ASIP Architecture for Future Wireless Systems: Flexibility and Customization

Joseph R. Cavallaro, Predrag Radosavljevic
Department of Electrical and Computer Engineering
Rice University, MS-366 - 6100 Main Street
Houston, Texas 77005
Email: {cavallar, rpredrag}@rice.edu

I. Introduction

Processors for mobile handsets in 3G cellular systems [1] require: high speed, flexibility and low power dissipation. In addition, computationally very demanding algorithms are needed to remove high levels of multiuser interference especially in MIMO case. Traditional architecture solutions are ASIC and DSP processors. While computationally efficient, ASIC processors are often not flexible enough to support variations of implemented wireless algorithms. On the other hand, DSP processors, although flexible and fully programmable, often cannot achieve high performance with low power dissipation (limited level of instruction and data parallelism). Because of that, there is a recent interest in new flexible architectures with limited programmability [2] and possibility for customization that is targeted to a class of wireless applications with high levels of data and instruction parallelism. These architectures are called Application Specific Instruction set Processors (ASIPs) and can replace multiple chip designs implemented as an ASIC architecture [3].

The ultimate goal of designing flexible ASIPs for implementation of physical layer of cellular applications is the most optimal architecture solution in terms of power dissipation and area that can meet high-demanding real-time requirements defined by 3GPP standards ([1] and [4]) with reasonable clock frequency. The frequency range of up to 200MHz is proven to be a good tradeoff between clock speed and power dissipation. We propose the implementation of base-band wireless algorithms on the ASIP processors based on Transport Triggered Architecture (TTA) [5]. It can be shown that the application specific processor based on TTA is programmable and flexible enough in order to handle different modifications of wireless application, operate in broad range of channel environments defined by 3GPP standard (from two-paths Pedestrian A to five-paths Vehicular A channels, speed of mobile subscriber varies from 3km/h to 120 km/h) and can achieve real-time requirements in high data rate downlink with a frequency of up to 200MHz and low power dissipation.

TTA software tools [6] are integral part of TTA design package and they are used to search for TTA processor with the most favorable cost/performance ratio. The integration of application-specific user-defined instructions and implementation of corresponding Special Function Units (SFUs) is achieved by modifying (recompiling) the existing TTA software tools. By implementing set of specialized operations some level of architecture flexibility is traded-off for smaller area occupation, lower dynamic power dissipation, and faster execution. VHDL representation of the processor core can be obtained by using processor generator software tool [6]. Generated processor cores together with the memory peripherals are synthesized for FPGA prototype implementation using Xilinx ISE Foundation synthesis tool [7] and also for CMOS gate-level analysis using Mentor Graphics [8] software tools. This hardware-software design flow, that is unique feature of ASIP architectures, provides fast and an efficient way to generate flexible hardware module directly from the High Level Language (HLL) description of the wireless application.

II. ASIP Architecture for Implementation of Wireless Applications

Efficiency and flexibility are crucial features of the processors in the next generation of wireless cellular systems. Processors need to be efficient in order to satisfy real-time requirements for very demanding algorithms in new emerging wireless standards (3GPP, 4G, 802.11x, WiFi, DVD-S2, DAB, just to name a few). Flexibility, on the other hand, allows design modifications to respond to the evolution of standards (from GPRS to 3G [9], for example), world-wide compatibility (UMTS in Europe and Asia, CDMA2000 in North America), changes of user requirements depending of the quality of service (QoS), etc. Often, efficiency and flexibility goals are conflicting. Efficiency is related to the more custom hardware implementation such as ASIC processors. On the other hand, flexibility is the basic feature of programmable platforms such as DSP processors.

While computationally efficient and low power solutions, ASIC processors for wireless applications [10] are often not flexible enough to support necessary variations of implemented algorithms. ASIC design, especially in deep submicron technologies, is very complex task and the manufacturing costs are also high [3]. It is cheaper to write and debug software (application written in high level languages) than directly design, debug and manufacture a hardware. Furthermore, there are increasing demands for products with low time-to-market which is not primary characteristic of the ASIC design. On the other hand, DSP

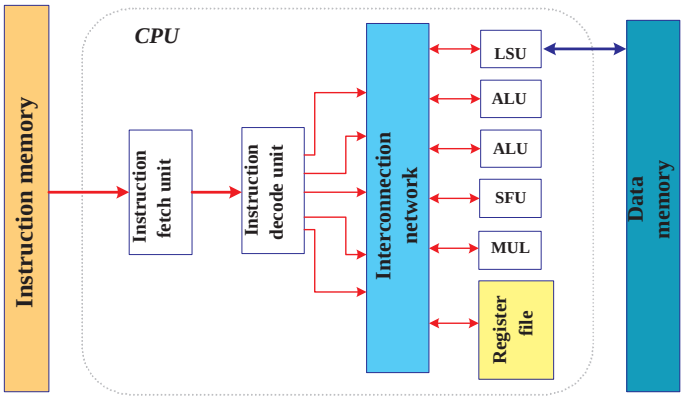

Fig. 1. General structure of TTA architecture

processor, although fully programmable, cannot achieve high performance with low power dissipation. DSP cores are often not able to achieve high level of instruction and data parallelism required for future generations of wireless systems.

New revolutionary architecture solution named ASIP combines the efficiency of ASICs and flexibility of DSPs and makes a huge impact in the processor design methodology. ASIPs are heterogenous platforms composed of programmable processor core and customized hardware modules [11] that allow designer to extend the instruction set with application-specific operations. In today market there is a large variety of customized ASIP processors such as Tensilica's Xtensa [12], dynamically reconfigurable Chameleon processor [13], Transport Triggered Architecture [5], etc. Supporting design tools allow fast and efficient processor design whose duration is comparable to the purely programmable DSP solutions. Fixing of errors and changing requirements at low cost are also great benefits of ASIP architecture design flow.

In order to illustrate the benefit of such architecture we propose as a generic example the implementation of channel equalization in presence of multiple transmit and receive antennas. Channel equalization is one of the most challenging base-band wireless algorithms [14] for 3GPP MIMO (Multiple Input Multiple Output) downlink transmission. This algorithm is implemented on the ASIPs based on TTA [5]. TTA processors exploit both instruction and data level parallelism. By implementing user-defined SFUs the sub-word parallelism can be exploited as well. The architecture is flexible and new function units, buses, and registers can be added without any restrictions. In addition, application specific support is provided by implementation of specialized user-defined function units customized for specific application. Available register files and memory banks play important role in parallel and efficient data flow. The main advantages of the ASIP architecture based on TTA are: flexibility (same architecture can be used for different algorithms, i.e modifications of channel equalization algorithms for different channel environments), speed (short processor cycle time), and, in addition, fast and application specific processor design [6]. Some limitations of the TTA also exist - the number of buses is often large especially if there is a need for high level of data parallelism, and consequently the instruction word can be long.

A. ASIPs Based on the Transport Triggered Architecture

TTA [5] is a superclass of Very Large Instruction Word (VLIW) architectures. A common approach to increase computational efficiency is to increase the number of function units that operates in parallel. This idea is implemented in traditional VLIW processors where multiple instructions are executed in parallel. In VLIW processors the ILP (instruction level parallelism) is significantly exploited. However, the data path of VLIW processors is very complex especially when they are designed for large amount of parallelism that is required in today's and future cellular applications.

In order to execute sequential HLL code of application on a VLIW processor the following steps need to be performed by compiler: frontend compilation, determination of dependencies, register allocation, scheduling and binding the operations to FUs. Binding of data transports to buses is the final remaining step and it is the responsibility of the VLIW hardware. One question that arises is whether it is possible to perform the last step in the software as well. TTA processors expanded the compiler characteristics in order to perform all of these tasks in software. As a result, TTA compiler is more complex but processor core is architecturally much simpler than traditional VLIW solutions. For example, data bypassing is scheduled by the compiler and not by bypass hardware units and consequently the bypass network is significantly simplified.

The general structure of the TTA processor is shown in the Figure 1. The interconnection network is composed of transport buses that moves data between function units (FUs) and between FUs and Register Files (RFs). FUs are connected to the buses via input and output sockets that are implemented as multiplexers that transfer data to/from the appropriate hardware unit. Load/store units (LSUs) represent the interface between processor core and data memory. TTA is 32-bit wide architecture solution - all buses, operands, and input/output FU/RF ports are 32 bits wide.

TTA processors have a large flexibility. The interconnection network is separated from the FUs and it can be designed independently. On the other hand, in VLIW processors, every time when new FU is added the interconnection network needs to be changed and new register file ports included. There are also extra scheduling freedom in the case when the data transports are directly scheduled instead of operations. For example, there is no need to move multiple source operands in the same cycle. In this case the operands can stay longer in the FUs that reduces the number of required registers. Also, it can be shown [15] that the splitting of FUs into multiple independent components (different arithmetic and logical operations doesn't need to be combined in the single ALU) results in significant speedup. In traditional VLIW processors this splitting is hard to implement since it increases complexity of interconnection network and register files. In addition, the ASIP architecture based on TTA can be customized for a given application by extending the instruction set with the user-defined application-specific operations that are implemented in hardware as SFUs (Special Function Units).

All of these properties make TTA processors very promising solutions for design of high performance ASIPs in future wireless systems and better solutions than traditional VLIW architectures. In addition, hardware-software co-design flow is very efficient and flexible and allows fast automatic generation of TTA processors for application written in C/C++ program.

B. TTA Design Flow

TTA design flow is presented on Figure 2. It consists of a retargetable frontend compiler, a back-end software tools and simulators. The frontend compiler is based on the gcc-move compiler. It compiles C/C++ application code and produces as an output sequential 'move' code that can be verified with sequential simulator. The simulator also provides profiling of data that is used by the back-end tools for better scheduling of the sequential code. The back-end software tools (scheduler, estimator and explorer) read the sequential 'move' code and profiled data, produce parallel (scheduled) code and run the exhaustive design space exploration in order to find the most optimal architecture solution. The accuracy of the parallel code and its execution on the target processor is verified by the parallel code simulator. The back-end consists of several stand-alone tools [16] such as: scheduler, estimator, and explorer.

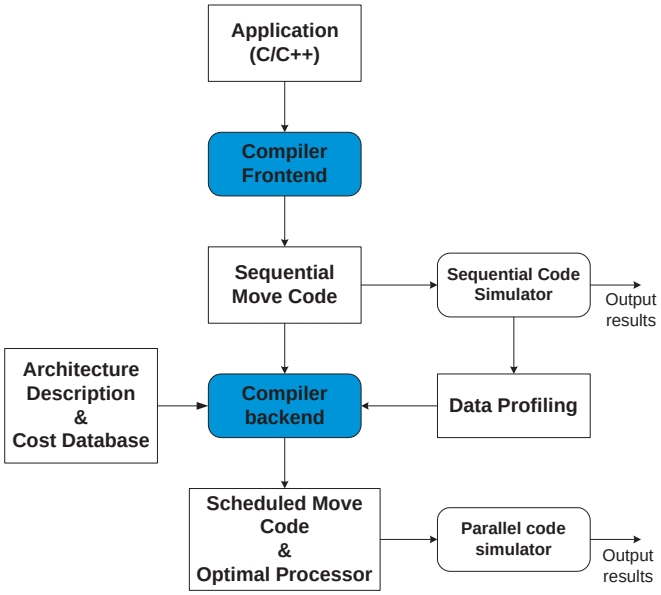

Fig. 2. TTA design flow

C. Design of TTA Processors for Channel Equalization with Special Function Units

As it is already said, design of TTA equalizer architecture can be efficiently customized for specific application by designing several Special Function Units (SFUs) while keeping sufficient generality in order to reuse the same architecture for different modifications of equalization algorithms in MIMO WCDMA downlink. Set of instructions is extended by implementation of user-defined application-specific operations that are executed on SFUs. TTA front-end and back-end software tools (compiler, scheduler and explorer) need to be modified in order to be able to schedule the execution of these operations, evaluate performance of user-defined special function units (estimation of power dissipation, area and latency) and search for the most optimal processor. Design space exploration (searching for the optimal processor) is based on the cost-database that contains estimates of all hardware components (including user-defined SFUs) for dynamic power dissipation and area occupation based on the 0.13 micron CMOS technology.

Complex multiplication (CXMUL in Figure 3) with shifting (normalization) is one of the designed SFUs. Pre-packing of two 16-bit data (real and imaginary part of received data samples) into 32-bit number enables that SFUs for complex arithmetic have only two input ports. Data are unpacked inside the CXMUL, complex arithmetic on four 16-bit numbers is performed and two results are packed back in 32-bit number (only one output port).

SFU for arithmetic operations with sub-word parallelism is also implemented. Since one 32-bit operand represents two 16-bit numbers it is worth wile to design function unit for parallel arithmetic operations on 16-bit portions (sub-words). Additional functionality of this Special Arithmetic Logic Unit (SALU in Figure 3) is sign-test and sub-word add/subtract operation that is a frequent operation in channel estimation algorithm.

The third kind of SFU, that is mainly utilized in updating of filter coefficients using Conjugate Gradient(CG) algorithm, is real multiplications with right shifting by various number of bits. This flexible operation is crucial for the stability of fixed point implementation of CG algorithm.

We show in [14] that by exploiting the custom nature of ASIP architectures the internal structure is significantly simplified especially the interconnection network between FUs. This issue is a major concern for area and power dissipation. It is proven that the performance results of customized ASIP architecture (execution time, dynamic power dissipation, and area) are close to ASIC solution. Furthermore, programmability of this architecture design allows channel equalization in MIMO WCDMA downlink of broad range of scattering environments. In Figure 3, the optimized architecture configuration for CG filter update (full equalization algorithm except filtering+despreading/descrambling) is presented.

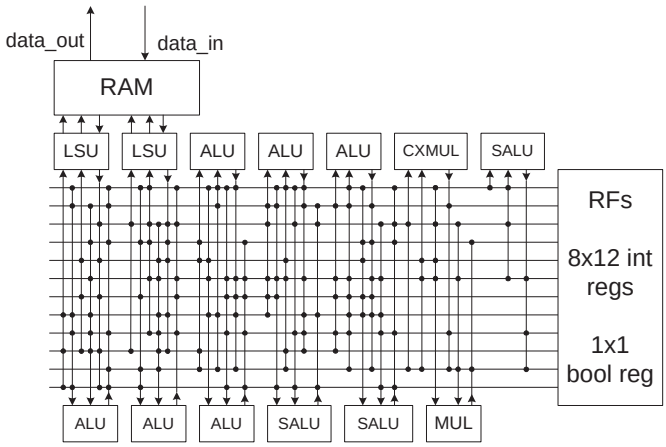

Fig. 3. Optimized CPU architecture: Co-processor for CG filter update

We propose two possible approaches for implementation of channel equalization algorithms on ASIP architectures [14]: single processor solution and the architecture with two interfaced co-processors. While the single processor solution processes all sub-parts of channel equalization algorithm, the architecture solution with two interfaced co-processors splits total workload into two parts. One co-processor is customized for the estimation of second order statistics (estimation of multiple channels and computation of receive covariance matrix) and iterative computation of equalizer's coefficients. The other co-processor is designed for filtering and user detection. Although the filtering part of the algorithm is computationally more complex than the filter-update part (see Figure 4), the filtering co-processor is less complex because of the uniform nature of implemented algorithm.

It can be observed from Figure 4 that the computational complexity strongly depends on the scattering of channel environment (number of multipaths) and on the speed of mobile subscriber. The computational complexity (number of required operations in one second) of the full channel equalization algorithm is significantly higher than the typical operation count for commonly used C5x TI DSP processor [17]. In addition, such a complex algorithm needs to be executed very fast in order to support high 3GPP data rate.

It is shown [14] that the single processor solution based on TTA can achieve 3GPP real-time requirements [1] with minimum clock frequency ranging from 78 MHz for two-paths Pedestrian A channel to 155 MHz for five-paths Vehicular A environment. The maximum dynamic power dissipation (obtained by using 0.13 micron technology TTA power model) is 78 mW for the computationally most complex case. The area is approximately 160,000 gates. All three kinds of special function units are designed.

Customized TTA co-processor for channel estimation/covariance matrix computation/filter update has area of about 140,000 gates, maximum dynamic power dissipation of 54 mW and the minimum clock frequency in order to achieve real-time requirements is ranging from 42 MHz (Pedestrian A channel) to 109 MHz (Vehicular A environment). Co-processor for

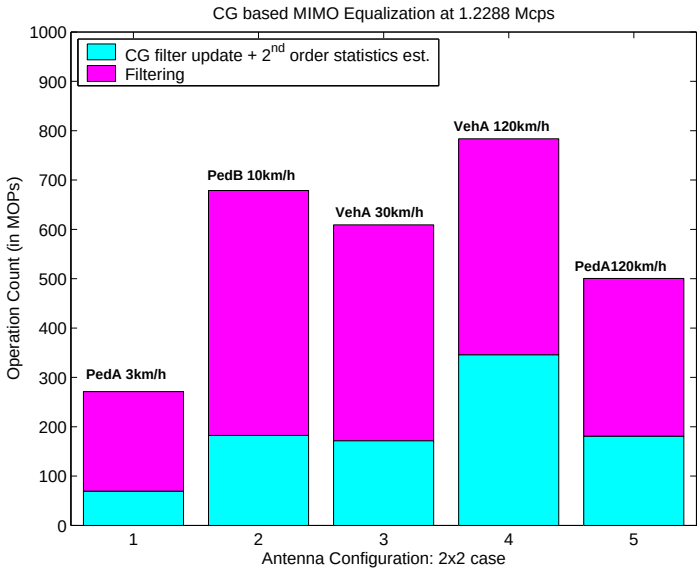

Fig. 4. CG equalization, two transmit and two receive antennas: Operation count per second for processing one chip - contributions of different parts of equalization

filtering/user detection is fairly simple because of the uniform nature of the algorithm. It has about 80,000 gates, maximum power dissipation of 38 mW and the minimum clock frequency is ranging from 45 MHz to 125 MHz.

III. Hardware Implementation of ASIPs Based on TTA

In this section, hardware implementation of previously customized and simulated ASIP processors based on TTA will be presented. General organization of the processor architecture (from the VHDL perspective) is first described as well as the software tools that are used for generation of VHDL representation of processor core. Synthesis results of Xilinx FPGA fast prototyping of designed processors are presented as well as the gate level design. We show that by using several different software tools we are able on fast and efficient way to design hardware implementation of target ASIP processor. The entire hardware-software co-design flow is presented on Figure 5.

In general, hardware/software co-design [18] is well known strategy for fast and flexible ASIP hardware implementation of applications described by HLL (C/C++ program) while efficiently avoiding design errors and decreasing design costs and time-to-market. In our case, hardware design starts with HLL description of the channel equalization algorithm. By using TTA software design tools and the library of traditional and special function units we are able to generate efficient ASIP processors with low cost/performance ratio. Next step is the conversion of processor description file into VHDL representation of the processor core by using MOVEGen software tool [19]. Automatically generated VHDL code of the processor core together with VHDL code of predesigned components (memories and peripherals) can be directly used by Xilinx synthesis tools for fast FPGA prototyping. Same VHDL design can be used by Mentor Graphics tools such as Leonardo Spectrum [20] and IC Station [21] in order to obtain gate-level design (layout) of the target processors. Consequently, CMOS libraries have to be also included in the this design flow. The entire hardware-software co-design flow (from HLL code of channel equalization to processor layout) is presented on Figure 5.

A. MOVEGen Design Flow

VHDL representation of the ASIP processors based on TTA is obtained by using processor generator design flow where MOVEGen software tool [19] is a central part. Processor generator design flow is shown in Figure 6. Processor generator is basically a set of software tools that performs transformation from one hardware description language to another.

Starting point of the design flow is machine description file of the optimal target processor and file that configures interface between load/store units and data memory as well as the external memory interface. MOVEGen software tool generates VHDL description of the processor core including control register and the interconnection network. Processor core together with memory peripherals, and the library of pre-designed and user-defined FUs represents target TTA processor.

B. VHDL Processor Representation

General VHDL template and the external interface of the processor core is shown in Figure 7. Structure of the TTA processor can be divided into two sub-parts: processor core and the peripherals. The processor core consists of three main

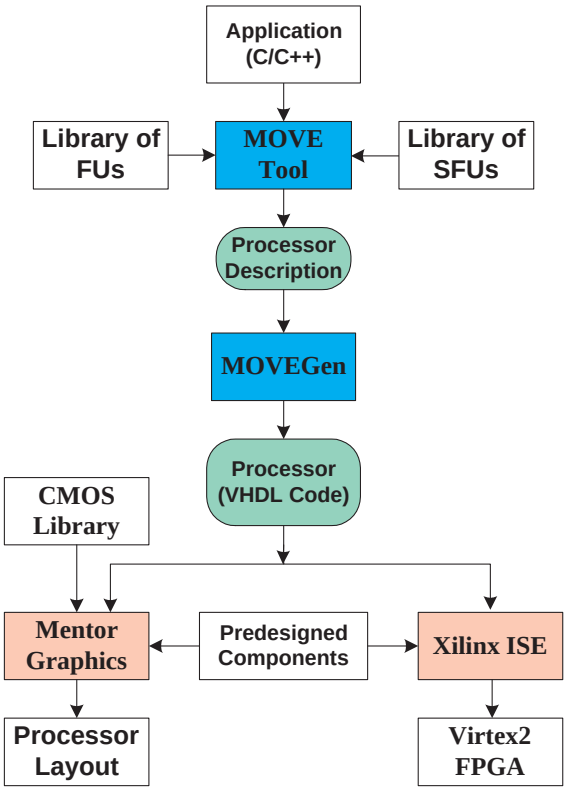

Fig. 5. Design flow from HLL code to hardware implementation

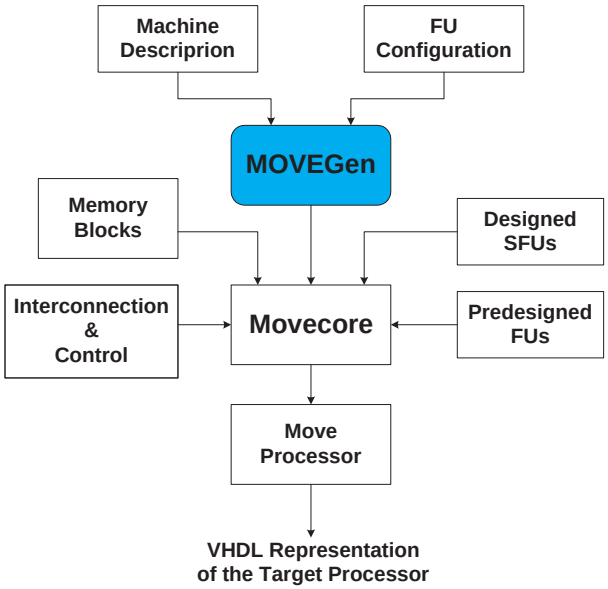

Fig. 6. MOVEGen design flow

parts: interconnection network, control register and function units (including user-defined special function units and register files). VHDL representation of processor core as well as VHDL representation of interconnection network and control register is generated automatically from the machine description of the target processor. The library of pre-designed components such as function units, special function units and general purpose registers needs to included in the processor design. Peripheral components are memory components such as instruction memory (ROM memory) and data memory (single or dual port RAM

memory) as well as the memory arbiter. VHDL representation of all peripheral components are originally pre-designed.

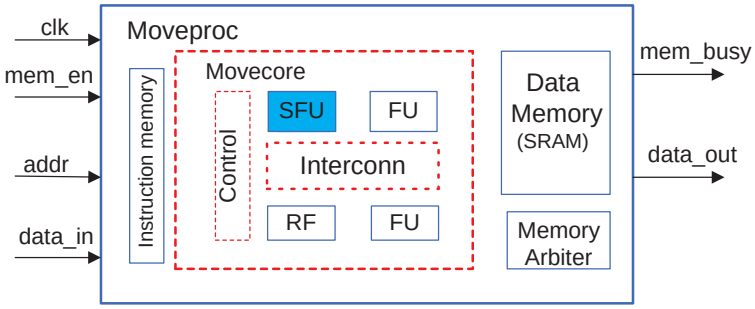

Fig. 7. General VHDL template of Move processor

It is assumed that the processor core employs one or two predesigned load/store units. In the case of two load/store units that access the same address space, dual-ported RAM memory needs to be used for data storage. The instruction memory of the processor is implemented using a ROM lookup-table (table of program instructions). Move instruction is VLIW instruction that is composed of bus-fields (every bus in the processor architecture has its own field). Each bus-field specifies one $move$ operation from source to destination whose addresses are encoded into bus-fields. The control register is used to fetch and decode move instructions.

C. Hardware Synthesis Based on VHDL Processor Representation

Hardware implementation of the target TTA processors can be achieved by using commercially available synthesis tools such as Xilinx ISE Foundation (XST synthesizer) [7] and Leonardo Spectrum from Mentor Graphics [20]. For the purpose of fast prototype design and testing, proposed TTA processors are synthesized for Xilinx FPGA boards. On the other hand, Leonardo Spectrum from Mentor Graphics contains CMOS libraries and consequently processors represented with VHDL code can be synthesized for a fixed layout implementation.

Before synthesis for Xilinx FPGA boards several adjustments needs to be performed. Since function units are originally pre-designed by using Synopsis DesignWare components [22], the appropriate libraries from Synopsys needs to be included in the Xilinx ISE Foundation synthesis tool. In addition, pre-designed RAM memory is not appropriate for Xilinx FPGA design since Xilinx synthesis tool (XST synthesizer) cannot recognize it as a RAM memory and does not utilize on-board RAM blocks. In order to prevent this and save large number of FPGA slices for processor core, data memory is implemented by using the Xilinx XCore Generator tool.

Xilinx ISE Foundation software tools [7] are used for synthesis of channel equalizer (full equalization algorithm) with special function units. Virtex2 FPGA board with 6 millions gates is used. The processor consists of 16 buses, 6 ALUs, 3 special ALUs (sub-word parallelism and sign-test operations), 2 complex multiplier (with shifting ability), 1 real multiplier (with shifting ability) and 4 load/store units that are interfaced with dual-ported RAM memory (two memory blocks). Synthesis statistics has the following estimates:

- Used FPGA board: XC2V6000
- Number of used Slices: 21,126 out of 33,792
- Number of used BRAMS: 107 out of 144
- Number of used IOBs: 229 out of 1104
- Number of MULT18x18s: 11 out of 144

The gate level design of the proposed ASIP processors can be obtained by using Mentor Graphics tools, in particular Leonardo Spectrum synthesis tool and CMOS libraries. The ASIP processor for full CG/LMS equalization with special function units is synthesized by using ASIC library for 0.5 micron CMOS technology. The synthesis results show that the area of the processor core (processor without peripherals) is 182,887 gates which is close to the area given by the TTA software simulator (see results in section II-C).

In this section we showed efficient automatic design flow for gate level synthesis of the target ASIP processor. The design flow starts with the optimized HLL code (16-bit fixed point C code) of a given application and initial processor configuration. Final result is layout design of the optimized processor that executes the specific application. Hardware design flow is composed of the chain of software tools from different vendors: MOVEGen processor generator, Xilinx synthesis tools and/or Mentor Graphics synthesis tools.

IV. Conclusion

In this work we propose the use of ASIP architecture based on TTA for implementation of future wireless applications. This programmable architecture has a possibility of customization and therefore it can achieve same efficiency as ASIC while keeping the large flexibility. In addition, we presented fast and efficient design flow for ASIP hardware implementation that starts with fixed-point C/C++ code of the application and finishes with the gate-level implementation of the target processor. Several design tools from different vendors are combined together in order to achieve the semi-automatic hardware design of ASIP processors starting from HLL description of the application. We strongly believe that this architecture class is a very strong candidate for handsets processors in next generations of mobile wireless systems.

V. Acknowledgements

This work was supported in part by Nokia Corporation, Texas Instruments, Inc., and by NSF under grants EIA-0224458, and EIA-0321266.

References

[1] "1xEV-DV Evaluation Methodology (Rev.26)." Third Generation Partnership Project Two (3GPP2), May 2001.
[2] F. Sun, S. Ravi, A. Raghunathan, and N. K. Jha, "Custom-Instruction Synthesis for Extensible-Processor Platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, pp. 216–228, February 2004.
[3] K. Keutzer, S. Malik, and A. R. Newton, "From ASIC to ASIP: The Next Design Discontinuity," in *IEEE International Conference on Computer Design*, pp. 84–90, September 2002.
[4] J. Fonollosa, R. Gaspa, X. Mestre, A. Pages, M. Heikkila, J. Kermoal, L. Schumacher, A. Pollard, and J. Ylitalo, "The IST METRA project," *IEEE Communications Magazine*, vol. 40, pp. 78–86, July 2002.
[5] H. Corporaal, *Microprocessor architecture from VLIW to TTA*. John Wiley and Sons, 1997.
[6] J. Heikkinen, J. Sertamo, T. Rautiainen, and J. Takala, "Design of transport triggered architecture processor for discrete cosine transform," in *15th Annual IEEE International ASIC/SOC Conference*, Sept. 2002.
[7] "Xilinx ISE Foundation [Online]." Available: http://www.xilinx.com/products.
[8] "Mentor Graphics [Online]." Available: http://www.mentor.com.
[9] J. Mitola, "Software radios: Survey, critical evaluation and future directions," *IEEE Aerospace and Electronic Systems Magazine*, vol. 8, pp. 25–36, April 1993.
[10] A. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. Brodersen, "Optimizing power using transformations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, pp. 12–31, January 1995.
[11] A. Wang, E. Killian, D. Maydan, and C. Rowen, "Hardware/software instruction set configurability for system on-chip processors," in *IEEE/ACM Design Automation Conference*, pp. 184–188, June 2001.
[12] "Xtensa Micsroprocessor [Online]." Available: http://www.tensilica.com.
[13] "CS2000 Reconfigurable Communications Processor Product BRIEF." Chameleon Systems Inc. San Jose, CA.
[14] P. Radosavljevic , "Channel Equalization Algorithms for MIMO Downlink and ASIP Architectures," Master's thesis, Rice University, May 2004.
[15] H. Corporaal, "ILP architectures: trading hardware for software complexity," in *IEEE International Conference on Algorithms and Architectures for Parallel Processing*, pp. 141–154, December 1997.
[16] "The Move Framework - User's Manual." Tampere University of Technology, March 2003.
[17] "TMS320VC5501 Fixed Point Digital Signal Processor." Texas Instruments, December 2002. Data Manual.
[18] W. Wolf, "A decade of hardware/software codesign," *IEEE Computer*, vol. 36, pp. 38–43, April 2003.
[19] "MOVEGen User's Manual." Tampere University of Technology, January 2004.
[20] "Leonardo Synthesis [Online]." Available: http://www.mentor.com/ leonardospectrum.
[21] "IC Station [Online]." Available: http://www.mentor.com/cicd/icstation.html.
[22] "Synopsys Designware Libraries [Online]." Available: http://www.synopsys.com/ products/designware/dwlibrary.html#blockip.