Using Arithmetic Transform to Calculate Ranges of Arithmetic Datapaths

Yu Pang[1], Yafeng Yan[1], Junchao Wang[1], Zhilong He[1], Ting Liu[2]
Chongqing University of Posts and Telecommunications, Chongqing, China[1]
Aquart Creation Inc., Montreal, Quebec, Canada[2]

Email: pangyu@cqupt.edu.cn, Frederic.liu@aquartcreation.com

***Abstract* — Range analysis is used to optimize circuits and to allocate the bit-width. In this paper we introduce a new and efficient static method to analyze the datapath represented by a polynomial, and compute the range for the datapath, then yield the smallest output bit-width. The method is based on Arithmetic Transform which can guarantee accuracy, and can explore polynomials with single or multiple word-level variables. The experiments show the tighter bounds than other recent static methods and significantly faster executions than simulation.**

***Index Terms* — Range, Arithmetic datapath, Static method, Arithmetic Transform**

I. INTRODUCTION

Finite bit-widths cause the imprecision in results, as well as the inability to represent all values within the function range. Customizable hardware such as field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs) can provide freedom for bit-width optimization. Minimizing the bit-width while achieving sufficient precision [24] and range is significant to high-level synthesis reducing the cost of the circuit.

However, it is not easy to determine the best bit-widths. The principal way is the *range analysis*. The tight range analysis is instrumental in exploring datapath and reducing the cost of arithmetic circuits. Allocating the output bit-width requires the calculation of the numerical value range in terms of the inputs. Previous explorations mostly relate to the finite bit-width causes. *Dynamic analysis* [1]-[4], as a simulation-based method, is used to explore numerical value and analyze range. *Nayak et al* [2] present a framework for generating an efficient hardware for signal processing applications described by Matlab. They rely on data range propagation, while precisions are analyzed and optimized by the DFG which is an acyclic graph representation of a circuit. A memory packing algorithm is proposed to generate faster hardware requiring less execution time. *C. Shi et al*. [3] set up a statistical model to estimate hardware resource in terms of perturbation theory. A tool that automates the floating-point to fixed-point conversion (FCC) process for digital signal system is described based on a simulation tool, Simulink. The tool automatically optimizes fixed-point data types of arithmetic operators, including overflow modes, integer word lengths, fractional word lengths, and the number systems.

To avoid tedious simulation, *static analysis* has been developed. Interval arithmetic (IA) is a usual method to calculate the range. The affine arithmetic model (AA) is a derivation of IA borrowed from numerical analysis. In AA, the quantities of interest are represented as linear combinations (affine forms) of certain primitive variables, which stand for sources of uncertainty in the data or approximations made during the computation. *Fang et al.* [5] take advantage of affine arithmetic modeling to analyze range and precision. Work in [6] adopts the static analysis to investigate bit-widths in the case of truncated and rounded data. Further, they explore hardware area and delay for FPGA implementations for different bit-widths. Work in [7] extends [6] to provide a method for optimizing word-lengths of hardware designs with fixed-point arithmetic based on analytical error models that guarantee accuracy.

Constantinides et al. [9] propose Synoptix - an optimization technique targeting linear time-invariant digital signal processing systems using an original resource binding technique. Synoptix is based on saturation arithmetic to perform the bit-width optimizations and range analysis. *Kinsman and Nicolici* [10] use SAT-Modulo theory (SMT) for range analysis in bit-width allocation. Bit-widths are determined for finite precision implementation of numerical calculations, and can get more accurate range estimation than IA and AA. The SMT technique relies on a user-specified threshold - if the threshold is close to zero, the obtained range approaches the exact range but huge calculation time might be needed since SMT splits the obtained range and backtracks in each iteration.

In this paper, we propose a new method based on Arithmetic Transform (AT) that can obtain the tight value range for a polynomial given input bit-widths, and allocate the smallest bit-width for the output. The method which can be applied in many fields such as artificial intelligence [23] directly helps engineers in

doi:10.4304/jcp.7.12.2906-2912

understanding the polynomial, and improving the cost and performance of datapaths.

II. Related Work

When undertaking simulation-based methods, one cannot in general use exhaustive stimuli to explore the range. For example, to process a 32-bit arithmetic circuit, one has to simulate 2^{32} cases to determine the range. The usual static analysis is commonly based on IA and AA. IA defines a set of operations on intervals. An operation <OP> on two intervals with <OP> for example being addition or multiplication, is defined by:

$$[x_1, x_2] < OP > [y_1, y_2] = \{x < OP > y \mid x \in [x_1, x_2], y \in [y_1, y_2]\}$$

For example, the practical application of addition can be simplified further as:

$$[x_1, x_2] + [y_1, y_2] = [x_1 + y_1, x_2 + y_2]$$

In AA, an ordinary interval $[x_{min}, x_{max}]$ for an input variable can be converted into an equivalent affine form $\hat{x} = x_0 + x_1 \varepsilon_1$ with [13]

$$x_0 = \frac{x_{\max} + x_{\min}}{2} \qquad x_1 = \frac{x_{\max} - x_{\min}}{2} \tag{1}$$

The intermediate signal or the output is represented as a first degree polynomial:

$$Y = y_0 + y_1\varepsilon_1 + y_2\varepsilon_2 + ... + y_n\varepsilon_n \tag{2}$$

where y_0, y_1, ... y_n are floating-point numbers and $\varepsilon_1, \varepsilon_2 ... \varepsilon_n$ are symbolic variables whose values are only known to lie in the range [-1,+1].

Example 1: *Consider a datapath represented as a polynomial z=ab+c-b, and the range of signals as shown in square brackets, Figure 1 [6]. We can get the maximum value max([-1,2]*[4,10]+4.3-[4,10]) = 2*10+4.3-4 = 20.3 and the minimum value -1*10+4.3-10= -15.7 by IA, so the range is [-15.7, 20.3]. Figure 1 shows how to use AA to get a range value.*

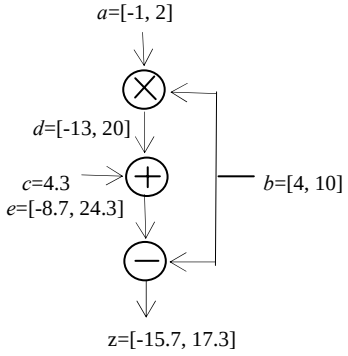


Figure 1. Example performing $z=ab+c-b$

In affine form, we get:

$$\hat{a} = 0.5 + 1.5\varepsilon_1 \qquad \hat{b} = 7 + 3\varepsilon_2 \qquad \hat{c} = 4.3$$

$$\hat{d} = \hat{a}\,\hat{b} = 3.5 + 10.5\varepsilon_1 + 1.5\varepsilon_2 + 4.5\varepsilon_1\varepsilon_2$$

$$= 3.5 + 10.5\varepsilon_1 + 1.5\varepsilon_2 + 4.5\varepsilon_3$$

$$\hat{e} = \hat{d} + \hat{c} = 7.8 + 10.5\varepsilon_1 + 1.5\varepsilon_2 + 4.5\varepsilon_3$$

$$\hat{z} = \hat{e} - \hat{b} = 0.8 + 10.5\varepsilon_1 - 1.5\varepsilon_2 + 4.5\varepsilon_3$$

The primary two input variables "*a*" and "*b*" are represented by Eqn. (1). The intermediate signals "*d*" and "*e*" and the output signal "*z*" are represented as Eqn. (2) by compound operations of the inputs. AA gets the value range as [-15.7, 17.3] and the range is tighter than that obtained by IA. However, the exact range is [-15.7, 14.3] because AA neglects correlation. As $\varepsilon_1 \varepsilon_2 = \varepsilon_3$ in $\hat{a}\ \hat{b}$, and the term $\varepsilon_1 \varepsilon_2$ has correlation with the two variables ε_1 and ε_2, but AA uses a new variable ε_3 as a substitution. This new variable is independent with ε_1 and ε_2 so AA has to extend the range. The presence of the correlation in a polynomial indicates that at least two of its monomials include the same variable. From the above example, we can see if using IA and AA, 6 bits must be used to represent the signed integer range but only 5 bits are enough for the exact range. The reason for the difference is that IA and AA provide no capability to keep track of the correlation among signals. Our method accounts for the correlation, resulting in much tighter range.

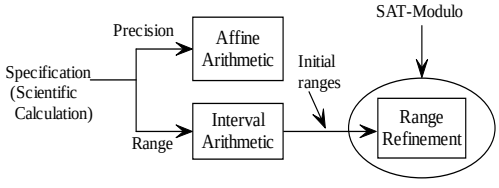
Our method is based on *Arithmetic Transform* which is a spectral method. The AT [11] is a canonical polynomial representing a word-level function $f : B^n \to w$ using an arithmetic operation "+", word- level coefficients c, binary inputs $x_1, x_2 \ldots x_n$ and binary exponents $i_1, i_2 \ldots i_n$:

$$AT(f) = \sum_{i_1=0}^{1}\sum_{i_2=0}^{1}...\sum_{i_n=0}^{1} c_{i_1 i_2 ... i_n} x_1^{i_1} x_2^{i_2} ... x_n^{i_n} \tag{3}$$

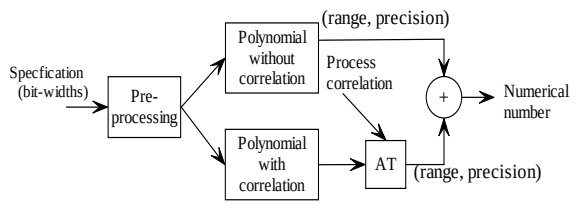
Given a real-valued polynomial with word-level variables X and Y composed of binary vectors (x_{N-1}, $x_{N-2}, ... x_0$, y_{M-1}, y_{M-2}, $\ldots$ $y_0, \ldots$), the AT is constructed by replacing variables by its defining polynomial. For instance, if X and Y are unsigned integers with 3 and 4 bits, and the polynomials is $f(X,Y) = 2X^2 + Y^3$, its corresponding AT polynomial form is:

$$AT[f(X,Y)] = 2 * (\sum_{i=0}^{2} x_i 2^i)^2 + (\sum_{k=0}^{3} y_k 2^k)^3$$

After the polynomial is converted to AT, the verification can be performed by a search over binary input variable assignments. A branch-and-bound algorithm from [12] that efficiently finds a maximum value of an AT polynomial is crucial for a variety of verification and optimization tasks. We now show how to exactly compute the range based on AT. Figure 2 compares our method with past explorations.



a)



b)

Figure 2. Flows comparison between our method and past explorations

Starting from a specification, IA and AA calculate range and precision respectively. Improvements such as SAT-Modulo technique were used to refine the obtained range by IA. Our method begins with a specification represented by a polynomial and a preprocess module partitions it into two sub-polynomials, then AT technique handles correlation as in Figure 2.b). Unlike past static methods, range and precision are processed together, and correlation is handled to guarantee accuracy.

III. Algorithm for Calculating Range

To develop the algorithm, we need to analyze the AT polynomial representing a datapath. In this section, we use the static analysis to replace simulation and develop a novel method to calculate range.

A. Polynomial Analysis

Given a polynomial to represent a datapath and finite bit-widths for the input, to find the range and allocate the output bit-width, the essential step is to efficiently compute the upper bound and lower bound for the polynomial. Consider an example of the analysis.

Example 2: *A polynomial with four word-level variables (A, B, C, D) has unsigned fractional bit-widths of (5,4,5,6) for each variable respectively. All intermediate variables have 6 fractional bits. The polynomial is:*

$$E = B^2 + 3AB - 2A^2 + 6CD$$

Using IA to analyze the polynomial, the upper bound can be obtained while each positive term is set to the maximum value and the negative term is set to "0". However, it is impossible that the terms "3*AB*" and "-$2A^2$" can be obtained the maximum value and "0" concurrently, since the variable *A* in the term "3*AB*" needs to be set the maximum value and the variable *A* in the term "-$2A^2$" needs to be set "0". The reason is that correlation exists in the two terms which leads that the obtained upper bound is bigger than the real upper bound. Although AA is improved, it does not yet handle the terms correlation perfectly. Now we adopt AT to handle polynomials, which can guarantee the exact range computation.

1) Separate input variables into two sets as S_c and S_{nc} which represent a correlative set and a non-correlative set respectively. If one variable appears beyond one time in the polynomial, it is classified into the set S_c, otherwise it belongs to S_{nc}. In this example, the variable *A* exists in two terms "*3AB*" and "*-2A²*" and the variable *B* exists in two terms "*B²*" and "*3AB"*. Since they both appear two times, the set S_c covers them. The variables *C* and *D* only exist in the term "*6CD*", so they belong to the set S_{nc}. After the step, we get $S_c = (A, B)$ and $S_{nc} = (C, D)$.
2) Partition the original polynomial into a correlative polynomial P_c and a non-correlative polynomial P_{nc}. If all variables in a term belong to S_{nc}, the term is classified into P_{nc}; otherwise the term is in P_c. In the example, the term "*6CD*" has two variables and they are both in S_{nc}, so $P_{nc} = 6CD$ and since the other terms contain at least one variable in S_c, they are all classified into the correlative polynomial, so $P_{nc} = B^2 + 3AB - 2A^2$.
3) Get the maximum value V_{nc_max} of the non-correlative polynomial P_{nc}. Since its coefficient is positive, that is, "5", it is easy to know while *C* and *D* both reach their maximum values, P_{nc} can obtain its maximum value. So

$$V_{nc\_max} = 6 * \sum_{i=0}^{4} 2^{-i-1} * \sum_{k=0}^{5} 2^{-k-1}$$
$$= 6 * 0.9688 * 0.9844 = 5.7221$$

4) Set variable values in the correlative polynomial. There are two variables, *A* and *B*, and three terms. Variable *A* has different signs for coefficients in present terms of "*3AB*" and "*–2A²*", so it is hard to be set directly. Variable *B* has same signs in present terms of "*3AB*" and "*B²*" because the coefficients "3" and "1" are both positive. Therefore, *B* can be set to the maximum value 0.9375, and P_c is changed to:
$$P_c = 0.8789 + 2.8125A - 2A^2$$
5) Convert P_c into AT and search its maximum value. Using the conversion algorithm in [12], the representation of $AT(P_c)$ is:

$$AT(P_c) = 0.8789 + 2.8125 \sum_{i=0}^{4} (2^{-i-1} a_i) - 2 (\sum_{i=0}^{4} (2^{-i-1} a_i))^2$$

By invoking the search algorithm [12], the maximum value V_{c_max} is obtained. As $A = 0.75$, it is 0.8789+0.9844 =1.8633.
6) Get the upper bound of the original polynomial. After V_{nc_max} *and* V_{c_max} have been obtained, the upper bound of the polynomial is 5.7221 + 1.8633 = 7.5854 when A=0.75, B=0.9375, C=0.9688 and D=0.9844. Similarly, we can get the lower bound –1.877.

Figure 3 describes how to determine overflow from above Step 1) to 7). If simulation is used, it has to calculate $2^{5+4+5+6}=2^{20}$ possible values. The static analysis sets direct values for three variables *"B"*, *"C"* and *"D"*, and only converts a simple polynomial with one word-level variable *"A"* to AT, so it is far more efficient.

If using IA and AA, the upper bounds are 9.3258 and 8.4352 respectively, they are both looser than our method which can get the exact range. After the range is obtained, we allocate the bit-width for the output *E*. The integer bit-width (IB) is calculated as:

$$IB = [\log2 (\max(|x_{low}|, |x_{upp}|))] + \alpha$$

$$\alpha = \begin{cases} 1, & \text{mode } (\log2 (x_{upp}), 1) \neq 0 \\ 2, & \text{mode } (\log2 (x_{upp}), 1) = 0 \end{cases} \quad \text{where}$$

In Example 2, the IB is [log2 (7.5854)] +1 = 4. Notice that if using the bound obtained by AA or IA, the IB equals 5, therefore the output requires one more bit for representation.

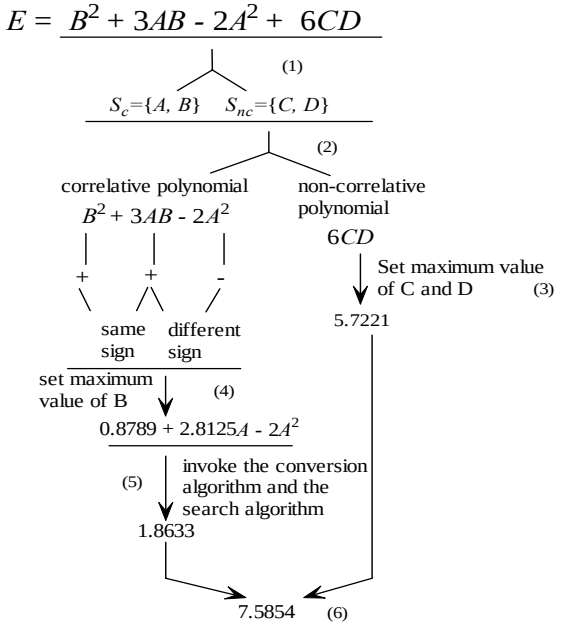


Figure 3. Steps to find whether overflow occurs for the example

Now we allocate the FB. The works [6] [7] introduce precision analysis. Each input *x* cause the maximum error $2^{-FB(x)-1}$ where *FB(x)* is the number of fractional bits for the fixed-point number *x*. The operation is shown as:

$$y = a * b,$$
$$y_{error} = 2^{-FB(y)-1} + a_{error} * b_{max\_range} + a_{max\_range} * b_{error}$$
$$y = a + b,$$
$$y_{error} = 2^{-FB(y)-1} + a_{error} + b_{error}$$

In Example 2, the intermediate variable $Z = A * B$ has 6 fractional bits, so its error is shown as:

$$Z_{error} = 2^{-FB(Z)-1} + A_{error} * B_{max\_range} + A_{max\_range} * B_{error} + A_{error} * B_{error}$$
$$= 2^{-7} + 2^{-6} * 0.9375 + 0.96875 * 2^{-5} + 2^{-6} * 2^{-5} = 0.0527$$

The output FB must satisfy the inequality:

$$E_{error} > (B^2)_{error} + (AB)_{error} - (A^2)_{error} + (CD)_{error}$$
$$\Rightarrow 2^{-FB-1} > 0.06641+0.0527-0.06836+0.03076 = 0.08151$$

So FB = 2. Therefore, the output bit-width is allocated as (IB, FB) = (4, 2).

B. Algorithm Design to Find Range

Figure 4 describes the algorithm how to confirm whether the implementation causes overflow. The algorithm first parses the implementation and corresponding bit-widths for variables from an external file (Step 1). Then the algorithm prepares to preprocess the original polynomial. The subroutine Separate divides input variables into two sets S_c and S_{nc} (Step 2). The symbol *m* represents the number of total input variables and the symbol *t[i]* records emerged times of the variable *v[i]*. The subroutine Partition divides the original polynomial into two sub-polynomials in terms of the two sets (Step 3). After preprocessing, the algorithm invokes the subroutine Get_noncorr_max to compute the maximum value of P_{nc}. If coefficients of the terms in P_{nc} are positive, each variable is set to its maximum

```
Find_range (f, imp)
1. { (in_bit, out_bit) = Parse (imp);
2.   (S_c, S_nc) = Separate (f);      // preprocessing
3.   (P_c, P_nc) = Partition (f, S_c, S_nc);
4.   V_nc_max = Get_noncorr_max (P_nc, in_bit);
5.   V_nc_min = Get_noncorr_min (P_nc, in_bit);
6.   Set_corr_poly_max (P_c);
7.   V_c_max = Get_corr_max (P_c, in_bit);
8.   Set_corr_poly_min (P_c);
9.   V_c_min = Get_corr_min (P_c, in_bit);
10.  upper_bound = V_nc_max + V_c_max;
11.  lower_bound = V_nc_min + V_c_min;
   }
Separate (f)
{ loop all terms in f
  { count times of each variable t; }
    for (i=0; i<m; i++)
        if (t[i] >1)   v[i] ∈ S_c ;
        else  v[i] ∈ S_nc;
  return (S_c, S_nc); }
Partition (f, S_c, S_nc)
{ loop all terms in f
    { if (all variables in a term ⊆ S_nc)  term ∈ P_nc;
      else  term ∈ P_c;  }
   return {P_c, P_nc}  }
Get_noncorr_max (P_nc, in_bit)
{ loop all terms
  { if (coeff > 0)
      { for (i=0; i<word_var; i++)
          { for (j=0; j< in_bit[v_i]; j++)
              var_value += pow (2, -j-1);
            temp_value *= pow (var_value, exponent); }
        V_nc_max += coeff * temp_value;   }
    }
   return V_nc_max;   }
Set_corr_poly_max (P_c)
{ for (i=0; i<m; i++)
  { loop all terms and record signs of v_i;
    if (all signs are positive)
    {   for (j=0; j< in_bit[v_i]; j++)
          v_i += pow (2, -j-1);   }
    else if (all signs are negative)   v_i = 0;
    else reserve v_i;  }  }
Get_corr_max (P_c, in_bit)
{  AT(P_c) = Convert (P_c, in_bit);
   V_c_max = Search (AT(P_c), in_bit);
   return V_c_max;  }
```

Figure 4. Algorithm to find the range for a polynomial

value according to its bit-width; if not, it is set to "0", so the maximum value V_{nc_max} of the non-correlative polynomial is obtained (Step 4). Calculation of the minimum value of the non-correlative polynomial is an opposite procedure, that is, if the term coefficient is negative, variables are set to their minimum values.

After that, the algorithm handles the correlative polynomial. The case is more complex than the process of the non-correlative polynomial. It invokes a subroutine Set_corr_poly_max to simplify the correlative polynomial. If the same variables have same signs of coefficients, it can be set a fixed value as its maximum value or "0"; if not, the variable is reserved, so the correlative polynomial is simplified.

TABLE 1.
PERFORMANCE OF THE OVERFLOW VERIFICATION ALGORITHM

Cases	Input Bit	Obtained Range	Output Bits	Time (s)	Space (MB)	Simul Time(s)	Time Saving	AA Range	AA Bits	Value Ratio
Cheby	16	-0.998,0.998	1, 11	0.09	0.27	0.12	22.5%	-1.16,1.13	2,11	12.7%
Cheby	20	-1, 0.999	1, 12	1.05	0.46	1.65	36.3%	-1.21,1.17	2,12	16%
Cheby	24	-1, 0.99997	1, 14	14.9	0.86	25.7	42%	-1.22,1.19	2,14	17%
Cheby	28	-1, 0.99999	1, 15	203	0.95	368	44.8%	-1.22, 1.2	2,15	17.4%
Dickson	10, 10	-1.994, 7.97	4, 7	1.11	1.93	1.79	38%	-2.43, 8.92	5, 7	12.2%
Dickson	12, 14	-1.998, 7.996	4, 9	59	10.3	105.3	44%	-2.56, 9.18	5, 9	14.9%
Dickson	10, 22	-1.994, 7.988	4, 8	126	11.2	>4000	>96%	-2.44, 9.07	5, 8	13.3%
Dickson	9, 26	-1.988, 7.976	4, 6	218	2.1	-----	>99%	-2.38, 8.98	5, 6	12.5%
Multivar1	12,12,12,12	-3.99, 8.99	5, 8	0.06	0.18	-----	>99%	-4.82, 10.38	5, 8	14.6%
Multivar1	16,16,16,16	-3.998, 8.997	5, 12	0.11	0.27	-----	>99%	-5.08, 10.55	5, 12	16.9%
Multivar1	20,20,20,20	-3.999, 8.999	5, 15	0.81	0.44	-----	>99%	-5.11, 10.61	5, 15	17.3%
Multivar2	8, 8, 8	-163840, 260480	19	0.05	0.16	21.8	>99%	-163840, 262144	20	0.63%
Multivar2	12,12,12	-41943040, 67082240	27	0.12	0.2	-----	>99%	-41943040, 67108864	28	0.04%
Multivar2	14, 14, 16	-2348744705, 2482749440	33	0.28	0.25	-----	>99%	-2348744705, 2483027968	33	0.01%

The conversion algorithm is invoked to convert the simplified polynomial to AT, and the search algorithm finds its maximum value (Step 7). The upper bound and the lower bound of the original polynomial are obtained by addition of corresponding values of the non-correlative polynomial and the correlative polynomial (Step 10 and 11). The algorithm has two advantages. Compared to simulation, the algorithm tries to directly set variable values as much as possible, then it reduces complexity of conversion and branch-and-bound search, by which the efficiency can be hugely improved. Compared to IA and AA, the algorithm does not separate range and precision, and handle them together. Furthermore, it does not neglect the correlation between terms, and processes it cautiously, so the obtained range can match the exact range.

IV. EXPERIMENTAL RESULTS

We implement the algorithm of overflow verification by C++. The benchmarks are described by Verilog HDL included the polynomial representation and bit-widths information. To verify its performance, we try several benchmarks. Experiments are done on a 512MB, 2.4GHz Intel Celeron machine under Linux.

1) Chebyshev Polynomial

Chebyshev polynomials are a sequence of orthogonal polynomials which are related to de Moivre's formula and which are easily defined recursively. The Chebyshev polynomials of the first kind are defined by the recurrence relation ($X \in [0, 1]$):

$T_0(X) = 1 \quad T_1(X) = 1 \quad T_{n+1}(X) = 2XT_n(X) - T_{n-1}(X)$

According to the relation, we get $T_3(X) = 4X^3 - 3X$. We use the polynomial with 3th order as a benchmark in the experiments.

2) Dickson Polynomial

Dickson polynomials have important applications in coding and communication areas. The definition for $n>0$ is (X, $a \in [0, 1]$):

$D_0(x, a) = 2$

$$D_n(x, a) = \sum_{p=0}^{[n/2]} \frac{p}{n-p} \binom{n-p}{p} (-a)^p x^{n-2p}$$

The polynomial contains two word-level variables. Here we verify the 6th order polynomial over real numbers:

$D_6(x, a) = x^6 - 6ax^4 + 15a^2x^2 - 2a^3$

3) Multivariate Polynomial

A datapath is always expressed by a polynomial with multiple variables which is most difficult to verify. The first polynomial is:

$f_1 = 2B^3 + 3A^2B - 4A^3 + 5CD$

It comprises four variables which are all fractional numbers and the total input bit-width will be beyond 32 bits which make simulation very low performance almost infeasible.

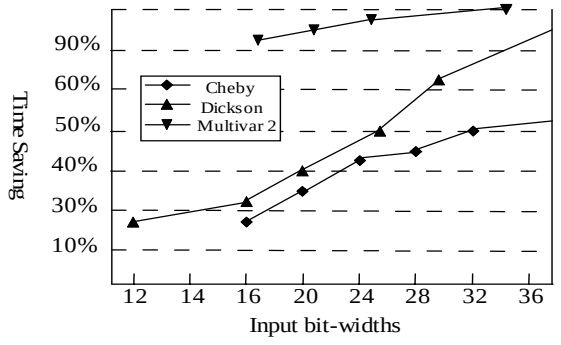
The second polynomial with three signed integer variables is $f_2 = 3A^2 - 6AB - 7BC$.

Table 1 shows the performance of the algorithm. Column 2 is the input bit-widths for single variable or multiple variables and Column 3 shows the obtained range. Obtained output bit-widths are listed in Column 4, and the first number represents IB and the second number is FB in Row 2 -12. Row 13 -15

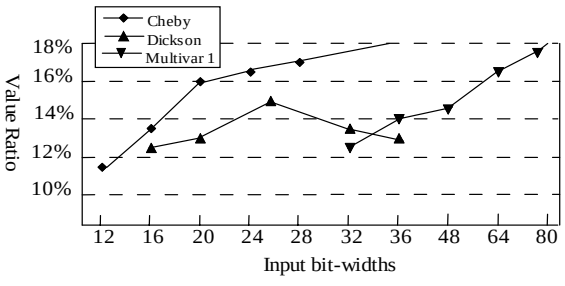
only has IB because there are no factional results. Column 5 and 6 describe the execution time and required memory. Column 7 – 11 denote the performance by simulation and AA corresponding to the same benchmarks.

It is obvious that the execution time of our method is far smaller than simulation time, especially for polynomials with multiple variables such as Row 8 - 15. The AA ranges are looser and may generate additional bit as Row 2 – 9 and Row 13 and 14. The value ratios calculated by obtained ranges dividing AA ranges are around 10% - 20% for FB and consistent with the results reported by the work [6] which finds the exact range is around 12% tighter than AA range. The benchmarks cover various polynomials with single and multiple word-level variables, so the algorithm has wide applications. The obtained range is easily calculated by the algorithm so it is convenient to determine the output bit-widths, and the performance of time and space are satisfied. Even though the examples include up to 80 bit inputs such as Row 12, the processing time is manageable and the memory required to execute the algorithms is only up to several megabytes.

Figure 5 describes the comparison of time saving and the value ratio respectively in a) and b). The running time has most difference for the multivariate polynomial and with increase of input bit-widths. Also, the algorithm can support any format of fixed-point number, so it can be applied to all fixed-point circuits.



a)



b)

Figure 5. Comparison of time and value between our method and simulation and AA

The theory of SAT-Modulo in [10] can refine the integer range. Table Ⅱ compares AA, SAT-Modulo and our method for same benchmarks. A rational function in [10] has one word-level variable and it is defined as:

$q_1 = 25t^2+125$ $q_2 = t^2+1$ $z_1 = q_1/q_2$

$q_3 = -200t$ $q_4 = q_2^2$ $z_2 = q_3/q_4$

here $-100 \leqslant t \leqslant 100$.

TABLE 2.
AA VS. SAT-MODULO VS. OUR METHOD FOR A RATIONAL FUNCTION

Out	AA	SAT-Modulo	Our Method
q_1	[125 , 250125]	[124 , 250126]	[125, 250125]
q_2	[1 , 10001]	[0 , 10002]	[1, 10001]
q_3	[-20000, 20000]	[-20001, 20001]	[-20000,20000]
q_4	[-24999999, 100020001]	[0, 100020008]	[1, 100020001]
z_1	[-250, 369]	[24, 126]	[25.01, 125]
z_2	[-67, ∞]	[-67, 67]	[-66.67, 66.67]

We can see that AA obtains coarsest range, and SAT- Modulo gets more precise results, but it only processes integer variables and is hard to obtain fractional results. Our method can get the tighter range and handle both integer and fractional numbers.

V. CONCLUSION

The datapath is often represented by a polynomial so exploring its range is helpful from both correctness and the performance point of view. Range analysis relies on simulation or static methods, but they both have weakness of low efficiency and loose bounds. In this paper, we describe the efficient static analysis to handle polynomials, and design a new algorithm to compute the range bounds by the efficient AT polynomial search. The method does not rely on simulation and handles efficiently for large arithmetic circuits. The output bit-widths are allocated by the range easily and procures smaller bits.

REFERENCES

[1] A. Gaffar, O. Mencer, W. Luk, and P. Cheung, "Unifying bit-width optimisation for fixed-point and floating-point designs," in *Proc. IEEE Symp. Field-Programmable Custom Comput*. Mach, FCCM 2004, pp. 79–88.

[2] A. Nayak, M. Haldar, A. Choudhary, P. Banerjee, "Precision and error analysis of Matlab applications during automated synthesis for FPGAs," *Proc. DATE*, 2001, pp. 722–728.

[3] C. Shi and R. Brodersen, "Automated fixed-point data-type optimization tool for signal processing and communication systems," *Proc. Design Automation Conf*. 2004, pp. 478–483.

[4] K. Kum and W. Sung, "Combined word-length optimization and highlevel synthesis of digital signal processing systems," *IEEE Trans. CAD* Vol. 20, No. 8, Aug. 2001, pp. 921–930.

[5] C. Fang, R. Rutenbar, and T. Chen, "Fast, accurate static analysis for fixed-point finite-precision effects in DSP designs," *ICCAD03*, pp. 275–282.

[6] D.-U. Lee, A. Gaffar, R. C. C. Cheung, O. Mencer, W. Luk, and G. Constantinides, "Accuracy-Guaranteed

Bit-Width Optimization", *IEEE Trans. CAD*, Vol. 25, No. 10, Oct. 2006, pp. 1990 –2000.
[7] W.G. Osborne, J. Coutinho, R. Cheung, W. Luk, O. Mencer, "Instrumented Multi-Stage Word-Length Optimization", *Proc. Field-Programmable Technology*, Dec. 2007, pp. 89 – 96
[8] Parhami B.; "Zero, sign, and overflow detection schemes for generalized signed-digit arithmetic", Signals, Systems and Computers, 1988, Twenty-Second Asilomar Conference on, Volume 2, Oct.31-Nov.2, 1988 Page(s): 636-639
[9] G. Constantinides, P. Cheung, and W. Luk, "Wordlength optimization for linear digital signal processing," IEEE Trans. on CAD vol. 22, no. 10, pp. 1432–1442, Oct. 2003.
[10] Kinsman, A.B.; Nicolici, N.; "Finite Precision bit-width allocation using SAT-Modulo Theory", *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09.* 20-24 April 2009 Page(s):1106 - 1111
[11] K. Radecka and Z. Zilic, "Arithmetic Transforms for Compositions of Sequential and Imprecise Datapaths", *IEEE Trans. CAD*, Vol. 25, No. 7, Jul. 2006, pp. 1382-1391.
[12] Pang, Yu; Radecka, Katarzyna; Zilic, Zeljko; "Arithmetic Transforms of Imprecise Datapaths by Taylor Series Conversion", *ICECS '06*. 10-13 Dec. 2006, pp.696 – 699
[13] J. Stolfi, L.H. de Figueiredo, "An Introduction to Affine Arithmetic", *TEMA Tend. Mat. Apl. Comput*.,4, No. 3 (2003), 297-312.
[14] D. Menard, and O. Sentieys, "A methodology for evaluating the precision of fixed-point systems," *Proc. IEEE Int. Conf on Acoust.,Speech, and Signal Processing*, vol. 3,2002. pp. 3152-3155.
[15] H. Choi and W. P. Burleson, "Search-based wordlength optimization for VLSI/DSP synthesis," in *Proc. VLSI Signal Processing*, La Jolla, CA, 1994, pp. 198–207.
[16] M.-A. Cantin, Y. Savaria, D. Prodanos, and P. Lavoie, "An automatic word length determination method," *ISCAS*, Sydney, Australia, 2001, vol. 5, pp. 53–56
[17] W. Sung and K. I. Kum, "Simulation-based wordlength optimization Method for fixed-point digital signal processing systems," *IEEE Trans. Signal Processing*, vol. 43, pp. 3087–3090, Dec. 1995.
[18] F. Catthoor, J. Vandewalle, and H. De Man, "Simulated annealing-based optimization of coefficient and data word-lengths in digital filters," Int. J. Circuit Theory Applicat., vol. 16, pp. 371–390, Sep. 1988
[19] O. Sarbishei, B. Alizadeh and M. Fujita, "Polynomial Datapath Optimization Using Partitioning and Compensation Heuristics", *DAC*, 2009.
[20] L. de Figueiredo and J. Stolfi, "Self-validated numerical methods and applications," in Brazilian Mathematics Colloquium Monograph. Rio de Janeiro, Brazil: IMPA, 1997. Available: http://www.ic.unicamp.br/~stolfi/ EXPORT /bibliography/FromBib.html#fig-sto-97-iaaa
[21] P. S. Wang, "An improved multivariate polynomial factoring algorithm", *Mathematics of Computation*, Vol. 32, No. 144, Oct. 1978, pp. 1215- 1231
[22] A. Ahmadi and M. Zwolinski, "Symbolic noise analysis approach to computational hardware optimization", *Design Automation Conference*, 2008,8-13 June 200,8 pp. 391 – 396
[23] Fei Kang, Junjie Li, Zhenyue Ma, Haojin Li, "Artificial Bee Colony Algorithm with Local Search for Numerical Optimization", *Journal of Software*, Vol 6, No 3, 2011, pp. 490-497.
[24] Chunna Zhao, Yu Zhao, Yingshun Li, Liming Luo, "Fractional Correlation Distance Method on Course Evaluation", *Journal of Software*, Vol 6, No 10, 2011, pp. 2016-2022

Yu Pang He received his Ph.D of microelectronics from McGill University (Canada) in 2010. His primary research area focused on ASIC design, logic synthesis, wireless sensor networks and bioelectronics. Currently he is an associate professor of the College of Electronic Engineering, Chongqing University of Posts and Telecommunications, Chongqing, China.

Yafeng Yan He received his BSc in Communication Engineering from Xi'an Aeronautical University in 2010. His primary research area focused on embedded system and hardware design. Currently he is a master student in College of Electronic Engineering, Chongqing University of Posts and Telecommunications, Chongqing, China.

Junchao Wang He will receive his bachelor degree from Chongqing University of Posts and Telecommunications, in July 2013. His primary research area is logic synthesis and ASIC design.

Zhilong He He received his BSc in Communication Engineering from Chongqing University of Posts and Telecommunications in 2010. His primary research area focused on ASIC design and embedded system. Currently he is a master student of School of Communication and Information Engineering, Chongqing University of Posts and Telecommunications, Chongqing, China.

Ting Liu He received his Ph.D and post-doctor degree of microelectronics from Université de Toulouse (France) in 2009 and 2010. His primary research area focused on ASIC design and hardware verification. Currently he is a senior engineer in Aquart Creation Cooperation, Montreal, Quebec, Canada.