

Software-Based ASIC Counter Design Using Verilog & Yosys

1. Introduction

This project demonstrates an **end-to-end ASIC-style digital design flow using only software tools**. A simple 4-bit synchronous counter is designed using Verilog HDL, synthesized into gate-level logic using Yosys, and visualized as a circuit diagram.

2. Objective

- Design a basic digital circuit (4-bit counter)
 - Implement it using **Verilog HDL**
 - Perform logic synthesis using **Yosys**
 - Generate and visualize a **gate-level circuit diagram**
 - Understand the internal working of a digital chip
-

3. Tools & Technologies Used

Verilog HDL : Hardware Description Language used to describe the digital circuit.

Yosys : Open-source synthesis tool used to convert RTL (Verilog) into a gate-level netlist.

ABC : Technology mapping and logic optimization tool (used internally by Yosys).

Graphviz : Tool used by Yosys to generate visual circuit diagrams.

Ubuntu (WSL) : Linux-based environment used for running EDA tools.

Windows + WSL : Host operating system integrating Windows with Linux tools.

4. System Requirements

- Windows 10/11
- WSL (Windows Subsystem for Linux)
- Ubuntu installed via WSL
- Basic terminal knowledge

5. Project Workflow (ASIC Design Flow)

- **Step 1:** Write Verilog HDL code (RTL Design)
 - **Step 2:** Synthesize RTL to logic gates using Yosys
 - **Step 3:** Optimize and map logic using ABC
 - **Step 4:** Generate gate-level netlist
 - **Step 5:** Visualize circuit as a diagram (PNG)
-

6. Flowchart

```
Start
↓
Write Verilog Code
↓
RTL Synthesis (Yosys)
↓
Logic Optimization
↓
Gate-Level Netlist
↓
Circuit Diagram (PNG)
↓
End
```

7. Verilog Design Explanation

7.1 Design Description

The circuit is a **4-bit synchronous up-counter** with: - Clock input (clk) - Asynchronous reset (reset) - 4-bit output (count)

7.2 Verilog Code



```
GNU nano 7.2                               counter.v
mihika@DESKTOP-GHJ3BLN: ~ + v
module counter (
    input clk,
    input reset,
    output reg [3:0] count
);

always @(posedge clk or posedge reset) begin
    if (reset)
        count <= 4'b0000;
    else
        count <= count + 1;
end

endmodule

[ Wrote 14 lines ]
^G Help      ^O Write Out  ^W Where Is  ^K Cut      ^T Execute
^X Exit      ^R Read File  ^\ Replace   ^U Paste   ^J Justify
```

7.3 Working

- On every **positive clock edge**, the counter increments by 1
 - When **reset = 1**, the counter immediately goes back to 0
-

8. Synthesis Process (Yosys)

8.1 Command Used

```
yosys -p "read_verilog counter.v; synth; show -format png -prefix counter_design"
```

8.2 What This Command Does

- Reads Verilog code
- Converts it into logic gates
- Optimizes the logic
- Generates a circuit diagram (counter_design.png)

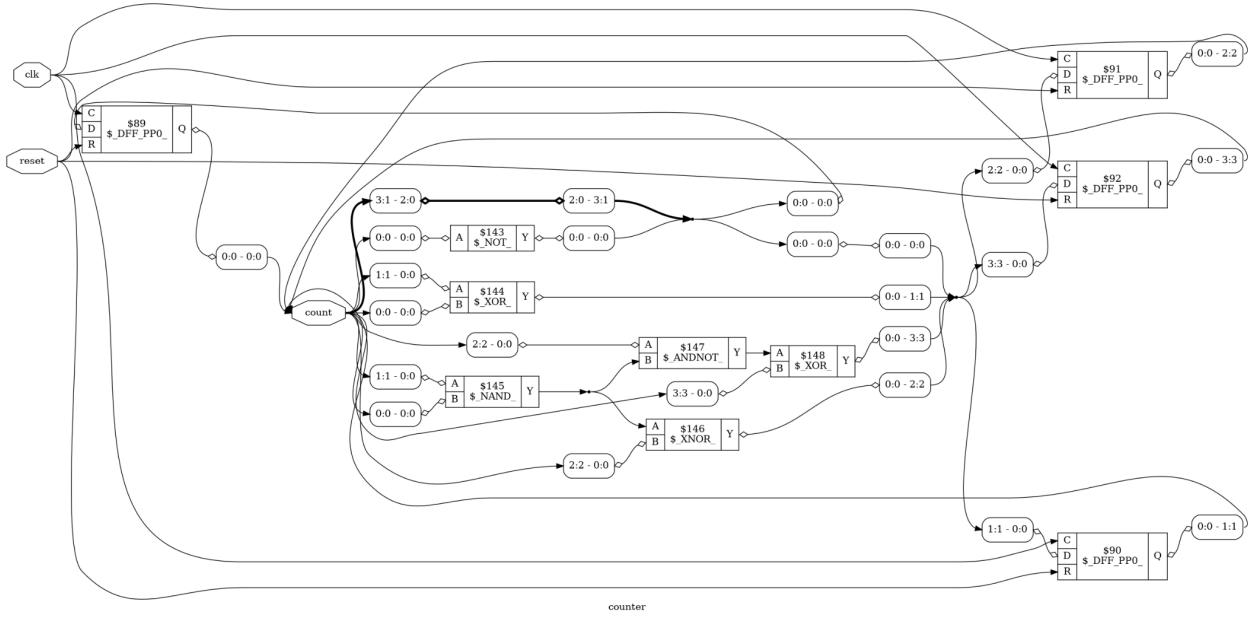
The image contains two side-by-side screenshots of Windows PowerShell windows. Both windows have the title bar 'mihika@DESKTOP-GHJ3I' and the tab 'Windows PowerShell'.
The left window shows the following log output:
2.21.5. Finished fast OPT passes.
2.22. Executing ABC pass (technology mapping using ABC).
2.22.1. Extracting gate netlist of module 'counter' to '<abc-temp-dir>/input.blif'..
Extracted 6 gates and 10 wires to a netlist network with 4 inputs and 4 outputs.
2.22.1.1. Executing ABC.
Running ABC command: "yosys-exe-dir>/yosys-abc" -s -f <abc-temp-dir>/abc.script 2>&1
ABC: ABC command line: "source <abc-temp-dir>/abc.script".
ABC:
ABC: + read_blif <abc-temp-dir>/input.blif
ABC: + read_library <abc-temp-dir>/stdcells.genlib
ABC: Entered genlib library with 13 gates from file "<abc-temp-dir>/stdcells.genlib".
ABC: + strash
ABC: + dftime
ABC: + map
ABC: + write_blif <abc-temp-dir>/output.blif
2.22.1.2. Re-integrating ABC results.
ABC RESULTS: NOT cells: 1
ABC RESULTS: NAND cells: 1
ABC RESULTS: XNOR cells: 1
ABC RESULTS: ANDNOT cells: 1
ABC RESULTS: XOR cells: 2
ABC RESULTS: internal signals: 2
ABC RESULTS: input signals: 4
ABC RESULTS: output signals: 4
Removing temp directory.

The right window shows the following log output:
2.25. Printing statistics.
==== counter ====
Number of wires: 7
Number of wire bits: 16
Number of public wires: 3
Number of public wire bits: 6
Number of memories: 0
Number of memory bits: 0
Number of processes: 0
Number of cells: 10
\$._ANDNOT_ 1
\$._DFP_PPO_ 4
\$._NAND_ 1
\$._NOT_ 1
\$._XNOR_ 1
\$._XOR_ 2
2.26. Executing CHECK pass (checking for obvious problems).
Checking module counter...
Found and reported 0 problems.
3. Executing Verilog backend.
3.1. Executing BMUXMAP pass.
3.2. Executing DEMUXMAP pass.
Dumping module 'counter'.
End of script. Logfile hash: ba56db6738, CPU: user 0.12s system 0.04s, M
EM: 13.12 MB peak
Yosys 0.33 (git sha1 2584903a060)
Time spent: 30% 1x abc (0 sec), 14% 3x read_verilog (0 sec), ...

statistics

9. Output Files Generated

- **counter.v** : Verilog source file describing the counter circuit
- **counter_netlist.v** : Gate-level netlist generated after synthesis
- **counter_design.dot** : Graphviz (.dot) file representing the synthesized design
- **counter_design.png** : Visual gate-level circuit diagram generated from the .dot file



10. Gate-Level Diagram Explanation

Key Components Seen in Diagram:

- **D Flip-Flops (\$_DFF_PPO_)** → Store counter bits
- **Logic Gates (XOR, XNOR, NAND, NOT)** → Implement addition logic
- **Clock & Reset Lines** → Control sequential behavior

Important Observation

The feedback loop from flip-flop output → logic → flip-flop input shows how the counter updates its value every clock cycle.

11. What This Project Replaces

Traditional Hardware	Software Replacement
----------------------	-------------------------

Physical chip fabrication Software simulation

Logic analyzer Gate-level diagram

FPGA board Verilog + Yosys

12. Applications

- Learning digital electronics
 - Understanding ASIC/VLSI flow
 - Resume & internship projects
 - Foundation for advanced chip design
-

13. Conclusion

This project demonstrates the complete workflow for designing, synthesizing, and visualizing a digital chip using a software-based ASIC design flow. The results validate the effectiveness of using industry-standard open-source tools to implement and analyze digital logic designs from RTL to gate-level representation.

14. Future Scope

- Add timing analysis
- Use OpenROAD for physical design
- Implement larger digital blocks
- FPGA implementation