Carlos Augusto Costa Garcia

Licenciado em Ciências da Engenharia Electrotécnica e de Computadores

Land Cover Classification Implemented in FPGA

Dissertação para obtenção do Grau de Mestre em

Engenharia Electrotécnica e de Computadores

Orientador: Rui Manuel Leitão Santos-Tavares,
Auxiliary Professor, FCT-UNL

Júri

Presidente: Professor Auxiliar Luís Augusto Bica Gomes de Oliveira, FCT-UNL
Arguente: Professora Auxiliar Anikó Katalin da Costa, FCT-UNL
Vogal: Professor Auxiliar Rui Manuel Leitão Santos-Tavares, FCT-UNL

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2019

Land Cover Classification Implemented in FPGA

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

"If you can't fly then run, if you can't run then walk, if you can't walk then crawl, but whatever you do you have to keep moving forward."

Martin Luther King Jr.

Acknowledgements

I would like to express my thanks to "Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa (FCT-UNL)"for these past 5 year. 5 years where I've learned so much and it has been a second house.

To my advisor, Professor Rui Santos-Tavares, for giving me the opportunity to accomplish this work, contributing to my academic training, even more in the image processing area which i like so much. Also, I would like to express my gratitude to Professors Luís Bica Oliveira and João Pedro Oliveira for the support with the hardware required for the task and Professors Anikó Costa, Filipe Moutinho and José Fonseca for the guide lines and always being available to clear issues and doubts.

To my family, who had the most patience with my time away and gave me all the support to get up every morning to continue the job. Thank you for all you gave me, not just for this dissertation but for teaching me to become who I am today. Above all, I wouldn't have ended my course without them. All the love a family like mine can give, words cannot express.

This house gave me a lot of good friends who I'll make sure to keep close. They where always there for when needed and have traveled this journey too. 5 years don't sum up what friends like these will experience. For friends who have been in my life longer than these 5 years, Mafalda, Filipe, Márcio and Duarte, who gave a slap on my back every time I needed consolation, please, let's keep this incredible friendship going.

For the person who shines in my life everyday, who accepts my stubborn, who chew me out and always corrects me, Beatriz, thank you with all my heart, I will always hope to give you back what you already gave me. You were there every single day from begin to end, you made me find my way when I was mostly lost, giving me the strength to keep on, again, thank you.

Thank you all.

Abstract

The main focus of the dissertation is Land Use/Land Cover Classification, implemented in FPGA, taking advantage of its parallelism, improving time between mathematical operations. The classifiers implemented will be Decision Tree and Minimum Distance reviewed in State of the Art Chapter. The results obtained pretend to contribute in fire prevention and fire combat, due to the information they extract about the fields where the implementation is applied to.

The region of interest will Sado estuary, with future application to Mação, Santarém, inserted in FORESTER project, that had a lot of its area burnt in 2017 fires. Also, the data acquired from the implementation can help to update the previous land classification of the region.

Image processing can be performed in a variety of platforms, such as CPU, GPU and FPGAs, with different advantages and disadvantages for each one. Image processing can be referred as massive data processing data in a visual context, due to its large amount of information per photo.

Several studies had been made in accelerate classification techniques in hardware, but not so many have been applied in the same context of this dissertation. The outcome of this work shows the advantages of high data processing in hardware, in time and accuracy aspects.

How the classifiers handle the region of study and can right classify it will be seen in this dissertation and the major advantages of accelerating some parts or the full classifier in hardware. The results of implementing the classifiers in hardware, done in the Zynq UltraScale+ MPSoC board, will be compared against the equivalent CPU implementation.

Keywords: Accuracy, Performance, Land Use/Land Cover Classifier, CPU, GPU, FPGA, Zynq UltraScale+ MPSoC.

Resumo

O principal foco da dissertação é a Classificação de Terrenos, implementada em FPGA, tirando vantagem do seu paralelismo, melhorando o tempo de execução entre operações matemáticas. Os classificadores implementados são Árvores de Decisão e Distância Mínima, analizados no capítulo do Estado da Arte. Os resultados obtidos pretendem contribuir na prevenção e combate de incêndios com base na informação que o classificador retira dos terrenos em que é aplicado.

A região a ser estudada é o estuário do Sado com futura aplicação para a zona de Mação, em Santarém, inserido no projeto FORESTER, que teve uma vasta área queimada devido aos incêndios de 2017. A informação proveniente da implementação deve também atualizar a presente na região de estudo.

Processamento de imagem pode ser desenvolvido sobre diversas plataformas, entre elas, CPU, GPU e FPGAs, com diferentes vantagens e desvantagens aplicadas aos mesmos. Podemo-nos referir a processamento de imagem como o tratamento de grandes quantidades de informação, aplicado a um contexto visual.

Diversos estudos foram feitos sobre o acelerar classificadores em *hardware*, mas poucos no mesmo contexto que esta dissertação. Este trabalho pretende demonstrar as vantagens de processar grandes quantidades de informação em *hardware*, tanto em tempo de processamento e precisão de resultados.

Como os classificadores conseguem tratar a região de estudo, assim como a precisão na sua classificação é revista nesta dissertação e as vantagens em acelerar parte ou totalmente um classificador em *hardware*. Os resultados dos classificadores implementados em *hardware*, mais concretamente na plataforma Zynq UltraScale+ MPSoC, vão ser comparados com uma implementação equivalente em CPU.

Palavras-chave: Precisão, Desempenho, Classificação de Terrenos, CPU, GPU, FPGA, Zynq UltraScale+ MPSoC.

Contents

List of Figures

List of Tables

Listings

Acronyms

AMBA	Advanced Microcontroller Bus Architecture.
ANN	Artificial Neural Network.
ARM	Advanced RISC Machines.
AXI	Advanced eXtensible Interface.
CC	Correlation Coefficient.
CIE	International Commission on Illumination.
CNN	Convolutional Neural Network.
COS	Carta de Uso e Ocupação do Solo de Portugal Continental.
CPU	Central Processing Unit.
DMA	Direct Memory Access.
DT	Decision Trees.
DUT	Design Under Test.
DWT	Discrete Wavelet Transform.
ESTARFM	Enhanced Spatial and Temporal Adaptive Reflectance Fusion Model.
FF	Fuzzy-Fusion.
FIMICA	Fixed Identity Monotonic Identity Commutative Aggragation.
FLAASH	Fast Line-of-Sight Atmospheric Analysis of Spectral Hypercubes.
FPGA	Field-Programmable Gate Array.
GAN	Generative Adversarial Network.
GLCM	Gray-Level Co-occurrence Matrix.
GPGPU	General-Purpose Graphic Processing Unit.
GPU	Graphic Processing Unit.
HDL	Hardware Description Language.

IBM	International Business Machines.
IP	Intellectual Property.
KNN	K-Nearest Neighbors.
LiDAR	Light Detection and Ranging.
LUT	Lookup Table.
MLC	Maximum Likelihood Classifier.
MPPA	Massively Parallel Processor Arrays.
mtry	number of variables in the random sampling.
NDSM	Normalized Digital Surface Model.
NIR	Near Infrared.
NMSE	Normalized Mean Square Error.
ntree	number of decision trees.
NVDI	Normalized Difference Vegetation Index.
OOB	Out-of-Bag.
PL	Programmable Logic.
PS	Processing System.
PSNR	Peak Signal-to-Noise Ratio.
QUAC	Quick Atmospheric Correction.
RGB	Red, Green and Blue.
ROI	Region of Interest.
SAR	Synthetic Aperture Radar.
SDK	Software Development Kit.
SIMD	Single Instructions Multiple Data.
SoC	System on a Chip.
STMRF	Spatio-Temporal Markov Random Fields.
SVM	Support Vector Machine.
SWIR	Shortwave Infrared.

tiff Tagged Image File Format.

UIQI Universal Image Quality Index.

VI Vegetation Index.

WFU Wildland Fire Use.
WLR Weighted Linear Regression.

Chapter 1

Introduction

This is an introductory chapter to contextualize the reader of the presented dissertation. In section 1.1 are presented context and motivation of the work being developed. In section 1.2 the problem being addressed with the proposed solution. At least, in section 1.3, a summary of the remaining chapters.

1.1 Context and Motivation

Fires have concerned the general public because of their highly presence in previous years, not just in Portugal, but in the rest of the world too. Their enormous devastation power destroys forests, cultivation lands and even buildings that become on their way. Even though all the effort by wildfire prevention programs, such as Wildland Fire Use (WFU), to control and put a stop into them, all the help from human and technology source is welcomed.

From the technology perspective, this dissertation pretends to implement a way to reduce the devastation power, minimize damages or prevent the start of a fire. The approach will be a Land Use/Land Cover Classification technique to label the region of study into different classes. The results obtained can be further used in a fire sensor/algorithm to, for instance, a dry land where fires have higher chance to occur, to be cleared, or in case of an existing fire, to predict the direction where it may go. The Land Use/Land Cover Classification technique will be performed using spatial imagery.

Image processing has been a major interest among developers since its potentials in diverse areas, the most common being automatic reading of printed and handwritten text, high energy physics, cytology, medical diagnosis, analysis of biomedical images and signals, remote sensing, industrial applications, identification of human faces profiles, fingerprints and speech recognition, detection of resources from earth and automatic classification of terrain [1]. This last topic mentioned, automatic classification of terrain, will be the main focus of the present dissertation.

In image processing there are two main problems that are commonly highlighted, the accuracy in decision making and the fast implementation [2]. It's important to understand the trade-offs of each one and what is most crucial to the problem being approached. The fast response time became a requirement in the real-time system world, as systems became larger and more complex over time, with most usage in human and robot interaction and imaging hardware [3]. In this case of study, it's prioritized a higher reliability of the algorithms implemented, compromising the execution speed and response time. This decision is taken because the algorithms are intended to be accelerated in hardware platforms for the reasons that will be explained in chapter 2.

As mentioned before, image processing will be applied to terrain classification. Several studies have been made on this subject, which will be reviewed latter on. This theme has been intensely explored for its highly versatile applications. Humans use it for social-economic proposes in which the map generated by the system helps in the conservation planning of the location in analysis and in researching the intensity of human activity [4]. Also, the data extracted from the studies give a lot of support in "*comprehension and analysis of natural phenomena such as climate change; provide a means to assess carbon stock accountability; and help monitor agriculture development, disaster management, land planning, defense of biodiversity, etc*" [5].

To implement the algorithms required for image classification, several platforms can be used. Which one to choose depends on the approach been taken and what are the main system requirements. Also, it's interesting to see, how a not so conventional platform, Field-Programmable Gate Array (FPGA), can present equivalent or better performance than other platforms such as Central Processing Unit (CPU) and Graphic Processing Unit (GPU).

In chapter 2 will be presented what methods are already being applied to Land Cover Classification, pros and cons of them and the comparison of different platforms to run the algorithms, such as CPU, GPU and FPGA.

1.2 Problem and Proposed Solution

This work will study the region of Sado estuary with the validation framework residing in itself. It's the intention of future application for the classifiers to be applied in Mação, Santarém district. This choice is made due to Sado estuary regions allocate more diversity of Land Cover classes as well as more water information in Ocean, River Banks and Humid Regions. The scope of this work belongs to the project FORESTER, due to the forest fires of August, 2017, the region had a vast of its area burnt and land data for the village is no longer accurate.

The propose of this dissertation is to re-qualify and update the data using the above mentioned methods with further explanation in chapter 2. The goal is to implement an algorithm able to correct qualify the land with the possibility to be used not just for the region of study but also in other scenarios. As mentioned, the results of this land classification have the intention to be used in future work to prevent a real fire scenario, before, during and after the event. The results should provide information about the area to cooperate in prevention, putting out and recovering.

The implementation will be performed in hardware because of its low power consumption and major capabilities in high processing imagery data. The classifiers to be run are Decision Tree and Minimum Distance. Due to their high accuracy and versatility (see 2.1.1), these algorithms should perform well in this context.

1.3 Outline

The following document is structured as follows:

Chapter 2 - Presents the algorithms already being used in Land Cover Classification with comparison between them with the scenarios where they are applied; also a comparison between platforms that are able to run such algorithms, first a more abstract point of view where the basic performance power of the platforms is analysed and then, a more contextualized approach for image classification; in the end, was chosen two algorithms, which will be implemented in the hardware, the literature reviewed has the objective to check the reliability of the implementation.

Chapter 3 - Describes the platform that will be used to develop the work; also the softwares required to support platform programming with the classifier. In the last section is presented the preprocessing operation done to the data so that it could be used in hardware.

Chapter 4 - In this chapter are described in detail the algorithms implemented for Land Use/Land Cover classification developed in software as well as in hardware. The two implementations are analyzed in detail with its correspondent models.

Chapter 5 - Presents the results obtained for the implementations. It compares the two classifiers performed in software and hardware in terms of performance time and accuracy.

Chapter 6 - This chapter concludes the work with what achievements have been made and future work to improve not just accuracy of the classifier but also the time to perform the classification.

Chapter 2

State-of-Art

This chapter main focus is to present what methods and platforms are being used to help in Land Use/Land Cover Classification. Divided in three sections, first, in section 2.1, different methods and mathematical probabilistic systems are shown to accomplish the highest accuracy possible. As will be noticed, no algorithm is perfect and there are different scenarios where one or another algorithm should be used, cause it presents better results for the case of study. Secondly, in section 2.2, a comparison of performance between CPU, GPU and FPGA analysing the speed and processing power on them with special attention to image processing. And thirdly, in section 2.3, it's discussed the possibility of implementing Land Use/Land Cover classification algorithms in hardware taking the benefits over software, what classifiers to implement and what platform will be used.

2.1 Land Use/Land Cover Classification

When considering Land Use/Land Cover Classification, it's important to understand the effects of a good algorithm and the platform where it will be performed. How significant is the platform is described in more detail in section 2.2. This section focuses on what is considered a good algorithm. In subsection 2.1.1 are presented what algorithms are nowadays being used in Land Use/Land Cover Classification and in subsection 2.1.2, some methodologies that improve land cover classification, such as removing environmental phenomena.

2.1.1 Classification Algorithms

For the problem being addressed, processing speed or accuracy should take more significance, rather than other factors, to accomplish the requirements of the system. Next will be presented several approaches in Land Use/Land Cover Classification, more specifically, different algorithms. When considering an implementation of a classifier, it's important to know if it will be a supervised or unsupervised classification. In supervised classification, the user has a training data set, in which pixel values/classes are known, then the computer algorithm labels the pixels to the class with highest probability of membership [6, 7]. In unsupervised classification, no training data set is required, the user defines the number of classes that should be presented, and rules based on clustering algorithms group the data into the classes. Although this implementation is faster than the previous one, its normal accuracy is less than the supervised classification [7, 8]. The cross validation between the image classified and ground truth knowledge is usually made to calculate the accuracy. The most worldwide used spatial sensors to provide high resolution satellite imagery data are RapidEye, Worldview-2, GeoEye, GF-2 and Landsat 8 [9, 10]. In table 2.1 are presented the supervised and unsupervised classification implementations, found in the reviewed literature, with a brief explanation of each one.

Table 2.1: Land Cover Classification Algorithms studied in the literature reviewed.

	Algorithms	Explanation
Supervised	UNINORM	The algorithm implements a supervised system using an inference scheme, based on the rules applied, the output of each one represents the most likely class to be assigned [11].
	Decision Tree	It's a tree-like model, built upon conditional statement nodes, as deep as it is navigated in the tree, smaller subsets are found, the tree ends when the subsets are homogeneous [11].
	Artificial Neural Network	It learns by analysing examples in the training data, the goal is to detect patterns, based on previous learning it classifies de validation data sets [11].
	Maximum Likelihood Classifier	The goal is to find the parameter that maximizes the likelihood function, in other words, in a scenario with more than one signature, it tries to assign the classifying object to the highest probability class [12].
	Minimum Distance	A distance is defined as an index of similarity so that the minimum distance is identified as the maximum similarity [13].
	Mahalanobis Distance	The algorithm assumes that the histogram of the band has a normal distribution. It measures the distance between a point P and a distribution D, this distance is zero if P is at the mean of D and grows as P moves away from the mean [14].
	K-Nearest Neighbor	The algorithm assigns the most common value of the *k* nearest neighbors to the object being classified [15].
	Parallelepiped	The algorithm uses a simple decision rule, the decision boundaries form an *n* dimensional parallepiped classification in the image data space. Based on thresholds, the object is assigned to a class [16].

	Support Vector Machine	It's a non-probabilistic binary classifier, the goal of this method is to find an hiperplane in an *n* dimensional space, dividing the categories with the widest gap as possible [17].
	Random Forest	Is a meta estimator that fits a number of decision tree classifiers on various sub samples of the 1 data set and uses averaging to improve the predicative [18].
	Pixel-Based	It evaluates each individual pixel based on its spectral information [19].
	Object-Based	Groups the pixels that have similar properties according to their spectral properties [19].
Unsupervised	K-Means Clustering	It's objective is to find *k* groups known *a priori*, the algorithm focuses in minimize the clustering variability [20].
	ISO Data	The principle is the same as K-Means Clustering, but it allows different numbers of clusters to be generated.

Like no photo and/or classification method is perfect, it can be also observed in the next literature reviewed, that to choose a land cover class, it is used the method with the smallest error or highest accuracy. To decrease the error, is also recommended the highest resolution image possible. Obviously it comes with a cost, so a resolution of 10 by 10 meters per pixel or less it's good for classifying urban land objects, and a moderate resolution between 10x10m and 250x250m for pixel is suitable for Land Cover Classification [4].

For a more close look, in [21] it's proposed the method of *"fuzzy-fusion inference approach for satellite image classification based on a fuzzy process"*. This method will be compared with Decision Trees (DT) and Artificial Neural Network (ANN) techniques in land cover classification for the district of Mandimba of the Niassa province, Mozambique. For this case scenario will be considered seven land cover classes summed up in table 2.2, and a classifier which uses five spectral bands, blue - band 1, green - band 2, red - band 3, Near Infrared (NIR) - band 4, Shortwave Infrared (SWIR) 2 - band 7, plus two indices Normalized Difference Vegetation Index (NVDI) and Vegetation Index (VI) 7.

Table 2.2: Classes used for Classification. (Based on: [21])

Class Name	**Class Description**
Waterbody	Areas covered by water (e.g. rivers, lakes)
River Bancks	Areas nearby water bodies
Bare Areas	Areas without vegetation (e.g. rock outcrops)
Croplands	Areas covered by crops
Grasslands	Areas covered by herbaceous vegetation
Thickets & Shrublands	Areas covered by shrubs (closed to open)
Forest & Woodlands	Areas with a tree canopy cover greater than 10%

The fuzzy membership functions chosen for the test were an inductive method using histograms and fitted Gaussian functions, these allowed for a relative area to be classified based on the frequency of pixel values within a class. To demonstrate the errors that can occur when creating a membership function, in figure 2.1, is an example of a bimodality in an histogram, this is due to multiple classes been represented in a single pixel. Another type of error that can occur is, for instance, considering two classes, and performing the class membership to them, different bands can assign high values to different classes, leading to inconclusive results. To help in decision making, aggregation operators are used to give a positive or negative quotation to the classification: average, minimum, and two reinforcement ones, Fixed Identity Monotonic Identity Commutative Aggragation (FIMICA) and UNINORM. These two last ones (FIMICA and UNINORM) present better results in decision making, being UNINORM the best one with highest accuracy.

Applying the UNINORM to the area in study, and comparing with the two validation models, DT and ANN, it's expected, with the proposed implementation, to obtain an accuracy close to those two. In table 2.3 can be observed what was hoped, being the overall accuracy not to disparate.

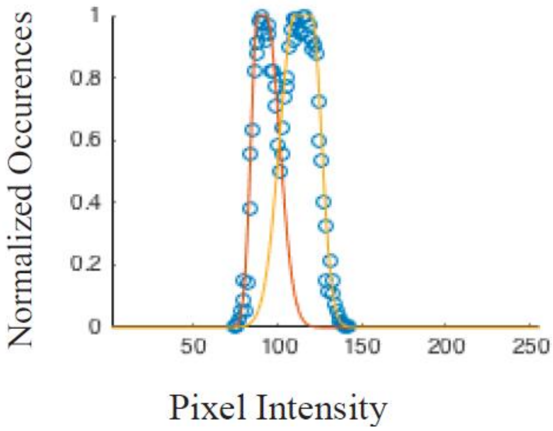


Figure 2.1: Example of a bimodal membership function, for band 1 [21].

Table 2.3: Accuracy of Classifying the training set using UNINORM, DT and ANN [21].

	Water Body	River Bank	Bare Area	Crop Land	Grass Land	Thickets & Shrublands	Forests & Woodlands	Total Average
UNINORM	100.0%	91.1%	94.5%	97.6%	89.6%	90.7%	93.9%	93.9%
DT	100.0%	100.0%	99.9%	95.1%	97.0%	91.1%	97.9%	97.2%
ANN	98.2%	91.6%	100.0%	99.3%	97.4%	99.1%	97.3%	97.6%

Although ANN got a higher average validation accuracy for the training set, the outcome should be analysed in more detail, since Bare Areas had 100% accuracy not being true, cause as shown in table 2.4, acquired from the paper, it says the area to be validated had no Bare Areas, and in fact, it has. This justifies why the training set for the ANN has to be big enough so it can learn correctly. UNINORN and DT adapt better for small training sets.

Table 2.4: Percentage of class presence for the studied region using UNINORM, DT and ANN. (Based on: [21])

	UNINORM	DT	ANN
Water Body	3.1%	3.1%	3.2%
River Bank	10.3%	13.9%	11.2%
Bare Area	0.4%	0.6%	0.0%
Crop Land	7.7%	6.6%	8.1%
Grass Land	33.9%	26.4%	30.2%
Thickets & Shrublands	29.4%	35.0%	30.6%
Forest & Woodlands	15.4%	14.5%	16.8%
Total	100.0%	100.0%	100.0%

The methodology presented in this paper has better results than DT and ANN when considering classification of Crop Land, also when considering River Banks, DT classified them as Bare Areas in awkward regions, UNINORM did not present that kind of misleading. So, the approach had a good performance for the studied region.

In extension of this work [21], authors propose a study with a time-lapse of three years

(1989, 2002, 2005) [11]. The aggregation operators, the five spectral bands and the two vegetation indices (rescaled to a normalized 8-bit unsigned [0,255] scale) to perform the analysis were kept from the paper [21]. Again seven classes are taken into consideration, but only five were used, merging Water Body with River Banks, and Bare Area with Croplands, for comparative results with [22]. The fuzzy membership functions were the same, histograms and Gaussian functions. An additional validation test was added for comparison, the K-Means clustering. The comparison of the tests is presented in table 2.5. It's expected for the ANN approach to have a good accuracy, since the training set was good and wide. As mentioned before, ANN does not adapt as well to small training sets, as the other techniques do.

Table 2.5: Accuracy of classifying the training set with FF-UNINORM, DT, ANN and K-Means. (Based on: [11])

Year	Method	Other	Crop Lands	Grass Lands	Thickets & Shrublands	Forest & Woodlands	Total Average
1989	FF-UNONRM	99.9%	83.0%	81.5%	69.6%	92.5%	88.2%
	DT	100.0%	88.8%	86.0%	87.0%	90.4%	90.2%
	ANN	97.7%	99.3%	66.8%	70.8%	98.4%	95.6%
	K-Means	91.4%	89.4%	44.0%	67.0%	75.0%	78.8%
2002	FF-UNINORM	99.9%	91.7%	59.9%	63.1%	76.8%	81.5%
	DT	100.0%	87.4%	73.4%	69.3%	85.5%	85.5%
	ANN	96.5%	99.6%	73.5%	22.4%	94.6%	91.8%
	K-Means	92.6%	53.0%	33.3%	41.9%	74.2%	63.0%
2005	FF-UNINORM	99.3%	87.3%	63.8%	67.3%	72.6%	78.1%
	DT	100.0%	90.4%	88.4%	78.1%	80.8%	86.4%
	ANN	99.7%	98.8%	84.4%	58.5%	94.4%	92.1%
	K-Means	94.7%	46.3%	21.0%	38.3%	68.0%	53.7%

The ANN method presented the best results as expected, and more consistency accuracy throughout the years. In analyse to table 2.5, it's noticeable that the Thickets & Shrublands class is the one where is harder to get a high correctness, might indicate a misclassified training set. Further tests presented in the paper showed that Fuzzy-Fusion (FF)-UNINORM and DT methods are the ones with most similarities and consistency when class attributing.

In the next case of study are used three other classification methods, Maximum Likelihood Classifier (MLC), Mahalanobis Distance and Minimum Distance [12], explained before. The coverage area are the districts Klang, Petaling, Gombak and Hulu Langat of Selangor, and the land cover classes taken to test are Water Bodies, Forest, Agriculture, Urban and Open Land. The images used to perform the classification were acquired from Landsat 8 satellite and the ground truth was used to verify the identity of class types attributed to the images and build the overall accuracy of the algorithms. To fully understand the process, in figure 2.2 is shown the steps taken, since image acquisition till land cover classification.

Previous studies demonstrated that the MLC is more accurate than the other two

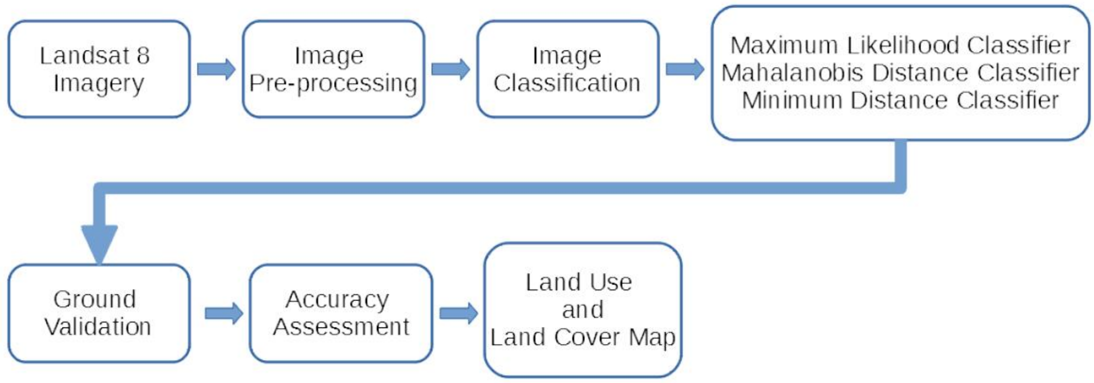


Figure 2.2: Methodology Structure. (Based on: [12])

methods [23], and although Mahalanobis and Minimum Distances have a similar methodology, differences shown in table 2.1, Minimum Distance algorithm is slower to classify the same data. In table 2.6 the advantage of using the Maximum Likelihood method with an overall accuracy of 88.88% compared to 74.44% and 78.88%, respectively Mahalanobis Distance and Minimum Distance. Producer Accuracy (PA) is related to the theoretical validation of the classifier and User Accuracy is related to the validation obtained from the case studied.

Table 2.6: Land Use and Land Cover classification efficiency of different methods [12].

Techniques / **Class Name**	**Maximum Likelihood**		**Mahalanobis Distance**		**Minimum Distance**	
	PA (%)	UA (%)	PA (%)	UA (%)	PA (%)	UA (%)
Water Bodies	50.00	100.00	25.00	100.00	50.00	100.00
Forest	25.00	66.67	25.00	100.00	50.00	80.00
Agriculture	11.11	100.00	25.00	100.00	55.56	62.50
Urban	85.71	85.71	85.71	85.71	85.71	33.33
Open Land	87.50	31.82	87.50	26.92	25.00	100.00
Overall Classification Accuracy	88.88%		74.44%		78.88%	

In a closer analyse, can be observed that the three different approaches, with a max accuracy difference of 10%, perform better for certain scenarios. For instance, having the MLC the highest accuracy, it shows clear difficulties when classifying Forest and Open Land. On the other hand, considering Minimum Distance with the interim result, it can classify well these two classes. At least, the Mahalanobis Distance, when classifying a terrain as Open Land, the results should be doubted cause the leak in accuracy showed in table 2.6. These results also agree with the k coefficient calculated in the paper, with Maximum Likelihood achieving 0.8216, Mahalanobis Distance - 0.6982 and Minimum Distance - 0.7893. The k coefficient is used to measure how certain the algorithm will be able to identify the classes, using a statistical method for qualitative rating, it takes into consideration agreements occurring by chance [24]. A k coefficient close to 1 means the method has a high reliability, conversely, a k coefficient closer to 0 has poor accuracy and should not be used as its doubtful classification.

Deeper analysis to the Minimum Distance Classifier, in [13], a different approach with Discrete Wavelet Transform (DWT) is presented, and as it will be seen, an improvement in the classification accuracy is made. A DWT decomposes the image in its frequency spectrum, in this case, up to eight levels. A wavelet is an oscillation, in which amplitude begins in zero, increases and decreases, returning in the end back to zero. A DWT is a discrete time signal obtained by sampling a continuous translation and scale parameter of a wavelet [25], also it is characterized by its components, the high and low frequency are detail and approximation coefficients, respectively. In this study, the decomposition method is Haar wavelet, which turns the wavelet into a square shape form taking the values 1 for $0 \leq t < 1/2$, -1 for $1/2 \leq t < 1$ and 0 for other values of t.

The performance tests were run to a five class classification, Urban, Fallow Land, Water, Vegetation and Agriculture and eight levels of image decomposition. The accuracy table is presented in table 2.7. To overcome the issues derived from atmospheric imperfections, an atmospheric correction method was implemented, Quick Atmospheric Correction (QUAC). It uses the information in the image/scene, visible and near infrared and shortwave infrared spectrum to adjust the compensation parameters to clear up the image [26].

Table 2.7: Overall Accuracy for wavelet decomposition levels with Minimum Distance approach [13].

Wavelet Decomposition Level	**Overall Accuracy (%)**
1	75.2693
2	85.9600
3	88.8737
4	93.4666
5	95.0346
6	93.0087
7	92.3187
8	90.8021

The fifth level of decomposition, with 95.0346% of accuracy, had the highest score. Also to be noticed is that, the first level has approximately the same result as [12], due to equivalent implementations. The first level implements the method for the original image, without any decomposition, this being said, the only difference between the two papers is the region of study. The DWT presents a 20% increase in the overall accuracy, allowing the method some reliability and the possibility to be executed in other scenarios, situation that had no chance before, because there was no trust in the classification.

As mentioned before, moderate and high resolution Earth imagery for Land Cover Classification comes with their inherent costs. In [4], the data is provided by Google Earth, a free program owned by Google Inc. with a sub-meter pixel resolution, other satellite images can be paid. The region of study is the city of Bangalore, India and the land classes to be defined are Water Body, Building, Vegetation, Road Network and Bare Land. The method to perform this classification and after, comparison of results, is K-Nearest Neighbors (KNN) with Euclidean Distance and Average Pixel Intensity as parameters when choosing labeling. Not just the Red, Green and Blue (RGB) colors were used to identify the classes, but they were converted into *Lab*. *Lab* was defined in 1976, by the International Commission on Illumination (CIE) as a color space [27]. Its purpose was to convert from a three channel RGB into lightness-L, green/red-a and blue/yellow-b. This way, in the scenario of [4], the illumination between objects could be identified and a easier and more accurate analysis made.

It's pretended to see what is the improvement of a generic KNN method to a Euclidean Distance and Average Pixel Density based one. The accuracy results for the studied region are presented in table 2.8.

Table 2.8: User's and Producer's Accuracy values of Generic KNN and Euclidean Distance and Average Pixel Density based KNN. (Based on: [4])

Land Classes	**Proposed Method in [4]**		**Generic KNN**	
	Producer's Accuracy	User's Accuracy	Producer's Accuracy	User's Accuracy
Water Body	94.0%	92.02%	89.02%	91.88%
Building	69.04%	78.06%	70.21%	78.12%
Bare Land	65.03%	66.32%	65.15%	65.04%
Road Network	61.73%	64.02%	60.03%	61.02%
Vegetation	90.03%	88.25%	78.53%	85.05%

As seen in the result table, the overall accuracy went from an average of 75.04% in the Generic KNN to 76.38% in the proposed method. The reason for the improvement is, instead of using only one parameter for class labeling, the presented method uses two, Euclidean Distance and Average Pixel Density. Furthermore, the Generic KNN implementation had misclassifications with waves of water body and some part of vegetation, labeling them, respectively, buildings and road network. The method also helped to solve the problem.

Another classifier is the Random Forest. How it performs can be observed in [28] were it was put the test in a six class Land Cover Classification scenario, in Xuchang city, Henan Province, China. The data provided by the GF-2 satellite and an airborne Light Detection and Ranging (LiDAR) was used together for accuracy improvement. GF-2 providing spectral features and texture information and LiDAR the three-dimensional coordinates. To generate seven scenarios to test the effectiveness of the classifier, NVDI, Normalized Digital Surface Model (NDSM) and Gray-Level Co-occurrence Matrix (GLCM) texture were used with different combinations of the input variables, described in more detail in table 2.9.

To properly function, the Random Forest classifier requires two parameters, the number of variables in the random sampling (mtry) used at each split to grow a decision tree and the number of decision trees (ntree). To optimize the parameters the Out-of-Bag (OOB) error grid search approach was used. The OOB is a method applied not just to

Table 2.9: Different Scenarios and Input Variables. (Based on: [28])

	Input Variables
Scenario 1	4 variables generated by the GF-2 (red, green, blue and near-infrared)
Scenario 2	5 variables generated by the GF-2 and NDVI
Scenario 3	85 variables generated by the GF-2, the NDVI and their texture features (7x7 and 9x9 pixels)
Scenario 4	1 variable generated by the NDSM
Scenario 5	17 variables generated by the NDSM and its texture features (7x7 and 9x9 pixels)
Scenario 6	6 variables generated by the GF-2, NDVI and NDSM
Scenario 7	102 variables generated by the GF-2, NDVI, NDSM and their texture features (7x7 and 9x9 pixels)

Random Forest but also to boost decision trees and other machine learning approaches, and it measures the estimated test error of the bag, this way it's possible to estimate the error in a node [**breiman1996out**, 29].

To fully optimize the accuracy in every scenario, it was used the mtry and ntree parameters. When the value of the OOB error is low, it indicates a good reliability of the model. The accuracy results for the seven scenarios are presented in table 2.10, where can be observed that, seventh scenario had the best accuracy with 93.32% and kappa value of 0.91.

In analysis of the results, can also be seen, that the fourth scenario presents poor results and should not be used. Again, a confrontation between accuracy and kappa values, they vary proportionally to each other. In further detail, the easiest class to be identified was the Cropland, since for all scenarios and producer's and user's accuracy it scored 100% accuracy.

Table 2.10: Total Accuracy and Kappa Coefficient for all Scenarios [28].

Scenario	**Total Accuracy (%)**	**Kappa Coefficient**
1	76.86	0.70
2	77.11	0.71
3	83.33	0.79
4	57.59	0.44
5	80.13	0.74
6	89.97	0.87
7	93.32	0.91

As mentioned, satellite imagery might have a lot of noise, including granularity and clouds, blocking the possibility to implement a method that can perform a Land Use/-Land Cover Classification for the studied region. To overcome this problem, in [30], it's used a spatial-temporal fusion technique. This is based on acquiring images from different sensors and/or at different times and grouping the data. The fusion model used is Enhanced Spatial and Temporal Adaptive Reflectance Fusion Model (ESTARFM) which compounds Landsat 8 and MODIS data. To extract the information from the imagery and identify the classes, an object-based strategy is used. An object-based classification not just considers and analysis the pixel itself but the surrounding ones too, this combines image segmentation with knowledge-based classification. Image segmentation decreases the complexity and divides the image into regions, when these become meaningful, they are considered image objects [31]. According to [32] an object-based approach is able to achieve higher accuracy than a pixel-based approach. In the paper, the technique used was provided by the platform eCognition9.0 and *"the algorithm is a region growing technique that starts with a pixel forming an object and merging the neighbouring pixels until the homogeneity criterion is achieved"*.

The data used in this paper was acquired from the Landsat 8 and MODIS sensors. The Landsat 8 OLI imagery acquired three times in the year 2015, had a cloud coverage less than 1% and a higher resolution than the one from the MODIS sensor. On the other hand, MODIS imagery (including Red and NIR bands) was acquired with a time span of 8 days during the year 2015. The fusion of these two helped to remove the effects of atmospheric interference by correlation of the combined data. The major problem was the poor resolution of 250m per pixel. As for the training and posterior validation sets, the data used was from the GF-1 with a resolution of 2m per pixel (2015-09-15) and Google Earth (2015-09-08). In diagram shown in figure 2.3 is a demonstration how the time series imagery was created with MODIS and Landsat 8 data sets.

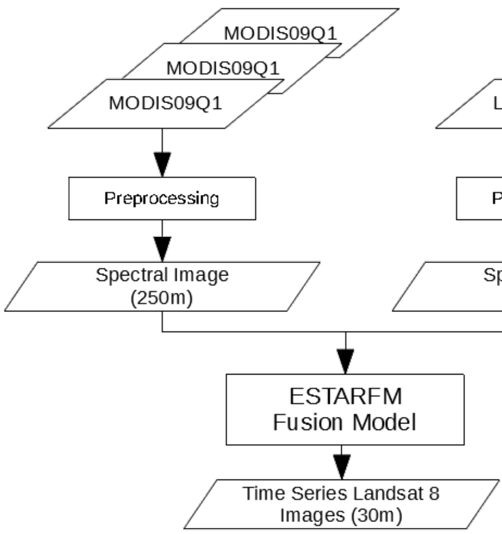


Figure 2.3: Flowchart of the fusion data sets [30].

The studied area was Changsha City, China and the main focus classification technique was an object-based method, as above explained, with fused Landsat 8 time series.

Also taken to test and comparison, other three techniques were performed, an object-based method with OLI images, an object-based method with single date Landsat 8 image and a pixel-based method with Landsat 8 data set. To verify the accuracy of the tests, a confusion matrix with ground truth reference data of the studied region was made. A summary of the training and validation samples for the Region of Interest (ROI) with the different classes is presented in table 2.11.

Table 2.11: Number of ROIs and Pixels for different classes in training and validation data sets [30].

	Number of ROI and Pixels	Water	Grassland	Cultivated Land	Dry Land	Forest	Building	Barren
Training Samples	Number of ROIs	196	102	124	95	202	113	92
	Number of Pixels	3256	4585	2168	3749	4053	3821	2897
Validation Samples	Number of ROIs	48	52	58	69	83	52	71
	Number of Pixels	1048	1204	850	1426	1987	967	1436

It's expected for the classes with highest number of training samples, to achieve better accuracy. This is not always linear, as different classes have lower/higher degrees of complexity in classification methods; other factors, for instance over-fitting, can introduce errors in the validation test, for not being a classifier able to general use, cause it adapts and memorizes the training samples and do not learn the method for a correct classification. A comparison will be made after the analyse of the accuracy result table 2.12.

Table 2.12: Classification Accuracy Comparison with different training data sets and methods. (Based on: [30])

(a) Landsat 8 data sets and object-based method

Cover Types	Producer Accuracy %	User Accuracy %	Total Classification Accuracy %
Water	97.25	97.12	
Grass Land	86.84	85.69	
Cultivated Land	88.72	87.26	
Dry Land	89.23	88.15	94.38
Forest	92.64	93.52	
Building	96.38	95.41	
Barren	92.69	91.83	

(b) Landsat 8 OLI images and object-based method

Cover Types	Producer Accuracy %	User Accuracy %	Total Classification Accuracy %
Water	95.82	95.02	
Grass Land	83.53	82.32	
Cultivated Land	85.68	85.09	
Dry Land	84.62	82.73	90.65
Forest	90.52	90.18	
Building	96.35	95.41	
Barren	92.68	91.83	

(c) Single Landsat 8 image and object-based method

Cover Types	Producer Accuracy %	User Accuracy %	Total Classification Accuracy %
Water	96.13	95.24	
Grass Land	78.25	76.28	
Cultivated Land	75.59	77.36	
Dry Land	79.15	78.47	86.52
Forest	88.49	88.46	
Building	94.23	93.82	
Barren	90.63	89.58	

(d) Landsat 8 data sets and pixel-based method

Cover Types	Producer Accuracy %	User Accuracy %	Total Classification Accuracy %
Water	96.68	96.31	
Grass Land	82.26	81.35	
Cultivated Land	83.82	82.69	
Dry Land	85.23	84.56	88.62
Forest	89.81	90.45	
Building	95.64	94.18	
Barren	91.59	90.27	

The approach presented in the paper, the object-based method with fusion time series imagery, had the highest accuracy of the four tests, proving that the implementation and the training data sets are two big factors in image classification, moreover in Land Cover Classification. When used the same data set, in sub tables 2.12a and 2.12d, it's possible to see the main difference between implementations, object-based and pixel-based, respectively. Considering the pixel itself and the surrounding information, more features can be extracted from the image, leading to a more accurate analysis. For sub tables 2.12a, 2.12b and 2.12c, that used the same method, it's compared the importance of a good training data set. From the data set obtained by the fusion of images explain in figure 2.3, to the three images acquired from the Landsat 8 OLI and finally to a single one also from Landsat 8, it's clearly seen the difference in accuracy, through the output results.

In consideration to the accuracy results for the individual classes, regardless of the implementation, it's important to notice the influence of enough training samples. The classes with more training samples are Water, Cultivated Land, Forest and Building, and in general, Water had the best accuracy. In contrast, Cultivated Land didn't had a good accuracy as foreseen, this is due its misclassification as Grass Land and Dry Land to their similar features.

Some improvements to increase the accuracy are, even better data sets to have a cloud free imagery and reduce the uncertainty of the ESTARFM fusion model images. A high resolution imagery would also help to solve the problem of misclassification. When small regions are attributed with the wrong class by the object-based method, a combination with spectral mixture analyses would improve the results.

The previous cases of study showed Land Cover Classification based on supervised methods, except for [11] which briefly presents K-Means clustering. The next paper explores in more detail unsupervised classification, with ISO-Data and K-Means implementations, also with other two new supervised methods, Parallelepiped and Support Vector Machine (SVM). In [7] are presented four supervised classification methods, Maximum Likelihood, Minimum Distance, Parallelepiped and SVM and two unsupervised, ISO Data and K-Means. From the above mentioned studies, the classification method expected to obtain the highest accuracy was the Maximum Likelihood. The scenario to perform these tests is Paonta Sahib, Himachal Pradesh, India. Since unsupervised algorithms depend on their rule set for the classification, the accuracy they're able to achieve is less than supervised ones which use training data sets in their approach, being much more flexible for different scenarios. For the Paonta Sahib region were defined seven land classes, River, Forest, Urban, Mountain, Scrub Land, Crop Land and River Associated Sand and for the supervised classifiers, the training data sets were restricted to thirty samples per class. The training data set imagery was acquired from Sentinel 2A and its resolution depends on the bands, varying between 10x10m and 60x60m per pixel. After a process of stacking, the final resolution was 10x10m per pixel. The final classification of the different implementations were compared to a ground truth data, and by a confusion matrix, the accuracy results obtained are presented in table 2.13.

As seen in the table, again MLC obtained the best accuracy with 89.30%, followed

Table 2.13: Accuracy measurements of supervised and unsupervised classification methods. (Based on: [7])

	Method	Accuracy (%)	Kappa Coefficient
Supervised	**Maximum Likelihood**	89.30	0.8481
	Minimum Distance	61.90	0.5031
	Parallelepiped	80.07	0.7054
	SVM	75.58	0.6546
Unsupervised	**ISO Data**	30.03	0.0917
	K-Means	22.09	0.0917

by Parallelepiped, SVM and Minimum Distance. The worst two were the unsupervised methods, as it was anticipated. Between them, the best one with 30.03% is the ISO Data. In relation with [12], where also the Maximum Likelihood Classifier was putted to the test, the outcome is really close to which other, proving the consistence of the algorithm. For a context where no previous imagery of the region is available, and an unsupervised method is the only way to proceed, the results are doubtful as poor accuracy was obtained.

From paper [7], SVM had a poor accuracy and an aforementioned methodology pixel-based to object-based approach seemed to increased the accuracy from the pixel-based implementation. It's in the interest of [10], to combine MLC and SVM classifiers with pixel and object-based strategy to perceive how they work together and the benefits they bring to the context. Therefore, for the region of Shandong Province, China were applied four classifications approaches, pixel-based with MLC, pixel-based with SVM, object-based with MLC and object-based with SVM. The imagery data was acquired (two adjacent scenes) from the spatial sensor GF-2 in 2016 with a resolution of 4m per pixel and a cloud free coverage, even though, an atmospheric correction was applied, the Fast Line-of-Sight Atmospheric Analysis of Spectral Hypercubes (FLAASH), to fully optimize the process. For the area addressed were defined five classes, winter wheat, woodland, water, vegetables and artificial surfaces that covers construction, roads and residential areas. These classes were chosen because agriculture is very important for the region and knowing the conditions of the terrain helps farming.

From the imagery data, were selected random pixels, with the help of ENVI version 5.0 software, to constitute the training data set. The total number of pixels used for training are 2,741 pixels for winter wheat, 4,780 pixels for woodland, 27,162 pixels for water, 11,816 pixels for artificial surfaces and 3,472 pixels for vegetation.

When performing the object-based, it's used segmentation to clear separate the objects identified. Different scales of segmentation can be applied, in the paper was studied a variation between 10 and 100 levels with the optimum found to be 25. Table 2.14 presents the results for the four tests.

Table 2.14: Land Cover Classification with Pixel and Object-Based strategies and MLC and SVM classifiers. (Based on: [10])

	Pixel-Based MLC		**Pixel-Based SVM**		**Object-Based MLC**		**Object-Based SVM**	
	PA (%)	UA (%)	PA (%)	UA (%)	PA (%)	UA (%)	PA (%)	UA (%)
Winter Wheat	86.54	83.00	89.16	82.23	85.99	82.85	93.43	85.48
Woodland	83.74	92.86	97.20	84.37	91.05	95.96	99.33	94.77
Water	87.18	97.70	88.32	98.75	91.75	99.32	94.58	98.73
Artificial Surface	87.34	74.61	89.76	79.50	94.47	83.79	94.13	89.74
Vegetables	84.45	61.95	84.71	76.21	85.37	70.99	86.90	84.91
Overall Accuracy	86.67		89.30		91.57		94.33	
Kappa Coefficient	0.796		0.836		0.870		0.911	

Although MLC was expected to present the highest classification accuracy [33], SVM had 94.33% for the object-based strategy. In analyse to table data, Water class is correctly identified in all four approaches. Also an improvement is noticeable between a pixel-based strategy and a object-based one, being the major difference in the Artificial Surface, with an almost 10% accuracy increase for both scenarios. SVM also had fewer misclassifications between Woodland and Vegetables. More detailed review, reviled that SVM performed better than MLC in regions with small area, and with the high resolution sensor used, this difference is nearly 3%. From [7], where the resolution was 10x10m per pixel, a better resolution demonstrates that other classifier should be used due to highest accuracy achievements.

In table 2.15 is summed up the cases of study with the classifiers they use in their approaches, with easy access to what classifiers are research in each study and for a specif classification technique, what studies approach it.

Table 2.15: Summary of cases of study and their implemented classifiers.

Algorithms Applied/ Cases of Study	**UNINORM**	**DT**	**ANN**	**MLC**	**Min Dist**	**Mah Dist**	**KNN**	**Paral**	**SVM**	**RF**	**PB**	**OB**	**K-Means**	**ISO-Data**
[21]	✓	✓	✓											
[11]	✓	✓	✓										✓	
[12]				✓	✓	✓								
[13]					✓									
[4]							✓							
[28]										✓				
[30]											✓	✓		
[7]				✓	✓			✓	✓				✓	✓
[10]				✓					✓					

2.1.2 Cloud Free Imagery

When considering satellite imagery, one of the biggest concerns is to remove the imperfections, mainly caused by atmospheric phenomena, namely clouds and fog. According to [34] there are three major implementations for cloud removal, spatial interpolation based, multi spectral based and multi temporal based. Spatial interpolation based approach uses the non cloud parts of the image to reconstruct the affected areas but with low accuracy results, the multi spectral based approach restore the image using different bands of the spectrum but it is restricted by thick clouds, the multi temporal based approach fuses images from different dates to use the good parts of each one, the only trouble being the time consumption. Previously was mentioned the QUAC approach which uses the band spectrum to correct the parameters and tries to clean the image. Also a time series implementation can be used, having different time imagery for the same region and the correlation of data is made and eliminate the affected areas. The following literature introduces several implementation where the goal is to achieve a cloud free satellite image.

In cloud removing procedure, it can be challenging to distinguish what is actually clouds or cloud-shadow from just bright areas. Even though thresholds can solve the problem when the difference is noticeable, rarely this happens. To overcome this issue, in [35] is implemented a mosaicking approach, with cloudy images acquired from IKONOS and SPOT satellites. The images from the satellite sensors had the influence of different atmospheric conditions and to correctly perform the mosaicking process it required that they become the most similarly possible, this means, if the brightness levels are disparate throughout the images, a gray balance is made to standardize it.

During the process, the classes taken to consideration were Clouds, Vegetation, Buildings and Bare Soil and some of the Bare Soil areas were wrongly labeled as Clouds. To solve this, a different criteria was applied, a three threshold determined from the histogram is made, Shadow Intensity, Cloud and Vegetation Intensity. Then, for the non affected pixels, they were again classified as Vegetation, Open Land and other. Briefly, the pixels are classified as problematic and non problematic, the non problematic ones are then classified as Vegetation, Open Land or other, and the problematic ones are classified as Clouds or Shadows. At this point, the images will be merged. If a pixel was labeled with the Vegetation class for one image, can be assumed for the other images where doubtful classification was made that the pixel belongs to the Vegetation class avoiding discontinuity.

An other implementation of cloud removal depicted in [36], shows an algorithm that only uses visible bands of the frequency spectrum. The proposed method is Generative Adversarial Network (GAN). GAN is an artificial intelligence algorithm that uses two networks, one to generate fake images, which are almost look a like the originals and the second one, evaluate the previous one [37]. The original application of GAN, used visible and invisible bands, yet, the paper's method restricts itself to the visible ones. This implementation was taken as the region of study (Paris) is very cloudy and clouds have high reflectance on NIR band.

The data set training from GAN was derived from Sentinel 2 satellite sensor, comprising twenty cloudy (10 to 100% cloud coverage) images and thirteen cloudless (0 to 5% cloud coverage). For the initial weights of the learning process of the neural network, a Gaussian distribution was done. As for the region of study, no complete cloud free imagery is available, no real scenario test could be analysed. To check de reliability of GAN approach a simulated context was made. Taken the imagery available, a Perlin noise was added to the cloud free images, which has the visual appearance of clouds. A summary of the five tested images is in graphic of figure 2.4 with Peak Signal-to-Noise Ratio (PSNR) results, where high result is better, meaning greater relation between cloudless and cloudy area.

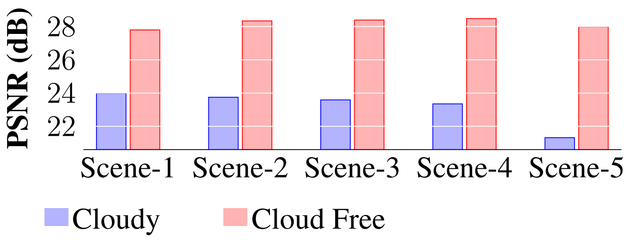


Figure 2.4: PSNR for simulated scenarios with GAN [36].

The cloud free imagery, after GAN been run, has more than 4dB in the relation treated and original images, proving the efficiency of the method. The application of GAN with only visible band usage has its limitations, for an image with significant cloud coverage, the result can be an over smoothed image or a complete fail performance.

A more detailed analyse to the three aforementioned approaches is presented in [34]. It studied the difference between results of the implementations and proposes a complementary multi source methodology to be added into multi temporal. When considering multi temporal image fusion, it's convenient that the imagery data acquired is done in the shortest time possible, so the land coverage do not suffers changes. As high resolution satellite images acquisition has limited periodicity, multi source will be performed and final results discussed. The multi source data derive from a low resolution spatial sensor but with much higher frequency of image acquirement.

The concept behind this is, in a high resolution image, if there is an affected area by means of a cloud, the images acquired from the lower resolution sensor with closest date and that do not present irregularities in that area, are used to cover the specified region. The high resolution images were acquired from Landsat satellite sensor and the low resolution ones from MODIS. To represent the three implementations, Poisson, Weighted Linear Regression (WLR) and Spatio-Temporal Markov Random Fields (STMRF) methods were used and compared with the purposed one. Two tests were performed, both cloud simulated, the second one presenting a more fragmented scenario. The parameters to observe the quality of the methods when reconstructing a scene are Normalized Mean Square Error (NMSE) where low results are better, as for Correlation Coefficient (CC) and Universal Image Quality Index (UIQI) higher is better. The final results are shown in table 2.16.

Table 2.16: Results for Spatial Resolution, Multi Spectral, Multi Temporal and Spatiotemporal approaches in cloud removal operation [34].

		Poisson	**WLR**	**STMRF**	**Proposed in [34]**
Test 1	CC	0.6662	0.7177	0.6946	0.8411
	NMSE	0.0505	0.0434	0.0616	0.0260
	UIQI	0.6268	0.7107	0.6909	0.8135
Test 2	CC	0.3894	0.6689	0.5861	0.5780
	NMSE	0.6852	0.0670	0.0900	0.0938
	UIQI	0.3404	0.6610	0.5682	0.4939

For the first test, the proposed method had the best results in every parameter taken to consideration, although, for the second test, neither STMRF or the proposed method score higher than WLR. This is majorly due to spectral mixed pixels from the MODIS sensor and radiometric inconsistency between Landsat and MODIS sensors.

2.2 FPGA vs CPU vs GPU

As mentioned, when discussing image processing, it's important to understand two main standards, the speed of data processing and accuracy in decision making. There are also other effects that should be taking in consideration such as energy consumption, cost, size, reconfigurability, application-design complexity and fault tolerance. All these have implementation trade-offs and what it's most required for the system being implemented must take higher priority.

Since the demand of greater and more complex problems have reached society, the need of powerful machines and high performance processing units is a must. To achieve a high performance either the operational frequency is very high or the architecture is implemented with parallelism. Systems have changed from a sequential processing approach to a parallel programming one [38], this can be observed in the past years, where the trend of parallelism architectures has increased over time [39].

Different platforms can be used to run the algorithms implemented, here it will be considered CPU, GPU and FPGA. Several studies have been made to compare the performance of these three. As shown in [40], the right balance between operational frequency and parallelism may lead to an optimal performance.

Till a few years ago, programmers have rely in fast and powerful processors, with high clock frequency, which enables a software application to run smoothly and with no apparently latency. This wasn't an issue when these applications didn't take full advantage of the processors capabilities, but applications started to demand more and more performance. In response to this scenario, new generation processors couldn't implement just a faster CPU clock rate as it wasn't significantly different from the previous generation, due to physical limitations such as energy consumption and heat dissipation [41]. This being said, processors started to be designed with several processing units, a multi-core processor. The first multi-core processor dates from 2001 and it was designed by International Business Machines (IBM), presenting a chip with two 64-bit microprocessors, then, four of these microprocessors working together were able to produce a clock speed of 1.3GHz [42].

By the time, software applications were written as sequential programs who depend on a single core CPU. With this new architecture, in which programs were not designed for, the time they took to run an application, in a single ou multi-core architecture, was similar since they didn't use the parallelism that a new multi-core CPU is able to. Software has still limited features and capabilities. When programmers started to think in their problem answers' with the ability to process multi operations at the same time, they took the advantage of a modern multi-core CPU.

Next will be presented the literature review about the performance of FPGA in comparison to other platforms, CPU and GPU. Section 2.2.1 pretends to show what advantages FPGA has in general algorithms, and section 2.2.2 a more contextualized scenario, where FPGAs were used in image processing.

2.2.1 Analyse of Performance Power in FPGA

An example of how parallelism takes great advantage over sequential is demonstrated in [43] where the platforms taken to test are a CPU (Intel Core2), a GPU (NVidia GTX 200), a FPGA (Xilinx Virtex-5) and a Massively Parallel Processor Arrays (MPPA) (Ambric AM2000). It is mentioned that the Monte-Carlo simulation really takes advantage of the parallel computation power as it will be analysed next. The Monte-Carlo simulation uses a powerful generic method to generate samples from an arbitrary distribution, it repeats random sampling to obtain numeric results [44].

In this paper is studied how these four platforms perform in random number generation. To accomplish the goal, three random number generator algorithms were used, they are Uniform method, Gaussian method and Exponential method. Both effective and mean results of the tests are presented in table 2.17, in which can be seen that the FPGA has better performance than the other platforms, approximately three times the GPU, thirty times the CPU and twenty times the MPPA. The main difference is in the efficiency results, FPGA takes an order of magnitude step up from the other platforms.

An other factor to take in consideration, is the base language in what the program is being written. To program an FPGA, it's required to be used an Hardware Description Language (HDL), such as VHDL or Verilog [45]. These operate at a very low abstraction layer and their main purpose is to describe the structure and behavior of an electronic

Table 2.17: Comparison of absolute performance and efficiency of random number generator across platforms. (Based on [43])

	Performance (GSamples/s)				Efficiency (MSamples/joule)			
	CPU	GPU	MPPA	FPGA	CPU	GPU	MPPA	FPGA
Uniform	4.26	16.88	8.40	259.07	15.20	140.69	600.00	8635.73
Gaussian	0.89	12.90	0.86	12.10	3.17	107.52	61.48	403.20
Exponential	0.75	11.92	1.29	26.88	2.69	99.36	91.87	896.00
Geo Mean	1.41	13.75	2.10	43.84	5.07	114.55	150.21	1461.20

	Relative Mean Performance				Relative Mean Efficiency			
	CPU	GPU	MPPA	FPGA	CPU	GPU	MPPA	FPGA
CPU	1.00	9.69	1.48	30.91	1.00	9.26	18.00	175.14
GPU	0.10	1.00	0.15	3.19	0.11	1.00	1.95	18.92
MPPA	0.67	6.54	1.00	20.85	0.06	0.51	1.00	9.73
FPGA	0.03	0.31	0.05	1.00	0.006	0.05	0.10	1.00

circuit or in this scenario, to describe the implementation of a digital logic circuit. These languages allow to model the parallelism and clocking of a hardware description. However, these languages are not commonly used and not the most intuitive to learn and program in. To solve this problem, in [45] is studied an approach with SystemC and CoSynth Synthesizer, which comprehend a set of C++ classes and macros, described in more detail in [46], that allow the source code to be compiled and an executable to be generated and an automatically generation of the hardware description required, respectively. The programmer can now face the problem using a more friendly C++ syntax.

In [45], it's analysed what is the difference between an implementation with SystemC approach and a native VHDL description. Three algorithms were tested, Demosaicing, Binary Morphology and Canny Algorithm. Two FPGAs were used, the LX45 (from the Spartan-6 family) for low cost and the LX50T (from the Virtex-5 family) for high performance. These two were compared to an Intel Core I7 2800MHz with OpenCV implementation. The results are presented in figure 2.5.

From the analyse of the bar chart, it's noticeable that the implementations written directly in VHDL have better results than the ones using SystemC. This confirms the theory that, that kind of implementation is less optimal due to the higher level of abstraction, leading to a executable program with more operations and redundant ones, also it has not much concern how memory is handled, so it *"makes retrieving any details of a running program a non-trivial task"* [47].

2.2.2 Performance Analysis of FPGA in Image Processing

To demonstrate how parallelism takes benefit, [40] shows a problem where it was run a bi-dimensional filter to compare performance between CPU, GPU and FPGA, the platforms used where an Intel Core 2 Extreme QX6850, 8MB L2 cache, quad-core with 3GHz of clock speed, a XFX GeForce 280GTX with 1024MB DDR3 and a Xilinx XC4VLX160

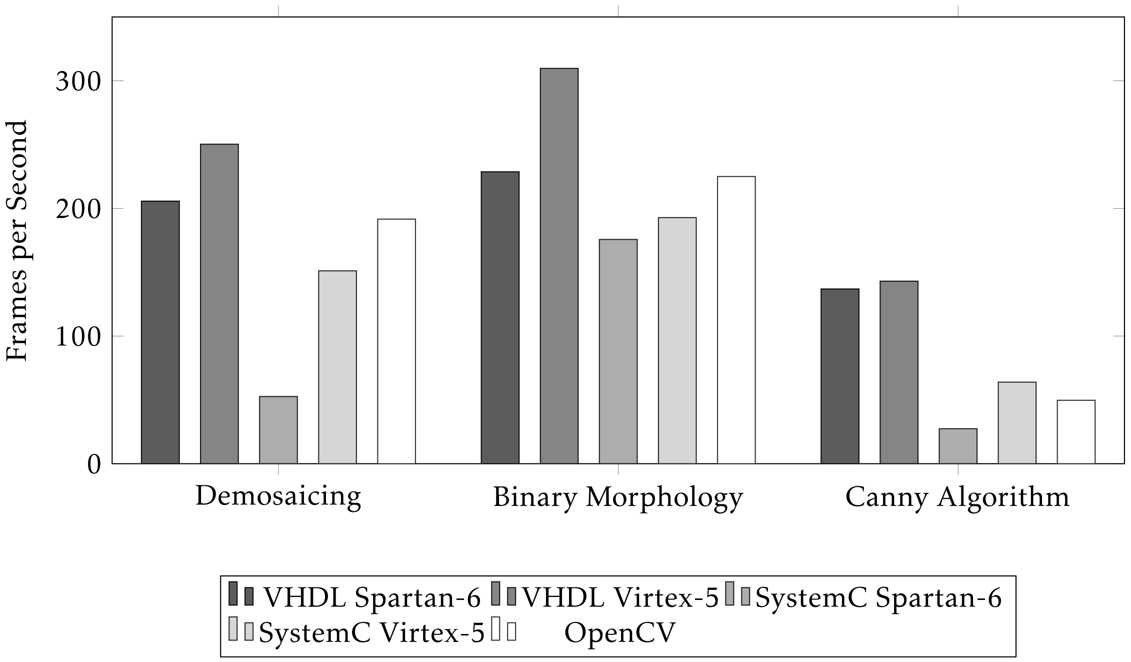


Figure 2.5: Performance comparison of SystemC, native VHDL and CPU implementations. Based on: [45]

respectively. As seen in chapter 3 of the described paper, CPU gets a higher performance with a higher number of cores available, because it uses Single Instructions Multiple Data (SIMD) instructions, where N can be processed in parallel using N cores. To use the graphics card in its full potential, 960 filtering operations were computed to run in parallel. CPU and FPGA showed the same number of operations. As known, CPU has a much higher operating frequency than the other two platforms, this means that, for a small filter size, where the parallelism as no big difference it should get better results.

In figure 2.6 can be seen the evolution of performance by the increase of filter size. As expected, CPU shows better results than FPGA for a small filter size, 3x3 up to 5x5. Although GPU operating frequency was limited to 100MHz it gets better outcomes than CPU for all tests performed. As for the FPGA, it's interesting to see how it performs through the tests, since it is expected to have similar results among them, taking full advantage of its parallel capabilities (a decrease in the performance should only be seen when the filter size k is big enough so that k^2 operations couldn't be taken at the same time, that situation can be noticed at stereo-vision and K-Means clustering algorithm performance test). According to the expected it maintains its frame ratio even for a greater demanding problem. For the last test (filter size of 15) FPGA and GPU get almost the same performance, and by analysis of the curvatures on the graphic, GPU performance tends to decrease and FPGA to maintain till its parallelism capability (much higher than the other two platforms) not been fulfilled. Equivalent results were acquired in the same paper for other tests.

To complement the study in performance between platforms, now will be analysed

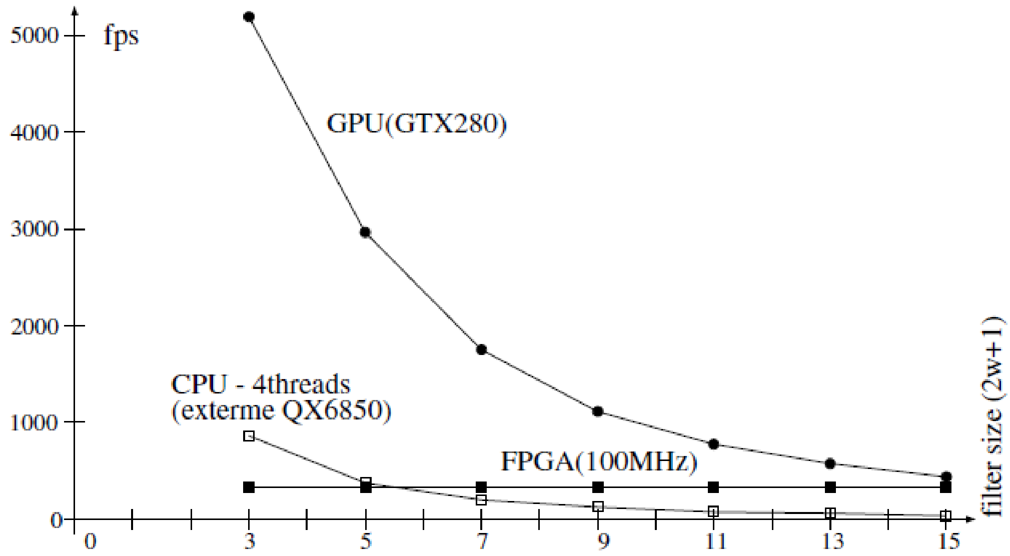


Figure 2.6: Performance of two-dimensional filters, CPU vs GPU vs FPGA [40].

the paper [48] which comprises a comparison among GPU and FPGA with CPU as a benchmark. Two methods of study were performed, the first being a primary color correction targeted at high definition video (1920x1080 frame size at 30 frames per second) and the second case of study, an $n*n$ 2D convolution targeted at a frame size of 512x512 pixels also at 30 frames per second, it's desired a throughput rate of 63MP/s and 8MP/s respectively. The platforms used were two GPUs, the GeForce 6800GT and GeForce 6600GT, two FPGAs, the Virtex II Pro and Spartan 3, respectively high and low performance ones, and the CPU used to benchmark was a Pentium 4 with clock rate of 3GHz. The first aspect to notice, as previous papers state, the clock rate of the CPU is the highest, followed by GPU and at last the FPGA. It's important to keep in mind that the FPGA is the one who implements a higher level of parallelism.

From this paper point of view, when choosing from GPU and FPGA as a platform to use in accelerating video processing, several decisions should be taken in consideration, such as the instruction set and the frequency of memory usage, for example.

In figure 2.7, from the study [48], are presented the results for the first test. There can be seen that both GPUs and FPGAs architectures are able to perform the desire throughput of 63MP/s. The two FPGAs get better results than the GPUs, even when comparing a high performance GPU to a low performance (cheaper) FPGA. One explanation for these results is that FPGA architecture is implemented with a fixed point bit-width optimized designed, this means it has a defined number of digits after the radix point. As GPUs and CPUs don't present this approach, depending on their instructions set, their outcome performance is inferior.

Considering now the second test of [48], the results are consistent to what expected. FPGA keeps its highest performance throughout the test. A minor detail is that for a small filter size, where a high parallelism doesn't make a big difference, GPU takes the lead due

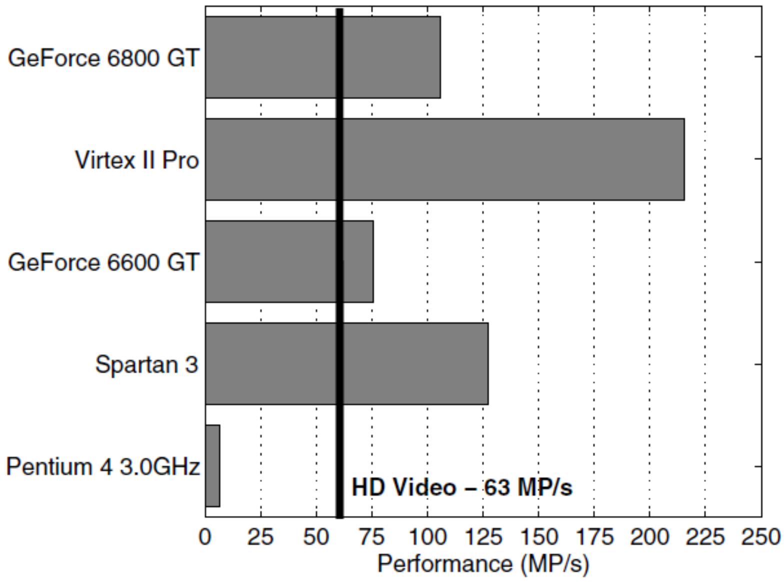


Figure 2.7: Performance for GPUs, FPGAs and a CPU for Primary Color Correction [48].

to its higher clock rate, this accords to [40]. As for bigger filter sizes, FPGA benefits from its parallelism capabilities, pipelining (an implementation technique where multiple instructions are overlapped in execution) and streaming data and shows the results demonstrated in figure 2.8. This proves that parallelism takes a big step into processing data, in this context image data where multiple pixels can be processed at the same time. Even though FPGA has a high parallelism, it has its limitations and it's shown in figure 2.8, for the low performance one, for a filter size over 9x9 it decays its throughput when the high performance keeps it.

In analyse of the graphic it's noticeable that FPGAs are not just more handy to complete the task with the desire throughput, but both GPUs can't be used in the application for a filter size over 7x7.

As mentioned in section 2.2.1 and in study [45], the programming language in which the problem will be approached, influences the processing speed. Moreover, the platform where it is run takes a big step in the performance and deep down, in throughput results. Next, is reviewed a paper with great comparison in performance according to the language where the algorithms were written, in a scenario of image processing [49].

The goal is to achieve a processing time under 40ms. The programming languages tested are C++, MATLAB and VHDL (ran in an FPGA board). For the MATLAB and C++ implementations, as they are designed for a serial CPU, running one instruction after another, translating the code between MATLAB, C++ and VHDL it's not the easiest task. To help, was used the Xilinx System Generator block set for a higher level of abstraction, which is an add-on to the MATLAB-Simulink environment that can *"convert the Simulink model into hardware for Xilinx FPGA"* [49]. To complete the implementation, the hardware used was a ML-555 board (Xilinx Virtex-5 family) and an Intel Pentium dual-core processor, 3.9GHz with 0.99GB of RAM. Ideal for the task as it provides high-speed I/O by

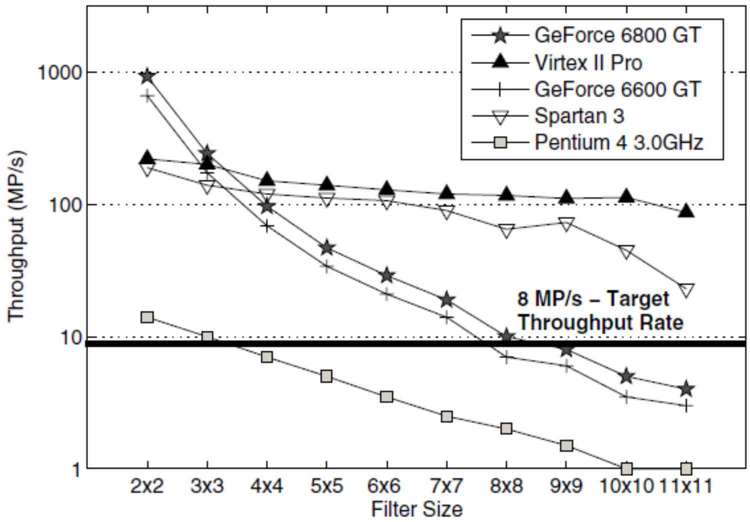


Figure 2.8: Maximum throughput of 2D convolution for GPUs, FPGAs and a CPU [48].

a PCI Express connector for interface with the PC. Three images of different size were taken to test. The results for the test are shown in figure 2.9 with exact processing speed time in table 2.18.

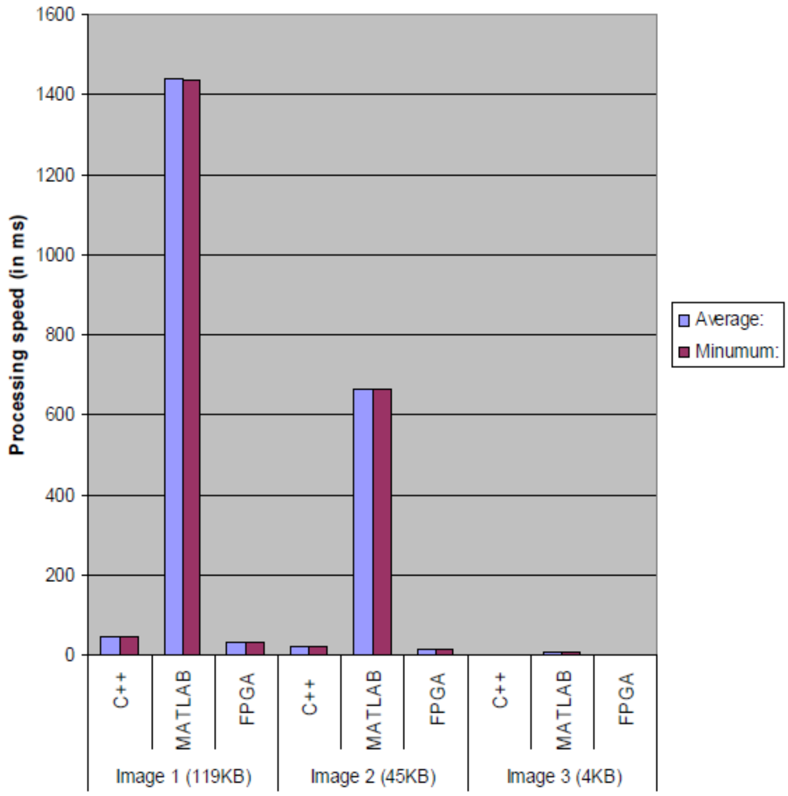


Figure 2.9: Comparison of Processing Speed for different image size and coding format [49].

Table 2.18: Processing Speed exact values for different image size and coding format [49].

Image Size	Coding Format	Minimum Time (ms)	Average Time (ms)
Image 1 (119KB)	C++	46	47
	MATLAB	1435.9	1440.33
	FPGA	30.7201	30.7201
Image 2 (45KB)	C++	23	24.5
	MATLAB	664.8	666.95
	FPGA	14.7456	14.7456
Image 3 (4KB)	C++	0	0.9
	MATLAB	8.1	8.32
	FPGA	0.4881	0.4881

As seen, there is a great variance between C++, MATLAB and FPGA time to process the algorithm. Furthermore, only FPGA can perform under 40ms for all three image sizes, followed by C++ and MATLAB with the worst results. The 0ms processing time for C++ in image 3m, should not be taken to consideration since it was limited by de precision of the timer. This shows that, despite a higher abstraction level, these two programming languages can't get close to the performance presented by the FPGA, taking again full advantage of its parallelism and description of an electronic circuit to optimize instructions. Also should be noticed, the FPGA is the only one that for the minimum and average time presents the same values, this is because applications ran in PC are affected background running parts of the operating system.

2.3 Classification and Image Processing in Zynq UltraScale+

From previous sections it was seen several classifiers obtaining good results, such as Decision Trees, Artificial Neural Networks, UNINORM. The ones decided to be implemented in Hardware to fully optimize the process are Decision Tree and Minimum Distance. This choice is based on the performance of the classifiers and the complexity required to implement in hardware. From these classifiers it's expected to achieve such good performance on hardware as in software, since the comparison will be made with exactly the same implementation in CPU.

To implement the design in hardware will be used a Xilinx UltraScale+ MPSoC ZCU102 Evaluation Kit. This platform is described in detail in table 3.1, but it consists of a processor with several I/O ports available and a Programmable Logic (PL) side in which the classifier will be implemented taking full advantage of parallelism provided by a typically known FPGA. The Digital Signal Processors will be fundamental for this work.

Literature about implementing Land Use/Land Cover classifiers into Zynq devices is little to none. But accelerating some kind of classification applied for other cases of study comprehending image processing can be found and it's good to understand that implementing image processing can be done and results obtained from this type of platforms.

When implementing image processing or image classification, the fast throughput must be a requirement, moreover for a real time system like in [50]. CPUs cannot provide the desired performance based on its sequential behavior, but FPGAs can process multiple operations simultaneously. In this paper, grayscale conversion is applied to RGB images. The method is run in a Zynq family device using the software and hardware parts of the board. The method used covers the image with a window filter of 3x3 applying consecutive mathematical operations. The results obtained were the ability for the design to run as high as 2000 fps but it was bottleneck by the transferring speed of 25ms per frame, summing up in a 40 fps real time system. From these outcomes, its seen that optimizing the process of transferring data into and from the board is a must.

An other example on real time systems is [51]. For this case it's a traffic sigh recognition in which the need of a fast output is more important. In this project it's used a Zynq-7000 along with Matlab. The results are compared with previous works on a MicroBlaze and the Ip core implemented on Virtex 5. The benefits are the faster clock frequencies the system is able to reach, achieving almost 10 times the performance. This advantages also came from an implementation of a System on a Chip providing better communication between software and hardware, rather than a microcontroller with I/O peripherals to the FPGA part.

A third paper will be studied since it uses the same platform as the one in the project. In [52], the Zynq UltraScale+ MPSoC is used for processing operations in 4K video streaming. It was implement simple averaging, Gaussian and Median filters, edge detection using Sobel and Canny methods, morphological erosion and dilatation. From this paper it's possible to understand the resource demanding to process such high resolution images without losing frames, one very important aspect in real-time systems.

Chapter 3

Platform and Software Framework

To develop a Land Use/Land Cover classifier with implementation based on FPGA image processing, it is required a validation platform. The one developed for this work it's represented in figure 3.1. The simplified block diagram consists of a computer and a Zynq board. The computer performs data acquisition and sends it to the board which is preconfigured with the classifier. More details of the work done in each component are described in following sections.

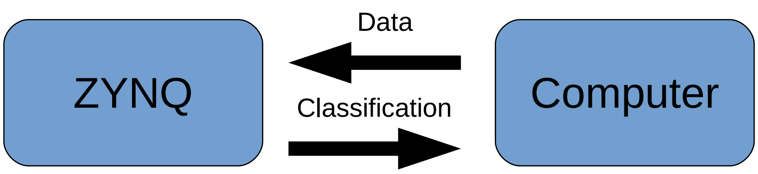


Figure 3.1: Validation Platform - Block Diagram

3.1 Xilinx Zynq UltraScale+ MPSoC ZCU102

The ZCU102 kit is equipped with a quad-core Arm Cortex-A53, dual-core Cortex-R5F real time processors, 4GB 64-bit DDR4 SODIMM (Processing System (PS)) and 512MB 16-bit DDR4 Component (PL) [53], being these three factors strictly important for a classifier, one for store the data required for the classifier and the one generated as an output and second, the high parallelism for calculations in the PL side. As it was seen in section 2.2, for instance, calculations takes major impact when performing multiplications. More detailed specifications from the PL side of the board are described in table 3.1.

As mentioned, the Zynq board has two sides, PS and PL. The biggest advantage from a System on a Chip (SoC) like this, is that the PS can control operations performed by

Table 3.1: Processing Speed exact values for different image size and coding format [54].

Programmable Functionality	System Logic Cells (K)	600
	CLB Flip-Flips (K)	548
	CLB LUTs (K)	274
Memory	Max. Distributed RAM (Mb)	8.8
	Total Block RAM (Mb)	32.1
Clocking	Clock Management Tiles (CMTs)	4
Integrated IP	DSP Slices	2520
	AMS - System Monitor	1
Transceivers	GTH 16.3Gb/s Transceivers	24
Speed Grades	Industrial	-1 -2L -2

the PL and even offload tasks to the PL which leaves the processor with more bandwidth for other tasks [55]. A representation diagram with PS and PL is shown in figure 3.2, there can be seen several components of the PS and a black box for the PL side. From the figure it is also possible to see the interfaces between PS and PL and their are described by the Advanced eXtensible Interface (AXI) protocol. This protocol is described in more detailed in subsection 3.1.1 as it is one important aspect for the implementation of the work.

Also worth to mention that the PS is able to boot with a standalone project or with an Operating System. The different boot mode options are: Quad SPI Flash Memory, eMMC18, NAND, Secure Digital Interface Memory, JTAG and USB. Boot from an SD Card was chosen for convenience. A closer look to this topic is made in section 3.2.

When creating a project in the Zynq UltraScale+ MPSoC with implication to image classification, the three main components above mentioned have a common implementation. More attention is paid in section 3.1.2.

3.1.1 AXI Protocol

The AXI protocol is part of the Advanced Microcontroller Bus Architecture (AMBA) specification, like it is present into the ZCU102 kit. This protocol allows the different modules of the Zynq device to communicate with each other implementing the rules required between them. This protocol implements an effective way of data transmission with a master/slave type of agreement, represented in figure 3.3. This kind of protocol has the following procedures [56]:

- Master and Slave agreement to confirm valid signals, task taken before data transmission;
- Transmission of control signal/address and data must be in different phases;

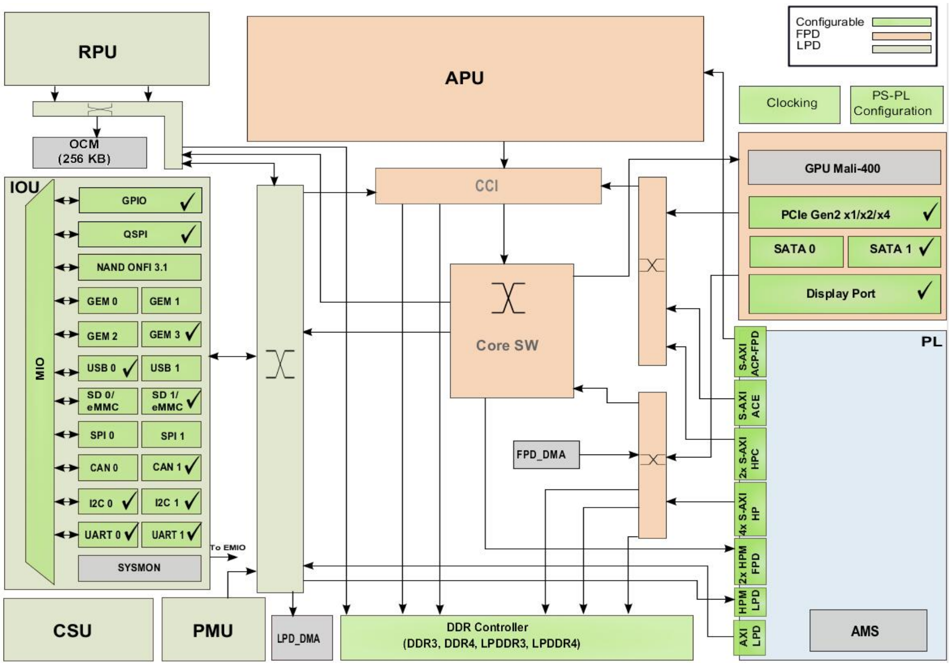


Figure 3.2: Zynq UltraScale+ MPSoC - Block Design

- Transmission of control signal/address and data must be in different channels;
- Continuous data transmission might be done through burst-type communication.

The burst-type communication allows the data to be sent in blocks. This feature will be useful when transmitting data from the PS to the PL, as will be seen in future sections, database and data to be classified are stored in SODIMM DDR4 in the PS side. The AXI protocol can be memory mapped or stream, for this implementation, memory mapped is chosen allowing memory and registers in the module to be associated with memory addresses on the PS.

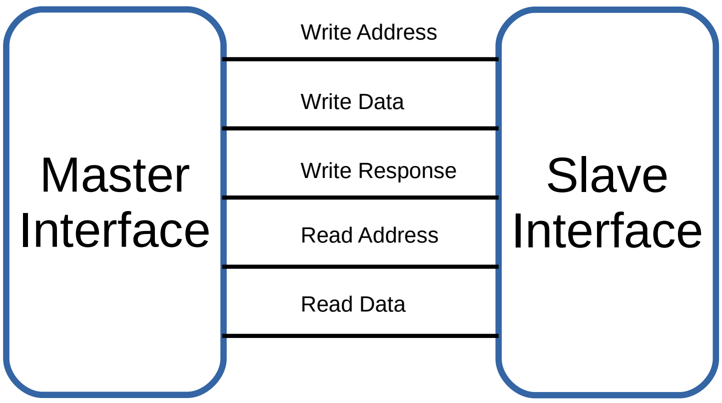


Figure 3.3: Master/Slave Channel Connections (Based on: [56])

3.1.2 Common Blocks

When creating a project which requires the use of the PS side of the Zynq board, one Intellectual Property (IP) that should be present in the block design is the Zynq UltraScale+ MPSoC. Through it, all I/O pins and ports will be controlled, like UART, I2C, DDR Controller, Display Port, etc. In the scope of this work, DDR4 is one other important component, since all the data for the classifier, pre and pos-classification, will be stored in it.

The usage of DDR4 to store the information that the classifier need have its pros and cons. One major benefit is its read/write speed which will affect the time outcome of the classifier, on the other hand, if the board is disconnected from the power supply all data stored in it is lost. In figure 3.4 it's possible to see the basic implementation. Also, between the processor IP and DDR4 IP it's present an AXI SmartConnect block, this will take care of all the different connections between the processor and peripherals applying the AXI Protocol previous studied.

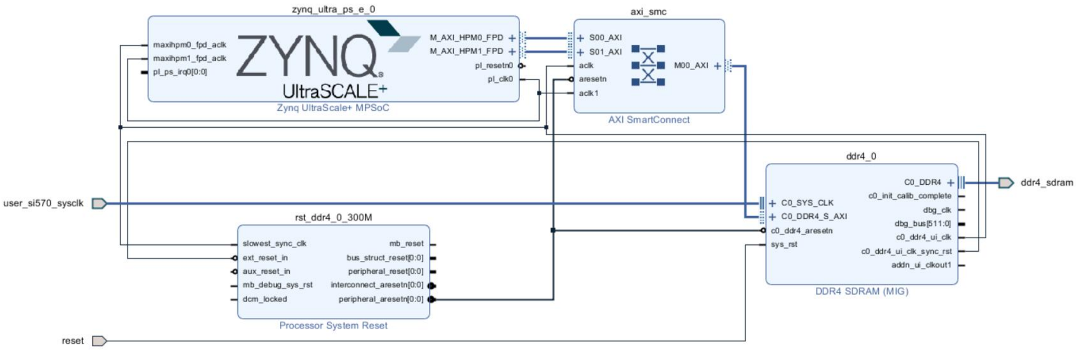


Figure 3.4: Simple Block Design with Processor and DDR4

At this point, no PL implementation is done, so no more connections are made. When the classifier is attached to the block design, more AXI SmartConnect blocks are expected to connect to the classifier and consequently to the PL side of the board.

3.2 Development Environments

To develop the classifier in the ZCU102 kit, two softwares will be used, Vivado HL Design Suite and MathWorks Simulink. Vivado HL Design Suite is a software for synthesis and analysis of HDL produced by Xilinx. The Vivado IP Integrator allows to integrate and configure several IP modules from the Xilinx IP repository and with MathWorks Simulink tools, design and build the projects [57, 58]. The versions required are Matlab R2019a and Vivado 2018.2 for compatibility issues.

The description design will be made in MathWorks Simulink and using Matlab HDL Coder synthesize the desire components to be implemented into the Zynq board. Further adjustments have to be made in the Vivado HL Design Suite software as well as synthesis, implementation and the generation of the bitstream to be loaded into the board.

As mentioned before, the PS as several boot options and can be booted in standalone or into an Operating System. For the case of study it's used an SD Card with a pre-built Operating System image provided by MathWorks. After the board starts the Operating System, the generated bitstream is loaded into the board via JTAG.

The overall description of the system implemented is shown in figure 3.5. In it can be seen the different step the data takes since it is loaded till the final classification.

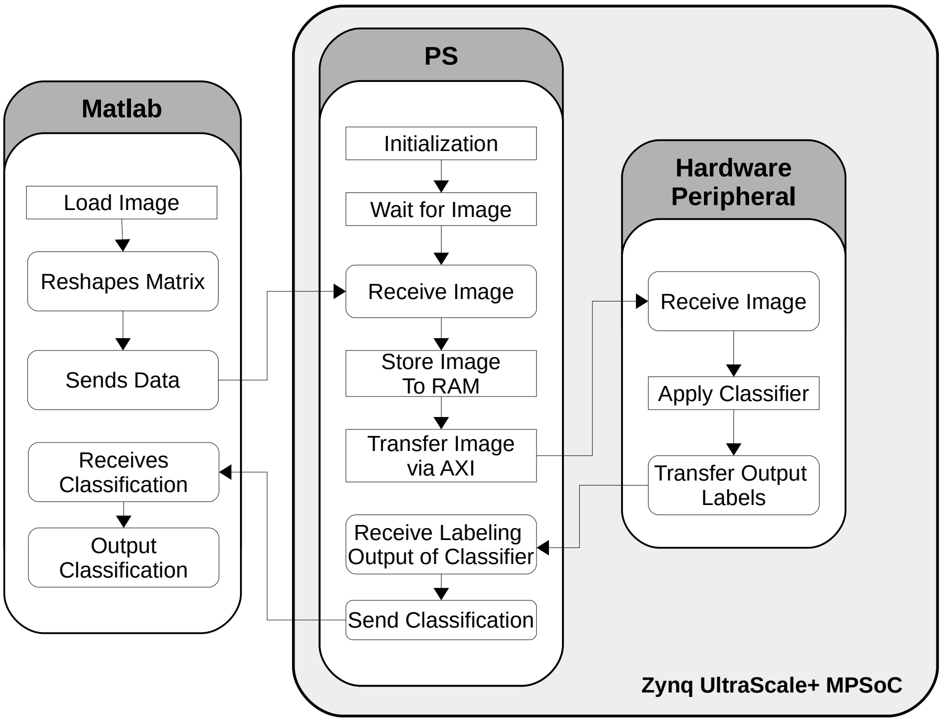


Figure 3.5: High Level Block Diagram for the Classifiers

3.3 Preprocessing Data for Implementation in Zynq UltraScale+ board

As mentioned, the computer has the function of loading the data which is intended to be classified and sends it to the Zynq board. The data used in this work is provided by Sentinel 2 satellite. Sentinel 2 is part of a European Union Copernicus Program and acquired Earth imagery with a time stamp of five days. It has a spatial resolution of 10x10m, 20x20m and 60x60m throughout the 11 bands [59]. For the scope of this work the resolution of 10x10m was chosen and for the bands that do not accomplished this resolution, the method of duplication of pixels was performed to achieve the desired standard. The data was acquired in 24th February 2019, with reference N0213, R037 and T29SNC. Although the area covered by this satellite imagery does not integrated the region of Mação as this work is intended to, it was chosen for the fact that it has more water data information so that the classifier can achieve a better accuracy. The class assessment for the region of study was obtained from Carta de Uso e Ocupação do Solo de Portugal Continental (COS) 2015 [60].

As a primary implementation, the design of the classifier will only range between two classes, water and non water. This approach was done due to the complexity of the work, time to perform it and to keep the highest accuracy possible of the classifier. From COS database, 10 classes are provided for the region of study, they are labeled as so: 0 - ocean, 1 - artificial territories, 2 - cropland, 3 - grassland, 4 - agroforestry systems, 5 - forest, 6 - bushlands, 7 - bare areas, 8 - humid areas, 9 - water bodies. For the first approach to the classifier, ocean, humid areas and water bodies will be considered as water regions and all other labels as non water regions.

The database comes with Tagged Image File Format (tiff) images describing all 11 bands. An additional band was created, NVDI, as it can clearly distinguish water from non water pixels and increase the overall performance of the classifier, the formula for the NVDI index is shown in equation 3.1.

$$NDVI = \frac{(NIR - Red)}{(NIR + Red)} \tag{3.1}$$

When operating tiff images it's possible to see that they are bi-dimensional vectors of single precision variables. This type of variable is a 32 bit floating point value, which means it can store fractions, as it does. When coding in HDL, decimal number is not an option and a different approach has to be taken. If the decimal or integer part of the number is constant, the number could be rounded to the closest exponential base 2 number, but not just neither of the integer or decimal parts are constant and an adaptation to the closest number might result in a non dynamic classifier. The solution is to multiply the database by a number great enough so that the fraction parts can be discarded not affecting the outcome of the classifier, by analyses, 10^5 is reasonable to the classifier become dynamic and perform with the images tested and others. Even though this multiplication, the input numbers for the classifier could be kept at 32 bit and the variable type chosen for implementing in the FPGA was integer 32 ranging from -2.147.483.648 up to +2.147.483.647.

With all these changes applied to the database, it is ready to be performed the classification in any method implemented.

Chapter 4

Classification Algorithms for Land Use/Land Cover

To implement a classification algorithm, in this dissertation, for Land Use/Land Cover classification, it's firstly required to know if there is a previous classified database. With this knowledge will be decided if the methods implemented will be based on a supervised or unsupervised classifier.

As previous mentioned, the database used for the case of study will be acquired from Sentinel 2 satellite with spatial reference N0213, R037 and T29SNC, dated from 24^{th} February 2019. The imagery consists of 11 bands of the frequency spectrum plus the NVDI created, with all the adaptations required described in section 3.3. The images size is 10980x10980 pixels, containing a total of 120560400 pixels. For the algorithms to perform, a training data totaling 3M pixels was chosen. For this data the selection of the pixels was random, with the only requirement being the need of at least 1M pixels containing water information.

The classification algorithms implemented are Decision Tree and Minimum Distance depicted in following sections.

4.1 Decision Tree

The Decision Tree learning algorithm is a tree-like chart which test the data across its nodes resulting in a classification. It's complexity increases by the number of samples and the number of attributes. The learning process of the model can be summarized into *"partionating the nodes, find the terminal nodes and allocate class labels to the terminal nodes"* [61]. The nodes test the attributes of the database by a threshold parameter, in the DT model used, the data passes through the parameter and splits into two categories, within the threshold or outside the threshold. The number of splits per node can be choose by the creator of the model, fitting the best for the database in study.

The building of a DT model can be a lengthy process. This is when the use of the Classification Learner package provided by MathWorks in Matlab software comes handy. With the Classification Learner package several classifiers can be build for the database provided, the right choice of them depends in the case of study, reproducing the model generate in HDL like it is intended in this project can be very complicated, DTs are a solid first approach to the problem with satisfactory results as study in chapter 2.

With the above mentioned sampled training data, in the Classification Learner the model Decision Tree - Fine Tree with no PCA was chosen. The parameters for training the model are described in table 4.1.

Table 4.1: Decision Tree model parameters for Matlab Classification Learner.

Decision Tree - Fine Tree (Parameters)	
Max Splits	100
Split Criteria	Fini's diversity index
Surrogate Decision Splits	off
Observations	3M
Features	12
Holdout Validation	25%
Minimum Accuracy	95%

Although DT classifiers could take long time to model they have a few advantages against other kind of classifier, like Minimun Distance (section 4.2). After the classifier is defined it no longer needs the database for assigning the output saving a lot of memory when the project takes considerable size and since the nodes in the tree are typically defined by *if conditions* it's one of the fastest classifiers. These are trade-offs for not being the classifier with highest accuracy but high throughput.

The model built by Matlab Classification Learner has its own accuracy, calculated with the 25% holdout validation data, this will be compared with the results obtained when applying the model to the entire image. The same model will also be performed in the Zynq board and in a computer, further results (speed and accuracy) and conclusions will be taken in chapter 5. The summary of the process taken in this classifier is illustrated in figure 4.1, representing all the steps between satellite image and final accuracy assessment.

In section 4.1.1 will be explained in more detail how to implement the DT classifier into the Zynq UltraScale+ board, all the software required and adaptations made so it can run.

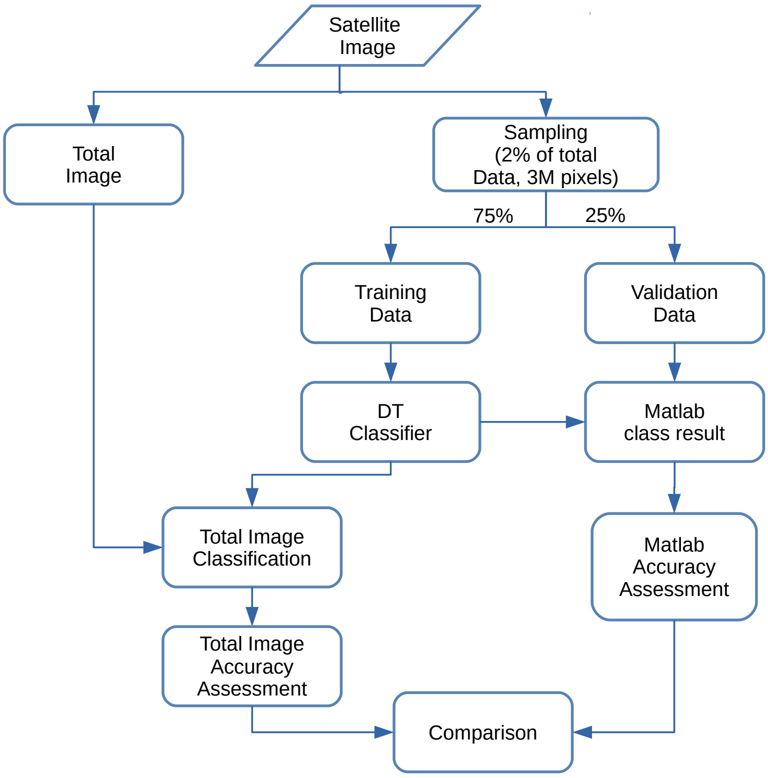


Figure 4.1: Decision Tree Model - Flowchart

4.1.1 Decision Tree Classifier - Software Support

In this subsection will be explained how the classifier was implemented into the board and decisions taken throughout the design in how to most efficiently outcome the classified pixels.

A first approach was done, using a standalone project, with a design project close to the one showed in figure 3.4. Then, the design was exported into the Xilinx Software Development Kit (SDK). Xilinx SDK is "an Integrated Development Environment for development of embedded software applications targeted towards Xilinx embedded processors" [62]. It works in parallel with Xilinx Design Suite and allows the system to be programmed in C/C++ programming languages and an easier debug solution. This alternative accelerates and overcomes the problems of programming an interface with board peripherals in any HDL, additionally, it also provides a vast library available for the user.

The library used was "xsdps.h". This decision was initially taken because the data was intended to be stored in a SD Card. From the library it was possible to initialize the SD Card, read and write data to it. The main concern was the time it was taking to perform the task. To send 200 pixels with 12 bands each it took longer than 10 minutes, so this approach was discarded. Although the optimizations explored were done, the 10 minute mark could not be overtaken.

This is when also MathWorks comes in. MathWorks provide three packages for Xilinx evaluation boards, in specific for Xilinx Zynq UltraScale+ MPSoC ZCU 102 Evaluation Kit. The packages are HDL Coder, HDL Verifier and Embedded Coder. These three tool work together to help the user designing the architecture of the work, creating personalized IP cores, verify and simulate the implementation. HDL Coder is the package responsible for the generation of the IP cores [63], HDL Verifier is the tool that allows to test the implementation of a block creating the test bench for VHDL [64] and Embedded Coder supports generation of C/C++ code for the ARM Cortex A processor family [65]. The support packages provide a boot loader image file containing a MathWorks Operating System. This file is strictly important without it none of the following work would run. The file is incorporated into a SD Card and the board booted from it.

To incorporate the support packages, Simulink will be used to model the implementation based on a graphical programming environment.

4.1.2 Simulink Model

A high level graphical design system of the work is shown in figure 4.2. It includes input and output variables so that the user can control the system. A Design Under Test (DUT) in which is present the classifier itself and a DDR block to interface with the DDR4 SODIMM from the board.

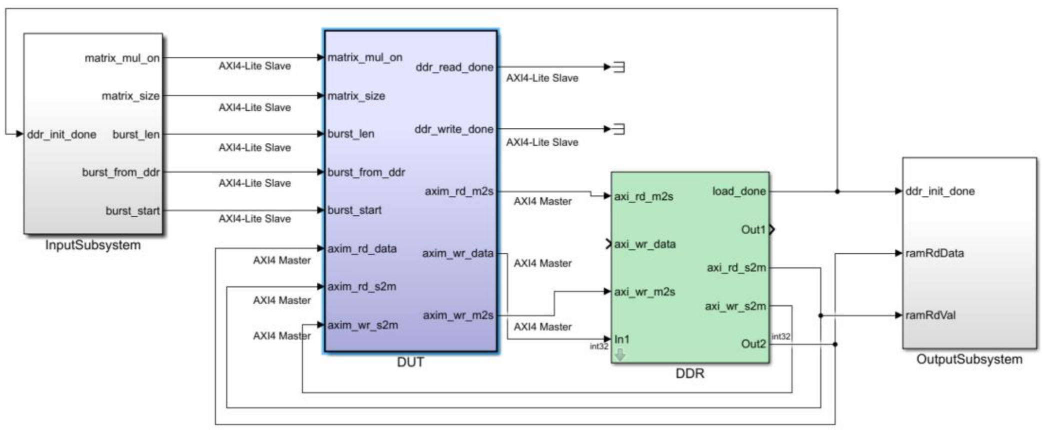


Figure 4.2: High Level Design System - Classifier, Zynq board and interface with the user

Next will be explained the several components of the design and the building process of the entire system.

The first requirement is to design where to store the data, the DDR4 (PS) RAM is the option to take because its high read/write speeds. Simulink provides a simple implementation to interface with the DDR4 (PS) RAM represented in figure 4.3.

The model provided is defined to work only to positive numbers. Some modifications had to be made because the classifier also carries negative integers. The modifications were applied to the block "load_init_data", the final function is described in annex I, Load_Init_Data Block Code. From figure 4.3 it's possible to see that the output of the Simple Dual Port RAM is, as expected, the type *int32* instead of the previous *uint32*.

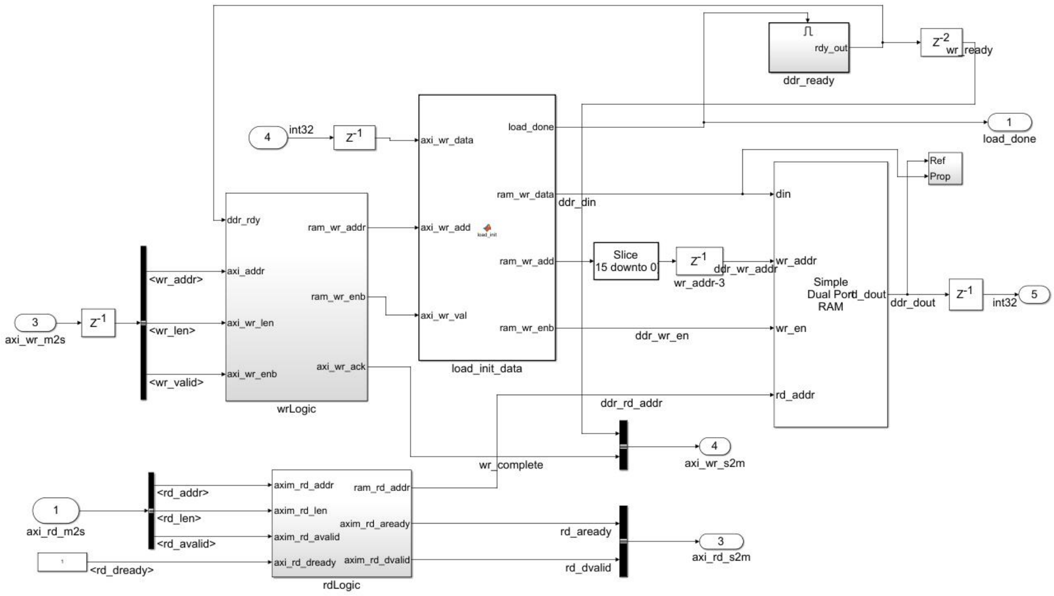


Figure 4.3: DDR4 RAM Interface Block Design

After the DDR (PS) block being defined and the memory of the system being well implemented, the read/write functions to access it must be established. Inside the DUT block it's possible to see a block called "DDR_Access", figure 4.4. In it two function are defined to read from the DDR (PS) into the classifier and write from the classifier into the DDR (PS), these two function are also presented in annex I.

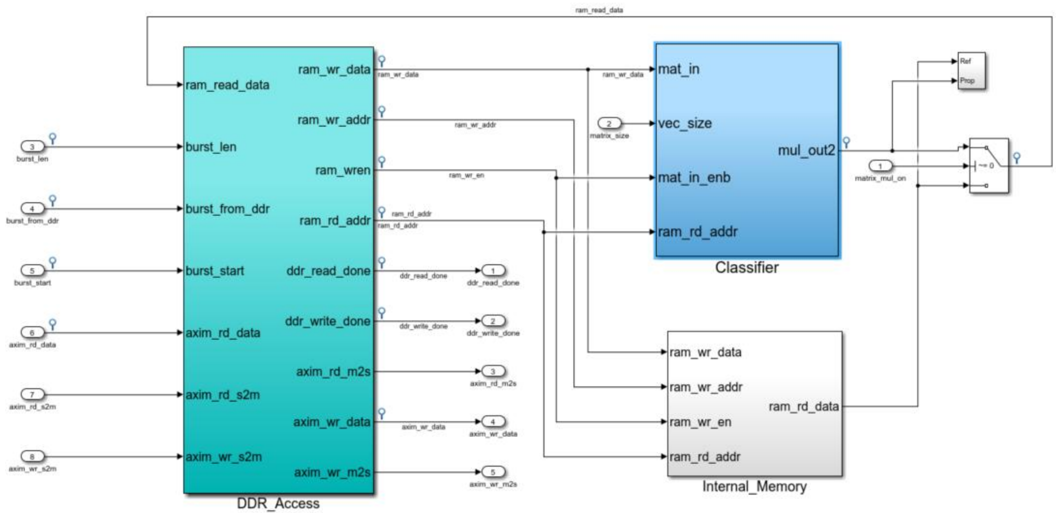


Figure 4.4: Design Under Test

From specific addresses values, the user can control when to read and when to write, from what address in the DDR4 (PS) to perform the task and how many bytes to work with. The process of read/write from/to the DDR4 (PS) is sequential, which means, if the user wants to access three bytes, although the process is automatic, it needs to get one at a time. For this condition the classifier has a few adaptations to handle with it.

Entering the Classifier block itself, figure 4.5, it shows the function that represents the DT classifier and a Simple Dual Port RAM block. After all the data to be classified is loaded into the DDR4 (PS), it is able to start the classification.

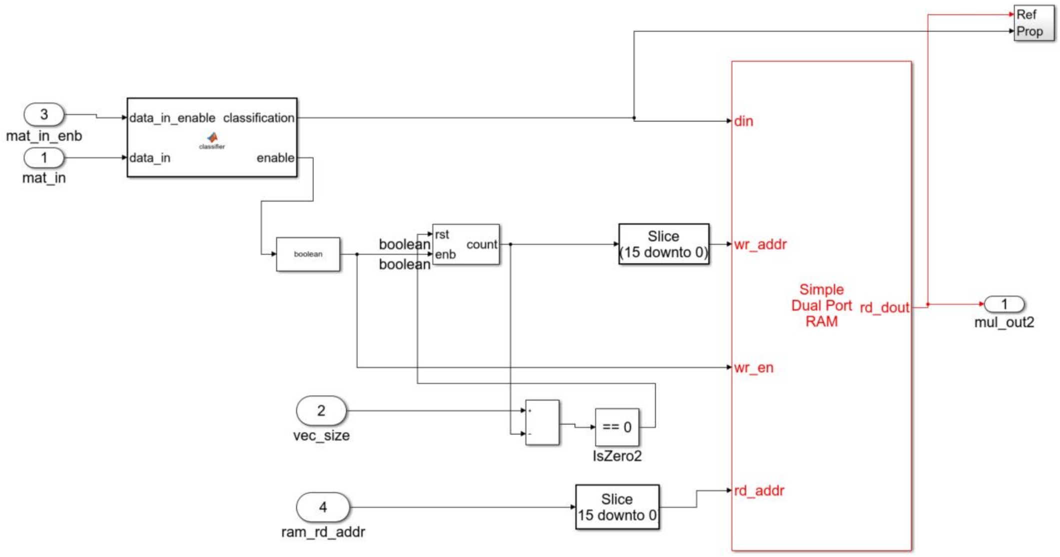


Figure 4.5: Classifier Design Block - Decision Tree

As mentioned, the classifier function only receives one byte at a time. For this classifier to perform it must have access to all 12 bands of the pixel being studied to perform the classification, this means only after 12 clock cycles all the data has arrived to the function. Typical variables inside functions are lost when the functions ends, to overcome the problem of loosing previous pixel data, the different band variables are store is *persistent variables*. This way, as long as the user wants and the board is not unplugged from the power, the variables are available. At every 12 clock cycles the classifier outputs a classification and an enable variable. These are written to the Simple Dual Block RAM. The enable signal increments a variable which indicates in what address the classification should be written to. Although Simulink allows Simple Dual Port RAM addresses to have 30 bits, the DDR4 (PL) is 16-bit representing a maximum of 65535 data to be classified. If more than 65535 pixels have to be processed, the previous ones should be read before new ones written.

Till this point the classification is stored in the DDR4 from the PL side not being yet available, after the user set the command to write from the PL to the PS the classification is ready in a specific address in DDR4 (PS) RAM. To emphasize that all these read/write processes are done through the AXI Protocol.

The classifier function code will be attached to annex I, "Decision Tree Classifier Function".

At this stage the Simulink model is ready to perform classification. The project to implement into the board should now be created so that all the IP cores and connections with peripherals are defined. This is explained in section 4.3

4.2 Minimum Distance

The Minimum Distance Classifier is one model that does not required previous training. The data it needs to perform it's just the database and the pixels to be classified. The method used in this work is the Minimum Distance Algorithm and Euclidean Distance Estimate. It is an iterative method being highly time consuming [66]. It assigns an unclassified pixel to the nearest class in the database. The nearest class is calculated according with the Euclidean Distance [67]. The Euclidean Distance is defined by equation 4.1.

$$D(p_1, p_2) = \sqrt{\left(\sum_{b=1}^{N} (BV_{b1} - BV_{b2})^2\right)} \tag{4.1}$$

From the equation it's possible to see that the number of multiplications to perform the algorithm increases by the number of bands to analyse and the number os samples in the database. This will present a major impact in the time to perform the classification and will be the major difference between CPU and Zynq.

When performing a multiplication, CPU and Hardware implementations don't have similar behaviors. CPUs tend to implement a multiplication by a successive addiction, when for the Hardware implementation, the result of a multiplication is ready at the clock the inputs are presented or at the next one depending how the system is designed to.

For comparison, a multiplication performed in Matlab software running in an I7-8750HQ at 3.9GHz, took an average of 1ns to perform, and the same multiplications applied in Hardware the result is available at the same clock of the inputs, like is presented in instructive figure 4.6. For this reason it's expected the classifier to take advantage of the implementation in the Zynq board. The major difference will also be the amount of multiplications that can be made at the same time, the Zynq board benefits from its 2520 Digital Signal Processing Slices.

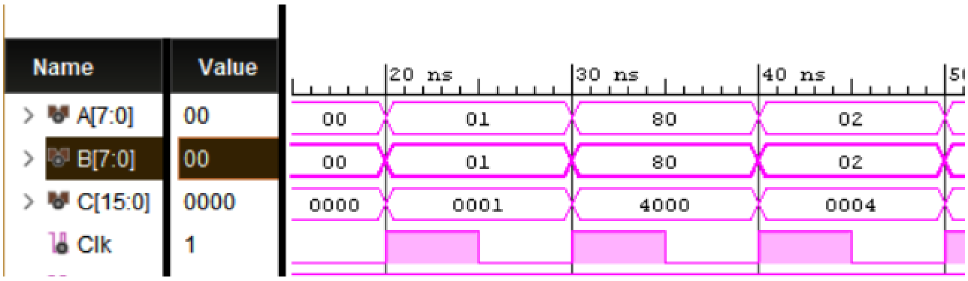


Figure 4.6: Output of a multiplication implemented in Hardware

This kind of classifier has its own advantages and disadvantages. It doesn't require any training process so the implementation of it is faster than the Decision Tree classifier, but after both been implemented its classification takes much longer, this will be seen in chapter 5 along with the accuracy results.

A flowchart explaining the concept of the algorithm is show in figure 4.7. This will be translated into the Graphical Programming Modulation in Simulink and further functions.

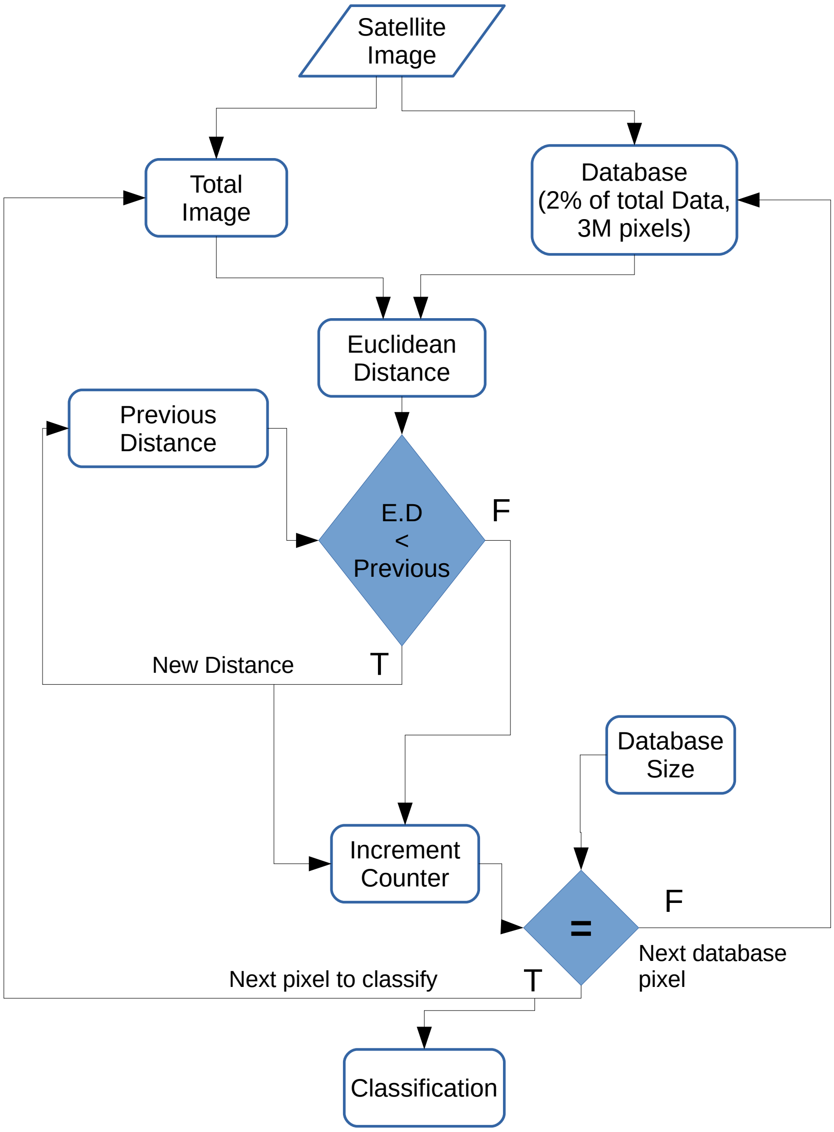


Figure 4.7: Minimum Distance Model - Flowchart

In following section, Simulink Model, will be described how the classifier was implemented and the changes made from the previous model. No need of any MathWorks Machine Learning package since the classifier is strictly mathematical.

4.2.1 Simulink Model

The similarities to the Decision Tree Simulink Model are many. The design of both DDR and DDR_Access modules is the same. The differences came when more variables are needed to the classifier. In the previous one, at every 12 bytes incoming, corresponding to the bands of a pixel, it would outcome with a classification. Here, in this classifier, it will receive the bands of one pixel to be classified and then run across the entire database to search for the one classification that minimizes the Euclidean Distance. To iterate the process for all the pixels to be classified, the function needs to previously know the database and data to be classified sizes. These are input variables of the system which go directly to the "Classifier Design Block", figure 4.8.

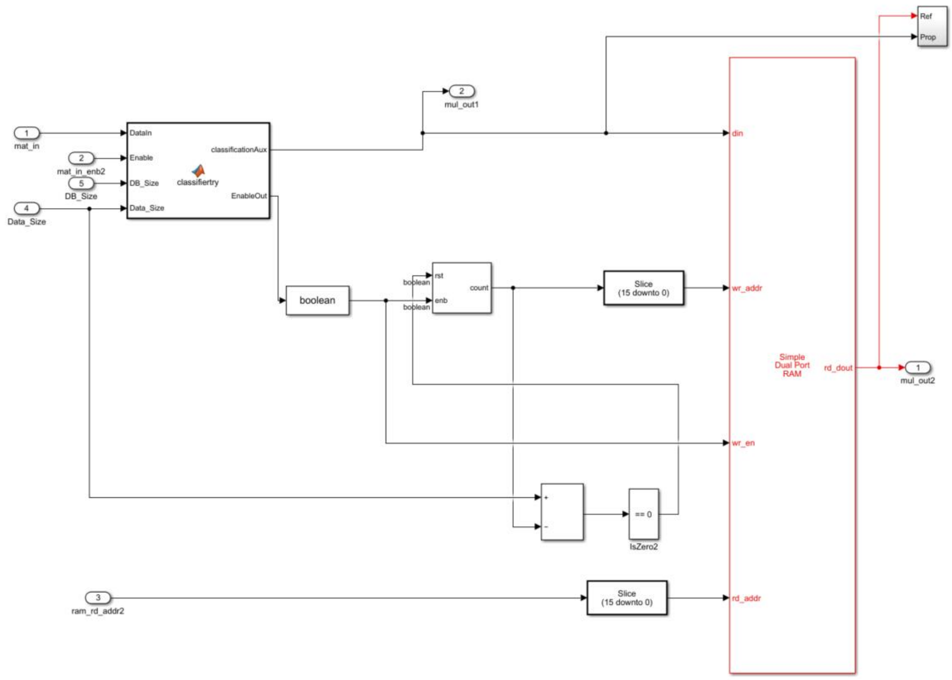


Figure 4.8: Classifier Design Block - Minimum Distance

The classifier function itself is an exhaustive process of addictions, subtractions and multiplications. One particular adaptation was made to implement the algorithm in HDL. When considering the Euclidean Distance formula, it has a root square at the end of the sum of all values. The root square could possible result in a decimal figure. Since the implementation of decimals in HDL it's not trivial, and the distance itself it's not important, but the relation, higher or lower, to the others, it was decided to remove it from the function.

The function is described in annex I, "Minimum Distance Classifier Function". The output of the classifier is saved into Simple Dual Port RAM after the pixel is compared with all the database.

All the limitations applied in section 4.1.2 are present in this model too, ranging from the maximum pixels to be classified and when the data is available to the user. One spec it is important to mention, although the maximum number of pixels are limited to 65535, this number is not the same for the size of the database. The database is stored in DDR4 from the PS side which is 64-bit addressed. By default its writable addresses range from "x8000000" to "x9FFFFFFF", leaving 536.870.911 free addresses. Considering for each pixel of the database have 12 bands plus 1 classification, $536.870.911/13 \approx 41.297.792$ pixels can be stored in RAM. For every pixel that the user wants to classify, one pixel of the database have to be removed. The max number of pixels that can be loaded into RAM is simply defined by equation 4.2. Since this classifier depends on how good the database is, the size limitation could be a constraint.

$$N_{DB\_pixels} = 41.297.792 - Number\_of\_classification\_pixels \tag{4.2}$$

4.3 Vivado Design Suite - Xilinx UltraScale+ MPSoC Classifiers Implementation

This section will describes the process of implementing the classifiers into the Zynq UltraScale+ MPSoC board.

After the design of the classifiers has been done in Matlab software, it's required to use Vivado Design Suite to select what IP cores to use, create new ones if needed and manage all the interconnections between them. The block from figure 4.2 intended to be optimized by the board, taking advantage of its high parallel operation capabilities is the "DUT", since the function to be accelerated is the classifier.

From Matlab HDL Workflow Advisor will be set all the parameters for the creation of the project. This parameters are the the working frequency of the board, the base addresses of the inputs/outputs, the control values when to read/write from/to DDR4 PS side RAM into DDR4 PL side and the addresses for the origin of the data and the output of the classification.

After all the parameters settled, the Vivado project is created. Although Matlab HDL Workflow Advisor creates *a priori* a project, few adjustments had to be made, establish a few more interconnections and address ranges inside the components defined. For this task it's run the Block Automation and the Connection Automation. Block Automation function allows to create a subsystem consisting of IP blocks needed to configure the IP, building an hierarchy and Connection Automation function completes the connections within the system, defining what components are connect, more even, when it's required an AXI Protocol interconnection, setting the specific addresses [68]. The complete design for the implementation is shown in figure 4.9.

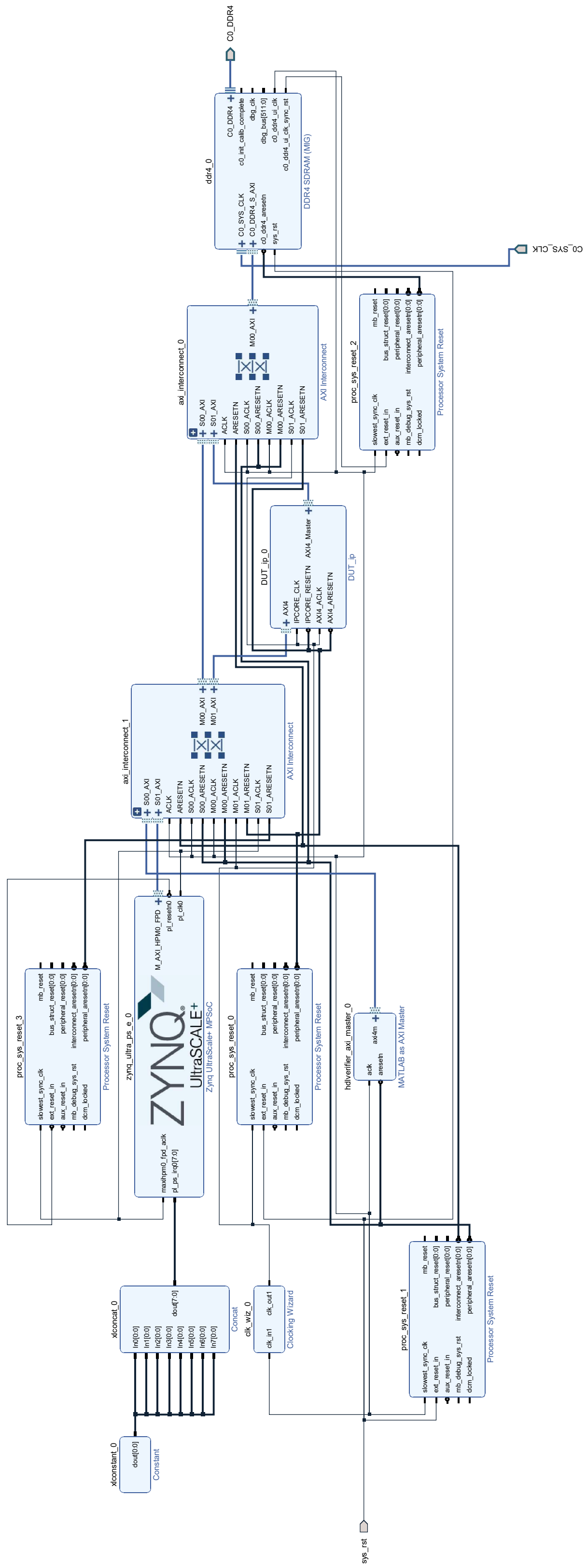


Figure 4.9: Classifier Design Block - Vivado Project

From the block design in figure 4.9, it's important to enhance four blocks, the Zynq UltraScale+ MPSoC, DUT_IP, DDR4_SDRAM (MIG) and MATLAB_as_AXI_Master. The Zynq UltraScale+ MPSoC IP is responsible to control all peripherals, being the *Master* of them through the AXI Protocol. The DUT_IP block consists of several smaller IP blocks generated by Matlab Workflow Advisor, its function is to reproduce the Simulink model. All the blocks inside the DUT are represented individually by an IP entity. It was chosen that the HDL for these IPs is VHDL. The DDR4_SDRAM (MIG) allows to store the data to be loaded into the classifiers, this RAM is presented in PS side. At last, the MATLAB_as_AXI_Master block represents the interface between Matlab and the Zynq board. As Matlab will be the software responsible to load the data into the board it is set as *Master*.

Before understanding the schematics, it is necessary to understand the instructions occurred between loading the data and the final output. The flow of instructions will also help to define which IP block is *Master* of which. The data is previously loaded in Matlab software, then, the same data is intended to be sent over to DDR4 PS RAM, only after that the classifier could start to run. After the classification, Matlab reads the output from a specific address in RAM.

From these instructions it is possible to conclude that MATLAB_as_AXI_Master IP block has to be one of the *Masters* of DDR4. The DUT need to access the data inside the DDR4 so it is its second master. The Zynq UltraScale+ MPSoC controls the DUT being its master and by hierarchy master of DDR4 too. Referring the *Master/Slave* is translated into AXI Interconnect blocks in Vivado design scheme. In figure 4.9, for the blocks mentioned, on the right side of the block there a *Master* port and on the left side, a *Slave* port.

As mentioned before, the frequency defined for the PL side was defined in Matlab and was left as default, 50MHz. Higher frequencies would required delay blocks due to the board could not make all the calculations in one clock cycle. This frequency is one of the input parameters of the DUT block. But DDR4 RAM has a default clock frequency of 300MHz, to synchronize this two blocks as they are interconnected, the Clock_Wizard IP is used.

The two classifiers have the same visual appearance in the Vivado Block Design, their difference comes in the DUT_IP block which has different IP blocks inside, they use different IP repositories.

This is the implementation of the classifiers in Vivado, next it's run Synthesis, Implementation and Generation of the Bitstream. The bitstream file will be configure to the board before anything else. The Zynq board will then be ready to perform classification, the results obtained for the database studied are presented in chapter 5.

Chapter 5

Results

In this chapter are presented the results acquired from the classifiers implemented in the Zynq UltraScale+ MPSoC ZCU102 board. A comparison of performance and accuracy will be made with the same implementation run in CPU.

When considering a classifier two results are expected from it, accuracy and processing speed. The higher both categories the best. Both Decision Tree and Minimum Distance classifiers are implemented in the Zynq board and in CPU, I7 8750HQ - 3.9GHz, in which the algorithm is run in Matlab.

It's important to understand that either way, the classifier algorithm is the same, so in terms of accuracy it's expected similar results. Similar results and not exactly equal results, since one of the classifiers involve mathematical calculations, Matlab using a floating point operations takes great advantage, having more significant figures, being more precise, but accuracy overall should not be affected.

Following sections compare the implementation of the classifiers in terms of processing speed and accuracy.

5.1 Accuracy Assessment - Zynq board vs CPU

The Accuracy Assessment is one important part of this work, if the classifier is not well built, its results cannot be trusted. To calculate the overall accuracy it's required an ground truth data. For the scope of this work and in Satellite Imagery Classification itself, the ground truth data is collected in the field [69]. The COS validation will be used for that. Both Decision Tree and Minimum Distance classifiers will be tested against this database. The accuracy results obtained are presented in following subsections.

5.1.1 Decision Tree Classifier - Accuracy

For the Decision Tree Classifier, as mentioned before, was used the Classification Learned Support Package from Matlab to help building the tree. One of the parameters required was that the accuracy should be over 95% for the validation test. This was chosen because it's expected it to drop when performing for the all image. The accuracy can be kept high as the implementation is being made for water and non water classes for the reasons explained before.

The output of the Classification Learner was a 100 nodes split tree with 98.7% accuracy for the holdout validation test. The tree was then programmed in Matlab and VHDL. Since the Classifier doesn't have any calculations and the data provided is the same no matter the platform, it's expected the output of the classification.

When assessing the accuracy, comparing the outputs of the two platforms, the results are the same, pixel per pixel. In table 5.1 it is presented the confusion matrix for the output of the classifier, label 0 represents water and 1 non water.

Table 5.1: Confusion Matrix for the Decision Tree Classifier

0 - water, 1 - non water

		Predicted Values	
		0	1
Actual Values	0	869440	63549
	1	899464	18167550

For a total of 20M pixels analysed in this classifier, the accuracy for the two proposed classes is 95.181%, presenting a good classification.

To search were the classifier is failing, it was run on CPU for the whole image, this could be done from previous comparison showing the output is absolute equal and as will be seen in next section, CPU performs faster for this classifier. The COS mega classes are represented in figure 5.1 representing the 2 mega classes. The real COS 9 mega classes was converted into 2, Ocean, Humid Areas and Water Bodies are represented as black pixels and all other labels are white pixels. In figure 5.2 it is demonstrated the two output classes of the classifier.

To totally understand what the classifier is missing, the true RGB image for the region of interest is shown in figure 5.3.

From the three images it is possible to understand the regions that algorithm is wrongly classifying. For the water pixels that the classifier attributed non water labels, they most reside in the river banks in Natural Reserve of the Sado Estuary. But even so, the contours for the shore are really sharp, also because, these were not the labels that the classifier mostly failed. Referring to the non water pixels that the classifier attributed water labels, they are also blended in the river banks and all across the image. From deeper analysis, it mostly confuses forest territory with water regions. One reason for this

Figure 5.1: COS Mega Classes used as validation and training samples

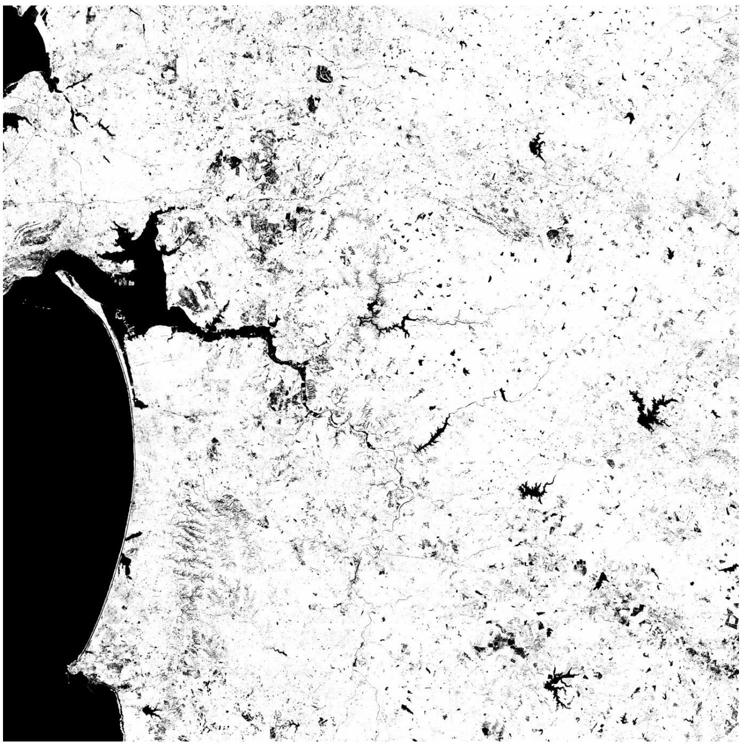


Figure 5.2: Output of Decision Tree Classifier with 2 classes - water and non water

could be that band number 9 of the satellite, representing water vapor have close values in both regions. These miss classified regions can be easier observed in the down left side,

Figure 5.3: RGB Satellite Image

by the shore in figure 5.2 and in regions close to Sado river. From the global view of the image can also be seen that bigger water regions where firefighting vehicles can refuel are clear distinguishable.

In summary, the results obtained are not far off from what expected from the foreseen in Matlab Classification Learner with only a 3% decrease in accuracy.

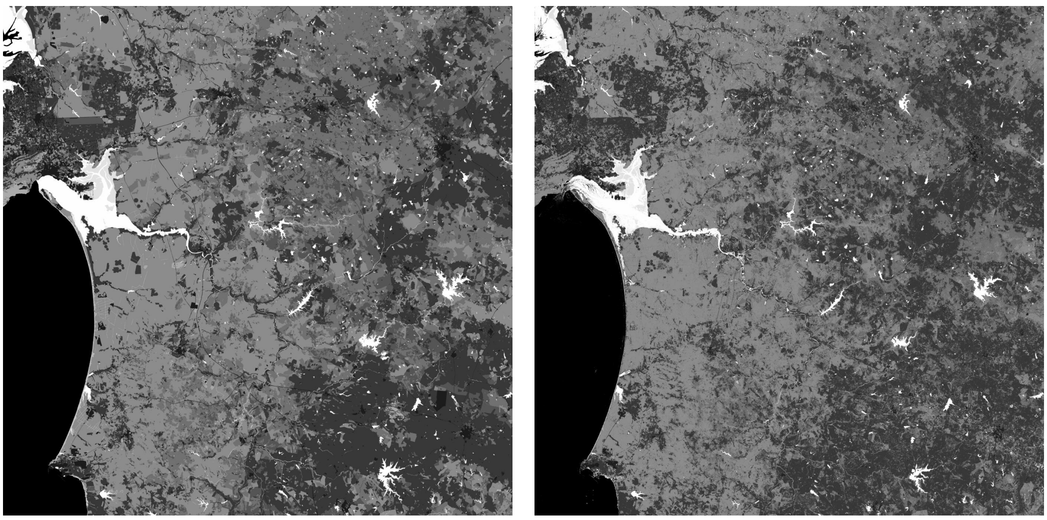
Further work was developed in the Decision Tree Classifier. To start the classification with more than two classes, a third classes was added. It was chosen the three main classes that are considered more important in the context of preventing and fighting fires, the classes are water, forest and everything else. Forest was added since it is one easy way for the fire to spread out and it is the habitat of a lot of living species.

To build this three class classifier the same method as before was applied. Ocean, Humid Areas and Water Bodies are merged into one class (class 0), Forest is the second one (class 1) and all other classes provided by COS are in class 2. Since more classes are set a bigger data training set is needed. For this scenario a 10M pixels database is used for training with 10k split nodes. The same 25% holdout is used for validation achieving 88% theoretical validation. In practice, the classifier was able to achieve 82.21% accuracy. The output of the classification is shown in figure 5.4. There can be seen that the miss leading water pixels were cleaned from the image and the contours of the water regions are more sharp. Even Sado river flow is noticeable in the image.

To conclude the study of Decision Tree classifiers, a last approach was taken to full compare the performance with all the classes in the COS imagery. The same database

Figure 5.4: Output of Decision Tree Classifier with 3 classes - water, forest and everything else

from the DT with 3 classes served to build the classifier, again with 10k split nodes. Any update in size could be made for constraints in the computer used to perform the task. When comparing the output, represented in sub figure 5.5b with COS classification, figure 5.5a, the accuracy achieved is 63.33%. The confusion matrix for this 10 class algorithm is shown in figure 5.6.



a - COS Classification

b - Output of Decision Tree Classifier

Figure 5.5: Region of Study with 9 classes

Figure 5.6: Confusion Matrix - Decision Tree Classifier with 10 classes

		Classifier Labels									
		0	1	2	3	4	5	6	7	8	9
True Labels	0	11439372	3031	423	6	0	102	1	570	1505	78921
	1	9387	1164161	1009334	16486	64592	569086	548	1178	16887	99666
	2	1424	253426	23023874	961179	2825196	3030074	2021	7187	82359	397996
	3	124	468819	7626485	1133931	2239275	1152288	347	1052	5619	74214
	4	15	18030	5187860	450814	13018082	6551840	688	62	3261	117750
	5	873	163757	3461113	161198	5382259	23330637	5712	6347	22402	195797
	6	149	31431	153588	5406	76402	470217	14059	2043	4148	12003
	7	12441	30797	27746	267	6056	93073	26	61663	2466	11168
	8	407	2241	17824	98	416	16490	49	1	370113	309883
	9	42909	16463	275976	9153	82956	162878	66	607	105620	2802158

From the images, the clear differences are, all the artificial territories inside Portuguese mainland are lost. In contrast, in Sado estuary, comparing with figure 5.3 can be noticed more detail, distinguishing the river banks from the ocean. The overall differences of the two images are not big even though the accuracy does not says the same.

5.1.2 Minimum Distance Classifier - Accuracy

When applying the Minimum Distance Classifier, the same database used to train the Decision Tree Classifier, will be here used to calculate the Euclidean Distance and then labeling the pixels.

As said before, this classifier could suffer differences between Matlab and Zynq implementations. Between calculations, VHDL could round some figures and all across the search in the database, the minimum distance could be affected by this.

For time and resource restrictions the classification was performed for 250k pixels. When looking for the classification from Matlab or Zynq board, the labeling was exactly the same. In study of the course of the program could be seen that the Zynq board did not round any figures. This was in fact due to the multiplication algorithm does not constitute a problem in VHDL, in contrast to division and square root. As explained, the square root was not applied in the Euclidean Distance formula meaning a perfect match for the two implementations. The output results of the classifications is presented in table 5.2. As before, 0 represents water pixels and 1 non water pixels.

Table 5.2: Confusion Matrix for the Minimum Distance Classifier

0 - water, 1 - non water

		Predicted Values	
		0	1
Actual Values	0	43969	4072
	1	14826	187133

This classifier for the regions analysed achieved a 92.4% accuracy validation. The 250 pixels used were picked so that the relation of water and non water pixels was kept closely to the one in the Decision Tree classifier.

As previous, this classifier can be applied for the two class classification. This type of classifier is more affected for the chosen database than the one before.

5.2 Processing Speed - Zynq board vs CPU

The time taken for an algorithm to be completed not just depends on the machine in which it is being run, but also the what kind of algorithm is being implemented. If the algorithm is based on sequential conditions like the Decision Tree, it is less time consuming than a mathematical approach, even more, an intensive method, comparing one pixel to all the database like the Minimum Distance classifier.

The classification process is the same for the implementation in the Zynq board or in CPU. The time taken for loading the variables in RAM, either in the Zynq board or in the computer RAM, is not considered.

The expected results in this field for what platform runs faster differ from what classifier is considered. For the Decision Tree, it doesn't involve calculations, so, a higher working frequency of the CPU should take advantage. On the other hand, as explained in figure 4.6, multiplications have the output sooner in hardware than in software and several multiplications can be made at the same time when performing the Euclidean Distance, so the Zynq board should have a faster classification.

In following subsections are presented the time to perform each algorithm.

Although the time taken to transfer the data into the Zynq board is not considered in the scope of this work, the average baud rate was 9446 32bit integer per second.

5.2.1 Decision Tree

When performing the Decision Tree algorithm, the data used reduced into 20M random samples, due to the time the classifiers took to perform. 20M samples are still enough to see the difference in the two platforms. Also, to see if the time to perform would increase linearly or exponentially with more samples, the classifier was run with different number of samples. The results for the Zynq board and CPU are presented in figures 5.7 and 5.8 respectively.

From the images it's possible to observe the big advantage of using a CPU for processing an algorithm based on a Decision Tree. Its high working frequency is much more useful for a sequential kind of algorithm. Further work was done to improve Decision Tree processing time results for the FPGA. With a few adaptations required to increase the clock speed from 50MHz to 500MHz on the board, the time difference is demonstrated in figure 5.10.

For both implementations there is a linearity between time to finish processing and the number of pixels. The Zynq board takes an average of 1000 times the time to process the same data than the CPU.

Table 5.3: Decision Tree: Time to Perform Classification - Zynq board

Nº of Pixels	Time to Perform Classification (s)
50000	1.034
100000	2.065
200000	4.131
500000	10.34
1000000	20.676
2000000	41.342
5000000	103.21
10000000	206.33
20000000	411.56

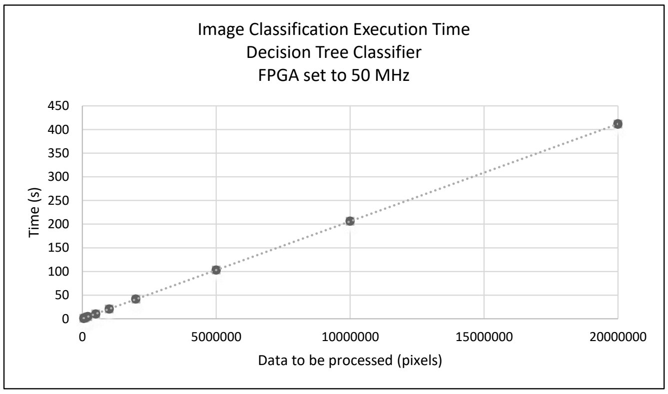


Figure 5.7: Image Classification Execution Time: Decision Tree Classifier - FPGA set to 50 MHZ

Table 5.4: Decision Tree: Time to Perform Classification - I7 8750HQ 3.9GHz

Nº of Pixels	Time to Perform Classification (s)
50000	0.001
100000	0.003
200000	0.005
500000	0.010
1000000	0.021
2000000	0.044
5000000	0.113
10000000	0.220
20000000	0.435

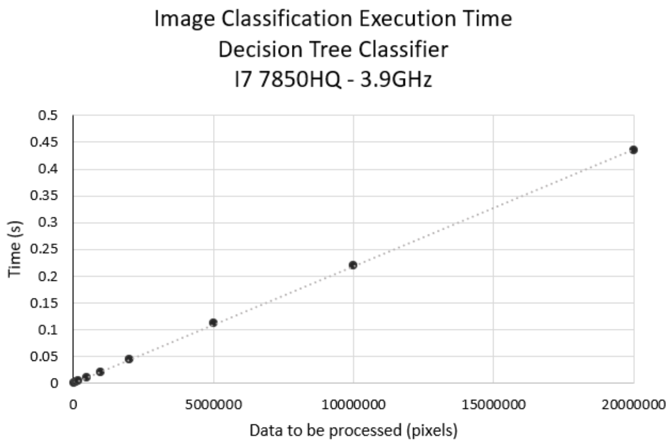


Figure 5.8: Image Classification Execution Time: Decision Tree Classifier - I7 8750HQ 3.9GHz

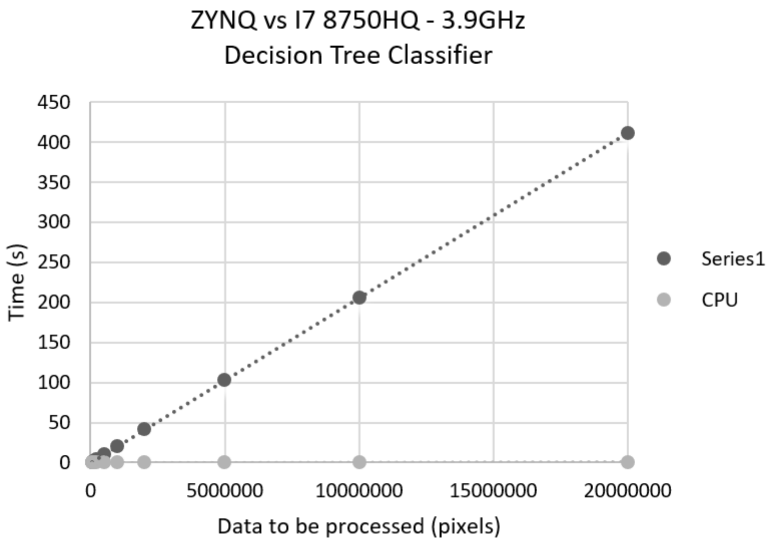


Figure 5.9: Decision Tree speed comparison between Zynq board and CPU

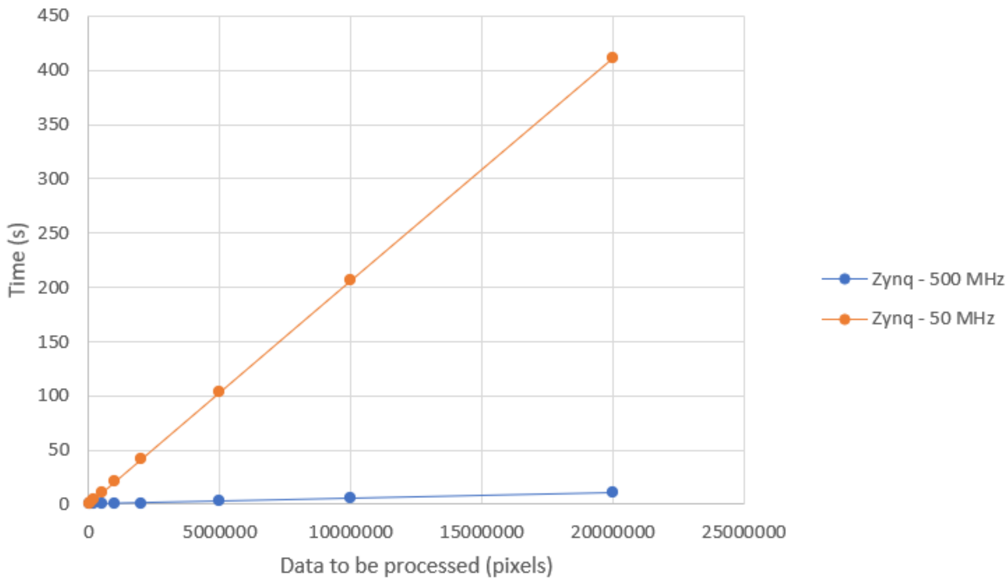


Figure 5.10: Decision Tree speed comparison between Zynq board and CPU

5.2.2 Minimum Distance

For the Minimum Distance Classifier the 20M samples could not be used because the demand of the classifier. The database selected was the same one used to build the Decision Tree as mentioned in the Accuracy Assessment section.

Again the time to load the variables into RAM it's not considered for the classification time.

In multiplications processing parallelism performs better than processing speed, the balance of these two variable will decide with implementation will benefit and achieve better results for this kind of classifiers. The I7 8750HQ is a hexa-core processor, having 12 processing units, comparing to the high parallelism of the Zynq board PL side, it's much less. So it's expected for the Zynq board to come up ahead.

The results of the processing time for both Zynq and CPU implementation are presented in figures 5.11 and 5.12 respectively.

From the figure it's clear to see that this classifier ir more suitable for the Zynq board. It takes advantage of its high parallelism and fast output of multiplications.

To perform the classification of 250k pixels, the processor takes an average of 4 times the time of the Zynq board. It is important to notice that the 250k pixels are being compared to a data base of 3M pixels. This database could be reduced to improve classification time. A good database will then be necessary preventing the accuracy to drop.

To compare the time dispense of this classifier, implemented with this database, to classify the same 20M pixels of the Decision Tree classifier it is expected to take close to 100 hours for the Zynq board and 420 hours for the CPU to complete the task.

Table 5.5: Minimum Distance: Time to Perform Classification - Zynq board

Nº of Pixels	Time to Perform Classification (s)
30	1.591
100	2.819
500	9.427
1000	18.682
5000	87.667
10000	174.2
20000	410.1
50000	928.1
100000	1799.8
150000	2701.3
200000	3598.9
250000	4478.4

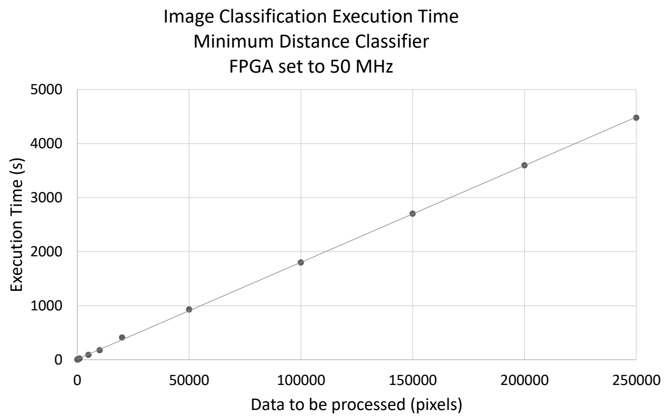


Figure 5.11: Image Classification Execution Time: Minimum Distance Classifier - FPGA set to 50 MHZ

Table 5.6: Minimum Distance: Time to Perform Classification - I7 8750HQ 3.9GHz

Nº of Pixels	Time to Perform Classification (s)
30	2.353
100	7.65
500	38.825
1000	76.62
5000	404.24
10000	786.44
20000	1557.3
50000	4082.6
100000	8404.2
150000	12084.7
200000	17525.6
250000	19059.1

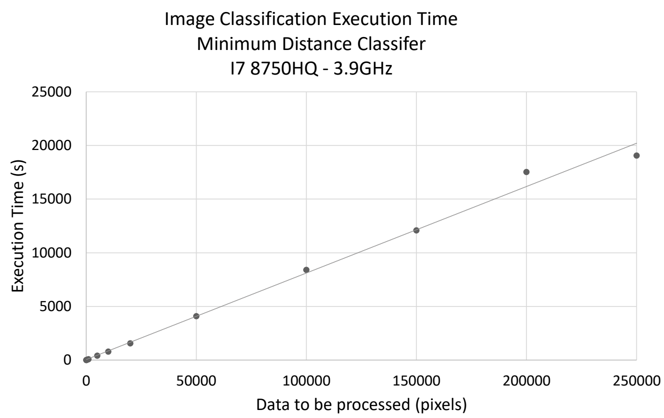


Figure 5.12: Image Classification Execution Time: Minimum Distance Classifier - I7 8750HQ 3.9GHz

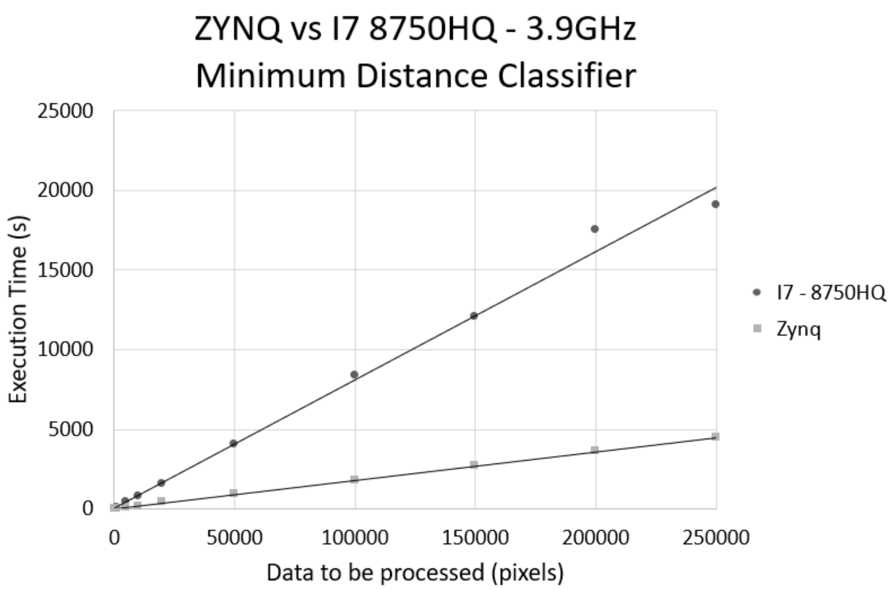


Figure 5.13: Minimum Distance speed comparison between Zynq board and CPU

Chapter 6

Conclusions and Future Work

Designing a project to be implemented in hardware it's not simple. Image processing is in itself a laborious and time consuming work. Optimizing the process of image processing, applied in this dissertation for Land Use/Land Cover classification in hardware takes a lot of time and resource consumption.

To accelerate the process of implementation Matlab gives a great support in this theme. Other type of implementations could be performed, such as restrict the software support to Xilinx SDK to implement all the processor programming and AXI Interconnections in Vivado but would required more time to implement and probably not so good results.

The outcome of the project presents two good classifiers, performing satisfactory results in the water/non water departments and further more, the expected accelerated performance for implementing in the Zynq board was acquired in the Minimum Distance classifier. For the majority of other classifiers, they comprehend any or a lot of calculations. From this approach it is concluded that the Zynq board will accelerate the classification processing time. More even, the power consumption of the Zynq board is much less that one normal CPU.

There are variables that should be taken into account, such as the loading of the data into the board. Even though the time was not taken into account in the classification process, if it was, for the Decision Tree classifier, the gap time between the two implementation would be even bigger. On the other hand, for the Minimum Distance classifier, the data took an average of 7 minutes to load into the board, still much faster than CPU. This is one aspect to be optimized in future work, the need of a higher baud rate. The implementation that will be approached is with ethernet transferring data between computer and board.

The comparison with the state of art classifiers is limited, since they are implemented to several classes and here are only two. Either way, the results are still according from what was seen in studied classifiers, Decision Tree performs better than Minimum Distance. Here by a slight difference but the tested data for the second one was must shorter, for a bigger evaluation data the accuracy would fall more. In future work is pretended to implement these two classifiers for the COS mega classes so a more accurate comparison could be made with other works. The continuity of the work will lead to more classifiers and is intended to accelerate the learning process in the board.

Also to be optimized, is the working frequency of the board. Register blocks will need to be implemented because some parts of the Simulink design cannot be processed within one clock cycle.

To sum up the work, the classifiers were implemented in the right direction, confirmed by the accuracy results. Moreover, the time to perform them was surpassed in the one expected.

Bibliography

[1] T. Skramstad. "Image Processing Methods Employed on Visual Quality Inspection Procedures Of Selected Industrial Applications." note. Doctoral dissertation. Norwegian Institute of Technology, June 1980, pp. 6–7.

[2] H. Liu, T.-H. Hong, M. Herman, T. Camus, and R. Chellappa. "Accuracy vs Efficiency Trade-offs in Optical Flow Algorithms." In: *Computer Vision and Image Understanding* 72.3 (1998), pp. 271–286. ISSN: 1077–3142. DOI: https://doi.org/10.1006/cviu.1998.0675. URL: http://www.sciencedirect.com/science/article/pii/S1077314298906750.

[3] B.-G. Jung, Y.-J. Cha, H.-H. Kim, S.-I. Jun, and J.-H. Cho. "Dynamic code binding for scalable operating system in distributed real-time systems." In: *Real-Time Computing Systems and Applications, 1995. Proceedings., Second International Workshop on*. IEEE. 1995, pp. 96–100.

[4] D. Sowmya, V. S. Hegde, J Suhas, R. V. Hegdekatte, P. D. Shenoy, and K. Venugopal. "Land Use/Land Cover Classification of Google Earth Imagery." In: *2017 IEEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE)*. IEEE. 2017, pp. 10–13.

[5] A. Di Gregorio. *Land Cover Classification System: Classification concepts, Software version 3*. Food & Agriculture Org., 2016. ISBN: 678-92-5-109017-6. URL: http://www.fao.org/3/a-i5232e.pdf.

[6] *Supervied Classification*. 2015. URL: http://gsp.humboldt.edu/olm_2015/courses/gsp_216_online/lesson6-1/supervised.html (visited on 02/03/2019).

[7] K. N. Priyadarshini, M. Kumar, S. A. Rahaman, and S NitheshNirmal. "A COMPARATIVE STUDY OF ADVANCED LAND USE/LAND COVER CLASSIFICATION ALGORITHMS USING SENTINEL–2 DATA." In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 42 (2018), p. 5.

[8] *Unsupervied Classification*. 2015. URL: http://gsp.humboldt.edu/OLM/Courses/GSP_216_Online/lesson6-1/unsupervised.html (visited on 02/03/2019).

[9] E. Adam, O. Mutanga, J. Odindi, and E. M. Abdel-Rahman. "Land-use/cover classification in a heterogeneous coastal landscape using RapidEye imagery: evaluating the performance of random forest and support vector machines classifiers." In: *International Journal of Remote Sensing* 35.10 (2014), pp. 3440–3458.

[10] K. Jia, J. Liu, Y. Tu, Q. Li, Z. Sun, X. Wei, Y. Yao, and X. Zhang. "Land use and land cover classification using Chinese GF-2 multispectral data in a region of the North China Plain." In: *Frontiers of Earth Science* (Dec. 2018). ISSN: 2095-0209. DOI: 10.1007/s11707-018-0734-8. URL: https://doi.org/10.1007/s11707-018-0734-8.

[11] A. Mora, T. M. A. Santos, S. Łukasik, J. M. N. Silva, A. J. Falcão, J. M. Fonseca, and R. A. Ribeiro. "Land Cover Classification from Multispectral Data Using Computational Intelligence Tools: A Comparative Study." In: *Information* 8.4 (2017). ISSN: 2078-2489. DOI: 10.3390/info8040147. URL: http://www.mdpi.com/2078-2489/8/4/147.

[12] N. A. Mahmon, N. Ya'Acob, and A. L. Yusof. "Differences of image classification techniques for land use and land cover classification." In: *Signal Processing & Its Applications (CSPA), 2015 IEEE 11th International Colloquium on*. IEEE. 2015, pp. 90–94.

[13] J Sharma, R Prasad, V. Mishra, V. Yadav, and R Bala. "Land Use and Land Cover Classification of Multispectral LANDSAT-8 Satellite Imagery Using Discrete Wavelet Transform." In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 42 (2018), p. 5.

[14] G. J. McLachlan. "Mahalanobis distance." In: *Resonance* 4.6 (1999), pp. 20–26.

[15] N. S. Altman. "An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression." In: *The American Statistician* 46.3 (1992), pp. 175–185. DOI: 10.1080/00031305.1992.10475879. eprint: https://www.tandfonline.com/doi/pdf/10.1080/00031305.1992.10475879. URL: https://www.tandfonline.com/doi/abs/10.1080/00031305.1992.10475879.

[16] J. A. Richards and J. Richards. *Remote sensing digital image analysis*. Vol. 3. Springer, 1999. ISBN: 978-3-642-30062-2.

[17] C. Cortes and V. Vapnik. "Support-vector networks." In: *Machine Learning* 20.3 (Sept. 1995), pp. 273–297. ISSN: 1573-0565. DOI: 10.1007/BF00994018. URL: https://doi.org/10.1007/BF00994018.

[18] J. A. Sobrino. *Fifth recent advances in quantitative remote sensing*. Universitat de València, 2018. ISBN: 978-84-9133-201-5.

[19] R. C. Weih and N. D. Riggan. "Object-based classification vs. pixel-based classification: Comparative importance of multi-resolution imagery." In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 38.4 (2010), p. C7.

[20] K. Wagstaff, C. Cardie, S. Rogers, S. Schrödl, et al. "Constrained k-means clustering with background knowledge." In: *ICML*. Vol. 1. 2001, pp. 577–584.

[21] T. M. Santos, A. Mora, R. A. Ribeiro, and J. M. Silva. "Fuzzy-fusion approach for land cover classification." In: *Intelligent Engineering Systems (INES), 2016 IEEE 20th Jubilee International Conference on*. IEEE. 2016, pp. 177–182.

[22] M. P. Temudo and J. M. Silva. "Agriculture and forest cover changes in post-war Mozambique." In: *Journal of Land Use Science* 7.4 (2012), pp. 425–442. DOI: 10.1080/1747423X.2011.595834. eprint: https://doi.org/10.1080/1747423X.2011.595834. URL: https://doi.org/10.1080/1747423X.2011.595834.

[23] F. Al-Ahmadi, A. Hames, et al. "Comparison of four classification methods to extract land use and land cover from raw satellite images for some remote arid areas, kingdom of Saudi Arabia." In: *Earth* 20.1 (2009), pp. 167–191.

[24] R. G. P. Jr and M. Millones. "Death to Kappa: birth of quantity disagreement and allocation disagreement for accuracy assessment." In: *International Journal of Remote Sensing* 32.15 (2011), pp. 4407–4429. DOI: 10.1080/01431161.2011.552923. eprint: https://doi.org/10.1080/01431161.2011.552923. URL: https://doi.org/10.1080/01431161.2011.552923.

[25] Y. Meyer. *Wavelets and operators*. Vol. 1. Cambridge university press, 1992. ISBN: 978-0521458696.

[26] L. S. Bernstein, S. M. Adler-Golden, X. Jin, B. Gregor, and R. L. Sundberg. "Quick atmospheric correction (QUAC) code for VNIR-SWIR spectral imagery: Algorithm details." In: *2012 4th Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS)*. June 2012, pp. 1–4. DOI: 10.1109/WHISPERS.2012.6874311.

[27] *COLORIMETRY — PART 4: CIE 1976 L*A*B* COLOUR SPACE*. International Commission on Illumination. URL: http://www.cie.co.at/publications/colorimetry-part-4-cie-1976-lab-colour-space (visited on 01/27/2019).

[28] Q Wua, R. Zhonga, W. Zhaoa, K Songa, and L. Dua. "Land-cover classification using multi-source data." In: *Fifth recent advances in quantitative remote sensing* (2018), p. 174.

[29] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An introduction to statistical learning*. Vol. 112. Springer, 2013.

[30] Z. Deng, X. Zhu, Q. He, and L. Tang. "Land use/land cover classification using time series Landsat 8 images in a heavily urbanized area." In: *Advances in Space Research* (2019). ISSN: 0273-1177. DOI: https://doi.org/10.1016/j.asr.2018.12.005. URL: http://www.sciencedirect.com/science/article/pii/S0273117718309037.

[31] T. Blaschke, S. Lang, and G. Hay. *Object-based image analysis: spatial concepts for knowledge-driven remote sensing applications*. Springer Science & Business Media, 2008.

[32] B. Melville, A. Lucieer, and J. Aryal. “Object-based random forest classification of Landsat ETM+ and WorldView-2 satellite imagery for mapping lowland native grassland communities in Tasmania, Australia.” In: *International Journal of Applied Earth Observation and Geoinformation* 66 (2018), pp. 46 –55. ISSN: 0303-2434. DOI: https://doi.org/10.1016/j.jag.2017.11.006. URL: http://www.sciencedirect.com/science/article/pii/S0303243417302659.

[33] G. M. Foody, N. Campbell, N. Trodd, and T. Wood. “Derivation and applications of probabilistic measures of class membership from the maximum-likelihood classification.” In: *Photogrammetric engineering and remote sensing* 58.9 (1992), pp. 1335–1341.

[34] C. Zhang, Z. Li, Q. Cheng, X. Li, and H. Shen. “Correction of"Cloud Removal By Fusing Multi-Source and Multi-Temporal Images".” In: *arXiv preprint arXiv:1707.09959* (2017).

[35] M. Li, S. C. Liew, and L. K. Kwoh. “Automated production of cloud-free and cloud shadow-free image mosaics from cloudy satellite imagery.” In: *Proc. of the XXth ISPRS Congress*. 2004, pp. 12–13.

[36] P. Singh and N. Komodakis. “Cloud-Gan: Cloud Removal for Sentinel-2 Imagery Using a Cyclic Consistent Generative Adversarial Networks.” In: *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*. July 2018, pp. 1772–1775. DOI: 10.1109/IGARSS.2018.8519033.

[37] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. “Generative adversarial nets.” In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[38] D. B. R. V. B. N. Manjunatha Reddy Dr. Shanthala S. “Performance Analysis of GPU V/S CPU for Image Processing Applications.” In: *International Journal for Research in Applied Science Engineering Technology (IJRASET)* 5 (II Feb. 2017). ISSN: 2321-9653. URL: https://www.ijraset.com/fileserve.php?FID=6250.

[39] J. Fowers, G. Brown, P. Cooke, and G. Stitt. “A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications.” In: *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*. ACM. 2012, pp. 47–56.

[40] S. Asano, T. Maruyama, and Y. Yamaguchi. “Performance comparison of FPGA, GPU and CPU in image processing.” In: *Field programmable logic and applications, 2009. fpl 2009. international conference on*. IEEE. Aug. 2009, pp. 126–131.

[41] D. B. Kirk and W. H. Wen-Mei. *Programming massively parallel processors: a hands-on approach*. Morgan kaufmann, 2016.

[42] C. Anderson, R. Archambault, R. Arimilli, R. Blainey, G. Blandy, H. Chase, B.-L. Chu, L. Clark, S. Fields, J. Kahle, J. Klockow, H. Le, h.-Y. McCreary, B. McCredie, J. McInnes, B. Mealey, C. Mehta, M. Nealon, K. Reick, E. Seminaro, J. Warnock, and D. Willoughby. *Power 4 - The First Multi-Core, 1GHz Processor*. Ed. by I. Corp. Aug. 2011. URL: https://www.ibm.com/ibm/history/ibm100/us/en/icons/power4/.

[43] D. B. Thomas, L. Howes, and W. Luk. "A comparison of CPUs, GPUs, FPGAs, and massively parallel processor arrays for random number generation." In: *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*. ACM. 2009, pp. 63–72.

[44] R. Y. Rubinstein and D. P. Kroese. *Simulation and the Monte Carlo method*. Vol. 10. John Wiley & Sons, 2016, pp. 187–188.

[45] T. Tiemerding, C. Diederichs, C. Stehno, and S. Fatikow. "Comparison of different design methodologies of hardware-based image processing for automation in microrobotics." In: *2013 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*. July 2013, pp. 565–570. DOI: 10.1109/AIM.2013.6584152.

[46] "IEEE Standard System C Language Reference Manual." In: *IEEE Std 1666-2005* (2006), pp. 1–2. DOI: 10.1109/IEEESTD.2006.99475.

[47] J. Stoppe and R. Drechsler. "Analyzing SystemC designs: SystemC analysis approaches for varying applications." In: *Sensors* 15.5 (2015), pp. 10399–10421.

[48] B. Cope, P. Y. K. Cheung, W. Luk, and S. Witt. "Have GPUs made FPGAs redundant in the field of video processing?" In: *Proceedings. 2005 IEEE International Conference on Field-Programmable Technology, 2005.* Dec. 2005, pp. 111–118. DOI: 10.1109/FPT.2005.1568533.

[49] M. Kiran, K. M. War, L. M. Kuan, L. K. Meng, and L. W. Kin. "Implementing image processing algorithms using 'Hardware in the loop' approach for Xilinx FPGA." In: *2008 International Conference on Electronic Design*. Dec. 2008, pp. 1–6. DOI: 10.1109/ICED.2008.4786653.

[50] M. A. Altuncu, T. Guven, Y. Becerikli, and S. Sahin. "Real-time system implementation for image processing with hardware/software co-design on the Xilinx Zynq platform." In: *International Journal of Information and Electronics Engineering* 5.6 (2015), p. 473.

[51] Y. Han and E. Oruklu. "Real-time traffic sign recognition based on zynq fpga and arm socs." In: *IEEE International Conference on Electro/Information Technology*. IEEE. 2014, pp. 373–376.

[52] M. Kowalczyk, D. Przewlocka, and T. Krvjak. "Real-Time Implementation of Contextual Image Processing Operations for 4K Video Stream in Zynq UltraScale+ MPSoC." In: *2018 Conference on Design and Architectures for Signal and Image Processing (DASIP)*. 2018, pp. 37–42. DOI: 10.1109/DASIP.2018.8597105.

[53] XILINX, ed. *Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit*. URL: https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html#overview (visited on 05/07/2019).

[54] XILINX, ed. *Zynq UltraScale+ MPSoC Product Tables and Product Selection Guide*. URL: https://www.xilinx.com/support/documentation/selection-guides/zynq-ultrascale-plus-product-selection-guide.pdf (visited on 07/17/2019).

[55] A. Taylor. *The Zynq PS/PL, Part One: Adam Taylor's MicroZed Chronicles Part 21*. Ed. by XILINX. 2014. URL: https://forums.xilinx.com/t5/Xcell-Daily-Blog-Archived/The-Zynq-PS-PL-Part-One-Adam-Taylor-s-MicroZed-Chronicles-Part/ba-p/418935.

[56] XILINX, ed. *AXI Reference Guide*. Version 13.1. Mar. 7, 2011. URL: https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf.

[57] Xilinx, ed. *Vivado Design Suite HLx Editions - Accelerating High Level Design*. URL: https://www.xilinx.com/products/design-tools/vivado.html (visited on 08/02/2019).

[58] Mathworks, ed. *MATLAB for FPGA, ASIC, and SoC Development*. URL: https://www.mathworks.com/solutions/fpga-asic-soc-development.html (visited on 08/02/2019).

[59] ESA, ed. *Access to Sentinel data via download*. URL: https://sentinel.esa.int/web/sentinel/sentinel-data-access (visited on 02/24/2019).

[60] D. G. do Território, ed. *Carta de Uso e Ocupação do Solo de Portugal Continental (COS)*. URL: http://www.dgterritorio.pt/dados_abertos/cos/ (visited on 03/02/2019).

[61] I. Nurwauziyah, U. Sulistyah, I Gede, I. G. B. Putra, and M. Firdaus. "Satellite Image Classification using Decision Tree, SVM and k-Nearest Neighbor." In: (July 2018).

[62] XILINX, ed. *Xilinx Software Development Kit (XSDK)*. URL: https://www.xilinx.com/products/design-tools/embedded-software/sdk.html (visited on 08/09/2019).

[63] MathWorks, ed. *Xilinx Zynq Support from HDL Coder*. URL: https://www.mathworks.com/hardware-support/zynq-hdl-coder.html (visited on 08/17/2019).

[64] MathWorks, ed. *HDL Verifier*. URL: https://www.mathworks.com/products/hdl-verifier.html (visited on 08/17/2019).

[65] MathWorks, ed. *Xilinx Zynq Support from Embedded Coder*. URL: https://www.mathworks.com/hardware-support/zynq-embedded-coder.html (visited on 08/17/2019).

[66] M. E. Hodgson. "Reducing the computational requirements of the minimum-distance classifier." In: *Remote Sensing of Environment* 25.1 (1988), pp. 117 –128. ISSN: 0034-4257. DOI: https://doi.org/10.1016/0034-4257(88)90045-4. URL: http://www.sciencedirect.com/science/article/pii/0034425788900454.

[67] H. Anton and C. Rorres. *Elementary Linear Algebra, Binder Ready Version: Applications Version*. 11th. John Wiley & Sons, 2013, p. 145.

[68] Xilinx, ed. *Vivado Design Suite User Guide - Designing IP Subsystems Using IP Integrator*. June 22, 2018. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_2/ug994-vivado-ip-subsystems.pdf.

[69] ArcGis, ed. *Accuracy Assessment for Image Classification - An overview of the accuracy assessment workflow*. URL: https://desktop.arcgis.com/en/arcmap/latest/manage-data/raster-and-images/accuracy-assessment-for-image-classification.htm (visited on 09/10/2019).

Annex I

Annex

Listing I.1: Load_Init_Data block code

```
function [load_done,ram_wr_data,ram_wr_add,ram_wr_enb] =
load_init(axi_wr_data,axi_wr_add,axi_wr_val, INIT_ADDR, DDR3_DEPTH,
  INIT_DDR3,INIT_DATA)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Create persistent variables (registers)
ddrWidth = 32;
persistent reg_ram_wr_add;
if(isempty(reg_ram_wr_add))
reg_ram_wr_add = fi(0,0,32,0);
end

persistent reg_ram_wr_data;
if(isempty(reg_ram_wr_data))
reg_ram_wr_data = fi(0,1,ddrWidth,0);
end

persistent reg_ram_wr_data_d1;
if(isempty(reg_ram_wr_data_d1))
reg_ram_wr_data_d1 = fi(0,1,ddrWidth,0);
end

persistent state;
if(isempty(state))
state = fi(0, 0, 3, 0);
end

persistent reg_ram_wr_enb;
if(isempty(reg_ram_wr_enb))
```

```
reg_ram_wr_enb = false;
end

persistent reg_ram_wr_enb_d1;
if(isempty(reg_ram_wr_enb_d1))
reg_ram_wr_enb_d1 = false;
end

persistent reg_init_data;
if(isempty(reg_init_data))
reg_init_data = fi(zeros(1,DDR3_DEPTH),1,ddrWidth,0);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% State Memory Encoding
INIT_ST    = fi(0, 0, 3, 0);
LOAD_START = fi(1, 0, 3, 0);
LOAD_PROG  = fi(3, 0, 3, 0);
LOAD_DONE  = fi(2, 0, 3, 0);
DATA_ST    = fi(6, 0, 3, 0);

% init phase states
% 1. Every state MUST assign every output value
% 2. Use explicit data type always
switch (state)
case INIT_ST

reg_ram_wr_add = fi(0,0,32,0);;
reg_ram_wr_data = fi(0,1,ddrWidth,0);
reg_ram_wr_enb = false;
load_done = false;

case LOAD_START

reg_ram_wr_add = fi(INIT_ADDR,0,32,0);
reg_ram_wr_data = fi(reg_init_data(1,reg_ram_wr_add+1),1,ddrWidth,0);
reg_ram_wr_enb = true;
load_done = false;

case LOAD_PROG

reg_ram_wr_add = fi(reg_ram_wr_add+1,0,32,0);
reg_ram_wr_data = fi(reg_init_data(1,reg_ram_wr_add+1),1,ddrWidth,0);
reg_ram_wr_enb = true;
load_done = false;
case LOAD_DONE

reg_ram_wr_add = fi(reg_ram_wr_add,0,32,0);

reg_ram_wr_enb = true;
```

```
load_done = true;
case DATA_ST
%external (AXI ) inputs
reg_ram_wr_add = fi(axi_wr_add,0,32,0);
reg_ram_wr_data =  axi_wr_data;
reg_ram_wr_enb = axi_wr_val;
load_done = true;
otherwise

reg_ram_wr_add = fi(0,0,32,0);
reg_ram_wr_data = fi(0,1,ddrWidth,0);
reg_ram_wr_enb = false;
load_done = false;
end

ram_wr_add = fi(reg_ram_wr_add,0,32,0);

if(load_done == true)
ram_wr_data = fi(reg_ram_wr_data_d1,1,ddrWidth,0);
else
ram_wr_data = fi(reg_ram_wr_data_d1,1,ddrWidth,0);
end

if(load_done == true)
ram_wr_enb = reg_ram_wr_enb_d1;
else
ram_wr_enb = reg_ram_wr_enb;
end

if(load_done == false)
reg_init_data = fi(INIT_DATA,1,ddrWidth,0);
end

reg_ram_wr_enb_d1 = reg_ram_wr_enb;
%reg_ram_wr_data_d1 = reg_ram_wr_data;
reg_ram_wr_data_d1 = fi(reg_ram_wr_data,1,ddrWidth,0);
% Next State Logic
switch (state)
case INIT_ST

if ( INIT_DDR3 == true)
nextState = LOAD_START;
else
nextState = DATA_ST;
end
case LOAD_START
nextState = LOAD_PROG;
case LOAD_PROG
if(fi(ram_wr_add,0,32,0)==fi(DDR3_DEPTH-1,0,32,0))
nextState = LOAD_DONE;
```

```
else
nextState = LOAD_PROG;
end
case LOAD_DONE
nextState = DATA_ST;
case DATA_ST
nextState = DATA_ST;
otherwise
nextState = INIT_ST;
end

state = nextState;

end
```

Listing I.2: DDR_Read_Controller

```

function [valid_out, count_out, ddr_read_done, rd_addr, rd_len, rd_avalid] = ...
hdlcoder_external_memory_read_ctrl(burst_len, start, rd_aready, rd_dvalid)
%

%   Copyright 2017 The MathWorks, Inc.

% create persistent variables (registers)
persistent rstate burst_stop burst_count
if isempty(rstate)
rstate      = fi(0, 0, 4, 0);
burst_stop  = uint32(0);
burst_count = uint32(0);
end

% state Memory Encoding
IDLE               = fi(0, 0, 4, 0);
READ_BURST_START   = fi(1, 0, 4, 0);
READ_BURST_REQUEST = fi(2, 0, 4, 0);
DATA_COUNT         = fi(3, 0, 4, 0);

% state machine logic
switch (rstate)
case IDLE
% output to AXI4 Master
rd_addr   = uint32(0);
rd_len    = uint32(0);
rd_avalid = false;

% output to DUT logic
valid_out = false;
count_out = uint32(0);
ddr_read_done = true;

% State vars
burst_stop  = uint32(burst_len);
burst_count = uint32(0);

if start
rstate(:) = READ_BURST_START;
else
rstate(:) = IDLE;
end

case READ_BURST_START
% output to AXI4 Master
rd_addr   = uint32(0);
rd_len    = uint32(burst_stop);
rd_avalid = false;
```

```

% output to DUT logic
valid_out = false;
count_out = uint32(0);
ddr_read_done = false;

if rd_aready
rstate(:) = READ_BURST_REQUEST;
else
rstate(:) = READ_BURST_START;
end

case READ_BURST_REQUEST
% output to AXI4 Master
rd_addr   = uint32(0);
rd_len    = uint32(burst_stop);
rd_avalid = true;

% output to DUT logic
valid_out = false;
count_out = uint32(0);
ddr_read_done = false;

rstate(:) = DATA_COUNT;

case DATA_COUNT
% output to AXI4 Master
rd_addr   = uint32(0);
rd_len    = uint32(burst_stop);
rd_avalid = false;

% output to DUT logic
valid_out = rd_dvalid;
count_out = uint32(burst_count);
ddr_read_done = false;

% State vars
if ( rd_dvalid )
burst_count = uint32(burst_count + 1);
end

if ( burst_count == burst_stop )
rstate(:) = IDLE;
else
rstate(:) = DATA_COUNT;
end

otherwise
% output to AXI4 Master
rd_addr   = uint32(0);
```

```
rd_len     = uint32(0);
rd_avalid = false;

% output to DUT logic
valid_out = false;
count_out = uint32(0);
ddr_read_done = false;

rstate(:) = IDLE;
end

end
```

Listing I.3: DDR_Write_Controller

```

function [ram_addr, ddr_write_done, wr_addr, wr_len, wr_valid] = ...
hdlcoder_external_memory_write_ctrl(burst_len, start, wr_ready, wr_complete)
%

%   Copyright 2017 The MathWorks, Inc.

% create persistent variables (registers)
persistent wstate burst_stop burst_count
if(isempty(wstate))
wstate      = fi(0, 0, 4, 0);
burst_stop  = uint32(0);
burst_count = uint32(0);
end

% state machine encoding
IDLE              = fi(0, 0, 4, 0);
WRITE_BURST_START = fi(1, 0, 4, 0);
DATA_COUNT        = fi(2, 0, 4, 0);
ACK_WAIT          = fi(3, 0, 4, 0);

% state machine logic
switch (wstate)
case IDLE
% output to AXI4 Master
wr_addr  = uint32(0);
wr_len   = uint32(0);
wr_valid = false;

% output to DUT logic
ram_addr = uint32(0);
ddr_write_done = true;

% state variables
burst_stop  = uint32(burst_len);
burst_count = uint32(0);

if start
wstate(:) = WRITE_BURST_START;
else
wstate(:) = IDLE;
end


case WRITE_BURST_START
% output to AXI4 Master
wr_addr  = uint32(0);
wr_len   = uint32(burst_stop);
wr_valid = false;
```

```

% output to DUT logic
ram_addr = uint32(burst_count);
ddr_write_done = false;

if wr_ready
wstate(:) = DATA_COUNT;
else
wstate(:) = WRITE_BURST_START;
end


case DATA_COUNT
% output to AXI4 Master
wr_addr  = uint32(0);
wr_len   = uint32(burst_stop);
wr_valid = true;

% state variables
burst_count = uint32(burst_count + 1);

% output to DUT logic
ram_addr = uint32(burst_count);
ddr_write_done = false;

if ( burst_count == burst_stop )
wstate(:) = ACK_WAIT;
else
if ( wr_ready )
wstate(:) = DATA_COUNT;
else
wstate(:) = WRITE_BURST_START;
end
end


case ACK_WAIT
% output to AXI4 Master
wr_addr  = uint32(0);
wr_len   = uint32(0);
wr_valid = false;

% output to DUT logic
ram_addr = uint32(0);
ddr_write_done = false;

if wr_complete
wstate(:) = IDLE;
else
wstate(:) = ACK_WAIT;
```

```
end

otherwise
% output to AXI4 Master
wr_addr = uint32(0);
wr_len = uint32(0);
wr_valid = false;

% output to DUT logic
ram_addr = uint32(0);
ddr_write_done = false;

wstate(:) = IDLE;

end

end

% LocalWords:  AXI
```

Listing I.4: Decision Tree Classifier Function

```

function [classification, enable] =  classifier(data_in_enable, data_in)

classification = int32(0);
enable = int32(0);

persistent counter;
if(isempty(counter))
counter =  int32(0);
end

persistent b1;
if(isempty(b1))
b1 =  int32(0);
end

persistent b2;
if(isempty(b2))
b2 =  int32(0);
end

persistent b3;
if(isempty(b3))
b3 =  int32(0);
end

persistent b4;
if(isempty(b4))
b4 =  int32(0);
end

persistent b5;
if(isempty(b5))
b5 =  int32(0);
end

persistent b6;
if(isempty(b6))
b6 =  int32(0);
end

persistent b7;
if(isempty(b7))
b7 =  int32(0);
end

persistent b8;
if(isempty(b8))
b8 =  int32(0);
```

```
end

persistent b9;
if(isempty(b9))
b9 =  int32(0);
end

persistent b10;
if(isempty(b10))
b10 =  int32(0);
end

persistent b11;
if(isempty(b11))
b11 =  int32(0);
end

persistent b12;
if(isempty(b12))
b12 =  int32(0);
end

%count til number of input parameters
if data_in_enable ~= 0
counter = counter + 1;


%load data from RAM into the variables
switch counter
case 1
b1 = data_in;
case 2
b2 = data_in;
case 3
b3 = data_in;
case 4
b4 = data_in;
case 5
b5 = data_in;
case 6
b6 = data_in;
case 7
b7 = data_in;
case 8
b8 = data_in;
case 9
b9 = data_in;
case 10
b10 = data_in;
case 11
```

```
b11 = data_in;
case 12
b12 = data_in;
enable = int32(1);
end

%apply the DT classifier after the number of input cycles correspondent to
%the number of variables

if counter == 12
if b10 < 12912200
if b12 <= 6892
if b12 <= 6245
if b11 <= 9405320
if b12 <= 5751
if b1 <= 21847100
classification =int32(0);
else % b1 > 21847100
classification = int32(1);
end
else % b12 > 5751
if b10 <= 11990300
classification =int32(0);
else % b10 > 11990300
if b5 <= 7899690
classification = int32(1);
else % b5 > 7899690
classification =int32(0);
end
end
end
else % b11 > 9405320
if b5 <= 7940940
if b9 <= 16044000
classification = int32(1);
else % b9 > 16044000
classification =int32(0);
end
else % b5 > 7940940
classification =int32(0);
end
end
else % b12 > 6245
if b10 <= 10917200
if b10 <= 9876560
classification =int32(0);
else % b10 > 9876560
if b5 <= 690750
if b7 <= 16388800
classification =int32(0);
```

```
else  % b7 > 16388800
classification = int32(1);
end
else %b5 > 690750
classification =int32(0);
end
end
else % b10 > 10917200
if b5 <= 7933750
if b9 <= 17023500
if b9 <= 5639060
classification = int32(1);
else % b9 > 5639060
classification =int32(0);
end
else % b9 > 17023500
if b9 <= 25391800
classification = int32(1);
else % b9 > 25391800
classification =int32(0);
end
end
else % b5 > 7933750
if b5 <= 8704060
if b7 <= 19826600
classification =int32(0);
else % b7 > 19826600
classification = int32(1);
end
else % b5 > 8706060
classification =int32(0);
end
end
end
end
else % b12 > 6892
if b3 <= 4865000
if b9 <= 16753800
if b10 <= 10145300
classification =int32(0);
else % b10 > 10145300
if b5 <= 5410310
classification = int32(1);
else % b5 > 5410310
classification =int32(0);
end
end
else % b9 > 16753800
if b10 <= 8058440
if b5 <= 7639060
```

```
classification =int32(0);
else % b5 > 7639060
classification = int32(1);
end
else % b10 > 8058400
if b5 <= 7639060
classification =int32(0);
else % b5 > 7639060
if b8 <= 21735000
classification =int32(0);
else % b8 > 21735000
classification = int32(1);
end
end
end
end
else % b3 > 4865000
if b7 <= 35499700
if b9 <= 24126900
if b3 <= 5295000
if b10 <= 105156
classification =int32(0);
else % b10 > 105156
classification = int32(1);
end
else % b3 > 5295000
classification =int32(0);
end
else % b9 > 24125900
if b5 <= 8256880
if b7 <= 21752800
classification =int32(0);
else % b7 > 21752800
classification = int32(1);
end
else % b5 > 8256880
classification =int32(0);
end
end
else % b7> 35499700
if b9 <= 31521400
if b3 <= 5245000
classification = int32(1);
else % b3 > 524500
classification =int32(0);
end
else % b9 > 31521400
classification = int32(1);
end
end
```

```
end
end
else % b10 > 12912200
if b9 <= 18090900
if b2 <= 3055000
if b9 <= 16516100
if b4 <= 6225000
if b5 <= 6432190
if b7 <= 12295300
classification =int32(0);
else % b7 > 12295300
classification = int32(1);
end
else % b5 > 6432190
classification =int32(0);
end
else % b4 > 6225000
classification = int32(1);
end
else % b9 > 16516100
if b5 <= 7672190
if b7 <= 12863400
classification =int32(0);
else % b7 > 12863400
classification = int32(1);
end
else % b5 > 7672190
if b11 <= 9221560
classification =int32(0);
else % b11 > 9221560
classification = int32(1);
end
end
end
else % b2 > 3055000
if b11 <= 12854700
if b10 <= 16240900
if b2 <= 3465000
if b11 <= 10921600
classification =int32(0);
else % b11 > 10921600
classification = int32(1);
end
else % b2 > 3465000
classification =int32(0);
end
else % b10 > 16240900
if b9 <= 16802600
classification =int32(0);
else % b9 > 16802600
```

```
if b5 <= 8820000
classification = int32(1);
else % b5 > 8820000
classification =int32(0);
end
end
end
else % b11 > 12854700
if b8 <= 18425000
if b11 <= 17395900
if b5 <= 13470000
classification = int32(1);
else % b5 > 13470000
classification =int32(0);
end
else % b11 > 17395900
classification = int32(1);
end
else % b8 > 18425000
classification =int32(0);
end
end
end
else % b9 > 18090900
if b12 <= 5587
if b10 <= 17057200
if b5 <= 9499690
if b7 <= 15565300
classification =int32(0);
else % b7 > 15565300
if b11 <= 8787810
classification =int32(0);
else % b11 > 8787810
classification = int32(1);
end
end
else % b5 > 9499690
classification =int32(0);
end
else % b10 > 17057200
if b9 <= 22435900
if b6 <= 21110300
classification = int32(1);
else % b6 > 21110300
classification =int32(0);
end
else % b9 > 22435900
if b7 <= 17992800
if b7 <= 16450900
classification =int32(0);
```

```
else % b7 > 16450900
classification = int32(1);
end
else % b7 > 17992800
if b12 <= 964
classification =int32(0);
else % b12 > 964
classification = int32(1);
end
end
end
end
else % b12 > 5587
if b1 <= 3223220
if b5 <= 8629060
if b8 <= 13385000
classification = int32(1);
else % b8 > 13385000
if b5 <= 7631560
classification = int32(1);
else % b5 > 7631560
classification = int32(1);
end
end
else % b5 > 8629060
if b10 <= 14491600
if b12 <= 7934
classification =int32(0);
else % b12 > 7934
classification = int32(1);
end
else % b10 > 14491600
if b9 <= 19777400
classification = int32(1);
else % b9 > 19777400
classification = int32(1);
end
end
end
else % b1 > 3223220
if b10 <= 15793400
if b5 <= 9037810
classification = int32(1);
else % b5 > 9037810
if b7 <= 40633100
classification =int32(0);
else % b7 > 40633100
classification = int32(1);
end
end
```

```
else % b10 > 15793400
classification = int32(1);
end
end
end
end
end
end

if counter == 12
counter = int32(0);
end

end

end
```

Listing I.5: Minimum Distance Classifier Function

```
function [classificationAux, EnableOut] =
  classifiertry(DataIn, Enable, DB_Size, Data_Size)

EnableOut = int32(0);

persistent classification
if(isempty(classification))
classification = int32(0);
end

persistent state
if(isempty(state))
state = int32(1);
end

persistent classificationON
if(isempty(classificationON))
classificationON = int32(0);
end

persistent nextAddr
if(isempty(nextAddr))
nextAddr = int32(0);
end

persistent counter
if(isempty(counter))
counter = int32(1);
end

persistent distance
if(isempty(distance))
distance = int32(10000000);
end

persistent newClassification
if(isempty(newClassification))
newClassification = int32(0);
end

persistent counterDB
if(isempty(counterDB))
counterDB = int32(0);
end

persistent counterData
if(isempty(counterData))
counterData = int32(0);
```

```
end

persistent saveAddr
if(isempty(saveAddr))
saveAddr = int32(0);
end

%% Variable bands
persistent b1_DB;
if(isempty(b1_DB))
b1_DB =  int32(0);
end

persistent b2_DB;
if(isempty(b2_DB))
b2_DB =  int32(0);
end

persistent b3_DB;
if(isempty(b3_DB))
b3_DB =  int32(0);
end

persistent b4_DB;
if(isempty(b4_DB))
b4_DB =  int32(0);
end

persistent b5_DB;
if(isempty(b5_DB))
b5_DB =  int32(0);
end

persistent b6_DB;
if(isempty(b6_DB))
b6_DB =  int32(0);
end

persistent b7_DB;
if(isempty(b7_DB))
b7_DB =  int32(0);
end

persistent b8_DB;
if(isempty(b8_DB))
b8_DB =  int32(0);
end

persistent b9_DB;
if(isempty(b9_DB))
```

```
b9_DB =  int32(0);
end

persistent b10_DB;
if(isempty(b10_DB))
b10_DB =  int32(0);
end

persistent b11_DB;
if(isempty(b11_DB))
b11_DB =  int32(0);
end

persistent b12_DB;
if(isempty(b12_DB))
b12_DB =  int32(0);
end

persistent b13_DB;
if(isempty(b13_DB))
b13_DB =  int32(0);
end

%---------------
persistent b1_Data;
if(isempty(b1_Data))
b1_Data =  int32(0);
end

persistent b2_Data;
if(isempty(b2_Data))
b2_Data =  int32(0);
end

persistent b3_Data;
if(isempty(b3_Data))
b3_Data =  int32(0);
end

persistent b4_Data;
if(isempty(b4_Data))
b4_Data =  int32(0);
end

persistent b5_Data;
if(isempty(b5_Data))
b5_Data =  int32(0);
end

persistent b6_Data;
```

```
if(isempty(b6_Data))
b6_Data =  int32(0);
end

persistent b7_Data;
if(isempty(b7_Data))
b7_Data =  int32(0);
end

persistent b8_Data;
if(isempty(b8_Data))
b8_Data =  int32(0);
end

persistent b9_Data;
if(isempty(b9_Data))
b9_Data =  int32(0);
end

persistent b10_Data;
if(isempty(b10_Data))
b10_Data =  int32(0);
end

persistent b11_Data;
if(isempty(b11_Data))
b11_Data =  int32(0);
end

persistent b12_Data;
if(isempty(b12_Data))
b12_Data =  int32(0);
end


%% State Machine
if Enable ~= 0
switch state

case 1
switch counter
case 1
b1_Data = DataIn;
counter = counter + int32(1);
case 2
b2_Data = DataIn;
counter = counter + int32(1);
case 3
b3_Data = DataIn;
counter = counter + int32(1);
```

```
case 4
b4_Data = DataIn;
counter = counter + int32(1);
case 5
b5_Data = DataIn;
counter = counter + int32(1);
case 6
b6_Data = DataIn;
counter = counter + int32(1);
case 7
b7_Data = DataIn;
counter = counter + int32(1);
case 8
b8_Data = DataIn;
counter = counter + int32(1);
case 9
b9_Data = DataIn;
counter = counter + int32(1);
case 10
b10_Data = DataIn;
counter = counter + int32(1);
case 11
b11_Data = DataIn;
counter = counter + int32(1);
case 12
b12_Data = DataIn;
state = int32(2);
counter = int32(1);
end

case 2
switch counter
case 1
b1_DB = DataIn;
counter = counter + int32(1);
case 2
b2_DB = DataIn;
counter = counter + int32(1);
case 3
b3_DB = DataIn;
counter = counter + int32(1);
case 4
b4_DB = DataIn;
counter = counter + int32(1);
case 5
b5_DB = DataIn;
counter = counter + int32(1);
case 6
b6_DB = DataIn;
counter = counter + int32(1);
```

```
case 7
b7_DB = DataIn;
counter = counter + int32(1);
case 8
b8_DB = DataIn;
counter = counter + int32(1);
case 9
b9_DB = DataIn;
counter = counter + int32(1);
case 10
b10_DB = DataIn;
counter = counter + int32(1);
case 11
b11_DB = DataIn;
counter = counter + int32(1);
case 12
b12_DB = DataIn;
counter = counter + int32(1);
case 13
b13_DB = DataIn;
state = int32(3);
end
end

if state == int32(3)
newDistance = int32((b1_Data - b1_DB)*(b1_Data - b1_DB) +
  (b2_Data - b2_DB)*(b2_Data - b2_DB) +
  (b3_Data - b3_DB)*(b3_Data - b3_DB) +
  (b4_Data - b4_DB)*(b4_Data - b4_DB) +
  (b5_Data - b5_DB)*(b5_Data - b5_DB) +
  (b6_Data - b6_DB)*(b6_Data - b6_DB) +
  (b7_Data - b7_DB)*(b7_Data - b7_DB) +
  (b8_Data - b8_DB)*(b8_Data - b8_DB) +
  (b9_Data - b9_DB)*(b9_Data - b9_DB) +
  (b10_Data - b10_DB)*(b10_Data - b10_DB) +
  (b11_Data - b11_DB)*(b11_Data - b11_DB) +
  (b12_Data - b12_DB)*(b12_Data - b12_DB));
if newDistance < distance
distance = int32(newDistance);
newClassification = b13_DB;
end

counterDB = counterDB + int32(1);
if counterDB == DB_Size
counterDB = int32(0);
counterData = counterData + int32(1);
distance = int32(10000000);
EnableOut = int32(1);
classification = newClassification;
if counterData == Data_Size
```

```
state = int32(1);
counterData = int32(0);
counter = int32(0);
else
counter = int32(1);
state = int32(1);
end
else
counter = int32(1);
state = int32(2);
end
end


end

classificationAux = classification;
end
```