ETI 220 – Lab Seminar

Martin Anderson
edited by Pietro Andreani
Department for Electrical and Information Technology
Lund University

Presentation Objectives

- Administrative issues – Organisation and Requirements
- Lab Report Requirements
- Schedule
- Introduction to A/D and D/A Conversion
- Introduction to mixed signal design, tools and design methodology
- Introduction to the assignments
- Assignment 1 – FFT, Sampling, and Quantization
- Assignment 2 – Analog Modeling and Simulation with Verilog-A
- Assignment 3 – Mixed Signal Modeling and Simulation with SpectreVerilog

Organisation and Requirements

- MATLAB and Cadence files are available for download at www.eit.lth.se/course/eti220
- The course homepage contains **Errata** and FAQ page
- The assignments are well suited for self studies, but 3 supervised lab sessions are scheduled, where you can get help
- The total workload is about 60 hours, so it is **inevitable** that you **work outside the scheduled lab hours**. Start working now!
- Undergraduate students work in groups of **2 persons**
- Ph.D. students work alone
- One clearly **written report** per group is required
- Cheaters will be **prosecuted by the law**
- Lab report **hard deadline**: **14/12** ←←←←←←←←←←

Lab Report Requirements

- Include answers to homework exercises
- Include simulation results and calculations
- Include circuit schematics if you think you have done something wrong (it is easier for us to help you)
- Include the code you have written yourself
- Answer all questions in the lab manual
- Write concisely (no novels please)
- No handwritten reports
- Reports shall be handed in (paper copy, no email attachments) to **Pietro Andreani**

Lab Session Schedule

- Lab Seminar – Monday Nov. 7th 18-20 (E:2311)
- Lab Session 1a – Wednesday Nov. 9th, 8-12 (Blåtunga)
- Lab Session 1b – Wednesday Nov. 9th, 18-22 (Blåtunga)
- Lab Session 2a – Wednesday Nov. 16th, 8-12 (Blåtunga)
- Lab Session 2b – Wednesday Nov. 16th, 18-22 (Blåtunga)
- Lab Session 3a – Wednesday Nov. 23rd, 8-12 (Blåtunga)
- Lab Session 3b – Wednesday Nov. 23rd, 18-22 (Blåtunga)

Converters – Fundamental limitations

- **Sampling** – Bandwidth limitation (Nyquist)
- **Quantization** – Accuracy limitation (quantization errors)

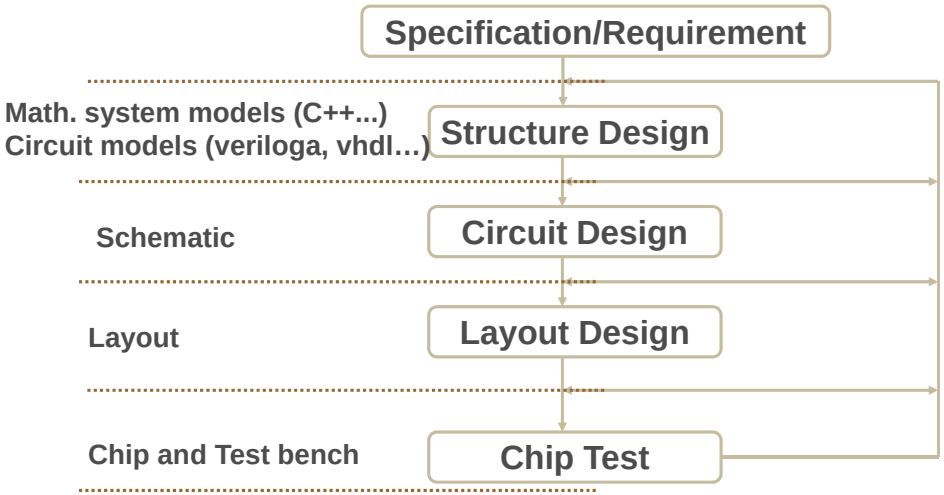
Converters – Practical limitations

- **Random noise**
  (ex: clock jitter, thermal noise, 1/f noise ...)
- **Distortion**
  (ex: clock jitter, substrate noise, circuit element mismatch and nonlinearity, charge injection, clock feedthrough, finite amplifier gain, clock skew and signal crosstalk)
- **Bandwidth limitations**
  Due to nonzero time constants of active and passive circuit elements.

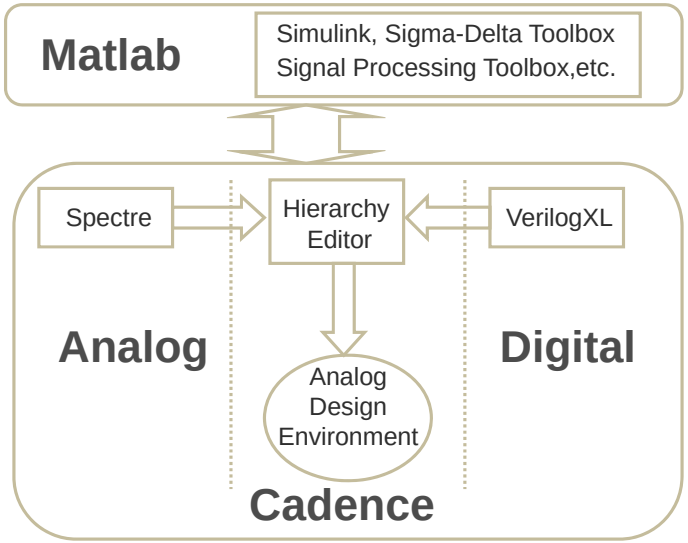
Mixed-Signal System Design (I)

- Mixed-signal systems are both analog and digital → both sensitive and complex
- Larger complexity than "analog-only" circuitry (5k-50k transistors)
- However, same accuracy requirements as purely analog circuits, at least for some of the circuit blocks
- Analog circuit simulators are accurate but slow
- Analog circuit simulators are dedicated to typical analog design issues such as noise, distortion and bandwidth
- Digital circuit simulators are fast, but not accurate
- Simulating digital circuits using analog simulators is extremely slow!

Mixed-Signal System Design (II) - Flow



Mixed-Signal System Design (III) - Flow



Assignments – Overview

1. Studies of the DFT, sampling and quantization using MATLAB – illustrates mathematical modeling of mixed-signal systems using MATLAB.

2. Circuit level modeling / simulation of mixed-signal circuits using Verilog-A modules

3. Circuit level modeling / simulation of mixed-signal circuits applied to a Flash ADC and to a Pipelined ADC using the spectreVerilog simulator

Assignment 1 – Learning Objectives

- Mathematical models and simulation using MATLAB
- Efficient and reusable MATLAB code
- How to perform a good FFT
- Fundamental BW and accuracy limitations in ADCs
- Sampling
- Quantization
- System simulations
- Performance metrics for ADCs / DACs (SNR, SNDR, IM, INL, DNL, ENOB, ERBW, SFDR ...)

Assignment 1 – Making a good FFT

- Coherent sampling! Make the FFT window an integer number of signal periods, with no repetition of the samples → make sure that $f_{in} = m*f_s/N$, where m is an odd integer prime and N (number of samples) is a power of 2
- The required simulation time will be proportional to *N,* so do not choose N unnecessarily large
- If windowing is used, compensate for the amplitude loss introduced by the window function. In assignment #1 this is done automatically by the code in *adcperf.m*
- The noise floor is also affected by windowing. For a hann window, the noise bandwidth is $NBW = 1.5/N$

Assignment 1 – Sampling

- Signal folding (aliasing)
- Sampling noise (kT/C)
- Sampling-time uncertainty (clock jitter)
- Nonlinearity

Assignment 1 – Quantization

- Noise equivalent of the quantization error
- Quantization error power and SNR
- Statistical properties of quantization errors. Random signal or … ?
- Dithering
- Additive circuit noise and nonlinearity

Assignment 2 – Learning Objectives

- Principles of modeling mixed-signal designs or complex analog designs using Cadence
- Familiarize with the Verilog-A syntax
- Build circuit models using Verilog-A
- Simulating Verilog-A based modules
- Using the Hierarchy Editor (config views)

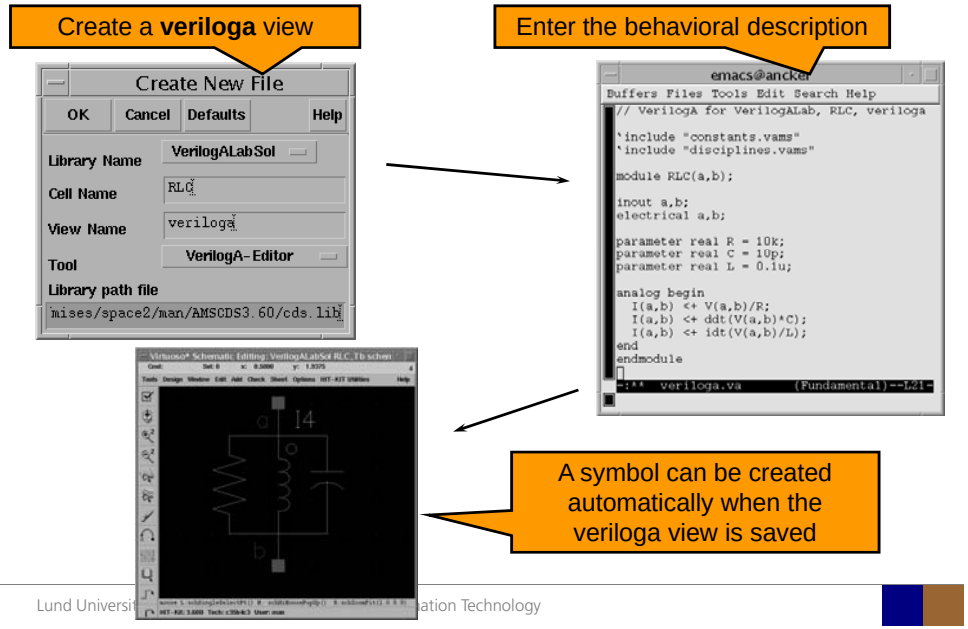
Assignment 2 – Getting help

- Seminar slides + FAQ on the course homepage
- Online documentation
  - Open through forms and windows in Cadence
  - Type **cdsdoc &** to launch online documentation
- Cadence Verilog-A Language Manual, available online at $CDSDIR/doc/veriaref/veriaref.pdf
- Reference material
  - Fitzpatrick and Miller, "Analog Behavioral Modeling with the Verilog-A Language", Kluwer Academic Publishers, 1998.
  - Kundert and Zinke, "The Designers Guide to Verilog-AMS", Kluwer Academic Publishers, 2004.

Assignment 2 – Verilog-A Overview

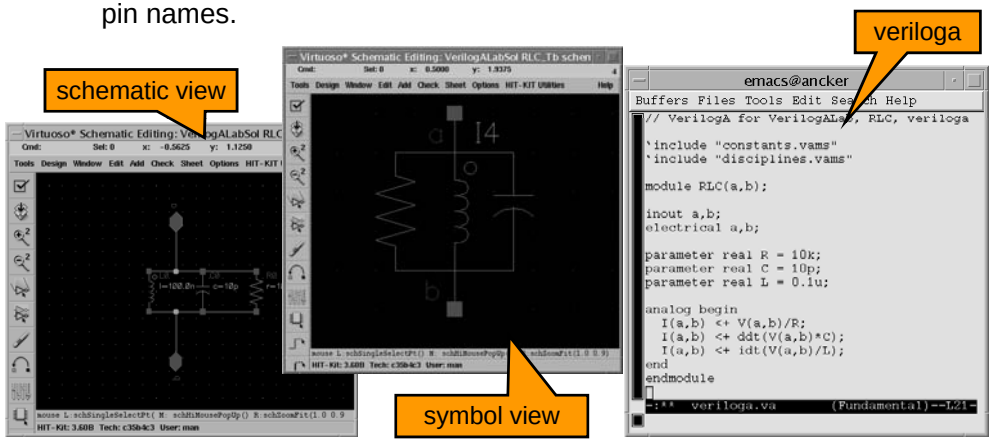
- Verilog-A is an extension of Verilog, used to describe analog / mixed signal circuit elements. It is an Open Verilog International (OVI) standard.
- Multidisciplinary, models electrical, mechanical, fluid and thermodynamic systems
- Used with *spectre* circuit simulator in the Virtuoso Analog Design Environment, in the Analog Workbench (AWB) or in Virtuoso AMS Designer
- Supports top-down design and speeds up system simulation times

Assignment 2 – Creating Verilog-A Modules

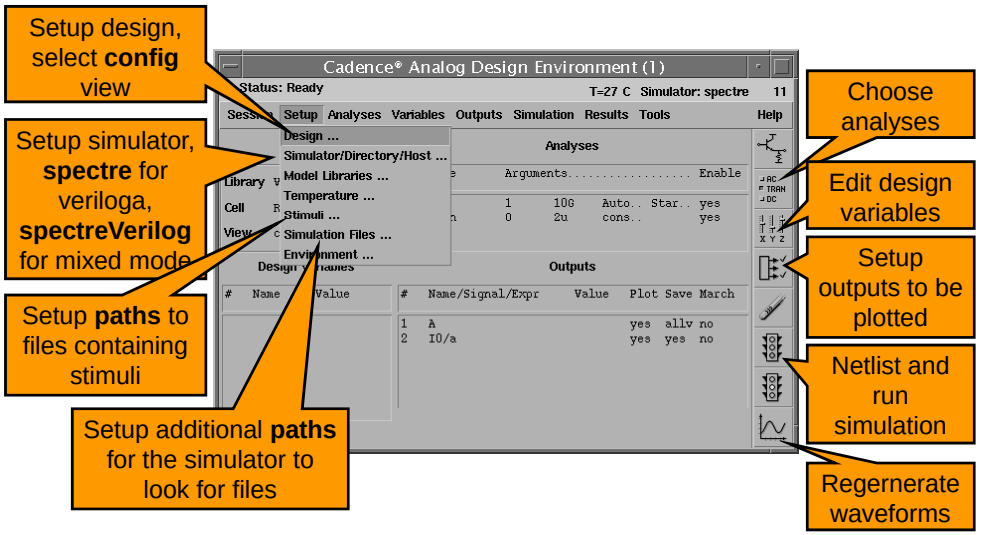


Assignment 2 – Verilog-A blocks in schematics

- A veriloga view requires a symbol view in order to be instantiated in a schematic. If a schematic view of the same cell also exists, all three views (veriloga, schematic, symbol) must have the same pin names.



Assignment 2 – Simulate Verilog-A in ADE



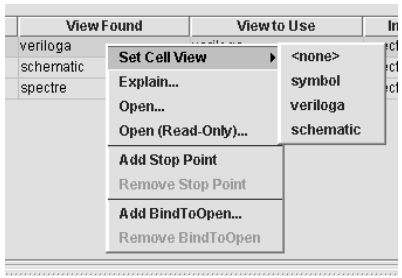
Assignment 2 – Simulation view list

- In order for spectre to simulate your modules, the **veriloga view** must be in the Switch View List (select **Setup → Environment**) – by default, veriloga is assigned the next-to-last position
- When creating cellviews with names other than the default names (for example *veriloga_2*), adjust the Switch View and Stop View Lists accordingly
- In mixed signal mode, or with large analog configurations, use the Hierarchy Editor to modify the Switch View List and the Stop View List

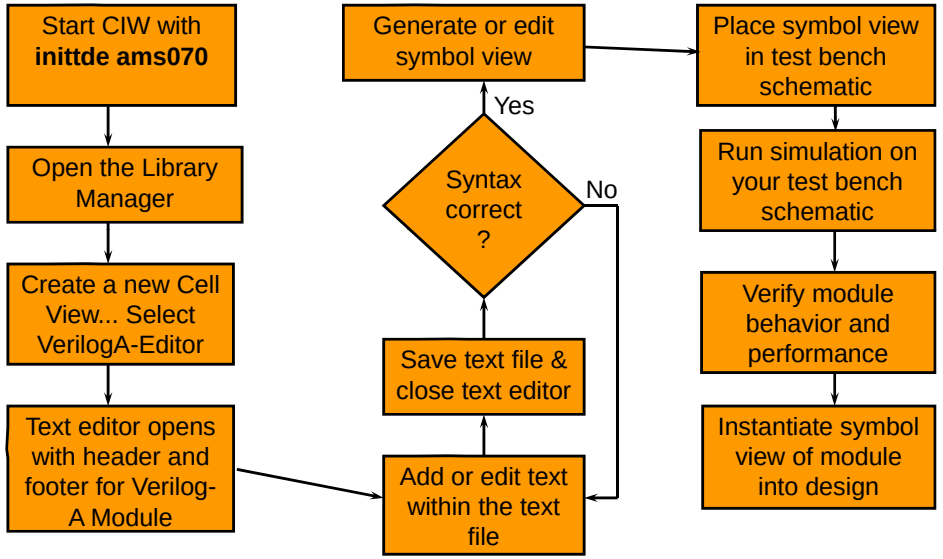
Note: The Hierarchy Editor is strictly required only in the mixed-signal mode, and the simulator is specified as spectreVerilog (which is selected as template when creating the config view of the design).

Assignment 2 – Simulation view list

- The Hierarchy Editor is opened automatically when you create a config view
- The Hierarchy Editor permits changing views for each cell, and then saving the cell bindings to a configuration file
- Use the right mouse button to bring up the menu to select among the different views
- Don't forget to update the configuration and save it
- Open the configured schematic from the Hierarchy Editor for running a simulation on the new configuration



Assignment 2 – Verilog-A in ADE



Assignment 2 – Verilog-A in ADE

'include compiler directives to files that are needed for behavioral models

Interface declarations including:
- Module name
- Port directions and disciplines
- Parameter declarations with default values
- Internal variables

A module scope describes the function of a module
Behavioral models use analog behavioral statements.
Structural models use instantiated components.

```
'include "constants.vams"
'include "disciplines.vams"

module MyRes(p,n);

inout p,n;
electrical p,n;
parameter real R = 1 from (0:inf);
parameter real tc = 1.5m from [0:3m);

real Reff;

analog begin
  @(initial_step) begin
    Reff = R*(1+tc*$temperature);
  end
  I(p,n) <+ V(p,n) / Reff;
end

endmodule
```


Assignment 2 – Standard Include Files

- 'include "disciplines.vams"
  - Defines a set of disciplines and natures that define the signal types
  - Cadence provides a working sample of a disciplines.vams file at: `<install_dir>/tools/spectre/etc/ahdl`
- 'include "constants.vams"
  - Defines a set of mathematical and physical constants.
  - Cadence provides a working sample of a constants.vams file at: `<install_dir>/tools/spectre/etc/ahdl`
  - Mathematical constants have 'M_ prefix, ex: 'M_PI
  - Physical constants have 'P_ prefix, ex: 'P_K (Boltzman's constant)

Assignment 2 – Port Declaration

- Input signals
  - Cannot be set, but can be used in expressions
- Output signals
  - Can be set, but cannot be used in expressions
- Inout signals (default)
  - Are bidirectional. Can be both set and used in expressions

```
module gainer (out,in);          // Declares two ports
output out;                      // Declares port as output
input in;                        // Declares port as input
electrical in, out;              // Declares discipline of
                                 // ports

parameter real gain = 2.0;
analog
  V(out) <+ gain * V(in);
endmodule
```


Assignment 2 – Declaring Parameters

- Use parameter declarations to customize a module for each instantiation in a design
  - Specify default value for each parameter
  - Specify an optional type and an optional valid range

```
module sdiode(np,nn);
  inout np,nn;
  electrical np,nn;
  parameter real area = 1;
  parameter real is = 1e-14;
  parameter real n = 2;
  parameter real cjo = 0;
  parameter real m = 0.5;
  parameter real phi = 0.7, tt = 1p;
  real vd,id,gd;
    analog begin
      vd = V(np,nn);
      id = area*is*(exp(vd/(n*$vt))-1);
      gd = tt*id + area*vd*cjo/pow((1-vd/phi),m);
      I(np,nn) <+ id + ddt(gd);
    end
endmodule
```

- $vt is the thermal voltage, kT/q, with temperature T set as in the simulator

Assignment 2 – Parameter Ranges

- Use the optional range specification to express valid parameter ranges for the behavioral model formulation
- If the range specification is violated during instantiation, the simulation will be terminated
- Range boundaries can be included or excluded, as shown in the example:

```
parameter real resistance = 1;
parameter real vol_high = 5, vol_low = 0;
parameter real gain = 1 from [0:1K);
parameter real pole = 1 exclude [0];
parameter real frequency = 1 from (0:inf);

"[" or "]" implies the boundary is included
"(" or ")" implies the boundary is NOT included
```


Assignment 2 – Contribution Operator <+

- The contribution operator <+ is only applied to signal access functions (potential or flow signals) within the module
- The contribution operator <+ accumulates potential or flow for the target of contribution on successive contribution statements

```
V(a)   <+ 0;               // sets 'a' to a voltage of zero
I(p,n) <+ V(p,n)/r;        // models resistance between p and n
V(p,n) <+ I(p,n)*r;        // models resistance between p and n
I(p,n) <+ ddt(c*V(p,n));   // models a capacitance between p and n
V(p,n) <+ ddt(L*I(p,n));   // models inductance between p and n
V(out) <+ 1.0;             // out is 1 volt relative to ground
V(out) <+ gain*V(in);      // out is gain times more than in
V(out) <+ idt(V(in), 0);   // integrates the output voltage with
                           // respect to time, setting initial
                           // condition of 0 volts
```


Assignment 2 – Contribution / Assignment

- Contribution statements (<+) define behavior and allow input signals to be mapped to output signals, and implicitly define branch relations. The branch is directed from the first node of the access function to the second node, or reference node if the second node is omitted
- Assignment operators (=) can only be applied to variables, whereas contribution operators an only be applied to access functions (potential or flow source branches)
- Assigning a number to a variable (x=4) overrides the number previously contained in that variable, whereas contributing to an access function (V(x)<+4) adds to any previous contributions within that analog block cycle
- Contributions to current can be viewed as adding current sources in parallel. In contrast, contributions to voltages can be viewed as adding voltage sources in series.

Assignment 2 – Contribution example

- Behavioral descriptions are defined only in the analog block
- The behavioral description is a relationship between the input signals and the output signals
- The contribution operator accumulates the values on the right side of the operator in the placeholder on the left side
  - Accumulates potentials or flows to nodes, ports and branches
  - The order of contribution is not significant

```
module rlc(a,b);
  inout a,b;
  electrical a,b;
  parameter real R = 1k exclude 0;
  parameter real C = 1p;
  parameter real L = 1n exclude 0;
    analog begin
      I(a,b) <+ V(a,b)/R;
      I(a,b) <+ ddt(V(a,b)*C);
      I(a,b) <+ idt(V(a,b)/L);
    end
endmodule
```

Assignment 2 – Implicit Equations

- The contribution operator supports implicit equations, where the same expression is used on both sides of the equation.
  - Solves for x when x <+ f(x), as illustrated below in the diode model. The equation that describes the diode has I(a,c) on both sides of the contribution operator.

```
module diode(a,c);
  inout a,c;
  electrical a,c;
  parameter real r = 1f from (0:inf);
  parameter real rs = 0 from [0:inf);
    analog begin
      I(a,c) <+ is * ($limexp((V(a,c) - rs * I(a,c))/$vt) - 1);
    end
endmodule
```

- The *limexp* function includes special limiting algorithms to improve convergence

Assignment 2 – Viewing Variables

- Variables from your **veriloga** modules can be viewed as follows:

  1) Set the simulator to save Verilog-A variables by executing **Outputs → Save All**, and checking "Select AHDL Variables (saveahdlvars)" to save all ahdl variables

  2) Simulate the design using the desired analyses

  3) **Execute Tools → Results Browser** from the Virtuoso Analog Design Environment simulation window to open the WaveScan Results Browser

  4) In the WaveScan Results Browser, select the analysis (e.g. tran-tran, dc-dc), and find the varable of interest to plot

  5) Use a right-click to append the signal to the waveform display

Assignment 2 – Internal Nodes

- When necessary, modules can have local internal nodes to define:
  - Internal branch relationships
  - Higher order or nonlinear differential equations

```
module dline(p1,p2);
  inout p1,p2;
  electrical p1,p2,n1;
  parameter real delay = 50p, zo=75;
  real L,C;
    analog begin
      @(initial_step) begin
        L = zo*delay/2;
        C = delay/zo;
      end
    V(p1,n1) <+ ddt(L*I(p1,n1));
    V(p2,n1) <+ ddt(L*I(p2,n1));
    I(n1) <+ ddt(C*V(n1));
  end
endmodule
```


Assignment 2 – Declaring Buses

- Use the net discipline declaration to associate nets with previously defined descriptions.
  - Nets declared without a range is called scalar nets
  - A net declared with a range is called a vector or bus net

```
electrical [1:10] n1;
electrical [3:0] n2;
```

- The module below uses an input bus, where only the name is appearing in the list of ports, the range is defined inside the module ([4:0])

```
module five_inputs(portbus);
  input [4:0] portbus;
  electrical [4:0] portbus;
    analog begin
      generate i (4, 0)
        V(portbus[i]) <+ 0.0;
    end
endmodule
```

Assignment 2 – Event Driven Modeling

- The simulator generates analog events that control model behavior. Modules detect these events and use them to determine wheter specific statements shall run
  - Event-driven modeling uses the @block construct to execute a block of code when an event occurs
  - Types of events supported by Verilog-A modeling are:

```
initial_step            At beginning of simulation
final_step              At end of simulation
cross()                 At analog signal crossings
last_crossing()         At analog signal crossings
timer()                 Periodically at a specific time
```


Assignment 2 – The Timer Event Operator

- The timer event operator can be used to generate events at a specified time or at specified time intervals during simulation
  - Example: The squarewave module produces a square wave voltage waveform:

```
module squarewave(out);
  output out;
  electrical out;
  parameter period = 1.0;
  integer x;
    analog begin
      @(initial_step) x = 1;
      @(timer(0, period/2)) x = -x;
      V(out) <+ transition(x, 0.0, period/100.0);
    end
endmodule
```


Assignment 2 – Obtain Simulator Settings

- Verilog-A provides an environment parameter function that can be used to obtain the current simulation time in seconds: $abstime
  - Example: $strobe("Simulation time = %e", $abstime)
- The ambient temperature of a circuit in Kelvin degrees can be obtained using: $temperature
  - Example: temp := $temperature
- The thermal voltage (V_T=kT/q) can be obtained using $vt.
  - Example: vt_function := $vt(temp)
  - temp is the temperature in degrees Kelvin
  - When not specifying temp, the thermal voltage is calculated at the temperature returned by the $temperature function

Assignment 2 – Verilog-A Table Models

- A Table Model represents component behavior by constructing a look-up table derived from Spice or Spectre models or measurements
  - Allows running simulation on LUTs instead of complex spectre netlist for shorter simulation times
  - IP protection
  - Can be used to import any table of values
  - Can be extracted using Skill scripts
  - Interpolation used to approximate between tabulated values
- Example Verilog-A syntax for a mosfet table model:

```
module mynmos(g,s,d)
electrical g,s,d;
inout g,s,d;
  analog begin
    I(d,s) <+ $table_model(V(g,s), V(d,s), "nmos.tbl","3CL,3CL");
  end
endmodule
```

Assignment 2 – More Modeling Details...

- Random number support
- Noise modeling
- Functions and operators (mod, shift ...)
- Conditional statements (if-else, case, while, for ...)
- Analog operators (sin, tan ...)
- Slew and transition filters
- File handling (open, close, read and write)
- Laplace transform and z-transform
- Displaying results ($display, $strobe ...)

Assignment 3 – Learning Objectives

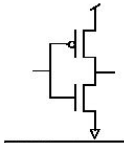
- Principles of modeling mixed-mode designs using Cadence
- Partitioning of designs to enable mixed mode simulations using the spectreVerilog simulator
- Using Cadence Hierarchy Editor (config views)
- Principles of flash A/D conversion
- Principles of pipelined A/D conversion
- Evaluating static and dynamic circuit performance using Cadence and MATLAB

Assignment 3 – spectreVerilog simulation

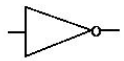
- Simulation will be done using the spectreVerilog simulator (verilogXL + spectre)
- Build testbench
- Create, edit and save the config view of the testbench
- Open the schematic window from the config view
- Open Analog Design Environment from schematic window
- Change simulator to spectreVerilog
- Setup design files and environment
- Partition the design
- Select and run simulations of your choice

Assignment 3 – Learning Objectives

An Inverter



```
Module Inverter (in, out)
  [V,I] in;
  [V,I] out;
  real tmp;
  analog{
    tmp=5-V(in);
    V(out)<-$transition(...
```



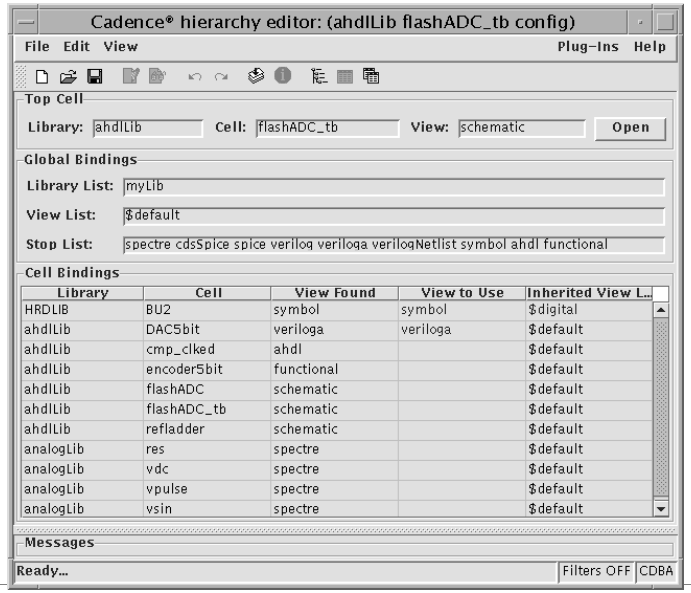
```
Module Inverter (in, out)
  input in;
  output out;
always
  begin:
    out<- ~in;
```

View Name	Simulator	Accuracy & Speed
schematic	spectre	Most Accurate Slowest
veriloga	spectre	More Accurate Slower
symbol	verilogXL	Accurate Fast
functional	verilogXL	Roughly Fastest

Assignment 3 – Library, view, simulator

- **analogLib** – Ideal analog passives, signal sources, grounds
  Views to use: spectre Simulator: spectre
- **CORELIB** – Digital standard cells from the foundry (AMS)
  Views to use: symbol/cmos_sch
  Simulator: VerilogXL/spectre
- **PRIMLIB** – Analog circuit elements such as transistors
  Views to use: spectre Simulator: spectre
- **Your design libraries**
  Views to use: veriloga, functional, schematic, layout ...
  Simulator: spectre / VerilogXL

Assignment 3 – The Hierarchy Editor



Select which view of each instance to use

Partition the design into digital and analog blocks

Spectre?

VerilogXL?

Assignment 3 – Analog / Digital Partitioning

Must agree with the Stop List in the Hierarchy Editor

Analog
veriloga, schematic view → spectre

Digital
functional view → VerilogXL