System Level Specification Beyond RTL

Tutorial

Design and Test Conference in Europe

Jianwen Zhu

Electrical and Computer Engineering

University of Toronto

March 4th, 2002

jzhu@eecg.toronto.edu

http://www.eecg.toronto.edu/~jzhu

The Language Myths

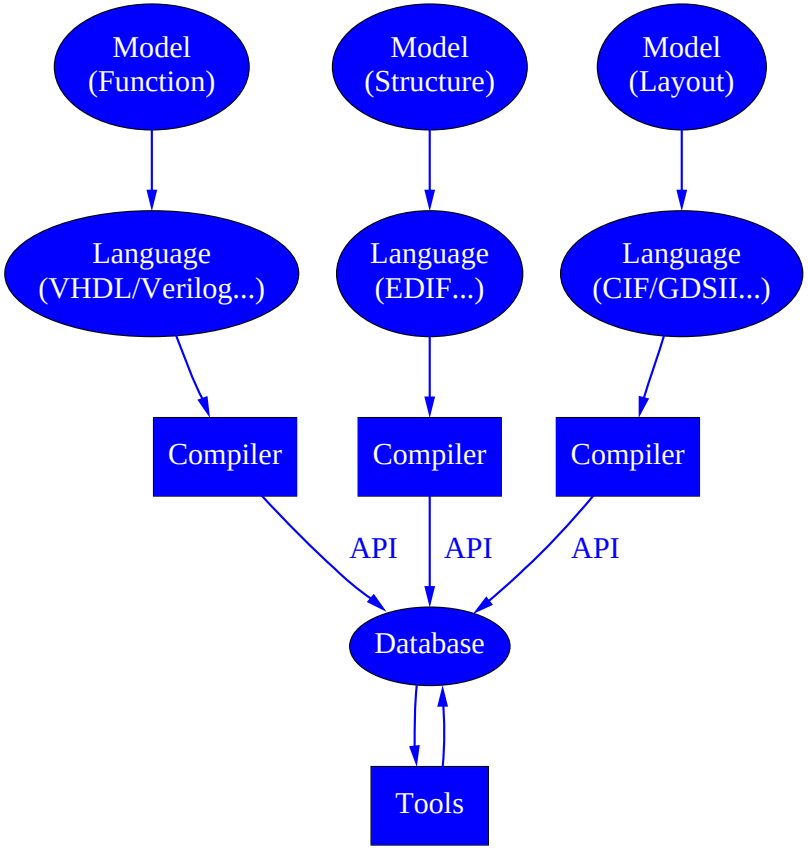
- The Language Panacea Myth
  - Language is only as good as the model it captures
  - Language is only as good as the tool that supports it
- The Single Language Myth
  - As few as possible, but not fewer
- The Standard Language Myth
  - Another billion-dollar mistake?
- The C/C++ Language Myth
  - New semantics for old constructs introduce new language

Outline

- *Overview*
- Design models
- Problems of HDLs
- Languages beyond RTL

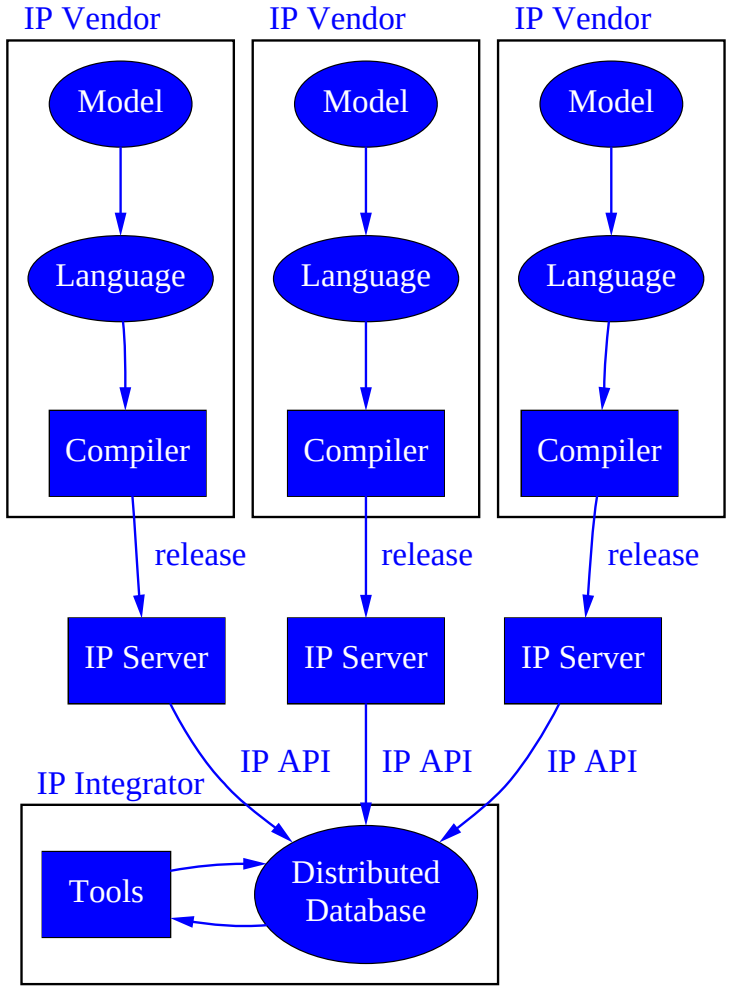
Languages in Traditional CAD Infrastructure

- Models
  - Functional model
  - Structural model
  - Layout model
  - Metric model
- Languages
  - Standardized
- Design Database
  - Largely proprietary, unintegrated
- Tools



Languages in IP-Centric CAD Infrastructure

- IP Components
  - Any reusable design artifacts
  - Soft, Firm, Hard
  - Functional, Architectural, Physical
- Toronto **ipsuite** Infrastructure
  - Standardization at binary level
  - Definition of IP creation API with COM interfaces
  - IP API Implemented by IP Integrator's CAD framework
  - IPs are remote software components invoking IP API via middleware



New Challenges for Language Development

- The Content
  - Functional IP
    - Redefining RTL sign-off
    - Raising abstraction level beyond RTL
  - Non-Functional IP
    - Redefining physical sign-off
    - Introducing architectural IP
- The Form
  - What are the right languages?
  - How to incorporate legacy design?
  - Different models of an IP are related: How do languages cross reference each other?

Outline

- Overview
- *Design models*
- Problems of HDLs
- Languages beyond RTL

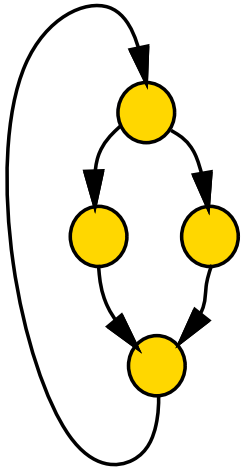
Functional Model

- Signals: function from time to value
- Goal: establish relationship between system input and output signals
- Denotational semantics
- Operational semantics: describe how system evolves with the reference set **time**
  - Identify system state S
  - Express O as trivial function of S and I
  - Key: models state change

$$\begin{aligned} \frac{dS}{dt} &= f(S, I) \\ O &= g(S, I) \end{aligned}$$

Functional Model: Finite Automata

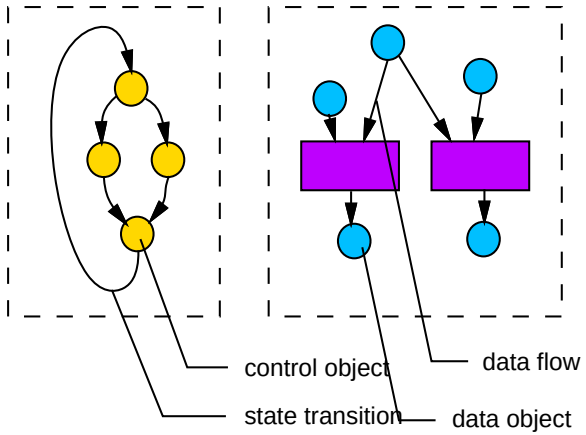
- $FSM = \langle S, f : S \mapsto S, s_0 \rangle$
- Application
  - Software: lexical analysis, real time control, UI
  - Hardware: random logic
- Advantage: general (Turing complete)
- Disadvantage: too general (state explosion)



Functional Model: FSMD

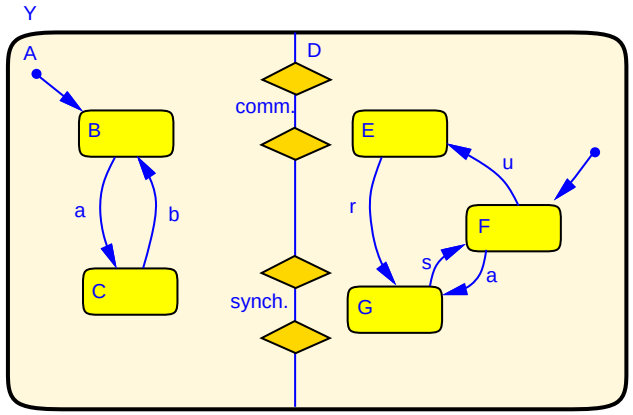
- Separation of Control and Computation
  - Object view: partition state space into
    - Control objects S
    - Data objects D
  - Dynamic view f:
    $S(next) = f(S, D)$
  - Functional view g:
    $D(next) = g(S, D)$

- Finite State Machine with Data (Timed, untimed)
- Refinement of data object modeling
  - Object Modeling Technology (OMT, Rumbaugh 1991)
  - Unified Modeling Language (UML)



Functional Model: Concurrent FSMD (CFSMD)

- Separating threads of control
- $CFSMD = \langle FSMD_1, ...FSMD_n \rangle$
- Processes are interacting:
  - Communication: processes sharing computational states
  - Synchronization: processes sharing control states



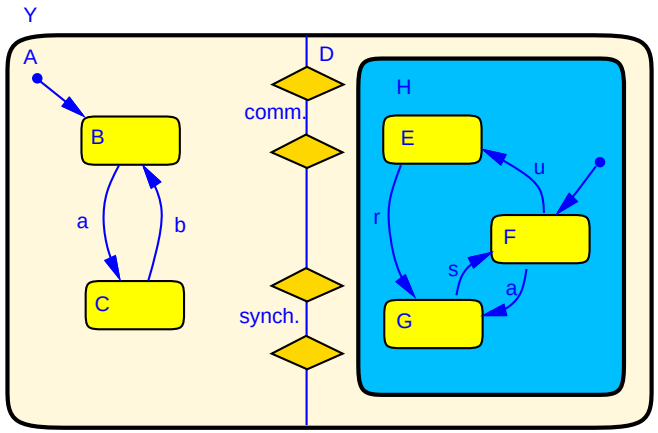
Functional Model: Hierarchical Concurrent FSMD

- State chart (Harel 1987)

  Each control state can be recursively defined by another machine.

- Program state machine (PSM, Gajski et. al. 1994)

  Each object can be a program, or sequential process.



Architectural Model

- Micro-architecture Model
  - Enable the mapping of FSMD model to software or ASIC core
  - Instruction set
    semantics, encoding, assembly syntax
  - Application Binary Interface (ABI):
    calling convention, relocation, dynamic linking
  - Instruction-level parallelism (ILP)
    temporal and spatial
- Macro-architecture Model
  - Enable the mapping of application to system-level architecture
  - On-chip network
  - High-level communication protocol
  - RTOS abstraction
  - System configuration
  - Testing architecture

Physical Model

- Enable the generation of silicon masks
- Logic and Wire planning
- Delay metric
- Power metric
- Constraints
- Layout

Outline

- Overview
- Design models
- *Problems of HDLs*
- Languages beyond RTL

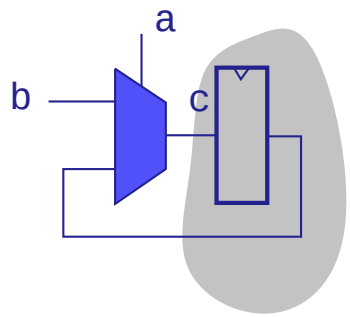
RTL Abstraction

- Importance of RTL
  - Semantics: Timed FSMD
  - Syntax: Synthesizable VHDL/Verilog (IEEE)
  - Established industrial standard for ASIC design
  - Backend interface for higher level design
  - *de facto* soft IP exchange standard
- Productivity improved by moving to behavioral/system level
- Better language even **at** RTL can improve productivity

Problems of HDLs: Simulation Semantics

- HDL designed for simulation
  - How to synthesize delay
  - How to synthesize signal
- synthesis subset
  - Problematic constructs excluded
  - Infer hardware that exhibits discrete event semantics

```
...
if( a = '0' ) then
    c <= b;
end if;
...
```

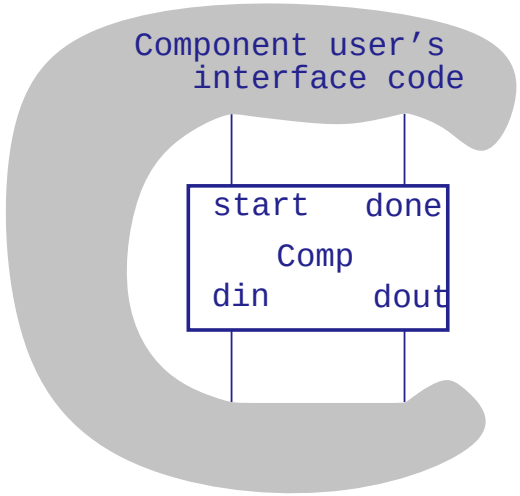


Problems of HDLs: Design Reuse

- Hardware reuse by component instantiation
- Not sufficient for sequential components

  No interface protocol captured

```
u1 : Comp( start, done, din, dout );
...
...
start <= '1';
for i in 0 to 8 do
    wait until clk'event and clk = '1';
    din <= a(i);
end for;
while( done = '0' ) do
    wait until clk'event and clk = '1';
    start <= '0';
end while;
while( done = '1' ) do
    wait until clk'event and clk = '1';
    b(j) := dout;
    j := j + 1;
end while;
...
```



Problems of HDLs: Type System

- Type system: most effective error prevention in software
- Untyped system in synthesizable HDL
  - enumerate type
  - bit vector
- More abstract data type needed at RTL level

Problems of HDLs: Memory Abstraction

- Any interesting application involves the use of memory
  - Multimedia: data sample storage
  - Networking: routing table, protocol states
- No abstraction of memory at RTL level
  - Interface protocol with memory
  - Dynamic allocation: pointer concept
  - Address calculation: array and record access

```
...                                          1
struct {                                     2
   int    field1;                            3
   struct {                                  4
      char field3;                           5
      } field2;                              6
   } *p1;                                    7
short a, *p2;                                8
char  b[10];                                 9
                                            10
...                                         11
a = 0;                                      12
b[3] = 'a';                                 13
p1->field1 = 1;                             14
p1->field2.field3 = 2;                      15
p2 = &a;                                    16
...                                         17
```


Outline

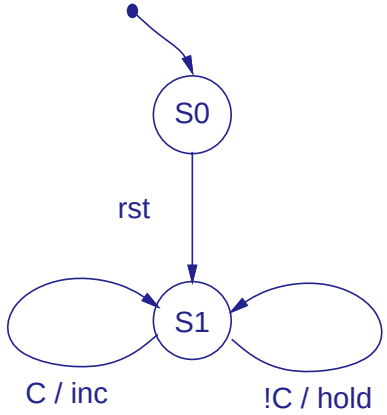
- Overview
- Design models
- Problems of HDLs
- *Languages beyond RTL*

Approaches for Language Definition

- Library Extension
  - Approach: new C++ class library, wrapper of IP API
  - Example: OCAPI
- Semantics Extension
  - Approach: new C++ class library, new semantics
  - Examples: SystemC, Cynlib
- Syntax Extension
  - Approach: extension of existing language
  - Examples: SpecC, Superlog, VHDL+, MetaRTL
- Extensible Language
  - Approach: language with extensible semantics
  - Examples: XML, Rosetta, Babel

One's Counter "System"

- Not really a big system (a.k.o hello world)
- Use it to compare different languages
- Disclaimer: an example taken from Schaumont et. al DAC'99
- Input: IB
- Output: OB
- Two states: s0, s1;
- Three RTL operations: rst, inc, hold




```
register C;
register N;
input IB;
output OB;

rst :  N = 0;
       C = IB;

inc:   N = N + 1;
       C = IB;
       OB = N;
hold: C = IB;
       OB = N;
```

Library Extension: OCAPI (DAC'99 paper)

- Developed at IMEC, Belgium
- A success combined with SOC++ methodology
- A C++ wrapper around IP API
- Captures FSMD model
- Leverage C++ inheritance for "behavioral" reuse
- `http://www.imec.be/ocapi/`

```
clk ck;                                                    1
sig C(ck, 0), N(ck, 0), input, output;                     2
bus IB, OB;                                                3
                                                           4
sfg rst;                                                   5
N = 0; C = inputs;                                         6
rst << in(input, IB);                                      7
                                                           8
sfg inc;                                                   9
N = N+1; C = input; output = N;                            10
inc << in(input, IB) << out(output,OB);                    11
                                                           12
sfg hold;                                                  13
C = input; output = N;                                     14
hold << in( input, IB) << out(output,OB);                  15
                                                           16
fsm ones_cnt;                                              17
state S0, S1;                                              18
ones_cnt << deflt(S0);                                     19
ones_cnt << S1;                                            20
S0 << always << rst << S1;                                 21
S1 << cnd(C) << inc << S1;                                 22
S1 << !cnd(C) << hold << S1;                               23
```


Library Extension: Pros and Cons

- Pros:
  - Functional model can be captured in OO fashion

    But it doesn't capture an OO model
  - No need for compiler frontend
    - Simulation can be done by interpretation
    - Simulation can be done by code generation
    - Synthesis can be done by code generation
  - Well defined synthesis semantics
- Cons:
  - Stretches the C++ syntax to the limit
  - Specification not as concise as approaches shown later

Semantics Extension Example: SystemC (Version 2.0)

- The Open SystemC Initiative
- A rich set of data types
  - Bit-true types
  - Fixed-point types
  - RTL: ports, signals, clocks
- Modules
- Processes
  - Method processes
  - Thread processes
  - Clocked Thread processes
- `http://www.systemc.org`

```
SC_MODULE( ones_counter ) {                                  1
   enum State  S0, S1 ;                                      2
   sc_in<bool>        IB;                                    3
   sc_out<int>        OB;                                    4
   sc_in_clk          CLK;                                   5
   sc_signal<int>     C, N;                                  6
   sc_signal<State>   state;                                 7
   SC_CTOR( ones_counter ) {                                 8
      SC_CTHREAD( entry, CLK.pos() );                        9
      state = S0;                                           10
      }                                                     11
   void entry( void ) {                                     12
      switch( state ) {                                     13
      case S0 : N = 0; C = IB; // reset                     14
         state = S1;                                        15
      case S1 : if( C == 0 )   // hold                      16
            C = IB, OB = N;                                 17
         else   // inc                                     18
            C = IB, OB = N;                                 19
          state = S1;                                       20
         }                                                  21
      wait();                                               22
   } };                                                     23
```


Semantics Extension: Pros and Cons

- Pros: Expressive power of C++:
  - Simulator comes for “free”
  - Ease of extension
  - Leverage of C/C++ legacy design
  - SystemC
    - Strong support from EDA leader
    - A fast growing user community
- Cons: Expressive power of C++:
  - Synthesis tool does not come for free
  - Another synthesis subset?
    - Inheritance?
    - Multiple-dispatch?
    - Runtime typing?
    - Exception handling?
  - Another synthesis superset?
    - Have to define semantics for design patterns/style
    - Ease of extension: standardization efforts

Syntax Extension Example: SpecC (Version 2.0)

- SpecC Technology Open Consortium
- A rich set of data types
  - Bit true data types
  - RTL: events, buffered variables, signals
- Statements
  - RTL: fsmd
  - Concurrency: par, pipe
  - State transition: fsm
  - Communication: channel
  - Exception and interrupt
- `http://www.specc.org`

```
behavior ones_counter(                          1
  signal in bool CLK,                           2
  signal in bool IB, signal out int OB          3
  ) {                                           4
  buffered[CLK] int  C, N;                      5
                                                6
  void main( void ) {                           7
    fsmd( clk ) {                               8
    S0 :                                        9
      N = 0; C = IB; // reset                   10
      goto S1;                                  11
    S1 :                                        12
      if( C == 0 )   // hold                    13
        C = IB, OB = N;                         14
      else   // inc                            15
        C = IB, OB = N;                         16
      goto S1;                                  17
    }                                           18
  };                                            19
```


Syntax Extension Example: MetaRTL (DATE'00)

- A meta syntax for extension
- An object-oriented, polymorphic type system
- Fields ↦ Storage
  - wires: `in, out, inout, wire`
  - registers: `latch, reg`
  - memory: normal fields
- Methods ↦ Logic
  - Assignment: predicated connection semantics
  - Method dispatch: protocol inlining
  - Statement: logic
  - State label: state boundary

```
public class ones_counter {                                  1
   in clock;                                                 2
   in bit IB;                                                3
   out int OB;                                               4
   positive reg int C( clk), N( clk );                       5
                                                             6
   always positive main( clk ) {                             7
:     N = 0; C = IB; // reset                                 8
      for( ; ; ) {                                           9
:        if( C == 0 )   // hold                              10
            C = IB, OB = N;                                  11
         else   // inc                                       12
            C = IB, OB = N;                                  13
        }                                                    14
      }                                                      15
   }                                                         16
```


Syntax Extension: Pros and Cons

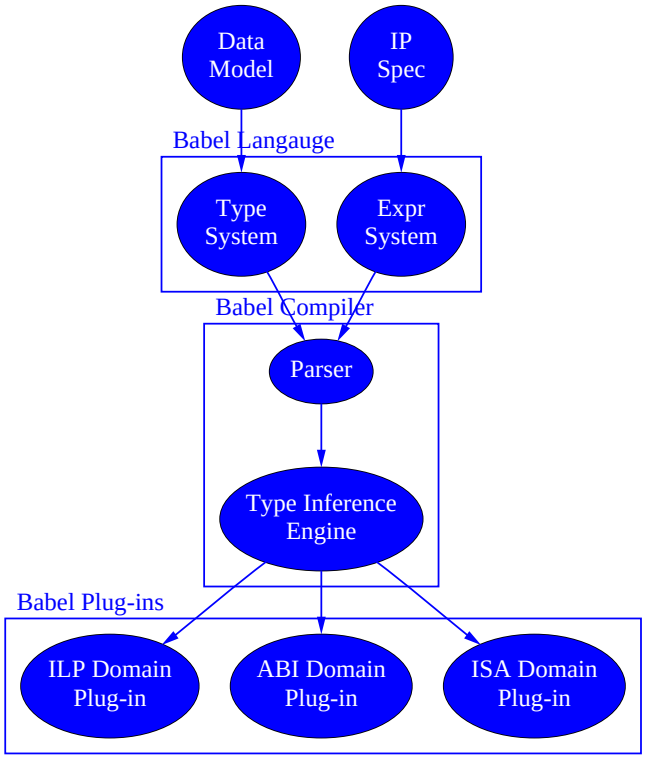
- Pros
  - Transparent simulation process
  - Synthesis semantics can be well defined
  - Leverage of C/C++ legacy design
- Cons
  - Requires a compiler for simulation
  - Requires a separate frontend for debugging
  - Learning curve: new syntax new semantics
  - Not extensible

Extensible Language Example: Rosetta

- System-Level Design Language Standard (under Accellara)
- The language substrate: meta language
  - Type system
  - Component, facet, domain, item
- Domain-specific models
  - Logic and mathematics
  - Axiomatic state based
  - Finite state
  - Infinite state
  - Discrete and continuous time
  - Constraints
  - Mechanical
- Pros
  - Can specify functional model of heterogeneous systems
  - Can specify non-functional models
  - Allows the interaction of models
- Cons
  - Hard to incorporate legacy design
  - Domain completeness
  - Complexity of domain interaction
- `http://www.sldl.org`

Extensible Language Example: Babel

- Developed at University of Toronto
- Focus only on Non-functional model
- A general-purpose database language
- Non-linear: captures arbitrary graph of data
- Extensible: Type system to define domain-specific data model
- Specification are checked using type inference engine
- Domain-specific plug-ins to compile spec into IP



Conclusion

- System-level design calls for the use of “new” languages for system specification
- IP-based design calls for the use of “new” languages for IP component specification
- Languages need to be evaluated under the context of models, infrastructure, tools and methodology
- Suggestion:
  - Be patient: the new system-level languages will improve
  - Keep an open mind: research community will continue to contribute