A METHODOLOGY FOR IP INTEGRATION INTO DSP SoC: A CASE STUDY OF A MAP ALGORITHM FOR TURBO DECODER

P. Coussy[1], D. Gnaedig[2,3], A. Nafkha[1], A. Baganne[1], E. Boutillon[1], E. Martin[1]

[1] LESTER LAB, UBS University - [2] TurboConcept - [3] ENST Bretagne

ABSTRACT

Re-use of complex Digital Signal Processing (DSP) coprocessors can be improved using IP cores described at a high abstraction level. System integration, that is a major step in SoC design, requires taking into account communication and timing constraints to design and integrate IP. In this paper we describe an IP design approach that relies on three main phases: constraints modeling, IP constraints analysis steps for feasibility checking and synthesis. Based on a generic architecture, the presented method provides automatic generation of IP cores designed under integration constraints. We show the effectiveness of our approach in a case study of a Maximum a posteriori MAP algorithm for Turbo Decoder.

1. INTRODUCTION

IP reuse is a key part of improving productivity for system-on-chip (SoC) designs. Unfortunately, the main problem when re-using RTL pre-designed component arises from their integration and more particularly from the communication architecture features. System integrator can use standard interface such as Virtual Component Interface proposed by VSIA [2] and Open Core Protocol proposed by the OCPIP [3]. However in DSP applications, in addition to the protocol aspects, a SoC designer has also to synchronize the components and to buffer data to ensure the system behavior and to meet timing constraints. Virtual components are indeed delivered at the RTL level that is, following the VSIA taxonomy, the highest abstraction level for synthesizable IP models (soft cores). However, such a description may be parameterizable, it relies on a fixed architectural model with very restricted customization capabilities. This lack of flexibility of RTL IPs is especially true for the communication unit whose sequence orders and timing requirements are set. IP cores are hence connected to the SoC bus through specific interfaces [4] or wrappers [5], that adapt the system communication features to the IP core requirements: SoC integrator must manage both IP execution requirements and integration constraints. This critical step requires a good modeling of both sets of constraints and techniques to design the interface module [6]. Unfortunately, this adaptation increases the final SoC area and also decreases system performance. In some cases, the I/O timing requirements cannot be respected due to the wrapper overhead and can cause the SoC design to fail.

In [8], we propose a SoC design methodology based on algorithmic IP core re-use. Based on high-level synthesis techniques under constraints, our approach aim to optimally synthesize the IP by taking into account the system integration constraints: application rate, technology, bus format, I/O timing properties specified by timing frames of transfers.

This paper is organized as follows: First in section 2 we give the problem formulation of the IP design under timing constraints. Section 3 reminds the main steps and formal models on which our design approach is based. In section 4, we show the effectiveness of our approach by presenting an integration case study of a MAP algorithm [12] into a turbo decoder architecture for decoding the error correcting codes called turbo codes [10].

2. PROBLEM FORMULATION

At the application level, the SoC designer describes the system as a set of communicating functions that specify what the system is supposed to do. The application level description is refined into a system level description when choosing implementation of functions to pre-designed or not SW/HW components. Performance and cost are then the primary issues. System level description allows to trade-off throughput against memory-size: architecture design of DSP applications indeed focus on (i) avoiding bottlenecks in the buses and I/O buffers for data-transfer, (ii) the cost of data storage and (iii) respecting strict timing constraints. According to both the application and the architecture design constraints, IP cores are selected from a database. SoC communication refinement then leads to refine and determine detailed integration constraints that must be supported by the IP core. They can be for instance: 1) the communication features used for data transfer (burst type and size...); 2) timing requirements for each data or group of data transfer (transfer delay estimation, date or timing frame of transfer) 3) architecture topology (point-to-point, shared bus).

Let us consider an application composed of three functions: demodulation, turbo decoding and MPEG2 (see Fig. 1).

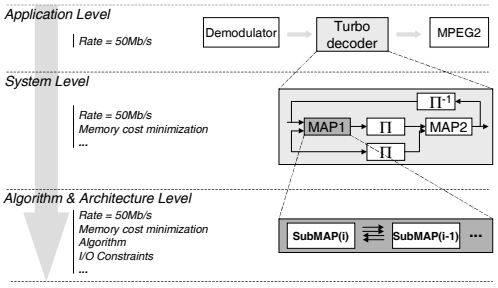


Fig. 1: SoC Design & Constraints refinement

The SoC integrator hence refines the IP integration constraints during the system design steps. These constraints therefore originate from the application requirements (data rate...), the architectural design criteria (area, power consumption, memory cost...), implementation choice (bus width, protocol...) and

technological library. IP design constraints have to be modeled to support all the previous features and to drive efficiently the IP synthesis. The integration constraints include all the communication features but also the application rate and the technological constraints.

A new design flow based on synthesis under constraints is needed to solve the problem of IP core reuse. This includes (1) modeling styles to represent system constraints and algorithm requirements, (2) analysis steps checking the feasibility and the consistency of the system constraints according to the algorithm ones and (3) methods and techniques for optimal synthesis of different IP core parts: processing unit (*PU*), memory unit (*MU*), control unit (*CU*) and communication unit (*COMU*).

3. DESIGN APPROACH OVERVIEW

Our methodology proposes to raise the abstraction level of IP synthetizable models by introducing the concept of behavioral IP [9], described as an algorithm and specified using HDL language such as SystemC. Starting from the system description and its architecture model, the integrator, for each bus or port that connects the IP to SoC components, refines and specifies I/O protocols, data sequence orders and transfer timing information.

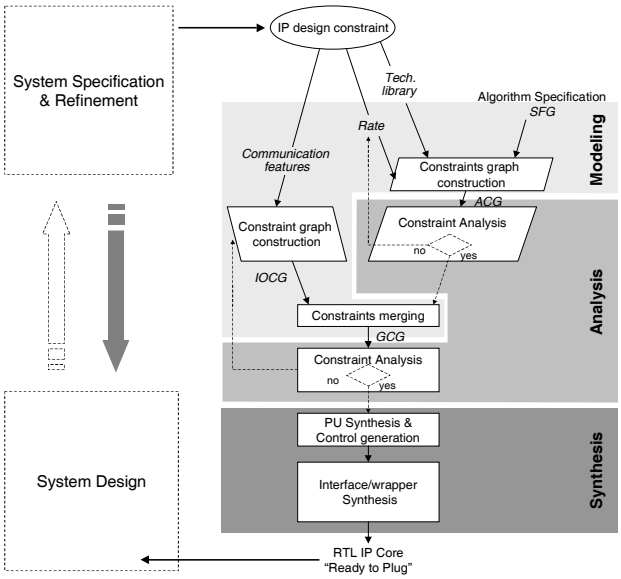


Fig. 2: Design Approach

The virtual component specification is modeled by a Signal Flow Graph *SFG*. The difference between a *SFG* and a Data-Flow Graph *DFG* consists in the *delay* operation used in DSP modeling to express the use of a data computed in the previous iteration of the algorithm. This intermediate *SFG* representation is produced by the compilation phase in the behavioral synthesis flow. In a first step, we generate an Algorithmic Constraint Graph *ACG* from the operator latencies and the data dependencies expressed in the *SFG*. The latencies of the operators are assigned to operation vertices of the *ACG* during the operator's selection step of the behavioral synthesis flow. Having described the IP behavior and the IP design constraints in a formal model, we analyze the feasibility between the rate, the data dependencies of the algorithm and the technological constraints (see Fig. 2). This analysis checks the *ACG* for positive cycles to ensure that the constraint graph is feasible without considering input arrival dates.

In order to support the features of communication architectures specific to DSP application, we extend the *SIF* model *(Sequential Intermediate Format* [7]*)*. This new formal model named *IOCG (IO Constraint Graph)* supports expressing of integration constraints for each bus (id. port) that connects the IP to the SoC components. It allows (1) to specify transfer related timing constraints such as ordered transactions, relative timing specification, min-max delay, (2) to include architecture features and (3) to express non determinism in the data transfer time.

Finally we generate a Global Constraint Graph *(GCG)* by merging the *ACG* with the *IOCG* graph. Merging is done by mapping the vertices and associated constraints of *IOCG* onto input and output vertices set of *ACG*. A minimum timing constraint on output vertices (earliest date for data transfer) of the *IOCG* are transformed into the *GCG* in maximum timing constraints (latest date for data computation/production).

With the formal description of the set of constraints, we analyze the consistency of the IP design constraints according to the algorithm ones. Consistency analysis refers to the dynamic behavior of the *GCG* graph.

The entry point of the IP core design task is the global constraint graph *GCG*. This design step relies on the synthesis of different functional units of the IP core: *PU*, *MU*, *CU* and *COMU*.

Further information about formal models, analysis and synthesis steps, which are not described here, can be found in [8].

4. IMPLEMENTATION OF THE MAP ALGORITHM

We apply our design methodology on the design of the *maximum a posteriori* (MAP) algorithm which is used to provide a soft-in soft-out (SISO) decoder of a convolutional code. Two SISO decoders are used in turbo-decoders [10] to iteratively refine reliability of the decoded symbols.

4.1. MAP Algorithm Overview

The MAP algorithm, also called the Forward-Backward algorithm (FB), provides for each transmitted symbol u_k, $k = 0$. N-1 a soft output $L^e(u_k)$ called the extrinsic information. It is computed by using the received values of the channel $Y=\{y_k\}_{k=0..N-1}$ and the a priori information $L^a(u_k)$ provided by the other decoder, which is its extrinsic soft output. The soft estimate $L^e(u_k)$ is computed by exhaustively exploring all possible paths in the trellis using a forward recursion and a backward recursion. To simplify hardware implementation, we use the Max-Log-MAP algorithm described in the logarithmic domain [11]. This algorithm consists of three steps (Fig. 3(a)).

- Forward recursion. The forward state metrics A_k are recursively calculated using the symbols in an increasing order from 0 to N-1. The branch metrics of the forward recursion are computed (BMC) with the received symbols and the *a priori* information. The forward state metrics are stored in an internal memory in order to be used to compute the soft output.
- Backward recursion. The backward state metrics B_k are recursively computed using the symbols in a decreasing order from *N-1* to *0*.

- Soft-Output Computation. The soft output for each symbol at time k is computed by using the backward state metric B_k and the corresponding forward state metric A_{k-1} read from the memory.

The initial state metrics of the forward and backward recursions A_0 and B_{N-1} are provided as an input of algorithm. The final metrics A_{N-1} and B_0 are part of its outputs. More details on recursions initializations can be found in [12] and related references.

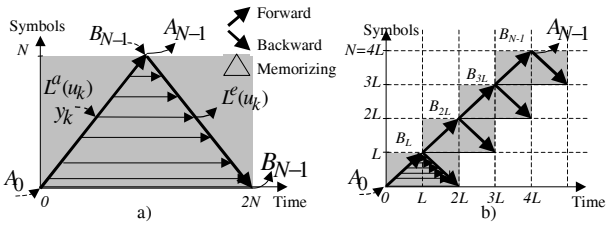


Fig. 3: Graphical representation of the (a) FB and the (b) SW-FB algorithms

4.2. Algorithm Architecture Matching (AAM)

We described the classical FB algorithm. We now focus on its implementation by matching the algorithm to the architectural architectural constraints (memory, latency, throughput,...).

Due to its schedule, the backward recursion cannot start before the end of the forward recursion, and hence, the maximal latency of the FB algorithm is *2.N*. Moreover, *N* state metrics vectors need to be memorized, which requires a large amount of memory. To reduce both the memory requirements and the latency, a sliding window algorithm (SW) is used [13]. Using the same graphical representation as in [12], our modified SW-FB algorithm with windows of size *L* symbols is depicted in Fig. 3. In this figure, the horizontal axis represent time, with units of a symbol period. The vertical axis represents the received symbol. For each window working on symbols αL to $(\alpha+1)L$, a forward recursion with memorization is performed from the initial state metric given by the previous window. For the first window the initial state metrics are initialized with A_0. The backward recursion is then performed on the same window from an initial state metric $B_{(\alpha+1)L}$ given by the previous iteration [12]. At the first iteration, the backward state metrics are initialized with the all-zero vector. The windows are then processed sequentially by a single elementary component called *SubMAP* and performing a forward-backward algorithm on *L* symbols. This SW-FB algorithm requires to store *L* state metrics vectors and its maximal latency is reduced to *2.L* symbols.

We now present the description of the elementary *SubMAP* component, used to build the complete SW-FB algorithm. The simplest and natural solution to implement the *SubMAP* component consists in using the same symbols αL to $(\alpha+1)L$ for the forward and backward recursions. This solution corresponds to the natural description of the algorithm in a classical sequential language (C for example). It implements *horizontally* (same symbols for the forward and backward recursions) the algorithm and is therefore denoted *SubMAP-H*. This component basically implements one forward and one backward processor. With the scheduling represented in Fig. 3.b, the SW-FB algorithm requires two *SubMAP-H* components working in parallel in staggered rows: the first one uses the symbols αL to $(\alpha+1)L$, while the other uses the symbols $(\alpha+1)L$ to $(\alpha+2)L$. However, this scheduling requires to store $2.L$ forward state metrics and to duplicate the forward and backward processors. The amount of memory be divided by two by using only one SubMAP-H component with the scheduling of Fig. 4.a, but the real time constraint is not met anymore. The timing behavior and *IOCG* of the *SubMAP-H* component are shown in Fig. 5 and Fig. 6, respectively.

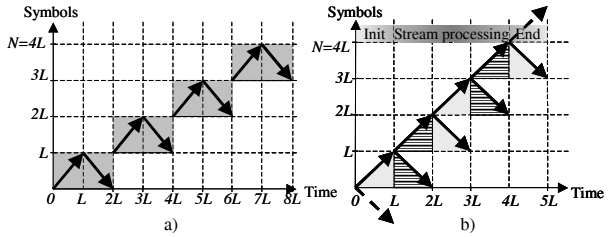


Fig. 4: *SW-FB* scheduling with one (a) SubMAP-H or one (b) *SubMAP-V* component

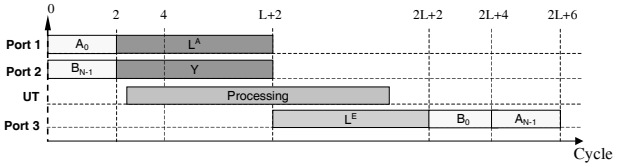


Fig. 5: Timing Behavior of the *SubMAP-H* component

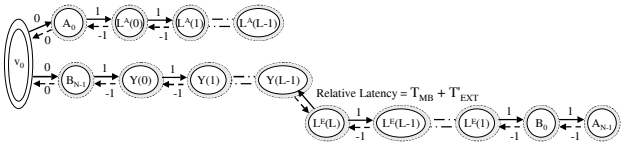


Fig. 6: *IOCG* of the *SubMAP-H* component

In order to overcome the problem of memory duplication and to meet the real-time constraint, the elementary *SubMAP* is modified by specifying the parallel execution of the two recursions. It implements *vertically* the algorithm (*SubMAP-V*) by working on two consecutive windows as shown in Fig. 4.b: the forward recursion works on the current window with the symbols αL to $(\alpha+1)L$ while the backward recursion works on the previous window with symbols $(\alpha-1)L$ to $\alpha.L$. In this case, the backward recursion is performed on data stored in the internal memory of the *SubMAP-V* component.

The timing behavior of the *SubMAP-V* component is depicted in Fig. 7. The required latency of the *SubMAP-V* component corresponds to the delay of the critical path to produce an extrinsic information: the first soft output is produced after 4 clock cycles.

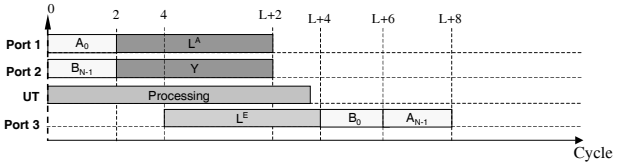


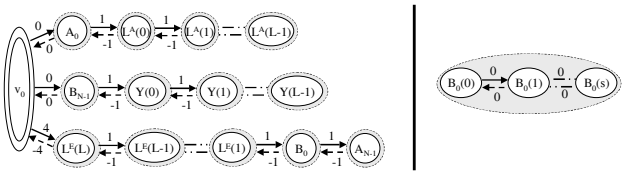
Fig. 7: Timing behavior of the *SubMAP-V* component

4.3 Synthesis Results

We described in the previous section how a SoC designer chose a specific algorithm following an AAM approach. We present now the results of the synthesis under constraints, obtained using the HLS tool *GAUT* [15], [16]. The convolutional code used for this experience is a duo-binary circular recursive systematic

convolutional code used in the DVB-RCS standard [14]. The trellis is composed of 8 states with 4 branches leaving each state. The SW-FB is implemented with *L*=32.

Fig. 8(a) represents the *IOCG* constraint graph used for the MAP algorithm synthesis. I/O timing requirements are depicted in Fig. 7. Hierarchical vertices (see [8]) described in Fig. 8(b) are used to express simultaneous arrival date of input data. For instance, the 4 backward state metrics, are simultaneously transmitted (i.e. the 8 metrics are transferred in 2 steps). Hierarchical vertices are used for all input and output vertices of the *IOCG*. Notice that output production dates are specified -contrarily to Fig. 6, relatively to the vertex v_0 that represents the start time.



(a) **(b)**

Fig. 8: *IOCG* of the *SubMAP-V* component

The *SFG* includes 95210 edges and 44048 vertices, divided in 23168 operations and 20880 data. The source file describing the algorithm uses 128 lines of behavioral VHDL code to specify the entire *SubMAP-V* computation. Table 1 presents the *SFG* operation vertices classified by function types.

Inv.	Add.	Sub.	Comp.	Data vertices
8448	11520	2304	896	20880

Table 1: *SFG* Composition

The VHDL RTL file, describing the *SubMAP-V* architecture includes 2469272 lines (20 thousand times more than the source code). The complete generation took only 35 minutes on a 900 MHz SunBlade2000, with 1G RAM and running Solaris8. This time is composed of 4 minutes to parse, compile and generate the intermediate representation *ACG*; 2 minutes for the scheduling and 29 minutes to generate the control unit, processing unit and their respective VHDL files. Notice that most of the time was spent in the RTL file generation. RTL description has been simulated using Modeltech's ModelSim5.5c simulator and functional validation has been completed by comparing RTL results to the C model ones.

The design results are shown in Table 2, for a 200MHz clock frequency using a technological library in which the latency of the adder, the subtractor and the comparator is one cycle. The final architecture includes 64 inverters, 96 adders, 32 subtractors and 52 comparators. The FSM controlling the processing unit includes 144 states and 8 bit width instructions. The generated RTL architecture includes 996 registers: this corresponds to an average of 3 registers by operator (notice that our architectural model imposes a minimum of 2 registers for 1 operator [8]).

FSM states	Inv.	Add	Sub.	Comp.	Registers UT
144	64	96	32	52	996

Table 2: *SubMAP-V* under constraint synthesis results

Table 3 depicts the awaited amount of operators for the *SubMAP-V* component. The number of components given by the synthesis are in accordance to our estimations. Less comparators and far more inverters are required by our architecture because of component re-using and lack of reduction of regular expressions, respectively. We are intensively working on this last point that will have an impact on the total amount of operators and registers since it can drastically reduce the number of component for the computation of the branch metrics.

Computation	Inv.	Add.	Sub.	Comp.	
BMC	4	32	16	0	
Forward		32		24	
Backward		32		24	
Extrinsic	8	32	16	28	
	12	**96**	**32**	**72**	**Total**

Table 3: Awaited amount of operators for *SubMAP-V*

5. CONCLUSION

In this paper, a methodology for IP integration into DSP SoC has been presented. This approach, that relies on constraints modeling, constraints analysis and synthesis, help the designer to efficiently implement complex applications. Based on an integration case study of a MAP algorithm into a turbo decoder architecture, an illustration of our design flow has been presented. This experiment has shown the interest and the effectiveness of our approach to quickly design complex DSP applications by using behavioral IP cores.

Acknowledgements

These works have been realized within the French RNRT Project ALIPTA.

6. REFERENCES

[1] H. Chang, et Al., *"Surviving the SoC revolution, A guide to Platform-Based Design"*, Kluwer academic publishers, 1999

[2] Virtual Socket Interface Alliance, http://www.vsi.org

[3] OCP-IP International Partnership, http://www.ocpip.org/

[4] D. Hommais, F. Pétrot, I. Augé, *"A Practical Toolbox for System Level Communication Synthesis"*, In Proc. of CODES,2001

[5] G. Nicolescu, et al., *"Validation in a Component-Based Design Flow for Multicore SoCs"*, In Proc. of ISSS, 2002.

[6] P. Coussy, A. Baganne, E. Martin, *"A Design Methodology for IP Integration",* In Proc. of ISCAS, 2002.

[7] D. Ku and G. De Micheli, *"Relative Scheduling Under Timing Constraints: Algorithms for High-Level Synthesis of Digital Circuits"*, IEEE Trans. CAD/ICAS, vol. 11, pp. 696-718, June 1992

[8] P. Coussy, A. Baganne, E. Martin, *" Communication and Timing Constraints Analysis for IP Design and Integration ",* In Proc. of IFIP VLSI-SOC, 2003.

[9] G. Savaton, P. Coussy, E. Casseau, E. Martin *"A Methodology for Behavioral Virtual Component Specification Targeting SoC Design with High-Level Synthesis Tools",* In proc. of FDL, 2001.

[10] C. Berrou, A. Glavieux and P. Thitimajshima, *"Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes", Proc. ICC'93*, Geneva, Switzerland, pp. 1064-1070, 1993.

[11] P. Robertson, et Al., "A Comparison of Optimal and Sub-optimal Decoding Algorithm in the Log Domain", *Proc ICC*, 1995.

[12] E. Boutillon, et Al., *"VLSI Architectures for the MAP Algorithm"*, IEEE Trans. On Communications, vol51, No.2, 2003.

[13] H. Dawid and H. Meyr, *"Real time algorithms and VLSI architectures for soft-output MAP convolutional decoding,"* in Proc. PIMR'95, vol.1, pp.193-197, 1995.

[14] C. Douillard, *al*., *"The Turbo Code Standard for DVB-RCS"*, Proc. 2nd Int. Symp. on Turbo Codes and Rel. Topics, Sept. 2000

[15] A. Baganne, J.L Philippe, E. Martin *"A Formal Technique for Hardware Interface design"*, In. IEEE Trans. On Circuits And Systems, Vol.45, N5, 1998,

[16] GAUT - HLS Tool for DSP, *http://lester.univ-ubs.fr:8080/*