Verifying Imprecisely Working Arithmetic Circuits*

M. Huhn, K. Schneider, Th. Kropf, and G. Logothetis
Universität Karlsruhe
Institut für Rechnerentwurf und Fehlertoleranz (Prof. Dr.-Ing. D. Schmid)
P.O. Box 6980, 76128 Karlsruhe, Germany
mailto: {huhn,schneide,kropf,logo}@informatik.uni-karlsruhe.de
http://goethe.ira.uka.de/hvg/

Abstract

If real number calculations are implemented as circuits, only a limited preciseness can be obtained. Hence, formal verification can not be used to prove the equivalence between the mathematical specification based on real numbers and the corresponding hardware realization. Instead, the number representation has to be taken into account in that certain error bounds have to be verified.

For this reason, we propose formal methods to guide the complete design flow of these circuits from the highest abstraction level down to the register-transfer level with formal verification techniques that are appropriate for the corresponding level. Hence, our method is hybrid in the sense that it combines different state-of-the-art verification techniques. Using our method, we establish a more detailed notion of correctness that considers beneath the control and data flow also the preciseness of the numeric calculations. We illustrate the method with the discrete cosine transform as a real-world example.

1. Introduction

Many hardware systems implement algorithms that work on the real numbers. Typical applications are consumer electronics like mobile phones or digital cameras which heavily use digital signal processing to perform filter computations like a Fast Fourier Transform (FFT) or a Discrete Cosine Transform (DCT) [9].

Although the underlying algorithms assume real numbers, hardware realizations only allow a number representation with a finite, fixed number of bits. Therefore, hardware implementations of these algorithms are only approximations where the real numbers are implemented with a limited preciseness.

Most approaches to formal verification ignore this fact and perform the verification at an abstract level, where the data values are viewed as real numbers. Thereby, the control and data flow can be proven correct, which is clearly necessary to avoid malfunctioning circuits. However, even if the verification at the abstract level succeeds, the circuit may still produce wrong results due to the impreciseness of the data values.

To establish a more detailed notion of correctness for these circuits, the impreciseness of the data words has to be taken into account. We therefore propose a verification flow that guides the design flow starting at the algorithmic level down to the register-transfer level.

At the algorithmic level, we are able to compare different algorithms where we view the data words as real numbers. For this reason, we propose to use *computer algebra* or *automated theorem proving*, in particular *term rewriting* as methods to solve the resulting verification problems. At the next level of abstraction, the bitwidths of the circuits are fixed. Hence, our verification problem is now to verify that certain error bounds for the results are met. To determine these error bounds, we propose to analyze the numerical operations with a computer algebra system.

As usually a lot of optimizations are performed at the register-transfer level, we must additionally be able to compare different circuits at this level. It will turn out that *model checking techniques* are most appropriate at this level. In contrast to automated theorem proving, these methods allow a fully automated verification and moreover, they directly support a bit-oriented description. On the other hand, they are restricted to finite state systems and can therefore not deal with real numbers.

We illustrate the usefulness of our method by comparing different algorithms for the discrete cosine transform (DCT). In particular, we consider the verification problems in the design flow at different abstraction levels and show how these can be solved with state-of-the-art methods.

*This work has been financed by the 'Deutsche Forschungsgemeinschaft' by projects Automated System Design, SFB 358/C2 and project 'Verification of embedded systems'.

Loeffler, Ligtenberg and Moschytz (LLM_DCT):

$$
\begin{aligned}
L_{0,0} &:= x_0 + x_7 & L_{1,0} &:= L_{0,0} + L_{0,3}\\
L_{0,1} &:= x_1 + x_6 & L_{1,1} &:= L_{0,1} + L_{0,2}\\
L_{0,2} &:= x_2 + x_5 & L_{1,2} &:= L_{0,1} - L_{0,2}\\
L_{0,3} &:= x_3 + x_4 & L_{1,3} &:= L_{0,0} - L_{0,3}\\
L_{0,4} &:= x_3 - x_4 & L_{1,4} &:= \mathrm{ROT}_0(L_{0,4}, L_{0,7}, 3)\\
L_{0,5} &:= x_2 - x_5 & L_{1,5} &:= \mathrm{ROT}_0(L_{0,5}, L_{0,6}, 1)\\
L_{0,6} &:= x_1 - x_6 & L_{1,6} &:= \mathrm{ROT}_1(L_{0,5}, L_{0,6}, 1)\\
L_{0,7} &:= x_0 - x_7 & L_{1,7} &:= \mathrm{ROT}_1(L_{0,4}, L_{0,7}, 3)
\end{aligned}
$$

$$
\begin{aligned}
L_{2,0} &:= L_{1,0} + L_{1,1} & z_0 &:= L_{2,0}\\
L_{2,1} &:= L_{1,0} - L_{1,1} & z_1 &:= (L_{2,4} + L_{2,7})\\
L_{2,2} &:= \mathrm{ROT}_0(L_{1,2}, L_{1,3}, 6) & z_2 &:= \sqrt{2}\, L_{2,2}\\
L_{2,3} &:= \mathrm{ROT}_1(L_{1,2}, L_{1,3}, 6) & z_3 &:= \sqrt{2}\, L_{2,5}\\
L_{2,4} &:= L_{1,4} + L_{1,6} & z_4 &:= L_{2,1}\\
L_{2,5} &:= L_{1,7} - L_{1,5} & z_5 &:= \sqrt{2}\, L_{2,6}\\
L_{2,6} &:= L_{1,4} - L_{1,6} & z_6 &:= \sqrt{2}\, L_{2,3}\\
L_{2,7} &:= L_{1,5} + L_{1,7} & z_7 &:= L_{2,7} - L_{2,4}
\end{aligned}
$$

Arai, Agui and Nakajiama (AAN_DCT):

$$
\begin{aligned}
L_{0,0} &:= x_0 + x_7 & L_{1,0} &:= L_{0,0} + L_{0,3}\\
L_{0,1} &:= x_1 + x_6 & L_{1,1} &:= L_{0,1} + L_{0,2}\\
L_{0,2} &:= x_2 + x_5 & L_{1,2} &:= L_{0,1} - L_{0,2}\\
L_{0,3} &:= x_3 + x_4 & L_{1,3} &:= L_{0,0} - L_{0,3}\\
L_{0,4} &:= x_3 - x_4 & L_{1,4} &:= L_{0,4} + L_{0,5}\\
L_{0,5} &:= x_2 - x_5 & L_{1,5} &:= L_{0,5} + L_{0,6}\\
L_{0,6} &:= x_1 - x_6 & L_{1,6} &:= L_{0,6} + L_{0,7}\\
L_{0,7} &:= x_0 - x_7 & q_0 &:= c_2 - c_6\\
 & & q_1 &:= c_2 + c_6
\end{aligned}
$$

$$
\begin{aligned}
L_{2,0} &:= c_4(L_{1,2} + L_{1,3}) & u_0 &:= L_{1,0} + L_{1,1}\\
 & & u_1 &:= L_{3,1} + L_{2,4}\\
L_{2,2} &:= q_0 L_{1,4} + L_{2,5} & u_2 &:= L_{2,0} + L_{1,3}\\
L_{2,3} &:= c_4 L_{1,5} & u_3 &:= L_{3,3} - L_{2,2}\\
L_{2,4} &:= q_1 L_{1,6} + L_{2,5} & u_4 &:= L_{1,0} - L_{1,1}\\
L_{2,5} &:= c_6(L_{1,4} - L_{1,6}) & u_5 &:= L_{3,3} - L_{2,2}\\
L_{3,1} &:= L_{0,7} + L_{2,3} & u_6 &:= L_{1,3} - L_{2,0}\\
L_{3,3} &:= L_{0,7} - L_{2,3} & u_7 &:= L_{3,1} - L_{2,4}
\end{aligned}
$$

Figure 1. Optimized DCT algorithms

2. The Discrete Cosine Transform

The DCT is a spectral transformation that is often used for audio and image compression. Examples of DCT applications are GSM speech transcoding, JPEG still video and MPEG motion video compression. For a more detailed presentation of the DCT and its application in JPEG see [9].

As the DCT is a representative of complex, data driven algorithms based on real numbers, it is a good example to illustrate our methods. We stress however that the techniques presented in the following are not limited to the DCT.

2.1. Formal Definition of the DCT

The DCT is a two-dimensional operation that transforms a matrix $X \in \mathbb{R}^{8\times 8}$ to a matrix $Y \in \mathbb{R}^{8\times 8}$ according to the following sum, where $a_{i,j} := \frac{1}{2}\cos\left((2j+1)i\frac{\pi}{16}\right)$ for $i > 0$ and $a_{0,j} := \frac{1}{2}\sqrt{2}$ holds:

$$y_{i,j} := \sum_{k=0}^{7}\sum_{l=0}^{7} x_{k,l}\, a_{j,l}\, a_{i,k} \tag{1}$$

The two-dimensional DCT is often reduced to the one-dimensional DCT that is defined as follows: Given a real valued vector $\vec{x} = (x_0, \ldots, x_7) \in \mathbb{R}^8$, the one-dimensional DCT is a linear transformation $\Phi : \mathbb{R}^8 \to \mathbb{R}^8$ with the constants $a_{i,j}$ as defined above:

$$\begin{pmatrix} y_0 \\ \vdots \\ y_7 \end{pmatrix} := \underbrace{\begin{pmatrix} a_{0,0} & \ldots & a_{0,7} \\ \vdots & \ldots & \vdots \\ a_{7,0} & \ldots & a_{7,7} \end{pmatrix}}_{=:\,\mathcal{A}} \cdot \begin{pmatrix} x_0 \\ \vdots \\ x_7 \end{pmatrix} \tag{2}$$

Using the above matrix $\mathcal{A}$, the two-dimensional DCT is computed as $Y = \mathcal{A}X\mathcal{A}^{-1}$. Hence, the two-dimensional DCT can be implemented by 16 applications of the one-dimensional DCT (eight for the rows followed by eight on the columns). Although the calculation of the two-dimensional DCT by 16 one-dimensional DCTs is not optimal, there exist efficient realizations of the one-dimensional DCT that lead to almost optimal results also for the two-dimensional case.

2.2. Optimized Implementations

Optimizations of the DCT can be obtained, if symmetries of the trigonometric functions are exploited to share common subterms. Note that the matrix $\mathcal{A}$ contains only 7 different coefficients $c_1, \ldots, c_7$ that are defined as $c_k := \cos\left(k\frac{\pi}{16}\right)$.

In figure 1, the DCT algorithm (LLM_DCT) by Loeffler, Ligtenberg and Moschytz [7] is given. This implementation makes use of trigonometric addition theorems to generate additional common subterms. Moreover, it uses a rotation operation is used which is defined as

$$
\begin{aligned}
\mathrm{ROT}_0(x_0, x_1, k) &:= x_0 c_k + x_1 s_k,\\
\mathrm{ROT}_1(x_0, x_1, k) &:= -x_0 s_k + x_1 c_k,
\end{aligned}
$$

with $s_k := \sin\left(k\frac{\pi}{16}\right)$. Using an intermediate variable $\ell := c_k(x_0 + x_1)$, we get $\mathrm{ROT}_0(x_0, x_1, k) := \ell + (s_k - c_k)x_1$ and $\mathrm{ROT}_1(x_0, x_1, k) := -(s_k + c_k)x_0 + \ell$. This requires 3 multiplications and 3 additions instead of 4 multiplications and 2 additions. As multiplications are in general more expensive than additions, this is reasonable. Summing up,

LLM_DCT requires only 13 multiplications and 29 additions. Note however, that LLM_DCT computes a *scaled* DCT, i.e. we have $z_i = 2\sqrt{2}y_i$, where the y_i are the original DCT results. For most applications like JPEG, the scaled values are sufficient.

One of the most efficient versions currently known is based on the Discrete Fourier Transform and has been presented by Arai, Agui and Nakajiama [9]. This implementation is given in figure 1 as AAN_DCT. It only requires 5 multiplications and 29 additions (the additions for determining the constants q_0 and q_1 can be done in advance). In contrast to LLM_DCT, AAN_DCT makes use of further non-trivial trigonometric laws to construct common subterms. The AAN_DCT is related to the original DCT by the following scaling equations: $u_0 = 2\sqrt{2}y_0$ and $u_i = 4c_i y_i$ for $i = 1, \ldots, 7$.

3. Verifying DCT Implementations

In this section, we present our verification approach which guides the design flow, ranging from the initial mathematical definition of the two-dimensional DCT to the final register-transfer level implementation (figure 2). Obviously, the verification problems differ for the various steps of figure 2.

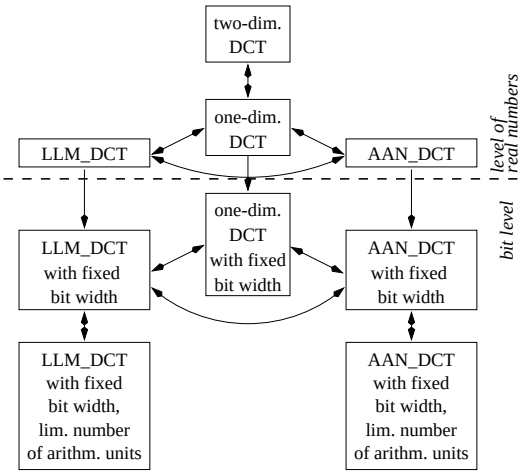

Figure 2. DCT Design Flow

3.1. At the level of real numbers

Symbolic computation. The reduction from the two-dimensional DCT to 16 one-dimensional DCTs on the rows and columns is an easy exercise in linear algebra. For an automated proof of this step, computer algebra systems like Mathematica [11] or Maple [3] are most appropriate. These systems are tailored for symbolic calculations in mathematics. In particular, they offer data types and operations for common mathematical structures like real numbers, matrices, trigonometry, etc. . All required data types for the DCT verification are available in the computer algebra system *Mathematica*. The proof of the equivalence of equation 1 and the alternative definition $Y = \mathcal{A}X\mathcal{A}^{-1}$ is done by Mathematica's `Simplify` command, which performs a fully-automated proof in a matter of seconds.

The very same approach based on Mathematica can be used to establish the equivalence of different implementations of the one-dimensional DCT as e.g. those given in figure 1. For instance, for LLM_DCT we show that $\mathcal{A} \cdot \vec{x} - \frac{1}{4}\sqrt{2}\vec{z} = \vec{0}$ where $\vec{z}$ is the output vector of the LLM_DCT with input $\vec{x}$. As again symbolic simplification is used, the proof is valid for arbitrary inputs. However, in LLM_DCT two out of eight vector coefficients cannot be reduced to zero, since Mathematica lacks some trigonometric laws.

Term Rewriting. Alternatively, the correctness of DCT variants on the basis of real numbers can be proven by term rewriting, e.g. using the term rewrite system RRL [6]. Term rewriting is a semiautomatic technique for proving theorems in first order predicate calculus with equality. A finite set of axioms specifying a first order theory can be extended by a completion procedure to either obtain a decision procedure for the theory or to deduce the inconsistency of the theory. A major application field for term rewriting is the reasoning on abstract data types which specify the structure of data using constructors and the operations on them in terms of recursive equations.

Compared to computer algebra, the advantage of term rewriting is that it is not restricted to given data types. The disadvantage is that the verification has to be done in a more interactive manner. In particular, we first have to specify a data type for real numbers in form of a term rewrite system.

In the following, we illustrate how to prove the equivalence of equation 2 and LLM_DCT by RRL. First, we have to establish parts of the real number theory by listing some required axioms. In particular, the optimizations used in the DCT algorithms rely on the fact that real numbers are an algebraic field and some properties of the trigonometric functions. The required axioms are listed in the upper part of table 1. Additionally, the operators $+$ and $*$ are specified to be associative and commutative[1].

We enter all these laws as rewrite rules and add operator precedences to establish the rewrite directions. Then we obtain a confluent rewrite system by means of the Knuth-Bendix completion procedure. After that we add the addition theorems on trigonometric functions that are used in LLM_DCT (given in the lower part of table 1). The resulting

[1] Essentially, the laws of table 1 specify that the real numbers are mathematically speaking an algebraic field. Hence, we have not characterized the reals completely. In particular, we do not need the supremum axiom.

Laws for the data type *Real*	
$x + 0 := x$	$s_1 = c_7$
$x * 1 := x$	$s_2 = c_6$
$x * 0 := 0$	$s_3 = c_5$
$x * (y + z) := (x * y) + (x * z)$	$s_4 = c_4$
$x + -(x) := 0$	$s_5 = c_3$
$x * -(y) := -(x * y)$	$s_6 = c_2$
$-(-(x)) := x$	$s_7 = c_1$
$\sqrt{2} * \sqrt{2} := 1 + 1$	$s_0 = 0 = c_8$
$\sqrt{2} * c_4 = 1$	$s_8 = 1 = c_0$
$c_4 + c_4 = \sqrt{2}$	

Addition theorems for cos	
$\sqrt{2} * c_1 := c_3 + c_5$	$\sqrt{2} * c_5 := c_1 - c_7$
$\sqrt{2} * c_3 := c_1 + c_7$	$\sqrt{2} * c_7 := c_3 - c_5$

Table 1. Axioms for the real numbers

rewrite system is then powerful enough to prove for all coefficients $z_i = 2\sqrt{2}y_i$ by simple rewriting (using the boolean ring method).

3.2. From the real number to the bit level

The optimized DCT algorithms are targeted towards hardware implementation. As emphasized before, this requires the use of a bit vector representation of the real numbers, where usually a fixpoint number representation is used. For this reason, the constants c_k are scaled by 2^n, where n is the corresponding bitwidth, i.e. they are replaced with $\tilde{c}_i := \lfloor 2^n c_i \rfloor$. Hence, the results of the DCT are also scaled by the factor 2^n. As the (scaled) real numbers are approximated by integers (which are implemented with a finite number of bits) the implementations are often viewed as *integer DCTs*.

We now establish suitable bounds for computing the LLM_DCT versus the original DCT (the other comparisons can be done in the same lines). We first obtain that $z_i = 2\sqrt{2}y_i$ holds for $i \in \{0, 2, 3, 4, 5, 7\}$ no matter what integer constants $\tilde{c}_i \in \mathbb{Z}$ are used. Hence, the finite number representation is not critical for these outputs. However, the outputs y_1 and y_6 crucially depend on the choice of the integer constants $\tilde{c}_i \in \mathbb{Z}$. We determine the following limit for $\left|z_1 - 2\sqrt{2}y_1\right|$:

$$\delta_1(n) := \left(\begin{array}{l} |2\tilde{c}_4\tilde{c}_7 - \tilde{c}_3 + \tilde{c}_5| + |2\tilde{c}_4\tilde{c}_5 - \tilde{c}_1 + \tilde{c}_7| + \\ |2\tilde{c}_4\tilde{c}_3 - \tilde{c}_1 - \tilde{c}_7| + |2\tilde{c}_4\tilde{c}_1 - \tilde{c}_3 - \tilde{c}_5| \end{array} \right) 2\hat{x}$$

$\hat{x} := 2^{n-1} - 1$ is thereby the largest number that can be implemented with the corresponding number of bits[2]. Note that the above term evaluates to 0 if we could use the real valued constants $c_k := \cos\left(k\frac{\pi}{16}\right)$, or even $2^n c_k$ instead of $\tilde{c}_k \in \mathbb{Z}$. However, it can be easily shown that there are no integer constants $\tilde{c}_i$ that would reduce the above term to 0. Hence, the outputs will differ and the above term gives us an upper bound for the error. This upper bound in terms of the number of bits is given in figure 3. The line with the circles gives the value of $\delta_1(n)$, while the line with the crosses gives $80\delta_1(n)/(2^n - 1)$. It can be seen that $\delta_1(n)$ converges quickly to zero while $80\delta_1(n)/(2^n - 1)$ is limited by roughly 2π. This means that the integer version of LLM_DCT is exact for the outputs z_0, z_2, z_3, z_4, z_5, and z_7 and is correct at least up to the last three bits for the outputs z_1 and z_6. As similar result can be obtained for $\left|z_6 - 2\sqrt{2}y_6\right|$.

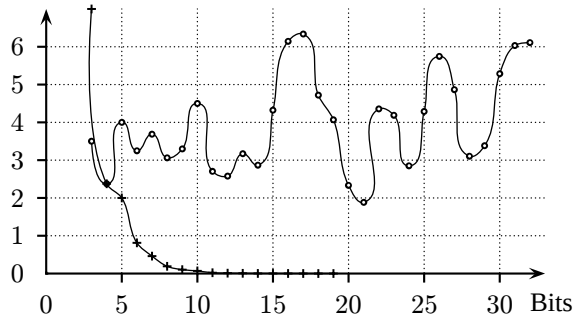

Figure 3. Analytically determined error bounds for fixed bitwidths

3.3. At the bit level

Model checking. Symbolic model checking of temporal logics has become a very popular verification method for reactive systems. Specifications are given in temporal logics like CTL [4]. The model checking itself is based on computing fixpoints of state set transformers. These state sets are represented by their characteristic functions which are implemented by OBDDs [1].

The advantage of the method is that it works fully automatic, the disadvantage is that only finite state systems can be modeled. Consequently, we can not reason at the level of real numbers, i.e. these methods can only be used to compare different DCT algorithms with the same number of bits.

Moreover, one usually has to fight the state explosion problem that arises from the very large finite state models. In particular, it is well-known that the multiplication function can only be represented with OBDDs of exponential size. Although the optimizations of the DCT algorithms aim at using as few multiplication operations as possible, still too much of the complexity remains for a reasonable

[2] For $n + 1$ bits, we have $\hat{x} = 2^n - 1$, since we need one bit for the sign.

representation with OBDDs. We have chosen SMV [8] for the verification of the bit level specifications. However, although SMV is a quite efficient tool, our experiments were limited to about 4 bits.

To fight the complexity problem, a couple of enhanced verification techniques have been developed to verify larger systems. In particular, the abstraction methods developed in [5] are important means to fight the complexity in the DCT example. We use a special abstraction technique based on the Chinese Remainder Theorem (CRT): The CRT states that given relatively prime numbers $p_1, \ldots, p_n$, two numbers $o \leq a, b \leq o + p_1 \cdot \ldots \cdot p_n$ are equal iff for all i, we have $a \bmod p_i = b \bmod p_i$. As additionally the laws $(i_1 + i_2) \bmod p = ((i_1 \bmod p) + (i_2 \bmod p) \bmod p$, $(i_1 * i_2) \bmod p = ((i_1 \bmod p) * (i_2 \bmod p) \bmod p$, and $(i_1 - i_2) \bmod p = ((i_1 \bmod p) - (i_2 \bmod p) \bmod p$ hold, we can reduce the equality of DCT results to the equality of the corresponding remainders wrt. p_i. This allows us to reduce the problem with large bitwidths to a finite number of problems with only a small number of bits. For example, using the numbers $p_1 = 11$, $p_2 = 13$, $p_3 = 14$, and $p_4 = 15$, the range $-105105 \leq x \leq 105105$ is covered which contains all 17 bit broad integers. The abstraction technique allowed to reduce the problem to the remainders which only have 4 bits. The runtimes using SMV were about 140 seconds and roughly 200000 BDD nodes were required which needed 30 MBytes of main memory. Using the CRT abstraction, we were able to verify circuits with more than 20 bits (see also [10]).

Another way to fight the state explosion problem in the verification of arithmetic circuits has been proposed in [2] by introducing Binary Moment Diagrams (BMDs). BMDs allow a compact representation of functions that only have exponentially sized BDDs (e.g. the multiplication). For our experiments on BMDs, we used the latest version of WSMV implemented at the Carnegie Mellon University. WSMV is an extension of SMV based on BMDs.

However, while the runtime for 4 bits was only 17.72 seconds on a Sun Ultra 1 Station, we did not manage to run it for more than 4 bits and stopped the processes after 20 hours. The problem was not the storage consumption of WSMV: the process ran with only about 40 MByte of main memory, but runtime grew too rapidly with the bitwidth. For us, it seems that BMDs are very efficient in terms of storage consumption, but runtime can become very large even with small BMDs.

4. Conclusions

We have shown that formal verification can be applied to circuits that implement algorithms working on real numbers at various abstraction levels of the design. The presented method makes use of state-of-the-art techniques that are suited for the corresponding abstraction level. Clearly, at the levels where the number representation of finitely many bits has to be considered, the correctness can only be stated wrt. certain error bounds. We have shown how formal verification can guide the entire design process and how formal verification tools can be used to compute error bounds and verify the correctness of the circuits wrt. to these bounds.

In particular, we have shown the use of computer algebra and term rewrite systems at the algorithmic level, where we can reason on the real numbers. Computer algebra systems are also useful for determining error bounds for the bit level that can then be verified using model checking techniques. We have also shown that modern abstraction techniques enables the latter to verify extremely large state spaces. Model checking techniques can also be applied to optimized sequential register-transfer level circuits where they additionally verify the correctness of the scheduling.

References

[1] R. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[2] R. Bryant and Y.-A. Chen. Verification or Arithmetic Circuits with Binary Moment Diagrams. In *ACM/IEEE Design Automation Conference (DAC)*, Pittsburgh, June 1995. Carnegie Mellon University.

[3] B. Char, K. Geddes, G. Gonnet, B. Leong, M. Monagan, and S. Watt. *Maple V Language Reference Manual*. 1992.

[4] E. Clarke and E. Emerson. Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic. In D. Kozen, editor, *Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71, Yorktown Heights, New York, May 1981. Springer-Verlag.

[5] E. Clarke, O. Grumberg, and D. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and systems*, 16(5):1512–1542, September 1994.

[6] D. Kapur and H. Zhang. RRL: a rewrite rule laboratory. In Lusk and Overbeek, editors, *Conference on Automated Deduction (CADE)*, pages 768–769. Springer-Verlag, 1988.

[7] C. Loeffler, A. Ligtenberg, and G. Moschytz. Practical fast 1-D DCT algorithms with 11 multiplications. In *International Conference on Acoustics, Speech, and Signal Processing 1989 (ICASSP '89)*, pages 988–99, 1989.

[8] K. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Norwell Massachusetts, 1993.

[9] W. Pennebaker and J. Mitchell. *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, ISBN 0-442-01272-1, 1993.

[10] K. Schneider and M. Huhn. Comparing model-checking and term-rewriting in the verification of an embedded system. In F. Rammig, editor, *DIPES98: International IFIP Workshop on Distributed and Parallel Embedded Systems*, pages 93–102. Kluwer, January 1999.

[11] S. Wolfram. *Mathematica - A System for Doing Mathematics by Computer*. 1991.