Design Flow for Hardware/Software Cosynthesis of a Video Compression System

Jörg Wilberg[+], *Raul Camposano* *, *Wolfgang Rosenstiel* **

[+] GMD-SET, Schloß Birlinghoven, D-53757 Sankt Augustin, Germany, wilberg@gmd.de

* Synopsys Inc., 700 East Middlefield Road, Mountain View, CA 94043-4033, USA, raul@synopsys.com

** University of Tübingen, Sand 13, D-72076 Tübingen, Germany, rosenstiel@peanuts.informatik.uni-tuebingen.de

Abstract- *The implementation of a cosynthesis design flow in the CASTLE system is presented. The design flow generates a synthesizable hardware description and a C, C++, or Fortran compiler for an application-oriented processor. The approach is illustrated by the design of an embedded video compression system which can be integrated into the video card of a PC. The design flow is structured as follows: First, the requirements of the application programs are analyzed. Based on these analysis results, the designer decides on the appropriate processor structure. The processor structure is entered on a block diagram level into the CASTLE system by using a schematic entry. The CASTLE system performs the processor cosynthesis based on a VHDL library of processor components. Several processor datapaths for the video compression system were synthesized to illustrate the trade-offs between flexibility and performance when designing application-oriented processors.*

1.0 Introduction

As mentioned by K. Hwang [18], achieving peak performance in a computer system demands a perfect match between the machine capability and the program behavior. Embedded Systems (ESs) [6] are special computer systems which perform specific functions within a host system. The statement above suggests a two step procedure for designing high-performance ESs: First, the requirements of the ES's application domain should be analyzed and second, the ES should be designed in such a way that it matches the determined requirements. Realizing this procedure in form of a system-level synthesis tool poses a number of challenging problems. The paper describes the approaches used for solving these problems in the CASTLE (Codesign And Synthesis TooL Environment) system [6].

One of the most challenging problems in ES design is the necessity of a multidisciplinary approach [6][32][4]. The knowledge of the application domain, the software (SW) area, the hardware (HW) design, and the design automation must be combined to build a high-performance ES in an efficient design process.

For example, the following design time distributions were experienced when manually developing an RT level hardware description and the necessary software for video processing systems [31]: 10% of the time for developing the concept, 30% for developing the HW description, and 60% of the time for developing the SW. Although these distributions might vary for different designs, they already indicate the necessity of a cosynthesis design approach. The HW design is just one part of the system. Especially if high-performance systems with a high degree of pipelining and instruction level parallelism are considered, the SW development on an assembler level can become extremely cumbersome. Therefore cosynthesis tools are required which synthesize both the processor HW and the compiler to generate SW for the designed processor.

Another important point is the conception phase. This phase requires a detailed knowledge of the application domain and the creativity and intuition of a designer. For example, a typical trade-off in video processing systems exists between flexibility and performance. In most cases specialized systems offer higher performance than flexible systems of the same size, but also have a limited application domain and hence a smaller market segment. The most suitable design point will be in the range between an off-the-shelf microprocessor and a function-specific ASIC [22]; i.e., an application-oriented processor. Specifying the design point should be done by the designer since, for example, future market developments must be anticipated, etc. Hence, decisions in the conception phase are not always clear cut and difficult, if not impossible, to automate. But these design decisions should be based on quantitative data [16][32]. The synthesis tool must provide a comfortable environment to measure and visualize these data, in order to support the designer as much as possible in specifying the system structure.

The paper describes the major steps of a cosynthesis design flow and the implementation of this design flow in the CASTLE system. The design of an embedded video compression system is used to illustrate the design flow. The remainder of the paper is organized as follows. The next section summarizes related work from the synthesis and the video processing domain. Section 2 presents an overview of the design flow for high-performance systems (HPS) in CASTLE. Section 3 describes the architecture of an embed-

This work has been sponsored by Bundesministerium für Forschung und Technologie BMFT, project 01M2897A SYDIS. The Compression Processor is developed in cooperation with Thesys GmbH, Erfurt, Germany.

0-8186-6315-4/94 $04.00

ded video compression system. Section 4 describes the cosynthesis environment. The main implementation results are given in section 5.

1.1 Related Work

This section briefly reviews recent work in the field of synthesis and the field of video processing. Significant progress was achieved in both fields. Therefore it is challenging to combine these fields and to use the video processing applications as an example for synthesis. The USC system [13] was used for synthesizing a dedicated JPEG video compression system from a system-level specification. The synthesis results obtained by the system-level synthesis were comparable to results obtained by RT-level synthesis.

A VLIW (Very Long Instruction Word) processor architecture is used as the synthesis framework in the CAPSYS[24] and the CATHEDRAL2/3 [12] systems. The opcode in case of the CAPSYS system is generated from programs written in an Ada-like language, whereas the CATHEDRAL2/3 currently uses Silage as specification language.

The PEAS-1 [2] system generates a compiler based on the GNU C compiler along with the hardware. But currently only a scalar, RISC-like processor architecture is supported.

The CODES [4] system supports concurrent design of hardware and software. Emphasis is put on formal specification and on the configuration of the system from off-the-shelf components, similar to the approach used in MICON [3].

COSMOS [20] also stresses formal specification and, similar to the CASTLE system, standard input and output languages are used. VULCAN [14] and COSYMA [8] focus on automatic partitioning between tasks for a general-purpose processor and an ASIC. The cosynthesis described in this paper extends this approach by considering the spectrum of more or less application-oriented architectures in between general-purpose microprocessors and ASICs.

Ptolemy [21] offers a heterogeneous codesign environment. The system is written in C++ and uses an object-oriented mechanism for abstracting communication among different tools. This approach allows to integrate external tools into the design environment.

The first generation of video processing chips were based on ASICs [9][27] synthesized from RT level HW descriptions. Currently, these dedicated systems are offered as macro cells by some ASIC vendors (e.g., LSI Logic offers a JPEG cell). As a disadvantage of dedicated architectures, a major redesign of the chips is required, if any changes in the video processing algorithms occur.

Therefore the current generation of video processing chips uses flexible processor architectures which are adapted to video processing [1][15][30]. An open problem in most cases is the compiler support for these complex and heavily pipelined processors.

2.0 Overview of the CASTLE Design Flow for High-Performance Systems

The CASTLE [6][32] system is a workbench for HW/SW codesign. All tools in CASTLE are connected by a SIR (System Intermediate Representation) format. The tools in CASTLE can be configured to form design flows for specific application domains. This paper presents a design flow for High-Performance Systems (HPS) such as the video compression system discussed in the next section.

A VLIW (Very Long Instruction Word) processor architecture is used as the framework for a cosynthesis of processor HW and corresponding compiler. The generic architecture is shown in Fig. 1.

The processor can be subdivided into a control unit and a datapath. The datapath consists of a number of functional units (e.g., adder, ALUs, multiplier) and storage units like register, register files, etc. These units are connected by multiplexers and busses. The operation of the datapath is

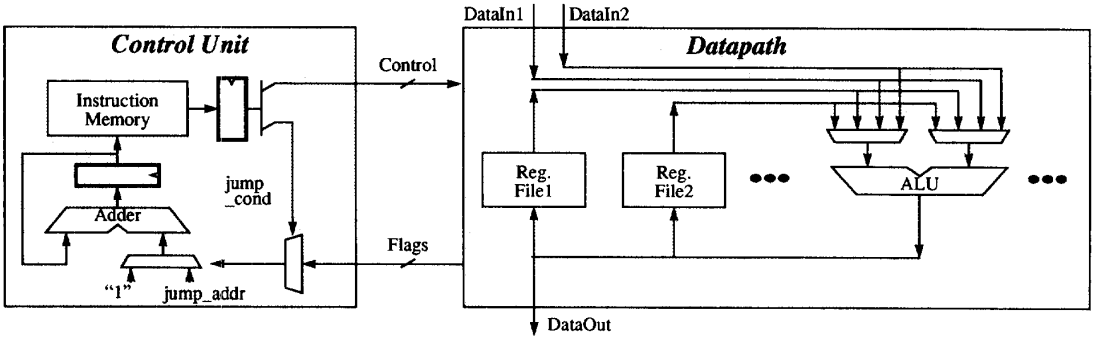

Fig. 1 Generic structure of a VLIW processor.

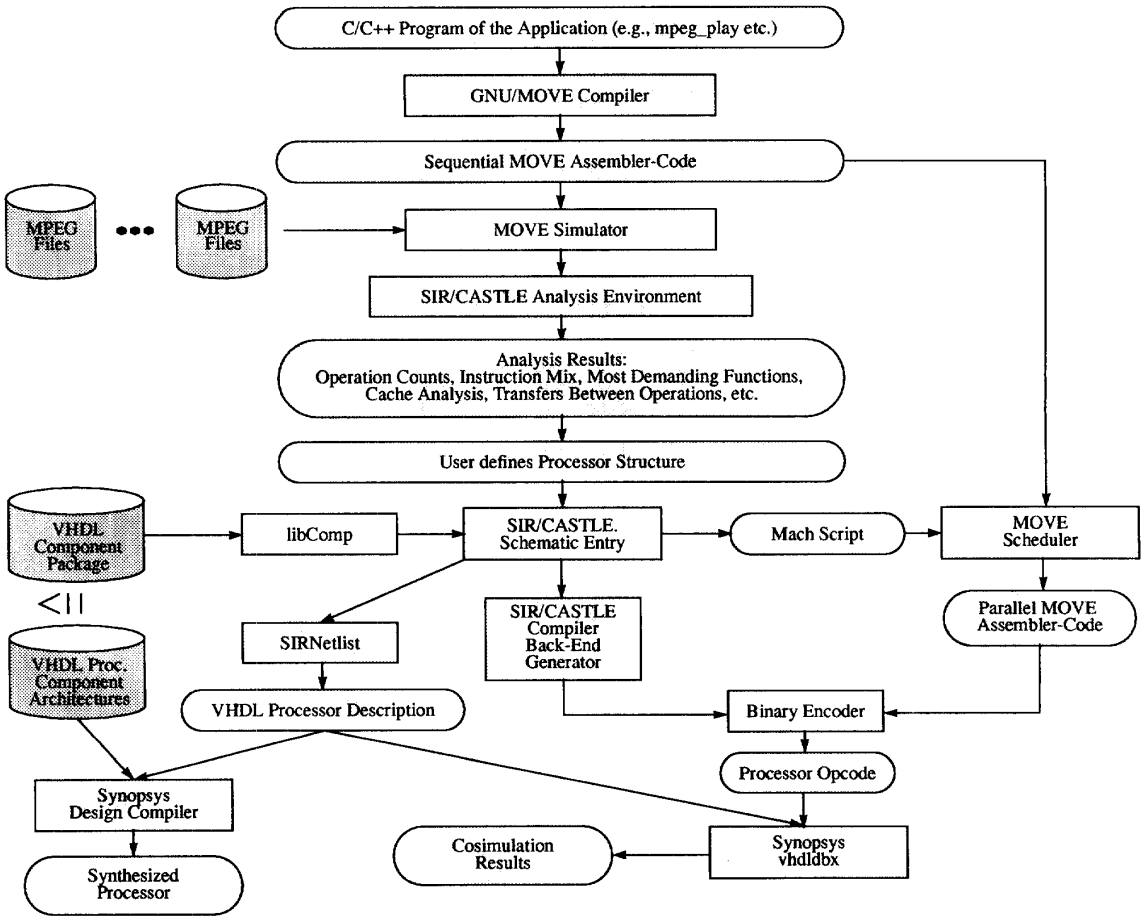

Fig. 2 Cosynthesis of application-oriented VLIW-Processors.

controlled by a control signal which corresponds directly to the instruction word of the VLIW processor. Flags of the functional units (e.g., carry out, etc.) can influence the control flow (i.e., branches can depend on a certain flag value). This generic architecture is similar to the typical architecture used in High-Level Synthesis (HLS) [5](e.g., the FSMDs in [10]), except that the control unit is kept programmable, whereas in HLS the control unit is typically hardwired into an FSM.

The design flow used for the cosynthesis of HPSs in CASTLE is depicted in Fig. 2. The following main steps are performed:

(i) One or more programs from the considered application domain are developed in a high-level language (C, C++, Fortran) on a workstation or PC. The conventional programming environment can be used for debugging and testing of the program(s). This offers a significant advantage concerning the design time, since for example a functional "simulation" of the application program can be done by simply executing the program on the workstation (this is currently about 30 times faster than simulating an equivalent VHDL or Verilog program). Furthermore, existing software, like the mpeg_play [26] program, can be used as application programs. Again, this can significantly reduce the design time when developing complex ESs.

(ii) The application program is compiled into a sequential intermediate code which consists of RISC-like 3-operand instructions. The compilation is handled by the MOVE compiler (which is based on the GNU compilers). The MOVE compiler was developed by J. Hoogerbrugge, et al. at the University of Delft [17].

(iii) A requirement analysis [32] is performed by simulating the 3-operand code with representative data files (e.g., different MPEG sequences in case of the mpeg_play). The execution profiles are recorded and the corresponding analysis data are visualized by the SIR/CASTLE Analysis Environment.

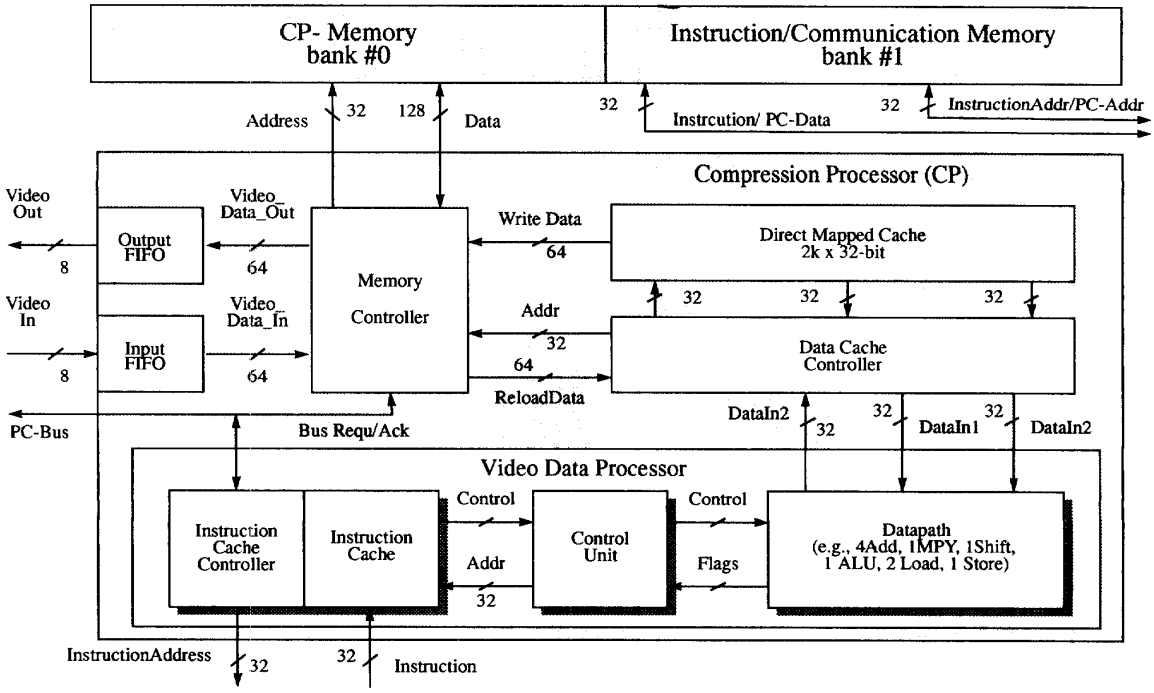

Fig. 3 Blockdiagram of the Compression Processor (CP).

(iv) The designer interprets these data and specifies the main processor structure.

(v) The SIR/CASTLE Schematic Entry is used to enter the specified processor on a block diagram level into the synthesis tool. The schematic is translated into synthesizable VHDL. The VHDL-description of the processor can be synthesized by any commercial synthesis tool (e.g., Synopsys' Design Compiler). Additionally, a compiler back-end is generated for the specified processor. The back-end allows to translate the sequential 3-operand code into a binary code for the processor. This opcode and the VHDL-hardware description of the processor can be used for a cosimulation on a conventional VHDL-simulator. The cosimulation results allow to validate the design by comparing these results with the results obtained from executing the application program on a workstation and from simulating the sequential 3-operand code.

The processor synthesis is discussed in more detail in section 4.0. The next section gives a brief description of the video compression system to be designed.

3.0 Embedded Video Compression System

Handling real-time video data in a computer system poses extreme demands on the storage capacity and the data transfer bandwidth. Sophisticated video compression schemes like MPEG [11][19] can be used to reduce these demands. Typically, the compression schemes require high processing power which makes application-oriented video processors necessary. This section describes a video compression system which is embedded into the video card of a PC.

A typical decoding sequence would proceed as follows: The PC retrieves the coded video data from a file or from a network. The data are transferred to the compression processor (CP) which decodes the data, and finally the decoded picture is written into the video buffer of the PC. The next section explains the architecture of the video compression system in more detail.

3.1 System Architecture

The toplevel block diagram of the compression system is shown in Fig. 3. The compression system consists of three main components:

(i) The video data processor performs the processing of the compression algorithms. The processor uses a direct mapped cache for both data and instructions.

(ii) The memory controller performs all address calculations and data transfers in the system. In addition, simple low level tasks, like filtering, decimation, and interpolation, can be

performed on the video input and output data.

From the SW point of view the memory controller implements an OS function similar to a small micro kernel for memory management. Four threads of control can be distinguished: loading of the data cache, interpolation and output of the video data, input and decimation of the video data, and arbitration between PC-memory access and the instruction fetch. From the HW point of view the memory controller can be realized as a second VLIW processor with special functional units for address calculation.

This system structure allows to overlap communication performed by the memory controller and data processing performed by the video data processor.

(iii) The memory of the compression system is subdivided into two banks. A homogeneous address space with high-order interleaving is used; i.e., the MSB of the address determines which bank is accessed. Typically, bank #0 will be used by the compression processor and bank #1 will be used by the instruction fetch and for communication with the PC.

The SIR/CASTLE Analysis Environment and the requirement analysis of the *mpeg_play* [26] program are presented in [32]. The next section describes the cosynthesis of a processor which matches these requirements.

4.0 Processor Cosynthesis

The basic VLIW architecture framework was presented in section 2.0. The main parts to be defined for the datapath are: the number and type of the functional units and the interconnection between these units. From the requirement analysis [32] the designer knows the number and the type of operations in the application programs and the number of data transfers between these operations. Based on these results, the designer can specify the most suitable processor structure. This step requires to consider future market developments, to trade-off flexibility and performance in order to obtain a good performance/cost ratio, and so on. As a result of this step, the designer will have a block diagram of the processor datapath in mind. Using the SIR/CASTLE Schematic Entry, the designer can directly enter this block diagram into the CASTLE system. That is, the Schematic Entry presents the designer a list of available processor components, the designer selects the component types (e.g., register file and adder for the simple example processor of Fig. 4), specifies a name for each unit instance, and determines certain generic values (e.g., the number of words in the register files). Next, the designer specifies the connections between the units by clicking on the unit ports to be connected. The system inserts the connection and multiplexers or register, if necessary. The resulting block diagram for a simple example processor is depicted in Fig. 4. This block diagram is translated into a synthesizable VHDL description, as described in the next section.

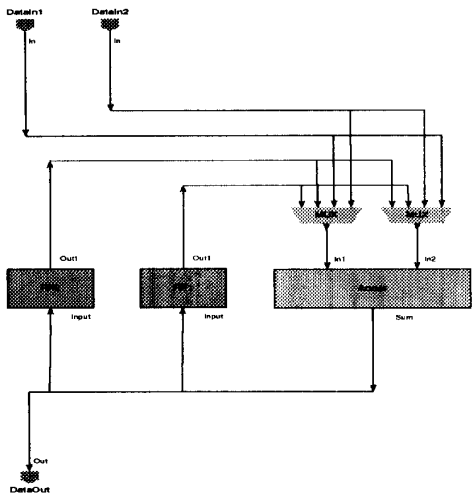

Fig. 4 Simple example processor datapath.

4.1 Hardware Synthesis

The synthesis in the HPS design flow is based on a VHDL library of generic components. The concept is similar to Synopsys' Design Ware [7] approach. Each component in the library defines a complete class of units (e.g., a register file with two generics: wordwidth and number of words in the file). Upon instantiation, the generics are fixed and a specific unit type is inserted into the netlist.

The library can be subdivided into two parts. A VHDL-component package which declares the interface of each library component, and a VHDL-entity/architecture file which describes the internal realization of each component. The VHDL-entity/architecture file is only used for the final HW synthesis. Hence, the component library can be adapted to different synthesis tools (e.g., Synopsys' Design Compiler, Mentor's Autologic, etc.) by simply changing the VHDL-entity/architecture file. Furthermore, full-custom macros can be used for demanding components. Due to the separation between VHDL-component package and VHDL-entity/architecture file, low-level optimizations of the implementation can be performed, without effecting the HW description generated by the CASTLE system.

The structure of the CASTLE Schematic Entry is shown in Fig. 5. The program is written in *itcl* [23] an object-oriented variant of the Tcl/Tk language [25].

The VHDL-component package is translated by a library compiler into a Tcl script. When the Tcl script is executed by the circuit manager, two data structures are created for each component: a schematic template and a netlist template. The schematic template contains the data ports and

the schematic symbol to be used when the component is displayed by the Schematic Entry.

The netlist template contains all ports of the component and the information necessary for creating a netlist instance of that component. The ports in the netlist template are subdivided into five groups: (i) dataIn, (ii) dataOut, (iii) control, (iv) flags, and (v) broadcastIn. The connections for dataIn and dataOut are specified by the designer. All other ports are hidden from the designer and connected automatically by the circuit manager. The port groups control and flags are concatenated and connected to the control unit of the processor. The port group broadcastIn contains signals like clock and reset which are passed to the toplevel of the processor.

A typical processor synthesis is performed as follows. The user enters the block diagram of the processor as a schematic. This specification is very abstract since the circuit manager "knows" the processor architecture (a VLIW architecture in the current implementation), and hence, most details of the implementation can be handled automatically by the circuit manager. Next, the circuit manager calculates all generics and creates a SIRNetlist of the schematic based on the netlist templates from the component library. The SIRNetlist can be dumped as synthesizable VHDL code.

4.2 Compiler Back-End Generation

The generation of a compiler back-end can be subdivided into two main steps:

(i) A mach script is generated. The script informs the scheduler of the MOVE system about the processor topology; i.e., it specifies the number and type of the functional units and the interconnection structure between these units. This allows the MOVE scheduler to perform the scheduling and allocation of the sequential assembler code. As a result, a parallel assembler code is generated where each instruction is annotated with the functional unit which executes this instruction.

(ii) The parallel assembler code is translated into the binary opcode for the processor. In a VLIW processor the instruction word directly corresponds to the control vector of the hardware. As an example, the instruction word for the simple processor of Fig. 4 and the encoding of an assembler instruction is depicted in Fig. 6.

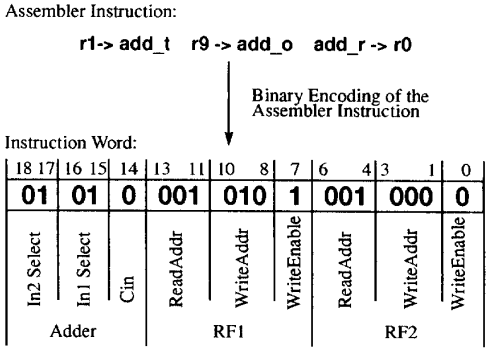

Fig. 6 Encoding of an addition into the instruction word for the example processor of Fig. 4. It is assumed that the registers r0 ... r7 are in RF1 and the registers r8 ... r15 are in RF2.

The next section summarizes the main implementation results.

5.0 Results

The CASTLE system was used to synthesize several different processor datapaths for the video compression system. All datapaths correspond to the principal analysis results of [32], but different degrees of flexibility were used when selecting the types of the functional units and the interconnection network. This illustrates the trade-off between flexibility and performance in the design of high-performance ESs.

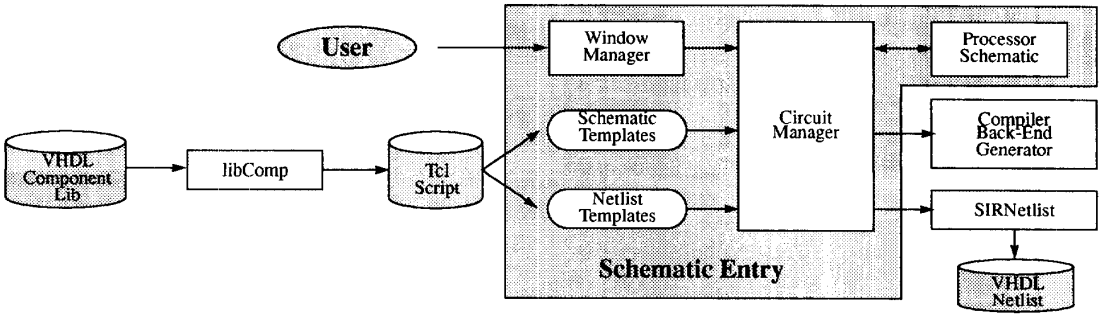

Fig. 5 Structure of the schematic entry in CASTLE.

The synthesis results obtained from synthesizing the VHDL processor descriptions with the Synopsys Design Compiler and the LSI 10k library are shown in Table 1.,and Table 2.

Processor Components	Parameters	Number of Instances	Control Bits	Gate-count
ALU_Type		1	5	1274
AddSubType		1	2	1601
AdderType		3	3	1605
CompareType		1	4	487
MPY_Type	3 stages	1	0	12617
MuxType	2 to 1	4	4	512
MuxType	4 to 1	2	4	754
RegFileR2W1Type	2read port, 1 write port, 4 words	2	14	4302
RegFileR2W1Type	2read port, 1 write port, 8 words	1	10	3825
RegisterType		3	3	963
ShifterType		1	5	985
TransferType		1	4	679
Processor1			58	29604

Table 1. Synthesis results for the Processor1.

The program sizes of utilized CASTLE programs are summarized in Table 3. The user interface of the schematic entry is shown in Fig. 7.

Program	Language	Lines of Code
Analysis Environment	C++ Other	3344 368
Schematic Entry	itcl	3836
libComp	GAWK	723
Library Component Package	VHDL	306
Library entity/architecture file	VHDL	1637
Library Testbenches	VHDL	1825

Table 3. Code sizes of different CASTLE programs.

Processor Components	Parameters	Number of Instances	Control Bits	Gate-count
ALU2 _CompareType		1	14	3035
ALU_Type		1	5	1274
AddSubType		1	2	1601
AdderType		1	1	535
MPY_ShifterType	4 stages	1	5	13602
MuxType	2 to 1	12	12	1536
MuxType	4 to 1	8	16	327
RegFileR2W1Type	2read port, 1 write port, 4 words	1	7	2151
RegFileR2W1Type	2read port, 1 write port, 8 words	3	30	11475
ShifterRotateType		1	6	1156
Processor2			98	36692

Table 2. Synthesis results for the Processor2.

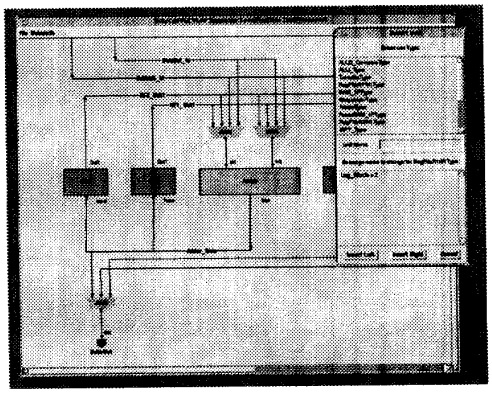

Fig. 7 User interface of the schematic entry in CASTLE.

6.0 Conclusion

The CASTLE design flow for high-performance systems was presented. Application-oriented VLIW (Very Long Instruction Word) processors are used as the basic architecture framework of the design flow. The design flow produces both, a synthesizable VHDL description of the hardware and a C, C++, Fortran compiler for the specified processor.

This approach improves the design efficiency significantly since application programs can be developed in high-level languages and existing software can be reused. Furthermore, the application programs can be directly tested on a workstation or PC which is much faster than simulation of a complex ES in VHDL or Verilog. The hardware is configured in terms of a VHDL library of complex processor components. The library is subdivided into two parts, a VHDL-component package which defines the interfaces of each component, and a VHDL-entity/architecture file which defines the implementation of each component. This allows to perform low level optimizations of the components without effecting the over-all processor structure. In addition, user defined components can be appended to the library. This supports a reuse of hardware components from existing designs. The described cosynthesis design flow was used for the design of an embedded video compression system.

7.0 References

[1] B. D. Ackland, et al.: "A Video-Codec Chip Set for Multimedia Applications", AT&T Technical Journal, vol. 72, no. 1, pp. 50 - 66, Jan./Feb. 1993.

[2] A. Alomary, et al.: "PEAS-I: A Hardware/Software Co-design System for ASIPs", Euro DAC, Hamburg, pp. 2-7, 1993.

[3] W.P. Birmingham, et al.: "MICON: Automated Design of Computer Systems", in R. Camposano and W. Wolf (editors): "High-Level VLSI Synthesis", pp.205-229, Kluwer Academic Publishers, Boston/Dordrecht/London, 1991.

[4] K. Buchenrieder, et al.: "HW/SW Co-Design With PRAMs Using CODES", in D. Agnew et al.(eds.): "Computer Hardware Description Languages", IFIP Trans., vol. A-32, 1993.

[5] R. Camposano, W. Wolf (eds.): "High-Level VLSI Synthesis", Kluwer Academic Publishers, Boston, 1991.

[6] R. Camposano, J. Wilberg: "Embedded System Design", to appear in T. Lengauer (ed.), Springer, Lecture Notes in Computer Science.

[7] "Design Ware Databook", Synopsys® Inc., vers. 3.1a, March 1994.

[8] R. Ernst, J. Henkel and Th. Benner: "HW/SW cosynthesis for microcontrollers", IEEE Design & Test, pp. 64-75, Dec. 1993.

[9] H. Fujiwara, et al.: "An all-ASIC Implementation of a Low Bit-Rate Video Codec", IEEE Trans. Circuits And Systems On Video Technology, vol. 2, no. 2, pp. 123-134, June 1992.

[10] D. D. Gajski, F. Vahid, S. Narayan: "System-Level Methodology and Technology", E-DAC Tutorial, 1994.

[11] D.J. Le Gall: "The MPEG video compression algorithm", Signal Processing: Image Communication, vol. 4, pp.129-140, 1992.

[12] G. Goossens, et al.: "Integration of medium-throughput signal processing algorithms on flexible instruction-set architectures", to appear in J. VLSI Signal Processing (special issue in synthesis for real-time DSP), 1993.

[13] P. Gupta, et al.: "Experience with Image Compression Chip Design using Unified System Construction Tools", to appear DAC, 1994.

[14] R.K. Gupta, G. De Micheli: "Hardware-Software Cosynthesis for Digital Systems", IEEE Design & Test, vol. 10, no. 3, pp. 29-41, Sept. 1993.

[15] K. M. Guttag: "Multimedia Powerhouse", Byte, pp. 57-64, June 1994.

[16] J.L. Hennessy, D. A. Patterson: "Computer Architecture: A Quantitative Approach", Morgan Kaufmann Publ., 1990.

[17] J. Hoogerbrugge, H. Corporaal: "Transport-Triggering vs. Operation-Triggering", Compiler Construction Conference, 1994.

[18] K. Hwang: "Advanced Computer Architecture: Parallelism, Scalability, Programmability", McGraw-Hill, 1993.

[19] ISO/IEC DIS 11172 : "Informationtechnology-- Coding of moving pictures and associated audio for digital storage media up to about 1.5 Mbit/s", 1992.

[20] A. A. Jerraya, et al.: "Linking System Design Tools and Hardware Design Tools", in D. Agnew et al.(eds.): "Computer Hardware Description Languages", IFIP Trans., vol. A-32, 1993.

[21] A. Kalavade, E.D. Lee: "A Hardware-Software Codesign Methodology for DSP Applications", IEEE Design & Test of Computers, vol. 10, no. 3, pp. 16-28, Sept. 1993.

[22] K. Keutzer: "Trends and Problems in Electronic System Design Automation", Synopsys Summer Session, Aug. , 1993.

[23] M.J. Mc Lennan: "[incr Tcl] – Object-Oriented Programming in Tcl", AT&T Bell Laboratories, Allentown, PA 18103, michael.mclennan@att.com.

[24] G. Menez, et al.: "A Partitioning Algorithm For System-Level Synthesis", Int. Conf. CAD, pp. 482-487, 1992.

[25] J. K. Ousterhout: "Tcl and the Tk Toolkit", Addison-Wesley Publishing Company, Reading, MA, 1994.

[26] K. Patel, et al.: "Performance of a Software MPEG Video Decoder", Proc. 1st ACM Int. Conf. on Multimedia, Anaheim, CA, 1993.

[27] P. A. Ruetz, et al.: "A High-Performance Full-Motion Video Compression Chip Set", IEEE Trans. Circuits And Systems on Video Technology, vol. 2, no. 2, pp. 111-122, 1992.

[28] D. E. Thomas, J.K. Adams, H. Schmit: "A Model and Methodology for Hardware-Software Codesign", IEEE Design & Test, vol. 10, no. 3, pp. 6-15, Sept 1993.

[29] R. A. Walker, R. Camposano: "A Survey of High-Level Synthesis Systems", Kluwer Academic Publishers, Boston, MA, 1991.

[30] P. Wayner: "Digital Video Goes Real-Time", BYTE, pp. 107-112, Jan. 1994.

[31] J. Wilberg, et al.: "Hierarchical multiprocessor system for video signal processing", Proc. SPIE , vol. 1818, Nov. 1992.

[32] J. Wilberg, R. Camposano, U. Westerholz, U. Steinhausen: "Design of an Embedded Video Compression System – A Quantitative Approach", to appear Int. Conf. Computer Design, Cambridge, MA, Oct. 10-12, 1994.