15

Benchmarking DSP Architectures for Low Power Applications

David Hwang, Cimarron Mittelsteadt and Ingrid Verbauwhede

15.1 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, pagers and personal digital assistants, 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 Digital Signal Processor (DSP) processor [13]. An experiment has been designed to verify this claim for a realistic signal processing application in a low-power environment [2]. The goal is to quantify this claim. A meaningful example, one quite larger than a simple FIR filter or autocorrelation, will for the most part execute signal processing functions but will also include some control code and book-keeping operations encountered in many signal processing applications. A Linear Prediction Coefficient (LPC) speech coder was chosen for this task. It is described in subsequent paragraphs. Secondly, the design methodologies for each design approach, ASIC or programmable DSP, will be explained and compared in design effort.

This chapter will 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 – Ocap, and AIRT Designer. Each design is optimized to reduce cycle count and power consumption. All five designs will be compared based on energy, area, clock frequency/MIPS and design time. In the first section, a brief introduction to the LPC speech coder algorithm will be given. The main computational bottlenecks are identified. In the second section, the details of the design methodology are given. In the third section, the different signal processing platforms are introduced. Finally, in Section 15.5, the results are presented and compared. The last section gives the conclusions.

15.2 LPC Speech Codec Algorithm

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 [7]. 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. This excitation source then passes through gain a stage and a time varying IIR filter that uses the α values as coefficients.

The main building blocks of the LPC encoder are summarized in Figure 15.1.

Figure 15.1 LPC speech encoder example

The algorithm consists of the following main steps:

15.2.1 Segmentation

The voice samples are segmented in frames of 240 samples with an overlap of 80 samples between frames. The samples pass through a Hamming window (a raised cosine transfer function).

15.2.2 Silence Detection

Silence is detected by calculating the 0th lag autocorrelation and comparing that value to a threshold value. If silence is detected a bit is set in the parameter list, the gain G is set to 0 and the LPC and pitch detection phases are skipped.

15.2.3 Pitch Detection Algorithm

In order to classify each frame as voiced/unvoiced the autocorrelation function is examined. Indeed, if the frame is voiced, it must be periodic, thus forcing its autocorrelation to be periodic with an identical period. The algorithm used is due to Sondhi [8] and is described below:

1. The frame is low-pass filtered at 1 kHz.
2. A Clipping Level (CL) is set to 30% of the maximum value in the frame.
3. The frame, $x(n)$, is then clipped according to the following equation

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

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

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

where N is the length of the frame. Since minimum and maximum pitch frequencies for men and women are 80 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

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

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

15.2.4 LPC Analysis – Vocal Tract Modeling

To model the vocal tract, an all-pole function $H(z)$ is assumed

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

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 the 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 [7]. So the main calculations in this step are the computations of the autocorrelation values and the solutions of the Toeplitz system. The set of ten Linear Prediction Coefficients (LPC), i.e. the α_i values, as well as the prediction error $E^{(p)}$, which corresponds to the square of the gain G^2 , are computed and transmitted for each frame.

15.2.5 Bookkeeping

The last step consists of combining all the calculated values in a vector to be sent over the channel: an indication of silence/no silence, a vector with the α_i values, the gain, an indication of voice/unvoiced frame and the pitch.

15.2.5.1 Compute Intensive Functions

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

1. Pitch detection. A total of 78 correlations have to be computed – $R(22)$ through $R(100)$. The computations are simplified using the “clipped” coefficients, as explained in step 3 of the algorithm above, to reduce the computational complexity of this module.
2. Levinson–Durbin algorithm. This algorithm requires a lot of computations because it involves 11 correlations as well as an iterative 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.

15.3 Design Methodology

This section describes the design methodology for implementing the LPC speech coder on the various platforms. The different steps are illustrated in Figure 15.2. The codec is first designed in floating-point format in MATLAB. The challenge is to efficiently map the software algorithm onto the fixed-point hardware. This involves the conversion of floating-point computations into fixed-point word lengths (along with the ensuing design decisions) as well as the allocation and software mapping/scheduling on the available hardware.

15.3.1 Floating-Point to Fixed-Point Conversion

Since the algorithm executes in real-time on fixed-point hardware, one has to make decisions concerning the internal word lengths of each of the system hardware modules. An inadequate word length can lead to reduced Signal-to-Noise Ratio (SNR), deterioration of sound quality, and clipping. However, a surfeit of word length can create extraneous hardware, leading to wasted area and power.

For some of the platforms (i.e. the TI DSPs), the internal word lengths are fixed to a particular number (i.e. 16 bits). However, on the other platforms, the word lengths can be decided by the designer. There are several criteria which affect the fixed-point word length 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.

Figure 15.2 Design methodology to map an LPC speech coder on signal processing specific platforms

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 word length parameters and checking output saturation, the minimum bit requirements for each module can be found. This iterative refinement was done on the Ocapi and AIRT Designer platforms with the built-in fixed-point C++ libraries. This resulted in varying word lengths 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. As an example, the C++ code for the low pass FIR filter at the input of the pitch detection module is given below. Note the fixed-point annotations, `Fix(x,y)` means the data has a length of x bits, with y bits behind the decimal point. This annotated C++ code can be simulated with the fixed-point class libraries and is input into the behavioral level synthesis part of the AIRT design environment.

```

void fir (Fix(8,7) b[24], Int(9) seg[240], Fix(8,3) out[240])
{
    int i, j;
    Int(9) state[23];
    for (i = 0; i < 23; i++) state[i] = 0;
    for (i = 0; i < 240; i++)
    {

```

```

    out[i] = Fix(8,3) (b[0] * seg[i]);
    for (j = 1; j < 24; j + + )
        out[i] = out[i] + Fix(8,3) (b[j]*state[j-1]);
    }
    for (j = 23; j > 0; j - -)
        state[j] = state[j-1];
    state[0] = seg[i];
    return;
}

```

The fixed-point processors (TI 54x, TI 55x, TI 6x) have internal word lengths set to 16 bits for most arithmetic operations. To obtain a fixed-point C++ code suitable for such a processor, it is necessary to rewrite the entire algorithm using 16-bit C arithmetic (i.e. using ANSI C short format). In addition, the code must be heavily modified 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 generating suitable fixed-point code using Q15 functions, data scaling needs to be performed to prevent saturation. For example, the autocorrelation function has its maximum value at $R(0)$, which itself has an absolute 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 ensure 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. A more optimistic approach for scaling makes sense. If the speech is unvoiced the input will resemble noise and thus the autocorrelation will obtain a maximum value equal to the variance of the $C[x(n)]$ sequence. Therefore, each $C[x(n)]$ will be scaled 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.

15.3.2 Division Algorithm

Division is necessary as part of the Levinson–Durbin algorithm. To illustrate the conversion from floating point to fixed-point and to illustrate the care a designer has to take when making this translation, a specific division algorithm was chosen. Instead of using the more familiar restoring, non-restoring, or SRT division algorithms, a fixed-point Newton–Raphson reciprocation algorithm was chosen [3], because it matches well with a 16-bit fixed-point processor. The goal is to calculate $A = x/y$, in this case x is a 32-bit number and y is a 16-bit number. Using the Newton–Raphson technique, first $1/y$ is calculated and then multiplied by x to produce the final answer. The algorithm works as follows:

1. First, using the MSBs of y , linear interpolation is used to produce a rough estimate of $1/y$, called $z1$.
2. This rough estimate of $1/y$ is used to calculate a new estimate, $z2$ using the following equation: $z2 = z1(2 - y \times z1)$ where y is the 16-bit value of y .
3. Step 2 is repeated until the solution converges to a value z that is accurate in 16-bit resolution. This normally requires three iterations [10].
4. The final answer is calculated using: $A = x \times z$

Using this technique, the cycle count is significantly reduced. A full division is computed within 18 cycle counts.

15.3.3 Hardware Allocation

AlRT Designer [18] assumes a VLIW architecture, where the user is free to choose the data path modules in the architecture. Ocapi [17] 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, Address Control Units, 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 AlRT Designer. The initial design uses the AlRT Designer default minimum hardware allocation. This means that all operations have only one choice of hardware module. Operations on different execution units can still occur in parallel since AlRT uses a VLIW architecture. This implementation requires 8000 cycles to complete one frame. By examining the scheduling load graph, it is found that the autocorrelation function occupies 80% of the processing time and that one autocorrelation iteration takes three cycles where the Address Calculation Unit (ACU) is used every cycle.

Thus, the cycle count can be reduced by 4000 cycles by inserting an additional ACU and a second MAC combined with loop pipelining. By further investigation of the code, one finds that the coefficients for the windowing filter can be reallocated onto a ROM instead of being soft coded and calculated for every frame. 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.

15.4 Platforms

As mentioned previously, five implementation platforms are under examination, which are described briefly below.

15.4.1 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 [4]. 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 17×17 -bit MAC unit and an addressing unit. The program fetch is 16 bits and the instruction length is also 16 bits.

15.4.2 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 [15]. The C55x has additional hardware, including a 17×17 -bit MAC, a 16-bit ALU and a total of four 40-bit accumulators. The instruction length is variable between 8 and 48 bits, the program fetch is 32 bits.

15.4.3 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 are crucial. In our experiment, the C62x processor was chosen, a fixed-point DSP used for multi-channel broadband communications. The core implements a VLIW architecture with eight functional modules [12]. 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 [16].

15.4.4 Ocapi

Ocapi is a C++ based design environment developed by IMEC [14,17]. 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 next can be synthesized.

15.4.5 AIRT Designer

AIRT Designer is a software environment designed by Frontier Design [5,18]. As with Ocapi, AIRT 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. Simulations with the fixed-point libraries are performed. Upon completion of the fixed-point code, the user directs the software tools to perform resource allocation, resource assignment, and operation scheduling (based upon data interdependencies). AIRT generates synthesizable RTL level code which describes the entire VLIW machine.

15.5 Final Results

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 15.1, the overall area/memory and cycle counts for each platform are summarized.

Table 15.1 Implementation results

	Area– memory	Cycles/ frame (K)	Power at min clock ^a	Energy/ frame (μJ)	Technology (μm)	Power supply (V)
TMS320C6201 core [12]	16 kB ^b	30	3.3 mW	66	0.15	1.5 core
TMS320VC5410A core [9]	8.7 kB	240	7.2 mW	144	0.15	1.6 core
TI C5510 core [10]	10.2 kB	120	2.64 mW	53	0.15	1.5 core
Ocapi	1.4 mm ²	11	107 μW	2.1	0.25	2.5
AIRT	3.2 mm ² 2.3 kB ROM 1 kB RAM	3	215 μW	4.3	0.35	3.3

^a Minimum clock means the minimum clock frequency that needs to be applied to meet the real-time throughput requirement.
^b This includes only the program code.

15.5.1 Area Estimate

The Ocapi solution is slightly over half the size of the AIRT 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 for each circuit. For the Ocapi design, a 0.25-μm process was used while for AIRT Designer, a 0.35-μm process was used. Assuming perfect scalability, the AIRT Designer circuit would be only 1.63 mm². This is comparable to the Ocapi design area of 1.4 mm² as one would expect.

An ASIC design can afford to include the minimum amount of memory needed to implement the given applications. For instance for the AIRT design, one small data ROM is provided to store the coefficients of the Hamming window and the coefficients of FIR filters. The size is 264 bytes. The program memory is a ROM of size 2 kB. A data memory of about 1 kB RAM is also included.

For the DSP processor implementations, an estimate is made of what fraction of the available on-chip memory will be used by the program. For the C6x implementation, only the program size is included.

15.5.2 Power Estimate

Power figures for each design are given in Tables 15.1 and 15.2 in units of power at the minimum clock frequency and energy per frame. Power at the minimum clock frequency means that the processor is allowed to run at the lowest clock frequency that still guarantees to meet the real-time constraints. Speech is sampled at 8 kHz, one sample is 8 bit, resulting in 64 kbits/s. One frame is 240 samples but there is an overlap of 80 samples between frames. This results in a 50-Hz frame rate or 0.02 s/frame throughput requirement. The clock frequency of each of the processors is reduced to just meet this throughput requirement.

The power numbers for the ASIC platforms are estimated after synthesis with a standard

Table 15.2 Detailed power estimations

	W	Clock	Duty	mW/MHz	Cycles/ frame (K)	Mcycles/s	mW	Energy/ frame (μJ)	Technology (μm)	Voltage (V)
TMS320C6204 core [12]	0.44 W	200 MHz	50% high; 50% low	2.175	30	1.5	3.3	66	0.15	1.5 core
TMS320VC5410A core [9]	96 mW	160 MHz	50% MAC; 50% NOP	0.6	240	12	7.2	144	0.15	1.6 core
TI C5510 core [10]				0.44	120K	6	2.64	53	0.15	1.5 core
AIRT [5] – reference design	660 μW	2.5 MHz		0.264	3	0.15			0.25	1.05 I/O + core
AIRT [5] – voltage and tech scaling				3.65	3	0.15	0.55	11	0.35	3.3

cell library but before placement and routing. The power numbers for the DSP processors are estimated by using the power numbers published on the TI application reports and the TI webpage and multiplying these by actual cycle count and correcting them according to the type of instruction [1,9–12].

The ASIC circuits designed with Ocapi and AIRT Designer resulted in the lowest energy per frame with 2.1 and 4.3 μJ , respectively. One should note that the energy for each design refers to the energy of the core and memory and not for the I/O. One should also note that if only a small fraction of the available memory is used, a larger amount of energy will be wasted by accessing memories which are too large for the application. On the other hand, the DSP processors are built in a more advanced technology and can rely on optimized full custom layout compared to synthesized layouts for the ASIC implementations.

The assembly code for the C6x code was generated by C compiler of the TI Code Composer Studio software environment. The original C code with no optimization requires about 80,000 cycles to process a single frame. Successive optimizations of the C code and the use of the *fir_r8* subroutine (assembly optimized FIR code with the loop unrolled eight times) from TI's DSP library reduced the cycle count to about 30,000 cycles per frame. The newest member in the fixed-point C62x series is the C6204. At 200 MHz and 1.5 V, it has a power consumption of 290 mA for the core, at a 50% high, 50% low activity [12]. Thus, if only 30 K cycles are needed to perform one frame, the C6204 processor can either handle 86 channels running in parallel or reduce its clock frequency by this number, resulting in 65 $\mu\text{J}/\text{frame}$.

The C55x processor has a much lower mW/MIPS or mW/MHz number but does not have the same amount of parallelism as the C62x VLIW architecture. Therefore, it is interesting to note that the final number of 53 $\mu\text{J}/\text{frame}$ is about the same as the C62x. The C54x implementation is almost three times higher. One reason is that more effort has been spent to optimize the code for the C55x. Another reason is that the average instruction on the C55x will execute more operations than the average instruction on the C54x. An example of this is the usage of *fir2* routine in the DSP library which uses the dual MAC instead of a single MAC.

One can argue that making power estimations for ASIC circuits is tricky and error prone before actual processing. Therefore, we have compared our power estimation results with a processed and published processor which is built with the same AIRT environment [5]. This processor consumes 660 μW at 1.05 V running at 2.5 MHz in a 0.25- μm technology. While the application is different, the general architecture is still a VLIW architecture optimized towards the application and also in this case the minimum required clock frequency is very low, which is very beneficial to reduce the power consumption.

This processor has a power consumption of 0.264 mW/MHz. When this number is scaled up according to the scaling rules for short-channel devices [6], a value of 3.65 mW/MHz is obtained. This results in an estimate of 11 $\mu\text{J}/\text{frame}$ instead of the 4.3 μJ estimated from the synthesized results. This is still a factor 5 better than the lowest programmable DSP implementation. A similar comparison cannot be made for the Ocapi environment. The reason is that Ocapi does not have a “built-in” assumption of a basic generic architecture, which makes it hard to compare two designs built with the same environment.

15.6 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 5 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, Ocap and AIRT Designer. The availability of design environments that raise the level of abstraction from Verilog to C++ or from assembly to C, increases the productivity and allows the exploration of design alternatives. It also shortens the design time.

Acknowledgements

The authors would like to thank Alan Gatherer for the careful review of the TI processor power numbers and the EE213A class of Spring 2000. DH acknowledges the support from the Fannie and John Hertz Foundation. This work was partially sponsored by UC-MICRO#00-097.

References

- [1] Castille, K., TMS320C6000 Power Consumption Summary, Application Report SPRA486B, November 1999, available from www.ti.com.
- [2] Hwang, D., Mittelstadt, C., Verbrauwhe, I., ‘Low Power Showdown: Comparison of Five DSP Platforms Implementing an LPC Speech Codec’, *Proceedings ICASSP 2001*, Salt Lake City, May 2001.
- [3] Koren, I., *Computer Arithmetic Algorithms*, Prentice Hall, Englewood Cliffs, NJ, 1993, pp. 158–160.
- [4] Lee, W., Landman, P., Barton, B., Abiko, S., Takahashi, H., Mizuno, H., Muramatsu, S., Tashiro, K., Fusumada, M., Pham, L., Boutaud, F., Ego, E., Gallo, G., Tran, H., Lemonds, C., Shih, A., Nandakumar, M., Eklund, R. and Chen I., ‘A 1-V Programmable DSP for Wireless Communications’, *IEEE Journal of Solid-State Circuits*, Vol. 32, No. 11, November 1997, pp. 1766–1776.
- [5] Mosch, P., van Oerle, G., Menzl, S., Rougnon-Glasson, N., Van Nieuwenhove, K., Wezelenburg, M., ‘A 660-mW 50-MOPS 1-V DSP for a Hearing Aid Chip Set’, *IEEE Journal of Solid-State Circuits*, Vol. 35, No. 11, November 2000, pp. 1705–1712.
- [6] Rabaey, J., *Digital Integrated Circuits: A Design Perspective*, Prentice-Hall, Englewood Cliffs, NJ, 1996.
- [7] Rabiner, L., Schafer, R., *Digital Processing of Speech Signals*, Prentice-Hall, Englewood Cliffs, NJ, 1978.
- [8] Sondhi, M.M., *New Methods of Pitch Extraction*, *IEEE Transactions in Audio and Electroacoustics*, Vol. AU-16, No. 2, June 1968, pp. 262–266.
- [9] Texas Instruments, TMS320VC5410A, Fixed Point Digital Signal Processor, Document SPRS139A, November 2000, Revised February 2001.
- [10] Texas Instruments, TMS320C55x DSP Function Library (DSPLIB), February 2000, pp. 56–57.
- [11] Texas Instruments, TMS320C6211 Cache Analysis Application Report SPRA472, September 1998.
- [12] Texas Instruments, TMS320C6204, Fixed Point Digital Signal Processor, Document SPRS152A, October 2000, Revised June 2001.
- [13] Verbrauwhe, I., Nicol, C., ‘Low Power DSP’s for Wireless Communications,’ *Proceedings of the 2000 International Symposium on Low Power Electronics and Design*, pp. 303–310.
- [14] Vernalde, S., Schaumont, P., Bolsens, I., ‘An Object Oriented Programming Approach for Hardware Design’, *IEEE Computer Society Workshop on VLSI*, 1999, Orlando, FL, April 1999.
- [15] www.ti.com/sc/docs/products/dsp/c5000/index.htm.
- [16] www.ti.com/sc/docs/products/dsp/c6000/index.htm.
- [17] www.imec.be/Ocapi/.
- [18] www.frontierd.com.