Fixify: A Toolset for Automated Floating-point to Fixed-point Conversion

P. Belanović and M. Rupp
Vienna University of Technology
Institute for Communications and RF Engineering
Gusshausstr. 25/389, 1040 Vienna, Austria

ABSTRACT

Automation of the floating-point to fixed-point conversion of algorithmic descriptions is expected to produce significant improvements in the design process of embedded systems, in terms of its efficiency and error resilience. Although several analytical and statistical approaches to help the designer make the conversion trade-off between design parameters and implementation costs do exist, none are able to replace the designer in making the trade-off itself. Fixify, a toolset for automated floating-point to fixed-point conversion is presented here. It uses existing optimisation heuristics to perform the conversion trade-off, disburdening the designer and significantly improving the design process.

Keywords: Floating-point to fixed-point conversion, design methodology, embedded systems.

1. INTRODUCTION

The development of modern embedded systems, especially in the domain of wireless communications, experiences high pressures in several design requirements, such as power, area and performance, but also time-to-market, cost and quality [1, 2]. A number of new system design environments aimed at meeting these challenges are currently intensively researched [3, 4]. These environments integrate all resources needed by the designer during the development of the embedded system, thus making the design process significantly more efficient and error resilient. This in turn reduces the time-to-market and the overall cost of the product, while improving its quality.

A part of the design process where some of the most significant improvements in efficiency are possible is the conversion from floating-point to fixed-point numeric formats. Fixify, a toolset for automated floating-point to fixed-point conversion of algorithmic-level system descriptions is presented here.

The rest of this paper is organised as follows. Section 1.1 explains the floating-point to fixed-point conversion process. Further on in Section 1.2 existing research efforts for automating the conversion process are described. The proposed Fixify toolset is explained in detail in Section 2. Finally, Section 3 presents the conclusions of the paper.

1.1 Floating-point to Fixed-point Conversion

Initial steps in the design of embedded systems are typically performed in high-level algorithmic modelling environments, such as Matlab, C/C++, SystemC [5], or Ptolemy [6].

Invariably, all values in these algorithmic models are represented in floating-point formats, rather than fixed-point formats. This is due to the significantly larger dynamic range and finer precision of floating-point formats, which in turn gives the designer vastly more freedom in modelling operations, without overflow and quantisation errors.

However, at the end of the implementation stage of the design process, practically all custom hardware parts of embedded systems are built using fixed-point formats only. This is due to the fact that implementations using correctly determined fixed-point formats offer the same (or acceptably and predictably lower) numeric performance as floating-point implementations, but with vastly superior throughput performance, using far less power and consuming far less silicon area.

Therefore, during the design process, the designer must refine the algorithmic description in floating-point formats to a description using suitable fixed-point formats. This refinement step is traditionally performed manually, as no Electronic Design Automation (EDA) tools that support this task currently exist on the market.

The process of manually converting a floating-point model into an appropriate fixed-point model usually consists of two steps. Firstly, the designer determines the fixed-point formats of some of the variables in the system, relying heavily on previous experience and a priori knowledge of the final system architecture. The second part of the conversion involves a series of simulations of the system description which contains a mixture of floating-point and fixed-point formats, with iterative moving of floating-point variables into appropriate fixed-point formats. Appropriate fixed-point formats are chosen so as to minimise the hardware cost and the execution times of the final implementation, while maintaining overall system performance criteria, such as Signal to Quantisation Noise Ratio (SQNR), Bit Error Rate (BER), and Symbol Error Rate (SER), within acceptable bounds. This iterative simulation procedure ends when all floating-point variables have been converted to appropriate fixed-point formats.

While this conversion process traditionally produces results which are mostly close to optimal, it consumes a considerable amount of manual effort and thus significantly prolongs the design process. Hence, substantial improvements in design efficiency can be made by automating the floating-point to fixed-point conversion process. Fixify is a toolset presented here, aimed at making these efficiency improvements.

This work has been funded by the Christian Doppler Pilot Laboratory for Design Methodology of Signal Processing Algorithms.

1.2 Related Work

Several research efforts to automate the conversion of algorithms written in floating-point formats into optimal fixed-point descriptions exist. They can be divided on the basis of their approach to the conversion process into analytical and statistical approaches.

Analytical approaches [7–10], also known as static approaches, use various forms of code analysis to deduce optimal fixed-point formats. They have several advantages over statistical approaches.

Since analytic approaches do not involve simulations of the system description, they possess the advantage of not requiring any input test vectors for the system description. Also, since analytical approaches do not take data into consideration, their results are valid for all possible input test vectors. Furthermore, analytical approaches execute significantly faster than statistical approaches, due to the absence of time-consuming simulations of the system description.

However, analytical approaches also have some critical disadvantages. Firstly, they produce unrealistically conservative results, compared to those obtained by the designer performing manual floating-point to fixed-point conversion of the system description. In fact, analytical approaches determine the upper bound for all fixed-point bitwidths in the system description.

Also, analytical approaches are not suited to make the trade-off between design parameters such as SQNR, SER, and BER against implementation costs such as area, throughput, and power. This is due to their independence from test vectors and hence inability to realistically estimate the aforementioned design parameters.

Additionally, analytical approaches require information on ranges and precisions of inputs to the system description, in order to successfully analyse all variables in the system and determine the appropriate fixed-point formats for them.

Finally, analytical approaches encounter difficulties in obtaining fixed-point formats for system descriptions that contain complex loop operations. With a large number of loop iterations, analytical approaches significantly overestimate range requirements of variables. System descriptions containing infinite loops (e.g. IIR filters) cannot at all be processed by analytical approaches. Thus, analytical approaches place unrealistic and unnecessary restrictions on the designer.

On the other hand, statistical approaches [11–17], also known as dynamic approaches, employ iterative simulations of the system description to determine optimal fixed-point formats.

The most important advantage of statistical approaches over analytical approaches is the superior quality of their results. Employing statistical approaches, fixed-point formats which are close to or equal to optimal and very similar to those found by labourious manual conversion can be found.

Also, statistical approaches involve simulations of the system description and can thus be used to realistically estimate design parameters (SQNR, BER, SER) in various fixed-point scenarios. Hence, it is possible to extend these approaches to automate the trade-off between design parameters and implementation costs (area, power, throughput).

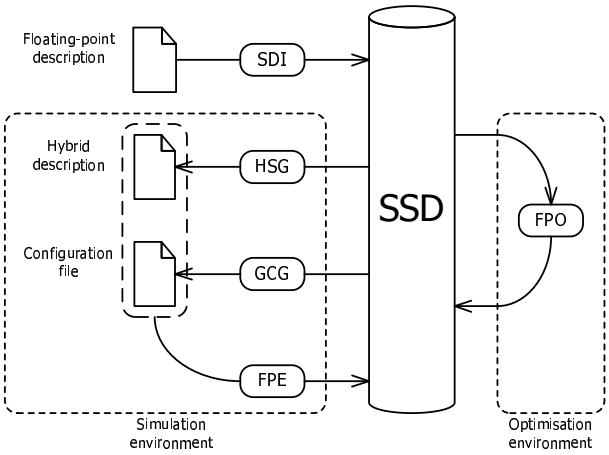

Figure 1: The Fixify toolset for floating-point to fixed-point conversion

One drawback of statistical approaches is their dependence on input test vectors. Since these approaches rely on simulations of the system description, the quality of the conversion results is dependent on the quality of the input test vectors. Hence, when employing statistical floating-point to fixed-point conversion methods, it is necessary to provide high quality input vectors, exhibiting high coverage of realistic operation scenarios.

Another weakness of statistical approaches lies in their relatively long run times. Iterative simulations these methods rely on may require relatively long periods of time, especially using the aforesaid comprehensive input vectors. Hence, successful implementations of statistical approaches for floating-point to fixed-point conversion require careful attention to be paid to minimising the number of needed simulations as well as minimising the duration of each.

The toolset presented here is based on a statistical approach to the conversion process, taking into consideration results of iterative simulations of the system description. Additionally to all previously presented approaches, Fixify also automates the trade-off between design parameters and implementation costs, completely replacing designer's manual effort in the conversion process. All parts of the Fixify toolset are discussed in detail in the following sections.

2. FIXIFY TOOLSET

The Fixify toolset for automated floating-point to fixed-point conversion is composed of the following parts:

- System Description Interface
- Simulation Environment
- Optimisation Environment

These are shown graphically in Figure 1 and explained in the following sections.

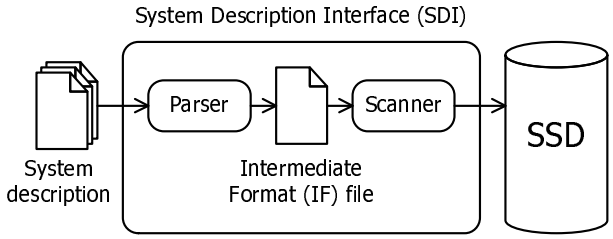

Figure 2: Processing a system description into SSD by SDI

2.1 System Description Interface

A key feature of the design methodology this work has been based on [3] is the reading in of the description of the design into the Single System Description (SSD). The tool responsible for this task is the System Description Interface (SDI) [18,19]. Figure 1 illustrates how SDI can be used to read in a system description which uses floating-point formats.

Regardless of the language in which the system description is written, the SDI tool firstly processes the description using the Parser module, extracting key design information into an Intermediate Format (IF) file. The IF file is independent of the language and structure of the system description. This file is read by the Scanner module, storing this key design information into the SSD. The Scanner module is responsible for managing the structure and the content of the SSD. This design flow is represented in Figure 2.

Therefore, the Parser module is the only part of the SDI tool which is dependent on the design language used. Hence, a dedicated Parser module needs to be developed for every language used in describing the system, while the single Scanner module can be reused in every case. Currently, Parser modules for the following system design languages have been developed:

- SystemC (up to current version 2.01)
- VHDL
- COSSAP (GenericC)

2.2 Simulation Environment

The performance of the design with fixed-point effects taken into consideration, relative to the performance of the design in full precision of floating-point formats is determined by the simulation environment. This environment consists of the following tools (see Figure 1):

- Hybrid System Generator (HSG)
- Global Configuration Generator (GCG)
- Fixed-Point Evaluator (FPE)

The HSG tool is responsible for creating the hybrid system description. This system description is made up of two parallel instances of the system - one in original floating-point formats, while the other uses the fixed-point formats determined by the optimisation environment (see Section 2.3). Both parts of the hybrid system description contain embedded extraction blocks which are used in monitoring the flow of data through the two versions of the system description.

It is important to note that the hybrid system description created by HSG refers to the fixed-point format of each variable in the system only symbolically. These fixed-point formats are then explicitly defined in the separate configuration file shown in Figure 1. In this way, the code for the hybrid system description is generated exactly once by the HSG tool, while the considerably smaller configuration file gets generated by the GCG tool in every iteration of the conversion process.

The FPE tool evaluates the relative degradation of numeric performance of the algorithm through introduction of fixed-point formats. The actual simulation of the hybrid system description occurs in this part of the toolset. Through the embedded extraction blocks mentioned above, the FPE tool is able to observe the flow of data through both the floating-point and the fixed-point versions of the system description, detecting and analysing the relative degradation of numeric performance. Results of this analysis are then placed into the SSD, for later use by the optimisation environment.

An example of the processing performed by the simulation environment is shown in Figure 3. Firstly, as explained in the previous section, the floating-point system description labelled `design` is read into the SSD by the SDI tool. At this point it is possible to generate the `hybrid description` by executing the HSG tool. As illustrated in Figure 3, the hybrid description is composed of two versions of the original algorithm, namely the `design_float`, containing the floating-point version and `design_fixed`, containing the fixed-point version of the algorithm.

During simulation, both versions of the algorithm receive at the inputs the same input test vectors. Both versions of the algorithm also contain embedded extraction modules, which are used to monitor the flow of data through both. The extraction modules are also used to inspect the outputs of both versions of the algorithm.

It is by the analysis of these extracted data and the comparison of the relative performance of the fixed-point algorithm to the full-precision floating-point version that the FPE tool enriches the SSD with information on the relative numeric performance of all the fixed-point formats previously chosen by the optimisation environment.

Although the simulation environment is dependent on the language the hybrid description is implemented in, it can in principle be based on any language which supports both floating-point and fixed-point formats natively. The simulation environment is initially developed based on SystemC and its support for fixed-point formats. However, the modular structure of the HSG and GCG tools allows an extension to be built for every additional language that should be supported. Thus, the simulation environment can in principle support a palette of design languages. Hence, the presented framework is open for inclusion of any modern or future design languages.

2.3 Optimisation Environment

Optimisation of fixed-point formats is performed in the optimisation environment of the conversion toolset. This environment contains a single tool, the Floating-Point Optimiser (FPO), as shown in Figure 1.

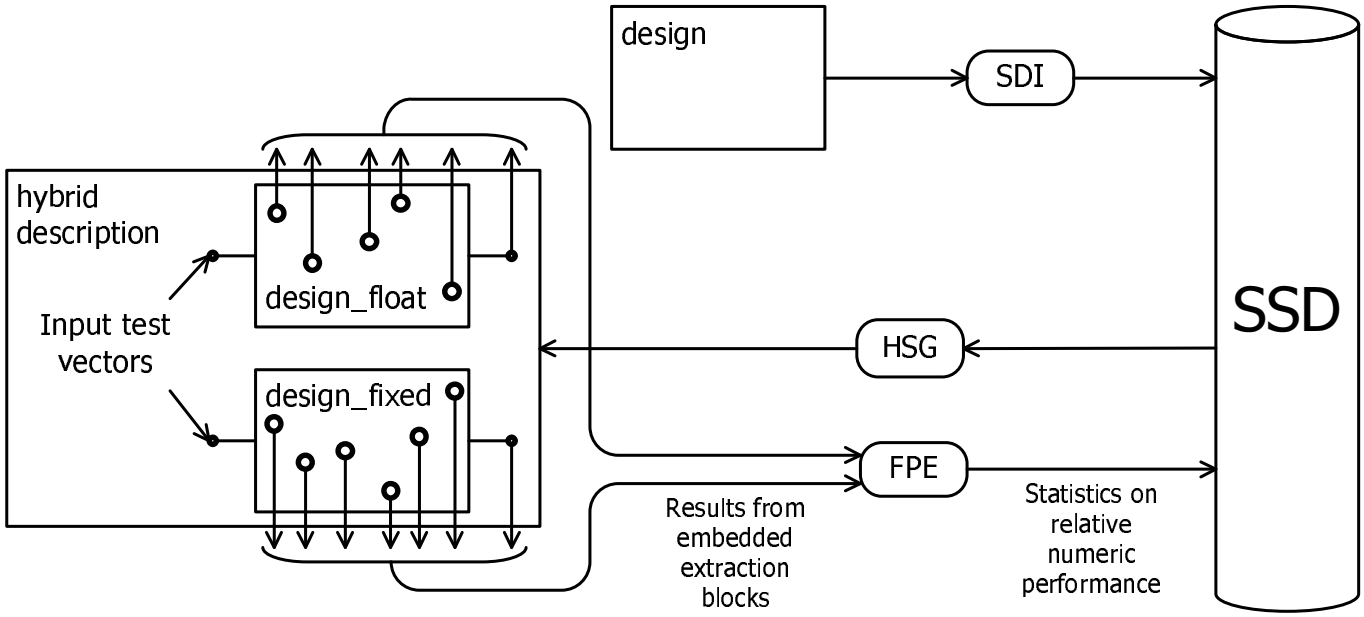

Figure 3: Generation and simulation of a hybrid system description

The FPO tool performs the trade-off between design parameters (such as SQNR, BER, or SER) and implementation costs (such as are, power, or throughput), which has so far been performed manually, thus disburdening the designer and making the conversion process more error resilient.

By analysing the results returned by the FPE tool, indicating relative performance of the current fixed-point formats, the FPO tool is able to construct realistic estimates of the design parameters such as SQNR. On the other hand, by analysing the fixed-point formats in the system, the FPO tool is also able to derive realistic metrics for implementation costs, such as area, power, or execution time for all system modules. Thus, the FPO tool is able to construct optimised cost functions in order to employ existing heuristics, such as Simulated Annealing (SA), Genetic Algorithm (GA), or Taboo Search (TS). Using these heuristics, the FPO tool minimises implementation costs, while maintaining design parameters within acceptable ranges.

For example, a signal named `sig075` found in the system description `design` in Figure 3 has been reported by the FPE tool to contain values between 25 and 100, with the smallest required precision step of 0.25. Hence, as the first step, the optimisation environment assigns the following fixed-point format to the signal:

```
<unsigned,9,7,closest,saturate>
```

This assigns the signal `sig075` to be represented as an unsigned 9-bit fixed-point number, where the 7 most significant bits lie to the left of the radix point (integer bits). The rounding mode used for this signal will be rounding to closest and on overflow, the signal value will saturate to the highest representable value. Based on the results returned by the FPE tool, this fixed-point format will be able to represent all values passing through the signal `sig075` without overflow or quantisation errors.

At this point, the simulation environment is engaged again and based on the results it returns, the optimisation environment estimates the SQNR to be 45dB and the approximate area cost for the implementation of the design to be 10500 gates. In a similar way it is established that removing the two fractional bits of `sig075` will reduce the SQNR to 37dB and the area cost to 9500 gates. However, because the minimum acceptable level of SQNR for this design is 40dB, the optimisation environment tries removing just one of the two fractional bits. This configuration results in an SQNR of 42dB, satisfying the design parameter requirement, with an estimate for the hardware cost at 9800 gates, or a saving of 700 gates. Hence, in this example, the optimisation environment succeeded in trading off 3dB of system performance for 700 gates of reduced hardware cost, which will in turn likely produce a reduction in the power consumption of the design as well.

It is important to note that the optimisation environment is independent of the language in which the system description is written or the design environment in which it is created. This is due to the fact that the optimisation environment does not interface to the hybrid system description directly, but only to the SSD itself.

3. CONCLUSIONS

Fixify, an automated, flexible and modular toolset for performing fixed-point refinement from floating-point descriptions has been presented. It introduced several advantages compared to exiting approaches, including independence of the actual heuristic being used for the optimisation, independence of the language the system is described in, optimisation for multiple design goals, and improved integration into any existing design flow.

Future work on Fixify includes completion of the simulation and optimisation environments, implementation of

several appropriate optimisation heuristics, such as Genetic Algorithm, Simulated Annealing, Taboo Search and others, and applying these heuristics to optimise the conversion process for various design goals, such as area, execution time, and power, but also for various system performance criteria such as SQNR, BER, and SER.

REFERENCES

[1] Markus Rupp, Andreas Burg, and Eric Beck. Rapid Prototyping for Wireless Designs: the Five-Ones Approach. *Signal Processing Europe 2003*, 83(7):1427–1444, July 2003.

[2] R. Subramanian. Shannon vs. Moore: Driving the Evolution of Signal Processing Platforms in Wireless Communications. In *IEEE Workshop on Signal Processing Systems SIPS'02*, October 2002.

[3] P. Belanović, M. Holzer, D. Mičušík, and M. Rupp. Design Methodology of Signal Processing Algorithms in Wireless Systems. In *International Conference on Computer, Communication and Control Technologies CCCT'03*, pages 288–291, July 2003.

[4] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty. Model-Integrated Development of Embedded Software. In *Proceedings of the IEEE*, volume 91, pages 145–164, January 2003.

[5] Open SystemC Initiative. www.systemc.org.

[6] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy: A Framework for Simulating and Prototyping Heterogenous Systems. *International Journal in Computer Simulation*, 4(2), 1994.

[7] M. Coors, H. Keding, O. Lüthje, and H. Meyr. Design and DSP Implementation of Fixed-point Systems. In *EURASIP Journal of Applied Signal Processing*, number 9, pages 908–925, September 2002.

[8] Synopsys. Converting ANSI-C into Fixed-Point Using CoCentric Fixed-Point Designer. Technical report, Synopsys, Inc., April 2000.

[9] M. Stephenson, J. Babb, and S. Amarasinghe. Bitwidth Analysis with Application to Silicon Compilation. In *SIGPLAN Conference on Program Language Design and Implementation PLDI'00*, pages 108–120, June 2000.

[10] C. F. Fang, R. A. Rutenbar, and T. Chen. Fast, Accurate Static Analysis for Fixed-point Finite Precision Effects in DSP Designs. In *International Conference on Computer Aided Design*, San Jose, November 2003.

[11] S. Kim, K. Kum, and W. Sung. Fixed-Point Optimization Utility for C and C++ Based Digital Signal Processing Programs. *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, 45(11):1455–1464, November 1998.

[12] Y. Cao and H. Yasuura. Quality-Driven Design by Bitwidth Optimization for Video Applications. In *IEEE/ACM Asia and South Pacific Design Automation Conference*, January 2003.

[13] R. Cmar, L. Rijnders, P. Schaumont, S. Vernalde, and I. Bolsens. A Methodology and Design Environment for DSP ASIC Fixed Point Refinement. In *Design, Automation and Test in Europe Conference DATE'99*, pages 271–276, Munich, March 1999.

[14] C. Shi and R. W. Brodersen. An Automated Floating-point to Fixed-point Conversion Methodology. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 529–532, April 2003.

[15] G. Caffarena, A. Fernandez, C. Carreras, and O. Nieto-Taladriz. Fixed-point Refinement of OFDM-based Adaptive Equalizers: A Heuristic Approach. In *European Signal Processing Conference EUSIPCO 2004*, Vienna, September 2004.

[16] E. Özer, A. P. Nisbet, and D. Gregg. Stochastic Bitwidth Approximation Using Extreme Value Theory for Customizable Processors. Technical report, Trinity College, Dublin, Ireland, October 2003.

[17] T. Aamodt. Floating-point to Fixed-point Compilation and Embedded Architectural Support. Master's thesis, University of Toronto, Canada, 2001.

[18] P. Belanović, M. Holzer, B. Knerr, M. Rupp, and G. Sauzon. Automatic Generation of Virtual Prototypes. In *International Workshop on Rapid System Prototyping RSP'04*, Geneva, June 2004.

[19] M. Holzer, B. Knerr, P. Belanović, and M. Rupp. Faster Complex SoC Design by Virtual Prototyping. In *International Conference on Cybernetics and Information Technologies, Systems and Applications CITSA'04*, Orlando, FL, July 2004.