USING RISCs ON DIGITAL SIGNAL PROCESSING

Eduardo B. Wanderley Netto
eduardo@etfrn.br
ETFRN-CEFET/RN
Av. Sen. Salgado Filho, 1559
Natal / RN

Celso L. Mendes
celso@dpi.inpe.br
INPE
Caixa Postal 515
São José dos Campos / SP

Osamu Saotome
osamu@ita.cta.br
ITA
Pça Mal. Eduardo Gomes, 50
São José dos Campos / SP

Abstract

RISC processors have proved to be efficient for processing general purpose tasks. Recently, RISCs have been successfully used also on some Digital Signal Processing (DSP) applications. This work shows how a generic RISC processor (DLX) performs on DSP algorithms, in comparison to a conventional digital signal processor, the TMS-320C25. As a contribution of this work, we propose and evaluate a RISP (RISC dedicated for DSP) processor: *dlxdsp*. To measure performance, we use DSP kernels with simulators of the three processors. Our simulation results show that slight modifications on modern, general purpose RISCs can promote better performance for some kernels, even in comparison to a dedicated processor.

1. INTRODUCTION

Since the last decade RISC processors have been studied and present good performance characteristics in comparison to its counterpart family, the CISC processors (a good introduction to RISC processors can be found in [5]). The RISC philosophy was fastly spread among scientists and manufacturers. In 1992, a new approach to using RISC processors on Digital Signal Processing (DSP) has been taken. Some compelling advantages of modern RISC processors match DSP requirements, like fast instruction cycle, efficient use of pipeline, zero branching overhead, and floating point capabilities. These factors provided RISC processors with a good performance when dealing with DSP.

In this paper, we present some measurements on performance and instruction set usage of a typical digital signal processor (Texas Instruments' TMS-320C25) and of a generic RISC processor (DLX) proposed by Hennessy and Paterson [2]. Then we propose the architecture of a new RISC processor designed for DSP – the *dlxdsp*, including its performance characteristics in comparison to the other processors studied.

The rest of this paper is organized as follows. In Section 2 we present the programs and the metric used in our performance study, and in Section 3 the simulation infrastructure in which we conducted our experiments. Section 4 contrasts the behavior of a dedicated digital signal processor and of a general purpose RISC processor. In Section 5 we introduce an enhanced RISC processor, and show corresponding simulation results in Section 6. Finally, Section 7 contains our conclusions and future work.

2. BENCHMARKING AND METRICS

The benchmarking methodology and the metrics used determine the reliability of the results shown in this work. The following subsections detail the methods applied.

2.1. Benchmarks

The best benchmark is the application itself, but sometimes it is not possible (or desirable) to use it. In these cases the choice is to use an application similar to that will be really used. Another approach is to use fragments of code which are frequently used in the application area (called kernels). Some advantages of using this last method in DSP are [3]: relevance; easy of specification; easy of optimization; and easy of implementation. To compose the set of algorithms used in this work, the base was the DSPstone [7]. The considered fragments were:

- 67 taps, low pass, FIR Filter;
- 6^{th} order, low pass, IIR filter – canonic implementation;
- 8th order discrete cosine transform;
- 256 point complex DIT FFT.
- 6^{th} order, low pass, IIR filter – cascadable implementation;
- Adaptive FIR filter, order 67, with LMS adaptation;
- Two 8x8 matrix multiplication;

2.2. Metrics frame

The metrics used was composed by the following CPU time equation [2]:

$$Time_{CPU} = Intructions \times \frac{Cycles}{Instructions} \times \frac{Time}{Cycles} \qquad \text{Eq. 1}$$

The number of instructions used in an algorithm depends, among other factors, on the ability of the compiler to schedule the right instructions at the right time. So, the compiler technology strongly determines the code size. To minimize the compiler's influence on the architectural parameters measured, all the algorithms were hand-coded like in [3].

The second term on the right side of Eq. 1 is the CPI (average number of cycles per instruction). This is a dynamically measurable parameter, which takes into account factors like cache misses, pipeline stalls, interruptions, etc. The CPI has the most complex determinants: the instruction set architecture is one of the most important. In this work, the memory system was considered perfect. It means that cache miss problems were not investigated and all data needed was immediately available. I/O was not considered either. Thus, the CPI presented here is in fact an approximation.

Nevertheless, those parameters that were not taken into account are not expected to affect strongly the obtained results, because of the nature of the algorithms we are dealing with: in most of them the data set sizes should fit a reasonable cache (see details on [4]), and there is nearly no I/O involved after data are in memory.

3. SIMULATION ENVIRONMENT

The following subsections have brief descriptions of the studied processors.

3.1. Processors

The digital signal processor used was the Texas Instruments' TMS-320C25, in which we find multiply-and-accumulate instructions (MAC), three-stage pipeline, Harvard architecture, on-chip RAM and ROM, fixed-point ALU and an auxiliary ALU (ARAU) for pointer incrementation/decrementation, barrel shifters for adjusting results, etc. There are special operand registers to evaluate MAC operations, to point to memory pages, etc. This can be considered a limiting factor for temporary storage of values. A complete description of the TMS-320C25 can be found in [6].

The RISC model used was the DLX. This processor has thirty-two 32-bit general purpose registers and a set (of register) of equal size for floating point operations. There are four specialized arithmetic units (FP multiplication, FP add, FP divide and Integer operations), some control registers (PC = Program Counter, IAR = Interrupt Address Register,...) and delayed branch is implemented.

Figure 1 presents a Kiviat graph with some architectural parameters of the processors studied. The circle in this graph shows the usual parameters for RISC's [1].

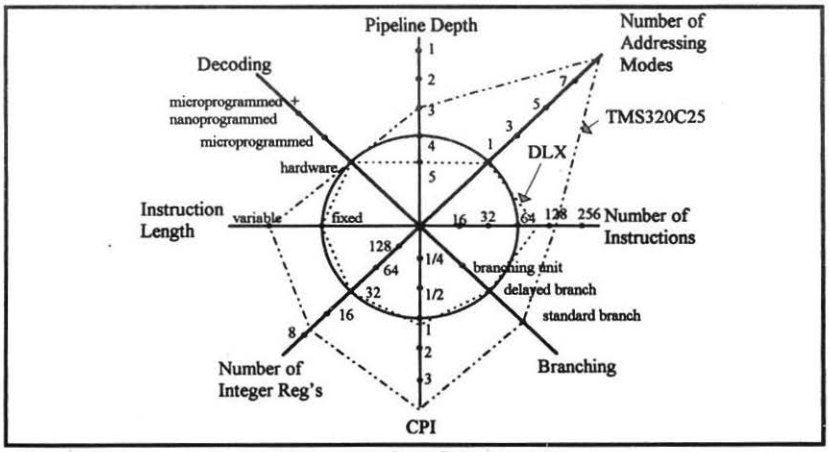

Figure 1: Kiviat graph showing DLX and TMS-320C25 parameters.

3.2. Simulators

To monitor the performance of these architectures, we used two simulators. The first one was the TMS-320 simulator, available from Texas Instruments; it provides detailed information about the execution of a given program, like number of executed instructions, execution traces, etc. The second simulator was a public domain tool called DLXsim, which provides complete information on the execution of a program by DLX. In this tool, the source code is also available, thus making easy the task of experimentation regarding modifications to the original DLX's architecture.

4. INSTRUCTION SET USAGE

The following subsections show instruction set usage by the TMS-320C25 and DLX processors on the selected kernels. This is quite important for a new instruction set design suitable for DSP.

4.1. TMS-320C25

The TMS-320C25 instruction set has 133 instructions and 9 addressing modes (considering 7 cases of indirect addressing). Figure 2(a) presents the most used instructions: `MACD` and `MAC` (multiply-and-accumulate), `SACH` (store accumulator high), `APAC` (add), and `MPY` (multiply). If the instructions are classified in Control, Arithmetic and Memory, the kind of load submitted to the processor is highlighted as in Figure 2(b). As we expect from a DSP load, the arithmetic instructions are predominant. So, Figure 2 also shows how Arithmetic instructions are used. It is easy to realize how important multiply-and-add instructions are.

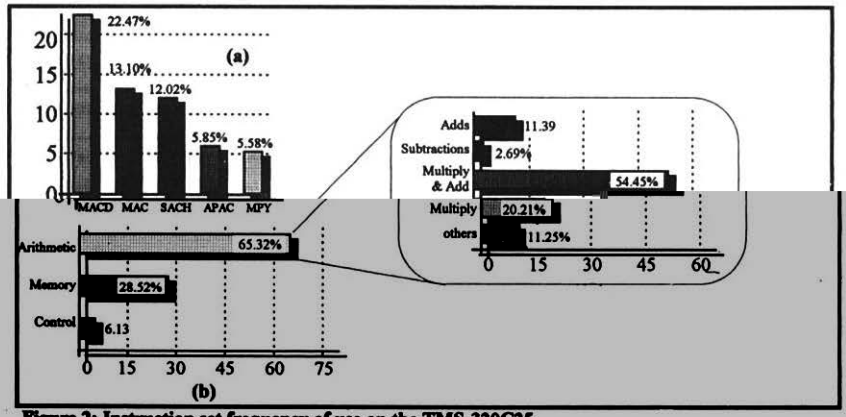

Figure 2: Instruction set frequency of use on the TMS-320C25.

Table 1: Addressing Mode usage on the TMS-320C25.

Addressing Mode	%	Addressing Mode	%
Post-decrement indirect	24,59	Direct	19,44
Post-increment indirect	24,66	Immediate	12,46

Table 1 refers to the addressing mode usage frequency. Post-increment and post-decrement indirect addressing modes have a compelling role on this distribution. It means that data items are regularly disposed in memory. Together, they represent almost 50% of use.

4.2. DLX

The DLX measurements on instruction set usage are presented in Figure 3. The most used instructions are: `LD` (load), `BNEZ` (branch), `SUBI` (subtract), `ADDD` (add), and `MULTD` (multiply). Memory access is now highlighted because DLX is a LOAD/STORE machine, so it can not execute arithmetic instructions with data in memory like the TMS-320C25 does. Other instructions that are frequently used are branches and subtractions. That is because DLX has only one addressing mode. Thus, to implement access to neighbor positions in memory (remember that DSP has a good uniformity on data access in memory), subtraction instructions are used to decrement pointers inside a loop. This corresponds to a post-decrement addressing mode.

An interesting result is that about 60% of instruction set usage is distributed only among five instructions. This is almost the same result for the TMS-320C25. As DLX is a RISC, this result might seem not to be realistic. Nevertheless, it happens because DSP problems are quite uniform [8].

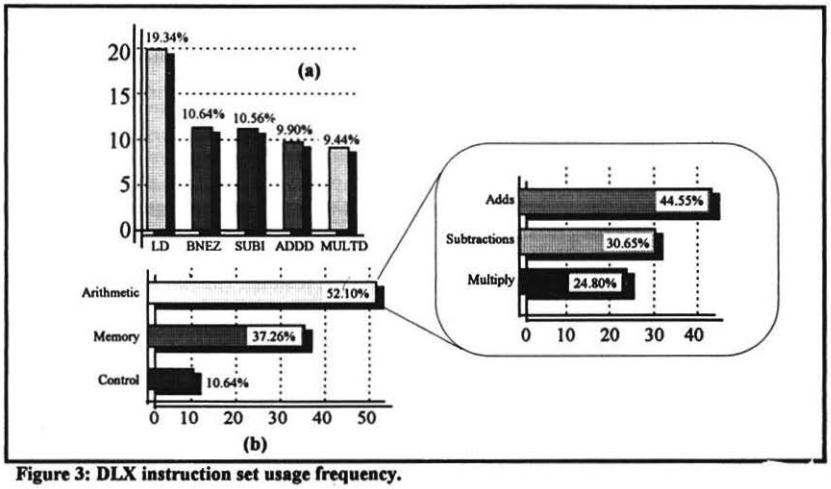

Figure 3: DLX instruction set usage frequency.

5. RISP PROPOSAL: *dlxdsp*

The motivation to look for a specialized architecture lies on the uniformity of the problem. DSP presents this characteristic very impressively. *dlxdsp* is a RISP (**RI**SC for **S**ignal **P**rocessing) which is designed based on results presented here. *dlxdsp* is an extension to DLX, and thus maintains many of the characteristics that are present on DLX (including its number of cycles per instructions). However, the architecture was slightly modified to support essential DSP features.

The main modifications to the DLX architecture were a new Floating-Point arithmetic unit, to provide multiply-and-add operation support, and a decrementer connected to integer registers, to provide automatic decrementation of pointers. By using adequate hardware support, such modifications would **not** cause any change in the clock cycle relatively to DLX (for details, see [8]). With those architectural modifications, five new instructions were added to the original instruction set of DLX: `DBNEZd` and `DBNEZf` (**D**ecrement (of 8 or 4 respectively)[1] and **B**ranch if **N**ot **E**qual to **Z**ero), `CMvlt` (**C**ompare and **M**odify if value is **l**ower **t**hen 0), `MACd` and `MACf` (**M**ultiply and **AC**cumulate for double and single precisions respectively).

The question is: 'would these few modifications make a difference?' To answer this question, we conducted extensive simulations. In the next section, we show the most important results.

6. PERFORMANCE EVALUATION

To have a basis for comparison between the architectures, we show in the following subsections the results for the processors on each analyzed item of performance. The assumed test conditions were: five clock cycles for *dlxdsp*'s MAC (multiply-and-accumulate) unit and 40MHz of clock frequency.

6.1. FIR Filter

Figure 4 shows the obtained results. TMS-320C25 presents great facilities to implement FIR filters. Basically, only two instructions are responsible for FIR processing. Thus, the number of executed instructions is much smaller on TMS-320C25 than on the other processors, although the number of clock cycles is almost the same for 'C25 and *dlxdsp*. This happens because each machine cycle on TMS-320C25 has four clock cycles, while on DLX and *dlxdsp* it has only one.

[1] Eight bytes is the space between two consecutive double precision floating-point numbers in memory. For single precision the spacing is four bytes.

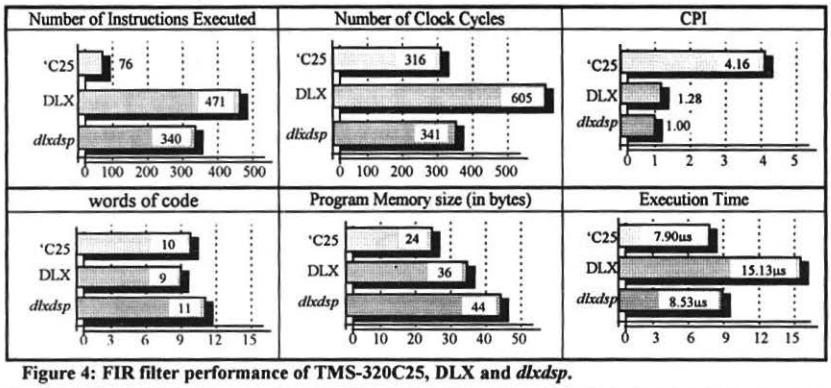

Figure 4: FIR filter performance of TMS-320C25, DLX and *dlxdsp*.

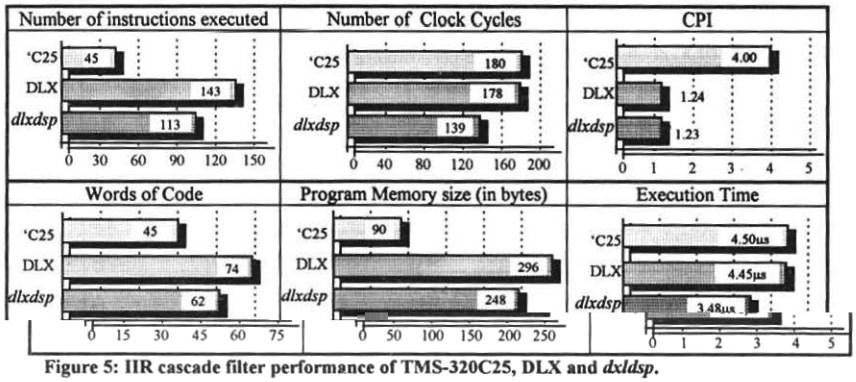

Figure 5: IIR cascade filter performance of TMS-320C25, DLX and *dxldsp*.

Despite the variations in the number of instructions executed, the number of words of code is almost the same for the three processors. The way the algorithm is coded determines loops of different lengths, and thus the small static difference become large in dynamic terms. Another comment is about program memory size. Most of the instructions on TMS-320C25 are coded in 16bits, while on DLX and *dlxdsp* all the instructions are coded in 32bits. Because of this, the memory requirements (in bytes) are bigger on *dlxdsp* and DLX than on 'C25.

6.2. *IIR Filter*

The set of results that typically represents the performance of the three processor is shown in Figure 5. In this case TMS-320C25's `MACD` instruction is not so determinant in this implementation, and although the number of instructions executed is greater on *dlxdsp* and DLX than on TMS-320C25, the fast instruction cycle of RISCs and RISPs maintains the number of clock cycles lower on *dlxdsp* and DLX than on 'C25. This yields a faster Execution Time for *dlxdsp*.

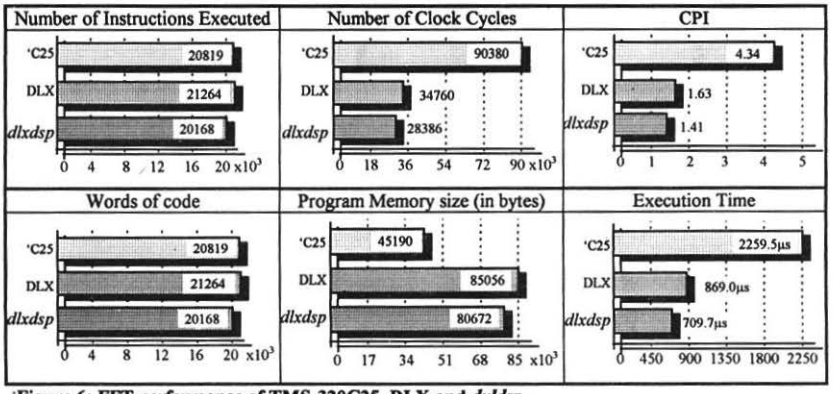

Figure 6: FFT performance of TMS-320C25, DLX and *dxldsp*.

Table 2: DLX, *dlxdsp* and TMS-320C25 performance for other Kernels.

Kernel		*Number of Instructions Executed*	*Number of Clock Cycles*	*CPI Estimation*	*CPU Time*	*Code Instructions*	*Memory* (in bytes)
IIR Filter canonical	DLX	80	110	1,38	2,75µs	21	84
	'C25	26	124	4,77	3,10µs	16	38
	dlxdsp	63	77	1,22	1,93µs	18	72
Adaptive FIR Filter	DLX	811	1284	1,58	32,10µs	349	1396
	'C25	292	1188	4,07	29,70µs	226	460
	dlxdsp	611	949	1,55	23,73µs	282	1128
DCT	DLX	466	589	1,26	14,73µs	23	92
	'C25	138	704	5,10	17,60µs	19	52
	dlxdsp	339	371	1,09	9,28µs	19	76
Matrix Multiplication	DLX	4661	5193	1,11	129,83µs	34	136
	'C25	1129	5456	4,83	136,40µs	33	82
	dlxdsp	4149	4681	1,13	117,03µs	33	132

6.3. FFT

FFT results show an abnormal situation. TMS-320C25 has no resources to improve performance considerably, thus a general purpose processor can easily reach better results. That is the case for DLX. *dlxdsp* has no special resources for FFT either, except for the possibility of using two or more arithmetic units simultaneously, what makes operations with complex numbers faster. Figure 6 shows the results for FFT.

6.4. Other Kernels

The results for the remaining kernels are depicted in table 2. By considering all the algorithms studied, we can extract some typical patterns. A particular analysis comes from program memory size: on DLX it was required 21.11% more words of code then on 'C25 and 110.44% more bytes to store programs (remember that DLX instructions are coded in 32bits and most part for the 'C25 in 16 bits). On *dlxdsp* 10.88% of increased code and 92.68% of increased memory size was observed relatively to the TMS-320C25. The way algorithms are coded strongly determines these

parameters, and sensitivity is also critical. Former versions of these algorithms presented 5% less words of code on *dlxdsp* then on TMS-320C25 [9].

7. COMMENTS AND CONCLUSIONS

Execution time is still the best factor to consider a processor better then the others. Figure 7 shows the speed-up (relation between execution times) of DLX and *dlxdsp* relative to 'C25.

Taking all the results obtained and shown in this work, it is possible to realize that the best characteristic inherited by RISP from RISCs is the fast instruction cycle. An adequate instruction set is also viewed as a determinant factor for the good performance of RISPs.

Our experiments confirmed previous results indicating that general purpose RISC processors can achieve the same performance levels of dedicated processors on DSP applications. We have also shown that even better performance can be obtained with small architectural enhancements to a general purpose RISC. Although we did not consider any cost factors in our study, it seems reasonable to expect that these enhancements would be cheaper than the development of a special purpose processor, designed specifically for DSP. But, despite the advantages of using RISCs as DSP (e.g. lower execution time), costs strongly affect decisions.

Of course, for a specific application one can always build a processor with matching characteristics, which will be the ideal platform for that particular application. As we observed with the FIR-filter results, the TMS-320C25 could be such an example. However, for the execution of distinct types of DSP applications, our results show that RISC processors provide, in general, equal or better performance than conventional digital signal processors. With the present rate of improvements in RISC technology, we can expect RISC processors to assume a predominant role on DSP area in the future.

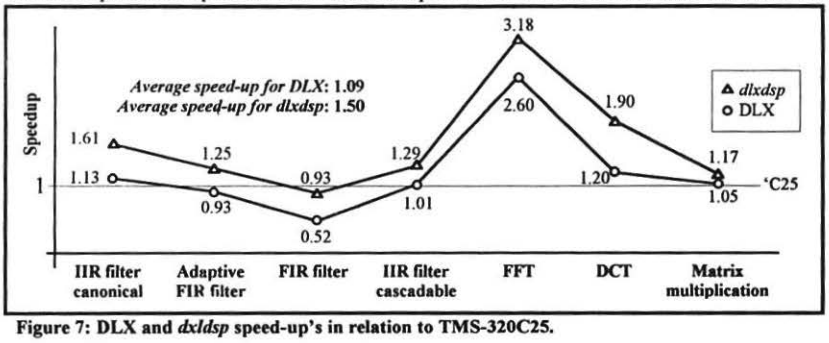

Figure 7: DLX and *dxldsp* speed-up's in relation to TMS-320C25.

7.1. Future work

We have shown that small architectural enhancements to a basic RISC architecture provided significant performance improvements on DSP applications. One can expect that even more improvements would be achieved with the use of superscalar RISC processors. We intend to analyze how different architectural features of superscalar processors would affect the execution of the kernels presented in this work.

Another interesting aspect is the processing of these DSP applications on parallel systems (because code seems to fit parallelism), composed of a collection of RISC processors that communicate by a high-speed interconnection network.

References

[1] ESPONDA, M. and ROJAS, R. A graphical comparison of RISC processors. *Computer Architecture News*, v.20, n.4, p.2-8, Sept. 1992.

[2] HENNESSY, John. and PATTERSON, David A. *Computer architecture*: a quantitative approach. San Marteo:Morgan Kaufmman Publishers, 1990.

[3] LAPSLAY, P. and BIER, J. DSP benchmarks: methodology and results. In: INTERNATIONAL CONFERENCE ON SIGNAL PROCESSING APPLICATION AND TECHNOLOGY, 5, Oct. 18-21, 1994. Texas. *Proceedings...* Texas:[s.n.], 1994. p.871-876.

[4] LEACH, R. Use of RISC processor in multicomputer environments. In: INTERNATIONAL CONFERENCE ON SIGNAL PROCESSING APPLICATION AND TECHNOLOGY, 6, Oct. 24-26, 1995. Massachusetts. *Proceedings...* Massachusetts:[s.n], 1995. P.698-700.

[5] PATTERSON, David A. and DITZEL, David R. The case for the reduced instruction set computer. *Computer Architecture News*, v. 8, n. 6, p. 25-33, Oct. 1980.

[6] TEXAS INSTRUMENTS. *TMS320C2x*: user's guide. [s.l.:s.n.], 1993.

[7] VELARDE, Juan M. et. al. DSPstone: a DSP-oriented benchmarking methodology. In: INTERNATIONAL CONFERENCE ON SIGNAL PROCESSING APPLICATION AND TECHNOLOGY, 5, Oct. 18-21, 1994. Texas. *Proceedings...* Texas:[s.n.], 1994. p.715-720.

[8] WANDERLEY NETTO, Eduardo B. *Uma arquitetura RISC para processamento digital de sinais*, MSc thesis, São José dos Campos:ITA. 1995.

[9] WANDERLEY NETTO, Eduardo B. OLIVEIRA, Rivanaldo S. and SAOTOME, Osamu. RISC processor for digital signal processing purposes. In: INTERNATIONAL CONFERENCE ON SIGNAL PROCESSING APPLICATION AND TECHNOLOGY, 6, Oct. 24-26, 1995. Massachusetts. *Proceedings...* Massachusetts:[s.n.], 1995. p.805-810.