```
    variable j : integer := 1;
    variable l : integer := 1;

 begin

     loop1:
       for i in 1 to 100 loop
        write(K,"Time = ");
        write(K,T_IN1);
        write(K," ");
        clock <= '0', '1' after 2 ns, '0' after 3 ns;
        wait for 5 ns;
        if t < 195 ns then
            error_value(i) <= error1(anaout);
        elsif t > 195 ns then
            extern_feed1 <= true;
            error_value <= error_value;
            sigdel_real_out(l) <= anaout;
            wait for 0 ns;
            lowpass_feed2 <= true;
            lowpass_input2 <= sigdel_real_out;
            lowpass_out <= v_output2(2);
            j := j + 1;
            l := l + 1;
        end if;
        write(K,"SIGDEL_OUT = ");
        write(K,lowpass_out);
        write(K," ");
        writeline(OUTPUT,K);
        t := t + 5 ns;
        exit loop1 when t = 700 ns;
     end loop loop1;
     stop_sim <= true;

end process p1;

end sigdel;
```

```
        port map (v_output => v_output2, T_IN => T_IN2,
                  stop => stop_at2,
                  extern_feed => extern_feed2,
                  error_value => error_value, Vgs => Vgs2,
                  lowpass_feed => lowpass_feed2,
                  lowpass_input => lowpass_input2,
                  FILE_FLAG => FILE_FLAG2);

   a2d1 : A2D
        generic map (hi_cutoff => 3.7, lo_cutoff => 1.7)
        port map (LOGIC_OUTPUT => LOGIC_OUTPUT, ANALOG_INPUT
          => v_output1(8),T_input => T_IN1, stop_at => stop_at1,
                   Vgate_src => Vgs1);

   d_ff1 : d_ff
           generic map (1 ns)
        port map (data => LOGIC_OUTPUT, clock => clock, q => dff_out);

   d2a1 : D2A
        port map (LOGIC_IN => dff_out, Time_in => T_IN2, stop_after =>
             stop_at2, ANALOG_OUTPUT => anaout);



stop_control : process

 begin

   wait until stop_sim = true;
   assert false report "Stopping simulation"severity failure;

end process stop_control;


p1 : process

   variable K : LINE;
   variable t : time := 0 ns;
   variable i : integer := 1;
```

```
  signal v_output1      : VALUE_ARRAY(1 to 100);
  signal v_output2      : VALUE_ARRAY(1 to 100);
  signal T_IN1          : TIME;
  signal T_IN2          : TIME;
  signal stop_at1       : TIME;
  signal stop_at2       : TIME;
  signal LOGIC_OUTPUT : SWITCH_LEVEL;
  signal clock          : SWITCH_LEVEL;
  signal dff_out        : SWITCH_LEVEL;
  signal stop_sim       : boolean := false;
  signal anaout         : real;
  signal extern_feed1   : boolean := false;
  signal extern_feed2   : boolean := false;
  signal error_value  : VALUE_ARRAY (1 to 100);
  signal Vgs1           : real;
  signal Vgs2           : real;
  signal sigdel_real_out : VALUE_ARRAY (1 to 100);
  signal lowpass_feed1 : boolean := false;
  signal lowpass_feed2 : boolean := false;
  signal lowpass_input1  : VALUE_ARRAY (1 to 100);
  signal lowpass_input2  : VALUE_ARRAY (1 to 100);
  signal lowpass_out : real;
  signal FILE_FLAG1 : BOOLEAN := TRUE;
  signal FILE_FLAG2 : BOOLEAN := FALSE;


file FILE2 : TEXT is out "STD_OUTPUT";

begin

 ana1 : anavhdl
        port map (v_output => v_output1, T_IN => T_IN1,
                  stop => stop_at1, extern_feed => extern_feed1,
                  error_value => error_value, Vgs => Vgs1,
                  lowpass_feed => lowpass_feed1,
                  lowpass_input => lowpass_input1,
                  FILE_FLAG => FILE_FLAG1);

 ana2 : anavhdl
```

```
            stop          : out TIME;       -- Simulation end time.
            incri         : out TIME;       -- Time incriment step.
            Vgs           : out real;
            extern_feed   : in boolean;
            FILE_FLAG     : in boolean;
            lowpass_feed  : in boolean;
            lowpass_input : in VALUE_ARRAY;
            error_value   : in VALUE_ARRAY);
  end component;

  component A2D
     generic (hi_cutoff : real;
              lo_cutoff : real);
     port (LOGIC_OUTPUT : out SWITCH_LEVEL;   -- Digital voltage output.
           ANALOG_INPUT : in real;            -- Analog voltage output.
           T_input      : in TIME;            -- Circuit simulation time.
           stop_at      : in TIME;            -- Simulation end time.
           Vgate_src    : in real);
  end component;

  component D2A
     port (LOGIC_IN      : in SWITCH_LEVEL;   -- Digital voltage input.
           Time_in       : in TIME;           -- Simulation time from AnaVHDL.
           stop_after    : in TIME;           -- Simulation end time.
           ANALOG_OUTPUT : out real);         -- Analog voltage output.
  end component;

  component d_ff
     generic (nand_delay : time := 0 ns);
     port(clock, data : in SWITCH_LEVEL;
          q, qb : out SWITCH_LEVEL);
  end component;


for all : anavhdl use entity work.anavhdl(anavhdl);
for all : A2D use entity work.A2D(A2D);
for all : D2A use entity work.D2A(D2A);
for all : d_ff use entity work.d_ff(d_ff);
```

```
p1 : process

   variable K   : LINE;            -- Debugg variable.

  begin

    if (t1 < (stop_after1 - 1 ns)) then
       wait for 0 ns;
       write(K,t1);
       write(K," ");
       wait for 0 ns;
       write(K,analog);
       writeline(OUTPUT,K);
    end if;
    wait for 5 ns;
    if t1 = 295 ns then
      stop_sim <= true;
    end if;
end process p1;

end system;
```

## Sigma-Delta Converter

```
use STD.TEXTIO.all;
use work.COMPUTE_PACK.all;
use work.IO_PACK.all;
use work.INTERFACE_PACK.all;

entity sigdel is
end sigdel;

architecture sigdel of sigdel is

 component anavhdl
   port (v_output    : out VALUE_ARRAY;     -- Analog output.
         T_IN        : out TIME;        -- Circuit simulation Time.
```

```
ana2 : anavhdl
        port map (v_output => input2, stop => stop_after2,
                incri => incriment2, T_IN => t2, Vgs => v_gate_src2,
                extern_feed => extern_feed1, FILE_FLAG => FILE_FLAG2,
         lowpass_feed => lowpass_feed2, lowpass_input =>
         lowpass_input2,error_value => error_value2,
         DC_IN => DC_IN2, DC_FLAG => DC_FLAG2);

a2d1 : A2D
      generic map (hi_cutoff => 3.7, lo_cutoff => 1.7)
      port map (ANALOG_INPUT => input1(2), LOGIC_OUTPUT => in1,
              T_input => t1, stop_at => stop_after1,
              Vgate_src => v_gate_src1);

a2d2 : A2D
      generic map (hi_cutoff => 3.7, lo_cutoff => 1.7)
      port map (ANALOG_INPUT => input2(2), LOGIC_OUTPUT => in2,
              T_input => t2, stop_at => stop_after2,
              Vgate_src => v_gate_src2);

nand1: nand_gate
        generic map(2,0 ns)
        port map(input(1) => in1, input(0) => in2,
                  output => out1);

d2a1 : D2A
      port map (LOGIC_IN => out1, Time_in => t1,
              stop_after => stop_after1, ANALOG_OUTPUT => analog);

stop : process

   begin
     wait until stop_sim = true;
     assert false report "Stopping Simulation" severity failure;

end process stop;
```

```
signal in2            :  SWITCH_LEVEL; --      same as above.
signal out1           :  SWITCH_LEVEL; --  Signal feeding Nand o/p to D2A.
signal stop_after1    :  TIME;   --  Signal passing the simulation stop_time.
signal stop_after2    :  TIME;   --      same as above.
signal incriment1     :  TIME := 0 ns;  --  Time incriment for simulation step.
signal incriment2     :  TIME := 0 ns;  --     same as above.
signal analog         :  real:= 0.0;          --  Analog o/p from D2A entity.
signal t1             :  TIME;       --  Passing Time values to all the entities.
signal t2             :  TIME;
signal v_gate_src1    :  real;
signal v_gate_src2    :  real;
signal extern_feed1   :  boolean;
signal error_value1   :  VALUE_ARRAY(1 to 100);
signal extern_feed2   :  boolean;
signal error_value2   :  VALUE_ARRAY(1 to 100);
signal FILE_FLAG1     :  boolean := TRUE;
signal FILE_FLAG2     :  boolean := FALSE;
signal lowpass_feed1 :  boolean;
signal lowpass_feed2 :  boolean;
signal lowpass_input1:  VALUE_ARRAY(1 to 100);
signal lowpass_input2:  VALUE_ARRAY(1 to 100);
signal DC_IN1:  VALUE_ARRAY(1 to 100);
signal DC_IN2:  VALUE_ARRAY(1 to 100);
signal DC_FLAG1:  boolean;
signal DC_FLAG2:  boolean;
signal stop_sim:  boolean := false;


file FILE2 : TEXT is out "STD_OUTPUT";

begin

  ana1 : anavhdl
          port map (v_output => input1, stop => stop_after1,
                    incri => incriment1, T_IN => t1, Vgs => v_gate_src1,
                    extern_feed => extern_feed1, FILE_FLAG => FILE_FLAG1,
           lowpass_feed => lowpass_feed1, lowpass_input =>
           lowpass_input1, error_value => error_value1,
           DC_IN => DC_IN1, DC_FLAG => DC_FLAG2);
```

```
        lowpass_feed : in boolean;
        lowpass_input : in VALUE_ARRAY;
          error_value  : in VALUE_ARRAY);
        DC_IN          : in VALUE_ARRAY;
        DC_FLAG        : in boolean);
  end component;

  component A2D
    generic (hi_cutoff : real;
             lo_cutoff : real);
    port (LOGIC_OUTPUT : out SWITCH_LEVEL;   -- Digital voltage output.
          ANALOG_INPUT : in real;            -- Analog voltage output.
          T_input      : in TIME;            -- Circuit simulation time.
          stop_at      : in TIME;            -- Simulation end time.
          Vgate_src    : in real);
  end component;

  component D2A
    port (LOGIC_IN      : in SWITCH_LEVEL;   -- Digital voltage input.
          Time_in       : in TIME;           -- Simulation time from AnaVHDL.
          stop_after    : in TIME;           -- Simulation end time.
          ANALOG_OUTPUT : out real);         -- Analog voltage output.
  end component;

  component nand_gate
    generic(width : integer; delay_time : time);
    port (input : in SWITCH_LEVEL_VECTOR((width-1) downto 0);
          output : out SWITCH_LEVEL);
  end component;

for all : anavhdl use entity work.anavhdl(anavhdl);
for all : A2D use entity work.A2D(A2D);
for all : D2A use entity work.D2A(D2A);
for all : nand_gate use entity work.nand_gate(simple_nand);

  signal input1    :  VALUE_ARRAY(1 to 100);   -- Signal feeding the A2D
                                                -- entity from AnaVHDL.
  signal input2    :  VALUE_ARRAY(1 to 100);   --     same as above
  signal in1       :  SWITCH_LEVEL; -- Signal feeding Nand entity from A2D.
```

```
      lowpass_feed1 <= false;
      lowpass_feed2 <= false;
      write(K, "Vdc(1) = ");
    write(K,v_output1(1));
    write(K, "  ");
    DC_IN2 <= v_output1;
    write(K, "opamp_out = ");
    write(K,v_output2(2));
    writeline(OUTPUT,K);
    wait;
    stop_sim <= true;

end process p1;

end DAC;
```

## Inverter-Nand gate Combination

```
use work.COMPUTE_PACK.all;
use work.IO_PACK.all;
use work.INTERFACE_PACK.all;
use STD.TEXTIO.all;

entity system is
end system;

architecture system of system is

  component anavhdl
    port (v_output     : out VALUE_ARRAY;    -- Analog output.
          T_IN         : out TIME;           -- Circuit simulation Time.
          stop         : out TIME;           -- Simulation end time.
          incri        : out TIME;           -- Time incriment step.
          Vgs          : out real;
          extern_feed  : in boolean;
          FILE_FLAG    : in boolean;
```

```
  ana1 : anavhdl
        port map (v_output => v_output1, T_IN => T_IN1,
                  stop => stop_at1, extern_feed => extern_feed1,
                  error_value => error_value, Vgs => Vgs1,
                  lowpass_feed => lowpass_feed1,
                  lowpass_input => lowpass_input1,
                  FILE_FLAG => FILE_FLAG1, DC_IN => DC_IN1,
           DC_FLAG => DC_FLAG1);

  ana2 : anavhdl
        port map (v_output => v_output2, T_IN => T_IN2,
                  stop => stop_at2,
                  extern_feed => extern_feed2,
                  error_value => error_value, Vgs => Vgs2,
                  lowpass_feed => lowpass_feed2,
                  lowpass_input => lowpass_input2,
                  FILE_FLAG => FILE_FLAG2, DC_IN => DC_IN2,
           DC_FLAG => DC_FLAG2);



stop_control : process

begin

  wait until stop_sim = true;
  assert false report "Stopping simulation" severity failure;

end process stop_control;


p1 : process

    variable K : LINE;
    variable i : positive;

 begin
```

```
      DC_FLAG       : in boolean);
  end component;


for all : anavhdl use entity work.anavhdl(anavhdl);

   signal v_output1      : VALUE_ARRAY(1 to 100);
   signal v_output2      : VALUE_ARRAY(1 to 100);
   signal T_IN1          : TIME;
   signal T_IN2          : TIME;
   signal stop_at1       : TIME;
   signal stop_at2       : TIME;
   signal LOGIC_OUTPUT : SWITCH_LEVEL;
   signal clock          : SWITCH_LEVEL;
   signal dff_out        : SWITCH_LEVEL;
   signal stop_sim       : boolean := false;
   signal anaout         : real;
   signal extern_feed1  : boolean := false;
   signal extern_feed2  : boolean := false;
   signal error_value  : VALUE_ARRAY (1 to 100);
   signal Vgs1           : real;
   signal Vgs2           : real;
   signal sigdel_real_out : VALUE_ARRAY (1 to 100);
   signal lowpass_feed1 : boolean := false;
   signal lowpass_feed2 : boolean := false;
   signal lowpass_input1  : VALUE_ARRAY (1 to 100);
   signal lowpass_input2  : VALUE_ARRAY (1 to 100);
   signal lowpass_out : real;
   signal FILE_FLAG1 : BOOLEAN := TRUE;
   signal FILE_FLAG2 : BOOLEAN := FALSE;
   signal DC_IN1      : VALUE_ARRAY(1 to 100);
   signal DC_IN2      : VALUE_ARRAY(1 to 100);
   signal DC_FLAG1    : boolean := FALSE;
   signal DC_FLAG2    : boolean := TRUE;

file FILE2 : TEXT is out "STD_OUTPUT";

begin
```

# Appendix B

# Examples

## 8-bit DAC

```
use STD.TEXTIO.all;
use work.COMPUTE_PACK.all;
use work.IO_PACK.all;
use work.INTERFACE_PACK.all;

entity DAC is
end DAC;

architecture DAC of DAC is

  component anavhdl
    port (v_output      : out VALUE_ARRAY;     -- Analog output.
          T_IN           : out TIME;       -- Circuit simulation Time.
          stop           : out TIME;       -- Simulation end time.
          incri          : out TIME;     -- Time incriment step.
          Vgs            : out real;
          extern_feed  : in boolean;
          FILE_FLAG     : in boolean;
          lowpass_feed  : in boolean;
          lowpass_input  : in VALUE_ARRAY;
          error_value  : in VALUE_ARRAY;
      DC_IN          : in VALUE_ARRAY;
```

```
    if probe  = 0 then
     if delta_real  <= Ref_time_real then  -- when delta is equal to Ref_time.
       wait for 0 ns;
       ANALOG_OUTPUT <= digi2ana(LOGIC_IN,t_real);
     elsif delta_real > Ref_time_real then  -- delta is greater than Ref_time.
       wait for 0 ns;
       ANALOG_OUTPUT <= digi2ana_delay(LOGIC_IN,t_real,delta_real);
     end if;

-- Calling conversion functions when the digital value goes from '1' to '0'.

    elsif probe = 1 then
     if delta_real <= Ref_time_real then   -- delta is equal to Ref_time.
       wait for 0 ns;
       ANALOG_OUTPUT <= digi2ana_inv(LOGIC_IN,t_real);
     elsif delta_real > Ref_time_real then  -- delta is greater than Ref_time.
       wait for 0 ns;
       ANALOG_OUTPUT <= digi2ana_delay_inv(LOGIC_IN,t_real,delta_real);
     end if;
    end if;
   wait for 1 ns;

 end process p3;

end D2A;
```

```
        d1 := t;
        wait until LOGIC_IN = '0' or LOGIC_IN = '1';
        d2 := t;
      end if;
      delta <= abs(d1 - d2);
      wait for 1 ns;

  end process p2;


  p3  : process

-- This process converts digital values into analog voltage values for three
-- logic levels '0','1' & 'X'.

      variable delta_real : real;
      variable t_real     : real;
      variable Ref_time   : TIME := 1 ns;       -- Constant reference time.
      variable Ref_time_real: real;
      variable j          : positive;
      variable probe      : integer;
      variable K          : LINE;               -- debugg variable.

    begin

      t1 <= Time_in;                      -- Simulation time is assigned to a signal.
      t_real := TIME_TO_REAL(t1);                    -- Time t to real.
      Ref_time_real := TIME_TO_REAL(Ref_time);       -- time Ref_time to real.
      delta_real := TIME_TO_REAL(delta);             -- Time delta to real.
      if (t1 >= 0 ns and t1 < (stop - 1 ns)) then
       wait for 0 ns;
       if logic_value(1) = '0' then
          probe := 0;
       elsif logic_value(1) = '1' then
          probe := 1;
       end if;
      end if;

-- Calling conversion functions when the digital value goes from '0' to '1'.
```

```
file FILE2 :TEXT is out "STD_OUTPUT";

begin


 p1  : process

        variable j    : positive;
        variable K    : LINE;              -- debugg variable.

   begin

    t <= Time_in;                          -- Simulation time is assigned to a signal.
    stop <= stop_after;

-- To store the input logic values into the array.

    if (t >= 0 ns and t < (stop - 1 ns)) then  -- stop_after is the time after
                                                -- which the simulation has to stop.
        logic_value(j) <= LOGIC_IN;     -- logic_value = array for storing the
                                         -- digital input values.
        j := j + 1;
    end if;
    wait for 1 ns;

 end process p1;


 p2 : process

        variable d1 : time := 0 ns;        -- d1 : time at which 'X' starts.
        variable d2 : time := 0 ns;        -- d2 : time at which 'X' ends.
        variable j  : positive;
        variable K  : LINE;

     begin


     if LOGIC_IN = 'X' then
```

```
            LOGIC_OUTPUT <= threshold(ANALOG_INPUT);
            wait for 0 ns;
          else
            LOGIC_OUTPUT <= high_imped(current_in1);
          end if;
        end if;
      end if;
      wait for 1 ns;
 end process p2;

end A2D;
```

## D2A Module

```
use work.COMPUTE_PACK.all;
use work.IO_PACK.all;
use work.INTERFACE_PACK.all;
use STD.TEXTIO.all;

entity D2A is
  port (LOGIC_IN      : in SWITCH_LEVEL;      -- Digital voltage input.
        Time_in       : in TIME;              -- Simulation time from AnaVHDL.
        stop_after    : in TIME;              -- Simulation end time.
        ANALOG_OUTPUT : out real);            -- Analog voltage output.
end D2A;

architecture D2A of D2A is

 signal logic_value : logic_array(1 to 1000);  -- Array for storing digital input
 signal ana_out     : real;                    -- Analog voltage output.
 signal t           : TIME;
 signal t1          : TIME;
 signal t2          : TIME;
 signal stop        : TIME;
 signal delta       : TIME := 0 ns;
```

```
variable actual_delta      : real := 0.0;   -- time lapse between lo and
variable lo_cut_time       : real := 0.0;   --             hi cutoff points.
variable hi_cut_time       : real := 0.0;
variable Ref_time_real     : real;
variable t_real            : real;
variable i                 : integer := 1;
variable K                 : LINE;          -- debugg variable.


begin

    t <= T_input;            -- Time value is assigned from port to signal.
    volt_out <= ANALOG_INPUT;
    t_real := TIME_TO_REAL(t);      -- time to real conversion of time 't'.
    Ref_time_real:= TIME_TO_REAL(Ref_time);
    actual_delta := TIME_TO_REAL(temp);

-- when the actual_delta is greater than Ref_time then the inertial delay is
-- used to assign the logic value to the port.

    if Vgate_src < 0.8 then
       LOGIC_OUTPUT <= 'Z';
    else
     if t >= 0 ns then
      if actual_delta > Ref_time_real then
        wait for 0 ns;
        LOGIC_OUTPUT <= threshold(ANALOG_INPUT) after Ref_time ;

-- when the actual_delta is less than the Ref_time the logic value is assigned
-- to the port immediately with either '0' or '1'.

      elsif actual_delta < Ref_time_real then
        wait for 0 ns;
        LOGIC_OUTPUT <= threshold_1(ANALOG_INPUT);

-- when the actual_delta is equal to the Ref_time then logic value is assigned
-- to the port immediately with values '0', '1', or 'X'.

      elsif actual_delta = Ref_time_real then
```

```
signal temp          : TIME := 0 ns;


file FILE2 :TEXT is out "STD_OUTPUT";



begin

p1  : process

     variable i  : integer := 1;
     variable d1 : time := 0 ns;
     variable d2 : time := 0 ns;
     variable K  : LINE;

     begin

        if ANALOG_INPUT <= hi_cutoff then
           d1 := t;
           wait until (ANALOG_INPUT <= lo_cutoff or ANALOG_INPUT >= hi_cutoff);
           if ANALOG_INPUT <= lo_cutoff then
              d2 := t;
           else
              wait until ANALOG_INPUT <= hi_cutoff;
              d1 := t;
              wait until ANALOG_INPUT <= lo_cutoff;
              d2 := t;
           end if;
           temp <= abs(d1 - d2);
        end if;
        wait for 1 ns;

end process p1;


p2  : process

     constant Ref_time          : TIME := 1 ns;  -- Constant Reference time.
```

# Appendix A

# Interface modules

## A2D Module

```
use work.COMPUTE_PACK.all;
use work.IO_PACK.all;
use work.INTERFACE_PACK.all;
use STD.TEXTIO.all;

entity A2D is
  generic (hi_cutoff : real;
        lo_cutoff : real);

  port (LOGIC_OUTPUT : out SWITCH_LEVEL;    -- Digital voltage output.
        ANALOG_INPUT : in real;             -- Analog voltage input.
        T_input      : in TIME;             -- circuit simulation time.
        stop_at      : in TIME;             -- Simulation end time.
        Vgate_src    : in real);
end A2D;

architecture A2D of A2D is

 signal volt_out    : real;
 signal LOGIC_OUT   : SWITCH_LEVEL;
 signal current_in1 : real;
 signal t           : TIME;
```

[Tuin88] P. W. Tuinenga, SPICE *: A Guide to Circuit Simulation and Analysis Using PSpice,* Prentice Hall, New Jersey, 1988.

[Webe91] S.Weber, *Digital meets analog in a robust mixed market*, Electronics, pp. 53-56, Mar. 1991.

[Zhou92] Wenying Zhou, *Investigation of Extension to a Digital Behavior-Level Hardware Description Language to Support Analog and Mixed-Mode Simulation*, MS Thesis, University of Cincinnati, April 1992.

# Bibliography

[ADPS89] E.L. Acuna, J.P. Dervenis, A.J. Pagones & R.A. Saleh, *i*SPLICE*3 : a new simulator for mixed analog/digital circuits*, in Proceedings of IEEE 1989 Custom Circuits Conference, San Diego, CA, pp. 13.1.1-4, May 1989.

[Breu75] M.A. Breuer, Ed., *Digital System Design Automation: Languages, Simulation and Data Base*, Computer Science Press, 1975.

[Brya80] R.E. Bryant, *An Algorithm for* MOS *logic simulation*, LAMBDA pp. 46-53, 4th quarter 1980.

[HaSt91] R.E. Harr & A.G. Stanculescu, *Applications of* VHDL *to Circuit Design*, Kluwer Academic Publishers, 1991.

[InSL84] B. Infante, A. Sanders, E. Lock, *Hierarchical Modeling in a Multi-level Simulator*, International Conference on Computer-Aided Design, pp. 39-41, Santa Clara, CA. 1984

[Morr89] T.H. Morrin, *Mixed-mode simulation for time-domain fault analysis*, in 1989 International Test Conference, pp 231-241, 1989.

[OdNa86] P. Odryna & S.R. Nassif, *The* ADEPT *timing simulation Algorithm*, VLSI System Design, March 1986.

[OdNC88] P. Odryna, K. Nazareth & C. Christensen, *A Workstation-Based Mixed-Mode Simulator*, in Proceedings IEEE 23rd Design Automation Conference, Las Vegas, Nevada, pp. 186-192, June 1988.

[SaNe90] Resve A. Saleh & A. Richard Newton *Mixed-Mode Simulation*, Kulwer Academic Publishers., 1990.

[StBr89] Stanisic B.R. and Brown M.W., *VHDL Modeling for Analog-Digital Hardware Designs*, in Proceedings IEEE Int. Conf. Computer Aided Design, pp. 184-187, Nov. 1989.

possible to exchange data between different partitions.

With the implementation of *Interfaces* which has two separate modules A2D & D2A, it is possible to exchange data between analog and digital partitions. The requirement for handling both slow and fast varying signals during type conversion is also satisfied. The simulator supports the IEEE Standard 1164 digital logic which is acceptable by the industry.

The experimental results in the previous chapter shows that mixed-mode simulation can be conducted with a reasonable amount of accuracy. The execution time taken to simulate is large which can be improved by optimizing the algorithms used in AnaVHDL.

## 6.2 Future Work

Future work can be concentrated on improving the execution time. This can be done either by replacing the algorithms in AnaVHDL with better algorithms like sparse matrix data structures, waveform relaxation method and parallel algorithms or by embedding the Analog simulation capability into the existing VHDL simulators.

More sophisticated algorithms can be used in the *interfaces* for signal translation such as multiple threshold method, sliding threshold method or detective hysteresis methods.

Presently, the Extended AnaVHDL supports only MOSFET models other than passive components. The bipolar-junction transistors (BJT) model, junction field effect transistor (JFET) model, and insulated-gate field effect transistor (IGFET) model are still needed to be implemented to make it a general purpose simulator. Only a subset of SPICE options are implemented in AnaVHDL which can be further enhanced by adding all possible options in SPICE.

# Chapter 6

# Conclusions

## 6.1 Summary

A VHDL based mixed-mode simulator has been developed which is the combination of Extended AnaVHDL and *Interfaces*. To achieve the mixed-mode simulation capability following were the major requirements:

1. To enhance AnaVHDL to support multiple analog circuit simulation in the same simulation environment and to reorganize the entity by adding input/output ports for data communication between different partitions.

2. To implement signal translation modules (*interfaces*) for time and type conversion of analog and digital signals which should help in data exchange between analog and digital partitions in a mixed system.

With the implementation of the Extended AnaVHDL and *interfaces*, all the requirements were satisfied. The Extended AnaVHDL is capable of simulating multiple analog circuits in a single environment which is the primary requirement of any mixed-mode simulator. Also by the addition of external ports to the AnaVHDL it is

1. One major limitation is the execution time which increases dramatically with the complexity of the circuit under simulation. The algorithms in AnaVHDL have to be optimized to improve the execution speed.

2. Number of analog circuits that can be simulated in the same simulation environment using the Extended AnaVHDL is limited to 10.

| Benchmark | Execution Time (sec) | Percentage Error |
|---|---|---|
| 8-bit DAC | 748.18 | 0.0 |
| Inv-Nand Combo | 52.8 | 3.3 |
| Sigma-Delta Converter | 1003.9 | 1.22 (RMS error) |

Table 5.3: Summary of Execution Time & Simulation Accuracy of the Extended AnaVHDL

| Circuits | Errors (VHDL-SPICE)/SPICE % | Execution Time |
|---|---|---|
| One Stage RCL | 0.21 | 11.2 |
| Single NMOS | 0.02 | 36.5 |
| Single PMOS | 0.01 | 35.6 |
| CMOS Inverter | 0.05 | 62.9 |
| CMOS Nand Gate | 0.60 | 173.3 |
| CMOS Op-amp | 0.05 | 716.3 |

Table 5.4: Simulation Accuracy and Execution Time of AnaVHDL[Zhou92]

### 5.1.6 Execution Time Analysis

It is evident from the Table 5.3 that as the complexity of the circuit under simulation increases, the execution time also increases. This is due to the inherent slowness of AnaVHDL which can be seen in Table 5.4 [Zhou92].

## 5.2 Limitations

The experimental results shows that the Extended AnaVHDL and *interfaces* combined together can perform mixed-mode simulation. The results obtained are of reasonable accuracy but there exists some limitations:

| Experiment | Technology Type | Simulation End Time (ns) | Simulation Step Time (ns) | Reference Time (ns) |
|---|---|---|---|---|
| 8-bit DAC | CMOS | 300 | 5 | 1 |
| Inv-Nand Combo | CMOS | 300 | 5 | 1 |
| Sigma-Delta Converter | CMOS | 300 | 5 | 1 |

Table 5.2: User definable values used for different experiments

demonstrates that the output is following the input pulse. The output pulse is not showing any time lag or lead with respect to the input pulse, which helps in inferring that performance of the Extended AnaVHDL and *interfaces* are satisfying the requirements.

### 5.1.4 User defined values

The Extended AnaVHDL and *interfaces* allow user to define values for the circuit under simulation. The Table 5.2 gives all the user definable values for the 3 experiments explained above.

### 5.1.5 Overall Simulation Accuracy

As said above a set of benchmark circuit have been simulated using the Extended AnaVHDL and *interfaces*. Table 5.3 gives the summary of the results obtained with emphasis on simulation accuracy and speed. The overall simulation error in the mixed-mode simulation conducted is with in 3.3% relative to the actual result expected. The input/output graphs shown above also confirms that the analog and digital domains are communicating fairly well for the circuits simulated.

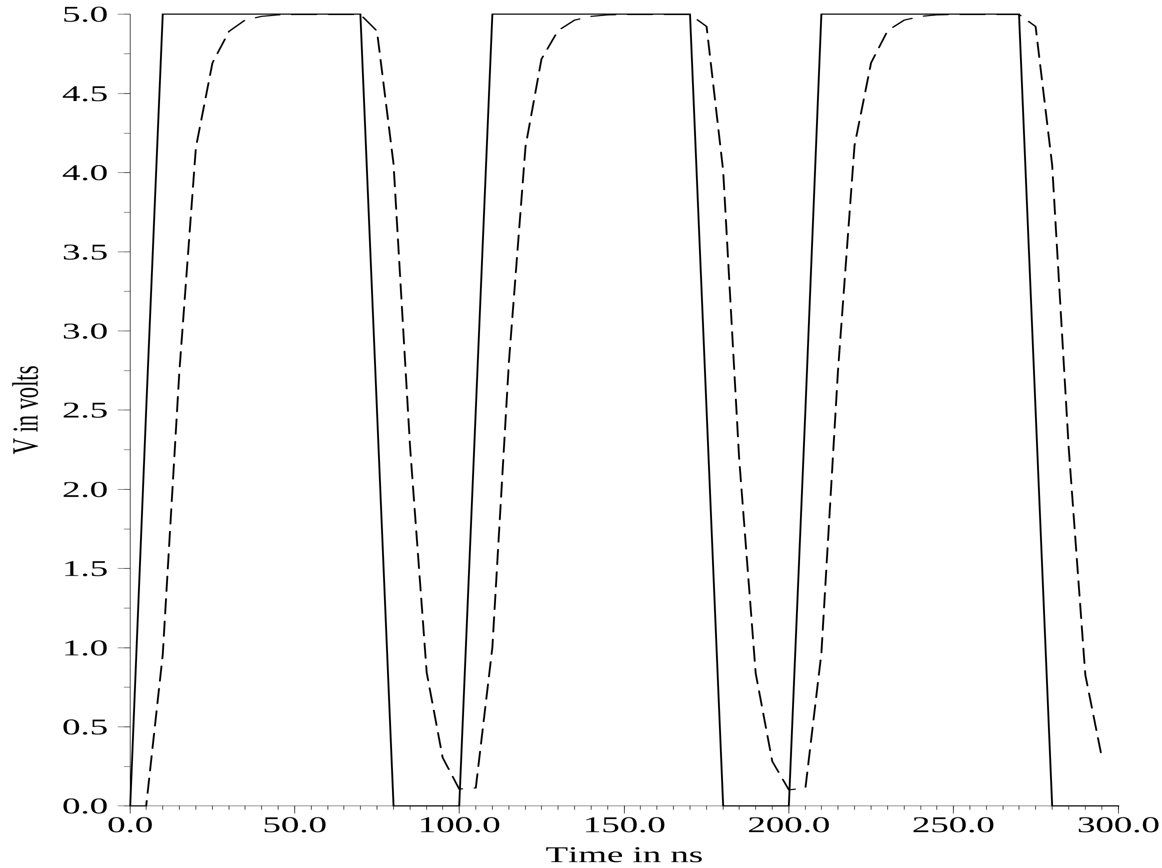
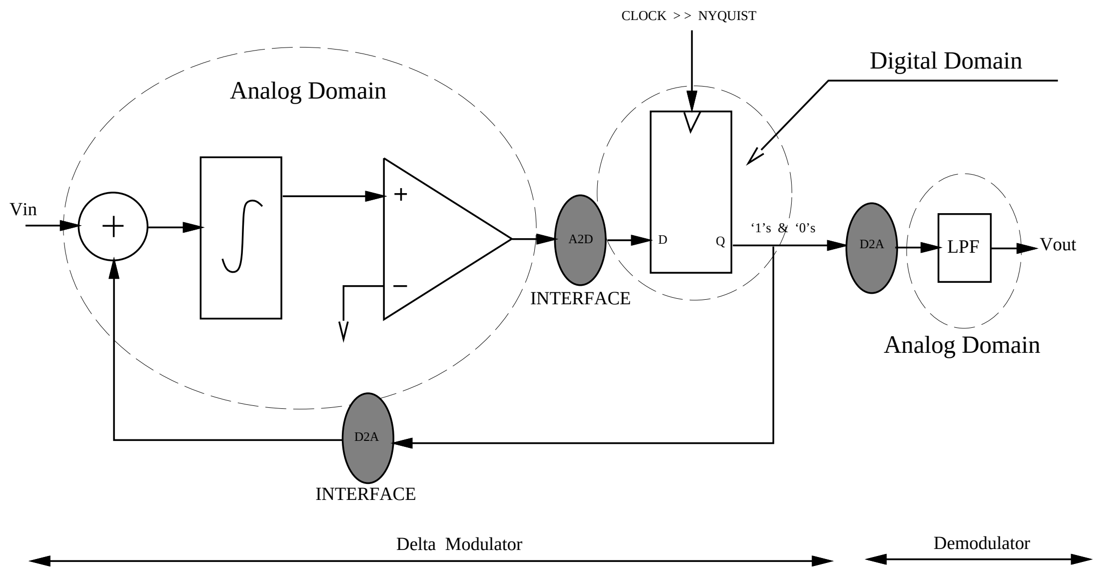
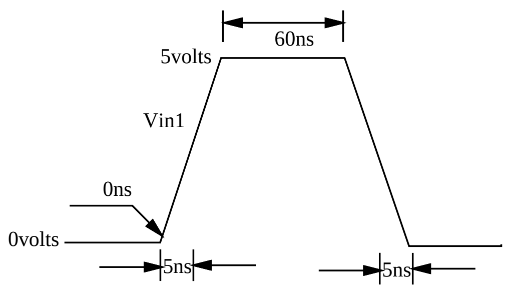



Figure 5.5: Sigma-Delta Converter Simulation Results




(a)




(b)

Figure 5.4: Sigma-Delta Converter

and output pulses. Simulation is done for 300ns and the relative error between the expected result and the simulation result is within 3.3%.

### 5.1.3 Sigma-Delta Converter Experiment

#### Circuit Diagram

Sigma-Delta conversion is one of the most significant advances in DSP (digital signal processing). The Figure 5.4(a) shows a simple Sigma-Delta converter which is one of the benchmarks. As shown in the figure it has two analog domains and one digital domain. It is combination of comparator, integrator, operational amplifier, D Flip-flop, and a low pass filter. The input waveform is a pulse of 5volts with rise and fall times of 10ns, pulse width of 60ns and period of 100ns (see Figure 5.4(b)). All components are modeled at the circuit level except for D Flip-flop which is a structural VHDL description. The analog domains are simulated using the Extended AnaVHDL package where as the digital domain is simulated using the standard VHDL simulator. Both the domains are simultaneously simulated in the same simulation environment. The analog and digital domains are integrated using the A2D and D2A signal conversion modules as shown in the Figure 5.4(a). The simulation is conducted between 0ns and 300ns time interval.

#### Simulation Results

The simulation results has been shown in the Figure 5.5. The solid line indicates the input pulse and the broken line indicates the output pulse. As said above the simulation is conducted for a time interval of 300ns. The RMS error between the input and the simulation output is 1.22%. The Sigma-Delta converter is a combination of modulator and demodulator as shown in the Figure 5.4(a), hence the output should be an exact reproduction of the input waveform. The graph in Figure 5.4

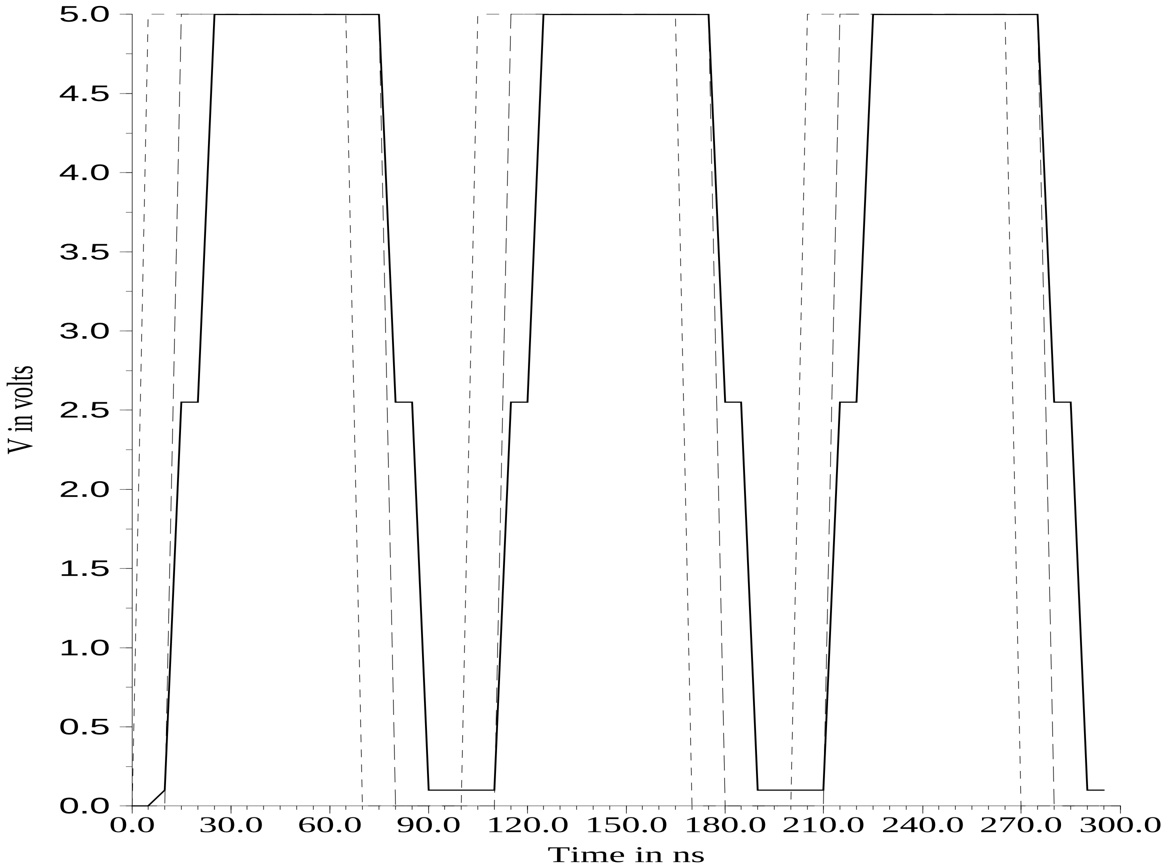



Figure 5.3: Inverter-Nand Gate Combination Simulation Results

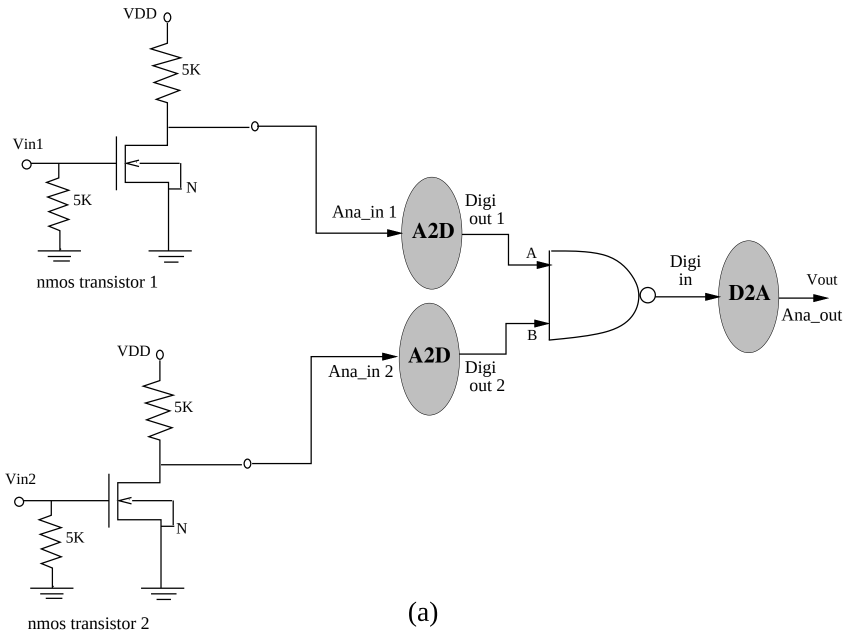
| Digital input msb lsb | Simulation output in volts | Expected output |
|---|---|---|
| 0000 0011 | 0.05859375 | 0.05859375 |
| 0000 1000 | 0.15625 | 0.15625 |
| 0000 1111 | 0.29296875 | 0.29296875 |
| 0001 1111 | 0.60546875 | 0.60546875 |
| 0011 0000 | 0.9375 | 0.9375 |
| 1000 0000 | 2.5 | 2.5 |
| 1100 0000 | 3.75 | 3.75 |
| 1111 0000 | 4.6875 | 4.6875 |
| 1111 1110 | 4.9609375 | 4.9609375 |
| 1111 1111 | 4.9804688 | 4.9804688 |

Table 5.1: AnaVHDL performance on 8-bit DAC example for selected inputs

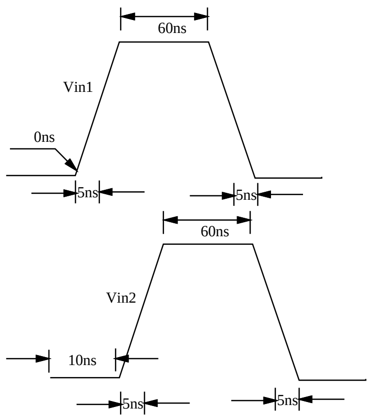
gate is passed through digital to analog conversion module (D2A) and the output of D2A is verified for accuracy. Vin1 and Vin2 are the input voltage pulses with low of zero volts, high of 5.0volts, 5ns rise and fall times (see Figure 5.2(b)). Vin1 has zero delay time whereas Vin2 is delayed by 10ns. The period of both pulse waveforms are 100ns with pulse width of 60ns.

### Simulation Results

The simulation results are shown in the Figure 5.3. The solid line indicates the output of the circuit at the node Vout. The dotted line and broken line indicates the two input pulses Vin1 and Vin2 respectively. The two input pulses are out of phase in time by 10ns. The circuit is required to obey the truth table shown in the Figure 5.2(c). From the input/output graph in Figure 5.3 it is evident that the circuit is obeying the truth table. Thus can be inferred that the communication between the analog and digital domain is good and the Extended AnaVHDL & *interfaces* are functioning as required. There is a time phase difference of 10 ns between the input




(a)




(b)

| Vin1 | Vin2 | A | B | Vout |
|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |

(c)

Figure 5.2: An Inverter-Nand Gate Combination

the digital partition. 'A' and 'B' are the digital inputs to the nand gates. The OPAMP is used for the buffering purpose. The ladder network forms one analog domain where as the OPAMP forms the other analog domain. Both of them are simulated in the same simulation environment.

### Simulation Results

The 8-bit DAC is simulated for all combination of digital input. Table 5.1 some shows selected inputs to the DAC at the nodes numbered 9 through 16 (which are the outputs of the nand gates), the corresponding simulation outputs and expected outputs calculated analytically. The equation used to analytically calculate expected outputs is:

$$\{(2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0) \div 2^8\} \times V_{in} \tag{5.1}$$

In the above equation $V_{in}$ = 5volts. The simulated outputs are 100% accurate compared to the analytical results (Table 5.1). The results shows that the two analog partitions (R-2R ladder network and OPAMP) are exchanging data fairly well and the communication between the analog and digital partitions is good. This helps in inferring that the Extended AnaVHDL is performing as required.

## 5.1.2 Inverter-Nand Gate Combination Experiment

### Circuit Diagram

The Figure 5.2(a) shows a combination of inverters at the transistor level and a Nand gate. The inverters used in the circuit are simple nmos inverters and forms the analog domain. The Nand gate used forms the digital domain which is a behavioral description in VHDL. The Nand gate has two inputs fed by the inverters after the signal translation from analog domain to digital domain. The output of the Nand

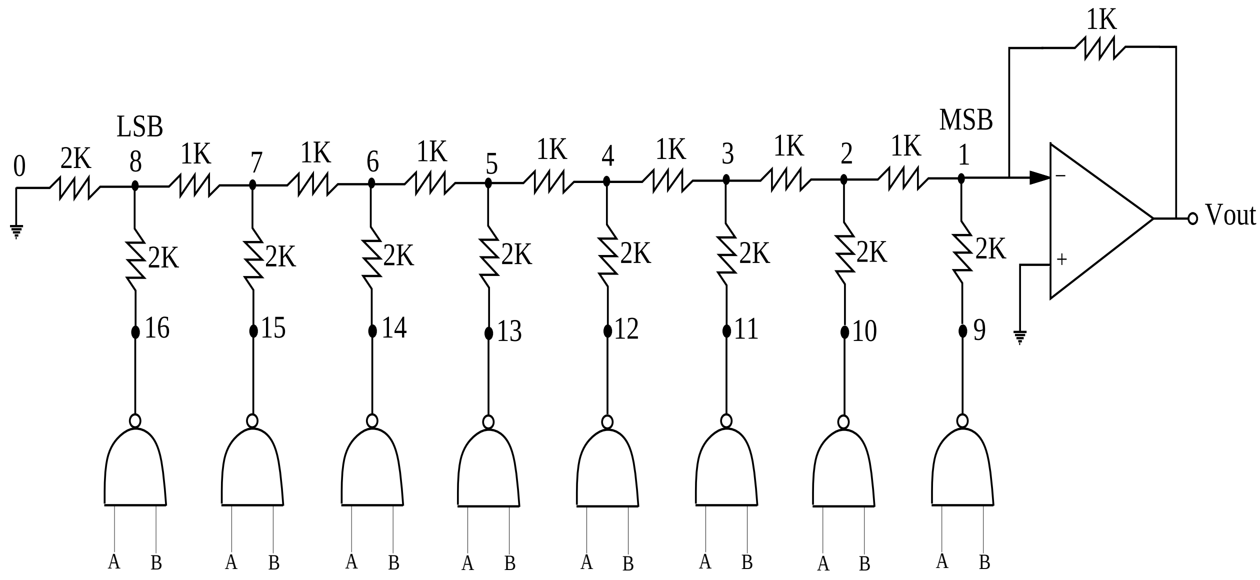



Figure 5.1: 8-bit Digital to Analog Converter (DAC)

sible difficulties in the Extended AnaVHDL and *interfaces*. Since it has a combination of digital and analog partitions with feedback paths, it served as one of the very good benchmarks. Each example represents an experiment which are fully discussed below. All the simulations are conducted using MCC VHDL simulator version 3.2.

### 5.1.1 8-bit DAC

#### Circuit Diagram

The Figure 5.1 shows an 8-bit DAC which is based on the National Semiconductor CMOS MicroDAC-0830 design. It consists of an R-2R ladder network (which uses voltage switching approach) and an operational amplifier (OPAMP). This circuit is modeled in such a way that the R-2R network forms one analog partition and the OPAMP forms the other analog partition. The Nand gates feed the DAC and it forms

# Chapter 5

# Results and Analysis

A set of experimental benchmarks is discussed in this chapter. The objective is to apply these benchmarks to Extended AnaVHDL and *Interfaces* and verify that it meets all the requirements described in Chapter 4. The results of simulation are discussed and analyzed with emphasis on the simulation accuracy and the simulation speed.

## 5.1 Simulation Accuracy Evaluation

The benchmark circuits are divided into three groups: 1) 8-bit DAC(Digital to Analog Converter), 2) Inverter at transistor level in combination with a Digital gate (NAND gate), and 3) Sigma-Delta converter.

The 8-bit DAC is a very common analog/digital circuit. It has a passive part formed by the R-2R ladder network and an active part formed by the OPAMP. The Inverter-Nand gate combination circuit is used to fully characterize the *interfaces*. This is the first benchmark circuit used and is being continuously used through out this work. The Sigma-Delta converter is used to test the time phasing and other pos-

## 4.8 Use of AnaVHDL and Interface

The Extended AnaVHDL and the *Interfaces* can be used together to describe and simulate a mixed-mode VHDL system. To use these packages user has to define an input file for each analog circuit. The behavior or the structure of the digital circuit has to be described in a separate file. A main *process* can be created for the mixed-mode system where in the analog circuits, the digital circuits and the *interfaces* are instantiated as components and the necessary interconnections are made. With in the main *process* Extended AnaVHDL package is referenced for performing circuit analysis and the *interface* package is referenced for performing signal translations. The digital circuits are simulated using the standard VHDL libraries. Communication between different components are done through the signals which represent the interconnections.

| Technology Type | Standard Logic | Logic Type | Threshold Limits in volts | |
|---|---|---|---|---|
| | | | High | Low |
| CMOS | '0' | Forcing '0' | 1.7 | 0.0 |
| | '1' | Forcing '1' | 5.0 | 3.7 |
| | 'X' | Forcing '0' or '1' | 3.7 | 1.7 |
| | 'Z' | High Impedance | (Vgs $<$ 0.8) | – |
| TTL | '0' | Forcing '0' | 0.4 | 0.0 |
| | '1' | Forcing '1' | 5.0 | 2.4 |
| | 'X' | Forcing '0' or '1' | 2.4 | 0.4 |
| ECL | 'L' | Weak '0' | -1.7 | -1.9 |
| | 'H' | Weak '1' | -0.8 | -1.045 |
| | 'W' | Weak '0' or '1' | -1.045 | -1.7 |
| | 'U' | Un-initialized | 0.0 | -0.8 |
| | '-' | Don't Care | -1.9 | – |

Table 4.1: IEEE Standard Logic 1164 & Threshold limits

| Standard Logic | Resolved Logic |
|---|---|
| 0 | 0 |
| 1 | 1 |
| X | X |
| Z | Z |
| L | 0 |
| H | 1 |
| W | X |
| U | X |
| - | Z |

Table 4.2: Resolution of Standard Logic into Four-level Logic

**Algorithm 4.2**

```
Input : Port (Digital input, Time, Simulation end time);
        Generic(Reference time, Tech_type);

Output : Port (Analog output)

process DELAY
  /* Calculates the time for which the input digital voltage remains at 'X' */

begin
  if (Digital input = 'X') then
     set 1st variable = Time
     wait until (Digital input = '0' or '1')
     set 2nd variable = Time
  end if;
  delta = abs(1st variable - 2nd variable)
end;

process CONVERT
      /* Converts digital signal to analog signal */

begin
  Check the incoming voltage if it is of rising or falling type
  if (digital input is a rising voltage) then
     set probe = TRUE          /* probe is a variable of type boolean := FALSE */
  end if;
   if (Current_Simulation_Time > 0ns AND < Simulation_End_TIME) then
  {
    if (probe = TRUE) then
      if (delta <= Reference time) then
         Analog output <= DIGI2ANA(Digital input, Tech_type)
      elsif (delta > Reference time) then
         Analog output <= DIGI2ANA_DELAY(Digital input, Tech_type)
      end if;
    else
      if (delta <= Reference time) then
         Analog output <= DIGI2ANA_INV(Digital input, Tech_type)
      elsif (delta > Reference time) then
         Analog output <= DIGI2ANA_DELAY_INV(Digital input, Tech_type)
      end if;
    end if;
  }
end;
```

for which the input digital signal remains at 'X'. The delta is used in calculating the speed of the input digital signal comparing it with the reference time. The process CONVERT is used in converting the digital signal to analog signal. Initially the input digital signal is verified for either rising or falling type. Different conversion functions are invoked depending on the delta and rising or falling type of the input signal. DIGI2ANA, DIGI2ANA_DELAY, DIGI2ANA_INV and DIGI2ANA_DELAY_INV are the functions used.

## 4.7 The Standard Logic 1164

### 4.7.1 Rationale for using 1164

The signal translation done by an interface either A2D or D2A supports the IEEE Standard 1164 logic levels. The reason for supporting 1164 logic in signal translation is to permit acceptance of AnaVHDL in the industry. The standard 1164 has nine different logic levels. The table 4.1 summarizes the logic levels and the voltage threshold limits used by an interface for different technology types.

### 4.7.2 Incorporating 1164 in AnaVHDL

Even though the A2D conversion modules supports the nine IEEE standard logic levels, the output is resolved to four level logic ('0', '1','X' & 'Z'). This is done to ensure the proper operation of digital gates. Conversion function CONVT_TO_01XZ (see Figure 4.8) does the necessary conversion from the nine level to four level logic output. The table 4.2 shows how the standard logic 1164 is resolved to four level logic.

**Algorithm 4.1**

**process** CONVERT

*/* Converts the analog signal into digital signal */*

**begin**
**if** (Vgs < 0.8) **then** */* Vgs is Gate to Source voltage of MOSFET */*
Digital output = ‘Z’
**else**
if (Current_simulation_Time > 0ns AND < Simulation_End_Time)
{
**if** (delta > Reference time) **then**
*/* Inertial Delay signal assignment */*
Digital output <= THRESHOLD(Analog Input, Tech_type) **after** Reference Time
**elsif** (delta = Reference time) **then**
Digital output <= THRESHOLD(Analog Input, Tech_type)
**else**
*/* when delay is less than Reference time */*
Digital output <= THRESHOLD_1(Analog Input, Tech_type)
**end if**;
}
**end**;

**Algorithm 4.1**

Input : Port (Analog input, Time, simulation end time, Gate to source Voltage
of transistor);
Generic (Reference Time, Tech_type);

Output : Port (Digital output)

**process** DELAY
*/* Calculates the time delay for analog signal to pass between hi_cutoff
and lo_cutoff limits */*
**begin**
*/* hi_cutoff and lo_cutoff are the high and low threshold values
respectively, Reference time is the constant time value used to
evaluate the speed of the input analog signal */*
**if** (Analog input < hi_cutoff) **then**
set 1st variable = Time
**wait until** (Analog input <= lo_cutoff OR
Analog input >= hi_cutoff)
**if** (Analog < lo_cutoff) **then**
set 2nd variable = Time
**else** */* Analog input is greater than lo_cutoff */*
**wait until** (Analog input <= hi_cutoff)
set 1st variable = Time
**wait until** (Analog input <= lo_cutoff)
set 2nd variable = Time
**end if**;
*delta* = abs(1st variable - 2nd variable) */* delta is the time delay */*
**end if**;
**end**;

**Output :** The A2D module outputs four level digital logic equivalent to the input analog voltage.

**Function :** The analog to digital conversion is performed using to two processes DELAY and CONVERT. The process DELAY (Figure 4.11) calculates the time delay for the analog signal to pass between two threshold limits (hi_cutoff & lo_cutoff). The delay time *delta* is used in determining the speed of the input analog voltage by comparing it with the reference time. If the delay is more than the reference time then it is classified as slow varying input where as if the delay is less than or equal to the reference time it is classified as fast varying signal. The process CONVERT is the main process used to convert analog signal to digital signal. Before calling the conversion functions the gate to source voltage (Vgs) value of MOSFET(s) is checked and if it is less than threshold voltage (Vt), digital value of 'Z' is assigned as the output. If Vgs is greater than Vt then different conversion functions are invoked depending on the speed of the analog signal. THRESHOLD and THRESHOLD_1 are the two conversion functions used.

## Digital to Analog Conversion Algorithm

**Inputs :** The inputs to the D2A module are the digital signal, the digital signal sample time, and the simulation termination time (from the input SPICE deck). Similar to A2D module this module has two generic inputs the Reference time and Tech_type. These generic inputs are user definable. The reference time is used in evaluating the speed of the input digital signal.

**Output :** The D2A module outputs analog signal equivalent to the input digital signal.

**Function :** The digital to analog conversion is performed using two processes DELAY and CONVERT. The process DELAY is used in evaluating the time (*delta*) where V = $V_{dd}$ for '1'.

V = $V_{ss}$ for '0'.

V = $(V_{dd} + V_{ss})/2$ for 'X'.

$x$ = damping factor.

$\Delta$t = delay time.

t = sampling time.

In the equations above, the values for $V_{dd}$ and $V_{ss}$ are assumed to be 5.0 volts and 0.1 volts respectively. The delay time ($\Delta$t) in the equations above are the time for which the input digital signal remains at 'X' (see Figure 4.10). The value of the delay time is compared with a reference time. If the delay time is more than the reference time then the input digital signal is classified as slow and the appropriate conversion function is invoked.

## 4.6 Interface Implementation

As described in the previous section, an *interface* is implemented in two main modules and a package with all the conversion functions. The algorithms used in the A2D and D2A modules are shown in the Figures 4.11 and 4.12.

### Analog to Digital Conversion Algorithm

**Inputs :** In the A2D algorithms (Figure 4.11) the inputs to the module are provided internally from the AnaVHDL. They are the analog voltage, the analog signal sample time, the analog simulation termination time (from the input SPICE deck), and the gate to source (Vgs) voltage of the MOSFET(s) driving the inputs. There are two generic inputs which are user definable. They are the Reference Time and the Tech_type. Reference time is used for evaluating the speed of the input signal and tech_type can be either CMOS, ECL or TTL.

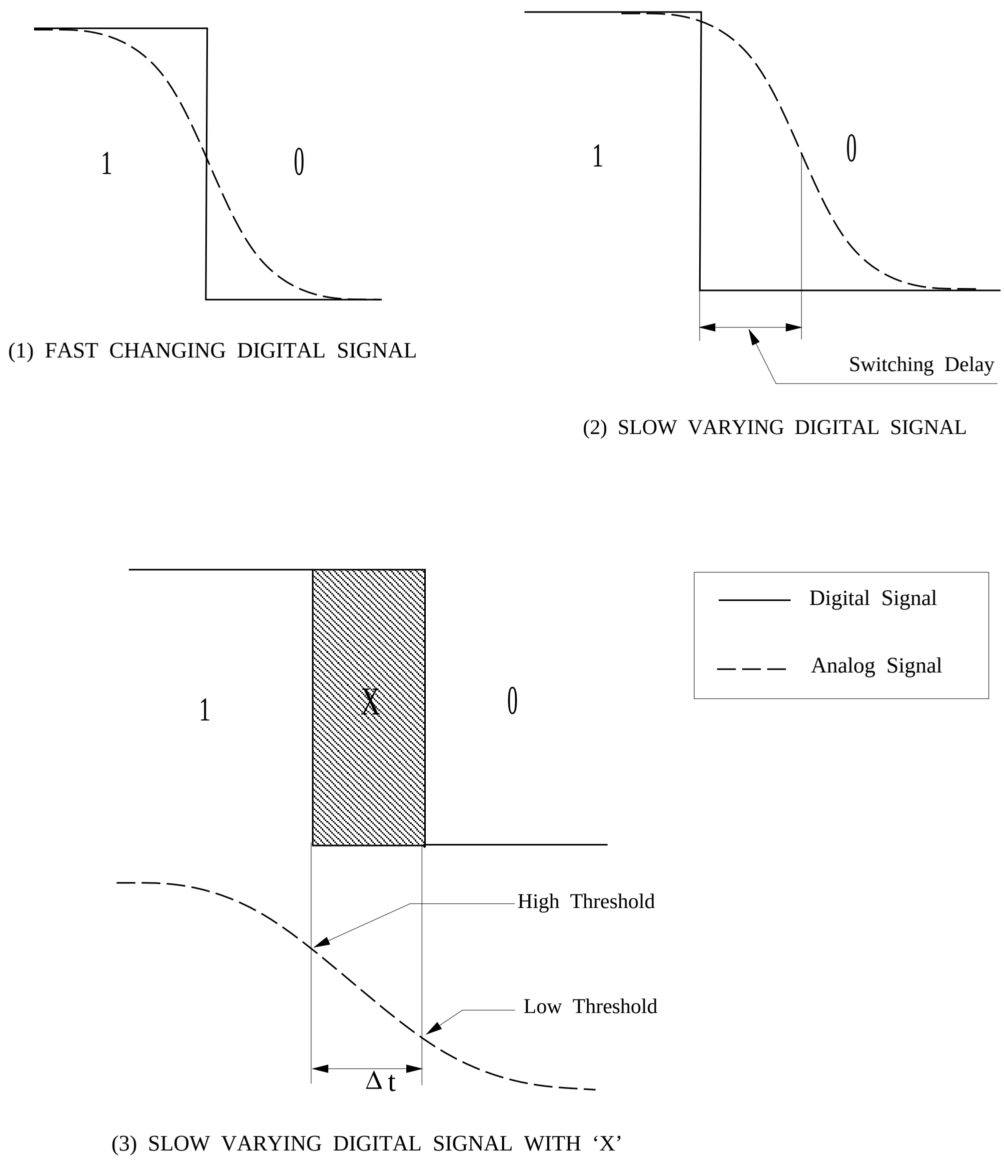



Figure 4.10: Digital to Analog Conversion

concept of VHDL. Threshold_1 is used for conversion in the case of fast varying analog signals wherein there will be only two digital states '0' and '1'. The above mentioned functions are referenced from the package INTERFACE_PACK.

### 4.5.2 Digital to Analog conversion module

The table lookup method is used in converting the digital signals into analog. In this method a specific digital value becomes a constant analog value, for example '1' = 5.0 volts, 'X' = 2.5 volts and '0' = 0 volts. These constant values are substituted in a set of equations to get the analog value. As said in the previous section the constant value is determined by the technology type specified by the user. The Figure 4.10 shows the conversion principle. There are four conversion functions used for this purpose, namely: DIGI2ANA, DIGI2ANA_DELAY, DIGI2ANA_INV and DIGI2ANA_INV_DELAY. While DIGI2ANA is used for converting rising and fast varying digital signal, DIGI2ANA_DELAY is used for converting rising but slow varying digital signal into analog. The function DIGI2ANA_INV is used in converting falling and fast varying digital signal where as DIGI2ANA_INV_DELAY is used for falling and slow varying digital signal. All the above conversion functions are referenced from the package INTERFACE_PACK.

The set of equations used in this conversion process are listed below.

(i) For Falling input voltage :

$$v(t) = \{\mathrm{V} - (\mathrm{V}(e^{-xt}))\}$$

(ii) For Falling input voltage with delay :

$$v(t) = \{\mathrm{V} - (\mathrm{V}(e^{-x(\Delta t)t}))\}$$

(iii) For Rising input voltage :

$$v(t) = \{\mathrm{V}e^{-xt}\}$$

(iv) For Rising input voltage with delay :

$$v(t) = \{\mathrm{V}(e^{-x(\Delta t)t})\}$$

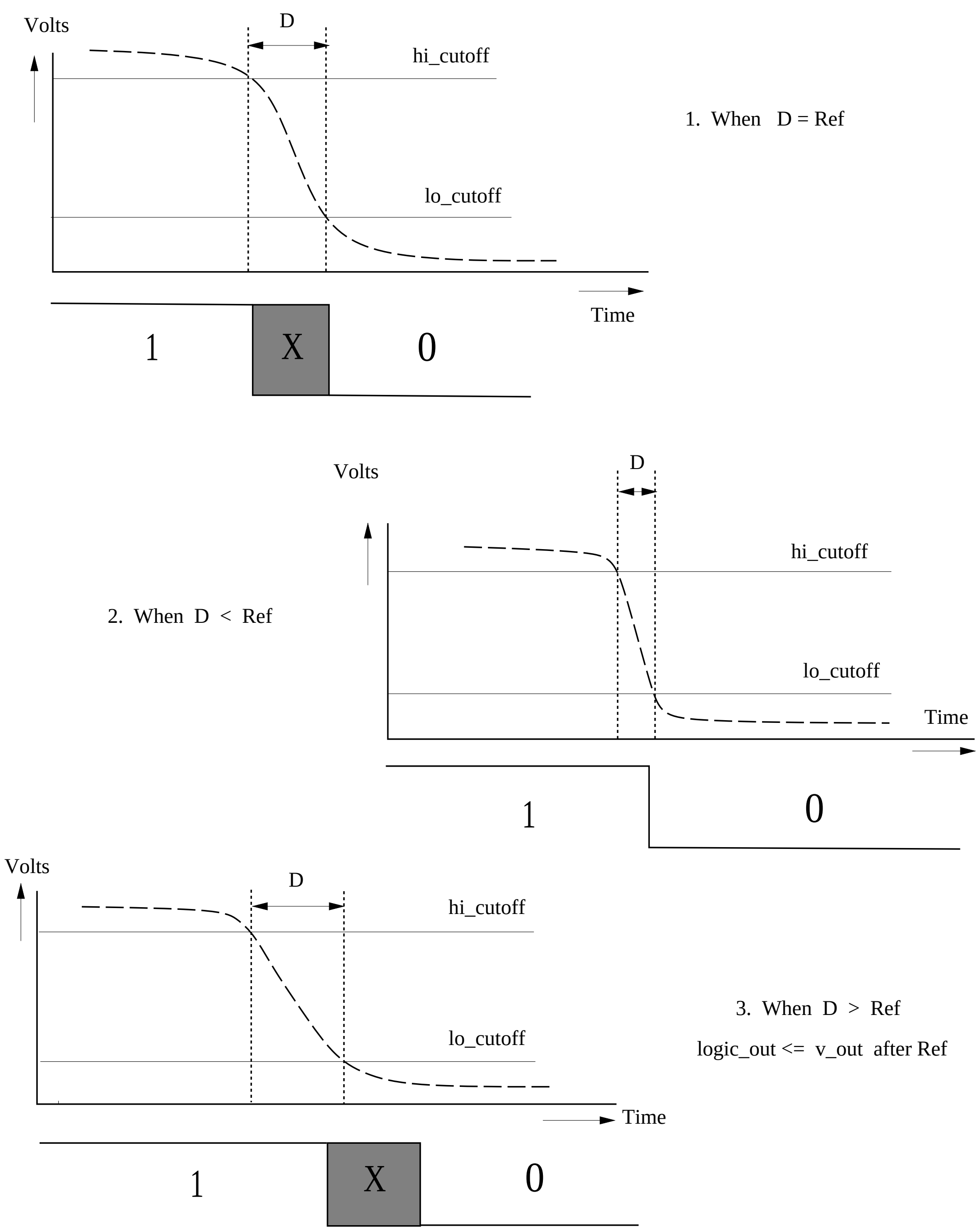



Figure 4.9: Analog to Digital Conversion

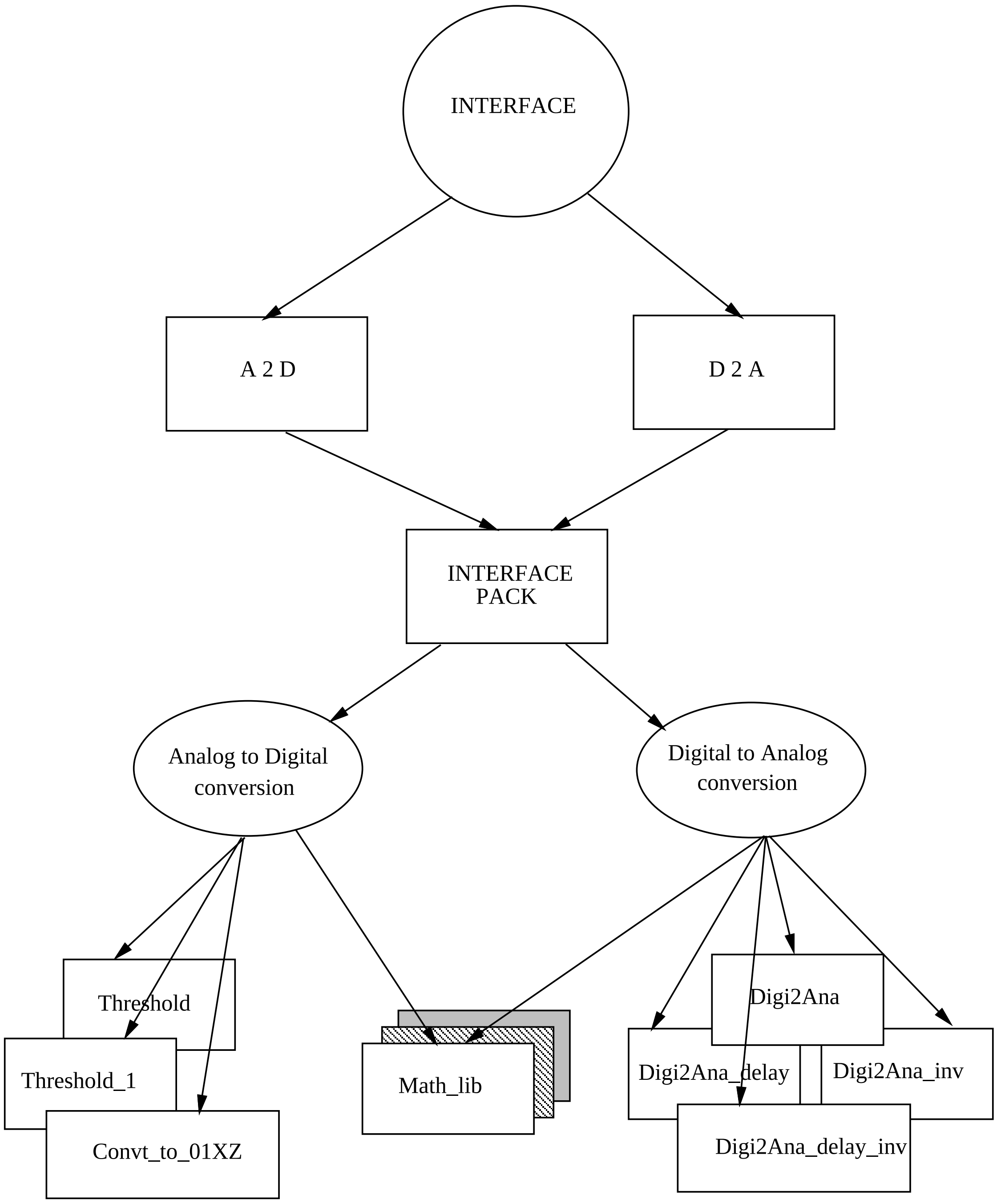



Figure 4.8: Interface Organization

## 4.5 Detailed Design of the Analog/Digital Interfaces

The overall organization of an *Interface* is shown in Figure 4.6. Basically, an *interface* consists of two separate modules **A2D**, **D2A** and a package **INTERFACE_PACK**. The module A2D is used for the analog to digital signal translation and the module D2A is used for digital to analog signal translation. The package INTERFACE_PACK consists a set of functions used by the two modules A2D, D2A and several other type conversion functions which is referred to as MATH_LIB in Figure 4.8.

### 4.5.1 Analog to Digital conversion module

Approach used to convert analog signals into digital signals is to establish thresholds on the analog signals. For example, for a three-state digital signal value set '0', '1' and 'X', any value greater than or equal to 3.5 volts is '1', any value less than or equal to 1.5 volts is a '0' and values in between 1.5 volts and 3.5 volts is 'X'. The maximum and minimum voltage in the above example is assumed to be 5.0 volts and 0 volts respectively. The above principle is illustrated in Figure 4.9. THRESHOLD and THRESHOLD_1 are the two conversion functions used for this purpose. Threshold is the main conversion function which does the conversion supporting all the nine level digital logic of IEEE standard 1164. When the time delay (shown as 'D' in the Figure 4.9) in crossing the threshold limits is greater than a reference time (which is user defined & shown as 'Ref' in the Figure 4.9), the analog input signal is considered to be slow. As mentioned in the previous section the threshold limits are determined by the technology type specified by the user. The digital conversion of slow varying analog signal is handled by assigning the digital output to the output port after a delay time equal to the reference time using the *inertial delay signal assignment*

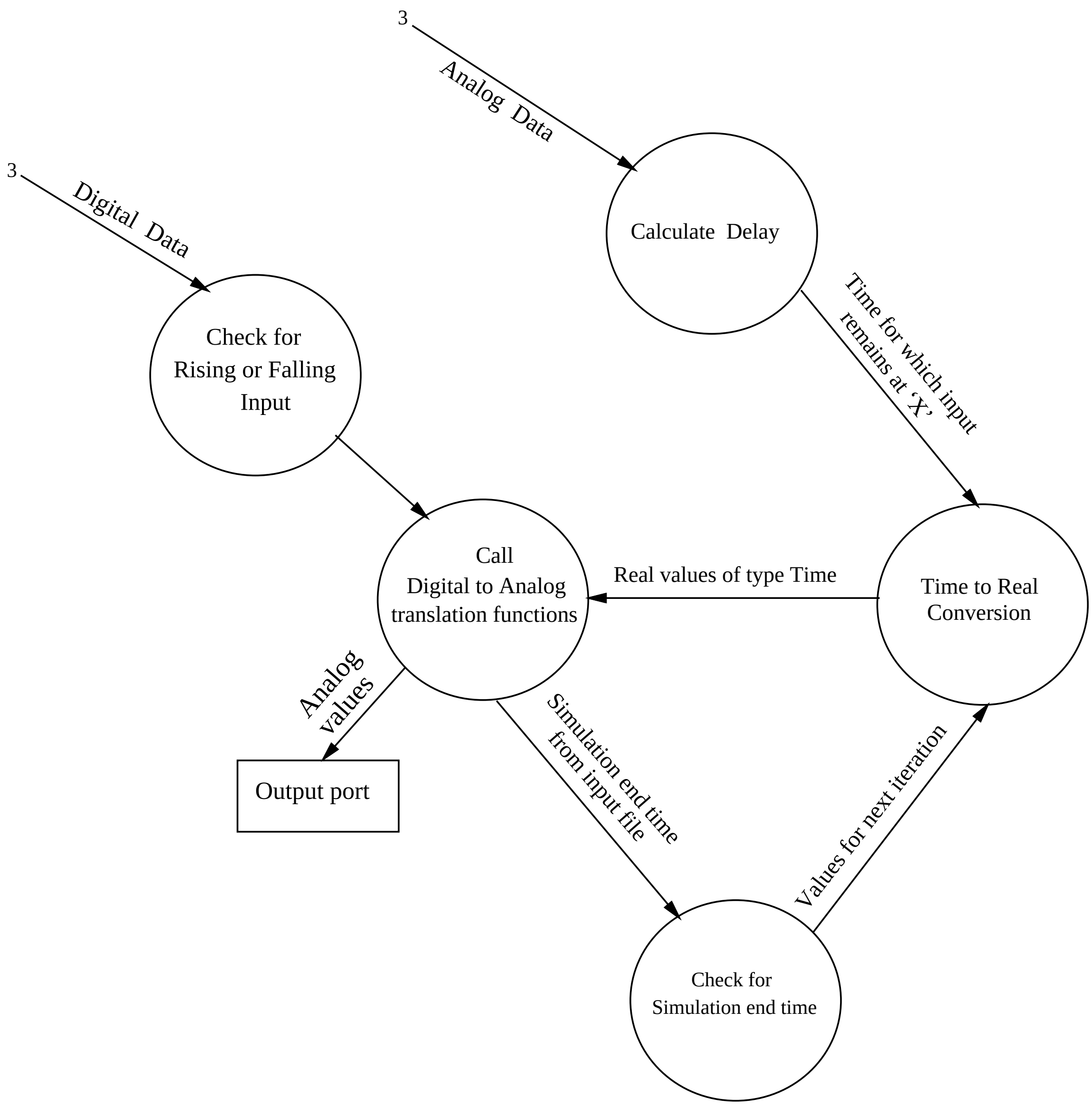



Figure 4.7: Level 2 diagram for "Perform D2A"

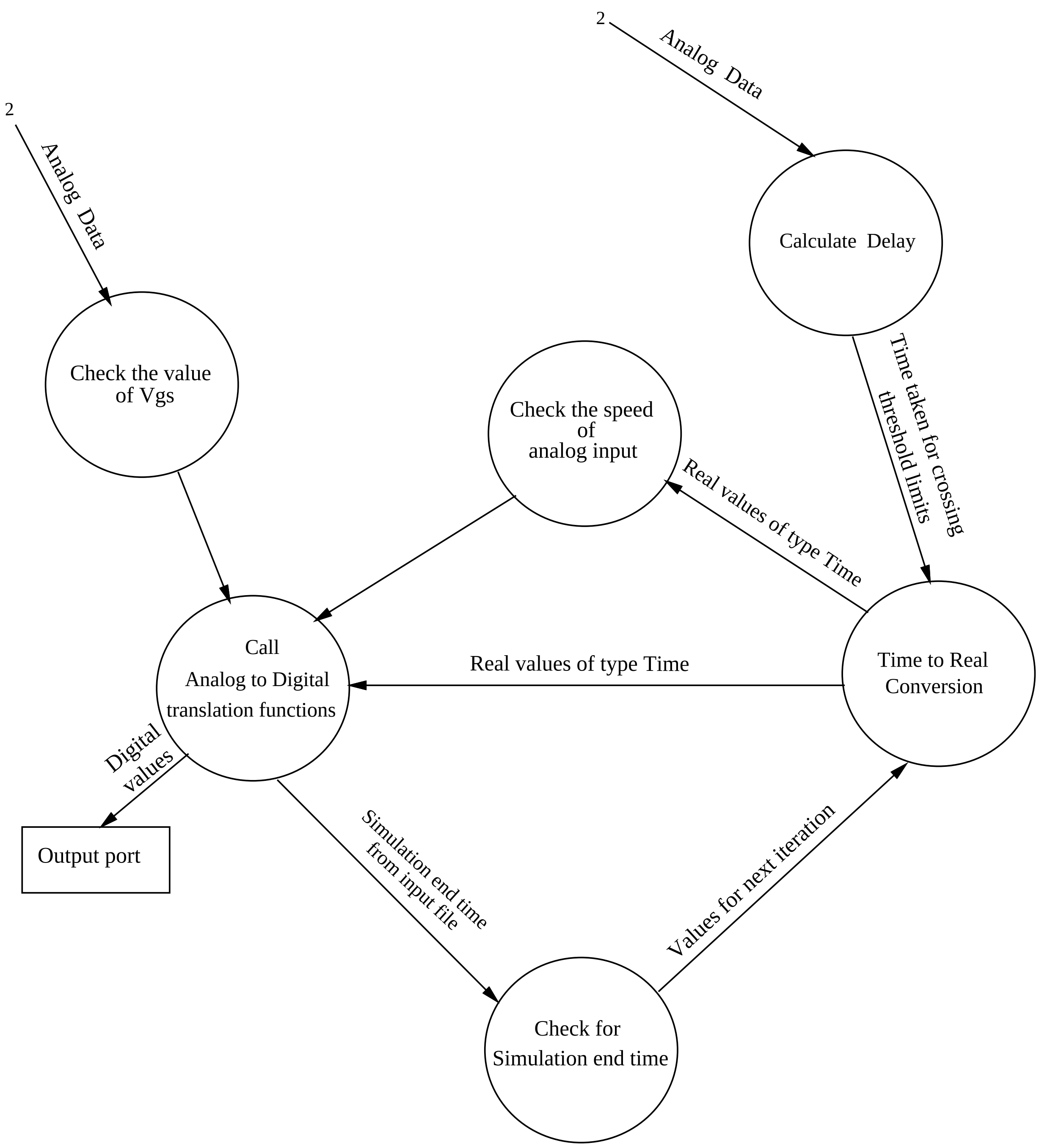



Figure 4.6: Level 2 diagram for "Perform A2D"

## Converting signals from Analog to Digital

The A2D conversion process is shown in the Figure 4.6. After the analog signal is obtained, the time delay for the analog signal to traverse between the high and low threshold limits is estimated. This is necessary to determine the speed of the analog signal. The values of the threshold limits is determined by the technology type specified by the user at the generic port of the module. Depending on the value of the delay time the appropriate signal translation functions are invoked from the package INTERFACE_PACK and the output is assigned to the output port. Before invoking the conversion process the value of the gate-to-source voltage (Vgs) of MOSFET provided by the AnaVHDL is verified and if the Vgs is less than the threshold voltage (Vt) of the MOSFET then a digital value 'Z' is assigned to the output port at that particular instant of time. The simulation is repeated until the simulation end time specified by the user is reached. The resulting digital output is conveyed to the external world through ports. In the process of A2D conversion the time values are converted to real values to perform mathematical calculations.

## Converting signals from Digital to Analog

The process of D2A conversion is shown in Figure 4.7. The digital input acquired for conversion is estimated for the time delay for which it remains at 'X'. The analog conversion of digital signals are done by referencing the appropriate conversion functions from the package INTERFACE_PACK depending on the rising or the falling nature and the time delay of the input digital signal. The conversion is done by substituting a constant value in a set of equations until the simulation end time specified by the user is reached. The above said constant value is determined by the technology type specified by the user at the generic port of the module. The analog output is conveyed to the external world using the ports.

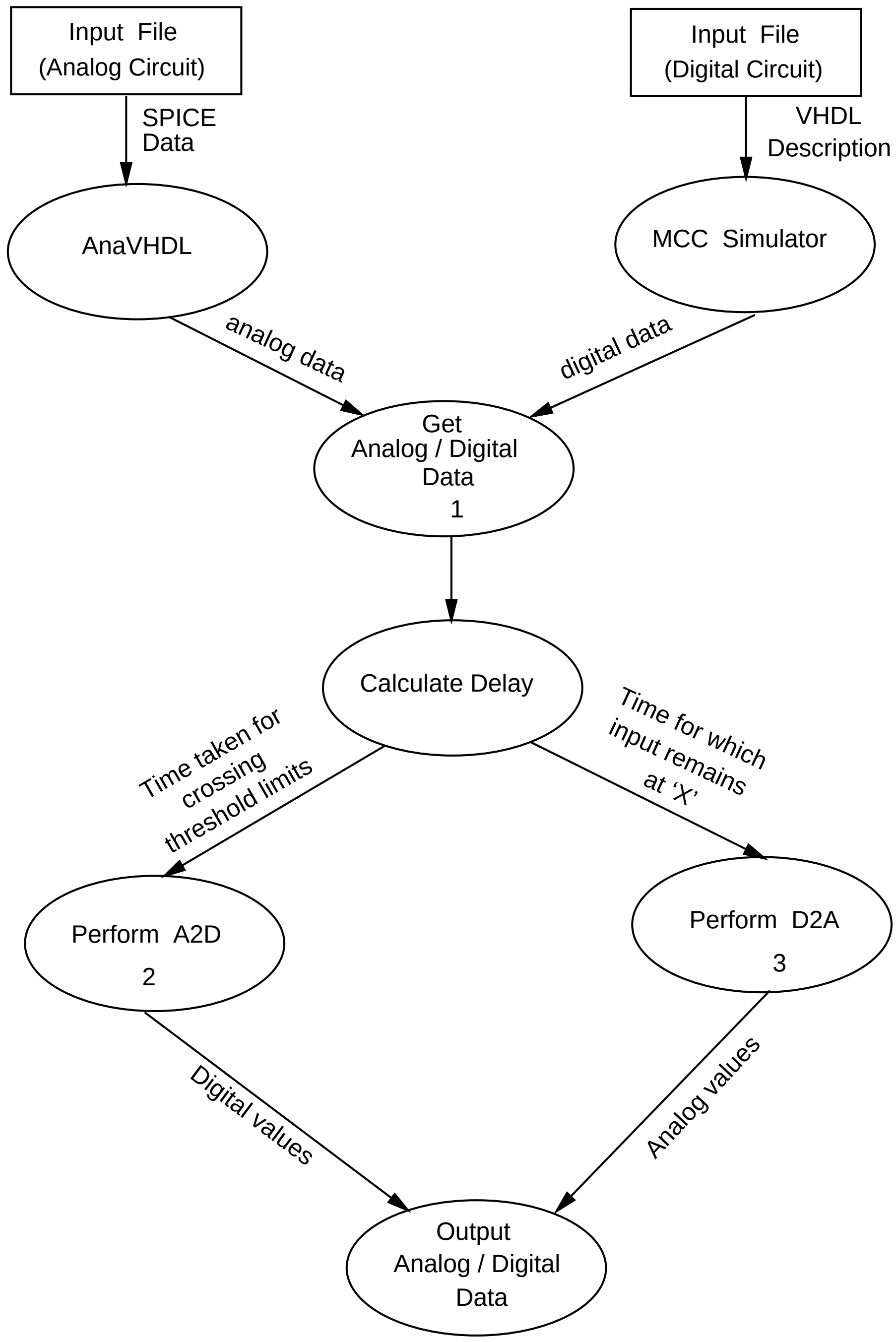



Figure 4.5: Level 1 Data-flow diagram of Interfaces

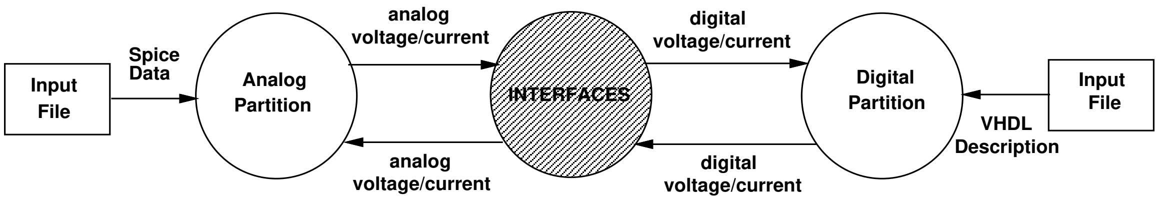



Figure 4.4: Context Data-flow diagram of Interfaces

Standard 1164. The conversion also supports different technology types like TTL, CMOS and ECL.

### 4.4.2 Interface Data flow

We use a data-flow structured analysis approach to describe the design of *interface* modules. The Figures 4.4 to 4.7 shows the data flow in different modules of an *interface*.

The Figure 4.4 shows the overall data flow to and from an *interface*. Basically, an *interface* gets analog data from the Extended AnaVHDL by simulation of analog circuits which is described in the SPICE format and digital data by simulation of digital circuits. As shown in the Figure 4.5 once the data is collected from either analog partition or the digital partition the appropriate conversion modules either analog-to-digital (A2D) conversion module or digital-to-analog (D2A) conversion module is referenced. The top most file which describes the mixed system under simulation determines, which module to be referenced and at what time. The conversion modules then executes the appropriate conversion algorithms by referencing the signal translation functions.

- All inputs and outputs are ideal, thus digital circuit inputs and outputs must have their electrical behavior reflected in the analog circuit behavior.

### 4.3.2 Requirements of an Interface

To achieve the data communication between analog/digital partitions in a mixed-mode system a number of requirements must be satisfied. They are:

1. Analog to digital signal translation functions should be developed and it should be capable to handle both slow and fast varying analog signals.
2. Digital to analog signal translation functions should be implemented and it should be capable to handle both slow and fast varying digital signals.
3. The signal translation functions should support the 9-level digital logic standard specified by IEEE Standard 1164. This will permit better AnaVHDL acceptance by industry.
4. Different technology types like TTL, CMOS & ECL needs to be supported.

## 4.4 General Design of the Analog/Digital Interfaces

### 4.4.1 Functional view

The *Interfaces* are used in association with Extended AnaVHDL to perform mixed-mode simulation in VHDL. The main function of an *interface* is to establish communication between analog and digital partitions in a mixed analog/digital environment by performing analog to digital or digital to analog signal translation. The signal translation conforms to all the nine level digital logic specified by the IEEE

### 4.2.3 Input & Output ports

To communicate data to the external world and between different partitions in a particular system, ports had to be included in the entity description of the AnaVHDL. The Extended AnaVHDL will take both currents and voltages as input and will output the DC or transient voltages and currents. The bi-directionality of analog signal is handled by using signals of the type INOUT internally in the Extended AnaVHDL.

### 4.2.4 Tapping the Gate-to-Source voltage of MOSFET

In determining the logic level 'Z' during the conversion of analog signal to digital signal, we use gate-to-source voltage (Vgs) of the MOSFET as the threshold limit. This requires tapping of Vgs from the function TRAN_ANA in the main program of AnaVHDL, and also the value of Vgs has to be made available at one of the external port so that it can be accessed by the analog to digital conversion module.

## 4.3 The Analog/Digital Interfaces

### 4.3.1 Assumptions

We have made the following assumptions in our approach to design and implementation of an analog/digital *interface*:

- All inputs to the analog circuits must be either an analog primary input (in SPICE file) or from a digital output.
- All inputs to the digital circuits must be either an digital primary input or from an analog output.

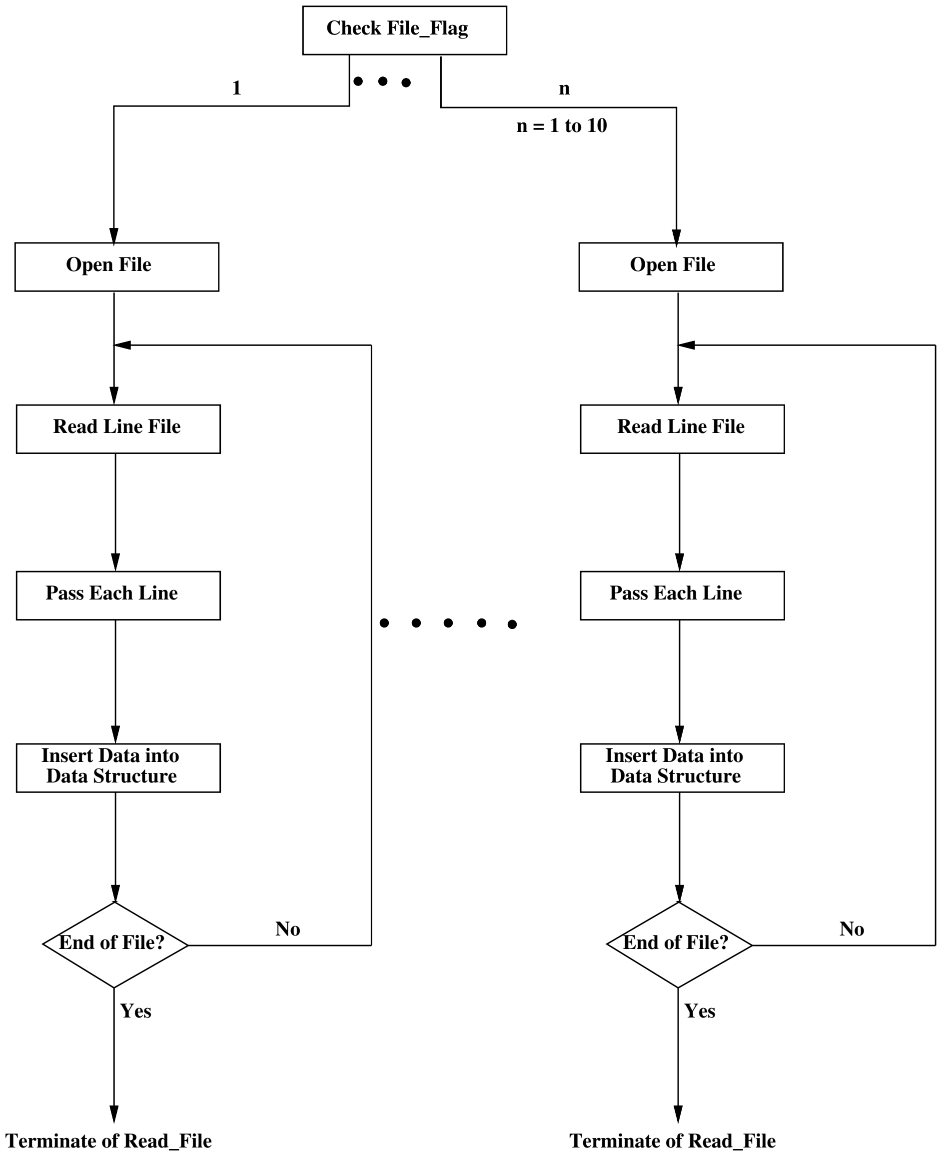



Figure 4.3: Read Multiple Files Program Sequence in Extended Anavhdl

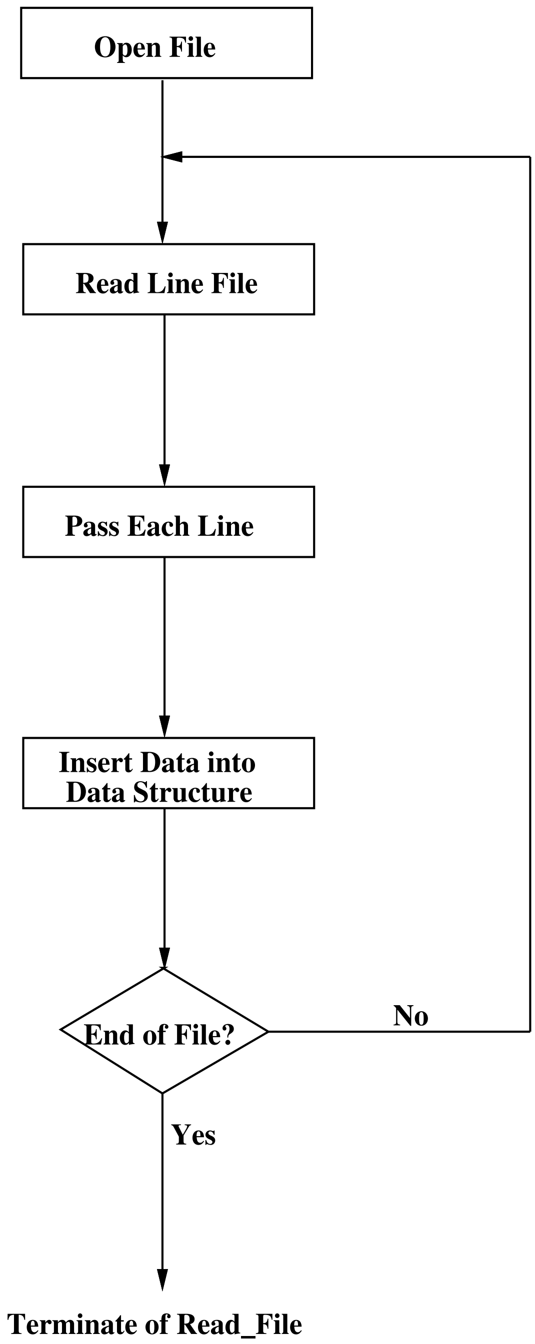



Figure 4.2: Read File Program Sequence in AnaVHDL

time conversion of signals between analog and digital partitions is a must. This imposed the requirement for the development of signal translation functions which we call *Interfaces*. Also the bi-directionality of analog signals has to be handled.

### 4.2.2 Support for multiple analog descriptions

In upgrading the AnaVHDL to support simulation of multiple analog descriptions, changes for the procedure READ_FILE in the package IO_PACK of AnaVHDL was necessary. This extension is achieved using the standard TEXTIO package of VHDL which allows multiple input file declarations. The Extended AnaVHDL has the capability where in an user can simulate upto a maximum of 10 different analog description in the same simulation environment.

The Figure 4.2 shows file read-in sequence in the initial version on AnaVHDL. The Figure 4.3 shows file read-in sequence in the Extended AnaVHDL where in the user has to set a File_Flag with the file number to be simulated. To do this, user has to put each analog description in a separate file. The input files should be named as INPUT_TEXTIO1, INPUT_TEXTIO2 and so on depending on the number of analog descriptions in a particular system. The File_Flag should be set to '1' to simulate file INPUT_TEXTIO1, '2' to simulate INPUT_TEXTIO2 and so on. As shown in the Figure 4.3 the procedure READ_FILE reads in each input file into an internal data structure. In the input file the user describes the circuit and specifies the analysis type (DC or transient) to be performed, the time period for the transient analysis, and the nodes and branches that the voltages are to be output.

the AnaVHDL package. The digital circuit description either behavioral or structural will be described in a *process* using the VHDL library utilities. Primary inputs/outputs will be defined in an entity as signals. The signals between analog and digital domains will reference the analog/digital (A2D) or digital/analog (D2A) interfaces which in turn call the signal conversion functions before passing to the opposite type of circuits. The conversion functions are in a package called INTERFACE_PACK shown as (I-PACK) in the Figure 4.1. The simulation results are passed on as signals to the primary outputs which are defined in the system entity.

## 4.2 AnaVHDL Extensions

As described in the Chapter 3, initial version of AnaVHDL had some limitations to support mixed-mode simulation. Thus occurred the need for an extension to the AnaVHDL so that it supports mixed-mode simulation. All aspects of extensions implemented on AnaVHDL are described in this section.

### 4.2.1 Required Extensions

The initial version of the AnaVHDL had the capability to conduct DC and time domain transient analysis on analog circuit descriptions. To enhance the capability of AnaVHDL to support mixed-mode simulation the following extensions are required:

1. In a mixed system environment, there will be multiple analog and digital partitions. AnaVHDL was capable of simulating only a single analog circuit description. This necessitated the need to upgrade AnaVHDL to support simulation of multiple analog descriptions in a single simulation environment.

2. In a typical mixed system environment there will be regular data communication between the analog and digital partitions and the external world. Thus type and

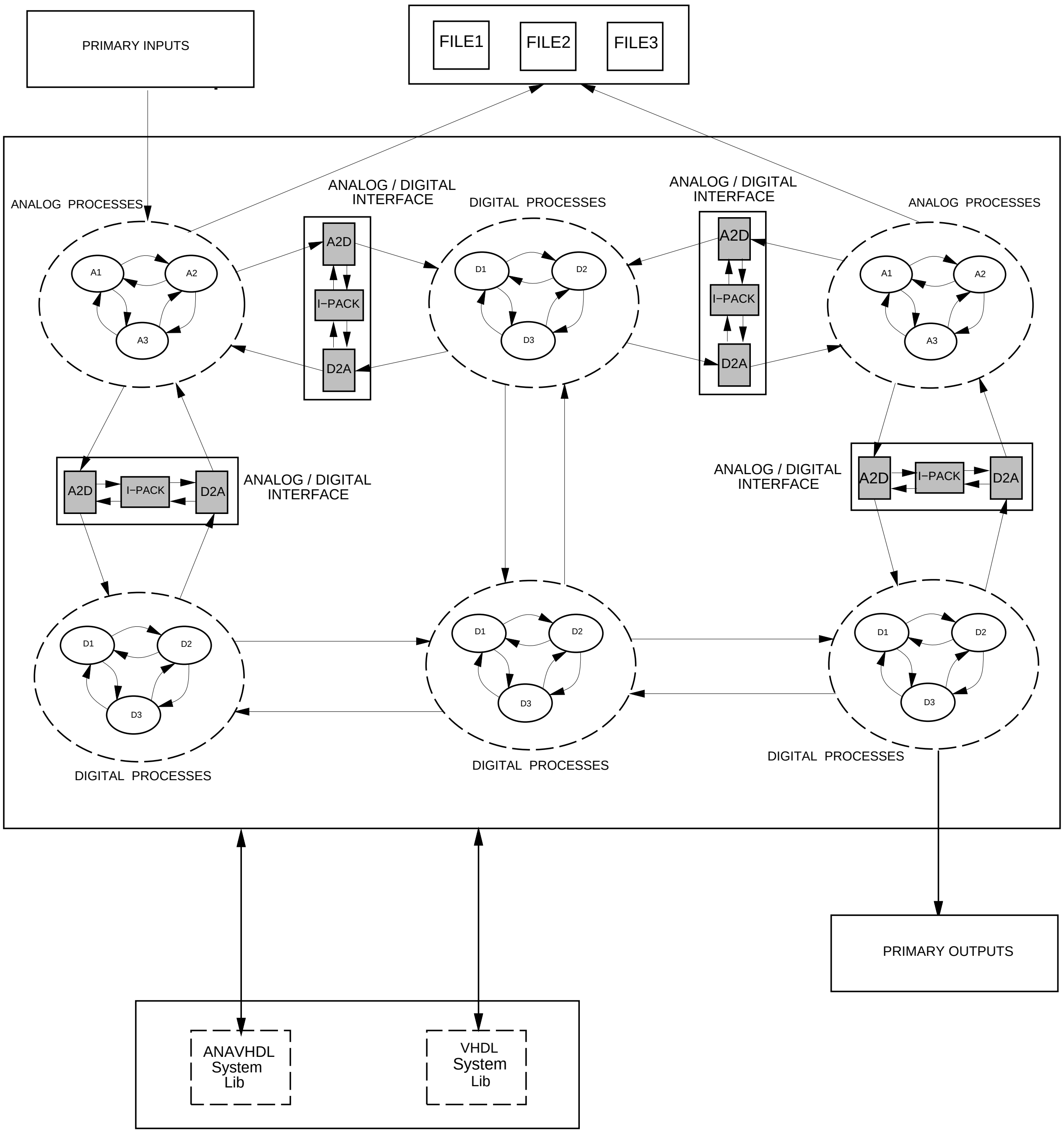



Figure 4.1: A Mixed-mode system

# Chapter 4

# Extended AnaVHDL & Interfaces

This and the succeeding chapters present significant extensions to AnaVHDL as defined by Ms. Zhou. Basically the extensions we have added permit complete mixed-mode simulation.

## 4.1 Overview of a Mixed-mode simulation system in VHDL

A mixed analog/digital system usually contains the primary inputs and outputs, several analog and digital components. Figure 4.1 shows the organization of a mixed-mode system which can be described in VHDL. This system contains four digital partitions and two analog partitions. Mixed-mode VHDL system includes AnaVHDL in a package format, analog/digital *interfaces* for signal type conversions in a package format and digital circuit descriptions.

To describe the whole system in VHDL, a VHDL *process* will be defined for each analog circuit and an input file of each analog circuit description will be referenced by the *process*. The analog circuit behavior simulation will be carried out by referencing

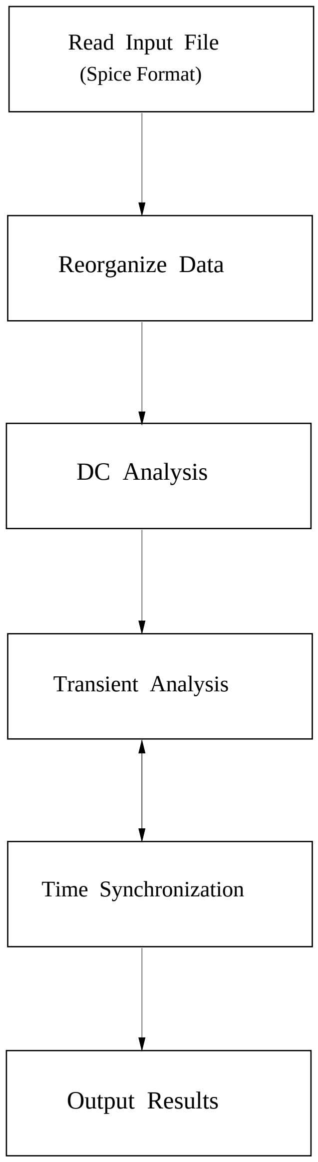



Figure 3.3: Sequential Data Flow in AnaVHDL

The communication among the processes representing different analog circuits is done with **signals**.

internal data structure, reorganization of data to ease circuit analysis methods, performing DC & transient analysis and writing the analysis results into an output file.

## 3.4 Limitations

Even though AnaVHDL is capable of performing analog simulation it has some limitations. The limitations are:

1. AnaVHDL does not handle multiple analog circuit descriptions. It is capable of simulating only a single analog circuit.

2. Since it can only perform analog simulation, to support mixed-mode simulation interface functions capable of signal translation between analog and digital partitions has to be developed.

3. The execution time taken is large compared to that of SPICE, and increases dramatically with the increase in the circuit size. The algorithms needs to be optimized so that execution speed can be increased.

4. Only a subset of SPICE options are implemented and the program can only handle limited number of components.

5. Current version can support only MOSFET models. Other models such as BJT, JFET and IGFET has to be implemented.

## 3.5 Summary

AnaVHDL can be used in mixed-mode VHDL systems as packages to simulate the analog part. User has to define input file for each analog circuit and in the top file used for simulation, a **process** can be created for each analog circuit to be simulated.

output is produced by OUTPUT_FORMATDC and transient analysis output is produced OUTPUT_FORMMATTR.

### 3.2.2 DC & Transient Analysis

As described above, AnaVHDL reads the input file containing the analog circuit description, performs DC and transient analysis and outputs the DC response and transient results to an output file.

DC analysis is accomplished by calling the procedure DC_POINT and it operates over circuit element parameters, constructs a system of nodal equations and solves the nodal voltages of the circuit. A vector of nodal voltages calculated as the initial DC point for each node is passed on to the transient analysis and also written into an output file.

Transient analysis is invoked after DC analysis which is accomplished by calling the procedure TRAN_ANA, which in turn call several other procedures and functions. Transient analysis involves five tasks, first: calculation of numerical integration time step, second: linearizing the model of MOSFET, third: linear model equations of active and passive elements are created, fourth: nodal voltages and voltage source branch currents are calculated using linear system of equations and finally: convergence check is performed on the voltage and current solutions. Once convergence is satisfied, next numerical integration time step is calculated and the process described above repeats until the user specified time. The converged results are interpolated from the calculation time point and output at the user-specified time points.

## 3.3 Data Flow

The Figure 3.3 shows the general sequence of operation of AnaVHDL. The major functions of AnaVHDL includes reading input circuit description, initializing

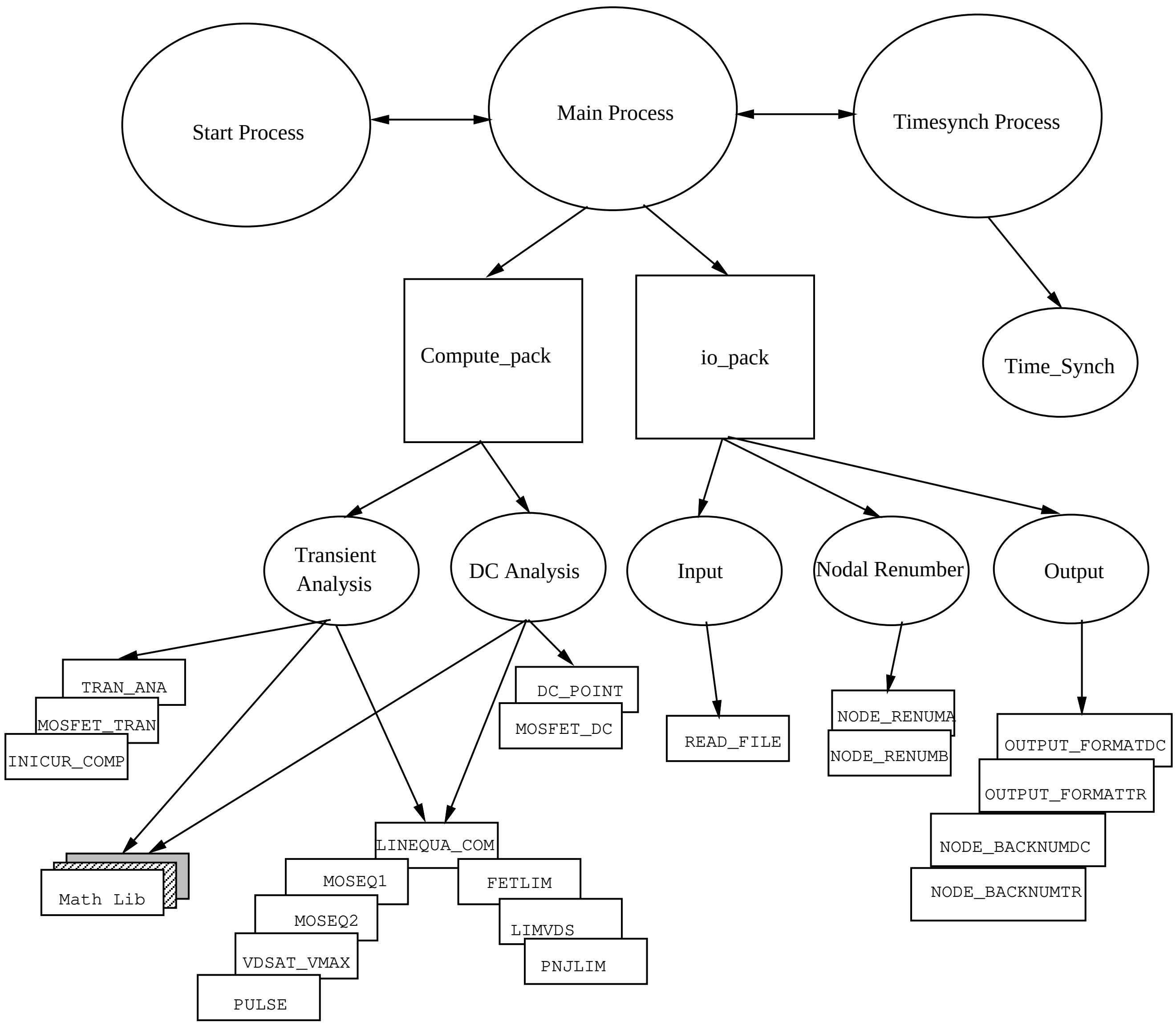



Figure 3.2: General Structure of AnaVHDL [Zhou92]

helps in easy comparison of results.

### 3.1.5 Time Synchronization

The results of transient analysis at each time step of simulation is stored and maintained, until time synchronization has finished and the next analog time step starts. This is done to ensure proper interfacing of analog and digital partitions in the mixed-mode simulation, which is the main objective of the AnaVHDL

## 3.2 Structure

AnaVHDL [Zhou92] consists of three processes in the main program: **start**, **main** and **time-synch**. **Start** initializes the **main** process to begin reading the input file. The process **main** calls functions to perform DC analysis and transient analysis. The process **time-synch** synchronizes the time at each transient analysis time step with the **main** process. All the functions and data structures are organized in two VHDL packages: **io_pack** and **compute_pack**. The Figure 3.2 shows the general structure of AnaVHDL.

### 3.2.1 Input/Output Procedures

**Input :** The input file is read by the procedure READ_FILE which reads it into an internal data structure. The input which is given by the user specifies the type of analysis (DC or transient) to be performed, time period for transient analysis and the nodes and branches at which the currents and voltages are to be output to an output file.

**Output :** OUTPUT_FORMATDC and OUTPUT_FORMATTR are the two procedures used to output the simulation results into an output file. The DC analysis

| No. | Element Name | Max Number of elements |
|---|---|---|
| 1 | Resistor | 100 |
| 2 | Capacitor | 100 |
| 3 | Inductor | 100 |
| 4 | Current Source | 50 |
| 5 | Voltage Source | 50 |
| 6 | Current Pulse | 10 |
| 7 | Voltage Pulse | 10 |
| 8 | MOSFET | 50 |

Table 3.1: Circuit Size Limitation

### 3.1.2 Limitation on the Circuit Size

AnaVHDL [Zhou92] has limitation on the size of the circuits that can be simulated. These are summarized in Table 3.1 [Zhou92]:

### 3.1.3 Input Format

The description of the circuit to be simulated can be input to the AnaVHDL in the SPICE input format. The SPICE input syntax is a free-format style which does not require the data to be entered in fixed column locations. The input file defines the circuit on an element to element basis. The elements are defined by the element card which contains element name, nodes for which element is connected to and the electrical parameters of the element.

### 3.1.4 Output Format

The output of AnaVHDL is in a tabular form which gives the nodal voltages and branch currents similar to the SPICE output. The tabular form of the output

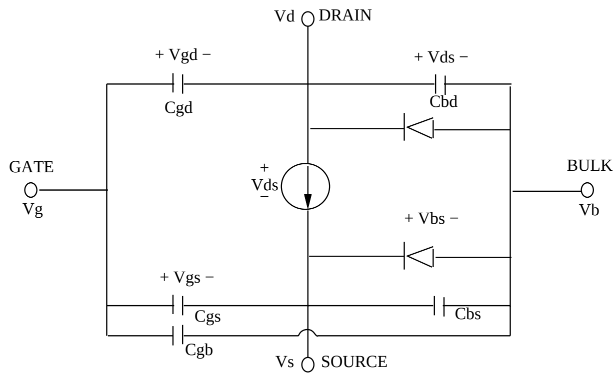



Figure 3.1: A Schematic Diagram of a MOSFET Model

- LEVEL 2 adopts an analytical one dimensional model which incorporates most of the second-order effects of small-size devices such as:

    1. Backgate bias and small-size (short and narrow-channel) effect on threshold voltage considering also the static feedback from drain to gate.
    2. Saturation due to the scattering limited drift velocity of carriers and finite voltage-dependent output conductance.
    3. Surface field dependent mobility.
    4. Weak inversion conductance.
    5. Charge controlled model of regenerative effects.
    6. Variation of all quantities with temperature.

components such as MOSFETs modeled at level 1 and level 2 using SPICE circuit element models.

2. Provides basic circuit simulation functions such as DC and time domain transient analysis by using SPICE numerical methods to solve mathematical equations.

3. Can simulate circuits containing atleast one hundred passive elements and thirty active elements.

4. Reads files in SPICE format description of the circuits to be analyzed.

5. Outputs nodal voltages and currents in a tabular form.

6. Simulation result accuracy is comparable to that of SPICE.

### 3.1.1 Circuit Elements Models

AnaVHDL [Zhou92] uses the same models for circuit elements used by the SPICE. The passive and active circuit element models can be divided into two terminal and four terminal components.

**Two Terminal Components :**

The two terminal components supported by AnaVHDL are resistors, capacitors, inductors, diode, independent current source and independent voltage source. Independent voltage and current sources are of two types: either DC or pulse.

**Four Terminal components :**

Commonly used four terminal components in circuit simulators are MOSFET devices [OdNC88]. Both LEVEL 1 and LEVEL 2 transistors are modeled in AnaVHDL. Figure 3.1 shows the schematic diagram of a MOSFET model.

- LEVEL 1 adopts simple Shichman-Hodges model and is less accurate than LEVEL 2, but much faster in simulation.

# Chapter 3

# AnaVHDL

AnaVHDL(was originally created by Ms. Wenying Zhou at the University of Cincinnati in 1992) as an analog simulation package which demonstrates that VHDL can be used to describe circuits at the component level, even though VHDL was primarily developed for describing digital circuits at gate-level or higher. Hence AnaVHDL is used in the mixed-mode simulation environment for analog (circuit level) simulation.

In this chapter we discuss in detail the various aspects of AnaVHDL. Section 3.1 discusses the properties of AnaVHDL, Section 3.2 shows the structure and implementation details, data flow is discussed in Section 3.3 and Section 3.4 summarizes the overall features.

## 3.1 Properties

The main properties of AnaVHDL as defined by Ms. Zhou are listed below:

1. Analyzes circuits containing passive components such as linear resistors, capacitors, inductors, independent voltage and current sources and pulses, and active

action and the physical interconnection of Bipolar/CMOS chip used in hard disk drive products. In the general simulation model the transfer function modeling method is used. A transfer function is a means of describing the outputs of a component or network as a function of its input. Usually, any component can be modeled if the inputs and outputs are uniquely distinguishable and the external load impedances have only second-order effects on the component behavior. These transfer function simulation models are limited to realizable analog functional blocks. Passive components such as resistors, capacitors and inductors cannot be modeled with this technique because they have no unique inputs and outputs.

Since the outputs are functions of inputs, analog behavior must account for the loading on its outputs. VHDL can describe bi-directional (INOUT) ports, hence the load may broadcast its value to the behavior at time zero and update it during simulation, thus providing a method of coupling the functional block to its load. Transfer function modeling methods reduce the simulation efficiency and increase the model complexity and thus it is most applicable to the circuits with well defined transfer functions and low sensitivity to loading variation.

simulation. Since only switch-level timing algorithm and logic-level simulation are included in the simulator, it is obvious that it cannot be used as a general purpose mixed-mode simulator. Besides, it lacks the behavior modeling capability of circuits.

### 2.2.3 The L-Simulator Approach

The L-simulator [OdNC88] is a mixed-mode simulator which combines a behavioral timing simulator, a switch-level simulator and a circuit simulator based on ADEPT [OdNa86] timing simulation algorithm. These simulation algorithms are combined in a single, consistent, interactive MOS simulator. Again, the L-Simulator incorporates only the MOS model at the circuit level which cannot be used as a general purpose mixed-mode simulator.

### 2.2.4 Augmented Analog Simulator

There has apparently been some attempts made to augment SPICE by adding logic elements to build the mixed-mode system (possibly in PSPICE [Tuin88]). The information available to describe these approaches is very limited.

## 2.3 VHDL based Mixed-mode Simulation

VHDL, rich in the constructs found in both HDL and a programming language, has the flexibility and structure necessary for an effective analog-digital behavioral simulation language reflecting analog-digital interconnection [HaSt91].

Even though VHDL was developed for modeling digital circuits, there has been attempts to develop analog circuits modeling techniques. One of the famous techniques is *general simulation modeling technique* by Mark Brown and Balsha Stanisic [StBr89]. This technique was used to verify both the analog-digital functional interof semiconductor IC technology, there is a rapidly growing need for simulating ICs where both analog and digital functions reside on the same chip [Webe91]. These circuits often include tightly coupled feedback between the analog and digital partitions. Simulating the analog-digital interaction and assuring that it reflects the physical analog-digital interconnection can greatly improve the design. To get a good estimate of the entire chip behavior, simulation has to be done within the same environment. As a result, the development of mixed-mode simulation CAD has become extremely important. Over the years, various approaches have been proposed for building mixed-mode simulators so that analog-digital circuits can be simulated in one environment and the interconnection between analog and digital circuits can be verified. The major approaches are summarized in the following subsections.

### 2.2.1 Traditional Approach

A traditional approach to build a mixed mode simulator is to "glue" an existing analog simulator with a digital simulator [Morr89]. The digital simulator can be either logic-level or switch-level or both depending on the needs of verification. This approach can handle pure analog or pure digital circuits even though one portion of the circuit requires higher accuracy than the rest of the circuit, but it does not have an integrated time for both analog and digital simulation. The bidirectional characteristics of the analog signals is also not handled.

### 2.2.2 iSPLICE3 Approach

The iSPLICE3 [ADPS89] is a simulator which combines an analog algorithm with digital algorithm in a single program. The electrical switch-level timing and logic simulation modes are combined using event-driven selective-trace techniques. It features a hierarchical schematic capture package called iSPI for design entry and simulation control. Different representations, models and signals are combined in one

In switch-level simulation [Brya80] the circuits are simulated at the transistor level rather than at the gate level. Here the transistors are modeled as switches, if the transistor is ON it is considered as closed switch and if it is OFF then it is an open switch. The network is considered to be made of a set of nodes connected by these idealized transistor models. Charge-sharing and bi-directionality of MOS transistor can be modeled but the timing information obtained is limited.

The Register transfer level simulation [Breu75] is used on the circuits with higher levels of abstraction like multiplexors, decoders, encoders, adders and arithmetic units. Usually it is used extensively in the data path design and for evaluating designs for different architectures. The circuit operation is described by a set of equations which involves register transfers and arithmetic operations that are similar to data-flow descriptions. This level of simulation lacks the capability to provide information about races, hazards, illegal states or critical timing constraints.

In behavioral level simulators [InSL84] circuits can be described as functional blocks which can later be used in system level simulation. Both combinational and sequential circuits can be described as functional blocks. The circuits can be defined in two types - *structural* and *behavioral* wherein structural blocks describes the interconnections of the functional blocks and functional blocks describes the operations to be performed on the inputs to get the required output. These descriptions are written in a high level language and passed through a simulator which emulates the particular description.

## 2.2 Mixed-mode Simulation

Historically the analog and digital partitions of a hardware design have been modeled in different languages and simulated using separate simulators. Each simulation has been adequate within its partition, but these partitions could not be easily or automatically combined. In today's semiconductor market, with the development simulation. Different approaches used to develop mixed-mode simulators in the past are summarized in the Section 2.2 and, finally, Section 2.3 summarizes VHDL-based mixed-mode simulation approaches done in the past.

## 2.1 Levels of simulation

Circuit simulation [SaNe90] can be done at many different levels where the accuracy and complexity of information obtained as a result of simulation varies from one level to another. These levels are [SaNe90]:

- Electrical simulation
- Gate-Level simulation
- Switch-Level simulation
- Register-Transfer level simulation
- Behavioral simulation

The Electrical or circuit level simulation [SaNe90] provides the greatest amount of information by solving a system of nonlinear first-order differential equations. The dynamic characteristics are modeled by these equations for a set of input voltages and given initial conditions.

The Gate-level simulation (also called logic simulation) [SaNe90] is performed when the IC design complexity reaches such a point where electrical simulation is no longer feasible. Here transistors are grouped into *gates*. The simulation at the gate level increases the execution speed by reducing number of models to be processed and by simplifying the arithmetic operations. Gate level simulation can provide first order timing details and also can detect hazards, glitches and race conditions.

# Chapter 2

# Background & Related Work

Computer simulation [SaNe90] is used in many areas of engineering & science, where it is expensive or time consuming or both to observe the behavior of the actual system. In the area of electrical engineering, circuit simulation is commonly used to observe the behavior of integrated circuits (ICs) and to obtain the timing information before performing expensive fabrication processes. The Computer Aided Design (CAD) tools greatly assist in this process, by providing precise electrical waveform information in response to input stimuli and information on the associated parasitics of the circuits with complex devices.

The increasing number of devices on a chip due to higher levels of integration has led to the development of a number of dynamic analysis tools such as behavioral simulators, register-transfer-level (RTL) simulators, gate-level logic simulators and switch-level simulators. Circuit simulation programs like SPICE provides circuit analysis like time-domain transient analysis, ac analysis, dc analysis, noise analysis and distortion analysis. The SPICE can consume excessive computer time for complex analysis on circuits with lot of components.

In this chapter we discuss the background and some of the related work done in the area of mixed-mode simulation. Section 2.1 outlines different levels of circuit

## 1.4 Summary of Results

In this research we defined and implemented the necessary extensions to AnaVHDL including *interfaces* (signal translation modules) to make AnaVHDL a fully integrated mixed-mode simulator. The AnaVHDL and *Interfaces* are completely implemented in VHDL since VHDL is rich in constructs that are found both in hardware description language and a general purpose programming language. To validate our results, we defined and simulated several representative benchmark circuits consisting of both analog and digital circuits. The maximum relative error of AnaVHDL is 0.1% of the actual result expected for all cases considered. Simulation can be done on multiple analog-digital descriptions in a single system but the simulation time taken is large due to the inherent slowness of AnaVHDL and increases with the complexity of the circuit under simulation. The maximum relative error of *Interfaces* is 3.3% of the actual result expected for all cases considered. Complete analysis and results of simulations performed are presented in Chapter 5.

## 1.5 Thesis Organization

Chapter 2 presents the background and related work done in mixed analog-digital simulation. The properties, structure and design of AnaVHDL are presented in Chapter 3. The Extended AnaVHDL, the design of *Interfaces* and algorithms used in the implementation of *interfaces* are explained in Chapter 4. Evaluation of the Extended AnaVHDL & *Interfaces* with some example simulations and software limitations are shown in Chapter 5. Finally, Chapter 6 summarizes the research work described in this thesis and future work that can be done.

for analog to digital type conversion as shown below:

1. Two Threshold limits : This is a simple approach where in two threshold limits (high & low) are established on the analog signal. Depending on the value of the analog signal with respect to the threshold limits equivalent digital value is determined.

2. Multiple Threshold limits : In this approach multiple threshold limits can be established on the analog signal instead of only two as indicated above.

3. Threshold limits with Rise & Fall times : In addition to establishing either two or multiple threshold limits on the analog signal, rise and fall times of the analog signal can also be taken into consideration for determining the equivalent digital value.

The approach used in our work to convert analog signal to digital signal is to establish two threshold limits. This involves establishing two threshold limits (hi_cutoff and lo_cutoff) and checking the analog signal at each simulation step if it is above hi_cutoff, below lo_cutoff or between the two limits. Depending on the above result a '1', '0' or 'X' is assigned to the digital output. In this approach both slow and fast varying analog signals are taken into consideration.

The reverse problem of converting digital output to analog input can be done in a straight forward manner using the table lookup method, where a specific digital value becomes a constant analog value and then using these constant values in a set of equations which describes the rising and falling analog signals. The algorithms used are explained in detail in Chapter 4.

Two signal translation modules (*Interfaces*) has been developed, one for converting continuous voltage or current signals to digital (A2D) and the other for translating digital signals to analog voltages or currents (D2A). Both modules implement IEEE Standard logic 1164.

- to upgrade AnaVHDL to simulate multiple analog circuits in a system
- reorganize the entity by adding input/output ports such that it allows data communication between analog & digital partitions
- enhance the capability to support the bi-directionality of analog signals.

The other major requirement in enhancing AnaVHDL to support mixed-mode simulation is to develop proper analog/digital signal translation functions to interface analog and digital domains. These interfaces integrate the communicating analog and digital domains.

## 1.3 Signal Translation

AnaVHDL can support mixed-mode simulation with the extensions specified above and by overcoming the difficulty in interfacing analog and digital circuits. As said in the previous section this requires the conversion of signal types between the analog domain and digital domain. The basic problems in analog/digital interfacing are:

- The continuous nature of the analog signals which must be modeled both in magnitude and time. The output of the circuits under simulation may suffer either time lag or time lead with respect to input.
- The type conversion between analog voltages/currents and discrete logic levels.
- The bi-directionality of analog signals verses uni-directionality of digital signals.

### Approach

VHDL permits one to define type conversion functions to appropriately provide analog/digital signal type conversion. There are several approaches that can be used

## 1.1 Mixed-mode Simulation

During the last few years research has been conducted on different simulation approaches for mixed analog-digital descriptions of hardware. The L-simulator, and iSPLICE3 are two examples of mixed-mode simulators. The L-Simulator approach, implemented by Mentor Graphics Inc., integrates a behavioral timing simulator, a switch-level simulator and a circuit-level simulator in a single language and simulation environment [OdNa86]. iSPLICE3 is a simulator which combines a single analog algorithm with a digital algorithm in a single program [ADPS89]. Each of these mixed-mode simulators performs some level of mixed analog-digital simulation, but they are inadequate on a number of points. For example the L-simulator approach is specific to MOS transistor level models only and thus is not considered as a general purpose simulator, iSPLICE3 adopts only the switch-level and logic-level view and thus fails to be a general purpose mixed-mode simulator as well.

## 1.2 VHDL Based Mixed-Mode Simulation

Even though VHDL was primarily developed for modeling digital circuits, it has been used to model analog behavior for event driven simulation[StBr89]. Mark Brown and Balsha Stanisic have used transfer function modeling method to behaviorally model analog circuits in VHDL [StBr89]. These transfer function simulation models are limited to realizable analog functional blocks.

In our effort to develop a mixed-mode simulator, a SPICE like analog simulation capability called "AnaVHDL" [Zhou92] was developed in VHDL. AnaVHDL performs DC and transient analysis on analog circuits using SPICE algorithms [Zhou92]. However extensions to AnaVHDL are necessary to make it support mixed-mode simulation. Some desirable extensions are:

# Chapter 1

# Introduction

In today's semiconductor market sales in the growth of mixed signal ICs, where both analog and digital functions reside on the same chip is significantly increasing [Webe91]. As a result there is a need for development of efficient mixed-signal CAD and testing. We present a mixed-mode (which can simulate both analog and digital domains in a single description) simulation facility to easily analyze multiple-level models (multiple level models for analog domain can be of any level model from electrical-level to behavioral-level abstraction and for digital domain from gate-level to device level abstraction) in the same simulation environment, thus providing flexibility in design and simulation. A mixed-mode simulator allows the designer to represent the same part of the circuit at different levels of abstraction. The simulator then combines the different representations, models and signal types in one simulation and produces the desired results [ADPS89].

In this chapter we discuss mixed-mode simulation in general followed by VHDL-based mixed-mode simulation. We also discuss in brief, the approach used by us in developing a mixed-mode simulator, summary of the results obtained and the organization of the chapters in the rest of the thesis.

# List of Tables

# List of Figures

# Contents

## ACKNOWLEDGMENTS

I like to take this opportunity to express my gratitude to Dr. Harold W. Carter for his guidance and excellent ideas, which helped me in conducting research and completing my thesis work at the University of Cincinnati.

I am grateful to Dr. Joseph Nevin and Dr. Dinesh K. Bhatia for serving in my thesis committee. Their suggestions and comments helped in improving the contents of my thesis.

I would like to thank Ms. Wenying Zhou for her help and support at the every stage of my thesis. Also special thanks to my labmates for all the help they have extended to me during the course of my thesis work.

My deepest gratitude to my roommates Krishna, Prasad and Guruprasad for their continuous support and understanding during both good and bad times. I am also grateful to Shankar for taking time to proof read my thesis.

Last but not least, I am ever grateful to my parents, sisters and relatives for their endless affection, encouragement and love. Their continuous support has made my effort easy, and stay in this country far from home more comfortable.

## Abstract

In this thesis, we describe a Mixed-mode simulation capability in VHDL which is capable of simulating both analog and digital circuits in the same simulation environment. Extended AnaVHDL and *Interface* packages are implemented. Extended AnaVHDL is a SPICE like analog simulation capability embedded in VHDL which can perform DC & transient analysis on multiple analog circuits including MOS transistors. *Interfaces* are the signal translation mechanism for integrating analog and digital circuits in a mixed-mode system. This capability supports IEEE standard logic 1164.

# MIXED-MODE CIRCUIT LEVEL SIMULATION IN VHDL

A thesis submitted to the

Division of Graduate Studies and Research
of the University of Cincinnati

in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in the Department of Electrical and Computer Engineering
of the College of Engineering

March 1993

by

Subramanyam Bangalore

B. E., Bangalore University, Bangalore, India, 1989

Thesis Advisor and Committee Chair: Dr. Harold W. Carter