AUTONOMOUS MEMORY BLOCK FOR RECONFIGURABLE COMPUTING

Wim J. C. Melis, Peter Y. K. Cheung

Department of Electrical & Electronic Engineering
Imperial College London
United Kingdom
{*wim.melis, p.cheung*}@*imperial.ac.uk*

Wayne Luk

Department of Computing
Imperial College London
United Kingdom
w.luk@doc.imperial.ac.uk

ABSTRACT

Current FPGAs include large blocks of memory that require separate address generation circuits. This not only uses logic resources surrounding the memory blocks, but also results in unnecessary routing congestions. This paper proposes the integration of the address generation circuit into the block memory to form an Autonomous Memory Block (AMB). Quantitative comparison between using AMB and conventional FPGA block memory architectures demonstrates that this approach is promising.

1. INTRODUCTION

The evolutionary development of FPGA and reconfigurable logic devices has seen a gradual drift from relatively fine grain architectures, mostly based around lookup tables (LUTs), to ones that include dedicated hardware blocks such as multipliers, microprocessors and block memories. Although the Stratix [1] and the Virtex-II Pro [2], the latest offerings from Altera and Xilinx respectively, have significant amount of memory, the focus of development for these devices remain firmly grounded in their processing capabilities.

One interesting observation is that many DSP applications have inherent parallelism, making them natural targets for implementation on 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 address 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 on 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. This paper assesses the advantages and drawbacks of such an extension to existing FPGA architectures.

The contributions of this paper are: 1) to examine the limitations of the embedded memory in existing FPGA architectures; 2) to propose the Autonomous Memory Block (AMB) which integrates a configurable address generator unit (AGU) within each block memory; 3) to evaluate quantitatively the area saving and performance improvement of employing such an extension when compared with existing FPGA devices.

This paper is organised as follow: after examining related work in Section 2, detail of the proposed Autonomous Memory Block is described in Section 3. In Section 4, the area and speed of the Autonomous Memory Block (AMB) is compared to that of an equivalent implementation on the Virtex-II FPGA. Section 5 concludes the paper.

2. RELATED WORK

Recent developments in FPGA architectures for DSP applications have all been focussing on improving computation. Examples of these are Elixent's Reconfigurable Arithmetic Array [3], and the Field Functional Arrays [4]. Such improvements in processing capability is not accompanied by similar advancements in dataflow. While embedded memories are now found in most modern FPGA devices, these are general purpose block memories that require separate address generation units. Therefore routing of data has to be accompanied by the routing of addresses. Careful examination of common DSP algorithms shows that the patterns in which memory is accessed are usually repetitive and structured. This leads to most DSP processors containing separate address and data ALUs [5]. Such an idea could be extended to FPGA architectures. The Configurable Logic Blocks (CLBs) found in FPGAs could be regarded mostly as data computational resources. The block memories, which normally serve as buffers for the data, could be redesigned to incorporate the address ALU found in DSP processors.

The streaming architectures proposed in [6, 7] also contain special address generation logic that helps in storing or providing streams of data from and to the processing logic. However, the data access model is restricted to streaming of data as found in video signal processing. For general DSP algorithms, this streaming requirement can be limiting. In contrast, the AMB proposed in this work provides a more configurable data access model which can be exploited in a larger class of DSP applications beyond video.

3. THE AUTONOMOUS MEMORY BLOCK (AMB) DESIGN

The latest FPGAs such as Xilinx's Virtex-II Pro and Altera's Stratix devices provide embedded block memory that can be configured as 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

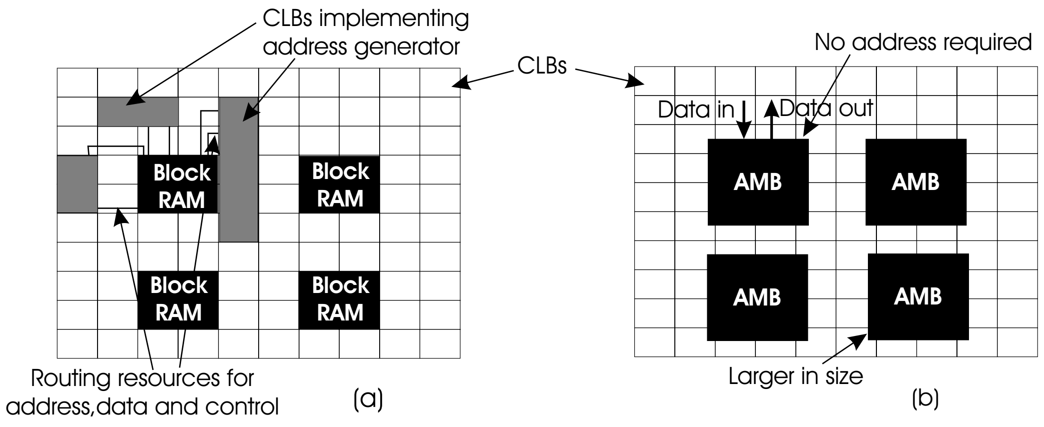

Fig. 1. (a) Existing FPGA block RAM requiring external address generation circuit; (b) Proposed Autonomous Memory Block that includes the Address Generation Unit (AGU)

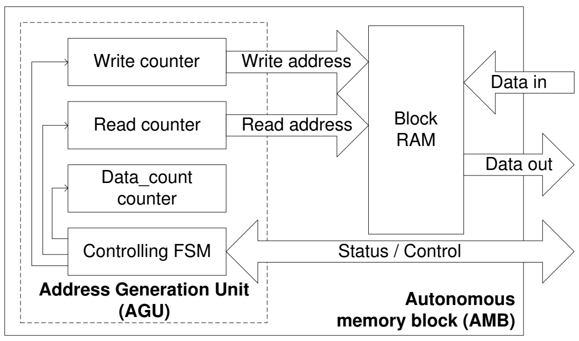

Fig. 2. Address Generation Unit (AGU) in FIFO mode

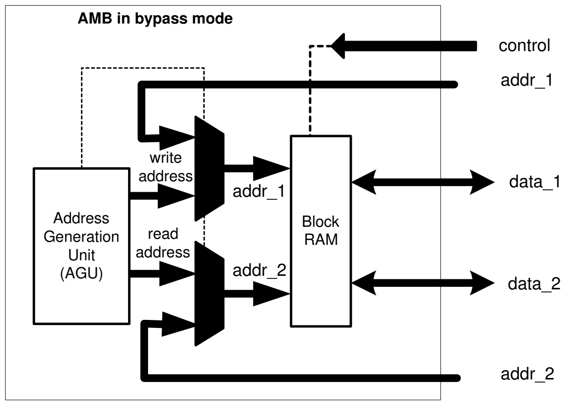

Fig. 4. AMB with bypass mode

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, it would eliminate the need for external address generation circuits for most DSP applications.

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. The current version of the AMB is designed to provide the following modes of operation:

1) *first-in-first-out (FIFO) circular buffer* mode - the first data stored is the first data that is retrieved. This mode is particularly useful when data read and write rates are different. Flow control is implemented to detect and handle overflow and underflow conditions to avoid loss of data. 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 *datacount* counter stores the number of data items currently in the FIFO. The 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.

2) *first-in-multiple-out (FIMO) circular buffer* mode - this is an extension to the FIFO buffer where for each data written, a number of previously stored data are supplied in sequence. The earliest data stored in the buffer is then overwritten. This mode is particularly useful for implementing 1-D convolution, matching and filtering. The circuit of the AGU to implement the FIMO mode is almost the same as that of the FIFO except that the FSM is slightly modified. An additional output-count counter is also required.

3) *last-in-first-out (LIFO)* mode - this is an implementation of a stack-based store. The AGU for the LIFO implementation 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 ac-

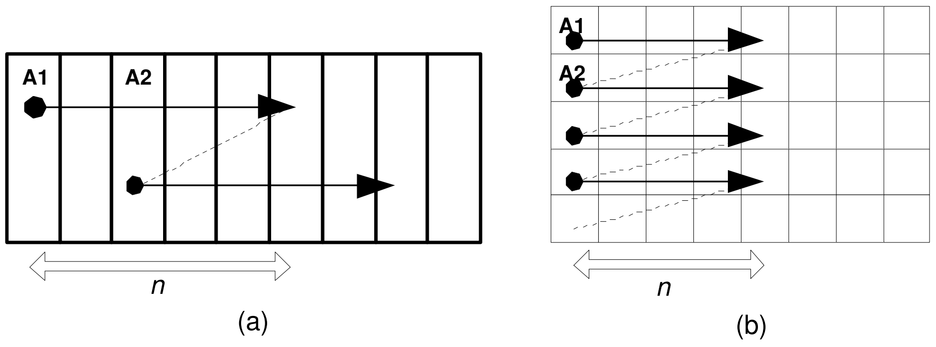

Fig. 3. (a) 1-D ”Striped” access mode; (b) 2-D ”Striped” access mode

	Area (10E-18 m²)	Area Factor	Speed (MHz)	Speed Factor
AGU as dedicated logic	141626	**1**	333	**1**
FIFO design on Virtex II	284340	**2**	161	**0.5**
Striped access design on Virtex II	451236	**3**	110	**0.33**
AGU on Virtex II	12918926	**91**	72	**0.21**

Fig. 5. Area and speed comparison between AMB and conventional block memory

cess 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. 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 3. In Figure 3 (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 3 (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.

In addition to the above 5 different access modes, the AMB can also be configured in the random access mode as shown in Figure 4, bypassing the AGU altogether. This mode is necessary to handle data access models not currently provided by the AGU design.

4. RESULTS

In order to evaluate the advantages of the AMB over conventional FPGA architectures, the address generation unit (AGU) was implemented using dedicated logic and using Virtex-II CLBs. The AGU design including all five modes of data access described in

Using all memory blocks	Xilinx Virtex II Device		
	XC2V40	XC2V1500	XC2V8000
as FIFOs	25%	10%	6%
for Striped Access	75%	30%	17%

Fig. 6. Percentage of CLBs saved depending on memory access mode and size of Virtex II device

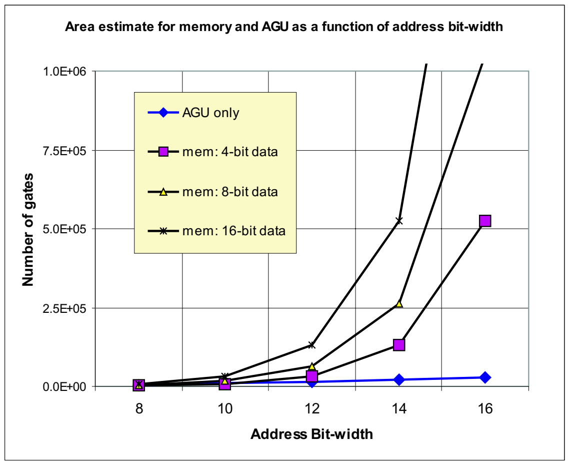

Fig. 7. Gate count for memory and AGU as a function of address widths

Type	Number of address bits for AGU	AGU size (gates)	Memory size (gates)
M512	9	6306	1024
M4k	12	15533	8192
MRAM	16	26943	1048576

Fig. 8. Size of AGU for the different memory types in Stratix

the last section were synthesised (from a VHDL description) to a custom circuit using Synopsys's Design Compiler targetting a typical 0.13 micron process. The area and speed figures for the AGU design was taken from Synopsys's report.

Two reduced AGU designs, one implementing the *FIFO-only mode*, and another implementing the *"striped" access mode*, were synthesised for Xilinx's Virtex-II XC2V8000 FPGA with Synplicity's Synplify synthesis software. The optimized design was 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 was estimated by assuming that the XC2V8000 die area is almost 2.5cm X 2.5cm. Such an estimate should provide area usage for both logic cells and routing resources.

Figure 5 compares the area and speed of the AGU design assuming an 8-bit address width implemented in dedicated logic, against two address generator circuits implementing the FIFO and the "striped" access modes on Virtex-II CLBs. It can be seen that adding the AGU to the block RAM in an AMB provides 2 to 3 times reduction in area while improving the operating speed by a similar factor. The speed of the AGU was deliberately designed to match the speed of the block memory in a 0.13 micron process [1, 2]. 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.

For the sake of completeness, the full AGU design was also implemented on the Virtex-II. This is of course normally not necessary since the FPGA would only need to implement one specific mode of operation instead of the full AGU. However, it is still interesting to see that the CLB based implementation is almost two orders of magnitude larger than a dedicated logic implementation.

By using AMB compared to the conventional memory blocks a certain percentage of CLBs would therefore be saved. This percentage is shown in Figure 6 and is based on an 8-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.

Figure 7 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 derived directly 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 straight forward as Figure 7 suggests. Consider the case of the three different types of block RAM in the Stratix FPGA [1]. The table shown in Figure 8 illustrates that if the AGU were 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. For example, the M4K block RAM in Stratix has 4K bit of static RAM which can be configured with any data width from 1-bit to 36-bits, the address width can therefore vary from 12 bits to 7 bits. The 12-bit AGU would be almost twice the size of the memory array. However, for the larger 512K block RAM in Stratix, the AGU size becomes relatively insignificant.

Due to the AMB architecture, a small delay to the signal paths is added in the bypass mode. This is equivalent to 2 NAND gates followed by a tristate buffer.

5. CONCLUSION

The various modes of data access commonly found in DSP algorithms can be implemented using nearly identical address generation circuits. This leads to an efficient design of a configurable AGU which can easily be integrated into the block memory of FPGA devices.

It has been shown that combining an AGU with memory into an AMB has several benefits for reconfigurable devices. Firstly, the amount of routing resources required 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.

The AGU design reported in this paper is by no means optimal. However it illustrates 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. This should give better improvement in both area and speed.

The design of AMB is another attempt to raise the granularity of FPGA functional blocks. This should lead to a higher level of abstraction in the hardware implementation of DSP algorithms. However, how to map a high level description of an algorithm to these AMBs remains an unsolved problem which will be investigated in the future.

6. REFERENCES

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

[2] Xilinx, "Virtex II platform FPGA user guide," 2003, http://direct.xilinx.com/bvdocs/publications/ds031.pdf.

[3] A. Marshall, T. Stansfield, I. Kostarnov, J. Vuillemin, and B. Hutchings, "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.

[4] 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.

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

[6] 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.

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