

Electronic System Level to Hw/Sw Design Flow

S. HUET
LESTER Laboratory
UBS University
BP 92116
56321 LORIENT Cedex
France
+33 (0)2 97 87 45 60
shuet@iuplo.univ-ubs.fr

P. BOMEL
LESTER Laboratory
UBS University
BP 92116
56321 LORIENT Cedex
France
+33 (0)2 97 87 45 60
pierre.bomel@univ-ubs.fr

E. CASSEAU
LESTER Laboratory
UBS University
BP 92116
56321 LORIENT Cedex
France
+33 (0)2 97 87 45 60
emmanuel.casseau@univ-ubs.fr

B. Le GAL
LESTER Laboratory
UBS University
BP 92116
56321 LORIENT Cedex
France
+33 (0)2 97 87 45 60
bertrand.legal@univ-ubs.fr

O. PASQUIER
IREENA
Polytech'Nantes
BP 50609
44306 Nantes Cedex 3
France
+33 (0)2 40 68 30 35
olivier.pasquier@polytech.univ-nantes.fr

Abstract

The increasing needs for higher data rates associated with mobility constraints motivates the development of the fourth generation of wireless systems. In this paper we focus on rapid prototyping of such emerging software radio applications. To address this problem, we propose a five steps methodology based on both an Electronic System Level (ESL) design tool and a High Level Synthesis (HLS) tool. The methodology has been applied on the design of a Multiple Input Multiple Output (MIMO) system using orthogonal space time codes.

Key words: rapid prototyping, software radio, system level design, high level synthesis

1 Introduction

A typical Digital Signal Processing (DSP) application design usually starts from the specifications of the DSP algorithms and ends to their physical Hardware/Software implementation in a top down fashion. The increasing computational complexity of new software radio systems makes this classical design process time consuming and consequently incompatible with an efficient design space exploration. In order to reduce time to real world tests of new software

radio applications, our approach consists in automatically refining high abstraction level models. The five steps methodology we proposed is based on both an Electronic System Level (ESL) design tool and a High Level Synthesis (HLS) tool. This project named PALMYRE involves multidisciplinary scientific teams working on new DSP algorithms. In this context DSP, HLS and Codesign teams collaborate around a modular rapid prototyping platform from Sundance [1] embedding several DSPs from Texas Instrument [2] and FPGAs from Xilinx [3]. This paper is organised as follows. In section 2 we detail each steps of our design flow. Section 3 demonstrates its flexibility on the design of a Multiple Input Multiple Output (MIMO) system using orthogonal space time codes.

2 The design flow

System Level Design (SLD) flows are promising approaches since they aim at automating the refinement of the specifications down to the implementation [4]. In this paper we put an SLD flow into practice with the view to build rapid prototypes of software radio systems on our Sundance platform. Even though we target a specific domain of application this flow can be extended to design a wide range of electronic systems.

As shown in Fig 1, our methodology is structured on the basis of 5 main steps [5]. These five steps are linked by three models: the Functional Architecture model, the Platform model and the Macro-Architecture model as presented in Fig. 2. This methodology is now supported by CoFluent Studio [6] which is the ESL design tool we use. The next 5 sub sections detail each step of the methodology.

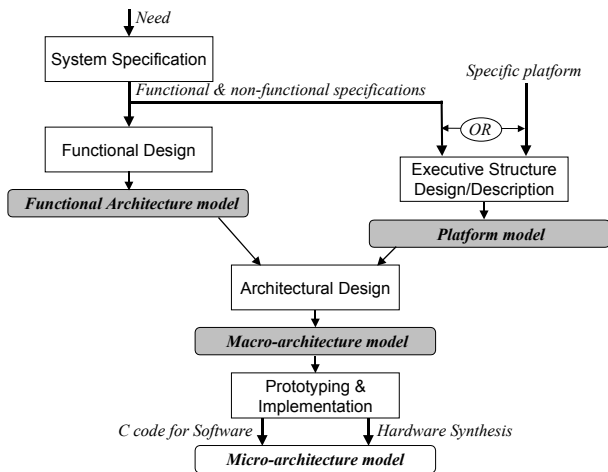


Fig. 1: Methodology steps

2.1 System specification

The need is first analyzed in the System Specification Step. This step provides the System Specifications which include both the functional and non-functional (technical for example) specifications.

These System Specifications (mainly functional) are used in the Functional design step.

2.2 Functional Design

It consists in defining the Functional solution by extracting and exploring the parallelism of the specifications. This Functional solution is depicted according to the Functional Architecture model. The solution is graphically described by a set of functions interconnected with three kinds of relations (Fig.2 a): events to model synchronisation, shared variables to model data sharing and communication ports to model producer/consumer like relations. A function is described by a behaviour which can be refined by a set of concurrent functions linked by relations.

This model is free of any details of implementation but could be simulated [7], [8]. So it is possible to verify the system functionality and to test several descriptions of the system. Some reference models are selected through this iterative validation process and will be considered as golden references for the next design steps.

2.3 Executive Structure Design/Description

The next step, Executive Structure Design/Description, aims at describing the physical organisation of the platform on which the application will run.

The platform model (Fig. 2 b) is based on Processing Elements interconnected by Communication Nodes. In the context of the PALMYRE project, our rapid prototyping platform is modelled as a set of DSPs from TI (c62, c67, c64) and FPGAs from Xilinx (Virtex E1000) collaborating through a high speed communication network composed of 20 Mbytes/s Sundance's comm.-ports and 200 Mbytes Sundance's SDB links.

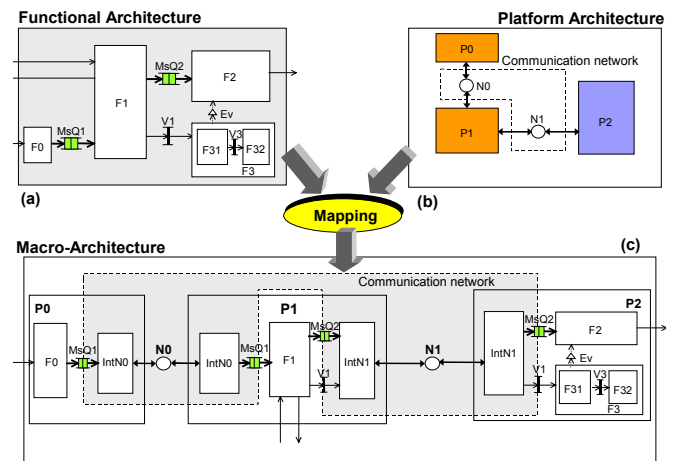


Fig. 2: Description models

2.4 Architectural Design

The fourth step, architectural design, consists in mapping the Functional Architecture to the Platform Architecture. The Functional Architecture model is mapped to the Platform Architecture Model, that is to say Functions to Processing Elements and Relations to Communication Nodes. The resulting Macro-Architecture model is heterogeneous: it contains both functional and

executive elements. In this model are also introduced interfaces functions to map Functional Relations to Communication Nodes. This is represented by the “Communication network” on Fig. 2 c.

Like the Functional model, the Macro-Architecture model, also named virtual prototype, can be simulated [9], [10]. This is very important for architecture exploration and performance estimation since this model describes both the behaviour of the system and the hardware on which it runs. Besides, this model is a high abstraction level description. This allows a rapid simulation and thus to explore several platform configurations.

2.5 Prototyping and implementation

Finally, the Macro-Architecture is refined to the Micro-Architecture composed of the programming code for the microprocessors and the synthesizable description of the hardware parts of the system. Consequently two majors activities are involved to perform this task: Software generation and hardware generation.

2.5.1 Software generation

As in the ESL design tool we use the behaviour of the system is described with either C/C++ or SystemC, the software code of the functions mapped onto software processors is generated in C. This code uses common services of a Real Time Operating System (RTOS) parameterised by a set of attributes like task priorities. According to the targeted usage –rapid prototyping of DSP systems- the flexibility and the speed up in term of time to prototype of a such approach prevails on the performance losses.

Two templates are available to customize the generated code: either the VxWorks application programming interface (API) or a generic RTOS API could be used. In the last case, the API as to be implemented with the primitives of the targeted RTOS.

Anyway the generated code relies on common RTOS services like tasks creations, message queues and semaphores handling... For example if the function F0 of Fig. 2 a) is mapped on the software processor P0 of Fig. 2 b) the generated

code will contain a message queue corresponding to MsQ1 and two tasks, the first corresponding to F0 and the second IntN0 corresponding to an interface function automatically introduced to enable F0 to use the communication node N0.

Concretely, as our platform embeds TI DSP, we have implemented the generic RTOS API with the DSP/BIOS primitives. Fig. 3 illustrates this approach by giving an example of the generic RTOS message queue send primitive implemented with DSP/BIOS.

To end the software code generation, platform specific code has to be written to solve the inter processing elements communication problem. The communication drivers of the targeted platform are called inside the interface functions introduced in the macro-architecture model through an API mechanism.

```
int msgQueueSend (
    MSG_QUEUE_ID *msQ, /* message queue on which to send */
    void *data,        /* message to send */
    int timeout,
    int priority )
{
    return( MBX_post( msQ->mbx_handle, data,
        timeout == WAIT_FOREVER ? SYS_FOREVER : timeout ) );
}
```

Fig. 3: Generic RTOS API primitive example

Thereby we have developed C++ Concurrent Sequential Process like I/O drivers: we provide a specific class for each type of link available on the point to point cable based communication network of our Sundance platform. These objects are used thanks to the simple interface described on fig. 4.

```
void send(int *w, int n );
void recv( int *w, int n );
void sendReq( int *w, int n );
void recvReq( int *w, int n );
bool sendRun();
bool recvRun();
```

Fig. 4: I/O drivers interface

Methods send(...) and recv(...) are blocking communication calls whereas sendReq(...) and recvReq(...) are the non-blocking ones; sendRun() and recvRun() methods are predicates to check if the communication is still active or not. A benchmark has been developed to characterise and measure the performance of our I/O drivers. The effects of data location, compiler optimisations, DMA usage on the throughput and

CPU load have been studied allowing a fine tune of the extra DSP communications.

2.5.2 Hardware generation

Characterised Hand-written or IP hardware components can be used to implement the hardware parts of the system. In this paper we promote a flexible approach based on HLS to generate these IP components.

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

A typical HLS tool performs four main tasks [11]: (1) source specification analysis (identify computations); (2) hardware resources selection and allocation for each kind of operation; (3) operation scheduling; (4) optimised 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 ...) where components are characterised 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 behavioural description for HLS can be made customisable 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.

In the PALMYRE project, the HLS tool we use is GAUT [12], [13]. Its input language is a subset of behavioural VHDL. A SystemC entry point will be soon available, thus the languages used in the design flow will be unified. With respect to the functions to be computed the HLS tool could synthesise the hardware parts of the system. Data

Flow oriented functions (Fig. 5) are typical candidates.

Beyond the synthesis of the algorithmic cores the HLS tool also synthesises our platform specific communication interfaces: CP or SDB can be used as an input or an output of the synthesised component. Therefore, the synthesized function can be easily interfaced with others software or hardware processors of our rapid prototyping platform.

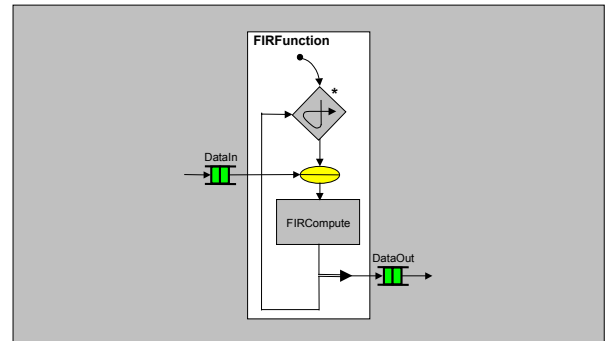


Fig. 5 A typical candidate for the HLS tool

At this last step of our design flow the system is refined enough to reach the entry point of the downstream tools we use: Code Composer from TI for the software parts, ISE from Xilinx for the hardware parts.

3 MIMO System design

As the PALMYRE project targets MIMO applications prototyping, this section presents the flexibility of our design flow applied to a MIMO system using orthogonal space time codes [14]. In this paper we especially focus on the encoding parts.

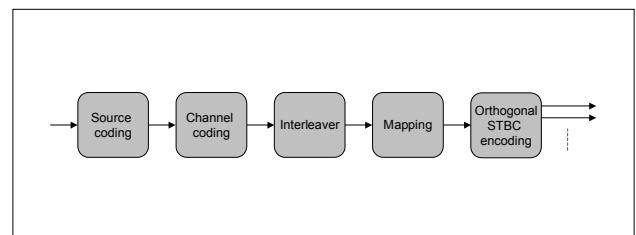


Fig. 6 MIMO encoder

3.1 Functional exploration

As shown on Fig. 6, the targeted system is composed of following cascading blocks: the source coder, the channel coder, the interleaver, the mapper and the orthogonal Space Time Block Code (STBC) encoder.

Therefore, the corresponding functional model is composed of five functional blocks interconnected by communication ports. To have a complete test bed, we also model the system environment (Fig. 7): known test patterns are send to the source coder, so, when the functionality of each blocks has been validated, we dispose of a golden reference composed of these test patterns and the resulting data produced by each functional block.

At this stage of our flow, high levels models of the system are used to explore and validate various alternatives for each functional blocks. Simulations of these models are used to experiment different mappings: 4 QAM and 16 QAM, different channel coding strategies: convolutional coder, convolutional coder plus Reed Solomon coder, turbo codes, ... and different orthogonal STBC encoders. Beyond the functional results obtained at this stage, the profiler integrated in our ESL tool allows to measure the relative computational complexity of each blocks. Although these results depend on the characteristics of the simulation platform (CPU, Operating System), the more complex functions can be identified and the optimisations quantified.

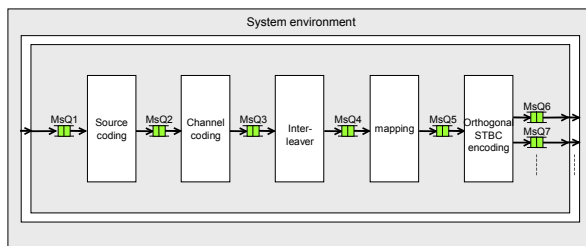


Fig. 7 MIMO System Functional Architecture

3.2 Architectural exploration

This topology and the data flow orientation of this application offers great opportunities in term of architectural exploration. Indeed each block can be implemented either in software or in hardware. Our strategy consists in first trying a full software

implementation mapped on a unique DSP. We profiled the timing behaviour of this first prototype so that we are able to back annotate our macro architecture models with an accurate timing of the software implementation of each blocks. As our platform embeds three types of DSP (c62, c67, c64), this experience has to be repeated for each DSP.

If this implementation does not fits the non functional specifications, in our case the throughput, the simulation of the timed macro architecture model allows the identification of the bottlenecks. As the timing of the functions is known, others software based architectural configurations (solutions embedding more than one DSP) can be simulated at the macro architecture level. If such software based architectures do not fit the specifications hardware accelerators are introduced. Thanks to the HLS tool we use, the design of these components is speeded-up.

To be more concrete, the following list presents the conceivable mappings of each functional block:

- The source coder is implemented in software. Indeed it is a complex function: a large diversity of data (voice, video, ...) could be processed by this block with different quality / data rates tradeoffs. Hardware implementations of the source coder are generally Hard IP (a HLS tool can be useful when designing a such function but can't handle the complexity of the whole function).
- The channel coding can be either implemented in software or in hardware. Hardware implementation is being preferred for the high computational complexity coders. For example [15] and [16] presents successful results obtained with the HLS tool we use in the fields of turbo codes and Reed Solomon codes.
- The interleaver can be mapped either on a software processor or on a hardware processor. The first mapping is interesting in terms of flexibility: changing the interleaving algorithm is simple and the second one unloads the software processors from a task easily realized in hardware.
- The mapper implementation tradeoffs are similarly to these of the interleaver.
- The orthogonal STBC encoder can also be implemented either in software or in hardware.

The flexibility offered by a software implementation allows to rapidly test several orthogonal STBC or to change the code at run time (for example if the channel changes). Similar to the mapper and the interleaver an hardware implementation unloads the software processors from a task easily realized in hardware.

Our design flow applied to MIMO systems is very promising. Numerous mixed hardware-software alternatives can be rapidly prototyped. In this manner the design space can be efficiently explored without focusing on implementation.

4 Conclusion and future works

Our methodology releases the digital signal processing application designer from the implementation constraints: the golden functional model of the system expressed at a high level of abstraction is rapidly prototyped: software coding, generation of complex hardware parts of the system, software-hardware interfaces are handled by the collaboration the HLS and ESL design tools. Further efforts in our design flow will focus on improving the system level profiling by introducing new metrics, improving the reuse of existing hardware components and having a unified language in our flow.

Acknowledgements

This work takes place in the PALMYRE project sponsored by "Conseil Régional de Bretagne", "Conseil Général du Morbihan" and "Communauté de communes du pays de Lorient".

References

[1] Sundance Multiprocessor Technology, <http://www.sundance.com>
[2] Texas Instruments, <http://dspvillage.ti.com/>
[3] Xilinx, <http://www.xilinx.com/>
[4] A. Gerstlauer, R. Dömer, J. Peng, D.D. Gajski. System Design: A Practical Guide with SpecC. Kluwer Academic Publishers, 272 pages, 2001.
[5] J.P. Calvez, Embedded Real-Time Systems. A Specification and Design methodology, John Wiley, 670 pages, 1993.
[6] Cofluent design, Cofluent Studio, <http://www.cofluentdesign.com>

[7] O. Pasquier, J.P. Calvez, "An object-based executable model for simulation of real-time Hw/Sw systems", Proceedings of DATE 99, March 1999, Munich.
[8] R. Le Moigne, O. Pasquier, J.P. Calvez, "A Graphical Tool for System Level Modeling and Verification with SystemC", FDL' 03, September 2003, Frankfurt.
[9] R. Le Moigne, O. Pasquier, J.P. Calvez, "A Generic RTOS Model for Real-time Systems Simulation with SystemC", DATE 04, February 2004, Paris.
[10] R. Le Moigne, O. Pasquier, J.P. Calvez, "An Abstract Communication Bus Model for Performance Estimation in SoCs with SystemC", FDL 04, September 04, Lille.
[11] D.D. Gajski, N.D. Dutt, A. C.H. Wu, S. Y.L. Lin, High-Level Synthesis: Introduction to Chip and System Design, Kluwer Academic Publishers, 359 pages, 1992.
[12] LESTER-UBS, GAUT, <http://web.univ-ubs.fr/lester/www-gaut/>
[13] E. Martin, O. Sentineys, H. Dubois, J.L. Philippe, "GAUT: An Architectural Synthesis Tool for Dedicated Signal Processors", EURO-DAC 93, September 1993, Hambourg.
[14] V. Tarokh, N. Seshadri, and A. R. Calderbank, "Space-time codes for high data rate wireless communications: Performance criterion and code construction," IEEE Trans. Inform. Theory, vol. 44, pp. 744–765, Mar. 1998.
[15] E. Casseau, B. Le Gal, C. Jégo, N. Le Héno, E. Martin, "Reed-Solomon behavioral virtual component for communication systems", ISCAS 2004, IEEE International Symposium on Circuits and Systems, Vancouver, Canada, 21-23 mai 2004.
[16] Philippe Coussy, David Gnaëdig, Amor Nafkha, Adel Baganne, Emmanuel Boutillon, Eric Martin, "A Methodology for IP integration in DSP Soc: a case study of a MAP algorithm for turbo decoder", ICASSP'04, Montreal, Vol.5, pp. 45-49, May 2004.