Hardware/Software Co-Design of Embedded Real-Time Systems from an Undergraduate Perspective

Kevin C. Kassner*
RF & Electronic Systems Department
Dynetics Corporation
Huntsville, AL 35806, USA
kevin.kassner@dynetics.com
*Contact author

Kenneth G. Ricks
Electrical and Computer Engineering
The University of Alabama
Tuscaloosa, AL, 35487, USA
kricks@coe.eng.ua.edu

Abstract

The increasing complexity of embedded systems parallels the difficulty of adequately preparing students to design them. Two topics key to the success of a graduate in the area of embedded systems are hardware/software co-design and real-time computing. This paper serves as a case study describing how an undergraduate applied hardware/software co-design in the design of a spectrum analyzer with real-time constraints for a Capstone senior design project. The goal of this work is to produce a co-design approach more suited for undergraduates having little design experience.

1. Introduction

How can we prepare our electrical and computer engineering students to design embedded systems? There is so much material to cover at the undergraduate level it hardly seems possible to adequately prepare students for a career in embedded systems development. Thus, educators are faced with the difficult task of selecting a subset of critical topics to include in their curriculum. Two critical topics are hardware/software (HW/SW) co-design and real-time computing. In spring 2004 The University of Alabama offered for the first time an embedded systems class at the undergraduate level. An educational result of this course was the design of a spectrum analyzer with real-time constraints which was successfully completed December 2004 as a Capstone Design project. This paper is an examination of how HW/SW co-design was employed in an undergraduate design class. Completion of such a project suggests that a student is well prepared for a career in embedded systems development. The remainder of this paper is organized as follows. First, some background material is presented describing HW/SW co-design. Traditional implementations are presented that lead to a customized implementation implemented by the author. A short description is then given about the specific design project undertaken in this effort. This is followed by a detailed description of the custom HW/SW co-design technique as applied to this specific project. Finally some conclusions and observations are made.

2. Background

Hardware/software co-design is a design methodology which exploits the synergism of hardware and software through their concurrent design [1] and achieves this by delaying the allocation decision. Hence, as much as possible is known about the system prior to allocating pieces of the system to the hardware or software domains. This methodology has two primary advantages; more time to evaluate tradeoffs and it creates better hardware/software interfaces. However, it requires engineers to be familiar with both hardware and software caveats. Any design methodology should:

- provide a checklist for the design process
- facilitate the communication of design team members
- help to predict costs
- aid in the creation of a working prototype
- aid in the creation of a timeline for the development cycle
- help with the identification of metrics
- aid with requirements specification, and
- assist with the development of test procedures.

The goal of HW/SW co-design is to do all of these things as well as allow designers to "predict" implementation, "incrementally refine" a design over "multiple levels of abstraction", and create a "working first implementation" [2]. HW/SW co-design is a cyclic design methodology. Implementations of HW/SW co-design are as varied as embedded systems

themselves. Institutions and individuals tailor the methodology to fit their application and institutional framework. All these different implementations make it difficult to apply co-design, especially for an undergraduate student having limited design experience. An implementation of HW/SW co-design suitable for an undergraduate applying it (the methodology) for the first time was needed. To meet this requirement a custom version (shown in Figure 4) based upon Wolf's and Axelsson's descriptions of HW/SW co-design was created [2, 4].

Wolf's and Axelsson's implementations of HW/SW co-design are presented here for reference and comparison to the author's version. In [2], Wolf divides co-design into four major tasks:

- *partitioning* the function to be implemented into smaller, interacting pieces;
- *allocating* those partitions to microprocessors or other hardware units, where the function may be implemented directly in hardware or in software running on a microprocessor;
- *scheduling* the times at which functions are executed, which is important when several functional partitions share one hardware unit;
- *mapping* a generic functional description into an implementation on a particular set of components, either as software suitable for a given processor or logic which can be implemented from the given hardware libraries.

In [3] Wolf also describes HW/SW co-design in the following way: "Front end activities such as specification and architecture simultaneously consider hardware and software aspects. Similarly, back-end integration and testing consider the entire system. In the middle, however, development of hardware and software components can go on relatively independently – while testing of one will require stubs of the other, most of the hardware and software work can proceed relatively independently" [3]. A block diagram of the co-design process from [3] is shown in Figure 1. Wolf's two descriptions of HW/SW co-design are very different, yet they both demonstrate the core concept of delayed allocation.

Though the cyclic nature of co-design is missing from Figure 1, it is demonstrated in Axelsson's diagram shown in Figure 2. The structure of Figure 2 also emphasizes the delayed allocation decision by including allocation as a separate task in the design flow diagram. Axelsson [4] defines the tasks in his figure as follows:

- *System behavioral description*, giving an executable specification of what the system is supposed to do.

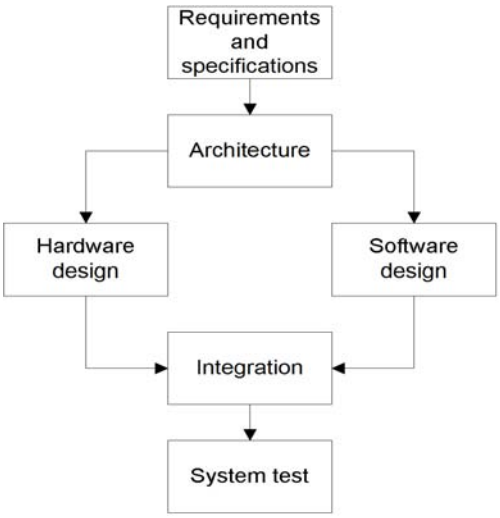

Figure 1. A simple HW/SW co-design methodology [3].

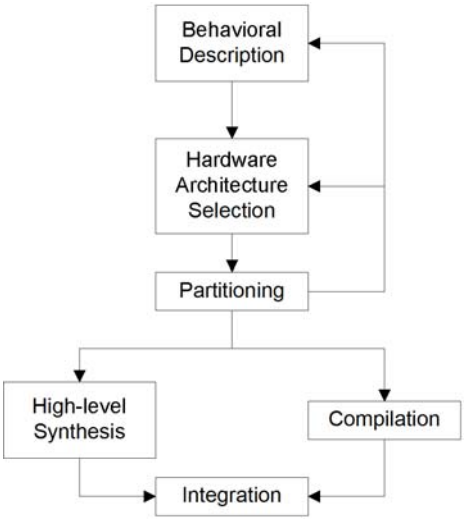

Figure 2. Axelsson's diagram of HW/SW Co-design [4].

- *Hardware architecture selection*, describing what hardware components should be used and how they should be connected.
- *Partitioning*, deciding which parts of the system behavior should be realized by what parts of the hardware architecture.

Please note that Axelsson's use of the term partitioning is analogous to our use of allocation thus far.

Figure 3 is a comparison of Axelsson's design flow diagram and a typical top-down model. This figure illustrates the advantage of a detailed behavioral description that is domain independent; the more information known about a system prior to hardware architecture selection the better.

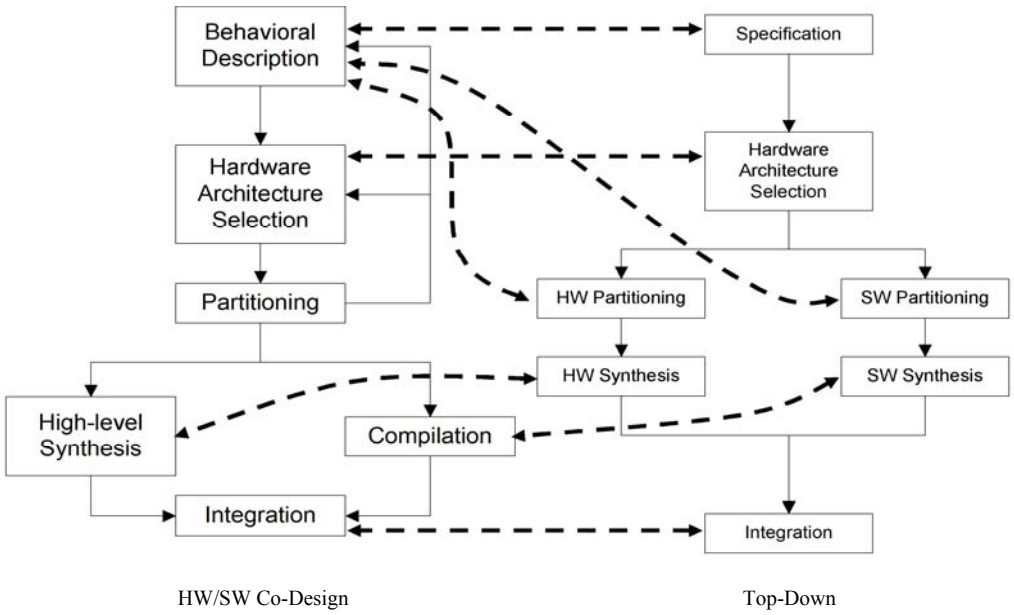

Figure 3. Axelsson's diagram versus a typical top-down model.

Figure 4 shows the author's flow diagram for HW/SW co-design. The nomenclature used here is slightly different from that of Wolf and Axelsson.

- *Specification*, usually consists of a collection of metrics, both functional and non-functional, which provide a precise description of the top-level system attributes and requirements. Examples of metrics include throughput, latency, unit cost, NRE cost, power consumption, maintainability, and time-to-market.
- *Partitioning* is the action of breaking the system functionality into small domain-independent, concurrent and interacting/communicating processes. The size of the processes is called the granularity. The result of the partitioning step should be a fully defined behavioral description of the system, with well defined interfaces between processes. Performance requirements for the processes such as frequency, throughput, and latency should also be defined.
- *Allocation* is the action of assigning each process to either the hardware domain or the software domain. Communication bandwidth alternatives/limitations between hardware and software should be considered. For example, two processes exchanging lots of data frequently would likely best exist in the same domain.
- *Hardware Architecture* means describing what hardware components should be used and how they should be connected [4] to support the execution of the processes.
- *Mapping* is the selection of specific hardware components and mapping the processes onto parts of the hardware architecture. This includes mapping processes from the software domain to the processor(s) on which they will be executed. Much consideration should be given to the execution requirements of the processes. Manufacturability should be considered during component selection.
- *Synthesis* is the implementation of the hardware and software processes for the selected hardware.
- *Integration* is the recombination and testing of processes and interfaces after implementation.
- *Scheduling* is the assignment of resources to all system processes such that their execution requirements are satisfied including inter-process communication dependencies.

Those who are familiar with HW/SW co-design may not see the need to break the design process down into this many steps. However, undergraduates find this decomposition beneficial because it requires one to think about each step separately and consider trade-offs that may not have otherwise be considered. Figure 5 shows how this design flow compares to Axelsson's. Allocation is placed above hardware architecture because the allocation process provides helpful intuition going into the hardware architecture selection. This was done even though the first hardware architecture selection usually causes some immediate feedback into the allocation.

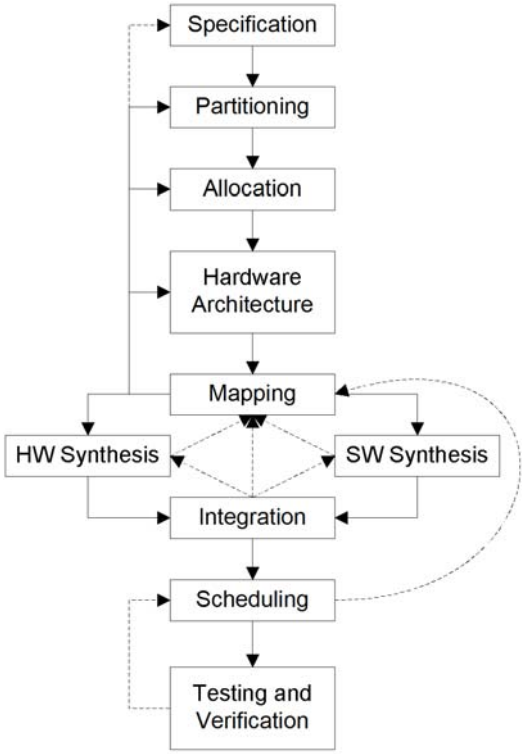

Figure 4. Customized diagram of HW/SW co-design. Dashed arrows indicate feedback paths that may not occur in every design.

Scheduling appears near the end of the design process, though a system schedule is defined in the partitioning step and considered throughout the design process. The finer granularity of the design tasks makes them more manageable for an undergraduate without much intuition gained through experience. The direct correlation to the definitions listed above serve as a reference to keep the student on track during each design task. For these reasons this design flow is believed to be much more accessible to undergraduates applying HW/SW co-design for the first time. The remainder of this paper is a case study of how this customized HW/SW co-design methodology was used in the design of a spectrum analyzer with real-time constraints for a Capstone senior design project.

3. Project Background

The project under examination is the design of an FFT based low-bandwidth real-time spectrum analyzer. The inspiration for the project was an ASIP designed by SiWorks Inc. This FFT processor is capable of computing a 1024-point FFT in just 250 clock cycles. Unfortunately these chips were not available for purchase during the initial stages of the design project. Ultimately the implementation technology used to compute the FFT was an FPGA. This resulted in a computational bandwidth well beyond that of our specifications and the analog interface. The customer for the design was the Department of Electrical and Computer Engineering at The University of Alabama for use in sophomore and junior level laboratories. The goal of the project was to design and build a beta prototype of a stand-alone spectrum analyzer with these basic requirements:

- enough bandwidth to view the spectrum of ADSL signals
- a flexible input interface for general purpose use
- VGA interface
- $300 proposed maximum unit cost per thousand

The user interface and VGA resolution details were not specified. One of the primary metrics was the real-time requirement. These goals were met and surpassed with the exception of some op-amp stability issues and one known firmware bug. The specifications of the completed system are listed in Table 1.

Table 1.
Specifications of Completed System

Real-Time	-- Input data stream sampled continuously -- Every sample must be processed -- No results are to be discarded
FFT size	1024 points
Frequency Range	0 to 1.10 MHz
Resolution	1.95 kHz
Sample Frequency	4 MHz
Input Voltage Range	0 to 100V_{peak}
Input Impedance	1MΩ, 20pF
Input Range Selection	Automatic
System Latency	-- 0.5 ms (input to video processor) -- 80 ms (input to display)
Configuration Interface	PS/2 Mouse
Output Interface	VGA (640x480x6-bit color)
Power Source	Single Phase, 120V, 60Hz
Manufacturability	No BGA or leadless chip packages
Unit Cost per Thousand	$87.44

4. Implementing HW/SW Co-Design

The original ad-hoc system diagram that was created prior to the application of HW/SW co-design is shown in Figure 6.

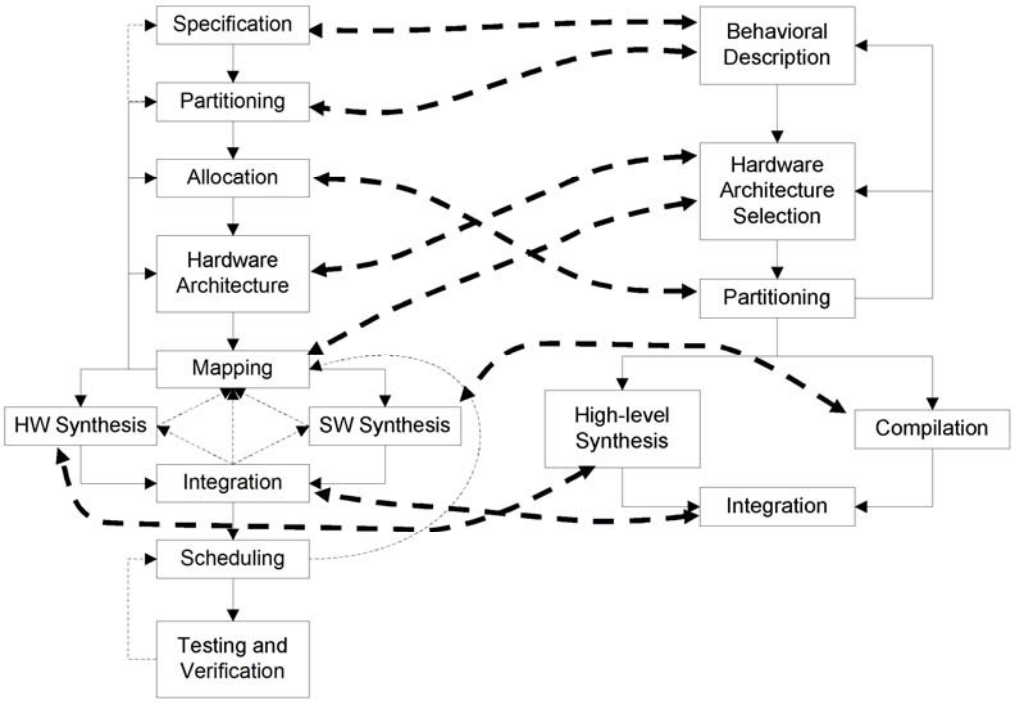

Figure 5. Customized design flow versus Axelsson's.

It is evident from the figure that partitioning, allocation, and hardware architecture selection were all occurring simultaneously. Early in a design process very little is known about how the system will function; therefore, at that point it is dangerous to attempt to define a hardware-architecture to support the operation of the system. Instead, Figure 7 shows a system partitioning resulting from a co-design approach. The immediate advantage of applying HW/SW co-design is a domain and architecture independent partitioning. Figure 8 shows one component of the system, the system control unit, decomposed into its constituent parts. This is the progression of partitioning that should continue until the processes are simple enough that they are readily implemented and the interfaces between them are fully defined, representing a system having the desired granularity. The partitioning step is also the time to define performance requirements for the processes such as frequency, throughput and latency. These will be important factors to consider in the mapping step to ensure that the final scheduling process will be successful.

During the allocation, hardware architecture, and mapping stages many tradeoffs must be analyzed before settling on a particular system implementation. It is during these stages of the co-design process that decisions must be made that may ultimately affect the partitioning and even the system specification. These are the feedback loops built into the co-design process that lead to multiple iterations through this process before project completion. For example, the original intent was to use an FPGA to implement a custom optimization of the FFT algorithm to achieve the desired performance. However, during initial *hardware architecture* selection it was realized that a sufficiently large FPGA would be cost prohibitive. The next alternative explored was an FFT ASIP although those found were not available (Zarlink PDSP16510, I&C Tech. STARFFT). Finally it was decided to use a DSP, the TI TMS320C6711, which is a 272-pin BGA device. It met the minimum performance requirements, was inexpensive and readily available. Having made this selection required a *re-partitioning* of the system. This second top-level partitioning is that shown in Figure 7. As part of the *allocation*, each process was assigned an anticipated implementation technology. At this point dual-port RAM was the chosen implementation technology for buffering. Unfortunately, dual-port RAM is very expensive in sizes as large as 1Kbyte. The memory did not need to be random access, so a 2Kbyte FIFO from TI, SN74V235, was used instead.

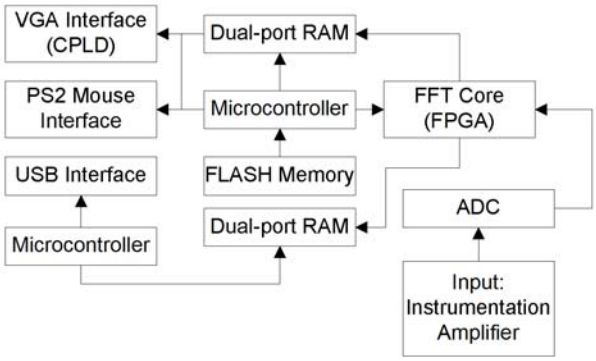

Figure 6. Original ad-hoc system partitioning.

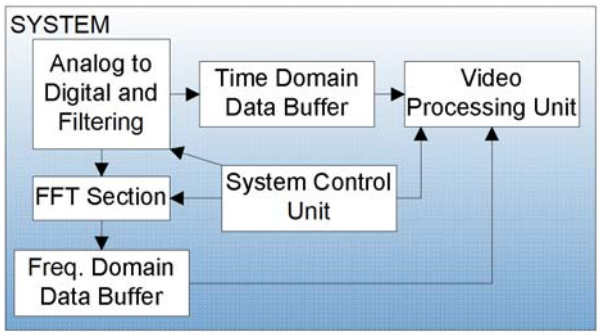

Figure 7. Co-design top-level system partitioning.

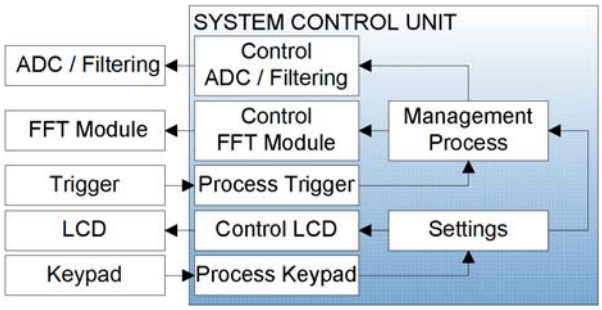

Figure 8. Partitioning of the system control unit.

As another example, there were concerns about the thermal characteristics of the circuit board and difficulty in mounting the device prior to proceeding with the *hardware architecture* using the TMS320C6711. The *specifications* were changed to include manufacturability, which meant no BGA parts. This required another tradeoff to a different DSP device, the TMS320C5402 which comes in a 144-pin QFP. This processor is capable of computing the FFT at an input sample rate of greater than 2MHz. The last frequency bin in the FFT corresponds to $F_{sample}/2$ providing a 1MHz bandwidth, just barely satisfying the minimum performance requirements. Therefore this change did not affect the *partitioning*, *allocation,* or the hardware architecture used for the TMS320C6711.

As another example of these feedback loops through the co-design process, changes were required to prevent aliasing. In order to prevent aliasing (frequencies above $F_{sample}/2$ from wrapping around into the low end of the spectrum), a low-pass filter was needed to attenuate the frequencies above $F_{sample}/2$ to less than the LSB of the input data; the input data being the output of a 10-bit ADC. However, the -3dB point of the filter needed to be 1MHz or higher to meet the performance requirements. To meet the minimum performance requirements two DSPs would need to be used in parallel, each one processing every other set of data, to compute the FFT. By putting two DSPs in parallel and using a six-pole Bessel low-pass filter the F_{sample} would be 4MHz. With this new configuration the -3dB point was calculated to be 1.10MHz. Again, these changes would ripple through all phases of the co-design process resulting in a new partitioning, allocation, and hardware architecture shown in Figure 10.

One final example of the need for feedback in the co-design process resulted from the introduction of new technology midway through the design process. In this case, it was discovered that Altera had recently made an FFT IP core available on their web site. The FFT IP core could be configured as a streaming FFT (one input and one output every clock cycle), meaning the entire system could be pipelined requiring little additional memory for buffering and greatly simplifying the overall implementation of the system. It was also determined that the Altera Cyclone EP1C12Q240C7, the largest FPGA offered by Altera or Xilinx and available in the QFP package, was available and within budget. The embedded memory blocks in the cyclone line of FPGAs are true dual-port RAM. A review of the *partitioning* showed that switching from the DSPs and external FIFOs would not require changing the algorithms; disregarding those for the VGA interface which would in fact be simplified due to the interfaces being completely internal to the FPGA. The decision was made to change the mapping to make use of this new technology. This resulted in yet another cycle through the co-design process starting with partitioning and continuing through allocation, hardware architecture, all the way to the system integration, scheduling and testing phases.

The final top-level system *partitioning* using the FPGA device is shown in Figure 9. The final partitioning shows remarkable similarity to the original system partitioning shown in Figure 7, with the

exception that Figure 9 has significantly more detail at the top level. Finally, a screen shot of the output of the system and a photo of the finished spectrum analyzer are shown in Figure 11.

5. Conclusions

This case study demonstrates that the application of HW/SW co-design can be employed in senior design classes to increase the complexity of projects accomplishable by undergraduate students. The custom HW/SW co-design process presented here should be applicable to any embedded system. The structure of the design flow diagram and the accompanying definitions make it ideally suited for undergraduates.

6. References

[1] G. De Michell, R. K. Gupta, "Hardware/software co-design", Proceedings of the IEEE, Vol. 85, no. 3, March 1997, pp. 349.

[2] W. H. Wolf, "Hardware-Software Co-Design of Embedded Systems", in Proceedings of the IEEE, Vol. 82, no. 7, July 1994, pp. 967-989.

[3] W. H. Wolf, *Computers as Components, Principals of Embedded Computing System Design*, Morgan Kaufmann, New York, New York, 2001, pp. 502–503.

[4] J. Axelsson, "Hardware/Software Partitioning of Real-Time Systems", IEE Colloquium on Partitioning in Hardware-Software Codesigns, February 13, 1995, pp. 5/1-5/8.

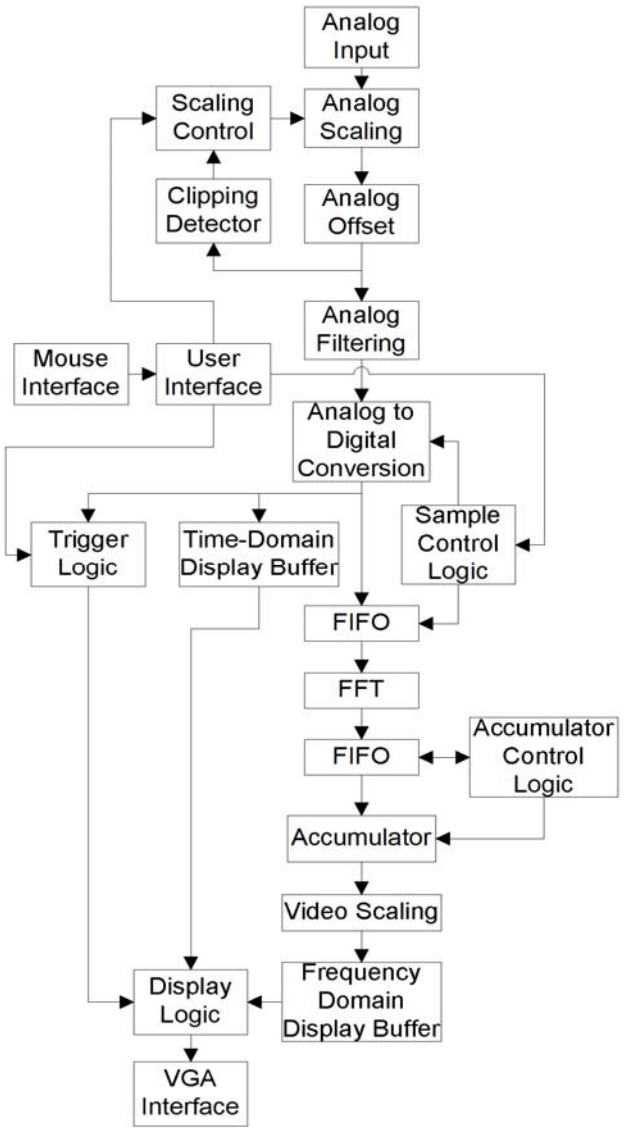

Figure 9. Final top-level system partitioning.

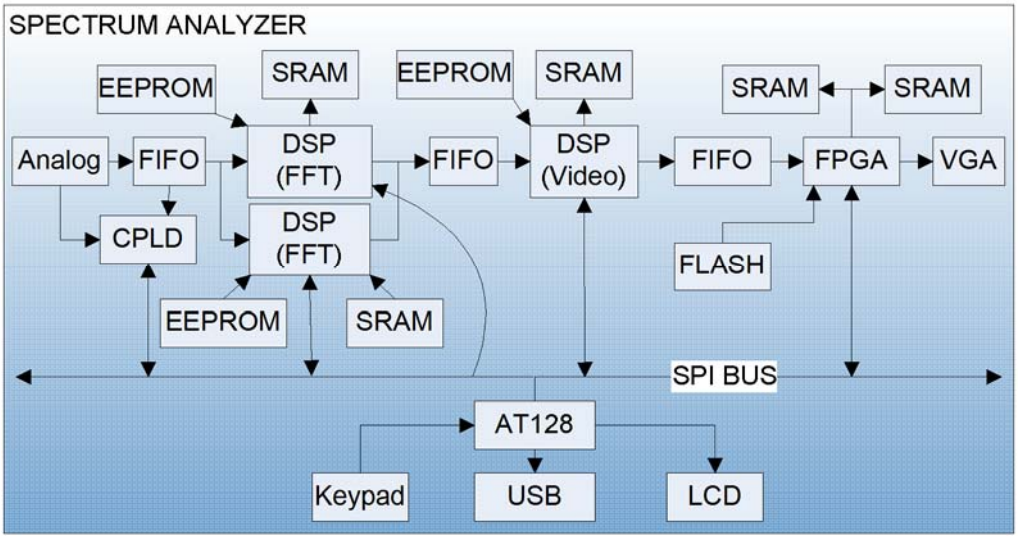

Figure 10. One version of the hardware architecture.

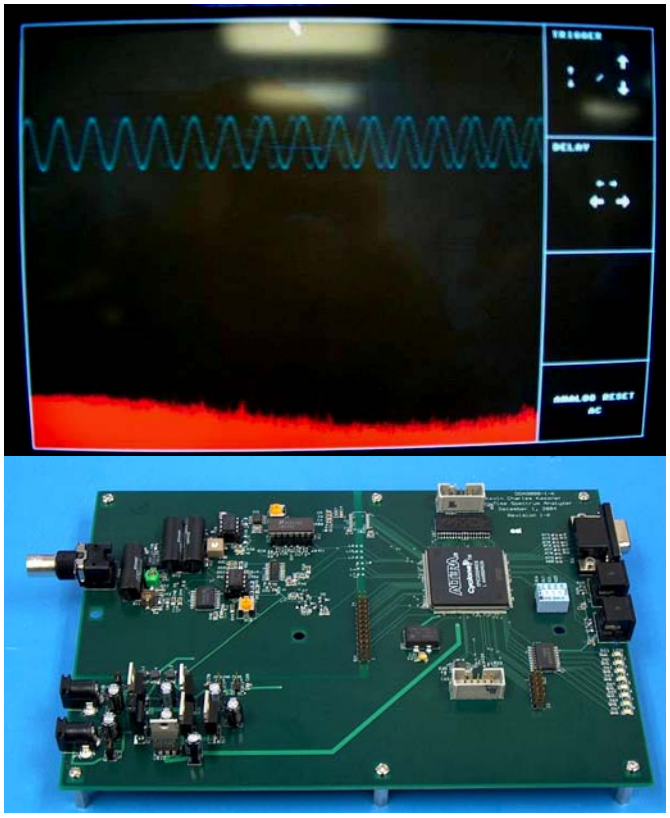

Figure 11. System output (top) and completed spectrum analyzer (bottom).