ECE3270

CAD NOTES

Introduction

- Traditional digital design:
  - Manual process of designing and capturing circuits
    - Schematic entry
    - Low level primitives (Boolean gates)
- Bottom-up design
  - Design simple components or modules (registers, counters, etc.)
  - Test desired functionality
  - Integrate into larger designs

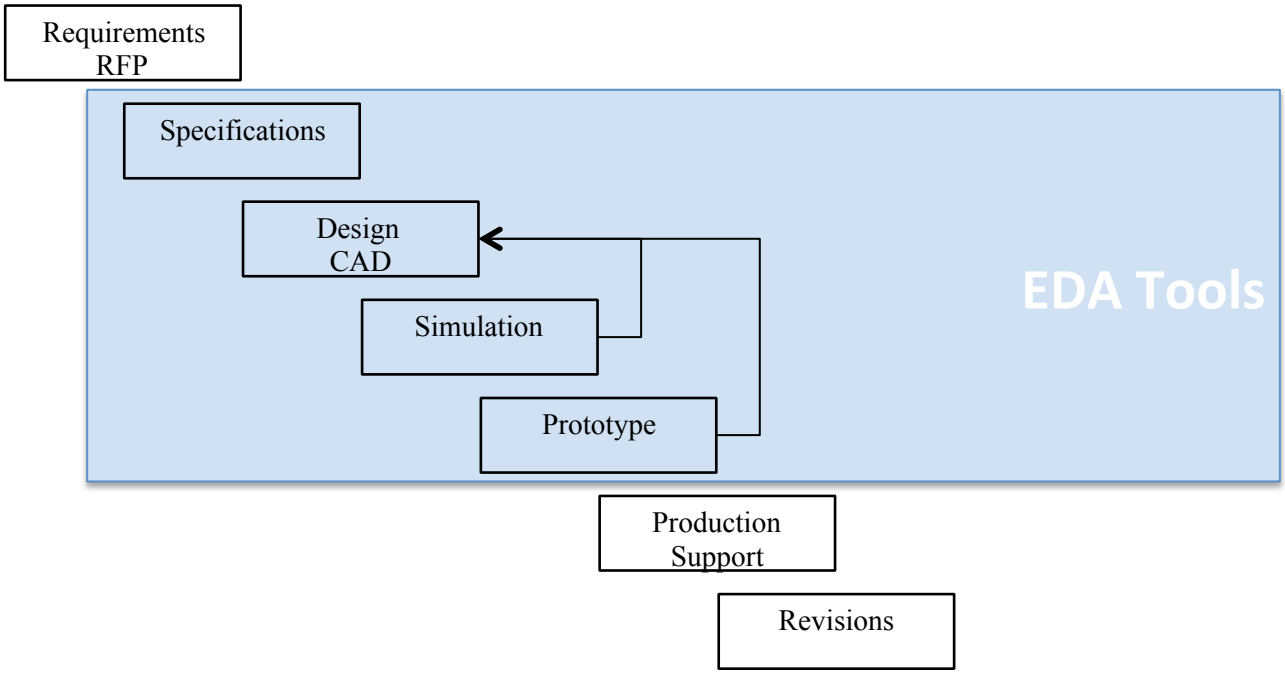
Introduction

- System-level design for complex systems:
  - Top-down design
    - Required due to design complexity
    - Meets rapid time-to-market requirements
    - Utilize libraries of tested modules and interfaces
  - Hardware Description Language (HDL) use
    - Synthesis – Use a "hardware compiler" to implement a design
    - Applicable to different digital "product" types
      - ASIC, CPLD, FPGA, etc.

Top-down HDL Design Methodology Benefits

- Increased productivity
  - Shorter development cycle
  - Can incorporate more product features
  - Reduced time-to-market
- Reduced non-recurring engineering (NRE) costs
  - Controlled design process and fewer mistakes
- Design reuse facilitated
- Flexibility in design changes
- Rapidly explore alternative
  - Architectures
  - Implementation technologies

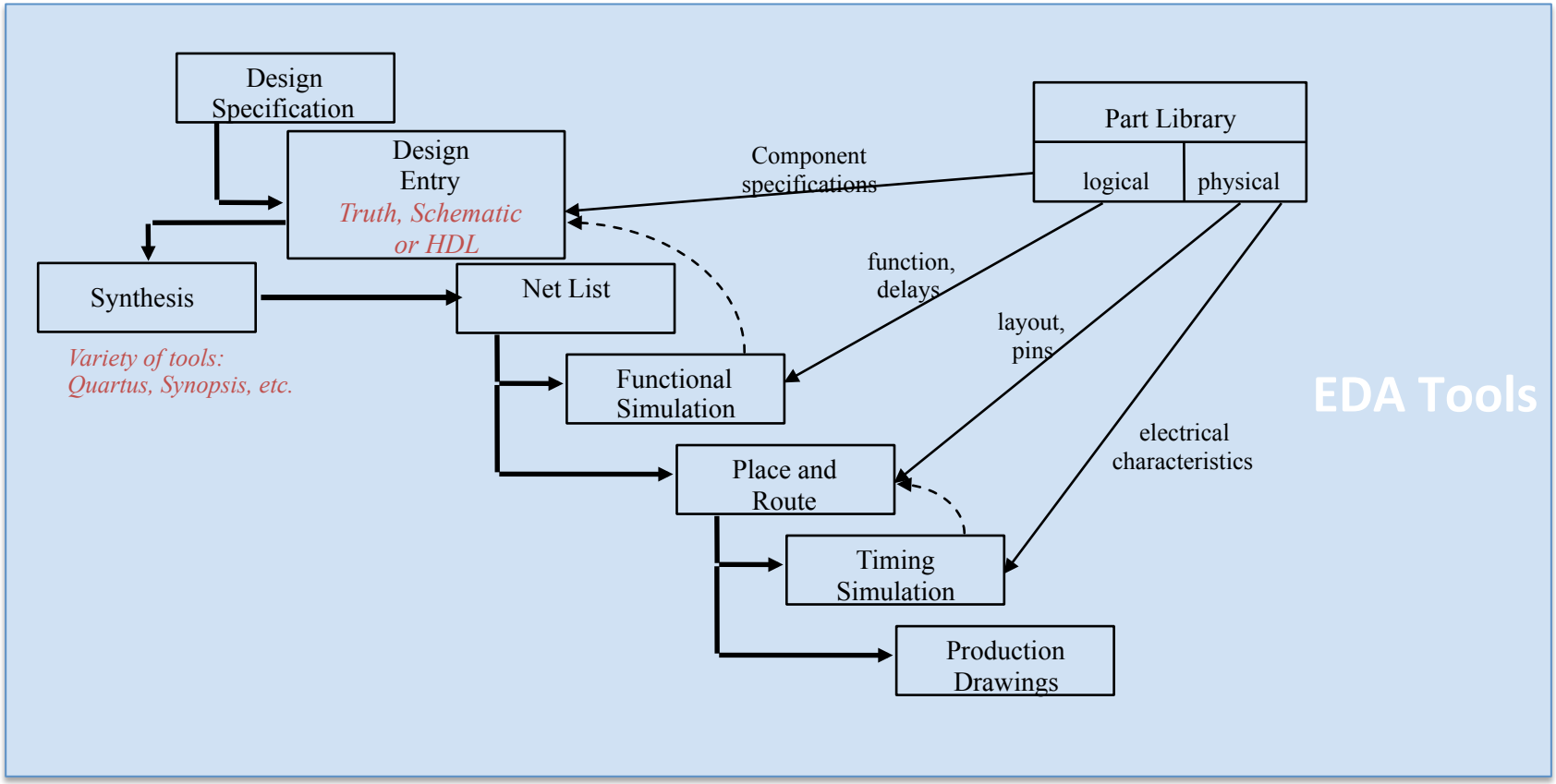
Design Cycle



Electronic Design Automation Tools

- An EDA system usually includes the following tools
  - Design entry
  - Synthesis and optimization
  - Simulation
  - Physical design

Design Entry and Simulation

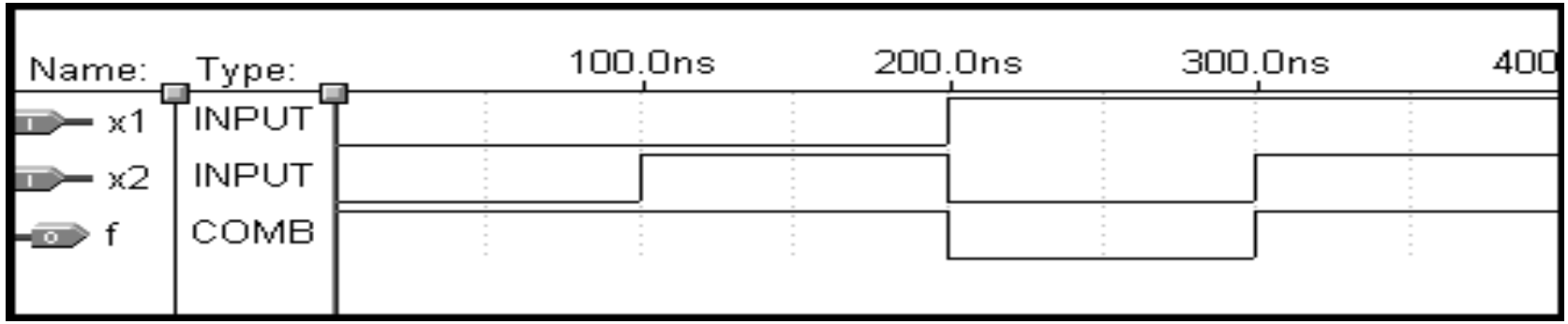


Design Entry

- The process of entering into the CAD system a description of a circuit being designed is called ***design entry***
- Three common design entry methods
  - Truth tables, waveforms, state diagrams
    - User enters a truth table in plain text format or draws a waveform that represents the desired functional behavior
  - Schematic capture
    - User graphically enters a desired logic circuit
  - Hardware description languages
    - User enters a programming language-like description of a desired logic circuit

Design Entry with Truth Tables

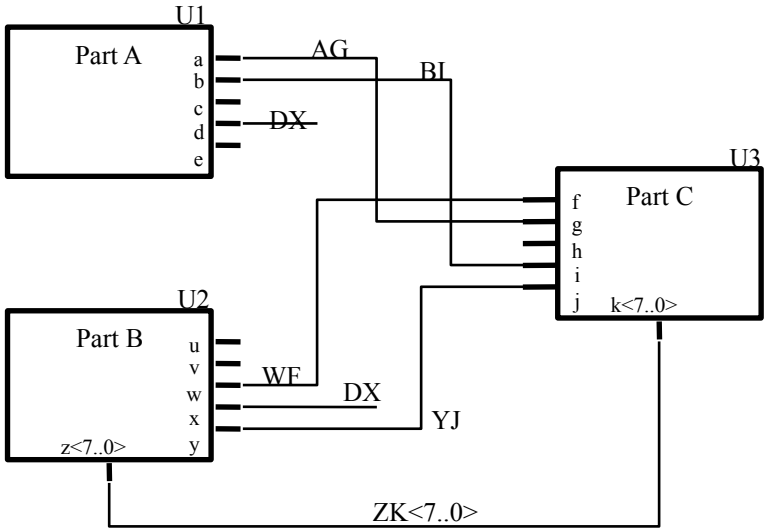
- Commonly use a ***waveform editor*** to enter a timing diagram that describes a desired functionality for a logic circuit
  - CAD system transforms this into equivalent logic gates
  - Not appropriate for large circuits, but can be used for a small logic function that is to be part of a larger circuit



Schematic Capture

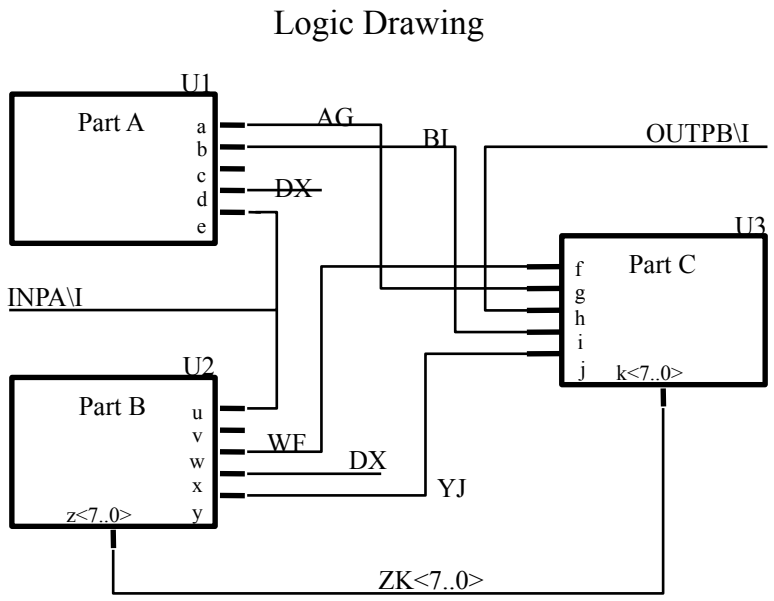
- ***Schematic***: refers to a diagram of a circuit in which circuit elements (logic gates) are shown as graphical symbols and connections between them are drawn as lines
- Tool provides a collection of symbols that represent gates of various types with different inputs and outputs - A ***library***
- Previously designed circuits can be represented with a graphical symbol and used in larger circuits
  - Known as ***hierarchical design*** and provides a way of dealing with complexities of large circuits

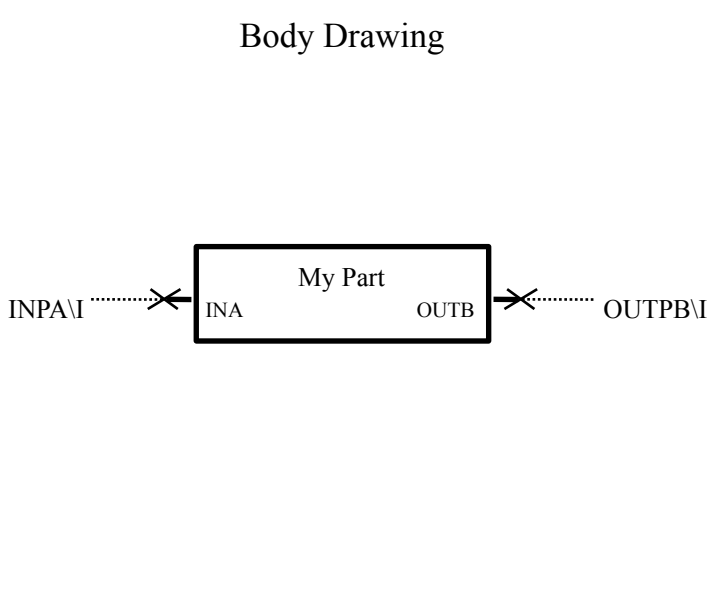
Schematic Capture



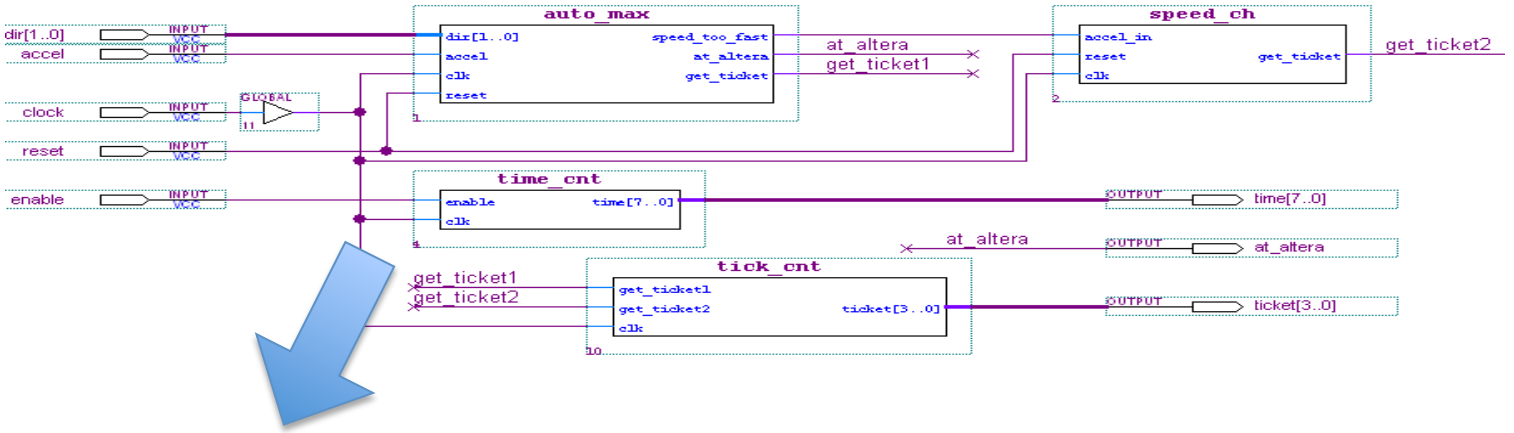
- Parts are **generic objects** with:
  - symbol
  - pins
  - part name (Part A)
  - instance name (U1)
- Wires connect pins
  - may be 1 or more bits (bus)
  - have a signal name
  - may be implicit (DX)
- Netlist
  - connected signal names
    - U1.a, AG, U3.g
    - U2.w, WF, U3.f
    - etc.
  - each part instance
    - U1 is a Part A
    - U2 is a Part B
    - U3 is a Part C

Schematic Capture - Hierarchical





Hierarchical Design




```
SUBDESIGN time_cnt
(
     enable, clk       : INPUT;
     time[7..0]        : OUTPUT;
)
VARIABLE
     count[7..0]       : DFF;
BEGIN
     count[].clk = clk;
     time[] = count[];

     IF enable THEN
          count[].d = count[].q + 1;
     ELSE
          count[].d = count[].q;
     END IF;

END;
```

Hardware Description Languages

- A hardware description language (HDL) is similar to a computer program except that it is used to describe hardware
- Common HDLs
  - VHDL (VHSIC Hardware Description Language)
  - Verilog
  - Many others (vendor specific)
- VHDL and Verilog are standards
  - Offer the option for portability across different CAD tools and different types of programmable chips
- Provide
  - Textual definition of netlists
  - User-defined primitive models
  - High-level functionality models
  - Mixed-level simulation

HDLs

Support of hierarchical design methodology

- Behavioral models
  - describe behavior, not implementation
  - supports top-down design
  - supports mixed-mode simulation
  - some methods support synthesis
- Structural models
  - textual netlists
  - supports synthesis
  - supports bottom-up design

Schematic vs. HDL Capture

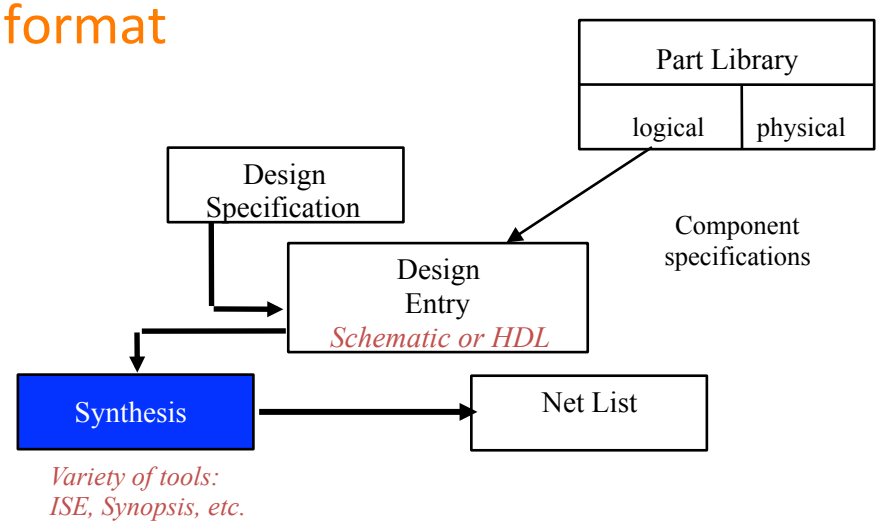
- Schematic capture can be awkward for large circuits
- HDL advantages:
  - Very portable
  - Focus is on functionality - not technology
  - Text based, easy to include documentation in the design
  - Modular

Synthesis

- Synthesis CAD tools generate a logic circuit from stated functional behavior (schematic or HDL)
- ***Translating*** (***compiling***) VHDL code/schematic into a network of logic gates is a part of synthesis
- Not only will the CAD tool produce a logic circuit, but it can also optimize that circuit (from the designer's original input)
  - In terms of speed and/or size (*logic optimization*)
  - Called ***logic synthesis*** or ***logic optimization***
- Finally, ***technology mapping*** and ***layout synthesis*** (physical design) completes the synthesis process

Netlist Output

- Each tool's unique format
  - complex set of translators
- EDIF
  - early attempt at a standard format
  - various versions
- HDLs (structural)
  - Verilog
  - VHDL

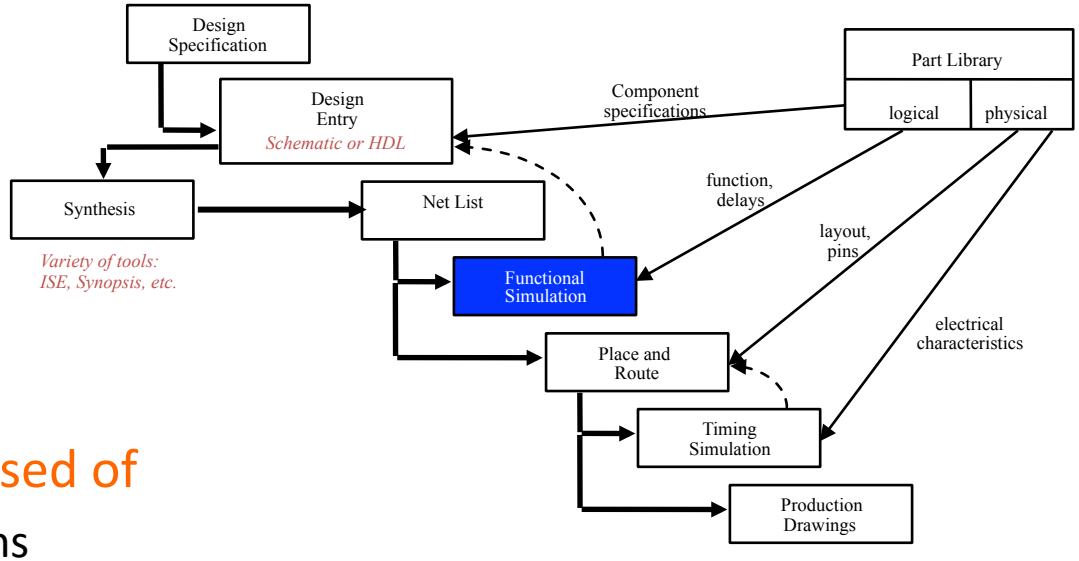


Simulation

- Once designed, it is necessary to verify that the designed circuit functions as expected
- In a ***functional simulation*** the user specifies valuations of the circuits inputs and the CAD tool generates the outputs (commonly in the form of a timing diagram)
  - Simulator provides computer model of the circuit primitives (from the netlist) and simulation engine to compute their interactions providing the resulting outputs
  - User verifies generated outputs against expected outputs
- Functional simulators assume the time needed for signals to propagate through the logic gates is **negligible**
  - Sometimes called zero-delay simulators
  - For a real implementation this is not sufficient
  - Use a ***timing simulator*** to obtain accurate (complete) simulation

Inputs to Simulation

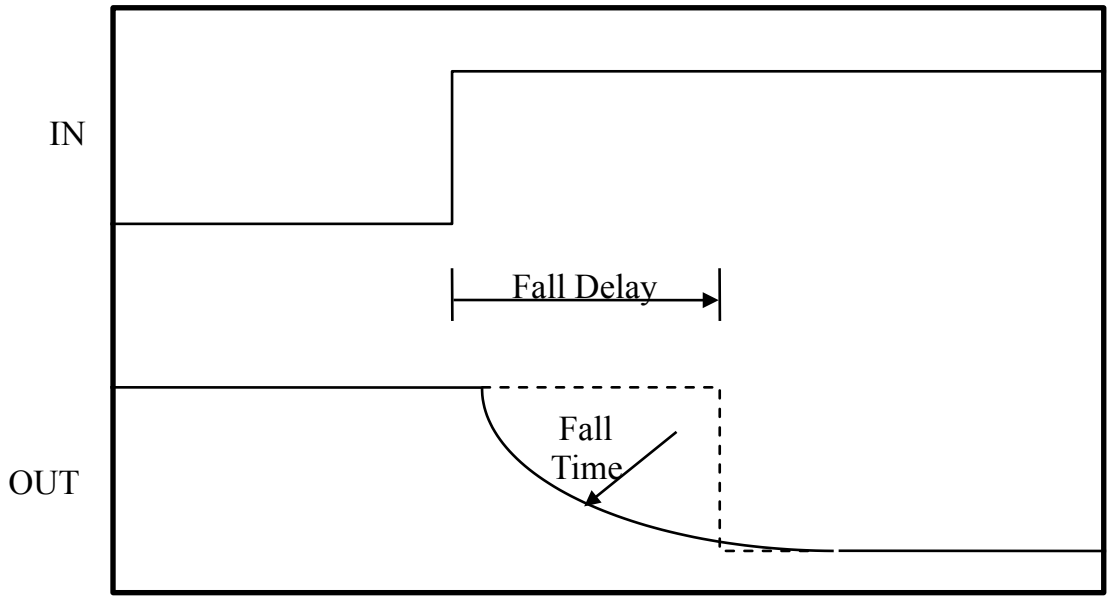
- Netlist (circuit)
- Stimulus
  (input waveforms)
- Device models
  (from device library)
  - Device models composed of
    - Primitive logic functions
    - Timing information



Simulation Models

- Primitive logic functions
  - AND, OR, NOT, NAND, NOR, XOR
  - TSB, TG, pull-up, etc.
  - All treated as no-delay
- Add Timing
  - Rise time: best, worst, typical
  - Fall time: best, worst, typical
  - Setup time, hold time, pulse width

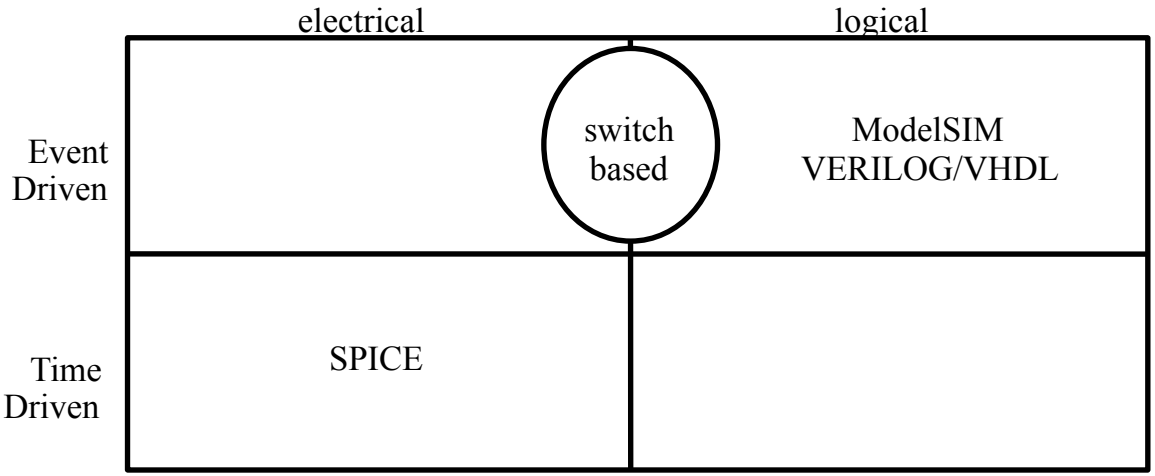
Rise/Fall Time and Delay



Similar for Rise Time

Demonstrate Setup and Hold Time

Simulation Styles



Time-Driven Simulation

- Approach used by SPICE
- Primitives are resistors, capacitors, transistors, etc. --> continuous!
- Node voltages can be computed at any moment in the simulation
- Simulator computes node voltages every Δt
- Adjust Δt as needed to control error
- Can be VERY expensive (time-wise)

Event-Driven Simulation

- Only re-compute node voltages at interesting times
  → when something changes
- Estimate logic levels (0 and 1)
- Operates on a set of logic tables that model the circuit
- Tracks logic state and strength of the source driving the node
- Estimate rise and fall delay (Ignore rise and fall time)
- Use basic logic operations (AND, OR, NOT) as primitives

- Simulator keeps sorted list of future events
- Simulator execution
  - remove next event from list
  - set sim time to time of the event
  - simulate the event (primitives)
  - insert new events (caused by the current event) into the event list
  - run until no more events

Event-Driven Simulation

- Example - an inverter
  - Primitive model

```
PRIMITIVE inverter BEGIN
  when input goes high to low
    schedule output high in RDELAY cycles
  when input goes low to high
    schedule output low in FDELAY cycles
END
```

  - Use netlist to locate all other gates that are connected to output

Event-Driven Simulation

- Simulation output
  - list of changes to signals (high-low, low-high)
  - usually used to create a timing diagram
  - also may be used as input to another simulation
- Simulation cost
  - reduced to time needed to schedule and cause events - computation is minimal
  - no longer sensitive to circuit density, speed, or technology

Event-Driven Simulation

- **Race and hazard conditions** present problems in zero-delay simulators since the simulator does not indicate when the event occurs
- **Race condition** - when two or more signals are changing simultaneously in a circuit - may result in an incorrect state when a condition is assumed to be stable
- **Hazard condition** - possible occurrence of a momentary value opposite to what is expected causing unanticipated events

We will discuss these in more detail later in the semester…

Basic Logic Values

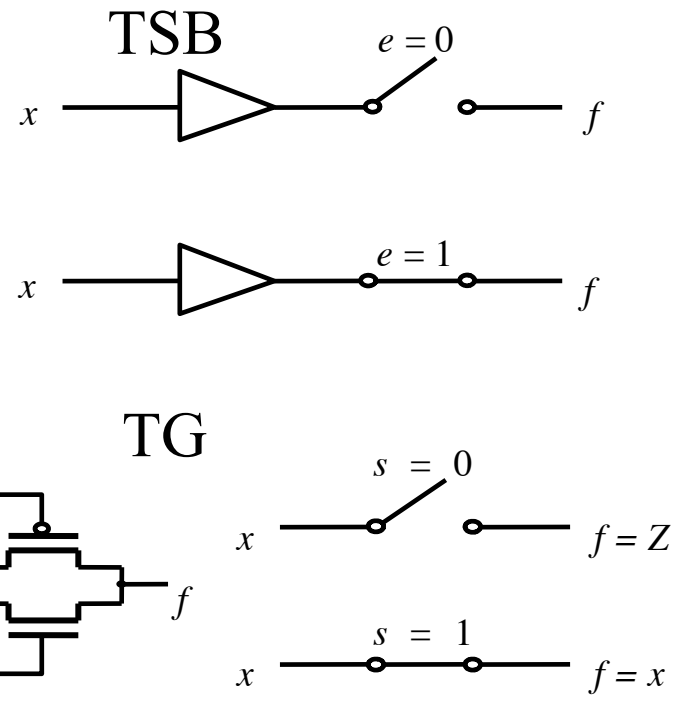
- Each node of circuit in one state
  - 0
  - 1
  - U (unknown or undefined)
  - Z (high impedance)

Conflict Resolution Table

	0	0+	1-	1	Z	U
0	0	0	0+	U	0	U
0+	0	0	U	1-	0+	U
1-	0+	U	1	1	1-	U
1	U	1-	1	1	1	U
Z	0	0+	1-	1	Z	U
U	U	U	U	U	U	U

TSB and TG Problems For Event Simulators

- TSBs (tri-state buffers) and TGs (transmission gates) can charge a node and then stop driving it
  - Add Z+
  - Add Z-
  - Add charge decay
- TGs are bi-directional, and can exhibit charge-sharing
  - VERY hard to simulate

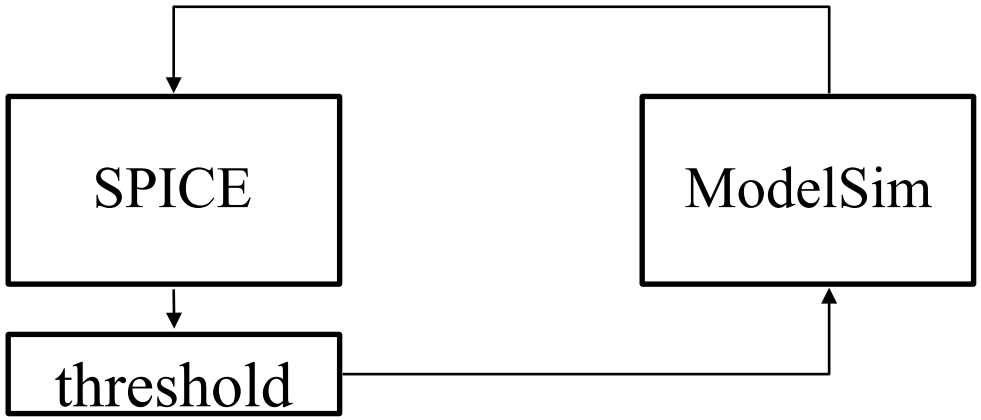


Advanced Logic Simulators

- Can operate at switch- or gate-level
- Switch-level models Xsistors as switches - on/off
  - Keeps track of voltage and logic levels
  - Provides more precise timing than gate-level
  - Lacks ability to use logic element delay as a parameter
- Gate-level treats each logic element as a "black box" modelled by a function
  - Function may model the delay thru the logic element
  - Setting delays to unit value == functional simulation

Hybrid (Mixed-Mode) Simulation

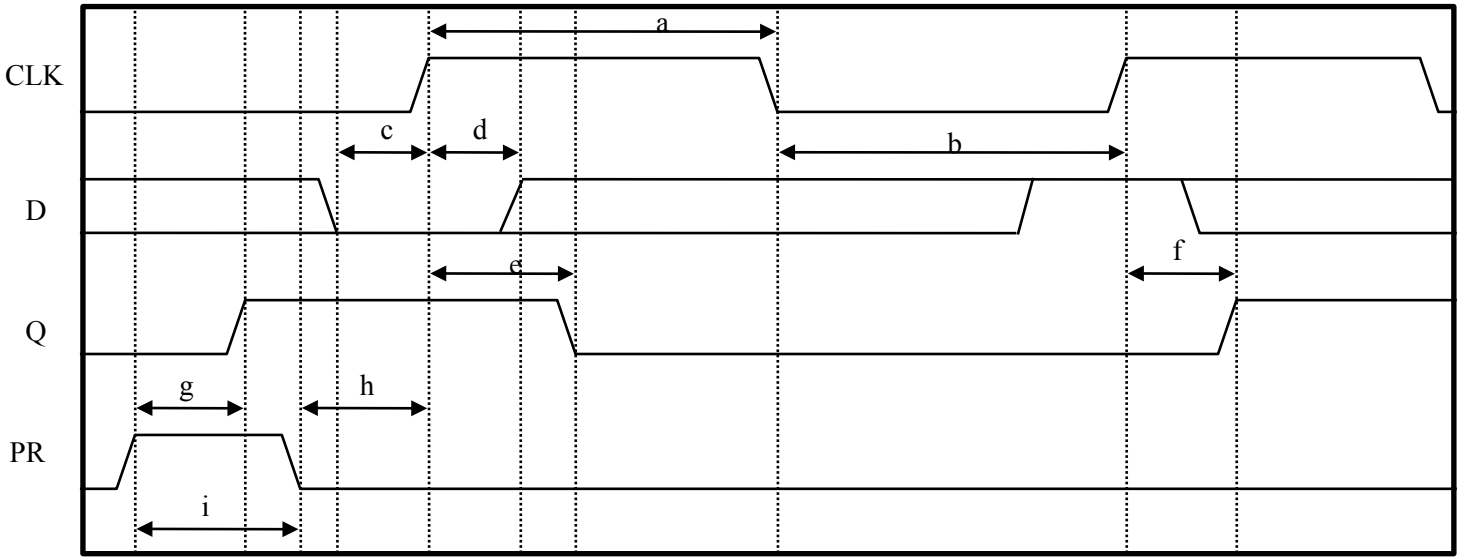
- Electrical (i.e. spice) and logic simulation together
- Partition problem into those areas needing critical timing analysis, and perform full electrical simulation of only those parts



Typical FF Model

- Behavior
  - primitive, no-delay gates, or HDL model
- Timing
  - rise/fall delay from clock to Q
  - setup time, hold time
  - clock minimum high/low pulse width
  - preset/reset delay to Q
  - min preset/reset pulse width
  - min preset/reset to clock spacing

Typical FF Timing



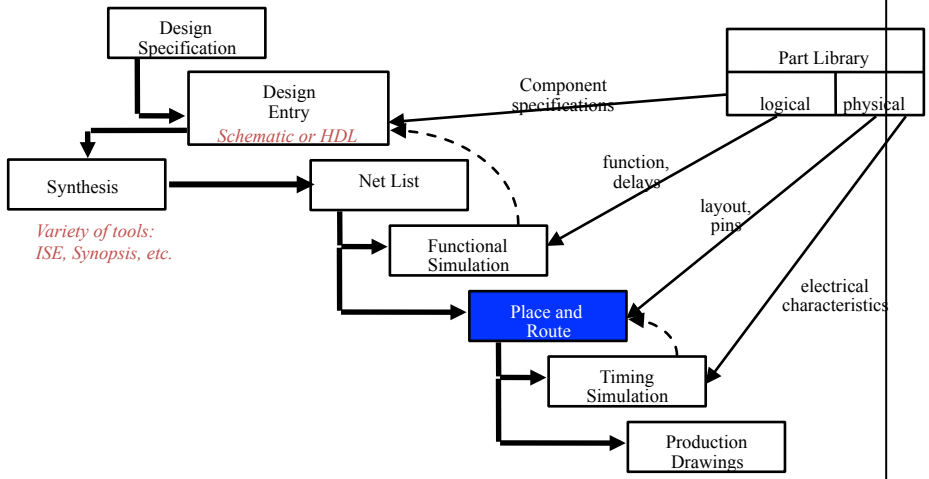
a - minimum clock pulse width high
b - minimum clock pulse width low
c - minimum setup time
d - minimum hold time

e - clock to Q fall delay (avg, worst)
f - clock to Q rise delay (avg, worst)
g - preset to Q rise delay (avg, worst)
h - minimum preset removal to clock
i - minimum preset pulse width

Sample data sheet example…

Routing - Inputs

- Netlist from schematic capture
  - table of part numbers and type
  - connection list by part and pin number
- Physical part data from part library
  - physical dimension
  - pin data
    - placement, size, & mount type
    - electrical characteristics (resistance/capacitance)
    - signal name
    - special flags
- Technology and user settings



Routing - Parameters

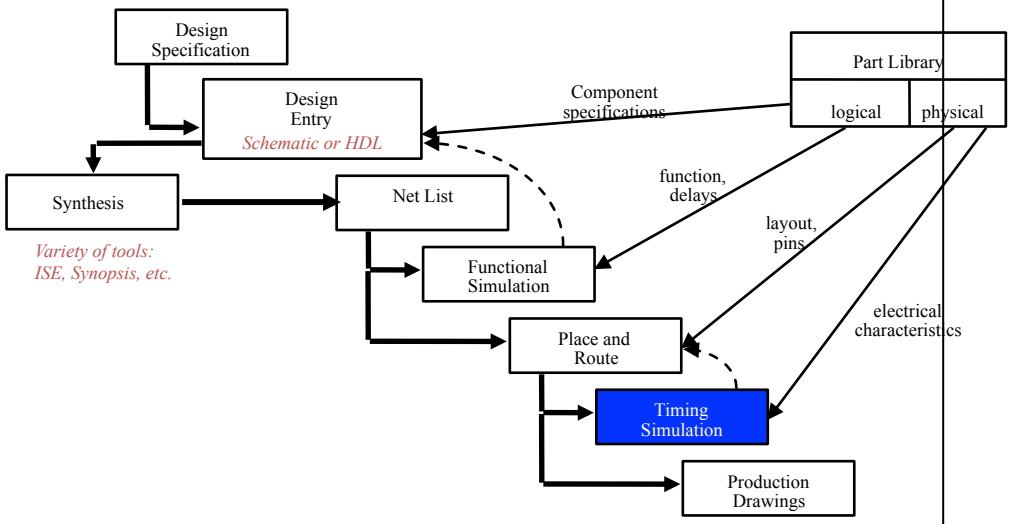
- Router settings
  - dimensions of layout space
  - placement of connectors, some parts
  - routing layers, spacing, interconnects
  - electrical characteristics (resistance, capacitance)
  - routing goals (trace length, power density, etc.)

Routing - Outputs

- Layout
  - part placement
  - interconnect
  - power distribution
- Other Information
  - errors
  - routing density
  - routing data by net in netlist - feed back to simulator

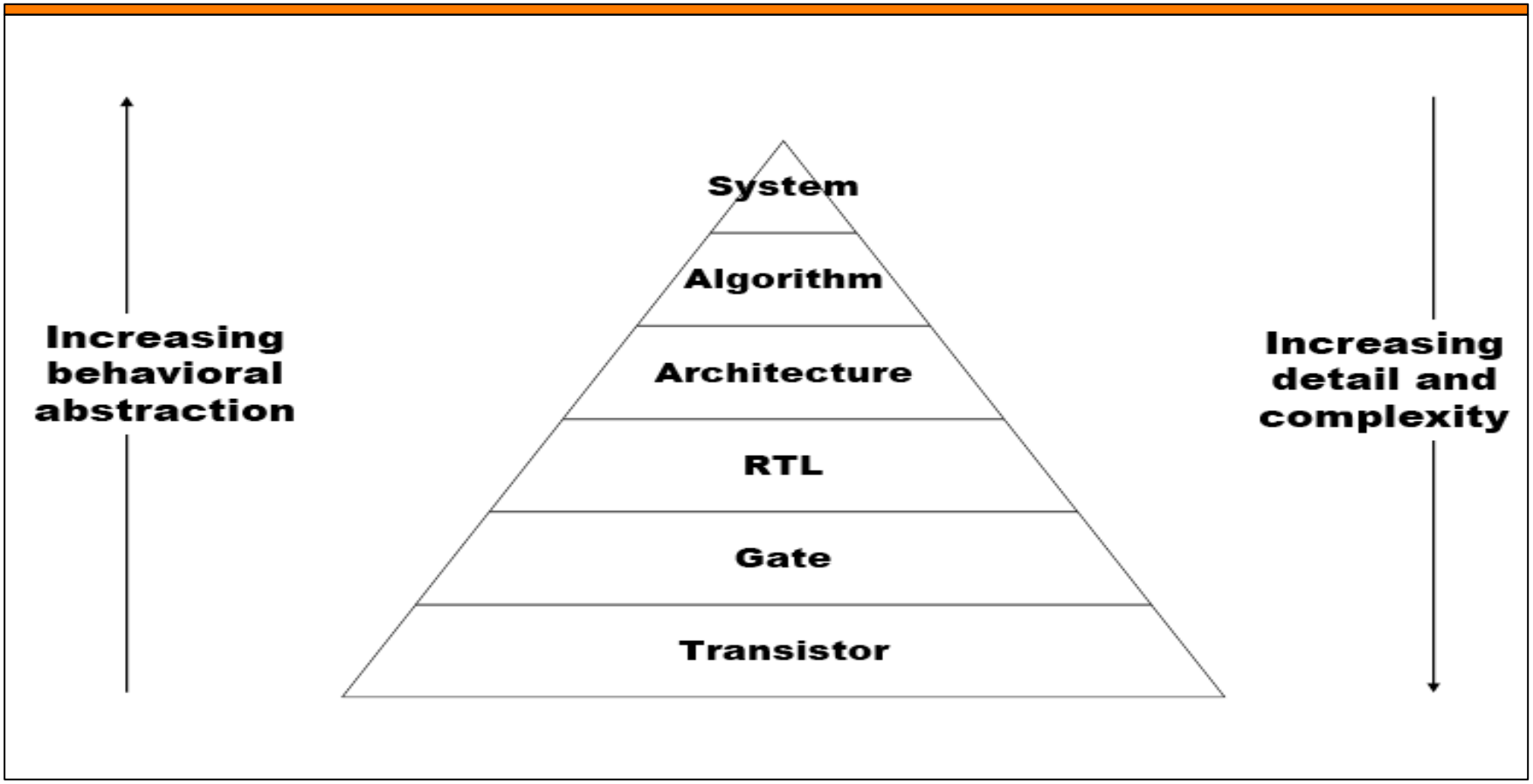
Back Annotation

- Output data from router added to schematic
- Simulations to estimate timing
  - identify critical paths
  - full logical simulation
  - full electrical simulation
- Leads to re-routing, or re-design



Summary

Top-down Design Methodology



Design Domains