Application-Oriented System Design as an Embedded Systems Development Strategy: a critical analysis

Danillo Moura Santos, Roberto Matos and Antônio Augusto Fröhlich
Laboratory for Software and Hardware Integration - LISHA
UFSC/CTC/LISHA P.O.BOX 476, Florianópolis, SC, Brazil
{danillo,roberto,guto}@lisha.ufsc.br,

Rafael Luiz Cancian
Department of Automation and Systems Engineering - DAS
Federal University of Santa Catarina - UFSC
cancian@das.ufsc.br

Abstract

Nowadays development strategies are not suitable for the design of many embedded systems applications, because they do not guide the developer in the use of nowadays software engineering concepts as aspects and generic programming. The present work shows the pros and cons of Application-Oriented System Design (AOSD) strategy used in the design of a case study embedded system, aiming at AOSD methodology improvement. The disadvantages found in this case study may contribute to improve hardware generation according to AOSD.

Keywords: *embedded systems design methodology, aplication-oriented system design*

1. Introduction

Embedded systems are pervasive in our daily lives, from brake control systems in our vehicles to *smart appliances* in our homes. Hardware production technology advances and manufacturing costs reductions have allowed the development and popularization of very complex embedded applications, however, this complexity is reflected in these systems design. The lack of an efficient Software/Hardware design methology makes the embedded systems design an error-prone task.

Hardware/Software Co-design methodologies, such as IP-Based Design (IBD) [1] and Platforma-Based Design (PBD) [2], despite the name, do not address software design issues comparably to modern software engineering methodologies. The main focus of current co-design methodologies seems to be on the design of a hardware platform for a given application domain, on top of which some software will be running. The software itself, is mostly regarded as application programmer's duties. Furthermore, the guidelines to decompose an application domain in IPs that will eventually build up a platform are mostly empirical.

The *Platform-Based Design* methodology, proposed by Vincentelli is widely used in embedded systems design. The development process in PBD consists in developing a platform that fulfils the requirements of a set of applications (for example, multimedia applications), and that may be used in diferent systems design. This methodology is used, for example, in the Philips Nexperia platform for multimedia applications [3], and in the TI OMAP platform for mobile phones [4]. The development of a platform that fulfils the requirements of a set of applications is not trivial, detailed requirement and viability analysis must precede a new platform design. Therefore, a new platform development time must thus be considered in the final product time-to-market estimation. This methodology is an evolution of the IBD, where the design consisted in assembling IPs to create the final hardware system, what resulted in difficulties in the software development.

Polpeta in [5] presents an AOSD extension to allow hardware automatic generation, that enables operating system and hardware to be tailored to a specific application. In this paper the analysis of the development of a MPEG-2 Multiplexer designed according to AOSD is used to elucidate the strong and weak points of this methodology. This paper is organized as follows: section 2 elaborates on the development methodology used. Section 3 present the case study, section 4 shows development strategy employment results and section 5 concludes and presents future perspectives.

2. Strategy adopted

Application-Oriented System Design (AOSD) is a domain engineering methodology that elaborates

This work was partially supported by FINEP (Financiadora de Estudos e Projetos) grant no. 01.04.0903.00.

on the well-known domain decomposition strategies behind Family-Based Design (FBD) and Object-Orientation (OO), i.e. *commonality* and *variability* analysis, to add the concept of *aspect* identification and separation yet at the early stages of design [6]. In this way, AOSD guides a domain engineering towards families of components, of which execution scenario dependencies are factored out as "aspects" and external relationships are captured in a component framework. This domain engineering strategy consistently addresses some of the most relevant issues in component-based software development:

Reusability: components tend to be highly reusable, for they are modeled as abstractions of real elements of a given domain and not as parts of a target system. More over, by factoring out execution scenario dependencies as aspects, components can be reused unmodified in a variety of scenarios simply by defining new aspect programs.

Complexity management: the identification and separation of execution scenario dependencies implicitly reduces the number of components in each family, since those components that would have been modeled to express a variation in the domain that originates from a scenario dependency are suppressed whenever the dependency can be modelled as an aspect. These directly improves on maintainability, as the overall complexity is now confined in fewer constructs.

Composability: by capturing component relationships in a component framework, AOSD enables components to be more easily combined while generating a system instance. It also put some limits to the misbehaviors that can arise from applying aspect programs to pre-validated components. *Feature-based models* are of great value at this point to capture configuration knowledge and thus make system generation a more predictable procedure.

In order to allow the components and the operating system to be portable to most architectures, a system designed according to AOSD makes use of Hardware Mediators [7]. The main idea of this portability artifact is to keep an *interface contract* between the operating system and the hardware. Each hardware component is accessed through its own mediator, thus granting the portability of abstractions that use it without creating unnecessary dependencies. Mediators are mostly static-metaprogrammed and "dissolve" themselves in the system abstractions as the interface contract is met. In other words, a hardware mediator delivers the functionality of the corresponding hardware component through an operating system oriented interface.

As other components in AOSD, hardware mediators are grouped in families where each member is the representation of an entity in the hardware domain. This approach guarantees that the system will have only the object-code necessary to support the application. Non-functional aspects and cross-cuting properties of the mediators are encapsulated as scenario aspects that may be applied to family members as required.

Hardware mediators have initially been created to facilitate the portability of systems developed according to AOSD methodology. However, because of its straight relation to hardware components, they may also be used to identify the hardware components (IPs) that are really necessary to build a system to support a specific application.

In the context of Programable Logic Devices (PLD) where IPs are usually implemented using hardware description languages like VHDL and Verilog, hardware mediators might be used to infer the IPs really required to the system hardware and some of these IPs properties. These IPs are identified as the hardware mediator is instantiated by the application, thus the system hardware will have only the components required to support the application.

There are three different ways to select an IP in this methodology: *discrete IP-selection*, when there is only one IP that fulfills the application requirements, *combined IP-selection*, if there are more than one IP that fulfills the application requirements, one will have to select a specific IP and *explicit IP-selection*, the programmer can choose independently all hardware components that will be synthesized.

3. Case study

The embedded system developed as a case study was an ITU-T H.222 standard MPEG audio and video Multiplexer. This multiplexer was developed in the context of the Brazilian Digital Television System (SBTVD) and consists of an embedded system responsible for assembling a MPEG transport stream, receiving the elementary streams of audio and video. The transport stream is sent to the modulation system in order to be transmitted to users' reception units.

The application uses an arbitrary number of threads to handle elementary stream reception (ES). These threads execute with high priority in order to avoid hardware reception buffer overflows. Two control threads provide stream timing information (T) and packet synchronization (S). The multiplexer thread gathers data from the ES, T and S threads and outputs a transport stream through an output thread (See figure 1).

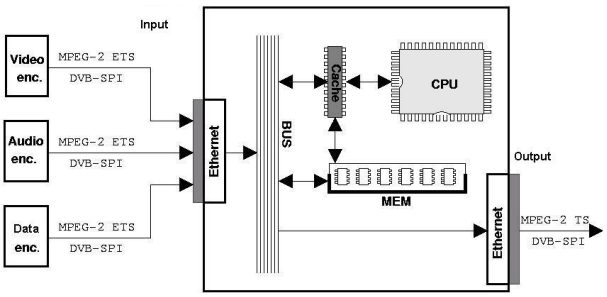

Figure 1. MPEG-2 Multiplexer schematic

We used a ML310 development board from Xilinx

which contains a VirtexII-Pro XC2V30 FPGA (*Field Programable Gate Array*) to implement the system hardware. This FPGA has two IBM PowerPC 405 32-bits *hardcore* processors, and one of then was used in this project. Besides the PowerPC 405 processor core, some other IPs were infered from the application code. These IPs, required by the application, were also synthetized on the VirtexII-Pro FPGA. The application software was written using Embedded Parallel Operating System (EPOS) Application Programming Interface (API).

In this application Ethernet controllers were used as communication interfaces. These were infered straight from application (*discrete selected*), as it instantiates Ethernet hardware mediators to send and receive the streams packets. The ethernet controller used is part of the ML310 platform, and was accessed through the PCI bus, moreover, a PCI Bridge was *discrete selected* to satisfy a dependency of the PCI Ethernet controller.

An interrupt controller was also *discrete selected* to compose the system hardware. The interrupt controller provides the means by which I/O devices (such as UART and ethernet controllers) request attention from the processor to deal with data transfers. A timer IP was also *discrete selected* to satisfy a thread requirement. The timer was used to invoke the scheduler.

The application required access to RAM memory, so a DDR controller was *discrete selected* to the system hardware. The memory controller generates the necessary signals to control the reading and writing of information from and to the memory, and interfaces the memory with the other major parts of the system. The ML310 platform has 256 Mbytes of DDR memory and EDK has a memory controller IP to support memory access.

The Xilinx CAD tools EDK and ISE (Integrated Software Enviroment) were used to integrate and synthesize these components. EDK has a large repository of IPs, and it provides easy interfaces to configure and integrate such IPs. According to the AOSD strategy, some IPs available in the platform were not selected to avoid unnecessary overhead to the system. Unselected IPs were USB interface controller, the SPI controller, the parallel port controller, and some others.

One of the requirements of this project was the use of DVB-SPI (Digital Video Broadcasting - Synchronous Parallel Interface) interfaces to receive and send streams. The programmable logic group developed a hardware component (IP) to carry through this function. This IP was integrate to the system using the PLB Bus, described later. This integration was a difficult and time consuming task. The absence of a tool to adequate the IP interface to the PLB standard interface generated with EDK demanded a lot of simulations and extreme tests in ML310 platform.

The hardware components connection to the central processor (PowerPC 405) is done using an OnChip bus. The bus pattern used by PowerPC processor and Xilinx in this FPGA is CoreConnect, developed by IBM. CoreConnect is a bus logic [8], wich is composed by three buses: Processor Local BUS (PLB), On-Chip Peripheral BUS (OPB) and Device Control Register BUS (DCR).

High-performance peripherals (such as DDR memory controller) connect to the high-bandwidth, low-latency PLB. Slower peripheral cores (as UART) connect to OPB, which reduces traffic on the PLB, resulting in greater overall system performance. The DCR provides fully synchronous movement of GPR (Global Purpose Register) data between CPU and slave logic.

The hardware components provided by Xilinx, such as UART and Ethernet controllers, are compatible with these buses OnChip patterns, and integrating and configuring them is an easy task. However, integrating non-bus-standard IPs has showed a bit difficult. The use of the AOSD methodology, Xilinx CAD tools and ML310 prototype platform made the hardware development quick (compared to the development of a new physical hardware platform) and enabled the development of *application-tailored* hardware and software.

4. Results: pros and cons of the strategy adopted

The AOSD strategy has shown itself efficient in the design of embedded system software and hardware. In this case study, we realized that it reduces a lot the development time, compared to the development using traditional microcontroller-based strategies and reduces a little the development time if compared to platorm based design. The AOSD methodology inferes the hardware components straight from the application code, based on the application requirements, this simplifyed the overall process. The developer knows exactly which hardware components will be necessary to support the application.

As showed in the case study, only the hardware components necessary to the application function will be in the final system hardware. The developer can choose the necessary components and discard the unnecessary ones. If more than one component with similar functionality is available to the developer, either the developer may explicitly select one, or the simplest one that satisfies the application requirements may be selected by default. When using established platforms, however, the developer isn't able to modify the entire system hardware.

The use of EPOS, which is an operating system developed following AOSD concepts, makes the application code extremely efficient. As EPOS is based in software components, only the components necessary to the application functioning will be compiled to object-code. This results in an efficient code with reduced size, since no unecessary resources management will be present. Moreover, the use of EPOS enabled the application software to be executed in different architectures. EPOS is a multiplatform operating system, thus the application may be compiled to run in different architectures, as long as this architecture has the hardware components necessary to support the application requirements.

In addition to these advantages found in AOSD methodology, the use of FPGAs enables the possibility of later upgrades to system hardware. FPGAs dynamic hardware properties allow the system to be changed even after its project conclusion. This is an important characteristic of embedded systems because of the continuous emerging features and applications in this area.

Most of the time, the infer process creates a relation of one hardware mediator to one hardware component. It would be ideal that a hardware component could be tailored to a hardware mediator, instead of using ready-made IPs. The designer should be able to configure an IP according to the hardware mediator being instantiated by the application. This would result in smaller hardware components, extremely adapted to the application.

Hardware components aren't usually developed to be totally configurable, only the minority have some *generic* fields that can be modified by the developer. If hardware components were developed following AOSD approach, every IP would be specifically tailored to the system in which it is being used. This would reduce the size of these components, saving FPGA area.

As seen in the case study, IPs that haven't a bus interface standard similar to the system bus standard requires large efforts for adaptation. In addition to this, OnChip buses bridges creates additional overhead to the system and requires more FPGA area. A common definition of an IP core is a design function with well-defined interfaces. Many IP cores are initially designed to perform specific functions, evolving to more complex components, which incorporates others functions, thus becoming reusable to others designs. If the IP function is well separated from the IP interfaces in the design, it should be simple to create new interfaces. Thus, besides developing IPs according to AOSD principles, IP interfaces might be independent.

In the traditional EPOS design process, the application developer defines an architecture and then writes the application. However, in embedded systems development, it is common to choose the cheapest architecture that fulfils the application requirements. Design space explorations might allow application programmers to first write an application, and then choose the cheapest suitable architecture for the design.

5. Conclusion

This work was motivated by the growing complexity in embedded systems development and the deficiencies in currently design methodologies. We ellaborated on a strategy, based on the Application-Oriented System Design methodology, which allows hardware and software to be tailored to specific applications.

By using AOSD methodology, it was possible to infer specific hardware components from application code, using the artifact of hardware mediators. This reduced development time and helped the development of a system with the least number of hardware components possible.

The evolution of the AOSD methodology, especially hardware automatic generation from application code, proved to be useful in embedded systems design. Future improvement in hardware components development process should provide more adaptable and specialized, application-tailored components, with smaller size, enabling more refined hardware choices.

References

[1] D. D. Gajski, "IP-based design methodology", June 1999, p. 43.

[2] A. Sangiovanni-Vincentelli and G. Martin, "Platform-based design and software design methodology for embedded systems", *IEEE Design & Test of Computers*, vol. 18, no. 6, pp. 23–33, November-December 2001.

[3] Philips, "Philips Nexperia Platform", Technical report, Philips, 2004.

[4] T. Instrument, "OMAP Texas Instrument Technology", Technical report, Texas Instrument, 2004.

[5] F. V. Polpeta and A. A. Fröhlich, "On the Automatic Generation of SoC-based Embedded Systems", volume 1, September 2005, pp. 873–880.

[6] A. A. Fröhlich, *Application-Oriented Operating Systems*, Sankt Augustin: GMD - Forschungszentrum Informationstechnik, 1 edition, 2001.

[7] F. V. Polpeta and A. A. Fröhlich, "Hardware Mediators: a Portability Artifact for Component-Based Systems", *In: Proceedings of the International Conference on Embedded and Ubiquitous Computing*, vol. 3207, pp. 271–280, 2004.

[8] IBM, "The CoreConnect Bus Architecture", Technical report, IBM, 1999.