Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2007, Article ID 85318, 15 pages
doi:10.1155/2007/85318

Research Article

A High-End Real-Time Digital Film Processing Reconfigurable Platform

Sven Heithecker, Amilcar do Carmo Lucas, and Rolf Ernst

Institute of Computer and Communication Network Engineering, Technical University of Braunschweig, 38106 Braunschweig, Germany

Received 15 May 2006; Revised 21 December 2006; Accepted 22 December 2006

Recommended by Juergen Teich

Digital film processing is characterized by a resolution of at least 2 K (2048 × 1536 pixels per frame at 30 bit/pixel and 24 pictures/s, data rate of 2.2 Gbit/s); higher resolutions of 4 K (8.8 Gbit/s) and even 8 K (35.2 Gbit/s) are on their way. Real-time processing at this data rate is beyond the scope of today's standard and DSP processors, and ASICs are not economically viable due to the small market volume. Therefore, an FPGA-based approach was followed in the FlexFilm project. Different applications are supported on a single hardware platform by using different FPGA configurations. The multiboard, multi-FPGA hardware/software architecture, is based on Xilinx Virtex-II Pro FPGAs which contain the reconfigurable image stream processing data path, large SDRAM memories for multiple frame storage, and a PCI-Express communication backbone network. The FPGA-embedded CPU is used for control and less computation intensive tasks. This paper will focus on three key aspects: (a) the used design methodology which combines macro component configuration and macrolevel floorplaning with weak programmability using distributed microcoding, (b) the global communication framework with communication scheduling, and (c) the configurable multistream scheduling SDRAM controller with QoS support by access prioritization and traffic shaping. As an example, a complex noise reduction algorithm including a 2.5-dimension discrete wavelet transformation (DWT) and a full 16 × 16 motion estimation (ME) at 24 fps, requiring a total of 203 Gops/s net computing performance and a total of 28 Gbit/s DDR-SDRAM frame memory bandwidth, will be shown.

1. INTRODUCTION

Digital film postprocessing (also called *electronic film postprocessing*) requires processing at resolutions of 2 K × 2 K (2048 × 2048 pixels per anamorphic frame at 30 bit/pixel and 24 pictures/s resulting in an image size of 15 Mibytes and a data rate of 360 Mbytes per second) and beyond (4 × 4 K and even 8 K × 8 K up to 48 bit/pixel). Systems able to meet these demands (see [1, 2]) are used in motion picture studios and advertisement industries.

In recent years, the request for real-time or close to real-time processing to receive immediate feedback in interactive film processing has increased. The algorithms used are highly computationally demanding, far beyond current DSP or processor performance; typical state-of-the-art products in this low-volume high-price market use FPGA-based hardware systems.

Currently, these systems are often specially designed for single algorithms with fixed dedicated FPGA configurations. However, due to the ever-growing computation demands and rising algorithm complexities for upcoming products, this traditional development approach does not hold for several reasons. First, the required large FPGAs make it necessary to apply ASIC development techniques like IP reuse and floorplanning. Second, multichip and multiboard systems require a sophisticated communication infrastructure and communication scheduling to guarantee reliable real-time operation. 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. Finally, the increasing needs concerning product customization and time-to-market issues require simplifying and shortening of product development cycles.

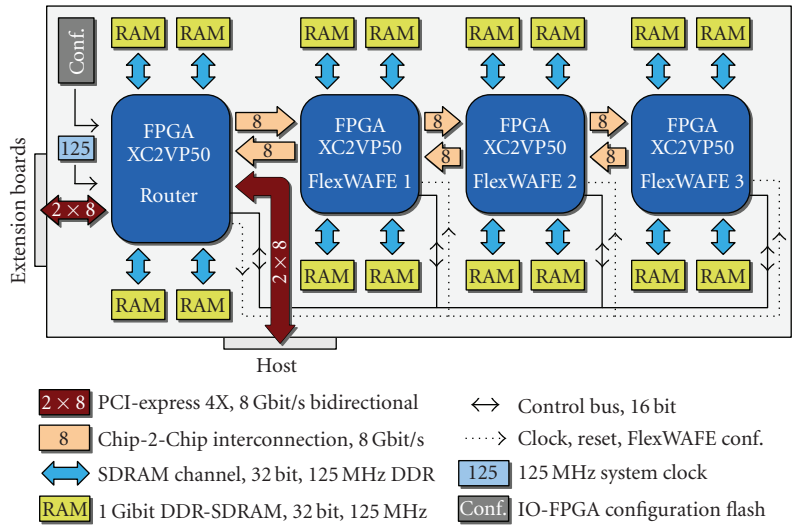


Figure 1: FlexFilm board (block diagram).

This paper presents an answer to these challenges in the form of the FlexFilm [3] hardware platform in Section 2.1 and its software counterpart FlexWAFE [4] (Flexible Weakly-Programmable Advanced Film Engine) in Section 2.2. Section 2.3.1 will discuss the global communication architecture with a detailed view on the inter-FPGA communication framework. Section 2.4 will explain the memory controller architecture.

An example of a 2.5-dimension noise-reduction application using bidirectional motion estimation/compensation and wavelet transformation is presented in Section 3. Section 4 will show some example results about the quality of service features of the memory controller. Finally, Section 5 concludes this paper.

This design won a *Design Record Award* at the DATE 2006 conference [5].

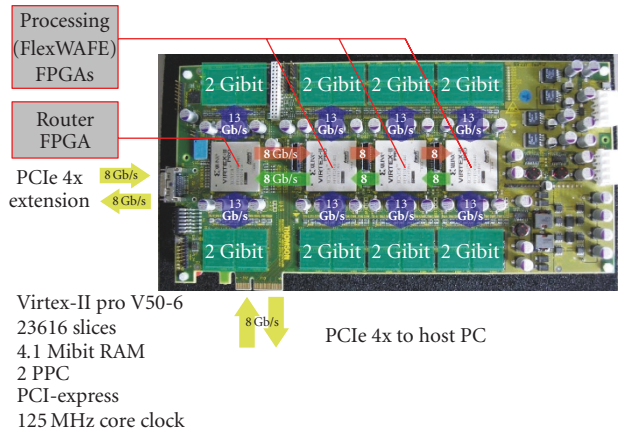


Figure 2: FlexFilm board.

1.1. 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* [6]). With this massive amount of resources, it is possible to build circuits that compete with ASICs-regarding performance, but have the advantage of being configurable, and thus reusable.

PCI-Express [7] (PCIe), mainly developed by Intel and approved as a PCI-SIG [8] standard in 2002, is the successor of the PCI bus communication architecture. Rather than a shared bus, it is a network framework consisting of a series of bidirectional point-to-point channels connected through switches. Each channel can operate at the same time without negatively affecting other channels. Depending on the actual implementation, each channel can operate at speeds of 2 (X1-speed), 4, 8, 16, or 32 (X16) Gbit/s (full duplex, both directions each). Furthermore, PCI-Express features a sophisticated quality of service management to support a variety of end-to-end transmission requirements, such as minimum guaranteed throughput or maximum latency.

Notation

In order to distinguish between a base of 2^{10} and 10^3 the IEC-60027-2 [9] norm will be used: Gbit, Mbit, Kbit for a base of 10^3; Gibit, Mibit, Kibit for a base of 2^{10}.

2. FLEXFILM ARCHITECTURE

2.1. System architecture

In an industry-university collaboration, a multiboard, extendable FPGA-based system has been designed. Each FlexFilm board (Figures 1 and 2) features 3 Xilinx XC2PV50-6 FPGAs, which provide the massive processing power required to implement the image processing algorithms.

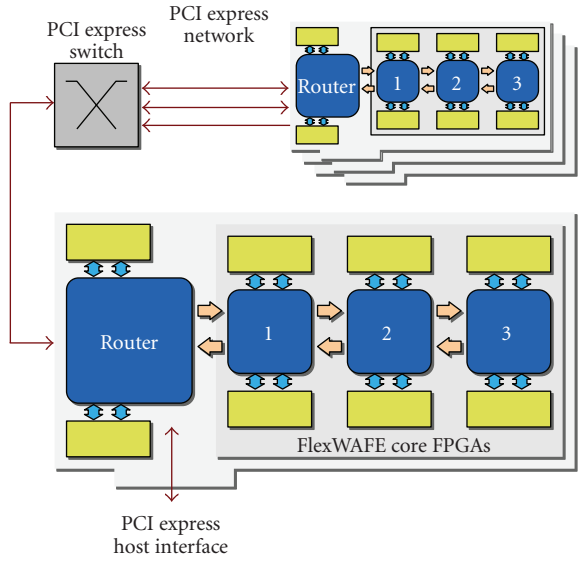


Figure 3: Global system architecture.

Another FPGA (also a Xilinx XC2PV50-6) acts as a PCI-Express router with two PCI-Express X4 links, enabling 8 Gbit/s net bidirectional communication with the host PC and with other boards (Figure 3).

The board-internal communication between FPGAs uses multiple 8 Gbit/s FPGA-to-FPGA links, implemented as 16 differential wire pairs operating at 250 MHz DDR (500 Mbit/s per pin), which results in a data rate of one 64-bit word per core clock cycle (125 MHz) or 8 Gbit/s. Four additional sideband control signals are available for scheduler synchronization and back pressuring.

As explained in the introduction, digital film applications require huge amounts of memory. However, the used Virtex-II Pro FPGA contains only 4.1 Mibit of dedicated memory resources (232 RAM blocks of 18 Kibit each). Even the largest available Xilinx FPGA provides only about 10 Mibit of embedded memory, which is not enough for holding even a single image of about 120 Mibit (2 K resolution). For this reason, each FPGA is equipped with 4 Gibit of external DDR-SDRAM, organized as four independent 32-bit wide channels. Two channels can be combined into one 64-bit channel if desired. The RAM is clocked with the FPGA core clock of 125 MHz, which results at 80% bandwidth utilization in a sustained effective performance of 6.4 Gbit/s per channel (accumulated 25.6 Gbit/s per FPGA, 102.4 Gbit/s per board).

The FlexWAFE FPGAs on the FlexFilm board can be reprogrammed on-the-fly 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 letting 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. After that, either multiple boards are used to carry out the algorithm in a full parallel way, or a single board is used to execute each one of the processing steps in sequence by having its FPGAs reprogrammed 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.

This paper will focus on an example algorithm that requires a single FlexFilm board to be implemented. This example algorithm does not require the FPGAs to be reprogrammed at run time because it does not need more than the three available FlexWAFE FPGAs.

2.2. FlexWAFE reconfigurable architecture

The FPGAs are configured using macro components that consist of local memory address generators (LMC) that support sophisticated memory pattern transformations and data stream processing units (DPUs). Their sizes fit the typical FPGA blocks and they can be easily laid out as macro blocks reaching a clock rate of 125 MHz. They are parameterized in data word lengths, address lengths and supported address and data functions. The macros are programmed via address registers and function registers and have small local sequencers to create a rich variety of access patterns, including diagonal zigzagging and rotation. The adapted LMCs are assigned to local FPGA RAMs that serve as buffer and parameter memories. The macros are programmed at run time via a small and, therefore, easy to route control bus. A central algorithm controller (AC) sends the control instructions to the macros controlling the global algorithm sequence and synchronization. Programming can be slow compared to processing as the macros run local sequences independently. In effect, the macros operate as weakly-programmable coprocessors known from MpSoCs such as VIPER [10]. This way, weak programmability separates time-critical local control in the components from non-time-critical global control. This approach accounts for the large difference in global and local wire timing and routing cost. The result is similar to a local cache that enables the local controllers to run very fast because all critical paths are local. An example of this architecture is depicted in Figure 4. In this stream-oriented processing system, the input data stream enters the chip on the left, is processed by the DPUs along the datapath(s), and leaves the chip on the right side of the figure. Between some of the DPUs are LMC elements that act as simple scratch pads, FIFOs or reordering buffers, depending on their program and configuration. Some of the LMCs are used in a *cache* like fashion for the larger external SDRAM. The access to this off-chip memory is done via the CMC, which is described in detail in Section 2.4. The algorithm controller changes some of the parameters of the DPUs and LMCs at run time via the depicted parameter bus. The AC is (re-)configured by the control bus that connects it to the PCIe router FPGA (Figures 1 and 4).

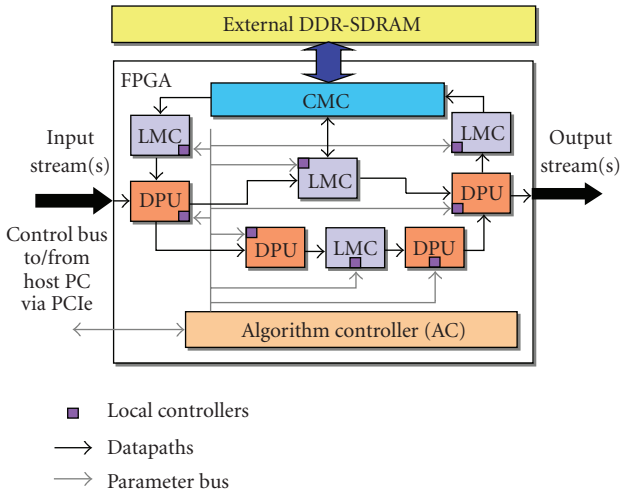


Figure 4: FlexWAFE reconfigurable architecture.

2.2.1. Related work

The Imagine stream processor [11] uses a three-level hierarchical memory structure: small registers between processing units, one 128 KB stream register file, and external SDRAM. It has eight arithmetic clusters each with six 32-bit FPUs (floating point units) that execute VLIW instructions. Although it is a stream-oriented processor, it does not achieve the theoretical maximum performance due to stream controller and kernel overhead.

Hunt engineering [12] provides an image processing block library—*imaging VHDL*—with some similarities with the FlexWAFE library, but their functionality is simpler (window-based filtering and convolution only) than the one presented in this paper.

Nallatech [13] developed the Dime-II (DSP and imaging processing module for enhanced FPGAs) architecture that provides local and remote functions for system control and dynamic FPGA configuration. However, it is more complex than FlexWAFE and requires more resources.

The SGI reconfigurable application-specific computing (RASC) [14] program delivers scalable configurable computing elements for the Altix family of servers and superclusters.

The methodology presented by Park and Diniz [15] is focused on application level stream optimizations and ignores architecture optimizations and memory prefetching.

Oxford Micro Devices' A436 Video DSP Chip [16] operates like an ordinary RISC processor except that each instruction word controls the operations of both a scalar arithmetic unit and multiple parallel arithmetic units. The programming model consists of multiple identical operations that are performed simultaneously on a parallel operand. It performs one 64-point motion estimation per clock cycle and 3.2 G multiply accumulate operations (MAC) per second.

The leading Texas Instruments fixed-point TMS320C64x DSP running at 1 GHz [17] reaches 0.8 Gop/s, and the leading Analog Devices TigerSHARC ADSP-TS201S DSP operates at 600 MHz [18] and executes 4.8 Gop/s.

Motion estimation is the most computationally intensive part of our example algorithm. Our proposed architecture computes 155 Gop/s in a bidirectional 256 point ME. The Imagine running at 400 MHz reaches 18 Gop/s. The A436 Video DSP has 8 dedicated ME coprocessors, but it can only calculate 64 point ME over standard resolution images.

Graphics processing units (GPUs) can also be used to do ME, but known implementations [19] are slower and operate with smaller images than our architecture. Nvidia introduced a ME engine in their GeForce 6 chips, but it was not possible to get details about its performance.

The new IBM cell processor [20] might be better suited than a GPU, but it is rather optimized to floating point operations. A comparable implementation is not known to the authors.

2.3. Global data flow

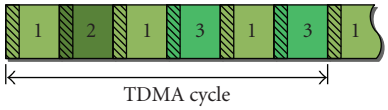
Even if only operating at the low 2 K resolution, one image stream alone comes at a data rate of up to 3.1 Gbit/s. With the upcoming 4 K resolution, one stream requires a net rate of 12.4 Gbit/s. At the processing stage, this bandwidth rises even higher, for example, because multiple frames are processed at the same time (motion estimation) or the internal channel bit resolution increases to keep the desired accuracy (required by filter stages such as DWT). Given the fact that the complete algorithm has to be mapped to different FPGAs, data streams have to be transported between the FPGAs and—in case of future multi-board solutions—between boards. These data streams might differ greatly in their characteristics such as bandwidth and latency requirements (e.g., image data and motion vectors), and it is required to transport multiple streams over one physical communication channel. Minimum bandwidths and maximum possible latencies must be guaranteed.

Therefore, it is obvious that the communication architecture is a key point of the complete FlexFilm project. The first decision was to abandon any bus structure communication fabric, since due to their shared nature, the available effective bandwidth becomes too limited if many streams need to be transported simultaneously. Furthermore, current bus systems do not provide a quality of service management, which is required for a communication scheduling. For this reason, point-to-point channels were used for inter-FPGA communication and PCI-Express was selected for board-to-board communication. Currently, PCI-Express is only used for stream input and output to a single FlexFilm board, however in the future multiple boards will be used.

It should be clarified that real-time does not always means the full 24 (or more) FPS. If the available bandwidth or processing power is insufficient, the system should function at a lesser frame rate. However, a smooth degradation is required without large frame rate jitter or frame stalls.

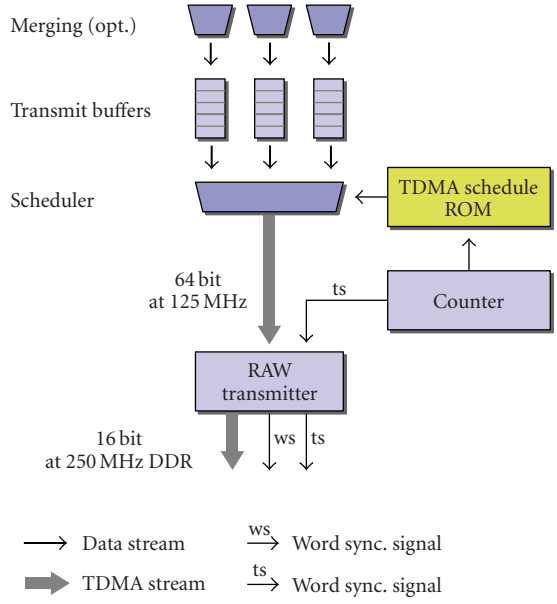
Furthermore, the system is noncritical, which means that under abnormal operating conditions such as short bandwidth drops of the storage system a slight frame rate jitter is allowed *as long as this does not happen regularly*. Nevertheless, *even in such abnormal situations the processing results have to be correct*. It has to be excluded that these conditions result in

(a) TDM with variable packet size, one packet per cycle and stream.



(b) TDM with variable packet size, multiple packets per cycle and stream.

Figure 5: Chip-to-chip transmitter.



Data valid and enable signals omitted for readabillity.

Figure 6: Chip-to-chip transmitter.

data losses due to buffer overflows or underruns or in a complete desynchronization of multiple streams. This means that back pressuring must exist to stop and restart data transmission and processing reliably.

2.3.1. FPGA-to-FPGA communication

As explained above, multiple streams must be conveyed reliably over one physical channel. Latencies should be kept at a minimum, since large latencies require large buffers which have to be implemented inside the FlexWAFE FPGAs and which are nothing but "dead weight." Since the streams (currently) are periodic and their bandwidth is known at design time, TDM[1] (time division multiplex) scheduling is a suitable solution. TDM means that each stream is granted access to the communication channel in slots at fixed intervals. The slot assignment can be done in the following two ways: (a) one slot per stream and TDM cycle, the assigned bandwith is determined by the slot length (Figure 5(a)) and (b) multiple slots at fixed length per stream and TDM cycle (Figure 5(b)). Option (a) requires larger buffer FIFOs because larger packets have to be created, while option (b) might lead to a bandwidth decrease due to possible packet header overhead.

For the board-level FPGA-to-FPGA communication, option (b) was used since no packet header exists. The communication channel works at a "packet size" of 64 bit. Figure 6 shows the communication transmit scheduler block diagram. The incoming data streams which may differ in clock rate and word size are first merged and zero-padded to 64-bit *raw* words and then stored in the transmit FIFOs. Each clock cycle, the scheduler selects one raw word from one FIFO and forwards it to the raw transmitter. The TDM schedule is stored in a ROM which is addressed by a counter. The TDM schedule (ROM content) and the TDM cycle length (maximum counter value) are set at synthesis time. The communication receiver is built up in an analogous way (demultiplexing, buffering, demerging). To synchronize transmitter and receiver, a synchronization signal is generated at TDM cycle start and transmitted using one of the four sideband control signals.

As explained in Section 2.1, the raw transmitter-receiver pair transmits one 64-bit raw word per clock cycle (125 MHz) as four 16-bit words at the rising and falling edge of the 250 MHz transmit clock. For word synchronization, a second sideband control signal is used.

The remaining two sideband signals are used to signal arrival of a valid data word and for back pressuring (not shown in Figure 6).

Table 1 shows an example TDM schedule (slot assignment) with 3 streams, two 2 K RGB streams at 3.1 Gbit/s with a word size of 30 bit and one luminance stream at 1.03 Gbit/s with a word size of 10 bit. The stream clock rate f_{stream} is a fraction of the core clock rate $f\,\text{clk} = 125\,\text{MHz}$, which simply means that not on every clock cycle one word is transmitted. All streams are merged and zero-padded to 64-bit streams. The resulting schedule length is 12 slots, and the allocated bandwidth for the streams are 3.125 Gbit/s and 1.25 Gbit/s.

2.3.2. Board communication

Since PCI-Express can be operated as a TDM bus, the same scheduling techniques apply as for the inter-FPGA communication. The only exception is that PCI-Express requires a larger packet size of currently up to 512 bytes.[2] The required buffers however fit well into the IO-FPGA.

[1] Also referred as TDMA (time division multiple access).

[2] Limitation by the currently used Xilinx PCI-Express IP core.

Table 1: TDM example schedule.

Stream	Requirements			Merging + padding		TDM scheduling		Result	
	BW (Gbit/s)	width (bits)	f_{stream} (MHz)	n_{merge}	f_{64} (MHz)	n_{slots}	f_{TDM} (MHz)	real BW (Gbit/s)	Over-allocation
1 (RGB)	3.1	30	103.3	2	51.65	5 of 12	52.08	3.125	0.8%
2 (RGB)	3.1	30	103.3	2	51.65	5 of 12	52.08	3.125	0.8%
3 (Y)	1.03	10	103.3	6	17.22	2 of 12	20.8	1.25	21%
total	6	—	—	—	—	12	—	7.5	—

TDM schedule: 1 2 1 2 1 3 2 1 2 1 2 3.
TDM cycle length: 12 slots = 12 clock cycles; $f_{slot} = f_{sys}/12 = 10.41$ MHz.

n_{merge} Merging factor, how many words are merged into one 64-bit RAW word. Zero-padded to full 64 bit.
n_{slots} Assigned TDM slots per stream.
f_{stream} Required stream clock rate to achieve desired bandwidth at given word size.
f_{64} Required stream clock rate to achieve desired bandwidth at 64 bit.
f_{slot} Frequency of one TDM slot.
f_{sys} System clock frequency (125 MHz).
f_{TDM} Resulting effective stream clock rate at current TDM schedule: $f_{TDM} = n_{slots} \cdot f_{slot}$.

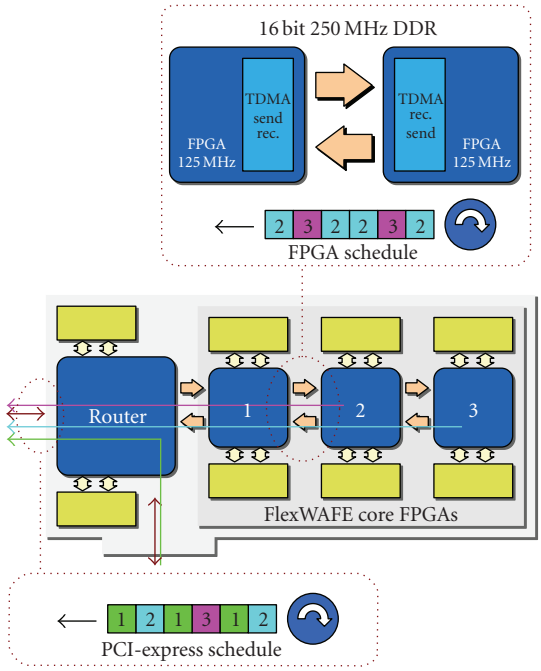


Figure 7: Communication scheduling.

Figure 7 shows an inter-FPGA and a PCI-Express schedule example.

2.4. Memory controller

As explained in the introduction, external SDRAM memories are required for storing image data. The 125 MHz clocked DDR-SDRAM reaches a peak performance per channel of 8 Gbit per second. To avoid external memory access becoming a bottleneck, an access optimizing scheduling memory controller (CMC[3]) was developed which is able to handle multiple, independent streams with different characteristics (data rate, bit width). This section will present the memory controller architecture.

2.4.1. Quality of service

In addition to the configurable logic, each of the four XC2PV50-6 FPGAs FPGA contains two embedded PowerPC processors, equipped with a 5-stage pipeline, data, and instruction caches of 16 KiByte each and running at a speed of up to 300 MHz. In the FlexFilm project, these processors are used for low computation and control-dominant tasks such as global control and parameter calculation. CPU code and data are stored in the existing external memories which leads to conflicts between processor accesses to code, to internal data, and to shared image data on the one hand, and memory accesses of the data paths on the other hand. In principle, CPU code and data could be stored in separate dedicated memories. However, the limited on-chip memory resources and pin and board layout issues renders this approach too costly and impractical. Multiple independent memories also do not simplify access patterns since there are still shared data between the data path and CPU. Therefore, the FlexFilm project uses a shared memory system.

A closer look reveals that data paths and CPU generate different access patterns as follows.

(a) Data paths: data paths generate a fixed access sequence, possibly with a certain arrival jitter. Due to the real-time requirement, the requested throughput has to be guaranteed by the memory controller (*minimum memory throughput*). The fixed address sequence allows a deep prefetching and usage of FIFOs to increase

[3] Central memory controller; historic name, emerged when it was supposed to only have one external memory controller per FPGA.

the maximum allowed access latency—even beyond the access period—and to compensate for access latency jitter. Given a certain FIFO size, the maximum access time must be constrained to avoid buffer overflow or underflow, but by adapting the buffer size, arbitrary access times are acceptable.
The access sequences can be further subdivided into periodic regular access sequences such as video I/O and complex nonregular (but still fixed) access patterns for complex image operations. The main difference is that the nonregular accesses cause a possible higher memory access latency jitter, which leads to smaller limits for the maximum memory access times, given the same buffer size.
A broad overview about generating optimized memory access schedules is given by [21].

(b) CPU: processor access, in particular cache miss accesses generated by nonstreaming, control-dominated applications, shows a random behavior and are less predictable. Prefetching and buffering are, therefore, of limited use. Because the processor stalls on a memory read access or a cache read miss, memory access time is the crucial parameter determining processor performance. On the other hand, (average) memory throughput is less significant. To minimize access times, *buffered and pipelined latencies* must be minimized.
Depending on the CPU task, access sequences can be either hard or soft real-time. For hard real-time tasks, a minimum throughput and maximum latencies must be guaranteed.

Both access types have to be supported by the memory controller by quality of service (QoS) techniques. The requirements above can be translated to the following two types of QoS:

(i) guaranteed minimum throughput at guaranteed maximum latency
(ii) smallest possible latency; (at guaranteed minimum throughput and maximum latency).

2.4.2. Further requirements

Simple, linear first-come first-served SDRAM memory access can easily lead to a memory bandwith utilization of only about 40%, which is not acceptable for the FlexFilm system. By performing memory access optimization, that is by executing and possibly reordering memory requests in an optimized way to utilize the multibanked buffered parallel architecture of SDRAM architectures (bank interleaving [22, 23]) and to reduce stall cycles by minimizing bus tristate turnaround cycles, an effectiveness of up to 80% and more can be reached. A broad overview of these techniques is given in [24].

Since the SDRAM controller does not contribute to the required computations (although absolutely required) it can be considered as "ballast" and should use as little resources as possible, preferably less than 4% of total available FPGA resources per instance. Compared to ASIC-based designs, at the desired clock frequency of 125 MHz the possible logic complexity is less for FPGAs and, therefore, the possible arbitration algorithms have to be carefully evaluated. Deep pipelining to achieve higher clock rates is only possible to a certain level leads to an increasing resource usage and is contrary to the required minimum latency QoS requirement explained above.

Another key issue is the required configurability at synthesis time. Different applications require different setups, for example, different number of read and write ports, client port widths, address translation parameters, QoS settings, and also different SDRAM layouts (32- or 64-bit channels). Configuring by changing the code directly or defining constants is not an option as this would have inhibited or at least complicated instantiation of multiple CMCs with different configurations within one FPGA (as we will see later, the motion estimation part of the example application needs 3 controllers with 2 different configurations). Therefore, the requirement was to only use VHDL *generics* (VHDL language constructs that allow parameterizing at compile-time) and use coding techniques such as deeply nested if/for generate statements procedures to calculate dependant parameters to have the code self-adapt at synthesis-time.

2.4.3. Architecture

Figure 8 shows the controller block diagram (example configuration with 2 low latency and 2 standard latency ports, one read and one write port each and 4 SDRAM banks). The memory controller accesses the SDRAM using auto precharge mode and requests to the controller are always done at full SDRAM bursts at a burst length of 8 words (4 clock cycles). The following sections will give a short introduction into the controller architecture, a more detailed description can be found in [25, 26].

Address translation

After entering the read (r) or write (w) ports, memory access requests first reach the address translation stage, where the logical address is translated into the physical bank/row/column triple needed by the SDRAM. To avoid excessive memory stalls due to SDRAM bank precharge and activation latencies, SDRAM accesses have to be distributed across all memory banks as evenly as possible to maximize their parallel usage (known as bank interleaving). This can be achieved by using low-order address bits as bank address since they show a higher degree of entropy than high-order bits. For the 4-bank FlexFilm memory system, address bits 3 and 4 are used as bank address bits; bits 0 to 2 cannot be used since they specify the start word of the 8-word SDRAM burst.

Data buffers

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 SDRAM is reserved. To

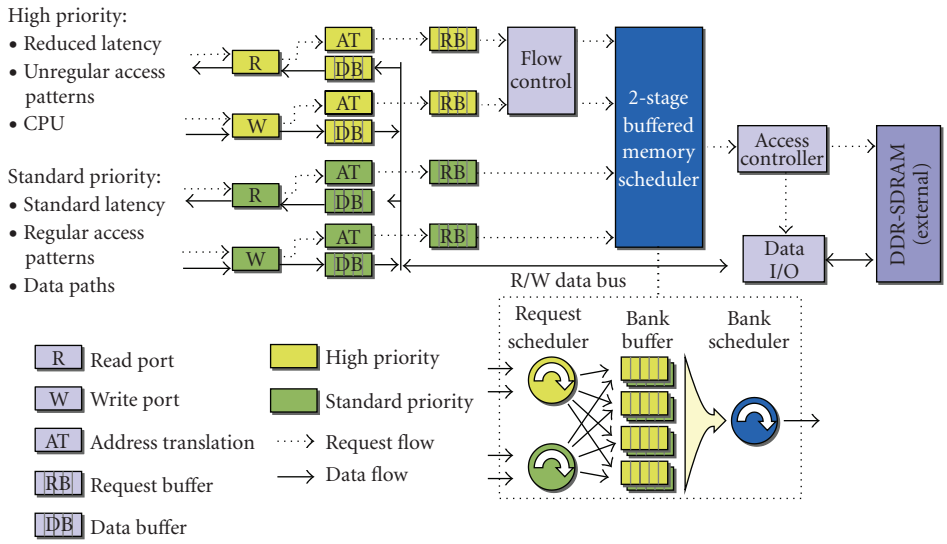


Figure 8: Memory controller block diagram.

address the correct buffer slot later, a tag is created and attached to the request. This technique reduces the significant overhead needed if the write-data would be carried through the complete buffer and scheduling stages and allows for an easy adaption of the off-chip SDRAM data bus width to the internal data paths due to possible usage of special two-ported memories. It also hides memory write latencies by letting the write requests passing through the scheduling stages while the data is arriving at the buffer.

For reads requests the data buffer is also responsible for transaction reordering, since read requests from one port to different addresses might be executed out-of-order due to the optimization techniques applied. The application however expects reads to be completed in-order.

Request buffer and scheduler

The requests are then enqueued in the request buffer FIFOs which decouple the internal scheduling stages from the clients. The first scheduler stage, the request scheduler, selects requests from several request buffer FIFOs, one request per two clock cycles, and forwards them to the bank buffer FIFOs (flow control omitted for now). By applying a rotary priority-based arbitration similar to [27], a minimum access service level is guaranteed.

Bank buffer and scheduler

The bank buffer FIFOs store the requests sorted by bank. The second scheduler stage, the bank scheduler, selects requests from these FIFOs and forwards them to the tightly coupled access controller for execution. In order to increase bandwidth utilization, the bank scheduler performs *bank interleaving* and *request bundling*. Bank interleaving reduces memory stall times by accessing other memory banks if one bank is busy; request bundling is used to minimize data bus direction switch tristate latencies by rearranging changing read and write request sequences to longer sequences of one type.

Like with the request scheduler, by applying a rotary priority-based arbitration a minimum access service level for any bank is guaranteed.

Access controller

After one request has been selected, it is executed by the access controller and the data transfer to (from) the according data buffer is started. The access controller is also responsible for creating SDRAM refresh commands in regular intervals and performing SDRAM initialization upon power-up.

Quality of service

As explained above, for CPU access a low-latency access path has to be provided. This was done by creating an extra access pipeline for low-latency requests (separate request scheduler and bank buffer FIFOs). Whenever possible, the bank scheduler selects low-latency requests, otherwise standard requests.

This approach already leads to a noticeable latency reduction, however a high low-latency request rate causes stalls for normal requests that must be avoided. This is done by the flow control unit in the low-latency pipeline, which reduces the maximum possible low-latency traffic. To allow a bursty memory access,[4] the flow control unit allows n request to pass within a window of T clock cycles (known as "sliding window" flow control in networking applications).

2.4.4. *Configurability*

The memory controller is configurable regarding SDRAM timing and layout (bus widths, internal bank/row/column

[4] Not to be confused with SDRAM bursts!

organization), application ports (number of ports, different data, and address widths *per port*), address translation per port, and QoS settings (prioritization and flow control).

As required, configuration is done almost solely via VHDL generics. Only a few global configuration constants specifying several maximum values (e.g., maximum port address width, ...) are required, which do not, however, prohibit instantiation of multiple controllers with different configurations within one design.

2.4.5. Related work

The controllers by Lee et al. [28] and Sonics [29], and Weber [30] provide a three-level QoS: "reduced latency," "high throughput," and "best effort." The first two levels correspond to the FlexFilm memory controller with the exception that the high throughput level is also bandwidth limited. Memory requests at the additional third level are only scheduled if the memory controller is idle. The controllers further provide a possibility to degrade high priority requests to "best effort" if their bandwith limit is exceeded. This however can be dangerous, as it might happen in a highly loaded system that a "reduced latency" request observes a massive stall after possible degradation—longer than if the request would have been backlogged until more "reduced latency" bandwidth becomes available. For this reason, degradation is not provided by the CMC. Both controllers provide an access-optimizing memory backend controller.

The access-optimizing SDRAM controller framework presented by Macián et al. [31] provides bandwidth limitation by applying a token bucket filter, however they provide no reduced latency memory access.

The multimedia VIPER MPSoC [10] chip uses a specialized 64-bit point-to-point interconnect which connects multiple custom IP cores and 2 processors to a single external memory controller. The arbitration inside the memory controller uses runtime programmable time-division multiplexing with two priorities per slot. The higher priority guarantees a maximum latency, the lower priority allows the leftover bandwidth to be used by other clients (see [32]). While the usage of TDM guarantees bandwidth requirements and a maximum latency per client, this architecture does not provide a reduced latency access path for CPUs. Unfortunately, the authors do not provide details on the memory backend except that it performs access optimization (see [32, chapter 4.6]). For the VIPER2 MPSoC (see [32, chapter 5]), the point-to-point memory interconnect structure was replaced by a pipelined packetized tree structure with up to three runtime programmable arbitration stages. The possible arbitration methods are TDM, priorities, and round robin.

The memory arbitration scheme described by Harmsze et al. [33] gives stream accesses a higher priority for M cycles out of a service period of N cycles, while otherwise ($R = N - M$ cycles) CPU accesses have a higher priority. This arbitration scheme provides a short latency CPU access while it also guarantees a minimum bandwidth for the stream accesses. Multiple levels of arbitration are supported to obtain dedicated services for multiple clients. Unfortunately, the authors do not provide any information on the backend memory controller and memory access optimization.

The "PrimeCell^{TM} Dynamic Memory Controller" [34] IP core by ARM Ltd. is an access-optimizing memory controller which provides optional reduced latency and maximum latency QoS classes for *reads* (no QoS for writes). Different from other controllers, the QoS class is specified per request and not bound to certain clients. Furthermore, memory access optimization supports out-of-order execution by giving requests in the arbitration queue different priorities depending on QoS class and SDRAM state.

However, all of these controllers are targeted at ASICs and are, therefore, not suited for the FlexFilm project (too complex, lack of configurability).

Memory controllers from Xilinx (see [35]) do not provide QoS service and the desired flexible configurability. They could be used as backend controllers, however they were not available at time of development.

The memory controller presented by Henriss et al. [36] provides access optimization and limited QoS capabilities, but only at a low flexibility and with no configuration options.

3. A SOPHISTICATED NOISE REDUCER

To test this system architecture, a complex noise reduction algorithm depicted in Figures 9 and 10 based on 2.5-dimensions discrete wavelet transformation (DWT will be explained in Section 3.3) between consecutive motion compensated images was implemented at 24 fps. The algorithm begins by creating a motion compensated image using pixels from the previous and from the next image. Then it performs a Haar filter between this image and the current image. The two resulting images are then transformed into the 5/3 wavelet space, filtered with user selectable parameters, transformed back to the normal space and filtered with the inverse Haar filter. The DWT operates only in the 2D space-domain; but due to the motion-compensated pixel information, the algorithm also uses information from the time domain; therefore, it is said to be a 2.5D filter. A full 3D filter would also use the 5/3 DWT in the time domain, therefore, requiring five consecutive images and the motion estimation/compensation between them. The algorithm is presented in detail in [37].

3.1. Motion estimation

Motion estimation (ME) is used in many image processing algorithms and many hardware implementations have been proposed. The majority are based on block matching. Of these, some use content dependent partial search. Others search exhaustively in a data-independent manner. Exhaustive search produces the best block matching results at the expense of an increased number of computations.

A full-search block-matching ME operating in the luminance channel and using the sum of absolute differences (SAD), search metric was developed because it has predictable content-independent memory-access patterns and can process one new pixel per clock cycle. The block size is

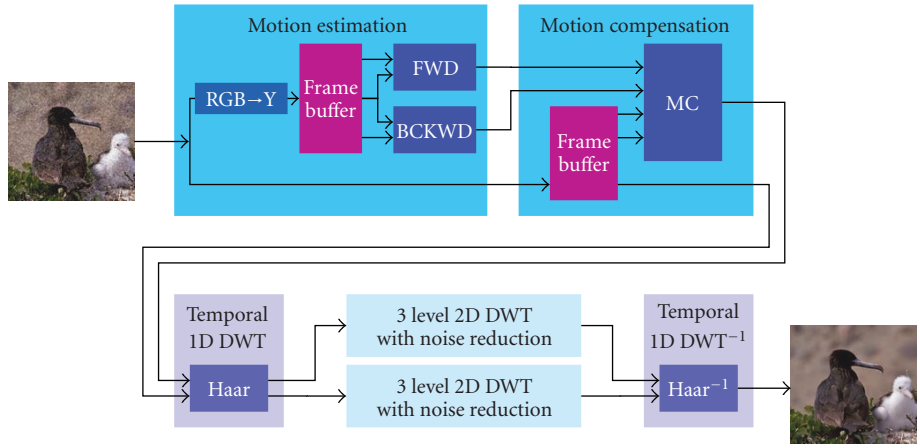


Figure 9: Advanced noise-reduction algorithm.

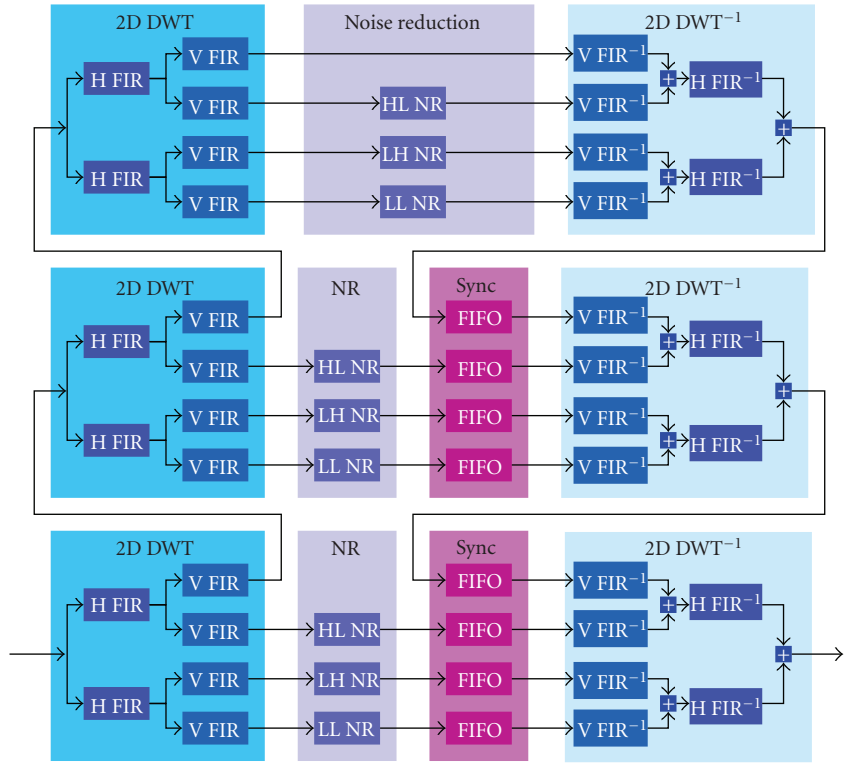


Figure 10: Three-level DWT-based 2D noise reduction.

16×16 pixels and the search vector interval is $-8/+7$. Its implementation is based on [38]. Each of the 256 processing elements (PE) performs a 10-bit difference, a comparison, and a 18-bit accumulation. These operations and their local control was accommodated in 5 FPGA CLBs (configurable logic blocks) as shown in Figure 11. As seen in the rightmost table of that figure, the resource utilization within these 5 CLBs is very high and even 75% of the LUTs use all of its four inputs. This block was used as a relationally placed macro (RPM) and evenly distributed on a rectangular area of the chip. Unfortunately each 5 CLBs only have 10 tristate buffers which is not enough to multiplex the 18-bit SAD result. Therefore, the PEs are accommodated in groups of 16 and use 5 extra CLBs per group to multiplex the remaining 8 bits. Given the cell-based nature of the processing elements, the timing is preserved by this placement. To implement the 256 PEs with

Resource	Usage	Percentage
RAMB	62 out of 232	26%
Slices	19.157 out of 23.616	81%
TBUF	5.408 out of 11.808	45%
FF	24.752 out of 47.232	52%

- Bidirectional ME with block size 16×16
- Bidirectional MC
- Search $-8/+7$ vector interval
- 24 fbs at 2048×2048, 10 bpp (125 MHz)
- 1024 net add/sub opreations/pixel
- 514 net comparations operations/pixel
- 155 net Goperations/s

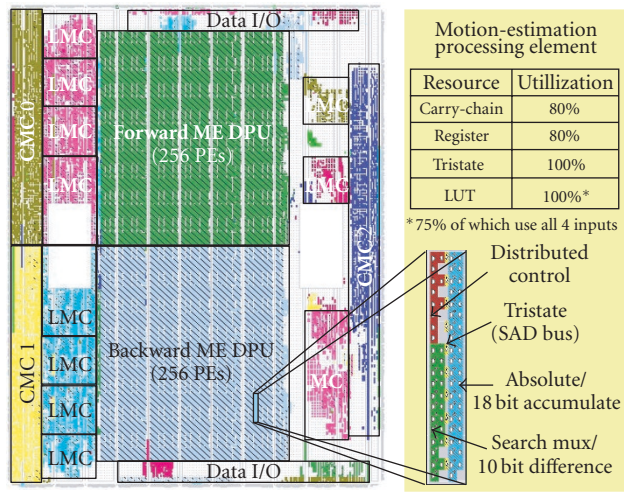


Figure 11: Mapping and resource usage in a Xilinx XC2V50P device.

corresponding SAD bus, 1360 CLBs, and 59 extra CLBs are required for distributed control and finding the minimum SAD. On the edge of the images the motion vectors can only have limited values, a fact that is used to reduce the initial row latency of [38] from 256 to 0.

Bidirectional motion-estimation is achieved using two of these blocks. The ME core-processing elements require that the images be presented at its inputs in a column-major way, but the images are transferred between FPGAs and stored in SDRAM in a row-major order. Therefore, each of the PE's three inputs gets data from memory via a group of two LMCs. The first hides the SDRAM latency by performing prefetching as explained in [4], while the second transforms the accesses from row-major to column-major order using a small local blockRAM. When fed with the luminance component of 2048×2048 pixels, 10 bit-per-pixel images at 24 frames per second, the core computational power (ignoring control and data transfers) is 155 Gop/s. The intra-PE bandwidth is 1 Tbit/s.

3.2. Motion compensation

Motion compensation (MC) uses the block motion vectors found by the ME to build an image that is visually similar to the current image, but only contains pixels extracted in a blockwise manner from the previous/next image. The criterion to choose the image block from the previous or next image is the SAD associated with that block, the image block with the smallest SAD (and, therefore, more similar to the current image block) of the two is chosen. On a scene cut, one of the images will produce large SADs (because the contents of it are from a different scene and most probably completely different from the current image) and all blocks will be chosen from the other image, the one that belongs to the same scene. This has the advantage of making the noise reduction algorithm *immune* to scene cuts.

3.3. Discrete wavelet transform

The discrete wavelet transform (DWT) transforms a signal into a space where the base functions are wavelets [39], similar to the way Fourier transformation maps signals to a sine-cosine-based space. The 5/3 wavelet was chosen for its integer coefficients and invertibility (the property to convert back to the original signal space without data loss). The 2D wavelet transformation is achieved by filtering the row major incoming stream with two FIR filters (one with 5, the other with 3 coefficients) and then filtering the resulting two signals columnwise using the same filter coefficients. The four resulting streams can be transformed back to the original stream by filtering and adding operations. The noise reduction algorithm requires three levels of decomposition, therefore, three of these blocks were cascaded and the noise reduction DPUs added (Figure 10). To compensate the latency of the higher decomposition levels, LMCs were used to build FIFOs using internal FPGA RAM (mid *sync FIFOs* on Figure 10) and using external SDRAM via a CMC (bottom *sync FIFOs* on Figure 10). The resulting system was presented in detail in [4].

The filter implementation uses polyphase decomposition (horizontal) and coefficient folding (vertical). To maximize throughput, the transformation operates line-by-line instead of level-by-level [40]. This allows for all DPUs to operate in parallel (no DPU is ever idle), minimizes memory requirements, and performs all calculations as soon as possible. Because the 2D images are a finite signal, some control was added to achieve the symmetrical periodic extension (SPE) [41] required to achieve invertibility. This creates a dynamic datapath because the operations performed on the stream depend on the data position within the stream. Almost all multiply operations were implemented with shift-add operations because of the simplicity of the coefficients used. One 2D DWT FPGA executes 162 add operations on the direct DWT, 198 add operations on the inverse DWT, and 513 extra

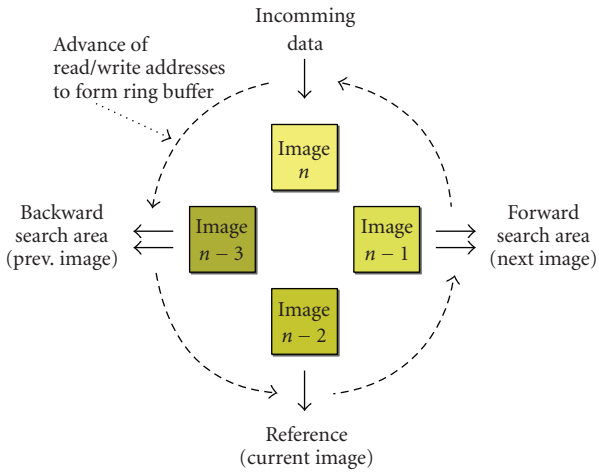


Figure 12: Frame buffer access sequence.

add operations to support the SPE, all between 10 and 36 bits wide.

3.4. External memory

Figure 12 shows the required frame buffer access structure of the motion estimation. As can be seen, three images are accessed simultaneously, one image as reference ($n-2$), and two images as backward and forward search area ($n-3$ and $n-1$). The two search areas are read twice with different addresses. Besides that, the current incoming image (n) needs to be buffered. Each of the two ME engines contains its own frame buffer to store four full-size images of up to $4\,\mathrm{K} \times 4\,\mathrm{K}$ accessed via its respective CMC0 or CMC1 (Figure 11). Each of the CMCs writes one stream to memory and reads three streams. For ease of implementation, each pixel is stored using 16 bits (only 10-bit luminance per pixel are valid). This translates to 1.5 Gbit/s write and 4.1 Gbit/s read bandwidth to off-chip SDRAM, amounting to a total of 6.1 Gbit/s, which is below the maximum practical bandwidth of 6.4 Gbit/s. The MC block operates in the RGB color space unlike the ME block that uses the luminance only. It stores one RGB pixel in a 32-bit word (10 bits per color component) and uses its own memory controller (CMC2 on Figure 11). It uses a similar ring-buffer scheme as CMC0 and 1 and is also capable of storing four images of up to $4\,\mathrm{K} \times 4\,\mathrm{K}$ resolution, but it groups the two external memory banks and accesses them via a 64-bit bus and is therefore capable of twice the troughput of the ME's CMCs. Due to the nature of SDRAM accesses, it is only possible to access blocks of 16 pixels at addresses that are multiples of 16 (memory alignment). This means that in the worst case, two blocks of 16 pixels need to be fetched in order to access a nonaligned group of 16 pixels to build the motion compensated image. The MC block also needs to access the current image in order to do intrablock pixel-by-pixel validation of the results. This leads to a worst-case bandwidth of 3.0 Gbit/s write and 9.2 Gbit/s read, which is below the practical limit of 12.8 Gbit/s. The area occupied by these 3 memory controllers is about 12% of the FPGA area, leaving enough room for the stream processing units.

As explained in Section 3.3, the DWTs need synchronization FIFOs to compensate the additional latency of the higher level DWTs. The level 2 FIFOs completely fit into the FPGA internal memory, so only for the level 1 FIFOs, external SDRAM was required. Due to layout issues, two memory controllers in a 64-bit configuration were used for separate buffering of the red channel and the green/blue channels.

The ME/MC-FPGA requires 3 CMCs in 2 different configurations; the DWTs each require 2 controllers in 2 configurations. Together with the 2 controllers in the router FPGA, 9 memory controllers in 5 configurations were used all together.

Since in this application the PowerPC and therefore the QoS features are not (yet) used, these features were separately tested; see Section 4.

3.5. Mapping and communication

The complete algorithm was mapped onto the three FlexWAFE image processing FPGAs of a single FlexFilm board. Stream input and output is done via the router FPGA and the PCI-Express host network. The second PCI-Express port remains unused. Input and output streams require a net bandwidth of 3 Gbit/s each, which can be easily handled by the X4 PCI-Express interface. Since only single streams are transmitted, no TDM scheduling is necessary. The packetizing and depacketizing of the data, as well as system jitter compensation, are done by double-buffering the incoming and outgoing images in the external RAM.

Figure 13 shows the mapping solution (router FPGA omitted). The first FlexWAFE FPGA contains the motion estimation and compensation, the second FPGA the Haar and inverse Haar filters and one 2D DWT noise reduction block, the third FPGA contains the other 2D DWT noise reduction block.

As can be seen, between the first and the second FPGA two independent image streams—the original and the motion compensated images—with a total bandwidth of 6 Gbit/s need to be transported over one physical channel. The word size of both streams is 30 bit (10 bit RGB). For transport, always two words of each stream were merged and zero-padded to form a 64-bit word. The TDM scheduler was programmed with (in this case a simple) sequence of 1-2, which means that the 64-bit words were transmitted in an alternating way. Due to the small TDM packet size of two words, no external SDRAM is required for buffers.

Due to the mapping onto different SDRAM channels as explained in Section 3.4, the maximum effective bandwidth per SDRAM channel of 6.4 Gbit/s (12.8 Gbit/s for a 64-bit combined channel, respectively) was not an issue.

3.6. Implementation

Each building block has been programmed in VHDL using an extensive number of *generics* to increase the flexibility and reuse. The sequence of run-time programmable parameters for the LMCs image transfers (the contents of the AC

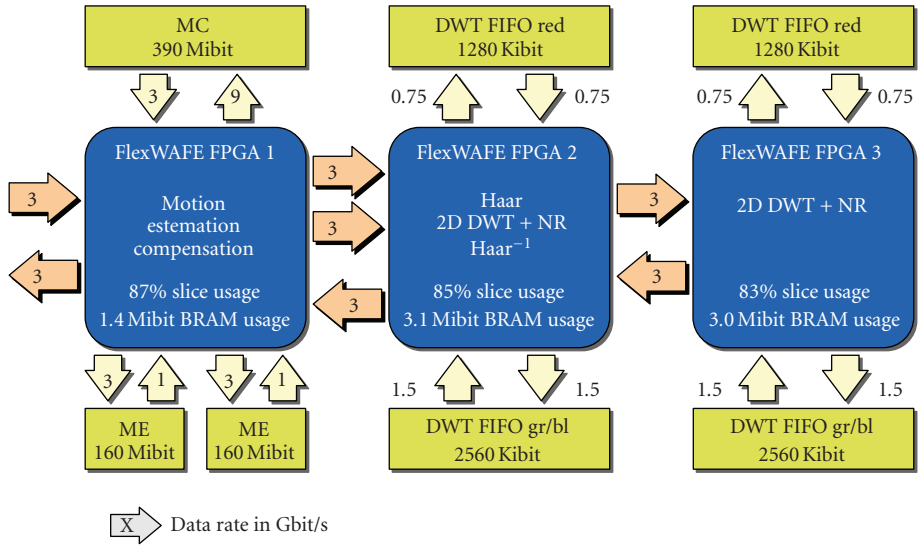


Figure 13: Algorithm mapping.

memory) were described in XML and transformed to VHDL via a PHP script. In the future, it is planned to use even more scripts and XML based high-level system descriptions. Each block was behaviorally simulated individually using Modelsim 6.2d and synthesized using Xilinx ISE 7.1i SP4 (because ISE 8.xi 9.1i had regressions). All blocks except the motion compensation have been tested in hardware and the desired functionality and speed were achieved.

Currently, the ME and MC are being integrated in a single chip (Figure 11), and due to the large resource utilization (81% of the FPGA slices are used) floorplanning is necessary to achieve the required speed. So far the Xilinx Floorplanner and PlanAhead 8.2.2 have been used, but in the future it is planned to use Synplicity Premier 8.6.3 with Design Planner.

3.7. Outlook

Currently, the 56 filter parameters used by the DWT filter are static. However, in [37] it is shown that the results can be significantly improved by run-time adapting the filter coefficients depending on the image. The required calculations will be done by a PowerPC which will have to access parts of the images in the motion compensation FPGA frame memory, thus requiring the QoS service features of the memory controller.

4. SDRAM QOS

In [25, 26], a complex simulator setup was used before availability of the FlexFilm board to evaluate the CMC QoS architecture. However, due to lack of a cycle-accurate instruction set simulator for the embedded PowerPC 405 core, these results were inaccurate. Therefore, a real test environment was created, consisting of the PowerPC, two SDRAM controllers, two load generators, and the required PowerPC-CMC interfaces (Figure 14).

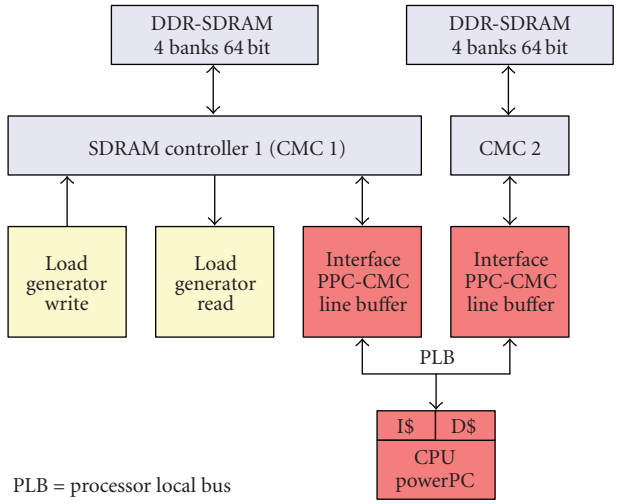


Figure 14: CMC QoS test environment.

The load generators (one read and one write) created memory access streams with linear address patterns similar to the DWT filters and a programmable period. Requests to the SDRAM were done at 64 bit and a burst length of 8 words, which means the maximum possible period is 8 clock cycles. Since one SDRAM data transfer takes 4 clock cycles (8 words at 64 bits, 2 words per clock cycle), two load generators running at a period of 8 clocks would have created a theoretical SDRAM load of 100%. However, due to memory stall cycles (refresh cycles, bus switching stall cycles), a maximum period of 10/9 (or 9/10) was possible resulting in a bandwidth utilization of 72%. If the period is too small, the load generators start losing memory requests (they cannot operate in real-time any more).

Table 2: SDRAM controller test results. Flow control T/n: n requests within T clock cycles.

Number	Pri. CPU access	Flow control T/n	Load gen. period		CPU exec. time (ms)	Lost requests	
			Read	Write		Read	Write
1.1	No	Deactivated	12	12	2195	0	0
1.2	No	Deactivated	12	11	2217	0	0
1.3	No	Deactivated	11	11	2263	5	7
2.1	Yes	Deactivated	12	12	2119	258	175
2.2	Yes	Deactivated	12	11	2121	4324	7598
2.3	Yes	Deactivated	11	11	2121	25872	25871
3.1	Yes	32/1	12	12	2144	0	0
3.2	Yes	61/2	12	12	2123	0	0
3.3	Yes	45/1	12	11	2169	0	0
3.4	Yes	93/2	12	11	2163	0	0
3.5	Yes	57/1	11	11	2209	0	0
3.6	Yes	113/2	11	11	2193	0	0

The PowerPC was clocked at 250 MHz and executed an adapted version of the JPEG decompression program from the MiBench benchmark suite [42] (we have chosen a real application rather than artificial benchmarks like SPEC for more realistic results). Code and data were completely mapped to the first memory controller. Since the original program accessed the harddisk to read and write data, in our environment the Xilinx Memory-FileSystem was used which was mapped to the second memory controller. Both PPC instruction and data caches (16 K each) were activated, at a cache miss 4 words at 64 bits were read and/or written to the memory. Since one memory access burst reads or writes 8 words at 64 bits, the PowerPC-CMC interface contains an additional line buffer (1-burst-cache).

CPU accesses to the first memory controller were optionally priorized and flow controlled. Table 2 shows the test results.

With the CPU activated, but without prioritization (and flow control) the load generators start losing requests (that means missing the deadline) at periods of 11/11 (no. 1.1 to 1.3). Activating CPU priorities leads to a noticeable CPU speedup, however the load generators start losing requests very quickly, with results getting worse at periods of 11/11 (no. 2.1 to 2.3). With flow control enabled, there is still a CPU speedup compared to the nonpriorized test; however, this time the load generators are fully operational again (no. 3.1 to 3.6). Moreover, results 3.5 and 3.6 show that with CPU and traffic shaping enabled load periods of 11/11 are possible, which is not the case without any QoS service (1.3). Activating complex traffic shaping patterns shows a positive albeit very small effect.

5. CONCLUSION

A record performance reconfigurable HW/SW platform for digital film applications was presented. The combination of programmable and parameterized macros that can easily be handled in floorplanning and decentralized weak programmability with noncritical timing was key to a high designer productivity.

We have also shown that the scheduling memory controller with QoS support helps to improve the overall system performance.

The FPGA resource utilization is very satisfactory including memory and routing resources. The FlexWAFE architecture is part of a larger project towards an extendible PCI-Express-based real-time film processing system.

In the future, we plan to test and explore other algorithms to further validate the proposed architectures.

ACKNOWLEDGMENT

This work was funded in part by the German Federal Ministry of Education and Research (BMBF).

REFERENCES

[1] Quantel, http://www.quantel.com/.
[2] Discreet, http://www.discreet.com/.
[3] FlexFilm, http://www.flexfilm.org/.
[4] A. do Carmo Lucas and R. Ernst, "An image processor for digital film," in *Proceedings of the 16th IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP '05)*, pp. 219–224, Samos, Greece, July 2005.
[5] A. do Carmo Lucas, S. Heithecker, P. Rüfer, et al., "A reconfigurable HW/SW platform for computation intensive high-resolution real-time digital film applications," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '06)*, vol. 1, pp. 194–199, Munich, Germany, March 2006.
[6] Xilinx, http://www.xilinx.com/.
[7] PCI-SIG, *PCI-Express Base Specification Revision 1.0*, July 2002.
[8] PCI-SIG, http://www.pcisig.com/.
[9] *IEC 60027-2: Letter symbols to be used in electrical technology—part 2: telecommunications and electronics*. IEC, 3.0 edition, August 2005.
[10] S. Dutta, R. Jensen, and A. Rieckmann, "Viper: a multiprocessor SOC for advanced set-top box and digital TV systems," *IEEE Design and Test of Computers*, vol. 18, no. 5, pp. 21–31, 2001.

[11] J. H. Ahn, W. J. Dally, B. Khailany, U. J. Kapasi, and A. Das, "Evaluating the imagine stream architecture," *ACM SIGARCH Computer Architecture News*, vol. 32, no. 2, p. 14, 2004.
[12] Hunt Engineering, http://www.hunteng.co.uk/.
[13] Nallatech, http://www.nallatech.com/.
[14] SGI, http://www.sgi.com/products/rasc/.
[15] J. Park and P. C. Diniz, "Synthesis of pipelined memory access controllers for streamed data applications on FPGA-based computing engines," in *Proceedings of the 14th International Symposium on System Synthesis (ISSS '01)*, pp. 221–226, Montreal, Quebec, Canada, September-October 2001.
[16] Oxford Micro Devices, http://www.omdi.com/.
[17] Texas Instruments, http://www.ti.com/.
[18] Analog Devices, http://www.analog.com/.
[19] R. Strzodka and C. Garbe, "Real-time motion estimation and visualization on graphics cards," in *Proceedings of the 15th IEEE Visualization Conference (VIS '04)*, pp. 545–552, Austin, Tex, USA, October 2004.
[20] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy, "Introduction to the cell multiprocessor," *IBM Journal of Research and Development*, vol. 49, no. 4-5, pp. 589–604, 2005.
[21] P. R. Panda, F. Catthoor, N. D. Dutt, et al., "Data and memory optimization techniques for embedded systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 6, no. 2, pp. 149–206, 2001.
[22] V. Cuppu, B. Jacob, B. Davis, and T. Mudge, "High-performance DRAMs in workstation environments," *IEEE Transactions on Computers*, vol. 50, no. 11, pp. 1133–1153, 2001.
[23] V. Cuppu, B. Jacob, B. Davis, and T. Mudge, "A performance comparison of contemporary DRAM architectures," in *Proceedings of the 26th International Symposium on Computer Architecture (ISCA '99)*, pp. 222–233, Atlanta, Ga, USA, May 1999.
[24] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," in *Proceedings of the 27th International Symposium on Computer Architecture (ISCA '00)*, pp. 128–138, Vancouver, BC, Canada, June 2000.
[25] S. Heithecker, A. do Carmo Lucas, and R. Ernst, "A mixed QoS SDRAM controller for FPGA-based high-end image processing," in *Proceedings of the IEEE Workshop on Signal Processing Systems (SIPS '03)*, pp. 322–327, Seoul, Korea, August 2003.
[26] S. Heithecker and R. Ernst, "Traffic shaping for an FPGA-based SDRAM controller with complex QoS requirements," in *Proceedings of the 42nd Design Automation Conference (DAC '05)*, pp. 575–578, Anaheim, Calif, USA, 2005.
[27] M. Weber, "Arbiters: design ideas and coding styles," *In Synopsys Users Group (SNUG), Boston, Mass, USA*, 2001. http://www.snug-universal.org/cgi-bin/search/search.cgi?Boston,+2001.
[28] K.-B. Lee, T.-C. Lin, and C.-W. Jen, "An efficient quality-aware memory controller for multimedia platform SOC," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 5, pp. 620–633, 2005.
[29] Sonics, Sonics MemMax 2.0 Multi-threaded DRAM Access Scheduler, Data sheet, Sonics, 2005. http://www.sonicsinc.com/.
[30] W.-D. Weber, "Efficient Shared DRAM Subsystems for SOCs," In *Microprocessor Forum*, 2001.
[31] C. Macián, S. Dharmapurikar, and J. Lockwood, "Beyond performance: secure and fair memory management for multiple systems on a chip," in *Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT '03)*, pp. 348–351, Tokyo, Japan, December 2003.
[32] K. Goossens, O. P. Gangwal, J. Röver, and A. P. Niranjan, "Interconnect and memory organization in SOCs for advanced set-top boxes and TV," in *Interconnect-Centric Design for Advanced SOC and NOC*, chapter 16, Springer, New York, NY, USA, 2004.
[33] F. Harmsze, A. Timmer, and J. van Meerbergen, "Memory arbitration and cache management in stream-based systems," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '00)*, pp. 257–262, Paris, France, March 2000.
[34] ARM. *PrimeCell Dynamic Memory Controller (PL340)*. ARM, 2005.
[35] Xilinx, Xilinx Memory Solutions, http://www.xilinx.com/products/design_resources/mem_corner/index.htm.
[36] K. Henriss, P. Rüffer, and R. Ernst, "A reconfigurable hardware platform for digital real-time signal processing in television studios," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '00)*, p. 285, Napa Valley, Calif, USA, April 2000.
[37] S. Eichner, G. Scheller, U. Wessely, H. Rückert, and R. Hedtke, "Motion compensated spatial-temporal reduction of film grain noise in the wavelet domain," in *Proceedings of the Society of Motion Picture and Television Engineers Technical Conference (SMPTE '05)*, New York, NY, USA, November 2005.
[38] C. Sanz, M. J. Garrido, and J. M. Meneses, "VLSI architecture for motion estimation using the block-matching algorithm," in *Proceedings of the European conference on Design and Test (EDTC '96)*, pp. 310–314, Paris, France, March 1996.
[39] S. Rout, "Orthogonal vs. biorthogonal wavelets for image compression," M.S. thesis, Virginia Polytechnic Institute and State University, Blacksburg, Va, USA, 2003.
[40] N. D. Zervas, G. P. Anagnostopoulos, V. Spiliotopoulos, Y. Andreopoulos, and C. E. Goutis, "Evaluation of design alternatives for the 2-D-discrete wavelet transform," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 12, pp. 1246–1262, 2001.
[41] C. M. Brislawn, "Classification of nonexpansive symmetric extension transforms for multirate filter banks," *Applied and Computational Harmonic Analysis*, vol. 3, no. 4, pp. 337–357, 1996.
[42] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: a free, commercially representative embedded benchmark suite," in *Proceedings of the 4th Annual IEEE International Workshop on Workload Characterization (WWC-4'01)*, pp. 3–14, Austin, Tex, USA, December 2001.