MiniBit: Bit-Width Optimization via Affine Arithmetic

Dong-U Lee, Altaf Abdul Gaffar, Oskar Mencer and Wayne Luk
Department of Computing, Imperial College, London, United Kingdom
{dong.lee, altaf.gaffar, o.mencer, w.luk}@imperial.ac.uk

ABSTRACT

MiniBit, our automated approach for optimizing bit-widths of fixed-point designs is based on static analysis via affine arithmetic. We describe methods to minimize both the integer and fraction parts of fixed-point signals with the aim of minimizing circuit area. Our range analysis technique identifies the number of integer bits required. For precision analysis, we employ a semi-analytical approach with analytical error models in conjunction with adaptive simulated annealing to find the optimum number of fraction bits. Improvements for a given design reduce area and latency by up to 20% and 12% respectively, over optimum uniform fraction bit-widths on a Xilinx Virtex-4 FPGA.

Categories and Subject Descriptors

B.5.2 [**Register-Transfer-Level Implementation**]: Design Aids—*automatic synthesis, optimization*

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

Affine Arithmetic, Bit-Width, Fixed-Point, FPGA, Simulated Annealing

1. INTRODUCTION

One of the main objectives of hardware designers is to find the optimal design in terms of area, latency, throughput and power. Bit-widths of signals is one of the parameters designers can tweak to improve these metrics.

Bit-width optimization has enjoyed much attention in the research community. The work in this area can be classified in many different ways, one such classification is static analysis versus dynamic analysis. Dynamic analysis [1, 2] relies on the use of stimuli input signals. Though this approach provides bit-widths closer to the optimal when compared to static analysis techniques, it is problematic since a large set of stimuli signals are required to analyze a design with enough confidence, leading to prohibitively long simulation times. Static analysis [3, 7] gives more conservative bit-width estimates than dynamic analysis. However, static analysis is the more attractive solution, especially for large designs, since only the characteristics of the input signals are needed. In this work, we use a static analysis technique via affine arithmetic [6] to optimize both ranges and precisions for the signals in a fixed-point design.

DAC 2005, June 13–17, 2005, Anaheim, California, USA.

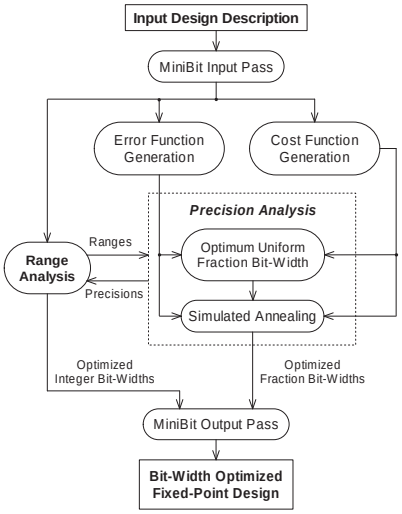

Figure 1: Overview of the MiniBit automated bit-width optimization approach.

Another way of classifying bit-width optimization is the error metric. Most existing work involves the signal to noise ratio (SNR) error criterion. The SNR criterion is popular with digital signal processing applications. On the other hand, many computer arithmetic and scientific applications require a maximum absolute error bound. This error metric is good for portability, especially when a module needs to be integrated to a larger design. Our criterion for evaluating the accuracy is the unit in the last place (ulp). The ulp of a fixed-point number with eight bits of fraction bit-width would be 2^{-8}. In this work we aim for faithful rounding, meaning that results are accurate to 1 ulp compared to infinite precision arithmetic. Hence, if the result has eight fraction bits, our approach guarantees a maximum absolute error of less than or equal to 2^{-8}.

The main contributions of this paper are:

- Analytical range and error models for fixed-point designs using affine arithmetic (Section 3).
- Range optimization using range models with guaranteed overflow/underflow protection (Section 3.1).
- Analytical uniform fraction bit-width determination using error models with guaranteed maximum absolute error bounds (Section 3.2).

- Multiple fraction bit-width determination via adaptive simulated annealing using error models and area cost functions with guaranteed maximum absolute error bounds (Section 3.2).
- Demonstration of our approach with five case studies: polynomial approximation, RGB to YCbCr conversion, matrix multiplication, B-splines and DCT realized in a Xilinx Virtex-4 FPGA (Section 4).

2. OVERVIEW

The automated framework for bit-width optimization we propose in this paper targets fixed-point numbers. In fixed-point representation a real number is represented by two parts: an integer part which represents the range, and a fraction part which represents the precision. Two's complement fixed-point is used throughout this work.

We split the bit-width optimization problem into two steps: range analysis and precision analysis. Range analysis involves inspecting the dynamic range and working out the integer bit-widths. Using insufficient bits for the range can cause overflows, and using bits excessive to the required range wastes valuable hardware resources. Hence, we are looking for the optimal integer bit-widths to avoid both shortcomings. Precision analysis involves minimizing the fraction bit-widths, while respecting the user-specified output error criterion. Again, the aim is to find the optimum fraction bit-widths that meet the output error criterion while keeping hardware cost minimal.

An overview of our bit-width optimization framework is illustrated in Figure 1. Our approach performed at the algorithmic level, and is implemented as a series of compilation passes inside MiniBit, which is built on top of the BitSize bit-width analysis system [1]. The input to MiniBit is a design description in ASC [8], C/C++ or Xilinx System Generator. MiniBit uses this design description together with user-supplied information, such as the output error specification, specified in terms of the fractional bit-width and the range of the input of values, specified in terms of the input integer bit-width, to perform the range analysis and then to generate the error function and the cost function. The error function captures the output error as a function of the fraction bit-width of the signals in the design via affine arithmetic. The area cost function returns the area cost as a function of the bit-widths of the signals.

We first perform range analysis, then pass the ranges found to the precision analysis phase. Finally, we run range analysis again since the ranges could have slightly changed with the new precisions. Range analysis is performed via affine arithmetic and computes the integer bit-width required for each signal in the design. Precision analysis operates in two phases: (1) using the error function generated by MiniBit, we find the optimum uniform fraction bit-width (UFB). The UFB serves as the initial set of parameters for the next phase. (2) we use both the error and cost functions to find the optimum multiple fraction bit-widths (MFBs) which minimize the cost function, while meeting the constraints in the error function. The MFBs are found by using adaptive simulated annealing (ASA) [5], a fast simulated annealing technique.

3. BIT-WIDTH ANALYSIS VIA AFFINE ARITHMETIC

Affine arithmetic (AA) [6] is a refinement of interval arithmetic (IA) and deals with the correlation problem of interval arithmetic. If we consider a signal x, over the range $[x_{min}, x_{max}]$, the mid-point of this range is $x_0 = (x_{max} + x_{min})/2$, the maximum variation of x in this range is $x_1 = (x_{max} - x_{min})/2$. The range $\hat{x}$ can now be expressed as $[x_0 - x_1, x_0 + x_1]$, in terms of x_0 and x_1. We express this as a single equation as

$$\hat{x} = x_0 + x_1\varepsilon_1 \tag{1}$$

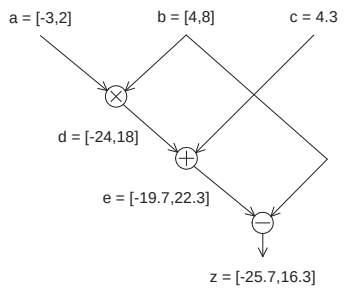

Figure 2: An example circuit performing $z = ab+c-b$.

where ϵ_1 models the uncertainties in x and lies in the range $[-1, 1]$. Expressing the range as shown in Equation (1), is known as the affine form of x. Affine arithmetic deals with the correlation problem of interval arithmetic, by encoding the range information using a separate uncertainty parameter ϵ for each signal.

We write an addition or a subtraction $\hat{x} \pm \hat{y}$ in affine arithmetic form as

$$\hat{x} \pm \hat{y} \quad = \quad (x_0 \pm y_0) + \sum_{i=1}^{n} (x_i \pm y_i)\varepsilon_i$$

For multiplication, we eliminate high order terms to obtain:

$$\hat{x} \times \hat{y} \quad = \quad x_0 y_0 + \sum_{i=1}^{n} (x_0 y_i + y_0 x_i)\varepsilon_i + uv\varepsilon_k$$

$$\text{where} \qquad u = \sum_{i=1}^{n} |x_i| \quad v = \sum_{i=1}^{n} |y_i|$$

Affine forms for other elementary operations such as division and square root are given in [6]. It has been shown that affine arithmetic gives tighter bounds than interval arithmetic for both fixed-point [3] and floating-point designs [4].

3.1 Range Analysis

The authors in [3] propose a single affine expression to capture both range and precision. However, we believe range and precision expressions should be kept separately, since the expressions can easily explode. Precision is a function of range for operations such as multiplication and division, and hence, the number of error terms ε_i can easily explode. After range analysis, we obtain numerical values for the ranges, hence the affine expressions for precisions remains manageable.

We implement range analysis using affine arithmetic to find the minimum integer bit-widths required for each signal. For instance, let us consider the evaluation of $z = ab + c - b$ as illustrated in Figure 2. Since we want to obtain the range for each signal, we set $d = ab$, $e = d + c$ and $y = e - b$. In affine form we get

$$\begin{array}{ll} \hat{a} = -0.5 + 2.5\varepsilon_1 & \hat{b} = 6 + 2\varepsilon_2 \\ \hat{c} = 4.3 & \hat{d} = -3 + 15\varepsilon_1 - 1\varepsilon_2 + 5\varepsilon_3 \\ \hat{e} = 1.3 + 15\varepsilon_1 - 1\varepsilon_2 + 5\varepsilon_3 & \hat{z} = -4.7 + 15\varepsilon_1 - 7\varepsilon_2 + 5\varepsilon_3 \end{array}$$

Hence, the ranges of the signals are $d = [-24, 18]$, $e = [-19.7, 22.3]$ and $z = [-25.7, 16.3]$. We perform range analysis on all signals for a given design and find the range for each signal. The integer bit-width (IB) required for a signal x is computed with

$$IB_x \quad = \quad \begin{cases} \lceil \log_2(d) \rceil & \text{if} \quad |d| > 1 \\ 1 & \text{if} \quad |d| \leq 1 \end{cases}$$

$$\text{where} \quad d = \max(|x_{min}|, |x_{max}|)$$

3.2 Precision Analysis

We use affine arithmetic for precision analysis in a similar fashion as for range analysis. There are two main ways to quantize a signal: truncation and round-to-nearest. Truncation and round-to-nearest can cause a maximum error of 2^{-FB} (1 ulp) and 2^{-FB-1} (0.5 ulp), respectively. Truncation chops bits off the least significant bits and requires no extra hardware resources. Round-to-nearest involves a small adder followed by truncation. For simplicity, we shall perform round-to-nearest throughout this work. Hence, the quantized version $\tilde{x}$ of a signal x is given in affine form by

$$\tilde{x} = x + 2^{-FB_{\tilde{x}}-1}\varepsilon \quad \text{where } \varepsilon = [-1, 1]$$

where $FB_{\tilde{x}}$ is the fraction bit-width of $\tilde{x}$. Hence, the error at $\tilde{x}$ due to finite precision effects is given by

$$E_{\tilde{x}} = 2^{-FB_{\tilde{x}}-1}$$

For addition/subtraction, the affine error expression is given by

$$\begin{aligned} \tilde{z} &= \tilde{x} \pm \tilde{y} = x \pm y + E_{\tilde{x}} \pm E_{\tilde{y}} + 2^{-FB_{\tilde{z}}-1}\varepsilon_3 \\ \Rightarrow E_{\tilde{z}} &= E_{\tilde{x}} + E_{\tilde{y}} + 2^{-FB_{\tilde{z}}-1}\varepsilon_3 \end{aligned}$$

For multiplication:

$$\begin{aligned} \tilde{z} &= \tilde{x}\tilde{y} \\ &= xy + xE_{\tilde{y}} + yE_{\tilde{x}} + E_{\tilde{x}}E_{\tilde{y}} + 2^{-FB_{\tilde{z}}-1}\varepsilon_3 \\ \Rightarrow E_{\tilde{z}} &= xE_{\tilde{y}} + yE_{\tilde{x}} + E_{\tilde{x}}E_{\tilde{y}} + 2^{-FB_{\tilde{z}}-1}\varepsilon_3 \end{aligned}$$

The application of the error models to the circuit in Figure 2 is shown below.

$$\begin{aligned} E_{\tilde{a}} &= 2^{-FB_{\tilde{a}}-1}\varepsilon_1 \\ E_{\tilde{b}} &= 2^{-FB_{\tilde{b}}-1}\varepsilon_2 \\ E_{\tilde{c}} &= 2^{-FB_{\tilde{c}}-1}\varepsilon_3 \\ E_{\tilde{d}} &= aE_{\tilde{b}} + bE_{\tilde{a}} + E_{\tilde{a}}E_{\tilde{b}} + 2^{-FB_{\tilde{d}}-1}\varepsilon_4 \\ E_{\tilde{e}} &= E_{\tilde{d}} + E_{\tilde{c}} + 2^{-FB_{\tilde{e}}-1}\varepsilon_6 \\ E_{\tilde{z}} &= E_{\tilde{e}} - E_{\tilde{b}} + 2^{-FB_{\tilde{z}}-1}\varepsilon_7 \end{aligned}$$

Since we are interested in the worst case error, we take the maximum absolute values for each signal.

$$\begin{aligned} \max(E_{\tilde{z}}) &= 2^{-FB_{\tilde{b}}} + 2^{-FB_{\tilde{a}}+2} + 2^{-FB_{\tilde{a}}-FB_{\tilde{b}}-2} + \\ &\quad 2^{-FB_{\tilde{d}}-1} + 2^{-FB_{\tilde{c}}-1} + 2^{-FB_{\tilde{e}}-1} + 2^{-FB_{\tilde{z}}-1} \end{aligned}$$

For faithful rounding, the output error $E_{\tilde{z}}$ needs to be less than or equal to 1 ulp, i.e.

$$\begin{aligned} 2^{-FB_{\tilde{z}}} &\geq \max(E_{\tilde{z}}) \\ \Rightarrow 2^{-FB_{\tilde{z}}-1} &\geq 2^{-FB_{\tilde{b}}} + 2^{-FB_{\tilde{a}}+2} + 2^{-FB_{\tilde{a}}-FB_{\tilde{b}}-2} \\ &\quad + 2^{-FB_{\tilde{d}}-1} + 2^{-FB_{\tilde{c}}-1} + 2^{-FB_{\tilde{e}}-1} \quad (2) \end{aligned}$$

From Equation (2), we see that the aim is to find minimal FB for each signal that satisfy the inequality and results in minimal circuit area. This is a non-linear optimization problem and analytical methods are of little use. Hence, we use simulated annealing to solve this optimization problem. We choose the adaptive simulated annealing (ASA) package available from [5]. ASA, also known as very fast simulated re-annealing, performs adaptive global optimization on multivariate non-linear stochastic systems. In ASA, the user supplies a constraint function and a cost function. Error functions such as the inequality above are supplied as the constraint function. Since our aim is to minimize circuit area in this work, we supply an area model of the circuit as a function of the signal bit-widths as the cost function. In this area model, the area for the addition $x + y$ is modeled with $\max(IB_x + FB_x, IB_y + FB_y)$ and the area for the multiplication xy is modeled with $(IB_x + FB_x)(IB_y + FB_y)$. These area models are derived to correspond with the operator area usage of our hardware compilation system (ASC) [8]. Other area models can be used to target different hardware compilers and device technologies.

The annealing process can be accelerated significantly by supplying good initial parameters (FBs in our case). Optimum uniform FBs are analytically computed and used as the initial parameters. For Equation (2), substituting the fraction bit-widths in the computation chain with a uniform fraction bit-width UFB,

$$2^{-FB_{\tilde{z}}-1} \geq 2^{-UFB} + 2^{-UFB+2} + 2^{-2UFB-2} + 3(2^{-UFB-1})$$

Let $FB_{\tilde{z}} = 16$ bits. Solving the equation for the minimum value of UFB which satisfies the inequality, gives us $UFB = 20$ bits, an analytical solution of the uniform bit-width selection problem.

4. RESULTS

We implement the five case studies with ASC, A Stream Compiler, for FPGAs [8]: degree four polynomial approximation, RGB to YCbCr color space conversion, Strassen's 2×2 matrix multiplication, cubic B-splines, and 8×8 DCT.

ASC code makes use of C++ syntax and ASC semantics, which allow the user to program on the architecture-level, the arithmetic-level and the gate-level. Designs are synthesized with ASC and placed-and-routed with Xilinx ISE 6.3 on a Xilinx Virtex-4 XC4VLX100-11 FPGA. The device contains user-programmable elements known as slices, dedicated multiply-and-add units and embedded RAMs. In order to make fair comparisons, we implement designs using slices only and combinatorially without any pipelining.

Table 1 shows the optimization times and error statistics of multiple fraction bit-width designs, and comparisons between uniform fraction bit-width (UFB) and multiple fraction bit-width (MFB) designs. The UFB and MFB designs use the same number of integer bits for all signals, as computed in our range analysis phase. The optimization times have been measured on an AMD Athlon XP 2600+ PC with 2GB DDR-SDRAM, and include both range and precision analysis times. The ulp errors and SNRs are computed by simulating MFB optimized designs using random input vectors. Double precision floating-point is assumed to be the true value for the error computations, since it is significantly more accurate than the precisions we are targeting. The maximum ulp error for all designs are well below 1 ulp to guarantee faithful rounding. Also, the average ulp error is less than 0.3 for all case studies. Looking at the area and speed comparisons, although we optimize designs for minimal area, the reduction in the multiplier and adder sizes leads to reductions in latencies as a byproduct. We note that designs using MFBs are always smaller and faster than designs using UFBs. Some of the savings may seem rather small, but we get these savings for free, as the MFB designs have the same error bound as the UFB designs.

Figure 3 shows the area variation for B-splines with increasing target precision. It can be seen that the area differences between UFB and MFB are increasing with the target precision. Figures 4 shows the area variation for various polynomial degrees with target precision fixed at eight bits. Since we use Horner's rule to evaluate polynomials, one extra degree causes one more adder and one more multiplier. In order to make a fair comparison, the coefficients are set to the number π throughout. We can see that as we increase the depth of the computation chain (i.e. increase the polynomial degree), the area difference between UFB and MFB increases.

Table 1: Optimization times and error statistics of multiple fraction bit-width (MFB results), and comparisons between uniform fraction bit-width and multiple fraction bit-width (UFB/MFB comparisons).

Case Studies		MFB Results				UFB/MFB Comparisons							
Application	Prec	Opt	Max Err	Avg Err	SNR	Area [slices]				Latency [ns]			
	[bits]	Time [s]	[ulp]	[ulp]	[dB]	UFB	MFB	Diff	Impr [%]	UFB	MFB	Diff	Impr [%]
Degree 4	8	1.9	0.694	0.253	51.5	803	723	80	9.96	114.00	105.39	8.61	7.55
Polynomial	16	2.0	0.731	0.256	99.5	1921	1797	124	6.45	168.55	151.81	16.74	9.93
RGB to YCbCr	8	8.9	0.662	0.260	97.2	1165	1132	33	2.83	37.47	36.95	0.52	1.39
	16	9.7	0.793	0.272	144.9	1641	1602	39	2.38	50.26	48.83	1.43	2.85
2×2 Matrix	8	16.1	0.520	0.251	54.4	1896	1799	97	5.12	44.20	42.73	1.47	3.33
Multiplication	16	19.5	0.528	0.247	102.5	4240	4072	168	3.96	59.22	56.14	3.08	5.20
B-Splines	8	27.7	0.716	0.267	49.8	1189	952	237	19.93	88.39	78.58	9.81	11.10
	16	32.8	0.774	0.278	96.6	2652	2165	487	18.36	130.11	114.03	16.08	12.36
8×8 DCT	8	154.3	0.702	0.254	103.1	5368	5217	151	2.81	54.83	50.73	4.10	7.48
	16	179.1	0.708	0.257	151.3	7320	7167	153	2.09	66.39	59.42	6.97	10.50

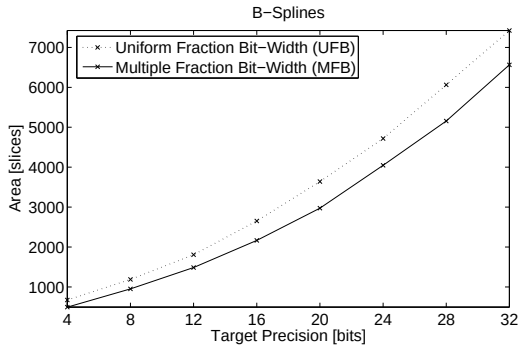

Figure 3: Area variation for B-splines with increasing target precision.

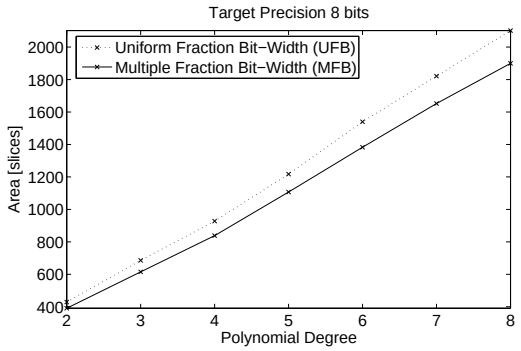

Figure 4: Area variation for various polynomial degrees with target precision fixed at eight bits.

5. CONCLUSIONS

We present MiniBit, an automated approach for optimizing bit-widths of fixed-point designs with static analysis. We describe methods to minimize both integer and fraction parts of fixed-point signals based on affine arithmetic. Our range analysis technique finds the minimum number of required integer bits. For precision analysis, we employ a semi-analytical approach, where analytical error models in conjunction with adaptive simulated annealing are used to find the optimum number of fraction bits. The analytical models allow us to guarantee overflow protection and numerical accuracy for all inputs over the user-specified input intervals.

One limitation of our approach is that the search space for ASA could can be vast for large designs, leading to slow optimization times. For future work, we hope to use clustering techniques by optimizing parts of a large design independently, which will result in suboptimal bit-widths but faster execution times.

REFERENCES

[1] A. Abdul 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 Computing Machines*, pages 79–88, 2004.

[2] R. Cmar, L. Rijnders, P. Schaumont, S. Vernalde, and I. Bolsens. A methodology and design environment for DSP ASIC fixed point refinement. In *Proc. ACM/IEEE Design Automation and Test in Europe Conf.*, pages 271–276, 1999.

[3] C. Fang, R. Rutenbar, and T. Chen. Fast, accurate static analysis for fixed-point finite-precision effects in DSP designs. In *Proc. ACM/IEEE Int'l Conf. on Computer-Aided Design*, pages 275–282, 2003.

[4] C. Fang, R. Rutenbar, M. Püschel, and T. Chen. Toward efficient static analysis of finite-precision effects in DSP applications via affine arithmetic modeling. In *Proc. ACM/IEEE Design Automation Conf.*, pages 496–501, 2003.

[5] L. Ingber. *Adaptive Simulated Annealing (ASA) 25.15*, 2004. `http://www.ingber.com/#ASA`.

[6] L. de Figueiredo and J. Stolfi. Self-validated numerical methods and applications. In *Brazilian Mathematics Colloquium monograph*. IMPA, Brazil, 1997.

[7] D. Menard and O. Sentieys. Automatic evaluation of the accuracy of fixed-point algorithms. In *Proc. ACM/IEEE Design Automation and Test in Europe Conf.*, pages 1530–1591, 2002.

[8] O. Mencer, D. Pearce, L. Howes, and W. Luk. Design space exploration with A Stream Compiler. In *Proc. IEEE Int'l Conf. Field-Programmable Technology*, pages 270–277, 2003.