COMPARING MODEL CHECKING AND TERM REWRITING FOR THE VERIFICATION OF AN EMBEDDED SYSTEM

Klaus Schneider and Michaela Huhn

Institute for Computer Design and Fault Tolerance
(Prof. Dr.-Ing. D. Schmid)
University of Karlsruhe, P.O. Box 6980, 76128 Karlsruhe
Klaus.Schneider@informatik.uni-karlsruhe.de
Michaela.Huhn@informatik.uni-karlsruhe.de
http://goethe.ira.uka.de/people

Abstract: *There are two main streams for the verification of digital systems: Theorem proving methods such as term rewriting are used for the verification of data oriented systems, and model checking of temporal logics is usually used for the verification of control dominated systems. While theorem proving is an inherently interactive verification method, model checking is performed automatically.*
In this paper, we investigate for the verification of algorithms for computing the discrete cosine transform by means of term rewriting and model checking. We show the advantages and disadvantages of both approaches at different abstraction levels of the design.

1 INTRODUCTION

The development of complex concurrent systems is, in general, an error prone task. This holds in particular for the design of parallel embedded systems, as there are still no design flows that are seamlessly supported by appropriate tools. Hence, the integration of formal verification into the design of these systems has been shown fruitful in that (i) more clearly structured designs are obtained and (ii) design errors can be detected early. For this reason, it is on the one hand desirable to integrate formal verification in the first phases of the design. On the other hand, some relevant details e.g. arising from the partitioning of the system into hardware and software components are yet unknown in these early phases. Hence, formal verification must be used in very early design phases, but must then guide the succeeding design when more implementation decisions are added.

Adding verification to the design of embedded systems requires however another expert in the team: Besides the details and requirements of the application under construction, this expert has to be familiar with different verification techniques (see [Gup92] for an overview). For control oriented systems, model checking is one of the favorite techniques which is supported by well developed tools such as SMV [McM93], SPIN [HP96], HSIS [ABC$^+$94] or SVE [FSS$^+$94]. Data oriented systems can be handled by theorem provers as HOL [GM93] or PVS [ORR$^+$96] and in particular by term rewrite systems such as RRL [KZ88]. Since most real world problems contain both, a nontrivial data and a nontrivial control part, verification often needs a combination of different techniques.

To make verification part of the industrial design process, frameworks and tools have to be provided that allow the uniform and convenient access to different verification techniques. In particular, we aim at integrating different verification tools such as the SMV [McM93], SPIN [HP96], RRL [KZ88], and HOL [GM93]. Providing a collection of different techniques, verification can be done efficiently by choosing the technique that is suited best in the sense that the system can be verified with as little human effort as possible.

In this paper, we study the verification of the discrete cosine transform (DCT) which is a significant part of the JPEG or MPEG standard [PM93] for still image data compression. In particular, we compare the application of two different verification techniques, namely model checking and term rewriting. We concentrate on two aspects which are in our opinion crucial for a wide acceptance of formal verification: (i) the appropriateness of the formal description of the system and (ii) the interactive human creativity required for the verification. Appropriateness of the system description is twofold: On the one hand, the formal model must contain enough details to capture the relevant behavior of the system such that all desired properties can be checked. On the other hand, the complexity of the formal description raises the costs of verification in both, human effort and run time. We consider interactive human creativity instead of pure execution runtimes consumed by the verification tools. Usually, the latter grows exponentially with the systems size, such that the automated part is either executed in a matter of seconds, or it is impossible.

Our two approaches to verify the DCT differ substantially with respect to the appropriateness criteria: While the term rewriting solution is based on an abstract data type that partially captures the axiomes the real numbers, the model checking solution is based on bitvectors with a fixed width. For this reason, the term rewriting solution allows a concise straightforward description of the mathematical definition of the DCT as well as different DCT algorithms at an abstract level. The human creativity required to check the correctness of the algorithm against the definition is limited to listing some trigonometric laws that have been used for optimizing the implementations (these can be viewed as the essential design ideas). On the other hand, effects as for instance arithmetic overflows are not captured at this abstract level.

The model checking approach is nearer to real implementations as fixed bit widths must be assumed and therefore problems such as arithmetic overflows or the limited precision are considered. However, the model checking approach suffers in general from a bad scalability, especially for data oriented systems as the DCT. Moreover, symbolic model checking based on BDDs [McM93] seems not to be well suited for the DCT verification since it is well known that BDDs [Bry86] lead to exponentially sized representations of the multiplication function. Multiplication is, however, heavily used in DCT algorithms. For the DCT, the problem can be overcome by abstraction techniques [Lon93] based on the Chinese Remainder Theorem. This however, requires advanced knowledge in model checking and therefore limits the degree of automation of this approach.

2 THE DISCRETE COSINE TRANSFORM

In this section, we briefly present the formal definition of the DCT to be able to explain afterwards what the verification problem is. For a more detailed presentation of the DCT and its application in the JPEG format see [PM93].

2.1 Formal Definition

One has to distinguish between the *one-dimensional* and the *two-dimensional* DCT. Given the input values $x_0, \dots, x_7 \in \mathbb{R}$, the one-dimensional DCT is defined as the following linear function $\Phi : \mathbb{R}^8 \to \mathbb{R}^8$:

$$\begin{pmatrix} y_0 \\ \vdots \\ y_7 \end{pmatrix} := \underbrace{\begin{pmatrix} a_{0,0} & \cdots & a_{0,7} \\ \vdots & \cdots & \vdots \\ a_{7,0} & \cdots & a_{7,7} \end{pmatrix}}_{=:\,\mathfrak{A}} \begin{pmatrix} x_0 \\ \vdots \\ x_7 \end{pmatrix}$$

where $a_{i,j} := \frac{1}{2}\cos\left((2j+1)i\frac{\pi}{16}\right)$ for $i > 0$ and $a_{0,j} := \frac{1}{2\sqrt{2}}$. It is important that the matrix $\mathfrak{A}$ is orthonormal, which means that its inverse matrix exists and is obtained by transposition, i.e. the coefficients $b_{i,j}$ of the inverse $\mathfrak{A}^{-1}$ are simply defined as $b_{i,j} := a_{j,i}$. Hence, the *inverse DCT* can be implemented very similar to the DCT.

The *two-dimensional DCT* is a transformation of a 8×8 matrix X of real number to a 8×8 matrix Y of real number. Formally, it is defined with the above matrix $\mathfrak{A}$ as $Y := \mathfrak{A}X\mathfrak{A}^{-1}$. The two-dimensional DCT is separable, i.e. it can be implemented by means of one-dimensional DCTs. Using an intermediate matrix $(u_{j,k})$, this separation is done as follows $u_{j,k} := \sum_{l=0}^{7} x_{k,l}\, a_{j,l}$ and $y_{i,j} := \sum_{k=0}^{7} u_{j,k}\, a_{i,k}$. For this reason, the two-dimensional DCT can be implemented by 16 one-dimensional DCTs (eight for the rows and eight for the columns). Although this does not lead to optimal results (see page 53 in [PM93]), there are very good algorithms for the one-dimensional DCT that lead to almost optimal results for the two-dimensional case. Hence, we completely focus on the one-dimensional DCT in the following.

2.2 Optimizations

The formal definition of the one-dimensional DCT requires 64 multiplications and 56 additions, which would be too much for practical use. However, the DCT can be

Common to all versions:

$$\begin{array}{llll}
L_{00} := x_0 + x_7 & L_{02} := x_2 + x_5 & L_{04} := x_3 - x_4 & L_{06} := x_1 - x_6 \\
L_{01} := x_1 + x_6 & L_{03} := x_3 + x_4 & L_{05} := x_2 - x_5 & L_{07} := x_0 - x_7
\end{array}$$

Simple Version (SIMP_DCT): [PM93]

$$\begin{array}{ll}
L_{10} := L_{00} + L_{03} & 2y_0 := c_4(L_{10} + L_{11}) \\
L_{11} := L_{01} + L_{02} & 2y_1 := c_1 L_{07} + c_3 L_{06} + c_5 L_{05} + c_7 L_{04} \\
L_{12} := L_{00} - L_{03} & 2y_2 := c_2 L_{12} + c_6 L_{13} \\
L_{13} := L_{01} - L_{02} & 2y_3 := c_3 L_{07} - c_7 L_{06} - c_1 L_{05} - c_5 L_{04} \\
 & 2y_4 := c_4(L_{10} - L_{11}) \\
 & 2y_5 := c_5 L_{07} - c_1 L_{06} + c_7 L_{05} + c_3 L_{04} \\
 & 2y_6 := c_6 L_{12} - c_2 L_{13} \\
 & 2y_7 := c_7 L_{07} - c_5 L_{06} + c_3 L_{05} - c_1 L_{04}
\end{array}$$

Loeffler, Ligtenberg and Moschytz (LLM_DCT) [LLM89]:

$$\begin{array}{lll}
L_{10} := L_{00} + L_{03} & L_{20} := L_{10} + L_{11} & z_0 := L_{20} \\
L_{11} := L_{01} + L_{02} & L_{21} := L_{10} - L_{11} & z_1 := L_{24} + L_{27} \\
L_{12} := L_{01} - L_{02} & L_{22} := \mathfrak{Rot}_0(L_{12}, L_{13}, 6) & z_2 := \sqrt{2}\, L_{22} \\
L_{13} := L_{00} - L_{03} & L_{23} := \mathfrak{Rot}_1(L_{12}, L_{13}, 6) & z_3 := \sqrt{2}\, L_{25} \\
L_{14} := \mathfrak{Rot}_0(L_{04}, L_{07}, 3) & L_{24} := L_{14} + L_{16} & z_4 := L_{21} \\
L_{15} := \mathfrak{Rot}_0(L_{05}, L_{06}, 1) & L_{25} := L_{17} - L_{15} & z_5 := \sqrt{2}\, L_{26} \\
L_{16} := \mathfrak{Rot}_1(L_{05}, L_{06}, 1) & L_{26} := L_{14} - L_{16} & z_6 := \sqrt{2}\, L_{23} \\
L_{17} := \mathfrak{Rot}_1(L_{04}, L_{07}, 3) & L_{27} := L_{15} + L_{17} & z_7 := L_{27} - L_{24}
\end{array}$$

Arai, Agui and Nakajiama (AAN_DCT): [PM93]

$$\begin{array}{lll}
L_{10} := L_{00} + L_{03} & L_{20} := c_4(L_{12} + L_{13}) & u_0 := L_{10} + L_{11} \\
L_{11} := L_{01} + L_{02} & & u_1 := L_{31} + L_{24} \\
L_{12} := L_{01} - L_{02} & L_{22} := q_0 L_{14} + L_{25} & u_2 := L_{20} + L_{13} \\
L_{13} := L_{00} - L_{03} & L_{23} := c_4 L_{15} & u_3 := L_{33} - L_{22} \\
L_{14} := L_{04} + L_{05} & L_{24} := q_1 L_{16} + L_{25} & u_4 := L_{10} - L_{11} \\
L_{15} := L_{05} + L_{06} & L_{25} := c_6(L_{14} - L_{16}) & u_5 := L_{33} + L_{22} \\
L_{16} := L_{06} + L_{07} & & u_6 := L_{13} - L_{20} \\
 & L_{31} := L_{07} + L_{23} & u_7 := L_{31} - L_{24} \\
q_0 := c_2 - c_6 & L_{33} := L_{07} - L_{23} & \\
q_1 := c_2 + c_6 & &
\end{array}$$

Figure 1 Optimized DCT algorithms

implemented more efficiently if occurrences of common subterms are exploited: Note first that all trigonometric values that occur in the matrix $\mathfrak{A}$ can be restricted to 8 basic values. To see this, we define the constants $c_k := \cos\left(k\frac{\pi}{16}\right)$ and $s_k := \sin\left(k\frac{\pi}{16}\right)$ for $k \in \mathbb{Z}$ (integer values). Clearly, the following relations hold:

$$\begin{array}{ll} (1)\ c_{-k} = c_k & (2)\ c_k = c_{k \bmod 32} \\ (3)\ c_k = c_{32-k} & (4)\ c_k = -c_{16-k} \end{array}$$

The above equations allow the reduction of the domain of k from $\mathbb{Z}$ to $\{k \mid k \geq 0\}$, $\{0,\ldots,31\}$, $\{0,\ldots,16\}$, and $\{0,\ldots,8\}$, respectively. Moreover, the special cases $c_0 = 1$, $c_4 = \frac{1}{2}\sqrt{2}$, $c_8 = 0$ can be used for optimizations. Clearly, also s_k is only interesting in the interval $\{0,\ldots,7\}$ and it can be reduced to c_k by the equation $s_k = c_{8-k}$. Considering all this, and computing common subterms only once, we obtain the implementation SIMP_DCT given in figure 1. This implementation requires only 22 multiplications and 28 additions.

Numerous efficient DCT implementations have been presented. The most efficient one, a true two-dimensional approach requires only 54 multiplications, 464 additions and 6 arithmetic shifts [FL90]. Some of the fast DCT algorithms go back to the discrete Fourier transform [CT65], others consider the DCT as a separate field [CSF77]. A survey is given in [DV90].

Figure 1 lists two efficient DCT algorithms, namely the version of Loeffler, Ligtenberg and Moschytz (LLM_DCT) [LLM89] and Arai, Agui and Nakajiama's version (AAN_DCT) [PM93]. The additional optimization obtained by these implementations is based on the following trigonometric addition theorems:

$$\begin{array}{l} \cos(x) + \cos(y) = 2\cos\left(\frac{x+y}{2}\right)\cos\left(\frac{x-y}{2}\right) \\ \cos(x) - \cos(y) = 2\sin\left(\frac{x+y}{2}\right)\sin\left(\frac{x-y}{2}\right) \end{array}$$

Using these theorems, additional common subterms can be generated that can then be shared. The LLM_DCT makes additionally use of a rotation operation that is defined with an angle α as follows:

$$\begin{pmatrix} y_0 \\ y_1 \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}$$

As written above, the rotation operation would require four multiplications and two additions. Rotations are however computed with an intermediate value ℓ as given in figure 2. This requires three multiplications and three additions. Hence, we decrease the number of multiplications by increasing the number of additions which is reasonable as multiplications are in general more expensive. For figure 1, we define $\mathfrak{Rot}_0(x_0, x_1, \alpha) := x_0 c_\alpha + x_1 s_\alpha$ and $\mathfrak{Rot}_1(x_0, x_1, \alpha) := -x_0 s_\alpha + x_1 c_\alpha$, but we assume that the algorithm of figure 2 is used for the computation. Hence, LLM_DCT requires only 13 multiplications and 29 additions. AAN_DCT does even only require five multi-

$$\begin{array}{l} \ell := \cos(\alpha)(x_0 + x_1) \\ y_0 := \ell + (\sin(\alpha) - \cos(\alpha))x_1 \\ y_0 := -(\sin(\alpha) + \cos(\alpha))x_0 + \ell \end{array}$$

Figure 2: Rotations with 3 Multiplications

plications and 28 additions.

Both LLM_DCT and AAN_DCT produce scaled results (these are sufficient for a lot of applications as e.g. JPEG): For the variables defined in figure 1, the following equations hold: $z_i = 4c_4y_i = 2\sqrt{2}y_i$, $u_0 = 4c_4y_0 = 2\sqrt{2}y_0$ and $u_{i+1} = 4c_{i+1}y_{i+1}$.

3 VERIFYING DCT ALGORITHMS BY TERM REWRITING

Laws for $\mathbb{R}$
$x + 0 := x$
$x * 1 := x$
$x * 0 := 0$
$x * (y + z) := (x * y) + (x * z)$
$x + -(x) := 0$
$x * -(y) := -(x * y)$
$-(-(x)) := x$
$\sqrt{2} * \sqrt{2} * x := x + x$

Figure 3: Axioms for $\mathbb{R}$

In this section, we show how the correctness of the DCT algorithms of figure 1 is proved with the term rewriting system RRL [KZ88]. For this reason, we establish a canonical term rewrite system that handles real numbers $\mathbb{R}$ and the trigonometric functions. Therefore, some axioms on real numbers and the trigonometric functions have to be added to the definition of the DCTs. The laws for $\mathbb{R}$ are given in figure 3. Moreover, the operators $+$ and $*$ are specified to be associative and commutative. The above laws essentially specify that the real numbers are mathematically speaking a field. Note however, that we do neither need the ordering relation nor the supremum axiom of $\mathbb{R}$ and hence, have not characterized the real numbers completely.

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

Figure 4: Addition Laws

For the verification of LLM_DCT, we also need some instances of the trigonometric addition laws. These are given in figure 4. Moreover, we need the relationship between $\sqrt{2}$ and the trigonometric functions: $c_4 := \frac{1}{2}\sqrt{2}$ and $c_4 * \sqrt{2} := 1$. Using all these equations, the relationship $z_i = 2\sqrt{2}\ y_i$ can be easily proved by means of RRL.

This is done as follows: We first enter all rewrite equations apart from the above addition theorems and establish a confluent rewrite system by means of the Knuth-Bendix completion procedure. After that, we add the above addition theorems and do not invoke a further completion (this does not work with our ordering). Instead the rewrite system is yet powerful enough to make the proofs with simple rewriting (using the boolean ring method).

For the verification of AAN_DCT, we need more instances of the trigonometric addition laws, which are given below. The verification runs then in the same lines as for LLM_DCT: We first enter all rewrite equations apart from the addition theorems and establish a confluent rewrite system by completion. After that, we add the addition theorems below and obtain the proofs with simple rewriting (using the boolean ring method). The proof of the relationship with SIMP_DCT, i.e. the equations $u_0 = 4c_4y_0 = 2\sqrt{2}y_0$ and $u_{i+1} = 4c_{i+1}y_{i+1}$ follows then automatically.

Addition Laws used for AAN_DCT		
$\sqrt{2} * c_6 := c_2 - c_6$	$\sqrt{2} * c_2 := c_2 + c_6$	$c_6 * c_2 := \frac{1}{2}c_4$
$c_1 * c_3 := \frac{1}{2}(c_2 + c_4)$	$c_1 * c_5 := \frac{1}{2}(c_6 + c_4)$	$c_1 * c_7 := \frac{1}{2}c_6$
$c_3 * c_7 := \frac{1}{2}(c_4 - c_6)$	$c_3 * c_5 := \frac{1}{2}c_2$	$c_5 * c_7 := \frac{1}{2}(c_2 - c_4)$
$c_2 * c_2 := \frac{1}{2}(1 + c_4)$	$c_6 * c_6 := \frac{1}{2}(1 - c_4)$	$c_1 * c_1 := \frac{1}{2}(1 + c_2)$
$c_3 * c_3 := \frac{1}{2}(1 + c_6)$	$c_5 * c_5 := \frac{1}{2}(1 - c_6)$	$c_7 * c_7 := \frac{1}{2}(1 - c_2)$

4 VERIFYING INTEGER DCT ALGORITHMS

The definition of the DCT is based on real numbers. However, all implementations, regardless whether they are done in hardware or software, do only work on floating point or fixpoint numbers, i.e. on approximations of the real numbers with a limited precision. For this reason, it might be the case that some of the above optimized DCT algorithms do compute slightly different outputs than others.

Often special fixpoint implementations are used for the DCT that are called integer DCTs: These are obtained by replacing the real valued cosine constants c_i by the integers $\tilde{c}_i := \lfloor 2^n c_i \rfloor$ when n bits are available. As the inputs x_i are also given as integers, only integer operations are then needed.

$$\begin{aligned}
&(1)\ qc_1 = c_3 + c_5\\
&(2)\ qc_3 = c_1 + c_7\\
&(3)\ qc_5 = c_1 - c_7\\
&(4)\ qc_7 = c_3 - c_5
\end{aligned}$$

Figure 5: Equation System for LLM_DCT

The verification based on term rewriting presented in the last section did not fix the data domain. The obtained results hold for any data domain with operations $+, -$ and $\cdot$ that has constants $\frac{1}{2}, \sqrt{2}, 0, 1, c_1, c_2, c_3, c_4, c_5, c_6, c_7$ such that the presented equations hold. Hence, if the equations given in figure 5 hold for a specific data domain, then the equivalence between LLM_DCT and SIMP_DCT holds for this domain.

$c_1 = \sqrt{2}c_5 + c_7$	$c_1 = -\sqrt{2}c_5 + c_7$
$c_3 = c_5 + \sqrt{2}c_7$	$c_3 = c_5 - \sqrt{2}c_7$
$q = \sqrt{2}$	$q = -\sqrt{2}$

Figure 6: Solution of the Equations of figure 5

So, let us first consider the solutions of the equation system of figure 5. Apart from the trivial solution where all constants equal to zero, there are infinitely may solutions: c_5 and c_7 can be chosen arbitrarily and the other constants are then determined as given in figure 6. As in particular $q = \pm\sqrt{2}$ holds, there is no integer solution of the equation system. For this reason, the results given in the previous section that are based on term rewriting do not hold for the integer variant of LLM_DCT. The same holds for AAN_DCT.

5 VERIFYING INTEGER DCTS BY MODEL CHECKING

As the proof from the abstract level cannot be transferred to the bitvector level we tried to establish term rewrite systems for integer versions based on bitvector arithmetic. However, this approach turned out to become too complicated such that we dropped this approach.

Instead, we propose to use tautology checking based on BDDs for the verification at the integer level. Our first attempt to this was quite depressing: By a direct invocation of the SMV model checker, we could only handle implementations up to three bits. Even the word-level extension of SMV could not manage implementations with more than four bits.

To circumvent the complexity, we used a reduction technique that is based on the Chinese Remainder Theorem (CRT) that is given below. The CRT is the basis for residue arithmetic that is roughly explained as follows: Given relatively prime numbers p_1,..., p_n, i.e. numbers with $\mathsf{GCD}\,(p_i, p_j) = 1$, each integer x in the interval $[0, \prod_{i=1}^{n} p_i[$ can be uniquely represented as the list of its residues, i.e. $(x \bmod p_1, \ldots, x \bmod p_n)$ (residue number representation). Arithmetic operations are then simply performed by applying the corresponding operation on the residues: Given that x and y have the residue number representations $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$, then $x + y$, $x - y$, and xy, have the residue number representation $(x_1 + y_1, \ldots, x_n + y_n)$, $(x_1 - y_1, \ldots, x_n - y_n)$, and $(x_1y_1, \ldots, x_ny_n)$, respectively.

Theorem 1 (Chinese Remainder Theorem) *Given integers r_1,..., r_n, b and relatively prime integers p_1,..., p_n with $0 \leq r_i < p_i$ then, there is a unique integer x with $b \leq x < \prod_{i=1}^{n} p_i$ such that $x \bmod p_i = r_i$.*

The advantage of residue arithmetic is that it can be performed in parallel as the arithmetic operations on the single residues are independent of each other. This independence is also useful for the verification as it allows to split the entire problem into a couple of smaller ones: Instead of verifying the equivalence of two b bit implementations, we use some prime numbers p_1,..., p_n such that $2^b < \prod_{i=1}^{n} p_i$ and verify the equivalence of the two versions modulo p_i. As the latter implementation has only $\lfloor log_2(p_i) \rfloor + 1$ bits, it can be checked much faster.

For example, using the prime numbers 7, 11, 13, 17, 19, 23, 25, 27, 29, 31, 32, we can prove for LLM_DCT that $z_i = 4c_4y_i$ holds for $i = 0, 2, 3, 4, 5, 7$. Also, we can show that $z_1 \neq 4c_4y_1$ and $z_6 \neq 4c_4y_6$ holds for some inputs which is due to the limited precision of the integer approximation[1].

The experimental results that we have obtained with the SMV system for proving $z_0 = 4c_4y_0$ are given in figure 7 (Sun UltraSparc with 128MByte Memory). The runtime given in figure 7 is thereby the sum of the runtimes for checking the equivalence modulo all p_i's. For small bit widths we used however only the necessary prime numbers, e.g. for 8 bits, we only used the prime numbers 7, 11, 13, 17, 19, 23 as $2^{2 \cdot 8+3} = 524288 < 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 = 7436429$ (note that the DCT with b bit wide inputs yields in $2^{2 \cdot b+3}$ wide outputs).

6 CONCLUSIONS

We have illustrated the use of term rewriting and model checking for the verification of real number algorithms like the DCT. Some errors in the presentation of algorithms in the literature [PM93] we found prove that the DCT as an embedded system is a non-trivial example that requires formal verification.

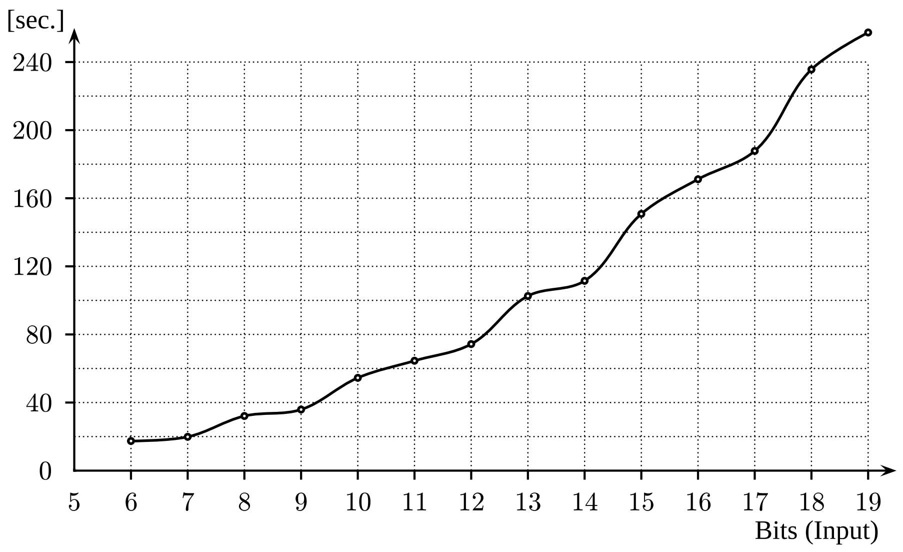

Figure 7 Experimental Results for Model Checking with CRT abstraction

Without doubt, model checking is currently the most popular verification technique for concurrent systems. However, the case study presented in this paper shows that term rewriting is more appropriate for the DCT since it provides a higher degree of abstraction that is often desired in early design phases.

Nevertheless, at the bitvector level where the limited preciseness of the implementation of the real numbers has to be taken into account, model checking seems to be more adequate. But already for small bit widths the systems are far too big to be checked by a direct attempt. To overcome these limits we applied the Chinese Remainder Technique as an appropriate abstraction technique to verify greater bit widths by model checking. The Chinese Remainder Technique is easily applicable by using predefined parameterized modules for residue arithmetic. However, none of the currently available model checkers offers such a feature. Thus, it would be desirable to extend these tools by such abstraction methods.

The Chinese Remainder Abstraction as applied here is only suited for proving or disproving equivalence of two implementations. In particular, it is not possible to verify that certain error bounds hold, since this would require to compare numbers in their residue number representation which is not possible. Our future work will therefore aim at extending the Chinese Remainder Abstraction by an efficient conversion back to the usual radix number representation.

Notes

1. Note that we can only prove that the equations $z_1 = 4c_4y_1$ and $z_6 = 4c_4y_6$ do not hold. It is not possible to prove error bounds $|z_1 - 4c_4y_1| \leq \varepsilon_1$ and $|z_6 - 4c_4y_6| \leq \varepsilon_6$ by the CRT abstraction since it is not possible to compare the size of numbers given in residue number representation.

References

[ABC+94] A. Aziz, F. Balarin, S.-T. Cheng, R. Hojati, T. Kam, S.C. Krishnan, R.K. Ranjan, T.R. Shiple, V. Singhal, S. Tasiran, H.-Y. Wang, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. HSIS: A BDD-Based Environment for Formal Verification. In *ACM/IEEE Design Automation Conference (DAC)*, San Diego, CA, June 1994. San Diego Convention Center.

[Bry86] R.E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[CSF77] W. Chen, C.H. Smith, and S.C. Fralick. A fast computational algorithm for the discrete cosine transform. *IEEE Transactions on Communication*, 25(9):1004–1009, 1977.

[CT65] J.W. Cooley and J.W. Tukey. An algorithm for the machine calculations of complex Fourier series. *Mathemtical Computation*, 19:297–301, 1965.

[DV90] P. Duhamel and M. Vetterli. Fast fourier transforms: A tutorial review and a state of the art. *Signal Processing*, 19:259–299, 1990.

[FL90] E. Feig and E. Linzer. Discrete cosine transform algorithms for image data compression. In *Proceedings Electronic Imaging'90 East*, pages 84–87, Boston, MA, 1990.

[FSS+94] T. Filkorn, H. A. Schneider, A. Scholz, A. Strasser, and P. Warkentin. *SVE User's Guide*. Siemens AG, TR ZFE BT SE 1-SVE-1, Munich, 1994.

[GM93] M.J.C. Gordon and T.F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.

[Gup92] A. Gupta. Formal Hardware Verification Methods: A Survey. *Journal of Formal Methods in System Design*, 1:151–238, 1992.

[HP96] G. J. Holzmann and D. Peled. The state of SPIN. In Rajeev Alur and Thomas A. Henzinger, editors, *Conference on Computer Aided Verification (CAV)*, volume 1102 of *Lecture Notes in Computer Science*, pages 385–389, New Brunswick, NJ, USA, July/August 1996. Springer Verlag.

[KZ88] 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.

[LLM89] 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.

[Lon93] D.E. Long. *Model Checking, Abstraction, and Compositional Verification*. PhD thesis, Carnegie Mellon University, 1993.

[McM93] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Norwell Massachusetts, 1993.

[ORR+96] S. Owre, S. Rajan, J. M. Rushby, N. Shankar, and M. K. Srivas. PVS: Combining specification, proof checking, and model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *Conference on Computer Aided Verification (CAV)*, volume 1102 of *Lecture Notes in Computer Science*, pages 411–414, New Brunswick, NJ, USA, July/August 1996. Springer Verlag.

[PM93] W.B. Pennebaker and J.L. Mitchell. *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, ISBN 0-442-01272-1, 1993.