Using Codesign Techniques to support Analog Functionality

Francis G. Wolff Michael J. Knieser Dan J. Weyer Chris A. Papachristou

Computer Engineering and Science Department
Case Western Reserve University
Cleveland, OH 44106

{wolff, knieser, weyer, cap}@alpha.ces.cwru.edu

ABSTRACT

With the growth of System on a Chip (SoC), the functionality of analog components must also be considered in the design process. This paper describes some of the design implementation partitioning issues and experiences using analog and digital techniques for embedded systems. To achieve a quick turn around for new embedded system development, a design methodology was extended for analog codesign based on the specify-explore-refine paradigm and system-level design methodology [3]. Many system-level issues were addressed including hardware/software codesign trade-offs.

Keywords

Hardware/Software Codesign, Analog, Design Methodologies.

1. INTRODUCTION

The promise of placing "Systems on a Chip" (SoC), Sematech, predicts state of the art ICs will exceed 12 million gates, operating at 600MHZ by the year 2001 [7] causes one to look for or extend a new methodology in order to handle this ever increasing level of integration complexity. The integration of analog with digital on a SoC will become more common, especially for industrial and automotive applications. Older methodologies are too inefficient to generate this type of design.

The embedded systems developed for the industrial automation as well as automotive markets have long product life spans, legacy IP interfaces and operate in an industrial environment that requires operation at 60°C with no forced air-cooling. They also possess a common but an important characteristic, in that they heavily interface with the real world through analog input and output ports. Interfacing with the real world will never disappear. The analog circuits needed to interface between the digital and the real world are also being integrated on the SoC for the same reasons of cost and performance but it is not clear how much of that interface should be done by analog or the embedded digital processor.

The hardware/software codesign techniques developed recently, allow a designer to decide how to partition the functional behavior between hardware and software in an embedded system. The question rises as how hardware/software codesign techniques could include analog. The hardware/software codesign techniques have been applied successfully on digital designs. With system on a PCB, this has been done in a very ad-hoc manner if at all, but with the advent of SoCs, this codesign technique is more appealing. The current design paradigm is to minimize analog and maximize on digital signal processing. The growth of digitally compensated analog circuits has made analog more and more acceptable and overcome many on its inherent weaknesses.

The application domain consists of where the real world interface transition between analog and digital should be. The analog and digital functionality is described at the behavioral level. This allows for a higher level exploration of the analog and digital codomains in the design before components are implemented. The exploration of where this transition lies is based on cost and performance metrics.

This paper first discusses briefly the current design methodologies and in particular the specify-explore-refine paradigm. Presented here is our methodology of the system-level methodology based on the specify-explore-refine paradigm [1]. A analog/digital embedded subsystem application was designed using this methodology. System design and component implementation tradeoffs and IP issues related to our design are then discussed. The conclusion will present comments related to our prototype and future direction.

2. BACKGROUND

The goal of each CAD methodology is to increase the productivity of the design engineer. Each methodology raises the level of abstraction of the design details of the system being designed, allowing for larger design sizes. In recent years the methodology of capture-and-simulate has given way to the describe-and-synthesize methodology [3]. Tools such as Synopsys Behavioral and Design Compiler have supported this methodology. The advantage of this describe-and-synthesize methodology is that the embedded system may be described in a behavioral form. This

technique is successful for 1M gate designs and the designer productivity was about 6K gates per design per engineer/month.

The specify-explore-refine paradigm differs from the other methodologies by the fact that the specification is an executable specification, which allows for automatic exploration of alternate design options. The refinement results in another executable specification for the next level of the design [9].

Gajski, et al. [3-4] have proposed a methodology targeted at the higher level. This is a formalized System-level design methodology based on the specify-explore-refine paradigm. The key components are the Specification Capture, System Design and Component Implementation. The Specification Capture is performed by specifying the desired system functionality and is described by functional objects, which can be of one three types: variables, behaviors and channels. Design alternatives are explored via estimators that best satisfy the cost and performance constraints of the Functional Specification. The exploration process consists of three tasks, which are applied to each functional object class [4]: allocation, partitioning and refinement. This particular ordering of tasks was suggested, as a good choice but is not a requirement. This refined specification results in a set of interconnected system-component blocks. The component implementation process then further explores the new refinement specification into RTL and software.

This system-level methodology is strong in hardware/software codesign, cost and performance techniques. Additional citations of design methodologies on hardware/software codesign are in [1,2,5,6,11]. However, none of these consider analog as part of the design methodology or the ability to identify cores in particular analog cores in the system design exploration process.

3. APPLICATION DOMAIN

A typical class of embedded systems has a host, which communicates with distributed embedded subsystems. Each of these embedded systems interfaces with the real world and consist of a cascade of analog to digital and digital to analog interfaces. The application domain for this methodology includes analog to digital interfaces on the particular embedded system. An embedded system may have analog input only, analog output only, or both. The analog input typically requires filtering, or signal conditioning before passed onto the host. The analog output is treated in the same way. The concern here is how much of this signal processing or other data processing should be done by the analog and how should be done by the digital as shown in Figure 1.

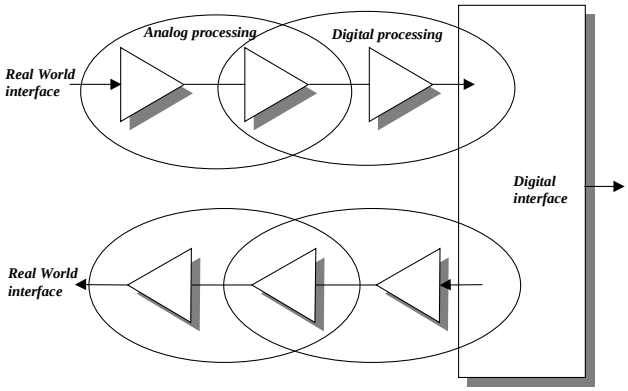

Figure 1. The Analog and Digital Codomains of a digital subsystem.

A less intuitive design exploration is the alternation of analog to digital as shown in Figure 2. For example, should the enclosed circle be all digital or a mixture of analog and digital. The interface cost of moving between analog and digital codomain is high but this cost may not be the determining factor in the cost and performance metrics.

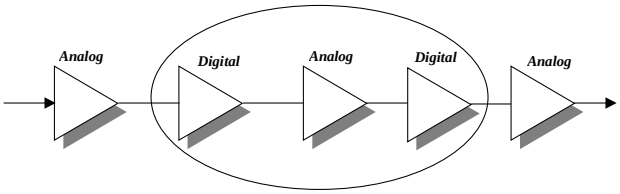

Figure 2. Exploration of analog and digital.

4. PROPOSED DESIGN METHODOLOGY

The proposed design methodology is shown in Figure 2. The function specification embodies the digital as well as the analog objects. These objects may be specified as existing encapsulated IP components as well as abstract components [10]. Since, industrial automation products have long live spans, legacy bus protocols and specialized analog interfaces. This often results in either soft cores or more commonly, firm and hard cores that are part of the specification. The specify-explore-refine paradigm is applied in system design step and also in the component implementation step. Each design step creates an executable specification to allow further exploration. Since, a refined specification is also executable this makes verification easier with automated tools, hence specify-explore-refine-verify paradigm may be more appropriate. The system design exploration deals with deciding how much of the input signal to digital should be analog and visa versa how much of output signal path should be digital.

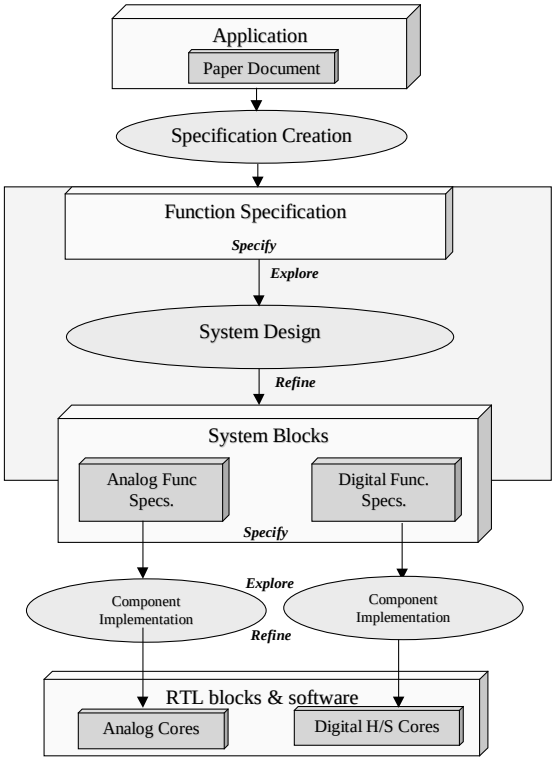

Figure 3. Proposed Design Methodology.

The system design exploration must consider what must be placed in analog and digital. After this has been allocated, partitioned, and refined, the component implementation exploration of the analog and the digital can now proceed separately. Certainly for verification they are simulated together. This early partitioning of analog and digital, rather than in the component implementation which is typical of describe –and-synthesize methodology supports the need of a higher system design abstraction as originally proposed by [3-4].

The allocation and partition, which are guided by the design metric estimators, must now consider analog estimators. In many ways, analog estimation may be easier, especially when using IP analog cores. Two key issues characterizing analog estimators are quality and sensitivity. Quality is the accuracy of the signal in terms of number of bits. Sensitivity is how much the component will deviate from the ideal component. This could be noise, interference, degradation, temperature drift, etc.

Analog presents a problem whether allocation should proceed partitioning. For example, It is not simply mapping functional behaviors to analog. Certainly analog can be viewed as a non-programmable hardware processor. Variables and channels are no longer strictly in the digital domain. Since the ordering of tasks is not fixed [4], the proposed methodology chose to partition, allocate and then refine. Partitioning the analog can be approached several ways as shown in Figure 4.

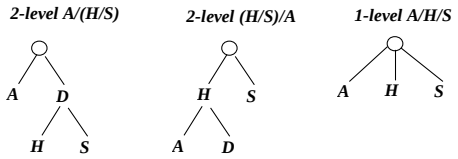

Figure 4. Analog-digital partitioning approaches.

The most intuitive and common method is the 2-level A/(H/S) partitioning. Here the analog is first partitioned from the digital. The Digital is then partitioned into hardware/software.

A less intuitive method is the 2-level (H/S)/A partitioning. Here, the hardware/software codesign proceeds in the traditional way. In next level, the analog is treated as a type of hardware. The hardware is then partitioned into analog and digital. This method allows the first level to be the upper bound in area constraints. Analog IP cores lend themselves well with the two level partitioning as opposed to the single level partitioning since the granularity is large.

The last approach is a single level A/H/S partitioning which requires more computational effort on the partitioning algorithms, since it must simultaneously consider all cases. The ability to explore alternative designs quickly is a critical issue. The 2-level methods lend themselves well with 0-1 integer-programming algorithms. The single level method is usually practical when the interface costs between domains is low, which is not the case here. Whether this method has any merit is a subject of future research.

The exploration and refinement was achieved by having a rules list, which allowed the mapping of certain functional objects to generic components with estimators. For example, common differential equations can be mapped to well-known analog and digital designs. High level signal processing functions such as filters can be mapped to analog or digital system blocks.

The executable functional specifications for the design were MATLAB Models, C++ Models, and VHDL Models. Observe that these languages range from very abstract to software-abstraction to hardware abstraction. . The latter are well suited for specifying existing encapsulated IP components [5]. We feel that no one language can efficiency suite all functional specifications. Figure 3 shows our modification to the system design portion of system-level methodology [3]. Matlab provides the mathematical analysis tools, functions and modeling for the analog or signal functional objects and components within the system. C++ provides the software modeling and synthesis language for sequential algorithm constructs. VHDL provides the hardware modeling and synthesis language for some sequential algorithms and concurrent algorithms.

4.1 DESIGN MODELS AND TOOLS

The initial functional specification for the embedded system uses all three languages. The functional behavior specification is written in the language that best fits the particular specification. If in the exploration, it is required to move from one language to the next, software bridges are provided. These bridges allow execution of all languages as a single specification. A subroutine stub was written in each language, which provided input/output file data between the languages. For example, the analog filtering

algorithm was described with Matlab. The analog to digital interfaces and serial interface was described in VHDL. The serial host communication protocol was described with C.

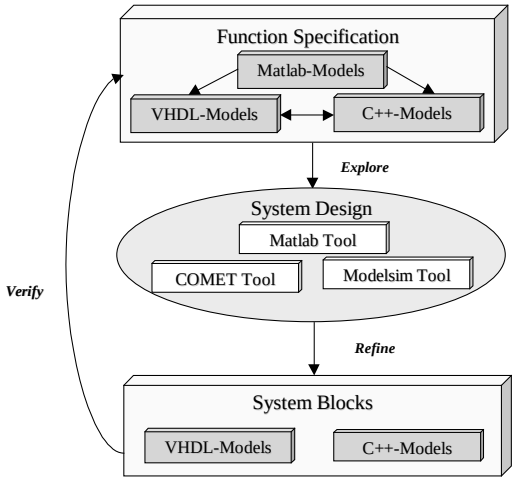

Figure 5. Design models and tools.

Given the Functional Specification, The Math Works Inc. Matlab program, COMET [6] program and Mentor Graphics Modelsim program were used as the system design exploration and refinement tools to generate the system blocks. Matlab allows functional behaviors to be described in many ways, such as algorithms or differential equations, which can be solved numerically or symbolically. This favors an analog description of functional behaviors. Matlab models can explore the analog/digital codesign issues. The COMET tools/framework was used to explore hardware/software codesign issues within the embedded system and refine the C++ Models and VHDL Models. The ModelSim allowed VHDL exploration.

Verification was achieved by several methods. Test benches were created for the complete system, which was transformed into VHDL code. This was done using Mentor Graphics VHDL simulator, Modelsim. The analog model portions were translated into VHDL as behavioral. Writing test benches in VHDL was a good choice, in that it exploited the property that VHDL is used in the later component implementation step. The Synopsys Design Compiler after synthesis can also output a VHDL gate-level file. Hence, the system design test bench can be applied again at the component implementation for further verification. Furthermore, the AVANT! Floorplan manager via software bridge can back annotate actual timing data from the embedded system routing for an additional reuse of the same digital system design test bench which now does timing violation verification. Thus, the system design test bench first verifies functional specifications, component implementation specifications, and finally gates level timing violations.

Analog verification was handled via bi-directional software bridges from VHDL to Matlab and SPICE. The Matlab tool provides a rich set of analysis and functions to allow for easy verification of analog signals. Test analog signals were generated in Matlab converted to VHDL format, simulated in VHDL and transformed back to Matlab for frequency/bode plot analysis.

4.2 Functional Models

The Matlab tools allow for frequency analysis based on a C like programming language interface. The tools have signal processing and control system library functions to easily model systems. Initially Matlab was used in the functional specification design phase. Models of blocks were generated, simulated and analyzed. As the blocks migrated into a system design and eventually into a component hardware implementation, Matlab was used to both analyze the results as well as to generate the test benches.

For example, analog data values were chosen within Matlab to meet the application. A system model was created using the Matlab functions to show the functionality of the desired application. This functional specification had many implementations. A microprocessor could implement the model. A hard-wired datapath with a state-machine could implement the model. To explore the above possible system designs, the Matlab Model was convert to another Matlab model that implemented the build-in Matlab function given parameters like 16-bit word widths. Then this architecture model was explored to generate the system design. Re-use of the Verification test-bench for these system blocks aided the exploration process. The Matlab test generation and analysis were re-used to test the component implementation models and synthesis hardware for the application.

VHDL is a simulation language of which certain sequential constructs and concurrent constructs can be simulated. As well a restricted subset of the sequential and concurrent constructs can be synthesize into hardware. VHDL also allowed for certain timing critical aspects of the design to be behaviorally described. For example, the test bench signals to test the implementation of the application could be toggled at precise time instances described easily within a VHDL test bench.

For our application the analog data sources were behaviorally modeled in VHDL. A VHDL test-bench was generated to encapsulate this portion of the application of which Matlab generated data was used as the stimulus. Since there were eight individual analog data sources, the applications interface could share a common datapath by multiplexing the incoming data or have a unique datapath for each data source for the system design.

COMET is a hardware/software codesign tool whose input consists of C, VHDL and a text-based rules file [8]. The tool was applied to determine the viability of using a microprocessor for the analog filtering or using other state-machine based hardware formatting implementation. The COMET tool allows for determination of whether the block is better implemented in hardware versus software running on a microprocessor. This tool explores the area and timing characteristic of the C and VHDL functions for a given set of microprocessors. For our application we used the COMET tools to explore hardware/software tradeoffs given either an ARM or a 386SX microprocessor.

4.3 Results

An embedded subsystem consisting of several analog inputs and a serial host interface was designed. Exploration consisted of how much to allocate to analog and to digital. The cost metric of reducing chip area and time-to-market was the most important. In this design, analog to digital codesign tradeoff was the number of bits to process the signal with the required accuracy. The more

signal bits that were required, resulted in increasing the digital area much more than the analog. This is due to the size of the multipliers, serial or parallel.

The digital filter design trade-off was how to perform a multiplication. A parallel multiply-accumulator or a serial multiply-accumulator could be created. Since we needed multiplication for the data processing of each data source this implied that we needed either a fast multiplier to handle each data source in series or multiple slow multipliers for parallel multiplication. The data widths were 16 bits. Therefore a 16 by 16 multiplier would be required of which only the upper 16 bits was of interest for the result of the multiplication. Exploration revealed that serial ALU best met the area and time constraint.

The exploration suggested more analog, but due to difficulty of mixing analog and digital simulation, interfacing between analog/digital tools and a factor of ten of analog simulation over digital, a decision was made towards using digital filtering in order to reach the design schedule deadlines.

The hardware/software codesign trade-off was to utilize a microprocessor to perform the digital task or have a state-machine digital approach. The design challenge was to handle many channels of digital data. The benefit of a microprocessor would be a single datapath with minimum utilization of RAM. However, the challenge is data bandwidth. The microprocessor would have to either employ multiple data processing paths or a high enough clock rate to handle the multiple data channels. The result was too much power, due to high microprocessor clock rate, area and the complexity for a microprocessor architecture indicated that a digital state-machine design was desirable.

The embedded systems host digital interface design trade-off was whether to utilize a vendor's asynchronous serial IP block. Given Synopsys scripts, layout area and architectural design were compared and contrasted with a custom asynchronous serial IP. The application's serial interface required a host to connect either synchronously or asynchronously. A vendor supplied IP UARTs proved to be difficult to meet the application specifications. The design required a synchronous design, and it appeared easier to design a custom IP UART.

In industrial environments, it is a goal practice to reduce the power. Component exploration resulted in running at a very slow clock frequency of 24 Mhz as to minimize the CMOS power consumption. Also, to prevent digital noise from interfering with the analog to digital conversion, digital activity was minimized whenever an analog to digital sample was taken.

5. STATUS OF WORK

Although we have implemented an embedded system, the original design methodology proceeded in mixed formal and ad-hoc fashion to a completely formal system. At this point, we are currently working on automating the system design exploration. Some of the authors are in collaboration with industry, but due to the proprietary nature of industrial embedded system, many details where left out.

6. CONCLUSION

By using the proposed design methodology using Matlab, C++ and VHDL as conceptual models, we were able to specify a system-level design for an actual industrial application. This was explored and refined first into analog/digital codomains and then proceeded hardware/software codomains. Future work, will investigate into alternative partitioning approaches between analog and digital.

7. REFERENCES

[1] Adams, J.K. and Thomas, Donald E. "The Design of Mixed Hardware/Software Systems," in *Proc. 33th* DAC, 1995

[2] Ernst, R., Henkel, J., and Benner, T., "Hardware-Software Cosynthesis for Microcontrollers," *IEEE Design & Test of Computers*, vol. 10, no 4, pp. 64-75, 1993.

[3] Gajski, D. D., Vahid, F., Narayan, S., Gong, J., Specification and Design of Embedded Systems, Prentice Hall, 1994.

[4] Gajski, D. D., Vahid, F., and Narayan, S., Gong, A System-Design Methodology: Executable-Specification Refinement, EDAC, 1994.

[5] Gupta, R. K. and De Micheli, G., "Hardware-Software Cosynthesis for Digital Systems," *IEEE Design & Test of Computers*, vol. 10, no. 3, pp. 29-41, 1993.

[6] Kalavade, A. and Lee, E. A. "Hardware/Software Codesign Methodology for DSP Applications," *IEEE Design & Test of Computers*, vol 10, no 3, pp. 16-28, 1993.

[7] Keating M., Bricaud P., Reuse Methodology Manual For System-on-a-Chip Design, Kluwer Academic Publishers, 1998.

[8] Knieser, M. J., Papachristou, C., A., "COMET: A Hardware-Software Codesign Methodology," *European Design Automation Conference, September 1996*, pp. 178-183.

[9] Vahid, Frank, Hardware/Software Codesign of Embedded Systems, APCHDL'97, Hsinchu, Taiwan, August 18-20, 1997.

[10] Vahid, Frank, and Givargis, Incorporating Cores into System-Level Specification, International Symposium on System Synthesis, December, 1998.

[11] Yen T.-Y., and Wolf, W., "Sensitivity-Driven Co-Synthesis of Distributed Embedded Systems," in *Proc. 8th Int. Symposium on System Synthesis*, 1995.