# D. The Map-oriented Machine with Parallel Data Access

The data sequencing concept presented in chapter 6 is versatile as it may be the basis for various implementations. This chapter focuses on a hardwired data sequencer implementation being targeted to build a custom computing machine (CCM) on the basis of field programmable devices. This type of machine is commonly called FPGA-based custom computing machine (FCCM). This FCCM should implement an data sequencing mechanism independent from the computational datapath. The prototype is called Map-oriented Machine with Parallel Data Access (MoM-PDA, see section D.1 at page 262). It will support concurrent data access with 2 parallel banks of Multibank DRAM (MDRAM, see appendix C).

The MoM-PDA is the fourth generation machine in the series of Map-oriented Machines (see section 4.2 at page 42, section 4.4 at page 44, and section 4.6 at page 51). All implementations of this series are based on the Xputer paradigm (see section 6.1.1 at page 90, [HHW89], [HHS92], or [AHR94]), utilize a data sequencer, and operate on 2-dimensional data memory. While all earlier implementations are generally suited to utilize reconfigurable devices, the MoM-PDA is the first prototype, which has been optimized for this task. But the most important novel feature of the MoM-PDA is the first comprehensive realization of memory access optimizations. The first time the 2-dimensional memory organization is exploited in this respect.

The hardwired data sequencer is based on a parameter stack (see section 6.4 at page 114). It provides the complete hardware needed to execute a 2-dimensional video scan. A parameter stack holds multiple video scan descriptions, which supports the generation of nested-, meshed-, and compound scans. Further this concept has been extended to a multitasking data sequencer, i.e. the hardwired data sequencer supports the generation of multiple scan patterns and thus the movement of multiple scan windows at the same time.

This chapter is structured as follows. First the MoM-PDA architecture will be introduced. The major focus will be the data sequencer of the MoM-PDA. After the detailed description of the data sequencer, the hardware details of the MoM-PDA will be reported.

## D.1 The MoM-PDA Overall Architecture

The MoM-*PDA* ([HHH99b], [HHN99a]) is an accelerator to be connected to a host computer via a PCI interface. The most important new feature of the MoM-PDA prototype is concurrent high speed access to the data [HHH98c]. Therefore it has 2 parallel banks of Multibank DRAM (MDRAM, see appendix C). Addresses for the MDRAMs are computed by the Data Sequencer and extended with burst information by the Burst Control Unit (BCU, see appendix D.2.4 at page 275 and appendix D.3.2.2 at page 286, or [Bed98]).

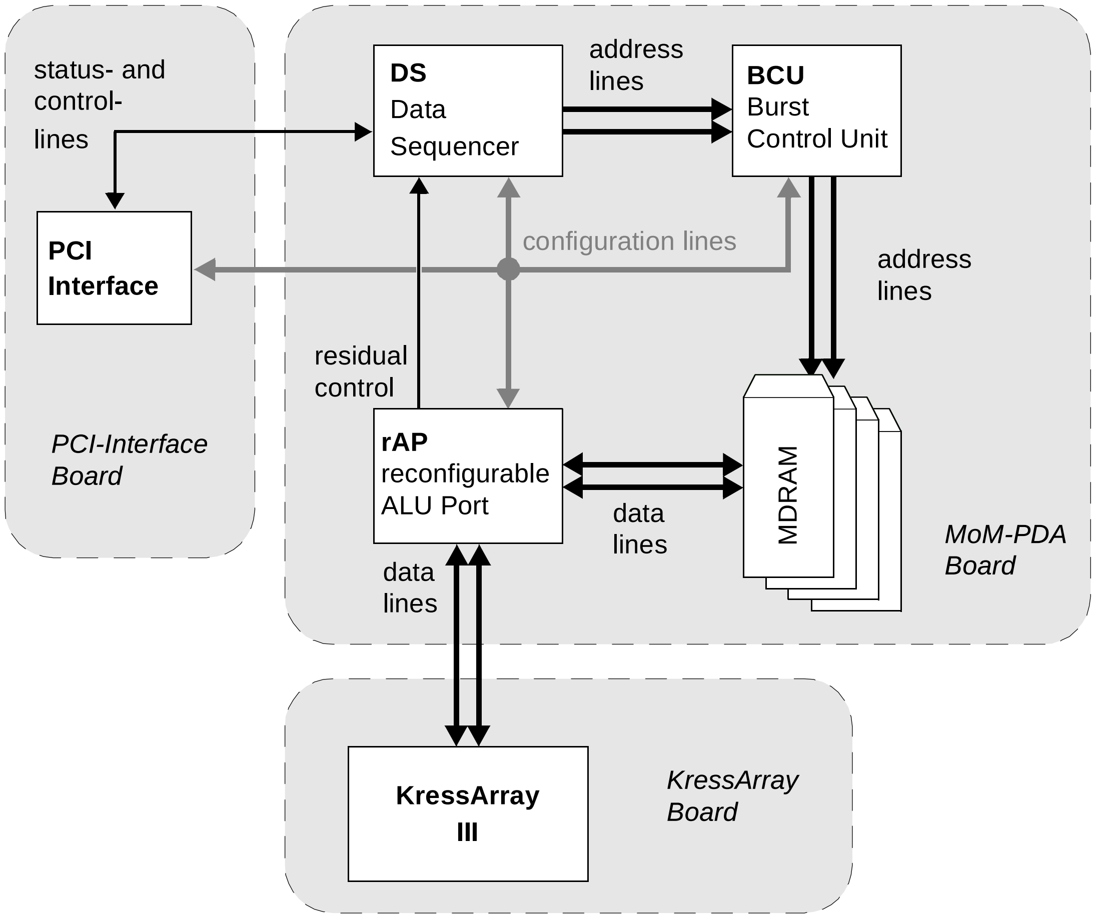

*Figure D-1:* The MoM-*PDA* machine overview.

Computations are usually performed by the KressArray. Therefore data is first routed to a reconfigurable ALU port (rAP, see appendi xD.3.2.3 at page287, or [Gil98]). The rAP acts as an interface to the memory subsystem and implements some glue logic. For small applications the rAP may also be used to perform computations. In that case the KressArray is not needed.

All components of the MoM-PDA are implemented with field-programmable logic. Figure D-1 at page 262 gives an overview on the general machine architecture. The host computer is connected to the data sequencer, the burst control unit (BCU), and to the rAP via configuration lines. Programming of the KressArray and of the MDRAMs is performed indirectly via the rAP or the BCU respectively. To enable concurrent data access all address and data lines are implemented twice. During operations the machine is controlled by the data sequencer. The host initiates only computations and observes computations via the status and control lines.

Appendix D.3.2 at page 283 deals with the MoM-PDA prototype board, which contains all elements needed to execute simple applications. A KressArray emulator has been implemented and is described in [Zim99]. In appendix D.3.3 at page 293 it is briefly introduced. The PCI interface is explained in appendix D.3.1 at page 282.

In the following the address generation pipeline of the MoM-PDA will be explained in detail. After that the multitasking capabilities will be elucidated.

## D.2 The MoM-PDA Data Sequencer

The address generation of the MoM-PDA is performed by four units, which form a pipelined datapath with three stages. The pipelined architecture forms an efficient implementation of the address generation datapath, because all units of the address generator operate simultaneously.

According to the concepts introduced in chapter 6, the Handle Position Generator (HPG) generates handle positions on the basis of the slider model (see figure 6-6 at page 100), which has already been proposed in [HHW90]. The handle positions are the basis for the second pipeline stage, where the Scan Window Generator (SWG) performs memory accesses according to the scan window model and under application of the hardware level optimizations described in section 7.1.4 at page 153.

While the generation of the handle position is independent from the number of parallel memory banks, the SWG has to generate accesses for both banks concurrently. Therefore all datapaths behind the SWG exist twice.

A Memory Mapper (MemM) optimizes the mapping of the 2-dimensional memory to the linear memory banks. After that the burst informations provided by the SWG are used by the BCU to access the MDRAM devices.

Figure D-2 at page 264 pictures the address generation datapath. The necessary hardware is distributed to two devices:

- the Data Sequencer device (see appendi xD.3.2.1 at page284), which is divided into a central control unit and the HPG, the SWG, and the MemM, and
- the BCU device (see appendi xD.3.2.2 at page286).

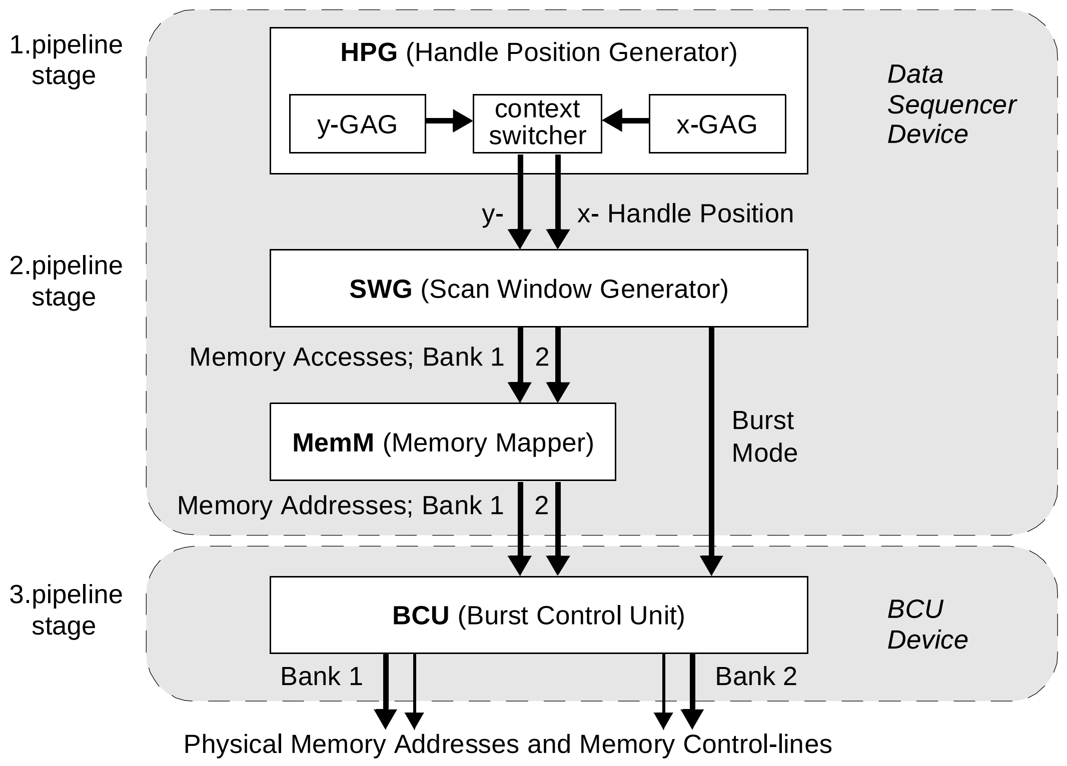

*Figure D-2:* The address generation datapath of the MoM-PDA data sequencer.

## D.2.1 The MoM-PDA Handle Position Generation

In this subsection the hardware required for handle position generation will be described. Therefore the HPG utilizes two identical x- and y-generic address generators (GAG, see figure D-2). The context switcher shown in figure D-2 will be explained in section D.2.3 at page 273.

### Application of Stack-based and Multi-level Scan Pattern Generation

Both GAGs (x and y) provide the complete hardware to generate 1-dimensional video scans. In that way the GAGs form a video scan generator according to the principle explained in section 6.3.3 at page 113. To generate compound, nested, and meshed scans the GAGs employ a parameter stack. This parameter stack works according to the parameter stack method introduced in section 6.4 at page 114. Figure D-3 at page 265 illustrates the structure of a single GAG.

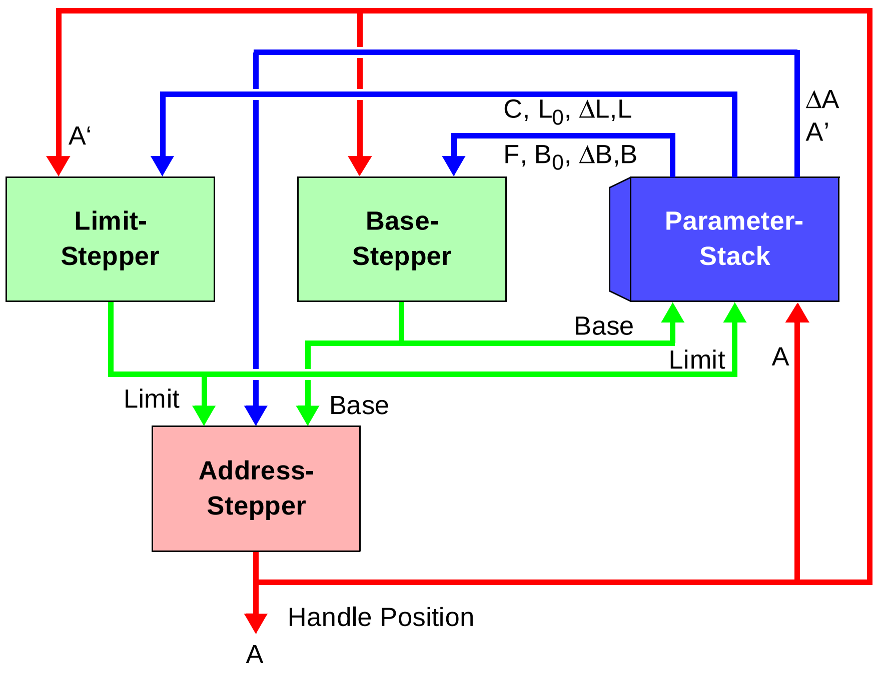

*Figure D-3:* The 1-dimensional Generic Address Generator of the MoM-PDA data sequencer.

## Optimized Stepper Implementation

Since the MoM-PDA data sequencer is implemented with an Altera CPLD (see appendix D.3.2.1) its design is a compromise between performance requirements and available design space, i.e. the Limit-, Base-, and Address Steppers have been optimized for their specific purpose. Further the Handle Position Generator is designed without the step counter proposed in section 6.3.3 at page 113. If a step counter is needed the parameters of the steppers are saved to the stack, and one of the steppers itself is utilized for step counting. While this method is slower it saves valuable design space of the CPLD. The counter mode of the stepper is described in detail in [Buc99b]. To illustrate the optimizations of the Base-, Limit, and Address Steppers, they are explained below.

## The Base- and Limit Stepper

The Base- and Limit steppers of the MoM-PDA data sequencer have an identical design. It is pictured in figure D-4 at page 267. All generated addresses are 16 bit values. Therefore the complete datapath is 16 bit.

The required parameters for Base- or Limit slider implementation are stored in a parameters stack. The MoM-PDA parameter stack has a capacity to store 64 video scans. For a single Base- or Limit stepper the parameter stack has four entries:

- The *Initial Position*:
  The *Initial Position* is a read only parameter. It is always a positive number. For Base slider implementation this value is $B_0$. For Limit slider implementation this value is $L_0$.
- The *Step Width*:
  The *Step Width* is a read only parameter. It may be positive or negative. For Base slider implementation this value is $\Delta B$. For Limit slider implementation this value is $\Delta L$.
- The *End Value*:
  The *End Value* is a read only parameter. It is relative to the *Initial Position*, and therefore it may be positive or negative. For Base slider implementation this value is Floor ($F$). For Limit slider implementation this value is Ceiling ($C$).
- The *Slider Position*:
  The *Slider Position* is a read and write parameter. If the actual programming of the stepper hardware is changed, e.g. to call a nested scan, this value is stored in the parameter stack. It is always a positive number. For Base slider implementation this value is Base ($B$). For Limit slider implementation this value is Limit ($L$).

All values of the parameter stack are initialized by the host computer. For this the *init* signal is set high and the multiplexers of the parameter memory hand over the control over the parameter stack to the host computer. For this the *Address_bus* and the *Configuration_Data* lines, driven by the PCI interface are connected to the parameter stack.

During address generation the *init* signal is low. The data input of the parameter stack is connected to the only read/write parameter, the *Slider Position.* The stack pointer *SP* points to the parameter set of the actual video scan and the *area* signals address the current stack window position.

For relative video scans, e.g. the inner scan of a nested scan, the stepper provides a register for the *Scan Pattern Location*. During parameter initialization the actual address of the outer video scan is stored in this register.

For stepper operation the *Slider Position* is initialized at the *Initial Position*. A relative scan is generated by adding the value of the *Scan Pattern Location* register. For initialization the input *BL_sel* of the input multiplexer of the *Slider Position* register is set to zero.

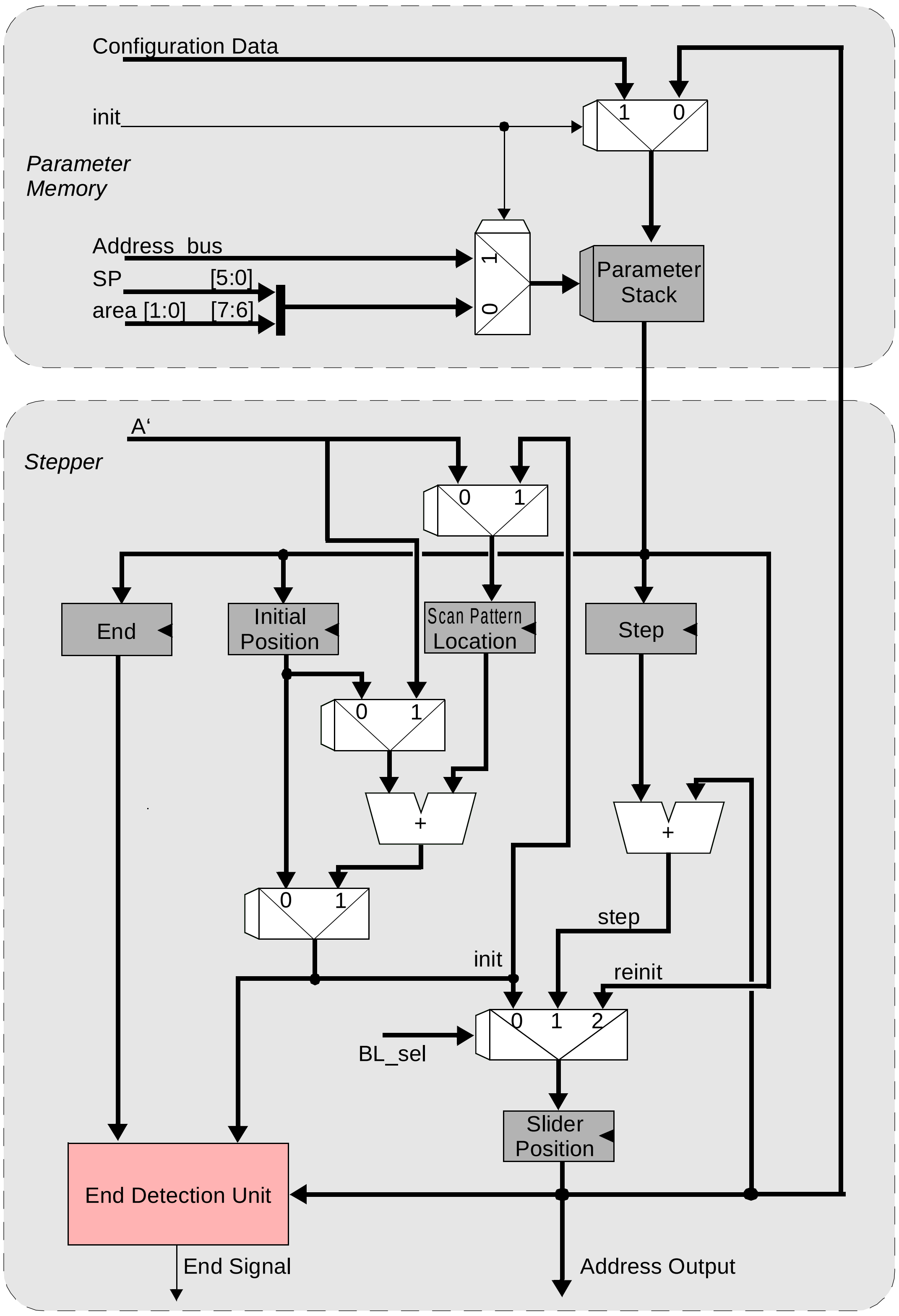

*Figure D-4:* The MoM-PDA Base / Limit stepper implementation.

For re-initialization of the stepper with already used parameters from the parameter stack, the *Slider Position* register is also loaded from the parameter stack. For this the *BL_sel* input of the input multiplexer of the *Slider Position* register is set to two.

During normal stepper operation the *Step Width* is added to the actual *position* each step to generate the next *Slider Position*. For this the *BL_sel* input of the input multiplexer of the *Slider Position* register is set to one.

The stepper generates new *Slider Position*s until the *Slider Position* exceeds the *End Value*. Figure 6-10 at page 111 shows the two possibilities for the end of address generation. For the MoM-PDA data sequencer the *End Value* is relative to the *Initial Position*. Therefore the sign of the *End Value* is used to determine, which relation has to be chosen to detect the *End_Signal*. Figure D-5 gives an overview on the End Detection Unit. The *End Value* and the *Initial Position* are added to get an absolute value to be compared with the current *Slider Position*. If the *End Value* is negative, the addition results in a subtraction. Based on the sign of the *End Value* a multiplexer selects the output of the "greater as" or "lower as" relation as an *End_Signal*.

For more details on the MoM-PDA Base- / Limit stepper please refer to [Buc99b].

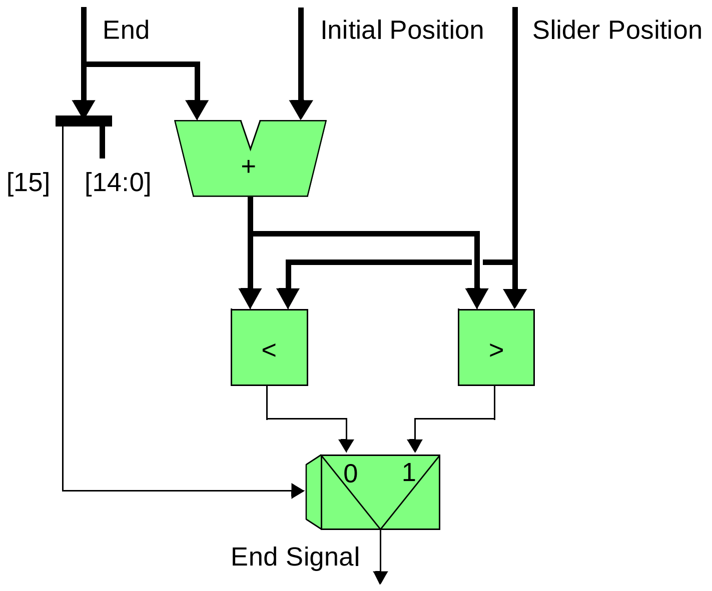

*Figure D-5:* The end detection unit of the MoM-PDA Base- and Limit steppers.

### The Address Stepper

The address stepper has been simplified to save design space of the target CPLD. The parameter stack is the same as for the Base- / Limit stepper (see figure D-3 at page 265). The optimized Address Stepper is shown in figure D-6 at page 270. Only two values are accessed from the parameter stack:

- The *Step Width*:
  The *Step Width* is a read only parameter. It may be positive or negative. For Address slider implementation this value is $\Delta A$. While all datapaths of the Address stepper are 16 bit, the *Step Width* is only 15 bit. Further the address stepper has no register for the *Step Width*, it is directly accessed from the parameter stack.
- The *Slider Position*:
  The *Slider Position* is a read and write parameter. If the actual programming of the stepper hardware is changed, e.g. to call a nested scan, this value is stored in the parameter stack. It is always a positive number.

The *Initial Position* for the Address stepper is obtained from the Base stepper. The actual *Base* value is not stored in a register since the Base stepper already contains a register for *Base*. Also the *End Value Limit* is not stored in a register, since the Limit stepper provides a register for *Limit*.

For the Address stepper the *Initial Position* (*Base*) and *End Value* (*Limit*) are absolute parameters. Similar to the *Base*- and *Limit* generation the *End Value* may be higher or lower as the *Initial Position*. Therefore the *Step Width* must be adequately chosen, i.e. it must be positive or negative.

The End Detector of the Address stepper uses the sign of the *Step Width* to select between the "greater as" or "lower as" relation to drive the *End_Signal*.

## D.2.2 The MoM-PDA Scan Window Generation

Scan window generation is the second stage of the two level data sequencing. On this stage all memory access optimizations are performed. Therefore the scan window generation of the MoM-PDA data sequencer includes two important steps:

- the generation of the memory accesses to all scan window locations on the basis of the actual handle position, and
- the dynamic assignment of scan window rows to the two parallel memory banks of the MoM-PDA.

Section 6.1.3 at page 94 has already introduced a basic scan window generator architecture for one memory bank. Figure D-7 at page 271 shows the overall MoM-PDA scan window generator structure. It consists of the following units:

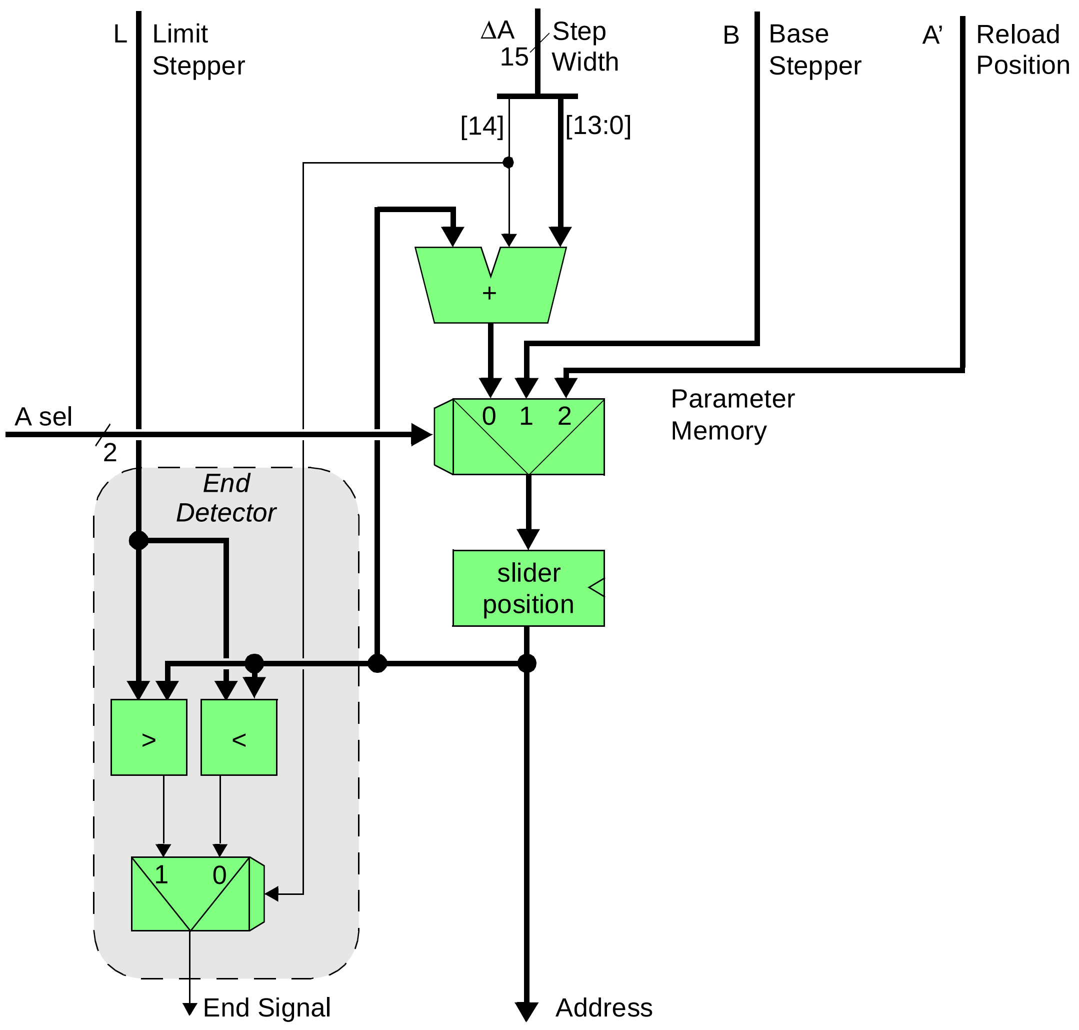

*Figure D-6:* The Address stepper of the MoM-PDA data sequencer.

- The offset generation. This unit provides two identical offset generators, which generate relative addresses to be added to the actual handle position. One offset generator produces all accesses to even scan window rows and the other offset generator is assigned to the odd scan window rows.
- The adder units add the output of the offset generators to the actual handle position. There is one adder unit for each offset generator.
- The memory bank switcher assigns dynamically the scan window rows, i.e. the sum of the handle position and the outputs of the offset generators, to the parallel memory bank. The assignment depends on the handle position (see figure 7-4 at page 151). Since one memory banks contains all odd rows of the

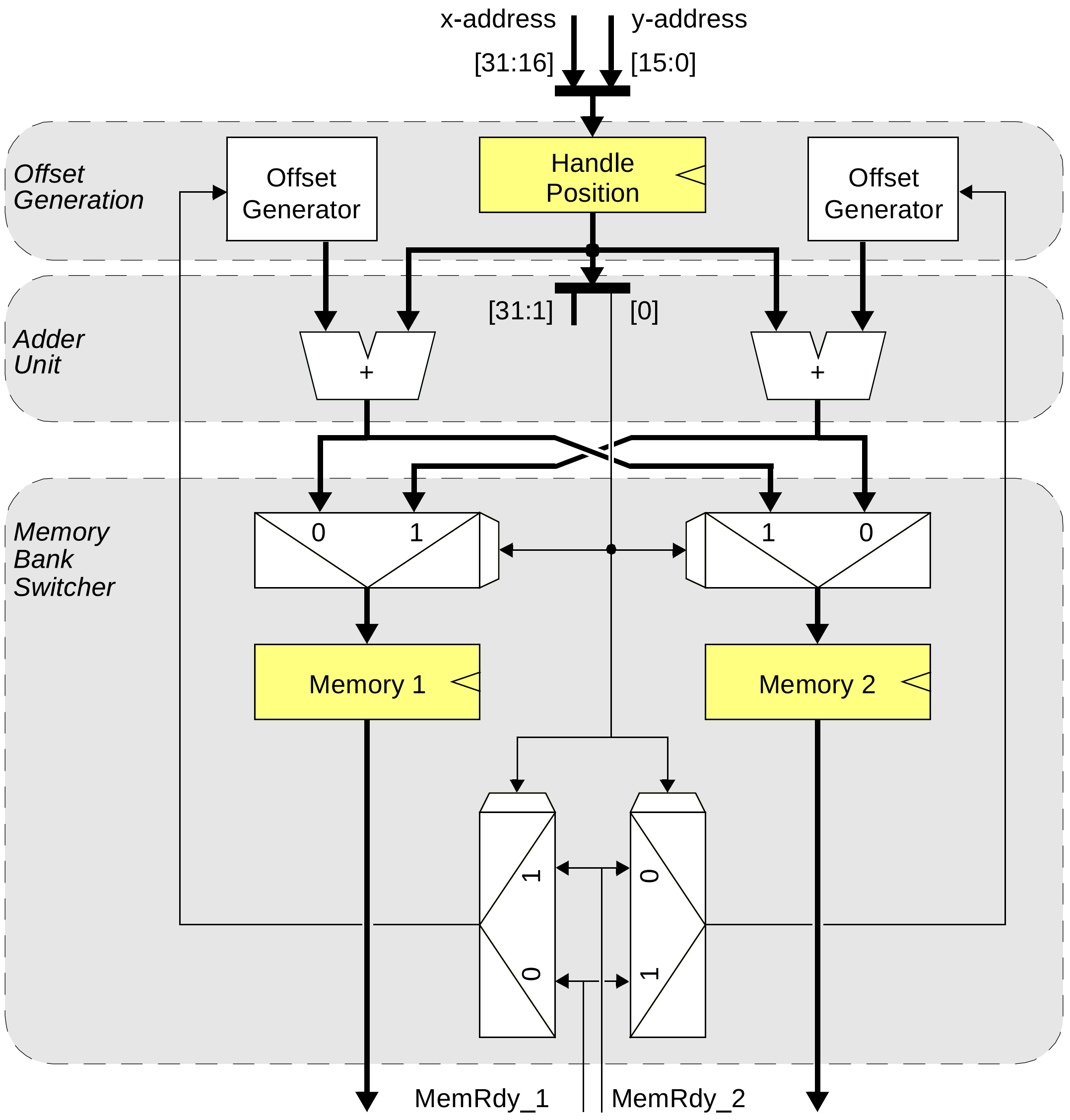

*Figure D-7:* The Scan Window Generator of the MoM-PDA data sequencer.

2-dimensional memory, and the other all even rows, the scan window rows must be assigned appropriately. If the y-part of the handle position is odd, the odd scan window rows are assigned to the memory bank with the odd rows. Otherwise, if the y-part of the scan window is even, the odd scan window rows are assigned to the memory bank with the even rows.

## The Offset Generator

Each offset generator produces relative addresses to be added to the actual handle position. If a new handle position is generated, the offset generator starts generating addresses. Since the accesses inside a scan window are often irregular, they are stored in a look-up table. Figure D-8 shows the organization of the look-up table of one offset generator. It has a capacity of 16 different scan windows (Task 0-15) each of 512 entries.

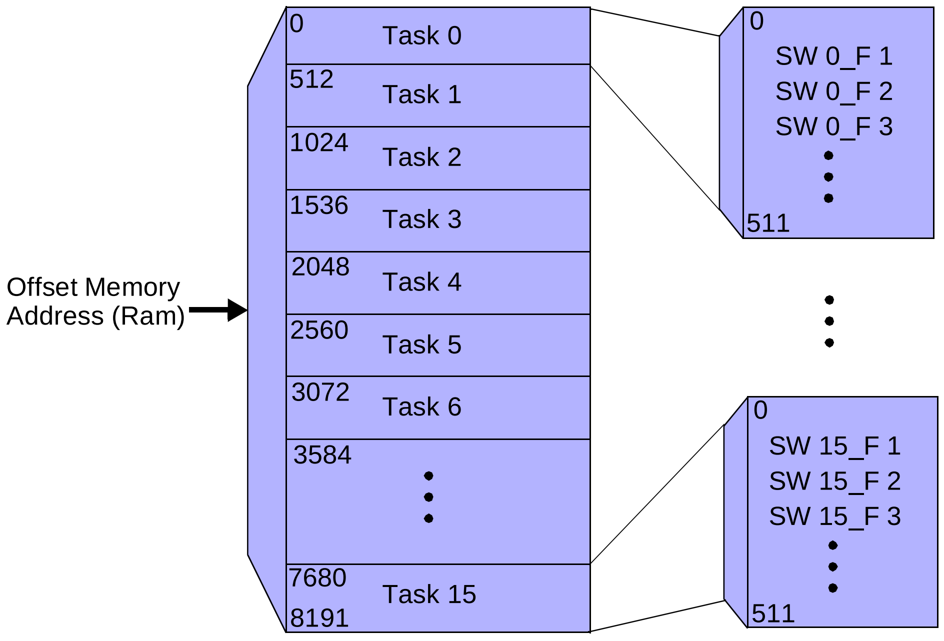

*Figure D-8:* The organization of the Offset Memory of the MoM-PDA Scan Window Generator.

The offset generator (see figure D-9 at page 273) selects between the different scan windows on the basis of the signal *Task_Nr*. This four bit signal is used as the MSW[1] of the *Offset_Memory_Address*. The main components of the offset generator are an incrementer and a 16 word by 9 bit register file. Each word of the register file is assigned to a scan window and selected by *Task_Nr*. The register file holds pointers to the look-up table memory. This way the current state of the scan windows may be saved.

The output of the look-up table (*Offset_Memory_Data*) may also be used to load the register file with a new value. For this the signal *detect* must be set high. All other multiplexer are needed for reconfiguration of the scan window generator by the host computer. The signal *init* is used to pass the control of all address lines and control signals to the host.

For more details please refer to [Buc99b].

[1]. MSW = most significant word

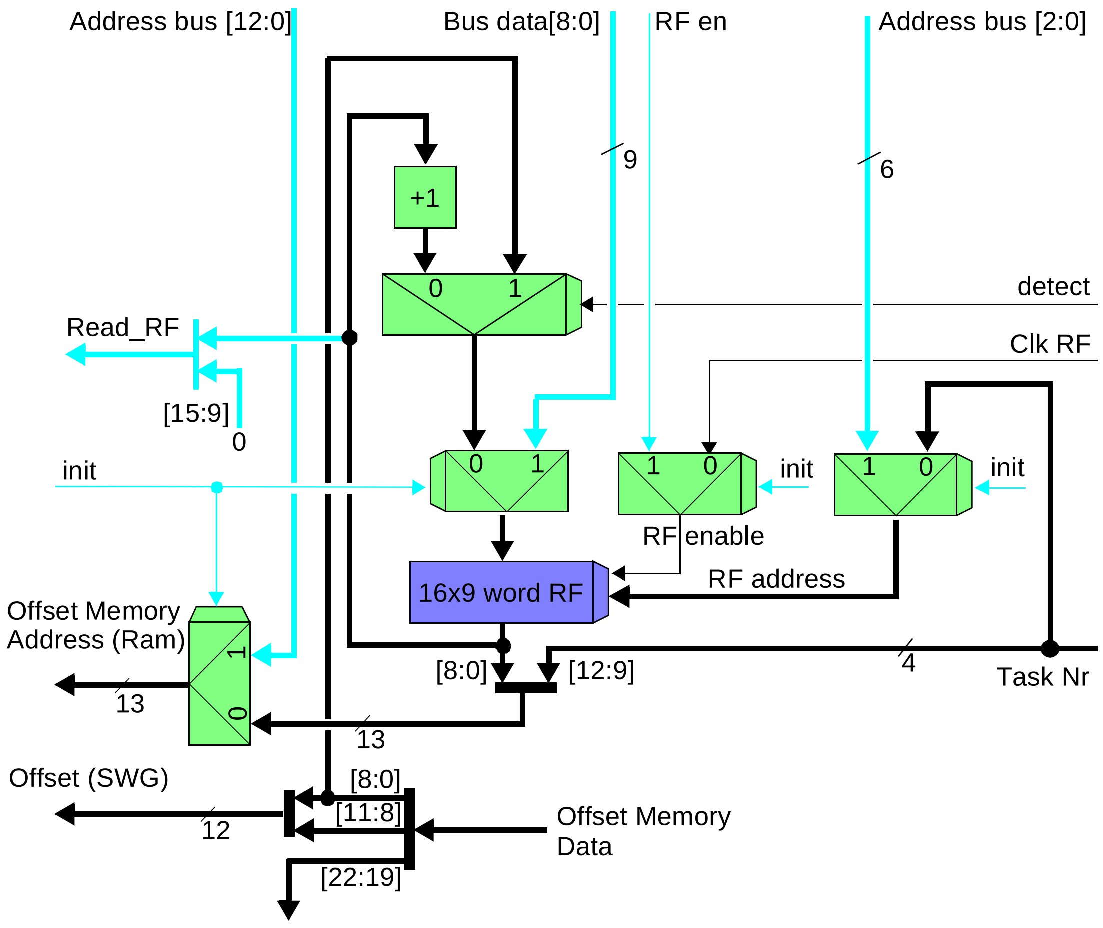

*Figure D-9:* The offset Generator of the MoM-PDA Scan Window Generator.

## D.2.3 The MoM-PDA Memory Mapping

To optimize the exploitation of the available memory resources the MoM-PDA performs an enhanced memory mapping. This mapping is needed for two reasons:

- Since the MoM-PDA provides multitasking features (see sectionD.2.5 at page 281), the application data of multiple applications, may be in the data memory at the same time. Such applications may also use the same address range.
- The MoM-PDA utilizes 16 bit address generators for x andy, but many applications do not use the complete address range (see section7.1.2.3 at page 151). Therefore an enhanced mapping of 2-dimensional addresses to the linear memory banks is needed.

### Dynamic Data Mapping for Multitasking Applications

Usually in multitasking systems it is not known during design time, which applications will be executed at the same time. Multitasking systems respectively the runtime system of multitasking systems perform a dynamic scheduling of application tasks. This may cause a fragmentation of the data memory, or that two application use the same address range. In both cases a hardware support for re-mapping of the application data is needed.

- If multiple applications use the same address range, at least one application must be re-mapped to an unused address space.
- If multiple applications use non-overlapping address ranges, but do also not border on each other, the data memory is fragmented and address space is wasted. In this case the address ranges are moved together.

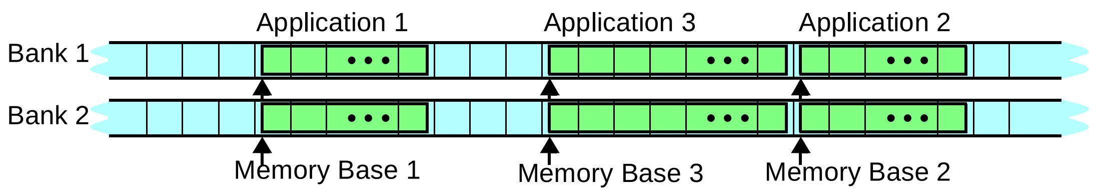

*Figure D-10:* Data of several applications mapped to the parallel memory banks.

This strategy leads to an own address space for each data block in the 2-dimensional memory. If there is data for several applications in the physical memory the runtime system has to secure that there is no memory violation. For each application a *Memory Base* address is added and the necessary space is allocated. The required mapping is done by the context switcher, which is located in the HPG (see figure D-2 at page 264). The *Memory Base* address simply moves the application data in the data memory to the allocated location. It is determined dynamically by the runtime system. To achieve an arbitrary movement the *Memory Base* address may be positive or negative. Figure D-10 illustrates the data of three applications mapped the parallel memory banks.

Figure D-11 at page 275 shows the context switcher hardware. Two 16 word by 16 bit register files hold the x- and y- *Memory Base* address for all applications. The appropriate *Memory Base* address is selected by the *Task_Nr* signal and added to each handle position generated by the HPG.

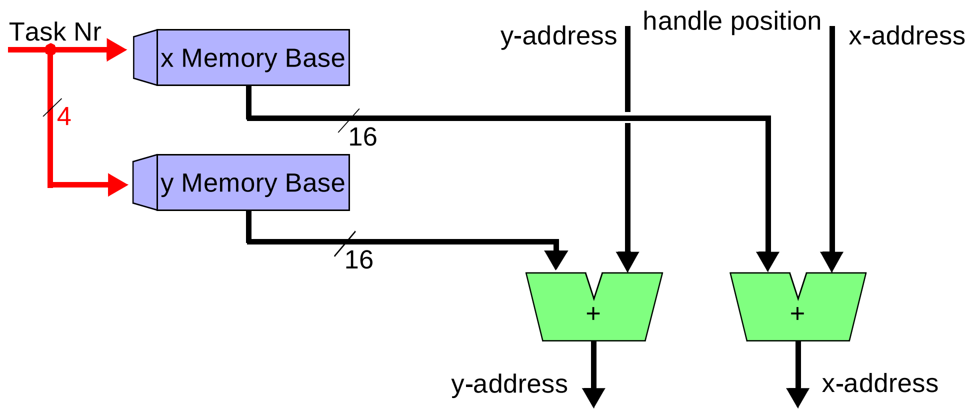

*Figure D-11:* Context switcher implementation of the MoM-PDA data sequencer.

### Avoiding Memory Fragmentation

The x- and y-parts of the handle position are composed as described in sect i on7.1.2.3 at page 151 to avoid memory fragmentations caused by unused MSBs of the x address. Figure D-12 at page 276 gives an overview on the memory mapper (MemM) implementation. Since two parallel address streams are generated by the SWG, the memory mapping hardware is needed twice. The shift operations for each memory bank is the same. A barrel shifter may shift the y part of the address between zero and 15 bits. A 15 bit or gate merges the output of the barrel shifter and the 15 MSBs of the x address. This mapping scheme represents an improvement compared to the Map-oriented Machine 3, where only four fixed address mappings are supported (see conclusions of section 4.6 at page page 55).

## D.2.4 Interfacing Multibank DRAM

To support memory devices with burst options an additional unit has to control the burst operations. The required signals for variable burst lengths are generated by the Burst Control Unit (BCU) based on the burst information provided by the SWG. Because the memories are accessed in parallel, the BCU hardware is instantiated for each memory bank.

This section will give an overview on the Burst Control Unit (BCU) characteristics and functionality. For a detailed description of the BCU refer to [Bed98]. For implementation details see appendix D.3.2.2 at page 286. A description of the Multibank DRAM (MDRAM) functionality is given in appendi xC.

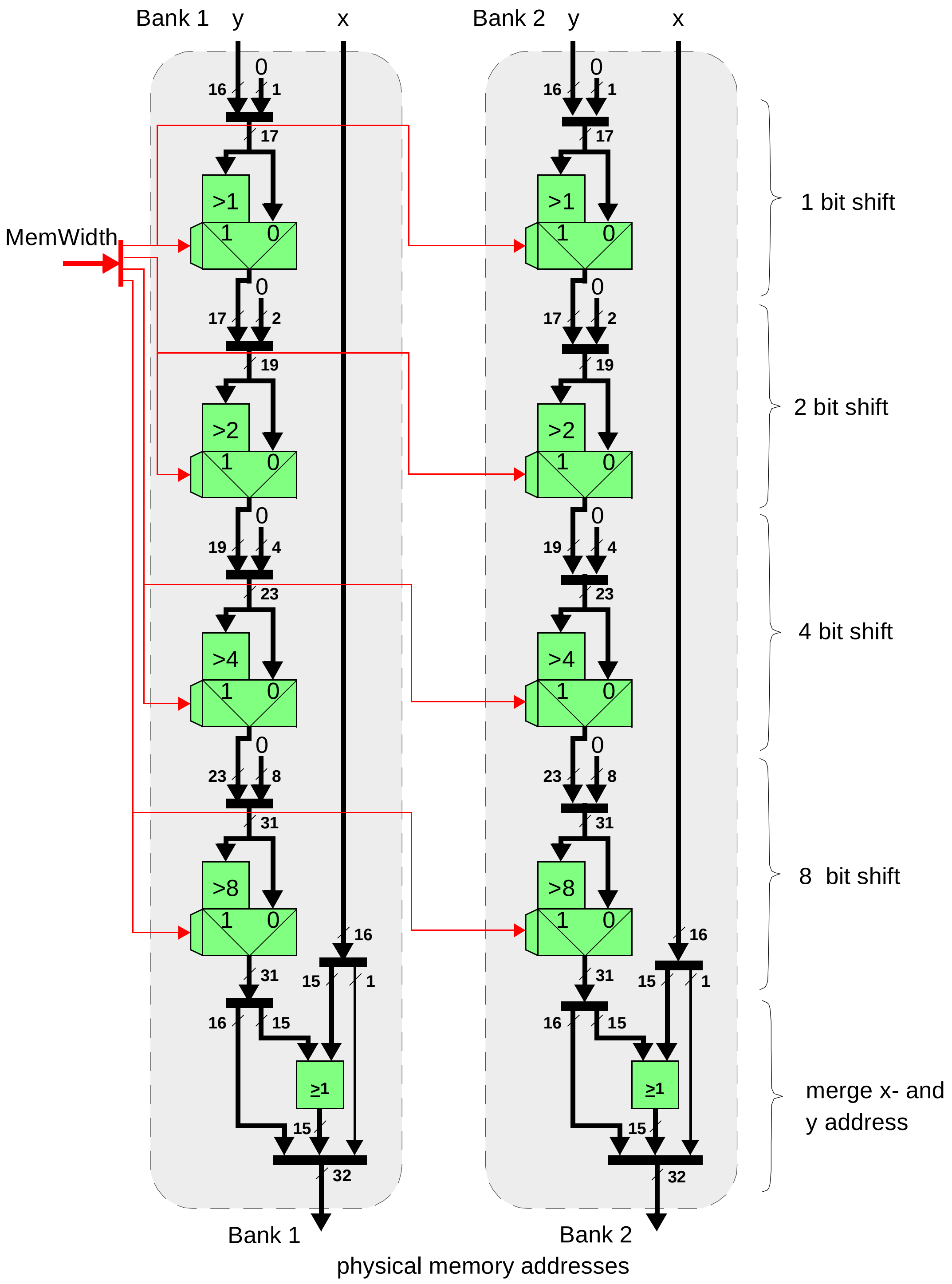

*Figure D-12:* The memory mapper of the MoM-PDA data sequencer.

## The Main Tasks of the Burst Control Unit

The main tasks of the BCU are

- interfacing the MDRAM to the host computer for initialization,
- getting burst requests from the data sequencer and starting the appropriate burst operation,
- performing handshake to the reconfigurable ALU (rALU), and
- performing MDRAM-related operations like bank ID reprogramming, refreshing etc.

The MoM-PDA memory is divided in two concurrently accessible sectors (see figure 7-4 at page 151) in order to implement a parallel memory system as described in section 7.1 at page 147. Thus, the BCU is connected to two parallel MDRAM busses and all address and handshake lines to the data sequencer and the rALU must be instantiated twice. The structure of the BCU/MDRAM bus system is shown in figure D-13 at page 278.

According to the two bank interleaving scheme, some of the BCU components are instantiated twice and others only once. The components for the MDRAM setup and initialization via the host computer are only active before and after scan window operations. Thus, the memory sectors don’t need to be accessed simultaneously for these tasks and these components exist only once. All other parts have two instances, because during scan window operation the sectors must be accessed concurrently.

## Burst Operations

During scan window operation, every read/write burst is initialized by a burst request signal from the SWG. When generating a burst request, the SWG has to supply a parameter set describing the burst. It contains the start address of the burst, the burst length and the burst type (read or write). Based on this information the burst split unit decides, if the burst data can be read/written completely from/to a single line of a DRAM bank. That is, if the start- and end- (see equation 9-1) address of the burst are located in the same bank.

$$\textit{end address} = \textit{start address} + \textit{burst length} \qquad \textit{(Eq. 9-1)}$$

If this is not the case, the burst will be split into two smaller bursts executed subsequently in different DRAM banks. To ensure that both parts of the burst will be located in different banks, internally an interleaving scheme is used for bank addressing (see figure 7-10 at page 159). The benefit of this scheme is the possibility to use hidden RAS (refer to appendi xC for hidden RAS). If both sub-bursts would be located in different rows of the same DRAM bank, the second row could not be activated before the first sub-burst is completed. When the rows are located in different banks, both banks can be activated before the burst starts. Figure 7-10 at page 159

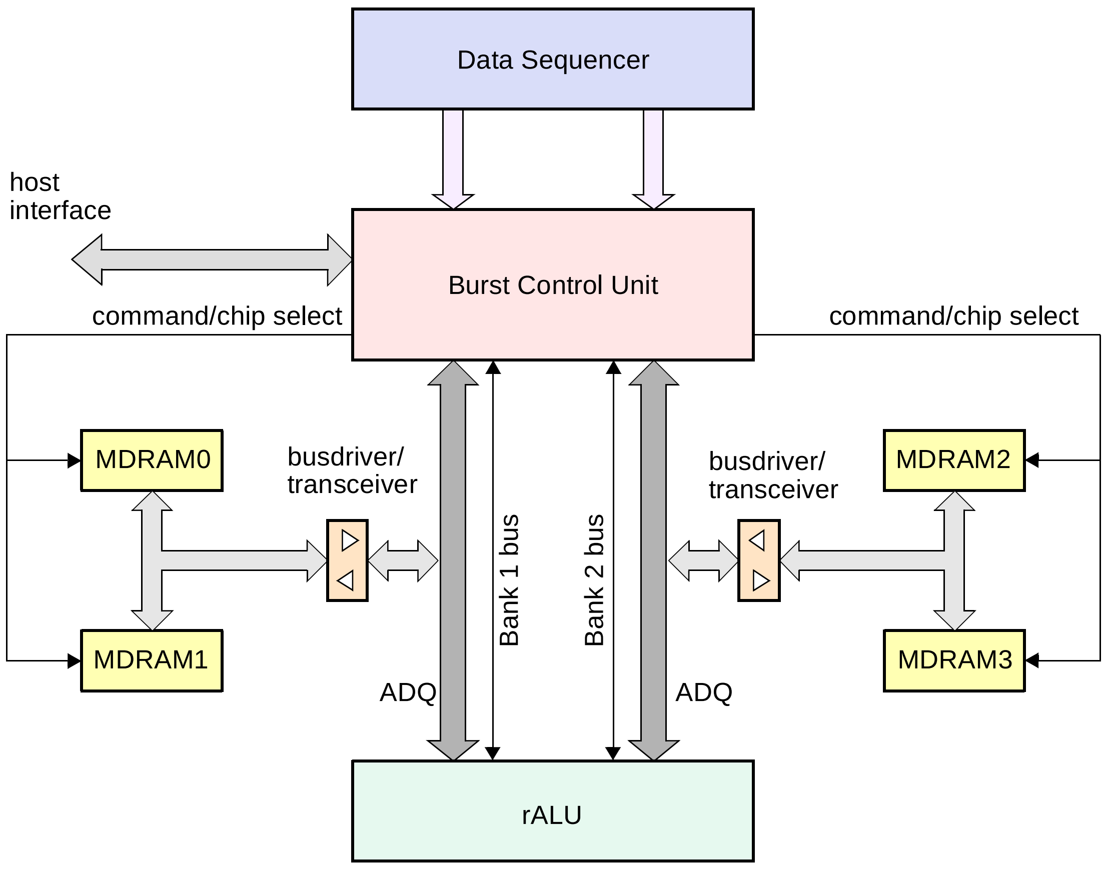

*Figure D-13:* Schematic of the Burst Control Unit / MDRAM bus system of the MoM-PDA.

clarifies the addressing scheme. The start address of a burst consists of the three parts: bank address (in MoM-PDA 6 bits), row address (8 bits) and column address (5 bits) in the following order:

$$row \Rightarrow bank \Rightarrow column \qquad \text{(Eq. 9-2)}$$

Besides the address bits for addressing each parallel bank, another address bit is used for selection of the bank to be addressed when loading the memory via the host computer interface.

The burst split unit (see figure D-14 at page 279) also performs any handshake to the data sequencer. After receiving a burst request from the data sequencer, the burst split unit will forward the start address and the burst length to the register file, which contains registers and counters for burst control. The burst control state machine is also informed about the burst request. If neither a burst nor a refresh is in working on, the burst control state machine will activate the appropriate banks for the requested bursts using the addresses stored in the register file. After that, control is passed on the read/ write logic unit, which mainly performs handshake to the rALU. Performing a burst,

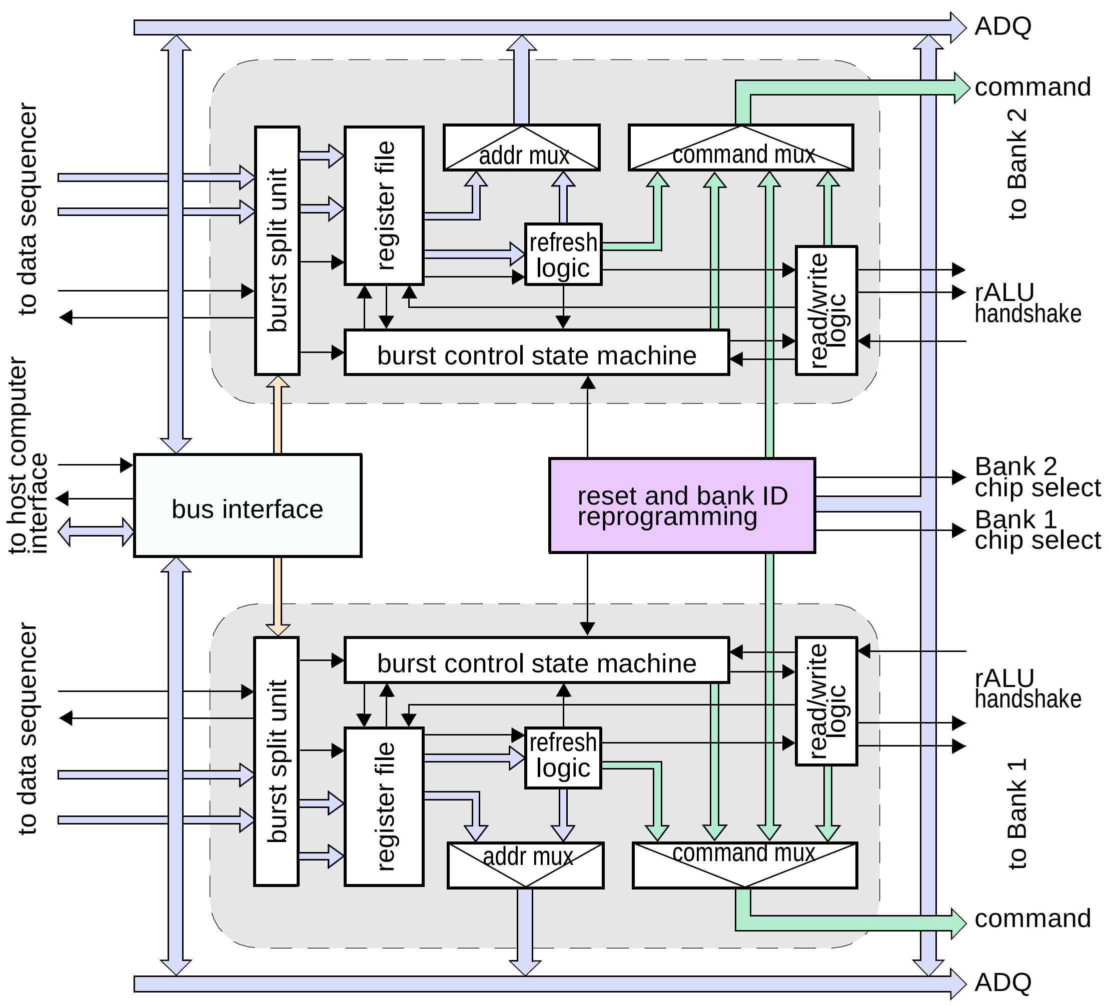

*Figure D-14:* Structure of the MoM-PDA Burst Control Unit.

the register file will hold the number of the currently transferred data words in order to generate a ready signal when the burst is completed. The ready signal will pass control back to the burst control unit, which will precharge the banks, using addresses from the register file again.

The burst control state machine is the most complex part of the BCU. Its task is to organize the incoming burst request in an optimal way. Further it detects if two successive bursts take effect to the same row of the same bank. In that case no precharge/activate operation is necessary in between.

## The Address and Command Multiplexers

All addresses for the controlled MDRAM are generated either by the register file or by the refresh logic. The address multiplexer will select the currently active source for the addresses. When a burst operation is in work, data will transferred over the *ADQ* bus (see figure D-13 at page 278 and figure D-14 at page 279), either driven by the rALU or by the MDRAM itself. During data transfer, the address multiplexer, the host computer interface connection, and the reset and bank ID reprogramming unit are disconnected from the BCU-internal part of the *ADQ* bus.

### The Refresh Logic

The refresh logic contains a counter which is initialized with an appropriate value and then decremented by one every clock cycle. If the refresh counter is zero, refresh logic will generate a refresh request signal. If no other operation is in work, refresh logic will get an *OK* signal from all other units and a quadruple refresh will be performed immediately [Bed98]. Otherwise, the currently running burst operation must be interrupted. The read/write logic unit will send a *STOP* command to break the burst and send a hold signal to the rALU. Just then it will generate the *OK* signal for the refresh logic. When the refresh operation begins, the refresh counter is re-initialized to its start value.

### Initialization via Host Interface

Reading and writing MDRAM through PCI interface is almost the same process as data accesses initiated by the data sequencer except the fact that only bursts of the length 1 are performed. The bus interface buffers two 16bit data words and an address from the host interface. Then a length one burst request is sent to burst split unit and the burst is initialized as described above. The read/write logic unit performs the handshake to the bus interface instead to the rALU. The buffered data words are written directly to the BCU-internal *ADQ* bus.

### The MDRAM Setup

MDRAM has to be initialized before scan window operations. Setting up includes the following processes:

- memory reset to synchronize internal state machines and init bank ID registers,
- loading the mode register with an appropriate value. This must be done in order to switch off the PLL for operation at low clock frequencies (refer to data sheet, [Sie97]), and
- reprogramming of the bank ID registers.

The setup process is performed by the reset and bank ID reprogramming unit. This unit is started immediately after a hardware reset. When the setup procedure is completed, the *init_ready* signal will be generated to enable the other units for normal operation.

### D.2.5 Multitasking

Because of the stack based implementation of the HPG, all steppers provide an infrastructure for a quick parameter exchange. This infrastructure can also be exploited to set up a multitasking system. Parameters are not only changed to generate complex scan pattern but also to run multiple scan patterns simultaneously. This novel data sequencer structure handles up to 16 parallel tasks each consisting of a complex scan pattern. Therefore up to 16 scan windows may operate on the data memory concurrently.

The computation of the parallel tasks is done like known from multi-tasking systems. All tasks have the same priority and are stored in the Task-List (figure D-15). The Task-List is a 16 word by 6 bit memory which holds pointers (*SP*) to a Scan Parameter List and the Stepper Parameter Stack. The Task-List is processed according to the

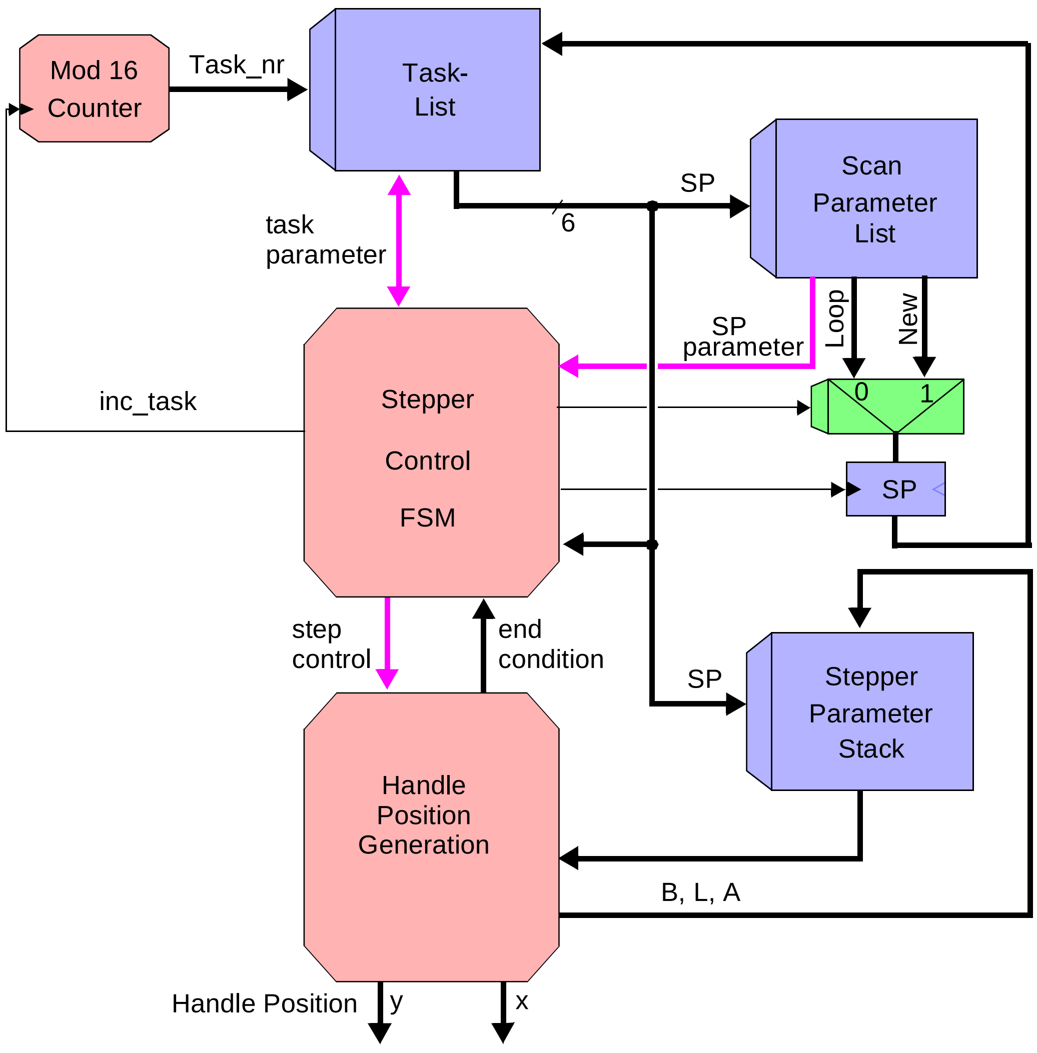

*Figure D-15:* The task manager of the MoM-PDA data sequencer.

Round Robin method, i.e. the Task-List is processed in an infinite loop. Each task generates exactly one handle position before the next task is called by the Stepper Control FSM. The Stepper Control FSM sets the signal *inc_task* to trigger the Mod 16 Counter to increment the *Task_nr*.

The Stepper Parameter Stack holds all parameters for the HPG (see also fig u reD-4 at page 267). The Scan Parameter List stores pointers for each video scan to other video scans for the generation of compound-, nested-, and meshed scans. If the HPG indicates with the *end_condition* signals, that another video scan has to be invoked, the Stepper Control FSM loads a new pointer from the Scan Parameter List into the Task List. If the computation of a task has finished the Stepper Control FSM resets the running flag of the current task in the Task-List via the *task_parameter* lines.

For more details on the multitasking concept please refer to [Buc99b].

## D.3 The Hardware Components of the MoM-PDA

In this chapter the hardware components of the Map-oriented Machine with Parallel Data Access (MoM-PDA) will be introduced. The concepts of the MoM-PDA data sequencer have been described in appendix D.2 at page 263. The complete MoM-PDA prototype consists of three printed circuit boards (PCBs):

- the PCI interface board,
- the MoM-PDA board, and
- the KressArray emulator board.

In the following subsections these boards and the main components utilized on these boards will be described.

### D.3.1 The PCI Interface Board

To implement the interface to a host computer, the MoM-PDA prototype utilizes a commercial FPGA board. The FPGA of this board is used to implement glue logic to interface the MoM-PDA components. Figure D-16 at page 283 shows the used HOT Works PCI Board by Virtual Computer Corporation [VCC]. The FPGA board consists of:

- a XC6216 (see [Xil96]) for the user-designs,
- a XC4013 (see [Xil99]) implementing the PCI bus interface,
- up to 2M Bytes of fast SRAM for user-data, and
- a programmable clock-generator.

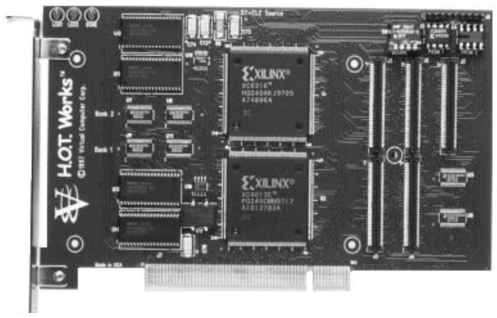

*Figure D-16:* XC6216 PCI Board from Virtual Computer Corporation.

The XC6216, the SRAM and a set of configuration registers are mapped to the memory space of a host CPU. This allows the host CPU to read or write the SRAM memory of the board and configure or read or write the user FPGA and the user design. The board memory is organized into two banks (Bank 1 and Bank 2, see figure D-17 at page 284) of 128Kx8 SRAMs. The banks have separated address and data buses. With the use of multiplexers and bus switches various modes of operation are allowed.

Configuration of the XC6216 is done using the east address and west data ports (see figure D-17 at page 284). The control port of the XC4013 is needed to switch the multiplexer and bus switches to the different modes of operation. During read/write operations from PCI to board memory the XC6216 must keep its output ports (east data) in high impedance state. The control port is also used to program the clock generator.

## D.3.2 The MoM-PDA Board

The main components of the prototype are placed on the MoM-*PDA* prototype board (figure D-18 at page 285). The complete prototype implementation is based on FPGAs. The board contains the Data Sequencer, the reconfigurable ALU Port (rAP) and the Burst Control Unit (BCU). Further 2 parallel banks of Siemens MDRAM chips are contained. Each MDRAM chip has a capacity of 1MB and operates at a clock frequency of 120 MHz.

The MoM-*PDA* prototype board has 3 connectors:

- An ISA-port connector is used for power supply.
- A connector to the host interface.
- The connector to the rALU board allows to connect an external board with a KressArray. Since the rAP is able to perform computations, a KressArray is not always needed.

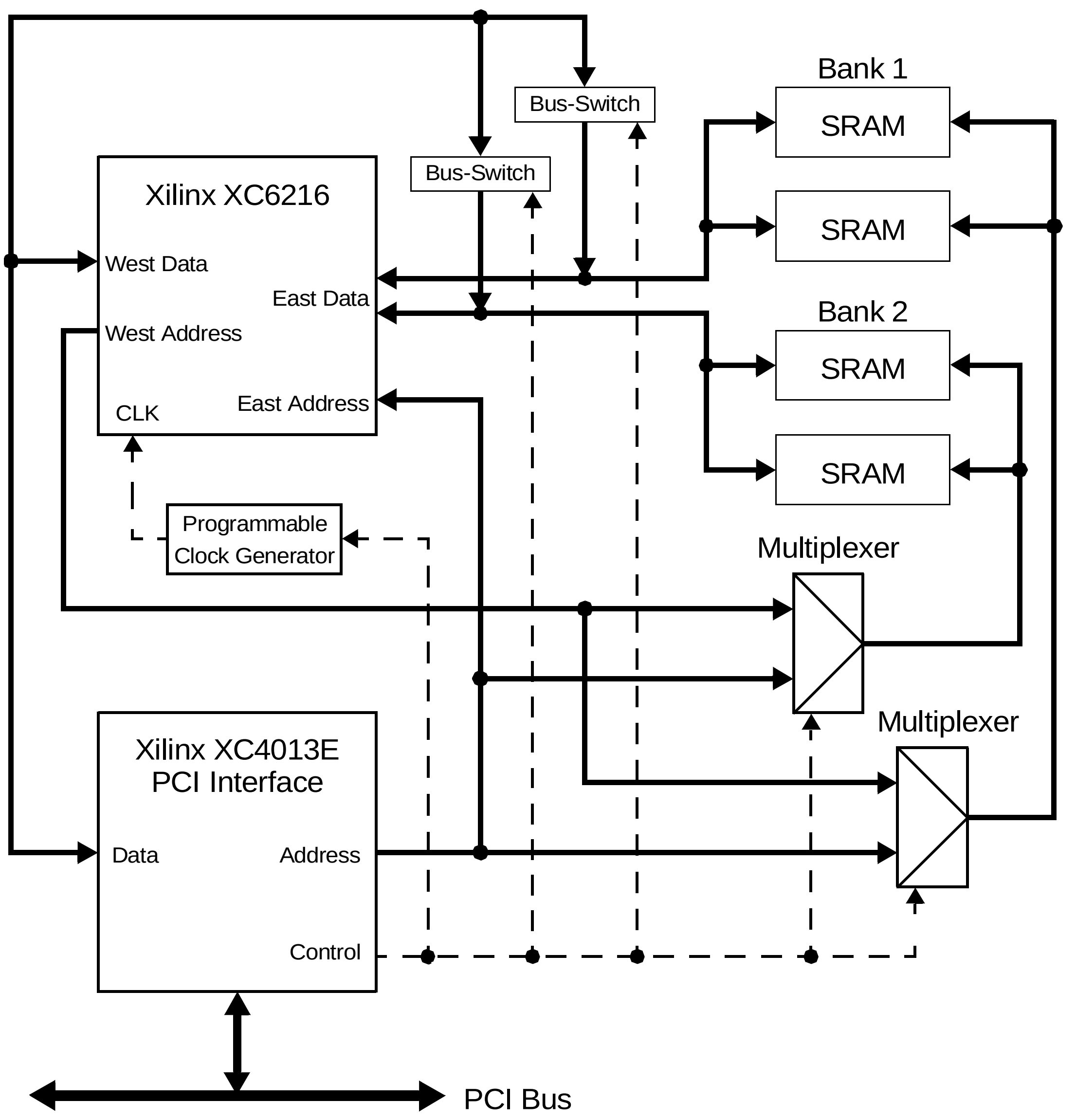

*Figure D-17:* Hot Works Board Architecture.

### D.3.2.1 The Data Sequencer

The concepts of the external data sequencer used in the MoM-PDA have already be described in appendix D.2 at page 263. In this subsection only the technical details of the CLPD[1]-based implementation will be listed. The target architecture for the implementation is an Altera FLEX[2]10k100 GC503-4 CPLD.

[1] CPLD = Complex Programmable Logic Device
[2] FLEX = Flexible Logic Element matriX

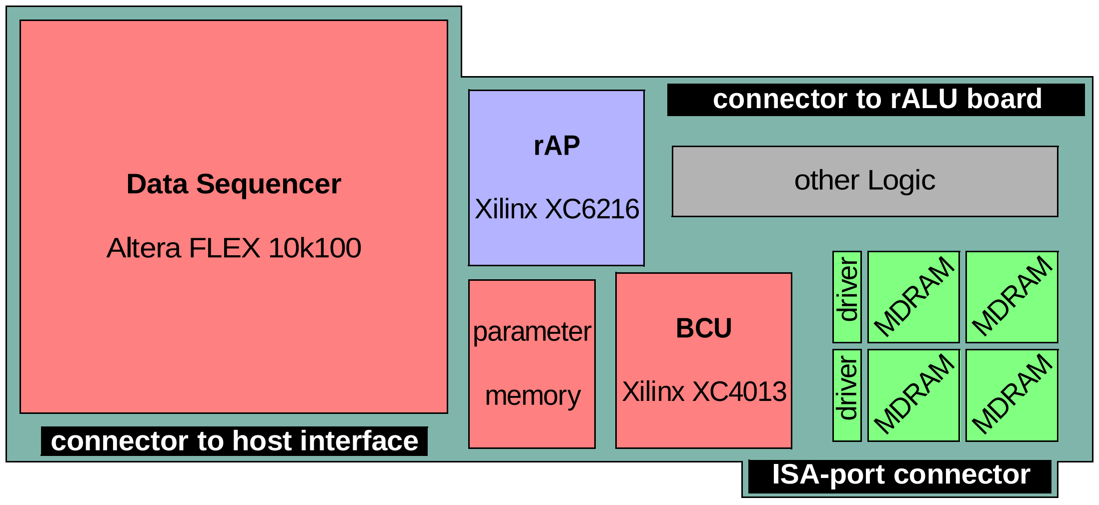

*Figure D-18:* MoM-PDA prototype board.

## Technical Details of the Altera FLEX 10k1000 CPLD

| | |
|---:|:---:|
| Number of Logic Elements (LEs) | 4992 |
| Number of Logic Array Blocks (LABs) | 624 |
| Number of Embedded Array Blocks | 12 of 2048 Bit |
| Number of memory bits | 24576 |
| Number of Input/Output Elements (IOEs) | 406 |
| Number of gates | 62000 - 158000 |
| Typical number of gates | 100000 |
| Number of device pins | 503 |
| Maximum clock frequency | 63 MHz |
| Size of the configuration file | 149134 Bytes |

*Table D-1:* Technical details of the Altera FLEX 10k100 CPLD.

More details can be found in section 3.2.3 at page 31, [Bra98], [Buc99a], [Buc99b], and [Alt98]. Figure D-19 at page 286 illustrates the CPLD adaptor developed in [Bra98] with the Altera FLEX 10k100 CPLD chip. Table D-2 at page 286 lists details of the CPLD implementation of the data sequencer. Technical details of the data sequencer are given in table D-3 at page 286, and of the scan window generator in table D-4 at page 287.

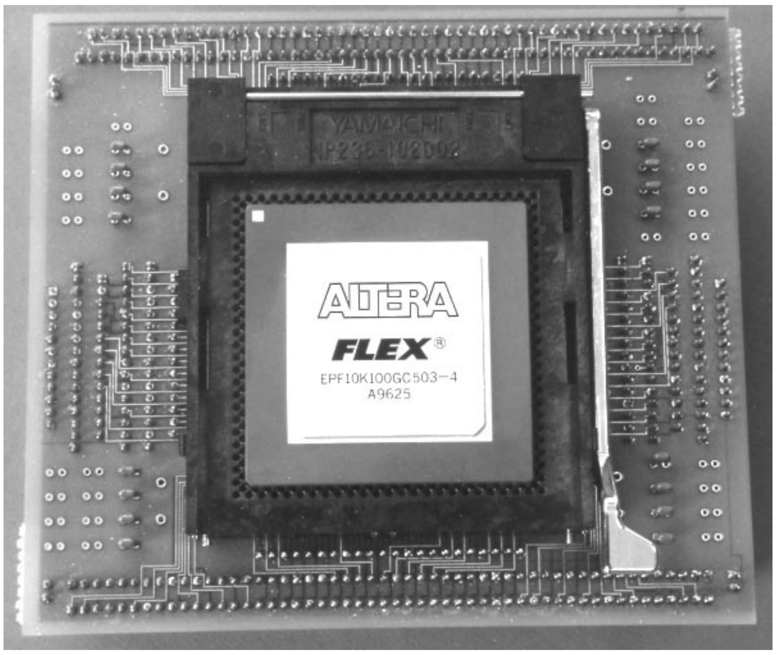

*Figure D-19:* Altera FLEX 10k100 device with adaptor board.

| | |
|---|---|
| Number of used Logic Elements (LEs) | 4758 (95%) |
| Number of used Embedded Array Blocks (EABs) | 12 |
| Number of used Input/Output Elements (IOEs)<br>bidirectional<br>input only<br>output only | 357 (88%)<br>16<br>161<br>180 |
| Maximum clock frequency | 16,4 MHz |

*Table D-2:* Data sequencer implementation details.

| | |
|---|---|
| Number of parallel memory banks | 2 |
| Maximum address range | $2^{16}$ x $2^{16}$ |
| Maximum number of concurrent tasks | 16 |
| Video scans memory capacity | 64 |
| Scan window memory capacity | 16 scan windows |

*Table D-3:* Data sequencer details.

### D.3.2.2 The Burst Control Unit

This chapter will give a technical description of the hardware components used for the memory system of the MoM-PDA. A description of the MDRAM functionality can be

| | |
|---|---|
| External offset RAM (look-up table size) | 32k Byte |
| External rALU address RAM | 32k Byte |
| Maximum burst length | 31 words |
| Number of entries per scan window | 512 |

*Table D-4:* Scan window generator details.

found in appendix C. For a more detailed description especially with regard to timing values, package measurements and maximum ratings refer to the MDRAM data sheet [Sie97]. MDRAM components are available in different memory sizes and clock frequencies. In the MoM-PDA, a MDRAM type with a memory size of 1 MB (= 256k Words) and a clock frequency of 120MHz is utilized. A photo of this component is shown in figure D-20.

Since all MDRAM types use a supply voltage of 3.3V and the Xilinx FPGAs 5V, a 3.3V/5V transceiver is used for coupling these components. The transceiver is described in [Bed98].

The memory controller called Burst Control Unit (BCU) of the MoM-PDA is implemented using the Xilinx XC4000 FPGA family [Xil99]. For details of the implementation refer to [Bed98]. A floor plan of the MDRAM controller in the XC4013e FPGA is given in figure D-21 at page 288.

### D.3.2.3 The Reconfigurable ALU Port

The main task of the reconfigurable ALU Port (rAP) is to form an interface between the memory system and the rALU. Since MDRAMs use the same bus for address- and data transfers (see appendix C), the rAP is directly connected to the memory bus and must follow its restrictions.

The rAP is implemented with a Xilinx XC6200 FPGA (see figure D-22 at page 289 and [Xil96]). Its programmable space is partitioned into two functional units.

- One unit will be the parallel memory interface for the MDRAMs. This unit is the same for every application and is configured once at power up.

*Figure D-20:* SIEMENS MDRAM device.

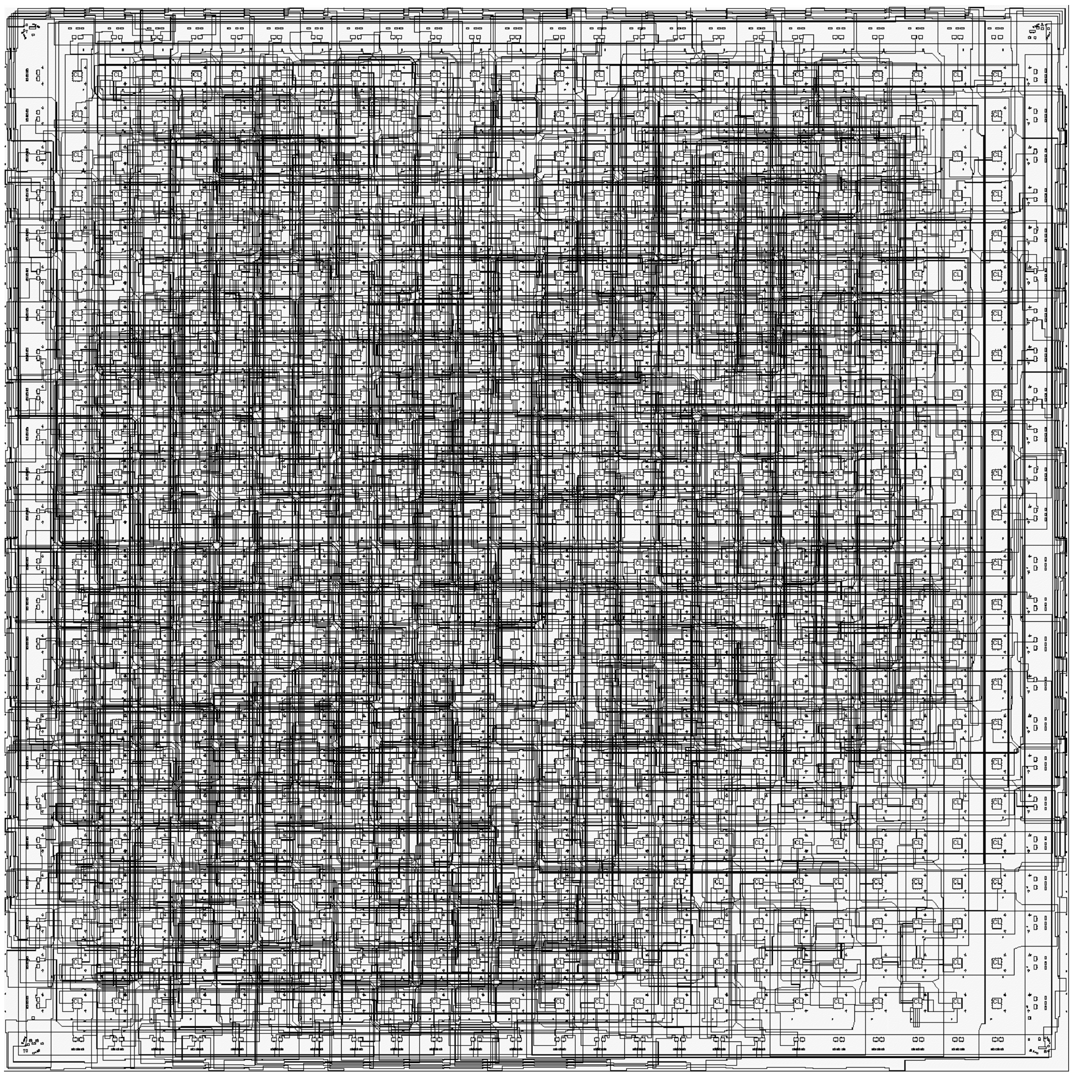

*Figure D-21:* Floor plan of the Burst Control Unit on the Xilinx XC4013e chip.

- The remaining programmable space may be used in two different ways:
  for calculations it is (re-)configured for every task. Computations are performed by applying the configured operations on the data. This allows to build a small version of the MoM-PDA without the KressArray. Simple problems may be computed with the XC6216 only. Therefore an operator library has been implemented, mainly containing image processing applications (see [Gil98] and [HHG98]).
  as a connection to KressArray: In that operation mode the XC6216 optimizes data exchange between the KressArray and the data memory and implements some glue logic.

a)

b)

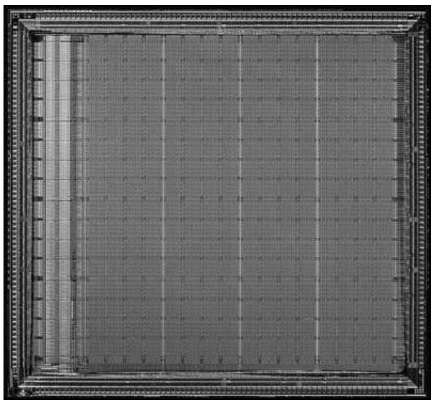

*Figure D-22:* Xilinx XC6216 FPGA:
(a) device, and
(b) die photograph [MV97].

## The Data Transmission Protocol between rALU and Burst Control Unit

Any transmission between rALU and MDRAM is performed over a 16-bit bus using handshake signals. The MDRAM has an internal 32-bit memory organization but data words are split into two 16-bit words. The first half-word is read/written with the rising clock edge, the second half-word with the following falling clock edge of the system clock (*SYSCLK*, see figure D-23). For a detailed description of the interface refer to [Bed98].

Unfortunately the XC6200 FPGA [Xil96] is not capable of clocking flipflops with the rising edge and the falling edge. In addition, inverting the SYSCLK signal will lead to clock-skew problems. Therefore, the rAP is clocked by *DCLK*, a clock signal with the double frequency of *SYSCLK*. The rising clock edge of *DCLK* is used to sample the read-/write signal (*RS/WS*) and the data (*DB*).

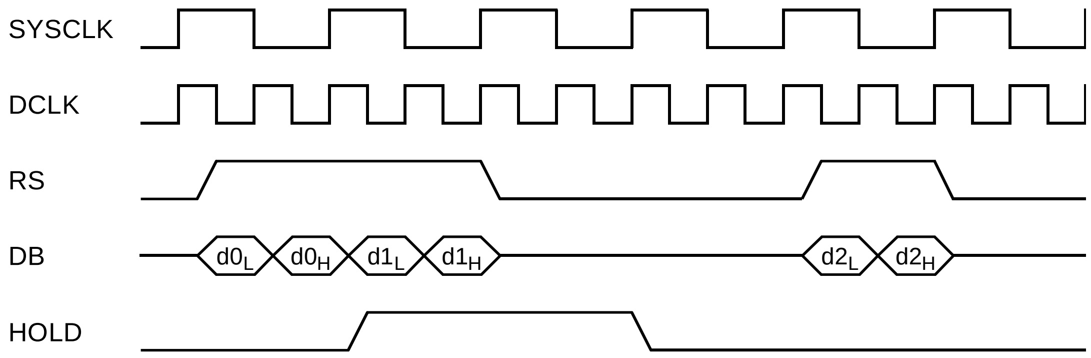

*Figure D-23:* Timing of an rALU-interrupted read burst.

In some situations the rALU may be not fast enough to proceed all data in the speed required by the memory system. In that case the rAP must interrupt the data transmission. An interrupted read transmission is shown in figure D-23 at page 289. The rAP must sample *RS* and *DB* every rising edge of *DCLK*. If *RS* is active, a transmission is in progress. A 0/1 counter is used to distinct between the lower halfword and the upper halfword. A write transmission is illustrated in figure D-24. If the rAP samples *WS* active on the rising edge of *DCLK*, it must drive *DB* in the following two clock cycles for a complete 32-bit transfer. In both cases the rALU can assign *HOLD* to interrupt the transfer, but the actual transmission must be finished because the Burst-Control-Unit samples *HOLD* every rising edge of *SYSCLK*.

To perform parallel data access two interfaces, as described above, are integrated into the rAP.

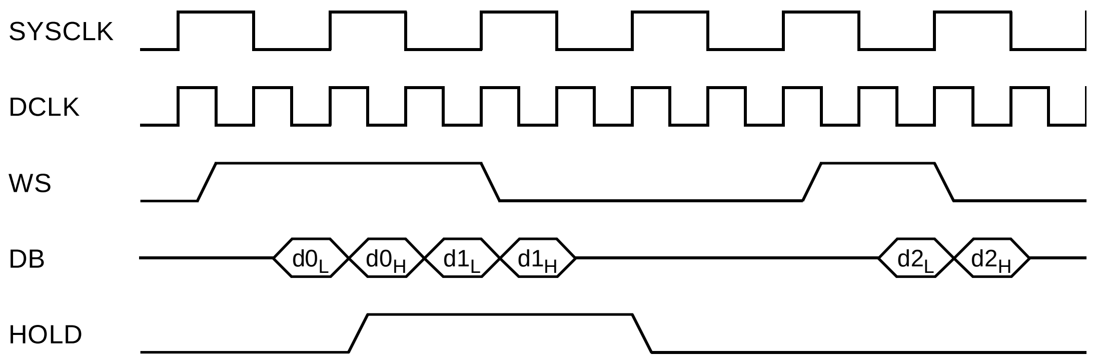

*Figure D-24:* Timing of an rALU-interrupted write burst.

## Structured Overview of the MoM-PDA rALU Implementation

The MoM-PDA rALU consists of two parts:

- the application independent MDRAM-Interface, and
- the implementation of the application in the reconfigurable datapath.

The application dependent part of the rALU may be implemented in the KressArray (section D.3.3 at page 293) or in the rAP. Implementing applications directly in the rAP is only possible for rather simple tasks. Some example applications have been presented in [Gil98] and [HHG98]. Figure D-25 at page 291 shows the rALU organization for an application implemented in the rAP only. If an application is implemented in the KressArray the rAP implements besides the MDRAM interface some glue logic to interface the KressArray. A Pipe Control Unit is not needed in that case.

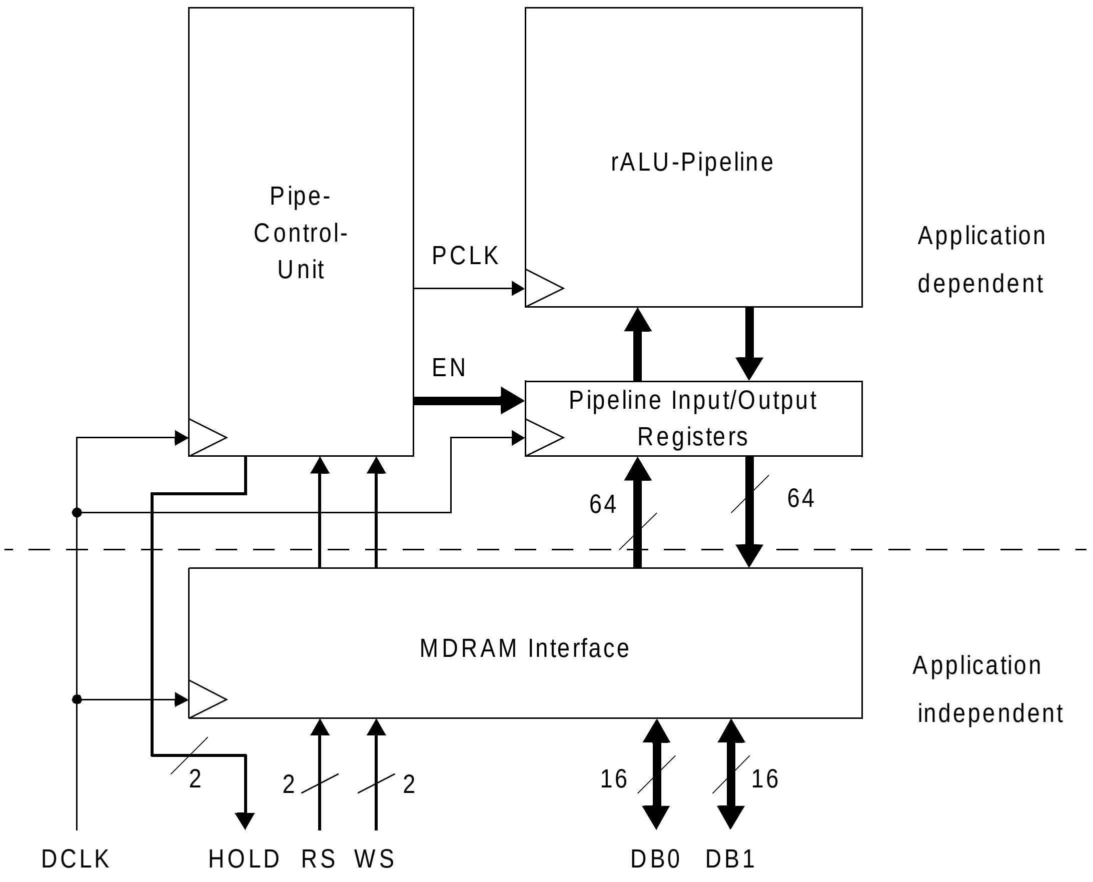

*Figure D-25:* The MoM-PDA rALU organization.

## The MDRAM Interface

The MDRAM-Interface (figure D-26 at page 292) is an application independent device. Its major function is to distribute a maximum of four 16 bit transfers initiated by the Burst Control Unit every *SYSCLK*-cycle (two *DCLK*-cycles) to 64 input-flipflops for rALU-Read operations and to switch 64 output-flipflops to four 16 bit transfers for rALU-Write operations. The ReadStrobe (*RSx*) and WriteStrobe (*WSx*) signals distinct between read- and write-operations to or from the two MDRAM banks.

The MDRAM interface control must select between two 16-bit registers to form the rALU-Write operation and distribute the rALU-Read transmission to the input registers. After a rALU-Write transmission is completed, a new set of values is loaded into the output registers.

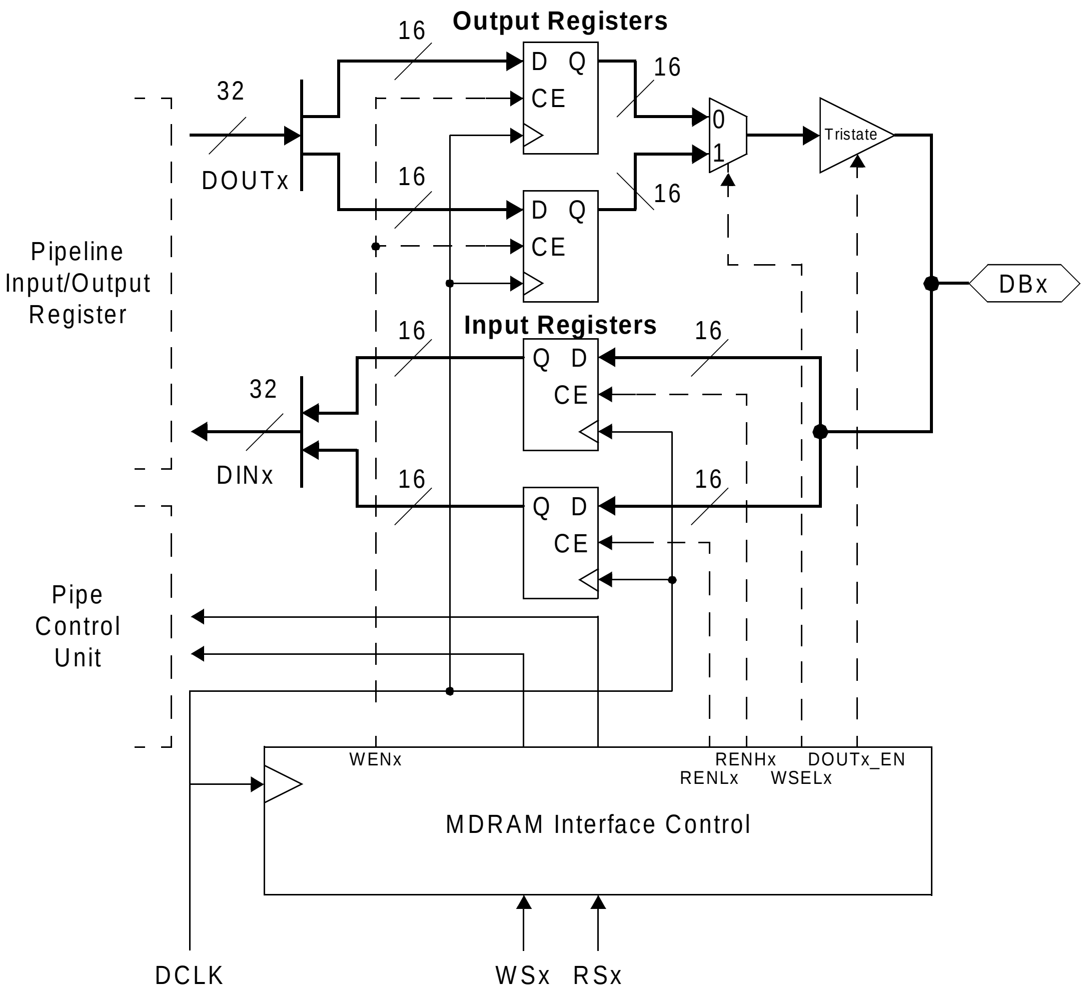

*Figure D-26:* MDRAM interface diagram.

## D.3.3 The KressArray Emulator

Figure D-27 shows the KressArray emulator [Zim99]. It can be configured to implement different KressArray architectures and is especially designed for testing of applications. In the MoM-PDA environment it may be used to perform computations.

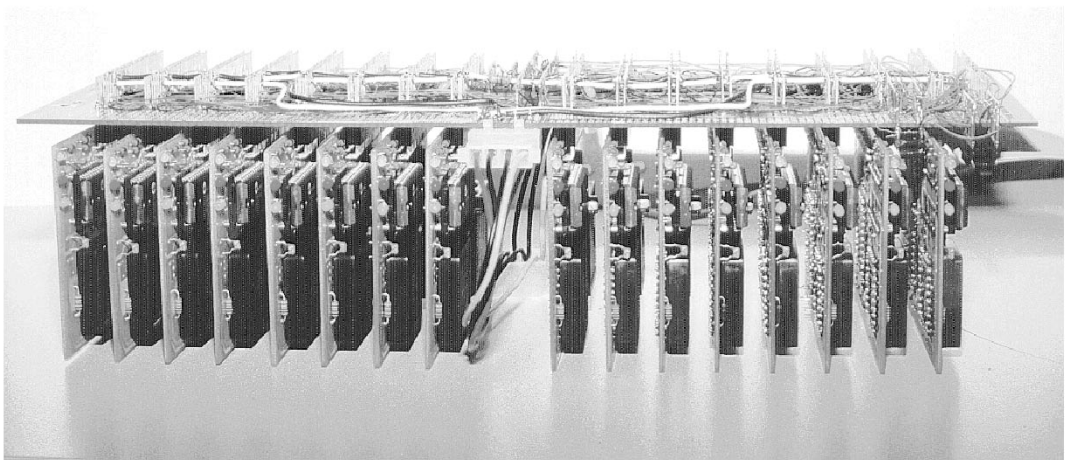

*Figure D-27:* The KressArray emulator.