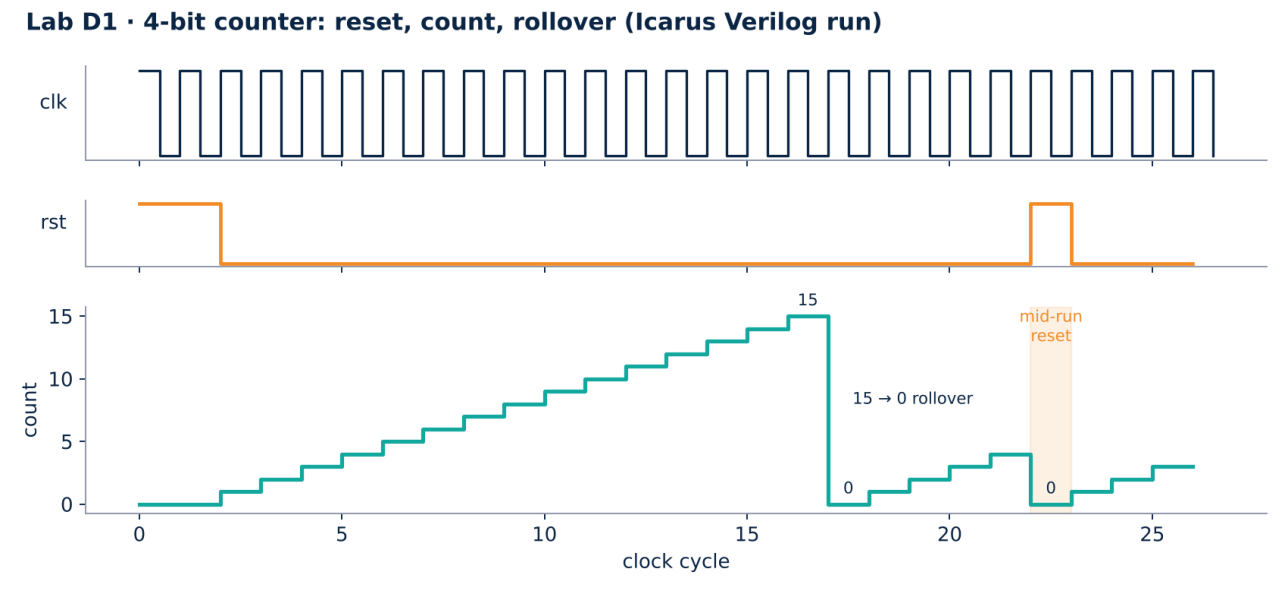
BREAK INTO CHIP DESIGN · FREE

Starter Lab Manual

All four lab files, the commands to run them, and the expected output



D1 4-bit counter · Icarus Verilog

D2 FIFO bug hunt · Icarus Verilog

A1 RC low-pass filter · ngspice

A2 CMOS inverter · ngspice

STARTER LAB MANUAL

Run all four labs in about an hour

Every file is printed in full below, so you can copy it even on a phone. Two ways to run them:

- **No install (Verilog labs):** open edaplayground.com, sign in free, choose *Icarus Verilog*, paste the design into the right pane and the testbench into the left, press Run.
- **On your laptop:** Ubuntu or Windows WSL, then `sudo apt install iverilog ngspice gtkwave`. Save each listing with the file name shown and run the commands.
- **Expected output** is shown for every lab. If yours differs, compare line by line — that is debugging practice too.

The fixed FIFO differs from the buggy one in three places: the write condition becomes `wr_en && !full`, the same change is made in the count logic, and the count simply increments. Try making the fix yourself before reading `fifo_fixed.v`.

© 2026 Skill N Success · skillnsuccess.com · Educational resource, not a job-placement guarantee.

LAB D1 · DIGITAL

4-bit counter

Tool: Icarus Verilog

counter4.v

```
// Lab D1: 4-bit counter with synchronous reset
module counter4 (
    input  wire       clk,
    input  wire       rst,
    output reg  [3:0] count
);
  always @(posedge clk) begin
    if (rst)
      count <= 4'd0;
    else
      count <= count + 1'b1;
  end
endmodule
```

tb_counter4.v

```
`timescale 1ns/1ps
module tb_counter4;
  reg clk = 0, rst = 1;
  wire [3:0] count;
  integer errors = 0, cycle = 0, sample = 0;
  counter4 dut(.clk(clk), .rst(rst), .count(count));
  always #5 clk = ~clk;               // 100 MHz clock (10 ns period)
  initial begin
    $dumpfile("counter4.vcd"); $dumpvars(0, tb_counter4);
    $display("cycle,rst,count");
    repeat (2) @(posedge clk);         // hold reset for 2 cycles
    #1 rst = 0;
    repeat (20) @(posedge clk);        // 20 counts -> must roll over 15 -> 0
    #1 rst = 1;                        // mid-run reset
    @(posedge clk); #1 rst = 0;
    repeat (3) @(posedge clk);
    #2;
    if (errors == 0) $display("RESULT: PASS - reset and rollover verified (%0d checks)", cycle);
    else $display("RESULT: FAIL - %0d errors", errors);
    $finish;
  end
  // self-checking: compare with expected value after every edge
  always @(posedge clk) begin
    #1;
    sample = sample + 1; $display("%0d,%0d,%0d", sample, rst, count);
  end
  reg [3:0] model = 0;
  always @(posedge clk) begin
    model <= rst ? 4'd0 : model + 1'b1;
  end
  always @(negedge clk) begin
    cycle = cycle + 1;
    if (count !== model) begin
      errors = errors + 1;
      $display("MISMATCH at cycle %0d: count=%0d expected=%0d", cycle, count, model);
    end
  end
endmodule
```

Run

```
iverilog -o sim counter4.v tb_counter4.v
vvp sim
```

Expected output

```
RESULT: PASS - reset and rollover verified (25 checks)
```

© 2026 Skill N Success · skillnsuccess.com · Educational resource, not a job-placement guarantee.

LAB D2 · VERIFICATION

FIFO bug hunt

Tool: Icarus Verilog

fifo_buggy.v

```
// Lab D2: 4-entry synchronous FIFO -- contains ONE deliberate bug.
module fifo #(parameter W = 8, D = 4) (
    input  wire         clk, rst,
    input  wire         wr_en, rd_en,
    input  wire [W-1:0] din,
    output reg  [W-1:0] dout,
    output wire         full, empty
);
  reg [W-1:0] mem [0:D-1];
  reg [1:0]   wptr, rptr;
  reg [2:0]   count;
  assign full  = (count == D);
  assign empty = (count == 0);
  always @(posedge clk) begin
    if (rst) begin
      wptr <= 0; rptr <= 0; count <= 0; dout <= 0;
    end else begin
      // BUG: a write is accepted even when the FIFO is full
      if (wr_en) begin
        mem[wptr] <= din;
        wptr <= wptr + 1'b1;
      end
      if (rd_en && !empty) begin
        dout <= mem[rptr];
        rptr <= rptr + 1'b1;
      end
      case ({wr_en, rd_en && !empty})
        2'b10: count <= (count == D) ? count : count + 1'b1;
        2'b01: count <= count - 1'b1;
        default: count <= count;
      endcase
    end
  end
endmodule
```

tb_fifo.v

© 2026 Skill N Success · skillnsuccess.com · Educational resource, not a job-placement guarantee.

```
`timescale 1ns/1ps
// Self-checking testbench with a reference queue model
module tb_fifo;
  reg clk = 0, rst = 1, wr_en = 0, rd_en = 0;
  reg [7:0] din = 0;
  wire [7:0] dout; wire full, empty;
  fifo dut(.clk(clk), .rst(rst), .wr_en(wr_en), .rd_en(rd_en), .din(din), .dout(dout), .full(full), .empty(empt
  always #5 clk = ~clk;
  reg [7:0] model [0:15]; integer head = 0, tail = 0, errors = 0, i;
  task push(input [7:0] v); begin
    @(negedge clk); wr_en = 1; din = v;
    if (!full) begin model[tail % 16] = v; tail = tail + 1; end
    $display("WRITE %0d  (full=%0b before write)", v, full);
    @(negedge clk); wr_en = 0;
  end endtask
  task pop; reg [7:0] exp; begin
    @(negedge clk); rd_en = 1;
    @(negedge clk); rd_en = 0;
    exp = model[head % 16]; head = head + 1;
    if (dout !== exp) begin errors = errors + 1; $display("READ  %0d  expected %0d  <-- MISMATCH", dout, exp);
    else $display("READ  %0d  expected %0d  ok", dout, exp);
  end endtask
  initial begin
    $dumpfile("fifo.vcd"); $dumpvars(0, tb_fifo);
    repeat (2) @(negedge clk); rst = 0;
    // Boundary test: write 5 items into a 4-deep FIFO (one too many)
    for (i = 1; i <= 5; i = i + 1) push(i * 10);
    // Read back the 4 items that should be stored
    for (i = 0; i < 4; i = i + 1) pop;
    if (errors == 0) $display("RESULT: PASS - full-boundary test passed");
    else $display("RESULT: FAIL - %0d mismatches at the full boundary", errors);
    $finish;
  end
endmodule
```

Run

```
iverilog -o sim fifo_buggy.v tb_fifo.v
vvp sim

# then the fixed design:
iverilog -o sim fifo_fixed.v tb_fifo.v
vvp sim
```

Expected output

```
Buggy:  READ 50 expected 10 <-- MISMATCH ... RESULT: FAIL
Fixed:  RESULT: PASS - full-boundary test passed
```

© 2026 Skill N Success · skillnsuccess.com · Educational resource, not a job-placement guarantee.

LAB A1 · ANALOG

RC low-pass filter

Tool: ngspice

rc_lowpass.cir

```
* Lab A1: first-order RC low-pass filter, R = 1k, C = 100n
V1 in 0 DC 0 AC 1 SIN(0 1 1k)
R1 in out 1k
C1 out 0 100n
.control
  ac dec 50 10 1meg
  let gain_db = db(v(out))
  meas ac fc WHEN gain_db=-3.0103
  wrdata rc_ac.txt gain_db
  tran 2u 4m
  wrdata rc_tran_1k.txt v(in) v(out)
  quit
.endc
.end
```

Run

```
ngspice -b rc_lowpass.cir
```

Expected output

```
fc = 1.591551e+03   (theory 1,591.5 Hz)
```

© 2026 Skill N Success · skillnsuccess.com · Educational resource, not a job-placement guarantee.

LAB A2 · ANALOG MEETS DIGITAL

CMOS inverter

Tool: ngspice

inverter.cir

```
* Lab A2: CMOS inverter, VDD = 1.8 V
* Teaching models (generic level-1 MOSFETs), not a foundry PDK.
.model nmos_t NMOS (LEVEL=1 VTO=0.45 KP=200u LAMBDA=0.05)
.model pmos_t PMOS (LEVEL=1 VTO=-0.45 KP=80u LAMBDA=0.05)
VDD vdd 0 1.8
VIN in 0 DC 0 PULSE(0 1.8 1n 50p 50p 4n 10n)
MP out in vdd vdd pmos_t W=2.5u L=0.18u
MN out in 0 0 nmos_t W=1u L=0.18u
CL out 0 20f
.control
  dc VIN 0 1.8 0.005
  wrdata inv_vtc.txt v(out)
  meas dc vm WHEN v(out)=v(in)
  tran 5p 12n
  wrdata inv_tran.txt v(in) v(out)
  meas tran tphl TRIG v(in) VAL=0.9 RISE=1 TARG v(out) VAL=0.9 FALL=1
  meas tran tplh TRIG v(in) VAL=0.9 FALL=1 TARG v(out) VAL=0.9 RISE=1
  alter CL 80f
  tran 5p 12n
  wrdata inv_tran_80f.txt v(in) v(out)
  meas tran tphl80 TRIG v(in) VAL=0.9 RISE=1 TARG v(out) VAL=0.9 FALL=1
  quit
.endc
.end
```

Run

```
ngspice -b inverter.cir
```

Expected output

```
vm = 0.900 V   tphl = 29.6 ps (20 fF)   tphl80 = 80.5 ps (80 fF)
```

© 2026 Skill N Success · skillnsuccess.com · Educational resource, not a job-placement guarantee.

BONUS

fifo_fixed.v

```
// Lab D2: 4-entry synchronous FIFO -- fixed version.
module fifo #(parameter W = 8, D = 4) (
    input  wire         clk, rst,
    input  wire         wr_en, rd_en,
    input  wire [W-1:0] din,
    output reg  [W-1:0] dout,
    output wire         full, empty
);
  reg [W-1:0] mem [0:D-1];
  reg [1:0]   wptr, rptr;
  reg [2:0]   count;
  assign full  = (count == D);
  assign empty = (count == 0);
  always @(posedge clk) begin
    if (rst) begin
      wptr <= 0; rptr <= 0; count <= 0; dout <= 0;
    end else begin
      // FIX: only write when there is space
      if (wr_en && !full) begin
        mem[wptr] <= din;
        wptr <= wptr + 1'b1;
      end
      if (rd_en && !empty) begin
        dout <= mem[rptr];
        rptr <= rptr + 1'b1;
      end
      case ({wr_en && !full, rd_en && !empty})
        2'b10: count <= count + 1'b1;
        2'b01: count <= count - 1'b1;
        default: count <= count;
      endcase
    end
  end
endmodule
```

rc_10k.cir

```
* Same filter driven at 10 kHz (above cutoff)
V1 in 0 SIN(0 1 10k)
R1 in out 1k
C1 out 0 100n
.control
  tran 0.5u 0.6m
  wrdata rc_tran_10k.txt v(in) v(out)
  quit
.endc
.end
```

© 2026 Skill N Success · skillnsuccess.com · Educational resource, not a job-placement guarantee.