Hardware / Software Co-design of a SIMD-DSP-based DVB-T Receiver

H. Seidel, G. Cichon, P. Robelly, M. Bronzel, G. Fettweis
Mobile Communications Chair, TU-Dresden
D-01062 Dresden, Germany
seidel@ifn.et.tu-dresden.de

Abstract

The demand for increasing design complexity has to face decreasing design time and design cost issues. Traditional full custom design flows are not reasonable to cope with this challenges. We have developed a design exploration platform which provides access to main chip design parameters beginning from early design stages. With this platform, architecture and hardware/software partitioning failures are correctable with low change cost in a short time. Our designs are based on Synchronous Transfer Architecture SIMD DSP Cores with a tailored instruction set. Automatic Core Generation Tools speed up the design and verification process. In this paper we present our design exploration platform and a use case study for a DVB-T/H like OFDM-Receiver and Transmitter build with our design methodology.

– SFB358-Demonstrator[1]

1. Introduction

Highly optimized single instruction multiple data (SIMD) DSPs are used for state of the art signal processing. They fill the gap between full custom ASICs and general purpose DSPs. Powerful tools are needed, for fast and efficient design and validation of those DSPs. The Synchronous Transfer Architecture (STA) presented in [3] and [1] provides a compiler friendly architecture template for SIMD-DSPs. STA-based DSP cores with an optimized application specific instruction set are generated automatically with GenCore [3]. Compared to the more general processor design solutions like CoWare/LisaTek [11][7], PEAS-III [5] and EXPRESSION-ADL [4] which are targeted to RISC/CISC architectures, GenCore is targeted to DSP architectures with special SIMD support. In our case of study we have developed a SIMD DSP for OFDM broadcast transmission and reception. GenCore has been used to generate a CoWare/LISAtek-based instruction set simulator (ISS), the VHDL Model and the development tools. The design is implemented on Stratix25 FPGA Dev. Boards and embedded in an Octave[2] Simulation. This enables automatic design validation of DSP and firmware with the reference system.

This paper gives an overview of our current design flow with GenCore. We present our different hardware and software validation techniques based on a multi-dimensional design space exploration platform.

2 Design Space Exploration Platform

Starting point of our system development is a SIMD-DSP machine description and a Octave language reference implementation of the firmware. In the machine description all elements of the DSP are defined. Bitwidth of the interconnections, number of register files and their entries, functional units with a behaviour description and memories. After algorithm evaluation the basic elements of the DSP are determined. GenCore is fed with a first machine description to generate LISA description[3] and VHDL framework. Now, the Octave programs are either compiled with our tensor compiler [9] and STA-backend [2] or implemented in assembly language.

With the instruction set simulator, clock cycle count and computational accuracy are investigated. The GenCore VHDL-framework consists of Memories, Register files, Decoder and the interconnection matrix. The Functional Unit(FU) behaviour section is filled manually. Therefore, a behavioural verification of the FUs is essential. After synthesis of the DSP-VHDL-Description, die size (or number of FPGA cells) and maximum clock frequency are calculated.

The functional behaviour of VHDL-Model and ISS is kept identical, since a single description is used to gener-

[1]This work has been supported by the Deutsche Forschungsgemeinschaft, Sonderforschungsbereich 358

[2]GNU Octave - Matlab^{TM} and GNU Octave provide a common language for effiecent signal processing algorithm implementation

[3]With the LISA description an instruction set simulator, assembler and linker tools are generated

ate hardware model and instruction set simulator. Moreover, the assembler output is suitable for the instruction set simulator as well as for hardware simulation and FPGA implementation.

2.1 Architecture Design Validation

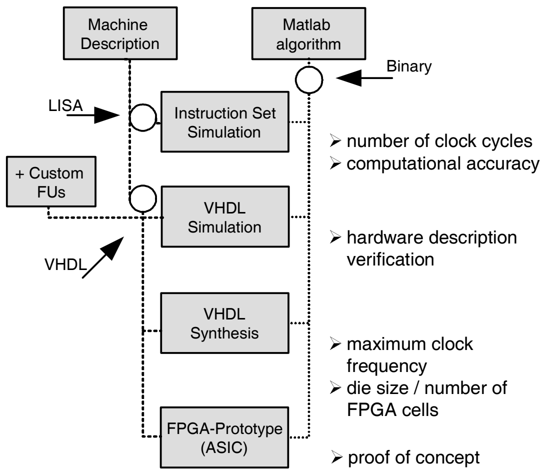

Figure 1. Hardware / Software Co-exploration design phases

The system is implemented in an Octave testbench to enable the validation with the reference system in each design phase. Within this testbench all design implementations are tested against each other permanently. Figure 1 shows the different design phases and obtainable design parameters.

Each design phase is subdivided into two verification processes:

- **instruction set check**: During this check the behaviour descriptions of all instructions are tested. Hence, every instruction has to have an corresponding testprogram and testbench.
- **functional test**: The functional test ensures that software fulfills computational accuracy and timing constraints.

Each design has to pass the following design phases:

- **Instruction Set Simulation** In this design phase computational accuracy and timing (time measured in clock cycles) can be either improved by DSP architecture change or optimization of the implemented software. Hence, machine description or assembly/Octave program are edited.
- **VHDL Simulation** The VHDL-Model is tested against ISS and Octave reference. This is done by comparing the waveform plot created by the VHDL-Simulation with the waveform plot created by the ISS. If the VHDL-implementation is error-free: VHDL-Simulation waveform plot and ISS waveform plot are identical.
- **Synthesis and Place & Route** In this design phase we calculate die size or FPGA cells and the achievable maximum clock frequency of the design. If the design does not fulfill the requirements, further FU optimization or a complete hardware/software redesign has to be done.

The ISS (generated with the LISA description generated by GenCore) is embedded into a wrapper written in C-language. The compiled executable of the wrapper is executed from the octave testbench and fed by a Octave matrix. If the ISS has finished the program, the results are loaded into the testbench for verification.

Similar methods are used for the VHDL-simulation and the FPGA-prototype. The memory-content-hexfile is loaded either into the VHDL-Simulation or with the Debug-Client into the FPGA-prototype.

2.2 Debug Interface

GenCore also generates a unified debug bus for every DSP design. Memories and register files can be accessed through this bus. The primary use of the debug bus is to load the program memory with the firmware. For functional tests of the DSP FPGA Implementation, the debug bus is used to make memory dumps. To verify, that the VHDL-synthesis and the place and route was successful, the instruction set test programs are run on the FPGA and on the ISS; the octave testbench validates the FPGA design by memory dump comparison.

In addition to the functional tests on the FPGA, the debug interface helps to test the communication with environmental devices connected to the FPGA-prototype. For example: we make memory dumps of an analog to digital converted data stream with the FPGA-prototype. The memory dumps are transferred into the octave testbench to design the synchronization algorithm with real data, offline.

The FPGA-prototype helps to speed up software algorithm tests. Software implemented into an FPGA-prototype runs at almost the same speed as a later silicon-version of the DSP.

Implementation Since different FPGA development boards are accessed on different ways, we have developed for each board a wrapper on the Server-Site and on the Client-Site. This wrappers are used to tunnel the debug bus through different transmission medias like serial link, USB and PCI.

We have tested this approach with a Gidel PCI FPGA Board and two Altera FPGA development boards. Two different wrappers were developed: A PCI-BUS wrapper for the Gidel PCI Board and a USB 2.0 interface wrapper including the essential software for the Altera FPGA development boards.

3. OFDM-DSP Core

The developed DSP is a fixed point SIMD parallel signal processor with a high degree of instruction level parallelism (A block diagram is shown in figure 2). The DSP consists of 4 parallel data paths (Vector Data Paths), one scalar data path, two interconnection units and three single port memories: a program Memory, a scalar memory and a vector memory.

3.1 Vector Data Path

Each vector data path consists of six functional units (FU). Four of them are common known DSP FUs [6]: a ALU, a multiply and accumulate (MAC) unit, a barrel-shifter (BS) unit, and an 8 entry register (REG) file. We added a conditional unit (COND) and a conditional data transfer (IF)-unit for optimized data flow control. The bitwidth of all interconnections is 16 bits, except for a 32 bit connection between the accumulator of the MAC unit and the barrel shifter unit which supports increased computational accuracy. The IF and COND unit enable data flow control by executing the basic block

```
for (i=0;i<S;i++)
{
        if (ai {>,=,<} bi) xi = yi;
        else xi = zi;
}
```

in two processor cycles. Where S is the number of parallel data-paths (S equals 4 in the presented design).

3.2 Scalar Data Path

The scalar data path is primarily involved with address calculations and program control. It consists of two ALUs, a multiplier (MUL)-unit, a REG-unit, a BS-unit and a sequencer (SEQ). In the scalar data path we have connected a COND unit to the sequencer for conditional branches. Similar to the Vector Data Path the bitwidth of all interconnections is 16 bits.

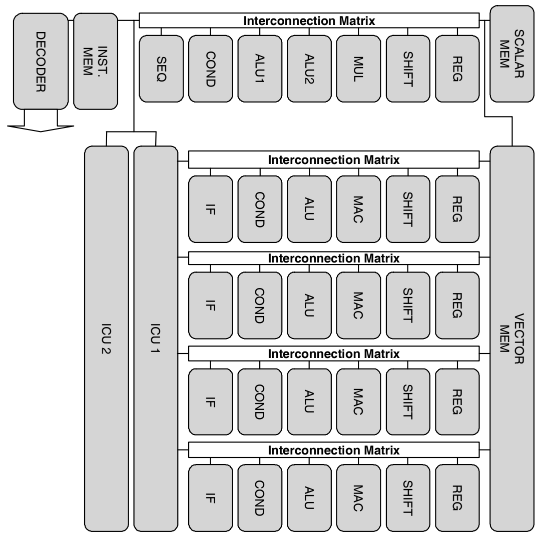

Figure 2. OFDM-DSP Block Set

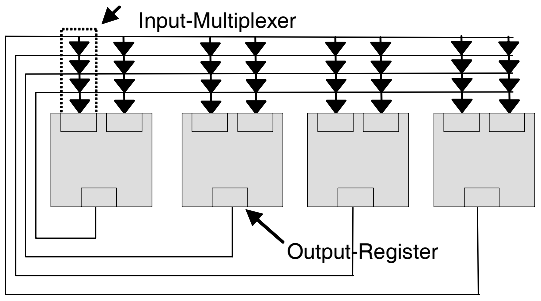

Figure 3. Complete interconnection matrix

3.3 Interconnection Matrix

The FUs are connected by means of an Interconnection matrix. In our case of study this interconnection matrix is complete. Hence, all output registers of the FUs are connected to all input multiplexer of other FUs with the same bitwidth. The disadvantage of such a design is the high number of connection wires. For high clockrates, low power and small die size, the number wires can be taylorized after algorithm profiling. An example for a typical interconnection matrix is shown in figure 3.

3.4 Interconnection Unit

The data transfers between vector data paths are accomplished via the interconnection units (ICU). We have implemented the basic ICU instructions: Zurich-ZIP, Select and Broadcast (Figure 4a, Figure 4b and Figure 4c). Permutations can be achieved with two Dresden Rotate instructions.

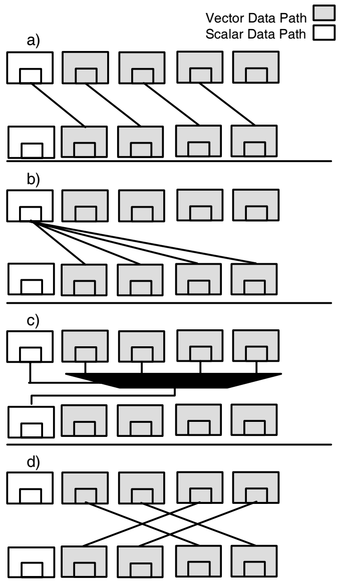

Figure 4. Interconnection Unit Instructions (a) Zurich Zip (b) Broadcast (c) Select (d) Dresden Rotate

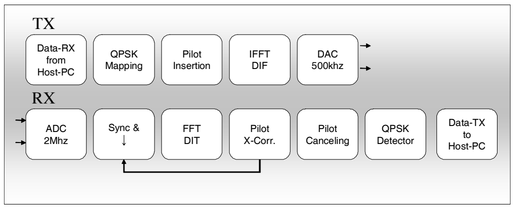

Figure 5. OFDM Broadcasting Transmitter and Receiver

An example for Dresden Rotate “2” is shown in figure 4d. Combined with the IF-unit which is controlled by a scalar value to setup the input multiplexer manually, every permutation is possible. The first ICU is shifting the data to the right and the second to the left.

4 OFDM-Transceiver

Software was developed for OFDM-based transmitter and receiver [10]. We implemented functional blocks similar to those being used in DVB-T/H broadcasting systems. Each functional block at the transmitter has a corresponding block at the receiver as shown in figure 5. After insertion of BPSK pilot carriers (Barker Sequence) the IFFT (with decimation in frequency) is calculated. The resulting time domain signal is finally converted into an analogue signal using a dual DAC with 500kHz sampling frequency. At the receiver the incoming symbols are oversampled at 2MHz with a dual ADC. Depending on the cross-correlation results after the FFT, the phase of the incoming stream is shifted by the synchronisation unit. Every forth symbol is stored into the memory. After calculation of the FFT, the BPSK pilot signals are cross-correlated with the barker sequence and the complex result is stored for later use in the synchronisation unit. [8]

In table 1 and 2 the Clock Cycles for each algorithm are presented. Mapping, Detecting, Pilot Insertion and Removal take only a little percentage of the overall computational effort. The main parts are IFFT/FFT and Memory Transfers.

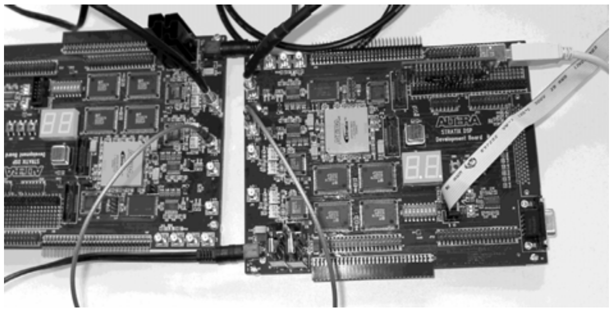

Figure 6. Development Board Setup

Algorithm	Clock Cycle Count
Memory Transfers	450
Mapper	141
Pilot Insertion	116
IFFT	2273

Table 1. Clock Cycles for the OFMD-Transmitter

Algorithm	Clock Cycle Count
Memory Transfers	438
FFT	2615
Cross-Correlation	30
Pilot Remove	80
Detector	115

Table 2. Clock Cycles for the OFMD-Receiver

5 Conclusions and Future Work

We have presented a hardware/software co-design platform for our SIMD-DSPs cores. With this platform a OFDM-tailored DSP with software for a DVB-T/H like OFDM receiver and transmitter was developed. A proof of concept was done with a FPGA-implementation of receiver and transmitter. DSP hardware and software design and FPGA-implementation were done within 5 weeks by two engineers.

Based on our research activities during the last years, we have developed a design exploration platform which provides access to main chip design parameters beginning from early design stages.

Moreover, architecture and hardware/software partitioning failures are correctable with low change cost in a short time.

Future work Power Analysis is not being implemented into the platform yet. To meet future design threads of handheld terminals and mobile devices, this has to be done soon. Compiler frontend and backend have to be improved continously and the integration into the design platform has to be tightened.

6 Acknowledgements

The authors are grateful to the MPEG-4 team of the Heinrich-Hertz-Institut (Fraunhofer Gesellschaft) for providing MPEG-4 broadcasting and visualisation tools.

References

[1] G. Cichon, P. Robelly, and H. Seidel. *STA Core Generator Manual*. TU-Dresden, 2003. www.radionetworkprocessor.com.

[2] G. Cichon, P. Robelly, H. Seidel, M. Bronzel, and G. Fettweis. Compiler scheduling for sta-processors. In *Proc. of International Conference on Parallel Computing in Electrical Engineering*, Dresden, Germany, September 2004.

[3] G. Cichon, P. Robelly, H. Seidel, M. Bronzel, and G. Fettweis. Synchronous transfer architecture (sta). In *Proc. of Fourth International Workshop on Systems, Architectures, Modeling and Simulation (SAMOS'04)*, Samos, July 2004.

[4] A. Halambi and P. Grun. Expression: A language for architecture exploration through compiler/simulator retargetability, 1999.

[5] A. Kitajima, M. Itoh, J. Sato, A. Shiomi, Y. Takeuchi, and M. Imai. Effectiveness of the asip design system peas-iii in design of pipelined processors. In *Proceedings of the 2001 conference on Asia South Pacific design automation*, pages 649–654. ACM Press, 2001.

[6] P. Lapsley, J. Bier, A. Shoham, and E. A. Lee. *DSP Processor Fundamentals*. IEEE Press, 1997.

[7] S. Pees, A. Hoffmann, V. Zivojinovic, and H. Meyr. Lisa-machine description language for cycle-accurate models of programmable dsp architectures. In *Proceedings of the 36th ACM/IEEE conference on Design automation conference*, pages 933–938, 1999.

[8] J. G. Proakis and D. G. Manolakis. *Digital Signal Processing*. Prentice Hall, third edition, 1996.

[9] P. Robelly, G. Cichon, H. Seidel, and G. Fettweis. Automatic codegeneration for simd-dsp architectures: A algebraic approach. In *Proc. of International Conference on Parallel Computing in Electrical Engineering*, Dresden, Germany, September 2004.

[10] H. Seidel, E. Matus, G. Cichon, P. Robelly, M. Bronzel, and G. Fettweis. Generated dsp cores for implementation of an ofdm communication system. In *Proc. of Fourth International Workshop on Systems, Architectures, Modeling and Simulation (SAMOS'04)*, Samos, July 2004.

[11] V. Zivojnovic, S. Pees, and H. Meyr. Lisa - machine description language and generic machine model for hw/sw co-design, 1996.