METHODS FOR RETARGETABLE DSP CODE GENERATION

Rainer Leupers, Ralf Niemann, Peter Marwedel
University of Dortmund
Dept. of Computer Science XII
44221 Dortmund, Germany

Abstract – Efficient embedded DSP system design requires methods of hardware/software codesign. In this contribution we focus on software synthesis for partitioned system behavioral descriptions. In previous approaches, this task is performed by compiling the behavioral descriptions onto standard processors using target-specific compilers. It is argued that abandoning this restriction allows for higher degrees of freedom in design space exploration. In turn, this demands for retargetable code generation tools. We present different schemes for DSP code generation using the MSSQ microcode generator. Experiments with industrial applications revealed that retargetable DSP code generation based on structural hardware descriptions is feasible, but there exists a strong dependency between the behavioral description style and the resulting code quality. As a result, necessary features of high-quality retargetable DSP code generators are identified.[1]

1 INTRODUCTION

Embedded systems in general comprise heterogeneous components like standard DSPs, DSP cores, ASIPs, dedicated datapaths, storages and communication hardware. Mapping a system behavioral description onto a heterogeneous architecture requires partitioning the description into hardware and software components and synthesizing both for the final implementation. This is a subproblem of hardware/software codesign which has been recognized to be a key problem in electronic design automation at the system level. However, there is still no agreement in the research community upon an exact problem definition. In particular, this holds for the target architectural styles to be investigated. While synthesizing a dedicated hardware structure from behavioral specifications for various technologies is quite well understood, synthesis of software components of a partitioned system behavioral description strongly depends on the underlying programmable hardware model. Currently, most of the research on hardware/software codesign is based on the assumption that the target machine is a standard processor [1, 2] or a standard DSP core [3]. In these cases, the problem of software synthesis is reduced to the usage of available compilers for standard processors. For industrial needs, it may often turn out that using a standard processor is inadequate because of area or speed constraints. Instead, application-specific instruction set processors (ASIPs) are better suited for running software components in embedded systems. This fact was recently outlined by industrial surveys [4]. Employing ASIPs instead of off-the-shelf processors in turn requires compilers that may be easily retargeted towards different machines with a minimum amount of work for compiler re-design, since the processor structure itself becomes subject to hardware/software tradeoffs in this case. We believe that this less restrictive approach yields the highest degree of implementation freedom for system level design, and thus will gain more and more importance in the future.

While ASIP hardware design might be addressed by traditional high-level synthesis or by special approaches [5], software synthesis for ASIPs requires a new class of CAD tools: retargetable compilers. Besides system-level design, such compilers have applications for industrial in-house DSPs, for which no target-specific compiler exists, and where code generation still has to be done manually at the machine code level.

The CodeSyn compiler presented in [4] is tailored towards embedded DSP system design. However, retargeting still requires a large amount of manual preprocessing. In contrast, the retargetable compiler

[1] This work has been partially supported by ESPRIT BRA project 9138 (CHIPS)

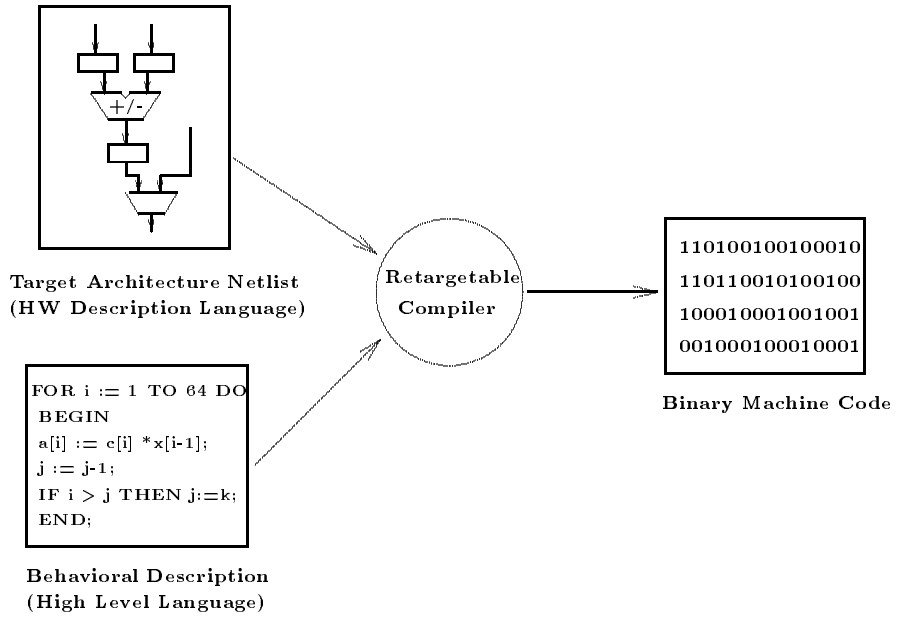

Figure 1: Retargetable compilation based on structural descriptions

MSSQ, which is part of the MIMOLA system [6], is based on pure structural hardware descriptions. Since a netlist of the target processor is usually available during the design process, retargeting the compiler to a new machine requires no or only few manual work. However, MSSQ was not intended to be a special compiler for DSP applications. Therefore, there exist various schemes for code generation when applying MSSQ to compilation of DSP algorithms. The code quality strongly depends on the style of the input behavioral description. The purpose of this paper is to outline the tradeoff between the behavioral description style and the resulting code quality, and to identify necessary improvements in code generation for DSPs or ASIPs. Nevertheless, it is shown that retargetable code generation with MSSQ for embedded systems is feasible. This is done using a real-life example. The following section gives a short overview of the functionality of MSSQ. Then, the sample DSP application and the target architecture are presented. Section 4 deals with different styles for describing the application and their impact on the resulting code quality. The paper ends with conclusions and hints for further research.

2 THE MSSQ MICROCODE COMPILER

The MSSQ microcode compiler is integrated within the MIMOLA Design System [6]. It reads a behavioral specification in a PASCAL-like language as well as a RT-level netlist of the target processor in a hardware description language. Both are given in the MIMOLA language [7]. The behavioral specification is compiled into binary microcode for this processor (fig. 1). Convenient retargetability is guaranteed by the fact, that code generation is based on a true structural hardware description, including the controller part. Changing the target architecture only requires adapting the hardware

model. Furthermore, a RT-level netlist is usually available during the design process. This is in sharp contrast to CodeSyn [4], which is based on manually specified instruction sets. Code generation in MSSQ proceeds in several separate phases:

1. Mapping of the abstract behavioral description into a RT-level program.
2. Optionally, application of user-specified high-level transformation rules.
3. Allocation of different versions for IF-statement implementation.
4. Allocation of available hardware operations and the corresponding partial instruction word settings based on a graph representation of the target structure.
5. Statement allocation within the graph structure by pattern matching.
6. Microcode compaction based on a heuristic scheduler.

Due to the final code compaction phase, MSSQ is able of exploiting potential parallelism within the hardware structure. This feature turned out to be crucial for the sample application which is presented in the following section. See [8] and [6] for more detailed descriptions of MSSQ and its integration into a complete CAD system.

3 SAMPLE APPLICATION AND TARGET STRUCTURE

This section gives a short overview of the DSP application which was compiled onto a given ASIP target architecture.

3.1 Behavioral description

The investigated DSP application consists of two identical discrete filters. The application is used in scope of digital audio signal processing. For this reason, each of both filters represents one of two stereo channels. Both parts are a combination of a FIR-filter and an IIR-filter. The outputs of the FIR-filters xl and xr represent the inputs of the IIR-filters. The filter schematic for one channel is depicted in figure 2.
The behavior of the channels can be easily explained by the transmission functions (figure 3) of both filters. The FIR-filter eliminates high frequencies and the IIR-filter boosts the lower ones. Therefore, the complete filter realizes a digital bass booster.
The application was originally described in the data-flow language SILAGE ([9]). In contrast to imperative languages where programs are represented as a sequence of assignments, a SILAGE program is realized as a set of signal definitions. Consequently, the relative ordering of statements in the source code has no effect. Concurrency is not determined through operation order, but by the information about concurrency inherent in the corresponding data-flow graph. For this reason, SILAGE as an applicative behavioral specification language is well suited to describe DSP applications.
The SILAGE description was translated into the MIMOLA language for code generation purposes.

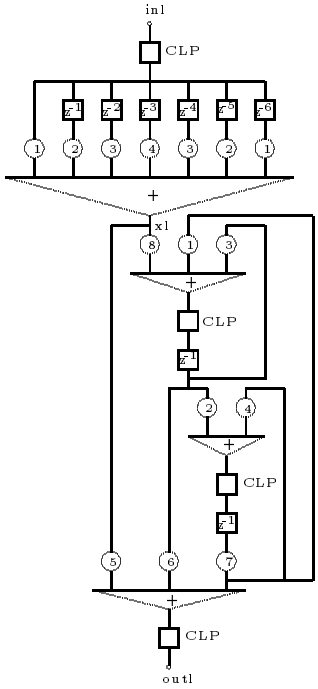

Figure 2: Filter Schematics

3.2 Target architecture

The behavioral description had to be compiled onto the target architecture shown in fig. 4. The main characteristics of this architecture are the following:

- Horizontal 42-bit microinstruction word
- Address calculation unit realizing a ring buffer
- 120 × 18 bit RAM
- Coefficient ROM
- 22/24 bit multiply-accumulate section
- Extensive data pipelining for high throughput

3.3 Hardware model

The above target architecture has been modelled in the MIMOLA language by specifying all RT-level modules with their behavior (similar to VHDL descriptions) and the module interconnect. The MIMOLA hardware model contains about 400 text lines. Since the instruction format and the instruction sequencer are implicitly part of the model, no additional information about restrictions due to encoding or sharing had to be specified manually.

4 TRADING OFF DESCRIPTION EFFORT AGAINST CODE QUALITY

In the original design of the filter application, microcode generation has been done manually. In case of extensive parallelism and data pipelining, manual code generation is quite a difficult task. In our approach, we applied the MSSQ compiler to generate microcode automatically, based on the structural hardware model. MSSQ reads the behavioral description as well as the hardware model and produces a binary program listing, immediately executable on the target machine:

```
*---------+-----------+-----------------------------------------+
*I-Address| L A B E L |  40  36  32  28  24  20  16  12   8   4   0|
*---------+-----------+---|---|---|---|---|---|---|---|---|---|---|+
#000      |L0         |  00000000010000000000x00010000000xxxxxxxxxxx|
#001      |Line0487   |  10000000100000000010001000001111xxxxx000000|
#002      |Line0490   |  00000000110010000010000xxxxxxxxxxxxxxxxxxx|
#003      |Line0492   |  00000001000000000100000xxxxxxxxxxxxxxxxxxx|
#004      |Line0493   |  00000001010000010010000xxxxxxxxxxxxxxxxxxx|
 .....................
#03C      |Line0794   |  00001111010000111010000xxxxxxxxxxxxxxxxxxx|
#03D      |Line0798   |  00001111100100001000x0000xxxxxxxxxxxxxxxxxx|
#03E      |Line0802   |  00001111111000000000x0100xxxxxxxxxxxxxxxxxx|
#03F      |Line0804   |  00000000000000000000x0000xxxxxxxxxxxxxxxxxx|
```

Since MSSQ cannot exploit DSP-specific hardware automatically (e.g. ring buffer addressing), we used three models for the behavioral description, differing in the level of abstraction. The target hardware model was retained in each of the three versions. Code could be generated in each case, but the code quality was heavily dependent on behavioral description style. The characteristics of the different versions are given in the following.

4.1 Version 1: low level

In the "low-level" version, behavior was specified as a RT-level program. Memory management and parallelisation were done manually. The behavioral description consisted of a sequence of parallel blocks, each being compactable into one microinstruction:

```
PARBEGIN
 data := DataRAM[adr];
 adr := base + ((adr + 1) "MOD" mod);
PAREND;
PARBEGIN
 m := data * coef;
 data := DataRAM[adr];
 adr := base + ((adr + 1) "MOD" mod);
 coef := 2;
 s1 := 0;
PAREND;
PARBEGIN
 data := DataRAM[adr];
 s1 := s1 + m;
 m := data * coef;
 coef := 3;
 adr := base + ((adr + 1) "MOD" mod);
PAREND;
```

Addressing was done using physical storages instead of abstract user variables. In this case, MSSQ worked more like a micro-assembler instead of a compiler. Nevertheless, important subtasks were still performed automatically, e.g. constant and operator allocation, disabling unused storages within each cycle, and binary code generation. Behavioral description at this low level required quite a large amount of work, but the achieved code quality was nearly optimal (64 microinstructions for both filter channels).

4.2 Version 2: high level

In contrast to the previous version, behavior was modelled at a very high level of abstraction. Signals were declared as abstract user variables, and memory management and parallelisation were left to MSSQ. The program was specified in a PASCAL-like manner:

```
...
VAR xrin,xlin:       ARRAY[0..6] OF Short;
    yr1,yr2,yl1,yl2: ARRAY[0..1] OF Short;
    xr,xl:           Short;
BEGIN
 xrin[0] := indata;
```

```
 xr :=
        (xrin[0] * 1)
 +      (xrin[1] * 2)
 +      (xrin[2] * 3)
 +      (xrin[3] * 4)
 +      (xrin[4] * 3)
 +      (xrin[5] * 2)
 +      (xrin[6] * 1);
...
```

Behavioral description at this high level was quite effortless, e.g. a complete FIR filter computation could be described as one single assignment (`xr := ...`). Although MSSQ emitted compact code, the resulting code size was large (197 microinstructions). This was mainly due to the fact, that no ring buffer addressing was used. Instead, all sample delay operations were specified as explicit memory data moves which are impossible to parallelise. These delay operations required about 50 percent of the total code size.

4.3 Version 3: mixed high/low level

The third version of behavioral description offers a compromise between the two previous ones. Register and memory assignment were done manually, but all other tasks were left to the compiler, including the difficult subtask of parallelisation for data pipelining. The behavioral description in this case looks very similar to the one of version 2, besides the fact that physical memory cells are used instead of abstract user variables. Using this approach, the problem of code generation for sample delay operations could be eliminated, still retaining a low description effort. The resulting code size was 98 microinstructions, i.e. only about 30 percent above the optimum.

Table 1 summarizes the results of the application study:

version	behavioral description effort	code size (instr)
1	high	64
2	low	197
3	low	98

Table 1: Results of application study

5 CONCLUSIONS

It was shown that retargetable code generation for ASIPs based on true structural target hardware descriptions is feasible. We believe that this approach is better suited for integration into a CAD environment, compared to the ones based on instruction set specification. It extends the possible range of architectural styles for hardware/software codesign, which in turn offers higher degrees of freedom for embedded system implementation.
Different behavioral description styles have been investigated for a real-life example. This application study revealed that high-quality code generation is possible with existing tools, when behavior is modelled at a low to medium level of abstraction. Moving to higher levels of abstraction, yet retaining

good code quality, will require several improvements, e.g. automatic exploitation of DSP-specific hardware features (ring buffer, repeat counter), a more global scheduling and temporary cell allocation, extended capabilities for modelling target structures, and a concept of handling real-time constraints in the compilation process. Nevertheless, using MSSQ for code generation even at the lowest level of abstraction (see version 1) saves a large amount of work compared to totally manual code generation. Furthermore, MSSQ is capable of exploiting potential parallelism within the hardware structure automatically by microcode compaction. Up to five microoperations are issued in parallel for the sample structure, obeying data dependencies.
Further research will focus on developing a retargetable code generator dedicated to DSP applications, based on the experience with MSSQ.
The authors would like to thank Jef van Meerbergen from Philips Research Labs, Eindhoven (NL), for providing material on the driver applications used for this work.

References

[1] R.K. Gupta, G. De Micheli: System-level Synthesis using Re-programmable Components, Proc. EDAC 1992, pp. 2-8

[2] K. Buchenrieder, A. Sedlmeier, C. Veith: Design of HW/SW-Systems with VLSI Subsystems Using CODES, Proc. 6th IEEE Workshop on VLSI Signal Processing, 1993, pp. 233-241

[3] G. Goossens, F. Catthoor, D. Lanneer, H. De Man: Integration of signal processing systems on heterogeneous IC architectures, Proc. 6th Int. Workshop on High-Level Synthesis, 1992, pp. 16-25

[4] C.Liem, T.C.May, P.G.Paulin: Instruction Set Matching and Selection for DSP and ASIP Code Generation, Proc. European Design and Test Conference, 1994, pp. 31-37

[5] A. Fauth, M. Freericks, A. Knoll: Generation of Hardware Machine Models from Instruction Set Descriptions,Proc. 6th IEEE Workshop on VLSI Signal Processing, 1993, pp. 242-250

[6] P. Marwedel, W. Schenk: Cooperation of Synthesis, Retargetable Code Generation and Test Generation in the MIMOLA Software System, European Design and Test Conference, 1993, pp. 63-69

[7] R. Jöhnk, P. Marwedel: MIMOLA Reference Manual V 3.45, Technical Report No. 470, available from: University of Dortmund, Dept. of Computer Science, 44221 Dortmund, Germany

[8] L. Nowak, P. Marwedel: Verification of Hardware Descriptions by Retargetable Code Generation, Proc. 26th Design Automation Conference, 1989, pp. 441-447

[9] P. Hilfinger: SILAGE: A language for signal processing, Proc. CICC, 1985

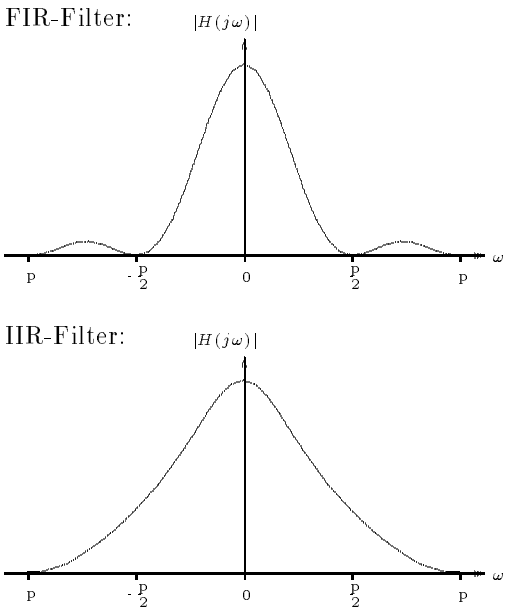

Figure 3: Filter transmission functions

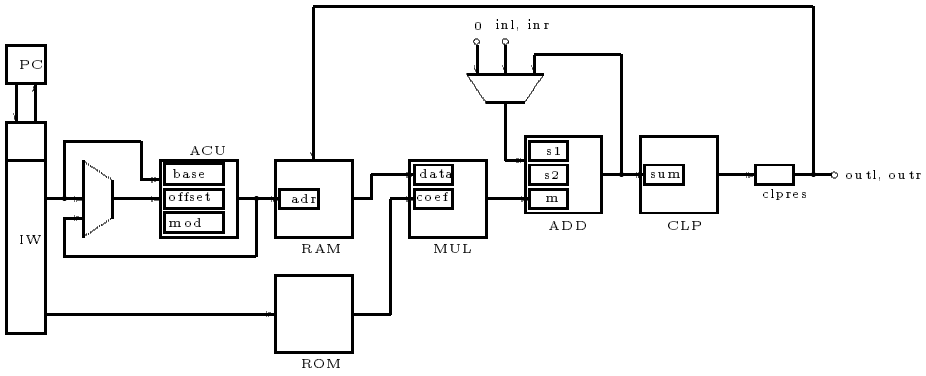

Figure 4: Target architecture for filter application