Minimization of Fractional Wordlength on Fixed-Point Conversion for High-Level Synthesis

Nobuhiro Doi[†] Takashi Horiyama[‡] Masaki Nakanishi[*] Shinji Kimura[†]

Graduate School of Information, Production and Systems[†]
Waseda University
Kitakyushu, 808-0135
nobuhiro_doi@fuji.waseda.jp,shinji_kimura@waseda.jp

Graduate School of Informatics[‡]
Kyoto University
Kyoto, 606-8501
horiyama@i.kyoto-u.ac.jp

Graduate School of Information Science[*]
Nara Institute of Science and Technology
Nara, 630-0101
m-naka@is.aist-nara.ac.jp

Abstract— In the hardware synthesis from high-level language such as C, bit length of variables is one of the key issues on the area and speed optimization. Usually, designers are required to specify the word length of each variable manually, and verify the correctness by the simulation on huge data. In this paper, we propose an optimization method of fractional wold length of floating-point variables in the floating to fixed-point conversion of variables. The amount of round-off errors are formulated with parameters and propagated via data flow graphs. The non-linear programming is used to solve the fractional wordlength minimization problem. The method does not require the simulation on huge data, and is very fast compared to ones based on the simulation. We have shown the effect on several programs.

I. Introduction

High level hardware synthesis [1] is a key technology for designing complex systems with short time-to-market, and there are many researches on the hardware generation from programming languages such as C.

In hardware generation, the bit length of variables in a program is closely related to the area and speed of the hardware. For example, a flag variable declared as an integer should be converted to 1-bit register for the area reduction.

Extended C languages are usually used to specify the exact bit length of each variable [2, 3, 4]. In other words, designers have the responsibility to optimize bit length of variables. That is a tedious and time consuming task in design process.

There are several researches on the conversion from floating-point operations to fixed-point operations [5, 6], related to digital signal processors and digital filters. In the FRIDGE project [5], a framework for the conversion is proposed: that is the simulation environment to check the correctness for the specified bit-length of variables.

A translator from C programs with floating-point operations to DSP programs with only integer operations have been proposed by Kim et al. [6]. In the translation, several properties of DSP such as the fixed wordlength and the fixed operation set are used for the optimization. These properties are too restrictive for ASIC design.

Kim and Sung also proposed a numerical analysis method for IDCT architectures [7], where the bit-lengths of coefficients and adder units are calculated using variance matrix. For the digital filter design, the method in [8] uses a similar method to ours. The method focuses on the propagation and analysis of the error under Z transfer function of IIR and FIR. These methods are for digital filters, and a more general framework is required.

In this paper, we propose an optimization method of bit-length of variables of C programs including floating-point variables. For integer variables, we use a method in [9], and we introduce a method to manipulate floating-point variables. The method is based on the CDFG analysis and non-linear programming. The necessary bit-length of each variable is decided automatically.

This paper is organized as follows: In the next section, we describe about typical fixed-point arithmetic. In Section 3, we provide the outline of our approach. Section 4 describes the optimization of fractional wordlength for fixed-point variables and operations. In Section 5, we show experimental results, and Section 6 describes the conclusion.

II. Fixed-point Arithmetic

A high speed hardware module is needed for the image and sound processing. Algorithms for such processing include many floating-point operations. To implement a processing algorithm as a hardware module, designers often convert floating-point operations into fixed-point ones, since the units for floating-point operations are slow and large compared to those for fixed-point operations.

The typical fixed-point format is specified by a 4-tuple $\langle \boldsymbol{wl}, \boldsymbol{iwl}, \boldsymbol{sign}, \boldsymbol{q\_mode} \rangle$

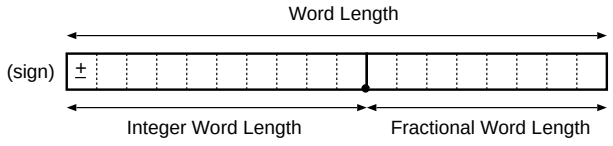

Fig. 1. Typical fixed-point format.

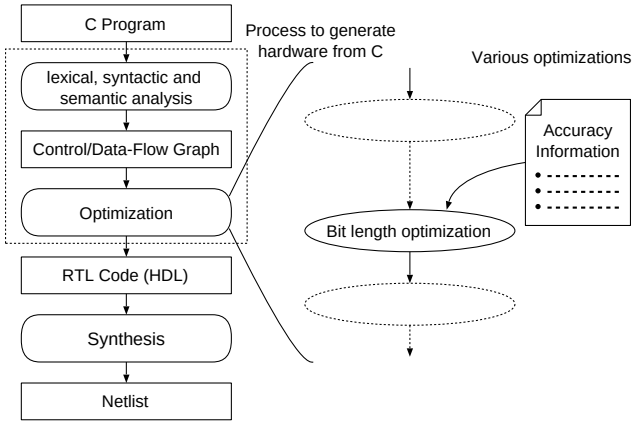

Fig. 2. Hardware generation from a C program.

wl	wordlength, the number of bits,
iwl	integer wordlength,
sign	the flag indicating signed or unsigned,
q_mode	quantization mode.

sign is the flag to indicate that the fixed-point format is signed or unsigned. If sign is set, the fixed-point format is signed and 2's complement representation is used for negative values.

q_mode defines the behavior of the fixed-point format when assigning a floating-point number with longer fractional wordlength. Several quantization modes such as *rounding* and *truncation* exist and *truncation* is used as default in hardware design.

In the following, we focus on the wordlength of the integer part and that of the fractional part. The wordlength of the fractional part is refered as $fwl\,(= wl - iwl)$. Figure 1 shows a typical fixed-point format.

III. Hardware Generation From a C Program

We have been developing a high-level synthesis tool from C programs. The original C program were written as a preformatted type (float/double), and the wordlength of variables is assumed to be sufficient for accurate computation. As shown in Figure 2, the generator performs lexical, syntactic and semantic analyses for an input C program, and constructs a Control/Data Flow Graph (CDFG) as a result of the analyses. The CDFG consists of basic blocks which contains a partially ordered set of operations and control flows to other basic blocks.

The CDFG is used in the optimization stage. In this stage, various optimizations such as *propagation of constants* and *sharing large operation units* are done. "*bit length optimization*" is also included in the stage.

The inputs and outputs of our optimization algorithm are as follows:

Input:

- Control/Data Flow Graph.
- Accuracy information.
  For input variables, the range information should be specified. For output variables, the range information and the accuracy information for the fractional part should be specified. For other variables and operations, no information is required, but designers can specify additional information for such variables.

Output:

- Optimized wordlength of each variable and each operation.

After the optimization step, an RTL code is generated.

IV. Optimization of Fractional Wordlength

This section shows optimization of fractional wordlength. Our algorithm uses the error propagation method and does not require simulation on huge data. The fractional wordlength of each variable is formulated based on the accuracy of outputs and is minimized with non-linear programming.

We explain our error model first. With this error model, the minimization of fractional wordlength can be formulated as a non-linear problem and a non-linear solver can be applied. Next, we describe each part of our method in series: "Value range analysis", "Error propagation" and "Minimization of fwl".

A. Error model

A.1 Round-off error

When representing real number with fixed-point format, we should adopt some error called *round-off error*, since the fractional part of the fixed-point format is finite.

Let X be a fixed-point variable and L_X be its fractional wordlength. The magnitude of the round-off error $E_R(X)$ included in X is denoted as,

$$E_R(X) \leq 2^{-L_X}.$$

Note that the fractional wordlength of some variables such as primary inputs would be specified by a designer.

A.2 Propagation error

An expression in a basic block is described as '$X_{\rm dst} \leftarrow X_{\rm src1} \circ X_{\rm src2}$', where $X_{\rm src1}$ and $X_{\rm src2}$ are source operands, $\circ$ is an arithmetic operation and $X_{\rm dst}$ is the result of the operation. When $X_{\rm src1}$ and/or $X_{\rm src2}$ include some error, the error is propagated to $X_{\rm dst}$. That is called *propagation error*.

Let $E_P(X)$ be the propagation error from source operands and ΔX be the amount of the error of X. For an addition ($X_{\rm dst} \longleftarrow X_{\rm src1} + X_{\rm src2}$), $E_P(X_{\rm dst})$ can be estimated as follows:

$$E_P(X_{\rm dst}) = \Delta X_{\rm src1} + \Delta X_{\rm src2},$$

since the errors are additive in the worst case. The propagation error can vary depending on the type of the operation.

At the assignment, there might exist the round-off error, so the error of $X_{\rm dst}$ can be described as follows:

$$\begin{aligned} \Delta X_{\rm dst} &= E_R(X_{\rm dst}) + E_P(X_{\rm dst}) \\ &= E_R(X_{\rm dst}) \\ &+ \mathit{Propagate}(X_{\rm src1}, \Delta X_{\rm src1}, X_{\rm src2}, \Delta X_{\rm src2}, \circ), \end{aligned}$$

where *Propagate*() is the estimated propagation error from $\Delta X_{\rm src1}$ and $\Delta X_{\rm src2}$ with the operation $\circ$.

B. Value range analysis

Many techniques for program analysis or tracing of a variable range have been developed, where the interval calculation approach is usually used [9, 10]. We adopted this approach and introduced a mechanism to analyze the round-off error and the propagation error. In the interval analysis, each variable has the range of possible value represented by the upper and lower bounds, and a program is executed symbolically based on the range. We usually use X.max as the upper bound and X.min as the lower bound of X.

B.1 Arithmetic operations

The result of addition, subtraction, and multiplication changes monotonically according to source operands. Thus, the upper and lower bounds of $X_{\rm dst}$ can be calculated from the upper and lower bounds of $X_{\rm src1}$ and $X_{\rm src2}$ as follows:

$$X_{\rm dst}.\max = \mathrm{Max}\left\{ \begin{array}{lcl} X_{\rm src1}.\max & \circ & X_{\rm src2}.\max, \\ X_{\rm src1}.\max & \circ & X_{\rm src2}.\min, \\ X_{\rm src1}.\min & \circ & X_{\rm src2}.\max, \\ X_{\rm src1}.\min & \circ & X_{\rm src2}.\min \end{array} \right\},$$

$$X_{\rm dst}.\min = \mathrm{Min}\left\{ \begin{array}{lcl} X_{\rm src1}.\max & \circ & X_{\rm src2}.\max, \\ X_{\rm src1}.\max & \circ & X_{\rm src2}.\min, \\ X_{\rm src1}.\min & \circ & X_{\rm src2}.\max, \\ X_{\rm src1}.\min & \circ & X_{\rm src2}.\min \end{array} \right\}.$$

The upper and lower bounds of the result of the division can be calculated in the same way, if the range of the divisor $X_{\rm src2}$ does not include 0. If the range of the divisor $X_{\rm src2}$ includes 0, we redefine the maximum and minimum values as follows:

$$X_{\rm src2}.\max = 2^{-L_{X_{\rm src2}}}, \quad X_{\rm src2}.\min = -2^{-L_{X_{\rm src2}}}.$$

Note that $X_{\rm dst}$ takes the largest value when $X_{\rm src1}$ takes the maximum value and $X_{\rm src2}$ takes the minimum value. If the fractional wordlength of divisor is not known, it is fixed to the longest wordlength which is allowed in the specification.

By the redefinition, we can decide the range of $X_{\rm dst}$ for all possible combinations of source variables and operations including division.

B.2 Conditional structure

When a conditional structure exists, we cannot decide which branch is to be taken at compilation time. So, both the 'then' part and 'else' part are analyzed and the worst case is adopted.

Let Btrue(X) and Bfalse(X) be the results of the analysis of the 'then' part and the 'else' part, respectively. The final result $X_{\rm result}$ after the conditional structure is calculated as follows:

$$\begin{aligned} X_{\rm result}.\max &= \mathrm{Max}\{\mathrm{Btrue}(X).\max, \mathrm{Bfalse}(X).\max\}, \\ X_{\rm result}.\min &= \mathrm{Min}\{\mathrm{Btrue}(X).\min, \mathrm{Bfalse}(X).\min\}. \end{aligned}$$

B.3 Loop structure

Loop structures can be categorized into the following three types:

1. the number of loop iterations is known,
2. the number of loop iterations is not known, but the maximum number of iterations is known,
3. there is no information about the number of iterations.

When a loop structure belongs to type 1 or type 2, that can be unfolded. We can apply our analysis method to unfolded loop structures. It is impossible to apply our method to a type 3 loop structure directly (an extension of the fixed point calculation [9] should be devised in the future).

Let the upper and lower bounds of X after t-th iteration be $X.\max(t)$ and $X.\min(t)$, respectively. The upper and lower bounds of the variable $X_{\rm result}$ after loop iterations can be calculated as follows:

$$\begin{aligned} X_{\rm result}.\max &= \mathop{\rm Max}_{1 \le s \le t}\{X.\max(s)\}, \\ X_{\rm result}.\min &= \mathop{\rm Min}_{1 \le s \le t}\{X.\min(s)\}. \end{aligned}$$

C. Error propagation

Here, we show the propagation and estimation steps for the error of output variables.

C.1 Propagation of error

The error of source variable is propagated to a destination variable as mentioned before. For example, for $X_{\rm dst} = X_{\rm src1} + X_{\rm src2}$, $E_P(X_{\rm dst})$ is $\Delta X_{\rm src1} + \Delta X_{\rm src2}$.

The amount of errors depends on the operation and the result of the value range analysis. The computation of

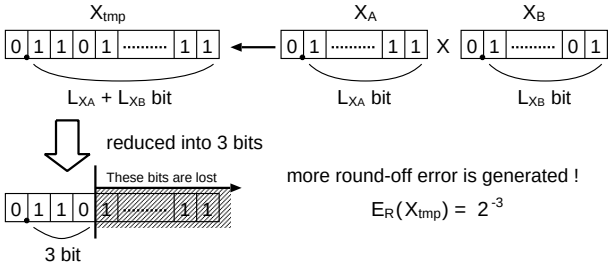

Fig. 3. The error generated when the fractional wordlength is reduced.

the propagation of errors is summarized in Table I, where $X.|\max|$ and $X.|\min|$ are defined as follows:

$$\begin{aligned} X.|\max| &= \mathrm{Max}\{\mathrm{abs}(X.\max), \mathrm{abs}(X.\min)\}, \\ X.|\min| &= \mathrm{Min}\{\mathrm{abs}(X.\max), \mathrm{abs}(X.\min)\}. \end{aligned}$$

Note that $X.|\max| \geq 0$ and $X.|\min| \geq 0$.

As for division, the function *Propagate*() can be drawn as follows:

$$\begin{aligned} X_{\mathrm{dst}} + \Delta X_{\mathrm{dst}} &= \frac{X_{\mathrm{src1}} + \Delta X_{\mathrm{src1}}}{X_{\mathrm{src2}} + \Delta X_{\mathrm{src2}}} \\ \Delta X_{\mathrm{dst}} &= \frac{X_{\mathrm{src1}}.|\max| + \Delta X_{\mathrm{src1}}}{X_{\mathrm{src2}}.|\min| + \Delta X_{\mathrm{src2}}} - \frac{X_{\mathrm{src1}}.|\max|}{X_{\mathrm{src2}}.|\min|} \\ &\leq \frac{X_{\mathrm{src1}}.|\max| + \Delta X_{\mathrm{src1}}}{X_{\mathrm{src2}}.|\min|} - \frac{X_{\mathrm{src1}}.|\max|}{X_{\mathrm{src2}}.|\min|} \\ &\leq \frac{\Delta X_{\mathrm{src1}}}{X_{\mathrm{src2}}.|\min|} . \end{aligned}$$

Note that ΔX_{dst} takes the largest value when X_{src1} takes the maximum value and X_{src2} takes the minimum value. To simplify the function *Propagate*(), ΔX_{src2} in the denominator is ignored. Since ΔX_{src2} is far smaller than X_{src2}, the function *Propagate*() still represents the error in the worst case.

C.2 Rounding of temporal variables

By the repetition of operations (including multiplication), the fractional wordlength of temporal variables become longer. Rounding is an effective method to reduce the size of registers and data paths, while the amount of errors increases. The rounding is acceptable as long as the final accuracy is sufficient with respect to the specified accuracy.

Figure 3 shows a multiplication example ($X_{\mathrm{tmp}} \leftarrow X_A * X_B$). Let the fwl of X_A and X_B be L_{X_A} and L_{L_B} respectively. The fwl of the operation result X_{tmp} is $L_A + L_B$. To reduce the size of the register, the fractional part of X_{tmp} is reduced to 3 bits in this example, and the round-off error ($= 2^{-3}$) is introduced.

For an addition or a subtraction, the fwl of the result does not increase like multiplication. It is equal to the largest fractional wordlength of source variables plus 1. The effect of the rounding would be small for such case.

D. Minimization of fwl

After the error propagation step, the estimated error of the primary output is modeled as a polynomial of the fractional wordlength L_X, where X is a variable whose fractional wordlength is not known. Non-linear solver is applied to the polynomial for minimizing the fractional wordlength of variables.

Let $L_i (i = 0, 1, \cdots, n-1)$ be the fractional wordlength of variables whose fwl is not known, $O_j (j = 0, 1, \cdots, m-1)$ be primary outputs and $\mathrm{Lim}(O_j)$ be an accuracy limitation, which is the border value of the error ΔO_j. The estimated error can be described as follows:

$$\begin{aligned} \Delta O_0 &= F_0(L_0, L_1, \cdots, L_{n-1}), \\ \Delta O_1 &= F_1(L_0, L_1, \cdots, L_{n-1}), \\ &\cdots \end{aligned}$$

where F_j is a polynomial of 2^{-L_i} and represents the estimated output error based on the error propagation. It is specified that ΔO_j should not be exceed its accuracy limitation $\mathrm{Lim}(O_j)$. Then, the following conditional expressions are drawn:

$$\begin{aligned} \mathrm{Lim}(O_0) &> \Delta O_0 = F_0(L_0, L_2, \cdots, L_{n-1}), \\ \mathrm{Lim}(O_1) &> \Delta O_1 = F_1(L_0, L_2, \cdots, L_{n-1}), \\ &\cdots \end{aligned}$$

They are considered as conditional functions for the non-linear problem.

The number of total bits of fractional parts is the measure of the hardware area. So, the goal of the minimization of the fractional wordlength is formulated as follows:

$$\mathrm{Minimize} \sum_{i=0}^{n-1} L_i$$

This is considered as an objective function for the non-linear problem.

One of non-linear solvers 'SQP method' [11] is applied to solve the problem. As the result of the non-linear problem, we can get $L_i (i = 0, 1, \cdots, n-1)$ in real value. Since the fractional wordlength L_i should be an integer, these values are revaluated.

E. Outline of optimization algorithm

Figure 4 shows an example to explain the behavior of our optimization algorithm. It is a part of the color space conversion from RGB to YCrCb. This program contains three real constants, receives three 8-bit integers, and returns Y.

The accuracy information for the program is also shown. Each row includes 'name of variable', 'input or output', 'variable type', 'upper bound', 'lower bound' and information about accuracy. The last one indicates 'round-off error' for input, 'accuracy limitation' for output. One can see that three integer variables (red, green, blue) take values between 0 to 255 and these variables do

TABLE I
PROPAGATION ERROR FROM SOURCES AND A OPERATION TO A DESTINATION.

Operation	Magnitude of the propagation error
$X_{\mathrm{src1}} \pm X_{\mathrm{src2}}$	$E_P(X_{\mathrm{dst}}) = \Delta X_{\mathrm{src1}} \pm \Delta X_{\mathrm{src2}}$
$X_{\mathrm{src1}} \times X_{\mathrm{src2}}$	$E_P(X_{\mathrm{dst}}) = X_{\mathrm{src1}}.\lvert\max\rvert \cdot \Delta X_{\mathrm{src2}} + X_{\mathrm{src2}}.\lvert\max\rvert \cdot \Delta X_{\mathrm{src1}} + \Delta X_{\mathrm{src1}} \cdot \Delta X_{\mathrm{src2}}$
$X_{\mathrm{src1}} \div X_{\mathrm{src2}}$	$E_P(X_{\mathrm{dst}}) = \dfrac{\Delta X_{\mathrm{src1}}}{X_{\mathrm{src2}}.\lvert\min\rvert}$

Source Program

```
funcY(){
   int y, red,green,blue;
   float tmp0,tmp1,tmp2,tmp3;

   tmp0 = 0.29900 * red;
   tmp1 = 0.58700 * green;
   tmp2 = 0.11400 * blue;
   tmp3 = tmp0 + tmp1;
   Y = tmp2 + tmp3;
}
```

Accuracy Information

red	IN	int	255	0	0
green	IN	int	255	0	0
blue	IN	int	255	0	0
y	OUT	double	255	0	0.5

Fig. 4. Color space conversion from RGB to YCrCb

not have errors. The accuracy limitation of Y is specified as 0.5.

For hardware generation from this program, fractional wordlength of three constants (0.299, 0.587, 0.114), four temporal variables (tmp0, tmp1, tmp2, tmp3) and one output (Y) should be fixed. These eight variables are described as $X_i (i = 0, \cdots, 7)$ and their fractional wordlengths are described as $L_{X_i} (i = 0, \cdots, 7)$.

After the value range analysis and error propagation, we obtain the following conditional function and the objective function.

- Conditional function

$$0.5 > 255 \cdot 2^{-L_{X_0}} + 255 \cdot 2^{-L_{X_1}} + 255 \cdot 2^{-L_{X_2}} + 2^{-L_{X_3}} + 2^{-L_{X_4}} + 2^{-L_{X_5}}$$

- Objective function

$$\text{Minimize} \sum_{i=0}^{5} L_{X_i}$$

Note that rounding is not applied at the assignment to tmp3(X_6) and Y(X_7), since these are the results of addition. Fwl of tmp3(L_{X_6}) fully depends on tmp0 and tmp1, and fwl of Y(L_{X_7}) fully depends on tmp2 and tmp3. In other words, E_R(tmp3) and E_R(Y) are 0. Then, the output error is not affected by fwl of tmp3 and Y.

Table II is the optimization result of color space conversion. The table shows the number of bits of the fractional part of each variable/constant. The column of total bits shows the summation of the number of bits.

We compared the results of our algorithm with those of other two approaches based on the simulation. 'Binary based search' is a heuristic algorithm to decide the

TABLE II
RESULTS OF COLOR SPACE CONVERSION.

Var name	Manual optimization	Binary Based search	Our algorithm
$0.299(L_{X_0})$	11 bit	9 bit	11 bit
$0.587(L_{X_1})$	11 bit	9 bit	11 bit
$0.114(L_{X_2})$	10 bit	9 bit	11 bit
tmp0(L_{X_3})	3 bit	8 bit	3 bit
tmp1(L_{X_4})	3 bit	8 bit	3 bit
tmp2(L_{X_5})	3 bit	8 bit	3 bit
tmp3(L_{X_6})	3 bit	8 bit	3 bit
$Y(L_{X_7})$	3 bit	8 bit	3 bit
Total Bits	47 bit	67 bit	48 bit

wordlength of all variables using binary search. For example, if the initial wordlength is 32 and the optimum one is 12, then we can obtain the optimum value by searching 32, 16(=32/2), 8(=16/2) and 12(=8+(16-8)/2). We should check acceptability of each wordlength using the simulation. After the search, we performed the adjustment by reducing the length of each variable one by one in the predefined order.

'Manual optimization' is a method to optimize the wordlength by hand. The designer analyzes the program, and uses the knowledge to reduce the wordlength. For each setting, we should check the correctness using the simulation. The designer can use the binary search at the first stage, and try various combinations at the second stage. The result depends on the skill of the designer and the size of the program. For large programs, it is hard to apply the method.

The total bits of 'Our algorithm' and those of 'Manual optimization' are almost same for the program.

V. EXPERIMENTAL RESULTS

The optimization algorithm is implemented in C language (5000 lines) with SUIF library [12], and applied to 4 sample programs: Color space conversion, Sharping filter, FIR filter and 8×8 DCT. We compared the result of our algorithm with those of other two approaches 'Manual optimization' and 'Binary based search'.

Results are shown in Table III, where 'time' is the CPU

TABLE III
RESULTS OF BIT LENGTH OPTIMIZATION.

Program Name	C lines	#Var	Manual optimization		Binary based search		Our algorithm	
			Total Bits	Time	Total Bits	Time	Total Bits	Time
RGB2YCrCb	9	8	47 bit	30 min	67 bit	523.8 sec	48 bit	0.01 sec
Sharping Filter	25	12	37 bit*	30 min*	43 bit*	340.0 sec*	47 bit	0.01 sec
FIR filter	42	32	166 bit*	45 min*	137 bit*	105.2 sec*	225 bit	0.37 sec
8×8 DCT	68	29	—	—	—	—	536 bit	82.18 sec

* The correctness is not garanteed.
– The verification of the accuracy did not complete.

time for optimization of wordlength, '#var' is the number of variables and constants declared in the source program and '*' in the table denotes that the correctness is not guaranteed for the results because of the limited simulation patterns. Since the number of all combinations of input values for the programs blows up, the number of input patterns are limited to 1,000,000. There are some non-value '–' columns in the table. It shows that the validation of the accuracy could not be completed in a day.

From the results, we can conclude that our method successfully optimizes fractional wordlength. Total bit length by 'Manual optimization' and 'Our algorithm' are almost the same. Furthermore, our algorithm guarantees the correctness, is faster than the other two approaches, and is applicable to larger programs.

The results of our approaches are worse for several examples, but other approaches do not guarantee the correctness for the worst case input. These methods just test the equality of the programs with float type and that with fixed type on only 1,000,000 random patterns.

VI. Conclusion

We have proposed an optimization algorithm to estimate the fractional wordlength of variables in the high-level synthesis. Minimization of the fractional wordlength is transformed into a non-linear problem, and solved using a SQP method. Our algorithm does not require the simulation on huge data, and is fast and guarantee the worst case accuracy.

We have implemented the optimization algorithm and that is applied to 4 sample programs. From the results, we found that the method is useful for the estimation of the fractional wordlength and is applicable to large programs.

This method can be used for prototype development of a hardware module or an embedded system. Designers can estimate the bit length of data paths or the amount of registers easily.

Refinements of the algorithm including the manipulation of sin, exp will be needed as one of our future works. Sharing of execution units should also be considered.

Acknowledgments

The authors thank to all members of Watanabe Laboratory at Nara Institute of Science and Technology for their discussions and comments. The authors also thank to anonymous reviewers for their sincere comments. This work is supported in parts by funds from the Japanese Ministry of ECSST via Kitakyushu and Fukuoka knowledge-based cluster projects. This works was also supported by Grant in Aid for Scientific Research from JSPS, and by funds from NEC.

References

[1] D. D. Gajski, N. D. Dutt, A. C.-H. Wu, and S. Y.-L. Lin, *High Level Synthesis*, Kluwer Academic Publishers, 1992.

[2] The Opend SystemC Initiative, http://www.systemc.org.

[3] A. Yamada, R. Sakurai, M. Yamaguchi, T. Kambe, and H. Katata, "Hardware synthesis with the Bach system," *Proc. IEEE ISCAS'99*, pp.366–369, May 1999.

[4] K. Wakabayashi, "C-based synthesis experiences with a behavior synthesizer "Cyber"," *Proc. DATA'99*, pp.390–393, Jan. 1999.

[5] M. Willems, V. Bursgens, H. Keding, T. Grotker, and H. Meyer, "System level fixed-point design based on an interpolative approach," *Proc. DAC'97*, pp.293–298, Nov. 1997.

[6] S. Kim, K.-I. Kum, and W. Sung, "Fixed-point optimization utility for C and C++ based digital signal processing programs," *IEEE Trans. Circuits and Syst. II*, vol. 45, no. 11, pp.1455–1464, Nov. 1998.

[7] S. Kim and W. Sung, "Fixed-Point error analysis and word length optimization of 8x8 IDCT architecture," *IEEE Trans. Circuits and Syst. II* , vol. 8, no. 8, pp.935–940, Dec. 1998.

[8] D. Menard and O. Sentieys, "Automatic evaluation of the accuracy of fixed-point algorithms," *Proc. DATE'02*, pp.529–535, 2002.

[9] O. Ogawa, K. Takagi, Y. Itoh, S. Kimura, and K. Watanabe, "Hardware synthesis from C programs with estimation of bit length of variables," *IEICE Trans.*, vol. E82-A, no. 11, pp.2338–2346, Nov. 1999.

[10] M. Stephenson, J. Babb, and S. Amarasinghe, "Bitwidth analysis with application to silicon compilation," *Proc. SIGPLAN'00*, pp.108–120, 2000.

[11] T. Ibaraki and M. Fukushima. *FORTRAN 77 Optimization Programming* (in Japanese), Iwanami, 1991.

[12] The Stanford SUIF Compiler Group, http://suif.stanford.edu.