Efficient Implementation of a Rake Receiver on the TMS320C64x

D. Menard, M. Guitton, P. Quemerais, O. Sentieys

R2D2 team - IRISA/INRIA - Rennes I University
ENSSAT - 6, rue de Kerampont - F-22300 Lannion, FRANCE
eMail : *name*@enssat.fr

Abstract

The evolution of radio-communication systems to third generation based on the WCDMA technique leads to an increase of the digital signal processing complexity. Due to their flexibility, high-performance DSP processors are good candidates for the platform used in radio-communication infrastructures.

In this paper, an efficient implementation of a WCDMA rake receiver on the Texas Instrument TMS320C64x VLIW DSP is proposed. For minimizing the code execution time, the optimization techniques based on the exploitation of the data and instruction-level parallelism are presented.

1 Introduction

The complexity of digital signal processing applications in radio-communication systems is growing. For the third generation systems based on the WCDMA technique, the receiver is mainly made up of a FIR receiving filter and a rake receiver with synchronization mechanisms [4]. In order to reach the high computation performances required by this kind of applications, new DSP architectures have been proposed. These architectures allow to exploit the instruction level parallelism through VLIW (Very Long Instruction Word) structure and the data level parallelism with SWP (Sub-Word Parallelism) capabilities.

In this paper, an efficient implementation of a WCDMA rake receiver on the Texas Instrument TMS320C64x VLIW DSP [5] is proposed. The different techniques used in order to minimize the code execution time are presented. In section 2, the main features of a WCDMA receiver are presented. The C64x architecture based on VLIW structure are exposed in section 3. Then, the different techniques used for optimizing the instruction and data level parallelism are detailed in section 4 and 5.

2 WCDMA receiver

The European UMTS is based on the Wide-band Code Division Multiple Access (WCDMA) technique. The bandwidth of the transmitted signal is equal to 5MHz. The frequency of the code corresponding to the chip rate (F_{chip}) is fixed to 3.840 MHz. A QPSK modulation associated with a root raised cosine shaping filter is used. The transmitter interfaces for up and downlink are slightly different. A physical channel is defined by a variable spreading code. Each physical channel associates information data (DPDCH) and control data (DPCCH). The channel bits are split to the *In-phase* and the *In-quadrature* branches, then spreaded with an orthogonal variable spreading factor code (OVSF), and then scrambled by a specific spreading sequence (Kasami codes). On the downlink, all users share the same scrambling sequence so the OVSF codes are used to separate the different channels because the orthogonality properties lead to a null intercorrelation when the codes are synchronized.

The receiving filter and the rake receiver are the most critical elements in the implementation of the receiver. This filter is made up of two real FIR filters processing the in-phase and in-quadrature input over-sampled signals. Thus the complexity of the filter implemented with a 64-tap FIR filter is closed to 2 $GMAC/s$, for an over-sampling factor of 4. Given the complexity of this filter, only hardware implementations lead to efficient solutions.

The complex received signal s_e is made up of different delayed copies of the transmitted signal s_m and of an additive noise $w(n)$. If L' delayed replicas of the received signal for a number M of users are considered, the value of the signal $s(n)$ at the output of the FIR filter is

$$s_e(n) = \sum_{m=1}^{M} \sum_{l=1}^{L'} s_m(n - \tau_l) + w(n) \qquad (1)$$

The concept of the rake receiver is based on the combination of the different multipath components in

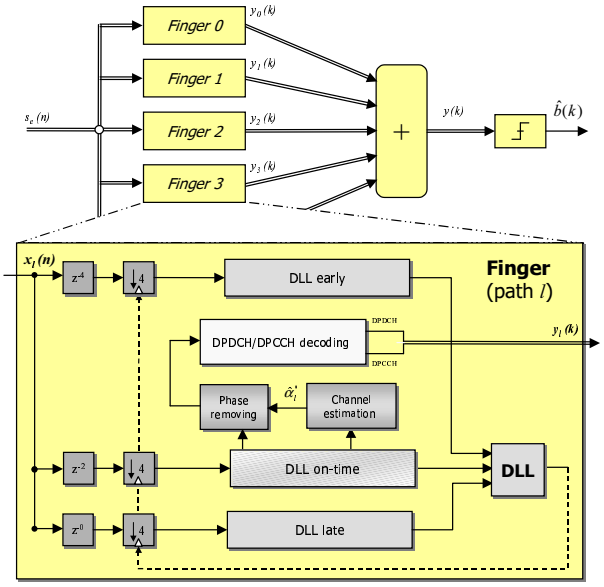

Figure 1: Principle of the rake receiver

order to improve the quality of the decision on the symbol. Each multipath signal is processed by a finger which correlates the received signal by a spreading code aligned with the delay τ_l of the multipath signal. The multipath components can be considered as uncorrelated when the delay exceeds a chip period. Demodulation results in a weighted decision at the outputs of the correlators. Using the maximum likelihood criteria the symbol is estimated from the $y(k)$ signal as

$$y(k) = \sum_{l=1}^{L} y_l(k) = \sum_{l=1}^{L} \alpha_l^*(k) r_l(k) \qquad (2)$$

The structure of the rake receiver and the different fingers is detailed in figure 1. The signal $y(k)$ corresponds to the combination of the different finger outputs $y_l(k)$. In order to combine the results of the different fingers, the complex amplitude α_l of the l^{th} path must be estimated and removed. The symbols are decoded by multiplying the received signal with a synchronized version of the code generated in the receiver. The synchronization of the code and the received signal is realized with a Delay-Locked Loop (DLL). The rake fingers must be reallocated when delays change of more than a chip delay, but small changes are processed by this code tracking loop.

For each finger, the symbols (DPDCH/DPCCH) are estimated with the structure presented in figure 2. Thanks to the complex multiplication of the received signal by the conjugate of the Kasami code the unscrambling operation is performed. Then, the phase distortion resulting from the transmission channel needs to be removed. At last the despreading operation from OVSF code transforms the wide band received signal into a narrow band signal.

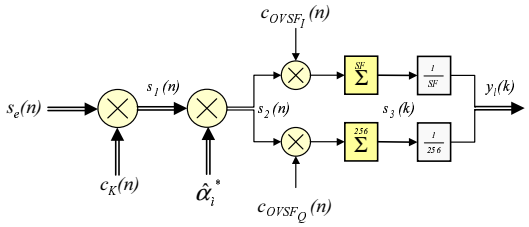

Figure 2: Symbol estimation subsystem (DPDCH-DPCCH decoding)

3 High-performance DSP

For increasing significantly the parallel processing capabilities of DSP processors, DSP based on a VLIW structure have been proposed. The C64x device is a 32-bit fixed-point processor clocked from 300 to 720 MHz, computing eight elementary instructions per cycle in eight independent functional units distributing amount two clusters. At each cycle a wide instruction packet made up of several elementary instructions is executed. This processor allows to reach according to the clock frequency a pick performance between 2400 and 5760 MIPS. A register file of 32 general-purpose registers (32 bits) is associated with each cluster. The eight functional units provide six arithmetic logic units (32/40 bits) and two 16-bit multipliers for a 32-bit result. The processor uses a load-store architecture and provides internal program and data memory.

To decrease the code execution time, data-level parallelism can be exploited with the processor SWP (Sub-Word Parallelism) capabilities. An operator of word-length N is split to execute k operations in parallel on sub-words of word-length N/k [1]. The C64x SWP capabilities for multiplication, addition and shift operations are presented in table 1. Thus, this processor can manipulate a wide diversity of data types (8, 16, 32, 40, 64 bits). The exploitation of this kind of technique requires to order properly the data in memory. Indeed, several data must be grouped into a single 32-bit memory word. The C64x instruction-set provides special instructions for grouping together ($PACK$) several data inside a 32-bit register or for extracting ($UNPACK$) a data from a 32-bit register.

Number of operations	Operand word-length (bits)		
	Multiplication	Addition	Shift
1	$16 \times 16 \Rightarrow 32$	32	32
1		40	
1	$16 \times 32 \Rightarrow 32/64$		
2	$16 \times 16 \Rightarrow 32$	16	16
4	$8 \times 8 \Rightarrow 16$	8	

Table 1: C64x SWP capacities

These data manipulation instructions for SWP operation increase the code execution time. Thus, the SWP capabilities exploitation leads to an execution time reduction if these data manipulation instructions are limited.

4 Instruction-level parallelism

Processors based on a VLIW structure are made up of several functional units able to work in parallel. Thus, these processors can execute at each cycle, in parallel, a maximal number of elementary instructions equal to the number of functional units. For improving the code efficiency, the number of elementary instructions executed per cycle (IPC) must be maximized. Nevertheless, in classical applications, the data dependencies limit the opportunities to execute in parallel several elementary instructions. Different techniques corresponding to loop unrolling [6] or software pipelining [2] are applied to increase the operation parallelism inside the loop. Thus, operations stemming from different loop iterations can be executed in parallel. Nevertheless, these techniques increase the code size. Compared to loop unrolling, the software pipelining allows to limit this code increasing and to maximize the ressource usage rate inside the loop kernel.

N_f	Average IPC	
	Sys_{symb}	Sys_{synch}
1	5.41	6.37
2	5.75	5.87
4	5.37	5.50

Sys_{symb} : symbol estimation subsystem
Sys_{synch} : synchronization subsystem

Table 2: Average IPC for the loop kernel

The benefit of this optimization technique in the case of the WCDMA receiver implementation in the TMS320C64x has been evaluated. The quality of the code generated by the T.I. compiler has been quantified for the different levels of optimization proposed by the compiler. For this application, the IPC is equal to 1.35 if no compilation optimization technique is used. The software pipelining technique leads to an IPC around 5.3 and allows to reduce the code execution time of a factor closed to 4.

The average IPC in the loop kernels has been studied for the rake receiver symbol estimation and the synchronization part and the results are presented in table 2. To test different configurations, experiments have been achieved with different levels of finger grouping. Let N_f be the parameter corresponding to the number of finger processed together. In this case, the C code of the N_f fingers are grouped together inside a same loop. Consequently, when this parameter N_f increases, the length of the C code describing the loop kernel raises. The results, given in table 2, underline that the software pipelining optimization technique allows to obtain a relatively high average IPC. Indeed, the loop kernel average IPC is around 5.7 even if only one finger is processed per loop.

The increase of the parameter N_f can lead to a decrease of the IPC inside the loop kernel. Indeed, the number of registers can be too limited for storing all the intermediate variables. In this case, the data live range must be decreased to free some registers.

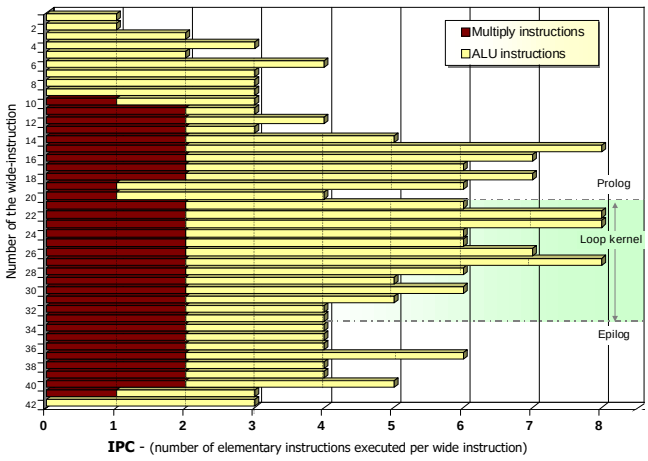

Figure 3: IPC for the loop kernel of the synchronisation subsystem

To study deeply this optimization technique quality, each wide-instruction packet is analyzed. More particularly, for each wide-instruction packet, the us-

age rate of the different kind of functional unit (multiplier, ALU) is evaluated. The analysis is presented in figure 3 for one DLL branch of the synchronization part and in figure 4 for the estimation symbol part. For these bar graphs, each horizontal bar represents a wide instruction packet. The Y-axis defines the number of this wide instruction packet and the X-axis represents the number of elementary instructions contained in a wide instruction packet. The elementary instructions controlling the multiplier or the ALU units are distinguished.

The results underline that each loop kernel wide instruction packet is composed of two elementary multiply instructions. Thus, the two multipliers are permanently used during the kernel execution. Indeed, the processing achieved in the different parts of a finger is based on complex multiplications and requires to execute a great number of real multiplications. Each complex multiplication requires to execute four real multiplications, one addition and one subtraction.

This in-depth analysis allows to decide if the code can still be optimized. Given that the multipliers are always used, the generated code can not be easily improved. In our case, the implementation improvement is not limited by the development tools but by the processor ressources for the computation.

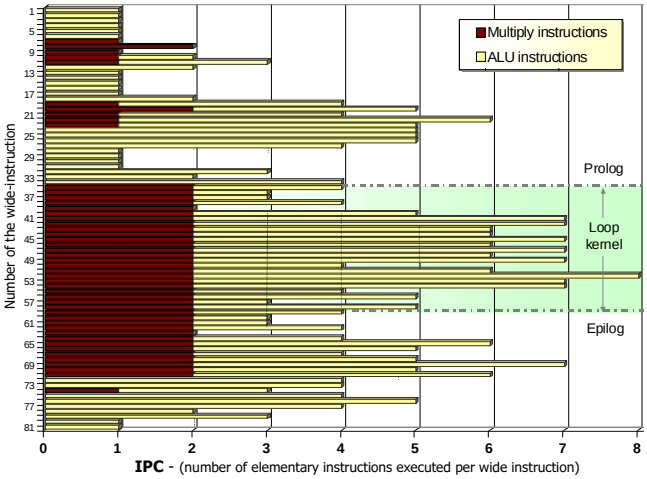

Figure 4: IPC for the loop kernel of the symbol estimation subsystem

5 Data-level parallelism

This high-performance DSP allows to manipulate a wide diversity of data types. Thus, opportunities for execution time reduction are offered if a computation accuracy diminution is allowed. Indeed, the diminution of the number of bit used for coding fixed-point data leads to an increase of the quantization noise power. For illustrating these concepts, the complex correlator used in the rake receiver has been implemented on the C64x with different data types. For each solution, the execution time and the computation accuracy are evaluated. This last parameter is determined with the Signal To Quantification Noise Ratio (SQNR) metric. The results are presented in table 3. The execution time is normalized with a classical solution (multipliers : $16 \times 16 \Rightarrow 32$ bits, adders : $32 + 32 \Rightarrow 32$ bits)

Solution	T_{exec}	SQNR (dB)	Operand word-length (bits) Multiplication	Addition
1	0.6	51	$8 \times 8 \Rightarrow 16$	$16 + 16 \Rightarrow 16$
2	1	89	$16 \times 16 \Rightarrow 32$	$32 + 32 \Rightarrow 32$
3	1.55	151	$32 \times 16 \Rightarrow 32$	$32 + 32 \Rightarrow 32$
4	2.1	170	$32 \times 16 \Rightarrow 64$	$64 + 64 \Rightarrow 64$

Table 3: Complex correlator results (T_{exec} is the normalized execution time)

For exploiting these capabilities offered by the DSP, it is necessary to determine the optimal fixed-point specification which allows to minimize the code execution time and to lead to a sufficient computation accuracy. A methodology for determining automatically the data word-length under accuracy constraint has been proposed in [3]. First, the application data dynamic range is evaluated. The dynamic range of a data can be computed from its statistical parameters which are obtained with a floating-point simulation. This approach allows to estimate accurately the dynamic range with the help of the signal characteristics but overflow can occur for signal with different statistical properties. Thus, the alternative based on an analytical approach is used. The expression of the different data dynamic range is computed from the dynamic range of the inputs.

The second methodology step corresponds to determination of the binary point position. The aim 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 position of the adder inputs.

The aim of the third step is to define the type (word-length) of each data to obtain a complete fixed-point format for each data. This stage must allow to explore the diversity of the data types available in new high-performance DSP like the TMS320C64x. Thus, the data word-lengths are optimized to reduce the application execution time as long as the accuracy constraint is fulfilled. This module selects the instructions which will respect the global accuracy constraint and minimize the code execution time.

When the fixed-point specification is defined, the C source code is modified to include the different data types. Moreover, intrinsic functions are used to express the data parallelism and to exploit the processor SWP capabilities. Thus, the parallelization of the data must be achieved by the user.

The fixed-point specification of the WCDMA receiver has been obtained with this methodology. The input data word-length corresponding to the receiving filter output was fixed to 8 bits. The data word-length for the symbol estimation part of the rake receiver are summarized in table 4.

Data	s_e	s_1	s_2	s_3	y
Word-length (bits)	8	16	16	16	16

Table 4: Data word-length for the symbol estimation subsystem of the rake receiver (figure 2)

The execution time of the code obtained with a classical approach and with our methodology are analyzed. Compared to our method, in the classical approach, the processor SWP capabilities are not used. For comparing the two approaches, the benefit due to the use of the processor SWP capabilities is evaluated. The computed metric corresponds to the ratio between the execution time of the code with SWP operations and the execution of the code without SWP operation. Different experiments have been achieved on the symbol estimation and the synchronization subsystems for several values of the parameter N_f defined in the previous section. The results, presented in table 5, underline the benefit of the SWP operations. Our approach allows to reduce the code execution time of a factor between two and four.

N_f	**SWP improvement factor**	
	Sys_{symb}	Sys_{synch}
1	2.83	1.91
2	2.79	2.79
4	3.51	3.18

Sys_{symb} : symbol estimation subsystem
Sys_{synch} : synchronization subsystem

Table 5: SWP improvement factor

6 Conclusion

In this paper, an efficient implementation of a WCDMA rake receiver on a VLIW DSP has been proposed. For minimizing the code execution time, the optimization techniques based on the data and instruction level parallelism exploitation are used. Compared to an unoptimized implementation, this approach allows to reduce the code execution time by a factor close to 16. For the real-time implementation of a five fingers rake receiver, the processor usage rate is equal to 18%. The execution time for computing a sample for the symbol estimation subsystem is equal to 5.45 cycles. Our future works are based on the integration of an FPGA accelerator to further reduce the system execution time.

References

[1] J. Fridman. Sub-Word Parallelism in Digital Signal Processing. *IEEE Signal Processing Magazine*, 17(2):27–35, March 2000.

[2] M. Lam. Software pipelining : an effective schedulin technique for VLIW machines. In *Proc. SIGPLAN Conf. Prog. Lang. Design and Implement.*, Atlanta, 1988.

[3] D. Menard, D. Chillet, F. Charot, and O. Sentieys. Automatic Floating-point to Fixed-point Conversion for DSP Code Generation. In *CASES 2002*, Grenoble, October 2002.

[4] T. Ojanperä and R. Prasad. *WCDMA : Towards IP mobility and mobile internet.* Artech House Universal Personal Communications Series, 2000.

[5] Texas Instruments. *TMS320C64x Technical Overview.* Texas Instruments, February 2001.

[6] Texas Instruments. *TMS320C6000 Programmer's guide.* Texas Instruments, August 2002.