Autonomous Memory Blocks: Scalable Structured Data Access for Reconfigurable Computing

Wim J. C. Melis[1]*, Peter Y. K. Cheung[1], Wayne Luk[2]

[1]Department of Electrical & Electronic Engineering
Imperial College, Exhibition Road
London SW7 2BT, England

[2]Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ, England

Abstract

Many hardware designs, especially those for signal and image processing, involve structured data access such as queues, stacks and stripes. This paper presents parametric descriptions as abstractions for such structured data access, and explains how these abstractions can be supported either as FPGA libraries targeting existing reconfigurable devices, or as dedicated logic implementations known as Autonomous Memory Blocks (AMBs). Scalable architectures combining the address generation logic in AMBs to provide larger storage with parallel data access are also examined. The effectiveness of this approach is illustrated with size and performance estimates for library based and dedicated implementations. It is shown that for two-dimensional filtering, using a dedicated AMB can result in an area improvement of up to 10 and a speed improvement of up to twice in comparison to using libraries.

1 Introduction

The evolutionary development of FPGA and reconfigurable logic devices shows a gradual drift from relatively fine grained architectures, mostly based around LookUp Tables (LUTs), to ones that include dedicated hardware blocks such as multipliers, microprocessors and block memories. Although the latest offerings from Altera and Xilinx also contain embedded memory, their firm focus of development has always been on improving their processing capabilities.

DSP algorithms with their inherent parallelism are frequently mapped to FPGAs instead of conventional DSP processors. However, existing FPGA devices have one major weakness when compared with DSP processors. Almost all DSP processors provide one or more ALUs in order to improve parallelism. While parallel address calculations on FPGA can easily be achieved, current architectures require this to be done in the reconfigurable logic cells. This is not only wasteful of reconfigurable logic resources, it also uses up valuable routing resources. Given that most DSP algorithms access memory in a structured manner, one approach is to integrate a configurable Address Generation Unit (AGU) with the FPGA's block memory to form an Autonomous Memory Block (AMB). The advantages and drawbacks of such an extension to existing FPGA architectures are assessed within this paper. This memory extension raises its level of abstraction, but only if this extra hardware is present. These abstractions can be used independent of the underlying architecture when they are made available as a library of access modes.

The memory blocks within normal FPGA devices are generally rather small in comparison with the memory available on DSP systems. Since the address generation unit can now be hardwired to a single memory block, methods of concatenating these memory blocks with AGU are investigated. These smaller blocks with local address generation also offer a better flexibility and can allow higher degrees of parallelism than what is often possible on conventional DSP platforms.

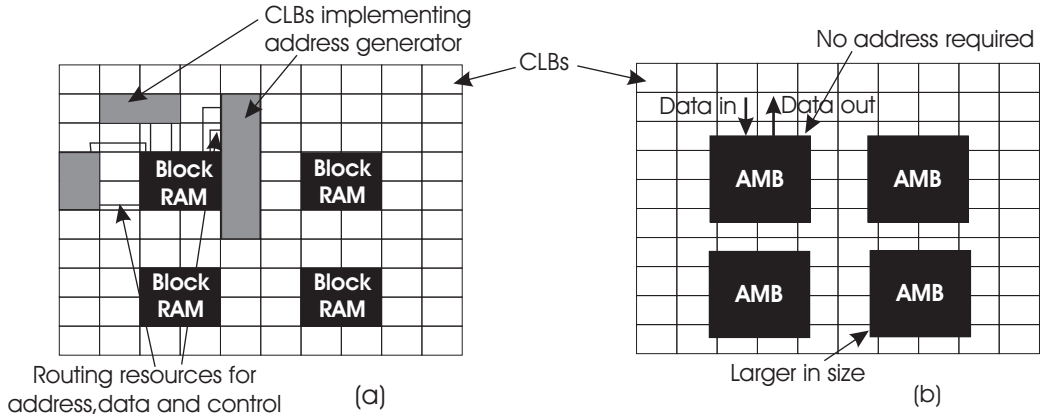


Figure 1: (a) Existing FPGA block RAM requiring external address generation circuit; (b) Proposed Autonomous Memory Block that includes the Address Generation Unit (AGU)

The contributions of this paper are: 1) abstractions for the designer by specifying the available modes of AMB, their implementation and the parameters required for implementation; 2) concatenation of these structured data access memory blocks with their local address generation units; 3) estimation of efficiency of adding these address generation units to memory and evaluate the performance and size of concatenating blocks using either FPGA resources or the dedicated logic of the Autonomous Memory Blocks.

The paper is organised as follows: the current set of modes for structured data access and their respective set of parameters are discussed in Section 2. Section 3 then describes the combination of several AMBs to provide larger buffering and/or higher degrees of parallelism depending on the application requirements. Followed by a discussion of the effectiveness and limitations of the design abstractions and their scalability in Section 4. Section 5 discusses related work after which the paper is concluded in Section 6.

2 The Autonomous Memory Block (AMB)

The latest FPGAs provide embedded block memory that can be configured for single or dual port access, and with a variety of data widths. The block RAMs are distributed on the FPGA amidst Configurable Logic Blocks (CLBs). One issue that limits the usability of these block RAMs is the routing resources required. For example, to implement a dual-port 2Kx9 bit FIFO on a Virtex-II Pro device, one requires 22 address wires and 18 data wires to be routed to and from the block RAM. Furthermore, counters and control circuits necessary to implement the FIFO must be implemented in the surrounding CLBs, using up logic resources and adding to the routing congestion problem (see Figure 1 (a)). In the proposed Autonomous Memory Block (AMB), a full-custom Address Generation Unit (AGU), which is highly configurable, is added to the embedded block RAM (see Figure 1 (b)). This would inevitably increase the size of the block memory. However, provided that the AGU is properly designed, the need for external address generation circuits for most DSP applications is eliminated.

Once configured, the AMB would operate autonomously without further supply of addresses. In order for such autonomous memory to be effective, the AGU must be designed to handle the most common data access models found in DSP algorithms. This set of modes for structured access can be provided as dedicated address generation units combined with the memory or as a library of modules for inclusion during synthesis. AMB is designed to provide the following modes of operation:

1) *First-In-First-Out (FIFO) Circular buffer* sends the first received data out first. This mode is particularly useful when data read and write rates are different. To avoid data-loss, flow control is implemented to detect and handle over- and underflow conditions. Figure 2 shows a simplified block diagram of the AGU implementing the FIFO buffer mode. The read and write counters keep track of the memory addresses for retrieving and storing data respectively. The `Data_count` counter stores the number of data items currently in the FIFO. The Finite State Machine (FSM) provides the necessary control functions to the counters and causes the ”wrap-around” behaviour in the address counters, necessary to implement a circular buffer.

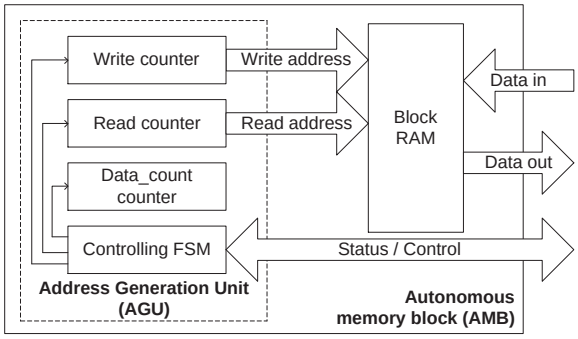


Figure 2: Address Generation Unit (AGU) in FIFO mode

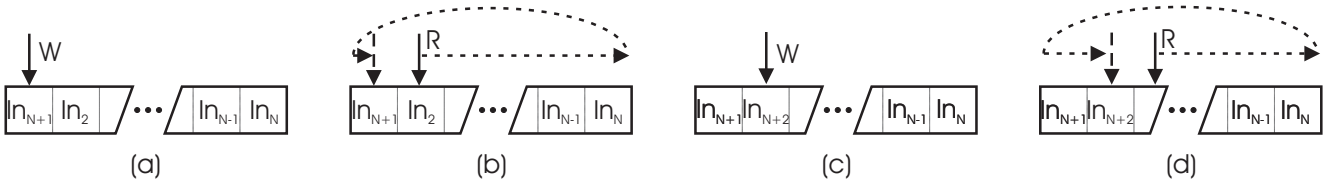


Figure 3: Indication of FIMO mode operation: overwriting of oldest item (a), followed by reading the full buffer (b), and again overwriting the oldest item (c) and reading the full buffer (d)

2) *First-In-Multiple-Out (FIMO)* is an extension to FIFO mode, where for each data item written the full set of previously stored data items are accessed in sequence (see Figure 3). The buffer is filled with items In_1 to In_N with N being the size of the buffer. This initialization can occur at configuration time or by the design itself. As soon as all items are written, the full set of items are read. Subsequently, the write pointer overwrites the oldest item (in this case In_1) with a new item In_{N+1} (Figure 3 (a)) and reading starts from the next item up to the most recently written item (Figure 3 (b)). This sequence of overwriting the oldest item and subsequent reading repeats (Figure 3 (c) and (d)). The output of a particular reading access (Out_j) can therefore be specified as in Equation 1. The specific features of this access pattern make it particularly useful for 1-D convolution, matching and filtering operations. The implementation of this mode is almost the same as that of the FIFO except that the FSM is slightly modified.

$$Out_j = In_k \tag{1}$$

with: $k = \lfloor \frac{j}{N} \rfloor + j \ mod \ N$

3) *Last-In-First-Out (LIFO)* mode, or stack-based store. This type of access mode is useful for image flipping operations, or in combination with the processors currently available on reconfigurable devices. The AGU implementation for the LIFO is almost the same as that of the FIFO except that the read and write counters can move in both directions.

4) *Swinging Buffer* mode - in this mode, the AGU maintains two identically sized buffers. While one is being filled (i.e. storing data), the other is being processed (i.e. retrieving data). When the write buffer is full and the read buffer is exhausted, the two buffers are automatically swapped over. This mode of data access is particularly useful for block data processing (such as FFT) where the DSP algorithm is applied to a block of data while new data is continuously being acquired. The AGU circuit to implement this mode is again very similar to that for the FIFO. The two counters are used to reference the two buffers. The FSM is slightly modified to implement the "swinging" action.

5) *"Striped" Access* mode - in this mode the writing and reading of the data can be "striped" in any manner as shown in Figure 4. In Figure 4 (a) `n` data items are accessed in sequence starting from address `A1` (forming the first stripe). The address is then adjusted to `A2`, which has a fixed offset from `A1`, and the second stripe begins. Figure 4 (b) presents a similar idea but applied to 2-D data stored as raster scan image. This access mode is particularly useful in processing data in overlapping windows or in image/video data, such as 2-D convolution, block matching and 2-D filtering. The implementation of this mode is again based on the same underlying structure as of the FIFO mode, but now the counters need to have an adaptable

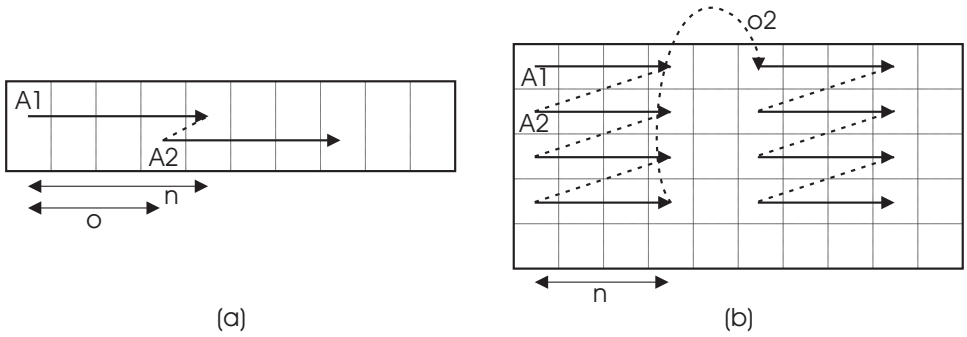


Figure 4: (a) 1-D "Striped" access mode; (b) 2-D "Striped" access mode

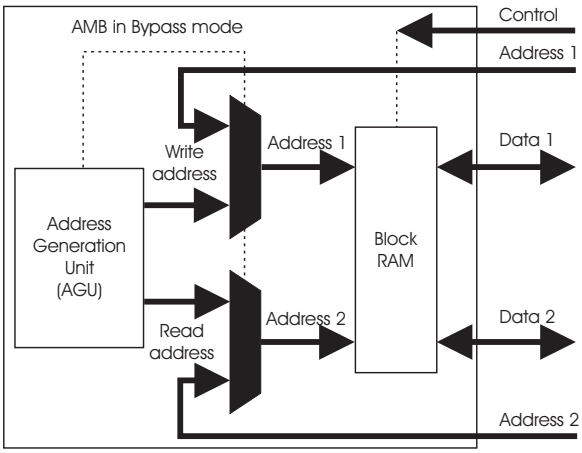


Figure 5: AMB with bypass mode

increase/decrease value and extra logic is required to control the number of additions/subtractions of a particular size.

In addition to these 5 different access modes, AMB can also be configured in the random access mode as shown in Figure 5, bypassing the AGU altogether. This mode is necessary to handle data access models not currently provided by the AGU design. However in many of these cases, only reading from memory is irregular, the data can still be written in a structured way, therefore it is possible to combine this bypass with the striped access mode, to allow for structured writing and then switch to random access for the reading.

To make use of these abstracted memory modes within a design environment, a certain set of parameters would need to be specified. These parameters are:

- number of required memory locations
- size of each memory location and whether this consists of separate items or not. For example in image processing, when the image is stored as RGBα values, then 32 bits would be required for each pixel. This could be stored as a single 32-bit value or 4 8-bit values. The latter case might be better when data is processed by byte, in which case storing the data closer to the processing node can result in shorter access times.
- requirement for over- and/or underflow detection
- a certain set of events encountered by the address generation logic can be signaled to the rest of the circuit. This is currently limited to reaching certain boundaries of the address space, but could be extended later on.

All of these parameters are required independent of the mode of operation. In case of striped access mode, the parameters for the stripe/jump also need to be specified. Besides these basic parameters, some extra information on the performed

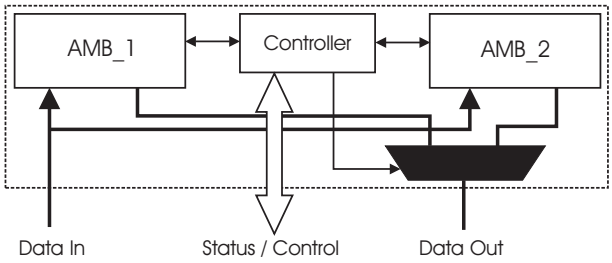


Figure 6: Concatenation of two AMBs to provide larger buffering in the case of FIFO, FIMO and Stack mode

processing is required to optimise for the level and type of parallelism. When required by the application, the mode of operation can be changed at run-time, in which case the parameters of the other mode will need to be provided when the change needs to take place. This run-time reconfiguration could for example be useful a striped access pattern needs to change because of a change of window size.

3 Scalable Architectures for Structured Access

The memory size of an AMB built around current FPGA embedded memory blocks is fairly small in comparison to the amount of memory commonly used in DSP environments. This requires multiple AMBs to be chaineable to form larger memories. Using multiple AMBs also offers the possibility of providing better concurrent data access and increased parallelism.

3.1 FIFO, FIMO and Stack

Combining multiple AMBs in FIFO, FIMO or Stack mode to form a larger circular buffer is relatively straightforward as shown in Figure 6. This diagram shows the merging of only two AMBs. A multiplexer is used to route the output from either AMB_1 or AMB_2 to Data_Out. A controller selects either AMB_1 or AMB_2 to be active at any one time. Since the implementation of this controller is application dependent, it is best implemented in reconfigurable logic instead of having it built into the AMB. This allows the controller logic to be closely coupled with the overall design control. Previously the design's control could be based on the address counters, which are now within the memory. Therefore the AGUs have to be extended to signal a certain set of events, for example when reaching the end of the address range. These events are not only useful to the controlling memory wrapper, but can also be used as a basis for the design's control. A completely built in controller would not only make the design's control harder, but would also incur unnecessary duplication.

FIFO, FIMO and Stack mode are essentially sequential in that data must be written and read in strict order. Therefore it is impossible to exploit the fact that multiple AMBs offer the possibility of concurrent data streams.

3.2 Swinging Buffer

The swinging buffer mode can be used in cases where one set of data is being retrieved, while another set is used for processing. Since this mode makes use of two or more identical size buffers, when combining AMBs to provide a larger swinging buffer, there are two possibilities: 1) the individual AMBs are configured as FIFOs or 2) each of the individual AMBs is used in swinging buffer mode. The first approach allows for easy swinging and/or parallelism between an odd number of memory blocks. For example three memory blocks are combined of which one is being written and 2 are read. The block for writing is alternated and this is taken care of by the overall controlling memory wrapper. In this case any parallelism among accessing the different blocks can easily be performed.

The second approach configures all AMBs into swinging buffer mode, which will reduce the size of the overall memory controller. This method of implementation ensures that the actual data block is split among different physical memories. Most algorithms which consist of block based processing involve a certain amount of parallelism. The multiple accesses for data of the same block can consequently be ensured because the data is physically split among different memories. This might however require well considered data splitting to prevent contention during access.

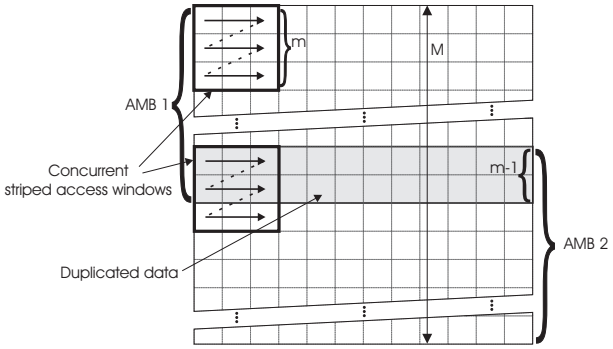


Figure 7: Data splitting for striped access with duplication

3.3 Striped Access

In this mode, the writing and reading of data can form a striped or Z-shape pattern as shown in Figure 4. This multiple level Z-shape data access is particularly useful for 2-D convolution, 2-D filtering and full search motion vector estimation.

Striped access mode offers the best opportunity for exploiting multiple data access, when using multiple AMBs for large buffer storage. The simple concatenation scheme shown in Figure 6 for the previous modes does not apply here because the Z-shape access pattern requires overly complex interfacing between the different AGUs to work cooperatively with each other. Instead, in striped access mode, it is assumed that each port of the AMB is supplying (or receiving) data to its own processing element(s). This immediately raises the question: how should data be split between multiple AMBs so that: a) it provides the necessary parallel accesses and b) it is easily implemented across multiple AMBs.

Three different ways of splitting the data between multiple AMBs are presented here. For the sake of clarity in explanation, we assume the combining of only two AMBs.

3.3.1 Data splitting with duplication

Figure 7 shows how a 2-D image frame may be stored in two AMBs. The shaded area represents the area of overlap where data must be duplicated in both AMBs. Assuming that each AMB has two memory ports, this method allows four-way parallelism. Obviously if the frame height M is large when compared to the window height m, the overhead incurred by the duplicated data is smaller. Otherwise, this method may waste storage resources unnecessarily up to the limit of full data duplication, which could render this approach inefficient.

3.3.2 Interleaved data storage

Figure 8 shows how a 2-D image frame is stored in an interleaved manner in two AMBs. Again, assuming dual port access of each AMB, this arrangement provides four-way parallelism, but without incurring any unnecessary data duplication. The current design of the AMB can handle the striped access sequence shown in Figure 8 with no modifications required. However, this implies a certain set of constraints on the movement of the windows and the interleaving pattern. The most basic constraint is that the processing window width n is an integral multiple of the interleave width k (see Figure 8). Each of these constraints is related to the striped access pattern that can be handled by the AGU. Otherwise the AGU would require substantial changes. The increased complexity renders this approach impractical. In comparison with the method using duplication, this method is limited to parallelism within the same processing window, whereas when duplication is used multiple windows can also be processed simultaneously.

3.3.3 FIFO and shift register for striped access

Image processing architectures quite often use shift registers to buffer one or more lines of the currently processed image frame. This can be adapted to work with AMBs and provide a striped access pattern as shown in Figure 9.

Each AMB, together with a small shift register, is operating in the FIFO mode and serves as a line buffer. Multiple lines can be concatenated together to provide effective striped data access. The number of lines stored should equal the

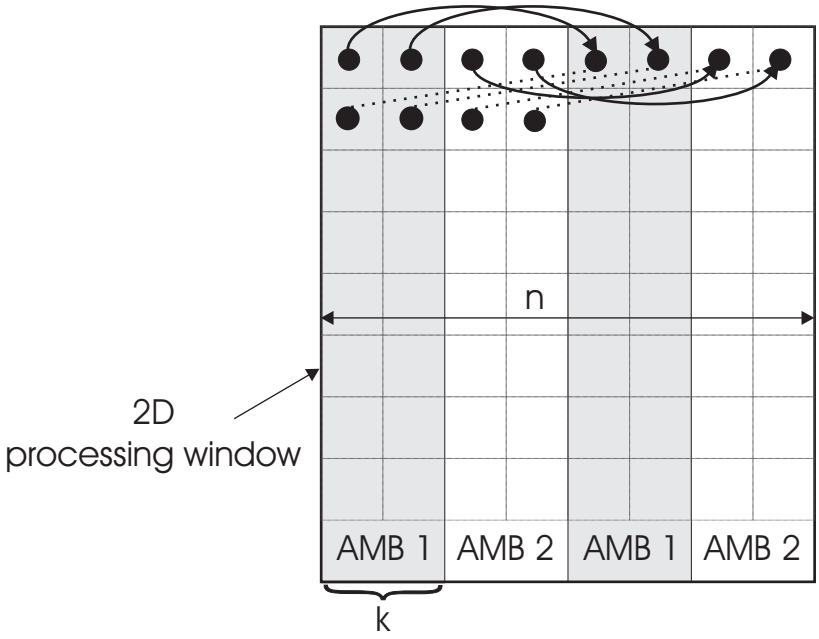


Figure 8: Data splitting for parallelism without duplication

height of the processing window as this makes the filling of memory easier while preventing the need for collecting the same items multiple times from external memory. On the other hand, the number of shift registers in each line is the degree of parallel access offered to concurrent processing elements. The advantage of this approach is that the shift registers allow both parallelism and maximise data reuse, while the AMB-based FIFO provides efficient, low energy storage. This scheme also automatically eliminates superfluous data after processing.

4 Results

This section evaluates the structured address generation modes discussed in Section 3 towards area and performance for a dedicated as well as library based approach. The second part of this section investigates the efficiency of concatenation after which it presents results for an application in which these structured data access blocks are combined. Again the dedicated and reconfigurable approach are considered.

4.1 Experimental Setup

In order to evaluate the advantages of using AMB, a VHDL implementation of an AGU with all of the discussed modes was created. This code was then synthesised with Synopsys' Design Compiler towards a custom circuit targeting a typical 0.13μ m process. The area and speed figures of this dedicated implementation were then taken from the Synopsys' Design Compiler report.

The implementations targeting the reconfigurable hardware are based on the same code supporting all modes. The designs were synthesised for Xilinx's Virtex-II XC2V8000 FPGA using Synplicity's Synplify software. The optimized design is then placed and routed using Xilinx backend tools, which provided both the logic cell usage and the estimated speed of operation. The actual area used is estimated by assuming that the XC2V8000 die area is almost 2.5cm X 2.5 cm. Such an estimate should provide area usage for both logic cells and routing resources.

4.2 Performance and Area for Dedicated and Reconfigurable Implementations

A first measure of comparison is the performance and area advantage of the dedicated AGU in comparison to using reconfigurable resources. Table 1 presents results for a 9-bit AGU with either all modes (dedicated implementation) or only

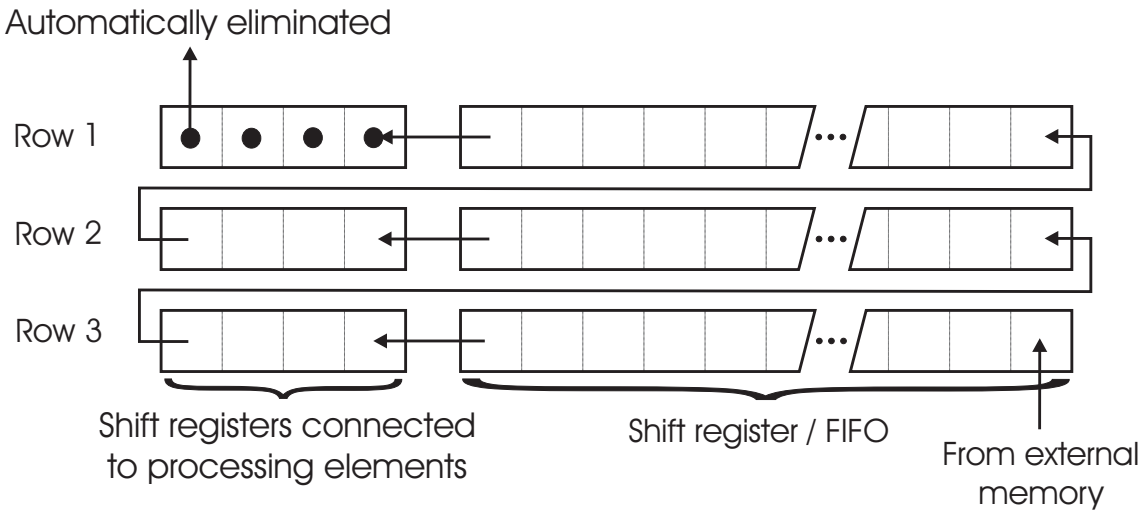


Figure 9: Striped access mode for larger data arrays by shifting data through registers in combination with FIFOs

a limited set of modes. The 9-bit address width is related to the 512x36-bit configuration of the block SelectRAM in the targeted device. It can be noticed that for the simplest mode an improvement of a factor of 4 for area and a factor of 1.09 for performance is achieved. As the design becomes larger, the area and performance advantages become more significant. Both area and speed improvements should even be higher if the AGU were designed as a handcrafted full-custom circuit instead of synthesised with Design Compiler. Area and performance results for a library based approach are presented in Table 2. Also this table indicates a nearly linear inverse relation between the size of the design and its performance.

By using AMB compared to the conventional memory blocks a certain percentage of CLBs would be saved. This percentage is shown in Figure 3 and is based on an 9-bit address for *each* memory block of the device operating in either FIFO or Striped Access mode. The percentages are obviously larger when the number of CLBs is small, but even for the largest devices there is a significant saving.

Even though a dedicated AGU has more advantages than using reconfigurable resources, it is questionable from which memory size the overhead for this address generator becomes worthwhile. Figure 10 shows an estimate of the area of the AMB, in terms of gate count, as a function of address width. The gate count of the AGU is directly derived from the Design Compiler report. The gate count for memory is estimated by assuming that each dual-port memory bit has 8 transistors (or 2 gates). The address decoder circuit is ignored in this estimation. Since the size of AGU increases approximately linearly with address width, while the memory area increases exponentially with address width, it is not surprising that beyond 8 or 10 bit of address, the area of memory dominates. The situation is however not as straightforward as Figure 10 suggests. Consider the case of the three different types of block RAM in the Stratix FPGA [1]. Table 4 illustrates that if the AGU are designed to cope with all the possible memory organizations down to 1-bit wide data path, the AGU could use more gates than the memory array itself. As can be noticed in Table 4, the AGU is significantly larger than the actual memory size for the smallest block. For the medium size memory block (M4k) in Stratix, the AGU is already close to half the memory size. The largest embedded memory blocks MRAM, have a relatively insignificant size for the address generation unit in comparison to the memory size, partly because no bit-width configuration is possible.

When the dedicated AGU is combined with memory, a multiplexer is inserted to use the memory in random access. Consequently, a small delay is added to the address signals when they are delivered from the application itself. The gates on this path of the synthesised design was found to consist of an inverter, an inverting 2-input multiplexer and a tristate buffer. The delay will therefore be reasonably small, especially when compared to some of the delays introduced by the reconfigurable routing resources.

4.3 Evaluation of AMB Concatenation

As mentioned previously, a certain set of events need to be signaled to allow concatenation of AMBs. For the simpler modes, these signals give an indication of reaching the end of the address space for that particular buffer. For striped access,

Table 1: Area and speed for a 9-bit dedicated AGU in comparison to the use of reconfigurable resources for address generation

	Area ($10^{-18}m^2$)	Area Factor	Performance (MHz)	Performance Factor
AGU as dedicated	110265	1	299	1
FIFO design on Virtex-II	420330	4	271	0.91
Striped access design on Virtex-II	2021291	18	113	0.38
AGU on Virtex-II	7207418	65	80	0.27

Table 2: Area and performance for library modules with 9-bit address-width

Mode of operation	Area (# LUT)	Performance (MHz)
FIFO	68	271
FIMO	78	295
Stack	283	159
Swinging buffer	80	172
Striped access	327	113
Hybrid mode	176	98

Table 3: Percentage of CLBs saved, depending on memory access mode and size of Xilinx Virtex-II device

	Xilinx Virtex-II Device		
Using all memory blocks	XC2V40	XC2V1500	XC2V8000
as FIFO	42 %	17 %	10 %
for striped access	-	97 %	56 %

Table 4: Size of AGU for the different types of memory in Stratix

Memory Type	Number of address bits for AGU	AGU size (gates)	Memory size (gates)
M512	9	3471	1024
M4k	12	4560	8192
MRAM	16	6131	1048576

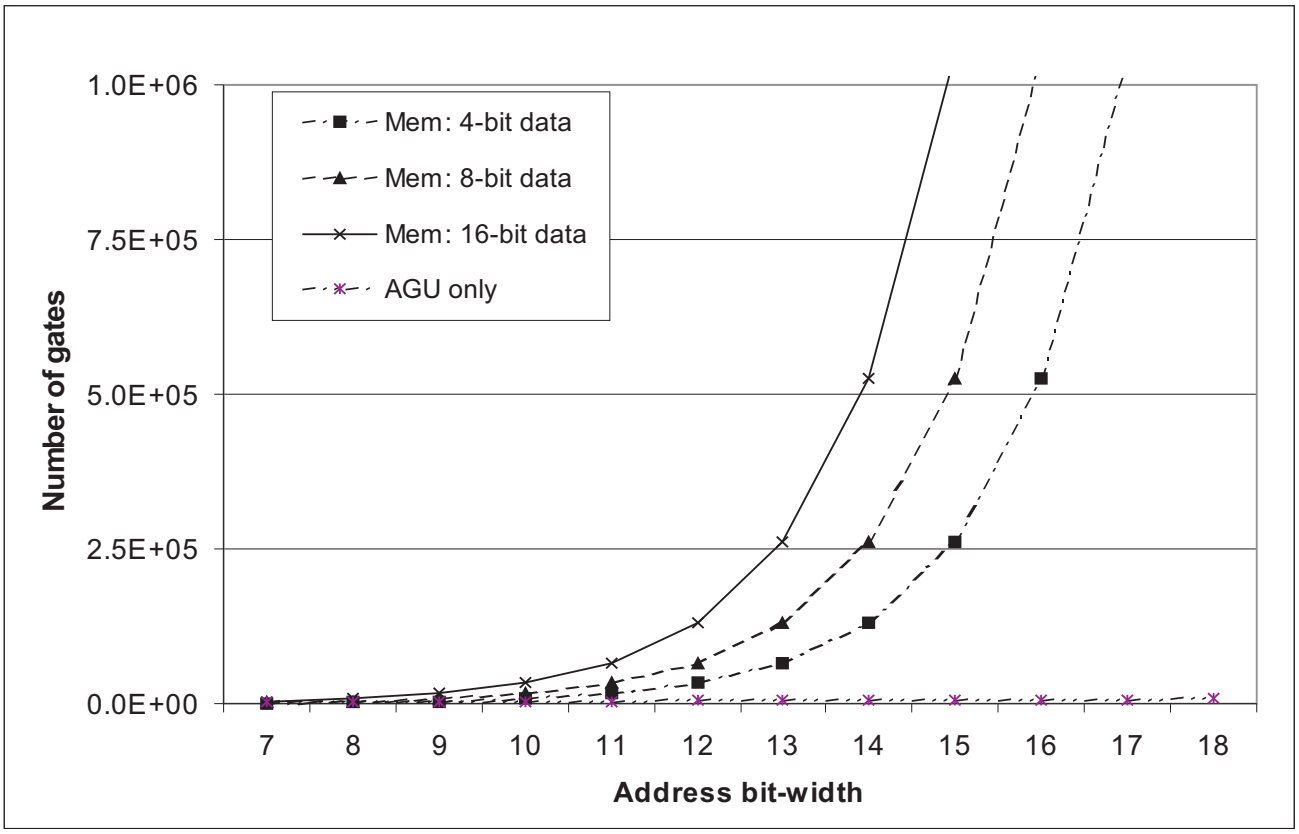


Figure 10: Gate count for memory and AGU as a function of address-width

these signals mark the different jump-sizes. This event signaling influences size and performance of the dedicated AGU implementation only to a minor extent as shown in Table 5. The library based approaches show a similar increases of size.

Furthermore, a 2-D filtering example is worked out to compare the different concatenation approaches for striped access. The size of the filter coefficient-window is 3 by 3 and the image size 720 by 576. The coefficient window is accessed by three parallel processors in a striped way. Results for the different approaches using only reconfigurable resources or assuming a dedicated AMB are presented in Table 6. In some approaches dedicated logic is combined with reconfigurable implementation, this requires for the number of LUTs to be converted to an equivalent dedicated area number, which is performed by considering the percentage of the total number of LUTs as a measure for the percentage of the used total chip area.

To apply the interleaving approach each row or column of the processing window has to be located in a new memory block, this is essential due to the movement of the processing window. In this implementation, data are interleaved row-wise

Table 5: Difference in area and performance for the dedicated normal AMB version and the one that allows concatenation

	Area ($10^{-18}m^2$)	Maximum Performance (MHz)
Normal AMB version	110265	299
AMB extended with event signaling	114132	280

and each row is stored in a different memory block. As soon as the processing window has moved to the right, redundant items are overwritten with new data. Since the parallel processors operate on one window at a time, there is no need for multiple ports on the memory block that buffers the calculated pixels.

In relation to the processing window size, the duplication approach requires an overlap area with a width/height of 2. This method implies that the parallel processors are spread along the complete width of the image and only one processor is assigned to a memory block since new data can then easily be appended. If splitting is done column-wise and new rows are appended, the maximum capacity of a memory block in the Virtex-II is reached after a small number of lines. Therefore, a second set of memory blocks is introduced that overlaps row-wise with the first set. Data are written simultaneously to these overlapping regions. Due to the parallel processing of multiple windows, the output buffer is now required to handle this concurrency. Since a result only becomes available every x cycles the writing of data can be delayed until a memory port is free.

The last approach combines FIFOs and shift registers for striped access. In this particular example, the number of shift registers needed equals 3 since this is the level of parallelism. The parallelism is again performed on different windows, but now the windows are consecutive. The number of lines to be stored is equal to the processing window-height, being 3. This results in the use of 3 FIFOs, each containing 717 items ($=$ line length $-$ number of shift registers). An extra FIFO was added to temporarily buffer the results, similar to the other implementations.

To conclude, the area and performance factors indicated in Table 6 make a comparison between reconfigurable and dedicated implementations. Therefore these numbers not only indicate the advantage of these different approaches but also of using a dedicated AGU. The differences in performance and area for each of the implementations is solely caused by the actual address generation and control logic, since the processing is implemented identically in each of the approaches. When considering only the dedicated implementations, the area difference between using FIFO and interleaved storage indicates a factor of (± 4) since the FIFO based approach has a simpler addressing and less complex overall controller. This simpler addressing mode also results in a higher performance for the FIFO based approach. The area difference between the first and second approach is then again due to the requirement of nearly twice the number of memory blocks, consequently doubling the necessary amount of address generation logic and control overhead. The performance of the duplication approach is however better than the interleaving approach. This is most likely due to the required switching between processor and storage memory for the interleaving approach. When the reconfigurable implementations are considered, a similar trend can be found between the different approaches. However, the performance of the second approach is now closer to the interleaving approach which would imply that in this case the actual address generation is limiting the performance. The area difference between the use of dedicated and reconfigurable approach are close to the well-known factor of 10 for the first two approaches. For the approach based around FIFOs and registers, the difference between dedicated and reconfigurable is much smaller. This is due to the dedicated approach offering a large set of modes, of which FIFO is the least complicated and therefore only a minor part of the full design. The performance difference between dedicated and reconfigurable approach indicates the suspected decrease of performance with an increased use of reconfigurable resources, unless these resources are more independent from one another as can be noticed from approach 2 in comparison to approach 1.

In conclusion, a dedicated AMB is most beneficial for applications which address a memory containing 4-bit or larger values stored in a memory of reasonable size, which is also the most commonly occurring situation. For cases where bits are stored in small memories, which is not that frequently occurring, a library based approach would be more beneficial since it only uses resources when absolutely required.

5 Related work

Several data centric architectures, sometimes classified as non Von Neumann architectures, have already been presented [2, 3]. Unfortunately these completely data driven architectures require a different way of thinking when implementing an application. In contrast to these architectures, this work retains the normal control driven computation but abstracts the data access from the actual computation. This is done by extending the memory with its own address generation unit and therefore make it more autonomous. A limited set of frequently occurring data access modes is provided as standard. Several of these blocks can then be combined to provide larger storage areas and/or to allow more parallelism.

Another area of related work where memory is made more autonomous is the family of Processing In Memory (PIM) systems [4, 5]. They form a parallel array in which each node combines a low complexity processor with a certain amount of memory. This approach is very useful in simple operations on large databases, however the benefits are limited to the amount of memory per node, and the complexity of the available processors.

Address Generation Units have been presented in several related fields, the best known example is the separate address

Table 6: Area and maximum performance for the implementation of a 2D filter (window size of 3 by 3) using the different striped access approaches. The third approach is used as a basis for comparing overall area and performance.

		Area				Performance			
		Dedicated ($10^{-18}m^2$)	Reconfigurable (# LUT)	Total ($10^{-18}m^2$)	Factor	Dedicated (MHz)	Reconfigurable (MHz)	Maximum (MHz)	Factor
Approach 1	Use of AMBs (dedicated AGU) with interleaving	528912	293	2340038	2.27	270	98	98	0.55
	Interleaving using reconfigurable resources	-	4879	30158654	29	-	64	64	0.36
Approach 2	Use of AMBs (dedicated AGU) with duplication area	925596	582	4523123	4.39	270	140	140	0.79
	Striped access with duplication area using reconfigurable resources	-	9118	56361264	55	-	71	71	0.4
Approach 3	Use of AMBs (dedicated AGU) as FIFOs in combination with Registers	528912	81	1029599	1	270	177	177	1
	Combining FIFO and Shift Register for Striped Access, using reconfigurable resources	-	292	1804945	1.75	-	156	156	0.88

and data ALUs in DSPs. This is due to DSP algorithms having repetitive and structured access, which recently lead to DSPs having a more specific set of modes instead of a fully flexible address generator [6, 7]. The streaming architectures presented in [8, 9] also contain address generators which help in storing or providing streams of data from and to the processing unit, for threads in a processing environment or streamed processing of video data respectively. The streaming data access model has also been proposed for reconfigurable devices [10], and a data stream manager for Handel-C was recently presented by Celoxica [11]. Unfortunately this data streaming model can be limiting for general DSP algorithms, since the data is not always streamed from one module to another with only one particular reading pattern.

Recent developments in FPGA architectures for DSP applications have all been focussing on improving the computational capabilities. Examples of these are Elixent's Reconfigurable Arithmetic Array [12], and the Field Functional Arrays [13]. However these advances in computational capabilities have not been accompanied by similar advances in data-flow. While small embedded memories are now found in most modern FPGA devices, these are general purpose blocks which require separate address generation. Adding an AGU to the embedded memory blocks can ease the data access, but to comply with real applications, these blocks will have to be concatenated.

6 Conclusions

The various modes of data access commonly found in DSP algorithms can easily be integrated into the block memory of FPGA devices. This has several benefits for reconfigurable devices. Firstly, the amount of required routing resources are reduced. Secondly, the area consumed by the AGU is considerably smaller and the speed of operation is faster than the implementation using Configurable Logic Blocks. Even though the AGU design is by no means optimal, it is illustrated that it could be worthwhile to invest time and effort in designing a handcrafted full-custom AGU that is tightly integrated with the block memory. Alternatively, the same set of structured access modes can be provided as a library for convenience, that can be mapped to different FPGA devices.

In addition, a detailed investigation of how these AMBs can be combined to either offer larger data storage and/or more parallelism is presented. It has been shown that the best concatenation method is in close relation to the actual data access, which is then again dependent on the type of parallelism. Each of the presented approaches however has its own limitations. Nevertheless, the current set of modes can easily be scaled to more memory blocks without the need for large amounts of extra resources.

Including AMBs into FPGAs raises the granularity of their functional blocks. Consequently they support a higher level of abstraction in the hardware implementation of DSP algorithms. However, the mapping and use of these AMBs from a high level description remains an unsolved problem which will be investigated in the future.

References

[1] Altera, "Stratix device handbook," 2003, `http://www.altera.com/literature/hb/stx/stratix_handbook.pdf`.

[2] R.W. Hartenstein, A.G. Hirschbiel, K. Schmidt, and M. Weber, "A novel paradigm of parallel computation and its use to implement simple high-performance hardware," *Future Generation Computer Systems 1990 International Conference on Information Technology (InfoJapan'90), 2-5 Oct. 1990*, vol. 7, no. 2-3, pp. 181–198, 1992.

[3] H. Voorhees, D. M. Fritzsche, and L. W. Tucker, "Exploiting data parallelism in vision on the connection machine system," in *Proceedings 10th International Conference on Pattern Recognition 1990*, Atlantic City, NJ, USA, 1990, pp. 617–622.

[4] R. Fromm, S. Perissakis, N. Cardwell, et al., "The energy efficiency of IRAM architectures," *Computer Architecture News*, vol. 25, no. 2, pp. 327–337, 1997.

[5] Kang Yi, Huang Wei, Yoo Seung Moon, et al., "FlexRAM: toward an advanced intelligent memory system," in *Proceedings 1999 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, Austin, TX, USA, 1999, pp. 192–201.

[6] Motorola Inc., "DSP 56000 24-bit digital signal processor family manual," 1995, `http://e-www.motorola.com/files/dsp/doc/inactive/DSP56000UM.pdf`.

[7] K. Kitagaki, T. Oto, T. Demura, Y. Araki, and T. Takada, "A new address generation unit architecture for video signal processing," in *Proceedings of the SPIE - The International Society for Optical Engineering Visual Communications and Image Processing '91: Image Processing, 11-13 Nov. 1991*, Boston, MA, USA, 1991, vol. 1606, pp. 891–900.

[8] E. Caspi, A. DeHon, and J. Wawrzynek, "A streaming multi-threaded model," in *Workshop on Media and Stream Processors*, Austin, Texas, 2001, p. 8.

[9] Jr. V.M. Bove and J.A. Watlington, "Cheops: a reconfigurable data-flow system for video processing," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 5, no. 2, pp. 140–149, 1995.

[10] M. Herz, R. Hartenstein, M. Miranda, et al., "Memory addressing organization for stream-based reconfigurable computing," in *9th International Conference on Electronics, Circuits and Systems (ICECS 2002), 15-18 Sept. 2002*, Dubrovnik, Croatia, 2002, vol. vol.2 of *ICECS 2002. 9th IEEE International Conference on Electronics, Circuits and Systems (Cat. No.02EX611)*, pp. 813–817.

[11] Celoxica, "Celoxica data stream manager (DSM)," 2003, `http://www.celoxica.com/techlib/files/CEL-W03120817F6-46.pdf`.

[12] A. Marshall, T. Stansfield, I. Kostarnov, et al., "A reconfigurable arithmetic array for multimedia applications," in *Proceedings of FPGA '99. ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays.*, Monterey, CA, USA, 1999, pp. 135–143.

[13] P. M. Heysters, J. Smit, G. J. M. Smit, and P. J. M. Havinga, "Mapping of DSP algorithms on field programmable function arrays," in *Field-Programmable Logic and Applications. 10th International Conference FPL 2000*, R. W. Hartenstein and H. Grunbacher, Eds., 2000, pp. 400–411.