Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2009, Article ID 851613, 15 pages
doi:10.1155/2009/851613

*Research Article*

# Providing Memory Management Abstraction for Self-Reconfigurable Video Processing Platforms

**Kurt Franz Ackermann,[1,2] Burghard Hoffmann,[2] Leandro Soares Indrusiak,[1] and Manfred Glesner[1]**

[1] *Institute for Microelectronic Systems, Darmstadt University of Technology, Karlstrasse 15, 64283 Darmstadt, Germany*
[2] *VITRONIC Dr.-Ing. Stein Bildverarbeitungssysteme GmbH, Hasengartenstr. 14, 65189 Wiesbaden, Germany*

Correspondence should be addressed to Kurt Franz Ackermann, kurt.ackermann@vitronic.com

Received 30 December 2008; Revised 24 April 2009; Accepted 15 June 2009

Recommended by Peter Zipf

This paper presents a concept for an SDRAM controller targeting video processing platforms with dynamically reconfigurable processing units (RPUs). A priority-arbitration algorithm provides the required QoS and supports high bit-rate data streaming of multiple clients. Conforming to common video data structures the controller organizes the memory in partitions, frames, lines, and pixels. The raised level of abstraction drastically reduces the complexity of clients' addressing logic. Its uniform interface structure facilitates instantiations in systems with various clients. In addition to SDRAM controllers for regular applications, special demands of reconfigurable platforms have to be satisfied. The aim of this work is to minimize the number of required bus macros leading to relaxed place and route constraints and reducing the number of critical design paths. A suitable interface protocol is presented, and fundamental implementation issues are outlined.



## 1. Introduction

Industrial video processing applications typically require high data-rates to obtain detailed information on target scenes. Since real-time demands of complex algorithms often cannot be satisfied by software solutions, suitable hardware implementations are required [1]. Adequate processing rates and a high degree of flexibility make today's FPGAs a preferable technology for video-processing implementations. Nevertheless, limited resources of FPGAs are bottlenecks for many complex algorithms. Modern FPGAs support the feature of partial dynamic reconfiguration enabling the change of FPGA sections during run-time while the remainder of the device continues to operate [2, 3]. Within the research work described in this article, this technology is applied to a video processing platform equipped with a single FPGA [4, 5]. If a complex algorithm can be decomposed into a set of smaller sequentially executed processing units, time-multiplexing might significantly reduce demands on FPGA resources. A reconfiguration order is determined according to an initially transferred scheduling plan by the FPGA itself. The ability of mapping tasks to reconfigurable processing units (RPUs) during run-time makes the platform suitable for a wide range of algorithms.

Depending on application demands, single lines or even multiple frames have to be buffered by the hardware in order to facilitate the desired functionality. In this regard, systems deploying high-resolution cameras, with frame dimensions of three mega pixels and more, further increase memory requirements. Today's largest FPGA devices contain approximately 3 MB on-chip memory [6, 7], allocated to some extent by a design's configuration data already. Thus, insufficient memory is available for buffering video data and intermediate processing results. Owing to this lack of memory, a suitable platform is equipped with additional fast external DDR SDRAM, connected to the FPGA. This article presents the concept of an application domain specific SDRAM controller providing an appropriate QoS.

The design of RAM controllers in multimedia applications has been widely discussed already [8–11]. Instead of targeting the physical access to the RAM, the presented work moreover focuses on providing services on higher

abstraction layers. A motivation for a generalized addressing scheme is not at least due to increasing the compatibility between various cores, producing differently arranged output data. Video processing applications typically aim to address data in units of frames, lines, and pixels, defining the desired level of abstraction. Furthermore, multiple clients aim to store results concurrently in a shared RAM and request data from different locations as well. Thus, a suitable memory management is indispensible and too complex to be handled by the processing cores themselves. This work encompasses a centric solution inside the memory controller, with support for multiple partitions and scalable frame buffers.

Memory controller designs with dynamically reconfigurable clients have further demands also not considered in research projects so far. Although dynamic reconfiguration does not inherently increase the resulting traffic within applications, problems arise due to a more complex place and route stage as part of the implementation process [5, 12]. Especially signals crossing reconfigurable boundaries potentially induce critical paths. Due to a majority of affected port signals in designs deploying a memory controller, the article discusses important related aspects and provides an adequate concept of a communication interface and protocol.

In the remainder of this article, Section 2 introduces the underlying self-reconfigurable frame grabber platform. Its functional components and essential data paths are outlined. Section 2 further defines the required QoS and provides scaling methodologies. The concept of the proposed memory controller is presented in Section 3. Demands and difficulties in dynamically reconfigurable designs are explained, and a suitable interface protocol is introduced. Section 3 details the implementation of the controller's modular structure. Focal points are priority arbitration, instruction decoding, and memory organization. An abstract case study in Section 4 provides results of real time analyses. Furthermore, the operating sequence of a sample client is demonstrated. Finally, Section 5 summarizes this work.

## 2. Reconfigurable Video Processing Platform

The aim of this section is to gain an elementary understanding of a dynamically reconfigurable framework required to deploy the proposed memory controller. A corresponding video processing hardware platform is realizable either as a smart camera or as a frame grabber, basically differing in the data source only. The latter will be exemplified in the following due to its wider range of applicability. Essential components of frame grabbers are interfaces to cameras and PCs, data processing cores, and fast on-board memories. Aiming to obtain a multifunctional and at the same time flexible platform, processing cores are typically implemented in FPGAs [1, 13].

Additional to conventional frame grabbers the proposed concept relies on partial dynamic reconfiguration. In spite of the implementation overhead, this technology offers significant advantages regarding resource utilization. Video processing systems typically contain a number of reusable components arranged in a pipeline. Each element has to be implemented and synthesized regarding the underlying hardware platform. Thus, a desired functionality is achievable by sequentially configuring the generated partial bitstreams into reconfigurable processing units (RPUs) during run-time. Time-multiplexing of IP-cores generally enables the designer to extend the depths of applications realizable on a certain device. Nevertheless, such a reconfiguration principle also incurs some considerable drawbacks not at least motivating for the improvements introduced in Section 3. The first to mention is the increase of the system's overall latency, which is directly correlated to occurring reconfiguration times. In order to compensate the latency overhead, a frame grabber based on a minimum of two RPU slots, operating mutually exclusive, inherently incorporates an adequate solution [5, 14]. An according reconfiguration scheduling for a sequence of $n$ jobs is presented in Figure 1. While one RPU is processing (R), the other one is configured (C) and parameterized (P), and vice versa. Therefore, the scheduling order must avoid reconfiguration times exceeding concurrent processing periods, making it imperative that the reconfiguration process allocates a minimum time slot. Thus, the writing of partial bitstreams to the FPGA's configuration memory needs to be controlled by the FPGA itself, referred to as self-reconfiguration.

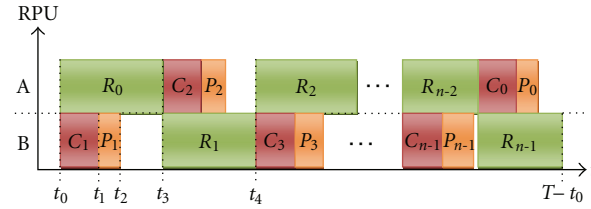


Figure 1: RPU job-scheduling.

*2.1. Modular Structure and Data Flow.* Essential components of the proposed framework are combined in Figure 2. Except for an external DDR-RAM and SRAM, all components are embedded inside a single FPGA. The paths of image data, parameters, and partial bitstreams are outlined subsequently aiming to explain the principle of operation and moreover to elaborate required features of the aspired memory controller.

A Camera-Link (C-Link) interface establishes a connection between the frame grabber and a camera. In order to support various configurations according to the C-Link standard definition [15], the interface is dynamically reconfigurable, too. All data IO passes through the proposed memory controller implementing the design's data communication center. Received video data is buffered in an assigned RAM section and made available to all clients. The platform contains $n$ RPUs. Each of them may access the memory controller with requests to read buffered data and to write back intermediate results. A Gigabit-Ethernet module realizes the communication interface to a connected PC. After the final pipelined application stage has been configured in an RPU and finished processing, the

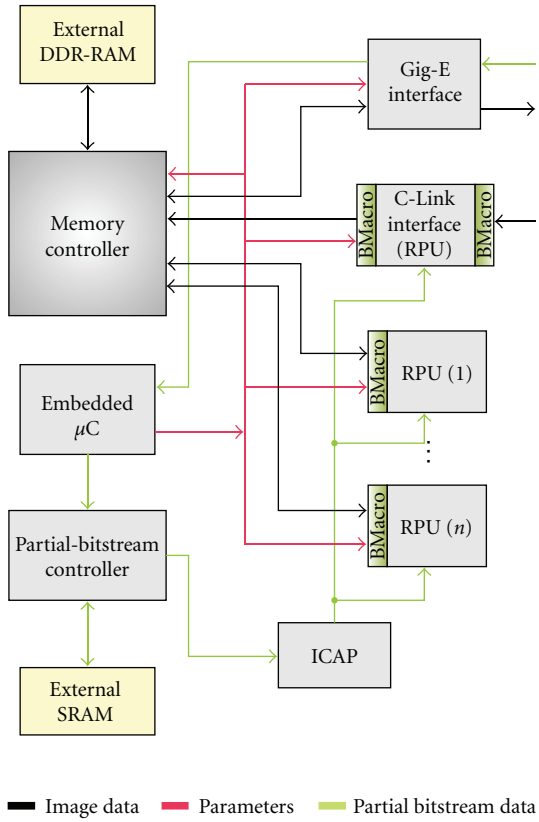


Figure 2: Frame grabber structure.

memory controller transfers the results to the PC. Finally, the processing cycle starts anew as the next image is received from the camera.

Not least because most algorithms are sensitive to changes in the environment—parameterization is indispensable for video processing applications. The capability to transfer information regarding the configuration, the application, and the environment to dedicated design-modules during run-time significantly increases flexibility. In this manner, an initial set of parameters is determined at system start-up time and transferred together with a core scheduling plan to a universal embedded microcontroller ($\mu$C), realizing the sequence in Figure 1. A suitable resource efficient parameterization interface enables a proper distribution of parameters from the $\mu$C to all frame grabber modules [5]. Furthermore, partial bitstreams, resulting from RPU-specific algorithm implementations, are initially transferred from the PC to the platform's SRAM. Eligible FPGAs support self-reconfiguration by providing an Internal Configuration Access Port (ICAP), implementing an interface between the user-logic and the configuration memory. During operation, the $\mu$C periodically triggers reconfigurations by initiating a partial bitstream transfer from the SRAM to the ICAP interface.

*2.2. Quality of Service (QoS) Requirements.* In industrial applications, all system components have to ensure a defined quality of their services to satisfy global real-time specifications. In case of the presented framework, the desired QoS can be expressed as the maximum frame-rate and resolution that can be processed without missing any input-data after a certain latency period.

Video data is continuous media, producing periodic processing loads. As a matter of principle, periodic demands are easier to comply with than sporadic ones [16]. Since the presented design implies a deterministic system with predictable bandwidth demands, a certain minimum QoS can be guaranteed to the user at system-startup time. The QoS is mainly negotiated between the PC, running the design automation flow [4] and the FPGA-embedded microcontroller on the hardware platform. The PC must be knowledgeable of the peak data-rate, transmitted by the camera. Major determining factors are frame size, line size, line gap, frame gap, and pixel frequency. The design automation software obtains detailed information about the underlying hardware structure by a hardware capabilities report of the microcontroller. Consequently, a reconfiguration scheduling plan is created regarding the data I/O demands of particular stages within the processing pipeline, and priorities are assigned to all RAM clients. Finally, communication requirements of the platform's Gigabit-Ethernet interface, such as packet-size, packet-rate, bandwidth, and latency, are limiting the overall QoS.

The heterogeneity of application demands requires the services of system components to be parameterizable. Thus, if a desired QoS is not achievable, the platform informs the PC about the location of the bottleneck. The QoS could be scaled down, if bottlenecks cannot be resolved by adaptations of the scheduling cycle or a redistribution of priorities. Therefore, entire input frames could be periodically skipped (temporal scaling) or the frame-size reduced (spatial scaling). The application specific definition of an adequate scaling methodology always depends on the location of the bottleneck. Within the presented framework, scaling is initiated by the PC and can be activated in the platform by a simple parameterization of its components. A general and more detailed discussion on achieving an appropriate QoS in scalable video applications is given in [17].

## 3. SDRAM Controller Concept

After the operational environment of the proposed memory controller has been outlined in previous sections, subsequent the key idea underlying this article is brought into focus. Not all video processing cores read their input data linearly, as received from a camera. Especially complex algorithms may require random access on data of multiple frames. The proposed SDRAM controller introduces an additional layer of abstraction, providing dedicated services to video processing applications. Thus, an abstracted addressing scheme increases the compatibility between diverse cores in the processing pipeline. Furthermore, the complexity of RAM clients can be drastically reduced, as they need not to consider the underlying memory organization. The fundamental architecture of the proposed memory controller

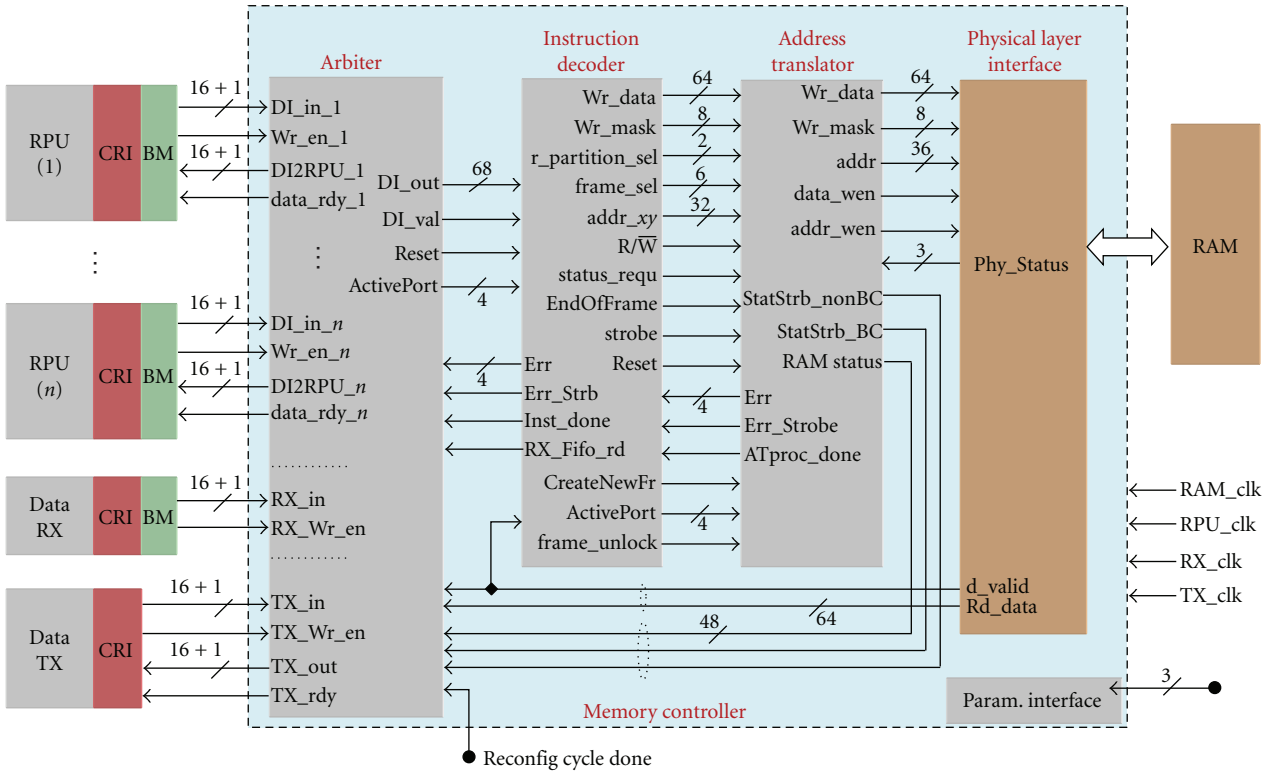


Figure 3: Architecture of the memory controller.

is presented in Figure 3. Four hierarchical modules and a parameterization interface provide the aspired functionality:

(i) support for multiple clients,

(ii) priority arbitration,

(iii) support for R/W data bursts,

(iv) memory partitioning,

(v) frame-based ring-buffers,

(vi) support for variable frame dimensions,

(vii) support for high-level addressing (units: partitions, frames, lines, pixels),

(viii) providing high-level status information.

Unlike other modules, there are no special demands on the physical layer interface, which is responsible for generating RAM clock signals, initializing the memory, transmitting data, and applying required refresh signals. Common memory interfaces, such as the Xilinx MIG [18], provide appropriate low-level services and are not subject of this article.

Although the projected memory abstraction is an important and far from trivial task, the major driven force for this work is to reduce the number of critical design paths emerging from constraints in the partial reconfiguration design flow [19]. The following subsection specifies these constraints and discusses in particular the impact on dynamically reconfigurable designs connecting an external RAM. In this regard, the problem is approached by a novel interface concept determining the further organization of the proposed memory controller.

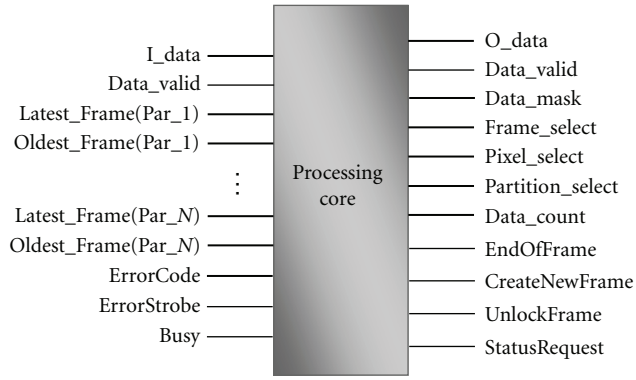


Figure 4: Client ports to memory controller (ver. 1).

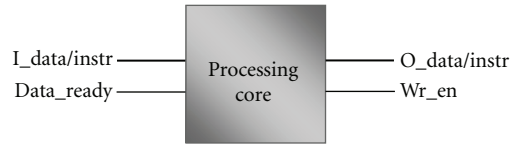


Figure 5: Client ports to memory controller (ver. 2).

*3.1. Memory Controller Interface.* Figure 4 exemplifies a client's I/O interface to a memory controller implementing the desired functionality. The client input signals (left) contain besides a data-vector moreover partition information

and reported error codes. Respective output signals (right) contain a write data-vector, corresponding mask, and high-level addressing signals. Depending on the memory size, the data widths and consequently the number of required port I/Os may vary.

The presented concept particularly targets designs with dynamically reconfigurable RAM clients. In order to facilitate proper run-time reconfigurations, port-signals crossing reconfigurable boundaries need to be routed through bus-macros (BM) [2, 5, 12, 19]. At this, the designer has to ensure that all BMs are placed directly on dedicated boundaries of corresponding regions. BMs are either implementable based on slice macros or on 3-state busses [12]. Note that, modern FPGAs, like the Xilinx Virtex-4, do not support internal 3-state busses anymore. Furthermore, slice-based macros are more efficient and easier to place. However, bus macros inherently impart routing restrictions affecting the signal timing. The client-RAM interface (CRI), as presented in Figure 4, occupies approximately 252 I/Os requiring 32 bus-macros for every reconfigurable client in a Virtex-4 implementation. The potential generation of critical paths as well as the time-consuming manual placement of the bus-macros motivates for a more efficient solution.

Figure 5 presents an optimized version of the memory controller's client interface with a drastically reduced set of IOs. Instead of using multiple channels to transmit addresses, data, and status information, only a single 17-bit wide bus and a corresponding control signal are required for each communication direction.

This significant improvement is achievable by sharing a bus for data and so called instruction telegrams. Instruction telegrams are part of a simple communication protocol and function as a container for remaining signals. They are decoded by the respective receivers and encompass the equivalent functionality as the complex unit in Figure 4. The most significant bit on the bus serves as a tag, distinguishing the two telegram types. Inside the memory controller, all data-/ instruction telegrams are buffered in asynchronous FIFOs, enabling concurrent access of multiple clients. Clients deploy the *Wr_en* signal to store data in respective FIFOs associated to their connection ports. Vice versa, the memory controller asserts the *Data_ready* signal as soon as data is available for readout. A thusly triggered client transfers an instruction telegram, signalizing its ready-state. Consequently, the memory controller initiates the readout of the corresponding FIFO and transfers available data.

Contents and sizes of instruction telegrams are different for either direction. Table 1 lists elements of instructions sent from clients to the memory controller. The "Size" column represents the required bits per instruction field.

Asynchronous FIFOs serving as data buffers between the memory controller and its clients do not necessarily have the same bus-widths and clock rates on both of their ports. Therefore, it is feasible to reduce the bus-widths of the interface (Figure 5) beyond the physical data width of the RAM. Instantiating the underlying architecture of Figure 3, clients read and write two pixels per clock cycle, resulting in a word width of 16 bits plus one tag bit. The controller obtains data in 8-byte words from the physical layer interface, defining its internal data bus width to 64 bits. Therefore, the memory controller inserts $k = 64/16$ tags into each data word written to the clients' read-FIFOs. The resulting size of instruction telegrams—for both directions—is 68 bits. Thus, clients require four clock cycles per instruction word while the memory controller needs one cycle only.

BlockRAMs of Xilinx FPGAs offer one parity bit per memory-byte, enabling proposed FIFO implementations without allocation of additional resources for telegram type tags. Contents of instruction words transmitted by the memory controller are primarily status information and listed in Table 2.

Entries of both tables will be self-explanatory as their processing logic is described in detail within the following subsections.

The proposed concept provides the desired functionality and minimizes the number of required bus macros. Nevertheless, the incurred protocol overhead increases the effective memory access time and represents a considerable drawback. Hence, particularly clients frequently addressing noncontinuous small data units are affected. However, due to the majority of processing cores preferring burst access to the memory, the protocol related latency, which does not exceed a few clock cycles, is negligible in most cases.

*3.2. Client RAM Interface.* In order to increase the concept's applicability, the RAM controller provides the interface on Figure 5 to all of its clients in the same manner. However, the reduced interface type leads to significant protocol-handling overhead inside clients. To counter negative effects, corresponding design parts are encapsulated and separated from clients' processing logic, according to Figure 3. Such client RAM interfaces simply connect to clients providing the original IO-set (Figure 4) and hide all protocol related operations.

The interface design is based on an RX-and TX-module, processing in parallel. The latter is presented in Figure 6 and is responsible for forwarding data to the memory controller as well as the encoding of instruction telegrams according to Table 1. As mentioned previously, in the presented design clients are writing to a FIFO inside the memory controller in words of 17 bits while the controller itself reads words of 68 bits. In this regard, the TX-module avoids telegram fragmentations by ensuring that always blocks matching the controller width are written into the memory controller. In the following, the architecture of the interface is explained in detail.

The TX-module contains two finite state machines (FSMs). FSM-1 handles clients' write requests and is furthermore triggered by unlock or create-new-frame (CNF) commands. The functions of commands are not relevant at this point and will be handled later in Section 3.5. Instruction telegrams are composed in states 1–4 and stored inside a FIFO. The FIFO is an essential component to buffer video data and write instructions during instruction word generations of FSM-2. A write request causes the video gate to open, enabling the client to transmit its data. FSM-1 remains in state 6 until the requested amount of data

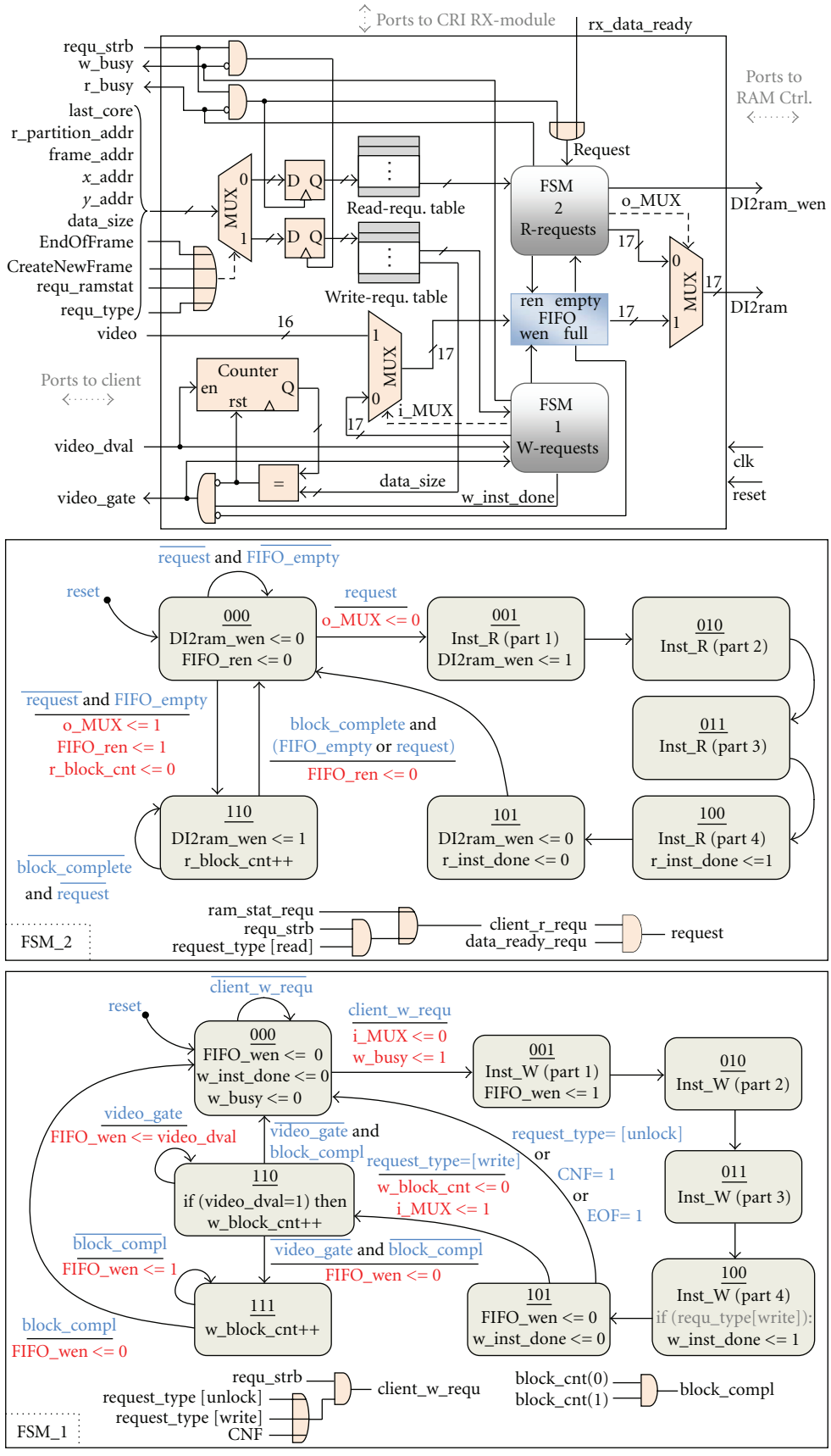


Figure 6: Client interface (TX-module).

has been buffered inside the FIFO, and the video gate is closed again. If the buffered data-size is not a multiple of the controller's input-buffer width, the data-block is complemented to avoid telegram fragmentations inside the memory controller. A busy signal informs the client when the interface is available for further write requests.

Although FSM-1 controls the execution of write requests, any kind of data is exclusively transferred to the memory controller by FSM-2. FSM-2 is triggered by a client's read or RAM-status request, and additionally when data is available for readout from the memory controller. In either case, FSM-2 composes an instruction and transfers it directly to the memory controller. Moreover, it is responsible to control the readout of the interface FIFO containing write instructions and corresponding data. In conformance with write requests, telegram fragmentations are avoided during FIFO readouts, respectively. The client RAM interface writes data to the memory controller in a deterministic order. Thus, video data is always preceded by a write instruction specifying the exact amount of data. Instruction telegrams with read or status requests are prioritized and may occur irregularly in the buffer, if supported by the memory controller.

The architecture of the RX-module of the client RAM interface is depicted in Figure 7. Complementary to the TX-module, it processes all data received from the memory controller. Receiving instructions and data is not obvious due to the lack of data valid signals. Hence, transmitted data blocks are encapsulated in two instruction words facilitating synchronization, according to Figure 8(a). Instruction telegrams are decoded by the FSM to obtain all information listed in Table 2. Received data blocks are forwarded to the client, and a corresponding valid signal is generated.

The memory controller transmits instruction telegrams replies either as on client requests or on frame-buffer content changes. In order to obtain solely valid status information, the RX-module exclusively extracts it from instruction telegrams with the "data valid" flag enabled. Thus, a pure status update requires an additional instruction telegram, respectively, disabling the update again. The corresponding protocol is illustrated in Figure 8(b).

*3.3. Priority Arbitration.* Concurrent requests of RAM clients make a suitable arbitration inside a memory controller indispensable. The achievable QoS, defined in Section 2.2, is strongly related to the arbitration algorithm. In order to satisfy various bandwidth requirements of clients the arbiter has to comply with priorities assigned by design automation software. The software, executed on the connected PC, is knowledgeable about the underlying hardware platform and defines a video processing scheduling plan according to user specifications [4]. Hence, bandwidth requirements and priority constellations can be determined in advance and are initially transferred to the frame grabber as parameters. Note that, due to the support of dynamically reconfigurable clients, priorities are not necessarily constant during a scheduling cycle. Thus, in order to avoid QoS limitations it is imperative that the arbiter provides priority updates during run-time, controlled by the embedded microcontroller.

Table 1: Instruction word: client → controller.

| Number | Size | Contents |
|---|---|---|
| 1 | 1 | Telegram type tag (data/instruction) |
| 2 | 1 | Request RAM status |
| 3 | 1 | Last core in processing pipeline |
| 4 | 1 | Start readout of Rd_FIFO |
| 5 | 2 | Memory access request type (read/write/unlock/NOP) |
| 6 | 2 | Partition select |
| 7 | 6 | Frame select |
| 8 | 1 | End-Of-Frame (EOF) |
| 9 | 1 | Create-New-Frame (CNF) |
| 10 | 16 | Column address (x) |
| 11 | 16 | Line address (y) |
| 12 | 16 | Number of bytes to read/write |

Table 2: Instruction word: controller → clients.

| Number | Size | Contents |
|---|---|---|
| 1 | $k*1$ | Telegram type tags (data/instruction) |
| | | Partition 1: |
| 2 | 6 | index of oldest frame |
| 3 | 6 | index of newest frame |
| | | Partition 2: |
| 4 | 6 | index of oldest frame |
| 5 | 6 | index of newest frame |
| | | Partition 3: |
| 6 | 6 | index of oldest frame |
| 7 | 6 | index of newest frame |
| 8 | 4 | Error code |
| | | Partition 4: |
| 9 | 6 | index of oldest frame |
| 10 | 6 | index of newest frame |
| 11 | 1 | Data valid |

Round Robin is a frame-based scheduling algorithm, serving flows in cycles [20]. Each client gets at least one opportunity per cycle to read or write data. The weighted round-robin (WRR) algorithm facilitates the distribution of available throughput corresponding to clients' priorities [21]. Thus, clients may transmit a fixed amount of data in provided time-slots, making WRR perform well in terms of fairness. To increase the arbitration efficiency the visits of clients have to be spread evenly in time, considered in advance by the design automation software.

A lightweight design of an arbiter implementing the desired functionality is illustrated in Figure 9. In the following, implementation details are outlined making the principle of operation clear. Client RAM interfaces connect directly to the arbiter implementing FIFOs to buffer data streams in both directions. Besides RPUs, two additional types of clients with special demands exist within this project. First, the RX client—a write-only client, continuously delivering incoming video data—does not utilize an RX-FIFO

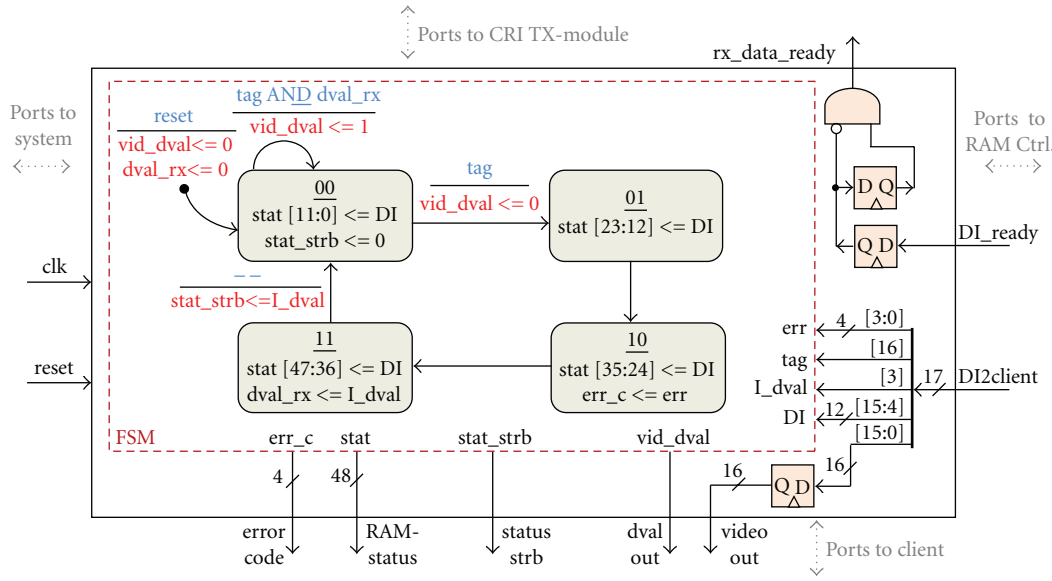


Figure 7: Client interface (RX-module).

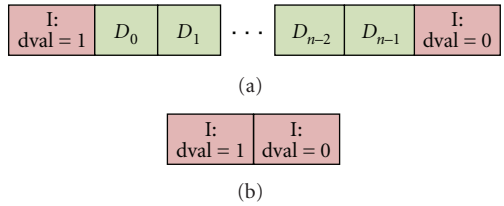


Figure 8: Transfer protocol.

on its port. In order to provide reliable frame acquisitions, this client always obtains the highest arbitration priority. Second is the TX client, a read-only client responsible to transmit the final processing results to a PC. Unlike the previous client, the TX-FIFO is not redundant here, as it is required to buffer read-out instruction telegrams. FIFO sizes determine the maximum allowed burst lengths of clients and are relevant for terms of real-time. Due to the line-oriented characteristic of a variety of image and video processing applications, appropriate buffering solutions are encompassed by choosing FIFO depths larger than the respective image line dimension.

The sequential characteristic of the arbitration process makes it natural to control it by a finite state machine. FSM-1 obtains the port-numbers from parameter registers according to the predetermined scheduling order. A timer is cyclically reloaded with the visit-time as the arbitration changes. If a client's TX-FIFO is empty, the arbiter immediately proceeds with the next scheduling element. Otherwise, an instruction plus optional subsequent data blocks (but no further instructions) is read out of the FIFO and passed to the instruction decoder. As soon as the instruction decoder finished processing, the sequence repeats until the visit-time elapsed. At this, a watchdog timer prevents the system from deadlocks if acknowledge signals exceed a certain time limit.

The second finite state machine receives status and error reports. It is responsible to encode instruction telegrams and to control all writing to clients' RX-FIFOs. Data received from the RAM is always internally buffered. FSM-2 controls the read-out and encapsulates containing data-blocks in two instruction telegrams, conforming to the protocol definitions. Furthermore, the address translator autonomously triggers broadcast status reports as contents in its frame-buffer table change. Corresponding instruction telegrams are created inside the arbiter and distributed to all clients.

The intended arbiter concept is characterized by a clear structure, devoid of redundancy. Though the achievable burst sizes depend on available FIFO capacities, the arbiter has no limitations regarding supported data dimensions. Thus, it facilitates a resource-efficient implementation and complies with the basic idea to support various video formats.

*3.4. Instruction Decoder.* The instruction decoder realizes the link between the arbiter and the address translator. Figure 10 presents an example implementation including a finite state machine depicted in Figure 11. It is responsible to decode client instructions and, furthermore, distinguishes supported request types for according instruction executions. Commands dedicated to the address translator—such as create-new-frame, unlock, end-of-frame, and status requests—are simply forwarded while read and write operations require adequate preprocessing. Thus, burst requests are unrolled, utilizing the implemented up-counter, in order to provide continuing high-level addresses to subsequent modules. Moreover, mask vectors are generated and output synchronously during write instructions.

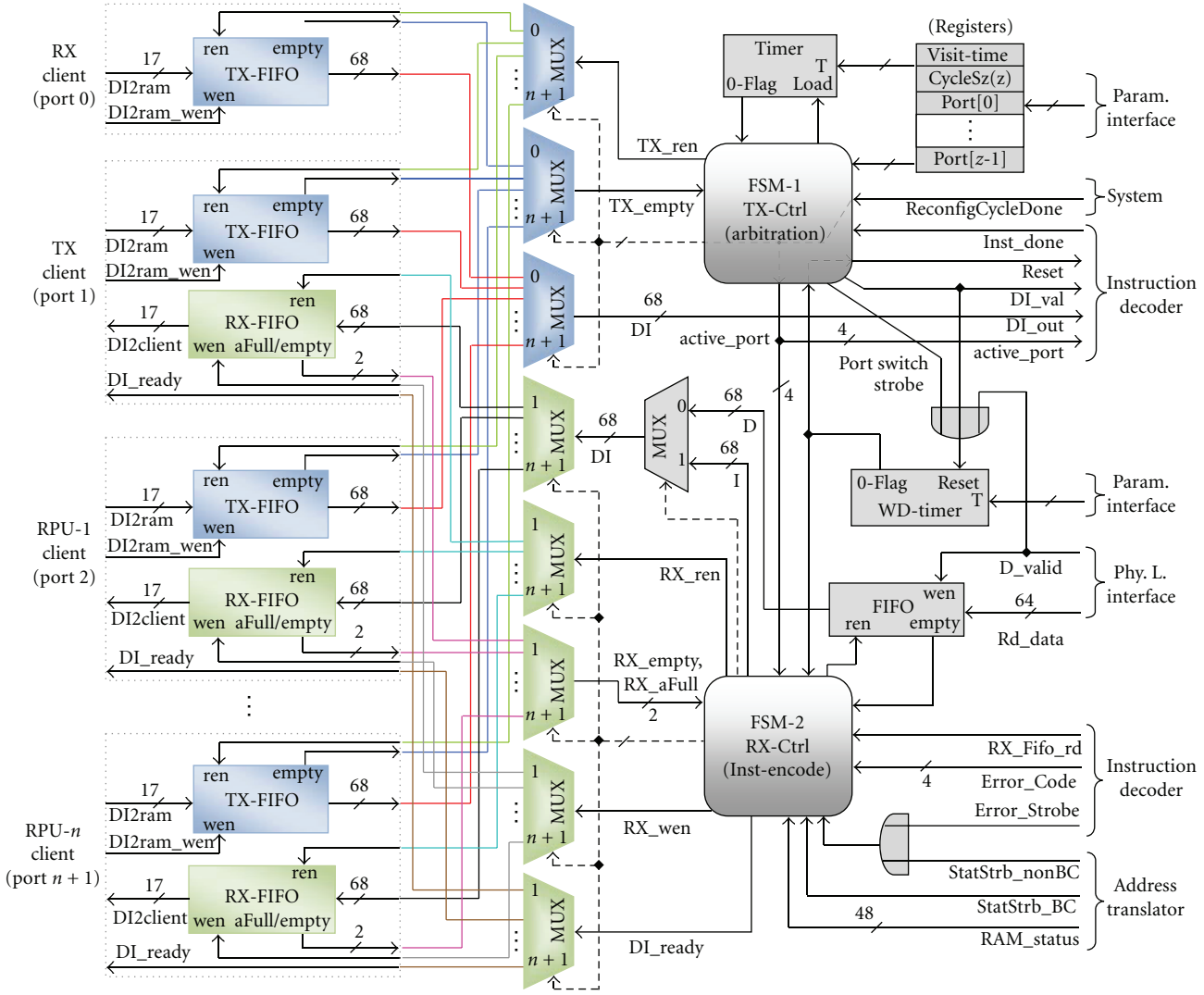


Figure 9: Arbiter architecture.

The instruction decoder features appropriate validation methodologies and provides corresponding error codes to the arbiter. In regard to write instructions a continuous stream of data, matching the requested size, is expected to be contained in the FIFO. The validation process of read instructions is more complex due to the fact that it relies on data consequently transmitted from the physical layer interface to the arbiter. Serving this purpose, an implemented down-counter is triggered by the received data valid signal and facilitates tracking of incoming data concurrent to address generation. The watchdog timer inside the arbiter implies to eliminate the eventuality of deadlocks within this state.

The proposed concept of the instruction decoder relies on a single finite state machine at the expense of parallelism in execution of RW-operations and forwarding of address translator commands. Respective improvements would impart further performance benefits for clients as demonstrated in Section 4.1.

*3.5. Address Translator.* The address translator undertakes the task to manage the physical memory in structures suitable for video processing applications. It maps high-level address descriptions in real-time to physical RAM addresses required by the physical layer interface, and it is responsible to provide services on higher abstraction layers to clients.

Video processing clients typically address data in units of frames, lines, and pixels. Due to multiple clients sharing a single physical RAM, support for partitions facilitates data integrity during write operations. Organizing partitions as frame ring-buffers provides access to the latest set of frames produced by particular clients and resolves the direct correlation between the totally required and physically available memory size. Owing to potentially nonuniform arranged output-data, especially in systems with dynamically reconfigurable clients, the address translator furthermore requires support for variable frame dimensions within these ring-buffers. In order to replicate a corresponding

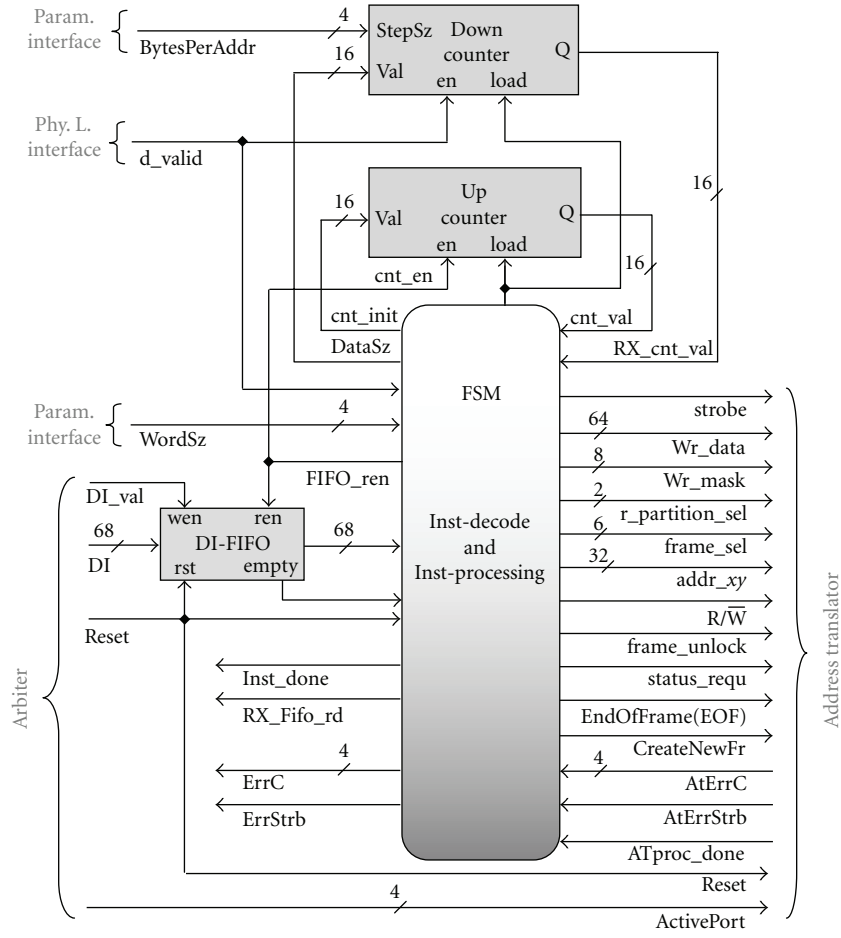


Figure 10: Instruction Decoder.

architecture in hardware, adequate status information has to be provided and managed for partition ring-buffers and containing frames. Status reports according to Table 2 make the availability of data transparent to clients. The transmission to the arbiter is triggered by either a request or autonomously when changes in partition tables occur.

The concept of the address translator is presented in Figure 12. Implementation details are outlined subsequently in order to establish better understanding of important aspects. The presented design supports up to four partitions with a maximum of 64 frames, respectively. BlockRAM memory inside the FPGA holds relevant status information for all frames:

(i) number of columns (16 bit),

(ii) number of lines (16 bit),

(iii) start address (16 bit),

(iv) end address (16 bit),

(v) empty flag (1 bit),

(vi) valid flag (1 bit),

(vii) closed flag (1 bit).

Start and end address entries define the partition internal location of frames, and various flags represent their current state. Create-new-frame (CNF) commands of clients allocate and determine new virtual frames for write access. To avoid conflicts caused by write requests of different clients to overlapping locations, partitions are not shared for write operations. Therefore, the embedded microcontroller initially assigns partitions exclusively to writing clients. This configuration is realized by the parameterization interface setting up the write-partition-allocation LUT inside the address translator. Read requests, in turn, are generally permitted to all partitions. The closed flag is assigned by an end-of-frame (EOF) command, indicating that a client finished writing to a frame. Consequently, broadcast status reports are triggered making the corresponding frame available to other clients. Nevertheless, the address translator supports unlock commands enabling further changes in previously closed frames.

A finite state machine serves the processing of time-noncritical tasks, such as client command executions and error code generation. Furthermore, it continuously updates frame and partition status entries. Latter are underlying status reports encoded by the arbiter. Partition information is implemented in registers, containing the following:

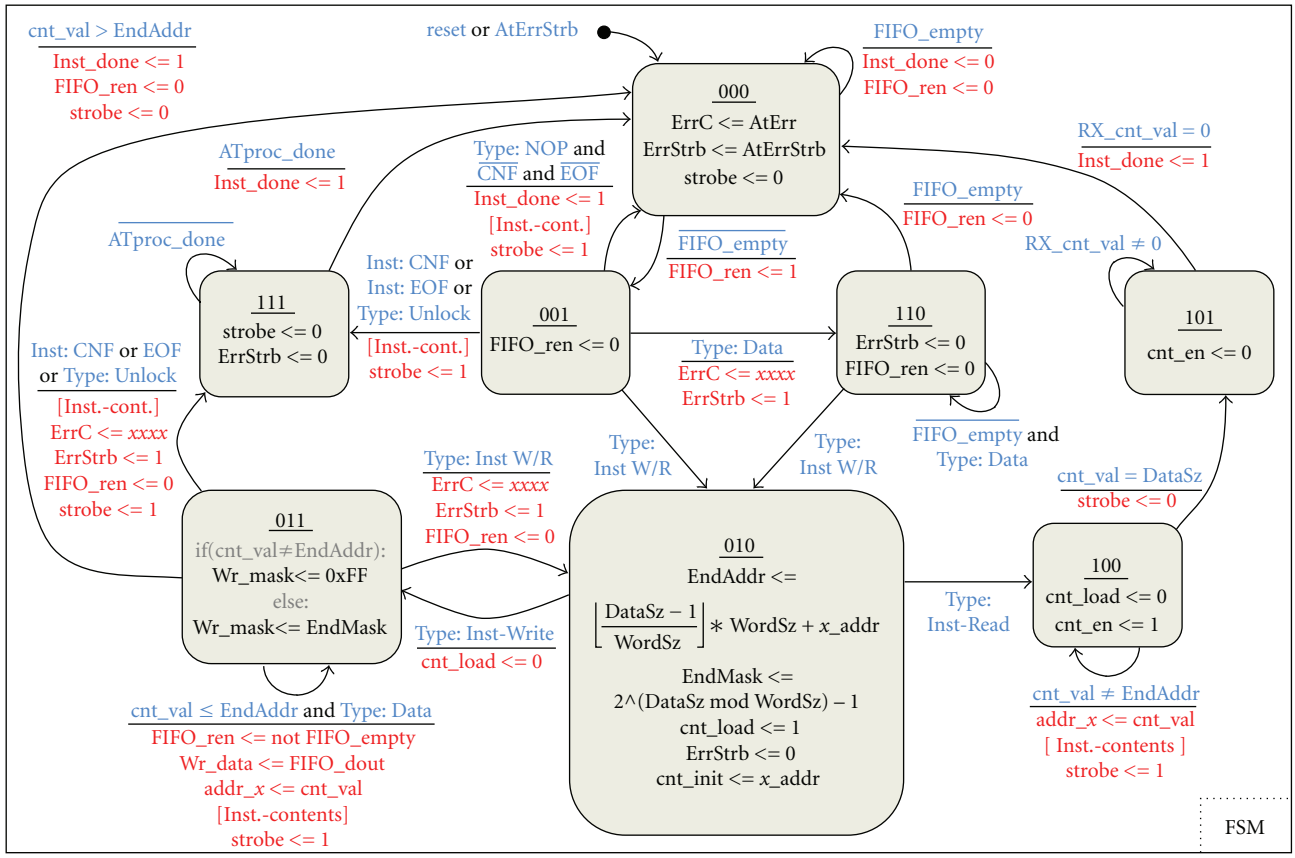


Figure 11: Instruction Decoder-FSM.

(i) size (bytes) (33 bit),

(ii) physical address (36 bit),

(iii) first (oldest) frame number (6 bit),

(iv) last (newest) frame number (6 bit),

(v) full flag (1 bit).

The full flag indicates that no free memory is available in a partition. However, since partitions are organized as ring-buffers, this does not imply that further write requests are prohibited but rather are old frames overwritten. Finally, the size and physical address entries are static parameters, dependent on the memory size and the number of connected clients.

The green highlighted function blocks in Figure 12 represent the calculation of the physical RAM addresses. Note that a pipelined architecture has to be implemented facilitating on-the-fly processing of incoming data from the instruction decoder. Finally yet importantly, the address translator provides versatile correlated error detections, too.

(i) Read attempts on empty or invalid frames (1).

(ii) Address exceeds frame boundary (2, 3).

(iii) Addr_x exceeds number of columns (4).

(iv) Addr_y exceeds number of lines (5).

(v) Write attempts to closed frames (6).

Error codes are generally forwarded to affected clients, facilitating coarse run-time debugging of both the memory controller and its clients. As processing cores configured in an application-specific scheduling pipeline are interdependent, the proposed error detection moreover establishes a fundamental validation of the system setup. For instance, owing to an incompatibility within the configuration queue, a client may expect intermediate results of its predecessor differently arranged. A consequential address violation emerging inside the address translator results in a corresponding error report, exposing the problem to the design automation software.

## 4. Case Studies

*4.1. Flip-Image Sample Client.* The relevance and implementation of the aspired memory abstraction within the proposed concept has been elaborated in previous sections. To point out the accomplished benefits from a clients point of view, two different sequences of operation are exemplified in Figure 13. At this, the complexity of the regarded task is not of relevance, as solely data-IO events are focused. Hence, the presented client simply intends to vertically flip buffered images. The statechart on the left depicts a process with sequential read and write operations. The memory controller periodically transmits broadcast status-reports

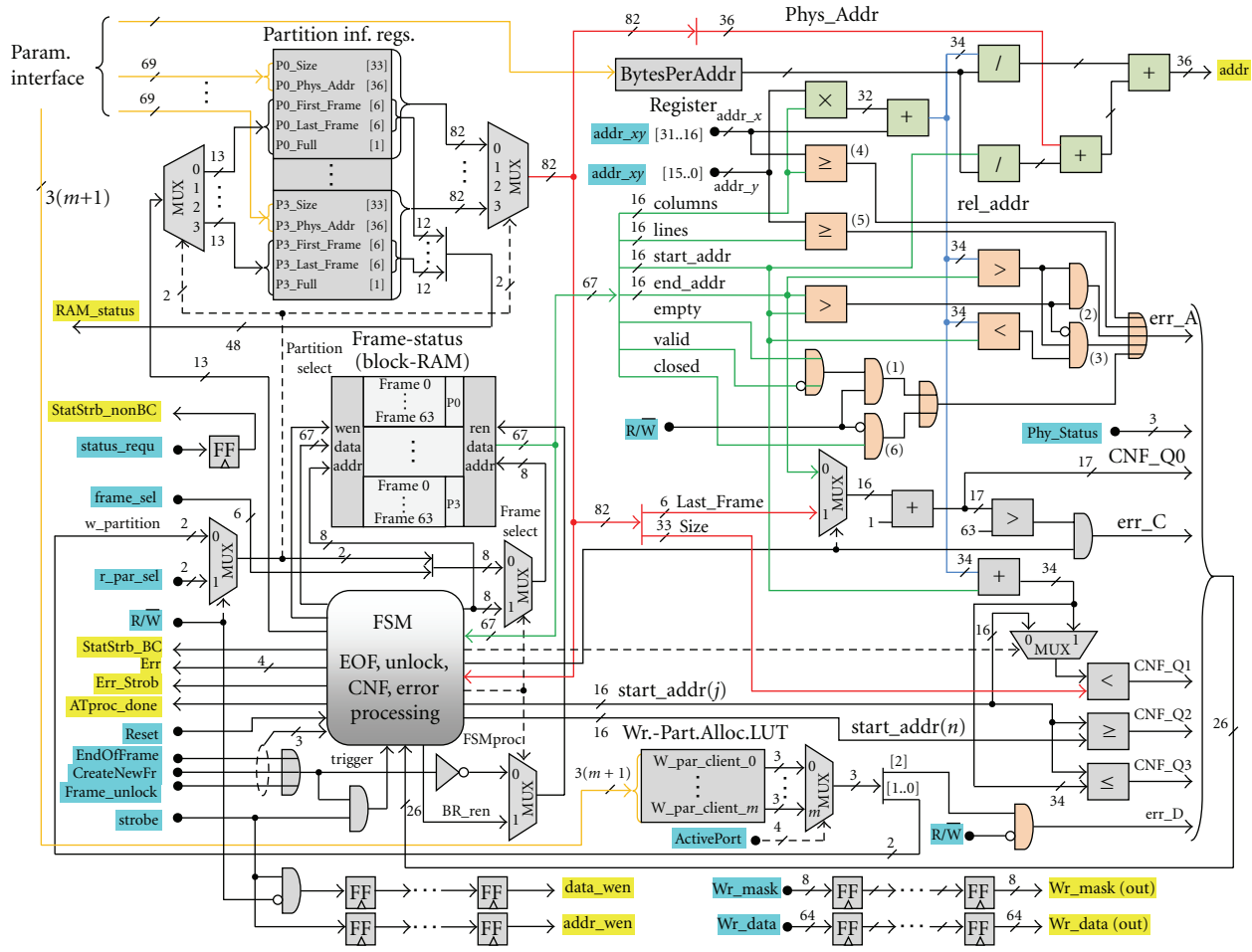


Figure 12: Address translator.

containing high-level addresses of buffered frames. Triggered by a new frame, the client initiates a CNF command to allocate RAM according the dimension of its output frame. Subsequently, the last line of the destination frame is requested and internally buffered as it is received. A write request, addressing the first line of the target frame, causes the client RAM interface to open the video gate, giving access to the corresponding arbiter FIFO. After all buffered data have been forwarded to the controller the procedure starts anew for the next line. Finally, a telegram containing an EOF command makes the created frame available to other clients and ends the process.

Yet, the example incurs the requirement for clients to buffer received data, causing increased processing latencies. In this regard, an optimized sequence, implementing parallel executions of read and write operations, is presented in the second statechart. This enhanced version intends to open the video gate before a read request is applied. Consequently, received data can be directly processed and forwarded without the need for additional buffers. Obviously the encompassed memory abstraction facilitates the desired uncomplex high-level access to a connected RAM and hides underlying memory organization tasks.

*4.2. Traffic Analysis.* Due to changing requirements of RPUs and varying input data-rates, general bandwidth demands regarding the proposed memory controller are unpredictable. Nevertheless, this section aims at establishing better understanding of relevant traffic aspects, finally instancing a certain case study. To begin with, general calculations of periodic transfer durations are provided for important intersections. As stated before, video data is typically subdivided into units of frames, lines, and pixels, including short gaps between frames and lines. Therefore, the given calculations refer to respective line transfers, representing decisive continuous data bursts. In order to provide numeric results, assumptions about the environment are made in the following.

Thus, assuming a camera producing 15 frames per second in a resolution of 2048 × 2048 pixels with an 8-bit color depth, the resulting incoming datarate is 60 MB/s. Within this case study, arbiter FIFOs are capable to store two image lines, and the scheduling is parameterized to serve one line per visit-cycle. Thus, incoming lines have cycle durations of

$$T_i = \frac{1}{f_{i\_frame} * \text{Lines}} = 32.55\,\mu s. \quad (1)$$

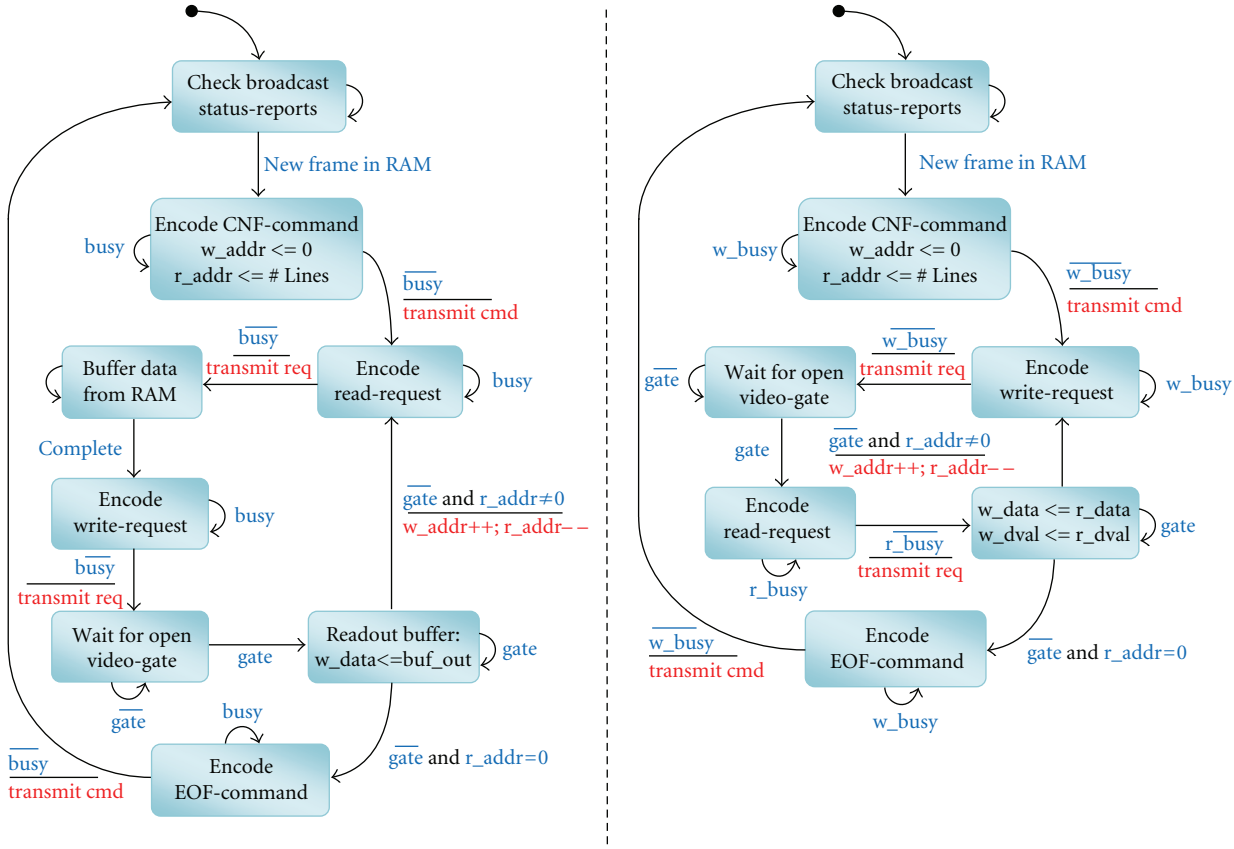


Figure 13: Client processing sequence: sequential (left), parallel (right).

The arbiter accesses included FIFOs in 8-byte wide words with a clock frequency of 100 MHz. Thus, the period for according line R/W operations results in

$$T_1 = \frac{1}{f_{\text{RAMclk}}} * \left( \frac{\text{Pixel/Line} * \text{Bits/Pixel}}{\text{BusWidth}} + 2 \right) < 2.6\,\mu\text{s}. \quad (2)$$

The bracketed expression in (2) represents the clock cycles required to transmit a single line plus the two data-block enclosing instruction words. The time after arbitration, required to perform transfers to or from the DDR RAM, calculates, respectively, as

$$T_2 = T_{\text{CtrlLat}} + \frac{1}{f_{\text{RAMclk}}} * \left( \frac{\text{Pixel/Line} * \text{Bits/Pixel}}{2 * \text{BusWidth}} + \text{RAM}_{\text{Lat}} \right). \quad (3)$$

In (3) it is assumed for simplicity that the desired data-blocks can be obtained in burst-mode without additional interrupting latencies. Furthermore, $\text{RAM}_{\text{Lat}}$ is less than 27 clock cycles, reflecting the memory access latency preceding bursts. Any overhead-time required by the memory controller to decode and execute instructions is considered in $T_{\text{CtrlLat}}$. Thus, with a low-level DDR RAM bus-width of 4 bytes $T_2$ results in less than 2.9 microseconds Clients, in turn, are connected via a 2-byte wide bus operating at 100 MHz. Thus, data transfer in the magnitude of a single line requires

$$T_3 = \frac{1}{f_{\text{RPUclk}}} * \frac{\text{Pixel/Line} * \text{Bits/Pixel}}{\text{BusWidth}} = 10.24\,\mu\text{s}. \quad (4)$$

After essential transfer times have been determined, Figure 14 illustrates a traffic analysis for an according case study deploying two clients. A dynamically reconfigurable system with multiple mutually exclusive RPUs leads to decreased bandwidth demands. This is due to impossible operational pipelining among clients, preventing concurrent requests to the memory. Therefore, the presented case study does not imply dynamic reconfiguration, addressing more challenging traffic aspects. The time-slots visualized in Figure 14 conform to (1) to (4). Corresponding labels contain information about accessed frame and line-numbers, with $n$, $m$, and $z$ representing frames of different partitions, respectively. Thus the first client processes lines of the three latest received frames, generating single output lines. These intermediate results are requested by the second client, storing single lines in its associated writing partition. The second client furthermore implements the final core in

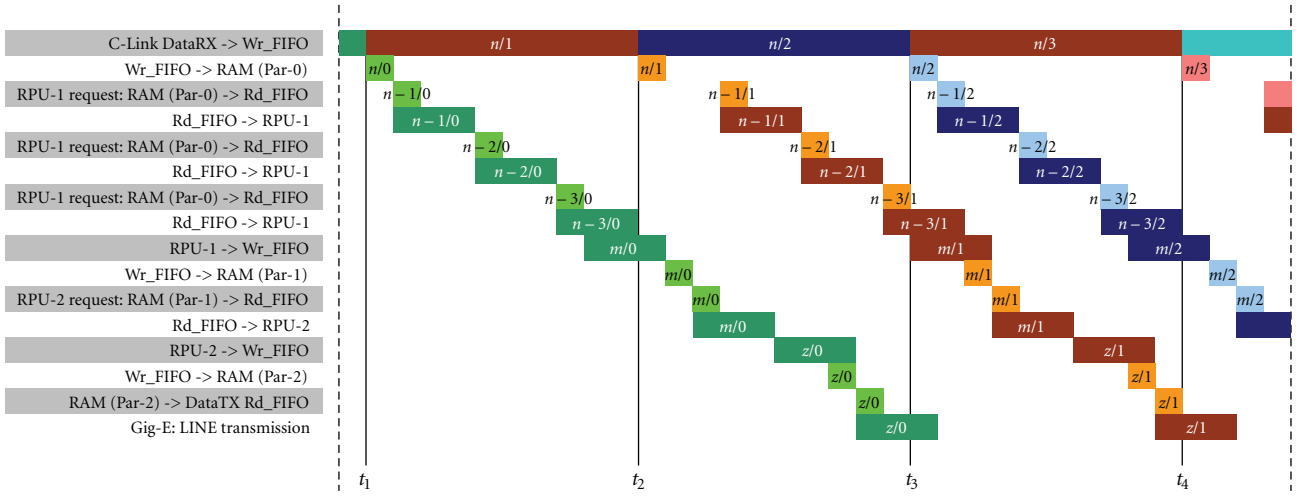


Figure 14: Traffic analysis.

the video processing pipeline. Therefore, its outputs are subsequently transferred to the connected PC. As a matter of principle, the arbitration grants only one client at a time access to the RAM, demonstrated in Figure 14. Finally, the overall latency in the regarded case study is approximately one frame and two lines, referring to the input data-rate.

## 5. Conclusion

This article presented a novel architecture of a memory controller satisfying special requirements of dynamically reconfigurable video processing platforms. At first, essential components and corresponding data paths of such a platform were specified. An adequate principle of reconfiguration, based on mutually exclusive RPUs, facilitates hiding of reconfiguration-times. In this context, application domain specific terms of real-time were regarded, and an appropriate QoS has been discussed.

Aiming at minimizing the amount of utilized bus-macros, it was a major objective of this work to define a uniform memory controller interface based on a reduced set of IOs. Thus, a corresponding interface-protocol, sharing a single bus for data and instructions telegrams, has been defined. As a result, the number of critical paths in dynamically reconfigurable designs decreases.

The article introduced a comprehensive concept of a memory controller providing adequate memory management abstraction. The proposed concept consists of four hierarchical modules and augments conventional SDRAM controllers focusing solely on low-level tasks. Architecture and implementation details were presented for all modules respectively to facilitate a deeper understanding of the nested structure.

A case study exemplified two client's sequences of events and pointed out according benefits due to the abstracted addressing scheme. Finally, a second case study implied general bandwidth calculations and provided results on abstract real time analyses.

## References

[1] K. F. Ackermann, F. Mayer, L. S. Indrusiak, and M. Glesner, "Adaptable image processing system based on FPGA modular multi kernel instantiations," in *Reconfigurable Communication-Centric SoCs (ReCoSoC '06)*, 2006.

[2] Xilinx, Inc., "Virtex-4 Configuration Guide," 2007.

[3] S. Craven and P. Athanas, "Dynamic hardware development," *International Journal of Reconfigurable Computing*, vol. 2008, Article ID 901328, 10 pages, 2008.

[4] K. F. Ackermann, L. S. Indrusiak, and M. Glesner, "System level design of a dynamically self-reconfigurable image processing system," in *Reconfigurable Communication-Centric SoCs (ReCoSoC '07)*, 2007.

[5] K. F. Ackermann, B. Hoffmann, L. S. Indrusiak, and M. Glesner, "Enabling self-reconfiguration on a video processing platform," in *Proceedings of the 3rd International Symposium on Industrial Embedded Systems (SIES '08)*, pp. 19–26, Montpellier, France, June 2008.

[6] Xilinx, Inc., "Virtex-5 Family Overview," 2008.

[7] Altera Corp., *Stratix IV Device Handbook*, vol. 1, 2008.

[8] B. Donchev, G. Kuzmanov, and G. N. Gaydadjiev, "External memory controller for Virtex II Pro," in *Proceedings of the International Symposium on System-on-Chip (SOC '06)*, Tampere, Finland, November 2006.

[9] 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 Signal Processing Systems (SIPS '03)*, August 2003.

[10] 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, June 2005.

[11] Z. Zhou, S. Cheng, and Q. Liu, "Application of DDR controller for high-speed data acquisition board," in *Proceedings of the Innovative Computing, Information and Control (ICICIC '06)*, 2006.

[12] M. Hübner, T. Becker, and J. Becker, "Real-time LUT-based network topologies for dynamic and partial FPGA self-reconfiguration," in *Proceedings of the 17th Symposium on Integrated Cicuits and Systems Design (SBCCI '04)*, pp. 28–32, Pernambuco, Brazil, September 2004.

[13] D. R. Lee, S. W. Lee, and J. W. Jeon, "Frame grabber circuit for IEEE1394 image transfer," in *Proceedings of the International Conference on Control, Automation and Systems (ICCAS '07)*, Seoul, Korea, October 2007.

[14] A. Warkentin and F. Dittmann, "Data transfer protocols for a two slot based reconfigurable platform," in *Reconfigurable Communication-Centric SoCs (ReCoSoC '06)*, 2006.

[15] PULNiX America, Inc., "Specifications of the camera link interface standard for digital cameras and frame grabbers," 2000.

[16] R. Steinmetz, *Multimedia Technology*, Springer, New York, NY, USA, 2nd edition, 1999.

[17] H. Eeckhaut, H. Devos, P. Lambert, et al., "Scalable, wavelet-based video: from server to hardware-accelerated client," *IEEE Transactions on Multimedia*, vol. 9, no. 7, pp. 1508–1519, 2007.

[18] Xilinx, Inc., "Xilinx Memory Interface Generator (MIG) User Guide," 2008.

[19] Xilinx, Inc., "Early Access Partial Reconfiguration (EAPR) User Guide," 2008.

[20] Z. Uykan, "A temporal round robin scheduler," in *Proceedings of the 68th Semi-Annual IEEE Vehicular Technology (VTC '08)*, Calgary, Canada, September 2008.

[21] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis, "Weighted round-robin cell multiplexing in a general-purpose ATM switch chip," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 8, pp. 1265–1279, 1991.