Software oriented specification languages

- ANSI-C
- SystemC
- Other SW language-based

SystemC

Advantages
- Support from big players
  - Synopsys, Cadence, ARM, Red Hat, Ericsson, Fujitsu, Infineon Technologies AG, Sony Corp., STMicroelectronics, and Texas Instruments
- Familiar for SW engineers

Disadvantages
- Extension of SW language
- Not designed for HW from the start
- Compiler available for limited number of SW processors
  - New

Graph-based specification languages

- Dataflow graph (DFG)
- Synchronous dataflow graph (SDFG)
- Control flow graph (CFG)
- Control dataflow graph (CDFG)
- Finite state machine (FSM)
- Petri net
- Periodic vs. aperiodic
- Real-time vs. best effort
- Discrete vs. continuous timing
- Example from research

Other SW language-based

- Numerous competitors
- Numerous languages
- ANSI-C, C++, and Java are most popular starting points
- In the end, few can survive
- SystemC has broad support

Dataflow graph (DFG)

- Nodes are tasks
- Edges are data dependencies
- Edges have communication quantities
- Used for digital signal processing (DSP)
- Often acyclic when real-time
- Can be cyclic when best-effort
System specification languages
- Abstract data types
- System-level modeling supported
- Better support for test harness design

Disadvantages
- Requires extensions to easily operate at the gate-level
- Difficult to learn
- Slow to code

Verilog
- Easy to learn
- Easy for small designs

Disadvantages
- Not designed to handle large designs
- Not designed for system-level

Advantages
- Supports abstract data types
- System-level modeling supported
- Better support for test harness design

Disadvantages
- Requires extensions to easily operate at the gate-level
- Difficult to learn
- Slow to code

March 1995, Synopsys Users Group meeting
Create a gate netlist for the fastest fully synchronous loadable 9-bit increment-by-3 decrement-by-5 up/down counter that generated even parity, carry, and borrow
5 / 9 Verilog users completed
0 / 5 VHDL users completed
Does this mean that Verilog is better?
Maybe, but maybe it only means that Verilog is easier to use for simple designs. VHDL has better system-level support.
Active HDL debate

- Synopsys CEO pushes System Verilog
  - No new VHDL project starts
- However, many FPGA designers prefer VHDL
- Many places replacing ASICs with FPGAs
- A lot of controversy recently
  - End result unknown

We’ll be introducing VHDL
- This will be helpful for later courses
- This course will only introduce the language
- If you know VHDL and C, learning Verilog will be easy
- Still has better support for system-level design
- Learn VHDL now but realize that you will probably need to know more than one system design language in your career, e.g., System Verilog, SystemC, or both

No single representation has been decided upon
- Software-based representations becoming more popular
- System-level representations will become more important
- Substantial recent changes in the VHDL/Verilog argument

This is an overview and introduction only!
- You may need to use reference material occasionally

VHDL roots
- Very High Speed Integrated Circuits (VHSIC)
- VHSIC Hardware Description Language (VHDL)
  - Model digital systems
  - Simulate the modeled systems
  - Specify designs to CAD tools for synthesis
- VHDL provides a blackboard for designing digital systems
- An initial design is progressively expanded and refined

System-level representations summary
- No single representation has been decided upon
- Software-based representations becoming more popular
- System-level representations will become more important
- Substantial recent changes in the VHDL/Verilog argument

Modeling
- VHDL designed to model any digital circuit that processes or stores information
- Model represents relevant information, omits irrelevant detail
- Should support
  - Specification of requirements
  - Simulation
  - Formal verification
  - Synthesis

VHDL capable of functional and structural specification
- Functional: What happens
- Structural: How components are connected together
- Supports different levels, from algorithmic to gate

Example functional specification

```vhdl
entity XOR2_OP is
  -- 10 ports
  port (A, B : in bit;
        Z : out bit);
endXOR2_OP;

-- Body
architecture EX_DISJUNCTION of XOR2_OP is
begin
  Z <= A xor B;
end EX_DISJUNCTION;
```

Functional and structural specification
A VHDL entity consists of two parts:

- Interface denoted by keyword `entity`
  - Describes external view
- Body denoted by keyword `architecture`
  - Describes implementation

**Interface**

- `entity [identifier] is`
- `port ([name]: in/out/inout bit/[type]);`
- `end [identifier];`

**Body**

- `architecture [identifier] of [interface identifier] is`
- `begin`
- `[code]`
- `end [identifier];`

**Data types**

- The type of a data object defines the set of values that object can assume and set of operations on those values
- VHDL is strongly typed
  - Operands not implicitly converted
- Four classes of objects
  - Constants
  - Variables
  - Signals
  - Files

**Constants**

- The value of a constant cannot be changed
- `constant [identifier] : [type] (:= [expression]);`
- Examples
  - `constant number_of_bytes : integer := 4;`
  - `constant prop_delay : time := 3ns;`
  - `constant e : real := 2.2172;`

**Variable declaration**

- The value of a variable can be changed
- `variable [identifier] [type] (:= [expression]);`
- Examples
  - `variable index: integer := 0;`
  - `variable sum, average, largest : real;`
  - `variable start, finish : time := 0 ns;`

**Variable assignment**

- Once a variable is declared, its value can be modified by an assignment statement
- `label (:) [name] := [expression];`
- Examples
  - `program_counter := 0;`
  - `index := index + 1;`
  - Variable assignment different from signal assignment
  - A variable assignment immediately overrides variable with new value
  - A signal assignment schedules new value at later time
### Scalar types

- Variable can only assign values of nominated type
- Default types: integer, real, character, boolean, bit
- User defined types: type small_int is range 0 to 255;
- Enumerated types: type logic_level is (unknown, low, driven, high);

### Sub-types

- A type defines a set of values
- Sub-type is a restricted set of values from a base type
  - subtype [identifier] is [name] range [simple expression] to [downto simple expression];

### Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
<th>Operand types</th>
</tr>
</thead>
<tbody>
<tr>
<td>+, -, *, /, mod, rem</td>
<td>add, mult, absolute value</td>
<td>integer, real, logical ops, remainder, add, subtract, equal, greater</td>
</tr>
<tr>
<td>not</td>
<td>logical ops</td>
<td>bit, boolean, 1-D array of bit/boolean, integer, real</td>
</tr>
</tbody>
</table>

### VHDL modeling concepts

- Meaning is heavily based on simulation
- A design is described as a set of interconnected modules
- A module could be another design (component) or could be described as a sequential program (process)

### Process statements

```vhdl
(process label): process
  -- declarative part declares functions, procedures, -- types, constants, variables, etc.
  begin
    -- Statement part
    sequential statement;
    sequential statement;
    -- E.g., Wait for 1 ms; or wait on ALARM_A;
    wait statement;
    sequential statement;
    -- wait statement;
    end process;
```

### Variable and sequential signal assignment

- Variable assignment
  - New values take effect immediately after execution
    - variable LOGIC_A, LOGIC_B : BIT;
      LOGIC_A := '1';
      LOGIC_B := LOGIC_A;
- Signal assignment
  - New values take effect after some delay (delta if not specified)
    - signal LOGIC_A : BIT;
      LOGIC_A <= '0';
      LOGIC_A <= '0' after 1 sec;
      LOGIC_A <= '0' after 1 sec, '1' after 3.5 sec;
Signal declaration and assignment

- Signal declaration: Describes internal signal
  - signal [identifier] : [type] [ := expression]
  - Example: signal a, b : bit;
- Signal Assignment: name <= value_expression [ after time_expression];
  - Example: y <= not a and b after 5 ns;
  - This specifies that signal y is to take on a new value at a time 5 ns later statement execution.
- Difference from variable assignment, which only assigns some values to a variable

Transport delay model

- Reflects inertia of physical systems
- Glitches of very small duration not reflected in outputs
  - Logic gates exhibit low-pass filtering
  - SIG_OUT <= not SIG_IN after 7 ns — implicit
  - SIG_OUT <= inertial (not SIG_IN after 7 ns)

If statement

```vhdl
if [boolean expression] then
    [sequential statement]
elsif [boolean expression] then
    [sequential statement]
else
    [sequential statement]
end if;
```

Example of an ALU operation

```vhdl```
case func is
when pass1 =>
    result := operand1;
when pass2 =>
    result := operand2;
when add =>
    result := operand1 + operand2;
when subtract =>
    result := operand1 - operand2;
end case;
```vhdl```

While loop

```vhdl```
while condition loop
    [sequential statements]
end loop;
```vhdl```

Wait statement

- A wait statement specifies how a process responds to changes in signal values.
  - wait on [signal name]
  - wait until [boolean expression]
  - wait for [time expression]
half_add: process is
begin
  sum <= a xor b after T_pd;
  carry <= a and b after T_pd;
  wait on a, b;
end process;

half_add: process (a, b) is
begin
  sum <= a xor b after T_pd;
  carry <= a and b after T_pd;
end process;

clock_gen: process (clk) is
begin
  if clk = '0' then
    clk <= '1' after T_pw, '0' after 2*T_pw;
  endif;
end process clock_gen;

mux: process (a, b, sel) is
begin
  case sel is
    when '0' =>
      z <= a after prop_delay;
    when '1' =>
      z <= b after prop_delay;
  end process mux;

-- Interface
entity XOR2_OP is
  -- IO
  port (a, b: in bit; z: out bit);
end XOR2_OP;

-- Body
architecture EX_DISJUNCTION of XOR2_OP is
begin
  z <= a xor b;
end EX_DISJUNCTION;

entity XOR3_OP is
  port (a, b, c: in bit; z: out bit);
end XOR3_OP;

architecture DISJ_STRUCT of XOR3_OP is
component XOR2_OP
  port (a, b: in bit; z: out bit);
end component;

signal a_int: bit;
begin
  x1: XOR2_OP port map (a, b, a_int);
  x2: XOR2_OP port map (c, a_int, z);
end DISJ_STRUCT;
Test bench for XOR2

```vhdl
entity test_bench is
end;

architecture test1 of test_bench is
signal a, b, z : BIT := '0';
component XOR2_OP
port (a, b: in BIT; z : out BIT);
end component;
for U1: XOR2_OP use
entity work.XOR2_OP(EX_DISJUNCTION);
begin
U1:XOR2_OP port map (a, b, z);
end test1;
```

Test bench for XOR3

```vhdl
architecture test2 of test_bench is
signal a, b, c, z: BIT := '0';
component XOR3_OP
port (a, b, c: in BIT; z : out BIT);
end component;
for U1: XOR3_OP use
entity work.XOR3_OP(DISJ_STRUCT);
begin
U1:XOR3_OP port map (a, b, c, z);
end test2;
```

Introduction to VHDL sequential system design

- Fundamental meaning of state variables
- They are not remembering something specific about the inputs
- Every state transition is a function of the current state and input only
- However, multiple cycles of memory are possible because the current state is a function of the state before it
- When designing an FSM, consider the meaning of each state
- Example: Design a recognizer for any sequence that ends with 01 and observed 1101 at any time.
- Use of asynchronous reset
- Multiple output sequence detector
- Multi-output pattern recognizers
- Laboratory four walk-through
- VHDL examples
Use VHDL to specify and synthesize a FSM

Design a pattern recognizer FSM

Specify it in VHDL

Simulate it with Mentor Graphics ModelSim

Synthesize it with Synopsys Design Compiler

If the last two inputs were 00, \( G \) is high

If the last three inputs were 100, \( H \) is high

entity RECOG is
  port (
    clk, a, reset: in bit;
    h: out bit
  );
end RECOG;

architecture STATE_MACHINE of RECOG is
  type state_type is (s0, s1, s2, s3);
  signal ps, ns : state_type;
begin
  STATE: process (reset, clk)
  begin
    if (reset = '1') then
      ps <= s0;
    elsif (clk'event and clk = '1') then
      ps <= ns;
    end if;
  end process STATE;

  NEW_STATE: process (ps, a)
  begin
    case ps is
      when s0 =>
        case a is
          when '0' => ns <= s1;
          when '1' => ns <= s0;
        end case;
      when s1 =>
        case a is
          when '0' => ns <= s2;
          when '1' => ns <= s0;
        end case;
      when s2 =>
        case a is
          when '0' => ns <= s2;
          when '1' => ns <= s3;
        end case;
      when s3 =>
        case a is
          when '0' => ns <= s1;
          when '1' => ns <= s0;
        end case;
    end case;
  end process NEW_STATE;

  OUTPUT: process (ps)
  begin
    case ps is
      when s0 => h <= '0';
      when s1 => h <= '0';
      when s2 => h <= '0';
      when s3 => h <= '1';
    end case;
  end process OUTPUT;
end STATE_MACHINE;

architecture test_recog of test_bench is
  signal clk, input, reset, output : bit := '0';
begin
  component RECOG
    port (clk, a, reset: in bit; h : out bit);
  end component;

  for U1: RECOG use entity work.RECOG(STATE_MACHINE);
begin
  U1: RECOG port map (clk, input, reset, output);
CLK_CHANGE: process
begin
  loop
    clk <= '1';
    wait for 5 ns;
    clk <= '0';
    wait for 5 ns;
  end loop;
end process CLK_CHANGE;

RESET_CHANGE: process
begin
  reset <= '1' after 0 ns,
  '0' after 5 ns;
  wait;
end process RESET_CHANGE;

INPUT_CHANGE: process
begin
  input <=
  '0' after 5 ns,
  '1' after 15 ns,
  '0' after 25 ns,
  '1' after 45 ns,
  '0' after 55 ns,
  '1' after 75 ns,
  '0' after 85 ns,
  '0' after 105 ns,
  '1' after 105 ns;
  wait;
end process INPUT_CHANGE;

end test_recog;

architecture primitive of and_or_inv is
  signal and_a, and_b, or_a_b : bit;
begin
  and_gate_a: process (a1,a2) is
  begin
    and_a <= a1 and a2;
  end process and_gate_a;
  and_gate_b : process (b1,b2) is
  begin
    and_b <= b1 and b2;
  end process;
end primitive;
Behavioral modeling example (cont.)

or_gate: process (and_a, and_b) is
begin
    or_a_b <= and_a or and_b;
end process or_gate;

inv: process (or_a_b) is
begin
    y <= not or_a_b;
end process inv;

end architecture primitive;

VHDL synthesis quirks

- Given a statement
  \[ y = a + b + c + d; \]
- Synthesis tool will create a tree of adders by adding \( a + b \), then adding to \( c \), and then to \( d \);
- Instead if specified as
  \[ y = (a + b) + (c + d); \]
- The synthesis tool will be forced to synthesize a tree of depth 2 by adding \((a+b)\), and \((c+d)\) in parallel, then adding results together.

- Another possible mistake
  \[ y = a \text{ or } b \text{ or } c \text{ and } d; \]
  Instead write as
  \[ y = (a \text{ or } b) \text{ or } (c \text{ and } d); \]

Reading assignment

Chapter 11
• More on VHDL
• Introduction to asynchronous FSM design