Optimizations in the Verification Technique of Automatic Assertion Checking with Non-linear Solver

***Abstract*— This paper presents some optimizations of a verification technique based on non-linear solvers. The optimized solver is able to automatically check assertions in behavioral descriptions of hardware systems. These descriptions are modeled with a set of integer polynomial inequalities. The techniques have been evaluated with real electronic systems, such as Viterbi decoders or vocoder digital filters.**

***Index Terms*—Assertion-based-Verification (ABV), non-linear solver, property checking.**

I. Introduction

According to the 2004 report of the International Technology Roadmap for Semiconductors [1], Verification has become the main bottleneck of the design flow as a result of two processes. Firstly, the functional complexity of modern designs is continuously growing. Secondly, the greater emphasis on other aspects of the design process has produced important progress (automated tools for logic synthesis, place-and-route, etc), leaving verification as the main bottleneck that will be a barrier to further progress in the semiconductor industry if there is not a major breakthrough.

Formal verification techniques are beginning to gain acceptance and they sometimes complement simulation methods in the process of verification. The main goal of formal hardware verification is to prove the functional correctness of a design instead of simulating some vectors. Most of the formal verification methodologies use Boolean equations to model some aspects of the design.

Popular techniques to solve these Boolean equation systems (or satisfiability problems) are based on Binary Decision Diagrams (BDD) [2]. BDDs are used to represent binary output value constraints in a canonical form. The main disadvantage of the use of BDDs is the "memory explosion" problem because of the huge size of the diagram even for medium complexity designs. Several optimizations have been proposed to compress the diagram (OBDD, ROBDD, etc).

Another way to solve Boolean equations is to use a SAT solver. This technique avoids the exponential space blow-up of BDD [3]. The main drawback is the handing of arithmetic operators. These operators are transformed into a large number of Boolean formulas which reduce the SAT efficiency and limit its application domain [11]. To overcome these disadvantages, hybrid satisfiability approaches, such as HSAT [4], have been proposed. The goal is to combine a SAT and a linear programming solver. The SAT checker is used to solve the logic equations and the linear programming solver is used to check the feasibility of the arithmetic equations. These two engines operate in separate domains. The performance of HSAT is limited by the heuristics that choose the set of assignments to Boolean variables. Other similar approaches (e.g. LPSAT [5]) are based on mixed integer linear programming (MILP) techniques [5]. However, general ILP solvers tend to be inefficient in solving real satisfiability problems. Firstly, they do not directly handle nonlinear operators (multipliers). Secondly, they have numerical convergence problems, and they are sensitive to a number of internal parameters. Other tools are based on Constraint Logic Programming (CLP) techniques [6]. The CLP works at Boolean level and/or Integer domain and it has similar problems to MILP techniques.

Non-linear solvers make it possible handle behavioral descriptions with non-linear expressions (multiplier operation) without transforming them into linear expressions. This advantage of the non-linear solvers is well known in handling complex systems [10]. Starting from this point, this paper presents different optimizations of a verification technique based on a commercial global non-linear solver [7]. These optimizations modify the polynomial model that the solver has to solve and study different selection algorithms to find the integer point (counter-example). These optimizations improve the efficient handling of non-linear systems and the CPU requirements.

II. System Modeling

The hardware system is described at behavioral level as a set of concurrent processes. The proposed verification technique is focused on individual process validation, thus only one process will be considered. This process is suspended in an initial wait statement until the input values

This work has been partially supported by the Spanish MCYT through the TIC-2005-03301.

change. After this, the process body is executed until the initial statement (wait statement) is reached (reactive system). Figure 1 shows this behavior.

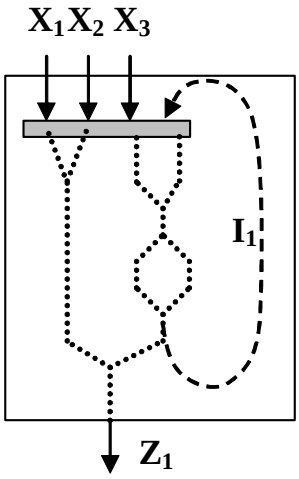

Figure 1: System Model.

The straight arrows model the external inputs (Xi) and outputs (Zi). The gray box represents the 'wait' statement and the dashed line the memories or state variables (Ii). The dotted lines represent the execution paths (functionality) of the process. The model includes integer variables and the directly supported operators are addition, subtraction, multiplication and relational (Figure 2). Other operators have to be transformed into equivalent polynomial equation systems (modulus operation and so on).

reg[7:0] A, B,	$A, B \in [-128,127]$
reg[8:0] C, D,	$C, D \in [-256,255]$
reg[15:0] E,	$E \in [-16384,16383]$
$C = A - B,$	$C = A - B;$
$D = A + B,$	$D = A + B;$
$E = A \times B,$	$E = A \times B;$

Figure 2. Example of Basic Operator Modeling.

Other operators (e.g. bit selection, bit-wise logic operator, etc) are transformed in a similar way.

Word-level logic operators (e.g. "or_reduce") and bit-level logic operations are transformed into integer polynomials. For example, the logic equation "a = b or c" is transformed into "a = b + c - (b*c).

It is assumed that all the previous equations take integer values.

Concerning control statements, conditional 'if' statements are totally supported. This sentence splits the execution flow into two paths (True and False paths). These paths are modeled with an additional variable whose values are '0' and '1'. This variable enables the true path with the value '1' and disables the false path and vice versa. Figure 3 shows an example and figure 4, the polynomial model. If 'K=1', the expression 'K' selects the true path and the expression '1-K' disables the false path. In the other case, when K=0, the 'K' expression disables the true path and the '1-K' expression enables the false path.

In order to model the expression of the 'if' conditional statement, an aspect has to be considered. The value space that goes through the false path is complementary to the true space. In figure 3, the true expression is '$(5y - x) > 0$' and the false expression is '$(5y - x) \leq 0$'. The last expression is transformed into '$(x - 5y) \geq 0$' → '$(x - 5y + 1) > 0$'. And finally they are weighted with the 'K' and '1 – K' expressions and summed (expression 'ret' of figure 4).

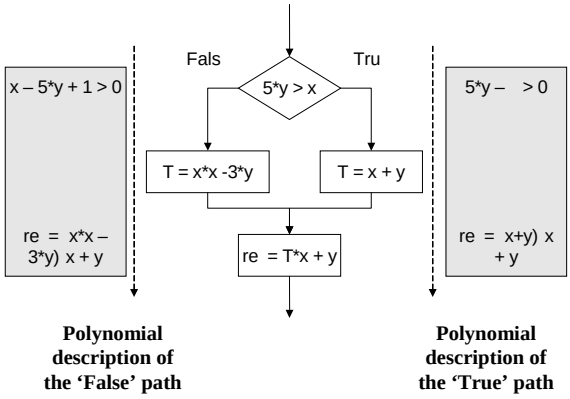

Figure 3. Polynomial description of a simple example

$$K(5y - x) + (1\text{-}K)(x - 5y + 1) > 0$$
$$ret = K\{(x + y)x + y\} + (1 - K)\{(x^2 - 3y)x + y\}$$
$$K \in [0,1]$$

Figure 4. Model of the conditional statement.

Finally, the loop operators are handled with restrictions. The 'for' loops are totally unrolled when the number of iterations can be statically determined. The 'while' loops cannot normally be totally unrolled because it is not possible to statically determine the number of iterations. In this case, the verification algorithm will unroll a new iteration in every step. This means that the algorithm will unroll one iteration in the first step, two in the second and it will repeat the process up to a user-defined maximum number of iterations. If several 'while' loops are nested, the number of unrolled statements will grow exponentially.

With the previously commented transformation, the process body (dotted line in Figure 1) will be modeled with polynomials whose external input values will change in every process execution. The assertion to be checked and the conditional statements will be modeled with polynomial inequalities.

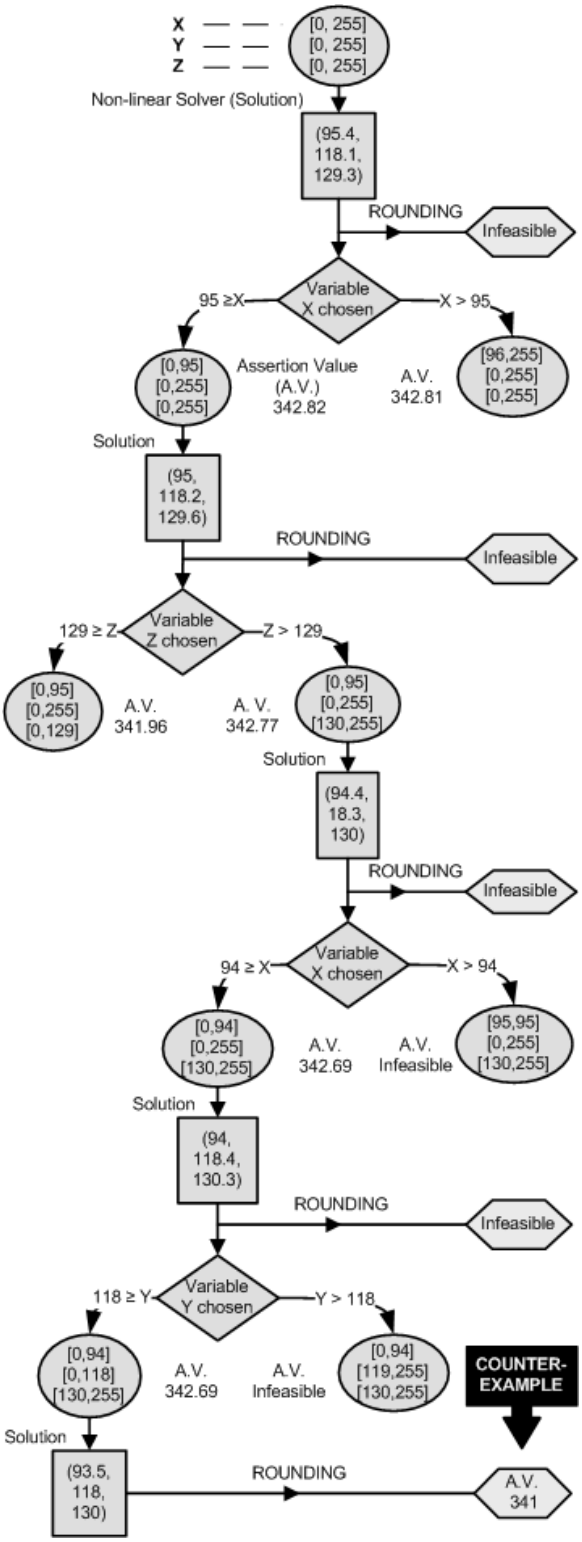

Figure 8: Steps of the algorithm to find the integer solution in the 'space3' example.

III. System Modeling Example

In this section, the generation of the polynomial model of a simple example (space3) is presented. This set of polynomial equations can be solved by a commercial global non-linear solver.

The process body has three external inputs, x,y and z. They are integers with range 0 to 255. Figure 5 presents the behavioral description of the 'space3' on the top. On the bottom, it shows the equivalent polynomial model of the system.

IV. Verification Methodology

The goal of the proposed verification technique [10] is to find a point that fulfills the set of integer inequalities that model the hardware system and violates an assertion. Three steps have been defined (Figure 6):

// BEHAVIORAL MODEL

```
void space3 (uint8 x, uint8 y, uint8 z)
{
    int temp, dat, ret;
    dat = (x – 110)^2 – (y – 28)^2;
    temp = dat – (z – 170)^2;
    if (10000 > temp)
        if (6*y – 2*x – 4*z > 0)
            ret = x + y + z;
    else
        ret = 0;
    ….
    Assertion(ret ≤ 340); // Assertion to check
}
```

// POLYNOMIAL MODEL

$dat = (x - 110)^2 + (y - 28)^2;$
$temp = dat + (z - 170)^2;$
$10000 - temp > 0;$
$6*y - 2*x - 4*z > 0;$
$ret = x + y + z;$
$ret > 340;$

Figure 5: Example 'space3'.

1.- Polynomial model generation

The behavioral description is transformed into an inequality system that can be handled mathematically.

2.- Solve the inequalities system

A non-linear solver is used to find a solution in the real domain. If there is a real solution, an algorithm that finds an integer solution has to be applied (step 3). If there is no real solution and the input description has "while" statements, a new iteration of a 'while' loop will be added to the polynomial system description before executing step 2 again.

3.- Derive an integer solution from the real solution

The goal is to find an integer solution taking into account the information that the real-domain solution provides. The technique defines two steps: variable rounding and branch-and-bound exploration of the solution space.

The first step is to round the real variables to the closest integer value. If there are 2 possible values (for example,

11.50 could be rounded to 11 or 12), a value will be randomly selected. Figure 8 presents the decisions that the integer-solution search algorithm takes with the 'space3' example. The inserted assertion verifies that the 'ret' variable is never greater than 340. In Figure 8, the ranges of the inputs are included in ellipses. The rectangles contain the solutions that the solver provides in the real domain, the hexagons, the solution of the rounded input points and the rhombuses, the variable that the "selection algorithm" reports.

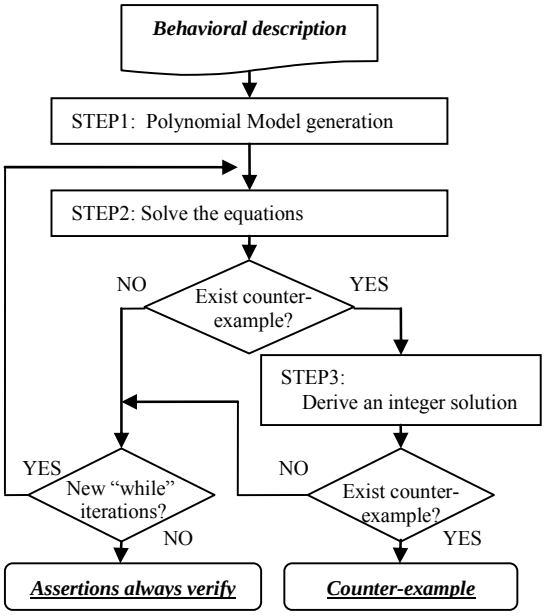

Figure 6: Verification Methodology.

The non-linear solver provides a first solution that is rounded by the searcher algorithm to an infeasible solution (the assertion is not violated or the inequalities are not fulfilled).

In this case, the second step (branch-and-bound based exploration) is applied. Firstly the selection algorithm (rhombus) is applied to decide the variable that splits the input space. Several selection algorithms are explained in section V.2. In Figure 8, the selected variable is x. Secondly, the input space of the selected variable is split into two parts: values greater than the integer part (x>95) of the solution and values less than or equal to the integer part (x<=95). This generates two new sets of polynomial inequalities. These sets are solved with the non-linear solver, thus two new sets of solutions and maximum values of the assertion (A.V.) can be generated. If a new set has no solution, its branch will be removed. The algorithm will select the set that produces a higher assertion value and it will repeat the searching process. This process will be finished in a branch if one of these conditions is verified:

1.- The solver cannot find a solution, thus the problem is infeasible.

2.- The solver provides a solution, but the assertion is always verified.

In these cases, the current branch will be removed and the last unselected branch will be selected. This process is repeated until a counterexample is found or all the branches are removed (the assertion is always fulfilled).

V. Optimizations Of The Verification Technique

Three optimizations of the verification methodology have been studied. One is applied in the generation of the polynomial model (step 1) and the others, in step 3 (deriving an integer solution).

A. Preprocessing of the polynomial model

The first optimization consists in transforming the polynomial model into a new extended set of simpler expressions. This helps the solver to find the optimized point. The transformation splits each complex inequality into two simpler inequalities: one with positive monomials and another, with negative monomials (see Figure 7).

$P = x^2 + 3*y*z^2 - t^2*z - 292*t^3$
$P^+ = x^2 + 3*y*z^2$ $P^- = -t^2*z - 292*t^3$ $P = P^+ + P^-$

Figure 7: Transformation of polynomials.

These polynomials are always monotonic increasing (P+) and decreasing (P–) in the positive area (all variables are greater than or equal to zero). In the other areas, these properties are not fulfilled. In this special area, the advantage is that the derivative is always increasing or decreasing. This preprocess adds new variables and constraints to the model: one variable and one constraint for each constraint that is transformed.

This increasing of the constraints and variables achieves worse results if the number of new variables is considerable compared to the total number of variables and the original expressions are simple (linear polynomials) and therefore, the complexity differences are similar between both models. The results are very notable in models with positive variables and complex expressions. Therefore, this optimization is applied to complex models and, especially, to polynomials with positive variables or negative variables. This decision is justified by the results of section VI.

B. Selection algorithms

The second and the third optimization are applied in the "Derive an integer solution" step (step 3). Starting from the continuous solution that the solver provides, the algorithms choose the dimension (variable) to split the input space and remove the real solution of the space of searching. Depending on the selected variable, the branch-and-bound exploration can be more efficient and it could need less computation resources. Two algorithms have been studied.

C. Gradient-based selection

The algorithm uses the partial derivative of the objective function to choose the next variable to split. The objective function is the polynomial expression of the assertion. The partial derivatives are calculated in the real solution point. The partial derivative enables the definition of a set of linear equations that approximate the behavior of the objective function in the area close to the real solution. This set of linear functions allows estimating the objective function when a variable is rounded to an integer value. The variable that produces the maximum variation is chosen. This decision produces the maximum variation of the objective function.

D. Selection based on maximum error

This algorithm depends only on the real solution that is provided by the solver. This algorithm is the simplest because it does not use the objective function. The variable, with the fractional part closest to .5, is chosen. For example, the point (158.43, 78.75, 64.21) have the (0.07, 0.25, 0.29) differences with the middle point (158.5, 78.5, 64.5). Therefore, the first value is closest to middle value and then, the first dimension is the dimension to split the input space.

VI. EXPERIMENTAL RESULTS

In order to validate the optimizations of the verification technique, three examples of behavioral level descriptions have been selected. The first is a data-dominated example [8]. The number of possible input values is 2564 and the number of points that violate the assertion is only two. The example has 6 constraints and the maximum order of the polynomials is 4. The second example is a Viterbi decoder algorithm [9]. This is a soft decoder with a rate of ½, a constraint length of 3 and a survivor window length of 16. The inserted assertion checks if there is overflow in the maximum value of the path metric accumulator. Finally, the third example is the 'Pre-Process' module of the GSM standard (ETSI EN 301.245, December 1997). This module is a second order high pass IIR digital filter with cut off frequency at 80 Hz and 4 taps. The assertion verifies that the accumulated values are not saturated.

The CPU times in the Tables correspond to seconds on a Pentium IV with 2 GB of RAM at 2.8 GHz under Windows XP.

The first optimization is evaluated for the three examples (Table 1) and compared with different tools. The first row is the time, in seconds, that a SAT tool (Berkmin [12]) needs to verify the system. The second row (Integer solver) shows the time that an integer solver (LINGO) needs to provide a solution. And the last two rows show the time that the proposed technique needs to solve the problem (without the preprocess optimization and with the optimization). Some cases have no time (OFL – Out oF Limit) because the program was aborted or it did not have enough resources or the time was greater than 24 hours.

The first example (Viterbi) has linear expressions and variables with range [0, 255]. The optimization reduces the execution time by 8 seconds, 5% of 158 seconds. The second example (IIR Filter) has linear expressions and variables with range [-32768, 32767]. The time is worse in the model with optimization. This example without optimization is composed of 22 iterations. The first iteration is a simple model with 3 variables and the last iteration is a complex model with 170 variables. The partial results are better in the optimized model when the description has few variables. But this difference is reversed when the description is more complex. The total time of the optimized model is a little worse. The third example has non-linear expressions and variables with positive ranges [0, 255]. In this case, the reduced time is near to 30%. The solver needs 65% of the total number of iterations that it requires to solve for the original description.

This optimization provides better results when the description has complex expressions (non-linear) and the values of the variables are either positive or negative.

		Viterbi	IIR Filter	Space4
SAT		2486	OFL	36
ILP		OFL	OFL	4
Our approach	without opt.	158	448	3
	with opt.	150	504	2.1

Table 1: Experimental results of the first optimization.

The others optimizations are evaluated with the 'space4' example. In order to carry out a better comparison between both selected algorithms, several versions of the example with different objective function have been proposed. 'Space4-0' is the original example and the other rows of the Table 1 are different versions (from 'space4-1' to 'space4-5'). All versions maintain the same properties as the original example. Table 2 shows the number of times that the proposed algorithms are applied to find the integer point (#Exe) and the total execution time in seconds (Time). The 'difference' column is the difference between 'Gradient-based' algorithm and 'Maximum error' algorithm.

	Gradient-based		**Maximum error**		**Difference**	
	Time	#Exe	Time	#Exe	Time	#Exe
Space4-0	10	6	9	3	1	3
Space4-1	14	12	8	5	6	7
Space4-2	15	17	17	21	-2	-4
Space4-3	7	2	5	1	2	1
Space4-4	15	22	10	5	5	17
Space4-5	14	9	14	11	0	-2
Total	**75**	**68**	**63**	**46**	**12**	**22**
Average	**12.5**	**11.3**	**10.5**	**7.7**	**2**	**3.6**

Table 2: Transformation of polynomials.

The temporal results have more dispersion in the second selection algorithm but the execution time is 33% better on average than the gradient-based algorithm.

VII. CONCLUSIONS

This paper presents several optimizations of the verification methodology based on a non-linear solver. These optimizations improve the capabilities and performance of the non-linear solver. One of the main optimizations is transforming the expressions of the polynomial model into simpler expressions. This improvement is greater with complex expressions and with variables that have only one sign (either positive or negative). The improvement in 'space4' is nearly 30%. The other optimizations affect the number of times that the verification algorithm is executed and, therefore, the execution time. The improvement of the execution time of the second algorithm is on average 33%.

REFERENCES

[1] "The International Technology Roadmap For Semiconductor". 2004 Edition. Design. http://www.itrs.net/Common/2004Update/2004_01_Design.pdf
[2] R.E.Bryant, "Graph Based Algorithms for Boolean Function Manipulation", IEEE Transactions on Computers, vol. C-35, pp. 677-691, August 1986.
[3] A. Biere, A.Cimatti, E.M.Clarke, M. Fujita, Y. Zhu, "Symbolic Model Checking Using SAT procedures instead of BDDs*". Proc. Of DAC'99. 1999.
[4] F. Fallah, S. Devadas, and K. Keutzer, "Functional Vector Generation for HDL models using Linear Programming and 3-Satisfiability Infrastructure using the Unite Recursive Paradigm", in Proc. Of DATE 2000, 2000, pp. 232 – 236.
[5] Z. Zeng, P.Kalla, and M. Ciesielski, "LPSAT: A unified approach to rtl satisfiability", in Proc. DATE, March 2001, pp. 398-402.
[6] Zeng Z., Ciesielski M., and Rouzeyere B., "Functional test generation using constraint logic programming", in VLSI-SOC Conference, 2001.
[7] LINDO API., www.lindo.com, Lindo Systems Inc.
[8] Reference omitted.
[9] John P. Elliot, "Understanding Behavioral Synthesis. A Practical Guide to High-Level Design", Kluwer Academic Publishers, 2000.
[10] Reference omitted.
[11] Ganapathy Parthasarathy, Madhu K. Iyer, Kwang-Ting (Tim) Cheng, and Li-C. Wang, (March-April 2004) "Safety Property Verification Using Sequential SAT and Bounded Model Checking", IEEE Design&Test of Computers 132-143.
[12] E. Goldberg, Y. Novikov, "BerkMin: a Fast and Robust Sat-Solver", DATE'02. 2002.