Conceptual Design of a Digitally Accelerated Arithmetic Sequence Generator (ASG)

Philippus Scholtz[†] and Asif Parker[‡]
EEE4084F Class of 2016
University of Cape Town
South Africa
[†]SCHPHI015 [‡]PRKASI003

***Abstract*—This paper details the conceptual design of a digital accelerator that generates an arithmetic sequence based on user provided input parameters. The idea is to design a parallel hardware implementation that will provide high performance computation.**

I. Introduction

An arithmetic sequence is an ordered list of numbers where each consecutive pair differs by the same constant [1]. Such a sequence can be represented by equation 1 where i is the index of the term, a_1 is the first term in the sequence and d is the common difference between two terms.

$$a_i = a_1 + (i-1)d \tag{1}$$

A. Applications of arithmetic sequences

Arithmetic sequences form part of an image encryption algorithm proposed by Zhen et.al. [2]. In their algorithm, arithmetic sequences are generated that have as many elements as there are pixels in the image. For high-resolution images, these sequences could have millions of elements. Speeding up the sequence generation would contribute to improving the performance of their image processing method.

Another potential application is pattern recognition, which is an important branch of machine learning [3]. Training a machine to recognize a pattern often requires providing the machine with a large set of examples. If sequences can be generated faster, the machines can be trained more quickly and accurately to recognize similar sequences. Successful acceleration of arithmetic sequences could lead to similar developments for other more complex patterns often used in machine learning.

Other applications of arithmetic sequences include:

- Frequency generation algorithms [4] [5]
- Arithmetic compression algorithms [6]
- Mathematical design [7]
- Prediction algorithms [8]

B. Computation of arithmetic sequences

An arithmetic sequence can be generated sequentially using a single loop. Each iteration evaluates equation 1 for a particular value of i and stores the result. The variable i is then incremented before the next iteration.

The calculation of one sequence element does not depend on the value of any other element. Each loop iteration depends only on the loop index and the global constant parameters given for the sequence. Each iteration stores its result in a separate memory location that is not accessed in any other iteration. The iterations could therefore run simultaneously or out of order. The resulting sequence would still be correct. The concurrent implementation could be implemented on a general-purpose multi-core processor. However, a specialized hardware accelerator would likely perform better since the hardware can be customized to fit the task. In the following sections of this paper, a design of such an accelerator is presented along with an evaluation of its performance.

II. Design Overview

The diagram in Figure 1 shows the main hardware components associated with the digital accelerator implementation. The CPU, DMA controller and RAM belong to a host computer while the hardware accelerator will act as a peripheral device.

The DDR4 RAM has a data rate of 3200 MT/s with a bus width of 64 bits per module. It has a transfer rate of 21.3 GB/s [9]. This is 170.4 Gbps, which is faster than the Ethernet connection. The RAM is therefore not a bottleneck.

A. Constraints

The accelerator only handles unsigned integers. No negative or floating point values are supported. The generated sequence

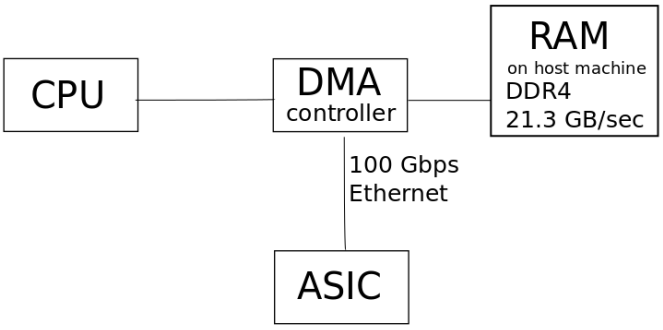

Fig. 1. Hardware architecture overview

elements are 64-bit unsigned integers. If overflow occurs in the arithmetic unit, an *overflow* flag will be pulled high to indicate the error condition. This flag will stay high until the system is reset for a new sequence. Overflow can occur at the subtracter or adder stages (see figure 3)

B. High-level implementation

An overview of the modular view of the hardware accelerator is shown in Figure 2. The hardware acceleration of the arithmetic sequence generation algorithm will be achieved through the use of arithmetic calculation units. The outputs of the arithmetic units are combined into a single bus and passed to the Ethernet controller for transmission back to the host. This is explained further in section III-H.

A control unit keeps track of the element indices being processed. It manipulates the inputs of the arithmetic units to calculate the sequence in groups of M elements, where M is the number of arithmetic units. It raises a *done* flag when the whole sequence has been generated. The aim is to have many arithmetic units to accommodate as much concurrent computation of arithmetic sequence terms as possible.

In addition to making the accelerator do concurrent computation, the throughput of data in the process can be increased through the use of pipelining. This will allow multiple arithmetic terms to be generated on each processing cycle.

C. How will the digital accelerator be used

A software application program interface (API) will be developed for use on the host computer. This API will be the used to specify the a_1, n and d values required to generate a particular arithmetic sequence. The host computer will store these values in RAM before sending them via 100 Gbps Ethernet to the hardware accelerator. The hardware accelerator will then use these inputs to compute the terms of the arithmetic sequence. The resulting data will be sent back to the host computer via 100 Gbps Ethernet. The host computer's DMA controller will be used to efficiently access the host computer's RAM where the data will be stored without having to use the host computer's CPU. The starting memory address of the sequence in RAM will be sent to the API so it is known where to access the data. For each new set of inputs provided to the API, the starting address where the sequence is stored in RAM will be received by the API.

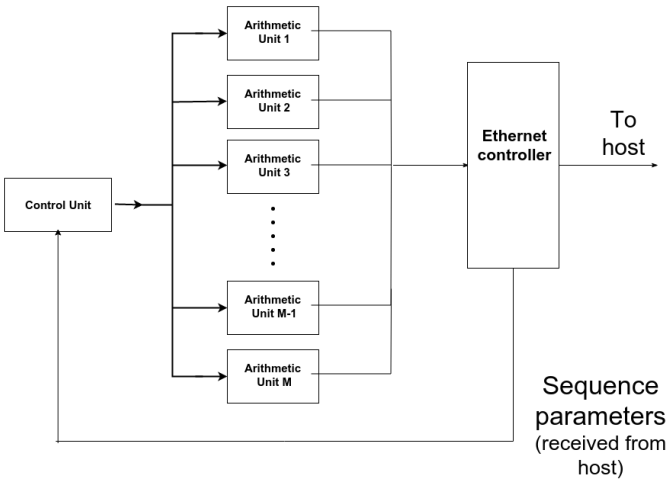

Fig. 2. Modular overview of accelerator

III. Detailed design

The accelerator consists of many arithmetic units and a single control unit.

Some symbols and their meanings are listed in table I.

A. Arithmetic units

The accelerator should have as many of these arithmetic units as possible so that the maximum number of sequence elements can be computed concurrently. Each arithmetic unit contains its index (N) as a constant value wired into its circuit. These indices do not need to be fed in as inputs.

An arithmetic unit is split into a *sequence block* and an *offset block* as shown in figure 5.

B. Calculation pipeline

Without pipelining, the maximum clock speed would be limited by the propagation delay of the entire combinatorial circuit. With pipelining, the clock speed is only limited by the combinatorial delay of the single slowest stage in the pipeline, which is likely to be the multiplier. The pipelined system would output new data on every clock cycle as long as the pipeline is kept full. This, combined with the faster clock, would give a higher throughput than the non-pipelined system. The sequence and offset blocks both have internal pipelines as shown in figures 3 and 4, respectively. Pipeline registers are also added between the two blocks (figure 5). This is necessary because the offset block introduces latency on the i input, which could lead to invalid inputs at the sequence block if it were not pipelined. The combined pipeline forms the arithmetic unit and is driven by a single clock connected to all registers. The sequence and offset blocks are explained individually in the following sections.

C. Sequence block

Each sequence block performs the operations required to evaluate the expression $a_1 + (i-1)d$ for a single value of i. This is one subtraction, one multiplication and one addition.

Each sequence block is pipelined with four stages. A pipeline register is placed between all the arithmetic operations.

Figure 3 shows a single instance of a sequence block. All registers are clocked from the same clock signal. Every clock cycle advances the data to the next stage in the pipeline.

The *data valid* line is pulled high when the inputs are valid. Its value is then clocked through the pipeline and becomes high at the output stage to indicate that the output is valid and ready to be stored or transmitted.

TABLE I
SYMBOLS USED IN THE DESIGN DESCRIPTION

Symbol	Meaning
M	Number of physical arithmetic units on the accelerator
N	Index of each arithmetic unit (1 to M)
n	Number of elements to calculate (given as input)
i	Index of each computed element in the sequence (1 to n)

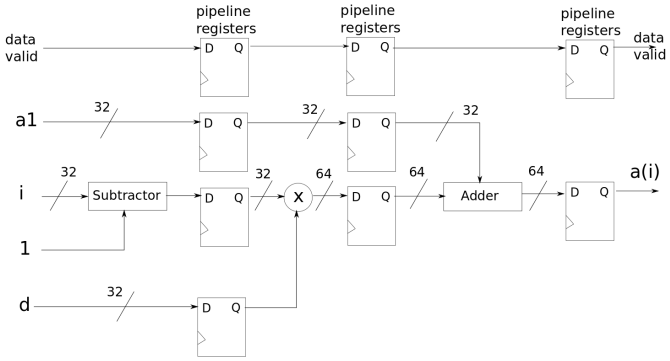

Fig. 3. Sequence block to evaluate $a(i)$

D. Offset block

The accelerator supports cases where $n > M$, i.e. where the required sequence has more elements than the number of arithmetic units. This is implemented by running calculations sequentially in groups of M. After the first M calculations have been clocked in, the values of i must be increased by M for the subsequent group of calculations. The first group of calculations are started with $1 \leq i \leq M$. The second group will start with $M + 1 \leq i \leq 2M$, and so forth. Each offset block receives an input called *groupOffset* (generated by the control unit). Each offset block will calculate $i = N+$groupOffset before placing the new value of i onto the input of the sequence block.

The offset block is added as an additional pipeline stage in front of each sequence block. The implementation of an offset block can be seen in figure 4.

Figure 5 shows how the sequence and offset blocks are connected to form a single arithmetic unit. It also shows an Ethernet core, which is discussed in section III-H.

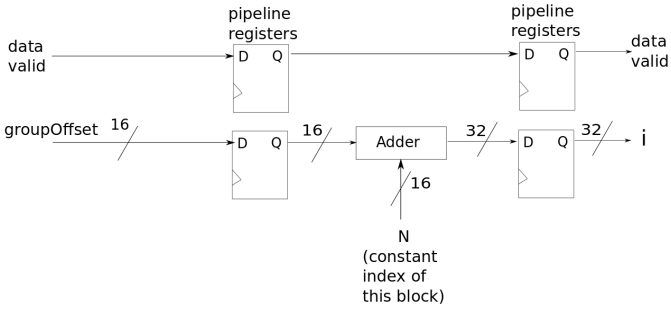

Fig. 4. Offset block to handle $n > M$ cases

E. Control unit

The control unit performs the following functions:

1) starts the calculation process in response to the 'activate' input line
2) counts up in steps of M to maintain the groupOffset value for each clock cycle
3) stops the calculation when n sequence elements have been calculated
4) receives the sequence parameters from the PCIe interface
5) sends the generated sequence out on the PCIe interface

Figure 6 illustrates the implementation of the first three points listed above. When the 'Activate' input is pulled high, the 32-bit counter is reset to zero and the 'Done' flag is pulled low. On every clock edge, the counter increments by M to reflect the new groupOffset value. When the offset becomes greater than $n + 6M$, the *done* flag is raised and the counter is reset. This threshold value was chosen because there are six pipeline register stages between the control unit and the outputs of the arithmetic units. The control unit has to wait six clock cycles after sending the last *groupOffset* value before it can raise the *done* flag. Each clock cycle increments the counter by M, hence the threshold $n + 6M$.

The *valid* flag is kept high as long as the *activate* flag stays high and the *done* flag is still low, i.e. the inputs stay valid until all calculations are done or the system is deactivated.

F. Example calculation

The process outlined above can be illustrated by means of a simple example. Consider a scenario with parameters as in Table II. In this case, the generated sequence should be $\{0, 2, 4, 6\}$. Before computation starts, the values 0 and 2 are placed on all the arithmetic units' a_1 and d inputs, respectively. The value 4 is placed on the control unit's n input. To start the calculation, the control unit's *activate* input is pulled high and the system is clocked. On the first clock edge, the *groupOffset* output will be 0 since the counter was reset. This value, along with a raised *valid* line, is clocked into the offset blocks. On the next clock, the two offset blocks will output values of 1 and 2 at their respective i outputs. These values are clocked into the i inputs of the sequence block. The subsequent three

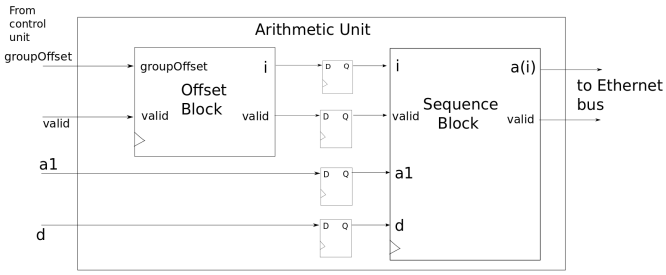

Fig. 5. Combined Arithmetic Unit

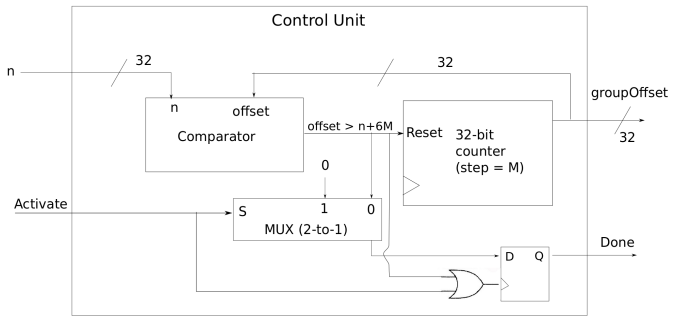

Fig. 6. Control Unit: Counter and flags

clock edges will drive the actual calculation, after which the two sequence blocks will present 0 and 2 at their respective $a(i)$ outputs. Upon detecting the raised *valid* flag, the system sends the values to the Ethernet controller for transmission.

Meanwhile, the counter in the control unit has counted up by $M = 2$, presenting a value of 2 at its *groupOffset* output. This is clocked into the offset blocks, which leads to values of 3 and 4 at their i outputs. On subsequent clock edges, the sequence blocks repeat the calculation with the incremented i values, resulting $a(i)$ outputs of 4 and 6, which are sent to the Ethernet controller.

When the control unit senses that *groupOffset* is larger than $4 + (6 \times 2)$, it raises the *done* flag to signal completion.

G. FPGA-based implementation

It was decided to implement the above designs on a high-end FPGA. The FPGA will include hardware multipliers (the more the better) capable of multiplying operands of at least 32 bits wide. This allows a multiply operation to happen in a single clock cycle. One or more phase-locked loops would also be useful if the clock speed needs to be reduced.

H. Communication over Ethernet

It was decided to use 100 Gbps Ethernet as the communication channel between the host and accelerator. Fiber optic cables will be used as defined in IEEE standard 802.3 [10]. Specifically, Amendment 3 of this standard is used, which specifies 100 Gbps Ethernet over multi mode fiber-optic cables (100GBASE-SR4) [11].

Some FPGA manufacturers provide off-the-shelf Ethernet controllers as soft IP cores that can be programmed onto the FPGA. For example, Altera provides an Ethernet MAC device as one of its MegaCore IP products [12]. A similar approach will be used in this design.

Altera's 100 G Ethernet core has a client-facing bus width of 320 bits (i.e. the switching fabric can send 320 bits to the Ethernet controller in a single transfer) [12]. To meet the 100 Gbps data rate, these transfers must be clocked at 312.5 MHz ($320 \times 312.5 \times 10^6 bps = 100 Gbps$).

A 320-bit bus can send five 64-bit values in a single transfer. At 312.5 MHz, five arithmetic units would saturate the 100 Gbps Ethernet connection. There is no point in having more than five arithmetic units on the accelerator if 100 G Ethernet is the bottleneck. This constraint means that buffering is not required on the accelerator. Sequence output values from the five arithmetic units are combined into a 320-bit bus and connected to the Ethernet controller directly, as shown in figure 1.

TABLE II
VALUES FOR SIMPLE EXAMPLE

Symbol	Value	Comments
M	2	Accelerator has 2 arithmetic units
a_1	0	Value of first element
d	2	Constant difference
n	4	Generate a sequence with 4 elements

IV. PROTOTYPE IMPLEMENTATION METHODOLOGY

A prototype was implemented in Verilog on the Digilent Nexys 3 development board [13] containing a Xilinx Spartan 6 FPGA [14]. The implementation largely followed the conceptual design. The sequence and offset blocks of the arithmetic unit were implemented as Verilog modules, as well as the control unit. There are 20 parallel arithmetic units on the prototype, each generating a 64-bit result. We found the number of arithmetic units to be limited by the MUXCY (carry chain) units available on the Spartan 6's logic blocks (SLICEM and SLICEL). These MUXCY blocks are used for fast lookahead carry inside the adders and multipliers [15]. At 20 parallel streams, 87% of the available carry chains were used.

The system was clocked at 100 MHz. A calculation throughput of 128 Gbps is therefore expected. The pipeline stages were implemented as in the conceptual design.

The prototype did differ from the conceptual design in the following respects:

A. UART communication

Time constraints prevented us from implementing an Ethernet connection. Instead, UART over USB was used to send the generated sequence elements back to the host. The bytes of each 64-bit value are transmitted in little-endian order (least significant byte first). Our UART module communicates at 115,200 bps, which is considerably slower than 100 Gbps Ethernet.

B. Output buffers

Since the UART link to the host is much slower than the calculation throughput, the generated values were buffered on the FPGA and sent out over UART after all calculations were complete. A 1 Kb memory block was connected to the output of each arithmetic unit as shown in figure 7. The control unit contains a register for the current memory address. On every clock cycle, the 20 calculated values are saved into the same address on each parallel memory block. The memory address is then incremented for the next group of M values. The memory address is placed on the bus labeled "Control Bus" in figure 7. From there, it advances through the pipeline

along with the other control signals such as the *valid* line, *groupOffset* value, etc. as discussed in section III.

V. Prototype Experimentation

After developing the hardware accelerator prototype, it had to be tested to determine its correctness and performance. It was decided to time the prototype's calculation for sequences of various lengths and compare it to the execution times of the golden measure. This involved developing a golden measure program as well as a method to accurately time the processing of the prototype. In order to obtain comparable results, the input parameters shown in Table III were set to be constant for all tests with only the sequence length being varied.

A. Golden measure

For the golden measure, a sequential C program was written to generate arithmetic sequences based on user input parameters and time how long a CPU takes to generate the sequence. The timing was done using special timer modules to accurately measure the processing time only. The golden measure testing was done on a computer with an 2.2 GHz Intel Core i5-5200U [16] CPU and 4 GB of RAM.

B. Prototype testing

The correctness of the sequence elements was verified by sending the data over UART to a computer and inspecting the data received over UART by comparing a sample set of them to the corresponding values from the golden measure C program.

The sequence generation runtimes for the prototype were obtained by routing the control unit's *done* signal to one of the Nexys 3 PMOD pins and probing it with an oscilloscope. A falling edge was detected on this pin when calculation starts, and a rising edge appeared upon completion. The distance between these edges were then measured on an oscilloscope trace and the elapsed time could then be accurately measured so that it could be compared to the golden measure.

An example of one of the oscilloscope traces can be seen in Figure 8. In that case, the oscilloscope was set to $4.3\ \mu s$ per horizontal division. The distance between the two *done* line edges is therefore $8.6\ \mu s$, corresponding to a sequence of 1 million elements as listed in table IV.

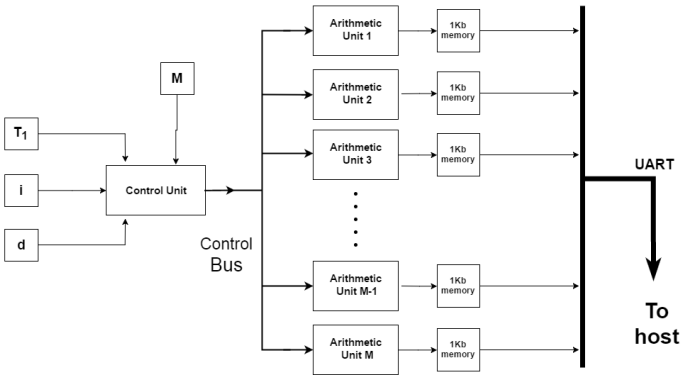

Fig. 7. Output buffer memory blocks

TABLE III
Symbols used in the design description

Parameter	Value
M (number of arithmetic units)	20
a_1 (first term)	255
d (common difference)	10

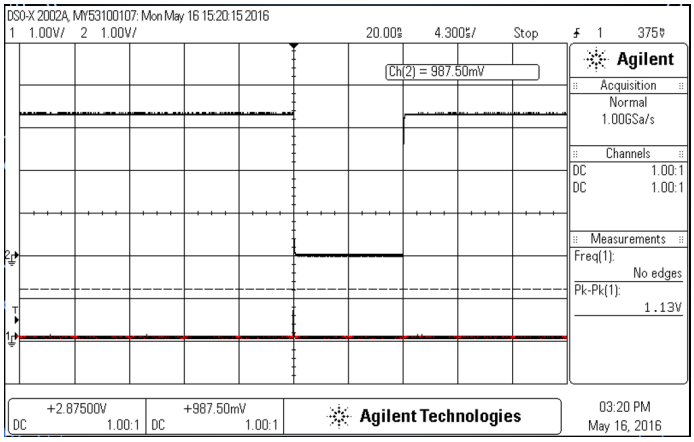

Fig. 8. Sample screen capture of oscilloscope trace with 20 ns horizontal divisions

VI. Testing Results

The resulting golden measure runtimes along with the prototype runtimes for generating arithmetic sequences of different lengths (based on the input parameters in Table III) are shown in Table IV. It can be seen that the runtimes of the prototype were faster, resulting in an increasing speed-up being obtained for increasing lengths of sequences. These results demonstrate that the digital accelerator prototype works successfully.

VII. Conclusion

The conceptual design of a digital hardware accelerator to generate arithmetic sequences has been detailed in this paper. Such a device is capable of contributing performance increases to applications which use arithmetic sequences in their algorithms. A prototype hardware accelerator was then implemented using a Nexys 3 evaluation board with a Xilinx Spartan 6 FPGA. The prototype implementation was based on the principles of the conceptual hardware accelerator design. In order to test the prototype, a golden measure was developed in the form of a sequential CPU based C program. The sequence generation runtimes of the two were then measured and compared with the prototype producing a significant speed-up compared to the golden measure. This demonstrated that the hardware accelerator prototype was successful and stands as a working proof of concept of the original conceptual design.

VIII. Future Work and Prototype Enhancements

In order to further enhance the developed prototype the following could be investigated:

TABLE IV
TEST RESULTS

Number of elements	Golden measure [s]	Prototype [s]	Speed-up
10	2.70×10^{-7}	1.24×10^{-7}	1.90
100	6.50×10^{-7}	1.91×10^{-7}	3.40
1,000	2.80×10^{-6}	6.40×10^{-7}	4.38
10,000	3.50×10^{-5}	5.10×10^{-6}	6.81
100,000	2.80×10^{-4}	1.73×10^{-5}	167.06
1,000,000	5.40×10^{-3}	8.60×10^{-6}	295.34
10,000,000	2.73×10^{-2}	1.94×10^{-5}	1,405.67
100,000,000	2.76×10^{-1}	2.89×10^{-5}	9,532.87

- Using a larger FPGA to accommodate more arithmetic modules, bits per sequence element and memory blocks
- Implementing high speed parallel output data using 100 Gb Ethernet
- Extending the design to work with floating point and negative numbers

REFERENCES

[1] M. Clemens and G. Clemens, *Barron's Regents Exams and Answers - Algebra2 Trigonometry*. Barron's Educational Series, Inc.

[2] W. Zhen, H. Xia, L. Yu-Xia, and S. Xiao-Na, "A new image encryption algorithm based on the fractional-order hyperchaotic lorenz system," *Chinese Physics B*, vol. 22, pp. 010 504–1–010 504–7, 2013.

[3] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[4] in *Advances in Multimedia Information Processing - PCM 2006: 7th Pacific Rim Conference on Multimedia, Hangzhou, China*, Y. R. Q. H. Yueting Zhuang, Shiqiang Yang, Ed.

[5] in *China Satellite Navigation Conference (CSNC) 2015 Proceedings*, E. a. Jiadong Sun, Jingnan Liu, Ed.

[6] E. a. Guy de Tr, Przemysaw Grzegorzewski, *Challenging Problems and Solutions in Intelligent Systems*.

[7] E. a. Thomas Leo McCluskey, Apostolos Kotsialos, *Autonomic Road Transport Support Systems*.

[8] in *XII Mediterranean Conference on Medical and Biological Engineering and Computing 2010: MEDICON 2010, Chalkidiki, Greece*, P. D. B. Nicholas Pallikarakis, Ed.

[9] "What is the difference between sdram, ddr1, ddr2, ddr3 and ddr4?" http://www.transcend-info.com/Support/FAQ-296, Transcend.

[10] "802.3-2015 - IEEE Standard for Ethernet," http://standards.ieee.org/findstds/standard/802.3-2015.html, IEEE Standards Association, 2015.

[11] "802.3bm-2015 - ieee standard for ethernet - amendment 3," https://standards.ieee.org/findstds/standard/802.3bm-2015.html, IEEE Standards Association, 2015.

[12] "40- and 100-gbps ethernet mac and phy megacore function user guide," https://www.altera.com/en_US/pdfs/literature/ug/ug_40_100gbe.pdf, Altera Corporation.

[13] "Nexys 3 Reference Manual," https://reference.digilentinc.com/nexys/nexys3/refmanual, Digilent Inc.

[14] "Spartan 6 Family Overview," http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf, Xilinx, Inc.

[15] "Spartan-6 FPGA Configurable Logic Block," http://xilinx.eetrend.com/files-eetrend-xilinx/download/201001/410-359-ug384.pdf, Xilinx, Inc.

[16] "Intel core i5-5200u processor (3m cache, up to 2.70 ghz) specifications," http://ark.intel.com/products/85212/Intel-Core-i5-5200U-Processor-3M-Cache-up-to-2_70-GHz, (Accessed on 05/21/2016).