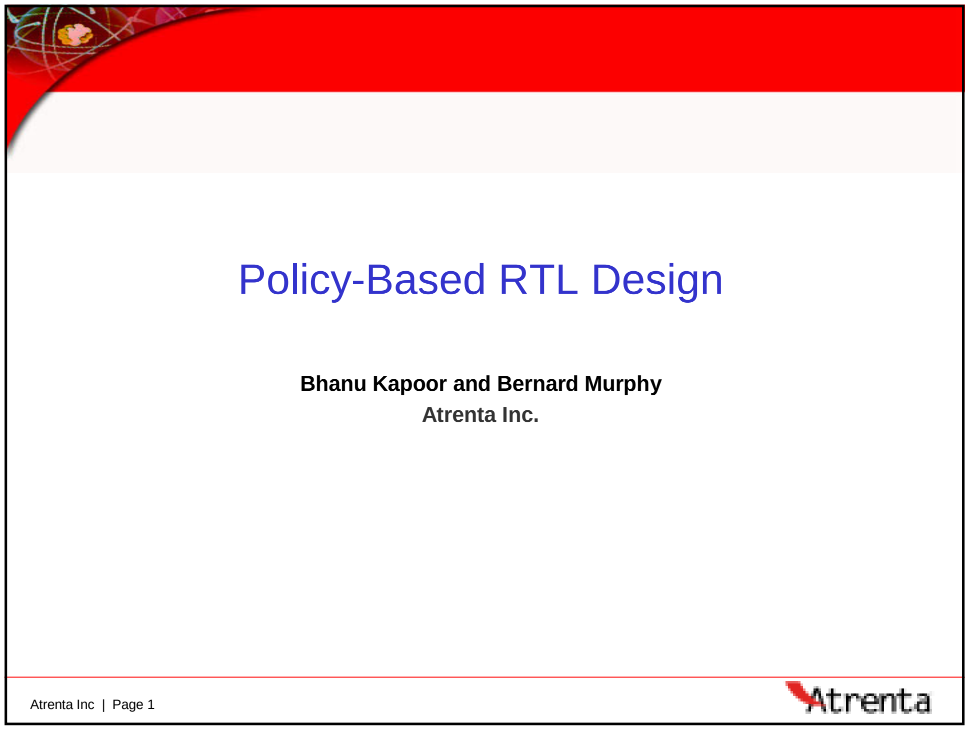
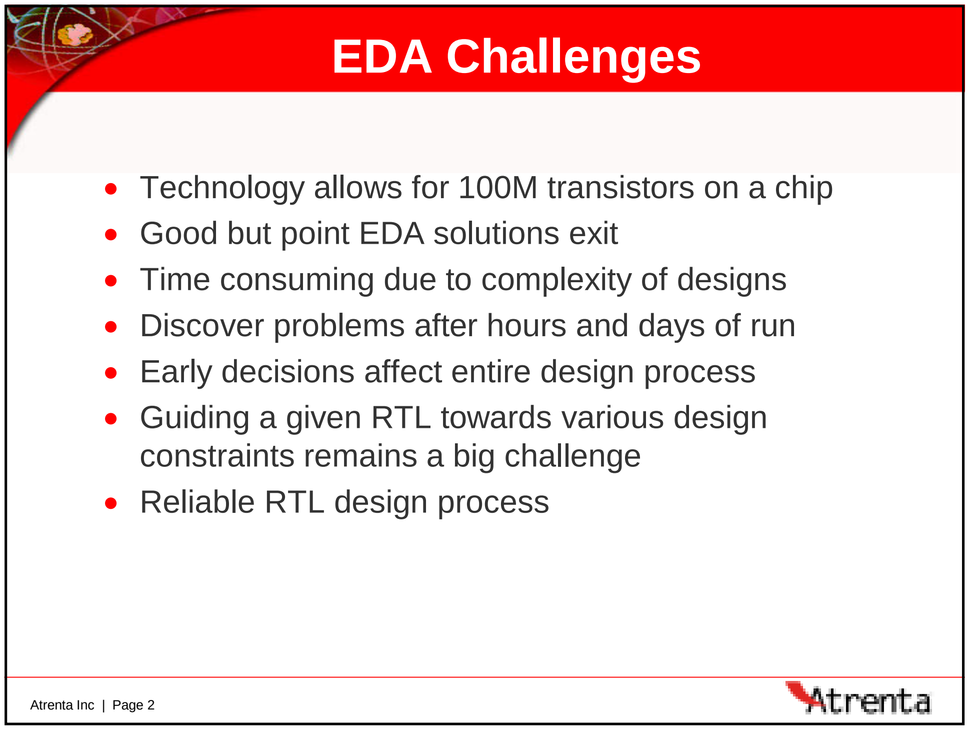


Policy-Based RTL Design

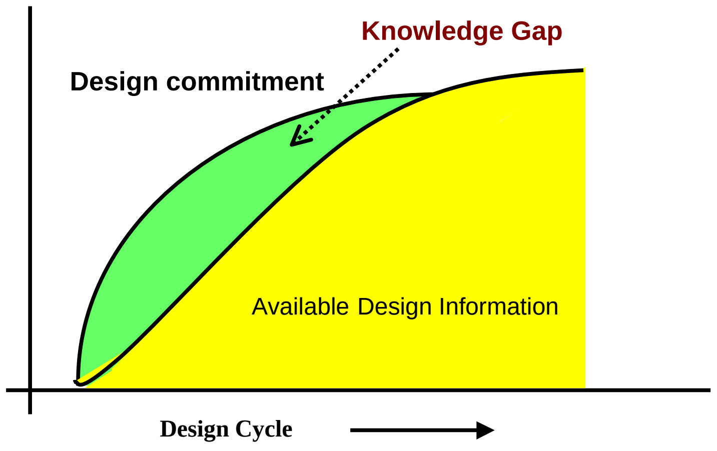
Bhanu Kapoor and Bernard Murphy
Atrenta Inc.



EDA Challenges

- Technology allows for 100M transistors on a chip
- Good but point EDA solutions exit
- Time consuming due to complexity of designs
- Discover problems after hours and days of run
- Early decisions affect entire design process
- Guiding a given RTL towards various design constraints remains a big challenge
- Reliable RTL design process

Challenges in Product Development



Atrenta Inc | Page 3

Atrenta

Policy-Based RTL Design

- Policies that guide the creation of efficient RTL
- Target design needs early in the design cycle
- Conflicting issues known early
- Long simulation and synthesis not always needed
- Disseminate expert knowledge
- Towards Golden RTL

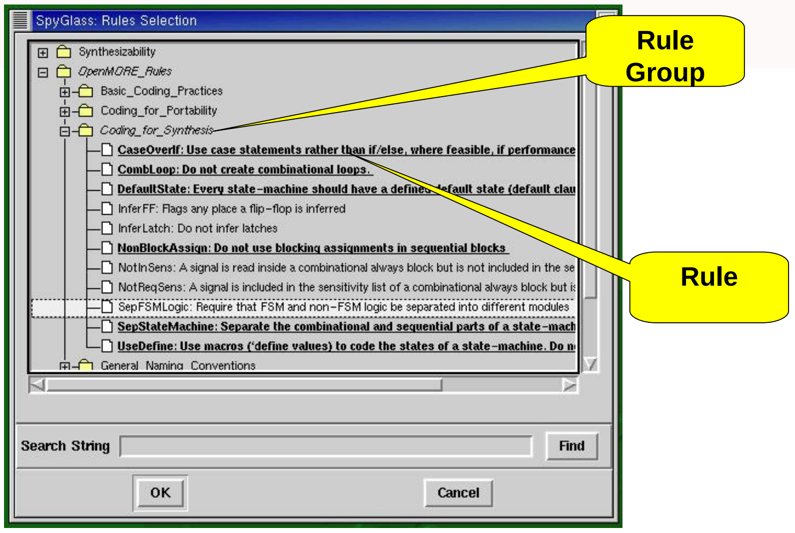
Atrenta Inc | Page 4

Atrenta

Model for Policy Management

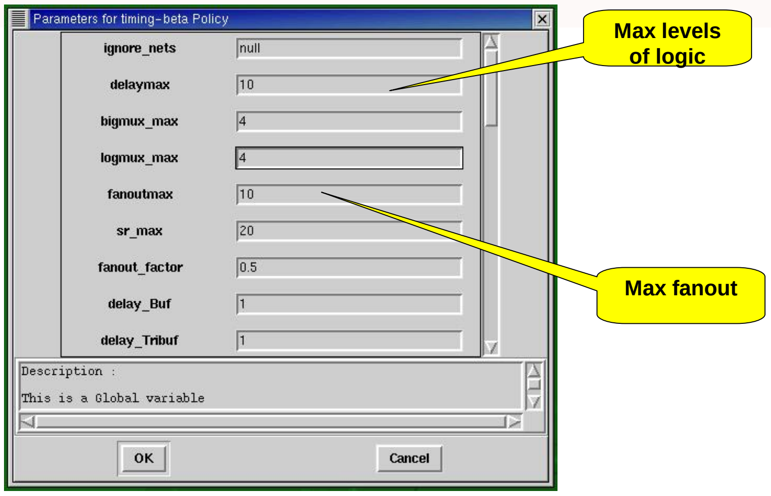
- Policy
  - Lint & Reuse [OpenMORE & STARC]
  - RTL Signoff
  - Design Timing, Testability, Power
  - Verification
  - SoC Integration
- Rule Groups
- Rules
- Policy Application
- Policy Creation
- Analysis and Report

Rules and Groups


SpyGlass: Rules Selection

- Synthesizability
- *OpenMORE_Rules*
  - Basic_Coding_Practices
  - Coding_for_Portability
  - *Coding_for_Synthesis*
    - **CaseOverIf: Use case statements rather than if/else, where feasible, if performance**
    - **CombLoop: Do not create combinational loops.**
    - **DefaultState: Every state-machine should have a defined default state (default clau**
    - InferFF: Flags any place a flip-flop is inferred
    - InferLatch: Do not infer latches
    - **NonBlockAssign: Do not use blocking assignments in sequential blocks**
    - NotInSens: A signal is read inside a combinational always block but is not included in the se
    - NotReqSens: A signal is included in the sensitivity list of a combinational always block but is
    - SepFSMLogic: Require that FSM and non-FSM logic be separated into different modules
    - **SepStateMachine: Separate the combinational and sequential parts of a state-mach**
    - **UseDefine: Use macros ('define values) to code the states of a state-machine. Do n**
  - General_Naming_Conventions

Parameterized Policies



ignore_nets	null
delaymax	10
bigmux_max	4
logmux_max	4
fanoutmax	10
sr_max	20
fanout_factor	0.5
delay_Buf	1
delay_Tribuf	1

```
Description :
This is a Global variable
```

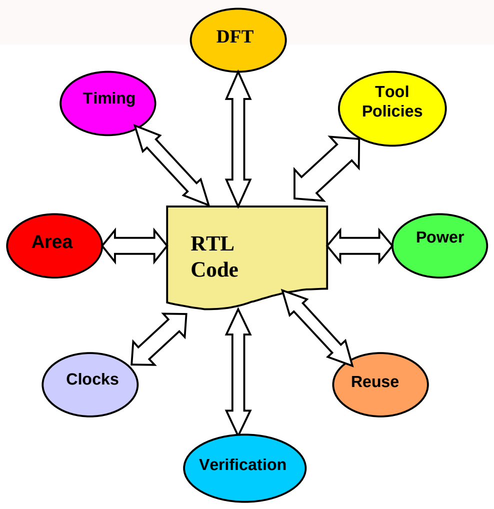
Policy Engines

- Fast high-level synthesis
- Traversal Engine
- Cycle-based simulator

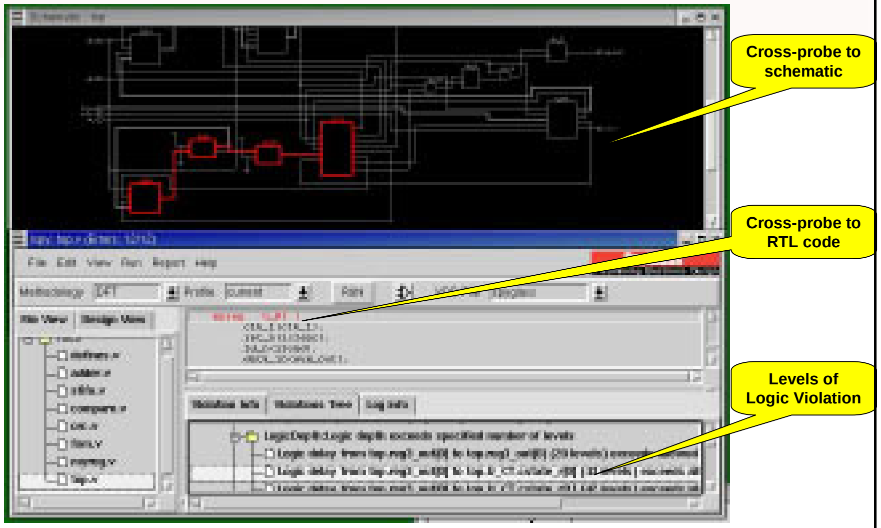
Analysis & Report

- Guiding from detection of issues to solution
- Various formats for reporting
  - **variations on summary**
  - **scoring**
- Software management of underlying issues
  - **manage by design units, files, design as a whole**
  - **manage the state of issues**
  - **mechanisms to control reporting**

Policies

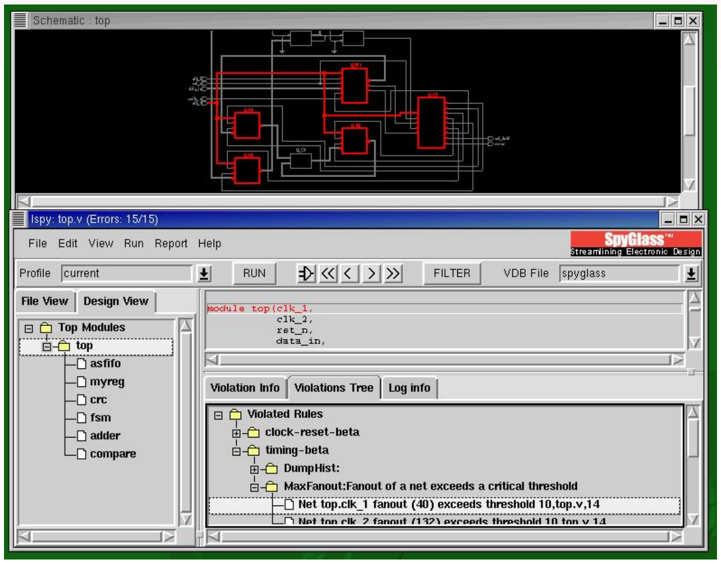


Levels of Logic



LogicDepth:Logic depth exceeds specified number of levels

Fanout Violations



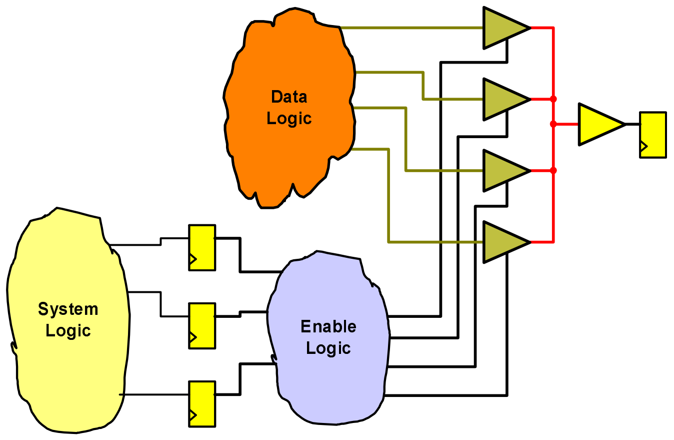
```
module top(clk_1,
           clk_2,
           rst_n,
           data_in,
```

- Top Modules
  - top
    - asfifo
    - myreg
    - crc
    - fsm
    - adder
    - compare

- Violated Rules
  - clock-reset-beta
  - timing-beta
    - DumpHist:
    - MaxFanout:Fanout of a net exceeds a critical threshold
      - Net top.clk_1 fanout (40) exceeds threshold 10,top.v,14
      - Net top.clk_2 fanout (132) exceeds threshold 10,top.v,14

Resolving Issues By Simulation

Only one driver active at a time for tri-state buses

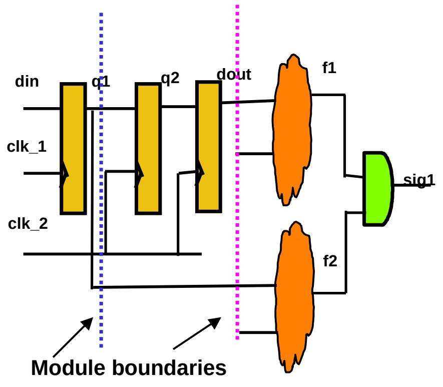


Multiple Clock Domain Issues

```
module abc
always @(posedge clk_1) begin // clk domain1
   q1 <= din;
end
endmodule  //end of module abc

module xyz
….
always @(posedge clk_2) begin // clk domain2
   q2 <= q1;
end
endmodule //end of module xyz
…..
always @(posedge clk_2) begin //synch
   dout <= q2;
end
assign f2 = func2(q1,..);
assign f1 = func1(dout, ..);
assign sig1 = f1 & f2;
```

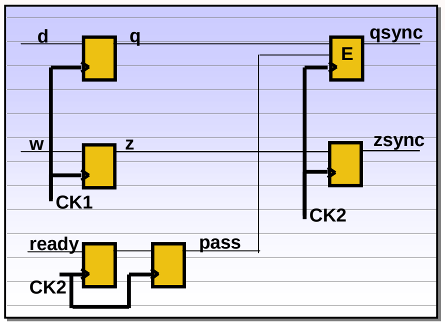
Signals from multiple clock domains converging to a common logic--multi-transition, multi-sample signals



Clock Synchronization Problems

```
module synch…
…
always @(posedge clk1) begin
   q <= d;
   z <= w;
   ready <= dready;
End
…
// generate sync signal
always @(posedge clk2) begin
   rds <= ready;
   pass <= rds;
end
…
// synchronize
always @(posedge clk2) begin
   if (pass) qsync = q;
   …

   zsynch = z;
end
```



- z-bus not correctly synchronized
- May not be found in simulation
- Impact
  - Yield issues
  - Field failures

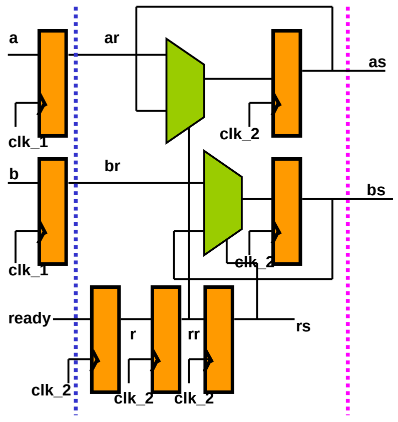
Atrenta

Clock Synchronization

```
module abc
always @(posedge clk_1) begin // clk domain1
   ar <= a;br <= b; r<=ready
end
endmodule  //end of module abc

module xyz
….
always @(posedge clk_2) begin // clk domain2
   rr <= r;
end
always @(posedge clk_2) begin // clk domain2
   rs <= rr;
end
always @(posedge clk_2) begin // clk domain2
   if (rs) as<= ar;
   if (rr) bs<= br;
end
…..
endmodule //end of module xyz
```

Enable of destination flop is driven by a synchronized signal



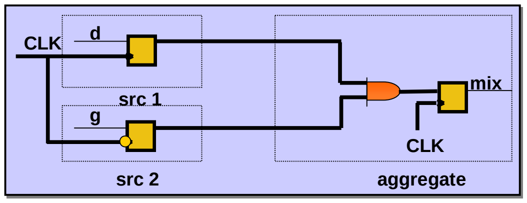
Atrenta Inc | Page 16

Atrenta

Testability Issues

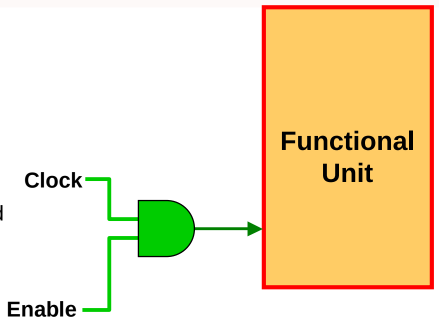
- Signals from mixed edge domains combined
  - Create problems for scan insertion
- Not found until ATPG check
- Impact
  - Wasted implementation cycles
  - Schedule delays

```
module src1…
…
always @(posedge clk)
   q1 = d
…
module src2…
…
always @(negedge clk)
   q2 = g
…
module aggregate…
…
always @(posedge clk)
   mix = q1 & q2
…
```



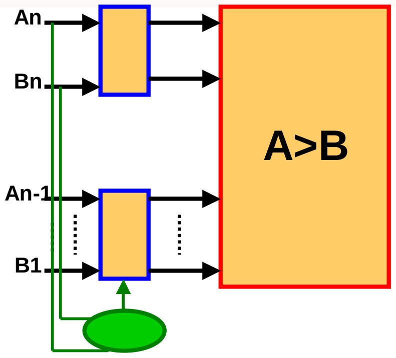
Clock Gating

- Use of gated clocks to selectively turn on/off various units in the design
- All large functional units are use enabled clock
  - All banks of memory controlled by enabled clock
  - Large fanout flops use enabled clocks



Pre-computation

- Suggest the use of pre-computation
- Example: A > B where A and B are 32-bit buses, result can be obtained by examining A[31] and B[31] and rest of the computation gated based on this result



Summary

- Policy-Based RTL Design
- Policy Management
- Elements of Policy
  - **Policy elements**
  - **Policy engine**
  - **Analysis and reporting**
- Examples of pertinent design issues