Reed-solomon behavioral virtual component for communication systems

Bertrand Le Gal, Emmanuel Casseau, Chirstophe Jégo, Nathalie Le Héno, Eric Martin

REED-SOLOMON BEHAVIORAL VIRTUAL COMPONENT FOR COMMUNICATION SYSTEMS

B. Le Gal[1], E. Casseau[1], C. Jego[2], N. Le Heno[3], E. Martin[1]

1. LESTER Laboratory
UBS University FRANCE
{First-Name.Surname}@univ-ubs.fr
http://lester.univ-ubs.fr:8080

2. ENST Bretagne, FRANCE
christophe.jego@enst-bretagne.fr
http://www.enst-bretagne.fr

3. Turbo Concept SAS, FRANCE
nathalie.leheno@turboconcept.com
http://www.turboconcept.com

ABSTRACT

In this paper, we focus on the design of a communication system based on reusing IP cores. We consider that traditional methods for hardware design at the RT level suffer from heavy limitations that prevent them from efficiently addressing the algorithmic complexity and the high flexibility required by the various application profiles. We propose to raise the abstraction level of the specification and introduce the notion of architectural flexibility of an IP core by benefiting from the emerging high-level synthesis tools. Our method has been successfully applied to the design of a Reed-Solomon (RS) decoder IP core, targeting the DVB-DSNG digital video broadcasting standard. We are able to generate a variety of RS decoder architectures, with varying hardware complexity and computation speed, from a single behavioral-level *VHDL* specification.

1. INTRODUCTION

As VLSI technology advances continuously, the complexity of hardware components has quickly increased and makes the design of SoC possible. Design teams are thus currently faced with increased device sizes and time to market constraint. As it is no longer practical to manage million gate systems at the gate-level, design by reuse - i.e. the use of intellectual property components (IPs) or so called virtual components (VCs) - seems to be the only way to cope [1]. However, the increasing complexity of the applications and of the IPs themselves require methodologies to be used, from the side of IP providers as well as IP integrators.

Our approach consists in raising the abstraction level of IP cores by benefiting from the emerging High-Level Synthesis (HLS) tools. Our methodology aims at facilitating design, validation and synthesis of IP cores at the behavioral level, and exploits functional as well as architectural flexibility by allowing straightforward instantiation of various RTL architectures – fulfilling various sets of functional parameters and performance constraints such as gate count, speed, etc. – starting from a single high-level description of the behavior.

This methodology is currently developed in the ALIPTA[1] (Algorithmic Level IP for Telecom Applications) project.

In this paper, we detail the design and synthesis of a Reed-Solomon decoder for channel decoding. This decoder is one of the blocks of the DVB-DSNG plate-form we develop. A behavioral description of this core has been written and synthesized using a HLS tool. From a single behavioral description, a variety of architectures were generated, spanning a wide range of performance, including the constraints of the DVB-DSNG standard we target.

This paper is organized as follows: in section 2, we introduce the notion of behavioral virtual component and provide a brief introduction to high-level synthesis. Section 3 is a summary of the Reed-Solomon decoder algorithm. In section 4, we detail the followed strategy for designing a Reed-Solomon IP core at the behavioral level. Finally, section 5 provides high-level synthesis results that demonstrate the flexibility of our IP core.

2. BEHAVIORAL VIRTUAL COMPONENTS

2.1 Virtual Components

A hardware Virtual Component can be defined as a hardware block with well-identified functionality, ready to be inserted into the design flow of an integrated system. The main purpose of IP-based design is to achieve shorter time-to-market and higher reliability by reusing pre-designed and pre-verified functional blocks instead of re-designing them from scratch [2].

Today, IP design, IP trade and IP integration still suffer from a lack of applicable methodologies for fast design of complex cores, fast selection of cores from IP delivery systems, fast insertion of a core into a new system and effective protection of the intellectual property. Recent trends in system-level design recommend that an IP core

1 this project is supported by the French National Network for Telecommunication Research (RNRT). Industrials : TNI-Valiosys, THALES, SACET, Turbo-Concept; Academics : LESTER-UBS and ENST Bretagne.

be delivered with a system-level simulatable model, written in a standard system-level language such as SystemC, in order to accelerate system refinement and verification. From the synthesis point of view, design-for-reuse methodologies and standards [3] still consider "soft" IP cores as the highest abstraction level for synthesizable IP models. Such models are delivered in the form of Register-Transfer Level (RTL) descriptions in a suitable Hardware Description Language (e.g. VHDL or Verilog). Though an RTL description may be parameterized – e.g. by making use of VHDL constructs such as "generic" parameters and "generate" statements – it relies on a fixed architectural model with very restricted customization capabilities.

Our methodology effort aims at accelerating the transition between IP specification at the algorithmic and system levels on the one hand, and RTL architecture generation on the other hand. We propose to raise the abstraction level of IP synthesizable models by introducing the concept of "behavioral" IP core [4]. Such a core is specified in a behavioral (algorithmic-like) fashion in a high-level language (VHDL, C, SystemC, etc.) and relies on High-Level Synthesis (HLS) tools for architecture generation. The efficiency of our approach relies on the ability of these tools to automatically and quickly instantiate a wide variety of RTL architectures – corresponding to various functional parameter sets and fulfilling user-defined optimization constraints – from a single high-level description of the behavior.

2.2 High level synthesis

HLS is analogous to software compilation transposed to the hardware domain: the source specification is written in a high-level language that models the behavior of a complex hardware component; an automatic refinement process allows to map the described behavior onto a specific technology target depending on optimization constraints.

A typical HLS tool performs four main tasks: (1) source specification analyze (identify computations); (2) hardware resources selection and allocation for each kind of operation; (3) operations scheduling; (4) optimized architecture generation, including a datapath and a control finite-state machine. HLS is a constraint-based synthesis flow: hardware resources are selected from technology-specific libraries of components (arithmetic and logic units, registers, multiplexors) where components are characterized in terms of gate count, delay, power consumption, etc; resource selection/allocation and operation scheduling can be constrained to limit hardware complexity (i.e. the number of allocated resources) and reach a given computation speed (given as the number of control steps for operation scheduling).

Due to its high abstraction level, a behavioral description for HLS can be made customizable through functional parameters. Each set of supported parameter values and synthesis constraints allows to instantiate a different dedicated architecture that will fulfill specific functional requirements and achieve specific performance. As a result, HLS tools can be seen as a relevant approach for designing and reusing highly flexible IP cores.

The HLS tool used in the ALIPTA project is called GAUT[2]. It is a pipeline architectural synthesis tool, dedicated to Signal and Image Processing applications under a real time execution constraint. The input specification language is currently a subset of behavioral VHDL; SystemC language is expected in 2004. Synthesis leads to the generation of a structural and functional VHDL description of the designed architectures. This VHDL file is a direct input for commercial, logical synthesis tools like Quartus from Altera and ISE/Foundation from Xilinx. GAUT allows the designer to check its refinements in each level of the conception flow by permitting the use of testbenches written for higher level specification, in order to validate the generated architectures.

3. REED-SOLOMON ALGORITHM OVERVIEW

Digital Satellite News Gathering (DSNG) and Digital Video Broadcasting applications by Satellite (DVB_S) consist in point-to-point or point-to-multipoint transmissions, connecting fixed or transportable up-link terminal and receiving stations, not intended to be received by the general public. Maximum cohesion with DVB-S1 is maintained, such as concatenated error protection strategy based on Reed-Solomon coding, convolutional interleaving and inner convolutional coding [6]. The system transmission frame is synchronous with the MEPG-2 multiplex transport packets.

Error correcting codes (channel coding) are one of the solutions available to improve the digital communication quality. The purpose of channel coding is to introduce, in a controlled manner, some redundancy in the binary information sequence to overcome the effects of noise and interference encountered during the transmission through the channel. Reed-Solomon codes are block error correction codes with burst error-correcting capabilities that have found widespread use in storage devices and digital communication systems. In particular, concatenated coding employing an inner convolutional code combined with a Reed-Solomon outer code constitutes an attractive scheme that is commonly encountered in many applications, and in particular in the DVB-DSNG standard that the ALIPTA project targets.

2 GAUT tool is downloadable after a free registration on LESTER web site http://lester.univ-ubs.fr:8080

The channel coding scheme in the DVB-DSNG standard is based on a concatenated code composed of an r=1/2, k=7 convolutional code, a convolutional interleaver of depth 12 and a (204,188) Reed-Solomon code. The (204,188) RS code is a punctured version of the RS(255,239) working on bytes. It is able to correct up to 8 erroneous bytes per received packet of 204 bytes. Decoding of Reed-Solomon codes is based on Galois field arithmetic. The RS(204,188) operates in $GF(2^8)$ whose generator polynomial $P(x)=x^8+x^4+x^3+x^2+1$ is defined in the standard [6].

The Reed-Solomon decoder consists in five major blocks as depicted in figure 1. One first calculates the syndrome values (16 in this case), which represent the sequence of errors in the frequency domain. Then one solves the key equation by determining two polynomials, the error locator polynomial $\sigma(x)$ and the error evaluator polynomial $\omega(x)$ from the syndrome values. This calculation is realized in an iterative way by the Berlekamp-Massey algorithm. 16 iterations are actually required in this case. The roots of $\sigma(x)$ are computed using the Chien search algorithm in order to provide the errors location in the received word. The corresponding magnitudes of the errors are then calculated with $\omega(x)$. The received word is finally corrected using the two aforementioned information. A mathematical description of the algorithms can be found in [7,8]

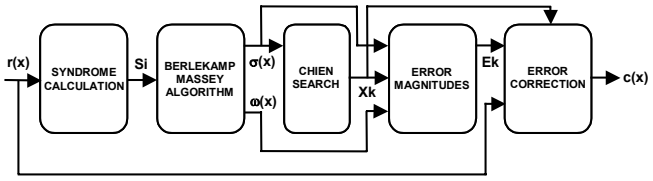

Fig. 1: RS decoder block diagram

4. REED-SOLOMON DECODER AT THE BEHAVIORAL LEVEL

First, an estimation step of the functional complexity of the Reed-Solomon algorithm has been done before the behavioral specification. The interest is to list the different operations of the algorithm to adapt the architectural synthesis library. This library contains the characteristics of components that come from logic synthesis. Different kinds of component can be defined: standard, multi-function, pipeline or macro-function. The use of dedicated components (multi-function and/or macro-function) can be noticed during the estimation step. In our study, we have estimated the Reed-Solomon algorithm complexity for two cases (figure 2). In the first case, only standard operations have been used and the complexity is about 16000 operations. Two operation types are necessary: Galois field computation (addition, inversion and multiplication) and control (comparison and variable control). A second case with the multi-function operation MAC (multiplication-accumulation) has been considered. Then, the complexity is about 11000 operations. In fact, the total number of MAC operation is 5084 in the targeted RS algorithm. This particularity is interesting because it allows us to decrease the operation complexity by 30 %.

In this study, a plate-form with the Excalibur FPGA from Altera is used. For this reason, the GAUT library has been defined according to the EPXA family. All calculations work on polynomials with coefficients in the Galois field $GF(2^8)$. So, the library is composed of components with a same size of 8 bits. Moreover, the 8 bit addition component in the Galois field is composed of only 8 XOR gates. Consequently, the characteristics of the components MAC_{GF} and $Mult_{GF}$ are similar. The use of the multi-function component MAC_{GF} that allows us to decrease the operation complexity has a few material cost. Finally, the inverse component in the Galois field Inv_{GF} is defined by a table that needs no logic element but a 2048 bit memory bloc.

The behavioral VHDL specification of the RS algorithm has been realized and compared with a C reference specification to validate our description. For this work, we have written a VHDL package that contains the functions that correspond to all the components of our architectural synthesis library.

	case 1					case 2					
	Galois field computation			control		Galois field computation				control	
	Add_{GF}	$mult_{GF}$	inv_{GF}	fc_di	comp	add_{GF}	$Mult_{GF}$	Inv_{GF}	MAC_{GF}	fc_di	comp
syndrome calculation	3248	6512	0	0	16	0	3264	0	3248	0	16
Berlekamp-Massey algo	320	502	15	13	254	0	182	15	320	13	254
Chien search	1429	3060	0	0	204	1	1632	0	1428	0	204
error magnitudes	88	192	8	0	0	0	104	8	88	0	0
error correction	8	0	0	0	204	8	0	0	0	0	204
total	5093	10266	23	13	678	9	5182	23	5084	13	678
	16073					10989					

Fig. 2: Complexity (number of operations) of the RS decoding algorithm main functions

5. REED-SOLOMON DECODER SYNTHESIS

Behavioral synthesis of the Reed-Solomon decoder has been performed using the GAUT tool. Several architectures have been generated using different bit rate values (latency constraint). Given results are based on EPXA Altera FPGA technology with a 10 ns clock period, which is the maximum estimated latency of the sequential operators in the technology library.

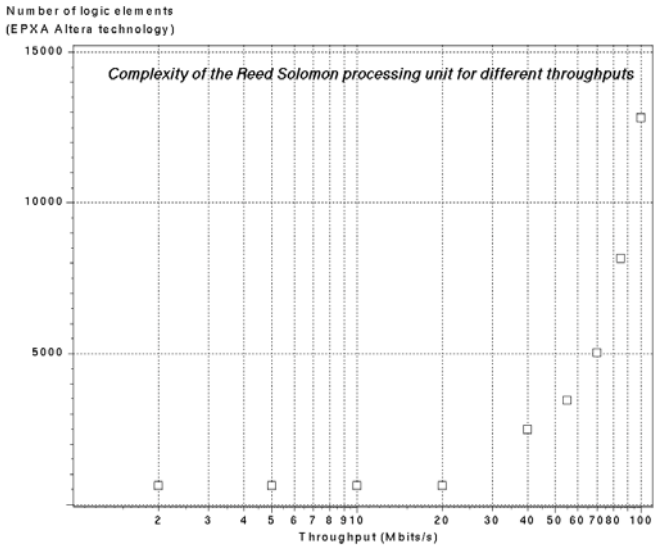

Fig. 3: RS architecture complexity for different throughputs

The DVB-DSNG standard allows transmissions from 1.5 Mbits/s to 72 Mbits/s. The synthesis of the behavioral RS decoder virtual component has been done from 2 Mbits/s to 100 Mbits/s. Figure 3 gives the complexity (number of logic elements) of the processing unit of the decoder. The complexity is about 650 logic elements until 20 Mbits/s. It increases until 12500 logic elements for 100 Mbits/s. In fact, a low computation speed allows the HLS tool to reuse arithmetic components for operations scheduled in separate clock cycles. This results in a significant reduction of the allocated hardware from 13 Galois field operators at 100 Mbits/s to only 3 when computation time reaches 20 Mbits/s. The HLS tool generates the same architecture for 2 Mbits/s up to 20 Mbits/s speed constraint (only 1 allocated operator for each kind of operation).

Figure 4 shows the floorplan of the RS decoder processing unit with a 40 Mbits/s throughput constraint. This constraint has been chosen according to the ALPITA project that targets the integration of the different functions of a first receiver at 40 Mbits/s. The RTL VHDL description of the RS decoder generated by the architectural synthesis tool GAUT has been synthesis and then integrated in an EPXA10 device (ALIPTA project target) by the Altera Quartus tool.

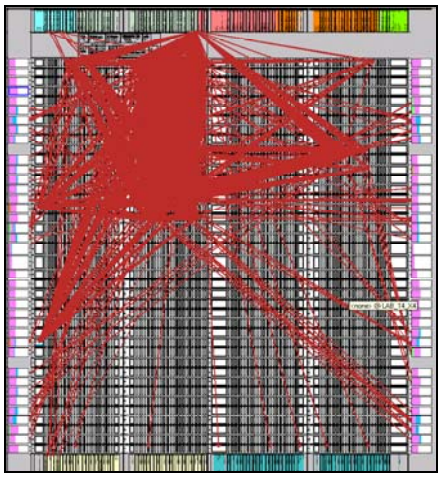

Fig. 4: Processing unit floorplan of RS decoder (EPXA10 device)

CONCLUSION

The ALIPTA behavioral virtual components based approach has been applied to the design of a reusable virtual component for Reed-Solomon decoding. A High-Level Synthesis tool has been used for generating a set of architectures fitting various application profiles starting from a single high-level description of the behavior. We thus quickly obtained a dedicated architecture for all of the throughputs that the DVB-DSNG standard covers.

Further efforts in our behavioral Reed-Solomon virtual component design will focus on exploiting the possibility on the integrator's side to parameterize the I/O according to the system communication constraints.

REFERENCES

[1] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, L. Todd, Surviving the SOC Revolution - A Guide to Platform-Based Design, Kluwer Academic Publishers; 1999.

[2] M. Keating and P. Bricaud, Reuse Methodology Manual for System-On-A-Chip Designs, Kluwer Academic Publishers, 240 pp., June 1998.

[3] VSIA Virtual Socket Alliance Interface, http://vsi.org.

[4] E. Casseau, SoC design using behavioral level virtual components, ICECS 02, IEEE International Conference on Electronics, Circuits, and Systems, Dubrovnik, Croatie, 15-18 septembre 2002, pp. 497-500.

[5] D. D. Gajski, N. D. Dutt, Allen C-H. Wu, Steve Y-L. Lin, High-Level Synthesis: Introduction to Chip and System Design, Kluwer Academic Publishers, Boston, MA, 1992.

[6] Standard ETSI ETS 300 429, Digital Video Broadcasting (DVB) ; Framing structure, channel coding and modulation for cable systems , December 1994.

[7] WW.Peterson, E.J Weldon, Jr, *Error Correcting Codes*, Cambridge, Mass., MIT Press, 1961.

[8] R. Blahut, *Theory and Practice of Error Control Codes*, Addison Publishing Company, 1983.