Finite Precision Bit-width Allocation using SAT-Modulo Theory

Adam B. Kinsman and Nicola Nicolici
Department of Electrical and Computer Engineering
McMaster University, Hamilton,ON L8S 4K1,Canada
kinsmaab@mcmaster.ca, nicola@ece.mcmaster.ca

Abstract

This paper explores the use of SAT-Modulo Theory in determination of bit-widths for finite precision implementation of numerical calculations, specifically in the context of scientific computing where division frequently occurs. Employing SAT-Modulo Theory leads to more accurate bounds estimation than those provided by other analytical methods, in turn yielding smaller bit-widths.

1 Introduction

Approximation of real numbers with infinite range and precision by a finite set of numbers in a digital computer gives rise to approximation error which is managed in two fundamental ways. This leads to two fundamental representations: floating-point and fixed-point which limit (over their range) the relative and absolute approximation error respectively. The suitability of limited relative error in practice, which also enables representation of a much larger dynamic range of values than fixed-point of the same bit-width, makes floating-point a favorable choice for many numerical applications. Because of this, double precision floating-point arithmetic units have for a long time been included in general purpose computers, biasing software implementations toward this format. This positive feedback loop between floating-point software applications and dedicated floating-point hardware units has produced a body of almost exclusively floating-point scientific software.

Although transistor scaling and architectural innovation have enabled increased power for the above mentioned general purpose computing platforms, advances in field programmable gate array (FPGA) technology have significantly tightened the performance gap between FPGAs and application specific integrated circuits (ASICs). This level of performance without the cost associated with producing an ASIC has stimulated interest in *reconfigurable hardware acceleration* platforms (see [14] for a comprehensive survey of architectures and design methods). These platforms stand to provide greater computational power than general purpose computing, a feat accomplished by tailoring hardware calculation units to the application, and exploiting parallelism by replication in FPGAs. Notable examples of applications presently undergoing active research include molecular dynamics simulations [12] and computational fluid dynamics [11].

To leverage the FPGA parallelism, calculation units should use as few resources as possible, standing in contrast to (resource demanding) floating-point implementations used in reference software. As mentioned above however, the use of floating-point in the software results mostly from having access to floating-point hardware rather than out of necessity of a high degree of precision. Thus by moving to a reduced yet sufficient precision floating-point or fixed-point scheme, a more resource efficient implementation can be used leading to increased parallelism and with it higher computational throughput.

This problem is significant in the context of architectural synthesis as it directly influences the cost of the functional units which are to be employed. By reducing bit-width, all elements of the data-path will be reduced, most importantly memories, as a single bit saved reduces the cost of every single address. Furthermore, propagation delay of arithmetic units is often tied to bit-width, as well as latency in the case of sequential units (e.g. sequential multiplier or divider).

2 Related Work / Motivation

In this section, we first overview the main existing methods for bit-width allocation, then provide a motivational example highlighting challenges faced by these methods.

Discovering the minimum number of bits necessary to accurately represent an intermediate variable from a calculation is a two part problem. Both the range and precision required must be determined, from which can be inferred the required number of *exponent* and *mantissa* bits in floating-point, or *integer* and *fraction* bits in fixed-point. A large volume of work exists targeted at determining range and precision in the context of both digital signal processing (DSP) and embedded systems, a good summary of which is provided by [6]. Existing solutions can be classified into one of two main categories: analytical (formal) and simulation based (empirical).

978-3-9810801-5-5/DATE09 © 2009 EDAA

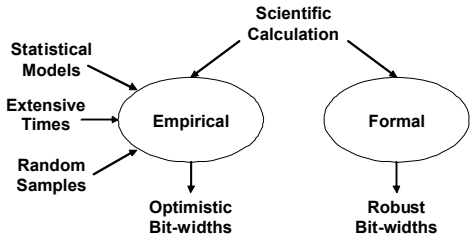


Figure 1. Contrasting empirical and formal approaches.

Simulation based approaches rely on a representative input data set and work by comparing the outcome of simulation of the reduced precision system to that of the "infinite" precision system, "infinite" being approximated by "very high" - e.g. double precision floating-point on a general purpose machine. Approaches including [1, 8] seek to determine the range of intermediate variables by direct simulation while [13] creates a new system related to the difference between the infinite and reduced precision systems, reducing the volume of simulation data which must be applied. Although simulation tends to produce more compact data representations than analytical approaches, often the resultant system is not robust, i.e. situations not covered by the simulation stimuli can lead to overflow conditions resulting in incorrect behavior.

These methods are largely inadequate for scientific computing, due in part to differences between general scientific computing applications and the DSP/embedded systems application domain. Many DSP systems can be characterized very well (in terms of both their input and output) using statistics such as expected input distribution, input correlation, signal to noise ratio, bit error rate, etc. This enables efficient stimuli modeling providing a framework for simulation, especially if error (noise) is already a consideration in the system (as is often the case for DSP). Also, given the real-time nature of many DSP/embedded systems applications, the potential input space may be restricted enough to permit very good coverage during simulation. Contrast these scenarios to general scientific computing where there is often minimal error consideration provided and where stimuli characterization is often not as extensive as for DSP.

As seen in Figure 1, simulation based methods are characterized by a need for models and stimuli, excessive runtimes and lack of robustness, due to which they cannot be relied upon in scientific computing implementations. In contrast, analytical methods depend on the calculation only and provide robust bit-widths. An obvious analytical approach to the problem is known as range or interval arithmetic (IA) [9], which establishes worst-case bounds on each intermediate step of the calculation by establishing worst-case bounds on individual operations. Expressions can be derived for the elementary operations, and compounded starting from the range of the inputs. However, since dependencies between intermediate variables are not taken into account, the *range explosion* phenomenon results; the range obtained using IA is much larger than the actual possible range of values causing severe over-allocation of resources.

In order to combat this, *affine arithmetic (AA)* has arisen which keeps track (linearly) of interdependencies between variables (e.g. [4, 10]) and non-affine operations are replaced with an affine approximation often including introduction of a new variable (consult [7] for a summary of approximations used for common non-affine operations). While often much better than IA, AA can still result in an overestimate of an intermediate variable's potential range, particularly when strongly non-affine operations occur as a part of the calculation, a compelling example being division. As [4] points out, this scenario is rare in DSP, accounting in part for the success of AA in DSP however it occurs frequently in scientific calculations.

Exactly solving the range determination problem is tantamount to solving global optimization in general for which no scalable methods are known. While relaxation to a convex problem is a common technique for solving some non-convex optimization problems [2], the resulting formulation for some scientific calculations can be extremely ill-conditioned, leading once more to resource over-allocation.

In order to demonstrate the hurdles encountered by both simulation and existing analytical methods when applied to scientific calculations, and thereby motivate this work, consider the example in the following subsection.

2.1 Motivational Example

Let $\mathbf{d}$ and $\mathbf{r}$ be vectors $\in \mathbb{R}^4$, where for both vectors, each component lies in the range $[-100, 100]$. Suppose we have:

$$z = \frac{z_1}{z_2} = \frac{\mathbf{d} \cdot \mathbf{r}}{1 + \|\mathbf{d} - \mathbf{r}\|}$$

and we want to determine the range of z for integer bit-width allocation. Table 1 shows the ranges obtained from simulation, affine arithmetic and the proposed method. Notice that simulation underestimates the range by 2 bits after 540 seconds, $\approx 5\times$ the execution time of the proposed method (98 seconds). This happens because only a very small but still important fraction of the input space where $\mathbf{d}$ and $\mathbf{r}$ are identical (to reduce z_2) and large (to increase z_1) will maximize z. In contrast, the formal methods always give hard bounds but because the affine estimation of the range of the denominator contains zero, affine arithmetic cannot provide a range

Var.	Empirical		Formal			
	Simulation		Affine		Proposed	
	Range	Bits	Range	Bits	Range	Bits
z_1	[-3.7e4,3.7e4]	17	[-4e4,4e4]	17	[-4e4,4e4]	17
z_2	[1,1.4e5]	18	[-8e4,1.6e5]	18	[0,1.6e5]	18
z	[0,1e4]	14	∞	–	[-864,4e4]	16

Table 1. Motivational example.

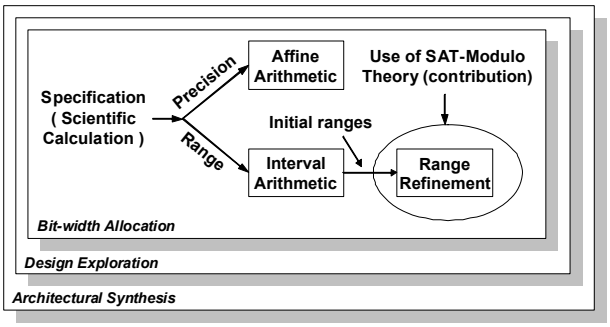


Figure 2. Overall bit-width allocation flow.

for the quotient z. *This scenario is handled correctly by the proposed method which maintains all the benefits of analytical (formal) methods while at the same time visibly tightening the range of the operands. The key to this is the usage of the recent developments in SAT-Modulo Theory and details of its operation are discussed next.*

3 Proposed Solution

Given the inability of simulation based methods to provide robust variable bounds, and the limited accuracy of bounds from affine arithmetic in the presence of strongly non-affine expressions (Section 2.1), we proposed a method of range refinement based on SAT-Modulo Theory. The context of this work is shown in Figure 2, note that we focus on the range determination problem. Existing analytical techniques (e.g. [4, 6, 10]) have had more success in precision analysis over range analysis largely because precision analysis deals with small ranges, over which affine approximations are usually sufficiently reliable. As a result, we focus this work on the range analysis problem.

To address this problem, we take a two stage approach as shown in Figure 2 where loose bounds are obtained from interval arithmetic, and subsequently refined using SAT-Modulo Theory which is detailed in the next section.

3.1 SAT-Modulo Theory

Boolean satisfiability (SAT) is a well known NP-complete problem which seeks to answer whether for a given set of clauses in a set of boolean variables, there exists an assignment of those variables such that all the clauses are true. SAT-Modulo theory (SMT) generalizes this concept to first-order logic systems. Under the theory of real numbers, boolean variables are replaced with real variables and clauses are replaced with constraints. This gives rise to instances such as: is there an assignment of $x, y, z \in \mathbb{R}$ for which $x > 10$, $y > 25$, $z < 30$ and $z = x + y$, which for this example there is not i.e., this instance is *unsatisfiable*.

Instances are given in terms of variables with accompanying ranges and constraints. The solver attempts to find an assignment on the input variables (inside the ranges) which satisfies the constraints. Most implementations follow a 2-step model analogous to modern Boolean SAT solvers: 1) the *Decision step* selects a variable, splits its range into two,

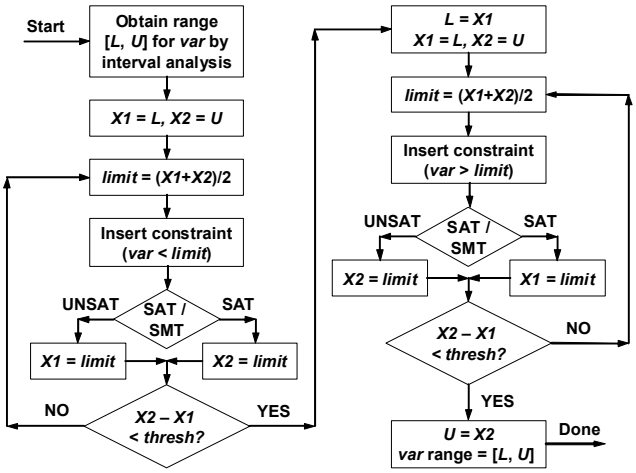


Figure 3. SAT/SMT range refinement of *var*.

and temporarily discards one of the sub-ranges then 2) the *Propagation step* infers ranges of other variables from the newly split range. Unsatisfiability of a subcase is proven when the range for any variable becomes empty which leads to backtracking (evaluation of a previously discarded portion of a split). The solver proceeds in this way until it has either found a satisfying assignment or unsatisfiability has been proven over the entire specified domain.

Building on this framework, an SMT engine can be used to prove/disprove validity of a bound on a given expression by checking for satisfiability. Section 3.2 details how bounds proving can be used as the core of a procedure addressing the range determination problem.

3.2 Range Refinement using SMT

Figure 3 illustrates the binary search method employed for range analysis on the intermediate variable: *var*. Note that each SMT instance evaluated contains the inserted constraint ($val < limit$ or $val > limit$). The loop on the left of the figure (between "$limit = (X1 + X2)/2$" and "$X2 - X1 < thresh$?") narrows in on the lower bound L, maintaining $X1$ less than the true (and as yet unknown) lower bound. Each time satisfiability is proven, $X2$ is updated while $X1$ is updated in cases of unsatisfiability, until the gap between $X1$ and $X2$ is less than a user specified threshold. Subsequently, the loop on the right performs the same search on the upper bound U, maintaining $X2$ greater than the true upper bound. Since the SMT solver works on the exact calculation, all interdependencies among variables are taken into account (via the SMT constraints) so the new bounds successively remove over-estimation arising in the original bounds resulting from the use of interval arithmetic.

The overall range refinement process, Algorithm 1, uses the steps of the calculation and the ranges of the input variables as constraints to set up the base SMT formulation, note that this is where *Insert constraint:* from Figure 3 inserts to. It then iterates through the intermediate variables

Input : CalculationSteps, InputVarList, InputVarRanges, IntermediateVarList

Output : IntermediateVarRanges

```
Use CalculationSteps to set up base SMT formulation;
foreach var in IntermediateVarList do
    Refine range of var (Figure 3);
    update IntermediateVarRanges for var;
    update base SMT formulation with range of var;
end
RETURN IntermediateVarRanges;
```

Algorithm 1: RangeRefine

applying Figure 3 to obtain a refined range for that variable. Once all variables have been processed the algorithm returns ranges for the intermediate variables.

3.3 Dealing with Division

As discussed in Section 2, non-affine functions with high curvature cause problems for AA, and while these are rare in the context of DSP (as confirmed by [4]) they occur frequently in scientific computing. Division is in particular a problem due to range inversion i.e., quotient increases as divisor decreases. While AA tends to give reasonable (but still overestimated) ranges for compounded multiplication since product terms and the corresponding affine expression grow in the same direction, this is not the case for division. Furthermore, both IA and AA are unequipped to deal with divisors having a range that includes zero.

Use of SMT mitigates these problems, divisions are formulated as multiplication constraints which must be satisfied by the SAT engine, and an additional constraint can be included which restricts the divisor from coming near zero. Since singularities such as division by zero result from the underlying math (i.e. are not a result of the implementation) their effects do not belong to range/precision analysis and SMT provides convenient circumvention during analysis.

3.4 Consideration of Run-time

While leveraging the mathematical structure of the calculation enables SMT to provide much better run-times than using Boolean SAT (where the entire datapath and numerical constraints are modeled by clauses obfuscating the mathematical structure), run-time may still become unacceptable as complexity of the calculation under analysis grows. To address this, a timeout is used to cancel the inquiry if it does not return before timeout expiry. In this way the tradeoff between run-time and the tightness of the variables' bounds can be user controlled. If canceled, the inquiry result defaults to satisfiable in order to maintain robust bounds, i.e. to assume satisfiable gives pessimistic bounds.

4 Case Studies

Given that target application domain for this method of hardware acceleration for scientific computing, we seek to address specifically the problem of division which is known both to be common in scientific calculations, and to cause problems for existing methods. This section details case studies involving division, as well as one non-affine example from DSP.

4.1 Energy Spectral Density

An application involving non-linearity which appears in DSP is the calculation of energy spectral density (ESD), which can be obtained as:

$$\Phi(\omega) = \frac{1}{2\pi} F(\omega) F^*(\omega)$$

where $F(\omega)$ indicates the Fourier Transform of the signal of interest, or the Fast Fourier Transform (FFT) for discrete signals. Since the FFT itself is affine, AA provides exact bounds on all intermediate variables however, the ESD involves magnitude of a complex number (non-affine) leading to range overestimation.

4.2 Doppler Effect

The Doppler effect is the apparent change in frequency observed when a sound source is in motion with respect to an observer. For a given emitted frequency v and a relative speed of u between the source and observer, the perceived frequency will be $v' = cv \div (c+u)$ where c is the speed of sound in the medium. If the medium is air and we wish to know how the rate of frequency change with respect to the relative speed u depends on temperature, we have:

$$z = \frac{dv'}{du} = \frac{-(331.4 + 0.6T)v}{(331.4 + 0.6T + u)^2}$$

using the approximation for the speed of sound in air $c \approx 331.4 + 0.6T$, T in degrees Celsius.

4.3 A Rational Function

This case employs a rational function such as those which arise when fitting curves to experimental data. Consider the following function and its derivative:

$$z_1 = \frac{25t^2 + 125}{t^2 + 1} \qquad z_2 = \frac{dz_1}{dt} = \frac{-200t}{(t^2+1)^2}$$

over the range $-100 \leq t \leq 100$. It is worth noting that in addition to being common in scientific computing, such calculations may also arise in an embedded system, e.g. as a part of the model used for prediction/control.

4.4 Newton's Method

The final case deals with application of Newton's method applied to a polynomial. Given a polynomial

$$f(x) = c_3x^3 + c_2x^2 + c_1x + c_0$$

roots can be obtained by using Newton's method:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

If we consider a single iteration, this results in:

$$z = x - z_3 \qquad z_3 = \frac{z_1}{z_2} = \frac{c_3x^3 + c_2x^2 + c_1x + c_0}{3c_3x^2 + 2c_2x + c_1}$$

5 Experimental Results

In this section we compare the results of AA and the proposed SMT approach applied to the case studies of the previous section. Experiments were carried out on 1.5 GHz Pentium 4 with 512 MB of RAM running Gentoo Linux, using the freely available *HySAT* implementation [5, 15] as the core SMT solver. Ranges were obtained using (unless otherwise specified) a timeout of 2 seconds, resulting in runtimes for all cases on the order of 100 seconds. In all cases the number of bits needed for range $[L,U]$ has been taken as $\lceil \log_2(U-L) \rceil$ to facilitate evaluation of the actual span of the numbers, taking into account how they are centered. In almost all cases however, the result is identical to taking $\lceil \log_2(\max(|L|,|U|)) \rceil + a$, where a is 0 if L and U have the same sign, and 1 otherwise.

5.1 Energy Spectral Density

As discussed in Section 4.1, the energy spectral density can be obtained from the magnitude of the FFT. This experiment uses an 8-point FFT with each of the 8 inputs a complex number in $[-128,128]+[-128,128]i$. Due to the affine nature of the FFT, both AA and SMT provide exact bounds on all intermediate variables in the FFT calculation however AA overestimates the magnitude (non-affine).

Table 2 shows the ranges obtained from both AA and SMT when applied to each of the 8 outputs of the ESD calculation (to obtain these ranges a SAT timeout of 5 seconds was used). Note that AA ranges are centered close to zero while SMT ranges start near zero which is correct as only positive values would be expected.

Clearly for this calculation, AA provides good estimates of the ranges and thus the bit-widths, since only one level of non-affine calculations occurs. In light of the inclusion

	Affine		SAT-Modulo	
Output	Range	Bits	Range	Bits
0	[-1835008 , 2097152]	22	[-1 , 2097153]	22
1	[-2373666 , 2635814]	23	[-1 , 1984106]	21
2	[-2269321 , 2531463]	23	[-1 , 1790022]	21
3	[-2373666 , 2635814]	23	[-1 , 2052757]	21
4	[-1835008 , 2097152]	22	[-1 , 2097153]	22
5	[-2373666 , 2635814]	23	[-1 , 1957096]	21
6	[-2269321 , 2531463]	23	[-1 , 1790023]	21
7	[-2373666 , 2635814]	23	[-1 , 2029555]	21

Table 2. Affine vs. SAT-Modulo for Energy Spectral Density.

	Affine		SAT-Modulo	
Output	Range	Bits	Range	Bits
q_1	[313 , 362]	6	[313 , 362]	6
q_2	[-473252 , 7228000]	23	[6267 , 7228000]	23
q_3	[213 , 462]	8	[213 , 462]	8
q_4	[25363 , 212890]	18	[45539 , 212890]	18
z	[-80,229]	9	[0 , 138]	8

Table 3. Affine vs. SAT-Modulo for Doppler.

	Affine		SAT-Modulo	
Output	Range	Bits	Range	Bits
q_1	[125 , 250125]	18	[124 , 250126]	18
q_2	[1 , 10001]	14	[0 , 10002]	14
q_3	[-20000 , 20000]	16	[-20001 , 20001]	16
q_4	[-24999999 , 100020001]	27	[0 , 100020008]	27
z_1	[-250 , 369]	10	[24 , 126]	7
z_2	∞	–	[-67 , 67]	8

Table 4. Affine vs. SAT-Module for A Rational Function.

of a large range of numbers below zero, subsequent calculations relying on the ESD can be expected to already begin experiencing range inflation, especially as more variable interdependencies arise. Consider also that division does not occur, which it does in the following examples.

5.2 Doppler Effect

This case study was broken intermediately into:

$$q_1 = 331.4 + 0.6T \quad q_2 = q_1 v$$
$$q_3 = q_1 + u \quad q_4 = q_3^2 \quad z = q_2/q_4$$

and the parameters that were used were:

- temperature: $-30°C \le T \le 50°C$
- audible frequencies: $20Hz \le v \le 20000Hz$
- relative speed: $-100m/s \le u \le 100m/s$

Observing Table 3, note that AA and SMT provide comparable ranges for all variables except for z where the division occurs and where the bit-width is overestimated by 1 bit. Despite the fact that this calculation has fewer levels of intermediate variables than the aforementioned ESD, and the fact that all upper bounds from AA were exact, the resultant range was still overestimated, a prime example of the result of range inversion mentioned in Section 3.3.

5.3 A Rational Function

For this case study there is only one free variable, $-100 \le t \le 100$ leading to strong correlations between all the intermediates:

$$q_1 = 25t^2 + 125 \quad q_2 = t^2 + 1 \quad z_1 = q_1/q_2$$
$$q_3 = -200t \quad q_4 = q_2^2 \quad z_2 = q_3/q_4$$

Table 4 shows how the ranges evolve, as before the output z_1 suffers because of the division. Notice as well that AA cannot provide bounds for z_2 because the range of the divisor (q_4) includes zero, according to the affine approximation. Even if we cheat and use the SMT lower bound of $q_4 \ge 1$, the resultant range will be $z_2 \in [-20000, 20000]$ requiring 16 bits, 8 more than allocated by SMT.

Output	Affine		SAT-Modulo	
	Range	Bits	Range	Bits
z_1	[-1205360 , 1170360]	22	[-1205361 , 1135361]	22
z_2	[-5753 , 35769]	16	[1 , 35769]	16
z_3	∞	–	[-39 , 38]	7
z	∞	–	[-69 , 72]	8

Table 5. Affine vs. SAT-Module for Newton's Method.

5.4 Newton's Method

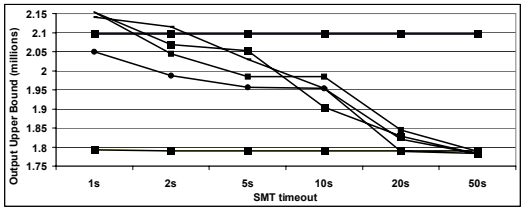
In this final case study, the fully expanded intermediates have been omitted for readability. The dividend and divisor polynomials from 4.4 (z_1 and z_2) were expanded in intermediate steps using Horner's method [3] to reflect a potential hardware implementation. The range of x was [-100,100] and the coefficient ranges were:

$$c_0 \in [-10, 10] \qquad c_1 \in [7.5, 8.5]$$
$$c_2 \in [-3.75, -3.25] \quad c_3 \in [0.833, 1.167]$$

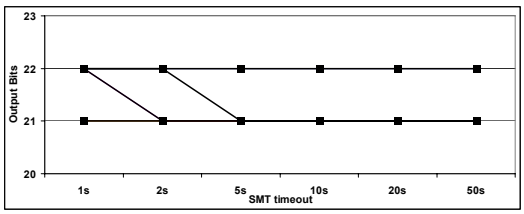
Table 5 shows the results for the major intermediates (the ones which have been omitted had identical bit-widths), where as before the quotient z_3 cannot be calculated due to the inclusion of zero within the range of the divisor z_2. If as before we use the bound from SMT (1.83 which has been floored to 1 in the table), we end up with $z_3 \in [-658668, 620416]$ and $z \in [-658768, 620516]$ requiring 22 integer bits each, at least 14 more than necessary per signal.

5.5 Run-time/Accuracy Tradeoff

We now briefly discuss the tradeoff between SMT timeout and accuracy of bounds. Figure 4 shows how the range and number of bits for the 8 ESD outputs (it is unimportant which is which) vary as timeout increases. Note first that the bounds are always hard (robust), and thus decrease as the SMT solver is allowed to run for longer. For short timeouts, the SMT assumes satisfiability when terminated, and extra bit-width ends up being allocated. Notice also however that as the timeout is increased the range comes down very slowly (Figure 4(a)), while the number of bits is met with much less effort (Figure 4(b)).



(a) ESD Output Ranges



(b) ESD Output Bit-widths

Figure 4. Effect of timeout on range/bit-width.

6 Conclusion / Future Work

This paper has demonstrated the use of SAT-Modulo theory for range analysis in bit-width allocation, and results have shown that robust bounds (unlike from simulation) can be obtained which are significantly tighter than from affine arithmetic. Ongoing work for this method will include expansion to deal with iterative methods such as Newton's method (for more than just one iteration as in Section 4.4) and Conjugate Gradient, as well as exploring solver runtime management for more complex calculations.

References

[1] P. Belanovic and M. Rupp. Automated Floating-point to Fixed-point Conversion with the Fixify Environment. In *Proc. International Workshop on Rapid System Prototyping*, pages 172–178, 2005.

[2] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[3] R. L. Burden and J. D. Faires. *Numerical Analysis, 7th Edition*. Brooks Cole, 2000.

[4] C. Fang, R. Rutenbar, and T. Chen. Fast, Accurate Static Analysis for Fixed-point Finite-precision Effects in DSP Designs. In *Proc. International Conference on Computer Aided Design (ICCAD)*, pages 275–282, 2003.

[5] M. Franzle and C. Herde. HySAT: An Efficient Proof Engine for Bounded Model Checking of Hybrid Systems. *Formal Methods in System Design*, 30(3):178–198, June 2007.

[6] D.-U. Lee, A. Gaffar, R. Cheung, O. Mencer, W. Luk, and G. Constantinides. Accuracy-Guaranteed Bit-Width Optimization. *IEEE Transactions on Computer-Aided Design*, 25(10):1990–2000, October 2006.

[7] J. Lopez, C. Carreras, and O. Nieto-Taladriz. Improved Interval-Based Characterization of Fixed-Point LTI Systems With Feedback Loops. *IEEE Transactions on Computer Aided Design*, 26(11):1923–1933, November 2007.

[8] A. Mallik, D. Sinha, P. Banerjee, and H. Zhou. Low-Power Optimization by Smart Bit-Width Allocation in a SystemC-Based ASIC Design Environment. *IEEE Transactions on Computer-Aided Design*, pages 447–455, March 2007.

[9] R. Moore. *Interval Analysis*. Prentice Hall, 1966.

[10] W. Osborne, R. Cheung, J. Coutinho, W. Luk, and O. Mencer. Automatic Accuracy-Guaranteed Bit-Width Optimization for Fixed and Floating-Point Systems. In *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, pages 617–620, 2007.

[11] K. Sano, T. Iizuka, and S. Yamamoto. Systolic Architecture for Computational Fluid Dynamics on FPGAs. In *Proc. International Symposium on Field-Programmable Custom Computing Machines*, pages 107–116, 2007.

[12] R. Scrofano, M. Gokhale, F. Trouw, and V. Prasanna. Accelerating Molecular Dynamics Simulations with Reconfigurable Computers. *IEEE Transactions on Parallel and Distributed Systems*, 19(6):764–778, June 2008.

[13] C. Shi and R. Brodersen. An Automated Floating-point to Fixed-point Conversion Methodology. In *Proc. International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 529–532, 2003.

[14] T. Todman, G. Constantinides, S. Wilton, O. Mencer, W. Luk, and P. Cheung. Reconfigurable Computing: Architectures and Design Methods. *IEE Proceedings - Computers and Digital Techniques*, pages 193–207, March 2005.

[15] University of Oldenburg. HySAT Download. http://hysat.informatik.uni-oldenburg.de/26273.html.