Software-Defined GPS Receiver Implemented on the Parallella-16 Board

Olesen, Daniel Madelung; Jakobsen, Jakob; Knudsen, Per

Published in:
Proceedings of the 28th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2015)

Publication date:
2015

Document Version
Peer reviewed version

Software-Defined GPS receiver Implemented on the Parallella-16 Board

Daniel Olesen, Jakob Jakobsen, Per Knudsen, *DTU Space*

BIOGRAPHY

Daniel Olesen obtained his MSc. degree in Electronics engineering with specialties in control theory and robot-technology from the Technical University of Denmark (DTU) in 2012. He is currently a PhD student researching in GPS/GNSS and inertial navigation methods for small UAVs at the department of Geodesy at DTU Space.

Jakob Jakobsen, researcher at DTU Space department of Geodesy, PhD, has worked with research, development and teaching related to GPS/GNSS for more than 20 years. His main interests are precise positioning and navigation for small UAVs.

Per Knudsen, Head of Department of Geodesy, PhD, is responsible for R&D activities related to GPS/GNSS at DTU Space/National Space Institute as well as for the operation of permanent GPS reference stations in Greenland.

ABSTRACT

This paper describes a GPS software receiver design made of inexpensive and physically small hardware components. The small embedded platform, known as the Parallella-16 computer has been utilized in conjunction with a commercial RF front-end to construct a 4-channel real time software GPS receiver. The Parallella-16 board is a kickstarter-funded platform consisting of a dual-core ARM A9 CPU, an integrated FPGA and a 16-core coprocessor known as the Epiphany. The main contribution in this paper has been the development of a GPS tracking algorithm, which utilizes the parallelism in the Epiphany processor. The total cost of the hardware is below 150$ and the size is comparable to a credit-card. The receiver has been developed for research in GNSS/INS integration on small Unmanned Aerial Vehicles (UAVs).

INTRODUCTION

In this paper, a 4-channel real-time software defined GPS receiver implemented on a low-cost, credit-card sized parallel-computing platform is presented. The chosen hardware is the kickstarter-funded Parallella-16 board from Adapteva [1]. The Parallella-16 provides a flexible base with a Xilinx Zynq-7010 System on a Chip (SoC) [2], which features a Processing System (PS) consisting of a dual-core ARM Cortex A9 processor and Programmable Logic (PL) from an integrated FPGA with

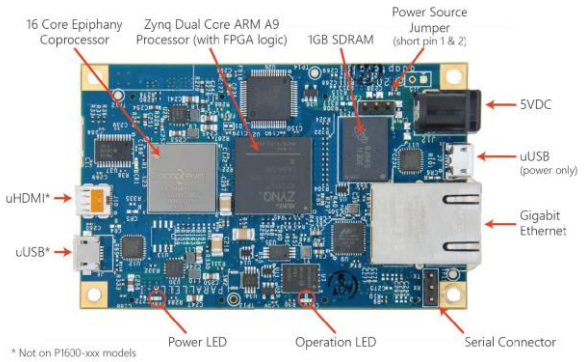

Figure 1: Parallella-16 board (top view) [1]

28000 logic cells. In addition the board contains a 16-core coprocessor, Epiphany E16G301 developed by Adapteva. The Parallella-16 design resources; software, HDL sources and schematics are open source and has a growing community for support and further development. The receiver front-end is the Maxim MAX2769 GNSS RF front-end, which provides flexible operation and capability to receive GPS L1, GLONASS G1 and Galileo E1 frequency bands.

The reasoning behind the choice of hardware has been to develop a low-cost GPS software defined receiver (SDR) which utilizes parallel processing in the Epiphany multi-core processor and thus provides a flexible base with direct access to low-level tracking variables. The receiver is intended to be used for research, with a special focus on integration schemes for inertial sensors in UAV applications. Parallel implementations of GNSS acquisition- and tracking algorithms are not a new concept. The work of [3] describes how OpenMP and MPI implementations significantly can speed up algorithms on multi-core processors. A number of authors have published papers for executing GNSS algorithms utilizing the massive parallelism of modern GPU's using CUDA or OpenCL [4], [5]. The aforementioned work, mostly applies to desktop-computers and in the design of high-end software receivers. In our work, we utilize parallelism in the small, low-cost Epiphany multi-core processor. The key advantages of the presented system are the flexibility of the software-defined receiver and the compact size of the hardware as it could easily be equipped into small UAVs. The integrity and performance of the receiver, has been compared to a MATLAB-based SDR [6].

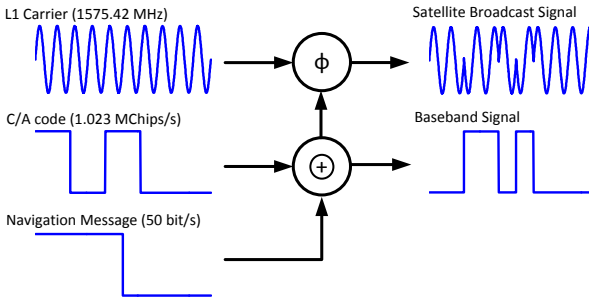

Figure 2: Modulation of C/A code and Navigation Message for GPS L1.

The paper begins with a brief review of the GPS C/A signal, as the tracking algorithm has been based on this.

A short review of the hardware architecture of the Parallella-16 board is then given and an explanation of how the data acquisition from the front-end is implemented in PL.

Hereafter, the developed parallel tracking algorithm for the Epiphany processor is discussed and it is shown how the receiver is able to process data within real-time constraints by using multiple cores per channel.

Finally the tracking of the receiver is compared to the aforementioned MATLAB-SDR implementation and further development and outlook is discussed.

GPS C/A Signal

The GPS Coarse / Acquisition (C/A) code is a 1023 chip Pseudo-Random Noise (PRN) sequence transmitted on the GPS L1 band with a carrier frequency of 1575.42 MHz. The C/A code is modulated on to the carrier by Binary Phase Shift Keying (BPSK). As all GPS satellites are using the same carrier-frequency, a Code Division Multiple Access (CDMA) coding scheme is applied on the C/A code. The C/A code is for each satellite a unique PRN code known as a gold code. Gold codes have a guaranteed minimum cross-correlation with other gold codes and hence the satellite transmitting the signal can be identified by correlation with a receiver generated replica.

The chiprate of the C/A code is 1.023 Mchips/s, corresponding to a code-sequence length of 1 ms. The C/A code is modulated with the satellite navigation message, which contains the ephemeris, time etc. for the satellite. The signal structure for the GPS L1 C/A signal is visualized in Figure 2.

Mathematically the broadcast signal is defined as:

$$S_{SV}(t) = \sin(2 \cdot \pi \cdot f_{carr} \cdot t + \phi) \quad (1)$$

$$\phi = \pi \cdot \left(S_{C/A}(t) \oplus S_{Nav}(t)\right) \quad (2)$$

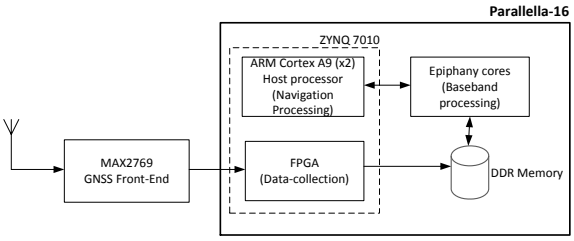

Figure 3: High-level block diagram for the developed GPS receiver.

where $\oplus$ is modulo-2 addition, also commonly referred as an XOR operation. $S_{C/A}(t)$ is the unique gold code for the satellite and $S_{Nav}(t)$ is the satellite navigation message. (1-2) assumes the binary levels of the C/A code and navigation message is $[0; 1]$.
If we instead should assume the levels to be $[-1; 1]$ we could equivalently state the broadcast signal to be:

$$S_{SV}(t) = \sin(2 \cdot \pi \cdot f_{carr} \cdot t) \cdot \left(S_{C/A}(t) \oplus S_{Nav}(t)\right) \quad (3)$$

The final equation (3) is commonly used in the GNSS literature.

HARDWARE DESCRIPTION

The system architecture of the receiver is based on the MAX2769 GNSS RF front-end [7] and the Parallella-16 board [1]. A high-level diagram of the hardware utilization is shown in Figure 3. The Figure furthermore shows the basic architecture of the Parallella-16 hardware. The RF front-end is configured to receive GPS L1 signals ($f_{carr} = 1575.42\ MHz$) and down-convert the signal to an Intermediate frequency (IF) of $f_{IF} = 4.092\ MHz$. The front-end discretizes the IF signal at a sampling rate of $16.368\ MHz$ with a resolution of up to 4-bits. The sign bit from the AD conversion and a reference clock is connected to digital inputs on the Parallella-16 board. Data acquisition of the Discretized IF signal (DIF) has been implemented in the PL on the Zynq SoC, in order to store the samples in a circular-buffer in DDR memory. Finally, the Epiphany multi-core processor is utilized for tracking of the satellites.

RF Front-end (MAX2769)

The RF Front-End is based on the super heterodyne receiver principle, by using a local oscillator with a fixed frequency to down convert (mix) the incoming RF signal down to IF. The local oscillator can be programmed to any given frequency in the range 1550-1610 MHz. In our application we have used a low side injection oscillation frequency of $F_{OSC} = 1571.328\ MHz$, as this gives the desired IF of $F_{IF} = F_{RF} - F_{OSC} = 1575.42 - 1571.328 = 4.092\ MHz$. The IF bandpass filter can be programmed to have a bandwidth of 2.5 MHz, 4.2 MHz, 8 MHz and 18 MHz. In addition the filter can be implemented as a 3rd or 5th order polyphase filter. We

have selected the bandwidth to 2.5 MHz using the 5th order filter setting.

Data Acquisition (Zynq PL/FPGA)

The data acquisition of the system is implemented in the PL of the Zynq7010 SoC. The design consists of Xilinx DMA IP core and a developed acquisition module. The implementation is made using VHDL and Xilinx ISE development tools. In Figure 4, a block diagram of the acquisition system is shown. The data acquisition block is connected to the sign-bit of the ADC output from the MAX2769 front-end and a reference sample clock. In the other end, an Advanced eXtensible Interface (AXI) stream master port connects to a Direct Memory Access (DMA) core [8]. The DMA core is a soft Intellectual Propererty (IP) core from Xilinx. The core is connected through a General Purpose (GP) and a High Performance (HP) interconnect to the Zynq PS. The IP core is memory mapped, such that configuration registers can be set directly from the PS by writes to a specific address in the memory-system. The DMA core can be configured in a simple operating mode, where each transfer is initiated by the Processing System writing to the memory-mapped control registers. In this design, the DMA is configured for Scatter-Gather operation. This allows the DMA to automatically process a linked-list of DMA descriptors, without requesting processing time from the PS system.

Host Processor (Zynq PS)

The host processor (Zynq PS) of the parallella-16 hardware is a dual-core ARM Cortex A9 processor running at 667 MHz. The processor features Gbit Ethernet connectivity, USB 2 compliance, 32 KB Level 1 cache for each core and a 512 KB Level 2 cache shared between the cores. In addition the ARM cores are equipped with a 128-bit Single-Instruction Multiple Data (SIMD) engine, NEON™, which can help accelerate multimedia and signal processing tasks. The processor is equipped with 1 GB DDR3 external memory.

The host-processor runs an ubuntu distribution and controls all peripheral devices in the Zynq SoC. The PS furthermore controls loading and launching of programs

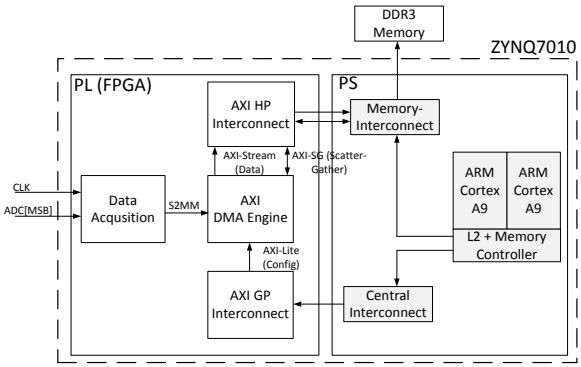

Figure 4: Block diagram of Data Acquisition system in Zynq SoC.

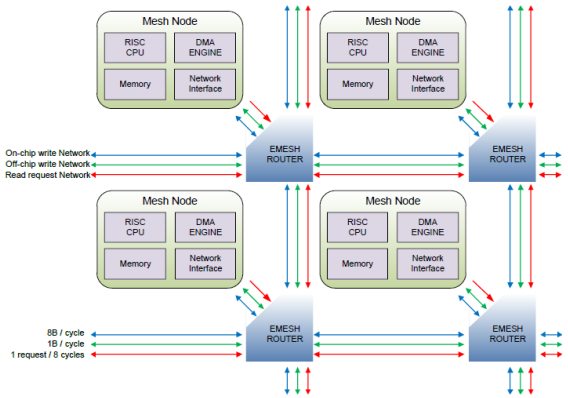

Figure 5: Epiphany eMesh architecture [9]

in the Epiphany multicore coprocessor.

Epiphany coprocessor

The basic layout of the Epiphany processor is shown in Figure 5. The processor is designed as a network-on-chip and consists of 16 mesh-nodes arranged in a 2D grid, where each node containing a DMA engine, a RISC processing core, a network interface and 32 KB of local memory. The cores are operated at 600 MHz and capable of doing two floating point operations and one integer calculation per clock-cycle. The entire memory system of the Epiphany can be accessed from each node, where 3 interconnects are responsible for write transactions, read transactions and off-chip memory access. For more detailed specifications about the Epiphany architecture the reader is referred to [9]. The programming of the Epiphany cores has been done using a native bare-metal C/C++ framework controlled from the Zynq PS. An OpenCL implementation would also have been possible, since a free compiler for the Epiphany architecture is available. The native framework allows loading of separate programs to each node. It furthermore allows configuration of a workgroup, where a single program can be loaded to each member simultaneously (SPMD). The definition of the workgroup, allows for relative addressing, such that each core can be identified from a relative row and column index.

BASEBAND PROCESSING

In the previous section, it was described how data continuously is stored in a circular buffer in the DDR memory. In this section, the hardware utilization and the algorithms necessary for processing and demodulation of the DIF signal is described. The first task is to identify which satellites are in sight and subsequently start tracking and demodulate the navigation message.

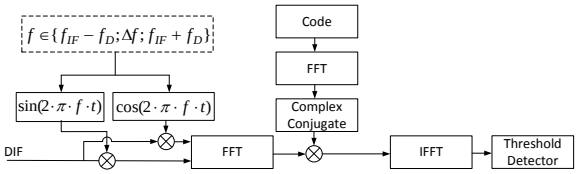

Figure 6: Parallel Code Phase Search

Satellite Acquisition

During receiver initialization, the Zynq PS performs satellite acquisition using the parallel code phase search algorithm (Figure 6) utilizing the fftw-library [10]. From version 3.3.1 the fftw-library includes support for the NEON™ hardware accelerator / SIMD engine featured in the ARM cores. Experiments was initially made with acquisition using the Epiphany multi-core processor, but ultimately have been discarded because of current lack of support for optimized FFT libraries, such as the fftw. After satellite acquisition, tracking loops are initialized in the Epiphany multi-core processor.

Tracking

Preliminary estimates of Doppler frequency and code-delay from satellite acquisition are used to initiate tracking for the acquired satellites. A satellite tracking algorithm for one channel was initially designed to be executed on a single Epiphany core. Despite various optimization efforts, this approach however proved undoable for real-time operation. Due to the high computational demands of the tracking algorithm, we have chosen to distribute the execution of the tracking algorithm for one channel to a workgroup consisting of multiple cores to obtain processing in real-time. The parts of the tracking algorithms which are executed in parallel are illustrated in Figure 7.

The tracking of a GPS satellite requires a Phase-Locked Loop (PLL) for keeping track of the down-converted carrier wave and a Delay-Locked Loop (DLL) for aligning the received C/A code sequence with a receiver generated replica. In Figure 7, the carrier discriminator and carrier-filter blocks comprises the PLL. A special note on the PLL is that it should be insensitive to phase-shifts induced by transitions in the navigation message. This has been accomplished by using a so-called Costas loop, with the following discriminator function:

$$D_{PLL} = \arctan(Q_P / I_P) \tag{4}$$

The DLL is comprised of the code discriminator and the code filter blocks. The discriminator function is evaluated from the Early, Prompt and Late correlations between the incoming DIF signal and receiver generated replicas of the C/A code.

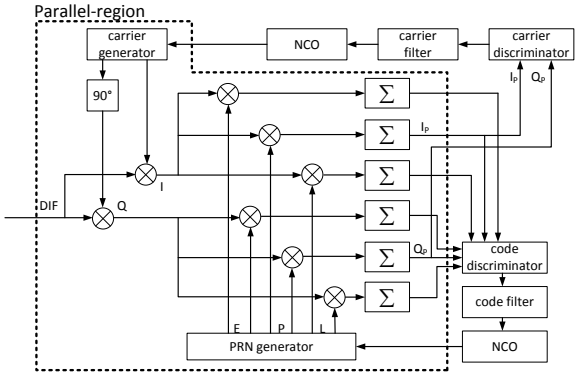

Figure 7: Tracking loops for one channel

The Early and Late correlators have in our implementation shifted the C/A code phase by 0.5 chips. The code-discriminator function has been implemented as:

$$D_{DLL} = \frac{(I_E^2 + Q_E^2) - (I_L^2 + Q_L^2)}{(I_E^2 + Q_E^2) + (I_L^2 + Q_L^2)} \tag{5}$$

In Figure 7, the parallel region includes carrier wipe off for the DIF signal, PRN (gold-code) generation, early, prompt and late correlations and integrations. The basic mathematical principle for parallelization to 4 cores is shown in (6) for calculation of the IP correlator output.

$$\begin{aligned} I_P &= \sum_{k=0}^{N} S_{DIF}(k) \cdot S_{carr}(k) \cdot S_{CA}(k) \\ &= \sum_{k=0}^{(N/4)-1} S_{DIF}(k) \cdot S_{carr}(k) \cdot S_{CA}(k) + \\ &\quad \sum_{k=N/4}^{(N/2)-1} S_{DIF}(k) \cdot S_{carr}(k) \cdot S_{CA}(k) + \\ &\quad \sum_{k=N/2}^{3(N/4)-1} S_{DIF}(k) \cdot S_{carr}(k) \cdot S_{CA}(k) + \\ &\quad \sum_{k=3(N/4)}^{N} S_{DIF}(k) \cdot S_{carr}(k) \cdot S_{CA}(k) \end{aligned} \tag{6}$$

Where N is the number of samples for one CA code sequence, S_{DIF} is the DIF samples from the front-end, S_{carr} and S_{CA} are the receiver generated carrier and C/A code respectively.

A flow-chart for the implementation of the tracking algorithm in the Epiphany multi-core processor are shown in Figure 8.

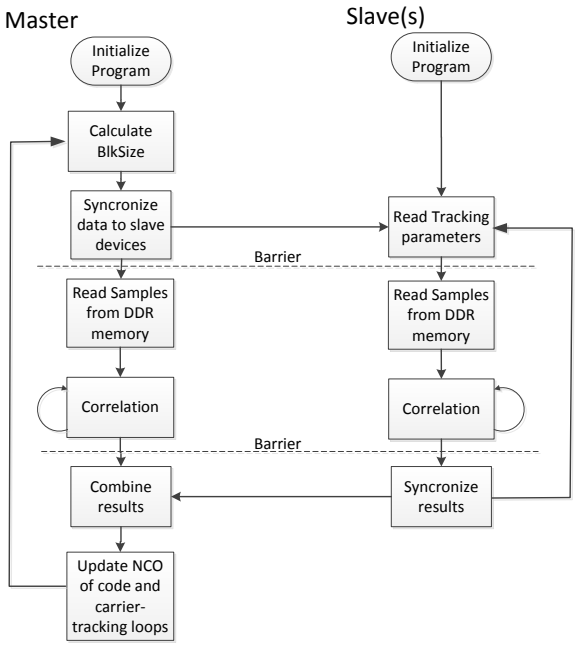

Figure 8: Flow-chart and thread synchronizing between master and slave cores.

The parallelization of the tracking loops have been obtained by assigning a master-thread for estimating the number of DIF samples for the C/A code sequence and later updating the code- and carrier tracking loops. The master thread is furthermore responsible for distributing sub-indices of the DIF samples for processing to the slave threads.

A barrier function is used to synchronize the cores after processing has been completed. The results are then read back to the master-thread. An analogy of the work distribution of the slave threads can be drawn to the OpenMP fork-and-join execution model. Finally, the master-thread combines the results and updates the NCO of the code- and carrier tracking loops before the next 1 ms period is being processed.

BENCHMARKING OF TRACKING ALGORITHM

In terms of benchmarking, Table 1 shows the number of processor-cycles for various program stages in the algorithm. The data has been generated using one core per tracking channel and based on 2000 ms prerecorded DIF samples with a sample rate of 16.368 MHz.

Task	Number of cycles	Execution time
Read samples from DDR-memory	190933053	0.318 s
Update PLL and DLL	25300772	0.042 s
Correlation-Loops	2627483517	4.379 s
Total		4.739 s

Table 1: Timings for single core tracking algorithm (2000 ms DIF data)

From the table, it is evident that most cycles are spent in the correlators, which accounts for 92 % of the total runtime. Transfer times from the DDR memory accounts for approximately 7 % of the total runtime. Finally computation of block-sizes, code- and carrier tracking variables accounts for less than 1 % of the total runtime.

The speed-up obtained by using 2-4 cores per tracking channel and the respective execution time is listed in Table 2.

Number of cores pr. channel	Speed-Up [x]	Execution time [s]
1	-	4.739
2	1.73	2.741
3	2.42	1.964
4	2.91	1.627

Table 2: Parallel speed-up for 2000 ms of DIF data

The obtained speed up from 2 cores is 1.73 times, the speed up using 3 cores is 2.42 times and finally the speed-up for 4 cores is 2.91 times. From Table 2, it can be seen that both 3 and 4 core workgroups obtains processing within real-time constraints. The parallel implementation, does introduce some parallel-overhead in terms of synchronization messages and barrier functions which limits the theoretical speed-up. Another limiting factor is the total bandwidth of the Epiphany interconnects. Each Epiphany core has a dedicated DMA engine, but DMA transfers are utilizing the same communication bus for off-chip (DDR) memory access and thus cannot obtain a linear speed-up by distributing transfers to multiple cores.

IMPLEMENTATION

The implementation in software is shown in Figure 9.
A GPS_main routine controls all the processes of the receiver. Initially a dma driver process is started, which is responsible for constantly updating a fixed-size circular buffer in DDR memory with DIF samples. As earlier explained, the PL handles the actual dma transfers, but it

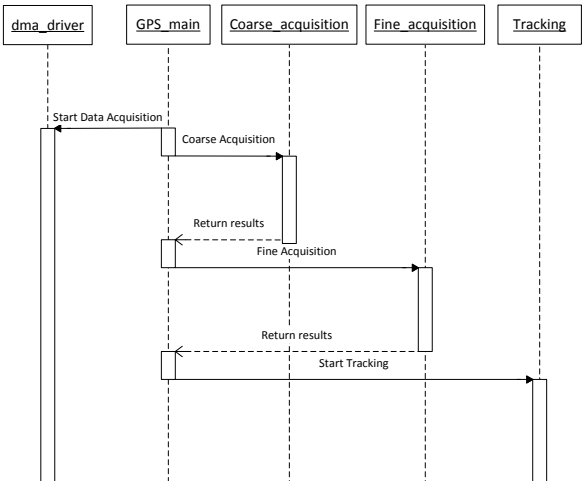

Figure 9: UML Sequence diagram of process execution (Zynq PS)

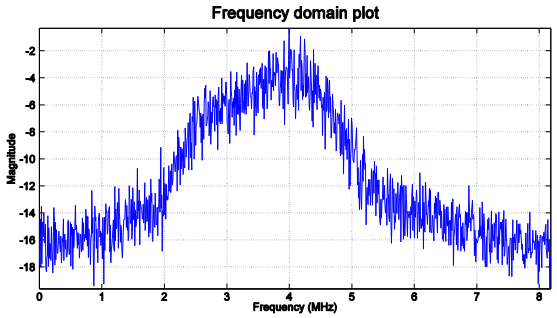

Figure 10: Power-spectral density plot of 10 ms of DIF data.

is the PS responsibility to occasionally update the tail-pointer of the dma-descriptors, such that the transfers are executed continuously. This process continues throughout the execution of the program. The following action is then to obtain a coarse acquisition to determine which satellites are in view. In this phase the parallel-code phase algorithm are executed for all the gold codes and with a frequency interval of 500 Hz. After the coarse acquisition, the task returns the cross-correlation values for each gold code and initiates a fine acquisition search for the 4 SV's with the highest correlation. The fine acquisition task, use a frequency interval of 50 Hz to obtain better estimates of the Doppler frequency and the code-delay. After the completion of the fine-frequency search, a tracking process is executed in the PS. The tracking process allocates 4 Epiphany cores to a workgroup for each channel and subsequently starts the programs. It should be noted from Table 2, that 3 cores theoretically is enough to ensure real-time operation, but since there is some control and management operation performed from the PS, this introduce some additional overhead. In the current implementation, navigation processing has not yet been implemented. Instead the IP and QP correlator outputs are stored in a csv file for inspection in MATLAB. The developed software provides an option for saving the raw DIF data into a binary file, such that the produced tracking results can be verified using the MATLAB SDR implementation provided by [6].

RESULTS

In this section, results from the RF front-end and a visual comparison of the tracking algorithm compared with a post-processed MATLAB version are made. In addition an C/N_0 estimate has been calculated for both implementations.

RF Front-End

A power spectral density plot based on 10 ms of 1-bit DIF data is shown in Figure 10. The figure has been made using a 10-ms average from Welch's power spectral density estimate in MATLAB.

From the figure, it is evident that the passband bandwidth of 2.5 MHz is achieved, with a center frequency of 4 MHz, which is in correspondence with the programmed settings.

Tracking

The system is able to acquire and track satellites from DIF data with a sample rate of 16.368 MHz for 4 channels simultaneously meeting real-time constraints.

In order to validate the obtained results, we have used the MATLAB SDR implementation [6] as a reference to compare the tracking of GPS SV12. The results for 5000 ms DIF data processed by the Parallella SDR system and the reference MATLAB implementation are shown in Figure 11.

A zoom-in on IP outputs in the interval from 1500 to 3500 ms are shown in Figure 12. The Figures show, that our implementation is able to track the SV. The Parallella-16 implementation has more visible noise and smaller amplitude than the reference implementation at the IP

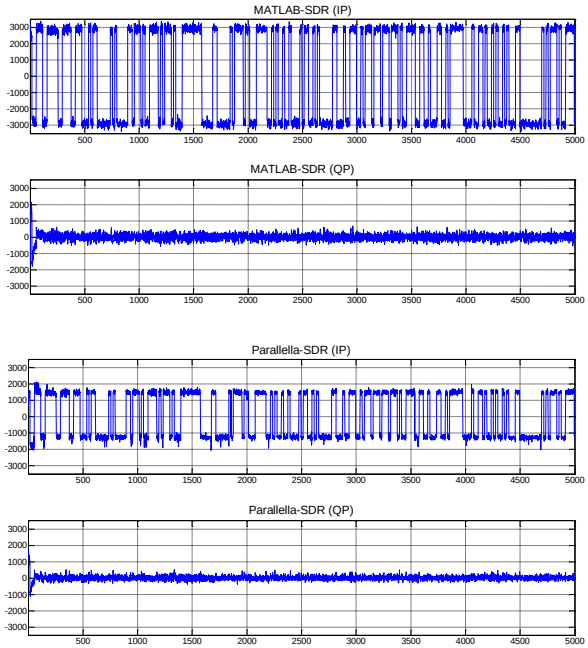

Figure 11: Comparison of 5000 ms output from IP and QP correlator outputs from MATLAB (top) and Parallella-16 implementations (bottom).

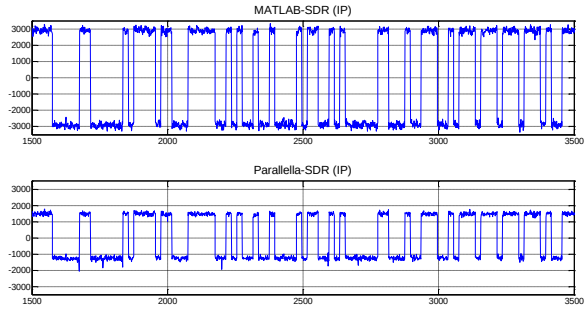

Figure 12: Comparison of IP correlators

correlator. This is believed to be caused by numerical precision issues, as the Parallella-16 implementation operates with single precision and the MATLAB implementation uses double precision. Another factor could be that the Parallella-16 implementation uses a square-wave approximation of the IF carrier-wave.

A carrier-to-noise density (C/N_0) estimate has been calculated on basis of the IP and QP correlator-outputs for both receivers. The estimation algorithm is the commonly known, narrow-band over wide-band power method first described by [11]. The C/N_0 estimate, for both the Parallella SDR and the MATLAB SDR are shown in Table 3.

Receiver	C/N_0
MATLAB-SDR	44.4781 dB-Hz
Parallella-SDR	43.5427 dB-Hz

Table 3: C/N0 estimates of MATLAB and Parallella receiver based on IP and QP outputs.

The estimates are calculated over the same sequence for both receivers using a 1 second average.

The MATLAB SDR performs approximately 0.9 dB-Hz better than the Parallella-SDR implementation.

Future Developments

At the current implementation only GPS L1 C/A code is tracked, it is however a future priority to incorporate support for multi-constellation operability, with inclusion of Galileo E1 and GLONASS L1 signals.

The MAX2769 RF front-end is already capable of reception of GLONASS L1 and Galileo E1 band signals, so the main challenge is to modify the tracking algorithm.

The maximum number of channels is currently limited to 4. Adapteva has announced a 64-core version of the Parallella-board is in the pipeline, which would expand the number of tracking channels with the current implementation to 16. Another option for adding more channels could be to use a cluster of parallella-16 boards.

CONCLUSION

In this paper, we have presented a parallel, low-cost system able to perform GPS tracking satisfying real-time constraints. A software design for a tracking algorithm utilizing parallel execution within the Epiphany multi-core processor has been presented and demonstrated.

Real-time Software-Defined GPS/GNSS receivers are becoming more and more accessible aided by the rapid evolution of computer technology. The current trend of using a higher numbers of cores in parallel requires that programmers are able to exploit the parallelism and identify which parts of a program that can be suitable for parallel execution. A number of publications have been made on parallel implementations of GPS/GNSS algorithms. Commonly these implementations, has been made on high-end dedicated hardware, GPUs or desktop computers.

Our implementation has been based on the small-size, low-cost and commercially available Parallella-16 board, which with the addition of a GPS/GNSS RF front-end can be configured as a real-time Software-Defined GPS receiver. The size and weight of the platform makes it ideally suited for applications with e.g. small UAVs.

ACKNOWLEDGMENTS

The authors would like to thank the Innovation Fund Denmark for partial funding of the first author's PhD study. We would also like to thank Michael H. Avngaard, assistant engineer at DTU Space for his help and assistance in the laboratory during the development of the GPS software-defined receiver.

REFERENCES

[1] Adapteva, *Parallella-1.x Reference Manual*, Lexington, 2014.

[2] Xilinx, *Zynq-7000 All Programmable SoC Overview (DS190)*, 2013.

[3] C.-C. Sun and S.-S. Jan, "GNSS signal acqusition and tracking using a parallel approach," *IEEE/ION Position, Location and Navigation Symposium, Vol.1-3*, pp. 1332-1340, 2008.

[4] U. Haak, H. G. Busing and P. Hecker, "Performance analysis of GPU based GNSS signal processing," *25th Internation Technical Meeting of the Satellite Division of The institute of Navigation*, pp. 2371-2377, 2012.

[5] R. Petr, O. Jakubov, P. Kovar, P. Karmarik and F. Vejrazka, "GNSS signal processing in GPU," *Artificial Satellites, Vol. 48*, pp. 2051-61, 2013.

[6] K. Borre, D. Akos, N. Bertelsen, P. Rinder and S. Jensen, A Software-Defined GPS and Galileo Receiver, Birkhauser, 2007.

[7] Maxim Integrated, "MAX2769 Datasheet," San Jose, 2010.

[8] Xilinx, *LogiCORE IP AXI DMA v6.03a (PG021)*, 2012.

[9] Adapteva, *Epiphany Architecture Reference*, Lexington, 2014.

[10] M. Frigo and S. Johnson, "The Design and Implementation of FFTW3," *Proceedings of the IEEE, Vol 93, Issue 2*, pp. 216-231, 2005.

[11] A. J. V. Dierendonck, "GPS Receivers," in *"Global Positioning System: Theory and Applications", Volume I, Edited by B.W. Parkinson, J.J. Spiker Jr*.