5 Analog Behaviour

Pushpak T

5.2 Analog procedural block

- As initial & always block is use for descrete time behaviour, similry here we use
- Analog block for continous time.
- No blovking event control statemetn are allowed in continous time.
- All statement in analog blocks execute sequenctially
- If multiple module, then it works concurrently

5.2 Analog procedural block

- Analog initial block, special block for simulation initialization.
- analog initial block shall not contain the following statements:
  — statements with access functions or analog operators;
  — contribution statements;
  — event control statements.
- Analog block cant access digital values.
- Gets Executed for each analysis, also execute for each sub task analysis.
- If parameter changed in between then it will execute again.

5.3 Block statements

- Are know as Sequential blocks
- Sequential blocks are used to group procedural statements together, ensuring they are executed one after another in the specific order in which they appear.
- Begin - end is use
- Block can be given name as identifier usefule when multiple blocks are present.
- Variable inside block are local.

```
module example;
   parameter integer p1 = 1;
   real moduleVar;

   analog begin
      begin: myscope
         parameter real p2 = p1;
         real localVar = 1.5 * p2;
      end
      moduleVar = myscope.localVar;
   end
endmodule
```

5.4 Analog signals

- The access function I() is used for flow (current), similar to how V() is used for potential (voltage).
- Named branch (e.g., b), you access the voltage using the access function V(b) and the current using I(b).
- For unnamed branch, V(n1, n2) accesses the voltage difference between n1 and n2.
- If only V(n1) means wrt ground.

5.4 Analog signals

- Probes are branches where the potential or flow is not explicitly assigned a value but rather monitored in expressions
- You cannot use both the potential and the flow of a probe branch together. It is illegal to combine them in an expression.
- Flow probe: If the flow (current) is used in an expression, it's a flow probe.
- Potential probe: If the potential (voltage) is used in an expression, it's a potential probe.

5.4 Analog signals

- Sources are branches where either the potential or flow is explicitly assigned a value. A branch can be either a potential source or a flow source, but not both at the same time. However, it can switch between being a potential or flow source.
- Potential source: If the potential (voltage) is assigned a value by a contribution statement, it's considered a potential source.
- Flow source: If the flow (current) is assigned a value by a contribution statement, it's considered a flow source.

5.4 Analog signals

- Accessing flow through a port
- I(<p1>) accesses the current flow into the module through the electrical port p1.
- Port access Used to access the flow (current) through a port, with the port name enclosed in <>.
- Cannot be used to assign values to a port or net using the <+ operator.

5.4 Analog signals

- Unassigned sources:
- If you do not assign a value to a branch (such as the current or voltage), and the branch is not a probe, automatically sets the flow (current) to zero.

5.5 Accessing net and branch signals

- Accessing net and branch signals..
- Potential & flow using V(),I().

5.5 Accessing net and branch attributes.

- Attributes are attached to the nature of a potential or flow. Therefore, the attributes for a net or a branch can be accessed by using the hierarchical referencing operator (.) to the potential or flow for the net or branch.
- This syntax shall not be used for the access, ddt_nature, or idt_nature attributes of a nature, nor any other attribute whose value is not a constant expression.

```
module twocap(a, b, n1, n2);
   inout a, b, n1, n2;
   electrical a, b, n1, n2;
   branch (n1, n2) cap;
   parameter real c= 1p;
   analog begin
      I(a,b) <+ c*ddt(V(a,b), a.potential.abstol);
      I(cap) <+ c*ddt(V(cap), n1.potential.abstol);
   end
endmodule
```

5.5 Accessing net and branch attributes.

- Accessing nets and branch signals hierarchically

```
module top;
   A a1();
   B b1();
endmodule

module A;
   electrical n,p;
   branch (n,p) b;
   analog V(b) <+ 1.34;
endmodule
```

```
module B;
   analog $strobe("voltage == %g", V(top.a1.b));
endmodule
```

5.6 Contribution statements

- 5.6.1 Direct branch contribution statements
- **Direct Contribution Statement (<+)**: Used to define the relationship between analog signals (voltage, current) in Verilog-AMS.
- The left-hand side (LHS) specifies the branch (signal) being updated.
- The right-hand side (RHS) specifies the expression (voltage or current) that assigns the value.
- Can be used to describe **resistors**, **capacitors**, and other components by relating voltage, current, and other analog expressions.
- You can use **access functions** (e.g., V, I, potential, flow) to **probe** the voltage or current at various points in the circuit.
- A relation between the flow and potential between two nets. If no second net is specified, the relation will be between the net and ground (the global reference node).
- Eg The branch relation between the input (in) and the output (out) is given by the equation V(out) <+ Gain * V(in), meaning the voltage at the output is a scaled version of the voltage at the input based on the gain parameter.

5.6 Contribution statements

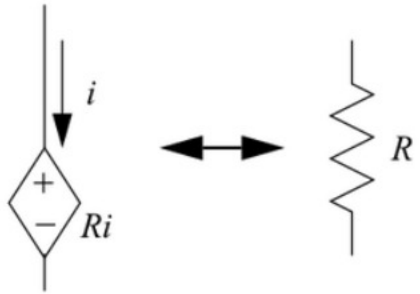
- 5.6.1 Direct branch contribution statements
- A relation between the flow and potential between two nets. If no second net is specified, the relation will be between the net and ground (the global reference node).
- Eg The branch relation between the input (in) and the output (out) is given by the equation V(out) <+ Gain * V(in), meaning the voltage at the output is a scaled version of the voltage at the input based on the gain parameter.
- Step 1: The right-hand side of the branch contribution (the expression involving parameters like Gain, Rin, Cin, etc.) is calculated.
- Step 2: The calculated value is added to any previously retained value of the branch (voltage or current).
- Step 3: At the end of the simulation cycle, the simulator updates the branch with the accumulated (retained) value.
- This process ensures that the simulation reflects the dynamic behavior of electrical circuits, including the effects of both active components (like the amplifier gain) and parasitic elements (like resistances, capacitances, and inductances).

5.6 Contribution statements

- R & C circuits .

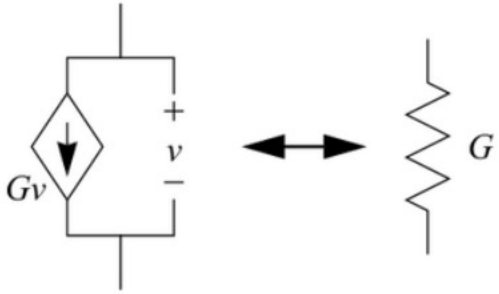
```
module my_resistor(p,n);
    inout p,n;
    electrical p,n;
    parameter real R=1;
    branch (p,n) res;

    analog begin
        V(res) <+ R * I(res);
    end
endmodule
```




```
module my_conductor(p,n);
    inout p, n;
    electrical p,n;
    parameter real G=1;
    branch (p,n) cond;

    analog begin
        I(cond) <+ G * V(cond);
    end
endmodule
```



5.6 Contribution statements

- RLC circuits .

A series RLC circuit is formulated by summing the voltage across its three components,

$$v(t) = Ri(t) + L\frac{d}{dt}i(t) + \frac{1}{C}\int_{-\infty}^{t} i(\tau)d\tau$$

which can be defined as

```
V(p, n) <+ R*I(p, n) + L*ddt(I(p, n)) + idt(I(p, n))/C;
```

A parallel RLC circuit is formulated by summing the currents through its three components,

$$i(t) = \frac{v(t)}{R} + C\frac{d}{dt}v(t) + \frac{1}{L}\int_{-\infty}^{t} v(\tau)d\tau$$

which can be defined as

```
I(p, n) <+ V(p, n)/R + C*ddt(V(p, n)) + idt(V(p, n))/L;
```

5.6 Contribution statements

- Switch branches
- You can dynamically switch between contributing voltage (potential) or current (flow) to a branch, useful for modeling ideal switches.
- Contribution statements can be conditional but should not use **event control** for simulation correctness.
- When switching between flow and potential, a **discontinuity of order 0** is assumed, so no need for special discontinuity handling.

5.6 Contribution statements

- Implicit Contributions
- An **implicit or fixed-point formulation** in Verilog-AMS allows the current or voltage of a branch to be defined as a function of itself. This creates a self-referential equation, which the simulator solves iteratively. The example you provided with I(diode) <+ is*(limexp((V(diode) - r*I(diode))/$vt) - 1); shows how the current I(diode) depends on both the voltage and the current itself, requiring the simulator to compute a consistent solution over several iterations. This is an important technique for modeling non-linear components like diodes or transistors, where the relationship between voltage and current is inherently non-linear and requires iterative methods to solve.

5.7 Analog procedure statement

- Analog procedural assignments are used to modify analog variables such as integers, real numbers, and strings, including array variables in Verilog-AMS. These assignments are typically used inside analog procedural blocks (like analog or analog initial blocks).
- The RHS of the assignment can include analog operators, but certain restrictions apply (no concatenation, no analog filter functions, and no hierarchical assignments).
- Packed arrays: Dimensions are defined before the data identifier.
- Unpacked arrays: Dimensions are defined after the data identifier.
- The arrays on the LHS and RHS must be compatible in type and size (the number of elements).

5.8 Analog conditional statement

- If else if
- Else is optional , for single statement
- Indentation is important, or mention begin end.

```
if (index > 0) begin
   if (i > j)
      result = i;
end
else result = j;
```

5.8 Analog conditional statement

- The case statement is a multi-way decision statement which tests if an expression matches one of a number of other expressions

```
analog_case_statement ::=
        case ( analog_expression ) analog_case_item { analog_case_item } endcase
      | casex  ( analog_expression ) analog_case_item { analog_case_item } endcase
      | casez  ( analog_expression ) analog_case_item { analog_case_item } endcase
analog_case_item ::=
        analog_expression { , analog_expression } : analog_statement_or_null
      | default [ : ] analog_statement_or_null
```

5.9 Looping statements

- repeat() executes a statement a fixed number of times. Evaluation of the expression decides how many
  times a statement is executed.

- analog_loop_statement ::=
  repeat (analog_expression) analog_statement

5.9 Looping statements

- while() executes a statement until an expression becomes False. If the expression starts out False, the statement is not executed at all.
- while (analog_expression) analog_statement

5.9 Looping statements

- Analog_for(), is similar to for().
- Can contain analog operator

```
analog begin
   tmp = 0.0;
   for (k = 1; k <= width; k = k + 1) begin
      tmp = tmp + V(dt[k]);
      V(out) <+ ddt(V(dt[k]));
   end
end
```