Automatic exploration of VLIW processor architectures from a designer's experience based specification

M. AUGUIN, F. BOERI, C. CARRIERE

Lab. Informatique Signaux Systèmes (I3S), 41 Bd Napoléon III
06041 Nice cedex - France. e-mail : auguin@mimosa.unice.fr

Abstract :

This paper presents a new synthesis approach for dedicated systems. The aim of our synthesis scheme is to achieve an automatic exploration of VLIW processor architectures from a pure C description of the input system. The innovation consists in the fact that unit allocation must manage the fact that a function may be realized either by dedicated functional units or by a set of lower-level efficiently controlled functional units. For example, execution of a square root function can be accomplished by two ways: either by a dedicated functional unit or by an oriented software implementation of Newton's iterations. The aim is to find the best global trade-off between all the candidate architectures. In order to illustrate this synthesis scheme, we give an example issued from a sonar application.

1. Introduction

Numerous applications of telecommunication, signal and image processing require high speed computing performances with severe embedding constraints. Improvements of both hardware and software technologies induce moving boundaries between hardware and software parts of a system. Moreover, each system has a life cycle :

- At beginning, a prototype which makes use mainly of software components is realized.
- After, successive realizations are produced with improved performances and higher embedding capabilities. At each step, software fonctionnalities are replaced either by more efficient ones (algorithmic or processing units) or by hardware units.

The ideal design approach consists in deriving or synthesizing successive realizations from the same specification. Thus, different explorations of an hardware/software design space defined by specifications and constraints are needed to exhibit successive solutions.

Recent works propose codesign or cosynthesis methodologies. For example, approaches in [10],[8] deal with a C style specification augmented with timing constraints, task concept and communication facilities. In these approaches partitioning the specification is based on cost functions that estimate software, hardware and communication characteristics. As mentioned in [19], performance prediction is extremely difficult and may lead to sub-optimal solutions. An object oriented approach is used in [20]. Performance measuring is performed through a complete compilation of the software part and a complete synthesis of the hardware part.

In contrast, cosimulation frameworks [4],[20] are able to analyze decompositions of specifications but important portions of respective simulation models are manually deduced. Nevertheless, an important aspect of analysis based approaches is that they can reap benefit from designer's experience to produce efficient realizations.

We propose a synthesis tool which is able to explore several solutions, corresponding to different decompositions according to the designer's experience. This approach consists in a trade-off between automatic synthesis methods based on characteristics estimation, and design methods.

The aim of advanced compilation techniques for VLIW (microcoded), superscalar and pipelined processors is to optimize utilization of hardware resources according to the fine grain parallelism of the target application [13],[9],[18],[15]. High performances on fine grain architectures are achieved with these microcode compaction techniques. Thus the relative efficiency of a hardware-software solution depends on i) the efficiency of the hardware part and, ii) the amount and speed of communications. On general purpose processors, parallelism exhibited by microcode compaction techniques is generally restrained by memory throughput and access time. Cache memories are used to reduce the average access time. Consequently, to achieve higher performances than software solutions, the synthesis of hardware modules must pay particular attention to the maximum use of resources. Advanced code generation techniques can be used in synthesis methods to

0-8186-6315-4/94 $04.00

improve efficiency of hardware units [3]. The basic concept of simultaneous generation of instruction set and processor synthesis for application specific design is a variation of microcode compaction [12].

The rest of the paper is organized as follows. Section 2 describes methodology and principles involved in our synthesis scheme. Next, more details on our synthesis methods are given. In section 4 and 5, the test application and synthesis results are presented. Section 6 draws some conclusions and futur works.

2. Overview of our firmware/hardware cosynthesis method

Generally, functional descriptions of telecommunication, signal and image processing applications consist in a program, often written in C. Considering this specification for synthesis avoid to translate it into a formalism dedicated to cosynthesis. The algorithmic expression of the application includes usually calls to specific functions (e.g. FFT, correlation, FIR filter) which consume the major part of the total number of numerical operations. Increasing their performances with respect to RISC or DSP processors requires specific functional units (FUs) which allow a more efficient chaining of operations. Therefore, according to the target application, FUs must be tailored in the hardware part. Consequently, we propose that in the initial specification, the realization (firmware or hardware) of consuming operations, identified by the designer according to his own experience, is not fixed. Evaluations of different firmware/hardware solutions are performed by the synthesis process of the specification (Fig. 1 •).

External functions can be declared in the initial algorithmic system model. Firmware/hardware implementations of these functions are defined in a library. Then, different specifications are build up. They make use of the different ways of implementation of external functions. A synthesis procedure of microcoded architectures is applied on these clever specifications (candidate descriptions). A set of implementation models satisfying designer's constraints are produced. Each implementation model consists in a VHDL structural description, an optimized microcode and a performance report. In the next section, details on system modeling are given.

2.1 Specification model

Generally, a functional model of applications from telecommunication, signal and image processing domains, consists in a program (often C). This program derives directly from the algorithmic expression of operations to execute on the system's inputs. It is of primary importance for verification that the specification model for synthesis remains executable.

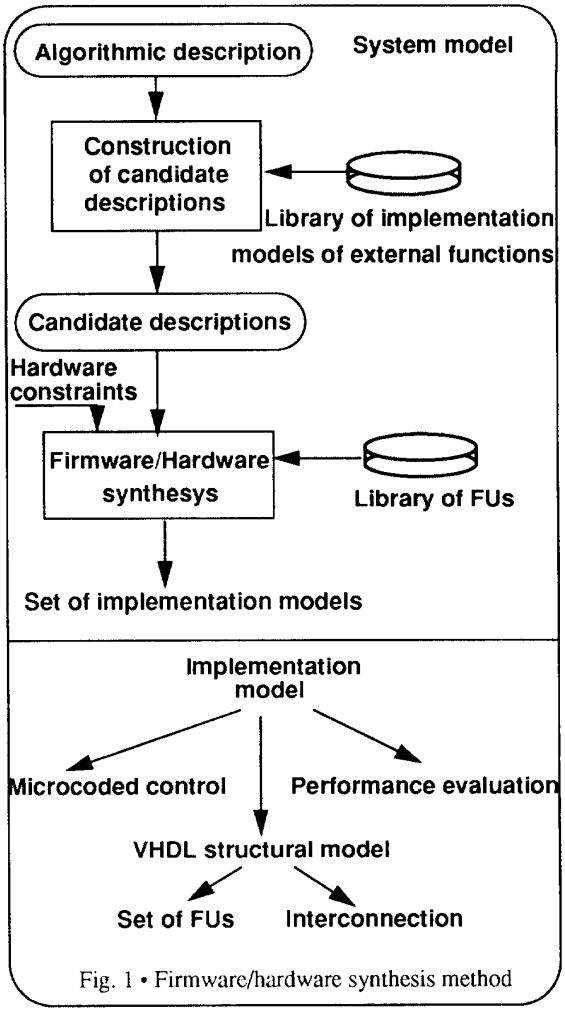

Fig. 1 • Firmware/hardware synthesis method

In Fig. 2 • is given an example of a bit reversal operation in a FFT program. The bit reversal function is external to the main program. Descriptions of algorithmic or hardware realizations of this function are placed in the library of implementation models. In the hardware model section, the name #bit_reversal refers to at least one entry in the library of FUs (Fig. 1 •). In Fig. 2 • only one algorithmic implementation is illustrated.

For each external function, several algorithmic models, corresponding to different software realizations, can be placed in the library of implementation models. An algorithmic model can refer itself to a functional unit (algorithmic or hardware model) of the library of implementation models.

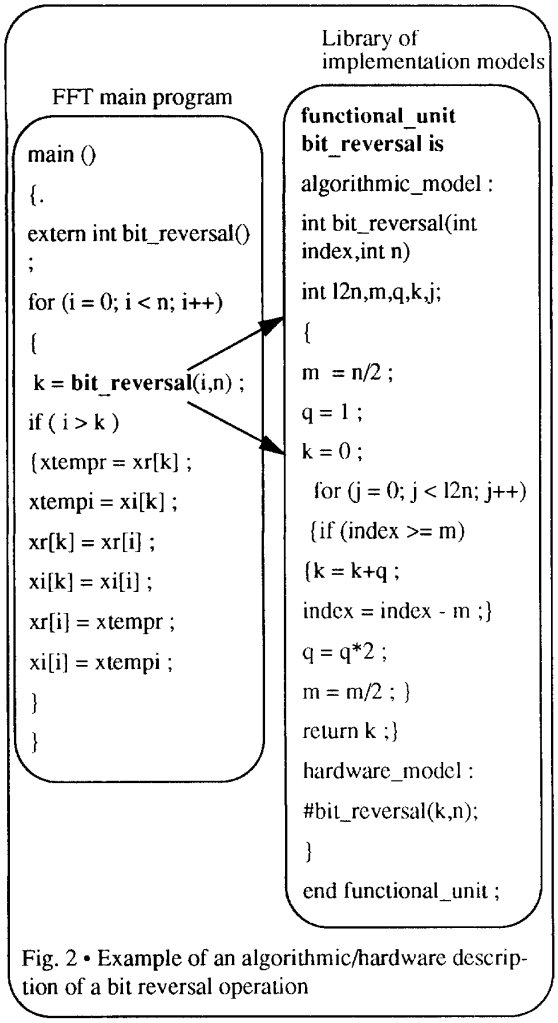

Fig. 2 • Example of an algorithmic/hardware description of a bit reversal operation

Generation of candidate descriptions for synthesis is accomplished by selecting different realizations of external functions of the main program.

This algorithmic specification model allows an easy generation of at least one executable software code for functional verification on real inputs.

2.2 Generic architecture

The target of our synthesis method is a VLIW architecture composed of three major parts:

- a microcoded control part,
- a data memory system,
- and a processing module with specific synthesized hardware units (FUs).

Guidelines for a microcoded style architecture suited for signal and video applications given in [6] depict a synthesis method of microcoded DSP processors from a Silage[11] description of the application. Unfortunately, data-path composition and memory organization are performed manually.

Multi-dimensional signal processing requires the memorization of large arrays of variables. Thus, our generic architecture contains a data memory and it is able to perform address computations.

Processing modules are constructed with FUs that are described in the library of FUs. FUs can be multi-functions and pipelined. Pipelining FUs may be of prime importance in order to reduce the cycle time of the architecture.

The generic architecture is of VLIW type. Scheduling and allocation phases of the synthesis process are issued from microcode compaction techniques for VLIW processors. For example, loops which represent regular computations on structured arrays are vectorized, i.e. several executions of the body loop are overlapped. The objective is to achieve an utilization rate of 100% of at least one resource of the architecture. A software pipelining technique [7] is used for implementing vector processing. But unfortunately vector processing increases the number of simultaneous live variables and thus the size of registers in the processing module.

The synthesis of VLIW microcoded processors is accomplished by the CAPSYS system [14],[2].

3. Overview of CAPSYS

CAPSYS inputs an algorithmic description of a target application, a set of physical constraints and an optional predefined architecture. It provides:

- optimized dedicated VLIW architectures,
- the associated object microcodes of the application
- performance reports.

An optional CAPSYS-type predefined architecture gives to the designer the possibility of performing incremental synthesis. If the predefined architecture is incomplete, with respect to the requirements of the target application, then CAPSYS adds FUs that are required to cover operations of the input program. For example, technology improvements of an architecture or the adaptation of an existing design to another application are possible [17].

The first phase of CAPSYS consists in compiling the input program of the application. The program is decomposed into basic blocks [1]. Each instruction of a basic block is

represented with a dependence graph. Each node of the dependence graph may be either a microtask deduced from the arithmetic or logical operations of the input program or a microtask required for memory management. The set of graphs of the whole program constitutes a control and data flow graph (CDFG). The synthesis process of CAPSYS may be divided into six steps (Fig. 3 •).

1. Labeling CDFG with the predefined architecture: If a predefined architecture is declared, microtasks which have a realization in this architecture are labeled only with declared FUs.

2. Labeling with FUs: For each microtask of the CDFG, CAPSYS looks for FUs in the library of FUs which are able to realize it. The microtask is then labeled with a list of FUs.

3. Allocation and scheduling: this step produces models of architectures. The scheduling algorithm is based on the list-scheduling or scalar code sections of the input program and software pipelining or vector code sections. When all microtasks of CDFG are scheduled, we get architectural models, each one composed of a list of FUs and the microcode associated with these FUs for executing the target application. The aim of the next steps is the synthesis of interconnection between FUs.

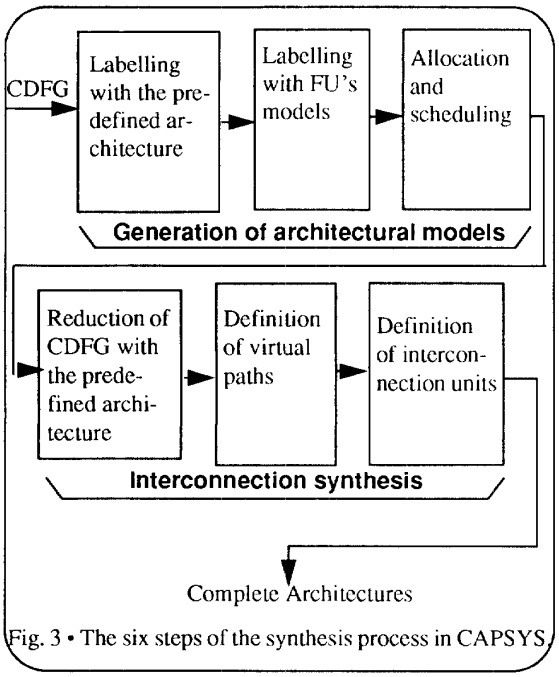

Fig. 3 • The six steps of the synthesis process in CAPSYS

4. Edges of a CDFG correspond to data transfers between FUs. If a predefined architecture is provided, all edges with data transfers covered by the predefined architecture are removed in the fourth step.

5. FUs produce and consume data on input and output ports. Edges with different production and consumption times may be gathered in virtual paths. The aim of the fifth step is to minimize the number of virtual paths. Four methods are coded in CAPSYS [5]. The first two ones operate directly on edges of CDFG. They are based on an edge coloring method and a greedy algorithm. The last methods perform a preprocessing on edges in order to reduce the number of multiplexers and physical links. For example, edges describing data transfers to the same input port of a FU are grouped in logical connections. Then, either a greedy algorithm or a maximum compatible method is applied in order to produce virtual paths.

6. In the last step, register files, multiplexers and logical links are determined. It is important to note that registers have a double purpose. They assume the memorization of data between production and consumption by FUs and between load (respectively store) operations in the data memory and consumption (production) by FUs. Furthermore, operators and registers are used not only for operations on variables but also, for address computations.

In the next section the translation of a specification into firmware/hardware is introduced.

3.1 Translation of a specification

When the input program involves external function, a set of intermediate programs is built by a recursive procedure. The input program is the first intermediate program. When an intermediate program contains an external function (EFi), intermediate programs are generated. In the first one, no change happens since external functions have a hardware realization. The others intermediate programs are obtained by replacing each external function EFi by either a software implementation or a hardware realization. This procedure is applied for each different external function that appears in the input program and for each intermediate program.

Notice that no use of external function reference is allowed in software implementation models.

Then each intermediate program is compiled into an internal representation (CDFG) which defines the abstract level (generic architecture) from which synthesized systems are produced. Nodes of the CDFG correspond to microtasks and edges describe control and data dependencies between microtasks. Fig. 4 • depicts a graph associated to an instruction of form "x=y + z;". Addition operates on values

issued from the data memory and produces a result which is in turn stored in the data memory. Synthesis of the graph of Fig. 4 • requires that for each microtask, there exists at least one FU in the library of FUs, that is able to realize it.

Microtasks of Fig. 4 • are primitive ones corresponding to basic operations of the input program. This type of graph is obtained when input C programs do not include hardware models of FUs.

When an external function of the C input program is implemented with an hardware model (Fig. 2 •), a generic operator is used in the internal representation (Fig. 5 •). The generic operator "op_generic" is labelled with the name of the function ("bit_reversal" in example of Fig. 5 •). At least one FU in the library of functional unit must have a method (or function) with the same name.

3.2 Scheduling and FUs Allocation

When the translation of the input program is ended, CAPSYS executes the scheduling and FU allocation phases in the same time. It begins to annotate nodes of CDFG by FUs that can run the corresponding microtask.

With the annotated CDFG, a tree of candidate architectures that verify the designer's constraints like cost, area, consumption is built with the following recursive procedure.

At the beginning, the initial list (LFUs) of FUs that compose the architecture is empty.

Then for each microtask Mi that may be scheduled, different cases occur:

- There exists one free FU∈ LFUs to realize Mi.

  Mi is scheduled.

- There is one FU∈ LFUs to realize Mi but this FU is busy.

  If the designer's constraints are not overstepped, the method adds a resource to LFUs. In the other case, Mi will be scheduled when one FU∈ LFUs will become free.

- There is no FU∈ LFUs to realize Mi.

  If the designer's constraints are not overstepped, the method adds a resource ∈ LFUs. In the opposite case, no solution are provided.

When a resource is added to LFUs, as much candidate architectures are created with the recursive procedure as there are FUs that are able to realize the considering microtask.

The scheduling and allocation phase of CAPSYS produces architectural models with instantiations of FUs performing microtasks of the internal representation. This specification space exploration technique is illustrated in the next section with an example of a signal processing unit of a simplified sonar system.

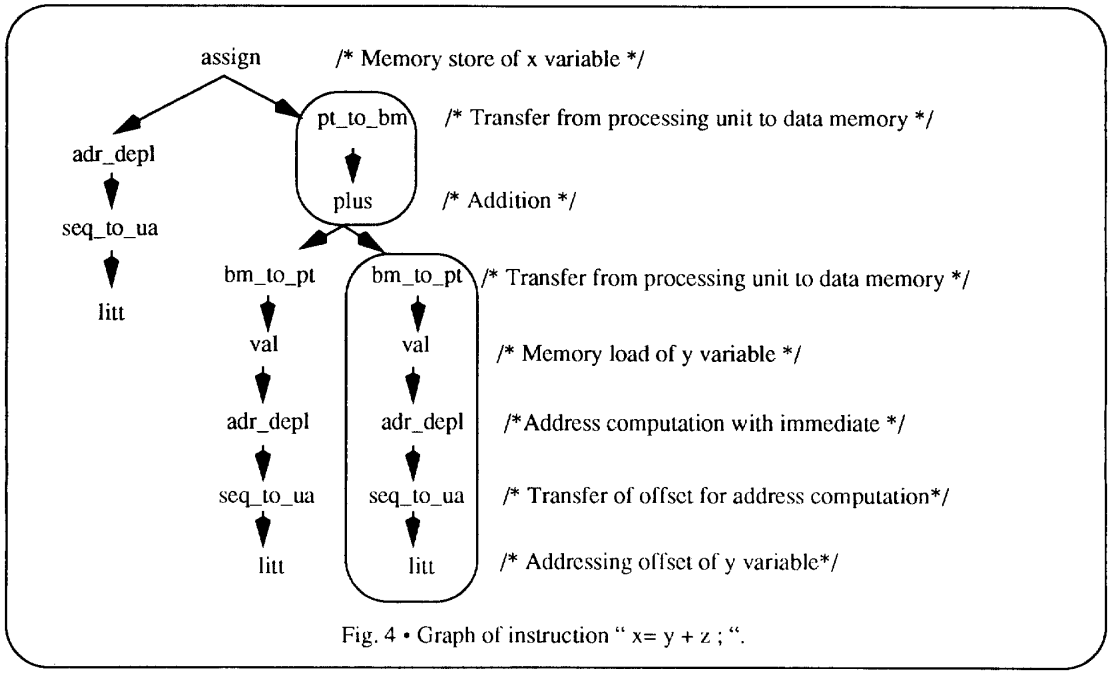

Fig. 4 • Graph of instruction " x= y + z ; ".

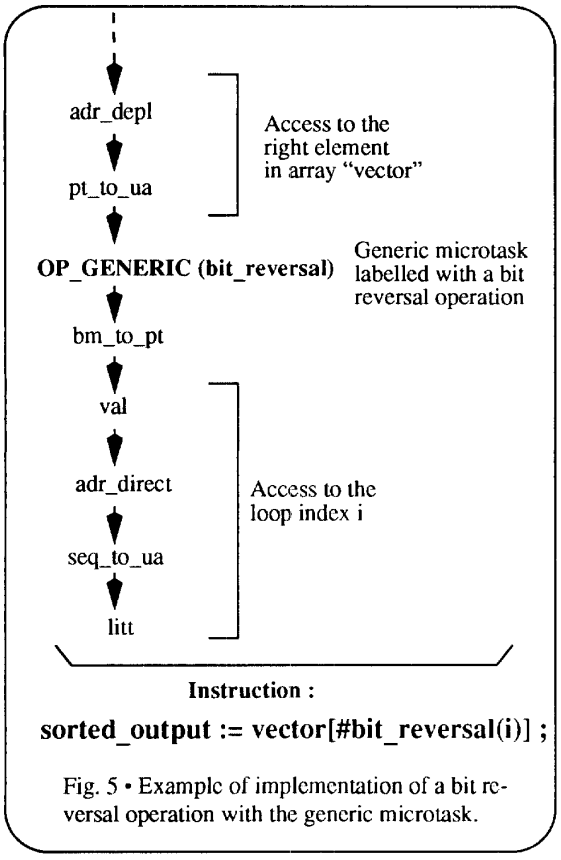

Fig. 5 • Example of implementation of a bit reversal operation with the generic microtask.

4. Example : a signal processing unit of a sonar system

In a simplified real time sonar application (Fig. 6 •), 64 samples are delivered every 5 ms to a signal processing unit (SPU). The SPU has two functions. First, it performs a 256 point-FFT on input samples.

First, it performs a 256 point-FFT on input samples. The second step consists in a signal detection process with a noise power estimation and an adaptative thresholding depending on the signal/noise ratio (SNR). SPU transmits bin numbers, bin values, and SNR of the detected signal to a data processing unit (DPU). The objective is the realization of an optimized structure of SPU.

We assume that signal samples are pushed into an input FIFO. SPU places values addressed to DPU into an output FIFO. Operations are performed in a 24 bits integer format. FUs used to synthesize (ALU, multipliers, file registers, sequencer for example) SPUs are developed with an ASIC VHDL synthesis tool and a 1.0 micron cmos library.

5. Results

A classical Cooley-Tuckey radix-2 FFT algorithm is implemented in the initial C specification program of the sonar application. This specification is executable on a workstation in order to verify its behavior on real values. In this program, input and output FIFOs are not described, samples are read from a data file and bin values are plotted on screen. These I/O operations are replaced in the C spec-

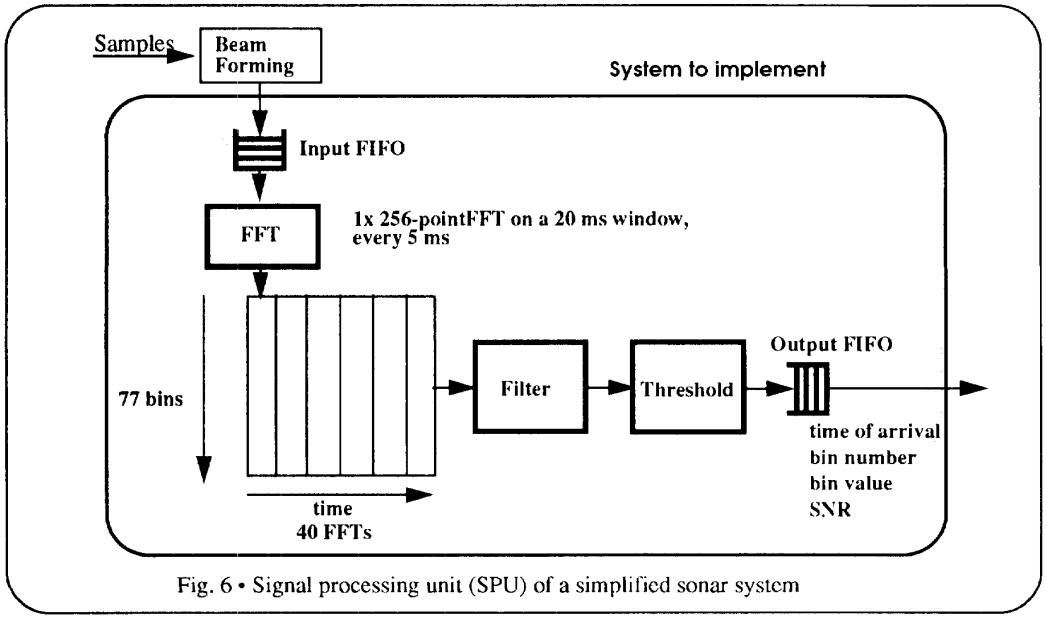

Fig. 6 • Signal processing unit (SPU) of a simplified sonar system

ification program by external function calls that perform pull and push operations respectively on input and output hardware FIFOs. The bit reversal operation is implemented with an algorithmic procedure (Fig. 2 •). This initial specification is then synthesized by CAPSYS. Without vectorisation of loops we get an architecture with an execution time of 80ms that is too large with respect to timing constraints (5 ms).

We dispose of three directions to achieve higher performances: i) utilization of specific hardware units, ii) vectorisation of loops, and iii) algorithmic modifications.

Since an important gap of performances is required, an hardware bit_reversal module is introduced in the library of implementation models, the library of FUs contains a model of a three stages pipelined multiplier (pMul), and the internal loop of FFT is vectorized. We get system S1 in Table 1.

An improvement in the vectorisation consists in a separate execution of the two first stages of FFT in order to consider particular values of the twiddle factors. This leads to system S2 which verifies the timing constraint. The execution time of 4.7 ms includes pull operations of samples from the input FIFO and push of detected bins into the output FIFO.

Each stage of the radix-2 FFT computes N/2 butterfly operators for a N points Fourier transform. Instructions of the internal loop of the FFT are replaced by an external function call to a butterfly hardware module. We get then system S3 which has about the same performance than S2 but with twice the number of gates. This result illustrates the efficiency of software pipelining: operations on ALUs and pipelined multipliers are very well interleaved and the fine grain parallelism is especially limited by the data memory throughput. Therefore, a hardware operator which results from a particular arrangement of adders and multipliers, cannot improve performances significantly.

In a vectorized radix-2 FFT, sizes of vectors are halved at each stage. Thus efficiency of software pipelining is reduced when sizes of vectors are too short. Consequently we consider an FFT with constant topology [16]which has vectors of size N/2 during all the processing flow. The resulting system S4 is composed of less FUs than previous ones and it satisfies the timing constraint with a lower clock cycle.

As mentioned above, the data memory limits performances. Thus we modify the input specification by declaring external functions implemented as hardware FUs:

- Two local memories are introduced in the processing unit in order to increase the global throughput of data.
- A read only memory for twiddle factors of the FFT is also placed in the processing unit.

With these features, we get system S5 which has a performance of 2.4 ms. Consequently processor S5 is able to realizes operations associated with two beams.

Realization of the FFT (without the bit reversal operation) is performed in 4*Nlog(N) cycles on S5. Notice that on a TMS320C40, a complex radix-2 FFT is executed in 5*Nlog(N) cycles. However, there are more operations required in the constant topology FFT than in the Cooley-Tuckey implementation. But, if two local memories are added to S5, the constant topology FFT is executed in 2*Nlog(N) cycles.

6. Conclusion

This paper illustrates the importance of a synthesis/evaluation tool of different specifications deduced from an initial C program description of the target application.

Our approach consists in a specification space exploration driven by various realizations of external function calls in the initial specification. These realizations are either of algorithmic type and translated into firmware or of hardware

Table 1. Synthesized sonar systems

	FUs	clock (ns)	Number of gates	Execution time
S1	3*ALUs, 2*pMuls, 1*Bit_reversal	45	28.000	10 ms
S2	3*ALUs, 1*pMul, 1*Bit_reversal	45	22.000	4.7 ms
S3	3*ALUs, 2*pMuls, 1*Bit_reversal, 1*Butterfly	45	42.000	4.9 ms
S4	1*ALUs, 1*pMuls, 1*Bit_reversal	60	19.000	4.8 ms
S5	2*ALUs, 2*pMuls, 1*Bit_reversal2*Local memories, 1*ROM factors	60	86.000	2.4 ms

type. The synthesis/evaluation tool proposes realizations of these derived specifications.

Designer's experience is of prime importance to guide the synthesis process in order to profile a system that matches constraints: we think that it would be preferable to consider a (co)synthesis tool as a program which is able to perform an efficient space and time decomposition of a specification rather than an automatic partitioning method that implements only one synthesis scheme.

In this paper, we focus on firmware/hardware target systems since scheduling techniques for VLIW type processors are very efficient. Particularly, an horizontal microcode is well suited to control parallel/pipeline processing in an hardware unit. This unit could be considered as a dedicated multi-function co-processor of a general purpose processor. We plan to extend our work to a software/firmware/hardware cosynthesis method from a C program description.

7. Bibliography

[1]AHO A.V., SETHI R., ULLMAN J.D. *Compilers, principles, techniques and tools*. Addison-Wesley, 1986.

[2]AUGUIN M., BOERI F., CARRIERE C., MENEZ G. Synthesis of dedicated SIMD processors. *Proceedings Application Specific Array Processors*. Venise, October 25-27, 1993.

[3]BRETERNITZ M, SHEN J. P. Architecture synthesis of high performance application-specific processors. *Proceedings 27th Design Automation Conf.*, pages 542-548. Orlando, june, 1990.

[4]BUCK J., HA S., LEE E.A., MESSERSCHMITT D.G. Ptolemy: a mixed paradigm simulation/prototyping platform. *Proceedings of Speech Tech*. New York, April 23-25, 1991.

[5]CARRIERE C., AUGUIN M., BOERI F., MENEZ G. A comparison study of minimization methods of unit interconnection in VLIW processors. *Proceedings EUROMICRO-92*, pages 595-602. Paris, september 14-17, 1992.

[6]CATTHOOR F. Microcoded processor architectures and synthesis methodology for real time signal processing. *Proceedings Algorithms and Parallel Architectures*, pages 403-429. Pont-a-Mousson, France, june 10-16, 1990.

[7]EISENBEIS, C. Optimization of horizontal microcode generation for loop structures. *Proceedings International Conf. on Supercomputing*. Saint Malo, july, 1988.

[8]ERNST R., HENKEL J., BENNER T. Hardware-Software Cosynthesis for Microcontrollers. *IEEE Journal Design and Test of Computers*. 64-75, december, 1993.

[9]GAO G.R. et al. A timed Petri net model for fine grain loop scheduling. *SIGPLAN'91 Conference on PLDI*. june, 1991.

[10]GUPTA R.K., DE MICHELI G. Hardware-Software Cosynthesis for Digital Systems. *IEEE Journal Design and Test of Computers*. 29-41, september, 1993.

[11]HILFINGER P.N. Silage a high level language and silicon compiler for signal processing. *IEEE Conference on Custom Integrated Circuits*, pages 213-216. Portland, may, 1985.

[12]HOLMER B.K., PANGRLE B.M. Hardware-software-codesign-using-automated-instruction-set-design. *Proceedings Int. Workshop on Hardware-Software Co-Design*. Cambridge, Mass., October 7-8, 1993.

[13]LAM M.S. Software pipelining : an effective scheduling technique for VLIW machine. *ACM SIGPLAN'88 Conference on PLDI*. Atlanta, june, 1988.

[14]MENEZ G., AUGUIN M., BOERI F., CARRIERE C. A partitioning algorithm for system level synthesis. *Proceedings ICCAD92*, pages 482-487. Santa-Clara, California, november 8-12, 1992.

[15]MOON S.M., EBCIOGLU K., AGRAWALA A.K. Selective scheduling framework for speculative operations in VLIW and superscalar processors. *IFIP Working conference on Architecture and Compilation Techniques for Fine and Medium Grain Parallelism*, pages 229-242. Orlando, january 20-22, 1993.

[16]STONE H. S. Parallel processing with the perfect shuffle. *IEEE Transactions on Computers*. C-20153-161, Feb., 1971.

[17]AUGUIN M., BOERI F., CARRIERE C., MENEZ G. Incremental synthesis of application domain specific processors. *Proceedings ICASSP-93*, pages 1425-1428. april 27-30, 1993.

[18]WANG J., EISENBEIS C. Decomposed software pipelining a new approach to exploit instruction level parallelism for loop programs. *IFIP Working conference on Architecture and Compilation Techniques for Fine and Medium Grain Parallelism*, pages 3-14. Orlando, january 20-22, 1993.

[19]WOLF W., MARTINEZ J.C. C Program Performance Estimation for Embedded Systems Architecture Sizing. *Proceedings Int. Workshop on Hardware-Software Co-Design*. Cambridge, Mass., October 7-8, 1993.

[20]WOO N.S., DUNLOP A.E., WOLF W. Codesign from cospecification. *IEEE Computer Journal*. 42-47, january, 1994.