

# Implementation of an HSDPA Receiver with a Customized Vector Processor

Kim Rounioja, Kimmo Puusaari

Nokia, Technology Platforms  
Oulu, Finland

{Kim.Rounioja, Kimmo.Puusaari}@nokia.com

**Abstract**— SIMD paradigm enhances cost and power efficiency of instruction set processors when the executed workload contains appropriate data level parallelism. In this paper, we investigate development of a SIMD ASIP for an HSDPA equalizer receiver for mobile handsets. The ASIP is based on a vector processor template with C-based programming interface. The template is enhanced to capture the 14.4Mbps HSDPA equalizer receiver functionality of the wireless modem. The solution provides improved flexibility compared to the conventional approach while requiring only modest design effort. The template and the design approach can be utilized also for other subsets of modem baseband.

## I. INTRODUCTION

The forthcoming wireless terminals embody an ever increasing number of radios, each radio system having its individual roadmaps with complicated new features. Consequently, the SoCs embodying these radios are significantly increasing in complexity. The conventional radio baseband design consists of standard-specific accelerators and one or more control processors, which are integrated as a SoC. The increase in complexity makes this approach very challenging. Rather, being able to employ more flexible computation elements would give advantages in terms of design effort (hardware component longevity), die area (timesharing hardware), time-to-market (reduce risk of ASIC respin). The catch here is that power consumption constraints in wireless handsets are very challenging, and increasing flexibility usually also means increasing power consumption. For instance, relying in conventional signal processors to implement the flexibility is just not feasible.

One approach to improving the flexibility of the modem baseband while maintaining power targets is utilizing application specific instruction set processors to replace the fixed function accelerators. The specialized feature set improves power efficiency when executing the targeted workload, compared to more generic instruction set processors. This is illustrated in, for instance, [9].

The SIMD paradigm is another approach that enhances cost and power efficiency of processors [1]. SIMD or vector processing is feasible provided that the targeted workload includes the appropriate data level parallelism. Baseband signal processing in wireless modems should, in principle, contain such data level parallelism.

Recently, the applicability of SIMD processors to modem signal processing has become a popular topic. For instance,

in [2] the authors describe two SIMD processors, and discuss their suitability for selected algorithms from 802.11a and UMTS-FDD. Another good treatment of the subject is provided in [10]. ASIPs that utilize SIMD in the datapath have also been discussed: [3] depicts development of an ASIP with a SIMD extension designed for video processing. [12] illustrates similar aspects using FFT as the target application.

In this paper, we study applicability of SIMD ASIPs for modem baseband. We use the ASIP toolkit from Target Compiler Technologies [11]. We first depict a generic vector processor template for modem signal processing. Then, we illustrate its usefulness by refining the design towards an HSDPA advanced receiver. The results are described and the merits of the applied approach are discussed.

## II. VECTOR-ASIP TEMPLATE

We believe that in general, design of application specific processors should be based on processor templates. This concentrates the design effort on the aspects of the design that most benefit of the customization. Our Vector-ASIP template defines both the hardware architecture and the programming interface.

### A. General Hardware Architecture

The template is a SIMD architecture that is applicable for general purpose vector processing. It is scalable and customizable, thus making it ideal for further development. Scalability means that we have parameterized the vector register width so that it can be 64, 128 or 256 bits. The functional units scale respectively. By customizable we mean

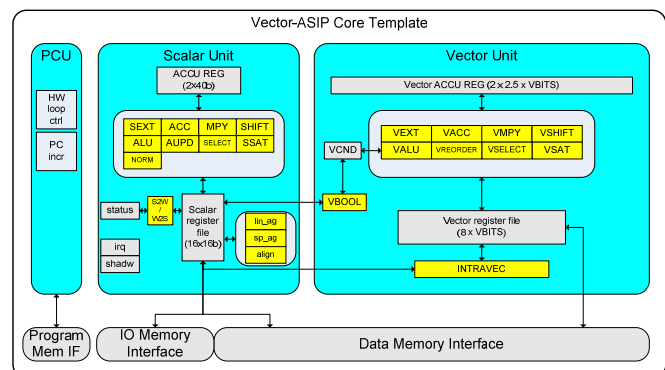


Fig 1. Vector-ASIP Template

that adding functional units and/or resources on top of the template is straightforward using the normal tool flow.

The template architecture is divided to three main instances (Fig 1). The controller (PCU) is mainly responsible for fetching the 24-bit single issue instructions from the program memory, and decoding them. The scalar data path is a basic 16-bit DSP processor responsible for calculations on scalar data including, e.g., address generation. The vector data path is an extension that is beneficial in vectorizable number crunching applications. The vector unit has an operation set analogous to that of the scalar unit, enhanced with vector-specific additions such as intra-vector reordering. There is also a unit for transferring data between the scalar and vector units.

The processor pipeline constitutes of six stages, two of which are execution stages. The scalar unit has sixteen 16-bit registers and two accumulator registers. The vector unit has eight vector registers and two accumulator registers. Accumulator registers are 2.5x wider than the normal ones. There are memory interfaces for both scalar and vector data.

### B. Programming model

The retargetable compiler in the ASIP toolkit does not support automatic translation of the algorithm software into vector form. It is our belief that such vectorization is in practice only a small part of the overall software development effort and can be considered normal software module optimization work. Therefore, we believe that currently the most practical way to program vector processors is by using a programming model that requires programmer to explicitly describe the software program in vector form. This can be supported by a simple Vector-API that enables efficient use of the vector processing, while hiding most of the hardware complexity. Another benefit of defining such V-API is that it goes nicely together with the template based design approach: the programming model is consistent throughout different V-ASIP instantiations basing on the same template.

The defined V-API extends standard C by adding the necessary data types and operations for vectors and complex arithmetic. For example, *uint8* is a data type defining a vector of 8-bit integers and *vcint16* is a data type defining a vector of 16-bit complex vectors. The number of scalar elements in a vector variable depends on the HW vector width: in a case where the V-ASIP implements 128-bit vectors, there are sixteen integer elements in one *uint8* variable. On top of these, the respective accumulator types (e.g., *vcint40*) are provided. In addition to C operations such as addition, V-API also defines a set of vector operations that appear similar to library functions in C code. Depending on the characteristics of the V-ASIP instance, such operations are mapped to one or more processor instructions by the compiler.

### III. HSDPA EQUALIZER

Release 5 of 3GPP UMTS introduced HSDPA mode for high data rate packet based access for downlink, improving peak data rate and network capacity of the UMTS system. In release 6, HSDPA capabilities were extended to terminal categories supporting up to 14.4Mbps peak data rate and enhanced performance requirements based on an advanced receiver were defined [4]. A handset supporting the enhanced performance requirements must implement a receiver different from the “rake” receiver conventionally used in UMTS handsets. Fig 2 illustrates a conventional UMTS baseband receiver and the role of the new receiver.

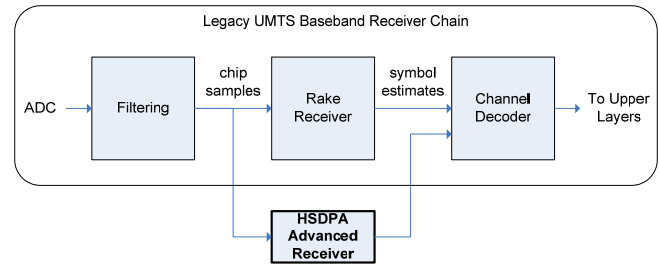


Fig 2. A conventional UMTS baseband and the new receiver for HSDPA.

The new receiver is called chip level equalizer. Chip level equalizers for CDMA handsets have been described in detail in numerous papers, such as [5–8]. Compared to the conventional rake receiver, it suppresses the multiple access interference from the received signal with a linear filter, which implements the well known MMSE solution. The filter coefficients are computed based on knowledge of the channel impulse response, which is, in turn, estimated using the pilot bits in the downlink signal. The HSDPA data symbols are recovered from the filter output by a simple despreading operation. In addition to the new receiver type, additional control channel processing is needed.

An HSDPA receiver fulfilling the enhanced performance requirements in the Release 6 standard is characterized by the functions in Table I. The given computational load figures correspond to the worst case scenario (peak data rate and space-time transmit diversity mode). The overall load is about 2GOPS in this case.

**Table I Characteristics of the HSDPA Detector Functions**

Algorithm	Arithmetic complexity [MOPS]	Characteristic Computation
Pilot correlation	338	Code generation and correlation
Channel estimate filtering	16	Pilot pattern
Equalizer coefficient calculation	36	Matrix inversion
Equalizer FIR Filtering	1291	Complex MAC
Despreading the HSDPA Channels	307	Code generation and correlation
TX diversity combining	12	Complex MAC
Control and overhead	not estimated	
Overall	2005	

We attempt to encapsulate this new receiver, its algorithm and control processing features, into a dedicated programmable module, which can be added into the legacy modem design with relatively small effort. This provides good time-to-market, while programmability reduces the risk owing to introduction of a new receiver type. It is also useful for possible future developments: we may expect that, for instance, receive diversity for the HSDPA equalizer could be supported by duplicating the device and accommodating to the necessary algorithm changes in software.

#### IV. DESIGN FLOW

Based on the MOPS estimates in Table I, we are able to identify the most computation intensive functions. By further analyzing each individual function, we can find out the characteristic data types and operations. The V-ASIP template is then customized based on the results from analysis. It is beneficial to concentrate most effort on applications with highest MOPS figure. For example, the FIR is supported by adding a parallel instruction that utilizes the pre-existing complex MAC, while at the same time updating the samples and coefficients in a single cycle. The idea with the parallel instructions is that the compiler can schedule an efficient SW pipeline using them, which greatly reduces the cycle count and improves the utilization of the processor hardware. Despreading and IRM both require efficient correlator functionality, so we added WCDMA code generators and correlator units and parallelized the operation. Correlation could also be done using 1-bit multiplications in MAC unit, but a dedicated correlator is a much more power efficient option. We also added more intra-vector reordering functionality to reduce the time spent on arranging data. For the less computationally dominant functions, we rely on the basic vector processing capabilities in the template.

Addr	Instruction	Execution Count
73	doi 7,78 /* MultiWord 2 */	80
74	/* MultiWord 1 */	80
75	VSmp10 = vset16(WM[ADa0+=1])	80
76	VSmp11 = vset16(WM[ADa0+=1])    SCRC = next_scrcode()    VCND = next_chcode(V1,V0)	80
77	VSmp10 = vset16(WM[ADa0+=1])    SCRC = next_scrcode()    VCND = next_chcode(V1,V0)    VACC1 = VACC1+corr(VSmp10,VCND,SCRC)	560
78	VSmp11 = vset16(WM[ADa0+=1])    SCRC = next_scrcode()    VCND = next_chcode(V1,V0)    VACC1 = VACC1+corr(VSmp11,VCND,SCRC)	560
79	SCRC = next_scrcode()    VCND = next_chcode(V1,V0)    VACC1 = VACC1+corr(VSmp10,VCND,SCRC)	80
80	VACC1 = VACC1+corr(V3,VCND,g12)	80

Fig 5. Example of the profile of the inner loop of vectorized and customized code.

We manually vectorized the application code using V-API. We replaced scalar data types with vector ones and modified the loop kernel to utilize the customized vector functional units via intrinsics. Intrinsics look like function calls in C, but they are directly mapped to the respective instructions by the compiler. Also we could now divide the loop trip count with the number of elements in a vector, thus reducing the cycle count by the same factor. The vectorization is performed so that a number of HSDPA channels are processed in parallel, causing no overhead to the innermost loop.

The compilation example in Fig 5 shows the profiling results for the vectorized C code of the Despreading algorithm. The excerpt is taken from the innermost loop. It shows how the compiler has efficiently SW-pipelined the loop and takes advantage of the parallel instruction. That is also where the most of the cycles are concentrated, as is visible from the profile obtained from executing the compiled routine in the generated ISS.

Having now established that the ASIP contains the features needed to process the key algorithms efficiently, we next determine the needed vector width for the vector unit, such that the worst case scenario can be executed. Exploring with the varied vector width is easy in an automatically generated instruction set simulator environment. Executing the functions in the ISS, we are able to determine the best-fit vector width for the application at hand. Fig 4 shows the processor peak load with different vector widths, given in millions of clock cycles per second. It can be seen that the target application contains the parallelism suitable for SIMD processing: increasing the HW vector size clearly reduces the processor load. Based on earlier exercises we can predict that it is feasible to obtain about 170-220 MHz clock frequency for the final processor. Therefore, the 128-bit design appears to be the best fit.

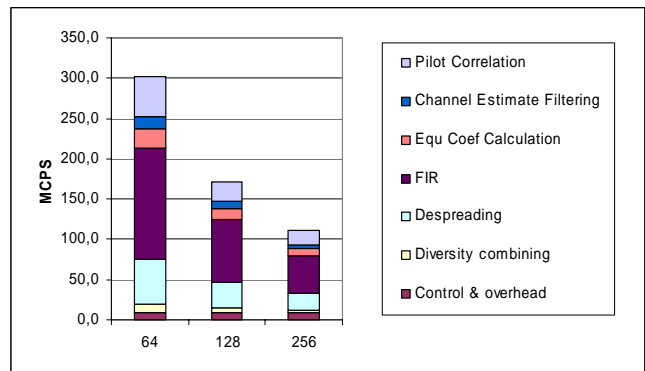


Fig 4. Measured peak load of the V-ASIP after customizations.

The VHDL for the developed processor was largely automatically generated by Target tools. Parts of the controller and the primitive operations were manually completed. We verified the functionality of V-ASIP in ISS and HDL simulators. Also gate level simulations were performed for verification and power analysis purposes.

**Table II Area of the 128-bit V-ASIP**

Unit	Gate count
V-ASIP 128-bit	252,000
Vector Unit	204,000
-Vector accumulator FU	- 29,100
-Vector shifter	- 33,200
-Vector multiplier	- 37,100
-Vector ALU	- 23,300
-Vector accum. register file	- 11,400
-Vector general register file	- 24,500
Scalar Unit	42,800
Controller (PCU)	5,200

## V. RESULTS

The VHDL code of the V-ASIP was synthesized with 90 nm technology libraries. The synthesis results can be seen in Table II. The table shows clearly that the vector unit and its function units dominate the area. The scalar unit and controller are only 19% of the total area. The critical path goes via vector accumulator FU. Using 200MHz clock frequency, we obtained a reasonable area/performance tradeoff, while satisfying the real-time performance requirements with reasonable margin (Table III).

We analyzed the power consumption based on activity measurements from gate level simulations using the HSDPA application. The typical scenario assumes no transmit diversity and 5 HSDPA code channels allocated to handset. Using a real-life application, the power analysis tool gives an accurate power estimate of 48mW for the V-ASIP core. It can be argued if 48mW typical power consumption is an acceptable price to pay for having a flexible advanced receiver for HSDPA.

While we have not implemented the same receiver functionality in other architectures, we can make limited comparisons of the developed solution to alternatives. In [2], a vector processor "EVP" for general purpose radio processing is characterized by 450kgates core area and 300MHz clock rate at 90nm process technology. Based on the presented data, we can estimate that EVP load in the filter part is lower bounded to about 80MHz, while we measured similar load for the proposed V-ASIP solution having half the vector size and also half the die area. The reason for the improved efficiency in this case is the special instruction for the 8-bit complex FIR in our solution. We may view this as one simple example of efficiency benefit due to application specificity. On the other hand, it is also clear that a fixed function architecture for the FIR filter corresponding to the same case would have only miniscule area compared to either of those two designs.

**Table III Properties of V-ASIP**

Property	Value
Clock Frequency	200 MHz
Typical Power Consumption	48 mW
Vector data path width	128 bits

## VI. CONCLUSIONS

In this paper, we described a Vector-ASIP template for modem processing. A simple Vector-API was introduced for programming the device. The template was then customized for HSDPA Equalizer receiver functionality. We illustrated that Equalizer receiver is rather well suited for vector processing, as increasing the vector length provides significant reduction in processor load at the cost of increased area. Area and power consumption estimates for the resulting ASIP were presented.

While the approach was used for only one example case in this paper, similar design flow can be used for other subsets of wireless baseband modems. In particular, SIMD processing is intuitively suitable for the future OFDM transceivers owing to the per-subcarrier signal processing characteristic of them.

## VII. REFERENCES

- [1] Kozyrakis, C.E. and Patterson D.A., "Scalable vector processors for embedded systems", IEEE Micro, Vol. 23, Issue 6, Nov.-Dec. 2003, pp. 36-45
- [2] van Berkel et. al. "Vector Processing as an Enabler for Software-Defined Radio in Handheld Devices", EURASIP Journal on Applied Signal Processing, 2005, pp. 2613-2625
- [3] Geurts, W., Goossens, G., Lanneer, D., Van Praet, J., "Design of application-specific instruction-set processors for multi-media, using a retargetable compilation flow", Proc. Of GSPx 2005
- [4] 3GPP Technical Specification 25.101 v6.11.0 "User Equipment (UE) radio transmission and reception (FDD)"
- [5] Krauss, T.P.; Zoltowski, M.D.; Leus, G.; "Simple MMSE equalizers for CDMA downlink to restore chip sequence: comparison to zero-forcing and RAKE", Acoustics, Speech, and Signal Processing, 2000. ICASSP '00. Proceedings. 2000 IEEE International Conference on Volume 5, 5-9 June 2000 Page(s):2865 - 2868 vol.5
- [6] K. Hooli, M. Juntti, M. J. Heikkilä, P. Komulainen, M. Latva-aho, and J. Lilleberg; "Chip-Level Channel Equalization in WCDMA Downlink", EURASIP Journal on Applied Signal Processing, Volume 2002, Issue 8, Pages 757-770
- [7] Sexton, T.A.; Heikkila, M.J.; Lilleberg, J.; Majonen, K.; Rounioja, K.; Ivanov, P.A.; "CDMA equalizer quantization", Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE Volume 4, 29 Nov.-3 Dec. 2004 Page(s):2318 - 2322 Vol.4
- [8] Geirhofer, S.; Mehlfuhrer, C.; Rupp, M.; "Design and real-time measurement of HSDPA equalizers" Signal Processing Advances in Wireless Communications, 2005 IEEE 6th Workshop on, 5-8 June 2005 Page(s):166 - 170
- [9] Wei, J.; Rowen, C., "Implementing low-power configurable processors - practical options and tradeoffs" Proc 42<sup>nd</sup> DAC, 13-17 June 2005, Pages: 706-711
- [10] Yuan Lin et al "SODA: A Low-power Architecture For Software Radio", Proc 33th ISCA, 17-21 June 2006, Page(s):89-101.
- [11] Target Compiler Technologies, www.retarget.com
- [12] Hoffmann, A., Fiedler, F.; Nohl, A., Parupalli, S., "A methodology and tooling enabling application specific processor design", Proc. 18th Int. Conf VLSI Design, 2005, Page(s) 399-404.