1995 High Level Synthesis Design Repository

Preeti Ranjan Panda
Nikil Dutt

Technical Report #95-04
April 11, 1995

University of California, Irvine
Irvine, CA 92717
(714) 824-7219

dutt@ics.uci.edu

Abstract

In this report we briefly describe a set of designs that can serve as examples for High Level Synthesis (HLS) systems. The designs vary in complexity from simple behavioral finite state machines to more complex designs such as microprocessors and floating point units. Most of the designs are described in the VHDL language at the behavioral level. The report describes two categories of designs. The first category contains designs that have documentation on the specifications of the designs along with the strategy used to test the individual design models. The second category contains examples used in many HLS papers, but lack comprehensive decumentation and/or test vectors.

Contents

List of Figures

1 Introduction

The effort at creating a repository of High Level Synthesis (HLS) benchmark designs has been under way since the 24th Design Automation Conference in 1987. An informal set of HLS benchmarks was created after the High Level Synthesis Workshop in 1988 and maintained at the SIGDA repository at *mcnc.mcnc.org*. The purpose of maintaining this repository was to serve as a basis of comparison of various approaches to High Level Synthesis and to provide a means for researchers and developers to exercise their synthesis systems on a wide range of digital circuits.

This informal set was consolidated [DuRa92] into nine benchmark designs whose functionality and verification schemes were well documented. These benchmarks, which include some simple controllers, digital filters, a microprocessor slice, and a USART design among others, have been widely used as examples by HLS researchers. In [DuRa92], guidelines for developing and submitting new benchmarks were also formulated so as to make the benchmark collection an ongoing process. With this report we release some of the interesting and important designs that were submitted in accordance with these guidelines, as well as a set of designs that have complete HDL descriptions but lack comprehensive documentation and/or test vectors.

We present a new set of design examples that augment the existing benchmark suite. Most of these are fairly large designs with reasonably complex data and control structures. In most cases, the models are accompanied by test patterns that were used to verify the correctness of the descriptions. The tests are, of course, not exhaustive but are intended to check for typical behaviors. In some cases, boundary conditions have been extensively checked, while in others (like the microprocessor example) a few cases have been tried out for every instruction. We must mention that while the models have all been subjected to simulation checks, they may not all be directly synthesizable, and might require modification when used as an example for a synthesis tool. Some of the constructs used might make sense only for simulation. In such cases, however, the test vector set should be useful in determining the correctness of any modifications to the design descriptions.

2 Overview of the Designs

This section presents a brief overview of the designs. Figures 1 and 2 summarize some of the important aspects related to the functionality and verification of these designs, such as typical control features present, style of description, major data types used and the extent to which the design example has been tested. The number of lines of code (LOC) is mentioned to give a rough idea of the design's size. The LOC includes lines with comments. (The lines of *executable* VHDL code is typically 50 % of the total lines of code. *The LOC figures must be used with caution, for writing styles vary and the sizes of the models are small enough to permit erroneous estimations about design size based on LOC alone.*)

In the following sections, we describe in brief the functionality of each of the design examples and mention the main testing strategy adopted. Some of the models are quite thoroughly tested for errors while others have not been exercised very much.

3 Designs with complete descriptions

This section contains the descriptions of the designs that have proper documentation and test cases.

Design Name	Design Description	Design Level	Description Style	Control features	Data Types	Test Vectors	Lines Of Code
FP_Adder	Floating Point Adder	Algorithmic Behavior	1 VHDL Process	Nested Ifs For Loops Proc/Func	Bit Vector Integer Enum	417 Cycles	640 (VHDL)
FP_Mult	Floating Point Multiplier	Algorithmic Behavior	1 VHDL Process	Nested Ifs For Loops Proc/Func	Bit Vector Integer Enum	169 Cycles	425 (VHDL)
FP_Divider	Floating Point Divider	Algorithmic Behavior	1 VHDL Process. Func in separate package	For Loop Case Stmt	Integer StdLogicVector	10 Cycles	410 (VHDL)
Prawn	CPU 8-bit, 40 instructions	Instruction Set Behavior	1 VHDL Process	Nested Ifs Case Stmt	Bit Vector	1600 Cycles	700 (VHDL)
RT-PC	CPU 8-bit, 119 instructions	Instruction Set Behavior	Multiple Entities. Functions in sep. pack.	For Loop While Loop Case Stmt	Subtypes Bit, int Array Overloaded op	900 Cycles	3000 (VHDL)
Barcode	Barcode Reader	Algorithmic/ High Level FSM	1 VHDL Process	Nested Loops	Subtypes Integer	1 Test Suite	110 (VHDL)
QRS	E.C.G Application Chip	Algorithmic/ High Level FSM	1 VHDL Process	Loops Nested Ifs	Subtypes Integer	4300 Cycles	280 (VHDL)
Adaptive	Adaptive Interpolation Algorithm	Algorithmic Behavior	Set of Silage Functions	Func Calls, Nested Loop	Multi Dimensional Integer Arrays	6 Test Suites	810 (Silage)
Volume	Bladder Volume Computation	FSM with Datapath	Set of Seq/Conc SpecCharts Behaviors	Transition Arcs, For/ While Loops	Bit Vector Integer Array	20 Test Cases	220 (SpecCharts)
Answer	Telephone Answering Machine	FSM with Datapath	Set of Seq/Conc SpecCharts Behaviors	Transition Arcs, For/ While Loops	Integer Bit Vector	23 Test Sequences	640 (SpecCharts)

Figure 1: Features (in brief) of Designs with Complete Information

Design Name	Design Description	Design Level	Description Style	Control features	Data Types	Test Vectors	Lines Of Code
Memory (7 models)	Image Processing Applications	Algorithmic Behavior	1 C function for each example	Nested Loops	2-Dimensional float Arrays	No Test Suite Available	Each ~20 Lines (C)
Filter	"Switchable" 3rd order FIR Filter	Algorithmic Behavior	1 Verilog module	If Stmt	Bit Vector	No Test Suite Available	35 (Verilog)
Period Counter	Period Counter	Algorithmic Behavior	1 Verilog module	While Loop If Stmt	Bit Vector	No Test Suite Available	90 (Verilog)
Beamformer	Filter	Vector Product/ Summation	1 VHDL Process	Nested For Loops (4 levels)	3-dim array of Integer	No Test Suite Available	100 (VHDL)
Jacobian	Robot Motion Computation	Algorithmic Behavior	Set of C Functions	For Loops	Struct Pointers 2-dim array of 'double' trigon. func	No Test Suite Available	450 (C)
FFT	Fast Fourier Transform	Algorithmic Behavior	1 VHDL Process	Nested While Loops	Array of Bit Vector	No Test Suite Available	145 (VHDL)
DHRC	Differential Heat Computation	Algorithmic Behavior	1 VHDL Process	While Loops	Array of Bit Vector	No Test Suite Available	100 (VHDL)

Figure 2: Features (in brief) of Designs with Incomplete Information

3.1 Floating Point Adder

Description

This model performs addition and subtraction of two floating point numbers. The IEEE Floating Point standard is used. The two operands and the result are represented by a sign bit, a 127-biased integer exponent in the range 0..255, and a 23-bit vector mantissa with a *hidden* 1 [HePa90].

The inputs to the model are the two operands, a clock and an opcode which can indicate one of three operations - ADD, SUBTRACT and IDLE. The outputs of the model are the result of addition and status flags indicating the special cases of zero, positive and negative infinities and Not-A-Number. While ADD and SUBTRACT represent the corresponding operations, the IDLE operation maintains the previous result at the output.

Testing Strategy

The test vectors to which this model is subjected include both typical cases and boundary conditions. The boundary conditions are exhaustively tested. The tests include boundary conditions on the operations (such as underflow, overflow, etc.), as well as tests involving boundary conditions on operands such as positive and negative infinity, NAN (Not-A-Number), smallest representable positive and negative numbers, etc.

3.2 Floating Point Multiplier

Description

This model performs multiplication of two floating point numbers. The IEEE Floating Point standard for single precision floating point numbers is used. The two operands and the result are represented by a sign bit, a 127-biased integer exponent in the range 0..255, and a 23-bit vector mantissa with a hidden 1.

The inputs to the model are the two operands, a clock and an opcode which can indicate one of two operations - MULTIPLY and IDLE. The outputs of the model are the product of the two operands and status flags indicating the special cases of zero, positive and negative infinities and Not-A-Number. The MULTIPLY operation results in the output being updated with the product of the operands and the IDLE operation maintains the previous result at the output.

Testing Strategy

The test vectors to which this model is subjected include both typical cases and boundary conditions. The boundary conditions are exhaustively tested. The tests include boundary conditions on the operations (such as underflow, overflow, etc.), as well as tests involving boundary conditions on operands such as positive and negative infinity, NAN (Not-A-Number), smallest representable positive and negative numbers, etc.

3.3 Floating Point Divider

Description

This model performs division of one floating point operand by another. The IEEE Floating Point standard for double precision floating point numbers is used. The two operands and the result are represented by a sign bit, a 10-bit 511-biased integer exponent in the range 0..1023, and a 53-bit vector mantissa with a hidden 1.

The Digit-Recurrence Algorithm [LaEr94] is employed for the division using radix 512. The division essentially consists of six iterations of a recurrence [LaEr94] in which each iteration produces nine bits of the quotient (i.e., one digit in radix 512), most significant digit first. The algorithm is implemented for the case where the divisor is a fraction between 0.5 and 1, while the dividend is a fraction between 0.25 and the divisor. Other cases can be derived by using appropriate scaling factors.

Testing Strategy

Test vectors available for verification of this model were generated for some typical and boundary cases of the divisor and dividend. A C program for generating more test vectors is provided.

3.4 The RT-PC Processor

Description

This is a VHDL description of the IBM RT PC [ThDu94], a RISC processor which uses the ROMP architecture. The description is based on the definition of the architecture from [RTPC85].

The architecture consists of an 8-bit processor which has an 8 bit data-bus and a 24 bit address-bus. (The model actually makes the address space programmable, in order to avoid large memory arrays which would be needed to store the entire range addressable by a 24 bit address.) The system consists of three main-blocks: the *Control* unit, the *Memory* unit and the *RTPC-Model* unit.

The *Control* unit reads the program file and writes it to *Memory*. It then triggers the *RTPC-Model* to start execution and the *RTPC-Model* indicates the completion of execution to the Control unit.

The instructions for the RT-PC include those in the category of memory load/store, address computation, conditional and unconditional branches, traps, moves and inserts, arithmetic operations, logical opeartions, shifts and system control.

Testing Strategy

The *RTPC-Model* block serves as the testbench for exercising the processor and memory blocks. An assembler was written to convert assembly code into code in the format understood by the model. The test suite developed for testing this model includes the exercising of every instruction the model is designed to handle, with two or three different operand sets.

3.5 The *Prawn* Microprocessor

Description

Prawn is based on the *Parwan* RISC processor described in [Navabi93]. The instruction set of [Navabi93] was enhanced to include interrupt handling and conditional branches.

Prawn is an eight-bit microprocessor which has an 8-bit Data Bus and a 12-bit Address Bus for external accesses, the address space being partitioned into sixteen pages of 256 bytes each. It has a limited number of arithmetic and logic instructions, several jump and branch instructions, subroutine call instructions, and interrupt instructions. Some instructions have an addressing mode that provides for direct and indirect addressing. *Prawn* has an accumulator, a reduced ALU, a shifter, program counter, and five flags (interrupt enable, overflow, carry, zero, sign). It uses a designated address of memory as a stack pointer register.

The ports of the model consist of a clock, two interrupt ports (reset and interrupt), an 8-bit data bus, a 12-bit address bus, signals for reading and writing to and from memory and an interrupt acknowledge signal.

Prawn has single byte and double byte instructions. Many of the double byte instructions have two addressing modes: direct and indirect. One of the memory pages is treated as the stack.

Testing Strategy

The test vectors basically ensure that all instructions are executed. Each test vector file consists of two sets of memories and one process for simulation control. The first set of memories, called the *Working Set*, contains the program, data, and working areas for the Prawn CPU. These are executed and used by the *Prawn* CPU during simulation. The second set, called *Expected Set*, has the expected contents for the set of memories when a correct simulation is finished. For every test suite the memory at the end of the testing is compared with the expected values at the memory locations.

3.6 Barcode Reader

Description

This model performs the function of a barcode reader [BhBrDe93]. The algorithm used by the reader essentially consists of reading in the bits that are generated by the scanner (from a single bit input signal) and recording the width of the black and white stripes. Whenever there is a transition from one color of stripe to another (whether black $\rightarrow$ white or vice versa), it is recorded in a single counter. The results are written to a memory location, whose address corresponds to the number of transitions encountered.

Testing Strategy

A few test cases were developed to verify the functionality of the model on some random input Barcode patterns (i.e., color of white and black bit streams.)

3.7 The QRS Chip

Description

This VHDL model describes a chip used for monitoring the heart-rate in ECG applications, called the QRS chip [BhBrDe93] (since it detects some points called Q-R-S points in the ECG data stream).

Three different descriptions of the same chip are given, two algorithmic and one in the form of a state machine, all expressing the same functionality.

This benchmark lacks sufficient documentation. The models all have equivalent functionality but no documentation exists on the algorithm itself.

Testing Strategy

The model was verified with a set of test cases for which a reference expected output set has been provided.

3.8 Adaptive Interpolator

Description

This design (written in the Silage language) implements an adaptive interpolation algorithm for digital audio signals [VeJaVr86]. In brief, this computation-intensive algorithm recovers a signal that contains a burst.

Testing Strategy

Six test suites are available for verifying the functionality of the design. A set of values of the signals, the length of the burst and positions of the burst form each test suite, along with the expected output patterns.

3.9 Volume System

Description

This design (written in SpecCharts [VaNaGa91]) models a *Volume System* for measurement of the volume of a human bladder [VaGoNa94]. The volume system controls a transducer, which is attached to a motor, to scan the related abdominal area along two-dimensional grids. At each scanning point, the transducer sends an ultrasonic wave into the anatomical region to be examined. When the ultrasonic wave strikes tissues of different acoustic impedance, an echo is reflected to the transducer. Two major peaks will be generated by the echoes from the anterior and posterior walls of the bladder. The distance between the anterior and the posterior walls of each section of the bladder is determined from this information and the volume of the bladder is computed.

Testing Strategy

A VHDL testbench is available to test the generated VHDL model on twenty sample inputs.

3.10 Answering Machine

Description

This design (written in SpecCharts) models the controller of a *Telephone Answering Machine*

[GaVaNaGo94]. The features modeled include an outgoing announcement, recording and playback of messages, forwarding and rewinding of the message tape, display of number of messages, etc.

Testing Strategy

A VHDL testbench is available to test the generated VHDL model on about twenty typical input sequences.

4 Designs with incomplete descriptions

This section contains the descriptions of the designs used in several publications, for which no test suites exist and for which documentation is also incomplete. References have been provided, however, indicating the source of these designs.

4.1 Image Processing Applications

Description

These examples implement (in the C language) a set of algorithms related to image processing [PrTeVeFl92] that were used to validate some algorithms on memory synthesis [KoNiDu94]. These designs are characterized by their memory-intensive behavior.

SOR – This is a Successive Over-Relaxation algorithm used in evaluating partial differential equations.

Wavelet This algorithm, used in image compression, implements the Debaucles 4-Coefficient Wavelet filter.

GSR – This is an algorithm for Red-black Gauss-Seidel relaxation method.

Linear – This implements a general linear recurrence solver.

LowPass – This code applies a low-pass filter to an image. Low-pass filters accentuate low frequencies in an image – that is, the resulting image has lower changes between neighboring color values.

Laplace – This code implements a Laplace algorithm to perform edge enhancement of northerly directional edges in an image.

Compress – This code implements an image compression scheme by estimating the current cell based on the neighbors' values. It then stores the difference between the prediction and the actual value.

Testing Strategy

Test cases for verifying the functionality do not exist for these models.

4.2 Switchable Filter

Description

This model describes (in Verilog language) an *Interpolating Switchable 3rd Order FIR Filter* [Ug95]. It is *switchable* because it samples its input either every two clock cycles or four clock cycles depending on the *switch* input. It is *interpolating* because although it only generates a true output value based on the third FIR filter equation at the same frequency as it samples its input, it still updates the output by linear interpolation in intermediate clock cycles.

Testing Strategy

No test cases are available.

4.3 Period Counter

Description

This model describes (in Verilog language) a *Period Counter* [Ug95]. It counts the length of a complete cycle of an input signal in terms of the number of clock cycles it takes. The circuit is asleep as long as *reset* is low. When reset is released, it computes the period by appropriately waiting for the positive and negative edges of the signal.

Testing Strategy

No test cases are available.

4.4 FFT

Description

This design, written in VHDL, encodes the Fast Fourier Transform (FFT) algorithm for converting information from the time domain to the frequency domain [CaSv93]. The design is a model of an N-point FFT, consisting of N inputs, N outputs and log N stages of computation, each stage requiring N/2 *butterflies* where a butterfly consists of one addition, one subtraction and one multiplication. Two consecutive stages are connected to each other via a shuffle network.

Testing Strategy

No test suites are available for this design.

4.5 Differential Heat Release Computation

Description

This design, written in VHDL, encodes a Differential Heat Release Computation algorithm that models the heat release within a combustion engine [CaSv93]. This model has been employed in memory mapping experiments.

Testing Strategy

No test suites are available for this design.

4.6 Beamformer

Description

This design, written in VHDL, describes the behavior of a Beamformer system, an example of a typical DSP system [BaGa93]. The beamforming operation involves the temporal alignment and summation of digitized signals from an N-element antenna array. The N antennas are spread over a distance of the order of a few kilometers, hence the samples arriving at the elements corresponding to different times (having travelled different distances.) Before these samples are summed, as the beamformer operation requires, it is essential to interpolate the samples such that they all correspond to the same time instant. A *beam* is formed by summing all the time-aligned signals.

Testing Strategy

No test suites are available for this design.

4.7 Jacobian

Description

This design, written in C language, evaluates the Jacobian of an open kinematic chain, whose representation is based on the *product-of-exponentials* formula [PaMu93].

The Jacobian of a robot is the linear transformation relating joint rates to end-effector rates. The input to the algorithm consists of a set of n joints of the robot and a set of n 4×4 matrices characterizing the joints. The result of the Jacobian computation is a set of n 4×4 matrices.

Testing Strategy

No test suites are available for this design.

5 Summary

In this report we described in brief the functionality and testing strategy of several new High Level Synthesis designs. All of these designs are available from the design repository at U.C Irvine (anonymous ftp; site: ics.uci.edu; location: pub/HLSynth95).

We welcome any feedback on the design examples and their accompanying documentation. We also welcome the submission of more designs for future inclusion - preferably, those whose functionality does not overlap with that of the existing designs. This variety in the design examples is important and is in accordance with our goal of making realistic design examples available to the HLS community as well as to serve as a reliable basis for stimulating new research, as well as for meaningful comparison of HLS systems and algorithms.

6 Acknowledgments

We would to thank the following people who have contributed towards the development of the designs: Jesse Pan and Bob McIlhenny (FPAdder and FPMult), Alberto Nannarelli (FPDivide), Alfred Thordarson (RT-PC), Tadatoshi Ishii (Prawn), Franc Brglez (Barcode and QRS), David Kolson (Image Processing Applications), Lode Nachtergaele (Adaptive Interpolator), Fatih Ugurdag (Filter and Period Counter), Jie Gong (Answering Machine and Volume System) and Smita Bakshi (Beamformer, FFT, DHRC and Jacobian.)

We also thank Manu Gulati for his useful comments and suggested improvements on the Barcode and QRS designs.

This work was supported in part by SRC Grant 94-DJ-146. We are grateful for their support.

7 References

[Arms89] J. Armstrong, "Chip-level Modeling with VHDL," Prentice Hall 1989.

[BaGa93] S. Bakshi and D. D. Gajski, "Design Space Exploration for The Beamformer System," Technical Report 93-34, University of California, Irvine, 1993.

[BhBrDe93] S. Bhattacharya, F. Brglez and S. Dey, "Transformations and Resynthesis for Testability of RT-Level Control-Data Path Specifications," IEEE Transactions on VLSI Systems, September 1993.

[CaSv93] F. Catthoor and L. Svesnsson, "Application-Driven Architecture Synthesis," Kluwer Academic Publishers, 1993.

[DuRa92] N. Dutt and C. Ramchandran, "Benchmarks for the 1992 High Level Synthesis Workshop," Technical Report 92-107, University of California, Irvine.

[GaVaNaGo94] D. D. Gajski, F. Vahid, S. Narayan and J. Gong, "Specification and Design of Embedded Systems," Prentice-Hall, 1994.

[HePa90] J. L. Hennessy and D. A. Patterson, "Computer Architecture - a quantitative approach," Morgan Kaufman Publishers 1990.

[KoNiDu94] D. J. Kolson, A. Nocolau and N. Dutt, "Integrating Program Transformations in the Memory-Based Synthesis of Image and Video Algorithms," Proceedings, ICCAD'94, pp 27-30, San Jose, CA, Nov. 1994.

[LaEr94] M. D. Ergegovac and T. Lang, "Division and Square Root - Digit-Recurrence Algorithms and Implementations," Kluwer Academic Publishers, 1994.

[Navabi93] Z. Navabi, "VHDL : analysis and modeling of digital systems," McGraw-Hill 1993.

[PaMu93] F. C. Park and A. P. Murray, "Computational and Modeling Aspects of the Products-of-Exponentials Formula for Robot Kinematics," IEEE Transactions on Automatic Control, 1993.

[PrTeVeFl92] W. H. Press, et. al., "Numerical Recipes in C: The Art of Scientific Computing," Cambridge University Press, 1992.

[RTPC85] IBM RT PC Hardware Technical Reference (C), 1985.

[ThDu94] A. B. Thordarson and N. Dutt, "A VHDL Model and Testbench for the IBM RT-PC Risc Processor," UCI CADLAB document, Univ of California, Irvine, 1994.

[Ug95] H. F. Ugurdag, Personal Communication, 1995.

[VaGoNa94] F. Vahid, J. Gong and S. Narayan, "The SpecCharts/SpecSyn User's Manual," University of California, Irvine, 1994.

[VaNaGa91] F. Vahid, S. Narayan and D. D. Gajski, "SpecCharts: A language for system level synthesis," Proceedings of the International Symposium on Computer Hardware Description Languages and their Applications, 1991.

[VeJaVr86] R.N.J. Veldhuis, A.J.E.M. Janssen and L. B. Vries, "Adaptive Interpolation of Discrete-Time Signals That Can Be Modeled as Autoregressive Processes," IEEE Trans. Acoustics, Speech and Signal Processing, Vol. 34, No. 2, 1986.