LOW POWER SHOWDOWN: COMPARISON OF FIVE DSP PLATFORMS IMPLEMENTING AN LPC SPEECH CODEC

David Hwang, Cimarron Mittelsteadt, Ingrid Verbauwhede
UCLA, Electrical Engineering Department, 7440B Boelter Hall, Box 951594, Los Angeles CA 90095-1594
e-mail: dhwang@ee.ucla.edu, cimarron@ee.ucla.edu, ingrid@ee.ucla.edu

ABSTRACT

An identical LPC Speech Coder has been implemented on a set of signal processing specific implementation platforms. The main goal of this experiment was to compare energy consumption. In addition, area/memory requirements and design time are also compared. The coder was first designed in floating-point C. Then, the fixed-point wordlengths were determined. Depending on the platform, either compiled code was generated, assembly code written or a Verilog/VHDL design was created. The platforms reported in this paper include the DSP processors TI C55x, TI C54x, TI C6x and the design environments Ocapi and A|RT designer. Energy consumption ranges from 2 μJ to 288 μJ per speech frame. Upon scaling the results to the same technology, our results indicate that the lowest power DSP processor (TI C55x) still consumes a factor of four more energy than an application specific processor.

1. INTRODUCTION

In recent years, the technological trend toward high-performance mobile communications devices has caused a burgeoning interest in the field of low-power design. Indeed, with the proliferation of portable devices such as digital cellular phones, designing for low-power with high throughput is becoming increasingly necessary.

It is often claimed that a full-custom ASIC will be "lower power" than a programmable approach. This is certainly the case when compared to a general purpose processor, but less apparent when compared to a programmable DSP processor. The goal of this experiment was to verify this claim for a realistic signal processing application. A meaningful example, one larger than a simple FIR building block, will for the most part execute signal processing functions but will also include some control code and bookkeeping operations. An LPC speech coder was chosen for this task. It is described in Section 2.

In this paper, we investigate five *signal processing specific platforms*: three programmable DSP processors—the TI C55x, the TI C54x, and the TI C6x; and two signal processing design environments—Ocapi, and A|RT Designer. Each design was optimized to reduce cycle count and power consumption. All five designs were compared based on energy, area, clock frequency/MIPS and design time.

This paper will briefly describe the LPC Speech Coder algorithm, explain details of the design methodology, present introductions to each of the five platforms, and then discuss the final comparison results.

The authors thank the UCLA EE213A class of Spring 2000. This work is in part funded by the Fannie and John Hertz Foundation and by the Atmel Corporation /UC-Micro Grant #98-162.

2. SPEECH CODEC

Linguistically, sounds can be divided into two mutually exclusive categories: vowels and consonants. Vowels are produced by periodic vibrations of vocal chords. The period of vibrations is known as the *pitch*. Hence, excitation of vowels can be approximated simply by an impulse train with a period equal to the pitch. For consonants, the excitation is produced by air turbulence, which is approximated by a white Gaussian noise (WGN) model [4]. If every frame is classified as voiced (periodic) or unvoiced (noisy), we only need to transmit a single bit indicating voiced/unvoiced and the value of pitch period (in the case of voicing). On the receiving side, excitation can then be modeled by either an impulse train or WGN.

In order to classify each frame as voiced/unvoiced we examine the autocorrelation function. Indeed, if the frame is voiced, it must be periodic, thus forcing its autocorrelation to be periodic with an identical period.

2. 1. Algorithm

The algorithm used is due to Sondhi [5] and described below:

1. The frame is low-pass filtered at 1 kHz.
2. A clipping level C_L is set to 30% of maximum value in the frame.
3. The frame, $x(n)$, is then clipped according to:

$$C[x(n)] = \begin{cases} +1 \text{ if } x(n) > C_L \\ -1 \text{ if } x(n) < -C_L \\ 0 \text{ otherwise} \end{cases}$$

4. Finally, the autocorrelation function $R(n)$ is computed on the clipped frame $C[x(n)]$ according to:

$$R(k) = \sum_{n=0}^{N-K-1} C[x(n)] \cdot C[x(n+k)]$$

0-7803-7041-4/01/$10.00 ©2001 IEEE

where N is the length of the frame. Since minimum and maximum pitch frequencies for men and women are 80 Hz and 350 Hz, we only need to compute $R(k)$ for k between 22 and 100 inclusive (for an 8 kHz sampling rate).

5. If the largest peak of $R(k)$, $max[R(k)]$, satisfies:

$$max[R(k)] = 0.3 \cdot R(0)$$

the frame is classified as voiced and the index k is transmitted as the pitch period, else the frame is classified as unvoiced.

2. 2. Transfer Function (LPC Analysis)

An all-pole function H(z) is assumed:

$$H(z) = \frac{G}{\left(1 - \sum_{k=1}^{p} a_k \cdot z^{(-k)}\right)}$$

where p is the model order, chosen to be 10. The predictor coefficients a_k can be found from solving the linear system:

$$\begin{bmatrix} R(0) & R(1) & \dots & R(p-1) \\ R(1) & R(0) & \dots & R(p-2) \\ \dots & \dots & \dots & \dots \\ R(p-1) & R(p-2) & \dots & R(0) \end{bmatrix} \times \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_p \end{bmatrix} = \begin{bmatrix} R(1) \\ R(2) \\ \dots \\ R(p) \end{bmatrix}$$

where $R(k)$ is k^{th} lag autocorrelation function of a frame. The Toeplitz structure of the leftmost matrix can be exploited and the linear system can be solved iteratively with the Levinson-Durbin recursion [4]. The set of 10 linear prediction coefficients (LPC) as well as the prediction error $E^{(p)}$ are computed and transmitted for each frame.

From an implementation viewpoint, the computation intensive modules are the following:

- Pitch detection. A total of 78 correlations have to be computed—*R(22)* through *R(100)*. The computations are simplified using the "clipped" coefficients to reduce the intensity of this module.
- Levinson-Durbin algorithm. This involves 11 correlations as well as a division algorithm.

Execution times on the DSP processor show that around 60% of the cycles are spent on the pitch detection while 25% are spent on the Levinson-Durbin algorithm. The remaining 15% are used for the Hamming window, low-pass filter and miscellaneous memory transfers.

3. DESIGN METHODOLOGY

This section describes the design methodology for implementing the LPC speech coder on the various platforms. The coder was first designed in floating-point format in MATLAB. Then, the challenge was to efficiently map the software algorithm onto the fixed-point hardware. This involved the conversion of floating-point computations into fixed-point wordlengths (along with the ensuing design decisions) as well as the allocation and software mapping/ scheduling on the available hardware.

3. 1. Floating-Point to Fixed-Point Conversion

Since our design performs in real-time on fixed-point hardware, we had to make decisions concerning the internal wordlengths of each of the system hardware modules. An inadequate wordlength can lead to reduced SNR, deterioration of sound quality, and clipping. However, a surfeit of wordlength can create extraneous hardware, leading to wasted area and power.

For some of the platforms (i.e. the TI DSPs), the internal wordlengths are fixed to a particular number (i.e. 16 bits). However, on the other platforms, the wordlengths can be decided by the designer. There are several criteria which affect the fixed-point wordlength decision, including recognizable synthesized speech, pitch frequency matching, avoidance of signal overflow/saturation at each point in the algorithm, and avoidance of saturation of the synthesized speech output.

Of all these factors, the most restrictive criterion is the avoidance of synthesized speech saturation. This particular problem, related to instability (and hence the poles of the system), is inherent to the Levinson-Durbin algorithm. In the fixed-point implementation, the quality of voice is dependent on the number of input bits in a highly non-linear fashion. If the number of bits is insufficient, the algorithm is unstable and clipping occurs. On the other hand, if the number of bits is sufficient, the Levinson-Durbin algorithm is stable and the reconstructed signal is virtually the same as the floating point signal. Hence, by adjusting the wordlength parameters and checking output saturation, the minimum bit requirements for each module can be found. This iterative refinement was done on the Ocapi and A|RT Designer platforms with the built-in fixed point C++ libraries. This resulted in varying wordlengths according to the modules. Even within one module, the position of the decimal point (Q-format) is adjusted at each point in the algorithm. The hardware modules for the Ocapi implementation vary from 8-bit clipped correlator units to a 24-bit multiplier and a 30-bit accumulator.

The fixed-point processors (TI 54x, TI 55x, TI 6x) have internal wordlengths set to 16 bits for most arithmetic operations. To obtain a fixed-point C++ code suitable for such a processor, we rewrote the entire algorithm using 16-bit C arithmetic (i.e. using ANSI C short format). We then heavily modified the code to exploit the TI Q15 library function. The Q15 format maps each 16-bit word into a fractional two's complement number in the range [-1,1).

After the fixed-point code was completed using Q15 functions, data scaling needed to be performed to prevent saturation. For example, the autocorrelation function has its maximum value at $R(0)$, which itself has a worst case value of 240 (if every $C[x(n)] = 1$ for all 240 samples). This would require the scaling of each $C[x(n)]$ by 1/240 to insure that $R(0)$ remains in the range [-1,1). However, this is a pessimistic approach to scaling, as a speech pattern would

never be DC. Hence, we took a more optimistic view of scaling and scaled each *C[x(n)]* by a factor of 1/128 (2^{-7}). This allows for a greater dynamic range, while still keeping *R(0)* confined to the range [-1,1) for most cases. In the few cases this range is exceeded, *R(0)* saturates to the boundary points.

3. 2. Hardware Allocation

A|RT Designer [12] assumes a VLIW architecture, where the user is free to choose the datapath modules in the architecture. Ocapi [11] gives the user only an environment to specify the architecture and does not impose a particular architecture. This has the advantage that any architecture can be described, but the disadvantage that the designer has to describe all features and details of the architecture.

Thus in both environments, the user allocates the data path modules (ROM, RAM, ALU, MAC, etc.) necessary to complete the design, as well as designate which modules perform each function in the algorithm code. Hence, by examining processor use statistics and by keenly examining the code structure, one can pinpoint design bottlenecks and alleviate them by reallocation and reassignment.

An example of this can be seen in the iterative design flow of A|RT Designer. The initial design (using the A|RT Designer default hardware allocation) requires 8000 cycles to complete. By examining the processor use statistics, it is found that the autocorrelation function occupies 80% of the processing time. Thus, the cycle count can be reduced by 4000 cycles by inserting an additional ACU (Address Control Unit) and a second MAC in the autocorrelation routine. By further investigation of the code, one finds that the windowing filter can be reallocated onto a ROM instead of being soft coded. This modification reduces the cycle count by another 1000, bringing the total cycle count down to 3000 cycles—a 63% decrease from the original design. These are examples of the design processes required to optimize performance (cycle count) using the various tools.

4. PLATFORMS

As mentioned previously, we chose five implementation platforms, which are described briefly below.

4. 1. Texas Instruments TI C54x

The TI C54x fixed-point DSP is a signal processor commonly used in cellular phones, digital audio players, and other low-power communications devices [1]. The TI core uses an advanced modified Harvard architecture that maximizes processing power with eight buses (four program data buses and four address buses). The core consists primarily of a 40-bit ALU, a barrel shifter, two accumulators, a 17 x 17-bit MAC unit and an addressing unit. The program fetch is 16 bits and the instruction length is also 16 bits. According to [9], the power consumption is 0.32 mW/MIPS and the processor can run 30-160 MIPS.

4. 2. Texas Instruments TI C55x

The TI C55x processor is the most recent DSP in the TMS320C5000 series. It builds on the C54x generation with a one-sixth reduction in power consumption alongside a (maximally) 500% increase in performance [9]. The C55x has additional hardware, including a 17 x 17 bit MAC, a 16-bit ALU and a total of four 40-bit accumulators. These additions, together with the scaling of the semiconductor technology, allow the C55x to operate at 0.05 mW / MIPS and perform at 140-800 MIPS [9].

4. 3. Texas Instruments TI C6x

The Texas Instruments TMS320C6000 series is the line of fixed-point and floating-point processors which emphasize high-performance as the key metric. As such, they are used in base stations and other systems in which bandwidth and processing power is crucial. In our experiment, we tested the C62x processor, a fixed-point DSP used for multi-channel broadband communications. The core implements a VLIW architecture with eight functional modules. These consist of six parallel 40-bit ALUs and two 16-bit multipliers (with 32-bit outputs). The C62x processor operates at 150-300 MHz and is capable of operating at 1200-2400 MIPS [10].

4. 4.Ocapi

Ocapi is a C++ based design environment developed by IMEC [7][11]. The Ocapi environment is based upon a library of fixed-point C++ classes that allow the user to fully describe an ASIC at the highest algorithmic and behavioral level. Through different design stages, the C++ code is refined and enhanced with architectural detail. The Ocapi toolset then maps the final code into an RTL level bit-parallel HDL code which is fully capable of synthesis.

4. 5. A|RT Designer

A|RT Designer is a software environment designed by Frontier Design [2][12]. As with Ocapi, A|RT Designer's purpose is to bridge the gap between the software algorithm design and the hardware implementation. The design is first created in floating-point C and then converted to fixed-point C using a fixed-point library. Upon completion of fixed-point code, the user directs the software tools to perform resource allocation, resource assignment, and operation scheduling (based upon data interdependencies). A|RT generates synthesizable RTL level code which describes the entire VLIW machine.

5. FINAL SIMULATION AND RESULTS

5. 1. Area / Memory

In circuit design, a measure of the cost for a particular design can be estimated from the total area. Similarly, on an embedded software platform, cost can be estimated by

memory and cycle counts to perform the algorithm. In Table 1, the overall area/memory and cycle counts for each platform are summarized.

The Ocapi solution is a slightly over half the size of the A|RT Designer solution. However, these figures are somewhat deceptive. The reason for this large difference in size is mostly due to the process libraries we had to synthesize each circuit. For the Ocapi design, a 0.25 μm process was used while for A|RT Designer, a 0.35 μm process was used. Assuming perfect scalability, the A|RT Designer circuit would be only 1.63 mm^2. This is comparable to the Ocapi design area of 1.4 mm^2 as one would expect.

Table 1: Implementation results

	Area—Memory	Cycles	Energy/Frame	Technology	Power Supply
TI C5402	8.7 kB	240K	42.7 μJ	0.18 μm[a]	1.8V core 3.3V I/O
TI C5510	10.2 kB	120K	3.2 μJ	0.15 μm[a]	1.6V core 3.3V I/O
TI C6211	16 kB[b]	30K	288 μJ	0.18 μm	1.8V core 3.3V I/O
Ocapi	1.4 mm^2	11K	2.1 μJ	0.25 μm	2.5V
A|RT	3.2 mm^2	3K	4.3 μJ	0.35 μm	3.3V

[a] The technology is not specified in the technical documentation, therefore it is estimated based on the power supply.

[b] This includes only the program code, since the data memory requirement depends on the number of channels.

5. 2. Power

Power figures for each design are given in Table 1 in units of energy per frame. The circuit designed with Ocapi resulted in the lowest energy per frame with 2.1 μJ of energy consumed in one frame. Not far behind was the TI C5510, with only 3.2 μJ per frame, and the A|RT designer solution at 4.3 μJ per frame. It might seem quite shocking how close the C5510 comes to the custom design from Ocapi in terms of power, but this small difference can be explained by the difference in technology.

Scaling with a $1/S^2$ factor (S is 0.35/0.15 and 0.25/0.15) reduces the energy for the full custom designs of Ocapi and A|RT Designer to 0.76 μJ and 0.79 μJ respectively. This shows that for the same technology and the same supply voltage, the full custom application specific processors are *a factor of four lower* in energy consumption compared to the lowest power DSP processor. The $1/S^2$ scaling factor corresponds to the power scaling in a general scaling model [3]. In this application, energy scales as the power since the time frame remains the same, as this is dictated by the application.

It is worth nothing that the application specific solutions use regular standard cell libraries, thus losing some energy advantage compared to fully custom designed datapaths for the low power DSP processors.

The TI C6211 DSP had, by far, the highest energy per frame consumption with 288 μJ. Clearly, this is an unacceptable amount when compared to all the other target platforms if the C6211 were used to encode only one voice channel. However, within this energy budget, the C6211 was capable of processing 75 simultaneous voice channels. When looking at each channel separately, this amounts to only 3.8 μJ per frame.

6. CONCLUSIONS

While large efforts have been made to make programmable DSP processors extremely low power, they still trail in comparison to application specific solutions, by a factor of four in this experiment. The above results were obtained in the span of one quarter, indicating the "ease of use" of both the TI programming environment (Code Composer) as well as the design environments, Ocapi and A|RT designer. This includes installing, learning and running each of the software tools.

7. REFERENCES

[1] W. Lee, et al., "A 1-V programmable DSP for wireless communications," *IEEE Journal of Solid-State Circuits*, Vol. 32, No. 11, pp. 1766 -1776, Nov. 1997.

[2] P. Mosch, et al., "A 720 μW 50 MOPS 1V DSP for a Hearing Aid Chip," *2000 IEEE International Solid-State Circuits Conference*, pp. 238-239, Feb. 2000.

[3] J. Rabaey, *Digital Integrated Circuits: A Design Perspective*, Prentice Hall, 1996.

[4] L.Rabiner, R. Schafer, *Digital Processing of Speech Signals*, Prentice Hall, Englewood Cliffs, New Jersey, 1978.

[5] M.M. Sondhi. *New Methods of Pitch Extraction*. IEEE Trans. Audio and Electroacoustics, Vol. AU-16, No. 2, pp. 262-266, June 1968.

[6] I. Verbauwhede, C. Nicol, "Low Power DSP's for Wireless Communications," *Proc. of the 2000 International Symposium on Low Power Electronics and Design*, pp. 303-310.

[7] S. Vernalde, P. Schaumont, I. Bolsens, "An Object Oriented Programming Approach for Hardware Design," *IEEE Computer Society Workshop on VLSI*, 1999, Orlando, April 1999.

[8] www.ee.ucla.edu/~ingrid/ee213a/index.html

[9] www.ti.com/sc/docs/products/dsp/c5000/index.htm

[10] www.ti.com/sc/docs/products/dsp/c6000/index.htm

[11] www.imec.be/Ocapi/

[12] www.frontierd.com