Advantages and Disadvantages of Application-Oriented System Design in Embedded Systems Design

Danillo Moura Santos, Roberto Matos,
Rafael Luiz Cancian and Antônio Augusto M. Fröhlich
Federal University of Santa Catarina
Laboratory for Software and Hardware Integration
UFSC/CTC/LISHA P.O.BOX 476, Florianópolis, SC, Brazil
{danillo,roberto,cancian,guto}@lisha.ufsc.br

Abstract

The development of complex embedded systems is feasible do to the low cost provided by the technology advance. This complexity is also present in the design process. The computer aided-design (CAD) tools evolution has aided this process. However, nowadays development strategies are not suitable for the design of many applications. This work shows advantages and disadvantages of application-oriented system design used to develop two embedded systems. The disadvantages found in these two designs may help AOSD methodology improvement in hardware generation. [1]

Keywords: embedded systems, CAD, reuse, IPs, AOSD

1. Introduction

Embedded systems are pervasive in our daily lives, from brake control systems in our automobiles to *smart appliances* in our homes. Advances in hardware technology and manufacturing costs reductions have allowed the development of very complex embedded applications. However, this complexity is reflected in the system development process. The evolution of CAD (Computer Aided Design) has aided this process, but existing development strategies fall short of providing an optimal solution.

The *Platform-based Design* methodology, proposed by Vincentelli [13] is a widely used embedded systems development strategy. It aims to reduce development and production costs and to reduce electronic systems development time, affecting as least as possible the system performance (whenever necessary). The development process in Platform-based design consists in specifying a platform that fulfills the requirements of not one but a set of similar applications (for example, multimedia applications), and that may be used in different systems designs. According to Vincentelli, a platform is a design abstraction layer that enables successive refinements, resulting in a more adequate abstraction. After a number of refinements cycles the result is the desired hardware and software platform.

This methodology is used, for example, in the Philips Nexperia platform for multimedia applications [10], and the TI OMAP platform for mobile phones [4]. The development of a platform that fulfills the requirements of a class of applications is not trivial. The platform development time must thus be considered in the final application's time-to-market estimative. Detailed requirement and viability analysis must precede the development of a platform, and one should consider if that platform will actually be used in many applications. Focusing on a class of applications makes Platform-based design inadequate for extremely custom-designed systems, that result in a single application.

Hardware platforms for embedded systems are usually one of three kinds: microcontrollers, dedicated integrated circuits (ASIC), or programmable logic devices (PLD), such as Field Programmable Gate Arrays (FPGA). Microcontrollers have being the most common platform in the last decades and nowadays they still are. Microcontrollers are efficient, cheap, and exist in a wide variety of families (architectures) and features. However they rarely exactly satisfy the application's requirements, making necessary the use of additional integrated circuits (usually called glue logic). The developer will search for a microcontroller and other circuits whose features are closer to what's needed, but often the platform offers a little more functionality that will never be used.

ASICs are economically viable only when the production scale is really large, and are not an option for prototyp-

1 1-4244-9701-006$20.00 ©2006 IEEE

ing and can not be customized. Until some years ago FPGAs were used only for ASIC prototyping and for low scale production. However, with the evolution of FPGAs technology, demands for time-to-market and NRE (nonrecurring engineering) reduction, there have being a significant increase in using FPGAs as hardware platforms for embedded systems [6]. Moreover FPGAs allow the integrated design of hardware and software (co-design), fulfilling exactly the application's requirements.

Polpeta [12] presents a strategy that enables operating system and hardware to be tailored together to form a complete system (hardware and software SoC) to support specific applications. This strategy is based on software and hardware components reuse and is an extension of Application Oriented System Design (AODS) methodology. In this paper we elaborate on that strategy, identifying advantages, disadvantages and shortcomings. We analyze the development of two embedded systems: a MPEG-2 Multiplexer and a CAN listener. This work was developed within PDSCE project [2]. The rest of this paper is organized as follows: section 2 elaborates on the development strategy used. sections 3 and 4 presents the two case-studies. section 5 analises the development strategy results. section 6 concludes and presents future perspectives.

2. AOSD strategy overview

The *Application Oriented System Design* (AOSD) methodology [1] was developed to build component-based run-time support systems that can be tailored according to the requirements of particular applications. This methodology address the developer, from design to implementation, to produce an application-oriented operating system. By this, we mean that this operating system will match exactly the requirements of a specific application. This methodology uses state-of-art software engineering techniques, such as Domain Analysis and Decomposition, Family Based Design (FBD) [9] and Aspect Oriented Programming (AOP) [5] to separate abstractions from scenarios dependent aspects (hardware and environment dependencies). Abstractions are free to reuse and extend, and are independent from a scenario execution [1].

The hardware dependency can be reduced using aspects separation concept of AOP in the decomposition process. This concept provides means to identify scenario variations, that instead of modeling a new family member, define a scenario aspect. For example, although modeling a new family member to communication mechanism family that can operate in shared, protected or multithreding modes, this could be modeled as scenario aspects that, when activated, would block the communication mechanism (or its operations) in a critical code section.

Application-Oriented Systems Design proposes a domain engineering procedure (see Figure 1) that models software components using three constructs: families of scenario-independent abstractions, scenario adapters and inflated interfaces.

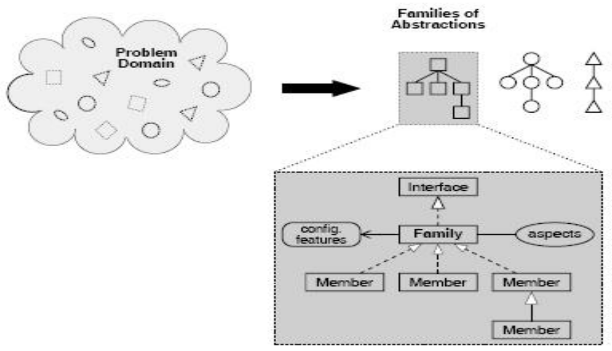

Figure 1. Overview of application-oriented domain decomposition.

Families of scenario independent abstractions are identified during the domain decomposition phase. Abstractions are identified from the domain entities and grouped in families according to their common characteristics. During this phase, the aspect separation process is also performed. This allows abstractions to be reused in differents scenarios. Software components are then implemented according to this abstractions.

Scenario adapters are used to solve scenario dependencies, these must be factored out as aspects, thus keeping abstractions scenario-independent. However, for this strategy to work, means must be provided to apply aspects to abstractions in a transparent way. This is achieved using scenario adapters [2], that wraps an abstraction, intermediating its communication with scenario dependent clients to realize the necessary changes, without overhead.

Inflated interfaces hold the features of all members in a family, resulting in a unique view of the family as if it were a "super component". This allows application developers to write applications with base in a clear and well-known interface, postponing the decision about which member of the family shall be used until the moment the system is generated. The association between an inflated interface and a specific member of the family will automatically be made by the configuration tool, that identify which properties of the family have been used, in order to choose the sim-

2 The Project PDSCE – Embedded Systems Development Environment is funded by FINEP under grant no. 01.04.0903.00

plest member of the family that implements the required interface. This member will so be agregated to the OS in compile-time.

2.1. Hardware Mediators

Aiming the OS and its components portability to virtually any architecture, a system designed according to AOSD makes use of Hardware Mediators [11]. 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. This means that a hardware mediator delivers the functionality of the corresponding hardware component through an operating system oriented interface.

Hardware mediators have *configurable features* that allow some hardware components features to be switched on or off, according to the requirements of the abstraction. These properties aren't only flags that can be turned on and off. Configurable features can be implemented using generic programming what allows implementing structures and algorithms in software without a significant overhead. An example could be the generation of CRC codes as a configurable feature of a communication hardware component.

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 necessary object-code to support the application. Non-functional aspects and cross-cutting properties of the mediators are encapsulated as scenario aspects that can be applied to family members as required.

2.2. Using Hardware Mediators to infer Hardware Components

Hardware mediators were initially been created to facilitate the portability of applications and run-time support systems developed following AOSD methodology through different fixed hardware platforms (microcontrollers or personal computers). However, exactly because of their straight relation/dependency to hardware, hardware mediators can be used to select the necessary hardware components that fulfill the requirements of an application.

When the hardware platform is configurable, that mean, when it's a PLD (Programable Logic Device), hardware components are implemented using hardware description languages as VHDL and Verilog and they are known as Intellectual Properties (IPs). In this scenario, hardware mediators could infer the necessary IPs (and their features) during the system composition [12]. For example: if a NIC (Network Interface Card) hardware mediator is used, then a NIC IP must be inferred, automatically or not.

There are three different ways to select an IP in this methodology: In the first case, if there is only one IP that fulfills the application requirements, the IP selection could be automatically done by a tool and no explicit programmer decisions must be taken. This case is named *discrete IP-selection*. An example is the use of paged memory in the application, allowing the tool to infer a MMU IP with paged memory support.

In the second case, if there are more than one IP that fulfills the application requirements, one will have to select a specific IP. This is called *combined IP-selection*. A list of these IPs is automatically inferred considering the requirements and then the application programmer manually selects a specific IP to be used.

In the third case, named *explicit IP-selection*, the programmer can choose independently all hardware components that will be synthesized. This is useful when a hardware component is hidden by a system abstraction.

In the first two cases, the configurable features of hardware mediators can be deployed in hardware components. A configurable feature can help the inference of a specific hardware component. An example could be the use of CRC codes by a NIC device: the IP representing this NIC should be capable of generating CRC codes.

3. Case study 1: Audio and Video MPEG-2 Multiplexer

The first embedded system developed as a case study of AOSD was the audio and video MPEG-2 Multiplexer. The MPEG-2 multiplexer was developed in the context of the Brazilian Digital Television System (SBTVD) and consists of an embedded system responsible for receiving the elementary audio and video streams and assembling them in a MPEG-2 transport stream. The transport stream is then sent to a modulation system in order to be transmitted.

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 2).

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

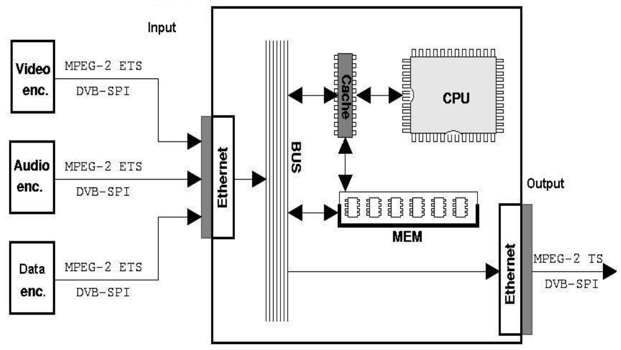

Figure 2. MPEG-2 Multiplexer schematic

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 might be infered from the application. These IPs, required by the application, were also synthetized on the VirtexII-Pro FPGA. The application software was written using EPOS API (Application Programming Interface).

In the development phase of the project, the MPEG-2 Multiplexer application required a display, that would be used to system debug. This display was implemented through an UART. The UART was *combined selected* to compose the system hardware. As the EDK (Embedded Development Kit) platform used to support hardware development had two UARTs IPs in its repository, the application programmer had to choose one.

In addition to the UART, Ethernet controllers were also used. 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 controllers.

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 256Mbytes of DDR memory and EDK has a memory controller IP to support memory access.

The Xilinx CAD tools EDK (Embedded Development Kit) 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 include the USB interface controller, the SPI controller and the parallel port controller, among others.

The hardware components connection to the central processor (PowerPC 405) is done using a OnChip bus. The bus pattern used by PowerPC processor and Xilinx in this FPGA is CoreConnect, developed by IBM. CoreConnect is a bus logic [3], 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. The use of the AOSD methodology and Xilinx CAD tools made the hardware development quick and enabled the development of *application-tailored* hardware and software.

4. Case study 2: CAN Listener

Controller Area Network (CAN) buses are present in most modern vehicles, air-planes, factories and others. CAN protocol is a real time, serial, broadcast protocol and it has a high security level .CAN packets have an identifier that must be unique in all the network and this identifier represents the priority of the message. Identifying the packets and its contents make possible monitoring the vehicle properties, like fuel consumption, break usage, acceleration rate and others properties.

The CAN Listener has been developed to transform CAN packets in packets that can be transmitted through a serial interface (UART), which allows monitoring of a CAN bus and externally identifying packets and the information they contain. It basically has one thread that is blocked waiting for messages from a CAN bus. When such message arrives, the thread process, transforming it in a sequence of bytes that's sent through a UART interface. The thread then waits for a CRC confirmation and, if no confirmation is received, the sequence is sent again.

The CAN listener was implemented in a VirtexII-Pro XC2V30 FPGA (Field Programable Gate Array) present in the ML310 evaluation board from Xilinx. Like the MPEG-2 Multiplexer, this system has also used one of the PowerPC 405 hard cores found in this FPGA. Other IPs were infered from the application. These IPs, required by the application, were also instantiated in the VirtexII-Pro FPGA. The application software was written using EPOS API (Application Programming Interface). The Xilinx CAD tools EDK (Embedded Development Kit) and ISE (Integrated Software Enviroment) were used to integrate and synthesize these components. The Mentor Graphics Modelsim was used to simulate and test the IPs.

The CAN Listener application required an UART serial interface to send the serial packets created with the contents of received CAN packets. This UART was *combined selected* to compose the system hardware. A CAN controller IP was necessary to allow the system to comunicate with the CAN buses. The CAN controller was infered straight from application (*discrete selected*), as it directly instantiates CAN hardware mediators to receive CAN packets.

As there was no CAN controller IP available in the EDK development tool, an open-source CAN IP found at [7] was used. This IP is compatible with the Wishbone bus interface [8]. As there is no native support for this bus interface in our development platform, the IP interface had to be adapted to the OPB bus. The IP developers made the interface definition and its logic very dependent, making the development of a new interface to the IP virtually impossible. Thus, a bridge between OPB Wishbone buses interface was created. Because of the differences between the two buses, the bridge development process was very laborious, requiring a complex state machine, and several simulation cycles before the integration to the system.

The CAN Listener application also required access to RAM memory, so a DDR memory controller was *discrete selected* to the system hardware. The ML310 platform has a 256Mbytes DDR memory and EDK has a memory controller IP to support this memory access. As in the MPEG-2 Multiplexer system, an interrupt controller was also *discrete selected* to compose the system hardware. As in the first study-case, some IPs present in the platform were removed to avoid unnecessary overhead to the system. The PCI Bridge, USB interface controller, the SPI controller, the parallel port controller, among others, were excluded. Figure 3 presents a simplified schematic of the CAN Listener hardware.

This case study demonstrated that integration of different IP bus technologies is not trivial, and that the lack of hardware components in the CAD development tools may generate large integration efforts.

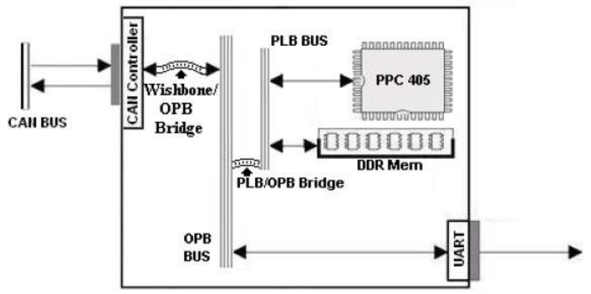

Figure 3. CAN Listener schematic

5. Results: Advantages and disadvantages in AOSD methodology

The AOSD methodology has shown to be efficient for the design of embedded system software and hardware. In these case studies, we realized that it reduces significantly the development time, compared to the platform based design and the development using fixed hardware. The automatically inference of hardware components facilitates the SoCs generation, once the code itself makes use of hardware properties that can infer an specific IP. If the developer has a repository with different kinds of IPs, the simplest one satisfying the application requirements is used.

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.

The use of EPOS makes the application code extremely efficient and only the components necessary to the application will be select to compose the binary code. Moreover, the use of EPOS allows the application to be executed in different architectures, from 8 bit microcontroller to 64 bit complex processors, and also using several types of FPGAs.

Besides these advantages found in AOSD methodology, the use of FPGAs enables the possibility of late upgrades to system hardware. The FPGAs configurable hardware property allows the system to be changed even after its project is finished. This is an important characteristic to embedded systems, because of continuous emerging features and functions 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 some of them 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 second 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 a base 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 his design.

6. Conclusion

The embedded system design growing complexity and the known design strategies deficiencies have motivated this work. We have designed two embedded systems following the AOSD methodology. We found advantages and disadvantages in this strategy, contributing to the evolution of this methodology.

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

The evolution of the AOSD methodology, especially in the area of hardware generation from application code, should prove 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] Antônio Augusto Fröhlich. *Application-Oriented Operating Systems*. Sankt Augustin: GMD - Forschungszentrum Informationstechnik, 1 edition, 2001.

[2] Antönio Augusto Fröhlich and W. Schroder-Preikschat. Scenario adapters: Efficiently adapting components. *In Proceedings of the 4th Wolrd Multiconference on Systemics, Cybernetics and Informatics*, July 2002.

[3] IBM. The coreconnect bus architecture. Technical report, IBM, 1999.

[4] Texas Instrument. Omap texas instrument technology. Technical report, Texas Instrument, 2004.

[5] et. all Kiczales, Gregor. Aspect-oriented programming. *In Proceedings of the European Conference on Object-oriented Programming*, 1241:220–242, June 1997.

[6] B. Mei, P. Schaumont, and P. Vernalde. A hardware-software partitioning and scheduling algorithm for dynamically reconfigurable embedded systems. In *Proceedings od the ProRISC2000*, Veldhoven, Netherlands, 2000.

[7] Opencores. Opencores. Technical report, Opencores, 2006.

[8] Silicore Opencores. Wishbone system-on-chip (soc) interconnection architecture for portable ip cores. Technical report, Opencores, 2002.

[9] David Lorge Parnas. On the design and development of program families. *IEEE Transactions on Software Engineering*, 2:1–9, March 1976.

[10] Philips. Philips nexperia platform. Technical report, Philips, 2004.

[11] Fauze Valério Polpeta and AntÃ´nio Augusto Fröhlich. Hardware mediators: a portability artifact for component-based systems. *In: Proceedings of the International Conference on Embedded and Ubiquitous Computing*, 3207:271–280, 2004.

[12] Fauze Valério Polpeta and Antôio Augusto Fröhlich. On the automatic generation of soc-based embedded systems. *In: Proceedings of the 10th IEEE International Conference on Emerging Technologies and Factory Automation*, 2005.

[13] Alberto Sangiovanni Vincentelli and John Cohn. Platform-based design. *IEEE Design e Test*, 18(6):23 – 33, Novembro 2001.

[14] Xilinx. Xilinx logiccore plb ipif (v2.01.a). Technical report, Xilinx, 2004.