An Efficient Hybrid Engine to Perform Range Analysis and Allocate Integer Bit-widths for Arithmetic Circuits

Yu Pang
College of Photo-Electronics
Chongqing University of Posts and Telecommunications
Chongqing, 400065, China
pangyu@cqupt.edu.cn

Katarzyna Radecka, Zeljko Zilic
Dept. of Electrical and Computer Engineering
McGill University
Montreal, Quebec H3A 2A7, Canada
{katarzyna.radecka, zeljko.zilic}@mcgill.ca

Abstract-Range analysis is an important task in obtaining the correct, yet fast and inexpensive arithmetic circuits. The traditional methods, either simulation-based or static, have the disadvantage of low efficiency and coarse bounds, which may lead to unnecessary bits. In this paper, we propose a new method that combines several techniques to perform fixed-point range analysis in a datapath towards obtaining the much tighter ranges efficiently. We show that the range and the bit-width allocation can be obtained with better results relative to the past methods, and in significantly shorter time.

I. Introduction

Since designers increasingly replace ASICs with FPGAs, which are not well-suited for floating-point arithmetic even if non-binary [1], there is a renewed interest in fixed-point arithmetic. Allocating bit-widths in a datapath has a direct impact on resources and circuit speed. The fixed-point arithmetic entails the precision and the range problem, where we focus on the latter. Manual or sub-optimal range analysis might result in misallocated bit-widths; too few bits cause overflow, while too many are costly. As many FPGA applications are in reconfigurable and parallel computing [2], where many optimizations are needed, the speed of range analysis is important.

To obtain the optimal allocation of bit-widths, a data representation that exposes well the variable ranges plays a key role. If we can find the exact ranges for all intermediate variables, we can achieve the smallest bit-widths, leading to a reduction in the circuit area and the delay. To perform range analysis, a simulation-based *dynamic analysis* [3-4] is common.

Although dynamic analysis is conceptually straightforward, the inherent low efficiency of exhaustive simulations confines them to small datapaths. Static analysis attempts to determine the range, often approximately, without simulations. *Interval arithmetic* (IA) is a usual approximate method to calculate the value bound, but it unavoidably leads to coarse results. The *affine arithmetic* model (AA) is a derivation of IA, in which the quantities of interest are represented as linear combinations (affine forms) of certain primitive variables standing for sources of uncertainty in the data or for approximations made during the computation. Work in [5] adopts AA to investigate integer bit-widths and implement FPGA by different bit-widths.

Other static methods include these based on *saturation arithmetic* [6], AT [7] and the *SAT-Modulo Theory* (SMT) [8]. Authors in [9] adopt a spectral technique, *Arithmetic Transform* (AT), to explore precision in the imprecise representation of Taylor series and real-valued polynomials. The above methods show that conventional approaches such as SAT and spectral ones used elsewhere [10] can be easily extended from handling precision [11] to the range analysis.

Fig. 1 compares the time requirement for each method. Static methods can lead to over-allocation of integer bit-widths (IBs) and, in consequence, enlarged area requirements. Dynamic methods can exhibit a double-sided error, including underestimating a range, while all other methods only err on the side of being conservative.

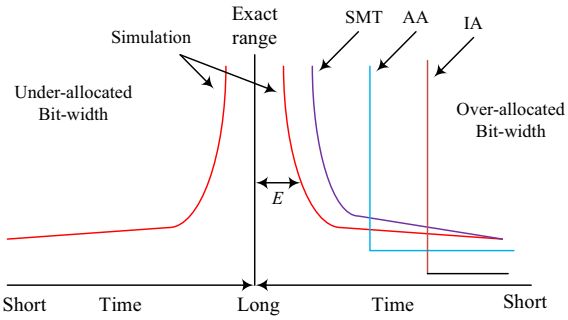

Fig. 1. Tradeoff between ranges and calculation times

In the range analysis so far, there was a clear separation among the solutions that focus on the quality of the result and those where the computation time has been the goal, without the explicit possibility to exploit well the specifics of a given problem. Dynamic methods and SMT focus on tight ranges, while IA and AA are designed to shorten the calculation time.

In this paper, we develop an efficient hybrid engine, which combines advantages of IA, AA and AT. Our solution can get tighter ranges, while reducing the calculation time. This is achieved through analyzing the correlation between variables, which then lends itself to the smallest bit-widths for a given (sub-)problem. We note that among the existing methods only the AT-based technique can obtain the tight range, while at the same time is being able to measure the distance to the precise solution, so we will capitalize on that.

II. Background

Definition 1: The error bound labeled as E in Fig. 1 is the largest difference between the exact and the obtained ranges, while the error ratio e_r calculated as $\left(\frac{\text{obtained range}}{\text{exact range}} - 1\right) * 100\%$ represents the effective size of the obtained range. ■

The objective is to find the smallest value of E and e_r whilst maintaining the one-sided error, i.e., not underestimating the bit-width. Towards this goal, two static methods are introduced in the following sections that rely on interval and affine arithmetic.

2.1Intervaland Affine Arithmetic

The traditional static analysis is based on IA and AA. IA defines a set of operations on intervals. An operation <OP>

978-1-4244-7516-2/11/$26.00 ©2011 IEEE

on two intervals with <OP> is defined by:
$[x_1, x_2]$ <OP> $[y_1, y_2]$ = { x<OP>y | $x \in [x_1, x_2], y \in [y_1, y_2]$ }

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

$$x_0 = \frac{x_{max} + x_{min}}{2}, \quad x_1 = \frac{x_{max} - x_{min}}{2} \qquad (1)$$

The intermediate signal or the output is represented as a first degree polynomial: $y_A = y_0 + y_1\varepsilon_1 + y_2\varepsilon_2 \ldots + y_n\varepsilon_n$ (2) where $y_0, y_1, \ldots y_n$ are real-valued numbers and $\varepsilon_1, \varepsilon_2, \ldots \varepsilon_n$ are symbolic *uncertain variables* whose values are not exact, but only known to lie in the range [-1,+1].

Example 1: Using IA and AA to calculate ranges. Consider a datapath represented by a polynomial $z=ab+c-b$ and the ranges of primary inputs as shown in square brackets in Fig.2.

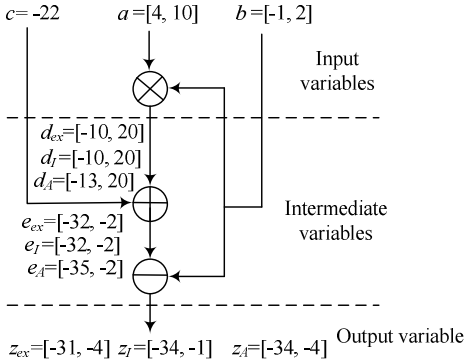

Fig. 2. An example performing range analysis by IA and AA

Fig. 2 describes the exact ranges (subscript *ex*), as well as the ranges calculated by IA (subscript *I*) and AA (subscript *A*) respectively. Both error bounds for the primary output z are computed as -31-(-34)=3, and the error ratios are $e_{r\_I} = \left(\frac{-1+34}{-4+31} - 1\right) * 100\% = 22.2\%$ and $e_{r\_A} = 11.1\%$. Therefore, the range by AA is better than that of IA. ■

Note that the error bounds and ratios in Example 1 are based on the known exact ranges, as IA and AA cannot obtain them by themselves. The intermediate variable e_A (AA method) as well as the primary outputs z_I *and* z_A calculated by IA and AA must be represented by 7 signed integer bits. However, 6 bits are enough for the exact ranges to represent e and z since their exact ranges are [-32, -2] and [-31, -4].

Definition 2: Correlation is present if at least two monomials in a polynomial have in their support the same variable. ■

The correlation in two monomials means that if the value of one monomial changes, the other will follow the change. Clearly, the correlation may lead to the overestimation of range because the two monomials might not reach their maximum or minimum values at the same time, so handling the correlation becomes a key task in range analysis.

2.2 Arithmetic Transform

Definition 3: The ***Arithmetic Transform (AT)*** [9] is a polynomial representing a pseudo Boolean function $f: B^n \rightarrow Z$ with an arithmetic operation "+", word-level coefficients c, binary inputs $x_1, x_2 \ldots x_n$ and binary exponents $i_1, i_2 \ldots i_n$:

$$AT(f) = \sum_{i_1=0}^{1} \sum_{i_2=0}^{1} \ldots \sum_{i_n=0}^{1} (c_{i_1 i_2 \ldots i_n} * x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n}) \qquad (3)$$ ■

By Def. 3 and Eqn. (3), it is easy to conclude that AT is a linear transformation, that is:

$$\begin{aligned} AT(f_1 + f_2) &= AT(f_1) + AT(f_2) \\ AT(const * f_1) &= const * AT(f_1) \\ AT(f^n) &= (AT(f))^n \end{aligned} \qquad (4)$$

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

$$AT[f(X,Y)] = AT(2X^3 + Y^2) = AT(2X^3) + AT(Y^2)$$

$$= 2AT(X)^3 + AT(Y)^2 = 2 * \left(\sum_{i=0}^{1} 2^i x_i\right)^3 + \left(\sum_{k=0}^{2} 2^k y_k\right)^2$$

$$= 16x_1 + 36x_1x_0 + 2x_0 + 16y_2 + 16y_2y_1 + 8y_2y_0 + 4y_1 + 4y_1y_0 + y_0$$

After an exact polynomial is converted to AT, we need to calculate its maximum and minimum values for the range and precision analysis. To achieve this goal, the AT generation algorithm and a branch-and-bound searching method have been fine tuned in [11] to efficiently handle the intermediate expression swell during the calculation of the extreme values of the AT polynomial. We note that since the uncertain variable ε in AA normalizes to [-1, 1], AT may easily represent it by a signed fractional number.

Based on the above analysis, we propose a hybrid method for range calculations, which combines IA, AA and AT. The above mathematical forms describe the fact that AA can get a tighter range than IA if the datapath has correlation such as the final output z (Example 1). Otherwise, IA is better, as illustrated by the intermediate variables d and e. Furthermore, AA can represent the arbitrary input range compactly while AT might not, so the input is better to be expressed by AA. For the intermediate variables, since the uncertain variable ε in AA normalizes to [-1, 1], AT may easily represent it by a signed fractional number, and hence is a useful tool to determine a suitable value of ε.

Hence, the advantages of IA, AA and AT are complementary and can be used together, as long as they are employed in suitable environments. We next present a hybrid algorithm for the static range analysis and the bit-width optimization.

III. Hybrid Range Analysis Engine

In this section, we describe the hybrid engine to allocate IBs for a fixed-point datapath and use an example to clearly explain it.

3.1 Description of the Hybrid Engine

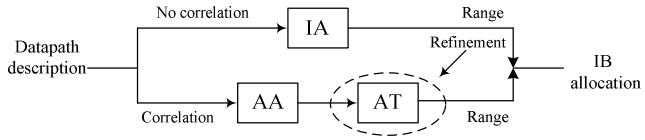

Fig. 3. Abstract model of the proposed algorithm

Fig. 3 illustrates the way that the algorithm invokes different methods to handle a datapath. Furthermore, it can distribute correlation to AA and AT for the two-step processing, in which AA partly handles correlation first, and AT continues to refine the results. In contrast to ours, the

SMT-based method is time consuming, as it invokes underlying exhaustive engine pretty much all the time to refine the initial IA ranges, Fig. 4.

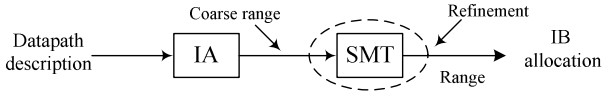

Fig. 4. SMT model to perform range analysis

Fig. 5 summarizes our algorithm for allocating integer bit-widths in a datapath. It first retrieves the polynomial description representing a datapath (Step 1) and generates the AA expression for future utilization (Step 2). If the polynomial has no correlation, IA is used to compute the exact range so the bit-widths will be determined (Steps 4 and 5); otherwise, the uncertain variables are quantized in the AA expression (Step 6), and the conversion algorithm is invoked to convert the expression to its AT form (Step 7). Then, the branch-and-bound searching algorithm is applied to find the upper and the lower bounds, and estimate the bound intervals (Step 8). Finally, the IBs of the datapath are allocated (Steps 9 and 10).

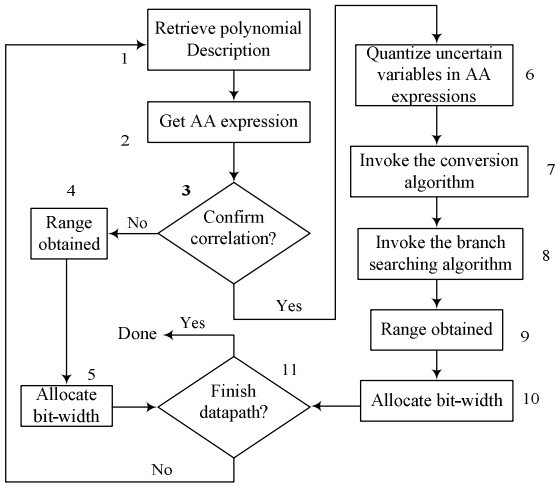

Fig. 5. Algorithm for allocating IBs in a datapath

In order to clearly explain the propose hybrid engine in Fig. 5, we apply it for Example 1 to calculate ranges in the following section.

3.2 Datapath Analysis

3.2.1 AA Expressions of the Outputs

The first task is to express the output range by AA. The datapath of Example 1 has three primary inputs, two intermediate outputs and one primary output. The three primary variables *a*, *b* and *c* are represented by Eqn. (1) as:

$$a_A = 7 + 3\varepsilon_1 \qquad b_A = 0.5 + 1.5\varepsilon_2 \qquad c_A = -22$$

The first intermediate variable is $d = ab$. Confirming no correlation in the polynomial is easy for only one monomial, so the range of *d* can be determined by IA as [-10, 20]. The AA expression is

$$d_A = a_A b_A = 3.5 + 1.5\varepsilon_1 + 10.5\varepsilon_2 + 4.5\varepsilon_2\varepsilon_1$$

The next intermediate variable in the datapath is $e=ab+c$. Again, there is no correlation because the two monomials "*ab*" and "*c*" do not include a same variable, so the range of *e* calculated by IA is [-32, -2]. The AA expression of *e* is:

$$e_A = d_A + c_A = -18.5 + 1.5\varepsilon_1 + 10.5\varepsilon_2 + 4.5\varepsilon_2\varepsilon_1$$

Finally, we determine the range of the primary output $z=ab+c-b$. As the variable *b* occurs two times, the two monomials of "*ab*" and "*-b*" are correlated making the case is much more complex. The AA expression of *z* is then:

$$z_A = e_A - b_A = -19 + 1.5\varepsilon_1 + 9\varepsilon_2 + 4.5\varepsilon_2\varepsilon_1 \quad (5)$$

3.2.2 Determining Quantization Bits of Uncertain Variables

As ε_2 and ε_1 belong to [-1, 1], AT can represent the scope approximately by an *m*-bit signed fractional number, where the contribution of each bit is shown in Table 1.

sign	0.5	0.25	0.125
x_0	x_1	x_2	x_3

Table 1.Format for representing a signed factional number

Let $\tilde{\varepsilon}_1$ and $\tilde{\varepsilon}_2$ be quantized uncertain variables to replace ε_1 and ε_2. Their corresponding AT representations are:

$$AT(\tilde{\varepsilon}_1) = (1 - 2x_0)\sum_{i=1}^{m_1} 2^{-i}x_i$$
$$AT(\tilde{\varepsilon}_2) = (1 - 2y_0)\sum_{k=1}^{m_2} 2^{-k}y_k \quad (6)$$

Here m_1 and m_2 are integers representing the quantization bits for $\tilde{\varepsilon}_1$ and $\tilde{\varepsilon}_2$ respectively. Note that compared to ε_1 and ε_2, $\tilde{\varepsilon}_1$ and $\tilde{\varepsilon}_2$ cannot be equal to -1 and 1 but only approximate these border values, that is, $-1 < (\tilde{\varepsilon}_1, \tilde{\varepsilon}_2) < 1$. If we can determine the values of m_1 and m_2, the output is represented compactly and the approximation error can be estimated. Hence, the next step is to calculate the suitable bit-widths for the uncertain variables.

The worst case occurs if the approximation error is beyond 1 for an integer number, when it is possible to generate an additional bit, Eqn. (7).

$$|z_{ex} - z| < 0.5 \Rightarrow z - 0.5 < z_{ex} < z + 0.5 \quad (7)$$

Based on the obtained value *z*, we can estimate a scope [*z*-0.5, *z*+0.5] for the exact output z_{ex}, and the approximation error will be limited in 1. Using the AA form z_A and two quantization variables $\tilde{\varepsilon}_1$ and $\tilde{\varepsilon}_2$, Eqn. (7) reduces to:

$$|(1.5\varepsilon_1 - 1.5\tilde{\varepsilon}_1) + (9\varepsilon_2 - 9\tilde{\varepsilon}_2) + (4.5\varepsilon_2\varepsilon_1 - 4.5\tilde{\varepsilon}_1\tilde{\varepsilon}_2)| < 0.5 \quad (8)$$

By the triangle inequality, if the following equation can be satisfied, then Eqn. (8) is also satisfied:

$$|(1.5\varepsilon_1 - 1.5\tilde{\varepsilon}_1)| + |(9\varepsilon_2 - 9\tilde{\varepsilon}_2)| + |(4.5\varepsilon_2\varepsilon_1 - 4.5\tilde{\varepsilon}_1\tilde{\varepsilon}_2)| < 0.5$$
$$\Rightarrow |1.5\tilde{\varepsilon}_1|_{err} + |9\tilde{\varepsilon}_2|_{err} + |4.5\tilde{\varepsilon}_1\tilde{\varepsilon}_2|_{err} < 0.5 \quad (9)$$

$|1.5\tilde{\varepsilon}_1|_{err}$, $|9\tilde{\varepsilon}_2|_{err}$ and $|4.5\tilde{\varepsilon}_1\tilde{\varepsilon}_2|_{err}$ represent error bounds of the three monomials respectively. The monomial error bound is defined as $|1.5\tilde{\varepsilon}_1|_{err} = max(|1.5\varepsilon_1 - 1.5\tilde{\varepsilon}_1|)$. Note that Eqn. (9) uses absolute values for each monomial error instead of actual ones. In some specific cases, all monomial errors have negative values, and hence contribute the most to the overall error. Although monomial errors with opposite polarity may be counteract leading to the smaller overall error than the addition of absolute monomial errors, Eqn. (9) avoids specific cases and guarantees that the obtained error cannot exceed the assumed one.

As it is impossible to address all monomial errors concurrently, we process each monomial case individually. Hence the assumed error value for the first processed monomial is the whole error space, that is, "0.5":

$$|4.5\tilde{\varepsilon}_1\tilde{\varepsilon}_2|_{err} < 0.5 \Rightarrow 4.5[1 - (1 - 2^{-m+1})^2] < 0.5 \quad (10)$$

The reason to choose the monomial $4.5\tilde{\varepsilon}_1\tilde{\varepsilon}_2$ as the first one is because it has an uncertainty degree "2" while for the remaining monomials $1.5\tilde{\varepsilon}_1$ and $9\tilde{\varepsilon}_2$, this degree is one. The preferential choice of the monomial with the highest uncertainty degree is helpful to decrease the calculation time. Obviously, when all bits in the data format are "1" except the

sign bit, the fractional number has the largest approximation error 2^{-m+1}. For instance, using four bits in Table 1 to represent the value "1" or "-1", when x_1, x_2 and x_3 are all set "1", the maximum error is $2^{-3} = 0.125$. If the representation of other values is restricted to the interval [-1, 1], then the error changes to $2^{-4} = 0.0625$. When the monomial reaches the maximum error, the corresponding values of $|\varepsilon_1\varepsilon_2|$ and $|\tilde{\varepsilon}_1\tilde{\varepsilon}_2|$ are equal to 1 and $(1\text{-}2^{-m+1})^2$ respectively. As there are no additional restrictions on individual variables, we assume that $\tilde{\varepsilon}_1$ and $\tilde{\varepsilon}_2$ have same bit-widths *m*. Therefore, the maximum error $|4.5\tilde{\varepsilon}_1\tilde{\varepsilon}_2|_{err}$ is represented as $4.5[1\text{-}(1\text{-}2^{-m+1})^2]$.

The value *m* is obtained to be 6 by solving Eqn. (10), which means that $\tilde{\varepsilon}_1$ and $\tilde{\varepsilon}_2$ both require at least six bits to satisfy Eqn. (10). The maximum fractional value represented by six bits is 0.96875, hence by substituting $\tilde{\varepsilon}_1$=$\tilde{\varepsilon}_2$= 0.96875, the real value $max(4.5\tilde{\varepsilon}_1\tilde{\varepsilon}_2)$ is 4.5 * 0.96875^2 = 4.223. This results in the obtained maximum error of 4.5 – 4.223 = 0.277 for the monomial "$4.5\varepsilon_2\varepsilon_1$". Therefore, the value of $|1.5\tilde{\varepsilon}_1|_{err} + |9\tilde{\varepsilon}_2|_{err}$ cannot exceed the remaining error space calculated to be 0.5–0.277 = 0.223, which is referred to Eqn. (11):

$$|1.5\tilde{\varepsilon}_1|_{err} + |9\tilde{\varepsilon}_2|_{err} < 0.223 \quad (11)$$

Next we explore the monomial "$1.5\tilde{\varepsilon}_1$". Again, similarly to Eqn. (10), the assumed error for this monomial is the remaining error space described by the following equation:

$$1.5 * 2^{-m_1+1} < 0.223 \quad (12)$$

To satisfy Eqn. (12), $\tilde{\varepsilon}_1$ must be expressed using at least four bits. Hence, to fulfill the two requirements on the size of $\tilde{\varepsilon}_1$, i.e., six bits in the monomial "$4.5\tilde{\varepsilon}_1\tilde{\varepsilon}_2$" and four bits in the monomial "$1.5\tilde{\varepsilon}_1$", $\tilde{\varepsilon}_1$ must be represented by 6 bits. This results in $max(1.5\tilde{\varepsilon}_1)$ = 1.5 * 0.96875=1.4531.

The obtained maximum error for the monomial $1.5\tilde{\varepsilon}_1$ is then 1.5-1.4531=*0.0469*, and the remaining error space is 0.223-0.0469=0.1761. The final monomial $9\tilde{\varepsilon}_2$ must satisfy this error, which is indicated by the following equation:

$$9 * 2^{-m_2+1} < 0.1761 \quad (13)$$

The bit-width of $\tilde{\varepsilon}_2$ is 7 determined by Eqn. (13). This, in combination with the bit-width of 6 in the monomial $4.5\tilde{\varepsilon}_1\tilde{\varepsilon}_2$, result in the final 7-bit representation of $\tilde{\varepsilon}_2$. So we determine bit-widths for $\tilde{\varepsilon}_1$ and $\tilde{\varepsilon}_2$ as m_1=6 and m_2=7. Hence, the error bound for the monomial $9\tilde{\varepsilon}_2$= $9*2^{-7+1}$ is *0.1406*. Since the final bit-width of $\tilde{\varepsilon}_2$ is different from the obtained bit-width in the monomial $4.5\tilde{\varepsilon}_1\tilde{\varepsilon}_2$, the maximum error of this monomial is re-calculated as:

$$4.5[1 - (1 - 2^{-6+1}) * (1 - 2^{-7+1})] = 0.2087$$

Above values indicated by italic numbers denote error bounds for each uncertain monomial:

$$\begin{aligned} |1.5\tilde{\varepsilon}_1|_{err} &= 0.0469 \\ |9\tilde{\varepsilon}_2|_{err} &= 0.1406 \\ |4.5\tilde{\varepsilon}_2\tilde{\varepsilon}_1|_{err} &= 0.2087 \end{aligned} \quad (14)$$

The obtained error bound for the primary output *z* is calculated as 0.0469+0.1406+0.2087= 0.3962, Fig. 6.

The AT representation of *z* is determined by expanding $\tilde{\varepsilon}_1$ and $\tilde{\varepsilon}_2$ into their bit-level expressions:

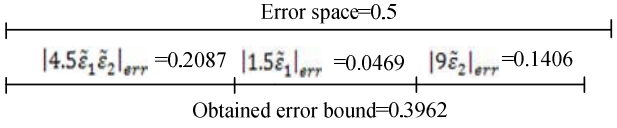

Fig. 6. Description of the final calculation

$$\begin{aligned} AT(z) = &-19 + 1.5(1 - 2x_0)\sum_{i=1}^{5} 2^{-i}x_i + 9(1 - 2y_0)\sum_{k=1}^{6} 2^{-k}y_k \\ &+4.5\left[(1 - 2x_0)\sum_{i=1}^{5} 2^{-i}x_i\right]\left[(1 - 2y_0)\sum_{k=1}^{6} 2^{-k}y_k\right] \end{aligned} \quad (15)$$

By invoking the conversion algorithm and the branch-and-bound search [11], the upper and the lower bounds for the scope of the output *z*, Eqn. (15), can be computed as -4.4 and -30.7. Based on Eqn. (14), we can conclude that the exact upper and lower bounds belong to the following intervals:

$$\begin{cases} z_{ex\_upp} = -4.4 \pm 0.3962 = [-4.7962, -4.0038] \\ z_{ex\_low} = -30.7 \pm 0.3962 = [-31.0962, -30.3038] \end{cases} \quad (16)$$

The values of upper and lower bounds of the range of outputs are estimated to lie within very tight intervals, Eqn. (16). The capability of our method to produce such estimations puts it in a clear advantage over other methods such as AA [5] and SMT [8]. Applying the ceiling function to the values in Eqn. (16), we obtain two ranges: [-31, -4] and [-32, -4]. To guarantee that there is no overflow, the range [-32, -4] is selected. The error bound between the obtained and the exact ranges in Fig. 2 is only "1" and the error ratio is 3.7%. Compared to the results of IA and AA in Example 1, both two indicators are much better. Note, that Eqn. (16) provides information of the exact bounds for the output variable *z*, and all past explorations only calculate course ranges and cannot estimate the intervals of the exact bounds leading to know the error bound and error ratio.

If the monomial $4.5\tilde{\varepsilon}_1\tilde{\varepsilon}_2$ is not chosen first, $\tilde{\varepsilon}_1$ and $\tilde{\varepsilon}_2$ need 9 and 8 signed bits, respectively. Although the obtained range of *z* is the same, the calculation time would significantly increase since more quantization bits burden the conversion and the branch searching algorithms. Hence, the first choice of the monomial with the highest uncertain degree is very significant.

3.3 Subroutine Description

Determine_uncertain (AA_Expre)

1. { loop all monomials in the AA expression
2. { if the uncertain degree in the current monomial is smaller than that of the next monomial, move forward the next monomial; } // sort monomials with higher uncertain degrees;
3. e_p = 0.5; // initial error space
4. for (*p=0; p*<monomial_num*; p*++) // loop all sorted monomials
5. { $V_q = \left(1 - \frac{e_p}{mono\_coeff}\right)$;
6. $m_p = ceiling[1 - \log_2(1 - \sqrt[d]{V_q})]$; // calculate quantization bits
7. e_p =e_p-mono_coeff * $\left[1 - (\sum_{i=1}^{m_p-1} 2^{-i})^d\right]$; // Update error space
8. store m_p for corresponding uncertain variable; }
9. for (*i*=0; *i*<uncertain_ num; *i*++) // loop all uncertain variables
10. { for (*p=0; p*<monomial_num*; p*++) // loop all obtained bit-widths for comparison
11. q_i= *max* (present in m_p); };
}

Fig. 7. Determining bit-widths for uncertain variables

The key step of quantizing uncertain variables in Fig. 5 (Step 6) is described in Fig. 7. The subroutine first sorts the monomials in the AA expression. The monomials with higher uncertain degrees would be explored first (Steps 1 – 2). Considering the worst case, the initial error space is set to be 0.5 (Step 3), and the subroutine begins to process each sorted monomial (Step 4). The quantization value V_q of one monomial is calculated in Step 5, and the bit-widths can be obtained in Step 6. Here *d* represents the uncertain degree of

Case	Out-put	Range			Bit			Error Ratio (%)		Time (s)			
		AA	SMT	Ours	AA	SMT	Ours	AA	Ours	Sim	SMT	AT	Ours
Image filter	q_1	[-350, 400]	[-1, 401]	[0, 400]	10	10	9	87.5	0	5.12	15.6	26.1	7.8
	q_2	[-8000, 7750]	[-8001, 1001]	[-8000, 1000]	14	14	14	75	0				
	q_3	[-158750,160000]	[-1, 160001]	[0, 160000]	19	19	18	99	0				
	q_4	[-511000,534000]	[-112, 512001]	[-109,512001]	21	20	20	99	<0.01				
	z	[-511000,542000]	[-2, 520001]	[-1, 520001]	21	20	20	99	<0.01				
Hermite	q_1	[-92, 100]	[-1, 101]	[0, 100]	8	8	7	92	0	9.32	52.7	83.5	33.7
	q_2	[-9036, 8948]	[-60, 8502]	[-58, 8501]	15	15	15	100	<0.01				
	q_3	[-865820, 865828]	[-42, 854501]	[-40, 854501]	21	21	21	103	<0.01				
	z	[-865835, 865813]	[-57, 854490]	[-55, 854486]	21	21	21	103	<0.01				
Dickson	q_1	[-2500, 2500]	[-1, 2501]	[0, 2500]	13	13	12	100	0	163	124	213	51.5
	q_2	[-2660, 2580]	[-162, 2582]	[-160, 2580]	13	13	13	91	0				
	q_3	[-6450000,6450000]	[-6401,6450003]	[-6401,6450001]	24	24	24	99.8	<0.01				
	q_4	[-2800, 3200]	[-1, 3201]	[0, 3200]	13	13	12	87.5	0				
	z	[-6452800,6453200]	[-6403,6453205]	[-6401,6453201]	24	24	24	99.8	<0.01				
Multi-variate polyno-mial	q_1	[-25650, 27000]	[-1, 27001]	[0, 27000]	16	16	15	95	0	>500	19	>500	9.2
	q_2	[-57000,72000]	[-48001, 72001]	[-48000, 72000]	18	18	18	7.5	0				
	q_3	[-28000, 48000]	[-16001, 48001]	[-16000, 48000]	17	17	17	18.8	0				
	q_4	[-82500, 60000]	[-45002, 60002]	[-45001, 60001]	18	17	17	35.7	<0.01				
	z	[-130500, 97000]	[-93004, 76003]	[-93001, 76001]	18	18	18	34.6	<0.01				

Table 2: AA vs. simulation vs. SMT vs. hybrid engine for various datapaths

the monomial. The subroutine updates the error space by the just calculated bit-widths m_p which is stored for the corresponding uncertain variables in Step 8. Then the subroutine processes the next monomial. After the loop terminates, the obtained bit-widths for each uncertain variable are compared, and the maximum one is chosen as the final bit-width for this uncertain variable (Steps 9 – 11). Note that the efficient AT conversion and branch-and-bound searching are instrumental to the high efficiency in performing the range analysis. Our method combines techniques of IA, AA and AT. For instance, range calculations of variables *d* and *e* use IA, while AA and AT are used for *z*. Although AA partly neglects the correlation, the quantization of uncertain variables in AA expressions helps tracing the correlation, leading to much more precise results. Therefore, our method avoids disadvantages of AA, IA and AT while emphasizing their advantages, and hence it can obtain tighter ranges faster.

IV. Experimental Results

We implemented our algorithm in C++ and applied it to several benchmarks to evaluate our method. All experiments are done on a 512MB, 2.4GHz Intel Celeron machine.

4.1 Ranges and IB Allocations

1) **Filter Polynomial**

Image processing applications often use a polynomial filter with a representation given by $z = a_1X^4 + a_2X^3 + a_3X^2$. Here we consider an example with the input X belonging to the interval $X \in [-20, 10]$:

$$F = 4X^4 + 16X^3 + 20X^2$$

The implementation has four intermediate variables:

$q_1 = X^2$ $q_2 = q_1X$ $q_3 = q_2X$ $q_4 = 4q_2+16q_3$ $z = q_4+20q_1$

2) **Hermite polynomial** is an orthogonal polynomial sequence which can be used in ultra wide-band communications. We consider the case of n =6, that is:

$$H_6(x) = z = x^6 - 15x^4 + 45x^2 - 15$$
$$= x^2(x^2(x^2 - 15) + 45) - 15 \qquad x \in [-6, 10]$$

The implementation contains three intermediate signals as:

$q_1 = x^2$ $q_2 = q_1(q_1\text{-}15)$ $q_3 = q_1(q_2+45)$ $z = q_3\text{-}15$

3) **Dickson polynomials** have applications in coding. Here we explore the implementation of the 4^{th} order polynomial over real numbers (assume $x \in [-50, 50]$, $a \in [-20, 40]$), and the goal is to find the integer ranges to allocate IBs:

$$D_4(x, a) = z = x^4 - 4x^2a + 2a^2 = x^2(x^2-4a) + 2a^2$$

The implementation has 4 intermediate variables, q_1 to q_4:

$q_1=x^2$ $q_2 = q_1\text{-}4a$ $q_3 = q_1q_2$ $q_4 = 2a^2$ $z = q_4 + q_3$

4) **Multivariate Datapath.** Some datapths realize multiple variable polynomials. For example, the following polynomial has 3 word-level variables:

$$F = 30A^2 - 60AB - 40BC$$

where $A \in [-20, 30]$, $B \in [10, 40]$ and $C \in [-10, 30]$.

The case is broken intermediately into:

$q_1 = 30A^2$ $q_2 = 60AB$ $q_3 = 40BC$ $q_4 = q_1 - q_2$ $z = q_4 - q_3$

We use AA and SMT for range comparison and adopt AA, simulation and SMT for time comparison presented in Table 2. Experimental results indicate that the ranges by SMT are close to the ranges obtained by our proposed method, which are much tighter than the ranges of AA, and the error ratios of our method are by far smaller than these of AA indicated by Column 5. In fact the values are approximately "0" which means the obtained ranges are very precise. Note that no other methods, such as AA and SMT, can calculate error ratio by itself, hence this value is based on the ranges calculated by our method, as it can automatically compute the error bound leading to estimation of the error ratio. Furthermore, AA and SMT may require one additional bit for representing some signals leading to the cost increase.

Column 6 of Table 2 describes the execution times of several past methods and our proposed method. Simulation method takes much longer time than our method for the datapaths beyond one variable such as benchmarks 3 and 4. SMT often needs a long time for computation since it is based on the first-order SAT theory. Processing high-order polynomials such as benchmarks 2 and 3 is very difficult resulting in long execution time. Since the pure AT method can require huge number of terms, it can lead to low efficiency. The execution time of our method is acceptable both for high order and multivariate polynomials. Considering the tight results and short calculation time, our method is the best overall, especially if the quick calculation is a must.

4.2 Area of Optimized Implementations

As the exact area of the resulting circuit depends on the

implementation technology, we perform further experiments based on the FPGA platform. We map the circuits to Xilinx Virtex5 FPGAs using ISE tool, version 8.1, to evaluate the real area impact of impact of the proposed algorithm in Table 3. Implementations obtained by AA are used as comparison.

Circuit	Area (Slices)			Delay (ns)		
	Ours	AA	Saving	Ours	AA	Saving
Filter	686	740	7.3%	23.5	25.4	7.5%
Filter	725	768	5.6%	24.6	26.2	6.1%
Filter	756	787	3.9%	25.4	26.8	5.2%
Hermite	809	870	7%	31.3	33.5	6.6%
Hermite	845	897	5.8%	32	33.9	5.6%
Hermite	876	919	4.7%	32.4	34.1	5%
Dickson	532	578	8%	27.4	29.9	8.3%
Dickson	557	596	6.5%	27.9	30.2	7.6%
Dickson	588	623	5.6%	28.7	30.7	6.5%

Table 3: Area comparison of our method and AA

We choose different input ranges for benchmarks 1 –3 resulting in different bit-widths. Of course, the increase of bit-widths reflects the area increase. Column 4 indicates the saving ratio of our method. With the increase of the input ranges, the saving ratio decreases because the auxiliary area caused by additional bits is reduced. Our method can achieve the implementations with area smaller for around 4% - 8%. The implementation delays are compared in Column 3. Due to the smaller bit-widths, we are able to decrease delay by around 5% - 9%. Hence, the proposed method is helpful for both area and delay reduction.

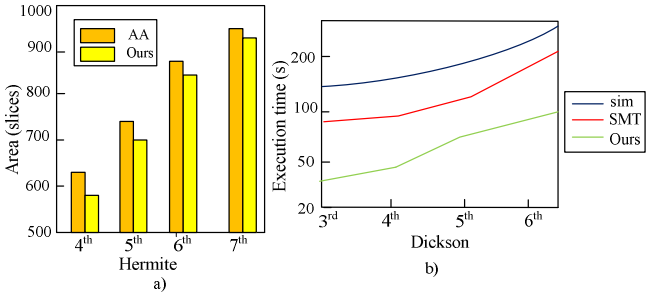

Fig. 8. Comparison of area and execution time for all methods

Fig. 8 illustrates comparison of area and execution time for Hermite and Dickson polynomials based on the same input range given in Section 4.1. Fig. 8.a) compares area of AA and our proposed method for the Hermite polynomial with orders 4 - 6.With the order increase, the area difference between the two methods decreases because the additional generated area grows more slowly. Fig. 8.b) shows calculation time in different methods for the Dickson polynomial with orders 3 – 6. The time of AA is obviously far shorter than other methods, and the two curves of SMT and simulation grow fast with order increase while the time of our method increases modestly.

Ⅴ. Conclusions and Future Work

Range analysis plays an important role in high-level synthesis of arithmetic circuits. Especially nowadays, when FPGAs are a preferred implementation technology, as well as useful computing substrate, fixed-point arithmetic has gained in prominence. Previous methods for fixed-point optimization and verification, including the improved simulation-based techniques, are of low efficiency, while the AA-based solution reaches coarse bounds. The coarse ranges may generate unnecessary bits, leading to more costly circuits. We propose a new static method to calculate ranges. It combines techniques of IA, AA and AT to find ranges efficiently, while, at the same time, the obtained ranges can be tight, hence avoiding the generation of additional bits. The key to our hybrid method is the ability to handle the correlation. Each intermediate output can be expressed using the smallest satisfying bit-width. The experiments indicate that our method needs much less time than SMT and obtains significantly tighter ranges than AA, and leads to the efficient area and delay of synthesized circuits. We plan to extend this work to various debug scenarios [12] and DSP applications on multiprocessors on chip [13].

REFERENCES

[1] Z. Zilic and Z. G. Vranesic, "Multiple Valued Logic in FPGAs", *Proc. Midwest Symposium on Circuits and Systems*, pp. 1553-1556, 1993.

[2] S. Brown, N. Manjikian, Z. Vranesic, S. Caranci, A. Grbic, R. Grindley, M. Gusat, K. Loveless, Z. Zilic and S. Srbljic, "Experience in Designing a Large-Scale Multiprocessor using Field-Programmable Devices and Advanced CAD Tools", *Proc. of ACM/IEEE Design Automation Conference* DAC `96, pp. 24-29, 1996.

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

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

[5] 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.

[6] 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.

[7] Y. Pang and K. Radecka, **“**Optimizing imprecise fixed-point arithmetic circuits specified by Taylor Series through Arithmetic Transform”, *ACM/IEEE Design Automation Conference,* pp. 397 – 402, Jun. 2008

[8] A. Kinsman and N. Nicolici, “Finite Precision bit-width allocation using SAT-Modulo Theory”**,** *Proc. DATE*, pp.1106 – 1111, 2009.

[9] Y. Pang, K. Radecka and Z. Zilic, “Optimization of Imprecise Circuits Represented by Taylor Series and Real-Valued Polynomials”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*,Vol. 29,No 8, 2010, pp.1177 - 1190

[10] Z. Zilic and Z. Vranesic, "A Multiple-Valued Reed-Muller Transform for Incompletely Specified Functions", *IEEE Transactions on Computers*, vol. 44, No. 8, pp. 1012-1020, August 1995.

[11] Y. Pang, K. Radecka and Z. Zilic, "Arithmetic Transforms of Imprecise Datapaths by Taylor Series Conversion", *Proc. Intl. Conference on Electronics, Circuits and Systems*, ICECS06, pp. 696-699, Dec. 2006.

[12] M. Boule, J-S. Chenard and Z. Zilic, “Adding Debug Enhancements to Assertion Checkers for Hardware Emulation and Silicon Debug”, *Proceedings of IEEE International Conference on Computer Design*, ICCD, pp. 294-299, 2006.

[13] S. Bourduas and Z. Zilic, “A Hybrid Mesh/Ring Interconnect for Network-on-Chip Using Hierarchical Rings for Global Routing”, *Proceedings of International Symposium on Networks-on-chip*, NOCS 2007, pp. 195-204, May 2007.