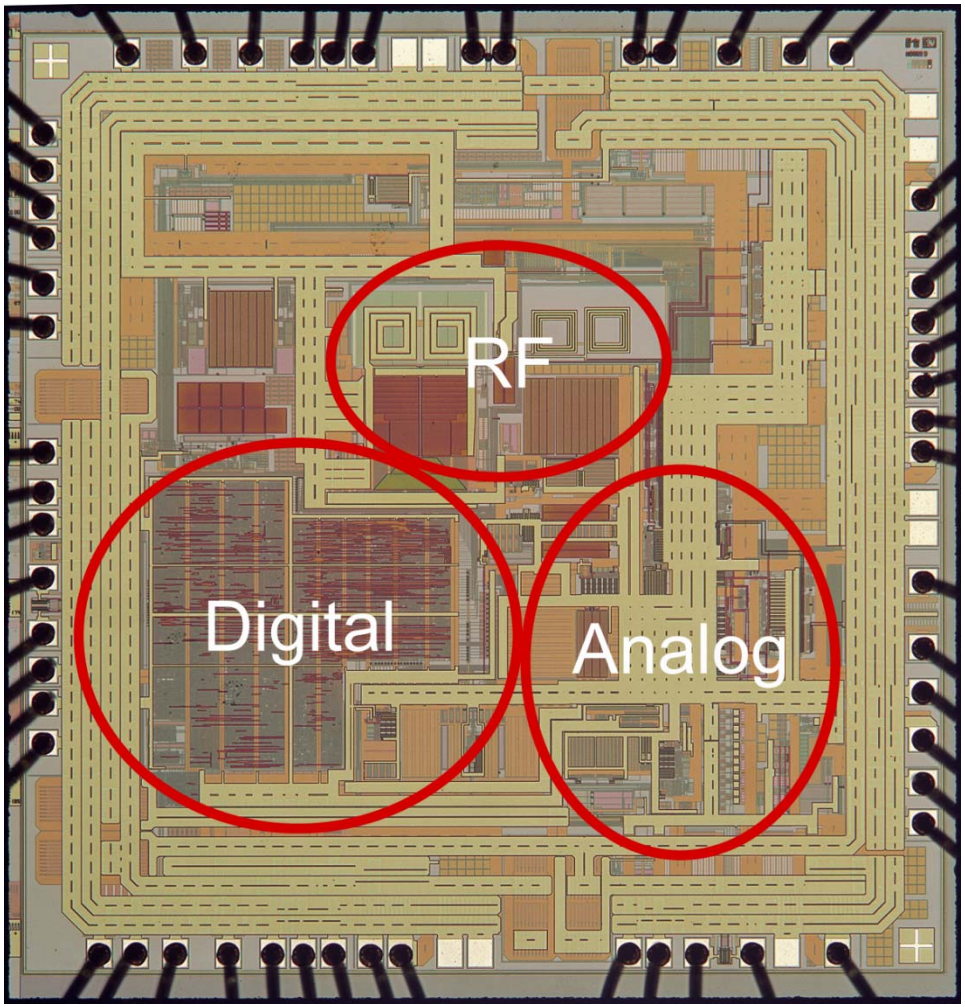
Short Course On
Phase-Locked Loops and Their Applications
Day 3, PM Lecture

Behavioral Simulation Exercises

Michael H Perrott

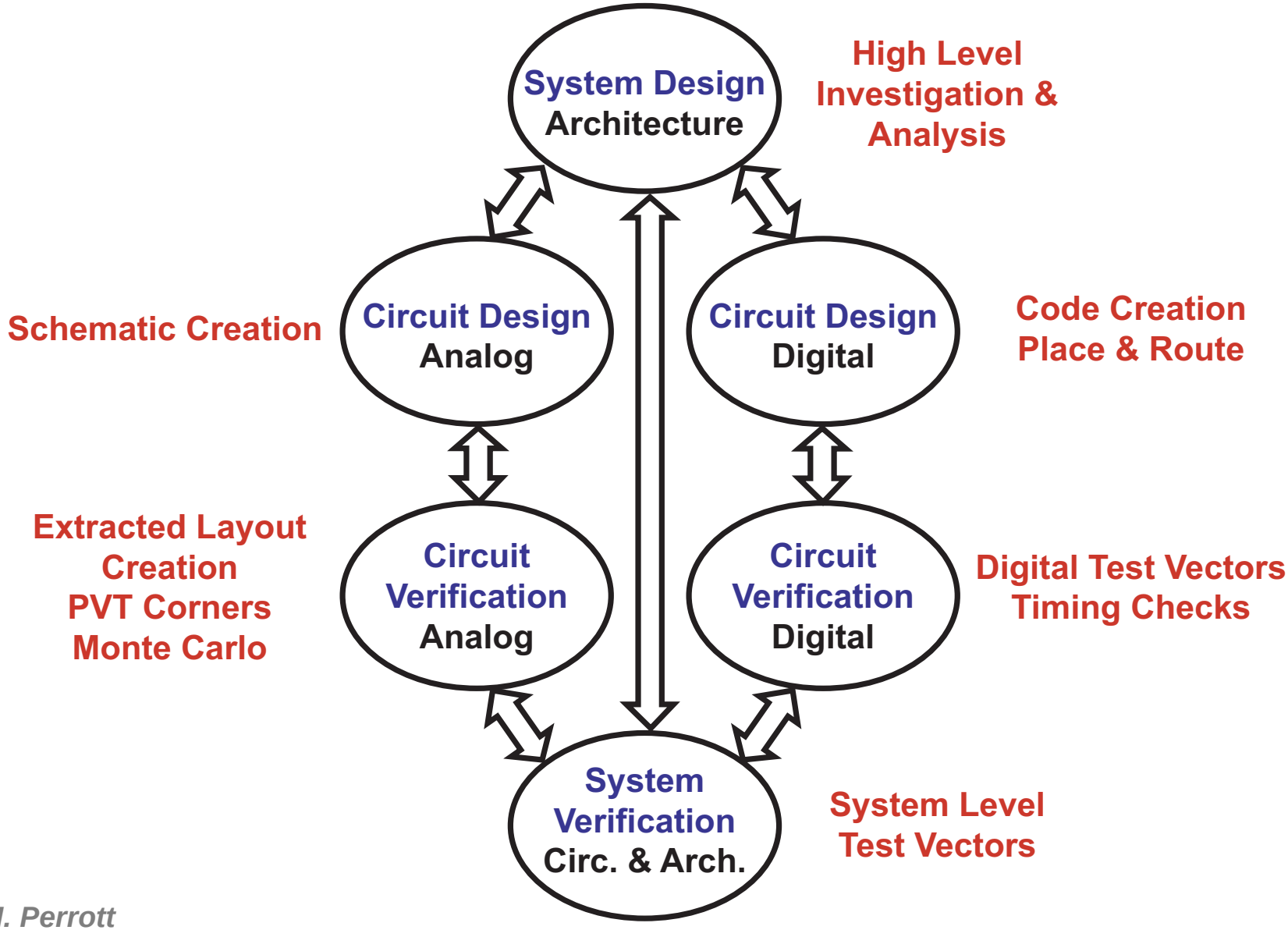
August 13, 2008

A Modern "Analog" Custom IC

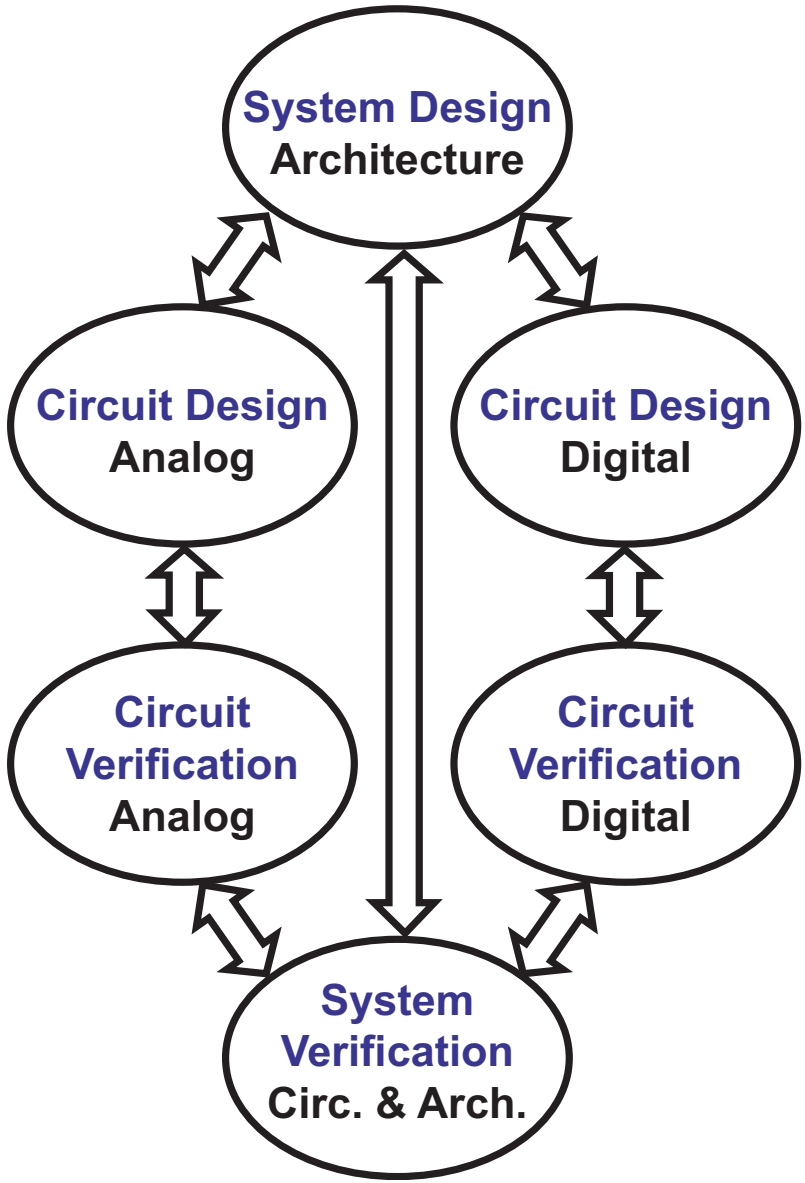


- A 2.5 Gb/s CDR for high speed links
  - *Analog* amplification and phase sensing
  - *Digital* filtering and calibration
  - *RF* clock generation (2.5 GHz)
- How do we design such chips?

Simplified View of a Top Down, Mixed-Signal Design Flow



The Many Simulation Tools Involved …



Popular Tools

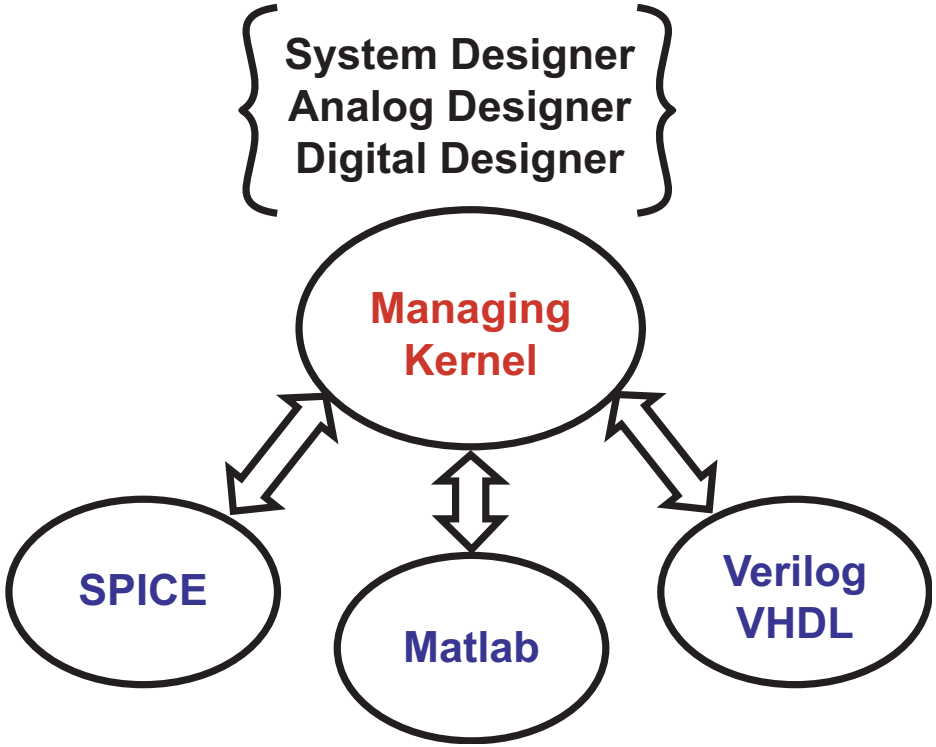
Matlab/Simulink
ADS

SPICE Verilog/VHDL

Verilog AMS
AMS Designer

????????

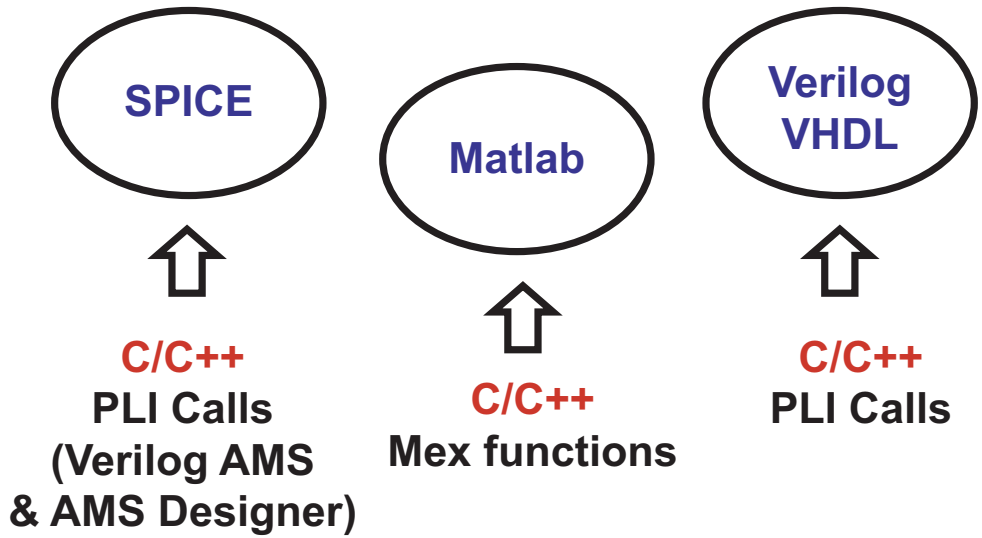
Goal: Create a Universal Simulator



- The challenges of developing the Managing Kernel
  - Difficult to match up simulator signals at their boundaries without overly complicating the life of the user
  - Difficult to maintain fast simulation speed
  - Difficult to retrain designers on a new tool with a new flow

A Different Approach

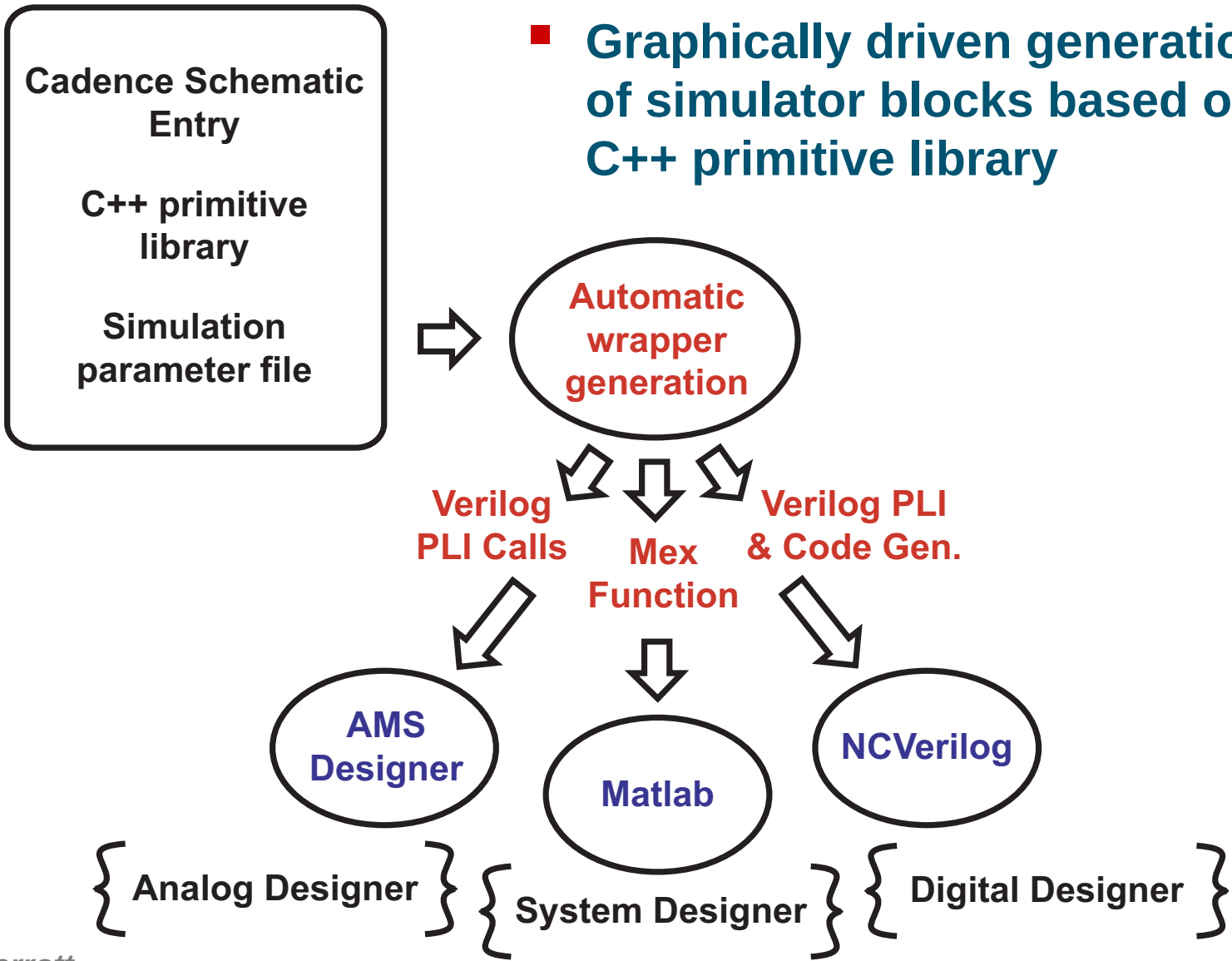
- Look for commonality among simulators to allow "universal" simulation models to be used
  - C++ provides one such hook
- Allow designers to use their tool of choice while sharing "universal" simulation models



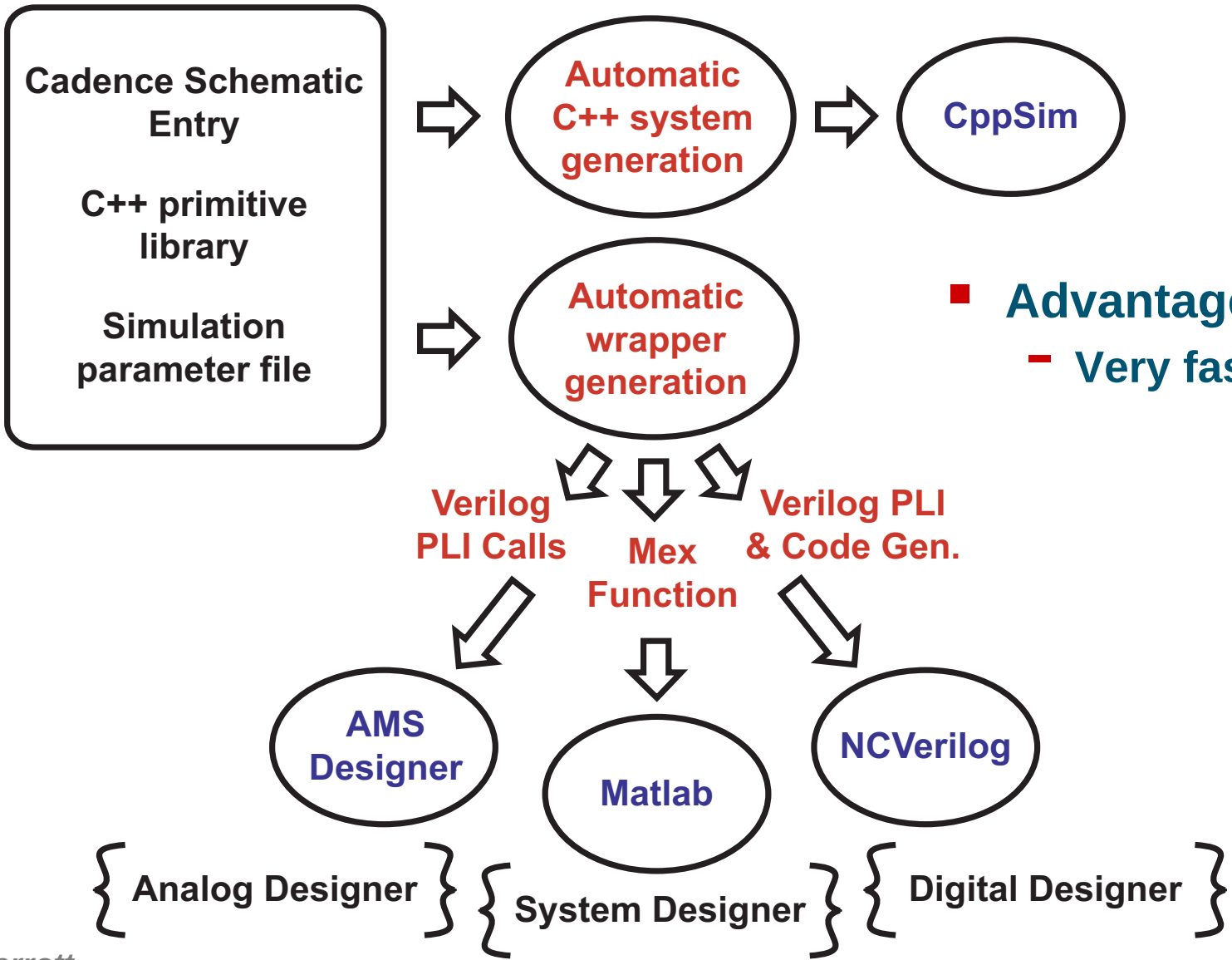
Key idea: bootstrap into existing simulators

The VppSim Simulator Greatly Simplifies This Process

- **Graphically driven generation of simulator blocks based on a C++ primitive library**

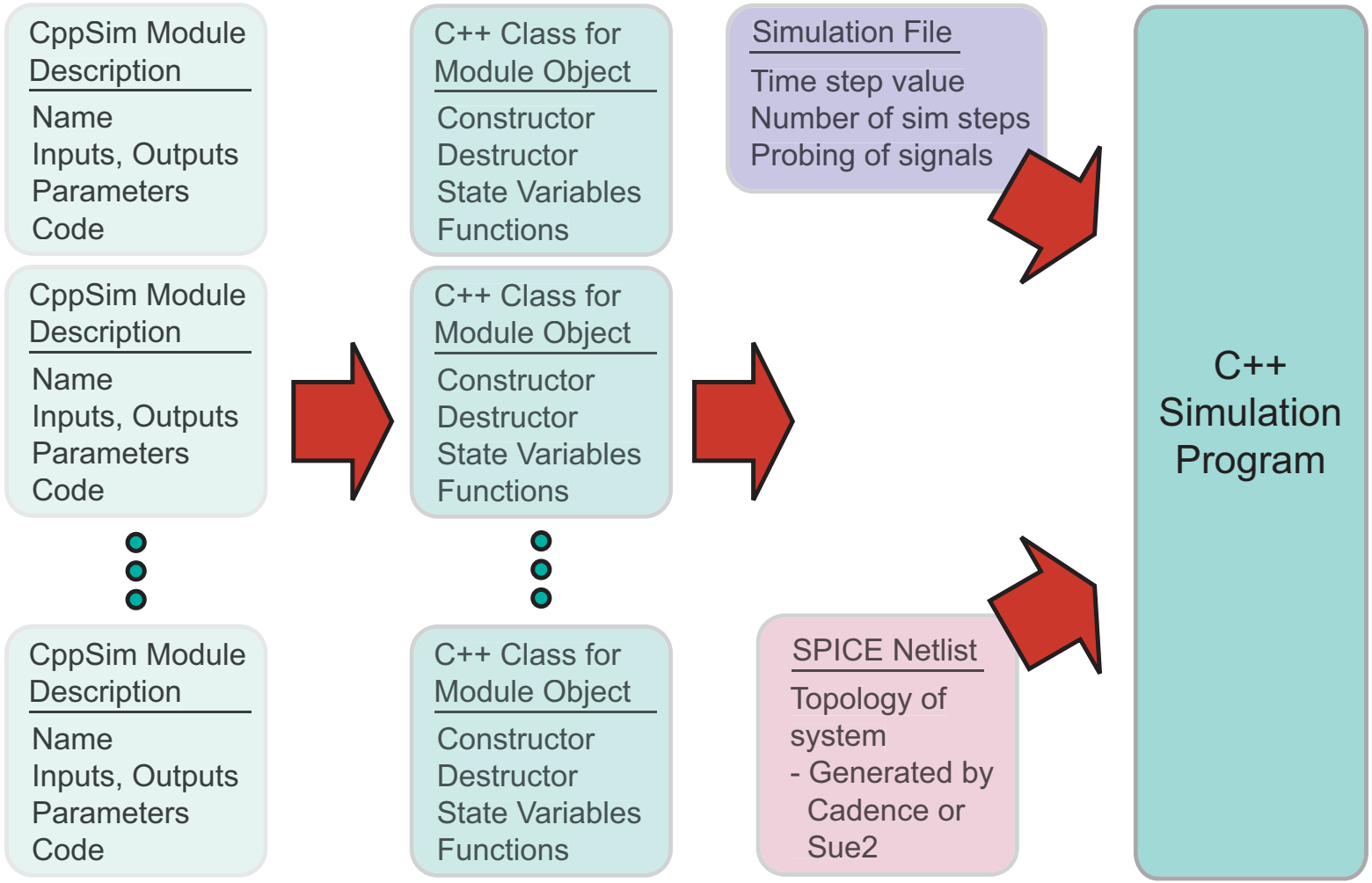


C++ Code Can Also Be Directly Simulated

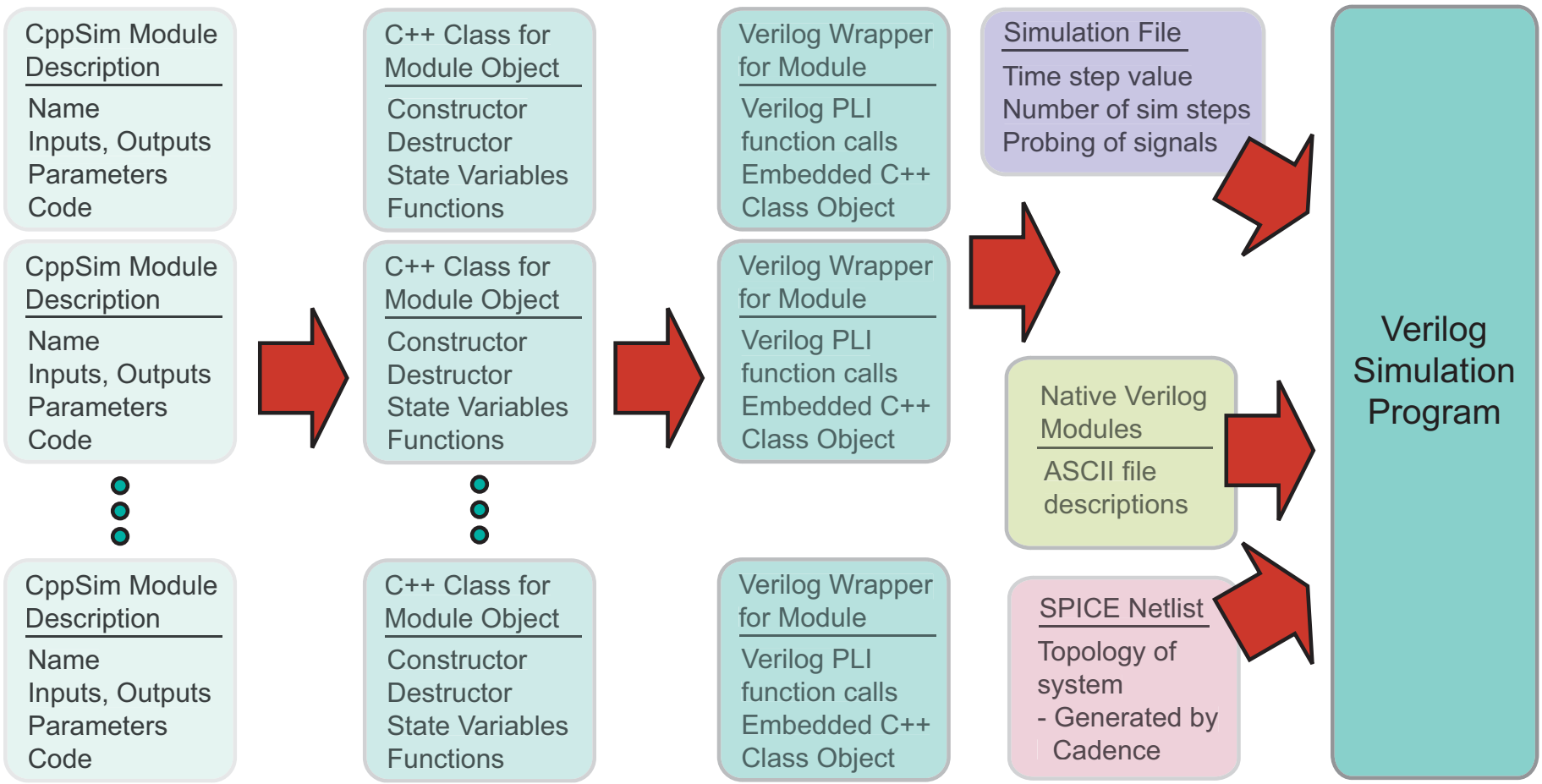


- **Advantage:**
  - **Very fast!**

CppSim Compiler

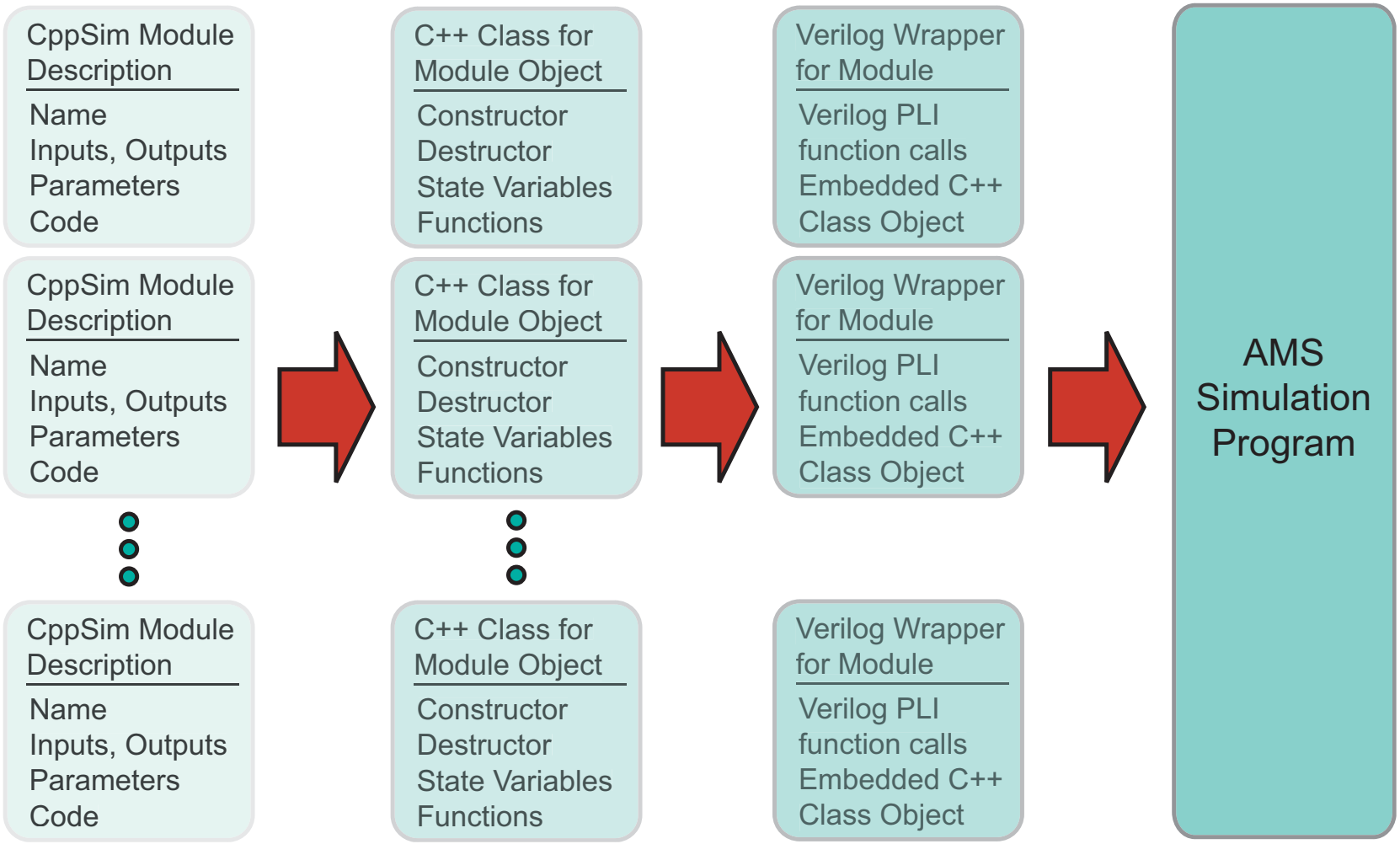


VppSim Compiler (for NCVerilog)



All CppSim modules supported (including vector signals)
Same timing approach for module execution as CppSim

VppSim Modules for AMS

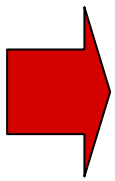


Not all CppSim modules are supported (i.e., no vector signals)
Different timing approach for module execution

Example with Double Signals

CppSim module

```
module: leadlagfilter
parameters: double fz, double fp,
                  double gain
inputs: double in
outputs:  double out
static_variables:
classes: Filter filt("1+1/(2*pi*fz)s",
             "C3*s + C3/(2*pi*fp)*s^2",
             "C3,fz,fp,Ts",1/gain,fz,fp,Ts);
init:
code:
filt.inp(in);
out = filt.out;
```



VppSim module

```
////// Auto-generated from CppSim module //////
module leadlagfilter(in, out);
  parameter fz = 0.00000000e+00;
  parameter fp = 0.00000000e+00;
  parameter gain = 0.00000000e+00;
  input in;
  output out;

  wreal in;
  real in_rv;
  wreal out;
  real out_rv;

  assign out = out_rv;

  initial begin
      assign in_rv = in;
    end

  always begin
      #1
      $leadlagfilter_cpp(in_rv,out_rv,fz,fp,gain);
    end
endmodule
```

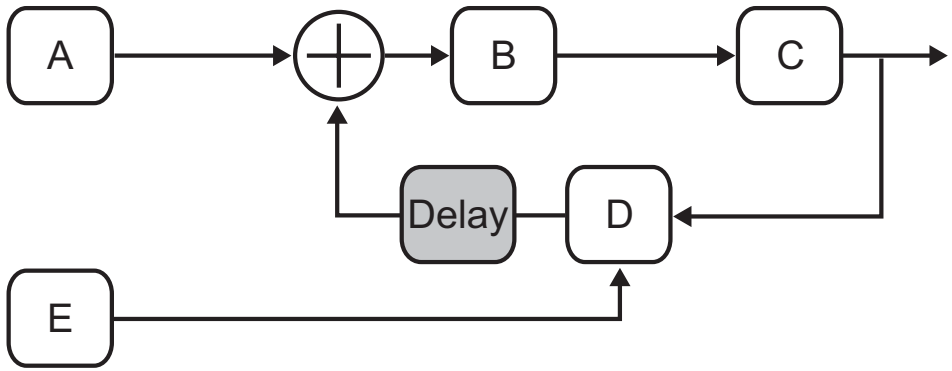
The Issue of Timing/Ordering

(1)

Execution Order

A
E
B
C
D

Best Execution Order

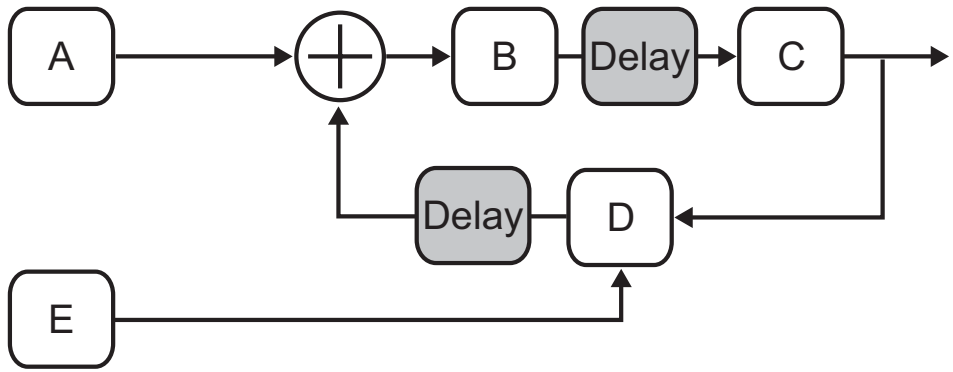


(2)

Execution Order

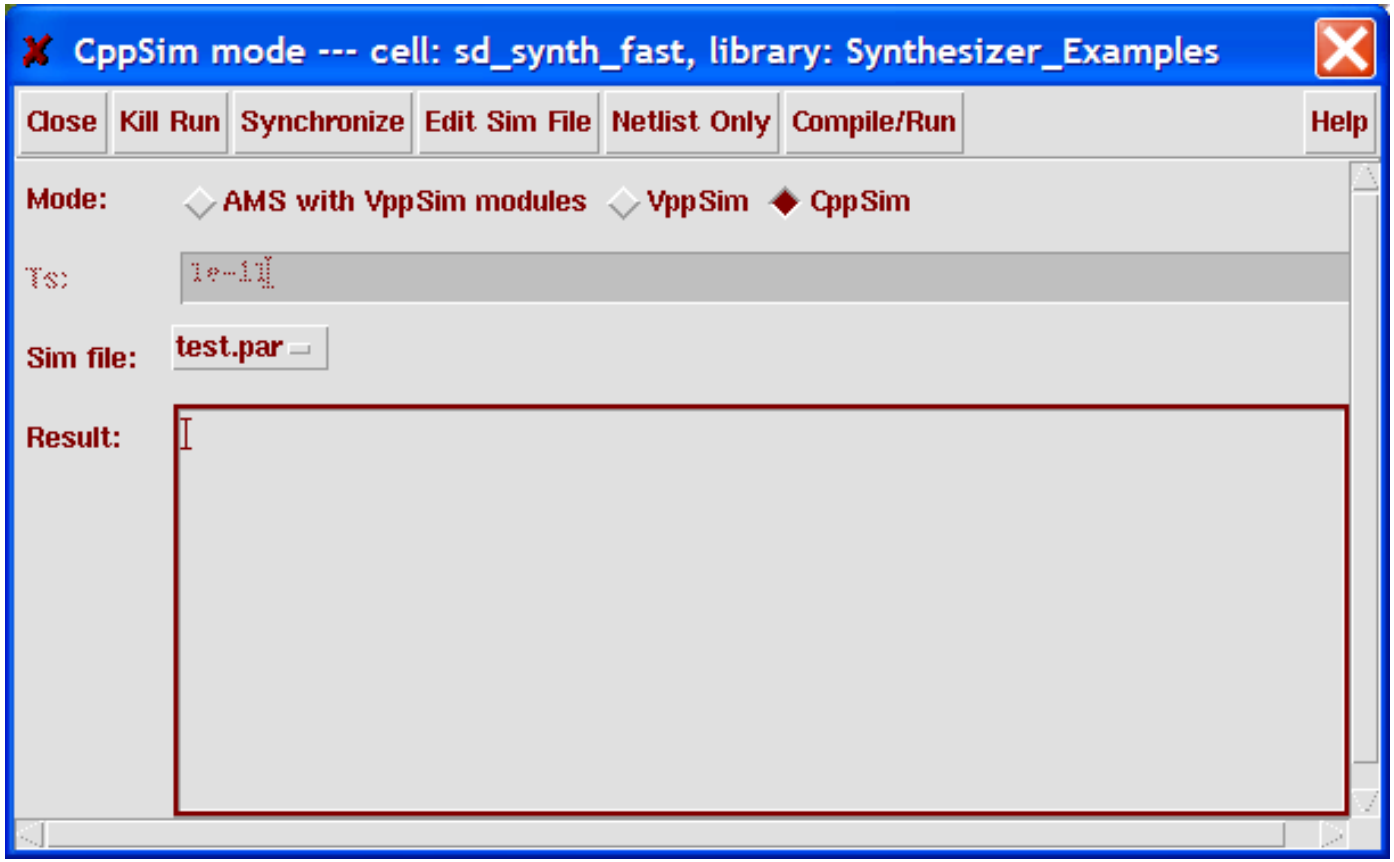
A
E
C
B
D

Impact of Alternate Ordering Arrangement



- **Ordering influences delays**
- **CppSim blocks are ordered automatically or by user**

Cadence Interface



- Choose between CppSim, VppSim (for NCVerilog), and AMS with VppSim modules
- CppSim and VppSim run in above window or in Matlab

Windows Version of CppSim

- **Will be used in class today**
- **Available for free download at http://www.cppsim.com**

- **Initial part of this session: fractional-N example**
- **Remaining part of this session: example of your choice**

Windows Interface