Ročník 2011 electroscope Číslo IV

Digital Signal Soft-Processor for Audio and Video Processing

M. Pristach[1], A. Husár[2], L. Fujcik[1], T. Hruška[2], K. Masařík[2]
[1] Dept. of Microelectronics, FEEC, BUT Brno, Technická 10, 616 00 Czech Republic
[2] Dept. of Information Systems, FIT, BUT Brno, Božetěchova 2, 612 66 Czech Republic
E-mail: xprist00@stud.feec.vutbr.cz, ihusar@fit.vutbr.cz, fujcik@feec.vutbr.cz,
hruska@fit.vutbr.cz, masarik@fit.vutbr.cz

Abstract:
The paper presents a digital signal soft-processor DVSP that was designed using architecture description language ISAC by a tool for processor design called Lissom. First, a basic version of the processor with simple RISC instruction set was designed. Then, based on the target audio and video-processing applications, several instruction set extensions were added. Architecture of the designed processor is described in this paper. Processor was synthesized for several Xilinx FPGAs and synthesis and performance results are presented.

INTRODUCTION

Digital signal processing applications require increasingly higher throughput while maintaining real-time performance. High-performance digital signal processors may be used, however, processors that run on high frequencies consume a lot of energy and are more expensive than simpler processors.
To optimize cost and power of a processor with respect to a certain application and performance requirements, application-specific instruction set processors (ASIPs, e.g. [1]) are used. This paper presents an application-specific digital signal processor DVSP (Digital Video Soft-Processor) that was designed for audio and video processing.

Design and optimization of an ASIP is a process that is very time-consuming and requires people with special skills. In order to make ASIP design more effective, special tools that accelerate this process are needed. One of these tools is developed in project Lissom.
The Lissom project is centered on an architecture description language (ADL) ISAC [2]. Architecture description languages are described in [3]. Both the instruction set architecture and the micro-architecture (the processor pipeline) can be described in ISAC. Then, from the model in the ISAC language, C language compiler, assembler, several types of simulators [4] and synthesizable hardware representation in VHDL can be automatically generated.

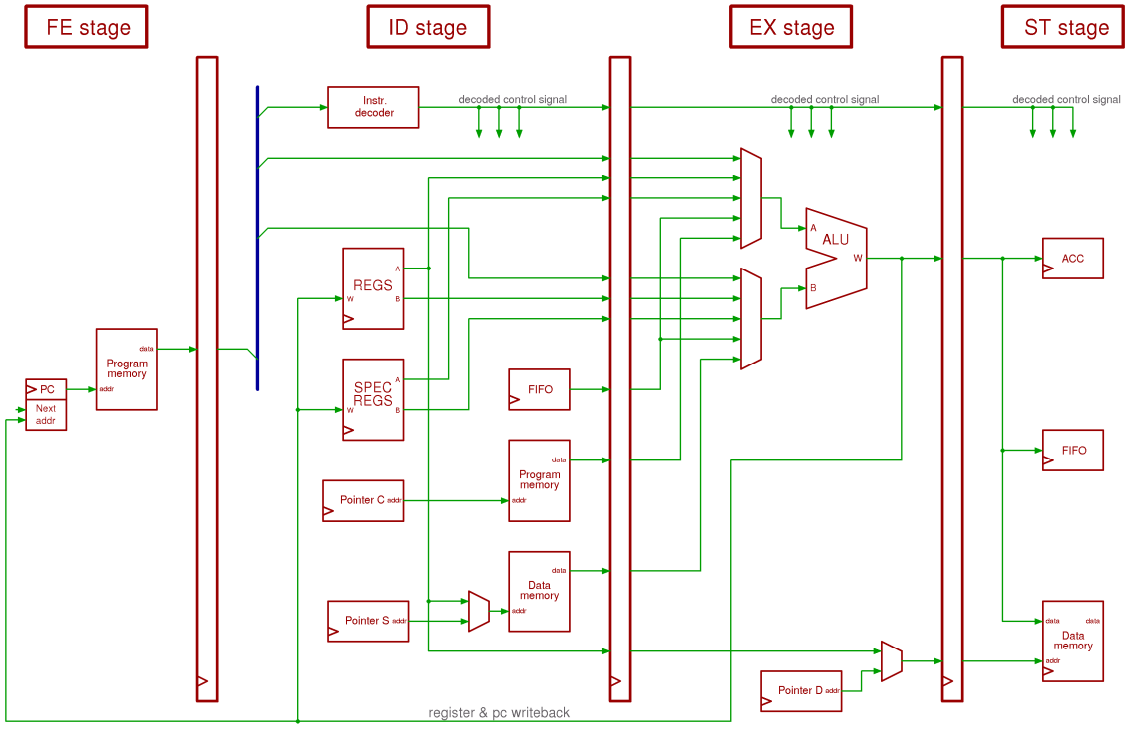

Fig. 1: Block diagram of processor DVSP

DESIGN OF THE PROCESSOR

There are several steps in an ASIP design. First, a base instruction set must be designed; this design is already affected by the target application but special instructions may not be included yet.
This basic instruction set and especially its binary coding must be prepared for a large number of extensions. Initial design of the processor pipeline can be done at this design stage. A guide for designing soft-processors on FPGAs can be found in [5], chapter 11.
Then the whole processor was described by the ISAC language. From this description, assembler and simulator to debug and optimize the design were generated. Also VHDL code was generated and synthesized in order to determine processor's physical properties.
Because there is only one description of the processor in the ISAC model, changes can be done quickly in one place and they are immediately reflected in the generated tools and hardware.
After verifying the basic version of the processor, modifications with respect to the target application or application domain are done.
Several instruction set extensions were added to the base instruction set to accelerate specific parts of the target application. The resulting processor DVSP and impact of the instruction set extensions on performance are described in the following sections.

ARCHITECTURE OF THE PROCESSOR

Processor DVSP is a 16-bit digital signal soft-processor designed mainly for implementation on FPGAs. The instruction set is partially based on a low-power microcontroller MSP430 from Texas Instruments [6], however compared to the MSP430, the micro-architecture is pipelined and highly optimized for implementation on FPGAs.
The processor utilizes Harvard architecture with separated address spaces for data and program. Instructions are of fixed length of 32 bits. Data memory is 16-bit wide and is addressed by 8-bit bytes. The base data type of the processor is a 16-bit signed value coded in two's complement. The architecture defines two register fields, each register is 16 bits wide. The first register field contains 16 general purpose registers, the second one contains 16 special registers.
Several special features like support of zero-overhead cycles, multiplication with 32-bit result, 32-bit accumulator and access to data or program memory by pointers with auto-increment and modulo addressing were added. Block diagram of the processor in simplified form is shown in Fig. 1.

Instruction set

Instructions are divided into four groups. The first group contains arithmetical and logical instructions: additions, subtractions, multiplications, shifts and logical operations. The second and third groups contain unconditional and conditional movements, and jumps. Conditions of moves and jumps are based on signed comparison of source data with zero. The fourth group contains special operations such as indirect access to data memory (load, store), load constant from program memory, call subprogram and return from subprogram.
Binary coding of any instruction can be divided into two parts: operation code and operands. Operation code is 8 bits wide, 6 bits define the operation, one bit defines accumulator mode usage and one is reserved for the future extensions. Arithmetical, logical and movement instructions consist of three operands – one destination and two sources. Destination and each source operand can use any addressing mode. Processor supports up to 16 addressing modes. In this version there are 7 different modes supported: access to the general or special registers field, positive or negative constant, access to input/output queues, access to memory with address from a general register and access to memory through pointers with optional auto-increment.
To accelerate 32-bit arithmetical operations, special instructions such as multiplication of signed and unsigned values, or support for computations with carry flag were added.

Pipeline description

The DVSP has a 4-stage pipeline that consists of the following stages:

- FE (Fetch) – instruction fetching and program counter value modification; zero-overhead loop support is implemented in this stage as well,
- ID (Instruction Decode) – instruction decoding, register value reading, memory address calculation (for load and store instructions), initiation of memory access, initiation of input FIFO access,
- EX (Execute) – finalization of memory or input FIFO access, result calculation, condition evaluation, register and program counter write-back,
- ST (Store) – accumulation to the 32-bit accumulator, storing result to data memory or to an output FIFO.

The whole pipeline may be stalled in the case that empty input FIFO is read in the ID stage or when an instruction wants to write into full output FIFO in the ST stage. No operand value forwarding (e.g. [7]) is supported. Instructions that write to a register have 2 cycles latency. This allows us

to save area and increase working frequency while keeping the pipeline simple. Latency of 2 cycles for access to registers usually does not impose significant performance limitation because pairs of independent instructions can be usually found in the source code (e.g. loop unrolling creates such independent instructions).

Input and output queues

The processor uses input and output queues for communication with the outside world, FIFO buffers are used for continuous data flow processing. They are implemented outside the processor where the user can easily set their size. One of the typical interconnections into a multi-core processing platform is shown in Fig. 2 where three processors DVSP are connected by FIFOs.

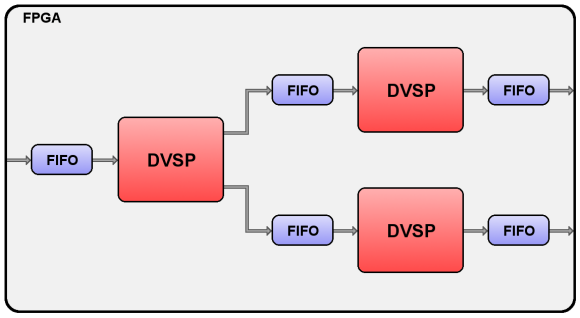

Fig. 2: Block diagram of typical usage in a FPGA

Interconnection through FIFOs is especially suitable for multi-core stream-processing applications, where several processing filters can be mapped on each core. Then the hardware FIFOs and pipeline stalling will automatically take care of data transfer scheduling and synchronization.
Several programming models such as Stream-IT [8] or other dataflow-based approaches can be used to program such multi-core processor.

Access to memory with pointers

The processor supports two means of accessing the memory. For data memory, basic access with load/store instructions through address stored in a register field with offset is available, also an addressing mode that uses register value as address is available. The other possibility is using special registers called pointers which provide access to memory indirectly. There are two pointers for data memory – one for reading and one for writing. For program memory, it is possible to directly load a constant using instruction MOVI, or to use code pointer register for reading constants stored in the program memory.
Each pointer supports auto-increment which gives the ability to quickly access sequential data or constants.

Pointers can also be used for cyclic buffers for repetitive loading/storing of data from addresses in a given range (modulo addressing).
Dedication of special registers for auto-increment and modulo addressing complicates compilation from the C language, but greatly increases the processor performance and data throughput.

Accumulator

The DVSP contains one 32-bit accumulator to accelerate typical DSP operations. The accumulator is placed in the store stage and uses output of the arithmetic-logic unit (ALU). Accumulator is very useful in applications that compute sum of data like convolution, correlation or total spectral power of input signal. Detailed scheme of the accumulator is shown in Fig. 3.

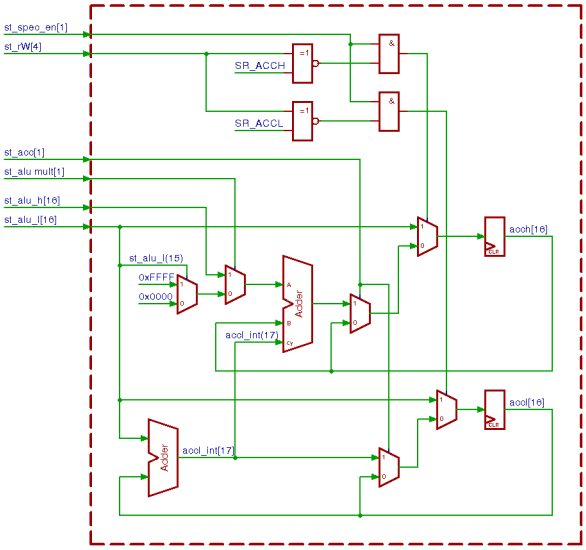

Fig. 3: Detailed scheme of accumulator

Zero-overhead cycles support

The most time-consuming parts of a DSP application are executed in for-loops. When only standard RISC (Reduced Instruction Set Computer) instructions are available, in each for-loop, loop counter must be modified, loop end condition calculated and a conditional branch must be resolved. This requires at least 2 instructions; also some of the branch delay slots may stay unused. Loop unrolling can be done, but this has a negative effect on the code size.
Zero-overhead cycles support hardware uses specially designed control of next instruction address calculation. This provides execution of loops without any unnecessary instructions. Before loop start, three special registers for loop control are initialized. This special feature saves up to 5 instruction clock cycles for each loop iteration. Detailed scheme of the program counter with zero-overhead cycles support is shown in Fig. 4.

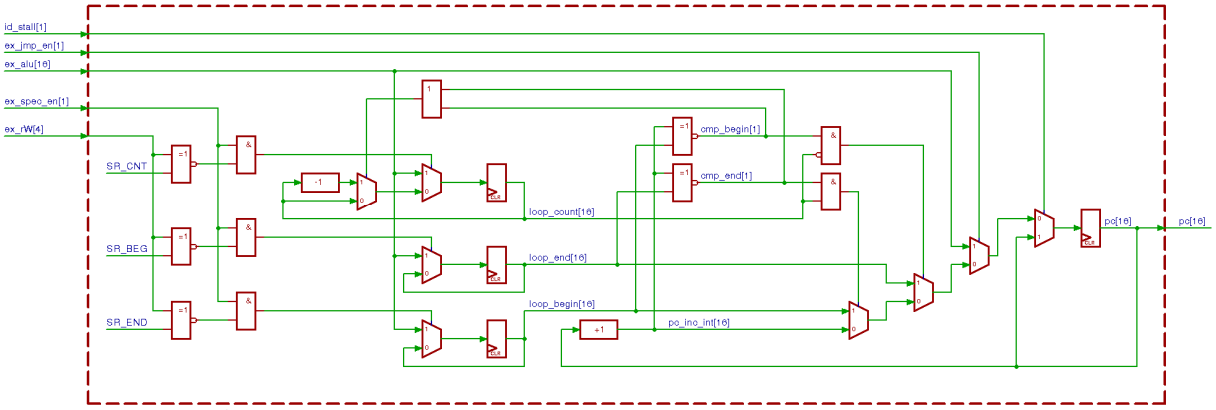

Fig. 4: Detailed scheme of program counter with zero-overhead cycles support

APPLICATIONS OF THE PROCESSOR

The DVSP is dedicated to signal processing applications. The main application for which the processor has been optimized was to calculate the spectral power of the input signal. Fast Fourier Transformation (FFT) based on Cooley-Tukey algorithms with decimation in time [9] was used. The algorithm was written in assembly language and optimized for this processor. Internal operations use 32-bit arithmetic to obtain more accurate results. Spectral power calculation of one FFT window (order 8, 256 samples) takes about 135 000 cycles.

The processor is suitable for other applications like FIR, IIR filter or control checksum calculation. For example, calculation of CRC16 for 1 byte of input data takes 8 cycles, for CRC32 it takes 11 cycles.

IMPLEMENTATION AND RESULTS

The processor was implemented in FPGAs Xilinx Spartan 3 and Spartan 6. The result of synthesis for different FPGAs is shown in Tab. 1 where utilization of area and maximum clock frequency are compared. The results were obtained by Xilinx ISE Design Suite 12.4. Look-up tables (LUT) are used only for core and registers fields. Program and data memory are synchronous and they are placed in BRAM blocks. Read only memory (ROM) or read/write memory (RWM) can be used as program memory. When RWM is used, it is necessary to load the program into the memory, for example by serial peripheral interface (SPI). Usage of SPI gives the possibility to quickly change the program without requiring the whole design reimplementation during development.

Tab. 1: Comparison of implementation in FPGAs Xilinx

Family	***Device***	***LUT [-]***	***f_{clk} [MHz]***
Spartan3	xc3s200-5	1846	67.588
Spartan3e	xc3s500e-5	1897	76.892
Spartan6	xc6slx16-3	1377	62.041
Virtex4	xc4vlx25-12	2097	124.723
Virtex5	xc5vlx30-3	1458	139.447
Virtex6	xc6vlx75t-3	1400	120.683

During development, the processor was tested on convolution algorithm application (FIR filter). Impact of special features on execution time is shown in Fig. 5. The program was written in assembly language and the result was calculated from 4 samples. Five versions of the processor were compared:

- version A – general purpose processor without any special features,
- version B – previous version with accumulator added,
- version C – previous version with zero-overhead cycles support added,
- version D – previous version with one pointer for program memory added,
- version E – previous version with two pointers for data memory added.

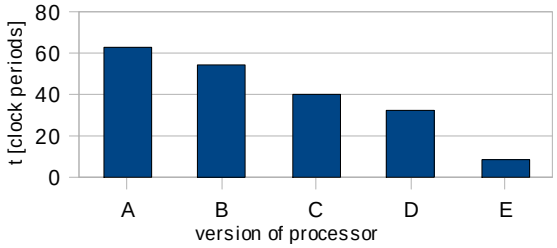

Fig. 5: Effect of instruction set extension on cycle count of convolution calculation

CONCLUSION

This paper presents design of the digital signal processor DVSP that was optimized for several audio and video processing algorithms, mainly for spectral power calculation that uses Fast Fourier Transform.
The processor was designed using an application-specific processor design tool Lissom that uses architecture description language ISAC to capture the processor's instruction set and pipeline design.
The Lissom tool provides tool-chain and hardware description generators that allow regeneration of all the required programming and simulation tools and hardware description after a change in the processor design was made. This approach provides fast design space exploration where the designers can easily test what impact the changes have and it allows them to quickly find a suitable balance between the processor performance and used area.
Original instruction set did not offer enough processing power, therefore several instruction set extensions and other modifications were introduced. These changes allowed processing of input data with 7 times higher throughput, while the area increased only 2 times.
Another important aspect of the processor design is usage of input and output FIFOs that offer a great advantage for multi-core stream-processing applications. When using streaming programming model, each instance of the processor core can serve as a DSP filter and the FIFOs take care of synchronization automatically. We plan to explore this problematic further because the size of required area allows us to place multiple instances of the DVSP processor even on a small FPGA.

ACKNOWLEDGMENTS

The research has been supported by project Prospective applications of new sensor technologies and circuits for processing of sensor signals No. FEKT-S-11-16, by the grant of MPO Czech Republic FR-TI1/038 and by the Research Plan MSM No. 0021630528.

REFERENCES

[1] Ienne, P., Leupers R., *Customizable Embedded Processors: Design Technologies and Applications*, Morgan Kaufmann, 2006, ISBN 978-0-12-369526-0.

[2] Masařík, K., Hruška, T., Kolář, D., *Language and Development Environment For Microprocessor Design Of Embedded Systems*, In: Proceedings of IFAC Workshop on Programmable Devices and Embedded Systems PDeS 2006, Brno, CZ, FEEC BUT, 2006, p. 120-125, ISBN 80-214-3130-X.

[3] Mishra, P., Dutt, N., *Processor Description Languages*, Morgan Kaufmann, 2008, ISBN 978-0-12-374287-2.

[4] Přikryl, Z., Hruška, T., Masařík, K., Husár, A., *Fast Cycle-Accurate Compiled Simulation*, In: 10th IFAC Workshop on Programmable Devices and Embedded Systems, PDeS 2010, Pszczyna, PL, IFAC, 2010, p. 97-102, ISSN 1474-6670.

[5] Nurmi, J., *Processor Design: System-On-Chip Computing for ASICs and FPGAs*, Springer, 2007, ISBN 978-1-4020-5529-4.

[6] Texas Instruments: *MSP430x2xx Family User Guide* [online], Texas Instruments, 2011 [cit. 2011.09.05]. Available from WWW: http://www.ti.com/lit/ug/slau144h/slau144h.pdf.

[7] Shen, J. P., Lipasti, M., *Modern Processor Design: Fundamentals of Superscalar Processors*, McGraw-Hill, 2004, ISBN 978-0-07-057064-1.

[8] Gordon, M. I., *Compiler Techniques for Scalable Performance of Stream Programs on Multicore Architectures*, PhD thesis, Massachusetts Institute of Technology, 2010.

[9] Lyons, R. G., *Understanding Digital Signal Processing, 2nd Edition*, Prentice Hall, 2004, ISBN 978-0-13-108989-1.