

Chapter 1

Hardware design environments

1.1 DIGITAL SYSTEM DESIGN PROCESS

1.1.1 Design Automation

1.2 The Art of Modeling

1.3 HARDWARE DESCRIPTION LANGUAGES

1.3.1 A Language for Behavioral Descriptions

1.3.2 A Language for Describing Flow of Data

1.3.3 A Language for Describing Netlists

1.4 HARDWARE SIMULATION

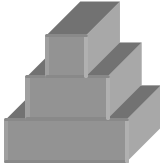
1.4.1 Oblivious Simulation

1.4.2 Event Driven Simulation

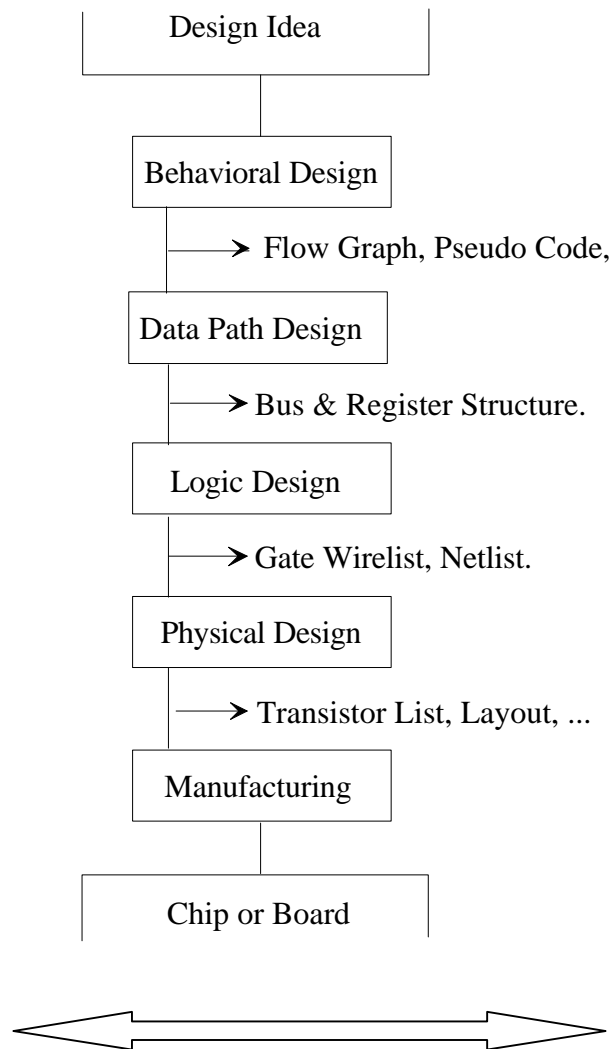
1.5 HARDWARE SYNTHESIS TEST APPLICATIONS

1.6 LEVELS OF ABSTRACTION

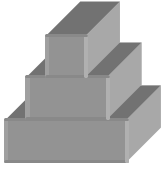
1.7 SUMMARY



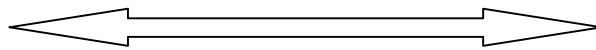
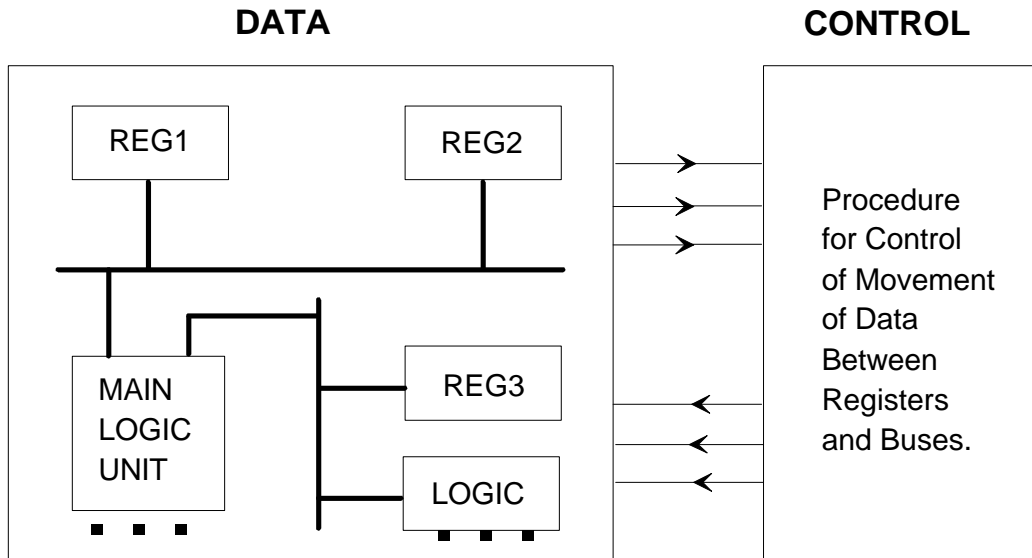
A digital system design process



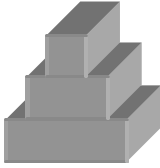
- Top-down design process
- Starting with a design idea
- Generating a chip or board



Result of the data path design phase.

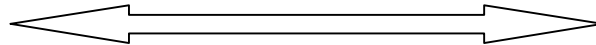


- Dataflow description
- Control Data partitioning

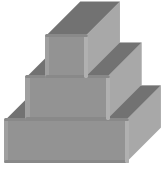


An ISPS example, a simple processor.

```
mark1 :=
  BEGIN
    ** memory.state **
    m[0:8191]<31:0>,
    ** processor.state **
    pi\present.instruction<15:0>'
    f\function<0:2> := pi<15:13>,
    s<0:12> := pi<12:0>,
    cr\control.register<12:0>,
    acc\accumulator<31:0>,
    ** instruction.execution ** {tc}
  MAIN i.cycle :=
    BEGIN
      pi = m[cr]<15:0> NEXT
      DECODE f =>
        BEGIN
          0\jmp      := cr = m[s],
          1\jrp      := cr = cr + m[s],
          2\ldn      := acc = - m[s],
          3\sto      := m[s] = acc,
          4:5\sub    := acc = acc - m[s],
          6\cmp      := IF acc LSS 0 => cr = cr + 1,
          7\stp      := STOP(),
        END NEXT
      cr = cr + 1 NEXT
      RESTART i.cycle
    END
```

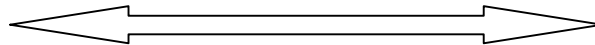


- Behavioral Example
- Only describing functionality

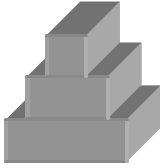


An AHPL example, a sequential multiplier.

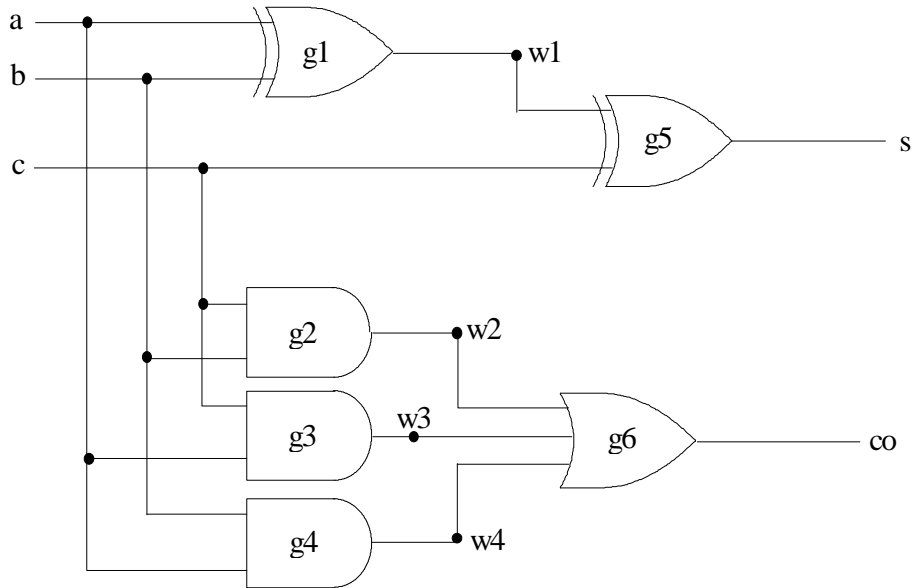
```
AHPLMODULE: multiplier.  
MEMORY: ac1[4]; ac2[4]; count[2]; extra[4]; busy.  
EXINPUTS: dataready.  
EXBUSES: inputbus[8].  
OUTPUTS: result[8]; done.  
CLUNITS: INC[2](count); ADD[5](extra; ac2);  
1   ac1 <= inputbus[0:5]; ac2 <= inputbus[4:7];  
    extra <= 4$0;  
    => ( ^dataready, dataready ) / (1, 2).  
2   busy <= \1\  
    => ( ^ac1[3], ac1[3] ) / (4, 3).  
3   extra <= ADD[1:4] (extra; ac2).  
4   extra, ac1 <= \0\  
    extra, ac1[0:2];  
    count <= INC(count);  
    => ( ^(&/count), (&/count) ) / (2, 5).  
5   result = extra, ac1; done = \1\  
    busy <= \0\  
    => (1).  
ENDSEQUENCE  
CONTROLRESET(1).  
END.
```



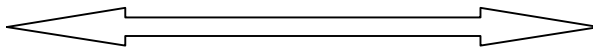
- Dataflow description
- Describing clock control timing
- AHPL, A Hardware Programming Language



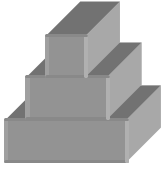
Full-adder, logical diagram and Verilog code.



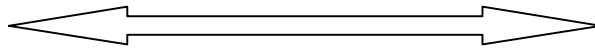
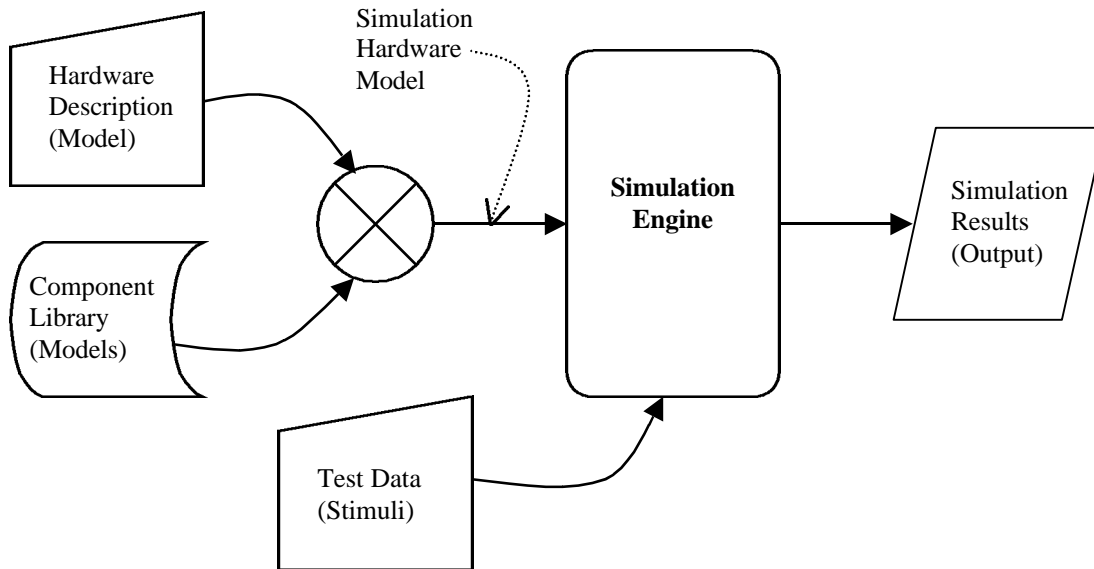
```
`timescale 1 ns / 1 ns
// A 6-gate full adder; this is a comment
module fulladder (s, co, a, b, c);
  // Port declarations
  output s, co;
  input a, b, c;
  // Intermediate wires
  wire w1, w2, w3, w4;
  // Netlist description
  xor #(16, 12) g1 (w1, a, b);
  xor #(16, 12) g5 (s, w1, c);
  and #(12, 10) g2 (w2, c, b);
  and #(12, 10) g3 (w3, c, a);
  and #(12, 10) g4 (w4, b, a);
  or #(12, 10) g6 (co, w2, w3, w4);
endmodule
```



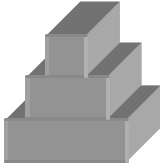
- Gate level structural description
- Describes gate level timing
- Graphical and language based descriptions



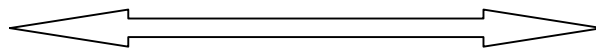
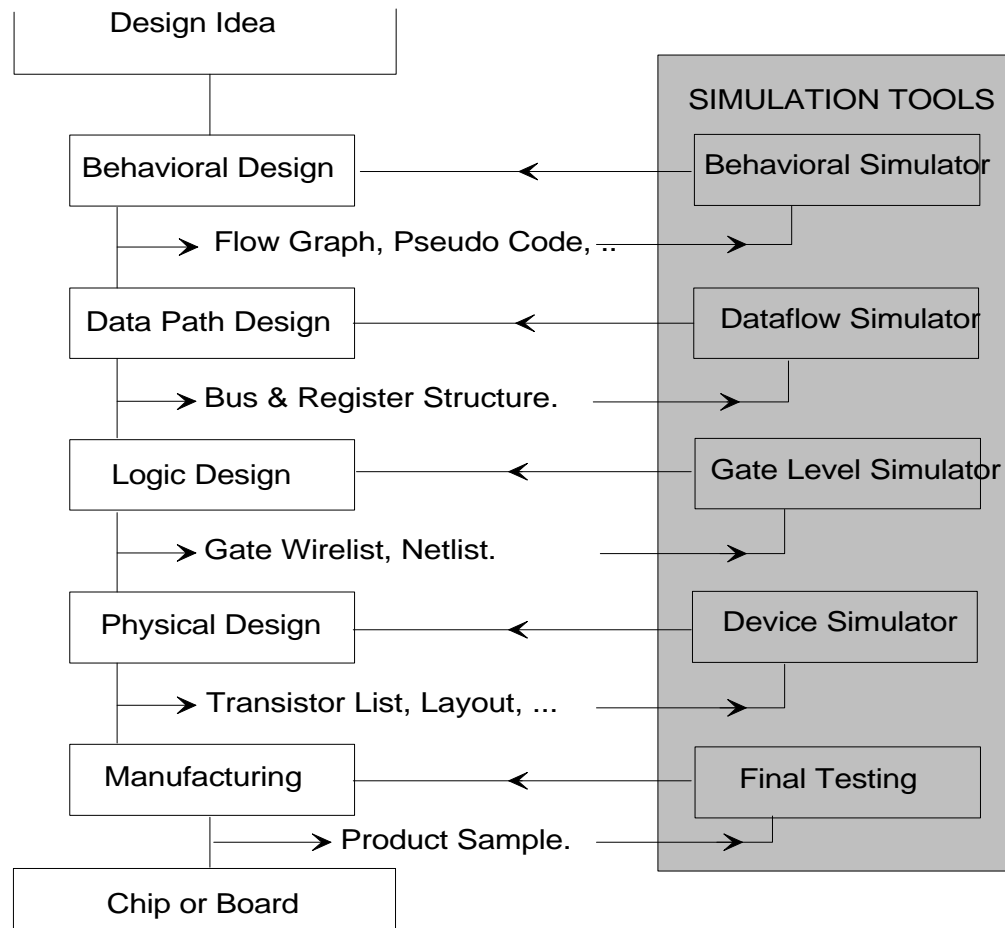
Hardware simulation.



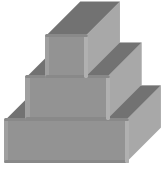
- Hardware simulation process
- Component models, unit model form hardware model
- Testbench may provide test data



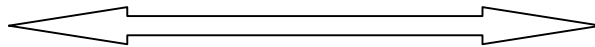
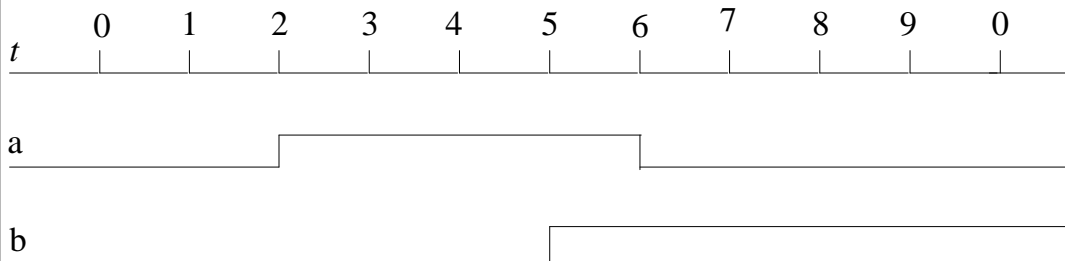
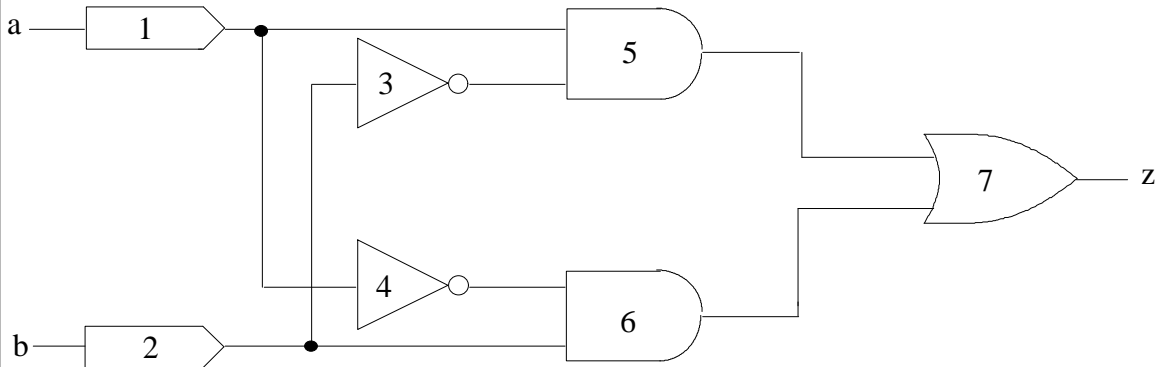
Verifying each design stage.



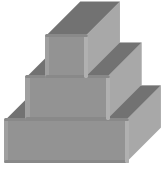
- Simulate at each step
- Simulate to verify translation into lower level
- Simulation cost increases at lower levels



Simulating an exclusive-OR

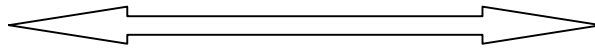


- Simulating an XOR
- Apply data at given time intervals or
- Apply data as events occur

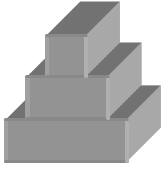


Oblivious simulation.

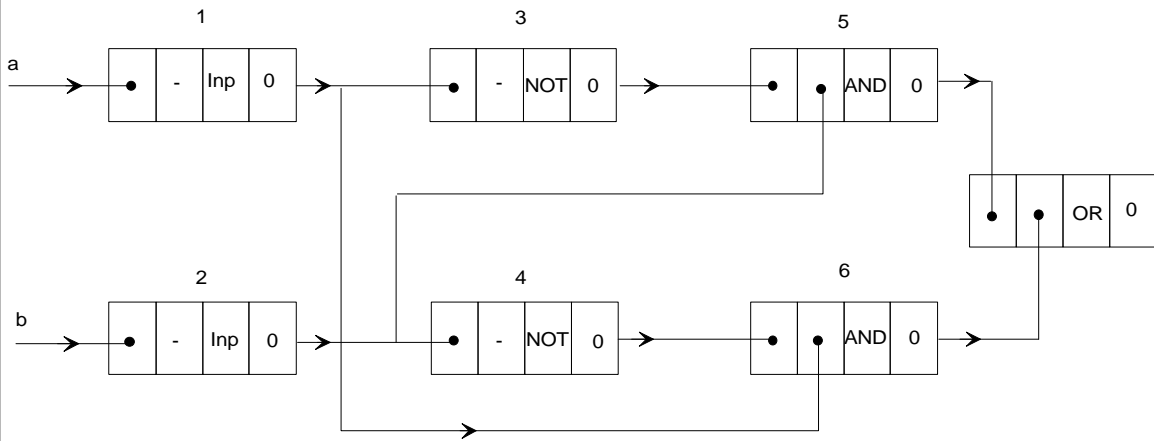
GATE	FUNCTION	INPUT 1	INPUT 2	VALUE
1	Input	a	--	0
2	Input	b	--	0
3	NOT	2	--	1
4	NOT	1	--	1
5	AND	1	3	0
6	AND	4	2	0
7	OR	5	6	0



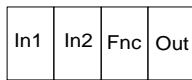
- Table representation
- Simulate until no changes are made
- Record values at table entries



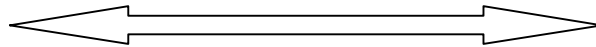
Event driven simulation.



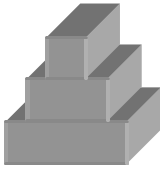
Legend:



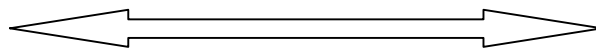
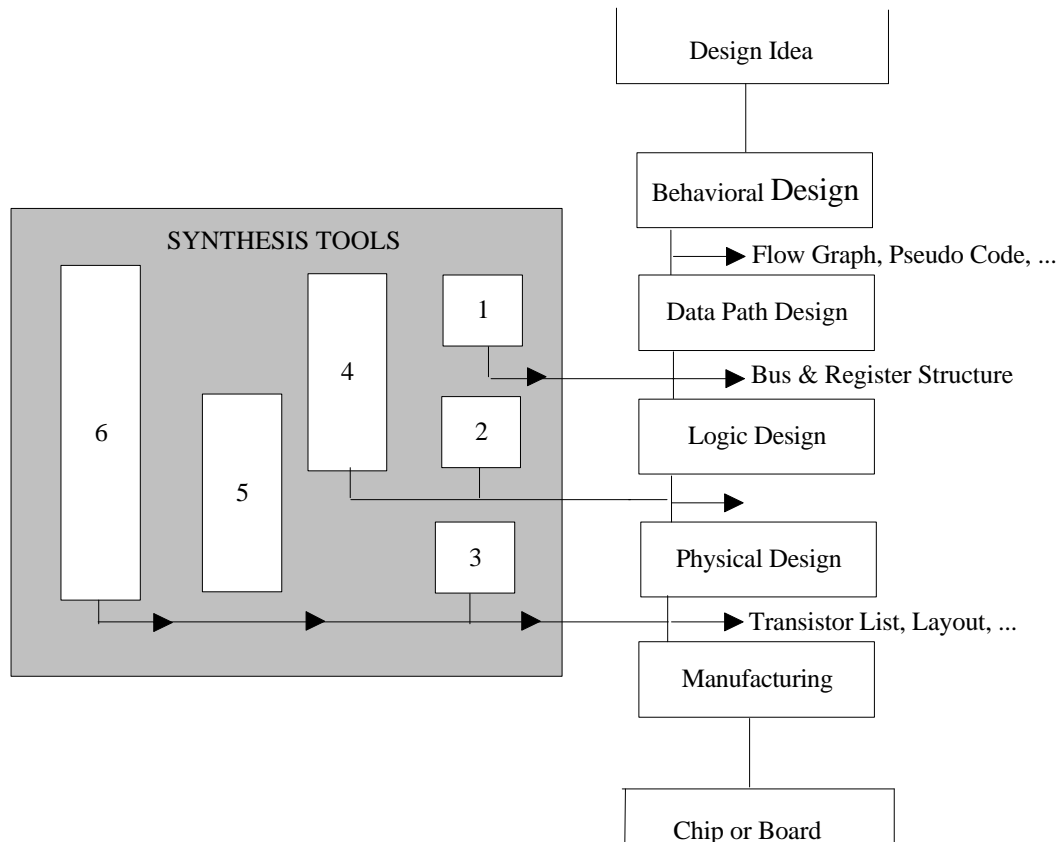
In1: Input 1; In2: Input2; Fnc: Function; Out: Output Value



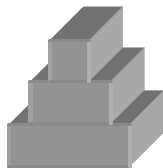
- Linked list representation
- Simulate links with input events
- Record values at node entries



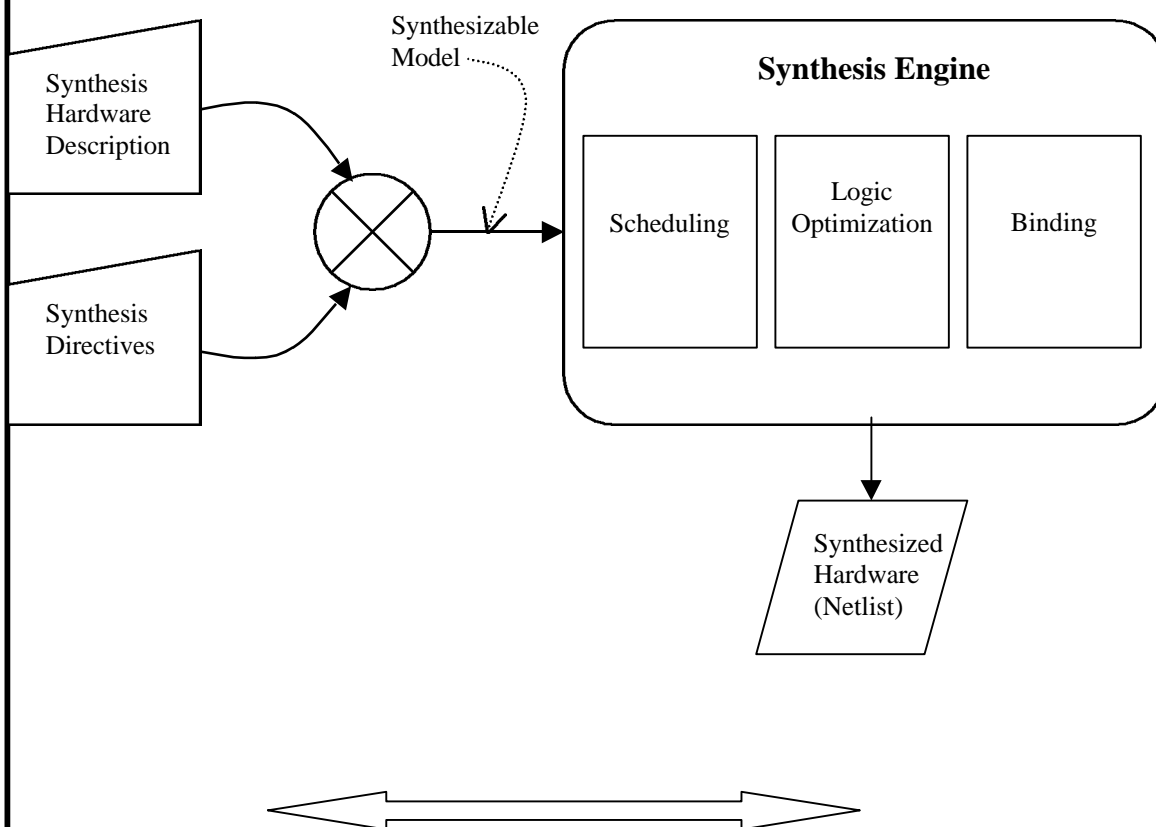
Categories of synthesis tools.



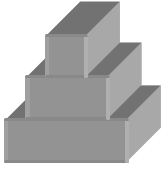
- Synthesis
- Transformation from one level to another
- Ideal is 6, most commercial tools are 2



Synthesis process.

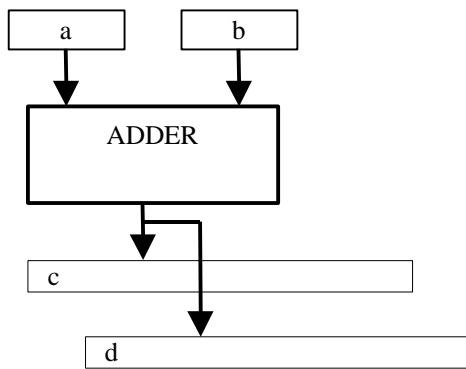


- Hardware description and directives are tool inputs
- Three synthesis stages
- Layout or netlist is generated

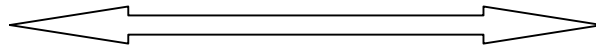
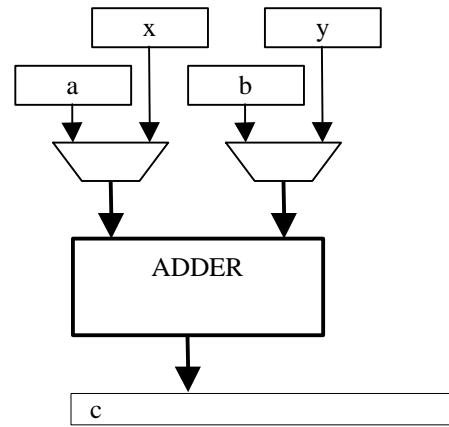


Resource sharing.

```
c <= a + b;  
d <= a + b;
```



```
c <= a + b;  
c <= x + y;
```



- Input description affects synthesis results
- Explicit specification of resource sharing
- Sharing without and with extra overhead