FPGA™ World

12th FPGAworld CONFERENCE

SEPTEMBER 8th-10th, 2015
STOCKHOLM + COPENHAGEN

EDITORS

Lennart Lindh, Vincent J. Mooney III, Ketil Røed, David Källberg,
Santiago de Pablo, Mohamed Shalan, Johnny Öberg and Peeter Ellervee.

ACADEMIC PROCEEDINGS 2015

The FPGAworld Conference addresses all aspects of digital and hardware/software system engineering on FPGA technology. It is a discussion and network forum for researchers and engineers working on industrial and research projects, state-of-the-art investigations, development and applications. The proceedings contain the academic presentations, a separate report contains also the industrial presentations; for more information see www.fpgaworld.com.

SPONSORS

aktuel elektronik

DTU

ELEKTRONIK TIDNINGEN

EXOSTIV LABS

acm In-Cooperation

Proceedings of FPGAworld 2015
Index

FPGAworld @ Stockholm

FPGAworld @ Copenhagen

Academic General Chairman's Message

Welcome to the 12th edition of the FPGAworld conference! With the dramatic shrinkage of chip technologies continuing to advance, capabilities of FPGAs are only increasing their expansion into areas traditionally only covered by ASICs. We cordially welcome you to this premier event mixing industrial and academic perspectives in an open forum of debate and inclusion.

As in previous years, this year the conference is again taking place in two locations, Copenhagen (Denmark) and Stockholm (Sweden). Earlier editions of FPGAworld have been organized in Finland (Tampere), India (Udaipur), Germany (Munich) and Sweden (Lund).

The FPGAworld conference has academic reviewed papers, industrial reviewed papers, keynote addresses and product presentations, resulting in a fertile mix of presentations. FPGAworld also has in both locations exhibitors as well as an occasional tutorials. This aims to cover an increase in demand from the academic and industrial participants for FPGA knowledge.

Please check out the website (http://www.fpgaworld.com) for more information about FPGAworld 2015. In addition, you may contact David or Mia (david@fpgaworld.com, mia@fpgaworld.com) for more information about product presentations, web advertisements, sponsoring, tutorials and exhibits. For academic and industrial presentations, please see the FPGAworld website for more information.

FPGAworld already now opens to receive suggestions for next year's conference in September 2016. We are interested in suggested keynote speakers, web advertisements (year around), sponsoring, technical papers, product presentations, student projects, exhibits and tutorials. Submissions are open to students, academics and industrial professionals. Together we can help make the FPGAworld conference exceed our best expectations!

The organizers of FPGAworld would like to thank all contributors; we hope that attendees will truly enjoy and benefit from the FPGAworld conference.

Sincerely,

Vincent John Mooney III

2015 ACADEMIC ORGANIZATION

General Academic Chair
Vincent J. Mooney III, Georgia Institute of Technology, USA

Academic Program Chair
Ketil Røed, University of Oslo, Norway

Academic Publication Chair
Santiago de Pablo, University of Valladolid, Spain

Academic Publicity Chair
Mohamed Shalan, American University of Cairo, Egypt

Steering Committee Members
Peeter Ellervee, Tallin University of Technology, Estonia
Lennart Lindh, Jönköping University, Sweden
Johnny Öberg, KTH Royal Institute of Technology, Sweden

Academic Programme Committee Members
Johan Alme, Bergen University College, Norway
Peeter Ellervee, Tallin University of Technology, Estonia
Timo D. Hämäläinen, Tampere University of Technology, Finland
Reiner Hartenstein, TU Kaiserslautern, Germany
Leandro Soares Indrusiak, University of York, United Kingdom
Paul Kolin, Indian Institute of Technology, IIT Delhi, India
Pramote Kuacharoen, National Institute of Development Administration, Thailand
Anshul Kumar, Indian Institute of Technology, IIT Delhi, India
Shashi Kumar, Jönköping University, Sweden
Vincent J. Mooney III, Georgia Institute of Technology, USA
Johnny Öberg, KTH Royal Institute of Technology, Sweden
Santiago de Pablo, University of Valladolid, Spain
Adam Postula, University of Queensland, Australia
Attiq Ur Rehman, University of Bergen, Norway
Ketil Røed, University, Oslo, Norway
Erno Salminen, Tampere University of Technology, Finland
Mohamed Shalan, American University of Cairo, Egypt

2015 GENERAL AND INDUSTRIAL ORGANIZATION

Industrial Program Chair
Lennart Lindh, FPGAworld, Sweden

Industrial Program Committee Members
Solfrid Hasund, Bergen University College
Kim Petersén, HDC, Sweden
Mickael Unnebäck, ORSoC, Sweden
Fredrik Lång, EBV, Sweden
Niclas Jansson, BitSim, Sweden
Göran Bilski, Xilinx, Sweden
Per Henricsson, Elektroniktidningen, Sweden
Espen Tallaksen, Bitvis, Norway
Tommy Klevin, ÅF, Sweden
Tryggve Mathiesen, InformASIC, Sweden
Fredrik Kjellberg, Net Insight, Sweden
Daniel Stackenäs, Altera, Sweden
Stefan Sjöholm, Realfast, Sweden
Torbjorn Soderlund, Xilinx, Sweden
Anders Enggaard, Axcon, Denmark
Doug Amos, Synopsys, UK
Guido Schreiner, The Mathworks, Germany
Stig Kalmo, Engineering College of Aarhus, Denmark
Hichem Belhadj, Microsemi, USA
Rolf Sylvester-Hvid, Aktuell Elektronik, Denmark
Tony Eriksson, Future Electronics, Sweden
Ann-Luise Vestrup Kristensen, SILICA, Denmark
Mircea Alexandru Dabacan, Digilent Ro, Romania
Andreas Engberg, ConMed, USA
Abbas Bigdeli, Nicta, Australia
Siegfried Weigert, ibw, Germany
Basavaraj Hooli, FPGAworld, India
Clint Cole, Digilent, USA
Gagan Puri, Coreel, India
Sudarshan Natu, Symphony, India
Udayprakash Raghunath Singh, SPSU, India
Yehoshua Shoshan, Innofour, Sweden
Hai Migdal, Gidel, Israel
Thorsten Trenz, Trenz Electronic GmbH, Germany
Gerd Prillwitz, Ansys, Germany
Juergen Kessler, BlackForest EDA, Germany
Willem groter, HDL Works, Netherlands
Maurizio Casti Thales Group, Italy
Andreas Schwarztrauber, MSC, Germany
Ben Liu, Digilent, Taiwan
Mattias Karlsson, Saab, Sweden
Antti Innamaa, Synopsys, Finland

Jacky Cheng, Huafan Tech, China
Mikko Rasa, ARROW, Finland
Thony Johansson, ÅF, Sweden
Henrik Eeckenhaut, Sigasi, Belgium
Mike Dini, Dini Group, USA
Jan Viktorin, RehiveTech, Czech Republic
Svend Modtgard, Wdiag, Germany
Soren Manicus, TekPartner, Denmark
Rune Domsten, IndesmaTech, Denmark

General Manager
Lennart Lindh, FPGAworld, Sweden

Registration, Publicity and Expo Manager
David Källberg, FPGAworld, Sweden

Administration and Finance Manager
Mia Lindh, FPGAworld, Sweden

Sales Manager
Ove Boström, FPGAworld, Sweden

Sponsors, Exhibitors and Product Presenters

ÅF, Sweden
AGSTU AB, Sweden
Aktuel Elektronik, Denmark
Altera, USA
Arrow Electronics, USA
DTU, Denmark
Elektroniktidningen, Sweden
Exostiv Labs, Belgium
InnoFour, Netherlands
KTH, Sweden
Linear Technology, USA
Microsemi , USA
Silica, USA
Synective Labs, Sweden
The Dini Group, USA
Xilinx, USA

In cooperation with ACM

FPGAworld 2015 @ Stockholm

Frösundaleden 2A
169 70 Solna, Sweden

Conference Program

08:30 Registration.

09:00 Conference opening.
Thony Johansson, ÅF* and *Lennart Lindh, FPGAworld.

09:15 Key Note Session.
Next Generation Massively Parallel VLSI Architectures and Design Methods
Ahmed Hemani, Professor, KTH, Stockholm, Sweden.

10:00 Coffee break & Exhibition, sponsor EXOSTIV LABS.

10:30 Parallel Sessions.

12:30 Lunch break & Exhibition.

13:30 Parallel Sessions.

15:30 Coffee break & Exhibition, sponsor ALTERA.

16:00 Panel Discussion.
What will be the impact of Intel on Altera?
Any reason to be frightened? Xilinx next?
Session Moderator: Mike Dini, Dini Group, USA.

The exhibition will be open during the day.
Coffee will be served in the exhibition area.

§ Next Generation Massively Parallel VLSI Architectures and Design Methods

Ahmed Hemani, Professor, KTH, Stockholm, Sweden.

Room: Renen.

ITRS for the mobile category has challenged the VLSI Design community to come up with solutions by 2020 to provide 1000X improvement in performance with 120% increase in power budget and no increase in the design team size to cope with a 10X increase in design complexity. We propose a solution based on Coarse Grain Reconfigurable Fabric for computation and storage called DRRA – Dynamically Reconfigurable Resource Array. This fabric provides a near ASIC performance and yet retains programmability. This fabric enables dataflow graphs along with their control to be implemented in arbitrary degree of parallelism in a true hardware fashion. The DRRA fabric also comes with a novel System-level to GDSII design flow based on a concept called SiLego. SiLego is based on a grid based design and use of large grain building blocks called SiLego instead of the prevalent standard cells. The SiLego blocks snap fit to compose a GDSII macro and provide a predictable micro-architecture level physical design target to empower true high-level and system-level syntheses.

FPGA™ world

Room: Renen
Sessions A1-A2-A3-A4
Session Chair: Roger Ericsson
ÅF, Sweden

§ Automotive ADAS feature saves lifes with flexible HW (FPGA etc.) – practical implementation reference

Automobile accidents are currently killing 1.25 million people per year worldwide and that figure is rising. The new mantra is "avoiding collisions."

Consumer surveys show that people want safety features in cars, that these safety features sell cars, and that interest in ADAS features is actually higher than for any other electronics-based automotive category.

FPGA enables ADAS systems, saving lifes in the traffic. A reflection of present system

Presenter: Tryggve Mathiesen, *Qamcom R&T AB, Sweden*.

§ High-speed FPGA-based stereovision system - a success story

The presentation concerns an industrial stereovision system based on the Xilinx Zynq 7020 FPGA SoC which combines Linux on ARM and with surrounding reconfigurable logic that works as a hardware accelerator. The system is capable of working in real-time with ~50 FPS in VGA and above 65 FPS in lower resolutions. The architecture includes sensor control logic, demosaicer, rectifier and stereovision cores

Presenter: Rafal Kapela, Ph.D., *Antmicro Ltd, Poznan University of Technology, Poland*.

§ Real-time operating system, hardware or software?

One important purpose of a real-time operating system is to present a result in right time. The question is how much time can be saved using a real-time hardware OS accelerator in hardware, compared to today's software kernel in a FPGA device with NIOS processor? The hardware based realtime kernel is designed in VHDL with a thin software device drivers. The software kernel used is FreeRTOS and the hardware kernel used is Sierra with exact same hardware architecture.

Presenter: André Norberg, *AGSTU, Sweden*.

§ The critically missing VHDL testbench feature - Finally a structured approach

Verification is 51% of total FPGA development time. Even more for control or protocol oriented design. Ignore this and you will iterate forever in the lab and suffer from bad product quality. The root cause is corner cases triggered by certain input and FSM combinations. Hitting a given corner case in normal simulation is very unlikely, - but in the final product it will hit you hard. A structured VHDL approach to verifying such corner cases has not been commonly available. UVVM (Universal VHDL Verif. Meth.) changes this picture completely.

Presenter: Espen Tallaksen, *Bitvis, Norway*.

FPGA™ world

Room: Räven
Sessions B1-B2-B3-B4
Session Chair: Johnny Öberg
KTH, Sweden

§ Analysis of FPGA-Based Reconfiguration Methods for Mobile and Embedded Applications

As mobile and handheld devices are gaining popularity, many applications have found their way into these devices. Apart from optimized hardware-software architectures, new techniques and design methodologies are needed to support applications running on mobile systems. FPGA-based dynamic reconfigurable systems are currently the most feasible option to deliver embedded applications that have stringent requirements. There are different methods of reconfiguring the hardware on chip dynamically. Selecting a specific reconfiguration method and designing the corresponding hardware architectures for an application are important and challenging tasks in reconfigurable computing systems. In this work, we investigate different FPGA-based reconfiguration methods; study their features, advantages and disadvantages; and analyze the reconfiguration time and space overhead for each method.

Presenter: Darshika Gimhani Perera, *University of Colorado, USA*.

§ An EG-LDPC Based 2-Dimensional Error Correcting Code for Mitigating MBUs of SRAM Memories

In this paper, a 2-D error correction code architecture based on EG-LDPC and single parity check (SPC) code is proposed as a solution to MBU problem of SRAM memories. EG-LDPC codes have better multiple error correction and detection capabilities than conventional codes and they have low complexity decoders. Therefore, they are suitable for fault tolerant memory applications. The proposed architecture uses (15,7,5) EG-LDPC as row encoding and SPC code for column encoding. In order to minimize decoding complexity of 2D structure, a standard array decoder is utilized. The investigated architecture is compared with previously proposed Matrix code method. The proposed architecture is able provide over 95% error correction coverage up to 4 errors and a significant 100% error detection up to 12 bit errors. In terms of MTTFs, the proposed approach achieves 63% improvement over Matrix codes at fault rates of 10-4 and 10-5. Matrix codes and the proposed architecture are implemented using Xilinx XC6SLX16 FPGA and comparison results in term of implementation complexity are provided.

Presenter: Enver Çavuş, *Yıldırım Beyazıt University*, *Turkey*.

§ High Level Synthesis Based Hardware Accelerator Design for Processing SQL Queries

About three exabytes of data is created and stored in databases each day, and this number is doubling approximately every forty months. Querying this enormous amount of data has been a challenge and new methods have been actively researched.

In this paper, we present hardware accelerators which are designed to speed up database analytics for in- memory databases. Unlike traditional hardware accelerator designs, our hardware accelerators are composed using High Level Synthesis (HLS), which enables high level descriptions of functionality such as data filtering, sorting, equijoins to be targeted directly into RTL. We have simulated TPC-H benchmark queries using Xilinx Vivado HLS managed in our custom simulation software framework. Our results have demonstrated the capabilities of HLS in database accelera- tion domain; such that the 200MHz FPGA accelerator can provide two orders of magnitude performance improvement compared to PostgreSQL based full software implementation running on a modern multicore system.

Presenter: Gorker Alp Malazgirt, *Bogazici University, Turkey*.

§ Comparison of Predictive-Corrective Video Coding Filters for Real-Time FPGA-based Lossless Compression in MultiCamera Systems

Combining multiple cameras in a bigger multi- camera system give the opportunity to realize novel concepts (e.g. omnidirectional video, view interpolation) in real-time. The better the quality, the more data that is needed to be captured. As more data has a direct impact on storage space and communication bandwidth, it is preferable to reduce the load by compressing the size. This cannot come at the expense of latency, because the main requirement is real-time data processing for multi-camera video applications. Also, all the image details need to be preserved for improving the computational usage in a later stage. Therefore, this research is focused on predictive-corrective coding filters with entropy encoding (i.e. Huffman coding) and apply these on the raw image sensor data to compress the huge amount of data in a lossless manner. This technique does not need framebuffers, nor does it introduce any additional latency. At maximum, there will be some line-based latency, in order to combine multiple compressed pixels in one communication package. It has a lower compression factor as lossy image compression algorithms, but it does not remove human invisible image features that are crucial in disparity calculations, matching, video stitching and 3D model synthesis. This paper compares various existing predictive-corrective coding filters after they have been optimized to work on raw sensor data with a color filter array (i.e. Bayer pattern). The intention is to develop an efficient implementation for System-on-Chip (SoC) architectures to improve the computational multi-camera systems.

Presenter: Yimu Wang, *Hasselt University, Belgium*.

Analysis of FPGA-Based Reconfiguration Methods for Mobile and Embedded Applications

Darshika G. Perera
University of Colorado, Colorado Springs
1420 Austin Bluffs Parkway
Colorado Springs, CO 80918, USA
+1-719-255-3404
darshika.perera@uccs.edu

ABSTRACT
As mobile and handheld devices are gaining popularity, many applications have found their way into these devices. Apart from optimized hardware-software architectures, new techniques and design methodologies are needed to support applications running on mobile systems. FPGA-based dynamic reconfigurable systems are currently the most feasible option to deliver embedded applications that have stringent requirements. There are different methods of reconfiguring the hardware on chip dynamically. Selecting a specific reconfiguration method and designing the corresponding hardware architectures for an application are important and challenging tasks in reconfigurable computing systems. In this work, we investigate different FPGA-based reconfiguration methods; study their features, advantages and disadvantages; and analyze the reconfiguration time and space overhead for each method.

Keywords
Reconfiguration methods, FPGAs, dynamic reconfiguration, mobile and embedded devices.

1. INTRODUCTION
With the advancement of mobile and embedded computing, a wide variety of applications are becoming common on these devices. However, these devices have numerous challenges including stringent area and power limitations, reduced cost and time-to-market requirements, and increased speed performance. In [23], an analysis of single-chip hardware support for mobile and embedded applications was carried out. These analyses illustrated that FPGA-based reconfigurable hardware provides a flexible and area efficient computing platform under the constraints associated with mobile and embedded devices. Multiple applications can be processed on a single chip, by dynamically reconfiguring the hardware on chip from one application to another as needed.

There are different FPGA-based reconfiguration methods, including single context, multi context, partial reconfiguration, and MultiBoot. These methods allow the dynamic reconfiguration of the hardware on chip to perform a variety of operations during the runtime life of an application without human intervention. Each of the FPGA-based reconfiguration methods has its own special features, advantages, and disadvantages. Selecting a specific reconfiguration method and designing the corresponding hardware architectures for an application are important and challenging tasks in reconfigurable computing systems. In this work, we investigate different FPGA-based reconfiguration methods and study their features, advantages and disadvantages. Analysis of reconfiguration time and space overhead is also carried out for each reconfiguration method.

FPGAWorld'15, September 8-10, Stockholm and Copenhagen.

2. FEATURES, ADVANTAGES, AND DISADVANTAGES OF FPGA-BASED RECONFIGURATION METHODS
In this section, we analyze, discuss, and present the features, advantages, and disadvantages of FPGA-based reconfiguration methods. The following terminologies are used within the context of reconfiguration methods.

- **Configuration bitstream:** Also known as a context, is a binary file that sets all of the FPGA's programmable bit locations to configure the logic blocks and routing resources. Initially, the configuration bitstreams are stored in an external non-volatile memory, and then downloaded to the chip and used to configure the chip's hardware circuitry.
- **Inactive context:** After a configuration bitstream is downloaded and stored in on-chip memory, it is called an inactive context. These inactive contexts can be kept on-chip and used repeatedly to reconfigure the chip as needed.
- **Active context:** After a configuration bitstream is downloaded to the chip and used to re/configure the chip, it is called an active context. Active context does not necessarily mean that it is running

2.1 Single Context
Traditionally, most of the applications running on an FPGA use single context, which allows only one full configuration bitstream for the entire chip to be downloaded to the chip at a time [6],[11]. With single context, a full configuration bitstream is downloaded to all the programmable bit locations of the chip at a time and the entire chip is configured to the appropriate hardware circuitry. As a result, the entire chip has to be reprogrammed even for the

smallest changes in the design. This method does not require extra hardware on chip for reconfiguration, thus all the resources on chip can be utilized for logic and routing of the design. Although this simplifies the reconfiguration hardware, it incurs higher reconfiguration time overhead when only a small part of the design needs to be changed [6]. The reconfiguration time overhead is the time required to download and change the configuration.

Single context requires an external controller [21] to control the configuration flow, since the controller performing the re/configuration (deciding which configuration bitstream to be downloaded next) has to be active during the whole process. An external controller, whose primary purpose might be to perform other tasks, can be used to coordinate the downloading of the configuration bitstream into the programmable bit locations on the chip [21]. After the execution of one application with one configuration bitstream, the external controller determines which of several configuration bitstreams (stored in the external memory) should be downloaded next and downloads the next configuration bitstream and reconfigures the chip. A disadvantage of external downloading and reconfiguring is that it incurs higher reconfiguration time overhead, which can be milliseconds [30] or more, compared to having configuration bitstreams stored on chip.

2.2 Multi Context

Unlike single context, multi context allows multiple independent configuration bitstreams, some as active contexts and some as inactive contexts, to be downloaded initially to the chip simultaneously [28]. Hence, multi context allows more than one active context to be executed in parallel on the same chip, and also allows several inactive contexts to be stored in on-chip configuration memory, which are the major advantages of this method.

In this case, the on-chip configuration memory (e.g., with *n* inactive contexts) is distributed around the chip, such that each configuration memory cell, associated with a programmable bit location, has *n* memory bits of storage for each of the *n* inactive context [5],[28]. If the next required inactive context(s) is (are) in configuration memory, the entire chip can be reconfigured in a single cycle [11],[25]. Therefore, in contrast to the conventional single context method, multi context can hold several different inactive contexts in on-chip configuration memory and can switch between the contexts on demand quickly, since no access to external memory is required. Hence, another compelling advantage of multi context is that it allows extremely fast reconfiguration within an order of nanoseconds [10],[25].

However, the ability to perform a reconfiguration of the full chip in a single cycle can be a disadvantage. In this case, all the programmable bit locations are loaded from the configuration memory simultaneously, and potentially the majority of the programmable bit locations may be changed from 0 to 1 or vice versa [11]. Switching the configuration bits in many locations in a single cycle, increases dynamic power, which could violate system power constraints [11],[28].

Another drawback of multi context is that it requires extra hardware on-chip for reconfiguration and for on-chip memory to hold inactive contexts, which occupy valuable area of the chip that could otherwise be used for logic and routing [28]. Therefore, the active area (for logic and routing of the design) of a chip using multi context is less than that of a chip using single context of the same area [7],[11]. For instance, a chip using multi context holding four inactive contexts in configuration memory has only 80 percent of the active area (for logic and routing) of that using single context [7].

Similar to single context, multi context also requires an external controller to control the configuration flow [21]. However, if there is a possibility to download and hold all the required contexts (active and inactive) on chip with the initial downloading, we can eliminate the need to interface with an external controller. In this case, when computation of one active context is near completion, the active context itself can trigger the on-chip configuration memory to reconfigure that part of the chip, which occupied by the active context, with the next required context [27]. This allows self-reconfiguration, which is another advantage of the multi context method.

In multi context, the external controller can also be used to pre-fetch configuration bitstreams to the chip from external memory [10]. Hence, another benefit of the multi context method is that it allows background loading of configuration bitstreams while the active contexts are still operational [7],[25],[28]. In addition, multi context allows reconfiguring parts of the chip, which require modification, while the rest of the active contexts are still operational, overlapping computation with reconfiguration [6],[11]. Pre-fetching and reconfiguring parts of the chip, potentially has the benefit of hiding some of the reconfiguration time overhead.

With multi context, it is possible to communicate signals between the current context and the next context (after reconfiguration), using registers [4],[17]. However, proper connections to the registers are not guaranteed after reconfiguration, since electrical signals of the registers might be affected during reconfiguration, which can cause glitches or loss of information [20].

2.3 Partial Reconfiguration

In some cases, only some parts of the design require modification. Partial reconfiguration, as indicated by its name, allows reconfiguring a portion of a chip that requires modification [6],[11],[19]. This has two advantages. One is that, similar to multi context, some parts of the chip are reconfigured while the remaining parts are in operation, thus overlapping computation with reconfiguration. Another is that, the reconfiguration can be carried out while interfacing with the remaining parts of the design.

With partial reconfiguration, first, the FPGA is configured by loading an initial full configuration bitstream for the entire chip upon power-up [8],[32]. After the FPGA is fully configured and operational, multiple partial bitstreams can be downloaded simultaneously to the chip, and specific regions of the chip can be reprogrammed with new functionality “without compromising the integrity of the applications” running in the remainder of the chip [8],[32]. Partial bitstreams are used to reconfigure only selective parts of the chip. Figure 1, which is modified from [32], depicts the basic premise of partial reconfiguration. During the design and implementation process, the logic in the reconfigurable hardware design is divided into two different parts: reconfigurable and static. As shown in Figure 1, the functions implemented in reconfigurable modules (RM), i.e. reconfigurable parts, are replaced by the contents of the partial bitstreams (.bit files), while the current static parts remain operational, completely unaffected by the reconfiguration [32].

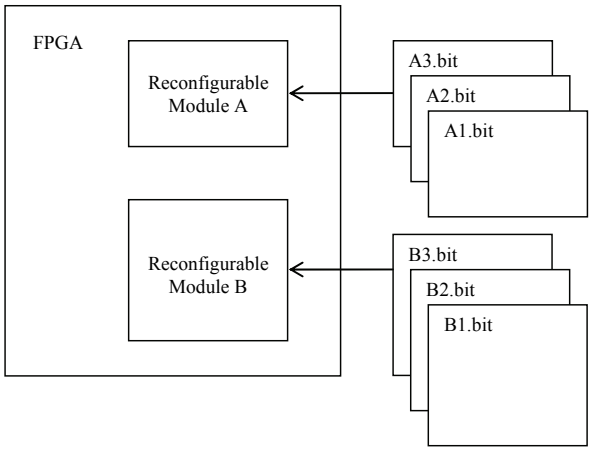


Figure 1. Basic Premise of Partial Reconfiguration [32].

In the late 2010s, partial reconfiguration tools used Bus Macros [16],[26], which ensures fixed routing resources for signals used as communication paths for reconfigurable parts, and when the parts are reconfigured [26]. With the new PlanAhead [33] tools for partial reconfiguration, Bus Macros become obsolete. Current FPGAs (such as Virtex-6 and Virtex-7) have an important feature: a "non-glitching" (or "glitchless") technology [8],[34]. Due to this feature, some static parts of the design could be in the reconfigurable regions without being affected by the act of reconfiguration itself, while the functionality of reconfigurable parts of the design is reconfigured [8]. For instance, when we partition a specific region and consider it as a reconfigurable part, some static interfacing might go through the reconfigurable part or some static logic (e.g., control logic) might exist in the partitioned region. These are overwritten with the exact program information, without affecting their functionalities [8],[36].

Since partial reconfiguration allows us to change some parts of the design without reconfiguring the entire chip, another benefit of partial reconfiguration is that it reduces the total reconfiguration time. The reconfiguration time is directly proportional to the configuration bitstream length (size of the configuration data) [15],[19]. Hence, by changing only portion of the configuration bitstream, as opposed to reconfiguring the entire chip, the total reconfiguration time is reduced to microseconds or more [14].

Another advantage of partial reconfiguration is that, after the initial downloading and configuration of the chip, it is possible to use an internal controller to download the next required configuration bitstreams [8],[32],[33]. Internal controller is either a microprocessor or routines (small state machine) programmed into the FPGA [8],[32]. Furthermore, the internal controller is often used to control the configuration flow, allowing self-reconfiguration. This eliminates the need to interface with an external controller. Apart from self-reconfiguration, having an internal controller allows quick decision making, and less time (2-3 cycles) to communicate with other peripherals as opposed to using an external controller (20-30 cycles). However, the internal controller must be kept as a static part of the design and communicate with the reconfigurable parts. This will guarantee the integrity of the control circuits during and after reconfiguration [4].

Similar to multi context, with partial reconfiguration it is also possible to pre-fetch the configuration bitstreams and to store them in on-chip Block Random Access Memory (BRAM), while the circuits are operational [3]. Hence, another benefit of partial reconfiguration is that it allows background loading of configuration bitstreams from the external memory while the circuits are still operational. This also has the benefit of hiding some of the reconfiguration time overhead, which includes downloading time and reconfiguration time. However, partial reconfiguration requires extra hardware for reconfiguration and for on-chip BRAM to hold the inactive contexts, which can be a drawback [4],[8],[32].

Before 2010, for partial reconfiguration, the extra hardware required on chip for reconfiguration was quite significant, due to Bus Macros. However, since Bus Macros have become obsolete, partial reconfiguration requires less extra hardware on chip for reconfiguration. Especially, if the full and partial bitstreams are stored in an external non-volatile memory, the only hardware required on chip for reconfiguration is the Internal Configuration Access Port (ICAP) and the Interface controller to the external memory [24]. For instance, if the memory is a Compact Flash (CF), then we can use the SystemAce Interface Controller [31]. On Virtex 6 chip, resource utilizations for ICAP [36] and the SystemAce Interface Controller are about 460 and 78 slices respectively, resulting in a total of 538 slices. In this case, the internal controller can be a state machine, which is typically quite small and takes very small space on chip. Considering the total slices, 37680, on the Virtex 6 (XC6VLX240T-1FFG1156) chip, extra hardware for partial reconfiguration occupies 1.43% of the whole chip.

2.4 MultiBoot

Similar to single context, MultiBoot is another reconfiguration method that requires full reconfiguration of the chip [12]. MultiBoot enables reconfiguration of the chip with different full configuration bitstreams stored in an external non-volatile memory [12],[13]. First, the chip is configured by the initial full bitstream from the memory, then the application running on the chip can trigger a MultiBoot event and reconfigures itself from different full bitstreams housed in the external memory [12].

Unlike single context, with MultiBoot, reconfiguration is typically done using an internal configuration controller. In this case, the user application on the chip incorporates a MultiBoot control module, which includes a small state machine (internal to the FPGA) and the ICAP [2],[29]. One advantage of having an internal controller is that it allows self-reconfiguration. Another advantage is that it helps quick decision making, which takes less time (2-3 cycles) to communicate with other peripherals as opposed to using an external controller (20-30 cycles). For instance, when deciding on the configuration bitstream to be loaded next, the decision is done on-chip, and then the internal MultiBoot controller fetches the bitstream from external memory.

During reconfiguration, the control logic on chip, i.e., MultiBoot control module (as shown in Figure 2) and the interface controller to the external memory, remains intact while the rest of chip is being reconfigured [12],[29]. Unlike partial reconfiguration, with MultiBoot, parts of the chip can not be selectively reconfigured. As a result, the entire chip (except control logic) has to be reprogrammed even for the smallest changes in the design. Also, external loading and reconfiguring the entire chip incurs higher

reconfiguration time overhead, which can be milliseconds or more [12], as compared to reconfiguring only parts of the chip.

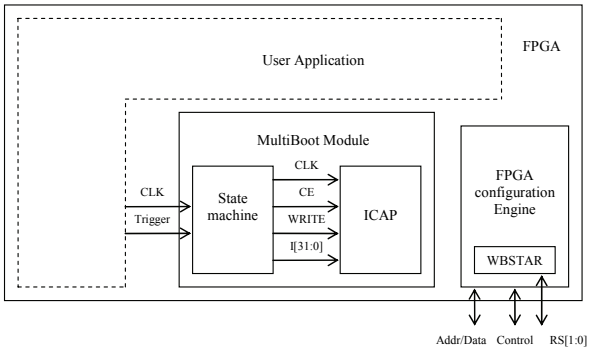


Figure 2. MultiBoot Block Diagram [12].

With MultiBoot, the only extra hardware required on chip for reconfiguration is the MultiBoot controller (state machine and ICAP) and the interface controller to the external memory. For example, if the memory is a Platform Flash PROM, then we can use the XPS multi-channel external memory controller [35]. On Virtex 6 chip, resource utilizations for ICAP [31] and the memory controller are about 460 and 701 slices respectively, resulting in a total of 1161 slices. In this case, the state machine is quite small and takes very small space on chip. Considering the total 37680 slices on the Virtex 6 (XC6VLX240T-1FFG1156) chip, the extra hardware for MultiBoot reconfiguration occupies 3.08% of the chip.

3. ANALYSIS OF RECONFIGURATION TIME AND SPACE OVERHEAD

In this section, we analyze and discuss the reconfiguration time and space overheads for different FPGA-based reconfiguration methods. Reconfiguration time and space overheads are the extra time required for reconfiguration and extra hardware required on chip for reconfiguration, respectively.

3.1 Reconfiguration Time Overhead

The reconfiguration time overhead is the time required to load and change the configuration. This has to be done whenever we want to change the application or the functionality of the hardware. Especially for dynamic reconfigurable hardware, it is necessary to ensure that the advantages of hardware acceleration are not overshadowed by the reconfiguration time overhead [6].

Reconfiguration takes a specific time, determined by the FPGA device type, bitstream size, and clock speed, but is independent of the pattern (corresponding to a specific design) of the configuration bitstream [1].

Usually, single context designs take from tens to hundreds of milliseconds to reconfigure the hardware. Single context typically uses serial interface for downloading and reconfiguration. In this case, the reconfiguration time is estimated using the full configuration bitstream and the configuration clock frequency [30], as in equation (1) below. With multi context, the reconfiguration from one context to another can be done in a single cycle [11], if the next required context is in the on-chip configuration memory. Hence, the reconfiguration time is equal to the configuration clock period. Partial reconfiguration is done through the ICAP; hence the reconfiguration time depends on the ICAP throughput and the size of the partial configuration bitstream, as in equation (2) below. According to the partial reconfiguration user guide [32], for Virtex chips, using an ICAP (100MHz and 3.2Gbps) a partial bitstream of 1358208 bits can be loaded and reconfigured in about 1358208 bits / 3200000000 bps = 424 microseconds [24]. However, depending on the reconfiguration area, partial reconfiguration might take several milliseconds to reconfigure when the bitstream is in the order of ten-millions. With MultiBoot, the reconfiguration time [2] is estimated using the full configuration bitstream, configuration clock frequency, and the bus width of the external non-volatile memory interface, as in equation (3) below. If the external memory is Platform Flash PROM, then the reconfiguration is done using a parallel programming mode such as the SelectMap protocol [12].

The following terms are used for the three equations below.

- *TotalNoOfConfigurationBits*: is the total number of configuration bits (corresponding to a full bitstream for single context and MultiBoot) for the entire chip, which is a constant for a specific chip.
- *NoOfConfigurationBits*: is the number of configuration bits for a part of the chip (corresponding to a partial bitstream for partial configuration), which varies with the size of the reconfigurable part.

$$\begin{aligned} &\mathrm{Re}configurationTime(serial) \\ &= TotalNoOfConfigurationBits * ConfigurationClkPeriod \end{aligned} \quad (1)$$

$$\begin{aligned} &\mathrm{Re}configurationTime(ICAP) \\ &= NoOfConfigurationBits / ICAPThroughput \end{aligned} \quad (2)$$

$$\begin{aligned} &\mathrm{Re}configurationTime(Parallel) \\ &= TotalNoOfConfigurationBits /(ConfigurationClkFreq * BusWidth) \end{aligned} \quad (3)$$

Xilinx Virtex-6 (XC6VLX240T) has 73859552 configuration bits [34]. With a configuration clock frequency of 100MHz, the reconfiguration time overhead to reconfigure (downloading and changing) the entire chip for single context, partial reconfiguration, and MultiBoot using the above equations are 739 ms, 23 ms, and 23 ms, respectively. In this case, for comparison purpose, we consider the worst case scenario for partial reconfiguration, where the entire chip is being reconfigured. These times are constant for single context and MultiBoot for a specific chip, whereas for partial reconfiguration, this time varies with the size of the reconfigurable module. For multi context, the reconfiguration can be done in a single cycle. For instance, with multi context on Xilinx XC4000E, the entire configuration takes 30ns [28]. In this case, the reconfiguration time remains the same (typically a single cycle) regardless of the size of the area being reconfigured. The reconfiguration for partial and MultiBoot are done using a parallel interface (both 32 bit), whereas single context is done using serial interface. Clearly, the reconfiguration time overhead using parallel mode is much lower (1/32) than that using the serial mode. With some FPGAs, it might be possible to use parallel mode for single context; thus, the reconfiguration time overhead for these three reconfiguration methods would be the same (23 ms), when reconfiguring the entire chip. However, for the applications that process large volume of data, where

processing time is significant, the reconfiguration time overhead of few milliseconds might not be an issue.

3.2 Reconfiguration Space Overhead

The reconfiguration space overhead is the extra hardware required on chip for reconfiguration and the on-chip memory required to hold inactive contexts. For some reconfiguration methods, the reconfiguration space overhead is unavoidable.

Single context designs do not require extra hardware on chip for reconfiguration, utilizing all the resources on chip (100 percent) for logic and routing of the design. With multi context, it requires extra hardware [28] on chip for reconfiguration and for on-chip memory to hold inactive contexts. In [7], a chip using multi context storing four inactive contexts in configuration memory has only 80 percent of the active area (for logic and routing of the design) of that using single context. Hence, in order to hold one inactive context (that would take up the space for the entire chip) in the configuration memory, the extra hardware required is 5% of the chip [7]. Although, the percentage of area occupied by an inactive context might change from one chip to another, in general, the extra hardware required for on-chip memory increases with the number of inactive contexts. For both partial reconfiguration and MultiBoot, the extra hardware required on chip for reconfiguration is ICAP [31] and the interface controller [36] to the external non-volatile memory. Considering the total slices, 37680, on the Virtex 6 (XC6VLX240T) chip, the extra hardware for partial reconfiguration, and MultiBoot are 1.43%, and 3.08% of the chip, respectively.

Although, at a glance, reconfiguration space overhead seems like a major drawback, since it occupies valuable real estate of the chip that could otherwise be used for logic and routing of the design, the benefits can far outweigh the drawbacks. In many cases, having reconfiguration circuitry on chip and having on-chip memory to hold the inactive contexts are essential to improve the reconfiguration process, particularly when the speed is an issue as in dynamic reconfiguration. These systems trade chip area for higher speed performance.

4. CONCLUSIONS

In this work, we investigated different FPGA-based reconfiguration methods: single context, multi context, partial reconfiguration, and MultiBoot. We studied their features, advantages and disadvantages. We also analyzed the reconfiguration time and space overhead for each method. In [22], we extended our investigations to computation models and application characteristics that could potentially benefit from FPGA-based reconfigurable hardware and discussed the how's and why's. Next, in [22], we proposed a design methodology for FPGA-based dynamic reconfigurable hardware, which gives guidelines to the embedded designers in the mapping of an application's computation models and characteristics to the most suitable reconfiguration methods based on their associated advantages and disadvantages. This design methodology can be generalized to any embedded applications.

5. REFERENCES

[1] Alfke, P., "Dynamic Reconfiguration - XAPP093", (Version 1.1), November 1997. http://www.xilinx.com/support/documentation/application_notes/xapp093.pdf.

[2] Arshak, K. and C.S. Ibala, "Using MultiBooting Approach to Reduce the FPGA Device Utilization and to Create a Redundant System", University of Limerick, Ireland.

[3] Berthelot, F., F. Nouvel, and D. Houzet, "Partial and Dynamic Reconfiguration of FPGAs: A Top Down Design Methodology for An Automatic Implementation", In *Proceedings of IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, March 2006.

[4] Chang, D., and M. Marek-Sadowska, "Partitioning Sequential Circuits on Dynamically Reconfigurable FPGAs", *IEEE Transactions on Computers*, vol.48, no.6, pp.565-578, June 1999.

[5] Chong, W., S. Ogata, M. Hariyama, and M. Kameyama, "Architecture of a Multi-Context FPGA Using Reconfigurable Context Memory", In *Proceedings of 19th IEEE International Symposium on Parallel and Distributed Processing*, April 2005.

[6] Compton, K. and S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software", *ACM Computing Surveys, (CSUR),* vol.34, no.2, pp.171-210, June 2002.

[7] Dehon, A., "DPGA Utilization and Application", In *Proceedings of 4th ACM International Symposium on Field Programmable Gate Arrays*, pp.115-121, 1996.

[8] Dye, D., "Partial Reconfiguration of Virtex FPGAs in ISE 12", Xilinx Inc., WP374 (v1.0), July, 2010.

[9] Dye, D., "Partial Reconfiguration of Xilinx FPGAs Using ISE Design Suite", WP374 (v1.1), July, 2011.

[10] Enzler, R., C. Plessl, and M. Platzner, "Co-simulation of a Hybrid Multi-Context Architecture", In *Proceedings of 3rd International Conference on Engineering of Reconfigurable Systems and Algorithms, (ERSA)*, pp.174-180, 2003.

[11] Hauck, S., and A. Dehon, "Reconfigurable Computing: The Theory and Practice of FPGA-Based Computing", Morgan Kaufmann Publishers, 2008.

[12] Hussein, J. and R. Patel, "MultiBoot with Virtex-5 FPGAs and Platform Flash XL", XAPP1100 (v1.0), November 2008.

[13] Hussein, J., "Multiple-Boot with Platform Flash PROMs", XAPP483 (v2.0.1), November 2007.

[14] Kalte, H., D. Langen, E. Vonahme, A. Brinkmann, and U. Ruckert, "Dynamically Reconfigurable System-on-Programmable-Chip", In *Proceedings of 10th Euromicro Workshop on Parallel, Distributed and Networked-based Processing (EUROMICOR-PDP'02),* pp.235-242, January 2002.

[15] Kao, C., "Benefits of Partial Reconfiguration", *Xcell Journal Online,* Xilinx Inc., Janusary 2005.

[16] Lim, D., and M. Peattie, "Two Flows for Partial Reconfiguration: Module Based and Small Bit Manipulation", XAPP290, 2002.

[17] Lodi, A., C. Mucci, M. Bocchi, A. Cappelli, M. De Dominicis, and L. Ciccarelli, "A Multi-Context Pipelined Array for Embedded Systems", In *Proceedings of International Conference on Field Programmable Logic and Applications, (FPL'06)*, pp.1-8, August 2006.

[18] Masselos, K., and N. Voros, "System Level Design of Reconfigurable Systems-on-Chip", Springer, 1st edition, 2005.

[19] McDonald, E.J., "Runtime FPGA Partial Reconfiguration", IEEE Aerospace and Electronic Systems Magazine, vol. 23, no.7, pp.10-15, July 2008.

[20] Mecker, J., M. Hubber, G. Hettich, R. Constapel, J. Eisenmann, and J. Luka, "Dynamic and Partial FPGA Exploitation", In *Proceedings of IEEE,* vol.95, no.2, pp.438-452, February 2007.

[21] Ng, M., and M. Peattie, "Using a Microprocessor to Configure Xilinx FPGA via Slave Serial or SelectMap Mode", XAPP502, v1.5, December 2007.

[22] Perera, D.G., "*Chip-Level and Reconfigurable Hardware for Data Mining Applications*", PhD Dissertation, University of Victoria, BC, Canada, April 2012.

[23] Perera, D.G., and K.F. Li, "Analysis of Single-Chip Hardware Support for Mobile and Embedded Applications", In *Proceedings of IEEE Pacific Rim International Conference on Communication, Computers, and Signal Processing (PacRim'13),* pp. 369-376, August 2013.

[24] Perera, D.G., and K.F. Li, "FPGA-Based Reconfigurable Hardware for Compute Intensive Data Mining Applications", In *Proceedings of 6th IEEE International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC'11),* pp.100-108, October 2011, **(Best Paper Award).**

[25] Scalera, S.M., and J.R. Vazquez, "The Design and Implementation of a Context Switching FPGA", In *Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines,* pp.78-85, April 1998.

[26] Sedcole, P., B. Blodget, T. Becker, J. Anderson, and P. Lysaght, "Modular Dynamic Reconfiguration in Virtex FPGAs", *IEE Computers and Digital Techniques,* vol.153, no.3, pp.157-164, May 2006.

[27] Sidhu, R.P.S., A. Mei, V.K. Prasanna, "String Matching on Multicontext FPGAs using Self-Reconfiguration", In *Proceedings of ACM/SIGDA 7th International Symposium on Filed Programmable Gate Arrays,* pp.217-226, 1999.

[28] Trimberger, S., D. Carberry, A. Johnson, and J. Wong, "A Time-Multiplexed FPGA", In *Proceedings of 5th Annual IEEE Symposium on FPGAs for Custom Computing Machines,* pp.22-28, April 1997.

[29] Wesselkamper, J., "Fail-Safe MultiBoot Reference Design", XAPP468 (v1.0), November 2008.

[30] Xilinx, Inc, "AR #7662 - XPLA2: Length of Time for Configuration or Download", www.xilinx.com/support/answers/7662.htm.

[31] Xilinx, Inc., "LogiCORE IP XPS HWICAP (v5.00a)", DS586, July 2010, http://www.xilinx.com/support/documentation/ip_documentation/xps_hwicap.pdf.

[32] Xilinx, Inc., "Partial Reconfiguration User Guide" UG702 (v12.3), October 2010, http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_3/ug702.pdf.

[33] Xilinx, Inc., "PlanAhead User Guide", UG632 (v 11.4), December 2009. http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/PlanAhead_UserGuide.pdf.

[34] Xilinx, Inc., "Virtex-6 FPGA Configuration User Guide" UG360 (v3.2) November 2010, http://www.xilinx.com/support/documentation/user_guides/ug360.pdf.

[35] Xilinx, Inc., "XPS multi-channel external memory controller", (XPS MCH EMC) (v3.01a), DS575, April 2010.

[36] Xilinx, Inc., "XPS SYSACE (System ACE) interface controller (v1.01a)", DS583, July 2009, http://www.xilinx.com/support/documentation/ip_documentation/xps_sysace.pdf..

An EG-LDPC Based 2-Dimensional Error Correcting Code for Mitigating MBUs of SRAM Memories

Ahmet Turan EROZAN
Yildirim Beyazit University
Electrical and Electronics Engineering Department
Ankara/Turkey
+90 312 324 1555
135102102@ybu.edu.tr

Enver ÇAVUŞ
Yildirim Beyazit University
Electrical and Electronics Engineering Department
Ankara/Turkey
+90 312 324 1555
ecavus@ybu.edu.tr

ABSTRACT
In this paper, a 2-D error correction code architecture based on EG-LDPC and single parity check (SPC) code is proposed as a solution to MBU problem of SRAM memories. EG-LDPC codes have better multiple error correction and detection capabilities than conventional codes and they have low complexity decoders. Therefore, they are suitable for fault tolerant memory applications. The proposed architecture uses (15, 7, 5) EG-LDPC as row encoding and SPC code for column encoding. In order to minimize decoding complexity of 2D structure, a standard array decoder is utilized. The investigated architecture is compared with previously proposed Matrix code method. The proposed architecture is able provide over 95% error correction coverage up to 4 errors and a significant 100% error detection up to 12 bit errors. In terms of MTTFs, the proposed approach achieves 63% improvement over Matrix codes at fault rates of 10^{-4} and 10^{-5}. Matrix codes and the proposed architecture are implemented using Xilinx XC6SLX16 FPGA and comparison results in term of implementation complexity are provided.

Categories and Subject Descriptors
Hardware~Error detection and error correction, Hardware~Self-checking mechanism, Hardware~Reconfigurable logic applications, Hardware~External storage

General Terms
Algorithms, Performance, Design, Reliability, Experimentation, Security.

Keywords
Multiple Bit Upsets (MBUs), EG-LDPC, two dimensional error correcting codes (ECCs), fault tolerant memories, soft errors.

1. INTRODUCTION
As CMOS process technology scales deep into sub-40 nm regime and supply voltage decreases, memory cells become geometrically smaller and hold less charge. As a result, radiation induced soft errors become increasingly important factor in the reliability of memories [1]-[8]. Soft errors occur when an energetic neutrons or alpha particles released from chip material strike a memory cell. When a single strike of such particles changes the content of the memory, this event is called Single Event Upset (SEU). An SEU can flip one bit cell (SBU: Single-Bit Upset) or multiple bit cells (MBU: Multi-Bit Upset). When memory elements are used in mission-critical applications, SBUs or MBUs can have serious consequences such as functional failure or system loss [9].

As SRAM memory chips manufactured with small feature size, low noise margins and low voltage level, MBUs become the dominant contributor of the overall soft error rate [6]-[8][10]. Therefore, Single Event Correcting–Double Event Detecting (SEC-DED) codes, such as Hamming codes, would be unable to mitigate the soft errors alone [6]. Other conventional memory protection methods such as bit-interleaving in combination with SEC-DED codes [16] and Triple Modular Redundancy (TMR) are not feasible either, as scaling up these techniques to cover large-scale MBUs will incur excessive increase in area, latency and power consumption of SRAM memories [17]-[22].

One promising solution to mitigate MBUs with large widths is to construct two dimensional (2-D) ECC structures [23]-[28], which can provide scalable multi-bit error protection against large clusters of soft errors. Compared to conventional schemes with similar error coverage, 2-D ECC architectures offer significantly smaller latency, resource usage and power consumption figures. The first applications of two dimensional ECC approaches for memories are presented in [26] and [27] in the form of product code, and later an advanced bidirectional parity code is given in [28]. More recent examples of 2-D schemes can be found in [23]-[25]. A new 2-D structure, constructed by a combination of Hamming codes and Parity codes, referred as Matrix code, is introduced in [24] and [25]. In [23], another 2-D error detection scheme, where a multi-bit error detection (MED) code is used row-wise, and vertical parity codes (VPC) are utilized as column code, is presented and compared with BCH, Hamming and Matrix codes.

Recently, Euclidian Geometry Low Density Parity Check (EG-LDPC) codes are proposed to overcome effects of MBUs in memories [29][30][36]. EG-LDPC codes have better multiple error correcting capabilities than conventional codes and they have low complexity and low delay decoders. Therefore, they are very suitable for fault tolerant memory applications. The first application of EG-LDPC codes for fault tolerant memory applications appeared in [29] and [30], where a one dimensional

FPGAWorld'15, September 8-10, Stockholm and Copenhagen.

serially implemented EG-LDPC code for nanoscale memories is presented. In a more recent work [36], EG-LDPC codes are proposed as an efficient multiple bit error correction method for memory systems and optimized schemes for (15,7,5), (63,37,9) and (255,175,17) EG-LDPC codes are presented with 2-, 4- and 8-bit errors correction capabilities, respectively.

In this paper, we present a 2-D ECC architecture using EG-LDPC in row-wise and single parity check (SPC) code in column-wise. The proposed decoding approach is based on a standard array decoding procedure, where a set of syndromes is pre-computed corresponding to correctable error patterns. The error correction bits are then set according to a Boolean function mapping of syndrome patterns. The combination of EG-LDPC code and SPC code improves error correction and detection capability while utilizing a standard array syndrome decoding minimizes resource allocation compared to standard majority logic decoding of EG-LDPC codes. The proposed architecture is able provide over 95% error correction coverage up to 4 errors and a significant 100% error detection up to 12 bit errors. In terms of MTTFs, the proposed approach achieves 63% improvement over Matrix codes at fault rates of 10^{-4} and 10^{-5}. Matrix codes and the proposed architecture are implemented using Xilinx XC6SLX16 FPGA and comparison results in term of implementation complexity are provided.

The rest of the paper is organized as follows. In Section II, EG-LDPC codes and fault tolerant SRAM architectures are reviewed. The explored 2-D ECC architecture is discussed in Section III. Then, the implementation results of studied architecture and its comparison to earlier works are presented in Section IV. Finally, Section V concludes this paper.

2. REVIEW OF EG-LDPC CODES

In this section, first the construction of EG-LDPC codes, which are based on line and points of corresponding finite geometries are explained and corresponding encoding and decoding schemes are discussed [31]. Then, the fault tolerant SRAM architecture that is assumed in this work is reviewed.

2.1 EG-LDPC Codes

Euclidean Geometry is finite geometry with following structural properties:

1. Every line consists of ρ points;
2. Any two points are connected by one and only one line;
3. Every point is intersected by γ lines;
4. Any two lines either intersect at only one point or they are parallel.

Let H be a binary matrix. Rows of H correspond to lines, and columns represent points of EG. Rows of H, called incidence vectors, display points in a line and has weight ρ. Columns of H, called intersecting vector of points in EG, represent the lines that intersect at a specific point. Columns have weight of γ. In H, h_{ij} equals to 1 if jth point lies on ith line. H can also be considered as an incidence matrix of the lines in EG over the points in EG. Parity check matrix H defined in EG is shown to fit to the definition of LDPC matrices. Therefore the code represented by H is an LDPC code. Throughout this work, type-I EG-LDPC code [29] is used in implementations. Any ECC can be characterized by code length (n), information bit length (k) and minimum distance (d) and represented by triple (n, k, d). Parity check matrix of Type-I EG-LDPC codes have following parameters as shown in [27] for any t>=2:

- information bits, $k = 2^{2t}-3t$;
- length, $n = 2^{2t}-1$;
- minimum distance, $d\_min = 2t+1$;
- dimensions of the parity-check matrix: n x n ;
- row weight of the parity-check matrix, $\rho = 2^t$;
- column weight of the parity-check matrix, $\gamma = 2^t$.

Parity check matrix H is constructed by taking an incidence vector in EG and $(2^{2t} – 2)$ shifts of the incidence vector as rows. As parity check matrix's rows formed by shifting a vector, EG-LDPC is a cyclic code. H's rows are not necessarily linearly independent. The rank of H is (n – k) therefore the code is a (n, k, d_{min}) linear code [29]. Since H matrix is (n x n), there are n syndromes instead of (n – k). A syndrome bit is generated for each bit at codeword, not only for information bits.

As parity check matrices of EG-LDPC codes are sparse (e.g. $\rho << n$, $\gamma << n$), EG-LDPC codes have low complexity and low delay decoders. This feature of EG-LDPC codes gives designers the opportunity of implementing encoders and decoders using basic logical elements like AND, XOR gates and majority gate logic. Type-I EG-LDPC codes are known to be one step majority logic correctable. One step majority corrector is a fast and compact error correcting method, which can corrects up to $\gamma/2$ error bits in the received information vector.

Encoding process of EG_LDPC codes is performed by multiplying information vector with generator matrix to get the codeword, e.g., c = i.G. Generator matrix is obtained from a generator polynomial vector, where (k - 1) shifts of polynomial vector forms the rows of H. For decoding of EG-LDPC codes, one step majority logic decoding (MLD) is used. In one step MLD, received n-tuple is loaded into the buffer register and the parity check equations are computed and fed to the majority logic gate. If a majority of parity checks have a value of one, the last bit is inverted. Then all bits are cyclically shifted and same operations are performed again. This process is repeated n times until the bits are in the same position in which they were loaded.

2.2 Fault Tolerant SRAM Architecture

SRAM memories are widely used for supplying dynamic storage medium needed by applications running on microprocessors or FPGAs. SRAM memories are popular for dynamic operations because write/read mechanisms are simpler compared to other memory devices such as Flash memories. In this work, the assumed architecture consists of an SRAM memory chip and an FPGA. The encoder and decoder circuitries are implemented on FPGA. The FPGA has a lot more physical resources than the implementation needs. Therefore, remaining resource can be used for implementing other targeted applications or a smaller FPGA can be chosen if targeted applications need to run on another environment (e.g., a microprocessor).

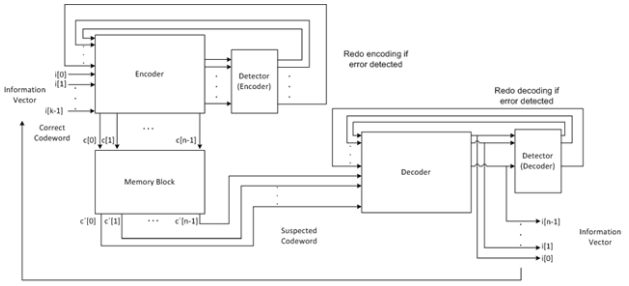


Figure 1. Fault Tolerant Memory Architecture

The fault tolerant SRAM memory architecture that consists of encoder, decoder, detectors, memory controllers and memory unit is given in Fig. 1. As the the decoder and encoder units are also susceptible to soft errors, detector units are needed to assure fault tolerance at encoder and decoder. In general, fault tolerant schemes like logic replication or concurrent parity prediction methods are used to build fault tolerant encoder and detector circuits [32]. However, as EG-LDPC codes has fault secure detector [FSD] feature [29], detector circuitry provides sufficient protection to encoder and decoder circuits without any need of additional fault-tolerant circuitry. In the following, the steps for accessing the fault tolerant SRAM memory architecture of Fig. 1 are given:

1. When a write operation is requested, memory controller asserts information vector, address and control signals.
2. Information vector is encoded according to selected coding scheme, codewords are constructed.
3. Output of encoder is checked against transient errors. If an error is observed, encoding is repeated.
4. If no error is detected, codewords are written to SRAM memory.
5. When a read operation is requested, data is read to FPGA.
6. Temporarily stored codewords are decoded

Output of decoder is checked by detector against possible transient errors. If no error is observed data is passed to microcontroller/FPGA on which application runs. If there is a transient error, decoding is repeated.

3. PROPOSED 2-D ARCHITECTURE BASED ON EG-LDPC CODES

In this section, a 2-D ECC architecture based on (15, 7, 5) EG-LDPC code and SPC code is explored for its capability to overcome MBU errors in SRAMs. As SRAMs in general have 32-bit data width, information bit vector is taken as 32 bit. Prior to encoding, data is arranged to fit the 2-D structure. Number of columns and rows are chosen to be compatible with (15,7,5) EG-LDPC code, SPC code and 32-bit information vector width. A 35-bit vector (32-bit information bit vector, plus three '0's padded at the end of information vector) is divided into a 5 x 7 matrix, as illustrated in Fig. 2,where M_{ij}, R_{ij}, and C_{ij} dedicate information bits, row parity bits, and column parity bits respectively. For each row of the matrix, a separate EG-LDPC encoder and decoder is built and implemented in parallel to minimize latency. Columns are handled by their dedicated SPC encoders and decoders concurrently.

M_{00}	M_{01}	M_{02}	M_{03}	M_{04}	M_{05}	M_{06}	R_{00}	R_{01}	R_{02}	R_{03}	R_{04}	R_{05}	R_{06}	R_{07}
M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}	M_{16}	R_{10}	R_{11}	R_{12}	R_{13}	R_{14}	R_{15}	R_{16}	R_{17}
M_{20}	M_{21}	M_{22}	M_{23}	M_{24}	M_{25}	M_{26}	R_{20}	R_{21}	R_{22}	R_{23}	R_{24}	R_{25}	R_{26}	R_{27}
M_{30}	M_{31}	M_{32}	M_{33}	M_{34}	M_{35}	M_{36}	R_{30}	R_{31}	R_{32}	R_{33}	R_{34}	R_{35}	R_{36}	R_{37}
M_{40}	M_{41}	M_{42}	M_{43}	M_{44}	M_{45}	M_{46}	R_{40}	R_{41}	R_{42}	R_{43}	R_{44}	R_{45}	R_{46}	R_{47}
C_{00}	C_{01}	C_{02}	C_{03}	C_{04}	C_{05}	C_{06}								

Figure 2. 2D Data Structure for the Proposed (15, 7, 5) EG-LDPC and SPC Code

Encoding operation of EG-LDPC codes is performed using a generator polynomial. In Equation (1), Generator polynomial, g(x), for (15, 7, 5) EG-LDPC code is given [33]:

$$g(x) = 1 + X^4 + X^6 + X^7 + X^8 \quad (1)$$

The rows of Generator matrix G is obtained from the polynomial vector, v_G=[1 0 0 0 1 0 1 1 0 1 0 0 0 0 0], and 6 circular shifts of v_G. The G matrix obtained from vector v_G is not in a systematic form. By pivoting and elementary column operations, systematic generator matrix G' is obtained. As G' is systematic, codeword bits $c_0,c_1,\ldots,c_6$ are equal to information bits $i_0,i_1,\ldots,i_6$. Parity bits $c_7,c_8,\ldots,c_{14}$ of (15,7,5) EG-LDPC code are estimated from Equations (2)-(9) using xor operations.

$$R_7 = M_0 + M_1 + M_2 \quad (2)$$

$$R_8 = M_1 + M_2 + M_4 \quad (3)$$

$$R_9 = M_2 + M_3 + M_5 \quad (4)$$

$$R_{10} = M_3 + M_4 + M_6 \quad (5)$$

$$R_{11} = M_0 + M_1 + M_3 + M_4 + M_5 \quad (6)$$

$$R_{12} = M_1 + M_2 + M_4 + M_5 + M_6 \quad (7)$$

$$R_{13} = M_0 + M_1 + M_2 + M_5 + M_6 \quad (8)$$

$$R_{14} = M_0 + M_2 + M_6 \quad (9)$$

After row encoding is completed, column encoding is performed using SPC code, where for 5-bits of data, 1 parity bit is generated, as given in Equation (10).

$$C_{0y} = M_{0y} + M_{1y} + M_{2y} + M_{3y} + M_{4y} \quad (10)$$

For decoding process, one and two bit error combinations and their corresponding syndromes are pre-calculated and saved in a look up table. At the beginning of decoding process, syndrome values for each row and first 7 columns are calculated concurrently and non-zero syndrome values are compared to pre-saved syndrome values to search for a matching syndrome pattern. If the calculated syndrome of row j is equal to one of the pre-saved syndromes, the error combination corresponding to the syndrome is exored with row j. If the calculated syndrome of row j is non-zero and it is not equal to any saved syndromes, the calculated 7 bit column syndrome of parity code is exored with row j. Block diagram of the studied error correction and error detection method is given in Fig. 3.

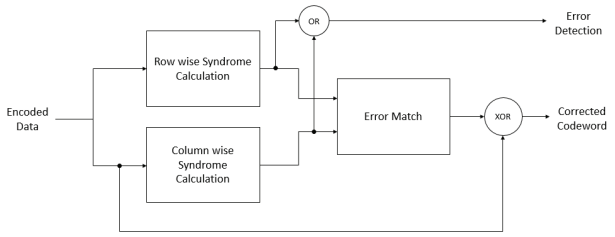


Figure 3. Proposed Approach for Error Detecting and Correction

4. RESULTS

The proposed architecture is implemented in Verilog and synthesized using Xilinx XC6SLX16 FPGA. A reference model is constructed in MATLAB to verify the functionality of circuitry and ISIM is used for simulations and test the Verilog implementation. Experimental data is exposed to errors by error injection, where errors occurring at adjacent cells are investigated as MBUs [34].

In order to compare the error correction and detection capability of the proposed method and Matrix codes, both methods are implemented in Matlab and one million experimental faults are injected to encoded bits. Table 1 illustrates that proposed 2D approach based on EG-LDPC codes greatly improves error correction and detection capability compared to Matrix code approach. As Matrix codes only provides up to 2 error correction and error detection capability of Matrix codes quickly deteriorates after 5 errors; the proposed architecture is able provide over 95% error correction coverage up to 4 errors and a significant 100% error detection up to 12 bit errors. Table 2 provides a MTTF reliability comparison between proposed method and Matrix codes. Using the calculation method presented in [24], the proposed approach achieves 63% better MTTF compared to Matrix code at fault rates 10^{-4} and 10^{-5}, while a 30% improvement is obtained at fault rate of 10^{-6}. These results reveal that employing EG-LDPC codes as a row code and utilizing syndrome based array decoding method greatly improves decoding error performance and reliability of 2D codes.

Table 1. Error Correction and Detection Comparison of Matrix Code with Proposed Method

	Correction		Detection	
	Matrix Code	Proposed Method	Matrix Code	Proposed Method
1	100	100	100	100
2	100	100	100	100
3	0	99,25	94,01	100
4	0	95,39	80,71	100
5	0	87,53	62,2	100
6	0	75,66	39,82	100
7	0	61,46	20,07	100
8	0	45,97	7,63	100
9	0	31,35	-	100
10	0	21,87	-	100
11	0	11,06	-	100
12	0	5,78	-	100

Table 2. MTTF Comparison of Matrix Code with Proposed Method

Fault Rate (upset/bit per day)	Matrix Code	Proposed Method	Improvement (%)
10^{-4}	175,48	286,446	63,24
10^{-5}	1754,76	2864,5	63,28
10^{-6}	17317,54	25059	30,90

Decoders of the Proposed Method and Matrix code are implemented in Xilinx XC6SLX16 FPGA. Comparison of resource utilization, latency and overhead of the Proposed Method and Matrix codes is given in Table 3. The Proposed Method uses %76 more resource and has %46 more latency compared to Matrix code. These implementation results are expected as the proposed architecture uses a more powerful EG-LDPC code as a row code in contrast to Hamming codes employed in Matrix code.

Table 3. Resource Utilization and Overhead Comparison of Matrix Code with Proposed Method

Resource Utilization and Latency of Decoders		
	Matrix Codes	Proposed Method
Slice LUTs	76	134
Latency	8.396 ns	12.301 ns
Overhead	0.5	0.5732

5. CONCLUSION

As a solution to MBU problem of SRAM memories, we investigated an EG-LDPC code based 2-D ECC architecture, (15, 7, 5) EG-LDPC code and SPC code, combined with standard array decoder. The proposed method is compared with Matrix codes in terms of error correction, detection capability, MTTF, allocated sources and latency. The studied scheme offers significantly improved error correction/detection performance and much higher reliability with an acceptable increase in terms of latency and resource usage. The results suggest that EG-LDPC code based 2-D architectures combined with standard array decoding can provide an enhanced solution against MBU problems of SRAM technologies.

6. REFERENCES

[1] Seifert, N.; Gill, B.; Foley, K.; Relangi, P.; "Multi-cell upset probabilities of 45nm high-k + metal gate SRAM devices in terrestrial and space environments," Reliability Physics Symposium, 2008. IRPS 2008. IEEE International , vol., no., pp.181-186, April 27 2008-May 1 2008.

[2] Radaelli, D.; Puchner, H.; Skip Wong; Daniel, S.; , "Investigation of multi-bit upsets in a 150 nm technology SRAM device," Nuclear Science, IEEE Transactions on , vol.52, no.6, pp. 2433- 2437, Dec. 2005.

[3] Song, Y.; Vu, K.N.; Cable, J.S.; Witteles, A.A.; Kolasinski, W.A.; Koga, R.; Elder, J.H.; Osborn, J.V.; Martin, R.C.;

Ghoniem, N.M.; , "Experimental and analytical investigation of single event, multiple bit upsets in poly-silicon load, 64 K×1 NMOS SRAMs," Nuclear Science, IEEE Transactions on , vol.35, no.6, pp.1673-1677, Dec 1988.

[4] Naseer, R.; Draper, J.; , "Parallel double error correcting code design to mitigate multi-bit upsets in SRAMs," Solid-State Circuits Conference, 2008. ESSCIRC 2008. 34th European , vol., no., pp.222-225, 15-19 Sept. 2008.

[5] Musseau, O.; Gardic, F.; Roche, P.; Corbiere, T.; Reed, R.A.; Buchner, S.; McDonald, P.; Melinger, J.; Tran, L.; Campbell, A.B.; , "Analysis of multiple bit upsets (MBU) in CMOS SRAM," Nuclear Science, IEEE Transactions on , vol.43, no.6, pp.2879-2888, Dec 1996.

[6] Buchner, S.; Campbell, A.B.; Meehan, T.; Clark, K.A.; McMorrow, D.; Dyer, C.; Sanderson, C.; Comber, C.; Kuboyama, S.; , "Investigation of single-ion multiple-bit upsets in memories on board a space experiment," Radiation and Its Effects on Components and Systems, 1999. RADECS 99. 1999 Fifth European Conference on , vol., no., pp.558-564, 1999.

[7] Seifert, N.; Slankard, P.; Kirsch, M.; Narasimham, B.; Zia, V.; Brookreson, C.; Vo, A.; Mitra, S.; Gill, B.; Maiz, J.; , "Radiation-Induced Soft Error Rates of Advanced CMOS Bulk Devices," Reliability Physics Symposium Proceedings, 2006. 44th Annual., IEEE International , vol., no., pp.217-225, 26-30 March 2006.

[8] Maiz, J.; Hareland, S.; Zhang, K.; Armstrong, P.; , "Characterization of multi-bit soft error events in advanced SRAMs," Electron Devices Meeting, 2003. IEDM '03 Technical Digest. IEEE International , vol., no., pp. 21.4.1-21.4.4, 8-10 Dec. 2003.

[9] B. Cooke, "Reed Muller Error Correcting Codes," MIT Undergraduate J. Math., vol. 1, pp. 21–26, 1999

[10] Johansson, K.; Ohlsson, M.; Olsson, N.; Blomgren, J.; Renberg, P.-U.; , "Neutron induced single-word multiple-bit upset in SRAM," Nuclear Science, IEEE Transactions on , vol.46, no.6, pp.1427-1433, Dec. 1999.

[11] Giot, D.; Roche, P.; Gasiot, G.; Autran, J.-L.; Harboe-Sorensen, R.; , "Heavy Ion Testing and 3-D Simulations of Multiple Cell Upset in 65 nm Standard SRAMs," Nuclear Science, IEEE Transactions on , vol.55, no.4, pp.2048-2054, Aug. 2008.

[12] Naseer, R.; Draper, J.; , "Parallel double error correcting code design to mitigate multi-bit upsets in SRAMs," Solid-State Circuits Conference, 2008. ESSCIRC 2008. 34th European , vol., no., pp.222-225, 15-19 Sept. 2008.

[13] Autran, J.L.; Serre, S.; Munteanu, D.; Martinie, S.; Semikh, S.; Sauze, S.; Uznanski, S.; Gasiot, G.; Roche, P.; , "Real-time Soft-Error testing of 40nm SRAMs," Reliability Physics Symposium (IRPS), 2012 IEEE International , vol., no., pp.3C.5.1-3C.5.9, 15-19 April 2012

[14] Durna, M.; Atar, O.; Ceylan, M.; Cakmakci, Y.; Demirci, M.; Kozal, O.A.; Oturak, M.; Ozdemir, A.; Turhan, O.; , "On board data handling subsystem featuring BiLGE," Recent Advances in Space Technologies (RAST), 2011 5th International Conference on , vol., no., pp.932-937, 9-11 June 2011

[15] Tang, H.; Xu, J.; Lin, S.; Abdel-Ghaffar, K.A.S.; , "Codes on finite geometries," Information Theory, IEEE Transactions on , vol.51, no.2, pp.572-596, Feb. 2005.

[16] S. Baeg, S.Wen, and R.Wong, "SRAM interleaving distance selectionwith a soft error failure model," IEEE Trans. Nucl. Sci., vol. 56, no. 4, pp. 2111–2118, Aug. 2009.

[17] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, J.C. Hoe, "Multi-bit error tolerant caches using two-dimensional error coding," 40th Annual IEEE/ACM Int. Symposium on microarchitecture 1–5, MICRO 2007, pp. 197–209, Dec. 2007.

[18] Shyue-Win Wei; Che-Ho Wei; , "High-speed hardware decoder for double-error-correcting binary BCH codes," Communications, Speech and Vision, IEE Proceedings I , vol.136, no.3, pp. 227- 231, Jun 1989.

[19] El-Medany, W.M.; Harrison, C.G.; Garrell, P.G.; Hardy, C.J.; , "VHDL implmentation of a BCH minimum weight decoder for double error ," Radio Science Conference, 2001. NRSC 2001. Proceedings of the Eighteenth National , vol.2, no., pp.361-368 vol.2, 2001.

[20] Bentoutou, Y.; Djaifri, M.; , "Observations of single-event upsets and multiple-bit upsets in random access memories on-board the Algerian satellite," Nuclear Science Symposium Conference Record, 2008. NSS '08. IEEE , vol., no., pp.2568-2570, 19-25 Oct. 2008

[21] Durna, M.; Atar, O.; Ceylan, M.; Cakmakci, Y.; Demirci, M.; Kozal, O.A.; Oturak, M.; Ozdemir, A.; Turhan, O.; , "On board data handling subsystem featuring BiLGE," Recent Advances in Space Technologies (RAST), 2011 5th International Conference on , vol., no., pp.932-937, 9-11 June 2011

[22] Bentoutou, Y.; , "Efficient Memory Error Coding for Space Computer Applications," Information and Communication Technologies, 2006. ICTTA '06. 2nd , vol.2, no., pp.2347-2352,

[23] Zhu, M., Xiao, L., Li, S., & Zhang, Y. (2010). Efficient Two-Dimensional Error Codes for Multiple Bit Upsets Mitigation in Memory. 2010 IEEE 25th International Symposium on Defect and Fault Tolerance in VLSI Systems, 129–135. doi:10.1109/DFT.2010.22

[24] Argyrides, C.; Zarandi, H.R.; Pradhan, D.K.; , "Matrix Codes: Multiple Bit Upsets Tolerant Method for SRAM Memories," Defect and Fault-Tolerance in VLSI Systems, 2007. DFT '07. 22nd IEEE International Symposium on , vol., no., pp.340-348, 26-28 Sept. 2007

[25] Argyrides, C.; Pradhan, D.K.; Kocak, T.; , "Matrix Codes for Reliable and Cost Efficient Memory Chips," Very Large Scale Integration (VLSI) Systems, IEEE Transactions on , vol.19, no.3, pp.420-428, March 2011

[26] F. Barton, "A fault tolerant integrated circuit memory," Ph.D. dissertation, Dep. Comput. Sci., Calif. Inst. Tech., Apr. 1980

[27] Tanner, R.M.; , "Fault-Tolerant 256K Memory Designs," Computers, IEEE Transactions on , vol.C-33, no.4, pp.314-322, April 1984

[28] Yamada, J.; Mano, T.; Inoue, J.; Nakajima, S.; Matsuda, T.; , "A submicron 1 Mbit dynamic RAM with a 4-bit-at-a-time

built-in ECC circuit," Solid-State Circuits, IEEE Journal of , vol.19, no.5, pp.627-633, Oct. 1984

[29] Naeimi, H.; DeHon, A.; , "Fault Secure Encoder and Decoder for NanoMemory Applications," Very Large Scale Integration (VLSI) Systems, IEEE Transactions on , vol.17, no.4, pp.473-486, April 2009.

[30] H. Naeimi and A. DeHon, "Fault secure encoder and decoder for memory applications," in Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Syst., Sep. 2007, pp. 409–417.

[31] Tang, H.; Xu, J.; Lin, S.; Abdel-Ghaffar, K.A.S.; , "Codes on finite geometries," Information Theory, IEEE Transactions on , vol.51, no.2, pp.572-596, Feb. 2005.

[32] M. Sipser and D. Spielman, "Expander codes,"IEEE Trans. Inf. Theory, vol. 42, no. 6, pp. 1710–1722, Nov. 1996.

[33] S. Lin and D. J. Costello, Error Control Coding, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 2004.

[34] Quinn, H.; Morgan, K.; Graham, P.; Krone, J.; Caffrey, M.; , "Static Proton and Heavy Ion Testing of the Xilinx Virtex-5 Device," Radiation Effects Data Workshop, 2007 IEEE , vol.0, no., pp.177-184, 23-27 July 2007

[35] Gustavo Neuberger, Fernanda de Lima, Luigi Carro, and Ricardo Reis. 2003. A multiple bit upset tolerant SRAM memory. ACM Trans. Des. Autom. Electron. Syst. 8, 4 (October 2003), 577-590.

[36] Reviriego, P.; Flanagan, M. F.; Liu, S.-F.; Maestro, J. A.; , "On the Use of Euclidean Geometry Codes for Efficient Multibit Error Correction on Memory Systems," Nuclear Science, IEEE Transactions on , vol.59, no.4, pp.824-828, Aug. 2012.

High Level Synthesis Based Hardware Accelerator Design for Processing SQL Queries

Gorker Alp Malazgirt
Bogazici University
Department of Computer Engineering
Bebek, Istanbul, Turkey
alp.malazgirt@boun.edu.tr

Nehir Sonmez
Barcelona Supercomputing Center
Barcelona, Spain
nehir.sonmez@bsc.es

Arda Yurdakul
Bogazici University
Department of Computer Engineering
Bebek, Istanbul, Turkey
yurdakul@boun.edu.tr

Adrian Cristal
Centro Superior de Investigaciones Cientificas (IIIA-CSIC)
Barcelona Supercomputing Center
Barcelona, Spain
adrian.cristal@bsc.es

Osman Unsal
Barcelona Supercomputing Center
Barcelona, Spain
osman.unsal@bsc.es

ABSTRACT

About three exabytes of data is created and stored in databases each day, and this number is doubling approximately every forty months. Querying this enormous amount of data has been a challenge and new methods have been actively researched. In this paper, we present hardware accelerators which are designed to speed up database analytics for in-memory databases. Unlike traditional hardware accelerator designs, our hardware accelerators are composed using High Level Synthesis (HLS), which enables high level descriptions of functionality such as data filtering, sorting, equijoins to be targeted directly into RTL. We have simulated TPC-H benchmark queries using Xilinx Vivado HLS managed in our custom simulation software framework. Our results have demonstrated the capabilities of HLS in database acceleration domain; such that the 200MHz FPGA accelerator can provide two orders of magnitude performance improvement compared to PostgreSQL based full software implementation running on a modern multicore system.

Categories and Subject Descriptors

C.1.3 [**Other Architecture Styles**]: Adaptable Architectures

1. INTRODUCTION

An in-memory database is a database management system (DBMS) that primarily relies on main memory for computer data storage, as opposed to a traditional disk storage mechanism. With the introduction of faster and more capable memory technologies, main memory databases are faster than disk-optimized databases, using RAMs as the main storage units.

Many previous studies, in order to have faster query processing capabilities, have looked into accelerating database analytics in hardware: using ASICs [13], or using FPGAs statically [4, 12, 11], or using dynamic reconfiguration [10, 2]. However, no previous work has looked into how such accelerators can be designed using High Level Synthesis (HLS).

Traditional accelerator design requires writing complex RTL code that is prone to errors and difficult to debug. HLS uses high level software implementations of algorithms. In this work, we make the following contributions:

- Using Vivado HLS, we design hardware accelerators for data filtering, aggregation, sort, merge, join and string matching operations for a Virtex-7 FPGA. We describe the design implementation, and the tradeoffs of employing each accelerator module.

- We use these modules to simulate an in-memory database accelerator. We present performance and area results of simulating three TPC-H benchmarks completely and compare the results with a modern DBMS, PostgreSQL software implementation that runs on 32-core 2.60 GHz 256GB Intel processor

The next section presents the implementation of our accelerators. Section 3 describes how we run full queries, and validate the results of our experiments. Section 4 includes the related work, and Section 5 concludes the paper.

2. ACCELERATOR IMPLEMENTATION

In order to support full and complex database analytics in hardware, we have focused on accelerating data filtering, arithmetic, logic, sorting, aggregation and equi-join operations. All the units are designed to work at 200 MHz on a Virtex-7 xc7vx690tffg1761-2 FPGA.

FPGAWorld '15 September 8-10, Stockholm and Copenhagen

2.1 Data Filtering, Arithmetic and Logic Operations

Database filtering operations are relational operations that test numerical or logical relations between columns, numerical and/or boolean values. For this purpose, we designed a pipelined, parametrizable width, n-way compute engine that takes rows as inputs, applies a filtering operation on the desired columns and produces an output bitmap. This bitmap determines the selected rows for further processing after the filtering operation. The main importance of filtering operations in an SQL query is to filter out unwanted data from further processing, thus reducing the size of the input set. The most important design choices of filtering operations is selecting the correct parallelism for the maximum utilization of memory bandwidth. Similarly, we have designed pipelined, parametrizable, n-way arithmetic and logical compute engines. The arithmetic compute engine supports integer ADD, SUB, MULT and DIV operations, whereas the logical compute engine supports the logical AND, OR and NAND operations.

2.2 Aggregation Operations

Aggregation operations compute a single value from a collection of values and we provide an n-way aggregator engine that supports MAX, MIN, COUNT, SUM and AVERAGE. Similar to the filtering unit, it takes multiple values in the form of an array as the input and calculates the final aggregate value. We apply the binary fan-in method, which is fully pipelined at each stage. The number of pipeline stages is dependent on the input array size such that $\log_2 arraysize$ stages should be designed. This way, the Vivado HLS tool can easily map equations to two-input comparators in Virtex-7 and support full pipelining.

2.3 Sort and Merge

Ordering data is widely used in queries for data organization and presentation. Therefore, efficient sorting is highly sought in database analytics. For this reason, we implemented a 64-way Bitonic sorting network [1], which is highly efficient on FPGAs, providing high throughput while using acceptable amounts of real estate.

Figure 1 presents the overview of the sort, merge and join processes. Bitonic network is fed with unordered data from RAM. All memory accesses between the hardware accelerators and the memory are shown with *mem access* labels in the figure. Bitonic network can only partially sort a large data set and it is not adequate for high volumes. Therefore, partially sorted values are written back to RAM. This first pass creates arrays of partially sorted data and these arrays must be merged for global sorting. For this purpose, we designed a 4-deep, 2-way merger that can output 4 values at a time, that is similar to the merge unit presented in [4]. It uses 14 comparators, and was written as a long case statement (16 possible output combinations for a 2x4 array).

The depth of four of the merger unit has allowed the utilize available bandwidth without consuming too much FPGA area, hence it is possible to use multiple instances given higher bandwidth specifications. Although larger mergers could be designed, the amount of comparisons would super-linearly increase.

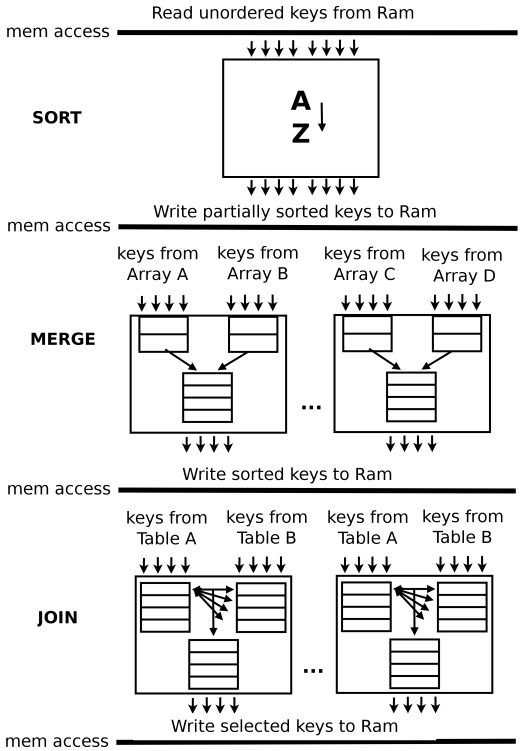


Figure 1: Required steps for Sort-merge Join operation

2.4 Table Joins

A join operator is used to combine fields from two tables by using values that are common to each. Joining two tables is a very time consuming operation, especially when dealing with decision support systems. Table joins account for more than 40% of total execution time while running TPC-H queries [13].

We opted for implementing the merge-join algorithm because of two reasons. Firstly, we already have a highly parallel sorting network which a merge-join requires. Secondly, the sort network and merge unit can be highly parallelized through HLS. In contrast, a hash table might require managing collisions, and other complicated circuitry which are difficult to express in HLS. The remaining option, nested loop joins are more suitable for software implementations. For these reasons, we have designed a parallel merge-join block (for equi-joins) that can match and join a window of (n x n) rows on two given columns by comparing sorted rows of database tables in parallel. In order to perform a merge-join, first we need to sort and merge both the input tables by using the Sort and Merge modules. Later the join operates on the sorted tables. Similar to our merge unit, our join operator can join 4 rows from two separate tables such as Table A and Table B in Figure 1. Our experiments have shown that increasing the number of rows in the merge unit presents different latencies and area consumptions. For example, in our design a 4x4 join has a 4-cycle latency.

3. RUNNING FULL, COMPLEX QUERIES

To test our engine, we ran example queries from TPC-H, a decision support benchmark, which illustrates decision support systems that examine large volumes of data, execute queries with a high degree of complexity, and give answers

to critical business questions [6]. In this work, we have run TPC-H queries 6, 14 and 19 [6] which involves filtering, join, aggregation operations. Although this is not a full stress-test, we believe that these queries provide a good testbench because they present some different properties: (i) q6 can be fully run without writing intermediate results to memory, (ii) q14 introduces memory overheads, as well as performing a large join operation (of size 200K x 74K), and (iii) q19 additionally performs various iterations on our accelerators. One exception to a software run is that currently, we do not provide support for floating point numbers, and we only use fixed point arithmetic. Therefore, we converted the dates into 32-bit timestamps and the often used `numeric(15,2)` type was converted to fixed-point, similarly to [13]. With Vivado HLS, it is straightforward to provide floating point support. However, it is not in the scope of this paper. In order to run full TPC-H queries, we also needed to design some additional special units such as: multiply-accumulate, divider, or conditional outputs.

In a typical SQL relational DBMS, a query execution planner does the job of establishing an ordered set of steps that are used to access data. Similarly, a planner in hardware can be devised to schedule operations to modules. The scheduling problem also affects if running multiple queries in flight can be supported or not. In this work, we manually establish the query plan, as we describe in the next subsection. We want to automatize this process in the future.

The input data is organized in memory in table columns. We assume that a given index `n` for column `a` points to the same row for column `b`, i.e. the tuple order is preserved across the columns. To have a realistic simulation environment, we constrained our memory accesses to a maximum of 512-bits each cycle/channel, and with a 16-cycle latency at 200 MHz, which matches the DDR3 specifications on our VC-709 FPGA board (featuring 2 channels of 4GB DDR3, up to 120 Gbits/s per channel). Although this does not perfectly model DRAM properties, we believe that it can be a representative way for adding memory latency and bandwidth constraints to our model. We used both the available DDR3 channels, whereas more channels can easily be converted into more performance by laying out the data columns in parallel and thus increasing bandwidth. We also assumed that for exploiting the maximum bandwidth, columns might be distributed into the two channels.

Depending on the query plan, we compose our accelerator modules together and write the intermediate results into memory as needed. A unified memory model enables our accelerator system to be customizable for each query and is capable of expressing complex queries. Combining all modules in a long pipeline would be another alternative design. However, it is not very clear how this design decision would affect performing iterations, such as in a merge-sort scenario.

3.1 Verification of Hardware Accelerators

Before discussing discuss each query in detail, we mention and detail our functional simulation framework. We have verified the functionality of our accelerators and queries in three steps. The first step is functional verification of the accelerators in software. Each accelerator has its own C/C++ source code. The source code of the accelerator under test is added to the software simulator framework. Then, all the necessary macro definitions and data types are arranged according to the underlying instruction set. The original or a synthetic input file is read and output is compared with the expected results. The full software simulation allows to guarantee the correctness of the hardware accelerators.

The second step of verification is to embed the hardware accelerator code in Vivado HLS and to test it. The major differences between the full software implementation of an accelerator and its Vivado HLS version are the directives, custom data widths and file I/O. Vivado HLS directives manage the compiler optimizations and synthesize the hardware accordingly. Similarly, custom data types allow fine grain width control on variables on the synthesized hardware. File I/O manages the data management between external memory and FPGA memory. This could be a major hurdle because Vivado does not have any mechanisms to replace the file I/O for memory management. For operations which require a lot of I/O like the join operator, the memory management controller hardware must be implemented.

After the accelerator source is modified to work on Vivado HLS, the original test bench software is also ported to the Vivado HLS framework. The major hurdle during this porting is the incompatible library usage between compilers. Vivado supports GCC 4.6, therefore, test benches which are designed with other compilers might need to be rewritten to work with Vivado. The last step is to generate the RTL files from C/C++ and to verify the functionality at the RTL level. The Vivado HLS automates the process of RTL verification and generates the RTL files of accelerators and test bench codes. Generating RTL test bench files from test bench source code turns out to be efficient because converting the complex logic in software test bench to RTL test bench saves a large amount of time.

3.2 Controlling Hardware Accelerators

The hardware accelerators that were defined previously need control logic in order to read, operate and send data to the RAM. For this reason, we implemented controllers in software and also synthesize them in Vivado HLS for obtaining their resource usage. In the software implementation, in addition, we also implement the memory controller. The memory read/write operations are implemented using C++ I/O library. Thus, in the software implementation, the memory controller is coupled with the controllers of hardware accelerators. This design choice was made to reduce the design time of the simulator. The hardware design of the memory controller is not in the scope of this paper. In an actual implementation, the memory controller and the controller of the hardware blocks can be either separate or combined.

3.3 Accelerator Code Generation and HLS Configuration Directives

All of the accelerators are coded from scratch using C++ for Vivado HLS. The development process has been very rapid compared to RTL. The source code for filtering units are straightforward because they consist of simple but many comparison operators which C/C++ language has very wide support of. The sorting network is also implemented as a large network of comparators and the merge block is coded as a long case block. Compared to RTL implementations, each C/C++ implementation of the accelerators have fewer lines of codes. The data widths of input data are selected according to the original data types in the software implementation. If a variable data type is defined for an input

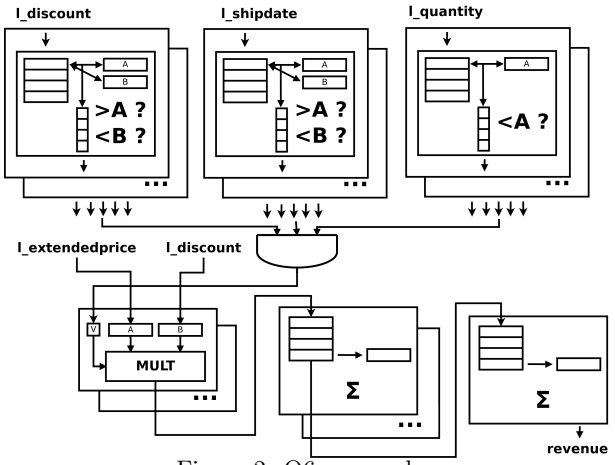


Figure 2: Q6 query plan

in the software implementation, we set the data width to 64-bits as the upper bound.

Hardware generation in Vivado HLS is controlled through optimization directives. For our hardware accelerators, we apply several directives. The *ARRAY_PARTITION* directive is used to partition input data into registers. This has prevented data access bottlenecks due to dual port BRAMs. The *INLINE* directive is used to remove function hierarchy of C/C++ language. All loops in the design are unrolled using the *UNROLL* directive. The level of unrolling is calculated based on the memory bandwidth requirements. We used the *PIPELINE* directive in order to reduce the initiation interval to 1, so that the all accelerators can process data at each clock cycle.

3.4 Experimental Results

In this work, we have used Vivado HLS and ISE version 14.1. As a basis for comparison, we ran the same TPC-H queries on a popular DBMS, PostgreSQL 9.2 running on a 32-core Intel Xeon E5-2670 at 2,60GHz, with 256GB DDR3-1600, using 4 channels and delivering up to 51.5 GB/s. We built all indexes, and ran the benchmarks on a RAMdisk to get the best possible performance out of our server.

The details of the queries are shown in [6]. Based on these queries, in Figures 2, 3 and 4, we present the query plans, where a sort sign also includes inherent merge steps. We ran TPC-H in the 1GB scale, and used the two main tables that the queries operate on. The lineitem table has around 6M rows and contains 16 columns that start with a `l_`, whereas the part table (starting with a `p_`) has 200K rows. The steps in the figures represent the execution order of the operations based on the query plan.

Q6: The 3 filter operations required were designed to work in a 5-way SIMD fashion, after which the resulting bit-maps get AND-ed. If the result is a 1, it gets multiplied-aggregated and finally outputted. This query, compared to the DBMS run, gave the highest speedup because it fits well our computational capacity and completes in a single step since we never need to write any intermediate results to memory.

Q14: In the first two steps, a 16-way filter operation reduces a 6M element table to approx. 74K rows and writes these into memory. Later, in preparation to the join operation, these columns are sorted on the key, l_partkey. Later,

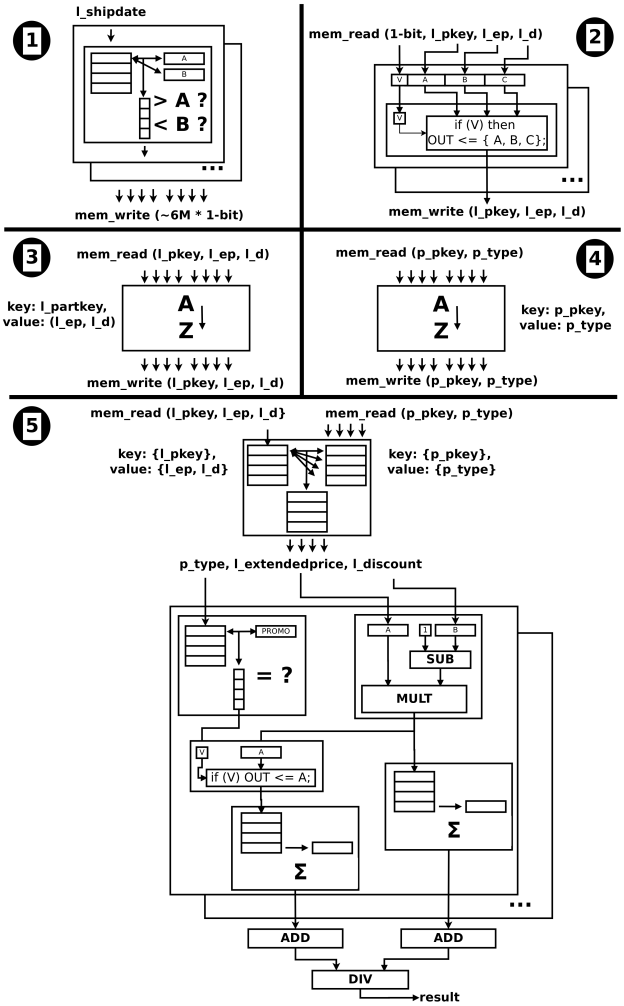


Figure 3: Q14 query plan

the 200K table is sorted on the p_partkey in step 4. Finally, these two tables get joined on the keys and two sums are kept, one for those rows that include "PROMO", and another for all the rows. The *LIKE* keyword is implemented with simple comparators and matches strings from the two tables. Finally, we divide the two aggregates and multiply this by 100 to get the final result.

Q19: This query needs to perform 3 iterations on the first 4 steps depicted. Then, these iterations are combined and 1-bit columns are created. These 1-bit columns reduce the number of input rows significantly, which implies fast sorting and fast joining afterwards. The l_table input is reduced to approx. 47K elements, and the p_table to less than 250 in all 3 cases. After the join, 2 columns are multiplied and aggregated. This whole process is repeated 3 times for each part of the case statement. The final result is obtained by summing the 3 aggregates in the end.

Our performance results running these queries are presented in Figure 5. It can be seen that our efficiency in performing n-way filtering and thus working with a reduced set of data to be sorted/joined has paid off, as we were able to achieve between 15–140x of speedup. For both systems, the reported runtimes assume that the tables are already in RAM. Furthermore, the speed up can decrease when our sys-

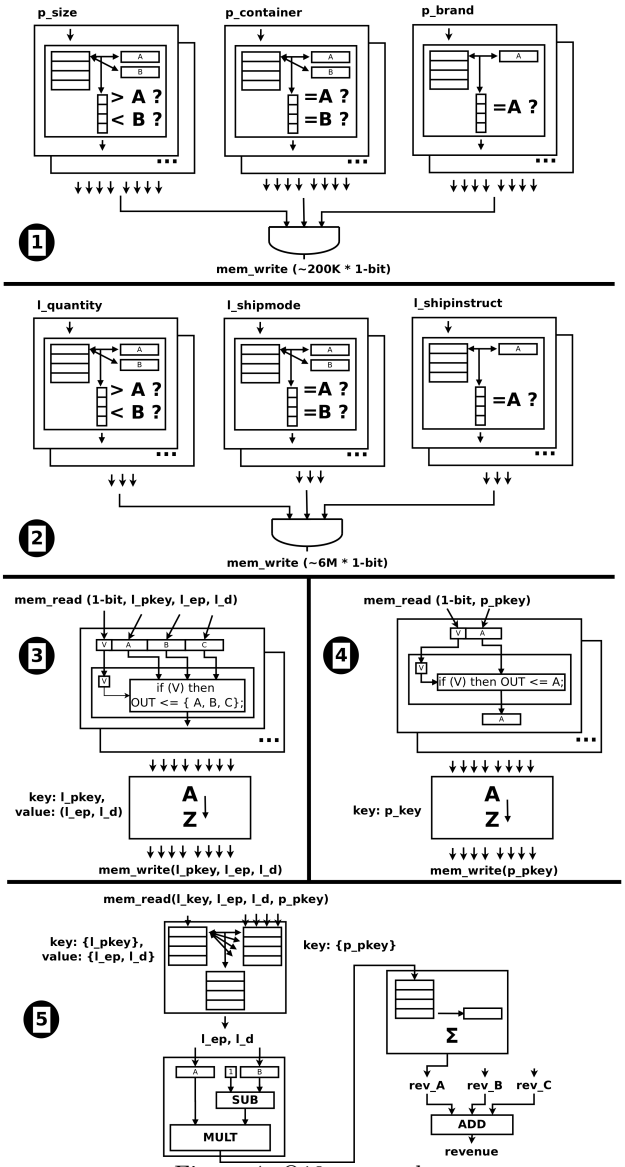


Figure 4: Q19 query plan

tem is migrated from simulator to actual hardware because the memory handling in the simulator is more optimistic than the hardware. In addition, the hardware synthesis time of the queries are not added. The synthesis time of a query is less than two minutes for queries with join operations and less than a minute with the ones without join operations. To run all queries using the same hardware, we implement a superset of all the necessary pieces, so that the FPGA is programmed with a bitstream that supports all the necessary operations for the entire query set, so that we don't reprogram/reconfigure the FPGA.

Table 1 shows area usage and latencies for the generated hardware for each query. Vivado HLS has been very efficient while synthesizing Query 6, it uses the least amount of area and provides the highest speedup. The most area consuming operations are the bitonic network due to its highly parallel nature and join unit. Other operations are negligible in size. Among the TPC-H benchmarks, Query 19 is the most area consuming query which consumes a quarter of

Table 1: Hardware usage for Q6, Q14 and Q19

Query 6	LUT	FF	DSP	Latency
5x 32-bit between	845			0
5x 32-bit LT	342			0
5x 64-bit Mult		19	80	18
5x 128-bit Sum	656	642		1
Q6 Total	1857	661	80	19
Query 14	**LUT**	**FF**	**DSP**	**Latency**
16x 32-bit between	2704			0
32-bit merge	2613			0
32-bit bitonic64	69216	20491		10
3x 200-bit comp	1284			0
2x 128-bit agg	267	265		1
2x 128-bit agg	267	265		1
3x 64-bit sub-mult	192	19	48	18
join 32-bit	42088	10137		9
Q14 Total	118659	31187	48	39
Query 19	**LUT**	**FF**	**DSP**	**Latency**
16x 32bit between	2704			0
10x 80-bit comp	980			0
8x 64-bit between	1274			0
5x 200-bit comp	642			0
32-bit 64-bitonic	69216	20491		10
32-bit merge	2613			0
3x 64-bit sub-mult	192	19	48	18
2x 128-bit Sum	267	265		1
5x 32-bit Logic	750			0
15x 70-bit comp	900	14		0
join 32-bit	42088	10137		9
Q19 Total	121654	30936	48	38
Virtex-7 FPGA	433200	866400	2940	

our Virtex-7 FPGA which is also shown in Table 1. Vivado maps multiplication and division operations to built-in DSP blocks. However, flip flop utilization is increased when array partition directive is used aggressively.

Query 14 and Query 19 have more complex control logic than Query 6 because the merge and join units are not completely data parallel, therefore the state machine has to check for the sequential cases. In addition, Vivado does not present any mechanisms to connect FPGA memories to external memory, similar to [3] and manage data transfers like [5]. The easiest solution is to synthesize an AXI interface and connect each accelerator to external memory which is additional work for the designer, but even in this case the data management is still left to the user.

A significant design tradeoff is to bring the columns next to a bitmap versus to send a column made up of pointers to memory. We have applied the first case here, but if the second case is implemented, it might be possible to achieve higher gains. Furthermore, we observed that the merge-join module is our slowest component, which justifies the flurry of research in table join acceleration [9, 8]. We believe that another important aspect that should be looked into is a hardware planner/scheduler. We ported the queries manually and we plan to automate the query planning process and port all TPC-H queries to run on our system in the near future.

4. RELATED WORK

Designing ASICs for database analytics is less flexible but more energy efficient than its software implementation counterpart. The work in [13] represents heterogenous compute tiles which manipulate rows and handle database operations in a coarse grain fashion. In order to manipulate streams of data according to the given query, the authors present spa-

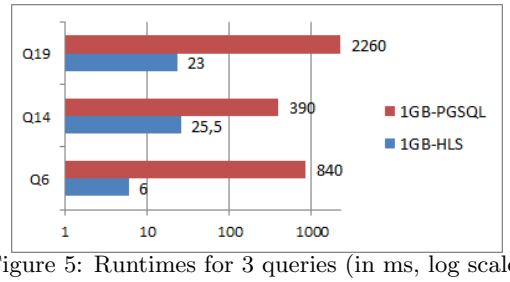


Figure 5: Runtimes for 3 queries (in ms, log scale)

tial and temporal planning which enables/disables compute units. The biggest difference of ASIC design compared to HLS is the flexibility. HLS can adjust to different sizes of blocks such as database columns by extending or shrinking data sizes in high level source code. Hence, it is a more flexible solution for hardware design.

The authors in [12] discuss efficient methodologies for decoupling accelerators from its host. In contrast, our work presents in-memory database acceleration where the memory is controlled by a host system. Thus, in this manner, our accelerator can be classified as a tightly coupled accelerator.

Complementary to our work, authors in [4] discuss implementation challenges of merge sort and join operations. They have designed and customized a merge-sort join implementation for a specific platform and have studied scaling and parallelization capabilities. Our work focuses on using HLS in the database analytics domain without any custom enhancements based on the underlying architecture.

While HLS produces hardware from high level software, Glacier compiles VHDL code from algebraic expressions [11]. This adds additional steps in the system design because algebraic expressions must be created from SQL expressions or they are taken directly from a query planner. Our approach binds accelerators to SQL operators semantically. Then, the query plan is made accordingly. In this work, the query plans are generated manually. The query plan creation is out of scope of this work. We plan to automate our resource allocation the near future.

Runtime query processing has been possible by runtime reconfiguration capabilities of FPGAs. Authors of [7] have built a database operations library which at runtime forms the data path based on the given SQL query. They have focused on data filtering operations. The main advantage of runtime reconfiguration is to eliminate the synthesis of queries, if the available runtime operator library can execute the given query. The flexibility that HLS provides is at compile time rather than runtime. Hence, there is possibility to combine runtime reconfiguration with HLS technology. Although our design could be enhanced by the use of dynamic reconfiguration, it does not necessarily require it, since the static design is able to work with different parameters and arguments, and it can fit in our FPGA.

5. CONCLUSIONS

As our results demonstrate while simulating an in-memory database accelerator, HLS tools can present high performance gains running complete database queries and is a promising way to address the big data explosion. Although designing the most of the database operations are straightforward in HLS, certain issues such as the memory - accelerator communication synthesis might require handwritten integration of HLS hardware and manual data management. In the simulation environment, we have shown between 15–140x speedup compared to Postgres software DBMS running selected TPC-H queries.

6. ACKNOWLEDGEMENT

Funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement No 318633, UPC project TIN2012-34557, the Turkish Ministry of Development under the TAM Project, number 2007K120610 as well as Severo Ochoa Mobility grant program support was received.

7. REFERENCES

[1] K. E. Batcher. Sorting networks and their applications. In *Proc. spring joint computer conference*, pages 307–314. ACM, 1968.

[2] A. Becher, F. Bauer, D. Ziener, and J. Teich. Energy-aware sql query acceleration through fpga-based dynamic partial reconfiguration. In *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, pages 1–8. IEEE, 2014.

[3] Canis and et al. Legup: high-level synthesis for fpga-based processor/accelerator systems. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, pages 33–36. ACM, 2011.

[4] J. Casper and K. Olukotun. Hardware acceleration of database operations. In *FPGA '14*, pages 151–160.

[5] Chung and et al. Coram: an in-fabric memory architecture for fpga-based computing. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, pages 97–106. ACM, 2011.

[6] T. P. P. Council. Tpc-h benchmark specification. *Published at http://www.tpc.org/tpch/spec/tpch2.6.0.pdf*, 2008.

[7] C. Dennl, D. Ziener, and J. Teich. On-the-fly composition of FPGA-based SQL query accelerators using a partially reconfigurable module library. In *Proc. FCCM, IEEE*, pages 45–52, 2012.

[8] Halstead and et al. Accelerating join operation for relational databases with FPGAs. In *Proc. FCCM*, pages 17–20, 2013.

[9] István and et al. A flexible hash table design for 10gbps key-value stores on fpgas. In *FPL*, pages 1–8, 2013.

[10] D. Koch and J. Torresen. Fpgasort: A high performance sorting architecture exploiting run-time reconfiguration on fpgas for large problem sorting. In *FPGA '11*, pages 45–54.

[11] R. Mueller, J. Teubner, and G. Alonso. Glacier: A query-to-hardware compiler. In *SIGMOD '10*, pages 1159–1162.

[12] A. Parashar et al. Triggered instructions: A control paradigm for spatially-programmed architectures. *SIGARCH Comput. Archit. News*, pages 142–153.

[13] L. Wu, A. Lottarini, T. K. Paine, M. A. Kim, and K. A. Ross. Q100: The architecture and design of a database processing unit. In *ASPLOS '14*, pages 255–268.

Comparison of Predictive-Corrective Video Coding Filters for Real-Time FPGA-based Lossless Compression in Multi-Camera Systems

Bart Stukken
Hasselt University
Diepenbeek
Belgium

Yimu Wang
Hasselt University
Diepenbeek
Belgium

Yu Bao
Communication Eng.
Yangzhou University
China

Caikou Chen
Communication Eng.
Yangzhou University
China

Luc Claesen
Hasselt University
Diepenbeek
Belgium

ABSTRACT

Combining multiple cameras in a bigger multi-camera system give the opportunity to realize novel concepts (e.g. omnidirectional video, view interpolation) in real-time. The better the quality, the more data that is needed to be captured. As more data has a direct impact on storage space and communication bandwidth, it is preferable to reduce the load by compressing the size. This cannot come at the expense of latency, because the main requirement is real-time data processing for multi-camera video applications. Also, all the image details need to be preserved for improving the computational usage in a later stage. Therefore, this research is focused on predictive-corrective coding filters with entropy encoding (i.e. Huffman coding) and apply these on the raw image sensor data to compress the huge amount of data in a lossless manner. This technique does not need framebuffers, nor does it introduce any additional latency. At maximum, there will be some line-based latency, in order to combine multiple compressed pixels in one communication package. It has a lower compression factor as lossy image compression algorithms, but it does not remove human invisible image features that are crucial in disparity calculations, matching, video stitching and 3D model synthesis. This paper compares various existing predictive-corrective coding filters after they have been optimized to work on raw sensor data with a color filter array (i.e. Bayer pattern). The intention is to develop an efficient implementation for System-on-Chip (SoC) architectures to improve the computational multi-camera systems.

Categories and Subject Descriptors

B.6.3 [**VHDL, Verilog**]: Language Constructs and Features – *lossless compression, image processing, system-on-chip.*

General Terms

Algorithms, Performance, Design, Experimentation, Theory.

FPGAWorld'15, September 8-10, Stockholm and Copenhagen.

Keywords

multi-camera, lossless compression, Bayer pattern, system-on-chip, SoC, entropy encoding, VLSI.

1. INTRODUCTION

Nowadays, CMOS image sensors are abundantly available due to the rapid evolution in VLSI technology. Many multi-media devices (e.g. smartphones, tablets) contain one or more high-quality cameras. By combining multiple of these cost-efficient cameras, it is possible to realize all kinds of novel concepts such as omnidirectional video and virtual viewpoint interpolation [5,6,7]. If we would extrapolate this trend, hundreds of cameras could work closely together to capture scenes of television plays or sports events in real-time from several fixed positions. The various viewpoints can collect all the information that is needed to synthesize any "virtual" viewpoint between the real cameras.

In a multi-camera system, it is possible to improve the end result by not only improving the recorded quality of each camera, but also to improve the number of (cheaper) cameras. However, this has a direct impact on the storage space and communication bandwidth. For example, if you have five 2MP cameras with 30 fps, the total raw data size that is captured reaches around 900 MB per second. Imagine the impact if the amount of cameras would increase to a hundred, unless the bandwidth requirement is not reduced by some compression, a bottleneck will arise.

Traditional compression standards (e.g. MPEG, H.264) reduce the size by removing redundant information for the human observer. Unfortunately, high frequency content or other image features that are invisible to the human eye, are crucial for computational purposes in multi-camera systems. These systems need to combine the visual data from several image sensors and their quality relies a lot on the accuracy of such high spatial frequency clues for accurate and detailed matching of features captured by the different cameras. For future improvements to disparity calculations, matching, video stitching and 3D model synthesis, all the images features need to be kept intact. This limits this research to lossless compression techniques, although the compression factor is lower than the previously mentioned lossy compression algorithms.

Furthermore, to achieve the lowest possible latency in a future System-on-Chip (SoC) architecture, this research is focused on

predictive-corrective coding filters as it does not need framebuffers to do the lossless compression of the image sensor data. The first step of the encoding sequence can be calculated directly from the active sensor data. The second step using entropy encoding as the compression technique, needs multiple pixels to compress it to one communication package and to gain in the long run. This, however, will introduce line-based latency, but there is no other known technique that preforms better for lossless compression with regards to latency. Lossless image compression algorithms that make use of image tiling will have a higher latency, because the entire tile needs to be captured before it can be send. This puts the minimum latency as high as the height of the tile in scan line numbers, thus latency = tile height × image width.

There are already several well developed predictive-corrective coding filters (e.g. DCPM [1], Paeth [2], GAP [3], JPEG-LS [4]), but they are designed and optimized for full color images. As the goal is to develop a SoC architecture, the filter can be applied on raw sensor data with a color filter array and improve the compression factor for each individual camera in the multi-camera system. In this work, the predictive-corrective coding filters will be adjusted to work on a color filter array (i.e. Bayer pattern) as these kind of cameras are going to be used in future work.

This paper is structured as follows. In the next section the next predictive-corrective coding filters for lossless image compression are explained : DCPM, Paeth, GAP and JPEG-LS. In section III the method for compression, entropy encoding, is elaborated. The results from the applied Huffman coding on the error-corrective values given by the predictive-corrective coding filters, are also displayed in section III. These can be used to compare the results in section IV, in which the algorithms have been updated to work with a color filter array (i.e. Bayer pattern). It is noticeable that the prediction algorithms on Bayer patterned mosaic images work better (i.e. a compression factor of 5.3:1) than the normal RGB compression can reach (i.e. a compression factor of 2:1). Section V concludes with a discussion on the lossless image compression results.

2. PREDICTIVE-CORRECTIVE CODING FILTERS

All the predictive-corrective coding filters try to find the best estimation of a certain pixel, based on previous known pixel values. As the image sensor will transmit the 2D image in a pixel serial order (i.e. left to right for each row and from top to bottom), a prediction can only be done for the next pixel in line. Therefore, none of the future information (i.e. on the right and below the current pixel) can be used, unless there is a framebuffer. But this research will not look into the techniques that use framebuffers, as that is much more memory intensive. Predictive-corrective coding filters only need a limited amount of line buffer, depending on the chosen technique.

Once a estimation of the currently transferred pixel value is done, the correction with the actual sensor value can be calculated. By only transferring these error-corrective values, the original image can still be recalculated lossless in a later stage. This process is fully illustrated in figure 1.

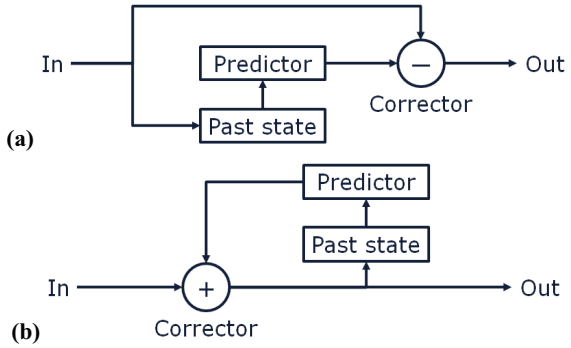


Figure 1. Predictive-corrective coding filter schematic diagram : (a) encoding image to error-corrective data stream, (b) decoding an error-corrective data stream back to the original image.

The better the prediction, the closer the prediction is to the actual value, and the smaller the error is, and the closer the error-corrective value is to zero. By gathering a lot of the same values (i.e. all of the values distributed near zero), it is possible to apply an entropy encoding technique (e.g. Huffman coding) to reduce the overall size.

2.1 DPCM

The 2D form of the Differential Pulse Code Modulation (DPCM) is one of the simplest forms of a predictive-corrective coding filter used for image compression. This technique has been added to the JPEG standard as Lossless JPEG in 1993 [1]. It makes a prediction of the next pixel by applying just one formula that calculates the interpolation of the three previous and neighboring pixels. As shown in figure 2, the prediction value is calculated as follows P = B + C – A. Due to this formula, it is possible to calculate over- and underflow. To keep the values in range, these are clipped within the possible range.

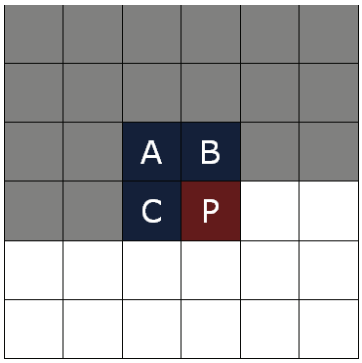


Figure 2. Predictive-corrective coding filters, DPCM and Paeth, use only the three previous and neighbouring pixels as window to calculate a prediction for P.

2.2 Simple predictive-corrective coding

For the PNG Working Group, Alan W. Paeth has proposed an algorithm that has a better prediction result than DPCM [2]. It uses the same window and formula as DPCM (see figure 2) to do the prediction (i.e. three previous and neighboring pixels), but the predicting algorithm is different. It will select a previous pixel value (i.e. A, B or C) that is closest to the calculated DPCM value. In pseudo code, this algorithm would look like the following sequence for each pixel :

```
P = B + C – A;
IF |P – A| ≤ |P – B| AND |P – A| ≤ |P – C|
    P = A; //A is closest to P
ELSE IF |P – B| ≤ |P – A| AND |P – B| ≤ |P – C|
    P = B; //B is closest to P
ELSE
    P = C; //C is closest to P
END
```

2.3 Context-based, adaptive lossless image coding

In response to the request for new techniques for lossless image compression from the JPEG Working Group, a new algorithm has been invented. Xiaolin Wu has proposed a Context-based, Adaptive Lossless Image Coding (CALIC) [3] that also uses a window of previous and neighboring pixels, but it is expanded up to seven previous known values to do a more substantiated prediction. As depicted in figure 3, the prediction has a much bigger window to make a prediction.

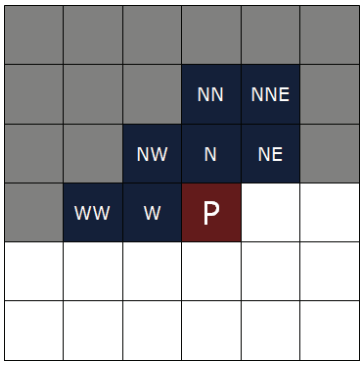


Figure 3. The window that is used by GAP algorithm uses the seven previous and neighbouring pixels to make a prediction for P.

By using more aligned pixel values, it is possible to use a Gradient-Adjusted Prediction (GAP). The GAP algorithm that is used in the CALIC design, also tries to take several different edge types into account. Therefore, it accumulates all absolute vertical changes and compares it with the accumulation of all the absolute horizontal changes. The difference between these two sums of absolute differences, has been parameterized to use different formulas on different types of edges. This is visible in the following pseudo code sequence that will be used for every pixel prediction :

```
Dh = |W – WW| + |N – NW| + |N – NE|; //horizontal
Dv = |W – NW| + |N – NN| + |NE – NNE|; //vertical
Edge = Dh – Dv;
IF Edge > 80 //sharp horizontal edge
    P = W;
ELSE IF Edge > 32 //horizontal edge
    P = W / 2 + (W + N) / 4 + (NE – NW) / 8;
ELSE IF Edge > 8 //weak horizontal edge
    P = W / 4 + 3 * (W + N) / 8 + 3 * (NE – NW) / 16;
ELSE IF Edge < –80 //sharp vertical edge
    P = N;
ELSE IF Edge < –32 //vertical edge
    P = N / 2 + (W + N) / 4 + (NE – NW) / 8;
ELSE IF Edge < –8 //weak vertical edge
    P = N / 4 + 3 * (W + N) / 8 + 3 * (NE – NW) / 16;
ELSE
    P = (W + N) / 2 + (NE – NW) / 4;
END
```

2.4 JPEG-LS

After the first proposition of JPEG-LS as a new Lossless JPEG technique that only detects horizontal and vertical edges and uses a different formula for the prediction, improvements where proposed to implement a diagonal edge-based prediction algorithm [4]. The window that is used for each edge-based prediction, is defined by the four previous and neighboring pixels as displayed in figure 4.

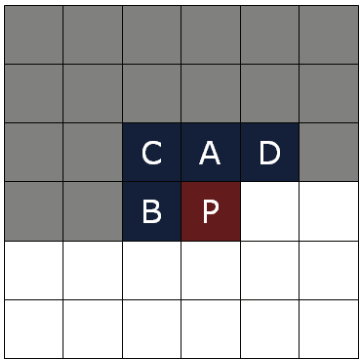


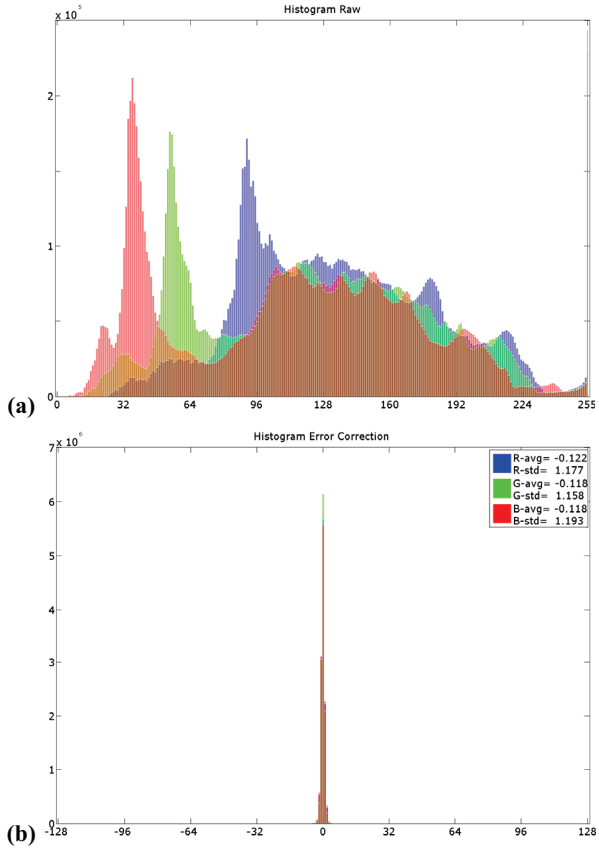
Figure 4. JPEG-LS uses four previous and neighbouring pixel to make an edge-based prediction for P.

Depending on the predefined thresholds, the algorithm will detect different types of edges. It is therefore important to determine thresholds that work well on a broad range of pictures. In this research, the two thresholds (T1 = 60 and T2 = 8) are founded on the results in [4]. Only when the top left pixel C is higher or lower than the two closest neighboring pixels A and B, an edge is detected. In all other cases, old DPCM formula is used to do a prediction. When de difference between C and its neighbors A and B is bigger than threshold T1, and the difference between A and B is not that bigger than threshold T2, a diagonal DPCM formula is used (i.e. P = B + D – A). For the vertical edges, one of the neighboring pixel values A or B is used. The following pseudo code clarifies the logic behind the algorithm, its edge detection and the different prediction formulas :

```
IF C ≥ max( A, B ) //light to dark edge
    IF C – max( A, B ) > T1 AND |A – B| ≤ T2 //diagonal
        P = B + D – A;
    ELSE
        P = min( A, B );
    END
ELSE IF C ≤ min( A, B ) //dark to light edge
    IF min( A, B ) – C > T1 AND |A – B| ≤ T2 //diagonal
        P = B + D – A;
    ELSE
        P = max( A, B );
    END
ELSE //no clear edge
    P = A + B – C
END
```

3. ENTROPY ENCODING

In previous predictive-corrective coding filters, the prediction errors are distributed near zero, which is well-suited for the subsequent coding. To compress these error-corrective values, the most frequent value must use the least amount of space at the expense of the values that are very rare. Therefore, in this research the Huffman coding is used as it is not the focus to compare entropy encoding techniques. Any optimization can always be obtained once the best predictive-corrective coding filters is found. Just using one entropy encoding technique is enough to demonstrate and compare the compression results. Figure 5 shows the histogram of pixel values before and after a predictive-corrective coding filter has been applied.



(a)

(b)

Figure 5. Pixel value histogram : (a) the full color of an image are distributed over the entire value spectrum, (b) the error-corrective values are close to zero.

In order to make a good comparison, all the predictive-corrective coding filters have been applied to the high resolution benchmark images from The *New Image Compression Test Set*[1]. The results of the compression factors after the Huffman coding are shown in table 1. To have a correct baseline the results from only applying Huffman coding is also included (i.e. the "none" filter). The compression factor = original size / compressed size, making the highest number the best compression algorithm.

[1] http://www.imagecompression.info/test_images

Table 1. Compression factors of the benchmark images after lossless compression

Image	None	DPCM	Paeth	GAP	JPEG-LS
artificial	1.271	4.848	5.077	4.775	**5.078**
big_building	1.059	1.845	1.877	1.901	**1.916**
big_tree	1.133	1.679	1.729	**1.776**	1.760
bridge	1.075	1.660	1.684	**1.739**	1.716
cathedral	1.206	1.877	1.942	1.967	**1.974**
deer	1.266	1.362	1.440	**1.489**	1.451
fireworks	2.242	3.461	3.597	**3.650**	3.641
flower_foveon	1.122	2.987	3.197	**3.387**	3.277
hdr	1.135	2.617	2.833	**2.949**	2.885
leaves_iso_1600	1.089	1.574	1.589	**1.630**	1.618
leaves_iso_200	1.083	1.797	1.781	**1.839**	1.829
nightshot_iso_100	1.240	2.886	3.035	3.066	**3.083**
nightshot_iso_1600	1.292	1.692	1.771	1.780	**1.782**
spider_web	1.040	3.917	3.879	4.017	**4.022**
TOTAL	**1.151**	**1.971**	**2.026**	**2.070**	**2.062**

In the comparison above, this research is only comparing the technique for the inside of the picture. All the special border algorithms are being dropped as these small optimizations should not have any influence on the end result. To conclude, the GAP and JPEG-LS are very good candidates for compression, with a small preference to the GAP. The lossless compression factor is around 2:1 for a various kind of images, but as this research is focused on the compression of the hardware image sensor data, the current techniques need to be adjusted.

4. COLOR FILTER ARRAY

In order to capture full color images with an image sensor, many different techniques arose. One of the most used techniques is the use of a color filter array on top of the image sensor. By filtering a specific color for a single light sensor, only that color is captured. Using a fixed color pattern, like the Bayer pattern, the raw sensor data will transmit mosaic images. Afterwards, the full color images can be obtained with the use of demosaicing algorithms. As portrayed in figure 6, each pixel contains originally only one color value.

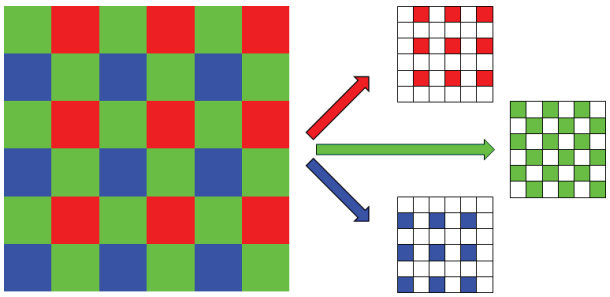


Figure 6. Bayer pattern as color filter array, only one color value is captured per pixel on the raw image sensor before the demosaicing blends them into a full color picture.

This would suggest that the images could have a lossless compression factor of 3:1 when saving the raw mosaic image. The demosaicing algorithm can always be redone and thus makes previous compression algorithms look useless. However, in this research both techniques are being combined. Therefore, the predictions have to be separated for each color and adjusted to

use the correct neighboring pixels. A primary and easy approach would be to just use the same technique, but jump two pixels in each direction. For the red and blue color, this is sufficient. However, the green color could have used its diagonal neighboring pixels to do a better prediction. The results shown in table 2, indicate that the compression can be higher than 2:1 or 3:1 by just using this first approach. Also the "none" filter has been used to provide a better baseline for comparison.

Table 2. Compression factors of the benchmark images after lossless compression with Bayer pattern

Image	None	DPCM	Paeth	GAP	JPEG-LS
artificial	3.805	11.606	**12.754**	11.433	12.624
big_building	3.177	4.496	4.672	**4.740**	4.732
big_tree	3.368	4.142	4.340	**4.474**	4.385
bridge	3.219	4.137	4.302	**4.439**	4.350
cathedral	3.618	4.552	4.769	**4.901**	4.816
deer	3.795	3.877	4.125	**4.283**	4.152
fireworks	6.622	8.457	8.926	**9.211**	8.994
flower_foveon	3.489	7.681	8.230	**8.518**	8.365
hdr	3.426	6.602	7.170	**7.449**	7.256
leaves_iso_1600	3.238	3.958	4.051	**4.148**	4.097
leaves_iso_200	3.228	4.224	4.327	**4.425**	4.383
nightshot_iso_100	3.682	6.783	7.228	**7.421**	7.299
nightshot_iso_1600	3.826	4.507	4.703	**4.840**	4.738
spider_web	3.121	7.566	8.026	**8.406**	8.204
TOTAL	**3.442**	**4.868**	**5.099**	**5.225**	**5.155**

In a first simple approach, it is noticeable that GAP is the better algorithm in combination with the Bayer pattern. It has an average compression factor of 5.2:1 which is much better than the 3:1 or 2:1 discussed before. To be correct, it must be mentioned that for this research the GBRG Bayer pattern (as in figure 6) has been chosen and reverted from the benchmark images. Therefore, the results only give a good indication, but are not the most correct values. Future work will be done on the raw sensor data which has the correct color filter array.

In attempt to study the influences of improving the green prediction algorithm, a few changes has been made to the existing predictive-corrective coding filters. For the DPCM and Paeth, the window has been upgraded to also use the diagonal neighboring pixel as demonstrated in figure 7. The DPCM prediction value is also calculated differently as the average between the 2D and diagonal interpolation. The formula is as follows P = (B + C - A) + (2 * D - A). As Paeth uses this to find the closest neighboring pixel value, the only improvement is that it also can pick the pixel value of D, if that one is closest.

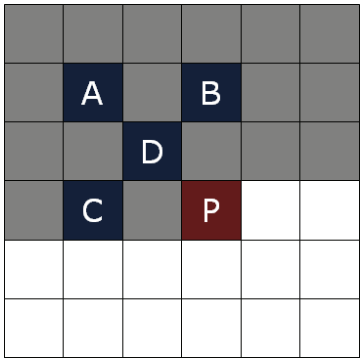


Figure 7. The improved window for the green pixels in mosaic image with a Bayer pattern, used for the predictive-corrective coding filters DPCM and Paeth.

For GAP and JPEG-LS a simple shift has been applied to the rows. While the horizontal neighboring pixels still require a two pixel jump, the vertical pixels are diagonal and require an additional jump to the left when going an odd number upwards. Figure 8 gives a clear view how these two algorithms windows have changed for the green color in a mosaic image.

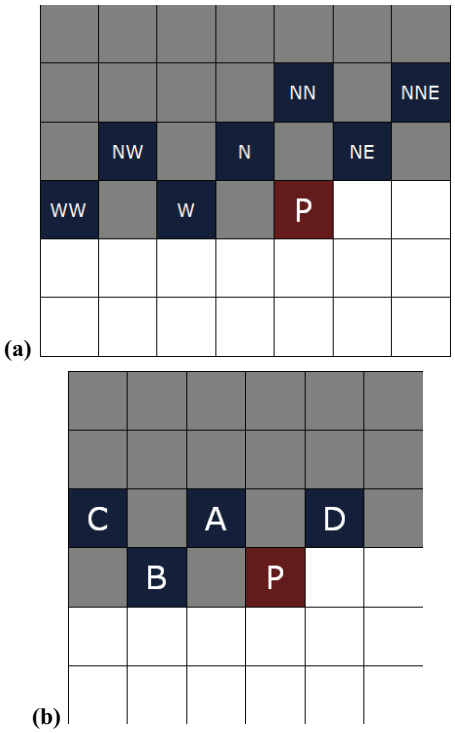


Figure 8. A simple shift applied to the green color windows of mosaic images with Bayer pattern in predictive-corrective coding filters : (a) GAP, (b) JPEG-LS.

These improved predictive-corrective coding filters are being tested on the Bayer patterned mosaic images of the benchmark set. The results are shown in table 3 and can be compared with all previous results.

Table 3. Compression factors of the benchmark images after improved lossless compression with Bayer pattern

Image	None	DPCM	Paeth	GAP	JPEG-LS
artificial	3.805	11,586	**13,210**	11,714	12,867
big_building	3.177	4,559	4,733	**4,857**	4,721
big_tree	3.368	4,206	4,403	**4,556**	4,432
bridge	3.219	4,184	4,337	**4,472**	4,331
cathedral	3.618	4,630	4,849	**5,021**	4,892
deer	3.795	3,918	4,135	**4,290**	4,153
fireworks	6.622	8,613	9,097	**9,490**	9,090
flower_foveon	3.489	7,835	8,346	**8,740**	8,456
hdr	3.426	6,722	7,326	**7,631**	7,284
leaves_iso_1600	3.238	4,032	4,124	4,226	**4,405**
leaves_iso_200	3.228	4,329	4,432	**4,533**	4,119
nightshot_iso_100	3.682	6,879	7,338	**7,555**	7,230
nightshot_iso_1600	3.826	4,563	4,748	**4,893**	4,756
spider_web	3.121	7,791	8,213	**8,720**	8,166
TOTAL	**3.442**	**4,943**	**5,170**	**5,326**	**5,170**

GAP is again the best overall algorithm and has like most other filters a compression factor gain of 0.1, making the end result 5.3:1. Although, this is a better result than previous reached, much improvements can be found in working directly with the color filter arrays. Future work should be done in the direction of improving the predictor with values from the other colors in the color filter array.

5. FPGA IMPLEMENTATION

To put the theory into practice, a real-time implementation is made on an Altera Cyclone-II EP2C70 FPGA equipped with a 5 mega pixel camera module inserted into the GPIO slot. This can be seen on figure 9. This camera module streams its raw Bayer pattern image data to the FPGA core with an implementation of the simple predictive-corrective coding algorithm. Each consecutive clock cycle during the image transfer, a pixel value is retrieved from the image sensor, a prediction is made by using the algorithm of Paeth. The error correction is then obtained by subtracting these two values.

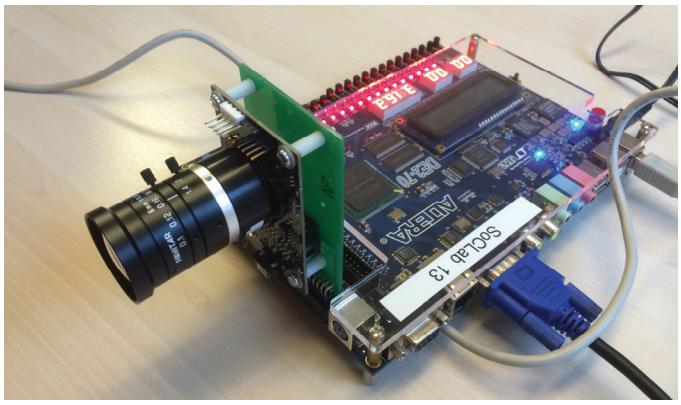


Figure 9.FPGA implementation on a DE2-70 with a D5M camera module for real-time simple predictive-corrective coding.

For demonstration purposes and visible in figure 10, these error corrective values are enhanced and displayed in real-time via the VGA signal onto a connected monitor. These values are also collected and used to calculate and display a histogram. Therefore, it the prediction algorithm is visible and in particular the influences of scenery. For instance, the histogram adapts dynamically and has a wider base on a more complex scene (i.e. harder to do a correct prediction), or has a peak when there is an overflow (i.e. too much lightning can give a big spot with all the same high value). In Table IV an overview of the hardware utilization for the FPGA implementation is summarized.

Table 4. FPGA demonstrator hardware utilization

Characteristic	Used	Total Available	% Utilization
Total logic elements	3,085	68,416	5
Total combinatorial functions	2,355	68,416	3
Dedicated logic regisers	2,021	68,416	3
Total registers	2,065		
Total memory bits	587,256	1,152,000	51
Multipliers	0	300	0

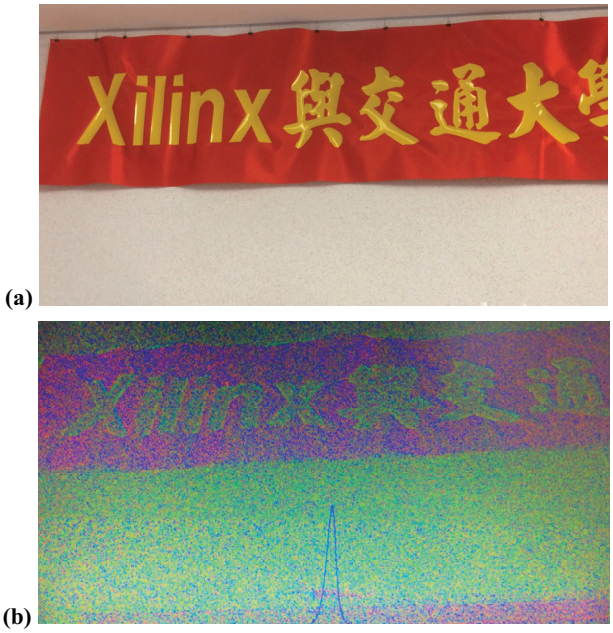


Figure 10. Real-time results of the FPGA implementation : (a) the original scene that is captured by the camera, (b) the error corrective values that are enhanced and the historgram of these values.

6. CONCLUSION

In this paper, it is proven that a synergy can be reached by combining a color filter array (e.g. Bayer pattern) with known predictive-corrective coding filters and an entropy encoding for a higher overall lossless compression of images. The four altered techniques (i.e. DPCM, Paeth, GAP and JPEG-LS) have been examined and compared with the use of a high resolution benchmarking image set. It is concluded that the update GAP algorithm has reached the best overall result. With an average compression factor of 5.3:1 it surpasses the same algorithm on the full color images, which only got a compression factor of 2:1. Nevertheless, there is still room for improvement as many different approaches have not been invented (e.g. Bayer pattern specific algorithms).

For our bigger research goal of implementing a compression algorithm in hardware for multi-camera systems, it already shows to apply these techniques before the demosaicing process. Therefore, in future work, the GAP algorithm will be implemented as a SoC architecture to reduce the bandwidth requirement in multi-camera systems, without compromising the latency. Although, this will not give enough room to have a hundred cameras interconnected, it is a big step forwards to reduce the communication bandwidth a fivefold. Other possible improvements (e.g. software-defined networking) benefit from this approach and are researched in our group in parallel.

The research in this paper was partly funded by the bilateral FWO-MOST (Belgian Research Council research cooperation contract G.0524.13.

7. REFERENCES

[1] Wallace, G.K., "The JPEG still picture compression standard," Consumer Electronics, IEEE Transactions on , vol.38, no.1, pp.xviii,xxxiv, Feb 1992

[2] Alan W. Paeth, "Image File Compression Made Easy", in Graphics Gems II (edited by James Arvo), Academic Press, 1995, ISBN 0-12-059756-X, pp. 93-101.

[3] X. Wu, N. Memon, "CALIC - A context based adaptive lossless codec", Proceedings IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP-96, 7-10 May 1996, Vol. 4, pp. 1890-1893.

[4] Eran A. Edirisinghe, Satish Bedi, Christos Grecos, "Improvements to JPEG-LS via diagonal edge-based prediction.", in Proc. SPIE 4671, Visual Communications and Image Processing 2002, 604 (January 7, 2002);

[5] A. Motten, L. Claesen, Y. Pan, "Trinocular Stereo Vision using a Multi Level Hierarchical Classification Structure", chapter in "VLSI-SoC: From Algorithms to Circuits and System-on-Chip Design", editors: A. Coskun, A. Burg, R. Reis, M. Guthaus, Springer ISBN 978-3-642-45072-3, pp. 45-63.

[6] R. Szeliski, "Computer Vision: Algorithms and Applications", Texts in Computer Science 2011, Springer, ISBN: 978-1-84882-934-3.

[7] Abdulkadir Akin, "Real-Time High-Resolution Multiple-Camera Depth Map Estimation Hardware and Its Applications", Ph.D. Thesis EPFL Lausanne, 2015.

Room: Gasellen
Sessions C1-C2-C3/4
Session Chair: Ove Boström
FPGAworld

§ Unlocking the full potential of 'in-lab' FPGA debug and verification

Using actual FPGA board 'in the lab' for debug and verification is very common to explore debug cases requiring extended system service times and to overcome potential environment modeling issues. However, with the evolution of FPGA complexities, debugging 'in the lab' is sometimes disregarded - mainly because it does not offer sufficient visibility.

This session explores the trade-offs of using in-lab FPGA verification in conjunction with other techniques. It shows when using FPGA 'in the lab' is truly useful to reduce the overall debug and verification process and what has to be improved to do so.

Finally, it introduces EXOSTIVTM, Exostiv Labs' flagship solution, that offers up to 200.000 times more visibility than standard embedded instrumentation during in-lab FPGA debug.

Presenter: Frédéric Leens, CEO*, Exostiv Labs*.

§ Designing power for FPGAs

Analysing the power requirements and planning the voltage rails to get an efficient and reliable solution is one of the key challenges in FPGA design. With this presentation, Linear Technology will show you how _LTpowerPlanner_ helps obtain, at the same time as it documents, the total solution without adding effort. Dynamic load response and output impedance is often the most important but also the most difficult behaviour to analyse; a guide to optimise it will be shown. By attending this seminar, you will learn how to find the best tools and documentation to help you design power for FPGAs.

Presenter: Thomas Ginell and Nicolai Mahncke, *Linear Technology*.

§ UVM Framework – an easy way to improve your verification job

Advanced verification methodologies like UVM (Universal Verification Methodology) enable higher level efficiency and re-usable structure. However many product teams do not take such productivity and quality benefits because they overestimate the ramp-up time required to introduce UVM. In order to increase the time-to-productivity Mentor Graphics created a framework. The so called UVM Framework provides a set of common UVM based testbench building blocks that are ready to use without the necessity of detailed UVM knowledge. In this session you will get a short overview of the UVM Framework followed by a live-demo.

Presenter: Stefan Bauer, *Mentor Graphics*, *InnoFour*.

FPGA™ world

Room: Renen
Sessions A5-A6-A7-A8
Session Chair: Kim Petersén
HDC

§ Accelerating embedded software development and ASIC verification with FPGAs

Two often underestimated use modes of high-density FPGAs are their viability as a verification tool for ASIC verification and as a development tool for embedded software development. This paper provides some insight into why and how FPGAs have become an essential methodology in today's ASIC and SoC development. The focus will be on ASIC-to-FPGA specific transformations like clocks and memories. The main use mode to be discussed is for providing a pre-silicon prototype for early, embedded software development.

Presenter: Juergen Jaeger, *Cadence Design Systems, USA*.

§ OpenCL in an Embedded Environment

With todays FPGA's that have ARM CPU's, comes the possibility to run OpenCL in an embedded environment. OpenCL gives an embedded SW developer the power of custom hardware acceleration without the need for writing low level HDL. It also enables a FPGA developer to be more productive in implementing algorithms. And because OpenCL code can be executed on a PC, implementations can be quickly evaluated and verified. This presentation will describe Alteras OpenCL solution for their SoC FPGA's.

Presenter: Johan Karlsson, *Xelmo AB, Sweden*.

§ New Low Level C programming book for students with Altera BeMicro 10

In an embedded computer system, resources are in general very limited and this is particularly true for memory (data and program memory). C code for embedded systems are characterized by small footprints, bit manipulations of variables, intimate hardware interaction and typical projects are focused on API design, i.e. writing device driver code for user applications. The textbook "Low Level C programming" treats the subject of embedded system programming with a mixture of theory and training (hands on). The training are focused on a non-expensive development board from Arrow (BeMicro MAX 10) and the necessary configuration files (about 10 different) for the FPGA can be downloaded from webpage for the different training.

Presenter: Lars Bengtsson, *University of Gothenburg, Sweden*.

§ The FPGA on a Printed Circuit Board

Routing out from a «FPGA (BGA)» related to BGA pitch. Design suggestions for BGA (0.8mm , 0.5mm and 0.4mm pitch). HDI Design rules

- Different HDI via-structures
- ALIVH (Anylayer Inner Via Hole)
- Stacked & Staggered mVias, Buried vias, Filled & Capped vias Capabilities
- Decide IPC Class (2 or 3) before you start your layout.
- Impedance requirements and considerations.
- The importance of a net-list in PCB production.

Presenter: John Steinar Johnsen, *Elmatica AS, Norway*.

Room: Räven
Sessions A9/10-A11-C1-C9
Session Chair: David Källberg
FPGAworld

§ OSVVM for VHDL Verification

Open Source VHDL Verification Methodology (OSVVM) is a comprehensive, advanced VHDL verification methodology that adds randomization (constrained and intelligent), and functional coverage.

This presentation provides an in depth overview of these capabilities. Like UVM, OSVVM is a library of free, open-source code. However, OSVVM goes a step beyond other verification languages with its Intelligent Coverage methodology that is currently the only portable, language-based, intelligent testbench solution.

Presenter: Jim Lewis, *SynthWorks Design Inc, USA*.

§ The IP-component I2C_Master_IP

An I2C Master IP-component based on VHDL-code from eewiki.net has been developed with Altera's tools. The majority of the logic is implemented in hardware, the software driver basically consists of simple read and writes. Additionally a software library with functions for communicating with the Digital Temperature Sensor MCP9808 has been developed. An example application is demonstrated with Altera's development board DE2-115.

Presenter: Lars Högberg, *AGSTU School, Sweden*.

§ Microsemi

More information later.

Room: Gasellen
Sessions C5-C6/7-C8
Session Chair: Ove Boström
FPGAworld

§ Synecitve Labs

More information later.

§ Optimizing HW/SW partition of a complex embedded system using C/C++

More information later.

Presenter: Olivier Tremois, *EMEAI DSP Specialist FAE, Silica, USA*.

§ Arrow (Altera)

More information later.

FPGA™ world

Room: Storken
Session D1
Tutorial

§ Universal VHDL Verification Methodology (UVVM)

Verification is 51% of total FPGA development time (cf 'Functional Verification Study' from Wilson Research Group). Even more for control or protocol oriented design.
Ignore this and you will iterate forever in the lab and suffer from bad product quality. The root cause is corner cases triggered by certain input and FSM combinations. Hitting a given corner case in normal simulation is very unlikely, - but in the final product it will hit you hard. A structured VHDL approach to verifying such corner cases has not been commonly available. UVVM (Universal VHDL Verif. Meth.) changes this picture completely.

Over the last few years VHDL 2008 and libraries like Bitvis Utility Library (BVUL), OVL and OSVVM has speeded up VHDL testbench development significantly, and at the same time allowed a major quality improvement.

So far however, there has been no proper support for handling the actual testbench structure when controlling multiple interfaces. For multiple simultaneously active interfaces, this could be really critical when it comes to verify cycle related corner cases. Even a simple UART may have several such hard to verify corner cases, and in most projects they are not verified or tested at all - resulting in a product that may have serious corner case bugs. And this even for a simple UART....

There are libraries available that do support control of simultaneously active interfaces, but they are either rather unstructured, very complicated to use or difficult to maintain. Due to their synchronization mechanisms they also make it really hard to get a proper overview of the testbench and test cases.

This presentation will show a typical corner case bug, the typical approaches to finding such bugs, and a structured approach to increase productivity. This includes coding efficiency, potential reuse, overview, readability, extendability, maintainability, debug-friendliness and a lowest possible user threshold.

UVVM is not only about testbench structure. It also provides great progress information at all levels and it allows both directed tests and constrained random and coverage based testbenches.

This tutorial will explain how UVVM may help you reduce your verification time significantly - and at the same time improve your product quality.

Corner cases in the specification and your implementation often result in lots of FPGA bugs. Most of these bugs are very difficult to detect in normal simulation or lab test, and very often quite a few of these bugs are not detected until experienced by the customer.

UVVM is a complete VHDL based verification environment platform that will cover all the most important aspects of FPGA verification. We will show how this works and how you can use this platform to make your own testbench and your own structured verification components based on the examples provided. As a simple example it took only one single hour to make a UART verification component from scratch (but based on a simple bus interface verification component and UART BFMs. This new UART verification component could then be used in a very structured and reusable verification environment.

Please note that this presentation assumes that you have attended the presentation 'The critically missing VHDL testbench feature - Finally a structured approach' in track A4 earlier today.

UVVM will be released very soon after FPGAworld.

Presenter: Espen Tallaksen, *Bitvis*.

FPGAworld 2015 @ Copenhagen

Technical University of Denmark
SCION, Building 372, Diplomvej Lyngby
Denmark

Conference Program

08:30 Registration *(Reception)*.

09:00 Conference opening.
Lars Dittmann, Technical University of Denmark
and *Lennart Lindh, FPGAworld.*

09:15 Key Note Session.
Hardware Security and FPGAs: Strategies and Counterattacks
Vincent Mooney III, Georgia Institute of Technology, Atlanta, USA.

10:00 Coffee break, sponsor ALTERA.

10:30 Parallel Sessions.

12:30 Lunch break & Exhibition.

13:30 Parallel Sessions.

15:00 Coffee break.

15:30 Parallel Sessions.

16:00 Panel Discussion.
What will be the impact of Intel on Altera?
Any reason to be frightened? Xilinx next?
Session Moderator: Rolf Sylvester-Hvid, *Aktuel Elektronik*.

The exhibition will be open during the day.
Coffee will be served in the exhibition area.

FPGA™ world

Key Note Session @ Copenhagen
Speaker: Vincent Mooney III
Georgia Institute of Technology, USA

§ Hardware Security and FPGAs: Strategies and Counterattacks

Vincent Mooney III, Georgia Institute of Technology, Atlanta, USA.

Recent highly publicized security attacks on businesses and governments have heightened the awareness of hardware vulnerability to malicious attacks. FPGA technology offers a unique strategy not available to application-specific non-programmable hardware: reconfiguration. This talk will give an overview of recent results in hardware security including an approach based on hardware signatures. The ability of hardware reconfiguration to protect run-time hardware and software highlights the advantages of FPGAs.

Vincent Mooney III (Senior Member, IEEE, and Member, ACM) received the B.S. degree from Yale University in 1991, where he double majored in Electrical Engineering and Computer Science. He was a member of the 1989 Ivy League Championship football team for Yale and was one of 29 American football players in the United States to be awarded the NCAA Postgraduate Scholarship upon his graduation in 1991. He received an M.S. degree in E.E. from Stanford University in 1994, an M.A. degree in Philosophy from Stanford in 1997, and the Ph.D. degree in E.E. from Stanford in June of 1998. He has worked at Bell Labs (Lucent), Allied Signal Aerospace VLSI Design Group, Hughes Network Systems, and Redwood Design Automation (acquired by Cadence). He is currently an Associate Professor in the School of Electrical and Computer Engineering and an Adjunct Associate Professor in the School of Computer Science, both at the Georgia Institute of Technology in Atlanta, GA. He is a recipient of the NSF Career Award. He serves on the Faculty Council of the Georgia Tech Institute for Information Security & Privacy (IISP). He has served as Program Chair of IFIP VLSI-SoC, CASES and FPGAworld. He was General Chair of IFIP VLSI-SoC 2007. He has served as an Associate Editor of both the IEEE Transactions on VLSI as well as the ACM Transactions on Embedded Computing Systems. His research interests include computer-aided design of integrated circuits with a particular emphasis on hardware-software codesign, reconfigurable computing, hardware security, and power-aware and probabilistic architectures and circuits.

§ Optimizing HW/SW partition of a complex embedded system using C/C++

More information later.

Presenter: Olivier Tremois, *EMEAI DSP Specialist FAE, Silica*.

§ Arrow (Altera)

More information later.

§ Designing power for FPGAs

Analysing the power requirements and planning the voltage rails to get an efficient and reliable solution is one of the key challenges in FPGA design. With this presentation, Linear Technology will show you how _LTpowerPlanner_ helps obtain, at the same time as it documents, the total solution without adding effort. Dynamic load response and output impedance is often the most important but also the most difficult behaviour to analyse; a guide to optimise it will be shown. By attending this seminar, you will learn how to find the best tools and documentation to help you design power for FPGAs.

Presenter: Thomas Ginell and Nicolai Mahncke, *Linear Technology*.

§ UVM Framework – an easy way to improve your verification job

Advanced verification methodologies like UVM (Universal Verification Methodology) enable higher level efficiency and re-usable structure. However many product teams do not take such productivity and quality benefits because they overestimate the ramp-up time required to introduce UVM. In order to increase the time-to-productivity Mentor Graphics created a framework. The so called UVM Framework provides a set of common UVM based testbench building blocks that are ready to use without the necessity of detailed UVM knowledge. In this session you will get a short overview of the UVM Framework followed by a live-demo.

Presenter: Stefan Bauer, *Mentor Graphics*, *InnoFour*.

Sessions A1-A2-A3
Session Chair: David Källberg
FPGAworld

§ Automotive ADAS feature saves lifes with flexible HW (FPGA etc.) – practical implementation reference

Automobile accidents are currently killing 1.25 million people per year worldwide and that figure is rising. The new mantra is "avoiding collisions."

Consumer surveys show that people want safety features in cars, that these safety features sell cars, and that interest in ADAS features is actually higher than for any other electronics-based automotive category.

FPGA enables ADAS systems, saving lifes in the traffic. A reflection of present system

Authors: Tryggve Mathiesen, *Qamcom R&T AB, Sweden*.

§ Accelerating embedded software development and ASIC verification with FPGAs

Two often underestimated use modes of high-density FPGAs are their viability as a verification tool for ASIC verification and as a development tool for embedded software development. This paper provides some insight into why and how FPGAs have become an essential methodology in today's ASIC and SoC development. The focus will be on ASIC-to-FPGA specific transformations like clocks and memories. The main use mode to be discussed is for providing a pre-silicon prototype for early, embedded software development.

Authors: Juergen Jaeger, *Cadence Design Systems, USA*.

§ High-speed FPGA-based stereovision system - a success story

The presentation concerns an industrial stereovision system based on the Xilinx Zynq 7020 FPGA SoC which combines Linux on ARM and with surrounding reconfigurable logic that works as a hardware accelerator. The system is capable of working in real-time with ~50 FPS in VGA and above 65 FPS in lower resolutions. The architecture includes sensor control logic, demosaicer, rectifier and stereovision cores

Authors: Rafal Kapela, Ph.D., *Antmicro Ltd/Poznan University of Technology, Poland*.

Sessions A4-A5-A6
Session Chairs: Ove Boström and David Källberg, *FPGAworld*

§ OSVVM for VHDL Verification

Open Source VHDL Verification Methodology (OSVVM) is a comprehensive, advanced VHDL verification methodology that adds randomization (constrained and intelligent), and functional coverage.

This presentation provides an in depth overview of these capabilities. Like UVM, OSVVM is a library of free, open-source code. However, OSVVM goes a step beyond other verification languages with its Intelligent Coverage methodology that is currently the only portable, language-based, intelligent testbench solution.

Presenter: Jim Lewis, *SynthWorks Design Inc, USA*.

§ Real-time operating system, hardware or software?

One important purpose of a real-time operating system is to present a result in right time. The question is how much time can be saved using a real-time hardware OS accelerator in hardware, compared to today's software kernel in a FPGA device with NIOS processor? The hardware based realtime kernel is designed in VHDL with a thin software device drivers. The software kernel used is FreeRTOS and the hardware kernel used is Sierra with exact same hardware architecture.

Presenter: André Norberg, *AGSTU, Sweden*.

§ Product presentation: Microsemi

More information later.

Call for FPGAworld Conference 2016

Academic/Industrial Papers, Product Presentations, Exhibits and Tutorials
September 2016, **Stockholm**, Sweden, Academic & Industrial programs
September 2016, **Copenhagen**, Denmark, Industrial program only

Submissions should be at least in one of these areas

- DESIGN METHODS - MODELS AND PRACTICES
  - Project methodology
  - Design methods as Hardware/software co-design
  - Modeling of different abstraction
  - IP component designs
  - Interface design: supporting modularity
  - Integration - models and practices
  - Verification and validation
  - Board layout and verification
  - Etc.

- TOOLS
  - News
  - Design, modeling, implementation, verification and validation
  - Instrumentation, monitoring, testing, debugging, etc.
  - Synthesis, compilers and languages
  - Etc.

- HW/SW IP COMPONENTS
  - New IP components for platforms and applications
  - Real-time operating systems, file systems, internet communications
  - Etc.

- PLATFORM ARCHITECTURES
  - Single/multiprocessor architecture
  - Memory architectures.
  - Reconfigurable Architectures
  - HW/SW architecture
  - Low power architectures
  - Etc.

- APPLICATIONS
  - Case studies from users in industry, academic and students
  - HW/SW component presentation
  - Prototyping
  - Etc.

- SURVEYS, TRENDS AND EDUCATION
  - History and surveys of reconfigurable logic
  - Tutorials
  - Student work and projects
  - Etc.