

# EVALUATION OF CPU UTILIZATION UNDER A HARDWARE-SOFTWARE PARTITIONED ENVIRONMENT (MIGRATING SOFTWARE TO HARDWARE)

Hsiang-Ling Jamie Lin and Jabulani Nyathi

School of EECS  
Washington State University  
102 Spokane Street  
Pullman, Washington, 99164

E-mail: [jamiehl@gmail.com](mailto:jamiehl@gmail.com), [jabu@eeecs.wsu.edu](mailto:jabu@eeecs.wsu.edu)

Clint Cole

Digilent, Inc.  
P.O. Box 428  
Pullman, WA 99163-0428

[ccole@digilentInc.com](mailto:ccole@digilentInc.com)

**Abstract** - *The embedded systems application space is growing at a fast pace and has a very wide range that encompasses minute sensor nodes through large FPGA based systems with multiple embedded processors within a single chip. The use of real-time operating systems (RTOS) has become pivotal in embedded systems design because RTOSes foster multi-tasking flexibility through the use of the operating system's services. Having a number of tasks running concurrently leads to increased CPU utilization, even with the existence of a number of task scheduling techniques it is still essential to mitigate CPU utilization. The art of embedding a processor or processors in an FPGA offers the system additional computational resources that could alleviate the processor loading by migrating some of the computational needs onto the hardware. In this paper hardware/software partitioning is explored with hardware oriented computations performed in the logic while those suitable for execution by the CPU are performed in software. Digital adaptive filtering is considered to demonstrate the benefits and flexibility offered by the integration of processors and logic on the same die. A well partitioned finite impulse response filter is shown to outperform a software based filter; executing 3.6 times faster.*

**Keywords:** CPU-loading, Hardware/Software partitioning, FIR filter, high-performance, embedded processor, FPGA.

## 1 Introduction

Traditional FPGAs have no processors and are designed to be generic requiring significant silicon real-estate to accommodate a variety of applications. They are configurable and programmable digital logic, can instantiate multiple processing units to run tasks in parallel, they have hardware acceleration for high speed tasks, are scalable and coded in hardware description languages (HDL).

This work is supported in part by donations from Digilent Inc., <http://www.digilentInc.com>.

Microprocessors on the other hand have fixed processing units accessed by predefined commands, they require greater knowledge, tools, installation, and engineering familiarity; they have a faster development time, lower parts costs and are coded in high level languages such as C and C++. The Virtex-II Pro development system combines both programmable logic and microprocessors offering a high degree of flexibility and diversity in usage. Fundamentally, microprocessors and FPGAs fulfill different needs/tasks as outlined above. Programmable logic is needed for wire speed, high bandwidth, custom digital processing, and hardware acceleration. However, that speed and customization comes at a price, both in parts and nonrecurring engineering costs.

Xilinx's Virtex-II Pro development system has two embedded IBM power PC cores, a software based microprocessor (MicroBlaze), memory blocks, input/output (IO) blocks, digital clock managers and a number of multiply-accumulate units. The Virtex-II Pro has Configurable Logic Blocks (CLBs) connected using a rich set of resources. By embedding processor cores within the FPGA fabric, the Virtex-II Pro architecture provides tight coupling between high-performance processors and the high-speed programmable logic [1]. Together, the two components enable the most optimal yet flexible partitioning of hardware and software in a programmable system. Each IBM PowerPC processor runs at 300 MHz or more and performs 420 Dhrystone MIPS [1]. Even though the PowerPC 405 core occupies a small portion of the die area, it provides tremendous system flexibility. Instead of attaching the PowerPC 405 processor next to the FPGA with a bus interface, the Virtex-II Pro engineering team embedded the processor entirely within the FPGA fabric. Using Xilinx IP Immersion and Active Interconnect technologies, hundreds of processor nodes can be directly connected to the FPGA logic and memory array [2].

Of particular interest in this paper is the investigation of the benefits and efficiency of system on a programmable chip, especially the hardware software partitioning. With the Virtex-II Pro FPGA, we will be able to migrate some tasks

from the software to the hardware for computational efficiency. For an example, the hardware multipliers can be used to improve performance of multiplication, which is slow if done in a processor. We can easily reduce the workload for processors by having hardware handle certain tasks, if such hardware has better performance over processors for the particular tasks. Since each processor has its own memory, we can have the processors perform independent tasks or work together towards a common goal. This research examines the above aspects on the Virtex-II Pro development system.

In this paper we highlight benefits of integrating the programmable logic with embedded processors by performing experiments that compare a design implemented in software and the same design partitioned to operate in both software and hardware. Performance measurements are in terms of number of CPU cycles spent in completing the tasks under each of the environments. Real-time processing is yet another important aspect of an embedded system and it is shown in this work how an adaptive finite impulse response (FIR) filter can be implemented on the V2Pro development system. The example design considered is compute intensive and depending on the application could require timeliness for proper functionality. Section 2 of the paper presents a brief on digital signal processing in an effort to justify the selection of the FIR filter as a design example. Section 3 highlights the well known features of FIR filters justifying their selection over their counterpart infinite impulse response (IIR) filters. It is also in Section 3 where details of the experimental set up are provided. In Section 4 performance results are presented while in Section 5 some concluding remarks are provided.

## 2 Digital Signal Processing

Digital signal processing provides probably some of the most compute intensive applications in engineering. Though the Virtex-II Pro development system is not primarily intended for digital signal processing, it has such powerful computational capabilities that can meet digital signal processing applications' heavy computational demands. In order to show how CPU loading can be improved and further demonstrate the hardware software partitioning concept we have chosen to implement a simple digital finite impulse response (FIR) filter as a design example. Hardware and Software partitioning of the FIR is currently done manually even though there exist algorithms to automate the partitioning function [3], [4], [5]. Efforts are under way to identify existing hardware/software partitioning or co-design tools and to select an appropriate one matching our design objectives. So far POLIS is the tool under consideration [6]. Real-time processing is another important aspect of an embedded system and digital signal processing lends itself well to experiments on this aspect of embedded systems. Applications such as noise cancellation and unknown system identification require the use of adaptive filters. Adaptive filtering reacts to run-time

events and the FIR filter will be used to experiment with real-time responsiveness and evaluate the benefits of integrating the microprocessor on the same die as the FPGA. Finite impulse response filters have been chosen over the infinite impulse response (IIR) filters to demonstrate these important embedded systems design issues on the Virtex-II Pro development system.

The FPGA and embedded processors co-existence feature of the Virtex-II Pro FPGA enables for the migration of the multiplication part of the FIR filter to the FPGA while memory management, task scheduling and switching of coefficients will be left as software tasks performed by the embedded core. This constitutes hardware and software partitioning. In simple terms hardware software partition involves being able to identify an application's components that can be better performed in hardware and those that can be better performed in software, and dividing them to compute in their respective units (hardware or software). The problem of hardware software partitioning has seen over a decade of activity and some of the notable work includes that presented in [7], [8]. This problem has been made easy by the development of logic systems with embedded cores. This study focuses on hardware/software partitioning in order to show how CPU loading can be alleviated. CPU utilization requires close monitoring particularly in multitasking environments with critical tasks. Reduced CPU loading is desirable as this allows time critical tasks to become more schedulable, hence meet deadlines. Figure 1 is a representation of the architecture of the software based as well as the hardware/software FIR architecture. In this study a 30-tap FIR has been implemented and the results reported are based on this tap size.

Figure 1 shows a comparison of the software and hardware/software 30-tap FIR architectures. The software architecture has fixed data width and its serial data processing requires more CPU time, the hardware/software architecture on the other hand is flexible and requires a single clock cycle to manipulate all the coefficients. Parallel processing allows for a shorter processing time leaving the CPU to perform other tasks such as memory management, executing other tasks etc.

## 3 Finite Impulse Response (FIR) Filters

Finite impulse response (FIR) filters are one of two primary types of digital filters used in Digital Signal Processing (DSP) applications the other type being infinite impulse response (IIR) filters. IIR filters use feedback and each type of filter has advantages and disadvantages [9]. Overall the advantages of FIR filters outweigh the disadvantages as a result they are used much more than IIR filters. The FIR filters offer the following advantages:

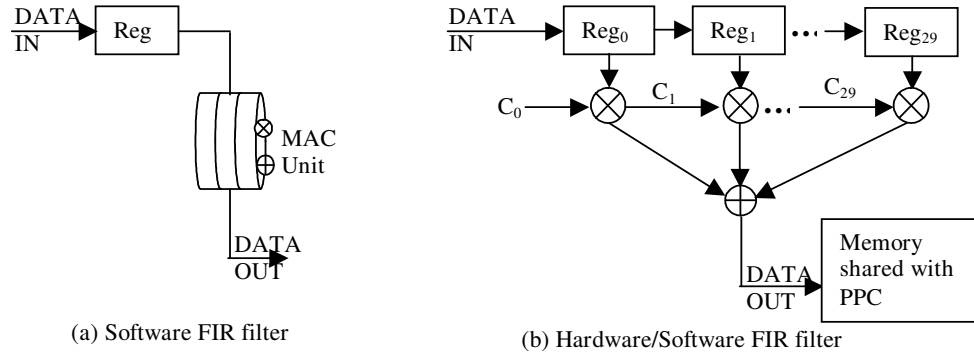


Figure 1: FIR filter implementation in (a) Software and (b) Hardware/Software

- They can easily be designed to have a linear phase, in other words they delay the input signal, but do not distort its phase.
- They are simple to implement and on most DSP microprocessors the computations can be done by looping a single instruction.
- They are easy to manipulate allowing decimation (reducing the sampling rate), interpolation (increasing the sampling rate), or both.
- Whether decimating or interpolating, the use of FIR filters allows some of the calculations to be omitted, thus providing an important computational efficiency. Coefficient symmetry also saves memory space.
- They have desirable numeric properties. In practice, all DSP filters must be implemented using "finite-precision" arithmetic, i.e. a limited number of bits. The use of finite-precision arithmetic in IIR filters can cause significant problems due to the use of feedback, but FIR filters have no feedback, so they can usually be implemented using fewer bits, and the designer has fewer practical problems to solve related to non-ideal arithmetic.
- They can be implemented using fractional arithmetic. Unlike IIR filters, it is always possible to implement a FIR filter using coefficients with magnitude of less than 1.0. (The overall gain of the FIR filter can be adjusted at its output, if desired).

Despite the highlighted advantages, FIR filters sometimes have the disadvantage that they require more memory and/or calculation to achieve a given filter response characteristic. In addition, certain responses are not practical to implement with FIR filters. Some of the most important FIR filter parameters and characteristics include:

- *Impulse Response* – this is a set of FIR coefficients. If an impulse input to an FIR filter with the impulse consisting of a “1” sample followed by many “0” samples, the output of the filter will be the set of coefficients, as the “1” sample moves past each coefficient in turn to form the output.

- *Tap* - A FIR "tap" is simply a coefficient/delay pair. The number of FIR taps is an indication of (i) the amount of memory required to implement the filter, (ii) the number of calculations required, and (iii) the amount of "filtering" to be done; in effect, more taps mean more stop-band attenuation, less ripple, narrower filters).
- *Multiply-Accumulate (MAC)* - In a FIR context, a MAC is the operation of multiplying a coefficient by the corresponding delayed data sample and accumulating the result. FIR filters usually require one MAC per tap.
- *Transition Band* - The band of frequencies between pass-band and stop-band edges. The narrower the transition band, the more taps are required to implement the filter.
- *Delay Line* - The set of memory elements that implement the  $Z^{-1}$  delay elements of the FIR calculation.

In this study there is no concentration on digital filter design expertise and the background presented is deemed sufficient to enable for the software/hardware and software implementations on the development system. Adaptive filtering is also considered to enable for an evaluation of real-time responsiveness of the system. The linear phase property of FIR filters gives a fixed amount of delay and provides no delay distortion. The symmetry of coefficients saves memory space for storage. The design of an FIR filter involves the following steps: Filter specification, Coefficients calculation, and Implementation. In this study, a basic FIR filter algorithm is used. The filter stores an input, calculates the output, and shifts the delay line. The output is described by:

$$y(n) = h(0)x(n) + h(1)x(n-1) + h(2)x(n-2) + \dots + h(N-1)x(n-N-1),$$

where  $h(i)$  represents the coefficient, with  $x(n)$ ,  $x(n-1)$ , ...,  $x(n-N-1)$  being the inputs. Since this experiment focuses on the hardware software partitioning capability of the system, the design of FIR filters is done using available tools. The source code is taken from dspguru.com [9], and modifications have been made to meet design requirements. The coefficients are generated by online FIR Filter Designer

Pro software, written by Vijaya Chandran Ramasami [9]. The Hamming Window method is used for all FIR filter designs in this experiment.

National Semiconductor's LM4550 AC97 audio CODEC on the Virtex-II Pro development system is fully supported by the Xilinx EDK. It is paired with a stereo power amplifier made by Texas Instrument. The LM4550 uses 18-bit Sigma-Delta A/Ds and D/As, providing 90 dB of dynamic range [10]. The implementation on this board allows for full-duplex stereo A/D and D/A with one stereo input and two mono inputs, each of which has separate gain, attenuation, and mute control. The mono inputs are a microphone input with 2.2V bias and a beep tone input from the FPGA [11]. In this experiment, the microphone input will be read by a PowerPC processor, the mixed data (voice and music) will be filtered and stored in the SDRAM, and the resulting signals will then be output to speakers. We have recorded a human voice (speech signal) while there is "music" playing in the background and we aim to filter the human voice and play back just the music. A high pass filter is therefore ideal in this case especially given that the frequency range of the human voice would be at low frequencies (200 – 5000 Hz) and those of music a bit higher since we are using the stereo input which handles audio bands of 20 – 20000 Hz. It is obvious that the audio bands below 5000 Hz are filtered out in this experiment.

A more complex problem would involve separating the mixed signals into the speech and the music signals. The blind source separation of real world signals is examined in [12]. In this study a control experiment involves recording the mixed signals, performing no filtering and then playing back. This enables us to determine that the filters are at least functioning as expected.

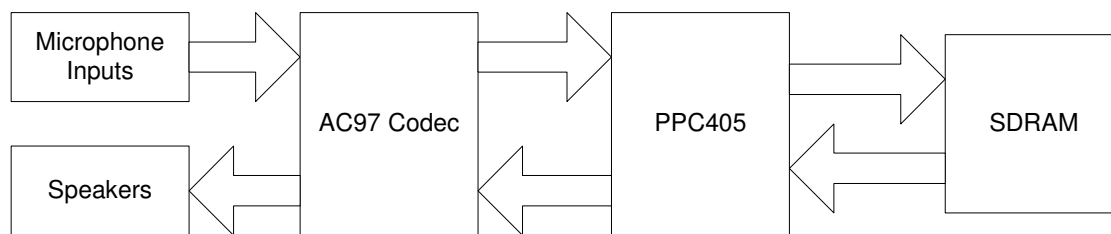


Figure 2: System flow diagram for the software based audio filtering implementation

### 3.1 FIIR Filter Specifications

The specifications of the selected FIR high-pass filter include: a pass-band frequency of 10 MHz, a stop-band frequency of 9.5 MHz, a sampling frequency of 44.1 MHz, a pass-band ripple of 0.1 and a 30 dB stop-band attenuation. The specification is based primarily on the fact that it is a human voice that will be filtered out and the pass-band frequency is well above the suggested highest value for the human voice frequency. Also the sampling rate has been influenced by the fact that we wanted to keep values within the range of frequencies the human ear can capture. Design restrictions include:

- Unsigned integers are used for input samples, coefficients, and outputs due to the fact that the embedded PowerPC processors of the Virtex II Pro development system do not have floating-point units, and any floating number computations have to be done using software emulation, use of floating numbers in the system adds significant delay and is not ideal for real-time embedded applications.
- 30 taps are used since the experiment is set-up to determine performance improvement offered by the hardware/software partitioning design approach. The filtering result is not the focus hence a fixed small amount of taps is used.

Though the accuracy of the filter is not the primary focus of this design, still the choice of the parameters ensures that the output signals are less noisy, further more the audio signals played back do confirm that the lower frequencies are filtered out. This confirms that the filter works as expected. Of significant importance are the performance gains arrived at through the partitioning of hardware and software tasks of the filter reported in the subsequent sections.

### 3.2 Software Based Audio Filtering Design

The block diagram shown in Figure 2 depicts the system event flow. The microphone inputs are converted to digital signals through the AC97 CODEC. The PowerPC processor does the filtering work, and sends the outputs to the AC97 for playback through speakers. Both inputs and outputs are stored in the SDRAM, so that the UART can retrieve the data.

Of significance in this exercise is the amount of time it takes the program to execute and playback the signal. CPU utilization is the primary metric of interest and the recorded values are compared to those obtained with partitioning implemented.

### 3.3 Hardware Software Partitioning Audio Filtering Design

The audio input and filtered output are stored in SDRAM. The FIR filter is created as a custom on-chip peripheral bus (OPB) core that attaches to the PowerPC processor. The multiply-and-accumulation (MAC)

operations are now done in the FPGA. The PowerPC processor reads samples from the audio input, sends them to the FIR filter core, reads the filtered outputs from the FPGA, stores them into SDRAM, and plays the results to the speakers. The system flow diagram for the hardware/software filter is shown in Figure 3 below. The design is different from the previous one because the filtering of audio signals is performed in the FPGA. The program structure is the same as the software based design. However, instead of having the PowerPC core computing outputs, this task is now done in the FPGA for fast computation as well as to reduce processor loading. A system with several tasks can now easily accommodate its tasks, blocking only for a short time. It must be noted that in this implementation the operating system kernel plays a significant role in scheduling events appropriately, computing the execution time and performing memory management.

Of interest to this study is CPU utilization and how much improvement could be gained by migrating some computational needs to the logic. There are some relevant definitions that need to be explained before the measurements are presented, they are:

- Sampling Time: Refers to the time it takes to read 200,000 samples from the AC97 CODEC
- Filtering Time: Is the time it takes for an input signal to be filtered
- Playback Time: Denotes the time it takes to play all filtered signals to speakers
- Adaptive Filtering: This refers to the adaptive filter processing (real-time filtering). The program takes an input, filters the signal, and plays it back, and it loops 200,000 times to process all samples. Adaptive filtering does not offer any significant performance improvement but shows the ability of the system in performing real-time computations.

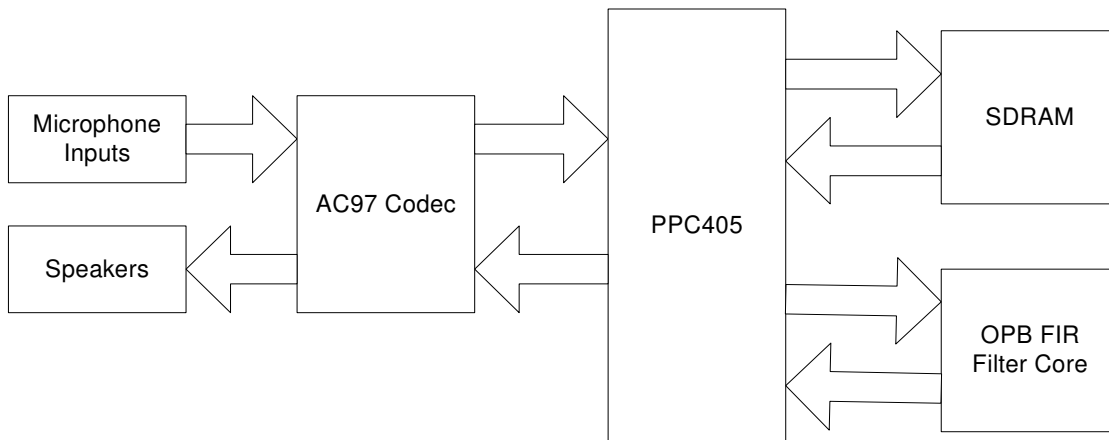


Figure 3: System flow diagram for the hardware/software audio filtering approach

## 4 Results (System Performance)

The block diagrams of Figures 2 and 3 show that there is processing at the CODEC and within the PowerPC requiring that there be different execution time measurements since each of these modules has its own clock rate. It is not that critical to record the amount of time spent sampling the data within the CODEC even though this adds significant delays in playing back the signal.

Table 1 shows the execution times in seconds within the different system modules. It must be noted that cache enabling is irrelevant when recording the different execution times except for the filtering process. Essentially sampling time, playback time and the duration of the Audio signal are features of the CODEC which runs at a different clock rate from that of the PowerPC. These times appear constant despite the filter configurations (software only or hardware/software partitioned).

Table 1: Execution time measurements for the high-pass filter

	Soft FIR Filter		Hard FIR Filter	
	Cached (seconds)	Non-cached (sec)	Cached (sec)	Non-cached (sec)
Sampling Time	4.165	4.165	4.165	4.165
Filtering Time	0.4826	4.64	0.1338	0.2301
Playback Time	4.158	4.162	4.165	4.154
	Real-Time Soft FIR Filter		Real-Time Hard FIR Filter	
	Cached	Non-cached	Cached	Non-cached
Adaptive Filtering	4.165	5.743	4.165	4.164

The filtering time is the most important metric of all since it enables for a comparison of CPU utilization when only the software filter is executed and when the multiplication is migrated to the hardware. Enabling the cache of the system brings about markedly improved results. Comparing the software based filter and the hardware/software filter with cache enabled it is determined that the hardware/software filter is 3.6 times faster. This implies that for this program the CPU is only loaded for 28 % of the time. It is apparent that the partitioning frees up the CPU and more tasks can thus be scheduled. In the event that the cache has not been enabled the software filter is even slower, with hardware/software filter being 20 times faster. For the real-time cached systems the execution times for the two filter implementations are comparable. This is due to the fact that AC97 CODEC is full-duplex, so the execution time is similar for read, write, and read/write. The main reason we do not see any performance improvement in the real-time system is because of the delay during audio compression/de-compression in the AC97 CODEC.

## 4.1 Adaptive Filtering

Applications such as noise cancellation and unknown system identification require the use of adaptive filters. Adaptive filtering reacts to run-time events, which is considered as real-time responsiveness. Design of an adaptive filter requires proper algorithms and specifications. The filter coefficients for an adaptive filter are generated at run-time in a DSP processor. For the completeness of this experiment in observing real-time responsiveness of the systems, an emulation of adaptive voice filter has been designed. The aim is to take advantage of the configurable logic to enable for run-time updating of the coefficients. The filters experimented with are 30-tap filters and according to [9] FIR filters commonly require anything from 10 to 256 taps. We consider both a low-pass and high-pass filter for this experiment. The filter specifications are tabulated in Table 2. The use of both low-pass and high-pass filters in this experiment is to show the ability to change the filter coefficients at run-time.

Table 2: Voice filter specification

	Low-pass	High-pass
Pass-band Frequency	1,000Hz	3,000Hz
Stop-band Frequency	2,000Hz	2,000Hz
Sampling Frequency	44,100Hz	44,100Hz
Pass-band Ripple	0.1	0.1
Stop-band Attenuation	30dB	30dB

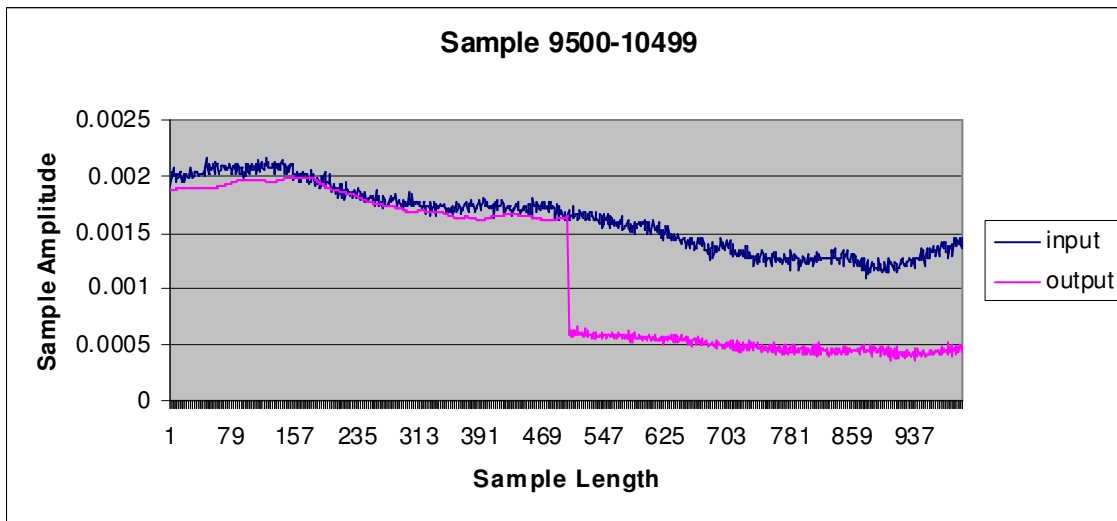


Figure 4: Input/output plot for the adaptive filter showing the effect of real-time coefficient processing

An analysis involving progressively increasing the number of samples shows that the delay increases with increasing sample size. The real-time system filter (adaptive filter) has improved sound quality and is even quieter when the filtering is done in hardware. Software filtering though not much slower than the hardware/software filter has noise that is obvious to the human ear.

The first 9,999 audio signals are filtered by the low-pass filter. The rest of signals are filtered by the high-pass filter. The filter should react to the event change at the 10,000<sup>th</sup> input signal. Figure 4 shows the plot of the real-time adaptive filtering system responsiveness.

## 5 Conclusions

From the experimental results, we can see that the filtering time is improved significantly by using the hardware in conjunction with the software compared with using just the software to implement the FIR filter. The design with the cache enabled shows the hardware/software filter executing 3.6 times faster than the software filter. The real-time responsiveness of the system is verified by implementing the adaptive filter. The overall execution time for the adaptive filter program is the same for both the software and hardware/software filters, however the reconfigurable logic allows for run time manipulation of the filter coefficients. For the adaptive filter there is no performance improvement of one approach over the other due to the significant signal compression/de-compression delay in the AC97 CODEC.

Traditionally, embedded applications are designed solely in a micro-controller. For complex applications, we usually need to make a trade-off between accuracy and performance. We can either choose to have accurate outputs in a slow system, or to lose some accuracy to achieve the real-time responsiveness. With the Virtex-II Pro FPGA, we can now achieve better performance by using the hardware for certain tasks, and still be able to maintain the accuracy. The idea of hardware/software partitioning has been experimented with for slightly over a decade with an emphasis on algorithms. In this study an application has been considered and analysis of CPU utilization performed. It has been established that hardware/software partitioning could improve task scheduling in a multi-tasking environment. Hardware and software partitioning is promoted by the emergency of system on chip and the resulting programmable logic systems such as Virtex-II Pro. Future work involves identifying tools to automate the partitioning process.

## 6 References

- [1] <http://www.xilinx.com/company/press/kits/v2pro/backgrounder.pdf>
- [2] Xilinx Inc, "Xilinx University Program Virtex-II Pro Development System Hardware Reference Manual", Xilinx Inc, March 8, 2005.
- [3] J. Henkel and R. Ernst, "An Approach to Automated Hardware/Software Partitioning Using a Flexible Granularity that is Driven by High-Level Estimation Techniques," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 9 No. 2, April 2001, pp. 273-289.
- [4] D. W. Eagles and S. Devadas, "A New Approach to Solving The Hardware-Software Partitioning Problem in Embedded System Design" *Proceedings of the 13<sup>th</sup> Symposium on Integrating Circuits and Systems Design*, September 18-24, 2000, pp. 275-280.
- [5] S. Sang, X. Li and Y. Ye, "Dependency driven partitioning objects generation for hardware/software partitioning," *Proceedings of the 2006 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 21-24, 2006, pp. 2685-2688.
- [6] <http://embedded.eecs.berkeley.edu/research/hsc/abstract.html>
- [7] D. E. Thomas, J. K Adams and H. Schmit, "A Model and Methodology for Hardware Software Codesign," *IEEE Design and Test for Computers*, Vol. 10, No 3, September 1993, pp. 5-15.
- [8] M. Baleani, A. Ferrari, A. Sangiovanni-Vincentelli, and C. Turchetti, "HW/SW Codesign of an Engine Management System," *Proceedings Europe Conference and Exhibition 2000 Design Automation and Test*, March 2000, pp.263-267
- [9] G. R. Griffin, "DSPGuru", <http://www.dspguru.com/info/faqs/firfaq.htm>
- [10] <http://cache.national.com/ds/LM/LM4550.pdf>
- [11] Xilinx Inc, "Platform Studio User Guide", Xilinx Inc, February 15, 2005.
- [12] T-W Lee, A. J. Bell, and R. Orglmeister, "Blind Source Separation of Real World Signals" *International Conference on Neural Networks*, Vol. 4, June 9-12, 1997, pp. 2129-2134.