Dordije Boskovic

Hardware implementation of a target detection algorithm for hyperspectral images

Master's thesis in Embedded Computing Systems

June 2019

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Dordije Boskovic

Hardware implementation of a target detection algorithm for hyperspectral images

Master's thesis in Embedded Computing Systems
Supervisor: Kjetil Svarstad and Milica Orlandic
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Abstract

Hyper-Spectral Imager for Oceanographic Applications (HYPSO) is being developed as a part of SmallSat laboratory at NTNU. The satellite capable of capturing and processing of hyperspectral images will be equipped with Zynq-7000 on-board processing system consisting of ARM®-based processor with the hardware programmability of an FPGA. FPGA offers inherent reconfigurability, smaller size and weight, substantially lower power consumption and reduced costs, compared to its counterpart in other technologies. In this thesis, a part of on-board hyperspectral processing in FPGA is explored.

Hyperspectral images obtained by imaging spectrometer contain a vast amount of data which require techniques such as target detection to extract useful information. This thesis presents implementations of target detection algorithms for hyperspectral images. The algorithms are implemented as hardware-software partitioned system on Xilinx Zynq-7000 development platform. Prior to FPGA implementation, the algorithms such as ACE, SAM, CEM and ASMF were reviewed and tested on 5 different hyperspectral datasets. The detection performance of the algorithms was evaluated using MCC score, visibility score and ROC curves.

Two FPGA solutions for target detection are presented: HW/SW partitioned design and full FPGA solution. The designs were modelled using VHDL in attempt to gain the most optimal solution for our application. The operations of algorithms are partitioned on the heterogeneous platform between processing system and programmable logic with special consideration of background estimation. In HW/SW partitioned design, background estimation is performed prior to detection statistic calculation, while in full-FPGA implementation background is estimated in real-time as image frames are captured using the imager. Both solutions provide certain advantages depending on the desired application. Algorithm modelling and testing was performed in MATLAB environment, while FPGA modules were synthesized using Xilinx Vivado Design Suite.

In addition to target detection algorithm implementations, PPI endmember extraction algorithm was implemented using Vivado HLS. This implementation illustrates productivity benefits of a C-based development flow using HLS.

Preface

This master's thesis is the final part of my Master of Science degree in Embedded Computing Systems (EMECS). The thesis work was conducted at the Norwegian University of Science and Technology within a SmallSat project (HYPSO). It is an exploratory work concerning topics such as hyperspectral remote sensing and FPGA design. As I have not been introduced to hyperspectral imaging prior to the specialization and thesis work, it has been a challenging learning experience, but also a gratifying and rewarding practice which has greatly improved my research skills.

First of all, I would like to thank my supervisor Milica Orlandić for her support, encouragement and great guidance. Our work together resulted in my first conference paper, awarded as the best student paper. Furthermore, I would like to thank my fellow students in the SmallSat lab for creating supportive environment in which one could pleasantly work and socialize. Special thanks to my family and friends for their continuous support throughout the years.

Đorđije Bošković
June, 2019

Table of Contents

List of Tables

List of Figures

Abbreviations

ACE	=	Adaptive Coherence/Cosine Estimator
ASMF	=	Adjusted Spectral Matched Filter
AXI	=	Advanced eXtensible Interface
CEM	=	Constrained Energy Minimization
DMA	=	Direct Memory Access
DSP	=	Digital Signal Processing
FPGA	=	Field Programmable Gate Array
GPU	=	Graphic Processing Unit
HLS	=	High-Level Synthesis
HSI	=	Hyperspectral Imager
HW	=	Hardware
HYPSO	=	Hyper-Spectral Imager for Oceanographic Applications
MAC	=	Multiplier–Accumulator
MCC	=	Matthews Correlation Coefficient
PL	=	Programmable Logic
PS	=	Processing System
SAM	=	Spectral Angle Mapper
SoC	=	System on Chip
SW	=	Software
VHDL	=	VHSIC Hardware Description Language

Chapter 1

Introduction

1.1 Motivation

The ocean covers more than 70% of the Earth's surface. This vast body of water is a principal component of life and home to the first organisms on Earth. Many years later, the ocean plays an integral role in the human economy, society and most importantly survival. The human kind is facing the world-changing challenges such as climate change and global warming, waiting for an extensive solution for the years to come. This is where HYPSO mission makes its impact on our everyday life.

Hyper-Spectral Imager for Oceanographic Applications (HYPSO) mission is being developed at the NTNU, Trondheim. The imager will observe the oceanographic phenomena by using a small satellite equipped with a hyperspectral camera on-board [1], operating in cooperation with aerial, surface, and underwater vehicles. This approach is employed since a variety of phenomena can be observed from the space on a large scale, such as ocean color data or harmful algae blooms approaching fish farms as shown in Fig. 1.1. This master thesis is a part of the HYPSO mission.

1.2 Hyperspectral imaging and processing

The hyperspectral camera captures hundreds of images, where each corresponds to a certain wavelength range in the electromagnetic spectrum, for the same area on Earth. This is the basis of hyperspectral imaging, which exquisitely combines remote sensing and spectrometry. In contrast to true color imaging which is adjusted to human spectral sensitivity, hyperspectral imaging can include abundance of wavelength channels outside and inside the visible spectrum. Such channels are usually referred to as bands.

The increasing amount of spatio-spectral data obtained using modern hyperspectral imagers (HSI) has brought in new challenges in the analysis and extraction of useful information from hyperspectral datasets, especially in scenarios which require real-time operation. Modern satellite missions tend to incorporate new HSI with increased spatial and

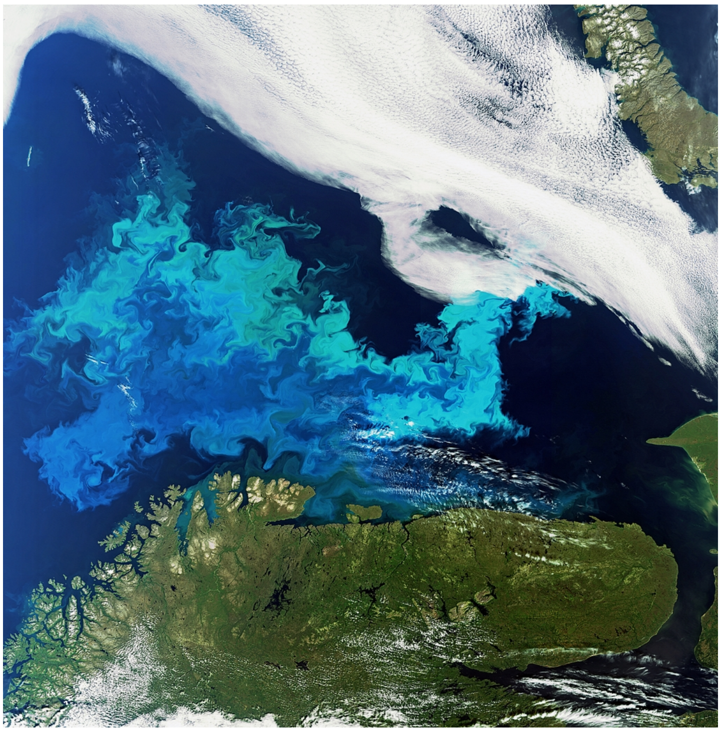


Figure 1.1: The satellite image shows an algae bloom in the sea off the coast of Northern Norway [2]. This rapid increase of algae population in marine water can be harmful and have significant negative impact on human health, economy and environment.

temporal resolution, causing dramatic growth of hyperspectral data and its dimensionality. Unfortunately, the available satellite down-link bandwidth to ground stations is not following this trend [3]. Those reasons lead to the necessary incorporation of high-performance computing platforms into certain stages of hyperspectral processing on-board and on the ground, such as: graphic processing units (GPUs), parallel computing and most importantly field programmable gate arrays (FPGAs) [4]. The hyperspectral data processing is usually pipelined, consisting of the following stages: binning, optical and sensor corrections, radiometric corrections, geo-referencing and registration, motion blur correction, super-resolution [5], atmospheric correction, dimensionality reduction [6], classification, target detection and compression [7, 8]. This thesis explores the target detection algorithms and how they could be implemented on FPGA.

1.3 Target Detection in context of HYPSO mission

The objective of target detection algorithms is to find an object of interest in the hyperspectral image, i.e, target detection algorithms are inspecting the presence of a specific material in the image. As such, they require spectral information about the target of interest obtained either from spectral libraries or extracted from the scene. With regard to HYPSO mission, target detection is the main part of mission objectives [1] listed below:

- To provide and support ocean color mapping through a Hyperspectral Imager (HSI) payload, **autonomously processed data**, and on-demand autonomous communications in a concert of robotic agents at the Norwegian coast;
- To collect ocean color data and **to detect and characterize spatial extent of algal blooms**, measure primary productivity using emittance from fluorescence-generating micro-organisms, and other substances resulting from aquatic habitats and pollution to support environmental monitoring, climate research and marine resource management;

as well as user and mission requirements:

- Operational data shall be compressed, have at least 20 spectral bands, and include radiometric calibration, atmospheric correction, classification, super-resolution and **target detection**;
- Should **determine** many specific plant pigments as well as certain organic and inorganic compounds.

The algorithms inspected in this thesis are based on the statistical approach, where spectral reflectance features are exploited to identify the target. The concept of many target detection algorithms has been established in the domain of signal processing, radar and pattern recognition [9]. Later, the algorithms have been adapted for hyperspectral processing, with some originally developed in this field due to the amount of spectral detail [10]. As such, hyperspectral target detection algorithms are usually affiliated with spectral rather than spatial processing techniques. In the domain of spectral processing, spatial arrangement of pixels and geometrical shapes are not carrying any information. Instead, each pixel corresponds to a certain ground-resolution cell containing a spectrum used to determine specific materials [11].

In statistical signal processing, target detection is regarded as binary hypothesis testing between a null and alternative hypothesis. A null hypothesis asserts that the pixel being tested is not a target, while an alternative hypothesis asserts the opposite. This way, each pixel is either regarded as a target pixel or something other than a target, which is referred to as background [12]. Modelling the signals under both hypothesis is different and characteristic for each target detection algorithm.

Target detection algorithms come with a set of challenges related to their application in a certain field, such as detection of plant pigments. Their performance is limited by many factors, while some of them are:

- Spatial extent of the target in the hyperspectral image
- Number of spectral bands used for target detection
- Appropriate usage of radiance or reflectance domain in target detection
- Estimation and modeling of the image background for optimal detection

- Selection of threshold for automated target detection
- Spectral variability of the target due to spectral mixing in a subpixel target

In this thesis, target detection algorithms are implemented in hardware, specifically designed for FPGA platform, and therefore, not only concerns about the detection performance of the algorithm arise. In addition to that, the algorithm has to perform in real-time with specific hardware platform constraints accounted for. The algorithms and tests performed are further explained in section 2.3 and chapter 3.

1.4 HYPSO mission payload

At NTNU SmallSat laboratory, we are developing HYPSO payload which will be able to capture and process a hyperspectral image. The camera used in HYPSO payload is a push-broom scanner, where the image is acquired one line (frame) at a time, as shown in Fig. 1.2. Each frame has two dimensions, one spatial, telling the position of captured pixel, and one

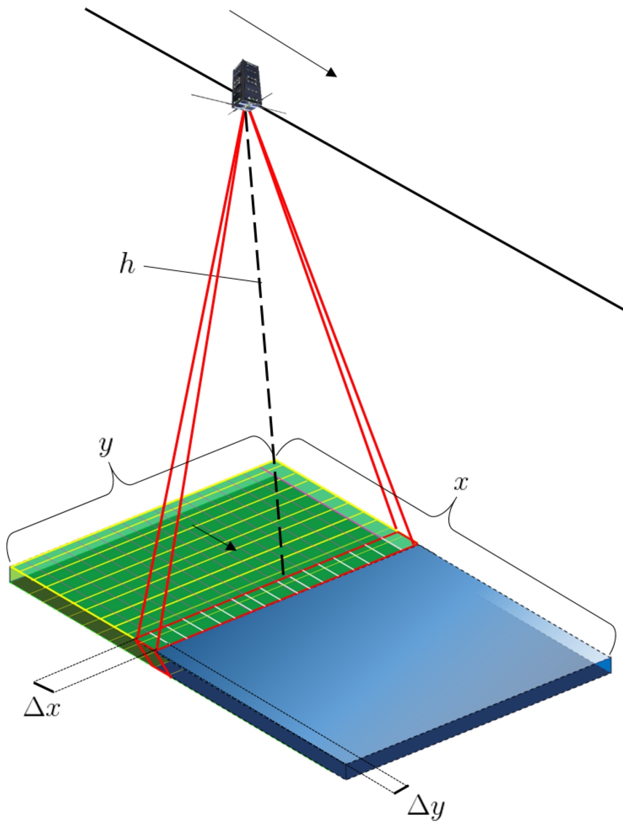


Figure 1.2: Push-broom scanner concept [1].

spectral telling the pixel intensity at each wavelength channel. The hyperspectral image is built from top to bottom, frame by frame. Therefore, the image is usually represented as a three-dimensional cube (two spatial and one spectral dimension).

To process hyperspectral data, HYPSO payload will be equipped with Xilinx Zynq-7000 All Programmable System on Chip (SoC). This SoC contains ARM Cortex-A9 processor along with FPGA programmable logic. Over the years, FPGAs have become one of the preferred choices for fast processing of hyperspectral data, due to their reconfigurability, parallelization properties, short development time, easy data handling and low power consumption. Also, new FPGA devices have enhanced resistance to ionizing radiation inherently present in space (does not refer to Zynq-7000).

Compared to computer clusters or GPUs, FPGAs offer much smaller size and weight, as well as substantially lower power consumption. The adaptivity of FPGA through reconfigurability offers on-the-fly changes which indeed extend the life span of remote sensing satellites. This opens possibility to select algorithms from a ground station [13]. It should be noted that GPUs are still not incorporated in satellite Earth observation missions due to radiation-tolerance and power consumption issues, despite the increasing programmability of low-power GPUs such as those available in smartphones.

The Zynq-7000 SoC offers high performance ports to connect ARM Processing System (PS) and FPGA programmable logic (PL). This is very desirable for hardware and software codesigned implementations. By deciding which elements will be performed by the programmable logic, and which elements will run on the ARM Cortex-A9, we partition the computation system leading to the accelerated execution of hyperspectral processing algorithms.

In conclusion, FPGA is a reasonable choice for HYPSO SmallSat mission. Example is a scenario where processing of hyperspectral images is performed on-board, so that only obtained results are transmitted to the ground station. This scenario does not exclude the possibility to compress the data and transmit it after transmitting operational data [14].

1.5 Main contributions

The project assignment for master thesis lists the main tasks:

- Further integration of FPGA and SW modules through implementation of correlation matrix and inversion matrix calculation
- Analysis of additional speed-up potential for HW/SW codesigned implementation
- Further analysis of numerical error induced by fixed-point implementation in FPGA

These tasks were approached through two proposed FPGA solutions - HW/SW implementation and full FPGA implementation.

HW/SW implementation includes the FPGA accelerator and software solution used to handle a selection of target detection algorithms. After partitioning of the system, main computation parts of the target detection algorithms were extracted and implemented on FPGA platform. Static elements of the algorithm which can be used for subsequent runs are software-based. The design was developed during the specialization project, and improved during the thesis work in terms of performance, interoperability and testability. Additionally, it was upgraded to support two more detectors - ASMF and CEM. Most importantly, the accelerator has the speed-up factor of **28.54** compared to SW model run on ARM Cortex-A9. Furthermore, detailed analysis of the influence of numerical error induced by fixed-point implementation in FPGA on detection performance is done.

Full FPGA implementation is a standalone FPGA solution which performs all required computation using FPGA fabric. The implementation combines correlation matrix and inversion matrix calculation using Sherman-Morrison formula, with the possibility of real-time operation. The implementation supports ACE-R, CEM and ASMF detectors, and achieves comparable or better performance than state-of-the-art solutions.

A conference paper has been submitted and presented based on the work during thesis and specialization project:

- Đ. Bošković, M. Orlandić, S. Bakken and T. A. Johansen, "HW/SW Implementation of Hyperspectral Target Detection Algorithm", 8th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, June 2019.

In addition to aforementioned target detection algorithm implementations, PPI endmember extraction algorithm was implemented using Vivado HLS. This implementation illustrates productivity benefits of a C-based development flow using HLS. The VHDL, MATLAB and C code developed during the thesis are available on [15].

1.6 Structure of the Thesis

The thesis is divided into 6 chapters. The remainder of the thesis is organized as follows:

Chapter 2 introduces the background information necessary for testing and development of target detection algorithm hardware implementation. The chapter presents target detection algorithms. In addition to that, brief overview of Zynq-7000 platform is provided, with the state-of-the-art target detection algorithm implementations on FPGA platforms. Chapter 2 also introduces Vivado HLS and PPI endmember extraction algorithm.

Chapter 3 reviews state-of-the-art target detection algorithms. The chapter describes hyperspectral datasets used for testing as well as metrics used to estimate their detection performance, and presents the findings of algorithm testing and evaluation.

Chapter 4 presents hardware-software codesigned implementation of target detection accelerator in section 4.1, full FPGA solution using Sherman-Morrison updating for matrix inversion in 4.2 and PPI algorithm implementation in 4.3.

Chapter 5 discusses the results of FPGA implementations with respect to hardware performance, resource utilization and detection performance.

Chapter 6 concludes with guidelines for future development.

Chapter 2

Background

2.1 Hyperspectral data representation

This section is strongly influenced and developed from [12] and [11]. Different data representations of hyperspectral images regarding target detection algorithms are presented in this section.

Hyperspectral images contain both spatial and spectral information. We can illustrate one image as a hyperspectral data cube shown in Fig. 2.1. The cube has three dimensions, of which two are spatial and one is spectral dimension. If we extract one pixel from a certain spatial location, we can plot the spectral reflectance curve as a function of wavelength. On the other side, by choosing one spectral channel, it is possible to extract an intensity (grayscale) image which will represent the spatial distribution of reflectance captured by the imager. As spectral information is of primary focus regarding target detection algorithms, data is typically represented as a set of spectral measurements. Therefore, a pixel $\mathbf{x}_i$ is a vector containing spectral information and it is formulated as:

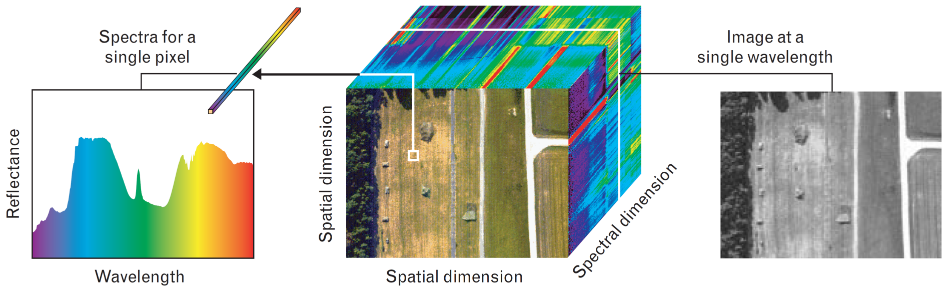


Figure 2.1: Illustration of a hyperspectral data cube (in the center), along with spectra plot (on the left) for a single pixel and a single spectral channel intensities shown in grayscale (on the right) [11].

$$\mathbf{x}_i = [L_i(\lambda_1) \quad L_i(\lambda_2) \quad ... \quad L_i(\lambda_K)]^T \tag{2.1}$$

where $L_i(\lambda_k)$ is a measurement of a spectral band with a center frequency λ_k. Hyperspectral image can then be stored as a matrix **X** with N pixels and K spectral bands:

$$\mathbf{X} = [\mathbf{x}_1 \quad \mathbf{x}_2 \quad ... \quad \mathbf{x}_N] = \begin{bmatrix} L_1(\lambda_1) & L_2(\lambda_1) & \dots & L_N(\lambda_1) \\ L_1(\lambda_2) & L_2(\lambda_2) & \dots & L_N(\lambda_2) \\ \vdots & \vdots & \ddots & \vdots \\ L_1(\lambda_K) & L_2(\lambda_K) & \dots & L_N(\lambda_K) \end{bmatrix}. \tag{2.2}$$

Matrix **X** does not expose spatial relations between pixels, but the ordering of pixels and information about image size parameters allow easy conversion to appropriate pixel arrangement for extraction of spatial information.

2.1.1 Geometrical data representation

As such, a pixel is a vector in K-dimensional Euclidean space, where K is the number of spectral bands. Since spectral measurements $L_i(\lambda_k) \geq 0$, all pixel vectors belong to the positive cone of K-dimensional space. As Euclidean spaces also generalize to higher dimensions, we can define two similarity metrics between pixels $\mathbf{x}_1$ and $\mathbf{x}_2$:

$$d_{12} = \sqrt{(\mathbf{x}_1 - \mathbf{x}_2)^T(\mathbf{x}_1 - \mathbf{x}_2)} \tag{2.3}$$

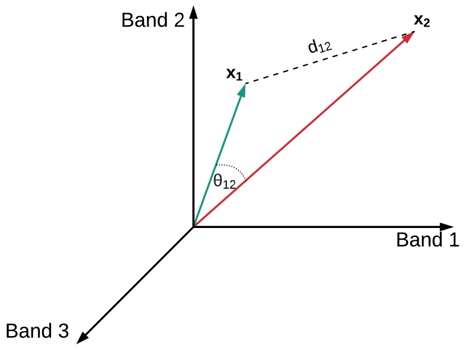
$$\theta_{12} = \cos^{-1} \frac{\mathbf{x}_1^T\mathbf{x}_2}{\sqrt{(\mathbf{x}_1^T\mathbf{x}_1)(\mathbf{x}_2^T\mathbf{x}_2)}} \tag{2.4}$$

where d_{12} represents the distance measure, and θ_{12} is the spectral angle measure. It is, however, advisable to use Eq. 2.4 rather then Eq. 2.3 to measure spectral similarity. Eq. 2.3 has a particular issue with the variation of pixel illumination, when one pixel is scaled with factor α, $\mathbf{x}_i = \alpha\mathbf{x}$. In this case, spectra has the same shape, but the change of illumination intensity scales the length of a vector representing the pixel. This is the basis for geometrical representation of hyperspectral data. An example is shown in Fig. 2.2(a), where two pixels $\mathbf{x}_1$ and $\mathbf{x}_2$ are placed in three dimensional space (three spectral bands). The distance and angle between pixels is annotated as d_{12} and θ_{12}, respectively.

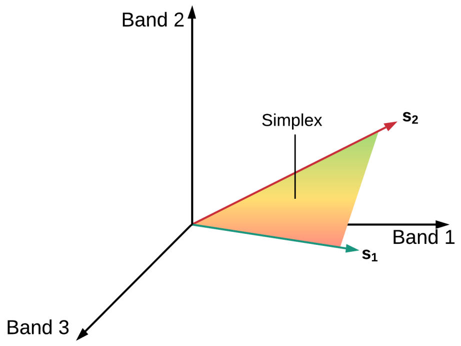
2.1.2 Spectral variability and mixing

The observed spectra in remote sensing applications is never fixed, but exhibits inherent variability due to atmospheric conditions, sensor noise, ground-cell position, material mixing, spatial resolution and other factors. As such, measured spectra with variability produces a cloud of points (tips of vectors) in spectral space. By contrast, deterministic invariable spectra is a single fixed vector and it is usually idealized spectra from spectral libraries. Moreover, hyperspectral data exploitation algorithms also encounter problem of sub-pixel abundances explained by spectral mixing.

Spectral mixing occurs when spatial resolution of HSI is lower than spatial extent of ground-cell materials. Therefore, it is very probable that many materials contribute to the



(a) Measures of spectral similarity and geometrical data representation



(b) Linear mixing model and a simplex formed with two pure materials

Figure 2.2: Measures of spectral similarity (on the left) and linear mixing model (on the right)

measured spectrum. Such spectrum is regarded as a mixed pixel (sub-pixel), whereas a pure pixel contains only one material called endmember. In other words, endmembers are spectrally pure, unique materials which occur in the scene. If the spectral mixture is macroscopic, then each reflected photon reacts with a single surface material present in the observed ground cell. The reflectance is therefore a combination of all reflected photons mixed linearly corresponding to the area each material covers on the ground cell. Assuming that $\mathbf{s}_i$, $i = 1...M$, are the endmembers and a_i their area fractions (abundances) of the ground cell, then linear mixing can be defined as:

$$\mathbf{x} = \sum_{i=1}^{M} a_i \mathbf{s}_i + \mathbf{n} \tag{2.5}$$

where $\mathbf{n}$ is the additive noise vector. An example is shown in Fig. 2.2(b), where vertices of two endmembers $\mathbf{s}_1$ and $\mathbf{s}_2$ form a simplex. Any linear mixture of these two endmembers would be a point lying in 2-simplex, regardless of how many spectral dimensions existed in the data. In general, the endmembers in the scene are not known, but extracted from the scene using many endmember extraction techniques, such as PPI and N-FINDR [16]. Pixel purity index (PPI) algorithm is described in section 2.2.

Not only low spatial resolution can cause spectral mixing in a pixel, but mixed pixels can also appear when distinct materials are combined into a homogeneous mixture. The spectral mixing can also be modelled using nonlinear mixing model, while spectral variability is further described using subspace model and probability density models. The probability density model is used for statistical representation of hyperspectral images.

2.1.3 Statistical data representation

Geometrical representation is particularly useful for describing spectral mixing, similarity and change transformation due to illumination and environmental conditions. However, geometrical representation is not an adequate model for noise and the stochastic nature of hyperspectral data. Therefore, statistical approach for describing hyperspectral images is employed using three main parameters: mean vector $\boldsymbol{\mu}$, covariance matrix $\boldsymbol{\Sigma}$ and correlation matrix $\mathbf{R}$.

If we assume noise model $\mathbf{n}$ and the additive model $\mathbf{x} = \mathbf{s}+\mathbf{n}$, where $\mathbf{s}$ is a deterministic signature, then the probability density function is:

$$p(\mathbf{x};\theta) = \frac{1}{2\pi^{K/2}} \frac{1}{\mid \boldsymbol{\Sigma} \mid^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})} \tag{2.6}$$

with parameters of the distribution $\theta = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}$. Assuming zero-mean noise, mean vector $\boldsymbol{\mu} = \mathbf{s}$, whereas covariance matrix $\boldsymbol{\Sigma} = \sigma^2\mathbf{I}$ due to wavelength independent variance assumption.

The multivariate probabilistic model in Eq. 2.6 with corresponding assumptions can only characterize the sensor noise of the dataset. This simple normal distribution is unable to fully incorporate the aforementioned stochastic nature of spectral properties of observed objects in the image. Typically, the parameters θ are unknown and are estimated from the provided dataset using sample statistics. The standard statistical method employed for estimating the unknown parameters is the maximum likelihood estimation, explained in detail in [12]. Resulting sample statistics are following:

$$\text{mean vector } \boldsymbol{\mu} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i, \tag{2.7}$$

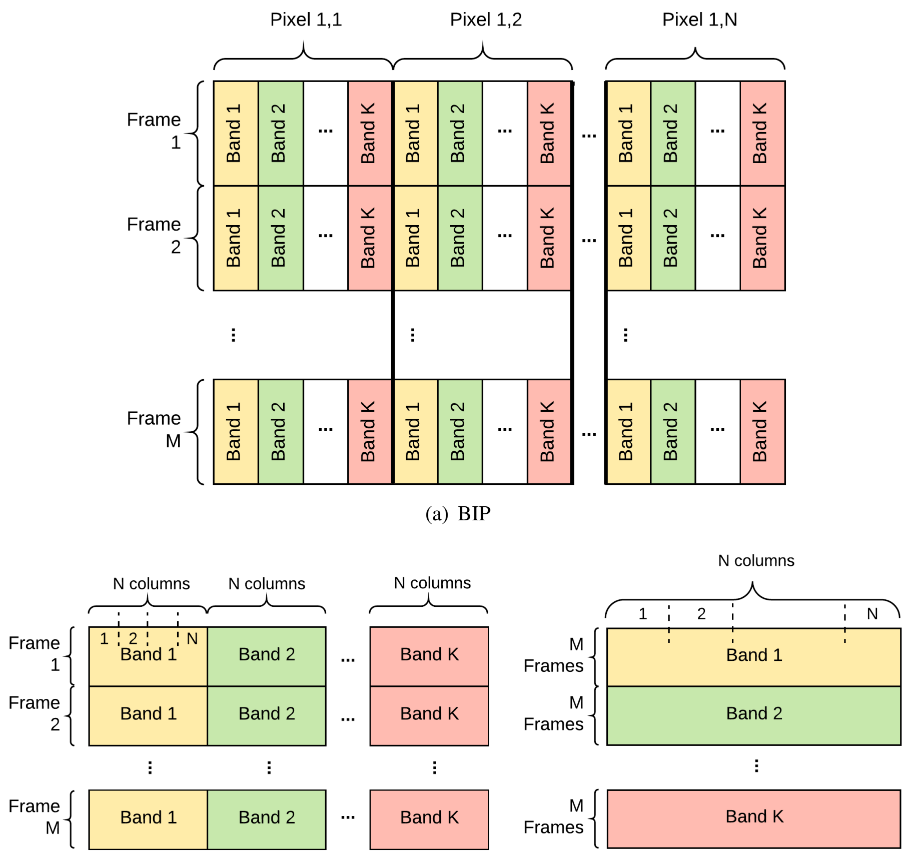
$$\text{covariance matrix } \boldsymbol{\Sigma} = \frac{1}{N}\sum_{i=1}^{N}[\mathbf{x}_i - \boldsymbol{\mu}][\mathbf{x}_i - \boldsymbol{\mu}]^T = \frac{1}{N}\mathbf{X}\mathbf{X}^T - \boldsymbol{\mu}\boldsymbol{\mu}^T \tag{2.8}$$

$$\text{and correlation matrix } \mathbf{R} = \frac{1}{N}\mathbf{X}\mathbf{X}^T. \tag{2.9}$$

2.1.4 Hyperspectral image organization

There are three common methods of organizing hyperspectral images in memory: band interleaved by pixel (BIP), band interleaved by line (BIL) and band sequential (BSQ). Each of these component ordering schemes provide different computational complexity for real-time processing of hyperspectral data [17], especially if the data is streamed in a certain fashion to the processing cores. The schemes are shown in Fig. 2.3.

In BIP format, all spectral components of one pixel are written in subsequent locations, followed by another pixel in a frame. Next row is formed for the next frame obtained by a push-broom imager. Therefore, bands 1 to K are written for pixel 1 which is part of frame 1, followed by components of pixel 2, and so on. On the other side, since the imager obtains one frame at a time, the image can be stored in BIL format. In this case, data is stored band by band for each frame. Finally, by storing data band by band for all pixels, we have created the image in BSQ format. Target detection algorithms presented in this thesis access the image in BIP format.



(a) BIP

(b) BIL

(c) BSQ

Figure 2.3: Storage formats for hyperspectral images.

2.2 Pixel Purity Index Algorithm

PPI algorithm is used to find the most spectrally pure pixels in a hyperspectral image. In other words, PPI is an endmember extraction algorithm, widely used with dimensionality reduction techniques such as minimum noise factoring (MNF) to reduce the computational complexity of the algorithm. The PPI algorithm is simple and highly parallelizable, thus very feasible for FPGA implementation. The full description of the original algorithm (ENVI's PPI) is not available in the literature; however similar interpretation is available in [18] and will be described here.

The PPI algorithm consists of 4 steps: initialization, PPI projections, candidate selection and endmember extraction. The PPI projections step is the most computationally expensive part of the algorithm. The steps are described as follows:

1. In the initialization step, M unit vectors, named *skewers*, are randomly generated.
2. For each $\mathbf{skewer}_j$, $j = 1, ..., M$, all pixel vectors $\mathbf{x}_i$, $i = 1, ..., N$, from a hyperspectral image $\mathbf{X}$, are projected onto the skewer. Each skewer forms an *extrema set* consisting of processed pixel vectors. An example of the projection step on three skewers is shown in Fig. 2.4.
3. In the candidate selection step, we find the PPI scores $N_{PPI}(\mathbf{x}_i)$ for all pixels. PPI score is defined as the sum of appearances in the extrema sets of all skewers for each pixel vector.
4. Endmember extraction step requires a threshold value t_v for the PPI score in order to extract all pixel vectors with $N_{PPI}(\mathbf{x}_i) \geq t_v$.

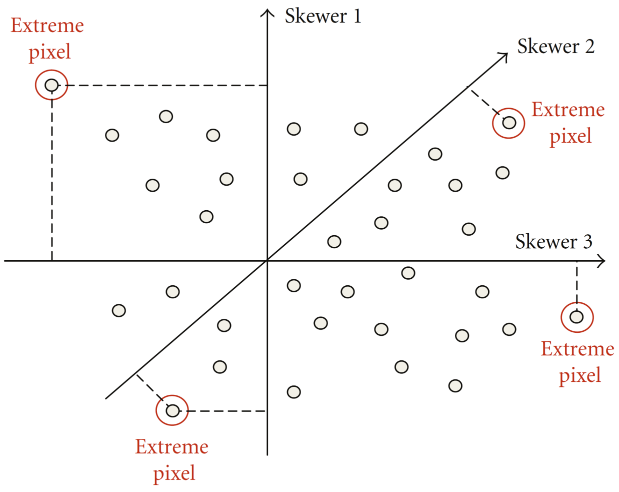


Figure 2.4: An example showing the operation of PPI algorithm with three skewers in 2-dimensional space [18].

2.3 Target detection algorithms

Target detection algorithms search for objects of interest in the remotely obtained hyperspectral scene. Detection in a general sense also includes searching for anomalies without the knowledge about a specific spectral signature. In this thesis, the focus is on target detection where a signature is provided to the algorithm to determine if the target is present in the scene. Those detectors belong to the group of signature-matched detectors. This section is mainly based on [11, 10, 12].

The simplest reasonable approach to determine the presence of a target in a pixel would be to use one of the spectral similarity metrics as in Eq. 2.4. The measure of angle between data pixel and a reference signature is known as a spectral angle, and under certain conditions it becomes spectral angle mapper (SAM) target detection algorithm. However, many algorithms are based on the estimation of the image background for better distinction between target and non-target pixels.

Typical target detection system is shown in Fig. 2.5. The system consists of a target detection algorithm ($D(\mathbf{x})$) and threshold selection system. The input pixel vector $\mathbf{x}$ is mapped onto a scalar value $y = D(\mathbf{x})$, which is called detection statistic. On the other side, threshold selection system constructs a background statistic model which would be used to produce the appropriate threshold value η.

The target detection algorithm provides our system with a value which, when compared with the threshold value, correctly determines if a pixel under test contains a designated target. In other words, the value produced by target detection algorithm corresponds to the probability of the input pixel to be a designated target. It is the task of threshold selection system to set the optimum threshold so that a significant amount of present tar-

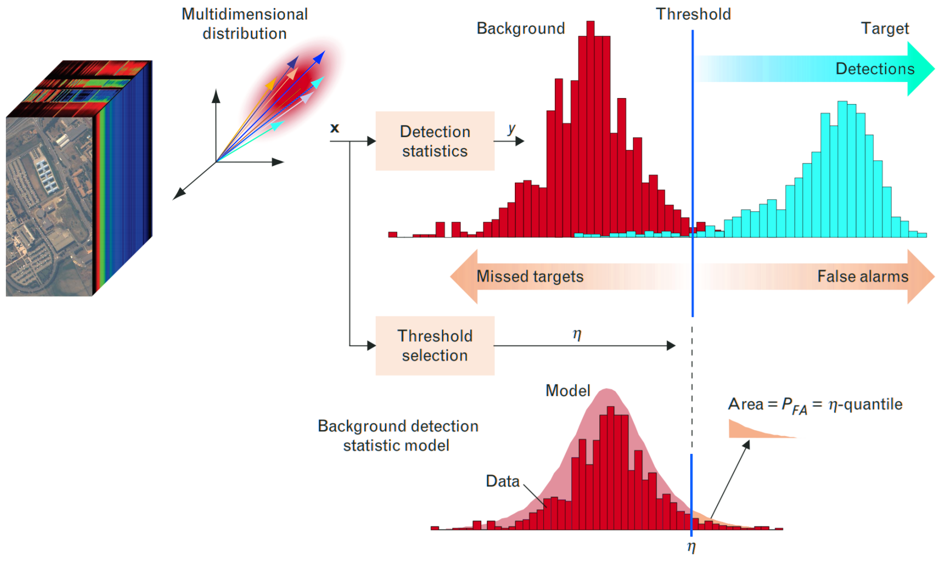


Figure 2.5: Illustration of a typical hyperspectral target detection system, modified from [11].

gets are detected while the false alarm rate (false positive rate, i.e., wrongly categorized as targets) is kept below a certain value. That system is typically constant false alarm rate (CFAR) processor, which is the important part of automatic target detection systems. Threshold selection systems are not discussed further in this thesis.

The mathematical framework for development and analysis of target detection comes from statistical signal processing, where target detection is regarded as binary hypothesis testing between two competing hypothesis. Hypothesis H_0, also called null hypothesis, asserts that the pixel being tested is not a target. On the contrary, an alternative hypothesis H_1 asserts that the pixel under test contains the target. Each target detection algorithm was developed under different models of the signals for both hypotheses. Those models are characteristic for the algorithm and they strongly influence the performance of detection. The models are introduced in the following subsections.

2.3.1 Spectral angle mapper

The detection problem can be modelled as follows:

$$\begin{aligned} H_0 &: \mathbf{x} = \mathbf{b} \\ H_1 &: \mathbf{x} = \alpha\mathbf{s} + \mathbf{b} \end{aligned} \tag{2.10}$$

where **b** is the combination of background and inherent random noise, **s** is the known spectral signature and α is the factor used to scale the unresolved intensity of signal **s**. The uncertainty modelled by α is due to illumination and spectral mixing characteristic for subpixel targets. To derive SAM algorithm, we assume that background **b** is random, zero-mean and normally distributed with variance σ. Then, SAM is defined as:

$$D_{SAM}(\mathbf{x}) = \frac{(\mathbf{s}^T\mathbf{x})^2}{(\mathbf{s}^T\mathbf{s})(\mathbf{x}^T\mathbf{x})}. \tag{2.11}$$

An equivalent detection statistic is:

$$D_{SAM}(\mathbf{x}) = -\cos^{-1}\frac{\mathbf{s}^T\mathbf{x}}{\sqrt{(\mathbf{s}^T\mathbf{s})(\mathbf{x}^T\mathbf{x})}} \tag{2.12}$$

since both square root as well as cosine operation are monotonic functions. Eq. 2.12 is equivalent to the aforementioned measure of the angle between a reference spectrum and a pixel under test in the spectral space. SAM is derived to achieve CFAR characteristics and therefore, the variance σ, which is presumably varying spatially across the image, is estimated as the square of spectrum magnitude $\mathbf{x}^T\mathbf{x}$. In conclusion, SAM is fast and simple algorithm relatively robust to illumination effects. However, the algorithm is limited in performance due to the initial assumptions about background modelling.

2.3.2 Constrained energy minimization

CEM is designed using a finite impulse response (FIR) filter with detection statistic defined as:

$$D_{CEM}(\mathbf{x}) = \mathbf{h}^T\mathbf{x}, \tag{2.13}$$

where $\mathbf{h} = [h_1, h_2, ..., h_K]^T$ is the weight factor used to minimize the output power of a filter [19]. The weight vector is optimized so that the algorithm better separates the background and the known signature $\mathbf{s}$. Then, the minimization problem is defined under the following constraint:

$$min_{\mathbf{h}}(\frac{1}{N}\sum_{i=1}^{N}(\mathbf{h}^T\mathbf{x}_i)^2) = min_{\mathbf{h}}(\mathbf{h}^T\mathbf{R}\mathbf{h}) \quad subject \quad to \quad \mathbf{h}^T\mathbf{s} = 1 \tag{2.14}$$

where $\mathbf{R}$ is the sample correlation matrix calculated as $\mathbf{R} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i\mathbf{x}_i^T$. The constraint $\mathbf{h}^T\mathbf{s} = 1$ forces the perfect detection statistic (full match) to have the value 1. Derived weight factor proposed by problem 2.14 is:

$$\mathbf{h} = \frac{\mathbf{R}^{-1}\mathbf{s}}{\mathbf{s}^T\mathbf{R}^{-1}\mathbf{s}}. \tag{2.15}$$

Finally, the resulting detection statistic after applying linear operation $\mathbf{h}^T$ is:

$$D_{CEM}(\mathbf{x}) = \frac{\mathbf{s}^T\mathbf{R}^{-1}\mathbf{x}}{\mathbf{s}^T\mathbf{R}^{-1}\mathbf{s}}. \tag{2.16}$$

As in majority of other signature-based detection algorithms, low-probability distribution of target pixels is assumed. This assumption has direct impact on estimation of the sample correlation matrix, which can be contaminated if the number of target pixels surpasses a certain limit. On the contrary, if the number of target pixels is small, their influence when estimating $\mathbf{R}$ can be neglected. Therefore, CEM is considered a small target detector.

2.3.3 Adaptive coherence/cosine estimator

The binary hypotheses defined in Eq. 2.17 lead to the formulation of ACE algorithm.

$$\begin{aligned} H_0 &: \mathbf{x} = \beta\mathbf{b} \\ H_1 &: \mathbf{x} = \alpha\mathbf{s} + \beta\mathbf{b} \end{aligned} \tag{2.17}$$

In the extension of hypotheses shown in Eq. 2.10 for deriving SAM algorithm, here we have newly introduced scaling factor β of the background and noise combination signal $\mathbf{b}$ under both hypotheses. If we assume that $\mathbf{b}$ is a zero-mean, white random vector process ($\mathbf{b} \sim N(\mu, \sigma)$) we can derive ACE algorithm:

$$D_{ACE}(\mathbf{x}) = \frac{(\mathbf{s}^T\mathbf{\Sigma}^{-1}\mathbf{x})^2}{(\mathbf{s}^T\mathbf{\Sigma}^{-1}\mathbf{s})(\mathbf{x}^T\mathbf{\Sigma}^{-1}\mathbf{x})}. \tag{2.18}$$

In Eq. 2.18, the covariance matrix $\mathbf{\Sigma}$ is typically estimated from the hyperspectral data of the scene. In certain cases, designated targets are extracted from the provided scenes and especially ACE algorithm performs well in these conditions. However, it is possible that the algorithms may have different performance with normalized laboratory signatures. Both ACE and CEM are able to achieve CFAR characteristics, and most importantly, they do not require knowledge about all present endmembers in the hyperspectral image. Unlike SAM which is considered first-order hyperspectral measure, both CEM and ACE are second-order hyperspectral measures because of covariance and correlation matrices [20].

2.3.4 Adjusted Spectral Matched Filter

Adaptive spectral matched filter (ASMF) [21] is constructed using the simplified Reed-Xiaoli (RX) anomaly detector [22] to adjust the output of CEM detector in Eq. 2.16. The adjustment is implemented by introducing a non-target pixel suppression factor A defined as:

$$A = \left| \frac{\mathbf{x}^T \mathbf{R}^{-1} \mathbf{s}}{\mathbf{x}^T \mathbf{R}^{-1} \mathbf{x}} \right|. \tag{2.19}$$

The factor A with the variable power n is then combined with CEM as follows:

$$D_{ASMF}(\mathbf{x}) = D_{CEM}(\mathbf{x}) \cdot A^n = \frac{\mathbf{s}^T \mathbf{R}^{-1} \mathbf{x}}{\mathbf{s}^T \mathbf{R}^{-1} \mathbf{s}} \cdot \left| \frac{\mathbf{s}^T \mathbf{R}^{-1} \mathbf{x}}{\mathbf{x}^T \mathbf{R}^{-1} \mathbf{x}} \right|^n. \tag{2.20}$$

Depending on the power n, different spectral features are amplified or suppressed. By setting n to 0, the ASMF becomes CEM algorithm. Alternatively, by setting n to 1, similar form to *ACE-R* is obtained (which is introduced in section 3.3.1). Weight n can also be non-integer number, such as 0.5, 1.5 or similar. Setting the weight to provide substantial detection performance depends on the related scenes used for testing as well as the designated targets and their properties.

2.4 Matrix inversion

The computation of the inverse of a matrix is a part of almost all considered target detection algorithms. This computationally intensive task is usually performed directly or iteratively, and the choice of the method affects the performance and/or precision of the solution. Direct computation of the inverse requires a fixed number of operations to obtain the solution, whereas iterative methods converge to a solution by subsequent updates of the estimate [23].

Direct methods for inverting matrices, such as Gauss-Jordan elimination, LU decomposition, and Cholesky decomposition require a-priori created matrix ready for inversion, for example correlation matrix. This usually means that they are not able to operate as a part of a system with real-time constraints. In such cases, iterative methods are employed, such as Sherman-Morrison formula. In the following subsections, Gauss-Jordan elimination, LU decomposition and Sherman-Morrison formula are briefly presented. Additionally, they have been compared in terms of computational complexity for real-time and non-real time problems.

2.4.1 Gauss-Jordan elimination

If $\mathbf{A}$ is a square invertible matrix, we augment it with identity matrix of the same size and form matrix $\mathbf{X}$, $\mathbf{X} = [\mathbf{A}, \mathbf{I}]$. Then, by performing Gauss-Jordan elimination as explained through MATLAB code in Listing 2.1 (inspired by [24]), the left block of matrix $\mathbf{X}$ is reduced to identity matrix through application of row operations. Finally, the right block of matrix $\mathbf{X}$ is the solution: inverted matrix $\mathbf{A}^{-1}$.

The total effect of all the row operations is equivalent to multiplication of $\mathbf{X}$ with $\mathbf{A}^{-1}$ from left side as shown in Eq. 2.21. If the algorithm is unable to reduce the left block of matrix $\mathbf{X}$ to an identity matrix, then $\mathbf{A}$ is not invertible.

$$[\,\mathbf{A} \mid \mathbf{I}\,] \rightarrow [\,\mathbf{A}^{-1}\mathbf{A} \mid \mathbf{A}^{-1}\mathbf{I}\,] \rightarrow [\,\mathbf{I} \mid \mathbf{A}^{-1}\,] \tag{2.21}$$

Listing 2.1: Gauss-Jordan elimination

```
for i = 1 : n

   if(X(i,i) == 0)
      for j = i+1 : n
         if(X(i,j) ~= 0)
            %row swapping
           X([i j],:) = X([j i],:);
           break;
         end
      end
   end

   if(X(i,i) == 0) error('singular matrix, exit!'); end

%building upper triangular matrix
```

```
   for j = i+1 : n
      %forward elimination
      X(j,:) = X(j,:) - X(i,:)*(X(j,i)/X(i,i));
   end
end

%building diagonal matrix
for i = n :-1: 2
   for j = i-1 :-1: 1
      %backward elimination
       X(j,:) = X(j,:) - X(i,:)*(X(j,i)/X(i,i));
   end
end

%building identity matrix
for i = 1 : n
   %last division to build matrix I
   X(i,:) = X(i,:)*(1/X(i,i));
end
```

2.4.2 LU decomposition with partial pivoting

After decomposing the matrix using LU decomposition, it is straightforward to find its inverse. LU decomposition starts with factoring of a matrix into a product of upper and lower triangular matrices, **U** and **L**, respectively. Assuming that **P** is the permutation matrix obtained by partial pivoting, and **A** is the matrix being decomposed, it follows:

$$\mathbf{PA} = \mathbf{LU}. \tag{2.22}$$

The method consists of three main steps:

1. Find the row in column k that contains the largest absolute value entry
2. Swap the rows of A and P (perform partial pivoting)
3. Perform the k-th step of Gaussian elimination

Upon obtaining **L** and **U**, we can find the inverse **X** of matrix **A**:

$$\begin{aligned}\mathbf{AX} &= \mathbf{I}\\ \mathbf{PAX} &= \mathbf{PI}\\ \mathbf{LUX} &= \mathbf{PI},\end{aligned} \tag{2.23}$$

which can be rewritten as:

$$\begin{aligned}\mathbf{LY} &= \mathbf{PI}\\ \mathbf{UX} &= \mathbf{Y}.\end{aligned} \tag{2.24}$$

Solving equations in 2.24 row by row (using forward and back substitution), leads to the inverse matrix $\mathbf{X} = \mathbf{A}^{-1}$.

2.4.3 Sherman-Morrison formula

If $\mathbf{u}$ and $\mathbf{v}$ are two arbitrary column vectors and $\mathbf{A}$ is an invertible square matrix, then Sherman-Morrison formula is expressed as:

$$(\mathbf{A} + \mathbf{u}\mathbf{v}^T)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T\mathbf{A}^{-1}}{1 + \mathbf{v}^T\mathbf{A}\mathbf{u}} \tag{2.25}$$

To invert the matrix $\mathbf{A}$ using Sherman-Morrison formula, we initialize matrix $\mathbf{X}$ to an identity matrix of the same size as $\mathbf{A}$. Then, the following iteration scheme is used:

$$\begin{aligned} \mathbf{X}_0 &= \mathbf{I} \\ \mathbf{X}_i &= \mathbf{X}_{i-1} - \frac{\mathbf{X}_{i-1}(\mathbf{a}_i - \mathbf{i}_i)\mathbf{i}_i^T\mathbf{X}_{i-1}}{1 + \mathbf{i}_i^T\mathbf{X}_{i-1}(\mathbf{a}_i - \mathbf{i}_i)} \end{aligned} \tag{2.26}$$

where $\mathbf{a}_i$ is the i-th column of matrix $\mathbf{A}$, and $\mathbf{i}_i$ is the i-th column of the identity matrix. After n iterations, where n is the number of columns of matrix $\mathbf{A}$, the inverse $\mathbf{X}_n = \mathbf{A}^{-1}$ is obtained. Most importantly, Sherman-Morrison formula can be used to compute the corrected inverse of matrix $\mathbf{A}$ after it is subjected to rank-1 update. Using Eq. 2.25, we avoid the costly computation of inverting $\mathbf{A} + \mathbf{u}\mathbf{v}^T$ anew.

2.4.4 Computational complexity of algorithms

In case of a non-real time system, full image or its subset is needed to calculate the correlation matrix. Then, its inverse can be found using the aforementioned methods. To calculate the correlation matrix of an image with N pixels consisting of K spectral components, it is necessary to perform NK^2 multiplications. Afterwards, the inverse is performed using a method with general complexity of $O(K^3)$ multiplications.

Using Sherman-Morrison formula as in Eq. 2.25, $N(3K^2 + K)$ multiplications are needed to obtain the final inverse matrix. This is much higher computational complexity than non-real time system when all pixels are used. Nevertheless, Sherman-Morrison formula enables the system to estimate certain spectral features on-the-fly as pixels are being acquired. If other matrix inversion methods were used under real-time constraints, they would need $K^2 + O(K^3)$ multiplications for each incoming pixel, while Sherman-Morrison formula requires only $(3K^2 + K)$ multiplications. As such, operation of a target detection algorithm in real-time using Sherman-Morrison formula is explained in section 4.2. On the other side, both Gauss-Jordan elimination and LU decomposition for matrix inversion are implemented in software for use in HW/SW codesign implementation explained in section 4.1.

2.5 Overview of Zynq-7000, FPGA cores and primitives

Zynq-7000 is a family of Xilinx System-on-Chip (SoC) products which provide full programmability using ARM-based processor and FPGA technology. The aforementioned high performance communication between PS and PL on a single chip allows integration and acceleration of software and hardware designs. Zynq-7000 chips are equipped with Artix-7 or Kintex-7 FPGA fabric with appreciable performance-per-watt metrics. The overview of the Zynq architecture is shown in Fig. 2.6, showing processing system with two processor cores and programmable logic connected using AXI ports.

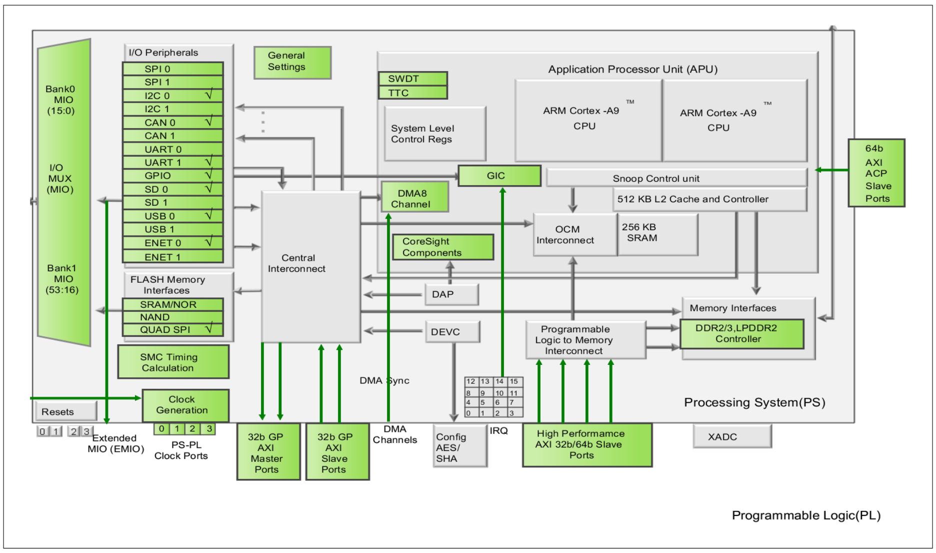


Figure 2.6: Overview of Zynq-7000 architecture

The remainder of this section will be dedicated to essential FPGA primitives and cores found on Zynq platform, such as AXI protocol, DSP blocks, BRAM blocks, DMA core and AXI divider.

2.5.1 DSP blocks on Zynq

The programmable logic on Zynq SoC contains digital signal processing (DSP) blocks with the structure as depicted in Fig. 2.7. These dedicated blocks are used to enhance the speed of the design involving multiply-accumulate (MAC) operations, barrel shifting or wide bus multiplexing. Zynq PL has a limited amount of DSP blocks available, and therefore the fundamental features and characteristics must be known to the designer so that implementation of the source code can appropriately exploit these resources.

The DSP block contains a multiplier and an accumulator, with three additional pipeline data registers. If full pipelining is utilized for multiplication and addition operations, then the DSP block can run at full speed for MAC units. The pipelining is used to boost the performance as well as reduce the overall power consumption.

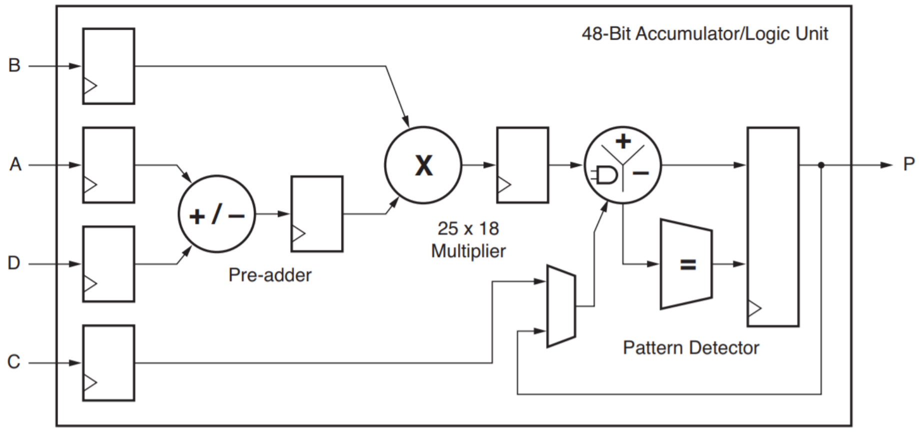


Figure 2.7: Simplified schematic of DSP48E1 block [25].

The accumulator is 48-bits wide, while one DSP multiplier has the capability to operate on a pair of operands with the width of 25 and 18-bits. For arithmetic operations requiring operands with wider data bus such as multiplication, the DSP block can perform a right wire shift by 17, which is used to combine and add partial products from two DSP slices and produce the final result. Therefore, it is possible to build larger multipliers, which are not limited by the inherent size of DSP block operands.

2.5.2 AXI protocols

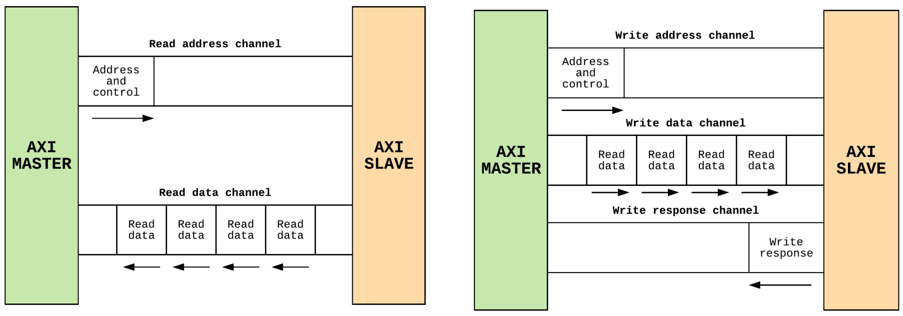
Advanced eXtensible Interface (AXI) is a part of ARM Advanced Microcontroller Bus Architecture (AMBA) interface widely used with the ARM Cortex-A processors, as well as in Zynq architecture. There are three types of AXI bus interfaces: AXI (full), AXI-lite and AXI-stream. As Zynq uses AMBA 4.0 released in 2010, the focus is placed on that version.

AXI4-lite interface

AXI and AXI-lite interfaces are defined by three channels: address, data and response channels, and two data flow directions: write and read. AXI4 allows simultaneous and bidirectional data transfer by separating data buses for reading and writing, along with the corresponding address buses.

Two end-points of the interface are called master and slave. The master initiates all data transfers by setting signals on corresponding address and data channels. One read transfer is shown in Fig. 2.8(a), where master sends the corresponding address on the read address channel and waits for the slave to respond with the requested data on the read data channel. Similarly, writing is performed by sending an address on the write address channel and corresponding data on the write data channel, as in Fig. 2.8(b). Additionally, there is a write response channel used to inform the master that the transfer was successful and complete.

The slave responds to master's requests using a common handshake protocol consisting of *VALID* and *READY* signal. If both handshake signals are asserted during one clock cycle, the transfer occurs. Moreover, all AXI and AXI-lite transactions are memory mapped, which allows direct access to the slave registers on PL from PS. AXI interface has more optional signals, allowing burst transfers, which are explained in detail in [26].



(a) AXI-Lite read channels (b) AXI-Lite write channels

Figure 2.8: AXI-Lite interface.

AXI-stream interface

AXI-stream is a simplified AXI interface used for high-speed data streaming with unlimited burst mode. The simplification consists of removing the concept of addresses and memory-mapping, and defining the protocol using a single write or read channel. Two consecutive transfers can be differentiated using additional handshake signal - *LAST*, which is asserted at the end of the transfer. Stream interfaces are usually connected to the DMA core which performs memory-mapped to stream conversion.

2.5.3 DMA core

In the process of design and verification, two DMA cores were considered, namely, AXI DMA and Cube DMA. AXI DMA [27] is a Xilinx IP module, while Cube DMA [28, 29] was specifically designed for hyperspectral images as a part of the HYPSO project. In general, DMA cores are used to allow direct memory access to peripherals with minimal CPU intervention. In the case of Zynq system, DMA is instantiated in PL and it is used to transfer data from DDR memory to the FPGA cores with AXI-stream interfaces.

The configurability of the general purpose DMA module provided by Xilinx reflects in options such as the number of read and write channels or usage of scatter-gather engine. It is configured using General Purpose (GP) AXI ports, while the memory data transfers occur over AXI High Performance (HP) ports. However, AXI DMA core has certain disadvantages when used for hyperspectral images which are overcome by the use of Cube

DMA. The disadvantages include inability to stream BSQ image formats without high rate of CPU interventions and unacceptable number of descriptors needed for block transfers.

2.5.4 Block RAM

In addition to distributed RAM available on the programmable logic, Zynq-7000 features 36Kb block RAM (BRAM) units. They represent dedicated blocks which allow wider memory utilization optimized for timing. BRAMs can be configured in single and dual-port modes with simultaneous read and write operations.

In Listing 2.2, a memory used for storing a matrix is defined as a VHDL array of *NB_COL* columns with predefined width. As BRAM devices are synchronous, on each positive clock edge this implementation can output one row *NB_COL*COL_WIDTH*-bits wide. On the other side, one element at a time can be written to the memory. As the number of columns and column width increases, memory is distributed into more RAM blocks. This behavior is depicted in Fig. 2.9.

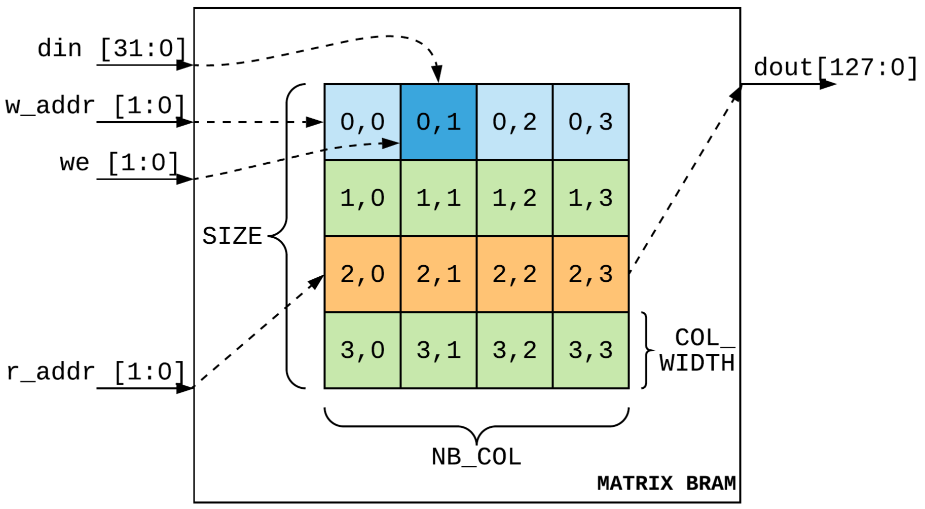


Figure 2.9: Matrix storage using BRAM

Listing 2.2: BRAM VHDL template

```vhdl
architecture behavioral of bram is

type ram_type is array (SIZE-1 downto 0) of
   std_logic_vector (NB_COL*COL_WIDTH-1 downto 0);
signal RAM : ram_type := (others => (others => '0'));

begin
```

```
process (clk)
begin
if rising_edge(clk) then
  dout <= RAM (conv_integer(r_addr));

  for i in 0 to NB_COL-1 loop
    if we(i) = '1' then
      RAM (conv_integer(w_addr))
      ((i+1)*COL_WIDTH-1 downto i*COL_WIDTH) <= din;
    end if;
  end loop;

end if;
end process;
```

2.5.5 AXI divider

Divider core provided by Xilinx is used for integer division based on High Radix, Radix-2 or look-up table method [30]. The core is instantiated in PL, with AXI-stream interfaces used as inputs and outputs. Depending on the division method chosen, the throughput and utilization of DSP, RAM blocks and FPGA fabric varies. High Radix division exploits resource reuse, especially of DSP and BRAM units, at the expense of a lower throughput. On the other side, Radix-2 division is designed for high throughput with increased usage of FPGA fabric. A divider with look-up tables simply estimates the reciprocal of the divisor based on a fixed table, which is afterwards multiplied with a dividend.

In the divider generator the user can configure flow control of the core, set optimization goals, latency options and additional control signals. All AXI-stream interfaces can be optimized in respect to data bus bit width, while setting the output channel to either fractional or division remainder type. Figure 2.10 shows the generated block in the Vivado Design Suite with the corresponding streaming interfaces.

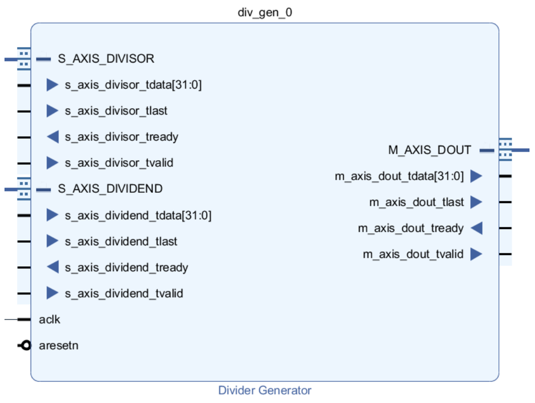


Figure 2.10: Generated Divider Block in Xilinx Vivado

2.6 Vivado HLS

The Vivado High-Level Synthesis (HLS) is a part of Vivado Design Suite produced by Xilinx. This HLS tool allows C, C++ and SystemC programs to be directly targeted into Xilinx devices without the need to create RTL design manually. In other words, HLS tool transforms a specification written in C into a RTL implementation which can be easily synthesized for FPGA platforms. Nowadays, HLS tools are gaining popularity as they improve developer productivity, allow designing at higher levels of abstraction as well as software/hardware cosimulation.

The design flow with Vivado HLS is shown in Fig. 2.11. The design starts with functional C code design and C testbench design. Usually, the golden reference is generated from a high-level model, for example in MATLAB. The reference serves to verify the function of the C code using the testbench. Once the code passes the functional verification phase, C-to-RTL HLS tool runs with provided directives and constraints used to define and refine the RTL implementation. Thus, the directives direct the HLS process to implement a specific behavior or hardware optimization.

Once the RTL is generated using HLS, C/RTL cosimulation verifies the RTL output, using the same testbench. The user evaluates the implementation, and this is usually one of many design iterations. Directives and constraints are then adjusted accordingly to obtain the required implementation before exporting RTL and packing it into an IP block.

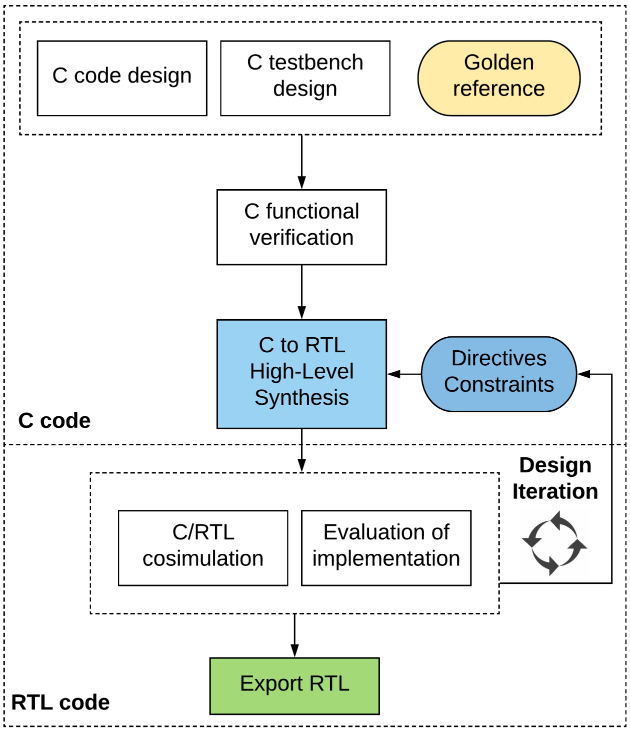


Figure 2.11: Vivado HLS design flow.

In this thesis, Vivado HLS is used to design the PPI algorithm implementation. The PPI is an ideal example for rapid development and design space exploration with optimization directives. The implementation is described in 4.3. Additionally, in Table 2.1, frequently used Vivado HLS directives (pragmas) are described.

Table 2.1: Vivado HLS optimization directives [31]

Directive	Description
#pragma HLS UNROLL	Unroll for-loops to create multiple independent operations rather than a single collection of operations.
#pragma HLS PIPELINE	Reduces the initiation interval by allowing the concurrent execution of operations within a loop or function.
#pragma HLS INTERFACE	Specifies how RTL ports are created from the function description.
#pragma HLS ARRAY_PARTITION	Partitions large arrays into multiple smaller arrays or into individual registers, to improve access to data and remove block RAM bottlenecks.
#pragma HLS DATAFLOW	Enable task level pipelining, allowing functions and loops to execute concurrently. Used to minimize interval.

2.7 State-of-the-art target detection algorithm implementations on FPGA platforms

As of today, not many hyperspectral target detection algorithms have been implemented on FPGA platforms. Main contributions in this field come from papers such as [32, 33, 24]. The goal of majority of FPGA implementations is achieving real-time operation, i.e., processing pixels as soon as they are registered from HSI and optionally pre-processed (e.g. atmospheric correction). Real-time performance is usually limited by background estimation techniques, which require collection of entire image data. Those algorithms are considered as global target detectors.

On the other side, we have local target detectors for which streaming background statistics (SBS) method has been proposed in [32]. Instead of estimating background statistics using all pixels in the image, the authors have proposed streaming structure which uses a subset of pixels. This subset is actually a sliding window of pixels with fixed size, which defines a local region used to estimate the local background statistics based on the correlation matrix. The chosen algorithm for implementation in this work is CEM, requiring inverse of the correlation matrix to compute detection statistic. Rather than inverting correlation matrix for each subset of pixels, Sherman-Morrison formula [34] has been used to gradually update the inverse of the correlation matrix as the window slides through the image. However, this approach has a few problems which have been analyzed in [33]. Namely, updating the inverse of correlation matrix is a very computationally intensive task and it is performed twice for the incoming and outgoing pixel of the window. Moreover, it creates data dependency since a new update cannot proceed until the last one has finished. This causes stalls in the processing pipeline and considerably degrades the overall computational performance. The sliding window adaptation is explained in detail in section 4.2.1.

In [33], a few optimizations of SBS method have been proposed. Firstly a non-sliding window has been introduced. This indicates that the outgoing pixel is actually never moved out of background statistic estimation leading to the substantial reduction in the number of calculations required to update the correlation matrix. In other words, only incoming pixels from stream are added into the window. Additionally, SBS has been deeply pipelined. Aforementioned data dependency has been partially resolved by making inverse calculations of neighboring pixels independent. Applying these optimizations has achieved a certain speed-up of the system, however, resource utilization is 5 times increased compared to [32]. Also, data accuracy is an issue since dynamic range of inverted correlation matrix varies dramatically due to the non-sliding window.

CEM algorithm and background estimation have also been implemented using Coordinate Rotation Digital Computer (CORDIC) in [35]. CORDIC algorithm successfully solves matrix inversion problem by employing QR-decomposition. This approach could allow real-time operation, however CORDIC is usually unable to support fast execution, in this case target detection.

It is worth mentioning the implementation of automatic target-generation process using an orthogonal projection operator (ATGP-OSP) [24]. Without mentioning complex operation of ATGP-OSP algorithm, we will explain main computation elements implemented on FPGA. Inverse matrix calculation was also considered in this work. Gauss-Jordan elimination method has been selected for matrix inversion, since it can be fully parallelized. Unfortunately, this solution is still far from real-time operation. ATGP was also done with Gram-Schmidt method for orthogonal projection [36] which does not involve an inverse calculation.

In [17], various detection algorithms have been analyzed for their real-time implementation in hardware depending on availability of data and data formats such as: BIP pixel-by-pixel processing, BIL line-by-line processing and BSQ band-by-band processing. BIP certainly provides advantages for target detection algorithms, whereas with BSQ processing we can compute detection statistics only after all the bands have been received. The paper also discusses usage of sample correlation matrix instead of sample covariance matrix in certain cases, as they both provide comparable results. Real-time implementations mostly benefit from using the correlation matrix which does not require data mean removal and allows intermediate data analysis as discussed before. In addition to that, to accelerate overall execution time, we can use a certain percentage of pixels to calculate the background statistics, instead of using all of the incoming pixels. This paper claims that the statistics with appropriately chosen percentage of pixels does not affect the detection performance. Finally, hardware architectures for iterative correlation matrix inversion are proposed using multiply–accumulate (MAC) blocks.

Collaborative-representation-based detector (CRD) was implemented on FPGA in [37]. This algorithm also requires matrix inversion operation, which was processed in real-time manner using Sherman-Morrison formula in this paper. The detection performance results of the algorithm implementation cannot be compared with other implementations due to different selection of hyperspectral datasets used for evaluation. Nevertheless, the algorithm shows promising results.

Chapter 3

Review of state-of-the-art target detection algorithms

In this chapter, detection performance of target detection algorithms, such as SAM, CEM, ASMF and ACE is tested using various hyperspectral scenes. To evaluate their performance, metrics such as Matthews correlation coefficient (MCC), visibility score (VIS) and receiver operating characteristic curve (ROC) are used. This analysis serves as a basis for the FPGA implementation, choice of algorithms and later verification of the implemented algorithms.

3.1 Hyperspectral datasets

In this thesis, five hyperspectral datasets have been used, namely, Salinas, Pavia, Indian Pines, Hopavågen and HyMap Cooke City. Salinas, Pavia and Indian Pines dataset are publicly available on [38] and preprocessed as described in [39], while HyMap scene can be found on [40]. They have been chosen due to the fact that the datasets contain known ground truth, which maps real objects and spectral features to the image data. Therefore, ground truth represents essential information for target detection algorithm testing. Hopavågen dataset provides a scene related to the HYPSO project, while other datasets do not contain oceanographic scenes.

3.1.1 Salinas scene

Salinas scene has been collected by Airborne Visible / Infrared Imaging Spectrometer (AVIRIS) with 224 spectral bands over Salinas Valley, California, USA. The spatial resolution of the scene is $3.7m$, which is considered very high in terms of hyperspectral imaging. The scene has width of 512 pixels and height of 217 pixels, forming a cube with 16 classes of data (endmembers), such as vegetables, bare soils and vineyard fields. It is important to mention that not all 224 spectral bands are used, but 20 water absorption bands are discarded. In Table 3.1, number of samples for each present endmember of Salinas scene are shown. The endmembers are mapped to form the ground truth map and depicted as in Fig. 3.1.

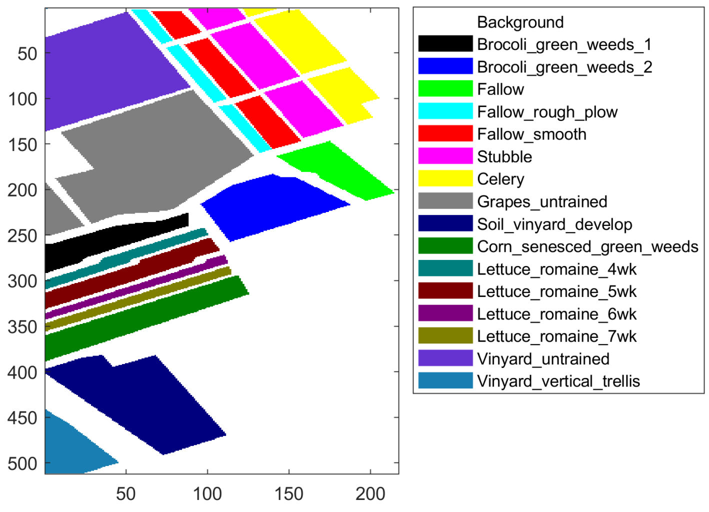


Figure 3.1: Salinas ground truth map

Table 3.1: Salinas ground truth endmembers

Number	Endmember	Samples
1	Broccoli_green_weeds_1	2009
2	Broccoli_green_weeds_2	3726
3	Fallow	1976
4	Fallow_rough_plow	1394
5	Fallow_smooth	2678
6	Stubble	3959
7	Celery	3579
8	Grapes_untrained	11271
9	Soil_vineyard_develop	6203
10	Corn_senesced_green_weeds	3278
11	Lettuce_romaine_4wk	1068
12	Lettuce_romaine_5wk	1927
13	Lettuce_romaine_6wk	916
14	Lettuce_romaine_7wk	1070
15	Vinyard_untrained	7268
16	Vinyard_vertical_trellis	1807

3.1.2 Hopavågen scene

In 2018, by joint efforts of Trondheim Biological Station and the Department of Engineering Cybernetics at NTNU, Hopavågen hyperspectral dataset was acquired. The image has been captured using commercial hyperspectral camera for underwater imaging (Ecotone) on an UAV as a prototype for the HYPSO mission. The resulting scene has 158 by 282 pixels, with corresponding 86 spectral bands. All classified endmembers and number of samples are listed in Table 3.2. The ground truth map is shown in Fig. 3.2.

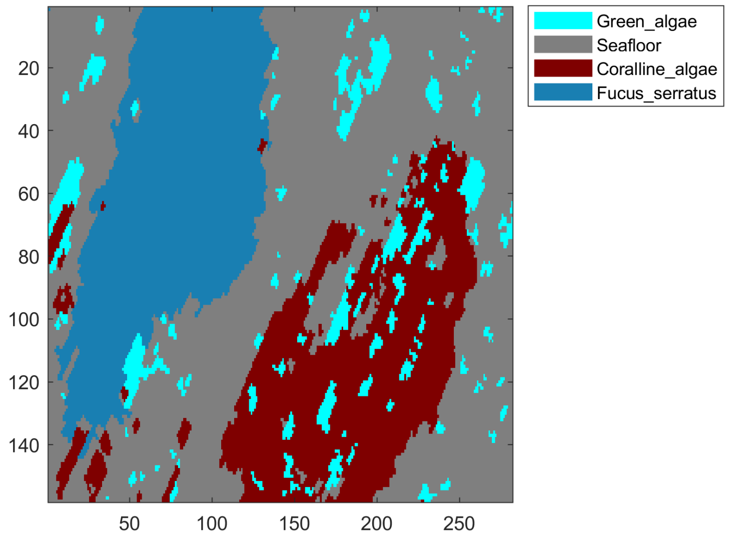


Figure 3.2: Hopavågen ground truth map

Table 3.2: Hopavågen ground truth endmembers

Number	Endmember	Samples
1	Green_algae	3021
2	Seafloor	1428
3	Coralline_algae	830
4	Fucus_serratus	237

3.1.3 Indian Pines scene

Indian Pines is another dataset collected by the AVIRIS sensor over Northwest Indiana, USA. The scene has height and width of 145 pixels with 224 spectral reflectance bands. Again, water absorption bands are discarded, thus reducing the number of bands to 200.

Present endmembers and their sample count is listed in Table 3.3, while ground truth map is shown in Fig. 3.3.

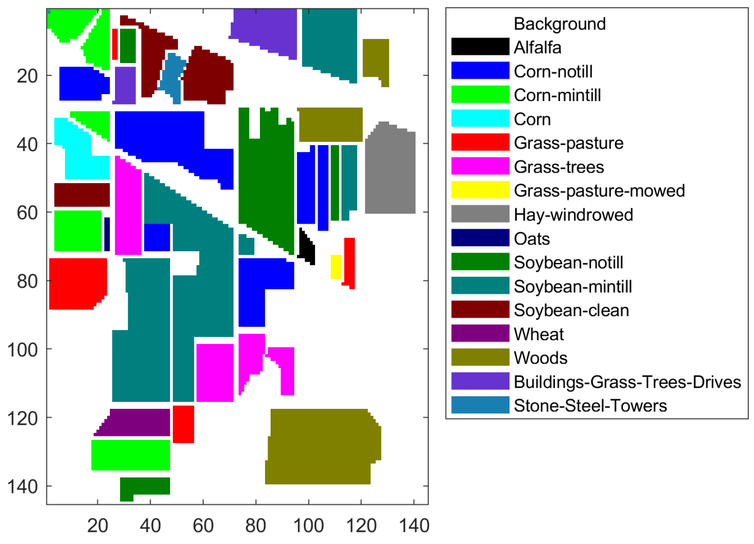


Figure 3.3: Indian Pines ground truth map

Table 3.3: Indian Pines ground truth endmembers

Number	Endmember	Samples
1	Alfalfa	46
2	Corn-notill	1428
3	Corn-mintill	830
4	Corn	237
5	Grass-pasture	483
6	Grass-trees	730
7	Grass-pasture-mowed	28
8	Hay-windrowed	478
9	Oats	20
10	Soybean-notill	972
11	Soybean-mintill	2455
12	Soybean-clean	593
13	Wheat	205
14	Woods	1265
15	Buildings-Grass-Trees-Drives	386
16	Stone-Steel-Towers	93

3.1.4 Pavia University scene

Pavia University scene has been acquired using Reflective Optics System Imaging Spectrometer (ROSIS) during a flight campaign over Pavia, northern Italy. The spatial resolution of the dataset is $1.3m$. The image has width of 610 pixels and height of 340 pixels, with 103 spectral reflectance bands. All endmembers with their abundances are reported in Table 3.4 and mapped in Fig. 3.4.

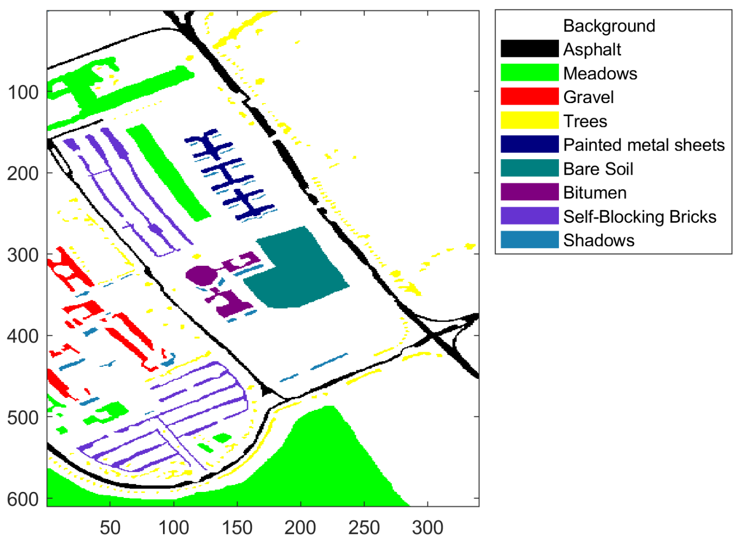


Figure 3.4: Pavia University ground truth map

Table 3.4: Pavia University ground truth endmembers

Number	Endmember	Samples
1	Asphalt	6631
2	Meadows	18649
3	Gravel	2099
4	Trees	3064
5	Painted metal sheets	1345
6	Bare Soil	5029
7	Bitumen	1330
8	Self-Blocking Bricks	3682
9	Shadows	947

3.1.5 HyMap Cooke City scene

HyMap dataset [41] includes atmospherically compensated hyperspectral images of Cooke City, Montana, USA, with the width of 800 pixels and the height of 180 pixels. The dataset is provided by the Digital Imaging and Remote Sensing Group in Center for Imaging Science at Rochester Institute of Technology [42], and it has been used successfully in previous studies [21, 32]. The pixels are composed of 126 spectral bands between $0.45\mu m$ and $2.4\mu m$, with the ground resolution of approximately 3m.

In the area marked by red square in Fig. 3.5, four types of real colored panels (Fig. 3.6) were placed. The details about the panels are listed in Table 3.5. Additionally, the dataset includes the exact positions in form of region of interest (ROI) files, as well as standard spectral library (SPL) files of the targets. The ROI is shown in Fig. 3.7(a) for the marked area. Spectral signatures of the targets are plotted in Fig. 3.7(b).

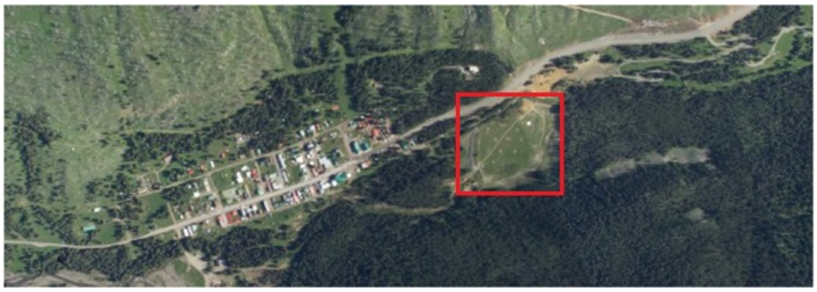
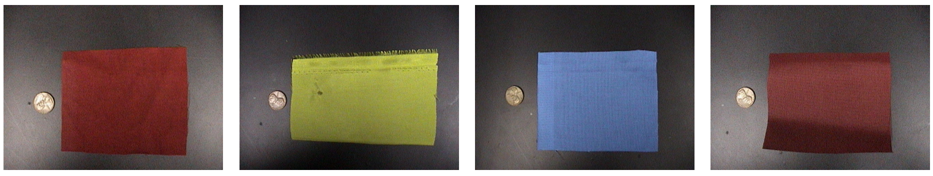


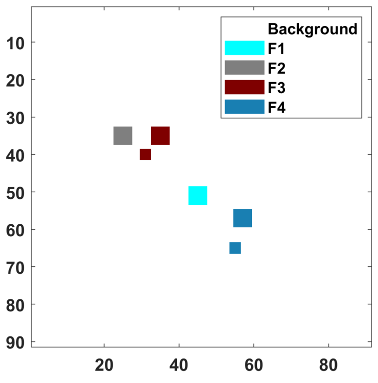
Figure 3.5: True color image of Cooke City scene



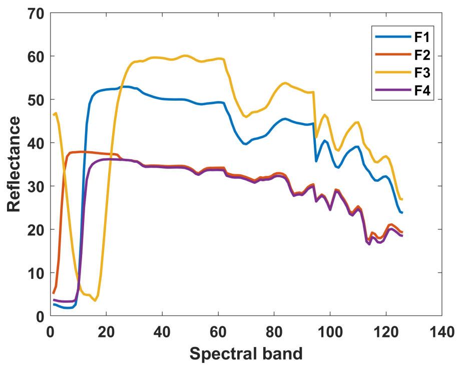
(a) F1 (b) F2 (c) F3 (d) F4

Figure 3.6: Photos of placed targets in Cooke City scene - F1, F2, F3 and F4.

Pixels defined by regions of interest are all regarded as target pixels, and include full-pixels, sub-pixels, and guard-pixels (border-pixels). Therefore, different ground truth maps can be generated based on the ROIs. In this thesis, ground truth is formed based on full-pixel and sub-pixel candidates. Only targets F1 and F2 can form near-full pixel targets due to their size and the ground resolution, while F3 and F4 are always sub-pixel.



(a) ROI



(b) Signatures

Figure 3.7: Region of interest and signatures in HyMap Cooke City scene

Table 3.5: HyMap Cooke City ground truth - targets

Target pixel	Size	Type	Full-pixel	Sub-pixel	Guard-pixel	Total
F1	3x3m	Red Cotton	1	8	16	25
F2	3x3m	Yellow Nylon	1	8	16	25
F3a	2x2m	Blue Cotton	1	8	16	25
F3b	1x1m	Blue Cotton	0	1	8	9
F4a	2x2m	Red Nylon	1	8	16	25
F4b	1x1m	Red Nylon	0	1	8	9

3.2 Target detection performance metrics

The evaluation of target detection algorithms is performed using metrics such as Matthews correlation coefficient (MCC), visibility score (VIS) and receiver operating characteristic curve (ROC). They are broadly used in different research areas including machine learning, signal detection theory, radar technologies and medical research. MCC and visibility metrics were previously used in the master thesis on target detection algorithms [39] as a part of HYPSO project in 2018, and they have demonstrated certain advantages over alternative metrics. To make use of the methods previously developed and allow comparisons and compatibility, this thesis adopts aforementioned metrics for binary classification and target detection.

To visualize the performance of a binary classifier (such as target detector), we usually use confusion matrix as depicted in Fig. 3.8. Since the result of target detection is binary, i.e. a target is present in a pixel or not, there are four elements in a confusion matrix: true positives, false positives, false negatives and true negatives. True elements of the matrix are correctly classified pixels: true positives are present targets regarded as detected, while true negatives are confirmed background pixels. On the contrary, undetected targets are false negatives, and background pixels classified as targets are false positives. From the confusion matrix, MCC score and ROC curve are developed.

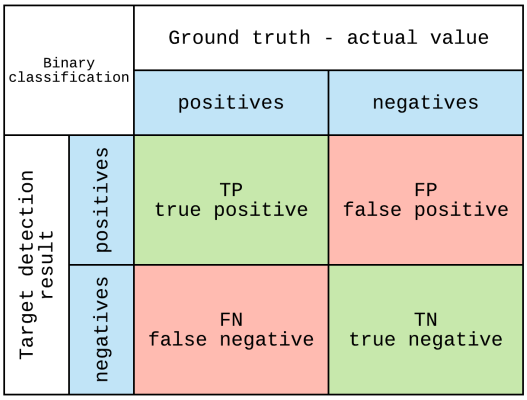


Figure 3.8: Confusion matrix for a binary classifier

3.2.1 Matthews correlation coefficient

MCC metric is defined as:

$$MCC = \frac{tp \cdot tn - fp \cdot fn}{\sqrt{(tp + fp)(tp + fn)(tn + fp)(tn + fn)}} \tag{3.1}$$

where tp are true positive, tn are true negative, fp are false positive and fn are false negative counts. MCC score takes value in the range from -1 to 1. Ideally, MCC score is

1, which means successful detection of all hyperspectral targets without any false positives or negatives. On the other side, $MCC = -1$ indicates that the algorithm always gives the opposite class in case of binary classification. It is important to mention that this metric involves all four elements of the binary confusion matrix.

3.2.2 Receiver operating characteristic curve

Receiver operating characteristic (ROC) is a statistical method employed to show the information of confusion matrices produced for a range of threshold values. The ROC curve was first used to analyze radar signals before the application has been expanded to other fields, such as signal detection theory.

Each detection threshold produces a corresponding confusion matrix. To construct a ROC curve, true positive rate (TPR) is plotted against the false positive rate (FPR), for an array of thresholds, where rates are calculated as:

$$\begin{aligned} TPR &= \frac{TP}{TP+FN} \\ FPR &= \frac{FP}{FP+TN}. \end{aligned} \tag{3.2}$$

The ROC space with example curves is shown in Fig. 3.9. In general, detection performance of the algorithm is higher as ROC curve approaches maximum boundary, annotated as *Perfect Classification* in the figure. Two example ROC curves have corresponding area under curve (AUC), which varies between 0 and 1. Example curve 1 corresponds to a detector with better detection performance and higher AUC, when compared to the curve 2. ROC data can also be plotted as logarithmic scale for the x-axis.

3.2.3 Visibility score

Using the visibility metric [43], the robustness of a detection algorithm can be evaluated. The robustness is defined as a measure of the algorithm's ability to separate background pixels and target pixels. Therefore, visibility score is calculated as:

$$Visibility = \frac{\mid T_t - T_b \mid}{T_{max} - T_{min}} \tag{3.3}$$

where T_t is the average detection statistic for target pixels, and T_b is the average detection statistic for non-target pixels. Factors T_{max} and T_{min} are the maximum and minimum evaluated detection statistics in the scene for a given algorithm, respectively. The best and the maximum score of visibility is 1, and the lowest score is 0.

3.3 Target detection algorithm adaptations

Not all target detection algorithms explained in section 2.3 meet real-time requirements for operation on FPGA platforms. In this context, real-time operation signifies processing of the pixels as soon as they are acquired by the imager, and no later than the next frame is

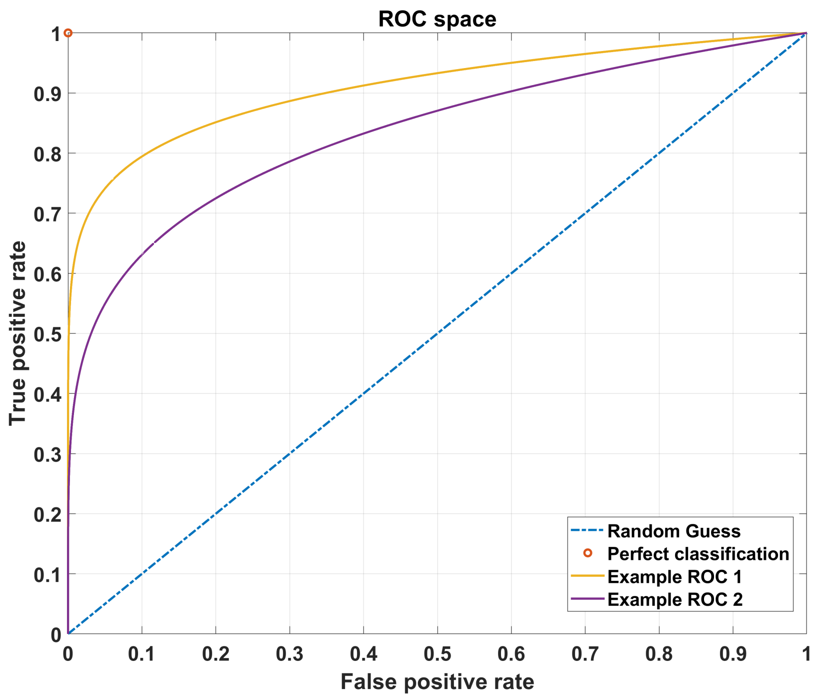


Figure 3.9: The ROC space and example ROC curves

captured. Therefore, they have been adapted in terms of background estimation methods used in algorithm operation. In this section, the ACE-R adaptation is introduced prior to performance testing, while adaptation of inverse matrix calculation for target detection algorithms is elaborated in section 4.2.1.

3.3.1 Adaptive Cosine Estimator using correlation matrix

As defined in Eq. 2.17, adaptive cosine estimator operates on three variables: pixel under test $\mathbf{x}$, target signature $\mathbf{s}$ and the inverse of the covariance matrix $\mathbf{\Sigma}^{-1}$. As such, the algorithm is not prepared for real-time operation, since the covariance matrix is estimated using de-meaned hyperspectral image data (Eq. 2.8). In other words, this approach requires acquisition of the entire hyperspectral cube prior to processing. Therefore, ACE has been adaptively adjusted to use the inverse of the correlation matrix $\mathbf{R}^{-1}$ as follows:

$$D_{ACE-R}(\mathbf{x}) = \frac{(\mathbf{s}^T\mathbf{R}^{-1}\mathbf{x})^2}{(\mathbf{s}^T\mathbf{R}^{-1}\mathbf{s})(\mathbf{x}^T\mathbf{R}^{-1}\mathbf{x})}. \tag{3.4}$$

Similar modifications of other algorithms have been made in [17, 44], with the claims that both solutions provide similar results. The proposed adaptation has been analyzed and compared with other algorithms in the remainder of this chapter.

3.4 Algorithm testing

In order to choose the adequate algorithm for FPGA implementation, detector performance has been considered and comparatively evaluated. The detectors have been tested using aforementioned hyperspectral scenes with full image dimensionality. Introduced performance metrics are calculated for each endmember of the corresponding scene (or each known spectral signature) by iterating through 10000 threshold values ranging from the obtained minimum to the maximum of the probability image. Final MCC values for each endmember are selected as the highest MCC values obtained over all threshold values. Furthermore, MCC values plotted in the following section are the average over all known signatures in the scene. Visibility values are plotted in the same manner.

3.4.1 Salinas scene

The Salinas scene contains 16 endmembers of which many are similar crops planted at different time. This represents a near-homogeneous scene, where target signatures are not fully distinct from the background. Therefore, it is a challenging task to perform target detection on scenes such as Salinas.

The performance of ACE, ACE-R, ASMF (with n equal to 1 and 2), CEM and SAM is shown in Fig. 3.10. It is unambiguous that ACE, ACE-R, ASMF and CEM are able to achieve very high MCC score. However, the visibility is drastically degraded for CEM and SAM. The adapted ACE-R achieves highest visibility score on average of all Salinas endmembers, with the score of 0.45. On the other side, ASMF with $n = 2$ achieved highest average MCC score of 0.7723.

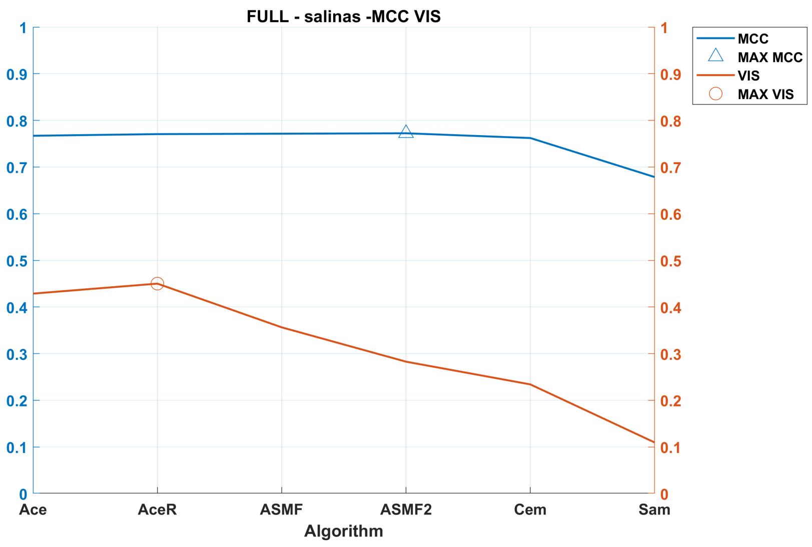


Figure 3.10: MCC and visibility values for Salinas scene

3.4.2 Hopavågen scene

The Hopavågen scene is an oceanographic scene, highly relevant to HYPSO project due to the similar signatures of interest. The scene contains 4 distinct endmembers, which were manually classified using a method similar to the SAM detector.

Performance of the tested algorithms for Hopavågen scene is shown in Fig. 3.11. Interestingly, adapted ACE-R has substantially higher MCC score than the original ACE algorithm with the covariance matrix. Also, ACE-R achieved highest visibility score of 0.2336. ASMF achieved highest MCC score of 0.7937, which is negligibly different from ACE-R (0.7932). CEM also shows good performance, both in terms of MCC and visibility in this scene.

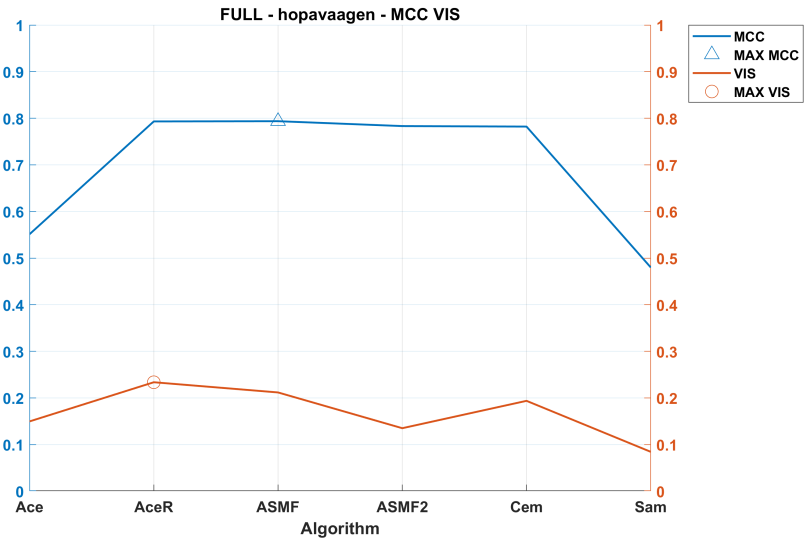


Figure 3.11: MCC and visibility values for Hopavågen scene

3.4.3 Indian Pines scene

The Indian Pines scene was cropped as a part of a larger hyperspectral image. It is another scene dominated by agricultural areas and plants, covering two-thirds of the scene. Apart from that, it contains forest areas, other perennial vegetation and infrastructure such as highway lanes, rail line, smaller roads and housing. It should be noted that 16 present endmembers in the Indian Pines scene are not all mutually exclusive.

As in previously analyzed scenes, ACE-R and ASMF achieved highest MCC and visibility score, 0.6766 and 0.3726, respectively, as shown in Fig. 3.12. However, ASMF has 13% lower visibility score than ACE-R algorithm. SAM has consistently lowest MCC and visibility scores, being unable to fully distinguish between different endmembers and the background. It is interesting to note that the visibility of ASMF drops as n increases, thus reducing the robustness of the algorithm for threshold selection.

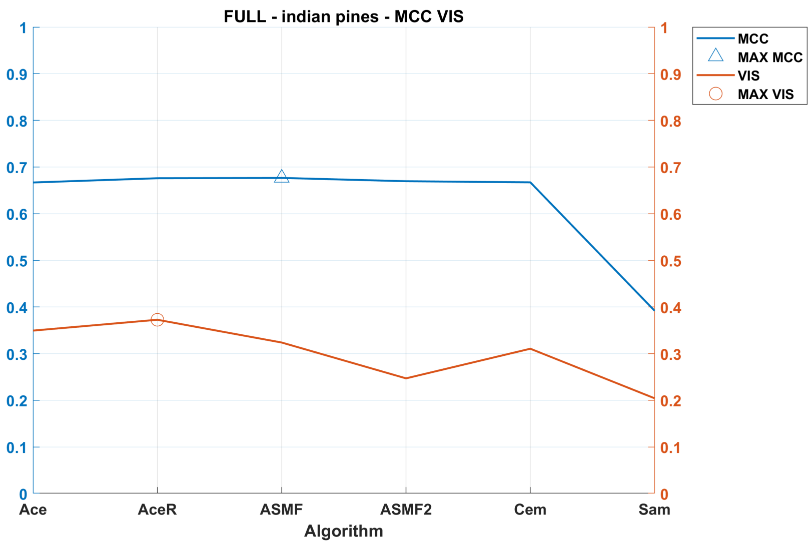


Figure 3.12: MCC and visibility values for Indian Pines scene

3.4.4 Pavia University scene

The detection performance results for Pavia University scene are shown in Fig. 3.13. The scene has 9 distinct endmembers, with large non-classified area regarded as background. None of the endmembers can be considered small targets, which significantly reduces the performance of algorithms with background statistics estimation. Nevertheless, the ASMF and ACE-R were again able to achieve the highest MCC and visibility values, 0.3533 and 0.2423, respectively.

3.4.5 HyMap Cooke City scene

The HyMap Cooke City is the only scene amongst used that is specifically designed for testing target detection algorithms. It contains target ground truth locations, with a very small number of target pixels compared to the image size.

The performance estimation results are shown in Fig. 3.14. ACE, ACE-R and ASMF achieved very high MCC and visibility scores, while CEM and SAM algorithm have not been able to put up with the challenging spectral features in the scene. ACE, ACE-R and ASMF achieved highest MCC score of 0.7608, while ACE achieved highest visibility of 0.6718.

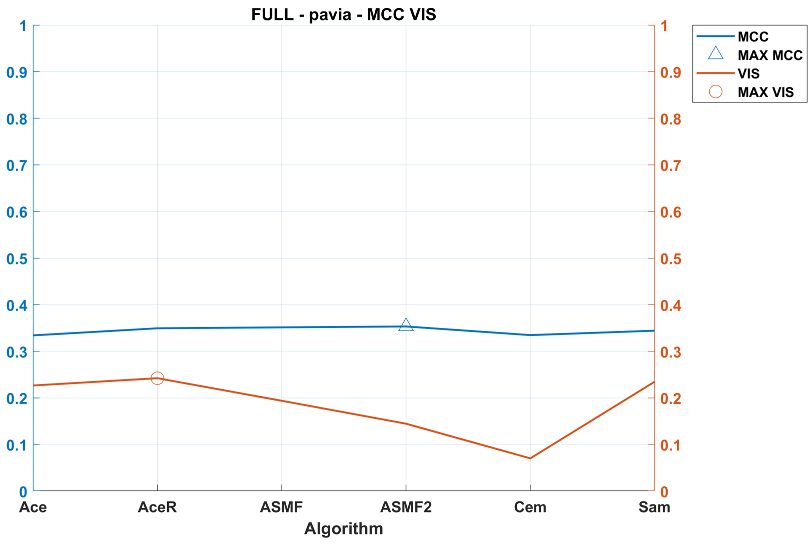


Figure 3.13: MCC and visibility values for Pavia University scene

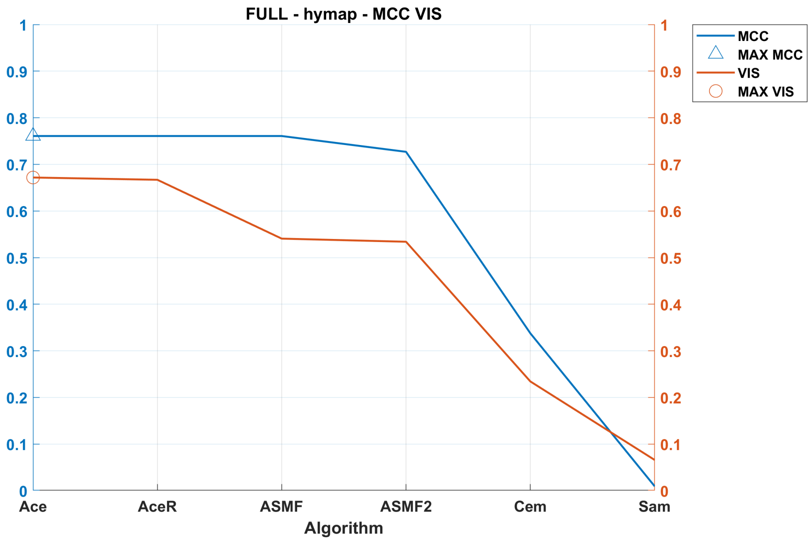


Figure 3.14: MCC and visibility values for HyMap Cooke City scene

Chapter 4

FPGA implementation

4.1 Hardware-Software codesign implementation

To reduce the execution time of target detection algorithms, specialized FPGA accelerator has been developed for Zynq-7000 SoC. The overview of the system is shown in Fig. 4.1. This hardware-software partitioned design features high customizability and flexibility for different hyperspectral imagers and pixel component bit widths. It utilizes DMA to transfer vast amounts of data to the accelerator, while the data is being stored in DDR memory.

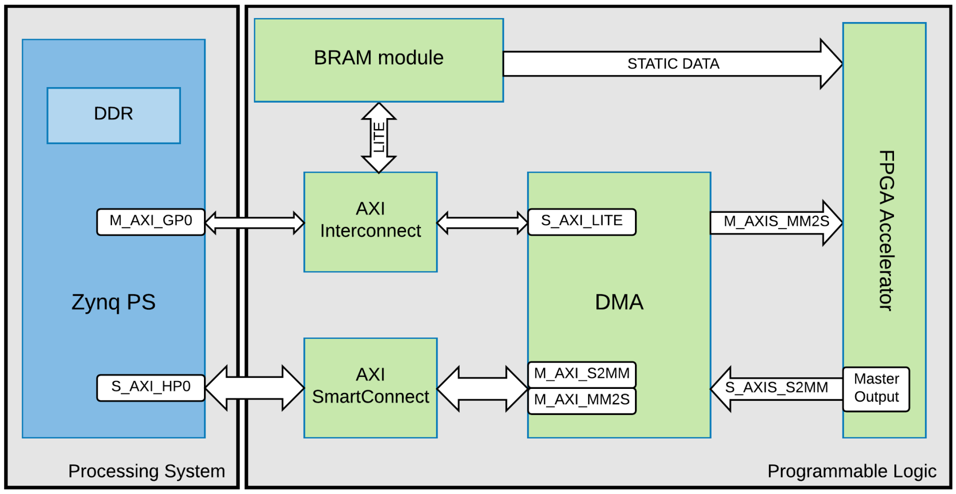


Figure 4.1: Block design of the FPGA accelerator for target detection.

The accelerator core was partly developed during specialization project [45] in autumn semester 2018 as a preparatory research project for the master thesis. Here, the specialization project accomplishments are briefly explained, with the focus on thesis work and improvements to the system in terms of performance, interoperability and testability. The FPGA accelerator is developed for ACE(-R), ASMF, and CEM algorithm. It is possible to synthesize the core for only one algorithm in order to minimize resource utilization, or to synthesize full FPGA core and control its function from the PS via AXI-lite registers.

In the following subsections, input, processing and output logic of the design are explained. It should be noted that during the thesis major speed-up of the system was achieved, as well as interaction of the core with Cube DMA. Furthermore, new detectors have been added to the design, such as ASMF and CEM.

4.1.1 Input logic

DMA module

Two different DMA cores have been used to transfer data to the FPGA accelerator, namely, AXI DMA and Cube DMA. Both DMA modules use AXI streaming interfaces, explained in section 2.5.3. Master stream interface of the DMA initiates the transfers and sends the hyperspectral cube in BIP format to the accelerator. The processed data is then received by the DMA slave interface.

DMA modules are configured in interrupt mode to control the data flow. The configuration is performed from PS in software using the corresponding driver. For AXI DMA, driver provided by *XILINX* has been used to initialize the module and set up the transactions using buffer descriptors, which carry information about the address of the data being transferred, length of the buffer and other control information. The buffers are stored in cache coherent memory, therefore they must be flushed from the cache before allocating a block descriptor to the driver. In the same manner, the receiving buffer has to be invalidated before accessing its elements after DMA transfer has completed.

Furthermore, the FPGA accelerator has been tested using Cube DMA for pixel streaming. Since Cube DMA module is designed for HSI images, it requires additional parameters such as cube dimensions (width, height and depth), format in which the cube is transferred and block dimensions if block transfer mode is enabled. Performance analysis and resource utilization details for different DMA cores is presented in section 5.1.2.

BRAM module

BRAM module shown in Fig. 4.1 and detailed in Fig. 4.2 consists of block RAM modules and AXI-lite register file used to control the accelerator and initialize/update the data stored in BRAM. Block RAMs are used in two port mode, where one port is used for writing from PS, and the other port is read port from PL with separate address buses, as shown in Fig. 2.9.

BRAM module is used to store the inverse of the correlation matrix $\mathbf{R}^{-1}$, the precalculated vector $\mathbf{s}^T\mathbf{R}^{-1}$ and the value $\mathbf{s}^T\mathbf{R}^{-1}\mathbf{s}$, all initialized from PS. The initialization is performed using keyhole writing through AXI-lite registers. As the writing to BRAM from PS should happen infrequently, one AXI slave register is used to sequentially send the data from PS. The customized logic design is used to control BRAM and assert corresponding write enable and write address signals. The register file used to handle the BRAM logic is shown in Table 4.1.

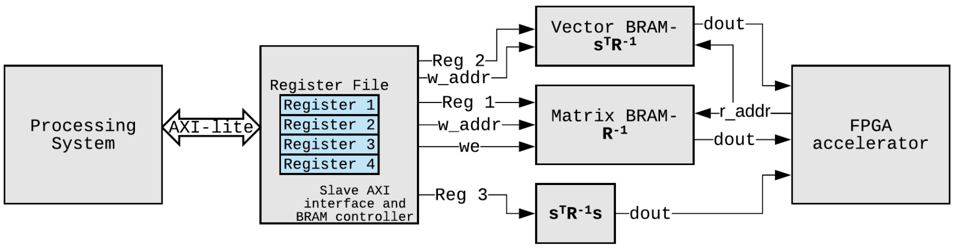


Figure 4.2: Block diagram of BRAM module for HW/SW codesign solution.

Table 4.1: AXI-lite register file description for HW/SW codesign solution

Register	Value	Description
0	Data	Writing Inverse Correlation Matrix elements or reading in Debug Mode
1	Data	Writing precalculated vector $\mathbf{s}^T\mathbf{R}^{-1}$ elements or reading in Debug Mode
2	Data or Address	Writing precalculated value $\mathbf{s}^T\mathbf{R}^{-1}\mathbf{s}$ or reading matrix or vector elements from PS in Debug Mode
3	[0,4]	Selecting Algorithm or Enabling Debug Mode

4.1.2 Processing logic

Processing logic of the FPGA accelerator is divided into three main stages. All stages are pipelined at task-level, allowing them to overlap in their operation. Therefore, the concurrency of the RTL implementation is increased, along with the the overall throughput of the accelerator.

Stage 1

Figure 4.3 shows the first stage in the accelerator pipeline. Uploaded matrix $\mathbf{R}^{-1}$, vector $\mathbf{s}^T\mathbf{R}^{-1}$ and incoming pixel $\mathbf{x}$ are used to calculate intermediate results $\mathbf{x}^T\mathbf{R}^{-1}$ and $\mathbf{s}^T\mathbf{R}^{-1}\mathbf{x}$, respectively. Corresponding matrix-vector and vector-vector products are performed using deeply pipelined dot product units placed on dedicated DSP blocks.

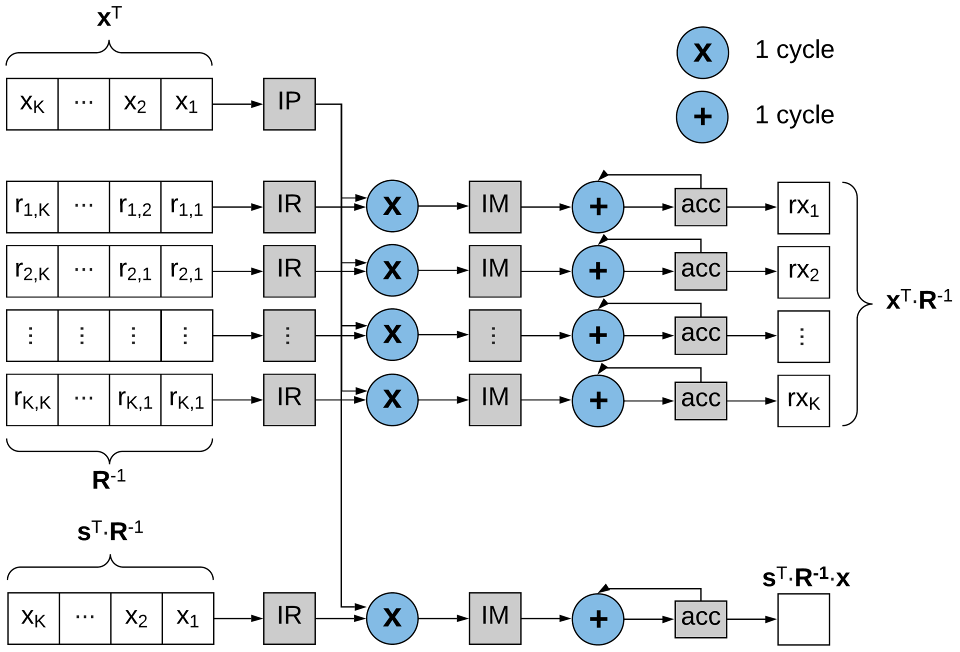


Figure 4.3: RTL design of the Stage 1 of the FPGA accelerator.

The stage 1 performs the following vector-matrix product:

$$\begin{bmatrix} x_1 & x_2 & \dots & x_K \end{bmatrix} * \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & \dots & r_{1,K} \\ r_{2,1} & r_{2,2} & r_{2,3} & \dots & r_{2,K} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{K,1} & r_{K,2} & r_{K,3} & \dots & r_{K,K} \end{bmatrix},$$

where x_i is a pixel component $x_i = L_p(\lambda_i)$. After $K + delay$ clock cycles, we obtain the result:

$$\mathbf{x}^T\mathbf{R}^{-1} = \left[\mathbf{x}^T \cdot \mathbf{row}_1(\mathbf{R}^{-1}) \quad \mathbf{x}^T \cdot \mathbf{row}_2(\mathbf{R}^{-1}) \quad \dots \quad \mathbf{x}^T \cdot \mathbf{row}_K(\mathbf{R}^{-1})\right].$$

K is number of spectral components in a pixel and delay corresponds to the pipeline delay of dot product units. Additionally, synchronous BRAM latency is incorporated in the mentioned delay. In parallel with this, dot product of vectors $\mathbf{x}$ and $\mathbf{s}^T\mathbf{R}^{-1}$ produces result $\mathbf{s}^T\mathbf{R}^{-1}\mathbf{x}$.

Stage 2

The inputs of stage 2 are pixel under test $\mathbf{x}$ (as in stage 1, delayed using a shift register) and results of stage 1, $\mathbf{x}^T\mathbf{R}^{-1}$ and $\mathbf{s}^T\mathbf{R}^{-1}\mathbf{x}$. The RTL design is shown in Fig. 4.4. Execution of stage 2 takes $K + 1 + delay$ clock cycles. Two parallel execution paths are balanced using additional delay registers. The second stage of the pipeline produces two results, p_1 and p_2, which depend on the chosen target detection algorithm. Table 4.2 summarizes the control signals and corresponding products of this stage. CEM detector does not require any of the intermediate results produced in stage 2.

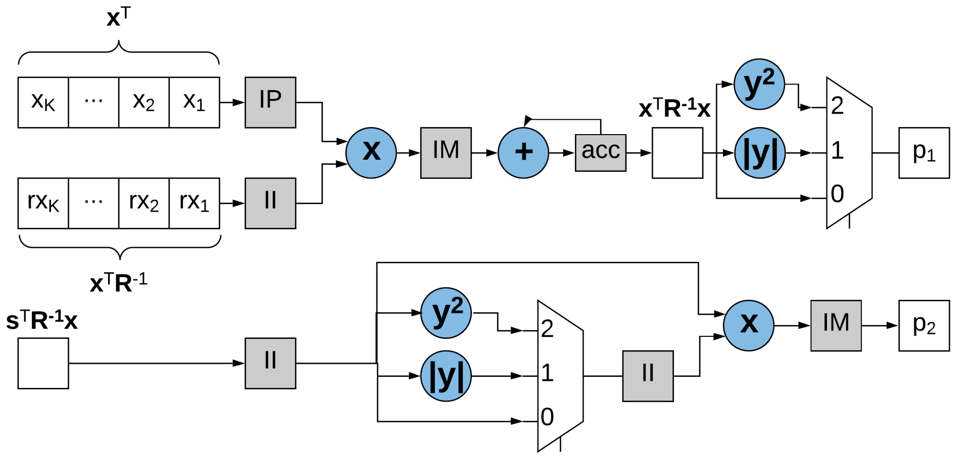


Figure 4.4: RTL design of the Stage 2 of the FPGA accelerator.

Table 4.2: Stage 2 multiplexer signals and products

Algorithm	Multiplexer signal	Product p_1	Product p_2
ACE(-R)	0	$\mathbf{x}^T\mathbf{R}^{-1}\mathbf{x}$	$(\mathbf{s}^T\mathbf{R}^{-1}\mathbf{x})^2$
ASMF	1	$\lvert\mathbf{x}^T\mathbf{R}^{-1}\mathbf{x}\rvert$	$(\mathbf{s}^T\mathbf{R}^{-1}\mathbf{x})\lvert\mathbf{s}^T\mathbf{R}^{-1}\mathbf{x}\rvert$
ASMF-2	2	$(\mathbf{x}^T\mathbf{R}^{-1}\mathbf{x})^2$	$(\mathbf{s}^T\mathbf{R}^{-1}\mathbf{x})\lvert\mathbf{s}^T\mathbf{R}^{-1}\mathbf{x}\rvert^2$
CEM	any	N/A	N/A

Stage 3

The last stage in this implementation is shown in Fig. 4.5. The signals to the divider are multiplexed, ultimately deciding what detection statistic is being produced. The multiplexer signals are derived from the selection register in the AXI-lite register file. Table 4.3 summarizes the final results.

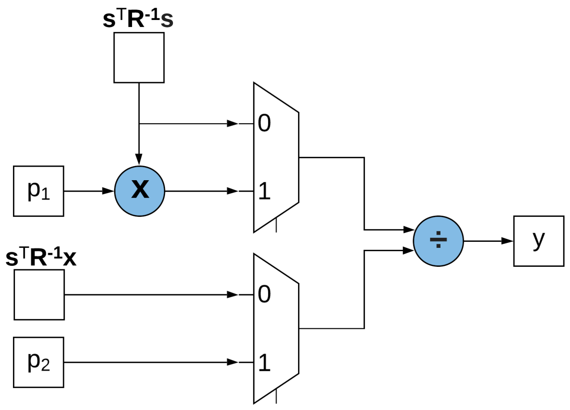


Figure 4.5: RTL design of the Stage 3 of the FPGA accelerator.

Table 4.3: Stage 3 multiplexer signals and products

Algorithm	Multiplexer signal	Result y
ACE(-R)	1	$p_2/(p_1(\mathbf{s}^T\mathbf{R}^{-1}\mathbf{s}))$
ASMF	1	$p_2/(p_1(\mathbf{s}^T\mathbf{R}^{-1}\mathbf{s}))$
ASMF-2	1	$p_2/(p_1(\mathbf{s}^T\mathbf{R}^{-1}\mathbf{s}))$
CEM	0	$(\mathbf{s}^T\mathbf{R}^{-1}\mathbf{x})/(\mathbf{s}^T\mathbf{R}^{-1}\mathbf{s})$

Task-level pipeline

The stages of the accelerator are pipelined as shown in Fig. 4.6, where the time-line initiates as first pixel components arrive to the accelerator. Stages 1 and 2 have latency of K clock cycles, while *ASMF-2* implementation has an additional delay in stage 2 (which overlaps with the next pixel calculation), as marked with light-blue color in the figure. The latency of stage 3 is dominated by the divider, which for 32-bit inputs has latency of 71 clock cycle. Nevertheless, the divider is able to absorb a new set of inputs on each cycle, masking the relatively long latency and maintaining the high throughput.

In summary, the throughput of the accelerator is one output per K clock cycles for ACE, ASMF and CEM detector, where K is the number of spectral bands in a hyperspectral image. When CEM is activated, stage 2 is not performed.

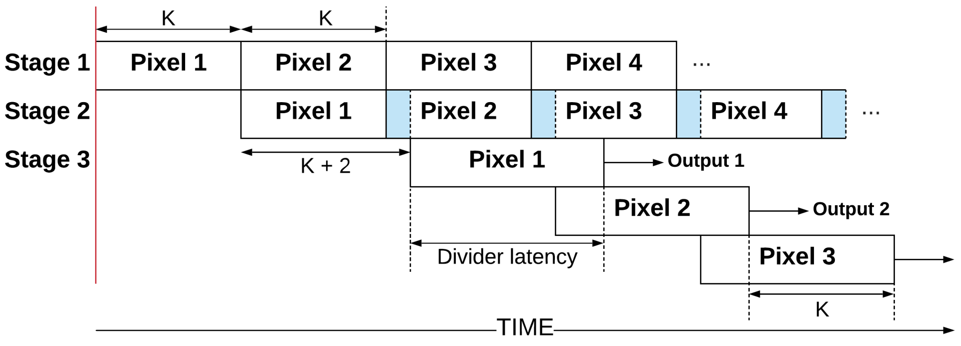


Figure 4.6: Task-level pipeline of the accelerator stages.

Fixed-point considerations

The accelerator has been designed to operate on fixed-point values, with full consideration of adequate bit widths for each stage in the pipeline. Therefore, using generic values it is possible to adapt the design to the new hyperspectral data with different bit width of inputs. Table 4.4 shows the generics, their description and example values for an image with 16 spectral bands, where each band component is represented with 16-bit value.

Table 4.4: Generics and example values for HW/SW codesign solution

Generic value	Description	Default value
NUM_BANDS	Number of bands in the hyperspectral image	16
PIXEL_DATA_WIDTH	Bit width of each component in a pixel	16
BRAM_DATA_WIDTH	Bit width of pre-processed data from PS	32
BRAM_ADDR_WIDTH	Address bit width	4
BRAM_ROW_WIDTH	Matrix full row/column bit width	512
ST2IN_DATA_WIDTH	Bit width of input data for stage 2	32
ST3IN_DATA_WIDTH	Bit width of input data for stage 3	32
OUT_DATA_WIDTH	Bit width of divider inputs	32
ST2IN_DATA_SLIDER	Slider cutting the output of stage 1 to fit as an input of stage 2	50
ST3IN_DATA1_SLIDER	Slider cutting the output of stage 2 to fit as an input of stage 3	50
ST3IN_DATA2_SLIDER	Slider cutting the output of stage 2 to fit as an input of stage 3	62
OUT_DATA1_SLIDER	Slider cutting the output of stage 3 to fit as input of the divider	62
OUT_DATA2_SLIDER	Slider cutting the output of stage 3 to fit as input of the divider	31

Special attention was given to the method for truncation of multiplier products (and MAC products). Sliders present in the table are used to adjust the start and stop bits of the truncation manually, along with output bit width generics. Slider value is the desired MSB bit position where the cut starts. In other words, the result is a consecutive slice of bits, starting at position pointed by slider value and ending at position *SLIDER-OUT_WIDTH+1*.

One example is shown in Fig. 4.7 featuring a dot product unit. Bit widths of each signal are annotated in the figure, including the accumulator which is $ceil(log_2(K))$ bits wider than result of multiplication. After K MAC operations, the result is truncated to the corresponding output data width using a slider, as depicted in the bottom half of the figure.

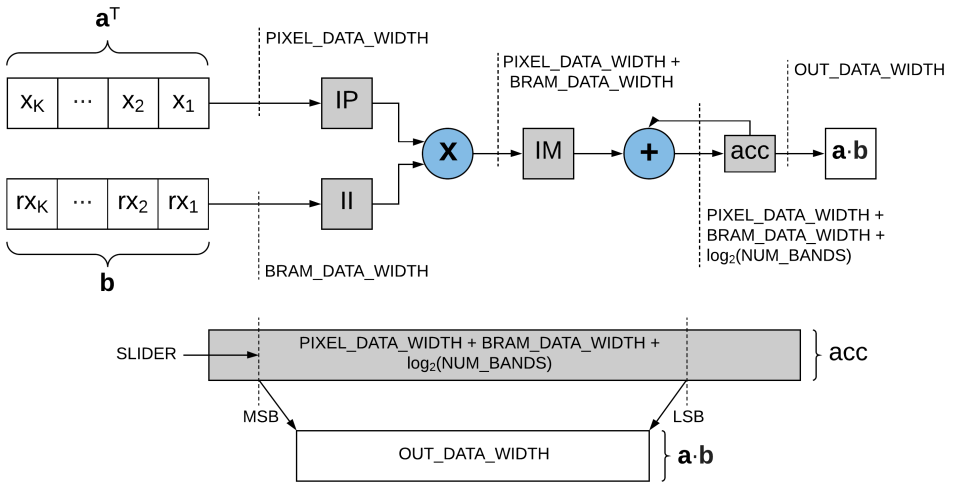

aᵀ
xK
···
x2
x1
PIXEL_DATA_WIDTH
IP
rxK
rx2
rx1
II
b
BRAM_DATA_WIDTH
X
PIXEL_DATA_WIDTH + BRAM_DATA_WIDTH
IM
+
acc
OUT_DATA_WIDTH
a·b
PIXEL_DATA_WIDTH + BRAM_DATA_WIDTH + log2(NUM_BANDS)
SLIDER
PIXEL_DATA_WIDTH + BRAM_DATA_WIDTH + log2(NUM_BANDS)
acc
MSB
LSB
OUT_DATA_WIDTH
a·b

Figure 4.7: Example of a dot product unit with specified bit widths. Using a slider to truncate the result at a specified position.

4.1.3 Output logic

Master Output is a module designed to comply with AXI-stream master interface and communicate with DMA or other stream interfaces. The module was designed during specialization project, so it is briefly mentioned here to keep the thesis self-contained.

The module resembles *AXI-4 Stream FIFO* IP provided by Xilinx, where processed detection statistic values are stored in intermediate registers or BRAM blocks, depending on the choice made during the synthesis. After a defined number of values are read, this module streams the stored values to the DMA core (Fig. 4.8).

This module is an important part of the design since it communicates with DMA and generates control signals that affect the whole hardware pipeline. For instance, if DMA cannot accept the stream coming from the output module, then the pipeline has to be stalled until DMA is free again. The output module accepts two VHDL generics: *DATA_WIDTH* and *PACKET_SIZE* which represents the depth of the packet FIFO. It accepts the incoming data until it is full, when it starts streaming to DMA (if DMA is ready). It asserts *TVALID* signal and puts data on the bus, compliant to the AXI protocol. As soon as the DMA asserts *TREADY*, the transfer begins and *read pointer* is incremented. As the last element of a packet is being transferred, *TLAST* can be set to let the DMA know that the transfer is complete. If the DMA is not ready and the packet FIFO is full, the stream is stalled. It should be noted that packet mode can be disabled.

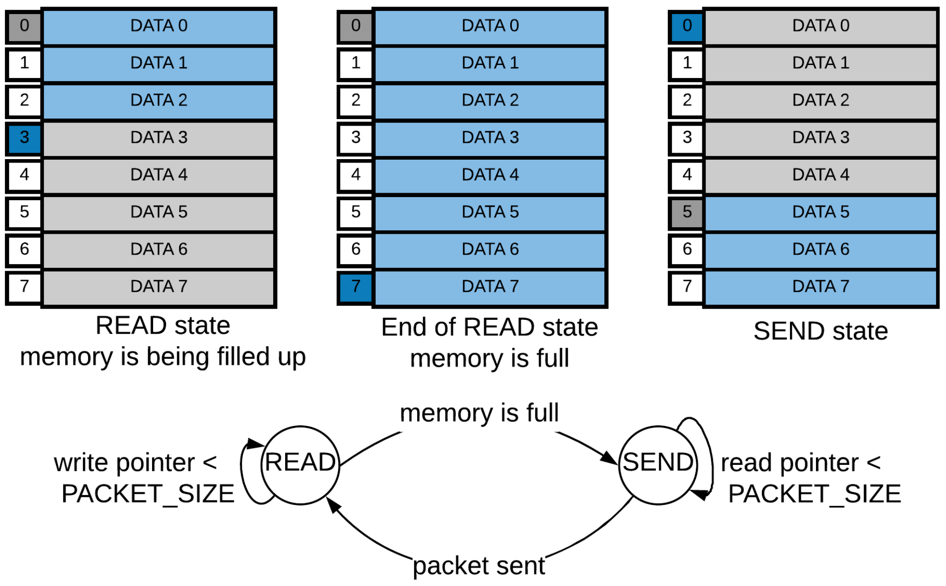


Figure 4.8: An example of packet FIFO behaviour with 8 elements.

4.1.4 Software solution for using the FPGA accelerator

Software solution used to control the FPGA accelerator on Zynq SoC platform is developed in C programming language. The program is executed on Zynq PS with the following procedure:

1. Processing system is initialized
2. Hyperspectral cube is loaded from the SD card to DDR memory
3. Designated target ($\mathbf{s}$) is loaded from the SD card to DDR memory
4. Pre-processing of the data is performed
    - Correlation matrix $\mathbf{R}$ is calculated using the entire cube or its subset
    - Inverse of the correlation matrix $\mathbf{R}^{-1}$ is calculated
    - Intermediate values $\mathbf{s}^T\mathbf{R}^{-1}$ and $\mathbf{s}^T\mathbf{R}^{-1}\mathbf{s}$ are obtained
    - Scaling factors for $\mathbf{R}^{-1}$, $\mathbf{s}^T\mathbf{R}^{-1}$ and $\mathbf{s}^T\mathbf{R}^{-1}\mathbf{s}$ are found
    - $\mathbf{R}^{-1}$, $\mathbf{s}^T\mathbf{R}^{-1}$ and $\mathbf{s}^T\mathbf{R}^{-1}\mathbf{s}$ are scaled to fit the predefined bit width for the FPGA accelerator
5. BRAM module is configured via AXI-lite register file
    - Debug mode is disabled while writing
    - Scaled inverse of the correlation matrix $\mathbf{R}^{-1}$ is written to BRAM
    - Scaled values $\mathbf{s}^T\mathbf{R}^{-1}$ and $\mathbf{s}^T\mathbf{R}^{-1}\mathbf{s}$ are written to BRAM
    - Algorithm choice is made using register number 4
6. DMA transfer is configured and initiated
7. Operation of FPGA accelerator starts as the first pixel component is received
8. As the last pixel is processed and detection statistic received by the DMA, cache is invalidated
9. (Optional) Resulting probability image is compared with the threshold value to obtain the detection map
10. Probability image and other results are written to the SD card

The steps for integration of software and hardware modules on Zynq SoC are explained in Appendix B.

4.2 Full FPGA implementation

To achieve (near) real-time operation of target detection algorithms, full FPGA solution has been developed for Zynq-7000 SoC. The system has the structure as shown in Fig. 4.9. As in HW/SW implementation, DMA is used to stream the vast amounts of hyperspectral data to the FPGA core, while configuration parameters are uploaded through dedicated AXI-lite register file in the initialization module. This implementation also includes ACE, ASMF and CEM detectors, which are highly integrated with Sherman-Morrison updating to achieve high performance and low resource utilization. The blocks presented in the figure are described in detail in the following text.

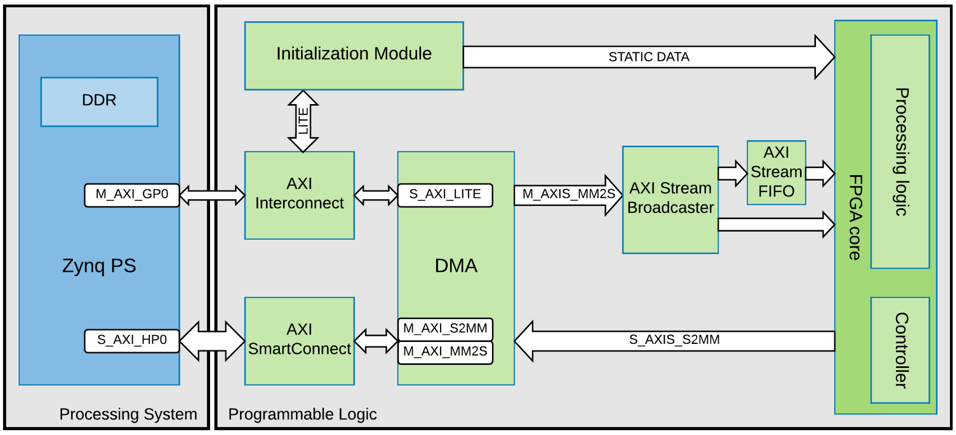


Figure 4.9: Block design of the full FPGA solution for target detection.

4.2.1 Adaptation of inverse matrix calculation for FPGA implementation

Assuming that the correlation matrix **R** is properly trained for a certain ground area, it is possible to use the accelerator developed in section 4.1 for subsequent runs of the target detection algorithms. Otherwise, the correlation matrix is estimated using the captured hyperspectral image. With the assumption that the hyperspectral image is stored in BIP format, the correlation matrix can be estimated from the data as following:

$$\mathbf{R} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^T, \tag{4.1}$$

resulting in a square symmetric matrix:

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & \dots & r_{1K} \\ r_{21} & r_{22} & r_{23} & \dots & r_{2K} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{K1} & r_{K2} & r_{K3} & \dots & r_{KK} \end{bmatrix}.$$

The inverse can be found using methods explained in 2.4, but we will put the focus on Sherman-Morrison formula as in Eq. 4.3. Sherman-Morrison formula has been successfully used for target detection in [32, 33]. Using **R**, we can define matrix **S** as:

$$\mathbf{S} = N \cdot \mathbf{R}, \tag{4.2}$$

thus eliminating the constant N in the further calculations. For the forthcoming pixel $\mathbf{x}_i$, the inverse can be updated with the assumption of having a matrix inverse $\mathbf{S}_{i-1}^{-1}$, thus obtaining $\mathbf{S}_i^{-1}$:

$$\mathbf{S}_i^{-1} = (\mathbf{S}_{i-1} + \mathbf{x}_i\mathbf{x}_i^T)^{-1} = \mathbf{S}_{i-1}^{-1} - \frac{\mathbf{S}_{i-1}^{-1}\mathbf{x}_i\mathbf{x}_i^T\mathbf{S}_{i-1}^{-1}}{1 + \mathbf{x}_i^T\mathbf{S}_{i-1}^{-1}\mathbf{x}_i}. \tag{4.3}$$

The initial inverse matrix $\mathbf{S}_0^{-1}$ can be set to:

- an identity matrix **I** multiplied with a scaling factor β [32] or
- a (pseudo) inverse of correlation matrix for a certain percentage of pixels.

Sliding-window adaptation

As described in [32], this adaptation includes a sliding window of P pixels surrounding a pixel under test used to estimate the correlation matrix. Mathematically, correlation matrix corresponding to the pixel $\mathbf{x}_i$ is expressed as:

$$\begin{aligned} \mathbf{R}_i &= \frac{1}{P}\sum_{n=i-P/2}^{i+P/2} \mathbf{x}_n\mathbf{x}_n^T \\ \mathbf{S}_i &= \sum_{n=i-P/2}^{i+P/2} \mathbf{x}_n\mathbf{x}_n^T \end{aligned} \tag{4.4}$$

therefore, the update of the relative matrix **S** is performed as following:

$$\mathbf{S}_i = \mathbf{S}_{i-1} + (\mathbf{x}_{i+P/2}\mathbf{x}_{i+P/2}^T - \mathbf{x}_{i-P/2}\mathbf{x}_{i-P/2}^T). \tag{4.5}$$

The inverse of the matrix is again obtained using Sherman-Morrison formula, which allows us to update $\mathbf{S}_i^{-1}$ for each incoming pixel in two steps as following:

$$\mathbf{T}^{-1} = (\mathbf{S}_{i-1} + \mathbf{x}_{i+P/2}\mathbf{x}_{i+P/2}^T)^{-1} = \mathbf{S}_{i-1}^{-1} - \frac{\mathbf{S}_{i-1}^{-1}\mathbf{x}_{i+P/2}\mathbf{x}_{i+P/2}^T\mathbf{S}_{i-1}^{-1}}{\mathbf{x}_{i+P/2}^T\mathbf{S}_{i-1}^{-1}\mathbf{x}_{i+P/2} + 1} \tag{4.6}$$

$$\mathbf{S}_i^{-1} = (\mathbf{T} - \mathbf{x}_{i-P/2}\mathbf{x}_{i-P/2}^T)^{-1} = \mathbf{T}^{-1} - \frac{\mathbf{T}^{-1}\mathbf{x}_{i-P/2}\mathbf{x}_{i-P/2}^T\mathbf{T}^{-1}}{1 - \mathbf{x}_{i-P/2}^T\mathbf{T}^{-1}\mathbf{x}_{i-P/2}} \tag{4.7}$$

This adaptation is also feasible for real-time operation, however it adds to the calculation complexity due to the sliding-window. It should be noted that removing pixel $\mathbf{x}_{i-P/2}$ from the estimation does not significantly (or at all) contribute to the improvement of the algorithm detection performance as discussed in [33].

Running Sherman-Morrison update

Tests performed during this thesis have confirmed that the estimation does not suffer from significant degradation if the tailing pixel from sliding window is not removed. Therefore, FPGA core developed in this thesis consistently uses Eq. 4.3 to update the previous estimation of the inverse matrix $\mathbf{S}_{i-1}^{-1}$. Therefore, matrix $\mathbf{S}$ can be expressed as:

$$\mathbf{S} = \frac{1}{\beta} \cdot \mathbf{I} + \mathbf{x}_1\mathbf{x}_1^T + \mathbf{x}_2\mathbf{x}_2^T + \cdots + \mathbf{x}_N\mathbf{x}_N^T \tag{4.8}$$

The adoption of this method increases the dynamic range of values of the matrix $\mathbf{S}^{-1}$ over the number of iterations, thus increasing the required FPGA resources due to extended word length. However, it certainly reduces the execution time of the full FPGA implementation. These claims are further discussed in results section 5.2.

4.2.2 Adaptation of target detectors for real-time processing

ACE-R, CEM and ASMF are target detectors feasible for real-time processing, as they use the correlation matrix to estimate the background statistics. In this implementation, they are integrated with Sherman-Morrison inverse updating method, as follows:

$$D_{CEM-RT}(\mathbf{x}_i) = \frac{\mathbf{s}^T\mathbf{S}_{i+k}^{-1}\mathbf{x}_i}{\mathbf{s}^T\mathbf{S}_{i+k}^{-1}\mathbf{s}}, \tag{4.9}$$

$$D_{ACE-RT}(\mathbf{x}_i) = \frac{(\mathbf{s}^T\mathbf{S}_{i+k}^{-1}\mathbf{x}_i)^2}{(\mathbf{s}^T\mathbf{S}_{i+k}^{-1}\mathbf{s})(\mathbf{x}_i^T\mathbf{S}_{i+k}^{-1}\mathbf{x}_i)}, \tag{4.10}$$

$$D_{ASMF-RT}(\mathbf{x}_i) = \frac{\mathbf{s}^T\mathbf{S}_{i+k}^{-1}\mathbf{x}_i}{\mathbf{s}^T\mathbf{S}_{i+k}^{-1}\mathbf{s}} \cdot \left|\frac{\mathbf{s}^T\mathbf{S}_{i+k}^{-1}\mathbf{x}_i}{\mathbf{x}_i^T\mathbf{S}_{i+k}^{-1}\mathbf{x}_i}\right|^n. \tag{4.11}$$

In the Eq. 4.9, 4.10 and 4.11, the target detection results are delayed by k pixels, allowing the initial k updates of the inverse matrix $\mathbf{S}^{-1}$. This behavior is described in MATLAB code segment in Listing 4.1. During this thesis, experiments were performed to determine the relationship between the detection performance and delay k. The experimental results demonstrate that the delay k can be set to K, which is the number of bands in a pixel.

Listing 4.1: Algorithmic level, target detectors and Sherman-Morrison inverse updating

```
% initialization of matrix S
S = (beta) * eye(num_bands,num_bands);

% initial k updates of matrix S
for i = 1:k
  x = M(:,i);
  [S] = ShermanMorrison(S,x);
end

for i = 1:num_pixels
```

```
    % target detection algorithm
     x = M(:,i);
     results(i) = algorithm(x,S,target);

    % remainder of Sherman-Morrison updates
    if(i < num_pixels-k)
       x = M(:,i+k);
       [S] = ShermanMorrison(S,x);
    end
end
```

4.2.3 Input logic

Initialization module

Initialization module shares similar structure with the BRAM module described in section 4.1.1. The core is controlled using AXI-lite register file presented in Table 4.5, where register 0 is used to enable the core operation after the data is initialized using registers 1 and 2.

Table 4.5: AXI-lite register file description for full FPGA solution

Register	Value	Description
0	Control	Enable Core operation
1	Data	Writing elements of the initial matrix $\mathbf{S}_0^{-1}$
2	Data	Writing target spectral signature vector **s**
3	[0,3]	Algorithm selection

The initialization process follows four control steps (shown in detail in Fig. 4.14), starting with state *Idle*, where the FPGA core waits for the initialization to start. The initialization begins with transfer of elements of the initial matrix $\mathbf{S}_0^{-1}$ through AXI-lite interface using keyhole writing procedure to register 1. As K elements are collected, they are written to BRAM dedicated for storing matrix $\mathbf{S}^{-1}$ in a parallel manner. This occurs in state *InitializeMatrix*, which lasts until all K columns/rows have been written to the BRAM. In other words, K^2 elements are sent to register 1 during this state. Finally, the target pixel **s** is uploaded to the FPGA core using register 2.

The next state is *WaitForStart*, where the controller waits for *Enable Core* signal corresponding to register 0. As soon as *Enable Core* signal is asserted from the processing system, the state changes to *WriteVector* and the core becomes ready to accept incoming pixel components by asserting the ready signal on AXI stream slave interface. The pixel components are streamed using DMA cores as in section 4.1.1. This completes the initialization process of the core.

AXI FIFO and AXI broadcaster

As shown in Fig. 4.9, the FPGA core has two AXI-stream slave interfaces. The first interface is connected to DMA via AXI broadcaster, while the other receives pixels delayed by AXI stream FIFO block. The delay is implemented according to Listing 4.1. As a single stream is coming from DMA core, it needs to be replicated into multiple outbound interfaces. For this purpose, the AXI4-Stream Broadcaster is used. The outbound interfaces are connected to the FPGA core and FIFO block. The broadcaster remaps *TDATA* ports of AXI-stream interface, while the handshake signals are implemented as follows:

- Broadcaster's slave interface becomes ready only if both master interfaces are receiving a ready signal.
- Broadcaster's master interface becomes valid only if slave interface receives a valid signal and the other master interface is ready.

FIFO block is generated using Vivado's FIFO generator with AXI stream interface type. Both master and slave interface have a common clock and a *TLAST* signal. The FIFO is implemented using BRAMs, with the depth corresponding to the delay k and number of components K. If the delay is set to $k = K$, the FIFO requires depth of at least K^2 elements. The read latency of the FIFO is 2 clock cycles, caused by inherent BRAM read latency and AXI-stream handshake.

BRAM for matrix storage

Two memory resources for storing the matrix $\mathbf{S}^{-1}$ and target pixel $\mathbf{s}$ are available during the synthesis process (chosen via VHDL architectures): BRAM and registers. For hyperspectral datasets with high numbers of spectral bands, it is recommended that dedicated BRAM blocks are used for higher performance and lower utilization of logic blocks. The design of matrix storage is similar to the design presented in section 2.5.4, with the difference that the matrix row can be written parallelly in one clock cycle, instead of writing each element sequentially. It should be noted that reading elements from BRAM blocks introduces latency of one clock cycle. Accordingly, controller for the FPGA core is designed.

4.2.4 Processing logic

The computation of Sherman-Morrison inverse matrix update is divided into three stages:

1. In the first stage, matrix-vector product $\mathbf{S}_{i-1}^{-1}\mathbf{x}_i$ is calculated.
2. In the second stage, outer vector product $\mathbf{S}_{i-1}^{-1}\mathbf{x}_i\mathbf{x}_i^T\mathbf{S}_{i-1}^{-1}$ is obtained. Simultaneously, $d = 1 + \mathbf{x}_i^T\mathbf{S}_{i-1}^{-1}\mathbf{x}_i$ is calculated. Then, inverse $p = \frac{1}{d}$ is calculated.
3. In the third stage, the updated matrix $\mathbf{S}_i^{-1}$ is obtained as $\mathbf{S}_{i-1}^{-1} - (\mathbf{S}_{i-1}^{-1}\mathbf{x}_i\mathbf{x}_i^T\mathbf{S}_{i-1}^{-1}) \cdot p$.

Due to apparent high data dependency between each stage in the calculation process, the stages are scheduled sequentially in hardware. Nevertheless, the FPGA core meets the objectives with optimized performance and resource utilization. In the following text, each stage is explained in detail.

Stage 1

The first stage of Sherman-Morrison inverse updating produces a vector $\mathbf{S}_{i-1}^{-1}\mathbf{x}_i$ using an array of dot product units, as in the upper part of Fig. 4.3. As stage 2 requires the result of stage 1 to commence the operation, they cannot be scheduled in parallel manner.

Stage 2

Figure 4.10 shows the stage 2 of the FPGA core. This stage has $\mathbf{S}_{i-1}^{-1}\mathbf{x}_i$ and $\mathbf{x}_i$ as data inputs, and the outputs are available after $K + delay$ clock cycles ($delay$ is pipeline latency).

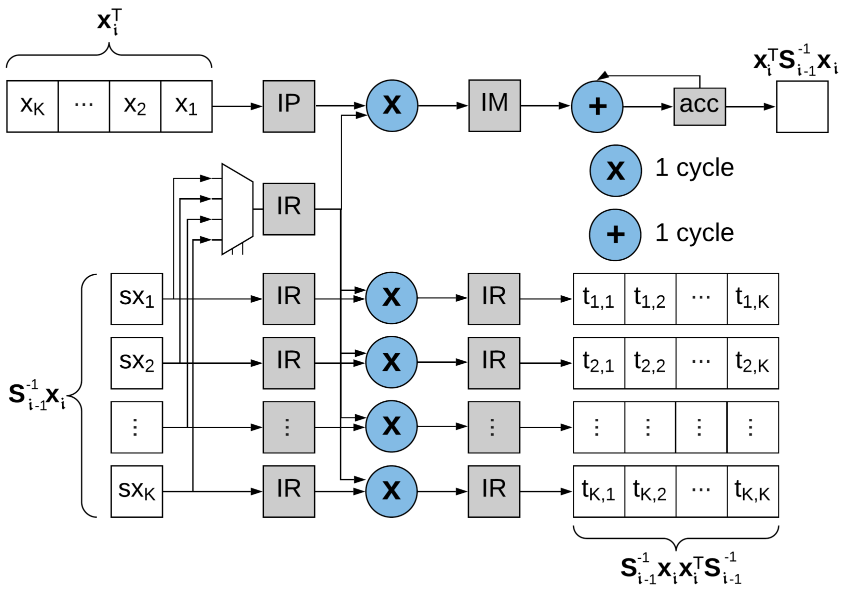

$\mathbf{x}_i^T$
x_K ··· x_2 x_1
IP
X
IM
+
acc
$\mathbf{x}_i^T\mathbf{S}_{i-1}^{-1}\mathbf{x}_i$
IR
X 1 cycle
+ 1 cycle
$\mathbf{S}_{i-1}^{-1}\mathbf{x}_i$
sx_1
sx_2
sx_K
$t_{1,1}$ $t_{1,2}$ ··· $t_{1,K}$
$t_{2,1}$ $t_{2,2}$ ··· $t_{2,K}$
$t_{K,1}$ $t_{K,2}$ ··· $t_{K,K}$
$\mathbf{S}_{i-1}^{-1}\mathbf{x}_i\mathbf{x}_i^T\mathbf{S}_{i-1}^{-1}$

Figure 4.10: RTL design of the Stage 2 of the FPGA core.

After $\mathbf{S}_{i-1}^{-1}\mathbf{x}_i\mathbf{x}_i^T\mathbf{S}_{i-1}^{-1}$ and $d = 1 + \mathbf{x}_i^T\mathbf{S}_{i-1}^{-1}\mathbf{x}_i$ are computed, inverse $p = \frac{1}{d}$ is obtained using an AXI divider. Therefore, the total execution time of this stage is $K + D + delay$ clock cycles, where D is divider latency.

Stage 3

The final stage in Sherman-Morrison updating procedure is performed as shown in Fig. 4.11. The output of the divider p is multiplied with each column in the matrix $\mathbf{S}_{i-1}^{-1}\mathbf{x}_i\mathbf{x}_i^T\mathbf{S}_{i-1}^{-1}$, using a multiplier array. Simultaneously, corresponding column of the matrix $\mathbf{S}_{i-1}^{-1}$ is subtracted from the product column.

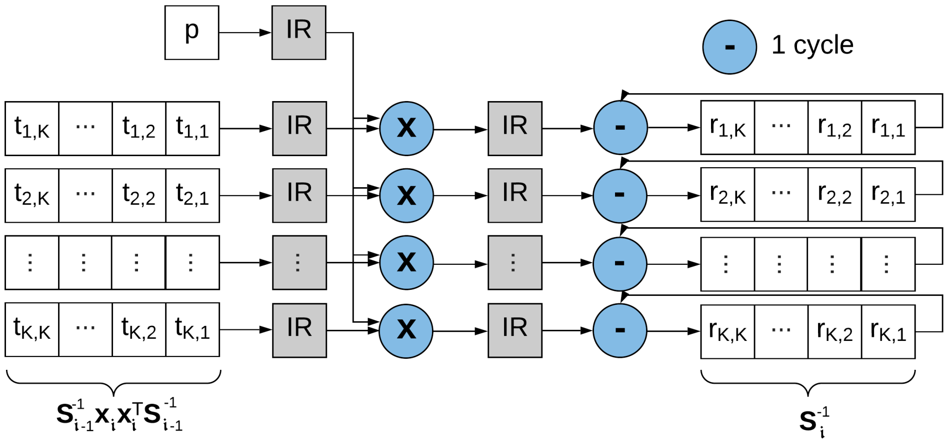


Figure 4.11: RTL design of the Stage 3 of the FPGA core.

Resource sharing and target detection algorithms

Due to the aforementioned data dependencies, in the corresponding stages of the processing algorithm, certain FPGA resources would be idling, such as multiplier array in stage 1 and dot product array in stage 2 and 3. This opens a possibility for sharing of these resources between stages of Sherman-Morrison updating and target detection algorithms. Sharing multi-cycle hardware blocks like the DSP blocks results in significant resource savings. However, operation scheduling becomes more complicated, as later explained in section 4.2.6.

Figure 4.12 shows the timeline of operations performed sequentially or simultaneously, depicted in parallel or perpendicular to the timeline axis, respectively. Memory and computation resources named identically are the same physical resources used at different time points. As such, one multiplier array is used in both stage 2 and 3 of Sherman-Morrison updating, while dot product array is used in stage 1, and during stage 2 and 3 to obtain the products required for target detection, $\mathbf{S}_{i-1}^{-1}\mathbf{s}$ and $\mathbf{S}_{i-1}^{-1}\mathbf{x}_{i-k}$, respectively. The remainder of target detection hardware is designed as in HW/SW implementation presented in section 4.1, with specific scheduling related to Sherman-Morrison updating hardware.

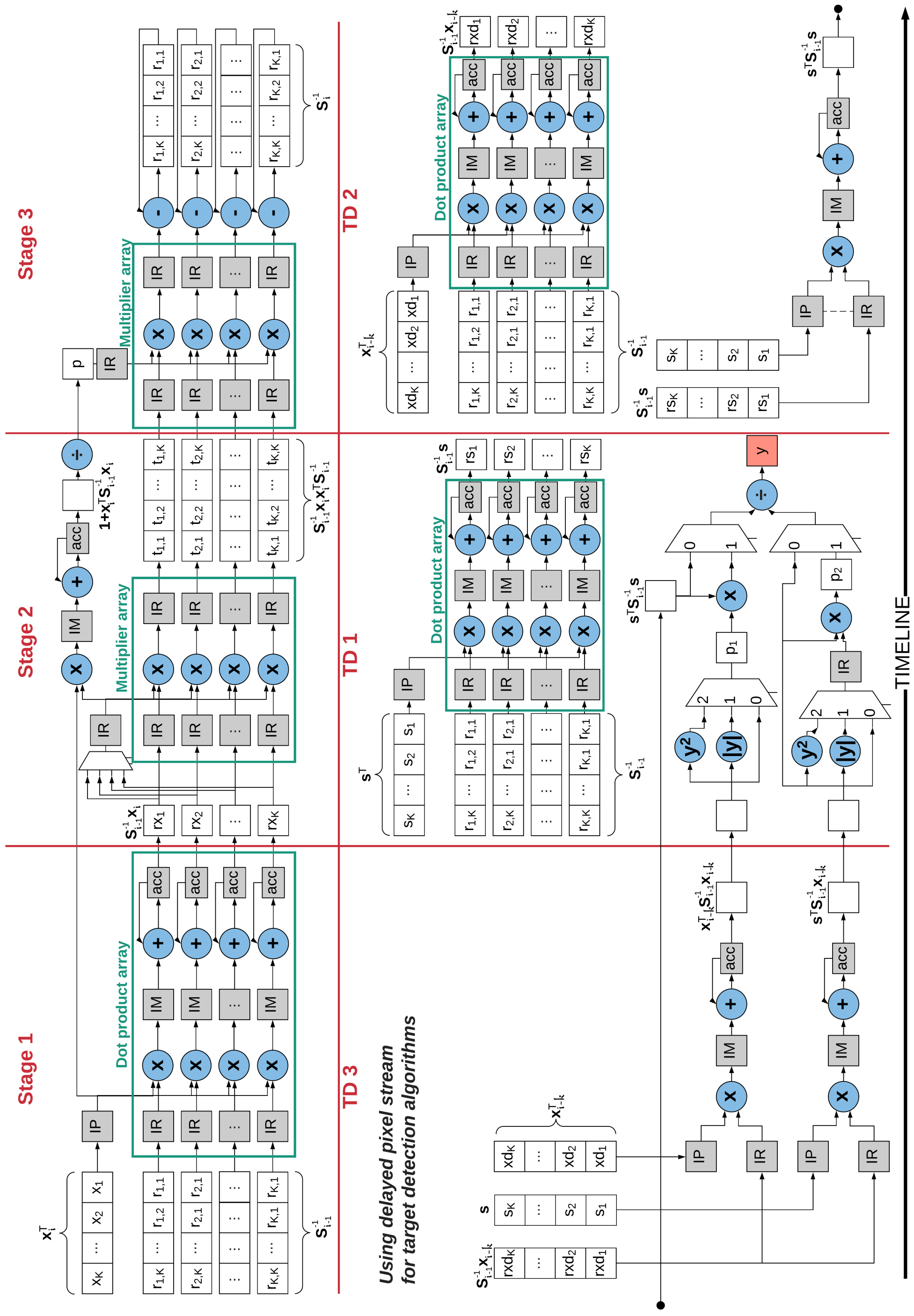


Figure 4.12: Timeline of operation stages in Sherman-Morrison updating and target detection.

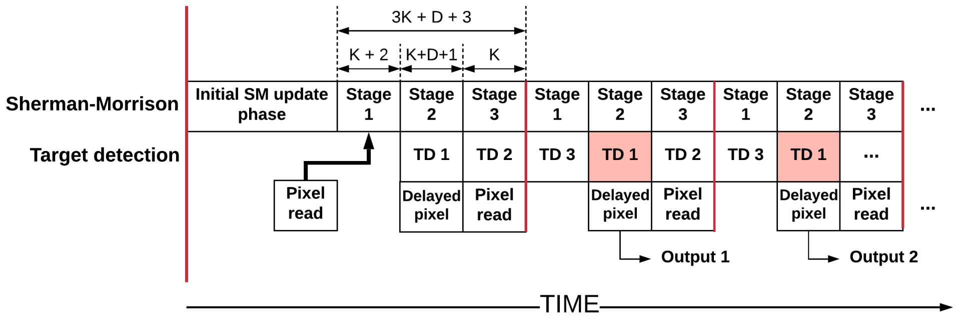


Figure 4.13: Timing diagram for Sherman-Morrison updating and target detection hardware.

Timing diagram in Fig. 4.13 shows initialization phase and processing of three pixels with Sherman-Morrison updating. After system initialization, pixels are read simultaneously during stage 3 of Sherman-Morrison updating. Additionally, the delayed pixel stream is read from FIFO in target detection step 1. The calculation of detection statistics overlaps with Sherman-Morrison updating. This leads to significant reduction of number of clock cycles needed for processing of each pixel. In this implementation, number of required clock cycles per pixel is $3K+D+3$, where K is the number of spectral bands and D is the divider latency. The comparison of run-time with other state-of-the-art implementations is presented in section 5.2.1.

4.2.5 Fixed-point considerations

As the Sherman-Morrison updating is an iterative process which highly influences target detection performance, special attention was given to the design of optimized fixed-point data types. Using MATLAB *Fixed-Point Converter*, proposed types are obtained which represent the generic inputs to the VHDL design. As such, the design can be adapted to various hyperspectral datasets with different degrees of precision.

As described in section 4.1.2, this design also involves slider values which correspond to the desired MSB bit position. In *Fixed-Point Converter*, we specify the desired word length, while the tool proposes fraction lengths based on simulation results. The simulation script is set up to use all the pixels in the scene, thus obtaining maximum and minimum simulation values for input, output and local variables of the function (in this case, Sherman-Morrison formula). Other fixed-point math attributes are set to:

- Rounding method: Floor
- Overflow action: Wrap
- Product mode: Full Precision
- Sum mode: Full Precision
- Signedness: Signed

which correspond to the actual hardware properties. The *Floor* rounding method is equivalent to two's complement truncation. As such, it provides the most efficient rounding implementation. Similarly, overflow is handled to reduce the amount of created logic using the standard method for wrapping with modulo arithmetic.

Proposed fixed point data types for HyMap scene are presented in Table 4.6 for 30, 35, 38, 42 and 52-bits word length. The word lengths are chosen according to the number of DSP blocks used to form a $P \cdot P$ multiplier. It can be observed that the integer part is not changing, while the fractional part is extended for longer words. The fractional part of $\mathbf{S}^{-1}\mathbf{x}\mathbf{x}^T\mathbf{S}^{-1}$ can be extended by $4-7$ bits without inferring additional DSP blocks to boost the detection performance. In this case, factor β is set to 1000. This is necessary measure to prevent dramatic degradation of target detection performance, which occurs for smaller fractional parts. Derived fixed-point data types are converted into VHDL generics as listed in Table 4.7.s

Table 4.6: Fixed-point data types for Sherman-Morrison implementation

Intermediate data	Word length-30	Word length-35	Word length-38	Word length-42	Word length-52
DSP blocks	4	**4**	5	**5**	**9**
$\mathbf{x}$	(16,1,15)	(16,1,15)	(16,1,15)	(16,1,15)	(16,1,15)
$\mathbf{S}^{-1}$	(30,11,19)	(35,11,24)	(38,11,27)	(42,11,31)	(52,11,41)
$\mathbf{S}^{-1}\mathbf{x}$	(30,11,19)	(35,11,24)	(38,11,27)	(42,11,31)	(52,13,41)
$\mathbf{S}^{-1}\mathbf{x}\mathbf{x}^T\mathbf{S}^{-1}$	(30,22,8)	(35,22,13)	(38,22,16)	(42,22,20)	(52,22,30)
d=$1+\mathbf{x}^T\mathbf{S}^{-1}\mathbf{x}$	(30,18,12)	(35,18,17)	(38,18,20)	(42,18,24)	(52,18,34)
p=$1/d$	(30,5,25)	(35,5,30)	(38,5,33)	(42,5,37)	(52,5,47)
$(\mathbf{S}^{-1}\mathbf{x}\mathbf{x}^T\mathbf{S}^{-1})\cdot p$	(30,11,19)	(35,11,24)	(38,11,27)	(42,11,31)	(52,11,41)

Table 4.7: Generics for full FPGA solution

Generic value	Description
NUM_BANDS	Number of bands in the hyperspectral image
PIXEL_DATA_WIDTH	Bit width of each component in a pixel
CORR_DATA_WIDTH	Bit width of elements of matrix $\mathbf{S}^{-1}$
OUT_DATA_WIDTH	Bit width of intermediate results
DP_DATA_SLIDER	Slider cutting the output of dot product array
MARR_DATA2_SLIDER	Slider cutting the output of multiplier array in stage 2
MARR_DATA3_SLIDER	Slider cutting the output of multiplier array in stage 3
XRX_DATA_SLIDER	Slider cutting the output of $\mathbf{x}^T\mathbf{S}^{-1}\mathbf{x}$ DP unit
C_S_AXI_DATA	Bit width of memory-mapped AXI data bus
C_S_AXI_ADDR	Bit width of memory-mapped AXI address bus

4.2.6 Controller design

The state machine for full FPGA implementation is shown in Fig. 4.14. First four states, namely, *Idle, Initialize, WaitForStart* and *WriteVector* are part of the initialization phase explained in section 4.2.3. Afterwards, the controller iterates through states *Step1Fetch* to *Step3* for each incoming pixel. As the last pixel component is received and processed, the controller goes to *Idle* state, where it waits for the new image.

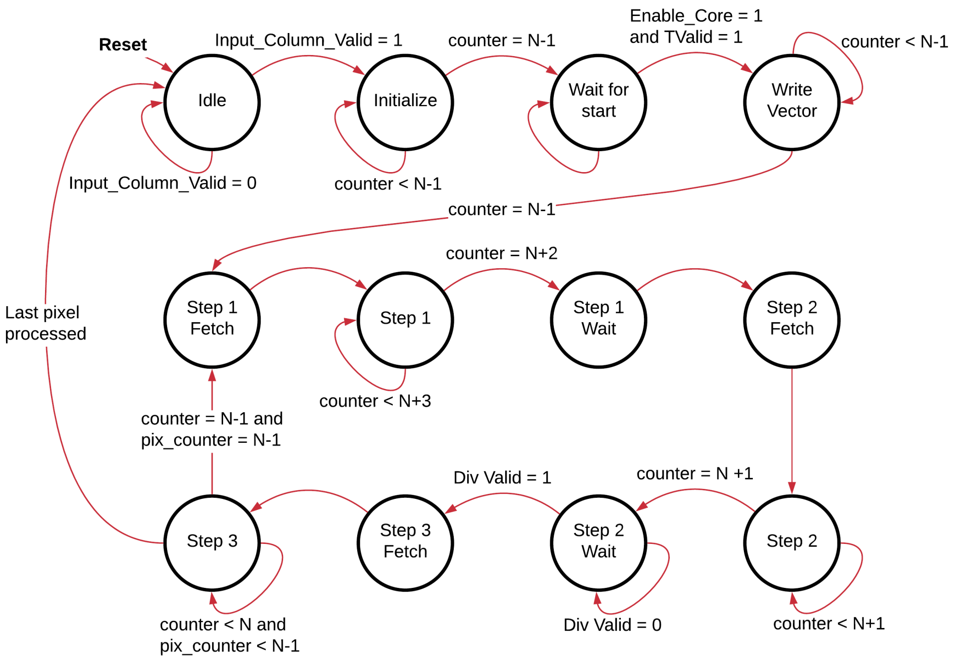


Figure 4.14: State machine for Sherman-Morrison updating and target detection hardware.

Each step in the state machine corresponds to the stages of Sherman-Morrison updating and target detection algorithms. *Fetch* and *Wait* states are designed to accommodate latency of BRAM blocks and other cores such as AXI divider. For instance, in state *Step2Wait*, the controller waits D clock cycles until the divider produces a valid result. Moreover, the controller is designed to control the inputs and outputs of processing blocks to enable sharing of resources.

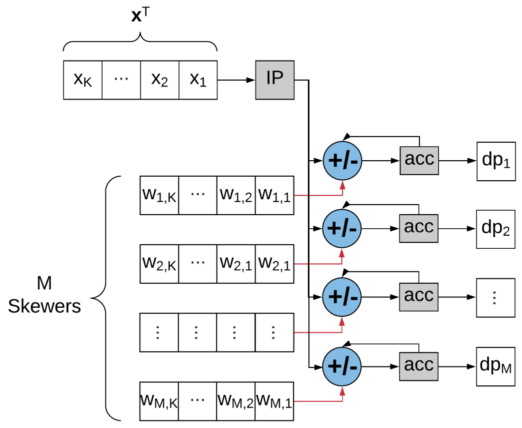
4.3 PPI algorithm implementation

The proposed PPI algorithm implementation for endmember extraction is inspired by FPGA implementation in [18]. This implementation uses parallelization strategy by skewers, where M PPI projections are calculated simultaneously for the same pixel. Parallelization strategy by skewers suits the AXI-stream interface, where pixels are streamed as they are captured by HSI or streamed from memory. The implementation proposed in this thesis follows Vivado HLS design flow, with adaptations of the communication architecture for Zynq-7000 platform. Along with architecture, C code for HLS with the corresponding directives is presented.

The most computationally expensive part of PPI algorithm are PPI projections obtained as:

$$PPI_{proj} = \sum_{n=1}^{K} \mathbf{x}_i^{(n)} \cdot \mathbf{skewer}_j^{(n)}, \tag{4.12}$$

which requires K multiplications. Previous publications [46] show that skewer values can be limited to $\{-1, 1\}$, which eliminates the need for multiplication. The PPI projection is then reduced to simple accumulator with adder/subtractor as shown in Fig. 4.15.



(a)

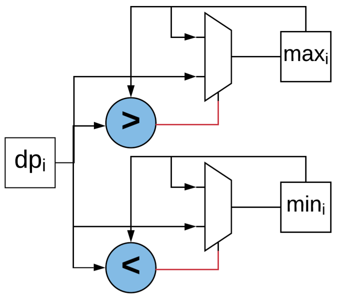
```
rng(1, skewers);

for(int i = 0; i < N_comp; i++) {
#pragma HLS PIPELINE

    for(int j = 0; j < M; j++) {
        #pragma HLS UNROLL

        if(i==0)
            vec_dp[j] = 0;

        if(skewers[j])
            vec_dp[j] -= vec_pix[i];
        else
            vec_dp[j] += vec_pix[i];

    }
    rng(0, skewers);
}
```

(b)

Figure 4.15: RTL design of PPI projections (a) and the corresponding C code in HLS (b).

The calculated dot products (dp_j or vec_dp_j) are then compared with the maximum and minimum projections obtained for each $\mathbf{skewer}_j$ (vec_max_j and vec_min_j), as shown in Fig. 4.16. Simultaneously, *endmembers* array is updated with the pixel index which has provided this extrema, which is not shown in the figure for simplicity.

The skewers are generated using the random number generation module presented in [18]. This simple generator module pseudo-random and uniformly distributed sequences using registers and XOR gates. Figure 4.17 shows the RTL design of the random generation module and corresponding C code for HLS. The module has four M-bit registers, of which two are used to store seeds and two are used in the generation process. The seeds



(a)

```
for(int t = 0; t < M; t++) {
#pragma HLS UNROLL

    if(vec_dp[t] > vec_max[t]) {
        vec_max[t] = vec_dp[t];
        endmembers_max[t] = pix_count;
    }

    if(vec_dp[t] < vec_min[t]) {
        vec_min[t] = vec_dp[t];
        endmembers_min[t] = pix_count;
    }
}
```

(b)

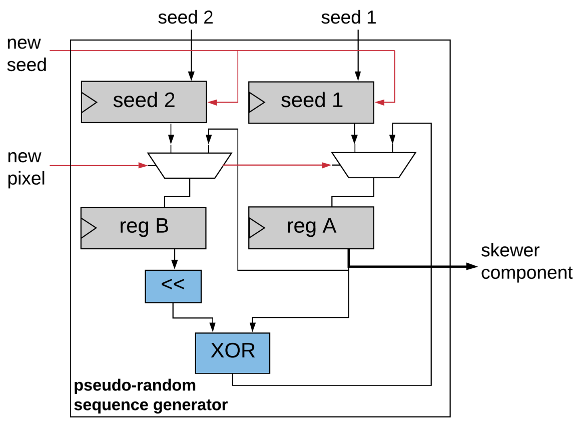
Figure 4.16: RTL design of a unit determining the maximum/minimum extrema (a) and the corresponding C code in HLS (b).

are initialized by the system each time that the PPI algorithm is executed for an image. The initialization is performed over AXI memory-mapped registers. At the beginning of PPI projections for a new incoming pixel, two seeds are copied into two generating registers. Afterwards, for each pixel component, content of register B is shifted and XOR-ed with content of register A, producing a component for each skewer.

The AXI-lite register file contains two seed registers, *new seed* write enable register, *endmembers* register and *done* register indicating that all pixels have been processed. Signal *Done* is asserted when endmembers are ready to be extracted and new seeds to be uploaded. The interfaces for mentioned ports are defined as follows:

- #pragma HLS INTERFACE s_axilite port=new_seed
- #pragma HLS INTERFACE s_axilite port=new_seed1
- #pragma HLS INTERFACE s_axilite port=new_seed2
- #pragma HLS INTERFACE s_axilite port=endmembers_mem
- #pragma HLS INTERFACE s_axilite port=done
- #pragma HLS INTERFACE axis port=vec_pix

PPI projections using the developed solution in HLS are repeated several times, depending on the number of skewers M that are synthesized on FPGA platform. Assuming that $M = 100$, the image has to be processed 100 times according to the experiments performed in [18]. In other words, each pixel is projected on 10^4 random skewers.



(a)

```
void rng(bool write_seed, bool skewer[M]){
#pragma HLS ARRAY_PARTITION variable=seed2 complete dim=1
#pragma HLS ARRAY_PARTITION variable=seed1 complete dim=1
#pragma HLS ARRAY_PARTITION variable=skewer complete dim=1

    static bool A[M] ;
#pragma HLS ARRAY_PARTITION variable=A complete dim=1
    static bool B[M];
#pragma HLS ARRAY_PARTITION variable=B complete dim=1
    bool shiftedB[M];
    int i;

    if(write_seed){
        INIT_LOOP: for (i = 0; i<M;i++){
            #pragma HLS UNROLL
                A[i]=seed1[i];
                B[i]=seed2[i];
            }
    }

    shiftedB[0] = B[M-1];
    SHIFT_LOOP: for (i =1; i<M;i++){
#pragma HLS UNROLL
        shiftedB[i] = B[i-1];
    }

    COPY_LOOP: for (i =0; i<M;i++){
#pragma HLS UNROLL
        B[i] = A[i];
    }

    XOR_LOOP: for (i =0; i<M;i++){
#pragma HLS UNROLL
            A[i] = A[i] ^ shiftedB[i];
    }

    OUTPUT_LOOP: for (i =0; i<M;i++){
    #pragma HLS UNROLL
            skewer[i]=A[i];
    }
}
```

(b)

Figure 4.17: RTL design of a pseudo-random sequence generating unit (a) and the corresponding C code in HLS (b).

Chapter 5

Results

This chapter presents results of HW/SW codesign of target detection algorithm for FPGA platform (in section 5.1) and full FPGA implementation using Sherman-Morrison method (in section 5.2). The results are divided into three subsections for each implementation, namely, hardware performance analysis, resource utilization and detection performance analysis. The results obtained during the specialization project for HW/SW implementation are briefly introduced, while additional analysis is performed for the proposed improvements in this thesis. Furthermore, the full FPGA implementation is discussed with the focus on aforementioned result sections.

In addition to that, hardware performance and resource utilization are reported for the PPI algorithm implementation using HLS in section 5.3.

5.1 HW/SW codesign implementation results

5.1.1 Performance analysis

The performance profile for HW/SW codesign implementation is obtained using post-synthesis results. The design and each sub-module have been individually analyzed to determine the maximum operating frequencies and the speed-up factor.

Due to the limitations of the used ZedBoard platform, the images used to determine the speed-up factor were pre-processed with dimensionality reduction technique - principal component analysis (PCA), explained in Appendix A. The effects of dimensionality reduction on target detection algorithms is explained in the specialization project [45]. Table 5.1 shows the measured execution time of ACE-R algorithm on Salinas dataset reduced to 16 spectral bands. The speed-up is evaluated in comparison with the software solution running on *ARM Cortex-A9* processor clocked with 666.67MHz. The software solution was developed during the specialization project [45]. It was tested on ZedBoard and optimized for performance using *-O3* compiler directive. It is assumed that the correlation matrix is uploaded once and reused for consequent algorithm runs to detect a target signature. The table shows full execution time containing correlation matrix calculation and its inversion in software, as well as the accelerator execution time, annotated as algorithm execution time. The FPGA accelerator is clocked with frequency of 100MHz, achieving a speed-up factor of 28.54 compared with ARM processor, and 4.34 compared with Intel i7-7500U clocked at 2.7GHz. It is important to note that the throughput of the design stays constant for any chosen algorithm implemented in the accelerator, thus not significantly affecting the execution time of the accelerator.

Table 5.1: Performance comparison for HW/SW codesign solution

Implementation	Full execution time	**Algorithm execution time**	Speed-up factor
SW model ARM (666.67MHz)	4s	0.59828s	**28.54**
SW model Intel i7-7500U	0.559s	0.091s	**4.34**
HW/SW design (100MHz)	3.29s	0.02096s	1

The highest achievable operating frequencies for *XC7Z020-CLG484-1* SoC are shown in Table 5.2. The generic values are set as presented in Table 4.4. These are not optimal values for the performance of the accelerator, due to the constrained input sizes of the DSP blocks on *ZYNQ PL*. Nevertheless, the design accepts generic values which will consequently affect the performance of the accelerator. Using the recommended input word sizes (25 x 18), it is possible to achieve higher operating frequencies. The dot product units (consisting of multipliers and adders) are pipelined with three pipeline stages, introducing the latency of three clock cycles. The addition of pipeline registers has a great impact on the overall design performance.

Table 5.2: Performance analysis of HW modules for HW/SW codesign solution

Module	Minimum period	Maximum frequency
PS-PL system (32 x 16)	$6.835ns$	$146.3MHz$
FPGA accelerator (32 x 16)	$5.745ns$	$174.06MHz$
Stage 1/2 (32 x 16)	$5.745ns$	$174.06MHz$
DP controller (16 bands)	$1.644ns$	$608.27MHz$
DP datapath (32 x 16)	$5.745ns$	$174.06MHz$
Master Output (16 packets)	$2.628ns$	$380.52MHz$
BRAM module (32)	$4.663ns$	$214.45MHz$

It is important to mention that both AXI DMA and Cube DMA are able to achieve the same throughput for BIP sequential transfer, as used in this design.

5.1.2 Resource utilization

Post-synthesis resource utilization is presented in Tables 5.3 and 5.4 for two different sets of word lengths of input data and intermediate data. All modules were synthesized out-of-context with default Vivado settings.

Table 5.3 shows resource utilization for FPGA accelerator synthesized for word length of pre-processed data from PS set to 32-bits and pixel component set to 16-bits. The intermediate data is truncated to 32-bits between stages of the accelerator. Finally, the divider has two inputs of 32-bits, generating a 64-bit result. This resource-wise non-optimal set of word length parameters provides high precision, however it requires substantially more resources to generate dot product units, such as 2 DSP blocks for one 16x32 multiplier.

Table 5.3: Resource utilization report 32x16, 16 bands, 32-bit intermediate data, HW/SW codesign solution

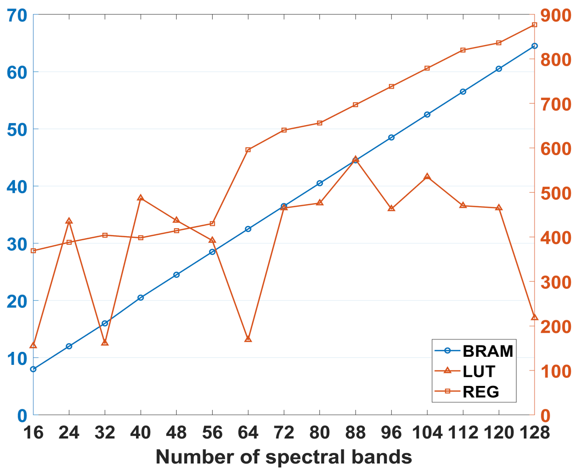
Module	Slice LUTs	Slice Registers	DSP blocks	BRAM tiles
PS-PL system	10105	18135	57	10.5
Input logic	2044	2859	0	10.5
AXI DMA	1882	2448	0	2.5
BRAM module (32-bit)	162	411	0	8
Processing logic	4265	9468	57	0
Stage 1	926	1161	34	0
DP controller	26	5	0	0
DP datapath	83	68	2	0
Stage 2	650	226	19	0
Stage 3 (w/o divider)	79	52	4	0
AXI Divider (64-bit)	2563	7345	0	0
Output logic	311	1114	0	0

Table 5.4: Resource utilization report 25x16, 16 bands, 25-bit intermediate data, HW/SW codesign solution

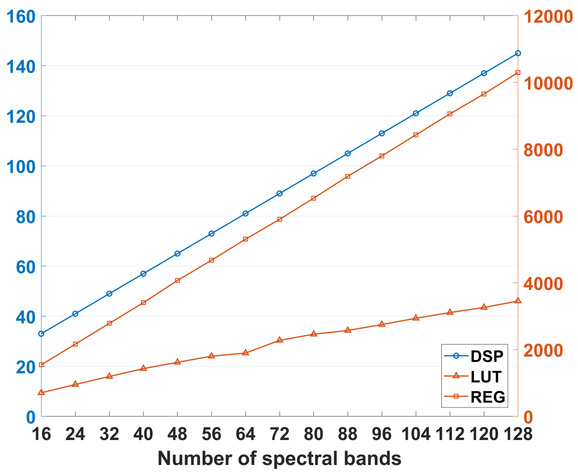
Module	Slice LUTs	Slice Registers	DSP blocks	BRAM tiles
PS-PL system	7889	12348	37	10.5
Input logic	2040	2817	0	10.5
AXI DMA	1882	2448	0	2.5
BRAM module (25-bit)	158	369	0	8
Processing logic	2105	5275	37	0
Stage 1	33	5	17	0
DP controller	8	5	0	0
DP datapath	25	0	1	0
Stage 2	314	99	17	0
Stage 3 (w/o divider)	27	27	2	0
AXI Divider (50-bit)	1657	4608	0	0
Output logic	259	438	0	0

The word lengths of the BRAM elements and input samples in Table 5.4 belong to the range of recommended input word lengths for DSP blocks on *ZYNQ* PL. It can be observed that the DP datapath uses exactly one block for this setup and no additional programmable logic is used to create a multiplier or an adder.

Figure 5.1 shows resource utilization as a function of number of spectral bands for BRAM module in 5.1(a) and processing logic in 5.1(b). Resource utilization increases as expected with the number of spectral bands. Most interestingly, BRAM module uses the least LUTs for number of bands which are power of two.



(a) BRAM module



(b) Processing logic (w/o divider)

Figure 5.1: Resource utilization as a function of different number of spectral bands for BRAM module and processing logic, respectively.

5.1.3 Detection performance analysis

In order to verify the operation of the implemented accelerator, it has been tested using aforementioned scenes and chosen spectral signatures as targets. Testing has been performed using simulation tools, as well as the ZedBoard prototyping platform.

The analysis starts with determining the range of the variables to be used by the processing logic. Using MATLAB NumericTypeScope, it is feasible to adequately determine the dynamic range and corresponding bit widths of variables. Since the accelerator uses the precalculated inverse of the correlation matrix $\mathbf{R}^{-1}$, vector $\mathbf{s}^T\mathbf{R}^{-1}$ and parameter $\mathbf{s}^T\mathbf{R}^{-1}\mathbf{s}$, they are scaled in SW prior to being uploaded to the BRAM. As all of these parameters are used during the whole execution of the algorithm for each pixel, their precision strongly influences the detection performance. Therefore, no data values should be outside range or below chosen precision. Additionally, C program developed for Zynq PS is used to adaptively scale the parameters and control the operation of the target detection accelerator.

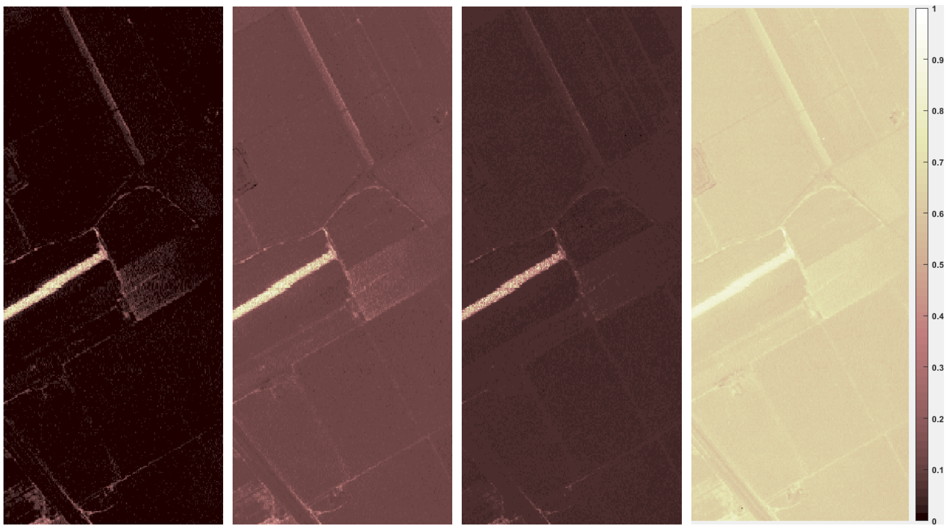
Salinas scene

Detection performance test on the implemented accelerator was performed using Salinas scene and *Lettuce romaine 4th week* endmember as a designated target (ground truth shown in Fig. 5.3(a)). The full dimensionality image was used to estimate the correlation matrix. Furthermore, pixel component bit width was set to 14 bits, while BRAM elements and intermediate data are of 30 bits. Minimal BRAM element bit width is 30 bits, thus chosen in order to accommodate high dynamic range of the inverse correlation matrix values. The obtained probability images using the FPGA accelerator are shown in Fig. 5.2. Additionally, table 5.5 shows the detection performance scores of the accelerator fixed-point and software floating-point solutions.

It can be observed that the accelerator fixed-point solution has certainly not degraded the detection performance of the algorithm. In cases of ACE-R and ASMF(-2) algorithms, visibility scores are slightly lower, but despite that, MCC scores are not degraded but higher. When comparing the implemented algorithms, it is clear that the proposed ACE-R method has the highest visibility, which can be perceived from Fig. 5.2. In Fig. 5.3(b), ROC curve obtained by FPGA implementation of algorithms for the detection of *Lettuce Romaine 4th week* target signature. Furthermore, the area under ROC curve (AUC) is given in table 5.5. The higher the AUC, the better the detection results.

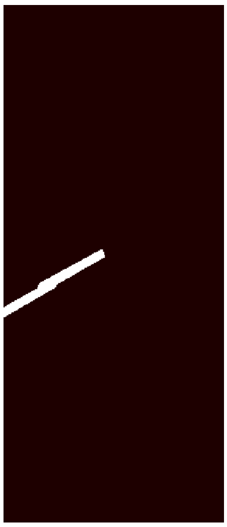
Table 5.5: Comparison of detection performance scores for fixed- and floating-point solutions; Salinas scene using HW/SW codesign solution

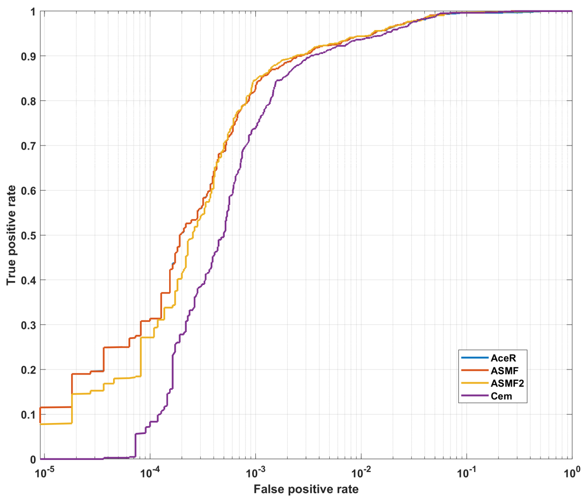
Algorithm	MCC score		Visibility		AUC	
	Fixed-point	Floating-point	Fixed-point	Floating-point	Fixed-point	Floating-point
ACE-R	**0.8639**	0.8465	0.5469	**0.5708**	**0.9965**	0.9963
ASMF	**0.8639**	0.8465	0.4844	**0.4928**	**0.9972**	0.9970
ASMF-2	**0.8690**	0.8500	0.3678	**0.4447**	0.9968	**0.9971**
CEM	0.8403	0.8403	0.1825	0.1825	0.9967	0.9967



(a) ACE-R (b) ASMF (c) ASMF-2 (d) CEM

Figure 5.2: Detection results (probability images) for *Lettuce romaine 4th week* target signature from Salinas scene obtained using the implemented accelerator.





(a) *Lettuce romaine 4th week* ground truth

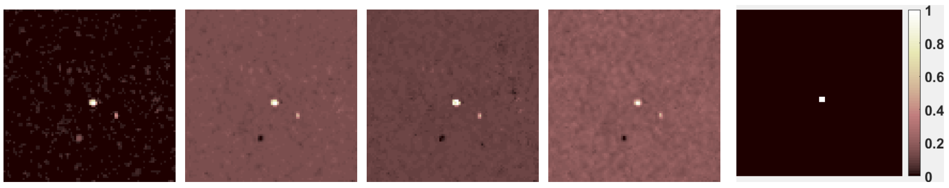
(b) ROC curve obtained by different algorithms for the detection of *Lettuce Romaine 4th week* target signature.

Figure 5.3

HyMap Cooke City scene

Another scene used to verify the operation of the implementation is HyMap Cooke City scene. The scene has been cropped off for testing purposes to 90 by 90 pixel image area containing all known target pixels (starting from locations $x = 90$ and $y = 458$ of the full image). Designated targets are *F1* and *F4*, with corresponding ground truth shown in Fig. 5.4(e) and 5.5(e), respectively. As all spectral bands of the image were used, BRAM elements and intermediate data word lengths are of 30 bits, while pixel component word length is 15 bits, accommodating all components.

The results from the accelerator are shown in Fig. 5.4 and 5.5 for target signatures *F1* and *F4*, respectively. Furthermore, tables 5.6 and 5.7 show the detection performance scores for all algorithm implementations in fixed-point and floating-point precision. Most importantly, the FPGA accelerator provides comparable results in shorter time than its floating-point counterpart, as confirmed by testing. Differences in resulting detection performance scores are therefore usually negligible. Additionally, ROC curves are shown in Fig. 5.6. As expected, implemented ACE-R and ASMF-2 algorithms demonstrate the best detection performance, while conventional CEM method shows inferiority by offering substantially higher false alarm rates for the same or lower true positive rates.

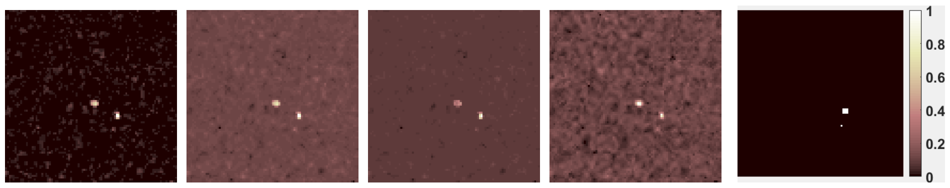


(a) ACE-R (b) ASMF (c) ASMF-2 (d) CEM (e) *F1* ground truth

Figure 5.4: Detection results (probability images) for *F1* target signature from HyMap Cooke City scene obtained using the implemented accelerator.

Table 5.6: Comparison of detection performance scores for fixed- and floating-point solutions; HyMap scene, target signature *F1*, using HW/SW codesign solution

Algorithm	MCC score		Visibility		AUC	
	Fixed-point	Floating-point	Fixed-point	Floating-point	Fixed-point	Floating-point
ACE-R	0.9486	0.9486	**0.7584**	0.7426	0.9999	0.9999
ASMF	0.9486	0.9486	**0.6536**	0.6418	0.9999	0.9999
ASMF-2	0.9486	0.9486	**0.7765**	0.7609	0.9999	0.9999
CEM	0.8430	0.8430	0.4778	0.4778	0.9998	0.9998

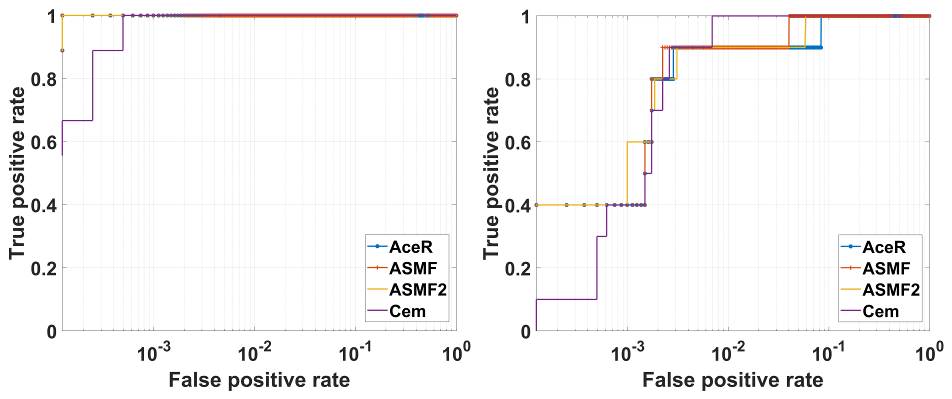


(a) ACE-R (b) ASMF (c) ASMF-2 (d) CEM (e) *F4* ground truth

Figure 5.5: Detection results (probability images) for *F4* target signature from HyMap Cooke City scene obtained using the implemented accelerator.

Table 5.7: Comparison of detection performance scores for fixed- and floating-point solutions; HyMap scene, target signature *F4*, using HW/SW codesign solution

Algorithm	MCC score		Visibility		AUC	
	Fixed-point	Floating-point	Fixed-point	Floating-point	Fixed-point	Floating-point
ACE-R	**0.5802**	0.5684	**0.4444**	0.4311	0.9959	**0.9966**
ASMF	0.5802	0.5802	**0.4067**	0.3934	0.9974	**0.9977**
ASMF-2	0.5802	**0.6322**	0.3623	**0.3643**	**0.9966**	0.9964
CEM	0.5370	0.5370	0.3122	0.3122	0.9983	0.9983



(a) F1 (b) F4

Figure 5.6: ROC curves obtained by different algorithms for the detection of *F1* and *F4* target signatures.

5.2 Full FPGA implementation results

5.2.1 Performance analysis

The performance of the full FPGA implementation is analyzed using post-synthesis results. The design performs at 123MHz on ZedBoard's XC7Z020-CLG484-1 SoC.

The design requires $3K+D+3$ clock cycles to process one pixel with K spectral bands, which allows real-time operation with respect to the HSI imager planned for usage. It is assumed that the imager ideally acquires 32 frames per second (FPS), with 1216 pixels containing 100 spectral bands after binning. Therefore, the FPGA core clocked with 100MHz has 31.25ms to process the incoming frame. As one pixel now requires 403 cycles (with D set to 100), it is estimated that one frame can be processed in 4.9ms. Table 5.8 presents the data rates for three hyperspectral sensors, namely HSI, HyMap and AVIRIS. For all considered instruments, the FPGA implementation is able to perform in real-time. Incoming data rate from sensor is compared with the processing data rate of the FPGA implementation for the provided parameters. The processing data rate is theoretically limited to 66.67MB/s for FPGA clocked with 100MHz and 16-bit components. It is important to note that the latency of AXI divider varies with precision from D=68 cycles for 32-bit inputs to D=100 for 48-bit precision.

Table 5.8: Sensors and FPGA processing data rate for full FPGA solution

Sensor	FPS	Spectral bands	Pixels per frame	Data Resolution	Incoming throughput	FPGA core throughput
HSI	32	100	1216	16 bits	7.78MB/s	49.63MB/s
HyMap	16	126	512	16 bits	1.97MB/s	49.96MB/s
AVIRIS	100	224	512	16 bits	22.93MB/s	57.80MB/s

Table 5.9 shows the number of clock cycles required by SBS-CEM, DPBS-CEM and implementation proposed in this thesis to process the full HyMap image of 224000 pixels. DPBS-CEM requires the lowest number of clock cycles; however, it is not suitable for all FPGA platforms due to very high resource utilization caused by the deep pipelined structure.

Table 5.9: Comparison of data processing speed for the FPGA implementations

Implementation	Number of clock cycles	Speed-up factor
SBS-CEM [32]	229,607,996	2.28
OUR IMP	**100,774,324**	1
DPBS-CEM [33]	31,360,557	0.31

5.2.2 Resource utilization

Post-synthesis resource utilization presented in Table 5.10 corresponds to full FPGA implementation synthesized for 32 spectral bands with 32-bit intermediate data. The synthesis has been performed for ZedBoard with default Vivado settings. As the ZedBoard has low amount of DSP blocks (220), the design has been synthesized for 32 bands, as it directly affects the number of required DSP blocks. Similarly, the intermediate data precision has impact on resource utilization, especially DSP blocks. The estimated power for FPGA implementation is $1.418W$. The processing system consumes additional $1.533W$. The largest fractions of power consumption belongs to DSP and BRAM blocks used in the design.

Table 5.10: Resource utilization report, 32 bands, 32-bit intermediate data, full FPGA solution

Module	Slice LUTs	Slice Registers	DSP blocks	BRAM tiles
PS-PL system	16018	20074	198	35
Input logic	2945	4080	0	2
AXI DMA	1669	2140	0	2
Initialization module	1276	1940	0	0
Processing logic	9267	11048	198	32
Dot Product Array	2983	2950	64	0
DP controller	7	6	0	0
DP datapath	43	69	2	0
Multiplier Array	1567	580	128	0
Mult. controller	34	5	0	0
Mult. datapath	59	18	4	0
AXI Divider	2439	7133	0	0
Matrix storage	117	0	0	14.5/17.5
Controller	275	39	0	0
Output logic	79	297	0	0

To compare with other implementations, such as DPBS-CEM and SBS-CEM, the core has been synthesized for a larger FPGA device, namely, Zynq-7035 with 900 DSP blocks. The results for HyMap image with 126 bands are shown in Table 5.11.

Table 5.11: Comparison of used resources for SBS-CEM, DPBS-CEM and our implementation

Implementation	Slice LUTs	Slice Registers	DSP blocks	BRAM tiles
SBS-CEM	21730	28245	265	120
DPBS-CEM	111073	217958	1396	379
OUR IMP 32-bit	36233	28719	762	135
OUR IMP 35-bit	45900	31035	762	136

5.2.3 Detection performance analysis

The implemented full FPGA solution with Sherman-Morrison inverse updating has been tested using Salinas and HyMap scene and chosen spectral signatures as targets. Testing has been performed using MATLAB fixed-point tools, as well as the ZedBoard prototyping platform. The detection results for different fixed-point data types are reported.

Salinas scene

Detection performance test on the implemented FPGA solution was performed using Salinas scene and *Lettuce romaine 4th week* endmember as a designated target. Figure 5.7 shows real-time operation on the full Salinas scene with 111104 pixels reduced to 20 spectral bands using PCA dimensionality reduction method. The algorithm chosen in the figure is adapted ACE-R. The target is clearly detected with high detection accuracy for 40-bits fixed-point word length, while results for other fixed-point types are reported in Table 5.12.

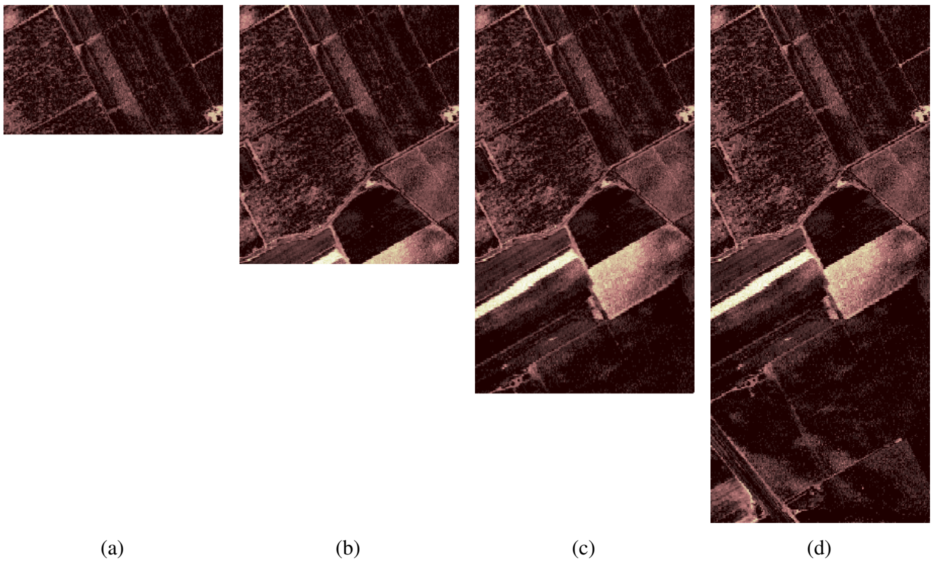


Figure 5.7: Real-time detection results using Salinas scene and *Lettuce romaine 4th week* as the target signature.

The results presented in Table 5.12 show that the detection performance proportionally increases with word length. For 32-bit intermediate data, the performance is significantly degraded due to high amount of occurring underflows during Sherman-Morrison updating. High dynamic range of data requires longer words for processing, however two solutions can be implemented to relieve the problem. First, change of fixed-point data type during execution would allow growth of fractional part, and therefore higher detection accuracy. Second, it is possible to use a subset of pixels for Sherman-Morrison method, thus additionally limiting the dynamic range of intermediate data. Publications such as [17] claim that if the subset is appropriately selected, the detection performance will remain unchanged. Experiments conducted during this thesis have shown that when using 10% randomly selected pixels from image, the detection performance changes negligibly.

Table 5.12: Comparison of detection performance scores using different fixed-point types

Algorithm	MCC score			
	32	36	40	Global
ACE-R	0.0154	0.6167	0.8284	0.8191
ASMF	0.0200	0.6167	0.8284	0.8191
CEM	0.2237	0.6312	0.6507	0.6811
Algorithm	Visibility			
	32	36	40	Global
ACE-R	$4.14 \cdot 10^{-5}$	0.7114	0.7052	0.7078
ASMF	$5.72 \cdot 10^{-5}$	0.3614	0.3619	0.4569
CEM	$11.2 \cdot 10^{-5}$	0.0487	0.0356	0.0268
Algorithm	AUC			
	32	36	40	Global
ACE-R	0.4524	0.9367	0.9811	0.9795
ASMF	0.4586	0.9447	0.9838	0.9825
CEM	0.8967	0.9833	0.9881	0.9785

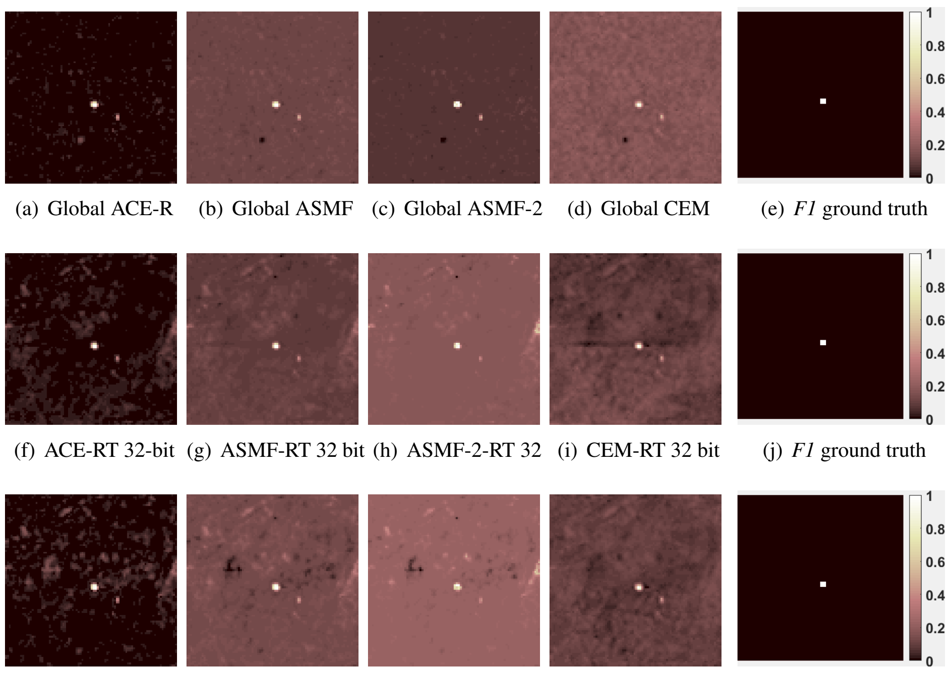
The improvement when using two different fixed-point types is presented in Table 5.13 with ACE-R as example. Firstly, types described in section 4.2.5 are used (annotated in table as FP1), with extended fractional part for $\mathbf{S}^{-1}\mathbf{x}\mathbf{x}^T\mathbf{S}^{-1}$. After K pixels are processed, the hardware switches to new fixed-point types for intermediate data (annotated in table as FP2). New types retain the same word length, but with increased fractional part. This modification would not significantly increase resource utilization on FPGA.

Table 5.13: Comparison of detection performance scores using different fixed-point types

Algorithm	MCC score		
	32	36	40
ACE-R FP1	0.0154	0.6167	0.8284
ACE-R FP2	**0.6063**	**0.8286**	**0.8328**
Algorithm	Visibility		
	32	36	40
ACE-R FP1	$4.14 \cdot 10^{-5}$	0.7114	0.7052
ACE-R FP2	**0.7083**	0.7049	**0.7285**
Algorithm	AUC		
	32	36	40
ACE-R FP1	0.4524	0.9367	0.9811
ACE-R FP2	**0.9832**	**0.9811**	**0.9849**

HyMap Cooke City scene

The solution has been tested using target signatures *F1* and *F4* from HyMap Cooke City scene dataset. As in section 5.1.3, the scene has been cropped off for testing purposes to 90 by 90 pixel image area containing all implanted target pixels. The results from the implementation are shown in Fig. 5.8 and 5.9 for target signatures *F1* and *F4*, respectively. In the figures, results of global algorithms, as well as results of implementations using 32 and 42 bits are shown. Additionally, tables 5.14 and 5.15 presents MCC and visibility scores for different bit widths of Sherman-Morrison implementation intermediate data when detecting signatures *F1* and *F4*, respectively.



(a) Global ACE-R (b) Global ASMF (c) Global ASMF-2 (d) Global CEM (e) *F1* ground truth

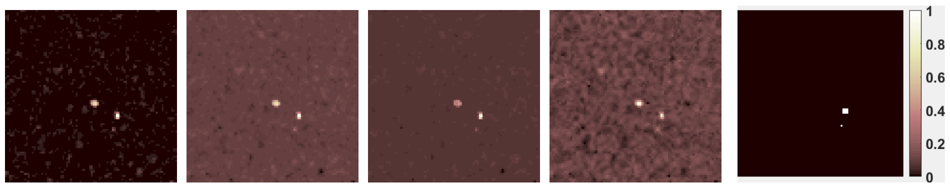
(f) ACE-RT 32-bit (g) ASMF-RT 32 bit (h) ASMF-2-RT 32 (i) CEM-RT 32 bit (j) *F1* ground truth

(k) ACE-RT 42 bit (l) ASMF-RT 42 bit (m) ASMF-2-RT 42 (n) CEM-RT 42 bit (o) *F1* ground truth

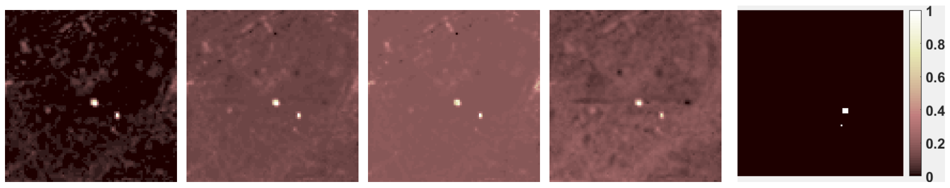
Figure 5.8: Detection results (probability images) for *F1* target signature from HyMap Cooke City scene obtained using the full FPGA implementation with Sherman-Morrison updating.

Table 5.14: Comparison of detection performance scores using different fixed-point types

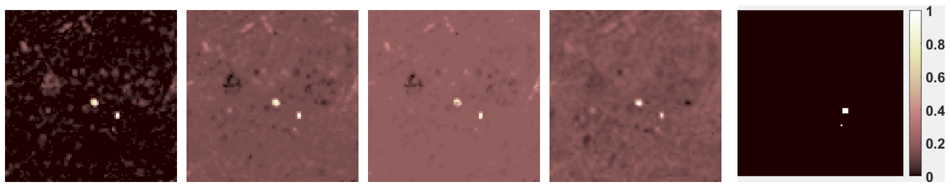
Algorithm	MCC score					Visibility				
	30	35	38	42	Global	30	35	38	42	Global
ACE-R	0.94	1	1	1	0.95	0.64	0.76	0.79	0.81	0.76
ASMF	0.94	1	1	1	0.95	0.61	0.68	0.71	0.71	0.67
ASMF-2	0.88	0.88	0.90	0.95	0.95	0.61	0.57	0.55	0.53	0.81
CEM	0.89	0.90	0.90	0.90	0.84	0.53	0.53	0.53	0.53	0.49



(a) Global ACE-R (b) Global ASMF (c) Global ASMF-2 (d) Global CEM (e) *F4* ground truth



(f) ACE-RT 32 bit (g) ASMF-RT 32 bit (h) ASMF-2-RT 32 bit (i) CEM-RT 32 bit (j) *F4* ground truth



(k) ACE-RT 42 bit (l) ASMF-RT 42 bit (m) ASMF-2-RT 42 bit (n) CEM-RT 42 bit (o) *F4* ground truth

Figure 5.9: Detection results (probability images) for *F4* target signature from HyMap Cooke City scene obtained using the full FPGA implementation with Sherman-Morrison updating.

Table 5.15: Comparison of detection performance scores using different fixed-point types

Algorithm	MCC score					Visibility				
	30	35	38	42	Global	30	35	38	42	Global
ACE-R	0.35	0.45	0.45	0.55	0.57	0.32	0.39	0.43	0.45	0.43
ASMF	0.35	0.45	0.45	0.55	0.58	0.30	0.34	0.38	0.39	0.39
ASMF-2	0.32	0.47	0.63	0.56	0.63	0.26	0.31	0.36	0.34	0.36
CEM	0.36	0.48	0.48	0.47	0.54	0.27	0.24	0.24	0.24	0.31

From the presented scores it can be concluded that detectors maintain high detection performance even with the short word length on small image sizes. In majority of cases, the resulting scores are very close to those obtained using global detectors. Similarly, as with Salinas scene, changing the fixed-point types in order to increase the fractional part, proportionally boosts the detection performance. This effect is shown in Fig. 5.10.

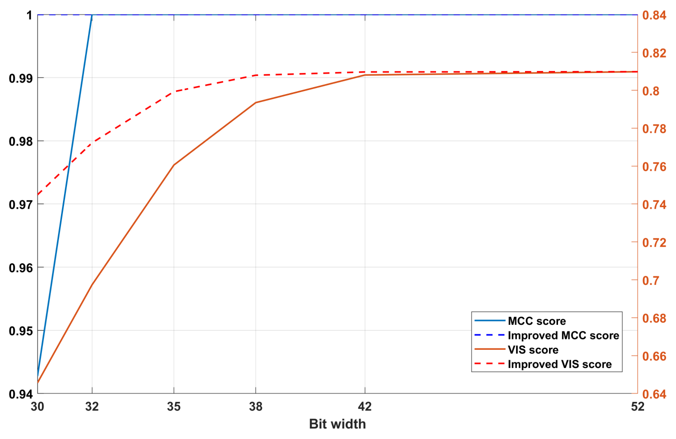


Figure 5.10: Comparison of detection performance scores using different fixed-point types for adapted ACE-R detector.

5.3 PPI algorithm implementation results

The performance profile and resource utilization for PPI algorithm implementation are obtained using post-synthesis results for ZedBoard platform. The results presented in Table 5.16 are synthesis results for an image with 224 spectral bands, where each pixel component is represented with 16 bits. The number of skewers varies from 20 to 120, with a constant increase of slice LUTs and slice registers on FPGA fabric.

Table 5.16: Resource utilization report for PPI algorithm implementation

Number of skewers	Slice LUTs	Slice Registers	BRAM tiles
20	2173	3130	6
40	4120	5963	6
60	6029	8827	6
80	7879	11667	6
100	9817	14509	6
120	11677	17346	6

The maximum achievable frequency is 196.6MHz on ZedBoard platform. For a pixel with 224 spectral components, PPI projections take 226 clock cycles, where 2 additional cycles are required for control logic and Min/Max unit to determine the extrema set.

It should be noted that this implementation was developed using Vivado HLS. The design achieves comparable performance to other FPGA implementations developed using VHDL, such as [18]. A very important aspect is the shortened development time which can be achieved using the automated approach with HLS compared with traditional HDL coding. Testing of PPI algorithm has been extensively performed and reported in publications such as [47, 18, 46].

Chapter 6

Conclusion

This thesis has investigated target detection algorithms and feasibility of real-time FPGA implementation of detectors. Two solutions were proposed, namely, hardware/software codesign implementation of target detection algorithms and full FPGA solution using Sherman-Morrison method for target detection algorithms in hyperspectral imagery. Moreover, PPI endmember extraction algorithm has been designed using Vivado HLS. This chapter draws conclusions from the implementation and results, and presents guidelines for further development.

6.1 Target detection algorithms

A selection of state-of-the-art target detection algorithms has been tested using 5 hyperspectral datasets. The datasets are not algorithmically created, but rather real scenes captured using different hyperspectral imagers. The evaluation has been performed using using MCC score, visibility metric and ROC curves.

The overall robustness of algorithms with respect to visibility corresponds with the reported findings in other publications [9, 11], especially for ACE algorithm. The ACE-R adaptation presented in this thesis maintains or even improves the detection performance of ACE algorithm, with the feasibility for real-time operation. ASMF algorithm, with increased complexity over other algorithms, shows promising results in terms of both MCC and visibility scores.

6.2 FPGA implementations

It is important to maintain the simplicity of FPGA solutions for on-board satellite systems, while satisfying all critical timing constraints that the system might have. Fast target detection is the crucial part of HYPSO mission, especially in case of satellite cooperation with aerial, surface, and underwater vehicles. With this in mind, two solutions proposed in this thesis were designed using VHDL.

HW/SW codesign implementation is the FPGA accelerator for ACE-R, ASMF and CEM detection algorithms, with static background estimation method. On ZedBoard platform with Zynq-7000 SoC, the implementation is able to achieve speed-up factor of 28.54 over its software counterpart running on ARM processor, without degradation of detection performance.

Full FPGA solution using Sherman-Morrison method for target detection algorithms does not require any computation in software. This implementation provides real-time operation for three analyzed hyperspectral imagers, including the HSI which will be part of HYPSO satellite payload.

PPI algorithm implementation was developed using Vivado HLS. This design showed how C-code can be rapidly converted into corresponding VHDL or Verilog code at RTL level. The synthesized design achieves comparable performance to other state-of-the-art implementations designed using traditional HDL design flow.

6.3 Future work

As with almost all hardware designs, there is still more room for potential paralellization, pipelining and performance improvements. Further testing on more hyperspectral images and with their corresponding spectral signatures is always necessary to fully characterize adaptability and usability of the developed systems. As the on-board processing pipeline is being developed for a small, energy-limited satellite, additional power consumption estimation should be performed. The end goal is integration of the developed FPGA implementations with other modules in the hyperspectral processing pipeline, for example, dimensionality reduction techniques.

Bibliography

[1] Mariusz E. Grøtte, Roger Birkeland, Joao F. Fortuna, Julian Veisdal, Milica Orlandic, Evelyn Honore-Livermore, Gara Quintana-Diaz, Harald Martens, J. Tommy Gravdahl, Fred Sigernes, Jan Otto Reberg, Geir Johnsen, Kanna Rajan, and Tor A. Johansen. Hyperspectral imaging small satellite in multi-agent marine observation system. *Unpublished-Internal document*, 2018.

[2] Norsk Romsenter. Norway and Earth Observation, 2019.

[3] Roland Trautner. ESA's roadmap for next generation payload data processors. *Proc. of DASIA Conference*, 2011.

[4] Jose M. Bioucas-Dias, Antonio Plaza, Gustavo Camps-Valls, Paul Scheunders, Nasser Nasrabadi, and Jocelyn Chanussot. Hyperspectral remote sensing data analysis and future challenges. *IEEE Geoscience and Remote Sensing Magazine*, 1(2):6–36, 2013.

[5] Karine Avagian, Milica Orlandic, and Tor Arne Johansen. An FPGA-oriented HW/SW Codesign of Lucy-Richardson Deconvolution Algorithm for Hyperspectral Images. *8th Mediterranean Conference on Embedded Computing – MECO*, 2019.

[6] Sivert Bakken, Milica Orlandic, and Tor Arne Johansen. The effect of dimensionality reduction on signature-based target detection for hyperspectral imaging. *CubeSats and SmallSats for Remote Sensing III*, 2019.

[7] Milica Orlandic, Johan Fjeldtvedt, and Tor Arne Johansen. A Parallel FPGA Implementation of the CCSDS-123 Compression Algorithm. *Remote Sensing*, 11(6):673, 2019.

[8] Johan Fjeldtvedt, Milica Orlandic, and Tor Arne Johansen. An Efficient Real-Time FPGA Implementation of the CCSDS-123 Compression Standard for Hyperspectral Images. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 11(10):3841–3852, 2018.

[9] Dimitris Manolakis. Is there a best hyperspectral detection algorithm? *SPIE Newsroom*, 2009.

[10] Shunlin Liang and James J. Butler. *Comprehensive remote sensing*. Elsevier Science, 1 edition, 2017.

[11] Dimitris Manolakis, David Marden, and Gary A. Shaw. Hyperspectral image processing for automatic target detection applications. *Lincoln laboratory journal*, 14(1), 2003.

[12] Michael T Eismann. *Hyperspectral remote sensing*. SPIE Press, 2012.

[13] Antonio Plaza, Qian Du, Yang-Lang Chang, and Roger L. King. High performance computing for hyperspectral remote sensing. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 4(3):528–544, 2011.

[14] Sebastian Lopez, Tanya Vladimirova, Carlos Gonzalez, Javier Resano, Daniel Mozos, and Antonio Plaza. The promise of reconfigurable computing for hyperspectral imaging onboard systems: A review and trends. *Proceedings of the IEEE*, 101(3):698–722, 2013.

[15] Dordije Boskovic Github repository. `https://github.com/dordijeb/HW-implementation-of-hyperspectral-target-detection-algorithm`, 2019.

[16] Pablo J. Martínez, Rosa M. Pérez, Antonio Plaza, Pedro L. Aguilar, María C. Cantero, and Javier Plaza. Endmember extraction algorithms from hyperspectral images. *Annals of Geophysics*, 49(1), 2006.

[17] Qian Du and Reza Nekovei. Fast real-time onboard processing of hyperspectral imagery for detection and classification. *Journal of Real-Time Image Processing*, 4(3):273–286, 2008.

[18] Carlos González, Javier Resano, Daniel Mozos, Antonio Plaza, and David Valencia. FPGA Implementation of the Pixel Purity Index Algorithm for Remotely Sensed Hyperspectral Image Analysis. *EURASIP Journal on Advances in Signal Processing*, 2010(1), 2010.

[19] Joseph C Harsanyi. *Detection and classification of subpixel spectral signatures in hyperspectral image sequences*. University of Maryland Baltimore County, 1993.

[20] Chein-I Chang. *Hyperspectral data processing*. Wiley-Blackwell, 2013.

[21] Lianru Gao, Bin Yang, Qian Du, and Bing Zhang. Adjusted spectral matched filter for target detection in hyperspectral imagery. *Remote Sensing*, 7(6):6611–6634, 2015.

[22] Chein-I C Chang, Hsuan Ren, and Shao-Shan Chiang. Real-time processing algorithms for target detection and classification in hyperspectral imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 39(4):760–768, 2001.

[23] M. Ylinen, A. Burian, and J. Takala. Updating matrix inverse in fixed-point representation: direct versus iterative methods. *Proceedings. 2003 International Symposium on System-on-Chip (IEEE Cat. No.03EX748)*.

[24] Carlos Gonzalez, Sergio Bernabe, Daniel Mozos, and Antonio Plaza. Fpga implementation of an algorithm for automatically detecting targets in remotely sensed hyperspectral images. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 9(9):4334–4343, 2016.

[25] 7 Series DSP48E1 Slice. `https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf`, 2018.

[26] AXI Reference Guide. `https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf`, 2018.

[27] AXI DMA v7.1 LogiCORE IP. `https://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf`, 2018.

[28] Johan Fjeldtvedt and Milica Orlandic. CubeDMA – Optimizing three-dimensional DMA transfers for hyperspectral imaging applications. *Microprocessors and Microsystems*, 65:23–36, 2019.

[29] Johan Fjeldtvedt. Direct memory access for hyperspectral imaging applications. Master's thesis, NTNU, 2018.

[30] Divider Generator v5.1 LogiCORE IP. `https://www.xilinx.com/support/documentation/ip_documentation/div_gen/v5_1/pg151-div-gen.pdf`, 2018.

[31] Vivado High-Level Synthesis Design Suite User Guide. `https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug902-vivado-high-level-synthesis.pdf`, 2018.

[32] Bin Yang, Minhua Yang, Antonio Plaza, Lianru Gao, and Bing Zhang. Dual-mode fpga implementation of target and anomaly detection algorithms for real-time hyperspectral imaging. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 8(6):2950–2961, 2015.

[33] Jie Lei, Yunsong Li, Dongsheng Zhao, Jing Xie, Chein-I Chang, Lingyun Wu, Xuepeng Li, Jintao Zhang, and Wenguang Li. A deep pipelined implementation of hyperspectral target detection algorithm on fpga using hls. *Remote Sensing*, 10(4):516, 2018.

[34] Jack Sherman and Winifred J. Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1):124–127, 1950.

[35] Jianwei Wang, Chein-I Chang, and Mang Cao. Fpga design for constrained energy minimization. *Chemical and Biological Standoff Detection*, 2004.

[36] Sergio Bernabe, Sebastian Lopez, Antonio Plaza, Roberto Sarmiento, and Pablo Garcia Rodriguez. Fpga design of an automatic target generation process for hyperspectral image analysis. *2011 IEEE 17th International Conference on Parallel and Distributed Systems*, 2011.

[37] Jingjing Wu, Yu Jin, Wei Li, Lianru Gao, and Bing Zhang. Fpga implementation of collaborative representation algorithm for real-time hyperspectral target detection. *Journal of Real-Time Image Processing*, 15(3):673–685, 2018.

[38] Hyperspectral Remote Sensing Scenes. `http://www.ehu.eus/ccwintco/index.php?title=Hyperspectral_Remote_Sensing_Scenes`, 2018.

[39] Sivert Bakken. Dimensionality reduction and target detection in hyperspectral remote sensing. Master's thesis, NTNU, 2018.

[40] Target Detection Blind Test. `http://dirsapps.cis.rit.edu/blindtest/`, 2009.

[41] T. Cocks, R. Jenssen, A. Stewart, I. Wilson, and T. Shields. The hymap airborne hyperspectral sensor: The system, calibration and performance. *Proceedings of 1st EARSEL Workshop on Imaging Spectroscopy, Zurich*, pages 37–42, 1998.

[42] D. Snyder, J. Kerekes, I. Fairweather, R. Crabtree, J. Shive, and S. Hager. Development of a web-based application to evaluate target finding algorithms. *IGARSS 2008 - 2008 IEEE International Geoscience and Remote Sensing Symposium*, 2008.

[43] Xiaoying Jin, Scott Paswaters, and Harold Cline. A comparative study of target detection algorithms for hyperspectral imagery. *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XV*, 2009.

[44] Qian Du and Reza Nekovei. Implementation of real-time constrained linear discriminant analysis to remote sensing image classification. *Pattern Recognition*, 38(4):459–471, 2005.

[45] Dordije Boskovic. HW/SW implementation of target detection algorithm. *NTNU*, 2018.

[46] Dominique D Lavenier, James P Theiler, John J Szymanski, Maya Gokhale, and Janette R Frigo. Fpga implementation of the pixel purity index algorithm. In *Reconfigurable Technology: FPGAs for Computing and Applications II*, volume 4212, pages 30–42. International Society for Optics and Photonics, 2000.

[47] Antonio Plaza, Pablo Martínez, Rosa Pérez, and Javier Plaza. A quantitative and comparative analysis of endmember extraction algorithms from hyperspectral data. *IEEE transactions on geoscience and remote sensing*, 42(3):650–663, 2004.

[48] Louise H. Crockett, Ross A. Elliot, Martin A. Enderwitz, and David Northcote. *The Zynq Book Tutorials*. Strathclyde Academic Media, 2015.

Appendix A

Dimensionality reduction

Principal component analysis (PCA) is a statistical method used in many fields, amongst others in hyperspectral processing. In this thesis, PCA was used as a dimensionality reduction technique where data is represented using fewer samples. Assuming that we have a spectral space with N dimensions, we want to reduce it to a lower-dimensional subspace $n < N$. Apart from reducing the number of dimensions, PCA would ideally surpress present noise in the hyperspectral data.

PCA transformation [12] diagonalizes a sample covariance matrix **C** (of the original data **X**), thus determining its eigenvalues and eigenvectors. They can be determined using the following equation:

$$det(\mathbf{C} - \sigma^2\mathbf{I}) = 0, \tag{A.1}$$

where σ^2 are the eigenvalues which represent the variance corresponding to a certain eigenvector, assuming that the sample covariance matrix **C** has full rank. Then, to find the eigenvectors $\mathbf{v}_j$, we solve a system of linear equations:

$$\mathbf{C}\mathbf{v}_j = \sigma^2\mathbf{v}_j. \tag{A.2}$$

If we place the eigenvalues on the diagonal of newly formed matrix **D** in descending order, and the corresponding eigenvectors in matrix **V**, it then follows that:

$$\mathbf{CV} = \mathbf{VD}, \tag{A.3}$$

which can easily be transformed to

$$\mathbf{C} = \mathbf{VDV}^T \tag{A.4}$$

due to the unitary property of the eigenvector matrix **V**. Finally, PCA transformation is:

$$\mathbf{Z} = \mathbf{V}^T\mathbf{X}. \tag{A.5}$$

where newly formed matrix **Z** has uncorrelated principal-component bands ordered in decreasing variance. Choosing n out of N PCA bands with largest eigenvalues is the last step of the dimensionality reduction using PCA.

Appendix B

Using HW/SW codesign implementation on Zynq platform

This tutorial describes the setup steps for HW/SW codesign implementation of target detection algorithms on ZedBoard development platform. The tutorial includes project creation, block design generation, simulation, synthesis and usage of SDK Environment for interaction between software and hardware modules. In this tutorial, Xilinx Vivado v2018.2 (64-bit) is used.

B.1 Creating project in Xilinx Vivado

Open Vivado and using Tcl Console change the directory to Vivado Project folder obtained from GitHub repository available at [15]. The folder contains all necessary VHDL files, SystemVerilog and VHDL testbenches, as well as Tcl scripts.

Run the *create_proj.tcl* script, which will recreate the project. The project is set to **xc7z020clg484-1** part number, which can be manually changed. First, it will include all source files, IP repositories, constraints and simulation files. Then, three block designs will be created:

- Simulation block diagram with AXI VIP
- Synthesis block diagram with AXI DMA
- Synthesis block diagram with Cube DMA

All block designs are automatically created and verified. The provided Tcl scripts create instances, set properties and create interface and port connections between corresponding IPs. Finally, HDL wrappers for block designs are generated. By selecting the active block design, the user can run simulation or synthesis tool.

Figures B.1 and B.2 show the generated block designs for simulation and synthesis.

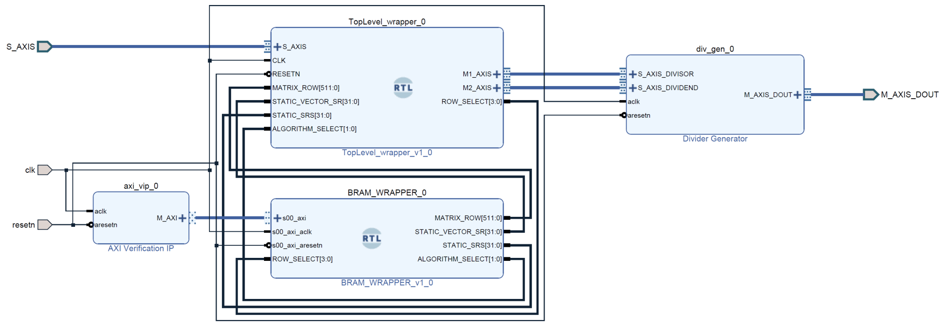


Figure B.1: Simulation block design diagram

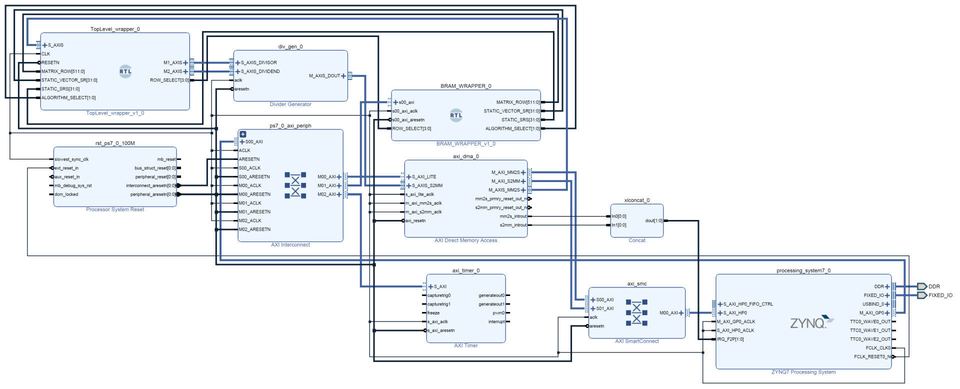


Figure B.2: Synthesis block design diagram

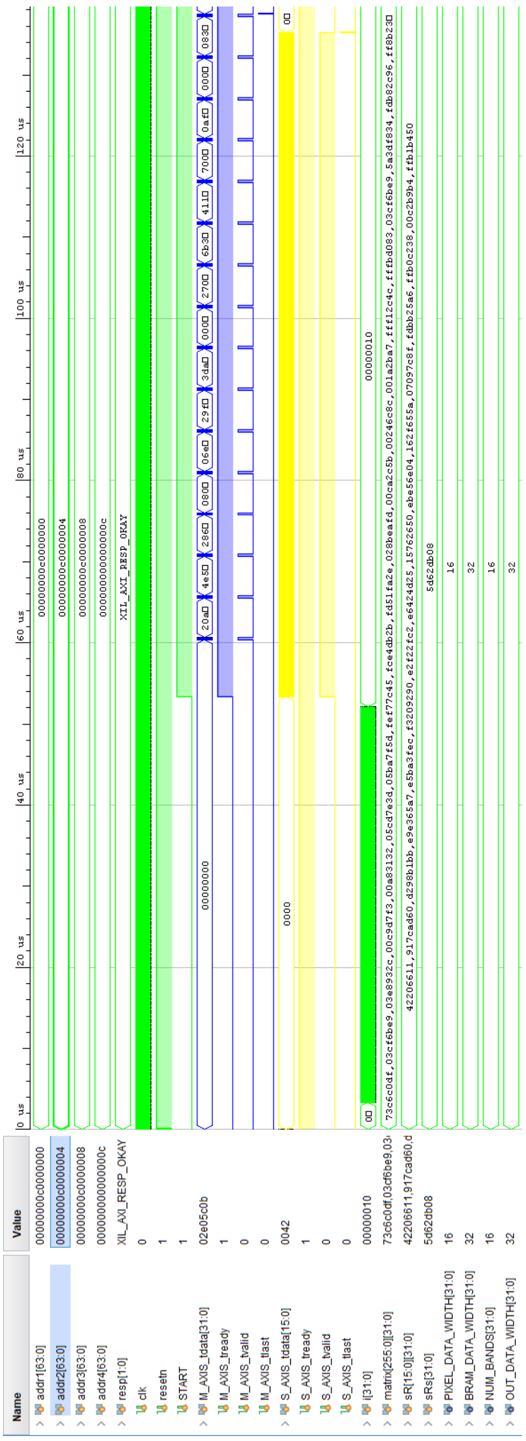
B.2 Simulation

First, check that SystemVerilog testbench is set to top simulation source. Simulation block design includes AXI Verification IP (VIP) which is used to control the AXI Register file in BRAM module. The testbench reads files containing data such as incoming pixel stream and corresponding static data described previously. Using AXI VIP instantiated in the testbench, we can write to the AXI-Lite register file:

$master\_agent.AXI4LITE\_WRITE\_BURST(addr1, prot, data1, resp);$

where *addr1* is the allocated address of the register and *data1* is the data being sent. The other VHDL testbench communicates over AXI-stream interface to send the pixels and receive detection statistics. The resulting statistics are saved in a file. An example waveform generated using aforementioned testbenches is shown in Fig. B.3.

The testbench can be easily modified to use different test patterns, data bit widths, etc.



Name	Value
addr1[63:0]	00000000c0000000
addr2[63:0]	00000000c0000004
addr3[63:0]	00000000c0000008
addr4[63:0]	000000000000000c
resp[1:0]	XIL_AXI_RESP_OKAY
clk	0
resetn	1
START	1
M_AXIS_tdata[31:0]	02e05c0b
M_AXIS_tready	1
M_AXIS_tvalid	0
M_AXIS_tlast	0
S_AXIS_tdata[15:0]	0042
S_AXIS_tready	1
S_AXIS_tvalid	0
S_AXIS_tlast	0
i[31:0]	00000010
matrix[255:0][31:0]	73c6c0df,03cf6be9,03
sR[15:0][31:0]	42206611,917cad60,d
sRs[31:0]	5d62db08
PIXEL_DATA_WIDTH[31:0]	16
BRAM_DATA_WIDTH[31:0]	32
NUM_BANDS[31:0]	16
OUT_DATA_WIDTH[31:0]	32

Figure B.3: Example simulation waveform

B.3 Synthesis and implementation

Using Vivado, run synthesis, implementation and generate bitstream. The implementation stage consists of routing and placement. Upon generating bitstream, the hardware should be exported. This is done in Vivado GUI as follows: *File* → *Export* → *Export Hardware*, with included bitstream.

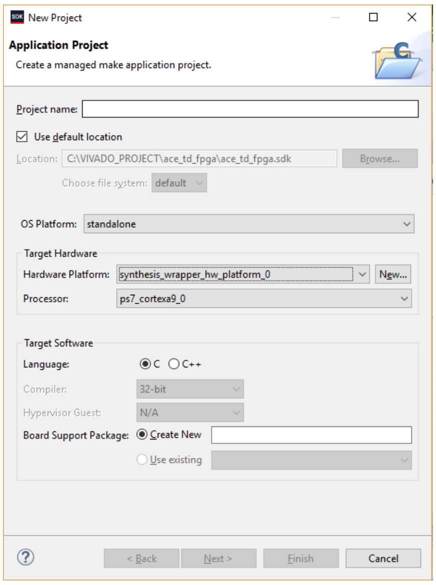
The synthesis, implementation and generate bitstream steps can be automatically executed when running *create_proj.tcl* if *conf.tcl* properties are set to:

- `set run_synthesis true`
- `set run_implementation true`
- `set gen_bitstream true`

When hardware configuration is exported, we can launch SDK from GUI.

B.4 Xilinx SDK

The Xilinx Software Development Kit (SDK) provides an environment for creating software platforms and applications targeted for Xilinx embedded processors. SDK works with hardware designs created with Vivado. SDK is based on the Eclipse open source standard. We first need to create a new application project in SDK. This is done by navigating to *File* → *New* → *Application Project*.


New Project

Application Project

Create a managed make application project.

Project name:

[✓] Use default location

Location: C:\VIVADO_PROJECT\ace_td_fpga\ace_td_fpga.sdk

Choose file system: default

OS Platform: standalone

Target Hardware

Hardware Platform: synthesis_wrapper_hw_platform_0

Processor: ps7_cortexa9_0

Target Software

Language: [✓ C] C++

Compiler: 32-bit

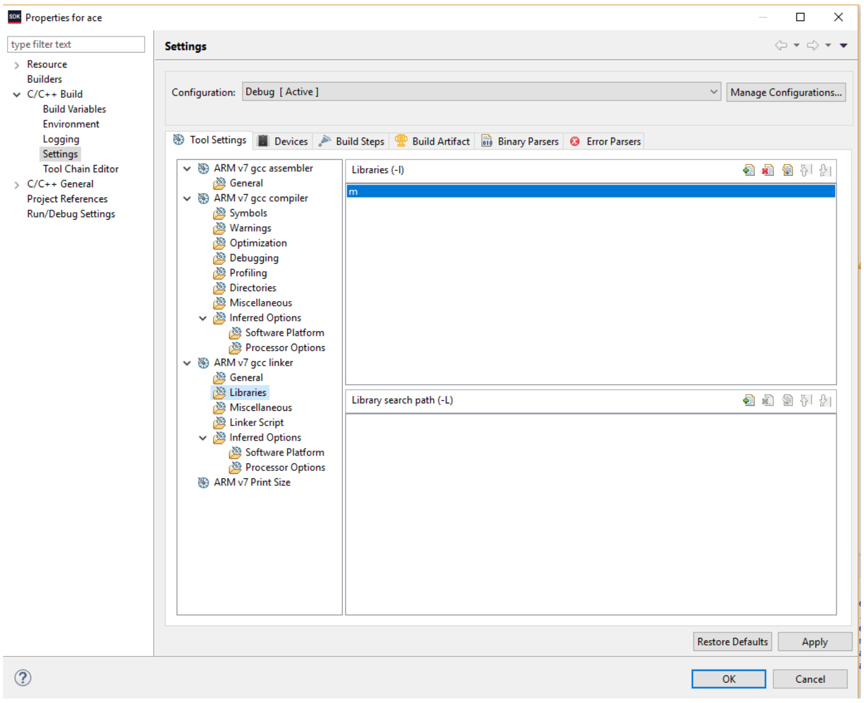
Hypervisor Guest: N/A

Board Support Package: [✓ Create New] Use existing

Figure B.4: SDK - create application project

In the *New Project* window, we set the desired project name, OS platform to standalone and create new Board Support Package (BSP). Hardware platform is obtained from exported hardware from Vivado. Now press *Next* → *Empty Application* → *Finish*.

The application project is now created, but we need to modify the BSP's settings to add support for SD card reading and writing. Navigate to *Modify BSP's settings* and add *xilffs* (Generic FAT file system library). Also, in project settings (Right click on the application project folder in project explorer and press the *C/C++ Build Settings*) we have to include the math library. Under *ARM v7 gcc linker* → *Libraries*, add **m**. In compiler settings, we can set the optimization level to -O3.



Configuration: Debug [Active]

Libraries (-l)
m

Library search path (-L)

Figure B.5: SDK - change project settings

Now available files in GitHub repository [15] under *HW_SW_CODESIGN/C/HW-SW/* can be copied to application project. *main.c* file runs the program which reads the data from SD card, pre-processes the data, and initiates transfers using DMA core. After all outputs from the dedicated hardware have been received, the program reports the execution time and writes the results to a file on SD card. The results can then be used in MATLAB environment to estimate the detection performance.

To read the printed strings from the program, USB UART port on the development board is used to connect with the PC. Then, inputs and outputs can be redirected to the console, using the terminal plugin program included with SDK. Detailed examples on how to set up the UART connection or create block designs are available in [48].

Dordije Boskovic

Hardware implementation of a target detection algorithm for hyperspectral images