

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



Compressive Sensing for Image Compression and Recovery

João Carlos Mateus da Silva Martins

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Advisor: Vítor Manuel Grade Tavares

Co-advisor: Xin Li (Carnegie Mellon University)

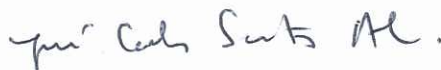
July 28, 2015

A Dissertação intitulada

“Compressive Sensing for Image Compression and Recovery”

foi aprovada em provas realizadas em 23-07-2015

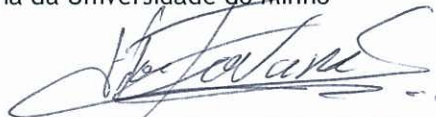
o júri



Presidente Professor Doutor José Carlos dos Santos Alves
Professor Associado do Departamento de Engenharia Eletrotécnica e de
Computadores da Faculdade de Engenharia da Universidade do Porto



Professor Doutor Jorge Miguel N. S. Cabral
Professor Auxiliar do Departamento de Electrónica Industrial da Escola de
Engenharia da Universidade do Minho



Professor Doutor Vitor Manuel Grade Tavares
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores
da Faculdade de Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projeto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extratos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são corretamente citados.



Autor - João Carlos Mateus da Silva Martins

Resumo

Um dos processos fundamentais em qualquer sistema digital que necessite de interagir com sinais e processos físicos é a amostragem, sendo responsável pela ligação entre o domínio digital e analógico. Este processo tem como objetivo aproximar um sinal contínuo através de medições discretas num certo espaço, tempo ou outro domínio. Utilizando o teorema de Nyquist, é conhecido que a reconstrução perfeita de um sinal a partir de um conjunto de medições discretas é possível desde que a frequência de amostragem seja pelo menos igual ao dobro da maior componente de frequência do sinal. Enquanto este procedimento tem sido utilizado nas mais variadas aplicações, existem casos em que a frequência necessária é manifestamente alta, resultando num volume de dados demasiado elevado para ser processado eficientemente ou até o material necessário para implementar o sistema ser demasiado caro. Por estas razões, a procura por novas soluções de amostragem é necessária. *Compressive sensing* emergiu como uma promissora técnica para amostrar, processar e comprimir um grande leque de sinais como por exemplo imagem, vídeo e sinais biomédicos, tirando partido da sua esparsidade para conseguir operar a frequências de amostragem abaixo das estipuladas pelo teorema de Nyquist, dando origem a um volume substancialmente inferior de dados. Enquanto os resultados desta técnica podem parecer bastante apelativos, a sua aplicação engloba a resolução de um sistema linear onde a solução é esparsa. Obter essa solução é computacionalmente intensivo, resultando numa operação bastante lenta. Uma maneira de acelerar o processo será aproveitar as capacidades de paralelização oferecidas por uma *field-programmable gate array* (FPGA).

Ao longo dos anos, vários algoritmos de recuperação têm sido propostos, uns focando-se mais na exatidão do resultado sem ter em atenção o tempo e esforço computacional necessários, outros tentando achar uma boa aproximação no menor tempo possível e utilizando o menor número de recursos, aceitando um pequeno erro na recuperação.

Esta dissertação apresenta e estuda a implementação em FPGA de um destes algoritmos de recuperação, o algoritmo *iterative hard-thresholding* (IHT).

Abstract

Sampling is a fundamental procedure in every digital system that needs to operate over physical signals and processes, being responsible for the bridge between the digital and analog domain. This process aims to approximate a continuous signal by taking discrete measurements (samples) over a certain space, time or other domain. Using the Nyquist theorem, it is widely known that it is possible to perfectly reconstruct a signal from a discrete set of samples as long as the sampling frequency is at least twice the maximum frequency component of the signal. While this approach has been used for a wide range of applications, in some cases the required sampling frequency is too high, the data volume derived from traditional sampling is too big to process or even the hardware too expensive. For these reasons a search for new sampling options is needed. Compressive sensing emerged as a promising technique to sense, process and store a broad range of signals such as image, video and biomedical signals leveraging on their sparsity and resulting in sub-Nyquist sampling frequencies and lower volumes of data. While the technique results may seem very attractive, its application also includes solving a linear system in which the solution is sparse. Obtaining such solution is computationally expensive, resulting in slow operation. One way to accelerate this process is to take advantage of the parallelization features of field-programmable gate arrays.

Over the years several compressive sensing algorithms have been proposed, some focusing in accuracy of the results with little regard for the time and computational efforts needed, others trying to achieve a good approximation in the least time possible and with fewer resources, accepting a small amount of error.

This dissertation presents and studies the implementation in an FPGA of one of these compressive sensing recovery algorithms, the iterative hard-thresholding (IHT) algorithm.

Agradecimentos

Este trabalho representa a conclusão de cinco anos de não só estudo e muito trabalho mas também de crescimento pessoal. Porque a vida não é feita só de aulas, estes anos de momentos, experiências e mesmo fracassos (sim, porque a vida não são só vitórias) ajudaram-me a crescer e tornar-me em quem sou hoje, e por isso estou grato a todos os que me acompanharam neste caminho.

À minha família, primeiro e mais importante do que quaisquer outras pessoas, e especialmente ao meus pais quero agradecer por tudo o que me deram ao longo da minha vida. Serei eternamente grato pelo apoio incondicional em todas as minhas decisões, os valores e princípios que me passaram, a educação e conselhos, o amor e carinho que me dão todos os dias e tantas outras coisas. O que me deram fez de mim a pessoa que sou hoje, são as bases de quem sou. Sem o apoio que me deram, não teria chegado onde estou e por isso estou extremamente grato.

Aos meus orientadores Professor Vítor Tavares e Professor Xin Li, por todo o apoio que me deram no desenvolvimento deste projecto. Gostaria de agradecer ao Professor Xin Li por me possibilitar desenvolver o projeto da minha dissertação na Carnegie Mellon University assim como pelos seus conselhos, ideias e ajuda dados durante a minha estadia. A oportunidade de trabalhar numa universidade americana de renome permitiu-me aprender e ter contacto com novos ambientes e por isso estou grato. Ao Professor Vítor Tavares um obrigado por toda a ajuda, sugestões e mesmo a ocasional garantia, quando necessário, de que haveria sucesso. Sem a vossa ajuda, o projecto não poderia ter sido terminado. Finalmente, um obrigado ao Minho Won pela ajuda fornecida durante a minha estadia na CMU.

A todos os Professores que tive o privilégio de conhecer e que me ensinaram, pela contribuição para a minha formação como Engenheiro.

Por último, mas não menos importante, algumas palavras para todos os amigos que partilharam estes anos comigo. À Lena, Nuno, Diana e Rita, por depois de todos estes anos de experiências, crescimento e vivências, continuarem sempre presentes. Ao Fred, Joana, Hernâni, Manuel e companhia por todas as conversas a altas horas da noite, caos e bons tempos. Ao Bernardo, César e Francisco, a eterna Equipa CMU, por todos os momentos estúpidos, *brainstorms* e tolices de viagem. Ao Diogo, Vinícius, Pedro Amaral, Mário, Duarte e Carlos por todas as ajudas e momentos de relax. Finalmente e mais importante, a todos os meus amigos, referidos ou não, por todas as experiências, pequenas ou grandes, que partilhamos e fizeram de nós quem somos.

João Carlos Mateus da Silva Martins

Acknowledgments

This work represents the conclusion of five years not only of study and hard work but also of personal growth. Because life is made not only of classes, these years of moments, experiences and even failures (yes, because life is not made only of victories) helped me grow and become who I am today, and for that I am grateful to everyone who accompanied me through this path.

To my family, first and foremost, and specially my parents I want to thank for everything you gave me during all my life. I will be forever grateful to you for the unconditional support in all of my decisions, the values and principles you passed onto me, the education and advice, the love and care that you give me every single day and so many other things. What you gave me made me the person I am today, they are the foundations of who I am. Without the support you gave me I would not have reached where I am, and for that I am extremely grateful.

To the advisers of this dissertation Professor Vítor Tavares and Professor Xin Li, for all the support given in the development of this project. I would like to thank Professor Xin Li for making it possible for me to develop my dissertation's project at Carnegie Mellon University and for all the advice, ideas and help given during that stay. The opportunity to work in a renowned American university allowed me to learn and experience new environments and for that I am also grateful. To Professor Vítor Tavares a thank you for all the help, suggestions and even reassurances that success was within our grasp when they were needed. Without your help, the project could not be finished. Finally a thank you to Minho Won for the help provided during my stay at CMU.

To all of the Professors that I had the privilege of knowing and learn from, for the contribution for my education as an Engineer.

Last, but not least, a few words for all the friends that shared these years with me. To Lena, Nuno, Diana and Rita, for after all these years of experiences, growing up and experiencing life, still being always present. To Fred, Joana, Hernâni, Manuel and company for all the late night talks, chaos and good times. To Bernardo, César and Francisco, forever the CMU Team, for all the stupid moments, brainstorming and traveling shenanigans. To Diogo, Vinícius, Pedro Amaral, Mário, Duarte and Carlos for all the help and all the relaxing moments. Finally and more importantly, to all my friends, referred or not, for all the experiences, small or big, that we shared and that made us who we are.

João Carlos Mateus da Silva Martins

*“Our greatest weakness lies in giving up.
The most certain way to succeed is always to try just one more time.”*

Thomas A. Edison

Contents

1	Introduction	1
1.1	Context and Motivations	1
1.2	Objectives and Challenges	2
1.3	Organization	3
2	Compressive Sensing	5
2.1	Compressive Sensing - A New Sampling Framework	5
2.1.1	Motivation	5
2.1.2	Concepts	6
2.1.3	Challenges	10
2.2	Recovering Data from Incomplete Sampling - An Analysis of the State of the Art	10
2.2.1	Recovery Algorithms	10
2.2.2	Previous Implementations	13
2.3	Image - a 2D Acquisition and Compression Problem	18
2.3.1	Acquisition	19
2.3.2	Compression	20
3	The IHT Reconstruction Method	21
3.1	The Algorithm	21
3.1.1	General Theory	21
3.1.2	Alterations Performed for Hardware Implementation	23
3.2	Image Data - The Search for a Sparsity Domain	25
3.2.1	The Discrete Cosine Transform	25
3.2.2	The Haar Wavelet	26
3.3	MATLAB - A First Approach to the Recovery Problem	28
3.3.1	Design Specifications	28
3.3.2	Implementation	30
3.3.3	Results	31
4	Implementation - From Theory to Hardware	35
4.1	System Architecture	35
4.1.1	High Level Architecture	35
4.2	IHT on Hardware - A Recovery Co-Processor	36
4.2.1	The Transform	37
4.2.2	First Matrix-Vector Multiplication	41
4.2.3	Second Matrix-Vector Multiplication	43
4.2.4	Scaling the Data - The Iteration Step	45
4.2.5	Threshold - Finding the Right Coefficients	47

4.2.6	Master Control - Managing the Iterative Processor	52
4.3	The Top Level Module - Integrating the Iterative Processor	54
4.3.1	Top Level Control	54
4.3.2	Mixed-Mode Clock Manager	55
4.3.3	AXI Lite Slave	56
5	Test Methodology and Results	57
5.1	Testing Environment	57
5.1.1	High Level System	58
5.1.2	ZYNQ Board Setup	59
5.1.3	Controlling the IHT IP Core through Software	60
5.1.4	MATLAB - Generating Measurements and Aggregating Results	61
5.2	FPGA Implementation - Resource Usage	64
5.3	Results	65
5.3.1	Image Quality	65
5.3.2	Execution Time	69
5.3.3	Power Consumption	71
6	Conclusions	77
6.1	Future Work	78
A	Image Recovery Results	81
	References	87

List of Figures

2.1	CS Framework. Taken from [4]	7
2.2	OMP Algorithm. Taken from [9]	12
2.3	Ren, F., Dorrace, R., <i>et al.</i> architecture. Taken from [27]	16
2.4	Stanislaus, J.L.V.M. and Mohsenin, T. architecture. Taken from [28]	17
2.5	Rabah, H., Amira, A., <i>et al.</i> architecture. Taken from [30]	18
2.6	Image acquisition and compression system	19
3.1	DCT 64×64 dictionary	26
3.2	Man1024x1024: Original (1MP)	32
3.3	Man1024x1024: Recovered (1MP)	32
3.4	Lion2560x1600: Original (4MP)	33
3.5	Lion2560x1600: Recovered (4MP)	33
4.1	High Level Architecture	36
4.2	IHT Architecture Block Diagram	37
4.3	DCT Flow graph algorithm. Taken from [41].	39
4.4	IDCT Flow graph algorithm. Taken from [40].	39
4.5	Transpose design based on [42]	40
4.6	RTL Schematic of the First Matrix-Vector Multiplication Block	43
4.7	First Matrix-Vector Multiplication Control State Diagram	44
4.8	RTL Schematic of the Second Matrix-Vector Multiplication Block	45
4.9	RTL Schematic of the Step Computation Block	46
4.10	RTL Schematic of the Scaling Block	47
4.11	RTL Schematic of the Thresholding Block	48
4.12	RTL Schematic of the Sorting Element Block and of one of its Cells	49
4.13	Threshold RAM Architecture	50
4.14	Threshold Control State Diagram	51
4.15	Master Control State Diagram	53
5.1	High Level System Diagram	58
5.2	ZYNQ Board Design	59
5.3	PSNR Results for Different Superimpositions	67
5.4	SSIM Results for Different Superimpositions	67
5.5	Man1024x1024 (1MP): (a) Original; (b) Recovery Result from MATLAB; (c) Recovery Result from Hardware	67
5.6	Lion2560x1600 (4MP): (a) Original; (b) Recovery Result from MATLAB; (c) Recovery Result from Hardware	68

5.7	Man1024x1024 (1MP): (a) Reconstruction from Patching with no Superimposition; (b) Reconstruction from Patching with 2 Superimposition; (c) Reconstruction from Patching with 4 Superimposition	68
5.8	Lion2560x1600 (4MP): (a) Reconstruction from Patching with no Superimposition; (b) Reconstruction from Patching with 2 Superimposition; (c) Reconstruction from Patching with 4 Superimposition	68
5.9	Comparison between the expected and observed speedups as the clock frequency increases	72
5.10	Power Consumption Evolution over Frequency Increase	72
A.1	Lena512x512 (0.26MP) Hardware Results: (a) Original; (b) Superimposition of 0; (c) Superimposition of 1; (d) Superimposition of 2; (e) Superimposition of 3; (f) Superimposition of 4; (g) Superimposition of 5; (h) Superimposition of 6; (i) Superimposition of 7	82
A.2	Man1024x1024 (1MP) Hardware Results: (a) Original; (b) Superimposition of 0; (c) Superimposition of 1; (d) Superimposition of 2; (e) Superimposition of 3; (f) Superimposition of 4; (g) Superimposition of 5; (h) Superimposition of 6; (i) Superimposition of 7	83
A.3	Lion1920x1080 (2MP) Hardware Results: (a) Original; (b) Superimposition of 0; (c) Superimposition of 1; (d) Superimposition of 2; (e) Superimposition of 3; (f) Superimposition of 4; (g) Superimposition of 5; (h) Superimposition of 6; (i) Superimposition of 7	84
A.4	Lion2560x1600 (4MP) Hardware Results: (a) Original; (b) Superimposition of 0; (c) Superimposition of 1; (d) Superimposition of 2; (e) Superimposition of 3; (f) Superimposition of 4; (g) Superimposition of 5; (h) Superimposition of 6; (i) Superimposition of 7	85

List of Tables

3.1	MATLAB Recovery Results	33
4.1	Input order for the transposition multiplexers	42
5.1	Hardware Implementation Resource Usage : Entire system	64
5.2	Hardware Implementation Resource Usage : IHT Co-Processor	65
5.3	Block Breakdown of Resource Usage	66
5.4	Metrics Results of "Lena512x512"	69
5.5	Metrics Results of "Man1024x1024"	70
5.6	Metrics Results of "Lion1920x1080"	71
5.7	Metrics Results of "Lion2560x1600"	73
5.8	Recovery Timing Analysis (100MHz): Lena512x512 and Man1024x1024	74
5.9	Recovery Timing Analysis (100MHz): Lion1920x1080 and Lion2560x1600	74
5.10	Recovery Timing Analysis for Different Clock Frequencies: Lena512x512 and Man1024x1024	74
5.11	Recovery Timing Analysis for Different Clock Frequencies: Lion1920x1080 and Lion2560x1600	75
5.12	Power Consumption Analysis	75

Abbreviations and Symbols

2D	Two Dimensions
ADC	Analog-to-Digital Converter
ASIC	Application-Specific Integrated Circuit
BP	Basis Pursuit
CoSaMP	Compressive Sampling Matching Pursuit
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
CS	Compressive Sensing
DCT	Discrete Cosine Transform
DSP	Digital Signal Processor
FF	Flip Flop
FPGA	Field-Programmable Gate Array
GPU	Graphics Processing Unit
HDL	Hardware Description Language
IHT	Iterative Hard-Thresholding
IP	Intellectual Property
JPEG	Joint Photographic Experts Group
LUT	Look Up Table
MGS	Modified Graham-Schmidt
MMCM	Mixed-Mode Clock Manager
MP	Matching Pursuit
MP	Megapixel
MSE	Mean Square Error
OMP	Orthogonal Matching Pursuit
PC	Personal Computer
PL	Programmable Logic
PS	Processing System
PSNR	Peak Signal-to-Noise Ratio
RAM	Random-Access Memory
RIP	Restricted Isometry Property
RTL	Register Transfer Language
SE	Sorting Element
SNR	Signal to Noise Ratio
SoC	System on Chip
SSH	Secure Shell
SSIM	Structure Similarity
VLSI	Very-Large-Scale Integration
WT	Wavelet Transform

Chapter 1

Introduction

1.1 Context and Motivations

As the needs for higher performance signal acquisition and processing systems edge towards the limits of what traditional approaches are capable of accomplishing, in terms of maximum operating frequency, price or even limits on the physical implementation, new sensing and processing frameworks must be found. In this sense, compressive sensing offers a solution where sampling may be done at lower rates and implementations with less costs may be developed. Leveraging on the theory that most natural signals can be sparsely represented in a suitable basis, sensing systems can be developed to work at lower frequency than the Nyquist rate where traditionally this would be impossible, or implemented with a single sensing component where traditionally there would be the need to create impossibly costly systems with arrays of several high priced sensors.

Compressive sensing is implemented taking advantage of inner product operations between the target signal and a sensing matrix, effectively doing the sensing and compression steps of traditional systems in one phase. While the acquisition phase is in most ways less computationally intensive than the traditional approach, the recovery of the original signal encompasses an NP-hard optimization problem where from a priori knowledge of a built-in dictionary and measurements, one must find the original, much larger signal.

The application of the Nyquist rate concept to the creation of sampling systems introduced the digital era. This reliable and solid base for the transfer between the physical and the digital worlds led to a torrent of new digital information that had to be processed and stored. In this digital data are included image, video and biomedical images. While at the beginning the amount of pixels per image was small, as the technology progressed and the need for higher resolutions increased, the ability of acquiring larger images also evolved. Nowadays, most smartphones have cameras with at least a 5 megapixel (5 million pixels) sensor, medium grade professional cameras offer sensors in the 20 megapixel range and as digital video reaches the ultra high-definition 4K standard the pixel count per frame reaches the 8 megapixel range. This amount of information creates the need for not only more storage but also efficient ways of storing the data. Since any user will aim to acquire as many images as possible with the available storage, compression techniques are commonly used

to optimize the available space. By taking advantage of the sparsity of the image representation in a given transform domain, these techniques eliminate unnecessary information thus creating an approximation of the original that occupies less storage space. Present day systems achieve this in two steps, first acquiring the entire image, i.e. the *RAW* format, and only compressing it afterward, normally into the *JPEG* format. This process is not the most efficient since resources and power are used to acquire data that will be eliminated during the compression. Thus, an image system capable of acquiring only the information needed to represent the image in its compressed form, i.e. on the transform domain, would be more efficient both in resource and power usage.

While many CS recovery algorithms have been formulated over the years, their real implementation is still a bottleneck in the whole system. As recovery algorithms rely heavily on matrix operations, software implementations have been only partially successful in getting good results within an acceptable processing time. In a search for more control over the calculations, GPU implementations were made, getting better results but being bottlenecked by the hardware architecture.

The next logical step is to study implementations using custom architectures, designed for the purpose of recovering CS measurements. With this in mind, FPGA implementations offer the flexibility of testing several design choices, can take full advantage of any parallelization potential and can finally lead to application-specific integrated circuits (ASICs).

1.2 Objectives and Challenges

The recovery of compressively sensed data, for which several different algorithms have been developed, is an optimization problem that is mainly focused on operations involving matrices. This type of operations is extremely unoptimized on general purpose CPU's, resulting on sub-par software based implementations running on consumer-grade machines.

The next implementation support would be the usage of GPU's. While this type of hardware offers programmable hardware structures, the speed at which data can flow between the processing cores and the memory is bottlenecked by the manufacturer specifications, resulting in a less than ideal implementation where the processing phase is hampered by the data exchange with the memory.

Since an FPGA offers full control over the internal structures, an implementation using it would allow for better data transfer between memory and processing cores, full control over the architecture and pathing between critical blocks and finally the opportunity to take advantage of any parallelization that may be implemented.

There has not been a lot of work developed on the implementation of CS recovery systems using FPGAs. This is mainly because most recovery algorithms employ operations difficult to implement on a hardware level such as l_2 -norms, matrix-vector multiplications, complex number computations, and others. In order to achieve an efficient and similar system to that obtained on a GPU-level or even on a software based implementation the designer is faced with a problem several times more complex since a hardware-level design must be envisioned. Nevertheless, a

well developed and implemented hardware-level design will most of the times achieve results better or similar to other approaches while simultaneously offering other advantages such as lower power consumption, lower area or the possibility of being integrated on a larger system.

An image acquisition system based on CS can take advantage of this framework, resulting on the reduction of the number of sensors and lower volume of data. This can lead to the design of systems with higher resolution at lower costs, the development of faster bio-imaging systems as well as other applications. The main challenge is the recovery of the sensed data which is closely tied to the reliability, speed and energy consumption of the recovery systems developed. By developing a hardware level implementation of these recovery algorithms (admittedly the most challenging part of the system) the realization of a full compressive sensing image acquisition system is possible, allowing for the miniaturization of the components and easier access to the possibilities of this technique.

Having these points in mind, this thesis aims to implement on FPGA and analyze the performance of the compressive sensing reconstruction algorithm Iterative Hard Thresholding. This analysis will mainly focus on the power consumption, recovery times, accuracy, resource usage and global performance. The hardware realization will have the objective of recovering compressively sensed image information for compression purposes.

1.3 Organization

In addition to this introductory chapter, this document is composed by five more chapters.

Chapter 2 focuses on Compressive Sensing as a framework, providing the motivation for its creation and usage, some of the concepts involved and challenges that a new sampling framework entails. As CS is a field with several contributions from different authors, an analysis of the state of the art is made, as well as a small exploration of the challenges behind designing an image acquisition and processing system.

Having been explored the theoretical background as well as the present state of the art of CS, a better understanding of the target algorithm of this work is given in **Chapter 3**. By analyzing the theoretical work, a modification is proposed, and two possible sparsity transforms are considered having the target data type in mind. To validate the algorithm, a high level implementation of the algorithm using MATLAB is also presented and shown to yield good results.

Moving to **Chapter 4**, building on the theoretical foundations presented, the entire hardware implementation of the algorithm is shown at a high level of abstraction (through block diagrams) and explained block by block. Architecture decisions and specific hardware design considerations taken during the design, targeting efficiency and fast implementation, are all discussed in this chapter.

The testing methodology and results of the different realizations are presented in **Chapter 5**. Since the test of the hardware is made using a non-trivial system, an explanation of its components is made, before an analysis of the results is performed on the image quality level, execution time and power consumption.

Finally, on **Chapter 6** the conclusions and future work considerations are described.

Chapter 2

Compressive Sensing

2.1 Compressive Sensing - A New Sampling Framework

2.1.1 Motivation

Joseph Fourier, a French mathematician from the early 19th century, proposed that any arbitrary continuous signal could be completely described in frequency-domain as the sum of a series of sine and cosine functions. This resulted on the formulation of the Fourier series and Fourier transform. Later, the discrete time case was described, resulting on the discrete time Fourier series and discrete Fourier transform. Any continuous signal is approximated by a set of discrete samples taken at a given frequency, meaning that with a sufficiently fast sampling rate, any continuous signal could be exactly reconstructed from a set of equally spaced samples. If the sampling frequency is lower than the needed, aliasing can occur leading to a reconstructed signal different from the original.

The minimum rate at which this sampling had to be made to achieve perfect reconstruction was not defined until the proposition of Harry Nyquist's and Claude Shannon's sampling theorem [1]. The Nyquist-Shannon sampling theorem states that in order to avoid aliasing, the sampling rate, also known as the Nyquist rate, should be greater than twice the highest frequency component of the target signal – the Nyquist frequency. This result is the foundation of today's digital systems.

While this theoretical result is applicable to well bound, noiseless signals, natural signals most of the times present noise and unexpected higher frequency components that need to be accounted for when designing any system. When a system is designed with a lower target sampling frequency that does not take these higher frequency components into account, aliasing occurs. In order to compensate for this, most present day systems are preceded by an anti-aliasing component, which normally would be a low-pass filter, to ensure that there are only frequency components lower than the Nyquist frequency. To allow for a feasible filter though, most of the times the effective sampling rate is some orders of magnitude higher than the Nyquist rate putting even more pressure on the sampling components of the system.

Even with these considerations the Nyquist-Shannon sampling theorem has been successfully used over the years of digital signal processing. However, as the digital revolution evolves, the

search for higher signal resolution, processing rate and new data structures is stressing the capabilities of traditional systems. As the operating frequency requirements rise, development of analog-to-digital (ADC) components is becoming more complex and expensive. With higher sampling rates, sometimes reaching frequencies in the Gigahertz range, a torrent of data is generated and sent to digital signal processing (DSP) systems, that need to cope with it. Furthermore, requirements such as high signal to noise ratio (SNR), low energy consumption and high efficiency and low cost increase the difficulty in developing these systems.

As the challenges presented to traditional signal acquisition and processing systems become harder, new methods need to be envisioned and implemented to overcome the challenges posed by the huge amount of data required by modern demands. Compressive sensing (CS) is one alternative. Building upon the theory of sparse representations, this approach aims to offer new solutions to several problems that traditional systems struggle to solve.

2.1.2 Concepts

With the works of Donoho [2] and Candes, Romberg, and Tao [3] it was demonstrated that a sparse or compressible signal can be reconstructed from an incomplete number of samples as long as some constraints are met. This means that, as long as a given signal is sparse (meaning that most of the coefficients on a given projection are zero) or approximately sparse in a transform domain it can be sampled incompletely, or at lower than the Nyquist rate. By leveraging the fact that a signal is sparse, a CS system can be built with the objective of recovering the "information content" of the signal instead of its "frequency content" since as a sparse signal, it can be represented on the transform domain by way fewer coefficients than the number of samples needed by a Nyquist rate based implementation. This concept is further explained on this section.

From the incomplete sampling rises the reconstruction problem inherent to finding an original signal with length N from a set of $M \ll N$ measurements. This leads to an NP-Hard problem, a combinatorial search for the least number of coefficients on the transform domain that best approximate the original signal. The concept is better explained later on, as some mathematical background is needed, but this is one of the main challenges in implementing a CS system. Although several recovery algorithms have been proposed, solving this CS problem is still a very hot topic.

In figure 2.1 a diagram of the framework of CS is presented. Since this work is not focused purely on the theory but more on the hardware implementation of these frameworks, not all of the theoretical background is clarified here. To best comprehend it though, several concepts, some of which were already briefly referred, need to be explained. Since the most important ones, and fundamental to a sufficient understanding of the theory behind CS, are signal sparsity, incoherent sampling and signal reconstruction, those will be better explained in this section.

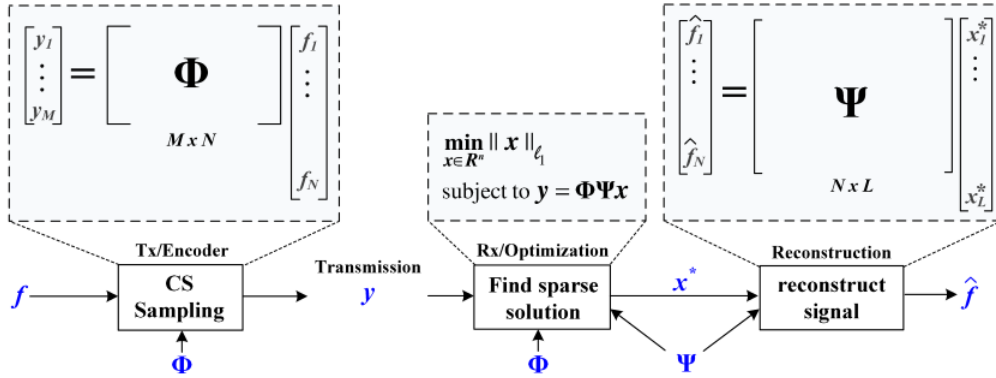


Figure 2.1: CS Framework. Taken from [4]

2.1.2.1 Signal Sparsity

One of the key assumptions behind the theoretical foundations of compressive sensing is that the target signal or class of signals are sparse in some basis (e.g. Fourier or Discrete Cosine Transform), $\Psi = [\psi_1 \psi_2 \dots \psi_n]$. Using this basis the signal can be represented as $f = \Psi \cdot x$ where x is the coefficient vector for f under the basis Ψ :

$$f = \sum_{i=1}^n \psi_i \cdot x_i \quad (2.1)$$

For a signal to be sparse in Ψ most of the coefficients x must be zero or very small such that they could be set to zero without any major loss. This would mean that in order to represent the signal y subject to a basis Ψ , k non-zero coefficients, where $k \ll n$, would be sufficient.

While a sparse representation implies that the signal is fully represented by the mentioned k non-zero coefficients where the other are really zero, a relaxation of this constraint leads to sparse approximations. A sparse approximation of a given signal means that the signal can be represented by k non-zero coefficients and the remaining, while not exactly zero, can be discarded without perceptible loss of signal quality. An example of signal sparsity would be a cosine or sine wave. When represented in the Fourier domain, these signals only need one coefficient to be fully recovered.

2.1.2.2 Incoherent Sampling

Data acquisition using compressive sensing is done through inner product operations between the target signal and a sensing matrix $\Phi \in \mathbb{R}^{m \times n}$:

$$y = \Phi \cdot f \quad (2.2)$$

The objective is to extract enough information with the measurements to recover the k largest coefficients in a given basis and reconstruct the signal. In [3] it was shown that a signal can indeed be recovered from a small number of measurements. Taking the samples from the discrete Fourier

transform domain, if a signal is k -sparse, meaning that there are at most k non-zero coefficients, it was proven that the signal could be exactly recovered by taking on the order of $m = k \cdot \log(N)$ measurements. Since the term k corresponds to the number of non-zero coefficients in the representation of the signal in a given basis, the number of measurements needed will be closely related with the amount of information contained in the signal, instead of the largest frequency component used in traditional sampling.

This first approximation on the number of samples was further refined to include the coherence between the sensing matrix (Φ) used on the acquisition of the signal, and the dictionary or signal model (Ψ) [5]:

$$\mu(\Phi, \Psi) = \sqrt{n} \cdot \max_{1 \leq k, j \leq N} |\langle \phi_k, \psi_j \rangle| \quad (2.3)$$

The coherence measures the correlation between any column of Ψ and any row of Φ and can be seen as an approximation of the degree of similarity between the sparsity and measurement systems. In order to be able to achieve a low minimum number of measurements, the rows of Φ must intersect as many dimensions of the basis space (Ψ) as possible. This will in turn allow for the discovery of the subset of dimensions where the signal lies with fewer attempts, or measurements. With this information, the lower bound estimative on the number of measurements needed from a K -sparse signal was shown to be:

$$M \geq C \cdot \mu^2(\Phi, \Psi) \cdot K \cdot \log(N) \quad (2.4)$$

where N is the dimensionality of the original signal and of the basis Ψ , and C is a scaling constant empirically found in [6] to be between 2 and 2.5.

Since the relation between the signal basis and the measurement matrix is so important in the decision of how many measurements are needed to achieve a perfect reconstruction, the construction of both has major repercussions on the global system. In order to be able to build a sensing hardware framework applicable in different signal situations, it is desirable that there is no need to know *a priori* the signal basis. This would allow for the design of sensing hardware that is capable of operating with different target signals. The solution is to use random matrices for the construction of Φ . As demonstrated in [7] random matrices, as long as having a sufficiently large dimensionality, show a low coherence with any fixed basis, meaning that a random sensing matrix could be used in any sensing system. Furthermore, the usage of random matrices as Φ would enable the sensing system to function with no knowledge of the signal basis used, provided that it is sparse in that basis.

2.1.2.3 Signal Reconstruction

Signal acquisition and processing systems can be divided in two main sub-systems: the acquisition and compression blocks, and the recovery part. In traditional sampling, the acquisition and compression portion are the most complex, comprising ADC's operating at the Nyquist rate, followed by a compression block that must sort the data and try to find the k largest coefficients

that represent the signal in a given basis. The recovery part is then simply focused on simple sinc-interpolation operations.

In compressive sensing there is a paradigm shift where the sensing blocks have low complexity but the recovery portion is more challenging. Instead of acquiring the entire signal and then compressing it, CS proposes that it should be possible to capture only the useful information from the target signal lowering the total number of samples needed. The sampling is then made from an N -dimensional signal f , into a M -dimensional set of measurements y , by means of inner product operations with a sensing matrix, Φ of size $M \times N$ as shown in (2.2). While M could take values greater than N , resulting in an overdetermined system, or equal to N , in which case the recovery would be trivial, the main focus of CS is for the cases where $M < N$.

For $M < N$, the number of measurements is less than the number of unknowns, resulting in an underdetermined system where the number of feasible solutions is infinite. While this is true, the fact that the signal is sparse can be used to find the correct solution. Since the correct solution is often the sparsest, the search should aim to find the solution with the least non-zero coefficients. Searching for the sparsest solution becomes then a combinatorial optimization problem:

$$\min_{x \in \mathbb{R}^N} \|x\|_{\ell_0} \quad \text{subject to} \quad y = \Phi \Psi \cdot x \quad (2.5)$$

where x is the coefficient vector from (2.1) and the ℓ_0 norm represents the number of non-zero elements of the vector, which means that the algorithm will search for the sparsest solution by trying to find the x with the least non-zero entries.

While (2.5) would lead to the solution with only $M = K + 1$, as a combinatorial search, this approach is not scalable and would be impracticable or even impossible to implement in real systems. In fact, the problem is NP-hard [8]. As a second alternative, some kind of approximation must be made or the ℓ_1 norm, where $\|x\|_{\ell_1} = \sum_{i=1}^N |x_i|$, can be used as a proxy to find the sparse solution as noted in [5]:

$$\min_{x \in \mathbb{R}^N} \|x\|_{\ell_1} \quad \text{subject to} \quad y = \Phi \Psi \cdot x \quad (2.6)$$

The substitution of the ℓ_0 norm for the ℓ_1 norm turns the problem into a convex optimization problem on which a vast and extensive work has been reported in different fields of science. While these are well defined problems, it is important to note that, as referred in [7], the exact recovery of a given signal from the set of incomplete measurements is always probabilistic. It is also possible for the sampling result to be a vector of zeros. From this measurement vector, no recovery algorithm would be capable of finding the original signal. While this may seem a major problem, as long as the sampling size is sufficiently large, the probability of these cases occurring falls so close to zero that they can be seen as negligible.

Several algorithms have been developed over the years with the objective of solving the compressive sensing signal recovery problem, but this work will mainly focus on the iterative hard thresholding (IHT) algorithm. This is the recovery algorithm chosen for the hardware realization given its highly regular iterative nature and good convergence guarantees, which will be presented in the next chapter. Furthermore, since it does not require extremely complex computation steps

such as the least-squares optimization needed for Orthogonal Matching Pursuit, this algorithm allows for simpler and more efficient hardware architectures.

2.1.3 Challenges

Having explored the motivation and theoretical foundations that make compressive sensing a novel signal acquisition and compression framework, the challenges on its implementation must be addressed. These can be divided into the two main parts of the system: the acquisition and the reconstruction.

The sampling of a signal using CS was already explained as the inner product operations between the target signal and a sampling or sensing matrix. While theoretically this may seem straightforward, on a hardware level the acquisition system can be fundamentally different from regular Nyquist-rate based sampling designs. Given this difference, the acquisition on itself is a challenge, either from the sensor layout standpoint or from the generation of the sampling matrices.

After solving the acquisition challenges, the recovery of the original signal must be done. As it was demonstrated, this is an extremely computationally intensive task and presently the most important point of research. Being implemented either in software or hardware, recovery algorithms struggle to give theoretical guarantees of convergence and a successful reconstruction. Furthermore, their computational requirements represent one of the biggest drawbacks in implementing this kind of system.

There are a lot of challenges that must be overcome to allow for a functional and efficient CS system to be implemented, but the possibilities it brings are the reason why research continues to be made in this field.

2.2 Recovering Data from Incomplete Sampling - An Analysis of the State of the Art

2.2.1 Recovery Algorithms

There are several classes of recovery algorithms, depending on the basis of their approach to the signal recovery problem. At least five of them can be seen as the major classes [9]:

1. **Greedy pursuit**

By employing an iterative approach, this class of algorithms tries to find the solution by making successive approximations to the expected answer.

2. **Convex relaxation**

Relax the combinatorial problem (finding the sparsest solution using ℓ_0) and replace it with a convex optimization problem (for example, by using the ℓ_1)

3. Bayesian framework

Relies on the development of estimators from the observed data to try and find the best solution.

4. Nonconvex optimization

Relax the ℓ_0 problem to a nonconvex problem and try to solve it by finding a stationary point.

5. Brute force

As the name suggests, this approach tries to find the solution by searching all the possible ones.

This work will focus on the implementation and characterization of a pursuit algorithm: the Iterative Hard Thresholding (IHT) algorithm. Given the iterative nature of greedy algorithms, these are the ones most studied for fast and reliable implementations. As such, other than basis pursuit (BP), which is a convex relaxation algorithm, the analysis over the present state of the art will mainly target greedy pursuit algorithms.

2.2.1.1 Matching Pursuit

One of the first reported recovery algorithms, the matching pursuit (MP) [10] takes as input the measurement vector, the dictionary (D) and a stopping criterion that can either be the number of iterations or a maximum accepted residual from the approximation. The algorithm then tries to iteratively find the column (one *atom*) of the dictionary with the most impact on the update of the approximation. Each iteration, based on the correlation with the residual, an atom from D is chosen to be added to a subset of the dictionary, after which the proposed approximation s_i is computed using the previous iteration result and the updated subset. As the iterative process continues, the residual will decrease and the subset of the dictionary is built. After the stopping criterion is met, an orthogonal projection of the approximation into the subset found is made resulting on the recovered vector.

The number of iterations, and by extension the number of columns of the subset and of non-zero coefficients on the final approximation, is dependent on the stop condition. This can be based on the sparsity, meaning that normally the algorithm would iterate over k cycles, or on the residual R_{n+1} norm computed from the difference between the previous R_n and the new approximation (R_1 is initialized at the y). Alternatively, the stop condition can be defined as the minimum change caused by the addition of a new coefficient. Before a new coefficient is added to the approximation its contribution to the residual is computed and if it gets reduced by an amount smaller than a given threshold, the algorithm stops.

2.2.1.2 Orthogonal Matching Pursuit

Building upon the MP algorithm, the orthogonal matching pursuit (OMP) ([11] [12]) adds an additional step to the iteration loop where the current approximation is always updated as the

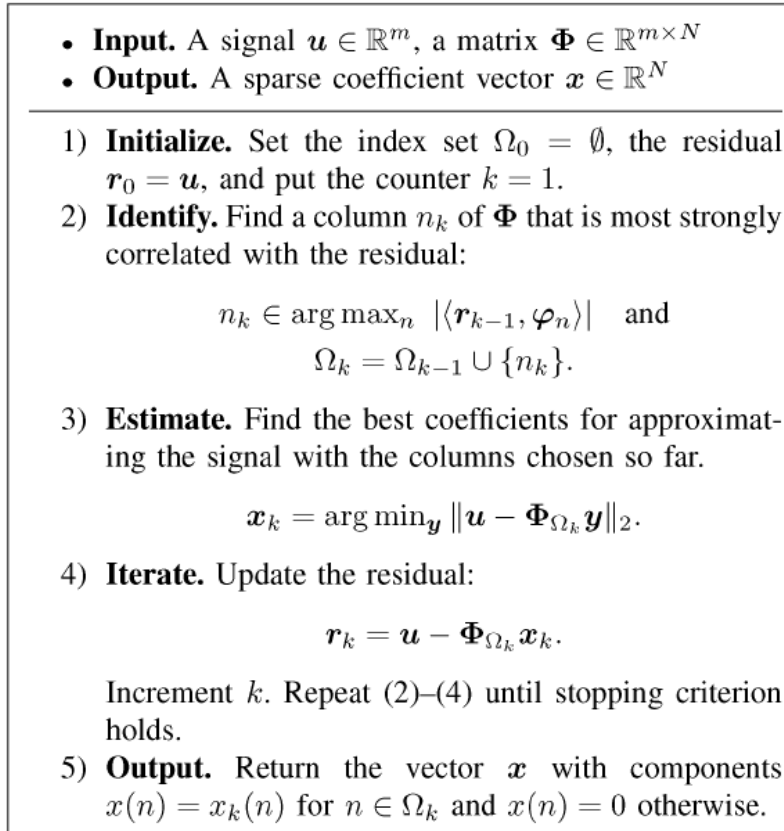


Figure 2.2: OMP Algorithm. Taken from [9]

orthogonal projection of the signal taking into account the *atoms* chosen so far. With this change the orthogonality of the residual is maintained at every iteration, improving the convergence and guaranteeing its occurrence in a finite number of iterations for a finite dictionary.

For the implementation of OMP for CS signal recovery the algorithm is presented in [9] as in the figure 2.2. The vector \mathbf{u} is the measurement vector and Φ is the sensing matrix. As with the MP, the stopping criteria can be, for example, one of the three previously mentioned.

The algorithm implementation has two main computationally demanding phases: the finding of the most correlated column with the residual, and the least squares optimization to achieve the orthogonal projection of the signal.

2.2.1.3 Compressive Sampling Matching Pursuit

While OMP guarantees the convergence, in some applications its performance is not adequate for the requirements, so other algorithms kept being researched and tested. The enhancements range from choosing multiple columns from the dictionary in each iteration to solving iteratively the least squares problem that arises from the orthogonal projection of the signal. One of such algorithms is the compressive sampling matching pursuit (CoSaMP) presented in [13]. In CoSaMP, while the framework is in everything equal to the one of OMP, a few key differences make it faster, more effective [9] and less complex [14]. Assuming a tuning parameter α and a sparsity k at

each iteration: $\alpha \cdot k$ atoms are chosen to add to the subset instead of only one, before updating the residual the updated approximation only retains the k largest coefficients.

2.2.1.4 Iterative Hard Thresholding

In iterative hard thresholding (IHT) [15] [16] instead of needing to solve a least square problem like OMP and CoSaMP, simply chooses the k largest coefficients and sets all of the other elements of the current approximation to zero. This small change allows for a compact representation of the core algorithm as

$$x_{i+1} = H_K(x_i + \mu \Phi^T(y - \Phi \cdot x_i)) \quad (2.7)$$

the term $(y - \Phi \cdot x_i)$ is the residual from the previous iteration and H_K is the operator that chooses the k largest coefficients. There is also a coefficient μ , the *step*, which is a constant responsible for scaling the weight of the residual in each iteration. While being quite simple, it was shown [16] that IHT can recover approximately sparse vectors with near optimal accuracy.

There have been several studies over the computation of μ , some of which reported in [17] [18] and [19]. All of these aim to find the best way to compute the μ based on the signal in question.

2.2.1.5 Basis Pursuit

Since until now all of the recovery algorithms presented are greedy algorithms, basis pursuit (BP) [20] [21], a convex optimization algorithm is presented. The basis pursuit algorithm tries to solve the recovery problem by relaxing the ℓ_0 norm ($\|\cdot\|_{\ell_0}$ which represents the number of non-zero coefficients) restriction and switching it for the ℓ_1 norm ($\|\cdot\|_{\ell_1}$ which is the normal sum of the absolute value of each coefficient). This change turns the problem from a combinatorial search into a convex optimization problem:

$$\min \|x\|_1 \quad \text{subject to} \quad \Phi \cdot x = y \quad (2.8)$$

Since convex optimization is a well documented problem with extensive research on efficient and fast ways of solving it, this formulation allows for the usage of those resources. For example, using BP, it would be possible to apply the simplex method, [20] to find the solution.

2.2.2 Previous Implementations

2.2.2.1 Software

In the literature, the implementation of the reconstruction algorithms in software is normally included as a simulation portion of a bigger publication so normally there is little information regarding the details of these implementations. Nevertheless, the results found are always invaluable for the construction of a solid understanding of the work previously done and published.

In [11] a comparison between the usage of MP and OMP for the approximation of a wavelet is made. While not directly applied to the recovery of a signal from a CS framework, the results show that as expected, the OMP is faster to find the coefficients for the representation of the wavelet. From the results it is visible that the MP takes close to ten times longer to reach the same level of error as the OMP.

As the CoSaMP and OMP algorithms share so many similarities, Gaur *et al* ([14]) compares them, as well as the least squares estimation, with different signal lengths and SNR. From the results it is possible to see that the CoSaMP shows very similar results to those of the OMP when seeded with the right sparsity value but its performance (the MSE) falls when the wrong value of sparsity is given to the algorithm. This is one of the major problems with this algorithm: since the sparsity value is needed to ensure a fast and accurate approximation, it could become impossible to implement in some systems.

Since the IHT algorithm has a *step* coefficient that can be tuned to achieve better results, in [17] Ollila *et al* propose some differences and present the simulation results. From the presented results it is visible that under Gaussian and Laplacian noise the performance and accuracy increase is small, while when in the presence of Student's *tv*-noise the modified algorithms present better mean square error rates at low number of degrees of freedom where traditional IHT has a very low performance.

In [19] a comparison between different hard thresholding variants is made by simulation using random sensing matrices Φ with independent and identically distributed normal entries. The columns of the sensing matrix were then normalized and the signal to noise ratio (SNR) and computation time for different values of $\frac{k}{M}$ taken. It was shown that accelerated IHT is indeed faster and has the same guarantees of the original IHT. The major problems come from the need to, in some cases, compute complex matrix operations which increases the computation time.

2.2.2.2 Hardware

Hardware implementations of CS recovery algorithms mainly focus on the OMP as it is relatively well documented and mathematically sound. There are two main implementation platforms, depending on the level of abstraction needed: graphics processing unit (GPU) or field programmable gate array (FPGA). When implementing the algorithm in GPU there is more control over the data flow and processing structures but since the platform is being repurposed, other bottlenecks appear. An implementation using an FPGA gives the developer total control over the architecture and data flow but adds design and implementation complexity since the abstraction level is so low.

In 2011, Fang, *et al.* proposed the usage of a GPU to accelerate the OMP algorithm for compressed sensing [22]. As they point out the bottleneck of the algorithm lies on matrix-vector multiplications used for the search of the best suited atom to add to the subset and the matrix inversion needed for the LS step. The architecture focuses on the parallelization and optimization of these. The implementation was done on a Nvidia GTX480 using Nvidia's Compute Unified Device Architecture (CUDA) which allows for the usage of GPU's for general purpose computing. For the matrix-vector multiplication a block approach is taken, using shared memory space to

transfer data between processing cores, while for the matrix inversion an update algorithm is devised and applied. The system was then tested, achieving a 40 times speed-up when compared to regular software implementations running on a CPU. As a final remark, it is stated that the final implementation uses too many processing cores (it is not optimized in terms of area and resources) and that the matrix-vector multiplication is the final bottleneck that has to be addressed by implementing a faster parallel algorithm for this phase.

Following the same idea of using a GPU, Sattigeri, *et al.* in [23] used a GTX460 to compare an OMP implementation on GPU with three others running on a CPU: normal OMP, OMP with Cholesky decomposition for the matrix inversion and Batch-OMP [24]. A GPU is comprised of multiple processing cores grouped into blocks which share small fast memory banks, so to allow for more parallelization the data was separated into small patches later on vectorized to be used by these memory banks. Since the data transfer between the CPU and GPU is slow, the information was sent to the GPU's main memory in big chunks which were then separated into the patches and only then transferred into the fast memory banks. This creates an overhead, since the data has to be transferred between the CPU memory, the main GPU memory and the fast memory banks of the blocks. Because of this, the results showed that the GPU implementation was the faster only from a given number of patches. As the number of patches decreased, the overhead caused by the data transfer was offset by the faster memory access that the CPU implementations had, resulting in a viable implementation of the GPU-OMP only from approximately 10 000 patches.

In 2013, Blanchard and Tanner, presented in [25] implementations of various CS recovery algorithms in GPU: hard thresholding, iterative hard thresholding, normalized iterative hard thresholding, hard thresholding pursuit, and a two-stage thresholding algorithm based on compressive sampling matching pursuit and subspace pursuit. Having recognized the matrix-vector multiplication as a bottleneck, the authors propose a regular approach, a fast approach relying on the DCT and a third approach for sparse matrices. It is also stated that minimizing the number of blocks used produced the best results. The implementations were tested and it was demonstrated that the speed-up achieved could get up to 70 times faster than its Matlab implementation counterpart.

As was seen on the GPU implementations found, most of the problems arise from the need to use the architecture defined for the graphics card, which in turn lead to problems on the data transfer rate between the CPU and the main memory of the GPU and from this to the faster memory banks that each block of processors has. To solve this, an FPGA can be used since it allows the designer to be much more flexible when building the architecture of the system.

Bai, *et al.* propose an FPGA implementation of the OMP and approximate message passing algorithms in [26]. The matrix inversion is done using the QRD algorithm implemented by the modified Graham-Schmidt (MGS) algorithm since the authors state that it is best suited for hardware implementation. To deal with the matrix-vector multiplication the authors propose the instantiation of a large number of parallel multipliers to build a vector multiplication unit. These units were then shared for different computations and to ensure that a large amount of data could be supplied to the unit in parallel, the memory was partitioned into several RAM blocks. The vector multiplication unit was designed with different operation modes depending on the algorithm

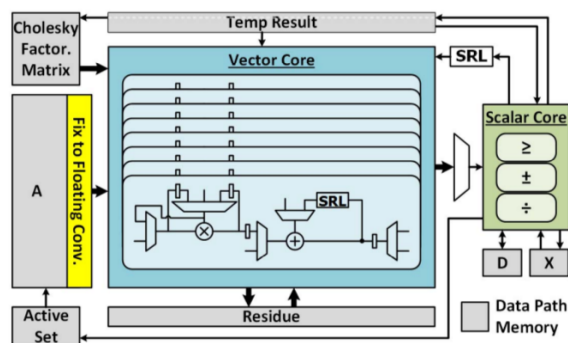


Figure 2.3: Ren, F., Dorrace, R., *et al.* architecture. Taken from [27]

being implemented, which in turn also allowed for this unit to be used on the same algorithm with different objectives, e.g., on the OMP implementation the unit has a matrix-vector mode (the regular multiplication), a scalar multiplication mode, and others. Since the speed increase on FPGA is closely related with the usage of fixed-point operations, the authors used this kind of operation throughout the implementation. To test the architecture, the design was implemented on a Xilinx Virtex-6 XC6VLX240T, the matrix size set to 256×1024 and the vector multiplication unit built with 256 parallel multipliers. When comparing with Matlab implementations running on a 2.6GHz dual-core CPU the authors refer a speed-up on the order of the thousands (4000 to 5000 faster).

Ren, Dorrace, *et al.* present in [27] a single-precision, floating-point OMP implementation for compressed sensing running on FPGA. To achieve a high degree of parallelization, the authors proposed an architecture, showed in figure 2.3, based on configurable processing elements that were instantiated in parallel to form the vector core. The proposed architecture relies on the modified Cholesky decomposition to achieve the matrix inversion of the least squares step and presents configurable datapaths and block operations based on microcode. The final implementation was synthesized to run on a Kintex-7 XC7K325T-FBG900 FPGA, used 128 processing elements and ran at 53.7MHz, accepting problem sizes up to $n \leq 1740$ and $m \leq 640$. To evaluate the accuracy and speed of the system, it was tested with electrocardiogram signals and found to be capable of achieving a $41 \times$ speed-up when compared with its CPU counterpart.

Stanislaus and Mohsenin presented in [29] the architecture for an MP recovery implementation using QR-decomposition for the matrix inversion on the least squares step. Although the authors refer to the implementation as being for OMP, upon close analysis of the literature and the architecture presented in figure 2.4 it is visible that indeed the algorithm implemented is MP. Nevertheless, the results and design insights are important. The optimization problem was designed for $N = 256$, $M = 64$ and sparsity $k = 8$ and the multiplication is done using 64 parallel multiplications followed by additions in a staged way. The authors refer in [29] that the QR-decomposition was chosen because Cholesky decomposition becomes more costly as the size of the problem increases. A fast QR-decomposition algorithm was proposed resulting on the implementation on a Xilinx Virtex 5 of the proposed architecture. This implementation yielded a reconstruction time

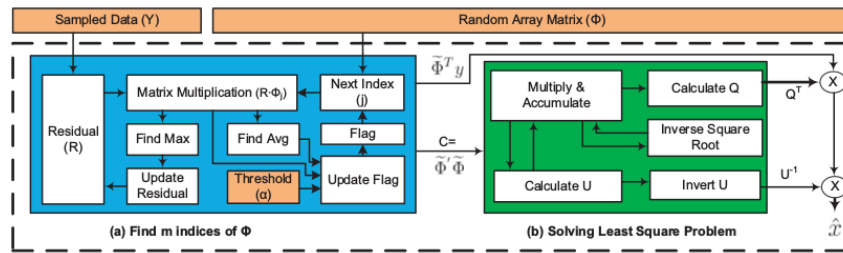


Figure 2.4: Stanislaus, J.L.V.M. and Mohsenin, T. architecture. Taken from [28]

of $27.12\mu s$ for a 256-length signal with sparsity 8 and the authors stated that the vector-matrix multiplication was the cause of the delay. The same authors later published another work, [28], where they tried to devise a way to expedite the vector-matrix multiplication. This was achieved by removing from Φ the columns where the product with the residue is lower than a threshold. Since this is effectively lowering the number of columns that can possibly be added to the subset, there is a trade-off on the quality of the reconstruction. The results presented achieve a reconstruction time of $17\mu s$ for the same circumstances from [29] and the same platform.

Rabah, Amira, *et al.* present in [30] a study of the complexity of the OMP implementation and try to achieve the highest throughput possible with the less area used by proposing a new architecture that used modified Cholesky decomposition for the matrix inversion of the least squares step. The system was designed for $N = 1024$, $M = 256$ and sparsity $k = 36$. The proposed architecture, in figure 2.5, was composed by 4 main blocks: an inner product and comparator unit (the K-IPCU), the Cholesky inversion unit, a residual computation unit and a reconstructed signal computation unit. The K-IPCU was composed of four 64-point inner products, a comparator, a dispatcher, and an adder tree. To implement the proposed architecture, the authors used the Simulink tool from MATLAB along with the Xilinx system generator (XSG) and Xilinx LogiCore for the system-level modelling. With those tools an hardware description language (HDL) implementation was produced and the architecture tested on a Xilinx Virtex-6 FPGA. The results, according to the authors, were 15dB better than other implementations and while using 328 more DSP48 involves 25 802 less slices, it is 1.85 times faster to compute the signal reconstruction. One last result was the development of a methodology for the optimization of area and execution time.

Finally, Xu, Jingwei, *et al.* present in [31] a very-large-scale integration design to recover data obtained by compressively sensing a one dimensional signal. A modified version of IHT is proposed in order to accommodate varying sparsity levels. This modification relies on a coefficient for the computation of the threshold value that allows for the varying number of non-zero coefficients but since the optimal value will depend on the sparsity of the signal, some *a priori* knowledge may be needed during the design phase. Using a 7 bits fixed-point representation, the proposed architecture aimed to recover a 256 point signal with an occupancy of 4%, meaning that the sparsity is close to 10, from sampling at 29% of the Nyquist rate of the target signal. The chosen transforms were the fast Fourier transform and the inverse fast Fourier transform and a binary ± 1 sampling matrix was used. The authors present a design which implements two clock domains

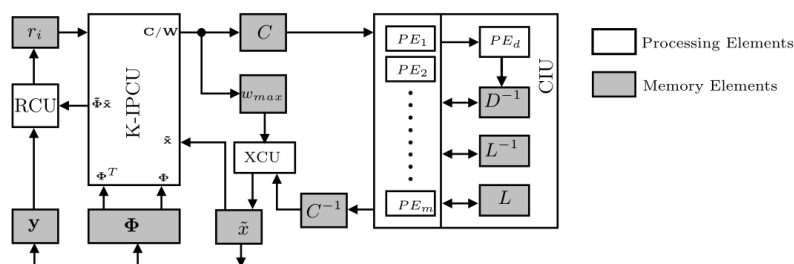


Figure 2.5: Rabah, H., Amira, A., *et al.* architecture. Taken from [30]

to achieve better power efficiency but while the slower clock domain may lead to a small power improvement, it entails the usage of another complete inverse transform block and other structures that lead to a less resource friendly implementation. In order to test the proposed architecture, a Verilog implementation was done and synthesized with the *TSMC 45nm* technology standard cell library using *Synopsys Design Compiler*. The results presented claim that this implementation was capable of operating at $88MHz$ with a power consumption of $165mW$ and execute one recovery in 22 iterations. This implementation results relate to a one dimensional signals with very low occupancy (only 10 coefficients for a total of 256 elements of the original signal) and a number representation with a low amount of bits. Furthermore, as the document does not mention the fabrication of the chip, the results were obtain purely from simulation and using an ASIC design instead of the proposed using FPGA.

Having explored the state of the art in the field of CS recovery algorithms and their implementations it is important to refer the objectives of the realization proposed in this document. Most of the recovery algorithm implementations found, either did not mention or used a one dimensional transform, hence enabling a simpler design. The target data for recovery in this work will be compressively sensed images, so two dimensional transforms, as well as some novel design structures will be included on the proposed design. Furthermore, the usage of an FPGA, while not as power efficient as an ASIC design, is expected to show improvements over the power consumption observed in GPUs as well as show major speedups in recovery run time when compared to software realizations. Thus, this work aims to present an efficient and fast hardware implementation of the IHT algorithm, using an FPGA as a development platform for the recovery of compressively sensed image information. Ultimately, the obtained architecture may become the foundation for an ASIC design.

2.3 Image - a 2D Acquisition and Compression Problem

From magnetic resonance imaging machines to cell phone cameras, current imaging systems are extremely varied in size, object of interest and price but all share the common objective of capturing one or more images with the best fidelity possible. Since the total amount of data generated from direct digitization is most of the times excessive, almost all of the traditional systems can be divided into two sub-systems or stages: acquisition (first row of figure 2.6), where the full signal

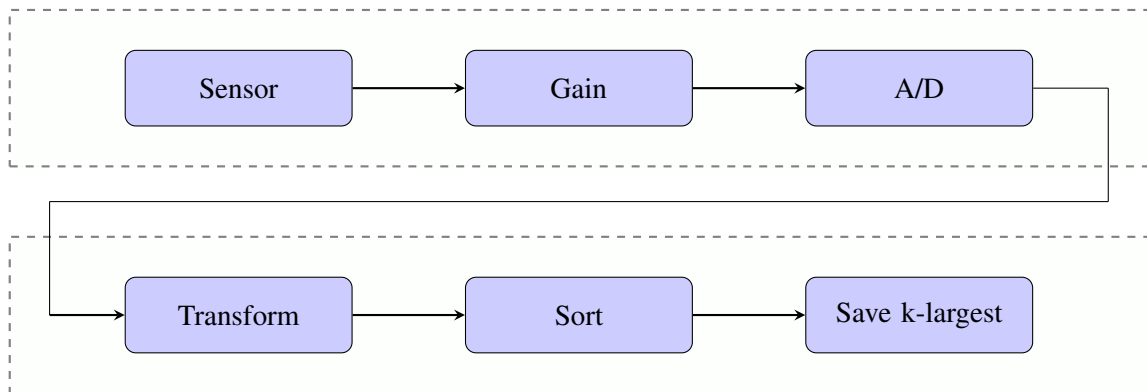


Figure 2.6: Image acquisition and compression system

is obtained, and compression (second row of figure 2.6), where the signal redundancy is ideally removed and further non-essential coefficients are removed to lower the total space or bandwidth occupied before the storing or transmission of the image.

A digital image is represented as a 2D matrix where each position corresponds to a pixel. While when capturing and processing color images one has to use at least 3 values per pixel, the most common color representation being RGB (red, green and blue), for the sake of simplicity this work will only address grayscale pictures where only one coefficient is stored per pixel. It follows that while there may be some added complexity when dealing with 3 coefficients instead of 1, the main explanations are still applicable.

Image information is sparse in various transform domains and as such compressible, being example of this the widely used JPEG compression. This allows for the design of image based compressive sensing systems and their application can lead to higher resolutions, lower sensor costs or lower power consumption systems. It is then necessary to first comprehend the processes behind traditional image acquisition and compression systems so that a compressive sensing system can be built.

2.3.1 Acquisition

The image acquisition portion of a traditional system can be divided into 3 main parts: the sensor, a gain step, and the analog-to-digital converters.

The sensor is the component responsible for translating light (or any other kind of radiation of interest) into electrical signals. In the case of a digital camera, this can be a CCD (charge-coupled device) or a CMOS (complementary metal-oxide semiconductor) sensor, and while they have different characteristics, their usage is very similar. Either sensor type is comprised of a matrix of cells, each one responsible for translating the incoming radiation into electrical signals. The effectiveness of this operation is closely linked with the amount of light, size of the cell and proximity between cells. As the number of cells per size increases, the amount of light captured by each one of them decreases, which results in increased difficulties in amplifying the signal and good

conversion performance. Since a compressive sensing implementation relies on the sum of intensities from several positions (pixels) instead of capturing every single one of them individually, the amount of light per measurement would always be higher than the obtained traditionally.

Before the analog-to-digital conversion, each signal is usually boosted by an amplifier to achieve better overall exposure [32]. While this aims to prepare the signal for a better conversion, it also amplifies any noise from the light to voltage translation made by the image sensor.

Finally, each pixel voltage is digitized and quantized. The architecture used for the analog-to-digital conversion may vary from chip to chip, since it is possible to use one ADC per pixel, operating at a low frequency, one ADC for all the conversions, operating at a very high frequency, or one ADC per column also known as column-parallel [33].

The final result of this phase is a digital representation of the image, with all of the captured information, also referred to as a raw image. At each step of the conversion noise sources add small changes to the original signal be it by Gaussian noise, shot noise, amplifier noise, and others ([32]).

2.3.2 Compression

Since the amount of data produced by the sampling phase is extremely high, some way of reducing that must be found to accommodate for the processing and storage of all of the images taken. To achieve this compression, the most usual approach is to find a basis in which the number of non-zero coefficient needed to represent the signal without perceptible loss of quality is small. Following this line of thought, two transforms that try to accomplish sparse representations of the original image will be presented: the discrete cosine and the wavelet transforms. To further lower the number of non-zeros it is also possible to apply a thresholding technique that sets all coefficients lower than a given value to zero, resulting in an approximation with losses.

Chapter 3

The IHT Reconstruction Method

3.1 The Algorithm

Compressive sensing has been a hot topic for several years and various recovery algorithms have been proposed, some of which were presented in 2.2.1. This research led to the development of algorithms that while giving solid guarantees of convergence and precision, are also extremely complex to implement on a hardware level. Since the objective of this work is to develop a recovery algorithm hardware realization, the complexity of its implementation on an RTL level had to be taken into account. As stated earlier, the iterative hard thresholding algorithm was the one chosen. It is an iterative algorithm with a regular dataflow and that offers good convergence and precision, if at the expense of more iterations. Furthermore, while still maintaining a degree of complexity it does not entail the inversion of matrices, least-squares steps or other highly complex computational operations.

3.1.1 General Theory

Iterative Hard Thresholding (IHT) is one of the best known and most used compressive sensing reconstruction algorithms. It is a greedy algorithm that iteratively tries to find the non-zero coefficients that best characterize the target signal in the sparsity domain chosen for the recovery. Although it has been briefly mentioned in a previous chapter, 2.2.1.4, since this algorithm is the one in which this work focuses, a more in-depth analysis of its operation and idiosyncrasies is needed and tackled in this section of the chapter.

$$x_{i+1} = H_K(x_i + \mu \Psi^T \Phi^T (y - \Phi \cdot \Psi \cdot x_i)) \quad (3.1)$$

At its core, the algorithm can be represented by a single equation (equation 3.1). From the analysis of the expression, it is trivial to decompose it into several steps:

1. The current recovered proposal is converted into the transform domain, Ψ , where the signal is sparse;

2. A matrix-vector multiplication is done between the transform domain signal and the Φ matrix;
3. The difference between the received measurements and the multiplication result is multiplied by the transposed Φ matrix;
4. The multiplication result is converted from the transform domain, Ψ^T ;
5. The new recovered proposal is computed from the sum of the current recovered proposal and the inverse transform result scaled by a μ factor.

While the algorithm is straightforward and simple to comprehend, the mathematical foundations and proofs, as well as requirements on the data that guarantee its success, are extensive. A lot of the work done over this algorithm has been presented by Blumensath, T *et. al* to first give the theoretical proofs needed to ensure that the algorithm leads to a successful recovery as well as under what circumstances this occurs.

In one of the first publications on the subject [15], Blumensath, T *et. al* present the usage of iterative thresholding applied to sparse approximations. This publication focuses heavily on the mathematical proofs needed to validate the algorithm, but since that is not the main objective of this work and since the concepts are non-trivial, only the main point is referred and it is that the convergence of the algorithm can be tied to the ℓ_2 -norm of the sensing matrix as $\|\Phi\|_2 \leq 1$.

In [16] the application of the iterative thresholding algorithm is studied on the context of compressive sensing. The concept of restricted isometry property (RIP), which was introduced by Candes, E J *et. al* in [3], is applied to analyze the sampling matrix Φ . The matrix will satisfy the restricted isometry property if it verifies that:

$$(1 - \delta_s) \cdot \|x\|_2^2 \leq \|\Phi \cdot x\|_2^2 \leq (1 + \delta_s) \cdot \|x\|_2^2 \quad (3.2)$$

for all s -sparse x and a $\delta_s < 1$. The coefficient δ_s is the *restricted isometry constant* and is defined as the smallest constant that holds the property previously presented. The mathematical proofs used to find a bound for the expected error, optimize the number of iterations needed, guarantee the convergence and other algorithm properties, rely heavily on the analysis and usage of this property. From using some of the properties of the RIP, the authors conclude that the algorithm is guaranteed to converge if $\|\Phi\|_2 \leq 1$ and that in this case the step size μ can be set to 1.

In [18] the case where the requirement $\|\Phi\|_2 \leq 1$ is not verified is addressed by the proposal of an alteration to the original algorithm. This alteration focused on the computation of a step size μ that leads to a successful recovery even when the matrix Φ does not have a norm smaller than 1. To achieve this, a new step size is computed every iteration by first

- Computing $g = \Phi^T \cdot (y - \Phi x)$;
- Computing the support of x^n (the proposed recovery from the n th iteration), Γ^n ;

- Finding g_{Γ^n} by discarding all elements of g except the ones in Γ^n and Φ_{Γ^n} by similarly discarding columns of Φ ;

and then computing the new step size using the equation

$$\mu = \frac{g_{\Gamma^n}^T \cdot g_{\Gamma^n}}{g_{\Gamma^n}^T \cdot \Phi_{\Gamma^n}^T \cdot \Phi_{\Gamma^n} \cdot g_{\Gamma^n}} \quad (3.3)$$

Together with the computation of μ in every iteration, the authors also try to find a bound for this value. First, it is defined that Φ is a matrix such that $0 < \alpha_{2K} \leq \frac{\|\Phi \cdot x\|_2}{\|x\|_2} \leq \beta_{2K}$ holds for all x with equal or less than $2K$ nonzero elements. Since the previously introduced g_{Γ^n} only has K nonzero elements, this can be used to define the bound presented on 3.4.

$$\frac{1}{\beta_{2K}^2} \leq \frac{1}{\beta_K^2} \leq \frac{g_{\Gamma^n}^T \cdot g_{\Gamma^n}}{g_{\Gamma^n}^T \cdot \Phi_{\Gamma^n}^T \cdot \Phi_{\Gamma^n} \cdot g_{\Gamma^n}} \leq \frac{1}{\alpha_K^2} \leq \frac{1}{\alpha_{2K}^2} \quad (3.4)$$

While this implementation is capable of guaranteeing the convergence in a smaller number of iterations, as well as removing the need for the sampling matrix Φ to verify the ℓ_2 -norm restriction formerly presented, the computation of the new step size in every iteration brings more complexity to the system as a whole. Because of this, implementing this version of the IHT is computationally more expensive and resource intensive.

3.1.2 Alterations Performed for Hardware Implementation

3.1.2.1 Modifying the Scaling Step

As was previously discussed, the convergence of the iterative hard thresholding algorithm is closely related to the RIP property of the sampling matrix Φ . Furthermore, it can be proven that if this matrix verifies the property $\|\Phi\|_2 \leq 1$ the algorithm is guaranteed to converge. There are cases though where this is not possible to achieve and in those cases, a scaling operation, or step, can be applied every iteration to ensure convergence.

The system implemented for this work uses random binary sampling matrices which can take the value of $+1/-1$. So, it is easily derived that the requirement stating that the convergence is guaranteed if the ℓ_2 -norm of the sampling matrix Φ is equal or less than 1 will not be verified. The other way of having a high probability of convergence is then to dynamically compute the step size based on the Φ matrix input. Using the algorithm modification presented in [18] would solve this problem, but the increase in complexity would lead to a more intricate and resource intensive design when developing the hardware implementation. For this reason, an alternative step size computation method was needed.

From [18] the computed step size is found to be bound by the both α_{2K} and β_{2K} , which in turn are bounded by the expression $0 < \alpha_{2K} \leq \frac{\|\Phi \cdot x\|_2}{\|x\|_2} \leq \beta_{2K}$. By computing $\alpha_{2K} = \frac{\|\Phi \cdot x\|_2}{\|x\|_2}$ an approximation of the upper bound of the step can be found and used as a step size, $\mu = \frac{1}{\alpha_{2K}^2}$ which should lead to convergence with a high probability, if not in the optimal number of iterations.

While this first approximation allows for a simplification of the algorithm, it still implies, if the proposed recovery was used as x after the first iteration, the computation of a new step size each iteration. To try and remove this need, a step size computation which only requires the sensing matrix Φ was developed. The proposed method to find this step is then the search for the maximum, absolute, weight of either the rows of the matrix, leading to equation 3.5, or the columns of the matrix, leading to equation 3.6. On the first case, the sum of each row is computed and the maximum absolute value between the results is chosen to be the coefficient, c , used to compute the step size as $\mu = \frac{1}{c^2}$. If the columns are used, the same process is applied but the sum is made column by column.

$$c = \max \sum_{j=0}^{N-1} \Phi_{i,j}, i = 0, \dots, M-1 \quad (3.5)$$

$$c = \max \sum_{i=0}^{M-1} \Phi_{i,j}, j = 0, \dots, N-1 \quad (3.6)$$

This new method of computing a step size shows, by the obtained recovery results later presented, that it can yield good convergence and ultimately a successful reconstruction. It is needed though, on a later work, to better explore and theoretically demonstrate the results observed.

3.1.2.2 Finding an Hardware Efficient Scaling Approach

With a method for finding the coefficient used for the computation of the step size, an hardware implementation must take into consideration some restrictions normally disregarded on software designs. A division operation is one of the most computationally expensive as in addition to using more resources than simpler computations, it introduces a high delay in the system, potentially leading to a very low maximum operating frequency. Because of this, alternatives must be found, when possible, to substitute divisions by other operations.

Having found a coefficient c from the previously explained computation over the sampling matrix Φ , the step size was defined as being $\mu = \frac{1}{c^2}$. From an hardware standpoint this division cannot be implemented easily, so to approximate it, a division by the closest power of 2 should be found. By substituting the coefficient c for an approximated power of 2, \tilde{c} , the division can be implemented as a right shift operation which in turn has a low computational cost, resulting on $\mu = \frac{1}{2^{\tilde{c}}}$.

With all of these considerations in mind, the hardware implementation must :

1. Compute the coefficient c from the result of the weight calculations over the matrix Φ ;
2. Using c , find the closest power of 2, $\tilde{c} = \lceil \log_2(c) \rceil$;
3. With \tilde{c} , implement the scaling with $\mu = \frac{1}{2^{\tilde{c}}}$ by using right shift operations.

3.2 Image Data - The Search for a Sparsity Domain

3.2.1 The Discrete Cosine Transform

The discrete cosine transform (DCT) uses a sum of cosine functions oscillating at different frequencies to decompose and represent the target signal. Building upon the theory of the Fourier transform where a signal is represented by a sum of sine and cosine signals, the DCT shares similarities with the discrete Fourier transform while using only real numbers. Furthermore a DCT of length N is equivalent to a DFT with length roughly equal to $2N$ over real and even symmetry data [34]. The formal definition of the DCT presents eight variants, each with small differences, but since for this work only two will have some importance those will be the ones addressed. They are, as presented in [34], the DCT-II

$$X_m = \left(\frac{2}{N}\right)^{1/2} k_m \sum_{n=0}^{N-1} x_n \cos \left[\frac{(2n+1)m\pi}{2N} \right] \quad m = 0, \dots, N-1 \quad (3.7)$$

and the DCT-III

$$X_m = \left(\frac{2}{N}\right)^{1/2} \sum_{n=0}^{N-1} k_n x_n \cos \left[\frac{(2m+1)n\pi}{2N} \right] \quad m = 0, \dots, N-1 \quad (3.8)$$

where

$$k_p = \begin{cases} \frac{1}{\sqrt{2}} & \text{when } p = 0 \text{ or } N \\ 1 & \text{when } p \neq 0 \text{ and } N \end{cases}$$

Being the most commonly used, the DCT-II is referred to as "the DCT" [34], while the DCT-III being its inverse (with some scaling if needed) is referred to as the inverse DCT. Together, these forms are used for various applications, one of which is image compression.

The usage of the DCT promotes the concentration of the signal information in the low-frequency coefficients, allowing for compression by discarding all but the k largest ones, resulting in a lossy data compression. In fact, the DCT-II is used for image compression in JPEG (Joint Photographic Experts Group) where after the application of the DCT to the image, a lossy compression is made where only a small percentage of the largest coefficients are stored, discarding all the remaining, smaller ones.

To use the DCT in image its two dimensional variant (2D-DCT) must be used. This is achieved by performing the transform over each one of the dimensions of the image, i.e. first the 1D-DCT is performed along each row, followed by the transform over each column or vice-versa. One way to achieve this is to build a dictionary, for example D , and do two matrix operations on the original A to find the transformed signal X .

$$X = D \cdot A \cdot D' \quad (3.9)$$

One example of a dictionary is the 64×64 dictionary shown in the figure 3.1. The components of a signal corresponding to lower frequency will be grouped on the superior left corner, while

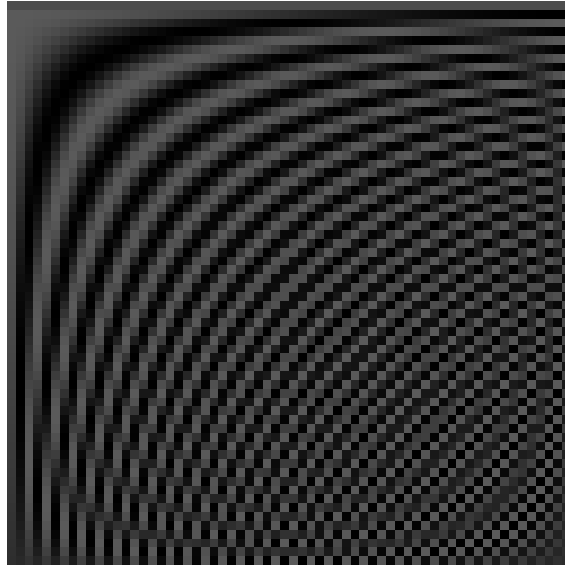


Figure 3.1: DCT 64×64 dictionary.

the frequency components will increase horizontally when moving to the left of the matrix or vertically when moving to the bottom.

3.2.2 The Haar Wavelet

One of the other possible transforms that can be used for signal compression is the Haar-wavelet transform. The Wavelet Transform (WT) is founded on the mathematical construction of wavelet series, short waves that have strict characteristics. Wavelets are a class of functions used to represent and localize a given function (or signal) in space and scaling. To construct a family of wavelets, one main function is used $\psi(x)$, called the *mother* wavelet, and a scaling function $\phi(x)$ which is called the *father* function. Using these two functions and by applying scaling and translating operations, *child* functions are created, and the subspace where the signal to be transformed is decomposed is created. The functions are given by the equations 3.10 and 3.11.

$$\psi_{m,n}(x) = 2^{-m/2} \psi(2^{-m}x - n) \quad (3.10)$$

$$\phi_{m,n}(x) = 2^{-m/2} \phi(2^{-m}x - n) \quad (3.11)$$

By employing the formula represented in the equation 3.12, where the *wavelet coefficient* is represented by $\langle g, \psi_{m,n} \rangle$ and corresponds to the inner product between the signal and a *child* function, any signal $g(x)$ can be reconstructed. While the mathematical proofs and requirements for a successful construction of wavelets are well documented ([35], [36]) they are not the focus of this work and would not be necessary.

$$g(x) = \sum_{m,n} \langle g, \psi_{m,n} \rangle \psi_{m,n}(x), \quad (3.12)$$

Similarly to the 2D-DCT, to apply the wavelet transform to two dimensional data, a separable approach can be used, resulting on the discrete wavelet transform over all rows first and then on all columns, or vice versa.

In image processing applications, the discrete wavelet transform (DWT) can be used to represent different scales of details, leading to a representation similar to the way images are perceived by humans: from a general representation first to the detail over time. Presently there is also an image compression standard based on wavelets, the JPEG2000. The standard allows for lossless (meaning that no coefficients are discarded) or lossy (some coefficients except for the k largest are discarded) compression. The wavelet transform is known for its better performance on the compression of transient signals, meaning that it will be better applied for higher frequency components compression, while the DCT is better suited for low frequency components.

Over the years, extensive research has been done on the field of wavelets, so naturally several families of these have been proposed some better suited to a given application, others simpler to implement. The family of wavelets considered for this work was the Haar-Wavelet.

The 1D discrete Haar Wavelet transform is simple to implement, being based on additions and subtractions. On the basis of the transform, a pair of elements is transformed by computing the mean and half of the difference between them. By expanding this algorithm to a larger vector with even number of elements, one can group pairs nested in pairs of elements, leading to the construction of a dictionary for the transform. For a signal with a length of 8, it will have 8 coefficients:

- the mean of all the elements;
- half of the difference, $\frac{a-b}{2}$, between the sum of the first 4, a , and the second 4, b ;
- half of the difference, $\frac{a-b}{2}$, between the sum of the first 2, a and the second 2, b ;
- half of the difference, $\frac{a-b}{2}$, between the sum of the third 2, a and the forth 2, b ;
- half of the difference between each pair, for the final 4 coefficients.

The resulting matrix that can be employed as a dictionary like in the case of the DCT, is shown next. This matrix can be used in a matrix vector multiplication, $G = H_8 \cdot g$ to compute the 1D Haar transform of a signal, g , with a length of 8.

$$H_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

The inverse transform is obtained by doing the inverse computations, in pairs like was done for the forward transform. It is important to note, however, that to implement the transform, a final normalization step must be taken which will imply the multiplication by some coefficients ($\sqrt{2}$ on the previous case).

3.3 MATLAB - A First Approach to the Recovery Problem

3.3.1 Design Specifications

To design the recovery system based on the Iterative Hard Thresholding algorithm, some specifications must first be defined. The type of data to be recovered, images, as well as the fact that the final objective is to design a hardware implementation of the algorithm should also have some weight on this first, software based implementation. With this in mind, first some considerations about the rationality behind the design, as well as the defined specifications, are given.

By present days standards, images have sizes on the order of the megapixel (MP) and even video has reached this pixel count with regular high definition (HD) video having resolutions of 1280x720 for HD video, 1920x1080 for Full-HD and 3840x2160 for Ultra-HD, leading to approximately 1, 2 and 8 megapixels, respectively. If the system was built to recover a whole image from a single set of measurements, the resource usage would be extremely high. Let us consider the case where the system was built to recover 1 megapixel images:

- N is equal to the size of the original signal so it is 1000000;
- Assuming that the image can be compressed with a ratio of 10 : 1, the sparsity would be $K = 100000$;
- Since to assure a good recovery, the number of measurements must be higher than the sparsity value, one could take two times more measurements than the sparsity, meaning that the length of y , $M = 200000$;
- The measurement matrix, Φ has the dimension $M \times N$, leading to $2 * 10^{11}$ positions and can be represented as a single bit per position;
- Considering a block of 1000x1000 pixels, the chosen 2D transform must take as input arrays of 1000 values corresponding to either rows or columns of the block;
- In both software or hardware implementation possibilities, the data must have a given width, meaning that the measurements, recovered signal and middle computation results must have a given dynamic range. Assuming 32-bits representations the proposed recovery will occupy $N * 32 = 32000000 \text{bits} \approx 4MB$, the measurements will occupy $M * 32 = 6400000 \text{bits} \approx 800KB$. The measurement matrix can be represented using 1 bit per position, so would occupy $2 * 10^{11} \text{bits} \approx 25GB$.

By taking into account only the 2 input arrays, Φ and y , and the output, \tilde{x} it is easy to conclude that a system like this could never be implemented in an efficient way since the resource usage would be too high. Because of this, a patch-by-patch approach must be taken in which the image is divided in patches, square for the simplicity of implementation, and the sampling and later on the recovery are made on a patch-by-patch basis. This means that after recovering each patch, they have to be aligned to recover the full image.

The patch size should be sufficiently large to achieve a good compression rate, meaning that the data it contains can be represented by a small enough number of transform domain coefficients when compared to the total length of the original data, as well as small enough so that the resource usage is sensible and allows for recovery cores parallelization.

With these considerations in mind, it was defined that each patch has a size of 8×8 , which leads to $N = 64$. A fixed sparsity value was set as $K = 8$ and the number of measurements needed to have a good probability of successful recovery as double the sparsity, meaning that $M = 16$. These specifications lead to an acceptable resource usage and, at $M/N = 16/64 = 0.25$, a good compression ratio.

Finally, the step size computation was defined as being found using the weight of the rows approach, since both approaches are similar and for a later hardware implementation it will lead to a more straightforward design.

3.3.1.1 Sampling and Sparsity

As was defined previously, the system will use a patch approach where the image is divided into smaller square portions of 8×8 pixels and both sampling and recovery are done over these. Since the algorithm works under the assumption of a two-dimensional sampling matrix, $\Phi^{M \times N}$, and a vector representing the original signal, the patch must be rasterized. In this process, the 8×8 patch f is reshaped into a vector of 64 positions f_v , where the first 8 positions correspond to the first row, the positions from 8 to 15 correspond to the second row, and so on.

The original vector is assumed to be sparse in a given transform domain Ψ . It can then be represented by the form $f_v = \Psi x_v$ where x_v is a vector of the coefficients that represent the signal on the transform domain, and will be the target of the recovery process. This stems from the principle that a compressive sensing recovery algorithm tries to recover the coefficients that represent a signal in the transform domain, and not to recover the signal on its original domain.

The sampling vector is created by the operation: $y = \Phi \cdot f_v = \Phi \cdot \Psi x_v$. Both Φ and y are then inputs of the recovery function that will try and recover the original vector's transform domain coefficients \tilde{x}_v . Finally, the vector form must be reshaped into its 2D corresponding form \tilde{x} and inversely transformed to find the recovered \tilde{f} .

3.3.1.2 Choosing the Sparsity Domain

In section 3.2 two possible transforms were presented that would allow for a sparse representation of 2D data. Both have proven results and would be possible to implement in a image data centered

system but, as was referred each has its advantages and disadvantages. To achieve good results, the choice between the two must take into account the expected data distribution and its impact on the transform domain representations.

The patches used will have a size of 8×8 and the target images will have considerable dimensions with sizes of at least 512×512 . As was discussed, the discrete cosine transform excels in representing lower frequency components of the data over which it operates, while the Haar wavelet, and wavelets in general for that matter, are better suited for representing higher frequency components. In visual terms, this equates to the DCT being better suited for smooth transitions in an image and the Haar wavelet better for high contrast and sharp tone differences. Given the fact that the target images will not be small, it is expected that sharp transitions in a patch of size 8×8 will be at least rare, and that the most frequent type of data will be mainly composed of lower frequency components.

A system focused on smaller images would probably take a better advantage of the usage of the Haar wavelet, while larger images will be better represented by a DCT transform. Because the objective is the design of a system capable of recovering big images on the order of the megapixel, the DCT was the chosen transform. Furthermore, it is expected that the results of the recovery improve as the size of the image increases, since each patch will represent a smaller percentage of the whole image and the transitions between tones will be longer, meaning that they span multiple pixels with less drastic changes between adjacent pixels.

3.3.2 Implementation

To assert the validity of the specifications defined so far, as well as gain a better understanding of the algorithm, a first software-based implementation was done using MATLAB. Since the program serves as the software approximation of the hardware design later on implemented, it tries to follow the same flow and data manipulation.

In order to test the recovery algorithm, samples had to be created from an existing image. As was referred, the designed system follows a patch approach where the image is divided into square portions of 8×8 pixels. The MATLAB implementation follows this by using a selected image, fetching a patch of the appropriate dimensions and then applying a random matrix Φ , binary with the values of either 1 or -1 , to simulate the sampling process. This matrix is created randomly but every time the main program runs, the same seed is provided to the random number generator resulting on the same "random" matrices every execution. This ensures that the results can be repeated.

A function was created to implement the core algorithm. It receives the sampling matrix, the measurement data, the sparsity and the number of iterations. As was defined previously, the sparsity was set as 8 and the number of iterations was set as equal to the number of measurements, 16. The 2D-DCT forward and inverse transforms of the MATLAB standard library were used, and since it was already known that the scaling step in hardware would be implemented solely as a right-shift operation, the function calculates the step size and scales the data the data based on a division by a power of 2.

Before sending Φ and the measurement vector $y = \Phi \cdot f_v$, a scaling operation is done on the measurements by multiplying them by 2^8 . This is done to ensure that there is no under or overflow during the iterations and is undone by dividing the function result by the same factor, 2^8 .

Having sent the inputs to the function, the result is the vector containing the K non-zero coefficients that represent the data on the transform domain. This vector is then reshaped into an 8×8 matrix and the recovered patch \tilde{f} is obtained from the 2D-IDCT of the coefficient matrix.

The recovered patch is then aligned in its corresponding position on the complete image and the process is repeated for the next patch until the whole image's patches are sampled and recovered.

While a straightforward division of the image results, in most cases, in a good enough recovery, stitching artifacts may occur. Stitching occurs when the frontiers of the patches are visible, meaning that the image will appear to have a grid of squares of size 8×8 . This effect is visibly less apparent as the size of the image increases since the size of the pixels, and subsequently of the patches, becomes smaller in comparison to the overall size, but another technique may be applied: superimposition of patches. When applying a superimposition technique, instead of fetching patches that do not overlap, meaning that if one patch goes from the column 0 through 7 the next one will go from 8 to 15 and so on, there is a superimposition of s pixels between the first and second patch. The superimposition can go from 1 to 7, meaning that on the first case, the second patch starts on column 7, effectively running the recovery over this column again, and on the other limit the number of columns reused is 7 meaning that there is only the addition of one new column. Furthermore, when changing row, the same applies, meaning that on the case of a superimposition of 7 at a certain point only 1 pixel changes between one patch and the next. The application of superimposition leads to two problems: the alignment of the recovered data and the overall increase on patch number. The first one is solved by adding the superimposed pixels and then applying a weight mask that saves the number of additions done in each pixel allowing for a final pixel by pixel division that results on the mean of each pixel value. The second problem is only solved by reducing the superimposition to the bare minimum, depending on the size of the image and the target application of the recovered data.

3.3.3 Results

After implementing the recovery framework in MATLAB, it had to be tested with real images. To test the system, images of different sizes were divided into patches, sampled, recovered and then reconstructed using the procedure previously explained. In addition to testing images of different sizes, the superimposition of the patches decomposition was also varied to evaluate the recovery in visual and mathematical results. Since only using visual analysis of the result is a subjective method, some evaluation metrics had to be implemented, namely Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity (SSIM) index. Considering an original image, f , and its recovery, \tilde{f} , with size $W \times H$ (width x height) the application of these metrics is done as follows.

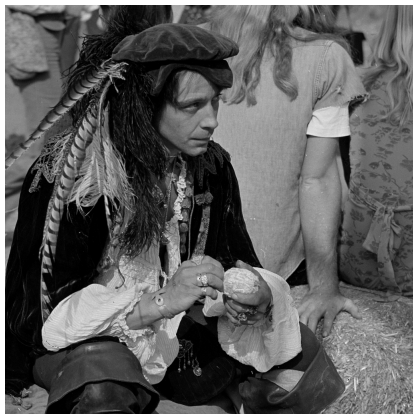


Figure 3.2: Man1024x1024: Original (1MP)



Figure 3.3: Man1024x1024: Recovered (1MP)

The PSNR is a measurement of the ratio between the maximum possible power of a signal and the power of the noise that is present. It can be computed using equation 3.13.

$$PSNR = 20 \log_{10} \left(\frac{\max(f)}{\sqrt{\frac{1}{W \cdot H} \cdot \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} \|f(i, j) - \tilde{f}(i, j)\|^2}} \right) \quad (3.13)$$

The SSIM index [37] is an alternative quality measurement that tries to approximate the evaluation to the way images are perceived by vision. This is achieved by comparing local patterns of pixels intensities. The measurement, ranging between 0 and 1 is obtained by computing the expression presented on equation 3.14. The variables c_1 and c_2 are dependent on the dynamic range of the pixel representation and serve to stabilize the division.

$$SSIM(f, \tilde{f}) = \frac{(2\mu_f \mu_{\tilde{f}} + c_1) \cdot (2\sigma_{f\tilde{f}} + c_2)}{(\mu_f^2 + \mu_{\tilde{f}}^2 + c_1) \cdot (\sigma_f^2 + \sigma_{\tilde{f}}^2 + c_2)} \quad (3.14)$$

Since this chapter is focused on the validation of the implementation and on a more in-depth analysis of the recovery, only a small amount of results are presented here. Later in chapter 5 a more extensive analysis will be performed, along with a comparison between the hardware and the present implementation. With that in mind, the recovery results from two images are shown in this subsection: one with a size of 1024x1024, approximately 1 megapixel (MP), and another with size 2560x1600, approximately 4MP. Both images were sampled using a superimposition of 7.

In figures 3.2 and 3.3 the original and recovered result of the 1MP image are shown. It is visible the loss in tone range and edge definition. Mainly the reasons relate to the facts that the DCT is better suited for smooth transitions and that the number of coefficients used is fixed. This number is closely related to the sparsity value fixed as 8 during the definition of the system since the threshold step maintains a number of non-zero coefficients equal to the sparsity of the signal.

In figures 3.4 and 3.5 the original and recovered result of the 4MP image are shown. While the loss of definition and dynamic range previously seen still occurs, it is less visible given the



Figure 3.4: Lion2560x1600: Original (4MP)



Figure 3.5: Lion2560x1600: Recovered (4MP)

increase of the image size in comparison to the size of the patch.

On the table 3.1 the measurement results are shown.

Table 3.1: MATLAB Recovery Results

Image	PSNR (dB)	SSIM
Man1024x1024	22,31	0,5515
Lion2560x1600	22,98	0,6557

With these preliminary results, the algorithm software implementation was shown to be capable of recovering images with the defined specifications so an hardware implementation can be developed.

Chapter 4

Implementation - From Theory to Hardware

4.1 System Architecture

Having described the theoretical foundations behind compressive sensing and fully explored the iterative hard thresholding recovery algorithm, a first software implementation of the IHT was realized using a high level language. With a good comprehension of the algorithm, the next step was the design and implementation of the hardware iterative processor that realizes the IHT algorithm using as a development platform a field-programmable gate array (FPGA). This chapter is focused on the entire hardware design process, presenting the development using a top to bottom approach. Hence, the system will be presented first at a high abstraction level, with a block diagram architecture, followed by an increasing detail for the main structures focusing on the architecture and design choices at a hardware register level.

4.1.1 High Level Architecture

The designed system was developed having in mind the specifications set discussed in section 3.3:

- Original signal length, after rasterization : $N = 64$;
- Number of measurements : $M = 16$;
- Fixed sparsity : $K = 8$.

In addition to these, a hardware implementation has to take into consideration other specifications related with the data representation, width and resource usage. By equating the repercussions of these specifications, it was defined that:

- The data inside the recovery core processor has a width of 32 bits. This decision, while leading to a larger resource usage, was taken in order to prevent under or overflow during the intermediate computations. Since the data is not easily bound, this width reduces the likelihood of such cases to occur;

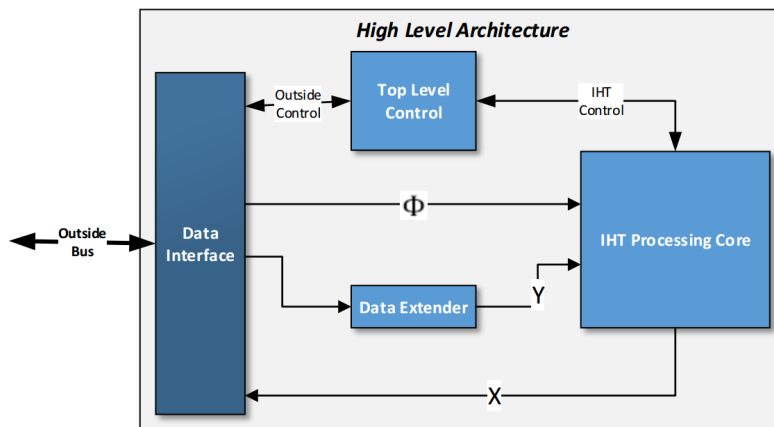


Figure 4.1: High Level Architecture

- A $Q16$ fixed-point representation is used, meaning that the data is represented with 16 fractional bits and 16 integer bits;
- The sampling matrix Φ is binary, with the value 0 mapped as a -1 and a 1 as 1 on the computations. It should be inputted as a vector of length $N * M$, which in the present case means that the vector has a length of 1024.
- The rasterization of the sampling matrix is done line by line, with the least significant bits of the vector corresponding to the first row of Φ and the least significant bit to $\Phi_{0,0}$
- The system has inputs with a width of 16 for each measurement. This allows for an extension of the signal before starting the recovery and preventing the occurrence of under or overflow;
- The measurement data is extended to the 32 bit fixed-point representation;
- The measurements are aligned in a way that leads the first measurement to be represented by the lower bits of the measurement vector, with its least significant bit being $Y_{0,0}$.

At a high abstraction level, the hardware system to be developed must have an interface that allows it to: receive measurement data and output the recovery results, receive control signals, and give information to the exterior of the current recovery status. Inside, a processing core responsible for the implementation of the IHT algorithm should be instantiated, as well as a control structure and some additional supplementary blocks (e.g. the block responsible for expanding the representation of measurements, from the initial 16 to the final 32 bits). Figure 4.1 shows the high level architecture.

4.2 IHT on Hardware - A Recovery Co-Processor

The main focus is now given to the processing core. Its function is to implement the iterative hard thresholding algorithm for the recovery of image-based measurements. The specifications defined

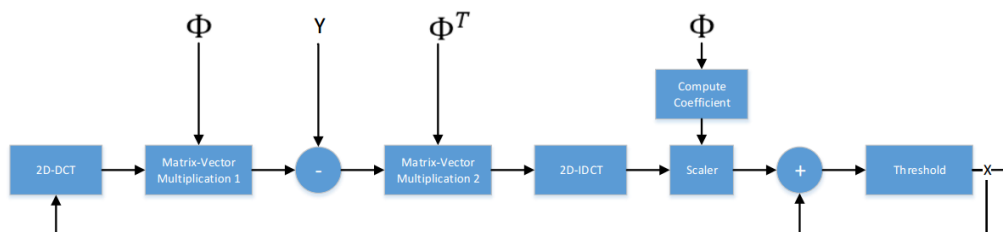


Figure 4.2: IHT Architecture Block Diagram

on the high-level software implementation in MATLAB presented in section 3.3 will also be used here. As referred earlier, the algorithm is represented by the function 4.1, which can be broken down into a sequence of operations, starting with a currently proposed recovery x_i :

1. A forward transform, in this case using the 2D-DCT;
2. A matrix-vector multiplication with the measurement matrix;
3. A subtraction with the measurements vector;
4. A matrix-vector multiplication with the transposed measurement matrix;
5. An inverse transform, in this case using the 2D-IDCT;
6. A scaling operation to promote the convergence of the algorithm;
7. A sum with the present proposed recovery x_i .

$$x_{i+1} = H_K(x_i + \mu \Psi^T \Phi^T (y - \Phi \cdot \Psi \cdot x_i)) \quad (4.1)$$

In addition to the main computation blocks, some control over the pipeline and the operation inside each block is needed and will be addressed accordingly.

This sequence of operations is presented as a block diagram in figure 4.2, to better understand the data flow. Several design options and choices were taken based on relationships between operations to improve the pipeline efficiency. A closer look at each block can give a better understanding of their operation. The iterative processor is composed by several complex sub-blocks. The following subsections will describe each of these blocks in detail.

4.2.1 The Transform

One of the most important decisions for the specification of the IHT algorithm processor was the definition of which transform should be used for the recovery process. Two alternatives were evaluated: the discrete cosine and the Haar wavelet transforms, having been chosen the 2D-DCT for implementation. As two of the most computationally intensive operations of the whole design – the forward and the inverse transforms – the performance, resource usage and accuracy of the resulting design was crucial to obtain good recovery results.

One of the possible approaches to the implementation of the 2D-DCT and 2D-IDCT is the row-column decomposition method. This is the adopted procedure, where the two dimensional transform is achieved by applying first a one dimensional transform to the rows of the matrix and then to the columns that result from the first transform, or vice-versa. In this work, the 2D-DCT is implemented by inputting the rows first, while the 2D-IDCT will operate over the columns first. This approach results in a dataflow consisting of two 1D transforms with a transposition between them. The application of this method implies then the development of:

- The 1D transforms, the 1D-DCT and 1D-IDCT;
- The transpose architecture used between the first and second transforms.

It is important to note that the output data from the second transform operation will be column-by-column in the case of inputting rows on the first transform, or row-by-row if the inputs are the columns. The data alignment can be solved by using another transposition block before the first transform, or at the end of the second transform, leading to a higher delay on the pipeline and resource usage. Another possibility is to simply take into account the alignment of the data, which is what was done in the case of this work.

4.2.1.1 The 1D-Transforms

Over the years, several algorithms for the application of the transform have been proposed and tested focusing mainly on the optimization of the number of operations needed to achieve the transform. One of such designs, and also one of the most efficient proposed, was the one presented by Loeffler *et. al* [38] in 1989, which by using only 11 multiplications and 29 additions, was proven to have achieved the theoretical bound for the lowest possible number of multiplications. Although this design may need the least number of multiplications possible to be implemented, a hardware realization may be slower because of the multiplication blocks. To achieve higher clock frequencies and lower hardware resource usage it was then necessary to develop multiplier-less designs of the forward and inverse transforms. These designs eliminated the usage of multiplication specific blocks by implementing the multiplication operations as combinations of sums and data shifts. Both Chen *et. al* [39] and Aakif *et. al*, [40] and [41], presented works focused on this objective. The work of Aakif *et. al* presents a comparison between their implementation and the one proposed by Chen *et. al*, showing better results. From this reason, the designed modules for this work were based on the architecture proposed by Aakif *et. al*.

Both the DCT and IDCT transform computations, as referred before, encompass the implementation of several multiplications, but these multiplications are in fact between a value dependent on the inputs, and a constant operand. It is the constants that allow a multiplier-less design of the transforms, since the operands can be implemented as hard coded blocks containing sums and shifts that correlate to that multiplication.

Another important fact is that there is a duality between the forward and inverse transforms, where both use the same constant operands but in symmetric locations on the corresponding flow

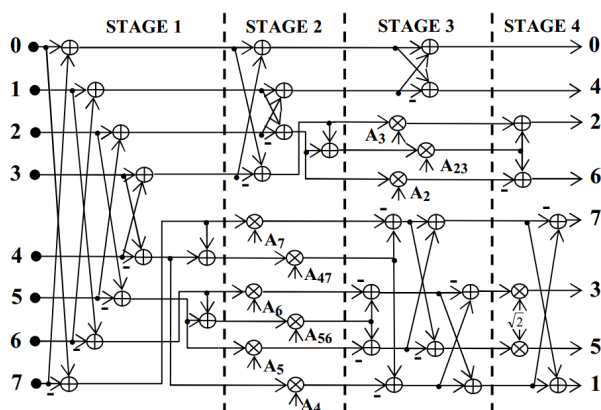


Figure 4.3: DCT Flow graph algorithm. Taken from [41].

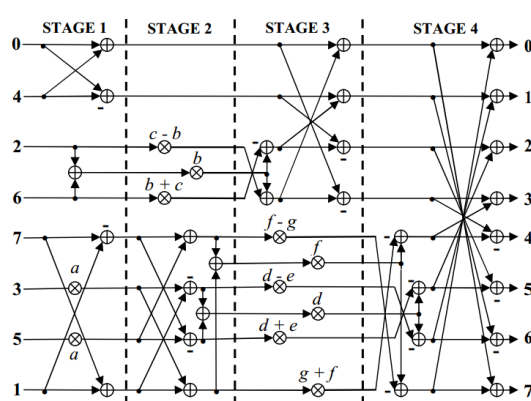


Figure 4.4: IDCT Flow graph algorithm. Taken from [40].

graph algorithm (FGA). This translates into the possibility of using the same multiplier-less blocks that implement the constant multiplications in both the DCT and IDCT, only changing the location where they are computed on the flow graph algorithm.

The flow graph algorithms presented by Aakif *et. al* in [40] and [41] are shown in the figures 4.3 and 4.4, being the symmetry between them easily detected. The 11 multiplications needed to implement the transforms represent 10 different constant operands, since the multiplication by $\sqrt{2}$ is used two times, leading to the need of designing 10 blocks that replace the multiplication by shifts and additions. To achieve this, a fixed point representation of the coefficients had to be defined first, so that afterwards it could be mapped into the desired operations. In their works, Aakif *et. al* chose an unsigned 12 bit precision representation, which was then used to implement each multiplication block with a maximum delay of 3 additions/subtractions. To better understand the process involved, let us take into consideration the multiplication a from figure 4.4. It corresponds to a multiplication by the constant value $\sqrt{2}$, which in turn can be represented by 1.011010100001 in the chosen 12 bit precision representation. By breaking down the multiplication operation at the binary level, the result z of the multiplication of an input x by $\sqrt{2}$ would be:

$$z = x + x \gg 2 + x \gg 3 + x \gg 5 + x \gg 7 + x \gg 12 \quad (4.2)$$

To achieve the maximum delay of 3 additions/subtractions though, a common expression analysis can be made where one tries to find intermediate operations that could be calculated parallelly and lead to a lesser overall delay. By employing this approach, the previous operation could be represented as:

$$\begin{aligned} y_1 &= x \gg 12 + x \gg 7; \\ y_2 &= x + x \gg 2; \\ y_3 &= y_1 + y_2; \\ z &= y_3 + y_2 \gg 3; \end{aligned}$$

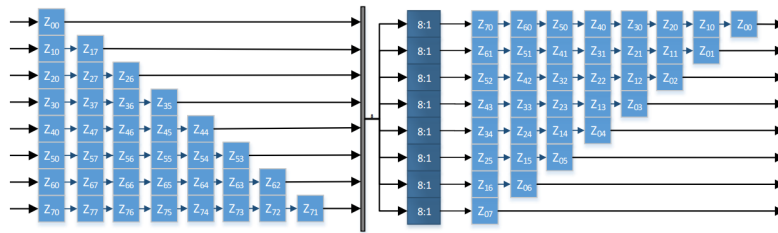


Figure 4.5: Transpose design based on [42]

While by using this method the result uses 4 additions and 4 shifts, the final delay is in fact 3 additions since y_1 and y_2 can be computed in parallel. By repeating this analysis for all constant operands of the other 9 multiplications needed, a multiplication substitute can be designed for each one of the operations. Since the development of these is not the focus of this work, the reader can consult the work of Aakif *et. al* ([40] and [41]) where all computations are presented.

Having developed all of the multiplication substitution blocks, both the 1D-DCT and 1D-IDCT were implemented following the FGA's from figure 4.3 and 4.4, respectively. A pipelined architecture was achieved by using registers at the end of each stage allowing for a more efficient overall operation. Finally, the 2D implementation of both transforms must include a 3-bit right shift to ensure that the location of the decimal point in the fixed point representation of the input data is maintained at the end of the two dimensional transforms.

4.2.1.2 The Transposition

The transposition of a matrix is an important data manipulation operation that can be very resource intensive. As such, over the years several designs have been proposed to achieve an efficient or fast transposition through the utilization, for example, of shift register constructions, systolic arrays or RAM blocks. The transpose architecture chosen for this work was based on an article by Aggoun, A and Jalloh, I ([42]), which used a shift register approach to do the transposition. This design was developed with a data flow of parallel inputs and outputs in mind meaning that in each clock cycle a new row or column can be input into the transposition block and the output will also be an entire column if the input were rows, or vice-versa. Given the parallel inputs and outputs of the one dimensional transform blocks, this transposition design offered a good option, while maintaining the overall pipeline of the entire 2D transform. Furthermore, by using a shift register approach opposed to a RAM block, a later development into an ASIC design does not need to take individual RAM blocks for this portion of the iterative processor.

The dataflow leads to an architecture presented in figure 4.5. It uses $N^2 + N$ registers and N multiplexers $N : 1$, being N the number of elements per row or column of the matrices to be transposed. Since the transposition will operate over 8×8 matrices, $N = 8$. By analyzing the architecture it can be detected that the registers are divided into two stages organized as skewed registers, each with $(N^2 + N)/2$ registers, being the data transferred between the two stages done through the multiplexers. The inputs of the transposition block are then the output vector of N

coefficients of the first transform, which is shifted into the first stage parallelly. The skewed registers are arranged into shift registers of varying lengths, from 1 to N that can be represented, using the notation presented in [42], by R_p^m where m represents if the array is from the first or second stage ($m = \{1, 2\}$) and p represents the length of the skewed register array, taking the values from 1 to N . The operation of the designed transposition block is based on the way data is transferred between the two stages, since each of the shift registers from R_1^1 to R_N^1 will produce one output that is fed into the registers R_1^2 to R_N^2 through the multiplexers. The control of these multiplexers is then crucial to the success of the transposition and is implemented, clock-by-clock as follows:

1. The first N coefficients from the first transform are shifted into the R^1 arrays;
2. The second N coefficients from the first transform are stored on the first positions of the R^1 arrays, leading to the shift of the first coefficients. The data from R_1^1 is shifted into R_N^2 ;
3. A similar shift occurs as in the previous clock cycle. The data from R_1^1 is shifted into R_{N-1}^2 and the data from R_2^1 is shifted into R_N^2 ;
4. Following the same process as before, the data from R_1^1 is shifted into R_{N-2}^2 , R_2^1 is shifted into R_{N-1}^2 and R_3^1 is shifted into R_N^2

The described process continues until $N + 1$ clock cycles have passed. At this cycle, R_1^1 is shifted into R_1^2 , R_2^1 is shifted into R_2^2 and so on, and the first transposed values are ready to be input into the second transform. From this point onward, at each clock cycle, a new set of values is ready to be input into the transform until all N columns, or rows if the input were columns, are processed.

To control the multiplexers a signal must be generated in such a way that the right R^1 output is selected by each multiplexer as the input of its corresponding R^2 array. This was achieved by using a 3 bit control signal generated by a counter, started at the first clock cycle, which is input as the selector to all multiplexers, while simultaneously arranging the input order of these in such a way that for each selector value, the right output was generated. Having this in mind, the order of the data input to each multiplexer is presented on the table 4.1. In this table each column correlates to one multiplexer M_m , identified by the number that corresponds to the index of R_m^2 shift register array, and the rows represent the order, from 0 to 7, by which each R^1 shift array is connected to these.

4.2.2 First Matrix-Vector Multiplication

After the 2D-DCT has been applied to the previous iteration result, a matrix-vector multiplication must be done. Taking into account the design of an efficient pipeline, the production of an output

M1	M2	M3	M4	M5	M6	M7	M8
1	8	7	6	5	4	3	2
2	1	8	7	6	5	4	3
3	2	1	8	7	6	5	4
4	3	2	1	8	7	6	5
5	4	3	2	1	8	7	6
6	5	4	3	2	1	8	7
7	6	5	4	3	2	1	8
8	7	6	5	4	3	2	1

Table 4.1: Input order for the transposition multiplexers

value per clock from this block was preferred, so an architecture that allowed for that was the objective. To achieve the goal, the hardware implementation of equation 4.3 had to be done.

$$out_i = \sum_{j=0}^{N-1} \Phi_{i,j} \cdot in_j \quad , \quad i = 0, \dots, M-1 \quad (4.3)$$

Since the sampling matrix Φ is binary, being mapped into the values of ± 1 , a design could be developed where discrete usage of multiplication blocks was not needed, allowing for a higher maximum clock for the system. The operation can then be seen as a sum of positive and negative values. A tree of adders was then designed keeping in mind the generation of one value per clock cycle as a goal. This tree uses the fixed output vector of N values from the 2D-DCT and at each clock cycle takes one line of Φ as input to choose if a given input value will be the original, i.e., if the coefficient is mapped as a multiplication by 1, or its symmetric if the coefficient is -1 .

A diagram of the proposed block is presented in figure 4.6. The tree of adders is divided into stages, each one comprised of the sum between pairs of two values and a register that allows for a pipeline design. This means that the number of pipeline stages will be equal to $P = \lceil \log_2(N) \rceil$ and that on the stage i there will be 2^{P-i} registers and 2^{P-i} adders. While having a pipeline architecture allows for a smaller path between registers, and subsequently the possibility of using higher clock rates, it also means that the first value out_0 will take the same amount of clock cycles as the number of stages to be sent to the next operation.

From the diagram, it is trivial to find that this design will use:

- Registers : $2^{\log_2(N)} - 1$;
- Adders : $2^{\log_2(N)} - 1$;
- Selectors : N

In addition to the computational block, there must be some control over the data input, since it is premised that the N length DCT transform is known before starting the multiplications and the rows of the Φ matrix must be input at a certain rate. To achieve this an extra control block, represented by the state diagram in figure 4.7 was implemented. Relevant control signals set to the logic value 1 are shown under the name of the state where this happens. The complexity of

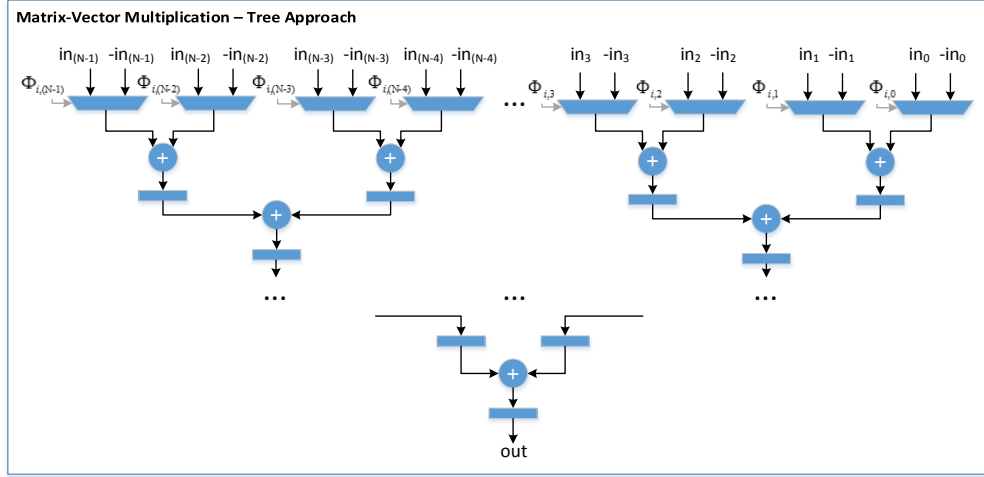


Figure 4.6: RTL Schematic of the First Matrix-Vector Multiplication Block

its architecture is low, since it implements a simple finite-state machine (FSM) with the following states:

1. Wait for a master control signal that starts the block operation;
2. For 8 clock cycles, save the output data from the 2D-DCT into an N length vector, 8 coefficients per cycle:

$$in_{(8 * clockCounter + ind)} = dctOut_{ind}, \quad ind = 0, \dots, 7;$$

3. Having saved and aligned the N length vector, start the multiplication. At each clock cycle, feed one row of the sampling matrix Φ , $phiLine_{ind}$, to the selectors of the input stage of the tree of adders block, for a total of M clock cycles;

$$phiLine_{ind} = \Phi_{clockCounter, ind}, \quad ind = 0, \dots, N - 1;$$

4. After the final row has been input, maintain the enable of the adders tree until the final value has been output and then return to the waiting state.

4.2.3 Second Matrix-Vector Multiplication

Differently from what happens in the first matrix-vector multiplication, because of the way the pipeline was designed, the second matrix-vector multiplication block will receive one value from the subtraction per clock cycle. In order to take advantage of this, a different design must be employed. The operation to be implemented, $\Phi^T (y - \Phi \cdot \Psi \cdot x_i)$, is presented in the equation 4.4.

$$out_j = \sum_{i=0}^{M-1} \Phi_{i,j} \cdot in_i, \quad j = 0, \dots, N - 1 \quad (4.4)$$

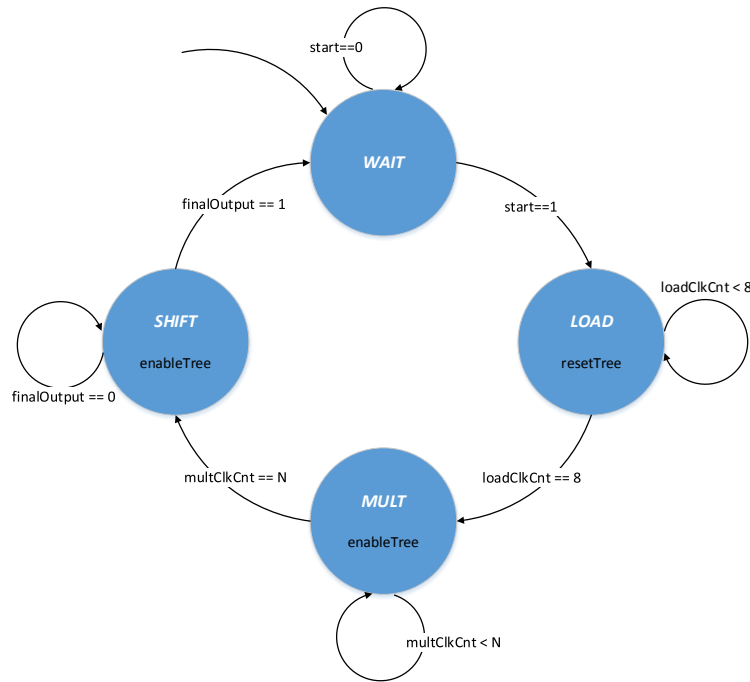


Figure 4.7: First Matrix-Vector Multiplication Control State Diagram

As the equation shows, by having access to one value in_i per clock cycle, a sum, or subtraction depending on the corresponding Φ coefficient, can be made to each currently stored out_j element until the final vector is computed. Since the input data arrives at a rate of one value per clock cycle, the employment of a group of accumulators is both the most efficient and the least resource intensive implementation. A diagram of the design is shown in the figure 4.8.

Similarly to the input step used on the first matrix-vector multiplication block, the usage of a digital signal processor (DSP) for the multiplication is not needed by employing a multiplexer that selects either the original value or its symmetric depending on the coefficient from Φ . Using accumulators, the multiplication operation takes the same number of clock cycles as the number of input values, M , after which the resulting vector can be sent to the next operation.

Since this design employs an accumulator approach, resource usage is very low and easily scalable:

- Registers : N ;
- Adders : N ;
- Selectors : N

As in the previous block, some matter of control is needed to ensure the success of this design as in addition to the feeding of the correct row from the sampling matrix at the correct time, the data stored in each accumulator must be reseted accordingly. Following the same principle, the computational block is controlled by a small control sub-block, which at the start of the multiplication receives a reset signal that forces the reset of all of the accumulators. Then a counter is used

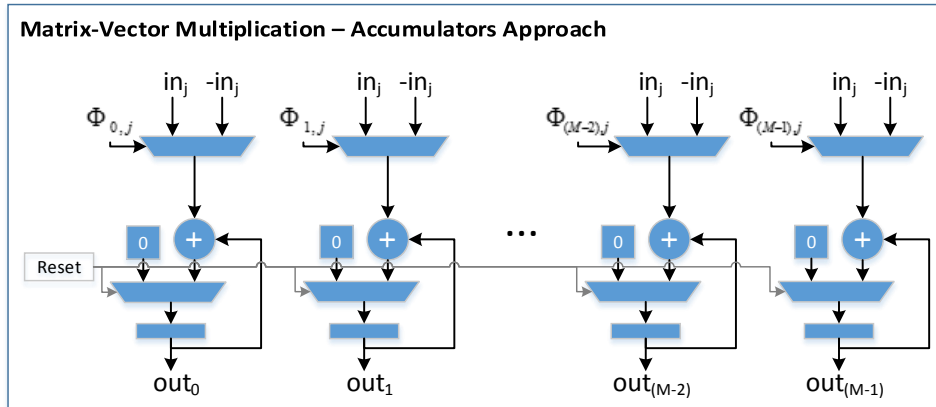


Figure 4.8: RTL Schematic of the Second Matrix-Vector Multiplication Block

to select one row of the sampling matrix each clock cycle and subsequently the control sub-block feeds it into the computational block.

4.2.4 Scaling the Data - The Iteration Step

To guarantee the convergence of the algorithm, as discussed in previous chapters, one of the main requirements would be that $\|\Phi\|_2 \leq 1$. Given the structure of the sampling matrices used by this system, the requirement will never be verified, so an alternative method to achieve convergence was proposed: the application of a scaling step. The process to compute the step size includes two parts:

1. The computation of the weights from the sampling matrix, and the selection of the maximum absolute value from the results;
2. The computation of the closest power of 2 that will be used to scale the data using a single shift operation.

An RTL diagram of the step computation block is shown in figure 4.9. From its analysis it is possible to find each of the two parts highlighted and the design followed. To compute the weights, a similar tree of adders as the one presented in 4.2.2 is used but without implementing the input stage. The inputs of the tree will then be the elements of the matrix to be processed, extended to 6 bits length and mapped as -1 if the coefficient is 0 and 1 if it is 1, and a sequential sum of these is done. At the final stage, after the final sum, the result can be positive or negative, so a modulo operation must be made to ensure that the weight is positive. Since the tree of adders is divided into stages, a pipeline approach can be used leading to the possibility of inputting one set of Φ elements per clock cycle and obtaining, after an initial delay, one result per clock. During this part of the operation of the step size block, each time a new result is output from the tree, it is compared to the temporary maximum weight and if the new one is larger, it becomes the temporary maximum. After all the weights have been computed, a maximum is found and the next computation phase can begin.

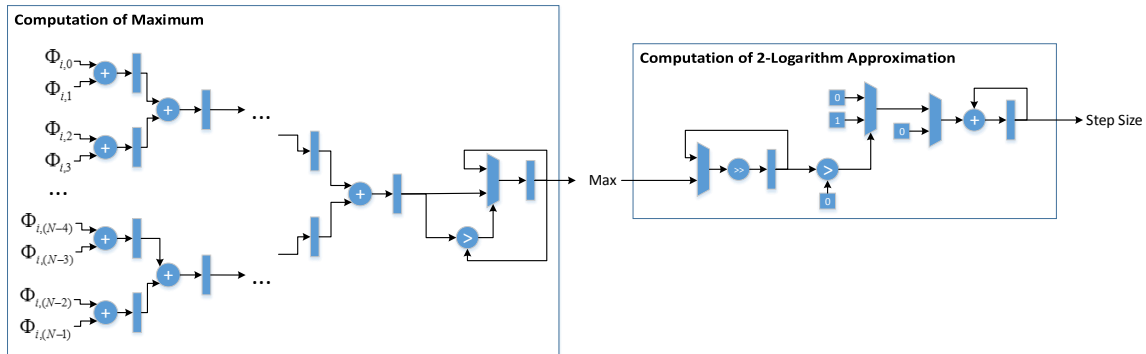


Figure 4.9: RTL Schematic of the Step Computation Block

Having found the weight w that will ultimately set the step size, a computation of the closest power of 2 must be made. This is done in order to allow the scaling operation to be implemented as a single shift operation as opposed to implementing a division block or trying to find a suitable division by multiple shifts. The method used to approximate the 2-logarithm operation is simple but effective and is based on the following algorithm to calculate $L = \log_2(w)$:

1. At the start, set $r_0 = w$ and $L = 0$;
2. On iteration i of the algorithm compute $r_i = r_{i-1}/2$;
3. If $r_i > 0$, increment L by one, $L = L + 1$, if is not larger than zero, maintain L . If $i < 6$, repeat the previous step 2, if it is not smaller than 6 go to the next step;
4. The computation has finished and L contains an approximation of the logarithm.

After the step size has been computed, a scaling block, for which an RTL schematic is shown in figure 4.10, is used to implement the shift operation. As was referred on the section 3.1.2, the effective step size will be $\frac{1}{w^2}$, so the designed scaling block should execute the shift operation two times to achieve the desired result. With this in mind, the architecture behind the scaling block is quite simple:

- There are two types of main inputs: the shift coefficient that will be the result of the $L = \log_2(w)$ operation and the data to be scaled, which in the case of the envisioned data-path corresponds to an 8 elements vector;
- An array of parallel chains of two right shift blocks receive the data as input and the coefficient and compute the scaling in two steps;
- A registered output vector stores the results.

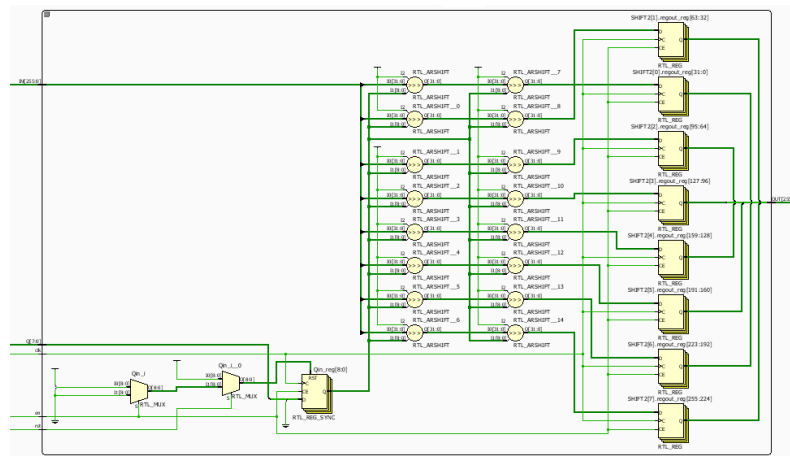


Figure 4.10: RTL Schematic of the Scaling Block

4.2.5 Threshold - Finding the Right Coefficients

The final block, and one of the most important of the whole system is the threshold block. This block implements the thresholding operation that gives the algorithm its name, being responsible for finding the K largest coefficients of the input vector and setting all other values except those to zero. This is accomplished in two steps:

1. The K largest coefficients and their position on the vector are sorted;
2. The sorted data is aligned on a new vector where all other positions are 0.

The general design of the thresholding block can be seen in figure 4.11. Since each step has a certain level of complexity, they are explained separately.

4.2.5.1 The Sorter

The sorting of a vector can be a computationally expensive operation since to find the sorted position of a given value, it must be compared to all other already sorted values and inserted into the position between a higher and a lower value than itself. Because of this, allied to the need for fast and efficient ways of sorting data, several sorting algorithms and designs have been proposed, some better suited for small sets of data while others excel at sorting bigger arrays. For this work, the objective was to design a sorting structure that used a small amount of clock cycles, had low resource impact and could be easily scaled for systems with other specifications.

The sorting operation has the objective of finding the K largest coefficients, and maintaining their original location on the vector when the threshold is completed, so that the other positions can be set to zero. There is then, a special specification for this sorting operation: since the smaller $N - K$ values will be set to zero at the threshold step, only the K largest vector coefficients entries have any importance. This allows for a simplification of the design of the sorter by the removal of any sorting or storing of values positioned after the K th largest one. By using this approach, there

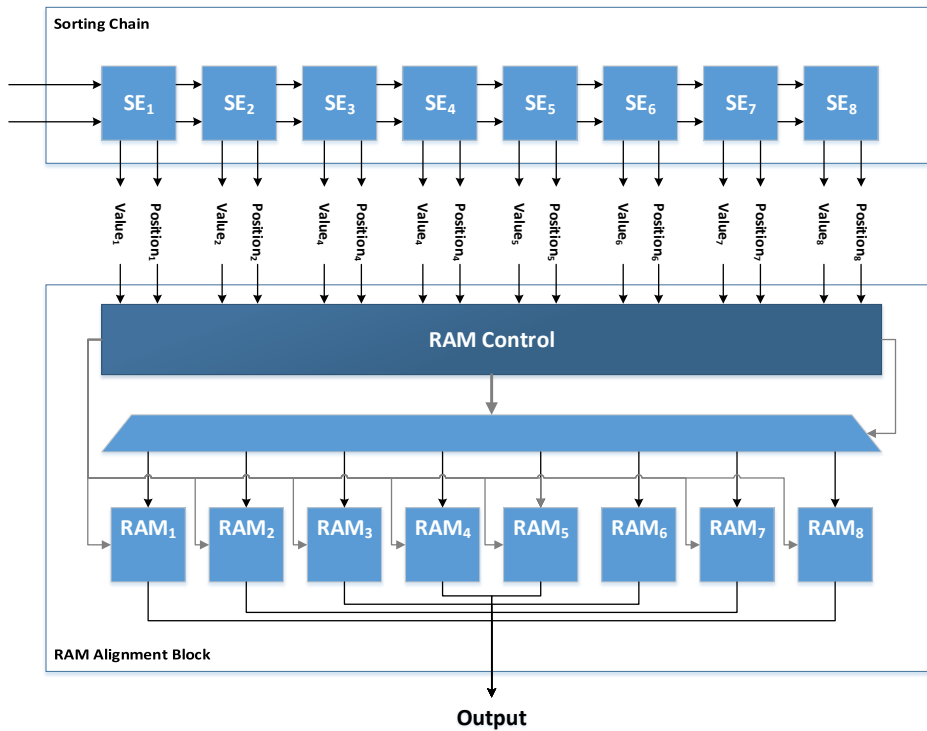


Figure 4.11: RTL Schematic of the Thresholding Block

is a lower resource usage and more efficient sorting structure since past the K th comparison, if the value is not larger than an already stored value, it can be discarded, saving $N - K$ comparisons.

Having defined the objectives and general structure of the sorting block, the final block should be able to handle the input of 8 coefficients per clock cycle corresponding to the parallel outputs from the 2D-IDCT block. This means that the sort block should ingest 8 unsorted values and their respective positions on the N length vector and do at least the comparison with the largest of the temporarily stored sorted values before the next clock cycle occurs and the next 8 values arrive. The design of this sorter block was adapted from the work of Hussain *et. al.*, [43], where a systolic array structure is presented with sorting elements, arranged in chains. By using this idea, a sorter chain was implemented by designing a sorting element (SE) that was capable of handling the specifications previously referred.

In figure 4.12 an RTL level schematic of the designed sorting element is shown, as well as of the smaller cell blocks that compose it. A sorting element receives as inputs the values V_i to be compared and their respective positions P_i on the N length vector. From these, the maximum value is found and then compared to the presently stored on the SE maximum value. If the new maximum is larger than the stored on it, the new maximum and its position are stored on the SE, if it is not larger, the maximum is maintained. After this, the values and positions not stored are sent to the next element in the chain. This means that the SE corresponding to the largest value will be at the start of the chain, receiving data from outside while the rest of the elements have their inputs connected to the outputs of the previous element on the chain. Each SE will then have a total of 8 value inputs, 8 position inputs, 8 value outputs and 8 position outputs for the next element and

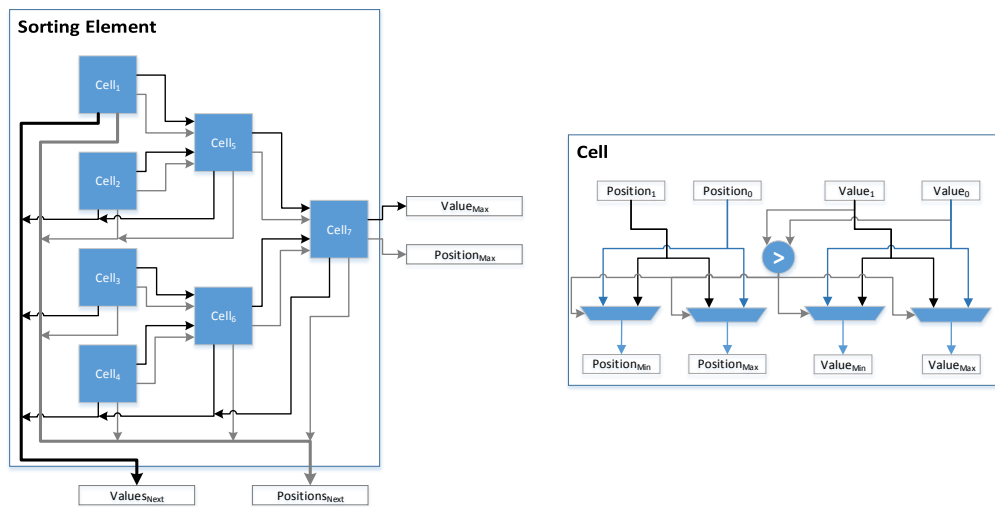


Figure 4.12: RTL Schematic of the Sorting Element Block and of one of its Cells

two final outputs: the maximum value and its position. After all data has been passed through the chain, the stored values and positions on each element of the chain will correspond to the K largest values of the vector. Since the objective was to allow for the input of a new set of 8 values and positions in each clock cycle, the operations inside each SE had all to be made in one clock cycle, resulting on the implementation of a tree of comparators in order to find the proposed new maximum and an extra comparison between this and the current maximum.

The sorting operation will then take $8 + K$ clock cycles, the number of output cycles from the 2D-IDCT plus a final delay for the last input data to pass through the entire chain.

4.2.5.2 Aligning the Data

After the sorting operation has ended, the N length vector must be built with the K largest values on their respective positions and all of the other positions set to zero. To achieve this, several designs were considered, being the most time efficient, without being very resource intensive, the usage of a RAM architecture. This can take advantage of the knowledge of the location of the K largest values. Following this reasoning, the RTL schematic of the designed architecture is shown on the figure 4.13. The parallel RAM design, the input multiplexers and the output aggregation register are visible in the figure. The operation of this RAM is done as follows:

- The write operation uses a 6 bit address to choose first, using the lower 3 bits, one of the 8 RAM blocks and then, using the higher 3 bits, the position of that block where the input value will be stored;
- The read operation uses a 3 bit address utilized to choose the read position from each one of the RAM blocks. The 8 values are concatenated and an eight elements vector is formed as an output.

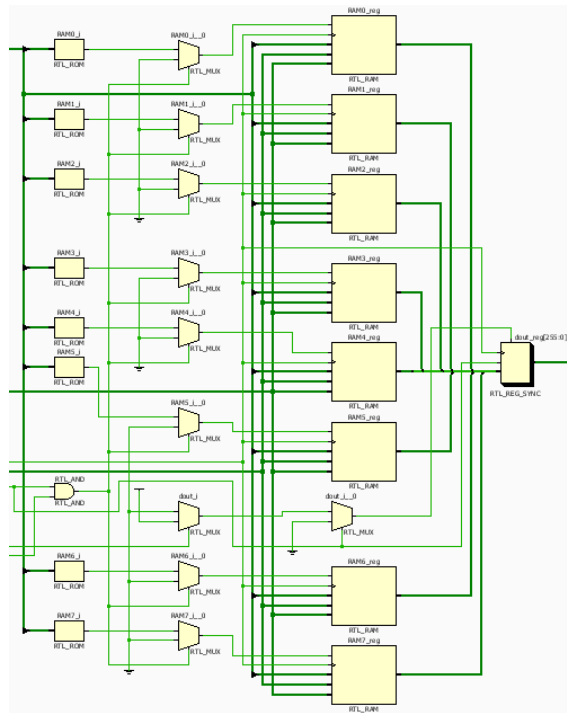


Figure 4.13: Threshold RAM Architecture

This particular arrangement of RAMs and access control was built with the consideration that there are no guarantees about the relative location between each sorted value, but at the same time, after storing all of the K values, the output N length vector must be produced as fast as possible. This led to a design that uses 8 RAM blocks, which allows the intended write operation in an element-by-element basis, while reducing the read delay from N clock cycles to only 8.

By guaranteeing that at the start of the operation all of the data stored in the RAMs is equal to 0, this block can be used to first store the sorted values into their respective positions and then reading the N positions in order to build the N length vector, which represents the new proposed recovered data. This means that after the reconstruction of the output N length vector, which is the new proposed recovery, the RAM must be reset to the values of 0 to ensure that only the K written positions of the next iterations are present on the next recovery vector.

4.2.5.3 Control

Having addressed the architecture and operation of each one of the two main blocks that implement the threshold computation, their control and dataflow must be addressed since there are some considerations to have in mind:

- As was referred, the sorting block must receive as inputs both the values and their location, which must be generated by the control;
- After the final 8 values are input, there must be an extra K clock cycles where the sorter operates and the inputs during this phase must be carefully selected to prevent errors;

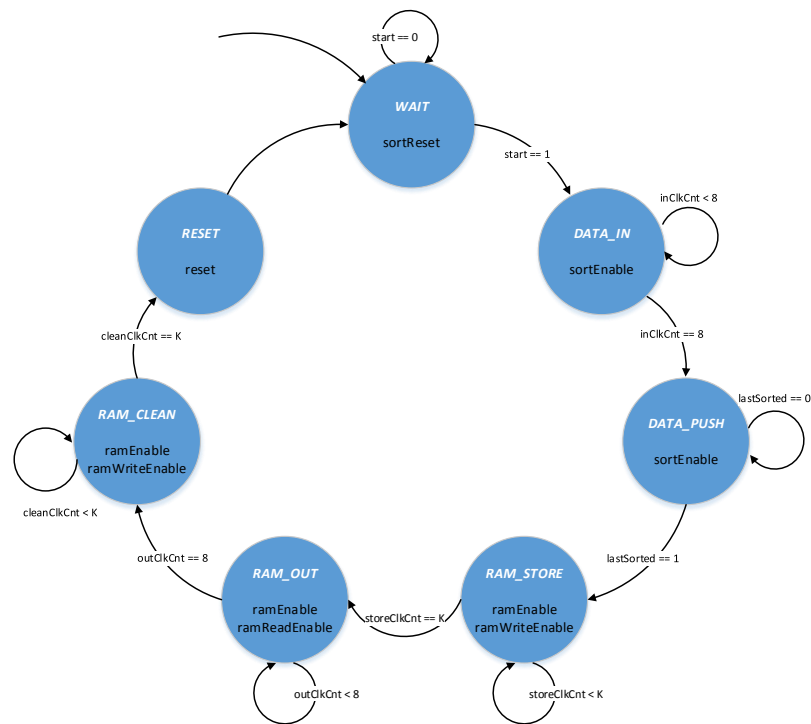


Figure 4.14: Threshold Control State Diagram

- After the data has been output from the RAM block, it must be prepared for the next iteration, since all positions must be set to zero before starting the write of the K sorted values.

The control block was then developed to take care of the data transfers into each block, the alignment of their outputs and the enabling or disabling of their operations. The most important element of this is the FSM responsible for the control of all structures and that has the following states:

1. Wait for the master control signal to start the threshold operation;
2. After receiving the start signal, start the input of the 8 values that arrive each clock cycle from the previous block into the sorter block. Simultaneously, the position values are generated starting from 0 to 7 on the first cycle, 8 to 15 on the second, and so on for a total of 8 clock cycles;
3. After 8 clock cycles the sorting operation is maintained for an extra K clock cycles to allow the final input to reach the end of the chain. The inputs of the sorter are all set to zero so that these values are not stored on the chain;
4. At the end of the sort operation, for a total of K clock cycles, each one of the sorted values is written into the RAM block;

5. Having written all K largest values, the RAMs are read 8 values per clock cycle and the data is aligned into the output register vector of the threshold block with N length. This operation takes 8 clock cycles;
6. After the output is ready, using the positions still stored in the sorting chain which give the addresses where the RAM values are not zero, the RAM block is prepared to the next iteration by writing 0 in each one of the K positions previously used. This process takes K clock cycles since only one value can be written per clock cycle;
7. A final reset signal is propagated to the sorting elements to ensure that all values are set to zero and the block returns immediately to the wait state.

These states can be represented in a state diagram such as the one shown in figure 4.14. In it, relevant control signals are shown under the name of the state where their logic value is set to 1.

Finally, as can be perceived, the whole computational block in addition to the sorting block, the RAM and the control has some smaller and simpler structures like the one responsible for the generation of the positions vector that must be input with the values on the sorter, the alignment of the output data from the RAM block that allows for the output of length N , and other multiplexers and counters.

4.2.6 Master Control - Managing the Iterative Processor

In order to control the iterative process, a master control block had to be developed. Its main function is, as the name implies, to manage the control signals of each of the main blocks of the IHT processor, the most important ones being the enable and reset signals of each one. This control is mainly achieved by implementing an FSM that will then be used to assert if a control signal should be set to 0 or 1. The referred finite-state machine is comprised by the following sequence of states:

1. The processor is in a wait state until an outside signal is detected which indicates the arrival of new data to be processed. The 2D-DCT block is maintained with its reset signal high, so it is ready to start computing the transform when needed. When the start signal is detected, the state changes to the next one;
2. The 2D-DCT block is enabled and the transform starts. When the first vector of 8 elements is output from the transform block, the next state is reached;
3. Having started outputting transform results, the 2D-DCT is maintained enabled until all results are output. Simultaneously, the first matrix-vector multiplication block is enabled so that its control block starts storing the 64 length transform result;
4. The 2D-DCT operation has ended, so the block is disabled. The first multiplication block is maintained active, starting the multiplication portion of its processing flow. The reset signal for the second multiplication block is set to 1 in preparation for the moment the first value reaches it;

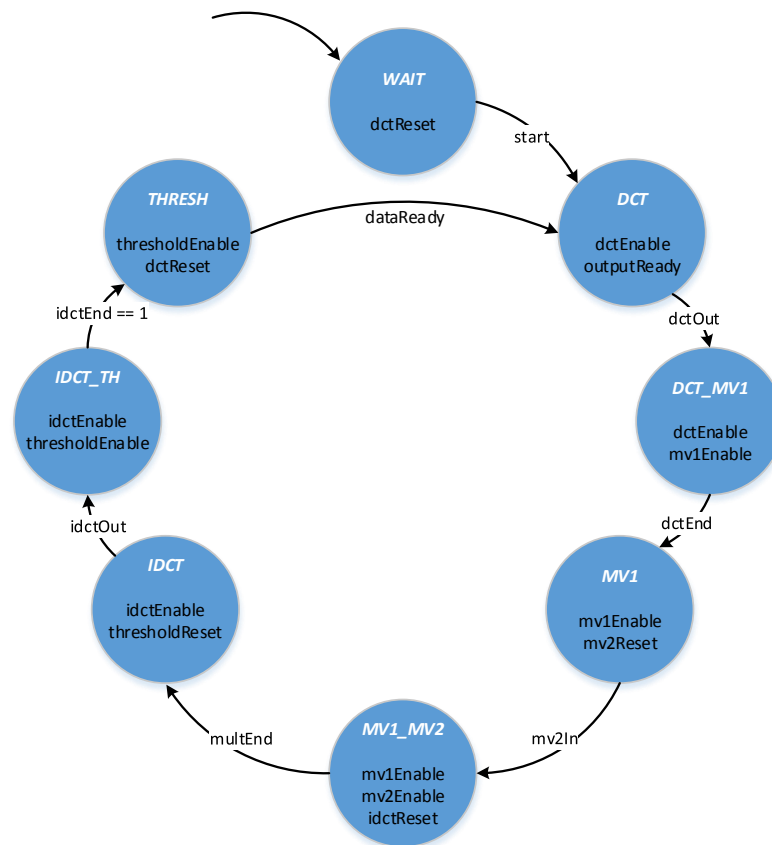


Figure 4.15: Master Control State Diagram

5. When the first subtraction element reaches the second matrix-vector multiplication block, this state is reached resulting on the enabling of the block, and maintaining the operation of the first multiplication block until all multiplications are complete. Simultaneously the reset signal for the 2D-IDCT block is set, so that it is ready for the next computation;
6. After both the first and the second multiplication blocks have completed their operations, the vector which is to be the input of the 2D-IDCT block is available. Both multiplication blocks are disabled, the 2D-IDCT block is enabled and the inverse transform operation starts. A reset signal prepares the threshold block for its operation. When the first output from the 2D-IDCT block arrives at the threshold block, the next state starts;
7. As the first input of the threshold block arrives, it is enabled. The 2D-IDCT block is maintained enabled until the transform has been complete;
8. When the 2D-IDCT has output all the transform results, it is disabled and the threshold block is maintained enabled until a signal is output from the block indicating the end of the computation. The detection of the end of the threshold computation means that a new proposed recovery is ready, a signal is sent to the exterior of the IHT processor indicating this and the state is changed to 2, effectively starting a new iteration.

A states diagram of the described FSM can be seen in figure 4.15. The control signals that are active in each state are shown below their respective names. For simplicity, the diagram does not represent the behavior from the reception of a general reset signal, which would be represented as a transition from all states to WAIT.

The control tries to achieve an efficient pipeline, activating each block when needed and disabling it when their computations have ended. It also maintains the fixed values from a block for as long as possible, sending the reset signal only before the block will be needed. By implementing the FSM flow as explained, the processor will continue iterating as long as the general enable signal is set to 1, meaning that the number of iterations that a recovery takes is not defined by the iterative core itself, but by the control environment created by the top level module where it is inserted. This means that the iterative processor can be used in different situations, where the number of iterations is dynamically computed by the top level control or supplied by a user.

4.3 The Top Level Module - Integrating the Iterative Processor

Having explained the operation of the iterative core, and its components extensively, the top level module where it is integrated must be addressed. The main objective of this module is to integrate the processing core with a control unit and an interface to enable the access to it allowing for the creation, later on, of an intellectual property (IP) block. This IP block is important for the development cycle since it will allow for an abstraction that will be used on the board design step discussed in the next chapter.

To operate the processor, as was previously referred, an interface and control must be used. The interface is used to allow for a standardized way of communicating with the iterative processor, enabling as well the transmission of status reports and reception of control commands. To receive and react to these commands, a top level control was developed, resulting on the architecture previously presented in figure 4.1. On the figure an extender block is also represented. This block is responsible for receiving the 16 bit measurements and extending them to a 32 bit, $Q16$, representation. During this operation, the equivalent to an 8-bit right shift is done on the measurement data, resulting on a 32-bit representation where the original 16-bits have an 8-bit "padding" on each side.

Since the control and interface blocks require a more extensive explanation, they are separated into two subsections. A fourth peripheral, the Mixed-Mode Clock Manager (MMCM), is also referred, since it has the crucial task of generating the clock used by the block. The block is not represented on the general figure since it is connected to all blocks as the clock signal.

4.3.1 Top Level Control

The top level control block is responsible for the reception of commands from the exterior, reporting the hardware status and controlling the iterative processor's operation with general control directives such as: start a new recovery, reset the core or disable its operation. It is also responsible for controlling the number of iterations that the iterative processor runs before signaling that the

recovery has ended and is ready to be read. With this functions in mind, an FSM was developed with the following states:

1. The control waits for an exterior command to start the operation. As the signal to start is received, the state changes to the next point;
2. Having received the command to start, the control waits for the exterior to load the data to be recovered through the interface. After the loading is done, a signal is received to let the hardware know that it can start the recovery;
3. A reset signal is sent to the iterative processor to ensure it is ready for the computation;
4. The iterative processor is enabled and the *data_in* signal is set to 1 indicating that the processor should store the input values;
5. The control awaits in this state for the signal indicating the output of a new proposed recovery and switches to the next state;
6. A counter that stores the number of iterations is incremented and if the target number of iterations is met, the state switches to the next point, if not the state is returned to the previous one where a new proposal is awaited;
7. Having run the target number of iterations, the recovered data is stored and the exterior signaled that the recovery has ended. After this, the state returns to the first point, where the iterative processor is disabled, waiting for new data.

The control is responsible for using the interface to transmit the recovered data. Since it ensures that the processor is kept disabled during the time between the end of a recovery and the exterior command signaling a new one, this control leads to a power consumption decrease on the system by preventing the unneeded switching of the elements that form the processor.

4.3.2 Mixed-Mode Clock Manager

The Mixed-Mode Clock Manager (MMCM) is an IP block included in the Vivado software, that can be used to generate different clock signals. It can generate up to 8 different clock signals based on an input clock signal by means of multiplication and division. Other more complex operations such as the manipulation of the phase of the signal are features present in this block.

In the implemented design, the MMCM is used to generate the clock signal of the IHT iterative processor and other control blocks except the AXI Slave Interface, which has a clock signal automatically generated for it. Using the base clock signal available from the interface, which operates at 50MHz , the usage of the MMCM allowed the implementation of the IHT IP core to operate with a clock signal between 50MHz and 100MHz , as discussed in the next chapter.

4.3.3 AXI Lite Slave

The interface chosen to access and control the programmable logic hardware was an AXI Lite interface. Part of ARM's Advanced Microcontroller Bus Architecture (AMBA), the Advanced eXtensible Interface (AXI) protocol is an open standard developed for on-chip communications and interconnects working at high clock frequency. This protocol is based on a master-slave connection, so in this design, the IHT IP block will be the slave, meaning that it cannot initiate data transfers, only answering to the master's requests.

The AXI Lite interface is then used to transfer control signals and data (measurements from the master to the slave and recovery results from the slave to the master). Each data transaction between master and slave has a length of 32 bits. Therefore, arrays of 32 bits registers are integrated into the interface to allow the storage of the information exchanged between master and slave. The access to these registers is then possible through addressing, when the access is made from the master, or through input and output ports of the interface when made from the PL design. The registers that are present in the interface are:

- 2 control registers: one for the commands written by the master, one for the master to read the state of the programmable logic;
- 32 registers for the sampling matrix data, written by the master and read by the slave;
- 8 registers for the measurement data, written by the master and read by the slave;
- 64 registers for the recovered data, written by the slave and read by the master.

The AXI Lite Slave was not developed as a product of this work, since it is included as an IP core of the Vivado Design Suite, the design tool used, and can be instantiated directly. It was rather, altered to include the referred registers, since by template it does not include them, allowing instead the designer to use the addressing functionality as he deems fit to the objective of the work being developed.

Having described the interface of the implemented design to the exterior, as well as all the blocks inside the design, it is now necessary to test and validate it. This is the subject of the following chapter.

Chapter 5

Test Methodology and Results

5.1 Testing Environment

During the development and testing of the IHT design the hardware platform used was the ZYNQ-7000 ZC706 SoC evaluation board from Xilinx. This board contains the XC7Z045, a system on chip (SoC) from Xilinx, that is divided into a Processing System (PS) and a Programmable Logic (PL) sections. These offer different features and can be used either in conjunction, in parallel or only one of them at a time.

Some of the hardware contained in the PS section is:

- ARM Cortex A9 dual core processor with 32KB L1 cache, 512KB L2 cache and maximum frequency of 800MHz;
- 256KB on-chip memory;
- External memory interfaces with support for DDR3, ECC and other features;
- 8-Channel DMA controller;
- Several I/O peripherals and interfaces such as USB, CAN, UART, SPI and Ethernet, among others.

The PL section, based on a Xilinx Kintex-7 FPGA, also offers several hardware structures such as:

- 350K logic cells, which correlates to approximately 5.2M ASIC gates;
- 218,600 look up tables (LUTs);
- 437,200 flip-flops (FFs);
- 545 units of 32Kb Block RAM, amounting a total of 2,180KB;
- 900 DSP slices;
- Gen2 x8 PCI Express[®].

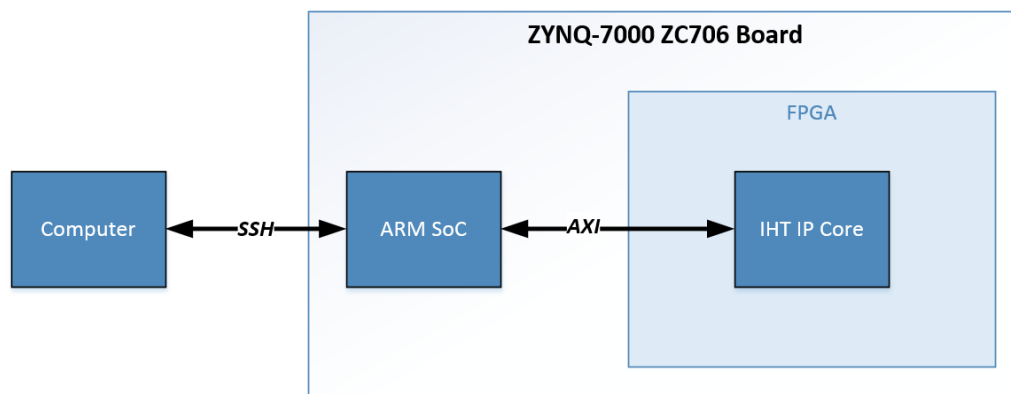


Figure 5.1: High Level System Diagram

5.1.1 High Level System

At the highest level abstraction, the whole system comprises 3 components:

- The **computer** utilized by the user to connect to the ZYNQ development board, and execute data transfers;
- The **ARM CPU** that is part of the PS section of the ZYNQ development board;
- The **IHT IP core**, the recovery processor running on the FPGA, the PL section of the ZYNQ board.

By taking notice of the connections between the 3 elements represented in figure 5.1, in a diagram form, one can see that the first link, between the computer and the ARM CPU, is a Secure Shell, or SSH, connection. A Secure Shell connection is an encrypted network protocol that allows a user to remotely access a machine and execute commands through a shell. In addition, this solution allows for the parallel development of several projects that are tested using the same development board. Together with the access to the embedded operating system running on the ZYNQ, which will be used to run the C program explained in the next subsection, the SSH connection is used to send the binary files containing the measurement samples and matrices to the development board and retrieve the recovered data after the process has been completed. This is important since it is the only way to test the recovery core: measurement data must be created on the computer and then a recovery must be attempted by running that data through the recovery processor. Since the objective of the work is the recovery core, this data can be created off-board, sent to recovery and then retrieved and aggregated again off-board.

The second link represented is the one between the ARM CPU and the FPGA core. This link is used to interface the input and output of data from the IHT IP core, and also to enable some control from the application running on the CPU. Since the recovery core has the task of finding the coefficients in the transform domain, from a set of measurements and a sampling matrix, the input and output control can be made by means of a higher abstraction method: a C program running on the CPU. To achieve this communication, the AXI protocol previously referred in 4.3.3 is used.

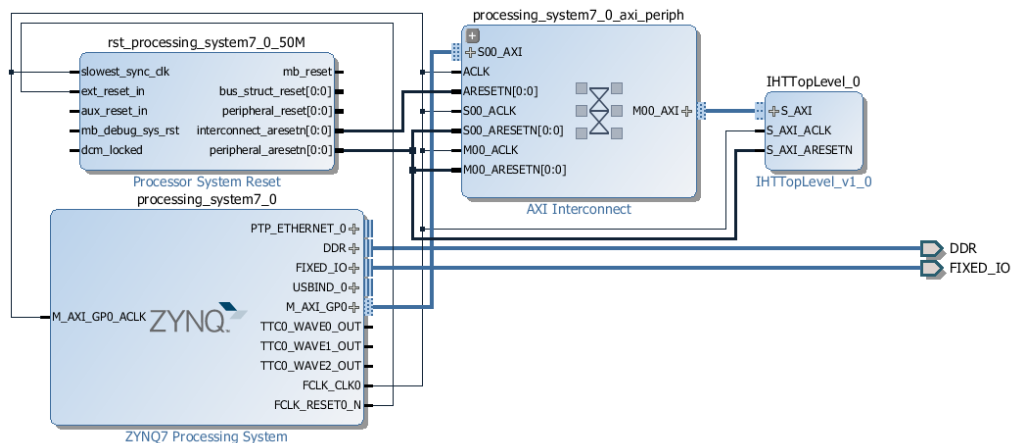


Figure 5.2: ZYNQ Board Design

It allows the C program to read and write directly on registers instantiated on the programmable logic, enabling the input and output of data from the PL section of the board.

5.1.2 ZYNQ Board Setup

As referred earlier the development of this work was done using the Vivado Design Suite, a software development tool from Xilinx, the same company that sells the ZYNQ board used. As such, the tool has features that directly link to the board being used as a development platform (Vivado is a board-aware). Since the target board for the design can be defined directly, the software offers features such as automatically defining the board inputs and outputs restrictions, allowing a better IP configuration based on the board resources, among others.

To develop a design that includes IP cores, connections between different board resources, connections between the processing system and the programmable logic, and other complex systems, the design software offers a tool that allows the user to connect and configure these components using a block design approach. Figure 5.2 shows the block diagram developed for this work. The main blocks presented in this design are the *ZYNQ7 Processing System* and the *IHT-TopLevel_v1_0*. The first one is an IP block part of the design suite catalog that allows the user to instantiate the processing system of the target board. Since the board is known by the development tool, this block is configured with presets specific to the board as well as allowing the user to fully customize its operation. By using the included configuration tool, the clock frequency, enabled inputs and outputs, and other features of the PS can be controlled to match the specifications of the final system. The second block is the IP block created from the top module described in 4.3. This is a feature from the development software, which allows the packaging of the design created by the user into an IP block. By doing this, the block can be included in the board design and connected to other components through its interfaces. Since the AXI Lite protocol was chosen as the sole interface with other blocks, the signals pertinent to it are the only inputs and outputs from the IHT IP core. Some other blocks can be seen in figure 5.2. One is responsible for managing

the reset signals, *Processor System Reset* and the other is the peripheral used to interconnect the PS and the IHT IP block using the AXI protocol.

By using this block design approach, all of the connections related with the AXI, reset signals, clock signals and others are automatically implemented by the tool. The tool also allows the user to define the address space that should be reserved to the IHT IP block. On the development board, each block can be addressable and have a range of addresses allocated to it. This allows communication between IP blocks using for example the AXI interface. By choosing the address and range of the IHT IP block, it can then be addressed by a C program running on the PS, as explained in next subsections.

After having completed the block design, an HDL wrapper is created by the tool, the design can be synthesized and a bitstream created to program the PL.

5.1.3 Controlling the IHT IP Core through Software

The hardware implementation of the IHT processing core was the main focus of this work, but in order to validate it a test environment had to be created. In this line of thought, an iterative processor was included on a top level module that allows an access to the IHT IP Core through a standardized protocol – ARM's AXI Lite – enabling the transfer of data to be recovered, the output of said recovery and some status and command signals. To use these features a C program, responsible for the control of these transactions, had to be developed.

The ZYNQ board, allows the execution of applications using two methods:

- On a bare metal way, by which the ARM CPU is directly programmed through a connection like UART and the programmable logic bitstream must be loaded via, for example, JTAG;
- Using an embedded operating system running on the ARM CPU, giving the user features such as a file system structure and the possibility of programming the PL by means of a device driver. It also allows for more advanced ways of interfacing with the board, for example by means of the creation of SSH sessions.

As the testing framework leverages heavily on the utilization of binary files to store the measurement data to be loaded into the IHT IP core, as well as to store the results retrieved from it, a file system is an important feature. Furthermore, the ability to program the PL using a device driver together with the possibility of accessing the development board through SSH are advantages that enabled a more streamlined development process. For these reasons, the ZYNQ board was used with an operating system running on the ARM CPU, an Arch Linux distribution for ARM systems specifically prepared to run on this development board.

Having the support of a file system, a C program was developed to serve as a software front that enables the control over the data transfers and the programmable logic block operations, using the AXI interface between the PS and PL. As referred, the instantiated IHT IP core is addressable through a range of addresses. These addresses will correspond to registers that can either be read by the PS (and written by the PL) or written by the PS (and read by the PL), meaning that by using this connection, the C program can control the data flow, as master of the AXI Lite connection.

The program will then have the following overall tasks:

- Open both binary files containing the measurement data and the respective sampling matrices;
- Fetch each set of measurement data and sampling matrix and load it into the IHT IP core;
- At the end of each recovery retrieve the data from the PL outputs and write the results into the output binary file.

The connection between the PS and PL is controlled by the C program so it must follow a protocol defined at the development of the master control block of the top module of the IHT IP core. This protocol leads to the following steps:

1. The CPU waits for the status register of the hardware core to signal that it is waiting for new input data;
2. After the hardware core signal arrives, the CPU writes the next sampling matrix and sampling measurements directly on the input registers of the hardware core;
3. The CPU signals, using its status register, the end of the transfer and the hardware core starts the recovery iterations;
4. After running the defined number of iterations, the hardware core signals the CPU that it has ended the recovery;
5. The CPU retrieves the recovered data and signals that it is ready to send new data for recovery;
6. The process repeats until there is no more groups of measurements and sampling matrix to send for recovery.

As a result of running this program, the binary files with the measurement data and matrices are loaded into the file system through the SSH connection, with a user's computer, are processed and an output binary file is created. This file is then transferred to the user's computer using again an SSH connection.

5.1.4 MATLAB - Generating Measurements and Aggregating Results

Up until now, the development of the iterative processor that implements the iterative hard thresholding algorithm was fully explained followed by the design of a framework that allows for a user to access and use this same processor running on the programmable logic of a development board. Having done this, the recovery can only be tested with real data if that data is first generated. For this matter, a MATLAB program had to be developed to create that data and save it in some way that could then be relayed to the ARM CPU to pass onto the IHT IP core. After the recovery of all patches is done, the result is a set of transform coefficients that need to be inversely transformed and aggregated into a recovered image, so another MATLAB program had to be developed for that purpose.

5.1.4.1 Before the Recovery - Generating Measurement Data

The image sampling framework follows the same main points discussed in 3.3.2, where the image is divided into several patches and the sampling is done by generating a $\Phi_{M \times N}$, $+1/-1$, random binary matrix and rasterizing the patch into a vector of length N . The main difference, however is that the program flow cannot follow the sampling of a single patch, while immediately doing the recovery and alignment of the recovered patch before going for the next patch. Instead, this program must generate and write into a file all sampling matrices and the corresponding measurement data for all patches of the image.

The results of the execution of the program that generates the measurement data are two binary files: the first containing the measurement matrices and the second one containing the corresponding measurement results. To achieve this, the program developed uses, for each patch, the following sequence of steps:

- A random, $+1/-1$, binary matrix Φ is generated with dimensions $M \times N$;
- The patch is rasterized into an N length vector, p ;
- The measurement vector is computed by $y = \Phi \cdot p$;
- The sampling matrix is written into the corresponding file with each position being encoded into one bit: 1 if the element is $+1$ and 0 if it is -1 ;
- The measurement vector is written into the measurements file being encoded as an unsigned integer with 16-bits.

The above sequence indicates that the measurement vector is encoded as an unsigned integer, with 16-bits. But in fact, the measurements are not bounded to just positive values. To make the correct binary encoding of the measurement data, using a two's complement, a computation is previously performed. The necessary operation is the addition of 2^{16} to the measurement if it is negative. This is implemented by using the equation 5.1 for $i = 0, \dots, M - 1$.

$$\tilde{y}(i) = \begin{cases} y(i) & \text{if } y(i) \geq 0 \\ y(i) + 2^{16} & \text{if } y(i) < 0 \end{cases} \quad (5.1)$$

At the end of the generation process, two binary files have been created containing all measurement matrices and all measurement samples, respectively. These files are then sent from the computer to the embedded operating system file system to be used by the C program explained previously.

Finally, to ensure the possibility of repeating the generation process with the same results, a seed is used for the random number generator responsible for the creation of the sampling matrices. This means that if this seed is input at the start of the program, the pseudo-random sampling matrices will always follow the same sequence. By employing this approach, a random testing methodology is possible while maintaining the power to repeat a failed test and debug the problems.

5.1.4.2 After the Recovery - Using the Recovered Coefficients

As was referred in the subsection 5.1.3 the recovered coefficients of each set of measurements and sampling matrices are written into an output binary file by the program running on the ARM CPU. This file will contain all of the coefficients needed to assemble the original image, but since this data corresponds to the recovery of each patch, separately, some final processing is needed. To do that, another MATLAB program was developed with the following steps:

1. Read the output binary file transferred from the ZYNQ board;
2. Retrieve a coefficient's patch from the data read, \tilde{x}_i ;
3. Scale the data by a factor of 2^{-8} ;
4. Apply the final 2D-IDCT to acquire the patch \tilde{f}_i from the recovered coefficients;
5. Align the i th patch on the full sized image, adding any superimposed pixels, and increment the weight mask accordingly to the positions that were changed;
6. Repeat from the step 2 until all patches are processed;
7. Compute a division pixel-by-pixel of the the full image by the weight mask;

From the presented algorithm it is noted that a scaling step is taken. This occurs since the the hardware recovery processor follows a 32-bit, $Q16$ number representation meaning that the input measurements, as integers, would be represented on the 16 most significant bits (should be scaled by 2^{16} when transformed from the 16 to the 32 bit representation). If the data representation followed only this rule, the scaling after the recovery should be 2^{-16} , but as previously discussed that is not the case, since to prevent under or overflow the scaling is in fact of 2^8 , which puts the input in the 16 middle bits. For this reason, and given the fact that the data is read from the binary file as 32 bits integers, the MATLAB program should revert an input scaling of 2^8 , resulting in the referred value.

Another important point of this final processing steps is the need for a final 2D-IDCT transformation. Since compressive sensing recovery algorithms have the objective of recovering the coefficients that represent the original signal on the transform domain, where they are sparse, this step is not included on the recovery algorithms formulations. This is one of the foundations of this method, that a signal can be represented by a small amount of coefficients on a transform domain where it is sparse, and as such its objectives are to capture enough data from the sampling to then recover those coefficients. For a compression application this could be even more beneficial since the signal will be already represented by a small amount of non-zero coefficients, but for the sake of verification, the real image must be reconstructed and evaluated leading to this transform step.

Table 5.1: Hardware Implementation Resource Usage : Entire system

Resource	Available	Utilization	Utilization (%)
FF	437200	28428	6.50
LUT	218600	38604	17.65
Memory LUT	70400	1658	2.35
BUFG	32	2	6.25
MMCM	8	1	12.5

5.2 FPGA Implementation - Resource Usage

The design explained in chapter 4 was implemented using the Xilinx Vivado Design Suite which offers a development environment for the creation of applications to run on programmable logic. Furthermore, the development board used in this project, the ZYNQ-7000 SoC ZC706 board was developed by the same company so the software used offered specific tools for the development since it is board-aware. Using the design suite, and given the fact that it is prepared to do the implementation directly for the board in question, a resource usage breakdown of the IHT IP core, as a whole, both in absolute values and in percentage of the total available on the target board is presented in table 5.1.

While the implementation may seem to occupy a small percentage of the board, it is important to note that the development board used offers a large amount of resources, so a better understanding over the usage can be ascertained from an analysis of the absolute numbers. Nevertheless, taking into account that this design implements one recovery processor, it is possible to see that an evolution of this could mean the usage of several recovery cores working in parallel. This would lead to a division of the recovery time, at least the time allocated for the iterative processing, by the number of processors, and to a considerable speed-up. To evaluate the possible number of iterative processors that could be used in the ZYNQ board, a utilization analysis of a single processor without the top module blocks overhead can be seen in table 5.2. From the utilization values obtained, it can be seen that the bottleneck on a parallel system lies on the LUT usage. The number of processors that could be added would round to 4 more, totaling a 5 patches recovery in parallel. This would lead to an expected 5 times speedup on the results obtained for this work.

Since the total utilization values do not give a sufficiently detailed analysis over the implementation results, an analysis of the synthesis and implementation logfiles can be used. From these, a breakdown of the resources used by each implemented block can be found and the ones responsible for the major contribution for the overall resource usage identified. This is shown in table 5.3.

From the analysis of the breakdown table, it is easy to detect that the two most expensive blocks in terms of resource usage are the first multiplication and the threshold blocks. This is easily explained by the fact that the first one implements a tree of adders design and the second a similar tree of comparators. Each 2D transform block represents a weight of close to 14% both in LUTs and FFs, roughly half the usage of each of the previous blocks. Most of the resources used by these blocks are attributed to the transposition and the arithmetic operations needed to

Table 5.2: Hardware Implementation Resource Usage : IHT Co-Processor

Resource	Available	Utilization	Utilization (%)
FF	437200	20868	4.77
LUT	218600	36341	16.68
Memory LUT	70400	830	1.17

achieve a multiplier-less design. Since it employs an accumulator approach the second matrix-vector multiplication block is the most efficient in terms of resource needs. Finally the remaining resources are used by the other operations.

5.3 Results

5.3.1 Image Quality

As presented in subsection 3.3.2, to evaluate the results of image data recoveries, in addition to a visual verification, which is always subjective, two evaluation metrics were used: the PSNR and the SSIM. The first one is used extensively in the literature and as such allows for a better comparison with other results. The second one claims to be better suited for image analysis, given its focus on some specific characteristics that this type of data presents, such as luminosity and structure.

To test the described system, four different images with different resolutions (512x512, 1024x1024, 1920x1080 and 2560x1600) were used, being these dimensions used as part of the name given to them, for an easier comprehension during the analysis. Each image was tested on MATLAB using the methodology described on section 3.3 and on the hardware counterpart using the methodology described in this chapter. Furthermore, to test the impact of stitching, as well as the perception of this artifact as the image dimensions increase, each image was tested for all possible superimposition values: from no superimposition to a superimposition of 7 (from patch to patch, the row or column is incremented only by 1).

For each tested pair of variables (image and superimposition) both metric values were computed, resulting on the elaboration of 4 tables: one for *Lena512x512*, 5.4, one for *Man1024x1024*, 5.5, one for *Lion1920x1080*, 5.6, and finally a fourth one for *Lion2560x1600*, 5.7. In these tables it is easily detected that as the superimposition increases, both metrics get better results, as expected. It is also observed that as the image dimensions increase, so do the metrics values for similar levels of superimposition. A degradation between the MATLAB and hardware results can be detected. This is mainly explained by differences in the data representation, computations and transform precision. The hardware implementation uses fixed-point computations, a 32 bits number representation and employs transforms with a precision of 12 bits, while MATLAB uses floating-point computations, a 64 bits representation and transforms with higher precision.

While the quality metrics values may give some insight, the visual representation of them, as well as the evaluation of the recovered images allow for a better understanding of the results. As

Table 5.3: Block Breakdown of Resource Usage

Block	LUT (% of total used)	FF (% of total used)
2D-DCT	14	14
1st Matrix Vector Multiplication (Adders Tree)	25	23
2nd Matrix Vector Multiplication (Accumulators)	8	10
2D-IDCT	13	14
Threshold	34	23

such, graphs showing the variation of both metrics in relation to the superimposition level used in the recovery of each image are presented in figures 5.3 and 5.4. From analyzing the graph related to the PSNR it is visible a tendency to a linear quality increase as the superimposition increases. It is also detected that the MATLAB implementation achieves better results for the larger image (*Lion2560x1600*) while the hardware recovery of this image is also at the top of the hardware results, even if not of the MATLAB ones. In the case of the graph related to the SSIM, the tendency is more exponential for larger values of superimposition, also being visible a clear better result for the larger image.

Since the amount of images generated in the development of this work is too large to show all in this chapter, they are shown in the appendix A. Nevertheless, a few are shown with the objectives of: comparing the results between MATLAB and hardware implementations of the recovery of the large (4MP) and medium (1MP) images, as well as the impact of superimposition on the perceived visual quality of these images.

From figures 5.5 and 5.6 a visual comparison between the MATLAB recovery and the IHT IP core recovery can be made. In both MATLAB and hardware recoveries it is visible the loss of edge definition, caused mainly by the usage of the DCT transform, which promotes smooth tone transitions. This loss of definition is more visible on the smaller image, since the edges will occur in a lesser number of pixels, which would equate to the need of more coefficients than the defined per patch. While the recovery is good for both cases, there is a loss of dynamic range on the hardware recovery, caused by a number representation with less bits and a transform implementation designed for hardware with only 12 bits of precision. Nevertheless, visually the results are good and the images are recognizable.

To better understand the impact of the superimposition, on figures 5.7 and 5.8 the difference between three lower levels of superimposition (no superimposition, superimposition of 2 and superimposition of 4) for the images previously presented can be seen. On the smaller image the impact is more visible which could be explained by the size of the patch in comparison with the total size of the image, while for the larger one at a superimposition of 4 the results are already good. The level of superimposition will depend on the size of the image and the application for which it will be used.

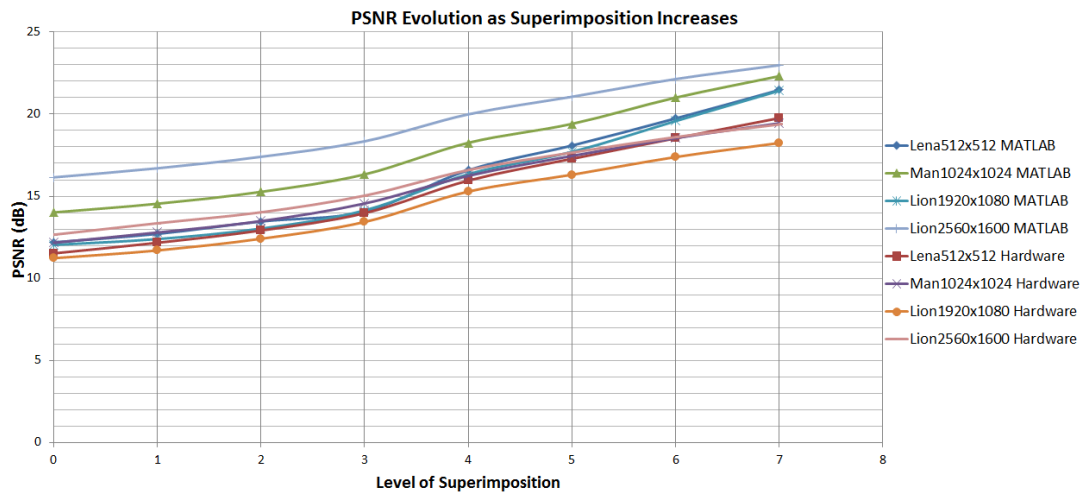


Figure 5.3: PSNR Results for Different Superimpositions

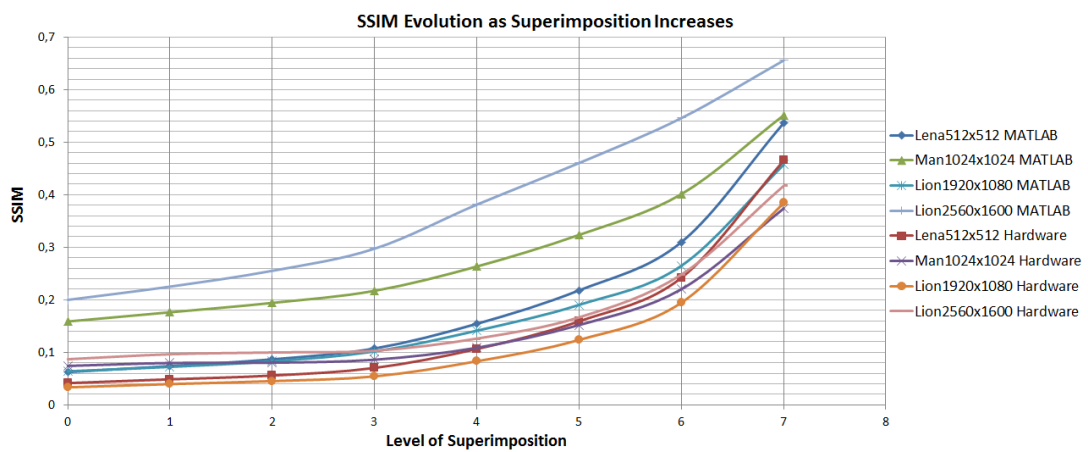


Figure 5.4: SSIM Results for Different Superimpositions

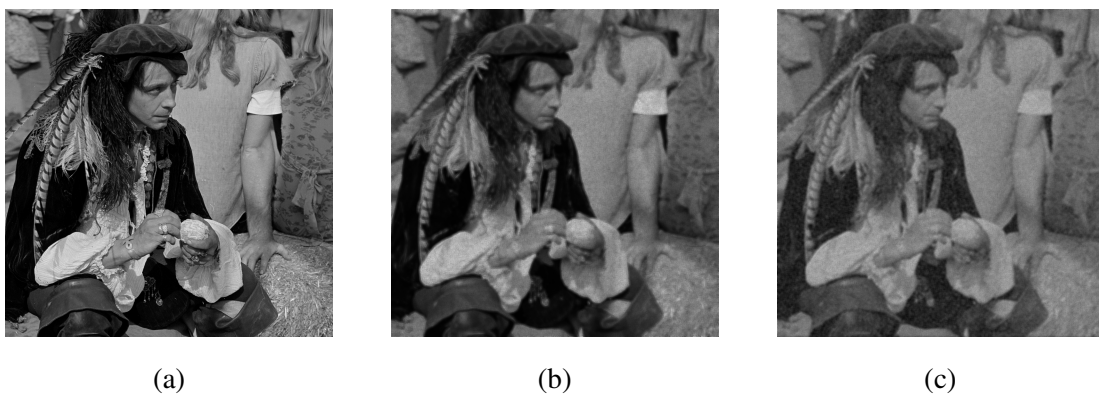


Figure 5.5: Man1024x1024 (1MP): (a) Original; (b) Recovery Result from MATLAB; (c) Recovery Result from Hardware

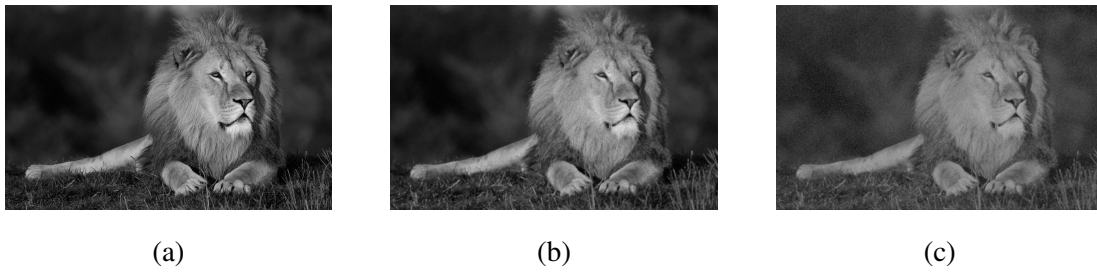


Figure 5.6: Lion2560x1600 (4MP): (a) Original; (b) Recovery Result from MATLAB; (c) Recovery Result from Hardware

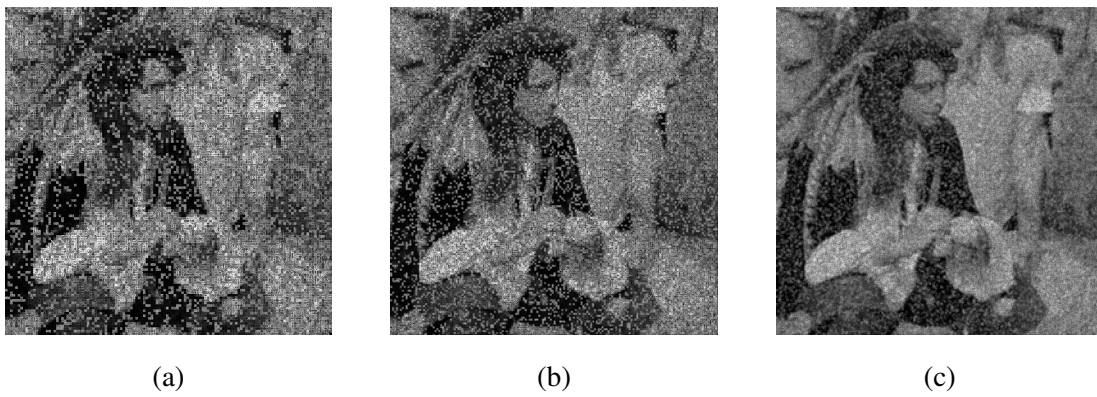


Figure 5.7: Man1024x1024 (1MP): (a) Reconstruction from Patching with no Superimposition; (b) Reconstruction from Patching with 2 Superimposition; (c) Reconstruction from Patching with 4 Superimposition

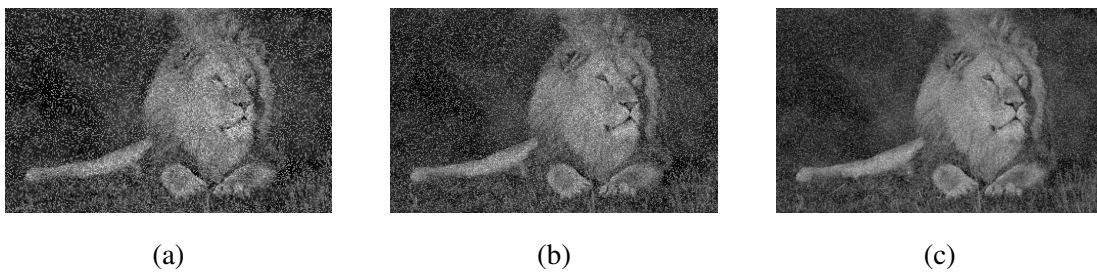


Figure 5.8: Lion2560x1600 (4MP): (a) Reconstruction from Patching with no Superimposition; (b) Reconstruction from Patching with 2 Superimposition; (c) Reconstruction from Patching with 4 Superimposition

Table 5.4: Metrics Results of "Lena512x512"

Superimposition Level	Patch Count	PSNR (dB)		SSIM	
		MATLAB	Hardware	MATLAB	Hardware
0	4096	12.193	11.516	0.062384	0.041404
1	5476	12.706	12.165	0.074020	0.048641
2	7396	13.462	12.915	0.086766	0.055814
3	10609	14.075	13.969	0.107376	0.070307
4	16384	16.589	15.957	0.154351	0.106408
5	29241	18.088	17.282	0.218086	0.159206
6	65536	19.741	18.534	0.309808	0.241938
7	262144	21.472	19.765	0.536658	0.465185

5.3.2 Execution Time

One of the main objectives of this work was the speedup of the recovery process. To evaluate this speedup, the execution times taken by the MATLAB and the hardware implementations to recover all of the patches needed to reconstruct a given image had to be computed. In the case of MATLAB, the time taken to recover the patches was measured by counting the time for each patch between calling the function responsible exclusively for implementing the IHT algorithm implemented in hardware and it returning the recovered patch, and adding all those measurements. In the case of the hardware, the measurement was done by cumulatively adding the time between the command telling the IHT IP core that the load operation was finished and the block communicating that it has ended the recovery.

The IHT IP core was implemented and tested operating at 3 clock frequencies: 50MHz, 75MHz and 100MHz. First, since it is the fastest and the recovery results are identical for all clock frequencies, a comparison between the 100MHz and the MATLAB implementation is done. The MATLAB implementation was executed on a computer with an Intel i5-4570 processor with a clock of 3.2GHz which can be overclocked to 3.6GHz. The results of the measurements are presented in the tables 5.8 and 5.9. In them, it is possible to see the times, in seconds, that each implementation took for each of the superimposition and image pairs. With the measurements taken, a speedup analysis can be made by computing $Speedup = \frac{MATLAB_{delay}}{Hardware_{delay}}$. From the results it is possible to ascertain that the speedup is approximately 190 using only one hardware iterative processor.

The speedup result can be deceiving though, as the desktop CPU and the hardware implementation are running at different clock frequencies. In fact, knowing that the computer CPU achieved a mean clock frequency of 3.4GHz while running the recovery, a more realistic speedup analysis would be done by taking into account the fact that the CPU will run 34 clock cycles for each one that the IHT IP block runs. While this straightforward comparison may not be the most accurate possible, it gives a better insight over the real expected speedup had both implementations run on the same clock frequency. As such, the speedup taking into account the fact that the CPU operates at 34 times the clock frequency of the IHT IP block is approximately 6460. It is important to take into account though, that a MATLAB implementation while easier to achieve in software,

Table 5.5: Metrics Results of "Man1024x1024"

Superimposition Level	Patch Count	PSNR (dB)		SSIM	
		MATLAB	Hardware	MATLAB	Hardware
0	16384	14.017	12.163	0.15880	0.074153
1	21609	14.544	12.791	0.17632	0.079694
2	29241	15.268	13.486	0.19415	0.080306
3	42025	16.335	14.558	0.21715	0.085936
4	65536	18.257	16.246	0.26350	0.108813
5	116964	19.412	17.457	0.32369	0.152056
6	262144	21.010	18.547	0.40122	0.220177
7	1048576	22.319	19.448	0.55152	0.373914

is not the most efficient way of implementing the algorithm in software, being expected that a C implementation offers a faster execution. Nevertheless, the speedup value is such that even a C implementation would either be slower or equivalent to the IHT IP block, while consuming more power to run the CPU.

By doing an analysis of the data using the formula $TimePerPatch = \frac{Time}{NumberPatches}$, it was calculated that the mean time taken to recover a patch is approximately $3.2ms$ for the MATLAB implementation, while in the case of the hardware this value goes down to $16.9\mu s$. The higher precision transform, floating point calculations and number representation using a larger number of bits present in MATLAB also contribute to this difference. It is also important to note that while the maximum and minimum values calculated for the hardware implementation only oscillated by a maximum of 0.17% , on the case of the MATLAB implementation this variation was 6.7% . This result may imply that since the processor running the MATLAB instance is not exclusively allocated for that task, these times can change drastically depending on the system's load at the moment of execution.

Taking into account that the recovery results are identical for all operating clock frequencies that were implemented, tables 5.10 and 5.11 show the effective times of recovery. These results offer a better insight of the overhead of the communication between the ARM CPU and the IHT IP core.

Finally a graph of the recovery speedup between clock frequencies is presented in 5.9. The mean value of the speedup observed between each pair of frequencies was taken from the data presented previously and plotted into a graph also showing the expected speedup values. As can be seen, the real speedup is higher than the expected. This values were obtained from different hardware implementations, so this is the speedup between them, which would lead to conclude that in fact for lower clock frequencies the transactions delays have a much more adverse result on the overall system speed. This is result of the need for the ARM CPU to check if the IHT IP core has finished the recovery before the next patch is loaded, leading to an overhead if the hardware finishes right after the CPU has checked. These results only show that in addition to the expected speedup from having a higher clock frequency, there are some overhead delays that also benefit from this, contributing to the real speedup.

Table 5.6: Metrics Results of "Lion1920x1080"

Superimposition Level	Patch Count	PSNR (dB)		SSIM	
		MATLAB	Hardware	MATLAB	Hardware
0	32400	12.037	11.228	0.063913	0.033291
1	42625	12.390	11.698	0.072338	0.039592
2	57600	13.029	12.410	0.083024	0.045195
3	82944	14.139	13.432	0.101668	0.054225
4	129600	16.347	15.285	0.141285	0.083013
5	230400	17.704	16.311	0.190489	0.123710
6	518400	19.580	17.389	0.264826	0.194538
7	2073600	21.430	18.246	0.457376	0.384458

5.3.3 Power Consumption

Finally, it is important to take notice of the power consumption of the developed hardware. Nowadays there is an ever-increasing search for low power hardware and as such, this is more and more an important factor on the development phase. One of the variables tied to the power consumption is the clock frequency at which a hardware block works. For this matter, a comparison between the different frequencies for which the IHT IP core was synthesized can give some information over the trade-offs between the frequency and power consumption.

Using the Vivado Design Suite reports, it is possible to have an estimate of the power consumption of a given implemented design, so in table 5.12 are shown the estimates for implementations for clock frequencies of 50MHz, 65MHz, 75MHz, 90MHz and 100MHz. The table is divided mainly into two power analysis: the static and the dynamic power consumption, being also presented a total, which is the sum between these two. The dynamic power is further broken down into four of the main contributors of its consumption, but not the only ones (since a simple sum of those does not result in the total dynamic power value). The observed total consumption range between the 2.1 and 2.25 Watts (W). In figure 5.10 a graph is also presented where the dynamic power components as well as the static power consumption are plotted to allow for a better understanding of the results. As expected, the static power consumption remains constant for all frequency values, since it will not depend on this variable, while the dynamic power consumption increases. From the graph, it is apparent that the dynamic power components increase linearly with the frequency.

These results lead to the expected conclusion that the power efficiency can be improved by using lower clock cycles, but at the same time a slower recovery will occur. For this matter, a designer or responsible for integrating the hardware block into a larger system can weigh the advantages and disadvantages and choose which frequency is better suited for the design in question. Nevertheless, it is observed that the power consumption increase between the lower (50MHz) and higher (100MHz) tested frequencies is expected to be of 161mW, only approximately 7% more power for a speedup of 100%, or 2 times faster.

While a GPU-based implementation is not analyzed in this work, a small note regarding power consumption of these may be interesting. Recent commercial grade graphics cards have been

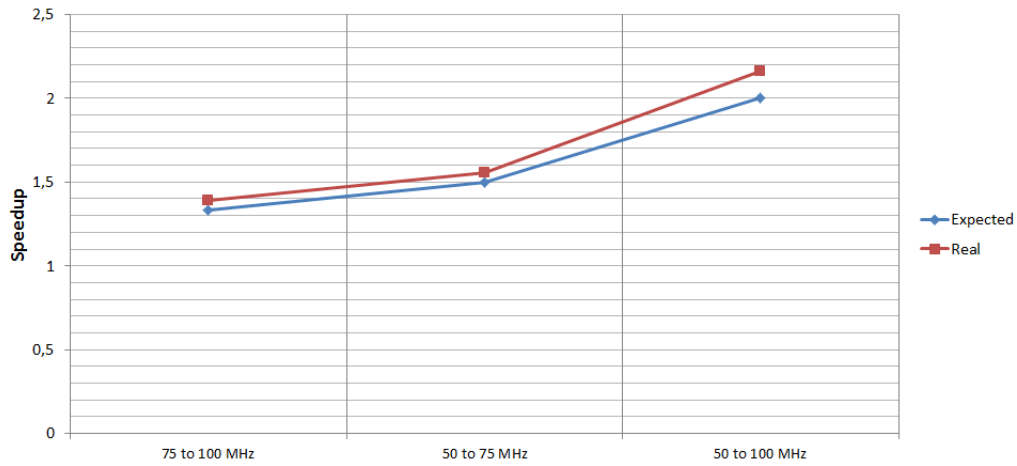


Figure 5.9: Comparison between the expected and observed speedups as the clock frequency increases

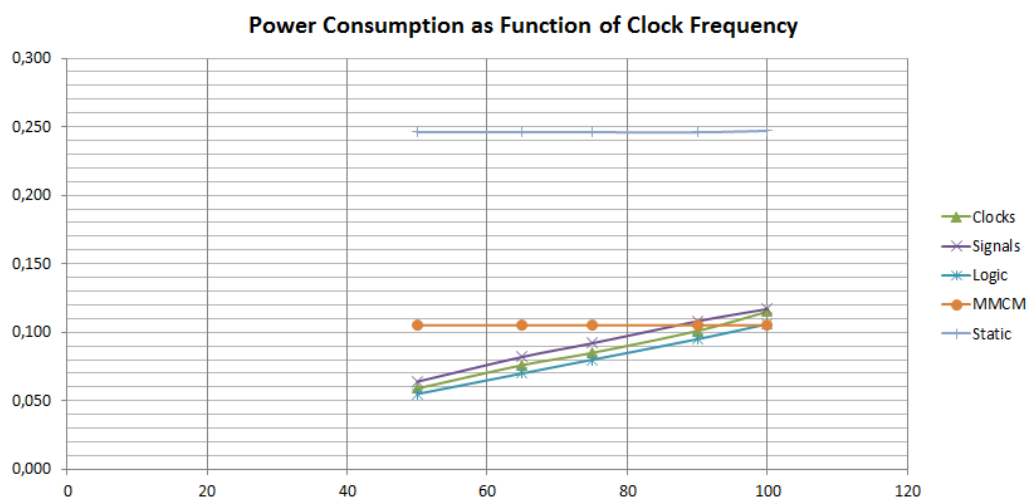


Figure 5.10: Power Consumption Evolution over Frequency Increase

Table 5.7: Metrics Results of "Lion2560x1600"

Superimposition Level	Patch Count	PSNR (dB)		SSIM	
		MATLAB	Hardware	MATLAB	Hardware
0	64000	16.152	12.657	0.199928	0.086941
1	83814	16.706	13.356	0.225126	0.096326
2	114009	17.404	14.023	0.255285	0.099795
3	163840	18.349	15.038	0.297298	0.103031
4	256000	19.995	16.597	0.381280	0.126174
5	456036	21.062	17.662	0.460477	0.166618
6	1024000	22.141	18.607	0.546009	0.248093
7	4096000	22.988	19.377	0.655701	0.417063

aiming for lower power consumption, but these numbers still edge towards the 6W at idle operation, meaning that the graphics card is not running any taxing application, at least for an *NVIDIA Geforce GTX 750 Ti* which is reported as being one of the most power efficient graphics cards available. With this base power consumption, the implementation presented here should already show better power efficiency than a GPU-based implementation running on these graphics cards, since the real operating consumption will be higher than the idle values.

Table 5.8: Recovery Timing Analysis (100MHz): Lena512x512 and Man1024x1024

Superimposition Level	Lena512x512			Man1024x1024		
	MATLAB (s)	Hardware (s)	Speedup	MATLAB (s)	Hardware (s)	Speedup
0	13.618	0.069	197	55.976	0.276	202
1	16.913	0.092	183	70.341	0.364	193
2	24.246	0.125	194	97.216	0.494	197
3	32.301	0.179	181	139.38	0.710	196
4	51.102	0.276	185	211.01	1.106	191
5	98.970	0.493	200	377.05	1.974	191
6	202.84	1.104	188	852.60	4.419	193
7	821.62	4.423	186	3471.1	17.71	196

Table 5.9: Recovery Timing Analysis (100MHz): Lion1920x1080 and Lion2560x1600

Superimposition Level	Lion1920x1080			Lion2560x1600		
	MATLAB (s)	Hardware (s)	Speedup	MATLAB (s)	Hardware (s)	Speedup
0	109.27	0.546	200	196.33	1.081	182
1	141.27	0.718	197	251.10	1.414	178
2	186.28	0.971	192	342.73	1.922	178
3	273.62	1.399	196	532.21	2.764	192
4	418.44	2.185	191	794.51	4.315	184
5	726.93	3.891	187	1368.0	7.696	178
6	1616.8	8.757	185	3499.5	17.25	203
7	6428.2	34.98	184	13505	69.08	195

Table 5.10: Recovery Timing Analysis for Different Clock Frequencies: Lena512x512 and Man1024x1024

Superimposition Level	Lena512x512			Man1024x1024		
	50MHz (s)	75MHz (s)	100MHz (s)	50MHz (s)	75MHz (s)	100MHz (s)
0	0.149	0.0960	0.069	0.597	0.384	0.276
1	0.199	0.128	0.092	0.787	0.506	0.364
2	0.269	0.173	0.125	1.06	0.685	0.494
3	0.387	0.249	0.179	1.53	0.985	0.710
4	0.597	0.384	0.276	2.39	1.54	1.11
5	1.06	0.685	0.493	4.26	2.74	1.97
6	2.39	1.54	1.10	9.55	6.14	4.42
7	9.55	6.14	4.42	38.2	24.6	17.7

Table 5.11: Recovery Timing Analysis for Different Clock Frequencies: Lion1920x1080 and Lion2560x1600

Superimposition Level	Lion1920x1080			Lion2560x1600		
	50MHz (s)	75MHz (s)	100MHz (s)	50MHz (s)	75MHz (s)	100MHz (s)
0	1.18	0.759	0.546	2.33	1.50	1.08
1	1.55	0.999	0.718	3.05	1.96	1.41
2	2.10	1.35	0.971	4.15	2.67	1.92
3	3.02	1.94	1.40	5.97	3.84	2.76
4	4.72	3.04	2.18	9.33	6.00	4.31
5	8.40	5.40	3.89	16.6	10.7	7.70
6	18.9	12.1	8.75	37.3	24.0	17.2
7	75.5	48.6	35.0	149	95.9	69.1

Table 5.12: Power Consumption Analysis

Type	50MHz (W)	65MHz (W)	75MHz (W)	90MHz (W)	100MHz (W)
Total	2.095	2.145	2.174	2.221	2.256
Dynamic	1.849	1.882	1.915	1.971	2.010
Clocks	0.059	0.076	0.085	0.101	0.115
Signals	0.064	0.082	0.092	0.108	0.117
Logic	0.055	0.070	0.080	0.095	0.106
MMCM	0.105	0.105	0.105	0.105	0.105
Static	0.246	0.246	0.246	0.246	0.247

Chapter 6

Conclusions

This work presented a study of the Iterative Hard Thresholding algorithm, one of the several compressive sensing recovery algorithms, and the development and implementation of a hardware co-processor which executes it on an FPGA board. This co-processor was then intended to be used to recover CS-based measurements taken from images. By leveraging on the high level of design liberty offered by the FPGA as a development platform, the implemented IHT IP block achieved a considerable speedup over a software based implementation, while maintaining a low power consumption.

The work was developed in several phases, to allow for a more organic comprehension and problem solving of the overall project. As such, a theoretical background basis was created as a means to better understand the concepts behind compressive sensing and the target recovery algorithm, as well as to gain a better insight over the key points that had to be tackled on the final design. Having a comprehension of the theory, a high level implementation of the algorithm was accomplished using MATLAB. To approach the hardware final design, the software based implementation followed as much as possible the steps and simplifications needed for a hardware design, as well as serving as a testing framework for a modification to the algorithm. With a working software based implementation and a full comprehension over the algorithm's requirements and execution steps, the development of the hardware was the next step. This was the main phase of this work. It involved several steps starting with a high level block architecture design and analysis of the overall data flow that was possible to implement. Having defined a desired data flow between the main blocks of the architecture, each individual block was designed and implemented. This also entailed a research phase, since some of the blocks, such as the transform, were based on work previously done and presented, being modified to suit the target design. After each block was implemented, its functionality was validated and the synthesized result verified. Only after having all of the blocks ready was the overall design implemented, through connections between the blocks and a master control responsible for their operation. Again, this design was functionally verified and its synthesis verified, ending on the creation of the IHT IP block which was ready to be integrated into a fully functional system. With the hardware developed and functional, the final phase was its testing and performance evaluation. To achieve this, several analysis were made,

mainly in terms of the recovered image quality through the evaluation of the PSNR, SSIM and visual comparison between the hardware and software based implementations, run time needed to recover the same amount of patches on the hardware and the software implementations and the power consumption of the hardware implementation based on the clock frequency at which it operates.

The results presented showed that the hardware implementation achieved the proposed objective of recovering image data from CS-based measurements.

Caused by factors such as the number of iterations, sparsity level and the sparsity domain chosen being the 2D-DCT which promotes smooth color transitions, it was observed that there is a loss in edge definition in both the MATLAB and hardware implementations. This loss is less visible as the image size increases, resulting in visually more pleasing reconstructions for images of larger sizes such as 4MP, and as such is expected to have a lesser impact on images of even larger sizes. Between the MATLAB and the hardware implementation there was mainly a loss of color dynamic range, which can be explained mainly by the lower bit representation of the data on the hardware, as well as the less precise transforms employed there.

In terms of the run time efficiency, it is clear that the IHT IP block achieves a recovery speed several orders of magnitude greater than the one observed on the MATLAB implementation. While the software based implementation is done using MATLAB, which is not the most optimized way of achieving a software based implementation, a C approach, for example, would need to obtain a speedup on the order of 6460, considering both implementations running on the same clock frequency, to achieve the same results as the IHT IP block.

The last IHT IP results considered were its power consumption values. Based on the results obtained through the Vivado Design Suite it was shown that the block should operate at power values around the 2.1 – 2.25 Watts, being the most important contributor to this the dynamic power consumption which increased with the clock frequency. While this result may seem high in a *mW* or less era, it is the result of an implementation on an FPGA instead of an ASIC and other power efficient modifications could be made on a design for ASIC purposes. Nevertheless, a comparison with recent commercial grade graphics cards which could be used to achieve a GPU-based implementation of the algorithm shows that with idle consumption values on the order of the 6W for one of the most power efficient boards, the *Geforce GTX 750 Ti*, the implemented hardware block has a good power consumption result.

6.1 Future Work

Having finished the development of the proposed hardware block, some work can be developed in the future to build on these results to either further validate assumptions made during the development or improve the final design.

The implemented design uses a single processing core, meaning that only one set of samples is processed each time. As was discussed when presenting the resource utilization of the design, a parallel implementation of the IHT co-processor would lead to an expected speedup equal to

the number of processors working in parallel. A modification of the top module could allow this parallelization to happen and the speedup could be evaluated. Furthermore, a multiplexing technique could be used where some of the main resource heavy blocks were used by several iterative processors in different time slots allowing for further parallelization with less resource usage.

The chosen transform, while adequate has a poor performance for lower resolution images. With this in mind a different transform or a better precision of the 2D-DCT and 2D-IDCT should be tested to evaluate the trade-offs of changing on the recovery for the first case and on the resource usage and design complexity for the second.

To improve the compression ratio, a larger patch could be used. This alteration would lead to the modification of the whole datapath to accommodate a larger vector, as well as the design of the transform and the size of each of the architecture blocks. In addition to potentially increasing the compression ratio, the usage of a larger patch could also lead to the usage of a higher number of coefficients and a better image quality result.

Finally, on the development of the work, a step computation was needed to ensure the convergence of the algorithm. While the proposed solution yielded good results, a better theoretical analysis of it is needed to ensure the generality of this method. Furthermore, other step computation approaches could be analyzed and tested to achieve a better convergence.

The final product of this work achieves all of the objectives set at the beginning of its development. Nevertheless, as with most development projects, there are always several possible tests, improvements and changes that can be made with the objective of either achieve better results or further validate the already accomplished design.

Appendix A

Image Recovery Results

Given the large amount of images obtained in the process of testing the hardware implementation of the IHT algorithm, those had to be presented in this appendix. For each one of the four base images tested, results for all levels of superimposition are shown. It is visible that for the smaller image the lower superimposition has an extremely large effect, while for the final, 4MP image, this is not the case. While an analysis of both images at the same pixel size could lead to the same visual quality, the fact that larger images are almost never shown at their native resolution on the display or printed result leads to a visually more pleasing result with lesser or no superimposition.

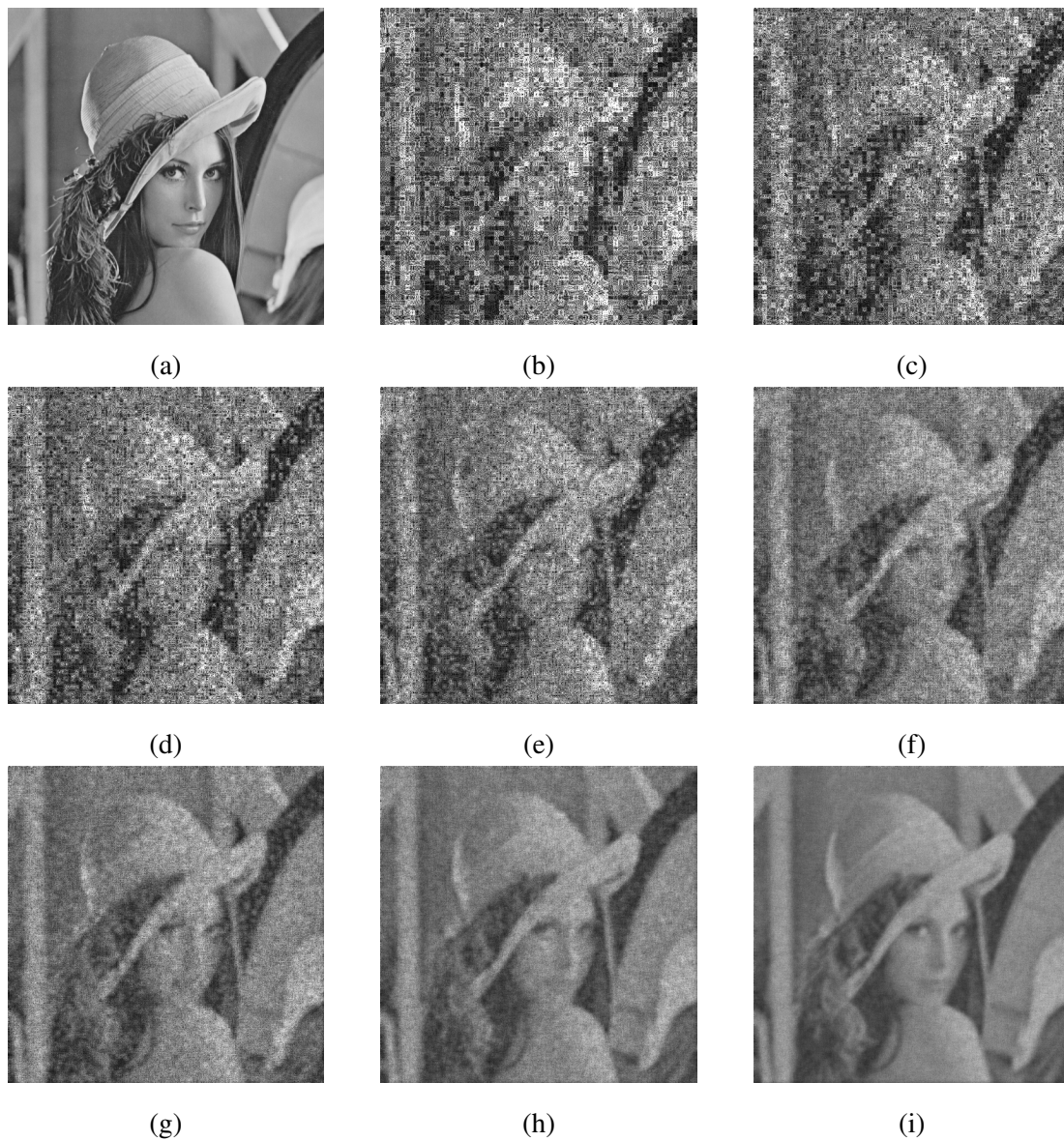


Figure A.1: Lena512x512 (0.26MP) Hardware Results: (a) Original; (b) Superimposition of 0; (c) Superimposition of 1; (d) Superimposition of 2; (e) Superimposition of 3; (f) Superimposition of 4; (g) Superimposition of 5; (h) Superimposition of 6; (i) Superimposition of 7

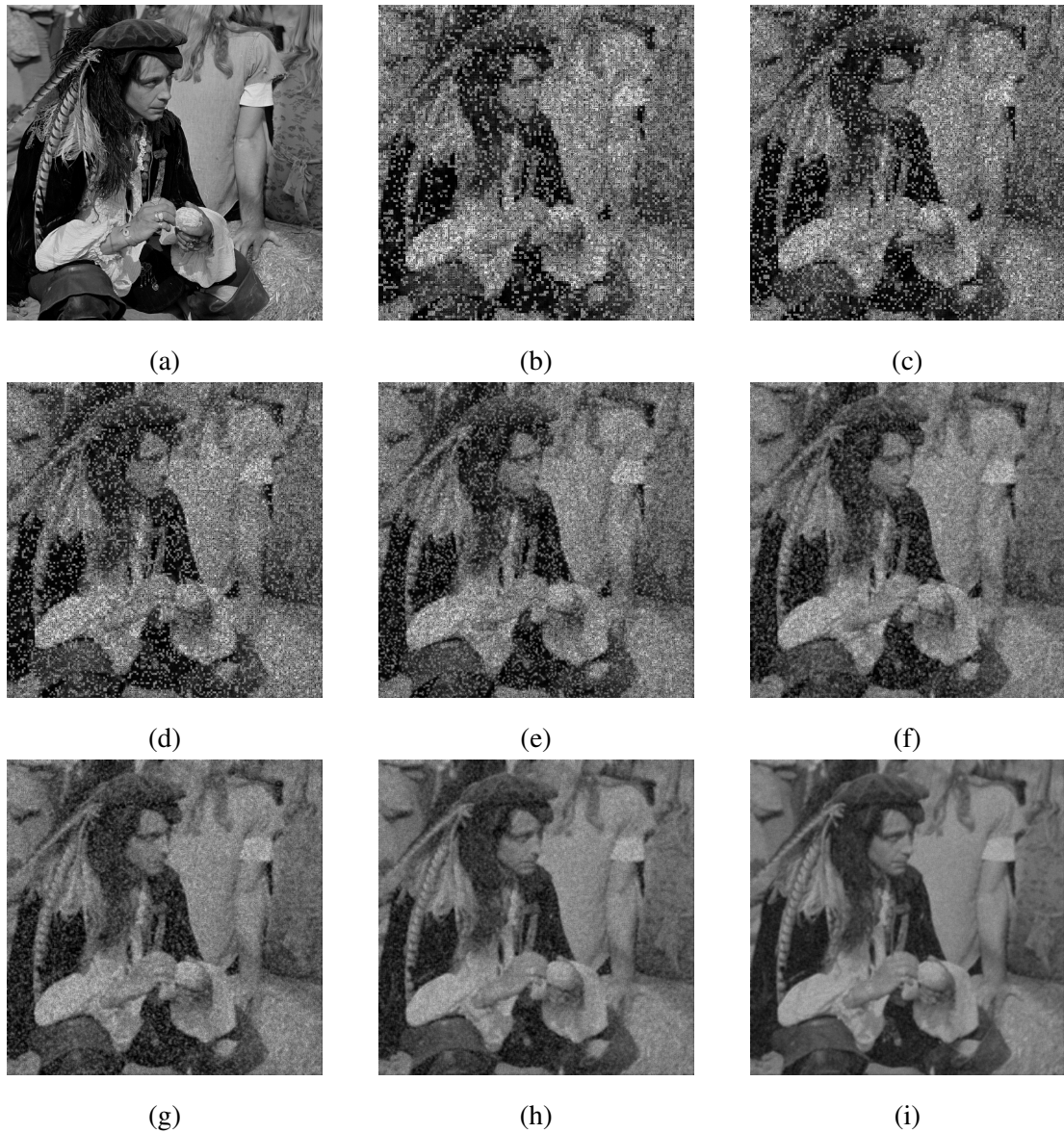


Figure A.2: Man1024x1024 (1MP) Hardware Results: (a) Original; (b) Superimposition of 0; (c) Superimposition of 1; (d) Superimposition of 2; (e) Superimposition of 3; (f) Superimposition of 4; (g) Superimposition of 5; (h) Superimposition of 6; (i) Superimposition of 7

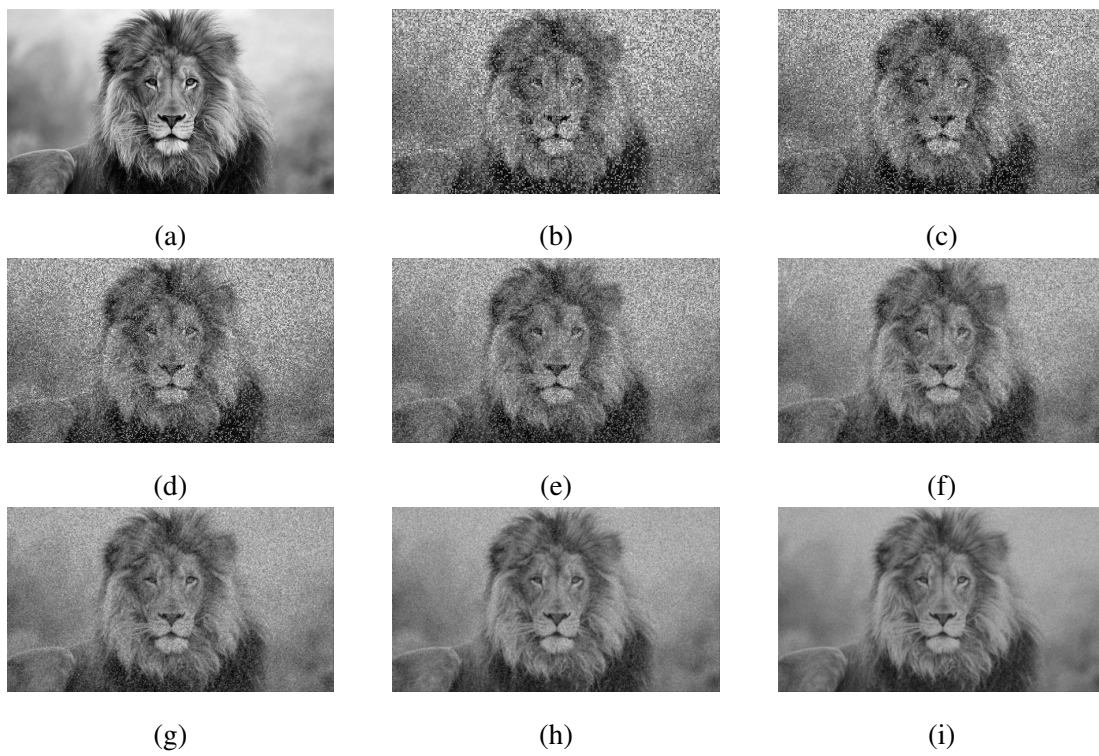


Figure A.3: Lion1920x1080 (2MP) Hardware Results: (a) Original; (b) Superimposition of 0; (c) Superimposition of 1; (d) Superimposition of 2; (e) Superimposition of 3; (f) Superimposition of 4; (g) Superimposition of 5; (h) Superimposition of 6; (i) Superimposition of 7

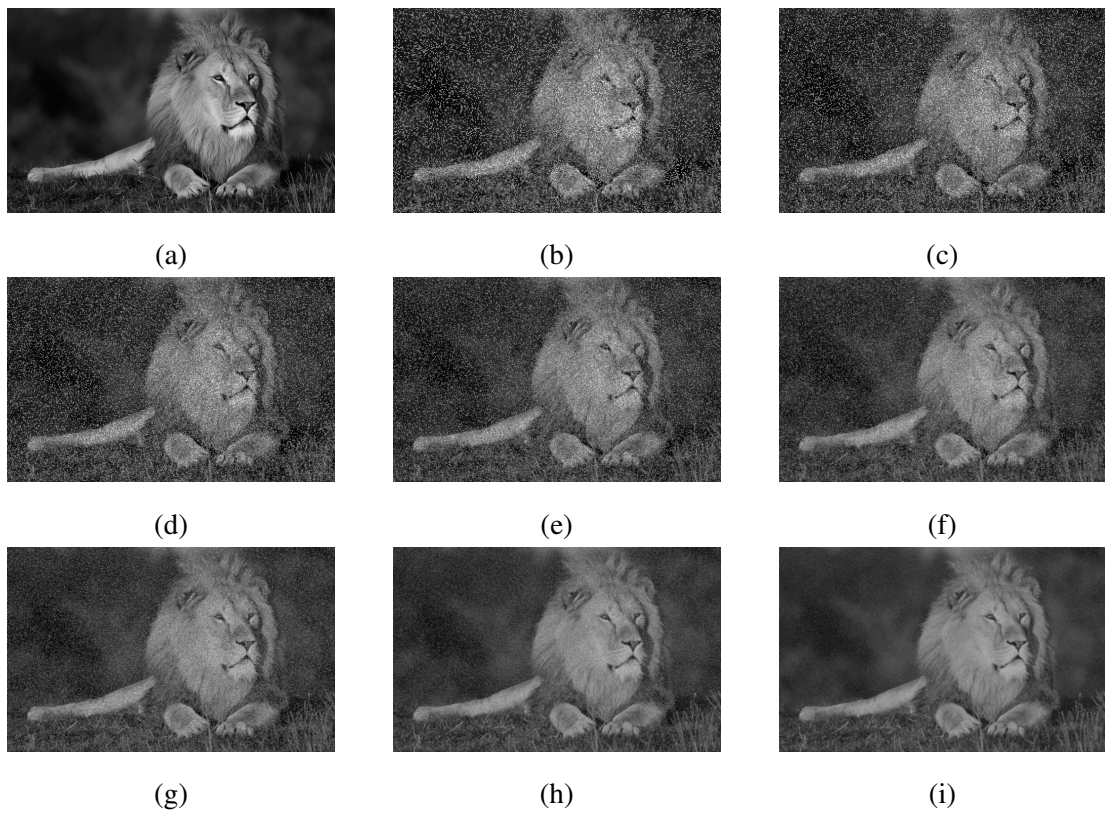


Figure A.4: Lion2560x1600 (4MP) Hardware Results: (a) Original; (b) Superimposition of 0; (c) Superimposition of 1; (d) Superimposition of 2; (e) Superimposition of 3; (f) Superimposition of 4; (g) Superimposition of 5; (h) Superimposition of 6; (i) Superimposition of 7

References

- [1] C E Shannon. *Communication In The Presence Of Noise*, 1998.
- [2] D L Donoho. *Compressed sensing*, 2006.
- [3] E J Candes, J Romberg, and T Tao. *Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information*, 2006.
- [4] Fred Chen. *Energy-efficient wireless sensors: fewer bits, Moore MEMS*. PhD thesis, Massachusetts Institute of Technology, 2011.
- [5] E J Candes and M B Wakin. *An Introduction To Compressive Sampling*, 2008.
- [6] J A Tropp and A C Gilbert. *Signal Recovery From Random Measurements Via Orthogonal Matching Pursuit*, 2007.
- [7] E J Candes and T Tao. *Near-Optimal Signal Recovery From Random Projections: Universal Encoding Strategies?*, 2006.
- [8] Shanmugavelayutham Muthukrishnan. *Data streams: Algorithms and applications*. Now Publishers Inc, 2005.
- [9] J A Tropp and S J Wright. *Computational Methods for Sparse Solution of Linear Inverse Problems*, 2010.
- [10] S G Mallat and Z Zhang. *Matching pursuits with time-frequency dictionaries*, 1993.
- [11] Y C Pati, R Rezaifar, and P S Krishnaprasad. *Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition*, 1993.
- [12] Geoff Davis, Stephane Mallat, and Marco Avellaneda. *Adaptive greedy approximations*. *Constructive approximation*, 13(1):57–98, 1997.
- [13] Deanna Needell and Joel A Tropp. *Cosamp: iterative signal recovery from incomplete and inaccurate samples*. *Communications of the ACM*, 53(12):93–100, 2010.
- [14] Y Gaur and V K Chakka. *Performance Comparison of OMP and CoSaMP Based Channel Estimation in AF-TWRN Scenario*, 2012.
- [15] Thomas Blumensath and Mike E Davies. *Iterative thresholding for sparse approximations*. *Journal of Fourier Analysis and Applications*, 14(5-6):629–654, 2008.
- [16] Thomas Blumensath and Mike E Davies. *Iterative hard thresholding for compressed sensing*. *Applied and Computational Harmonic Analysis*, 27(3):265–274, November 2009.

- [17] E Ollila, Hyon-Jung Kim, and V Koivunen. Robust iterative hard thresholding for compressed sensing, 2014.
- [18] T Blumensath and M E Davies. Normalized Iterative Hard Thresholding: Guaranteed Stability and Performance, 2010.
- [19] Thomas Blumensath. Accelerated iterative hard thresholding. *Signal Processing*, 92(3):752–756, March 2012.
- [20] Shaobing Chen and David Donoho. Basis pursuit. In *Signals, Systems and Computers, 1994. 1994 Conference Record of the Twenty-Eighth Asilomar Conference on*, volume 1, pages 41–44. IEEE, 1994.
- [21] Scott Shaobing Chen, David L Donoho, and Michael A Saunders. Atomic decomposition by basis pursuit. *SIAM journal on scientific computing*, 20(1):33–61, 1998.
- [22] Yong Fang, Liang Chen, Jiaji Wu, and Bormin Huang. GPU Implementation of Orthogonal Matching Pursuit for Compressive Sensing, 2011.
- [23] P Sattigeri, J J Thiagarajan, K N Ramamurthy, and A Spanias. Implementation of a fast image coding and retrieval system using a GPU, 2012.
- [24] Ron Rubinstein, Michael Zibulevsky, and Michael Elad. Efficient implementation of the K-SVD algorithm using batch orthogonal matching pursuit. *CS Technion*, 40(8):1–15, 2008.
- [25] Jared Blanchard, Jeffrey D. and Tanner. GPU accelerated greedy algorithms for compressed sensing. *Mathematical Programming Computation*, 5:267–304, 2013.
- [26] Lin Bai, P Maechler, M Muehlberghuber, and H Kaeslin. High-speed compressed sensing reconstruction on FPGA using OMP and AMP, 2012.
- [27] Fengbo Ren, R Dorrace, Wenyao Xu, and D Markovic. A single-precision compressive sensing signal reconstruction engine on FPGAs, 2013.
- [28] J.L.V.M. Stanislaus and T Mohsenin. Low-complexity FPGA implementation of compressive sensing reconstruction, 2013.
- [29] J.L.V.M. Stanislaus and T Mohsenin. High performance compressive sensing reconstruction hardware with QRD process, 2012.
- [30] H Rabah, A Amira, B K Mohanty, S Almaadeed, and P K Meher. FPGA Implementation of Orthogonal Matching Pursuit for Compressive Sensing Reconstruction, 2014.
- [31] Jingwei Xu, E Rohani, M Rahman, and Gwan Choi. Signal reconstruction processor design for compressive sensing, 2014.
- [32] R Szeliski. *Computer Vision: Algorithms and Applications*. Texts in Computer Science. Springer, 2010.
- [33] E R Fossum. CMOS image sensors: electronic camera-on-a-chip, 1997.
- [34] K R Rao and P Yip. *Discrete Cosine Transform: Algorithms, Advantages, Applications*. Elsevier Science, 1990.
- [35] I Daubechies. The wavelet transform, time-frequency localization and signal analysis, 1990.

- [36] O Rioul and M Vetterli. Wavelets and signal processing, 1991.
- [37] Zhou Wang, A C Bovik, H R Sheikh, and E P Simoncelli. Image quality assessment: from error visibility to structural similarity, 2004.
- [38] C Loeffler, A Ligtenberg, and George S Moschytz. Practical fast 1-D DCT algorithms with 11 multiplications, 1989.
- [39] Zhang-jin Chen and Zhi-gao Zhang. A High-Speed 2-D IDCT Processor for Image/Video Decoding, 2009.
- [40] M El Aakif, S Belkouch, and M M Hassani. An efficient pipelined fast and multiplier-less 2-D IDCT for image/video decoding, 2011.
- [41] M El Aakif, S Belkouch, N Chabini, and M M Hassani. Low power and fast DCT architecture using multiplier-less method, 2011.
- [42] A Aggoun and I Jalloh. Two-dimensional DCT/IDCT architecture, 2003.
- [43] H Hussain, K Benkrid, Chuan Hong, and H Seker. An adaptive FPGA implementation of multi-core K-nearest neighbour ensemble classifier using dynamic partial reconfiguration, 2012.