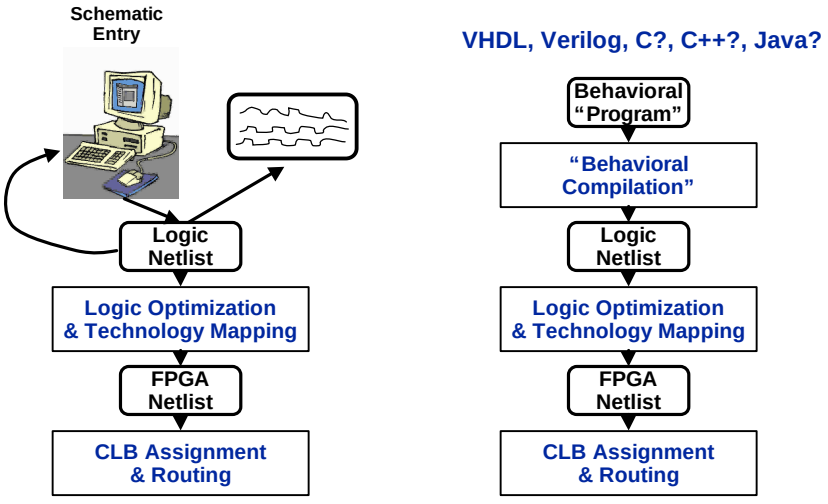
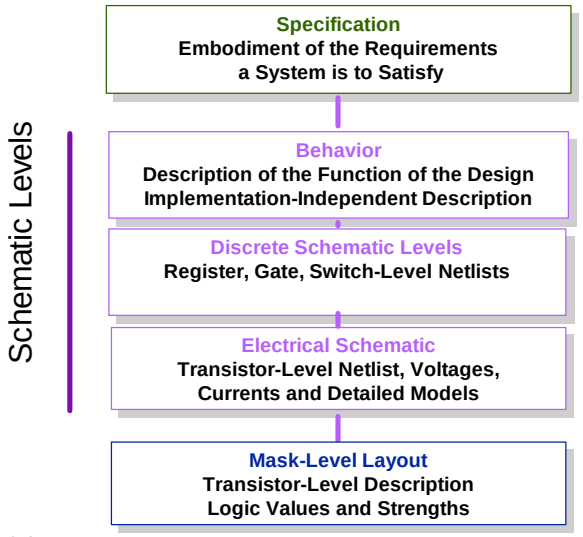
Outline

- Last time:
  - → Review State Tables & State Transition Diagrams
  - → Implementation Using D Flip-Flops
  - → Machine Equivalence
  - → Incompletely Specified Machines
  - → State Assignment & State Coding Schemes
  - → Design Example: Assign Codes to States
  - → Design Example: Implement Using D flip-flops
  - → Design Example: Implement Using T flip-flops
- This lecture:
  - → Introduction to VHDL

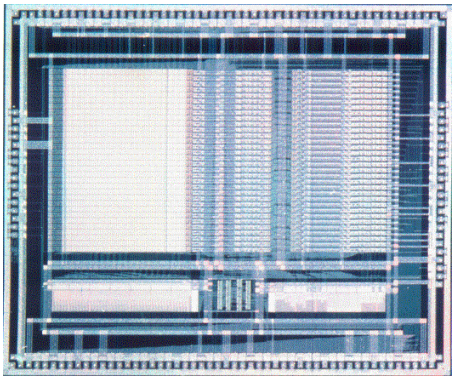
Overall Sequential Design Flow



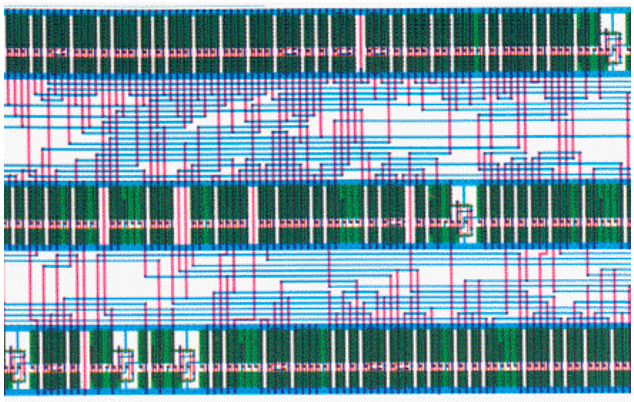
Common Representations Used in the Design Process



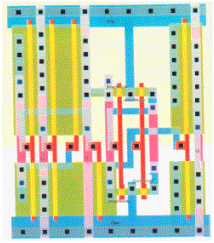
Structured Custom Chip Layout



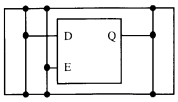
Standard Cell Layout



Layout Abstractions for Cell-Based Design

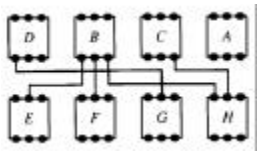


Mask-Level Layout

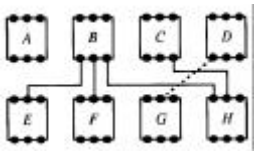


Cell Abstraction for Automatic Placement

Standard Cells vs. Gate Array



(a) Two tracks required and all connections routed.



(b) Shorter wire length but three tracks required.

In a Standard Cell design, an additional track could be added while in a Gate-Array, the designer is faced with extra wire length or no connection.

Sequential Control-Flow Model

- **Based on Von Neumann "fetch;execute;store" with single (serial control flow) or synchronized multiple (parallel control flow) thread(s) of control.**
- **Proposes an ordering of computation based on strict, temporal sequencing of operations, in line with above cycle.**
- **Not well suited to the description of hardware since arbitrary hardware rarely fits such a model of computation.**
- **Ability to combine several operations and treat them as a unit, or block (**begin-end, parbegin-parend, ...)
- **Ability to allow an operation to be executed in a loop.**

```
j=0;
while(j < 10){
      x[j]=y[j]+c;
      j=j+1;
}
```

```
for(i=0 to 9) {
    x[i]=y[i]+c;/* "forall" */
}
```

- **e.g. ISP, DDL/P(1973), Adlib/Sable(1980), AHPL(1973), SLANG(1982).**

Converting Procedural Descriptions to a Dataflow-Oriented Representation

- **Determine scope of variables and apply single-assignment in the scope**
- **Convert Complex data-structures into simple types**
- **Unroll loops with constant loop-counts (if appropriate)**
- **Perform simple syntactic optimizations:**
  - **Move operations out of loops where possible**
  - **Simplify complex expressions**
  - **Extract common sub-expressions**

e.g. CMUDA: Value-trace (VT)
HAL: CDFG
YSC: YIF

Complications to Dataflow Analysis*

- **Unrestricted goto statements (jumps)**
  - **Introduce overly-pessimistic dependencies during static analysis**
  - **Values of variables references in the statements following a label depend on the assignments to variables in the code sections that can jump to the label.**
- **Global variables**
  - **i.e. variables declared outside of any function and therefore able to be shared by all functions**
  - **Last-use cannot be determined without examining the entire program.**

*J. T. Deutsch, "Behavioral-Level Simulation and Synthesis of Digital Systems," UCB/ERL M83/47, August 1983

Complications to Dataflow Analysis

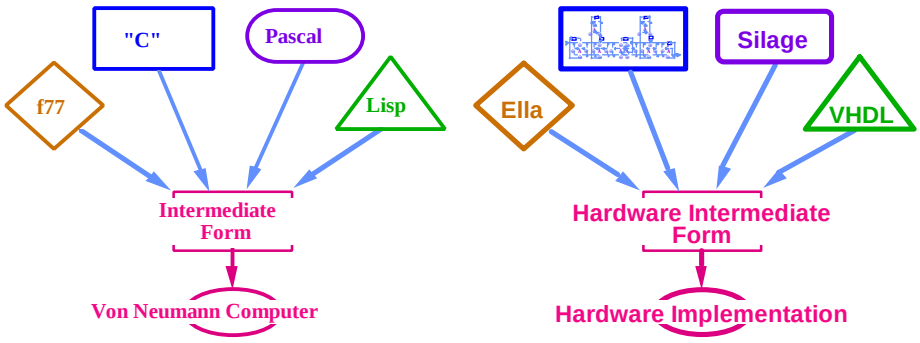
- **Static Variables**
  - **i.e. variables local to a function that retain their values between calls to the function**
  - **Create dependencies between the order of the calls to a function and, in effect, represent a communication path between all functions which call the function.**
- **Aliasing (often due to call-by-reference or call-by-name)**
  - **i.e. one or more name used to represent the same variable, which can result in hidden dependencies.**
  - **Several functions can be called with the name (or address) of the same object at the same time and multiple functions may try to modify the object concurrently. Unfortunately, this condition depends on run-time behavior.**

Complications to Dataflow Analysis

- **The Single-Assignment Rule**
  - **Variables used as "scratchpad" temporaries (assigned a value then re-assigned another value within the same section of program) create false dependencies between the old value and the new value of the variable.**
  - **Apply rule that a variable may only be written to once within a scope.**
    - **A := A+1 next A := A+1**
- **Applicative or Functional Languages**
  - **No goto's, global or static variables, call-by-reference or aliasing**
  - **Enforces the single-assignment rule**

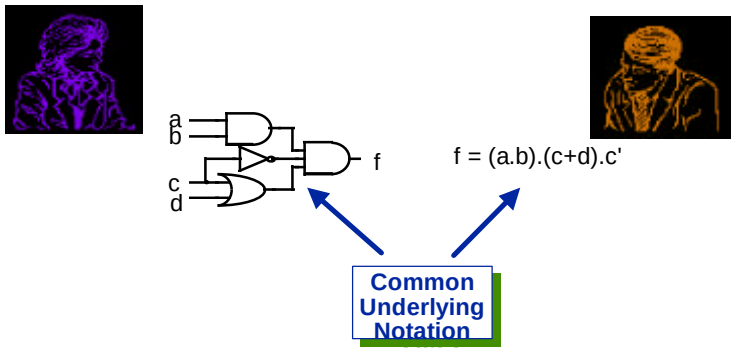
e.g. Silage, Ella

Languages versus Models: A Software Analogy



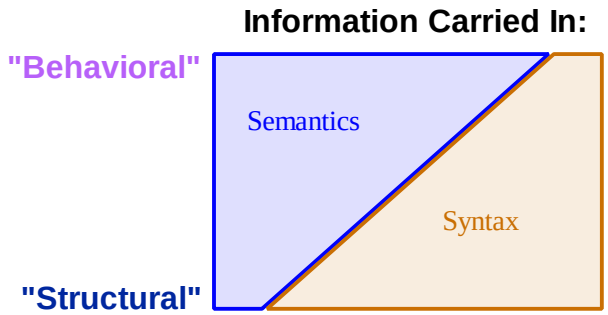
CS150 Newton/Pister 8.2.13

Behavior and Structure: Two Faces of the Same Coin



CS150 Newton/Pister 8.2.14

Behavior and Structure



State and Statements

- Consider symbolic state vector:

```
(serial
   (statement A)
   (statement B)
   (parallel
         (statement C)
      (serial
         (statement D)
         (statement E)
   )))
```

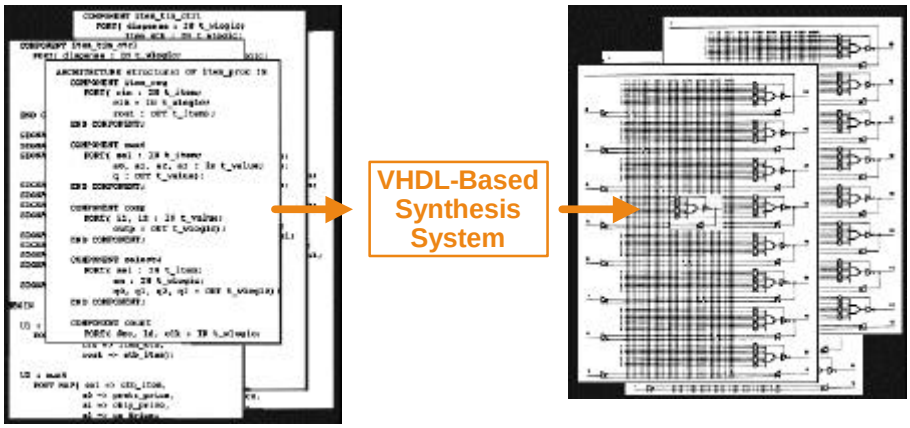
State Number	Statements Executed
1	A
2	B
3	C
3.1	D
3.2	E

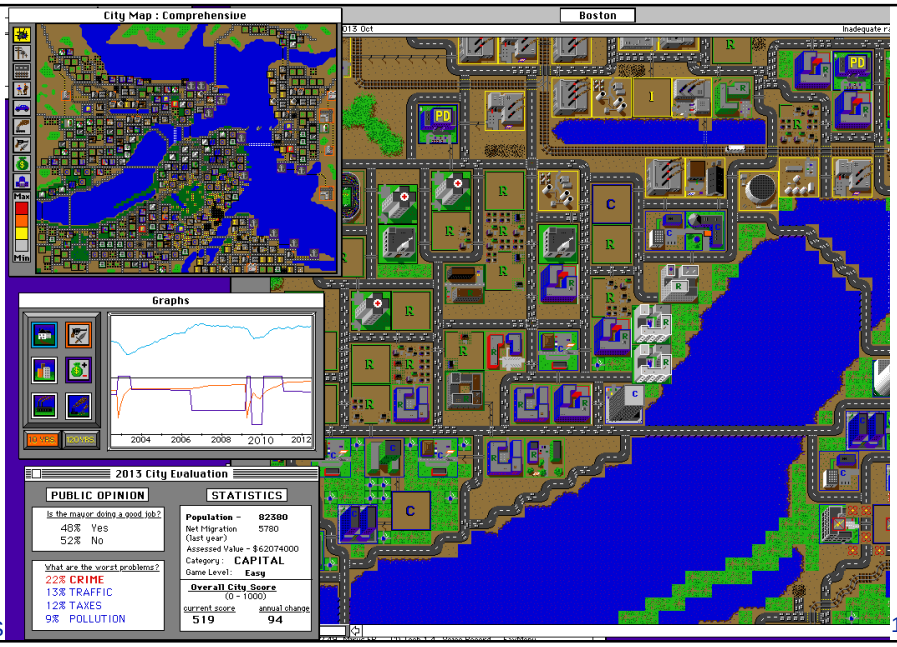
- Every time (serial (parallel (serial occurs, *pLevel++; stateVector[pLevel]* becomes current.

Data and Control

- Control is a necessary by-product of the synthesis process of mapping a behavior into the "real world" (i.e. to hardware and to "slow time")
  - **While it is a useful abstraction for certain classes of design, It should not be treated in a special way in the fundamental notation used for describing behavior/structure.**
  - **Consider the "Microscope Test": Control and data signals appear the same. The notion of control is a design-style oriented concept (and is therefore important) but control is not fundamentally different from other artifacts of the description (e.g. "datapath").**
- Disclaimer: This is the view of this presenter but this view is not held today by the majority of synthesis researchers (e.g. VT, DDL, VHDL).

VHDL: The “nroff/latex” of Design

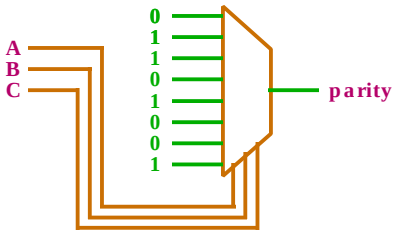




CS 19

3-Bit Parity Function: "Control-Oriented"

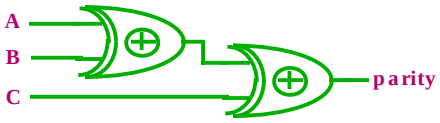
```
if A = '1'  then
  if B = '1' then
    if C = '1' then parity <= '1';
    else parity <= '0';
    endif;
  else
    if C = '1' then parity <= '0';
    else parity <= '1';
    endif;
  end if;
else
  if B = '1' then
    if C = '1' then parity <= '0';
    else parity <= '1';
    endif;
  else
    if C = '1' then parity <= '1';
    else parity <= '0';
    endif;
  end if;
end if;
```



CS150 Newton/Pister 8.2.20

3-Bit Parity Function: "Dataflow-Oriented"

parity <= ((A xor B) xor C);



3-bit Parity Function:Possible VHDL Implementation

(Adapted from D. R. Coelho, "The VHDL Handbook," Kluwer, 1989)

```
ENTITY parityFunction IS
    PORT( A, B, C : IN t_wlogic; parity : OUT t_wlogic )
END parityFunction;
ARCHITECTURE full OF parityFunction IS
BEGIN
    PROCESS (A, B, C)
        VARIABLE count : integer;
        BEGIN
            count := 0;
            IF A = '1' THEN count := count +1; END IF;
            IF B = '1' THEN count := count +1; END IF;
            IF C = '1' THEN count := count +1; END IF;
            IF (count MOD 2) = 0 THEN
                OUT <= '0';
            ELSE
                OUT <= '1';
            END IF;
    END PROCESS;
END full;
```

"I synthesize from C" or "I synthesize from VHDL"

- No you don't!
  (with one known exception [AT&T Cones])

  OR

- If you do, you're probably being very silly!

Remember:

A subset of a language is a different language

A sequential language has sequential semantics

```
a = b+c;
d = e+f;
```

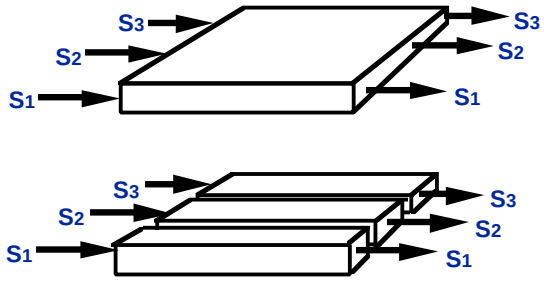
```
a = 1;
xp = &a;
yp = xp+1-3+2;
*xp = *xp+1;
*yp = *yp+1;
print( *yp );
```


Representing Time for Behavioral Description

- **Must distinguish *sequence* (causality) vs. *passage of time***
- **Continuous time variable:**
  A *after* B
  A, B *independent*
- **"Quantized" time variable:**
  A *after* B
  A, B *independent*
  A, B *at the same time*

  where the meaning of "at the same time" is across an interval (must be valid at the end of the interval).

Why Should Time be Discrete?



Representing Time for Behavioral Descriptions

- Such an interval is called a "*timeslot* "; analogous to a "control state" in control-flow-based synthesis.
- Timeslot may have many models associated with it: unspecified, fixed, interval (EDIF miNoMax), distribution.
- System temporal behavior is relative; external inputs map behavior to "slow time."

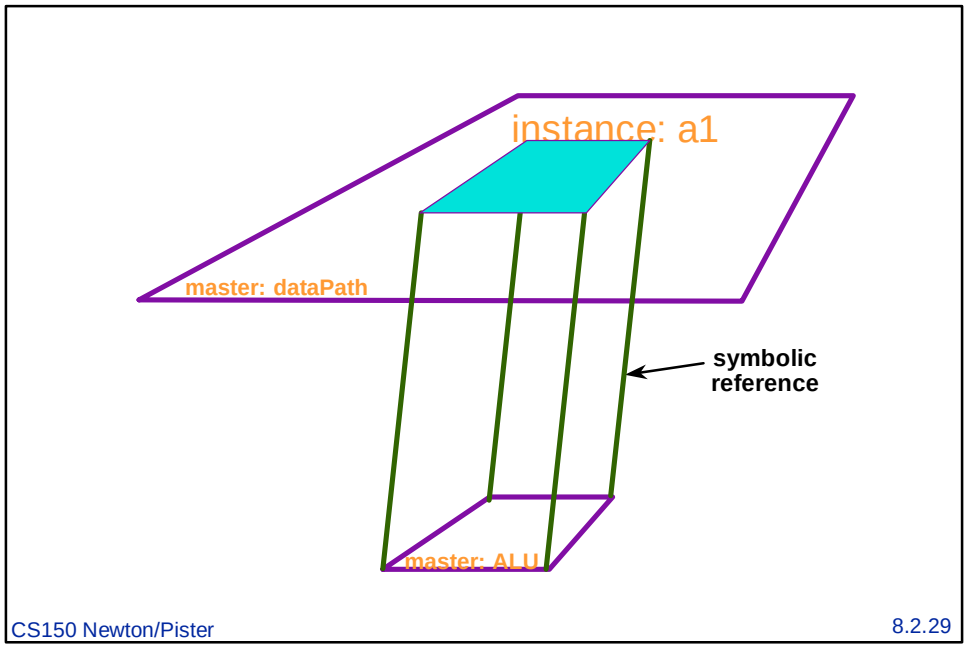
Encoding Information in Time & Space

```
PROCESS (L(7))
    VARIABLE  V_TRAN_1 : INTEGER;
    VARIABLE  V_DEC_25 : INTEGER;
BEGIN
    IF REG000 <= V_N THEN
      R_SUM <=  0.0E+00;
      REG002 <= 1;
      V_TRAN_1 := REG000 - 1;
      R_OPT_T <= V_TRAN_1;
      V_DEC_25 := ABS (V_TRAN_1);
      REG003 <= V_DEC_25 * 81;
      L(8) <= NOT(L(8));
    ELSE
      IF R_RESID < V_EPSLON THEN
        L(5) <= NOT(L(5));
      ELSE
        REG001 <= REG001 + 1;
        L(6) <= NOT(L(6));
      END IF;
    END IF;
END PROCESS;
```

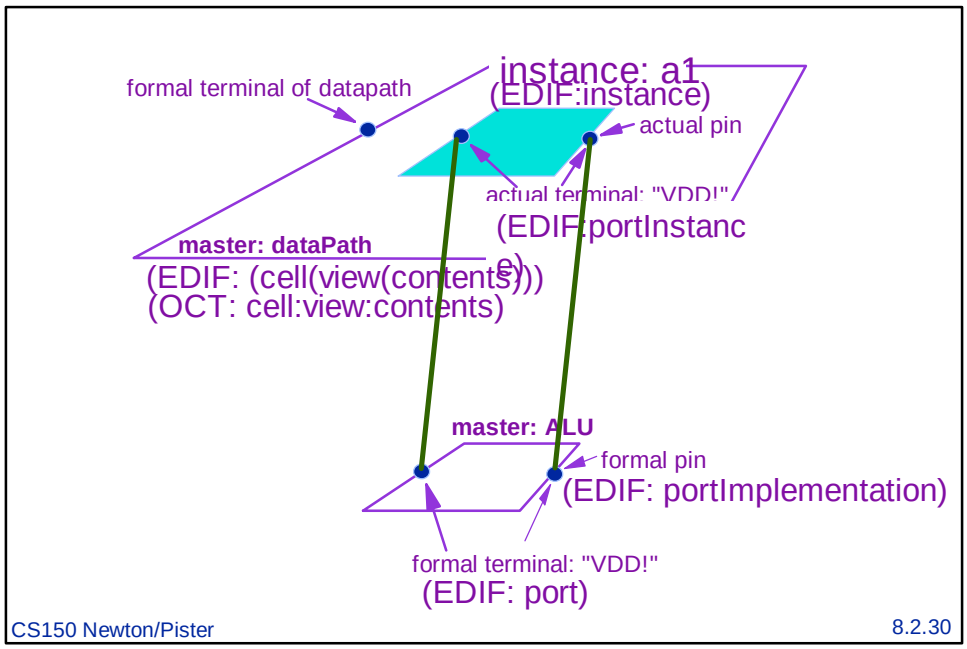
- In Most HDLs, "wires" are declared but the passage of time is embedded in the control structures.
- We are caught up (once again!) with imperative, sequential thinking and a Von Neumann model.
- We need a way of capturing both temporal and spatial encoding in a single, unified mathematical model.
- Use a type mechanism: "τ-types"

Design Representation

- Some variation of entity-relationship model most common today.
- Data model usually based on existing interchange format or design language (e.g. EDIF, VHDL).
- Open Issue: Forced Consistency (conventional database approach) versus Periodic Check (C "Lint" approach).
- Integration Environment provides *lingua franca* for design representation. Useful side-effect of standards meetings is a common understanding of terms and data objects.



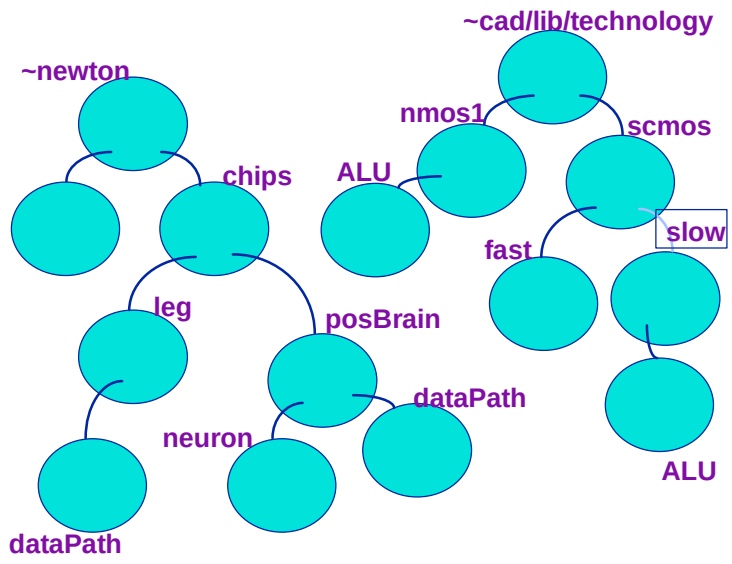
CS150 Newton/Pister 8.2.29

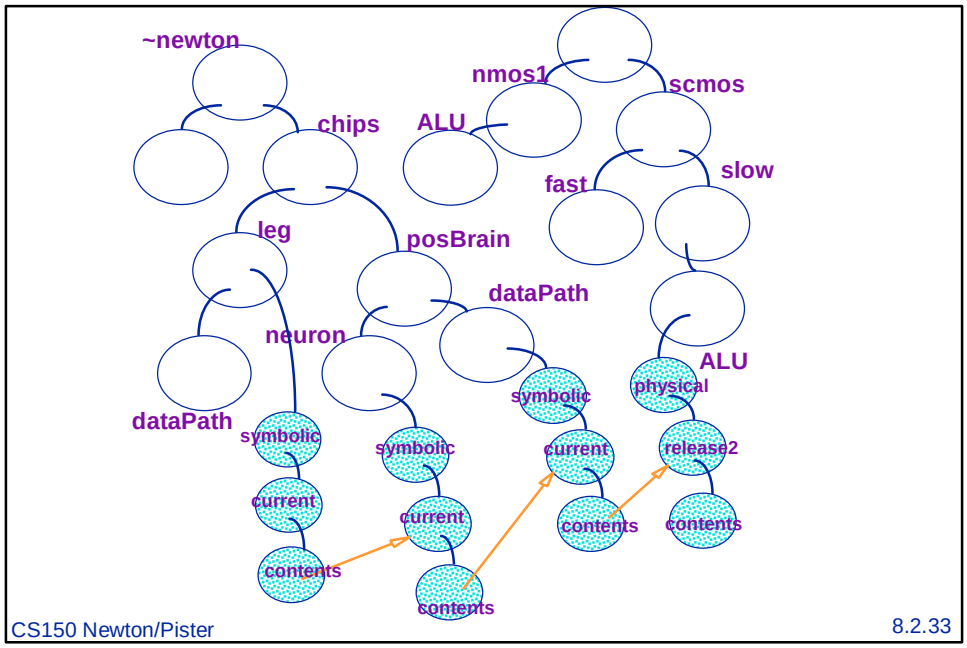


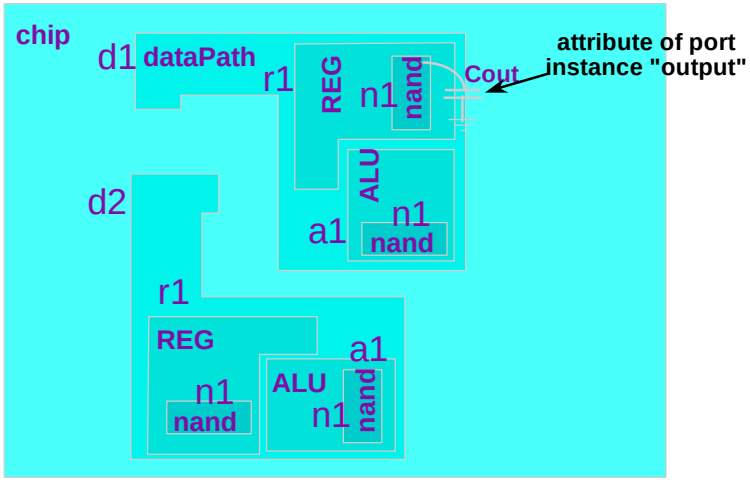
CS150 Newton/Pister 8.2.30

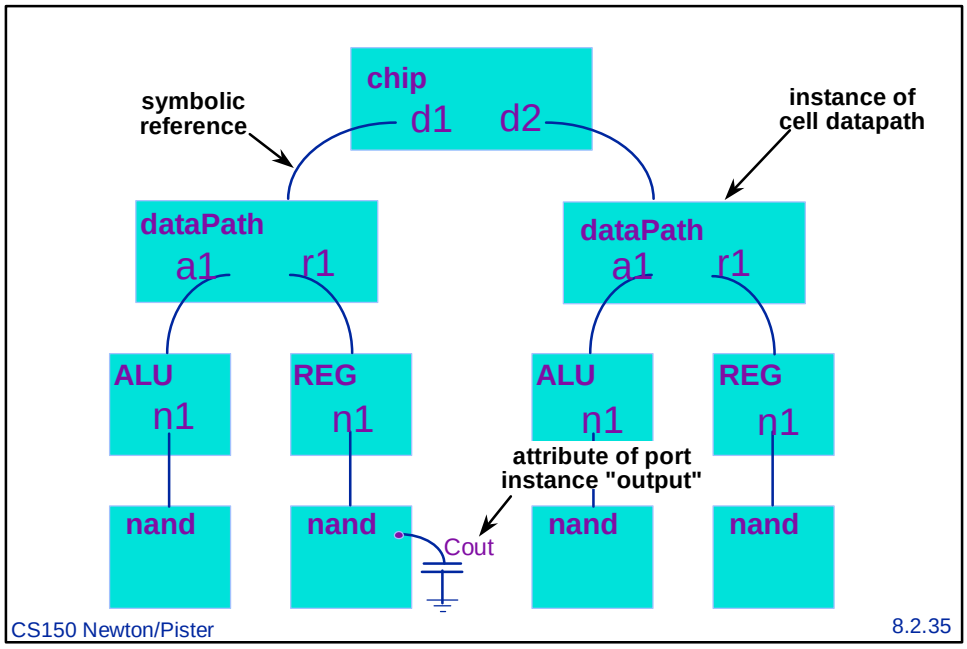
What's in a Name?

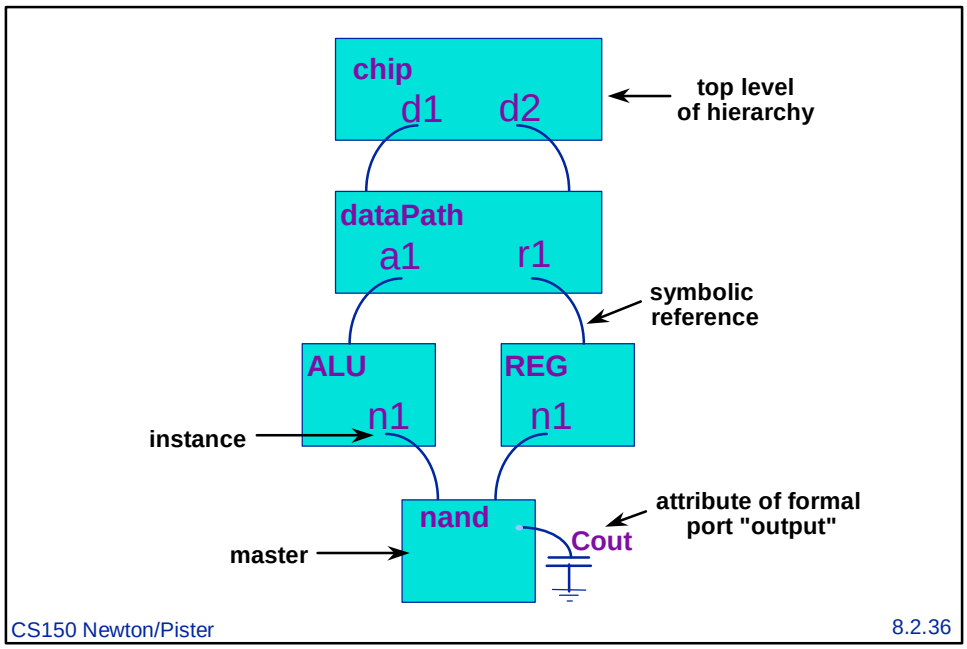
- Just about Everything!
- Efficient name resolution - resolving references to design objects - is one of the most important, "undecided" research problems.
- Strongly related to multiprocessor distributed cache consistency problem, distributed file system problem, general distributed data management problem.
- Ultimate issue is efficient pruning of "global search." (replication of read-only data, use of "hints," management of domains and dynamic data migration are all important.)




~newton
chips
nmos1
ALU
scmos
fast
slow
leg
posBrain
dataPath
neuron
ALU
physical
symbolic
dataPath
symbolic
symbolic
current
release2
current
current
contents
contents
contents
contents


chip
d1
dataPath
r1
REG
n1
nand
Cout
attribute of port
instance "output"
ALU
d2
a1
n1
nand
r1
REG
a1
ALU
n1
nand
n1
nand


chip
d1
d2
symbolic
reference
instance of
cell datapath
dataPath
a1
r1
dataPath
a1
r1
ALU
n1
REG
n1
ALU
n1
REG
n1
attribute of port
instance "output"
nand
nand
Cout
nand
nand
CS150 Newton/Pister
8.2.35


chip
d1
d2
top level
of hierarchy
dataPath
a1
r1
symbolic
reference
ALU
n1
REG
n1
instance
nand
attribute of formal
port "output"
master
Cout
CS150 Newton/Pister
8.2.36