Model Validation for Mixed-Signal Verification

Carsten Wegener

Dialog Semiconductor, Germany

Model Validation for Verification

Overview:

- Verification target clarification by example
- Digital model of an analog cell: functionality/structure
- Functionality modeling and validation
- Structure preserving model assembly
- Case study: finding bugs by modeling
- Conclusion

Verification target clarification

- Top-down specification

- Design phase

- Bottom-up verification

Verification target clarification

- Top-down specification
  - **Specification model** describes intended behavior
  - Verify that block specs match system requirements
- Design phase
- Bottom-up verification

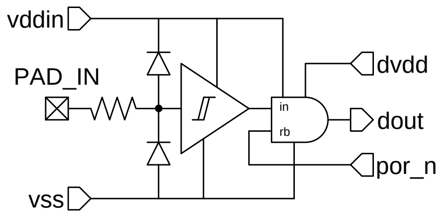
Verification target clarification

- Top-down specification
  - **Specification model** describes intended behavior
  - Verify that block specs match system requirements
- Design phase
  - Adjust parameters of chosen design architecture
  - Verify performance vs. specification
- Bottom-up verification

Verification target clarification

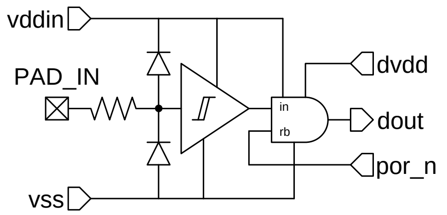
- Top-down specification
  - **Specification model** describes intended behavior
  - Verify that block specs match system requirements
- Design phase
  - Adjust parameters of chosen design architecture
  - Verify performance vs. specification
- Bottom-up verification
  - **Implementation model**: functionality and structure
  - Verify integration of implemented block is robust in system

Simple input buffer example



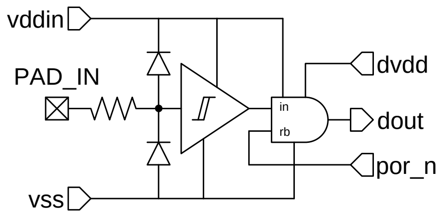
Simple input buffer example

- Specification model



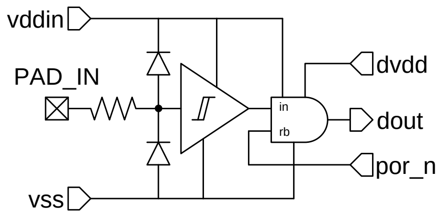
Simple input buffer example

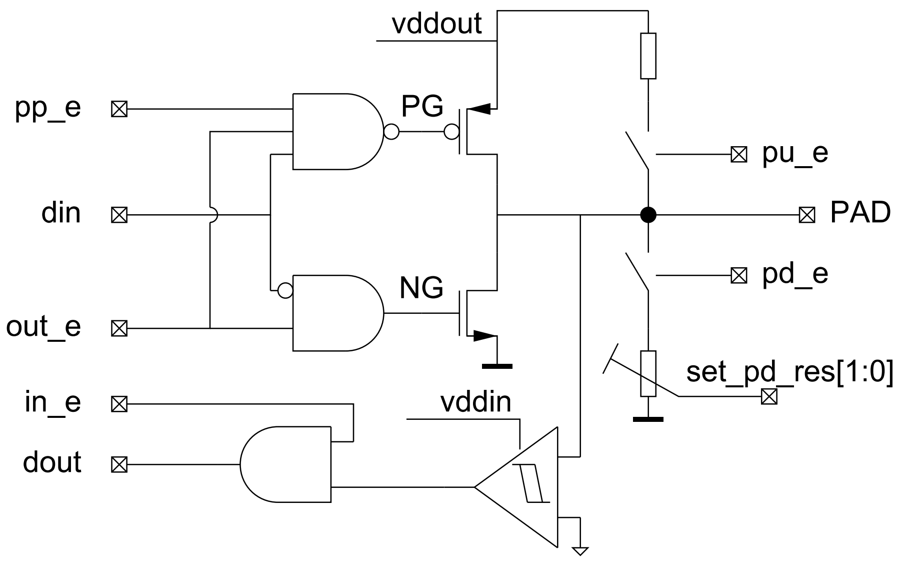
- Specification model
  - `assign dout= por_n && PAD_IN;`
  - Verify system level interactions



Simple input buffer example

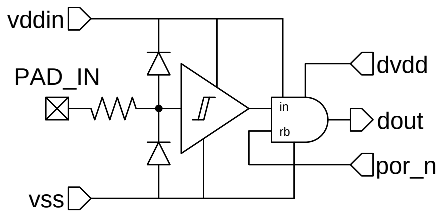
- Specification model
  - `assign dout= por_n && PAD_IN;`
  - Verify system level interactions
- Design by re-using existing cell

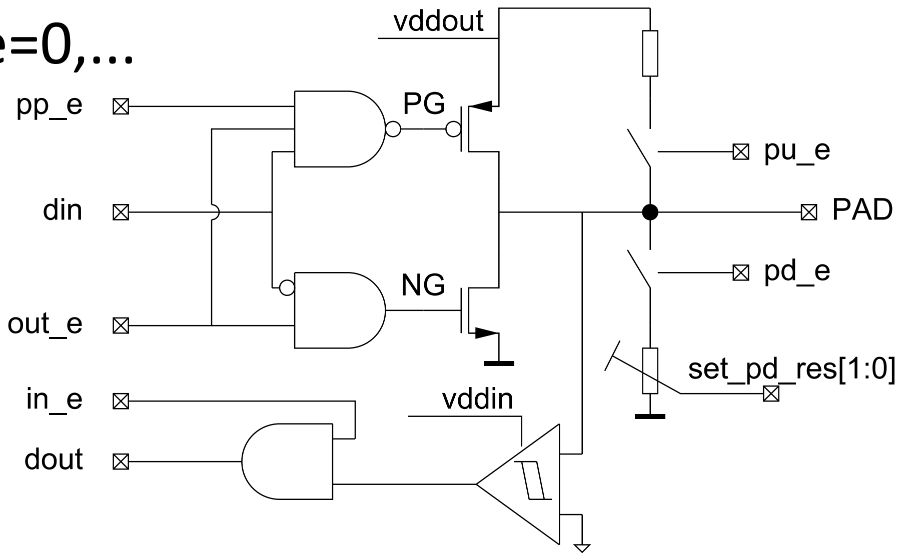




Simple input buffer example

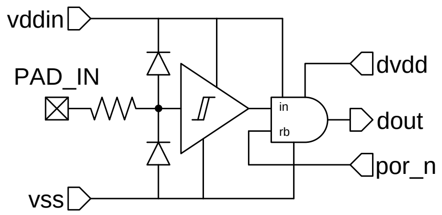
- Specification model
  - `assign dout= por_n && PAD_IN;`
  - Verify system level interactions
- Design by re-using existing cell
  - Connect: in_e=por_n, out_e=0,...
  - Verify performance

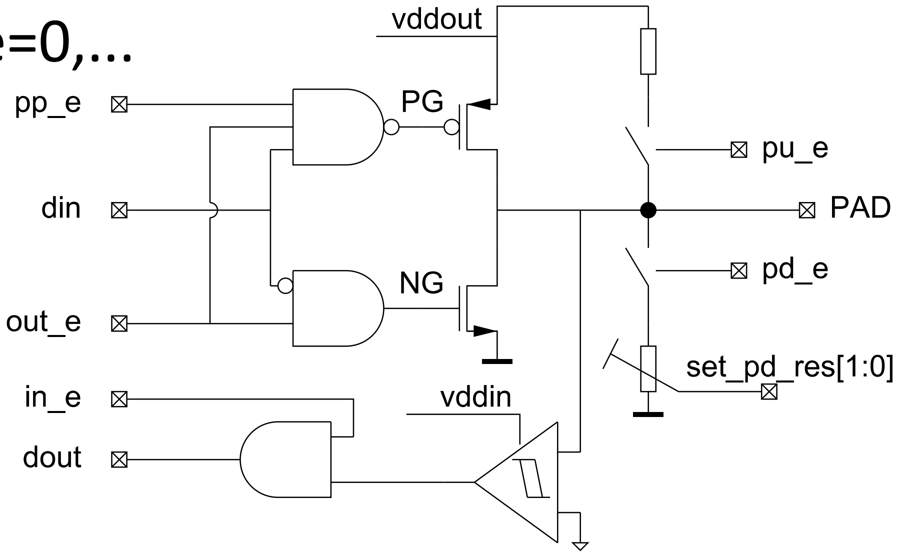




Simple input buffer example

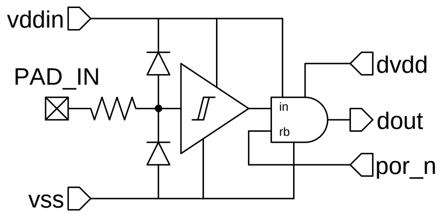
- Specification model
  - `assign dout= por_n && PAD_IN;`
  - Verify system level interactions
- Design by re-using existing cell
  - Connect: in_e=por_n, out_e=0,...
  - Verify performance
- Implementation model

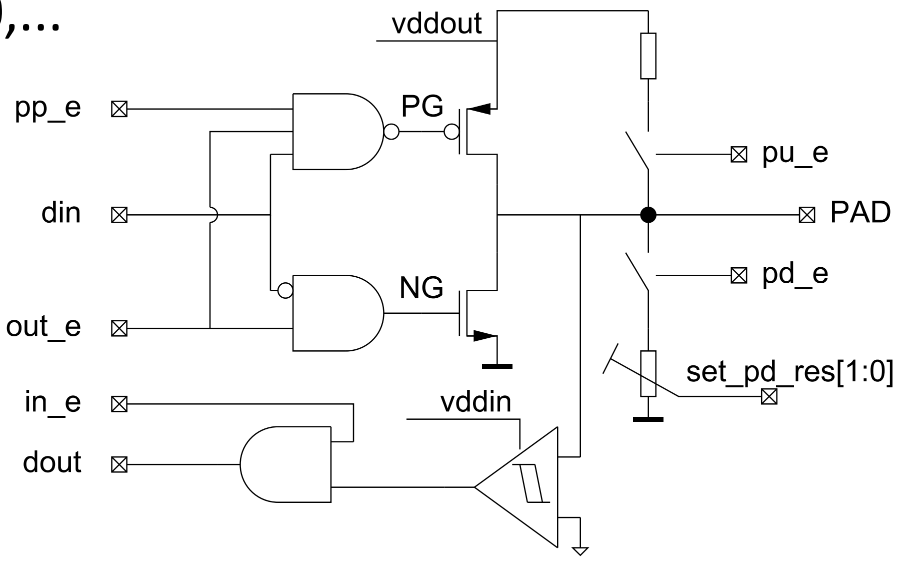




Simple input buffer example

- Specification model
  - `assign dout= por_n && PAD_IN;`
  - Verify system level interactions
- Design by re-using existing cell
  - Connect: in_e=por_n, out_e=0,...
  - Verify performance
- Implementation model
  - Netlist connectivity/models
  - Verify integration including supply availability





Digital model of an analog cell

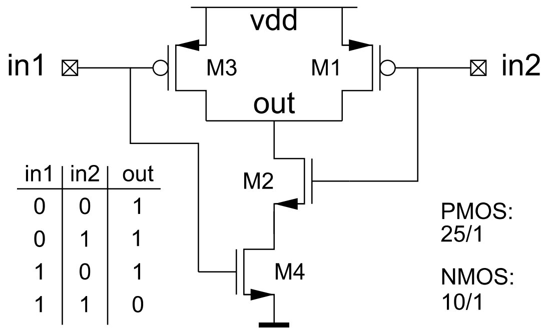
- Specification model → structure independent

- Implementation model → needs structure

Digital model of an analog cell

- Specification model → structure independent

```
module NAND2 (
 input gnd, //# TYPE=G
 input vdd, //# TYPE=P, HI=2.5
 input in1,
 input in2,
 output out //# TYPE=D, OK=DUT.supply_ok
 );
 assign supply_ok = gnd===1'b0 && vdd===1'b1;
 assign out = (supply_ok) ? !(in1 && in2)
                                     : 1'bx;
endmodule
```



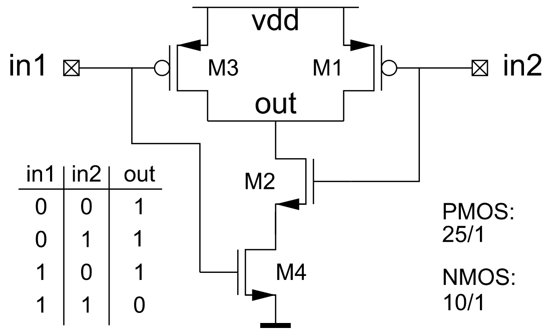
in1	in2	out
0	0	1
0	1	1
1	0	1
1	1	0

- Implementation model → needs structure

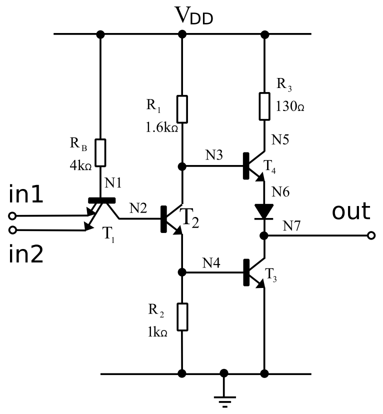
Digital model of an analog cell

- Specification model → structure independent

```
module NAND2 (
 input gnd, //# TYPE=G
 input vdd, //# TYPE=P, HI=2.5
 input in1,
 input in2,
 output out //# TYPE=D, OK=DUT.supply_ok
 );
 assign supply_ok = gnd===1'b0 && vdd===1'b1;
 assign out = (supply_ok) ? !(in1 && in2)
                          : 1'bx;
endmodule
```



in1	in2	out
0	0	1
0	1	1
1	0	1
1	1	0

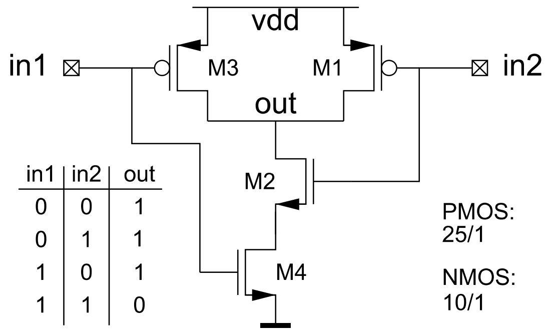


- Implementation model → needs structure

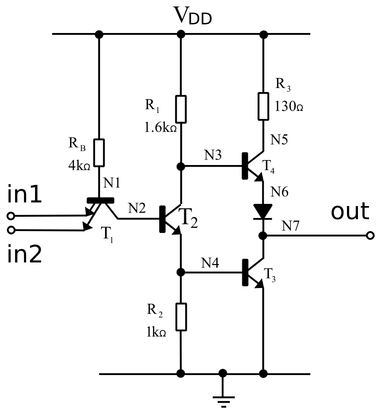
Digital model of an analog cell

- Specification model → structure independent

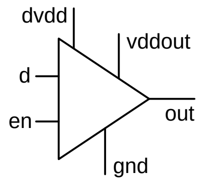
```
module NAND2 (
 input gnd, //# TYPE=G
 input vdd, //# TYPE=P, HI=2.5
 input in1,
 input in2,
 output out //# TYPE=D, OK=DUT.supply_ok
 );
 assign supply_ok = gnd===1'b0 && vdd===1'b1;
 assign out = (supply_ok) ? !(in1 && in2)
                          : 1'bx;
endmodule
```

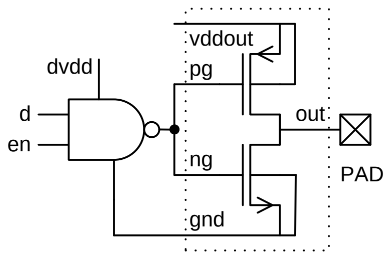


in1	in2	out
0	0	1
0	1	1
1	0	1
1	1	0



- Implementation model → needs structure



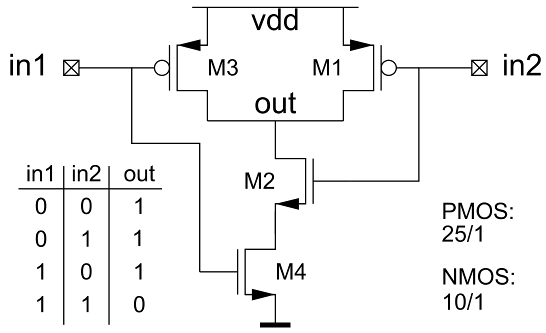


```
module cmos_out (
 input gnd, //# TYPE=G
 input vddout, //# TYPE=P, HI=1.8
 input pg,
 input ng,
 output out //# TYPE=D, OK=DUT.out_ok
 );
 assign supply_ok = gnd===1'b0 && vddout===1'b1;
 assign out = (supply_ok) ? 1'bz : 1'bx;
 assign out_ok = supply_ok && (ng ^ pg)===1'b0;
 pmos P0 (out, vddout, pg);
 nmos N0 (out, gnd, ng);
endmodule
```

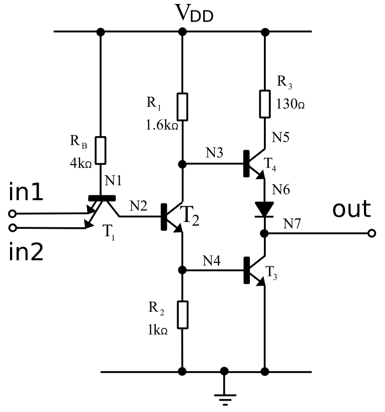
Digital model of an analog cell

- Specification model → structure independent

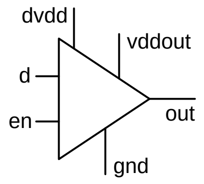
```
module NAND2 (
 input gnd, //# TYPE=G
 input vdd, //# TYPE=P, HI=2.5
 input in1,
 input in2,
 output out //# TYPE=D, OK=DUT.supply_ok
 );
 assign supply_ok = gnd===1'b0 && vdd===1'b1;
 assign out = (supply_ok) ? !(in1 && in2)
                          : 1'bx;
endmodule
```

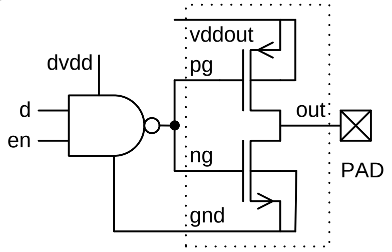


in1	in2	out
0	0	1
0	1	1
1	0	1
1	1	0



- Implementation model → needs structure
  - No spec for vddout up and down
  - Structure → well-defined out=0 for en=0, dvdd > vddout= 0→1→0



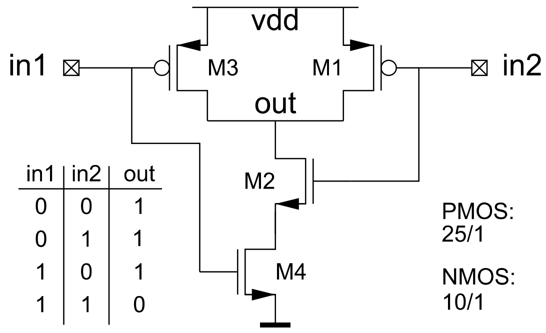


```
module cmos_out (
 input gnd, //# TYPE=G
 input vddout, //# TYPE=P, HI=1.8
 input pg,
 input ng,
 output out //# TYPE=D, OK=DUT.out_ok
 );
 assign supply_ok = gnd===1'b0 && vddout===1'b1;
 assign out = (supply_ok) ? 1'bz : 1'bx;
 assign out_ok = supply_ok && (ng ^ pg)===1'b0;
 pmos P0 (out, vddout, pg);
 nmos N0 (out, gnd, ng);
endmodule
```

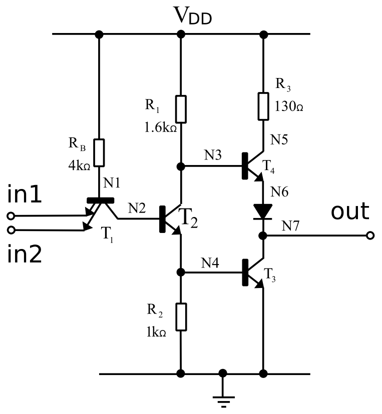
Digital model of an analog cell

- Specification model → structure independent

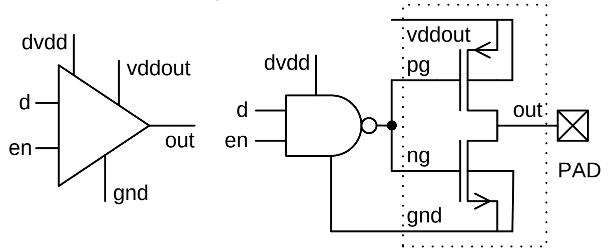
```
module NAND2 (
 input gnd, //# TYPE=G
 input vdd, //# TYPE=P, HI=2.5
 input in1,
 input in2,
 output out //# TYPE=D, OK=DUT.supply_ok
 );
 assign supply_ok = gnd===1'b0 && vdd===1'b1;
 assign out = (supply_ok) ? !(in1 && in2)
                          : 1'bx;
endmodule
```



in1	in2	out
0	0	1
0	1	1
1	0	1
1	1	0



- Implementation model → needs structure
  - No spec for vddout up and down
  - Structure → well-defined out=0 for en=0, dvdd > vddout= 0→1→0

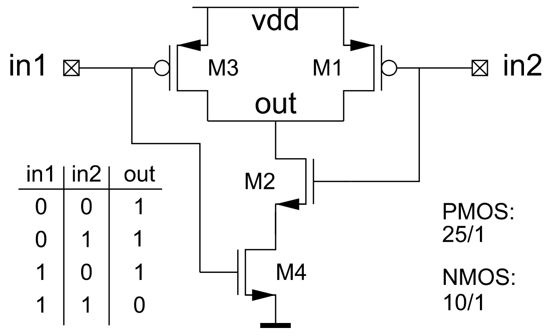


```
module cmos_out (
 input gnd, //# TYPE=G
 input vddout, //# TYPE=P, HI=1.8
 input pg,
 input ng,
 output out //# TYPE=D, OK=DUT.out_ok
 );
 assign supply_ok = gnd===1'b0 &&(vddout===1'b1;
 assign out = (supply_ok) ? 1'bz : 1'bx;      || ng && pg)
 assign out_ok = supply_ok && (ng ^ pg)===1'b0;
 pmos P0 (out, vddout, pg);
 nmos N0 (out, gnd, ng);
endmodule
```

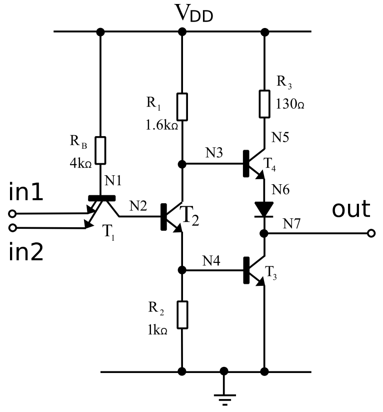
Digital model of an analog cell

- Specification model → structure independent

```
module NAND2 (
 input gnd, //# TYPE=G
 input vdd, //# TYPE=P, HI=2.5
 input in1,
 input in2,
 output out //# TYPE=D, OK=DUT.supply_ok
 );
 assign supply_ok = gnd===1'b0 && vdd===1'b1;
 assign out = (supply_ok) ? !(in1 && in2)
                          : 1'bx;
endmodule
```

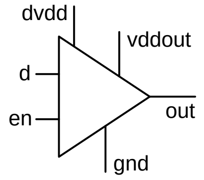


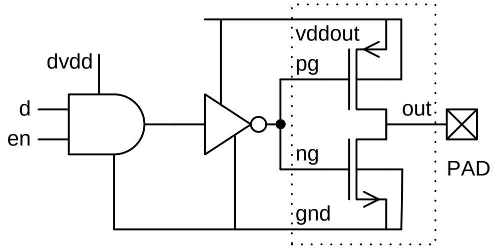
in1	in2	out
0	0	1
0	1	1
1	0	1
1	1	0



- Implementation model → needs structure
  - No spec for vddout up and down
  - Structure → ng=pg=x →out=x

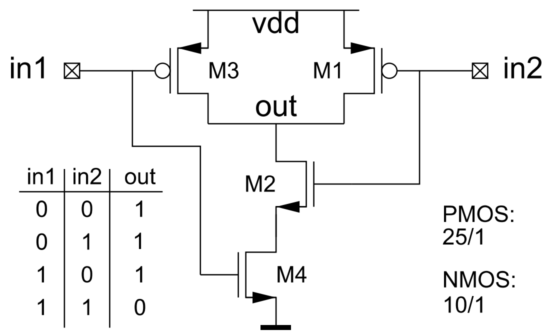
    for en=0, dvdd > vddout= 0→1→0





```
module cmos_out (
 input gnd, //# TYPE=G
 input vddout, //# TYPE=P, HI=1.8
 input pg,
 input ng,
 output out //# TYPE=D, OK=DUT.out_ok
 );
 assign supply_ok = gnd===1'b0 &&(vddout===1'b1;
 assign out = (supply_ok) ? 1'bz : 1'bx;     || ng && pg)
 assign out_ok = supply_ok && (ng ^ pg)===1'b0;
 pmos P0 (out, vddout, pg);
 nmos N0 (out, gnd, ng);
endmodule
```

Functionality modeling and validation: The Importance of Being Earnest

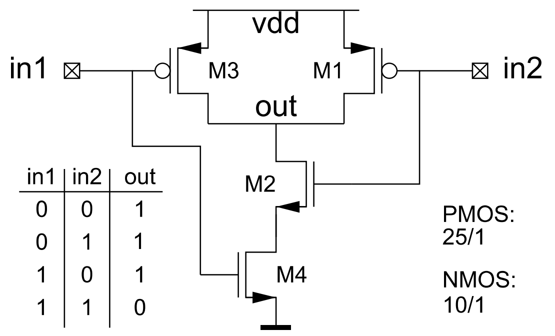


in1	in2	out
0	0	1
0	1	1
1	0	1
1	1	0

```
module NAND2 (
 input gnd, //# TYPE=G
 input vdd, //# TYPE=P, HI=2.5
 input in1,
 input in2,
 output out //# TYPE=D, OK=DUT.supply_ok
 );
 assign supply_ok =
          gnd===1'b0 || (in1===1'b0||in2===1'b0)
       && vdd===1'b1 || ({in1,in2}===2'b11);
 assign out = (supply_ok) ? !(in1 && in2)
                                     : 1'bx;
endmodule
```

Functionality modeling and validation: The Importance of Being Earnest

- Validation: stimulate inputs and compare response vs. expectation and transistor implementation

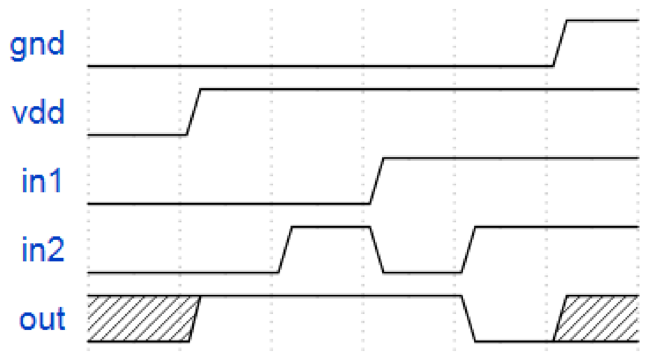


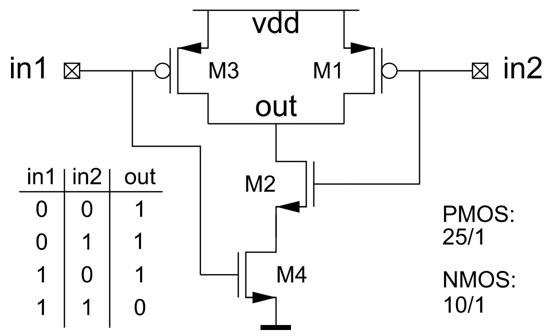
in1	in2	out
0	0	1
0	1	1
1	0	1
1	1	0

```
module NAND2 (
 input gnd, //# TYPE=G
 input vdd, //# TYPE=P, HI=2.5
 input in1,
 input in2,
 output out //# TYPE=D, OK=DUT.supply_ok
 );
 assign supply_ok =
        gnd===1'b0 || (in1===1'b0||in2===1'b0)
     && vdd===1'b1 || ({in1,in2}===2'b11);
 assign out = (supply_ok) ? !(in1 && in2)
                          : 1'bx;
endmodule
```

Functionality modeling and validation: The Importance of Being Earnest

- Validation: stimulate inputs and compare response vs. expectation and transistor implementation



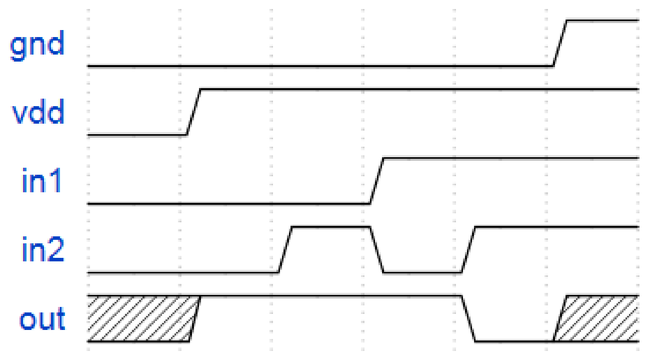


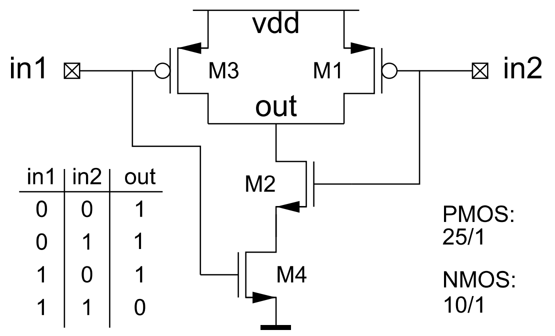
in1	in2	out
0	0	1
0	1	1
1	0	1
1	1	0

```
module NAND2 (
 input gnd, //# TYPE=G
 input vdd, //# TYPE=P, HI=2.5
 input in1,
 input in2,
 output out //# TYPE=D, OK=DUT.supply_ok
 );
 assign supply_ok =
           gnd===1'b0 || (in1===1'b0||in2===1'b0)
        && vdd===1'b1 || ({in1,in2}===2'b11);
 assign out = (supply_ok) ? !(in1 && in2)
                          : 1'bx;
endmodule
```

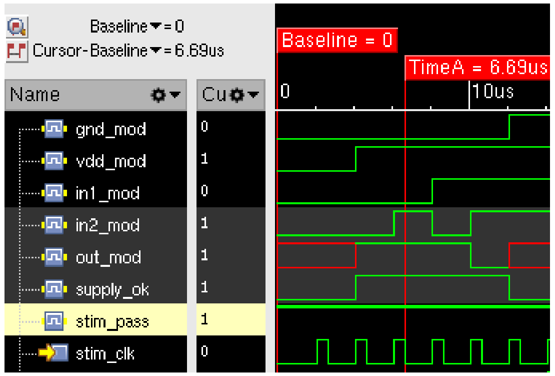
Functionality modeling and validation: The Importance of Being Earnest

- Validation: stimulate inputs and compare response vs. expectation and transistor implementation
- Describe expectation with Wavedrom stim_pass=1





in1	in2	out
0	0	1
0	1	1
1	0	1
1	1	0



Baseline = 0
Cursor-Baseline = 6.69us

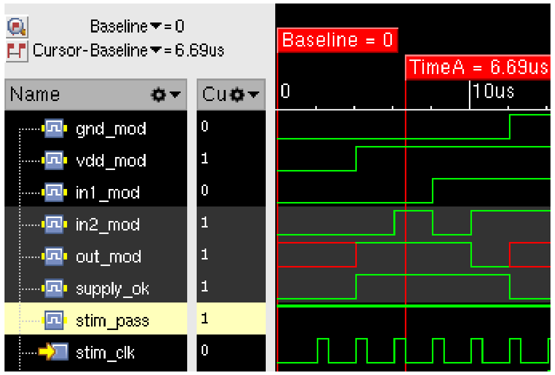
Baseline = 0
TimeA = 6.69us

Name	Cu
gnd_mod	0
vdd_mod	1
in1_mod	0
in2_mod	1
out_mod	1
supply_ok	1
stim_pass	1
stim_clk	0

```
module NAND2 (
 input gnd, //# TYPE=G
 input vdd, //# TYPE=P, HI=2.5
 input in1,
 input in2,
 output out //# TYPE=D, OK=DUT.supply_ok
 );
 assign supply_ok =
          gnd===1'b0 || (in1===1'b0||in2===1'b0)
       && vdd===1'b1 || ({in1,in2}===2'b11);
 assign out = (supply_ok) ? !(in1 && in2)
                          : 1'bx;
endmodule
```

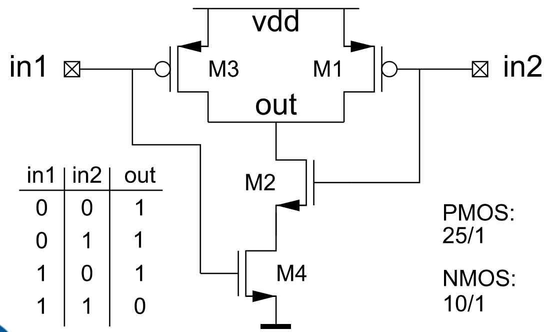
Functionality modeling and validation: Expectations beyond the obvious

- Validation: stimulate inputs and compare response
- Describe expectation with Wavedrom →stim_pass=1



Baseline = 0
Cursor-Baseline = 6.69us

Name	Cu
gnd_mod	0
vdd_mod	1
in1_mod	0
in2_mod	1
out_mod	1
supply_ok	1
stim_pass	1
stim_clk	0

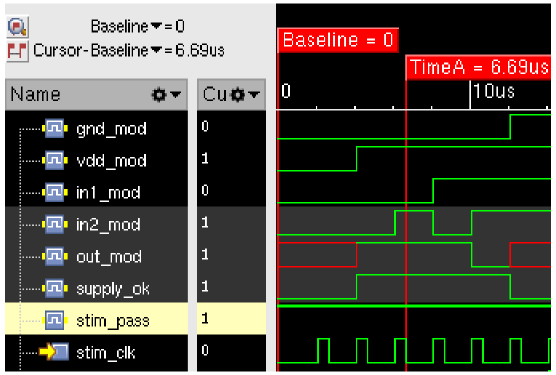


in1	in2	out
0	0	1
0	1	1
1	0	1
1	1	0

```
module NAND2 (
 input gnd, //# TYPE=G
 input vdd, //# TYPE=P, HI=2.5
 input in1,
 input in2,
 output out //# TYPE=D, OK=DUT.supply_ok
 );
 assign supply_ok =
        gnd===1'b0 || (in1===1'b0||in2===1'b0)
     && vdd===1'b1 || ({in1,in2}===2'b11);
 assign out = (supply_ok) ? !(in1 && in2)
                          : 1'bx;
endmodule
```

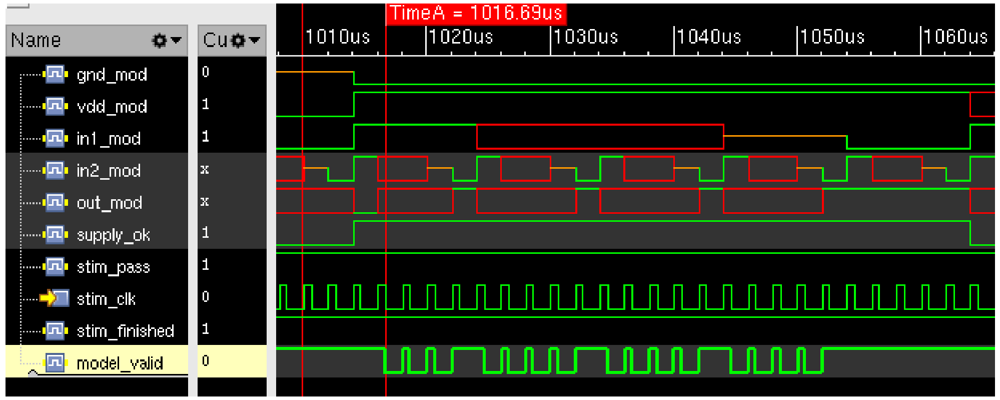
Functionality modeling and validation: Expectations beyond the obvious

- Validation: stimulate inputs and compare response
- Describe expectation with Wavedrom →stim_pass=1
- Apply all stimulus combinations →model_valid=1



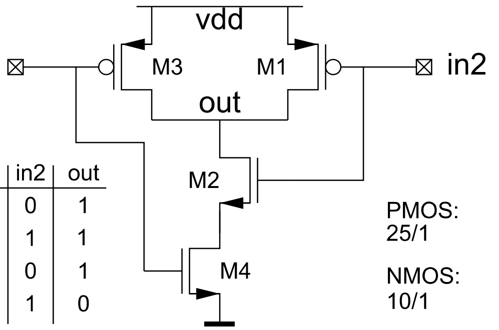
```
Baseline = 0
Cursor-Baseline = 6.69us
Baseline = 0
TimeA = 6.69us
```

Name	Cu
gnd_mod	0
vdd_mod	1
in1_mod	0
in2_mod	1
out_mod	1
supply_ok	1
stim_pass	1
stim_clk	0



```
TimeA = 1016.69us
1010us 1020us 1030us 1040us 1050us 1060us
```

Name	Cu
gnd_mod	0
vdd_mod	1
in1_mod	1
in2_mod	x
out_mod	x
supply_ok	1
stim_pass	1
stim_clk	0
stim_finished	1
model_valid	0

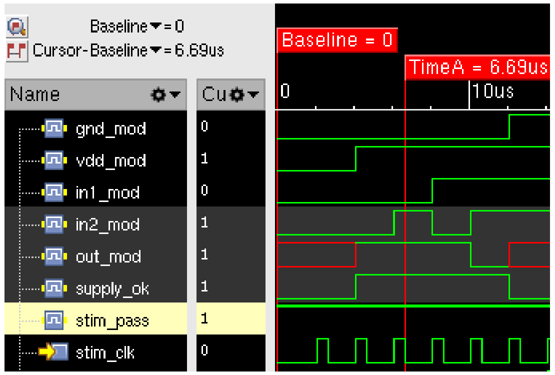


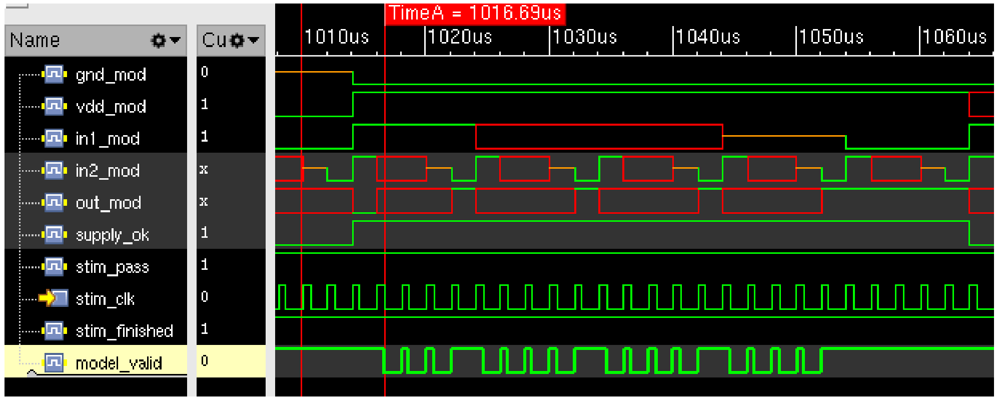
in1	in2	out
0	0	1
0	1	1
1	0	1
1	1	0

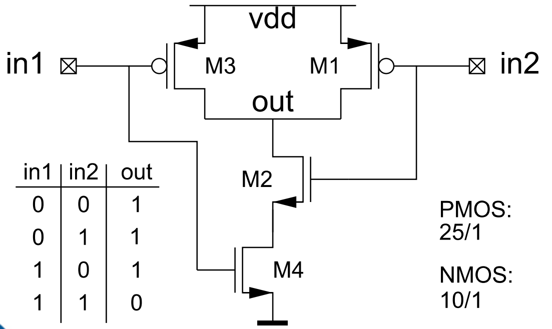
```
input in1,
input in2,
output out //# TYPE=D, OK=DUT.supply_ok
);
assign supply_ok =
       gnd===1'b0 || (in1===1'b0||in2===1'b0)
    && vdd===1'b1 || ({in1,in2}===2'b11);
assign out = (supply_ok) ? !(in1 && in2)
                         : 1'bx;
endmodule
```

Functionality modeling and validation: Expectations beyond the obvious

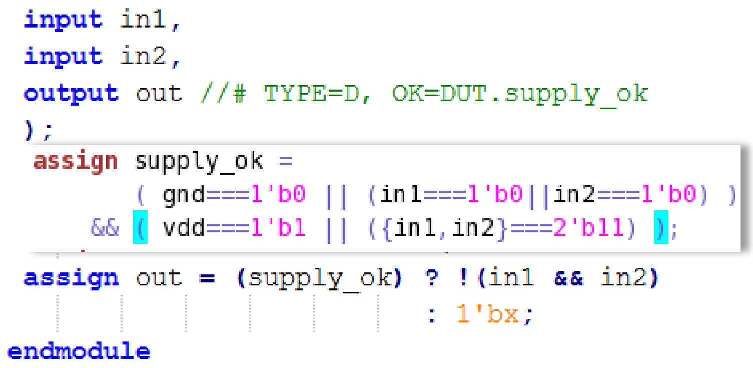
- Validation: stimulate inputs and compare response
- Describe expectation with Wavedrom →stim_pass=1
- Apply all stimulus combinations →model_valid=1 →Fix model!







in1	in2	out
0	0	1
0	1	1
1	0	1
1	1	0



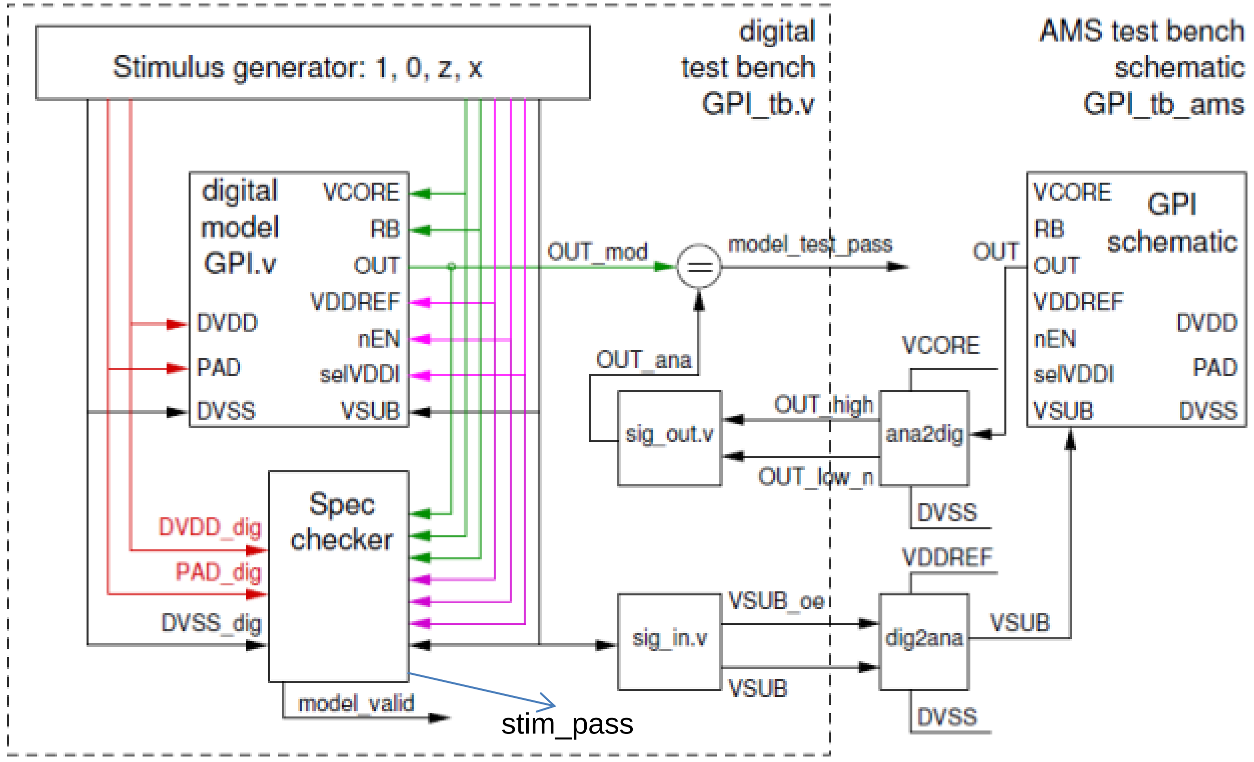
```
input in1,
input in2,
output out //# TYPE=D, OK=DUT.supply_ok
);
 assign supply_ok =
        ( gnd===1'b0 || (in1===1'b0||in2===1'b0) )
     && ( vdd===1'b1 || ({in1,in2}===2'b11) );

assign out = (supply_ok) ? !(in1 && in2)
                         : 1'bx;

endmodule
```

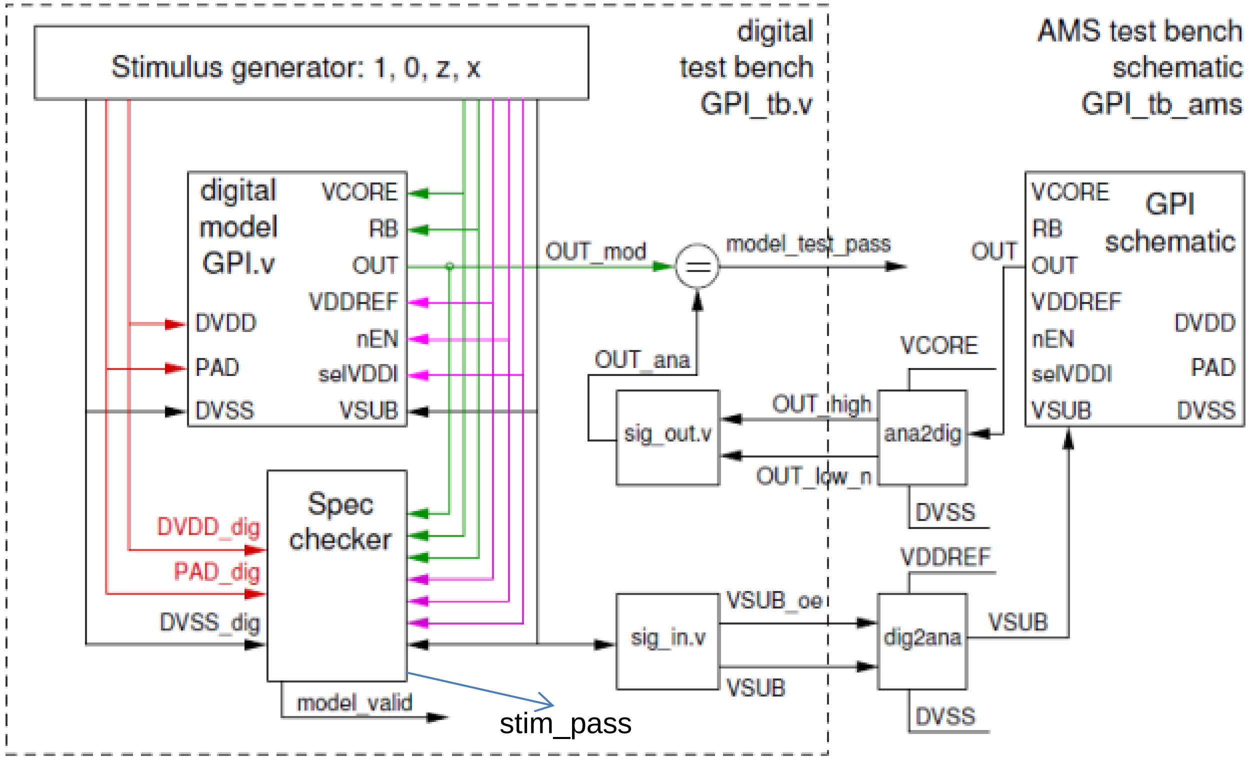
Functionality modeling and validation: Expect to match schematic

- Validation: stimulate inputs and compare response
- Sign-off: stim_pass, model_valid, model_test_pass=1



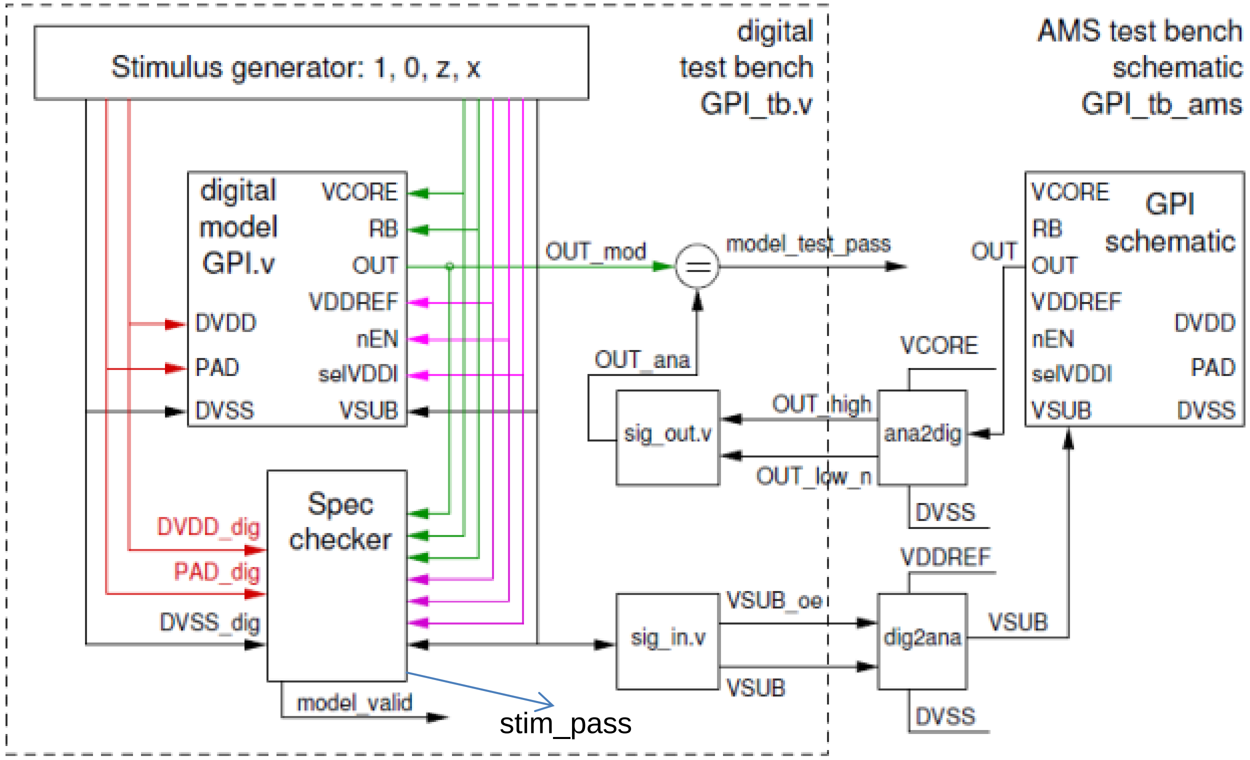
Functionality modeling and validation: Expect to match schematic

- Validation: stimulate inputs and compare response
- Sign-off: stim_pass, model_valid, model_test_pass=1
- Script-based AMS test bench generation



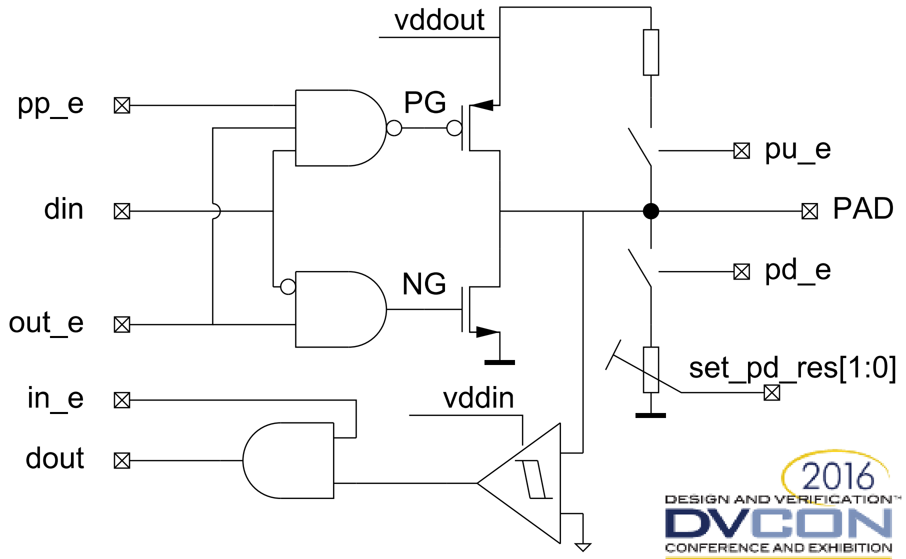
Functionality modeling and validation: Expect to match schematic

- Validation: stimulate inputs and compare response
- Sign-off: stim_pass, model_valid, model_test_pass=1
- Script-based AMS test bench generation
- Regression over all models



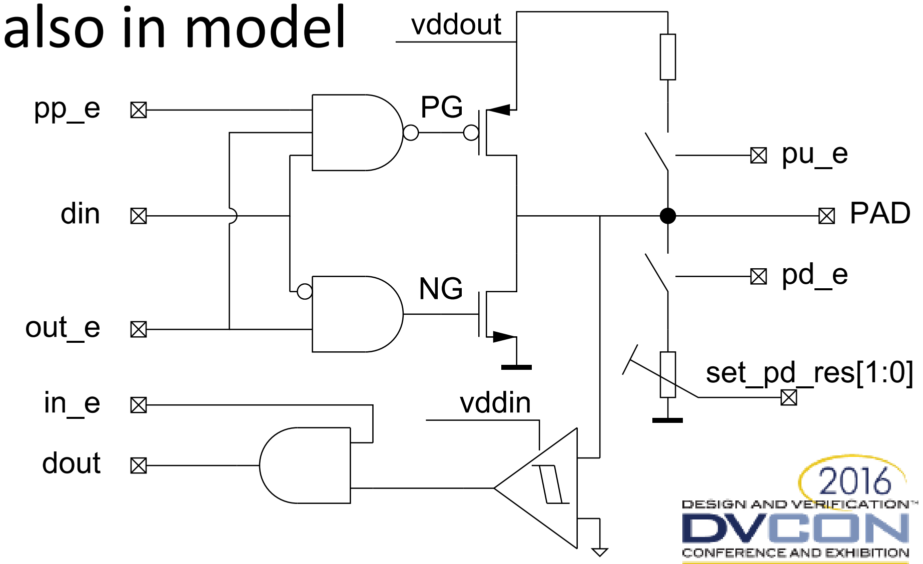
Structure preserving model assembly

- Assume models for low-level analog structures



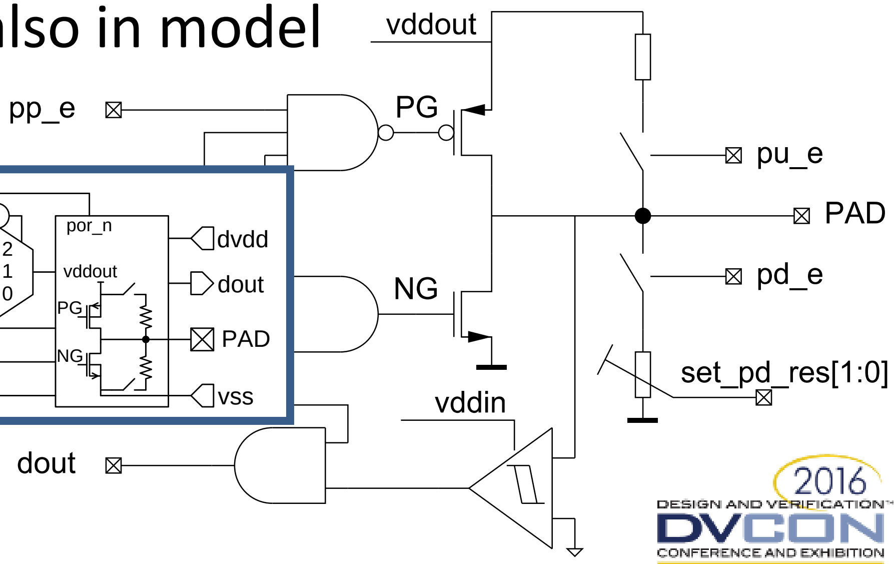
Structure preserving model assembly

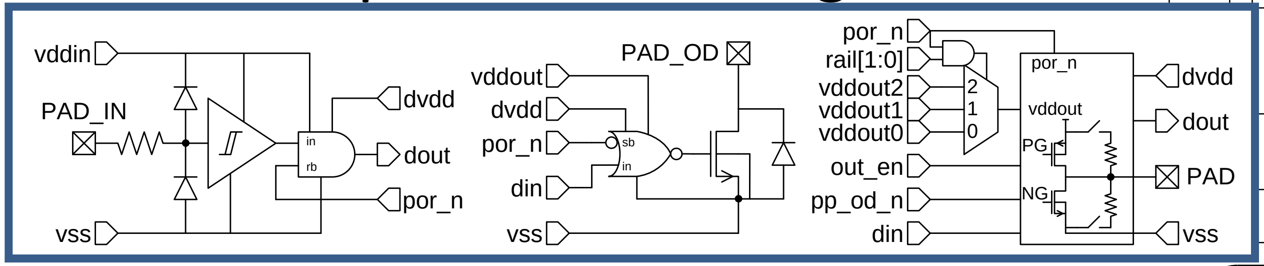
- Assume models for low-level analog structures
- Use netlister for model assembly
  - Accommodate late design changes
  - Re-use sub-blocks and validated models
  - Implementation model valid by construction
  - Structural design bugs are also in model



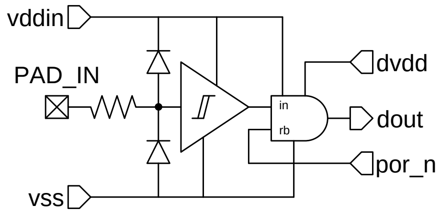
Structure preserving model assembly

- Assume models for low-level analog structures
- Use netlister for model assembly
  - Accommodate late design changes
  - Re-use sub-blocks and validated models
  - Implementation model valid by construction
  - Structural design bugs are also in model
- Example: IO cell designs



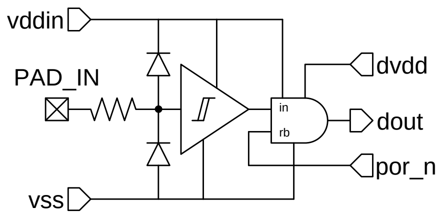


Case study: Simple input buffer



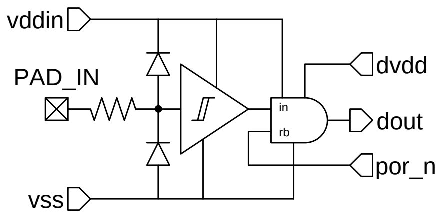
Case study: Simple input buffer

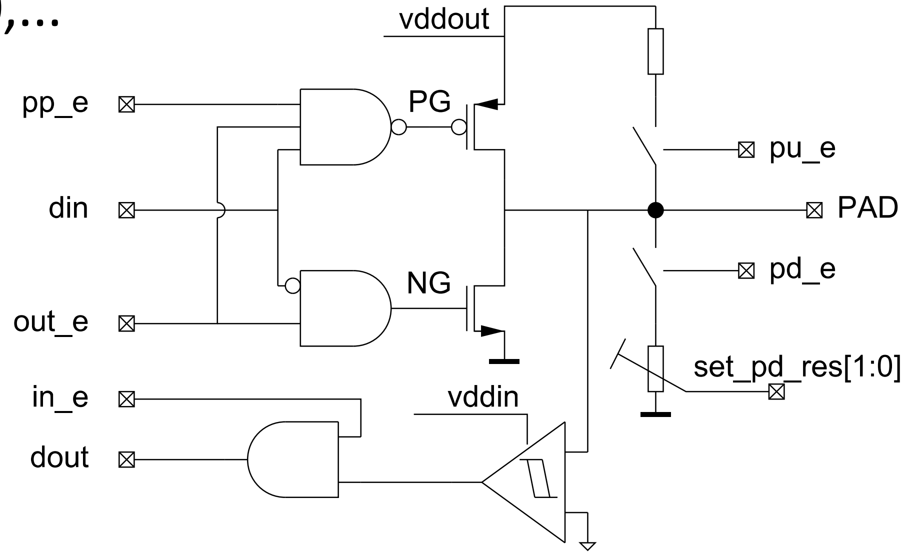
- Specification model
  - `assign dout= por_n && PAD_IN;`
  - Verify system level interactions



Case study: Simple input buffer

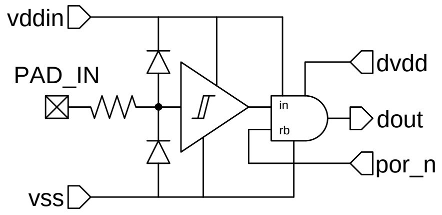
- Specification model
  - `assign dout= por_n && PAD_IN;`
  - Verify system level interactions
- Design by re-using existing cell
  - Connect: in_e=por_n, pu_e=0,...

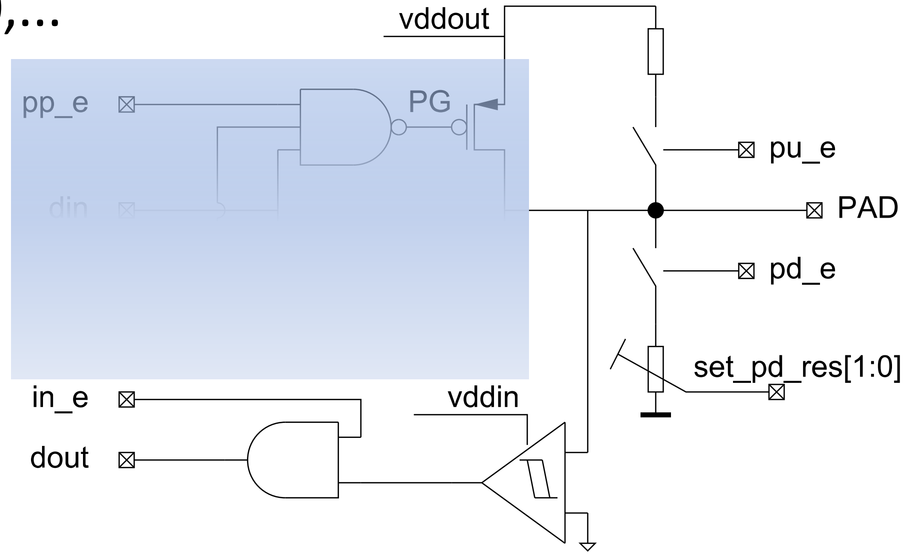




Case study: Simple input buffer

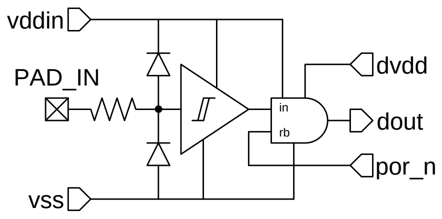
- Specification model
  - `assign dout= por_n && PAD_IN;`
  - Verify system level interactions
- Design by re-using existing cell
  - Connect: in_e=por_n, pu_e=0,...
  - Remove output stage

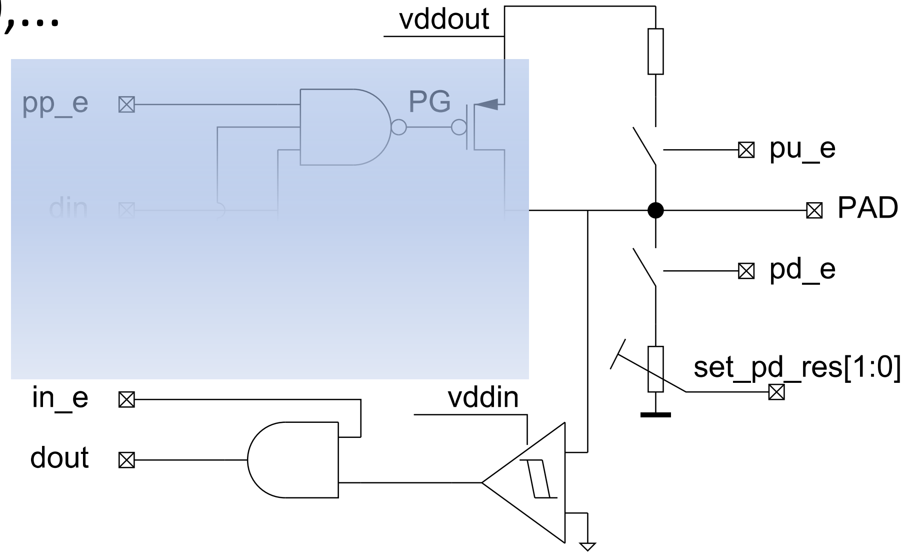




Case study: Simple input buffer

- Specification model
  - `assign dout= por_n && PAD_IN;`
  - Verify system level interactions
- Design by re-using existing cell
  - Connect: in_e=por_n, pu_e=0,...
  - Remove output stage
- Implementation model
  - Netlist connectivity/models
  - Verify integration including supply availability

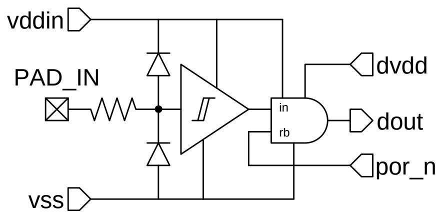


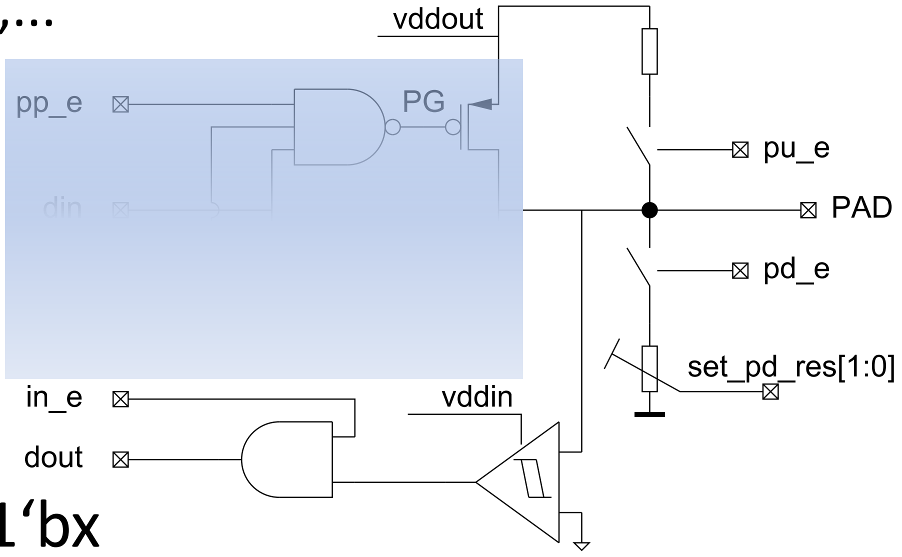


Case study: Simple input buffer

- Specification model
  - `assign dout= por_n && PAD_IN;`
  - Verify system level interactions
- Design by re-using existing cell
  - Connect: in_e=por_n, pu_e=0,...
  - Remove output stage
- Implementation model
  - Netlist connectivity/models
  - Verify integration including supply availability

Validation exhibits PAD_IN=1‘bx

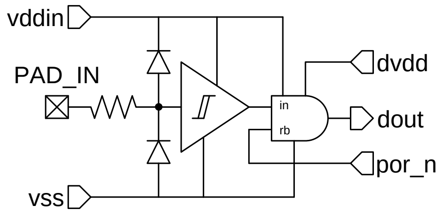




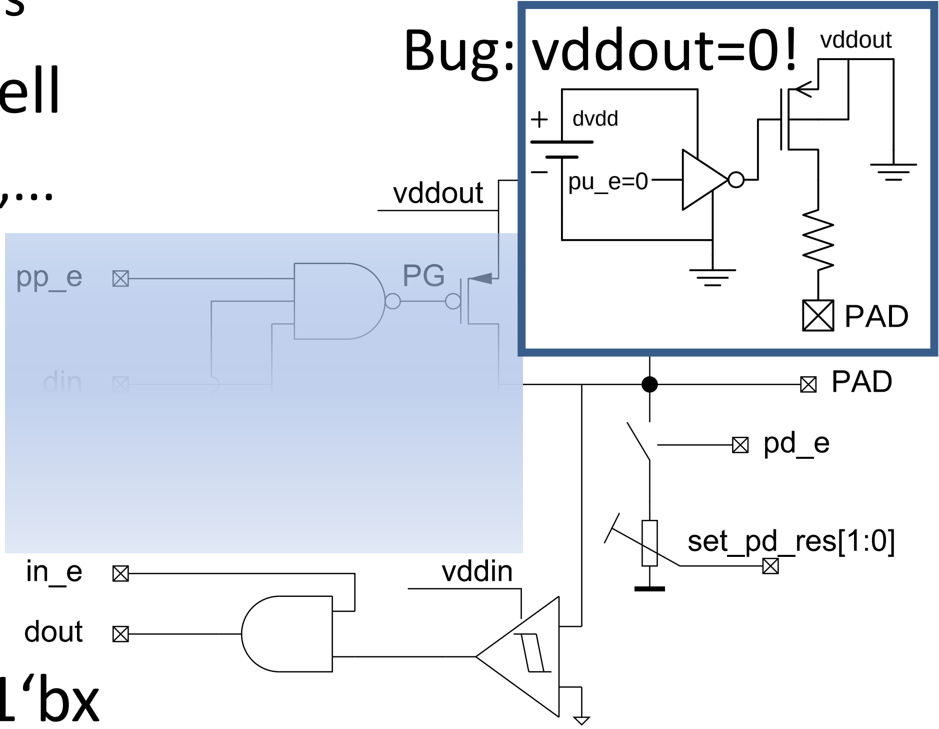
Case study: Simple input buffer

- Specification model
  - `assign dout= por_n && PAD_IN;`
  - Verify system level interactions
- Design by re-using existing cell
  - Connect: in_e=por_n, pu_e=0,...
  - Remove output stage
- Implementation model
  - Netlist connectivity/models
  - Verify integration including supply availability

Validation exhibits PAD_IN=1'bx



Bug: vddout=0!



Conclusions

- Distinguish specification and implementation models
  → different purpose, different construction

Conclusions

- Distinguish specification and implementation models
  → different purpose, different construction
- Validate models against expectation and schematics
  → need script-based test bench generation

Conclusions

- Distinguish specification and implementation models
  → different purpose, different construction
- Validate models against expectation and schematics
  → need script-based test bench generation
- For modeling and validation:
  - Keep models small (many inputs, but **one** output)
  - Signal flow from inputs to output; no feedback to input
  - Design hierarchy supports netlist-based model assembly

Conclusions

- Distinguish specification and implementation models
  → different purpose, different construction
- Validate models against expectation and schematics
  → need script-based test bench generation
- For modeling and validation:
  - Keep models small (many inputs, but **one** output)
  - Signal flow from inputs to output; no feedback to input
  - Design hierarchy supports netlist-based model assembly
- Event-driven model also finds bugs in analog design

Conclusions

- Distinguish specification and implementation models
  → different purpose, different construction
- Validate models against expectation and schematics
  → need script-based test bench generation
- For modeling and validation:
  - Keep models small (many inputs, but **one** output)
  - Signal flow from inputs to output; no feedback to input
  - Design hierarchy supports netlist-based model assembly
- Event-driven model also finds bugs in analog design
- Model is formulation of circuit understanding

Conclusions

- Distinguish specification and implementation models
  → different purpose, different construction
- Validate models against expectation and schematics
  → need script-based test bench generation
- For modeling and validation:
  - Keep models small (many inputs, but **one** output)
  - Signal flow from inputs to output; no feedback to input
  - Design hierarchy supports netlist-based model assembly
- Event-driven model also finds bugs in analog design
- Model is formulation of circuit understanding

Questions?