Combined Control Flow Dominated and Data Flow Dominated High-Level Synthesis

E. Berrebi[1,3] P. Kission[1] S. Vernalde[2] S. De Troch[2]
J.C. Herluison[3] J. Fréhel[3] A.A. Jerraya[1] I. Bolsens[2]

[1] TIMA 46, av. Félix Viallet 38031 Grenoble, France
[2] IMEC Kapeldreef 75 B-3001 Leuven, Belgium
[3] SGS-Thomson Microelectronics 850, rue Jean Monnet, BP16 38921 Crolles, France

Abstract

This paper presents the design of a Videophone Coder-Decoder Motion Estimator using two High-Level Synthesis tools. Indeed, the combination of a Control Flow Dominated part (complex communication protocol) with a Data Flow Dominated part (high throughput computations) makes this circuit difficult to be synthesized by a single HLS tool.

The combination of two HLS tools for the high-level design of this operator required the definition of a sophisticated design flow allowing mixed-level and multi-language simulations. When compared to design starting from RTL specifications, HLS induces only a negligible area overhead of 5%, and provides gain in description length (divided by 5), design time and flexibility.

1. INTRODUCTION

In video-transmission, for instance, the rapid evolution of the standards, with an increasing complexity, rises dramatically the number of transistors on chips and the corresponding design-time. To meet time to market constraints, electronic industrialists have to gradually increase the productivity. Hence, the design flow of digital circuits have to start at gradually higher abstraction levels. Today, designers begin to apply Behavioral (or High Level) Synthesis. But the lack of maturity of existent tools and methodologies sets limitations.

This paper presents the High-Level Synthesis (HLS) of a complex operator (the Motion Estimator) of a chip (SGS-Thomson-Microelectronics H261 Videophone Coder-Decoder) in an industrial environment.

1.1 Design Flow

Figure 1.a presents different abstraction levels used for circuit design and the corresponding translation steps. Figure 1.b shows the RTL design process commonly applied in industry. Figure 1.c shows the High-Level Synthesis based methodology.

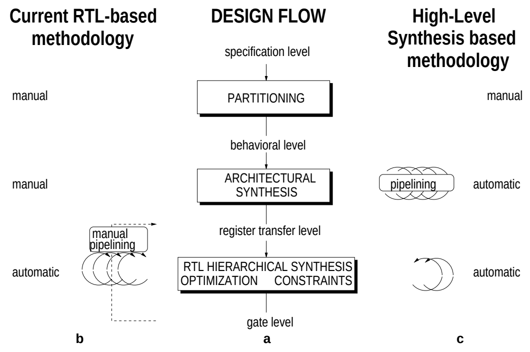

Figure 1: Design Processes

From the initial specification level, the circuit is partitioned manually for both methodologies and described at the behavioral level. This description is simulated to check the functionalities and protocols with external processes.

Then, the architectural synthesis is done manually in the RTL-based methodology (figure 1.b), whereas it is automatically generated in the high-level synthesis based methodology (figure 1.c). The automation of this step allows: gain in design time, flexibility towards modifications of the specifications, architectural exploration and automatic pipeline stage introduction.

Once validated, the generated RTL description is then synthesized, optimized and constrained with an RTL and logic synthesizer resulting in a gate level description to be placed and routed.

Section 5 compares the results of the Motion Estimator design obtained: with the RTL-based methodology and with the proposed High-Level Synthesis based methodology.

1.2 Taxonomy of High-Level Synthesis Designs

The basic function of HLS is the mapping of a behavioral description of a digital system onto an RTL design consisting of a data path and a control unit [4]. The main steps executed by HLS are scheduling and allocation. Most existing HLS tools are specialized for a restricted class of applications. According to Gajski [4], hardware designs may be classified in three categories:

- control dominated designs represented by Control Flow Graph;
- data dominated designs represented by Data Flow Graph;
- control-data dominated designs represented by Control-Data Flow Graph.

1.2.1 Control Flow Dominated Circuits

A typical Control-Flow Dominated design consists of a large control unit, possibly with a small data path. In this case, a behavioral description consists generally in a sequential process. This model describes computation steps separated by synchronization statements (e.g. *wait* statements in VHDL). The most critical step when synthesizing this type of circuit is scheduling.

Examples of tools dedicated to Control-Flow synthesis are CALLAS-VOTAN [8], HIS [2], YSC [1], CAMAD [15], Olympus [13], and Amical [10].

1.2.2 Data Flow Dominated Circuits

A typical Data-Flow dominated design has minimal control but a large data path to perform operations. In this case a behavioral description is composed of parallel data flow operations. The most critical step when synthesizing this kind of circuit is allocation, i.e. resource sharing.

Examples of tools dedicated to Data-Flow synthesis are AWB [16], Phideo [12], and Cathedral-2/3 [7].

1.2.3 Control-Data Flow Dominated Circuits

A Control-Data-Flow dominated design consists of a large control unit with a large data path to perform operations. The HLS of this kind of circuits requires the capabilities of both above tools.

The Motion Estimator is an example of a Control-Data Flow dominated circuit. Indeed, it has both a DSP-like computation feature and a complex control communication with the rest of the CODEC. This combination of a Control Flow Dominated (CFD) part and of a Data Flow Dominated (DFD) part makes the Motion Estimator difficult to be synthesized by one single HLS tool.

Today, none of the existing tools is able to synthesize efficiently such a circuit starting from a true behavioral description. Some existing tools such as in [11] are able to handle such a design, partially. But in this case, several parts of the Motion Estimator would have to be written at the Register Transfer Level.

1.3 Mixed CDFD HLS

When analyzed, the ME function can be decomposed easily into two separate main sub-functions: a Control-Flow dominated part and a Data-Flow dominated part. Starting from such a decomposition the idea is to use two separate tools allowing to combine their respective advantages without any restriction.

This paper presents the High-Level Synthesis of the Motion Estimator circuit, combining two HLS tools: a "CFD" compiler, Amical [10], with a "DFD" compiler, Cathedral-2/3 [7]. To evaluate the cost of the "high-level" methodology against the "RTL" one, we applied the same industrial conditions and used the same cycle-based test-bench for the verification at the behavioral and RT levels.

Several industrial experiments are reported in the HLS literature [3, 5]. However, to our knowledge, no such experiment, using the results of different HLS tools, has been carried out. The use of more than one HLS tool implies the definition of a sophisticated design flow with the integration of the results of different compilers, mixed-level and possibly multi-language simulations in particular at the behavioral level.

This paper is organized as follows: section 2 presents the Video-CODEC Motion Estimator and section 3 the tool set used for this experiment. Section 4 describes the new methodology, section 5 shows its application to the studied example and the corresponding results, section 6 analyzes the benefits of HLS and section 7 gives some conclusions.

2. THE MOTION ESTIMATOR

The considered Motion Estimator is part of the Videophone CODEC chip [6]. The CODEC chip codes and decodes sequences of pictures through a pipeline of 12 operators, communicating through a command bus and a data bus (figure 2). The coding part may also compress sequences of pictures without significant deterioration of the quality, avoiding redundancies. The coding efficiency depends on the Motion Estimator results. Therefore this operator is a bottleneck with consequences on the whole chip efficiency.

All the operators of the CODEC share the same Video-RAM. Access conflicts to this RAM are managed by a specific memory controller. Each operator has its local memory. Transfers between the VRAM and the local memories are made through a complex asynchronous protocol.

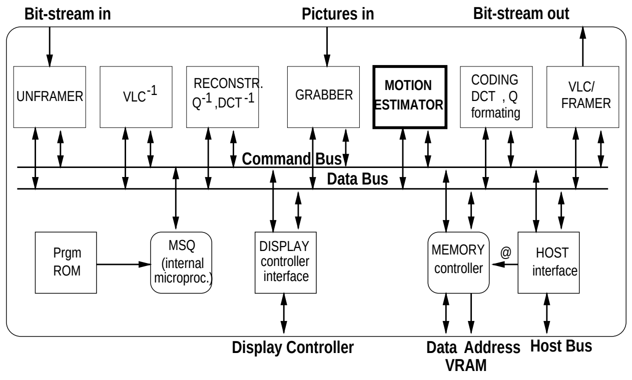

Figure 2: Videophone CODEC Architecture

The purpose of the Motion Estimator is to determine the motion vector of a moving part of a picture by comparison of the current frame and the previous one, stored in the VRAM (figure 2).

According to the H261[1] CCITT[2] algorithm [18], a picture is cut into 99 macro-blocks to allow parallelism between the different operators. The search window of the motion vector is limited to scan 16 horizontal vectors and 16 vertical vectors in the previous picture. The Motion Estimator calculates the "distance" or distortion between the current macro-block and the 256 existing possible positions of the macro-blocks in the search window.

The function of the Motion Estimator is mainly a DFD operation. However, it also involves a quite complex Control Flow. Indeed, because of the distributed architecture of the CODEC, it includes two local memories that are filled using a complex external protocol. Besides it communicates with the MSQ (sequencing the overall operations of the chip) through a well defined protocol.

3. HLS TOOL SET USED

This section describes the design-flow of the HLS tools used for this experiment: Amical and Cathedral-2/3. It also draws up their differences and complementarity.

3.1 The CFD tool : Amical

Amical is based on a flexible target architecture model allowing hierarchy and design re-use [10]. It is composed of a top controller, a set of functional units and a communication network (figure 3). These last two constitute the data path.

[1]standard for the coding of moving picture information
[2]International Telegraph and Telephone Consultative Committee

The communication network (between functional units, and with the external world) is composed of buses, switches, multiplexers and registers.

The top controller sequences the operations executed (possibly in parallel) by the functional units and the communication network. It is generated automatically during the synthesis process.

Functional units are used as black boxes. They may correspond to already existing blocks (memory, I/O unit,...) or a processor performing a specific function that has to be synthesized using Amical, Cathedral-2/3 or any other specific HLS tool. These FUs may communicate directly with external buses or with other FUs.

The different steps involved in the Amical design-flow are: scheduling, allocation and architecture generation (figure 3).

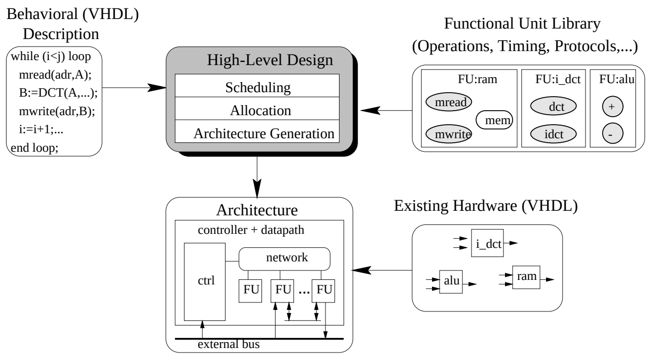

Figure 3: Amical design-flow

3.2 The DFD tool : Cathedral-2/3

Cathedral-2/3 is a software environment for synthesizing Digital Signal Processing (DSP) algorithms into Application Specific Integrated Circuits (ASICs) [7]. To target high throughput DSP applications such as video and image processing (critical parts speed-up, more area-efficient mappings), Cathedral-2/3 is well tuned for the synthesis of Data Flow Processors (**DFP**).

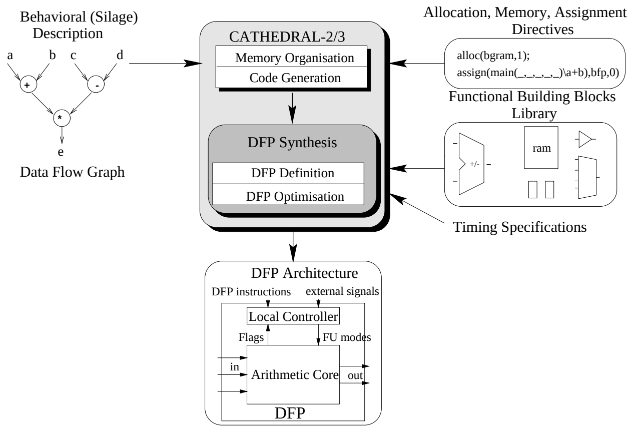

Figure 4: Cathedral design-flow

A DFP consists of an arithmetic core and a local controller. The local controller performs instruction decoding and local decision making. No limits are imposed on the complexity of the DFP.

The different steps involved in the DFP synthesis process are: memory organization, code generation, DFP definition and DFP optimization (figure 4).

The entry to the silicon compiler is an algorithmic description given in Silage. Silage is an applicative language especially suited to describe DSP functions [17].

3.3 Complementarity of the tools

Table 1 draws up the main features of the considered HLS tools. It stands out their complementarity. This section details why Amical is rather suited for CFD circuits and Cathedral-2/3 for DFD circuits.

According to this table it is clear that the high-level synthesis of the Motion Estimator needs the combination of both styles, as it includes DFD and CFD parts.

Indeed Amical is not suited for the synthesis of the whole circuit: to reach the requested high throughput, a manual description of the whole data path pipelining structure would be necessary. This would come to write the RTL description of the initial design process for the high-throughput calculation part.

Moreover Cathedral-2/3 is not suited for the synthesis of the whole circuit: for example, there is no procedural IF-THEN-ELSE or WHILE statement in Silage. Additional control signals and loop counters would be defined. This would come to write the finite state machine.

Features	**Amical**	**Cathedral-2/3**
Communication and protocol	Unrestricted handshaking	stream oriented
Loops	Data dependent loops + fixed-length loops	Fixed-length loops
Pipelining	Control pipelining*	Unrestricted data-path pipelining
Input description	VHDL description	Silage algorithm
Output description	RTL VHDL	RT or gate level VHDL

* possible insertion of a pipeline stage between the control part and the data-path

Table 1: Comparison of Amical and Cathedral features

4. METHOD

Here is presented the design process and the verification stages for the HLS of a circuit with a complex control part, by the "CFD" compiler Amical, and a complex calculation part, by the "DFD" compiler Cathedral-2/3.

4.1 Design process

The design method proceeds in a hierarchical way [10] (figure 5). The decomposition of a system specification into a global controller and (e.g. DFD) sub-systems performing specific tasks, allows to handle very complex design through hierarchy with the dedicated HLS tools. Of course the integration of the results of different compilers must be possible. Here follows the description of the different steps (figure 5).

4.1.1 The manual partitioning

A high-level specification of a complex system needs only to describe the sequencing of the tasks to be executed by the sub-systems. From the specifications, the designer has to determine which tool will synthesize each task. It consists in partitioning the circuit into a CFD part and a DFD part taking into account the easiest way to describe them (Silage

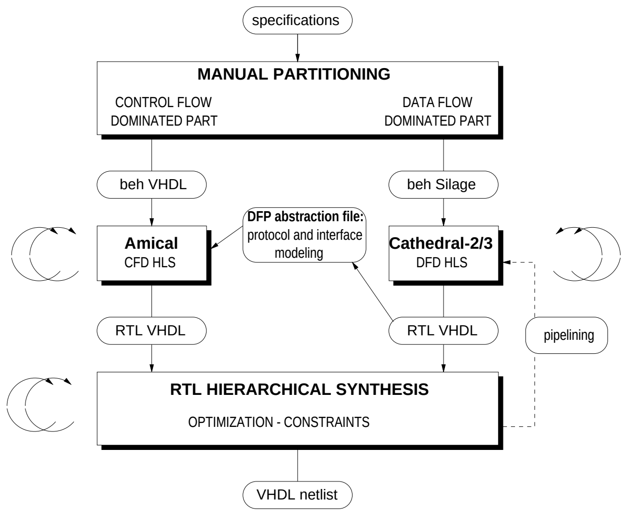

Figure 5: New design process used in the experiment

or behavioral VHDL), the way they will communicate before and after synthesis and in particular the ability to abstract the DFP for its re-use as a functional unit.

4.1.2 Cathedral-2/3 synthesis

Once the tasks to be synthesized by Cathedral-2/3 are defined, they are described in Silage. After code generation and DFP definition, the critical path is calculated. If it does not satisfy the timing requirements, one or more pipeline stages can be added to increase the throughput.

4.1.3 Abstraction and design re-use of the DFP

The DFD system produced by Cathedral-2/3 needs to be abstracted in order to be re-used for the design of the full system [9, 10]. This step consists in defining the communication protocol between the Top Control and the component. Each Functional Unit (FU) can be specified at three different abstraction levels. Starting from the implementation of the DFP (figure 6(a)), we produce a conceptual view (figure 6(b)). The object DFP is able to execute a set of functions (start, stop...). This abstraction will be used to produce the behavioral model of the components (figure 6(c)). These operations will hide the implementation details during the re-use of this component.

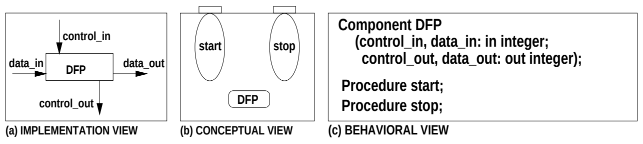

Figure 6: The three abstraction levels

This solution allows parallelism with other FU activity. The protocol consists in selecting the DFP mode of processing, and to wait for an output control signal meaning the end of the calculation and that the results are available. An encapsulation of the component inside a VHDL RTL architecture may ease the protocol.

4.1.4 Amical synthesis

It starts with two kinds of information required for synthesis: a behavioral specification and a functional unit library. The use of complex sub-systems is made through procedure and function calls. For each procedure or function used, the library must include at least one functional unit able to execute the corresponding operation. The system will select automatically the "best" solution according to an allocation/binding heuristic [19].

To allow DFP allocation [10], Amical needs a dedicated abstraction file including the interface (specifying the control signals), its call-parameters (corresponding to the operation parameters), the operation set as well as the parameter passing protocol for each operation.

4.2 Verification and mixed-level simulations

The validation of the specifications and the different synthesis steps is a critical stage when different tools, with different input languages, at different levels of abstraction are involved.

In order to verify that the global behavior of the circuit is the same whatever the level of abstraction, a unique VHDL RTL test-bench is used during the whole synthesis process to validate: the behavioral specifications, the RTL specifications produced by HLS and finally the netlist produced by RTL synthesis.

Figure 7 sums up different cases of co-simulation [14] and the role they play. None of these is useless. Each may point out some problems and detect errors on specifications and synthesis process. The simulation '1' adjusts the behavioral descriptions. The co-simulations '2' (resp. '3') validate Cathedral-2/3 synthesis (resp. Amical synthesis). '4' validates the synthesis of the DFP activation protocol and checks delays. '5' and '6' validate each RTL and logic synthesis. And finally '7' validates the complete design-process.

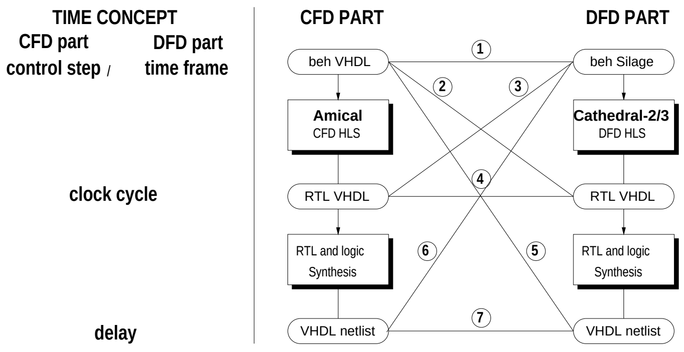

Figure 7: Mixed-level simulations

5. RESULTS

This section presents how the methodology has been applied to the example of the Motion Estimator and its results [14]. To compare the new methodology with the "RTL" one, constraints were imposed on the frequency, memories and external communications:

• To follow some of the external protocols described at the clock cycle level: an external data-bus updates these RAMs using a handshake protocol between the motion estimator, the CODEC Master Controller (MSQ in figure 2) and the VRAM controller, organizing the traffic of the required data.

• The chip frequency is 13.5MHz, but the distortion calculation must be processed three times more rapidly, at 40.5MHz, for a throughput of 15 images/sec implying 97.3 million operations/sec;

• For macro-block and search window storing, the use of two hardwired RAMs with setup time details was imposed.

5.1 Partitioning

The high-level synthesis of the Motion Estimator begins with the partitioning of the circuit into a CFD part and a DFD part to be processed respectively by Amical and Cathedral-2/3.

5.1.1 The main tasks

Figure 8 illustrates the sequencing of the main tasks involved in the Motion Estimator computation. In order to reach the required speed, the algorithm is organized in such a way that computations can be performed in parallel. In this case, these computations include the processing and the search cache updating.

After the initialization step, the current cache filling begins the process loop. For each current macro-block, the complete calculation of the motion vector is divided into two halves: the "1st half processing" and the "2nd half processing". During each half of the processing, a part of the search cache must be updated in order not to waste time for the next macro-block.

For this example, the partitioning between Amical and Cathedral-2/3 works, is guided by the parallelism and the frequency requirement: it stood out that the tasks at 40.5 MHz (distortion calculation and pixel fetching) would be ensured by a DFP and that the rest of the circuit (instructions sequencing, management of the memories and the communications) would be synchronized by the 13.5 MHz clock.

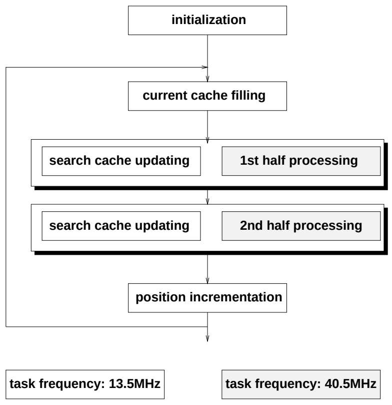

Figure 8: Specifications

5.1.2 Circuit architecture

The global architecture (figure 9) of the resulting circuit is made of a set of cooperative FUs ("ALU", "DFP", "search cache", "current cache") controlled by a Top Controller.

The ALU computes the address calculation for writing.

The DFP is activated through a mechanism of "protocol-encapsulation", defined in [10]. The goal of encapsulation is to reduce the number of actions to be performed by the global controller in order to use a given FU. This is achieved by adding an extra controller to the FU and a definition of a set of high-level primitives for the communication with the FU, (procedure call and wait statement until the "end signal").

We decided to encapsulate the RAM containing the search window, adding a local controller and to activate it through another "protocol-encapsulation". Indeed, it is used four times in the main process (twice in the initialization step), and the sequence of the address calculations is quite complex for this memory.

As the RAM containing the current macro-block is filled only once in the loop processing, and as the address calculation needs a simple increment, we just instantiated it without encapsulation. The protocol is flattened in the Top Controller.

The Top Controller works at 13.5 MHz whereas the DFP works at 40.5 MHz. As stated above the global architecture is generated by Amical. This is almost the same organization as the one made by the designer for RTL methodology. As shown in (figure 9), the architecture needs three kinds of communication. The communication between the data path and the controller is handled by Amical. The inter-FUs communications and the interaction of FU with the external world are not synthesizable by Amical, but preserved and included in the RTL model.

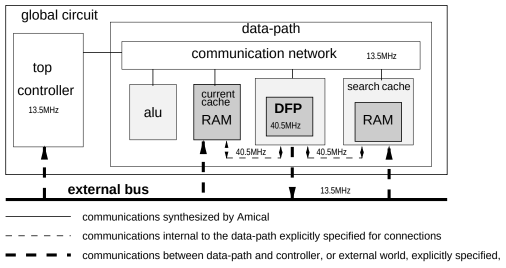
global circuit
data-path
communication network
13.5MHz
top
controller
13.5MHz
alu
current cache
RAM
DFP
40.5MHz
search cache
RAM
40.5MHz
40.5MHz
external bus
13.5MHz
communications synthesized by Amical
communications internal to the data-path explicitly specified for connections
communications between data-path and controller, or external world, explicitly specified,

Figure 9: Global architecture of the circuit

5.2 Synthesis Results

We used the same commercial synthesis tool for the RTL synthesis and the technology mapping applied for the previous design of the motion estimator. The circuits of both methodologies have been implemented using a specific library of standard cells (0.5μm CMOS technology).

RTL synthesis produced the results presented in table 2. These results (number of VHDL or Silage lines, number of standard cells and registers, area, critical path) are compared to those of the manual methodology described in section 1. Separate comparisons are done for the Data Flow part, the Control Flow part and the overall circuit.

To reach the required frequency, we applied retiming and one level of pipelining for the DFP with Cathedral-2/3 and we pipelined the control part and the data path with Amical. These results are analyzed in the next section.

Parameters	# lines	# logic cells	# Flip-Flops	area (μm)	critical path (ns)
DATA FLOW PART					
manual	1381	3153	551	1174525	22.0
Cathedral-2/3	300	2630	877	1220000	22.4
overhead	-78%	-17%	+59%	+4%	+2%
CONTROL FLOW PART					
manual	668	1146	82	299640	65.90
Amical	136	1039	133	320650	62.44
overhead	-80%	-9%	+62%	+7%	-5%
whole circuit	-79%	-15%	+60%	+5%	

Table 2: Synthesis results of the "HLS" methodology compared to the "RTL" one

6. ANALYSIS

The advantage of this experiment is obviously the automation of the passage to the RT Level from a behavioral specification, the flexibility towards architectural modifications, **without significant area increase!** Here follows a detailed analysis of the results in table 2.

A very reasonable area overhead on the whole circuit of 5%: 7% for the CFD part and to 4% for the DFD part. It is due to the pipe stages introduced. In this case, when compared to the total chip area [6] this represents less than one tenth of a percent overhead!

The reduction of design-time is one of the major advantages of this methodology. It is due to several reasons:

• **The behavioral description is flexible** due to its shortness (length divided by 5).

• **High-level synthesizers are less time-consuming** than manual translation.

Amical synthesis time amounts to a few minutes. The adjustment of the functional units abstraction files may need a few iterations through Amical.

Cathedral-2/3 synthesis time amounts to a few minutes for the high-level synthesis part. The pipelining and gate level optimizations take another hour. Reiterating with different numbers of pipeline stages takes 10 minutes.

The gate level optimizations can also be done by commercially available logic synthesis tools.

• **The number of RTL to netlist synthesis iterations is reduced** because of the automated generation of the VHDL description. Besides, it depends only on the adjustment of the functional units RTL descriptions. One iteration amounts to many hours. The synthesis script adjustment is otherwise the same as in the methodology starting from the RT level.

A disadvantage of this method is that the debug stage is tricky and hierarchical for this methodology, beginning with the validation of the sub-systems. In this case this issue was simplified by the use of a single test-bench for all the design steps.

7. CONCLUSIONS

The successful high-level synthesis of the Motion Estimator by Amical and Cathedral-2/3 pointed out that compared with the methodology starting from the Register Transfer Level, the gain in description length (divided by 5) is very appreciable, design time and flexibility, in particular versus the standards in video transmission and technology evolutions. The cost of this automated step is negligible with an area increase of 5% on the whole circuit, but it needs more registers.

Consequently, to perform the High-Level Synthesis of complex industrial circuits, it is possible to call for the most adapted combination of a CFD tool and a DFD tool if the circuit consists of a CFD part and a DFD part. In this case the designer must consider that the partitioning between these parts is critical, and that the verification needs mixed-level and possibly multi-language simulations with a unique test-bench. The most comfortable solution would be to use a unique environment that supports both types of HLS tool, but no such industrial development is reported in the present state-of-the-art, and is still an open challenge.

Acknowledgments

This design would not have been possible without the great help of the TIMA/SLS and IMEC/VSDM teams. We would also like to thank especially Michel Harrand and the following persons from SGS-Thomson Microelectronics: Jean-Pierre Moreau, Pierre Paulin and Jean-Pierre Schoellkopf for their discussions and suggestions regarding this work, and in particular Joseph Borel for his encouragements.

References

[1] R.K. Brayton, R. Camposano, G. de Micheli, R.H.J.M. Otton, and J. Van Eijndhoven. *Silicon Compilation*, chapter The Yorktown Silicon Compiler System, pages 204–310. Addison-Wesley Publishing Company, D.D. Gajski edition, 1988.

[2] R. Camposano, R.A. Bergamaschi, C.E. Haynes, M. Payer, and S.M. Wu. *Trends in High-Level Synthesis*, chapter The IBM High-Level Synthesis System. Kluwer Academic Publishers, 1991.

[3] T.E. Fuhrman. Industrial Extensions to University High-Level Synthesis Tools: Making It Work in the Real Work. In *Proc. of 28th ACM/IEEE Design Automation Conference*, 1991.

[4] D.D. Gajski and L. Ramachandran. Introduction to High-Level Synthesis. *IEEE Design and Test of Computers*, pages 44–54, 1994.

[5] Mark Genoe, Paul Vanoostende, and Gert Van Wauwe. On the use of VHDL-based behavioral synthesis for telecom ASIC design. In *Proc. of 8th International Symposium on System Synthesis*, pages 96–101, September 1995.

[6] M. Harrand, et al. A Single Chip Videophone Video Encoder/Decoder. In *Proc. of IEEE International Solid-State Circuits Conference*, pages 292–293, February 1995.

[7] IMEC. *Cathedral-2/3 Silicon Compiler for Real Time Signal Processing.*

[8] M. Koster, et al. J. Biesenack. The Siemens High-Level Synthesis System, CALLAS. In *6th Intl. High-Level Synthesis Wkshop*, November 1992.

[9] A.A. Jerraya, et al. A Pragmatic Approach to Behavioural Synthesis. *Electronic Engineering*, 1995.

[10] P. Kission, H. Ding, and A. A. Jerraya. Structured Design Methodology For High-Level Design. In *Proc. of 31st Design Automation Conference*, San Diego, USA., June 1994.

[11] D. Knapp, T. Ly, D. MacMillen, and R. Miller. Behavioral Synthesis Methodology for HDL-Based Specification and Validation. In *Proc. of 32nd Design Automation Conference*, pages 286–291, June 1995.

[12] P.E.R. Lippens, J.L. van der Werf, and W.F.J Verhaeg et al. PHIDEO, A Silicon Compiler for High-Speed Algorithms. In *Proc. of EuroDAC/EuroVHDL*, Amsterdam., March 1991.

[13] G. De Micheli, D.C. Ku, F. Mailhot, and T. Truong. The Olympus Synthesis System. In *IEEE Design and Test*, pages 37–53, October 1991.

[14] P. Paulin, et al. High-Level Synthesis and Codesign Methods: An Application to a Videophone Codec. In *Proc. of EuroDAC/EuroVHDL*, Brighton, U.K., September 1995.

[15] Z. Peng. Synthesis of VLSI Systems with the CAMAD Design Aid. In *Proc. of 23rd Design Automation Conference*, San Diego, USA., June 1986.

[16] D.E. Thomas, E.M. Dirkes, R.A. Walker, J.V. Rajan, J.A. Nestor, and R.L. Blackburn. The system Architect's WorkBench. In *Proc. of 25th Design Automation Conference*, San Diego, USA., June 1988.

[17] S. Vernalde, P. Schaumont, I. Bolsens, H. De Man, and J. Fréhel. Synthesis of high throughput DSP ASICs using Application Specific Datapaths. *DSP & Multimedia Technology*, June 1994.

[18] Adrian Wise. *Introduction To Motion Picture Coding and the CCITT Algorithm.*, December 1989.

[19] W. Wolf, et al. Architectural Optimization Methods For Control Dominated Machines. In R. Camposano and W. Wolf, editor, *High Level VHDL Synthesis.* Kluwer Academic Publishers, 1991.