ADDRESS GENERATION FOR FPGA RAMS FOR EFFICIENT IMPLEMENTATION OF REAL-TIME VIDEO PROCESSING SYSTEMS

*Najeem Lawal, Benny Thörnberg, Mattias O'Nils**

Electronic Design Division, Department of Information Technology & Media,
Mid Sweden University,
SE-851 70, Sundsvall, Sweden
email: Najeem.Lawal@miun.se, Benny.Thornberg@miun.se, Mattias.Onils@miun.se

ABSTRACT

FPGA offers the potential of being a reliable, and high-performance reconfigurable platform for the implementation of real-time video processing systems. To utilize the full processing power of FPGA for video processing applications, optimization of memory accesses and the implementation of memory architecture are important issues. This paper presents two approaches, base pointer approach and distributed pointer approach, to implement accesses to on-chip FPGA Block RAMs. A comparison of the experimental results obtained using the two approaches on realistic image processing systems design cases is presented. The results show that compared to the base pointer approach the distributed pointer approach increases the potential processing power of FPGA, as a reconfigurable platform for video processing systems.

1. INTRODUCTION

In a Real-Time Video Processing System (RTVPS), the operations performed on each pixel are often neighborhood oriented [1]. A neighborhood of pixels is an operation window about a point in the image and is usually a square or rectangular sub-image about that point. These pixels act as input data on which operations are performed to yield a pixel that corresponds to the central neighborhood pixel in the output image. The neighborhood is formed for each pixel in the input image to produce an output image. One consequence of this is that a large amount of data is required to be buffered depending on the size of the video frame and the operation window. The data flow dependencies require data to be stored in buffers, where each buffer normally corresponds to a row in the video frame. The size of each element in this buffer depends on the dynamic range of the video signal. In addition, the dimensions of the process window affect the number of buffers required for the process operation. For a 5x5 window, four line buffers are required while two line buffers are required for a 3x3 window.

Due to its high processing capability, FPGA is an attractive alternative for implementing high speed, real-time, computation-intensive operations, as in the case of real-time digital signal processing [2]. Also, the on-chip memories of an FPGA allow frequently accessed data to be stored close to the data path, thus eliminating the overheads associated with data fetches to external memory.

The available FPGA on-chip memories are limited and organized as Block RAMs and distributed RAMs. Distributed RAMs are built from the logic resources and are ideal for register files closely integrated with logic. The Block RAMs are more suited to larger on-chip memory storage requirements, such as buffers and caches. For Xilinx FPGA, each Block RAM is a configurable, synchronous block of memory [3], [4]. Additionally, it is also possible to configure each Block RAM as a single- or dual-port memory. For the dual port configuration, the two data ports permit independent synchronous read/write access to the common Block RAM. In addition, the data path widths at each port are independent of each other. Examples of allowable data path widths at each data port for Spartan 3 are 1, 2, 4, 8 (9), 16 (18) and 32 (36) [4]. The values shown in brackets are available when parity bits are used for data storage.

Attempts at optimizing energy consumption, total area and availability of data from the memory subsystems of FPGA and Digital Signal Processor (DSP) have attracted many research efforts [5], [6], [7], [8], [9]. This could be explained by the fact that the area occupied and the power consumed by the memory subsystems are up to ten times larger than that of the data-path [10]. Memory accesses are a major contributor to the power consumption for RTVPS. In addition, latency in memory accesses affects the throughput of the RTVPS. Effective optimization can be achieved through efficient memory architecture and addressing procedure. The work presented by Doggett et al. is optimal where large numbers of memory banks are used as is typical in volume rendering in medical applications [5]. The address generation scheme by Grant el al. is an efficient

* The authors wish to acknowledge the Mid Sweden University and the KK-foundation for their financial supports.

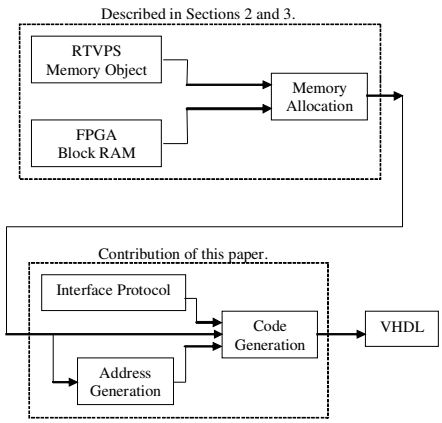

Fig. 1. Memory synthesis design flow.

option for accessing data with addresses within the power range of two [6]. The scheme uses a register and optionally an offset, to specify memory read/write addresses.

The memory exploration algorithm in [7] implements memory allocation and array-mapping to RAMs through tight links to the scheduling effect and non-uniform access speeds among the RAM ports to achieve near optimal memory area and efficient energy requirement. The algorithm is however complex and the execution time may slow down hardware design. Moreover the exploration targets SRAM and DRAM as opposed to the on-chip FPGA Block RAMs, which are the focus of this paper. The address generation technique in [8] is based on address bit inversion to yields effective access time to memory at the cost of up to extra 17.4% of used memory. In [9], various high-level optimizations were explored in order to reduce addressing overhead. Many efficient, often heuristics based, memory optimization algorithms have been developed similar to those in [11], [12], however, most of these are tailored to be efficient on DSP. Thus a memory access methodology is required for the FPGA platform which takes advantage of the global memory allocation architecture proposed by O'Nils et al. for RTVPS implementation [13].

Based on this global memory allocation architecture, an allocation algorithm has been developed and implemented to maximize memory usage while minimizing the read/write accesses [14]. In this paper two approaches for generating addresses for allocation results created by the developed allocation algorithm are presented. The design flow of memory synthesis showing the relationship between memory allocation and code generation is shown in Fig. 1. The result of the Memory Allocation stage is information about organization and allocation of the RTVPS memory objects on FPGA and serves as the input to the work presented in this paper. The generation of VHDL code for hardware implementation of the RTVPS memory objects on FPGA is the final contribution of this work. The approaches proposed

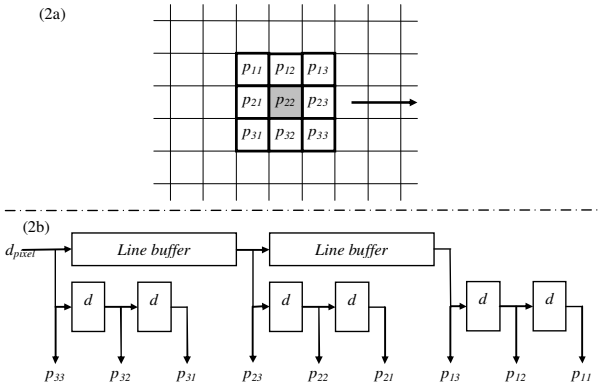

Fig. 2. Basic architecture for the implementation of line buffers in neighborhood oriented image processing operations. Part a) shows an example of a 3×3 neighborhood and b) its implementation.

in this work are also applicable to any memory access formulation in general, where the target of the design tool is the FPGA platform.

2. ARCHITECTURAL ASSUMPTIONS

Fig. 2 shows a 3x3 pixel neighborhood for which an image processing operator calculates an output value for the central pixel. Line buffers and registers are used to store the data flow dependences and provide access to the pixel neighborhood at taps p_{11} - p_{13}, p_{21} - p_{23} and p_{31} - p_{33}. Since only one operator can use these memory objects (line buffers) and all the memory objects are used simultaneously in the RTVPS, O'Nils et al. proposed [13] that the memory objects can be grouped together to form a global memory object (GMO) for the operator. This grouping can be achieved through:

$$W_{Ri} = n_{lines} \times w_p \tag{1}$$

where W_{Ri} is the width of the GMO, n_{lines} is the number of required memory objects for an operator and w_p is the bit width representing a pixel. The length of the GMO is equal to those of the memory objects that formed it [13].

This architecture is preferable to that proposed by Norell et al. in which each line buffer (memory module) is mapped directly to memory [15] since GMOs require a minimal number of required memory entities in comparison to the direct mapping architecture. Consequently, the number of memory accesses for an RTVPS operation is minimal for a GMO.

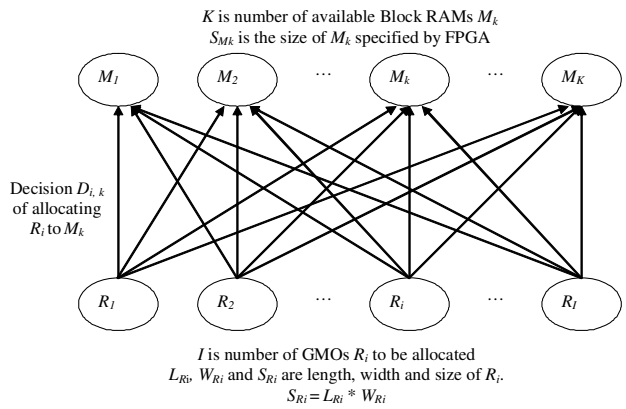

Fig. 3. Allocation model.

3. MEMORY ALLOCATION

A heuristics-based algorithm for efficient allocation of the GMOs, based on the architecture stated above has been developed and implemented [14]. The algorithm creates the GMOs based on (1) and partitions them to ensure that their widths conform to those specified by the FPGA. It also takes advantage of the dual port capabilities of the Block RAMs to achieve near optimal allocations and the possibility of allocating a GMO to as many Block RAMs as required. Hence it is required that the Block RAM support independent read and write accesses at both ports. For every Block RAM available on the FPGA, the allocation algorithm attempts to allocate one or two GMOs to it. Final allocation decisions requiring as few Block RAMs as possible are made, based on the allocations offering the least amount of unused memory space on the allocated Block RAMs. Fig. 3 depicts a model of the allocation.

The allocation algorithm takes the sets of Block RAMs and GMOs as input to generate information about allocations of the GMOs. To illustrate the function of the algorithm, if an operator in an RTVPS requires a neighborhood of 5x5 window with 12-bit gray scale and 640 by 480 frame size as the input video stream this would result in four memory objects each of length L (=640) and width 12 being created. The memory objects would be combined to create a GMO R_i of width 48. Fig. 4 depicts this illustration and *op_id* represents the operator requiring the GMO. Partitioning and allocation of GMOs are potentially complicated tasks depending on the length and width of GMOs in RTVPS applications and the FPGA Block RAM size. Thus an efficient technique for tracking the position of the GMO to be accessed is required. If the GMO in Fig. 4 were to be allocated on a Xilinx Spartan 3 FPGA, it would be partitioned into two segments, of widths 32 and 16, since it would be not possible to have a data path width of 48 on a

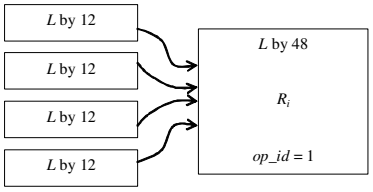

Fig. 4. Creating a GMO.

Xilinx Spartan 3 FPGA. In addition, since each Block RAM is 16KBit (excluding parity feature), the first segment, of width 32, would require 2 Block RAMs, thus creating two partitions. The second segment would require a single partition on a Block RAM. Fig. 5a illustrates the partitioning of a GMO and Fig 5b depicts the allocations of the partitions on two Block RAMs and unused memory space. The major objective of the allocation algorithm is to ensure that the amount of unused memory is minimal. The unused memory in Fig. 5b is however unavoidable since the memory required by the GMO is less than that provided by two Block RAMs.

Hence organizing a GMO into many segments and partitions for allocation purposes would be usual encounters in RTVPS applications. Implementation of accesses to these GMOs irrespective of their organization, and the Block RAM constraints is the focus of the work presented in this paper.

4. ADDRESS GENERATION

The allocation software ensures that each entry of a Block RAM data object stores information about width and length of the GMO segment allocated to it, the port used for allocation and the hierarchy of its segment in the GMO. In addition, each partition stores information about the Block RAM to which it is allocated, the port of allocation and its start address on the Block RAM, the GMO and segment to which it belongs.

The advantage of sequential accesses to memory for RTVPS applications can lead to improved memory performance by using pointers that increase incrementally when there are valid pixel values. Using the GMO architecture further reduces the number of such pointers to one for each RTVPS operator. The pointers may be implemented by using a single register for each GMO, further referred to as base pointer, or by using a register for each partition in a GMO, further referred to as distributed pointers.

To this end, the results of the memory allocation stage in Fig. 1. are imported into the address generation module. From these allocation results GMOs are reconstructed, and address spans for each partition in a Block RAM are generated. The start and end addresses of each partition are calculated. Offsets are considered where dual ports are used

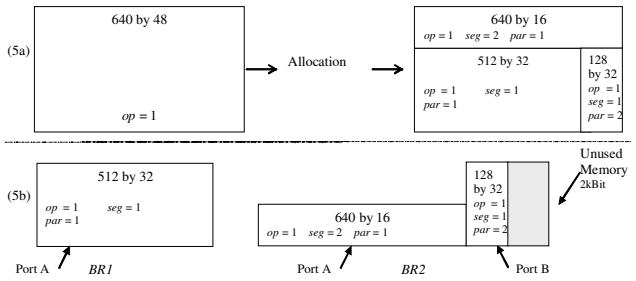

Fig. 5. Organization and Allocation of a GMO.

for the allocation on Block RAMs for different partitions in order to avoid memory overlap. The generated addresses are used to determine the location of each GMO element. The descriptions for accessing the GMO elements using two approaches, namely the base pointer approach and the distributed pointer approach are presented as follows.

4.1. Base Pointer Approach

In this approach, a single pointer is used to track the location of the element to be accessed in the GMO. The pointer starts at zero and increases to one less than the length of the GMO and then resets to zero. Since the memory accesses are clocked, the value of the pointer increases with clocked access to the Block RAM when there are valid data. Address spans for each partition of the GMO are used to determine the relevant Block RAM relating to the element accessed, depending on the value of the pointer. Hence, only the relevant Block RAMs are enabled while the other related Block RAMs are disabled. Fig. 6a depicts this approach for a simplified case in which a GMO consists of a single segment with two partitions.

In the figure, partitions *p1* and *p2* are allocated to Block RAMs *BR1* and *BR2*. From Fig. 6a, when the value of *base* is within the span of *p1*, the appropriate port on *BR1* is enabled and accessed while the relevant port on *BR2* is disabled. The reverse is the case when *base* is no longer within the span of *p1*, i.e. within the span of *p2*. This simple example could be extended to cases in which more than one segment makes up a GMO and each segment has more than 2 partitions. A formal description of this approach is shown in Fig. 6b. Fig. 6c depicts the base pointer implementation of the GMO shown in Fig. 5. In the figure, *BR1_EN_A*, *BR2_EN_A* and *BR2_EN_B* represent the enable signals on port A of *BR1,* port A of *BR2*, and port B of *BR2* respectively. Likewise, *BR1_A_Adr*, *BR2_A_Adr* and *BR2_B_Adr* are the address signals on port A of *BR1,* port A of *BR2*, and port B of *BR2* respectively. A Block RAM is enabled or disabled by assigning '1' or '0' to its enable signal.

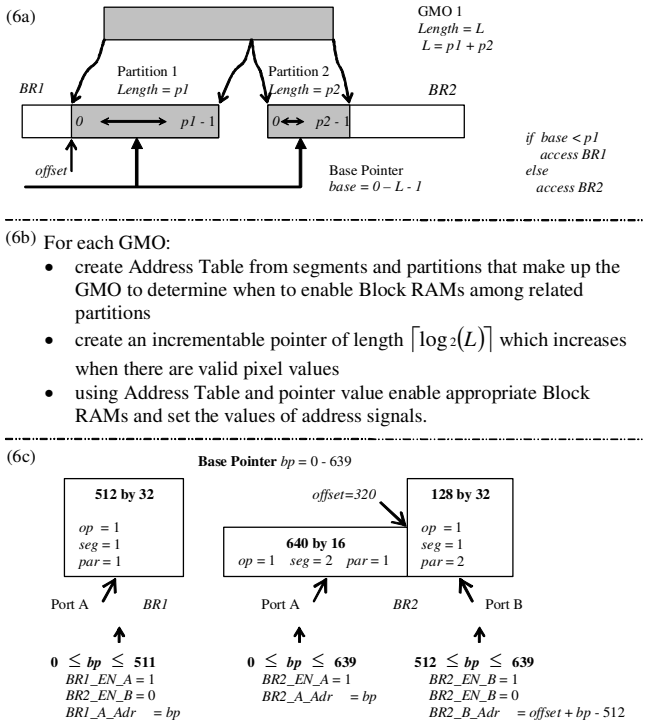

Fig. 6. Base Pointer Approach.

4.2. Distributed Pointer Approach

In this approach, each partition is handled separately, starting with the first partition in a segment. Local pointers equal in length to the length of each partition are created. As long as the enable signal of Block RAM for a partition is high, memory access is started at its first position using its pointer and continues incrementally when there are valid data until its full length is reached. During this period, the partition ensures its enable signal is re-asserted while the enable signals of the neighboring partitions of the same segment are kept low. Controls are transferred to the next partition of similar segment when the upper limit of the partition is reached. If however, the partition is the last in the segment, controls are transferred to the first partition. Since the address buses of partitions on Block RAMs provide appropriate bit vectors to cover their entire lengths, they are used as the local pointer. In this approach, the enable signals of all the first partitions are set to high at start-up to ensure that memory accesses start with the first partitions. Fig. 7a depicts this approach. A simplified case of a GMO consisting of a single segment with two partitions *p1* and *p2* allocated on Block RAMs *BR1* and *BR2* respectively is considered in Fig. 7a. Fig. 7b and 7c show formal description and implementation of the GMO depicted in Fig. 5 using this approach. Signals in Fig. 7c have similar

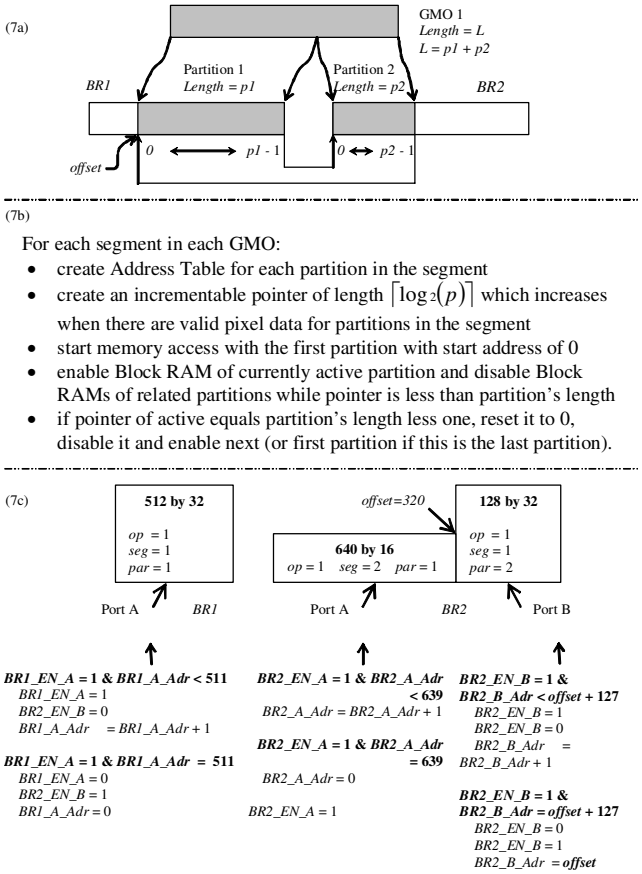

Fig. 7. Distributed Approach.

meanings as in Fig. 6c. Since the 640-by-16 partition is the only one in its segment, it is always enabled and the address is reset to 0 when it reaches its the upper limit.

5. EXPERIMENTAL RESULTS

The work presented in this paper incorporates address generation using the two approaches described above. The address generation and VHDL code generation were implemented in C++ which takes FPGA technology files and allocation results (described in Section 3) as input. The implementation was simulated using the memory requirements of real-time video processing design cases [13]. The first design case was a spatio-temporal median filter with a neighborhood of seven frames and two line buffers. Two instances of this design case were considered. The first, (1-1), being a VGA frame with 24-bit RGB pixels and a 640 frame length, while the second, (1-2), was a PAL frame with an 8-bit gray scale pixel and a 708 frame length. The second design case was a machine vision system with a median filter, segmentation and three 1-bit morphological operations. Also, in this design case two instances were considered. The first, (2-1), being an 8-bit gray scale with VGA resolution as the input video stream while the second (2-2) had a 12-bit gray scale with 1.3 MPixel resolution as the input video stream. Table 1 shows the summary of the memory requirements for the design cases considered.

Table 1. Memory requirement for design cases.

Design Case	#	Rows	Width	Length	Size (kbit)
Case 1-1	7	2	24	640	210
Case 1-2	7	2	8	708	77.4
Case 2-1	1	4	8	640	20.0
	1	1	19	256	4.75
	3	16	1	640	30.0
Case 2-2	1	4	12	1300	60.94
	1	1	21	4096	84.0
	3	16	1	1300	60.94

Table 2 shows the resources required to access the allocated memory objects for the design cases in Table 1, the number of Block RAM required for the allocations and the hardware operating frequency for the two approaches. Xilinx Spartan 3 FPGA was the target platform for implementing both approaches.

6. DISCUSSIONS

Depending on the number of partitions relating to a GMO, address look-up tables are required to set the enable signals and the values of the address signals to the appropriate Block RAMs on which the element of the GMO currently being pointed at is allocated, while also disabling related Block RAMs. In the Base Pointer Approach, these accesses to the Block RAMs are centrally controlled at the GMO level using a pointer. Hence only one set of address look-up tables is required for each GMO. By contrast, in the Distributed Approach, each partition has its separate address look-up table that is unrelated to those of related partitions. The use of a partition's address look-up table depends on the value of its enable signal. Hence the total number of address look-up tables for one GMO depends on the number of partitions making up the GMO. This is evident by comparing Figures 6c and 7c. The first row of Table 2 confirms this. Thus the Base Pointer Approach yields more efficient use of hardware resources than does the Distributed Approach. The differences in resource requirements are however marginal amounting to less than 3% of the available resources, for example, Xilinx Spartan 3 XC3S400 series [3].

Delays associated with the distribution of a single base pointer caused by long delays in the FPGA are eliminated in the Distributed Pointer Approach since each Block RAM partition will have one local pointer. The use of separate

Table 2. Comparison of the two approaches.

	Case 1-1		Case 1-2		Case 2-1		Case 2-2	
	BP	**Dist**	**BP**	**Dist**	**BP**	**Dist**	**BP**	**Dist**
No. of 4 input LUTs:	653	994	334	356	155	191	560	804
No. of BRAMs:	14	14	6	6	5	5	13	13
Max. Frequency (MHz):	116	186	106	183	140	214	91	173
Frequency Comparison (%):	100	160	100	173	100	153	100	190

address look-up table for each partition in the Distributed Pointer Approach increases the speed of memory accesses and consequently, increases operating frequency. This is because all signals required for memory accesses are calculated simultaneously at the clock edge. As the third row in Table 2 shows, the Distributed Approach yields more rapid access to data than does the Base Pointer Approach.

7. CONCLUSIONS

In this paper two approaches for generating addresses of memory objects grouped at the operator level in real-time video processing system has been presented. This work makes the implementation of accesses to memory data trivial irrespective of their allocation. The two approaches, Base Pointer Approach and Distributed Pointer Approach were compared with respect to used hardware resources and speed performance. The results indicate that the Base Pointer Approach, which is the straightforward method, requires fewer resources than does the Distributed Pointer Approach. This resource reduction is however marginal when compared to the total capacity of the device. The Distributed Approach is at least 50% faster than the Base Pointer Approach for all the design cases considered. Hence, the Distributed Approach improves the processing power of FPGA as a reconfigurable platform for RTVPS implementation. Additionally, the automatic generation of addresses and VHDL code will simplify the implementation of real-time video processing systems.

8. REFERENCES

[1] Gonzalez, R., and R. Woods, *Digital Image Processing*, 2nd edition, Addison-Wesley Pub., 2002.

[2] G. Liersch, and C. Dick, "Reconfigurable Gate Array Architectures for Real Time Digital Signal Processing", *Conf. Record of the Twenty-Eighth Asilomar Conf. on Signals, Systems and Computers,* pp. 1383 - 1387, Nov 1994.

[3] XILINX, *Using Block SelectRAM+ Memory in Spartan-II FPGAs*, XAPP173 (v1.1), Dec 2000.

[4] XILINX, *Using Block RAM in Spartan-3 FPGAs*, XAPP463 (v1.1.2) July 23, 2003.

[5] M. Doggett, and M. Meissner, "A Memory Addressing And Access Design for Real Time Volue Rendering", *Proc of IEEE Int. Symp. on Circuits and Systems*, pp. 344 - 347, June 1999.

[6] D. Grant, P.B. Denyer, and I. Finlay, "Synthesis of Address Generators", *Digest of Tech. Papers of IEEE Int. Conf on Computer-Aided Design*, pp 116-119, Nov 1989.

[7] J. Seo, T. Kim, and P.R. Panda, "Memory Allocation and Mapping in High-Level Synthesis - An Integrated Approach", *IEEE Trans. on VLSI Syst.*, pp. 928 - 938, Oct. 2003.

[8] H. Schmit and D. E. Thomas, "Address generation for memories containing multiple arrays", *IEEE Trans. on CAD of Integ. Cct. and Sys.*, pp. 377 – 385, May 1998.

[9] M. Miranda, F. Catthoor, M. Janssen and H. De Man, "High-level address optimization and synthesis techniques for data-transfer-intensive applications", IEEE Trans. on VLSI Systems, pp. 677-686, Dec 1998.

[10] F. Balasa, F. Catthoor, and H.M. Man, "Background memory area estimation for multidimensional signal processing systems," *IEEE Trans. on VLSI Syst*. Vol 3, pp. 157 - 172, June 1995.

[11] R. Leupers, and P. Marwedel, "Algorithms for Address Assignment in DSP Code Generation", *Digest of Tech. Papers of IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 109 - 112, Nov. 1996.

[12] N. Sugino, H. Miyazaki, S. Iimuro, and A. Nishihara, "Improved Code Optimization Method Utilizing Memory Addressing Operation and its Application to DSP Compiler", *IEEE International Symposium on Circuits and Systems*, pp. 249 - 252, May 1996.

[13] M. O'Nils, B. Thörnberg and H. Norell, "A Comparison between Local and Global Memory Allocation for FPGA Implementation of Real-Time Video Processing Systems", in *Proc of IEEE Int. Conf. on Signals and Electronics Systems,* Sept 2004.

[14] N. Lawal, *Global Block Ram Allocation Algorithm For FPGA Implementation Of Real-Time Video Processing Systems*, Masters Thesis*,* Mid Sweden University, Nov 2004.

[15] H. Norell, B. Thörnberg and M. O'Nils, "Automatic Hardware Synthesis of Spatial Memory Models for Real-Time Image Processing Systems", *Proc. of the 21th Norchip Conf.,* Riga, Latvia, Nov 2003.