Comparison of ASIP and Standard Microprocessor 
based Navigation Processors

Götz Kappen, Lothar Kurz, Tobias G. Noll
Chair of Electrical Engineering and Computer Systems, Aachen University, Germany

BIOGRAPHY

Götz KAPPEN received the Dipl.-Ing. degree in 2002 from RWTH Aachen University. Since then he has been working as a PhD student at the Chair of Electrical Engineering and Computer Systems, RWTH Aachen University. His fields of research are satellite navigation systems and digital signal processing.

Lothar KURZ recently received the Dipl.-Ing. degree in electrical engineering from RWTH Aachen University. His research interests and the topic of the thesis, he accomplished at the Chair of Electrical Engineering and Computer Systems, are in the field of satellite navigation and digital signal processing.

Tobias G. NOLL received the Ing. (grad.) degree in Electrical Engineering from the Fachhochschule Koblenz in 1974, the Dipl.-Ing. degree in Electrical Engineering from the Technical University of Munich in 1982, and the Dr.-Ing. degree from the Ruhr-University of Bochum in 1989. From 1974 to 1976, he was with the Max-Planck-Institute of Radio Astronomy, Bonn. Since 1976 he was with the Corporate Research and Development Department of Siemens and since 1987 he headed a group of laboratories concerned with CMOS circuits for digital signal processing. In 1992, he joined the RWTH Aachen University where he is a Professor holding the Chair of Electrical Engineering and Computer Systems. His activities focus on low power deep submicron CMOS architectures, circuits and design methodologies, as well as digital signal processing for communications and medicine electronics.

1. INTRODUCTION

Application Specific Instruction Set Processors (ASIPs) are an interesting option if flexibility of a programmable architecture and lowest costs in terms of chip area and power consumption are required. Examples from communication and image processing domain show how energy and area efficiency can be significantly improved by the use of ASIPs as described in [1], [2] and [3]. In this context area and power efficiency are considered as Million Operations Per Second (MOPS) per mm² and as mW per MOPS respectively. By offering increased efficiency, ASIPs seem ideally suited for area and energy critical mobile and handheld applications.

The main idea behind ASIPs is to enhance area and energy efficiency by exploiting a priori knowledge about the processor’s target application. Or in other words, cut back the processor’s flexibility where not needed and increase area and energy efficiency. For instance, reducing the number of required processing cycles for a specified task by using special instructions can be used to slow down the processor’s clock frequency and thus decrease the dynamic power consumption. In addition to this, inefficient processor instructions can be taken out to reduce the complexity of the Arithmetic Logical Unit (ALU) and thus increase the area efficiency of the processor.

Figure 1 shows ASIPs in the context of the hardware architecture design space. Therefore, varieties of standard signal processing algorithms have been implemented on commonly used hardware architectures and are classified in terms of area and energy efficiency. It can be seen that flexible architectures such as programmable General Purpose (GP) processors or Digital Signal Processors (DSPs) offer the lowest area and energy efficiency. On the other hand Application Specific Integrated Circuits (ASICs) optimized for a specific task are very hardware efficient by drowning back the flexibility. Field Programmable Gate Arrays (FPGAs) belong to the class of reconfigurable architectures and offer a trade-off between flexibility and efficiency. However, compared to ASIPs offering this trade-off as well, FPGAs have to be programmed in a Hardware Description Language (HDL) which is in most of the cases a more time-consuming and complex task.

Thus, the main advantages are: (1) ASIPs offer enhanced flexibility through high level programming in C/C++. (2) The instruction set architecture is tailored to a specific application. Hence, ASIPs feature increased area and energy efficiency compared to standard embedded processors.
In [4] a flexible, low-power GNSS receiver architecture based on application specific hardware blocks and using an ASIP as central processing unit is proposed. This paper focuses on the implementation of an ASIP based navigation processor as well as the comparison of ASIP and standard embedded processors’ performance and costs in terms of area and energy consumption. A feasibility study for an implementation of a Position, Velocity and Time (PVT) estimation algorithm on an ASIP is presented. Therefore, the developed ASIP should provide position estimation at a rate of 1-10 Hz at a maximal commonly used processor clock rate of 20 MHz. Special emphasis is placed on performance and cost comparisons of standard embedded processors and ASIPs developed throughout this work as well as a detailed description of the design flow used to implement the ASIP and associated software development tools.

This paper is organized as follows: Section 2 describes the standard GNSS receiver hardware architecture. Standard embedded microprocessors of currently available GNSS single-chip receivers are presented and the potential range of an ASIP’s application is pointed out. Details of the design flow including the automatic generation of software tools (i.e. compiler, assembler and linker) as well as hardware description files are introduced in section 3. The applied modular optimization flow is described in section 4. Section 5 compares the performance of different ASIP versions and costs in terms of silicon area and energy consumption with embedded processors currently used in single chip GNSS receivers. Finally, section 6 concludes the paper and presents modifications of the ASIP’s architecture to take over tasks of the correlator channel control.

2. STANDARD GNSS RECEIVER IMPLEMENTATIONS

From a hardware designer’s point of view state-of-the-art single-chip GNSS receivers like [5], [6], [7] and [8] are composed of a variety hardware macros. Here, the term “macro” is used to emphasize that these are functional blocks in a single chip instead of dedicated chips in a chipset solution. Commonly used macros include an analog frontend, a dedicated correlator and a programmable embedded processor. Additionally, current receivers usually include on-chip data and program memories, a variety of external interfaces as well as acquisition accelerators and real-time clocks. A simplified hardware architecture block diagram of a single-chip GNSS receiver is shown in Figure 2.

As shown, this architecture can be considered as a programmable central processing unit (CPU), enhanced with various co-processors. For the realization of the CPU, receivers from SiRF, ST and ATMEL use an ARM7TDMI embedded
processor macro, which is widely used in mobile consumer electronic devices [9]. The ARM7TDMI offers a 16 and 32 bit instruction set, a three-stage pipeline and is designed for low power processing. Besides an optimized version a synthesizable ARM7TDMI is available.

In contrast to these ARM processor based receivers, Nemerix’ single-chip receivers feature the LEON processor. This processor is based on SPARC V8 architecture and features a five-stage pipeline. To increase the flexibility the LEON can be configured using a graphical user interface [10]. Cross compilers as well as Simulators allow software development and debugging of processor and application code.

As shown in Figure 2 coupling of co-processors and CPU is realized by an on-chip bus. Co-processor access is achieved by mapping the components into the CPU’s memory address space.

Main tasks of the CPU are correlator channel control (i.e. acquisition and tracking) and PVT estimation, together commonly referred to as navigation processor. Figure 3 shows a simplified signal processing flow of a general GNSS receiver. To perform channel control and PVT estimation, the CPU has to read the correlator’s integration registers, to synchronize the incoming data bits and to save the received raw ephemeris data. Additionally, the channel control acquires and tracks the satellites by adjusting the frequencies of the correlator’s digitally controlled oscillators (DCOs) and periodically measuring the observables of tracked satellites. Based on raw ephemeris data and measured observables, the PVT estimation computes receiver’s position, velocity and time and applies a variety of corrections transforms the position to a local coordinate system.

The standard PVT estimation algorithm (Figure 3) starts with some initial calculations supporting the determination of line of sight vectors and values which do not depend on the following iterative position estimation process. Subsequently, based on local time estimation and decoded ephemeris data the satellite positions are determined and the measured time of flight is corrected for ionospheric and tropospheric delays. Finally, the receiver position can be computed using the satellite positions and the corrected time-of-flight. Generally, the estimated position is transformed into WGS84 coordinates, converted into a standard serial format (e.g. NMEA 0183) and transferred to a host. Additionally, the Dilution of Precision (DOP) value can be used to assess the geometrical constellation of the received satellites.

This paper focuses on the implementation and performance comparison of this PVT estimation algorithm on an ASIP and standard embedded microprocessors (e.g. ARM7TDMI, LEON). As will be described in section 4 the ASIP’s design and optimization flow is based on a detailed profiling of the application software. For this purpose, the PVT estimation algorithm shown in Figure 3 allows for a detailed breakdown of receiver costs and profiling of the PVT sub-functions.

3. APPLICATION SPECIFIC INSTRUCTION SET PROCESSORS (ASIP)

The ASIP’s design flow shown in Figure 4 starts with a description of the ASIP architecture using an Architecture Description Language (ADL). First implementations are based on a template processor belonging to the same processor class (i.e. RISC) as the target processor. For the development and any further ASIP optimizations the LISA ADL, associated tools and a template RISC processor have been used throughout this work. The LISA tools offer a framework allowing for an efficient exploration of the ASIP’s design space by offering an automatically generated cycle accurate simulator as well as processor specific software development tools including a semi automated compiler generation using the CoSy compiler system. Basic LISA language constructs allow for definition of the processor’s instruction set,
register file and memory access. Definitions of new instructions contain three main parts: (1) Definition of assembler syntax. (2) Mapping to internal binary representation. (3) Behavioral model of the new instruction. The new instruction has to be inserted into the processor’s coding tree. Based on the ADL description assembler, linker and compiler are generated, enabling software development. In a next step the executable program generated through the compilation process can be tested on an automatically generated cycle accurate simulator to verify the correct functionality. Another important feature of the simulator is a detailed profiling to determine performance values.

As a link to hardware, LISA offers the option to generate a hardware description of the defined processor in VHDL or Verilog. This description can be functionally verified using HDL simulators and synthesized using FPGA or standard cell synthesis tools. Additionally, the generated Hardware Description Language (HDL) code allows for real-time implementation and functional verification of the ASIP using an FPGA. Based on performance and cost values provided by this design flow, the designer can characterize and compare the developed ASIP.

For this paper the PVT estimation algorithm presented in section 2 has been implemented on the simple RISC-like template processor. After compilation of the PVT estimation source code the processor simulator allows for a characterization of the PVT sub-functions in terms of cycles per function call and function calls per PVT estimation. The results of this profiling are depicted in Figure 5 and allow identification of performance critical functions as well as the required overall performance.

The different colors in Figure 5 refer to the sub-functions and the symbols refer to the number of satellites received. It can be seen that the required performance of some functions varies depending on the number of received satellites which may affect the number of calls as well as the required cycles per call. Therefore, in Figure 5 the dashed lines indicate points in the diagram where the product of function calls per PVT and cycles per function call is constant and hence a constant performance is required. For instance, the sub-function responsible for the computation of the satellite position of 5 satellites nearly causes the same computational load as the computation of the mean anomaly for 9 satel-
lites. Additionally, the diagram can be used to determine the overall number of cycles required to perform one PVT estimation by summing the required cycles of all sub-functions. For the case of seven satellites in view this leads to a number of approximately 75 million cycles. This clearly conflicts the given constraints of 1-10 Hz position rate and a processor clock frequency of 20 MHz and makes the processor impracticable for real-time processing.

A comparison of these results with an ARM processor reveals that the execution of the same PVT estimation code takes only approximately 1 million cycles on an ARM7 processor. A comparison of the required cycles on the ARM processor and the ASIP as well as a detailed investigation of the basic functions reveals that floating point operations as well as trigonometric and arithmetic functions frequently used in the PVT estimation are responsible for the large number of processing cycles compared to the ARM.

One possible approach to enhance the performance of critical functions is using a co-processor as presented in [11]. Using a co-processor results in a three times better performance than the ARM processor and thus real-time requirements are easily achieved. However, the costs are increasing through the use of additional hardware. Therefore, this work focuses on an optimization of the processor’s standard libraries and instruction set to enhance the overall performance and to achieve the real-time requirements.

4. ASIP OPTIMIZATIONS

This section gives an overview about the use of standard libraries in an ASIP’s compilation process and introduces a modular approach to enhance the ASIP’s performance by optimizing the standard floating point libraries as well as the processor’s ALU. The key idea is to choose an optimized implementation for each performance critical library function and further enhance this performance by inserting special instruction into the processor’s ALU.

Standard and Floating Point Libraries

Figure 6 visualizes a simplified compilation process used for programmable architectures to show the use of standard libraries. In a first stage the application code is translated by the compiler to a platform independent Intermediate Representation (IR). Subsequently, the IR is transferred into a hardware specific machine language using the assembler and the processor’s instruction set. In the last stage the linker resolves references to precompiled functions in archived in libraries and generates an executable.

A commonly used standard library is for instance, the math library (i.e. math.h) implementing mathematical functions like square root, power or trigonometric functions. Additionally, processors featuring a fixed point ALU and no dedicated Floating Point Unit (FPU) emulate single and double precision floating point instructions in software using floating point libraries.

Compared to libraries used for general purpose processors the ASIP environment has to provide a flexible library concept. Hence, as the instruction set changes with every ASIP optimization the CoSy compiler generation environment provides generic implementations of standard and floating point libraries. While these libraries are essential for a generic processor design and the exploration of the processor’s design space, they result in a poor ASIP performance of the PVT estimation function. The main goal of this paper is the optimization of the standard library performance to achieve real-time requirements of the ASIP. In the next section the optimization flow used throughout this work is presented. It incorporates work on the software and library level as well as adding new processor instruction in the processor’s ALU.
Optimization of the standard libraries

As shown in Figure 5 and stated in section 3 real-time requirements cannot be met using the template processor. A detailed profiling of the PVT functions reveals that floating point instructions and mathematical functions cause a high computational load. Thus, floating point instructions as well as mathematical functions will be optimized throughout this work.

To keep the amount of flexibility as high as possible a modular optimization approach is used, starting with a profiling of the application and identification of performance critical functions using the LISA processor debugger (Figure 7). In a next step these functions are optimized and the ASIP’s instruction set is adapted. For instance, the performance of trigonometric functions implemented in the math standard library is enhanced by porting the newlib library to the ASIP [12]. In contrast to this floating point libraries are implemented in assembler based on libraries of different high performance embedded processors (e.g. ARM, LEON). Additional processor instructions required for the implementation of the optimized floating point libraries are added to the ASIP’s instruction set using the LISA ADL. For example, a carry flag based addition and subtraction instruction as well as a count leading zeros instruction are added to the ASIP’s ALU. For performance-cost evaluation a dedicated 32 bit and 64 bit multiplier have been subsequently added to the ASIP. Finally, hardware and software libraries are tested using the LISA processor debugger and new functions are inserted into the standard libraries. Performance enhancements as well as area and energy costs of each added instruction can be characterized using the LISA design flow and are depicted in the following section.

5. RESULTS

Performance

This section presents the performance results which could be achieved using the optimization flow presented in section 4. The successively performed optimization steps are: (1) Replacement of the copy function (i.e. memcpy) (2) Optimization of the floating point instructions (3) Substitution of the math standard libraries (4) Enhancement of the floating point performance using 64 bit unsigned multiplier.

To support generic data types the template ASIP uses the memcpy function for each internal data transfer. The overall number of processing cycles can be reduced by replacing the memcpy function using dedicated load and store instructions. By applying this first optimization the number of processing cycles required for a test case with 7 satellites in view is reduced from about 75 to 70 million.

The first optimization step targets the floating point instructions. Figure 8 compares the number of cycles required for the PVT sub-functions after applying this optimization step and the implementation on the template RISC processor. In this step arithmetic functions (e.g. subtraction, addition and multiplication), compare operations, conversion functions (e.g. integer to floating point) and integer operations are optimized. Additionally, a 32 bit dedicated signed integer multiplier has been added to the processor architecture. As the functions of the math standard library are generally implemented using a Taylor series approximation they are optimized implicitly in this step.

As can be seen in Figure 8 the number of required processor cycles of the PVT sub-functions could be reduced by more than one order of magnitude. Especially the function computing the distance matrix which is called 21 times for 7 satellites in view could be reduced by a factor of about 30. The overall number of processing cycles for a test case of 7 satellites in view could be reduced from 70 (i.e. optimization step 1) to 1.9 million processor cycles.
For a detailed overview about the achieved performance enhancements Table 1 and Table 2 summarize the mean number of processing cycles required for floating point instructions and mathematical functions before and after the optimization.

### Table 1. Optimization of basic operations

<table>
<thead>
<tr>
<th>Function</th>
<th>Mean Number of Cycles (standard)</th>
<th>Mean Number of Cycles (optimized)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>3977</td>
<td>98</td>
</tr>
<tr>
<td>Subtraction</td>
<td>4081</td>
<td>100</td>
</tr>
<tr>
<td>Multiplication</td>
<td>6354</td>
<td>124</td>
</tr>
<tr>
<td>Division</td>
<td>10375</td>
<td>709</td>
</tr>
</tbody>
</table>

### Table 2. Optimization of mathematical functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Mean Number of Cycles (standard)</th>
<th>Mean Number of Cycles (optimized)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cosine</td>
<td>170393</td>
<td>4228</td>
</tr>
<tr>
<td>Sine</td>
<td>154102</td>
<td>3771</td>
</tr>
<tr>
<td>ArcCos</td>
<td>296474</td>
<td>9313</td>
</tr>
<tr>
<td>ArcSine</td>
<td>249012</td>
<td>7790</td>
</tr>
<tr>
<td>Square Root</td>
<td>175445</td>
<td>5996</td>
</tr>
</tbody>
</table>

The next optimization step targets the ASIP’s mathematical standard libraries. Here, this approach is based on the newlib library optimized for 32 bit embedded systems which are adapted for the ASIP architecture [12]. Figure 9
summarizes the results of this optimization step. The overall execution time for the test case could be reduced to 1.4 million cycles.

To further enhance the performance of the GNSS ASIP a dedicated 64 Bit unsigned multiplier was inserted into the processor architecture. The 64 bit multiplier reduces the mean number of required processing cycles for a floating point multiplication from about 124 to approximately 75 cycles. Figure 10 compares the required processing cycles for a GNSS ASIP with and without the 64 bit multiplier for the PVT sub-functions. By using the 64 bit multiplier a reduction of the overall number of processing cycles for the test case from 1.4 to 1.2 million cycles could be achieved.

![Figure 10](image)

**Figure 10:** Performance comparison of GNSS ASIP with (green) and without (blue) 64 bit multiplier

**Area**

As each inserted instruction enlarges the silicon area of the ASIP Table 3 compares the required area of the template ASIP and the GNSS ASIP with and without the 64 bit multiplier. The area values are derived by synthesizing the automatically generated VHDL code using a 180 nm TSMC standard cell technology and the Synopsys Design Compiler.

It can be seen that the insertion of the 32 and 64 bit multiplier into the ASIP architecture results in area enlargement by a factor of 1.7 and 1.3 respectively.

**Table 3:** Comparison of the Silicon Area of Processor Configurations

<table>
<thead>
<tr>
<th>ASIP Configuration</th>
<th>Template ASIP</th>
<th>GNSS ASIP</th>
<th>GNSS ASIP (64 bit multiplier)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area [mm²]</td>
<td>0.19</td>
<td>0.33</td>
<td>0.45</td>
</tr>
</tbody>
</table>

**Power consumption**

This section compares the power consumption of the three processor configurations for the standard cell implementation. As the power consumption mainly depends on processor’s activity and thus on the executed application the power is estimated using the flow presented in Figure 11. Basis of the power estimation is the synthesized processor at gate level and the activity files of each PVT implementation determined by a gate level simulation. Using this design flow the design’s power consumption can be estimated depending on the application and the processor design.

Table 4 shows that the optimization presented above result in an increased power consumption of approximately a factor of 2.6.

**Table 4:** Comparison of the power consumption of processor configurations

<table>
<thead>
<tr>
<th>ASIP Configuration</th>
<th>Template ASIP</th>
<th>GNSS ASIP</th>
<th>GNSS ASIP (64 bit multiplier)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power [mW] @ 20 MHz</td>
<td>2.22</td>
<td>6.28</td>
<td>6.36</td>
</tr>
</tbody>
</table>
To determine the dynamic of the ASIP’s power consumption three test scenarios have been implemented including execution of No Operation (NOP), a typical scenario and PVT estimation. Summarize the results for the different scenarios revealing a dynamic of the power consumption by a factor of approximately 4.

Table 5: Comparison of the GNSS ASIP’s power consumption for different scenarios

<table>
<thead>
<tr>
<th>Scenario</th>
<th>NOP</th>
<th>Typical</th>
<th>PVT estimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power [mW] @ 20 MHz</td>
<td>1.65</td>
<td>3.44</td>
<td>6.28</td>
</tr>
</tbody>
</table>

Benchmarking against Standard Processors

This section compares the GNSS ASIP’s performance determined in the preceding sections and design costs in terms of power and area consumption with two commercially reference navigation processors (i.e. ARM7TDMI and LEON 2).

Figure 12 compares the required processing cycles of the GNSS ASIP and the LEON 2 assembled in the Nemerix base-band processor [6]. The values of the LEON 2 processor have been determined using the cycle accurate LEON 2 simulator [13]. As can be seen the GNSS ASIP reduces the overall number of required processing cycles for a complete PVT estimation with 7 satellites in view from 7.1 million by factor of 5 to 1.4 million cycles.

Figure 13 shows a comparison between the GNSS ASIP and ARM7TDMI used in commercial GNSS receivers by ST, ATMELE and SiRF. As can be seen the ARM processor offers a better performance than the GNSS ASIP. Even if the 64 Bit multiplier is used the ARM processor exceeds the current version of the GNSS ASIP by a factor of 1.3.
main calculations
 GPS system time
 satellite position
 azimuth, elevation
 pseudorange
 pseudorange error
 code

Figure 13: Comparison GNSS ASIP (blue) and ARM7TDMI (green)

Table 6. Comparison of navigation processor power consumption

<table>
<thead>
<tr>
<th>Processor</th>
<th>ARM7TDMI</th>
<th>LEON 2</th>
<th>GNSS ASIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power [mW] @ 20 MHz</td>
<td>4.2</td>
<td>---</td>
<td>6.28</td>
</tr>
<tr>
<td>Area [mm²]</td>
<td>0.59</td>
<td>0.35</td>
<td>0.33</td>
</tr>
</tbody>
</table>

Table 6 compares processor area of GNSS ASIP, LEON and ARM7TDMI as well as power consumption of GNSS ASIP and ARM7TDMI. Nevertheless, the power consumption values in Table 6 are just estimates as for the ARM processor there is no information about the switching activity of the design. The main information which can be deduced from Table 6 is that although the ASIP’s VHDL code is generated automatically and a basic standard cell synthesis script is used, the GNSS ASIP’s power consumption and chip area are in the order of magnitude of commercial processors.

7. CONCLUSION

As a step towards a flexible, low-power GNSS receiver architecture presented in [4] this paper presents the implementation of the ASIP based central processing unit. The results of this feasibility study show that the developed ASIP offers performance comparable to standard embedded processors and allows for real-time implementation of a standard PVT algorithm. To achieve real-time requirements the ASIP’s floating point and standard libraries have been optimized using a modular hardware and software based approach. Using this optimization strategy PVT estimation at a rate of 1-10 Hz and a typical navigation processor clock speed of 20 MHz could be achieved.

For the standard PVT estimation algorithm used throughout this work the additional 64 bit multiplier is not needed to achieve real-time requirements and is left out in current designs to reduce the overall chip area. Nevertheless, it is an interesting option to reduce the number of processing cycles of PVT algorithms causing high computational load.

A first comparison of the developed GNSS ASIP and embedded processors used in standard navigation receivers show comparable results concerning performance and costs in terms of area and power consumption.

As the idea introduced in this paper is not limited to a special kind of algorithm, the developed ASIP will be used to realize high performance PVT algorithms and functions of the correlator channel control (e.g. FFT acquisition, serial search, tracking loop control) in a next step. Therefore, additional ports will be added to the GNSS ASIP’s ADL description to ensure a tight coupling of ASIP and correlator channels. Additionally, interrupt functionality will be inserted into the ASIP architecture to support widely used interrupt based acquisition and tracking.
8. REFERENCES


[9] www.arm.com


