System Level Verification of Digital Signal Processing Applications Based on the Polynomial Abstraction Technique

Tarvo Raudvere, Ashish Kumar Singh, Ingo Sander and Axel Jantsch
Royal Institute of Technology
Stockholm, Sweden
{tarvo,ashish,ingo,axel}@imit.kth.se

Abstract

Polynomial abstraction has been developed for data abstraction of sequential circuits, where the functionality can be expressed as polynomials. The method, based on the fundamental theorem of algebra, abstracts a possibly infinite domain of input values, into a much smaller and finite one, whose size is calculated according to the degree of the respective polynomial. The abstract model preserves the system's control and data properties, which can be verified by model checking. Experiments show that our approach does not only allow an automatic verification, but also gives considerably better results than existing methods.

1. Introduction

Abstraction techniques are used in order to avoid the well known state space explosion problem that model checking suffers from. This paper introduces the *polynomial abstraction* technique that is a modification of *spatial abstraction* [6]. Spatial abstraction separates the control part from the data path of a design and based on the assumption about the correctness of the data path, it allows to reduce the data path so that the behaviors of the control part are preserved. This reduces the state space and allows to verify the correctness of the control part and data path predicates with a model checking tool.

In comparison to spatial abstraction, our method based on polynomial abstraction allows to verify the system functionality as a mix of data and control properties, i.e. it can be verified that a system output is a function of the system inputs.

Our technique is applicable at a high abstraction level where a design is described in terms of infinite data types, ideal computation and storage units, while abstracting from techniques like saturation arithmetic and fixed-point. The method reduces the domains of data signals of a sequential design block, whose functionality can be expressed as a polynomial or rational function. Based on the fundamental theorem of algebra even an infinite domain can be replaced with a finite one, where the number of values is determined by the degree of the respective polynomial. An additional reduction of the model state space is achieved through the application of the Chinese remainder theorem [2].

2. Related Work

Various abstraction techniques are proposed for simplification of model checking tasks. The idea of uninterpreted function symbols is used to make the verification task smaller. In [1] an out-of-order processor is verified through uninterpreted functions, which allows to show the correctness of the machine independently of the actual instruction set architecture and the implementation of the functional units. Since this technique is based partly on theorem proving, it needs a remarkable amount of designer contribution. A disadvantage of the technique is that it does not allow to improve the state space reduction from the distributive laws of arithmetic operations. Also it may lead to a false negative result that often involves with abstraction.

Hojati and Brayton [5] describe a methodology for integer combinational/sequential systems. According to the notion of data independence they separate a design into control part and data path. All the data path variables are replaced with binary variables.

Spatial abstraction [6] combines this idea with the interval propagation theory [4]. Through this technique the bit widths of the data signals can be reduced so that all the possible behaviors of the control part are preserved. A signal that does not determine the control flow is classified as a data signal. These signals are initialized with one bit variables. Through interval propagation [4] their bit widths are calculated until a fix-point or the actual range of the variable is achieved. All other signals (control) are irreducible.

The disadvantages of the method are: (1) the domains of the variables must be determined before abstraction, (2) the abstract model is valid only for verification of the properties related to the control part and interconnecting signals between data path and control part. Unfortunately the data properties based on the system functionality given in the specification are not verifiable. Polynomial abstraction allows to verify the implementation model against the system specification. And the domains of data signals in the design may be undetermined.

Polynomial methods have been applied for verification also at lower abstraction levels. In [9] a method for component reuse based on matching an arithmetic specification with bit-level implementations is introduced. Our technique targets design descriptions where the word sizes of an implementation are not yet specified.

Compared to our technique where proper ranges of variables for verification are found according to their degrees, in [7] Pnueli et. al. describe a method to analyze the structure of an equality formula including source and target models, and to determine ranges of the variables. Similar to polynomial abstraction it is possible to verify systems at an abstraction level, where the domains of variables are unspecified. Unfortunately their approach is sensitive to arithmetic optimization and thus may not give correct answer about equality of the source and target models. Our method does not lead to false negative answers about system correctness.

A method for verification of combinational circuits, that consists of the functions $\{+,-,*\}$ is introduced in [8]. This study finds the order of a functional implementation and uses the simulation with a restricted set of input vectors to verify the system correctness. Compared to polynomial abstraction, they do not address sequential designs and rational functions.

Clarke et. al. use in [2] the Chinese remainder theorem to reduce the verification task of integer arithmetic functions. Although it simplifies the task significantly, it is valid only for a finite input domain. Our abstraction technique makes it possible to verify a system with undetermined input domains through a bounded set of input vectors. Thus we can additionally apply the remainder theorem in order to reduce the number of input vectors even more.

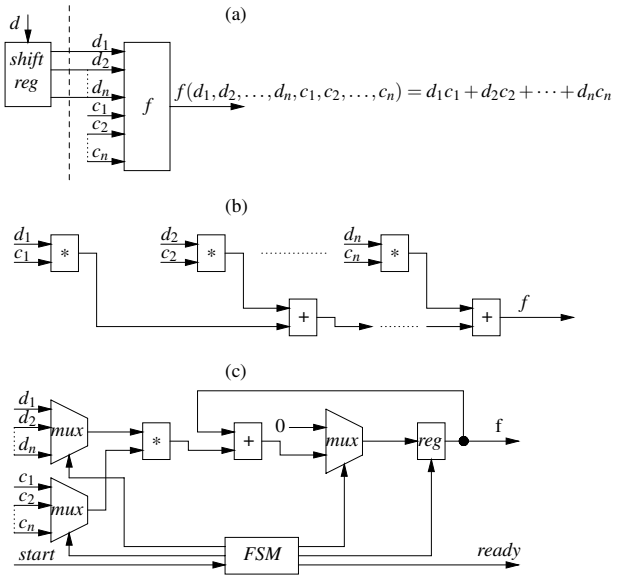


Figure 1. a) Specification, b) Intermediate Model and c) Implementation

3. Polynomial Abstraction

3.1 Scope and Target Designs

The design process of many *digital signal processing* applications starts at a high abstraction level, where the system specification is described as a combinational function in terms of ideal data types. It is natural to start at that level, since it abstracts from lower-level implementation details, and allows to concentrate on the system functionality. The specification model may have an implicit mapping to arithmetic and logic operations, denoted as an intermediate model. The direct implementation of this model as a combinational circuit is often impractical, if the model contains a large number of identical computation units. An improvement can be achieved through component reuse. This can be performed through a design transformation, which refines the intermediate model into a sequential implementation model.

As an example, a clock domain refinement of the order-n FIR filter is shown in Figure 1, which transforms the combinational function f of the filter into an implementation as a circuit with data path and control part. The data input of the filter is labeled with d, and d_i denotes the i-th cycle delayed input value. The filter coefficients are denoted by c_i. The execution of the FSM is triggered by the input *start* and the completion of the calculation is announced by the signal *ready*. The FSM is configured so that in the first cycle the register is initialized with a constant value 0 from the multiplexer and in the following cycles the sum of multiplications $d_i * c_i$ is calculated. In order to simplify the verification of a sequential implementation against the combinational specification data abstraction has to be applied.

Polynomial abstraction is applicable on sequential designs, whose functionality can be expressed as a polynomial or a rational function. The design (Figure 2) may contain boolean, enumerated, integer and real number signals. There is a restriction for real number input signals; these are allowed only in the case, if they do not determine any computation path in the controller. The design may contain logic operations, and arithmetic operations, which are based on the operations: $+$, $-$, $*$ and $/$. The abstraction technique assumes that the functionality of logic and arithmetic operators are correct or verified separately.

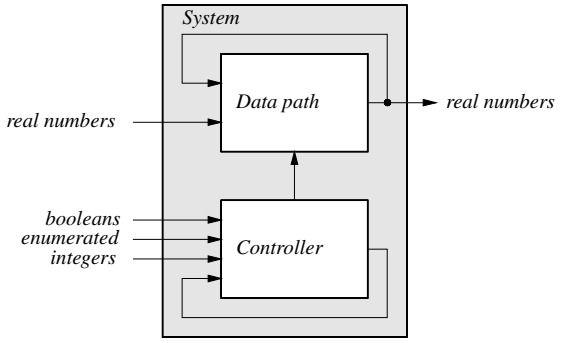


Figure 2. System structure

3.2 Roadmap of the Technique

The roadmap of the abstraction technique is presented in Figure 3. The starting point of the algorithm is a sequential design. In the first step the system inputs are classified as data or control that specifies if the domain of a signal is reducible or not. The next task is to find the degrees of input variables in the output polynomial, which the system calculates. These degrees will be used to determine reduced domains for the input signals in an abstract model. If the design contains any division operator, the design is

mapped to a fractional model, in order to avoid real number signals. The abstraction flow continuous with the application of an additional abstraction technique, which relies on the Chinese remainder theorem. Finally we translate the abstract model into the input language of a model checker - in our case into the SMV language [10].

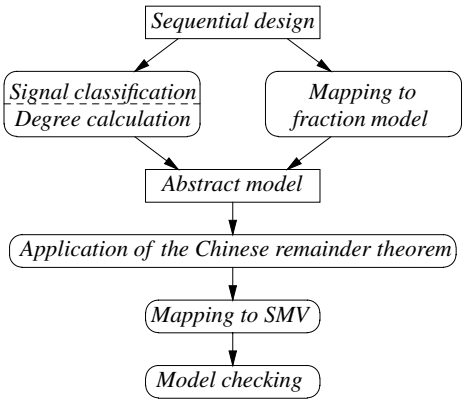


Figure 3. Roadmap of abstraction

One possibility to verify systems like in Figure 1.c is to use symbolic execution, which finds the output function as a polynomial in symbolic variables. The correctness of the refinement can be verified through equivalence checking of the corresponding specification and implementation polynomials. Unfortunately this polynomial presentation does not contain all the behaviors of the sequential circuit. Let's consider the FIR filter in Figure 1. In this case the verification based on equivalence checking of polynomials does not cover multiple executions in sequence. For instance: (1) does the FSM properly turn back to the initial state; or (2) does the output value stay correct if the next *start* signal appears at the same time when an announcement about completion of the last calculation is issued? This kind of properties can be verified by model checking, but the technique is not proper for targeted designs with infinite input domains, since model checkers can handle only finite models.

We address the latter problem by introducing polynomial abstraction. The technique is based on the fundamental theorem of algebra [3]. Using this theorem we have proven that two degree k uni-variable polynomials are equivalent if they evaluate to pairwise the same values for $k+1$ different input assignments. We have extended this for multi-variable polynomials and proven the corresponding theorem. The proofs are given in Appendix A. Based on the latter theorem, we can decide about the equivalence of two n-variable polynomials $P(x_0,...,x_n)$ and $Q(x_0,...,x_n)$, through assigning to them all the possible input vectors $\langle x_0,...,x_n\rangle$, where the domain of the input x_i contains $k+1$ different values if the degree of x_i in P and Q is k.

The given theoretical background allows us to reduce the infinite domains of the input signals to finite ones, and in this way makes it possible to use a model checker for verifying an implementation against a specification. Implicitly the method can be applied also to the systems with finite input domains in order to reduce the state space and thus to simplify the model checking task.

Since the symbolic execution is memory and time consuming and the exact implementation polynomial is not required to determine the degrees of the input variables, we only statically analyze the circuit in order to calculate the maximum degrees.

To be precise we explain some notations used in the paper. The number of clock cycles needed for a single input assignment to calculate the function in the implementation model, is denoted as an *execution cycle*.

Let $\overline{x}$ denote the set of variables $x_0,...,x_n$,i.e., $P(x_0,...,x_n) = P(\overline{x})$. The function $\mathcal{D}(P(\overline{x}),x_i)$ gives the maximum degree of x_i in $P(\overline{x})$. The maximum degree of multiplication of two polynomials is calculated as follows: $\mathcal{D}(P(\overline{x})Q(\overline{x}),x_i) = \mathcal{D}(P(\overline{x}),x_i) + \mathcal{D}(Q(\overline{x}),x_i)$.

3.3 Signal Classification

According to their tasks we classify the signals as *data signals* or *control signals*. If a signal steers a process to take one of many computation branches then the signal is a control signal. In other words all signals, which appear in the conditional part of an *if* or *case* expression or in a *pattern matching* construction as a pattern are control signals. Also all binary and scalar signals are control signals. If an output of a process is a control signal then all the input signals of the process are regarded as control signals as well. Clearly the last definition is recursive. Finally all the signals, which are not marked as control signals are data signals. The domains of the data signals can be reduced by the abstraction technique. The domains of the control signals will not be modified in order to preserve all the possible behaviors of the controller. If the design contains any control input with an infinite domain, then the designer has to specify a finite domain for that input signal.

3.4 Maximum Degree Calculation

We have developed a tool that finds the degrees of the output polynomial of an implementation through analyzing the design for one execution cycle. If the design has control inputs, then it calculates the degrees of a data variable for any input combinations of the control values. The domain of an input in the abstract model will be determined according to the maximum degree of this input.

Since designs may contain division operations, we represent polynomials and their maximum degrees in fractional form through this section. For a polynomial $\frac{P(\overline{x})}{Q(\overline{x})}$ the maximum degree fraction looks like $\frac{\langle \nu_{x_0},...,\nu_{x_n}\rangle}{\langle \delta_{x_0},...,\delta_{x_n}\rangle}$, where ν_{x_i} and δ_{x_i} are the maximum degrees of the input variable x_i in the numerator and denominator polynomials respectively. For example consider a data path with two data inputs x_1 and x_2. If some internal process in the data path calculates the function $\mathcal{G} = \frac{x_1^3x_2^2+2x_1}{x_1x_2+7}$ then the respective maximum degree fraction of the process output signal is $\frac{\langle 3,2\rangle}{\langle 1,1\rangle}$, since the maximum degrees of x_1 and x_2 in the numerator polynomial are three and two respectively. And the maximum degrees of x_1 and x_2 in the denominate polynomial are both one.

The degree of any variable of a system input is one, thus the maximum degree fraction of a data input variable x_i is: $\frac{\langle \nu_{x_0},\ldots,\nu_{x_{i-1}},\nu_{x_i},\nu_{x_{i+1}},\ldots,\nu_{x_n}\rangle}{\langle \delta_{x_0},\ldots,\delta_{x_{i-1}},\delta_{x_i},\delta_{x_{i+1}},\ldots,\delta_{x_n}\rangle} = \frac{\langle 0,\ldots,0,1,0,\ldots,0\rangle}{\langle 0,\ldots,0,0,0,\ldots,0\rangle}$. The rules for the calculation of the maximum degree for arithmetic operations are given in Table 1.

Table 1. Rules for the Calculation of the Maximum Degree

Maximum Degree of x_i
$\mathcal{D}((\frac{P(\overline{x})}{Q(\overline{x})}+\frac{R(\overline{x})}{S(\overline{x})}),x_i) = \frac{\max(\mathcal{D}(P(\overline{x})S(\overline{x}),x_i),\mathcal{D}(R(\overline{x})Q(\overline{x}),x_i))}{\mathcal{D}(Q(\overline{x})S(\overline{x}),x_i)}$
$\mathcal{D}((\frac{P(\overline{x})}{Q(\overline{x})}-\frac{R(\overline{x})}{S(\overline{x})}),x_i) = \frac{\max(\mathcal{D}(P(\overline{x})S(\overline{x}),x_i),\mathcal{D}(R(\overline{x})Q(\overline{x}),x_i))}{\mathcal{D}(Q(\overline{x})S(\overline{x}),x_i)}$
$\mathcal{D}(((\frac{P(\overline{x})}{Q(\overline{x})})(\frac{R(\overline{x})}{S(\overline{x})})),x_i) = \frac{\mathcal{D}(P(\overline{x})R(\overline{x}),x_i)}{\mathcal{D}(Q(\overline{x})S(\overline{x}),x_i)}$
$\mathcal{D}((\frac{P(\overline{x})}{Q(\overline{x})}/\frac{R(\overline{x})}{S(\overline{x})}),x_i) = \frac{\mathcal{D}(P(\overline{x})S(\overline{x}),x_i)}{\mathcal{D}(Q(\overline{x})R(\overline{x}),x_i)}$

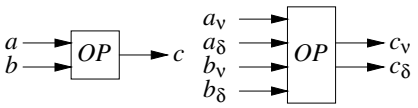
Let's consider the degree calculation for the output signal f in Figure 1.c. Let the fraction be in the form $\frac{\langle \nu_{d_1},\nu_{c_1},\nu_{d_2},\nu_{c_2},\ldots,\nu_{d_n},\nu_{c_n}\rangle}{\langle \delta_{d_1},\delta_{c_1},\delta_{d_2},\delta_{c_2},\ldots,\delta_{d_n},\delta_{c_n}\rangle}$. In the initial state the fraction contains only zeros. In the first execution cycle the register is initialized with constant 0, which do not change the degree fraction. In the second cycle the circuit calculates the operation $d_1 * c_1 + f$. At that moment the degrees corresponding to d_1, c_1 and f are $\frac{\langle 1,0,0,0,\ldots,0,0\rangle}{\langle 0,0,0,0,\ldots,0,0\rangle}$, $\frac{\langle 0,1,0,0,\ldots,0,0\rangle}{\langle 0,0,0,0,\ldots,0,0\rangle}$ and $\frac{\langle 0,0,0,0,\ldots,0,0\rangle}{\langle 0,0,0,0,\ldots,0,0\rangle}$ respectively. According to the degree calculation rules the new fraction of f is $\frac{\langle 1,1,0,0,\ldots,0,0\rangle}{\langle 0,0,0,0,\ldots,0,0\rangle}$. In the third cycle the operation $d_2 * c_2 + f$ increases the values of ν_{d_2} and ν_{c_2} by one. In the end of the execution cycle, when the signal *ready* goes high, the numerator vector contains only ones and the denominator vector contains only zeros.

Since the degree calculation for a sequential circuit is done step by step based on the degrees instead of the exact functions in the previous steps, then the simplification of a degree fraction through reducing the respective degrees ν_i and δ_i is not allowed. For example we cannot change the fraction $\frac{\langle 3,2\rangle}{\langle 1,1\rangle}$ to $\frac{\langle 2,1\rangle}{\langle 0,0\rangle}$. Although this simplification is valid for the function $\frac{x_1^3x_2^2}{x_1x_2}$, it does not hold for the function $\mathcal{G} = \frac{x_1^3x_2^2+2x_1}{x_1x_2+7}$. Thus we actually find the upper bounds of the degrees.

If the output polynomial according to the specification is $\frac{P(\overline{x})}{Q(\overline{x})}$ and the implementation functionality can be expressed as $\frac{R(\overline{x})}{S(\overline{x})}$, then based on the mathematical rule $\frac{a}{b} = \frac{c}{d} \Rightarrow ad = bc$, we calculate the final maximum degree for the input variable x_i as $\max(\mathcal{D}(P(\overline{x})S(\overline{x}),x_i),\mathcal{D}(Q(\overline{x})R(\overline{x}),x_i))$.

3.5 Mapping to the Fractional Model

As common for model checkers, the SMV tool supports only boolean, enumerated and integer variables. Although the abstract domains of the input signals can be defined such that they contain only integers, a division operation may still give a real number result. Due to this reason we map the design to the fraction model. In this model all signals (s) related to arithmetic operations are presented as a pair of signals (s_ν ,s_δ), so that $s = \frac{s_\nu}{s_\delta}$. For this purpose all arithmetic operations have to be replaced as shown in Figure 4. Since none of the operations in the new model does include explicit division, all the internal and output signals belong to the domain of integers.



OP	c_ν	c_δ
$c = a+b$	$a_\nu * b_\delta + b_\nu * a_\delta$	$a_\delta * b_\delta$
$c = a-b$	$a_\nu * b_\delta - b_\nu * a_\delta$	$a_\delta * b_\delta$
$c = a*b$	$a_\nu * b_\nu$	$a_\delta * b_\delta$
$c = a/b$	$a_\nu * b_\delta$	$b_\nu * a_\delta$

Figure 4. Operations in fractional model

Additionally to the arithmetic operations also all registers, multiplexors and other logic units, which are connected with data signals have to be modeled as a pair, where one unit is used for numerator and the other for the denominator signal.

3.6 Application of the Chinese Remainder Theorem

Based on the arithmetic identity presented in Equation 1 the Chinese remainder theorem can be used to simplify the verification task of arithmetic circuits [2]. Let the index I denote implementation and the index S specification variables.

$$((a_I \bmod m_i)\ OP_j\ (b_I \bmod m_i)) \bmod m_i \equiv (a_S\ OP_j\ b_S) \bmod m_i \mid OP_j \in \{+,-,*\} \text{ and } m_i \in Pos.Int \text{ and } a,b \in Int \quad (1)$$

If Equation 1 holds for a set of numbers, which are relatively prime numbers $M = \{m_1, m_2, \ldots, m_n\}$ then $(a_I\ OP\ b_I) \equiv (a_S\ OP\ b_S)$ holds for the integer domain $0 \le (a\ OP\ b) \le \prod_{i=1}^{n} m_i$, $m_i \in M$. According to this knowledge, the system verification for the domain $\{0,\ldots,\prod_{i=1}^{n} m_i\}$ can be replaced with n verifications for the domains $\{0,\ldots,m_i\}$. In order to perform the series of verification we have to change the assignment statements *reg_value* := *input_value* to *reg_value* := (*input_value mod* m_i).

Let's consider an arbitrary design, which contains a calculation of the value x^{10}. Polynomial abstraction makes it possible to verify the design such that the input x gets integer assignments from 0 to 10. Although the domain is small, we need 33 bits to present the value 10^{10}, as $10^{10} \approx 2^{33}$. Instead of verifying directly this model, we can apply the described modulo computation method, to verify eleven much smaller models, since the multiplication of the first eleven prime numbers $\{2,3,5,\ldots,31\}$ is approximately 2^{37}, that is greater than 2^{33}. Although we need to verify several models, the size of these are significantly smaller, e.g., to present the largest prime number in the set we need only 5 bits ($31 < 2^5$) instead of 33 bits.

3.7 Model Checking

We translate the specification model and the implementation model into the SMV language [10]. According to the maximum degrees of the input variables in the output polynomial, that have been found previously, we specify new domains for all the data signals. If the maximum degree of the signal s_i is k then the model checker can assign values from 0 to k to that input. The domains of the control signals are unchangeable. The model checker verifies, that if the input values $\langle\overline{x}\rangle$ of specification $(\frac{S_v(\overline{x})}{S_\delta(\overline{x})})$ and implementation $(\frac{I_v(\overline{x})}{I_\delta(\overline{x})})$ are the same then after an execution cycle the output values are equal as well ,i.e., $S_v(\overline{x})I_\delta(\overline{x}) = S_\delta(\overline{x})I_v(\overline{x})$.

4. Case Studies

The efficiency of our methodology is illustrated with several case studies, where we have verified the sequential implementations of well-known digital signal processing applications. The time and the number of BDD nodes required for verification are presented in Table 2 for some examples. The experiments are done on a Sun machine with 900MHz CPU and 1GB RAM, using the Cadence SMV tool [10].

Let's consider the verification of the order 16 FIR filter, which structure is shown in Figure 1.c. The circuit has 33 inputs - 16 data inputs (d_i), 16 coefficients (c_i) that can vary in time, and the signal *start*. According to the signal classification the signals d_i and c_i are data signals, since they do not determine any computation path in the model. According to the proposed degree calculation algorithm, the degree of these signals is one. Thus in the abstract model the signals get assignments from 0 to 1, instead of the initial infinite date types. This corresponds to one bit variables, which are the smallest possible.

In a similar way we have verified an IIR filter, cosines function, an FFT application and Sigma-Delta-Demodulator. These are examples of applications, which can be found almost in every design that contains signal processing, e.g. today's audio, video and communication devices.

Table 2. Verification Time and BDD Nodes

Design	*Function*	*Time sec.*	*BDD− nodes*
FIR filter	$\sum_{i=1}^{16} d_i * c_i$	29.4	1.4*M*
IIR filter	$\sum_{i=0}^{7}(d_i * b_i - d_i * a_i * b_0)$	2.5	458*k*
DFT	*8 point FFT*	1.5	91*k*
Cosines	$\sum_{i=0}^{4} \frac{x^{2*i}}{(2*i)!}$	9.5	153*k*
ΣΔ− *Demodulator*	*Includes order 4 CIC and order 16 FIR filters*	67.1	3.8*M*

In order to compare our abstraction technique to the spatial abstraction [6], we have selected the 8-bit repetitive multiplier that is the only design presented in a detailed way in [6]. The multiplier has two input and one output registers. The multiplication is performed through decrement operation of one register and addition of the value in the other register to the output register in every clock cycle. The computation is finished when the value in the first register is equal to zero. The initial model has 67 state variables and through polynomial abstraction the number of states can be reduced to 17 (Table 3). In comparison with the spatial abstraction, which reduces the number of states to 38, our method gives a significant improvement, since in a rough estimation the number of BDD nodes and the amount of time required for model checking grows exponential with the number of states.

Table 3. Polynomial Abstraction versus Spatial Abstraction

	Number of states	*Reduction*
Original model	67	–
Model through spatial abstraction	38	43.3%
Model through polynomial abstraction	17	74.6%

In contrast to the methods of Pnueli [7] and Berezin [1] that use the advantages of uninterpreted functions and can verify larger circuits, our approach allows to verify design implementations on which arithmetic and logic optimization has been applied. This is possible, since our abstraction technique preserves the system's actual functionality instead of mapping it to an uninterpreted one.

The technique of Smith and Micheli [9] can handle only designs with fixed bit-widths and therefore has a limitation compared with our polynomial method that is applicable in an earlier design phase.

5. Conclusion

Based on the fundamental theorem of algebra and the Chinese remainder theorem we have proposed an approach that enables to verify efficiently sequential designs whose functionality can be expressed as a polynomial or rational function, such as DSP applications and VLSI implementations of cryptographical algorithms. Experiments show that polynomial abstraction can be efficiently applied at a high abstraction level.

In comparison with the existing methods, our technique has many advantages. It is not sensitive to logic and arithmetic optimization, and accordingly does not give false answers about system's correctness. It allows to verify designs at a high abstraction level, where the bit-widths of variables are not yet decided. The abstraction process preserves the data and control properties of an initial design, which can be verified by model checking.

6. References

[1] S. Berezin, A. Biere, E. M. Clarke, and Y. Zhu. Combining symbolic model checking with uninterpreted functions for out-of-order processor verification. In *Formal Methods in Computer-Aided Design*, pages 369–386, 1998.

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

[3] J. E. Eaton. The fundamental theorem of algebra. *American Mathematical Monthly*, 67(6):578–579, 1960.

[4] E. Hansen. A generalized interval arithmetic. *Lecture Notes in Computer Science 29*, 1975.

[5] R. Hojati and R. K. Brayton. Automatic datapath abstraction in hardware systems. *Lecture Notes in Computer Science 939*, 1995.

[6] V. Paruthi, N. Mansouri, and R. Vemuri. Automatic data path abstraction for verification of large scale designs. In *ICCD '98 Topic : Verification and Test*, 1998.

[7] A. Pnueli, Y. Rodeh, O. Strichmann, and M. Siegel. The small model property: how small can it be? *Information and Computation*, 178(1):279–293, 2002.

[8] P. Sanchez and S. Dey. Simulation-based system-level verification using polynomials. In *IEEE International High Level Design Validation and Test Workshop (HLDVT'99)*, November 1999.

[9] J. Smith and G. D. Micheli. Polynomial methods for component matching and verification. In *Proceedings of the ACMIEEE International Conference on Computer-Aided Design*, pages 678–685, San Jose, California, USA, 1998.

[10] The SMV model checker. online [available] http://www-cad.eecs.berkeley.edu/~kenmcmil/smv/.

APPENDIX

A. Theorem about Polynomials Identity

Let $\mathbb{N}$ denote the set of values $\{0,1,2,...\}$, and let the set $\mathbb{N}^i$ consist of all possible tuples with i components from the set $\mathbb{N}$. The notation $[i,j]$ denotes the set of integers between i and j including i,j.

LEMMA 1. *Let $P(x)$ be a uni-variable polynomial of degree k. If $P(x)$ is equal to 0 for $(k+1)$ distinct values then all the coefficients of $P(x)$ are zero.*

Proof: Lemma 1 is based on a standard result from *the fundamental theorem of algebra* [3]. The theorem states that a uni-variable polynomial of degree k has exactly k complex roots[1] unless all of its coefficients are zero. Since integers are a special case of the complex numbers, the number of integer roots of a polynomial cannot exceed the degree of the polynomial unless all the coefficients are 0. This establishes the statement of the lemma. □

THEOREM 1. *Two polynomials $Q_1(x_1,\ldots,x_i)$ and $Q_2(x_1,\ldots,x_i)$ are identical if and only if for all the tuples $(y_1,\ldots,y_i) \in \mathbb{N}^i$ the following condition is satisfied:*

$$\forall j(1 \le j \le i), \quad y_j \in [0, max(\mathcal{D}(Q_1(\overline{x}),x_j), \mathcal{D}(Q_2(\overline{x}),x_j))] \Rightarrow Q_1(y_1,\ldots,y_i) = Q_2(y_1,\ldots,y_i) \tag{2}$$

[1] A value α is a root of the polynomial P(x) if P(α) = 0

Proof: The 'only if' part of the theorem is straightforward since if $Q_1(\overline{x})$ and $Q_2(\overline{x})$ are identical then they will always evaluate to the same value for the same input assignment. Hence Condition 2 is satisfied. The 'if' part claims that $Q_1(\overline{x})$ and $Q_2(\overline{x})$ are identical assuming that Condition 2 holds. Let $C_{Q_1}^{e_1,\ldots,e_i}$ and $C_{Q_2}^{e_1,\ldots,e_i}$ denote the coefficients of the term $x_1^{e_1}\ldots x_i^{e_i}$ in $Q_1(\overline{x})$ and $Q_2(\overline{x})$ respectively. The difference polynomial of $Q_1(\overline{x})$ and $Q_2(\overline{x})$ is denoted as $R(\overline{x})$,

$$R(x_1,\ldots,x_i) = Q_1(x_1,\ldots,x_i) - Q_2(x_1,\ldots,x_i) \tag{3}$$

The coefficient of the term $x_1^{e_1}\ldots x_i^{e_i}$ in $R(\overline{x})$ is $C_R^{e_1,\ldots,e_i} = C_{Q_1}^{e_1,\ldots,e_i} - C_{Q_2}^{e_1,\ldots,e_i}$. In order to prove the identity of $Q_1(\overline{x})$ and $Q_2(\overline{x})$ we must show that all the coefficients of the polynomial $R(\overline{x})$ are zero. We observe that the degree of a variable x_j in the polynomial $R(\overline{x})$ is bounded by the maximum degree of x_j in $Q_1(\overline{x})$ and $Q_2(\overline{x})$ e.g.

$$\mathcal{D}(R(\overline{x}),x_j) \le max(\mathcal{D}(Q_1(\overline{x}),x_j), \mathcal{D}(Q_2(\overline{x}),x_j)) \tag{4}$$

We can translate Condition 2 into the following condition over R using (4):

$$R(\overline{y}) = 0, \forall (y_1,\ldots,y_i) \in \mathbb{N}^i \text{ and } y_j \in [0, \mathcal{D}(R(\overline{x}),x_j)] \tag{5}$$

We give an inductive proof on the number of variables i for the following *statement*: if Condition 5 holds then all the coefficients of $R(\overline{x})$ are zero.

Base Case: If $i = 1$ then Condition 5 implies that uni-variable polynomial $R(\overline{x})$ has at least $m+1$ distinct integer roots, where $m = \mathcal{D}(R(\overline{x}),x_1)$. In this case the proof follows directly from the statement of Lemma 1.

Induction Step: We assume the correctness of the statements for $i-1$ variables and based on this induction hypothesis we prove that the statement is true for i variables as well. We group together the terms of $R(x_1,\ldots,x_i)$ which have the same exponent of the variable x_1. We can re-write such a group of terms as a multiplication of a power of x_1 and a polynomial in the variables $x_2,\ldots,x_i$. Thus we can re-write $R(x_1,\ldots,x_i)$ in the following form, where $m = \mathcal{D}(R(\overline{x}),x_1)$.

$$R(x_1,\ldots,x_i) = x_1^m H_m(x_2,\ldots,x_i) + x_1^{m-1} H_{m-1}(x_2,\ldots,x_i) + \ldots + H_0(x_2,\ldots,x_i) \tag{6}$$

$$\mathcal{D}(H_k(\overline{x}),x_j) \le \mathcal{D}(R(\overline{x}),x_j),\ (2 \le j \le i) \text{ and } (0 \le k \le m) \tag{7}$$

For each assignment of the tuple values $(p_2,\ldots p_i) \in \mathbb{N}^{i-1}$ to $(x_2,\ldots,x_i)$ such that

$$0 \le p_j \le \mathcal{D}(R(\overline{x}),x_j) \tag{8}$$

we can evaluate each of the polynomials H_k in (6). Thus we get a polynomial in one variable x_1 on the right hand side of (6). By the application of Condition 5 this polynomial evaluates to 0 for the integer range $0 \le x_1 \le (\mathcal{D}(R(\overline{x}),x_1) = m)$. Since the degree of the polynomial is m and it has at least $m+1$ integer roots we can apply Lemma 1 to deduce that $H_k(p_2,\ldots,p_i) = 0$ for all k such that $(2 \le k \le i)$. The same procedure can be repeated for any $(p_2,\ldots,p_i)$ satisfying Condition 8. Thus we can conclude that $H_k(x_2,\ldots,x_i) = 0$ if $x_j \in [0, \mathcal{D}(R(\overline{x}),x_j)]$. By the application of Condition 7 we deduce that $H_k(x_2,\ldots,x_i) = 0$ if $x_j \in [0, \mathcal{D}(H_k(\overline{x}),x_j)]$. Hence each of the coefficients of the polynomial H_k should be zero on applying the induction hypothesis. Using (6) we deduce that each of the coefficients of the polynomial $R(\overline{x})$ must also be zero.

□