Benchmark-Circuits for Hardware -Verification

v1.2.1

Thomas Kropf
Institut für Rechnerentwurf und Fehlertoleranz
Universität Karlsruhe, Kaiserstr. 12, 76128 Karlsruhe, Germany
email: Thomas.Kropf@informatik.uni-karlsruhe.de
WWW: http://goethe.ira.uka.de/hvg/

This document describes the IFIP WG10.5 hardware-verification benchmark circuits, intended for evaluating different approaches and algorithms for hardware verification. The paper presents the rationale behind the circuits, describes them briefly and indicates how to get access to the verification benchmark set.

1 Introduction

Although having many drawbacks, benchmark circuits allow a more succinct and direct comparison of different approaches for solving a certain problem. This has lead to sets of widely accepted circuits e.g. in the area of testing [BrPH85, BrBK89] or high-level synthesis [VRMK91].

In the area of hardware verification, this has lead e.g. to the suggestion of "interesting" circuits, like Paillet's set of seven sequential circuits [Pail85]. One of the first efforts to provide circuits for a broader community has been done by Luc Claesen for the 1990 International Workshop on Applied Formal Methods for VLSI Design [FMVD90]. The most prominent circuit evolving from this effort was the "Min_Max-Circuit".

The lack of additional, generally available verification benchmark circuits got aware in the preparation of the 2nd International Conference on Theorem Provers in Circuit Design (TPCD94). The motivation to provide additional benchmark circuits together with already ongoing standardization efforts of IFIP, coordinated by Jørgen Staunstrup [Stau93], has led to an enhanced set of circuits. Thanks to J. Staunstrup, in the meantime these circuits have become the official "IFIP WG10.2 Hardware Verification Benchmark Circuit Set". It will be maintained and enhanced on a long term basis to promote a standardized benchmark set in the hardware verification community.

For the circuits a complete and self-contained implementation is provided, done in a commercial design system [Fram92] (leading e.g. to additional timing diagrams for clarification) as well as a clear specification using the standardized hardware description language VHDL (see Section 2.2 "Verification Problem Presentation"). This puts a comparison of different verification approaches on a sound basis, since - if the given implementations are used - identical circuits are verified instead of different designs implemented in a way especially suited for a certain approach. Moreover, people are not forced to tediously design the circuits before they can be verified - the latter being the main interest of people looking into these circuits. Naturally, when dealing with formal synthesis, the implementations provided here are less interesting and the specifications are the main thing to deal with.

A set of verification benchmark circuits has to provide easily usable circuits to evaluate and to compare different approaches to hardware verification. This comprises:

- availability via the World Wide Web and anonymous ftp,
- a high degree of diversity with regard to the underlying verification task (see Section 2.3 “Classification of the Verification tasks”),
- circuit descriptions without ambiguity, which are succinct and self-contained and
- circuits which span a wide range from introductory examples to real verification challenges [Stau93].

2 The Benchmark Circuits

Currently, there is the set of benchmark circuits given in table 2-1. The main sources for these circuits have been the previous IFIP benchmark set [Stau93] and various textbooks on circuit design.

Circuit Name	Short Circuit Description
Single Pulser	cuts input pulses to a fixed length
Traffic Light Controller	simplified controller for a traffic light
N-bit Adder	sum of two bitvectors of length N
Min_Max	the mean value of incoming integers
Black-Jack Dealer	the dealer’s hand of a card game
Arbiter	access to shared resources for N clients
Rollback Chip	coprocessor for distributed simulation
Tamarack Processor	simplified microprocessor
Stop-Watch	digital stopwatch with 3 digits and 2 buttons
GCD	Greatest Common Divisor
Multiplier	N-bit Multiplier
Divider	N-bit Divider
FIFO	asynchronous FIFO queue with N places
Assotiative Memory	simple assotiative $N \times M$ memory
1dim Systolic Array	onedimensional systolic filter array
2dim Systolic Array	two-dimensional systolic array for matrix multiplication

Table 2-1: Current Benchmark Circuits

2.1 Releases

Each release of the circuit is labeled with a version number, release number and patch level in the form v<Version>.<Release>.<Patchlevel>.

A version number is provided to make the inclusion of new circuits more explicit. Unfortunately we did not succeed in providing full implementation descriptions for all circuits in the first release. As these are provided, the release number is incremented.

Although the circuits have been designed with much care, reality-driven pessimism suggests that there will be the necessity for “patches” (i.e. bug fixes) at least in the first stage of the release process.

All changes are documented in a history file, patches are also explained in the documentation of the relevant circuits. You may want to be added to an emailing list dedicated to inform people about the actual verification benchmark status (see Section 3.4 “Email list”).

2.2 Verification Problem Presentation

Unambiguously specifying circuits in a general way without being forced to a certain description philosophy by the underlying notation is a challenge itself. The only way to circumvent a fixed description formalism is to use only informal descriptions (natural language, drawings[1], timing diagrams, etc.). However, then the descriptions are often not as crisp and exact as it is necessary especially for formal verification, where the information given here has to be translated into formal description languages like predicate logics, temporal logics, process algebras, Z and so forth.

Besides all drawbacks, we decided to provide more (and more formal) information for each circuit. This comprises for each circuit of:

- a description of the specification and the implementation in plain English,
- schematic diagrams of the implementation[2],
- a netlist of the implementation based on structural VHDL and
- a specification of the circuit in VHDL.

2.2.1 VHDL

The decision to use VDHL as a formalization means especially for specifications is probably the most disputable one for obvious reasons: the lack of a clean VDHL semantics, the danger of imposing a certain specification style, impossibility for expressing nondeterminism and so forth. However, in our opinion the advantages of providing a standardized (and simulatable) specification outweighs the disadvantages. Moreover, we tried to avoid ambiguous VDHL constructs and VHDL specifications may further encourage VHDL based verification (or at least the discussion about the “right” specification language is stimulated).

We use a restricted set of VHDL which should be sufficiently simple so that no semantic ambiguities occur.

2.2.2 Storage Elements and Multiplexers

A register is treated as a *base module* (see Section 2.2.3 “Base Module Library”). It is used as follows: The inputs **S** is connected to input lines named **`Store(Name)`**. A signal is stored, if **`Store(Name)`** =1 and there is a $0 \rightarrow 1$ transition at the clock input.

Lines named **`SelectXY`** are controlling a multiplexer or a demultiplexer: **`SelectXY`** = 0 connects input 0 of a multiplexer to the output (output 0 of a demultiplexer) and **`SelectXY`** = 1 connects input 1 to the output (output 1 of a demultiplexer).

Busses are provided with their names and number of signal lines: **`DataIn<31:0>`** denotes a 32 bit bus for reading data.

---

1. For some circuits, original schematics of a realization in the semi-custom design system CADENCE have been added. However, to fit these drawings on a single page they had to be shrunk in many cases so that labels etc. may not be well readable anymore. Nevertheless, the remaining circuit documentation without these figures is completely sufficient as an implementation description.
2. Schematics are based on an implementation of each circuit using a commercial semi-custom design system.

2.2.3 Base Module Library

Most circuit implementations use a predefined base module library which contains parameterized modules. Parameters are delay time and, for certain elements, the bitwidth. The library contains mainly simple gates and storage elements, listed in table 2-2. The respective VHDL descriptions can be found in the file `GateLib.vhd`.

Module Name	Description	Variable Bitwidth
INV	inverter	no
BUF	buffer	no
nBUF	generic n bit buffer	yes
NAND2	two input nand	no
NAND3	three input nand	no
AND2	two input and	no
AND3	three input and	no
AND4	four input and	no
AND5	five input and	no
NOR2	two input nor	no
NOR3	three input nor	no
OR_2[a]	two input or	no
OR3	three input or	no
OR4	four input or	no
OR5	five input or	no
NXOR2	two input equal	no
XOR2	two input exor	no
MUX	two input, one output, one select multiplexer	no
DMUX	one input, two output, one select demultiplexer	no
DL	D-latch	no
DFF	D-flipflop	no
DFFs	setable D-flipflop	no
DFFsr	setable and resetable D-flipflop	no
RSFFR	resetable RS-flipflop	no
RSFF	RS-flipflop	no
nMUX	generic n-bit 2:1 multiplexer	yes
nDMUX	generic n-bit 2:1 demultiplexer	yes
nREG	generic n-bit register with enable	yes
nREGr	generic n-bit register with enable and reset	yes
HA	half adder	no
FA	full adder	no
nINC	n-bit incremented	yes
CRA	n-bit carry ripple adder	yes
AddSub	n-bit carry ripple adder and subtractor	yes
nSREG	n-bit shift register	yes
nCMPO	n-bit equal zero test	yes

Table 2-2: Elements of the Base Module Library

Module Name	Description	Variable Bitwidth
cCMPN	compare tow n-bit vectors	yes
nLSH	n-bit left shift	yes
nRSH	n-bit right shift	yes
expand	expand bit vector by adding zeros	yes
priority	lower bit has priority others are suppressed	yes

Table 2-2: Elements of the Base Module Library

a. underscore necessary due to name conflicts in SYNOPSIS

2.2.4 Graphical Notation

The schematic diagrams consist of modules which are drawn according to the notation given in table 2-3.

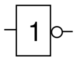
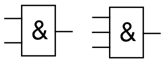
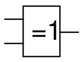
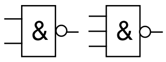
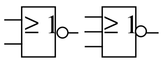
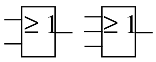
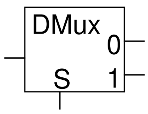
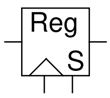
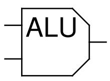
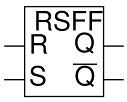
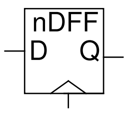
Module type	Graphical Notation
Inverter	
AND gates	
EXOR gate	
NAND gates	
NOR gates	
OR gates	
Demultiplexer 1:2	
Register	
Arithmetic Logic Unit	
RS-Flipflop	
Register with n Flipflops	

Table 2-3: Schematic Drawing Symbols

2.3 Classification of the Verification tasks

There are mainly three verification tasks to be distinguished when talking about hardware verification:

1. verifying that a circuit specification is what it should be,
2. verifying that a given implementation behaves identically to a given specification and
3. verifying important (e.g. safety critical) properties of a given implementation.

According to [Stau93], the first is called *requirements capture*, the second *implementation verification* and the third *design verification*.

The three tasks are often expressed in terms of a specification S and an implementation I, where a *complete* verification denotes some form of equivalence between S and I ($S = I$, $S \approx I$, $S \Leftrightarrow I$) and a *partial* verification denotes some form of implication ($I \Rightarrow S$, $I \supseteq S$, $I \mid\!- S$):

The first task is a partial verification (with S describing properties of the circuit specification and I being the circuit specification), the second is a complete verification (with S being the specification and I being the implementation) and the third is again a partial verification (with S describing the properties of the circuit implementation and I being the circuit implementation).

It is to be noted that if e.g. exact computation times are not stated in a specification then we have to cope with a verification problem of the third kind, since in that case an equivalence proof is not possible.

All three verification tasks are covered by the benchmark circuits.

2.4 Circuit Classification

Every classification scheme has its drawbacks, but we found the following circuit properties especially useful for classification purposes.

A circuit may be classified in several dimensions as depicted in table 2-4. Most of the criteria are self-explaining, besides complexity. In the area of testing usually the number of internal lines is directly used as a complexity measure [BrPH85, BrBK89], motivated by the designated application of the circuits: test pattern generation. Using a classical stuck-at fault model, the set of faults to be treated equals the number of internal lines. Hence a circuit s713 denotes a sequential circuit with 713 internal lines [BrBK89].

In the area of hardware verification a similar complexity measure is not as obvious (at least we did not found any meaningful). Hence we use the coarse measure, proposed by J. Staunstrup: an example is either introductory, illustrative or a real challenge [Stau93].

Classification Criterion	Value Set	
Abstraction level	system	(s)
	algorithmic	(a)
	register-transfer	(r)
	gate	(g)
	transistor	(t)
Synchronicity of the implementation	synchronous	(s)
	asynchronous	(a)
	combinational	(c)
Hierarchy of the implementation	hierarchical	(h)
	flat	(f)

Table 2-4: Possible Circuit Classifications

Classification Criterion	Value Set
Determinism	deterministic (d) nondeterministic (n)
Genericity	generic (g) concrete with (optional) bitwidth *n* (c*n*)
Type	controller (c) data path (d) mixed (both) (m)
Complexity	introductory (i) standard illustrative (s) challenge (c)

Table 2-4: Possible Circuit Classifications

The abbreviations given in table 2-4 may be used to characterize each circuit using the following “signature”:

`<Name>:<Spec>-<Imp>.<Sync>.<Hier>.<Det>.<Gener>.<Type>.<Compl>`

A circuit `GCD.a-r.s.h.d.g.m.i` denotes the Greatest Common Divisor circuit, which has a specification on algorithmic level and an implementation on register-transfer level. It is a synchronous, hierarchical and deterministic design with arbitrary bit width. Consisting of a controller and a data path it is a small, i.e. an introductory example.

Using this classification scheme, we can characterize all circuits as shown in table 2-5.

Circuit Name	Classification
Single Pulser	`Pulser.g-g.s/a.f.d.c1.c.i`
Traffic Light Controller	`TLC.r-g.s.f.d.c1.c.i`
N-bit Adder	`Adder.a-g.c.h.d.g.d.i`
Min_Max	`Min_Max.a-g.s.h.d.c8/g.d.s`
Black-Jack Dealer	`Dealer.a-g.?.f.n.c.c.s`
Arbiter	`Arbiter.r-g.s.f.n.g.c.s`
Rollback Chip	`Rollback.?-?.?.h.d.?.m.s`
Tamarack Processor	`Tamarack.a-g.s.h.d.?.m.s`
Stop-Watch	`Stopwatch.r-g.s.h.d.c.m.s`
GCD	`GCD.a-g.s.h.d.g.m.i`
Multiplier	`Mult.a-g.c.h.d.g.d.s`
Divider	`Div.a-g.c.h.d.g.d.s`
FIFO	`FIFO.a-g.a.h.d.g.m.s`
Assotiative Memory	`Assoc.r-g.s.h.d.g.d.s`
1dim Systolic Array	`1Syst.a-g.s.h.d.g.d.s`
2dim Systolic Array	`2Syst.a-g.s.h.d.g.d.s`

Table 2-5: Classification of the Benchmark Circuits

3 How to get the benchmark circuits

The most convenient way to a get access to the benchmark suite is via the World Wide Web. Using a WWW browser like Mosaic you can get the latest informations using the URL http://goethe.ira.uka.de/benchmarks/.

You can also directly use an anonymous FTP-server. All benchmark circuits as well as PostScript versions of the documents (including this paper) have been made available there in the directory pub/benchmarks. The FTP-server is reachable as goethe.ira.uka.de (current IP address: 129.13.18.22). The server will always hold the newest benchmark version (see Section 2.1 “Releases”).

Simple ASCII-files end in .txt, Postscript files always end in .ps, .ps.Z or ps.gz. The files with endings .gz (.Z) has been compressed using the UNIX gzip (compress) command. They must be transferred using binary ftp mode and must be expanded using the UNIX gunzip (uncompress) command before they are readable or printable. Tar-Files end in .tar (compressed .tar.gz or .tar.Z) and contain a whole directory in one file. The directory content may be rebuild by executing tar -xfv <name>.tar after having expanded the respective file.

3.1 Physical Organization

The main directory /pub/benchmarks contains:

- README — how to use and retrieve the benchmarks
- It_is_version_x.y.z — an empty dummy file indicating the current version, release and patchlevel
- Introduction.ps — this document as a PostScript file
- Whole_documentation.ps — this document plus all (currently available) circuit descriptions as a single PostScript File
- History.txt — the version history of the benchmarks
- GateLib.vhd — behavioral descriptions of all base modules (see Section 2.2.3 “Base Module Library”)
- ELEMpack.vhd — interface and componemt declaration of all base elements
- <circuit> — a directory for each benchmark circuit
- documentation — a directory containing the whole documentation in its original FrameMaker 4 format (for those who want to get the document “sources”)
- pending — a directory containing circuits which will probably be included in future releases of the benchmark circuits.[1]

Each circuit directory <circuit> contains at least:

- <circuit>.ps — a PostScript file describing the circuit
- <circuit>.vhd — different VHDL files covering the implementation (<circuit>struc.vhd), a behavioral specification (<circuit>behave.vhd), a testbench for simula-

1. For these circuits the documentation may be incomplete, inconsistent or completely missing.

tion (<circuit>tb.vhd or <circuit>test-bench.vhd) as well as various other files and scripts useful for simulation and synthesis.

3.2 Using the benchmarks

You can use the benchmark circuits in any way which suits your needs. Especially when specifying the problems, you are in no way obliged to use VHDL. The VHDL specifications are mainly provided to clarify the intended proof goals.

However, if for example, you use an implementation completely different from the ones given here (e.g. a simplified version) you should state this clearly whenever you refer to the circuits provided here.

Some of the circuits have been designed hierarchically. If you are flattening the circuits in order to verify them, you should state this also.

3.3 Please Contact us if …

Please contact us if you have any problems, especially

- if you have questions of any kind concerning the benchmarks,
- if you have comments or proposals for changes, additions or even new circuits,
- if you can provide “better” implementations for the circuits,
- if you have problems in printing the PostScript files,
- if you detect errors, inconsistencies or ambiguities, which should be fixed or
- if you have problems in accessing the files.

The easiest way is to send a brief email to Thomas Kropf or to Jørgen Staunstrup (Thomas.Kropf@informatik.uni-karlsruhe.de, jst@id.dth.dk).

3.4 Email list

To inform people about the latest release and patches of the benchmark circuits, we do maintain an informal emailing list. If you want to be added (or deleted) from this list, send a short note to Thomas.Kropf@informatik.uni-karlsruhe.de.

4 Present and Future Activity

At the moment, we are busy simply with completing all proposed circuits and — probably — by making the current set consistent.

The current set of circuits falls short of asynchronous verification examples. Moreover protocol verification problems are not covered, which may also be viewed at as important subaspects of circuit verification. To cover lower description levels, we also would like to add some switch-level or transistor level verification examples.

There is still a lack of “challenging” verification examples, i.e. circuits which are either of significant size or which reflect “real” commercial designs. As one of these circuits, we will probably provide a large RISC-processor: the DLX of Patterson and Hennesy [HePa90].

5 Acknowledgments

We like to thank the following people, who have helped in creating this benchmark set and documentation:

- Ralf Reetz — for significant ideas and contributions like the parameterized base module library,
- Hans-Peter Eich — for implementing all circuits in a commercial design system and for providing all the figures and timing diagrams,
- Claus-Jürgen Thomas — for updating the circuits and making them synthesizable
- Klaus Schneider — for pointing out interesting verification problems,
- Jørgen Staunstrup, — for supporting this activity via IFIP and suggesting the IFIP circuits and
- the Authors of TPTP — for the excellent documentation of their theorem proving benchmarks [SuSY93], which helped us in structuring this document

6 Literature

[SuSY93] C. Suttner, G. Sutcliff, and T. Yemenis. *The TPTP (Thousands of Problems for Theorem Provers) Problem Library*. TU Muenchen, Germany and James Cook University, Australia, via anonymous ftp flop.informatik.tu-muenchen.de, tptpv1.0.0 edition, 1993.

[BrBK89] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *International Symposium on Circuits and Systems*, May 1989.

[BrPH85] F. Brglez, P. Pownall, and R. Hum. Accelerating ATPG and fault grading via testability analysis. In *International Symposium on Circuits and Systems*, 1985.

[Fram92] CADENCE Design Framework II version 4.2a. Reference Manual, February 1992.

[Pail85] J.-L. Paillet. Un Modele de Fonctions Sequentielles pour la Verification Formelle de Systemes Digitaux. Technical Report 546, IMAG-ARTEMIS, Grenoble, June 1985.

[Stau93] J. Staunstrup. IFIP WG 10.2 collection of circuit verification examples, November 1993.

[HePa90] J.L. Hennessy and D.A. Patterson. *Computer Architecture: A quantitative Approach*. Morgan Kaufmann Publishers Inc., San Mateo, CA, USA, 1990.

[FMVD90] L. Claesen, editor. *International Workshop on Applied Formal Methods for VLSI Design*, Leuven, Belgium, 1990.

[VRMK91] R. Vemuri, J. Roy, P. Mamtora, and N. Kumar. Benchmarks for high-level synthesis. Technical Report ECE-DDE-91-11, Laboratory for Digital Design Environments, ECE Dept., University of Cincinnati, Ohio, USA, November 1991.