Algorithm and Architecture of a 1V Low Power Hearing Instrument DSP

Finn Møller
GN Danavox
Maarkærvej 2A
DK-2630 Taastrup
+45 72111111
finn@danavox.dk

Nikolai Bisgaard
GN Danavox
Maarkærvej 2A
DK-2630 Taastrup
+45 72111111
nb@danavox.dk

John Melanson
AudioLogic
4870 Sterling Drive
Boulder, CO 80301
+1 303 444 8445
john@audiologic.com

ABSTRACT

This paper presents a 1 V digital signal processor used in the Danalogic hearing aid manufactured by GN Danavox. The processor is the first general purpose programmable used in behind-the-ear and in-the-ear hearing aid applications. It is integrated with memories, in a 0.5µ CMOS process with standard thresholds. At 2 MHz processing speed, the processor consumes 800µA from a single cell battery. Using a dual multiply-accumulate architecture, the processor executes a 256 point block floating-point FFT in just 2900 instruction cycles.

1. INTRODUCTION

Recent years of CMOS technology advances have increased the focus on digital signal processing for hearing aid applications. Even though several hearing-instrument manufactures offer digital hearing aids, only few digital hearing instrument IC's have been reported in literature [2]. Some have digitally programmable analog signal processing [3], while our previous work combine hardwired digital signal processing with analog signal processing [4]. One other reported hearing aid IC [1] offers the possibility to construct different filter algorithms by allowing a 41 words "program" to control the order of executing wave digital filters and automatic gain control functions in an architecture highly optimized for these two applications. However the focus on these two functions prevents [1] from being able to perform standard DSP algorithms like FFT's and adaptive filters

One step is still missing in the development of digital hearing instruments, namely to convert hearing aids from hardware design into constructing DSP software on a general purpose Digital Signal Processor. This paper presents this last step in the digital hearing instrument evolution, thus offering far more flexibility in tailoring the function of the hearing aid to the needs of the individual hearing impaired user. The processor has been constructed as a joint development project between ReSound, AudioLogic and Danavox and is implemented in a standard 0.5µ CMOS process. Using a performance metric of milliwats pr. mega multiply-accumulates, the processor achieves 0.2 mW/Mmacs. Recent publications on digital signal processors, operating at 1V supply [5] [6], employs multi threshold voltage 0.25µ CMOS processes in order to achieve similar performance.

The design of this DSP is described in the following sections. First section II describes the top-level architecture with focus on algorithm vs. architecture. Next section III describes the algorithm/implementation relation of some of the key DSP elements in order to achieve a low power performance metric. Section IV outlines the measured results/performances and compares this to existing commercial and published designs. Finally section V concludes upon the architecture, and the design strategies employed

2. TOP LEVEL ARCHITECTURE

The Top level architecture of the DSP is shown in Figure 1. The processor is, based on a modified Harvard architecture with two separate 16-bit data buses and one 32-bit program bus. The program bus width also defines the width of the instruction words, which can be fetched and executed from either program ROM or RAM.

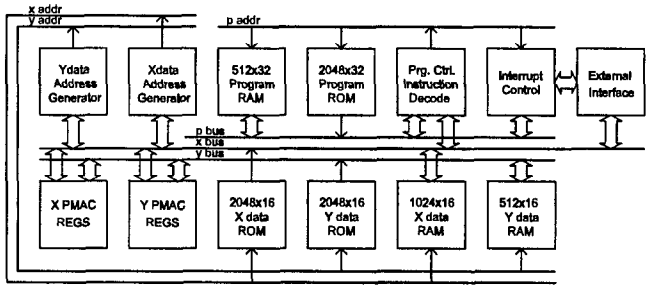


Figure 1 High level block diagram of the DSP architecture. (control signals not shown)

The DSP is furthermore equipped with two data ROM and two data RAM blocks, one for each data bus address space. Each data address space has its own data address generator capable of performing address calculations for data read/write in the ROM or RAM blocks. The x and y data busses both interfaces to two identical data path units, each containing a Pre-add Multiply-Accumulate (PMAC) unit and data registers for temporary storage of data and coefficient values. The PMAC unit is closer described in the following section. The instruction decoder is driven by the interrupt control unit which generates interrupts for the execution control whenever audio samples are ready/required at 16 kHz

ISLPED99, San Diego, CA, USA

from the external interface. Interfacing to both the program bus and the x data bus, the instruction decoder facilitates both direct (through control signals to the address generators), indirect (via the address generators) and immediate (via the x bus) addressing, thus allowing all the common DSP addressing modes (inclusive bit-reverse addressing for FFT's)

required control signals (leading to smallest possible instruction word width combined with ease of instruction decoding). For complex number calculation and FFT the double multiply-accumulate is a cost-effective compromise between consumed silicon area and performance, i.e. two instruction cycles performs a complex multiplication. The core radix-4 FFT butterfly can thus

```
 do 10, loopend
     a1=a0+x1*x0; b1=b0+y1*x0; x0=xmem[i0]; i0+=1; y0=ymem[i4]; i4+=1
loopend: b1+=x0*(x1+y1);  x0,y0=xymem[i1]; i1-=n
```

Figure 2: Instruction set example (not meaningful) displaying the instruction parallelism.

Figure 2 displays three sample instructions from the processor instruction set. The example illustrates the parallelism available in instruction execution. E.g. in a loop (instruction 1) two multiply-accumulate operations can be performed, concurrent with two data memory reads, and two indirect address pointer updates. The third sample instruction shows the PMAC operation in which the content of two registers (x1 and y1) are added before multiplied by x0 and summed and stored with the previous value of the b1 register.

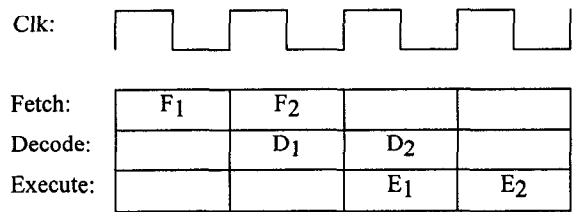


Figure 3: three stage instruction pipeline

The processor operates in a pipelined manner as shown in figure 3. Each instruction passes through three pipeline stages, fetch decode and execute. This allows for simple clocking of registers and pipe registers, but also for simple instruction control flow, as the execute-phase contains all computation and data move operations. This scheme also allows for zero overhead looping and single cycle interrupts, by letting the interrupt control unit slide in single instructions, e.g. fetching audio samples.

3. ALGORITHMS/ARCHITECTURE

In order to construct a DSP architecture suited for hearing aid applications, with the flexibility of a general purpose DSP, we first looked at different core algorithms which can be shown to be cost effective for implementing different filtering and processing functions. The selected suite of algorithms spanned FFTs, N-LMS adaptive filters, FIR and IIR filtering, complex number filters and multirate filterbanks based on parallel all-pass filters [7]. Also Log-domain processing and trigonometric operations was used in the algorithm suite for optimizing the architecture.

An intensive study was performed mapping different FFT butterfly's and filter structures onto as simple and consistent a data-path as possible. The degrees of freedom co-optimized were number of registers, arithmetic architecture, memory accesses and be executed in just 12 instruction cycles, leading to a 2900 cycle 256-point block floating point FFT.

The linear phase FIR filters and all-pass filters, benefits from the ability to add two numbers before multiplying by a third. Parallel all-pass sections are effective for performing band-split filtering [7], which is a commonly used function especially in hearing aid applications. The second order all pass section is given by:

$$y_n = a0 * (x_n - y_{n-2}) + a1 * (x_{n-1} - y_{n-1}) + x_{n-2}$$

This evaluation requires 4 MAC operations, but only 2 PMAC operations. An 8th order linear phase FIR filter is given by:

$$y_n = a0*(x_n+x_{n-7}) + a1*(x_{n-1}+x_{n-6}) + a2*(x_{n-2}+x_{n-5}) + a3*(x_{n-3}+x_{n-4})$$

Which again can be performed in 8 MAC operations, but just 4 PMAC operations.

3.1 Pmac Architecture

The observed need for an adder function before the multiply-accumulate unit would under normal circumstances generate an extra delay in the execution phase, but as will be shown in the following this delay can be minimized.

Booth encoding, of one multiplier input, has been shown in literature to be an efficient way to reduce power consumption of hardware multipliers. By reducing the number of partial products to be added in the multiplier, the Booth encoder inherently reduces the number of spurious switching events in the following summation array/tree. In order to maintain a simple overall clocking strategy, the complete execution cycle is contained in just one clock cycle. I.e. we would like the add-multiply-accumulate operation to happen in one combinatorial circuit. Inserting an adder of for instance ripple-carry or carry look-ahead type on any of the inputs to a booth encoded multiplier increases the delay of the PMAC operation. This is because the multiplication cannot start before both multiplicand inputs are ready and stable, thus inserting the adder delay directly in the delay path. Also power consumption is hurt badly because carry rippling in the pre-addition generates lots of extra spurious switching events.

The circuit shown in figure 4 performs the addition of two inputs a and b as an integral part of the Booth encoding process. The function of the described element is as follows:

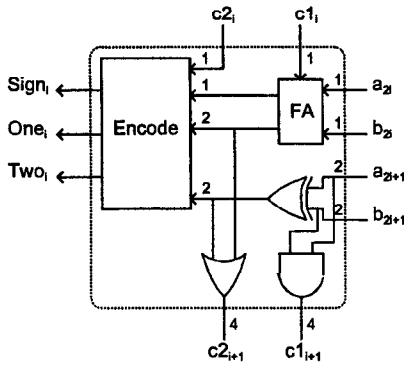


Figure 4 Booth adder/encoder element with only local carry and standard booth select output signals

	$c2_i = 0$			$c2_i = 1$	
Sum	$c2_{i+1}$	Enc.	Sum	$c2_{i+1}$	Enc.
0		0	1		1
1		1	2		2
2	4	-2	3	4	-1
3	4	-1	4	4	0
4	4	0	5	4	1
5	4	1	6	4	2

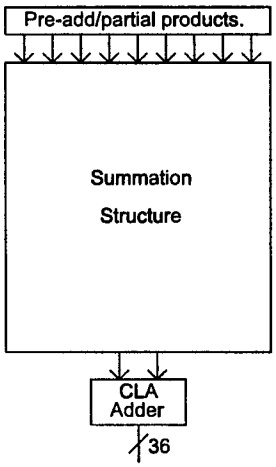
Figure 5 "truth-table" for the booth-encoder with integrated preadder.

- Four bits of the inputs $A = \sum_i^N a_i \cdot 2^i$, and $B = \sum_i^N b_i \cdot 2^i$, and a carry signal $\{a2_i, b2_i, a2_{i+1}, b2_{i+1}, c1_i\}$ with weight {1, 1, 2, 2, 1} are reduced by a full-adder and a half adder to four outputs having weight {1, 2, 2, 4}.
- The "four" is propagated on as a carry ($c1_{i+1}$) to the next pmac element.
- The wires having weight {1, 2, 2} are used as input to an encoder block with the "truth-table" shown in figure 4 (right).
- The encoder scheme is then as follows: When the sum of the inputs is less than two the encoder simply outputs the value (encoded as $Sign_i$, One_i and Two_i) possibly added by the "one" coming from $c2_i$. When the sum of the three inputs are greater than or equal to two, a value of "four" is propagated to the next stage (as $c2_{i+1}$). This is shown as an "or-gate" in figure 4.
- The correct numerical value is then generated on {$Sign_i$, One_i and Two_i} according to the table in figure 5. This correspond to the cases Sum$\in$ {2, 3, 4, 5} for $c2_i = 0$, and the cases Sum$\in$ {3, 4, 5, 6} for $c2_i = 1$

The Booth pre-adder is easily extended to accept 2's complement signed inputs $A = -a_N \cdot 2^N + \sum_i^{N-1} a_i \cdot 2^i$ and $B = -b_N \cdot 2^N + \sum_i^{N-1} b_i \cdot 2^i$, by replacing the FA in figure 4 with a "full-subtraction" for the MSB element adding the $-a_N$ and $-b_N$. Numerous architectures with more efficient gate-level implementations can be found for this element, but the general scope of this idea remains the same: To perform the addition of two numbers almost "for free", when doing booth encoding in a hardware multiplier

3.2 Multiplier summation tree

From a power saving point of view, another interesting part of the architecture is the part following the booth partial product generator. As the PMAC operation is executed non-pipelined, this unit needs to be carefully analyzed with respect to spurious transients in order to keep power consumption low. The design issues here are speed and power consumption vs. ease of implementation. Array summation structures are inherently well suited for custom layout, but their speed and power performance are poor compared to tree structures. Tree structures, however, are somewhat difficult to construct regular layout for. These are therefore often constructed using standard cell place and route tools, thus increasing the wiring capacitance over custom implementation.



Figur 6 Different architectures for the summation structure are examined

In order to analyze this, 5 different summation structures were studied, each capable of fitting into the "summation structure" box of figure 6. The 5 architectures were:

c_p: Array containing carry propagate (ripple-carry) adders

cs3: Array containing carry save Full-adders.

cs4: Array containing only carry save 4:2 counters

tr2: Binary tree consisting of carry propagate adders

tr4: Wallace tree containing Full-adders and 4:2 counters

C_p and tr2 consist of carry propagate adders, thus they have their own final ripple adders, which therefore substitutes the CLA adder in bottom of figure 6.

The following assumptions were made in the analysis:

1. Full adders, 4:2 counters and the radix-16 carry look-ahead adder are implemented using standard CMOS gates with up to 4 inputs.

2. Each CMOS gate has a generic unit delay.
3. The inputs (partial products) are assumed arriving at the input of the summation tree at the same time.

Assumption 1) and 3) are realistic assumptions for both custom and standard cell implementations, if e.g. the Booth-mux. partial product selectors contains a self latching function. Assumption 2 is a pretty rough approximation. However, as this experiment serves as the basis for selection of architecture and implementation style, assumption 2) is sufficiently precise.

	# of nodes in structure.	Gate delays	Average # of node transitions.
c_p	1264	82	7.42
cs3	1392	48	3.67
cs4	1251	38	2.26
tr2	1315	47	2.03
tr4	1279	30	1.56

Table 1 Gate delay and number of average node transitions pr. multiply cycle for each of the analyzed architectures

Table 1 displays the number of CMOS gate delays for each of the examined summation architectures. Also the average number of transitions pr. node are calculated for each structure. Calculating the power-delay product, the generic tr4 Wallace tree architecture outperforms the best array structure (cs4) by almost a factor of 2. This indicates that the fastest possible tree structure available is a cost efficient implementation even if implemented using automatic place and route tools.

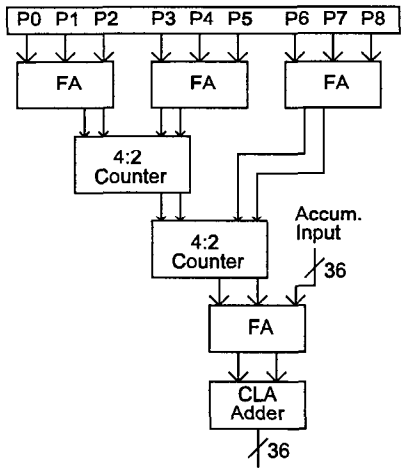
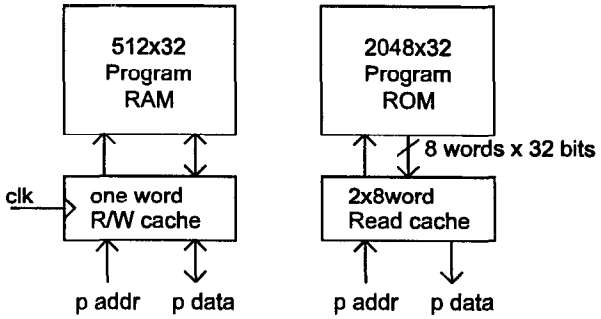


Figure 7 "Tr4" The selected summation tree structure feeding into the final Carry Look-ahead adder (CLA)

3.3 Memory Subsystem

The On-chip memory subsystem is divided into both program RAM and ROM, and data RAM and ROM divided into separate memory spaces. For a processor with a 32-bit instruction word, the fetching and decoding of instructions can be viewed as a necessary "overhead" on the computation load of the necessary signal processing algorithms. Much care has been put into the optimization of the memory configuration.



Figur 8 Architecture of the on-chip memory banks

Figure 8 shows the architecture of the on chip RAM and ROM blocks. In the RAM case a one-word cache register is inserted between the processor core and the memory core. This can be seen as a "delayed write register", thus for a 3 stage pipeline, a whole processor execution phase can be used to write an instruction in Program RAM.

For the design of the program ROM, the selected algorithm suite showed that supplying the ROM with two 8-word "pages" (an "even" and an "odd" page) of read cache improves power consumption. Optimizing the memory configuration against the "inner algorithm loops" during the instruction set design phase shows that all "inner loops" in the algorithm suite are shorter than 16 instruction words. This calls for two 8 word ROM caches, from which the core algorithm loops can be executed. The inner loops are thus placed as a library of callable "macros" in on chip ROM. Circular addressing in up to a 16 word loop enables the processor to execute out of ROM cache 90% of the time. This way the power hungry bit-lines and sense-amplifiers in the memories can be kept quiet most of the execution time.

Processor	Current	Volt.	Clock	mW/M macs
TMS320c62x	1.8A 258mA	2.5V 3.3V	200MHz	13.3 mW/M macs
ADSP2104	31 mA	3.3 V	20 MHz	5.1 mW/M macs
Mot. 56009	50 mA	3.3 V	40 MHz	4.1 mW/M macs
TMS320c5x	23 mA	3.0 V	40 MHz	1.9 mW/M macs
Mutoh [6]	29 mA	1.0 V	13 MHz	1.10 mW/M macs
Lee [5]	17 mA	1.0 V	64 MHz	0.27 mW/M macs
Danalogic	0.8 mA	1.0 V	2 MHz	*0.20 mW/M macs*

Table 2 Current consumption, Supply voltage, maximum clock frequency and miliWats / mega multiply-accumulate pr. second for four commercial low power dsp's, and two dsp's published literature

4. RESULTS

The DSP is integrated with an Audio input/output processor and external interface. It measures 28mm^2, and consumes 800µA

from a 1.0 Volt supply, when operated at 2.0 MHz. The performance metric mW/M macs for the processor is shown in table 2

Using its double MAC architecture and build in ROM twiddle factor tables, the processor executes a 256 point complex block-floating point FFT in just 2900 instruction cycles. For comparison the Motorola 56009 needs 8650 instruction cycles for this application (Table 3).

Processor	64 tab FIR filter	256 point BFP FFT
ADSP2104	64 cycles	7372 cycles
Motorola 56009	64 cycles	8650 cycles
TMS320c5x	64 cycles	8310 cycles
Danalogic DSP	*32 cycles*	*2900 cycles*

Table 3 Comparing the algorithm execution performance of three commercial available DSP's to the execution performance of the Danalogic DSP

This shows that the operation parallelism at 2.0MHz operation allows for fairly advanced algorithms to be executed. Thus, in the Danalogic hearing instrument the DSP executes several different algorithms simultaneously: FFT equalizing and multi band compression, noise reduction, adaptive feedback cancellation and a two-microphone zoom algorithm is processed concurrently.

Chip area:	28 mm^2
Current:	0.8 mA
Supply voltage:	1.0 V
Max speed:	2 MHz
Word width:	16 bit
Transistors:	544k
ROM bits:	131k
RAM bits:	49k

Figure 9: Main specifications of the Danalogic DSP.

5. CONCLUSION

This paper has described a programmable digital signal processor for hearing aid applications. This fills out the last gap in the digital hearing aid evolution history i.e. effectively moves hearing aid construction: from Hardware to DSP Software design.

The construction of a general purpose DSP under the very tight size and power constraints in a hearing aid, is only made possible by co-optimizing all levels of the system simultaneously. As demonstrated in chapter III, synergies need to be found from the application through algorithm, architecture, arithmetic and implementation levels. To reach the listed performance metrics, co-design exploiting just the limit between HW and SW is not enough. All levels from algorithm down to transistor level needs to be optimized, and their interactions needs to be exploited.

It is our belief that the transformation from hardware to software

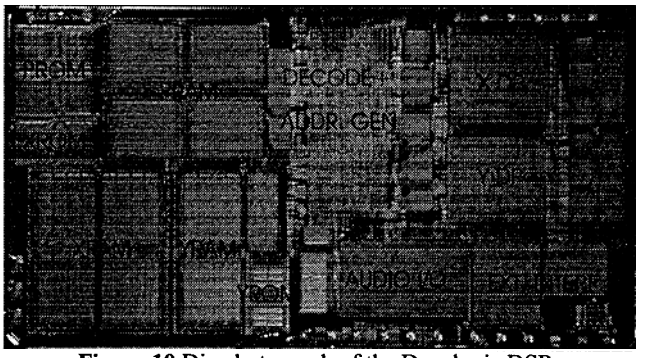


Figure 10 Die photograph of the Danalogic DSP

design enables far more advanced applications to be implemented for hearing aids in the future. Adaptive signal processing and "intelligent" algorithms are good examples of algorithms which could be implemented on a programmable platform, where their hardwired counterparts are almost impossible to implement without "turning" chips multiple times.

6. REFERENCES

[1] Harry NeuteBoom, Ben M. J. Kup and Mark Janssens: "A DSP-Based Hearing Instrument IC", in IEEE Journal of solid-state Circuits, vol.32 no. 11, November 1997.

[2] J. Krokstad et. Al: "An all digital concha hearing aid" in IEEE Workshop on Applications of Signal Processing. to Audio and Acoustics, Paper Summaries, pp. 85-88, 1993.

[3] F. Callias et al. "A set of four IC's in CMOS technology for a programmable Hearing aid", in IEEE Journal of Solid-state Circuits, vol. 24, pp 301-312, Apr. 1989

[4] N. Bisgaard and Ole Dyrlund: Acoustic feedback part 2: A digital system for suppression of feedback", Hearing Instruments, vol. 42, no. 10, 1991.

[5] Wai Lee et al. "A 1-V Programmable DSP for Wireless Communications", in IEEE Journal of Solid-state Circuits, vol. 32, no. 11, November 1997.

[6] Shin'ichiro Mutoh et al. "A 1-V Multithreshold-Voltage CMOS Digital Signal Processor for Mobile Phone Application", IEEE JSSC, vol. 31, no.11, November 1996.

[7] P Vaidyanathan, "Multirate Systems and Filter Banks" Englewood Cliffs, NJ: Prentice Hall, 1993. ISBN 0-13-605718-7