Performance Evaluation of Vision Algorithms on FPGA

Mahendra Gunathilaka Samarawickrama

DISSERTATION.COM

Boca Raton

Performance Evaluation of Vision Algorithms on FPGA

Dissertation.com
Boca Raton, Florida
USA • 2010

ISBN-10: 1-59942-373-1
ISBN-13: 978-1-59942-373-9

Abstract

The modern FPGAs enable system designers to develop high-performance computing (HPC) applications with large amount of parallelism. Real-time image processing is such a requirement that demands much more processing power than a conventional processor can deliver. In this research, we implemented software and hardware based architectures on FPGA to achieve real-time image processing. Furthermore, we benchmark and compare our implemented architectures with existing architectures. The operational structures of those systems consist of on-chip processors or custom vision coprocessors implemented in a parallel manner with efficient memory and bus architectures. The performance properties such as the accuracy, throughput and efficiency are measured and presented.

According to results, FPGA implementations are faster than the DSP and GPP implementations for algorithms which can exploit a large amount of parallelism. Our image pre-processing architecture is nearly two times faster than the optimized software implementation on an Intel Core 2 Duo GPP. However, because of the higher clock frequency of DSPs/GPPs, the processing speed for sequential computations on on-chip processors in FPGAs is slower than on DSPs/GPPs. These on-chip processors are well suited for multi-processor systems for software level parallelism. Our quad-Microblaze architecture achieved 75-80% performance improvement compared to its single Microblaze counterpart. Moreover, the quad-Microblaze design is faster than the single-powerPC implementation on FPFA. Therefore, multi-processor architecture with customised coprocessors are effective for implementing custom parallel architecture to achieve real time image processing.

To my loving wife Prasadi, parents and teachers for giving me constant support and motivation.

Acknowledgment

I wish to thank my supervisors Dr. Ajith Pasqual and Dr. Ranga Rodrigo for their support and encouragement during this research. Their insight, guidance, feedback and especially the constructive criticisms contributed enormously to the production of this thesis.

I am grateful to Dr. E.C. Kulasekere, the coordinator of this research and Dr. Chathura De Silva, the chairman of the progress review committies and Prof. (Mrs.) I.J. Dayawansa, the postgraduate research advisor for their feedback, kind advice and invaluable suggestions given.

I am deeply indebted to other academics and administrators who have provided helpful advice and knowledge during this research.

I also wish to extend my gratitude to Zone24×7 (Pvt) Ltd. for providing laboratory facilities.

I acknowledge the financial support given by the *University of Moratuwa Senate Research Committee grant SRC-297*, to enable me conduct the masters program at University of Moratuwa.

Finally, I am thankful to my parents, family and friends for their care, commitment and support they extended to me during this research program.

Mahendra G. Samarawickrama

July 2010

Contents

List of Figures

List of Tables

Abbreviations

Following abbreviations or acronyms have been used in this thesis.

Abbreviations/acronyms	**Meaning**
ADDR	Address: Memory location for read/write data
BRAM	Block RAM
CLK	Clock
CMP	Chip Multiprocessor
DDR	Double Data Rate
DIN	Data Input: Data written into memory
DOUT	Data Output: Synchronous output of the memory
DSP	Digital Signal Processor
EDK	Embedded Development Kit
EN	Enable: Enables access to memory
EEPROM	Electrically Erasable Programmable ROM
FPGA	Field-Programmable Gate Array
GPP	General Purpose Processor
HDL	Hardware Description Language
HLL	High-Level Language
HLS	High-Level Synthesis
LMB	Local Memory Bus
LUT	Lookup Table
ROM	Read-Only Memory
PIF	Performance Improve Factor
PLB	Processor Local Bus
SoC	System on Chip
WE	Write Enable: Allows data transfer into memory
XCL	Xilinx Cache Link
XPS	Xilinx Platform Studio

Nomenclature

Following symbols or notations have been used in this thesis.

Notation	Meaning
T_{nbhd}	Time to read neighborhood pixels around the first pixel of the image
N_{mask}	Kernel dimension
f_{clk}	Clock frequency
T_{img}	Total time needs to process all the pixels of the image
M_{img}	Number of pixels per image
T_{SM}	Time to execute in single-microblaze architecture
T_{QM}	Time to execute in quad-processor-microblaze architecture

CHAPTER 1

Introduction

1.1 Background

Real-time image processing is a primary requirement for applications such as navigation and tracking. Real-time image processing requires processing an image at video rate, i.e., at least 30 images per second. There are several technologies that are available to achieve this goal. For example, DSPs, FPGAs and GPPs can be mentioned. However, each technology has advantages and disadvantages when compared to the other. In fact, their usability depends on the requirements related to power consumption, cost, flexibility and design cycle time. Regarding the power consumption, DSPs and FPGAs are more suitable than GPPs. When considering the development cost of DSPs, it is very expensive and suitable only where there is mass scale production. Moreover, in DSPs, time-to-market is high and time-in-market is low compared to FPGAs. Nevertheless, available DSPs are generally cheaper and faster in performance than FPGAs. Regarding the GPPs, when they are implemented with software libraries with guaranteed high efficiency in machine level instructions handling, modern multi-cores GPPs run very fast. As an example, we can show the compatibility of Intel Core i7 and OpenCV library. In the case of design flexibility, FPGAs are more suitable with their parallel and reconfigurable architecture. However, depending on the designing approach, FPGA-based algorithm implementations are often complex and tedious compared to DSPs and GPPs.

Therefore, modern real-time vision systems are not relying upon a single technology, but more converge technologies to get the advantages of each technology. Fig. 1.1 shows, Texas machine vision solution [1] which is a good example for technology convergence. It is composed of FPGA, DSP and interface to communicate with the GPP. However, design flexibility in such systems is limited because of their heterogeneous nature.

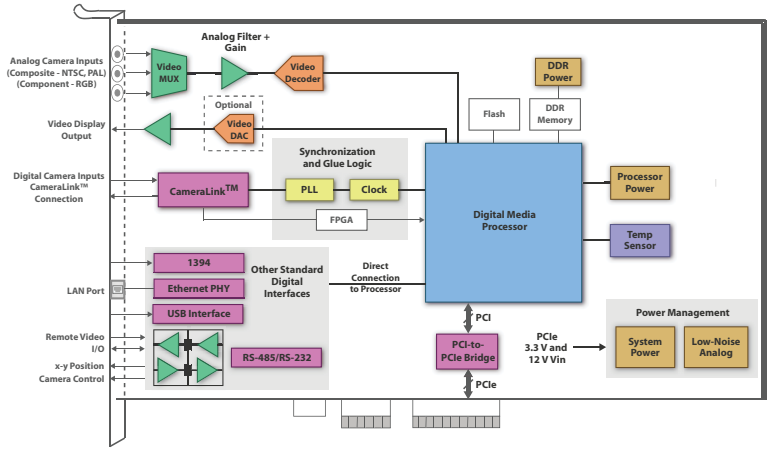


Figure 1.1: Texas machine vision solution [1] is a good example for technology convergence. It is composed of FPGA, DSP and interface to communicate with GPP.

By focusing on FPGAs as a flexible and reconfigurable solution, in recent years (2008-2010), FPGA devices are developed with significantly higher amount of internal memory and logic resources with much higher bandwidth [2]. Fig. 1.2 shows the growth of FPGA memory, resources and bandwidth in the past decade. This growth gives system designers a good opportunity to design vision coprocessors which consume large amount of hardware resources.

Moreover, it can be seen that (Fig. 1.3), high-performance computing (HPC) applications' demands have outpaced the conventional processors' performance. Therefore, as a solution to the real-time image processing, it is possible to make hardware acceleration with application-specific coprocessors. Due to the right

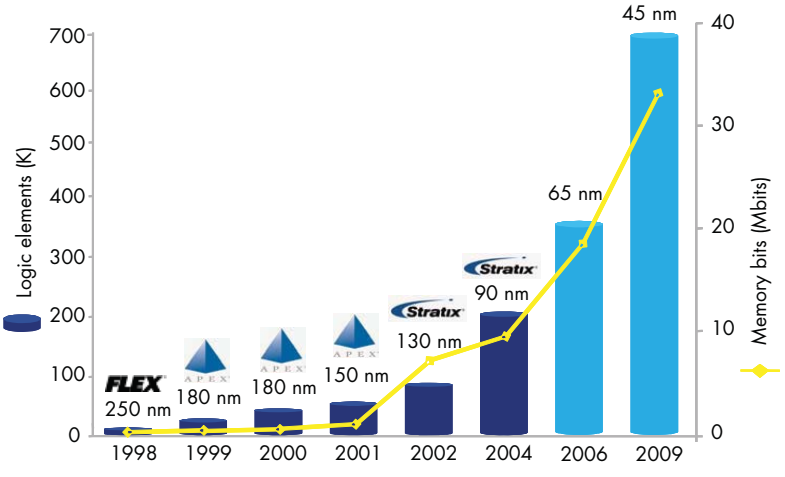


Figure 1.2: The growth of FPGA memory, logic resources and bandwidth [2]. This growth gives system designers good opportunity to design vision coprocessors which consume large amount of hardware resources.

combination of price, performance, and ease-of use, along with significant power savings, FPGA is a good option for coprocessor designing.

FPGA devices have not only grown in terms of resources and speed but also in the amount of embedded processors within their fabric. These embedded processors together with coprocessors are now capable of designing custom computers to achieve common tasks.

FPGAs offer many performance and implementation benefits for executing image processing applications [3]. The main advantage of FPGA-based design is the flexibility to exploit the inherently parallel nature of many image processing problems. For example, many vision algorithms require the repeated application of the same local operation, such as convolution, to every region in an image. In a serial processor this can be quite time consuming. However, in an FPGA, multiple convolutions can take place simultaneously.

There are many FPGA-based vision architectures that have been implemented with various types of resources and strategies. Huitzil and Estrada [4] proposed a design where FPGA device is incorporated with two external memory banks.

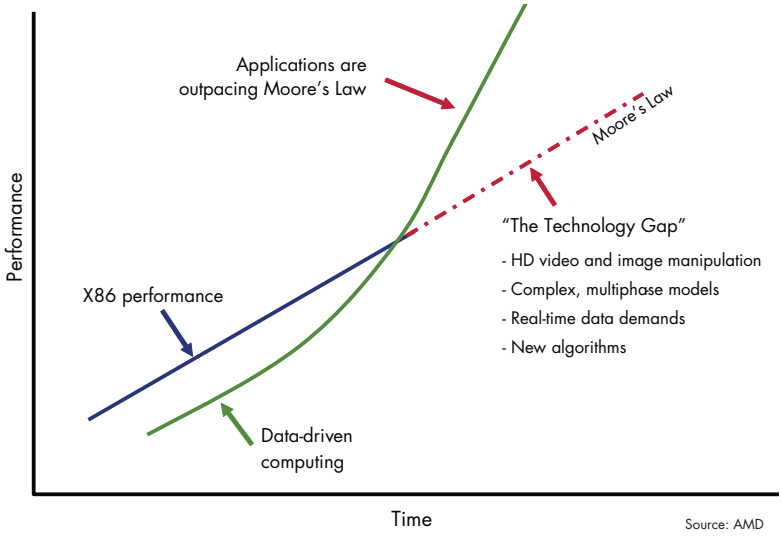


Figure 1.3: The technology gap between demand and performance of a real time system [2]. High-performance computing (HPC) applications are now demanding more than processors alone can deliver, creating a technology gap between demand and performance.

It can process the SUSAN algorithm for high-speed edge detection on 512×512 images with 50 MHz clock up to 120 images per second. An efficient address generator module with register-intensive hardware buffers significantly reduce the frequent real-time memory access, hence speed up the algorithm at the expense of high resource utilization.

FPGAs have also been used to process computationally complex algorithms fast. Wei *et al.* [5] implemented DDR memory based architecture for optical-flow algorithms with the help of Xilinx EDK tools. It can process 640×480 16-bit YUV images through an Ethernet interface up to 64 images per second. By implementing a PowerPC, rather than implementing a coprocessor to make calculations related to optical flow algorithm, they could handle floating points easily. It is a good tradeoff between accuracy and processing speed.

Modern FPGA/DSP systems have various types of hierarchical memories and caches that differ significantly in their sizes, latency times, and data bandwidths. Daniel Baumgartner *et al.* [6] showed that the memory selection, configuration and data access pattern have a significant influence on the achievable speed. From their results, Fig. 1.4 compares execution times of Gaussian pyramid function under different memory configurations on a C64x DSP. The results clearly show that memory access overhead critically slows down the algorithm execution speed.

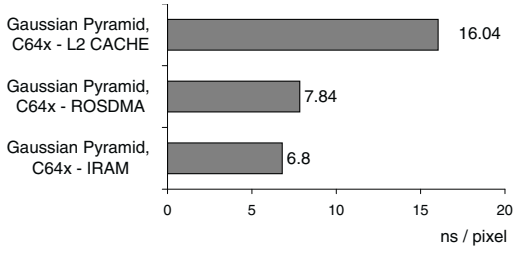


Figure 1.4: Execution times of Gaussian pyramid function under different memory configurations on a C64x DSP: IRAM (Internal RAM), ERAM (External RAM) + ROSDMA (Resource Optimized Slicing Direct Memory Access), ERAM + L2-Cache [6].

To improve the performance of FPGA-based implementations, pipeline inser-

tion is one of the main strategies. Teifel and Manohar [7] showed that pipelines improve overall clock frequency and speed up the FPGA-based systems. Fig. 1.5 shows the characteristics of the improvement of maximum clock frequency against the number of pipeline stages. However, the speed improvement is not uniform. When more pipeline stages are added in the design the latency increases and the maximum clock frequency tends to decrease.

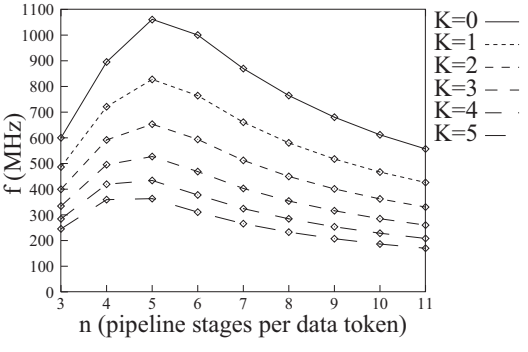


Figure 1.5: Maximum operating frequency curves for one token in a linear pipeline of n stages. K is the number of routing switches between every pipeline stage (K=0 is the "custom" case when there are no switches between stages) [7].

FPGA-based embedded processing requirements are growing at a rapid pace and system architects are not limited to coprocessors and are turning towards hybrid multiprocessor designs [8]. The Xilinx Platform Studio (XPS) and Embedded Development Kit (EDK) is a comprehensive solution for designing embedded programmable systems. These tools make it easy to design powerful chip multiprocessing (CMP) systems.

Implementation of an algorithm on the FPGA requires building and utilising FPGA-specific hardware such as fast multipliers and BRAM, on an open architecture platform. This is fundamentally different to the design of software for the fixed architectures of conventional processors. Although software techniques may help to define the image processing algorithm and facilitate its programming, they will provide little guidance on how to manage hardware resources on the FPGAs.

Gribbon *et al.* [9] are actively working for visual programming language which can incorporate hardware design patterns for FPGA-based image processing.

Moreover, generating hardware by translating High-Level Language (HLL) to Hardware-Description Language (HDL) using application mappers, is developed as a modern trend. In other words, this method can likely increase performance and productivity while minimizing unnecessary development time and effort. Curreri *et al.* [10] in their survey state that the development time using VHDL, including the researchers algorithm-acquaintance period, was roughly three times longer than using AccelDSP. However, the HDL version has higher achievable core frequency because the automated procedure of translating HLL to HDL creates overhead that increases critical-path delays [10]. The resulting clock frequency, pipeline length, and selected resource utilization of the HLL and HDL implementations are shown in Table 3.5.

Table 1.1: HLL vs. HDL comparisons in core design [10]

Core Design	Design 1		Design 2	
	HLL	HDL	HLL	HDL
Core Freq. (MHz)	140	180	150	200
Pipeline Len. (cycle)	31	34	31	34
MULT18×18s (%)	5	5	5	5
Slices (%)	18	17	20	17

By analyzing above design strategies and concepts, we can identify several challenges, which are related to FPGA-based image processing.

1.2 Design Challenges

Complete development of an image processing application for an FPGA-based system involves four stages: (1) problem specification, (2) algorithm development, (3) architecture selection, and (4) implementation. Considering the algorithm development and implementation, simply mapping of a software implementation into hardware often falls short of the potential benefits offered by an FPGA solution as they do not tend to leverage concurrency. On the other hand, schematic entry and HDLs are often too low-level and designing image processing algorithms at

this level is complex, tedious and error prone [11]. Therefore, when implementing vision algorithms on FPGAs, we have to deal with development issues in each stage.

Memory configuration is an important design issue concerning architecture selection. Since there are several different memory resources on the FPGA systems (registers, BRAM and DDR-RAM), at the designing stage it is important to use them appropriately. For example, registers are used to store intermediate values in the algorithms, while BRAM are ideally suited for row buffering which is necessary for local filtering. Although on-chip BRAM is usually sufficient to buffer several rows of an image, the off-chip memory (DDR-RAM or SRAM) is often necessary for applications that require frame buffering.

Regarding the implementation of some image processing tasks, they are harder to implement and tend to be inefficient in custom hardware because they may not be running frequently. For example, histogram equalization runs once per frame as opposed to convolution operation which need to be run for each pixel in an image scan. This can leave hardware idle for long periods of time which is often an ineffective use of the limited logic resources. An alternative is to implement some of these tasks in a processor with software programmability in the same system. This is how, PowerPC or Microblazes based designing techniques are becoming increasingly important in high performance computing on FPGAs [8].

Therefor, implementation and integration of software and hardware on FPGA, can be seen as an effective solution in real-time vision applications. A finite state machine can be built on the FPGA fabric for performing intermediate-level tasks with low processing requirements. For more complicated processing, software/hardware processors (PowerPC or Microblazes) can be used. Implementing a processor based architecture provides several advantages. Instruction sets can be customized for efficiency and wiring delays are minimized. High development costs can be offset against software programmability; the same FPGA configuration can be used for several applications and a custom parallel computer can be built quickly [12].

CHAPTER 2

Design and Implementation

By referring designing issues described in the chapter 1, we made three approaches to implement different FPGA-based architectures to benchmark and compare their performance on real-time vision computing. In our first approach, we implemented image pre-processing architecture with gate-level parallelism. Then we implemented an image coprocessor using HLL synthesis tool and investigated its performance and advantages of HLL synthesis tools. Next, multiprocessor architecture was implemented to benchmark software level parallelism. By realizing these architectures, we presented our comprehensive analysis, how modern FPGAs can be used in real-time image processing. In detailed description of architectures' implementations are described below as separate subsections.

2.1 Image Pre-Processing Architecture

A typical processing sequence of a vision system can be classified as a layered architecture (Fig. 2.1). The acquisition layer controls the sensor interface, pixel addressing, and passes source pixels to the pixel pre-processing layer, which in turn, performs corrections such as noise reduction and compensation [13].

For pixel pre-processing, it needs low amount of memory for buffering and can be implemented using on-chip block-memory on FPGAs. As an intermediate RAM structure, block RAMs [14] provide less complexity and speedy access (Fig. 2.4). Since the use of block RAMs do not affect the slices, they are well suited in terms

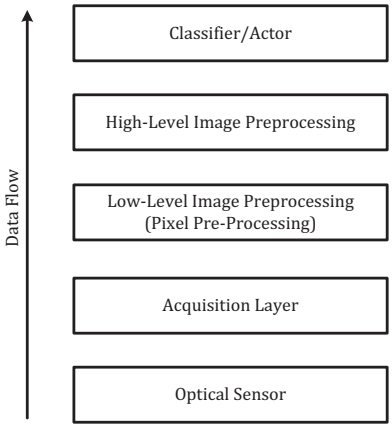


Figure 2.1: Layered architecture of a vision system

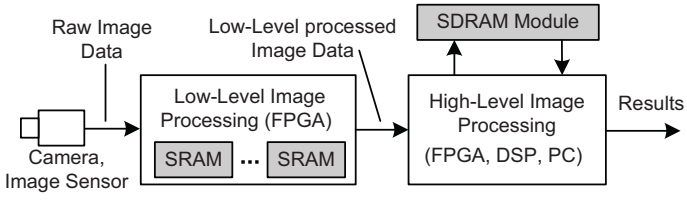


Figure 2.2: A typical vision system: The acquisition layer controls the sensor interface, pixel addressing, and passes source pixels to the pixel pre-processing layer, which, in turn, performs corrections such as noise reduction and compensation [13].

of FPGA resources.

Therefore, in our first approach to design a vision architecture, we developed an image pre-processing architecture which utilizes block RAMs for image buffering [15]. Furthermore, it consists of several organized modules (Fig. 2.3), which exploit parallelism with pipelining, loop splitting and independent operation execution.

In our architecture, we made a modular approach to achieve high flexibility in algorithm changes and modifications. The vision core is an application-specific co-processor where we implement image processing algorithms. The algorithm is initiated after the image or part of the image is stored in data store memory-bank. The results are stored in a separate memory-bank which is named as a

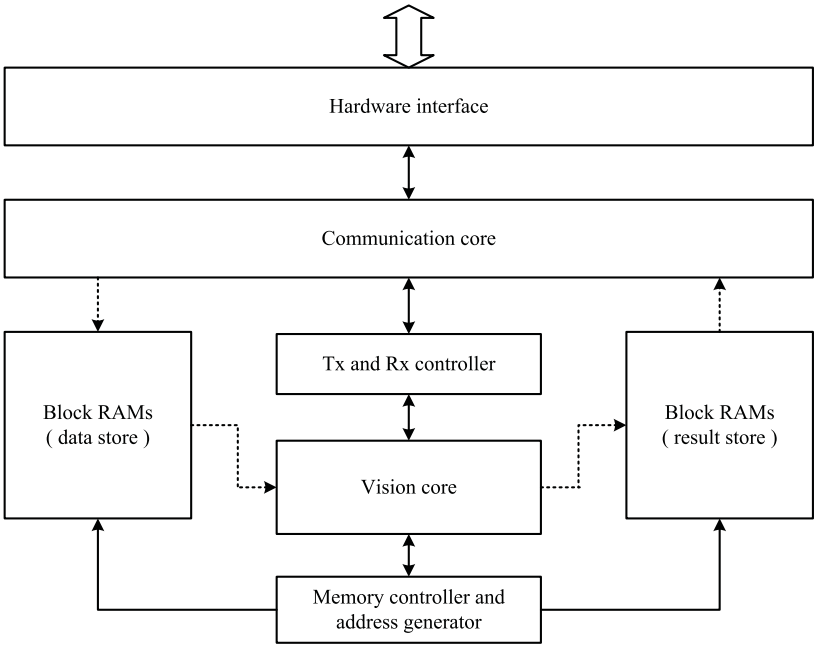
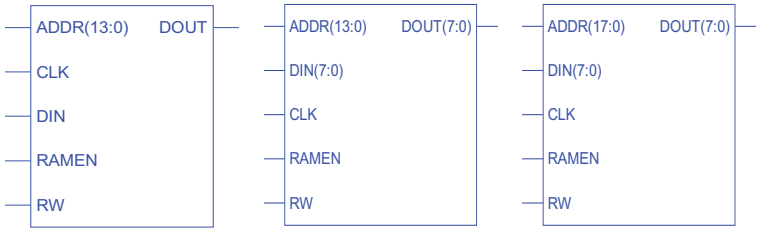


Figure 2.3: Image pre-processing architecture [15]: After the full frame is stored in the block-memory, the algorithm is initiated and results are stored in another block-memory. Dotted and continuous lines represent data and control busses, respectively.

result store. Since the memory controller and address generator modules are implemented apart from this module, the implementation of different algorithms is flexible and architecture independent.

In designing the memory-banks, we instantiated the basic RAM primitive (Fig. 2.4(a)) to achieve the maximum memory depth. However it limits the port-width of the memory to 1-bit and it required multi-level coupling. Fig. 2.4 and Fig. 2.5 shows, how we composed the memory-banks to achieve memory requirement.



(a) (b) (c)

Figure 2.4: Implemented memory-bank schematics: (a) Basic schematic of RAMB16_S1 RAM primitive; (b) RAMB16_S1 coupled to store and access Bytes: this is maximum of 16KB; (c) Memory-bank which is implemented by coupling multiple 16KB RAMs.

Regarding the memory management, the address generator module creates the addressing sequence to access the image pixels that are needed to perform the vision algorithm. The address generator module is coupled with the memory-bank through the address bus ADDR(m:0). The width of the bus is determined by the required address depth. To support the address generator, there exists a memory controller module. It selects and enables/disables memory blocks while the algorithm is in progress. Once the address generator module generates the address of the data source or data store, this enables the required memory block and performs the read/write operation.

The communication core is mainly concerned with the transfer of image frames in and out through the FPGA interface. This interface can be Ethernet, PCI or a customized interface. Since the algorithm and memory access are performed at a much higher speed, design of the communication module is critically important

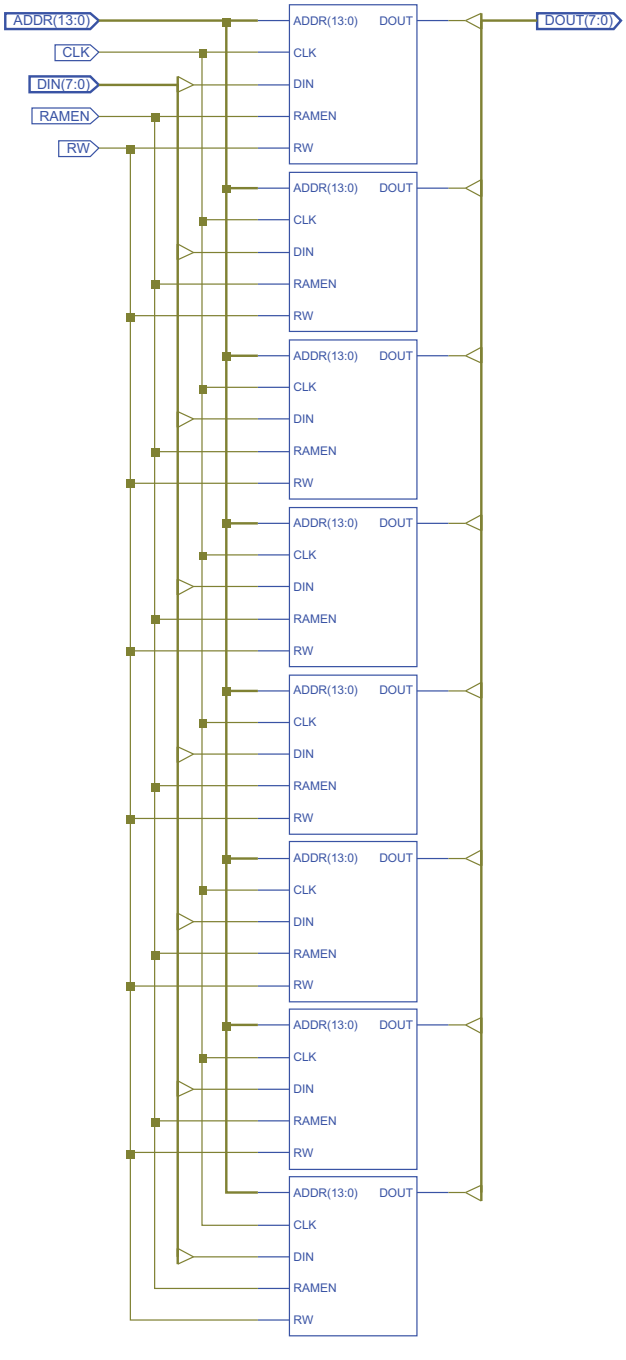


Figure 2.5: 16KB memory-block architecture