PROJECT #2: Equivalence Checking with Formality

ECE 582

Abram Fouts

Mike Tian

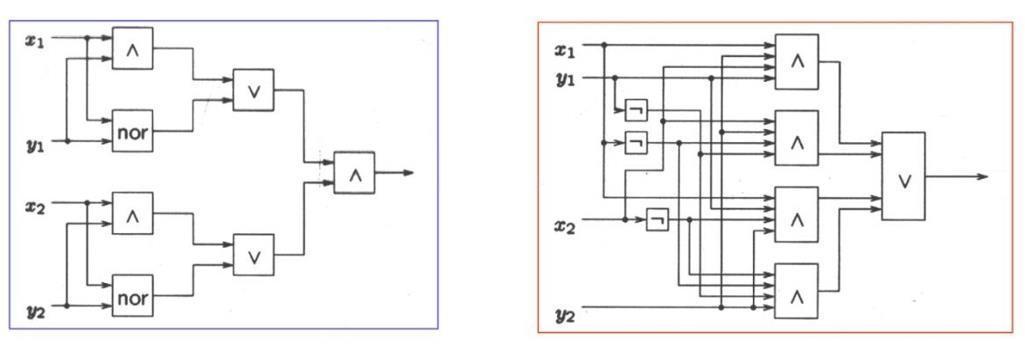
February 4th, 2021

**Background**

Using system verilog / verilog code compiled in modelSim was used to create design files for Formality. Formality is used to check for the equivalence between circuits, netlists, libraries, and many other items.

**Task 1:** Describe the following two circuits C1 and C2 in Verilog. Use the formality to check if they are equivalent. You need to use the same names on the corresponding inputs and outputs.

**Task 1.1:**



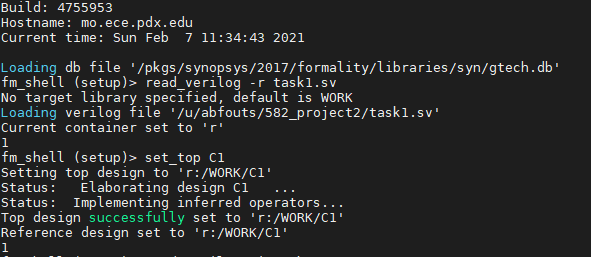
**.sv code**

|  |
| --- |
| **module C1(X1, Y1, X2, Y2, OUT);**  **input X1, Y1, X2, Y2;**  **output OUT;**  **wire AND1, NOR1, OR1;**  **wire AND2, NOR2, OR2;**  **assign AND1 = X1 & Y1;**  **assign NOR1 = ~(X1 | Y1);**  **assign AND2 = X2 & Y2;**  **assign NOR2 = ~(X2 | Y2);**  **assign OR1 = AND1 | NOR1;**  **assign OR2 = AND2 | NOR2;**  **assign OUT = OR1 & OR2;**  **endmodule** |

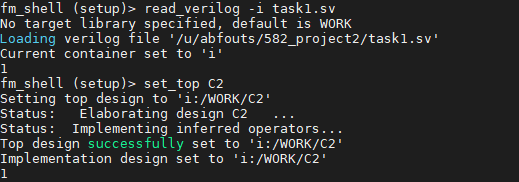
|  |
| --- |
| **module C2(X1, Y1, X2, Y2, OUT);**  **input X1, Y1, X2, Y2;**  **output OUT;**  **wire AND1, AND2, AND3, AND4;**  **assign AND1 = X1 & X2 & Y1 & Y2;**  **assign AND2 = ~X1 & X2 & ~Y1 & Y2;**  **assign AND3 = X1 & ~X2 & Y1 & Y2;**  **assign AND4 = ~X1 & ~X2 & ~Y1 & Y2;**  **assign OUT = AND1 | AND2 | AND3 | AND4;**  **endmodule** |

**Formality Screenshots**

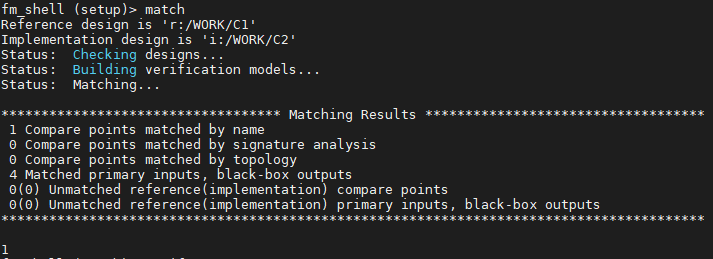
Read Verilog - loading reference design



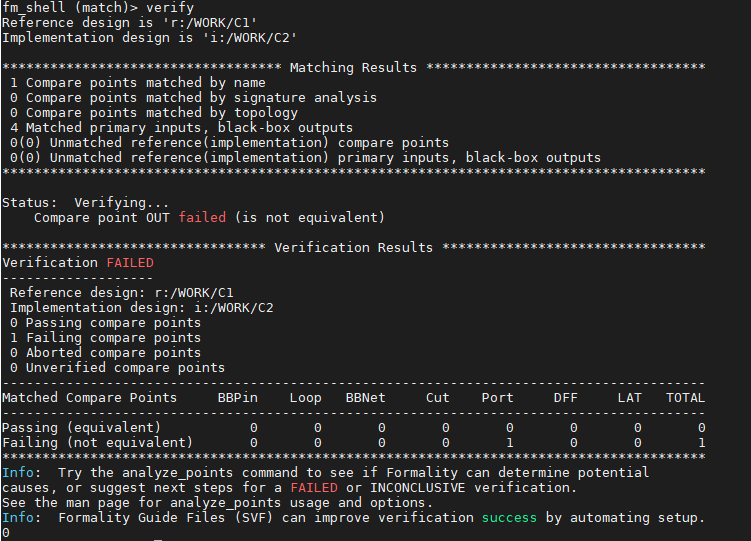
Read Verilog - loading implementation design



Match - running Formality’s matching algorithms



Verify - running Formality’s verification algorithms



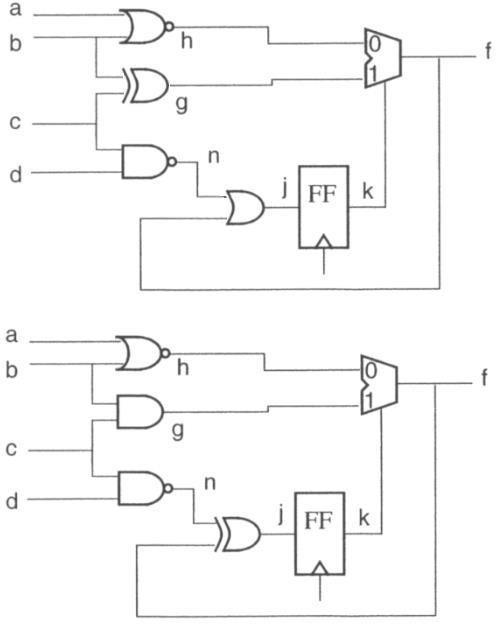
Formality Verification Process:

* Reading the designs - setting reference and implementation design
  + fm\_shell (setup) > read\_verilog - r
  + fm\_shell (setup) > read\_verilog - i
* Setup - set up constants / disable scan logic or other typical setup steps
* Match - run Formality’s matching algorithms to compare points
  + fm\_shell (setup) > match
* Verify - run Formality’s verification algorithms
  + fm\_shell (setup) > verify

Verification FAILED was returned. The implementation design C2 is not equivalent to the reference design C1. The circuits C1 and C2 are not equivalent.

**Task 2:** Use the formality to check if two sequential circuits are equivalent.

**Task 2.1:** Perform the equivalence check on the following two sequential circuits with formality.



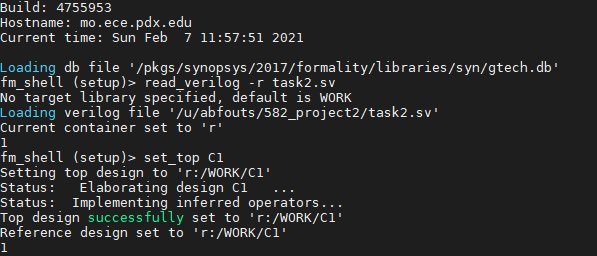
**.sv code**

|  |
| --- |
| **module C1(a, b, c, d, f, clk);**  **input a, b, c, d, clk;**  **output f;**  **wire h, g, n;**  **reg k, j;**  **assign h = ~(a | b);**  **assign g = (b ^ c);**  **assign n = ~(c & d);**  **assign j = (n | f);**  **assign f = (k) ? g : h;**  **always@(posedge clk) begin**  **#2 k <= j;**  **end**  **endmodule** |

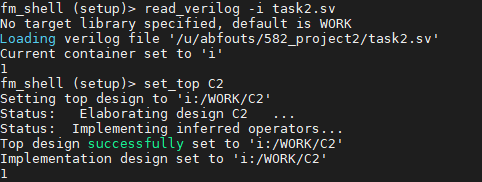
|  |
| --- |
| **module C2(a, b, c, d, f, clk);**  **input a, b, c, d, clk;**  **output f;**  **wire h, g, n;**  **reg k, j;**  **assign h = ~(a | b);**  **assign g = (b & c);**  **assign n = ~(c & d);**  **assign j = (n ^ f);**  **assign f = (k) ? g : h;**  **always@(posedge clk) begin**  **#2 k <= j;**  **end**  **endmodule** |

**Screenshots**

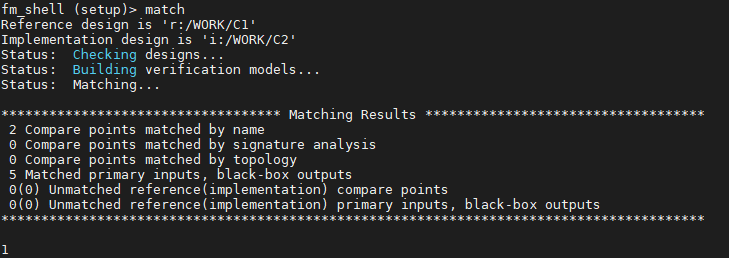
Read Verilog

****

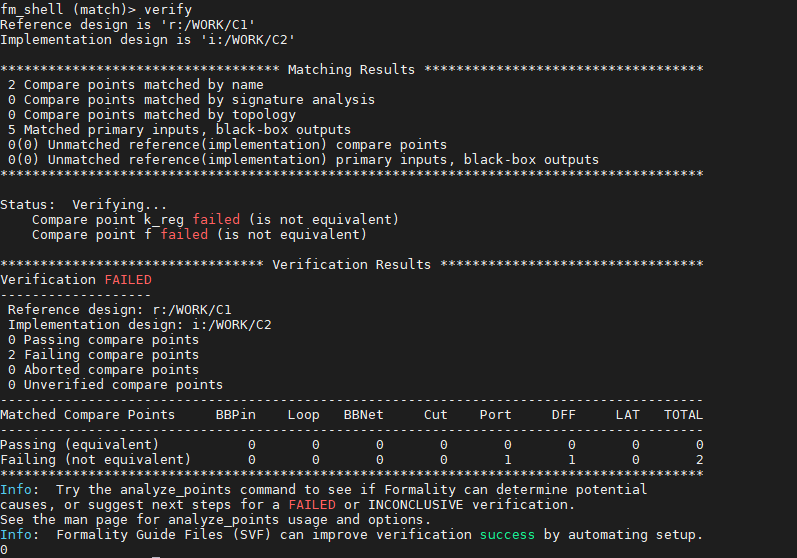
Read Implementation

****

Match



Verify



**Process**

Based off of the description in the tutorial when Formality returns a *Verification: FAILED* or *Verification: SUCCEEDED* determines whether or not the circuits are equivalent or not. The process was to create individual modules of both *C1* and *C2* circuits, and implement both circuits into the reference and implementation sections of Formality. The .sv file is sent via the command line arguments:

1. fm\_shell
2. read\_verilog -r {file.sv}
3. set\_top {module (C1)}
4. read\_verilog -i {file.sv}
5. set\_top {module (C2)}
6. match
7. verify

Each command was described in the tutorial to complete a basic verification. Since each module is stored in a single .sv file *-auto* was not used during the first *set\_top*, so Formality knew to use the correct module.