Alternate Digital Design Languages

By

Jim Brakefield

Brakefield Research

Presentation to

Austin IEEE Consultants Network

January 8, 2004

Glossary

SOC	System On a Chip: typically one or more uP (microprocessor), analog & digital circuitry
ASIC	Application Specific Integrated Circuit
FPGA	Field Programmable Gate Array
Structured ASIC	or Modular Array ASIC, base device pre-designed with gates & RAM User specifies only part of the wiring (only one or two design specific fabrication masks)
Gate Array	Chip with gates and memory specified, user specifies all wiring.
EDA	Electronic Design Automation
CAD	Computer Aided Design
RTL	Register Transfer Logic, designer specs registers and logic between regs
RTOS	Real-Time Operating System

Glossary cont'd

Matlab www.mathworks.com Supports Matlab to VHDL/Verilog/C/DSP asm.
"language of technical computing" Used for signal processing algorithm devel.

Confluence www.launchbird.com The language has ability to generate VHDL, Verilog, XML, C/C++, Java, Python, & Promela (SPIN model checker) models

Targets both digital hardware design and embedded systems.

Is the two year's work of a single individual: Tom Hawkins. First license is free.

VHDL VHSIC Hardware Description Language

IEEE Standard 1076: VHDL-85, VHDL-93, VHDL-2001

Analog extension: VHDL-AMS

Attributes may be assigned to any name

Verilog Started as a simulation language in 1984 by Gateway Design Automation

IEEE-1364: Verilog-1995, Verilog-2001

Attributes in Verilog-1995 using textual proximity

Glossary cont'd

SystemVerilog Additions to Verilog-2001 for "large gate count, IP-based, bus-intensive chips", "powerful links to the system level design flow" , vendor (Cadence, Synopsys, Mentor) support in 2004

SystemC www.systemc.org Design models via C++ library of hardware primitives

Handel-C www.colexica.com Superset of C, C to VHDL/Verilog converter

EDIF Electronic Design Interchange Format

VCD Value Change Dump format: Simulation log file

VHSIC Very High Speed Integrated Circuits

VHDL/Verilog Web Sites:

www.deeps.org/verilog Lots of folksy info on Verilog

www.accellera.org Unified VHDL/Verilog standards group

vhdl.org, verilog.org, eda.org & systemverilog.org all link to accellera.org

Has a list of IEEE-1076 and IEEE-1364 activities

Confluence 101

www.launchbird.com Is the Web Site
- Offers Confluence Training and Design Services

Web site has edit & see compiler

Eight Sample Designs at www.opencores.org:
- FIR Filter, Memory Interleaver, Floating-Point Multiplier, FFT, State Space Processor, LDPC Decoder, Reconfigurable Computing Array

Compiler versions for Linux x86, Windows via Cygwin, SPARC/Solaris, Ocaml on Unix, Mac and Windows

Is a *Work in Progress*:
- Keywords change, new data types get introduced
- Model generators for more languages get added (latest is XML)

Confluence 101 cont'd

Articles on Confluence:

"New language makes waves":
http://www.eedesign.com/columns/max_bytes/OEG20030529S0070

"Declarative programming language simplifies hardware design":
http://www.eedesign.com/features/exclusive/OEG20030918S0045

"Engineer creates HDL generation language":
http://www.eedesign.com/news/OEG20030919S0013

"Design language links to open-source model checker":
http://www.eedesign.com/news/OEG20031016S0068

Confluence 101 cont'd

Statements:

Connection	A <- B X <- Y -> Z
Instantiation	{ Component_name Port1_name Port2_name … Portn_name }
Local Namespace	**local** NameA NameB **is** statements **end**
Conditional if-else	**if** PredicateA statemts **ef** PredicateB statemts **else** statemts **end**
Component Definition	**component** Add +A +B –X **is** X <- A + B **end**

Data Types:

Boolean

Arbitrary precision integer, uses the C operator set

Double precision float

Record with named fields, fields accessed via ".name" or ".index"

List: now an extension of record, uses square brackets

String: implemented as a list of ASCII coded integers

Vector: VHDL & Verilog "signals"

Component: "subroutine" definitions

System: component invocations use curly brackets

Confluence Data Types cont'd

Vector:

- Used to represent "signals"
- Translate into fixed precision integers, e.g. a bit string
- Only have values in the VHDL/Verilog/C/Python simulation
  - In Confluence vector is a type with no "value"

Component: Component name and a list of named ports (e.g. parameters)

- Component definition "equivalent" to subroutine definition
- Prefix names (now optional, treated as comments)
  - Inputs: "+" Outputs: "-" Bidirectionals: "*"
- "$" can be used as a place holder
  - To select a result from a component instantiation
  - To route a symbol to a component port

Confluence Data Types cont'd

System: Is the result of a component instantiation

Result can be of any type:

Component instantiation can return a component definition

Is a standard feature of functional programming languages

Can use "_" as a port place holder

System Components

{IsSystem +Val –Bool}

{SystemToString +Sys –Str}

{Clock +ClockName +Sys}

{Reset +ResetVector +Sys}

{Enable +EnableVector +Sys}

Confluence 101 cont'd

Integers and Floats have the usual "C" operators

Vector Operators similar to those of VHDL/Verilog

And are enclosed in single quotes

Examples: A '+' B, A '<<' Int, A 'then' B 'else' C

Confluence Semantics & Syntax

Single Assignment semantics

All symbols are "defined" just once

Evaluation order of single assignments determined by operand availability

No looping constructs, must use recursion

Unification used to connect names and values

Confluence Advantages

Generates both C/Python and VHDL/Verilog models

- A single "**golden**" reference design that creates both simulation and implementation versions
- Generates test bench templates

Compact: 3X less code than Verilog

- For each clock domain, clock name, reset and enable applied to all registers
- Components for Registers, RAM, ROM & System ports
- Functional programming offers higher levels of abstraction and usage
  - Much more than a macro template generator
  - Lots of freedom in design layout and organization

Easy to learn: Typically productive in first day

Low Cost: $240/month for additional licenses

A Vision for Embedded Systems

(IMHO)

A real-time routine is just a VHDL/Verilog design component with a relatively long clock period constraint.

Interrupt handlers are dedicated hardware or software running in a dedicated controller.

A RTOS is unnecessary (for interrupt dispatch, message forwarding) !!!

Inter-processor communication via hardware or software FIFOs

On The Embedded Horizon

Real-time Validation via Simulation/Formal-Verification and Timing Analysis

From months to hours !!!

Implementation of this vision is via modified ASIC/FPGA tool flow.

Automatic code movement between hardware and software possible.

Confluence is one vehicle to achieve this vision.

Part of the larger vision of a practical methodology for parallel uP's + programmable logic SOC.

Sample Confluence Designs

OpenCores Project	Source Name	Description
FIR Filter	fir.cf	Finite impules response filter with variable taps and variable precision.
Memory Interleaver	common.cf	Memory interleaver with variable address and data widths.
Floating Point Multiplier	fp_mul.cf	Variable precision floating point multiplier.
FFT	fft.cf	Continuous throghput multi-butterfly FFT.
Cordic	cordic.cf	Pipelined cordic for calculating trigonometric functions. Separate cores for vectoring and rotation modes.
State Space Processor	ssp.cf	Ultra-light processor for calculating multi-variable linear functions common in DSP and control applications.
LDPC Decoder	ldpc.cf	Low-density parity-check (LDPC) hard-decision decoder. Implements Gallager's algorithm A.
Reconfigurable Computing Array	rca.cf	Fine-grained dynamic reconfigurable computing array, similar to an FPGA.

Example Confluence Design

```
(* Basic code generation template. Defines an up/down counter as an example. *)
(* Local variables. *) with UpDownCounter is
(* Define the UpDownCounter component. *)
component UpDownCounter
    (* Component ports. '+' is input, '-' is output. *)
    +CounterWidth (* Integer configuration parameter. *)
    +CountUp (* 1-bit signal. 1 counts up, 0 counts down. *)
    -CountValue (* The output counter signal. *)

with (* Component implementation. *)
    (* Local variables. *) ValueOne ValuePlusOne ValueMinusOne ValueSelected is
    ValueOne <- {One CounterWidth $}
    ValuePlusOne <- CountValue  '+'  ValueOne
    ValueMinusOne <- CountValue  '-'  ValueOne
    ValueSelected <- {Mux CountUp [ValueMinusOne ValuePlusOne] $}
    CountValue <- {VectorReg CounterWidth ValueSelected $}
end
(* Instantiate an 8-bit UpDownCounter and connect inputs and outputs. *)
{UpDownCounter 8 {VectorInput "count_up" 1 1 $} {VectorOutput "count_value" 2 $}}

(* Set code generation constraints. *)
    {Set "FileName" "up_down_counter"} {Set "BuildName" "UpDownCounter"}
    {Set "Header" "A complete code gen example of an up/down counter."}
    {Set "GenVerilog" true} {Set "GenVhdl" true} {Set "GenC" true} {Set "GenPython" true}
```

Example Confluence Design

Generated H file for C output

```
//  Generated by Confluence 0.7.7  --  Launchbird Design Systems, Inc.  --  www.launchbird.com
void UpDownCounter_ports(unsigned char* port_count_up_i, unsigned char* port_count_value_o);
void UpDownCounter_init();
void UpDownCounter_calc();
void UpDownCounter_cycle_clock();
void UpDownCounter_sim_init(const char* file);
void UpDownCounter_sim_end();
void UpDownCounter_sim_sample();
```

Generated Verilog output

```
//  Generated by Confluence 0.7.7  --  Launchbird Design Systems, Inc.  --  www.launchbird.com
`timescale 1 ns / 1 ns
module UpDownCounter (clock_c, count_up_i, count_value_o);
input  clock_c;  input  count_up_i;  output [7:0] count_value_o;  wire   [7:0] n1;
UpDownCounter_1 s1 (clock_c, count_up_i, n1);
assign count_value_o = n1;                                  endmodule
module UpDownCounter_1 (clock_c, i1, o1);
input  clock_c;  input  i1;  output [7:0] o1;  wire    n1;  wire    n2;  wire    [7:0] n3;
wire   [7:0] n4;  wire   [7:0] n5;  wire    n6;  wire   [7:0] n7;  reg     [7:0] n8;
assign n1 = 1'b1;  assign n2 = 1'b0;  assign n3 = 8'b00000001;  assign n4 = n8 - n3;
assign n5 = n8 + n3;  assign n6 = i1;  assign n7 = n6 ? n5 : n4;  initial n8 = 8'b00000000;
always @ (posedge clock_c)
  if (n2 == 1'b1)      n8 <= 8'b00000000;
  else if (n1 == 1'b1)      n8 <= n7;       assign o1 = n8;     endmodule
```