DSP Code Generation with Optimized Data Word-Length Selection

Daniel Menard and Olivier Sentieys

Reconfigurable and Retargetable Digital Device Group (R2D2) IRISA
6 rue de Kerampont, F-22300 Lannion, France
menard@irisa.fr
http://www.irisa.fr/R2D2/

Abstract. Digital signal processing applications are implemented in embedded systems with fixed-point arithmetic to minimize the cost and the power consumption. To reduce the application time-to-market, methodologies for automatically determining the fixed-point specification are required. In this paper, a new methodology for optimizing the fixed-point specification in the case of software implementation is described. Especially, the technique proposed to select the data word-length under a computation accuracy constraint is detailed. Indeed, the latest DSP generation allows to manipulate a wide range of data types through sub-word parallelism and multiple-precision instructions. In comparison with the existing methodologies, the DSP architecture is completely taken into account to optimize the execution time under accuracy constraint. Moreover, the computation accuracy evaluation is based on an analytical approach which allows to minimize the optimization time of the fixed-point specification. The experimental results underline the efficiency of our approach.

1 Introduction

Most digital signal processing algorithms are specified with floating-point data types but they are finally implemented into fixed-point architectures to satisfy the cost and the power consumption constraints of embedded systems. To reduce the embedded system time-to-market, high-level development tools are required to automate some tasks. The manual transformation of floating-point data into fixed-point data is a time-consuming and error prone task. Indeed, some experiments [8] have shown that this manual conversion can represent up to 30% of the global implementation time. Thus, methodologies for the automatic transformation of floating-point data into fixed-point data have been proposed [12][26].

For Digital Signal Processors (DSP), the aim is to define the optimal fixed-point specification which maximizes the accuracy and minimizes the size and the execution time of the code. Existing methodologies [13][26] achieve a floating-point to fixed-point conversion leading to an ANSI-C code with integer data types. Nevertheless, the different elements of the target DSP are not taken into account for the process of fixed-point data coding. The analysis of the architecture influence on the computation accuracy underlines the necessity to take into

H. Schepers (Ed.): SCOPES 2004, LNCS 3199, pp. 214–228, 2004.

account the DSP architecture to obtain an optimized fixed-point specification. Particularly, the different data types supported by the latest DSP generation offer an opportunity to make a trade-off between the computation accuracy and the code execution time.

In this paper, a new global methodology for the implementation of floating-point algorithms into fixed-point processors under accuracy constraint is presented, and especially, the stage of data word-length selection is detailed. Firstly, the available methodologies for the fixed-point conversion are presented in section two. After a review of the different datapath elements in a DSP which influence the computation accuracy, the fixed-point conversion methodology is presented in section four. In section five, the data word-length determination stage is detailed. Finally, some experimental results obtained on different applications are given.

2 Related Work

In this section the different available methodologies for the automatic implementation of floating-point algorithms into fixed-point architectures are presented. These methodologies achieve the floating-point to fixed-point transformation at a high-level (source code).

The FRIDGE [11] methodology developed at the Aachen University achieves a transformation of the floating-point C source code into a C code with fixed-point data types. In the first step called *annotations*, the user defines the fixed-point format of some variables which are critical in the system or for which the fixed-point specification is already known. Moreover, global annotations can be defined to specify some rules for the entire system (maximal data word-length, casting rules). The second step called *interpolation* [25][11] corresponds to the determination of the integer and fractional part word-length for each data. The fixed-point data formats are obtained from a propagation rule set and the program control flow analysis. This step leads to an entire fixed-point specification of the application. This description is simulated to verify if the accuracy constraint is fulfilled. The commercial tool *CoCentric Fixed-point Designer* proposed by Synopsys is based on this approach.

In [26], a method called *embedded approach* is proposed for generating an ANSI-C code for a DSP compiler from the fixed-point specification obtained previously. The data (source data) for which the fixed-point formats have been obtained with the technique presented previously, are specified with the available data types (target data) supported by the target processor. The freedom degrees due to the source data position in the target data allow to minimize the scaling operations. This methodology allows to achieve a bit-true implementation in a DSP of a fixed-point specification. But the fixed-point data formats are not optimized according to the target processor and especially according to the different data types supported by this processor.

The aim of the tool presented in [13] is to transform a floating-point C source code into an ANSI-C code with integer data types to be independent of the target

architecture. Moreover, a fixed-point format optimization is done to minimize the number of scaling operations. Firstly, the floating-point data types are replaced by fixed-point data types and the scaling operations are included in the code. The scaling operations and the fixed-point data formats are determined from the dynamic range information. The reduction of scaling operation number is based on the assignation of a common format to several relevant data allowing the minimization of the scaling operation cost function. This cost function depends on the processor scaling capacities. For processor with a barrel shifter, the scaling operation cost is equal to one cycle, otherwise the number of cycles required for a shift of n bits is equal to n cycles.

In this methodology, the scaling operations are minimized. But, the code execution time is not optimized under a global accuracy constraint. The accuracy constraint is only specified through the definition of a maximal accuracy degradation allowed for each data. Moreover, the architecture model used for the scaling operation minimization does not lead to an accurate estimation of the scaling operation execution time. For processor based on a specialized architecture, the scaling operation execution time depends on the data location in the datapath. Furthermore, for processors with instruction-level parallelism capacities, the overhead due to scaling operations depends on the scheduling step and can not be easily evaluated before the code generation process.

These two methodologies achieve a floating-point to fixed-point transformation leading to an ANSI-C code with integer data types. Nevertheless, the different target DSP elements are not taken into account for the fixed-point conversion. Especially, the opportunities offered by DSPs, able to manipulate a wide variety of data types, are not exploited.

3 Digital Signal Processor Architecture

DSP architectures are designed to compute efficiently the arithmetic operations involved in digital signal processing applications. Different elements of the datapath influence the computation accuracy as described in [17]. The most important element is the data word-length. Each processor is defined by its native data word-length which is the word-length of the data that the processor buses and datapath can manipulate in a single instruction cycle [14]. For most of the fixed-point DSPs, the native data word-length is equal to 16 bits. For ASIP (Application Specific Instruction-set Processor) or some DSP cores like the CEVA-X and the CEVA-Palm [20](CEVA), this native data word-length is customizable to fit better the architecture to the target application. The computation accuracy is linked to the word-length of the data which are manipulated by the operations and depends on the kind of instructions which are used for implementing the operation.

DSPs allow through classical instructions, the achievement of a multiply accumulate (MAC) operation without lost of information, by providing a double precision result. The adder and the multiplier output word-lengths are equal to the double of the native data word-length. Nevertheless, the data dynamic

Table 1. Word-length of the data which can be manipulated by different DSP for arithmetic operations (multiplication, addition, shift).

Processor	Data Types (bits)
TMS320C64x (T.I.) [23]	8, 16, 32, 40, 64
TigerSHARC (A.D.) [2]	8, 16, 32, 64
SP5, UniPhy (3DSP) [1]	8, 16, 24, 32, 48
CEVA-X1620 (CEVA) [3]	8, 16, 32, 40
ZSP500 (LSI Logic) [24]	16, 32, 40, 64
OneDSP (Siroyan)	8, 16, 32, 44, 88

range increase due to successive accumulations can lead to an overflow. Thus, some DSPs extend the accumulator word-length by providing guard bits. These supplementary bits allow the storage of the additional bits generated during successive accumulations.

Many DSPs support multiple-precision arithmetic to increase the computation accuracy. In this case, the data are stored in memory with a more important precision. The data word-length is a multiple of the natural data word-length. Given that multiple-precision operations manipulate greater data word-length, a multiple-precision operation is achieved with several classical operations. Consequently, the execution time of this operation is greater than the one of a classical operation.

To reduce the code execution time, some recent DSPs allow the exploitation of the data-level parallelism by providing SWP (Sub-Word Parallelism) capacities. An operator (multiplier, adder, shifter) of word-length N is split to execute k operations in parallel on sub-word of word-length N/k. This technique can accelerate the code execution time up to a factor k. Thus, these processors can manipulate a wide diversity of data types as shown in table 1. In [6], this technique has been used for implementing a CDMA synchronisation loop in the TigerSharc DSP [2]. The SWP capacities allow to achieve 6,6 MAC per cycle with two MAC units.

4 Fixed-Point Conversion

A new methodology for the implementation of floating-point algorithms into fixed-point DSPs under accuracy constraint has been proposed in [15]. In our methodology, the determination and the optimization of the the fixed-point specification are directed by the accuracy constraint. Moreover, the DSP architecture is completely taken into account. The code generation and the fixed-point conversion process are coupled to obtain an optimized fixed-point specification. The different phases of our methodology are presented in figure 1.

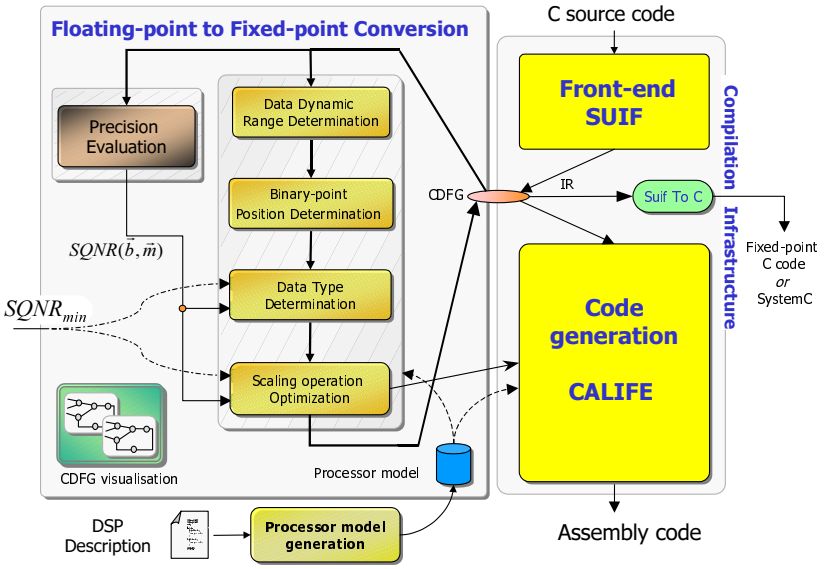


Fig. 1. Framework structure.

First, the floating-point C source algorithm is transformed with the SUIF front-end, into an intermediate representation (IR) described as a Control Data Flow Graph (CDFG). The first stage of the methodology, corresponding to the data dynamic range evaluation, is based on an analytical approach. A combination of the L1 and Chebychev norms [21] and the results of the arithmetic interval theory [9] allows to determine the data dynamic range in non-recursive systems and in recursive linear time-invariant systems.

The results obtained are used for the determination of the data binary-point position to avoid overflows. For the different application DFGs (Data Flow Graph), the binary-point positions are determined during the DFG traversal from the sources to the sinks. For each data and operator, a rule is applied for determining the position of the binary-point. This approach enables to manage adder with guard bits. In this stage, the scaling operations required to obtain a valid fixed-point specification are inserted.

The third stage corresponds to the data word-length determination to obtain a complete fixed-point format for each data. It is detailed in section five.

Finally, the scaling operation localization are optimized to minimize the code execution time as long as the accuracy constraint is fulfilled. The execution time is modified through the scaling operation moving. This stage aim is to find the scaling operation location which allows to minimize the execution time and to fulfill the accuracy constraint. Given that the instruction level parallelism is limited for conventional DSPs, the scaling operation execution time can be estimated from the execution time of the instructions used for implementing this operation. For processors with instruction level parallelism, the execution

time estimation must be coupled with the scheduling stage to take account of the partial instructions which are executed in parallel. The back-end of the compilation infrastructure is based on a retargetable code generation tool [4].

The determination and the optimization of the fixed-point specification are made under accuracy constraint. The accuracy is evaluated trough the Signal to Quantization Noise Ratio (SQNR) metric. An analytical method [18] that enables to obtain automatically the SQNR expression from the application CDFG is used in this flow.

5 Data Type Determination

In the fixed-point conversion design flow, each data type (word-length) is determined to obtain a complete fixed-point format for each data of the CDFG. This module must allow to explore the diversity of the data types available in recent DSPs as explained in section three. To increase the computation accuracy, multiple-precision operations can be used. Nevertheless, the execution time is more important. To reduce the computation execution time, data parallelism can be exploited trough Sub-Word Parallelism operations in recent DSPs. Consequently, the precision of the computation is reduced.

The main goal of the code generation process is to minimize the code execution time under a given accuracy constraint. Thus, our methodology selects the instructions which will respect the global accuracy constraint and minimize the code execution time. The methodology structure is presented in figure 2. Before to start the optimization process, different stages must be achieved. They correspond to the accuracy evaluation, the instruction selection and the application execution time estimation.

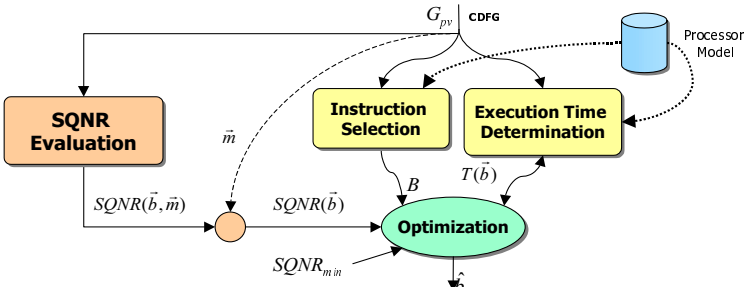


Fig. 2. Structure of the data type determination process. This process is applied to the CDFG (G_{pv}) obtained after the determination of the data binary-point positions.

5.1 Accuracy Evaluation

The most common used criteria for evaluating the computation accuracy is the Signal to Quantization Noise Ratio (SQNR) [12][10]. Traditional methods [5][12]

are based on fixed-point simulations. But these techniques lead to huge fixed-point optimization time. Indeed, multiple simulations are needed for this optimization process, since a new fixed-point simulation is required when any fixed-point data format is modified. Thus, the methodology based on an analytical method and proposed in [18], is used. It allows to determine automatically the analytical expression of the Signal to Quantization Noise Ratio for a CDFG. In a fixed-point system, the output quantization noise is due to the propagation of the different noise sources generated during cast operations. The output noise power expression is computed from the noise source statistical parameters and the gain between the output and each noise source. For linear time-invariant systems, these gains are determined with the help of the transfer function concept.

The CDFG is made up of N_o operations o_i. Let b_i and m_i be the word-length and binary-point position of the operation o_i operands. Let $\overrightarrow{b} = [b_1, b_2, \ldots, b_i, \ldots b_{N_o}]$ and $\overrightarrow{m} = [m_1, m_2, \ldots, m_i, \ldots m_{N_o}]$ be the vectors specifying respectively the word-length and the binary-point position of all CDFG operation operands. The SQNR expression is obtained according to the vector $\overrightarrow{b}$ and $\overrightarrow{m}$.

5.2 Estimation of the Code Execution Time

DSP Processor Modelization. The processor is modelized by a Data Flow (DF) instruction set. These instructions implement arithmetic operations and data transfers between the memory and the processing unit. The instructions are obtained from one or several instructions of the processor instruction set. Each DF instruction j_k is characterized by its function γ_k, by its operand word-length b_k and by its execution time t_k. This execution time is obtained from the processor model. For SWP instructions, the execution time is set to the processor instruction execution time divided by the number of operations achieved in parallel. For the multiple-precision instructions, the execution time is the sum of the execution time of the processor instructions used for implementing this operation. A processor model example is presented in figure 3.a.

Execution Time Estimation. The aim of this section is to estimate the global application execution time according to the instructions selected for each CDFG operation. Nevertheless, the goal is not to obtain an exact execution time estimation but to compare and to select two instruction series. Thus, a simple estimation model is used for evaluating the application execution time T. This time T is computed from the execution time t_i and the number of executions n_i of each operation o_i as follows

$$T = \sum_i t_i.n_i \tag{1}$$

This estimation method is based on the sum of the instruction execution times and leads to accurate results for DSPs without instruction parallelism. For DSPs with instruction level parallelism (ILP), this method does not take account of the instructions executed in parallel. Nevertheless, this estimation

gives results which allow to compare efficiently two instruction series in the case of processor with ILP.

For classical and SWP instructions, the gains due to the transformation (code parallelization) of the vertical code into an horizontal one are closed. Indeed, the two instruction series use the same functional units at the same clock cycles. The difference lies in the functionality of the processor unit. For SWP instructions, the functional units manipulate fractions of word instead of the entire word.

A multiple-precision instruction corresponds to a serie of classical instructions. Thus, in the best case and after the scheduling stage, the multiple-precision instruction execution time can be equal to the execution time of the classical precision instructions. In this case, the classical operations must be favored if the precision constraint is fulfilled to reduce the data memory size. Therefore, the multiple-precision instruction execution time is set to the maximal value to select these multiple-precision instructions only if the classical instructions can not fulfill the precision constraint.

This approach for the code execution time estimation can be improved with more accurate techniques such as those presented in [7][22].

5.3 Data Word-Length Optimization

For each CDFG operation o_i, the different instructions, which allow to achieved o_i, are selected. Let I_i, be the set specifying the instructions selected for the operation o_i. Let B_i, be the set specifying all the word-lengths which can be taken by the operation o_i operands. Thus, for each operation o_i, the optimal word-length $\widehat{b_i}$ ($\widehat{b_i} \in B_i$) which allows to minimize the global execution time $T(\overrightarrow{b})$ and to respect the minimal precision constraint must be selected. Consequently, the application execution time $T(\overrightarrow{b})$ is minimized as long as the accuracy constraint ($SQNR_{min}$) is fulfilled as described in equation 2. This optimization process is illustrated with a FIR filter example in figure 3.

$$\min_{\overrightarrow{b} \in B} \left(T(\overrightarrow{b}) \right) \quad \text{such as} \quad SQNR(\overrightarrow{b}) \geq SQNR_{min} \tag{2}$$

Given that the number of values for each variable b_i is limited, the optimization problem can be modelized with a tree and a *branch and bound* algorithm can be used to obtain the optimized solution. This technique leads to an exponential optimization time. Consequently, the success of this approach is based on the limitation of the search space. Four techniques presented in the next section are used for limiting the search space.

5.4 Search Space Limitation

Instruction Combination Restriction. The modelization of this optimization problem with a tree allows to enumerate exhaustively all the solutions. Nevertheless, all the instruction combinations are not valid. For illustrating this problem, an example is presented in figure 4. The operation inputs and output

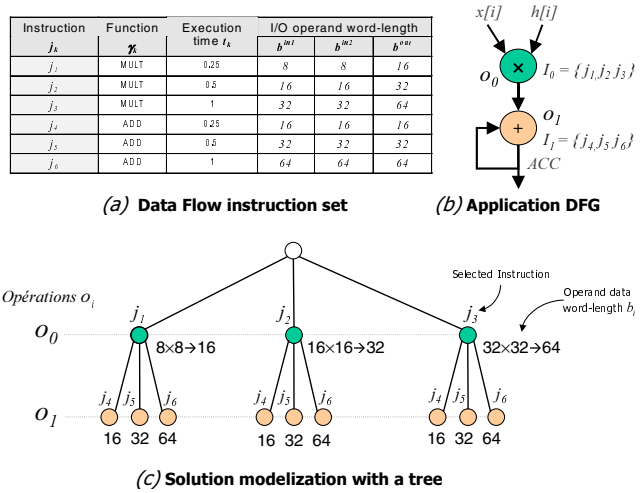


Fig. 3. Modelization of the data word-length optimization process for a FIR filter. (a) Modelization of the processor data flow instructions (b) FIR filter Data Flow Graph (c) Modelization with a tree of the different solutions of the optimization problem.

are annotated with their fixed-point format. Given that the operation o_l input is the result of the operation o_k, the number of bits for the fractional part of the o_l input can not be strictly greater than the number of bit for the fractional part of the o_k output. Thus, the instruction tested for the operation o_l is valid if the following conditions are fulfilled

$$\begin{aligned} n_k^s &\geq n_l^e \\ n_l^s &\geq n_h^e \end{aligned} \tag{3}$$

If the conditions 3 are not respected, the exploration of the subtree is stopped and a new instruction is tested for the operation o_l. This technique allows to reduce significantly the search space.

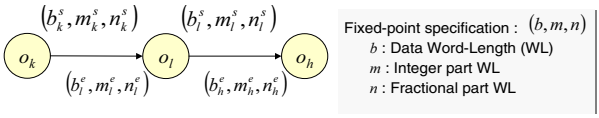


Fig. 4. Data flow example.

Partial Solution Evaluation. In the *branch-and-bound* algorithm, the partial solutions are evaluated to stop the tree exploration, if they can not lead to the best solution.

At the tree level l, the exploration of the subtree induced by the node representing $\widehat{b_l}$ can be stopped if the minimal execution time which can be obtained

during the exploration of this subtree is greater than the minimal execution time which has already been obtained. Given that only the word-lengths b_0 to b_l are defined, the minimal execution time is determined by selecting for the operation o_j ($j \in [l+1, N_o]$), the instruction with the minimal execution time t_j. In most of the case, it is equivalent to select for the operation o_j, the instruction with the minimal operand word-length ($\underline{b}_j$). In this case, for each tree level l, the following relation is obtained

$$T([\widehat{b}_0, \ldots, \widehat{b}_{l-1}, \widehat{b}_l, \underline{b}_{l+1}, \ldots, \underline{b}_{N_o}]) \leq T([\widehat{b}_0, \ldots, \widehat{b}_{l-1}, \widehat{b}_l, b_{l+1}, \ldots, b_{N_o}]) \tag{4}$$

At the tree level l, the exploration of the subtree induced by the node representing $\widehat{b}_l$ can be stopped if the maximal SQNR which can be obtained during the exploration of this subtree is lower than the precision constraint ($SQNR_{min}$). The SQNR maximal value is obtained by fixing the word-lengths b_j ($j \in [l+1, N_o]$) to their maximal values. Indeed, given that the SQNR is a monotonous and rising function, the SQNR maximal value is obtained for the maximal operand word-length ($\overline{b}_j$). Thus, for each tree level l, the following relation is obtained

$$SQNR([\widehat{b}_0, \ldots, \widehat{b}_{l-1}, \widehat{b}_l, \overline{b}_{l+1}, \ldots, \overline{b}_{N_o}]) \geq SQNR([\widehat{b}_0, \ldots, \widehat{b}_{l-1}, \widehat{b}_l, b_{l+1}, \ldots, b_{N_o}]) \tag{5}$$

Node Evaluation Order. This optimization technique based on a tree exploration is sensitive to the node evaluation order. To find quickly a good solution for reducing the search space, the variables with the most influence on the optimization process must be evaluated first. The variables are sorted by their influence on the global execution time and the SQNR.

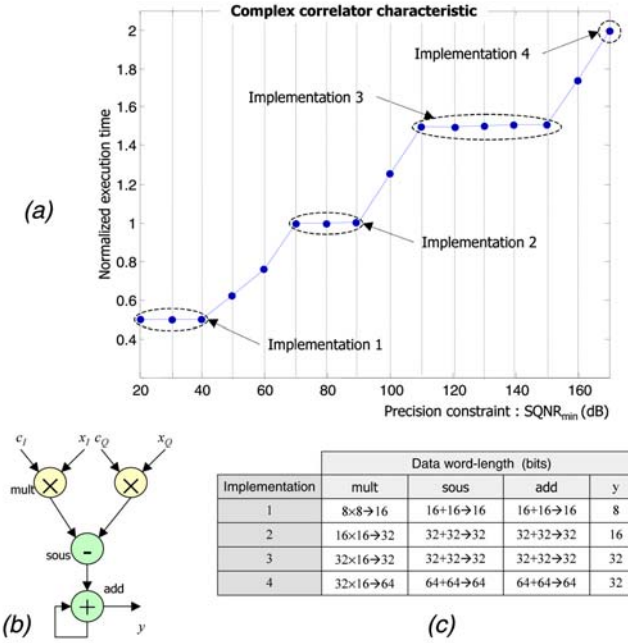
Reduction of the Number of Values per Variable. For applications with a great number of variables, the optimization time can become huge. To obtain reasonable optimization time, the optimization is achieved in two steps. Firstly, the variables corresponding to the data word-length are considered as positive integer numbers and a classical optimization technique is used for minimizing the code execution time under accuracy constraint. Let $\widetilde{b}_i$ be the optimized solution obtained with this technique for the variable b_i. Secondly, the technique based on the *branch-and-bound* algorithm presented previously is applied with a reduced number of values per variable. For each variable b_i, only the values which are member of B_i and immediately higher and lower than $\widetilde{b}_i$ are retained. Thus only, two values are tested for each variable and the search space is dramatically reduced.

6 Experiments

6.1 Complex Correlator

The complex correlator application is used for the de-spreading operation in a CDMA receiver [19]. It achieves the correlation between a complex signal x

and a complex code c. In this experiment, the complex correlator characteristic $T_{opt} = f(SQNR_{min})$ is determined for the TMS320C64x architecture model. This characteristic defines the optimized execution time obtained for a given SQNR constraint. It allows to analyse the trade-off between the execution time and the computation accuracy offered by the processor. The results are presented in figure 5. This characteristic is obtained by applying our word-length optimization method for different SQNR constraints. Each point (ρ_o, T_o) of the curve $T_{opt} = f(SQNR_{min})$ represents the optimized execution time (T_o) obtained for the SNQR constraint ρ_o. This experiment has been achieved with no constraint on the data types for the application input and output. The execution time is normalized with the execution time obtained with a classical implementation called implementation 2.



Implementation	Data word-length (bits)			
	mult	sous	add	y
1	8×8→16	16+16→16	16+16→16	8
2	16×16→32	32+32→32	32+32→32	16
3	32×16→32	32+32→32	32+32→32	32
4	32×16→64	64+64→64	64+64→64	32

Fig. 5. (a) Complex correlator characteristic $T_{opt} = f(SQNR_{min})$. For each SQNR constraint $(SQNR_{min})$, the optimized execution time T_{opt} is reported. (b) Data Flow Graph for the correlator real part computation (c) Details of the different particular implementations obtained for the complex correlator characteristic.

The curve evolves by stage. The fixed-point specifications associated with each particular implementation are detailed in the table presented in figure 5.c. The implementations 1, 2 and 4 are based on the same structure. The data are stored in memory with b_n bits. The multiplication leads to a result with $2.b_n$ bits and the addition operand word-length is equal to $2.b_n$. For the implementations

1, 2 and 4, the word-length b_n is respectively equal to 8, 16 and 32 bits. For the implementation 3, the data are stored in memory with 32 bits and the addition operand word-length is equal to 32 bits. Thus, the computations are achieved on single precision with this implementation. It allows to reduce the application execution time compare to the implementation 4. This characteristic underlines the trade-off between the execution time and the computation accuracy obtained for new DSP generation with SWP capacities. For the different implementations, the optimization time is less than 8 seconds.

6.2 Second Order IIR Filter

The previous example underlines our methodology interest for DSPs with SWP capacities. Nevertheless, this methodology can be used to implement applications which require high precision into conventional DSPs. Indeed, this methodology allows to select classical or multiple-precision instructions. For illustrating these aspects, a second order Infinite Impulse Response (IIR) filter is implemented into the DSP TMS320C54x.

The SQNR obtained with an implementation based on classical operations is low (50 dB). To increase the computation accuracy, double precision operations are used. The characteristic $T_{opt} = f(SQNR_{min})$ obtained with the TMS320C54x architecture model is presented in figure 6. The execution time is normalized with the classical implementation (implementation 1) execution time.

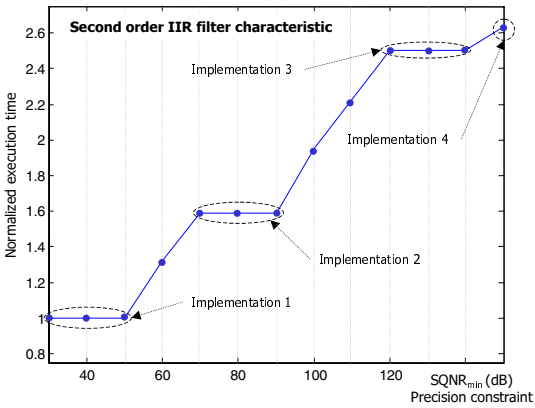
The implementation 2 leads to an interesting trade-off. For the implementations 1 and 2, the input and output filter word-lengths are equal to 16 bits. Thus, the input and output fixed-point specifications of implementation 1 and 2 are identical. Nevertheless, the implementation 2 allows to increase the output SQNR of 40 dB and leads to an execution time growth of only 60% compared to implementation 1. For this solution, the recursive part of the filter uses double precision operations and the data z_y are stored in memory with 32 bits to limit the accuracy loss. For the different implementations, the optimization time is less than 5 seconds.

6.3 WCDMA Receiver

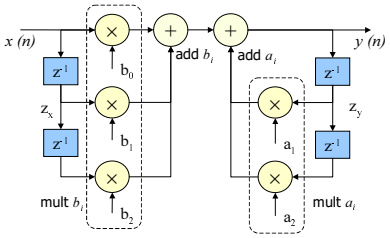
A more complex application, corresponding to a WCDMA rake receiver, has been tested. This application is integrated into third-generation radio communication systems (UMTS) [19]. This rake receiver is made up of a symbol estimation part and a synchronization part [16]. Before to start the fixed-point conversion process, the minimal precision constraint ($SQNR_{min}$) must be defined. The SQNR minimal value is obtained with the technique detailed in [16] according to the receiver performances evaluated through the Bit Error Rate (BER).

The methodology has been used to obtain the application fixed-point specification. The target processor is the TMS320C64x high performance VLIW DSP. The different scaling operations are moved towards the coefficients to reduce the code execution time. The opportunities offered by the SWP instructions allow to

(a)



(b)



(c)

Implementation	Data word-length (bits)						
	x, z_x	mult b_i	add b_i	mult a_i	add a_i	z_y	y
1	16	16×16→32	32+32→32	16×16→32	32+32→32	16	16
2	16	16×16→32	32+32→32	32×32→64	32+32→32	32	16
3	32	32×32→64	32+32→32	32×32→64	32+32→32	32	32
4	32	32×32→64	64+64→64	32×32→64	64+64→64	32	32

Fig. 6. (a) IIR filter characteristic $T_{opt} = f(SQNR_{min})$. For each SQNR constraint ($SQNR_{min}$), the optimized execution time T_{opt} is reported. (b) Data Flow Graph for the IIR filter (c) Details of the different particular implementations obtained for the IIR filter characteristic.

decrease significantly the code execution time. For the symbol estimation part, the execution time is reduced from a factor of 3.5 compared to a solution based only on classical instructions. For the synchronization part, the execution time is divided by a factor of 3.1. The different techniques proposed for the limitation of the search space allow to obtain an optimization time lower than 3 minutes for this application.

7 Conclusion

In this paper, a new methodology for the implementation of floating-point algorithms into fixed-point architectures under accuracy constraint has been presented. Compared to a manual based approach, this kind of tool enables to reduce significantly the application time-to-market. The stage for the data word-length

selection has been detailed. The aim of this phase is to select the set of instructions which allows to respect the global accuracy constraint and to minimize the code execution time. The different techniques used to limite the search space to obtain reasonable optimization time have been presented. The experiment results show the different possible trade-offs between the accuracy and the execution time. Our method underlines the different opportunities offered by DSPs with SWP capacities.

References

1. 3DSP. *SP-5 Fixed-point Signal Processor Core*. 3DSP Corporation, July 1999.
2. Analog Device. *TigerSHARC Hardware Specification*. Analog Device, December 1999.
3. CEVA. *CEVA-X1620 Datasheet*. CEVA, 2004.
4. F. Charot and V. Messe. A Flexible Code Generation Framework for the Design of Application Specific Programmable Processors. In *7th international workshop on Hardware/Software Codesign, CODES'99*, Rome, Italy, May 1999.
5. M. Coors, H. Keding, O. Luthje, and H. Meyr. Fast Bit-True Simulation. In *Design Automation Conference 2001 (DAC-01)*, Las Vegas, US, June 2001.
6. D. Esftathiou, J. Fridman, and Z. Zvonar. Recent developpements in enabling technologies for the software-defined radio. *IEEE Communication Magazine*, pages 112–117, August 1999.
7. N. Ghazal, R. Newton, and J. Rabaey. Predicting performance potential of modern dsps. In *DAC 00*, 2000.
8. T. Grötker, E. Multhaup, and O.Mauss. Evaluation of HW/SW Tradeoffs Using Behavioral Synthesis. In *7th International Conference on Signal Processing Applications and Technology (ICSPAT 96)*, Boston, October 1996.
9. R. Kearfott. Interval Computations: Introduction, Uses, and Resources. *Euromath Bulletin*, 2(1):95–112, 1996.
10. H. Keding, F. Hurtgen, M. Willems, and M. Coors. Transformation of Floating-Point into Fixed-Point Algorithms by Interpolation Applying a Statistical Approach. In *9th International Conference on Signal Processing Applications and Technology (ICSPAT 98)*, 1998.
11. H. Keding, M. Willems, M. Coors, and H. Meyr. FRIDGE: A Fixed-Point Design And Simulation Environment. In *Design, Automation and Test in Europe 1998 (DATE-98)*, 1998.
12. S. Kim, K. Kum, and S. Wonyong. Fixed-Point Optimization Utility for C and C++ Based Digital Signal Processing Programs. *IEEE Transactions on Circuits and Systems II*, 45(11), November 1998.
13. K. Kum, J. Kang, and W. Sung. AUTOSCALER for C: An optimizing floating-point to integer C program converter for fixed-point digital signal processors. *IEEE Transactions on Circuits and Systems II - Analog and Digital Signal Processing*, 47:840–848, September 2000.
14. P. Lapsley, J. Bier, A. Shoham, and E. A. Lee. *DSP Processor Fundamentals: Architectures and Features*. Berkeley Design Technology, Inc, Fremont, CA, 1996.
15. D. Menard, D. Chillet, F.Charot, and O. Sentieys. Automatic Floating-point to Fixed-point Conversion for DSP Code Generation. In *International Conference on Compilers, Architectures and Synthesis for Embedded Systems 2002 (CASES 2002)*, Grenoble, October 2002.

16. D. Menard, M. Guitton, S. Pillement, and O. Sentieys. Design and Implementation of WCDMA Platforms : Challenges and Trade-offs. In *nternational Signal Processing Conference (ISPC 03)*, Dallas, April 2003.
17. D. Menard, P. Quemerais, and O. Sentieys. Influence of fixed-point DSP architecture on computation accuracy. In *XI European Signal Processing Conference (EUSIPCO 2002)*, Toulouse, September 2002.
18. D. Menard and O. Sentieys. Automatic Evaluation of the Accuracy of Fixed-point Algorithms. In *Design, Automation and Test in Europe 2002 (DATE-02)*, Paris, March 2002.
19. T. Ojanper and R. Prasad. *WCDMA : Towards IP mobility and mobile internet.* Artech House Universal Personal Communications Series, 2002.
20. B. Ovadia and G. Wertheizer. PalmDSPCore - Dual MAC and Parallel Modular Architecture. In *10th International Conference on Signal Processing Applications and Technology (ICSPAT 99)*, Orlando, November 1999. Miller Freeman Inc.
21. T. Parks and C. Burrus. *Digital Filter Design.* Jhon Willey and Sons Inc, 1987.
22. A. Pegatoquet, E. Gresset, M. Auguin, and L. Bianco. Rapid Development of Optimized DSP Code from a High Level Description through Software Estimations. In *DAC 99*, 1999.
23. Texas Instruments. *TMS320C64x Technical Overview.* Texas Instruments, February 2000.
24. S. Wichman and N. Goel. *The Second Generation ZSP DSP*, 2002.
25. M. Willems, V. Bursgens, H. Keding, and H. Meyr. System Level Fixed-Point Design Based On An Interpolative Approach. In *Design Automation Conference 1997 (DAC 97)*, June 1997.
26. M. Willems, V. Bursgens, and H. Meyr. FRIDGE: Floating-Point Programming of Fixed-Point Digital Signal Processors. In *8th International Conference on Signal Processing Applications and Technology (ICSPAT 97)*, 1997.