Treating Constraints as Components: An Experiment in User Control

Dr Carl Seger

Senior Principal Engineer

Intel Corporation

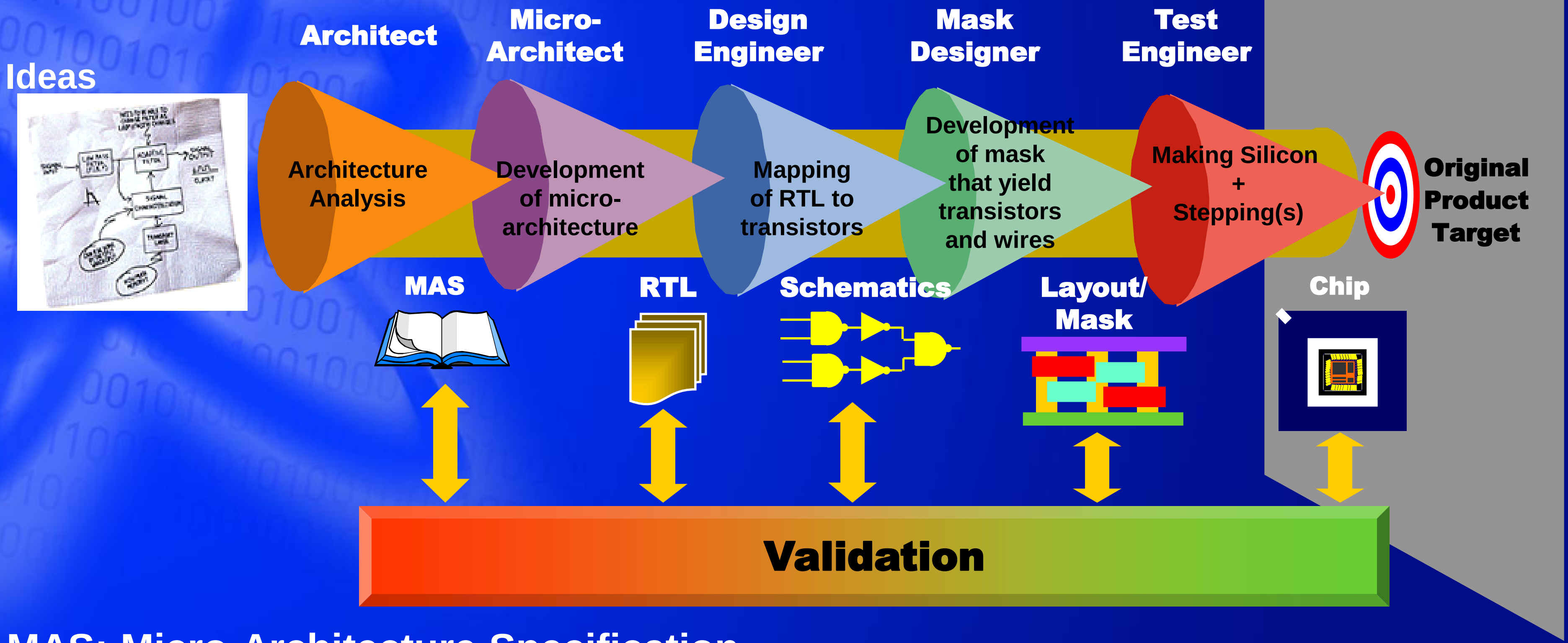
Nov. 10, 2011

Outline

- Background & Motivation
- Integrated Design and Verification System
- Property Handling in IDV
- Examples of Transformations using Properties
- Larger Design Example
- Conclusions & Future Work

Motivation

The Design Process at 10,000 ft

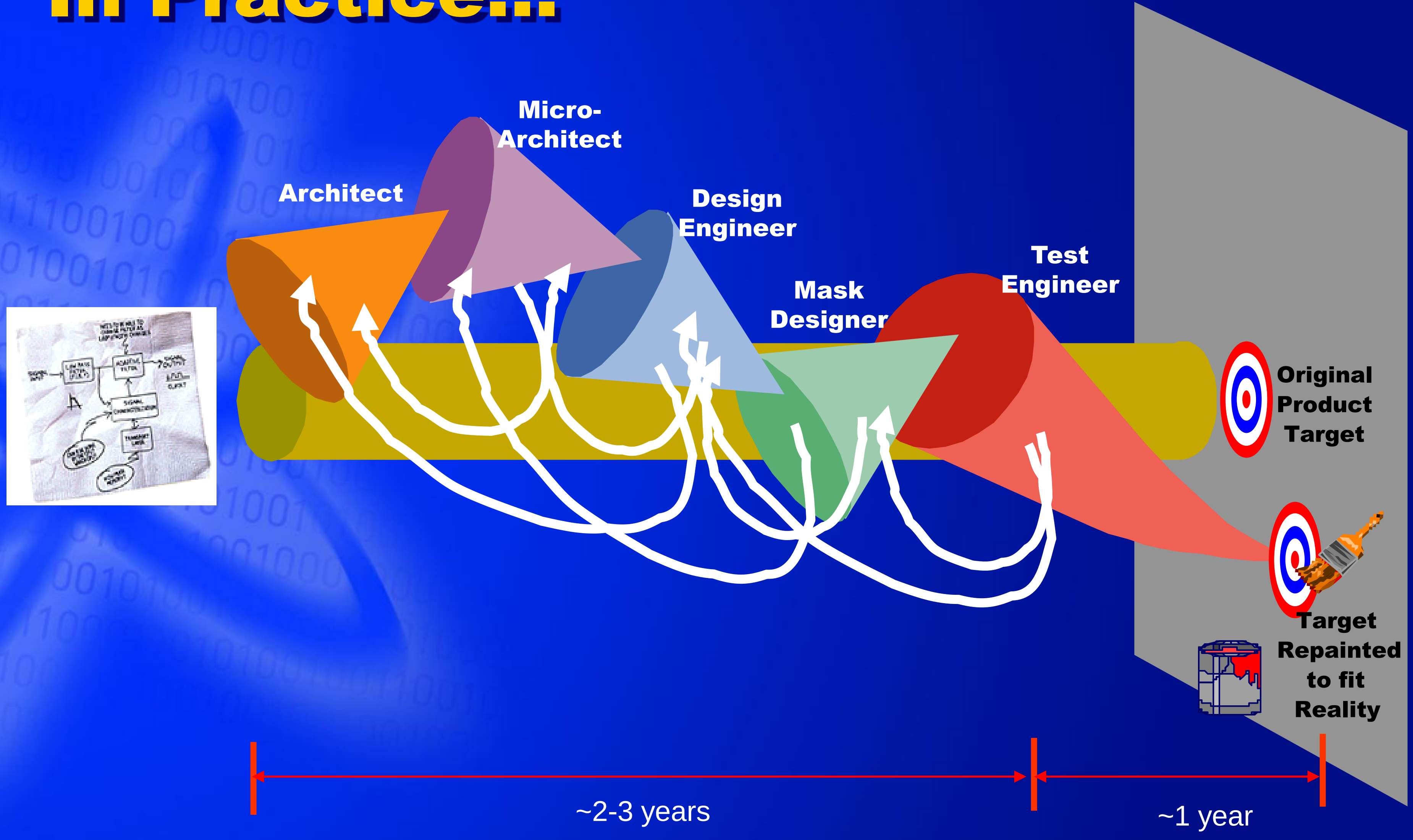


MAS: Micro-Architecture Specification

RTL: Register-Transfer Language

This is the theory…

In Practice...



Micro-Architect

Architect

Design Engineer

Mask Designer

Test Engineer

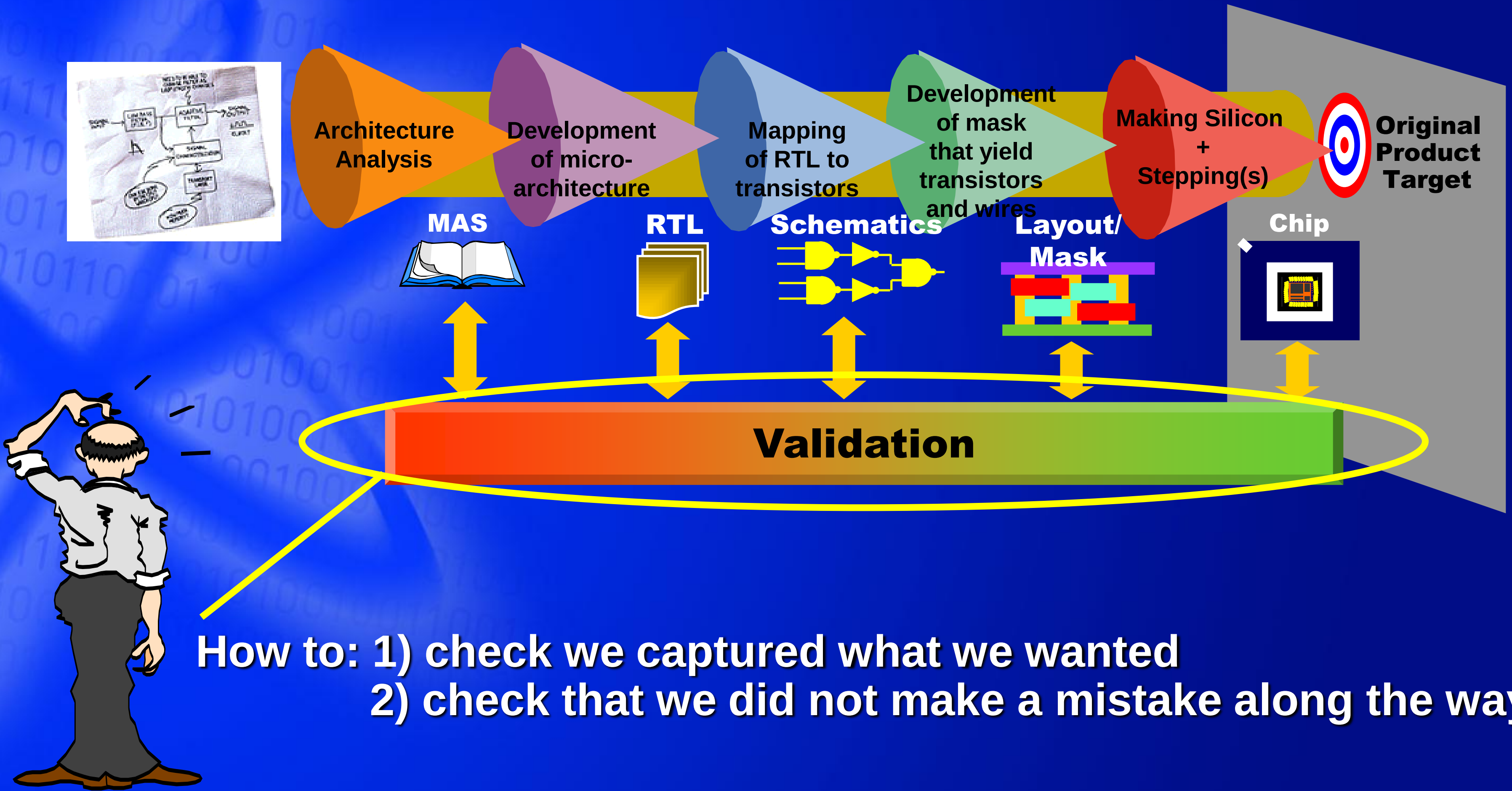
Original Product Target

Target Repainted to fit Reality

~2-3 years

~1 year

Validation

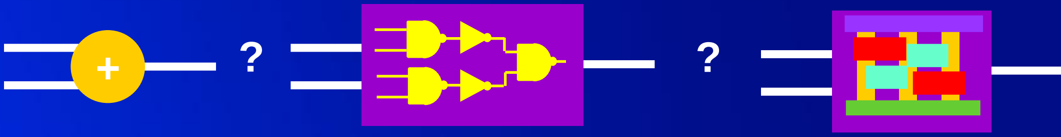

Architecture Analysis
Development of micro-architecture
Mapping of RTL to transistors
Development of mask that yield transistors and wires
Making Silicon + Stepping(s)
Original Product Target
MAS
RTL
Schematics
Layout/ Mask
Chip
Validation

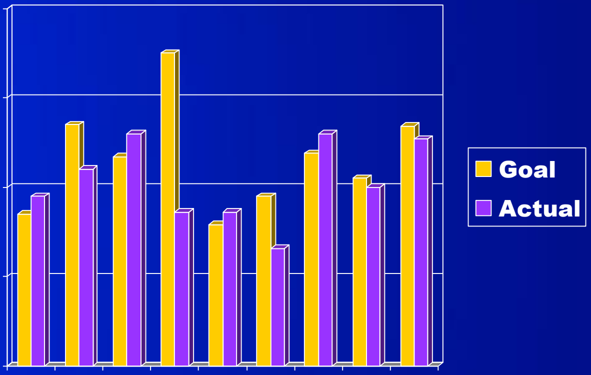
How to: 1) check we captured what we wanted

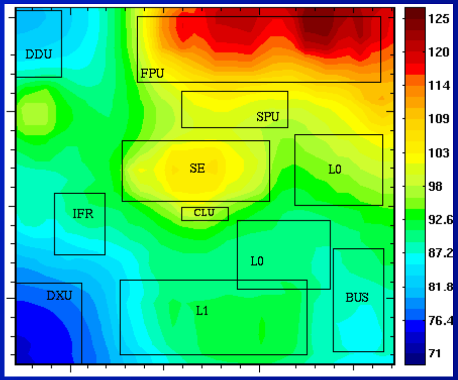
2) check that we did not make a mistake along the way

What Needs to be Validated?

- Functionality
- Performance
- Power & Thermal
- Physical form
- Documentation
- Reliability
- Testing procedure
- …

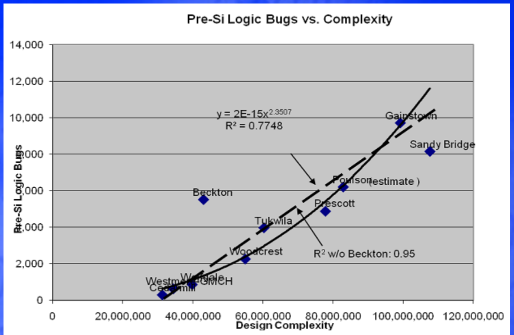




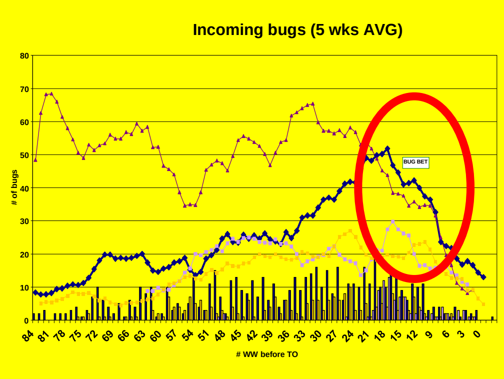


Logic Validation Brick Wall

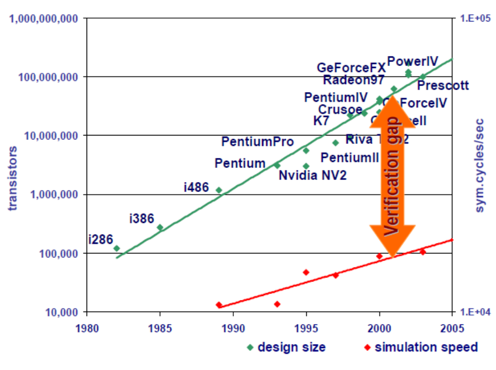
Too many pre-Si bugs!



Bugs found too late

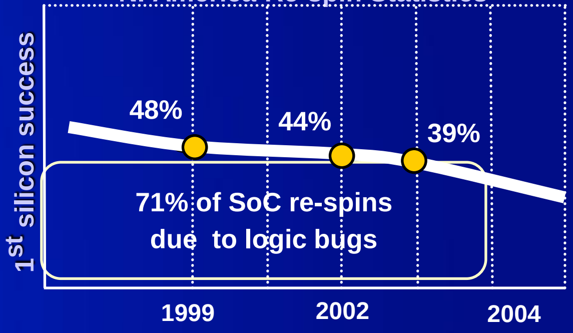


Simulation less efficient



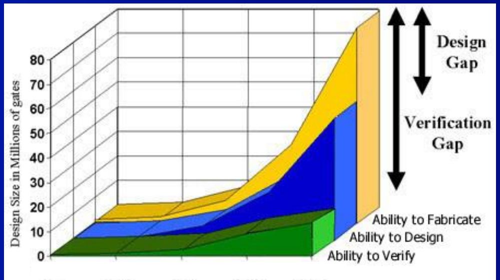
Verification killing schedules

N. America Re-spin Statistics



71% of SoC re-spins due to logic bugs

Validation is now limiting new features.



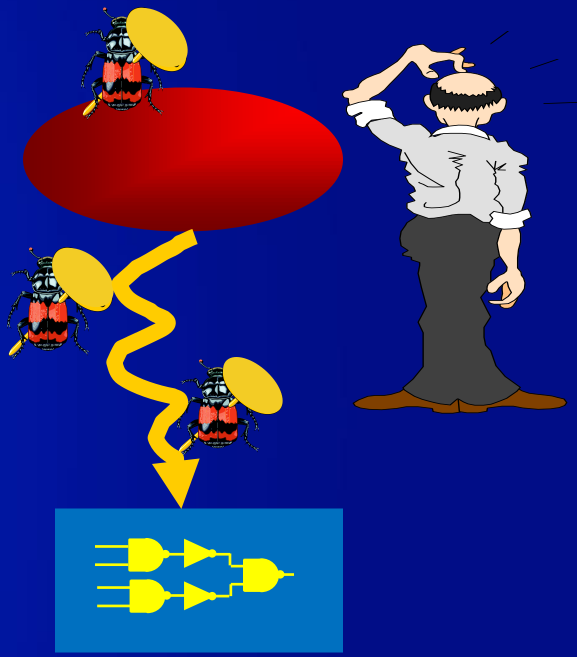
Without major breakthroughs, verification will be a non-scalable, show-stopping barrier to further progress in the semiconductor industry

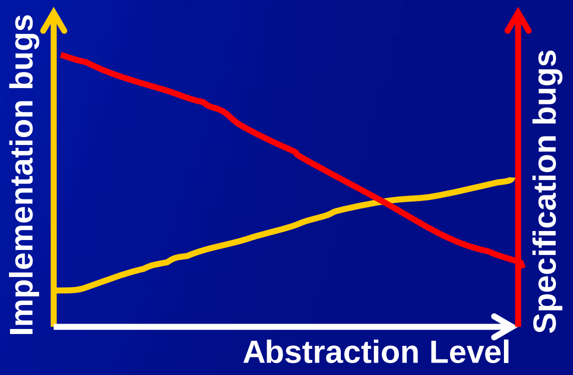
THE INTERNATIONAL TECHNOLOGY ROADMAP FOR SEMICONDUCTORS: *2005/6*

Integrated Design & Verification

Two Classes of Bugs:

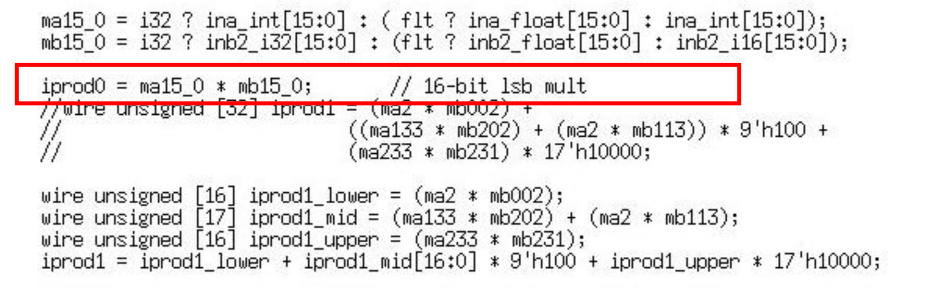
- Specification bugs
  - “What” is captured incorrectly
    - Unintended interactions
    - Deadlocks & Livelocks
- Implementation bugs
  - “How” is captured incorrectly
    - Incorrect optimization of algorithm
    - Misunderstanding of algorithm
    - Bug “fix” with unintended effects
- Note:
  - The more abstract the specification is, the more implementation bugs (and vice versa).
  - Anecdotal evidence indicate that the more abstract specification, the fewer total bugs





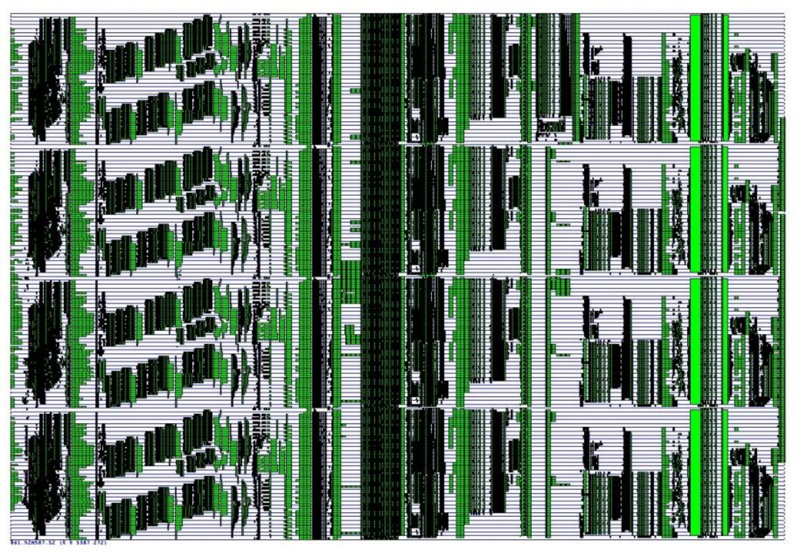
Real problem:

- How to go from:



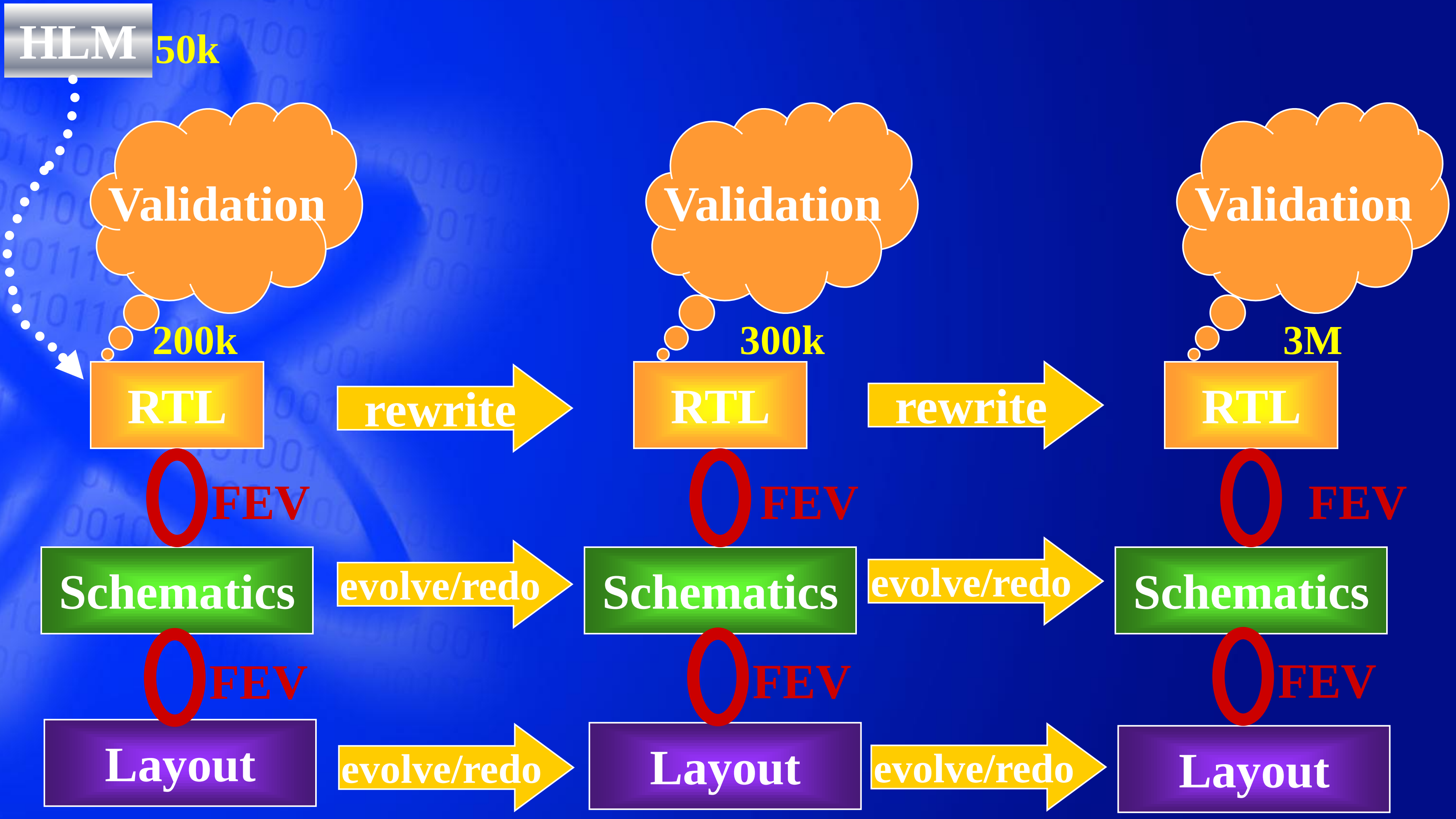
```
ma15_0 = i32 ? ina_int[15:0] : ( flt ? ina_float[15:0] : ina_int[15:0]);
mb15_0 = i32 ? inb2_i32[15:0] : (flt ? inb2_float[15:0] : inb2_i16[15:0]);

iprod0 = ma15_0 * mb15_0;        // 16-bit lsb mult
//wire unsigned [32] iprod1 = (ma2 * mb002) +
//                           ((ma133 * mb202) + (ma2 * mb113)) * 9'h100 +
//                           (ma233 * mb231) * 17'h10000;

wire unsigned [16] iprod1_lower = (ma2 * mb002);
wire unsigned [17] iprod1_mid = (ma133 * mb202) + (ma2 * mb113);
wire unsigned [16] iprod1_upper = (ma233 * mb231);
iprod1 = iprod1_lower + iprod1_mid[16:0] * 9'h100 + iprod1_upper * 17'h10000;
```

- to:

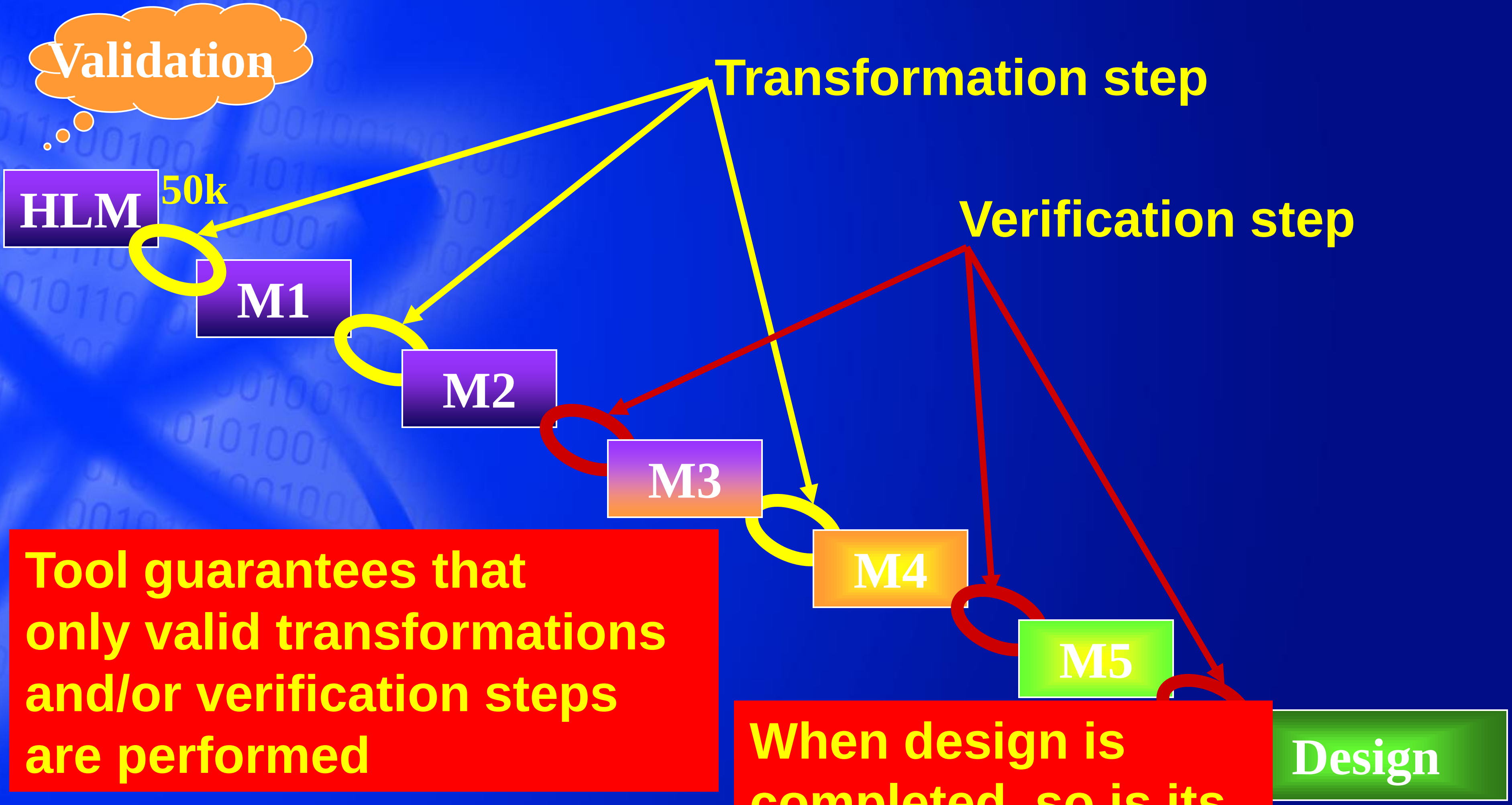


- **Quickly**
- **Correctly**
- **Meeting timing goals**
- **Meeting area goals**
- **Meeting power goals**
- **Meeting manufacturability goals**
- **…**

Today's Approach



A Different Approach: Integrated Design and Verification (IDV)



Tool guarantees that only valid transformations and/or verification steps are performed

When design is completed, so is its impl. verification.

Design in IDV

- Since IDV bridges HLM to symbolic layout, design activities inside IDV occur at several levels:

- High-level algorithmic refinements, e.g.
  - change an algorithm from "simple to write and validate" to an algorithm that "can be implemented efficiently in silicon"

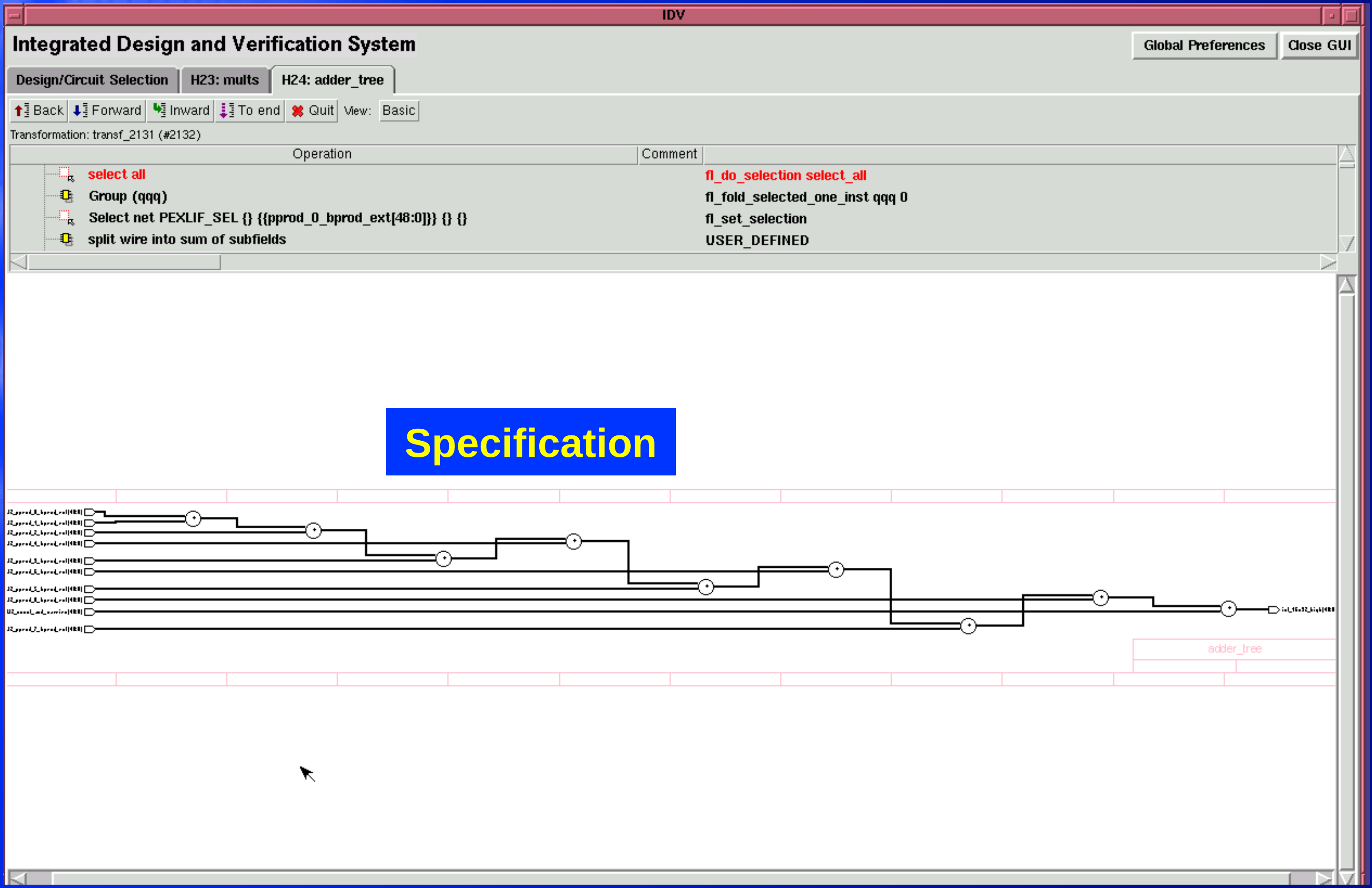
- Mid-level (implementation) refinements, e.g.
  - change an "a+b" component to an efficient (power/area/timing) gate implementation

- Low-level (physical) refinements, e.g.
  - placement directives, pre-routes, slope management by buffer insertions and/or mapping to different cells

High-Level Algorithmic Design

Example of Algorithmic Design

- Task: Split a chain of 9 49-bit adders into two chains; one for the higher bits and one for the lower bits

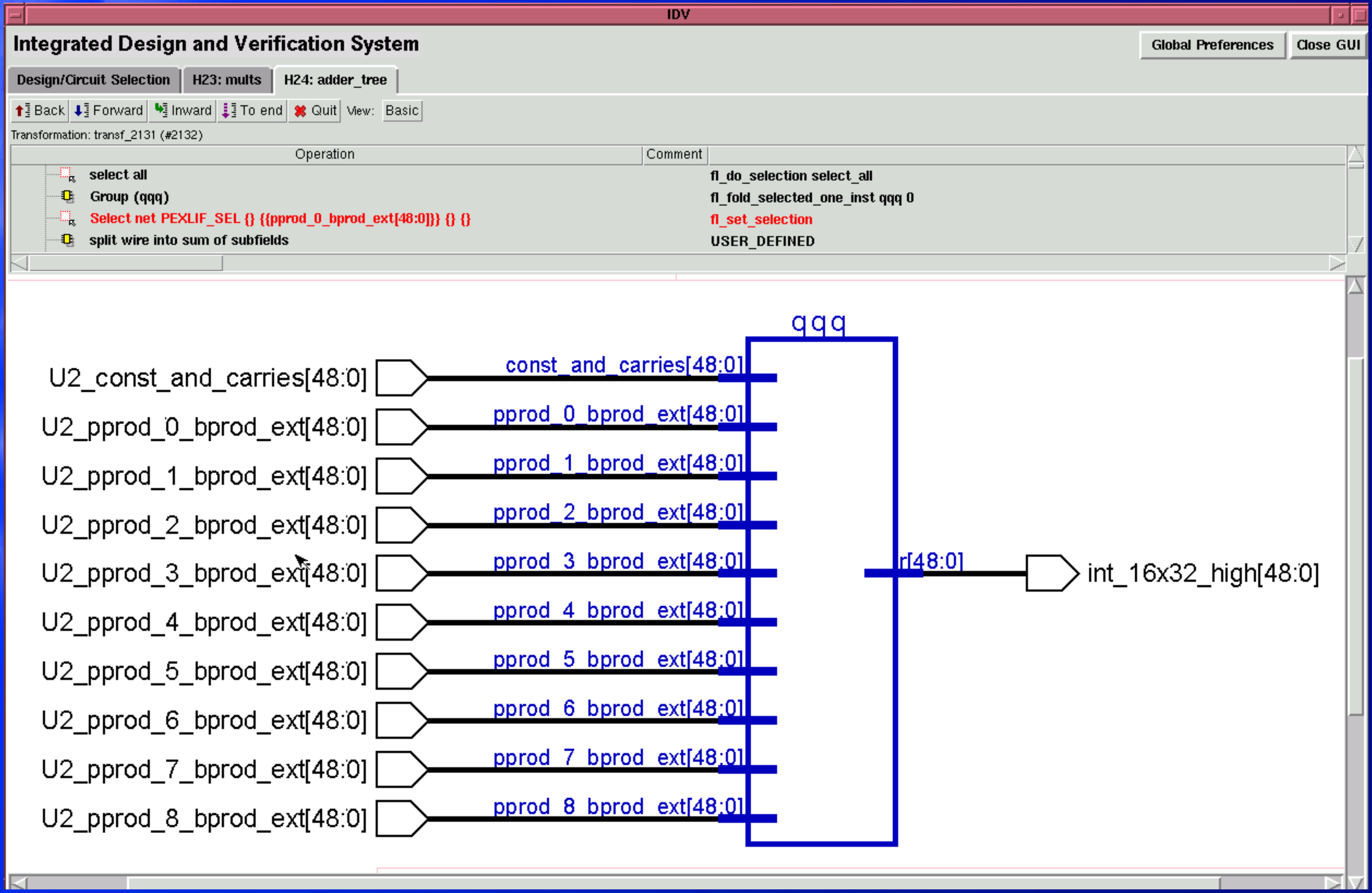


Transformation: transf_2131 (#2132)

Operation	Comment
select all	fl_do_selection select_all
Group (qqq)	fl_fold_selected_one_inst qqq 0
Select net PEXLIF_SEL {} {{pprod_0_bprod_ext[48:0]}} {} {}	fl_set_selection
split wire into sum of subfields	USER_DEFINED

Specification

Step 1: Group adders

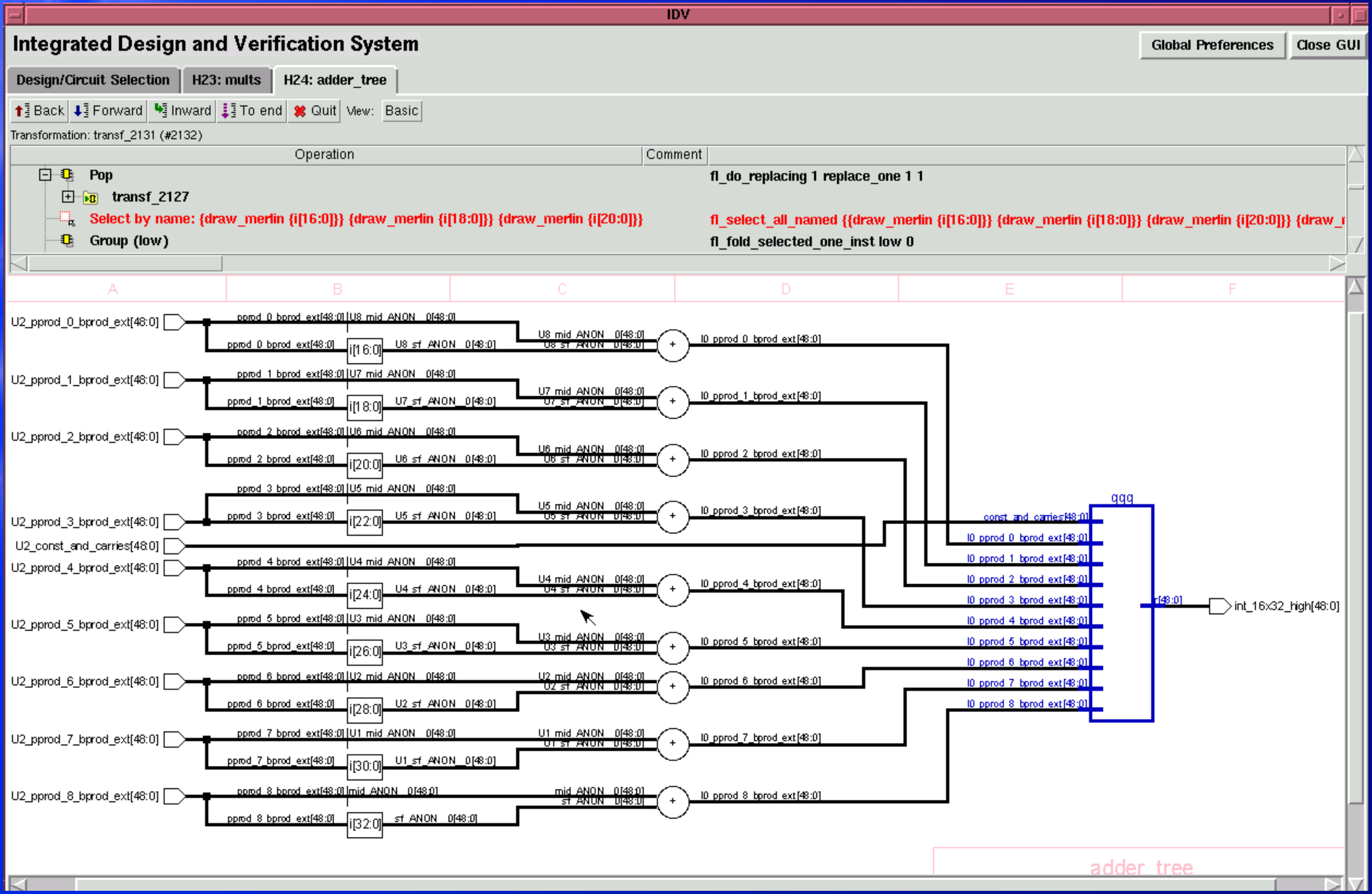


Transformation: transf_2131 (#2132)

Operation	Comment
select all	fl_do_selection select_all
Group (qqq)	fl_fold_selected_one_inst qqq 0
Select net PEXLIF_SEL {} {{pprod_0_bprod_ext[48:0]}} {} {}	fl_set_selection
split wire into sum of subfields	USER_DEFINED

Step 2: Insert high-low adder in each input wire

- Use FEV to verify e.g. that $a[48:0]=a[48:33]*2^{33}+a[32:0]$

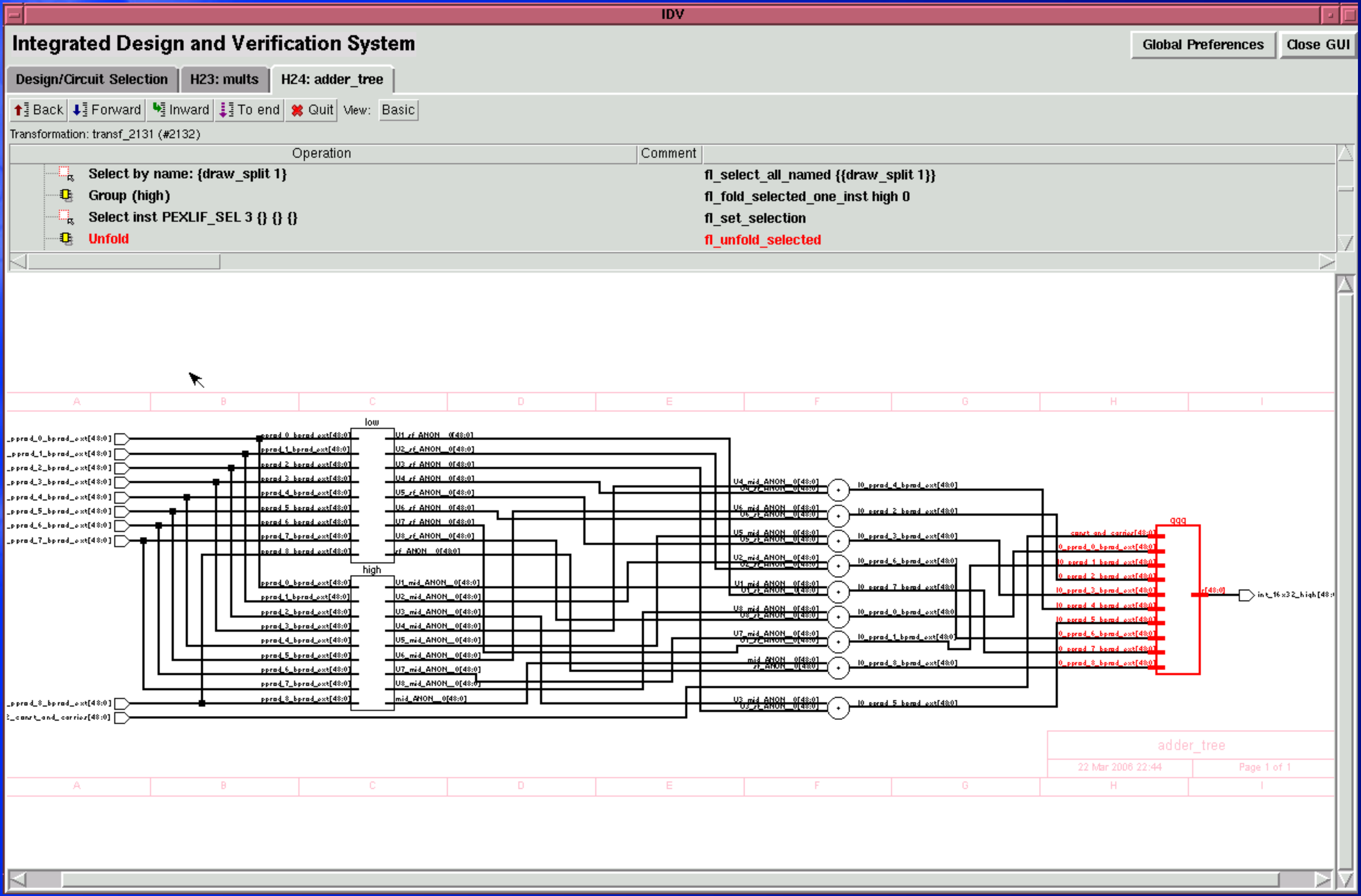


IDV
Integrated Design and Verification System
Global Preferences
Close GUI
Design/Circuit Selection
H23: mults
H24: adder_tree
Back
Forward
Inward
To end
Quit
View:
Basic
A
B
C
D
E
F
U2_pprod_0_bprod_ext[48:0]
U2_pprod_1_bprod_ext[48:0]
U2_pprod_2_bprod_ext[48:0]
U2_pprod_3_bprod_ext[48:0]
U2_const_and_carries[48:0]
U2_pprod_4_bprod_ext[48:0]
U2_pprod_5_bprod_ext[48:0]
U2_pprod_6_bprod_ext[48:0]
U2_pprod_7_bprod_ext[48:0]
U2_pprod_8_bprod_ext[48:0]
pprod 0 bprod ext[48:0] |U8 mid ANON 0[48:0]
pprod 0 bprod ext[48:0]
i[16:0]
U8 sf ANON 0[48:0]
U8 mid ANON 0[48:0]
U8 sf ANON 0[48:0]
+
I0 pprod 0 bprod ext[48:0]
pprod 1 bprod ext[48:0] |U7 mid ANON 0[48:0]
pprod 1 bprod ext[48:0]
i[18:0]
U7 sf ANON 0[48:0]
U7 mid ANON 0[48:0]
U7 sf ANON 0[48:0]
I0 pprod 1 bprod ext[48:0]
pprod 2 bprod ext[48:0] |U6 mid ANON 0[48:0]
pprod 2 bprod ext[48:0]
i[20:0]
U6 sf ANON 0[48:0]
U6 mid ANON 0[48:0]
U6 sf ANON 0[48:0]
I0 pprod 2 bprod ext[48:0]
pprod 3 bprod ext[48:0] |U5 mid ANON 0[48:0]
pprod 3 bprod ext[48:0]
i[22:0]
U5 sf ANON 0[48:0]
U5 mid ANON 0[48:0]
U5 sf ANON 0[48:0]
I0 pprod 3 bprod ext[48:0]
pprod 4 bprod ext[48:0] |U4 mid ANON 0[48:0]
pprod 4 bprod ext[48:0]
i[24:0]
U4 sf ANON 0[48:0]
U4 mid ANON 0[48:0]
U4 sf ANON 0[48:0]
I0 pprod 4 bprod ext[48:0]
pprod 5 bprod ext[48:0] |U3 mid ANON 0[48:0]
pprod 5 bprod ext[48:0]
i[26:0]
U3 sf ANON 0[48:0]
U3 mid ANON 0[48:0]
U3 sf ANON 0[48:0]
I0 pprod 5 bprod ext[48:0]
pprod 6 bprod ext[48:0] |U2 mid ANON 0[48:0]
pprod 6 bprod ext[48:0]
i[28:0]
U2 sf ANON 0[48:0]
U2 mid ANON 0[48:0]
U2 sf ANON 0[48:0]
I0 pprod 6 bprod ext[48:0]
pprod 7 bprod ext[48:0] |U1 mid ANON 0[48:0]
pprod 7 bprod ext[48:0]
i[30:0]
U1 sf ANON 0[48:0]
U1 mid ANON 0[48:0]
U1 sf ANON 0[48:0]
I0 pprod 7 bprod ext[48:0]
pprod 8 bprod ext[48:0] |mid ANON 0[48:0]
pprod 8 bprod ext[48:0]
i[32:0]
sf ANON 0[48:0]
mid ANON 0[48:0]
sf ANON 0[48:0]
I0 pprod 8 bprod ext[48:0]
qqq
const and carries[48:0]
I0 pprod 0 bprod ext[48:0]
I0 pprod 1 bprod ext[48:0]
I0 pprod 2 bprod ext[48:0]
I0 pprod 3 bprod ext[48:0]
I0 pprod 4 bprod ext[48:0]
I0 pprod 5 bprod ext[48:0]
I0 pprod 6 bprod ext[48:0]
I0 pprod 7 bprod ext[48:0]
I0 pprod 8 bprod ext[48:0]
r[48:0]
int_16x32_high[48:0]
adder tree

Transformation: transf_2131 (#2132)

Operation	Comment
Pop	fl_do_replacing 1 replace_one 1 1
transf_2127	
Select by name: {draw_merlin {i[16:0]}} {draw_merlin {i[18:0]}} {draw_merlin {i[20:0]}}	fl_select_all_named {{draw_merlin {i[16:0]}} {draw_merlin {i[18:0]}} {draw_merlin {i[20:0]}} {draw_r
Group (low)	fl_fold_selected_one_inst low 0

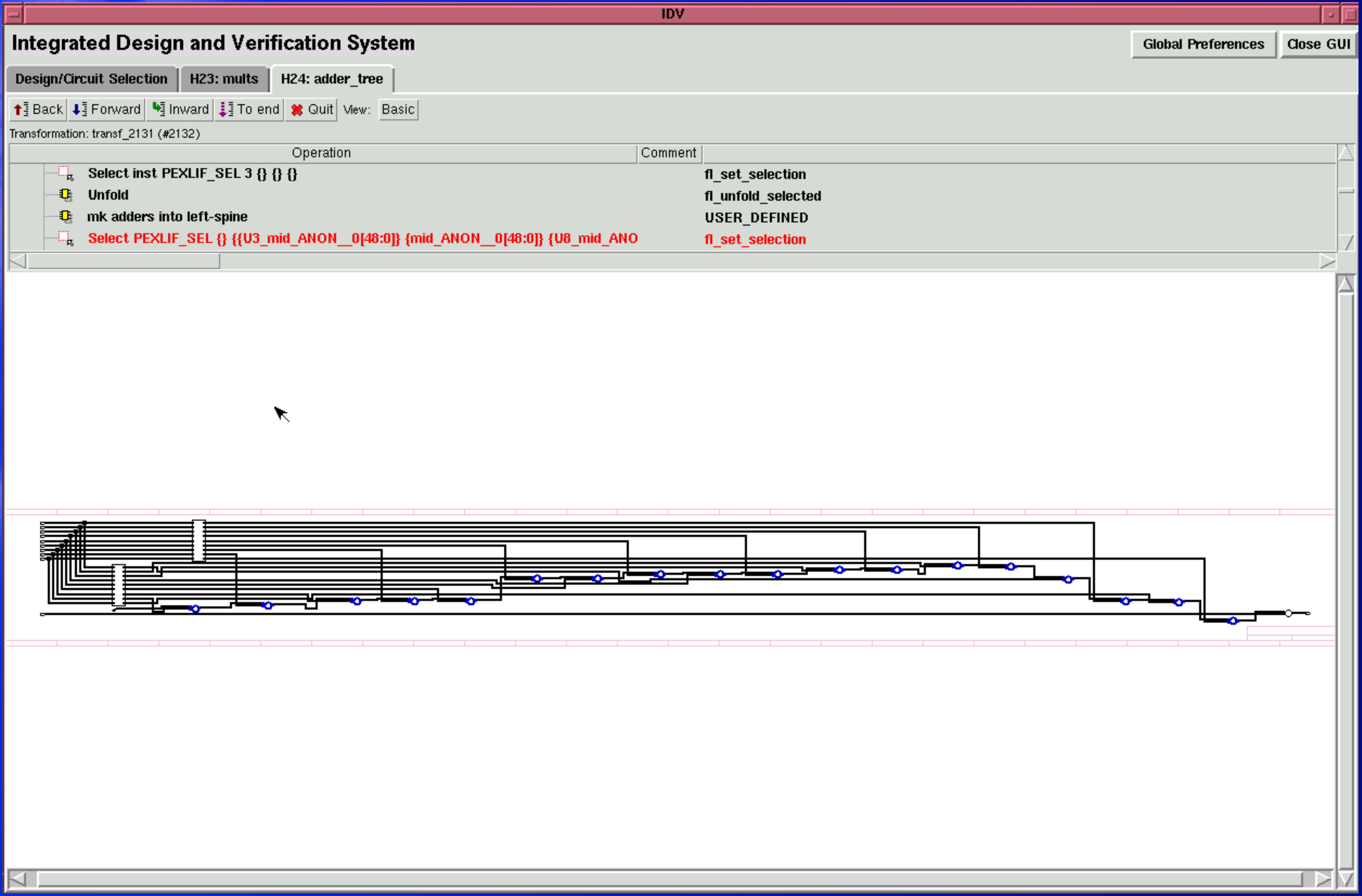
Step 3: Group high-low splitters



Transformation: transf_2131 (#2132)

Operation	Comment
Select by name: {draw_split 1}	fl_select_all_named {{draw_split 1}}
Group (high)	fl_fold_selected_one_inst high 0
Select inst PEXLIF_SEL 3 {} {} {}	fl_set_selection
Unfold	fl_unfold_selected

Step 4: Unfold adders and using associativity transform, make tree into single left-spine of adders

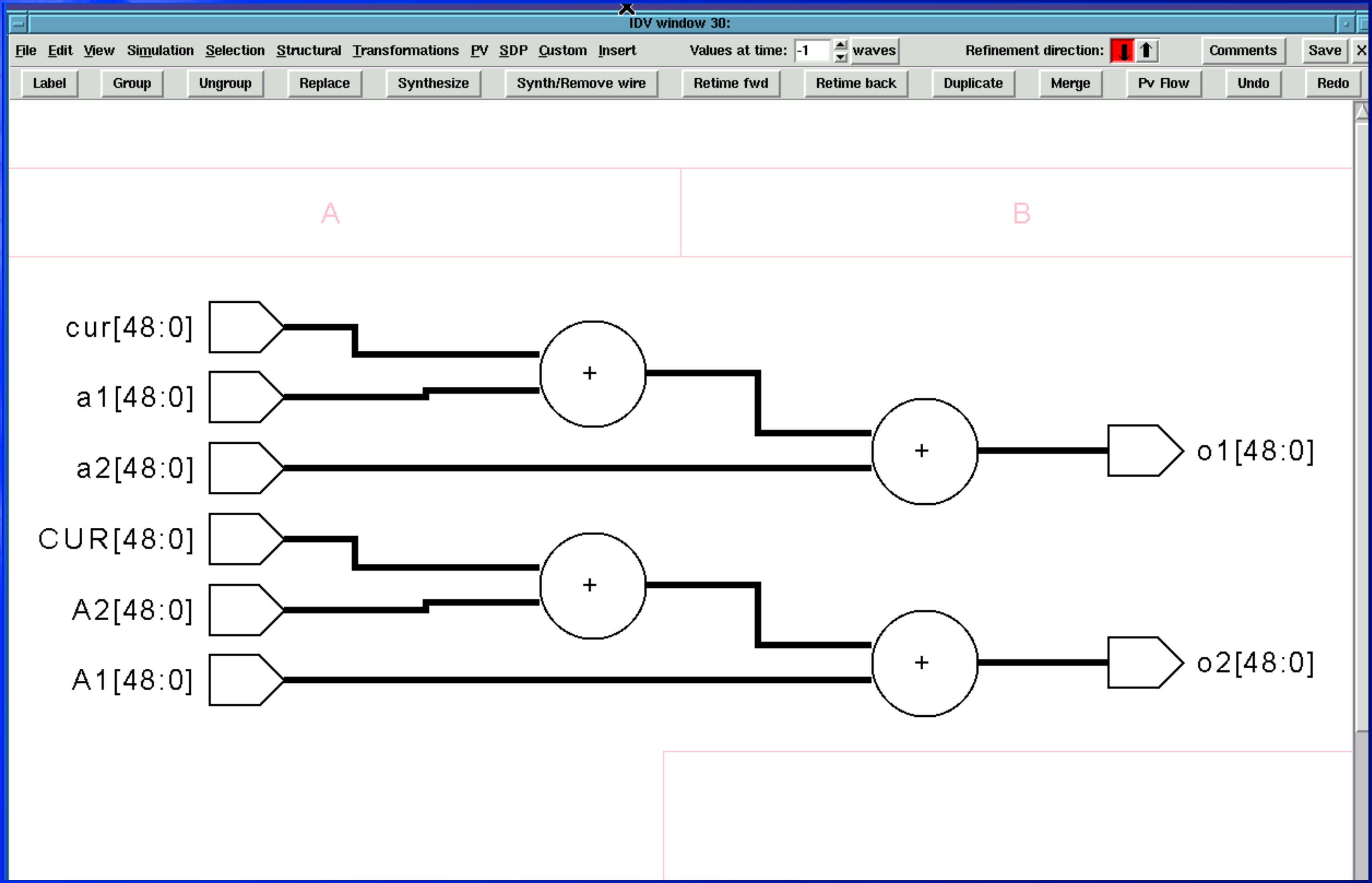


Integrated Design and Verification System

Transformation: transf_2131 (#2132)

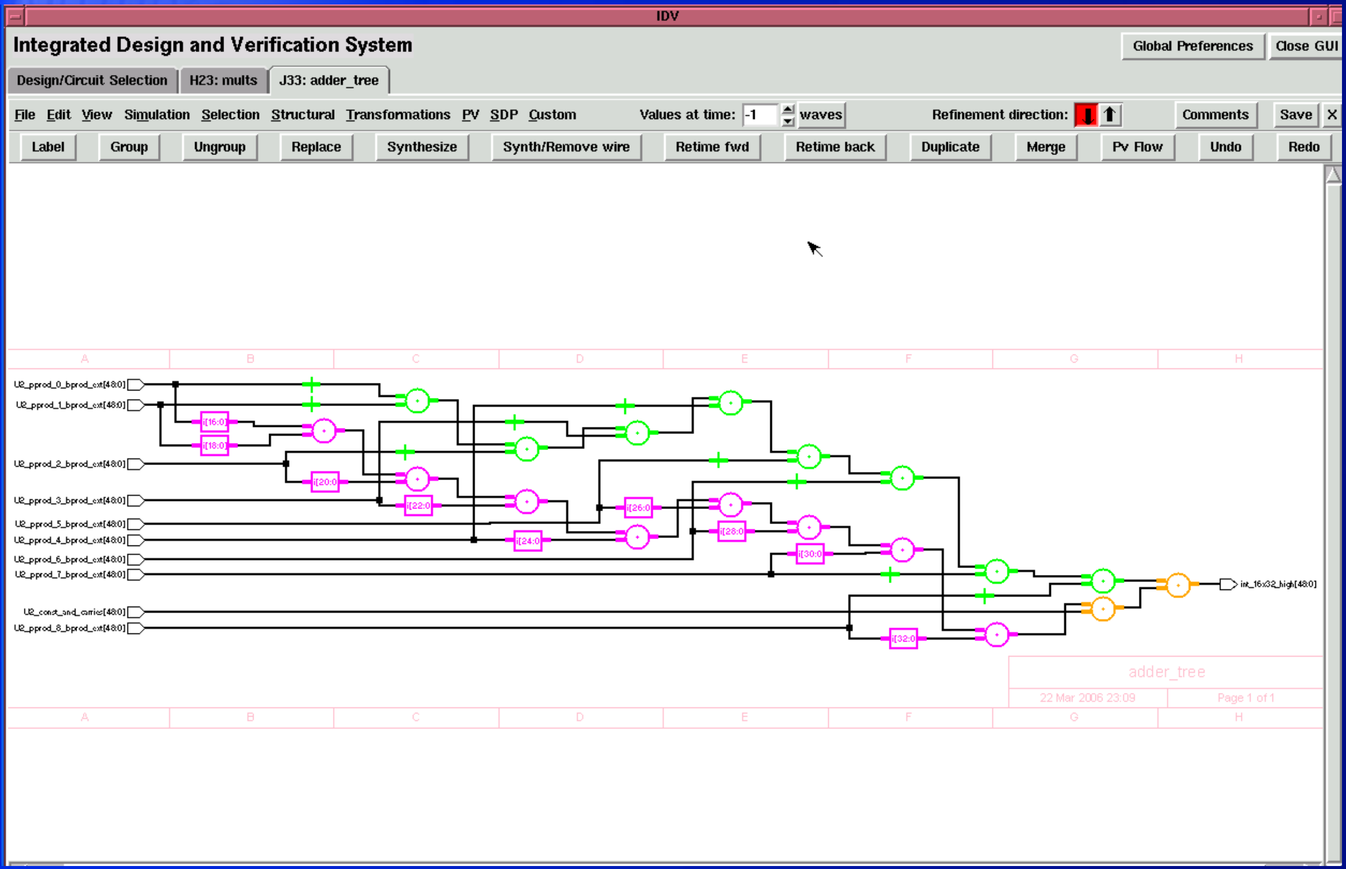
Operation	Comment
Select inst PEXLIF_SEL 3 {} {} {}	fl_set_selection
Unfold	fl_unfold_selected
mk adders into left-spine	USER_DEFINED
Select PEXLIF_SEL {} {{U3_mid_ANON__0[48:0]} {mid_ANON__0[48:0]} {U8_mid_ANO	fl_set_selection

Step 5: Use FEV to verify the small transformation (swap arguments):



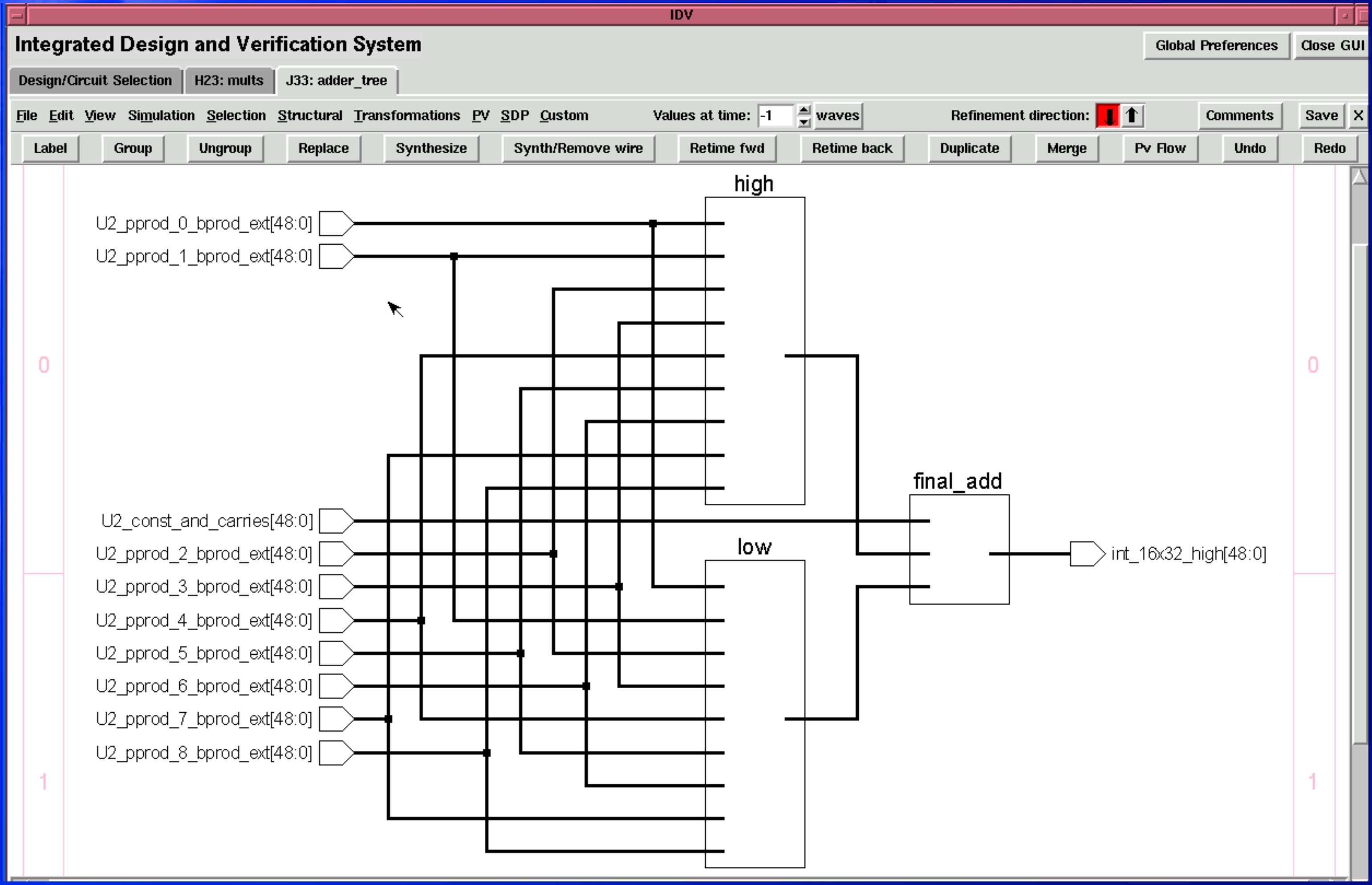
Step 6: Let IDV repeatedly apply this transformation to yield*:

- Where green is high-bits, purple is low-bits and yellow is merge-addition



* Somewhat simplified. Some extra “guidance” is needed to create the desired result.

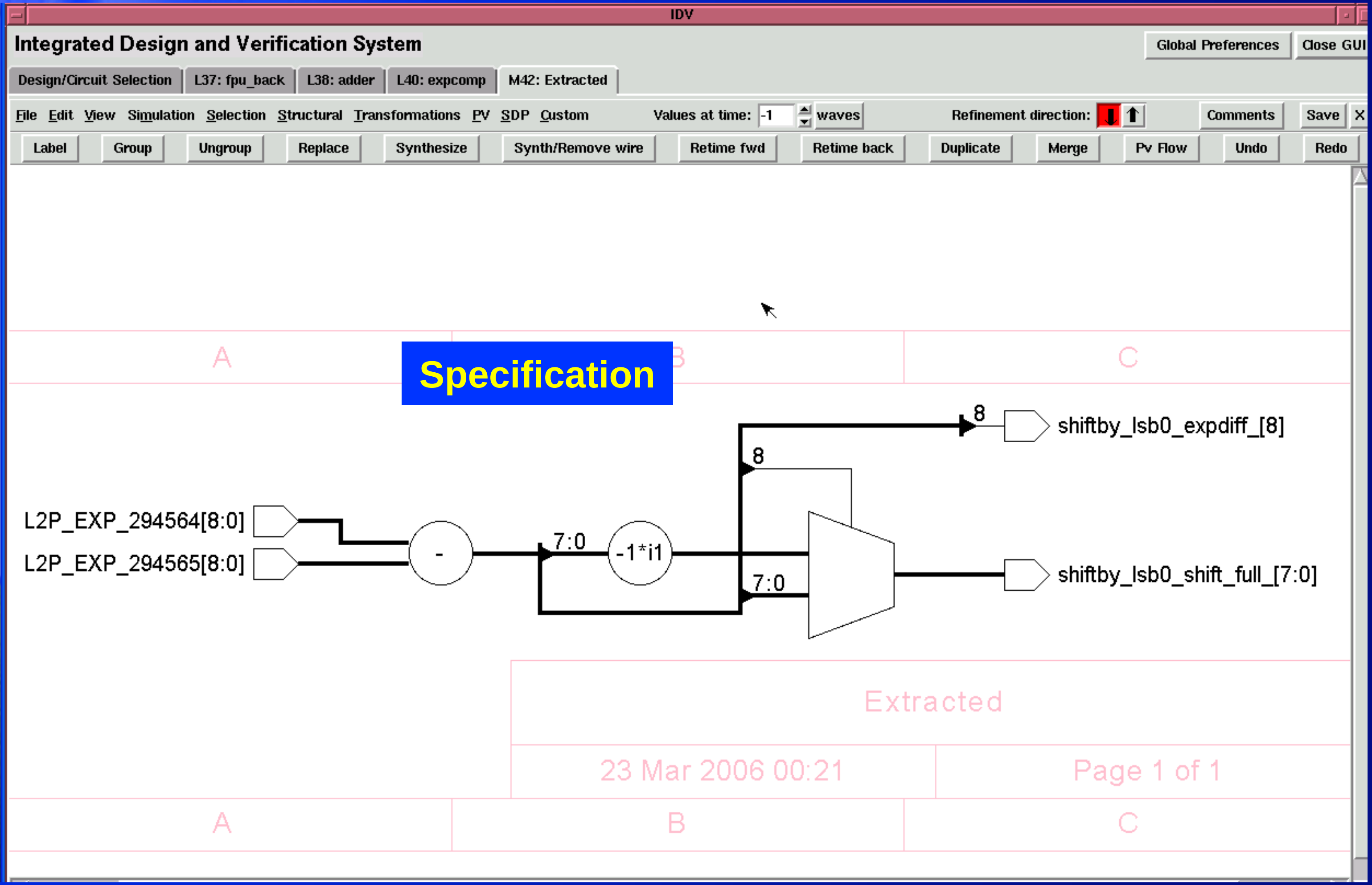
Step 7: Finally group the different pieces to get:



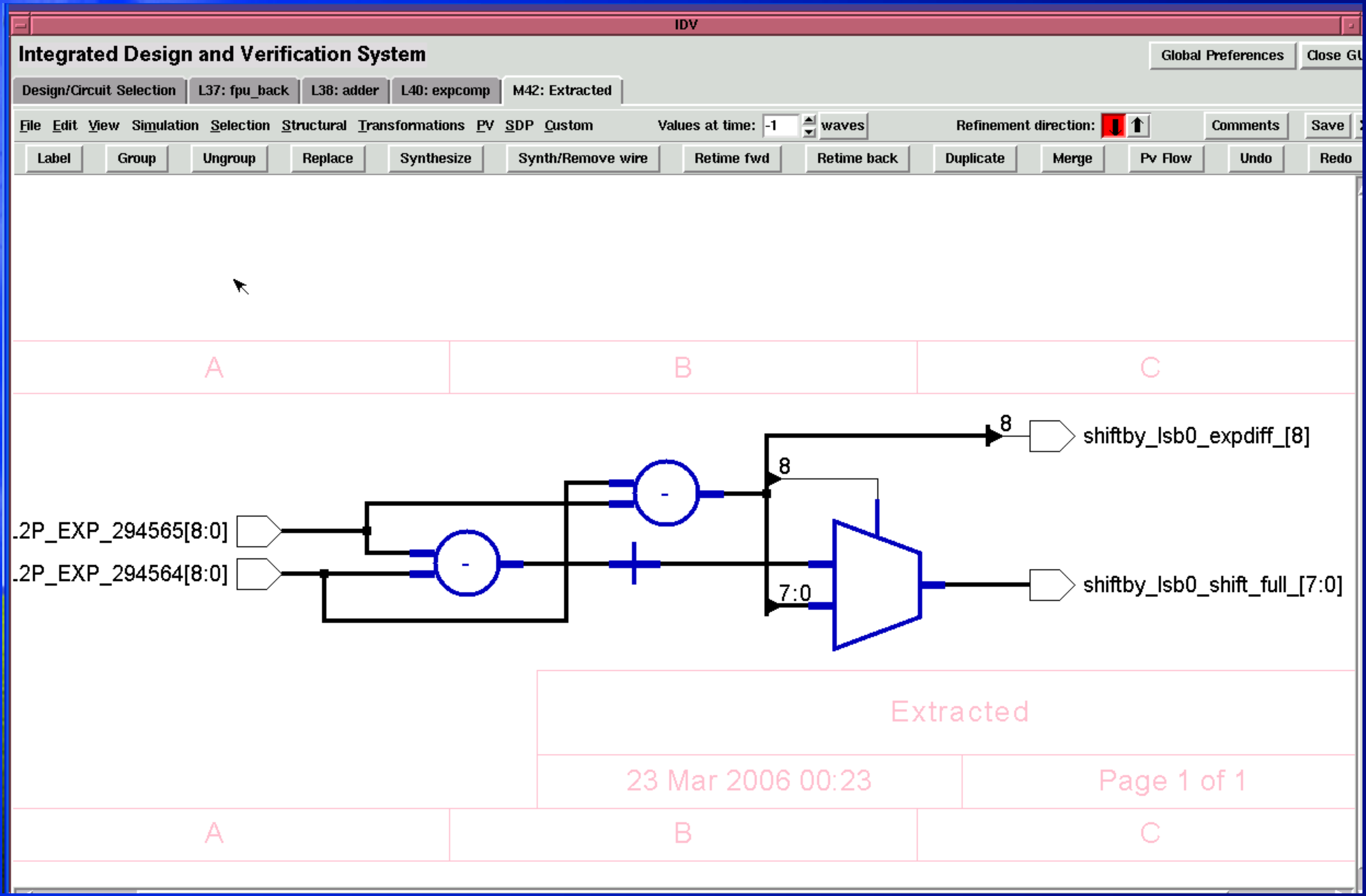
Mid-Level Design

Mid-Level Design in IDV

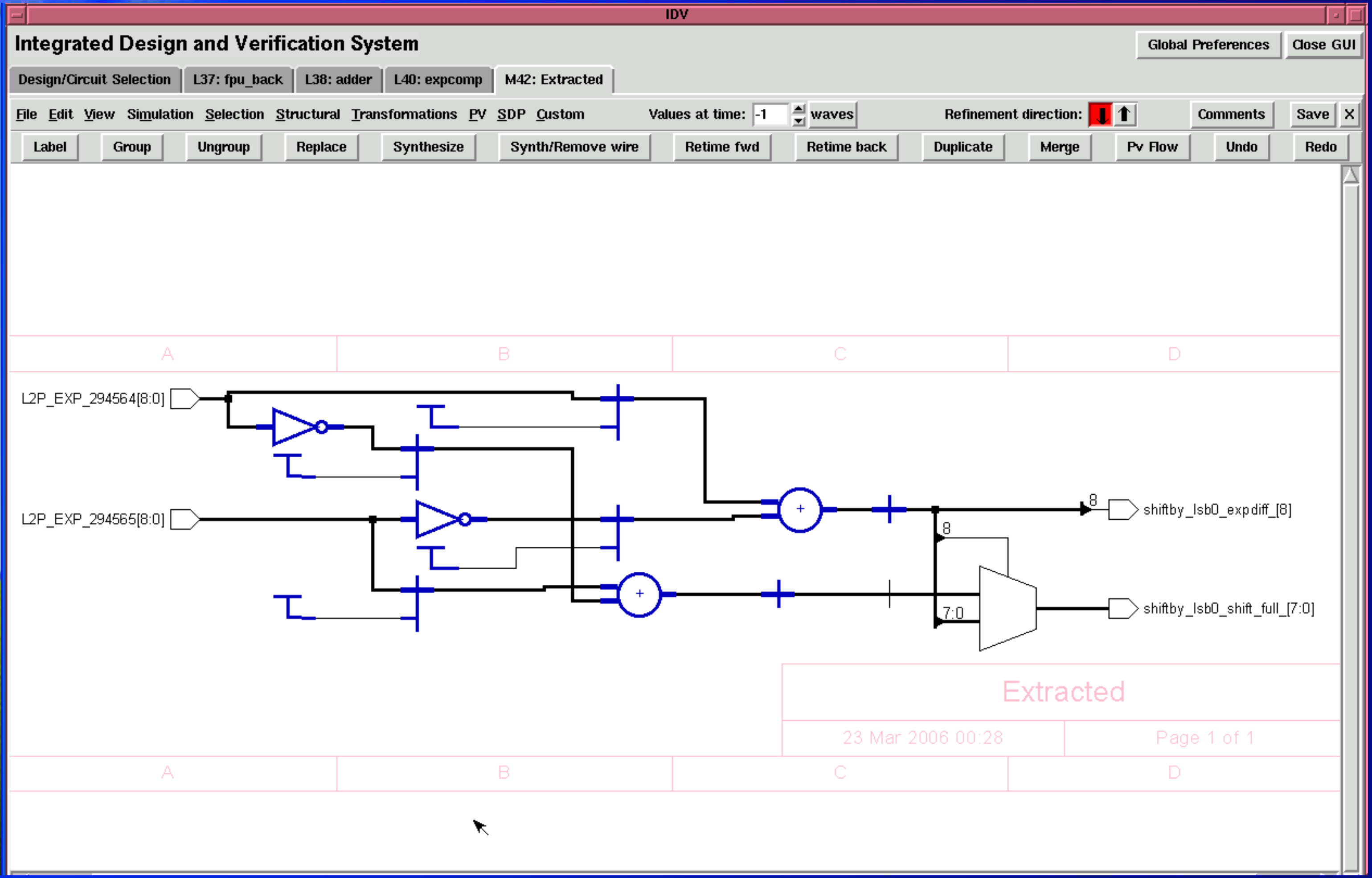
- Problem: Subtraction and negation in series!



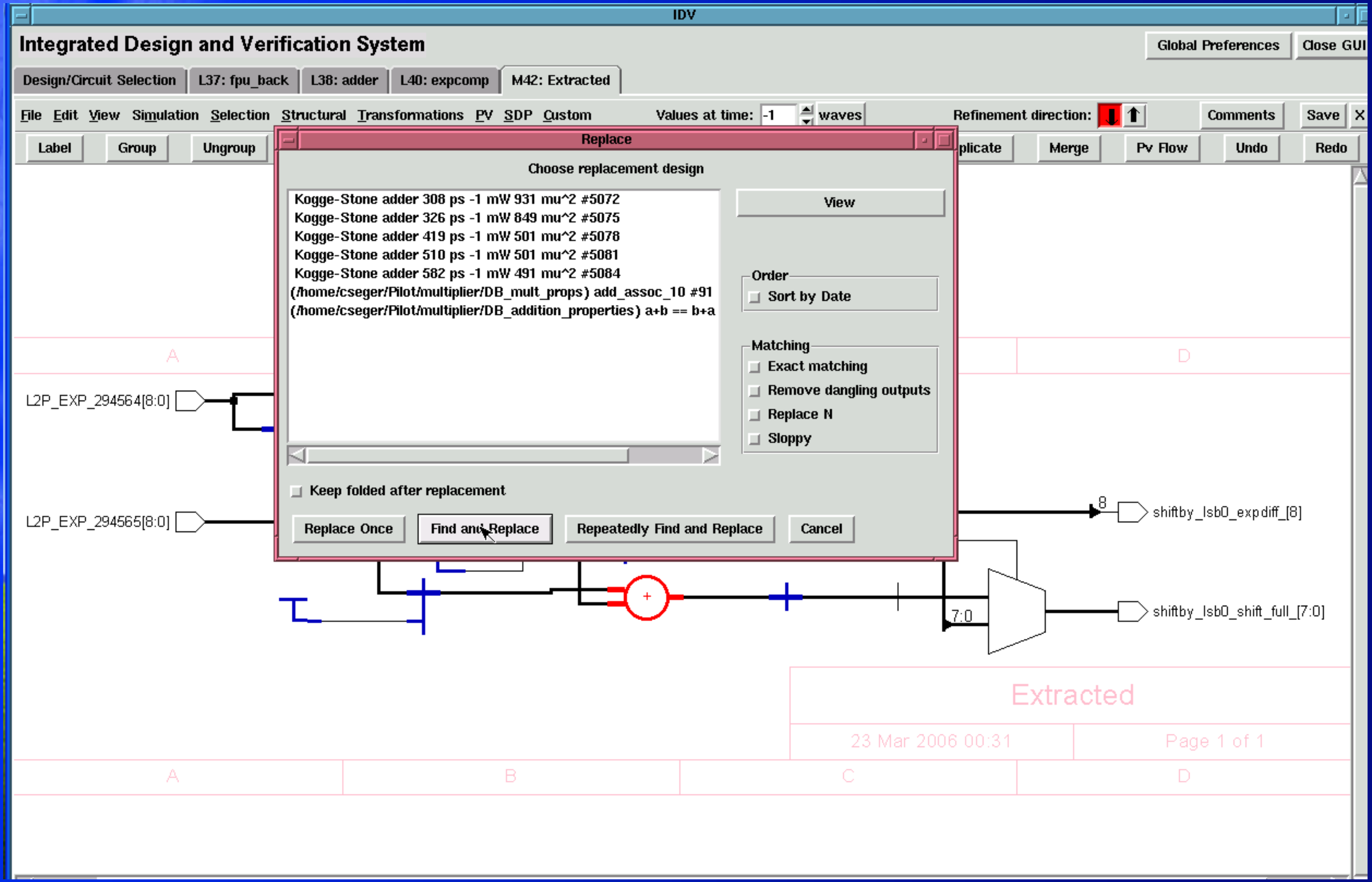
Step 1: Make new design and FEV immediately against spec.



Step 2: Design one subtractor from adder and find-and-replace to find every occurrence



Step 3: Get a suitable adder candidate from library (speed, power, area, …) and use find-and-replace again



Choose replacement design

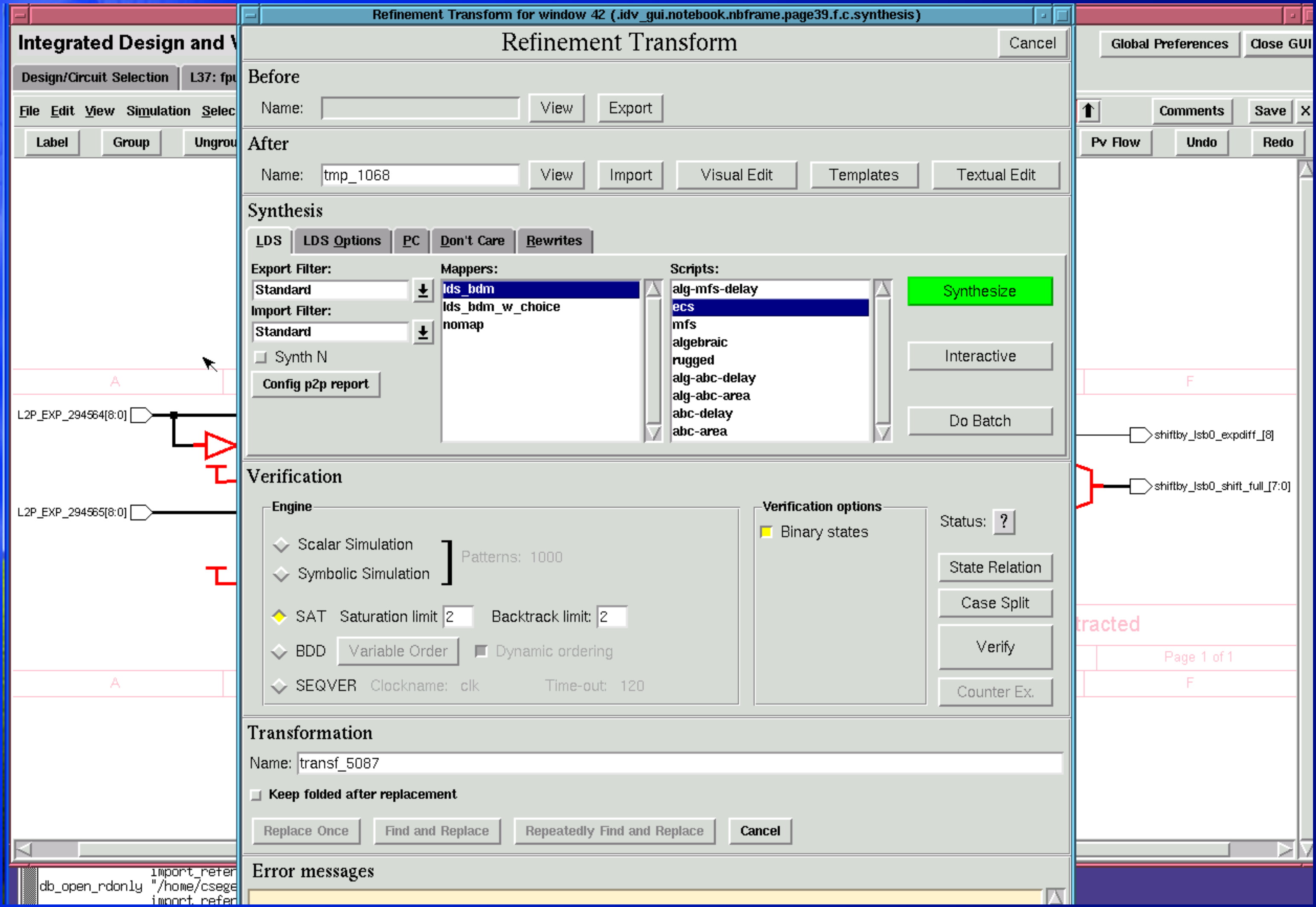
```
Kogge-Stone adder 308 ps -1 mW 931 mu^2 #5072
Kogge-Stone adder 326 ps -1 mW 849 mu^2 #5075
Kogge-Stone adder 419 ps -1 mW 501 mu^2 #5078
Kogge-Stone adder 510 ps -1 mW 501 mu^2 #5081
Kogge-Stone adder 582 ps -1 mW 491 mu^2 #5084
(/home/cseger/Pilot/multiplier/DB_mult_props) add_assoc_10 #91
(/home/cseger/Pilot/multiplier/DB_addition_properties) a+b == b+a
```

Order
☐ Sort by Date

Matching
☐ Exact matching
☐ Remove dangling outputs
☐ Replace N
☐ Sloppy

☐ Keep folded after replacement

Step 4: Use pocket-synthesis to implement remaining logic



Refinement Transform for window 42 (.idv_gui.notebook.nbframe.page39.f.c.synthesis)

Refinement Transform

Before

Name:

After

Name: tmp_1068

Synthesis

Export Filter: Standard

Import Filter: Standard

□ Synth N

Mappers:
- lds_bdm
- lds_bdm_w_choice
- nomap

Scripts:
- alg-mfs-delay
- ecs
- mfs
- algebraic
- rugged
- alg-abc-delay
- alg-abc-area
- abc-delay
- abc-area

Verification

Engine

○ Scalar Simulation

○ Symbolic Simulation

Patterns: 1000

● SAT Saturation limit 2 Backtrack limit: 2

○ BDD

□ Dynamic ordering

○ SEQVER Clockname: clk Time-out: 120

Verification options

□ Binary states

Status: ?

Transformation

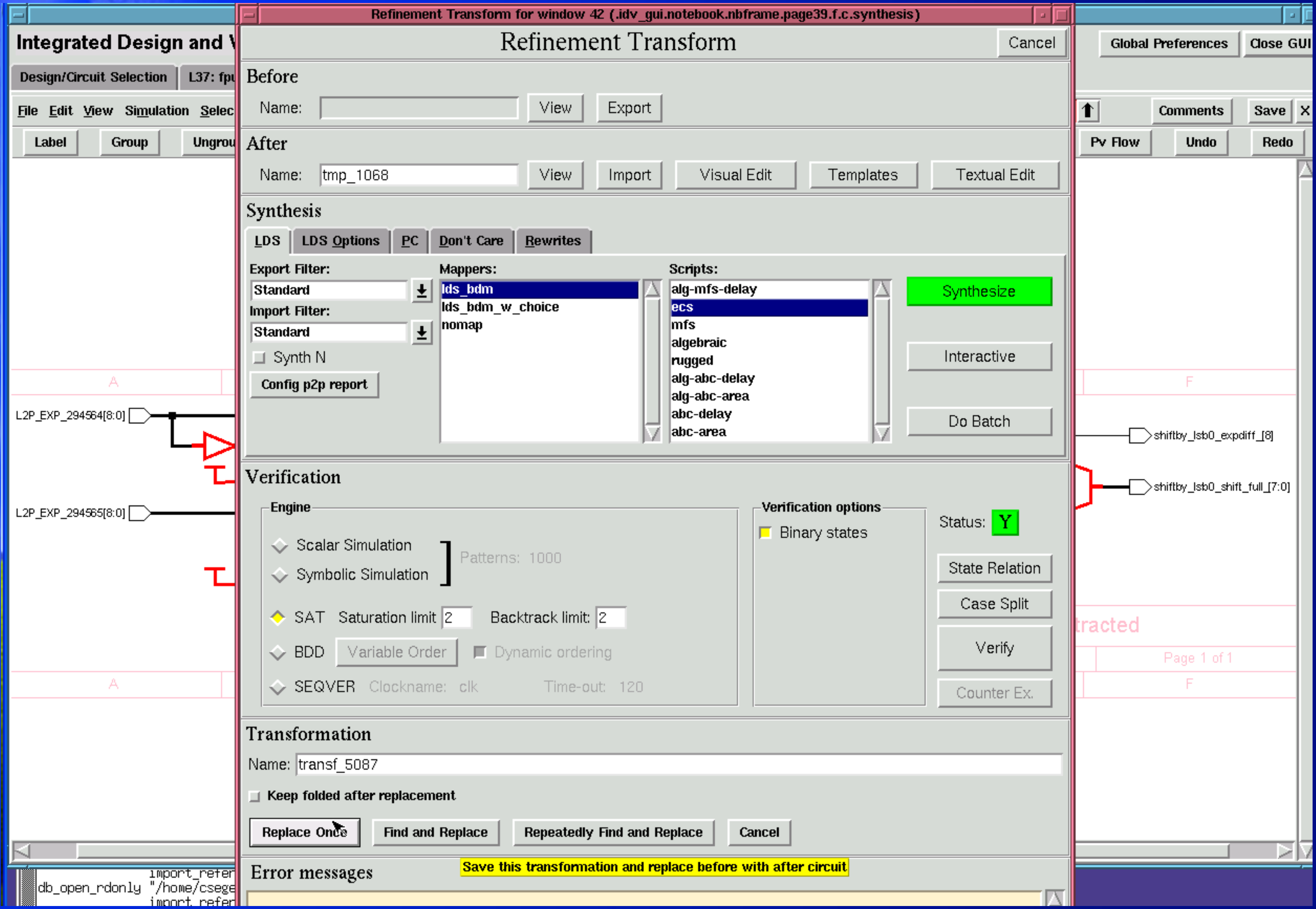
Name: transf_5087

□ Keep folded after replacement

Error messages

```
db_open_rdonly "/home/csege
import_refer
import_refer
```

Step 5: and run FEV on the result before using it inside IDV



Refinement Transform for window 42 (.idv_gui.notebook.nbframe.page39.f.c.synthesis)

Refinement Transform

Before

Name:

After

Name: tmp_1068

Synthesis

Export Filter: Standard

Import Filter: Standard

□ Synth N

Mappers:
- lds_bdm
- lds_bdm_w_choice
- nomap

Scripts:
- alg-mfs-delay
- ecs
- mfs
- algebraic
- rugged
- alg-abc-delay
- alg-abc-area
- abc-delay
- abc-area

Verification

Engine
- ○ Scalar Simulation
- ○ Symbolic Simulation

Patterns: 1000

- ● SAT Saturation limit 2 Backtrack limit: 2
- ○ BDD
- ☑ Dynamic ordering
- ○ SEQVER Clockname: clk Time-out: 120

Verification options

□ Binary states

Status: Y

Transformation

Name: transf_5087

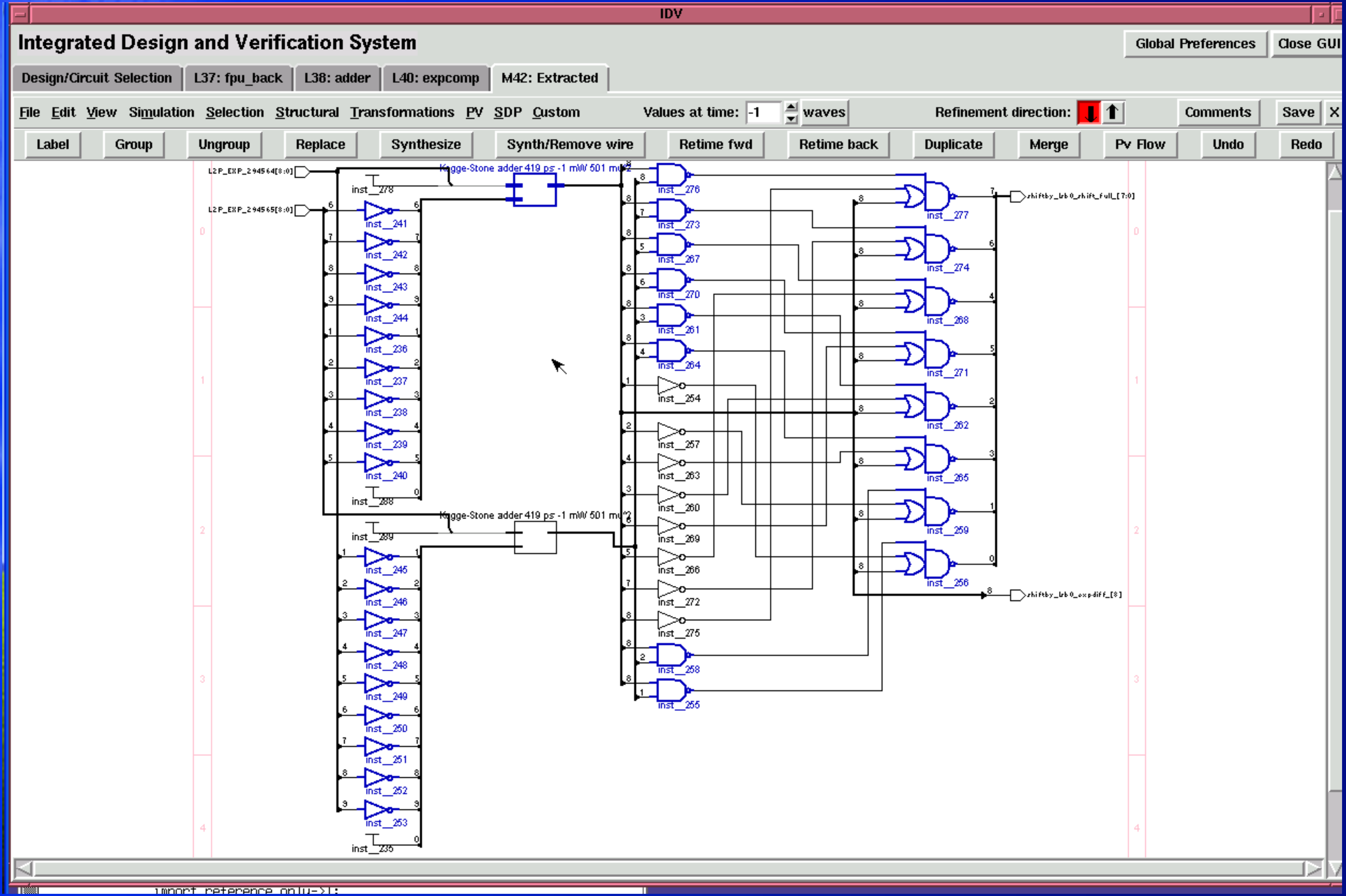
□ Keep folded after replacement

Save this transformation and replace before with after circuit

Error messages

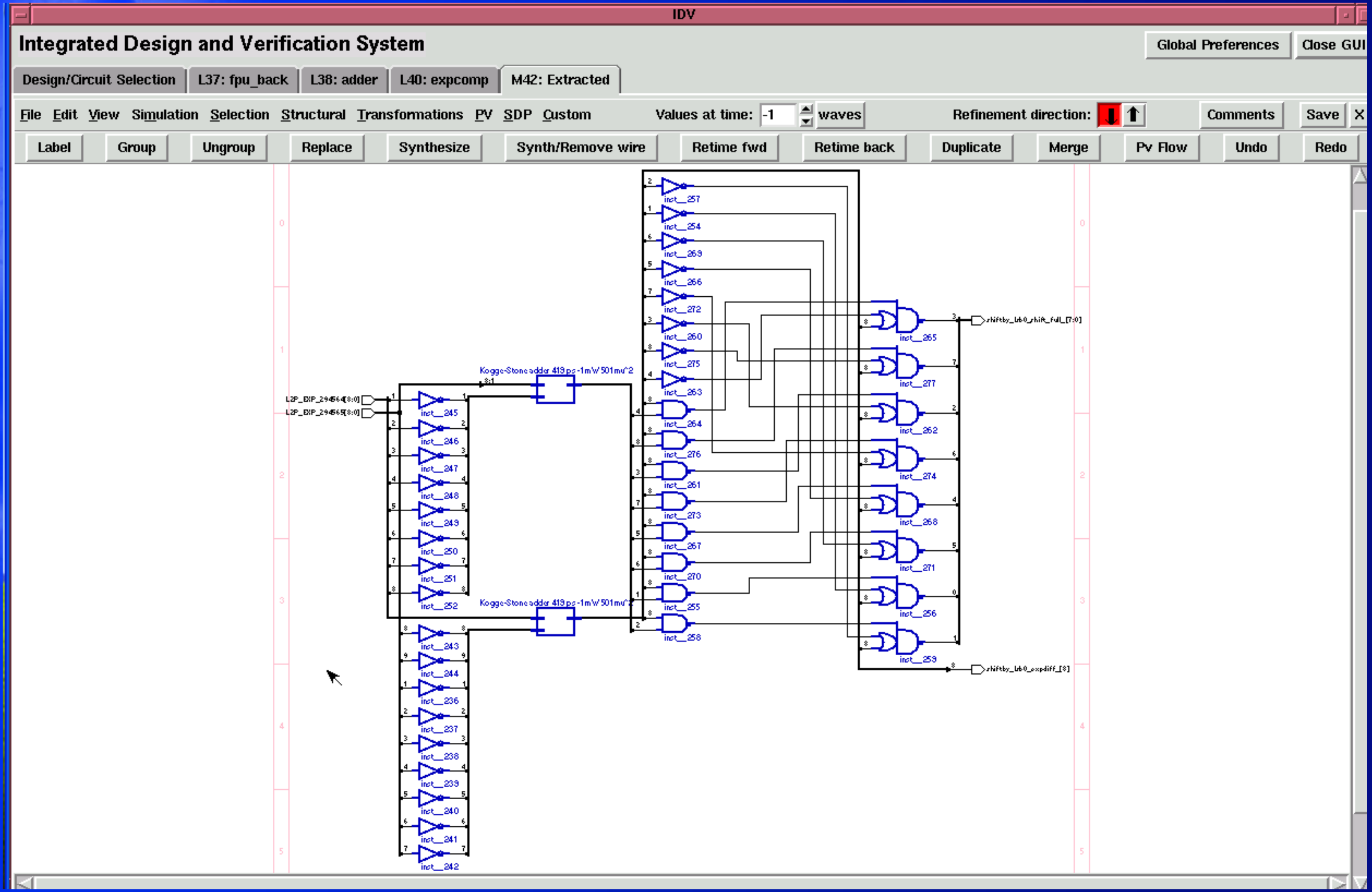
```
db_open_rdonly "/home/csege
import_refer
import_refer
```

This yields

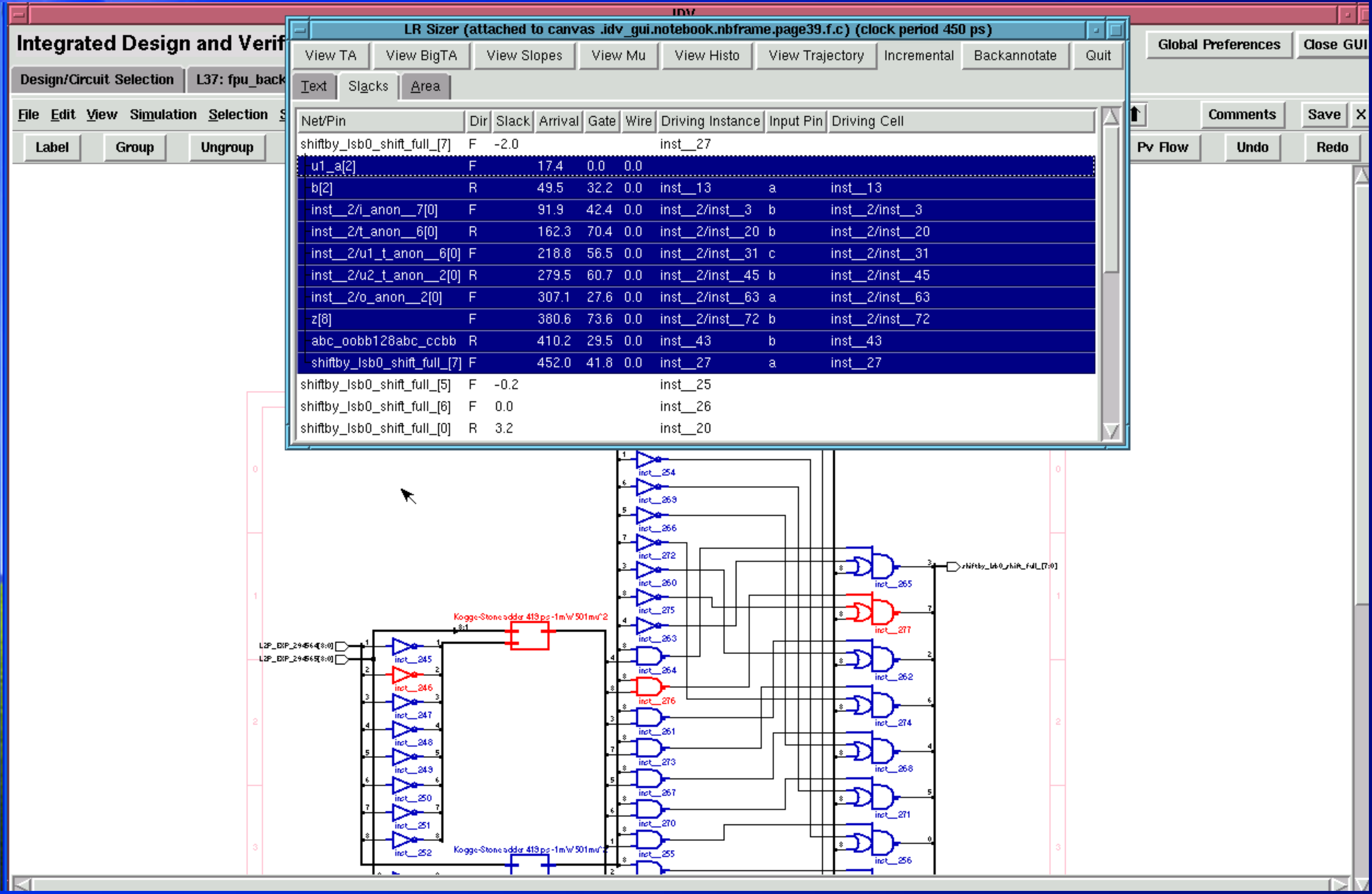


```
import reference only=>1;
```

Step 6: Perform constant propagation yielding

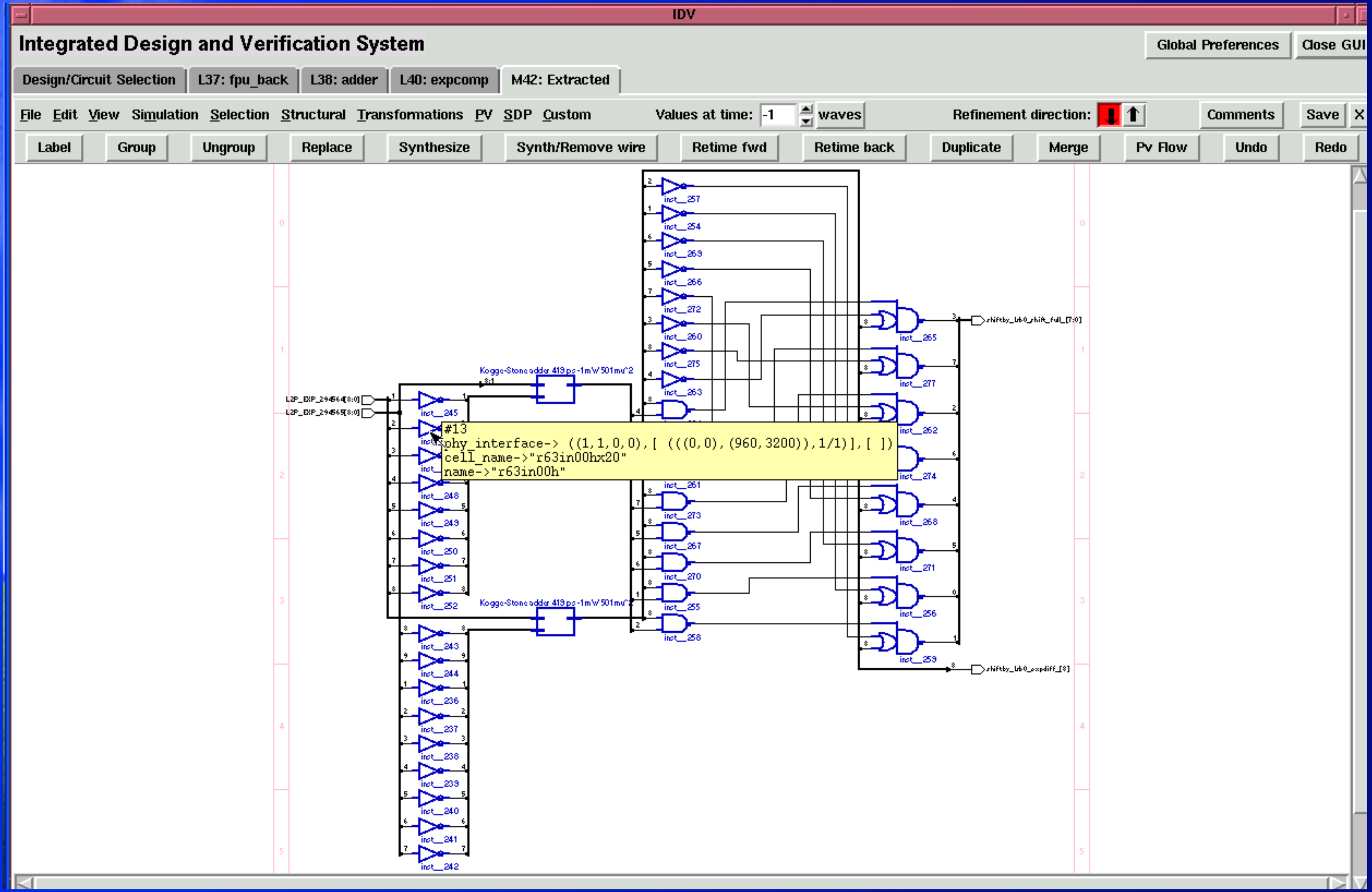


Step 7: Size the cells based on timing/power/area requirements.



Net/Pin	Dir	Slack	Arrival	Gate	Wire	Driving Instance	Input Pin	Driving Cell
shiftby_lsb0_shift_full_[7]	F	-2.0				inst__27		
u1_a[2]	F		17.4	0.0	0.0			
b[2]	R		49.5	32.2	0.0	inst__13	a	inst__13
inst__2/i_anon__7[0]	F		91.9	42.4	0.0	inst__2/inst__3	b	inst__2/inst__3
inst__2/t_anon__6[0]	R		162.3	70.4	0.0	inst__2/inst__20	b	inst__2/inst__20
inst__2/u1_t_anon__6[0]	F		218.8	56.5	0.0	inst__2/inst__31	c	inst__2/inst__31
inst__2/u2_t_anon__2[0]	R		279.5	60.7	0.0	inst__2/inst__45	b	inst__2/inst__45
inst__2/o_anon__2[0]	F		307.1	27.6	0.0	inst__2/inst__63	a	inst__2/inst__63
z[8]	F		380.6	73.6	0.0	inst__2/inst__72	b	inst__2/inst__72
abc_oobb128abc_ccbb	R		410.2	29.5	0.0	inst__43	b	inst__43
shiftby_lsb0_shift_full_[7]	F		452.0	41.8	0.0	inst__27	a	inst__27
shiftby_lsb0_shift_full_[5]	F	-0.2				inst__25		
shiftby_lsb0_shift_full_[6]	F	0.0				inst__26		
shiftby_lsb0_shift_full_[0]	R	3.2				inst__20		

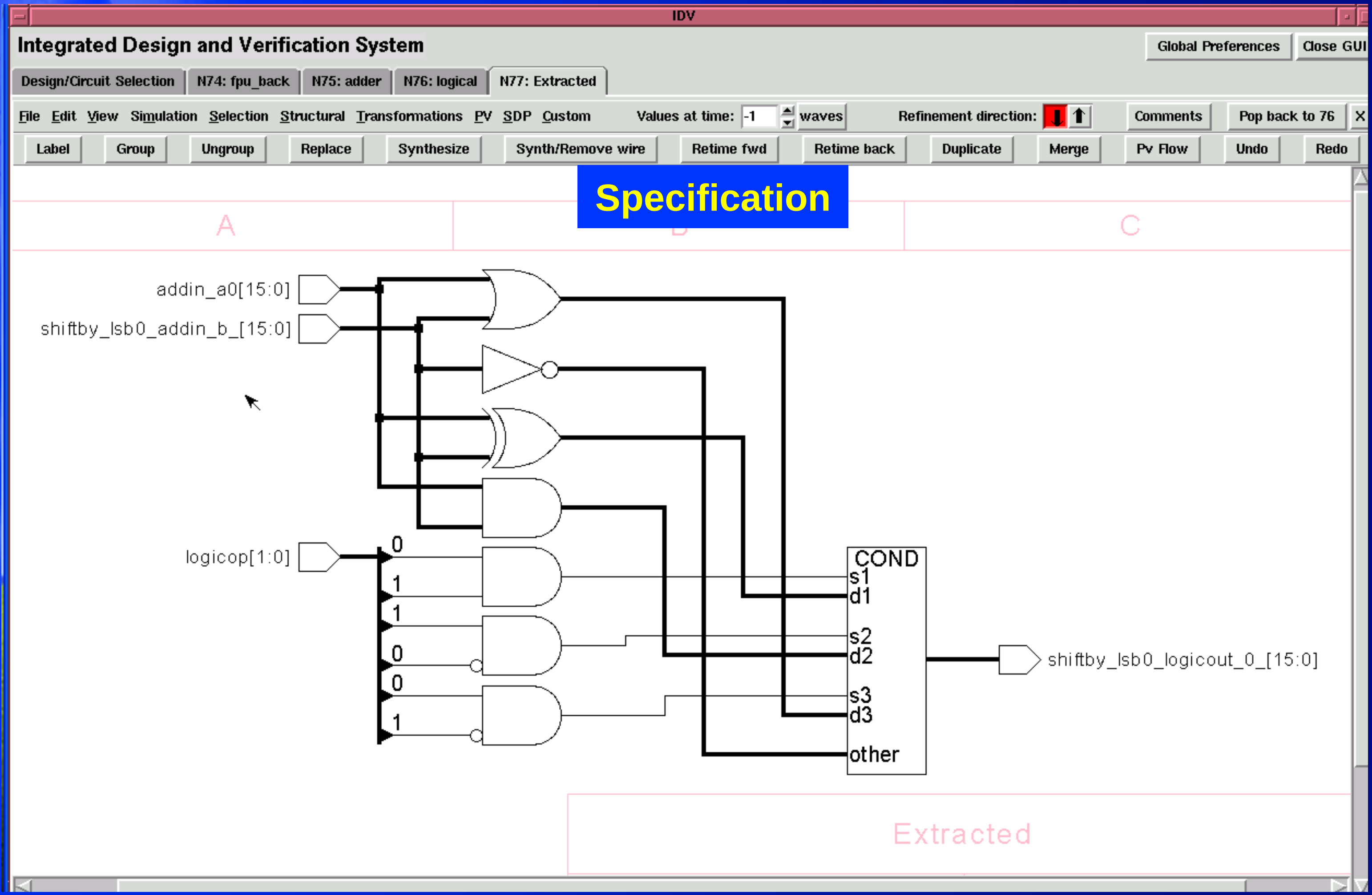
Step 8: After converging choose cells according to sizer and the mid-level design phase is over



```
#13
phy_interface-> ((1,1,0,0),[ (((0,0),(960,3200)),1/1)],[ ])
cell_name->"r63in00hx20"
name->"r63in00h"
```

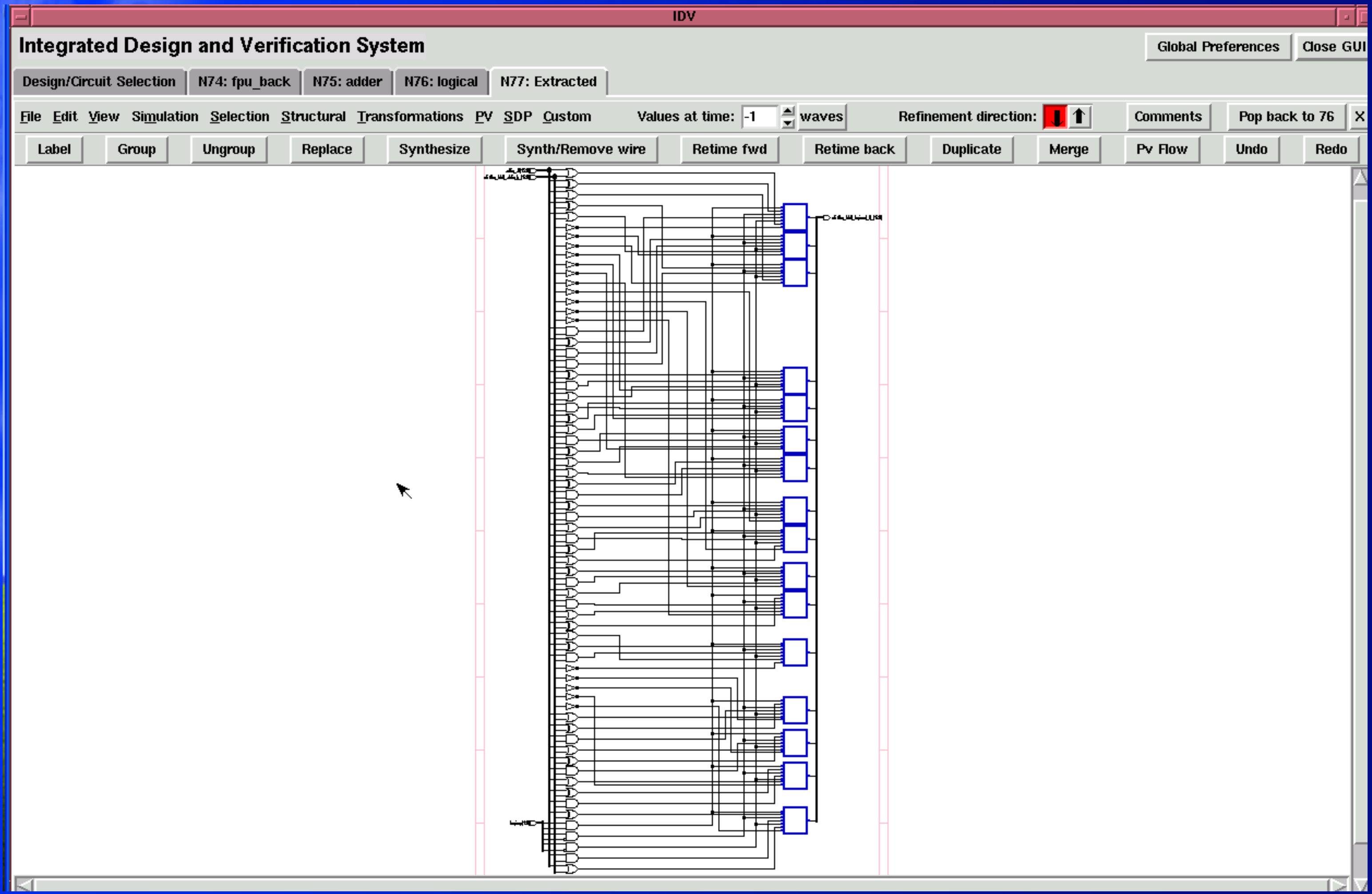
Low-Level Physical Design

Low-level/Physical Design in IDV



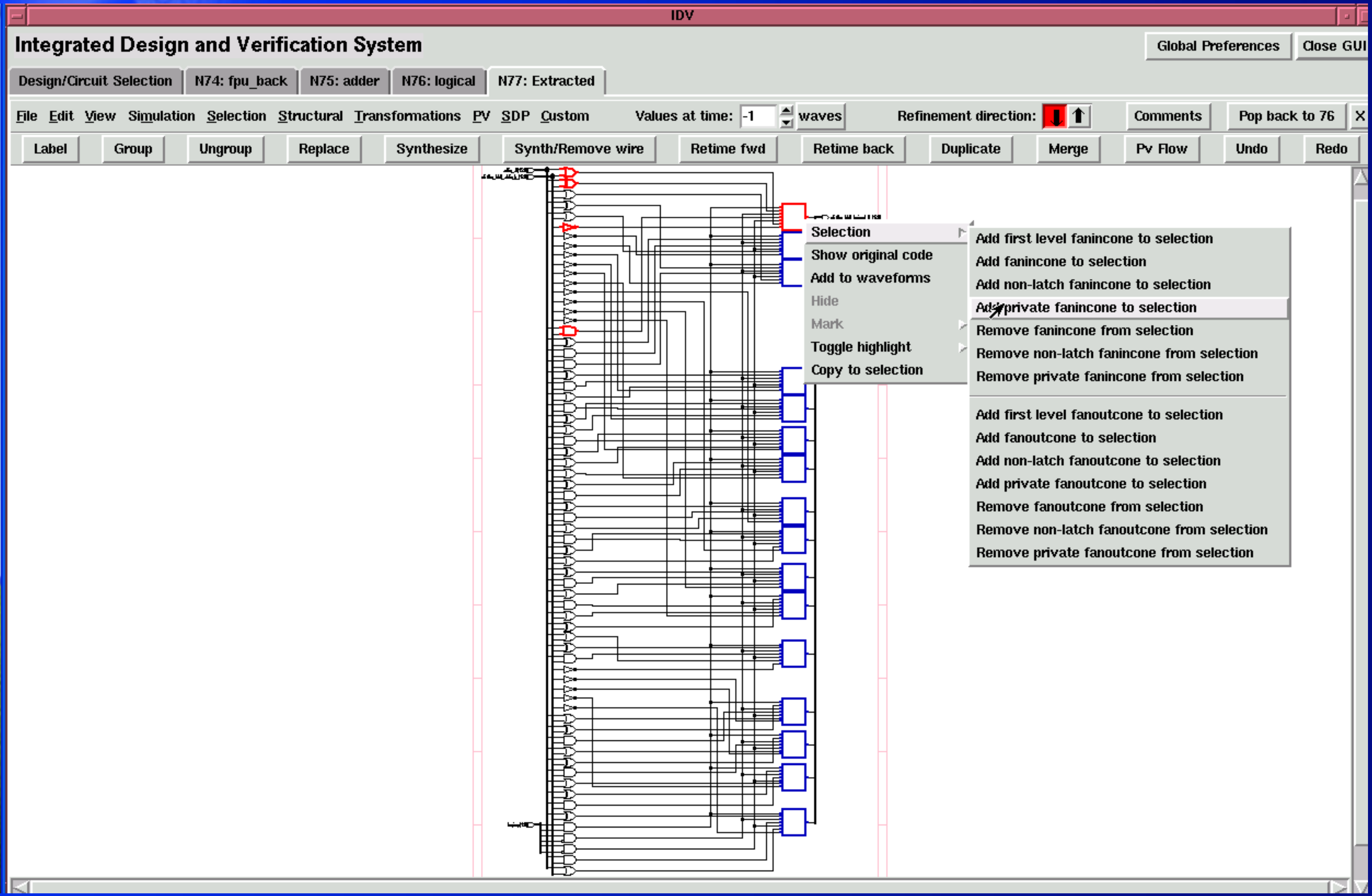
Specification

Step 1: Split into bit-slices



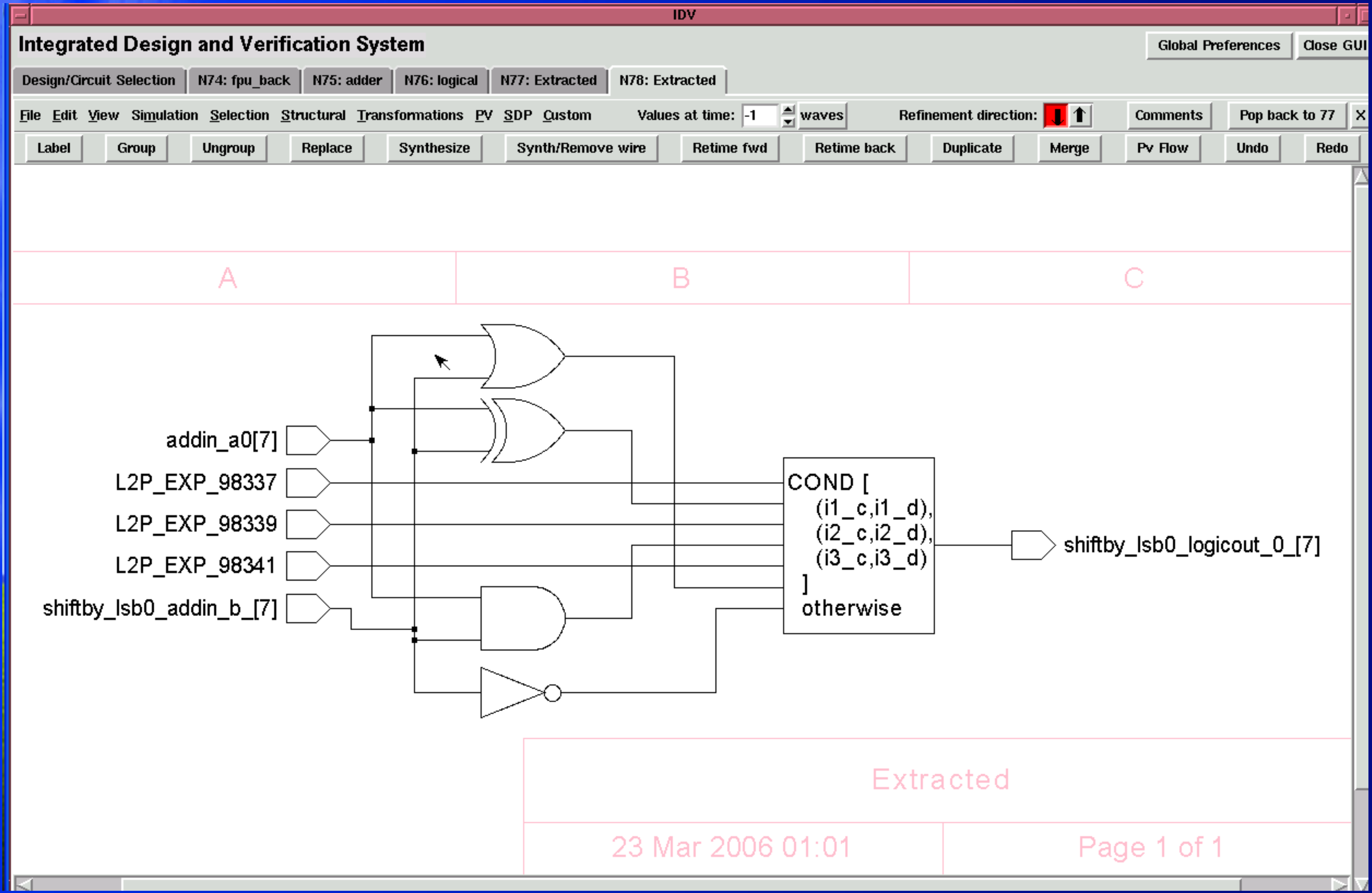
Values at time: -1

Step 2: Select the private fanin-cone, i.e., logic feeding only this output

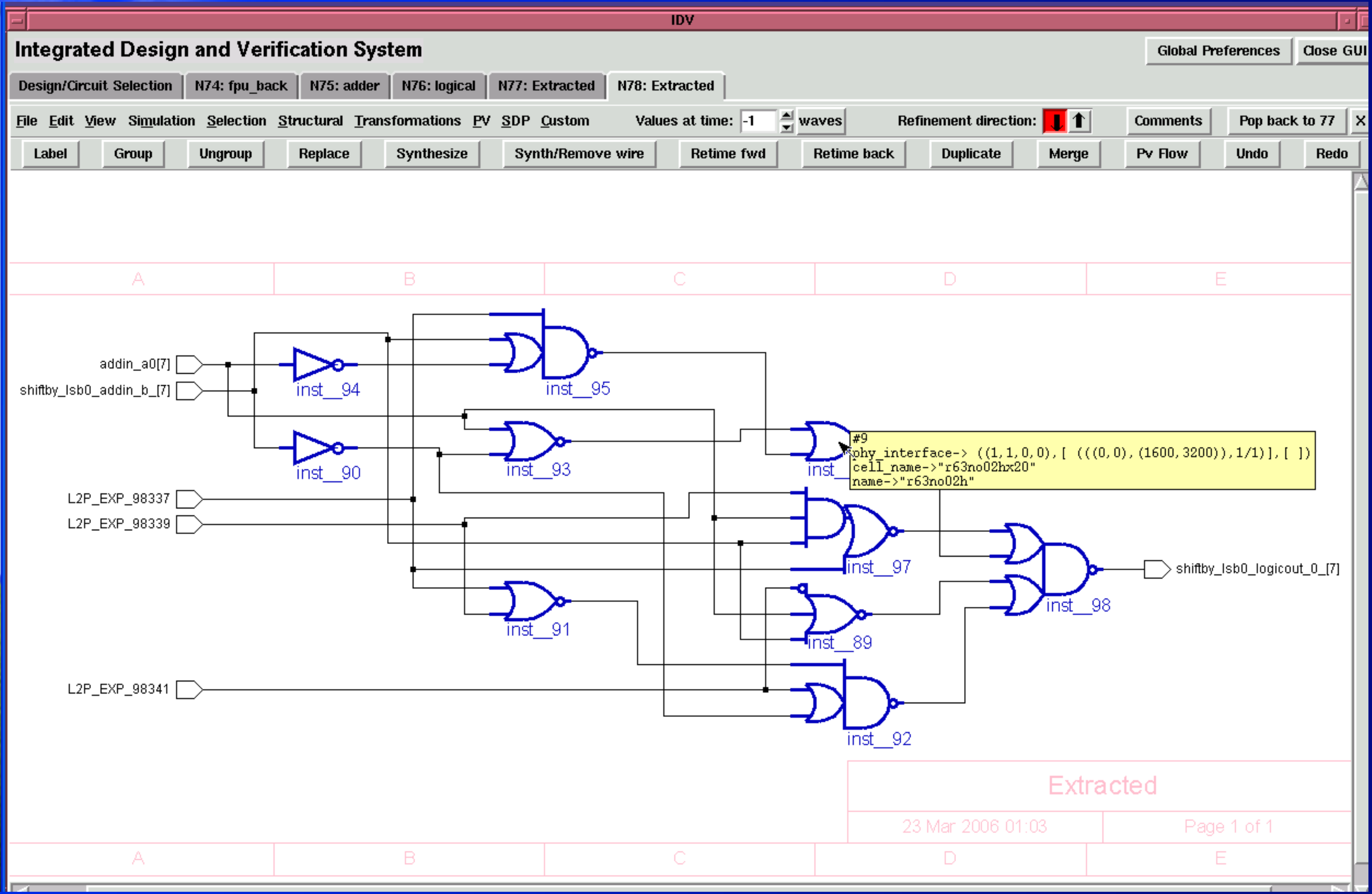


IDV
Integrated Design and Verification System
Global Preferences
Close GUI
Design/Circuit Selection
N74: fpu_back
N75: adder
N76: logical
N77: Extracted
File Edit View Simulation Selection Structural Transformations PV SDP Custom
Values at time: -1
waves
Refinement direction:
Comments
Pop back to 76
X
Label
Group
Ungroup
Replace
Synthesize
Synth/Remove wire
Retime fwd
Retime back
Duplicate
Merge
Pv Flow
Undo
Redo
Selection
Show original code
Add to waveforms
Hide
Mark
Toggle highlight
Copy to selection
Add first level fanincone to selection
Add fanincone to selection
Add non-latch fanincone to selection
Add private fanincone to selection
Remove fanincone from selection
Remove non-latch fanincone from selection
Remove private fanincone from selection
Add first level fanoutcone to selection
Add fanoutcone to selection
Add non-latch fanoutcone to selection
Add private fanoutcone to selection
Remove fanoutcone from selection
Remove non-latch fanoutcone from selection
Remove private fanoutcone from selection

Step 3: Push into the single bit

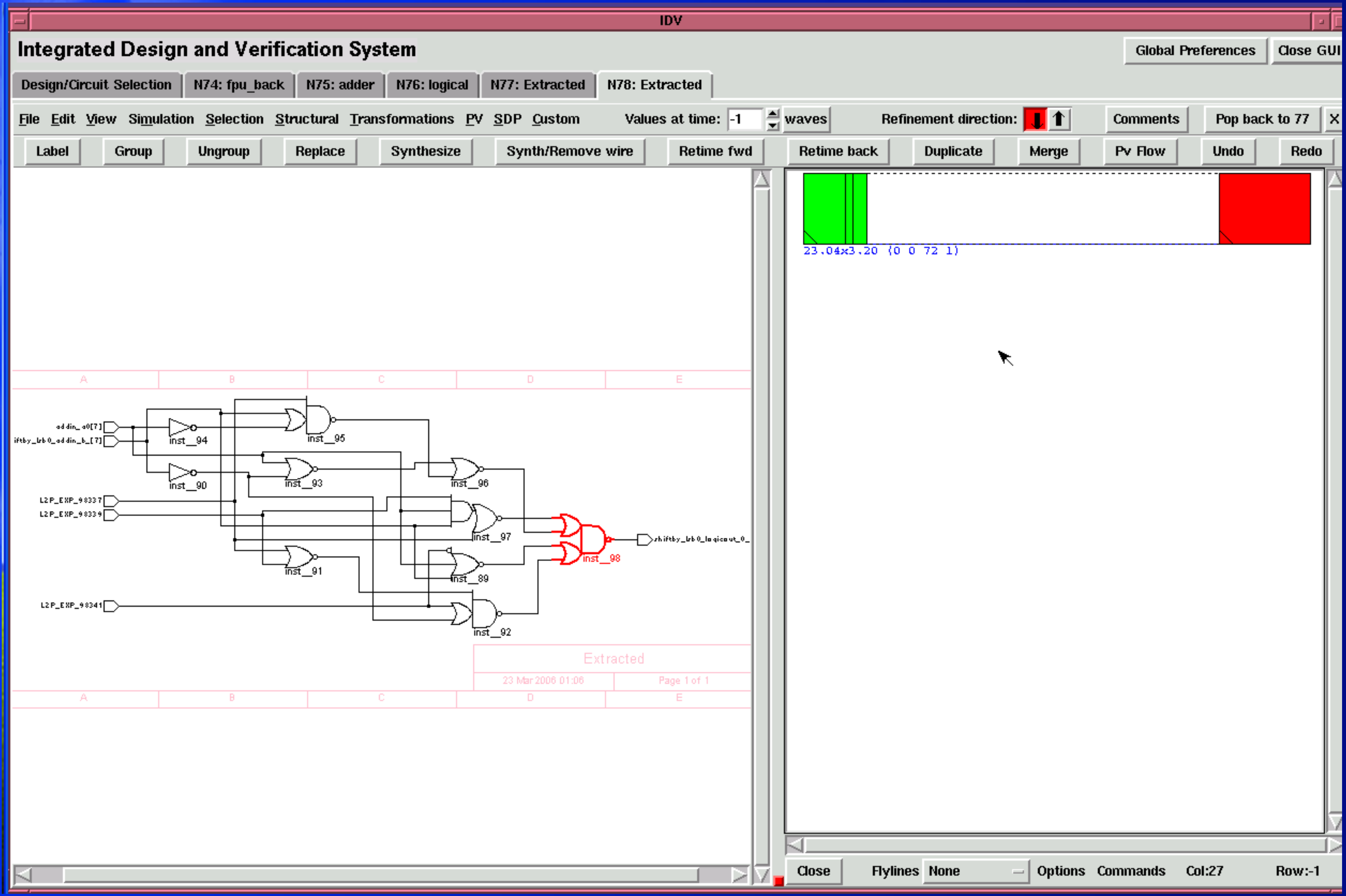


Step 4: Design one bit slice (both mapping to cells & sizing)

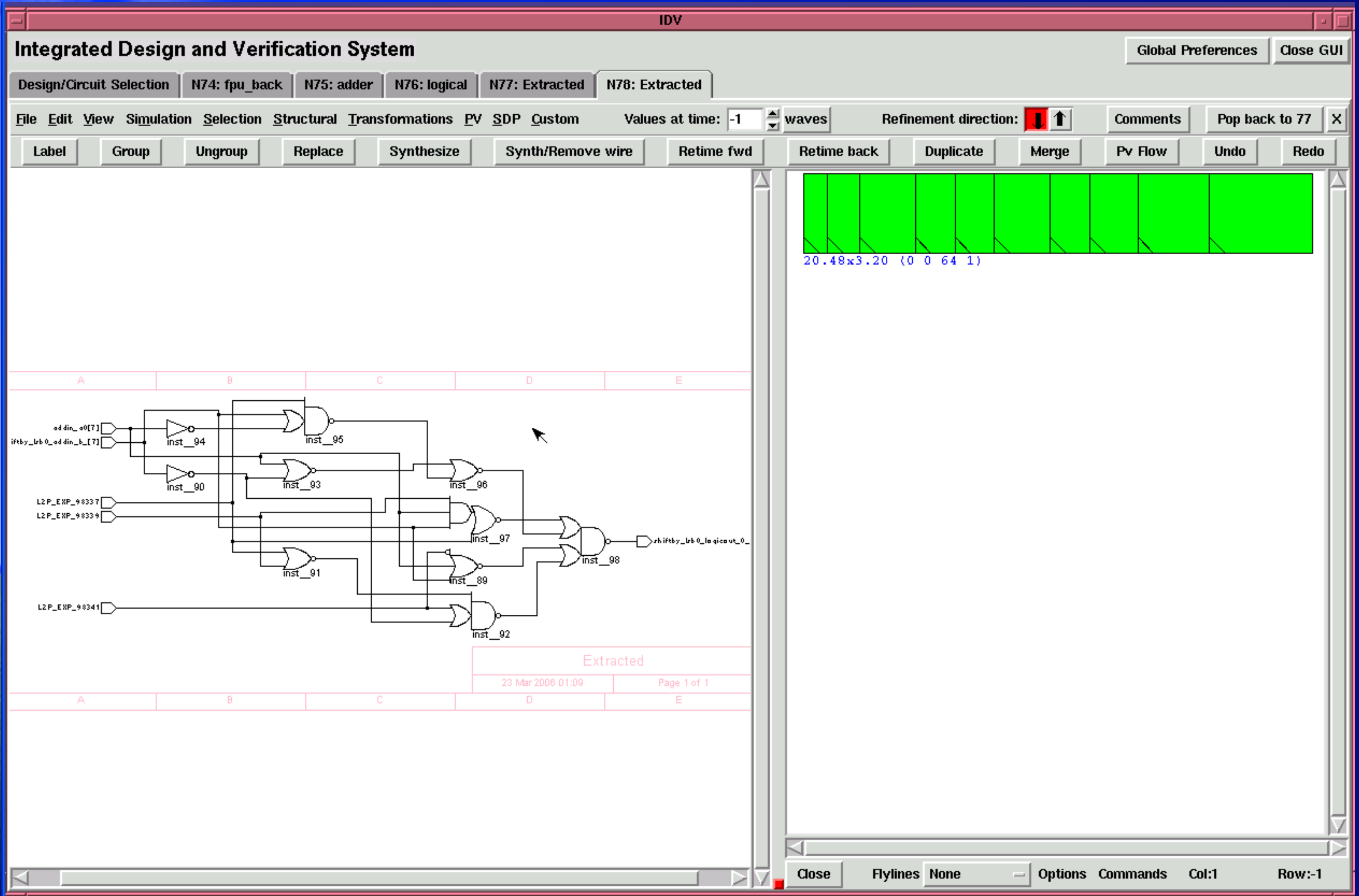


```
#9
phy_interface-> ((1,1,0,0),[ (((0,0),(1600,3200)),1/1)],[ ])
cell_name->"r63no02hx20"
name->"r63no02h"
```

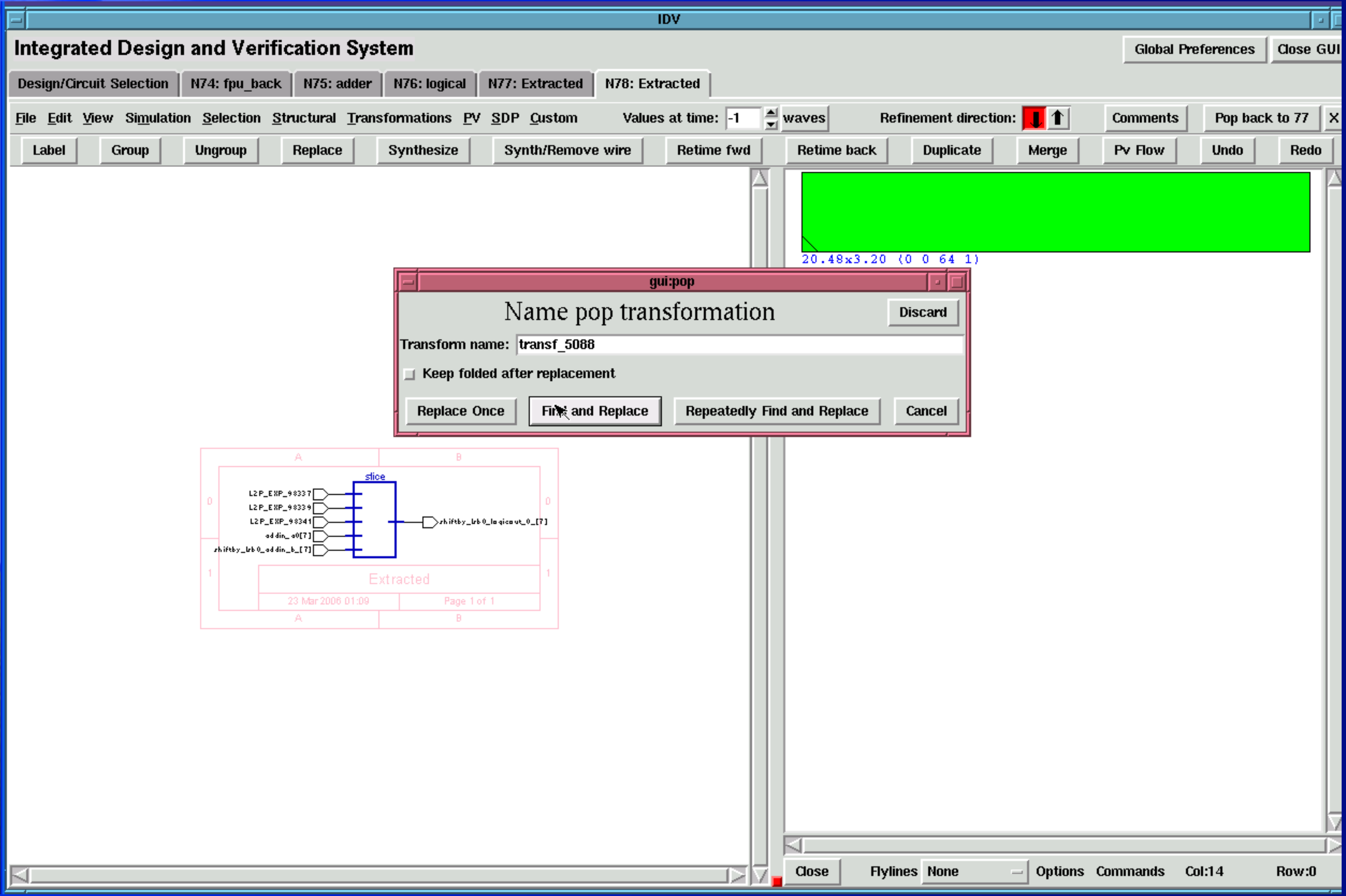
Step 5: Start placing the cells



Step 6: Finish placing the single bit-slice



Step 7: Save transformation and use it in a find-and-replace operation



IDV

Integrated Design and Verification System

gui:pop

Name pop transformation

Transform name: transf_5088

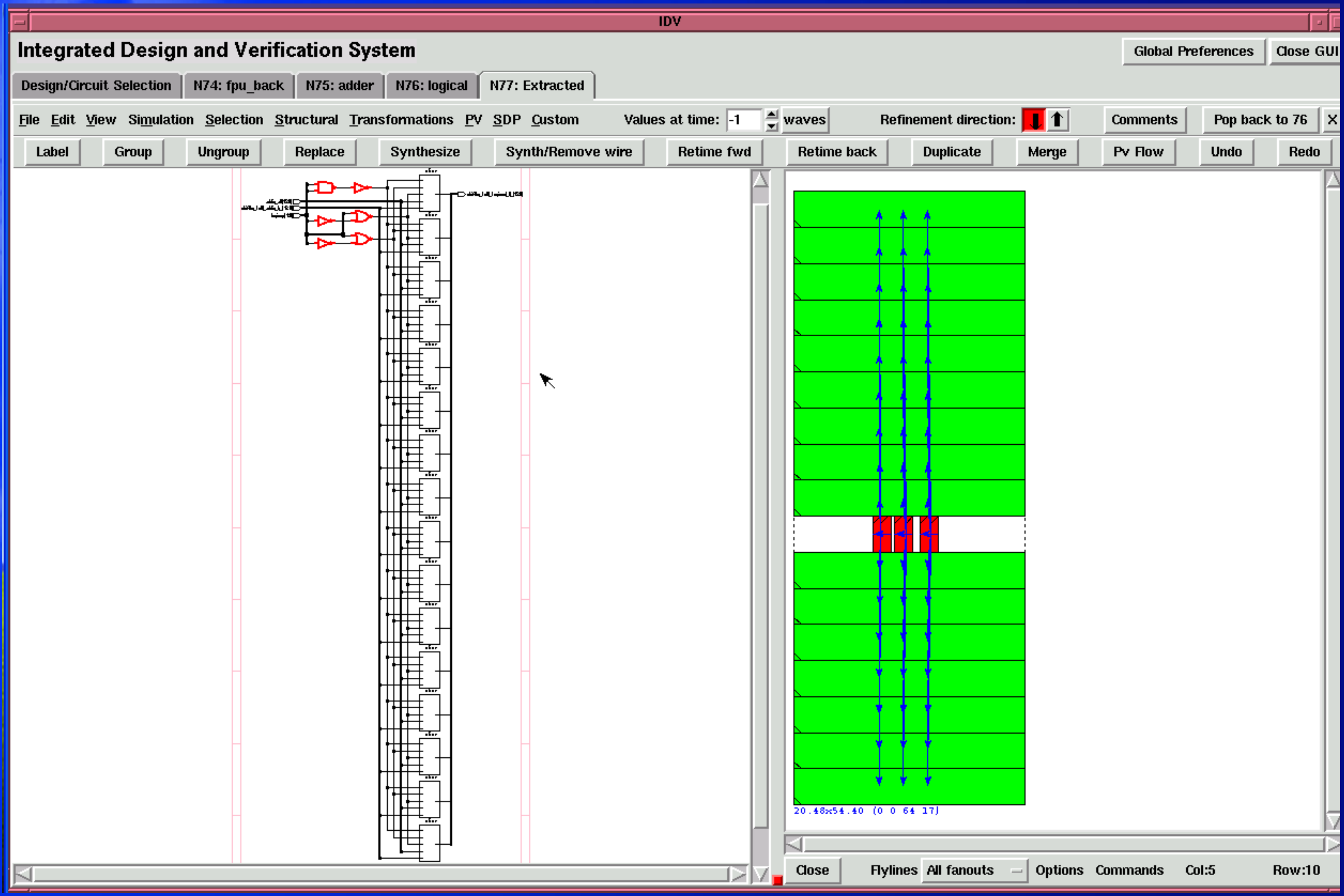
□ Keep folded after replacement

Values at time: -1

20.48x3.20 (0 0 64 1)

Col:14 Row:0

Step 8: Place the bit-slices according to output wire name and auto-place the decoder logic.



Property Handling in IDV

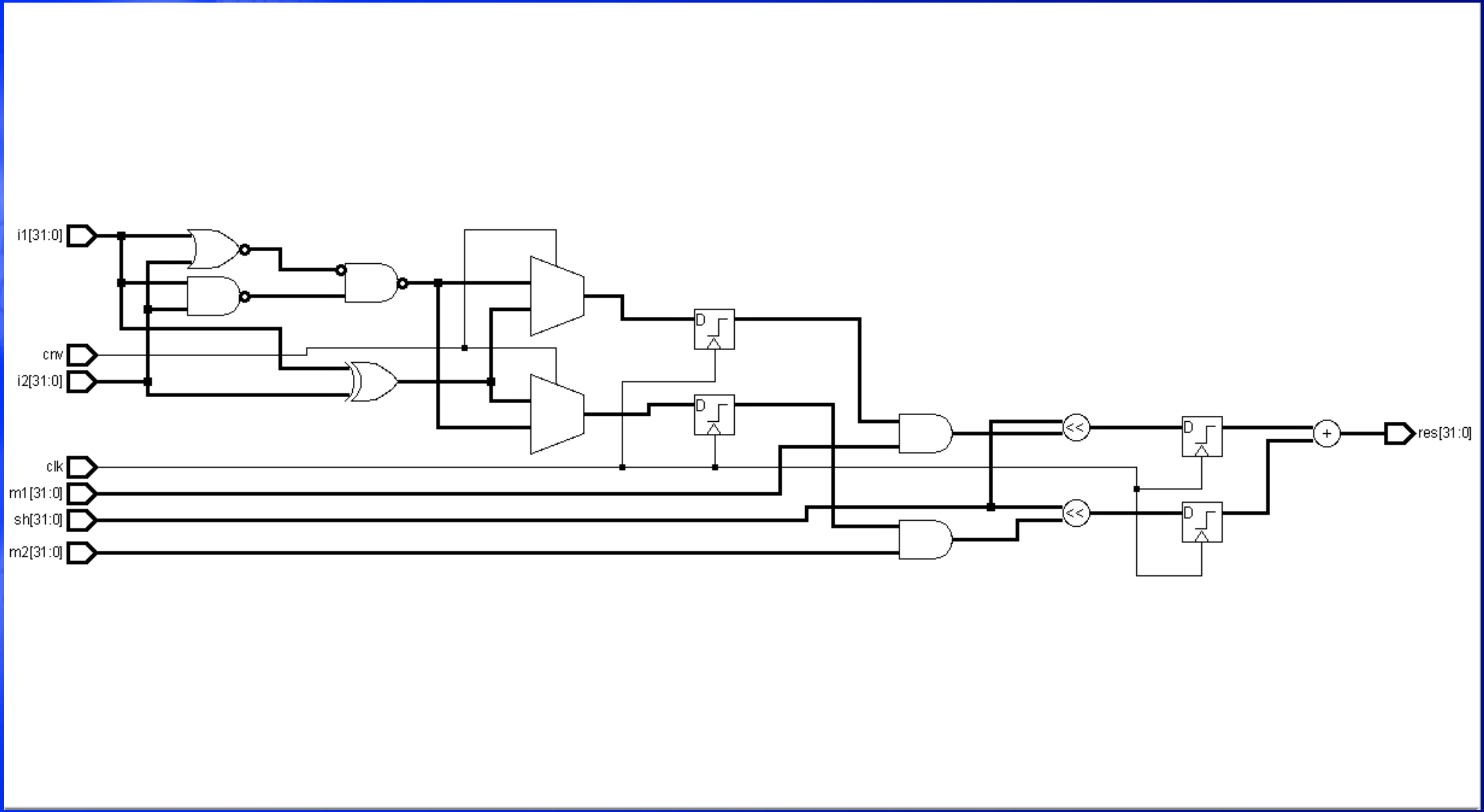
Properties

- Taking advantage of properties in the design process is often critical to reach a desired outcome
  - E.g., complex logic can be drastically simplified if some property is known to hold
- Two types of properties:
  - Assumptions
    - E.g., ”these inputs will always be mutually exclusive”
  - Don’t cares:
    - E.g., “the result will never be used if the valid bit is false”
- Properties are often treated as second class citizens
  - Managed in different languages, maintained differently, verified correct/valid only late in the design process, etc.
  - Many synthesis tools can only take advantage of “local” properties (if any!)
    - E.g., properties stated/proven several pipe stages away are rarely (ever?) visible/used by synthesis tools.

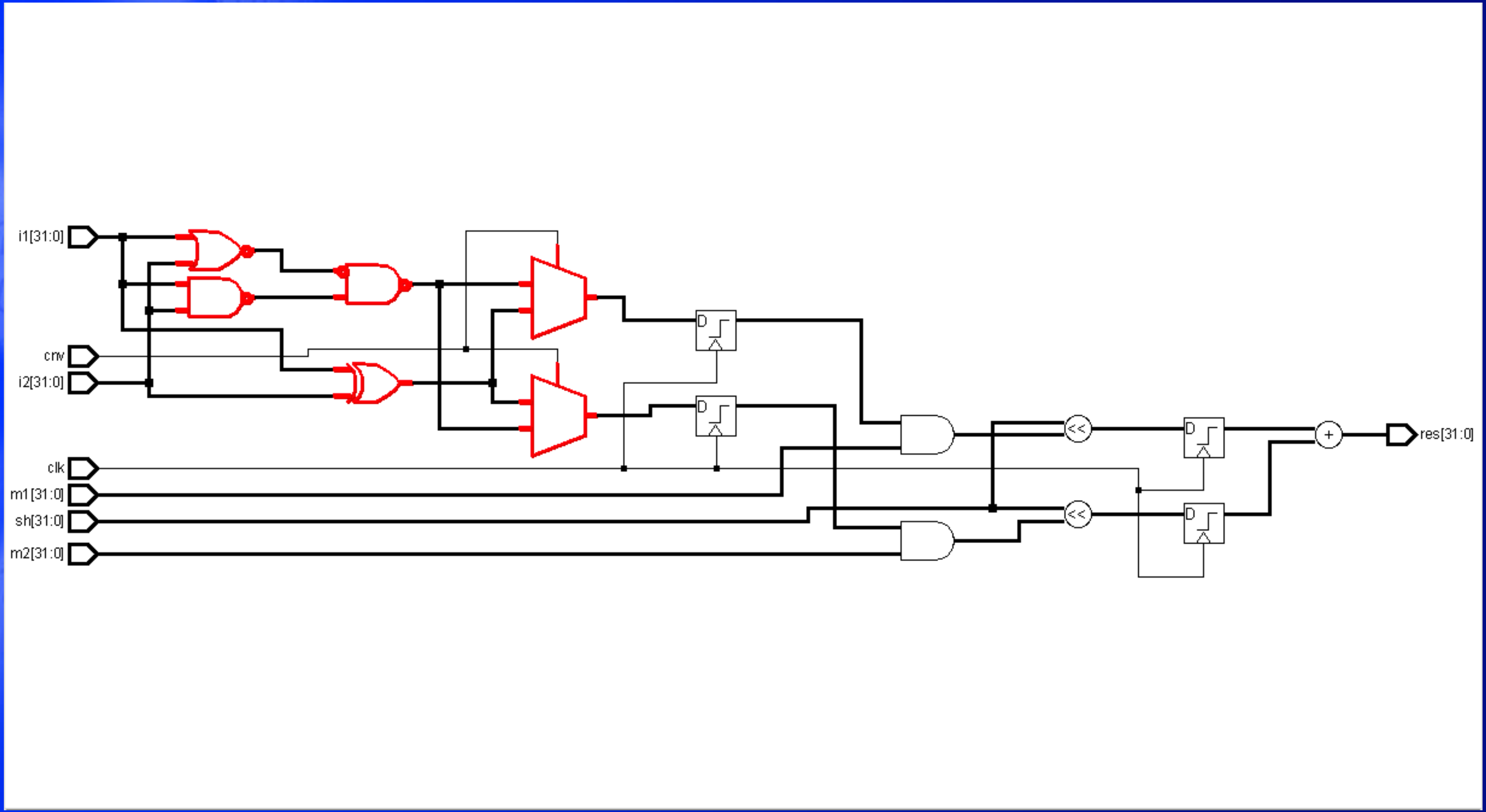
Assumptions in IDV

- Assumptions are treated exactly the same as hardware components.
  - An assumption is a finite state machine with some inputs and a “ok” signal.
  - Modeled as a combinational assertion together with some extra latches/flops and logic to create a checker.
  - Assumptions are visualized with the corresponding logic and can be transformed like other components, e.g., they can be:
    - Duplicated
    - Moved in the design hierarchy
    - Retimed either forward or backwards (usual restrictions)
  - Refinement verification both uses and verifies all properties in a spec/imp pair.
- Assumptions come from two main sources:
  - In the original HLM capturing the environment (input assumptions)
  - Implied from up-streams logic
- Assumptions can be added either manually or computed (semi-) automatically

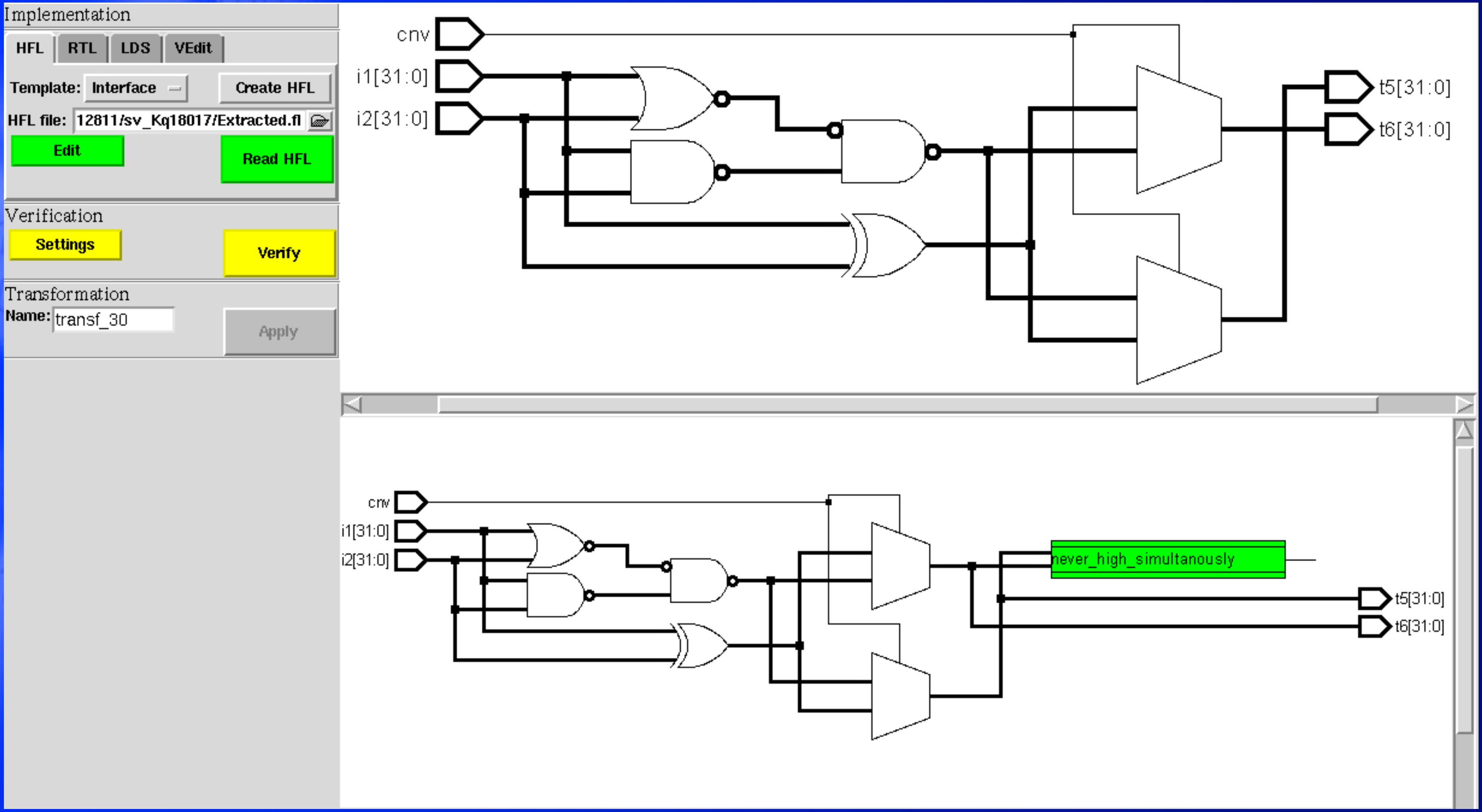
Example



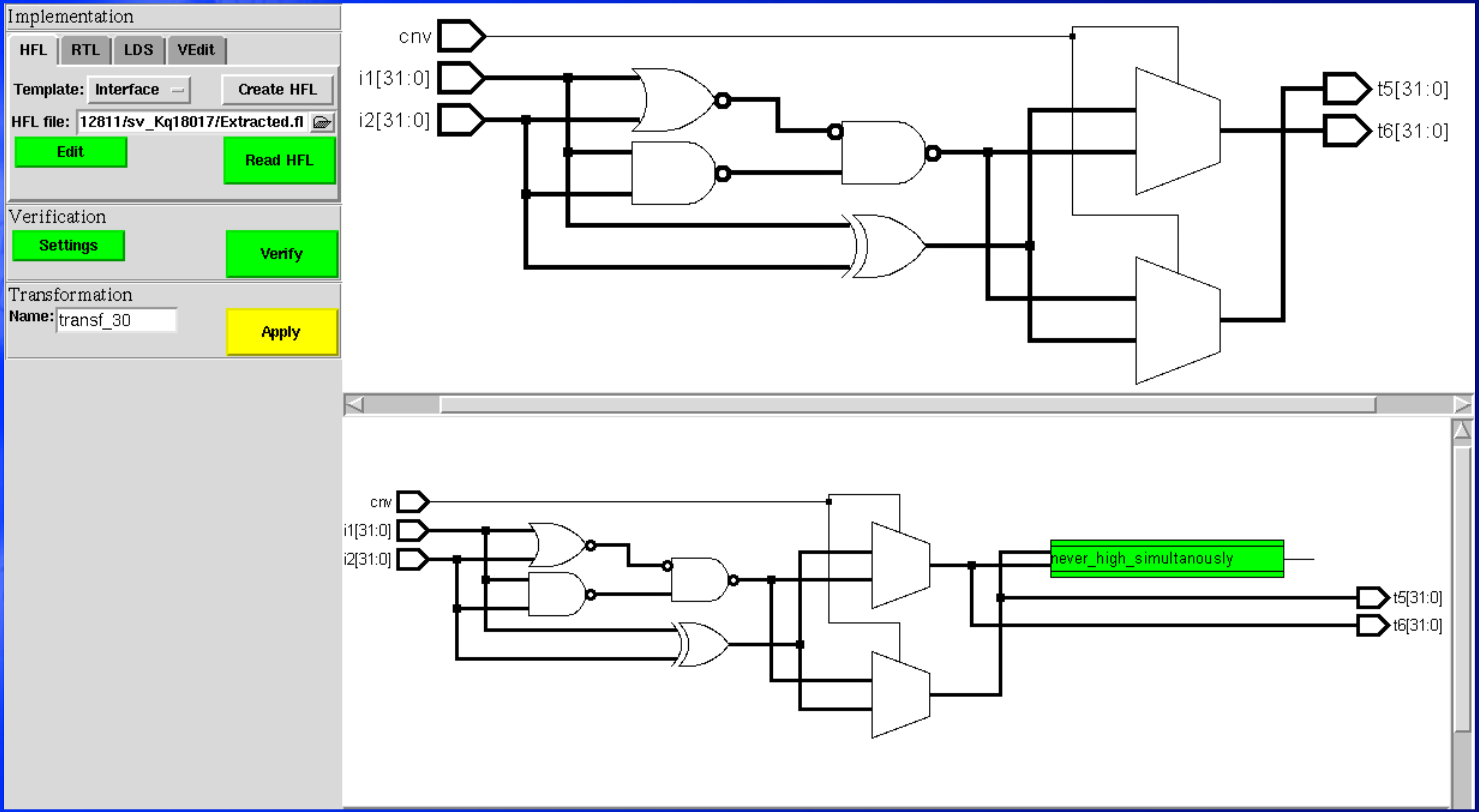
Select Logic Implying Property



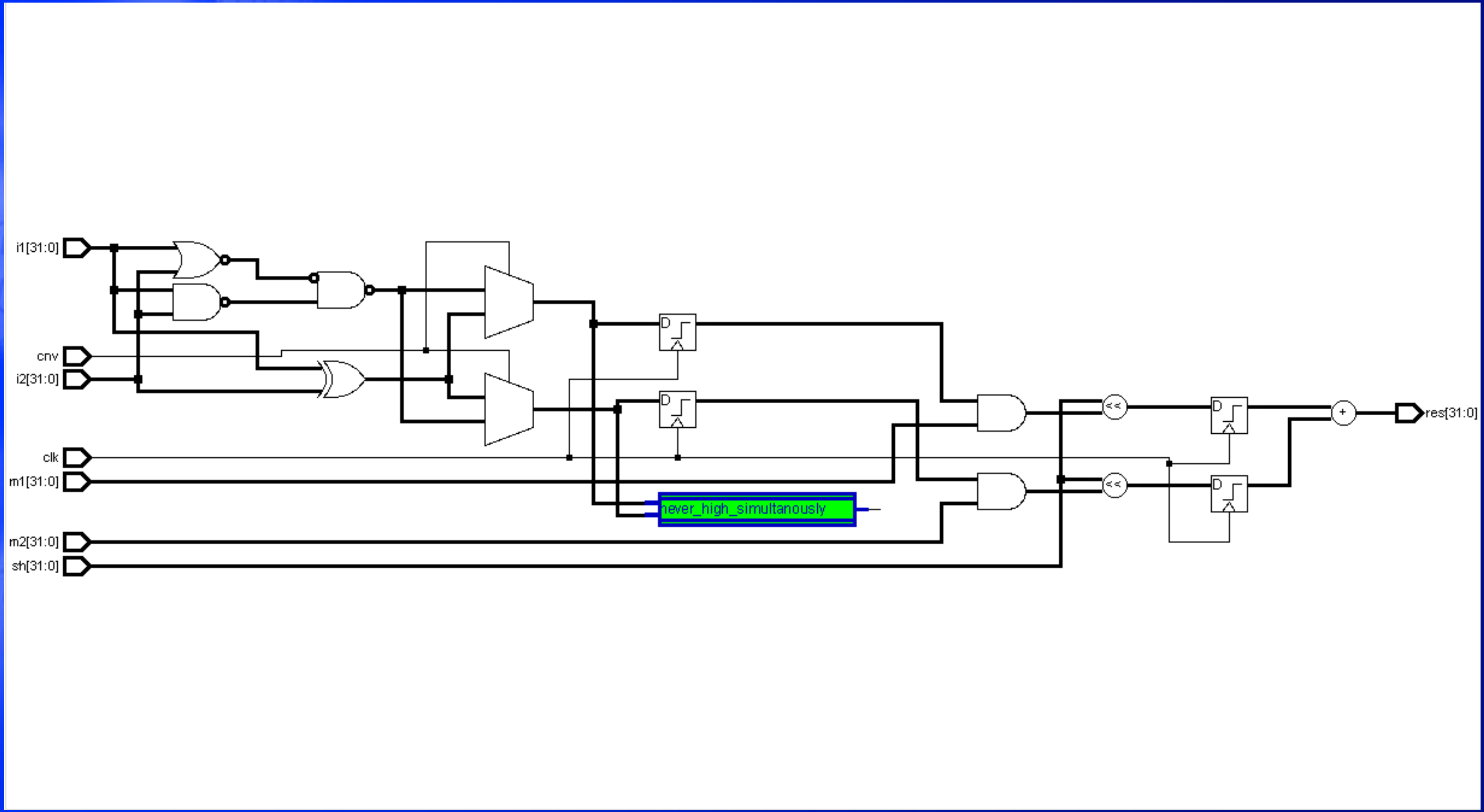
Add a Property Manually



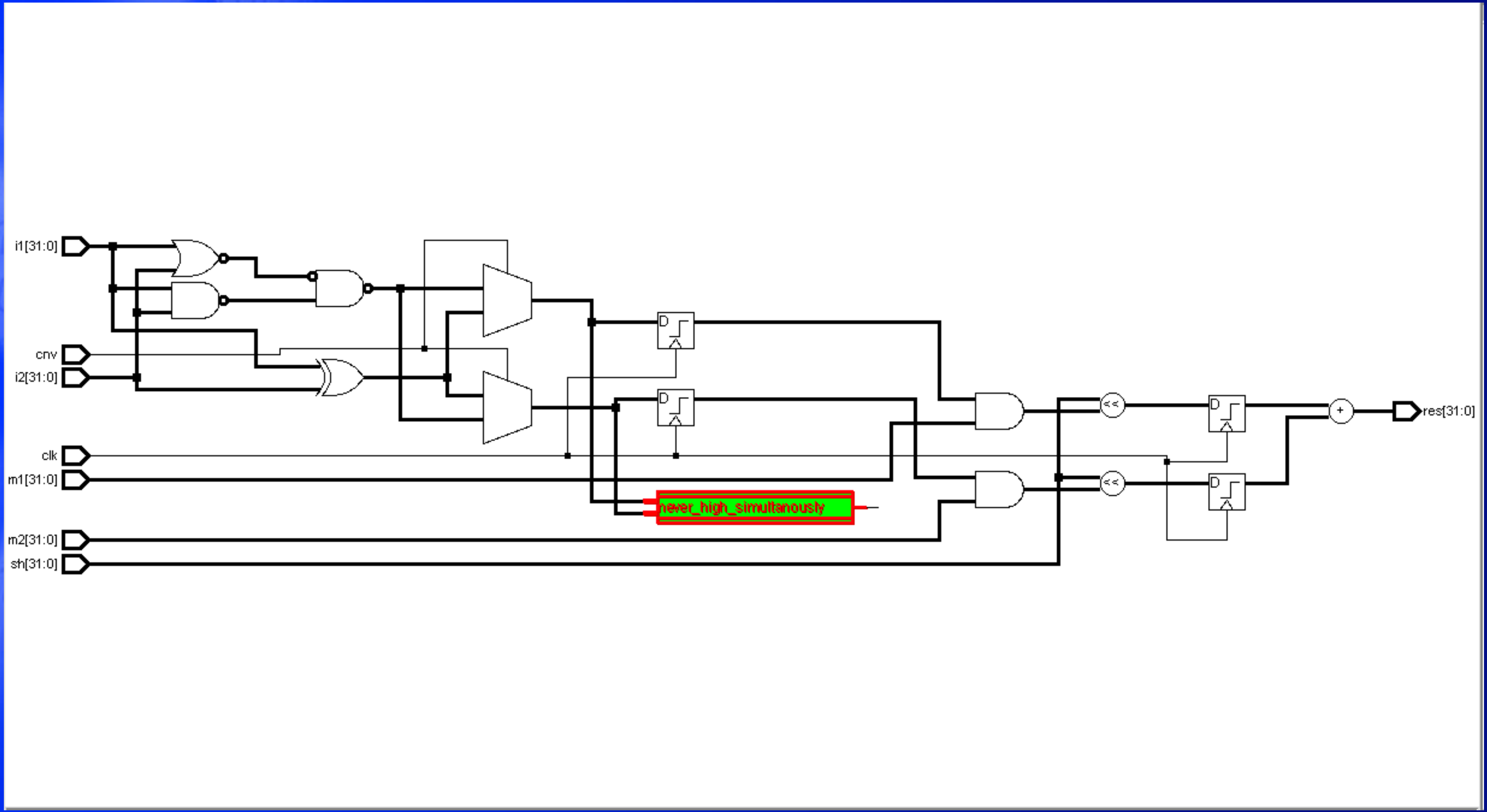
Verify the Validity of Property



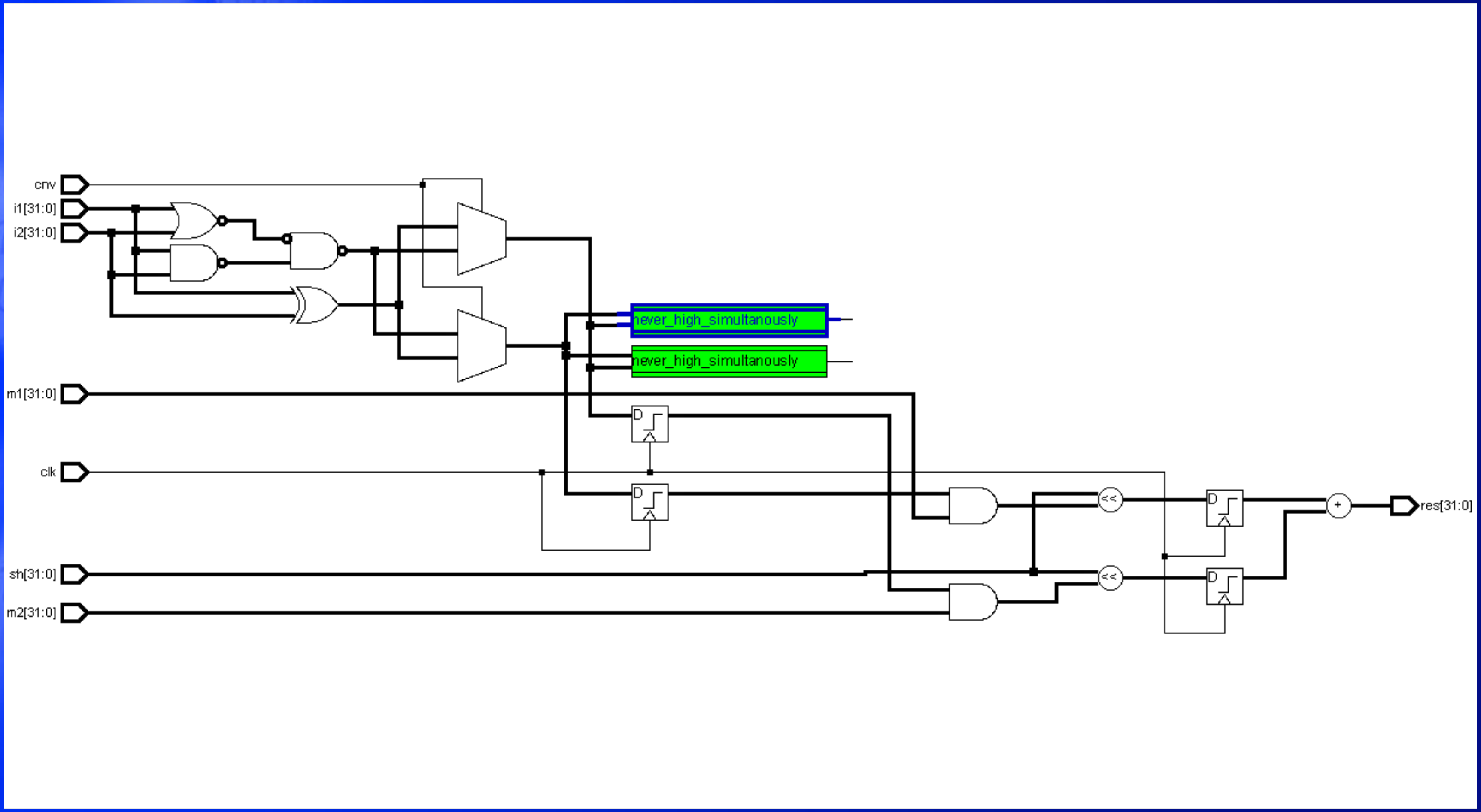
Replace Originally Selected Logic With Same Logic + Property



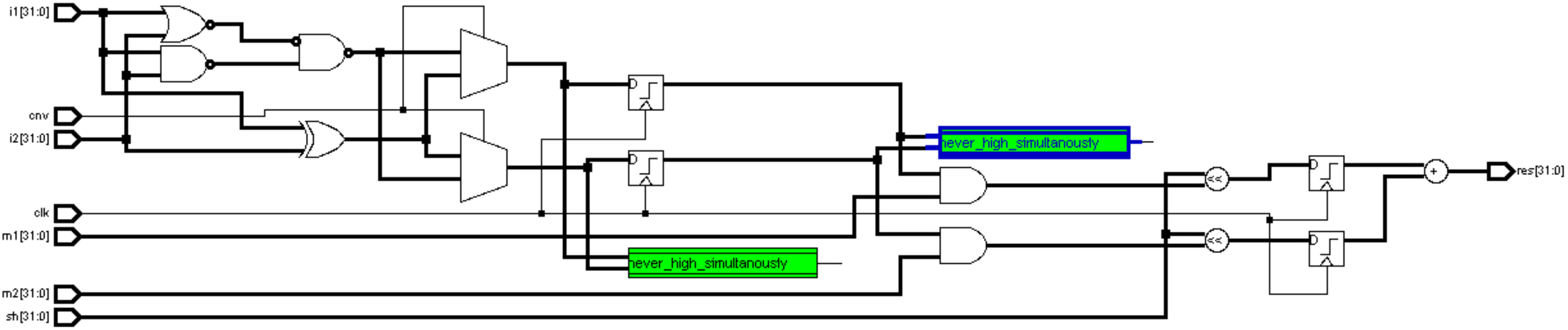
Select Property and Click on "Duplicate Logic"



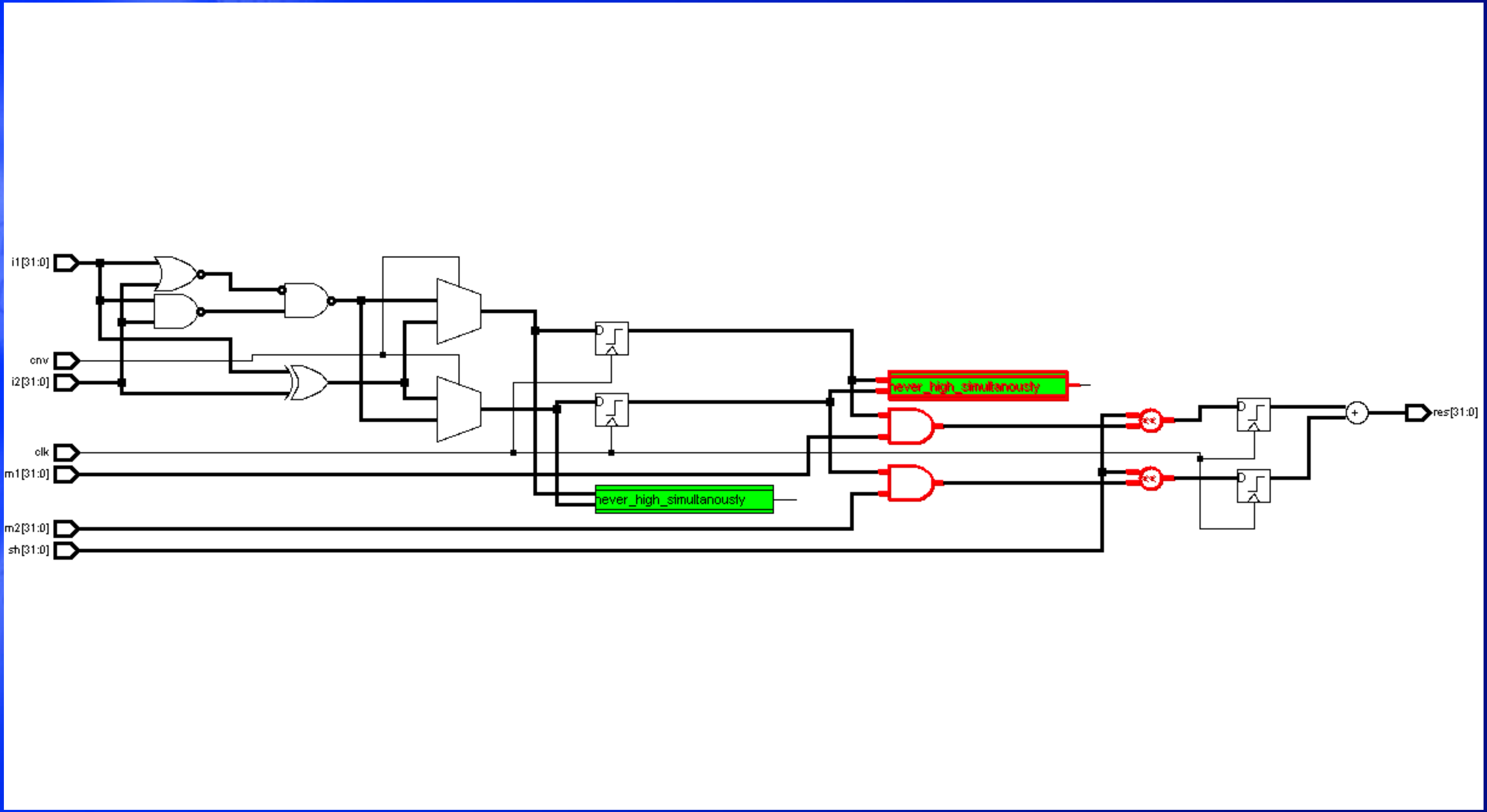
Now Select Property and Click on "Retime Forward"



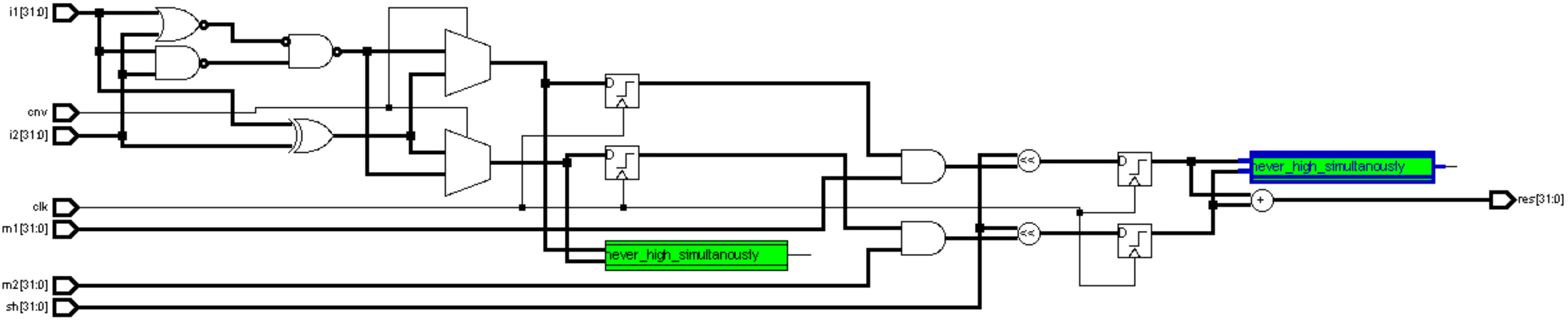
Result


i1[31:0]
cnv
i2[31:0]
clk
m1[31:0]
m2[31:0]
sh[31:0]
never_high_simultanously
never_high_simultanously
<<
<<
+
res[31:0]

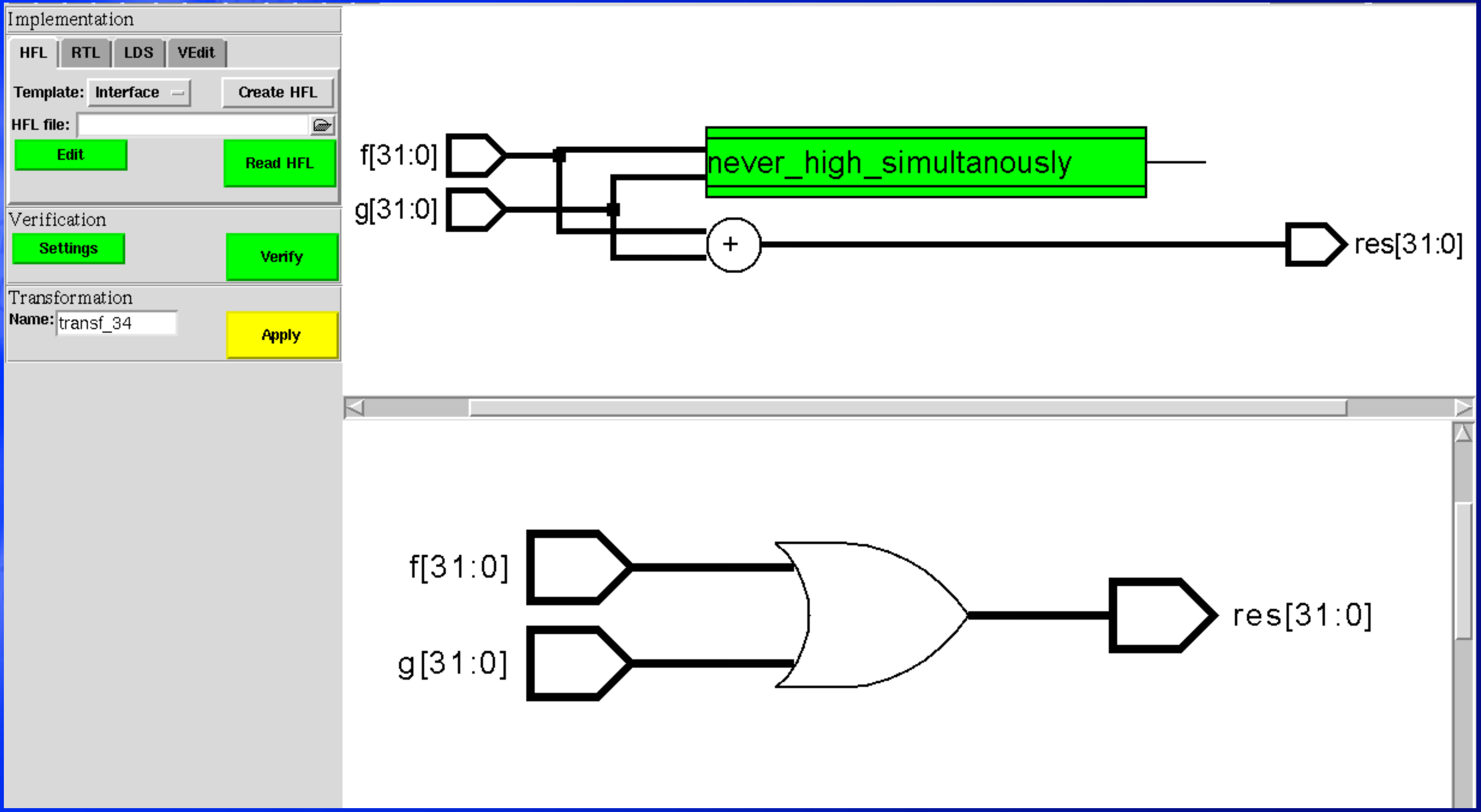
Now Select New Block and Repeat Complete Process



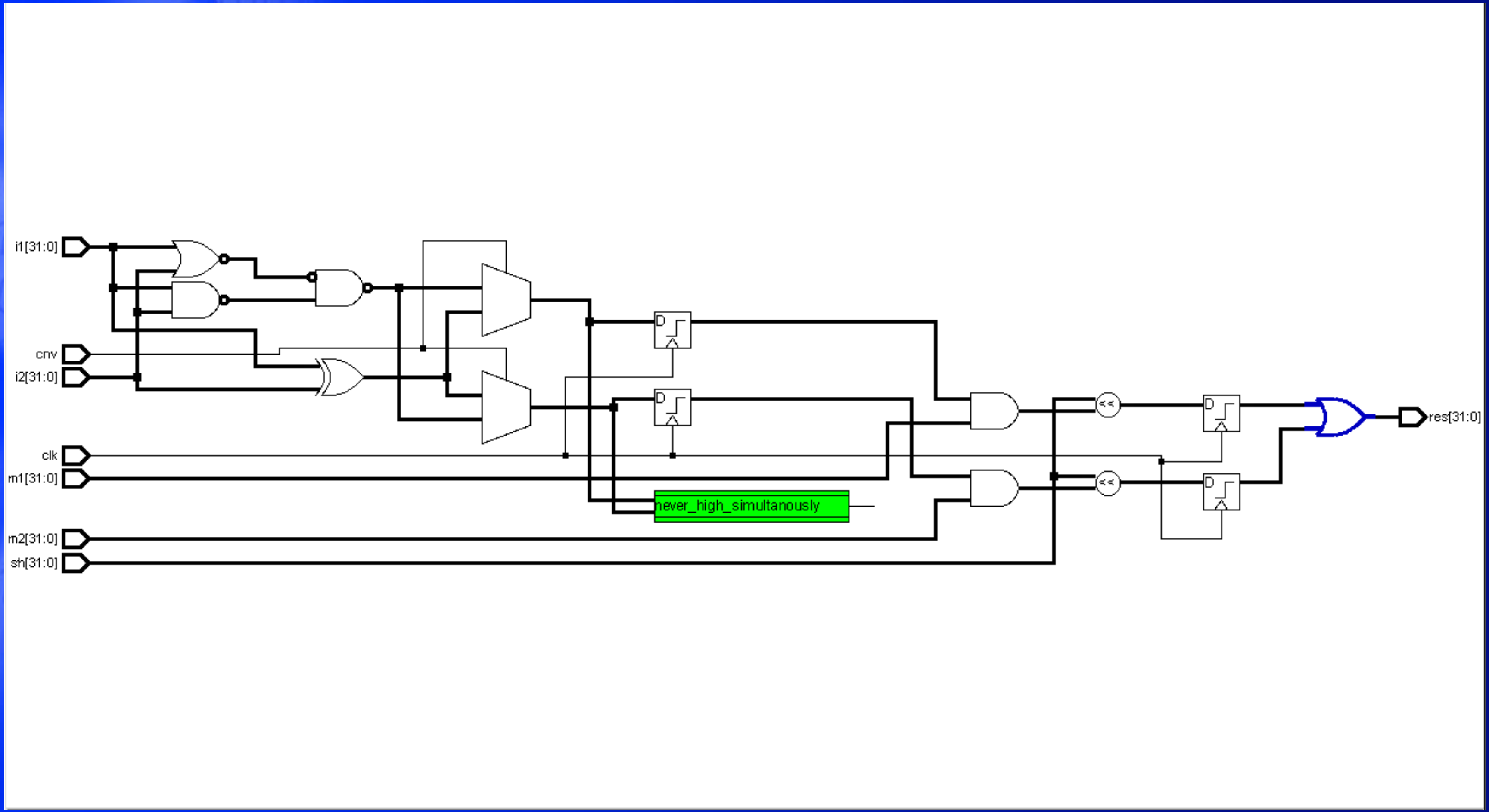
Result


i1[31:0]
cnv
i2[31:0]
clk
m1[31:0]
m2[31:0]
sh[31:0]
never_high_simultanously
<<
<<
never_high_simultanously
+
res[31:0]

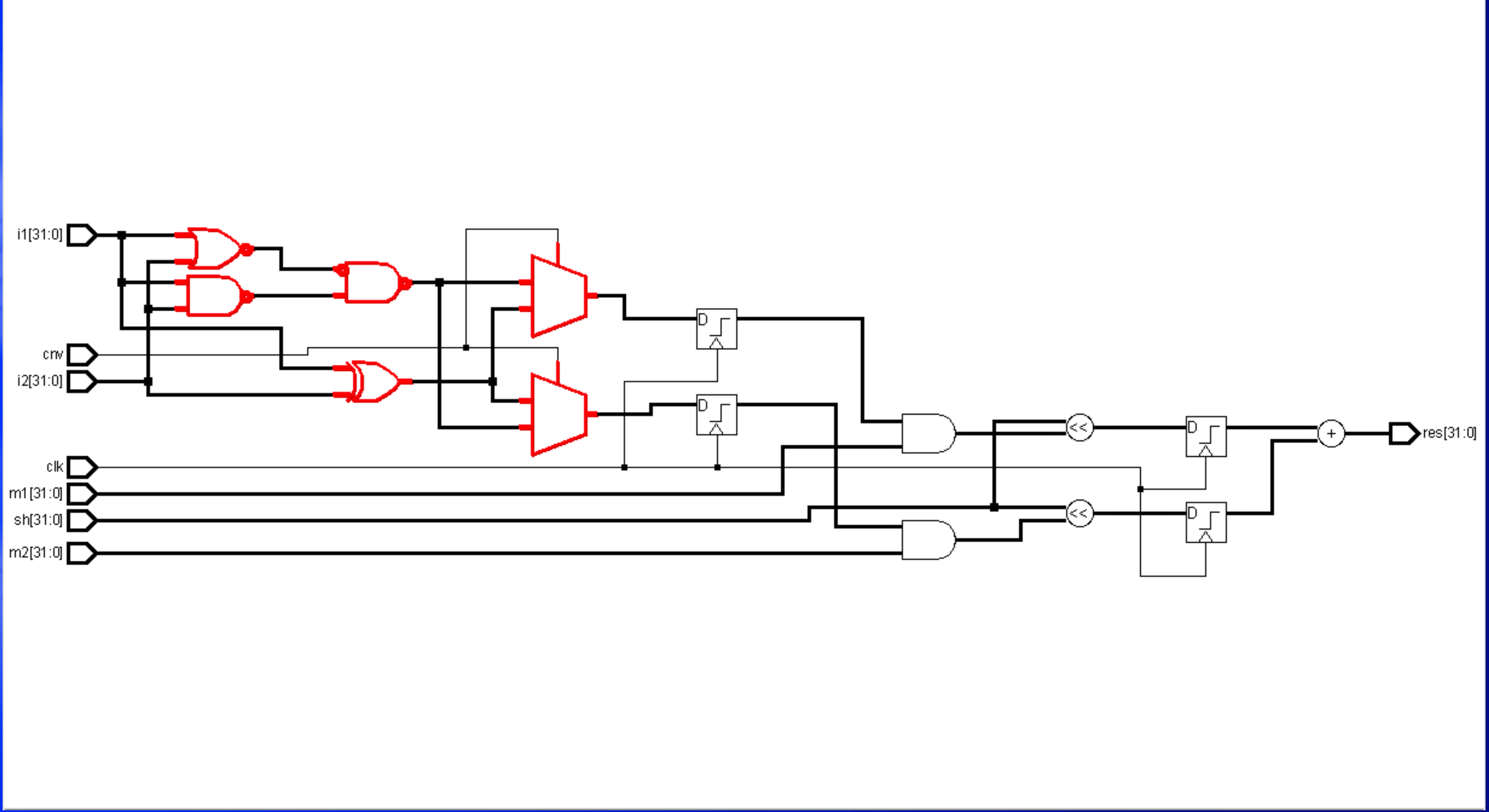
Finally use Property to Drastically Simplify Design



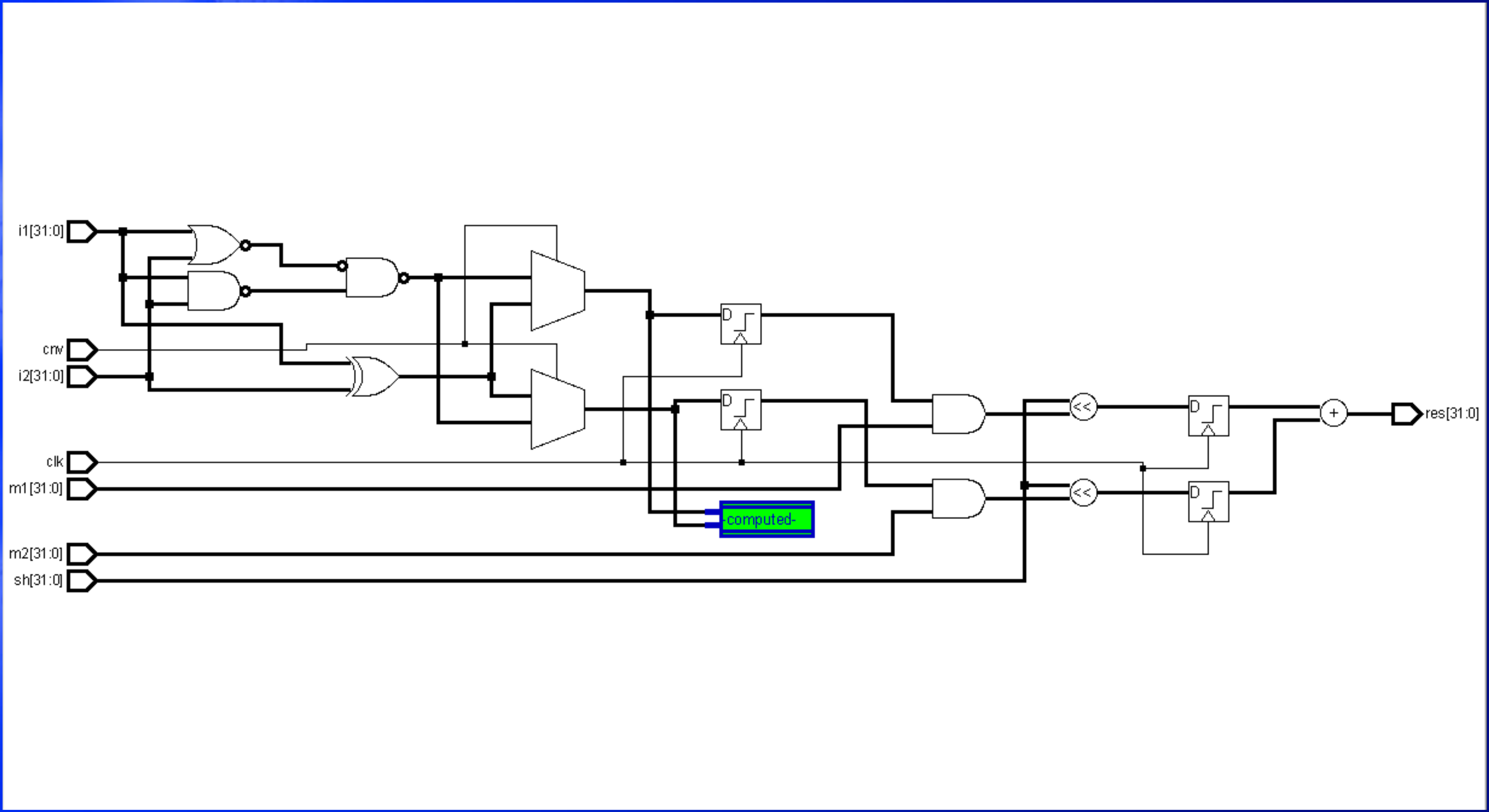
Final Result



Automation Can Also be Used:



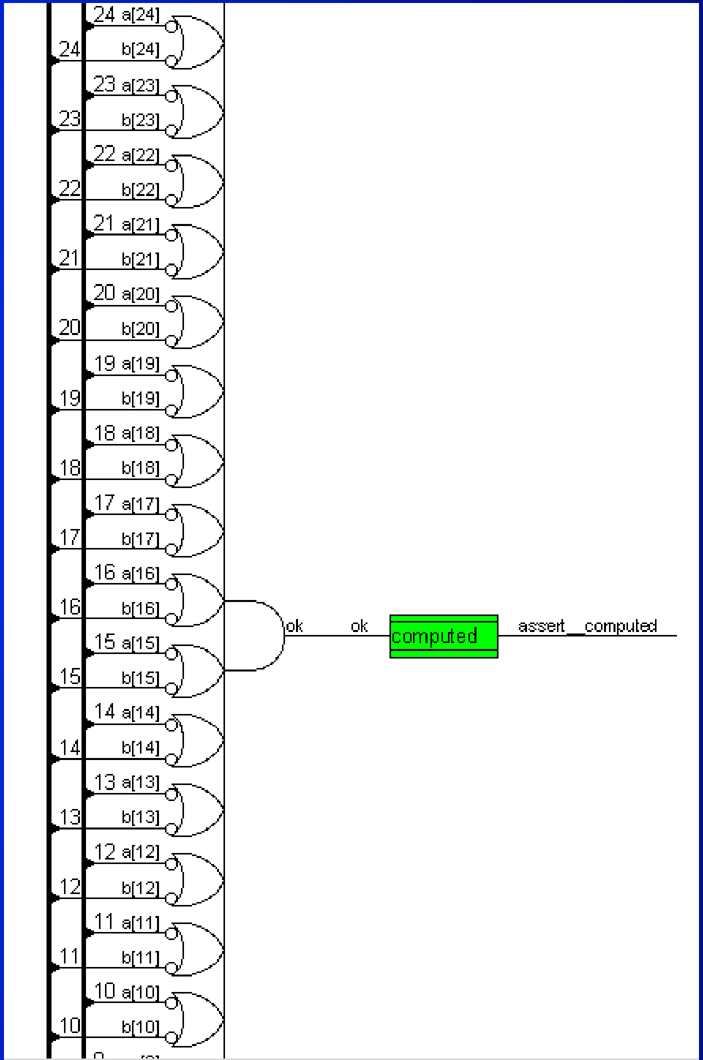
Automatically Computed Property



Where:

computed

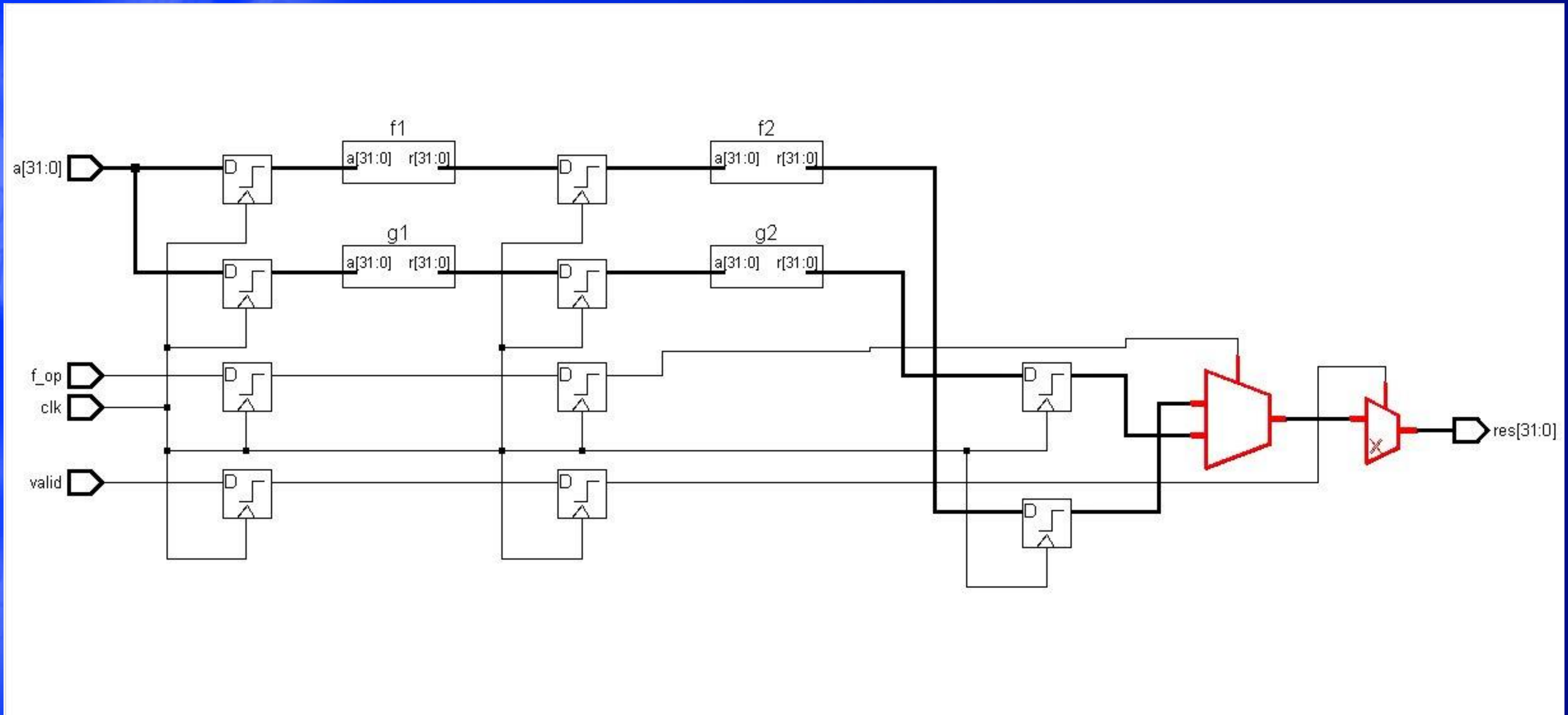
=


24 a[24]
b[24]
24
23 a[23]
b[23]
23
22 a[22]
b[22]
22
21 a[21]
b[21]
21
20 a[20]
b[20]
20
19 a[19]
b[19]
19
18 a[18]
b[18]
18
17 a[17]
b[17]
17
16 a[16]
b[16]
16
15 a[15]
b[15]
15
14 a[14]
b[14]
14
13 a[13]
b[13]
13
12 a[12]
b[12]
12
11 a[11]
b[11]
11
10 a[10]
b[10]
10
ok
ok
computed
assert__computed

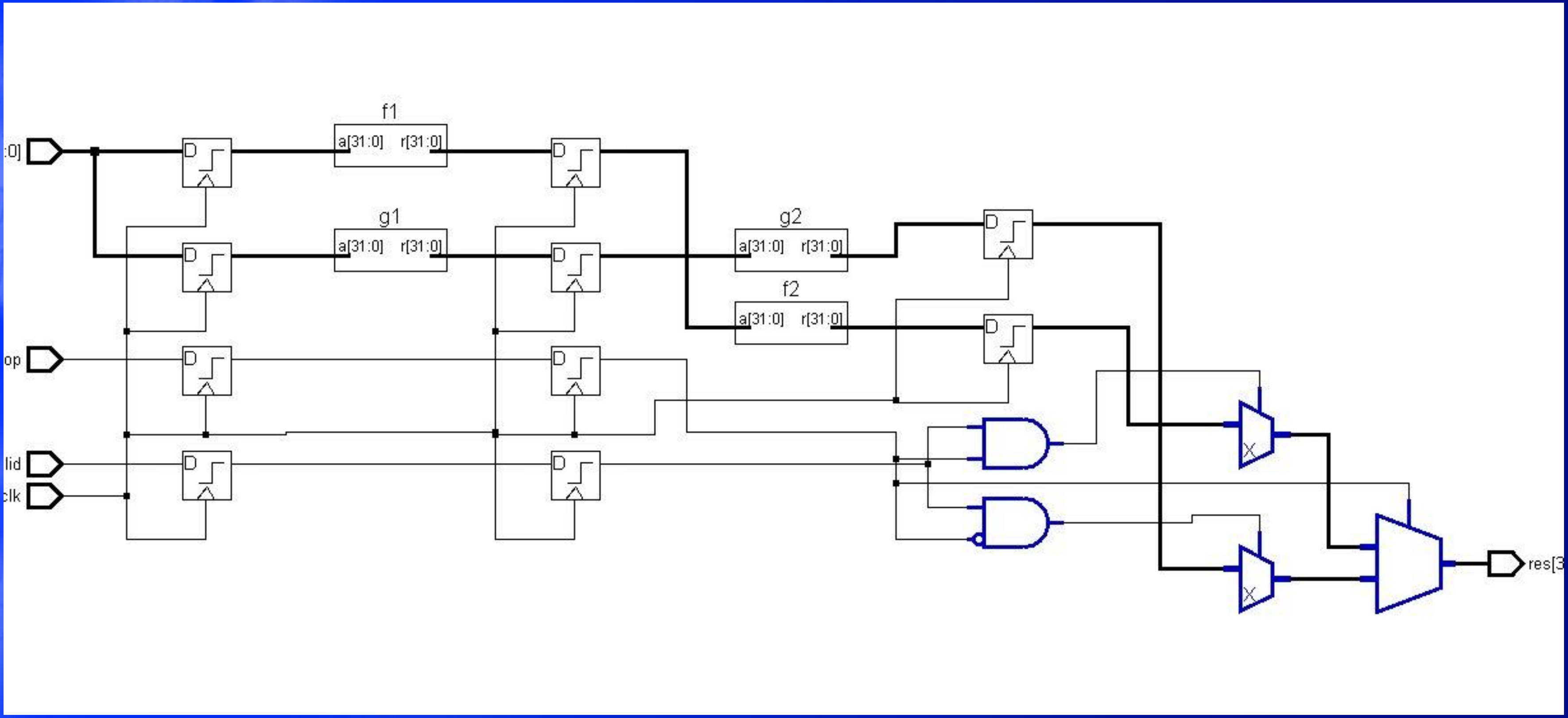
Care Properties

- Since IDV uses (qua)ternary logic in refinement verification, output cares are modeled using “tri-state drivers”
  - E.g., output is “X” when care condition is false.
  - As with properties, the basic care component is combinational. Extra circuit is used to create sequential care properties.
- Care properties have two major sources:
  - Initially in the HLM
    - Requires diligence to actually state them!
  - Implied by down-stream logic
    - Written by hand & verified or computed automatically using formal methods.
- Care properties can be added/moved/… like hardware components and the verify tool understands and checks correctness.

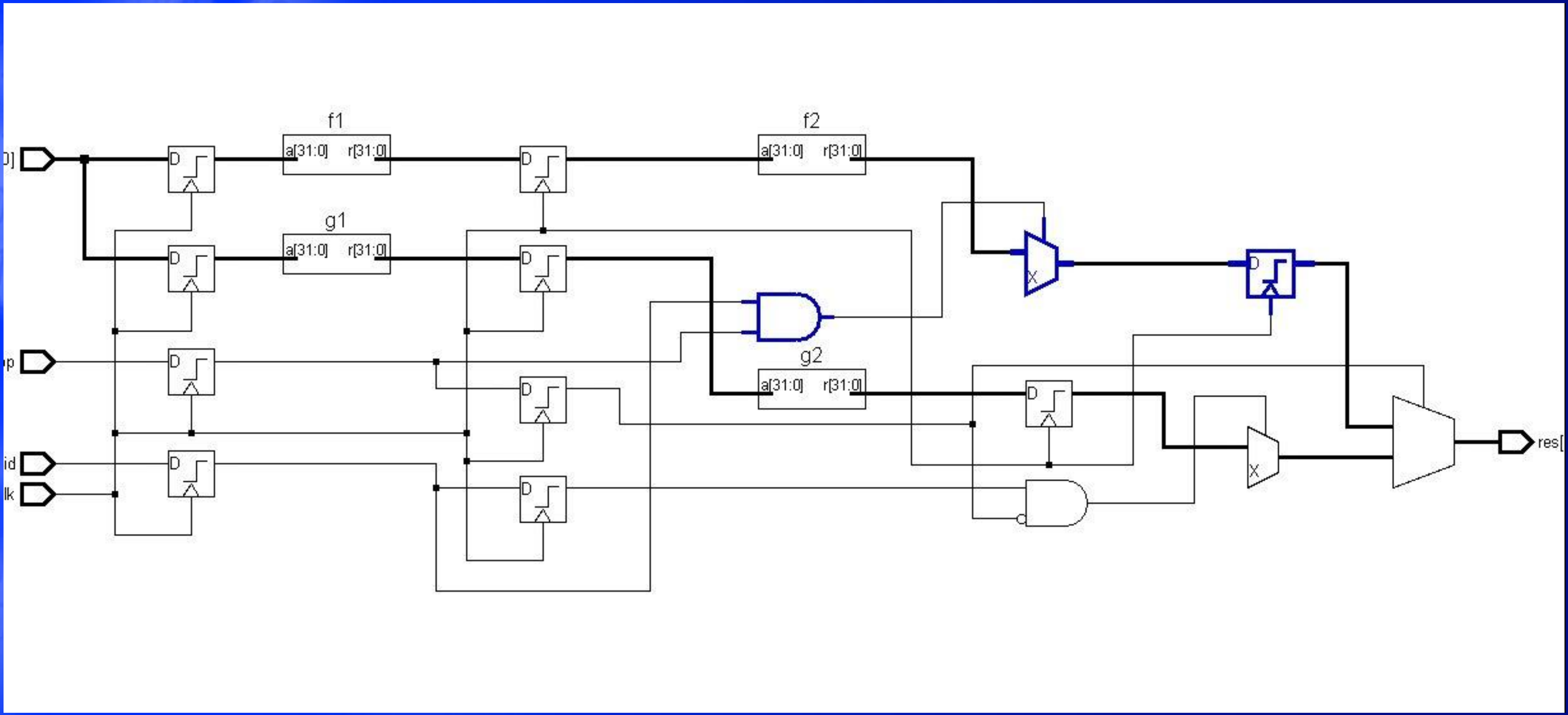
Circuit with Explicit Care



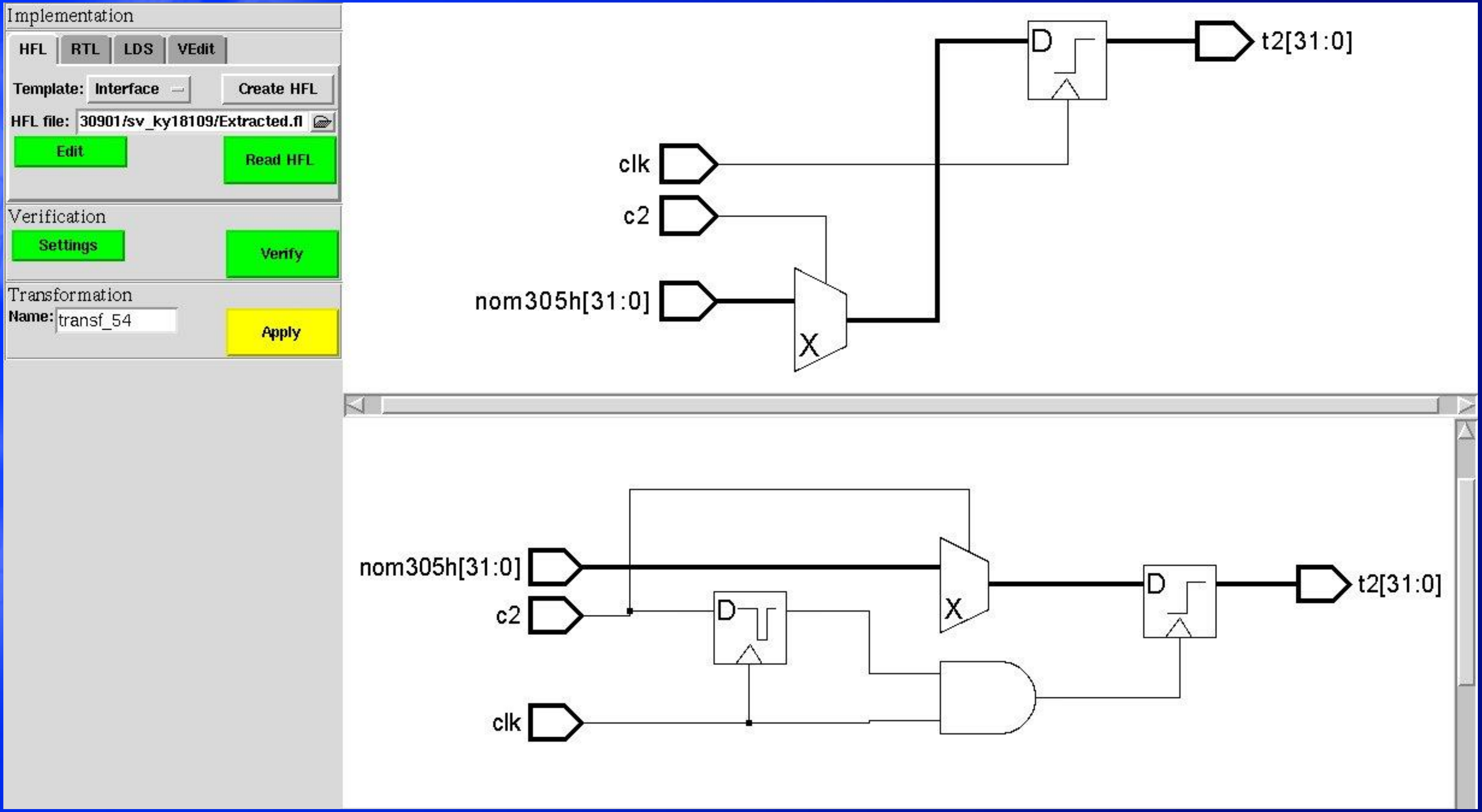
Combine Explicit Care with Implied Care (verified!)



Retime Care Backwards



Use Care to Introduce Clock Gating (Sequential FEV)

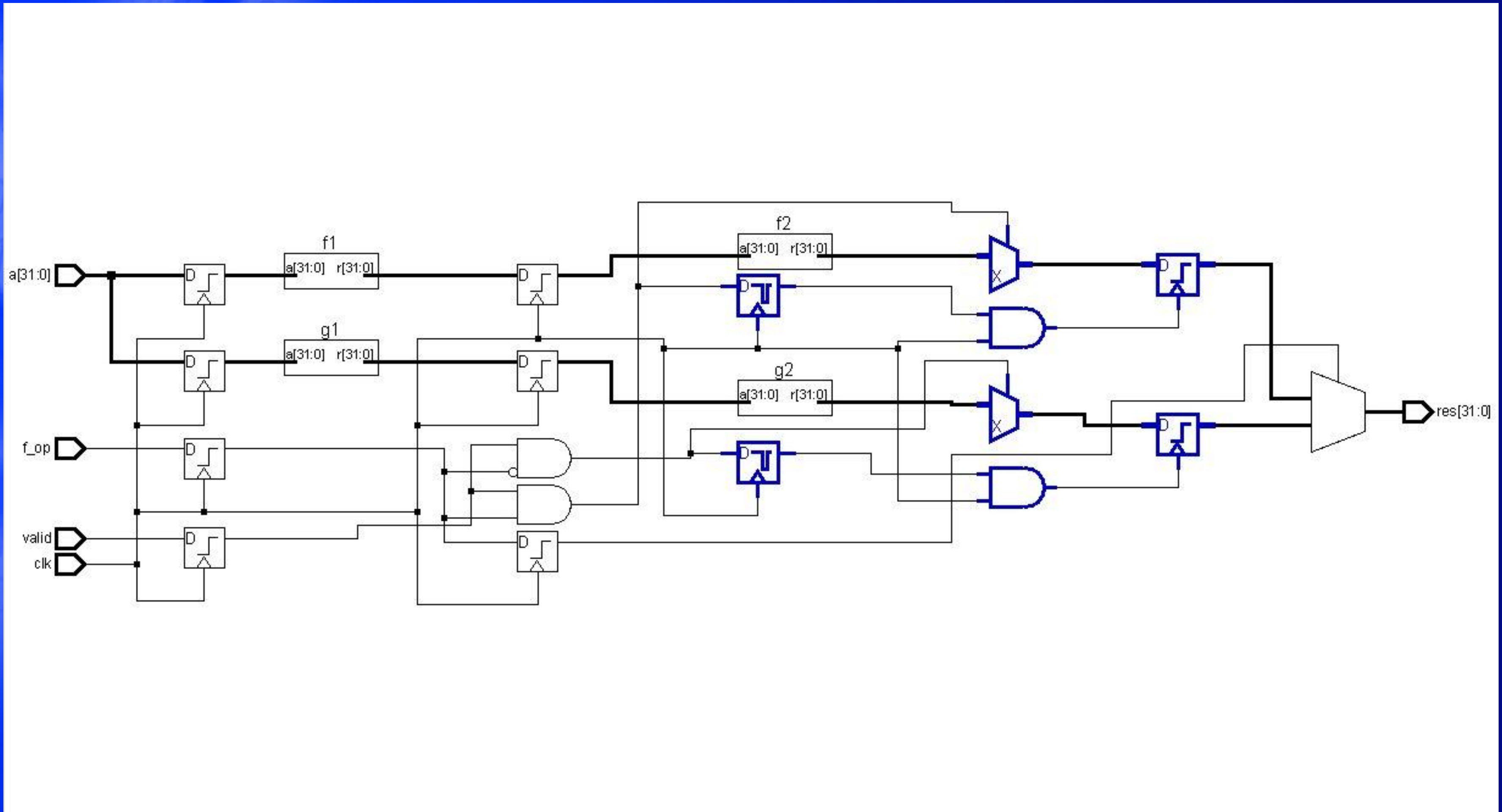


Template: Interface

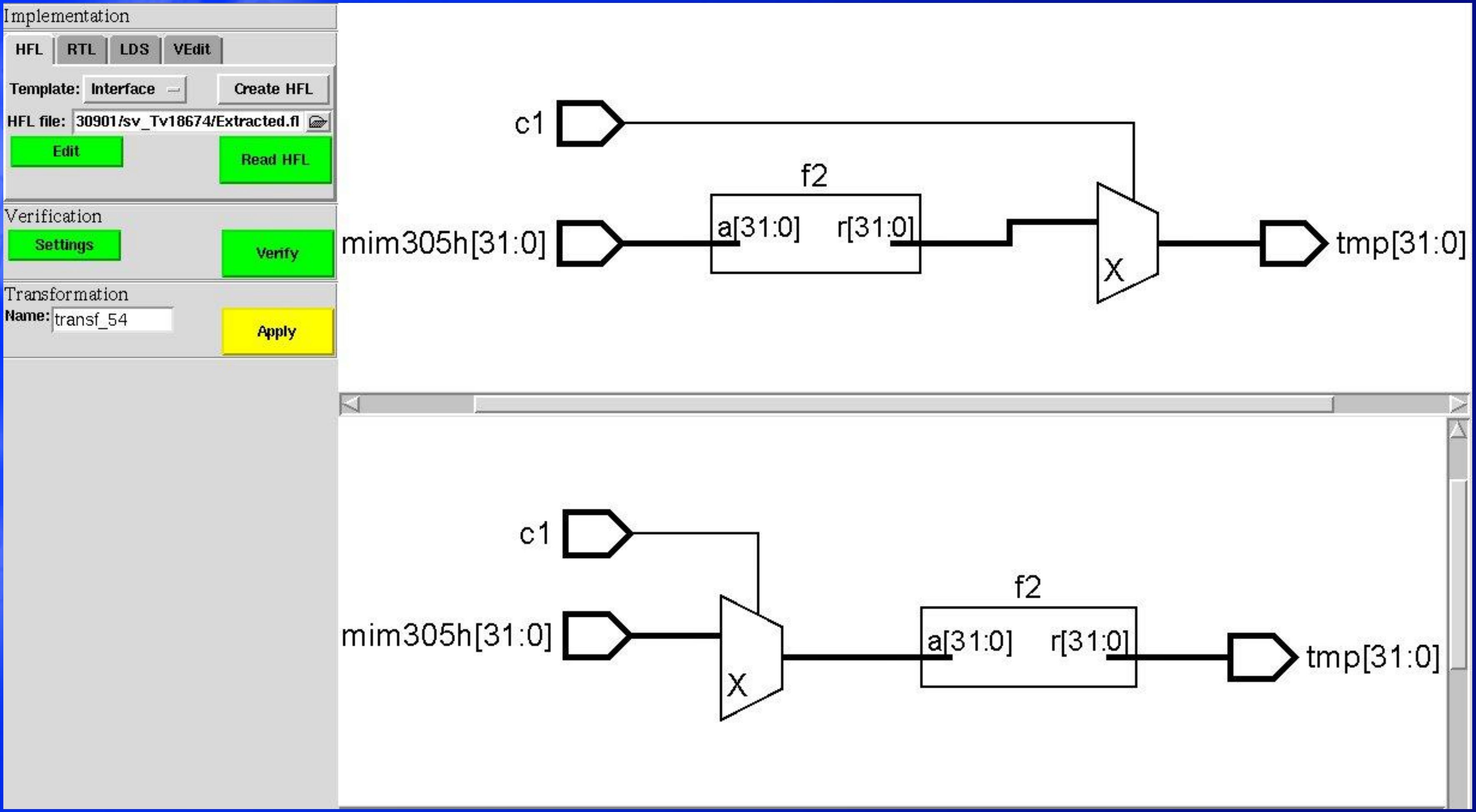
HFL file: 30901/sv_ky18109/Extracted.fl

Name: transf_54

Final Stage Clock Gated



Move Care Backwards Through Combinational Logic



Implementation

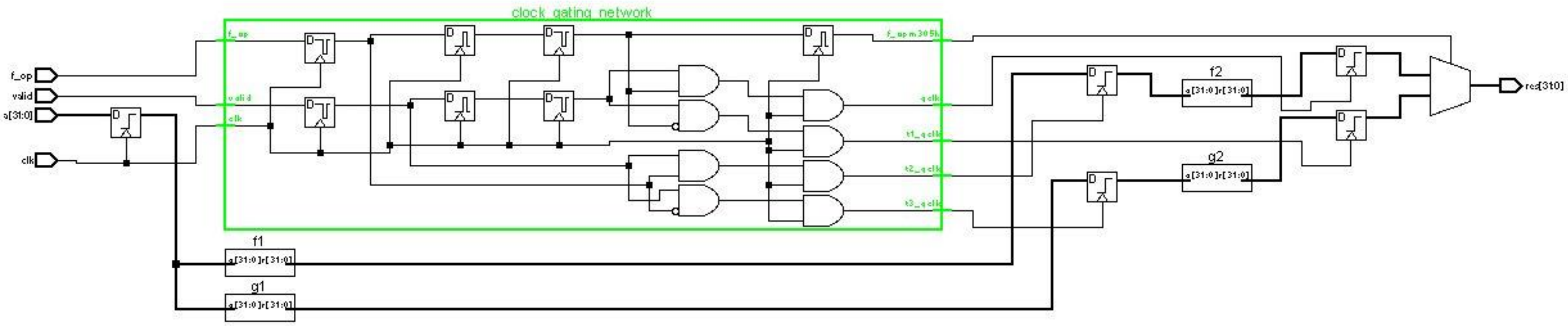
HFL file: 30901/sv_Tv18674/Extracted.fl

Template: Interface

Transformation

Name: transf_54

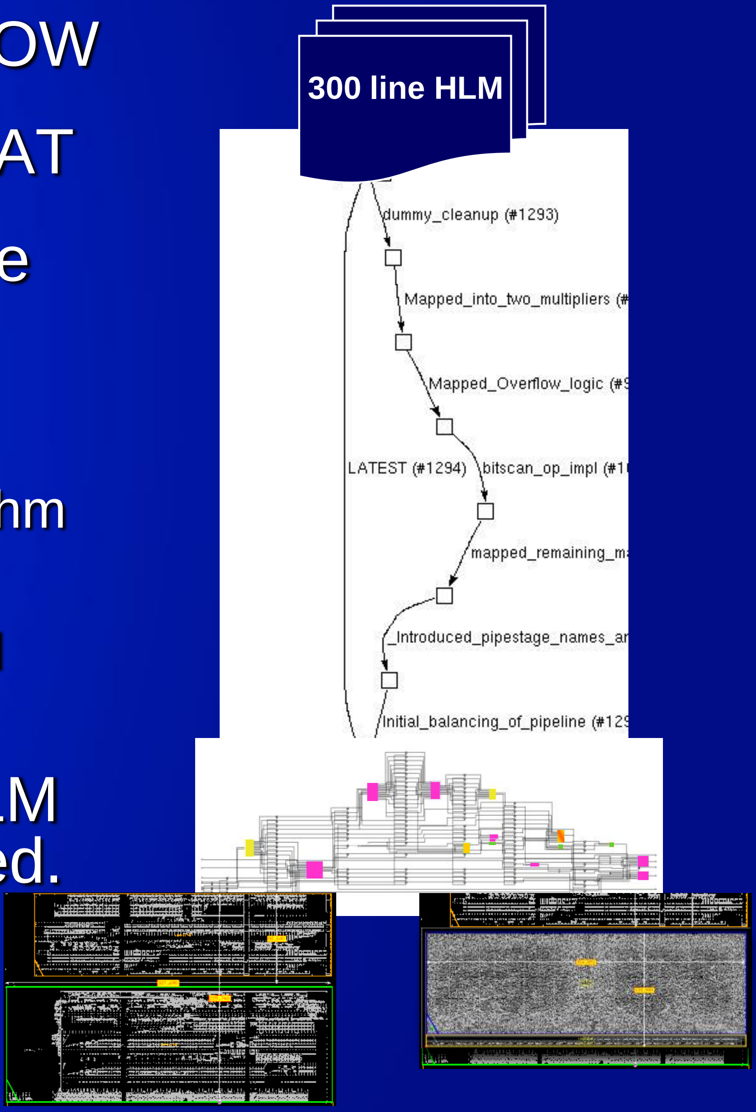
Final Result



Realistic Examples

Integer Execution Unit in Core

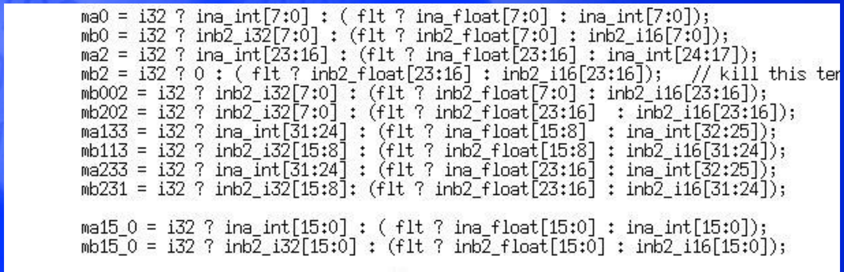
- RTL: ~3,000 lines with focus on HOW
- HLM: ~300 lines with focus on WHAT
- Two implementations derived inside IDV
  1. To the existing implementation
  2. New version using a different algorithm and partitioning
  - New version 20% smaller than original version
- Both versions provably equal to HLM and thus HLM validation was shared.



Graphics Execution Unit

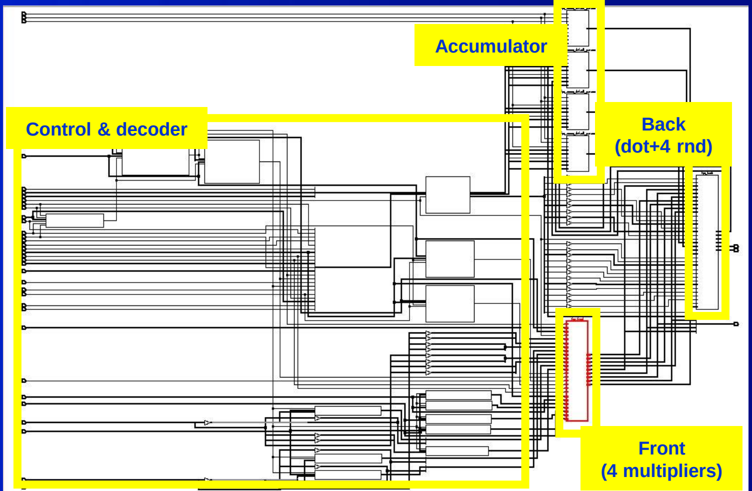
Graphics execution unit
HLM -> Placed cells
2k lines of code + 20 pages tables

HLM

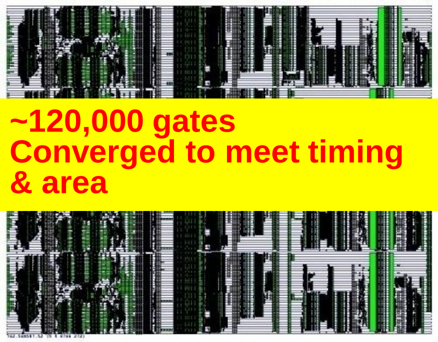


```
ma0 = i32 ? ina_int[7:0] : ( flt ? ina_float[7:0] : ina_int[7:0]);
mb0 = i32 ? inb2_i32[7:0] : (flt ? inb2_float[7:0] : inb2_i16[7:0]);
ma2 = i32 ? ina_int[23:16] : (flt ? ina_float[23:16] : ina_int[24:17]);
mb2 = i32 ? 0 : ( flt ? inb2_float[23:16] : inb2_i16[23:16]);    // kill this ter
mb002 = i32 ? inb2_i32[7:0] : (flt ? inb2_float[7:0] : inb2_i16[23:16]);
mb202 = i32 ? inb2_i32[7:0] : (flt ? inb2_float[23:16]  : inb2_i16[23:16]);
ma133 = i32 ? ina_int[31:24] : (flt ? ina_float[15:8]  : ina_int[32:25]);
mb113 = i32 ? inb2_i32[15:8] : (flt ? inb2_float[15:8] : inb2_i16[31:24]);
ma233 = i32 ? ina_int[31:24] : (flt ? ina_float[23:16] : ina_int[32:25]);
mb231 = i32 ? inb2_i32[15:8]: (flt ? inb2_float[23:16] : inb2_i16[31:24]);

ma15_0 = i32 ? ina_int[15:0] : ( flt ? ina_float[15:0] : ina_int[15:0]);
mb15_0 = i32 ? inb2_i32[15:0] : (flt ? inb2_float[15:0] : inb2_i16[15:0]);
```

High-level specification

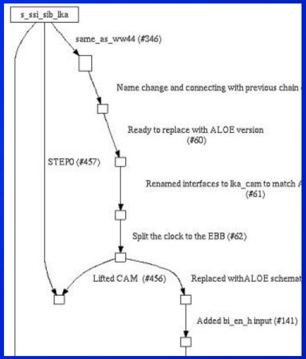


Final placed result

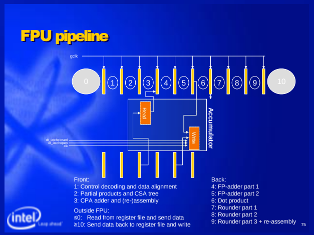


~120,000 gates
Converged to meet timing
& area

Design and verification in IDV


s_ssi_sib_lka
same_as_ww44 (#346)
Name change and connecting with previous chain
Ready to replace with ALOE version
(#60)
STEP0 (#457)
Renamed interfaces to lka_cam to match
(#61)
Split the clock to the EBB (#62)
Lifted CAM (#456)
Replaced withALOE schemat
Added bi_en_h input (#141)

New implementation algorithm ideas



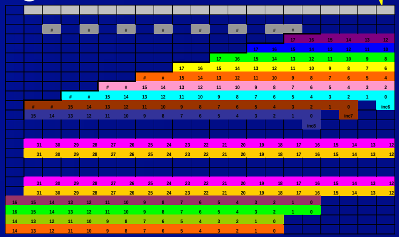
FPU pipeline

Front:
1: Control decoding and data alignment
2: Partial products and CSA tree
3: CPA adder and (re-)assembly

Outside FPU:
≤0: Read from register file and send data
≥10: Send data back to register file and write

Back:
4: FP-adder part 1
5: FP-adder part 2
6: Dot product
7: Rounder part 1
8: Rounder part 2
9: Rounder part 3 + re-assembly

75

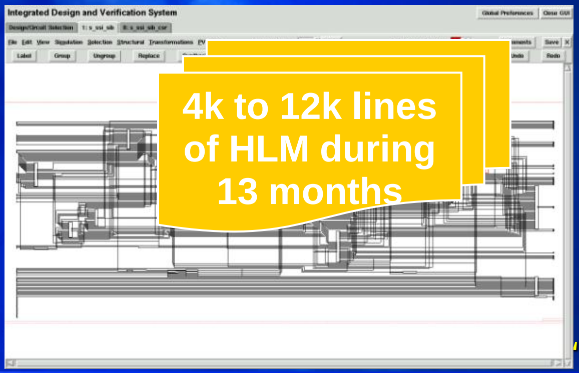


Communication Link Between Interconnect and Cache in “Uncore”

Original input buffer
- 1 designer
- 12 FUBs
- 2 RF, 1 CAM EBB

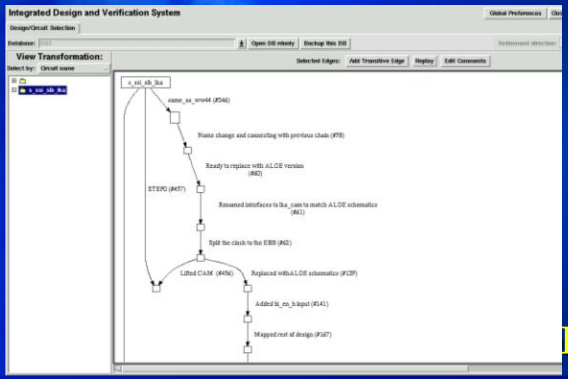
In production flow for more than 1 year

Top-level HLM Entry



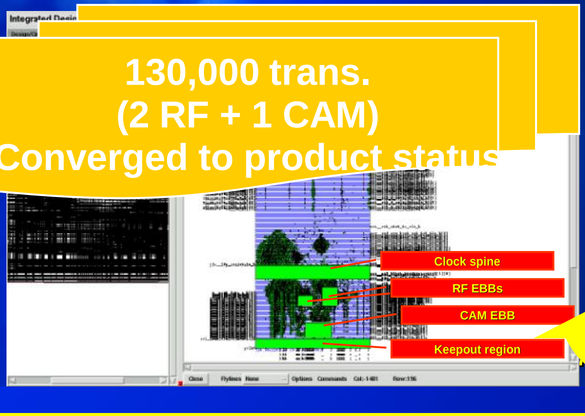
4k to 12k lines of HLM during 13 months

Early Design: HLM to netlist



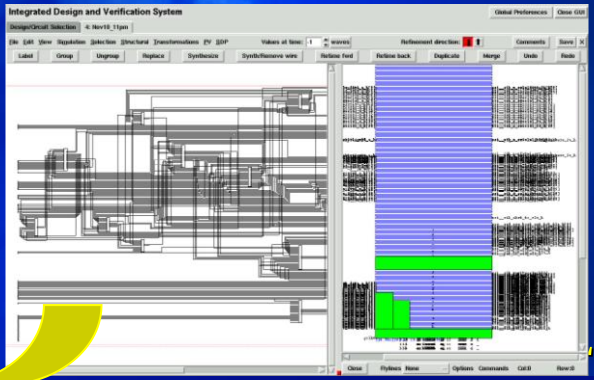
Integrated Design and Verification System
Global Preferences
Design/Circuit Selection
Database:
Open DB rdonly
Backup this DB
View Transformation:
Select by: Circuit name
Selected Edges:
Add Transitive Edge
Replay
Edit Comments
s_ssi_sb_lka
same_as_wu44 (#346)
Name change and connecting with previous chain (#58)
Ready to replace with ALOE version (#60)
STEPO (#457)
Renamed interfaces to lka_cam to match ALOE schematics (#61)
Split the clock to the EBB (#62)
Lifted CAM (#456)
Replaced with ALOE schematics (#129)
Added bi_en_h input (#141)
Mapped rest of design (#167)

Final Design Sent to Router



130,000 trans.
(2 RF + 1 CAM)
Converged to product status

Logic And Physical View



Bottom line: During 13 months of design effort, no HLM changes were needed because of implementation considerations.

Conclusions and Future Work

Experience in Handling Properties Like Hardware Components.

- Pros:
  - Automatically manage properties (e.g., wire renaming gets done the same for flops as properties!)
  - Make properties highly visible and explicit
  - Formalizes many "hand waving" arguments (and finds quite a few bugs!)
  - Ensures property verification gets the same priority as design verification!
- Cons:
  - Sometimes very tedious to manage
    - E.g., forgetting to duplicate a property used in a replacement!
  - "Global" properties are difficult to use/move around
  - Difficult to deal with for backend tools
    - Properties will eventually "disappear" since they will not result in any transistors on the chip!

Pros with an IDV Methodology

- Direct benefits:
  - Bugs about to be introduced during the design process will be caught immediately
    - "Goofs" (e.g., cut-and-paste errors)
    - Design complexity bugs, e.g., performance artifacts (speculation, re-timing, power-down), testability, etc.
  - No need to re-write the model to be “synthesis friendly” and (unintentionally) introduce bugs.
- Indirect benefits:
  - HLM much smaller and simpler than today's RTL
    - Can be written and maintained by a few people
    - Allows significantly faster simulation (DV)
    - Is a much better target for formal property verification
  - HLM much more stable
    - Can make emulation much more attractive
  - Same HLM can be refined to different implementations with different tradeoffs
    - Ideal in a System-On-Chip design environment

Cons with an IDV Methodology

- New role that require significant training and/or changed mindset:
  - Designers don’t know validation
  - Validators don’t know how to design
- Reacting to changes in the HLM can be tedious and require significant re-work
- Difficult to make use of “global” properties and don’t cares.
- Truly high-level models require significant FV expertise to refine & verify to abstract RTL
- Danger of “video-game” design:
  - Making large number of refinements & transformations without really converging towards a viable design.
    - My record is ~210 transformations to get back to where I started!

Open Questions

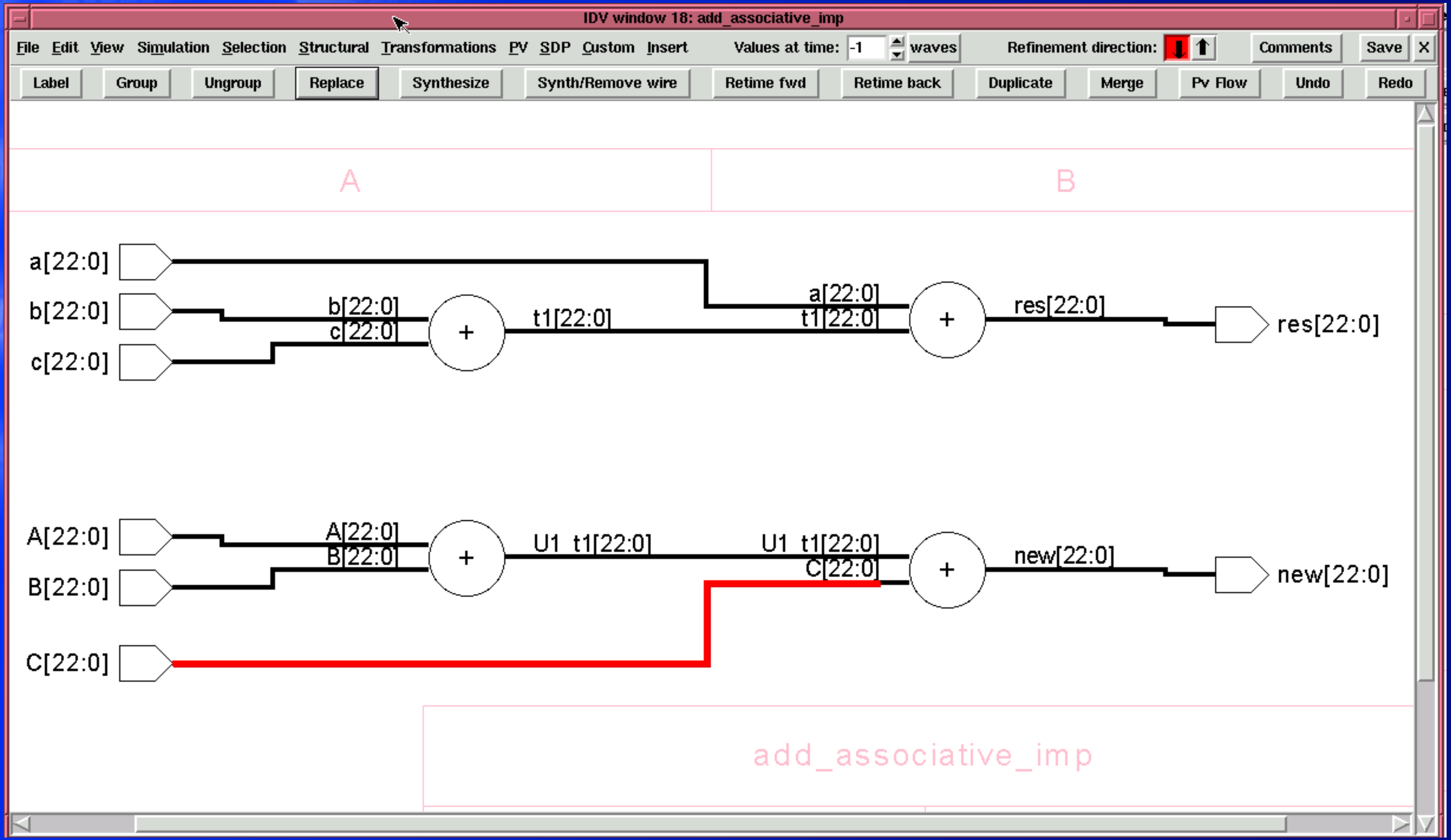
- What is the right level of a High-Level Model?
  - It’s not really a question of language (although a good/bad language can help/hinder abstraction)
  - How can a truly abstract model be used for other purposes than logic specification?
    - High-level models are needed for many non-logic purposes!
- What is the right refinement relation?
  - Tradeoff between flexibility and difficulty verifying.
- What is the best way of capturing “design intent” so that the process is captured, not only the end result.
- …

Thank You!

Questions?

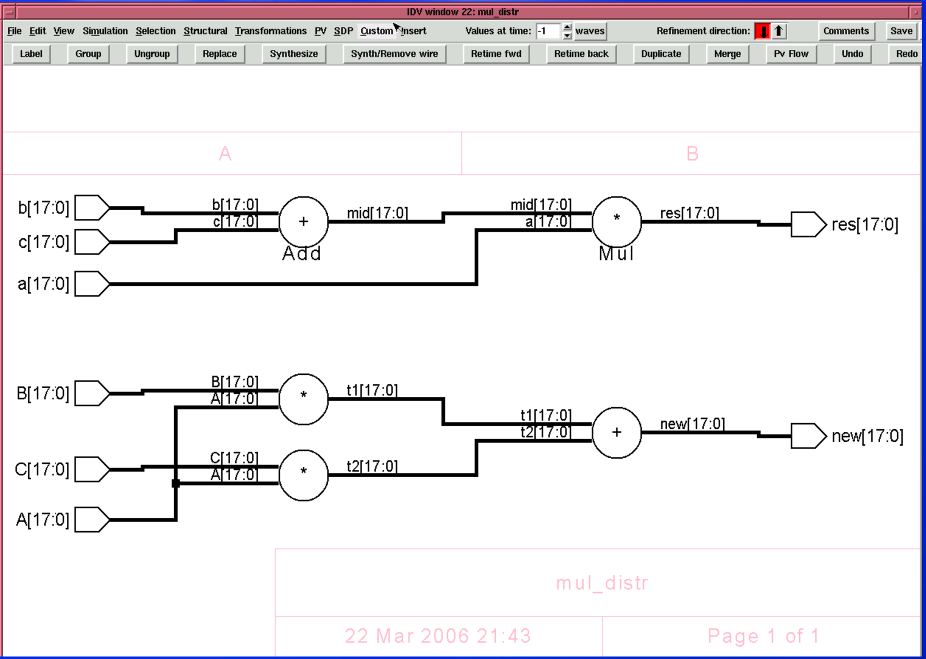
Example of High-Level Transformations

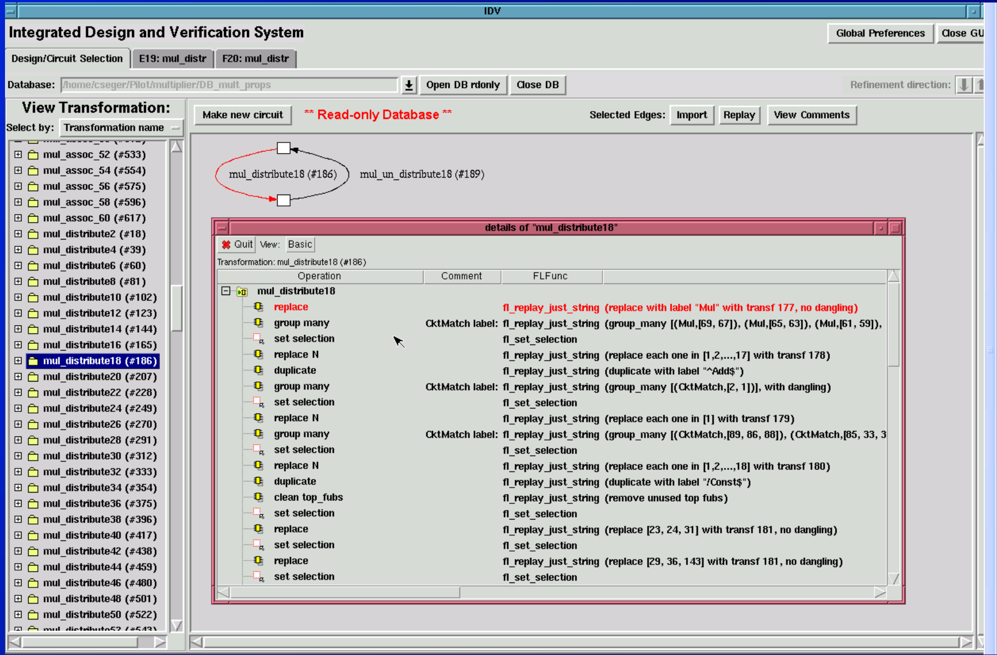
- Basic arithmetic facts: E.g., a+(b+c)=(a+b)+c.
- Verified through FEV for every size and stored in database



High Level Transformations

- Complex transformations: E.g., a*(b+c)=a*b+a*c
- Verified through a sequence of IDV transformations
- Sequence captured in reFLect program and result for every size stored in database





Database: /home/cseger/Pilot/multiplier/DB_mult_props

Select by: Transformation name

- mul_assoc_52 (#533)
- mul_assoc_54 (#554)
- mul_assoc_56 (#575)
- mul_assoc_58 (#596)
- mul_assoc_60 (#617)
- mul_distribute2 (#18)
- mul_distribute4 (#39)
- mul_distribute6 (#60)
- mul_distribute8 (#81)
- mul_distribute10 (#102)
- mul_distribute12 (#123)
- mul_distribute14 (#144)
- mul_distribute16 (#165)
- mul_distribute18 (#186)
- mul_distribute20 (#207)
- mul_distribute22 (#228)
- mul_distribute24 (#249)
- mul_distribute26 (#270)
- mul_distribute28 (#291)
- mul_distribute30 (#312)
- mul_distribute32 (#333)
- mul_distribute34 (#354)
- mul_distribute36 (#375)
- mul_distribute38 (#396)
- mul_distribute40 (#417)
- mul_distribute42 (#438)
- mul_distribute44 (#459)
- mul_distribute46 (#480)
- mul_distribute48 (#501)
- mul_distribute50 (#522)

Transformation: mul_distribute18 (#186)

Operation	Comment	FLFunc
mul_distribute18		
replace		fl_replay_just_string (replace with label "Mul" with transf 177, no dangling)
group many	CktMatch label:	fl_replay_just_string (group_many [(Mul,[69, 67]), (Mul,[65, 63]), (Mul,[61, 59]),
set selection		fl_set_selection
replace N		fl_replay_just_string (replace each one in [1,2,...,17] with transf 178)
duplicate		fl_replay_just_string (duplicate with label "^Add$")
group many	CktMatch label:	fl_replay_just_string (group_many [(CktMatch,[2, 1])], with dangling)
set selection		fl_set_selection
replace N		fl_replay_just_string (replace each one in [1] with transf 179)
group many	CktMatch label:	fl_replay_just_string (group_many [(CktMatch,[89, 86, 88]), (CktMatch,[85, 33, 3
set selection		fl_set_selection
replace N		fl_replay_just_string (replace each one in [1,2,...,18] with transf 180)
duplicate		fl_replay_just_string (duplicate with label "/Const$")
clean top_fubs		fl_replay_just_string (remove unused top fubs)
set selection		fl_set_selection
replace		fl_replay_just_string (replace [23, 24, 31] with transf 181, no dangling)
set selection		fl_set_selection
replace		fl_replay_just_string (replace [29, 36, 143] with transf 181, no dangling)
set selection		fl_set_selection