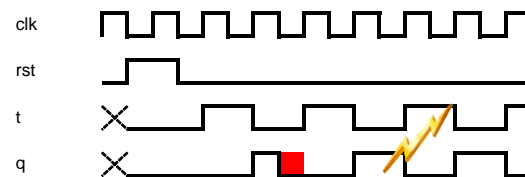


Single Event Upset

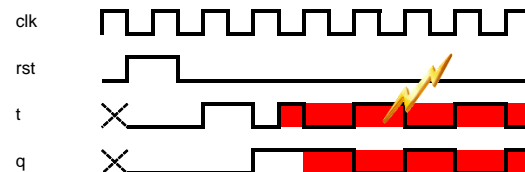
- Change of state caused by ions or electro-magnetic radiation
- Affects storage elements
- A fault retention path is a path with feedback where the fault arising from an upset can be retained in the circuit



This waveform diagram represents normal circuit operation



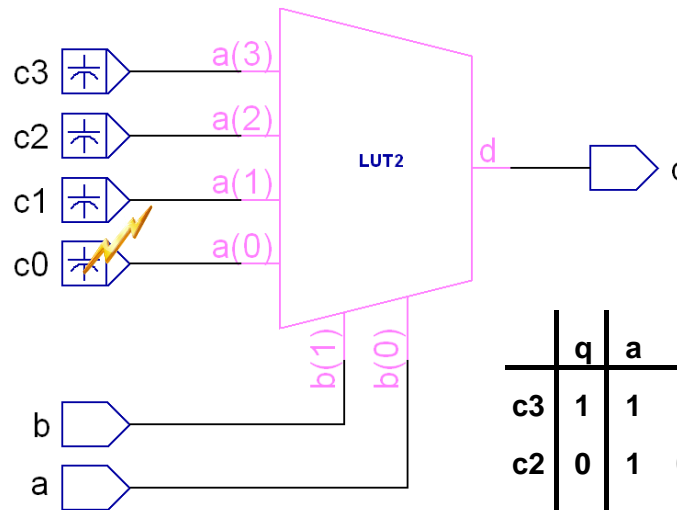
This waveform demonstrates the effect of an SEU on register reg_q, not in a fault retention path. The portion in red highlights the extent of the functional interrupt (i.e. deviation from normal circuit operation)



This waveform demonstrates the effect of an SEU on register reg_t, in a fault retention path. The portion in red highlights the extent of the functional interrupt (i.e. deviation from normal circuit operation)

Configuration Upset (from SEU)

- Change of LUT function or routing caused by ions or electromagnetic radiation striking a configuration bit
- Not corrected until configuration SRAM refreshed
- “Background scrubbing” refers to the process of constantly checking the configuration SRAM to correct SEUs



	q	a	b
c3	1	1	1
c2	0	1	0
c1	0	0	1
c0	0	0	0

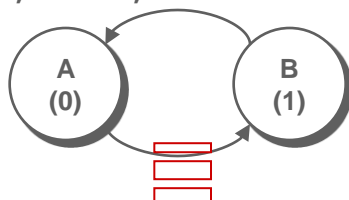
This truth table shows the normal operation of the lookup table. The LUT implements an AND2 function

	q	a	b
c3	1	1	1
c2	0	1	0
c1	0	0	1
c0	1	0	0

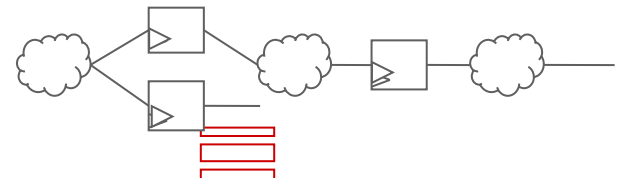
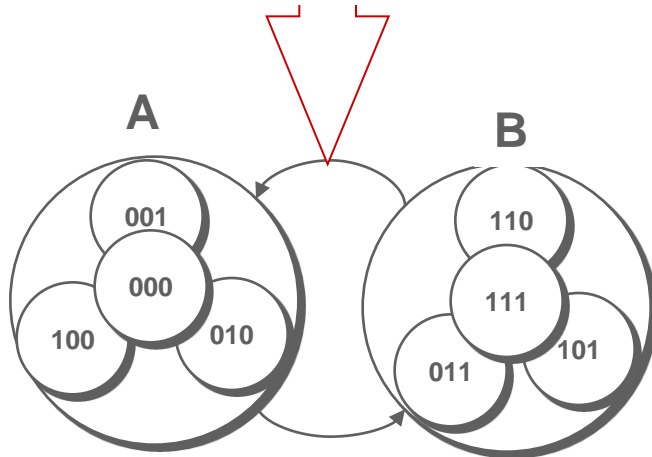
This truth table shows the modified function of the lookup table resulting from an SEU on configuration bit c0. The LUT now implements an XNOR2 function

Radiation Effect Mitigation (REM)

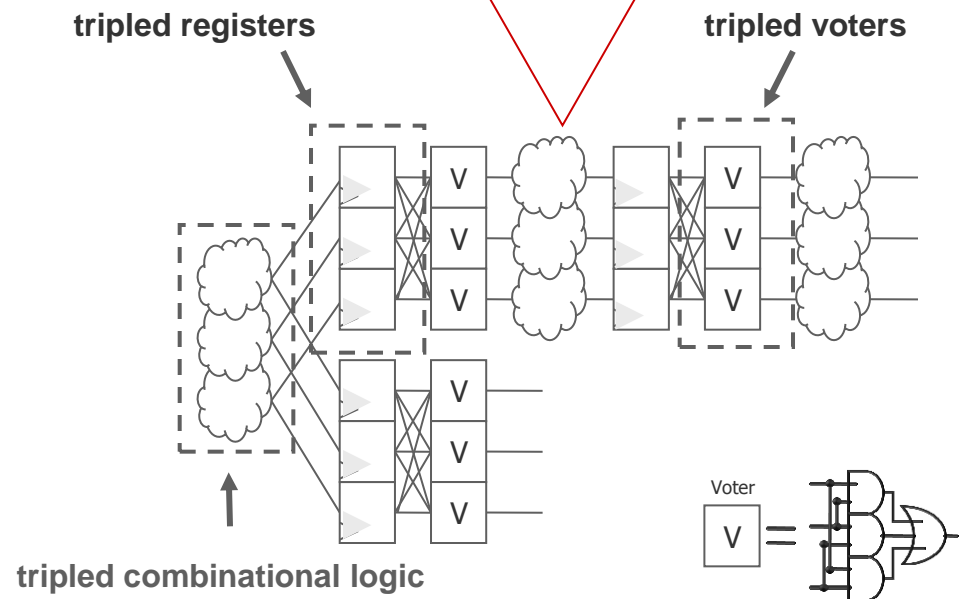
- Synthesis optimizations for safer operation under radiation
 - Safe & Fault Tolerant FSM, Multi-mode, multi Vendor TMR
- Support for multiple device vendors
 - Xilinx, Actel, more to come



Fault-Tolerant FSM



Triple Module Redundancy (TMR)



Fault Tolerant FSM

Hamming Distance 3

What

- FSM corrects SEU without interruption
- Adds k parity bits² to state vector

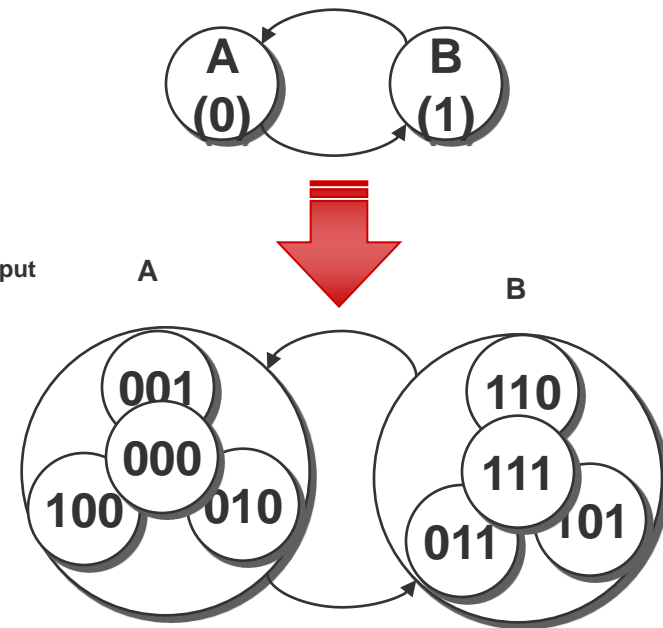
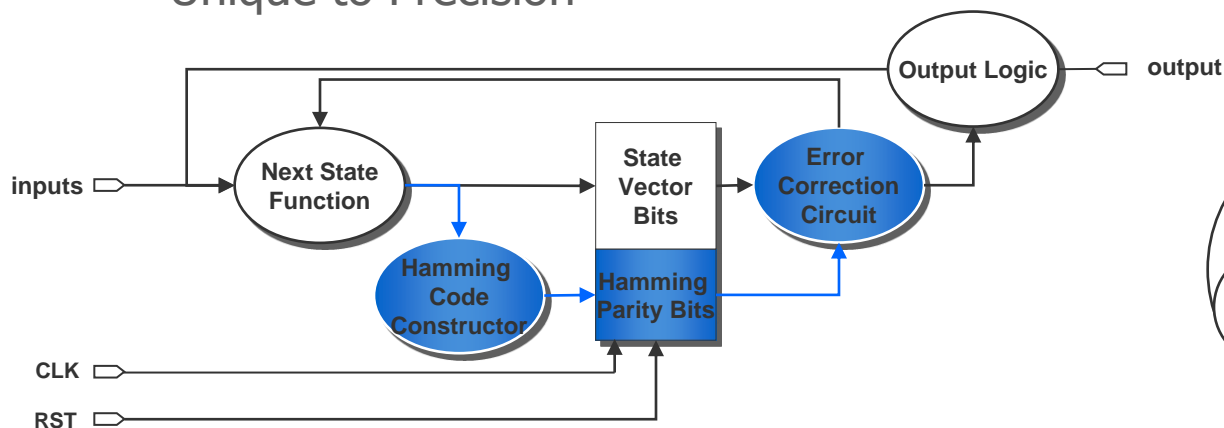
Benefit:

- Allows for normal operation after an SEU
- Applicable to all encoding schemes

Mentor Advantage

- Unique to Precision

State	Before	With parity bits	States map to same destination state
A	0	000	001, 010, 100
B	1	111	110, 101, 011

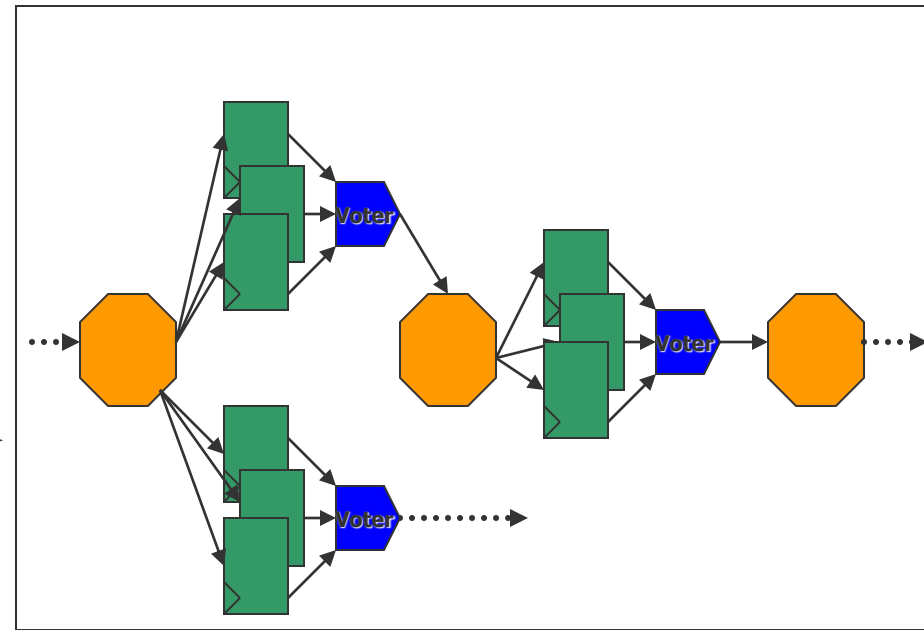
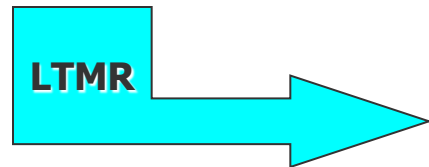
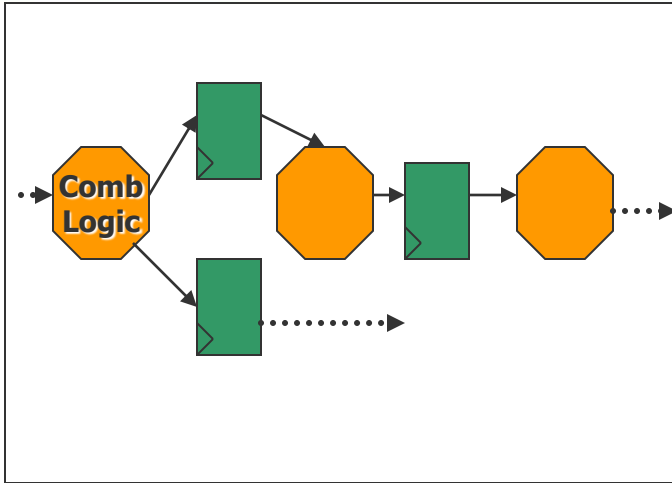


TMR Strategies – Local TMR

- Local TMR (LTMR)
 - Triplicate sequential elements only, and majority vote the outputs
 - Flip-flops, shift registers, block RAMs, and sequential DSPs
 - The input data, control signals and clock will be shared by the triplicated flops
 - Reduces SEE occurrence to frequency dependent SET capture; clock trees, global routes and IOs are still susceptible

[1] M. Berg, "Design for Radiation Effects, "Invited Talk Presented 2008 at Military and Aerospace Programmable Logic Design, MAPLD, Annapolis, MD, September. 2008

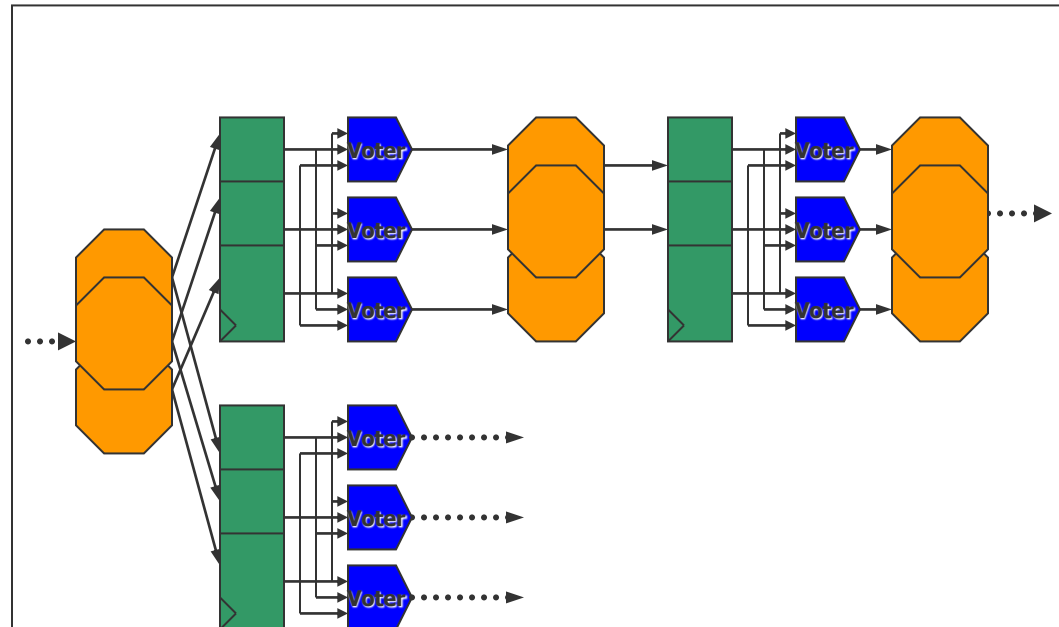
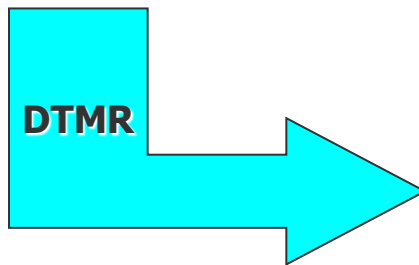
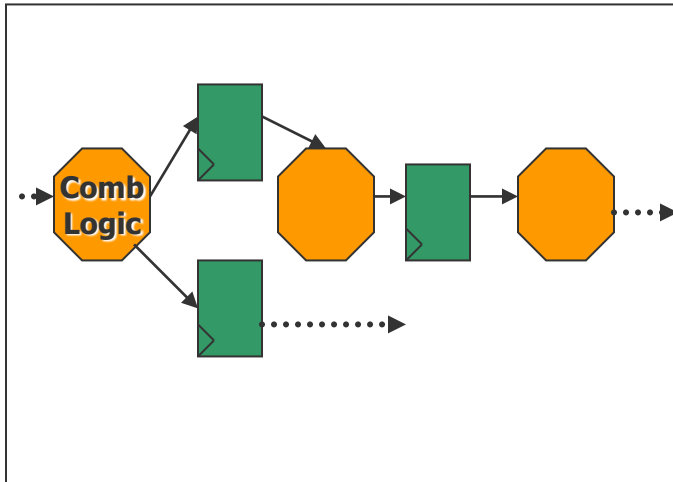
Local TMR illustrated



TMR Strategies – Distributed TMR

- Distributed TMR (DTMR)
 - Apply TMR on sequential and combinational logic
 - Triplicate sequential and combinatorial logic; global routes and I/O are not triplicated
 - Vote out the triplicated logic just after the sequential elements
 - Triplicate the majority voting circuit as well to protect SET effects on the voting circuit
 - Reduces SEE occurrence; clock trees, global routes and IOs are still susceptible
 - Preferrable scheme for SEU and SET protection of technologies with hardened clock trees

Distributed TMR illustrated

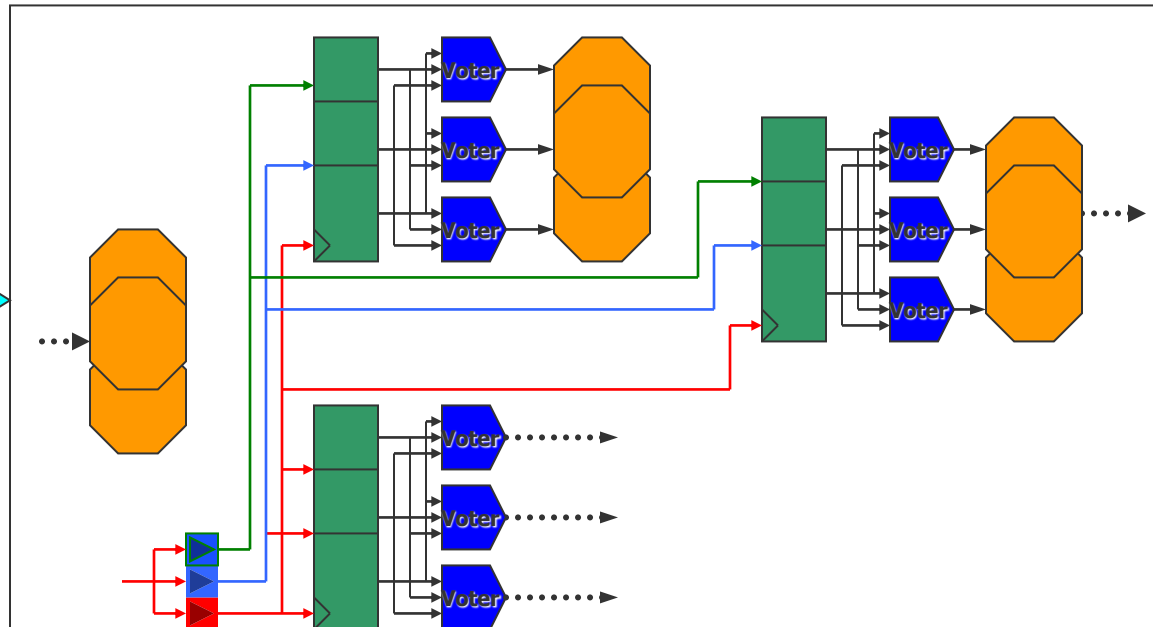
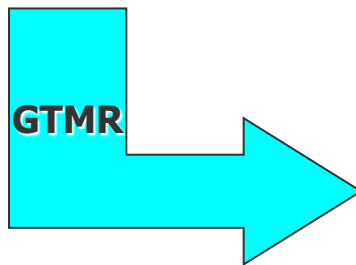
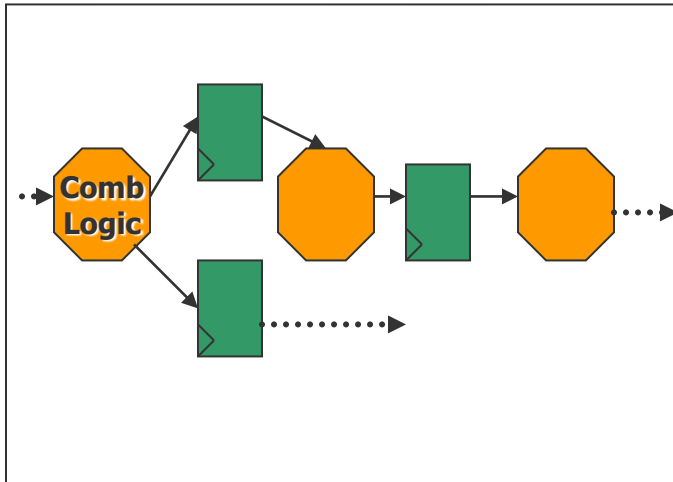


TMR Strategies – Global TMR

■ Global TMR (GTMR)

- Apply TMR on the entire design including global buffers
 - Triplicate the sequential elements, combinational logic, voters and the global buffers
 - Voters converge the triplicated flop outputs at clock and control domain crossovers
- This gives very high level of radiation protection
- This scheme requires that the triplicated global lines have minimum skew between them
- Preferred scheme for commercial SRAM based FPGAs

Global TMR illustrated



Precision Synthesis Family (Feature Sub-Set)

	Precision RTL	Precision RTL Plus	Precision Rad-Tolerant
FPGA Vendor Independence	✓	✓	✓
Advanced Optimizations	✓	✓	✓
SystemVerilog, VHDL-2008	✓	✓	✓
Precise-IP™	✓	✓	✓
Flows with Catapult, FormalPro, HDL Designer	✓	✓	✓
Flow with I/O Designer		✓	✓
Precise-Encrypt™ (Encryption)		✓	✓
Multi-Vendor Physical Synthesis (pre-P&R)		✓	✓
Low Power Synthesis		✓	✓
Precision- ReqTracer Integration		✓	✓
Assured Synthesis Mode		✓	✓
Safe FSM (Recovery & Detection, Fault-Tolerant)			✓
Multi-Mode, Multi-Vendor TMR			✓

Customer and Partner Testimonials



"Automating TMR logic insertion while allowing the user to select the type of TMR mitigation is very beneficial to an FPGA designer of critical space applications."



"Mentor Graphics is enhancing the FPGA design flow for high reliability... designers ... can use Precision Rad-Tolerant to protect critical data paths in ProASIC®3, IGLOO®, and Fusion® families."



"Altera ... found significant improvement in quality of results ...with Precision RTL Plus"



Precision RTL shows 10-20% average improvement on frequency or area over Synplicity, Xilinx, and Altera



"Mentor Graphics delivers a unique synthesis-based approach to soft error mitigation that complements Xilinx solutions for high reliability applications. "



"Mentor ... led vendors in serving the FPGA Design Community ...including a clear vision of the future."



Precision RTL Plus selected 2007 BEST
"FPGA Synthesis Tool Suite"



"Precision's advanced optimization increased the clock rate by almost 30 %"



Alcatel-Lucent

"Precision RTL helped us ... achieve timing, performance and resource utilization ... this enabled us meet our ... design goals."

Precision Synthesis Advantage

- ✓ Multi-vendor physically aware synthesis
- ✓ Fully automatic incremental synthesis
- ✓ Leading SystemVerilog support
- ✓ Integration with Mentor & Partner Flows
- ✓ Industry specific Capabilities (Low Power, Safety-Critical, Rad-Tolerant)



"The incremental flow, physical optimization, and resource control capabilities of [Precision RTL Plus] are exciting, and this solution should greatly improve our customer's productivity."

- Steve Lass, Senior Director of Software Product Marketing, Xilinx



Questions?





www.mentor.com

RTL through to PCB

