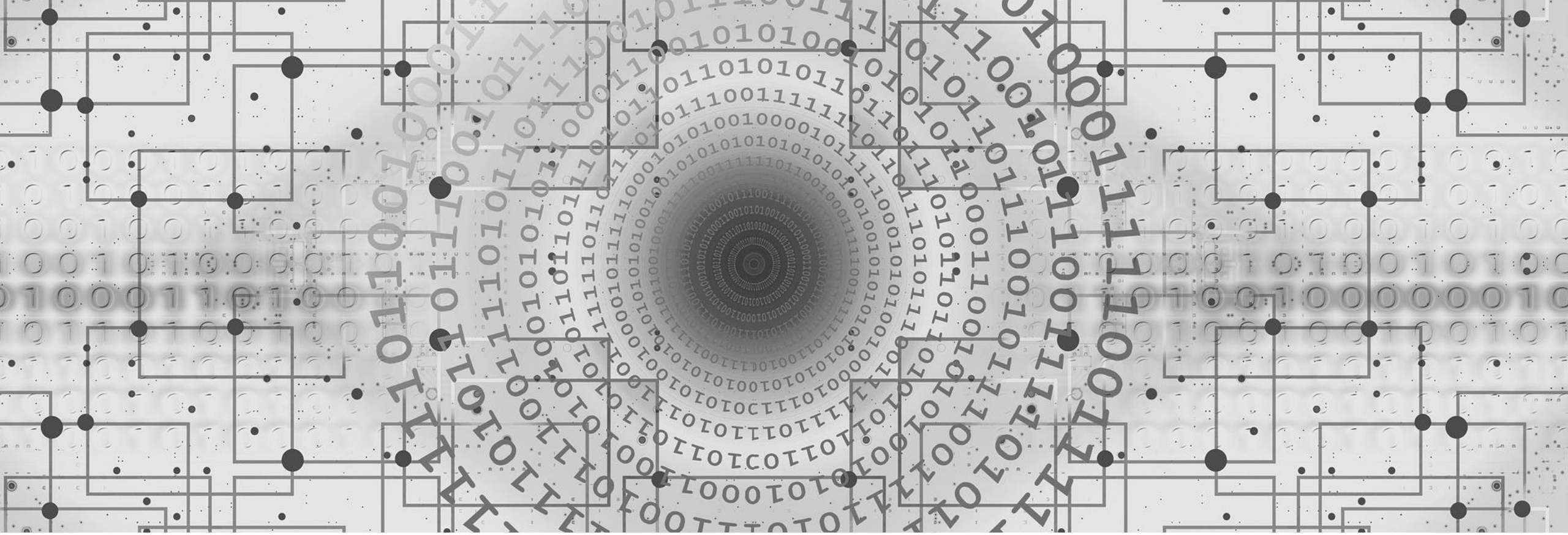


Lecture-3 : SystemVerilog Modeling Abstractions

ECE-111 Advanced Digital Design Project

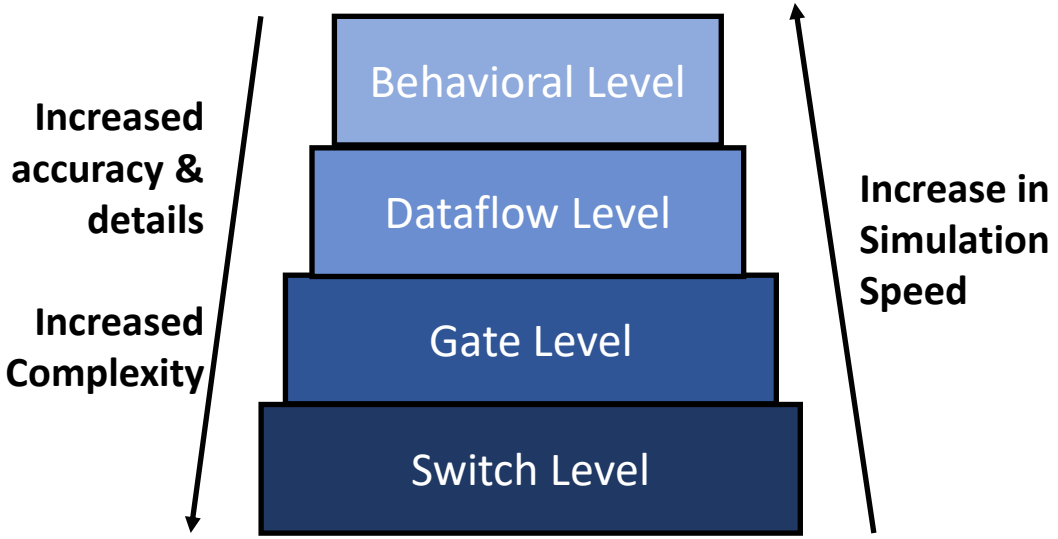
Vishal Karna

UC San Diego
JACOBS SCHOOL OF ENGINEERING
Electrical and Computer Engineering

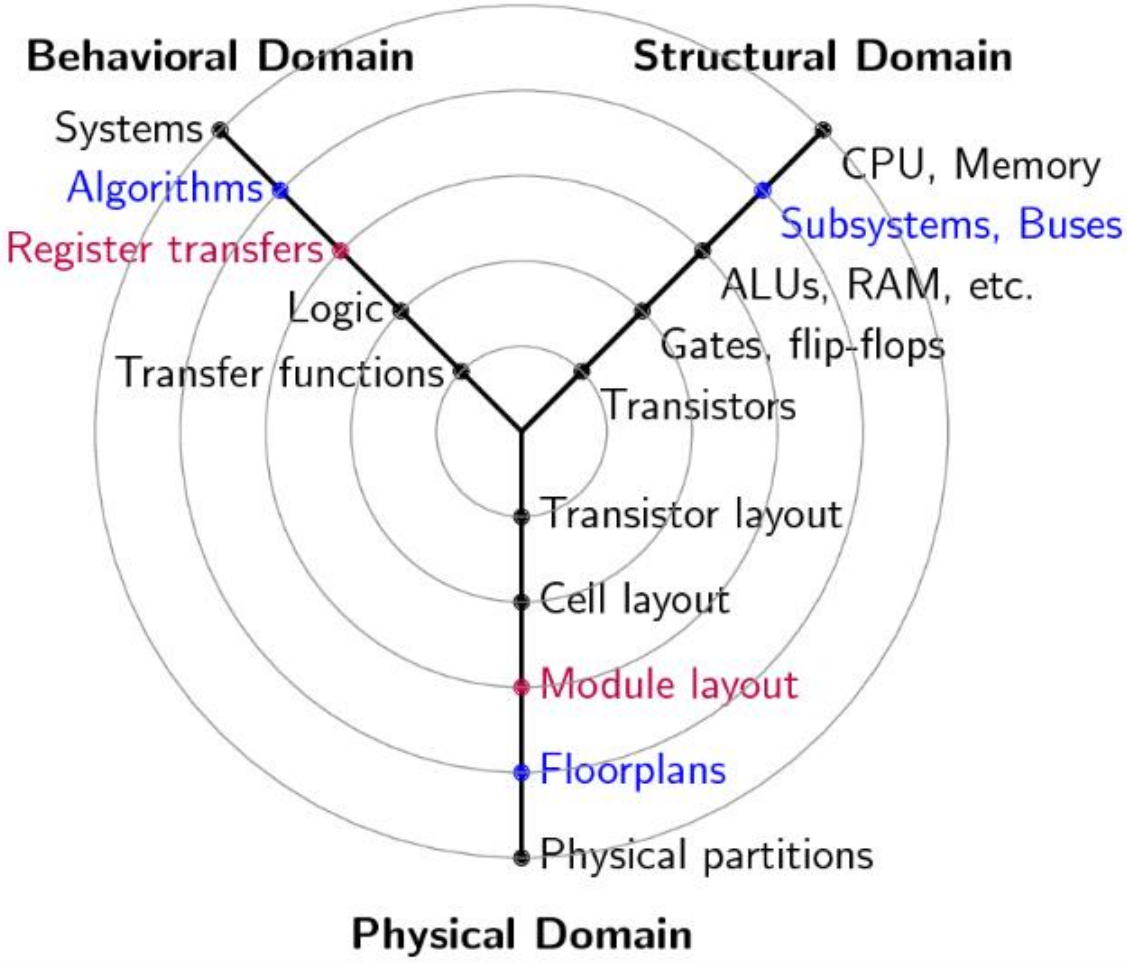
Winter 2022

Modeling Styles in SystemVerilog

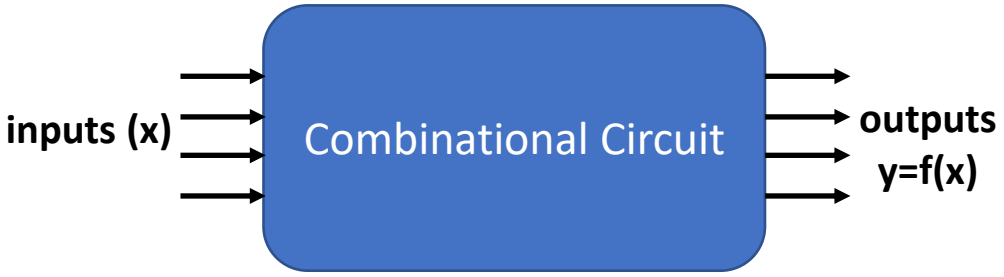
- **SystemVerilog modeling language supports three kinds of modeling styles:**
  - Behavioral level (Algorithmic level)
  - Dataflow level
  - Gate level (Structural level)
  - Switch level (Transistor level)
- **RTL (Register Transfer Level) model utilizes combination of behavioral and dataflow styles**



Design View Point Through Y-Diagram

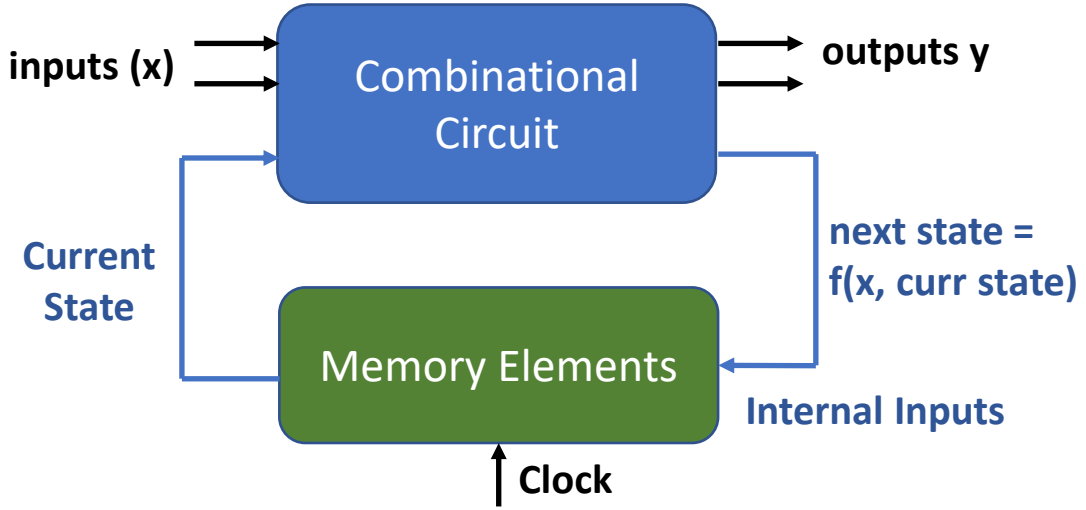


Combinational vs Sequential Circuit



Combinational Circuit

- At any instance of time present value of outputs depends solely on present value of inputs
- Does not store any intermediate values and hence does not require any memory elements
- Does not require any clock signal
- Behavior is described by set of output functions
- **Examples:** Full Adder, half adder, comparator, multiplexer, decoder, encoder, etc



Sequential Circuit

- Present values of outputs are determined from present values of inputs and past state (i.e. sequence of past inputs or known as past outputs)
- Behavior is described by set of output functions and set of next states functions stored in memory
- It contains memory elements to store past outputs and requires clock signals
- **Examples:** Flipflop, Latch, Shift Register, etc

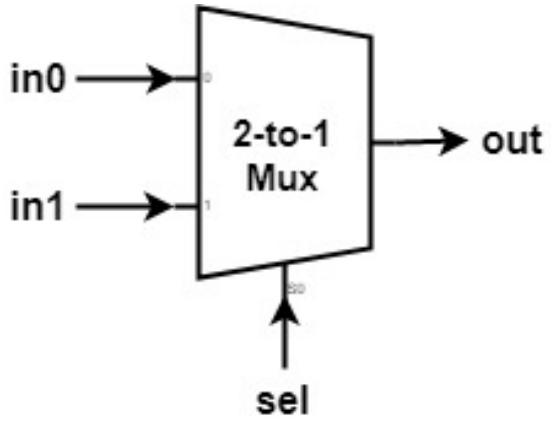
Behavioral Level Modeling

- **Hardware circuit is specified in terms of its expected behavior or an algorithm without concern of internal hardware or gate level implementation**
  - Full functionality of a complex circuit specified in C type natural language description
  - It is **not** a cycle timing accurate representation of hardware circuit
  - It may contain algorithms, Boolean equations, truth tables (Tables of input and output values)
  - Models at this abstraction level are also called as bus-functional or algorithmic models
  - This is the highest level of abstraction provided by SystemVerilog HDL
  - It is primarily used to model sequential circuits, but can also be used to model pure combinatorial circuits

- **Advantage :**
  - Behavioral models are faster to simulate compared to gate level models since it has less timing details

Behavioral level Modeling

❑ Behavioral level description of a 2-to-1 Multiplexer using **if/else conditional statement**



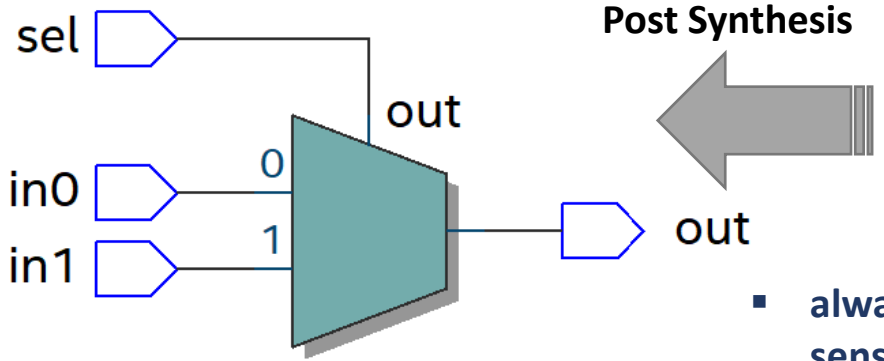

```
module mux_2x1_behavioral( // Start of module
  input logic in0, in1
  input logic sel,
  output logic out
);
 always@(sel or in0 or in1)
  begin
    if(sel == 1)
      out = in1;
    else
      out = in0;
  end
endmodule:mux_2x1_behavioral   // End of module
```

Primary port declarations with directions and data type of each port specified

Sensitivity list

Functionality of 2-to-1 Multiplexer using if/else statement within "always" procedural block

Post Synthesis

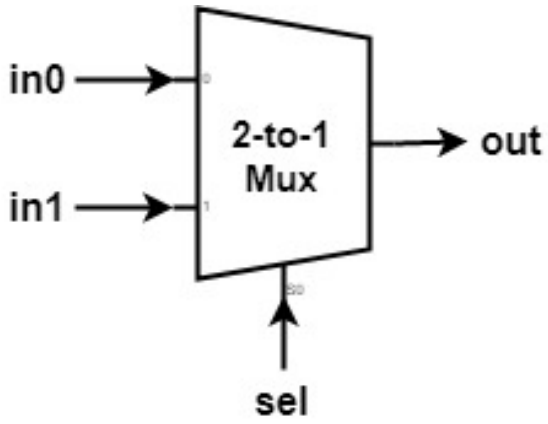


2-to-1 Multiplexer

- always block waits until either of the signal "sel" or "in0" or "in1" specified in its sensitivity list, its value changes and only then if/else statements within always block will get executed
- "always" block continuously runs and only terminates when simulation ends

Behavioral level Modeling

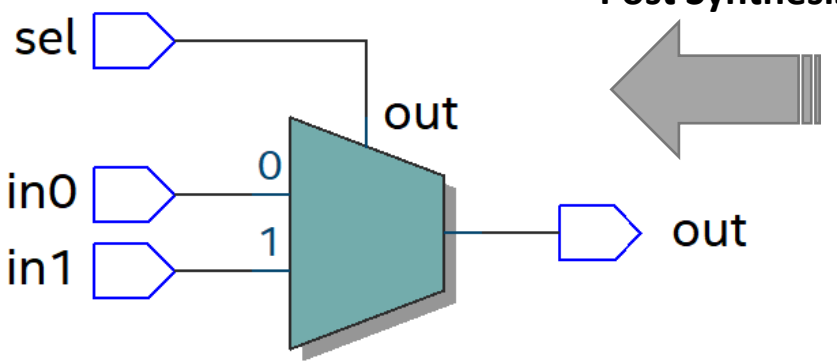
❑ Behavioral level description of a 2-to-1 Multiplexer using **case statement**




```
module mux_2x1_behavioral (
  input logic in0, in1,
  input logic sel,
  output logic out
);
  always @(sel or in0 or in1)
  begin
    case(sel)
      1'b0 : out = in0;
      1'b1 : out = in1;
      default : out = 1'bx;
    endcase
  end
endmodule
```

2-to-1 Multiplexer implementation using case statement

Post Synthesis



2-to-1 Multiplexer

- Case statement tries to match "sel" value with each case expression statements in the order it is specified. First matching case expression statement will execute.

Behavioral Level Modeling

- **Most behavioral modeling is done using two important constructs: initial and always**
  - Behavioral statements appear only inside **initial** and **always** blocks
  - A module may contain an arbitrary number of **initial** or **always** blocks and may contain one or more procedural statements within them.
  - All of the initial and always blocks execute concurrently (i.e. to model parallelism)
- **Behavioral model can utilize full set of constructs in SystemVerilog including both synthesizable and non-synthesizable constructs**
  - If behavioral model of a design is developed with intent to generate hardware from it then it should only use synthesizable constructs such as :
    - **Synthesizable : always, assign, generate, for, if/else, case, fixed size arrays, interfaces and more**
  - If behavioral model of a design is developed for architectural exploration or for verification purpose only with ***no intent*** to generate hardware from it, then it can use both non-synthesizable and synthesizable constructs such as :
    - **Non-Synthesizable : initial, class, dynamic arrays, wait, delays, fork/join, continue, break,** and more
    - **Synthesizable : always, assign, generate, for, if/else, case, fixed size arrays, interfaces** and more

Dataflow Modeling

- **In dataflow modeling, module is designed by describing how data flows through a circuit**
  - This is a higher level of abstraction then the gate level hence no hardware implementation details required
  - Data processing within design is specified using logical equations or boolean expressions.
  - Dataflow modeling style is mainly used to describe combinational circuits.
  - Dataflow models can be translated into a gate level design through process of logic synthesis

- **In dataflow modeling most of the design is implemented using continuous assignments (assign), which are used to drive a value onto a net.**
  - Continuous assignments are made using the keyword ***assign***.
  - Syntax Format : **assign net_name = expression**

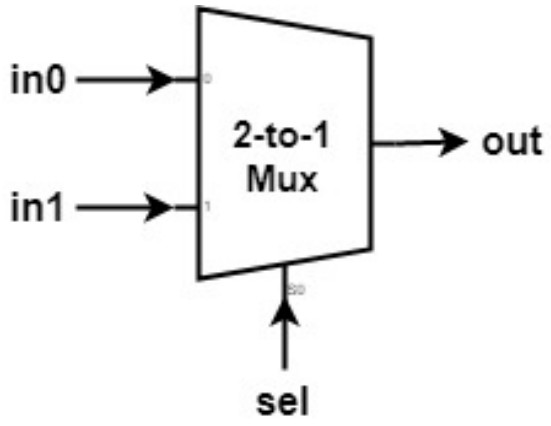
    Example :

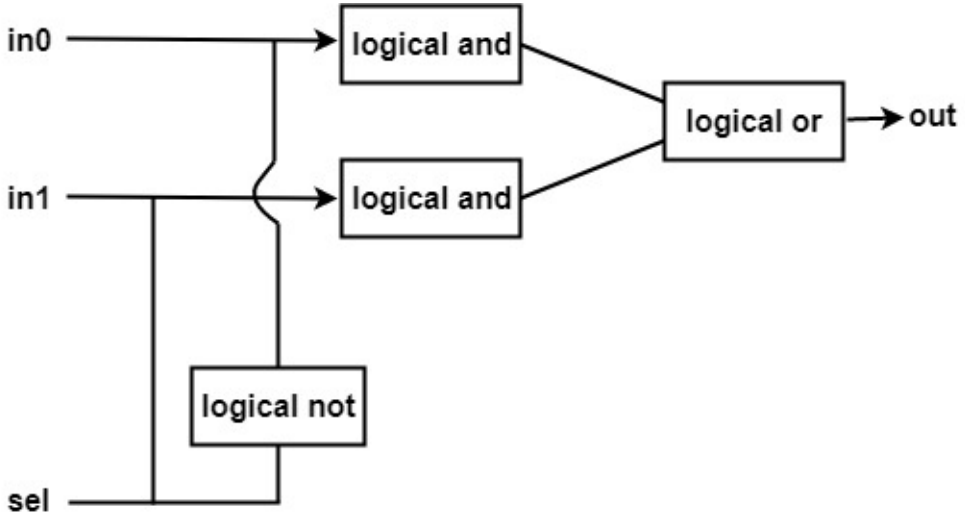
    ```
    wire A, B, C;
    assign C = A & B;   // Any change in A or B will result in change in C.
    ```

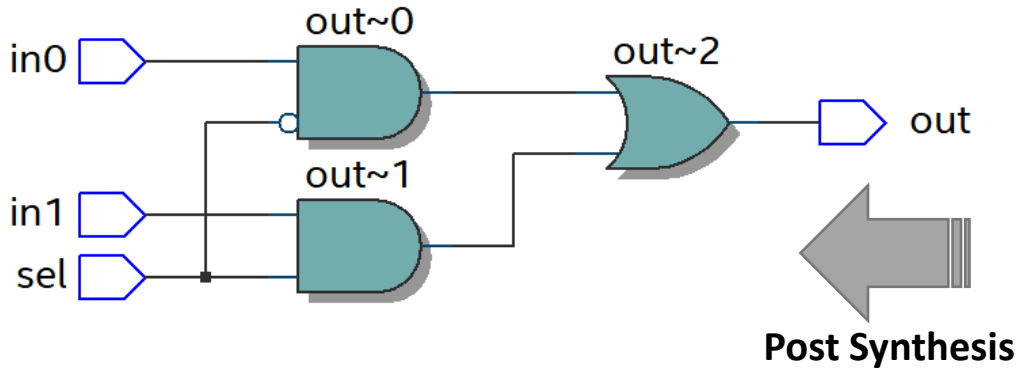
- **About continuous "assign" statement :**
  - Continuous statements are always active statements.
  - It is called **continuous assignment** because in example above, wire "**C**" is **continuously** updated whenever **A** or **B** changes.
  - The RHS **expression (A & B)** is evaluated only when one of its operands changes. Then the result is assigned to the LHS net (**C**).

Dataflow Modeling

- Dataflow description of a 2-to-1 Multiplexer (MUX)








```
module mux_2x1_dataflow (
  input logic in0,in1,sel,
  output logic out);
  assign out = (!sel && in0) || (sel && in1);
endmodule
```

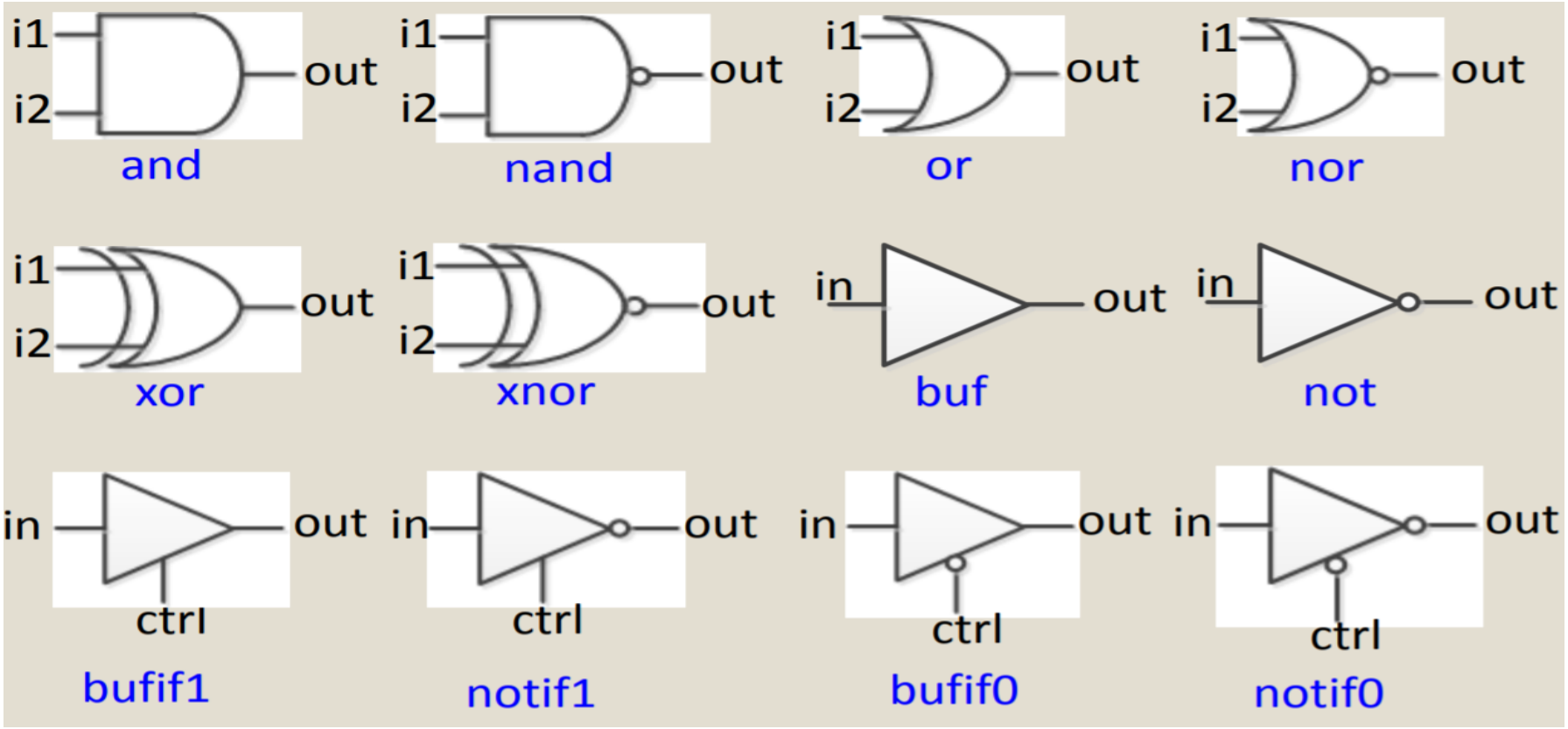
Multiplexer representation using logical expression.
No gates or hardware implementation details specified.

Gate Level Modeling

- **Design module is specified in terms of logic gates and interconnections between these gates.**
  - Resembles a schematic drawing with components connected with signals
  - Low level of modeling abstraction with focus on hardware implementation details and accuracy which includes :
    - Actual logic gates used to design the circuit
    - Timing such as gate propagation delays, rise delays, fall delays, turn-off delays, etc
  - This is closer to the physical implementation of design than the behavior and data flow model

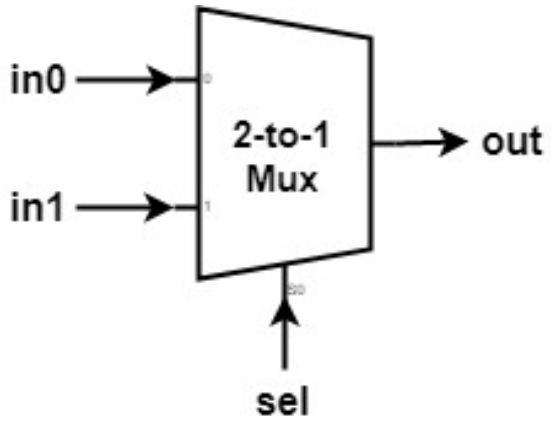
Gate level Modeling

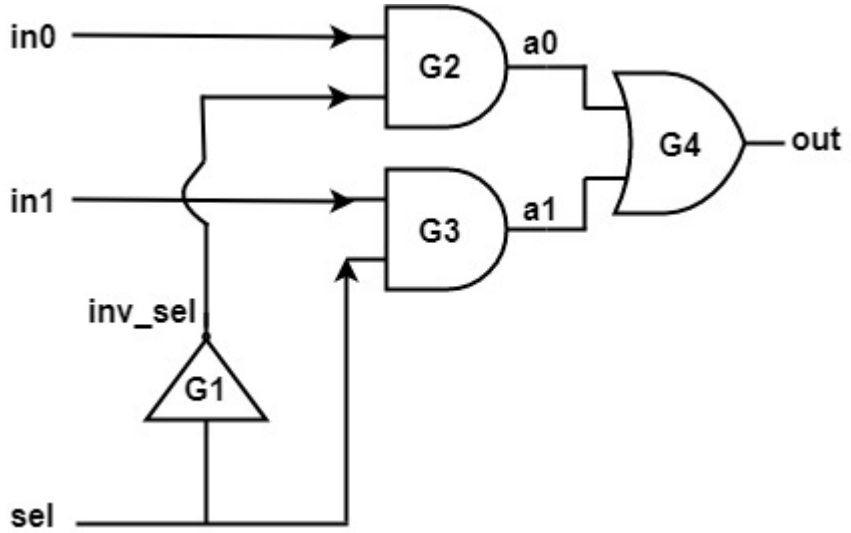
- **SystemVerilog supports modeling digital logic using built-in gate-level primitives**
  - Gate level primitives can closely approximate silicon implementation
    - Specify delays associated with transistors that would be in an actual silicon with a high degree of accuracy
  - Gate-level models are provided by the silicon manufacturing vendor in case of ASIC or by the target FPGA vendor in case of FPGA implementation



Gate level Modeling

❑ Gatelevel description of a 2-to-1 Multiplexer

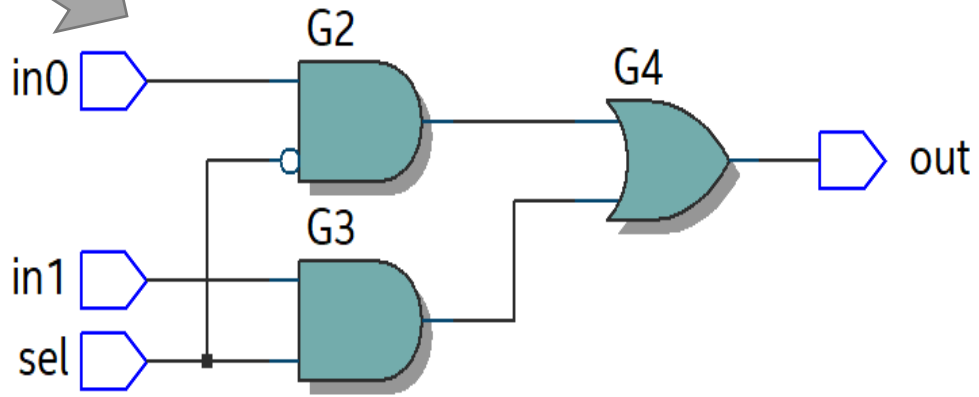





```
module mux_2x1_gatelevel(
  input logic in0, input logic in1, input logic sel,
  output logic out);
  wire a0, a1, inv_sel;
  not G1(inv_sel, sel);
  and G2(a0, in0, inv_sel);
  and G3(a1, in1, sel);
  or  #(1, 1)  G4(out, a0, a1);  // 1 time unit of rise and fall delay
endmodule
```

Mux representation using built-in gate level primitives

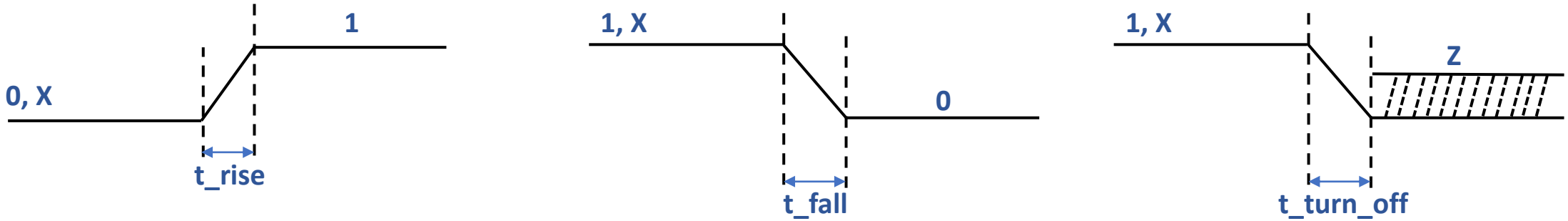
Post Synthesis



Gate level Modeling

❑ Types of delays which can be specified in each primitive

- Rise delay is associated with a gate output transition to 1 from another value (0 or X)
- Fall delay is associated with a gate output transition to 0 from another value (1 or 0)
- Turn-off delay is associated with a gate output transition to the high impedance value (Z) from another value (1 or X)

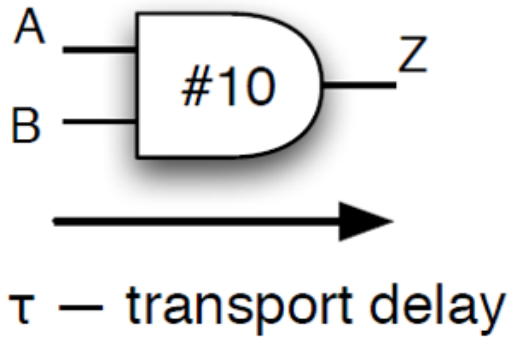


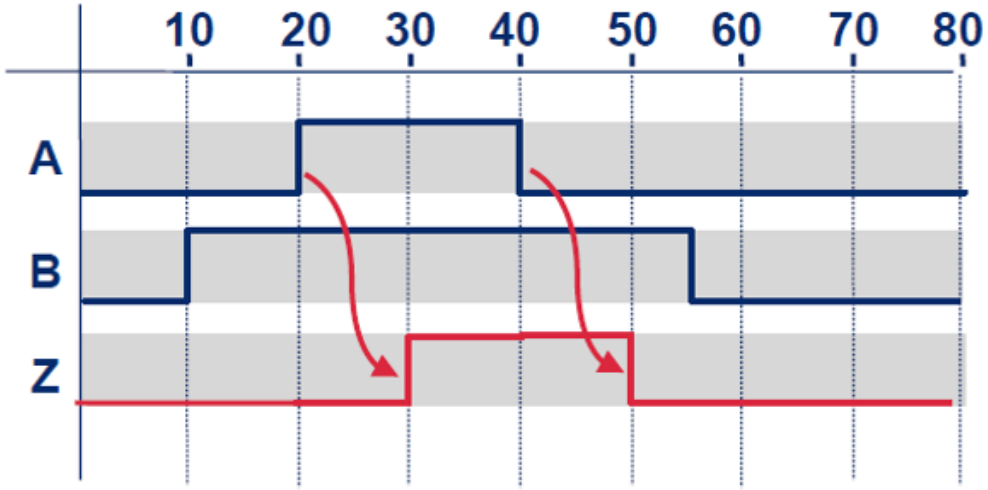
❑ Example :

- **bufif0** #(2, 3, 4) b1(out, in, control) ; // t_rise=2, t_fall=3, t_turn_off=4 time units
- **and** #(2, 3) g2 (out, in1, in2) ; // t_rise=2, t_fall=3 time units
- **nand** #(3) g1 (out, in1, in2) ; // all delay values are 3 time units
- **buf** b1(a, b) ; // t_rise=0, t_fall=0, t_turn_off=0 time units

Gate level Modeling

- **Example : AND Gate**
  - The output follows, after the specified delay, the inputs according to the **AND** function
  - Delay (**#10** here) is the input to output delay – known as **"transport delay"**





- **The delay is given in a unit-less "time-base" (Example : #10, #20, #1)**
  - Using `timescale directive, designer of SystemVerilog code can assign it to any desired time during simulation, e.g. `timescale 1ns/1ns or `timescale 10ps/10ps

- If no delay is specified, then a change on an input to the gate will be immediately reflected on the output of the gate. Also know as **zero delay gate level model**

Switch Level Modeling

- **In Switch level modeling, design module is implemented in terms of transistors, switches, storage nodes, and the interconnections between them**
  - This is the lowest modeling level of abstraction provided by SystemVerilog
  - This level of modeling can closely represent actual silicon implementation hence it is the most accurate representation of design
  - At this level designer requires knowledge of transistor-level implementation details.
  - Slowest to simulate compared to behavioral, RTL, dataflow and gate level models
  - Only digital systems can be modeled at switch level using Standard SystemVerilog
  - Digital simulators does not accurately reflect transistor behavior
  - Switch-level modeling is not used in FPGA design flows

Switch Level Modeling

- **SystemVerilog supports two types of switch primitives for switch level modeling :**
  - **Ideal switch :**
    - when switch is closed(ON), there is zero resistance, hence no signal degradation
  - **Resistive switch :**
    - when switch is closed(ON), there is low resistance, hence signal degradation
    - when signal passes resistive switch signal strength decreases
    - Resistive switch primitives in SystemVerilog starts with "**r**" such as **r**nmos, **r**pmos
- **Some of the Switch primitives supported in SystemVerilog are mentioned in table below:**

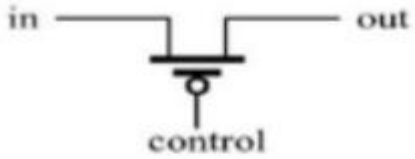
Switch Type	SystemVerilog Switch Primitives
Ideal MOS switches	pmos, nmos, cmos
Resistive MOS switches	rpmos, rnmos, rcmos
Ideal Bidirectional switches	tran, tranifo0, tranif1
Resistive Bidirectional switches	**r**tran, **r**tanif0, **r**tranif1
Power and Ground nets	supply1 and supply0
Pullup and Pulldown	pullup and pulldown

Switch Level Modeling

- **nmos, pmos and cmos switches**

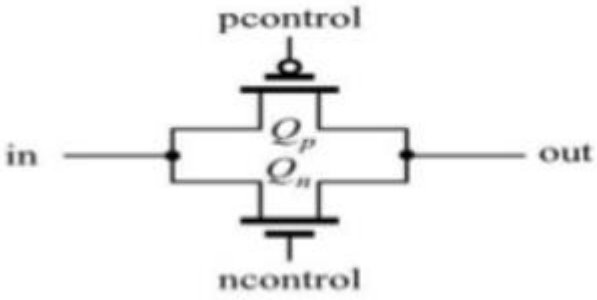
nmos		control			
		0	1	x	z
in	0	z	0	L	L
	1	z	1	H	H
	x	z	x	x	x
	z	z	z	z	z

(a) nMOS switch



pmos		control			
		0	1	x	z
in	0	0	z	L	L
	1	1	z	H	H
	x	x	z	x	x
	z	z	z	z	z

(b) pMOS switch



(a) Symbol

control		data			
n	p	0	1	x	z
0	0	0	1	x	z
0	1	z	z	z	z
0	x	L	H	x	z
0	z	L	H	x	z
1	0	0	1	x	z
1	1	0	1	x	z
1	x	0	1	x	z
1	z	0	1	x	z
x	0	0	1	x	z
x	1	L	H	x	z
x	x	L	H	x	z
x	z	L	H	x	z
z	0	0	1	x	z
z	1	L	H	x	z
z	x	L	H	x	z
z	z	L	H	x	z

Switch Instance Synax :

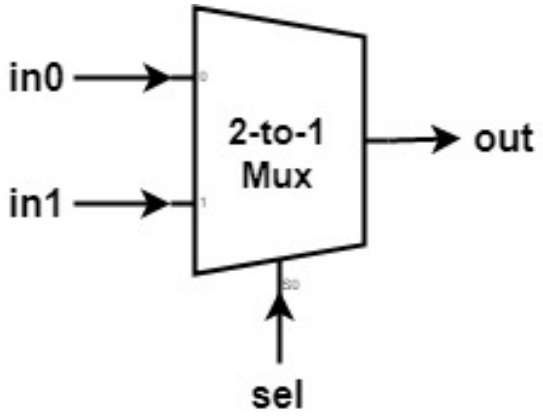
nmos <instance name> (out, in, ncontrol);

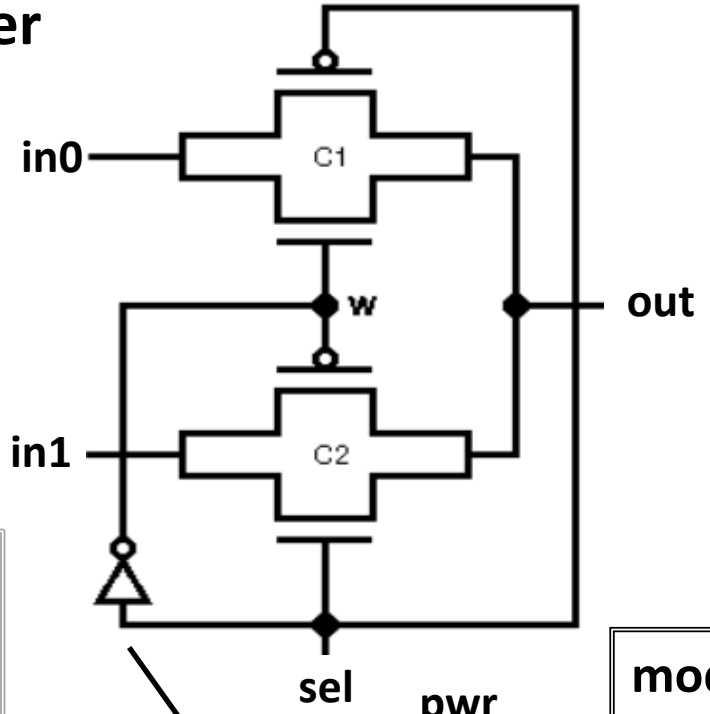
pmos <instance name> (out, in, ncontrol);

cmos <instance name> (out, in, ncontrol);

Switch Level Modeling

❑ Switch level description of a 2-to-1 Multiplexer

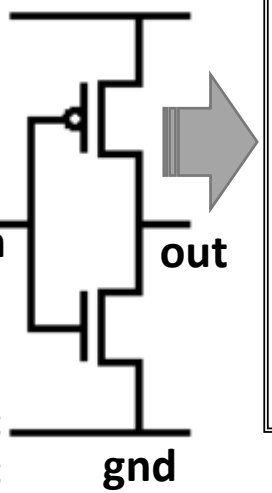





```
module mux_2x1_switchlevel (
  input in0, input in1, input sel, output out
);
  wire w;
  inv G1(w, sel);    // invertor switch model instantiated
  cmos C1(out, in0, w, sel);
  cmos C2(out, in1, sel, w);
endmodule
```

SV Built-in switch primitives instantiated

Invertor modeled using switch primitives

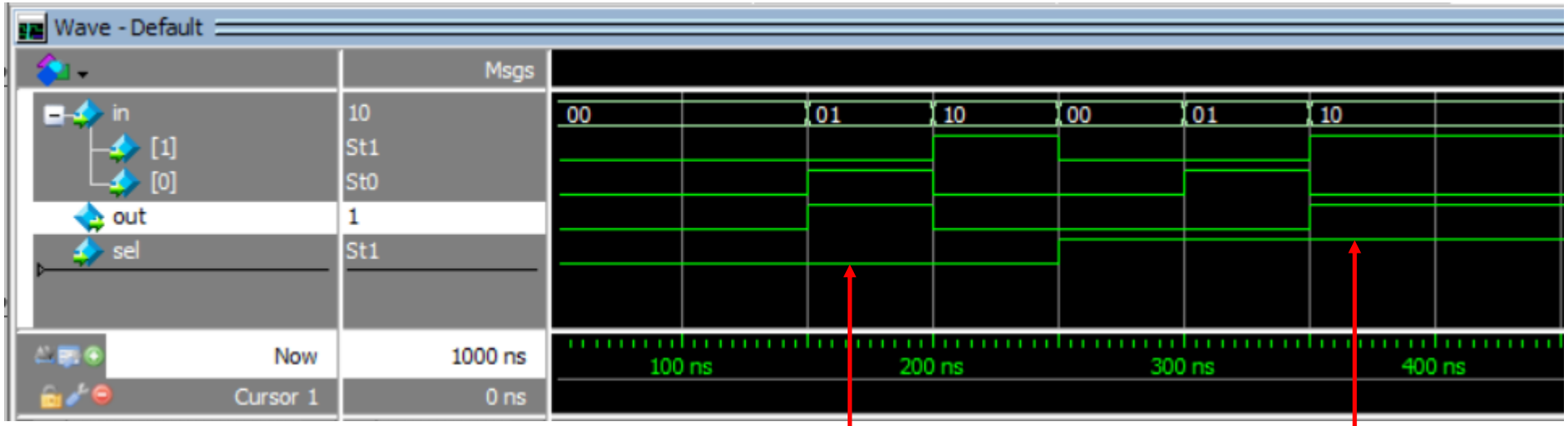



```
module inv(out, in);
  output out;
  input in;
  supply1 pwr;
  supply0 gnd;
  pmos (out, pwr, in);
  nmos (out, gnd, in);
endmodule
```

- When sel=0 then w = 1 and cmos C1 is ON and cmos C2 is OFF hence in0 will propagate to out
- When sel=1 then w = 0 and cmos C1 is OFF and cmos C2 is ON hence in1 will propagate to out

2-to-1 Multiplexer Functional Simulation Result

❑ Simulation Waveform of behavioral and dataflow model



When sel == 0, "in0" propagates to "out"

When sel == 1, "in1" propagates to "out"

❑ Simulation output log

```
Vsim > run 500ns
#  time=0    ,  in=00  sel=0  out=0
#  time=150,  in=01  sel=0  out=1
#  time=200,  in=10  sel=0  out=0
#  time=250,  in=00  sel=1  out=0
#  time=300,  in=01  sel=1  out=0
#  time=350,  in=10  sel=1  out=1
```

RTL (Register Transfer Level) Modeling

- **RTL modeling utilizes combination of behavioral and dataflow modeling**
  - **Register:** Storage element like Flipflop, Latches
  - **Transfer:** Transfer data between input, output and register using combinational logic.
  - **Level:** Level of Abstraction modeled using HDL
- **RTL modeling is used to describe both sequential and combinational logic circuits**
- **Two primary constructs for RTL modeling :**
  - continuous assignments (**assign**) and always procedural blocks (**always**)
- **RTL is a cycle accurate model of a design unlike behavioral model**
- **RTL (Register-Transfer-Level) model is always synthesizable unlike behavioral model**
- **Similar to behavioral model, RTL models :**
  - Does not contain low level hardware implementation details
  - Simulate faster than gate-level and switch-level models, making it possible to verify larger and more complex designs in simulation
  - Complex designs can be quickly and concisely modeled using RTL modeling technique compared to gate level modeling

Behavioral vs RTL Level Modeling

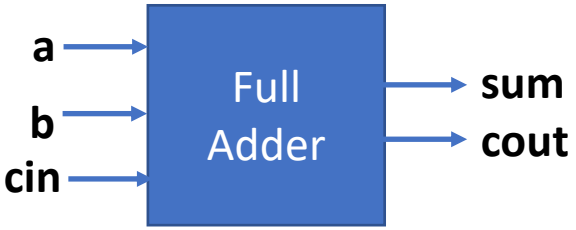
- **Behavioral model describes what intended design does. An RTL model describes how it does it.**
  - For example, an RTL half adder must describe the registers for the operands, the carry method, and a clock. It may be done in various ways to balance speed and accuracy
  - For a behavioral adder, A + B = C is sufficient. A + B = C also simulates a lot faster than all those gates and registers in the RTL model.

- **Behavioral model looks similar to RTL model since both uses always procedural always blocks, however they differ :**
  - RTL model executes its programming statements in a single clock cycle, or in zero cycles if combinational logic.
  - Behavioral block can take arbitrary number of clock cycles to execute is statements.

- **RTL (Register-Transfer-Level) modeling is used with intent to generate hardware hence it is always synthesizable unlike behavioral model**
  - Behavioral model though mimics the desired functionality of the hardware but not necessarily synthesizable since it can be used for verification or architecture exploration purpose

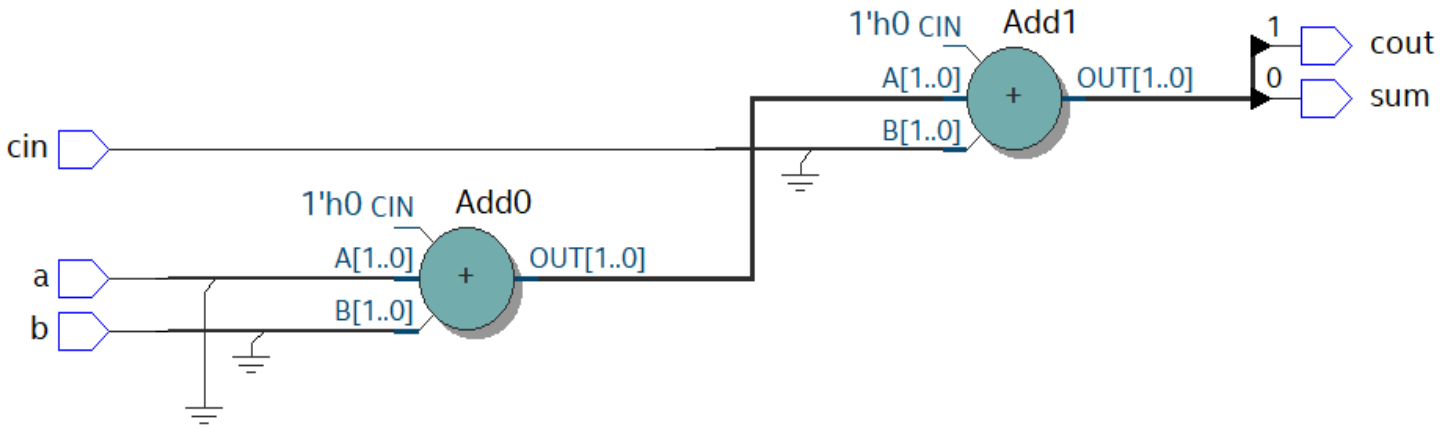
Behavioral Model of Full Adder

❑ Behavioral level description of Full Adder



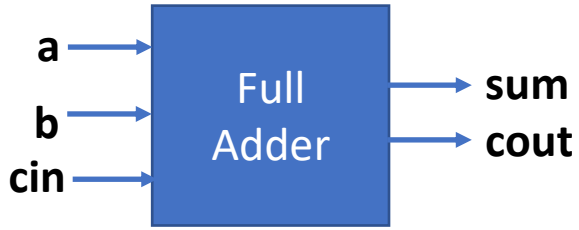

```
module fulladder_behavioral(
  input logic a, b, cin,
  output logic sum, cout
);
  assign {cout, sum} = a + b + cin;
endmodule
```

Post Synthesis



Dataflow level Model of Full Adder

❑ Dataflow level description of Full Adder




```
module fulladder_dataflow(
  input logic a, b, cin,
  output logic sum, cout
);
  logic p, q;
  assign p = a ^ b;
  assign q = a && b;
  assign sum = p ^ cin;
  assign cout = q || (p && cin);
endmodule
```

Full Adder Truth Table

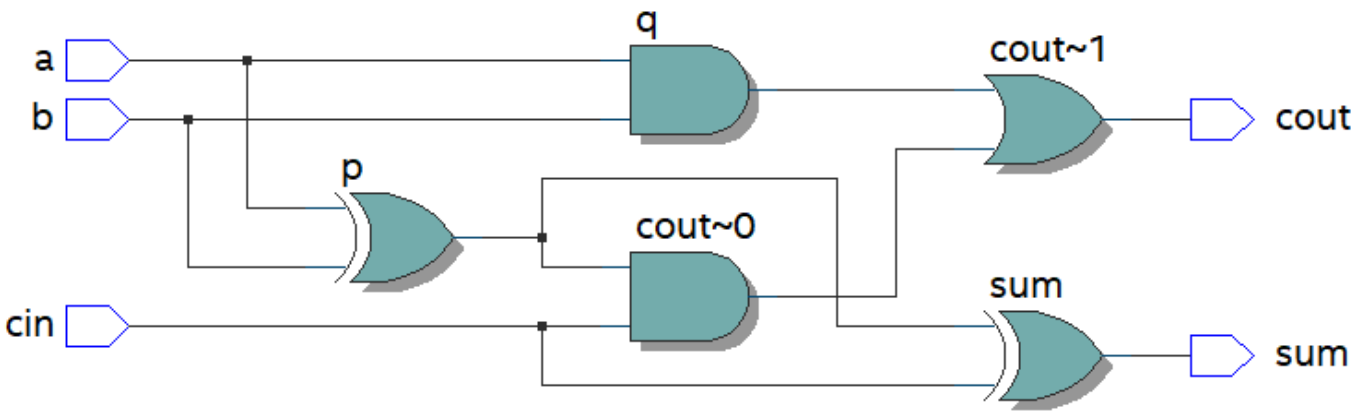
inputs			outputs	
a	b	cin	sum	cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Boolean Expression for Full Adder based on Truth Table

sum = ((a xor b) xor cin);

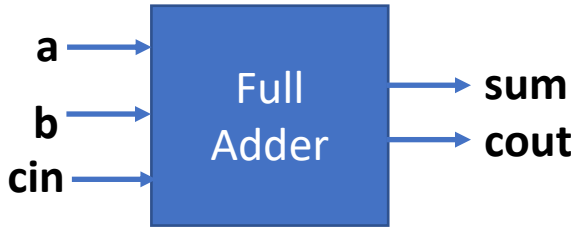
cout = (a and b) or ((a xor b) and cin)

Post Synthesis



Gatelevel Model of Full Adder

- Gate level description of Full Adder



Full Adder Truth Table

inputs			outputs	
a	b	cin	sum	cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

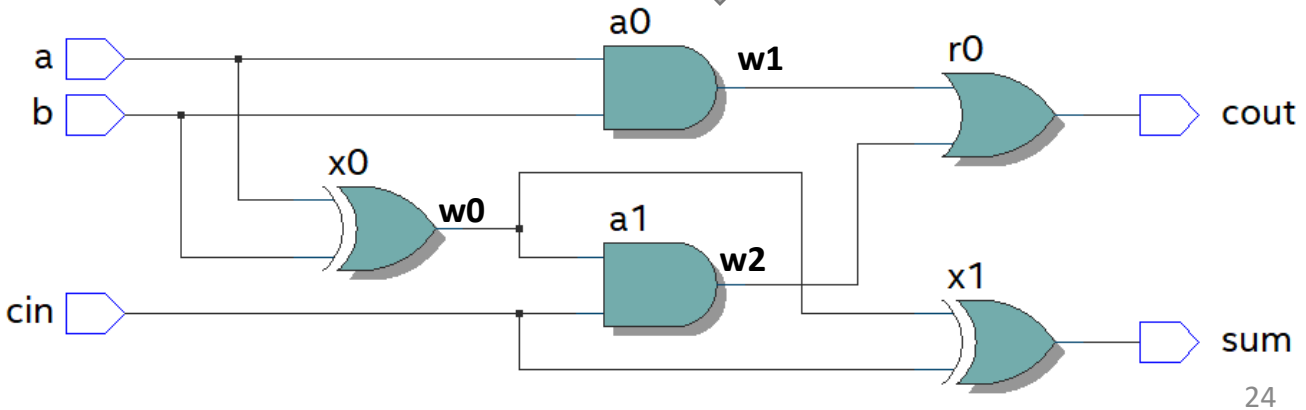
Boolean Expression for Full Adder based on Truth Table

sum = ((a xor b) xor cin);

cout = (a and b) or ((a xor b) and cin)

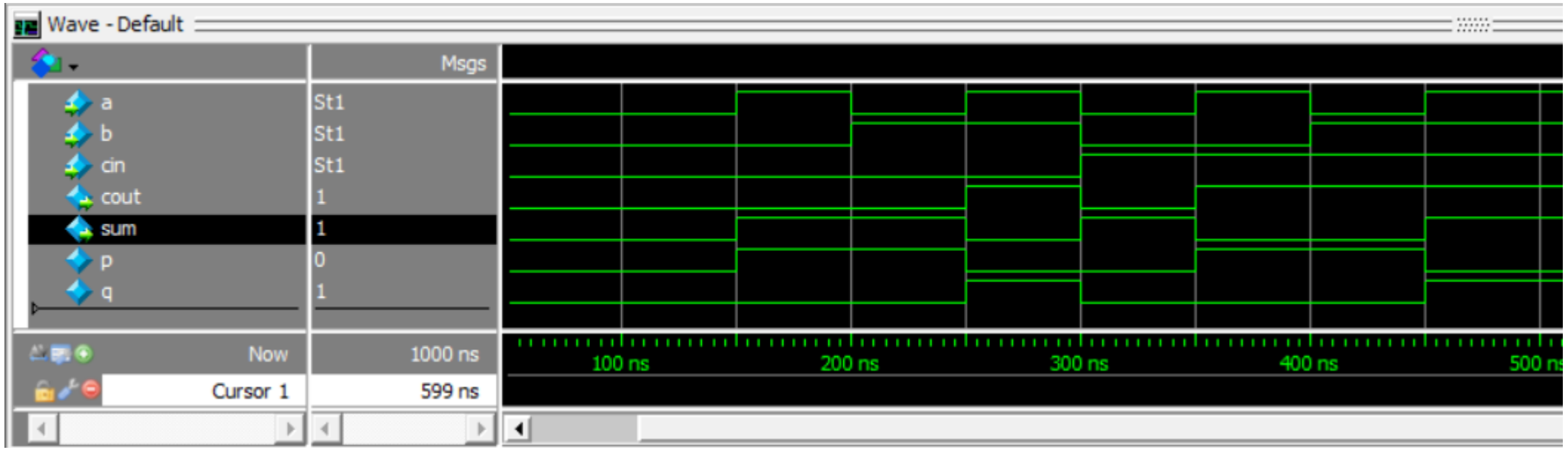
```
module fulladder_gatelevel(
  input logic a, b, cin,
  output logic sum, cout
);
   wire w0, w1, w2;
   xor x0(w0, b, a);
   and a0(w1, b, a);
   and a1(w2, w0, cin);
   or r0(cout, w2, w1);
   xor x1(sum, w0, cin);
endmodule
```

Post Synthesis



FullAdder Simulation Result

- **Waveform (Same results for behavioral, dataflow and gatelevel model simulation)**



- **Simulation output log**

```
Vsim > run 500ns
#  time=0     a=0  b=0  c=0  sum=0  cout=0
#  time=150   a=1  b=0  c=0  sum=1  cout=0
#  time=200   a=0  b=1  c=0  sum=1  cout=0
#  time=250   a=1  b=1  c=0  sum=0  cout=1
#  time=300   a=0  b=0  c=1  sum=1  cout=0
#  time=350   a=1  b=0  c=1  sum=0  cout=1
#  time=400   a=0  b=1  c=1  sum=0  cout=1
#  time=450   a=1  b=1  c=1  sum=1  cout=1
```

Homework Assignment-1

- **Using Quartus Alter Prime Software:**
  - Develop 2-to-4 decoder behavioral model and dataflow model
  - Synthesize 2-to-1 mux, 2-to-4 decoder and full adder behavioral, dataflow and gate-level models
  - At the end of synthesis :
    - Review RTL and post mapping netlist schematics
    - Make observations between RTL and post mapping schematics
    - Review FPGA resource allocation logs generated from Altera Quartus

- **Using Modelsim Software:**
  - Run simulation using testbench and design for mux, decoder and full adder
  - Review waveform for each design in Modelsim waveform viewer.
    - Observe if any difference in behavior of behavioral vs dataflow vs gate-level modes
    - Provide only 1 simulation waveform per design if the simulation results are same

- **Note:**
  - Intent of this first assignment is to get familiar using Altera and Modelsim tools, review synthesis and simulation results.
  - SystemVerilog design and testbench code will be provided to students on Piazza.