Specification and Verification of VHDL Designs

Damir Jamsek
Odyssey Research Associates
May 11, 1995

Hardware Verification at ORA

- ❒ Our goals for the overall effort include
  - ❍ specifying behaviors of system components
  - ❍ analyzing behaviors of entire systems from system component descriptions
  - ❍ verifying hardware designs of system components
  - ❍ Integrating our tools with CAD systems

Two projects are currently funded

- ❍ VHDLwise
  - A system for automatic semantic analysis of VHDL designs
  - Phase II SBIR from NASA Langley
- ❍ Larch/VHDL System
  - A system for formal specification and verification of VHDL designs
  - Phase II SBIR out of Rome Labs

Larch/VHDL System

- ❒ Initial project kickoff meeting was held September 27, 1994
- ❒ The goal is to develop formal semantics for VHDL suitable for specification and verification techniques
- ❒ We are using examples to drive the system implementation
- ❒ As more pieces of the language are implemented, larger and more complex examples are being verified.
- ❒ This allows concurrent development of examples that exercise the technology
- ❒ The goal is support for all of VHDL

VHDLwise

- ❒ This effort got under way May 1995
- ❒ A 2 year effort to develop tools for the hardware engineer
- ❒ VHDLwise will perform semantic analysis of VHDL designs automatically (prior to simulation)
- ❒ The semantic analysis will be based on the formal definition of VHDL semantics developed by ORA
- ❒ The analysis will have several goals
  - ❍ Guide designers in realizing correct designs
  - ❍ Aid simulators in speeding up simulations
  - ❍ Aid automatic synthesis routines in creating better hardware

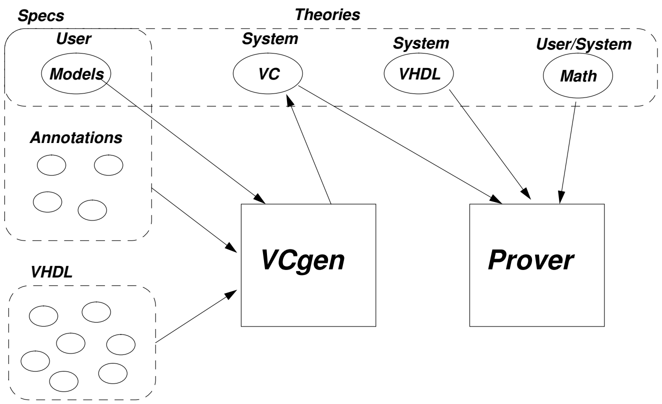
VHDLWise Development

- Phase I developed a basis for semantic analysis of VHDL designs prior to simulation
- Algorithms based on formal semantics of VHDL will automatically check for presence or absence of certain kinds of properties in VHDL designs.
- Examples of such properties include
  - Wait loop checker
  - Cancelled Signal Checker
  - UninitializedVariable Checker
- Each kind of check may aid in one of three ways
  - Guide designers in realizing correct designs
  - Aid simulators in speeding up simulations
  - Aid automatic synthesis routines in creating better hardware

Larch/VHDL

- Two-tiered approach to system specification
  - Mathematics
    - Expresses the basic model of the system
    - Allows analysis of system features independent of system implementation
  - Interface to Designs
    - Specifies the relationship between the actual design and the mathematical model
    - Allows analysis of the actual system implementation
- A theorem prover allows formal verification to be done

System Architecture



VHDL language constructs supported so far

- All phase I VHDL constructs are still supported
  - Concurrent signal assignments
  - Structural architectures
- Array types and enumeration types are supported.
- Constrained subtypes are supported.
- The declarations in package standard are supported.
- Generate statements are supported.
- Concurrent signal assignments with (transport) delay are supported.

Examples Verified

- A generic n-bit ripple-carry adder that has:
  - zero delay
  - a structural architecture using generate statement
  - a behavioral architecture using bit-vector operations
- A generic timed, n-bit ripple adder that has:
  - generic parameter, *del*, for gate delay
  - a structural architecture using generate statement
  - a specification that defines maximum delay as a function *(n*del)* of *n* and *del*.

Examples Verified (continued)

- A 2**n-bit tree-structured carry-lookahead adder:
  - zero delay
  - complex structural architecture using generate statements
- A timed version of the carry-lookahead adder:
  - Generic parameter, *del*, for gate delay
  - a structural architecture using timed components
  - a specification that defines maximum delay as a function *((4*log(n) +3)*del)* of *n* and *del*.

Examples Verified (continued)

- An n-bit word by k-bit word zero-delay multiplier
  - recursively generated architecture
- A timed version of the same multiplier
  - a structural architecture using timed components
  - a specification that defines the maximum delay as *(n+k)*del* where *del* is the delay of a basic component

Examples Verified (continued)

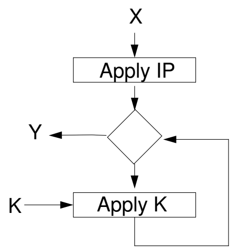
- Circuitry implementing DES algorithm
  - A mathematical specification of the algorithm has been encoded in the Larch notation
  - An architecture for a combinational circuit implementation was created in VHDL
  - Proofs of correctness have been completed.

DES Specification

- The DES standard consists of
  - a collection of tables
  - procedures for applying these tables to input words

K=
23 34 56 10 2 23 56 ...
34 9 8   11 3 12 12 ...
45 64 1 5 7 10 23 43 ...
...

A=e23 e34 e56 e76 ...



- A LARCH specification records both of these in a machine readable form suitable for various kinds of analysis

DES Specification in Larch (Components)

0.1 Des

Des : **trait**

includes (IP_def, K_def, E_def, S_def, P_def, Kluge)
Operation **enumeration of** encode, decode

introduces

K **:** Int, Bit_vector, Operation → Bit_vector
L0, R0 **:** Bit_vector → Bit_vector
E **:** Bit_vector → Bit_vector
A **:** Bit_vector, Bit_vector → Bit_vector
B **:** Bit_vector → Bit_vector
P **:** Bit_vector → Bit_vector
R **:** Bit_vector, Bit_vector → Bit_vector
Ri **:** Bit_vector, Bit_vector, Operation, Int, Bit_vector → Bit_vector
Li **:** Bit_vector → Bit_vector
Liter, Riter **:** Bit_vector, Bit_vector, Int, Operation, Bit_vector → Bit_vector
Y **:** Bit_vector, Bit_vector → Bit_vector
encode, decode **:** Bit_vector, Bit_vector → Bit_vector

Specification of Tables (E transformation)

0.1 E_def

E_def : **trait**

includes (Selection)

introduces

E **:** → Sel
E **:** Bit_vector → Bit_vector

asserts

equations

Econst **:** E = [32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8, 9, 10, 11, 12, 13, 12, 13, 14, 15, 16, 17, 16, 17, 18, 19, 20, 21, 20, 21, 22, 23, 24, 25, 24, 25, 26, 27, 28, 29, 28, 29, 30, 31, 32, 1]
∀ b:Bit_vector

E **:** E(b) = apply(E, b)

Hardware Design implementing the DES

- A VHDL fragment is annotated with a specification

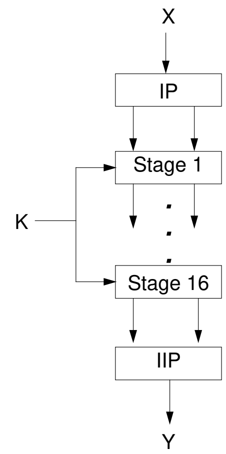
```
use work.DesTypes.all;
entity DEScipher is
  port(x : in Bit64; key : in Bit64; op : in Bit;
       y:out Bit64);
end cipher;
```

entity DEScipher
include DES
guarantees *if op='0' then y = encode(x,key)*
else y = decode(x,key)
end

Different Architectures, Same Function

- Several architectures can be designed to accomplish the same function
- All have the same specification and their functionality is understood via the specification not the implementation
- The simplest architecture is a combinational circuit design that could possibly be generated directly from the specification

Architecture 1 (Combinational)



Architecture 1 (Structure)

```
architecture structural of cipher is
  signal l : Bit32array;
  signal r : Bit32array;
begin
  IP : ip_unit port map(x,l(0),r(0));

  LR : for i in 1 to 16 generate
    LRU : lr_unit
            generic map(i)
            port map(l(i-1),r(i-1),key,op,l(i),r(i));
    end generate;

  IIP : iip_unit port map(l(16),r(16),y);

end structural;
```


Architecture 1 (Stage i)

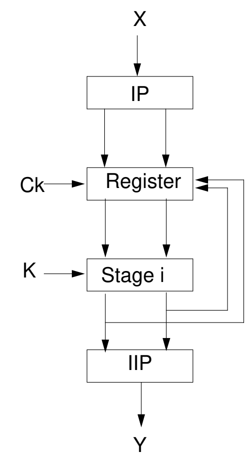
```
architecture structural of lr_unit is
  signal e,ke,kd,a : Bit48;
  signal b,p: Bit32;
begin
  E_unit : eunit     port map(ri,e);
  K_unit_e : kunit     generic map(i)      port map(key,ke);
  K_unit_d : kunit     generic map(17-i)     port map(key,kd);
  with op select
   a <= e xor ke when '0',
          e xor kd when '1';
  B_unit : bunit     port map(a, b);
  P_unit : punit     port map(b, p);
  ro <= p xor li;
  lo <= ri;
end structural;
```

Architecture 1 (Behavior)

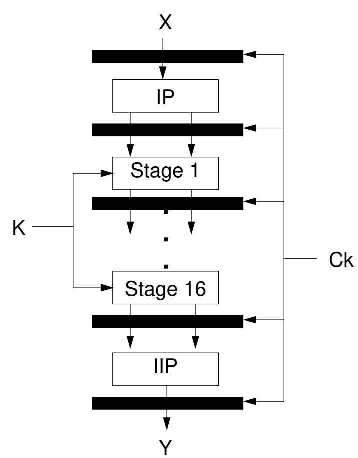
```
architecture behavior of kunit is
begin
 with i select
 k <=
key(22)&key(28)&key(39)&key(54)&key(37)&key(4)&key(47)&key(30)&key(5)&k
ey(53)&key(23)&key(29)&key(61)&key(21)&key(38)&key(63)&key(15)&key(20)&
key(45)&key(14)&key(13)&key(62)&key(55)&key(31)&key(10)&key(51)&key(34)
&key(60)&key(49)&key(17)&key(33)&key(57)&key(2)&key(9)&key(19)&key(42)&
key(3)&key(35)&key(26)&key(25)&key(44)&key(58)&key(59)&key(
1)&key(36)&key(27)&key(18)&key(41) when 1,

key(14)&key(20)&key(31)&key(46)&key(29)&key(63)&key(39)&key(22)&key(28)
&key(45)&key(15)&key(21)&key(53)&key(13)&key(30)&key(55)&key(7)&key(12)
&key(37)&key(6)&key(5)&key(54)&key(47)&key(23)&key(2)&key(43)&key(26)&k
ey(52)&key(41)&key(9)&key(25)&key(49)&key(59)&key(1)&key(11)&key(34)&ke
y(60)&key(27)&key(18)&key(17)&key(36)&key(50)&key(51)&key(58)&key(57)&k
ey(19)&key(10)&key(33) when 2
```

etc. for all 16 cases

Architecture 2 (Clocked)



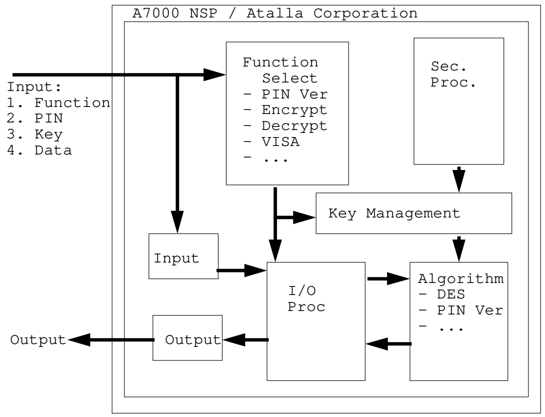
Architecture 3 (Pipelined)



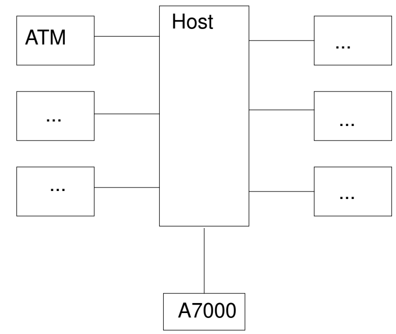
Different Architectures, Same Function

- Architecture 1 is the simplest
- Architecture 2 saves space
- Architecture 3 is fastest
- All have the same specification and their functionality is understood via the specification not the implementation

DES in a Larger Context



A7000 in a larger context



Our Methods Address

- Specification of Components
  - In a notation usable by engineers
- Verification of Implementations
  - In a standard design language
- Composability of Components
  - and analysis of the resulting system

Other examples to be considered

- Some examples we are currently interested in exploring
  - TAXIChip Set AM7968/7969
  - Intel i860 processor
  - VME Bus specification