

# A Synchronization Algorithm for VHDL-AMS Simulation with ADA Feedback Effect

**Presenter:**

**N. Bani Asadi** [nargesb@stanford.edu](mailto:nargesb@stanford.edu)

**Authors:**

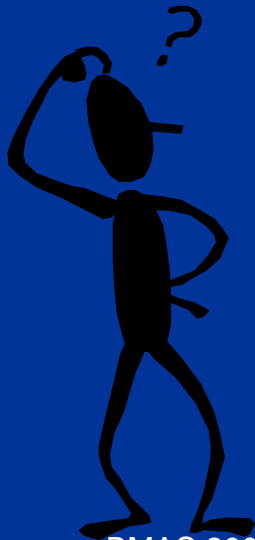
**H. R. Ghasemi** [hrghasemi@cad.ece.ut.ac.ir](mailto:hrghasemi@cad.ece.ut.ac.ir)

**Dr. Z. Navabi** [navabi@ece.neu.edu](mailto:navabi@ece.neu.edu)

**ECE Department  
University of Tehran  
Iran**

# Outline

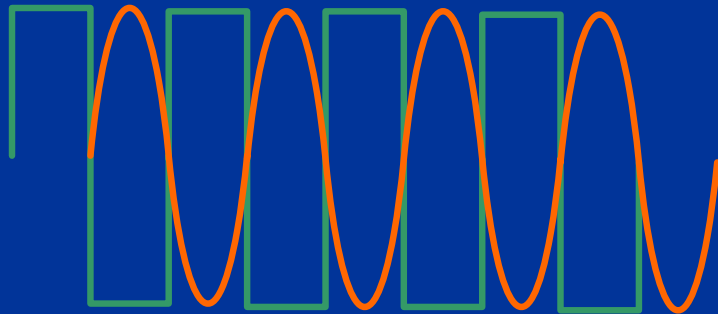
- Introduction
- Simulation Process
- Compilation
- Elaboration
- SIRE/M Intermediate Format and Simulation Kernel
- Synchronization



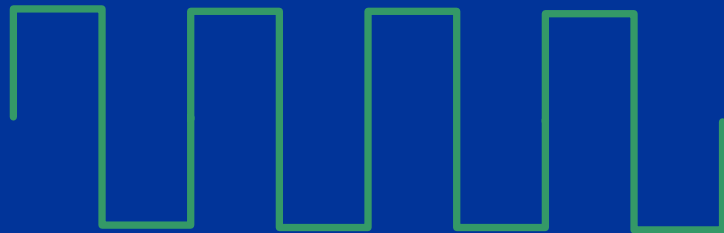
# Introduction



Analog



Mixed Signal



Digital Signal

Language: Spice, ...

Intermediate Format: Matrix

Language: ?

Intermediate Format : ?

Synchronization Algorithm: ?

Language : VHDL, Verilog,..

Intermediate Format : Machine code

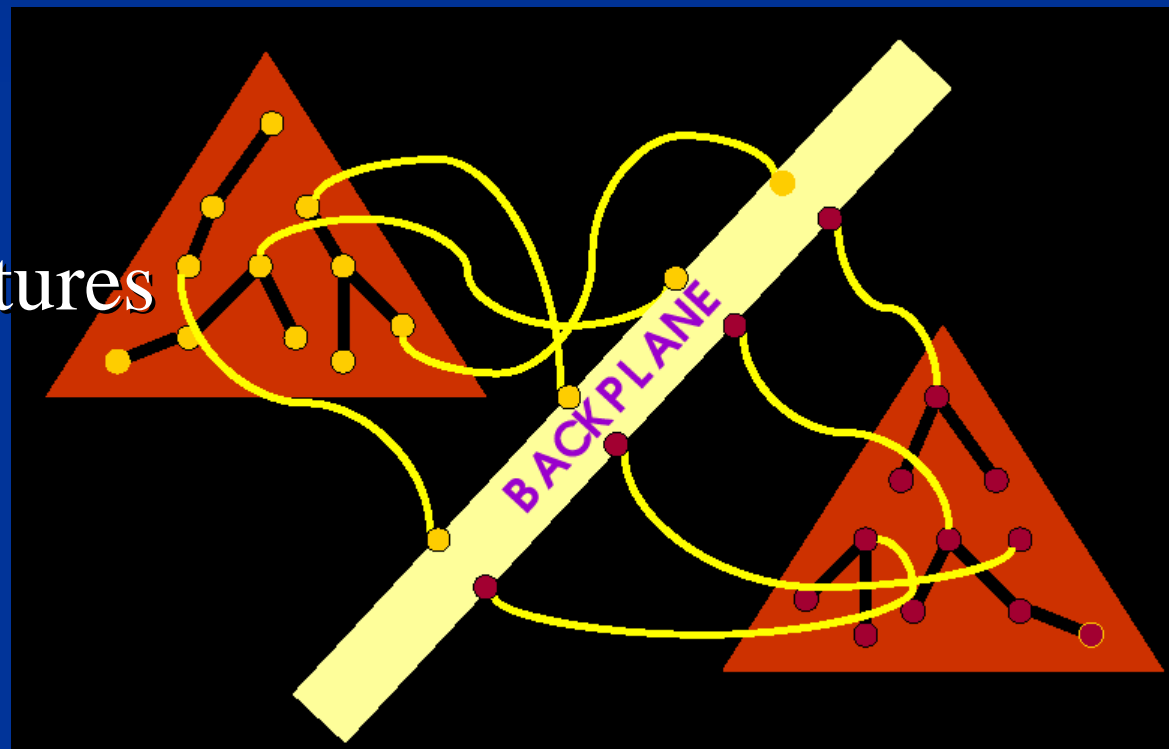
Algorithm : Event Driven

# Mixed Signal Languages

- Primary Mixed Signal Languages
  - Analog Behavioral Languages + VHDL or Verilog
- Standard Mixed Signal Languages
  - VHDL-AMS (1999)
  - Verilog-AMS

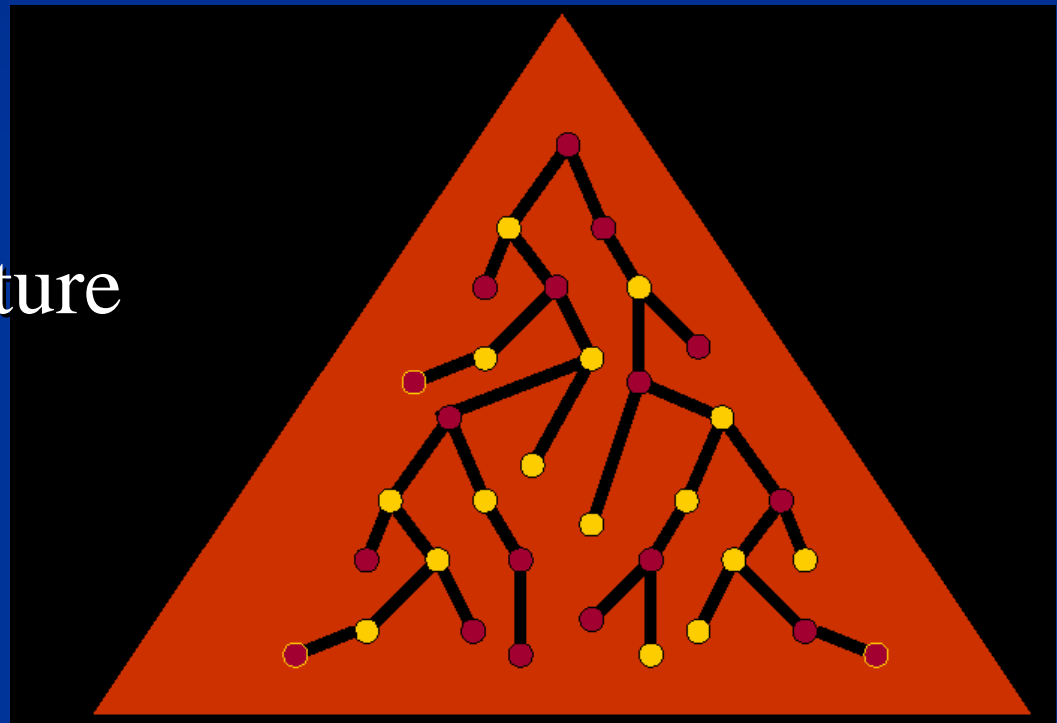
# Traditional Mixed Signal Simulation Engine

- Two Languages
- Two Data Structures



# Unified Mixed Signal Simulation Engine

- Unique Language
- Unique Data Structure



# Analog Simulation Algorithm

- Oblivious Algorithm:

*Generating data structure from Source code ;*

*Calculate initial values ;*

*Repeat*

*Generate a time-slice for DAE Solver ;*

*Solve DAEs with new time-slice ;*

*Until (current\_simulation\_time  $\leq$  End Time) ;*

# Digital Simulation Algorithm

- Event-Driven Algorithm:

*Generating data structure from Source code ;*

*Run all Statements Once at zero time;*

*Schedule generated events in the event queue ;*

*Repeat*

*Advance current\_simulation\_time into the first event-time in the event queue ;*

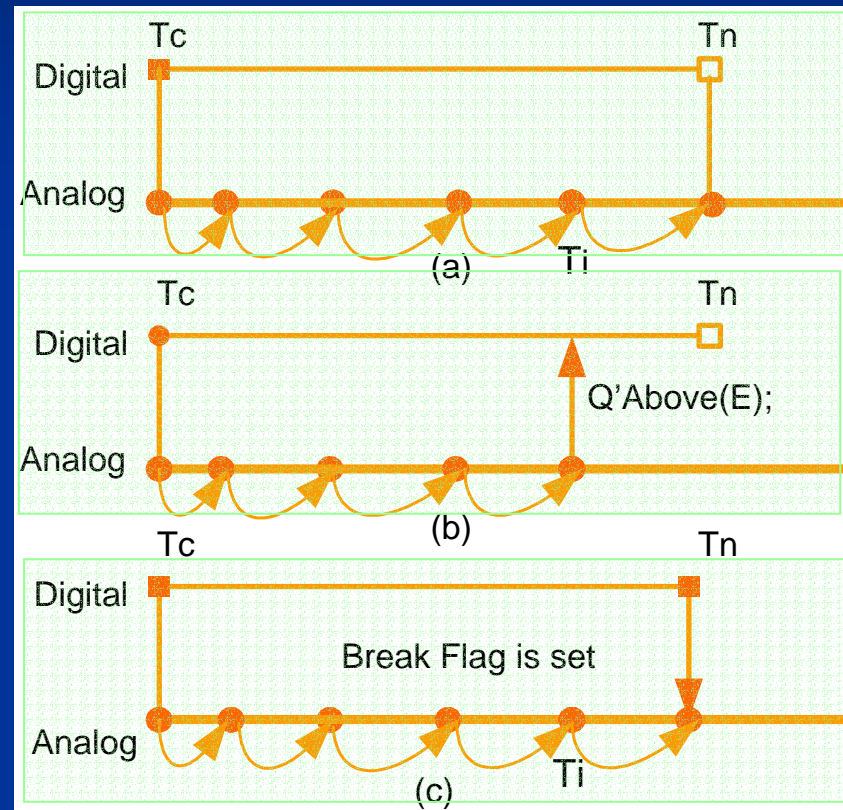
*Handle the events in current\_simulation\_time ;*

*Schedule new generated events in the event queue ;*

*Until (event queue is not empty);*

# Canonical Synchronization Algorithm

- Mixed oblivious and event driven algorithm
- The time between two digital events is advanced by analog solver and analog equations are solved by analog solver in these time units
- In event time, the digital kernel handles the Events.
- $Q'_{Above}(E)$  event suspends analog solver
- Break on quantities, break flag is set for analog solver



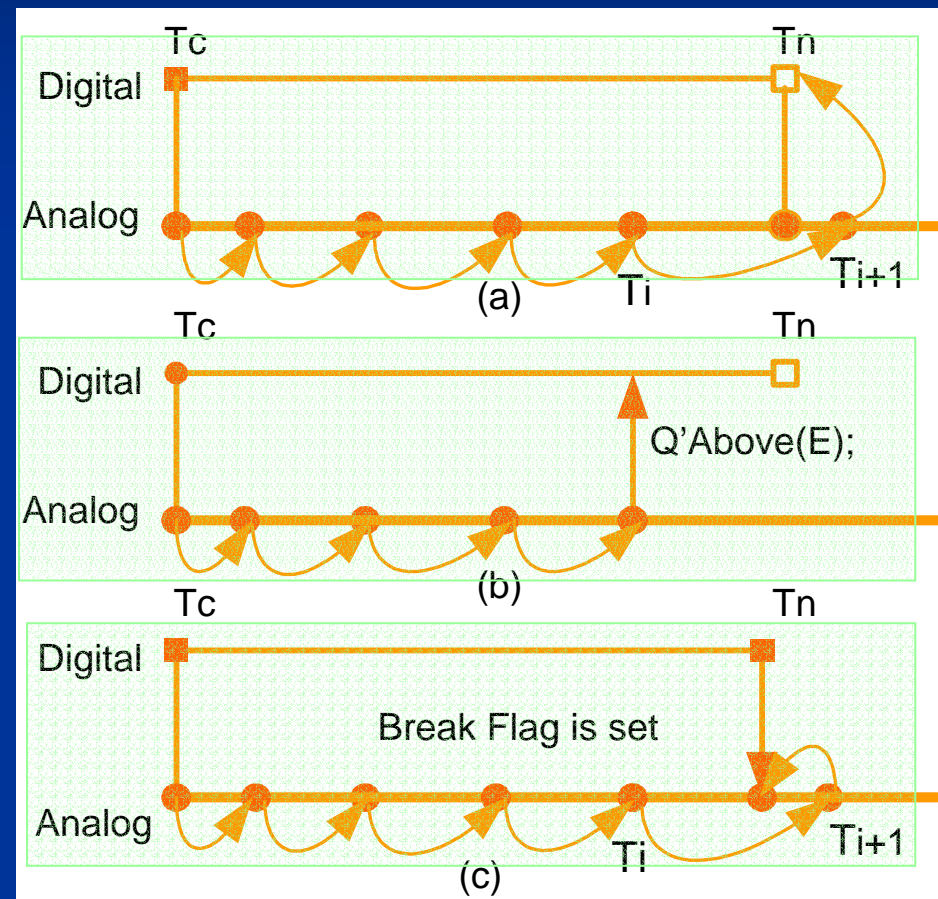
# Liyi Algorithm

The difference behavior  
at time-point  $T_n$ .

$$Q_n = Q_i + \frac{T_n - T_i}{T_{i+1} - T_i} (Q_{i+1} - Q_i)$$

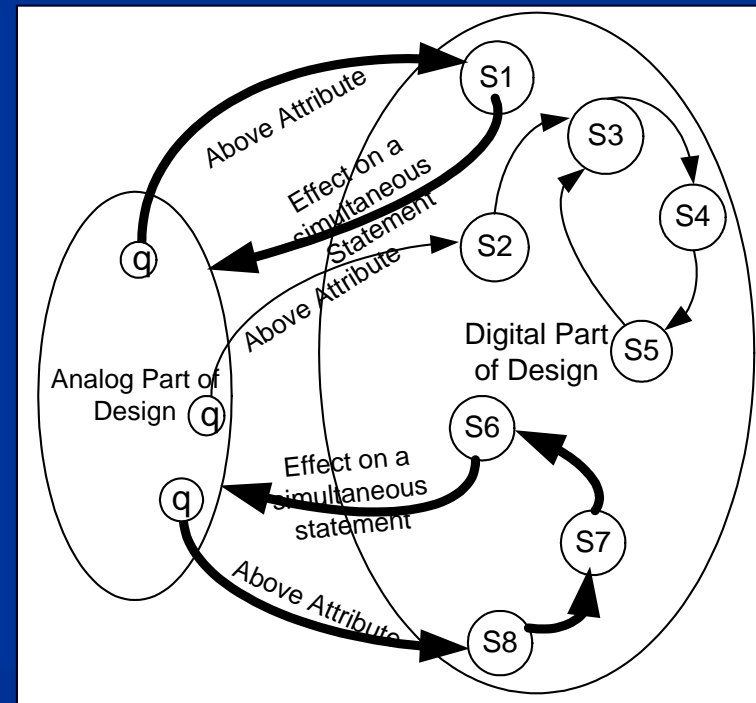
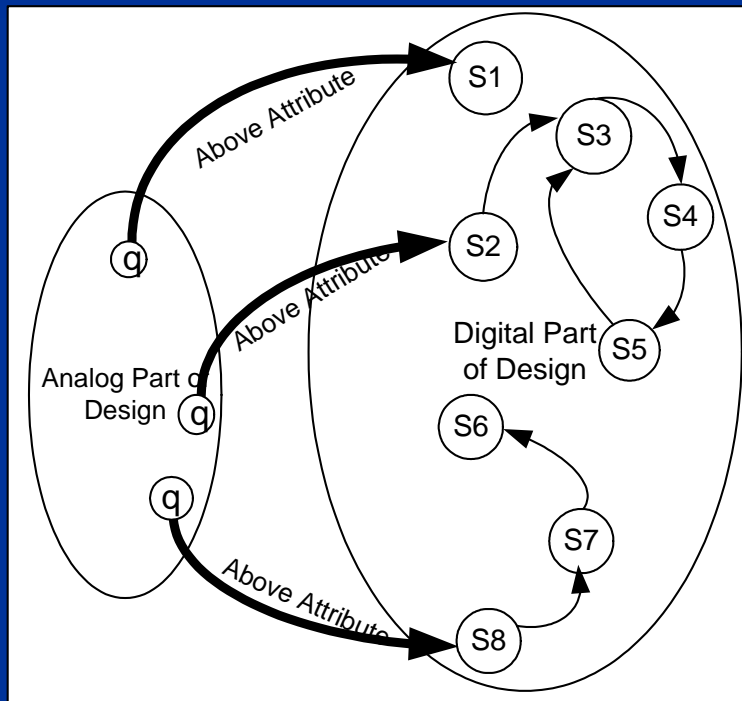
The same as canonical  
algorithm

The difference behavior  
at time-point  $T_n$ .

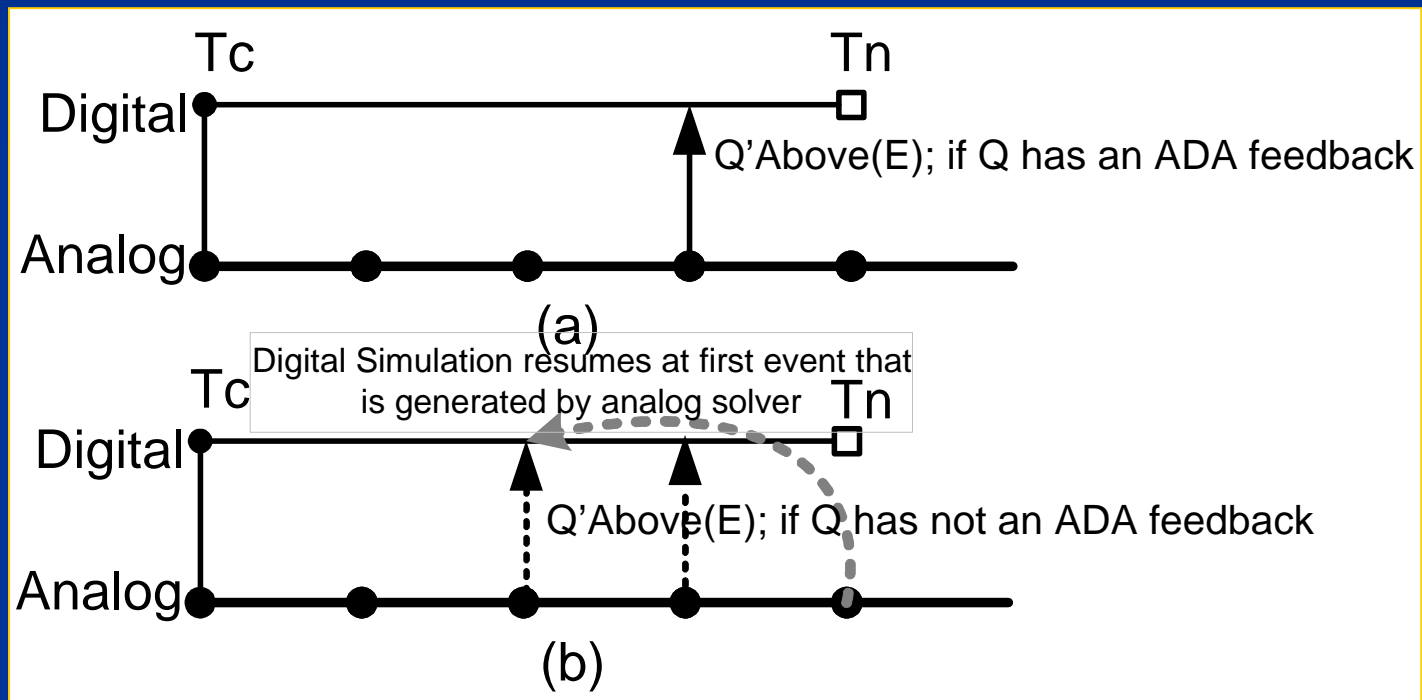


# Proposed Algorithm

- Basic idea: ADA feedback effect
  - Above attribute signals affect on only digital part of design
  - Above attribute signals affect on digital and analog part of design

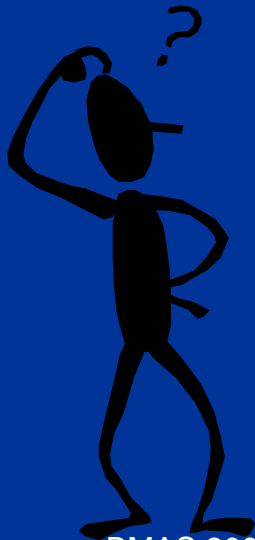


# Proposed Algorithm

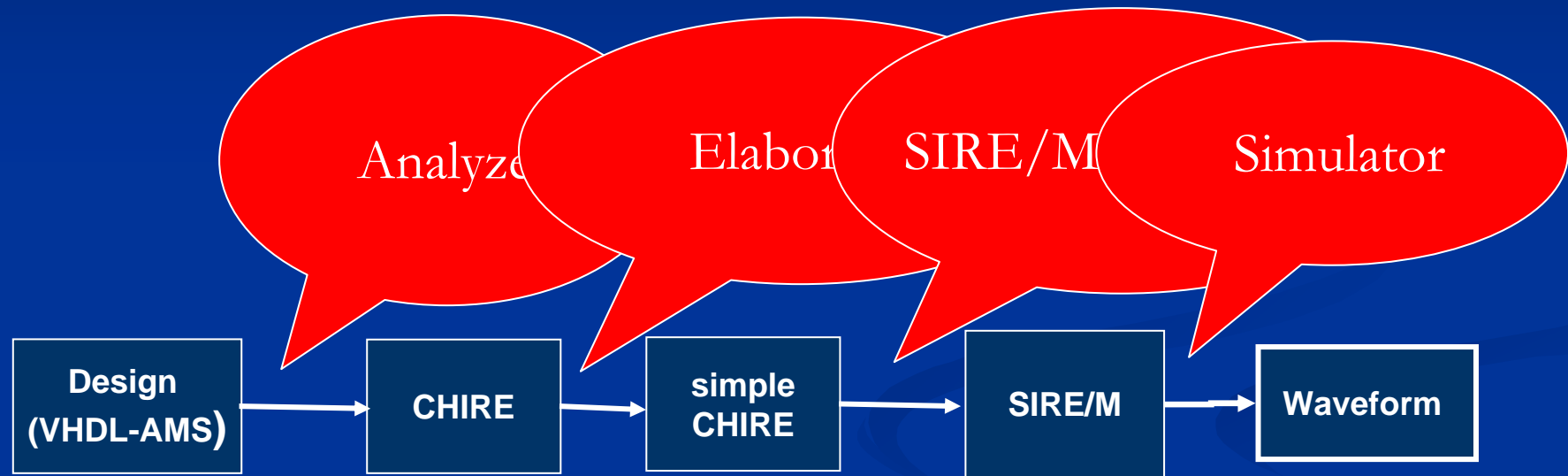


# Outline

- Introduction
- Simulation Process
- Compilation
- Elaboration
- SIRE/M Intermediate Format and Simulation Kernel
- Synchronization

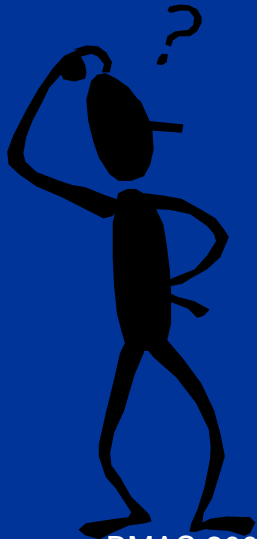


# Simulation Process

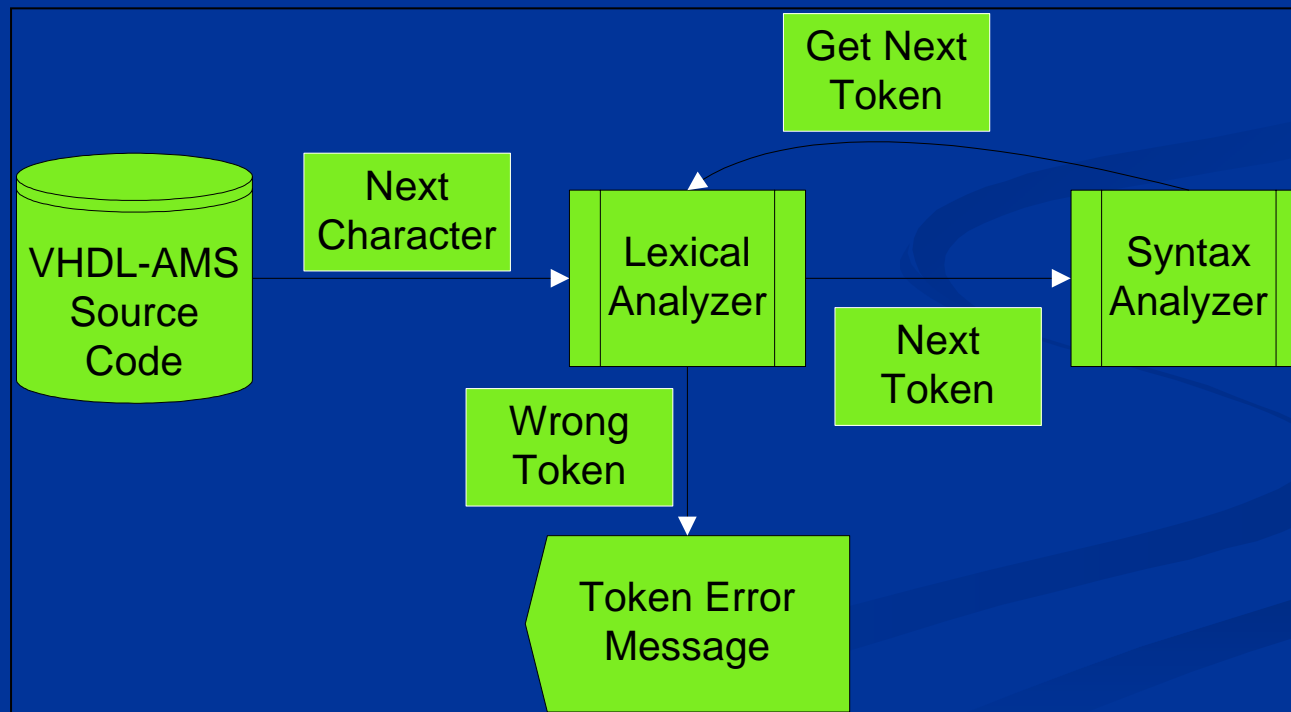


# Outline

- Introduction
- Simulation Process
- Compilation
- Elaboration
- SIRE/M Intermediate Format and Simulation Kernel
- Synchronization

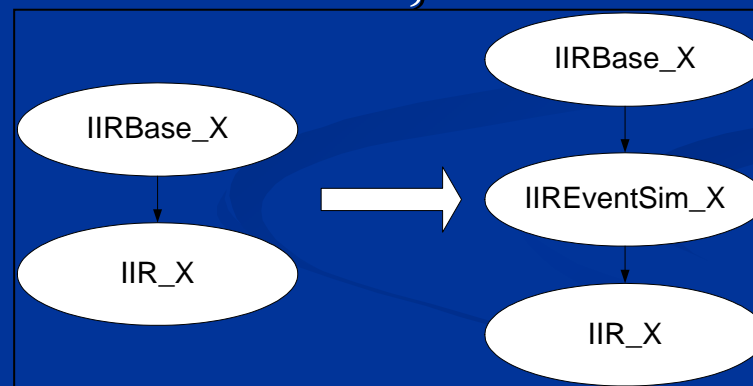


# Compilation



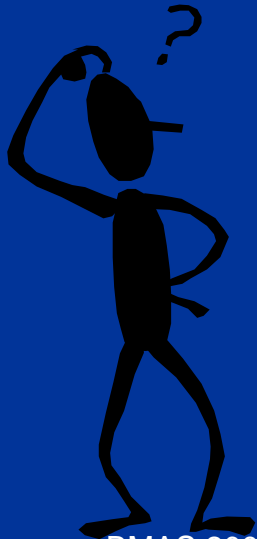
# CHIRE Data Structure

- Object Oriented Data Structure
- One-to-one correspondence to VHDL-AMS constructs
- Functions: Save to file, Load from file, Semantic Checking, Search,...
- Two level classes :
  - IIRBase\_X
  - IIR\_X
- Extension is a method for implementation of new application



# Outline

- Introduction
- Simulation Process
- Compilation
- Elaboration
- SIRE/M Intermediate Format and Simulation Kernel
- Synchronization



# Elaboration

- Input: CHIRE
- Output: Simplified CHIRE
- General Elaboration
  - Component Instantiation & Configuration Specification
  - Elimination of Generic Map Aspect & Port Map Aspect in Block Statement
  - Loop unrolling for If & For Generate Statement
- Simulation Specific Elaboration

# Example

```
entity Mux2x1 is port(a0, a1, s: in bit; o: out bit);
end Mux2x1;
architecture behavioral of Mux2x1 is
begin
    process (a0, a1, s)
    begin
        if (s = '0') then o <= a0; else o <= a1; end if;
    end process;
end behavioral;
entity TestMUX is end TestMUX;
architecture io of TestMUX is
Signal a0, a1, s, o : bit;
begin
u0 : Entity Work.Mux2x1 (behavioral)
    port map(a0 => a0, a1 => a1, s => s, o => o);
    a0 <= '1'; a1 <= '0'; s <= '1' AFTER 10 NS;
end io;
```

Elaboration



```
ENTITY TestMUX IS END TestMUX;
ARCHITECTURE io OF TestMUX IS
SIGNAL /io/a0:bit;
SIGNAL /io/a1:bit;
SIGNAL /io/s:bit;
SIGNAL /io/o:bit;
SIGNAL /io/BLOCK0/a0:bit;
SIGNAL /io/BLOCK0/a1:bit;
SIGNAL /io/BLOCK0/s:bit;
SIGNAL /io/BLOCK0/o:bit;
BEGIN
    /io/a0<=INERTIAL '1';
    /io/a1<=INERTIAL '0';
    /io/s<=INERTIAL '0','1' AFTER 10 NS;
    PROCESS(/io/BLOCK0/a0,/io/BLOCK0/a1,/io/BLOCK0/s)
    BEGIN
        IF /io/BLOCK0/s = 0 THEN
            /io/BLOCK0/o<=INERTIAL/io/BLOCK0/a0;
        ELSE
            /io/BLOCK0/o<=INERTIAL/io/BLOCK0/a1;
        END IF;
    END PROCESS;
    /io/BLOCK0/a0<=INERTIAL/io/a0;
    /io/BLOCK0/a1<=INERTIAL/io/a1;
    /io/BLOCK0/s<=INERTIAL/io/s;
    /io/o<=INERTIAL/io/BLOCK0/o;
END io;
```

# Simulation Specific Elaboration

- Above Attribute Elaboration

- Example :`Process ( Q'Above(E) )`

`Begin`

`...`

`End`

`Process (attribute0)`

`Begin`

`...`

`End`

`Attribute0 <= Q'Above(E);`

- Topology Equation Extraction

- All different topology equations in the circuits are declared in the declaration part of design
- They should be extracted for analog simulation

- Concurrent & Simultaneous Statement Partitioning

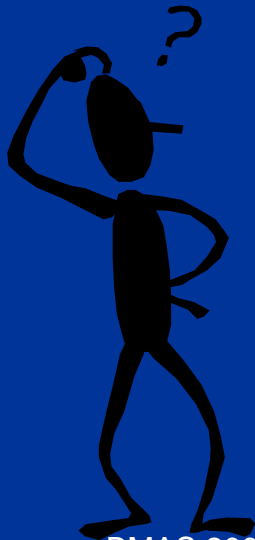
- Ground Terminal

- Find all ground terminals and change all of them to `electrical_reference` terminal
- They may be alias names

- Quantity partitioning based on ADA feedback

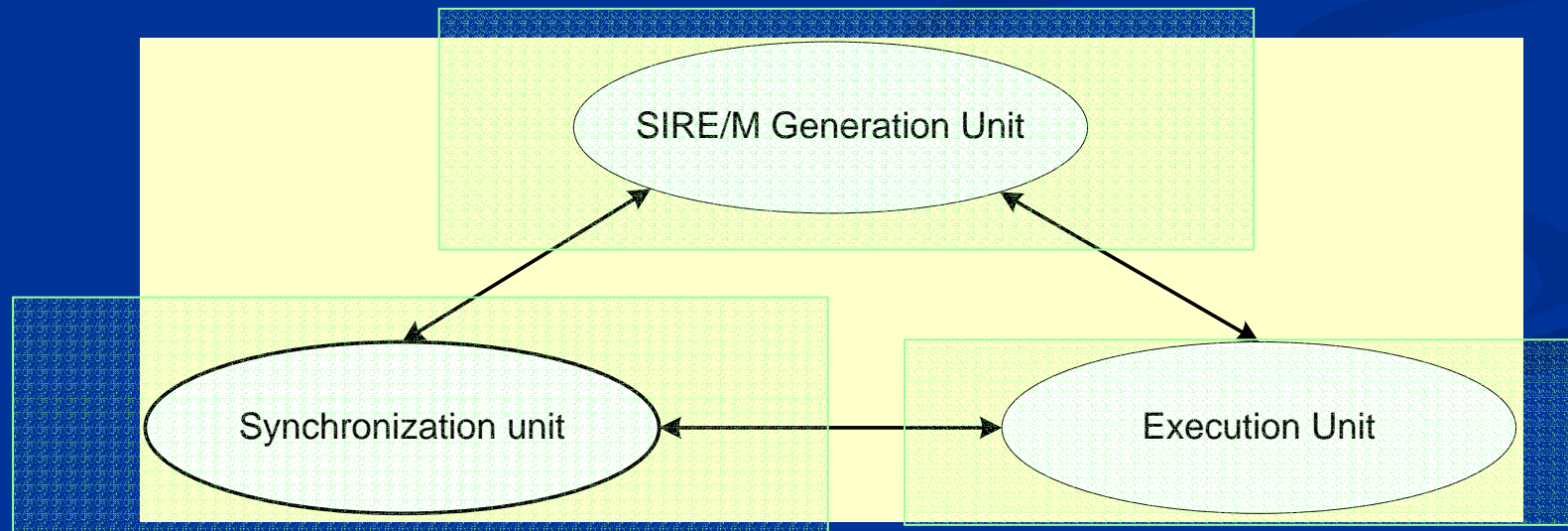
# Outline

- Introduction
- Simulation Process
- Compilation
- Elaboration
- SIRE/M Intermediate Format and Simulation Kernel
- Synchronization



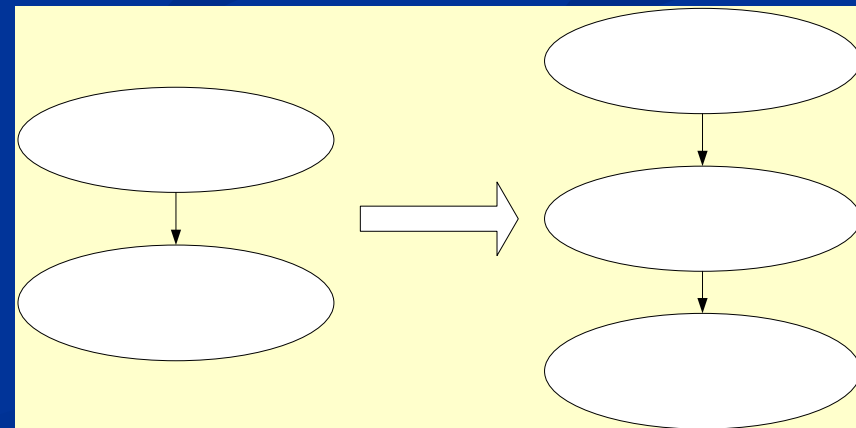
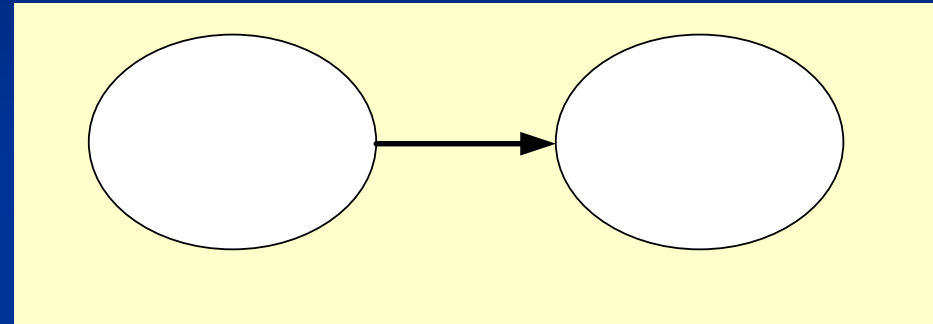
# Kernel of Simulator

- Includes these main units
  - Simulation specific elaborator and SIRE/M generator unit
  - Synchronization unit
  - Execution unit



# SIRE/M Generator Unit

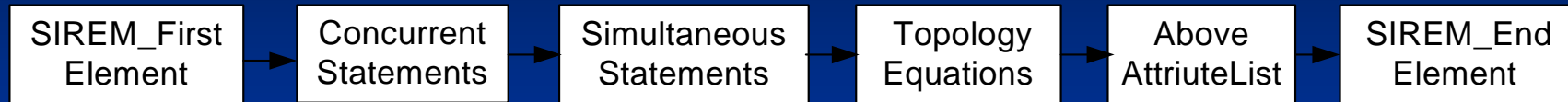
- Input : simplified CHIRE
- Output : SIRE/M data structure
- Traverse on CHIRE and generate SIRE/M data structure
- This unit is embedded in CHIRE data structure



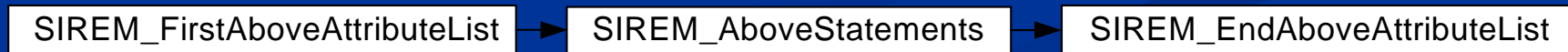
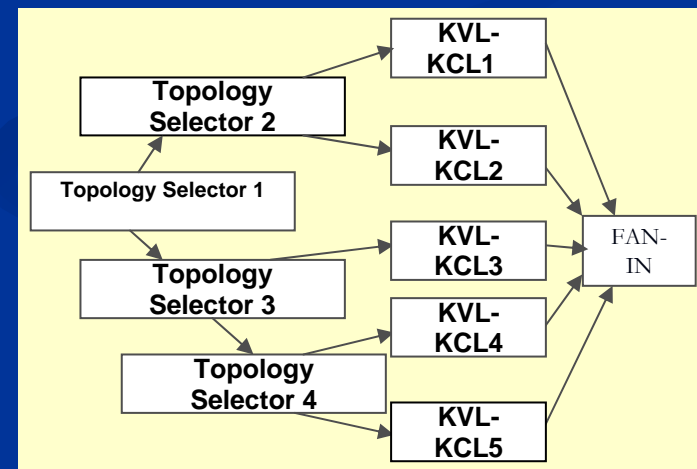
# SIRE/M Intermediate Format

- Execution unit uses this format for simulation
- Digital and Analog description can be simulated by this format
- Digital or analog behavior of each SIRE/M element depends on simulation phase
- Language independence
  - It can be used to simulate Verilog or Verilog-AMS
- Post order notation: **stack based execution**  
In-order :  $S \leftarrow 1+5*7$   
Post order :  $S \ 1 \ 5 \ 7 \ * \ + \ \leftarrow$

# General SIRE/M Data Structure



This format is generated for every VHDL-AMS source code

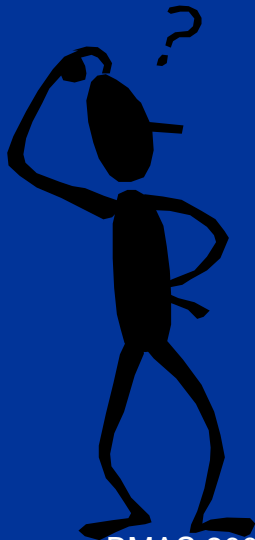


# Execution Method

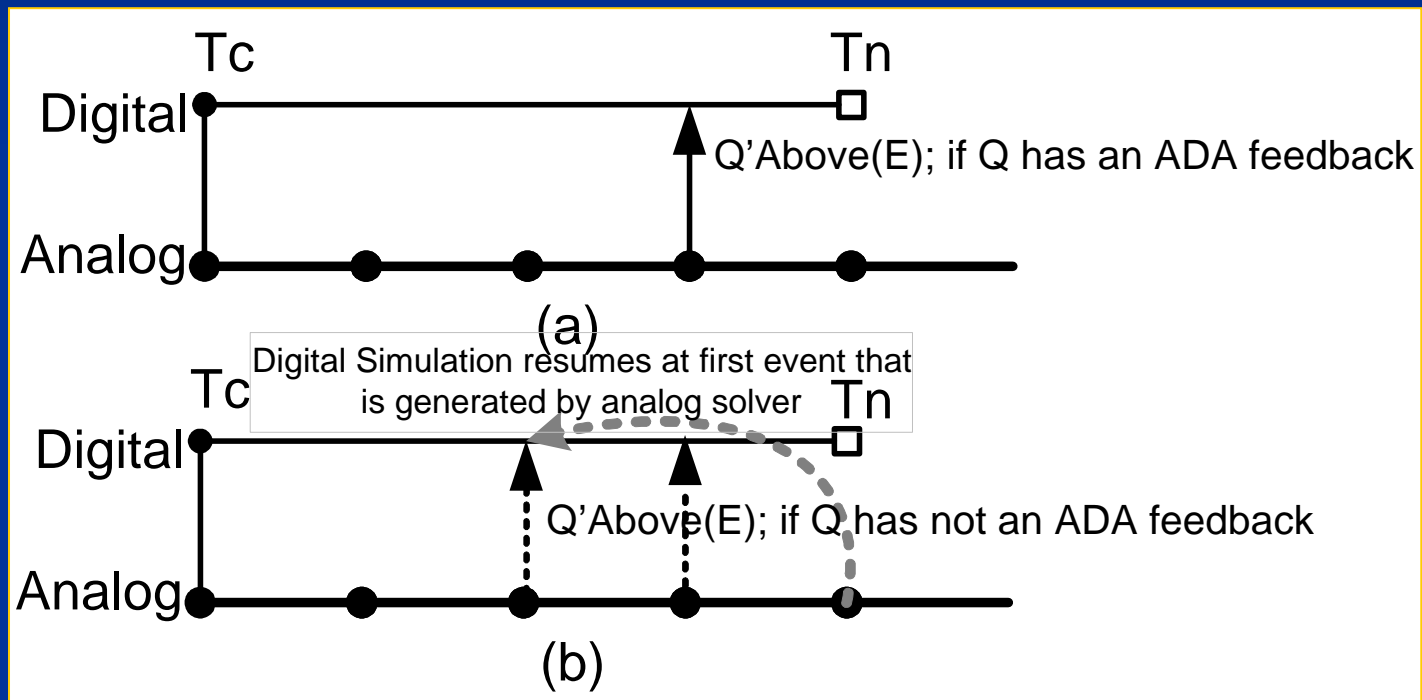
- Simulation Phases
  - DIGITAL
  - ANALOG\_INDEXING: selecting the equations
  - ANALOG\_INITVAL: calculating initial values
  - ANALOG\_NEXTVAL: generating time-slice & calculating values with new time-slice
- Synchronization unit orders the statements for execution
- Execution unit passes the events to synchronization unit

# Outline

- Introduction
- Simulation Process
- Compilation
- Elaboration
- SIRE/M Intermediate Format and Simulation Kernel
- Synchronization



# Proposed Algorithm



# Test case for simulation

- We notify these points in our test cases
  - The number of above signals
  - The number of above signals with ADA Feedback Effect
- We design and implement 4 examples :
  - ADC 8 and 16 bit output
  - Clock Generator with different duty cycles
- Environment
  - CPU : P4 2 GHz
  - Memory : 1 GByte

# Proposed Algorithm Vs. Liyi Algorithm

Models	Liyi	Proposed Algorithm	Speed Up
ADC_8	140.72 s	100.61 s	28.5 %
ADC_16	272.91 s	171.38 s	37.2 %
ClkGen1	14.75 s	12.00 s	18.6 %
ClkGen2	21.38 s	17.25 s	19.3 %

Simulation time is 1 second

# Performance of proposed algorithm

- Resuming analog simulation needs these phases
  - Reset all indexes and evaluate all conditions and select new topology and equations
  - Calculate Initial condition
  - Solve the equations
- For each time-point which we eliminate switching between analog and digital simulation, the first and second phases of simulation will not be done for that time-point

# Performance of proposed algorithm [1]

- The time-slice of the analog simulation increases step by step :



- When analog simulation stops and resumes, the time-slice is set from first small slice again, so a time-slice may be partitioned into two , three or ... time-slices.
- If we reduce switching between analog and digital simulation, the analog time-points will be reduced

Thank you  
Question ?

# Example of SIRE/M Data Structure for VHDL-AMS

```
ENTITY ShowSIREM IS END ShowSIREM;
ARCHITECTURE example OF ShowSIREM IS
```

```
Terminal Ta, Tb : ELECTRICAL;
quantity Vsrc Across Isrc Through Ta ;
quantity Vr Across Ir Through Ta to Tb;
quantity Vc Across Ic Through Tb;
quantity VI Across II Through Ta;
```

```
constant R1: real := 25.0;
constant C1: real := 0.000001;
constant L1: real := 0.01;
Constant Vth: real := 2.5 ;
Signal Switch : Boolean ;
```

```
BEGIN
```

```
    Vr == Ir * R1;
```

```
    if NOW < 2.5 ms use
```

```
        Vsrc == 5.0;
```

```
    else
```

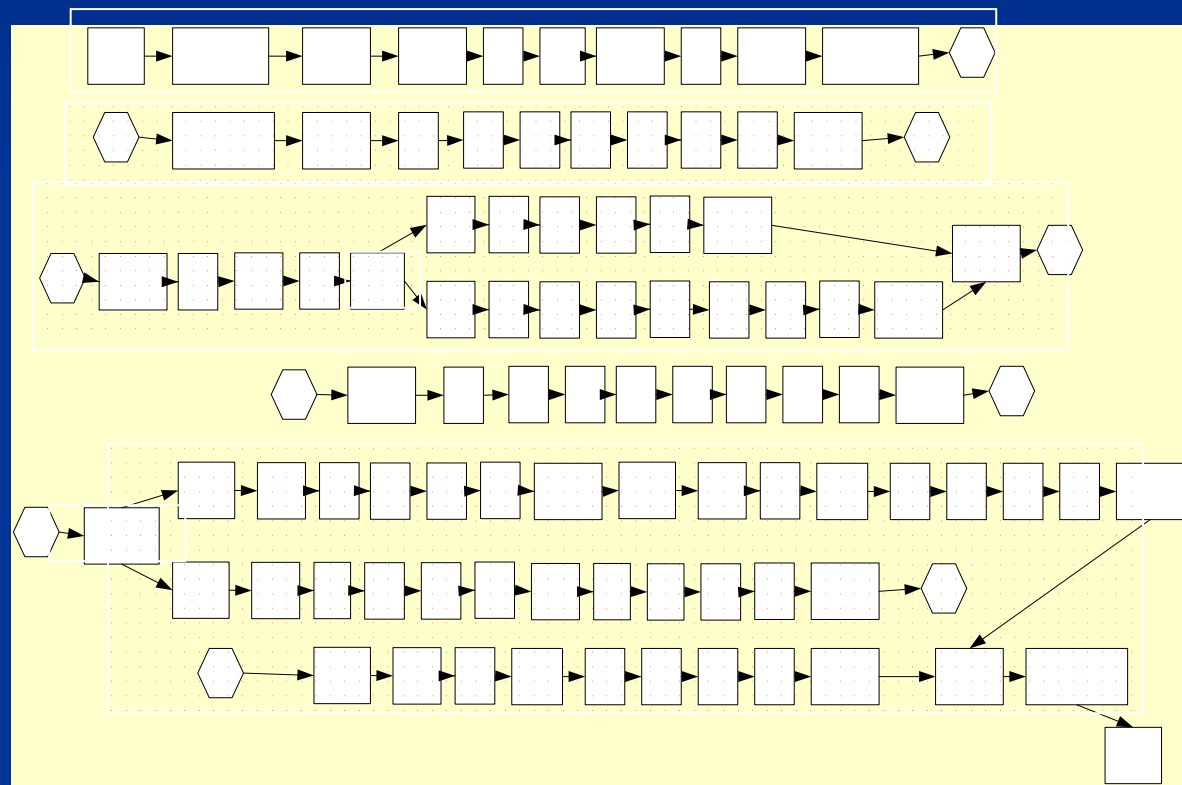
```
        VI == L1 * II'dot;
```

```
    end use;
```

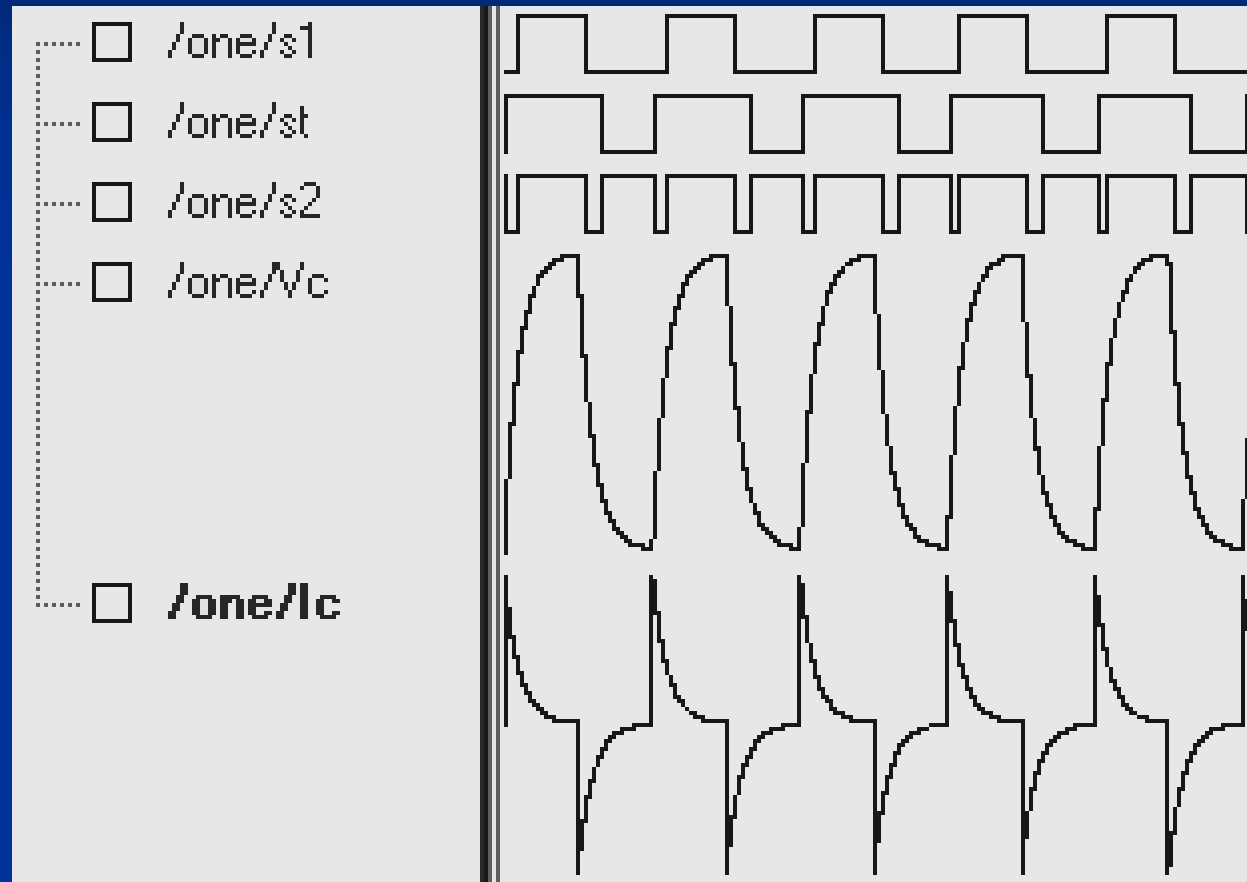
```
    Ic == C1 * Vc'dot;
```

```
    Switch <= Vc'above(Vth)
```

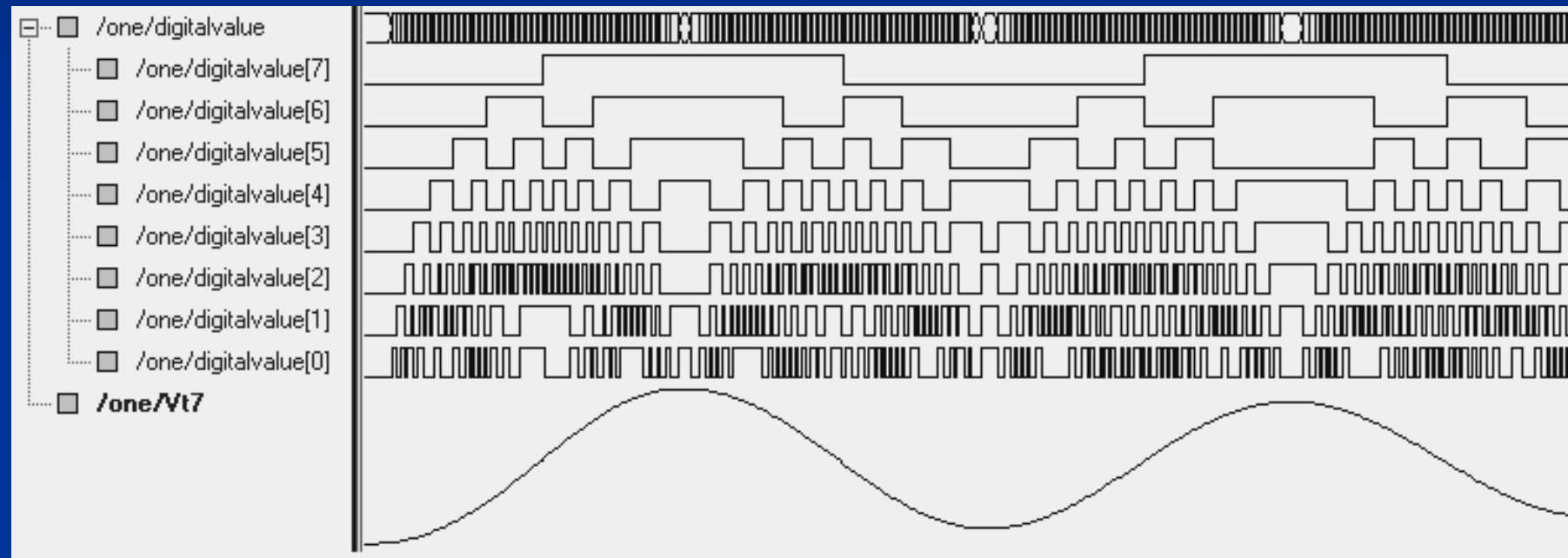
```
END two;
```



# Waveform of ClockGen1



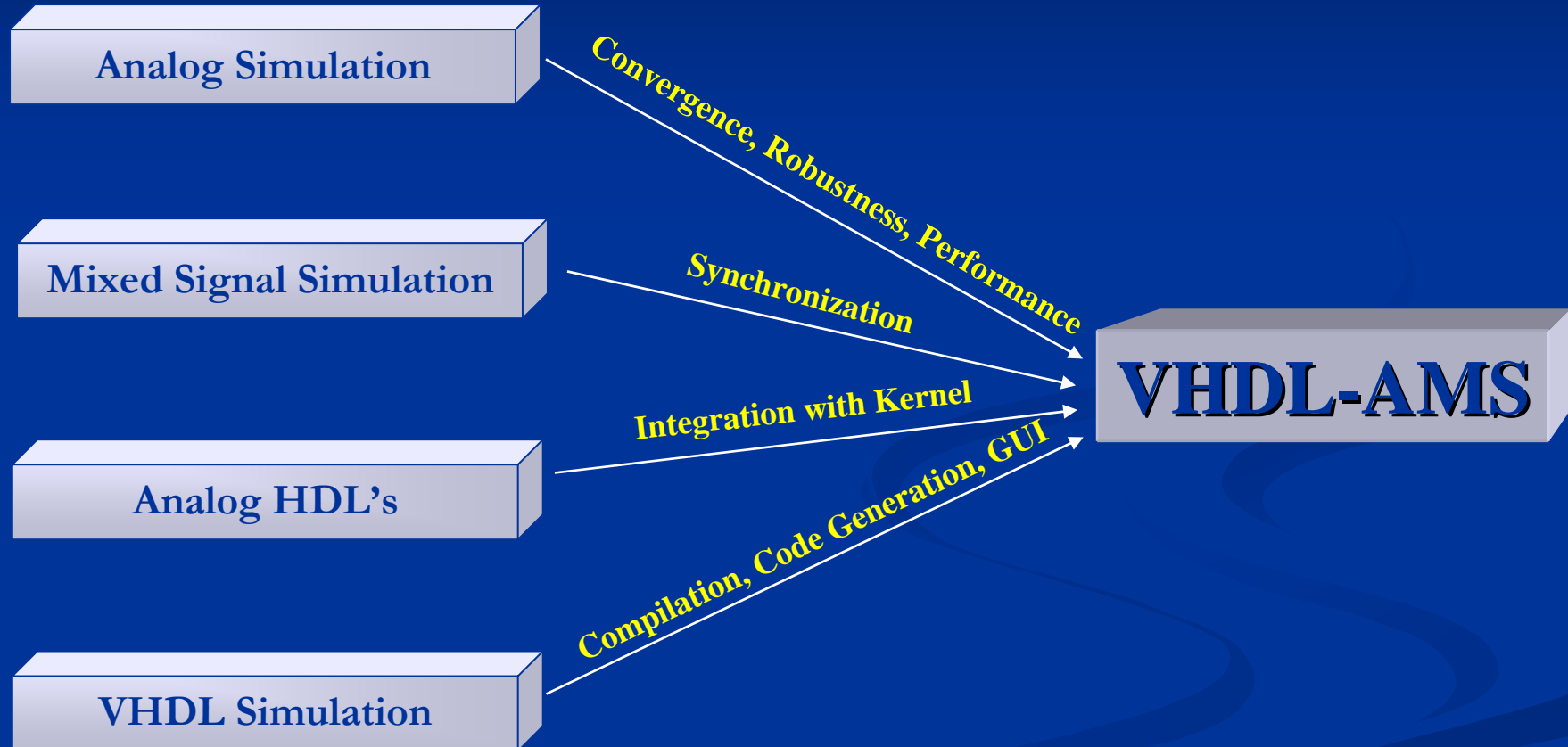
# Waveform of ADC\_8



# Software Specification


- More than 450 C++ classes
- More than 50,000 lines source code
- The source code is pure C++
- The project is a DLL that can be used in other projects
- Support all VHDL-AMS Construct

# VHDL-AMS Expertise



# Example of Analog Elaboration Process [1]

```
library IEEE;
use ieee.electrical_systems.all;
ENTITY Resistor IS Generic(R : Real); Port(Terminal Tp, Tm : Electrical); END
Resistor;
ARCHITECTURE Structural OF Resistor IS
    quantity Vr Across Ir Through Tp to Tm;
BEGIN
    Vr == Ir * R;
END Structural;
-----
ENTITY Capasitor IS Generic(C : Real); Port(Terminal Tp, Tm : Electrical); END
Capasitor;
ARCHITECTURE Structural OF Capasitor IS
    quantity Vc Across Ic Through Tp to Tm;
BEGIN
    Ic == C * Vc'dot;
END Structural;
-----
ENTITY Vsrc IS Generic(Value : Real); Port(Terminal Tp : Electrical); END Vsrc;
ARCHITECTURE Structural OF Vsrc IS
    quantity Vsrc Across Tp;
BEGIN
    Vsrc == Value;
END Structural;
-----
ENTITY RC IS END RC;
ARCHITECTURE Structural OF RC IS
    ...
    Terminal T1, T2 : Electrical;
BEGIN
    I1 : v1 Generic Map(5.0) Port Map(T1);
    I2 : r1 Generic Map(1000.0) Port Map(T1, T2);
    I3 : c1 Generic Map(0.000001) Port Map(T2, GROUND);
END Structural;
```



Before  
elaboration  
process



**Component Instantiation**

# Example of Elaboration Process [2]

```
ENTITY RC IS
END RC;
ARCHITECTURE Structural_ELB OF RC IS
Terminal /Structural/T1:ELECTRICAL
Terminal /Structural/T2:ELECTRICAL
Quantity /Structural/I1/Vsrc across /Structural/T1 to ELECTRICAL_REF;
CONSTANT /Structural/I1/Value:real := 5.000000e+000;
Quantity /Structural/I2/Vr across /Structural/T1 to /Structural/T2;
Quantity /Structural/I2/Ir through /Structural/T1 to /Structural/T2;
CONSTANT /Structural/I2/R:real := 1.000000e+003;
Quantity /Structural/I3/Vc across /Structural/T2 to ELECTRICAL_REF;
Quantity /Structural/I3/Ic through /Structural/T2 to ELECTRICAL_REF;
CONSTANT /Structural/I3/C:real := 1.000000e-006;
Quantity /Structural/I3//ATTRDUMMY0: CURRENT;
BEGIN
/Structural/I1/Vsrc == /Structural/I1/Value;
/Structural/I2/Vr == /Structural/I2/Ir * /Structural/I2/R;
/Structural/I3//ATTRDUMMY0 == /Structural/I3/Vc'Dot;
/Structural/I3/Ic == /Structural/I3/C * /Structural/I3//ATTRDUMMY0;
END Structural_ELB;
```



After  
elaboration  
process

# Component Instantiation & Configuration Specification

```
entity MyAnd is
  generic (g : bit := '0');
  port(a, b : in bit; o : out bit);
end MyAnd;
architecture Behavioral of MyAnd is
begin
  o <= a AND b;
end Behavioral;

entity testAND is end testAND;
architecture St of testAND is
  Signal ax, bx , ox : bit;
  component AnAnd
    generic (g : bit := '0');
    port(a, b : in bit; o : out bit);
  end component;
  for all : AnAnd USE ENTITY
    work.MyAnd(Behavioral) port map (a=>a, b=>b, o=>o);
begin
  u0 : AnAnd generic map ('0') port map (a => ax, b => bx, o => ox);
end St;
```



Elaboration

```
ARCHITECTURE St OF testAND IS
SIGNAL ax:bit;
SIGNAL bx:bit;
SIGNAL ox:bit;
BEGIN
u0 : BLOCK IS
  CONSTANT g:bit := '0';
  SIGNAL a:bit;
  SIGNAL b:bit;
  SIGNAL o:bit;
  BEGIN
    o <= a AND b; //component internal logic
    a <= ax; //Block interfaces
    b <= bx; //Block interfaces
    ox <= o; //Block interfaces
  END BLOCK;
END St;
```

# Generic Map Aspect & Port Map Aspect in Block Statement

```
entity BlockTest is
    port(a, b : in bit; o : out bit);
end BlockTest;
architecture Behavioral of BlockTest is
begin
    b0: block
        generic (x : bit := '1'); generic map (x => '0');
        port (aa, bb : in bit; oo : out bit); port map(aa=>a, bb=>b ,oo=>o);
    begin
        oo <= aa AND bb;
    end block b0;
end Behavioral;
```

Elaboration



```
ARCHITECTURE Behavioral OF BlockTest IS
BEGIN
    b0 : BLOCK IS
        CONSTANT x:bit := '0';
        SIGNAL aa:bit;
        SIGNAL bb:bit;
        SIGNAL oo:bit;
    BEGIN
        oo<=aa AND bb;           //Block internal statement
        aa<=a;                   //Block interfaces
        bb<=b;                   //Block interfaces
        o<=oo;                   //Block interfaces
    END BLOCK;
END Behavioral;
```

# If & For Generate Statement

```
LABL : FOR I IN 1 TO 2 GENERATE  
    SIGNAL S1: INTEGER;  
BEGIN  
    S1 <= P1;  
    Inst1 : and_gate PORT MAP ( S1 , P2(I) , P3 );  
END GENERATE LABL;
```

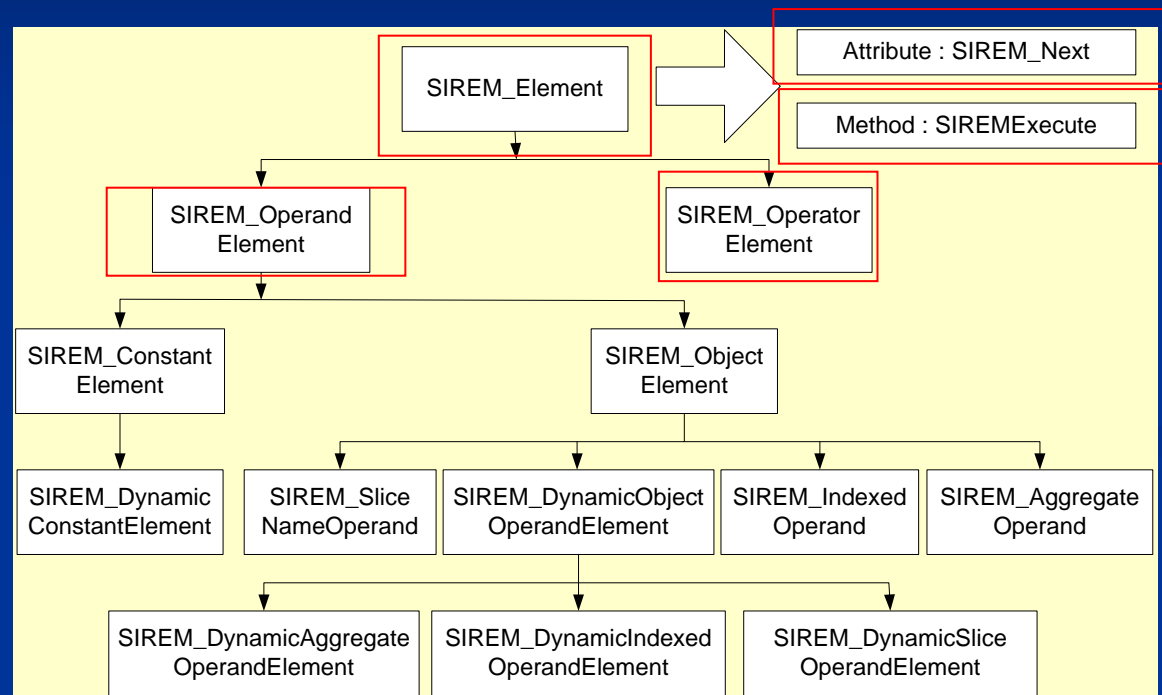


Elaboration

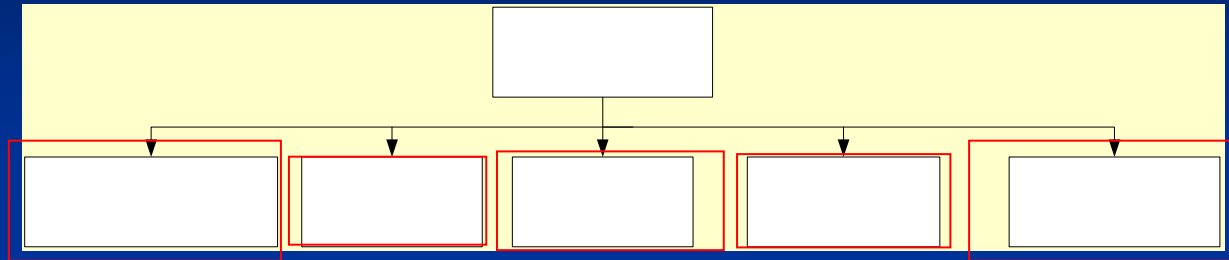
```
LABL1 : BLOCK  
    CONSTANT I : INTEGER := 1;  
    SIGNAL S1: INTEGER;  
BEGIN  
    S1 <= P1;  
    Inst1 : and_gate PORT MAP ( S1 , P2(I) , P3 );  
END BLOCK LABL1;  
  
LABL2 : BLOCK  
    CONSTANT I : INTEGER := 2;  
    SIGNAL S1: INTEGER;  
BEGIN  
    S1 <= P1;  
    Inst1 : and_gate PORT MAP ( S1 , P2(I) , P3 );  
END BLOCK LABL2;
```

# SIRE/M Classes - Operand

- SIRE/M element
  - SIREM\_Next
  - SIREMExecute
- Two main classes
  - Operand element
  - Operator element

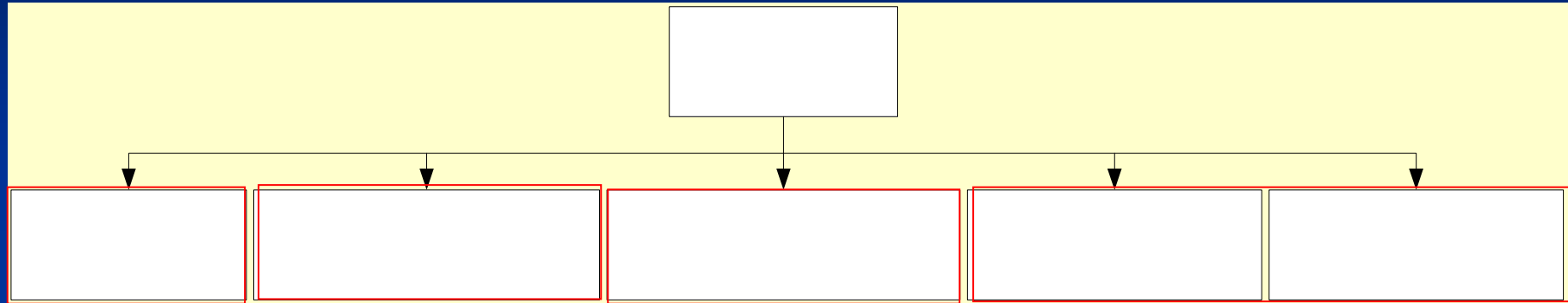


# SIRE/M Classes -Operator



- Operators (number of operands)
  - Monadic (not, unary minus,...)
  - Dyadic (add, sub, and, ...)
  - Triadic (assignment,...)
  - Extended (branch, ...)
  - No Operand (“NOW” function)

# SIRE/M Classes-Extended Operator

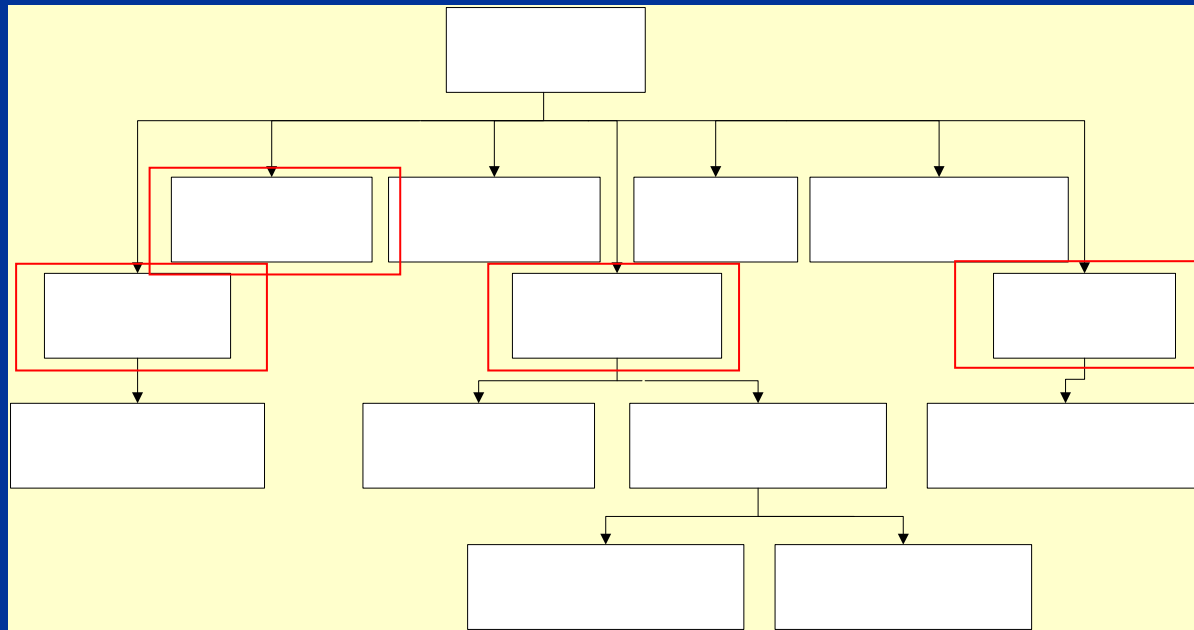


- Extended operators
  - If , simultaneous If
  - Case, simultaneous case
  - Procedure call, function call,
  - topology selection (Analog)



# SIRE/M Object Declaration Classes

- Signals, variables, ...
- Terminal, quantity, ...

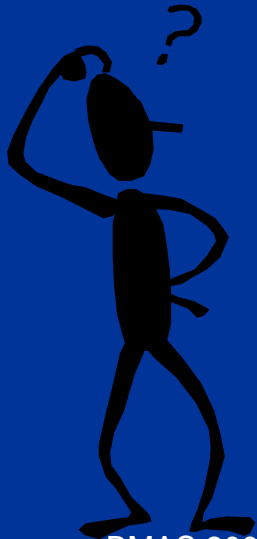


# Analog Solver

- Newton algorithm to solve DAE system (Differential and Algebraic Equation)
- Sundial library (public domain, open source)
  - Solve initial value condition
  - Solve DAE Equations
- We customize this library for our SIRE/M classes

# Outline

- Introduction
- Simulation Process
- Compilation
- Elaboration
- SIRE/M Intermediate Format and Simulation Kernel
- Synchronization
- Future Work



# Future Work

- Developing distributed and parallel analog simulation in mixed signal simulation
- Using PDES, P2EDAS, YADDES algorithm for mixed signal simulation

# Simulation Kernel Properties

- Modularity
- Independent interface among units
- O.S. independent memory management
  - Autonomous memory allocation and de-allocation
- Powerful utility package
  - To traverse a complicated graph

# SIRE/M Memory Management

