A Hardware Accelerator for Digital Holographic Imaging

Design Methodology and Implementation Aspects

Thomas Lenart

Thesis for the degree of
Licentiate in Engineering

2005
Department of Electroscience
Lund University, Sweden

Department of Electroscience
Lund University
Box 118, S-221 00 LUND
SWEDEN

This thesis is set in Computer Modern 11pt,
with the LaTeX Documentation System

March 2005

Abstract

Today, optical microscopes dominate medical and biological research laboratories. However, microscopes based on the principle of digital holography are emerging as an interesting alternative. The advantage with holography is that both phase and amplitude of the light is captured, while a traditional microscope only captures the amplitude. Phase information reveals material and object properties that cannot be seen in a traditional microscope, e.g. refractive index and the three-dimensional structure of an object. In the holographic microscope, images are captured using a digital image sensor and the optical lenses in a traditional microscope are replaced by reconstruction algorithms.

This thesis presents a hardware accelerator for image reconstruction in digital holographic imaging. The hardware accelerator executes a reconstruction algorithm, which transforms the light captured on a digital image sensor into visible images. The reconstruction algorithm, which is based on the FFT, is computationally demanding and requires a substantial amount of signal processing. Hence, a general-purpose computer is not capable of meeting the real-time constraints and a custom solution has been developed.

The hardware accelerator is based on a one-dimensional FFT, which has been implemented on silicon, fabricated, and verified. The FFT is used as a central building block in a two-dimensional signal processing datapath, optimized for executing reconstruction algorithms. Finally, the accelerator is integrated together with a microprocessor, memory controller, sensor and monitor interface to form a complete system. The system is synthesized to programmable logic and integrated into a prototype of the holographic microscope. To our knowledge, no other research project has combined these research areas before.

The important thing in science is not so much to obtain new facts
as to discover new ways of thinking about them

Sir William Bragg (1862 - 1942)

Contents

Preface

This thesis summarizes my academic work in the digital ASIC group at the department of Electroscience for the Licentiate degree in Engineering. The main contribution to the thesis is derived from the following publications:

> Thomas Lenart, Viktor Öwall, Mats Gustafsson, Mikael Sebesta, and Peter Egelberg, "Accelerating signal processing algorithms in digital holography using an FPGA platform," in *Proc. of International Conference on Field-Programmable Technology, ICFPT'03*, pp. 387–390, December 15-17, 2003, Tokyo, Japan.
>
> Thomas Lenart and Viktor Öwall, "A 2048 Complex Point FFT Processor using a Novel Data Scaling Approach," in *Proc. of International Symposium on Circuits and Systems, ISCAS'03*, vol. 4, pp. 45–48, May 25-28, 2003, Bangkok, Thailand.

Background information is derived from the following publications:

> Thomas Lenart and Viktor Öwall, "A Reconfigurable System for Image Reconstruction in Digital Holography," in *Proc. of Swedish System-on-Chip Conference, SSoCC'04*, April 13-14, 2004, Båstad, Sweden.
>
> Mats Gustafsson, Mikael Sebesta, Bengt Bengtsson, Sven-Göran Pettersson, Peter Egelberg, and Thomas Lenart, "High resolution digital transmission microscopy - a Fourier holography approach," in *Optics and Lasers in Engineering*, vol. 41, issue 3, pp. 553–563, March 2004.
>
> Thomas Lenart and Viktor Öwall, "A Pipelined FFT Processor using Data Scaling with Reduced Memory Requirements," in *Proc. of Norchip, NORCHIP'02*, pp. 74–79, November 11-12, 2002, Copenhagen, Denmark.

The following papers are also published, but not considered part of this thesis:

> Hugo Hedberg, Thomas Lenart, Henrik Svensson, "A Complete MP3 Decoder on a Chip," in *Proc. of Microelectronic Systems Education, MSE'05*, June 12-13, 2005, Anaheim, California, USA.
>
> Hugo Hedberg, Thomas Lenart, Henrik Svensson, Peter Nilsson and Viktor Öwall, "Teaching Digital HW-Design by Implementing a Complete MP3 Decoder," in *Proc. of Microelectronic Systems Education, MSE'03*, pp. 31–32, June 1-2, 2003, Anaheim, California, USA.

Acknowledgment

Thanks to all the people in the digital ASIC corridor, I really enjoy going to work every day. During my time at the department, there has never been a day without stimulating discussions regarding work, teaching and other less work-related topics.

I would like to thank my supervisor Viktor Öwall for his help and constructive criticism. Also many thanks to my associate supervisor Mats Gustafsson for explaining the reconstruction algorithms to me over and over again.

Several people has contributed to this work in different ways. Especially, I would like to thank Fredrik Kristensen, Matthias Kamuf, Henrik Svensson and Hugo Hedberg for commenting the contents and language in the thesis. I have probably removed a billion "the" by now and a bunch of other annoying stuffing words.

When it comes to chip design, I would first of all like to thank Anders Berkeman for being our mentor in ASIC design and also our universal guru on everything else. I would also like to thank Pontus Åström for a great LaTeX template. Over the years, all the people in the corridor has contributed in various ways to improve our design flow, including efforts put into developing courses and course material. I would like to thank all the people who has contributed, since this work has helped us all.

Finally I would like to thank the people working in the Digital Holography project. In particular, Mikael Sebesta for being the driving force behind the prototype and Bengt Bengtsson for rapidly and efficiently developing new PCBs to connect the strange and unknown analog world to the more understandable and logical digital world.

This work has been financed by the Competence Center for Circuit Design (CCCD).

Lund, 2005

Thomas Lenart

List of Acronyms

AHB	Advanced High-performance Bus
AGU	Address Generation Unit
ALU	Arithmetic Logic Unit
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
ASIP	Application-Specific Instruction Processor
CCD	Charge-Coupled Device
CMOS	Complementary Metal Oxide Semiconductor
CORDIC	Coordinate Rotation Digital Computer
CPU	Central Processing Unit
DAB	Digital Audio Broadcasting
DAC	Digital-to-Analog Converter
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
DIF	Decimation-In-Frequency
DIT	Decimation-In-Time
DRAM	Dynamic Random-Access Memory
DSP	Digital Signal Processor
DVB	Digital Video Broadcasting
FFT	Fast Fourier Transform
FIFO	First In, First Out
FIR	Finite Impulse Response

FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GDI	Graphics Device Interface
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
HDL	Hardware Description Language
IFFT	Inverse Fast Fourier Transform
IP	Intellectual Property
LSB	Least Significant Bit
LUT	Lookup Table
MAC	Multiply-Accumulate
MSB	Most Significant Bit
OFDM	Orthogonal Frequency Division Multiplexing
PCB	Printed Circuit Board
PE	Processing Element
RAM	Random-Access Memory
ROM	Read-Only Memory
SDF	Single-path Delay Feedback
SDRAM	Synchronous DRAM
SNR	Signal-to-Noise Ratio
SQNR	Signal-to-Quantization-Noise Ratio
SRAM	Static Random-Access Memory
SRF	Stream Register File
VGA	Video Graphics Array
VHDL	Very high-speed integrated circuit HDL
VLIW	Very Long Instruction Word

Chapter 1

Introduction

In 1981, IBM introduced a personal computer based on the Intel x86 architecture. In those days, computers were centralized around the microprocessor, not only for computations but also for graphics functionality and access to ports and disk drives. Computers were soon extended with additional processing units, or co-processors, instead of using a generic microprocessor for every task. One of the first co-processors available was a math processor supporting floating-point operations, a feature that was not yet available inside microprocessors. Over time, more specialized accelerators have been introduced to the market, to be able to meet the requirements in multimedia processing and other computationally demanding real-time applications.

Today, desktop computers contain specialized cards for all kinds of processing. Graphic cards render 3D-graphics in real-time, supporting z-buffers, α-channels, and texture mapping to name a few. Soundcards handle audio effects such as echo, reverb, and sound rendering using wave tables. In addition to this, plug-in cards containing programmable logic devices are now available, which makes it possible for anyone to build a custom accelerator. However, the design flow for utilizing the features of programmable logic inside a desktop computer is still a cumbersome procedure, requiring knowledge in both hardware- and software design. This will probably change in the future, simplifying the development of hardware accelerators to the same level as writing software programs today.

As processing elements become more complicated and powerful, another issue arises. Processors today can execute billions of operations per second. However, memory technologies has not evolved as fast and the result is an increasing gap between processing speed and memory bandwidth [1]. The problem significantly increases when designing high-performance custom hardware accelerators, which requires a substantially higher data rate. Improving the communication between the processing elements and memory could in many cases be more important than improving the hardware accelerator itself.

1.1 Research project

The research presented in this work is part of a larger multi-disciplinary project involving several research groups within the fields of optics, algorithm development, and digital hardware design. The project goal is to construct a microscope based on digital holography. The advantage of using holography is that not only the amplitude of

the light is captured, but also the phase. The phase information reveals object properties that cannot be seen in a traditional microscope, e.g. refractive index and the three-dimensional structure of an object.

This work presents a hardware accelerator for capturing and reconstructing images from a holographic microscope. To solve a specific problem, a custom hardware accelerator is more efficient than a general-purpose computer. Custom-designed accelerators can either be implemented in programmable logic, using a field programmable gate array (FPGA), or as an application-specific integrated circuit (ASIC). The result from this work has been integrated into a prototype of the holographic microscope, based on an FPGA platform. The research project on digital holography has resulted in a start-up company, *Phase Holographic Imaging AB*, or PHI.

1.2 Thesis outline

This thesis presents the work on a hardware accelerator for digital holography. Since this work is closely connected to the application, a brief background on holography is presented in Chapter 2. This includes traditional holography, digital holography, and how to reconstruct information from captured holographic images. This chapter also specifies the real-time requirements for the hardware accelerator. Chapter 3 gives an introduction to different processing units and their efficiency, followed by an overview of memories, discussing storage management, bandwidth limitations, and access pattern scheduling. The implementation work is divided into three chapters. The fast Fourier transform (FFT) is a central part of many reconstruction algorithms. Therefore, the work on a one-dimensional, high-precision pipeline FFT core is presented in Chapter 4. In Chapter 5, the FFT is reused when designing a two-dimensional FFT and constructing a pipeline with the required computational units for image reconstruction. The optimizations based on the design decisions in Chapter 5 are presented in Chapter 6. Chapter 7 presents the integration of the hardware accelerator into an embedded system, connecting external components for capturing and viewing holographic images. As a part of Chapter 7, a prototype of a complete holographic microscope is presented. The prototype includes the embedded system together with the optics, sensor, and interface to an external computer. Finally, conclusions and future work are presented in Chapter 8 and Chapter 9.

Chapter 2

Holography

In 1947, the British scientist Dennis Gabor invented a method to photographically create a three-dimensional recording of a scene [2]. However, Gabor needed a light source with a single frequency and a constant phase shift, properties known as *monochromatic* and *coherent* light, respectively. Since this device was not available at the time, he initially used a mercury lamp with spatial and temporal filtering, significantly degrading the image quality. A coherent light source was first discovered in 1960, an invention named *Light Amplification by Stimulated Emission of Radiation*, or Laser for short.

2.1 Traditional Holography

A holographic setup is based on a coherent light source, shown in Figure 2.1. The *interference* pattern between light from a reference wave and reflected light from an object illuminated with the same light source is captured on a photographic film. Interference between two wave fronts cancels or amplifies the light in each point on the holographic film. This is called *constructive* and *destructive* interference respectively.

A recorded hologram has certain properties that distinguish it from a traditional photograph. In a normal camera, the amplitude of the light is captured and the developed photography is directly visible. The photographic film in a holographic setup captures the interference, or phase difference, between two waves. Hence, depth information is stored in the hologram. By illuminating the developed photographic film with the same reference light as used during the recording phase, the original image is *reconstructed* and appears three-dimensional.

In a camera, a lens is used to focus the light onto the film. The location of the lens determines the focus position, and the characteristics of the lens determine the focal depth. This causes focus to be sharp only at a certain distance, limited by the properties of the lens. Since a hologram can be captured without lenses, the focal depth is high, and focus is extremely sharp regardless of the distance to the object.

2.2 Digital holography

In digital holography, a high-resolution image sensor replaces the photographic film. A computer algorithm is used to calculate a visible image based on the digital recordings [3]. One advantage over traditional holography is that image capturing only takes a

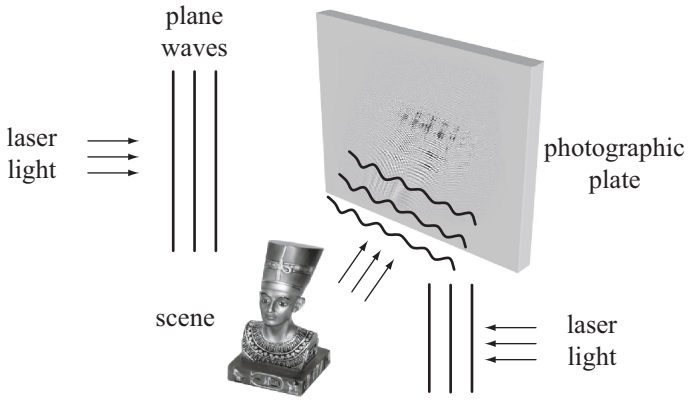


Figure 2.1: The reflected light from the scene and the reference light creates an interference patten on a photographic film.

fraction of a second, instead of developing a photographic film. The downside is that the sensor resolution is about 300 pixels/mm, whereas the photographic film contains 3000 lines/mm. However, the resolution of digital image sensors is continuously increasing, improving the quality and usefulness of digital holography.

2.3 A microscope based on digital holography

Digital holography can be used as an alternative to conventional microscopy and has several interesting features, e.g. improved depth focus and the possibility of generating three-dimensional and phase contrast images [4]. Since not only amplitude but also phase information of the object is recorded, special material characteristics can be determined that are useful in for instance biomedical applications [5]. A few examples are to determine the refractive index, object thickness, total volume, or volume distribution. While a conventional microscope uses lenses, the holographic setup uses a signal processing algorithm, where a parameter change in the algorithm will instantly reconfigure the *digital lens* into any shape or focal length. Figure 2.2 shows the holographic microscope setup.

2.4 Reconstruction algorithms

A reconstruction algorithm processes the captured wave patterns on the sensor to create a visible image. With three exposures, images of the hologram, object, and reference are gathered. First, the hologram is captured by measuring the interference between the object and the reference source. Then, by blocking either the object or reference beam, the reference and object light is measured. By subtracting the reference light ψ_r and object light ψ_o from the hologram ψ_h, the only remaining component is the interference pattern ψ as

$$\psi(\boldsymbol{\rho}) = \psi_h(\boldsymbol{\rho}) - \psi_o(\boldsymbol{\rho}) - \psi_r(\boldsymbol{\rho}). \tag{2.1}$$

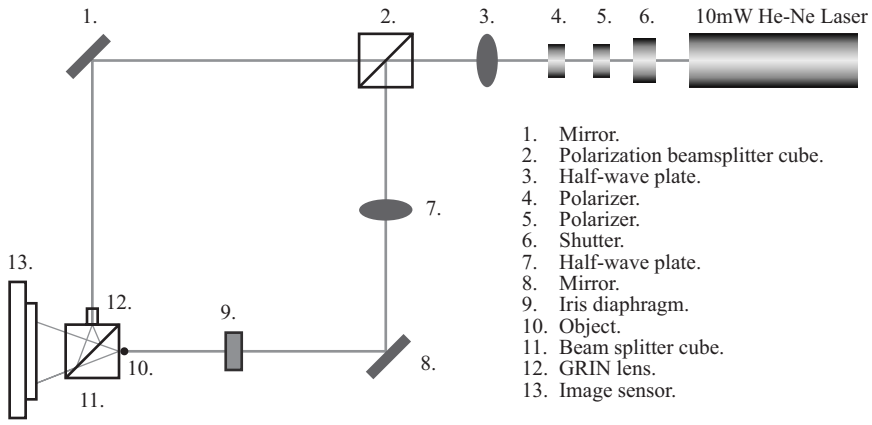


Figure 2.2: The experimental holography setup. The reference, object, and interference pattern are captured on a high-resolution digital image sensor, instead of using a photographic film as in traditional holography. The images are captured by blocking the reference or object beam.

To simplify the equations, the vector $\boldsymbol{\rho}$ is used for specifying the (x, y) position on the sensor. A visible image is reconstructed from $\psi(\boldsymbol{\rho})$, which is the object field in the sensor plane, by retrofocusing the light to the object plane. The reconstruction algorithm, or inversion algorithm, can retrofocus the light captured on the sensor to an arbitrary object plane, which makes it possible to change focus position in the object. This is equivalent to manually moving the object up and down in a traditional microscope.

Two reconstruction algorithms are presented in the following sections, first the convolution algorithm and then a simplified version based on the far-field approximation. The simplified version requires less computational power and is therefore of high interest, at the penalty of slightly reduced image quality. A comparison between the algorithms is presented in Section 2.4.3.

2.4.1 Convolution algorithm

The convolution algorithm is compute-intensive but accurate. Images are reconstructed by computing the Rayleigh-Sommerfeld diffraction integral as

$$\Psi(\boldsymbol{r}') = \int_{\text{sensor}} \psi(\boldsymbol{\rho}) \mathrm{e}^{-\mathrm{i}k|\boldsymbol{r}-\boldsymbol{r}_r|} \mathrm{e}^{-\mathrm{i}k|\boldsymbol{r}-\boldsymbol{r}'|} \,\mathrm{dS}_\rho. \tag{2.2}$$

Figure 2.3 illustrates the relation between the sensor and object plane. By specifying a point of origin, $\boldsymbol{r}'$ represents the vector to any point in the object plane, $\boldsymbol{r}$ is the distance to any point in the sensor plane and $\boldsymbol{r}_r$ is the position of the reference light source. Accordingly, $|\boldsymbol{r} - \boldsymbol{r}_r|$ is the distance between a point on the sensor and the reference position while $|\boldsymbol{r} - \boldsymbol{r}'|$ represents the distance between points in the object plane to pixels in the sensor plane. The integral can be expressed as a convolution, which in the frequency domain can be simplified to a matrix multiplication as

$$\Psi(\boldsymbol{r}') = \psi' * G = \mathcal{F}^{-1}(\mathcal{F}(\psi') \cdot \mathcal{F}(G)), \tag{2.3}$$

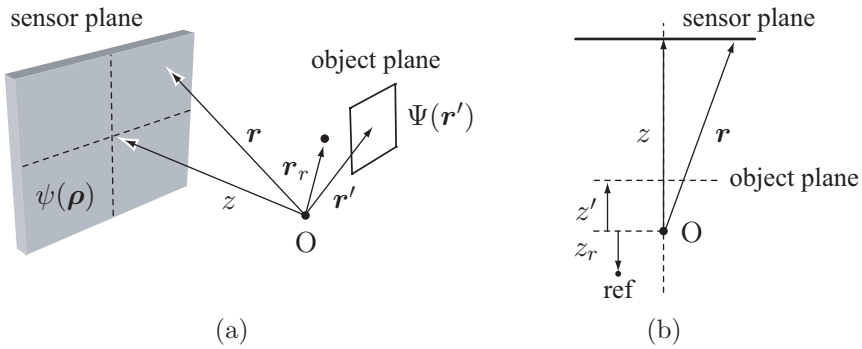


Figure 2.3: (a) Definition of vectors from origin of coordinates to the sensor surface, reference point, and object plane. (b) Sensor and object planes. Modifying the z-position of the object plane changes focus position.

where

$$\psi'(\boldsymbol{\rho}) = \psi(\boldsymbol{\rho})\mathrm{e}^{-\mathrm{i}k\sqrt{|z-z_r|^2+|\boldsymbol{\rho}-\boldsymbol{\rho}_r|^2}}$$

and

$$G(\boldsymbol{\rho}) = \mathrm{e}^{-\mathrm{i}k\sqrt{|z-z'|^2+|\boldsymbol{\rho}|^2}}.$$

The three-dimensional vectors representing the distance between the sensor and object planes has been divided into a z component and an orthogonal two-dimensional vector $\boldsymbol{\rho}$ representing the x and y positions as

$$\boldsymbol{r} = \boldsymbol{\rho} + z\hat{z}.$$

The distance z' specifies the location of the image plane to be reconstructed, whereas z is a constant value. The discrete version of the integral with an equidistant grid, equal to the sensor pixel size, generates a discrete convolution that can be evaluated with the FFT. The size of the two-dimensional FFT must be at least the sum of the number of the sensor size and the object size in each direction. Higher resolution is achieved by shifting the coordinates a fraction of a pixel size Δx, and combining the partial results into a larger holographic image. A larger image requires several iterations and increases the reconstruction time with a factor N^2 for a sub-pixel distance of $(\Delta x/N, \Delta y/N)$. One image reconstruction requires three FFT calculations, while each additional evaluation only requires two FFT calculations, since only $G(\boldsymbol{\rho})$ will change. For fast low-resolution reconstructions, $G(\boldsymbol{\rho})$ can be precalculated to avoid one FFT operation. The result from the calculations, as well as the recorded images, can be found in Figure 2.4.

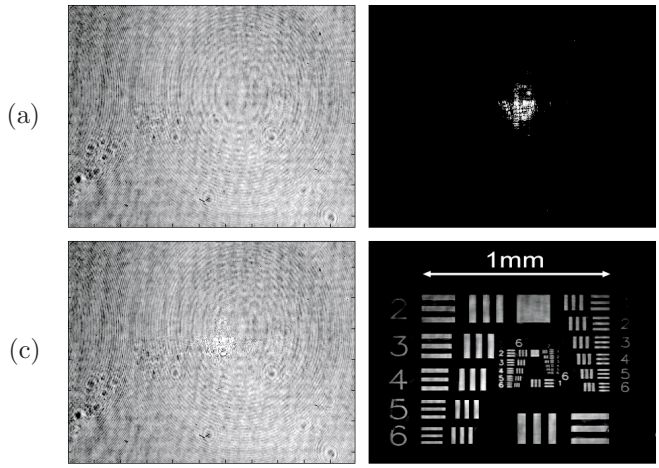


Figure 2.4: (a) The reference image ψ_r. (b) The object image ψ_o. (c) The interference pattern ψ_h. (d) Reconstructed holographic image Ψ.

2.4.2 Simplified far-field approximation

The convolution algorithm requires three two-dimensional FFTs to be evaluated as expressed in Equation 2.3. First, $\psi'(\boldsymbol{\rho})$ and $G(\boldsymbol{\rho})$ are transformed into the frequency domain. After a matrix multiplication between the two, the result is transformed back to the time domain. This section will present a simplified reconstruction algorithm that only requires a single FFT to be evaluated.

By observing that $|\boldsymbol{r}'| \ll |\boldsymbol{r}|$, the Fraunhofer or *far-field* approximation simplifies the distance between the sensor plane and object plane as

$$|\boldsymbol{r} - \boldsymbol{r}'| \approx r - \boldsymbol{r} \cdot \boldsymbol{r}'/r.$$

This assumption simplifies the integral expression in Equation 2.2 to

$$\begin{aligned}
\Psi(\boldsymbol{r}') &= \int_{\text{sensor}} \psi(\boldsymbol{\rho}) \mathrm{e}^{-\mathrm{i}k|\boldsymbol{r}-\boldsymbol{r}_r|} \mathrm{e}^{-\mathrm{i}k|\boldsymbol{r}-\boldsymbol{r}'|} \,\mathrm{dS}_{\boldsymbol{\rho}} \\
&\approx \int_{\text{sensor}} \psi(\boldsymbol{\rho}) \mathrm{e}^{-\mathrm{i}k(|\boldsymbol{r}-\boldsymbol{r}_r|-r)} \mathrm{e}^{-\mathrm{i}k\boldsymbol{r}\cdot\boldsymbol{r}'/r} \,\mathrm{dS}_{\boldsymbol{\rho}} \\
&= \int_{\text{sensor}} \psi(\boldsymbol{\rho}) \mathrm{e}^{-\mathrm{i}k(|\boldsymbol{r}-\boldsymbol{r}_r|-r)} \mathrm{e}^{-\mathrm{i}kzz'/r} \mathrm{e}^{-\mathrm{i}k\boldsymbol{\rho}\cdot\boldsymbol{r}'/r} \,\mathrm{dS}_{\boldsymbol{\rho}} \\
&= \int_{\text{sensor}} \psi'(\boldsymbol{\rho}) \mathrm{e}^{-\mathrm{i}k\boldsymbol{\rho}\cdot\boldsymbol{r}'/r} \,\mathrm{dS}_{\boldsymbol{\rho}}.
\end{aligned}$$

Except for the factor $1/r$, this is the definition of the two-dimensional FFT. Before carrying out the Fourier transformation, $\psi'(\boldsymbol{\rho})$ has to be calculated as

$$\psi'(\boldsymbol{\rho}) = \psi(\boldsymbol{\rho}) \mathrm{e}^{-\mathrm{i}k(|\boldsymbol{r}-\boldsymbol{r}_r|-r)} \mathrm{e}^{-\mathrm{i}kzz'/r}.$$

(a) (b)

Figure 2.5: Comparison of image quality when reconstructing the inner pattern on the test target. (a) Convolution algorithm with 6 interpolations in each direction, requiring 73 FFTs to be evaluated. (b) Simplified far-field approximation evaluated with a single FFT.

The expression can be further simplified by removing the first exponential term since it will only affect the location of the object in the reconstructed image. Instead, the coordinates of the object location can be modified after reconstruction. The final result is an image reconstruction algorithm containing only one FFT as

$$\Psi(\boldsymbol{r}') \approx \mathcal{F}(\psi(\boldsymbol{\rho})\mathrm{e}^{-\mathrm{i}kzz'/r}). \tag{2.4}$$

2.4.3 **Algorithm comparison**

Producing a visible image requires three images to be captured and processed using a reconstruction algorithm. The convolution algorithm needs three FFTs to be evaluated for the first image and two for each additional interpolation, whereas the simplified version only requires one FFT. However, the simplified reconstruction algorithm will degrade the image quality due to the mathematical approximations. Figure 2.5 shows a quality comparison between the two algorithms, reconstructing the inner pattern on a test target from Figure 2.4(d). There is a difference in smoothness, but the resolution is about the same. The convolution algorithm is here interpolated 6 times in each direction, which requires $1 + 2 \times 6^2 = 73$ FFTs (one extra FFT in the first iteration). The simplified reconstruction algorithm still only requires one FFT to be evaluated. Comparing the image quality and computational requirements, the simplified algorithm is chosen for implementation.

2.5 System requirements

For the holographic microscope to be useful, images have to be reconstructed in real-time. The term *real-time* is somewhat vague and calls for further explanation. For the human eye to interpret a series of images as motion video, a rate of about 25 frames per second (fps) is required. In a microscope, the response time between moving a sample and getting the visual feedback from the motion should be as short as possible. A longer response time is justified for images with higher resolution and quality. The system frame rate is limited by two factors, the sensor frame rate and the reconstruction time. These limitations are discussed in the following section, also specifying the requirements for the hardware accelerator in terms of resolution, frame rate, and other properties.

2.5.1 Digital sensors

The frame rate of image sensors is a limiting factor for the current application. In digital cameras, it is often not critical that it takes one second to transfer the high-resolution image from the sensor array to the memory. The delay is due to the number of pixels on the sensor array, currently somewhere in the range of 5-10 million pixels. However, in a digital video camera the real-time constraints require 25 fps for the images to appear as motion video. To satisfy this condition, resolution is substantially reduced, normally in the range of 0.5-1 million pixels. Since the images are continuously changing, the observer is less sensitive to the low resolution.

A problem to meet the real-time constraints in digital holography is that the best of two worlds needs to be combined, high resolution and high frame rate. However, lower resolution is sufficient when scanning and moving the sample. When the interesting part of the sample is located, a high-resolution image can be reconstructed.

CMOS versus CCD

Digital image sensors can be divided into two groups, Charge-Coupled Devices (CCD) and Complementary Metal-Oxide-Semiconductor (CMOS). CCD sensors are characterized by high resolution and quality but also high manufacturing cost. They require external circuits to access the sensor array and convert the analog charge into a digital value, shifted out row-by-row. A problem with CCD sensors is that charges leak between adjacent pixels, causing a saturated pixel to *bloom* (overflow) and affect closely located pixels.

CMOS sensors are still not available in the same high resolution as CCD sensors. However, they have many other advantages. CMOS sensors can be fabricated in standard manufacturing facilities, which significantly reduces the cost and increases the yield. Circuitry for accessing the sensor array is integrated on-chip, avoiding external components. The on-chip circuitry also enables random access to regions of interest, rather than reading out the complete sensor array. CMOS sensors have anti-blooming features, which is very important in digital holographic imaging. Finally, the power consumption is lower for CMOS sensors, but only relevant to mobile and battery-driven applications.

2.5.2 Image reconstruction

In this section, an initial specification is outlined based on the selected algorithm. One parameter is the sensor size. A larger sensor requires more processing, but generates a higher image quality. By evaluating available image sensors on the market, the maximum size of the array is set to 2048×2048 pixels, which also specifies the size of the Fourier transform. Another parameter is the clock frequency, which linearly affects the computation time. Initially, the clock frequency is estimated to 250 MHz, a reasonable and modest number for any modern fabrication process. However, when designing an FPGA platform for prototyping and evaluation, a much lower value has to be assumed.

- **FFT size** – 2048 points.
- **Frame rate** – 25 fps.
- **Clock frequency** – 250 MHz.
- **Architectural selection** – To be decided.

The transform size, frame rate, and clock frequency have been specified, but the architectural decisions are still open. Selecting an architecture is a trade-off between computation time and hardware resources. According to the specification, each frame has a clock cycle (cc) budget of

$$\frac{250\,\text{MHz}}{25\,\text{fps}} = 10\,\text{Mcc}.$$

With these assumptions, a two-dimensional FFT has to be evaluated in 10 million clock cycles. Due to the extremely large transform size, implementation of the two-dimensional FFT is separated into one-dimensional FFTs, first applied over the rows and then over the columns [6]. Hence, 2×2048 transforms of size 2048 must be computed. The requirements for the one-dimensional FFT is therefore

$$\text{FFT}_{\text{cc}} = \frac{10\,\text{Mcc}}{2 \times 2048} \approx 2400\,\text{cc}. \tag{2.5}$$

When selecting the architecture of the one-dimensional FFT, the computation time must be in the range of 2400 clock cycles. The architecture evaluation and selection is presented in Chapter 4.

Chapter 3

Processing and memory

This chapter presents a brief background on processors and memories as a motivation to build application-specific hardware accelerators. The chapter is divided into general-purpose, special-purpose, and application-specific processing. Microprocessors are limited by the sequential nature of execution. Designing an application-specific accelerator can significantly increase the computational efficiency by exploiting the parallelism in an algorithm.

The second part of this chapter addresses memories. As the speed of execution increases, the amount of data transferred to and from the storage space increases as well. Internal memory is fast but expensive in terms of area. External memory provides a larger storage space, but data must be transferred efficiently to maximize memory bandwidth and minimize latency. The non-uniform access time of modern high-capacity memories require data to be transferred in blocks, known as burst transfers. Memory access patterns are therefore extremely important, and sometimes require special scheduling units.

3.1 General-purpose processing

The most common processing unit is probably the microprocessor, the central core in desktop computers, workstations, laptops, and servers. Common for all processors is their programmability. The microprocessor, or central processing unit (CPU), is designed for general-purpose processing to solve all kinds of tasks. Since the execution is sequential, it has limited ability of exploiting parallelism on an algorithmic level. However, there are several ways of finding and exploiting parallelism on an instruction level. The implementation differs between processor architectures, e.g. very long instruction words (VLIW) containing multiple instructions, superscalar processors with several computational pipelines, and even processors supporting thread parallelism (hyper-threading).

VLIW processors exploit the instruction level parallelism by loading and executing parallel instructions. Every instruction word contains a set of independent instructions, which can be executed in parallel. The instruction level parallelism is determined at compile-time and requires more complex compilers. Examples of VLIW architectures are the 128-bit Crusoe and 256-bit Efficēon processors from Transmeta [7]. Another example is the TriMedia core from Philips.

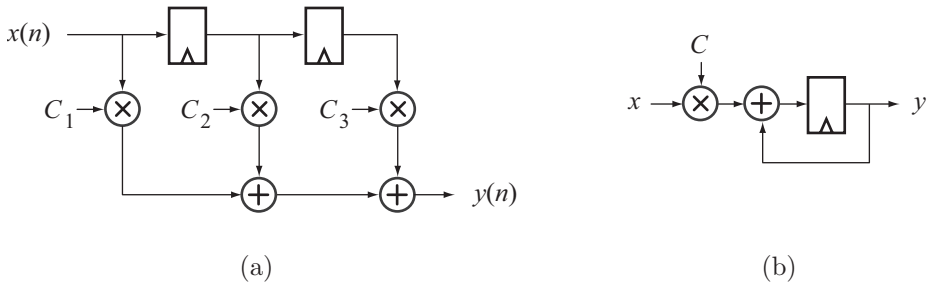


Figure 3.1: (a) Block diagram of a 3-tap FIR filter. (b) MAC unit suitable for implementing a FIR filter.

In contrast to the VLIW architecture, *superscalar* processors dynamically determines how to parallelize the execution. This is possible by analyzing the program and even reorder instructions to find more concurrency. Processors such as the Intel Pentium, AMD Athlon, and PowerPC are based on superscalar pipelines. Pentium 4 also supports *hyper-threading*, i.e. finding and exploiting parallelism between separate threads.

3.2 Special-purpose processing

Despite all improvements and special features, microprocessors are still generic computational units. For special-purpose processing, several alternatives to microprocessors exist. For example, a digital signal processor (DSP) includes features to improve execution of signal processing algorithms, while stream processors enhance the processing time in media application such as video compression and graphic rendering.

3.2.1 Digital signal processors

A digital signal processor has much in common with a microprocessor but contains additional features. In signal processing algorithms, a set of frequently used operations can be isolated. A common example is multiplication followed by accumulation, present in for instance FIR filters [8]. In a DSP, this operation is executed in a single clock cycle using the multiply-accumulate (MAC) operation, as shown in Figure 3.1. Another useful feature is the possibility of scaling a result from the arithmetic unit. By connecting a barrel shifter to the output, scaling can be applied without time penalty. In signal processing, overflow causes serious problems when the values exceed the maximum wordlength. In a conventional CPU, values wrap around on overflow and corrupt the computation result. DSPs often use saturation logic, preventing the value to wrap and cause less damage to the final result. In the MAC unit, overflow is avoided by adding guard bits to the accumulation register.

The memory architecture of a DSP differs from a CPU. Consider the MAC operation to execute in one clock cycle. In this case, two values are required every clock cycle, which demands two memory accesses in a single clock cycle. Therefore, DSPs often have more than one on-chip memory, single- or dual-ported, to supply the arithmetic units with data. In addition to the internal memories, some DSPs even have more than one external memory interface. Another issue is computational overhead due to address

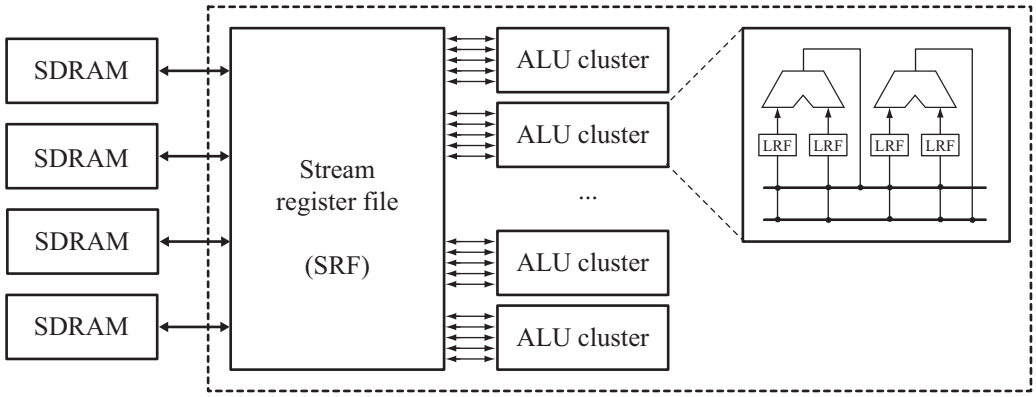


Figure 3.2: The *Imagine* stream processor. Kernels are operating concurrently, communicating through the stream register file (SRF).

generation. Instead of calculating addresses using an ALU, and wasting computational power, special address generation units (AGU) takes care of memory addressing. This is useful when addresses appear, for example, bit-reversed or for modulus counters. Instead of calculating complex addresses, these features are supported by the AGU.

3.2.2 **Stream processors**

The number of applications with multimedia processing continuously increases. Applications span from audio decompression such as MP3, to computationally demanding real-time video compression and three-dimensional graphics rendering. Traditional CPUs or DSPs are not suitable for this task, because they lack the required computational efficiency for real-time applications.

Media applications often involve operations on streams of data with high level of local dependency. This enables the possibility to parallelize media applications and design clusters of concurrent processors with separate local storage, referred to as a stream processor [9]. Each cluster contains kernels with a local register file (LRF), enabling a high bandwidth during execution. The kernels are connected to a stream register file (SRF) to handle communication between clusters. The SRF also communicates with external memories, storing global data available to all kernels. As an example, the *Imagine* stream processor [10] contains eight ALU clusters, connected to an SRF. An overview of the system can be found in Figure 3.2.

The drawback with conventional CPUs is their fixed structure with a computational unit, register file, cache, and main memory. In a stream processor, each arithmetic kernel works on local data, while streams can pass between kernels using a SRF. Therefore, data is exchanged where bandwidth is high, i.e. close to the kernels. Instead of communicating through the main memory, data is shuffled among kernels. This approach reduces the bandwidth between memory and processing unit and increases the processing capacity.

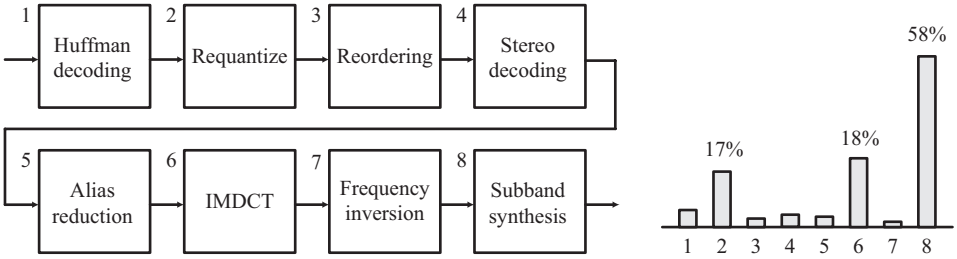


Figure 3.3: The MP3 decoding process and software profiling showing the time required in each step. Profiling is based on the ISO/IEC 11172-3 floating-point reference model.

3.3 Application-specific processing

Application-specific means that the architecture is designed to optimize a specific function or algorithm. The computational efficiency is likely to improve if available hardware resources match the algorithmic requirements. As an example of application-specific processing, the MP3 decoding algorithm is considered [11]. MP3, or MPEG1 layer 3, is a compression algorithm for sound. A block diagram is presented in Figure 3.3, divided into eight functional blocks. When executed on a microprocessor, blocks are processed one at a time, decoding the bitstream into audio samples. The graph in Figure 3.3 illustrates the software profiling [12]. The profiling shows the processing time for each block, and the most time-consuming blocks are the first candidates for hardware acceleration.

There are several approaches to accelerate the decoding algorithm, ranging from accelerating commonly used operations, to complete functions, or implement the entire algorithm in hardware. When mapping the algorithm directly to hardware, flexibility is lost but computational efficiency gained. Compared to a programmable solution, there is no longer an overhead in reading and executing instructions.

3.3.1 Extended instruction set

One approach to acceleration is to extend the instruction set of an ordinary microprocessor, referred to as an application-specific instruction set processor (ASIP). With basic knowledge about the MP3 decoding blocks, it would be suitable to extend the processor with instructions such as a MAC unit for the subband synthesis, or a butterfly operation for the IMDCT. However, changing the instruction pipeline requires a modified processor architecture. An easier and more common approach is to place the hardware extension parallel to the processor pipeline, as shown in Figure 3.4(a).

3.3.2 Accelerator on bus

Another approach is to isolate complete functions and build application-specific hardware accelerators. In a computer system, hardware accelerators are usually connected to the internal bus. Figure 3.4(b) shows a configuration with microprocessor and accelerator on a bus. Many companies develop and market accelerators for various bus architectures, often referred to as intellectual property (IP).

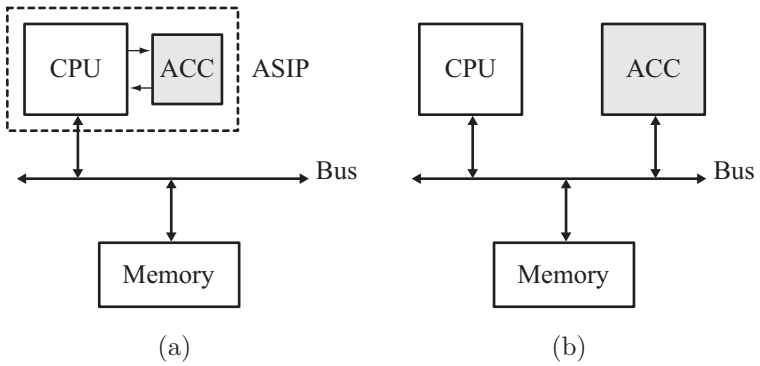


Figure 3.4: (a) Co-processor connected to a CPU pipeline. (b) Hardware accelerator connected to a bus.

The profiling in Figure 3.3 reveals that the microprocessor spends more than half the time on subband synthesis. By mapping this function to hardware the overall processing time can be reduced by more than 50%, according to the software profiling. The hardware accelerator and microprocessor operates concurrently, hence total execution time is the maximum of the two. A well-designed hardware accelerator must be balanced against the microprocessor for maximum performance with minimum resources. In this case, the accelerator must be fast enough to finish execution in the same time as the microprocessor, else idle time is introduced. If the accelerator is too fast, it requires more resources than actually necessary, without gain in the total execution time. When choosing to accelerate other functions of the algorithm as well, the time- and resource allocation have to be analyzed again. This is referred to as hardware/software partitioning.

3.4 Memory and bandwidth

Data processing requires storage space for input vectors, intermediate values, and results. A faster processing unit requires a higher bandwidth to transfer information to and from memory. When the efficiency of computational elements increases, it also requires more efficient memory management. Modern architectures are based on a memories hierarchy, with the fastest memory located close to the computational units, see Figure 3.5. In a microprocessor, this memory is referred to as a cache, a temporary storage space for frequently accessed information. The high-speed memory is located on-chip and can sustain high bandwidth requirements. When more storage space is required, it must be located off-chip. This limits the bandwidth in several ways, which is further discussed in Section 3.4.2. Accessing off-chip memory is time and power expensive, hence the communication with external memory should be minimized.

3.4.1 Internal memories

Along with the computational units in an integrated circuit, on-chip memory stores frequently accessed information and intermediate results between calculations. In a

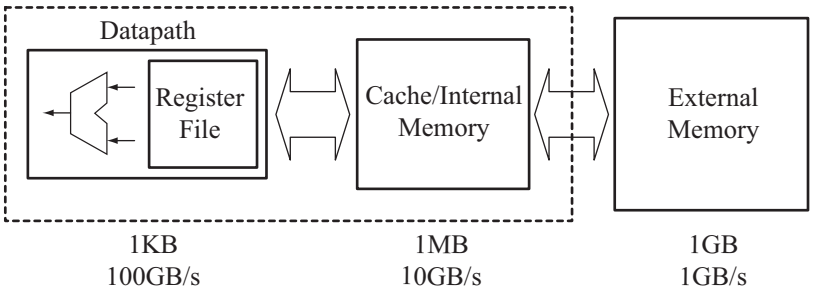


Figure 3.5: The memory hierarchy.

microprocessor there are several hierarchies of memory. Closest to the executional unit is the *register file*, a bank of registers for storing results. The register file stores only data and not instructions, and is usually implemented using a multi-port memory or a small bank of flip-flops.

Caches and internal memories improve both the bandwidth and the latency for a data access. Caches and other on-chip memories are often based on static random access memory (SRAM). For SRAMs, the access time to arbitrary memory elements is uniform, and the pattern of which elements are accessed is not of importance from a performance perspective. For demanding applications, several separate memories could supply the computational units with data.

3.4.2 External memories

Due to limited amount of on-chip memory, external memory is required when a larger storage space is needed. External memory banks are designed to store significantly more information, and hence demands a higher density of memory elements. High-density memory devices are based on dynamic random access memory (DRAM) cells. Today, different versions of *synchronous* DRAM (SDRAM) is commonly used in consumer electronics. These devices are burst oriented, and have a non-uniform access time. Still, many applications have substantial memory requirements [13], and have to cope with the non-uniform access time by reordering memory operations or physical location of data in the memory [14].

External memories increase the storage space, but there are several penalties. Since the memory is placed off-chip, internal signals must be routed from a pad (external connection), over a printed circuit board (PCB) and into another circuit. This will significantly increase the delay and limit the maximum data rate. The number of data bits is also limited, since each connection requires separate pads, which are expensive in terms of chip area. The function of the pad is to amplify weak on-chip signals to communicate with off-chip components. Therefore, sending a signal off-chip requires more current than internal communication, and results in higher power consumption.

Modern DRAMs are based on several individual *memory banks*, and the memory address is separated into *rows* and *columns*, as shown in Figure 3.6. The three-dimensional organization of modern memory devices results in non-uniform access time. The memory banks are accessed independently, but the two-dimensional memory array in each

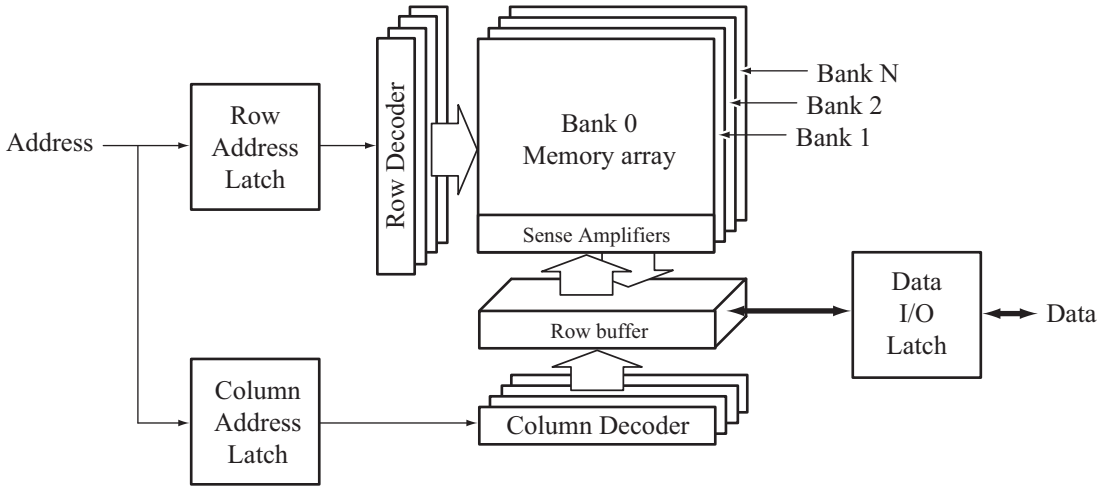


Figure 3.6: Modern SDRAM is divided into three dimensions: banks, rows and columns.

bank is more complicated. To access a memory element, the corresponding row has to be selected. Data in the selected row is then transferred to the *row buffer*. From the row buffer, data is accessed at high-speed and with uniform access time for any access pattern. When data from a different row is needed, the current row has to be closed, by precharging the bank, before the next row can be activated and transferred to the row buffer.

Memory bandwidth of modern DRAM is highly dependent on the access pattern. Accessing different banks or columns inside a row has a low latency, while accessing data in different rows has a high latency. When processing several memory streams, a centralized memory access scheduler can optimize the overall performance by reordering the memory accesses [15]. Latency for individual transfers may increase, but the goal is to minimize average latency and maximize overall throughput.

Chapter 4

A flexible FFT core

This chapter presents the implementation of a flexible pipeline FFT core, which is used as the central building block in a hardware accelerator for digital holographic imaging [16]. The design is based on a flexible and highly configurable hardware description, which supports several different architectures. The FFT core can be configured at run-time to calculate forward or inverse transforms for any number of points up to the maximum transform size, which is selected at design time.

The chapter starts with a description of the discrete Fourier transform (DFT) before introducing the more efficient fast Fourier transform (FFT). Different architectures for hardware implementation are presented, followed by scaling alternatives and important considerations when dealing with pipeline architectures. The implementation is then described in detail along with simulations and results. Finally, measurements from the fabricated design are presented.

4.1 Discrete Fourier transform

The discrete Fourier transform is a commonly used operation in digital signal processing [6]. Typical applications are linear filtering, correlation, and spectrum analysis. The Fourier transform is also found in modern communication systems using digital modulation techniques such as orthogonal frequency division multiplexing (OFDM). OFDM is used in several wireless network standards, for example 802.11a [17] and 802.11g [18], as well as in audio and video broadcasting using DAB and DVB. Another example is GPS receivers, that use the DFT to modify the spectrum and suppress interference [19].

The DFT is defined as

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \qquad 0 \leq k < N, \tag{4.1}$$

where

$$W_N = \mathrm{e}^{-\mathrm{i}2\pi/N}. \tag{4.2}$$

Evaluating Equation 4.1 requires N MAC operations for each transformed value in X, or N^2 operations for the complete DFT. Changing transform size significantly affects

computation time, e.g. calculating a 1024-point Fourier transform requires a thousand times more work than a 32-point DFT. An efficient way of computing the DFT is therefore of great importance, and presented in the next section.

4.2 FFT algorithms

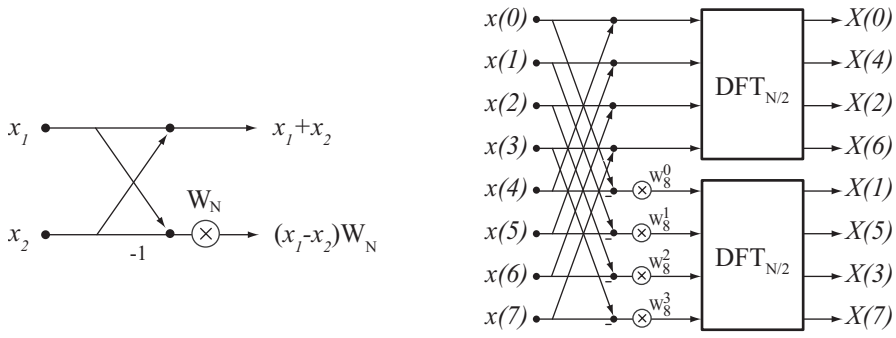
A more efficient way of computing the DFT is to use the FFT [20]. The FFT is a decomposition of an N-point DFT into successively smaller DFT transforms. The concept of breaking down the original problem into smaller sub-problems is known as a divide-and-conquer approach. The original sequence can for example be divided into $N = r_1 \cdot r_2 \cdot ... \cdot r_q$ where each r is a prime. For practical reasons, the r values are often chosen equal, creating a more regular structure. As a result, the DFT size is restricted to $N = r^q$, where r is called *radix* or decomposition factor. Most decompositions are based on a radix value of 2, 4 or even 8 [21]. Consider the following decomposition of Equation 4.1, known as radix-2

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) W_N^{kn} \\ &= \sum_{n=0}^{N/2-1} x(2n) W_N^{k(2n)} + \sum_{n=0}^{N/2-1} x(2n+1) W_N^{k(2n+1)} \\ &= \underbrace{\sum_{n=0}^{N/2-1} x_{even}(n) W_{N/2}^{kn}}_{DFT_{N/2}(x_{even})} + W_N^k \underbrace{\sum_{n=0}^{N/2-1} x_{odd}(n) W_{N/2}^{kn}}_{DFT_{N/2}(x_{odd})}. \end{aligned} \tag{4.3}$$

The original N-point DFT has been divided into two $N/2$ DFTs, a procedure that can be repeated over again on the smaller transforms. The complexity is thus reduced from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log_2 N)$. The decomposition in Equation 4.3 is called *decimation-in-time* (DIT), since the input $x(n)$ is decimated with a factor of 2 when divided into an even and odd sequence. Combining the result from each transform requires a scaling and add operation. Another common approach is known as *decimation-in-frequency* (DIF), splitting the input sequence into $x_1 = \{x(0), x(1), ..., x(N/2-1)\}$ and $x_2 = \{x(N/2), x(N/2+1), ..., x(N-1)\}$. The summation now yields

$$\begin{aligned} X(k) &= \sum_{n=0}^{N/2-1} x(n) W_N^{kn} + \sum_{n=N/2}^{N-1} x(n) W_N^{kn} \\ &= \sum_{n=0}^{N/2-1} x_1(n) W_N^{kn} + \underbrace{W_N^{kN/2}}_{(-1)^k} \sum_{n=0}^{N/2-1} x_2(n) W_N^{kn}, \end{aligned} \tag{4.4}$$

where $W_N^{kN/2}$ can be extracted from the summation since it only depends on the value of k, and is expressed as $(-1)^k$. This expression divides, or decimates, $X(k)$ into two



(a) (b)

Figure 4.1: (a) Butterfly operation and scaling. (b) The radix-2 decimation-in-frequency FFT algorithm divides an N-point DFT into two separate $N/2$-point DFTs.

groups depending on whether $(-1)^k$ is positive or negative. That is, one equation calculate the even values and one calculate the odd values as in

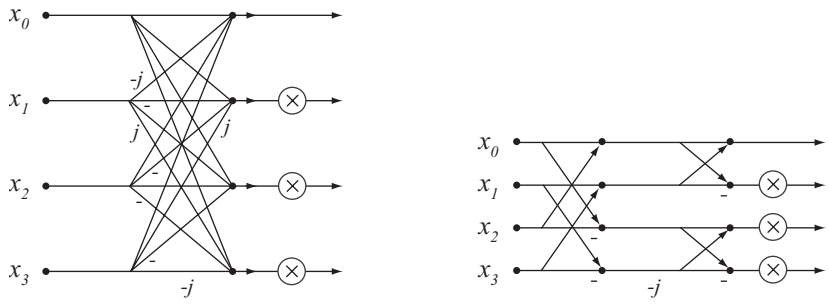
$$\begin{aligned} X(2k) &= \sum_{n=0}^{N/2-1} \Big(x_1(n) + x_2(n)\Big) W_{N/2}^{kn} \\ &= DFT_{N/2}(x_1(n) + x_2(n)) \end{aligned} \tag{4.5}$$

and

$$\begin{aligned} X(2k+1) &= \sum_{n=0}^{N/2-1} \Big[\Big(x_1(n) - x_2(n)\Big) W_N^n\Big] W_{N/2}^{kn} \\ &= DFT_{N/2}((x_1(n) - x_2(n)) W_N^n). \end{aligned} \tag{4.6}$$

Equation 4.5 calculates the sum of two sequences, while Equation 4.6 calculates the difference and then scales the result. This kind of operation, adding and subtracting the same two values, is commonly referred to as *butterfly* due to its butterfly-like shape in the flow graph, shown in Figure 4.1(a). Sometimes, scaling is also considered to be a part of the butterfly operation. The flow graph in Figure 4.1(b) represents the computations from Equation 4.5 and Equation 4.6, where each decomposition step requires $N/2$ butterfly operations.

In Figure 4.1(b), the output sequence from the FFT appears scrambled. The binary output index is *bit-reversed*, i.e. the most significant bits (MSB) have changed place with the least significant bits (LSB), e.g. 11001 becomes 10011. To unscramble the sequence, bit-reversed indexing is required.



(a) (b)

Figure 4.2: (a) Radix-4 butterfly. (b) Radix-2^2 butterfly.

4.3 Implementation aspects

In Section 4.2, the sequence is decomposed into two groups using a radix-2 butterfly, which is possible when the transform size is a power of 2. When the transform size is a power of 4, more hardware efficient decomposition algorithms exist. This section presents the radix-4 and the radix-2^2 algorithm, which both reduce the hardware requirements compared to the radix-2 algorithm. However, to calculate an FFT size not supported by radix-4, for example 2048, both radix-4 decompositions and a radix-2 decomposition will be needed since $N = 2048 = 2^1 4^5$. A short description of the FFT algorithms is presented below.

Radix-2

The radix-2 algorithm can be used when the size of the transform is $N = 2^q$, where q is an integer number. The algorithm requires q decomposition steps, each computing $N/2$ butterfly operation using a radix-2 (R-2) butterfly, shown in Figure 4.1(a).

Radix-4

When the size is $N = 4^q$, the radix-4 algorithm can be used. Radix-4 decomposition reduces the number of complex multiplications, with the penalty of increasing the number of complex additions. The complex radix-4 butterfly is found in Figure 4.2(a).

Radix-2^2

Another decomposition similar to radix-4 is the radix-2^2 algorithm, which simplifies the complex radix-4 butterfly into four radix-2 butterflies. On a flow graph level, radix-4 and radix-2^2 requires the same number of resources. The difference between the algorithms become evident when folding is applied, which is shown later. The radix-2^2 (R-2^2) butterfly is found in Figure 4.2(b).

4.3.1 Algorithm mapping

There are several ways of mapping an algorithm to hardware. Three approaches are discussed and evaluated in this section: direct-mapped hardware, a pipeline structure,

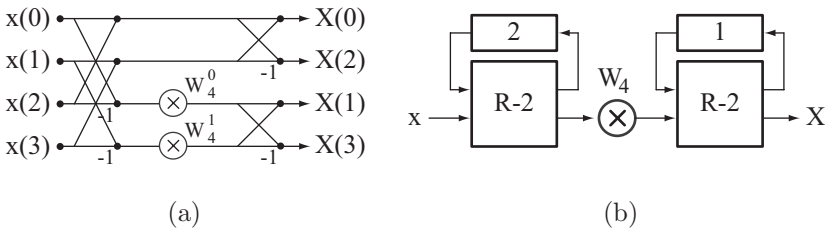


Figure 4.3: (a) Flow graph of a 4-point radix-2 FFT (b) Mapping of the 4-point FFT using two radix-2 butterfly units with delay feedback memories.

and time-multiplexing using a single butterfly unit. Direct-mapped hardware basically means that each processing unit in the flow graph is implemented using a unique arithmetic unit. Normally, using this approach in a large and complex algorithm is not desirable due to the huge amount of hardware resources required. The alternative is to fold operations onto the same block of hardware, an approach that saves resources but to the cost of increased computation time.

Figure 4.3(a) shows a 4-point radix-2 FFT. Each stage consists of two butterfly operations, hence the direct-mapped hardware implementation requires 4 butterfly units and 2 complex multiplication units, numbers that will increase with the size of the transform. Folding the algorithm vertically, as shown in Figure 4.3(b), reduces hardware complexity by reusing computational units, a structure often referred to as a *pipeline* FFT [22]. A pipeline structure of the FFT is constructed from cascaded butterfly blocks. When the input is in sequential order, each butterfly operates on sample x_n and $x_{n+N/2}$, hence a delay buffer of size $N/2$ is required in the first stage. This is referred to as a single-path delay feedback (SDF). In the second stage, the transform is $N/2$, hence the delay feedback memory is $N/4$. In total, this sums up to $N-1$ words in the delay buffers.

Folding the pipeline architecture horizontally as well reduces the hardware to a single time-multiplexed butterfly and complex multiplier. This approach saves arithmetic resources, but still requires the same amount of storage space as a pipeline architecture. The penalty is further reduced calculation speed, since all decompositions are mapped onto a single computational unit.

To summarize, which architecture to use is closely linked to the required computational speed and available hardware and memory resources. A higher computational speed normally requires more hardware resources, a trade-off that has to be decided before the actual implementation work begins.

Another important parameter associated with the architectural decisions is the wordlength. An increased wordlength improves the computational quality, measured in *signal-to-quantization-noise-ratio* (SQNR), but increases the hardware cost as well. This trade-off is known as wordlength optimization and is usually specified as a generic parameter during the design phase.

Table 4.1: Properties for different FFT architectures. Multipliers and adders are complex valued. The number of clock cycles depends on the transform length N.

Hardware architecture	Adders	Multipliers	Memory	Cycles
Direct-mapped radix-2	$N \log_2 N$	$(N/2)(\log_2(N) - 1)$	0	-
Direct-mapped radix-4	$2N \log_4 N$	$(3N/4)(\log_4(N) - 1)$	0	-
Direct-mapped radix-2^2	$2N \log_4 N$	$(3N/4)(\log_4(N) - 1)$	0	-
Pipeline radix-2	$2 \log_2 N$	$\log_2(N) - 1$	$N - 1$	$N - 1$
Pipeline radix-4	$8 \log_4 N$	$\log_4(N) - 1$	$N - 1$	$N - 1$
Pipeline radix-2^2	$4 \log_4 N$	$\log_4(N) - 1$	$N - 1$	$N - 1$
Time-multiplexed radix-2	2	1	N	$N \log_2 N$
Time-multiplexed radix-4	8	1	N	$N \log_4 N$
Time-multiplexed radix-2^2	4	1	N	$N \log_4 N$

4.4 Architecture selection

The hardware structures described in Section 4.3 are summarized in Table 4.1, which also presents requirements for implementations based on radix-2, radix-2^2 and radix-4. Based on the specification in Section 2.5, an appropriate architecture can be selected by evaluating the speed and area requirements. The maximum transform size is $N = 2048$, and the total number of clock cycles for a single one-dimensional transform must be in the range of $\text{FFT}_{\text{cc}} \approx 2400$ clock cycles.

4.4.1 Evaluation

With $N = 2048$, Table 4.1 can be used to select an appropriate architecture. The direct-mapped hardware architecture will generate the fastest implementation, but also the most area consuming. With the current selection of N, a radix-2 architecture requires 22528 adders and 10240 multipliers. This is not feasible to implement on a chip today. A direct-mapped hardware architecture is only realistic when N is small.

The time-multiplexed architecture has a fixed number of adders and multipliers regardless of the FFT size. However, the number of clock cycles does not comply with the specification, requiring over 20000 clock cycles for a single transform.

Hardware requirements for the pipeline radix-2 architecture are 22 adders and 10 multipliers, which makes pipeline architectures suitable for hardware implementation. The computation time is also in the range of the specified value, namely 2047 clock cycles. The pipeline architecture is therefore a suitable trade-off between computational speed and area.

Radix-2^2 and radix-4 requires fewer complex multipliers than radix-2. Furthermore, radix-2^2 requires fewer adders than radix-4. Hence, the selected architecture for this project is based on the radix-2^2 single path delay feedback (R2^2SDF) algorithm [22]. Since the transform size is 2048, an extra radix-2 unit is added to support transform length of size $N = 2^q$. The total pipeline is constructed from one radix-2 block and five radix-2^2 blocks, shown in Figure 4.4. The hardware requirements are 22 adders and 5 multipliers.

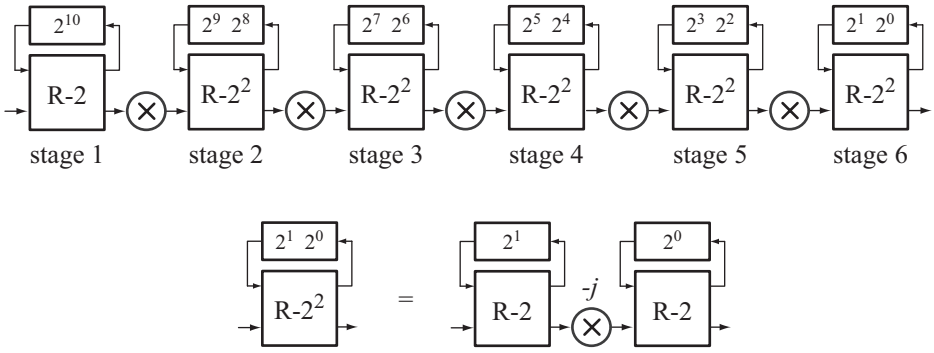


Figure 4.4: A 2048 point pipeline FFT constructed from one radix-2 butterfly and five radix-2^2 butterflies. A radix-2^2 butterfly is constructed from two radix-2 butterflies separated by a trivial multiplication with $-j$.

4.5 Scaling alternatives

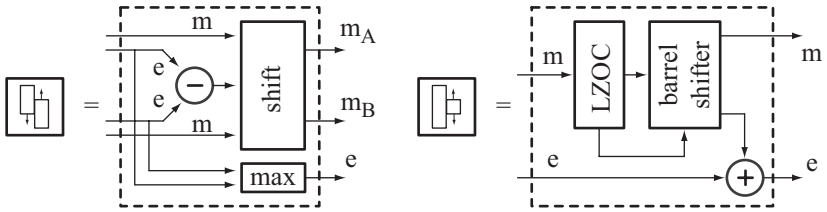
For application-specific hardware implementations, fixed-point is the most commonly used format due to the simple implementation of arithmetic units. When designing a datapath, the wordlength of arithmetic units are adjusted to support the required dynamic range and precision. Adjusting the wordlength can be done manually or by using a fixed-point simulation tool [23]. This is often performed by estimating the range of floating-point values, and then converting the program into a fixed-point model. In a fixed-point model, the result from an addition and subtraction requires an increased wordlength to avoid overflow, which changes the dynamic range. The result from a multiplication is usually rounded or truncated to avoid a significantly increased wordlength, hence causing a quantization error.

Independent of the dynamic range and the distribution of the input signal, the quantization energy P_q is nearly constant for a fixed-point value. When the energy in the input signal P_x varies, this affects the SQNR defined as

$$\mathrm{SQNR}_{\mathrm{dB}} = 10 \cdot \log_{10} \frac{P_x}{P_q}. \tag{4.7}$$

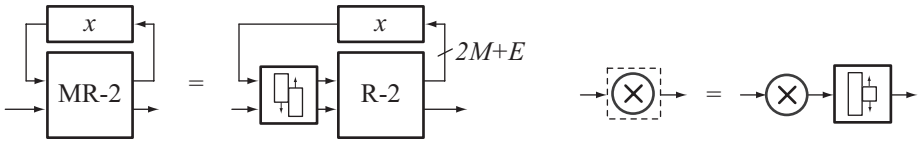
Therefore, precision in the calculations depends on properties of the input signal, caused by uniform resolution over the total dynamic range. Fixed-point arithmetic has low complexity, but usually requires an increased wordlength due to the trade-off between dynamic range and precision.

Floating-point arithmetic increases the dynamic range by expressing numbers with a mantissa m and an exponent e, represented with M and E bits, respectively. By changing the quantization steps, the energy in the error signal will follow the energy in the input signal, and the SQNR will remain relatively constant over a large dynamic range. Compared to fixed-point numbers, some bits are reserved to represent the exponent, lowering the precision but increasing the dynamic range. Arithmetic units implemented using floating-point representation increases in complexity since the format requires mantissa alignment prior to computation and that the result is normalized [24]. In the



(a) (b)

Figure 4.5: (a) Unit for aligning input operands for addition/subtraction. (b) Normalization unit selecting the most significant bits after a multiplication. LZOC is a leading zeros/ones counter to determine the shift amount.



(a) (b)

Figure 4.6: Building blocks for a hybrid floating-point implementation. (a) Modified butterfly containing an align unit on the input. (b) Modified complex multiplier containing a normalization unit on the output.

following subsections, different scaling alternatives is presented followed by a comparison in terms of hardware requirements and precision in Section 4.6.

4.5.1 Hybrid floating-point

A simplified scheme for floating-point representation of complex numbers is to use a single exponent for the real and imaginary part. Besides reduced complexity in the arithmetic units, especially the complex multiplication, the total wordlength for a complex number is reduced from $2(M+E)$ to $2M+E$ bits. This representation is referred to as a hybrid floating-point format.

Supporting hybrid floating-point includes pre- and post-processing in the arithmetic building blocks. Adding and subtracting values requires an alignment of input operands, a low complexity operation performed by right-shifting the smallest value by the difference between the exponents. The multiplication with a twiddle factor, represented with T bits, is not affected by misaligned operators but produces an $M+T$ bit full precision mantissa which requires normalization. Normalization is performed by finding the M most significant bits from the full precision result using a leading zeros/ones counter (LZOC). The alignment and normalization units are found in Figure 4.5.

In the general case, multiplication also requires adding the exponent values. Since the FFT twiddle factors are fixed-point values, this operation can be discarded in the current design. The building blocks for hybrid floating-point are shown in Figure 4.6, where MR-2 stands for modified radix-2 butterfly.

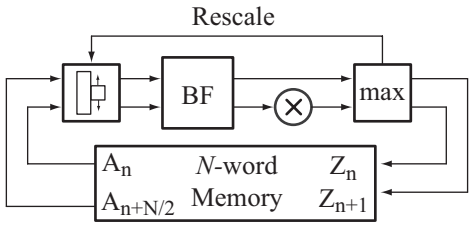


Figure 4.7: A simplified view of an iterative FFT processor using block floating-point. The peak value is monitored by the *max* unit, controlling the normalizing unit connected to the input of the butterfly.

4.5.2 Block floating-point

Block floating-point (BFP) combines the advantages of simple fixed-point arithmetic with floating-point dynamic range. Instead of having an exponent for each value, one single exponent is assigned to a group of values. There are two major reasons for using block floating-point, reduced memory requirements and a lower arithmetic complexity. However, output signal quality depends on the block size and characteristics of the input signal [25]. The dynamic range and memory requirements for representation in fixed- and floating-point are straightforward. These properties are more complex for block floating-point and depend on the block size. A large block size decreases the precision of low amplitude signals for a block set containing a high peak value, but consumes less memory.

Representing data in block floating-point only requires one single exponent, valid for a block of values. The exponent is based on the largest value in the block, hence all values in a block must be analyzed before a common exponent can be determined. For a time-multiplexed implementation of the FFT, using a single butterfly unit, this will not cause any problems. Figure 4.7 shows an architecture containing a single memory and butterfly unit. The data is processed one radix-r butterfly stage at a time, requiring $\log_r N$ iterations for an N-point FFT sequence. After each iteration, all information are transferred from the main memory, processed, and then written back to the memory. The output from the complex multiplication is monitored to find a common exponent, and scales the data accordingly at the next iteration. The drawback is that the wordlength of the memory must hold the values in extended precision from the complex multiplication unit, adding a number of guard bits that will increase the memory requirements.

4.5.3 Convergent block floating-point

When applying block floating-point to an architecture with cascaded butterfly units, scaling becomes more complicated. To cope with this situation, an algorithm known as convergent block floating-point (CBFP) is proposed in [26]. By placing buffers between intermediate stages, data can be rescaled using block floating-point before it is propagated to the next FFT stage, as shown in Figure 4.8. Since the pipeline FFT divides the sequence into smaller butterfly operations, the block size will decrease as

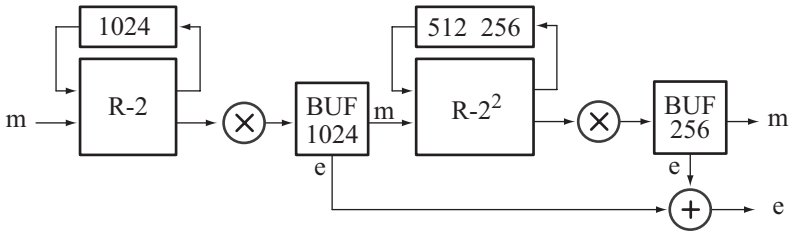


Figure 4.8: An example of convergent block floating-point. The buffer after each complex multiplier selects a common exponent for a group of values, allowing fixed-point butterfly units. The first buffer can be removed to save storage space, but will have a negative impact on the quality.

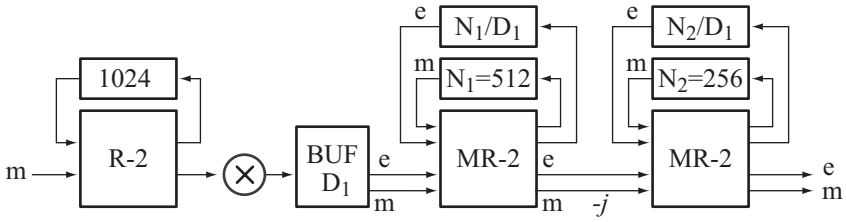


Figure 4.9: The buffer after each complex multiplier selects a common exponent for a small group of values. The optimal value is evaluated from Equation 4.8.

it propagates through the pipeline. Consider the architecture in Figure 4.4. After the first butterfly stage, two sequences of 1024 samples are propagated. Each radix-r stage produces r groups using separate exponents. As the data propagates, the groups will be further divided into smaller groups, until each value has its own exponent. Hence the name convergent block floating-point. The buffering of data between each stage requires a large amount of memory. The intermediate buffer is placed after the complex multiplication and normalization is performed at the output. In practical applications, the first intermediate buffer is often omitted to save storage space, but this leads to a reduced SQNR.

4.5.4 Co-optimization

As previously mentioned, convergent block floating-point places intermediate buffers between each radix-r block to rescale a group of values. Hybrid floating-point uses one exponent for each complex value, and the intermediate buffers become obsolete. This section describes how these architectures can be co-optimized, using hybrid floating-point extended with small intermediate buffers to reduce storage space for exponents.

The hybrid floating-point solution requires $N_x(2M + E)$ bits in the delay feedback, where N_x is the length of the FIFO. Using a block size D requires an intermediate buffer with D full precision words. At the same time, the number of exponents decreases with a factor of D to NE/D. For a radix-2 stage, the memory requirements are

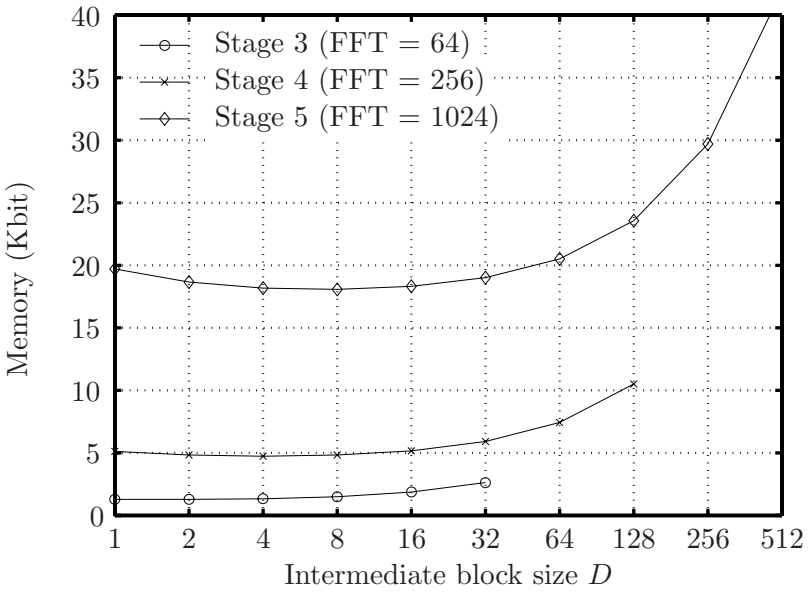


Figure 4.10: Memory requirements as a function of D for radix-2^2 blocks with and input FFT length from 64 to 1024. The memory for each stage includes the intermediate buffer connected to its input. $D = 1$ shows the direct hybrid floating-point approach. $M = 10$, $E = 4$.

$$\underbrace{N(2M + E/D)}_{\text{FIFO}} + \underbrace{2D(M + T)}_{\text{BUF}}. \tag{4.8}$$

For a radix-2^2 stage, the delay feedback for the second butterfly should be added to the equation. The resulting architecture is presented in Figure 4.9, with small intermediate buffers and a separate exponent feedback memory. The MR-2 building block is the modified butterfly found in Figure 4.6(a). For each radix-r stage, an optimal value of D can be found. Figure 4.10 shows how memory requirements depend on the intermediate buffer size. Each graph represents the memory requirements to implement a radix-2^2 stage with an intermediate buffer connected to its input.

4.6 Simulations

A simulation tool to evaluate different FFT architectures has been designed. The simulation environment can extract precision, dynamic range, memory requirements, estimated synthesis results and chip size based on architectural descriptions. The user can specify the number of bits for representing mantissa, twiddle factors, and exponents. Furthermore, the user selects FFT size, rounding type and finally stimuli type to produce random values, sine waves, peaks, or signal constellations used in for example OFDM.

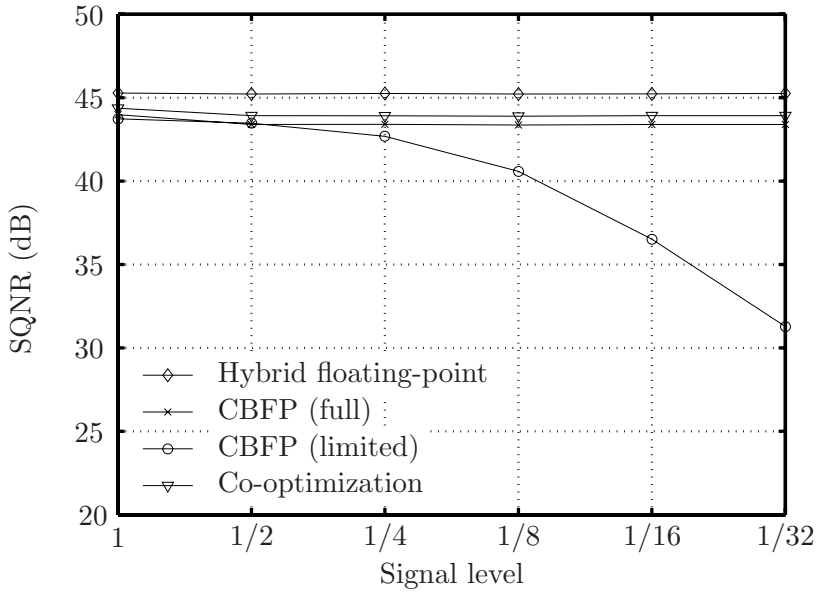


Figure 4.11: Decreasing the energy in a random value input signal affects only the architecture when scaling is not applied in the initial stage. Signal level=1 means utilizing the full dynamic range.

Four architectures have been evaluated. The first is the hybrid floating-point with bidirectional support. Next, two versions of convergent block floating-point, one with scaling starting after the first stage (*CBFP full*) and one with scaling starting after the second stage (*CBFP limited*). Avoiding the first intermediate buffer saves a large amount of storage space, but will have a negative affect on quality. The latter architecture is a co-optimization between hybrid floating-point and CBFP, using smaller intermediate buffers. Table 4.2 shows a comparison in required memory and SQNR on a random value signal utilizing the full dynamic range. The intermediate buffers used in CBFP consume a large amount of memory, a problem that can be avoided by using the co-optimized architecture. Figure 4.11 shows the result of changing energy in the input signal. In this case, the variations only affect the CBFP implementation with scaling applied later in the pipeline. Figure 4.12 shows the result when applying signals with a large crest factor, i.e. the ratio between peak and mean value of the input. For a signal

Table 4.2: Memory elements and SQNR for different architectures, based on a flexible 2048 point radix-2^2 implementation, with 10-bit input.

Scaling method	FIFO	BUF	Total	SQNR	Bidir.
Hybrid FP	49040	0	49040	45.3 dB	Yes
CBFP (limited)	45716	14690	60406	43.7 dB	No
CBFP (full)	45716	60016	105732	43.9 dB	No
Co-optimization	45779	1584	47363	44.3 dB	No

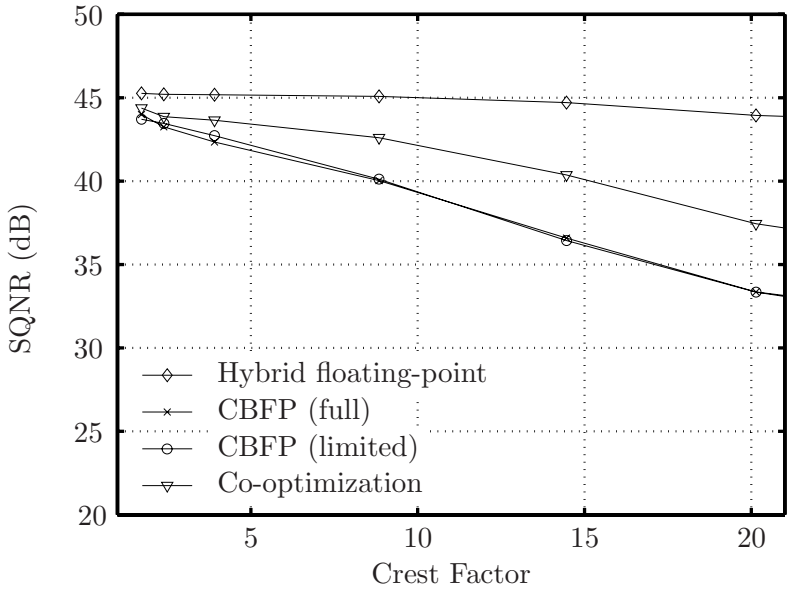


Figure 4.12: Decreasing the energy in a random value input signal with peak values utilizing the full dynamic range. This affects all block scaling architectures, and the SQNR depends on the block size. The co-optimized architecture performs better than convergent block floating-point, since it has a smaller block size through the pipeline.

$s(n)$ with N samples, the crest factor (CF) is defined as

$$\mathrm{CF} = \frac{\max|s(n)|}{\sqrt{\frac{1}{N}\sum_n |s(n)|^2}}. \tag{4.9}$$

In this case, both CBFP implementations are affected due to the large block size in the beginning of the pipeline. Hybrid floating-point is not affected by signal statistics, since every value is scaled individually. The SQNR for the co-optimized solution is located between hybrid floating-point and CBFP since the block size is in the range of 4 to 16 values.

For the current application, it is desirable to use an architecture capable of producing a high quality result for signals with high crest factors. Images captured in the digital holographic setup are pre-processed before the inverse FFT is applied. The crest factor after pre-processing is large, which requires an accurate FFT core to produce a high quality result.

4.7 Implementation

A pipeline FFT core using hybrid floating-point and based on the radix-2^2 decimation-in-frequency algorithm [22] has been implemented, fabricated, and verified. The pipelined implementation requires one radix-2 unit, 5 radix-2^2 units, and 5 complex multiplication

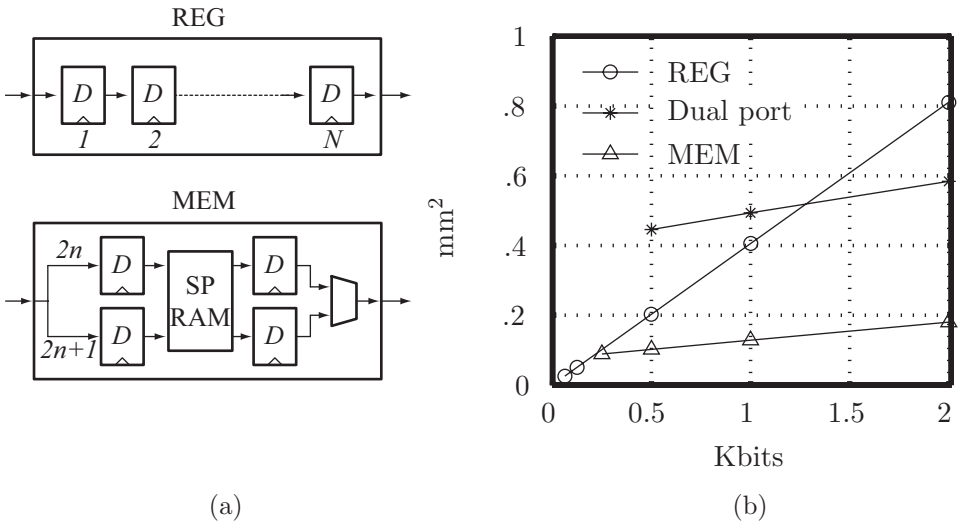


Figure 4.13: (a) Registers are used for short delays and single port memories for long delays. (b) Simulation of area requirements for different approaches.

and normalization blocks. The initial radix-2 stage is required since the size of the FFT is not a power of 4. This section presents internal building blocks and the fabricated ASIC.

4.7.1 Butterfly and complex multiplier

There are two radix-2 butterfly stages in each radix-2^2 stage that calculate the sum and the difference between the input values and the delayed output sequence from the single-path delay feedback. The butterfly has two different modes, one for calculating and one for transferring data to and from the delay feedback. When scale factors are used, it must be possible to align the inputs if they do not share the same exponent. An equalizer unit performs the alignment of input values to the butterfly stage. When the butterfly is filling or draining the delay feedback, the equalizer unit only propagates the current input values. The equalizer compares the exponents of the two inputs to detect if the values are aligned or not. If there is a difference, the smallest input value is right shifted with the same number of bits as the difference between the two exponents. The aligned values are propagated to the butterfly unit.

The output from the complex multiplier is normalized and sent to the next FFT stage. At the same time as the value is shifted by the normalizing unit, the exponent is incremented accordingly.

4.7.2 Delay feedback considerations

Delay feedback units provide a temporary storage space for the input sequence to each butterfly, i.e. the FIFOs in Figure 4.4. The delay feedback can be implemented either as registers or as memories (RAMs), as shown in Figure 4.13(a). For shorter sequences, serially connected flip-flops are used as delay elements. As the delay increases, this

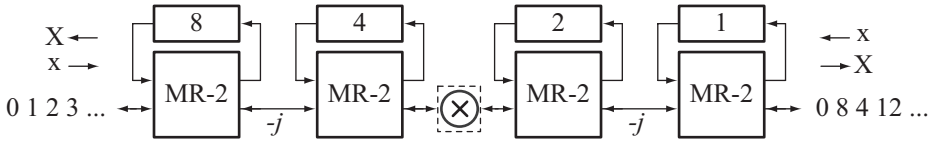


Figure 4.14: The bidirectional pipeline of a 16-point FFT. The input/output to the left is in normal bit order. The input/output from the right is in bit-reversed order.

approach is no longer area and power efficient. One solution is to use SRAM memories, single or dual port, and address pointers to keep track of read and write locations. To keep computational units supplied with data, one read and one write operation has to be performed each clock cycle. A dual port memory approach allows simultaneous read and write operations with simple control logic. However, dual port memories are both larger and consume more power than single port memories. Instead it is possible to use two single port memories, alternating between read and write each clock cycle. This approach can be further simplified by using only one single port memory with double wordlength to hold two consecutive values in a single location, alternating between reading two values in one cycle and writing two values in the next cycle. This scheme requires temporary storage to synchronize the dataflow. In this design, the latter single-port memory approach is used for delay feedback exceeding the length of 8 values. Simulation has shown this to be a good trade-off to minimize area [27]. A comparison between different approaches is presented in Figure 4.13(b). In the current 0.35 μm CMOS process, the dividing line between flip-flops and memories is located at approximately 250 bits. Changing to a different process implies a new evaluation to find the optimal solution.

4.7.3 Bidirectional pipeline

If the wordlength is constant in the pipeline, the dataflow can be reversed, as shown in Figure 4.14. The bidirectional FFT has several applications. Replacing the FFT/IFFT used in for example an OFDM transceiver with a bidirectional pipeline can minimize the buffering required for inserting and removing the cyclic suffix, which has been proposed in [28]. Today, wordlength optimized one-directional FFT are commonly used, increasing the wordlength through the pipeline to maintain accuracy. Implementations based on CBFP are also proposed in [29], but both solutions only work in one direction. Another application for a bidirectional pipeline is when evaluating a one- or two-dimensional convolution using the FFT as

$$x * h = \mathcal{F}^{-1}(\mathcal{F}(x) \cdot \mathcal{F}(h)). \tag{4.10}$$

If the forward transform generates data in bit-reversed order, input to the reverse transform should also be bit-reversed. In the first step, the dataflow is from left to right. In the second step, the dataflow is from right to left. Both the input and the output from the convolution is in normal bit order, hence no reorder buffers are required. The FPGA prototype is based on a bidirectional pipeline.

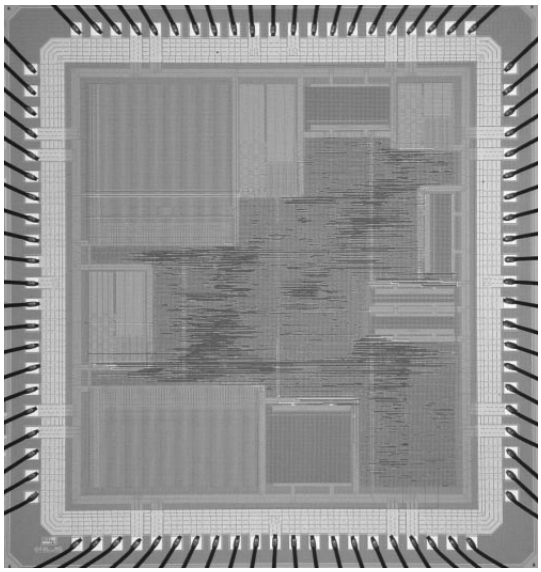


Figure 4.15: Chip photo of the 2048 complex point FFT core fabricated in a $0.35\,\mu$m 5ML CMOS process. The core size is $2632 \times 2881\,\mu\text{m}^2$ connected to 58 I/O pads and 26 power pads.

4.8 ASIC Prototyping

A 2048-point FFT supporting hybrid floating-point has been fabricated in a $0.35\,\mu$m five metal layer CMOS process from AMI Semiconductor, and a chip photo is shown in Figure 4.15. The core size is $2632 \times 2881\,\mu\text{m}^2$ connected to 58 I/O pads and 26 power pads. The FFT chip contains 11 delay feedback buffers, one for each butterfly unit. Seven on-chip RAMs are used as delay buffers, with approximately 49 Kbits of storage capacity. The four smallest buffers are implemented using flip-flops, which have been evaluated to be a good solution in terms of chip area. Twiddle factors for the complex multipliers are stored in three ROMs, with a size of approximately 47 Kbits in total. RAMs and the ROMs can be seen along the sides of the chip in Figure 4.15.

A pattern generator and a logic analyzer were used to confirm functionality and measure the power characteristics of the prototypes. By converting the test vectors from the preceding netlist simulations, the exact same test vectors are used for prototype verification. The comparison between the output results from the prototype and the netlist simulation showed that the circuits were functional up to 50 MHz using a supply voltage of 1.8 V. The power consumption of the core was measured to 234 mW at 1.8 V and 50 MHz.

Chapter 5

Streaming hardware accelerator

This chapter presents a generic and flexible hardware accelerator for general signal and image processing, which is targeted for digital holographic imaging. The developed accelerator, referred to as XSTREAM, contains a datapath with several modules for reordering, scaling, rotating, and transforming data. The FFT described in Chapter 4 is a central building block of the accelerator.

5.1 Requirements

The reconstruction algorithms described in Chapter 2 define the requirements for the accelerator. First, Equation 2.1 states that the recorded images are to be subtracted to obtain the interference pattern. Then, looking at Equation 2.4, each element in the interference pattern is rotated with a phase factor before Fourier transformation.

Figure 5.1 shows the actual dataflow and how the algorithm is executed. Figure 5.1(a) combines and subtracts the images, rotates each pixel vector with a phase factor, and calculates one-dimensional FFTs over the rows. Figure 5.1(b) takes the result from the first operation and calculates one-dimensional FFTs over the columns. The largest value is stored in a register before writing the sequence back to memory.

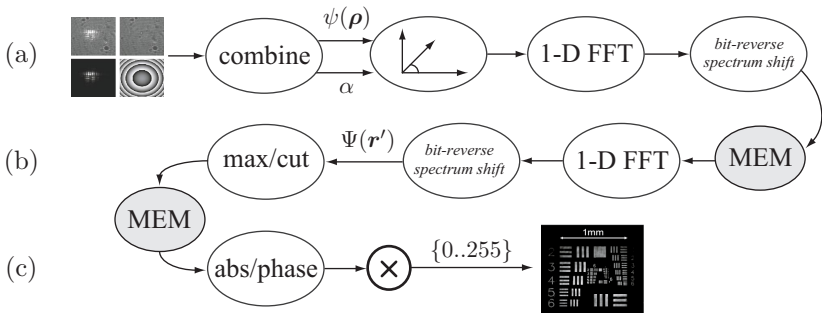


Figure 5.1: Processing dataflow. Grey boxes indicate memory transfers and white boxes indicate computation. (a) Combine, rotate, and one-dimensional FFT over rows. (b) One-dimensional FFT over columns and store maximum value. (c) Calculate magnitude or phase and scale result into pixels.

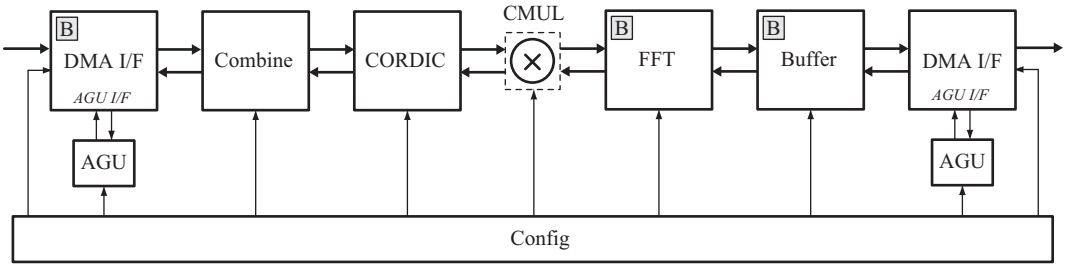


Figure 5.2: Functional units in the XSTREAM accelerator. Grey boxes indicate units that contain local buffers. Processing units communicate using a handshake protocol.

In the final step, shown in Figure 5.1(c), the vector magnitude or phase is calculated. The result is also scaled to a pixel value, using the largest value from the previous iteration.

5.2 Architecture

XSTREAM is constructed from a pipeline with several hardware units, mapping the dataflow in Figure 5.1 to hardware. The name XSTREAM denotes that the accelerator operates on data streams, and could potentially be reconfigured to support multiple input and output streams. Focus is on maximizing computational efficiency with parallel execution, and optimizing global bandwidth using burst transfers. Processing units communicate using a handshake protocol, where each produced value is acknowledged by the receiver. Without flow control, buffers could overrun with corrupted data as a result. The pipeline is configured using an external controller, e.g. a microprocessor. Each unit can be individually configured, and a global register allows units to be enabled and disabled.

Figure 5.2 shows the XSTREAM pipeline, and the processing direction is from left to right. The first unit combines recorded images and calculates the interference pattern. The combine unit is constructed from a reorder unit to extract pixels from the same position in each image, and a subtract unit. Each individual pixel in the interference pattern is actually complex-valued, with the imaginary part set to zero. This two-dimensional pixel vector is then rotated with a phase factor. The implementation is based on a *coordinate rotation digital computer* (CORDIC) operating in rotation mode. The CORDIC algorithm and implementation is described in Section 5.2.2. The next operation is the Fourier transformation, presented in Chapter 4. A two-dimensional FFT can be evaluated using a one-dimensional FFT, separately applied over rows and columns. Considering the size of the two-dimensional FFT, data must be transferred to the main memory between row and column processing. A buffer is required to unscramble the FFT output since it is produced in bit-reversed order, and a spectrum shift operation is also required to center the zero-frequency component. The CORDIC unit is finally reused to generate magnitude or phase images from the complex-valued result produced by the FFT. The result is scaled using the complex multiplier in the

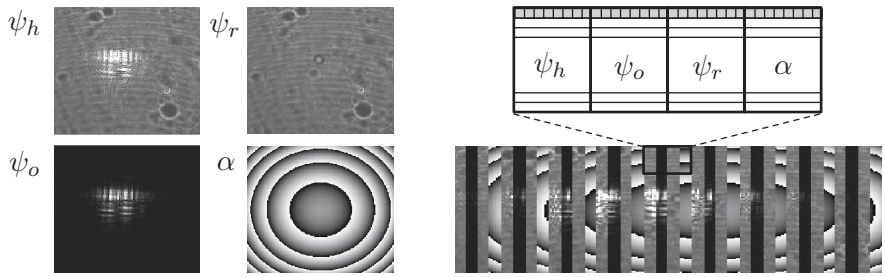


Figure 5.3: (a) The three captured images and the rotation factor α. (b) The interleaved images stored in memory and the interleaved memory map with M pixels in each row, where the buffer can hold $4M$ pixels. Grey indicates the amount of data copied to the input buffer in a single transfer. In this example the interleaving is 4 groups with 8 pixels in each group.

pipeline, from floating-point format back to fixed-point numbers that represent an 8-bit greyscale image. The following sections present the computational units in detail.

5.2.1 Image combine

The first operation is to combine the captured images and calculate the interference pattern as

$$\psi(x, y) = \psi_h(x, y) - \psi_o(x, y) - \psi_r(x, y). \tag{5.1}$$

First consider the images to be stored in separate memory regions, as shown in Figure 5.3(a). The read operation would then either be *single reads* accessing position (x, y) in each image, or *several burst transfers* from separate memory location. In the former case, simply accessing the images pixel-by-pixel from a burst oriented memory is inefficient since it requires three single transfers for each pixel. The latter approaches require several burst transfers, reading from different parts of the memory. Burst transfers are fast but the scheme is more complex, since reading and reordering information will become a complicated procedure of transferring data from three separate memory areas simultaneously. Accessing the data in one single burst would be desirable, and thus a memory reorder scheme is proposed in the following section.

Memory interleaving

One approach to combine images is to reorder the physical placement in memory. Instead of storing images in separate memory regions, they can be *interleaved*, as shown in Figure 5.3(b). Here, data is fetched in a *single burst transfer* and internally reordered. The transfer fills an input buffer, which reorders the information using a modified address generator to extract image information pixel-by-pixel. Using this scheme, both storing captured images and reading images from memory can be performed with single burst transfers.

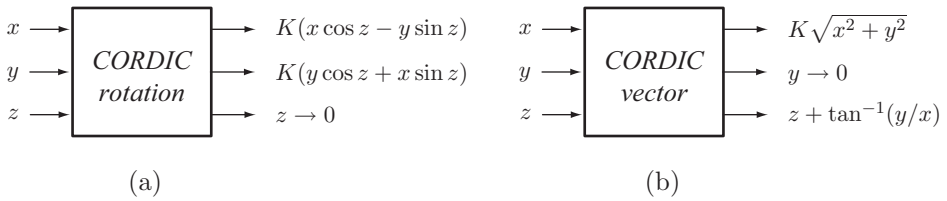


Figure 5.4: (a) CORDIC in rotation mode, $z \rightarrow 0$. (b) CORDIC in vector mode, $y \rightarrow 0$. Other functions are evaluated by changing from circular rotations to linear or hyperbolic rotations.

In addition to combining the three images, a phase factor is required for each pixel. The phase factor is propagated to the CORDIC unit to rotate the interference pattern. Assuming that the images are interleaved with a partial size of M pixels, the buffer must be able to hold $4M$ pixels. A transfer to the internal reorder buffer now includes information from each image, and the buffer output is reordered to produce a stream of pixels from the same location. Simulation results for the proposed method are presented in Chapter 6.

5.2.2 CORDIC

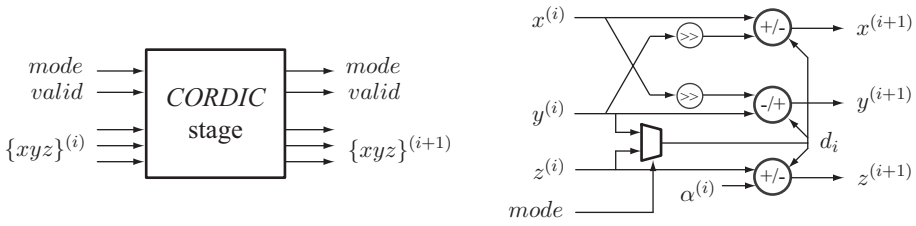
The second operation is to rotate each pixel in the interference pattern with a phase factor. The implementation is based on a CORDIC, which is a convergence method for evaluating trigonometric functions using a fixed number of iterations [30]. Each iteration step is assigned a pre-calculated angle $\alpha^{(i)}$, associated with the iteration number. The CORDIC algorithm does not perform real rotations, i.e. preserving the magnitude of the vector being rotated. Instead it uses pseudo-rotations, which reduces the complex rotation equations from

$$\begin{aligned} x^{(i+1)} &= x^{(i)}\cos\alpha^{(i)} - y^{(i)}\sin\alpha^{(i)} = \frac{x^{(i)} - y^{(i)}\tan\alpha^{(i)}}{(1+\tan^2\alpha^{(i)})^{1/2}} \\ y^{(i+1)} &= y^{(i)}\cos\alpha^{(i)} + x^{(i)}\sin\alpha^{(i)} = \frac{y^{(i)} + x^{(i)}\tan\alpha^{(i)}}{(1+\tan^2\alpha^{(i)})^{1/2}} \\ z^{(i+1)} &= z^{(i)} - \alpha^{(i)} \end{aligned}$$

to the more simple equations

$$\begin{aligned} x^{(i+1)} &= x^{(i)} - y^{(i)}\tan\alpha^{(i)} \\ y^{(i+1)} &= y^{(i)} + x^{(i)}\tan\alpha^{(i)} \\ z^{(i+1)} &= z^{(i)} - \alpha^{(i)} \end{aligned}$$

by removing the factor $1/(1+\tan^2\alpha^{(i)})^{1/2}$. Hence, the final result is scaled with a known expansion factor K, which is the product of all the expansion factors for every iteration step as



(a) (b)

Figure 5.5: (a) A basic CORDIC building block. (b) The internal hardware structure. The constant and shift factor is unique for each block.

$$K = \prod_i (1 + \tan^2 \alpha^{(i)})^{1/2}.$$

As long as the scale factor is known, it can always be compensated for later on. To reduce complexity even more, $\alpha^{(i)}$ is chosen such that the multiplication with $\tan \alpha^{(i)}$ becomes a simple operation. Setting $\tan \alpha^{(i)} = d_i 2^{-i}$, where $d_i \in \{-1, 1\}$ results in simple bit-shift and add/subtract operations. $\alpha^{(i)}$ can be precomputed and stored in a lookup table (LUT). The limitation is that it is no longer possible to rotate with any angle. The rotation is instead the sum of a set of angles, where the sign depends on d_i. Note that CORDIC is not limited to two-dimensional calculations. Several units can be connected together to evaluate even three-dimensional rotations [31].

CORDIC modes

CORDIC can operate in two different modes, and the mode determines the function to be evaluated, as shown in Figure 5.4. In *rotation* mode, d_i is chosen such that $z \to 0$, i.e. $d_i = \text{sign}(z^{(i)})$. In rotation mode, CORDIC can evaluate for example $x \sin(z)$ and $x \cos(z)$, which is actually a rotation of x with the angle z. The complex multipliers in the FFT actually perform rotations and can therefore also be evaluated with a CORDIC unit [32]. In *vector* mode, d_i is chosen such that $y \to 0$, by setting $d_i = -\text{sign}(x^{(i)} y^{(i)})$. In vector mode, the vector magnitude and phase can be evaluated from a complex value, which is used to convert the complex result from the FFT into visible images.

Implementation

Instead of performing iterations over time, the design can be constructed by connecting a number of identical stages in sequence. Figure 5.5(a) shows a CORDIC stage, and Figure 5.5(b) shows the actual implementation, which requires three adders and control logic for mode selection and d_i condition. A pipeline implementation of the CORDIC algorithm is chosen since it can produce one value each clock cycle, which is matched to the throughput of the one-dimensional FFT core.

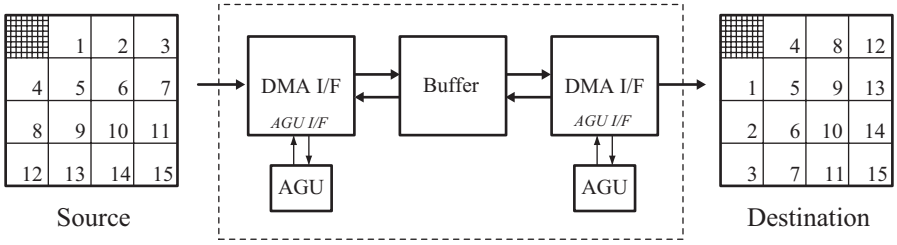


Figure 5.6: The transpose logic uses the DMA controllers, the AGU units and the internal buffer to transpose large matrices. In the figure, a 32×32 matrix is transposed by individually transposing and relocating 4×4 macro blocks. Each macro block contains an 8×8 matrix.

5.2.3 Two-dimensional FFT

A two-dimensional FFT can be calculated by first performing one-dimensional transforms over the rows, and the applying the one-dimensional transform over the columns of the result. However, the memory access pattern when transforming columns would cause a serious performance loss since new rows are constantly accessed, which will prevent burst transfers. An equivalent procedure to the row/column approach is to transpose the information between the computations. This means that the FFT is actually applied two times over rows in the memory with an intermediate transpose operation. If the transpose operation combined with FFTs over rows is performed faster than FFTs over columns, the overall calculation time is reduced. To evaluate this approach, a fast transpose unit is required. Simulation results and a comparison between the two approaches are presented in Chapter 6.

Transpose unit

The transpose operation is critical for system performance. Considering the transformation size, data cannot be stored on-chip and has to be transferred to the off-chip main memory. The access pattern for a transpose operation is normally reading rows and writing columns, or vice versa. Reading or writing a *row* will not cause performance problems, but the access pattern of reading or writing *columns* will significantly degrade performance.

Performance can be improved by changing the memory access pattern [33]. The transpose operation of a matrix can be broken down into a number of smaller transpose operations combined with block relocation as

$$H = \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix} \qquad H^T = \begin{pmatrix} h_{11}^T & h_{21}^T \\ h_{12}^T & h_{22}^T \end{pmatrix}. \tag{5.2}$$

The matrix can be further divided using a divide-and-conquer approach, all the way down to single elements. Breaking it into single elements is not desirable, since all accesses will become single read and write operations. However, if the matrix is divided into smaller blocks that fit into a cache memory, burst operations can be applied for

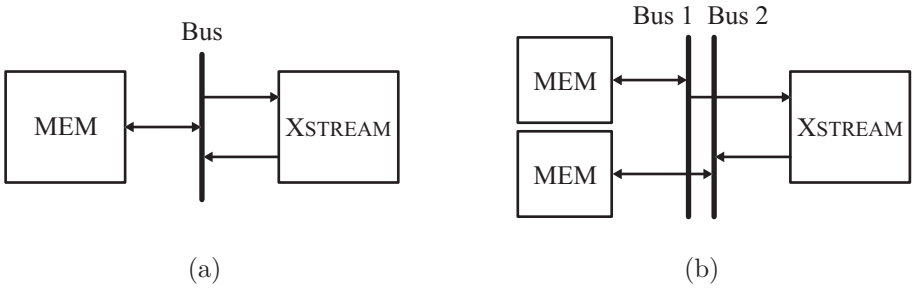


Figure 5.7: (a) XSTREAM connected to a single bus. (b) XSTREAM connected to separate busses. Throughput is increased without affecting behaviour.

both reading and writing. Memory burst length depends on the size of the transpose buffer. The transpose operation is illustrated in Figure 5.6, and the simulation results are presented in Chapter 6.

5.3 External interface

XSTREAM communicates using a standard bus interface to transfer data to and from the pipeline. Since the accelerator is based on a pipeline of computational units, there is one input interface and one output interface. Figure 5.7 shows two possible configurations for the system. The input and output can either be connected to the same bus, which would limit the performance, or to separate busses and memory banks. However, the problem with multiple memory interfaces is the large amount of input and output signals required. Each 32-bit SDRAM bank requires approximately 50 connections for data, address, and control signals. Designing a system with several memory interfaces is pad-consuming, hence the prototype presented in Chapter 7 uses a single interface due to board limitations.

Data is transferred using DMA modules, as shown in Figure 5.8. After a DMA transfer is initiated, the transfer is carried out by directly accessing the external memory. The DMA modules are connected to one or several on-chip busses, and arbitration is used to select the bus master. The DMA interface is constructed from three blocks, the bus interface, a buffer module and a configuration interface. One side connects to the external bus while the other side connect to the internal dataflow protocol.

5.3.1 Bus interface

The XSTREAM bus interface is compatible with the advanced high-speed bus (AHB) protocol which is a part of the AMBA specification [34]. Configuration data is transferred over the advanced peripheral bus (APB). The bus interface acts as a master on the bus, reading and writing data using burst accesses of any length. This is an important feature, which allows fast access to burst oriented memories.

The interface can not only access a linear address space, but also two-dimensional memory arrays. By specifying a base address (*addr*), the transfer size (*hsize, vsize*), and the space between individual rows (*space*), the bus interface can access a two-

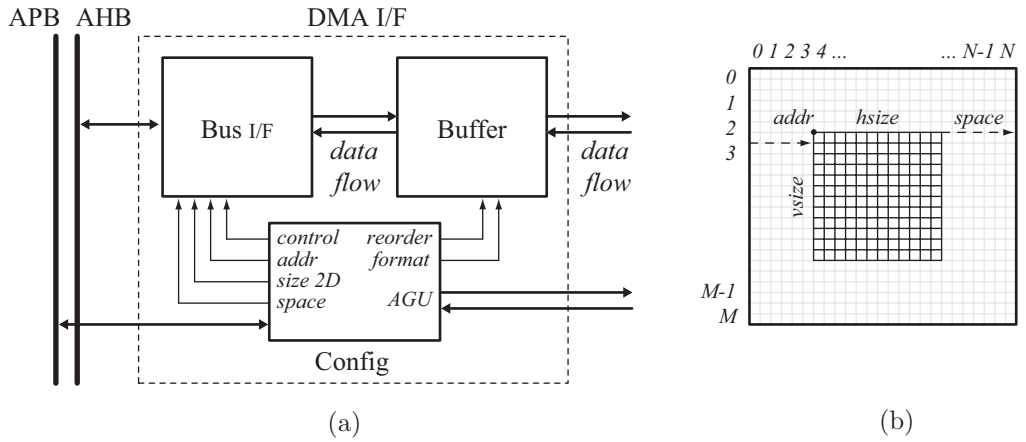


Figure 5.8: (a) The DMA interface connects the internal dataflow protocol to the external bus using a buffer and an AMBA bus interface. (b) The DMA interface can access a two-dimensional matrix inside a two-dimensional memory array. *space* is the distance to the next row.

dimensional matrix of any size inside a two-dimensional address space of any size. The *control* register contains signals to *start* and *stop* a transfer, and to specify the burst size.

5.3.2 Buffering

Bursting data requires internal buffering to guarantee that neither overflow nor underrun will occur on the dataflow side during the transfer. Therefore, the DMA interface contains a data buffer, shown in Figure 5.9, separating the bus from the dataflow modules. The size of the buffer determines the maximum burst size.

The buffer contains a format transformation unit that allows splitting of words into sub-word transfers on the read side and combining sub-words into words on the write side. This is useful when processing pixels from the sensor, for example, calculating the vector magnitude of the resulting image and then rescaling the value into a pixel (8 bits). The pixels are individually processed, but combined into words (groups of 4 pixels) in the buffer before transferred to the memory.

Another useful feature is the possibility of reordering information in the buffer. The buffer can be divided into 2 or 4 sections, and data is transferred by sending the first word in each section, then the second word from each section and so on. In this addressing mode, information from different data sets can be processed, used for example in the combine unit and for calculating convolutions.

Note that in a system containing a single bus, the burst length should not exceed the buffer size. Exceeding the buffer size could potentially create a deadlock situation, for example when a read access is stalled waiting for data to be written back to memory before accepting new data. In this situation, a bus architecture supporting *split transfers* has to be used, changing bus master even if the current transfer has not completely

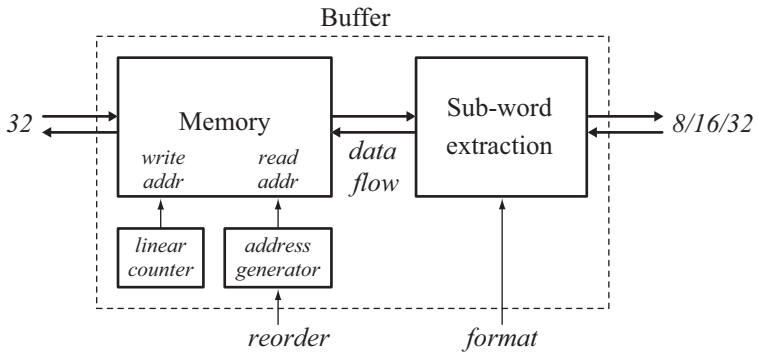


Figure 5.9: The DMA buffer can reorder data in several ways and extract (split and combine) subword for 8, 16 and 32 bit processing. The buffer can operate in both directions, depending on if it is used as a read buffer or a write buffer.

finished. In a system that contains more than one internal bus, the burst size is allowed to exceed the buffer size by inserting idle cycles on the bus when the dataflow network is empty or unable to accept data.

5.3.3 AGU interface

The DMA interface also supports an external address generation unit (AGU). It can automatically restart the DMA transfer from a new base address when the previous transfer has completed. Hence, there is no latency between transfers. This is useful when accessing several blocks of information inside a larger memory space, e.g. a matrix of blocks. An illustrating example is the transpose operation, which relocates blocks of data inside a matrix. The *block* is the actual transfer and *matrix* is the current address space.

Chapter 6

Optimizations

This chapter present results from the architectural decisions taken in Chapter 5. Focus is on optimizing the image combine unit and the two-dimensional FFT by changing memory access patterns, and to minimize internal buffering by sharing functionality. Modifying the original memory access patterns increases the memory transfer rate, while internal buffers are reused and shared between operations in order to save chip area.

6.1 Internal buffering

Buffering is required both on an algorithmic level as well as on architectural level. On an algorithmic level, buffers are used as time or sample delays. An example where delay is part of the algorithm is a reverb filter. A reverb filter creates the effect of sound reflected against walls in a room. The algorithm combines direct sound from the source with reverberated sound of reflections, in other words the delayed sound or echo. Hence, delay buffers are a part of the algorithm. Buffers are also required on an architectural level, for example when folding an algorithm. Taking the pipeline FFT as an example, delay feedback buffers are inserted to supply the butterfly units with the correct input sequence, calculating $x(n) + x(n + N/2)$. To access sample n and $n + N/2$ at the same time requires the sequence to be delayed with $N/2$ samples, if data appears on the input in sequential order.

The XSTREAM accelerator, shown in Figure 6.1, requires several buffers for storing, processing, and reordering information. However, by moving functionality between blocks, it is shown that buffers can be removed and memory requirements reduced. The unoptimized pipeline is described in Section 6.1.1, and possible optimizations are presented in Section 6.1.2. Buffers in the pipeline depend on two values, the minimum block transfer size N_{burst}, and the maximum transform size N_{FFT}. Based on simulations, the actual values for these constants are selected in Section 6.3.

6.1.1 Original pipeline

The unoptimized pipeline requires buffering in almost every unit. Starting with the DMA interface, data is read into a small buffer with the same size as the burst length N_{burst}. Data is then reordered, to extract information pixel-by-pixel, which also requires

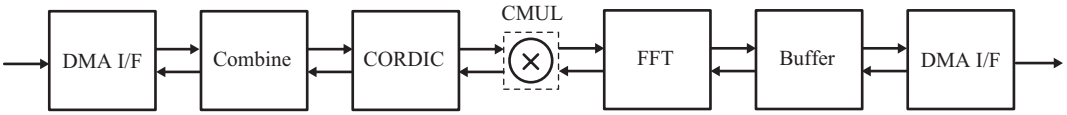


Figure 6.1: Functional units in the XSTREAM accelerator.

a buffer. After this, data is propagated to the CORDIC and complex multiplier, which are non-buffered processing units and only needs internal pipeline registers. The next unit is a one-dimensional pipeline FFT, requiring delay feedback elements. For a two-dimensional transform, a buffer is added to transpose the sequence before applying the one-dimensional FFT again. A buffer is also required for bit-reversing the FFT output, an operation that requires the complete sequence to be stored. FFT spectrum shift requires half the sequence to be stored in a buffer. Finally, the result is cached in the output DMA interface, before written back to main memory. The original buffer requirements in the XSTREAM pipeline can be found in Table 6.1.

6.1.2 Optimized pipeline

By moving functionality between blocks, buffer space can be saved. Starting with the reorder operation in the image combine block, this function is moved to the DMA input buffer. The only modification required is that the DMA input buffer should support both linear and interleaved addressing mode. The internal feedback in the one-dimensional FFT can not be removed or combined with any other block, since it is constantly being accessed. However, the transpose buffer for two-dimensional FFT transforms is reused for bit-reversal and FFT spectrum shift. Those are in fact only two different addressing modes, which are supported by the transpose buffer instead. A shared buffer saves a large amount of memory, since both bit-reverse and FFT shift requires the complete sequence and half the sequence to be stored, respectively. Table 6.2 shows the optimizations, moving buffers between functional units, and the memory requirements are found in Figure 6.5. The addressing mode for each operation is presented in Table 6.1, and is further explained in the next section.

Table 6.1: Required buffering for each processing block. Without optimization, the total memory is the sum of all buffers. Each buffer requires a different addressing mode, depending on the function that the block is evaluating.

Unit	Buffer	Buffer size	Addressing mode
DMA I/F	Input buffer	N_{burst}	Linear
Combine	Reorder	N_{burst}	Interleaved
FFT	FFT delay feedback	$N_{FFT}-1$	FIFO
Buffer	Transpose	${N_{burst}}^2$	Col-row swap
	Bit-reverse	N_{FFT}	Bit-reversed
	Spectrum shift	$N_{FFT}/2$	FFT shift
DMA I/F	Output buffer	N_{burst}	Linear

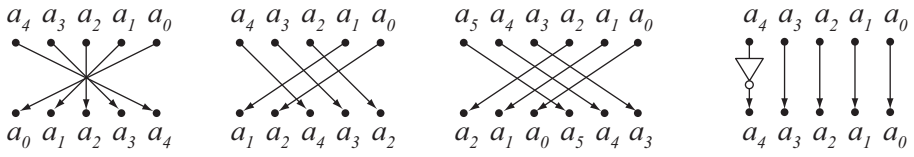
Table 6.2: Buffers can be reused for several operations. Buffer memory can also be shared between units to save storage space. The actual buffer size is the maximum of each individual operation.

Unit	Buffer size	Addressing mode
DMA I/F	N_{burst}	Linear
		Interleaved
FFT	$N_{FFT}-1$	FIFO
Buffer	$\max({N_{burst}}^2, N_{FFT})$	Col-row swap
		Bit-reversed
		FFT shift
		Linear

Addressing modes

The addressing mode controls the data output sequence from the buffer. Figure 6.2 shows how the address bits are rearranged in each mode. Data is always written to the buffer in linear addressing mode. When reading from the buffer, the address mode controls the sequence order. Mapping functionality onto the same buffers, as described in the last section, requires each buffer to support several modes, as stated in Table 6.1. The input buffer supports linear addressing and data interleaving. The buffer unit supports linear, bit-reversed, row and column swap for the transpose operation, and FFT spectrum shift.

Both bit-reverse and FFT spectrum shift depends on the transform size, which requires the addressing modes to be flexible. Since bit-reverse and FFT spectrum shift is often used in conjunction, this addressing mode can be optimized. The spectrum shift inverts the MSB, as shown in Figure 6.2(d), and the location of the MSB depends on the transform size. However, in bit-reversed addressing, the MSB is actually the LSB, and the LSB is always statically located. By reordering the operations, the cost for moving the spectrum is a single inverter.



(a) (b) (c) (d)

Figure 6.2: Buffers support multiple addressing modes and can be used for several operations. a_n represents the address bits, and the input address is a linear counter. (a) Bit-reversed addressing. (b) Interleaving with 4 groups and 8 values in each group. (c) Transpose for an 8×8 matrix by swapping rows and columns. (d) FFT spectrum shift by inverting the MSB.

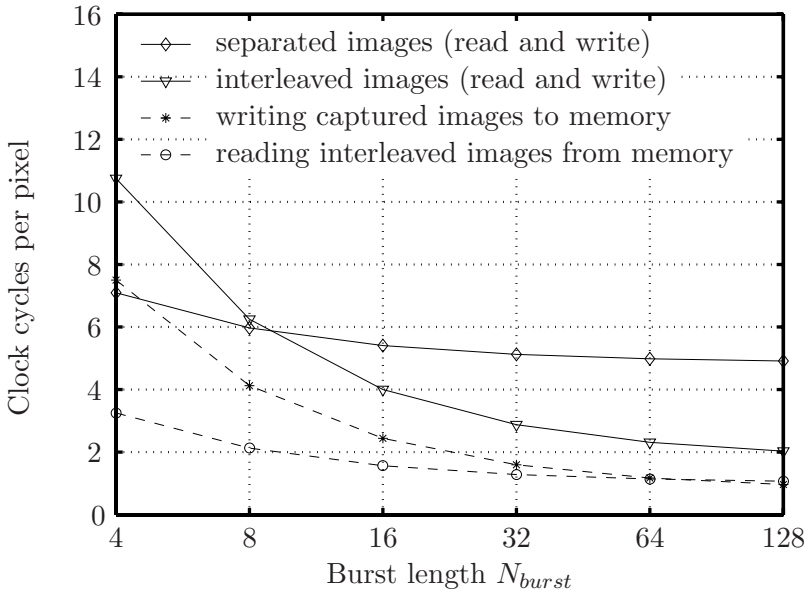


Figure 6.3: Average number of clock cycles for each produced pixel, including both write and read operation to and from memory. The dashed lines show the read and write part from the interleaved version.

6.2 Memory bandwidth

To achieve high memory transfer rate, the access pattern to the external memory is important. The memory has a non-uniform access time, which is described in Section 3.4. The combine unit increases the bandwidth by interleaving data for faster access. The transpose unit in the two-dimensional FFT increases the bandwidth by operating on smaller blocks of data, applying the divide-and-conquer approach.

6.2.1 Image combine unit

The combine module, described in Section 5.2.1, uses interleaving to increase overall bandwidth. Storing the images interleaved enables small burst transfers to be used. When reading the interleaved images, the burst length is equal to the buffer size, filling the complete buffer before reordering of the output. After a transfer, the buffer contains a set of pixels from four different images, and a simple reorder operation outputs the information pixel-by-pixel. By storing the images interleaved, burst transfers can be used both for storing and reading the images from memory. A comparison of the different approaches is shown in Figure 6.3.

Figure 6.3 shows how the reorder operation depends on the burst size. For a small burst length, it is faster to store the images separately. When the burst length increases, interleaving becomes a suitable alternative.

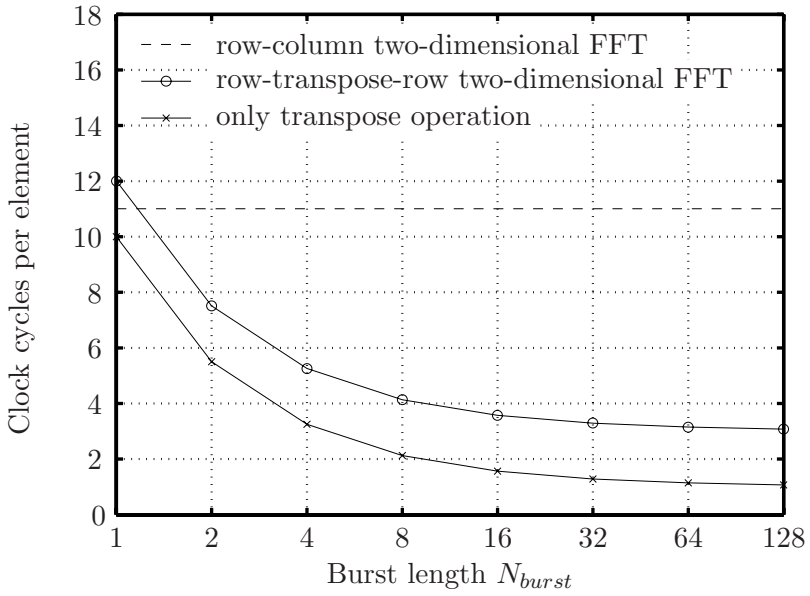


Figure 6.4: Number of clock cycles per element for calculating a two-dimensional FFT using the two methods. The transpose operation is also shown separately. Using a small burst size, the transpose operation dominates the processing time.

6.2.2 Two-dimensional FFT

The assumption in Section 5.2.3 is that transposing the matrix and process over rows is faster than processing the matrix over columns. However, transposing the large matrix would be time expensive, since it involves either reading or writing columns. By dividing the matrix into smaller blocks, each block can be individually transposed and relocated. Read and write operations are then performed in burst transfers and data is stored and transposed in a local buffer. Performance of the transpose operation depends on the internal buffer size.

Figure 6.4 shows a simulation of the row-column FFT and the row-transpose-row FFT. For the latter approach, the transpose operation is required, also shown separately in the graph. When the burst length is short, the transpose overhead dominates the computation time. When the burst length increases, the row-transpose-row FFT improves the overall performance.

6.3 Parameter selection

N_{burst} and N_{FFT} are selected based on simulations from previous sections. N_{FFT} is the maximum transform size and is chosen to 2048 in Chapter 2. Hence, it remains to select a value for N_{burst}, which is a trade-off between required buffer size and memory bandwidth.

The shared buffer unit in the pipeline must be at least the size of N_{FFT} to support the bit-reverse operation. It is reused for the transpose operation, which requires ${N_{burst}}^2$

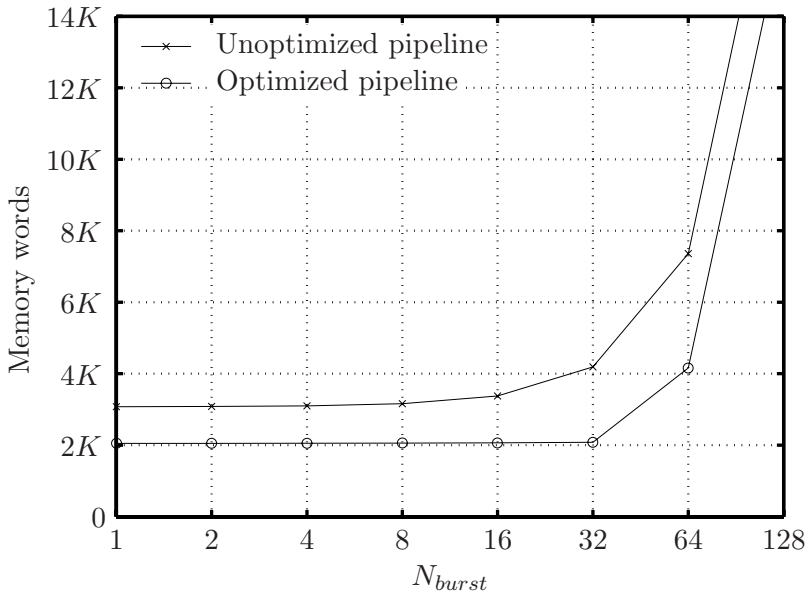


Figure 6.5: Memory requirements in words for different values on N_{burst} when N_{FFT}= 2048. The delay feedback memory is not included since it can not be shared with other units.

elements. The buffer size is hence the maximum of the two. The delay feedback memory in the FFT can not be shared and will not be included in the simulations. The total memory requirements for the optimized pipeline is therefore

$$\text{MEM} = N_{burst} + \max(N_{burst}^2, N_{FFT}) \tag{6.1}$$

Figure 6.5 shows how the memory requirements depend on N_{burst}. When ${N_{burst}}^2 > N_{FFT}$, the memory requirements rapidly increases, which has a high area cost and a relatively low performance improvement according to Figure 6.3 and Figure 6.4. However, selecting N_{burst}= $\sqrt{2048}$ complicates the transpose operation since it is not a power of 2. Selecting N_{burst}= 32 satisfies both conditions and has a relatively small overhead according to the simulations.

6.4 Summary

N_{burst} is chosen to minimize the buffer requirements and optimize the memory transfer rate. With N_{FFT}=2048, N_{burst} is chosen to 32. From Figure 6.4, it can be seen that this generates an overhead of about 20% in the transpose operation. Compared to a solution with uniform access time, the overhead for a two-dimensional row-transpose-row FFT is about 60% due to the extra transpose operation. But at the same time, overhead for a traditional row-column FFT is about 550%.

Chapter 7

System integration

This chapter presents the integration of the XSTREAM accelerator into an embedded system and a prototype of the holographic microscope. The functionality of the embedded system is to capture, reconstruct and present holographic images. The architecture is based a SPARC compatible microprocessor, the XSTREAM accelerator, and a memory controller for connecting external memory. Two interfaces provide functionality for capturing images from an external sensor device and to present reconstructed images on a monitor.

The chapter is divided into two sections. The first part describes hardware design, and how the XSTREAM accelerator is integrated together with a microprocessor and external interfaces. The second part describes software development and how to develop platform independent source code using a hardware abstraction layer.

7.1 Hardware design

Embedded systems are, in contrast to computers, designed for one specific task. They can be found in all kinds of electronic devices such as watches, DVD-players, video games, ATMs and car electronics. A system designed for the specific task of image reconstruction in digital holography can thus also be referred to as an embedded system.

To fully utilize the functionality of the XSTREAM hardware accelerator, it is integrated together with a CPU core and a memory controller to form a complete embedded system for digital holographic imaging [35]. The system also includes a controller to connect an external sensor device and a VGA generator to connect an external monitor. Figure 7.1 shows the complete system, using an internal bus architecture.

7.1.1 Integrated soft processor

A soft microprocessor core, designed for embedded applications, extends system flexibility and programmability. The embedded microprocessor does not have the same computational efficiency as a hardware accelerator, but is useful to control and configure on-chip modules. The word *soft processor* means that the design is available as a *soft macro*, i.e. as source code. Unlike *hard macros*, which are placed and routed designs for a specific fabrication process, soft macros can still be configured and modified. The

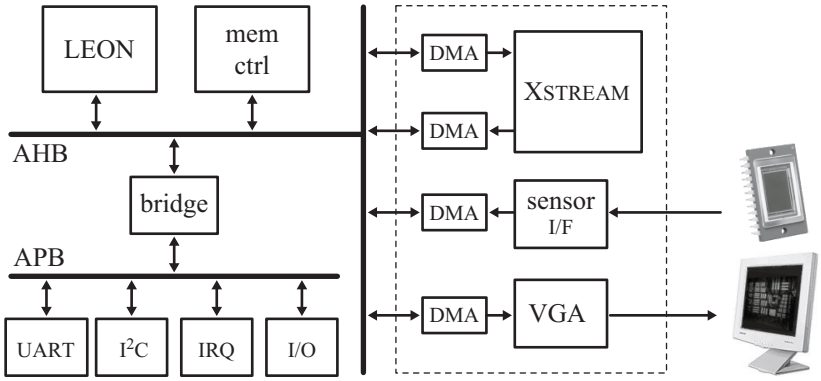


Figure 7.1: The system is based on a CPU (LEON), a hardware accelerator, a high-speed memory controller, an image sensor interface and a VGA controller.

soft macro has lower performance and logic density, but available for fabrication in any technology and for rapid prototyping with programmable logic.

The processor is used for configuration of the internal units and for controlling the user interface. It is distributed under the name LEON, and is compatible with the 32-bit SPARC V8 architecture. It was originally developed by the European Space Agency, but is now maintained by Gaisler Research [36]. LEON is chosen since it is available as open-source code, uses free development tools and is continuously updated and improved. The processor has an AMBA bus interface and an extended configuration for both FPGA and ASIC synthesis. When targeting FPGA devices, an alternative solution could be the MicroBlaze soft processor from Xilinx [37]. This would probably require fewer FPGA resources but will limit the design to run on only Xilinx devices. Another approach would be to use the embedded IBM PowerPC 405 available on Xilinx Virtex-II Pro. However, the choice of CPU is non-crucial for the project itself, since it is merely being used for control and configuration.

7.1.2 External memory interface

The memory controller interface provides the system with 128 MB of external SDRAM. This rather large memory bank is required to store high-resolution images from the sensor as well as intermediate calculation results and tables. The hardware accelerator, sensor interface and VGA controller have direct memory access through the DMA units. The highest priorities on the bus are assigned to the sensor and VGA controllers, which have the most demanding real-time constraints, since data is continuously streaming and a delay would cause data loss. The hardware accelerator is non-sensitive to delays and is therefore assigned a lower priority.

7.1.3 Sensor interface

The sensor interface is configurable to support several different image sensor devices. With a programmable vertical and horizontal synchronization delay and image size, it is possible to capture images from any location on the sensor. For compatibility with

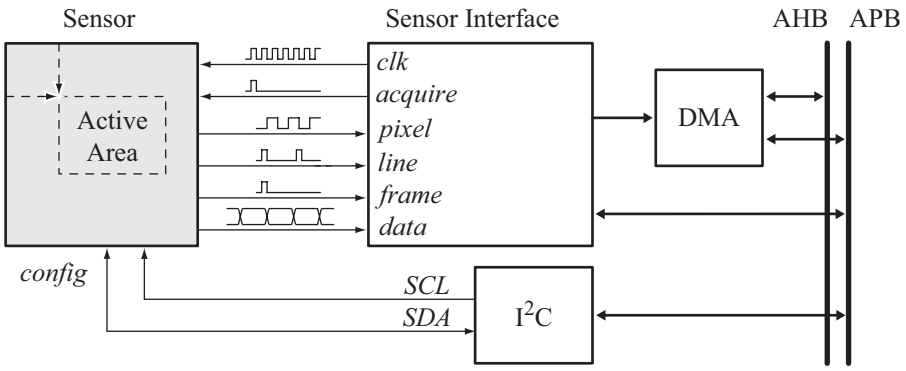


Figure 7.2: The sensor interface can be configured to work with a range of sensor devices. The information from the sensor is transferred to main memory by the DMA controller. Grey boxes represent external components.

various image sensors, the interface supports a resolution of up to 12 bits. Table 7.1 shows the specification of the two sensor devices currently used in the project. The sensor is clocked at a lower clock frequency than the system, a value configurable from the sensor interface. When the acquire signal is asserted, the sensor device starts the integration phase. After the image has been captured, data streams out from the sensor. Output signals from the sensor indicates start of a new frame, new line and when pixels are valid. The interface is illustrated in Figure 7.2. The *acquire* signal request the sensor to capture an image. Pixel *data* is produced by the sensor and synchronized using *frame*, *line*, and *pixel* clocks. The CMOS sensor have on-chip circuitry and can be configured using a serial interface supporting the inter-integrated circuit (I^2C) protocol.

7.1.4 Monitor interface

The result from the image reconstruction is presented on an external monitor, and the integrated VGA controller produces pixel data and synchronization signals. An external digital-to-analog (DAC) is connected between the VGA controller and the monitor, as shown in Figure 7.3. This circuit also contains a color palette, translating the 8-bit values from the VGA controller to 24-bit (3×8) color. The palette is configured using a parallel I/O interface.

The other side of the VGA controller is connected to the internal AHB bus using a DMA controller, continuously streaming video data from a specific location in the main memory, referred to as video area. The DMA base address indicates the location from

Table 7.1: Supported sensor devices. N_{Sx}, N_{Sy} and Δx is the number of pixels and the pixel size in the x, y directions respectively.

Device	N_{Sx}	N_{Sy}	$\Delta x = \Delta y$	Precision	Frame Rate
KAF3200E (CCD)	2184	1472	$6.8\,\mu m$	12 bits	2 fps
LM9638 (CMOS)	1032	1312	$6.0\,\mu m$	10 bits	18 fps

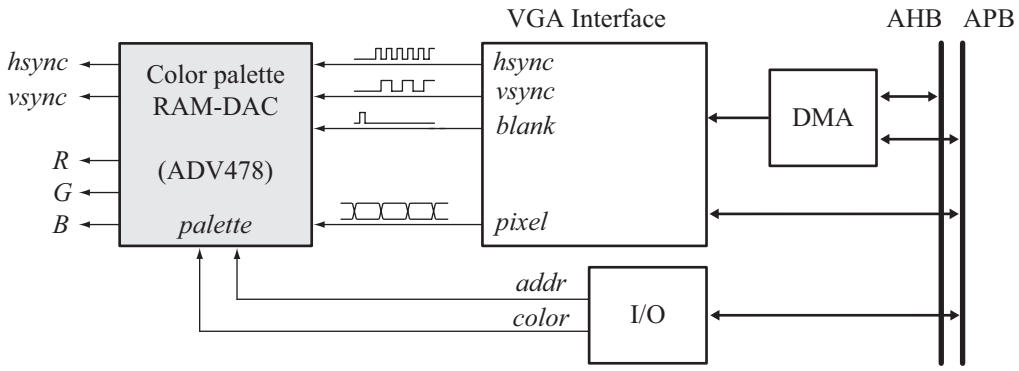


Figure 7.3: The VGA interface connects to an external monitor to display images from the microscope. Grey boxes represent external components.

which data is copied, and the DMA transfer size is equal to the dimensions of the screen. High-level functionality such as printing text on the screen and drawing bitmap images is handled in software. The graphical user interface is hence fully software controlled.

7.2 Software design

Software design is a major part of the project. The main purpose of the software is to execute a reconstruction program on the embedded processor, which utilizes the hardware resources for capturing, processing and presenting images. The software program is also capable of emulating hardware resources when they are not available. Hence, the system can run on any computer with a C compiler. The software also includes a floating-point model, which can be activated to compare the results from the computations with the full-precision result. The layers between the software application and the actual hardware is shown in Figure 7.4.

7.2.1 Hardware API

The application programming interface (API) is a set of instructions or rules that enable different part of a system to communicate. Three APIs has been defined to simplify communication between the hardware accelerator, sensor and monitor device. The function calls are found in Table 7.2. Applications use the API to communicate with hardware, e.g. capturing images from the sensor or drawing on the screen. Higher level functions are defined on top of the API, e.g. the graphics device interface (GDI). The GDI supports additional graphical function for printing text and defining fonts, manipulating bitmaps and opening a basic terminal window. These functions are useful not only for creating a graphical user interface, but also for debugging the program.

7.2.2 Hardware abstraction layer

The reconstruction software is intended to execute on the embedded processor, but it is more convenient to develop and debug the application on a computer. The only problem is that the hardware accelerator and sensor is not available, and must be emulated. To

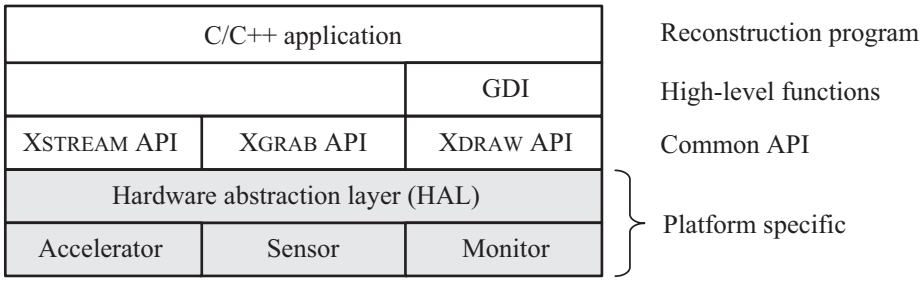


Figure 7.4: Hardware abstraction layer separating the software from the platform specific parts of the system.

simplify the development, hardware and software is divided using a hardware abstraction layer (HAL). The software side of the HAL is the API from the previous section and the hardware side implements platform specific functionality. With this approach, it is possible to transparently change underlying hardware without affecting system behaviour. Another reason for using a hardware abstraction layer is that software and hardware can be developed in parallel, since they communicate using a well-defined interface.

On the embedded system, platform specific drivers control the underlying hardware through configuration registers. On a desktop computer, hardware functionality must be completely emulated. Switching between hardware and emulated hardware is done transparently during compile time, based on the current platform.

The emulation procedure is different for each hardware function. Capturing images from the sensor is on the computer implemented by reading a bitmap image from a file. From an application point of view, the behaviour of the system is the same. Output to a monitor is emulated with a graphical window, as shown in Figure 7.5. Graphics

Table 7.2: API function calls for the hardware accelerator, sensor and monitor device.

API	Functions
XSTREAM	`fft1D(int* dst, int* src, int length, int params)`
	`fft2D(matrix* dst, matrix* src, int params)`
	`memcpy(int* dst, int* src, int size)`
	`transpose(int* dst, int* src, dim size)`
	`clear(matrix* dst)`
	`execute(matrix* dst, matrix* src, int params)`
XGRAB	`get(SensorDevice* dev, matrix* dst, rec region)`
XDRAW	`init(screen* s, int resolution)`
	`clear(screen* s)`
	`putpixel(screen* s, point position, pixel value)`
	`getpixel(screen* s, point position)`
	`draw(screen* s, bitmap* bmp, point position)`
	`setpalette(palette* pal)`

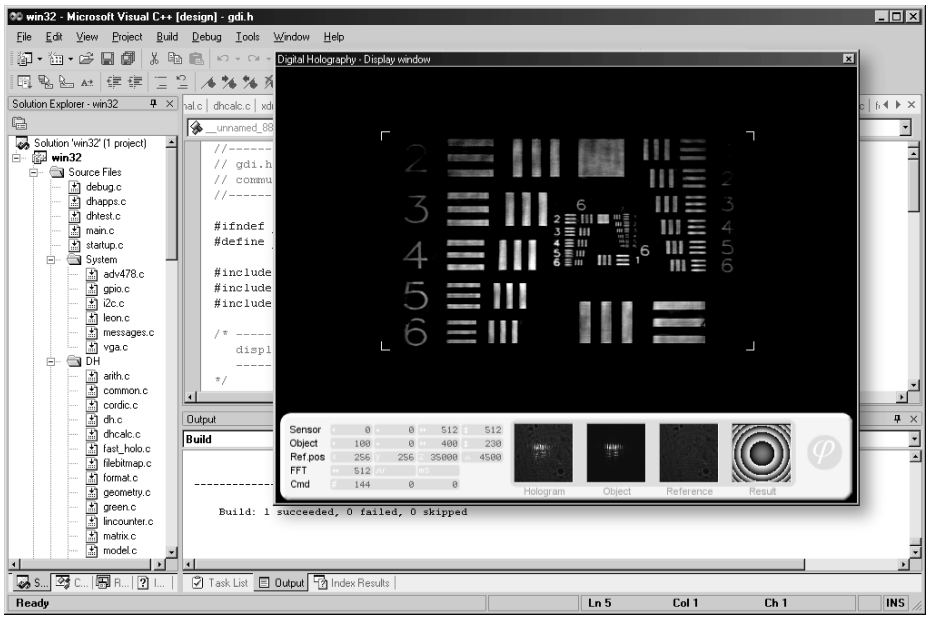
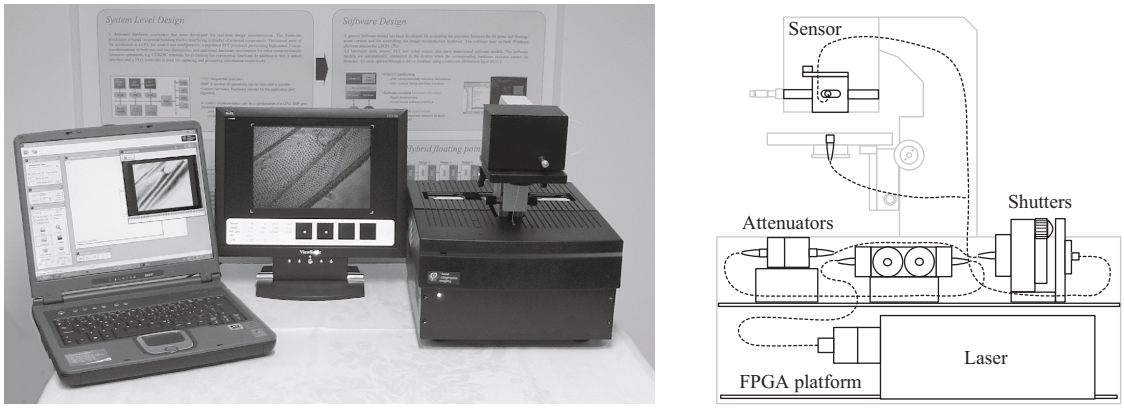


Figure 7.5: The platform independent software running in Microsoft Visual Studio. The hardware accelerator, sensor and monitor is emulated in software.

is rendered directly on the screen instead of being copied to the video memory area. The functionality of the hardware accelerator is completely emulated in software, and support one- and two-dimensional FFTs and other functions specified by the API.

7.2.3 **Memory management**

Some of the C library functions can cause problems in the interface between hardware and software. An example is the *malloc* function, which allocates memory on the heap. The problem with malloc is that it can allocate memory starting from any arbitrary memory address. However, DMA units transfer blocks of data, which are consecutive memory addresses. SDRAMs are indexed using rows and columns, and a burst transfer spanning over two rows is not valid and results in unpredictable behaviour. Hence, the start address of a memory area should be aligned to the burst size. Since the burst size is not a fixed value, it is more reliable to align the start address to the first column in a row. The modified malloc function simply allocates slightly more memory than required, then selecting a start address inside this memory space that satisfies the alignment condition.



(a) (b)

Figure 7.6: (a) The prototype containing optics and the embedded system for image reconstruction. The system is controlled using an external laptop running a graphical user interface. The resulting images are presented on a monitor. (b) Cross-section of the holographic microscope.

7.3 Prototype - A Holographic Microscope

The embedded system is integrated in a prototype of a holographic microscope, shown in Figure 7.6(a). The light source is a 633 nm He-Ne laser, connected to a single mode optical fiber with a core diameter of 4 μm. The laser beam is focused onto the fiber core using a lens, to fully utilize the intensity from the source. In the other end the optical fiber is connected to a beam splitter, dividing the light into two separate fibers for the object and reference light, respectively. The light intensity in each fiber is controlled using an optical attenuator and can also be completely blocked using an electric shutter. The electric shutters are used when capturing the images, blocking one beam at a time. A beam splitter cube combines the light from the fibers, one fiber illuminating the object and one used as a point reference source. The sensor is mounted close to the beam splitter cube and connected to the embedded system, previously described in this chapter. A cross-section of the holographic microscope is shown in Figure 7.6(b).

7.3.1 FPGA platform

The embedded system is running inside an FPGA on a custom designed development board. The board contains interface to the sensor device and a connection to an external monitor. A single 128 MB 32-bit external memory is connected to the FPGA. This limits the bandwidth compared to a dual memory configuration, but is chosen for practical design reasons due to limited expansion possibilities.

The embedded system controls the electric signals to the sensor and the shutters. Before the images can be captured, the sensor must be configured with an active window, a black level compensation value, integration time, and readout directions. The active window specifies where on the sensor to capture information. Selecting a small active

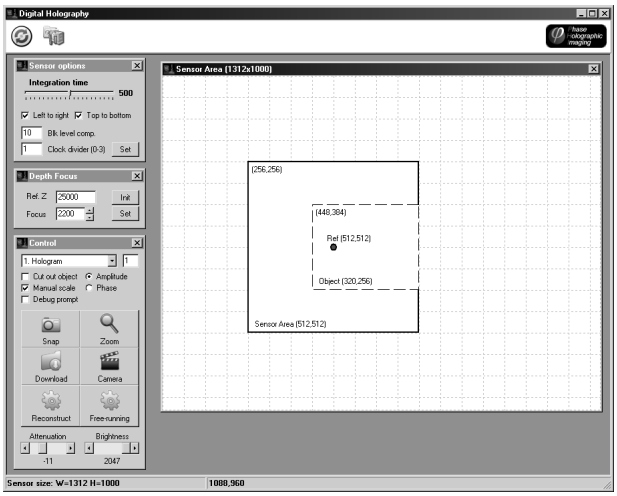
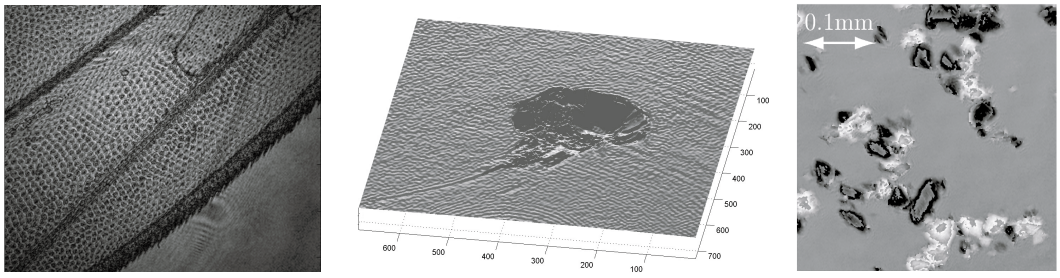


Figure 7.7: The system is controlled through the graphical user interface (GUI), to capture images, reconstruct images and download results. The white window represents the sensor device, and the boxes show the active window of the sensor and the location of the object and the reference point.

window reduces the computation time but also the quality. Black level compensation is a value to compensate for the natural offset of the pixel array. In other words, black should be represented with the value 0. The integration time controls the time during which the sensor array is illuminated by light. A longer integration time generates brighter images, but can cause the pixels to saturate. A short integration time results in a darker image with a higher noise ratio. Selecting an integration time to utilize the full dynamic range is important for the resulting quality.

7.3.2 User interface

The prototype can be connected to an external computer to control the functionality. From a graphical interface, users can control the microscope to capture images, reconstruct holographic images, and download the result. The graphical user interface is shown in Figure 7.7. Initially, a user selects a working area, i.e. the active window on the sensor. The FFT size is then automatically adjusted to cover the selected image area, zero-padding the unused space. When a user starts the reconstruction processes, the prototype will begin to capture and process images. The image is reconstructed using the hardware accelerator, and presented on an external monitor. The result and intermediate images can be downloaded to a computer for further processing. Example of images captured and processed with the holographic microscope (HoloMicro$^{\text{TM}}$) is presented in Figure 7.8.



(a) (b) (c)

Figure 7.8: Images captured in the holographic microscope. (a) Amplitude image of a fruit-fly wing. (b) Extracted depth information from a phase image of a slightly damaged microscope slide. (c) Unwrapped and background-corrected phase image of a mixture with KCl and NaCl crystals. The crystals are separated by their refractive index and identified as bright and dark objects, respectively.

7.3.3 Results

A comparison between the prototype, a standard computer, and an ASIC solution is shown in Table 7.3. The last two rows in the table shows recent related work scaled to the same transform size, and the speed index measures the effectiveness per MHz for the different solutions. The overhead represents time used by the processor and the monitor interface, continuously transferring data from memory to an external monitor. As the clock frequency increases, the overhead is reduced.

Scaling the clock frequency and frame rate between the prototype and a standard computer shows that the hardware accelerator in the prototype is 120 times more efficient. The computer is running nearly 50 times faster, hence the actual speed-up is about 2.5. An ASIC solution is more suitable since it allows a significantly increased clock frequency. Based on a dual bus and memory architecture, this solution nearly complies with the initial specification in Chapter 2, reconstructing holographic images at 20 frames per second.

Table 7.3: Comparison between the FPGA prototype, a standard computer, and an ASIC solution. The transform size is 2048×2048. Speed index is fps/freq, normalized to 1.0 for the prototype.

Target	Technology	Frequency (MHz)	Mem I/F	Overhead	Rate (fps)	Speed index
PC	Pentium-III	1130	Single	0	0.2	.008
FPGA	Virtex-E 1000	24	Single	25%	0.5	1.0
			Dual		1.5	3.0
ASIC	0.13 μm	250	Dual	2.5%	20	3.8
Miyamoto [38]	0.35 μm	133	Single	0	2.6	0.9
Uzin [39]	Virtex-E 2000	35	Quad	0	2	2.7

Chapter 8

Conclusions

A hardware accelerator for digital holographic imaging has been presented, along with a fully functional prototype of a holographic microscope. The hardware platform includes a soft processor with a burst oriented memory controller, sensor and monitor interface, and a two-dimensional signal processing datapath. Data is efficiently transferred using DMA units, capable of performing two-dimensional burst operations. All components, except for the soft processor, has been developed in this work.

The aim has been to construct a hardware accelerator with high computation efficiency, low on-chip memory requirements, and a high bandwidth to external memory. This thesis cover the work on developing a one-dimensional FFT core, constructing a two-dimensional hardware accelerator, and integrating the building blocks on system level.

A flexible one-dimensional FFT has been fabricated in a 0.35 μm five metal layer CMOS process from AMI Semiconductor. The FFT core supports a hybrid floating-point format, generating high precision with low area requirements. Several memory optimization schemes are presented, which shows that it is possible to save memory by implementing an FFT with fixed-point input and hybrid floating-point output. Furthermore, an architecture combining convergent block floating-point with hybrid floating-point can save buffer memory by selecting an optimum block size for each delay feedback memory. It is also shown that a bi-directional pipeline is suitable for convolution by avoiding the memory demanding bit-reverse operation.

The two-dimensional accelerator requires several buffers for storing, processing and reordering information. However, it is shown that by moving functionality between blocks, buffers can be removed and memory requirements reduced. Several operations can share a single buffer by supporting multiple addressing modes, for example bit-revered, spectrum shift, and matrix transpose.

Modern dynamic memories are burst oriented and have a non-uniform access time, which depends on the memory access pattern. Several optimization schemes are presented to increase the memory bandwidth, for example data interleaving and using a divide-and-conquer approach for transposing large data sets. In the former case, storing the images interleaved enables burst transfers for storing and reading the holographic images from

memory. In the latter case, the memory bandwidth is increased by introducing a fast transpose unit. It is shown that transposing the memory and applying FFTs over rows are faster than applying the FFT over columns. The efficient transpose unit drastically improves the performance of the two-dimensional FFT.

Chapter 9

Future work

The resolution and frame rate of digital image sensors are continuously increasing. With improved quality, holographic microscopes could seriously challenge existing technologies and provide additional features. A larger sensor device requires more processing to reconstruct holographic images, hence an even greater need for hardware acceleration.

A digital holographic microscope has been constructed as a proof of concept. It is based on a relatively small sensor device and a hardware platform based on an FPGA instead of a fabricated ASIC. The prototype is therefore limited in image quality and reconstruction time. The next step is to migrate to a high-resolution sensor, moving from a modest 1.3 million pixels to a 6.6 million pixels, which is currently under evaluation.

On the hardware side, two approaches are considered. The first is to fabricate the hardware accelerator in a modern technology process, for example UMC 0.13 μm, which is available for the university. Fabricating an ASIC would result in reconstruction hardware that complies with the speed requirements in Chapter 2. The other approach is to continue using programmable logic, but integrating the hardware accelerator in a standard computer. PC-cards that contain programmable logic is currently available for both desktop computers and laptops. The advantage is that it combines the best of two worlds, the generic microprocessor and the dedicated hardware accelerator. Hence, functionality not supported in hardware can instead be evaluated in software.

In this work, memory optimizations are two-dimensional while modern dynamic memories actually have a three-dimensional organization. Implementing a memory controller with access scheduling could improve memory bandwidth even further. Exploring different external memory configurations would also be of great interest.

Bibliography

[1] D. Burger, J. Goodman, and A. Kagi, "Limited bandwidth to affect processor design," *IEEE Micro*, vol. 17, no. 6, pp. 55–62, Nov. 1997.

[2] W. E. Kock, *Lasers and Holography.* 180 Varick Street, New York 10014: Dover Publications Inc., 1981.

[3] U. Schnars and W. Jueptner, *Digital Holography.* Springer-Verlag, Berlin, Heidelberg: Springer, 2005.

[4] M. Gustafsson, M. Sebesta, B. Bengtsson, S.-G. Pettersson, P. Egelberg, and T. Lenart, "High resolution digital transmission microscopy - a Fourier holography approach," *Optics and Lasers in Engineering*, vol. 41(3), pp. 553–563, Mar. 2004.

[5] W. Xu, M. Jericho, I. Meinertzhagen, and H. Kreuzer, "Digital in-line holography for biological applications," *Cell Biology*, vol. 98, no. 20, Sept. 2001.

[6] E. O. Brigham, *The Fast Fourier Transform and its Applications.* Englewood Cliffs, New Jersey: Prentice-Hall, 1988.

[7] Transmeta Corporation, "Transmeta Efficeon TM8620 Processor," http://www.transmeta.com, September, 2004.

[8] K. K. Parhi, *VLSI Digital Signal Processing Systems.* 605 Third Avenue, New York 10158: John Wiley and Sons, 1999.

[9] U. Kapasi, S. Rixner, W. Dally, B. Khailany, J. H. Ahn, P. Mattson, and J. Owens, "Programmable stream processors," *Computer*, vol. 36, no. 8, pp. 54–62, Aug. 2003.

[10] U. Kapasi, W. Dally, S. Rixner, J. Owens, and B. Khailany, "The Imagine Stream Processor," in *Proc. of IEEE International Conference on Computer Design: VLSI in Computers and Processors*, Freiburg, Germany, Sept. 16-18 2002, pp. 282–288.

[11] ISO/IEC 11172-3, "Coding of moving pictures and associated audio for digital storage media," Part 3 - Audio, ISO Standard, 1993.

[12] T. Lenart and S. Gadd, "A Hardware Accelerated MP3 Decoder with Bluetooth Streaming Capabilities," Master Thesis, Lund University, Sweden, 2001.

[13] T. Gleerup, H. Holten-Lund, J. Madsen, and S. Pedersen, "Memory architecture for efficient utilization of SDRAM: a case study of the computation/memory access trade-off," in *Proceedings of the 8th International Workshop on Hardware/Software Codesign*, San Diego, CA USA, May 3-5 2000, pp. 51–55.

[14] W. Xu, M. Jericho, I. Meinertzhagen, and H. Kreuzer, "An efficient memory arbitration algorithm for a single chip MPEG2 AV decoder," *IEEE Transactions on Consumer Electronics*, vol. 47, no. 3, Aug. 2001.

[15] S. Rixner, W. Dally, U. Kapasi, P. Mattson, and J. Owens, "Memory access scheduling," in *Proc. of the 27th International Symposium on Computer Architecture*, Vancouver, BC Canada, June 10-14 2000, pp. 128–138.

[16] T. Lenart and V. Öwall, "A 2048 complex point FFT processor using a novel data scaling approach," in *Proc. of IEEE International Symposium on Circuits and Systems (ISCAS'03)*, Bangkok, Thailand, May 25-28 2003, pp. 45–48.

[17] IEEE 802.11a, "High-speed Physical Layer in 5 GHz Band," http://ieee802.org, 1999.

[18] IEEE 802.11g, "High-speed Physical Layer in 2.4 GHz Band," http://ieee802.org, 2003.

[19] P. Capozza, B. Holland, T. Hopkinson, and R. Landrau, "A single-chip narrowband frequency-domain excisor for a Global Positioning System (GPS) receiver," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 3, pp. 401–411, Mar. 2000.

[20] J. Cooley and J. Tukey, "An algorithm for machine calculation of complex Fourier series," *IEEE Journal of Solid-State Circuits*, vol. 19, pp. 297–301, Apr. 1965.

[21] K. Zhong, H. He, and G. Zhu, "An ultra high-speed FFT processor," in *International Symposium on Signals, Circuits and Systems*, Lasi, Romania, July 10-11 2003, pp. 37–40.

[22] S. He, "Concurrent VLSI Architectures for DFT Computing and Algorithms for Multi-output Logic Decomposition," Ph.D. dissertation, Lund University, Lund, Sweden, 1995.

[23] S. Kim, K.-I. Kum, and W. Sung, "Fixed-Point Optimization Utility for C and C++ Based Digital Signal Processing Programs," *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, vol. 45, no. 11, pp. 1455–1464, Nov. 1998.

[24] G. Even and W. J. Paul, "On the Design of IEEE Compliant Floating Point Units," *IEEE Transactions on Computers*, vol. 49, no. 5, pp. 398–413, May 2000.

[25] K. Kalliojrvi and J. Astola, "Roundoff Errors is Block-Floating-Point Systems," *IEEE Transactions on Signal Processing*, vol. 44, no. 4, pp. 783–790, Apr. 1996.

[26] E. Bidet *et al.*, "A Fast Single-Chip Implementation of 8192 Complex Point FFT," *IEEE Journal of Solid-State Circuits*, vol. 30, pp. 300–305, Mar. 1995.

[27] A. Berkeman, "ASIC Implementation of a Delayless Acoustic Echo Canceller," Ph.D. dissertation, Lund University, Lund, Sweden, 2002.

[28] F. Kristensen, "Design and Implementation of Flexible OFDM Hardware," Licentiate Thesis, Lund University, Sweden, 2004.

[29] Christophe Del Toso *et al.*, "0.5-μm CMOS Circuits for Demodulation and Decoding of an OFDM-Based Digital TV Signal Conforming to the European DVB-T Standard," *IEEE Journal of Solid-State Circuits*, vol. 33, no. 11, pp. 1781–1792, Nov. 1998.

[30] B. Parhami, *Computer Arithmetic*. 198 Madison Avenue, New York 10016: Oxford University Press, 2000.

[31] T. Lang and E. Antelo, "High-throughput 3D Rotations and Normalizations," in *Proc. of Signals, Systems and Computers*, Pacific Grove, CA USA, Nov. 4-7 2001, pp. 846–851.

[32] C.-S. Wu and A.-Y. Wu, "Modified vector rotation CORDIC algorithm and its application to FFT," in *Proc. of IEEE International Symposium on Circuits and Systems (ISCAS'00)*, Geneva, Switzerland, May 28-31 2000, pp. 529–532.

[33] M. R. Portnoff, "An Efficient Method for Transposing Large Matrices and Its Application to Separable Processing of Two-Dimensional Signals," *IEEE Transactions on Image Processing*, vol. 2, no. 1, pp. 122–124, Jan. 1993.

[34] ARM Ltd., "AMBA Specification - Advanced Microcontroller Bus Architecture," http://www.arm.com, 1999.

[35] T. Lenart, V. Öwall, M. Gustafsson, M. Sebesta, and P. Egelberg, "Accelerating signal processing algorithms in digital holography using an FPGA platform," in *Proc. of International Conference on Field-Programmable Technology, FPT'03*, Tokyo, Japan, Dec. 15-17 2003, pp. 387–390.

[36] J. Gaisler, "A portable and fault-tolerant microprocessor based on the SPARC v8 architecture," in *Proc. of Dependable Systems and Networks*, Washington DC, USA, June 23-26 2002, pp. 409–415.

[37] Xilinx, "MicroBlaze Soft Processor Core," http://www.xilinx.com/microblaze.

[38] N. Miyamoto, L. Karnan, K. Maruo, K. Kotani, and T. Ohmi1, "A Small-Area High-Performance 512-Point 2-Dimensional FFT Single-Chip Processor," in *Proc. of European Solid-State Circuits (ESSCIRC'03)*, Estoril, Portugal, Sept. 16-18 2003, pp. 603–606.

[39] I. Uzun, A. Amira, and F. Bensaali, "A reconfigurable coprocessor for high-resolution image filtering in real time," in *Proc. of the 10th IEEE International Conference on Electronics, Circuits and Systems*, Sharjah, United Arab Emirates, Dec. 14-17 2003, pp. 192–195.