Application Development with the FlexWAFE Real-time Stream Processing Architecture for FPGAs

AMILCAR DO CARMO LUCAS, HENNING SAHLBACH, SEAN WHITTY, SVEN HEITHECKER and ROLF ERNST
Institute of Computer and Communication Network Engineering
Technical University of Braunschweig, Germany
{lucas | sahlbach | whitty | heithecker | ernst}@ida.ing.tu-bs.de

The challenges posed by complex real-time digital image processing at high resolutions cannot be met by current state-of-the-art general purpose or DSP processors, due to the lack of processing power. On the other hand, large arrays of FPGA-based accelerators are too inefficient to cover the needs of cost sensitive professional markets. We present a new architecture composed of a network of configurable flexible weakly-programmable processing elements, **Flex**ible **W**eakly-programmable **A**dvanced **F**ilm **E**ngine (FlexWAFE). This architecture delivers both programmability and high efficiency when implemented on an FPGA basis. We demonstrate these claims using a professional next generation noise reducer with more than 170G image operations/s at 80% FPGA area utilization on four Virtex II-Pro FPGAs. This paper will focus on the FlexWAFE architecture principle and implementation on a PCI-Express board.

Categories and Subject Descriptors: C.3 [**S**pecial-purpose and application-based systems]: Signal processing systems; C.3 [**S**pecial-purpose and application-based systems]: Real-time and embedded systems; C.1.3 [**P**rocessor architectures]: Data-flow architectures; B.6.1 [**L**ogic Design]: Logic arrays; D.3.2 [**P**rogramming Languages]: Specialized application languages

General Terms: Design, Performance

Additional Key Words and Phrases: communication centric, communication scheduling, digital film, FPGA, QoS, PCI-Express, real-time, reconfigurable, SDRAM-controller, stream-based architecture, weakly-programmable

1. INTRODUCTION

Post-production for high definition video and digital film processing in general requires real-time or close to real-time behavior in order to allow immediate feedback necessary for interactive processing. This represents a significant challenge, since the algorithms employed in each step of a digital film processing chain tend to be highly computationally intensive. Performance requirements are far beyond what current Digital Signal Processors (DSP) or general purpose processors can provide, especially with large image sizes of 2K (resulting in 12 MBytes of data per image) and beyond. Consequently, typical state-of-the-art products in this low-volume, high-price market use Field Programmable Gate Array (FPGA) based hardware systems with fixed configurations dedicated to a specific algorithm. This implies that multiple hardware platforms are required to implement the complete post-production processing chain.

For future products, however, this traditional development approach is no longer ideal, due to ever-growing computation demands and algorithm complexities. A novel idea is to

develop a common platform for all processing steps of the post-production chain. Such a platform must be reconfigurable to support each of the required algorithms. It should also take advantage of modern FPGA development techniques borrowed from Application Specific Integrated Circuit (ASIC) design, such as IP reuse and floorplanning. Multi-chip and multi-board systems will also require a sophisticated communication infrastructure and communication scheduling. Furthermore, large external memory space holding several frames is of major importance since the embedded FPGA memories are too small; if not carefully designed, external memory access will become a bottleneck.

As an answer to these challenges, the FlexFilm project introduced a multi-board, multi-FPGA hardware/software architecture based on Xilinx Virtex-II Pro FPGAs. The FPGAs contain the reconfigurable image stream processing data path, large external SDRAM memories for multiple frame storage, and a PCI-Express (PCIe) communication backbone network. This architecture supports a wide variety of real-time applications that can be rapidly developed using a custom-designed component library named **Flex**ible **W**eakly-programmable **A**dvanced **F**ilm **E**ngine (FlexWAFE).

The FlexWAFE library supports varying levels of configurability and reconfigurability, with some of the library blocks' parameters set at synthesis time via Very High Speed Integrated Circuit HDL (VHDL) generics and others configured at run-time. This combination allows increased flexibility without sacrificing FPGA area or speed.

This paper is organized as follows. The FlexFilm[1] board is reviewed in Section 2 and the FlexWAFE component library is explained in detail in Section 3. Several example applications are presented as in Section 4, including a 2.5 dimension noise reduction application using bidirectional motion estimation/compensation and wavelet transformation operating at 26 frames per second (fps). Finally, Section 5 concludes this paper.

1.1 Related Work

In several projects, various design methodologies for building FPGA based applications have been proposed. In the 1990's, FPGAs were used to couple a SIMD processor to a processing array [Cloutier et al. 1996]. Another approach splits the applications into simple operators, which are mapped to a FPGA [Crookes et al. 2000]. Today, there are various language extensions for high-level languages [Najjar et al. 2003; Loo et al. 2002] with corresponding compilers that convert a high-level program into its VLSI representation. For high-end applications, these approaches are not suitable, as the compilers are currently not able to fully optimize memory accesses by exploiting locality, which results in a significant decrease of performance.

Commercial vendors [Nallatech Ltd 2007; Hunt Engineering Ltd] often combine a library of HDL components with hardware products they sell. The HDL libraries usually lack flexibility, as they are optimized for the vendors' products and are therefore technology-dependent. Another approach [Guo et al. 2006] is a composition of applications by connecting wrapped IP cores using a high-level description. Each of these projects concentrate on the design flow. Weak programmability of the components is not taken into account. Other commercial modeling tools like Xilinx System Generator [Xilinx Inc. 2008] or Mathworks Simulink [The MathWorks 2008] offer a GUI-based composition of FPGA designs but are limited in flexibility, which results in suboptimal implementations that are insufficient for the targeted application domain.

[1] This design won a *Design Record Award* at the DATE 2006 conference [do Carmo Lucas et al. 2006]

In order to mitigate the design complexity of FPGA applications, a coarse grain overlay [Shukla et al. 2005] and a corresponding design flow [Shukla et al. 2007] for FPGAs have been introduced. As the data width of the coarse grain processing elements can only be changed between 8 and 16 bits, there is a design complexity vs. flexibility trade-off, which results in a significant area overhead when processing uncommon data widths like 10 bits per color components. Thus a coarse grain solution is not adequate for high-end applications that require optimized data structures for an efficient implementation.

Several other projects also present solutions similar to specific FlexWAFE components. In [Shukla et al. 2007], a similar two-level control hierarchy for a coarse grain array is introduced. In contrast to the flexible three-level FlexWAFE solution the local controllers are a fixed part of processing elements, which leads to a waste of chip area for arrays of simple but numerous processing elements that do not require local control.

For intra-chip communication, a Network-on-Chip (NoC) based approach is presented in [Kumar et al. 2007]. In that architecture, a flat control hierarchy is introduced, and local high-speed control is omitted.

For inter-block signaling a similar protocol is presented in [Hoover and Brewer 2008]. However, this protocol only allows the buffering of one token per link, which excludes complex activation patterns.

Concerning the memory controller, numerous different implementations exist. The controllers by Lee et al. [Lee et al. 2005] and Sonics [Sonics Inc. 2005; Weber 2001] are most similar to the FlexWAFE controller. A detailed study dealing with various memory controllers and their properties can be found in [Heithecker et al. 2007].

Regarding digital image processing, target platforms other than FPGAs are feasible. First of all several ASIC [Thomson GrassValley 2008; Digital vision AB 2008; Quantel Ltd. 2008] implementations exist, specialized to a certain application and therefore offering no hardware reuse.

A second alternative are hardware accelerators similar to the FlexFilm board, which are embedded in a standard PC system. An example is da Vinci's Resolve accelerator [Da Vinci Systems 2008], which is based on a field of 4096 parallel single bit processing units working as a vector processor called ASProCore [Aspex Ltd. 2008].

Storm-1 [Stream Processors Inc. 2008], a spin-off of the Imagine [Ahn et al. 2004] project, offers a field of up to 16 programmable processing arrays consisting of DSP-like processing elements with local memories. The array operates at 700 MHz, reaching a peak performance of 220 GOPS, which is comparable to the performance of a FlexFilm board operating at 125 MHz. These two architectures target the same market segment as the FlexFilm board.

Finally, recent graphics processors [Blythe 2008] and the Cell broadband engine [Kahle et al. 2005] have entered the image processing segment. With impressive computational power and fast memory accesses resulting from high clock frequencies, these platforms offer serious alternatives for digital signal processing. Compared to the FlexWAFE architecture, they both lack a flexible and composable control hierarchy, and their granularity is limited to a fixed word width.

1.2 Technology Status

Current FPGAs achieve up to 500 MHz, have up to 10 Mbit embedded RAM, 192 18-bit MAC units and provide up to 270,000 flipflops and 6-input lookup-tables for logic implementation (source Xilinx *Virtex-V* [Xilinx Inc. 2008]). With this massive amount of

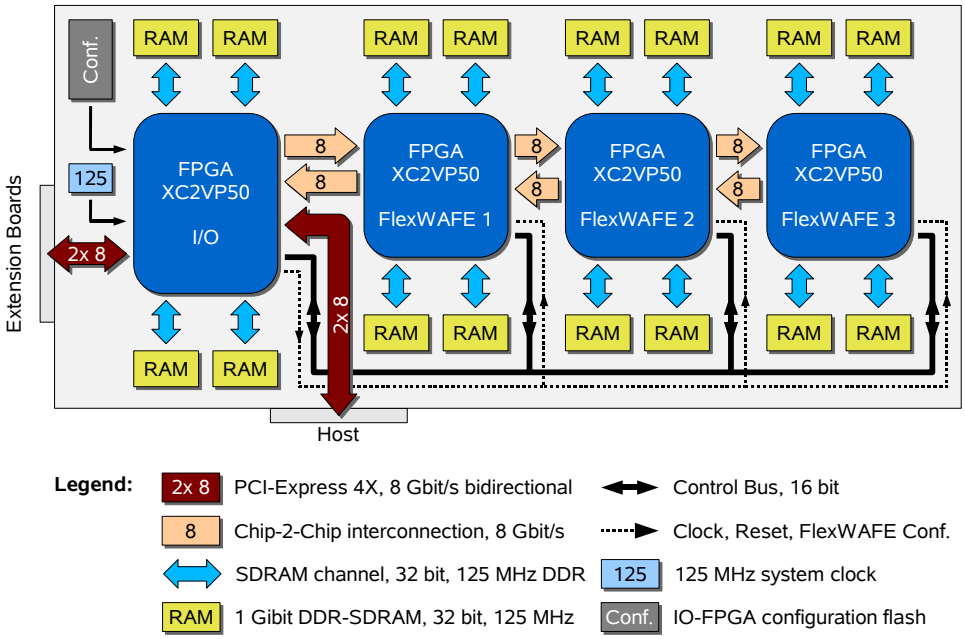


Figure 1: FlexFilm board (block diagram)

resources, it is possible to build circuits that compete with ASICs regarding performance, but have the advantage of being configurable and, thus, reusable.

2. FLEXFILM BOARD ARCHITECTURE

In an industry-university collaboration, a multi-board, extendable FPGA based system has been designed. The FlexFilm board has been presented in [do Carmo Lucas et al. 2006], and its block diagram is shown in Figure 1 for convenience.

The FlexWAFE FPGAs on the FlexFilm board can be reprogrammed on-the-fly (300 ms per FPGA) at run-time by the host computer via the PCIe bus. This allows the user to easily change the functionality of the board, therefore enabling hardware reuse by allowing multiple algorithms run one after the other on the system. Complex algorithms can be dealt with by partitioning them into smaller parts that fit the size of the available FPGAs. Subsequently, either multiple boards are used to execute the algorithm fully parallelized, or a single board is used to execute each one of the processing steps in sequence following a reprogramming of the FPGAs after each step. Furthermore, these techniques can be combined by using multiple boards and sequentially changing the programming on some/all of them, thus achieving more performance than with a single board but without the cost of the full parallel solution.

FPGA partial-reconfiguration techniques were not used due to the reconfiguration time penalty that they incur. To achieve some flexibility without sacrificing speed, weakly-programmable optimized IP library blocks were developed. Using this weakly-programmable library described in Section 3, several applications were developed, each running on a single FlexFilm board. These example algorithms do not require the FPGAs to be reprogrammed at run-time because they do not need more than the three available FlexWAFE

FPGAs. However, tests were made with two boards running one application each, and real-time was also achieved.

3. FLEXWAFE RECONFIGURABLE ARCHITECTURE

An important criterion to characterize a hardware architecture is the instruction set it implements. General purpose processors have a complete instruction set, which guarantees high flexibility but comes at the expense of area and complexity (running speed). Weakly-programmable architectures, on the other hand, offer a very limited instruction set but can be realized using a simple interconnect and with small area foot print. Therefore, these architectures are especially suitable for image processing algorithms, which only require a small set of operations and whose interconnect can be easily expressed.

The weakly-programmable FlexWAFE architecture was designed for image stream processing, following a graph-based compositional design approach that will be detailed in Subsection 3.1. It consists of different types of components, which have been introduced in [do Carmo Lucas and Ernst 2005] and will be presented in detail in the following Subsections.

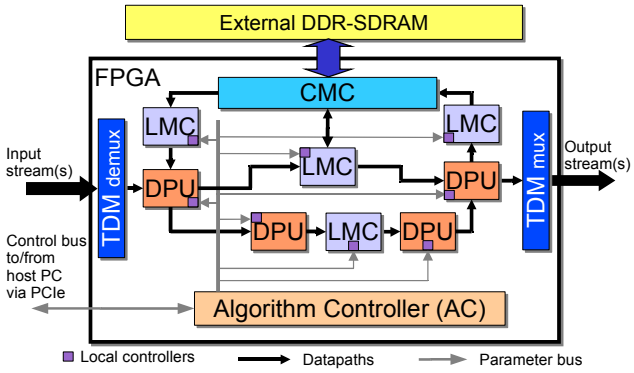


Figure 2: Generic example of the FlexWAFE reconfigurable architecture

In Figure 2, a generic example of the architecture is shown. An input stream enters the FPGA chip via the time division multiplexing (TDM) based stream demultiplexer (Subsection 3.5) on the leftside of the Figure, gets processed by the Data Processing Units (DPU) (Subsection 3.2) along its data path, and finally leaves the chip via the TDM based stream multiplexer on the right of the Figure. The processing units are either connected directly to each other or are intercepted by Local Memory with Controllers (LMC) (Subsection 3.3) that provide memory services like reordering buffers, access at external SDRAM or scratch pad memories. Accesses to off-chip memory are realized via a custom memory controller (CMC in Subsection 3.4). The DPU and LMC modules are locally controlled by their respective Local Controllers (Subsection 3.6.1) that are weakly-programmable by an Algorithm Controller (AC) (Subsection 3.6.2), which sends run-time configurable instructions, thereby controlling the global algorithm sequence and synchronization. The algorithm controller itself is configured by a control bus (see Section 3.6.3) that connects to the host PC via the I/O FPGA controlling the PCIe link (see Figure 1).

All components are implemented as technology independent VHDL modules and are optimized for speed. Other optimization factors like chip area or power consumption were not considered, as the targeted application domain of high-end image processing relies on large FPGAs, which are operated in standard workstations. The components can be parameterized in data word and address length, supported address, data functions, etc. via VHDL generics at design time. All VHDL modules were designed with portability in mind using techniques like inference instead of instantiation for specific FPGA parts like block RAMs. This portability of the architecture will be verified by the introduction of a new hardware platform in the near future.

The described architectural approach is well suited for the processing of non data-dependent streams but can also be used for algorithms with data dependencies like the one shown in Section 4.1.

3.1 Inter-block Signaling

The global data flow follows the synchronous data flow model (SDF, [Karp and Miller 1966; Lee and Messerschmitt 1987]): the system is modeled as a directed graph, where the nodes represent processing elements and the arcs represent communication channels for data tokens. In the general SDF model, arcs are communication channels with infinite buffer length. In the FlexWAFE implementation, the length must be adapted to the real needs and is user configurable. Node execution is enabled if enough tokens are available at their input arcs and the Local Controller (see subsubsection 3.6.1) is enabled. FlexWAFE supports colored tokens by attaching control information together with the token data.

Available data at processing element inputs is signalled by *valid* signals; data is processed only if all inputs are valid. With this technique, a blocking-read behavior is implemented. Consuming of tokens is indicated by the *enable* signal, which in turn lets the preceding processing element produce a new data token, thereby realizing blocking write transactions. This *back pressuring* of the *enable* signal to the preceding blocks eliminates the need for buffering at every arc and allows smooth and reliable starting and stopping of the overall operation.

When designing a new system, there might be a need to reuse code from existing blocks that do not have *valid* and *enable* signals (blocking-read/blocking-write behavior). The FlexWAFE library addresses this issue by providing two blocks that add this functionality. This is shown in Figure 3a, where block B shall be integrated with block A and C from the FlexWAFE library, has a processing latency of n clock cycles, and does not provide a *valid* control output.

Our library provides a *forwardsync* block (block FS in Figure 3b) that produces the missing *valid* control output, based on the n latency parameter of block B. It basically delays the *valid* input n cycles to produce the *valid* output.

Block B on Figure 3c is an example of a block that does not provide the *enable* signal. To integrate it a *backpressure* block can be used, which again is configured with a parameter n that is the maximum latency of block B. This extra block is basically a write-through optimized FIFO of depth n and is depicted as block BP on Figure 3d. As soon as block C disables its input, BP captures the data continuously sent by block B and disables block A, thereby stopping the preceding processing chain.

The *forwardsync* and the *backpressure* blocks can transform an existing non-blocking blockset into a blocking-read/blocking-write system by just adding simple, configurable, non-intrusive blocks that have been optimized for area and speed although written in tech-

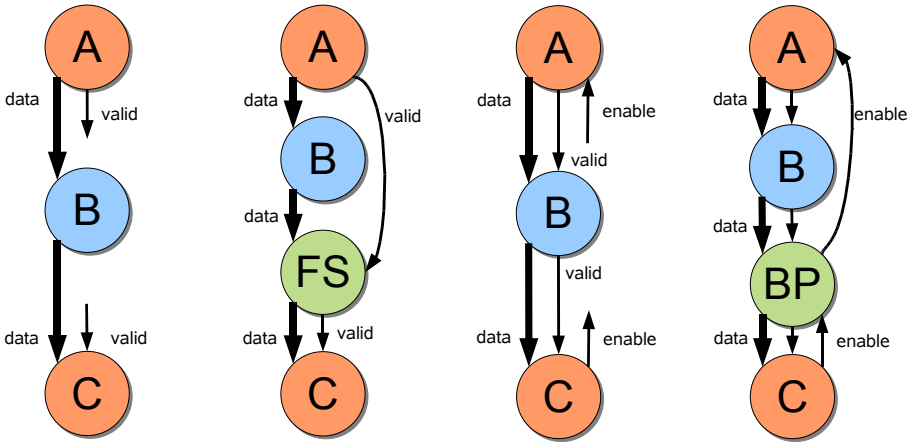


Figure 3: (a) Block B does not provide a *valid* signal; (b) Block FS allows blocking-read operation of block C; (c) Block B does not provide a *enable* signal and; (d) Block BP allows blocking-write operation of block A

nology independent VHDL.

3.2 DPU- Data Processing Unit

The functionality of an image processing application is implemented by partitioning the application into data processing units. These units (graph nodes in the previous Subsection) do the computational tasks and are equipped with multiple input and output ports (graph arcs in the previous Subsection), which allow the realization of blocks with generic complexity. Furthermore, DPUs typically possess local cache-like memories to accelerate their operations, thereby increasing the overall system's performance.

A typical example for a DPU is a Finite Impulse Response (FIR) filter depicted in Figure 4, which is a common element of image processing algorithms. The filter coefficients are configurable by the Local Controller (see Section 3.6.1) and can be exchanged during runtime, which allows the implementation of adaptive filters. Such filters are useful when wildly divergent images (e.g. scene changes between day and night) require different filter coefficients.

Some DPUs include information for relative placement on the FPGA. This allows the construction of computing grids of simple processing elements without floorplanning each DPU separately. The following list shows some DPU examples with varying complexity that are provided by the FlexWAFE library:

—adder

—RGB -> YCrCb

—soft shrinkage

—discrete wavelet transform

—histogram

—motion estimation

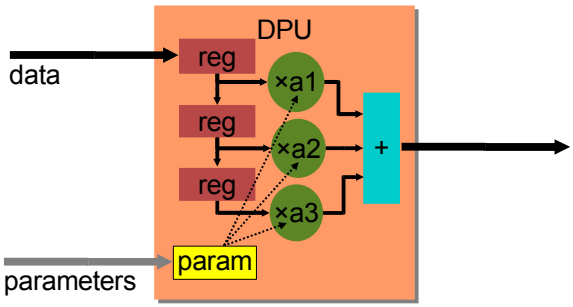


Figure 4: A FIR filter DPU

As the DPUs highly depend on the underlying algorithms, the implementation of new DPUs will be required when realizing new algorithms for the FlexWAFE library.

3.3 LMC- Local Memory with Controller

As described in Section 3.1, the communication channels of FlexWAFE have a buffer size of zero tokens. For buffering, explicit buffer nodes are introduced in the graph, which can have extra functionality, like synchronizing multiple streams, reordering streams, delaying streams, etc. These extra functionalities will be the topic of this Subsection. The reason for using these blocks is design reuse. They can be integrated at any point of the signal processing datapath and typically have weakly-programmable capabilities that give them flexibility to change their function at run-time. Like the DPUs, they are fast, since they are tightly coupled with the distributed memory that populates any modern FPGA.

There are several LMC types. The simplest, *lmc_fifo*, is a FIFO whose length and datawidth are configurable at synthesis time. Another simple LMC, *lmc_join*, is capable of synchronizing n multiple streams, each having its own datawidth. It stores incoming streams in independent token buffers. As soon as all buffers have at least one token, a combined token is outputted. Both these LMCs are not weakly-programable because their parameters cannot change at run-time.

More complex and flexible LMCs have a dual-ported local memory that is written and read using independent Address Generators (AG), an ingress AG for the incoming data (memory write address) and an egress AG for the outgoing data (memory read address). In order to generate those addresses in a flexible and reusable way, an *address stepper* base component was designed and later cascaded in a way that many different address patterns can be generated.

3.3.1 *Address Stepper.* This component is basically a flexible, loadable counter. It has a *start* input to set the counter's starting point, loadable by the *nload* signal; the value that is added in each counter step is controlled by the *delta* input and the counter limit is set by the *end* input. The function of this counter is rather simple and its structure is shown in Figure 5.

Since all inputs are implemented as two's complement signed values, positive and negative numbers can be used. It contains two architectures: one generic register transfer level (RTL) implementation that can be used in every FPGA family, and one Xilinx optimized architecture that uses less resources and is faster because it combines the functionality of a full-bit adder and a multiplexer in a single Xilinx Virtex slice.

3.3.2 *Cascaded Stepper.* It consists of three connected steppers, a register to buffer one incoming parameter and some control logic as shown in Figure 6. The Cascaded Steppers are based on those described in [Hartenstein et al. 1987] and generate address sequences using only six parameters. This set of parameters, although small, provides a large number of patterns, allowing the LMC to rotate, flip, decimate, extract a ROI (region-of-interest), scan in different zig-zag patterns or implement a combination of any of these operations.

The two outer steppers, named base- and limit-stepper, generate the parameters for the addr-stepper. The example pattern shown in Figure 7 will be used to explain the function of this structure. The desired address pattern is a simple walkthrough of a 6x6 picture, where every even value is read (0, 2, 4, 6, ... 34). The base-stepper calculates the address of the first pixel of every line, whereas the limit-stepper generates the address of the last active pixel in each line. Those results are passed on to the addr-stepper, which generates the final output. For the pattern in Figure 7, one set of six parameters is needed. This set includes one complete set of 3 stepper parameters for the base-stepper. The limit-stepper requires one parameter less, because it runs in sync with the base-stepper, so no end parameter is required. The parameter set is completed by an addr_delta parameter for the addr-stepper. This stepper does not need any other parameters, since these are provided by the two outer steppers. This leads to the parameters in Figure 7 (center).

To reorder a stream, a LMC address pattern transformer (*lmc_apt*) consists of one local memory and two Cascaded Steppers (one for the ingress, one for the egress stream address), each coupled with a counter that controls the amount of times the address pattern repeats itself. The local dual-ported memory is divided into m regions ($m = 2^k, k \in \mathbb{N} \wedge k \geq 1$), which allows a parallel operation of both address steppers in different regions. Each Cascaded Stepper accesses these regions sequentially and is guarded from the other stepper by hardware mutexes to exclude memory hazards. Basically, the egress can only access memory region n ($0 \leq n \leq m$) after the ingress finishes writing its pattern to it, and vice-versa. The memory's depth, width and number of regions are configurable by generics, and the memory consistency is kept automatically, making this block very flexible and easy to program. An example of an application of such an LMC is creating a block-based zig-zag stream: the input stream is stored sequentially block by block in the dual-ported local RAM and is read out using three patterns, whose parameters can be seen in Figure 8

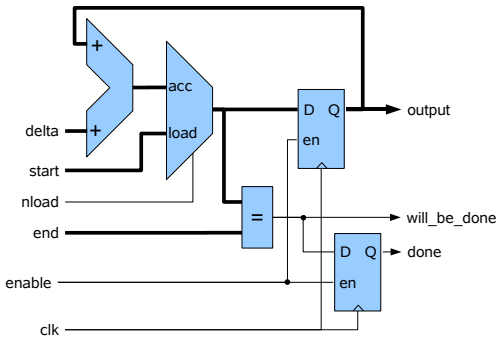


Figure 5: Detailed functional stepper diagram

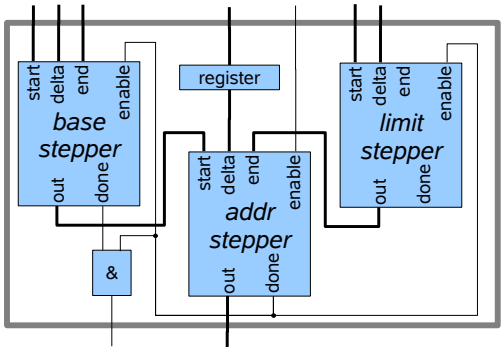


Figure 6: Cascaded Stepper functional diagram

(right). Ingress and egress sequential pictures are shown together with the calculations required for a generic square block size.

Some image processing algorithms require more memory than the one available inside the FPGA, so external memory must be used. For that purpose, another set of LMCs were created. They consist of a Cascaded Stepper to generate addresses for the external memory and a local FIFO. One LMC is used to write to external memory, another is used to read from it.

For applications that require large FIFOs, an LMC exists that uses an external memory address region as storage space. It needs two address generators to access it, two small local FIFOs and some full-empty control logic to create it.

Therefore, external memory interfaces are necessary. The FlexWAFE solution for external memory accesses is presented in the next Subsection.

3.4 Custom DDR-SDRAM Memory Controller

As previously stated, the FlexFilm board utilizes external DDR-SDRAM memory to store image data. Memory is clocked at 125 MHz, allowing peak performance of 8 GBit per second. To achieve real rates close to this theoretical maximum, an access optimizing memory controller CMC was developed. A brief overview of the controller is provided in this Section.

3.4.1 *Quality of Service.* While not currently utilized for the sample applications described in Section 4, each of the four XC2VP50-6 FPGAs contains two embedded Pow-

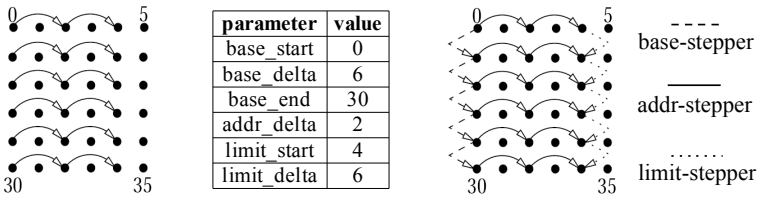


Figure 7: Simple address pattern, horizontal decimation by two, on a 6x6 pixel picture (left); the parameters that generate it (center) and its generation via the 3 steppers of the Cascaded Stepper module (right)

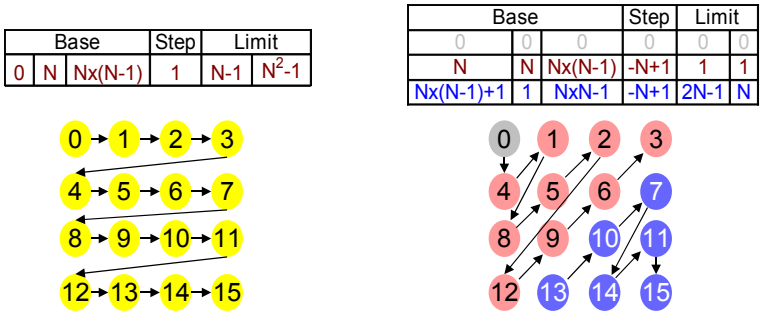


Figure 8: Zig-zag address sequence example for image blocksize of 4. Cascaded Stepper parameters for ingress AG (left) and egress AG (right)

erPC processors available for use for simple computations, control-dominant tasks, and parameter calculation. Due to limited on-chip memory sizes and the potential limited use of the PowerPC processors, as well as technical issues regarding board and pin layout, a shared memory system was used for both CPU code and data. This implies very different memory access patterns, which must be properly served by the SDRAM controller.

For effective operation, CPU cache miss memory accesses generated by nonstreaming, control-dominated applications should be served with a *smallest possible latency at guaranteed minimum throughput and guaranteed maximum latency*, since the processor must stall while waiting on a read access or cache miss. This makes memory access time the critical component for performance of the embedded PowerPC processors.

At the same time, data path memory requests, which have a fixed access sequence and allow deep prefetching and buffering due to increased latency tolerance, should be served with a *guaranteed minimum throughput at guaranteed maximum latency*. A more detailed explanation can be found in [Heithecker et al. 2003; Heithecker and Ernst 2005].

To handle these quality of service (QoS) requirements, two priority levels for memory access requests have been implemented in the SDRAM controller. High priority requests (*smallest possible latency*) are always executed before standard priority requests. This is implemented via distinct access paths for high and standard priority requests and a modified scheduler, which always executes high priority requests first.

With any priority-based design, starvation at the lower levels is often an issue. To avoid possible starvation of standard priority requests (*guaranteed minimum throughput at guaranteed maximum latency*), *traffic shaping* is used to reduce the maximum throughput of high priority requests. The traffic shaping unit can be configured to pass n requests within T clock cycles (known as "sliding window" in networking applications) to allow bursty CPU memory accesses when necessary.

3.4.2 *Additional Requirements.* For real-time digital film processing, the FlexWAFE architecture requires data rates that cannot be delivered by simple memory controller designs offering first-come first-served access. Actual data rates for such memory controllers can drop as low as 40% of theoretical maximum values. To increase memory throughput utilization, memory access optimizations must be performed to reorder memory requests. This allows the memory controller to take advantage of the structure of SDRAM memory, which has a buffered parallel architecture that is organized into independent memory

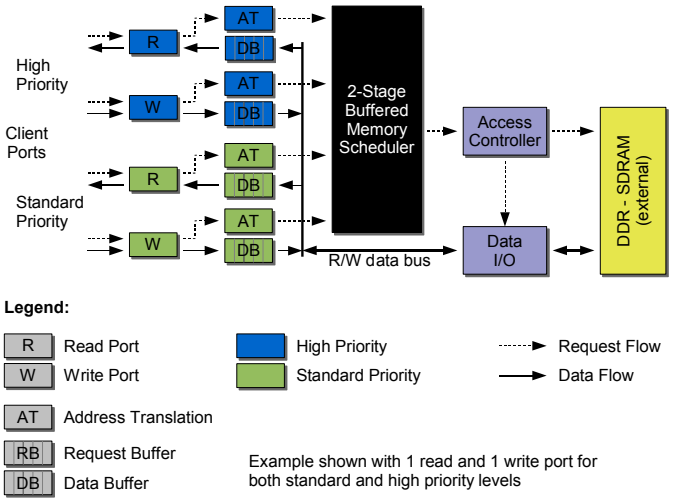


Figure 9: SDRAM controller architecture

banks.

Bank Interleaving exploits this structure by reordering memory requests to ensure that consecutive accesses occur to inactive banks. In this manner, a second bank can be accessed during idle times, effectively hiding precharge latencies and significantly increasing data rates.

Request Bundling minimizes the effects of idle cycles required during bus direction switches. These stalls (1 for a read-write change, 1-2 for a write-read change, depending on the SDRAM module) can decrease overall throughput by up to 27% [Heithecker et al. 2003]. By bundling like requests together into continuous blocks, these stalls can be avoided.

A more complete summary of such memory access optimizations can be found in [Rixner et al. 2000].

3.4.3 *General Architecture.* An architectural overview of the CMC (configured with two high and two standard priority ports, one read and one write port each and four SDRAM banks) is shown in Figure 9.

A brief overview of the controller functionality is as follows. After arriving at a read or write port of the CMC, memory access requests first reach the *address translator*, where the logical address is translated into the physical rank/bank/row/column quadruple required by the SDRAM, where a "rank" designates a single or group of SDRAM modules controlled by a unique chip select signal.

Concurrently, at the *data buffers*, the write request data is stored until the request has been scheduled; for read requests a buffer slot for the data read from the SDRAM is reserved.

The requests then enter the core part of the SDRAM controller, the two-stage buffered memory access scheduler (see Figure 10). After one request is selected, it is executed by the *access controller* and the data transfer to/from the corresponding data buffer is initiated by the *data I/O* module. Finally, external data transport and signal synchronization for DDR transfers is managed by the DDR Interface.

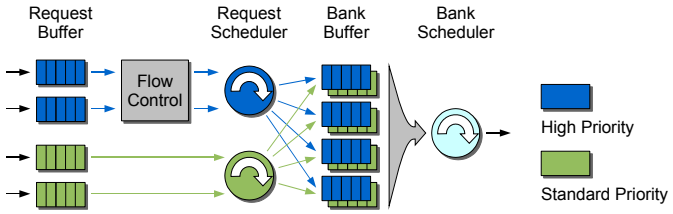


Figure 10: Two-stage buffered scheduler

The core of the controller, the two-stage buffered memory access scheduler, performs access optimizations and issues requests to SDRAM. Figure 10 shows the scheduling stages described in more detail below.

3.4.4 *Request Buffer, Request Scheduler.* The single-slot request buffers are used to decouple the clients from the following scheduling stages. The first scheduler stage, the request scheduler, selects requests from these buffers, one request per two clock cycles, and forwards them to the bank buffer FIFOs. By applying a round-robin arbitration policy, a minimum access service level is guaranteed. As stated above, high priority requests are serviced before standard priority requests when priority levels are enabled.

3.4.5 *Bank Buffer, Bank Scheduler.* The bank buffer FIFOs, one for each bank, store the requests according to the addressed bank. The second scheduler stage, the bank scheduler, selects requests from these bank buffer FIFOs and forwards them to the access controller for execution. In order to increase throughput utilization, the bank scheduler performs *bank interleaving* to hide bank access latencies and *request bundling* to minimize stalls caused by read-write switches.

The CMC remains flexible in terms of configurability, and has been designed to be compatible with other high-performance reconfigurable FPGA and ASIC platforms. Only a select group of the core SDRAM controller components have been described here. For a more thorough description of the CMC used in the FlexFilm project, see [Heithecker et al. 2003]. The CMC was further expanded to include a state-of-the-art NoC interface for use in the MORPHEUS Project [Thoma et al. 2007], providing a high-speed external memory link to the MORPHEUS reconfigurable computing platform. For details on these latest changes, consult [Whitty and Ernst 2008].

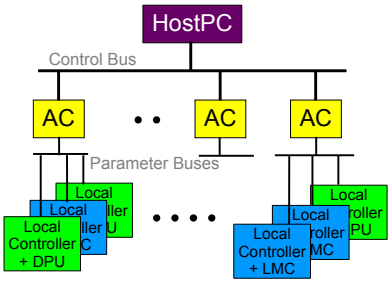
3.5 Inter-chip and Inter-board Communication

Because the architecture is designed for stream processing, a reliable transport for multiple image streams between the FPGAs needed to be designed. The realized stream protocol is explained in detail in [Heithecker et al. 2007]; for completeness, a short presentation will be given here. A TDM based protocol that supports the aforementioned backpressuring technique was implemented. On the physical layer, 16 parallel connections are driven at QDR (quad data rate, 4x125 MHz). Because of the well-known inter-chip data rates in image stream processing, a fixed TDM schedule is computed, guaranteeing an ideal usage of the available data rate. The schedule is calculated in VHDL at synthesis time using the number of channels and respective data rates as input generics. For inter-board communication the same techniques are applied. Clock skew is compensated by capture

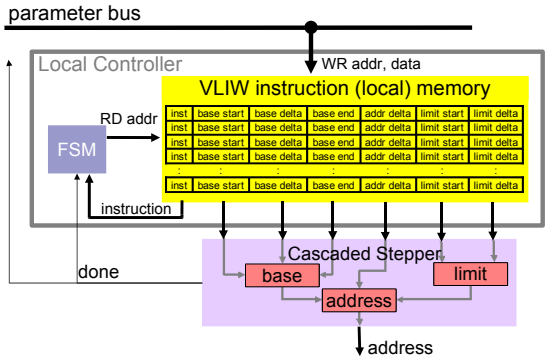
flipflops and asynchronous FIFOs. The back pressuring protocol is also supported between two boards.

3.6 Control and Programmability

As seen in the previous Subsections, the FlexWAFE architecture consists of stream oriented datapaths where some of the blocks have parameter inputs that can be changed at run-time. Weak programmability is achieved by a Very Large Instruction Word (VLIW) controller, local to each block, running its program from a local memory. Since each block can be equipped with its own Local Controller/memory pair, the overall system's control is truly distributed and parallel. The program sequences run by these Local Controllers are written from ACs that functions like a VLIW unit. There can be more than one of these per FPGA and their programs are in turn written by the host computer. Control is therefore hierarchical, as shown in the example in Figure 11a. Essentially, in the FlexWAFE approach, weak-programmability is used to share the hardware for different functions, instead of complex FPGA dynamic (partial)-reconfiguration. The crucial advantage is a much faster context switch that takes a single cycle, with little overhead cost, and support for a very high area utilization. On the other hand, flexibility is limited to the predefined functions. This loss in flexibility, however, can be compensated by a suitable collection of functions that will be presented later. Each of these controllers and the buses that interconnect them are explained in the following paragraphs.



(a) Control hierarchy



(b) Example of a Local Controller, attached to a Cascaded Stepper

Figure 11:

3.6.1 *Local Controller.* A Local Controller is a very simple Finite State Machine (FSM) that steps through the VLIW instructions previously stored in a local memory and has only two instructions: *load* and *reset*. A *load* instruction outputs an entire set of parameters whose values are part of the instruction itself. Afterwards, it waits for a *done* signal from the block it controls and steps to the next instruction. If that instruction is a *load*, the process repeats itself; if it is a *reset*, then it steps back to the very first instruction in memory.

Each of the parameters stored in the VLIW words are read in parallel, but are written independently from each other via a parameter bus depicted in Figure 11b. This allows the Local Controller to output one parameter set from one VLIW instruction while other(s) are being written. This kind of operation is known as *double-buffering*, but in our case it is actually *n-buffering*, where n is the depth of the VLIW instruction memory. This is necessary, for example, for the zig-zag pattern shown in Figure 8, where it avoids address generation interruptions by providing the next parameter set instantly.

The number of parameters in the VLIW words and the width of each parameter is configurable via VHDL generics. On Xilinx devices, the local memory gets translated to look-up-table (LUT) based dual-ported distributed memory of depth 16 that exists in every slice and will therefore be placed together with the datapath unit that it controls. It can feed the datapath with one new set of parameters per clock cycle. Figure 11b shows an example of tight integration of a Local Controller and a *Cascaded Stepper*.

In this manner, the DPUs and LMCs, which possess such Local Controllers, behave as weakly-programmable components, much like the type of weakly-programmable coprocessors used in MpSoCs such as Viper [Dutta et al. 2001] that separate time-critical local control inside the components from non time-critical global control.

3.6.2 *AC- Algorithm Controller* . This component is also a VLIW processor and also contains dual-ported local memory. A *send* instruction writes one parameter value via the parameter bus to one particular Local Controller's VLIW word, and have an optional *wait* mask that identifies one or more Local Controllers' *done* signals, for which the AC should wait before stepping into the next instruction. The *goto* instruction jumps to a particular memory location. This simple instruction set allows the implementation of control loops that depend on the distributed system's state.

The contents of the program memory are described in XML. The parameter bus, AC's wait mask and AC's VLIW word width are calculated and synthesized automatically. On Xilinx devices, the AC's program memory uses one or more BRAM blocks automatically.

There can be more than one AC per FPGA. Following the compositional FlexWAFE approach, one AC should control one group of inter-related DPUs and LMCs. Global synchronization between groups of non-related DPUs and LMCs is achieved via the inter-block signaling protocol described in Section 3.1.

The dual-ported program memory of the AC is reconfigurable at run-time using the Control Bus that is explained in the next Subsection.

3.6.3 *Control Bus.* This bus connects all ACs from all FPGAs to a host PC. In the FlexFilm board implementation, the datawidth is 16 bit, the address space is 14 bit wide and the I/O FPGA (see Figure 1) provides a bridge between the Control Bus and the host PCIe bus. In this implementation, single word read/write accesses take around 4 microseconds, resulting in an average data rate of 500 KByte/second.

3.7 Design Flow and Hierarchy

The design flow was first presented in [do Carmo Lucas et al. 2007] and will be further explained here. The first step in the design flow is to describe the application using the available DPUs and LMCs. If one of the required processing blocks does not yet exist in the DPU library, it must be coded in VHDL and added to the library for future use. The existing LMCs have enough flexibility to cover the needs of most algorithms and, therefore, only need to be parameterized at synthesis-time and (re-)configured by weak-programmability at run-time. In the second step, all projected blocks are instantiated and interconnected, adding the required InterFPGA mux and demux communication blocks, CMC (s) and AC(s). The third step is to describe parameters that require run-time flexibility in a very simple XML file format. Next, the FlexWAFE tool-chain analyzes this XML file and generates the appropriate distributed parameter memories inside each Local Controller, the microcode for the AC(s), and the synthesis-time parameters for all the blocks. Some VHDL files are also automatically parsed and a suitable `.ucf` (User Constraints File) is generated. At this point, the system can be synthesized and downloaded to the FPGA(s), and the tool-chain will then program the run-time configurable system parameters using once more the information from previous steps. The variables defined in step three can be changed if necessary without requiring a new synthesis, as long as they accept the limits defined in other stages. The device driver allows faster than real-time image transfer to and from the FlexFilm board using direct memory access (DMA), accessing the Control Bus and thus partially or completely reprogramming all the ACs and reprogramming each FPGA individually in 300 ms.

The next section will present various case studies that used this design flow.

4. CASE STUDIES

In order to test the performance, reconfigurability, and usability of the presented architecture, several image processing applications with mixed requirements and complexity have been implemented using FlexWAFE. In the following Subsections, four applications are presented and their implementation results and development processes are analyzed.

4.1 Film Grain Noise Reduction Application

To demonstrate the processing capabilities of the architecture, a complex algorithm for reducing the film grain noise produced by film scanners was implemented. The algorithm is depicted in Figure 12 and consists of a bidirectional Motion Estimation (ME) followed by Motion Compensation (MC) and a 2.5 dimension Discrete Wavelet Transformation (DWT). Its theory was presented in detail in [Eichner et al. 2005]. The application was developed and implemented in collaboration with Thomson Media Solutions and TU-Ilmenau [do Carmo Lucas et al. 2006].

The algorithm starts by calculating motion vectors, comparing an image with its predecessor and successor images. The motion is compensated for filtering and a pseudo-temporal Haar filter is performed on the resulting compensated and current image. The two results are transformed in 5/3 wavelet space, where the film grain noise is filtered by a shrinkage function with user selectable parameters. Afterwards, the image is retransformed into normal space by an inverse wavelet transformation and a pseudo-temporal inverse Haar filter. The DWT operates two-dimensionally in the horizontal and vertical directions; information from the time domain is introduced by the Haar wavelet. This sums

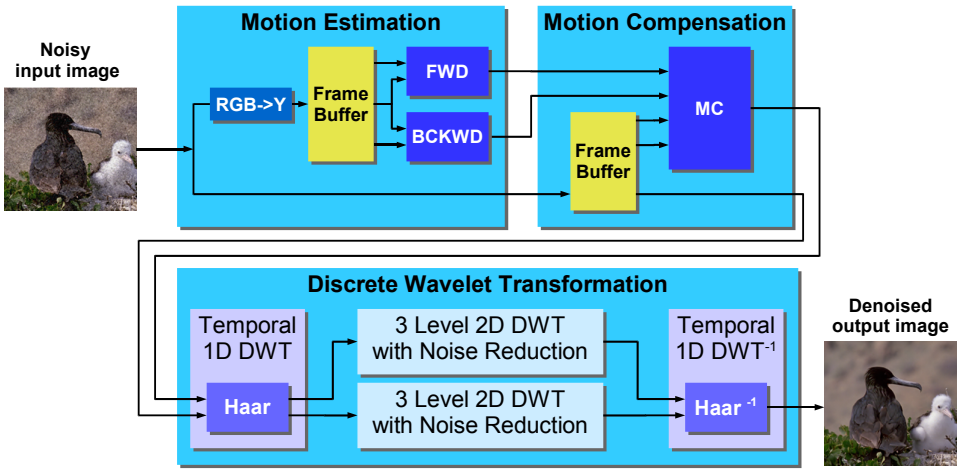


Figure 12: Advanced noise reduction algorithm

up to a 2.5 dimensional transformation, as the temporal transformation is not performed by a complete 5/3 wavelet and uses one bidirectional compensated image instead of five straight consecutive images. Several application parameters (e.g. number of DWT levels) are configurable at design time. Other parameters (e.g. coefficients for Noise Reduction (NR)) can be changed during runtime using weakly-programmable components. The algorithm uses more than 80 % of the available chip resources of each processing FPGA and is therefore the most demanding case study. Compared to a reference implementation on an Intel Pentium 4, the FPGA implementation is 2000 times faster. The complete noise reduction algorithm implementation is presented in detail in [Heithecker et al. 2007].

4.2 Lossless Data Compression

Data compression is one of the key applications in digital image processing and can be partitioned into two categories: lossless and lossy compression. One application domain for lossless compression is the space research domain, where compression artifacts produced by lossy compression techniques could have drastic consequences, such as leading to false discoveries of new celestial bodies. Therefore, the FlexWAFE architecture was used for prototyping a compression algorithm for a space camera that implements data compression following the CCSDS 121 [CCSDS 1997] recommendations. The compression subtracts two neighboring pixels and encodes the difference using Rice [Rice 1979] coding, which replaces parts of the data word with its unary representation depending on the data content. This technique is especially efficient for similar pixels, as their small pixel difference values result in short unary representations. The application was designed as a SystemC model and major parts are already implemented using the FlexWAFE library (implementation work ongoing). Various parameters can be changed at design time via VHDL generics.

4.3 Image Scaling

As a third application, a simple image down scaler was developed. It is based on a C implementation taken from the GIMP project and calculates a scaled pixel as the average of

the color components of the corresponding pixels from the original image. Originally, the application was implemented during a different project for a Xilinx Virtex FPGA, but was later adapted to the FlexWAFE architecture in a second step. The scaler DPU uses weak-programmability to change the scaling factors, thereby adjusting the output image size. A possible use case for this application is the scaling of cinema resolutions like 2K (2048x1568 pixels) to consumer formats like 1080p (1920x1080 pixels) or 720p (1280x720 pixels).

4.4 Block-based Histogram

Histogram is an important assistance application for image processing algorithms, as it extracts valuable information from an image that is useful for later processing steps. For example, filter coefficients can be dynamically adapted depending on the dye distribution of an image producing better processing results. Therefore, a block-based histogram for the pixel's luminance component was developed in a recent project.

The key difficulty when designing the algorithm was the demand of local memory. Due to the rowwise organization of the image data, all possible histogram values per block have to be buffered until all lines of a block are traversed. Assuming a block size of 32 pixels and 10 bits per color component, 1024 possible histogram values per block must be stored locally in the worst case. When processing 4K images (4096x3112 pixels, 128 blocks/row), 131072 values need be to kept in local memory, which leads to a storage requirement of 1.3 Mbit. This is about one third of the total amount of memory of a FlexFilm FPGA and is unacceptable for an assistance application.

Consequently, the histogram implementation is equipped with a fixed amount of local memory at design time and trades off the amount of possible histogram bins against the block size and resulting blocks per row before compilation. With this technique, a predictable memory footprint and an ideal usage of the allocated memory are guaranteed. Additionally, these parameters can be modified using weak-programmability, resulting in a highly flexible application.

4.5 Results

When evaluating the results, several aspects of the implemented applications must be considered. A key aspect is the performance of the implementations as it shows the processing capabilities of the FlexWAFE architecture. Configurability of the applications is identified as a second important aspect that measures the flexibility of the weakly-programmable architecture. This is a more or less abstract metric, as flexibility can hardly be expressed in numbers. Last but not least, the development effort for the applications is evaluated, measuring the programmability and usability of the architecture. Several other aspects exist, such as power efficiency, but they are omitted as they had a minor relevance during the implementation of the applications (e.g. power efficiency was only relevant for the data compression application).

4.5.1 *Performance.* The performance analysis will focus on the noise reduction algorithm, as it is the most complex and demanding application and allows the most detailed conclusions. The noise reduction algorithm performs 2000 operations on each pixel and consumes, as already mentioned, more than 80% of the available chip area. When executed on the FlexFilm board, it achieves a frame rate of 26.4 frames per second processing 2K images (2048x1568 pixels), which results in a processing performance of 170 GOPS.

	Ops/pixel	fps (Res.: 2048x1568 pixels)	GOPS	Area used
Noise reduction	2000	26.4 fps	170	80%
Lossless compression	54	37 fps	6.4	6%
Scaler	24	37 fps	2.9	2%
Histogram	22	37 fps	2.6	2%

Table I: Case study performance results

Thus, the real-time requirements of 24 fps for 2K images are met and even exceeded.

All other algorithms are of minor processing complexity and use only a small percentage of the chip area. Each is able to process one pixel per clock cycle. Their framerate is therefore limited only by the incoming data rate, which is 37 fps for 2K images. The numbers for the lossless compression are currently preliminary, as the implementation is still ongoing. A summary of the performance results can be found in Table I.

4.5.2 *Configurability.* For reconfigurable architectures, configurability often describes the method and the speed of chip reconfigurations. However, for the presented weakly-programmable approach, configurability designates the configuration possibilities of the implemented applications using the presented control hierarchy (see Section 3.6). Again, the focus will be on the noise reduction application as it offers the best possibilities to exploit configurability.

The algorithm itself can be customized using 54 run-time reconfigurable parameters. If the image size changes, the address generators must be reprogrammed, resulting in a complete exchange of 465 parameters via the control bus.

Due to the lower complexity of the other applications, only few parameters are exchangeable in the corresponding algorithms. Nevertheless, they all are run-time reconfigurable and allow flexible data processing.

4.5.3 *Development effort.* In this section, the effort required to develop the applications is evaluated. The first application developed was the noise reduction application. Due to the complexity of the algorithm, the dense packing on the FPGAs, and the parallel design and implementation of the FlexWAFE architecture, the realization of the complete algorithm required approximately four years of development time. It must be emphasized that these four years include the development of the architecture, the setup of the complete tool chain and the integration of a test framework. The separated development effort for the application can be estimated as 6.5 months (DWT: 2 months, ME: 3 months, MC: 1.5 months).

After the FlexWAFE architecture became available, the development time decreased significantly. The adaption of the scaler application required only two months and was carried out during an internship by a student with no previous knowledge of the project. Lossless data compression and the histogram implementation both took approximately three months and were realized by skilled students within the scope of student research projects. In each case, the FlexWAFE architecture helped to decrease development time, thereby increasing the developer's productivity. All projects except for the noise reduction were realized by students without in-depth knowledge of the ideas presented in this paper, which illustrates the excellent usability and relatively moderate learning curve of the FlexWAFE architecture.

5. CONCLUSION

The FlexWAFE architecture and design-flow were presented in detail. FlexWAFE consists of a network of configurable weakly-programmable processing elements that are controlled by one or more central Algorithm Controller(s). The processing elements contain Local Memory Controllers, allowing complex local access patterns to utilize the local RAM resources of an FPGA. Configurable auxiliary circuit functions for external memory control and chip-to-chip communication were also presented. Weak-programmability in combination with configuration is used as an alternative to dynamic partial reconfiguration, and achieves extremely high performance at high area utilization. The strength of the approach is finally demonstrated through several applications, including a very demanding next generation noise reducer for electronic film applications.

REFERENCES

AHN, J. H., DALLY, W. J., KHAILANY, B., KAPASI, U. J., AND DAS, A. 2004. Evaluating the Imagine Stream Architecture. *SIGARCH Comput. Archit. News 32,* 2, 14. 1.1

ASPEX LTD. 2008. ASProCore Overview website. http://www.aspex-semi.com. 1.1

BLYTHE, D. 2008. Rise of the Graphics Processor. *Proceedings of the IEEE 96,* 5 (May), 761–778. 1.1

CCSDS. 1997. *Lossless Data Compression, Blue Book*. Consultation Commitee for Space Data Systems. 4.2

CLOUTIER, J., PIGEON, S., BOYER, F. R., COSATTO, E., AND SIMARD, P. Y. 1996. Vip: An fpga-based processor for image processing and neural networks. *Microelectronics for Neural Networks and Fuzzy Systems, International Conference on/Microelectronics for Neural, Fuzzy, and Bio-Inspired Systems, International Conference on 0*, 330. 1.1

CROOKES, D., BENKRID, K., BOURIDANE, A., ALOTAIBI, K., AND BENKRID, A. 2000. Design and implementation of a high level programming environment for fpga-based image processing. *Vision, Image and Signal Processing, IEE Proceedings - 147,* 4 (Aug), 377–384. 1.1

DA VINCI SYSTEMS. 2008. Da Vinci Systems website. http://www.geniusofdavinci.com. 1.1

DIGITAL VISION AB. 2008. Digital vision DVNR website. http://www.digitalvision.se. 1.1

DO CARMO LUCAS, A. AND ERNST, R. 2005. An Image Processor for Digital Film. In *IEEE ASAP*. 3

DO CARMO LUCAS, A., HEITHECKER, S., AND ERNST, R. 2007. FlexWAFE - A high-end real-time stream processing library for FPGAs. In *DAC '07: Proceedings of the 44th annual conference on Design automation*. ACM Press, New York, NY, USA, 916–921. 3.7

DO CARMO LUCAS, A., HEITHECKER, S., RÜFFER, P., ERNST, R., RÜCKERT, H., WISCHERMANN, G., GEBEL, K., FACH, R., HUNTHER, W., EICHNER, S., AND SCHELLER, G. 2006. A reconfigurable HW/SW platform for computation intensive high-resolution real-time digital film applications. In *IEEE DATE*. 194–199. 1, 2, 4.1

DUTTA, S., JENSEN, R., AND RIECKMANN, A. 2001. Viper: A multiprocessor SoC for advanced set-top box and digital TV systems. In *IEEE Design and Test of Computers, Sip*. 21–31. 3.6.1

EICHNER, S., SCHELLER, G., WESSELY, U., RÜCKERT, H., AND HEDTKE, R. 2005. Motion compensated spatial-temporal reduction of film grain noise in the wavelet domain. In *SMPTE Technical Conference, New York*. 4.1

GUO, Z., MITRA, A., AND NAJJAR, W. 2006. Automation of IP Core Interface Generation for Reconfigurable Computing. In *Proc. International Conference on Field Programmable Logic and Applications FPL '06*. 1–6. 1.1

HARTENSTEIN, R., HIRSCHBIEL, A., AND WEBER, M. 1987. MOM - Map Oriented Machine. In *Proceedings of the International Workshop on Hardware Accelerators*. 3.3.2

HEITHECKER, S., DO CARMO LUCAS, A., AND ERNST, R. 2003. A Mixed QoS SDRAM Controller for FPGA-Based High-End Image Processing. In *Workshop on Signal Processing Systems Design and Implementation*. IEEE, TP.11. 3.4.1, 3.4.2, 3.4.5

HEITHECKER, S., DO CARMO LUCAS, A., AND ERNST, R. 2007. A High-End Real-Time Digital Film Processing Reconfigurable Platform. *EURASIP Journal on Embedded Systems, Special Issue on Dynamically Reconfigurable Architectures 2007*, Article ID 85318, 15 Pages. 1.1, 3.5, 4.1

HEITHECKER, S. AND ERNST, R. 2005. Traffic Shaping for an FPGA-Based SDRAM Controller with Complex QoS Requirements. In *Design Automation Conference (DAC)*. ACM, 575 – 578. 3.4.1

HOOVER, G. AND BREWER, F. 2008. Synthesizing Synchronous Elastic Flow Networks. In *Proc. Design, Automation and Test in Europe DATE '08*. 306–311. 1.1

HUNT ENGINEERING LTD. Homepage. http://www.hunteng.co.uk. 1.1

KAHLE, J. A., DAY, M. N., HOFSTEE, H. P., JOHNS, C. R., MAEURER, T. R., AND SHIPPY, D. 2005. Introduction to the Cell multiprocessor. In *IBM Journal of Research and Development*. 1.1

KARP, R. AND MILLER, R. 1966. Properties of a Model for Parallel Computations: Determinancy, Termination, Queueing. *SIAM Journal of Applied Math 40,* 6 (November). 3.1

KUMAR, A., HANSSON, A., HUISKEN, J., AND CORPORAAL, H. 2007. An FPGA Design Flow for Reconfigurable Network-Based Multi-Processor Systems on Chip. In *Proc. Design, Automation & Test in Europe Conference & Exhibition DATE '07*. 1–6. 1.1

LEE, E. A. AND MESSERSCHMITT, D. G. 1987. Synchronous Data Flow. In *Proceedings of the IEEE*. Vol. 75. 1235 – 1245. 3.1

LEE, K.-B., LIN, T.-C., AND JEN, C.-W. 2005. An Efficient Quality-Aware Memory Controller for Multimedia Platfrom SoC. *IEEE Transactions on Circuits and Systems for Video Technology 15,* 5 (May), 620–633. 1.1

LOO, S., WELLS, B., FREIJE, N., AND KULICK, J. 2002. Handel-C for rapid prototyping of VLSI coprocessors for real time systems. In *Proc. Thirty-Fourth Southeastern Symposium on System Theory*. 6–10. 1.1

NAJJAR, W., BOHM, W., DRAPER, B., HAMMES, J., RINKER, R., BEVERIDGE, J., CHAWATHE, M., AND ROSS, C. 2003. High-level language abstraction for reconfigurable computing. *Computer 36,* 8 (Aug.), 63–69. 1.1

NALLATECH LTD. 2007. DIMEtalk 3 Product Brief. 1.1

QUANTEL LTD. 2008. Quantel Pablo website. http://www.quantel.com. 1.1

RICE, R. F. 1979. Some Practical Universal Noiseless Coding Techniques. *JPL Publication 91-3, Part i11, Module PSI14, K+*. 4.2

RIXNER, S., DALLY, W. J., AND KAPASI, U. J. 2000. Memory Access Scheduling. In *International Symposium on Computer Architecture*. 128–138. 3.4.2

SHUKLA, S., BERGMANN, N., AND BECKER, J. 2007. QUKU: A FPGA Based Flexible Coarse Grain Architecture Design Paradigm using Process Networks. In *Proc. IEEE International Parallel and Distributed Processing Symposium IPDPS 2007*. 1–7. 1.1

SHUKLA, S., BERGMANN, N. W., AND BECKER, J. 2005. APEX - A Coarse-Grained Reconfigurable Overlay for FPGAs. In *Proceedings of the IFIP VLSI SoC*. 1.1

SONICS INC. 2005. Sonics MemMax 2.0 Multi-threaded DRAM Access Scheduler. Data sheet, Sonics Inc. 1.1

STREAM PROCESSORS INC. 2008. Storm-1 SP16HP-G220 Product Brief. http://www.streamprocessors.com. 1.1

THE MATHWORKS. 2008. Simulink - Simulation and Model-Based Design Homepage. http://www.mathworks.com/products/simulink/. 1.1

THOMA, F., KÜHNLE, M., BONNOT, P., PANAINTE, E. M., BERTELS, K., GOLLER, S., SCHNEIDER, A., GUYETANT, S., SCHÜLER, E., MÜLLER-GLASER, K. D., AND BECKER, J. 2007. MORPHEUS: Heterogeneous Reconfigurable Computing. In *Proceedings of 17th International Conference on Field Programmable Logic and Applications (FPL07)*. 3.4.5

THOMSON GRASSVALLEY. 2008. Scream 4K/2K/HD noise reducer website. http://www.thomsongrassvalley.com. 1.1

WEBER, W.-D. 2001. Efficient Shared DRAM Subsystems for SOCs. In *Microprocessor Forum*. 1.1

WHITTY, S. AND ERNST, R. 2008. A Bandwidth Optimized SDRAM Controller for the MORPHEUS Reconfigurable Architecture. In *Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 3.4.5

XILINX INC. 2008. Xilinx Virtex 5 Family Overview. http://www.xilinx.com. 1.1, 1.2

Received June 2008; revised September 2008; accepted November 2008