2021 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES
VIRTUAL | MARCH 1-4, 2021

Can Formal Outsmart Synthesis: Improving Synthesis Quality of Results through Formal Methods

Eldon Nelson M.S. P.E.

Synopsys, Inc.

accellera SYSTEMS INITIATIVE™

Outline

- What is Synthesis?
- What is Formal Coverage Analysis?
- Use Case
- Overview of our Example Design tinyalu
- Method
- Results
- Summary

RTL and Gates

RTL and Gates

RTL
(SystemVerilog)

Gates
(Gate Level Netlist)

RTL and Gates

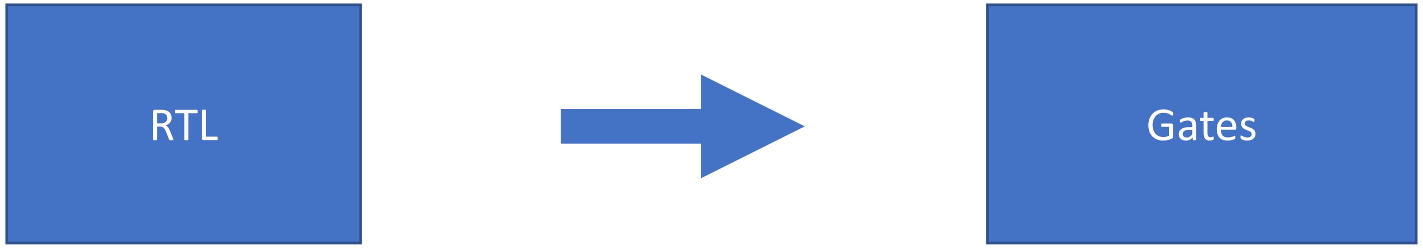
RTL

Gates

```
module single_cycle (
        input        [7:0]  A,
        input        [7:0]  B,
        input        [2:0]  op,
        input               clk,
        input               reset_n,
        input               start,
        output logic        done,
        output logic [15:0] result
);

always @(posedge clk) begin
        if (!reset_n) begin
                result <= '0;
        end else begin
                case(op)
                        OP_ADD : result <= A + B;
                        OP_AND : result <= A & B;
                        OP_XOR : result <= A ^ B;
                endcase // case (op)
        end
end
```

```
module single_cycle ( A, B, op, clk, reset_n, start, done, result );
  input [7:0] A;
  input [7:0] B;
  input [2:0] op;
  output [15:0] result;
  input clk, reset_n, start;
  output done;
  wire   N67, n56, n57, n58, n59, n60, n61, n62, n63, n64, n1, n2, n3, n4, n5,
         n6, n7, n8, n9, n10, n11, n12, n13, n14, n15, n16, n17, n18, n19, n20,
         n21, n22, n23, n24, n25, n26, n27, n28, n29, n30, n31, n32, n33, n34,
         n35, n36, n37, n38, n39, n40, n41, n42, n43, n44, n45, n46, n47, n48,
         n49, n50, n51, n52, n53, n54, n55, n65, n66, n67, n68, n69, n70, n71,
         n72, n73, n74, n75, n76, n77, n78, n79, n80, n81, n82, n83, n84, n85,
         n86, n87, n88, n89, n90, n91, n92, n93, n94, n95, n96, n97, n98, n99,
         n100;

  FD1 result_reg_8_ ( .D(n64), .CP(clk), .Q(result[8]), .QN(n100) );
  FD1 result_reg_7_ ( .D(n63), .CP(clk), .Q(result[7]) );
  FD1 result_reg_6_ ( .D(n62), .CP(clk), .Q(result[6]) );
  FD1 result_reg_5_ ( .D(n61), .CP(clk), .Q(result[5]) );
  FD1 result_reg_4_ ( .D(n60), .CP(clk), .Q(result[4]) );
  FD1 result_reg_3_ ( .D(n59), .CP(clk), .Q(result[3]) );
  FD1 result_reg_2_ ( .D(n58), .CP(clk), .Q(result[2]) );
  FD1 result_reg_1_ ( .D(n57), .CP(clk), .Q(result[1]) );
```

What is Synthesis



```
module single_cycle (
        input        [7:0]  A,
        input        [7:0]  B,
        input        [2:0]  op,
        input               clk,
        input               reset_n,
        input               start,
        output logic        done,
        output logic [15:0] result
);

always @(posedge clk) begin
        if (!reset_n) begin
                result <= '0;
        end else begin
                case(op)
                        OP_ADD : result <= A + B;
                        OP_AND : result <= A & B;
                        OP_XOR : result <= A ^ B;
                endcase // case (op)
        end
end
```

```
module single_cycle ( A, B, op, clk, reset_n, start, done, result );
  input [7:0] A;
  input [7:0] B;
  input [2:0] op;
  output [15:0] result;
  input clk, reset_n, start;
  output done;
  wire   N67, n56, n57, n58, n59, n60, n61, n62, n63, n64, n1, n2, n3, n4, n5,
         n6, n7, n8, n9, n10, n11, n12, n13, n14, n15, n16, n17, n18, n19, n20,
         n21, n22, n23, n24, n25, n26, n27, n28, n29, n30, n31, n32, n33, n34,
         n35, n36, n37, n38, n39, n40, n41, n42, n43, n44, n45, n46, n47, n48,
         n49, n50, n51, n52, n53, n54, n55, n65, n66, n67, n68, n69, n70, n71,
         n72, n73, n74, n75, n76, n77, n78, n79, n80, n81, n82, n83, n84, n85,
         n86, n87, n88, n89, n90, n91, n92, n93, n94, n95, n96, n97, n98, n99,
         n100;

  FD1 result_reg_8_ ( .D(n64), .CP(clk), .Q(result[8]), .QN(n100) );
  FD1 result_reg_7_ ( .D(n63), .CP(clk), .Q(result[7]) );
  FD1 result_reg_6_ ( .D(n62), .CP(clk), .Q(result[6]) );
  FD1 result_reg_5_ ( .D(n61), .CP(clk), .Q(result[5]) );
  FD1 result_reg_4_ ( .D(n60), .CP(clk), .Q(result[4]) );
  FD1 result_reg_3_ ( .D(n59), .CP(clk), .Q(result[3]) );
  FD1 result_reg_2_ ( .D(n58), .CP(clk), .Q(result[2]) );
  FD1 result_reg_1_ ( .D(n57), .CP(clk), .Q(result[1]) );
```

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2021 VIRTUAL | MARCH 1-4, 2021

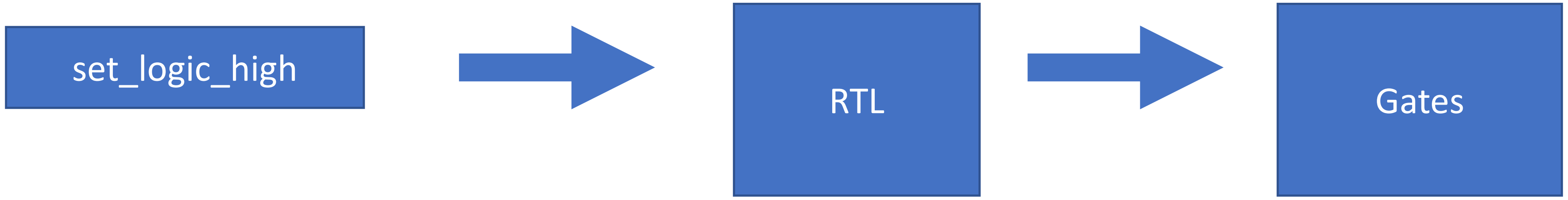
What is Synthesis

```
module single_cycle (
        input           [7:0]  A,
        input           [7:0]  B,
        input           [2:0]  op,
        input                  clk,
        input                  reset_n,
        input                  start,
        output logic           done,
        output logic [15:0] result
);

always @(posedge clk) begin
        if (!reset_n) begin
                result <= '0;
        end else begin
                case(op)
                        OP_ADD : result <= A + B;
                        OP_AND : result <= A & B;
                        OP_XOR : result <= A ^ B;
                endcase // case (op)
        end
end
```

```
module single_cycle ( A, B, op, clk, reset_n, start, done, result );
  input [7:0] A;
  input [7:0] B;
  input [2:0] op;
  output [15:0] result;
  input clk, reset_n, start;
  output done;
  wire   N67, n56, n57, n58, n59, n60, n61, n62, n63, n64, n1, n2, n3, n4, n5,
         n6, n7, n8, n9, n10, n11, n12, n13, n14, n15, n16, n17, n18, n19, n20,
         n21, n22, n23, n24, n25, n26, n27, n28, n29, n30, n31, n32, n33, n34,
         n35, n36, n37, n38, n39, n40, n41, n42, n43, n44, n45, n46, n47, n48,
         n49, n50, n51, n52, n53, n54, n55, n65, n66, n67, n68, n69, n70, n71,
         n72, n73, n74, n75, n76, n77, n78, n79, n80, n81, n82, n83, n84, n85,
         n86, n87, n88, n89, n90, n91, n92, n93, n94, n95, n96, n97, n98, n99,
         n100;

  FD1 result_reg_8_ ( .D(n64), .CP(clk), .Q(result[8]), .QN(n100) );
  FD1 result_reg_7_ ( .D(n63), .CP(clk), .Q(result[7]) );
  FD1 result_reg_6_ ( .D(n62), .CP(clk), .Q(result[6]) );
  FD1 result_reg_5_ ( .D(n61), .CP(clk), .Q(result[5]) );
  FD1 result_reg_4_ ( .D(n60), .CP(clk), .Q(result[4]) );
  FD1 result_reg_3_ ( .D(n59), .CP(clk), .Q(result[3]) );
  FD1 result_reg_2_ ( .D(n58), .CP(clk), .Q(result[2]) );
  FD1 result_reg_1_ ( .D(n57), .CP(clk), .Q(result[1]) );
```

SYSTEMS INITIATIVE

Constant Propagation in Synthesis



2021 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES VIRTUAL | MARCH 1-4, 2021

Constant Propagation in Synthesis

set_logic_high

Synthesis engines can do simple constant propagation such as:

set_logic_high and set_logic_zero

complex relationships in combinatorial or sequential logic is not supported

accellera SYSTEMS INITIATIVE

Formal Coverage Analysis

- Doing Formal Coverage analysis on a design to see if each
  - Line / Toggle is reachable in the design with certain stimulus
- Can take into the advantage of the SystemVerilog “assume” statement
- “assume” statement can describe complicated stimulus behavior elegantly
- “assume” is not readable by Synthesis engines

Current RTL Customization Techniques

- `ifdef
- Parameters
- Generates

Problem

- Increasingly hard for designers to create parameterized designs with high efficiency using these techniques
- Optimizations not pursued for simplicity of code readability
- Side effects and inefficiencies of interacting Parameters and `ifdef

Parameter and Define Overload

- Combinations of parameters and `defines must all be considered
- Burden to validate and plan combinations of all parameters and all `define

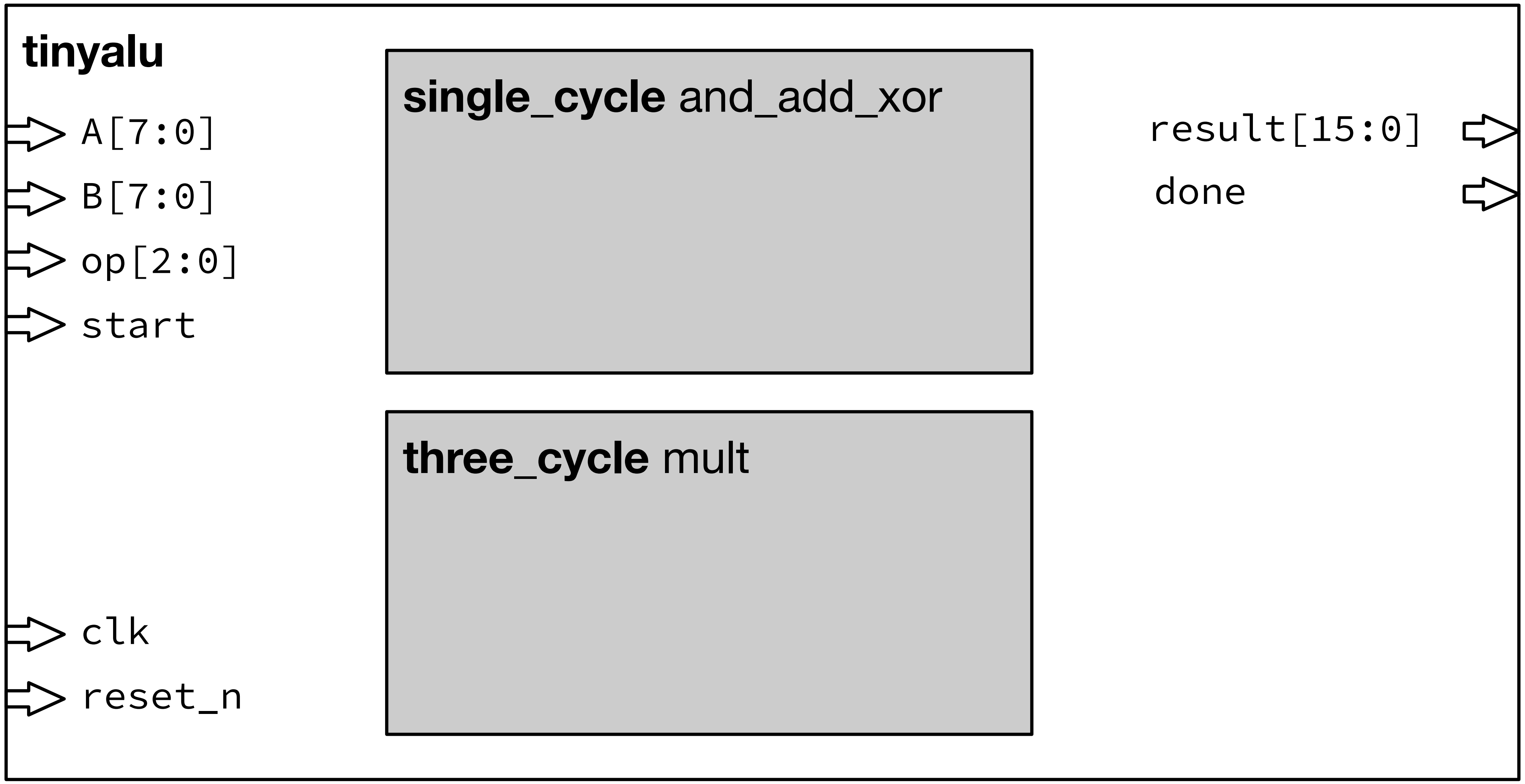
Feature Cost

- How do we gauge how much a design feature costs?
  - what if we only give even numbers to the integer adder in the design?
  - what if we do not allow for two memory operations to certain address back to back?
  - how many gates could we save if if we removed an operand?

Why? Changing Requirements and Deep Optimizations

- Deep optimization of a design for a particular application is a value add reducing power and gate number (area)
- Requirements change that would require a design to add parameters or `ifdef to a completed design
- A request to reduce gate count
  - What features of a design contribute the most to gate count quantitively?

tinyalu (our example DUT)



Operand Table for tinyalu

Operand	Value	bit[2]	bit[1]	bit[0]
OP_NULL	3’h0	0	0	0
OP_MULT	3’h1	0	0	1
OP_AND	3’h2	0	1	0
OP_ADD	3’h3	0	1	1
OP_XOR	3’h4	1	0	0

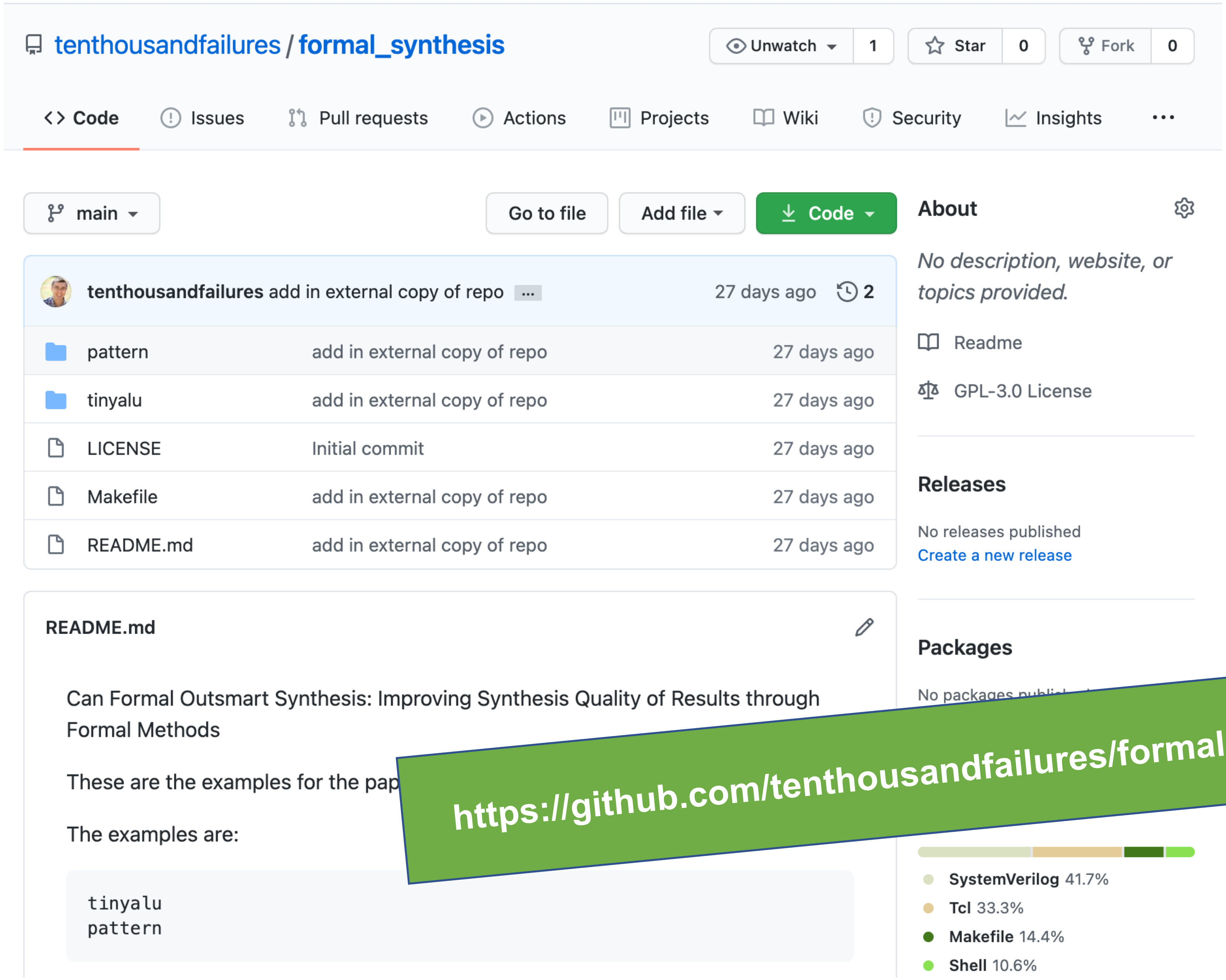
Operand Table for tinyalu

Operand	Value	bit[2]	bit[1]	bit[0]
OP_NULL	3’h0	0	0	0
OP_MULT	3’h1	0	0	1
OP_AND	3’h2	0	1	0
OP_ADD	3’h3	0	1	1
OP_XOR	3’h4	1	0	0

Constant Propagation with set_logic_high or set_logic_zero doesn’t work here

Remove OP_MULT

```
always @(*) begin
     only_single_cycle_ops : assume (op != OP_MULT);
end
```



tenthousandfailures / formal_synthesis

tenthousandfailures add in external copy of repo ... 27 days ago 2

pattern	add in external copy of repo	27 days ago
tinyalu	add in external copy of repo	27 days ago
LICENSE	Initial commit	27 days ago
Makefile	add in external copy of repo	27 days ago
README.md	add in external copy of repo	27 days ago

README.md

Can Formal Outsmart Synthesis: Improving Synthesis Quality of Results through Formal Methods

These are the examples for the pap

The examples are:

```
tinyalu
pattern
```

About

No description, website, or topics provided.

Readme

GPL-3.0 License

Releases

No releases published

Create a new release

Packages

No packages publi

● SystemVerilog 41.7%

● Tcl 33.3%

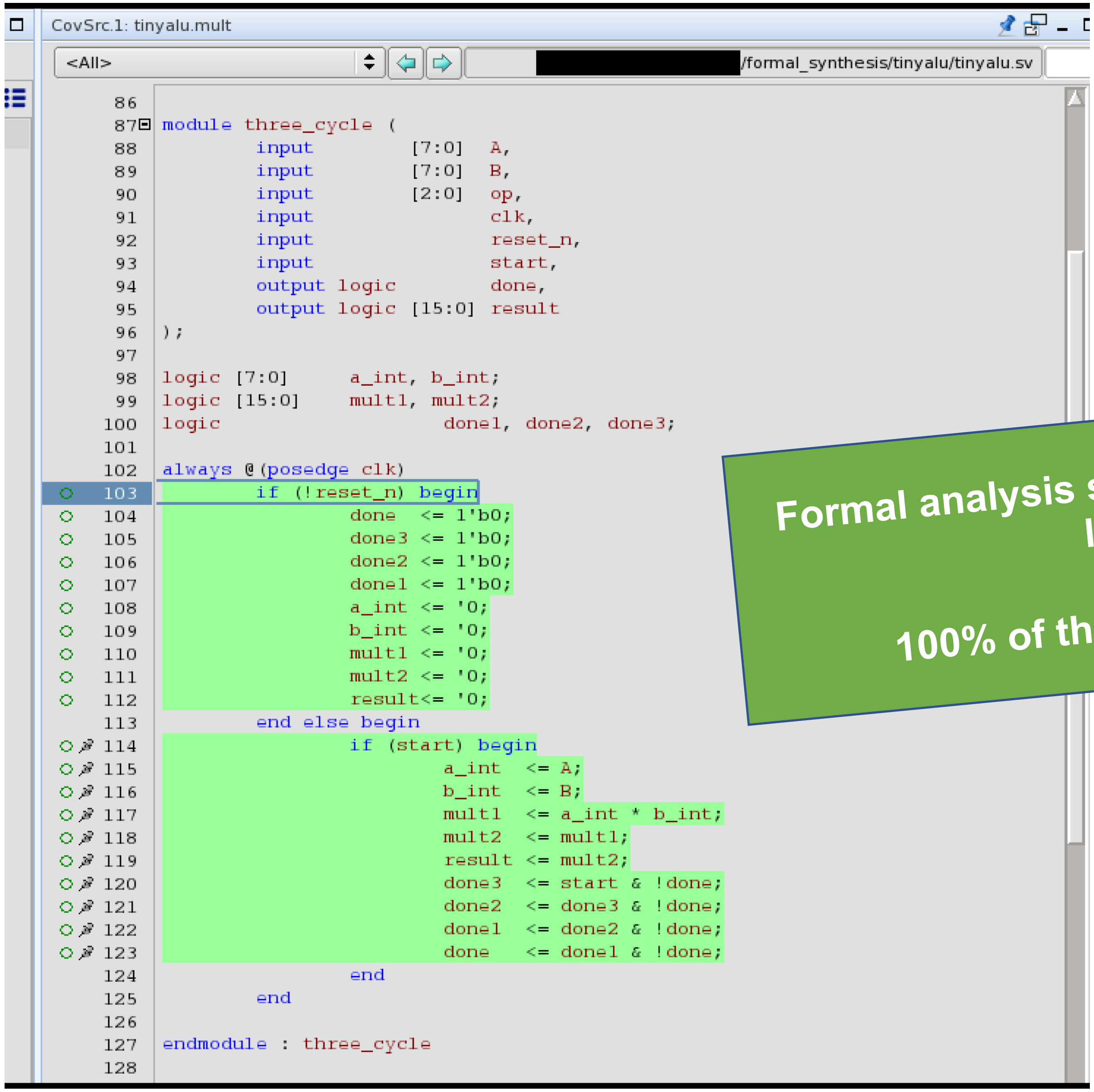
● Makefile 14.4%

● Shell 10.6%

https://github.com/tenthousandfailures/formal_synthesis

Start Formal

```
> make tinyalu
```

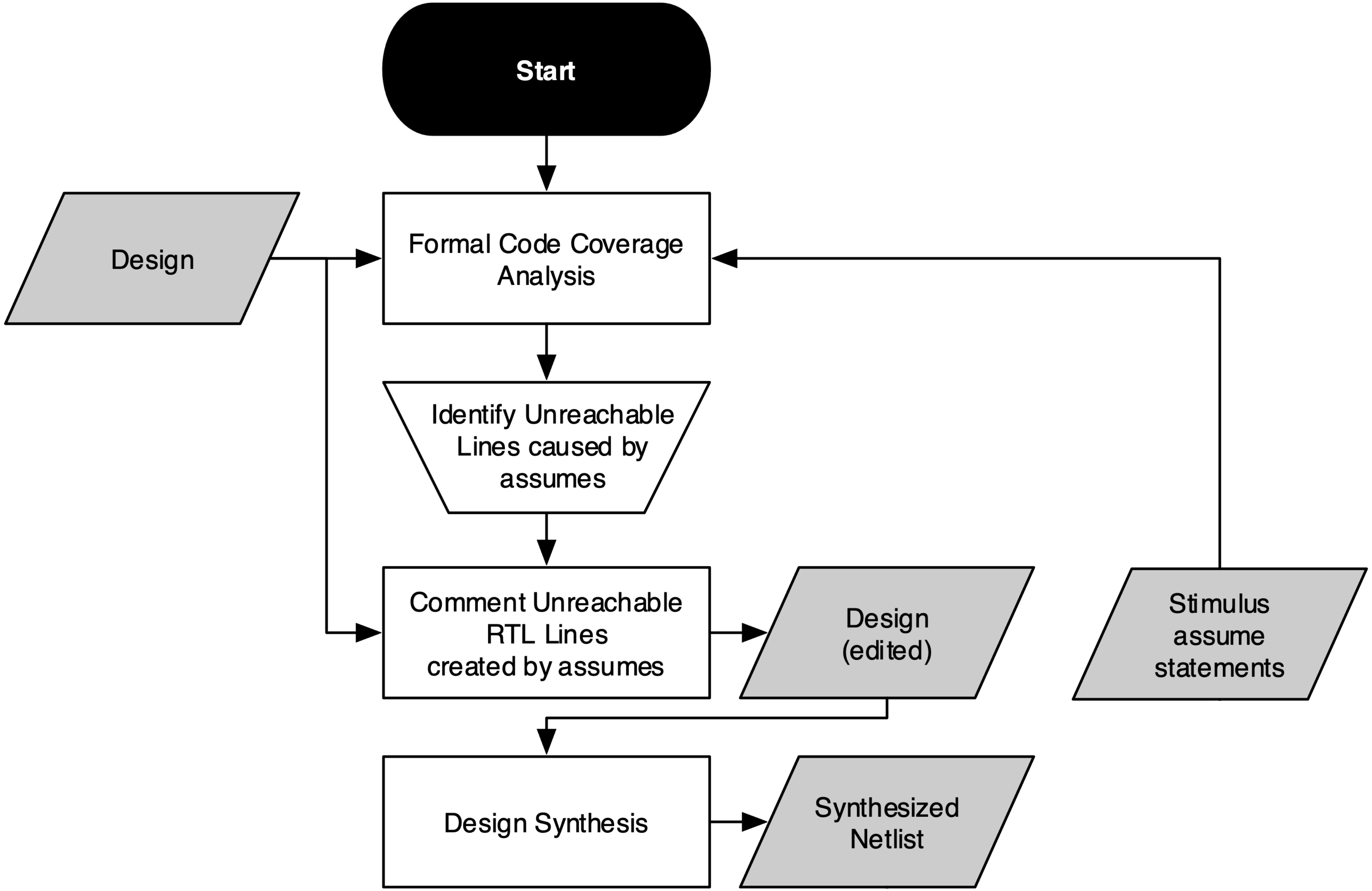

CovSrc.1: tinyalu.mult

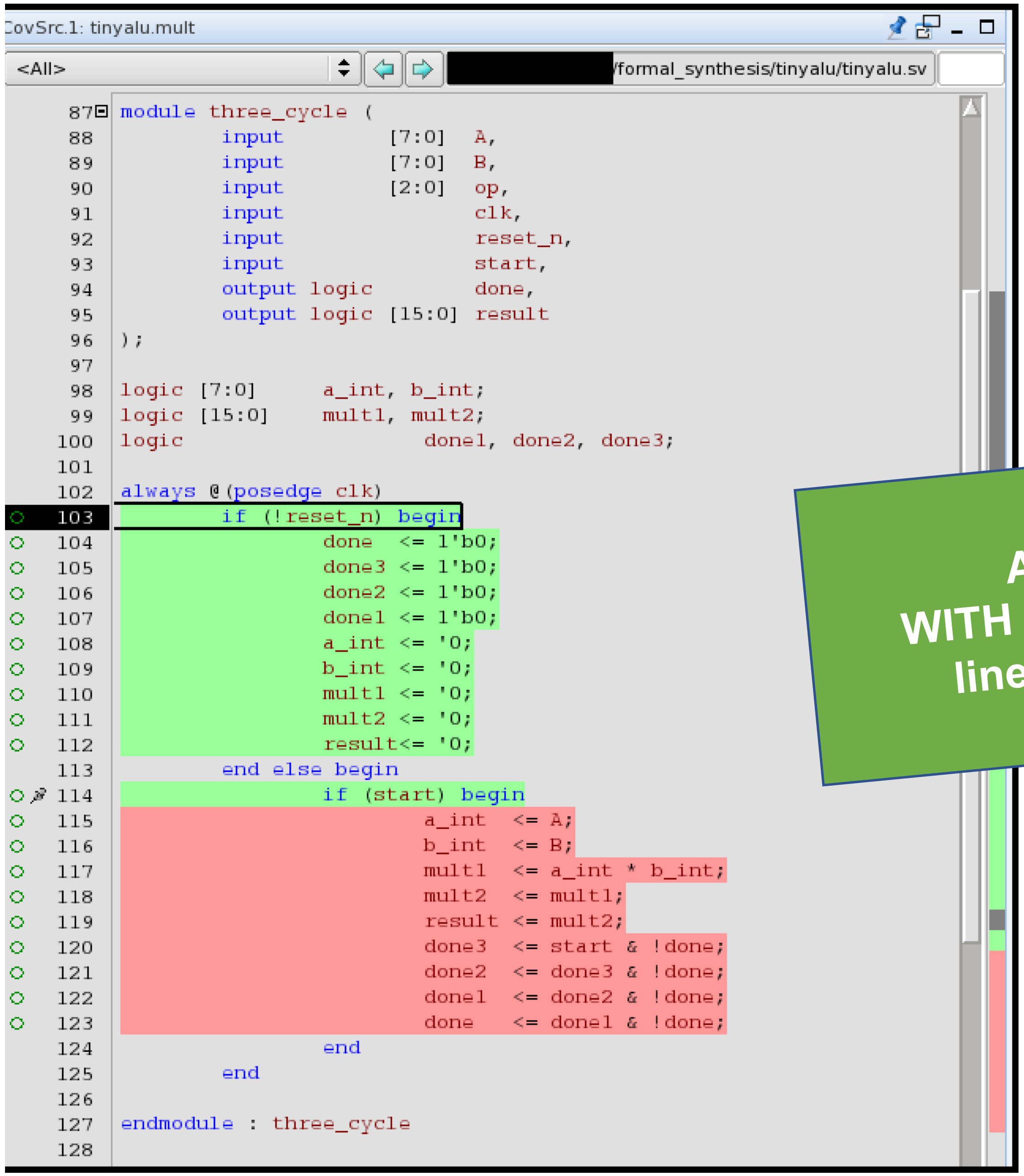
<All>   [redacted]/formal_synthesis/tinyalu/tinyalu.sv

```

module three_cycle (
        input           [7:0]   A,
        input           [7:0]   B,
        input           [2:0]   op,
        input                   clk,
        input                   reset_n,
        input                   start,
        output logic            done,
        output logic [15:0]     result
);

logic [7:0]     a_int, b_int;
logic [15:0]    mult1, mult2;
logic                   done1, done2, done3;

always @(posedge clk)
        if (!reset_n) begin
                done   <= 1'b0;
                done3 <= 1'b0;
                done2 <= 1'b0;
                done1 <= 1'b0;
                a_int <= '0;
                b_int <= '0;
                mult1 <= '0;
                mult2 <= '0;
                result<= '0;
        end else begin
                if (start) begin
                        a_int   <= A;
                        b_int   <= B;
                        mult1   <= a_int * b_int;
                        mult2   <= mult1;
                        result  <= mult2;
                        done3   <= start & !done;
                        done2   <= done3 & !done;
                        done1   <= done2 & !done;
                        done    <= done1 & !done;
                end
        end

endmodule : three_cycle

```

Formal analysis shows in green "reachable" lines of RTL

100% of this module is reachable


Start
Design
Formal Code Coverage Analysis
Identify Unreachable Lines caused by assumes
Comment Unreachable RTL Lines created by assumes
Design (edited)
Stimulus assume statements
Design Synthesis
Synthesized Netlist



CovSrc.1: tinyalu.mult

<All>

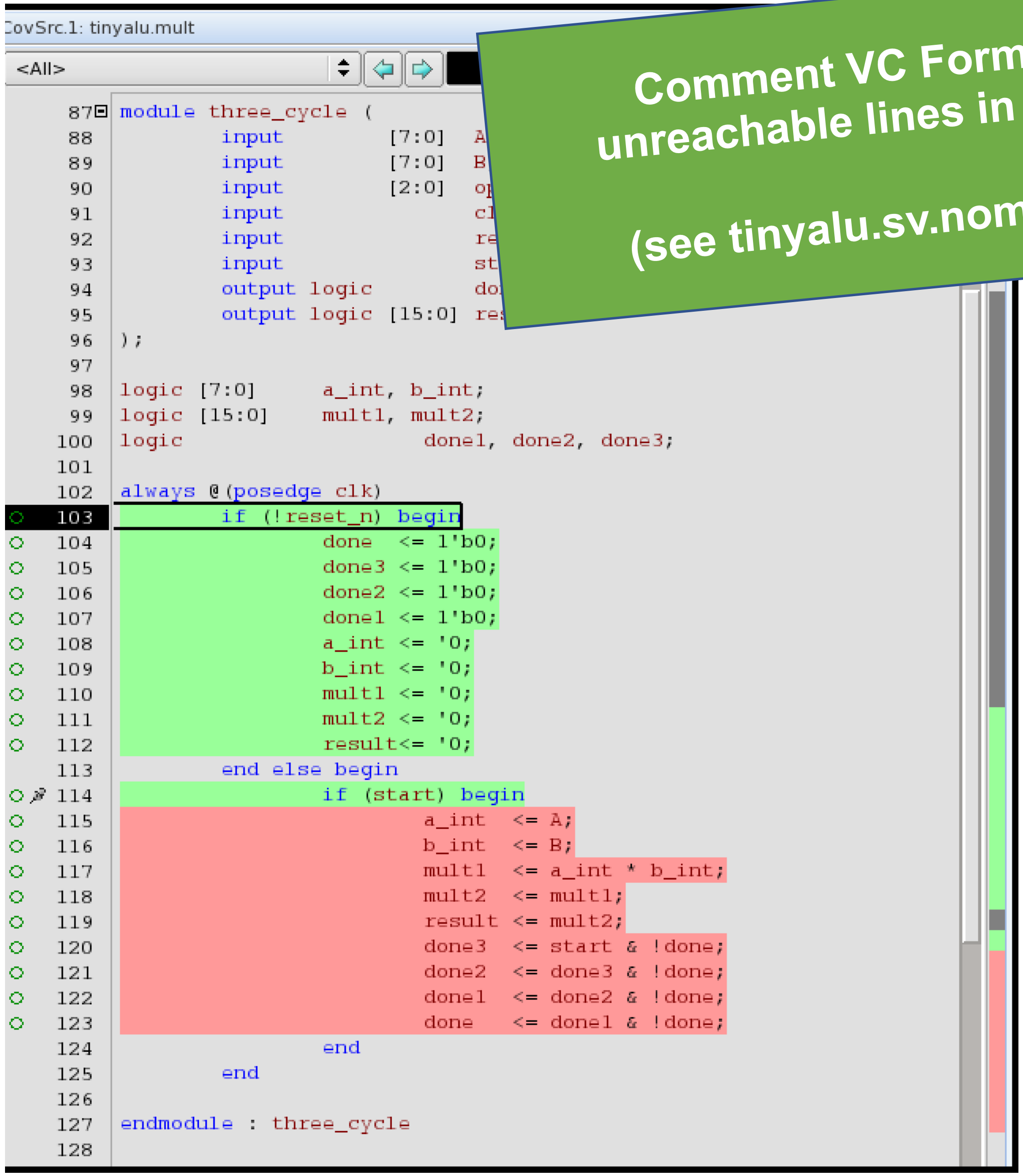
[redacted]/formal_synthesis/tinyalu/tinyalu.sv

```
 87 module three_cycle (
 88         input           [7:0]   A,
 89         input           [7:0]   B,
 90         input           [2:0]   op,
 91         input                   clk,
 92         input                   reset_n,
 93         input                   start,
 94         output logic            done,
 95         output logic    [15:0]  result
 96 );
 97
 98 logic  [7:0]      a_int, b_int;
 99 logic  [15:0]     mult1, mult2;
100 logic                     done1, done2, done3;
101
102 always @(posedge clk)
103         if (!reset_n) begin
104                 done  <= 1'b0;
105                 done3 <= 1'b0;
106                 done2 <= 1'b0;
107                 done1 <= 1'b0;
108                 a_int <= '0;
109                 b_int <= '0;
110                 mult1 <= '0;
111                 mult2 <= '0;
112                 result<= '0;
113         end else begin
114                 if (start) begin
115                         a_int  <= A;
116                         b_int  <= B;
117                         mult1  <= a_int * b_int;
118                         mult2  <= mult1;
119                         result <= mult2;
120                         done3  <= start & !done;
121                         done2  <= done3 & !done;
122                         done1  <= done2 & !done;
123                         done   <= done1 & !done;
124                 end
125         end
126
127 endmodule : three_cycle
128
```

After running Formal analysis WITH stimulus assumes we can see that line 115 (red) is no longer reachable

Comment VC Formal unreachable lines in RTL

(see tinyalu.sv.nomult)


CovSrc.1: tinyalu.mult

<All>

```
module three_cycle (
        input         [7:0]   A
        input         [7:0]   B
        input         [2:0]   op
        input                 cl
        input                 re
        input                 st
        output logic          do
        output logic [15:0]   re
);

logic  [7:0]     a_int, b_int;
logic  [15:0]    mult1, mult2;
logic                   done1, done2, done3;

always @(posedge clk)
        if (!reset_n) begin
                done   <= 1'b0;
                done3  <= 1'b0;
                done2  <= 1'b0;
                done1  <= 1'b0;
                a_int  <= '0;
                b_int  <= '0;
                mult1  <= '0;
                mult2  <= '0;
                result<= '0;
        end else begin
                if (start) begin
                        a_int   <= A;
                        b_int   <= B;
                        mult1   <= a_int * b_int;
                        mult2   <= mult1;
                        result  <= mult2;
                        done3   <= start & !done;
                        done2   <= done3 & !done;
                        done1   <= done2 & !done;
                        done    <= done1 & !done;
                end
        end

endmodule : three_cycle
```

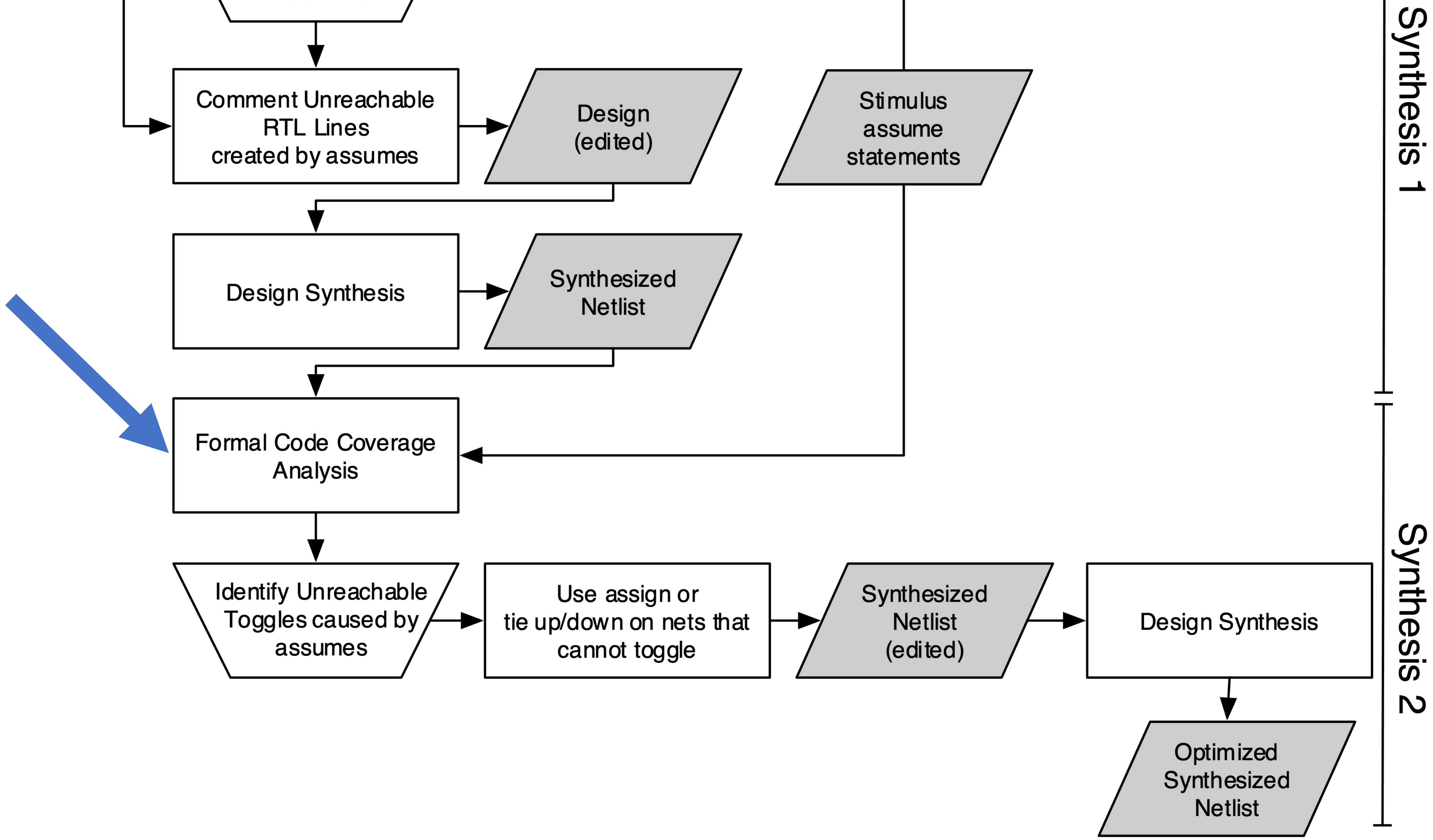
```
logic [7:0]     a_int, b_int;
logic [15:0]    mult1, mult2;
logic                   done1, done2, done3;

always @(posedge clk)
        if (!reset_n) begin
                done  <= 1'b0;
                done3 <= 1'b0;
                done2 <= 1'b0;
                done1 <= 1'b0;
                a_int <= '0;
                b_int <= '0;
                mult1 <= '0;
                mult2 <= '0;
                result<= '0;
        end else begin
           if (start) begin
                // VC FORMAL EXCLUDED LINES
                //      a_int   <= A;
                //      b_int   <= B;
                //      mult1   <= a_int * b_int;
                //      mult2   <= mult1;
                //      result  <= mult2;
                //      done3   <= start & !done;
                //      done2   <= done3 & !done;
                //      done1   <= done2 & !done;
                //      done    <= done1 & !done;
           end
        end
```

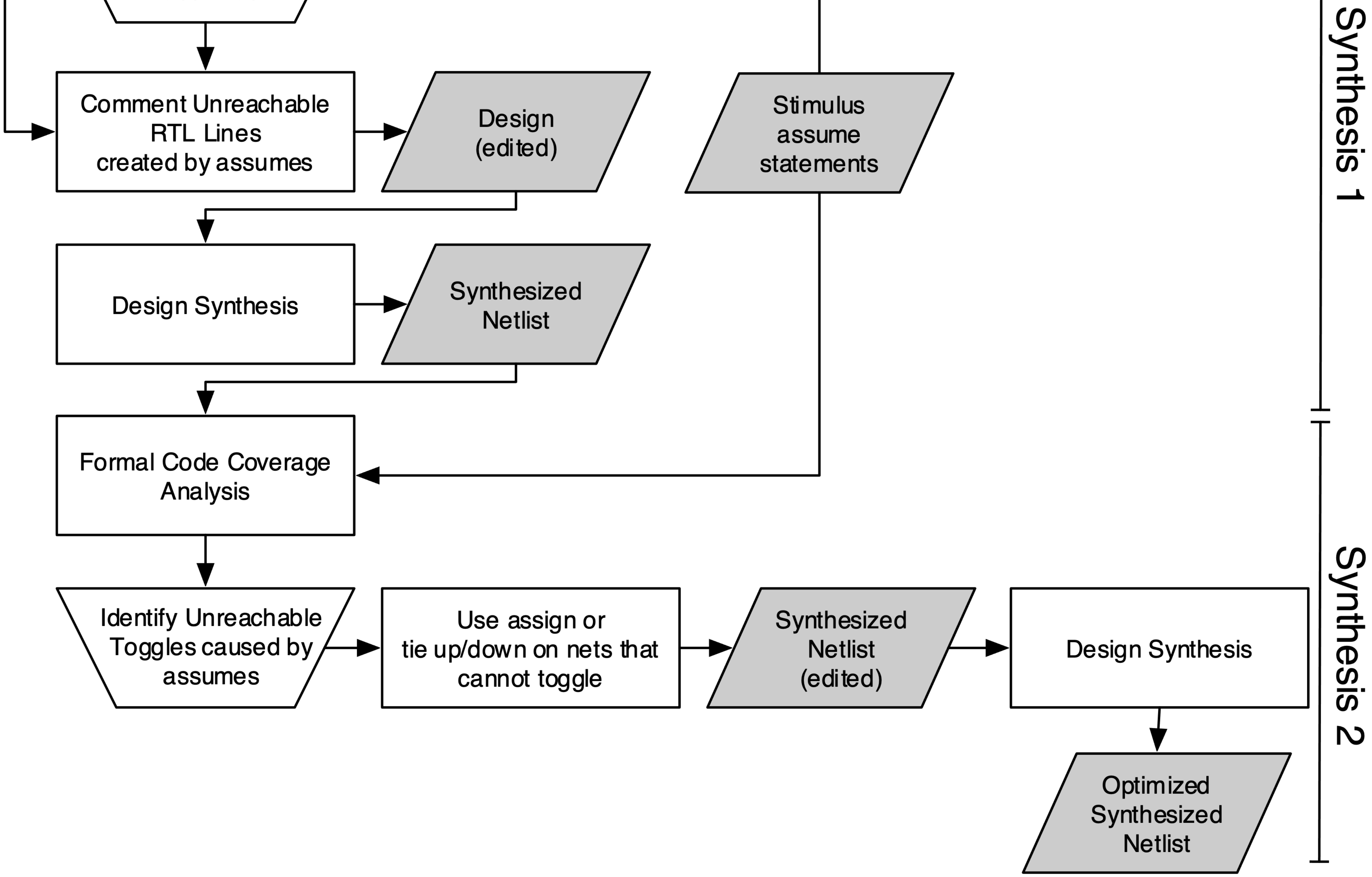
Run Synthesis

```
> make tinyalu_synthesis
> make tinyalu_synthesis_ex1
```

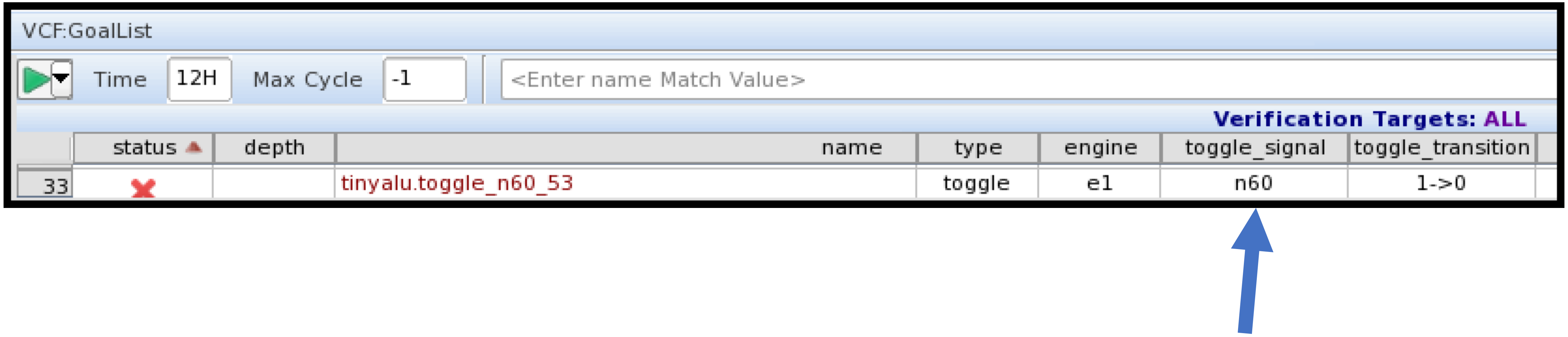
Optimizations versus Cell Count

Design	Cells	Combinational Cells	Sequential Cells	Total Cell Area
Original RTL (synthesis)	520	439	78	1377
Removed Multiplier Lines RTL (synthesis_ex1)	127	115	10	244
Improvement over Original RTL	4.1x	3.8x	7.8x	5.6x


Comment Unreachable RTL Lines created by assumes
Design (edited)
Stimulus assume statements
Design Synthesis
Synthesized Netlist
Formal Code Coverage Analysis
Identify Unreachable Toggles caused by assumes
Use assign or tie up/down on nets that cannot toggle
Synthesized Netlist (edited)
Design Synthesis
Optimized Synthesized Netlist
Synthesis 1
Synthesis 2


Comment Unreachable RTL Lines created by assumes
Design (edited)
Stimulus assume statements
Design Synthesis
Synthesized Netlist
Formal Code Coverage Analysis
Identify Unreachable Toggles caused by assumes
Use assign or tie up/down on nets that cannot toggle
Synthesized Netlist (edited)
Design Synthesis
Optimized Synthesized Netlist
Synthesis 1
Synthesis 2

Running Toggle Analysis on the Synthesized Netlist from Synthesis 1 with Assumes



VCF:GoalList

Time 12H Max Cycle -1 <Enter name Match Value>

Verification Targets: ALL

	status	depth	name	type	engine	toggle_signal	toggle_transition
33	✗		tinyalu.toggle_n60_53	toggle	e1	n60	1->0

Edit Gate Level Netlist with Input from Formal Coverage Analysis

```
// BEFORE
// ND2 U60 ( .A(op[0]), .B(n63), .Z(n60) );

// AFTER assign n60 to constant
assign n60 = 1'b1;
```

Edit Gate Level Netlist with Input from Formal Coverage Analysis

```
// BEFORE
// ND2 U60 ( .A(op[0]), .B(n63), .Z(n60) );

// AFTER assign n60 to constant
assign n60 = 1'b1;
```

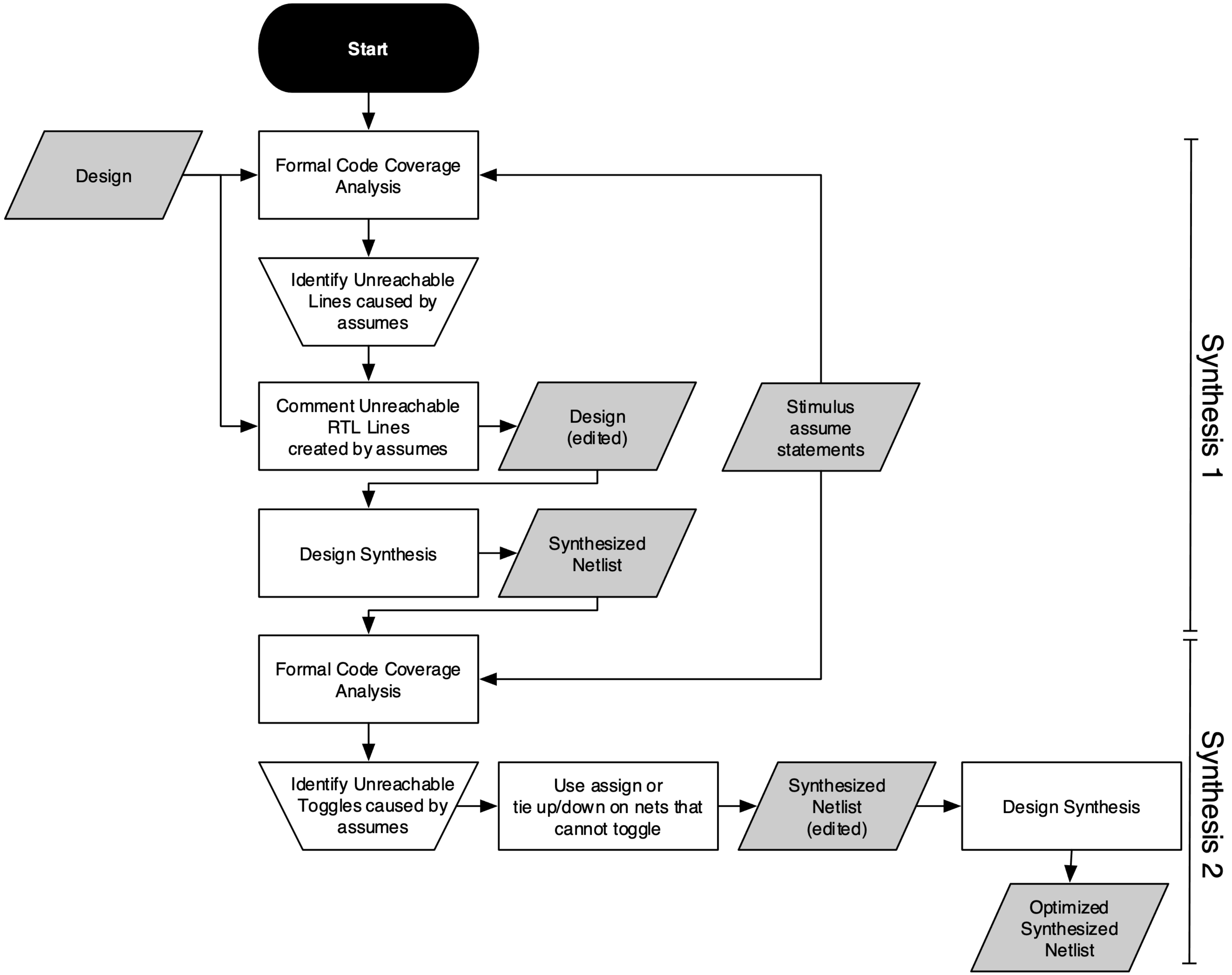
For more details on proof with this example see Paper

Run Synthesis

```
> make tinyalu_synthesis_ex2
```

Optimizations versus Cell Count

Design	Cells	Combinational Cells	Sequential Cells	Total Cell Area
Original RTL (synthesis)	520	439	78	1377
Removed Multiplier Lines RTL (synthesis_ex1)	127	115	10	244
Improvement over Original RTL	4.09x	3.82	7.80	5.64
Above with Toggle Removal (synthesis_ex2)	103	91	10	196
Improvement with Toggle Removal	5.05x	4.82	7.80	7.03


Start
Design
Formal Code Coverage Analysis
Identify Unreachable Lines caused by assumes
Comment Unreachable RTL Lines created by assumes
Design (edited)
Stimulus assume statements
Design Synthesis
Synthesized Netlist
Formal Code Coverage Analysis
Identify Unreachable Toggles caused by assumes
Use assign or tie up/down on nets that cannot toggle
Synthesized Netlist (edited)
Design Synthesis
Optimized Synthesized Netlist
Synthesis 1
Synthesis 2

Why Line Coverage with RTL?

- Line coverage with RTL is done because it is easy to automate adding comments RTL
- All signals names are still available at with original RTL
    - Easy to write assumes deep in design and at ports

Why Not Line Coverage with Gates?

- Line coverage does not exist in a gate level netlist!
- Gate level netlists do not ensure signal names remain
  - Assumes may no longer be valid or have access to internal signals

Why Toggle with Gates?

- Toggle coverage with Gate Level netlists because it is easy to automate
- Rewriting RTL is complicated
  - Would likely result in unreadable code
  - Imagine how breaking of arrays and reconnecting them in SystemVerilog at the bit level
- Toggle at the gate level can reveal deeper optimizations at the bit level instead of RTL which thinks about full arrays at a time

Usage Warning

- Removing functionality from the DUT with this method means that the design will have undetermined / invalid behavior for certain stimulus

- In our example, we asserted that there will never be a OP_MULT stimulus
- If you send in an OP_MULT the behavior will now be unknown

Assumes to Asserts

- Pushing the assumes to asserts at higher levels of integration to maintain the functionality of the optimized design
- Need to guarantee that the optimized gate netlists will never encounter stimulus that are explicitly forbidden (assumes)

Summary

- This paper describes a method to use allowed stimulus of a DUT to optimize the implementation of the synthesized gate level netlist
- Methodology can be automated because of deliberate decisions

- Quickly create a table for how different features or even stimulus patterns in sequential time can effect the implementation in terms of gates needed to implement

Future Work

- There are other types of code coverage analysis not used in this flow
  - Notably, condition coverage which may yield more optimization possibilities
- Refining the scripting to implement this flow
- Future products

Thank you and Questions