Automatic Floating-point to Fixed-point Conversion for DSP Code Generation

Daniel Menard, Daniel Chillet
R2D2 Project - INRIA
LASTI - University of Rennes I
6, rue de kerampont
22300 Lannion, FRANCE
menard@enssat.fr, chillet@enssat.fr

François Charot, Olivier Sentieys
R2D2 Project - INRIA
IRISA
Campus de Beaulieu
35042 Rennes cedex, FRANCE
charot@irisa.fr, sentieys@irisa.fr

ABSTRACT

The development of methodologies for the automatic implementation of floating-point algorithms in fixed-point architectures is required for the minimization of cost, power consumption and time to market of digital signal processing applications. In this paper, a new methodology of implementation in Digital Signal Processors (DSP) under accuracy constraint is presented. In comparison with the existing methodologies, the DSP architecture is completely taken into account for optimizing the execution time under accuracy constraint. The justification and the different stages of our methodology are presented.

1. INTRODUCTION

Most digital signal processing algorithms are specified with floating-point data types but they are finally implemented in fixed-point architectures in order to satisfy the cost and power consumption constraints of embedded systems. The manual transformation of floating-point data into fixed-point data is a time-consuming and error prone task. For reducing the time-to-market of applications, high-level development tools, which allow the automation of some tasks, are required. The manual conversion to the fixed-point level hinders the reduction of the development time. Indeed, some experiments [3] 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 [7, 17].

The aim of the methodology for Digital Signal Processors (DSP), is to define the optimal fixed-point data formats which maximize the accuracy and minimize the size and the execution time of the code. Existing methodologies [11, 17] achieve a floating-point to fixed-point transformation 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 influence of the fixed-point DSP architecture on the computation accuracy underlines the necessity of taking into account the DSP architecture for optimizing the data coding [12].

In this paper, a new methodology for the implementation of floating-point algorithms in fixed-point architectures under accuracy constraint is presented. After a presentation of the available methodologies, a new methodology is defined. The different elements of our methodology are detailed in the fourth section. Finally, some results obtained with the tool developed for implementing this methodology are given.

2. RELATED WORK

In this section the different available methodologies for the automatic implementation of floating-point algorithms in fixed-point architectures are presented. In [8] the method proposed for the floating-point to fixed-point conversion is achieved after the code generation process. It allows to implement floating-point algorithms in the TMS320C25/50 fixed-point DSP of Texas Instruments. This methodology is specialized for this particular architecture and can not be transposed to other kinds of architecture. The two methodologies presented below achieve the floating-point to fixed-point transformation at the source code level.

The FRIDGE [6] 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 in order to specify some rules for the entire system (maximal data word-length, casting rules). The second step called *interpolation* [16][6] corresponds to the determination for each data of the word-length of the integer and fractional parts. The fixed-point data formats are obtained from a set of propagation rules and the analysis of the program control flow. This step leads to an entire fixed-point specification of the application. This description is simulated in order to verify if the accuracy constrains are fulfilled. The commercial tool *CoCentric Fixed-point Designer* proposed by Synopsys is based on this approach.

In [17] 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 degrees of freedom due to the position of the source data 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 accuracy and the execution time are not optimized through the modification of the fixed-point format of some relevant variables.

The aim of the tool presented in [10, 11] is to transform a floating-point C source code into an ANSI-C code with integer data types in order to be independent of the target architecture. Moreover, a fixed-point format optimization is done in order 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 informations obtained with a statistical method [9]. The reduction of the number of scaling operations 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 takes account of the number of occurrence of each scaling operation and depends of the scaling capacities of the target processor. For processor with a barrel shifter, the cost of a scaling operation is equal to one cycle, otherwise the number of cycles required for a shift of n bits is equal to n cycles.

This methodology allows to convert a floating-point C code into a C code with integer data types. Moreover, 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. Specific elements of the architecture are not taken into account for optimizing the fixed-point data formats. Moreover, the architecture model used for the minimization of the scaling operations do not take account of the specialized shift registers which allow to make some specific shift operations without supplementary cycles. 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.

3. METHODOLOGY DEFINITION

In this part a new methodology for the automatic implementation of a floating-point algorithm in a fixed-point DSP is proposed. The structure of the methodology has been defined from the analysis of the influence of the architecture on the computation accuracy and from the study of the interaction between the fixed-point data coding and the code generation process. The results of these studies are summarized below.

3.1 Architecture Influence

In [12], the influence of the fixed-point DSP architecture on the computation accuracy has been analyzed. Different elements of the architecture like the natural word-length of the DSP, the number of accumulator guard bits, the scaling capacities and the available quantification law, influence the accuracy of the computation. New DSP architectures like TMS320C64x (Texas Instruments), TigerSharc (Analog Device), OneDSP (Siroyan), SP5 or UniPhy (3DSP) support a wide range of data types through sub-word parallelism (SWP) instructions. An operator (multiplier, adder, shifter) of word-length N is split in order to execute k operations in parallel on sub-word of word-length N/k. These SWP instructions require to explore the different opportunities offered by these architectures for optimizing the implementation. The results of the experiments presented in [12], show the necessity of taking the different elements of the DSP architecture into account in order to optimize the data coding.

3.2 Code Generation and Data Coding

The goal of this section is to analyze the interaction between the fixed-point data coding and the different stages of the code generation process. The aim of the code selection step is to select the set of instructions which allows to achieved the algorithm operations as efficient as possible. The code selection requires that the data types (data word-length) of the input and output operands of each operation are defined. The transfer of the data type selection during the code generation will lead to more complex code generation stages. Thus, the word-length of each data has to be defined before the code selection stage.

The register allocation step assigns a processor storage unit to each variable. This stage defines if a variable is stored in a register or in memory. Indeed, given that the number of registers is limited, some variables have to be spilled in memory. In traditional DSP, these spilling operations are very frequent with the limited number of registers due to the specialization of the architecture [18]. The execution time and the code size are increased due to the introduction of memory read and write instructions. In most of the DSP, the calculations are done in double precision inside the computation unit. The transfer of the data in simple precision allows to reduce the transfer overhead. However, this format conversion operation will introduce a quantization noise source and reduces the global application accuracy . Thus, these spilling operations have to be taken into account for coding the fixed-point data. Moreover, the evaluation of the real accuracy of the implementation can be achieved only after this register allocation step.

The scheduling stage defines the execution moment of each instruction. These moments are defined in order to respect the semantic of the application an to minimize the global execution time. For VLIW processors, this scheduling task achieves a code compaction phase which allows to group together partial instructions which can be executed in parallel. The aim of our methodology is to minimize the execution time under accuracy constraint. Thus the costly scaling operations have to be moved in order to reduce the execution time. For processor with instruction level parallelism the cost of a scaling operation depends on the opportunity of executing this operation in parallel with other instructions and can be evaluated only during the scheduling stage. These aspects are quantified in [12].

3.3 Methodology Presentation

The study of the available methodologies and the analysis of the influence of the architecture and the code generation lead to the definition of a new methodology for the implementation of floating-point algorithms in fixed-point DSP under accuracy constraint. In our methodology, the determination and optimization of the data format are directed by the accuracy constraint. Moreover, the DSP architecture is completely taken into account. The different phases of our methodology are presented in figure 1.

The first stage of the methodology is the evaluation of the data dynamic range. The results obtained are used for the determination of the binary point of the data in order to avoid an overflow. Then, the word-lengths of each data are defined in order to take account of the diversity of the data types available in DSP. Finally, the data formats are optimized in order to minimize the code execution time as long as the accuracy constraint is fulfilled. The determination and optimization of the data formats are made under accuracy constraint. The Signal to Quantization Noise Ratio (SQNR) is used for evaluating this accuracy . The analytical expression of this metric is automatically obtained with the methodology detailed in [13, 14]. For linear time-invariant systems, the method is based on the automatic computation of the transfer functions between the output and the quantization noise sources.

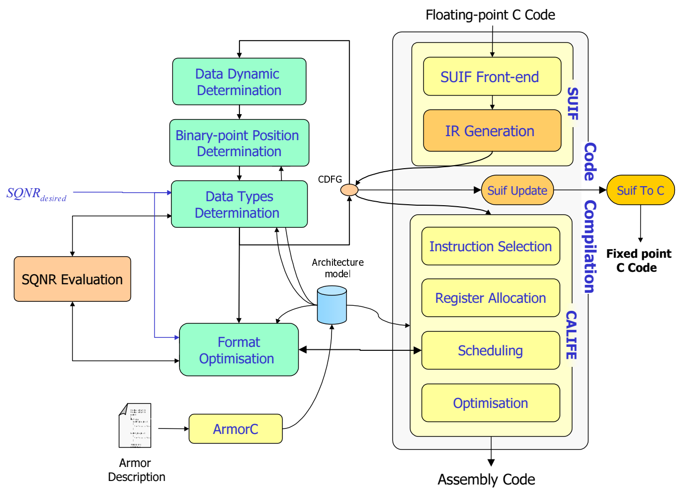

Figure 1: Methodology structure

4. METHODOLOGY DESCRIPTION

The aim of this section is to detail the different parts of our methodology. First, the floating-point C source algorithm is transformed in an intermediate representation (IR). This IR specifies the application with a Control Data Flow Graph (CDFG) which is generated from the IR obtained with the SUIF front-end [19]. This CDFG is made-up of Control Flow Graphs (CFG) which represent the control structures of the application. Each computation core of a basic bloc of the CFG is represented with a Data Flow Graph (DFG). Moreover, each control structure bloc contains a specification of its input and output data. The code generation is achieved with the flexible code generation tool CALIFE presented in [1] and the processor is described with the ARMOR language [2].

4.1 Data Dynamic Range Determination

Two kinds of method can be used for evaluating the data dynamic range of an application. The dynamic range of a data can be computed from its statistical parameters which are obtained with a floating-point simulation. These approaches allow to estimate accurately the dynamic range with the help of the signal characteristics. These methods guarantee a low probability of overflow for signal with the same characteristics. Nevertheless, overflow can occur for signal with different statistical properties.

The second kind of method corresponds to analytical approaches which allow to compute the expression of the dynamic range from the dynamic range of the inputs. These methods guarantee no overflow but it leads to a more conservative estimation. Indeed, the dynamic range expression is computed in the worst case. The determination of the data dynamic range can be obtained with the Interval Arithmetic theory [4]. The data dynamic range of an operator output is obtained from the dynamic of its inputs. A worst case dynamic range propagation rule is defined for each kind of operator. The dynamic range of each data is obtained with the help of the propagation rules during the traversing of the graph which represents the application. Thus, this technique can not be used if there is cycles in the graph like in the case of recursive structures. The determination of the data dynamic range in the tool FRIDGE [6] is based on this technique [16].

For the linear time-invariant systems, the data dynamic range can be computed from the L1 or Chebychev norm [15] according to the frequency characteristics of the input signal. These norms allow to compute the dynamic range of a data in the case of non-recursive and recursive structures with the help of the computation of the transfer functions between the data and each input.

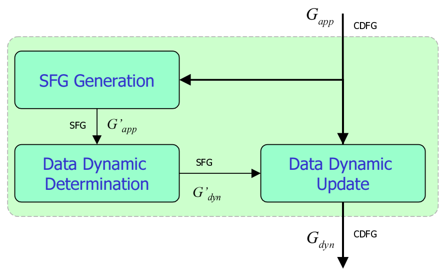

Figure 2: Data Dynamic Range Determination

These two approaches have been combined for determining the data dynamic range in non-recursive systems and in recursive linear time-invariant systems. The structure of this module is presented in figure 2. The input of this module is the intermediate representation which is a Control Data Flow Graph (CDFG). The first step transforms the CDFG in a Signal Flow Graph (SFG). The control structures are eliminated in order to obtain a Data Flow Graph (DFG). Then, the temporal informations are used for inserting the delay operations.

The second step of the transformation is the computation of the data dynamic range of each data of the DFG.

For non-recursive structures the dynamic range informations are obtained by traversing the graph from the sources to the sinks. For each operator a propagation rule as defined in [4], is applied. For recursive linear time-invariant structures the transfer functions between the critical data and the inputs are determined with the technique presented in [14]. These critical data correspond to the output of the addition or substraction operators. Then, the dynamic range is computed from the input dynamic range with the L1 or Chebychev norm. For the rest of the data, the dynamic range is obtained with the propagation rule technique.

The last step of this module is the annotation of the data of the CDFG with the dynamic range. For a data with only one instantiation in the CDFG, its dynamic range is equal to the dynamic range of the equivalent data in the SFG. For data defined as vector (i.e. tabular) and used in loop, the dynamic range of the vector in the CDFG corresponds to the greatest value of the different elements of this vector used in the SFG. The determination of the dynamic range is more complex in the case of data with multiple instantiations like in the FFT butterfly where the inputs and the outputs of the butterfly are stored in the same variables. The dynamic range of the output vector is multiplied by a factor of two at each stage of the FFT. Thus, the fixed-point format of the output vector must evolve at each stage. The first and final values of the dynamic range of the vector X are specified through the input and output loop bloc and the evolution of the dynamic range of the vector X are specified through the input and output CFG bloc which represents the i^{th} stage of the FFT. This is illustrated in figure 3. Consequently, the expression of the evolution of the dynamic range of a multiple instantiations data has to be determined.

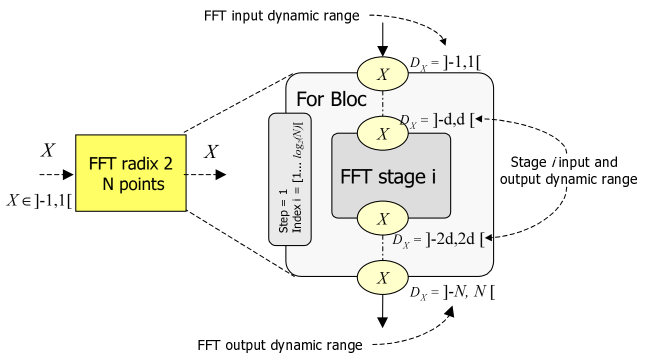

Figure 3: Data Dynamic Range for a FFT

4.2 Binary-Point Position Determination

The aim of this transformation is to obtain a correct fixed-point specification of the application which guarantees no overflow. Moreover, this transformation must allow to respect the different fixed-point arithmetic rules. Thus, scaling operations are included in the application in order to fit the fixed-point format of a data to its dynamic range or to align the binary-point of the adder inputs. The input of this transformation is the CDFG G_{Dyn} where all the data are annotated with their dynamic range. The output is a CDFG where all the data are annotated with their binary-point position. A hierarchical approach is used for determining the data binary-point position. First, all the DFG of the application are independently proceeded and then a global processing is applied at the CDFG in order to obtain a coherent fixed-point specification.

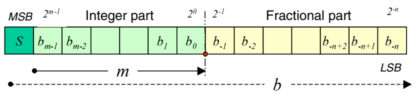

Figure 4: Fixed-point data specification

For determining the binary-point position of the data of a DFG, the graph representing the DFG is traversed from the sources to the sinks. For each data and operator a rule is applied for determining the position of the binary point. This kind of technique can be applied only on directed acyclic graph (DAG). Thus, the graph representing a DFG is firstly dismantled into a DAG if it contains cycles.

The binary-point position m_x of a data x is referenced from the most significant bit as presented in figure 4. For a data x the binary-point position is obtained directly from the dynamic range $\mathcal{D}_x$ with the following relation

$$m_x = \lceil \log_2(\mathcal{D}_x) \rceil \quad \text{with} \quad \mathcal{D}_x = \max_n(|x(n)|) \tag{1}$$

A binary-point position is assigned to each input and output operator $(m_{x'}, m_{y'}, m_{z'})$ as presented in figure 5. A propagation rule has been defined for each kind of operator. This rule defines the value of $m_{x'}, m_{y'}, m_{z'}$ according to the binary-point position of the input and output data (m_x, m_y, m_z). The binary-point positions of the multiplier inputs $(m_{x'}, m_{y'})$ correspond to those of the operator input data (m_x, m_y). The binary-point position of the multiplier output is directly obtained from the binary-point position of the operator inputs. Thus, the multiplier propagation rules are

$$\begin{cases} m_{x'} = m_x \\ m_{y'} = m_y \\ m_{z'} = m_{x'} + m_{y'} + 1 \end{cases} \tag{2}$$

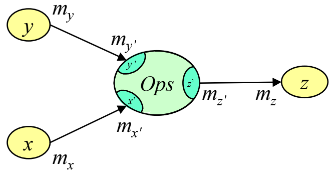

Figure 5: Operator Model

For addition and subtraction operations, a binary-point position which is common to the operator inputs has to be defined in order to align the operator input binary-point. This common position must guarantee no overflow. If there is no guard bits with the adder for storing the supplementary bits due to an overflow then this constraint must be taken into account for determining the common binary-point position. Thus, in order to avoid overflow the common binary-point position m_c is defined as follows

$$\begin{cases} m_c = \max(m_x, m_y, m_z) \\ m_{x'} = m_c \\ m_{y'} = m_c \\ m_{z'} = m_c \end{cases} \tag{3}$$

If there is guard bits with the adder, the word-length of the inputs and output are different. Then, a common reference has to be defined for comparing the binary-point positions. New binary-point positions $(m_{x''}, m_{y''}, m_{z''})$ referenced from the most significant bit of the data with the minimum word-length are computed for the inputs and the output as illustrated in figure 6. A new parameter g corresponding to the number of guard bits used by the data is introduced as follows

$$\begin{cases} m_{x''} = m_{x'} - g_x \\ m_{y''} = m_{y'} - g_y \\ m_{z''} = m_{z'} - g_z \end{cases} \quad (4)$$

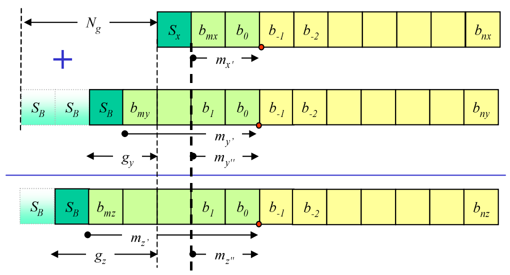

Figure 6: Binary-point position with guard bits

Given that the parameter g_z is unknown for determining m_c, it is fixed to N_g, which is the number of guard bits available with the adder.

$$m_c = \max\left(m_x - g_x, m_y - g_y, m_z - N_{gb}\right) \quad (5)$$

The real number of guard bits used by the adder output is equal to

$$\begin{cases} g_z = m_z - m_c & \text{if} \quad m_z > m_c \\ g_z = 0 & \text{if} \quad m_z \leq m_c \end{cases} \quad (6)$$

And, the binary-point positions of the adder inputs and output are equal to

$$\begin{cases} m_{x'} = m_c + g_x \\ m_{y'} = m_c + g_y \\ m_{z'} = m_c + g_z \end{cases} \quad (7)$$

The scaling operations required for obtaining a correct fixed-point specification are inserted in the CDFG. For each operator as represented in figure 5, a scaling operation is introduced if the binary-point position of the data m_x (or m_y) is different of the binary-point position of the operator input $m_{x'}$ (or $m_{y'}$). For the operator output a scaling operation is introduced if the binary-point positions $m_{z'}$ and m_z are different.

4.3 Data Type Determination

The aim of this section is to define the type (word-length) of each data in order to obtain a complete fixed-point format for each data. This module must allow to explore the diversity of the data types available in some DSP as explained in section 3.1. This module selects the instructions which will respect the global accuracy constraint and minimize the code execution time.

An instruction is characterized by the word-length of its input and output operands (b_{inst}) and the execution time of the operation (t). For SWP instructions the execution time of the operation is set to the execution time of the instruction dived by the number of computations achieved in parallel. Each operation o_i is annotated by the set of instructions I_i which allow to achieved this operation. Thus, an operation is characterized by a set of input and output data word-length B_i. The optimal data word-length b_{optim} is obtained by minimizing the application execution time $T(b)$ as long as the accuracy constraint is fulfilled

$$\min_{b \in B} \Big(T(b)\Big) \quad \text{such as} \quad SQNR(b) \geq SQNR_{min} \quad (8)$$

The most common used criteria for evaluating the accuracy of a fixed-point implementation is the Signal to Quantization Noise Ratio (SQNR) [7, 5]. The expression of this metric is automatically determined with the tool presented in [14]. A simple estimation model for evaluating the application execution time T is used. This application execution time T is computed from the execution time t_i and the number of occurrence n_i of each operation o_i as follows

$$T = \sum_i t_i . n_i \quad (9)$$

.

Given that the number of value of each variable is limited, the optimization problem can be modelized with a tree and a *branch and bound* algorithm can be used for obtaining the optimal solution. The SQNR and the global execution time are evaluated at each level of the tree in order to decide if the exploration of the subtree is stopped. The exploration of the subtree is stopped if the maximal SQNR which can be obtained with this subtree is smaller than the SQNR constraint or if the minimal execution time which can be obtained with this subtree is higher than the minimal execution time already obtained.

4.4 Fixed-point Format Optimization

The aim of this part is to optimize the fixed-point data formats in order to minimize the code execution time as long as the accuracy constraint is fulfilled. The execution time is modified through the moving of the scaling operations. For traditional DSP, the instruction level parallelism is limited. Thus, the execution time of a scaling operation can be estimated from the execution time of the instructions used for implementing this operation. The scaling operation execution time t_s is obtained from the difference between t_{ws} and t_{wos}. The time t_{ws} corresponds to the execution time of the expression tree where the scaling operation is located and t_{ws} is the execution time of the same expression tree without the scaling operation. Each execution time is estimated from the set of instructions obtained with the code selection stage applied at this expression tree. For processor with instruction level parallelism, the estimation of the execution time must be coupled with the scheduling stage in order to take account of the partial instructions which are executed in parallel.

The scaling operations are proceeded according to their execution time and they are moved as long as the accuracy constraint is fulfilled. The right shift operations are moved towards the sources of the Control Data Flow Graph (CDFG). In the case of linear systems, two alternatives are available for moving the right shift operations. These scaling operations can be moved towards the inputs of the system or towards the coefficients. For this last case the degradation of the SQNR is less important. But in the case of linear filters, the degradation of the frequency response due to the coefficient quantization is more significant.

5. EXPERIMENTATION AND RESULTS

```
float h[128] = { -0.0112970, ...,    0.1046790, ...,   -0.011297 };

void main()
{
float x[128];
float input;
float acc;
float y;
int i;

      x[0] = input;
      acc = x[0]*h[0] ;

      for(i=1; i<128; i++)
      {  acc = acc + x[i]*h[i];
         x[i] = x[i-1];
      }

     y = acc;
}
```

Figure 7: Floating-point FIR source code

```
/*
 * This file was created automatically from SUIF
 *   on Thu Jun  6 17:11:13 2002.
 */
extern int h[128];
extern void main();

int h[128] =  { -2961, ..., 27440,...,   -2961  };

extern void main()
  {
    int x[128];
    int input;
    long acc;
    int y;
    int i;

    *x =  input >> 4;
    acc = *x * *h;

    for (i = 1; i < 128; i++)
      {
        acc = acc + x[i] * h[i];
        x[i] = x[i - 1];
      }

      y = (int)(acc);
    return;
  }
```

Figure 8: Generated fixed-point FIR code

Experiments have been achieved with the different modules of our methodology already developed. That allows to implement automatically floating-point algorithms in fixed-point DSP and to optimize the data fixed-point format in the case of traditional architectures for which the instruction level parallelism is limited. For illustrating the capacities of our tool, a fixed-point C code for a FIR filter with 128 taps has been generated. This code and the original floating-point code are presented in the figures 7 and 8.

For illustrating the data type determination phase, a complex correlator has been implemented in the TMS320C64x (Texas Instruments). The quality of the implementation is evaluated through the computation of the Signal to Quantization Noise Ratio. Different SQNR constraints have been tested and the results are presented in the figure 9. Each point corresponds to the estimation of the minimal execution time obtained from the optimization process with a particular SQNR constraint. The execution time is normalized with the execution time obtained with a classical implementation. In this implementation, the data word-lengths of the input and output of the correlator are equal to 16 bits. This experiment has been achieved with no constraint on the data types of the application input and output. The best execution time is obtained when SWP instructions are used and the correlator input and output data word-lengths are equal to 8 bits. The best accuracy is obtained when the correlator input and output data word-lengths are equal to 32 bits. This experiment underlines the opportunities offered by the new DSP architectures which can manipulate a wide diversity of data types.

For illustrating the fixed-point format optimization phase the execution time of the assembly code generated has been measured with and without the optimization stage. In the first case it corresponds to the fixed-point specification after the data types determination. Different algorithms have been implemented and the results of the implementation of a 128 taps FIR filter in the TMS320C54x (T.I.) are presented in figure 10. The solution obtained without the optimization stage is called s_{fix} and the optimal solution is called s_{optim}. Moreover, the other solutions obtained with the optimization stage are presented.

These results underline the necessity of the optimizing stage for obtaining an efficient implementation and for reducing significantly the execution time. Thus, these results

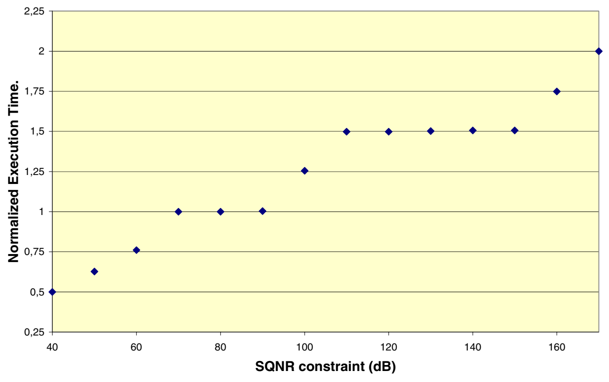

Figure 9: Complex correlator implementations

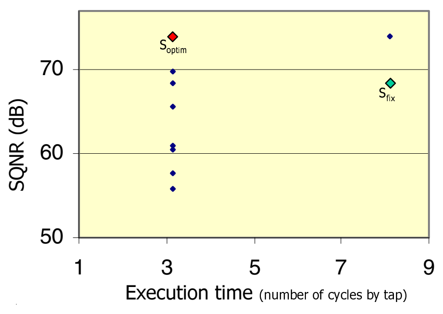

Figure 10: FIR filter implementations

show the interest of coupling the fixed-point format determination with the code generation and the interest of taking into account the processor architecture.

6. CONCLUSION

In this paper, a new methodology for the implementation of floating-point algorithms in fixed-point architectures under accuracy constraint has been presented. The modules for the determination of the dynamic range, the binary-point position and the data type have been developed. For the data format optimization module the interaction with the scheduling stage is not developed yet. The development of this module will allow to optimize the implementation of floating-point algorithms in VLIW fixed-point DSP. Nevertheless, implementations with traditional DSP can be already achieved with our tool.

In our methodology, the determination and optimization of the data formats are directed by the accuracy constraint. The DSP architecture is completely taken into account in order to optimize the accuracy of the implementation. The coupling of the fixed-point format optimization stage with the code generation process reduces the execution time as shown by the results obtained. Thus, this approach allows to reduce the time-to-market with the automatic transformation of a floating-point description into a fixed-point specification and to obtain more efficient implementations.

7. REFERENCES

[1] F. Charot, F. Djieya, and C. Wagner. Retargetable Compilation In The Service Of Interactive ASIP Design. Technical Report 1173, IRISA, Rennes, November 2000.

[2] 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.

[3] T. Grötker, E. Multhaup, and O.Mauss. Evaluation of HW/SW Tradeoffs Using Behavioral Synthesis. In *ICSPAT'96*, Boston, October 1996.

[4] R. Kearfott. Interval Computations: Introduction, Uses, and Resources. *Euromath Bulletin 2*, 2(1):95–112, 1996.

[5] H. Keding, F. Hurtgen, M. Willems, and M. Coors. Transformation of Floating-Point into Fixed-Point Algorithms by Interpolation Applying a Statistical Approach. In *ICSPAT'98*, 1998.

[6] 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.

[7] 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.

[8] S. Kim and W. Sung. A Floating-point to Fixed-point Assembly program Translator for the TMS 320C25. *IEEE Trans. Circuits and Systems*, November 1994.

[9] S. Kim and W. Sung. Fixed-Point Error Analysis and Word Length Optimization of 8x8 IDCT Architectures. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(8):935–940, December 1998.

[10] K. Kum, J. Kang, and W. Sung. A Floating-Point to Integer C Converter with Shift Reduction for Fixed-Point Digital Signal Processors. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing ICASSP'99*, pages 2163–2166, 1999.

[11] 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.

[12] 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.

[13] D. Menard and O. Sentieys. A methodology for evaluating the precision of fixed-point systems. In *International Conference on Acoustics, Speech and Signal Processing 2002 (ICASSP 2002)* , Orlando, May 2002.

[14] D. Menard and O. Sentieys. Automatic Evaluation of the Accuracy of Fixed-point Algorithms. In *Design Automation and Test in Europe (DATE-02)* , Paris, March 2002.

[15] T. Parks and C. Burrus. *Digital Filter Design.* Jhon Willey and Sons Inc, 1987.

[16] M. Willems, V. Bursgens, H. Keding, and H. Meyr. System Level Fixed-Point Design Based On An Interpolative Approach. In *Design Automation Conference (DAC-97)*, 1997.

[17] M. Willems, V. Bursgens, and H. Meyr. FRIDGE: Floating-Point Programming of Fixed-Point Digital Signal Processors. In *ICSPAT'97*, 1997.

[18] M. Willems and V. Zivojnovic. DSP-Compiler: Product Quality for Control Oriented Applications? In *ICSPAT'96*, pages 752–756, Boston, October 1996.

[19] R. Wilson and al. SUIF: An Infrastructure for Research on Parallelizing and Optimizing Compilers. Technical Report CA 94305-4055, Computer Systems Laboratory, Stanford University, May 1994.