A GENERAL DSP PROCESSOR AT THE COST OF 23K GATES AND 1/2 A MAN-YEAR DESIGN TIME

Eric Tell, Mikael Olausson, Dake Liu

Department of Electrical Engineering, Linköping University, 581 83 Linköping, Sweden
erite@isy.liu.se

ABSTRACT

This paper describes the design and implementation of a 16-bit fixed point DSP processor. The processor is intended as a platform for hardware accelerators and allows additional computational units and assembler instructions to be added. The I/O facilities can also be customized to the needs of a specific application.

Benchmarking has shown that the processor, without any hardware accelerators, has a performance comparable to single MAC commercial DSP processors. The architecture has been successfully synthesized in a 0.13μm process, resulting in a net-list of about 23000 gates, and a clock frequency of 195 MHz, making the performance/gate count ratio very competitive. It is also small enough to integrate 100 heterogeneous processors on a chip for example for communication infrastructure applications. The complete design time, including architecture and instruction set planning, assembler, debugger, instruction set simulator, RTL code and complete verification was about half a person-year.

1. INTRODUCTION

The processor described was designed due to the need for a platform for hardware accelerators. Application specific hardware accelerators as well as I/O and other peripheral units can easily be added either by connecting them to a memory mapped register file interface or by extending the instruction set with specialized instructions. Half the instruction space has been reserved for this purpose. However the processor should also be able to work without any accelerators as a small and efficient general purpose DSP processor for system on chip integration.

A number of benchmarking algorithms, such as those included in the BDTI DSP benchmarking suite [1] have been used to test the performance. Figure 1 describes an application specific processor design flow, that was used in this project.

Based on a requirement analysis an instruction set is designed. Concurrently the top level architecture is planned to make sure the instructions set can be implemented. The next step is to build a bit true, cycle true behavioral model or instruction set simulator (ISS). Using the ISS, benchmarking is performed to verify that performance requirements are fulfilled, if they are not, the instruction set is modified. The ISS is also used later in the flow for verifying the RTL implementation and for software development. When benchmarking results are satisfactory, top-down design of the complete processor is carried out, followed by HDL implementation.

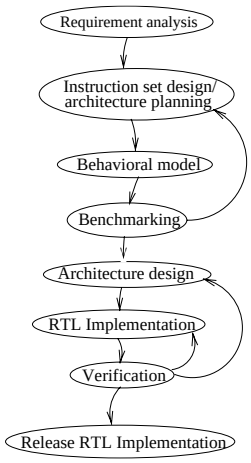
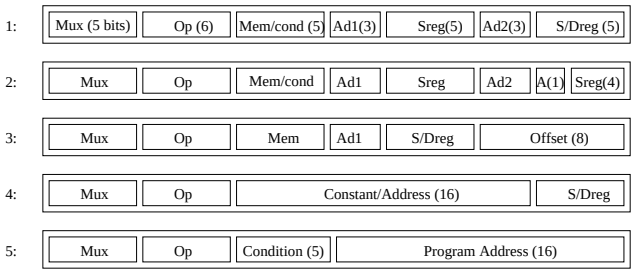


Fig. 1. The design flow

2. INSTRUCTION SET

The processor uses a 32-bit instruction word, which is quite long considering the rather small instruction set (about 75 instructions). A smaller instruction word could have been sufficient, but the longer instructions give a number of advantages such as better orthogonality which leads to simpler decoding, possibility to address many (32) source and destination registers, possibility to use 16-bit immediate data and plenty of space for accelerator instructions and future improvements.

In addition all computational operations, except those using 16-bit immediate data, can be used in one of two



Field:	Function:
Mux	Multiplexer switching between different instruction
groups Op	Operation code choosing the actual instruction
Mem/Cond	If the first bit of this field is zero, the rest of the bits specify what memory accesses are made in the instruction. If the first bit is one, conditional execution is enabled, and the rest of the bits specifies a condition for executing the instruction.
Ad1/Ad2	Pointers to address register for data memory addressing
Sreg	Pointer to source register (operands for instructions)
Dreg	Pointer to destination register
A	One bit selecting what accumulator register to use
Offset	Offset for offset addressing
Condition	Conditional execution, same as above

Fig. 2. Instruction word examples

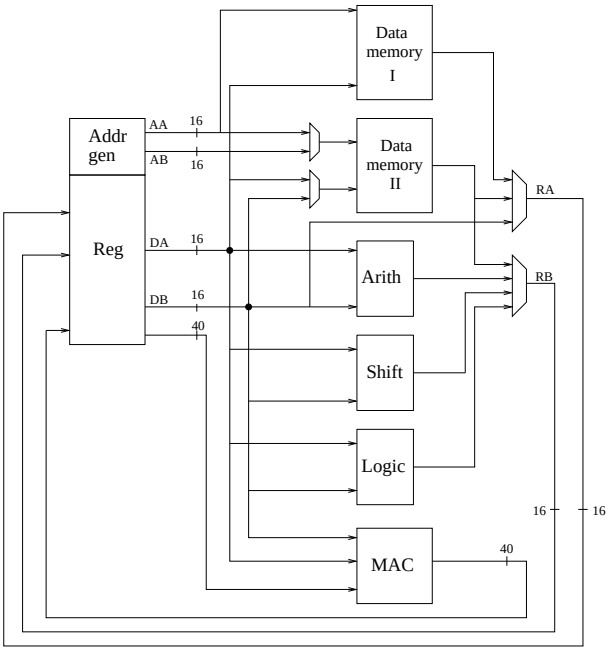


Fig. 3. Data path architecture

modes: If conditional execution is enabled a condition is specified with the instruction and if the condition is not true, the instruction is replaced by a nop instruction. The alternative mode allows one or two memory access operations to be executed in parallel with a computation.

Figure 2 gives examples of instruction formats. The first one is used for most computational instructions with two registered operands. The second is used for example for multiply and multiply-accumulate instructions. Both these formats allows one or two parallel memory operations or conditional execution. The third format is used by memory operations using offset addressing, the fourth is for instructions with immediate 16-bit data or immediate data memory address and the last format is used by (most) program flow instructions.

3. DATA PATH

The processor has a 16-bit fixed point data path with the following computational units: *Arithmetic unit* for addition, subtraction and other arithmetical operations. All arithmetic unit operations are executed with or without saturation. The mode is determined by a bit in the control register. *Logic unit* for bitwise 'and', 'or', 'xor' and 'not, and bit test operations. *Shift unit* for arithmetic shift, logic shift and rotation operations. *MAC unit* for multiplication and multiply-accumulate (MAC) operations.

Data is stored in two data memories and a register file with 32 general purpose registers of which 8 are also used as data address registers. Four of the registers also make up two accumulator registers for storing 40-bit results of mac-unit operations (one register in each pair is extended with eight guard bits).

All computations use only operands from registers, making this a real RISC architecture. The fact that address registers are also general registers improves orthogonality. Both these facts simplifies debugging.

3.1. The MAC Unit

The MAC unit has a 17x17-bit multiplier, a 40-bit adder and a 40-bit barrel shifter. It can execute multiplications and MAC operations on either integer or fractional numbers, 16- and 32-bit additions, 40-bit shift up to 32 steps, rounding and saturation. For MAC and 32-bit add operations, eight guard bits are used. MAC and multiply operations are pipelined in two stages.

3.2. Parallel Operations and Data Hazard Management

The data path allows up to one mac unit operation and two memory operations or one mac operation, one arithmetic, logic or shift operation and one memory operation to be executed every clock cycle. The instruction set however, imposes restrictions on which combinations of operations can actually be executed in parallel.

The risk of parallel instructions trying to write to the same register in the register file is in most cases eliminated

by restrictions in the instruction set. When this is not the case, result bus B (RB; see figure 3) has priority over the MAC result bus, and result bus A (RA) has priority over both RB and the MAC bus in determining what value is written. Whenever a memory write conflict occurs, an error flag is set in the status register.

4. DATA ADDRESSING

All eight address registers support direct addressing, offset addressing and post incremental addressing. For post incremental addressing one of the registers has the step size programmable to any size and four of the registers has step sizes programmable to between 0 and 15. The remaining three registers always uses step size one. Two of the registers support modulo (circular) addressing and one supports bit-reversed addressing for efficient implementation of FFT algorithms. Some instructions use an immediate 16-bit address from the instruction word.

Step sizes for post incremental addressing and modulo addressing ranges are programmed via registers. Modulo and bit-reversed addressing is enabled via bits in the control register.

64 kWords can be addressed for each data memory.

Using three bits of the machine code for address register selection, and hence eight address registers, has been sufficient for all DSP kernel algorithms that has been implemented. It is also enough to implement a CELP CODEC as in [5]. When it comes to modulo addressing possibilities, which are typically used for implementing circular data buffers, it is also believed that providing this for two of the address registers is enough for most applications. For efficient FFT implementation it is necessary to allow post-incremental address register update with an arbitrary step size for the address register supporting bit-reversal. For most other applications however, only small step sizes are useful, one being the dominating step size.

The design decisions regarding these features are all trade-offs between instruction word length, hardware size and programming complexity.

5. CONTROL PATH

5.1. Pipeline

The architecture has a variable pipeline depth with three or four stages (instruction fetch, instruction decode and one or two execution stages). The four stage pipeline is used by multiply and MAC operations only. Using only one cycle for execution and result saving eliminates data hazards. However, a four stage instruction followed by a three stage instruction may still cause data hazard. This is handled automatically by halting the pipeline if a four step instruction

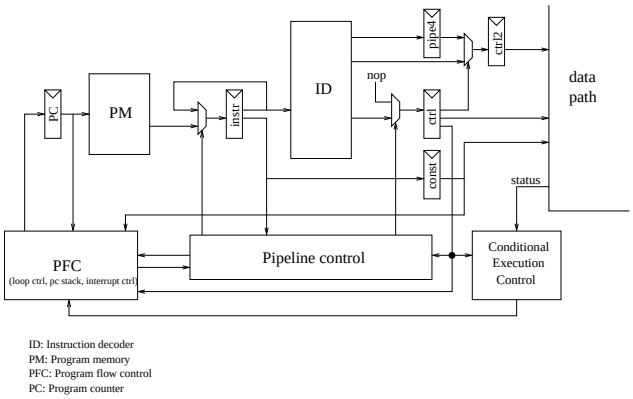


Fig. 4. Control path

is followed by an "incompatible" three stage instruction using some of the same registers. Since the destination of mac unit operations are limited to one of the two accumulator registers this is a quite simple check to implement.

Delayed jumps are used for both conditional and unconditional jumps, subroutine call and subroutine return instructions.

5.2. Hardware Looping

The instruction set includes two instructions for zero overhead hardware looping. The *repeat* instruction repeats only one instruction a number of times while the *loop* instruction handles larger loops, and also allows loop nesting. The reason for having two instructions is that the pipeline makes it difficult to handle very short loops (one or two instructions) in the same way as longer loops.

5.3. Interrupts

The Processor handles 16 different interrupts. Two of these are reserved for the two DMA channels. The other 14 can either be used as external interrupt pins or be associated with I/O devices, timers, hardware accelerators and other units connected to the expansion interface (as described in 7 below).

Interrupt vectors are programmable via memory mapped registers and interrupts are software nestable (interrupt can be explicitly reenabled in interrupt routines to allow up to four nested interrupts).

6. PERIPHERALS

The processor has two *DMA channels* that can be used to transfer blocks of data between memories or between memory and I/O devices. Optionally DMA transfers can be trigged by interrupt signals. For example data arriving at an input

port could trigger a DMA controller to move the incoming data to a data buffer area in memory. The DMA controllers are programmed via memory mapped registers

Via a *host port* the entire (data and program) memory of the DSP can be accessed for example by a host processor. This includes the memory mapped registers, which means that for example DMA transfers can be set up by the host.

An arbiter determines which of the two DMA channels, the host port and the cpu has access to each of the two memories. Each of these four units can be given either high or low priority. All units with high priority are always granted access before any unit with low priority.

7. EXPANSION INTERFACE

Up to 128 addresses in "data memory I" are mapped to registers. Some of these registers are used to set up interrupts, DMA etc, but for example the 32 general purpose registers are also memory mapped and can therefore be accessed via the host port, for example for debugging purposes.

The remaining 64 registers are intended as an interface for communication with application specific peripherals for example different kinds of ports, timers and hardware accelerators. In this case the registers are configured as either input (read only) or output registers. As mentioned above these peripherals can also easily be connected to one (or more) of the 14 general interrupt signals.

8. RESULTS

In addition to RTL code, an assembler and a cycle-true instruction set simulator has been developed. Using the simulator a number of kernel benchmark algorithms have been executed. The results are shown in table 1.

The results where compared to results from benchmarks performed by BDTI [1] where equivalent benchmarks exists. The approximate average cycle count from these benchmarks are also shown in the table. For all these benchmarks there are some processors that are better and some that are worse than this one. It should be noted that some of the processors are high-end processors with dual-mac units which gives a significant advantage in many benchmarks.

The DSP was synthesized using Cadence Physically Knowledgeable Synthesizer (PKS) resulting in a netlist of about 23000 gates of which 12600 corresponds to the register file (including decoding and address generation logic) and 3300 to peripherals (including 2700 for the two DMA controllers). The maximum clock frequency was estimated to 195 MHz in a 0.13μm process.

Function	**Cycles**	**BDTI avg.**
64-Point Discrete Cosine Transform	863	
256-Point Fast Fourier Transform	18763	10000
Division, 16/8 bits	60	
Real Block FIR Filter, 16 taps, 40 samples	882	900
Real Single Sample FIR Filter, 16 taps	22	22
Complex Block FIR, 16 taps, 40 samples	3644	3000
16-bit Integer to Floating Point	14	
Floating Point to Integer Conversion	14	
IIR Filter, 40 samples, 8 biquads	2435	
LMS Adaptive FIR Filter, single sample	87	65
LMS Adaptive FIR Filter, 16 taps 40 samples	3692	
Vector Add, 40 samples	102	85
Vector Dot Product	45	45
Vector Max, 40 samples	50	123
Block Transfer, 40 samples	80	

Table 1. Benchmarking results

9. CONCLUSIONS

A 16-bit fixed point DSP processor has been designed. The processor is intended as a platform for hardware accelerators and is prepared for this by means of the memory mapped register file and by reserving half the instruction space for adding acceleration related instructions.

It has also been shown that a DSP Processor IP core can be small enough to allow for up to at least 100 heterogeneous processors to be integrated on a chip for communication infrastructure applications.

Without accelerators the performance is still quite good. The performance/gate count ratio is very good. Very few comparable processors seem to have a gate count below 30000.

Furthermore the employed design methodology has proven to support high quality processor design using a short design time.

10. REFERENCES

[1] Berkeley Design Technology, Inc, *Buyer's guide to DSP Processors*, 1999.

[2] Eric Tell, master's thesis, *A Domain Specific DSP processor*, Linköping University, 2001.

[3] David A. Patterson, John L. Hennessy, *Computer Organization & Design - the hardware/software interface* (second edition), Morgan Kaufman, 1998.

[4] Phil Lapsley, Jeff Bier, Amit Shoham, Edward A. Lee, *DSP Processor Fundamentals*, IEEE Press, 1995.

[5] Mikael Olausson, Dake Liu, *Instruction and Hardware Acceleration for MP-MLQ in G.723.1*, SIPS'02, San Diego, CA, USA, 2002