

# A 32-bit RISC Microprocessor with integrated DSP Unit

M. Dolle and M. Schlett

hyperstone electronics GmbH  
Am Seerhein 8  
78467 Konstanz, Germany

**Abstract:** A 32-bit microprocessor will be described which combines the advantages of a general purpose RISC architecture and a fast DSP unit. It will be shown, that the integration of a DSP unit into a RISC architecture using parallelism and efficient sharing of resources enables fast execution of DSP algorithms with the speed advantage of RISC architectures.

## Introduction

Today's DSP processors are usually not based on RISC design principles. Their architecture is mostly driven by applications like video-, image- and voice-processing, by data (de)compression and acquisition, or in general by telecommunication and multimedia applications. Their instruction set is memory oriented and optimized for performing e.g. filter or FFT algorithms, they include dedicated registers, address units and multiply-accumulate units as well as on-chip memory. In summary, DSP architectures [1] are optimized to execute DSP algorithms [2] as fast as possible. This results in powerful instructions but the clock frequency and the area of suitable applications is limited.

In contrast, RISC architectures are based on a load/store principle and a general purpose instruction set using a large register set. They have sophisticated cache strategies and they use pipelining wherever possible resulting in higher clock frequencies.

Recently announced RISC based microprocessors with DSP enhancements just support DSP algorithms by a multiply-accumulate instruction or they provide a dedicated unit for graphics or image processing [3].

The 32-bit processor presented here gives full DSP support using RISC design principles without limiting general microprocessor capabilities. In addition, competitive DSP performance can be achieved without implementing complicate and area consuming hardware.

## RISC Architecture

The block diagram in Fig. 1 shows the basic architecture and the main data paths of the 32-bit RISC-DSP

processor (control paths and some additional hardware features are omitted for simplicity reasons). The main units of the processor are the ALU and the DSP unit which can execute instructions in parallel.

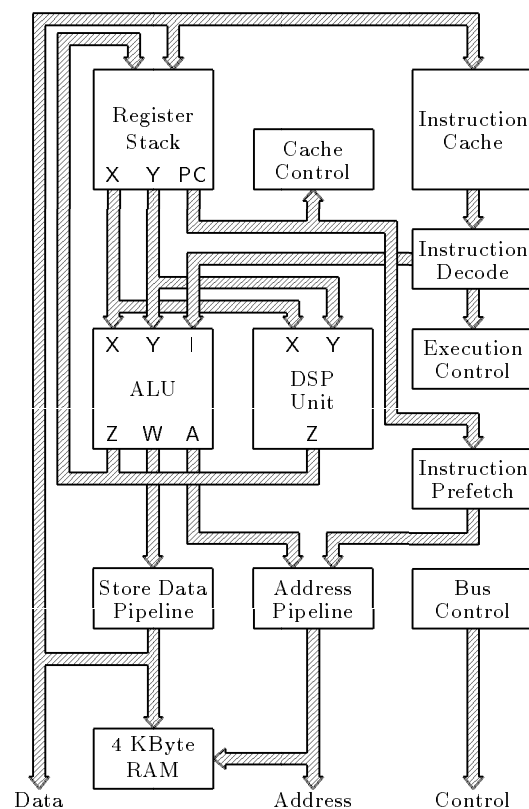


Fig. 1. Basic RISC-DSP Architecture

TABLE I: ALU Instruction Set Overview (not complete)

<b>Load and Store Instructions:</b>	<b>Integer Instructions:</b>	<b>Compare and Branch Instructions:</b>
LD Load from memory	ADD Signed or unsigned add	CMP Compare operands
ST Store to memory	SUB Signed or unsigned subtract	CMPB Compare bitwise
<b>Register Move Instructions:</b>	NEG Signed or unsigned negate	BR Branch
MOV Register Move	MUL Signed or unsigned multiply	BR <sub>cc</sub> Branch on condition
MOVD Register Move (Double)	DIV Divide	DBR Delayed Branch
<b>Logical Instructions:</b>	<b>Shift Instructions:</b>	DBR <sub>cc</sub> Delayed Branch on condition
AND Bitwise AND	SHL Shift left by n bits	<b>Subroutine Call Instructions:</b>
ANDN Bitwise AND (source inverted)	SHR Shift right by n bits	CALL Subroutine call
OR Bitwise OR	SAR Shift right arithmetically	TRAP Trap
XOR Bitwise exclusive OR	ROL Rotate left by n bits	FRAME Restructure register frame
NOT Bitwise Invert		RET Return from subroutine

### A. Pipeline

The processor uses a two stage pipeline consisting of a decode and an execute stage. Due to the regularity of the instruction format, all instructions are read from the internal instruction cache and decoded within one cycle. During the second cycle the instruction is executed. Almost all ALU instructions execute in one cycle: The required source operands are read from their registers, the result is calculated and written into the destination register. The pipeline depth of only two stages enables fast pipeline refill after branches. Using a *delayed branch* instruction where the instruction following the branch instruction is always executed regardless whether the branch is taken or not, no cycle is wasted for branches and a complicate branch prediction unit is superfluous.

In addition, memory accesses are pipelined with a depth of two stages. This means load or store instructions execute in one cycle. The address and the write data are just transferred to the address/data pipeline and execution proceeds independently from the actual memory access. Wait cycles are inserted only when data not yet loaded is needed by an instruction following the load instruction or when the memory pipeline is still entirely occupied.

### B. Register Stack

The RISC architecture provides a set of 64 local registers of 32 bits each. A *register frame* of six local registers is automatically allocated upon subprogram entry. A *frame pointer*  $FP$  points to the base of the current register frame and all registers are addressed relative to this pointer. However, the number of local registers can be dynamically altered by the FRAME instruction up to a maximum frame length of  $FL = 16$  registers. On any succeeding subroutine call a new register frame is allocated by adding the current frame length  $FL$  to the frame pointer  $FP$ . The return PC and the return SR are automatically written into the first two registers of the new register frame. Since register frames can overlap each other as specified by the FRAME instruction, parameters can be easily passed from the calling routine to the subroutine being called as shown in Fig. 2.

If the new register frame does not fully fit into the register stack, the register contents of the oldest register frame are automatically transferred to the external memory. However, assuming an average consumption of 8 registers per subroutine, a sequence of about 8 consecutive subroutine calls can be executed without any push operation. On subroutine return, the RETURN instruction checks whether the return register frame is still entirely in the register stack. If this is not the case, the missing part is automatically reloaded from the memory part of the stack. Hence, the local register set can be seen as the on-chip part of a larger stack. Since the frame pointer  $FP$  and the frame length  $FL$  are part of the status register their values are automatically restored after a return from subroutine.

In addition, 16 global registers including the program counter (PC) and the status register (SR) are provided. Nearly all instructions can use them as source or destination registers. Hence, no special instructions are required for dynamic branches or for modifying the SR.

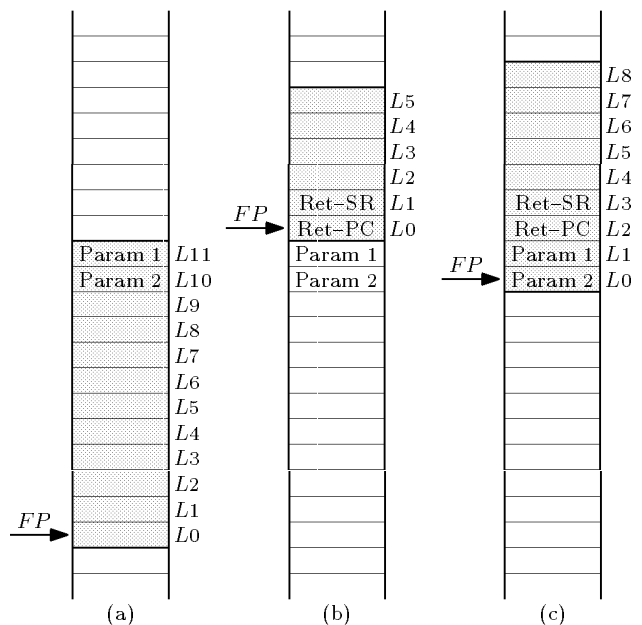


Fig. 2. Local Register Frame (a) before Subroutine-Call, (b) after Subroutine-Call and (c) after FRAME instruction

### C. Instruction Set Overview

The architecture uses a variable instruction length of one, two or three halfwords with a majority of 16-bit instructions. Hence, a larger code density can be obtained compared to architectures using a fixed 32-bit instruction length [4]. In addition, the number of memory accesses required for external instruction fetches is also reduced since memory access is always 32 bits.

A 16-bit instruction uses five bits each for addressing 32 destination and 32 source registers. Alternatively a five bit immediate operand encodes the 32 most often used constants. Only longer and less often used constants and displacements need a 16-bit or 32-bit extension. Table I shows a brief overview of the ALU instruction set.

### DSP Concept

The DSP unit is completely integrated into the 32-Bit RISC architecture and tightly meshed with the other units of the microprocessor. It provides the basic functional kernel of the entire DSP concept and offers instructions for 16-bit and 32-bit arithmetic. Additionally, DSP functionality requires fast loop control, fast branches and fast data accesses. This functionality, however, is already provided by the ALU

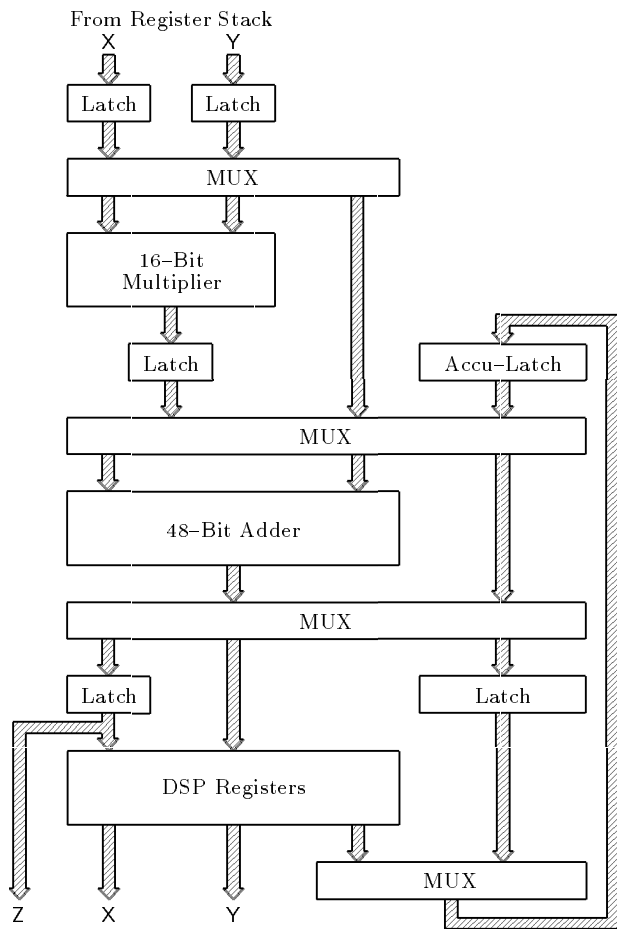


Fig. 3. Simplified Block Diagram of the DSP Unit

instructions. Single-cycle branches and a load/store pipeline enables the processor to execute inner loops of DSP algorithms very efficiently. For single-cycle accesses to often used data a 4 KByte RAM has been implemented on-chip.

A parallel execution concept which is based on the latency cycles of pipelined multi-cycle instructions has been implemented to achieve high-performance DSP processing without the requirement of a superscalar approach. This concept takes advantage of the basic two-stage pipeline with single-cycle ALU instruction execution.

### A. Architecture

Fig. 3 shows the block diagram of the DSP unit. For simplicity several data paths and latches are omitted.

The main part is a multiply-accumulate unit consisting of a 16-bit multiplier array with a modified Booth logic and a 48-bit Carry-Look-Ahead adder. Several intermediate latches used as accumulators are actually implemented as four parallel 16-bit latches. This partitioning is necessary to handle the DSP instruction set which is optimized for 16-bit data types. A 64-bit accumulator is realized by two 32-bit DSP registers, which can also be used as conventional registers because they are part of the general register stack of the microprocessor. Hence, all results calculated by the DSP unit are directly accessible by other instructions via a simple register access.

### B. DSP Instruction Set

The instruction set executed by the DSP unit is strictly based on RISC principles. All instructions operate on register contents. The instruction set provides functionality for 16-bit and 32-bit data types. In case of 16-bit data integer, real and complex fixed-point numbers are supported. Other kinds of arithmetics can easily be programmed — especially in case of 32-bit data types. Instructions for 16-bit data combine four operands within a single operation. Four 16-bit values are located in two 32-bit registers each using the lower or upper 16 bits of the registers. In addition, combining two identical operations into one instruction saves issue cycles and offers a high parallelism without using superscalar features. For an overview of the instruction set executed by the DSP unit see Table II.

### C. Parallelism

There is a strong relation between the parallelism concept and the instructions executed by the DSP unit. Based on its complexity, the DSP instructions need more cycles for completion than the ALU instructions which usually execute in one cycle. In addition to the first issue cycle, the DSP instructions need up to three latency cycles. During these latency cycles the ALU can already execute the next instruction or Load/Store operations can be performed while the DSP instruction is still in execution. The example in Fig. 4 illus-

TABLE II: DSP Instruction Set Overview

Mnemonic	Description	Operands	Cycles (Pipelined)	Operations	Result
<b>Multiply Instructions:</b>					
EMUL	Signed or unsigned 16/32-bit multiply	2 x 16/32b	1-3	1	32/64b
EHCMUL	Signed complex 16-bit multiply	4 x 16b	4	6	2 x 32b
<b>Multiply Accumulate Instructions:</b>					
EMAC	32-bit multiply accumulate	2 x 32b, 32/64b	2-3	2	32/64b
EMSUB	32-bit multiply subtract	2 x 32b, 32/64b	2-3	2	32/64b
EHMAL	16-bit multiply accumulate	4 x 16b, 32/64b	2-4	4	32/64b
EHCMAC	Complex 16-bit multiply accumulate	4 x 16b, 2 x 32b	4	8	2 x 32b
<b>Summation Instructions:</b>					
EHCSUM	Complex 16-bit addition-subtraction	4 x 16b	3	4	4 x 16b
EHCFEFT	Complex 16-bit addition-subtraction for FFT	2 x 16b, 2 x 32b	3	8	4 x 16b

trates this concept. Starting with a DSP instruction, a load/store instruction to access the next data can be issued in the cycle following the DSP instruction. While the access to external memory is still in progress it is possible to execute ALU instructions operating on register contents, e.g. loop control, address calculations or even branch instructions. Hence, the expense for loop control and address calculation can be hidden in the latency cycles of the DSP instructions.

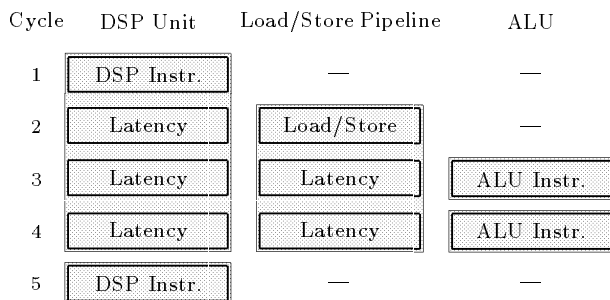


Fig. 4. Parallelism between DSP, Load/Store and ALU

Essential for this approach is the different number of latency cycles required for DSP, Load/Store and ALU instructions. Thus it is possible to parallelize the inner loop of a DSP algorithm, namely the DSP arithmetic part on one side and the loop control and load/store part on the other side. Since this parallelism is not based on superscalar hardware complexity and expense is drastically reduced.

#### D. DSP Performance

This DSP concept offers performance results which increase linearly with the clock frequency. The current implementation is able to perform a 2-dimensional discrete cosine transform with 64 pixels in about 20  $\mu$ s, a 1K complex FFT in about 0.6 ms and FIR filtering in about 2 cycles per filter tap.

#### Implementation

Fig. 5 shows a die photo of the RISC-DSP processor. It consists of about 210,000 transistors and is currently implemented in 0.8  $\mu$ m CMOS technology. The die

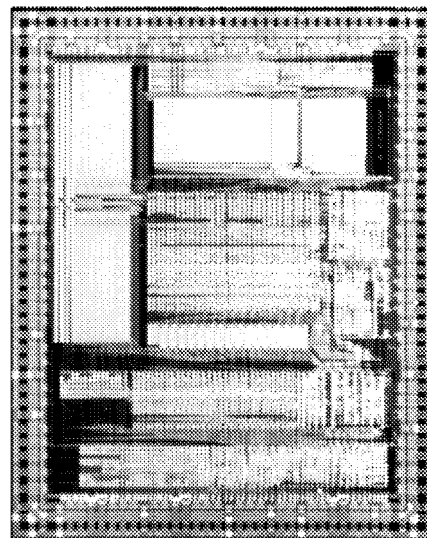


Fig. 5. Die photo of the RISC-DSP processor

size is about 56 mm<sup>2</sup>. The maximum clock frequency is 60 MHz and typical power dissipation is less than 0.8 W. By using a widely available 0.8  $\mu$ m process, production costs are still moderate, making the processor well suited for cost-sensitive embedded systems. Using a 0.5  $\mu$ m CMOS process we will achieve a clock frequency of about 100 MHz.

#### Acknowledgement

We thank the hyperstone design team for their cooperation and especially O. Müller for our fruitful discussions.

#### References

- [1] P. Denyer and D. Renshaw. VLSI Signal Processing. Addison-Wesley Publishing Company, 1985.
- [2] M. Bellanger. Digital Processing of Signals. John Wiley & Sons, 1989.
- [3] M. Dolle and M. Schlett. A Cost Effective RISC/DSP Microprocessor for Embedded Systems. *IEEE Micro*, Vol. 15, No. 5, pp 32-40, October 1995.
- [4] O. Müller. A Novel RISC Microprocessor for Embedded Systems. *IEICE Transactions on Electronics*, Vol. E75-C, No. 10, pp 1196-1201, October 1992.