STATISTICAL ANALYSIS AS A QUANTITATIVE BASIS FOR DSP ARCHITECTURE DESIGN

Bat-Sheva Ovadia
DSP Semiconductors, Ltd
Givat Shmuel, Israel

Yair Be'ery
DSP Semiconductors, Ltd
Givat Shmuel, Israel
and
Dept. of Electrical Engineering
Tel Aviv University
Ramat-Aviv, Israel

Abstract - The increased demand for efficient application-specific DSP platforms encourages the utilization of extensive statistical analysis of the applications as a quantitative basis for the design of the architecture of these platforms. The paper presents the methods used for the definition of two generations of DSP cores based on both subjective and objective statistics. The subjective statistics consist of a detailed programmers' questionnaire, whereas the objective statistics are measurements of real-life software of specific applications. The analysis of these measurements led to the design of the first generation PINE DSP core [1], [2], [3]. Later, similar objective and subjective statistics were gathered from the actual implementation of a few applications on PINE. Investigation of the post-design measurements confirmed the main conclusions reached in the design phase of PINE. These measurements also served for designing the second generation, high-end, OAK DSP core [4] .

INTRODUCTION

Defining a new processor starts with the definition of the design goals. For example, PINE is a 16-bit fixed point DSP designed for the consumer and telecommunications markets, and the requirements for the PINE were driven out of this goal. Investigating these markets led us to the following key observations: The major trend in this market is portability, leading us to design PINE as a low-power DSP with a wide range of operating voltage, as low as 3v, and with power management control [1],[2],[3]. Minimization and cost effective solutions for these portable applications are also required. These requirements translate into a low-cost chip which means small silicon area. An additional requirement is high performance, from the perspective of instruction cycle time (25nsec) and from the perspective of architecture features, which will be discussed in details later. The constraints on the area and the requirement for quick time to market penetration lead to the design of a ***core*** with several levels of modularity in RAM, ROM, user-definable registers and I/O. The core definition concentrated on its main units: the computation unit, the data addressing unit and the program control unit, as can be seen in Fig. 1 and as described in [1],[2],[3]. All other peripheral blocks, which are application dependent, are part of the user-specific logic, implemented around the PINE core on the same silicon die, creating a so-called PASIC (PINE Application Specific IC).

When defining the architecture of a DSP for specific applications, rather than a general purpose DSP, one may have the opportunity to identify the relevant algorithms for these

0-7803-2123-5/94 $4.00 © 1994 IEEE

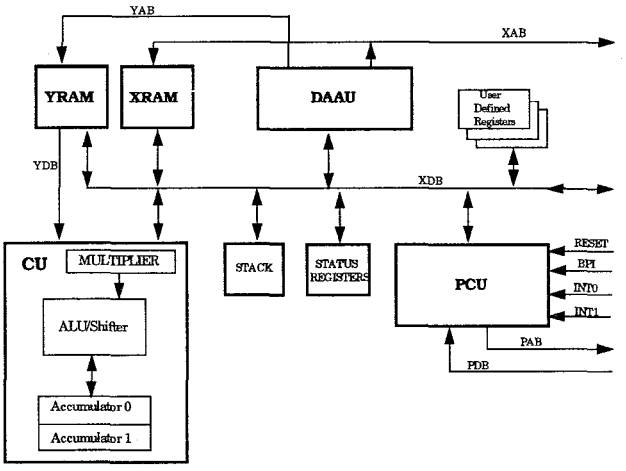

Figure 1. PINE Block Diagram

applications. As a result, the designer can figure out the architecture of the DSP which will optimize these algorithms, and execute them as fast as possible. Incorporating all the features into a new DSP may end up with a large chip which is not cost-effective. A quantitative design method is to statistically analyze the applications and carefully select those architecture features that best implement the algorithms, while tracking the silicon area cost, to achieve the best cost-performance ratio. The statistics include both subjective and objective statistics.

THE SUBJECTIVE METHOD

For the subjective statistics, 17 DSP programmers, who had an accumulated experience of writing about 200,000 lines of assembly code, were asked to fill a detailed questionnaire on the importance of DSP instructions. The ratings were given in two profiles, the *real time* profile, representing time critical DSP algorithms and the *control* profile, representing program flow and control functions. Rates were given with scales from 0 to 5 (5 representing the highest importance). Each questionnaire was given a weight reflecting the experience of the programmer.

In this questionnaire the instructions of two families of Texas Instruments DSP processors were examined: the first generation TMS320C1x (C1x) [5], and the derived second generation TMS320C2x (C2x) [6]. The instructions are divided into six categories: *accumulator memory reference*, *auxiliary registers and data page pointer*, *multiply*, *control*, *branch/call* and *I/O and data memory instructions* [5], [6]. Table 1 shows an example of the results for the first category, *accumulator memory reference instructions* (ALU instructions). Both *real-time* and *control* profiles in each category, included in the example, shown in Table 1, were independently sorted according to a weighted average of the instruction's rank, shown in the *AVG* column. Whenever there was a tie in the primary sorting, a second level of sorting was done according to the weighted standard deviation, shown in the *STD* column.

The main observations for all the categories are :

- The results are more consistent (small STD) in the *real-time* profile.
- Favorable (high ranked) instructions have, in general, smaller ambiguity (small STD).
- The sorted results give us a priority list of the C1x/C2x instructions for each profile.
- Specific instructions to C2x are marked with * or ** and in general have lower rank than the C1x instructions in all categories except for the *control instructions* category.
- In each category, two or three specific C2x instructions are clearly 'moving-up' in priority similar to the C1x instructions. These results are very important.

As an example of the last observation, note the *norm* and *lalk* instructions in the *real-time* profile of Table 1. The *norm* instruction is mainly used when evaluating the dynamic range of numbers and when converting fixed point numbers to floating point numbers. The *lalk* instruction is used to load the accumulator with long immediate numbers. Further evidence of the importance of these instructions was found from a third profile in which the programmers were asked to specify which essential instructions should be included in the new DSP. The *norm* instruction had a score of 92% and the *lalk* instruction had a score of 80%. Therefore these two instructions were candidates to be part of the PINE core.

REAL TIME			CONTROL		
	AVG	STD		AVG	STD
Accumulator Memory Reference Instructions					
SACH	4.97	0.17	LAC	4.97	0.18
LAC	4.97	0.17	LACK	4.90	0.30
LACK	4.87	0.34	AND	4.74	0.44
SACL	4.84	0.51	SACL	4.65	0.74
SUB+Shft	4.71	0.75	SUB+Shft	4.65	0.78
NORM •	4.70	0.81	OR	4.55	0.66
ADD+Shft	4.68	0.67	ADD+Shft	4.52	1.00
LALK •	4.55	0.81	LALK *	4.41	0.83
ADDH	4.46	0.94	XOR	4.26	1.08
ZALH	4.42	0.91	ZAC	4.23	1.13
SUBH	4.36	1.08	ADD	4.21	1.45
ZAC	4.35	1.06	SACH	4.19	1.20
ABS	4.35	1.03	XORK *	3.90	1.14
ADD	4.23	1.41	SUB	3.80	1.40
XOR	4.18	1.12	ADDH	3.62	1.42
ZALS	4.13	0.91	ZALS	3.58	1.39
SUBC	4.09	1.29	ANDK •	3.55	0.86
AND	4.06	1.26	ADDK **	3.50	1.16
OR	3.97	1.28	ORK *	3.45	1.07
SFR •	3.89	0.96	SFR •	3.30	1.42
ADDS	3.87	1.29	ZALH	3.18	1.63
NEG *	3.78	0.92	ADDS	3.16	1.48
SUBK **	3.75	0.97	SFL •	3.15	1.28
LACT •	3.71	0.61	ADLK *	3.10	0.94
SUB	3.68	1.45	SUBK **	3.10	1.37
ADLK •	3.67	1.05	ABS	2.82	1.20
SFL *	3.67	1.12	SUBS	2.79	1.58
SUBS	3.62	1.30	SUBH	2.76	1.75
ADDK **	3.52	1.23	SBLK *	2.75	0.89
ZALR **	3.38	1.07	CMPL •	2.65	1.01
XORK *	3.33	1.34	ROL **	2.55	1.20
ROR **	3.29	1.51	ROR **	2.55	1.20
SUBT *	3.24	0.97	NORM *	2.53	1.31
ROL **	3.17	1.65	LACT *	2.53	1.53
SBLK •	3.11	1.20	ADDT *	2.47	1.67
ADDT *	3.07	0.94	NEG *	2.35	1.31
CMPL •	3.00	1.55	SUBC	2.20	1.47
ORK •	2.93	1.05	SUBT *	2.11	1.29
ANDK *	2.85	1.15	ZALR **	1.74	0.91
SUBB **	2.48	1.10	SUBB **	1.70	1.00
ADDC **	2.44	1.13	ADDC **	1.67	0.84

• This instruction is specific to the TMS320C2x instruction set.
** This instruction is specific to the TMS320C25 instruction set.

Table 1. Subjective Statistics for TMS320C1x/TMS320C2x

Two other aspects must be reviewed before a final decision is taken: the objective statistics and the area cost. The objective statistics, discussed in the next section, were based on analyzing applications running on the C1x. The *lalk* and *norm* instructions are C2x instructions and therefore do not appear directly in the statistics. However the *lalk* instruction was implemented by the C1x programmers in two different sequences of *lt+mpyk+pac* or *mpyk+pac*. These sequences were detected during the statistical analysis phase and marked as the *lalk* instruction while removing them from the accumulated results. For the C2x *norm* instruction, different C1x programmers used different C1x instruction sequences which were very hard to track and therefore were not taken into account in the objective statistics. The results of the questionnaire, however, indicated it to be a very important feature of a future DSP. Therefore the area was carefully estimated for deciding whether or not to implement a normalization mechanism. Normalization can be performed in software by paying a lot of real-time or in hardware by using a large silicon area. The PINE normalization mechanism is a compromise between area and real-time: it includes an instruction for a normalization step and particular hardware to reduce the real-time.

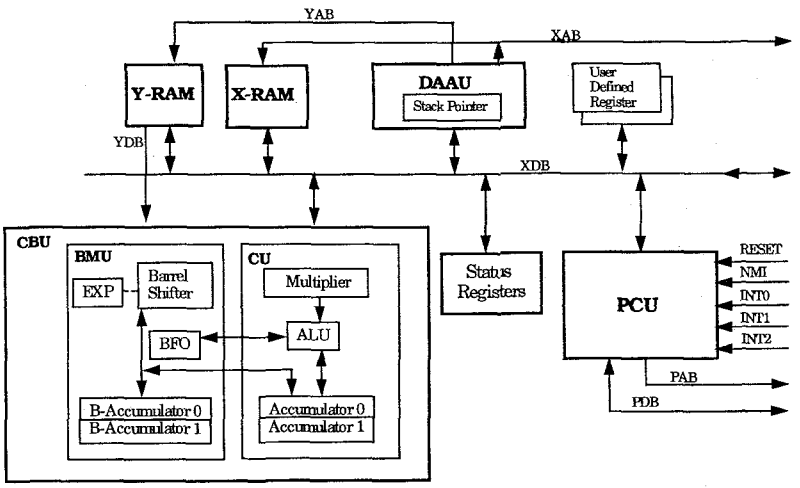

Figure 2. OAK Block Diagram

Normalization was reconsidered when defining the second generation, OAK. Similar objective and subjective statistics were made for designing the OAK, including a detailed questionnaire, exploring the importance of the PINE instructions. Once again the importance of normalization was observed. This fact was supported by the dynamic statistics on PINE-based applications. Since the definition of OAK as a high-end DSP, allowed for a larger core, the normalization hardware was extended. Full normalization is achieved in OAK in 2 cycles using an hardware unit for exponent evaluation with the aid of a 36 bit barrel shifter which was also included into the OAK to support bit field operations, as described in Fig. 2.

In addition to ranking the instruction-set, the questionnaire tempted to identify important features that would enhance the architecture, e.g. interrupt context switching and support

for a C-compiler. These were considered to be part of the first generation, but were found to be more appropriate to be included in the second generation. The most important feature to support the compiler was diagnosed as being a software stack, which was introduced in OAK, for the first time in a **fixed point** DSP.

THE OBJECTIVE METHOD

When defining a processor, designers search for the most important features that should be included. In PINE extra emphasis was on the die size, considering for each feature whether its performance justifies the required silicon area. The objective method serves to measure the instruction usage, which is an indication of various processor capabilities. The method is used to achieve a quantitative basis for performance evaluation. The question that was raised at this point, was on what to measure these statistics. One approach is to evaluate 'toy' benchmarks [7]. To compare between DSPs, the most common benchmarks are FIR, LMS, IIR, FFT, etc. These toy benchmarks however, are very small and part of them do not apply to PINE's target applications. Another approach is to write larger programs, by coding some of the target algorithms in the C or assembly language. However, we decided to draw a picture of the *real world* by measuring actual applications. The toy benchmark or kernel of some algorithms were only used for 'fine tuning' various instructions rather than evaluating the important features solely from them.

In the objective statistics, the actual applications were analyzed from two aspects: static code measurements and dynamic run-time measurements. As Hennessy and Patterson described in [7], for evaluating the performance we should be more interested in the dynamic measurements. In our case this meant counting the occurrence of instruction executions during real-time applications. The static measurements were made on the code, independent of execution [7], and are important for code compactness implications.

One of the problems when defining an architecture of a new processor, based on statistics made for another processor, is that it is sometimes hard or even impossible to measure the contribution of new ideas and features that are candidates to be incorporated into the new processor. Some of the problems can be alleviated by introducing an instruction sequencer that identifies sequences of instructions. One example was discussed in the previous paragraph when describing the C2x instruction - *lalk*. Another example is the obvious *mac* instruction which is not part of the C1x instruction-set, which was identified by the sequencer as *lta+mpy* or *ltd+mpy*. It is important to find these sequences and delete them from the total statistics, to have correct measurements. Another example, for which quantitative statistics are hard to obtain, is the software stack, implemented only in the OAK, as described in the previous section.

Static Code Measurements

The static code measurements were based on accumulating actual TMS320C1x code from different applications. Fig. 3 and Fig. 4 show some of the results of the objective static statistics. The left part of each figure displays the accumulation of three different all-digital Telephone Answering Device (TAD) applications. The right part shows the accumulation of two different Analog-Cellular-Phone applications. A total of about 40 Kwords of program code was analyzed. Histogram ***a*** in these figures represents the

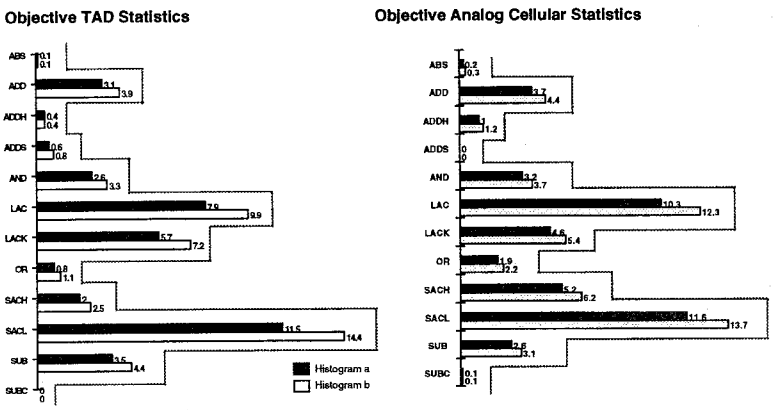

Figure 3. Static Statistics for TMS320C1x

percentage of the appearance of the **words of an instruction** relative to the total number of **words** in the code. Histogram ***b*** represents the percentage of the appearance of an **instruction** relative to the total number of **instructions** in the code. Since certain instructions are one-word instructions and others are two-word instructions, these two histograms are different. Histogram ***a*** reflects the actual number of words each instruction consumes in the measured code of the program memory. Histogram ***b*** enables to predict the relative appearances of each instruction in the designed DSP, where the number of words per instruction may be changed.

Fig. 3 displays the static statistics for part of the arithmetic and logical instructions. It can be observed that the same pattern of instruction usage appears in both types (classes) of applications. Even though the applications use different algorithms (the TAD includes a speech compression/decompression algorithm, DTMF tone generation and detection, and control functions, while the analog-cellular includes mainly various types of digital filters) the patterns look the same, suggesting a more general trend. This is further confirmed by the various applications in which PINE has been successfully used since its introduction to the market: digital cellular phone, fax modems, sound boards and disk servo controllers.

The static code measurements, based on the relative appearance of the instructions in several applications, can lead to conclusions regarding mechanisms that can contribute to the code compactness. Code compactness means reduction in the program memory size which in turn reduces the chip area of a PINE with on-chip ROM. An example is shown in Fig. 4 displaying the static measurements for the C1x branch instructions. These include one unconditional branch instruction (*b*) and eight conditional branch instructions (*bgez, bgz, bioz, blez, blz, bnz, bv* and *bz*). The overall branch instructions are about 19% of the total number of program words, in both classes of applications. These instructions are two-word instructions, where the first word is the opcode and the second word is the 16-bit address. A second level of analysis was performed on the second word of the opcode, and led us to the conclusion that most of the branch instructions are short branches, i.e. a branch to a location near the branch instruction, using about 6-7 bits of offset. Therefore

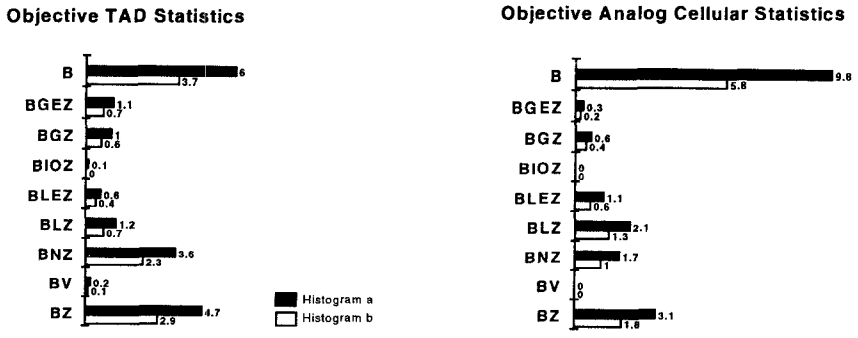

Figure 4. Branch Instructions Static Statistics for TMS320C1x

a new branch type, the relative branch, was introduced. The relative branch takes only one program word and includes the branch opcode, the condition and a 7 bit offset. Using this instruction (*brr*) a branch can be executed in the range of -63 to +64 from the location of the specific *brr* instruction. From chip size point of view, the cost of implementing the new relative branch mechanism in the core is smaller than the program memory area that is expected to be saved. In addition, when the relative branch mechanism was added to the core, it was also used for implementing a relative *call* instruction (*callr*), even though this type of instruction was expected to be used less frequently.

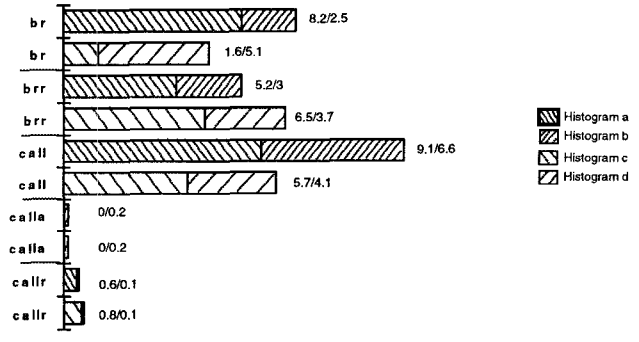

Figure 5. Branch and Call Instructions Static Statistics for PINE

Fig. 5 shows statistics gathered from PINE based applications. These measurements served to design the second generation, high-end, OAK DSP core, but also served to validate the PINE architecture. Histograms ***a*** and ***b*** represent the percentage of the appearance of the words of an instruction relative to the total number of **words** in the code, where histogram ***a*** describes the TAD application and histogram ***b*** the analog-cellular application. Histograms ***c*** and ***d*** represent the percentage of the appearance of an instruction relative to the total number of **instructions** in the code. Histogram ***c*** describes the TAD application and histogram ***d*** the analog-cellular application. Both applications use the relative branch option. As can be seen, there is a more extensive use of this option in the TAD application than in the cellular-phone application. Exploring the code written

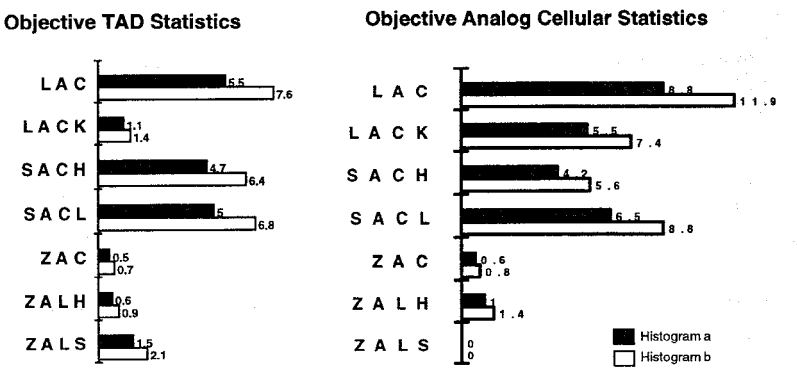

Figure 6. Accumulator Instructions Dynamic Statistics for TMS320C1x

for the cellular application, many program words could have been saved if the *brr* instruction had been used rather than the *br* instruction. This result was reported to the analog-cellular's programmers which changed their code and reduced the program size. The figures represent the code prior to the improvement. Analyzing the TAD results, 5.8% of the instructions-words (*brr* 5.2% + *callr* 0.6%) used the relative jump mechanism, e.g. assuming 10K words of program length, 580 words were saved using this feature of the PINE.

Dynamic measurements

The dynamic execution measurements consisted of sampling running code of various existing applications in different modes of operation. The idea was to locate real-time consuming instructions and look for ways to reduce the execution time for those cases.

Fig. 6 shows some of the dynamic statistical results for the TAD and analog-cellular based on C1x. In the TAD application, the statistics were based on approximately one million cycles of running code and in the analog-cellular application they were based on approximately 40,000 cycles. Each application class generated two histograms. In Fig. 6, histogram ***a*** represents the percentage of the appearance of the cycles of an instruction relative to the total number of **cycles** in the running code. Histogram ***b*** represents the percentage of the appearance of an instruction relative to the total number of **instructions** in the running code. As in the static measurements, histogram ***a*** reflects the actual number of cycles each instruction consumes out of the total number of cycles in the measured code. Histogram ***b*** enables to predict the relative appearances of each instruction in the designed DSP, where the number of **cycles** per instruction may be changed.

Fig. 6 shows that the load and store accumulator instructions (*lac, lack, sach, sacl, zac, zalh* and *zals*) play a major role in real-time consumption in both classes of applications. The question that we asked ourselves was "can something be done to reduce the execution time in these cases?". The applications ran on C1x which uses only one accumulator. Observing the code, we found that many computations were done as follows: first a 32 bit calculation is made followed by an accumulator storage, then a new 32 bit calculation

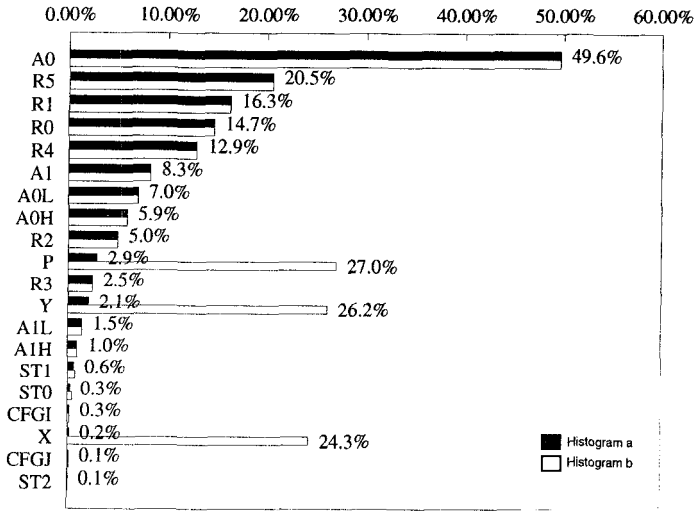

Figure 7. Dynamic Register Usage Statistics for PINE

is performed and a final addition of the first stored result is executed. Switching to an architecture which uses two accumulators, each of them connected directly and symmetrically to the ALU output and inputs, as in PINE, reduces the large overhead of such load/store accumulator instructions. The quantitative gain of this change was derived from Fig. 6 and compared to the relative cost of the additional silicon area required to implement the second accumulator.

For the definition of the second generation core, OAK, similar statistics were made, sampling PINE applications (TAD, analog cellular, speech recognition and ADPCM coder). Again, the analyzed statistics served for both post-design validation and as statistics for the definition of the next generation. Fig. 7 displays the dynamic usage of most of the PINE registers and confirms the need for two accumulators. Histogram ***a*** displays the register usage explicitly referenced in the instruction opcode, while histogram ***b*** also includes the implicit usage of other registers as a result of a specific operation. For example, in the multiply-accumulate instruction, the result can be saved in either A0 or A1 but implicitly the multiplier registers (X,Y,P) are also used. From exploring the applications and reviewing them with the programmers it became clear that when just one accumulator was necessary, accumulator A0 was used. When a second accumulator was needed in parallel to A0, accumulator A1 was used. In Fig. 7 we can see that accumulator A0 is used in almost every second instruction in comparison to the second accumulator (A1) which is used 8.3% of the time. If PINE contained only one accumulator, each time there was a need for another computation, A0 should have been stored using 1 or 2 extra cycles (depending on whether 16 or 32 bits are stored) and reloaded by either 1 or 2 additional cycles, meaning a total increase payment of 16.4 to 32.8% in real-time.

It should also be noted that 45% of the total C1x sampled instructions were move-type instructions, while in PINE-based applications only 35% were move-type instructions. This improvement of 10% in real-time was achieved by the use of two accumulators and the configuration of PINE's 31 registers as a global register set.

Examining the dynamic statistics we found a particular C1x instruction, *banz*, which was executed 10% of the real-time, a very large number for a single instruction. This instruction is used for loop control with an auxiliary register as a loop counter. This software mechanism can be exchanged by a hardware mechanism implementing a zero-overhead loop. Although the chip size for implementing this feature is relatively large, it dramatically reduces the real-time by 10%. It should be noted that this feature is not a new invention, but a quantitative conclusion that even a small size DSP should include this feature.

CONCLUSIONS

When planning a new low-cost DSP for a specific market, there are many features that may be incorporated or not. In the difficult tradeoffs (architecture performance versus die cost) considered when designing the DSP, statistical analysis plays a major role. The subjective statistics focus the design efforts on the most important instructions and mechanisms, while the objective statistics point to the time critical ones and ensure overall code compactness. Both types of statistics are required to keep costs down with minimal performance compromises. This statistical method has been used for defining the PINE DSP core and was reapplied on PINE applications in the form of post-design statistics to validate both the design and the design method. Based on this method and on the statistics of PINE applications, the second generation OAK DSP was designed.

ACKNOWLEDGEMENTS

The authors would like to thank Ronen Peretz and Amnon Rom for their contributions to the development of the software tools for the statistical analysis and for many helpful discussions and to Gil Bendelac for many comments that improved the presentation of this paper.

References

[1] S. Berger, Y. Be'ery, B. Ovadia, R. Peretz, G. Wertheizer and Y. Gross, "An Application Specific DSP for Speech Applications", *IEEE Trans. on Consumer Electronics*, Vol. 39, No. 4, pp. 733-739, November 1993.

[2] Y. Be'ery, S. Berger and B. Ovadia, "An Application-Specific DSP for Portable Applications", in *VLSI Signal Processing, VI*, IEEE Press, L.D.J. Eggermont, P. Dewilde, E. Deprettere and J. van Meerbergen Eds., 1993, pp. 48-56.

[3] DSP Semiconductors, "PINE Architecture Specifications", rev. 3.4, Sep.1993

[4] DSP Semiconductors, "OAK Architecture Specifications", rev. 2.0, May 1994

[5] Texas Instruments, TMS320C1x User's Guide, April 1990.

[6] Texas Instruments, TMS320C2x User's Guide, April 1990.

[7] J. Hennessy and D. Patterson, *Computer Architecture a Quantitative Approach*, Morgan Kaufman publishers Inc., 1990.