# DEVELOPMENT OF PERFORMANCE OPTIMIZED ROTATION TOLERANT VIOLA-JONES BASED BLACKBIRD DETECTION, A THROUGHPUT OPTIMIZED ASYNCHRONOUS MAC IMPLEMENTATION, AND AUTOMATED WHEAT LODGING ESTIMATION

A Dissertation
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Nauman Jalil

In Partial Fulfillment of the Requirements
for the Degree of
DOCTOR OF PHILOSOPHY

Major Department:
Electrical and Computer Engineering

Feburary 2020

Fargo, North Dakota

# North Dakota State University
Graduate School

**Title**

Development of Performance Optimized Rotation Tolerant Viola-Jones Based Blackbird Detection, a Throughput Optimized Asynchronous MAC Implementation, and Automated Wheat Lodging Estimation

**By**

Nauman Jalil

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

**DOCTOR OF PHILOSOPHY**

SUPERVISORY COMMITTEE:

Scott C. Smith
Chair

Ivan T. Lima Jr.

Sudarshan K. Srinivasan

Simone Ludwig

Approved:

April 27, 2020
Date

Benjamin Braaten
Department Chair

# ABSTRACT

The research described in this doctoral dissertation focuses on three main topics: 1) performance optimization of the Viola-Jones Algorithm (VJA) for red-winged blackbird (Agelaius phoeniceus) detection, 2) further increasing performance of an already optimized asynchronous Multiply and Accumulate (MAC) unit, and 3) development of a framework to differentiate between lodging and non-lodging areas of a field from visible and multispectral aerial drone images. The first topic explores VJA rotational robustness, since VJA object detection is inherently not invariant to in-plane object rotation. An efficient method to detect rotated blackbirds is developed, which provides a balance between detection accuracy and computational cost. The second topic further optimizes a previously developed high-speed asynchronous 72+32×32 MAC, which was the fastest in the literature, resulting in a speedup of 1.36 while also decreasing area by 8%. The third topic develops a model to distinguish lodging from non-lodging plots, using a Support Vector Machine model trained with color, texture, Normalized Difference Vegetation Index (NDVI), and height features. The model prediction accuracy is around 90%, indicating good performance in distinguishing lodging from non-lodging plots.

# ACKNOWLEDGEMENTS

First piece of work was supported by a cooperative agreement with the U.S. Department of Agriculture, Animal and Plant Health Inspection Service, Wildlife Services, National Wildlife Research Center (QA#2348). Reference to trade names does not imply endorsement of commercial products or exclusion of similar products by the United States government. Research was conducted with approval from the North Dakota State University Institutional Animal Care and Use Committee (IACUC #A14068) and under permits from the U.S. Fish and Wildlife Scientific Collecting Permit (#MB39327B-0) and the North Dakota Game and Fish Department (#GNF03799268). Access to the birds for photographing was provided by Lucas Wandrie.

Third piece of work was supported by USDA/ARS. Research was conducted with approval from the North Dakota State University ABEN faculty for providing access to drone images. Title: Advanced UAS/UAV application and data management systems and bioinformatics tools integrate GxExM data for precision agricultural management.

# DEDICATION

This research thesis is dedicated to my family who has accompanied me to finish this study.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF ABBREVIATIONS

UASs .....Unmanned aerial systems.

VJA .....Viola Jones Algorithm.

ROI.....Region Of Interest.

HOG .....Histogram of Oriented Gradient.

SS .....Segment Size.

MAC .....Multiply and Accumulate.

NCL.....NULL Convention Logic.

GLP .....Gate-Level Piplining.

NCR .....NULL Cycle Reduction.

PP .....Partial Product.

NDVI.....Normalized Difference Vegetation Index.

SVM.....Support Vector Machine.

CNN .....Convolutional Neural Network.

DCNN .....Deep Convolutional Neural Network.

RELU .....Rectified Linear Unit.

GCP.....Ground Control Points.

Fcrs.....Coarseness.

Fcon.....Contrast.

Flin .....Linelikeness.

Fdir.....Ddirectionality.

ExG .....Excess Green Index.

# LIST OF SYMBOLS

*f'(x)*.....Derivatives with respect to the x-axis.

*f'(y)*.....Derivatives with respect to the y-axis.

*Sx*.....Centered Vertical Gradients.

*Sy*.....Centered Horizontal Gradients.

A1n.....New accumulator PP #1 value.

A1p.....Previous accumulator PP #1 value.

A2n.....New accumulator PP #2 value.

A2p.....Previous accumulator PP #2 value.

*f(i, j)*.....Gray-value at point (x, y).

$A_k(x, y)$.....Averages for every point of the input image over the neighborhood.

$E_{k.h}(x, y)$ .....Averages Corresponding to pairs of non-overlapping neighborhoods.

$E_{max}$ .....Highest Output Value.

$\sigma_4^{1/4}$ .....Fourth Moment about the Mean.

$\sigma$ .....Variance.

$P_{Dd(i,j)}$ .....Local Direction Co-Occurrence Matrix.

n.....Number of Peaks in the Histogram.

np.....Total Number of Peaks in the Histogram.

$\Phi_p$ .....pth peak location.

# 1. INTRODUCTION

The three main topics within this Ph.D. dissertation are development of an optimized rotation-tolerant implementation of the Viola–Jones object detection algorithm, which is then utilized to detect blackbirds from an aerial viewpoint; performance enhancements to a previously optimized asynchronous NULL Convention Logic (NCL) Multiply and Accumulate (MAC) unit [23]; and development of a framework to differentiate between lodging and non-lodging areas from visible and multispectral aerial drone images using different cameras. This chapter provides a more detailed introduction for each of the 3 main chapters, since the vast majority of each of those is a self-contained scholarly article, including 1 paper published in Springer's Journal of Real-Time Image Processing, 1 completed paper awaiting the submission window for the IEEE International Midwest Symposium on Circuits and Systems, and 1 paper to be submitted to an agricultural remote sensing journal, respectively.

## 1.1. Viola-Jones Object Detection Framework

Viola-Jones algorithm (VJA) is the first ever real-time face detection system [5] that can be applied to any color image. The accuracy and speed of the VJA is dependent upon three main components: the integral image for feature computation, adaboost for feature selection, and an attentional cascade for efficient computational resource allocation. The second chapter provides an overview of the VJA, and develops an optimized method to achieve rotation tolerant VJA based blackbird detection. Due to their unpredictable appearance and the wide range of positions they can take, identifying birds in images is a challenging task. Typically, VJA gives multiple detections, therefore a post-processing step is also computed to reduce detection redundancy.

### 1.1.1. Features

The main objective of using features as the data input to a learning algorithm rather than raw pixel values is to reduce variability compared to raw input data, and thus facilitate better classification. The sophistication of feature evaluation is a very important element, as almost all algorithms for object detection slide a fixed-size window over the input image at all scales.

Locally normalized Histogram of Oriented Gradient (HOG) descriptors offer excellent performance compared to other existing feature sets, including wavelets [4]. There are several benefits to the HOG / SIFT (Scale-Invariant Feature Transform) representation. It captures edge or gradient structure that is very typical of local shape, and it does so with an easily controllable degree of invariance to local geometric and photometric transformations in a local representation.

### 1.1.2. Cascade of Classifiers

A cascade of classifiers is a degenerated decision tree in which a classifier is trained at each stage to detect nearly all objects of interest (frontal faces for example) while rejecting some fraction of non-object patterns [18]. The AdaBoost algorithm [18] was used to train each stage. Adaboost is a powerful algorithm for machine learning. By re-weighing the training samples, it can learn a strong classifier based on a (large) set of weak classifiers. It only allows poor classifiers to be marginally better than chance. All classifiers that use one feature from our feature pool in combination with a simple binary threshold decision are poor classifiers. The feature-based classifier that best classifies the weighted training samples is added at each boosting round. As shown in Figure 1.1, the first classifier, correctly classifies the negative samples, but three positive samples were classified incorrectly. For the $2^{nd}$ classifier, the weights of the incorrectly classified positive samples were increased; hence the second classifier was able

to classify the positive samples correctly, however three negative samples were then classified incorrectly. This same procedure is repeated again, such that the third classifier correctly classifies the previously incorrect negative samples. This procedure is repeated over and over again to generate strong classifiers, as overviewed in the below algorithm.

### 1.1.3. Adaboost Algorithm Overview

1. Weights load
2. Weight normalization
3. Based on the weighted error, the best weak classifier is selected
4. Update the weights based on the error of the selected classifier
5. Repeat Steps 2–4 n times (where n is the desired number of weak classifiers)

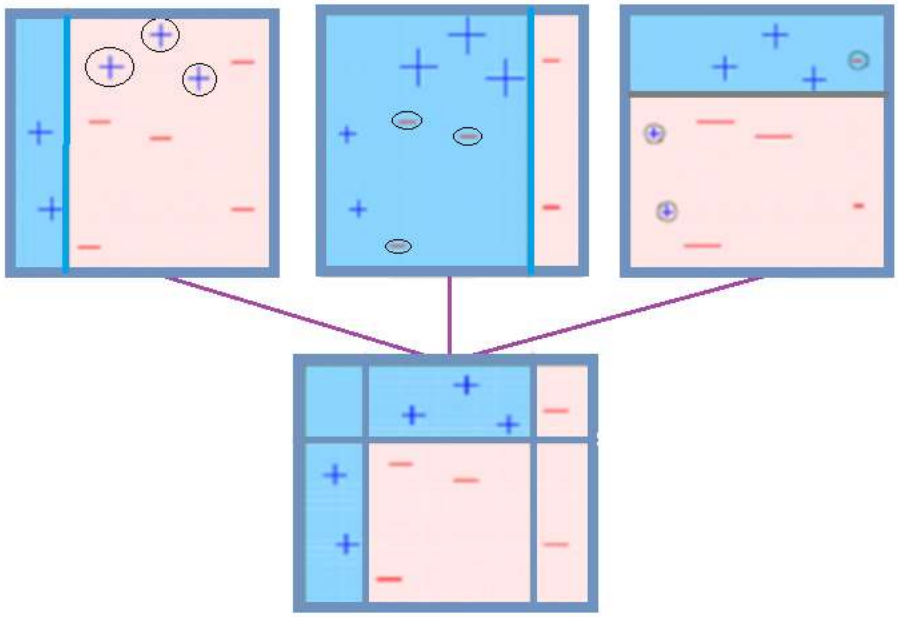


Fig. 1.1. Visual Display of Boosting.

Large numbers of negative samples are eliminated in the initial stages with very little processing time, to avoid unnecessary computation, whereas the following stages eliminate further negative samples but require additional computation. The cascade structure reflects the

fact that a vast majority of subwindows are negative in any single image. As such, at the earliest possible point, the cascade tries to eliminate as many negatives as possible.

### 1.2. Asynchronous Circuits Background

In the 1950s, the theory of asynchronous logic was first proposed (i.e. circuits being self-timed rather than being externally timed by a periodic clock signal, like synchronous circuits). Since then, industry and academia have carried out many research and development projects on asynchronous circuits, inventing and demonstrating various asynchronous design paradigms in silicon.

Table 1.1. Comparison of Asynchronous and Synchronous Logic Paradigms.

| **Asynchronous** | **Synchronous** |
|---|---|
| Computation is continuous | Computation is decrete |
| Handshaking control<br><br>*delay for some paradigms is average of possible circuit paths* | Global clock control<br><br>*delay based on maximum overall possible circuit paths* |
| Local data-driven switching<br><br>*Usually, computation results in lower energy operation* | Clock-driven global switching<br><br>*Calculation needs careful clock gating for low power operation* |
| Performance determined solely by device speed, for some paradigms | Performance determined by device speed and additional operating margins |
| some paradigms require data encoding, which requires additional wires | non-encoded data, i.e. one wire per bit |

Asynchronous circuits can be grouped into two broad types, bounded delay (BD) and quasi-delay insensitive (QDI), each with many different implementation paradigms. Typically, bounded delay circuits use a bundled data representation where data requires one wire per bit (just like synchronous circuits) and contains one additional wire to signal when a bundle of data wires is valid. On the other hand, QDI circuits encode the validity of data along with the actual

data being transmitted, thereby requiring more than one wire per bit, such as dual-rail logic, which utilizes 2 wires to represent 1 bit of data. Sometimes other data encoding is also used, including quad-rail logic (i.e., a 1-hot encoding scheme using four wires to represent 2 bits of data) [28], 1-hot encodings needing more than two wires per bit [28], and data encoding that relies on rail transitions [29]. These encodings enable the QDI circuit to know when its data is valid without referencing time, and to generate handshaking signals on the basis of this information to relay this to other parts of the circuit.

### 1.2.1. Asynchronous Circuit Advantages

Most asynchronous paradigms share a number of common benefits, as listed below. That's by no means a complete list, only a few general advantages, while particular asynchronous paradigms may have other more specific advantages.

- For timing / synchronization, asynchronous circuits do not use a clock. Alternatively, handshaking protocols control and organize the conduct of the circuit allowing more flexibility in timing. In BD asynchronous circuits, for example, the circuit will function properly as long as the propagation delay of each independent pipeline stage is shorter than the predetermined delay bound for that particular stage. Because data and control are encoded together, the timing requirement for QDI circuits is even more robust, and completion detection is used to produce handshaking signals rather than rely on a specified delay for control.
- Due to process / voltage / temperature (PVT) variability, synchronous circuits may delay fluctuations in circuit elements. Since these variables are unavoidable, especially as the transistor size continues to shrink, it is a challenge to ensure reliable operation for large complex synchronous circuits. On the other hand, asynchronous circuits are much more robust in

terms of PVT variability due to their agile timing requirements, as the handshaking protocols automatically tolerate the induced delay fluctuations, thus ensuring proper circuit behavior.

- In a synchronous circuit all stages of the pipeline are controlled by the same clock, the period of which should be longer than any stage's worst-case delay. QDI asynchronous circuits yield data-dependent, average-case performance as opposed to this worst-case performance. When new data comes, each stage of the pipeline finishes its computation as fast as possible; and after completion of its computation, each stage is able to pass the result to its next stage and receive new data from its previous stage. However, although the delay of each pipeline stage depends on the specific data pattern being processed, a given pipeline stage may finish its computation earlier or later than its adjacent stages, in which case it will automatically wait to transmit and receive data, as necessary, assured by its handshaking protocol. Therefore, there is no need to scale performance to accommodate the slowest stage, leading to improved average-case performance.
- In synchronous circuits, unless specifically gated, the clock switches continuously, typically at a high rate of speed; and thus, all components of the clock tree switch continuously, which uses up dynamic power, even if useful work is not undertaken. Furthermore, all flip-flops have internal gates which transition each edge of the clock, even if the output of the flip-flop sometimes doesn't change. Asynchronous circuits, on the other hand, are usually event-driven, which can be known as automatic clock gating. While waiting for new data to be processed after all previous computations have been completed, asynchronous circuits remain inherently idle, so transitions do not occur.
- The high-frequency clock signal in synchronous circuits causes substantial spikes in the emission of electromagnetic interference (EMI), especially at the fundamental clock

frequency, which can become a problem for surrounding circuits. In addition, the concentrated switching activity at the edge of the clock produces a large amount of electrical noise that might corrupt neighboring wires. On the other hand, asynchronous circuit switching activity is far more distributed and depends on localized handshaking signals rather than a periodic global clock, resulting in much reduced noise and a much flatter EMI emission spectrum, without large spikes, making it easier to integrate asynchronous circuits with other circuit and system components.

### 1.2.2. Summary of Applications for Asynchronous Circuits

Despite the many asynchronous circuit advantages discussed above, the semiconductor industry has been dominated by synchronous circuits. There are several explanations for this, but most are due to the fact that synchronous circuits have been successful enough to design most of the next-generation ICs until recently. Additionally, the synchronous model is typically what is taught in the curricula of Electrical Engineering, Computer Engineering, and Computer Science, such that the large majority of IC developers have little knowledge of asynchronous circuits to compare their advantages and tradeoffs with synchronous circuits.

Nevertheless, as transistor size continues to decrease, industry is looking at asynchronous circuits to solve the increasing power dissipation and process variance issues associated with today's cutting edge semiconductor processes. There are many applications in which asynchronous circuits explicitly outperform with unmatched advantages their synchronous counterparts, exploiting their previously discussed strengths. One example of artificial intelligence applications is the growing wave of neuromorphic computing. Because neurons are eventdriven, the natural choice for these devices is an asynchronous implementation.

A hardware implementation of the MAC operation is one of the main differentiators between a microprocessor and a Digital Signal Processor (DSP); and DSPs are often utilized to

implement neural networks (NNs), although an application specific integrated circuit (ASIC) implementation would be faster and more power efficient, at the expense of substantially increased cost. In either case, the enhanced performance MAC designed as part of this dissertation would be applicable.

## 1.3. Machine Learning Model to Estimate Wheat Lodging

Lodging occurs when the stems of the plant break or bend over to permanently displace plants from their appropriate upright position. For many crops, including wheat, it is a common issue and may be caused by external factors, including wind, rain, or hail, and morphological factors such as slim or fragile stem structures.

### 1.3.1. Data Collection and Processing

An agricultural drone is an unmanned aerial vehicle used for agriculture to help manage crops and track crop growth. Sensors and equipment for digital imaging will give farmers a clearer image of their fields. This knowledge can be useful to improve crop yields and farm production.

Fig. 1.2. DJI Phantom 4D RTK Drone.

DJI has revolutionized drone technology from the ground up, to achieve a new drone accuracy level – Phantom 4 RTK (shown in Figure 1.2) provides centimeter accurate data to customers while having less ground control points. In addition to automated flight safety and accurate data gathering, the Phantom 4 RTK stores satellite observation information for Post Processed Kinematics (PPK), which can be used with DJI Cloud PPK Service. Additionally, a new RTK module is specifically incorporated into the Phantom 4 RTK, offering real-time, centimeter-level targeting data to track absolute image metadata accuracy.

Fig. 1.3. DJI MATRICE 600 Pro.

For this work, a DJI Phantom 4D RTK drone, attached with an RGB camera (Phantom 4 Pro V2.0) was used to obtain the RGB images (DJI-Innovations, Inc., ShenZhen, China); and a DJI MATRICE 600 Pro (DJI-Innovations, Inc., ShenZhen, China), shown in Figure 1.3, attached

with a multispectral sensor (MicaSense RedEdge-MX Professional Multispectral Sensor, Simi Valley, CA, U.S.), was used to obtain the multispectral images.

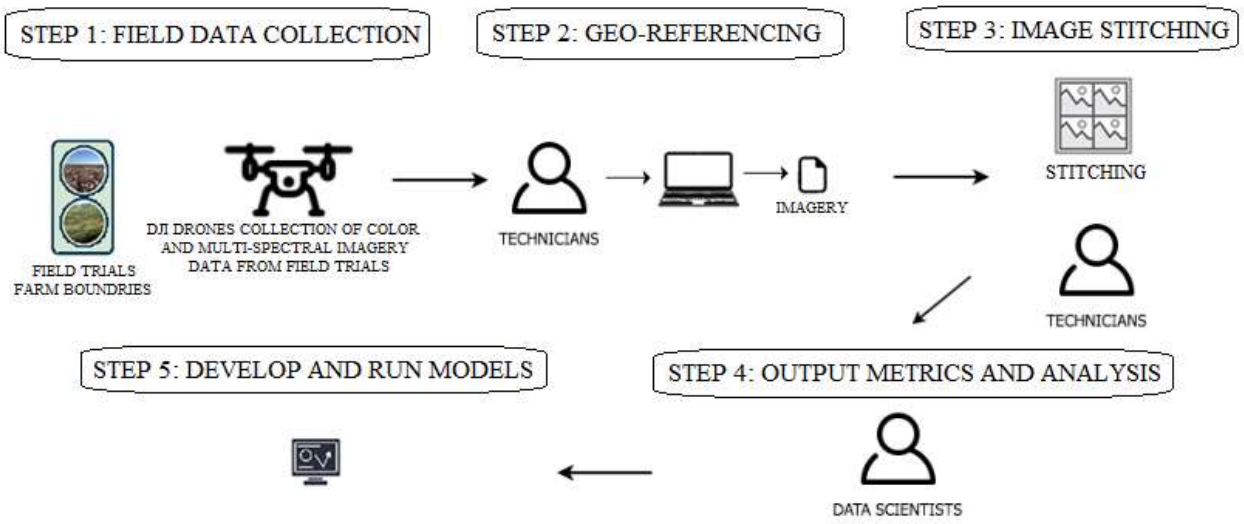



Fig. 1.4. UAV Imagery Analytics Design Flow.

The UAV is designed to collect, process and publish UAV imagery in frame environments. Imagery types will include RGB and Near Infrared (NIR / NDVI). The primary aim of the UAV Imagery Pipeline is to provide insights and measurements for customers to help them make corporate decisions. Completing each of the steps in figure 1.4 produces its own message which will then enable the subsequent process to run. Drone imagery strengthens functionality on construction sites, providing spatial data for decision-makers to inspect areas related to project planning and monitoring. Near-infrared images provide updates of fields in real time, enhancing workflow and increasing efficiency. In the end, advanced AI-enhanced drone mapping technology provides a safer environment for decision-makers and stakeholders to obtain and communicate insights. With fast delivery times and low maintenance costs, drones provide a mobile mapping system for the quick, cost-effective, and secure collection of high-resolution spatial data.

### 1.3.2. SVM (Support Vector Machine)

A Support Vector Machine (SVM) is formally defined by a separating hyperplane as a discriminatory classifier. In other words, given the labeled training data, an optimal hyperplane is produced by the algorithm that categorizes new examples. This hyperplane is a line dividing a plane into two sections in two dimentional spaces where it lies on either side in each class.

SVM is a fast and reliable classification algorithm that executes very well with a limited amount of data. A support vector machine collets the data points and outputs the hyperplane that best separates the tags (which is simply a line in two dimensions). This line is the decision boundary, which classifies everything that falls on one side as one thing, and anything on the other side as something else (e.g., square vs. circle, as shown in Fig. 1.5a).

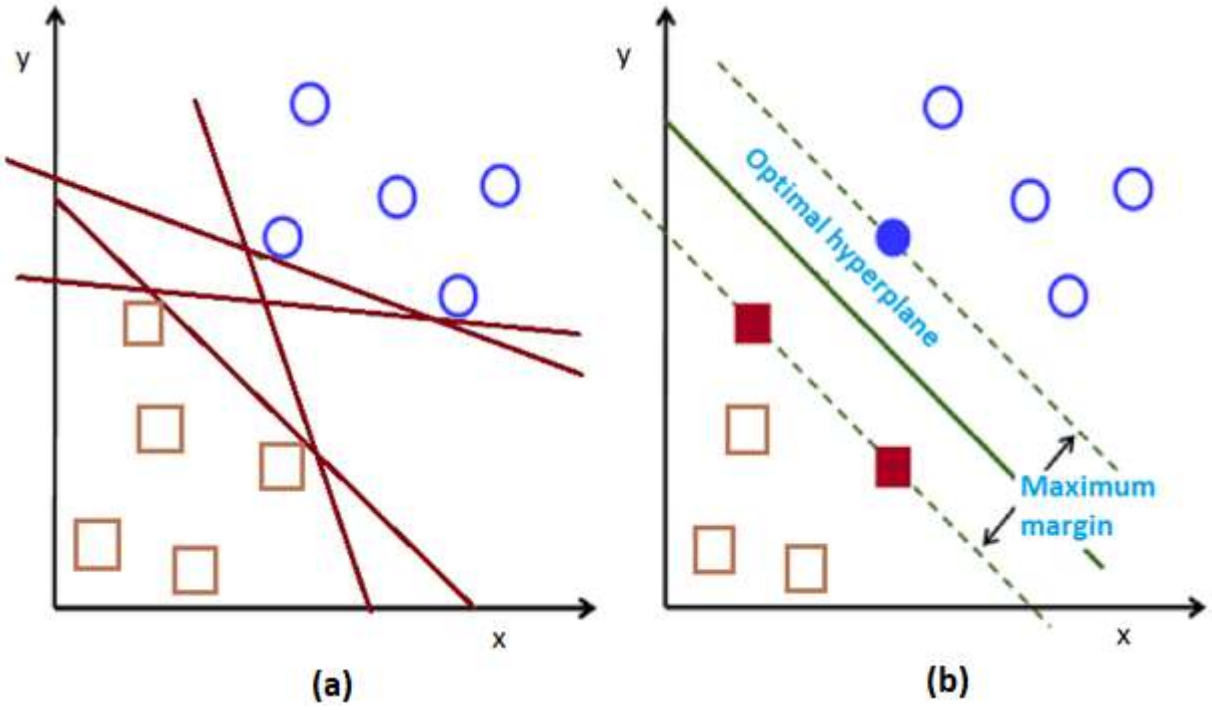



Fig. 1.5. Possible Hyperplanes.

But what's the best hyperplane, exactly? For SVM, it’s the one that maximizes the margins from both tags, as shown in Fig. 1.5b. There are many different hyperplans that could be

used to distinguish the two groups of data points. The goal is to find a plane with the maximum margin, i.e. the maximum distance between the two classes' data points. Maximizing the margin length provides some reinforcement in order to be able to classify future data points with increased confidence.

### 1.3.3. Convolutional Neural Networks (CNNs)

In pattern recognition and machine learning, deep learning neural networks (including recurrent ones) have won numerous contests in recent years. Shallow and deep learners are differentiated by the depth of their credit allocation routes, which are chains of potential learning causal links between actions and effects. Machine learning is the technique of using algorithms to analyze data, to learn from data and then make a prediction on new data.

Convolutional Neural Networks (CNNs) combine biology, math, and computer science, and are among the most prominent computer vision developments. In 2012, CNNs gained popularity when Alex Krizhevsky used them to win the ImageNet contest that year, decreasing the classification error record from 26 percent to 15 percent, which was a remarkable improvement at the time [66]. Classification of images is the process of taking an input image and creating a class (car, person, etc.), or a likelihood of classes, that best describe the image. CNNs exploit the fact that the input data is 2-D, so their architecture is tailored accordingly.

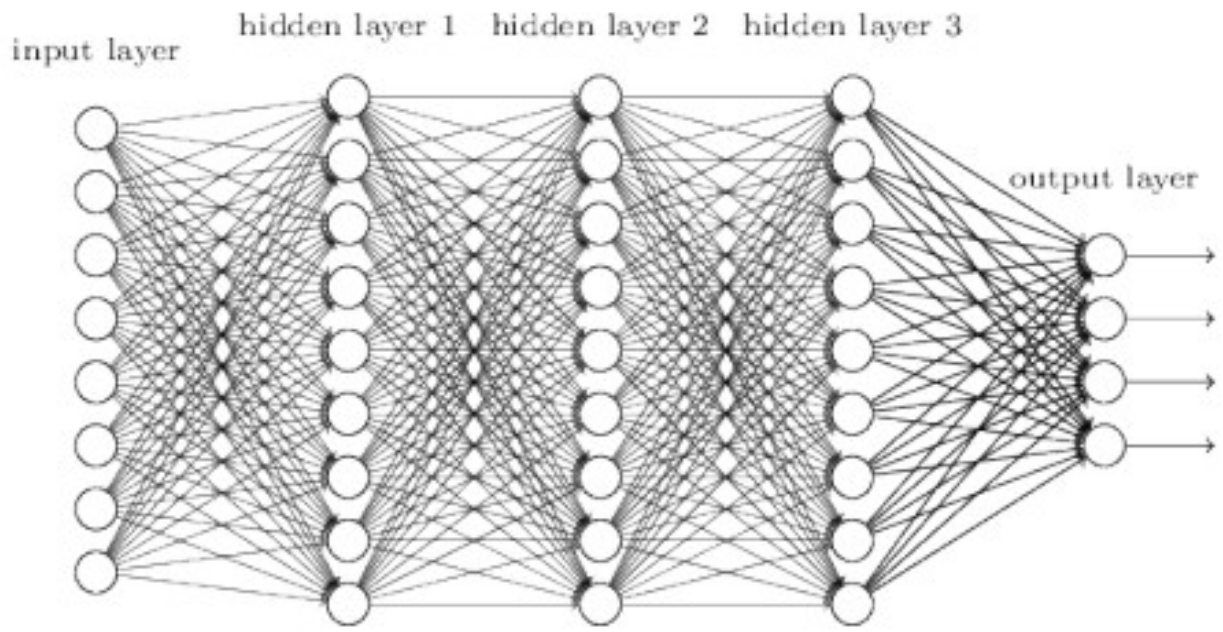



Fig. 1.6. 4-Layer Convolutional Neural Network [30].

A more comprehensive description of what the CNN utilized in this work on detecting crop lodging is doing is taking the picture, moving it through a series of convolutionary, nonlinear, pooling (downsampling) and fully connected layers, and obtaining an output. The result can either be a single class or a class probability that best describes the image. The hard part is understanding what each layer is doing. To construct a CNN architecture, as shown in Figure 1.6, three main categories of layers are utilized: Convolutional Layer (CONV), Pooling Layer (POOL), and Fully-Connected Layer (FC). These layers will be stacked to form a complete CNN architecture.

A simple CNN for lodging/non-lodging binary classification could have the architecture [INPUT - CONV - RELU - POOL - FC]. INPUT [ 64x64x3] contains the image's raw pixel values, in this case a picture of width 64, height 64, and three color channels: R, G, B. Convolutional layers (CONV) are precisely what makes a CNN a CNN. CONV receives data, transforms the data in some way, and then outputs the transformed data to the next layer. CNNs

are capable of detecting image patterns. For example CONV could be configured as a filter to detect an edge in an image. Some filters may detect corners, while others may detect squares, circles, etc. These are basic geometric filters that would form the beginning layers of a CNN. The deeper the network the more complex the filters become. In later layers, filters can detect specific objects, such as eyes, ears, hair, fur, feathers, scales, or beaks; and in even deeper layers, filters can detect even more complex objects, such as complete dogs, cats, lizards, or birds.

A CONV layer calculates the output of neurons connected to local input regions, each measuring a dot product between their weights and a specific region to which they are attached in the volume of input. A dot product is the element-wise multiplication between the input and filter, which is then summed up, resulting in a single value. Convolutional layers transform the input and transfer the result to the next layer. This can lead to volume like [ 64x64x8 ] when using eight filters.

An elementary activation function, such as max(0,x), will be applied to the RELU (rectified linear activation function) layer. The RELU layer consists of a simple equation (e.g., max(0,x) directly returns the given input value, or 0.0 if the input is 0.0 or less. This leaves the volume size unchanged ([ 64x64x8 ]). Since rectified linear units are almost linear, many of the properties that make linear models simple to optimize are retained. The output for a particular node is the result of the activation function, which is then transmitted as part of the input for the nodes in the next layer; and this continues for each layer in the network before the output layer, which generates the model output. This method is referred to as forward propagation. For example, in order to classify animal pictures, each of the output nodes would correspond to a

different animal type; and the output node with the highest activation would be the output that the model determined to be the best match for the corresponding input.

A POOL layer will perform a spatial dimension (width, height) downsampling operation, resulting in volume like [ 32x32x8 ]. The FC layer calculates the class scores, resulting in a volume [1x1xN], where each of the *N* outputs relate to a specific classification. For example, if you had a program for digit classification, *N* would be 10, because there are 10 digits. Every number in the entire *N* dimensional vector represents the likelihood of its respective class. For the digit classification problem, an output vector could be [0 .05 .1 0 0 .8 0 0 .05 0], which represents a 10% probability that the image is 2, a 5% probability that the image is a 1 or 8, and an 80% probability that the image is a 5. A typical CNN architecture would look like [INPUT -> CONV -> RELU -> CONV -> RELU -> POOL -> CONV -> RELU -> POOL -> FC].

To train the CNN, loss is calculated for each given input as the difference between what the model classified the input as and what the input actually was. Gradient descent is used to minimize the loss function by taking the loss function derivative with respect to the weights in the CNN through back propagation, where CNN node weights are updated starting from the output layer and working backward toward the input layer, to minimize the loss. This would be the mathematical equivalent of a dL/dW in which *W* is the weights at a given layer. All the weights of the filters are updated in the opposite direction of the gradient, given by mathematical equation W = Wi - n(dL/dW). The learning rate (n) is a parameter selected by the programmer. A high learning rate means that the weight updates take bigger steps and therefore, it may take less time for the model to converge on an optimum set of weights. However, a high learning rate may result in jumps that are too big and not precise enough to reach the optimum point, as shown in Figure 1.7. Gradient descent aims to increase the output value for the correct output node, while

decreasing the output values for incorrect output nodes, which in turn reduces the loss. Back propagation performs this in a backwards fashion because each layer's input depends on the weights in the previous layers, so when the weights in those previous layers change, that affects the outputs of subsequent layers.

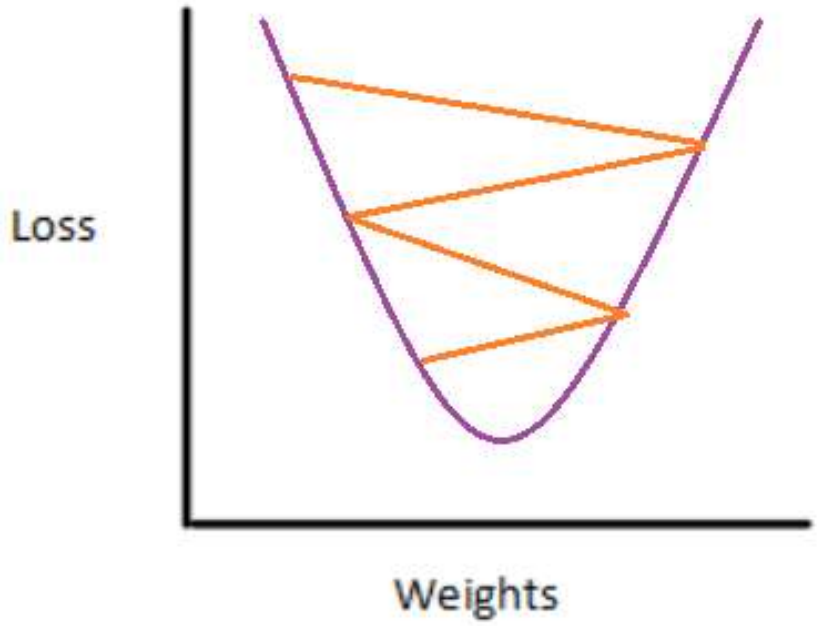



Fig. 1.7. Consequence of High Learning Rate.

### 1.3.4. Drone Applications in Precision Agriculture

Most agricultural monitoring is dependent on rigorous human-based practices, considered to have a high percentage of error [67]. As a result, technological solutions to support agricultural activities such as GPS-guided tractors and computer vision [68] and unmanned aerial vehicles (UAVs) are becoming a reasonable option due to improved sensor performance, developments in machine learning and inexpensive equipment. Currently, increasing use of unmanned aerial vehicles (UAV) has been recorded against a wide range of crops with promising results. The rapid development of agricultural UAV is primarily due to its major benefits which includes: (1) UAV does not need a dedicated airport or navigation station and may land at the

edge of the cultivated land, the highway and the top of the truck, reducing airline and agricultural and forestry expenditure; (2) UAV's short turning radius may allow it to swing and rotate in the air flexibly; (3) UAV's high climb rate could allow it to fly vertically and have super low flight performance; (4) UAV is ideal for high-efficiency operations in rough terrain and small plots; (5) high automaticity, reduced flight crew, low labor intensity and fast operation and maintenance compared to traditional manned aircraft [69-73].

#### *1.3.4.1. Need for Wheat Lodging Detection Applications*

Manual wheat lodging evaluation is laborious, as workers need to walk across a large field area, typically in a high temperature (e.g., 35°C). In addition, manual measurement is also subjective, as each individual worker may come up with different results. Furthermore, considering the inaccurate tools (e.g., tape measure) and accumulation error incurred during the measurement, the results may be inaccurate. Therefore, an automatic lodging system, as developed herein, is needed to replace the current manual method.

# 2. PERFORMANCE OPTIMIZATION OF ROTATION TOLERANT VIOLA-JONES BASED BLACKBIRD DETECTION

## 2.1. Introduction

Modern development is destroying habitat for many species, including pest birds, forcing them to rely more and more on farmed crops for survival, resulting in crop loss. In Northern U.S.A and Canada, extreme weather conditions only allow for 1 crop per year, such that crop losses in these areas have an even larger economic impact. Losing crops is not new; people have been using frightening and protection devices for hundreds of years to mitigate bird damage. While netting may work for smaller areas, it is not practical or cost effective for the typical much larger farm. Another approach, avicides, are expensive and negatively impact the environment [1,2]. Most often these control methods do not mitigate substantial damage caused by birds; and their cost can exceed crop losses.

Blackbirds (Icteridae) are one of the leading causes of damage to sunflower crops, an important North Dakota commodity. Nationally, blackbird damage to sunflower crops exceeds $13 million dollars. The most limiting factor on production of sunflowers is damage caused by blackbirds, which generally rank third behind insects and plant spacing [3]; however, blackbirds cannot be easily controlled, unlike insects and plant spacing. To help reduce damage and loss, new methods to detect blackbirds and associated sunflower damage are needed.

In combination with unmanned aerial systems (UASs) and other technologies, computer vision systems offer promise to detect blackbirds and/or crop damage. The advantage of this system is its autonomous nature, which saves labor cost as well as time. The UAV system combines visual, audio, and chemical methods to deter pest birds for long term periods [12]. Currently the demand of UAV systems in research and industry has grown rapidly to provide a

safe and cost-effective alternative compared to traditional methods [13]. InPixal delivers software and hardware components that can be integrated inside the UAS to provide real time images to a ground station [15]. From these top down aerial photos, blackbirds can be effectively detected through the developed image processing techniques.

This chapter focuses on blackbird detection with the Viola Jones Algorithm (VJA), which is a real-time object detection system that can be applied to any color image [4, 5]. Detecting blackbirds in images is a challenging task due to their variable appearance and wide range of poses. Since the VJA object detection method is inherently not invariant to in-plane object rotation, additional effort is required during training and detection [8]. The main objective in this chapter is to provide a balance between detection accuracy and computation cost. In order to achieve a high detection rate, the VJA detector is applied multiple times to cover all 360° angles. A large number of negative sample (non-bird) sub-window blocks can be eliminated in the initial stages with very little processing time. Hence, VJA detection covering all 360° only need be performed on selected blocks instead of all image blocks. Parameters such as number of stages, segmentation block size, and selected block criteria, have been investigated to determine their effect on detection accuracy and performance.

As proof of concept, top down photos of blackbirds from a single camera in an outdoor aviary setting were taken to obtain several hundred aerial images of blackbirds. A training set of over 500 birds was created, with each bird image being on average 48 x 80 pixels. The classifier was trained with blackbirds manually aligned at one specific angle.

The main contribution of this chapter is development of a VJA based detector to effectively identify rotated blackbirds covering the entire 360° while minimizing computation time. Compared to previous work [8, 14], the developed algorithm achieves an equivalent

detection rate (87% with 0 false detections) to running the full 20-stage VJA 360 times on an image rotated 1° each iteration, while decreasing computation time by 25% compared to the baseline algorithm that rotates the image 45° each iteration, with a detection rate of only 73% with 0 false detections, using the full 20 VJA stages.

The remainder of this chapter is organized as follows: Section 2 presents the related work; Section 3 describes the background of the VJA detection framework; Section 4 discusses the baseline algorithm; Section 5 presents the proposed algorithm; and Sections 6, 7, and 8 are the conclusions, acknowledgement, and references, respectively.

**2.2. Related Work**

The article "An analysis of the Viola-Jones face detection algorithm" gives the complete VJA description, the first ever real-time face detection system that can be applied to any color image. The accuracy and speed of the algorithm is dependent upon three main components: the integral image for feature computation, adaboost for feature selection, and an attentional cascade for efficient computational resource allocation. Typically, VJA gives multiple detections, therefore a post-processing step is also proposed to reduce detection redundancy [5].

In the paper "Bird's eye view: Detecting and Recognizing Birds using the BIRDS 200 dataset", the author has demonstrated that Naive Bayes-Nearest Neighbor (NB-NN) algorithm using C-SIFT feature descriptor gives better performance in comparison with Haar-like features. Since the dataset contains images of birds for which color is a key attribute for recognition, the author has selected the C-SIFT descriptor that includes color information and is also invariant to color variations. The results obtained using NB-NN with C-SIFT descriptors gives an accuracy of 25% [6]. Note however, that [6] is discriminating between different bird species, whereas

blackbirds are the predominant species being detected in this chapter, so color is not an important factor for this work.

Compared to previous algorithms (e.g., Rowley-Baluja-Kanade detector and Schneiderman-Kanade detector) that use statistical guarantees to discard regions not likely to contain an object of interest [18], the Viola-Jones Algorithm uses a method of combining complex classifiers in a cascade to quickly eliminate the background region, to focus on more promising object-like regions. VJA detection rate is comparable to these alternative previous algorithms; however, VJA is about 15 times faster than the Rowley-Baluja-Kanade detector [20] and roughly 600 times faster than the Schneiderman-Kanade detector [21].

Scale Invariant Feature Transform (SIFT) is a method for extracting distinctive invariant features from images [16]. Although SIFT features are invariant to image scale and rotation, and have proven to be very efficient in object recognition applications across a substantial range of affine distortion, SIFT requires a large computational complexity, which is a major drawback, especially for real-time applications [22].

Histogram of Oriented Gradient (HOG) descriptor is a method that primarily depends on evaluating a well-normalized local histogram of image gradients in a dense grid. It captures the key characteristics of an object's local shape by computing gradient orientation and magnitudes, and yields very good results for object detection, reducing false positive rates relative to the best Haar wavelet based detector [4]. Experimental results show that HOG descriptors outperform SIFT and other feature sets in terms of performance [22] and better detection rate with fewer false detections [4]. Hence, this chapter focuses on utilizing VJA with the HOG feature set.

One observation shared among the previous literature is that a classifier trained on all rotated angles appears to be hopelessly inaccurate, concluding that a single classifier trained to

detect all rotated angles of a bird is not learnable with existing classifiers [4, 7]. The research in [7] shows strictly aligned training data gives better performance for in-plane rotational robustness. Therefore, a multi-view detector must be carefully constructed by putting together a collection of detectors, with each one trained for a single viewpoint. This chapter addresses in-plane rotation variation, where birds are rotated in the image plane, and the multi-view detector is a collection of Viola-Jones detectors.

### 2.3. Background: Viola-Jones Object Detection Framework

References [5, 8, 9] provide a good overview of the Viola-Jones algorithm, which consists of two parts: training and detection. Training is computationally intensive, but is only required once, and can be used thereafter for detection; hence, optimizing detection is the focus of this chapter.

VJA consists of a series of classifiers divided among multiple cascades, as shown in Figure 2.1. Large numbers of negative samples are eliminated in the initial stages with very little processing time, to avoid unnecessary computation, whereas the following stages eliminate further negative samples but require additional computation. As the positions and sizes of the birds are uncertain, a fixed sized sub-window (as shown in Figure 1) will slide the entire image from top left to right bottom with 50% overlap; then the procedure is repeated with rescaled image, so birds of different sizes can be detected [11].

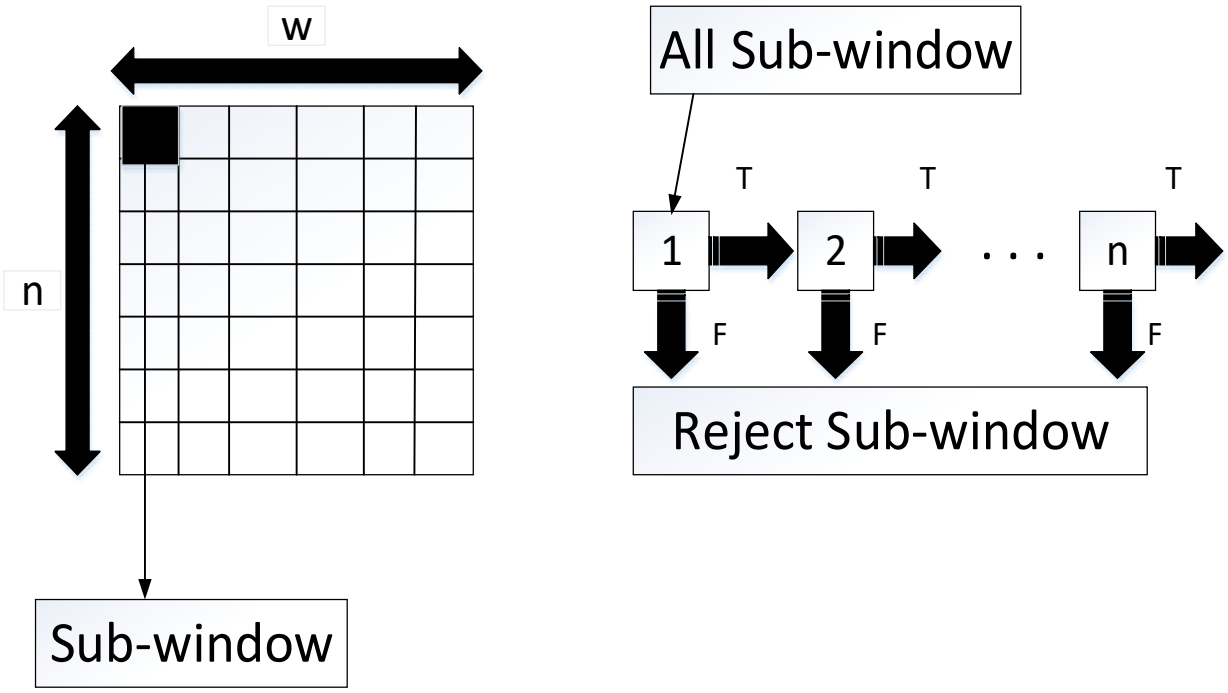



Fig. 2.1. Viola-Jones Detection Cascade Structure.

The first few stages of the classifier only contain the most effective features and omit most of the no-bird areas to reduce computation. Higher stage classifiers utilize many more features to further examine potential areas of interest. Therefore, the sub-window that passes through the last stage classifier is considered to be the region of interest (ROI).

### 2.3.1. The Viola-Jones Detection Algorithm

**Step 1**: Convert image to grey-scale.

**Step 2**: While sub-window is available repeat the following.

**Step 3**: Read next sub-window and resize according to selected scale.

**Step 4**: Apply VJA detector on image sub-window.

**Step 5**: If bird is found, display bird detected on the image.

**Step 6**: Go to **Step 2**.

### 2.3.2. Training Dataset

The training dataset for this work consists of over 500 top down bird images, of on average 48×80 pixels, taken with a single motion activated camera in an outdoor aviary setting. The classifier is trained with blackbirds manually aligned at a 0° angle (i.e., pointing North) with different pose variations, as shown in Figure 2.2. Note that these pose variations include birds with their head tilted or bent, such that they appear slightly off of a 0° alignment.

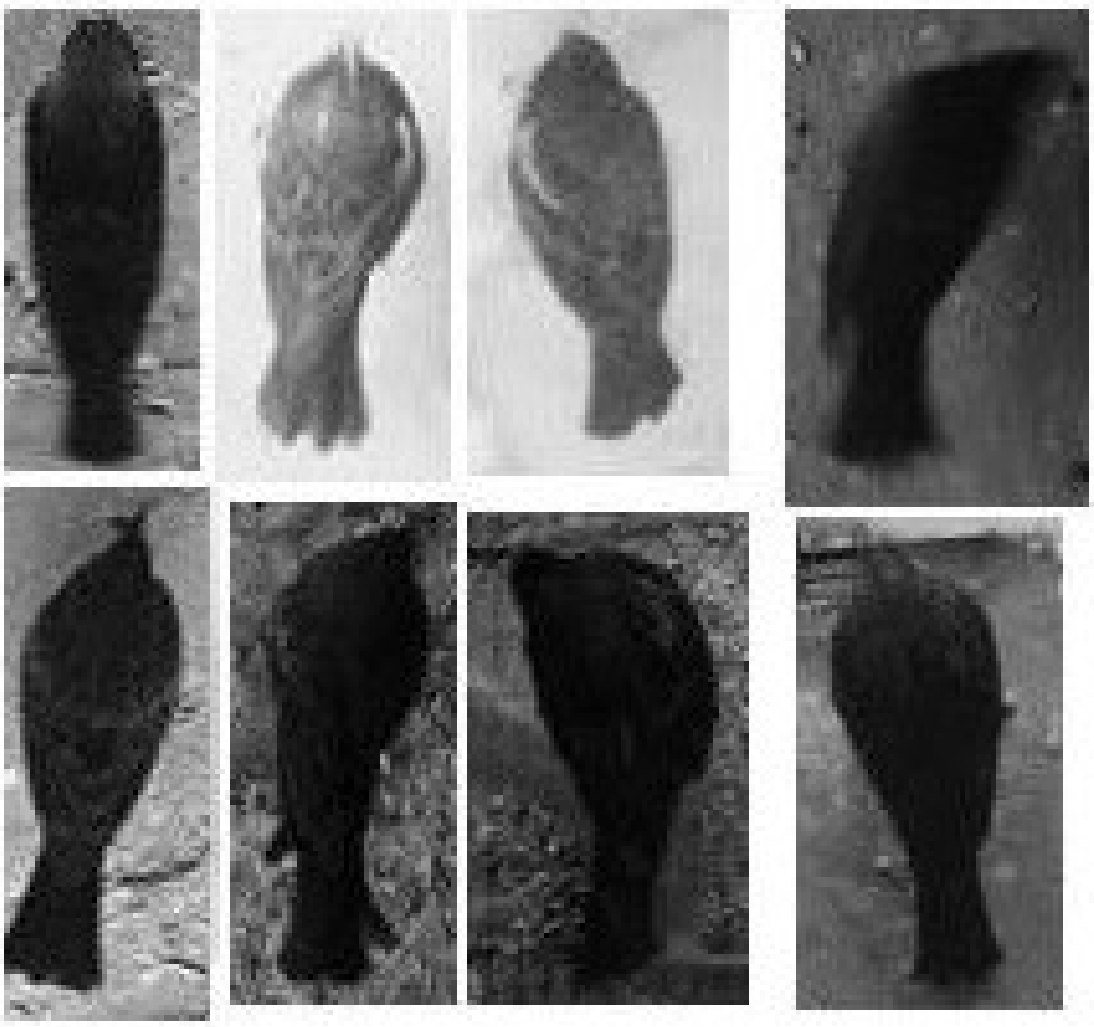


Fig. 2.2. Blackbirds Aligned at 0° Angle with Different Poses.

### 2.3.3. HOG Feature Extraction

The HOG descriptor is employed to train the blackbird classifier, as it has several advantages over alternative descriptors, as explained in Section 2. It captures the key characteristics of the object's local shape by computing gradient orientation and magnitudes [4], as follows.

*Computing gradient:*

Compute centered vertical and horizontal gradients, *Sx* and *Sy*, respectively, where *f'(x)* and *f'(y)* are the derivatives with respect to the x-axis and y-axis.

$$f'(x) = \lim_{h \to 0} \left( \frac{f(x+h) - f(x-h)}{2h} \right) \tag{2.1}$$

*Magnitude:*

$$\mathrm{S} = \mathrm{sqrt}(\mathrm{Sx}^2 + \mathrm{Sy}^2), \text{ where } Sx = f'(x) \text{ and } Sy = f'(y) \tag{2.2}$$

*Orientation:*

$$\theta = \arctan(\mathrm{Sx}/\mathrm{Sy}) \text{ , where } Sx = f'(x) \text{ and } Sy = f'(y) \tag{2.3}$$

For example, a 64×128 image detection window includes approximately 16×16 pixel blocks with 50% overlap, yielding a total of 7•15 = 105 blocks. Each block is divided into 2×2 cells with size of 8×8 pixels. Gradient orientation is quantized into 9 bins, equally spaced between 0° and 180°, with the gradient magnitude linearly interpolated between immediate neighbouring bins [4].

## 2.4. Baseline Algorithm

### 2.4.1. Blackbird Detection Overview

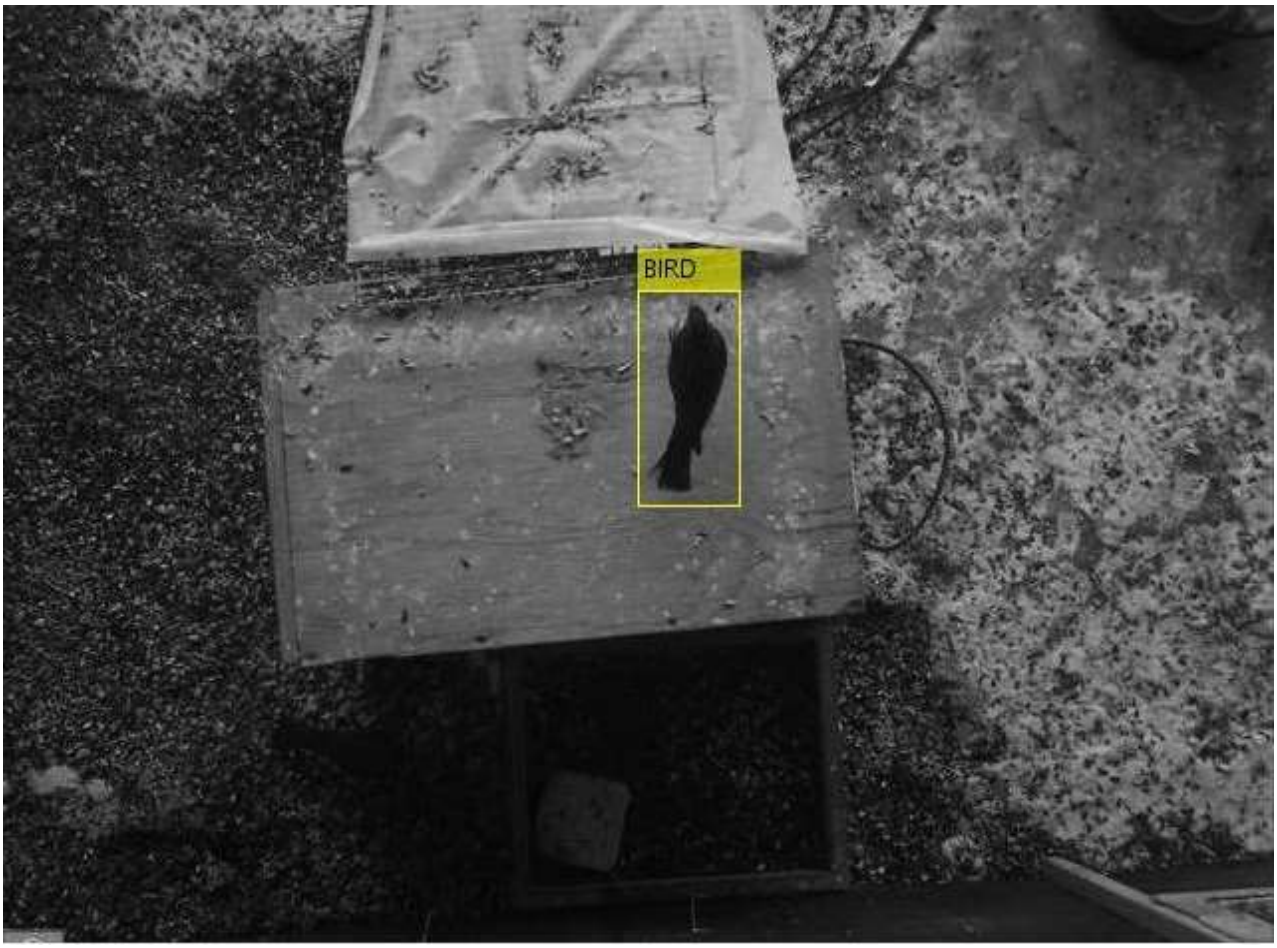
Blackbird detection is complicated by various factors including lighting, variable bird poses, and variable bird orientation (rotation). Inconsistent bird orientation poses cause significant difficulty for the Viola-Jones algorithm, which is not inherently invariant to in-plane

object rotation, as demonstrated in Figure 2.3(b), which shows the VJA results for an image containing two blackbirds, where the algorithm was trained using birds oriented at a 0° angle. The North-oriented bird is successfully identified; a false identification is made in the lower part of the picture; and the rotated bird is not identified.

Viola-Jones is naturally computationally intensive. It is a multi-stage detection method that slides through a large number of sub-images (at various levels of overlap and scale). The number of stages impact computational complexity, detection rate, and false detections. Figures 2.4 and 2.5 illustrate the impact of number of stages in the Viola-Jones algorithm on detection rate and false detections, respectively, for classifiers trained for 4 different rotations: North, East, South, and West, representing birds aligned at 0°, 90°, 180°, and 270°, respectively. These figures were generated by utilizing each trained classifier on the test dataset, which consists of 81 top down images, of on average 460×680 pixels, taken with a single motion activated camera in an outdoor aviary setting, containing 200 unique blackbirds, none of which were also contained in the training dataset. If each classifier could properly identify blackbirds rotated 45° left or right, these 4 classifiers together would cover all 360° of rotation; however, the amount of rotation needed to properly identify blackbirds at all 360° angles is less than 90°, as will be discussed in Section 4.2.

Figure 2.4 shows that detection rate decreases as number of stages increase, since the initial stages of the classifier exclude most of the no-bird areas, while the latter stages further discriminate using finer features. As shown in Figure 2.5, this further discrimination also significantly reduces the number of false detections to close to 0 with 20 stages. However, with 20 stages, the detection rate is also significantly reduced, since more blackbirds rotated further from the training angles (i.e., 0°, 90°, 180°, and 270°) are eliminated. Note that the detection rate

for each of the 4 angles only includes the birds that are oriented within ±45° of the trained angle, since birds rotated more than this are rarely properly detected (e.g., the East (90°) training angle includes birds rotated 45° – 135°). Also note that false detections can occur anywhere in the entire image.

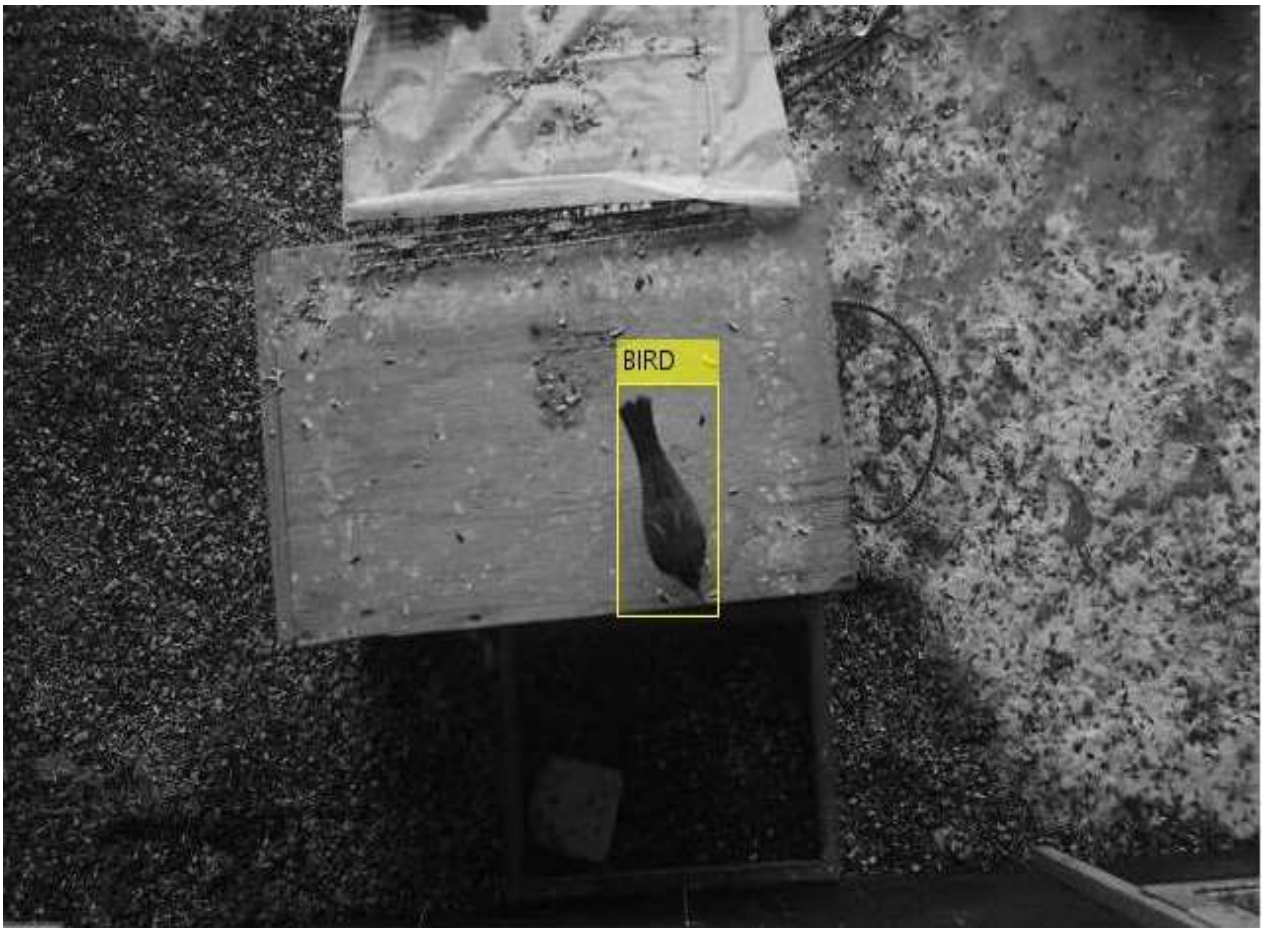



(a) Correctly Detected 0° Aligned Bird

Fig. 2.3. Examples of Viola-Jones based Blackbird Detection.


(b) Correctly Detected 0° Aligned Bird with missed Rotated Bird and falsely Detected Bird, using VJA with no Rotation

Fig. 2.3. Examples of Viola-Jones based Blackbird Detection (continued).




(c) Correctly Detected Rotated Bird using Proposed Algorithm

Fig. 2.3. Examples of Viola-Jones based Blackbird Detection (continued).

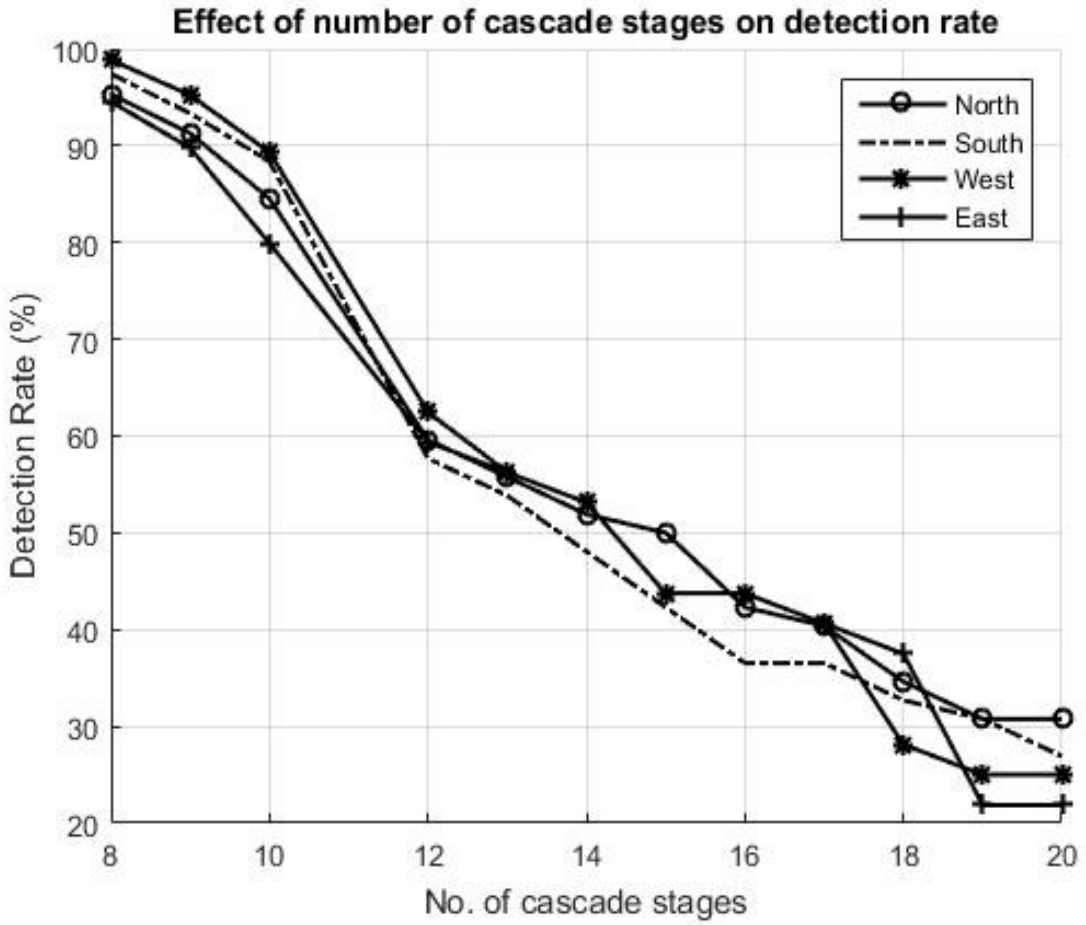



Fig. 2.4. Impact of Number of Stages on Detection Rate.

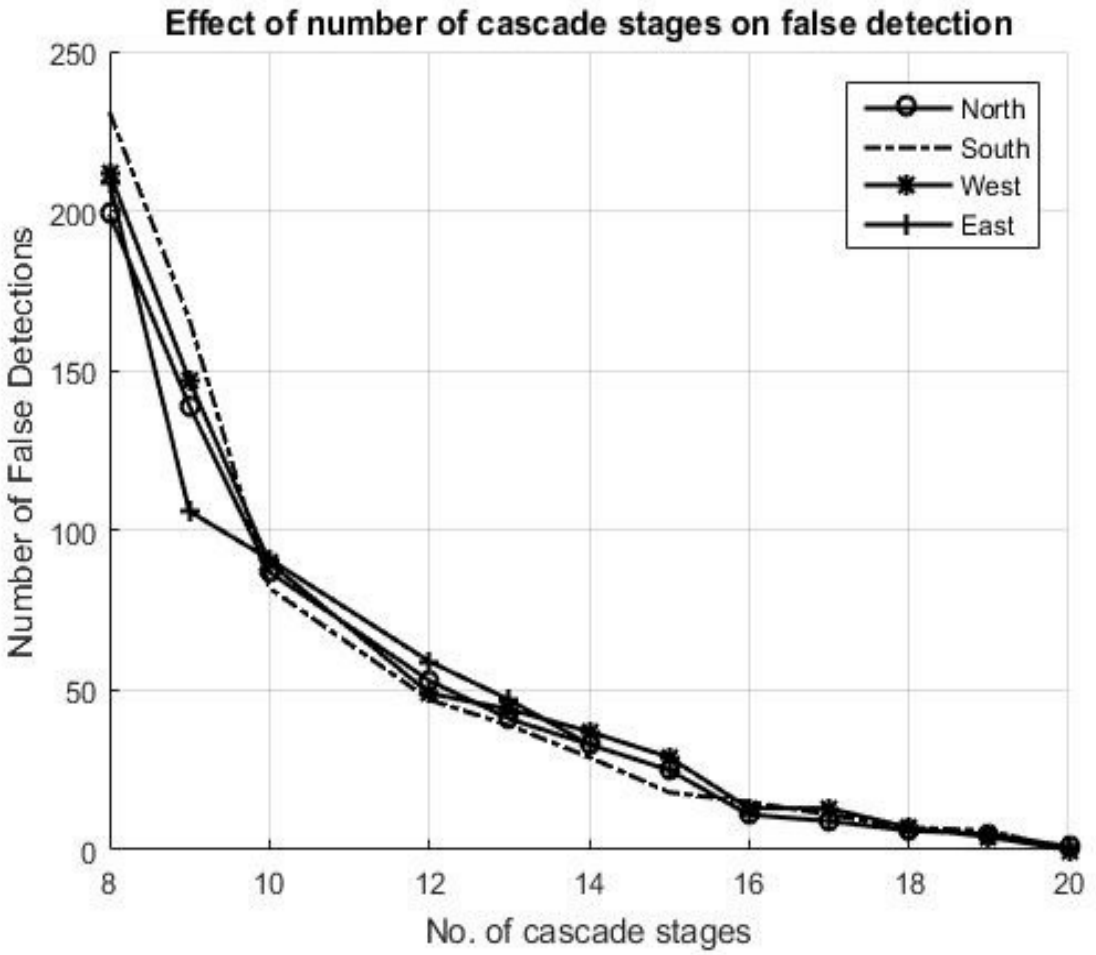



Fig. 2.5. Impact of Number of Stages on False Detection.

### 2.4.2. Rotational Angle Selection

To deal with rotation invariance, previous work includes running the full VJA 360 times on an image rotated 1° each iteration [8, 14], which for blackbird detection achieves a detection rate of 87% with 0 false detections using 20 VJA stages. However, as noted in the literature, rotating by 1° is far too computationally expensive, so a much larger degree of rotation is utilized in practice, as shown in Algorithm 1, typically 30° – 90° depending on the application [7, 8, 14]. Note that when rotating an image, the resulting image is larger than the original, with the extra space being all black, so as to require minimal processing for the additional extra space [6]. To

determine the best angle of rotation to utilize as a baseline for comparison of this work, rotation angle was swept from 1° to 180° using the full 20 stage classifier, to generate Figure 2.6. Note that *N* in Algorithm 1 is calculated as 360°/rotation angle.

**Algorithm 1: Baseline Algorithm**

Step 1: Run VJA detector on non-rotated image.

Step 2: Repeat Steps 3 and 4 N-1 times.

Step 3: Rotate image 360/N degrees clockwise.

Step 4: Run VJA detector on rotated image.

Figure 2.6 compares detection rate and computational cost with rotation degree angle, showing that as rotation degree angle decreases, detection rate increases at the cost of more computation time. Figure 2.6 contains three notable regions: T1, where further decreasing rotation angle only minimally increases detection rate while substantially increasing computation cost; T2, where decreasing rotation angle slightly increases detection rate while slightly increasing computation cost; and T3, where increasing rotation angle significantly reduces detection rate while only slightly decreasing computation cost.

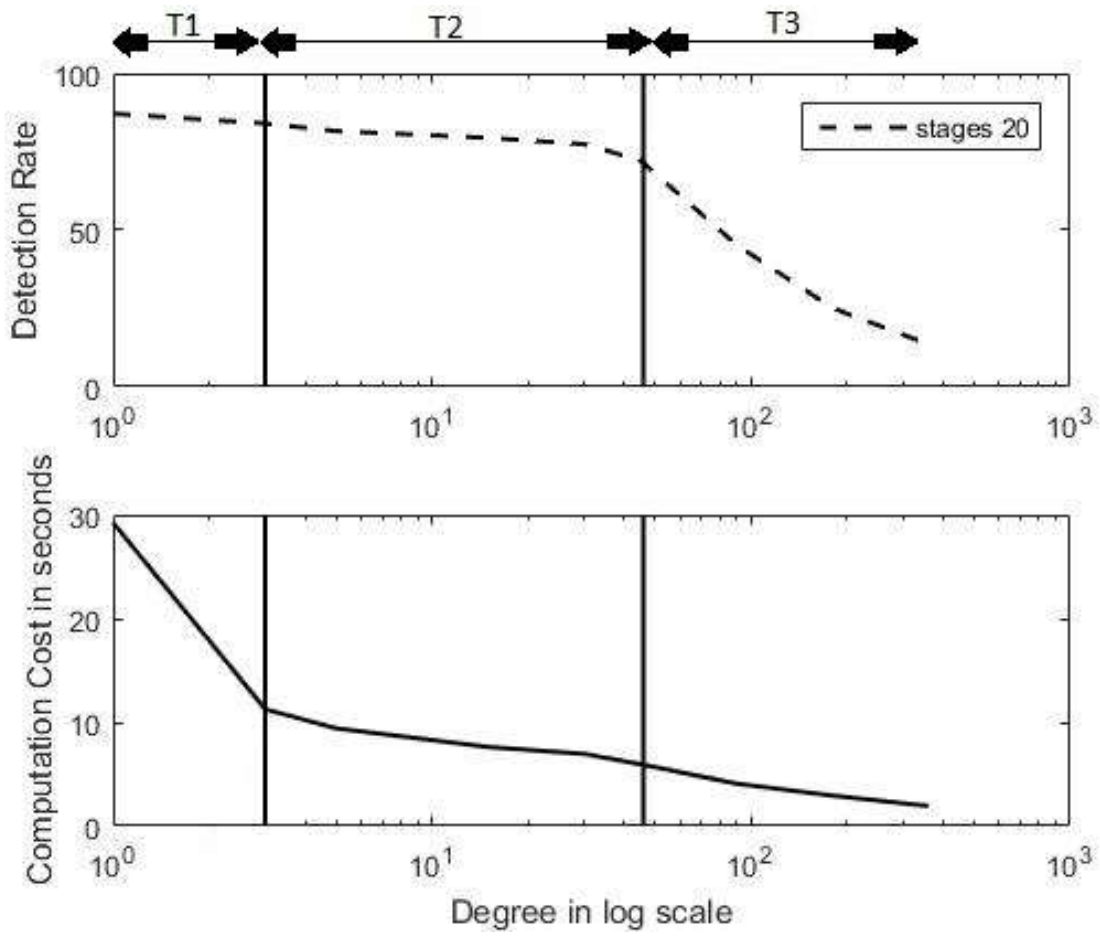



Fig. 2.6. Detection Rate and Computational Cost vs Angle of Rotation.

Figure 2.7 shows the relationship between detection rate and computational cost. In region T2, detection rate can be increased from 73% to 84%, but at the expense of a 2X increase in computation cost (i.e., 5.96 to 11.26 seconds). Therefore, 45°, the boundary between regions T2 and T3 was chosen as the rotational angle for the baseline algorithm for which to compare this work, as it reduces computational cost without significantly reducing detection rate.

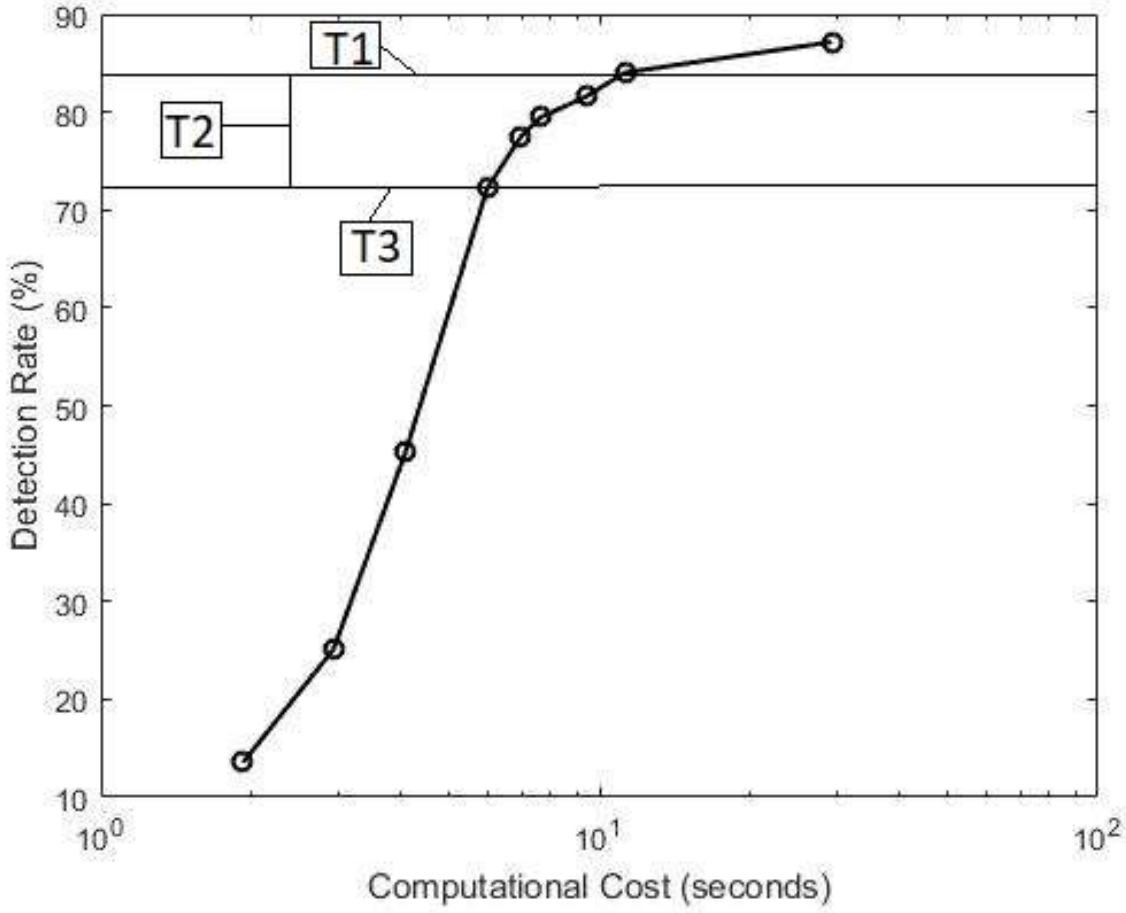


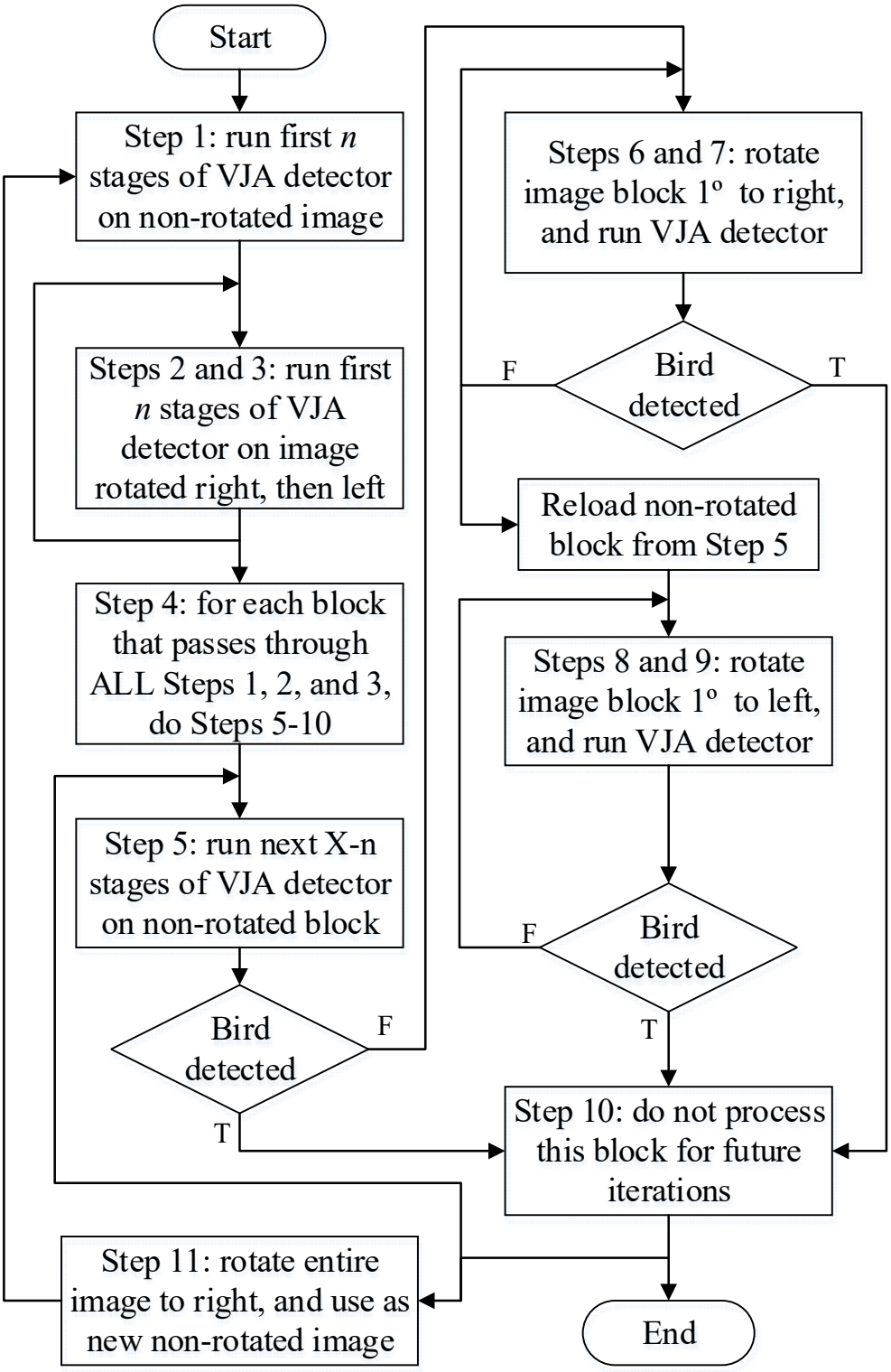

Fig. 2.7. Detection Rate vs. Computational Cost.

**2.5. Proposed Algorithm**

Since it is computationally expensive to apply the full VJA detector to the entire image at all rotated angles, and since the initial VJA stages eliminate a large number of negative (non-bird) sub-window blocks with very little processing time, the full VJA detector can instead be applied only to selected blocks instead of the entire rotated image each time, to still cover all 360° angles while requiring much less computation. Additionally, more negative sub-window blocks can be effectively eliminated in the early stages by running the initial VJA stages on the image rotated at various adjacent angles (referred to as Splits, which must be an odd number), and eliminating all blocks that were not independently identified by ALL of the Splits.

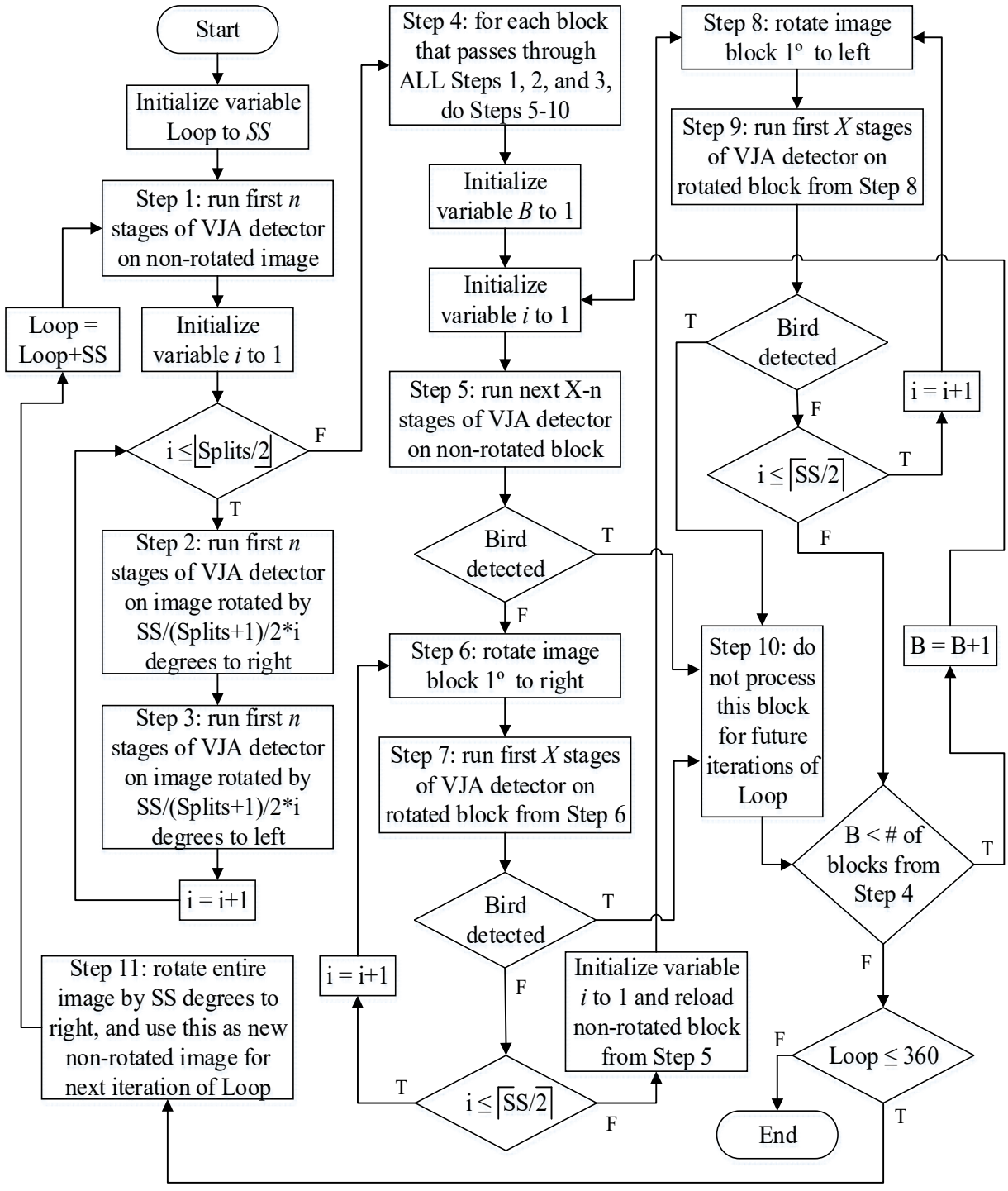
Parameters such as number of initial VJA stages, total number of VJA stages, Segment Size, and Splits are further investigated.

The proposed algorithm for a high-performance rotation tolerant blackbird detector is depicted in Figure 2.8, where (a) overviews the main algorithm steps and (b) includes all algorithm details. First, a loop variable is initialized to Segment Size, *SS*, (e.g., 45°), which is utilized to apply the algorithm to the image rotated by all multiples of *SS* to cover all 360° of rotation (e.g., 0°, 45°, 90°, 135°, 180°, 225°, 270°, and 315° for SS = 45°). Steps 1 – 3 then run only the first *n* stages of the VJA detector on the non-rotated image and any Splits for the current rotation angle (e.g., for the initial current rotation angle of 0° with SS = 45° and Splits = 3, the first *n* stages of the VJA detector would be run on the image rotated by 337.5°, 0°, and 22.5°). Next, Steps 5 – 10 run *X* stages of the VJA detector only on the blocks that were identified as potentially containing a bird by ALL Splits in Step 4, rotated by 1° each iteration to cover all angles within the current segment rotation (e.g., for the initial current rotation angle of 0° with SS = 45° and Splits = 3, a block would only be further processed by Steps 5 – 10 if it was detected by ALL 337.5°, 0°, and 22.5° Splits. This further processing would consist of running *X* VJA stages on the block rotated at all multiples of 1° within the current rotation angle, which would be 315°, 316°, …, 359°, 0°, 1°, …, 45° for this example). After all blocks detected by all Splits for the current segment rotation have been processed, Step 11 rotates the entire image by *SS* to the right and uses this as the new non-rotated image for the next iteration of Steps 1 – 10. Note that when a bird is detected, the corresponding block is not further processed, as indicated in Step 10, such that any remaining 1° block rotations are skipped; and that block is skipped if detected for another segment rotation.




*(a) high-level algorithm*

Fig. 2.8. Proposed Algorithm.




*(b) detailed algorithm*

Fig. 2.8. Proposed Algorithm (continued).

To determine the optimal number of needed VJA stages, *X*, for the proposed algorithm, the full 20 stages was initially utilized, and was then decreased until a further reduction resulted in false detections, in order to minimize computation time. Simulation shows that X=18 is the optimal value for this application, as it yields the same detection rate with 0 false detections as X=20 for the proposed algorithm, while further reducing *X* results in false detections. Also note that the full 20 stages are required for both the 1° rotation and 45° rotation baseline algorithms to achieve 0 false detections, as using fewer stages for these comparison algorithms yields false detections.

To determine the optimal Segment Size and number of Splits, a number of combinations of *SS* and Splits were simulated, as listed in tables 2.1 and 2.2, which show that a Segment Size of 60° with 7 Splits is the best combination for this particular application, as it yields the maximum 87% detection rate with the least amount of computation time. Further increasing *SS* reduces detection rate, while decreasing or increasing Splits, or decreasing *SS* yields the same maximum detection rate but requires additional computation time.

Table 2.1. Computation Time (sec) of Proposed Algorithm for different Splits and *SS* values, using 9 Initial Stages and 18 Total Stages.

| **Splits** | **Segment Size (degrees)** | | | | | |
|---|---|---|---|---|---|---|
| | 30 | 45 | 60 | 72 | 90 | 120 |
| 1 | 5.33 | 5.25 | 5.15 | 5.09 | 5.03 | 4.89 |
| 3 | 4.56 | 4.61 | 4.59 | 4.51 | 4.38 | 4.33 |
| 5 | 4.71 | 4.59 | 4.51 | 4.46 | 4.40 | -- |
| 7 | 4.93 | 4.58 | 4.47 | 4.43 | 4.37 | -- |
| 9 | 4.97 | 4.61 | 4.51 | 4.40 | 4.33 | -- |

Table 2.2. Detection Rate (%) of Proposed Algorithm for different Splits and *SS* values, using 9 Initial Stages and 18 Total Stages.

| **Splits** | **Segment Size (degrees)** | | | | | |
|---|---|---|---|---|---|---|
| | 30 | 45 | 60 | 72 | 90 | 120 |
| 1 | 87 | 87 | 87 | 82 | 78 | 59 |
| 3 | 87 | 87 | 87 | 82 | 75 | 55 |
| 5 | 87 | 87 | 87 | 80 | 75 | -- |
| 7 | 87 | 87 | 87 | 80 | 73 | -- |
| 9 | 87 | 87 | 87 | 77 | 68 | -- |

## 2.6. Conclusion

This chapter develops a high-performance rotation tolerant VJA-based algorithm to detect rotated blackbirds by innovatively covering the entire 360° rotation, but only for select blocks identified by an initial number of VJA stages as potentially containing a blackbird. As shown in Table 2.3, the proposed algorithm achieves an equivalent detection rate (87% with 0 false detections) to running the full VJA 360 times on an image rotated 1° each iteration, while also decreasing computation time by 25% compared to the baseline algorithm that rotates the image 45° each iteration to achieve a detection rate of only 73% with 0 false detections.

Table 2.3. Algorithm Comparison.

| | Algorithm | | |
|---|---|---|---|
| | **1º rotation [8,14]** | **45º rotation [8,14]** | **This Paper SS=60º; Splits=7** |
| **Cascade Stages** | 20 | 20 | 9 initial; 18 total |
| **False Detections** | 0 | 0 | 0 |
| **Detection Rate** | 87% | 73% | 87% |
| **Computation Time (sec)** | 29.1 | 5.96 | 4.47 |

This chapter describes how the optimal values for number of initial VJA stages ($n$), total number of VJA stages ($X$), Segment Size ($SS$), and Splits were determined for the blackbird detection problem discussed herein. However, these optimal values may change for a different problem; hence, one would need to follow the methods described herein to determine the optimal values of $n$, $X$, $SS$, and Splits for each unique application of the proposed algorithm.

# 3. FURTHER SPEEDUP OF A LARGE WORD-WIDTH HIGH-SPEED ASYNCHRONOUS MULTIPLY AND ACCUMULATE UNIT

## 3.1. Introduction

An asynchronous quasi delay-insensitive 72+32×32 Multiply and Accumulate (MAC) unit was designed in [23], utilizing the NULL Convention Logic (NCL) paradigm [24], which was shown to be the fastest asynchronous MAC in the literature. The MAC performs a 32-bit × 32-bit fixed-point fractional multiplication, accepting (signed × signed), (signed × unsigned), and (unsigned × unsigned) $2^s$ complement operands; the product can be added to or subtracted from the 72-bit accumulator, and includes a Multiply Only option. The outputs are the 72-bit $2^s$ complement result along with an *OV* bit that is asserted when an overflow occurs. The resulting design, shown in Fig. 3.1, consists of an 11-stage bit-wise pipelined [25] feed-forward multiplier, followed by bit-wise NULL Cycle Reduced (NCR) [26] accumulate feedback loop, then a 35-stage bit-wise pipelined 70-bit Ripple Carry Adder (RCA), and finally, bit-wise NCR overflow feedback loop, which achieves an average cycle time, $T_{DD}$, of 3.8 ns, using a 1.8V 0.18µm static NCL CMOS library [27].

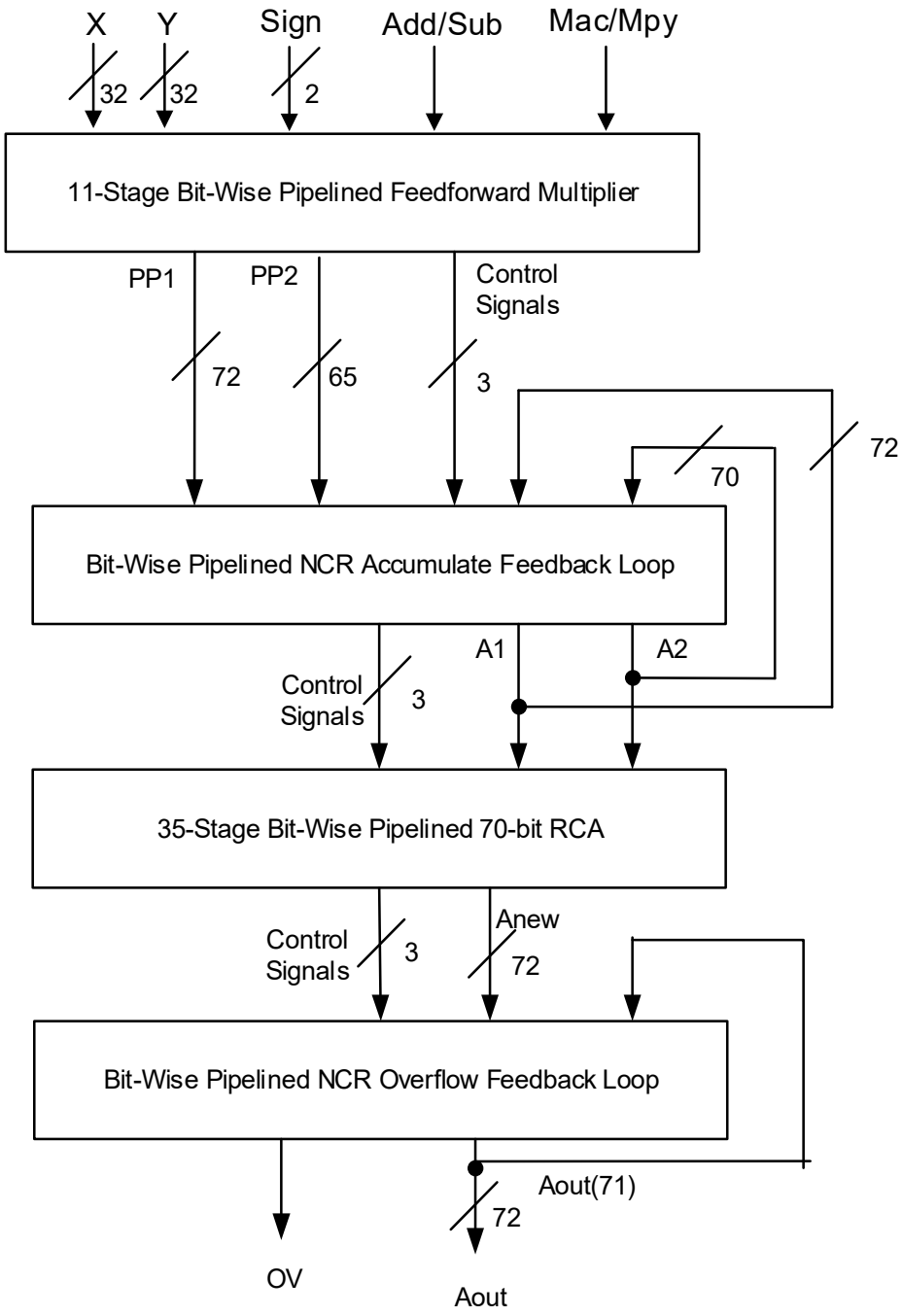



Fig. 3.1. 72+32×32 MAC Designed in [23].

As detailed in [23], the MAC's throughput is limited by the accumulate feedback circuitry, detailed in fig. 3.2, which feeds back the accumulator as two Partial Products (PPs) in carry-save form, *A1* and *A2*, which are zeroed when the Multiply Only option is selected. This feedback loop also contains two CSAs, the first which sums the two PPs from the feed-forward multiplication,

*PP1* and *PP2*, with *A1*, and the second which sums the first CSA output with *A2*, generating the new accumulator value, *A1* and *A2*, in carry-save form. The MAC's throughput was optimized by first maximizing the throughput of the accumulate feedback loop, and then optimizing the feed-forward multiplier, RCA, and overflow feedback loop to each achieve the same, or slightly better, performance as the throughput-limiting accumulate feedback loop, which maximized performance for the entire MAC while minimizing area. Hence, if the accumulate feedback loop throughput could be further increased, the throughput for the entire MAC could then be increased by re-optimizing the other portions.

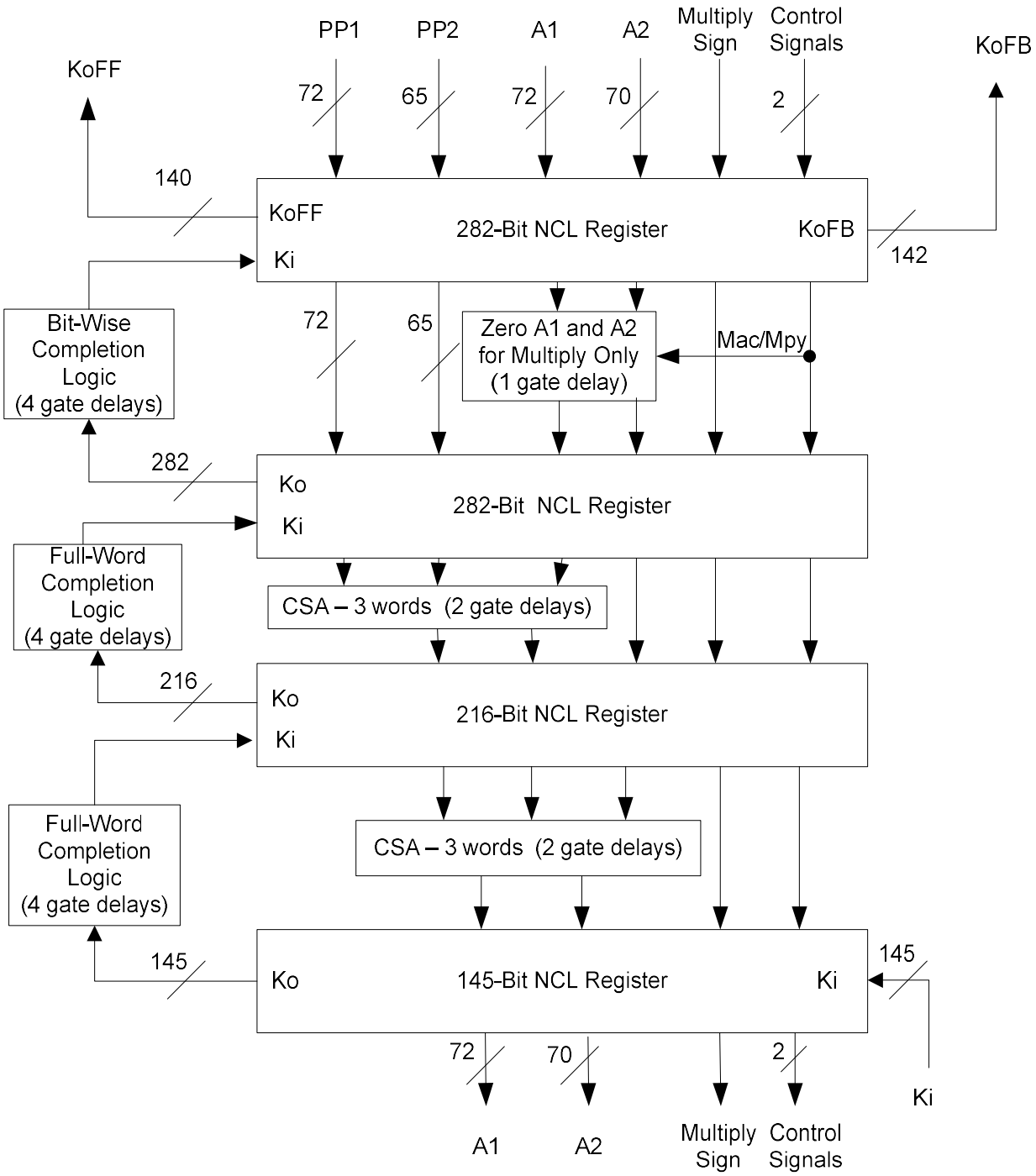



Fig. 3.2. Accumulate Feedback Circuitry Designed in [23].

## 3.2. Overview of NCL

### 3.2.1. Why Asynchronous Design

With the increase of synchronous chip clock rates while feature size has decreased, clock skew has become a major bottleneck for synchronous, clocked architectures. Also power dissipation for high performance chips has increased significantly, especially peak power at the clock edge when switching is most prevalent. Hence, with these inherent inefficiencies emerging as dominant factors hindering increased performance, the clock is becoming more difficult to manage. As a result, asynchronous, clockless digital circuits have been the subject of renewed interest in many companies. Compared to synchronous circuits, asynchronous digital circuits required less power, produce less electro-magnetic interference (EMI), and generate less noise.

### 3.2.2. NCL Gates and NCL Circuits Implementation

As shown in table 3.1 above, there are 27 fundamental gates for NCL circuits, consisting of four or fewer variables.

Table 3.1. 27 Fundamental NCL Gates in [28].

| NCL Gate | Boolean Function |
|---|---|
| TH12 | A + B |
| TH22 | AB |
| TH13 | A + B + C |
| TH23 | AB + AC + BC |
| TH33 | ABC |
| TH23w2 | A + BC |
| TH33w2 | AB + AC |
| TH14 | A + B + C + D |
| TH24 | AB + AC + AD + BC + BD + CD |
| TH34 | ABC + ABD + ACD + BCD |
| TH44 | ABCD |
| TH24w2 | A + BC + BD + CD |
| TH34w2 | AB + AC + AD + BCD |

Table 3.1. 27 Fundamental NCL Gates in [28] (continued).

| NCL Gate | Boolean Function |
|---|---|
| TH44w2 | ABC + ABD + ACD |
| TH34w3 | A + BCD |
| TH44w3 | AB + AC + AD |
| TH24w22 | A + B + CD |
| TH34w22 | AB + AC + AD + BC + BD |
| TH44w22 | AB + ACD + BCD |
| TH54w22 | ABC + ABD |
| TH34w32 | A + BC + BD |
| TH54w32 | AB + ACD |
| TH44w322 | AB + AC + AD + BC |
| TH54w322 | AB + AC + BCD |
| THxor0 | AB + CD |
| THand0 | AB + BC + AD |
| TH24comp | AC + BC + AD + BD |

As shown in figure 3.3, the primary type of threshold gate is the *THmn gate*, where 1 <= m <= n. The output is asserted when at least m of the n inputs are asserted.

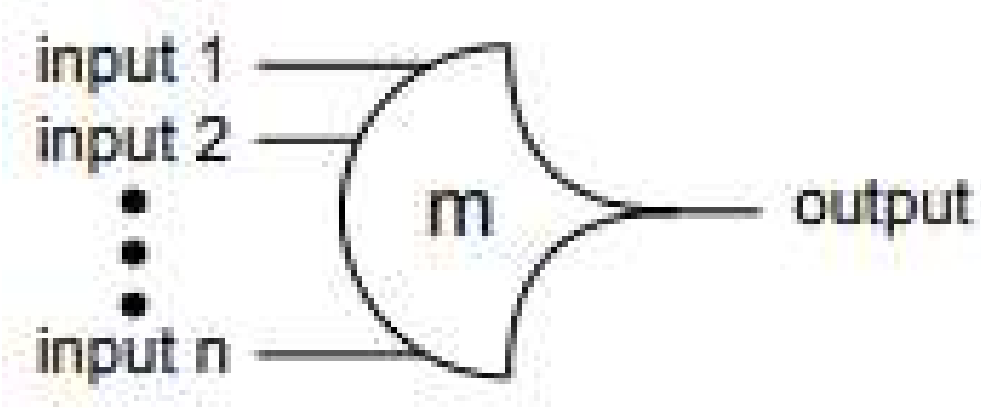



Fig. 3.3. THmn Threshold Gate in [28].

As depicted in figure 3.4, a single bit dual-rail NCL register consists of two TH22 gates that pass a DATA value at the input only when request for data (rfd) (i.e. Ki is logic is 1). Similarly, the TH22 gates pass NULL only when request for null (rfn) (i.e. Ki is logic is 0). Ko is

generated by the NOR gate; when the register output is set to DATA the Ko bit generates rfn, and likewise when the register output is set to NULL the Ko bit generates rfd.

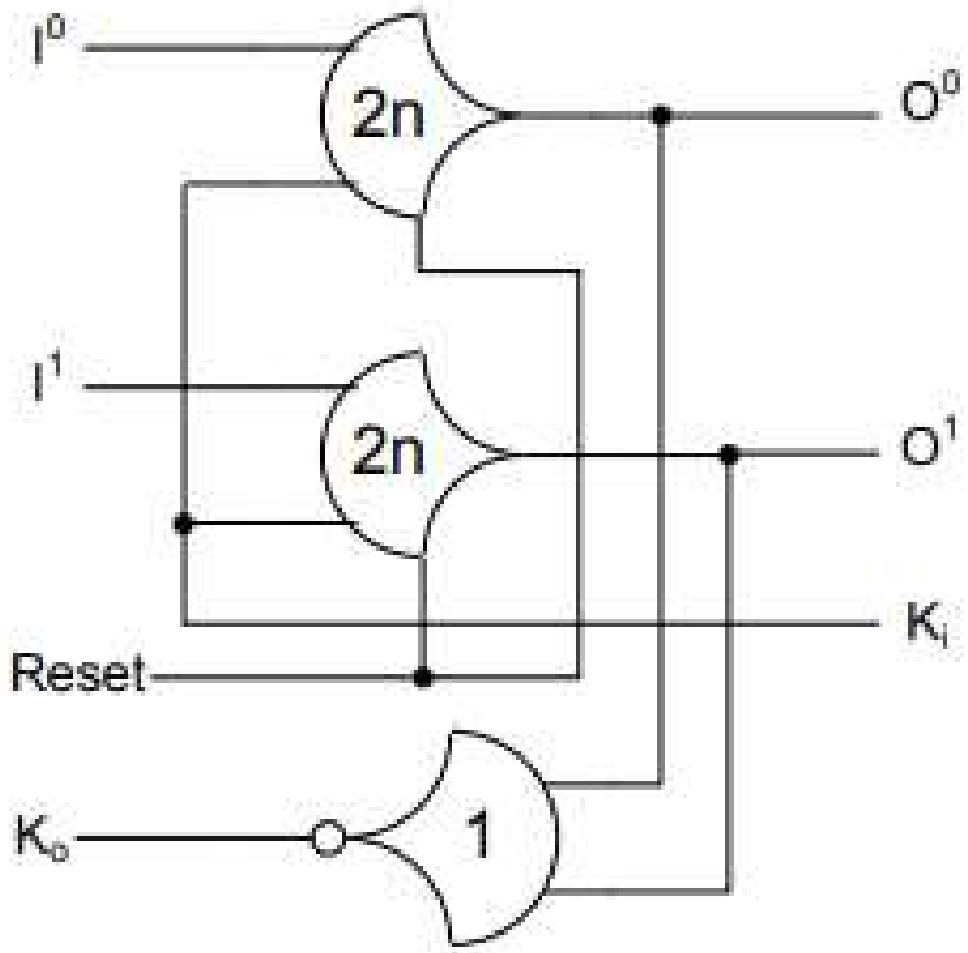



Fig. 3.4. Single bit Dual-Rail NCL Register in [28].

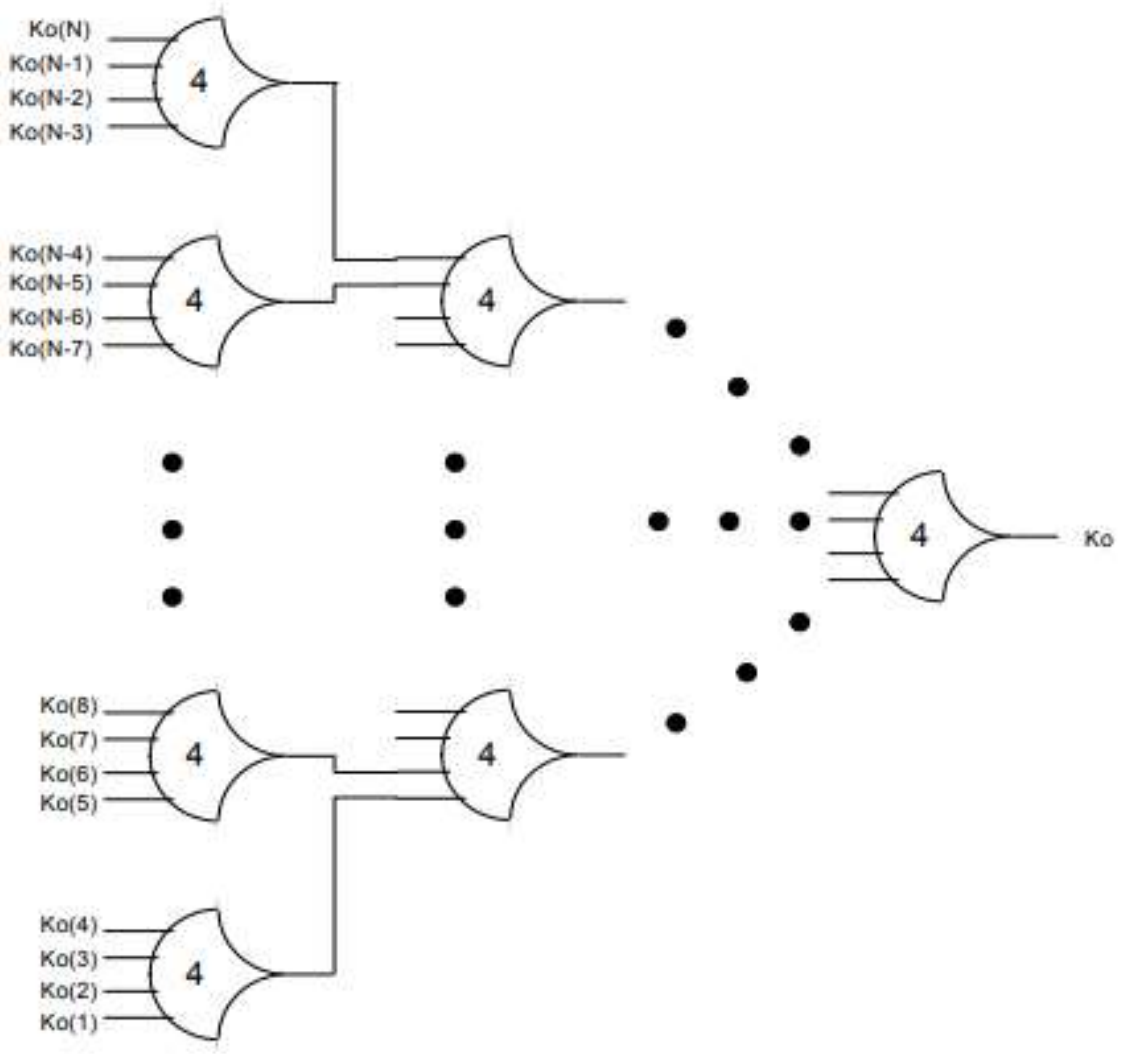



Fig. 3.5. NCL Completion Component in [28].

An N-bit register stage comprised of N signal-bit dual-rail registers requires N completion signals, one for each single-bit dual-rail NCL register. Figure 3.5 represents the NCL completion component, which uses the N Ko lines to detect complete DATA and NULL sets from the output of every register stage and request the subsequent NULL and DATA set, respectively. The number of logic levels in the completion component is given by $\lceil \log_4 N \rceil$ for an N-bit register, since the maximum number of inputs for a threshold gate is 4 (e.g., TH44 gate).

### 3.2.3. Major NCL Components

NCL is a quasi-delay-insensitive (QDI) asynchronous circuit, meaning that regardless of when the inputs become available, the circuit will operate correctly. Therefore, no timing analysis is necessary for correct operation for NCL circuits.

Unlike synchronous system, where inputs are controlled by a global clock signal, input wavefronts in asynchronous system are controlled by local handshaking signals and completion detection. As shown in Figure 3.6, the framework for NCL systems is comprised of QDI combinational logic along with completion units, sandwiched between QDI registers. Two adjacent registers interact through their request and acknowledge signals, Ki and Ko, respectively; and two consecutive DATA wavefronts are always separated by a NULL wavefront, to ensure the current DATA wavefront does not overwrite the previous DATA wavefront.

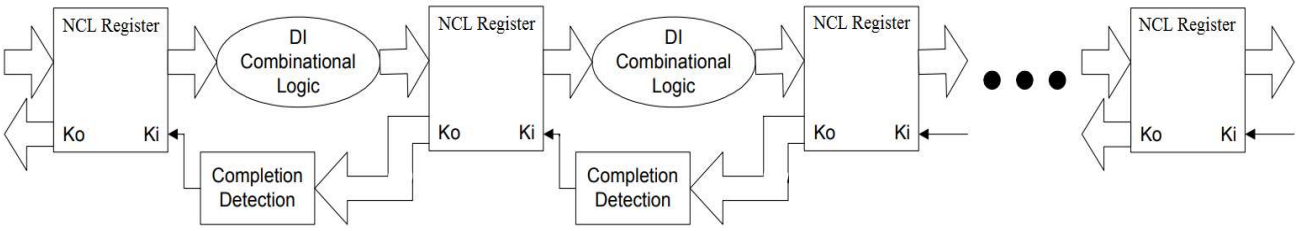



Fig. 3.6. NCL System Framework in [28].

### 3.2.4. NCL Gates and Implementation of NCL Circuits

Like Boolean circuits, the design process of NCL combinational circuits uses a Karnaugh map, and/or utilizes other simplification technique to determine the simplified sum-of-product (SOP) expressions for each output. However, SOP expressions are need for both rails, i.e. **rail$^0$** and **rail$^1$**. An assessment of the output expressions must be made to ensure that the circuit is input-complete and observable. Input-Completeness requires that all outputs of a combinational circuit can not switch from NULL to DATA until all inputs have switched from NULL to DATA, and vice versa. It's acceptable for some of the outputs to switch before all inputs switch, as long as all outputs cannot switch until after all inputs switch. Observability requires that no orphans be allowed to propagate through a gate. An orphan is described as a wire that transitions during the current DATA wavefront, but is not used to determine the output. In order to map the circuit to the 27 NCL gates, the output equations must be partitioned into sets of four or fewer variables.

| X | Y | Ci | Co | S |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Fig. 3.7. Full Adder Truth Table in [28].

Figure 3.7 shows the truth table of a full adder, where *X*, *Y*, and *Ci* denote the first input, second input and carry input, respectively. *S* and *Co* represent the *sum* and *carry* outputs, repectively. As shown in Figure 3.8, the K-map for the *Co* output is given by $Co^0 = X^0Y^0 + Ci^0X^0 + Ci^0Y^0$ and $Co^1 = X^1Y^1 + Ci^1X^1 + Ci^1Y^1$; both directly map to a TH23 gate. Since *Co* is not input-complete with respect to any inputs, *S* must therefore be input-complete with respect to all inputs.

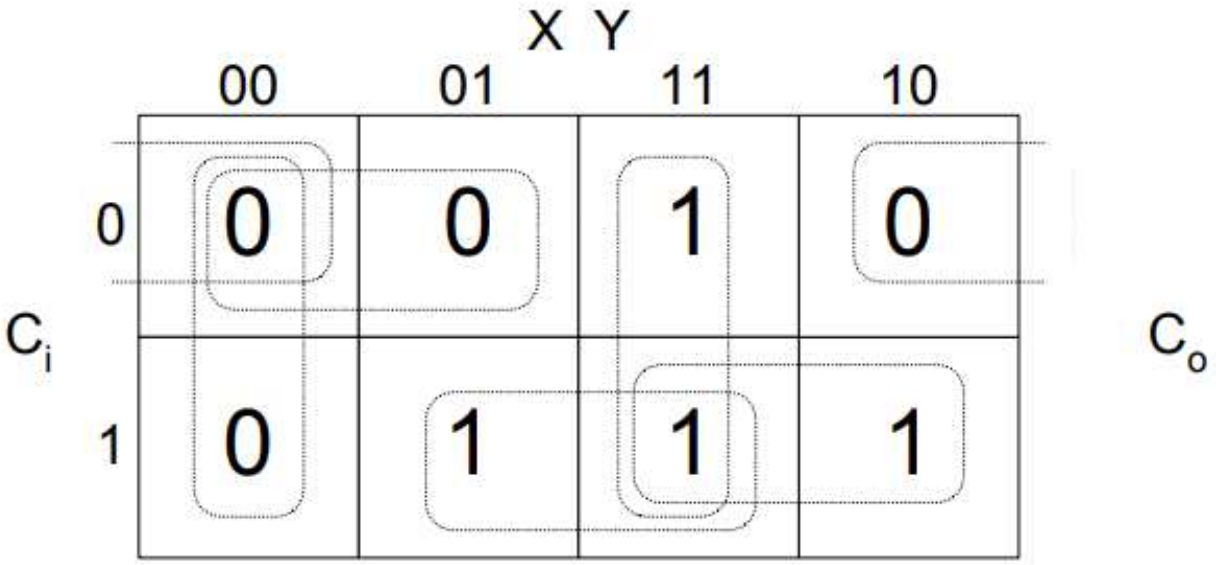



Fig. 3.8. Co Output K-map of Full Adder in [28].

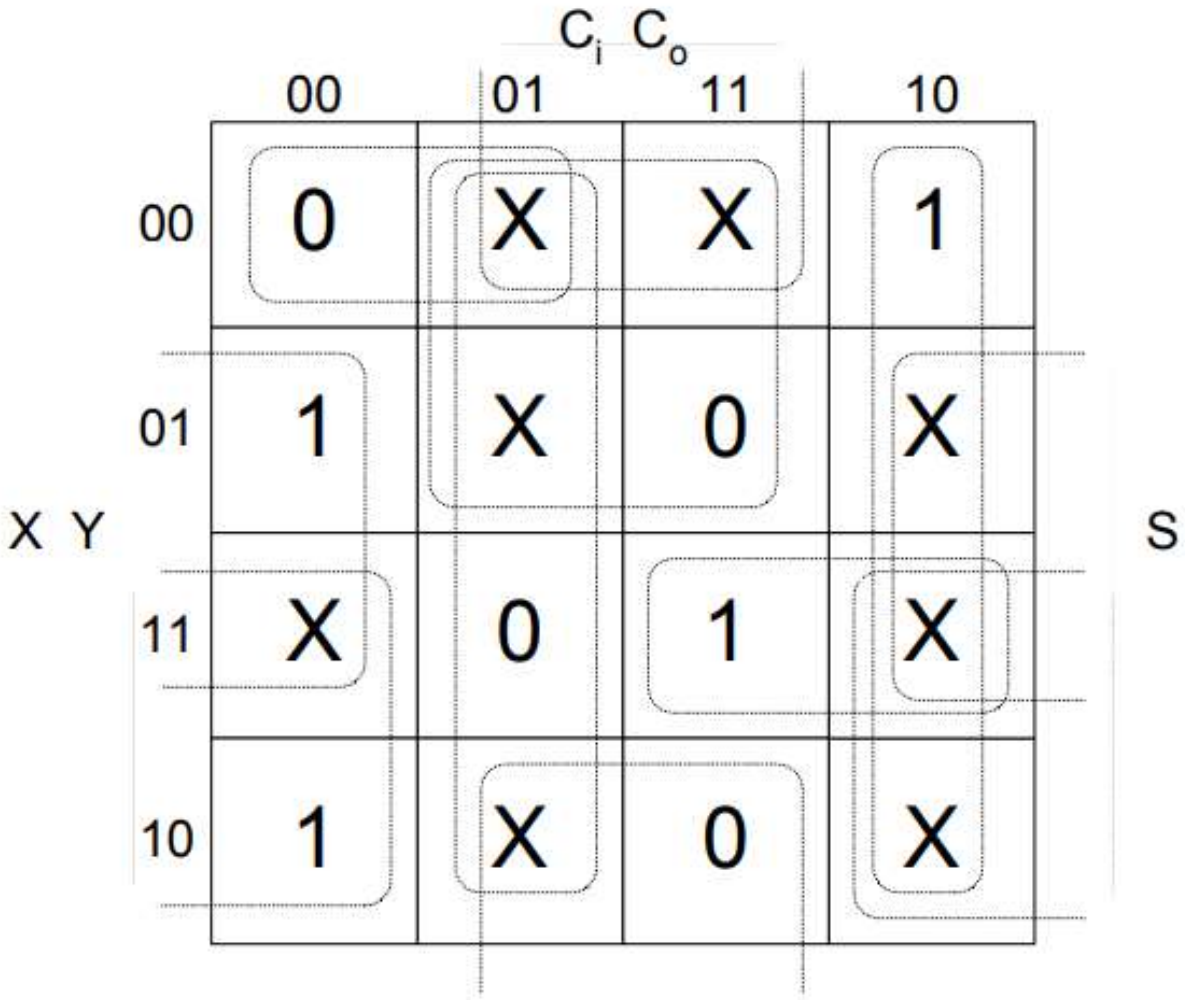



Fig. 3.9. S Output K-map of Full Adder in [28].

As shown in figure 3.9, the K-map for the *S* output is given by $S^0 = Co^1X^0 + Co^1Y^0 + Co^1Ci^0 + X^0Y^0Ci^0$ and $S^1 = Co^0X^1 + Co^0Y^1 + Co^0Ci^1 + X^1Y^1Ci^1$; both directly map to a TH34W2 gate. Since all three inputs are needed to generate the *S* output, *S*, and therefore the circuit as a whole, is input-complete even though *Co* in not input-complete. Figure 3.10, shows the resulting optimized NCL full adder circuit.

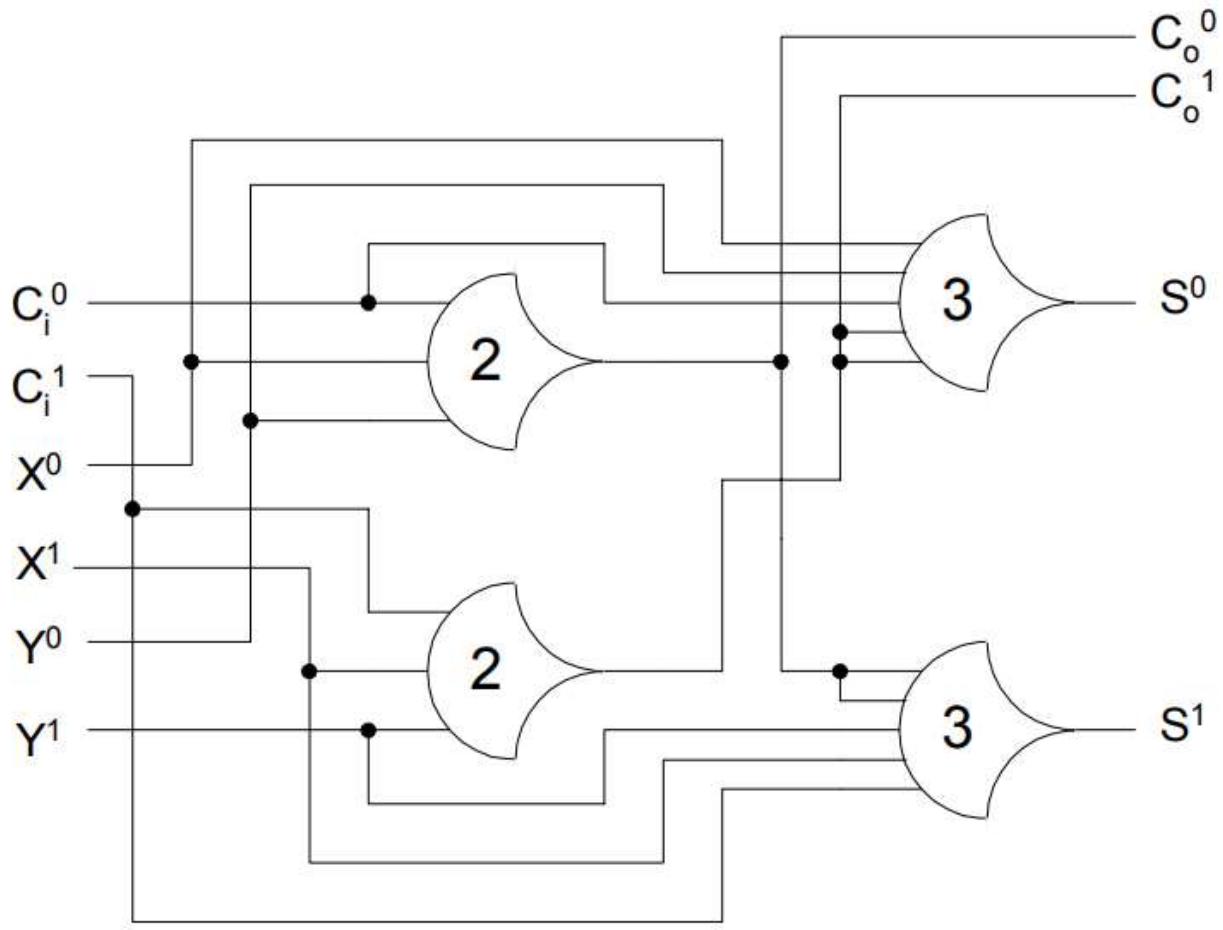



Fig. 3.10. Optimized NCL Full Adder in [28].

### 3.3. Redesigning the Accumulate Feedback Circuitry

This chapter increases the MAC's throughput by utilizing wavefront steering [28] to separate the Multiply Only function from the accumulate feedback loop. As shown in Fig. 3.11, demultiplexers, controlled by *Mac/Mpy*, flow the feedforward PPs, *PP1* and *PP2*, and the feedback accumulator value, *A1* and *A2*, through Accumulate Circuitry only when MAC is selected (i.e., *Mac/Mpy* = 0), and through Accumulate Bypass Circuitry that zeros *A1* and *A2* when Multiply Only is selected (i.e., *Mac/Mpy* = 1). Since *Mac/Mpy* takes part in the generation of all (139+142=281) demultiplexer outputs, this would require a completion delay of 5 gate delays (i.e., $\lceil \log_4 281 \rceil$), which would decrease throughput. Therefore, the *Mac/Mpy* signal is

instead replicated to 18 copies in the preceding Feedforward Multiplier Circuitry, as shown in Fig. 3.14 and explained in Section 4.C. Each of these identical *Mac/Mpy* signals can now be utilized to control up to 16 demultiplexer bits, which allows for a reduced completion delay of 2 gates (i.e., $\lceil \log_4 16 \rceil$) utilizing bit-wise completion. Additionally, note that the *Control Signals* and *Multiply Sign* are no longer needed to calculate *OV*, as will be explained in Section 4.A; hence, they are no longer propagated through the Accumulate Feedback Circuitry.

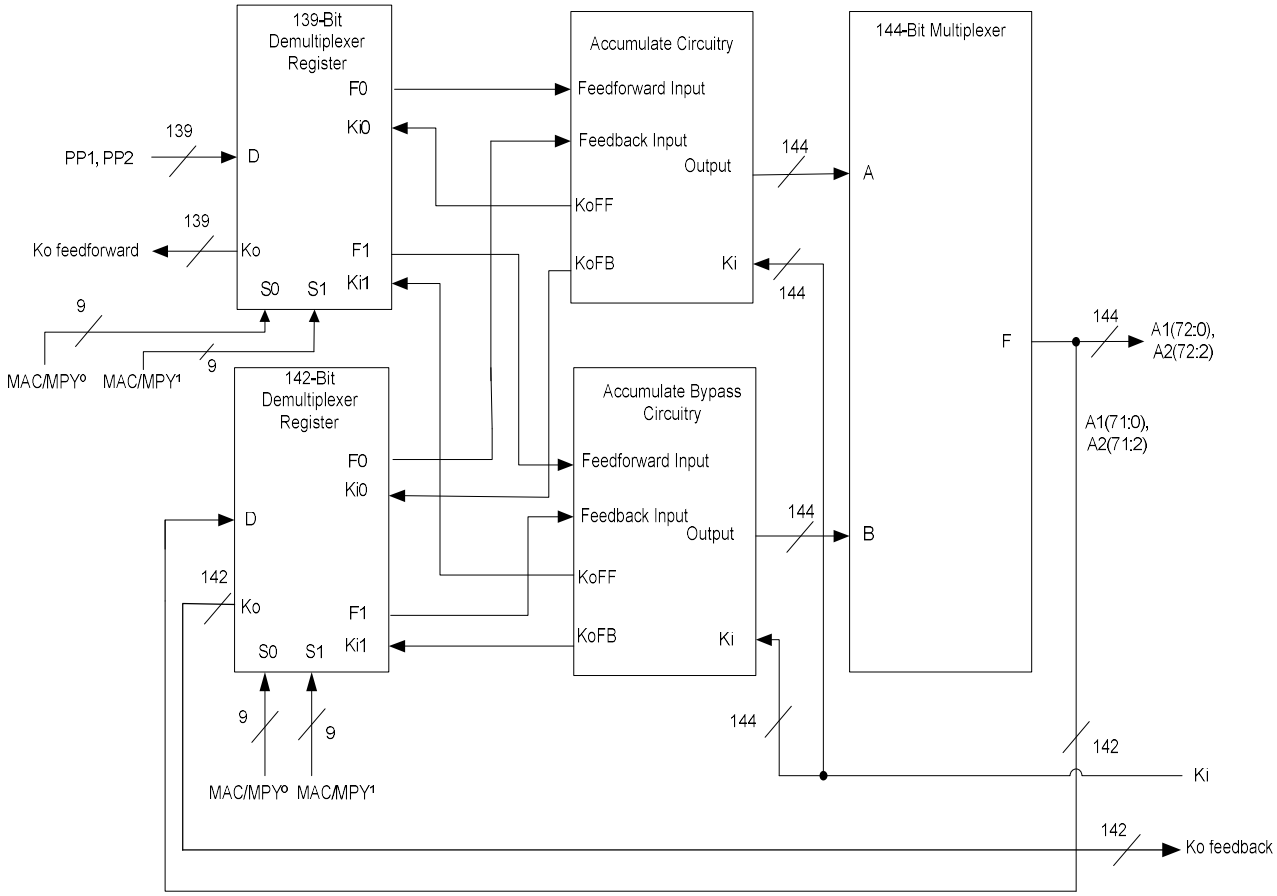



Fig. 3.11. Wavefront Steered Accumulate Feedback Circuitry.

Figure 3.12 shows the redesigned Accumulate Circuitry, where the Multiply Only function and *Control Signals* and *Multiply Sign* have been removed, and bit-wise completion has been utilized, resulting in a 2-stage pipeline vs. the previous 3 stages depicted in Fig. 3.2. The

Accumulate Bypass Circuitry passes *PP1* and *PP2* through to its output to become the new accumulator values, *A1n* and *A2n*, respectively, while maintaining input-completeness with respect to the previous fed back accumulator values, *A1p* and *A2p*, via Eqs. 1-8. *i* has a range of [0, 71], *j* of [0, 1] & [9, 71], and *k* of [2, 8]. Eqs. 1-4 each map to a single TH33w2 gate, while Eq. 5 maps to a TH12 gate to set A2[2:8] to DATA0, since there are no corresponding *PP2* bits for these. Note that the Accumulate Bypass Circuitry also includes an input and output register, such that there are 3 registers in its feedback loop, which are needed to avoid deadlock [28].

$$A1n_i^0 = PP1_i^0 \bullet (A1p_i^0 + A1p_i^1) \tag{3.1}$$

$$A1n_i^1 = PP1_i^1 \bullet (A1p_i^0 + A1p_i^1) \tag{3.2}$$

$$A2n_j^0 = PP2_j^0 \bullet (A2p_j^0 + A2p_j^1) \tag{3.3}$$

$$A2n_j^1 = PP2_j^1 \bullet (A2p_j^0 + A2p_j^1) \tag{3.4}$$

$$A2n_k^0 = A2p_k^0 + A2p_k^1 \tag{3.5}$$

$$A2n_k^1 = 0 \tag{3.6}$$

$$A1n_{72} = PP1_{72} \tag{3.7}$$

$$A2n_{72} = PP2_{72} \tag{3.8}$$

This redesigned Wavefront Steered Accumulate Feedback Circuitry achieves a $T_{DD}$ of 2.8 ns for continuous MAC operations, and a $T_{DD}$ of 1.6 ns for continuous Multiply Only operations; hence, its worst case delay is 36% faster than the previous version in [23]. And since the MAC's throughput is limited by the Accumulate Feedback Circuitry, as explained in Section 1, the throughput for the entire MAC can now be increased by re-optimizing the other portions (i.e., Feedforward Multiplier, RCA, and Overflow Calculation) to achieve the same or faster speed.

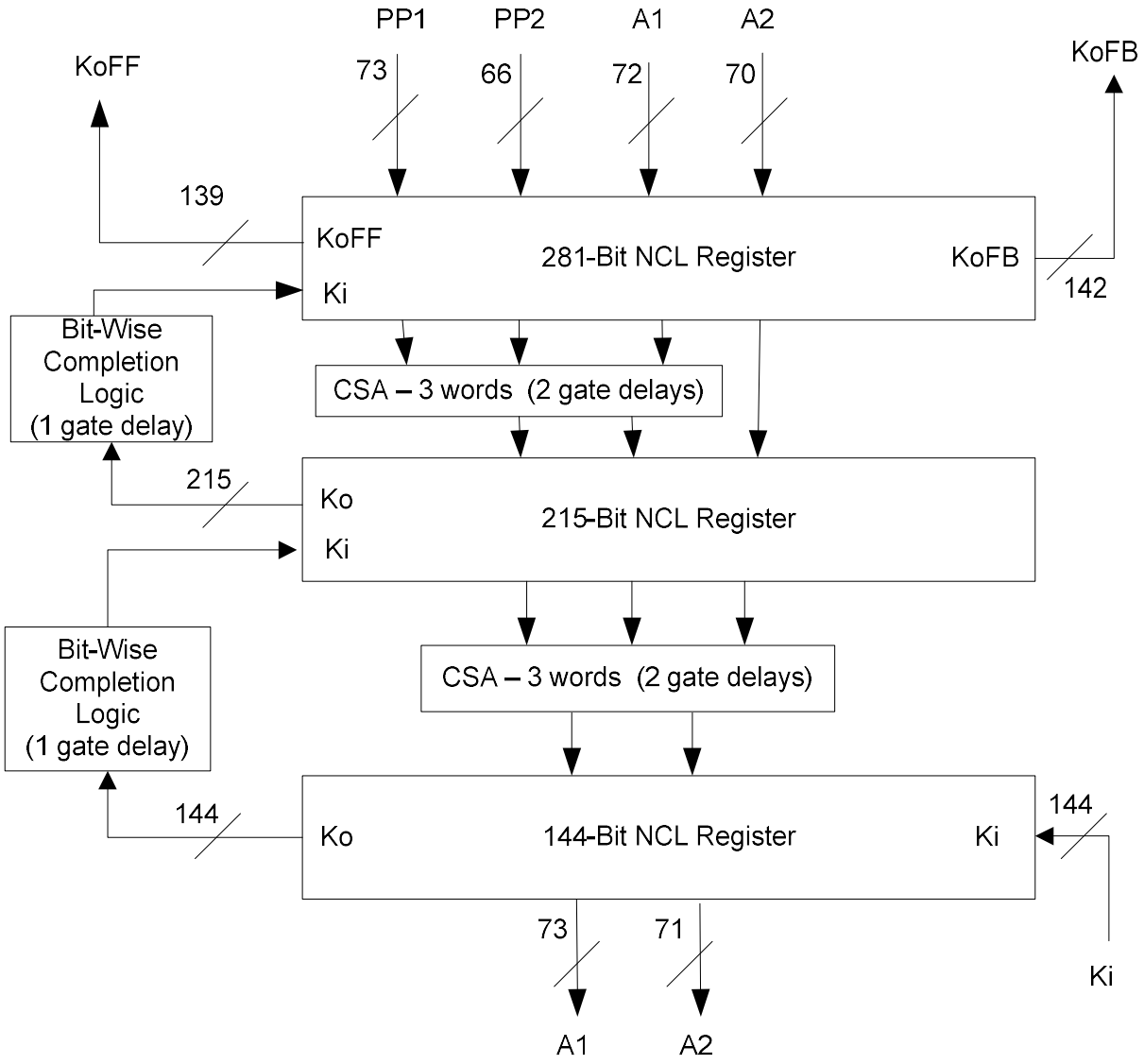



Fig. 3.12. Redesigned Accumulate Circuitry.

### 3.4. Redesigning other MAC Circuitry

This section details the redesign of the RCA, Overflow Calculation, and Feedforward Multiplier, in order to reduce the $T_{DD}$ for each to at least 2.8 ns, the $T_{DD}$ of the throughput-limiting redesigned Wavefront Steered Accumulate Feedback Circuitry, which will maximize throughput for the entire MAC.

### 3.4.1. Overflow Calculation

The previous MAC in [23] calculated *OV* based on the *Add/Sub* and *Mac/Mpy* control signals, along with the current accumulator sign bit and the previous fed back accumulator sign bit. For the redesigned MAC presented herein, *OV* is calculated by sign extending both the 72-bit fed back accumulator and feedforward multiplier inputs to the Wavefront Steered Accumulate Feedback Circuitry, both in carry save form, by 1 bit each, and also increasing the subsequent RCA by 1 additional bit, to produce a 73-bit accumulator value. The least significant 72 bits are then assigned to *Aout*, and OV = Aout(72) XOR Aout(71) (i.e., if the 2 MSBs differ, then the accumulator value cannot be stored in 72 bits, and an overflow has occurred). This overflow calculation method eliminates the need for a feedback loop to calculate *OV*, and instead only requires a few additional adders and 1 additional XOR function.

### 3.4.2. Feed-Forward Ripple Carry Adder

The previous MAC in [23] pipelined the 70-bit RCA into 35 stages of 2-bit RCAs utilizing bit-wise completion, yielding a $T_{DD}$ of 8 gate delays (i.e., 3 gate delays through a 2-bit RCA plus 1 gate delay through its completion logic, then multiplied by 2 to account for both the DATA and NULL wavefronts). The delay of this configuration is greater than the 2.8 ns of the redesigned Wavefront Steered Accumulate Feedback Circuitry; hence, the RCA requires further optimization. The now 71-bit RCA plus the additional XOR function to calculate *OV* could be further pipelined into a single full adder (FA) per stage, which would result in a $T_{DD}$ of 6 gate delays (i.e., 2 gate delays through a FA plus 1 gate delay through its completion logic), which would be sufficient; however, this would require an additional 36 registers on the critical path, which would significantly increase latency. A better alternative is to utilize NCR to speedup the RCA. With NCR applied to the entire RCA with additional *OV* XOR function, only 18 stages are

required, each consisting of 4 FAs, with the last stage instead containing 3 FAs plus the XOR function to calculate *OV*. The worst case stage delay for this configuration is comprised of 5 gate delays for the combinational logic plus 2 gate delays for the completion logic (i.e., $\lceil \log_4 5 \rceil$), since a 4-bit RCA has 5 outputs, a Carry plus 4 Sum bits); however, when combining these they can be overlapped by utilizing the least significant 3 Sum bits for the 1st level of completion logic (i.e., TH33 gate) and the most significant Sum bit and Carry output, along with the output of the 1st TH33 gate, for the 2nd level of completion logic (i.e., TH33 gate), as shown in figure 3.13. This reduces the $T_{DD}$ from 14 gate delays (i.e., (5+2)*2) to only 12 gate delays (i.e., (5+1)*2), which is sufficient to meet the new 2.8 ns $T_{DD}$ goal, after applying NCR.

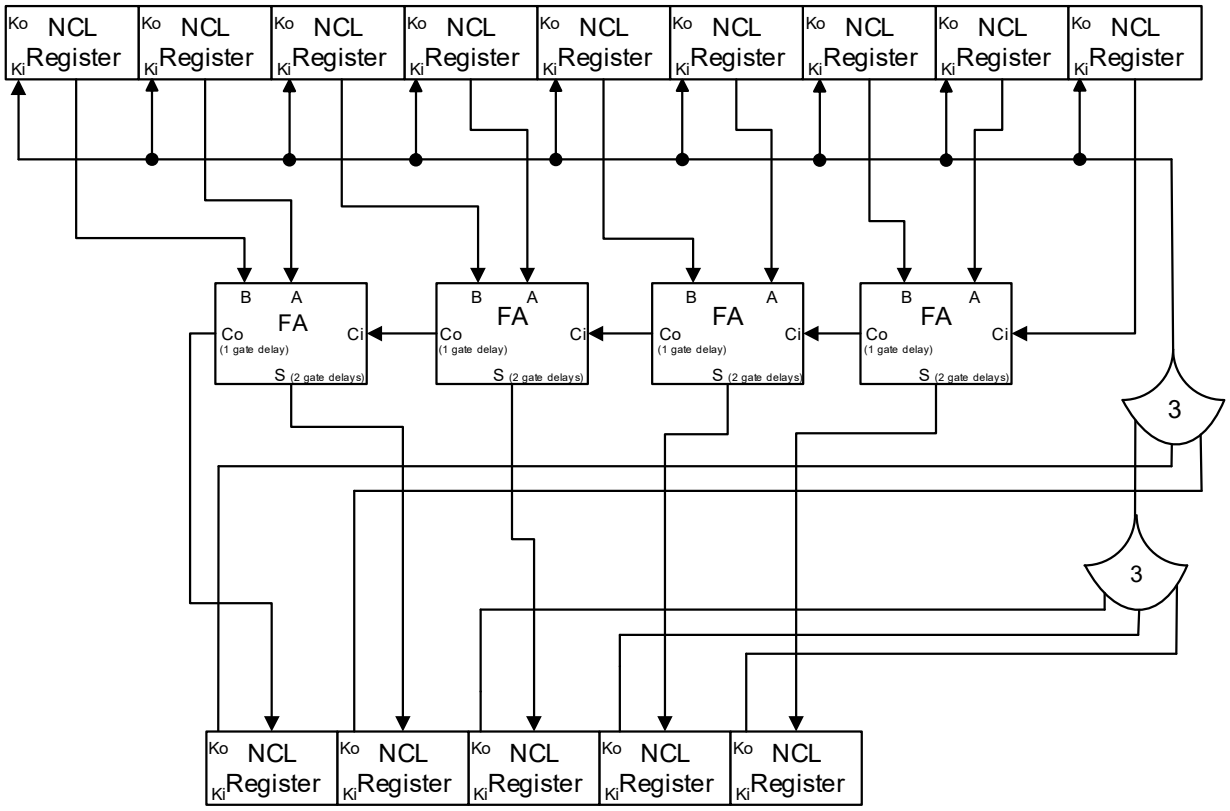



Fig. 3.13. RCA Bit-Wise Completion Logic.

### 3.4.3. Feed-Forward Multiplier

The previous Feedforward Multiplier in [23] consisted of a full-word NCR PP generation first stage, followed by 8 stages of bit-wise pipelined CSAs, then a full-word NCR $2^s$ Complement and Shift PPs stage, and one final bit-wise pipelined CSA stage. $T_{DD}$ for the 9 CSA stages was 6 gate delays, which is sufficient to meet the new 2.8 ns $T_{DD}$ goal; however, the other 2 stages require further optimization. Figure 3.14 shows the redesigned Feedforward Multiplier that meets the required 2.8 ns $T_{DD}$. The first 3 stages of the previous design have been repartitioned into 2 stages, where the 1st stage now includes the first CSA, but not generation of the PP MSBs, which has been moved to the subsequent stage, resulting in a total of 10 stages for this redesign. These modifications reduce the Stage 1 completion delay, such that the 1st stage $T_{DD}$ is now 12 gate delays when utilizing bit-wise completion (i.e., 1 gate delay to generate most PPs, plus 2 gate delays through the first CSA, plus 3 gate delays through the completion logic, since each *X* and *Y* input bit helps generate 31 PP bits in Stage 1, which are subsequently input to at most one CSA FA, such that each *X* or *Y* input bit generates at most 62 Stage 1 outputs, and $\lceil \log_4 62 \rceil = 3$); and the 2nd stage $T_{DD}$ is now 10 gate delays when utilizing bit-wise completion (i.e., 2 gate delays to generate the PP MSBs, plus 3 gate delays through the completion logic, since the MSB of *X* helps generate 32 PP bits in Stage 2, and $\lceil \log_4 32 \rceil = 3$). Since these first two stages were much slower than the CSA stages, bit-wise NCR was applied to these two stages together, to increase throughput.

The other slow stage in the previous Feedforward Multiplier in [23], $2^s$ Complement and Shift PPs, had a $T_{DD}$ of 14 gate delays (i.e., 3 gate delays through the combinational logic plus 4 gate delays through the completion logic, since *Add/Sub* and the 2 *Sign* bits help generate all 141 PP bits, and $\lceil \log_4 141 \rceil = 4$). Applying NCR to this stage was not sufficient to decrease its $T_{DD}$ to

the required 2.8 ns; hence, another optimization was needed. To reduce the completion delay, *Add/Sub* and the 2 *Sign* bits were each replicated 2 times in the previous stage, such that each of the 3 copies generated fewer than 64 PP bits, reducing the stage's completion logic to 3 gate delays when utilizing bit-wise completion, such that the overall $T_{DD}$ for the stage was reduced to 12 gate delays. Now bit-wise NCR can be utilized to sufficiently speedup this slow stage. Also note that the *Mac/Mpy* signal is replicated to 18 copies in the last stage of the Revised Feedforward Multiplier Circuitry to similarly reduce the completion delay for the subsequent Wavefront Steered Accumulate Feedback Circuitry, as explained in Section 3.

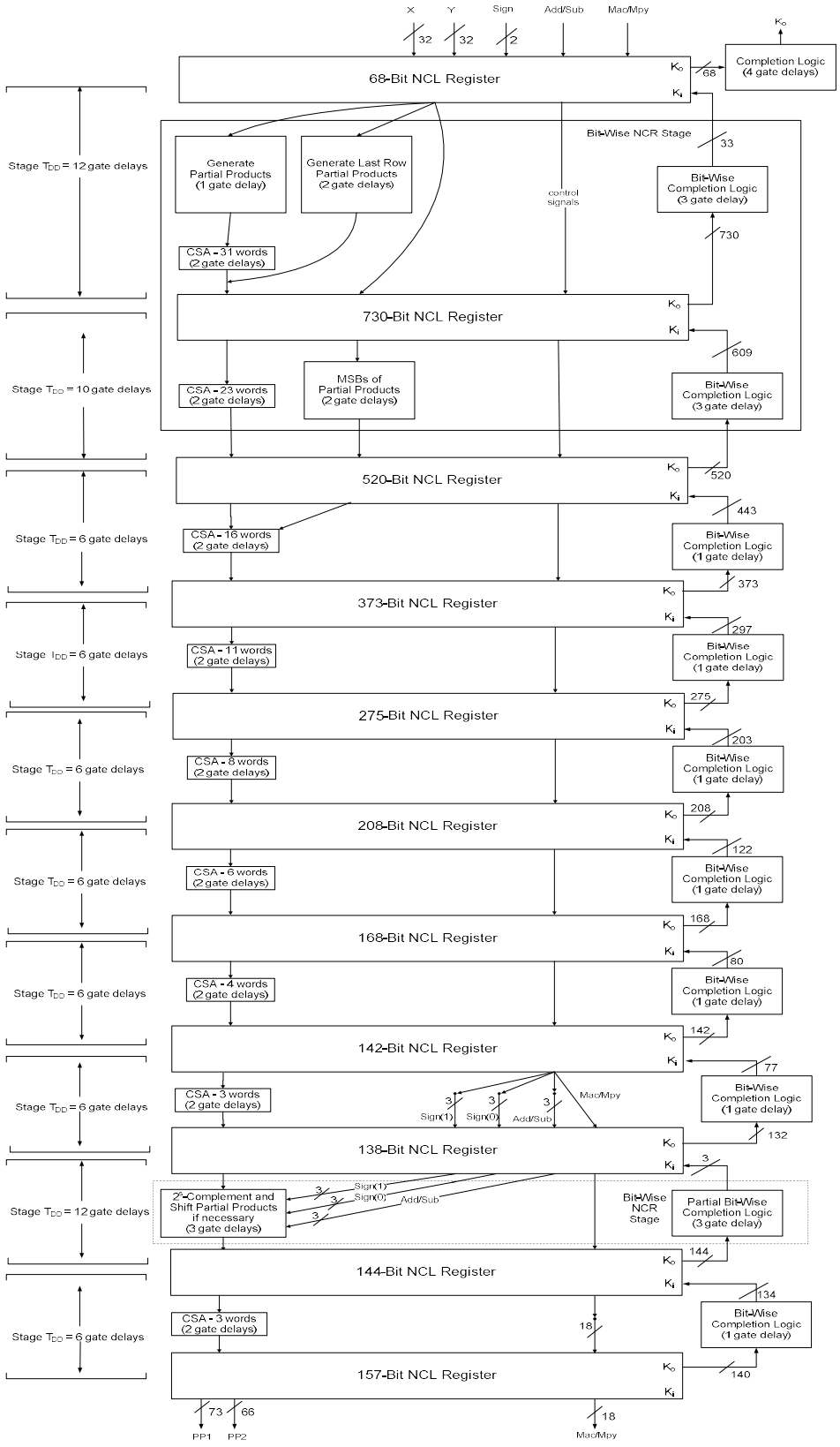



Fig. 3.14. Revised Feedforward Multiplier Circuitry.

### 3.5. Simulation Results

The optimized Feedforward Multiplier Circuitry, Wavefront Steered Accumulate Feedback Circuitry, and RCA were connected together, similar to Fig. 9, and the overall MAC simulated with Mentor Graphics ModelSim, using the same 1.8V 0.18µm static NCL CMOS library and the same testbench as in [23], to compare the overall MAC speedup. The testbench operation is Aout = $\sum_{i=0}^{N}(X_i \times Y_i)$, where $X_i = X_0 + (2^{-21} \times i)$ and $Y_i = Y_0 + (2^{-11} \times i)$, with N chosen to be 255. This allows for a variety of computations to be performed such that any unusually short or long operations will not significantly skew the average cycle time. X0 and Y0 were randomly selected as X0 = A61C039Dh = -0.702270077076 and Y0 = F0046718h = -0.124865639955; and (signed × signed) multiplication was selected. As expected, the outputs were the same as the previous MAC; however, overall $T_{DD}$ was less than the expected 2.8 ns, due to wavefront skew at the interface of the Feedforward Multiplier Circuitry and Wavefront Steered Accumulate Feedback Circuitry. Therefore, 4 additional registers, utilizing bit-wise completion, were inserted between these components to reduce the skew, resulting in the expected overall MAC $T_{DD}$ of 2.8 ns, equivalent to the $T_{DD}$ of the throughput-limiting Wavefront Steered Accumulate Feedback Circuitry. This is a speedup of 1.36 compared to the previous TDD of 3.8 ns for the MAC in [23]. Additionally, area is decreased by 8%, from 75,664 gates for the MAC in [23] to 69,603 gates, due to elimination of a separate Overflow Feedback Loop and a smaller redesigned Accumulate Feedback Circuit that uses wavefront steering in lieu of NCR.

### 3.6. Conclusion

In this chapter the fastest 72+32×32 asynchronous MAC in the literature [23] was redesigned by utilizing wavefront steering for the throughput-limiting accumulate feedback loop, resulting in a speedup of 1.36, while also decreasing area by 8%. Other major contributions

include a new way to calculate OV, which doesn't require a separate feedback loop, as explained in Section 4.A, and the notion of replicating control signals to reduce completion logic for a subsequent stage, as detailed in Sections 3 and 4.C.

# 4. ESTIMATE OF WHEAT LODGING PLOTS FROM DRONE IMAGES[1]

## 4.1. Introduction

Following corn and soybean, wheat [37] ranks as the third most important crop in the U.S. in terms of production, growing areas, and gross farm receipts (USDA, 2019). Beginning in the early 2010s, the U.S. wheat planted areas and yield started to decrease, due to lower returns and increased competition in global wheat market (USDA, 2019). Behind Kansas, North Dakota generally ranks second in U.S. wheat production, with a yield per acre of 37.7 bushels and $5.80/bushel economic value [50].

Crop lodging, defined as the bending over of crops, is one of the most critical issues during U.S. wheat production [31, 48]. Wheat lodging can occur by buckling of the stem or by failure of the root-soil complex [49, 37]. Lodging can lead to lower yield and poor quality, resulting from self-shading, lowered canopy photosynthesis, increased respiration, reduced translocation of nutrients and carbon for grain filling, and high susceptibility to pests and diseases [40, 53]. Weibel and Pendleton (1964), Stapper and Fischer (1990), Berry et al. (2007), and Berry and Spink (2012) reported that wheat lodging would reduce yield by 10% to 50% according to the complex interactions between crop, wind, rain, and soil [32]. In addition, the lodging issue would make the mechanical harvest more difficult [52]. Hence, wheat lodging

[1] The material in this chapter was co-authored by Nauman Jalil and Zhao Zhang. Nauman Jalil had primary responsibility for the handling of drone flight operations to collect field data and conducted experiments for analysis. Nauman Jalil was the primary developer of the color and texture features for SVM analysis and compared results with CNN. Nauman Jalil also drafted and revised all versions of this chapter. Zhao Zhang served as proofreader and checked the math in the statistical analysis conducted by Nauman Jalil.

monitoring over large areas will significantly contribute to yield prediction and loss evaluation [64].

A lot of countries have implemented compensatory measurements for agricultural losses caused by natural disasters [39, 35, 51]. In the U.S., policies follow the USDA Risk Management Agency, which insures farmers' crops to a certain value of production (USDA, Risk Management, 2019) [60]. When wheat lodging occurs, farmers (policyholders) need to walk into the field and visually evaluate the damage, after which they would submit a written notice of damage within a certain time period (48~72 hrs from the initial discovery) (Horvatic, 2019) [41]. Usually, a third party insurance loss adjuster would come to the farm, assess the loss, record measurement, and submit a claim, which finally determines if the farmers gets paid or not based on their insurable conditions [65].

Manual wheat lodging evaluation is laborious, as workers need to walk across a large field area, typically in a high temperature (e.g., 35°C). In addition, manual measurement is also subjective, as each individual worker may come up with different results. Furthermore, considering the inaccurate tools (e.g., tape measure) and accumulation of error during the measurement, the results may be inaccurate. Therefore, an automatic lodging system is needed to replace the current manual method.

To develop a model to distinguish lodging from non-lodging areas, this study first collected visible and multiple spectral aerial images of a field taken with different cameras from drones. The field consisted of 372 wheat plots with different sizes; and three days' data was collected. Next, the images were processed to extract features (e.g., color, texture, NDVI and height) of each plot. Each plot was manually classified into lodging and non-lodging. Random 300 and 72 plots (80% and 20% of the total plots) were used to train and test the Support Vector

Machine model, respectively. For all the three days' data, when one feature was used to train the model, the model accuracy ranged from 66% to 67%. With more features incorporated, the model accuracy increased. When incorporating all four features, the model prediction accuracy is about 90%, indicating that the model worked well to distinguish lodging from non-lodging plots. The model accuracy when using all four factors is not significantly different from using only two factors (i.e., texture and NDVI) for all three days. It is therefore concluded using texture and NDVI features to train a Support Vector Machine model would have similar performance as of using four features. Thus, future researchers could focus on extracting and using texture and NDVI features to train a model for wheat lodging detection instead of extracting and using all four features (e.g., color, texture, NDVI and height).

**4.2. Related Works**

Remote sensing technology, with a quick development over the past years, provides a potential tool to obtain timely information on crop lodging over vast areas of land [46]. To date, three major technologies have been used for crop lodging, including spectral image-based satellite sensing, radar-based optical sensing, and unmanned aerial vehicle (UAV) multiple imagery-based sensing [43]. For satellite images, the spectral features of the stalks/stems (lodged crops) should be distinguished from those of leaves (non-lodged crops), due to different chlorophyll content [45]. Though satellite remote sensing covers huge land plots, its performance on lodging evaluation is weak because of limited spatial and temporal resolutions and spectral band features [44, 46]. In addition, spectral difference supposed to be caused by lodging and non-lodging may be contributed by other factors, such as crop stress (e.g., nitrogen, salinity, and drought) and diseases. Radar-based optical sensing has been tested, but its accuracy on lodging monitoring has not been proven [64]. Considering radar system-based optical sensing is more suitable for

homogeneous and large area, the lodging, which usually occurs in a relative small area, detected by radar is inaccurate [54]. Due to its relatively small area and high resolution image requirements for wheat lodging detection, UAVs, with their rapid development over the past few years, can be considered as a potential tool [42, 46]. Compared to satellite and radar, UAVs have several advantages. On one hand, UAVs can fly relatively low above ground level and instantly capture bird's eye view images with high resolution; on the other hand, a variety of cameras (e.g., thermal, RGB, and multispectral camera) can be customized and attached to UAVs for different kinds of data collection [64]. In addition, based on other advanced techniques of computer vision and digital photogrammetry, UAV images can be used to produce geo-referred orthomosaic map, and digital surface models (DSMs) [38, 57, 63]. The DSMs are helpful in obtaining crop height information.

The Artificial Neural Network (ANN) [74] is among the often used geo-/bio-physical variable retrieval strategies and has been extensively researched in many applications. In Paloscia et al. [75] and Notarnicola et al. [76], the efficiency of neural network model inversion for soil moisture estimation compared to well-known inversion techniques, namely the Bayesian approach and the simplex algorithm, is investigated. Final assessments find out that ANNs are a good trade-off with regard to the other strategies examined in terms of accuracy, stability and computational speed. You may find various relevant examples in the field of extraction of vegetation parameters [77]. Support vector regression (SVR) [78] is another method that has become popular in recent years in the field of geo-/bio-physical parameter retrieval. The efficacy of this method for the retrieval of vegetation characteristics, open water chemical and biological particle concentration, and land and sea surface temperature has been investigated in papers [79, 80]. The results obtained showcase the promising features of this approach, such as the effective

intrinsic generalization capacity and the robustness to noise in the situation of limited supply of the reference samples.

Hyperspectral and multispectral imagery have been shown to have significant potential for real-time surveillance and visualization of metrics of grass quality. Combining multispectral and hyperspectral remote sensing imaging within situ analysis and local knowledge offers a valuable platform for examining the effect of management procedures on grass properties and tracking biomass (BM) and crude protein (CP) variability over time [81, 82]. The application of hyperspectral data will greatly enhance the accuracy of plant-related prediction models [83]. A greater proportion of narrow wavelength bandwidths, comprising a wider spectral range reported by hyperspectral sensors provide a greater chance to estimate grass parameters [84]. Recent advances in airborne and space-borne remote sensing alongside advancements in image quality in terms of spatial, spectral, temporal and radiometric resolution have made it possible to predict more accurately the plant attributes aligned with forage quality [82, 85]. Given the enhanced capacity of remote sensing imagery to assess GQ, the efficacy of these methods to accurately predict overground grass BM and CP as essential indicators of GQ remains uncertain, and few reports have been published on "grass-based food security" [82]. A detailed study on the efficiency of multispectral and hyperspectral data collected using various remote sensing methods (satellite, UAV, and fixed ground-based platform imagery) is of significant interest in supporting to classify appropriate wavelengths, reliable modeling methods and important spectral indices for estimating and plotting fresh GQ indicators.

(GIS) incorporates location data with location information, both quantitative and qualitative, allowing you to view, analyze and publish information via maps and charts. You can answer questions using the application, perform what-if scenarios and visualize outcomes. GIS is

defined as a system used as necessary for the management of infrastructure properties, natural resources and any artifacts. Assessment and management of the facilities and asset data stored in GIS is simpler, making planning, construction and maintenance more efficient and profitable [86].

GIS is the best technique to identify the potential land for any particular crop, as it brings all the data to analysis on a single platform. Various vegetation indexes, such as NDVI, FPAR, and TVI. are commonly used for controlling crop health and are also directly proportional to yield. To track crop health, different factors such as temperature, irrigation facilities and the most critical soil health condition come into play in its growth and development. GIS ' ability to study and analyze agricultural environments and workflows has proved to be helpful to those working in the agricultural sector. Although natural inputs in farming cannot be measured, GIS applications e.g. crop yield estimates, soil modification analysis, erosion identification and remediation can indeed be better understood and managed [86].

UAV platforms are becoming a successful approach in agricultural reliability assessments as they facilitate the non-destructive measuring of crop development status with a very high spatio-temporal resolution [87, 88]. Even though suitable weather-based image exposure can be set, data can be collected under sunny as well as overcast conditions [89]. For researchers and farmers to track crop growth status, color images can thus be acquired instantly [90]. Hunt et al.,[91] estimate the corn, alfalfa, and soybean biomass from RGB images using the normalized green-red difference index (NGRDI) and establish a linear correlation between NGRDI and biomass. To estimate chlorophyll content, Kawashima and Nakatani [92] used a video camera to analyze the color of wheat leaves. Woebbecke et al. [93] studied the ability of several color indices to differentiate vegetation from the background and noticed that the excess green vegetation index (ExG) may provide a picture of near-binary strength that outlines an area of

interest. In addition, color indices that include a significant amount of crop status information from RGB cameras could also be used to determine the vegetation fraction, plant height, biomass and yield [94-96]. Earlier studies have shown that estimating crop growth status with RGB and CIR images is achievable, but few studies have evaluated their effectiveness for checking the N status [88,97]. First, the ability of digital cameras to track LNC wheat at various stages of growth is still poorly defined. Because the application of N fertilizer has increased recently in China, it is important to effectively monitor LNC under middle to high application levels. Therefore, the capacity of digital cameras to estimate wheat LNC under different conditions also needs to be evaluated. The ultimate goal is to determine whether UAV-mounted digital cameras can be used to track LNC in winter wheat.

UAVs have been used in crop lodging detection, such as rice, corn, and canola [44, 48]. Some studies have been conducted using UAVs in wheat lodging detection; however, they all have some limitations. Li et al. (2014) and Du and Noguchi (2017) only used color information for lodging detection, but did not use the DSMs. Liu et al. (2017) fused color and temperature information for wheat lodging detection, but did not incorporate DSM information. Few studies have been identified that incorporate color information, spectral data, and DSMs to develop an automatic wheat lodging detection method. To fill this gap, the objectives of this study are to (1) extract color, texture, and NDVI (Normalized Difference Vegetation Index) features of lodging and non-lodging area; (2) apply 80% of the data to train a classifier, and use the other 20% of data to test the classifier; and (3) report the accuracy of different classifiers.

## 4.3. Materials and Methods

### 4.3.1. Field Experiments

The wheat field was established in Thompson, North Dakota, as shown in Figure 4.1. In total, 372 plots of wheat were planted, with three dimensions of fields, 1.5 m x 3.6 m (5 ft. x 12 ft., 204 plots), 1.5 m x 5.4 m (5 ft. x 18 ft., 120 plots), and 1.5 m x 14.6 m (5 ft. x 48 ft., 48 plots). The seeds were sown on June 9, 2019, and the seeds germinated after two weeks. The lodging occurred around July 18, 2019 due to heavy rain and wind (data retrieved from the field identified as NDOWN).

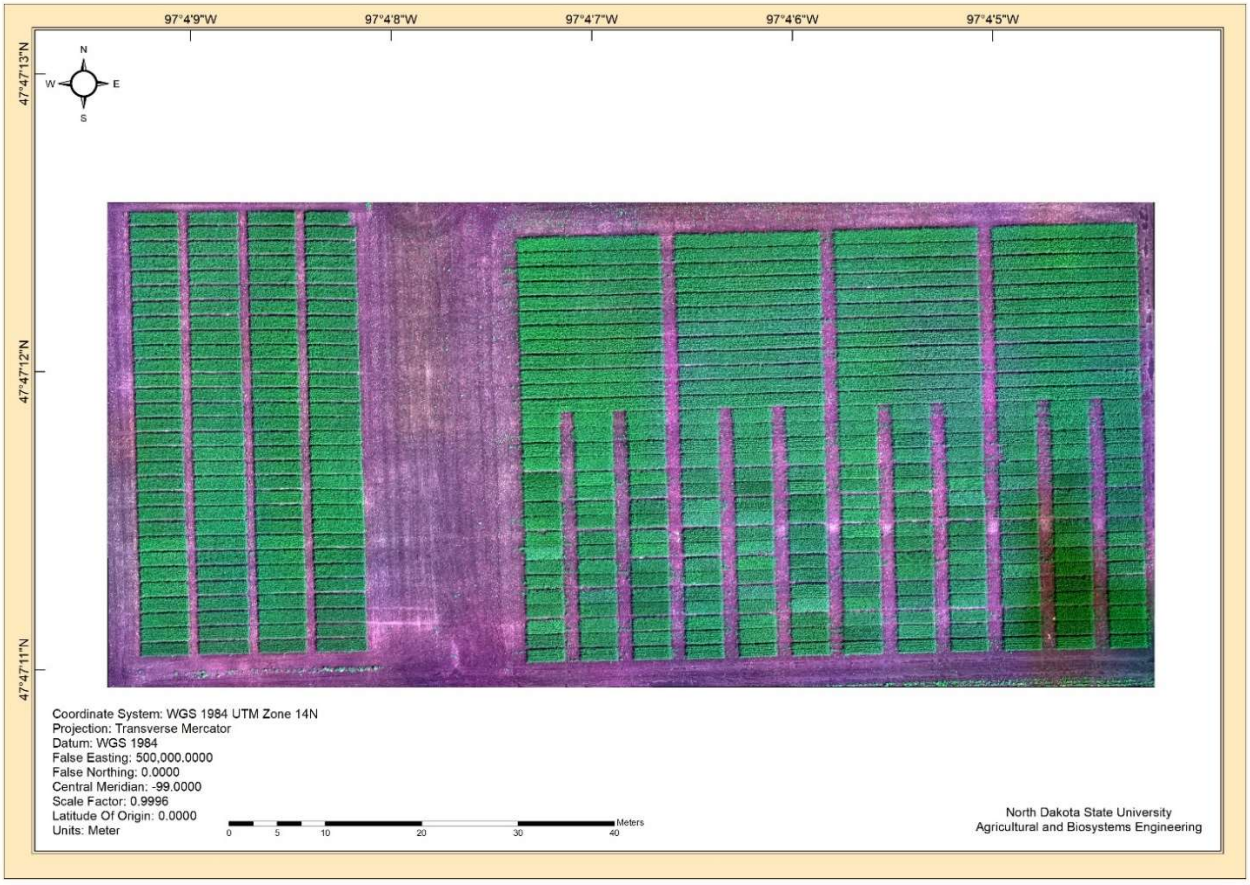



Fig. 4.1. Wheat Field Trial Established in Thompson, North Dakota, 2019.

### 4.3.2. Image Acquisition and Processing

For this research, two drones were used for data collection. A DJI Phantom 4D RTK drone, attached with an RGB camera (Phantom 4 Pro V2.0) was used to obtain the RGB images (DJI-Innovations, Inc., ShenZhen, China); and a DJI MATRICE 600 Pro (DJI-Innovations, Inc., ShenZhen, China), attached with a multispectral sensor (MicaSense RedEdge-MX Professional Multispectral Sensor, Simi Valley, CA, U.S.), was used to obtain the multispectral images. The DJI Phantom 4 flight altitude was 15 m (50 f.t.), and the DJI Matric 600 drone flight altitude was 46 m (150 ft.). Field data collection occurred three times – July 23, July 30, and August 8, 2019.

After each field data collection, obtained images, both RGB images from Phantom 4D and MicaSense 600, were stitched together using Pix4D mapper (Pix4D S.A., Switzerland) into an orthomosaic map, during which digital surface model and contour was established. The MicaSense Atlas was used to process the 5 band (blue, 475 nm; green, 560 nm; red, 668 nm; red edge, 717 nm; near-infrared 840 nm) tif file generated by Pix4D. Vegetable indices (e.g., NDVI) can be calculated using the 5 bands.

The first step is to crop the large plot into individual 372 plots. Then, extraction of RGB feature values can be conducted using Matlab (V2019a, MathWorks, Natick, MA, USA). The raw crop height information can be extracted from the ArcMap (ArcMap 10.6, Redlands, CA, USA). Eight Ground Control Points (GCP) were put into the field before the emergence of wheat seedlings, which were used to geo-reference the images from the two drones.

### 4.3.3. Extraction of Feature Values

In this study, three features (e.g., color, texture, and height) for each plot were extracted; and each plot was manually annotated to be lodged (0) or non-lodged (1).

#### *4.3.3.1. RGB Color Feature*

As an important discriminative property of objects, color information could help distinguish objects. The following equations (1), (2), and (3) are used for extracting red (r), green (g), and blue (b) channel characteristics from RGB images.

$$r = \frac{R}{R+G+B} \tag{4.1}$$

$$g = \frac{G}{R+G+B} \tag{4.2}$$

$$b = \frac{B}{R+G+B} \tag{4.3}$$

Considering chlorophyll has a large reflection rate in G and a large absorption rate in R and B, and the different orientations of lodging and non-lodging crops, Extra Green value (ExG, equ. 4) of wheat crop leafs is supposed to be larger than the stalks. In addition, extra green value has been cited and used extensively in crop growth evaluation [46].

$$ExG = 2 \times g - r - b \tag{4.4}$$

#### *4.3.3.2. Texture Feature*

Textural features similar to human visual perception are very useful for optimum selectivity. Specific texture features used in this work are coarseness (*Fcrs*), contrast (*Fcon*), linelikeness (*Flin*), and directionality (*Fdir*) [56]. The most important textural feature is coarseness, which is measured as the average of the largest window sizes needed to identify the pixel-centered texture elements. An image may contain different scales of texture. Coarseness aims to determine where texture occurs at the largest size, even if there is a smaller micro texture.

***Coarseness*** (*Fcrs*) is obtained following the below procedure.

***Step 1*:** For each pixel p(x,y), compute six averages for the windows of size k= 0, 1, 2, 3,

4, 5 around the 'pixel'. Take averages for every point of the input image over the neighborhood, with sizes ($2^k$) of 1x1, 2x2, 4x4, 8x8, 16x16, and 32x32, where *f(i, j)* is the gray-value at point (x, y).

$$A_k(x,y) = \sum_{i=x-2^{k-1}}^{x+2^{k-1}-1} \sum_{j=y-2^{k-1}}^{y+2^{k-1}-1} f(i,j) / 2^{2k} \quad (4.5)$$

***Step 2*:** For each point, take differences between pairs of averages corresponding to pairs of non-overlapping neighborhoods on opposite sides of the point in both horizontal (eq. 6) and vertical directions (eq. 7).

$$E_{k.h}(x,y) = |A_k(x+2^{k-1},y) - A_k(x-2^{k-1},y)| \quad (4.6)$$

$$E_{k.v}(x,y) = |A_k(x,y+2^{k-1}) - A_k(x,y-2^{k-1})| \quad (4.7)$$

***Step 3*:** For each point, pick the best size which gives the highest output value, where k maximizes E in either direction. $E_k = E_{max} = \max(E_1, E_2, \ldots E_L)$

$$S_{best}(x,y) = 2^k \quad (4.8)$$

***Step 4*:** Finally, take the average of *Sbest* throughout the picture to get a coarseness feature *Fcrs* (eq. 9), where m and n are the effective width and height of the individual plot image.

$$F_{crs} = \frac{1}{mxn} \sum_i^m \sum_j^n S_{best}(i,j) \quad (4.9)$$

***Contrast*** (*Fcon*) is a potential feature that can distinguish the lodging and non-lodging area. It determines how the intensity of the gray-level pixels varies in the image, and to what degree their distribution is biased to black or white. Contrast is calculated following equation 10, where $\sigma_4^{1/4}$is the fourth moment about the mean of the images, and $\sigma$ is the variance.

$$F_{con} = \frac{\sigma}{\sigma_4^{1/4}} \quad (4.10)$$

***Linelikeness***(*Flin*) is defined as the average coincidence of edge direction that co-occur

at pixels separated by a distance d along the direction α. The calculation procedure is shown in equation 10, where $P_{Dd(i,j)}$ is the m×m local direction co-occurrence matrix of points at distance. $P_{Dd(i,j)}$ is defined as the relative frequency with which two neighboring cells separated by a distance d along the edge direction occur on the image.

$$F_{lin} = \frac{\sum_{i=1}^{m}\sum_{j=1}^{m} P_{Dd(i,j)}\left(\cos(\mathrm{i-j})\frac{2\pi}{n}\right)}{\sum_{i=1}^{m}\sum_{j=1}^{m} P_{Dd(i,j)}} \quad (4.11)$$

***Directionality****(Fdir)* considers the edge strength and the direction of angle. $H_D(\phi)$, histogram of local edge probabilities, is computed against the directional angle. This histogram represents sufficiently global features of the input image, such as long lines and simple curves. This method utilizes the fact that gradient is a vector, so it has both magnitude and direction.

$$F_{dir} = \sum_{p}^{n_p}\sum_{\Phi}\left(\Phi + \Phi_p\right)^2 H_D(\Phi) \quad (4.12)$$

where,

p represents the number of peaks in the histogram,

np represents total number of peaks in the histogram,

$\Phi_p$ represents the pth peak location.

#### *4.3.3.3. NDVI Feature*

The Normalized Difference Vegetation Index (NDVI) uses two parameters (e.g., near-infrared and red light) to quantify vegetation [34]. The theory behind the NDVI parameter is vegetation strongly reflects near-infrared, but significantly absorbs red light.

$$NDVI = (NIR - Red)/(NIR + Red) \quad (4.13)$$

#### *4.3.3.4. Plant Height Feature*

Crop height is another important factor that can be used to estimate and measure the lodging area. For each plot, the raw height information of each pixel is obtained from the

ArcMap (ArcMap 10.6, Redlands, CA, USA), and then Stata (V14, College Station, TX) is used to process the raw height pixel data. Since wheat leaves/spikes are relative small, and the height information at pixel level varies a lot, it cannot represent the real condition or actual crop height information. In this study, all the heights at pixel level were first ranked from max to min, with the average top 10% height values used to represent the plot height. For one feature extracted from all plots, they are all normalized before running to the next step.

#### 4.3.4. Classifer and Datasets Separation

A Support Vector Machine (SVM), which is an effective binary linear classifier, is applied for data classification and prediction [47, 61]. To be specific, features of red (eq. 1), green (eq. 2), and blue (eq. 3), extra green (eq. 4), coarseness (eq. 9), contrast (eq. 10), linelikeness (eq. 11), directionality (eq. 12), NDVI (eq. 13), and plant height, coupled with the visually observed results (i.e., lodging and non-lodging) are used to train the SVM model. During the model training, the Gaussian kernel is selected, and the KernelScale is set to auto. As shown in Figure 4.2, there are 372 plots in total for one generated orthomosaic map. The 372 plot dataset is randomly divided into training (80%) and testing (20%) datasets. The label for each observation is manually added by visual observation.

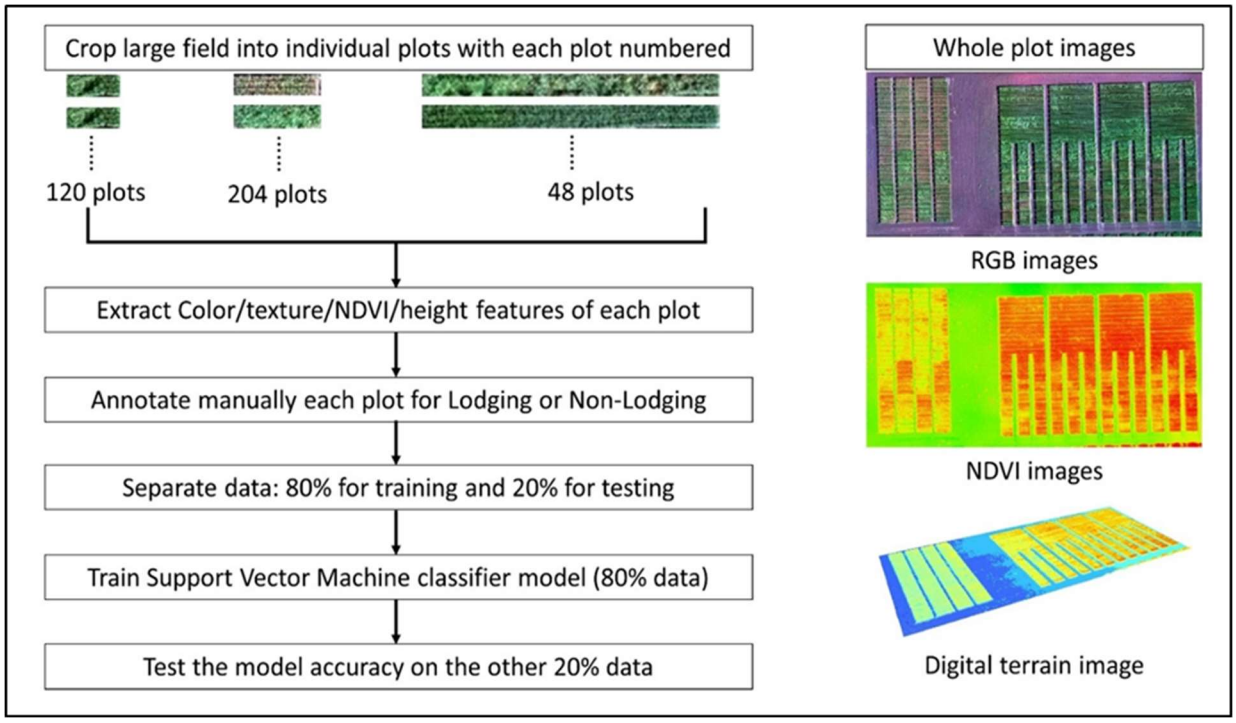



Fig. 4.2. The Schematic Diagram of Data Processing for Wheat Lodging Recognition.

### 4.3.5. Identification of Dominant Factors

In the above description, four factors (i.e., color, texture, NDVI, and height) are intended to be used to train the SMV model. With more factors used to train a model, the model prediction accuracy improves. However, more input factors require more time to train the model, as well as more time for prediction. Key features that play an important role in affecting model accuracy are identified.

## 4.4. Results and Discussion

### 4.4.1. Feature Analysis

#### *4.4.1.1. Color Feature Analysis*

Figure 4.3 shows the visual images of lodging and non-lodging individual plots, as well as the r, g, b, and Extra Green values of a lodging and non-lodging area. Due to their heavy overlaps, it is challenging to apply r, g, and b features to distinguish between lodging and non-lodging areas. ExG result is different between non-lodging and lodging plots, as lodging plots are associated with a larger value. Therefore the ExG result may possibly be used as an indicator to differ non-lodging from lodging. The presence of overlaps would decrease the accuracy.

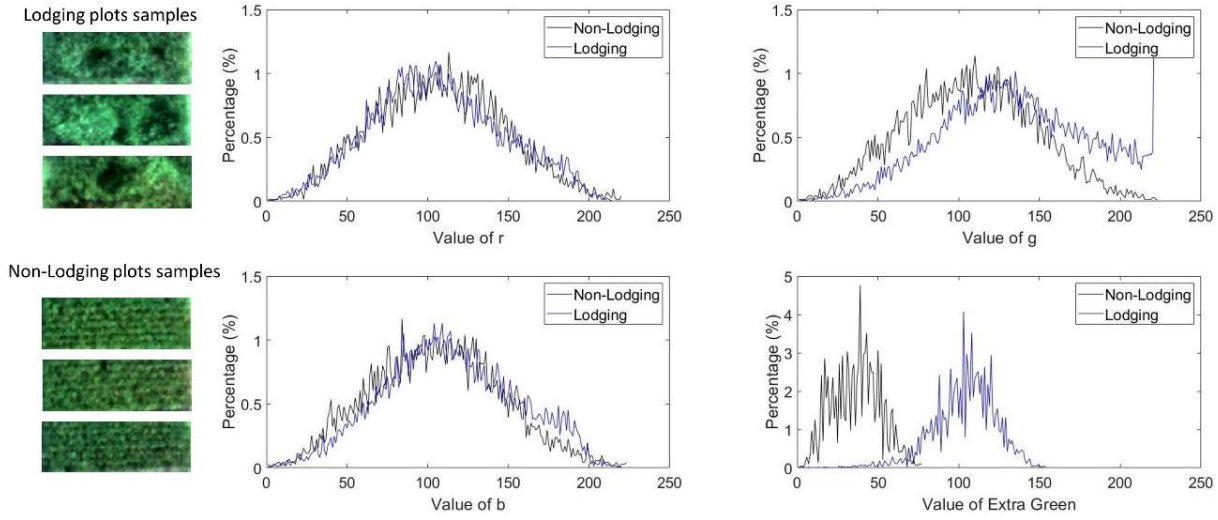



Fig. 4.3. Color Characteristics of Randomly Selected Lodging and Non-Lodging Areas in a Wheat Field (y-axis indicates the percentage of a certain value of pixels accounting for the total number of pixels in an image).

#### *4.4.1.2. Texture Feature Analysis*

As shown in figure 4.4, texture features of individual plots were extracted and analyzed according to the formulas described in Sections 3.3.2. The black line in each plot is the threshold value to distinguish between lodging and non-lodging. The threshold was calculated using maximizing intra-class (i.e., lodging and non-lodging) variance method. Intra-class variances are

used to obtain the desired characteristics of feature points. We want a threshold that holds the clusters as tight as possible and maximizes the distance between two clusters at the same time. The normalized lodging feature values of coarseness, contrast, and linelikeness are, to some extent, higher than that of non-lodging feature values, and they could potentially be used as a distinguishing feature. For the direction feature, however, it is difficult to distinguish between lodging and non-lodging plots, as the data are mixed. It is therefore difficult to use only one individual feature to differ lodging from non-lodging plots, so it is desirable to incorporate all four factors to improve accuracy.

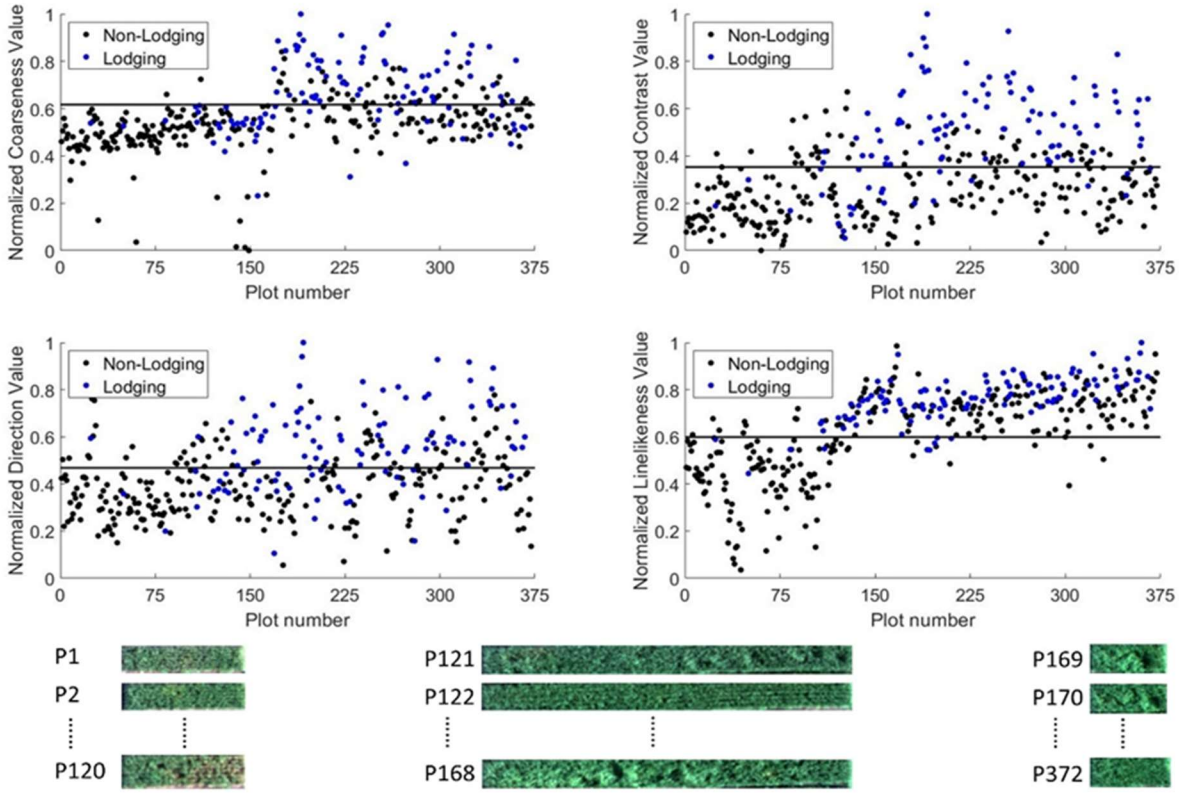



Fig. 4.4. Extracted Texture Feature Values of each Plot from an Orthomosaic Image (P represents plot, threshold line was calculated by maximizing the lodging and non-lodging class variance).

#### *4.4.1.3. NDVI Feature Analysis*

NDVI value of each pixel in a plot was averaged to obtain the plot's NDVI mean value; and the standard deviation of each pixel's NDVI value in a plot was also computed, with the results shown in Figure 4.5. The threshold line was calculated by maximizing the lodging and non-lodging class variance. For lodging plots, a majority of the normalized NDVI mean values are above 0.6, with only a few below this threshold. However, for non-lodging plots, the normalized NDVI mean values distribute in a large range, from 0 to 1. Therefore normalized NDVI mean value may be a desirable feature to distinguish lodging from non-lodging. Normalized NDVI standard deviation data for lodging are mixed with non-lodging, and it might not be a good indicator. At the right side of Figure 4.5, 17 color images and their corresponding NDVI data are shown. In the NDVI map, the red and green/yellow color stands for high and low values, respectively. The lodging plots (plots # 1, 2, 3, 4, 6, 7, 10, 13, and 14) have a larger NDVI value (more red) than the non-lodging area (more green). However, using NDVI is also not a satisfactory method, as some non-lodging areas (e.g., plot 5) has a red NDVI map.

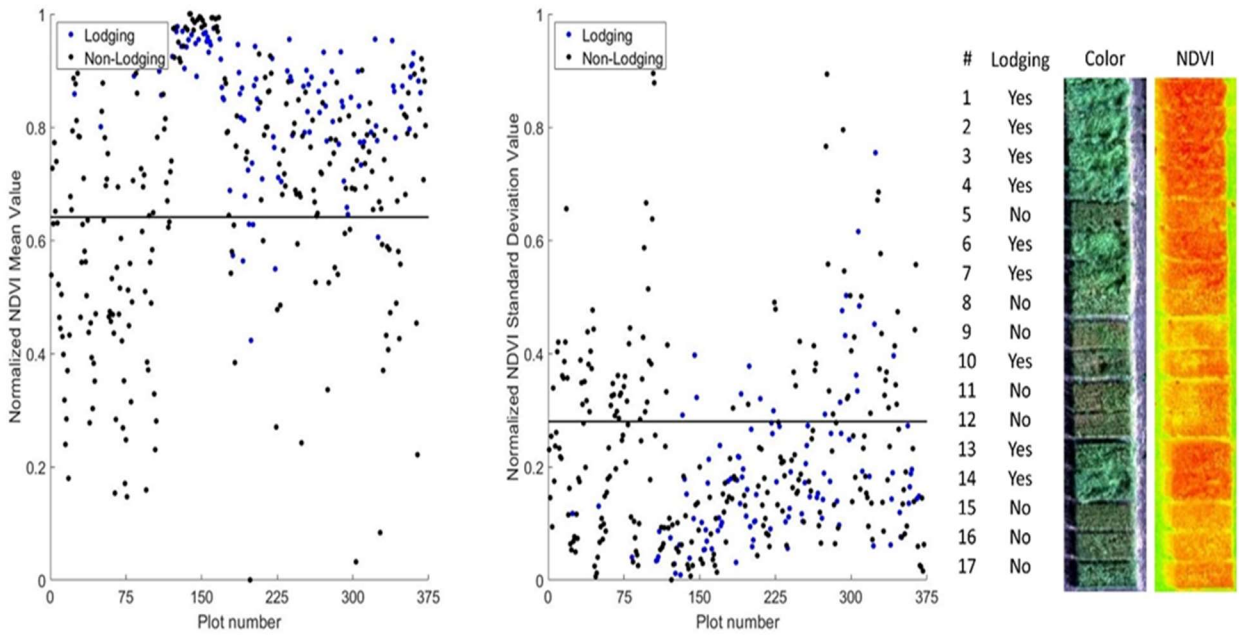



Fig. 4.5. RGB/NDVI Images and Extracted NDVI Mean and Standard Deviation Values of each plot from an Orthomosaic Image (plot number refers to Fig. 4.4; threshold line was calculated by maximizing the lodging and non-lodging class variance).

#### *4.4.1.4. Height Feature Analysis*

The plot height information at pixel level can be obtained from the digital terrain model (top photo in Figure 4.6). Height feature may be used as an indicator to differentiate lodging from non-lodging plots, as lodged crop height is usually lower than non-lodging crops. If part of the crops are lodged in a plot, the plot height standard deviation would be larger for lodged than non-lodged; if all crops lodged, the difference of plot height standard deviation between lodged and non-lodged would not be significant. To solve the all crop lodged issue, one more parameter of the average of top 10% height value was added, because the average of top 10% height value for lodging plot should be lower than that of a non-lodging plot. The results are shown in Figure 4.6. A threshold line, calculated by maximizing the lodging and non-lodging class variance, separates the two classes.

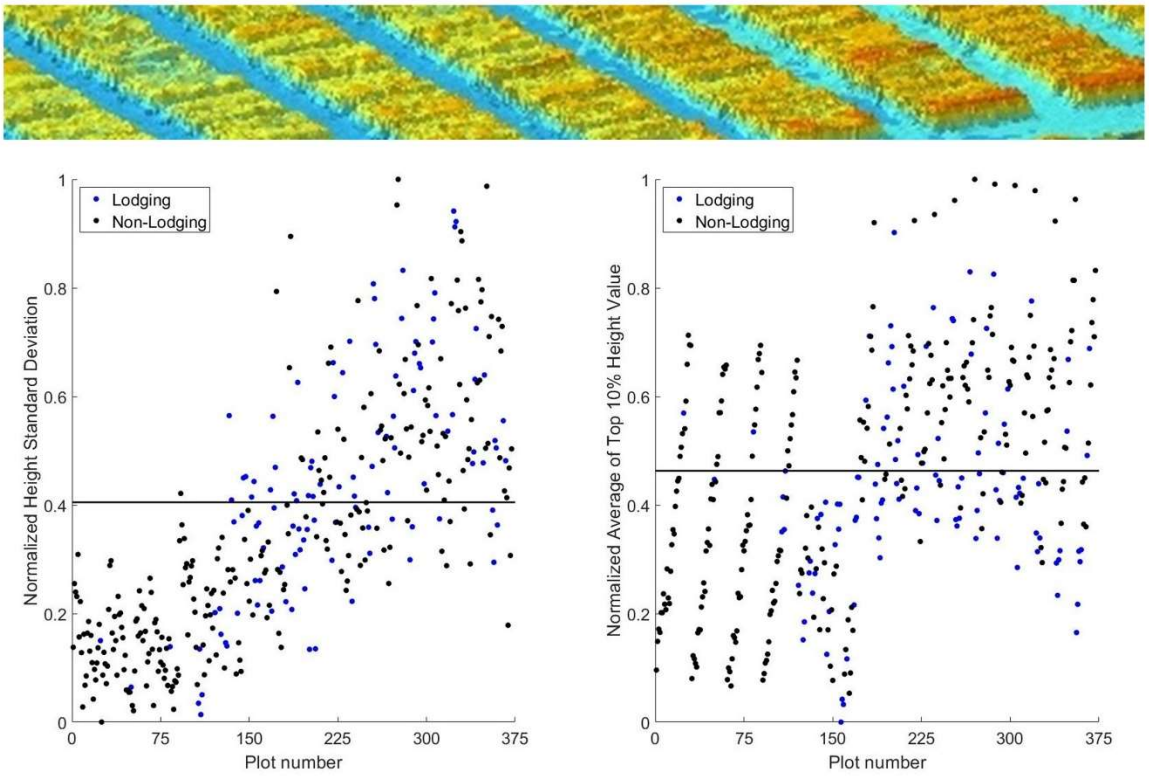



Fig. 4.6. Normalized Plot Height Standard Deviation and Average of Top 10% Height Value Features extracted from an Orthomosaic Image (plot number refers to Fig. 4.4; threshold line was calculated by maximizing the lodging and non-lodging class variance).

### 4.4.2. SVM Training and Predicting

In this study, we used single factor – color feature (CF), texture feature (TF), NDVI, and height feature (HF) to train the SVM model and test its accuracy. The performance of two factors (CF+TF, CF+NDVI, CF+HF, TF+NDVI, TF+HF, and NDVI+HF), three factors (CF+TF+NDVI, CF+TF+HF, TF+NDVI+HF, and CF+NDVI+HF), and four factors (CT+TF+NDVI+HF) was also explored to train the model and test its performance for different three days' data.

As shown in figure 4.7 for the July/23/2019 data results, the general trend is the more factors used to train the model, the higher the model prediction accuracy. Using only one of the four features (i.e., color, texture, NDVI, or height) to train the model, yields a prediction

accuracy ranging from 70% to 81%. The prediction accuracy from texture has a significantly higher accuracy than any of the other three individual factors. The single color feature does not perform well, mainly due to the significant overlapping for red, green, blue (Fig. 4.3). NDVI does not perform well, probably because the lodging is incomplete, and crop leafs hide the stems from top view. With two features factored in, the prediction accuracy ranges from 76% to 89%. Since texture is the most significant individual feature, any two factor combinations that include texture has a relatively high prediction accuracy ranging from 86% to 89%. The worse scenario occurs when the combination does not include the texture feature. With three factors and four factors included, the prediction accuracy is relatively high, ranging from 87% to 89% when texture is included; otherwise, the accuracy is only 79%. This dataset shows that the texture feature plays an important role in affecting the model prediction accuracy.

For the July/30/2019 data (shown in figure 4.8), results are similar to the July/23/2019 data; the texture feature is the most significant individual one, with one single feature leading to 84% prediction accuracy. However, the prediction accuracy when only using height feature is only 66%, significantly lower than any other individual feature. With two factors' combination, any one with texture feature incorporated results in a good prediction accuracy ranging from 86% to 88%. However, without texture feature included, the prediction accuracy ranges from 76% to 85%. When three or four factors are used to train the model, the accuracy ranges from 87% to 89% as long as the texture feature is included; otherwise, the prediction accuracy is only 82%.

For the August/8/2019 dataset (shown in figure 4.9), the results have a similar pattern as the July/23/2019 and July/30/2019 data, with the texture feature again being the most significant

one, and any combination of different features with texture included has a higher predication accuracy. The overall accuracy is generally higher than that of July/23/2019 and July/30/2019.

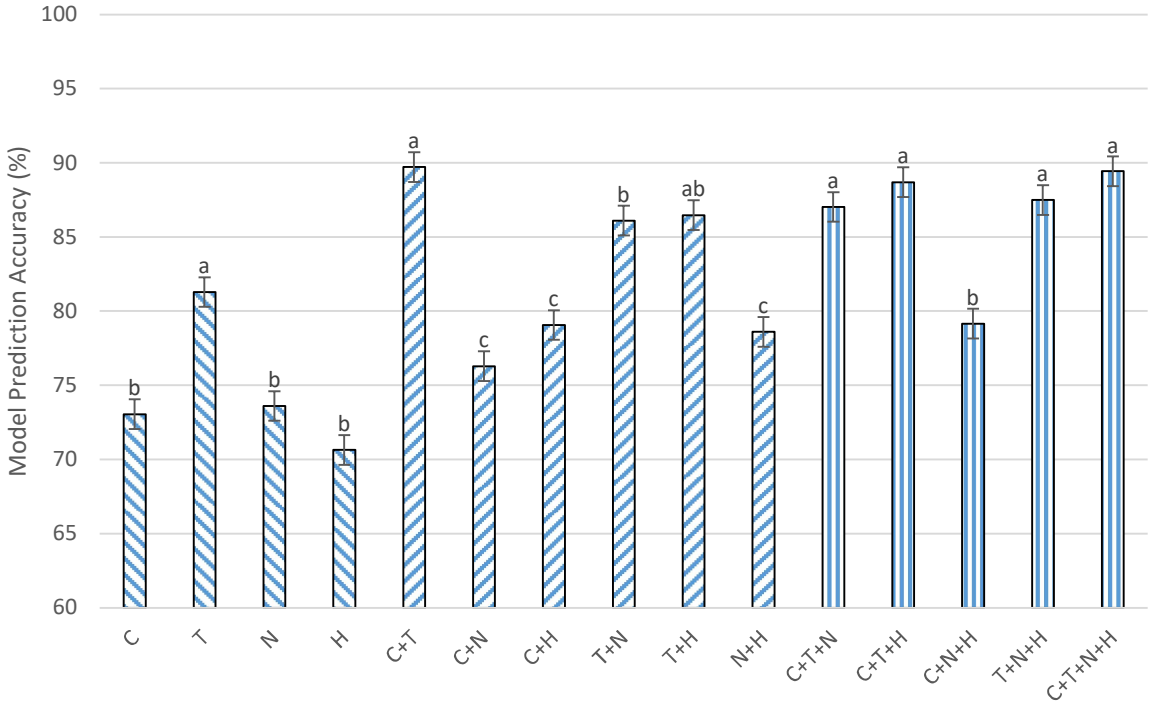



Fig. 4.7. Model Prediction Accuracy with Different Training Features for 7/23/2019 Data (C, T, N, and H stands for color, texture, NDVI, and height feature, respectively; statistic comparisons for the same number of training features were conducted by Tukey Test at the level of 0.05 with results showing no significant difference).

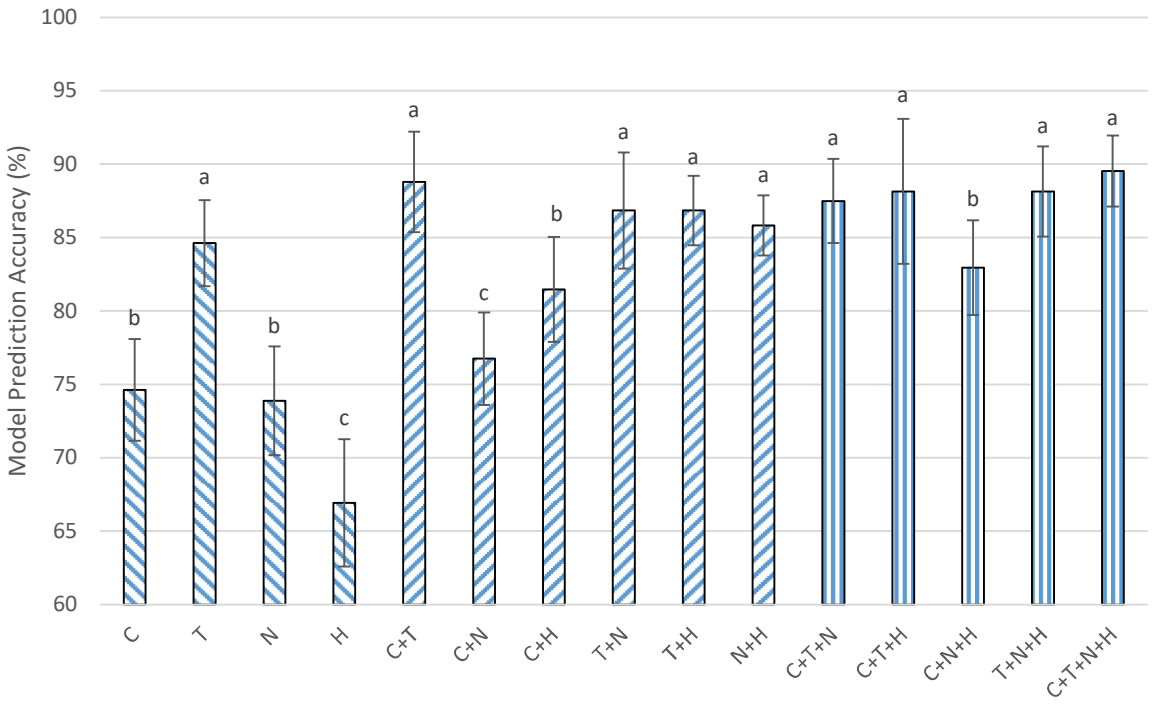



Fig. 4.8. Model Prediction Accuracy with Different Training Features for 7/30/2019 data (C, T, N, and H stands for color, texture, NDVI, and height feature, respectively; statistic comparisons for the same number of training features were conducted by Tukey Test at the level of 0.05 with results showing no significant difference).

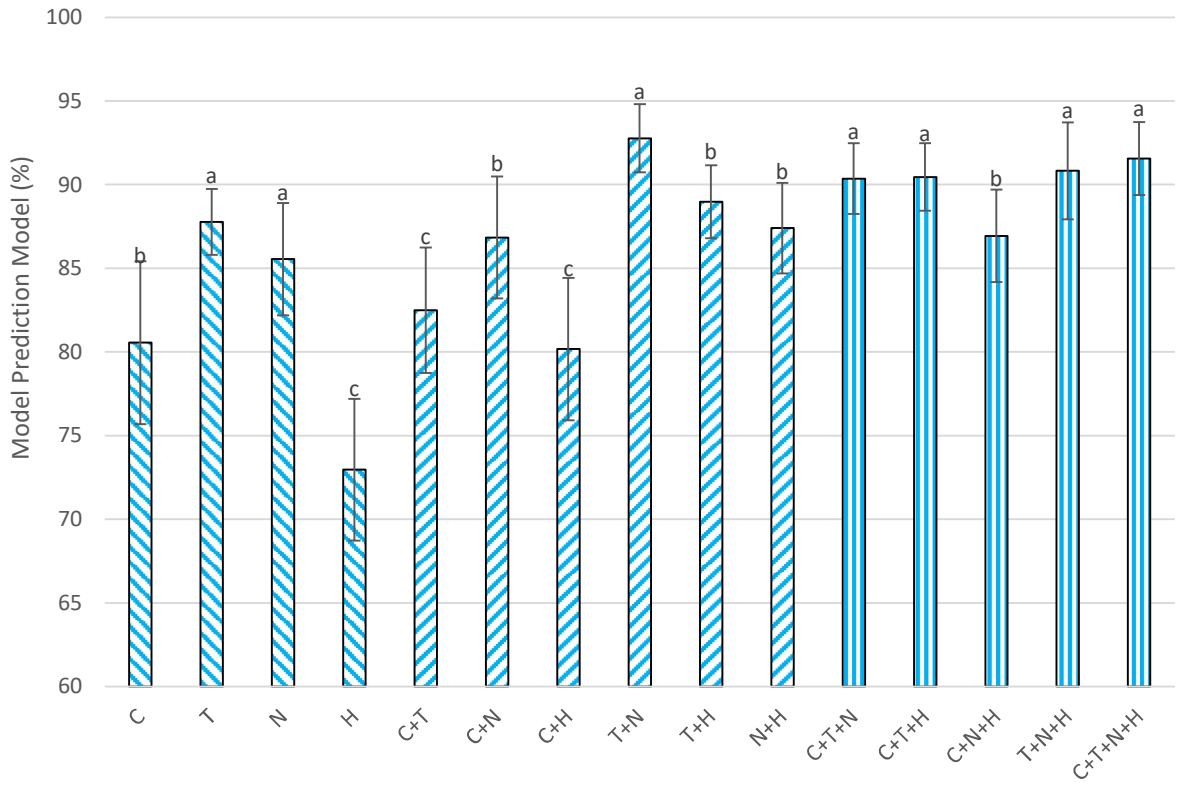



Fig. 4.9. Model Prediction Accuracy with Different Training Features for 8/8/2019 Data (C, T, N, and H stands for color, texture, NDVI, and height feature, respectively; statistic comparisons for the same number of training features were conducted by Tukey Test at the level of 0.05 with results showing no significant difference).

### 4.4.3. Identifying the Dominant Factor in Texture Features

From the above analysis of all three days' dataset, it has been identified that texture and the combinations with texture included have good performance in prediction accuracy. Figure 4.10 demonstrates the model prediction accuracy with different numbers of features included. For July/23/2019, only using one texture factor, the prediction accuracy is 81%, but with two (texture and NDVI) and four (texture, color, NDVI, and height) factors, the prediction accuracy improves to 86% and 89%, respectively. The same scenario occurs for the datasets of July/30/2019 and August/8/2019. It is also observed that the model prediction accuracies are not significantly different for two factors (texture and NDVI) vs. four factors (color, texture, NDVI, and Height). Therefore, it is suggested to only use texture and NDVI features for the model, to

simplify and speedup the classification process while not significantly reducing model prediction accuracy. Another overall trend is that model prediction accuracy increases as time passes after the lodging event. When only using Texture, the prediction accuracy increases from 81% to 87%; when using both texture and NDVI, the prediction accuracy increases from 86% to 92%; when using all four features, the prediction accuracy increases from 89% to 91%. A potential reason, based on the visual observation during the field data collection, is that the lodged crops are fully lodged after a longer time from the original lodging event.

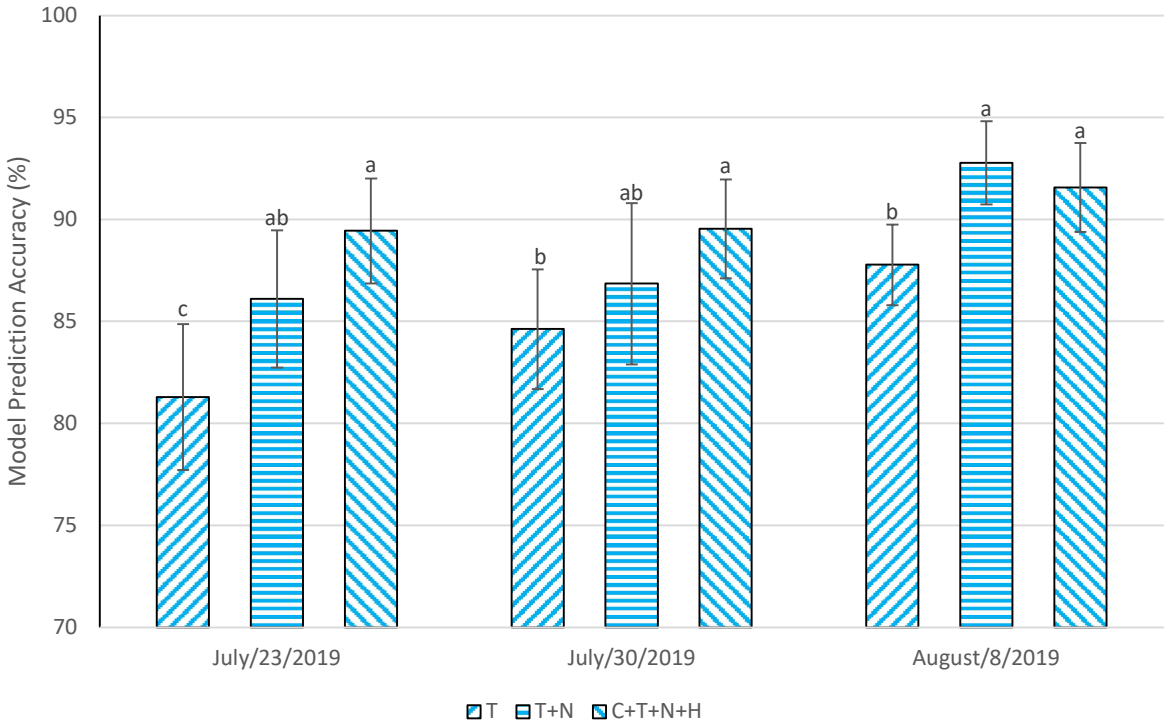



Fig. 4.10. Model Prediction Accuracy for three different days using one (texture), two (texture and NDVI), and four (color, texture, NDVI, and height) factors to train the Support Vector Machine model. (C, T, N, and H stands for color, texture, NDVI, and height feature, respectively; statistic comparisons for the same number of training features were conducted by Tukey Test at the level of 0.05 with results showing no significant difference).

Since texture is the most dominant feature in affecting model prediction accuracy, and there are four factors that form the texture feature, model prediction accuracy using the individual sub-features are also explored, with the results shown in Table 4.1. It's observed that among the four sub-features, contrast and direction lead to the highest model prediction accuracy. When only using the contrast sub-feature, model prediction accuracy is 75%±5% and 81%±4% for the datasets of July/23 and August/8/2019, respectively; when only using the direction sub-feature, the model prediction accuracy is 81%±3% for the dataset of July/30/2019.

Table 4.1. Model Prediction Accuracy using different Sub-Features of Texture.

| Feature | July/23/2019 | July/30/2019 | August/8/2019 |
|---|---|---|---|
| **Coarseness** | 72+3% | 70+5% | 67+4% |
| **Contrast** | 75+5% | 69+4% | 81+4% |
| **Direction** | 71+5% | 81+3% | 72+5% |
| **Linelikeness** | 67+5% | 63+5% | 64+5% |

### 4.4.4. CNN Model for Comparison

#### *4.4.4.1. Lodged/Non-Lodged Plots Classification using CNN*

The dataset was also classified using a CNN for comparison. After defining the CNN architecture [INPUT -> CONV -> RELU -> CONV -> RELU -> POOL -> CONV -> RELU -> POOL -> FC], the network was trained using stochastic gradient descent with a learning rate of 0.001. The max number of epochs is 10. An epoch is a complete training period on the entire set of training data. Given 300 plots training dataset, it can be split into 30 batches. This creates 10 iterations and shuffle the data in each iteration. The program trains the network on the training data and measures the accuracy during training of the validation data at regular intervals. The validation data are not used to adjust weights in the network. The 372 plot dataset is divided randomly into training (80%) and testing (20%) datasets, which resulted in prediction accuracies of 82%, 87%, and 89%, for July/23/2019, July/30/2019, and August/8/2019, respectively.

#### *4.4.4.2. Lodging Detection using Deep Learning Semantic Segmentation*

Semantic segmentation with the objective of assigning semantic labels to every pixel in an image is one of computer vision's fundamental topics. In comparison, deep convolutional neural networks (DCNN) have proved to be an efficient machine vision solution. However, models based on DCNN often require a large amount of data from the training.

As shown in Table 4.2, employing the ResNet-18 and ResNet50 network backbones improves the performance compared to the Mobilenetv2. We have observed 15.1% and 22.3% mIOU improvement for ResNet-18 and ResNet50 respectively.

Table 4.2. Comparison among the Lodging Detection Models Developed based on well established Architectures.

| Architecture | GlobalAccuracy | MeanAccuracy | MeanIoU | WeightedIoU | MeanBFScore |
|---|---|---|---|---|---|
| **Mobilenetv2** | 0.69826 | 0.64934 | 0.48608 | 0.56323 | 0.53532 |
| **ResNet18** | 0.82553 | 0.76411 | 0.63662 | 0.72049 | 0.70212 |
| **ResNet50** | 0.86067 | 0.85139 | 0.70917 | 0.77418 | 0.72634 |

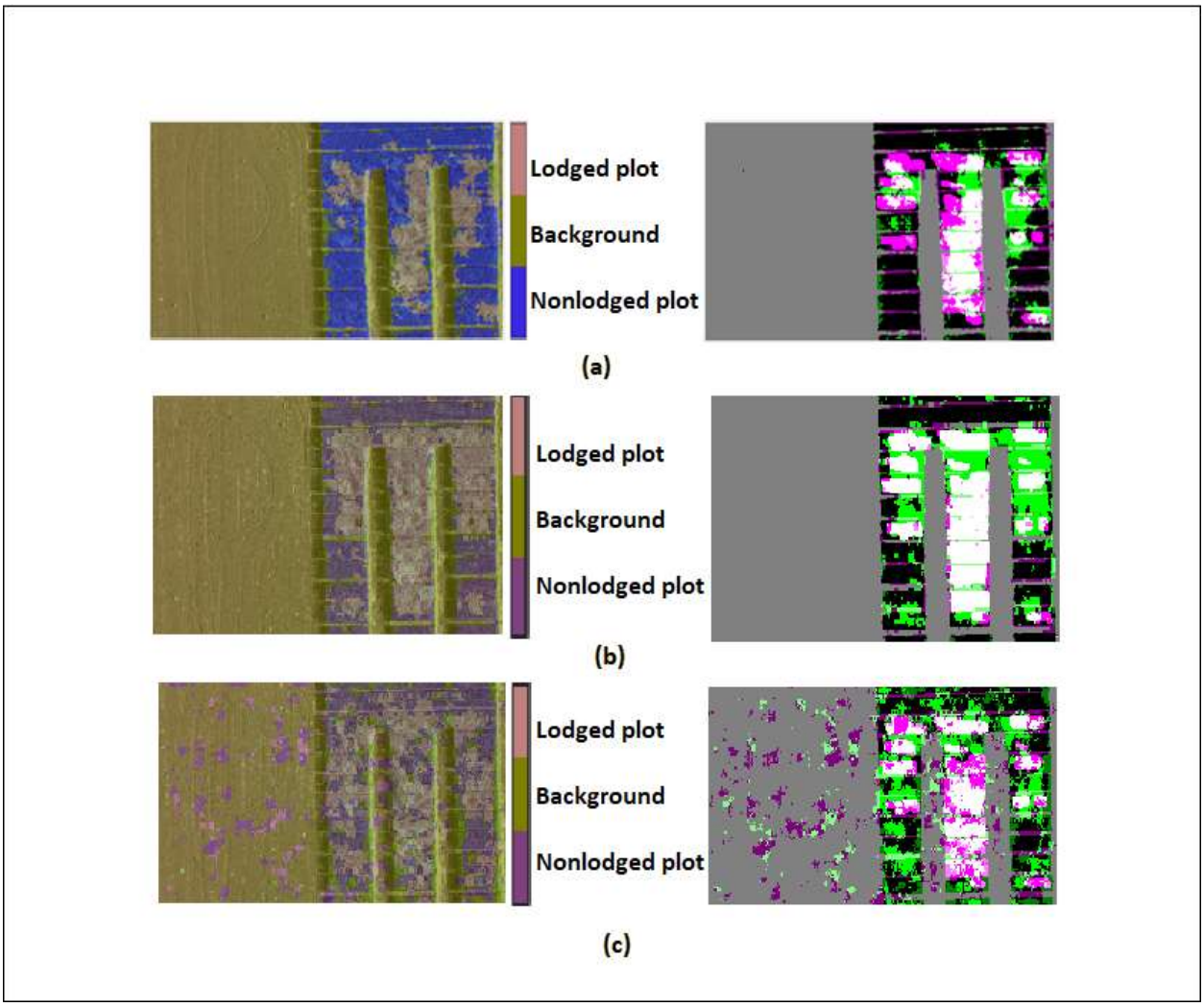



Fig. 4.11. Visualization Results for ResNet-18 (a), ResNet50 (b), and Mobilenetv2 (c).

As shown in figure 4.11, the green and magenta regions indicate places where the effects of the segmentation differ from the projected ground truth. Visually, the semantic segmentation results overlap well for ResNet-18 and ResNet50. However, the performance compared to the Mobilenetv2 is not as accurate.

**4.5. Conclusion**

In this study, we first collected RGB and multispectral images by attaching different cameras to a drone, then the images were stitched together, afterwhich individual plot images were cropped. Features of each plot image were then extracted, including color, feature, NDVI,

and height; and each plot was manually classified as lodging or non-lodging. 80% of the data for each day's dataset was used to train the developed SVM model, and the other 20% used for testing the model. The model with four factors included resulted in prediction accuracies of 89%, 89%, and 91%, for July/23/2019, July/30/2019, and August/8/2019, respectively. It was also found that the texture feature played the most important role in model prediction accuracy. With only texture and NDVI features used to train the model, prediction accuracy is 86%, 86%, and 92% for July/23/2019, July/30/2019, and August/8/2019, respectively. For comparison, the dataset was also classified using a CNN, which resulted in prediction accuracies of 82%, 87%, and 89%, for July/23/2019, July/30/2019, and August/8/2019, respectively. Hence, the proposed SVM model performs slightly better than a CNN, and is significantly faster to train (approximately 100 times faster). This study developed a simple method for using texture and NDVI features to train a SVM model, with prediction accuracy above 86%, providing theoretical and practical guidance for automated accurate wheat lodging monitoring.

# 5. CONCLUSION AND FUTURE WORK

The first part of this dissertation develops a VJA-based framework for detecting rotated blackbirds by innovatively covering the entire 360° rotation. The proposed algorithm achieves an 87% detection rate with 0 false detections, which is comparable to running the full VJA 360 times on an image rotated 1° each iteration, while also decreasing computation time by 25% compared to the baseline algorithm that rotates the image 45° each iteration to achieve a detection rate of only 73% with 0 false detections. The developed MATLAB code is included in Appendix A.

Future work includes investigation of additional parameters to further reduce the number of negative blocks passed to Steps 5 – 9, such as utilizing distance between overlapping ROI blocks to eliminate additional negative blocks from further processing, which will further reduce computation time. Additionally, this could be implemented on a GPU for parallel processing, and then customize the proposed algorithm for further high throughput computing, including potential modifications to optimal number of initial VJA stages (n), total number of VJA stages (X), Segmentation Size (SS), and Splits. Additional parameters will also come into play, such as the communication overhead during parallel processing.

For the second part of this dissertation, the fastest 72+32×32 asynchronous MAC in the literature [23] was redesigned by utilizing wavefront steering for the throughput-limiting accumulate feedback loop, resulting in a speedup of 1.36, while also decreasing area by 8%. Other major contributions of this work include a new way to calculate *OV*, which eliminated the need for an additional overflow feedback loop, and the notion of replicating control signals in a previous stage in order to reduce completion logic in the subsequent stage. The VHDL code for this throughput-optimized MAC is included in Appendix B.

For future work, performance of the Accumulate Feedback Loop could be further increased by utilizing a RCA at the end of the feedforward multiplier to sum the 2 feedforward PPs before the feedback circuit, such that only a single CSA would be needed in the feedback circuitry. Subsequently, the rest of the MAC circuitry would then need to be re-pipelined or re-optimized to meet or exceed the increased feedback loop performance, as was done in Section 3.4 for the design in this dissertation. Note that the trade-off for this proposed performance would be increased area.

In the third part of this dissertation, a support vector machine model was developed to distinguish lodging from non-lodging areas in a field, from aerial drone images, resulting in a prediction accuracy between 89% and 92%. Future work could entail further model optimization to enable real-time detection of lodging areas, and then potentially expanding this framework to be applicable to other crops as well, such as corn, soybean, and canola. The developed MATLAB code is included in Appendix C.

# REFERENCES


[1] P. E. Klug. CHAPTER 13: The Future of Blackbird Management Research. Ecology and Management of Blackbird (ICTERIDAE) in NORTH AMERICA, pp. 219-229.

[2] G. M. Linz; H. J. Homan; S. J. Werner; H. M. Hagy; W. J. Bleier. Assessment of Bird-management Strategies to Protect Sunflowers. American Institute of Biological Sciences, electronic ISSN 1525-3244, (2011).

[3] M. E. Klosterman. Comparisons between blackbird damage to corn and sunflower in North Dakota. Elsevier. Article in Crop Protection 53:1-5 · November (2013).

[4] N. Dalal; B. Triggs. Histograms of Oriented Gradients of Human Detection. Computer Vision and Pattern Recognition, IEEE, Montbonnot, France, 25 July (2005).

[5] Y. Wang. An Analysis of the Viola-Jones Face Detection Algorithm. Image Processing on line, IPOL Journal, ENS Cachan, France, pp 128–148, June 26, (2014).

[6] R. Anil. Bird's eye view: Detecting and Recognizing Birds using the BIRDS 200 dataset. University of California, San Diego, (2011).

[7] M. Kolsch; M. Turk. Analysis of Rotational Robustness of Hand Detection with a Viola-Jones Detector. Pattern Recognition. ICPR (2004).

[8] C. Huang; H. Ai; Y. Li; S. Lao. High-Performance Rotation Invariant Multiview Face Detection. IEEE Trans Pattern Anal Mach Intell. (2007) Apr. 29(4):671-86.

[9] L. Acasandrei; A. Barriga. AMBA bus hardware accelerator IP for Viola-Jones face detection. IET Computers Digital Techniques, September (2013).

[10] S. Ren; N. Deligiannis; Y. Andreopoulos. Dynamic Scheduling for Energy Minimization in Delay-Sensitive Stream Mining. IEEE Transactions on Signal Processing (Volume: 62, Issue: 20, Oct.15, 2014).

[11] W. Wang. Parallelization and performance optimization on face detection algorithm with OpenCL A case study. Tsinghua Science and Technology (Volume: 17, Issue: 3, June 2012).

[12] Y. Ampatzidis; J. Ward; O. Samara. Autonomous System for Pest Bird Control in Specialty Crops using Unmanned Aerial Vehicles. ASABE Annual International Meeting, paper #: 152181748, July 26–29, (2015).

[13] K. S. Christie; S. L. Gilbert; C. L. Brown; M. Hatfield; L. Hanson. Unmanned aircraft systems in wildlife research: current and future applications of a transformative technology. Front Ecol Environ, pp. 241–251, (2016).

[14] M. Jones; P. Viola. Fast Multi-view Face Detection. Mitsubishi Electric Research Laboratories, MERL, Cambridge, MA, August (2003).

[15] INPIXAL designs, develops and sells image processing products for the defence and video surveillance markets. (2019). Retrieved from http://www.inpixal.com/en/

[16] D. G. Lowe. Distinctive Image Features from Scale-Invariant Key points. International Journal of Computer Vision, Springer, Vancouver, B.C., Canada, pp 91–110, January 5, (2004).

[17] H. Ai. Vector Boosting for Rotation Invariant Multi-View Face Detection. Computer Vision, IEEE International Conference on (2005).

[18] P. Viola; M. Jones. Rapid Object Detection Using a Boosted Cascade of Simple Features. Computer Vision and Pattern Recognition. CVPR (2001).

[19] R. Lienhart. An extended set of Haar-like Features for Rapid Object Detection. Intel Labs, Intel Corporation, Image Processing, (2002).

[20] H. Rowley; S. Baluja; T. Kanade. Neural network-based face detection. In IEEE Patt. Anal. Mach. Intell., volume 20, pages 22–38, (1998).

[21] H. Schneiderman; T. Kanade. A statistical method for 3D object detection applied to faces and cars. In International Conference on Computer Vision, (2000).

[22] E. Karami; S. Prasad; M. Shehata. Image matching using sift surf brief and orb: Performance comparison for distorted images. Proc. Newfoundland Electr. Comput. Eng. Conf., Nov. (2015).

[23] L. Zhou; S. C. Smith. Speedup of a Large Word-Width High-Speed Asynchronous Multiply and Accumulate Unit. 52nd IEEE International Midwest Symposium on Circuits and Systems, pp. 499-502, August (2009).

[24] K. M. Fant; S. A. Brandt. NULL Convention Logic: A Complete and Consistent Logic for Asynchronous Digital Circuit Synthesis. International Conference on Application Specific Systems, Architectures, and Processors, pp. 261–273, (1996).

[25] S. C. Smith; R. F. DeMara; J. S. Yuan; M. Hagedorn; D. Ferguson. Delay-Insensitive Gate-Level Pipelining. Elsevier's Integration, the VLSI Journal, Vol. 30/2, pp. 103-131, October (2001).

[26] S. C. Smith. Speedup of NULL Convention Digital Circuits Using NULL Cycle Reduction. Elsevier's Journal of Systems Architecture, Vol. 52/7, pp. 411-422, July (2006).

[27] S. C. Smith. Physical-level static gates designed with 1.8V, 0.18um technology. (2019). Retrieved from https://www.ndsu.edu/pubweb/~scotsmit/NCL_gates.vhd

[28] Scott C. Smith; Jia Di. Designing Asynchronous Circuits using NULL Convention Logic (NCL). Synthesis Lectures on Digital Circuits and Systems, Morgan & Claypool Publishers, Vol. 4/1, July (2009).

[29] D. H. Linder; J. H. Harden. Phased logic: supporting the synchronous design paradigm with delay-insensitive circuitry. IEEE Transactions on Computers, Vol. 45/9, pp. 1031–1044, (1996).

[30] M. Nielsen. Neural Networks and Deep Learning. (Determination Press, 2015).

[31] Berry, P. M.; Spink, J. Predicting yield losses caused by lodging in wheat. (2012). Field Crops Research, 137, 19-26.

[32] Berry, P. M.; Sterling, M.; Baker, C. J.; Spink, J.; Sparkes, D. L. A calibrated model of wheat lodging compared with field measurements. (2003). Agricultural and Forest Meteorology, 119(3-4), 167-180.

[33] Berry, P. M.; Sylvester-Bradley, R.; Berry, S. Ideotype design for lodging-resistant wheat. (2007). Euphytica, 154(1-2), 165-179.

[34] Carlson, T. N.; Ripley, D. A. On the relation between NDVI, fractional vegetation cover, and leaf area index. (1997). Remote sensing of Environment, 62(3), 241-252.

[35] Chang, H. H.; Zilberman, D. On the political economy of allocation of agricultural disaster relief payments: application to Taiwan. (2013). European Review of Agricultural Economics, 41(4), 657-680.

[36] Chu, T.; Starek, M.; Brewer, M.; Murray, S.; Pruter, L. Assessing lodging severity over an experimental maize (Zea mays L.) field using UAS images. (2017). Remote Sensing, 9(9), 923.

[37] Crook, M. J.; Ennos, A. R. The mechanics of root lodging in winter wheat, Triticum aestivum L. (1993). Journal of Experimental Botany, 44(7), 1219-1224.

[38] Dandois, J. P.; Ellis, E. C. High spatial resolution three-dimensional mapping of vegetation spectral dynamics using computer vision. (2013). Remote Sensing of Environment, 136, 259-276.

[39] Enjolras, G.; Sentis, P. Crop insurance policies and purchases in France. (2011). Agricultural Economics, 42(4), 475-486.

[40] Hitaka, H. Studies on the lodging of rice plants. (1969). Jpn. Agric. Res. Quart, 4(3), 1-6.

[41] Horvatic, B. Using Drone Mapping for Crop Insurance. (2019). Retrieved from https://www.precisionag.com/in-field-technologies/drones-uavs/using-drone-mapping-for-crop-insurance/

[42] Laliberte, A. S. ; Rango, A. Texture and scale in object-based analysis of subdecimeter resolution unmanned aerial vehicle (UAV) imagery. (2009). IEEE Transactions on Geoscience and Remote Sensing, 47(3), 761-770.

[43] Li, X.; Wang, K.; Ma, Z.; Wang, H. Early detection of wheat disease based on thermal infrared imaging. (2014). Transactions of the Chinese Society of Agricultural Engineering, 30(18), 183-189.

[44] Li, Z.; Chen, Z.; Wang, L.; Liu, J.; Zhou, Q. Area extraction of maize lodging based on remote sensing by small unmanned aerial vehicle. (2014). Transactions of the Chinese Society of Agricultural Engineering, 30(19), 207-213.

[45] Liu, L. Y.; Wang, J. H.; Song, X. Y.; Li, C. J.; Huang, W. J.; ZHAO, C. J. The canopy spectral features and remote sensing of wheat lodging. (2005). JOURNAL OF REMOTE SENSING-BEIJING, 9(3), 323.

[46] Liu, T.; Li, R.; Zhong, X.; Jiang, M.; Jin, X.; Zhou, P.; Guo, W. Estimates of rice lodging using indices derived from UAV visible and thermal infrared images. (2018). Agricultural and forest meteorology, 252, 144-154.

[47] Lu, Y.; Lu, R. Detection of surface and subsurface defects of apples using structured-illumination reflectance imaging with machine learning algorithms. (2018).

[48] Mardanisamani, S.; Maleki, F.; Hosseinzadeh Kassani, S.; Rajapaksa, S.; Duddu, H.; Wang, M.; Vail, S. Crop Lodging Prediction from UAV-Acquired Images of Wheat and Canola using a DCNN Augmented with Handcrafted Texture Features. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. (2019).

[49] Neenan, M.; Spencer-Smith, J. L. An analysis of the problem of lodging with particular reference to wheat and barley. (1975). The Journal of agricultural science, 85(3), 495-507.

[50] North Dakota Wheat commission. (2019). Retrieved from https://www.ndwheat.com/

[51] Okhrin, O.; Odening, M.; Xu, W. Systemic weather risk and crop insurance: the case of China. (2013). Journal of Risk and Insurance, 80(2), 351-372.

[52] Peake, A. S.; Huth, N. I.; Carberry, P. S.; Raine, S. R.; Smith, R. J. Quantifying potential yield and lodging-related yield gaps for irrigated spring wheat in sub-tropical Australia. (2014). Field Crops Research, 158, 1-14.

[53] Setter, T. L.; Laureles, E. V.; Mazaredo, A. M. Lodging reduces yield of rice by self-shading and reductions in canopy photosynthesis. (1997). Field Crops Research, 49(2-3), 95-106.

[54] Somers, B.; Asner, G. P.; Tits, L.; Coppin, P. Endmember variability in spectral mixture analysis: A review. (2011). Remote Sensing of Environment, 115(7), 1603-1616.

[55] Stapper, M.; Fischer, R. A. Genotype, sowing date and plant spacing influence on high-yielding irrigated wheat in southern New South Wales. II. Growth, yield and nitrogen use. (1990). Australian Journal of Agricultural Research, 41(6), 1021-1041.

[56] Tamura, H.; Mori, S.; Yamawaki, T. Textural features corresponding to visual perception. IEEE Transactions on Systems, man, and cybernetics. (1978). 8(6), 460-473.

[57] Turner, D.; Lucieer, A.; Watson, C. An automated technique for generating georectified mosaics from ultra-high resolution unmanned aerial vehicle (UAV) imagery, based on structure from motion (SfM) point clouds. (2012), Remote sensing, 4(5), 1392-1410.

[58] USDA, Economic Research Service. (2019). Retrieved from https://www.ers.usda.gov/topics/crops/wheat/

[59] USDA, National Agricultural Statistics Service. (2019). Retrieved from https://www.nass.usda.gov/

[60] USDA, Risk Management (2019). Retrieved from https://www.rma.usda.gov/

[61] Vapnik, V. The nature of statistical learning theory. Springer science & business media. (2013).

[62] Weibel, R. O.; Pendleton, J. W. Effect of artificial lodging on winter wheat grain yield and quality 1. Agronomy Journal, (1964), 56(5), 487-488.

[63] Yahyanejad, S.; Rinner, B. A fast and mobile system for registration of low-altitude visual and thermal aerial images using multiple small-scale UAVs. ISPRS Journal of Photogrammetry and Remote Sensing, (2015), 104, 189-202.

[64] Yang, H.; Chen, E.; Li, Z.; Zhao, C.; Yang, G.; Pignatti, S.; Zhao, L. Wheat lodging monitoring using polarimetric index from RADARSAT-2 data. International Journal of Applied Earth Observation and Geoinformation, (2015), 34, 157-166.

[65] Yang, M. D.; Huang, K. S.; Kuo, Y. H.; Tsai, H.; Lin, L. M. Spatial and spectral hybrid image classification for rice lodging assessment through UAV imagery. Remote Sensing, (2017), 9(6), 583.

[66] A. Krizhevsky; I. Sutskever; G. Hinton. Imagenet classification with deep convolutional neural networks. In NIPS, (2012).

[67] S. Futch; S. Weingarten; M. Irey. Determining hlb infection levels using multiple survey methods in florida citrus. Proc. Fla. State Hort. Soc, vol. 122, (2009), pp. 152–157.

[68] E. B. dos Santos; C. C. T. Mendes; F. S. Osorio; D. F. Wolf. Bayesian networks for obstacle classification in agricultural environments. Intelligent Transportation Systems-(ITSC); 16th International IEEE Conference on. IEEE, (2013), pp. 1416–1421.

[69] Xue X Y; Liang J; Fu X M. Prospect of aviation plant protection in China. Chinese agricultural mechanization, (2008); (5): 72–74.

[70] He X K. The opportunity and challenge of pesticide application for agriculture and horticulture in China. International Advances in Pesticide Application, (2010); 99: 157–164.

[71] Zhang H H; Lan Y; Lacey R E; Huang Y B; Hoffmann W C; Martin D, et al. Analysis of variograms with various sample sizes from a multispectral image. Int J Agric & Bio Eng, (2009); 2(4): 62–69.

[72] Huang Y B; Thomson S J; Lan Y B; Mass J S. Multispectral imaging systems for airborne remote sensing to support agricultural production management. Int J Agric & Bio Eng, (2010); 3(1): 50–62.

[73] Huang Y B; Thomson S J. Characterization of spray deposition and drift from a low drift nozzle for aerial application at different application altitudes. Electronics Letters, (2011); 38(17): 967–968.

[74] Beale, R.; Jackson, T. Neural Computing—An Introduction. CRC Press: Boca Raton, FL, USA, (1990).

[75] Paloscia, S.; Pampaloni, P.; Pettinato, S.; Santi, E. A comparison of algorithms for retrieving soil moisture from ENVISAT/ASAR images. IEEE Trans. Geosci. Remote Sens. (2008), 46, 3274–3284.

[76] Notarnicola, C.; Angiulli, M.; Posa, F. Soil moisture retrieval from remotely sensed data: Neural network approach versus Bayesian method. IEEE Trans. Geosci. Remote Sens. (2008), 46, 547–557.

[77] Del Frate, F.; Ferrazzoli, P.; Schiavon, G. Retrieving soil moisture and agricultural variables by microwave radiometry using neural networks. Remote Sens. Environ. (2003), 84, 174–183.

[78] Vapnik, V. The Nature of Statistical Learning Theory, 2nd ed.; Springer: New York, NY, USA, (1999).

[79] Durbha, S.S.; King, R.L.; Younan, N.H. Support vector machines regression for retrieval of leaf area index from multiangle imaging spectroradiometer. Remote Sens. Environ. (2007), 107, 348–361.

[80] Moser, G.; Serpico, S. Automatic parameter optimization for support vector regression for land and sea surface temperature estimation from remote sensing data. IEEE Trans. Geosci. Remote Sens. (2009), 47, 909–921.

[81] Yiran, G.A.B.; Kusimi, J.M.; Kufogbe, S.K. A synthesis of remote sensing and local knowledge approaches in land degradation assessment in the Bawku East District, Ghana. Int. J. Appl. Earth Obs. Geoinf. (2012), 14, 204–213.

[82] Ali, I.; Cawkwell, F.; Dwyer, E.; Barrett, B.; Green, S. Satellite remote sensing of grasslands: From observation to management. J. Plant Ecol. (2016), 9, 649–671.

[83] Numata, I.; Roberts, D.A.; Chadwick, O.A.; Schimel, J.P.; Galvão, L.S.; Soares, J.V. Evaluation of hyperspectral data for pasture estimate in the Brazilian Amazon using field and imaging spectrometers. Remote Sens. Environ. (2008), 112, 1569–1583.

[84] Clevers, J.; Van der Heijden, G.; Verzakov, S.; Schaepman, M. Estimating grassland biomass using SVM band shaving of hyperspectral data. Photogramm. Eng. Remote Sens. (2007), 73, 1141–1148.

[85] Gao, J. Quantification of grassland properties: how it can benefit from geoinformatic technologies? Int. J. Remote Sens. (2006), 27, 1351–1365.

[86] Acharya S.M.; Pawar S.S.; Wable N.B. Application of Remote Sensing & GIS in Agriculture. ISSN: 2349-6495(P) | 2456-1908(O).

[87] Bendig, J.; Yu, K.; Aasen, H.; Bolten, A.; Bennertz, S.; Broscheit, J.; Gnyp, M.L.; Bareth, G. Combining UAV-based plant height from crop surface models, visible, and near infrared vegetation indices for biomass monitoring in barley. Int. J. Appl. Earth Obs. Geoinf. (2015), 39, 79–87.

[88] Zheng, H.; Cheng, T.; Li, D.; Zhou, X.; Yao, X.; Tian, Y.; Cao, W.; Zhu, Y. Evaluation of RGB, Color-Infrared and Multispectral Images Acquired from Unmanned Aerial Systems for the Estimation of Nitrogen Accumulation in Rice. Remote Sens. (2018), 10, 824.

[89] Yang, G.; Liu, J.; Zhao, C.; Li, Z.; Huang, Y.; Yu, H.; Xu, B.; Yang, X.; Zhu, D.; Zhang, X. Unmanned Aerial Vehicle Remote Sensing for Field-Based Crop Phenotyping: Current Status and Perspectives. Front. Plant Sci. (2017), 8, 1111.

[90] Vadrevu, K.P. Introduction to Remote Sensing (FIFTH EDITION); Campbell, J.B., Wynne, R.H., Eds.; Guilford Press: New York, NY, USA, (2013); Volume 28, pp. 117–118. ISBN 978-160918-176-5.

[91] Hunt, E.R.; Cavigelli, M.; Daughtry, C.S.T.; McMurtrey, J.E.; Walthall, C.L. Evaluation of Digital Photography from Model Aircraft for Remote Sensing of Crop Biomass and Nitrogen Status. Precis. Agric. (2005), 6, 359–378.

[92] Nakatani, M.; Kawashima, S. An Algorithm for Estimating Chlorophyll Content in Leaves Using a Video Camera. Ann. Bot. (1998), 81, 49–54.

[93] Woebbecke, D.M.; Meyer,G.E.; VonBargen,K.; Mortensen, D.A. Color Indices for Weed Identification Under Various Soil, Residue, and Lighting Conditions. Trans. ASAE (1995), 38, 259–269.

[94] Golzarian, M.R.; Frick, R.A.; Rajendran, K.; Berger, B.; Roy, S.; Tester, M.; Lun, D.S. Accurate inference of shoot biomass from high-throughput images of cereal plants. Plant Methods (2011), 7, 2.

[95] Geipel, J.; Link, J.; Claupein, W. Combined Spectral and Spatial Modeling of Corn Yield Based on Aerial Images and Crop Surface Models Acquired with an Unmanned Aircraft System. Remote Sens. (2014), 6, 10335–10355.

[96] Cui, R.-X.; Liu, Y.-D.; Fu, J.-D. Estimation of Winter Wheat Biomass Using Visible Spectral and BP Based Artificial Neural Networks. Guang pu xue yu guang pu fen xi = Guang pu (2015), 35, 2596.

[97] Li, Y.; Chen, D.; Walker, C.; Angus, J. Estimating the nitrogen status of crops using a digital camera. Field Crop. Res. (2010), 118, 221–227.

# APPENDIX A. VIOLA-JONES CODE

```
function HH = BB_HOGDescriptor(hodBB, BirdImg)

HH = [];
assert(isequal(size(BirdImg), hodBB.winSize))
bbhx = [-1,0,1];
bbhy = bbhx';

dx = filter2(bbhx, double(BirdImg));
dy = filter2(bbhy, double(BirdImg));

dx = dx(2 : (size(dx, 1) - 1), 2 : (size(dx, 2) - 1));
dy = dy(2 : (size(dy, 1) - 1), 2 : (size(dy, 2) - 1));

bbangles = atan2(dy, dx);
bbmagnit = ((dy.^2) + (dx.^2)).^.5;
%    Compute histogram
BBhistogram = zeros(hodBB.numVertCells, hodBB.numHorizCells, hodBB.numBins);

for NNrow = 0:(hodBB.numVertCells - 1)
   rowOffset = (NNrow * hodBB.cellSize) + 1;
   for NNcol = 0:(hodBB.numHorizCells - 1)
      colOffset = (NNcol * hodBB.cellSize) + 1;
      rowIndeces = rowOffset : (rowOffset + hodBB.cellSize - 1);
      colIndeces = colOffset : (colOffset + hodBB.cellSize - 1);
      cellAngles = bbangles(rowIndeces, colIndeces);
      cellMagnitudes = bbmagnit(rowIndeces, colIndeces);
      BBhistogram(NNrow + 1, NNcol + 1, :) = getHistogram(cellMagnitudes(:), cellAngles(:), hodBB.numBins);
   end
end

for NNrow = 1:(hodBB.numVertCells - 1)
   for NNcol = 1:(hodBB.numHorizCells - 1)
      BBHists = BBhistogram(NNrow : NNrow + 1, NNcol : NNcol + 1, :);
      magnitBB = norm(BBHists(:)) + 0.01;
      normalized = BBHists / magnitBB;
      HH = [HH; normalized(:)];
   end
end
end


function [IBirds2, IBirds1] = BBlock_compute(IBirds,x_axis_block,y_axis_block, checkx, checky,block_size,
SegmentSize)

  y2=1;
  for y=1:1:block_size
     for x=1:1:block_size
        IBirds1(y_axis_block*(y-1)+1:y_axis_block*y, x_axis_block*(x-1)+1:x_axis_block*x) = IBirds(y_axis_block*(y2-
1)+1:y_axis_block*y2, 1:x_axis_block);
        y2=y2+1;
```

```
    end
  end

  if(x == checkx && y == checky)
          IBirds2 = IBirds1;
       else
          IBirds2 = [IBirds2; IBirds1];
       end
    end
  end

function [stage1,.., stage20, count] = stage_classifier_BB_count(nclassifier,classifer_length,Birdimg, intImg, object)

count = 1;

stage1 = selectedClassifiers(classifer_length(1):classifer_length(2),:);
stage2 = selectedClassifiers(classifer_length(3):classifer_length(4),:);
stage3 = selectedClassifiers(classifer_length(5):classifer_length(6),:);
stage4 = selectedClassifiers(classifer_length(7):classifer_length(8),:);
stage5 = selectedClassifiers(classifer_length(9):classifer_length(10),:);
stage6 = selectedClassifiers(classifer_length(11):classifer_length(12),:);
stage7 = selectedClassifiers(classifer_length(13):classifer_length(14),:);
stage8 = selectedClassifiers(classifer_length(15):classifer_length(16),:);
stage9 = selectedClassifiers(classifer_length(17):classifer_length(18),:);
stage10 = selectedClassifiers(classifer_length(19):classifer_length(20),:);
stage11 = selectedClassifiers(classifer_length(21):classifer_length(22),:);
stage12 = selectedClassifiers(classifer_length(23):classifer_length(24),:);
stage13 = selectedClassifiers(classifer_length(25):classifer_length(26),:);
stage14 = selectedClassifiers(classifer_length(27):classifer_length(28),:);
stage15 = selectedClassifiers(classifer_length(29):classifer_length(30),:);
stage16 = selectedClassifiers(classifer_length(31):classifer_length(32),:);
stage17 = selectedClassifiers(classifer_length(33):classifer_length(34),:);
stage18 = selectedClassifiers(classifer_length(35):classifer_length(36),:);
stage19 = selectedClassifiers(classifer_length(37):classifer_length(38),:);
stage20 = selectedClassifiers(classifer_length(39):classifer_length(40),:);


window = intImg(i:i+xx,j:j+xx);
   check1 = cascade(stage1,window,1);
if check1 == 1 && count <= nclassifier
   check2 = cascade(stage2,window,.5); count=+1;
if check2 == 1 && count <= nclassifier
   check3 = cascade(stage3,window,.5); count=+1;
if check3 == 1 && count <= nclassifier
  check4 = cascade(stage4,window,.5); count=+1;
if check4 == 1 && count <= nclassifier
   check5 = cascade(stage5,window,.6); count=+1;
if check5 == 1 && count <= nclassifier
   check6 = cascade(stage6,window,.6); count=+1;
if check6 == 1 && count <= nclassifier
   check7 = cascade(stage7,window,.5); count=+1;
if check7 == 1  && count <= nclassifier
```

```
   check8 = cascade(stage8,window,.5); count=+1;
if check8 == 1 && count <= nclassifier
   check9 = cascade(stage9,window,.5); count=+1;
 if check9 == 1 && count <= nclassifier
   check10 = cascade(stage10,window,.5); count=+1;
 if check10 == 1 && count <= nclassifier
   check11 = cascade(stage11,window,.6); count=+1;
 if check11 == 1 && count <= nclassifier
   check12 = cascade(stage12,window,.6); count=+1;
 if check12 == 1 && count <= nclassifier
   check13 = cascade(stage13,window,.5); count=+1;
 if check13 == 1 && count <= nclassifier
   check14 = cascade(stage14,window,.5); count=+1;
 if check14 == 1 && count <= nclassifier
   check15 = cascade(stage15,window,.5); count=+1;
 if check15 == 1 && count <= nclassifier
     check16 = cascade(stage16,window,.6); count=+1;
  if check16 == 1 && count <= nclassifier
     check17 = cascade(stage17,window,.6); count=+1;
  if check17 == 1 && count <= nclassifier
     check18 = cascade(stage18,window,.5); count=+1;
  if check18 == 1 && count <= nclassifier
     check19 = cascade(stage19,window,.5); count=+1;
   if check19 == 1 && count <= nclassifier
     check20 = cascade(stage20,window,.5); count=+1;
  end

 if count == nclassifier
      bounds = [j,i,j+xx,i+xx,itr];
      object = [object; bounds];
 end

tempImg = imresize(Birdimg,.8);
img = tempImg;
[m,n] = size(Birdimg);
BirdintImg = integralImg(Birdimg);

function [AA, y_axis_block, x_axis_block] = matrixblockcount(path, Idetector, detector, block_size, SegmentSize,
            Splits, NSplits,rotation,loop_ss)

I = imread(path);

count=1;
for kk = 1:1:loop_ss

  bboxes= BBird_db(Idetector,I);
  Ix = imrotate_image_block(I,(kk-1)*SegmentSize);

  yx = size(I); %

  y_axis_block = yx(1)/block_size;
  x_axis_block = yx(2)/block_size;
```

```
  AA = initializing_matrix_block(block_size,block_size,NSplits);

  for y=1:1:block_size
     for x=1:1:block_size

        I_array = rotation_split(y_axis_block, x_axis_block, block_size, x, y, I, Ix, Splits);

        Bboxes_array= BBird_db(Idetector,I_array);

        Dnum_array  = size_bboxes(Bboxes_array);

        if (Dnum_array >= 1)
           AA(y,x,1)=1;

           for j = 1:1:rotation
              IG = imrotate(I2,j-1);
              dbboxes= BBird_db2(detector,IG);

              snum  = size(dbboxes);
                   if (snum(1)>= 1 )
                      AA(y,x,NSplitsxx)=1;
                       break;

           end
           snum4  = size(dbboxes);

                  if (snum4(1)>= 1)
                      AA(y,x,NSplitsxx)=1;
                       break;

           for j = 1:1:rotation
              IG2 = imrotate(I2,(-j));
              dbboxes2= BBird_db2(detector,IG2);
              snum5  = size(dbboxes2);

                  if (snum5(1)>= 1)
                      AA(y,x,NSplitsxx)=1;
                       break;
                   end

           count=count+1;
        end

  y_axis_block= ddcompute(y_axis_block, block_size);
  x_axis_block= ddcompute(x_axis_block, block_size);

end
```

# APPENDIX B. MAC CODE

4 NCL_* files can be downloaded from
https://www.ndsu.edu/pubweb/~scotsmit/CCLI_async.html;

```
library IEEE;
use ieee.std_logic_1164.all;
use work.ncl_signals.all;

entity demux is
  port (a: IN dual_rail_logic;
      rst: IN std_logic;
           ki1, ki2: IN std_logic;
      s1, s2: IN std_logic;
      z1, z2: OUT dual_rail_logic;
           ko: OUT std_logic);
end demux;

architecture arch of demux is
      signal t1, t2: dual_rail_logic;

      component th33nx0
        port(a: in std_logic;
           b: in std_logic;
           c: in std_logic;
           rst: in std_logic;
           z: out std_logic);
      end component;

      component th14bx0
        port(a: in std_logic;
           b: in std_logic;
           c: in std_logic;
           d: in std_logic;
           zb: out std_logic);
      end component;

begin
  i11: th33nx0
      port map(a.rail1, s1, ki1, rst, t1.rail1);

  i10: th33nx0
      port map(a.rail0, s1, ki1, rst, t1.rail0);
      z1 <= t1;

  i21: th33nx0
      port map(a.rail1, s2, ki2, rst, t2.rail1);

  i20: th33nx0
      port map(a.rail0, s2, ki2, rst, t2.rail0);
      z2 <= t2;
```

```
  k0: th14bx0
      port map(t1.rail1, t1.rail0, t2.rail1, t2.rail0, ko);

end arch;

use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity selct is
  port (ki, rst: IN std_logic;
       s1, s2: OUT std_logic);
end selct;

architecture arch of selct is
            signal d0, d1, d2, d3, r0, r1, r2, r3: std_logic;

      component th33nx0
        port(a: in std_logic;
            b: in std_logic;
            c: in std_logic;
            rst: in std_logic;
            z: out std_logic);
      end component;

      component th33dx0
        port(a: in std_logic;
            b: in std_logic;
            c: in std_logic;
            rst: in std_logic;
            z: out std_logic);
      end component;

            component invx0
        port(i: in std_logic;
            zb: out std_logic);
      end component;
begin

       g0: th33nx0
            port map(ki, d3, r1, rst, d0);

      g1: th33dx0
            port map(ki, d0, r2, rst, d1);

      g2: th33nx0
            port map(ki, d1, r3, rst, d2);

      g3: th33nx0
            port map(ki, d2, r0, rst, d3);

      i0: invx0
            port map(d0, r0);
```

```
	i1: invx0
		port map(d1, r1);
	i2: invx0
		port map(d2, r2);
	i3: invx0
		port map(d3, r3);

		s1 <= d2;
		s2 <= d0;
end arch;

library ieee;
use ieee.std_logic_1164.all;
use work.ncl_signals.all;

entity mux is
  generic(width: in integer := 1);
  port(a1, a2: IN dual_rail_logic_vector(width-1 downto 0);
      z: OUT dual_rail_logic_vector(width-1 downto 0));
end mux;

architecture arch of mux is

  component th12x0
		port (a: IN std_logic;
		b: IN std_logic;
		z: OUT std_logic);
  end component;

begin
		struct: for i in a1'range generate
			comp0: th12x0
				port map(a1(i).rail0, a2(i).rail0, z(i).rail0);

			comp1: th12x0
		      port map(a1(i).rail1, a2(i).rail1, z(i).rail1);
		end generate struct;

end arch;

-- ------------------------------------------------------------------
--   modification history :
-- ------------------------------------------------------------------
-- ------------------------------------------------------------------
-- ncl_combinational_synth.vhd
-- use instantiatiate gate during 2nd pass of synthesis
-----------------------------------------^M
-- Definition of inv1
---------------------------------------^M
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;
```

```
entity inv1 is
  port (z: IN dual_rail_logic ;
       nz: OUT dual_rail_logic);
end inv1;

architecture arch of inv1 is
begin
            nz.rail1 <= z.rail0;
            nz.rail0 <= z.rail1;
end arch;

----------------------------------------^M
-- Definition of xor2
----------------------------------------^M
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity xor2 is
  port (a: IN dual_rail_logic ;
       b: IN dual_rail_logic ;
       z: OUT dual_rail_logic);
end xor2;

architecture arch of xor2 is
            component thxor0x0
               port(a: in std_logic;
            b: in std_logic;
            c: in std_logic;
            d: in std_logic;
            z: out std_logic);
            end component;
begin
  g0: thxor0x0 port map(
                        a => a.rail0,
            b => a.rail1,
            c => b.rail0,
            d => b.rail1,
            z => z.rail0);

  g1: thxor0x0 port map(
            a => a.rail0,
            b => b.rail0,
            c => b.rail1,
            d => a.rail1,
            z => z.rail1);
end arch;
----------------------------------------^M
-- Definition of and2
----------------------------------------^M
use work.ncl_signals.all;
library ieee;
```

```
use ieee.std_logic_1164.all;

entity and2 is
  port (a: IN dual_rail_logic ;
        b: IN dual_rail_logic ;
        z: OUT dual_rail_logic);--*
end and2;

architecture arch of and2 is
      component thand0x0
         port(a: in std_logic;
              b: in std_logic;
              c: in std_logic;
              d: in std_logic;
              z: out std_logic);--*
      end component;

      component th22x0
        port(a: in std_logic;
             b: in std_logic;
             z: out std_logic);--*
      end component;
begin
  g0: thand0x0 port map(
              a.rail0,
              b.rail0,
              a.rail1,
              b.rail1,
              z.rail0);--#

  g1: th22x0 port map(
              a.rail1,
              b.rail1,
              z.rail1);--#
end arch;

-----------------------------------------^M
-- Definition of shift_comp_m_low for MAC
-----------------------------------------^M
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity shift_comp_m_low is
      port(Pu, Pp, AS, Sh, Sl: in DUAL_RAIL_LOGIC;
         P: out DUAL_RAIL_LOGIC);
end;

architecture arch of shift_comp_m_low is
      signal a1, a2, b1, b2, b3: std_logic;

      component th24w22x0
         port(a: in std_logic;
```

```
            b: in std_logic;
            c: in std_logic;
            d: in std_logic;
            z: out std_logic );
      end component;

      component thxor0x0
         port(a: in std_logic;
            b: in std_logic;
            c: in std_logic;
            d: in std_logic;
            z: out std_logic);
      end component;

      component th33x0
         port(a: in std_logic;
            b: in std_logic;
            c: in std_logic;
            z: out std_logic );
      end component;

      component th54w32x0
         port(a: in std_logic;
            b: in std_logic;
            c: in std_logic;
            d: in std_logic;
            z: out std_logic );
      end component;
begin

   g0: thxor0x0
      port map(Pu.rail0, Pu.rail1, AS.rail1, AS.rail0, a1);

   g1: thxor0x0
      port map(Pp.rail1, Pp.rail0, AS.rail0, AS.rail1, a2);

   g2: thxor0x0
      port map(Sh.rail1, Sl.rail1, a1, a2, P.rail1);

   g3: thxor0x0
      port map(Pu.rail0, Pu.rail1, AS.rail0, AS.rail1, b1);

   g4: th33x0
      port map(Sh.rail0, Pp.rail0, AS.rail0, b2);

   g5: th54w32x0
            port map(Sh.rail0, Sl.rail0, Pp.rail1, AS.rail1, b3);

   g6: th24w22x0
            port map(b3, b2, b1, Sl.rail0, P.rail0);

end arch;
----------------------------------------^M
```

```
-- Definition of shift_comp_low for MAC
-----------------------------------------^M
--
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity shift_comp_low is
      port(Pu, AS, Sh, Sl: in DUAL_RAIL_LOGIC;
        P: out DUAL_RAIL_LOGIC);
end;

architecture arch of shift_comp_low is
      signal a1, b1: std_logic;

      component thxor0x0
        port(a: in std_logic;
           b: in std_logic;
           c: in std_logic;
           d: in std_logic;
           z: out std_logic);
      end component;

      component th12x0
        port(a: in std_logic;
           b: in std_logic;
           z: out std_logic );
      end component;

      component th22x0
        port(a: in std_logic;
           b: in std_logic;
           z: out std_logic );
      end component;
begin

   g0: thxor0x0
           port map(Pu.rail0, Pu.rail1, AS.rail1, AS.rail0, a1);

   g1: th22x0
           port map(Sh.rail1, a1, P.rail1);

   g2: thxor0x0
           port map(Pu.rail0, Pu.rail1, AS.rail0, AS.rail1, b1);

   g3: th12x0
           port map(b1, Sh.rail0, P.rail0);

end arch;
-----------------------------------------^M
-- Definition of and2i          (incomplete AND)
-----------------------------------------^M
--
```

```
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity and2i is
  port (a: IN dual_rail_logic ;
        b: IN dual_rail_logic ;
        z: OUT dual_rail_logic);
end and2i;

architecture arch of and2i is
        component th12x0
         port (
            a   : IN  std_logic ;
            b   : IN  std_logic ;
            z   : OUT std_logic) ;
       end component ;

       component th22x0
         port(a: in std_logic;
            b: in std_logic;
            z: out std_logic);
       end component;
begin
  g0: th12x0 port map(
                        a.rail0,
                        b.rail0,
            z.rail0);

  g1: th22x0 port map(
            a => a.rail1,
            b => b.rail1,
            z => z.rail1);
end arch;
-----------------------------------------^M
-- Definition of nand2i  (incomplete NAND)
-----------------------------------------^M
--
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity nand2i is
  port (a: IN dual_rail_logic ;
        b: IN dual_rail_logic ;
        z: OUT dual_rail_logic);
end nand2i;

architecture arch of nand2i is
       component th12x0
         port (
            a   : IN  std_logic ;
            b   : IN  std_logic ;
```

```
            z   : OUT std_logic) ;
      end component ;

      component th22x0
        port(a: in std_logic;
            b: in std_logic;
            z: out std_logic);
      end component;
begin
  g0: th12x0 port map(
            a.rail1,
            b.rail1,
            z.rail0);

  g1: th22x0 port map(
            a => a.rail0,
            b => b.rail0,
            z => z.rail1);
end arch;
-----------------------------------------^M
-- Definition of sign_ss
-----------------------------------------^M
--
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity sign_ss is
   port (AS, Sh, Sl: in DUAL_RAIL_LOGIC;
       ShSlAS: out DUAL_RAIL_LOGIC);
end;

architecture arch of sign_ss is

      component th33x0
        port(a: in std_logic;
            b: in std_logic;
            c: in std_logic;
            z: out std_logic);
      end component;

      component th13x0
        port(a: in std_logic;
            b: in std_logic;
            c: in std_logic;
            z: out std_logic);
      end component;

begin
  g0: th13x0
            port map(AS.rail0, Sh.rail1, Sl.rail1, ShSlAS.rail0);

  g1: th33x0
```

```
      port map(AS.rail1, Sh.rail0, Sl.rail0, ShSlAS.rail1);

end arch;
----------------------------------------^M
-- Definition of e0 for MAC
----------------------------------------^M
--
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity e0 is
      port(A0, AS, MM: in DUAL_RAIL_LOGIC;
         e_0: out DUAL_RAIL_LOGIC);
end;

architecture arch of e0 is
            signal both1, one0: std_logic;

      component th12x0
        port (
            a   : IN  std_logic ;
            b   : IN  std_logic ;
            z   : OUT std_logic) ;
      end component ;

      component th22x0
        port(a: in std_logic;
            b: in std_logic;
            z: out std_logic);
      end component;

            component th24w22x0
               port(a: in std_logic; -- weight 2^M
            b: in std_logic; -- weight 2^M
            c: in std_logic;
            d: in std_logic;
            z: out std_logic );
            end component;

            component th54w22x0
               port(a: in std_logic; -- weight 2^M
            b: in std_logic; -- weight 2^M
            c: in std_logic;
            d: in std_logic;
            z: out std_logic );
            end component;

begin
            g0: th22x0
                        port map(AS.rail1, A0.rail1, both1);

            g1: th12x0
```

```
                port map(AS.rail0, A0.rail0, one0);

        g2: th24w22x0
                port map(both1, MM.rail1, AS.rail0, A0.rail0, e_0.rail0);

        g3: th54w22x0
                port map(one0, MM.rail0, AS.rail1, A0.rail1, e_0.rail1);
end arch;
-----------------------------------------^M
-- Definition of calc_MS for MAC
-----------------------------------------^M
--
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity calc_MS is
    port(x31, y31, Sh, Sl: in DUAL_RAIL_LOGIC;
         MS: out DUAL_RAIL_LOGIC);
end;

architecture arch of calc_MS is
        signal T1, T2, T3, T4, T5, T6: std_logic;

    component th33x0
      port(a: in std_logic;
        b: in std_logic;
        c: in std_logic;
        z: out std_logic );
    end component;

    component th22x0
      port(a: in std_logic;
        b: in std_logic;
        z: out std_logic );
    end component;

    component th44x0
      port(a: in std_logic;
        b: in std_logic;
        c: in std_logic;
        d: in std_logic;
        z: out std_logic );
    end component;

    component th13x0
      port(a: in std_logic;
        b: in std_logic;
        c: in std_logic;
        z: out std_logic );
    end component;

    component th14x0
```

```
        port(a: in std_logic;
           b: in std_logic;
           c: in std_logic;
           d: in std_logic;
           z: out std_logic );
      end component;
begin
           g0: th22x0
                     port map(Sl.rail1, y31.rail1, T1);

           g1: th33x0
                     port map(Sh.rail0, x31.rail0, y31.rail1, T2);

           g2: th44x0
                     port map(Sh.rail0, Sl.rail0, x31.rail1, y31.rail0, T3);

           g3: th22x0
                     port map(x31.rail0, y31.rail0, T4);

           g4: th22x0
                     port map(Sl.rail1, y31.rail0, T5);

           g5: th33x0
                     port map(Sl.rail0, x31.rail1, y31.rail1, T6);

           g6: th13x0
                     port map(T1, T2, T3, MS.rail1);

           g7: th14x0
                     port map(Sh.rail1, T4, T5, T6, MS.rail0);
end arch;

-----------------------------------------^M
-- Definition of completeness for MAC
-----------------------------------------^M
--
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity completeness is
      port(round, rnd_type, SCh, SCl, saturate, add_sub, mac_mpy, Sh, Sl: in DUAL_RAIL_LOGIC;
         complete: out DUAL_RAIL_LOGIC);
end;

architecture arch of completeness is
           signal T1: std_logic;
           signal k: std_logic_vector(4 downto 0);

      component th24compx0
         port(a0: in std_logic;
           a1: in std_logic;
           b0: in std_logic;
```

```
            b1: in std_logic;
            z: out std_logic );
        end component;

        component th44x0
           port(a: in std_logic;
            b: in std_logic;
            c: in std_logic;
            d: in std_logic;
            z: out std_logic );
        end component;

        component th12x0
           port(a: in std_logic;
            b: in std_logic;
            z: out std_logic );
        end component;

        component th22x0
           port(a: in std_logic;
            b: in std_logic;
            z: out std_logic );
        end component;

begin
        t0: th24compx0
            port map(round.rail0, round.rail1, rnd_type.rail0, rnd_type.rail1, k(0));

        t2: th24compx0
            port map(SCh.rail0, SCh.rail1, SCl.rail0, SCl.rail1, k(1));

        t4: th24compx0
            port map(saturate.rail0, saturate.rail1, add_sub.rail0, add_sub.rail1, k(2));

        t6: th24compx0
            port map(mac_mpy.rail0, mac_mpy.rail1, Sh.rail0, Sh.rail1, k(3));

            t9: th12x0
                    port map(Sl.rail1, Sl.rail0, k(4));

            g0: th44x0
                    port map(k(0), k(1), k(2), k(3), T1);

            g1: th22x0
                    port map(T1, k(4), complete.rail1);

            complete.rail0 <= '0';
end arch;

----------------------------------------^M
-- Definition of mult_only for MAC
----------------------------------------^M
--
```

```
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity mult_only is
      port(Ain_i, MM: in DUAL_RAIL_LOGIC;
         Ac_i: out DUAL_RAIL_LOGIC);
end;

architecture arch of mult_only is

      component th23w2x0
         port(a: in std_logic;
            b: in std_logic;
            c: in std_logic;
            z: out std_logic );
      end component;

      component th22x0
         port(a: in std_logic;
            b: in std_logic;
            z: out std_logic );
      end component;

begin

   g1: th22x0
            port map(Ain_i.rail1, MM.rail0, Ac_i.rail1);

   g2: th23w2x0
            port map(Ain_i.rail0, Ain_i.rail1, MM.rail1, Ac_i.rail0);

end;

----------------------------------------^M
-- Definition of shift_comp for MAC
----------------------------------------^M
--
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity shift_comp is
      port(Pu, Pp, Pl, AS, Sh, Sl: in DUAL_RAIL_LOGIC;
         P: out DUAL_RAIL_LOGIC);
end;

architecture arch of shift_comp is
            signal T1_1, T0_1, T0_0, T1_0, U1, U0, P1, P0, L1, L0: std_logic;

      component th33x0
         port(a: in std_logic;
            b: in std_logic;
```

```
        c: in std_logic;
        z: out std_logic );
    end component;

    component th22x0
      port(a: in std_logic;
        b: in std_logic;
        z: out std_logic );
    end component;

    component th44w3x0
      port(a: in std_logic;
        b: in std_logic;
        c: in std_logic;
        d: in std_logic;
        z: out std_logic );
    end component;

    component th12x0
      port(a: in std_logic;
        b: in std_logic;
        z: out std_logic );
    end component;
begin

        g2: th22x0
                port map(Sh.rail1, Pu.rail1, U1);

        g3: th22x0
                port map(Sl.rail1, Pp.rail1, P1);

        g4: th22x0
                port map(Sh.rail1, Pu.rail0, U0);

        g5: th22x0
                port map(Sl.rail1, Pp.rail0, P0);

        g6: th33x0
                port map(Sh.rail0, Sl.rail0, Pl.rail1, L1);

        g7: th33x0
                port map(Sh.rail0, Sl.rail0, Pl.rail0, L0);

        g8: th44w3x0
                port map(AS.rail0, U1, P1, L1, T1_1);

        g9: th44w3x0
                port map(AS.rail1, U0, P0, L0, T0_1);

        g10: th44w3x0
                port map(AS.rail0, U0, P0, L0, T0_0);

        g11: th44w3x0
```

```
                port map(AS.rail1, U1, P1, L1, T1_0);

        g12: th12x0
                port map(T1_1, T0_1, P.rail1);

        g13: th12x0
                port map(T0_0, T1_0, P.rail0);
end arch;
-----------------------------------------^M
-- Definition of shift_comp_high for MAC
-----------------------------------------^M
--
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity shift_comp_high is
    port(Pl, AS, Sh, Sl: in DUAL_RAIL_LOGIC;
      P: out DUAL_RAIL_LOGIC);
end;

architecture arch of shift_comp_high is
    signal a1, a2, b1, b2: std_logic;

    component th44w3x0
      port(a: in std_logic;
        b: in std_logic;
        c: in std_logic;
        d: in std_logic;
        z: out std_logic );
    end component;

    component th12x0
      port(a: in std_logic;
        b: in std_logic;
        z: out std_logic );
    end component;

    component th44x0
      port(a: in std_logic;
        b: in std_logic;
        c: in std_logic;
        d: in std_logic;
        z: out std_logic );
    end component;
begin

  g0: th44x0
    port map(Sh.rail0, Sl.rail0, Pl.rail1, AS.rail1, a1);

  g1: th44w3x0
    port map(AS.rail0, Sl.rail1, Sh.rail1, Pl.rail0, a2);
```

```
   g2: th12x0
      port map(a1, a2, P.rail0);

   g3: th44x0
      port map(Sh.rail0, Sl.rail0, Pl.rail1, AS.rail0, b1);

   g4: th44w3x0
      port map(AS.rail1, Sl.rail1, Sh.rail1, Pl.rail0, b2);

   g5: th12x0
      port map(b1, b2, P.rail1);

end arch;
-----------------------------------------^M
-- Definition of shift_comp_m_high for MAC
-----------------------------------------^M
--
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity shift_comp_m_high is
      port(Pp, Pl, AS, Sh, Sl: in DUAL_RAIL_LOGIC;
         P: out DUAL_RAIL_LOGIC);
end;

architecture arch of shift_comp_m_high is
      signal a1, a2, a3, a4, b1, b2, b3, b4: std_logic;

      component th54w32x0
         port(a: in std_logic;
            b: in std_logic;
            c: in std_logic;
            d: in std_logic;
            z: out std_logic );
      end component;

      component th14x0
         port(a: in std_logic;
            b: in std_logic;
            c: in std_logic;
            d: in std_logic;
            z: out std_logic );
      end component;

      component th44x0
         port(a: in std_logic;
            b: in std_logic;
            c: in std_logic;
            d: in std_logic;
            z: out std_logic );
      end component;
```

```
      component th33x0
         port(a: in std_logic;
            b: in std_logic;
            c: in std_logic;
            z: out std_logic );
      end component;
begin

   g0: th44x0
      port map(Sh.rail0, Sl.rail0, Pl.rail1, AS.rail1, a1);

   g1: th54w32x0
      port map(AS.rail0, Sh.rail1, Sl.rail1, Pp.rail0, a2);

   g2: th33x0
            port map(Sl.rail0, Pl.rail0, AS.rail0, a3);

   g3: th33x0
      port map(Sl.rail1, Pp.rail1, AS.rail1, a4);

   g4: th14x0
      port map(a1, a2, a3, a4, P.rail0);

   g5: th44x0
      port map(Sh.rail0, Sl.rail0, Pl.rail1, AS.rail0, b1);

   g6: th54w32x0
      port map(AS.rail1, Sh.rail1, Sl.rail1, Pp.rail0, b2);

   g7: th33x0
      port map(Sl.rail0, Pl.rail0, AS.rail1, b3);

   g8: th33x0
      port map(Sl.rail1, Pp.rail1, AS.rail0, b4);

   g9: th14x0
      port map(b1, b2, b3, b4, P.rail1);

end arch;

-----------------------------------------^M
-- Definition of MSB_31x31 for MAC
-----------------------------------------^M
--
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity MSB_31x31 is
      port(x, y, Sl: in DUAL_RAIL_LOGIC;
         pp1023: out DUAL_RAIL_LOGIC);
end;
```

```
architecture arch of MSB_31x31 is
        signal T1, T2, T3, T4, T5, T6: std_logic;

    component th34x0
      port(a: in std_logic;
        b: in std_logic;
        c: in std_logic;
        d: in std_logic;
        z: out std_logic );
    end component;

    component th33x0
      port(a: in std_logic;
        b: in std_logic;
        c: in std_logic;
        z: out std_logic );
    end component;

    component th13x0
      port(a: in std_logic;
        b: in std_logic;
        c: in std_logic;
        z: out std_logic );
    end component;

begin
        g0: th34x0
                port map(Sl.rail1, x.rail0, y.rail0, y.rail1, T1);

        g1: th34x0
                port map(Sl.rail1, y.rail0, x.rail1, x.rail0, T2);

        g2: th33x0
                port map(Sl.rail0, x.rail1, y.rail1, T3);

        g4: th34x0
        port map(Sl.rail0, x.rail0, y.rail0, y.rail1, T4);

         g5: th34x0
                port map(Sl.rail0, y.rail0, x.rail1, x.rail0, T5);

        g6: th33x0
                port map(Sl.rail1, x.rail1, y.rail1, T6);

        g7: th13x0
                port map(T1, T2, T3, pp1023.rail1);

        g8: th13x0
                port map(T4, T5, T6, pp1023.rail0);
end arch;

-----------------------------------------^M
-- Definition of LR_31x0_30 for MAC
```

```
----------------------------------------^M
--
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity LR_31x0_30 is
      port(x, y, Sh, Sl: in DUAL_RAIL_LOGIC;
         pp: out DUAL_RAIL_LOGIC);
end;

architecture arch of LR_31x0_30 is
            signal T0_1, T1_1, H0L0, H1, L1: std_logic;

            component th44x0
          port(a: in std_logic;
            b: in std_logic;
            c: in std_logic;
            d: in std_logic;
            z: out std_logic);
      end component;

      component th54w22x0
        port(a: in std_logic;
            b: in std_logic;
            c: in std_logic;
            d: in std_logic;
            z: out std_logic);
      end component;

      component th33w2x0
        port(a: in std_logic;
            b: in std_logic;
            c: in std_logic;
            z: out std_logic);
      end component;

      component th12x0
        port(a: in std_logic;
            b: in std_logic;
            z: out std_logic);
      end component;

      component th13x0
        port(a: in std_logic;
            b: in std_logic;
            c: in std_logic;
            z: out std_logic);
      end component;
begin
            g0: th54w22x0
                        port map(Sh.rail0, Sl.rail0, x.rail0, y.rail0, T0_1);
```

```
        g1: th54w22x0
                port map(x.rail1, y.rail1, Sh.rail1, Sl.rail1, T1_1);

        g2: th44x0
                port map(Sh.rail0, Sl.rail0, x.rail1, y.rail1, H0L0);

        g3: th33w2x0
                port map(Sh.rail1, x.rail0, y.rail0, H1);

        g4: th33w2x0
                port map(Sl.rail1, x.rail0, y.rail0, L1);

        g5: th12x0
                port map(T0_1, T1_1, pp.rail1);

        g6: th13x0
                port map(H0L0, H1, L1, pp.rail0);
end arch;

-----------------------------------------^M
-- Definition of MSB_0_30_31 for MAC
-----------------------------------------^M
--
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity MSB_0_30x31 is
    port(x, y, Sh: in DUAL_RAIL_LOGIC;
       pp: out DUAL_RAIL_LOGIC);
end;

architecture arch of MSB_0_30x31 is
        signal T0_1, T1_1, T0_0, T1_0: std_logic;

    component th33x0
      port(a: in std_logic;
        b: in std_logic;
        c: in std_logic;
        z: out std_logic);
    end component;

    component th33w2x0
      port(a: in std_logic;
        b: in std_logic;
        c: in std_logic;
        z: out std_logic);
    end component;

    component th12x0
      port(a: in std_logic;
        b: in std_logic;
        z: out std_logic);
```

```
      end component;
begin
            g0: th33w2x0
                  port map(Sh.rail0, x.rail0, y.rail0, T0_1);

            g1: th33x0
                  port map(Sh.rail1, x.rail1, y.rail1, T1_1);

            g2: th33w2x0
                  port map(Sh.rail1, x.rail0, y.rail0, T0_0);

            g3:th33x0
                  port map(Sh.rail0, x.rail1, y.rail1, T1_0);

            g4: th12x0
                  port map(T0_1, T1_1, pp.rail1);

            g5: th12x0
                  port map(T0_0, T1_0, pp.rail0);
end arch;

-----------------------------------------^M
-- Definition of or2
-----------------------------------------^M
--
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity or2 is
  port (a: IN dual_rail_logic ;
       b: IN dual_rail_logic ;
       z: OUT dual_rail_logic);
end or2;

architecture arch of or2 is
      component thand0x0
         port(a0: in std_logic;
            a1: in std_logic;
            b0: in std_logic;
            b1: in std_logic;
            z: out std_logic);
      end component;

      component th22x0
        port(a: in std_logic;
            b: in std_logic;
            z: out std_logic);
      end component;
begin
  g0: thand0x0 port map(
            a0 => a.rail1,
            a1 => a.rail0,
```

```
            b0 => b.rail1,
            b1 => b.rail0,
            z => z.rail1);

  g1: th22x0 port map(
            a => a.rail0,
            b => b.rail0,
            z => z.rail0);
end arch;

----------------------------------------^M
-- Definition of xor2_2
----------------------------------------^M
--
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity xor2_2 is
  port (a: IN dual_rail_logic ;
        b: IN dual_rail_logic ;
        z: OUT dual_rail_logic);
end xor2_2;

architecture arch of xor2_2 is
      signal T1, T2, T3, T4: std_logic;

      component th12x0
         port(a: in std_logic;
            b: in std_logic;
            z: out std_logic);
      end component;

      component th22x0
        port(a: in std_logic;
            b: in std_logic;
            z: out std_logic);
      end component;
begin
  g0: th22x0
      port map(a.rail0, b.rail0, T1);

  g1: th22x0
      port map(a.rail0, b.rail1, T2);

  g2: th22x0
      port map(a.rail1, b.rail0, T3);

  g3: th22x0
      port map(a.rail1, b.rail1, T4);

  g4: th12x0
      port map(T1, T4, z.rail0);
```

```
  g5: th12x0
      port map(T2, T3, z.rail1);
end arch;

-----------------------------------------^M
-- Definition of nor2i          (incomplete NOR)
-----------------------------------------^M
--
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity nor2i is
  port (a: IN dual_rail_logic ;
        b: IN dual_rail_logic ;
        z: OUT dual_rail_logic);
end nor2i;

architecture arch of nor2i is
      component th12x0
        port (
            a   : IN  std_logic ;
            b   : IN  std_logic ;
            z   : OUT std_logic) ;
      end component ;

      component th22x0
        port(a: in std_logic;
            b: in std_logic;
            z: out std_logic);
      end component;
begin
  g0: th12x0 port map(
                        a.rail1,
                        b.rail1,
                        z.rail0);

  g1: th22x0 port map(
            a => a.rail0,
            b => b.rail0,
            z => z.rail1);
end arch;

-----------------------------------------^M
-- Definition of nor2
-----------------------------------------^M
--
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity nor2 is
```

```
  port (a: IN dual_rail_logic ;
        b: IN dual_rail_logic ;
        z: OUT dual_rail_logic);
end nor2;

architecture arch of nor2 is
      component thand0x0
         port(a: in std_logic;--a0
            b: in std_logic;--a1
            c: in std_logic;--b0
            d: in std_logic;--b1
            z: out std_logic);
      end component;

      component th22x0
        port(a: in std_logic;
            b: in std_logic;
            z: out std_logic);
      end component;
begin


  g0: thand0x0 port map(
            a.rail1,
            a.rail0,
            b.rail1,
            b.rail0,
            z.rail0);

  g1: th22x0 port map(
            a => a.rail0,
            b => b.rail0,
            z => z.rail1);

end arch;

-- ----------------------------------------------------------------------
-- Definition of ncl_register
-- ----------------------------------------------------------------------
--
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity ncl_register is
   generic ( width : integer := 2 ;
          initial_value : integer := 2
   ) ;
   port (
      data_in  : in dual_rail_logic_vector(width-1 downto 0 ) ;
            ki      : in dual_rail_logic_vector(width-1 downto 0 ) ;
      rst     : in dual_rail_logic ;
      data_out : out dual_rail_logic_vector(width-1 downto 0 ) ;
```

```
      ko      : out dual_rail_logic_vector(width-1 downto 0 )
   ) ;
end ncl_register ;

architecture syn of ncl_register is
component thla12nx0 port (
       a0   : IN  std_logic ;
       a1   : IN  std_logic ;
       rst  : IN  std_logic ;
       ki   : IN  std_logic ;
       z0   : OUT std_logic ;
       z1   : OUT std_logic ;
       ko   : OUT std_logic ) ;
end component ;

component thla12dx0 port (
       a0   : IN  std_logic ;
       a1   : IN  std_logic ;
       rst  : IN  std_logic ;
       ki   : IN  std_logic ;
       z0   : OUT std_logic ;
       z1   : OUT std_logic ;
       ko   : OUT std_logic ) ;
end component ;

begin
     reg_file : for idx in data_in'range generate
--     begin
       ila1n: if initial_value = 2 generate          -- NULL
--       begin -- null
          reg: thla12nx0 port map (
             data_in(idx).rail0, data_in(idx).rail1, rst.rail1, ki(idx).rail1,
                                 data_out(idx).rail0, data_out(idx).rail1, ko(idx).rail1 );
       end generate ila1n;
       ila1r: if initial_value = 0 generate          -- DATA0
--       begin
          reg: thla12dx0 port map (
             data_in(idx).rail0, data_in(idx).rail1, rst.rail1, ki(idx).rail1,
                                 data_out(idx).rail0, data_out(idx).rail1, ko(idx).rail1 );
       end generate ila1r;
       ila1s: if initial_value = 1 generate          -- DATA1
--       begin
          reg: thla12dx0 port map (
             data_in(idx).rail1, data_in(idx).rail0, rst.rail1, ki(idx).rail1,
                                 data_out(idx).rail1, data_out(idx).rail0, ko(idx).rail1 );
       end generate ila1s;
            ko(idx).rail0 <= '0';

    end generate reg_file;
end syn;

-----------------------------------------^M
-- Definition of  full_add
```

```
----------------------------------------^M
--
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity full_add is
  port (
       c_in : IN dual_rail_logic ;
       x: IN dual_rail_logic ;
       y: IN dual_rail_logic ;
       c_out: OUT dual_rail_logic ;
       s     : OUT dual_rail_logic );--*
end full_add ;

architecture archthfax0 of full_add is
      signal s0_buffer: std_logic;
      signal s1_buffer: std_logic;
      signal y0_buffer: std_logic;
      signal y1_buffer: std_logic;

      component th23x0
            port(a: in std_logic;
                b: in std_logic;
                   c: in std_logic;
                   z: out std_logic);--*
      end component;

      component th34w2x0
            port(a: in std_logic;
                b: in std_logic;
                   c: in std_logic;
                   d: in std_logic;
                   z: out std_logic);--*
      end component;

begin
      s.rail0 <= s0_buffer;
      s.rail1 <= s1_buffer;
      c_out.rail0 <= y0_buffer;
      c_out.rail1 <= y1_buffer;
      th0: th23x0
            port map(x.rail0,
                  y.rail0,
                  c_in.rail0,
                  y0_buffer);--#
      th1: th23x0
            port map(x.rail1,
                  y.rail1,
                  c_in.rail1,
                  y1_buffer);--#
      th2: th34w2x0
            port map(y0_buffer,
```

```
                x.rail1,
                y.rail1,
                c_in.rail1,
                s1_buffer);--#

    th3: th34w2x0
        port map(y1_buffer,
                x.rail0,
                y.rail0,
                c_in.rail0,
                s0_buffer);--#

end archthfax0;
-----------------------------------------^M
-- Definition of  half_add
-----------------------------------------^M
--
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity half_add is
  port (
      x: IN dual_rail_logic ;
      y: IN dual_rail_logic ;
      c_out: OUT dual_rail_logic ;
      s     : OUT dual_rail_logic );--*
end half_add ;
architecture arch of half_add is
    component th24compx0
       port(a: in std_logic;
          b: in std_logic;
          c: in std_logic;
          d: in std_logic;
          z: out std_logic );--*
    end component;

    component th12x0
      port (
          a   : IN  std_logic ;
          b   : IN  std_logic ;
          z   : OUT std_logic);--*
    end component ;

    component th22x0
      port(a: in std_logic;
          b: in std_logic;
          z: out std_logic);--*
    end component;
begin
  g0: th12x0 port map(
          x.rail0,
          y.rail0,
```

```
            c_out.rail0);--#

  g1: th22x0 port map(
            x.rail1,
            y.rail1,
            c_out.rail1);--#

  g2: th24compx0 port map(
            x.rail0,
            y.rail1,
            y.rail0,
            x.rail1,
            s.rail0);--#

  g3: th24compx0 port map(
            x.rail0,
            y.rail0,
            y.rail1,
            x.rail1,
            s.rail1);--#
end arch;

use work.ncl_signals.all;
use work.dual_rail.all;
use work.functions.all;
library ieee;
use ieee.std_logic_1164.all;

entity feedforward_stage1_2_3_4 is
  port(FF_in: in DUAL_RAIL_LOGIC_VECTOR(67 downto 0);
                  reset : in STD_LOGIC;
                        ki: in STD_LOGIC_VECTOR(519 downto 0);
                  ko: out STD_LOGIC_VECTOR(67 downto 0);
                  PPC_out: out dual_rail_logic_VECTOR(519 downto 0)); -- width = 139
end;

architecture TEST of feedforward_stage1_2_3_4 is

signal x1, y1: dual_rail_logic_VECTOR(31 downto 0);
signal temp1, temp1_in, temp1_out : dual_rail_logic_VECTOR(67 downto 0);
signal ki_1, ko_1: std_logic_vector(67 downto 0);

signal temp17, temp17_in, temp17_out : dual_rail_logic_VECTOR(1030 downto 0);
signal ki_17, ko_17: std_logic_vector(1030 downto 0);
--------------------------------------------------------------------------------
signal ko_17b1: std_logic_vector(34 downto 0);
  signal ko_17b2,ko_17b3: std_logic_vector(1023 downto 0);
  signal ki1b2: std_logic_vector(31 downto 0);
  signal ki1b3: std_logic_vector(30 downto 0);
  signal ki1: std_logic;
  signal add_sub2, mac_mpy2, e_31, e_32: dual_rail_logic;
  signal sign2: dual_rail_logic_VECTOR(1 downto 0);
--------------------------------------------------------------------------------
```

```
signal temp4, temp4_in, temp4_out: dual_rail_logic_VECTOR(729 downto 0);
signal ki_4, ko_4: std_logic_vector(729 downto 0);
signal ko_4b1: std_logic_vector(32 downto 0);
signal ko_4b2: std_logic_vector(34 downto 0);
signal ko_4b3: std_logic_vector(1 downto 0);
signal ko_4b4: std_logic_vector(1 downto 0);
signal ko_4b5: std_logic_vector(57 downto 0);
signal ko_4b6: std_logic_vector(1 downto 0);
signal ko_4b7: std_logic_vector(1 downto 0);

signal ko_4b8_2,ko_4b8_3,ko_4b8_4,ko_4b8_5,ko_4b8_6,ko_4b8_7,ko_4b8_8,ko_4b8_9,ko_4b8_10:
        std_logic_vector(57 downto 0);
signal ko_4b9: std_logic_vector(17 downto 0);
signal ki17: std_logic;
signal ki17b2,ki17b3,ki17b4: std_logic;
signal ki17b5: std_logic_vector(28 downto 0);
signal ki17b6,ki17b7: std_logic;
signal ki17b8_2,ki17b8_3,ki17b8_4,ki17b8_5,ki17b8_6,ki17b8_7,ki17b8_8,ki17b8_9,ki17b8_10:
        std_logic_vector(28 downto 0);
signal ki17b9: std_logic_vector(8 downto 0);
--------------------------------------------------------------------------------
signal e_31b, e_32b, add_sub2b, mac_mpy2b, MS1, x1_31b: dual_rail_logic;
signal sign2b: dual_rail_logic_VECTOR(1 downto 0);
signal y1b: dual_rail_logic_VECTOR(31 downto 0);
signal pos2o4: dual_rail_logic_VECTOR(1 downto 0);

signal s0_1, c0_1, s0_1r, c0_1r: dual_rail_logic_VECTOR(33 downto 2);
signal s0_2, s0_2r: dual_rail_logic_VECTOR(35 downto 3);
signal c0_2, c0_2r: dual_rail_logic_VECTOR(35 downto 4);
signal s0_3, s0_3r: dual_rail_logic_VECTOR(38 downto 6);
signal c0_3, c0_3r: dual_rail_logic_VECTOR(38 downto 7);
signal s0_4, s0_4r: dual_rail_logic_VECTOR(41 downto 9);
signal c0_4, c0_4r: dual_rail_logic_VECTOR(41 downto 10);
signal s0_5, s0_5r: dual_rail_logic_VECTOR(44 downto 12);
signal c0_5, c0_5r: dual_rail_logic_VECTOR(44 downto 13);
signal s0_6, s0_6r: dual_rail_logic_VECTOR(47 downto 15);
signal c0_6, c0_6r: dual_rail_logic_VECTOR(47 downto 16);
signal s0_7, s0_7r: dual_rail_logic_VECTOR(50 downto 18);
signal c0_7, c0_7r: dual_rail_logic_VECTOR(50 downto 19);
signal s0_8, s0_8r: dual_rail_logic_VECTOR(53 downto 21);
signal c0_8, c0_8r: dual_rail_logic_VECTOR(53 downto 22);
signal s0_9, s0_9r: dual_rail_logic_VECTOR(56 downto 24);
signal c0_9, c0_9r: dual_rail_logic_VECTOR(56 downto 25);
signal s0_10, s0_10r: dual_rail_logic_VECTOR(59 downto 27);
signal c0_10, c0_10r: dual_rail_logic_VECTOR(59 downto 28);
signal pp, pp1: dual_rail_logic_VECTOR(1023 downto 0);
--------------------------------------------------------------------------------
signal temp10, temp10_in, temp10_out: dual_rail_logic_VECTOR(524 downto 5);
signal ki_10, ko_10: std_logic_vector(524 downto 5);

signal s1_1, s1_1r: dual_rail_logic_VECTOR(35 downto 3);
 signal c1_1, c1_1r: dual_rail_logic_VECTOR(36 downto 3);
      signal s1_2, s1_2r: dual_rail_logic_VECTOR(38 downto 4);
```

```
    signal c1_2, c1_2r: dual_rail_logic_VECTOR(39 downto 6);
    signal s1_3, s1_3r: dual_rail_logic_VECTOR(44 downto 8);
    signal c1_3, c1_3r: dual_rail_logic_VECTOR(45 downto 10);
    signal s1_4, s1_4r: dual_rail_logic_VECTOR(47 downto 13);
    signal c1_4, c1_4r: dual_rail_logic_VECTOR(48 downto 15);
    signal s1_5, s1_5r: dual_rail_logic_VECTOR(53 downto 17);
    signal c1_5, c1_5r: dual_rail_logic_VECTOR(54 downto 19);
    signal s1_6, s1_6r: dual_rail_logic_VECTOR(56 downto 22);
    signal c1_6, c1_6r: dual_rail_logic_VECTOR(57 downto 24);
    signal s1_7, s1_7r: dual_rail_logic_VECTOR(62 downto 26);
    signal c1_7, c1_7r: dual_rail_logic_VECTOR(61 downto 28);
    signal s1_8, s1_8r: dual_rail_logic_VECTOR(72 downto 31);
    signal c1_8, c1_8r: dual_rail_logic_VECTOR(59 downto 31);

 signal ki4b: std_logic_VECTOR(272 downto 0);

 signal ko_10b1a: std_logic_VECTOR(11 downto 0);
 signal ko_10b1: std_logic_VECTOR(57 downto 0);
 signal ko_10b2,ko_10b3,ko_10b4,ko_10b5,ko_10b6,ko_10b7,ko_10b8,ko_10b9,ko_10b10,ko_10b11,ko_10b12:
         std_logic_VECTOR(1 downto 0);
 signal
         ko_10b13,ko_10b14,ko_10b15,ko_10b16,ko_10b17,ko_10b18,ko_10b19,ko_10b20,ko_10b21,ko_10b22,
         ko_10b23,ko_10b24: std_logic_VECTOR(1 downto 0);
 signal ko_10b25: std_logic_VECTOR(59 downto 0);
 signal ko_10b26: std_logic_VECTOR(1 downto 0);
 signal ko_10b27: std_logic_VECTOR(53 downto 0);
 signal ko_10b28,ko_10b29,ko_10b30: std_logic_VECTOR(1 downto 0);
 signal ko_10b31: std_logic_VECTOR(59 downto 0);
 signal ko_10b32,ko_10b33: std_logic_VECTOR(1 downto 0);
 signal ko_10b34: std_logic_VECTOR(1 downto 0);
 signal ko_10b35: std_logic_VECTOR(53 downto 0);
 signal ko_10b36,ko_10b37,ko_10b38: std_logic_VECTOR(1 downto 0);
 signal ko_10b39: std_logic_VECTOR(59 downto 0);
 signal ko_10b40,ko_10b41: std_logic_VECTOR(1 downto 0);
 signal ko_10b42: std_logic_VECTOR(1 downto 0);
 signal ko_10b43: std_logic_VECTOR(53 downto 0);
 signal ko_10b44,ko_10b45,ko_10b46,ko_10b47: std_logic_VECTOR(1 downto 0);
 signal ko_10b48: std_logic_VECTOR(61 downto 0);
 signal ko_10b49: std_logic_VECTOR(1 downto 0);
 signal pos2: dual_rail_logic_VECTOR(2 downto 0);
 signal sign2o4: dual_rail_logic_VECTOR(1 downto 0);
 signal add_sub2o4, mac_mpy2o4, MS1o4 : dual_rail_logic;

signal pp4: dual_rail_logic_VECTOR(1023 downto 0);
signal e: dual_rail_logic_VECTOR(72 downto 63);
signal nSh: dual_rail_logic;
          component ncl_register_D
             generic(width: positive ;
        initial_value: integer);
             port(D: in dual_rail_logic_vector(width-1 downto 0);
                ki: in std_logic_vector(width-1 downto 0);
                rst: in std_logic;
                Q: out dual_rail_logic_vector(width-1 downto 0);
```

```
                ko: out std_logic_vector(width-1 downto 0));
 end component;

 component inv1
          port(z: in dual_rail_logic;
             nz: out dual_rail_logic);
 end component;

 component and2     -- complete AND function
    port(a, b: in dual_rail_logic;
       z: out dual_rail_logic);
 end component;

 component MSB_0_30x31
          port(x, y, Sh: in dual_rail_logic;
             pp: out dual_rail_logic);
 end component;

 component LR_31x0_30
    port(x, y, Sh, Sl: in dual_rail_logic;
       pp: out dual_rail_logic);
 end component;

 component MSB_31x31
    port(x, y, Sl: in dual_rail_logic;
       pp1023: out dual_rail_logic);
 end component;

component nor2i             -- incomplete NOR function
          port(a, b: in dual_rail_logic;
             z: out dual_rail_logic);
 end component;

  component nor2           -- input complete NOR function
          port(a, b: in dual_rail_logic;
             z: out dual_rail_logic);
 end component;

 component full_add
    port(c_in, x, y: in dual_rail_logic;
       c_out, s: out dual_rail_logic);
 end component;

 component half_add
    port(x, y: in dual_rail_logic;
       c_out, s: out dual_rail_logic);
 end component;

 component calc_MS
    port(x31, y31, Sh, Sl: in dual_rail_logic;
       MS: out dual_rail_logic);
 end component;
```

```
  component comp
        generic(width: in integer := 4);
        port(a: IN std_logic_vector(width-1 downto 0);
        ko: OUT std_logic);
  end component;

begin

  temp1_in <= FF_in(67 downto 0) ; --temp1;
  ko <= ko_1;

  REG1: ncl_register_D
        generic map(68, -4)
        port map(temp1_in, ki_1, reset, temp1_out, ko_1);
    ki_1(0) <= ko_17(1);
    ki_1(1) <= ko_17(2);
    ki_1(2) <= ki1;
    ki_1(3) <= ki1;

        ACK1: for i in 0 to 30 generate
                ki_1(i+4) <= ki1b2(i);
                ki_1(i+36) <= ki1b3(i);
        end generate ACK1;

                ki_1(31+4) <= ki1b2(31);
                ki_1(31+36) <= ko_17(0);

  x1 <= temp1_out(67 downto 36);
  y1 <= temp1_out(35 downto 4);
  sign2 <= temp1_out(3 downto 2);
  add_sub2 <= temp1_out(1);
  mac_mpy2 <= temp1_out(0);

  PARTIAL_PRODUCTS_X: for i in 0 to 30 generate
        PARTIAL_PRODUCTS_Y: for j in 0 to 30 generate
          INCOMP: if (i /= j) generate
                AND_ARRAY: and2 -- and2i
                        port map(x1(i), y1(j), pp1((i*32)+j));
          end generate INCOMP;
          COMP: if (i = j) generate
                COMP_AND: and2
                        port map(x1(i), y1(j), pp1((i*32)+j));
          end generate COMP;
    end generate PARTIAL_PRODUCTS_Y;
  end generate PARTIAL_PRODUCTS_X;

  MSB: for i in 0 to 30 generate
    MSB_0_30: MSB_0_30x31
        port map(x1(i), y1(31), sign2(1), pp1((i*32)+31));
  end generate MSB;

  e_31 <= sign2(0);
```

```
   e32: nor2   --- nor2i
              port map(sign2(1), sign2(0), e_32);

   temp17 <= y1 & pp1(991 downto 0) & e_32 & e_31 & sign2 & add_sub2 & mac_mpy2 & x1(31) ;

   temp17_in <= temp17;
-------------------------------------------------------------------------------
   ko_17b1 <= ko_17(998) & ko_17(966) & ko_17(934) & ko_17(902) & ko_17(870) & ko_17(838) & ko_17(806) &
            ko_17(774) & ko_17(742) & ko_17(710) & ko_17(678) & ko_17(646) & ko_17(614) & ko_17(582) &
            ko_17(550) & ko_17(518) & ko_17(486) & ko_17(454) & ko_17(422) & ko_17(390) & ko_17(358) &
            ko_17(326) & ko_17(294) & ko_17(262) & ko_17(230) & ko_17(198) & ko_17(166) & ko_17(134) &
            ko_17(102) & ko_17(70) & ko_17(38) & ko_17(6 downto 5) & ko_17(4 downto 3);

   ko_b2: for i in 0 to 31 generate
      ko_17b2( (31+ i*32) downto (0 + i*32) ) <= ko_17(999+i) & ko_17(967+i) & ko_17(935+i) & ko_17(903+i) &
            ko_17(871+i) & ko_17(839+i) & ko_17(807+i) & ko_17(775+i) & ko_17(743+i) & ko_17(711+i) &
            ko_17(679+i) & ko_17(647+i) & ko_17(615+i) & ko_17(583+i) & ko_17(551+i) & ko_17(519+i) &
            ko_17(487+i) & ko_17(455+i) & ko_17(423+i) & ko_17(391+i) & ko_17(359+i) & ko_17(327+i) &
            ko_17(295+i) & ko_17(263+i) & ko_17(231+i) & ko_17(199+i) & ko_17(167+i) & ko_17(135+i) &
            ko_17(103+i) & ko_17(71+i) & ko_17(39+i) & ko_17(7+i);
    end generate ko_b2;

  ko_b3: for i in 0 to 30 generate
      ko_17b3( (31+ i*32) downto (0 + i*32) ) <= ko_17((38 + i*32) downto (7 + i*32));
    end generate ko_b3;

   COMP17b1: comp
      generic map(35)
      port map(ko_17b1, ki1);

   COMP17_b2: for i in 0 to 31 generate
     COMP17b2: comp
      generic map(32)
      port map(ko_17b2( (31+ i*32) downto (0 + i*32) ), ki1b2(i) );
    end generate COMP17_b2;

   COMP17_b3: for i in 0 to 30 generate
     COMP17b3: comp
      generic map(32)
      port map(ko_17b3( (31+ i*32) downto (0 + i*32) ), ki1b3(i) );
    end generate COMP17_b3;

    temp17_out <= temp17_in;
    ko_17 <= ki_17;

    ki_17(0) <= ki17;
    ki_17(2 downto 1) <= ko_4(2 downto 1);
    ki_17(3) <= ki17b2;
    ki_17(4) <= ki17b2;

    ki_17(1029 downto 999) <= ko_4(88 downto 58);
    ki_17(1030) <= ki17b3;
```

```
ki_17(7) <= ko_4(5);

ki_17(8) <= ki17b4;
ki_17(39) <= ki17b4;

        ACK17_1: for i in 0 to 28 generate
                ki_17(9 +i) <= ki17b5(i);
                ki_17(40+i) <= ki17b5(i);
                ki_17(71+i) <= ki17b5(i);
        end generate ACK17_1;

ki_17(69)  <= ki17b6;
ki_17(100) <= ki17b6;
ki_17(5)   <= ki17b6;
ki_17(101) <= ki17b7;
ki_17(969) <= ki17b7;
ki_17(6)   <= ki17b7;

ki_17(970) <= ko_4(729);

        ACK17_2: for i in 0 to 8 generate
                ki_17( (104+i*96) downto (103+i*96) ) <= ko_4((666-i*33) downto (665-i*33));
                ki_17(135+i*96) <= ko_4(338-i*31);
        end generate ACK17_2;

        ACK17_3: for i in 0 to 28 generate
                ki_17(105+i) <= ki17b8_2(i);
                ki_17(136+i) <= ki17b8_2(i);
                ki_17(167+i) <= ki17b8_2(i);
                ki_17(201+i) <= ki17b8_3(i);
                ki_17(232+i) <= ki17b8_3(i);
                ki_17(263+i) <= ki17b8_3(i);
                ki_17(297+i) <= ki17b8_4(i);
                ki_17(328+i) <= ki17b8_4(i);
                ki_17(359+i) <= ki17b8_4(i);
                ki_17(393+i) <= ki17b8_5(i);
                ki_17(424+i) <= ki17b8_5(i);
                ki_17(455+i) <= ki17b8_5(i);
                ki_17(489+i) <= ki17b8_6(i);
                ki_17(520+i) <= ki17b8_6(i);
                ki_17(551+i) <= ki17b8_6(i);
                ki_17(585+i) <= ki17b8_7(i);
                ki_17(616+i) <= ki17b8_7(i);
                ki_17(647+i) <= ki17b8_7(i);
                ki_17(681+i) <= ki17b8_8(i);
                ki_17(712+i) <= ki17b8_8(i);
                ki_17(743+i) <= ki17b8_8(i);
                ki_17(777+i) <= ki17b8_9(i);
                ki_17(808+i) <= ki17b8_9(i);
                ki_17(839+i) <= ki17b8_9(i);
                ki_17(873+i) <= ki17b8_10(i);
                ki_17(904+i) <= ki17b8_10(i);
                ki_17(935+i) <= ki17b8_10(i);
```

```
        end generate ACK17_3;

        ACK17_4: for i in 0 to 8 generate
                ki_17(165+i*96) <= ki17b9(i);
                ki_17(196+i*96) <= ki17b9(i);
                ki_17(971+i*3) <= ki17b9(i);

                ki_17(197+i*96) <= ko_4(697-i*33);
        end generate ACK17_4;

        ACK17_5: for i in 0 to 28 generate
                ki_17(38+i*32) <= ko_4(7+i);
        end generate ACK17_5;

        ki_17(968 downto 966) <= ko_4(38 downto 36);

        ACK17_6:for i in 0 to 7 generate
                ki_17((973+i*3) downto (972+i*3)) <= ko_4((40+i*2) downto (39+i*2));
        end generate ACK17_6;

        ki_17(998 downto 996) <= ko_4(57 downto 55);

 --end process;
-------------------------------------------------------------------------------

 x1_31b <= temp17_out(0);
 mac_mpy2b <= temp17_out(1);
 add_sub2b <= temp17_out(2);
 sign2b <= temp17_out(4 downto 3);
 e_31b <= temp17_out(5);
 e_32b <= temp17_out(6);
 pos2o4(0) <= temp17_out(7);     --pos2o4(0) <= pp(0);
 pp(991 downto 0) <=  temp17_out(998 downto 7);
 y1b <= temp17_out(1030 downto 999);

LR: for j in 0 to 30 generate
        LR_0_30: LR_31x0_30
        port map(x1_31b, y1b(j), sign2b(1), sign2b(0), pp((31*32)+j));
end generate LR;

MSB_31_31: MSB_31x31
  port map(x1_31b, y1b(31), sign2b(0), pp(1023));
--pp(1023) <= x1_31b;

MULT_SIGN: calc_MS
         port map(x1_31b, y1b(31), sign2b(1), sign2b(0), MS1);

HA0_1_1: half_add
        port map(pp(1), pp(32), c0_1(2), pos2o4(1));

WALLACE_TREE_F0_1: for i in 2 to 30 generate
      FULL_ADDERS: full_add
        port map(pp(i), pp(i+31), pp(i+62), c0_1(i+1), s0_1(i));
```

```
end generate WALLACE_TREE_F0_1;

FA0_1_31: full_add
        port map(e_31b, pp(62), pp(93), c0_1(32), s0_1(31));

FA0_1_32: full_add
   port map(e_32b, pp(94), pp(962), c0_1(33), s0_1(32));

s0_1(33) <= pp(963);

s0_2(4 downto 3) <= pp(97 downto 96);
c0_2(4) <= pp(128);

WALLACE_TREE_F0_2: for i in 5 to 33 generate
      FULL_ADDERS: full_add
         port map(pp(i+93), pp(i+124), pp(i+155), c0_2(i+1), s0_2(i));
end generate WALLACE_TREE_F0_2;

FA0_2_34: full_add
   port map(pp(158), pp(189), pp(964), c0_2(35), s0_2(34));

s0_2(35) <= pp(190);

s0_3(7 downto 6) <= pp(193 downto 192);
c0_3(7) <= pp(224);

WALLACE_TREE_F0_3: for i in 8 to 36 generate
      FULL_ADDERS: full_add
         port map(pp(i+186), pp(i+217), pp(i+248), c0_3(i+1), s0_3(i));
end generate WALLACE_TREE_F0_3;

FA0_3_37: full_add
   port map(pp(254), pp(285), pp(967), c0_3(38), s0_3(37));

s0_3(38) <= pp(286);

s0_4(10 downto 9) <= pp(289 downto 288);
c0_4(10) <= pp(320);

WALLACE_TREE_F0_4: for i in 11 to 39 generate
      FULL_ADDERS: full_add
         port map(pp(i+279), pp(i+310), pp(i+341), c0_4(i+1), s0_4(i));
end generate WALLACE_TREE_F0_4;

FA0_4_40: full_add
   port map(pp(350), pp(381), pp(970), c0_4(41), s0_4(40));

s0_4(41) <= pp(382);

s0_5(13 downto 12) <= pp(385 downto 384);
c0_5(13) <= pp(416);

WALLACE_TREE_F0_5: for i in 14 to 42 generate
```

```
      FULL_ADDERS: full_add
         port map(pp(i+372), pp(i+403), pp(i+434), c0_5(i+1), s0_5(i));
end generate WALLACE_TREE_F0_5;

FA0_5_43: full_add
   port map(pp(446), pp(477), pp(973), c0_5(44), s0_5(43));

s0_5(44) <= pp(478);
s0_6(16 downto 15) <= pp(481 downto 480);
c0_6(16) <= pp(512);

WALLACE_TREE_F0_6: for i in 17 to 45 generate
      FULL_ADDERS: full_add
         port map(pp(i+465), pp(i+496), pp(i+527), c0_6(i+1), s0_6(i));
end generate WALLACE_TREE_F0_6;

FA0_6_46: full_add
   port map(pp(542), pp(573), pp(976), c0_6(47), s0_6(46));

s0_6(47) <= pp(574);

s0_7(19 downto 18) <= pp(577 downto 576);
c0_7(19) <= pp(608);

WALLACE_TREE_F0_7: for i in 20 to 48 generate
      FULL_ADDERS: full_add
         port map(pp(i+558), pp(i+589), pp(i+620), c0_7(i+1), s0_7(i));
end generate WALLACE_TREE_F0_7;

FA0_7_49: full_add
   port map(pp(638), pp(669), pp(979), c0_7(50), s0_7(49));

s0_7(50) <= pp(670);
s0_8(22 downto 21) <= pp(673 downto 672);
c0_8(22) <= pp(704);

WALLACE_TREE_F0_8: for i in 23 to 51 generate
      FULL_ADDERS: full_add
         port map(pp(i+651), pp(i+682), pp(i+713), c0_8(i+1), s0_8(i));
end generate WALLACE_TREE_F0_8;

FA0_8_52: full_add
   port map(pp(734), pp(765), pp(982), c0_8(53), s0_8(52));

s0_8(53) <= pp(766);
s0_9(25 downto 24) <= pp(769 downto 768);
c0_9(25) <= pp(800);

WALLACE_TREE_F0_9: for i in 26 to 54 generate
      FULL_ADDERS: full_add
         port map(pp(i+744), pp(i+775), pp(i+806), c0_9(i+1), s0_9(i));
end generate WALLACE_TREE_F0_9;
```

```
  FA0_9_55: full_add
     port map(pp(830), pp(861), pp(985), c0_9(56), s0_9(55));

  s0_9(56) <= pp(862);
  s0_10(28 downto 27) <= pp(865 downto 864);
  c0_10(28) <= pp(896);

  WALLACE_TREE_F0_10: for i in 29 to 57 generate
        FULL_ADDERS: full_add
           port map(pp(i+837), pp(i+868), pp(i+899), c0_10(i+1), s0_10(i));
  end generate WALLACE_TREE_F0_10;

  FA0_10_58: full_add
     port map(pp(926), pp(957), pp(988), c0_10(59), s0_10(58));

  s0_10(59) <= pp(958);

  temp4 <= s0_1 & s0_2 & s0_3 & s0_4 & s0_5 & s0_6 & s0_7 & s0_8 & s0_9 & s0_10 & c0_1 & c0_2(35 downto 6)
          & c0_2(4) &
                    c0_3(38 downto 9) & c0_3(7) & c0_4(41 downto 12) & c0_4(10) & c0_5(44 downto 15) &
          c0_5(13) &
                    c0_6(47 downto 18) & c0_6(16) & c0_7(50 downto 21) & c0_7(19) & c0_8(53 downto 24) &
          c0_8(22) &
                    c0_9(56 downto 27) & c0_9(25) & c0_10(59 downto 30) & c0_10(28) & pp(1023 downto 989) &
          pp(987 downto 986) &
                    pp(984 downto 983) & pp(981 downto 980) & pp(978 downto 977) & pp(975 downto 974) &
          pp(972 downto 971) &
                    pp(969 downto 968) & pp(966 downto 965) & pp(961 downto 959) & pp(927) & pp(895) &
          pp(863) & pp(831) &
                    pp(799) & pp(767) & pp(735) & pp(703) & pp(671) & pp(639) & pp(607) & pp(575) & pp(543) &
          pp(511) &
                    pp(479) & pp(447) & pp(415) & pp(383) & pp(351) & pp(319) & pp(287) & pp(255) & pp(223) &
          pp(191) &
                    pp(159) & pp(127) & pp(95) & pp(63) & pp(31) &
          pos2o4 & sign2b & add_sub2b & mac_mpy2b & MS1;

     temp4_in <= temp4;
-------------------------------------------------------------------------------

  ko_4b1 <= ko_4(89 downto 58) & ko_4(0);
  ko_4b2 <= ko_4(89 downto 58) & ko_4(4 downto 3) & ko_4(0);
  ko_4b3 <= ko_4(89) & ko_4(0);

  ko_4b4 <= ko_4(369) & ko_4(6);
  ko_4b_5: for i in 0 to 28 generate
      ko_4b5( (1+ i*2) downto (0 + i*2) ) <= ko_4(370+i) & ko_4(698+i);
   end generate ko_4b_5;
  ko_4b6 <= ko_4(727) & ko_4(399);
  ko_4b7 <= ko_4(728) & ko_4(400);

  ko_4b_8_2: for i in 0 to 28 generate
     ko_4b8_2( (1+ i*2) downto (0 + i*2) ) <= ko_4(339+i) & ko_4(667+i);
  end generate ko_4b_8_2;
```

```
ko_4b_8_3: for i in 0 to 28 generate
    ko_4b8_3( (1+ i*2) downto (0 + i*2) ) <= ko_4(308+i) & ko_4(634+i);
end generate ko_4b_8_3;
ko_4b_8_4: for i in 0 to 28 generate
    ko_4b8_4( (1+ i*2) downto (0 + i*2) ) <= ko_4(277+i) & ko_4(601+i);
end generate ko_4b_8_4;
ko_4b_8_5: for i in 0 to 28 generate
    ko_4b8_5( (1+ i*2) downto (0 + i*2) ) <= ko_4(246+i) & ko_4(568+i);
end generate ko_4b_8_5;
ko_4b_8_6: for i in 0 to 28 generate
    ko_4b8_6( (1+ i*2) downto (0 + i*2) ) <= ko_4(215+i) & ko_4(535+i);
end generate ko_4b_8_6;
ko_4b_8_7: for i in 0 to 28 generate
    ko_4b8_7( (1+ i*2) downto (0 + i*2) ) <= ko_4(184+i) & ko_4(502+i);
end generate ko_4b_8_7;
ko_4b_8_8: for i in 0 to 28 generate
    ko_4b8_8( (1+ i*2) downto (0 + i*2) ) <= ko_4(153+i) & ko_4(469+i);
end generate ko_4b_8_8;
ko_4b_8_9: for i in 0 to 28 generate
    ko_4b8_9( (1+ i*2) downto (0 + i*2) ) <= ko_4(122+i) & ko_4(436+i);
end generate ko_4b_8_9;
ko_4b_8_10: for i in 0 to 28 generate
    ko_4b8_10( (1+ i*2) downto (0 + i*2) ) <= ko_4(91+i) & ko_4(403+i);
end generate ko_4b_8_10;

ko_4b_9: for i in 0 to 8 generate
     ko_4b9( (1+ i*2) downto (0 + i*2) ) <= ko_4(368-i*31) & ko_4(696-i*33);
  end generate ko_4b_9;

COMP4b1: comp
   generic map(33) --33 --(15 downto 0)
   port map(ko_4b1, ki17);

COMP4b2: comp
   generic map(35)--35 -- (15 downto 0)
   port map(ko_4b2, ki17b2);

COMP4b3: comp
   generic map(2)
   port map(ko_4b3, ki17b3);

COMP4b4: comp
   generic map(2)
   port map(ko_4b4, ki17b4);

COMP4_b5: for i in 0 to 28 generate
  COMP4b5: comp
   generic map(2)
   port map(ko_4b5( (1+ i*2) downto (0 + i*2) ), ki17b5(i) );
  end generate COMP4_b5;

COMP4b6: comp
   generic map(2)
```

```
   port map(ko_4b6, ki17b6);

COMP4b7: comp
   generic map(2)
   port map(ko_4b7, ki17b7);

COMP4_b8_2: for i in 0 to 28 generate
  COMP4b8_2: comp
   generic map(2)
   port map(ko_4b8_2( (1+ i*2) downto (0 + i*2) ), ki17b8_2(i) );
 end generate COMP4_b8_2;

COMP4_b8_3: for i in 0 to 28 generate
  COMP4b8_3: comp
   generic map(2)
   port map(ko_4b8_3( (1+ i*2) downto (0 + i*2) ), ki17b8_3(i) );
 end generate COMP4_b8_3;

COMP4_b8_4: for i in 0 to 28 generate
  COMP4b8_4: comp
   generic map(2)
   port map(ko_4b8_4( (1+ i*2) downto (0 + i*2) ), ki17b8_4(i) );
 end generate COMP4_b8_4;

COMP4_b8_5: for i in 0 to 28 generate
  COMP4b8_5: comp
   generic map(2)
   port map(ko_4b8_5( (1+ i*2) downto (0 + i*2) ), ki17b8_5(i) );
 end generate COMP4_b8_5;

COMP4_b8_6: for i in 0 to 28 generate
  COMP4b8_6: comp
   generic map(2)
   port map(ko_4b8_6( (1+ i*2) downto (0 + i*2) ), ki17b8_6(i) );
 end generate COMP4_b8_6;

COMP4_b8_7: for i in 0 to 28 generate
  COMP4b8_7: comp
   generic map(2)
   port map(ko_4b8_7( (1+ i*2) downto (0 + i*2) ), ki17b8_7(i) );
 end generate COMP4_b8_7;

COMP4_b8_8: for i in 0 to 28 generate
  COMP4b8_8: comp
   generic map(2)
   port map(ko_4b8_8( (1+ i*2) downto (0 + i*2) ), ki17b8_8(i) );
 end generate COMP4_b8_8;

COMP4_b8_9: for i in 0 to 28 generate
  COMP4b8_9: comp
   generic map(2)
   port map(ko_4b8_9( (1+ i*2) downto (0 + i*2) ), ki17b8_9(i) );
 end generate COMP4_b8_9;
```

```
   COMP4_b8_10: for i in 0 to 28 generate
     COMP4b8_10: comp
      generic map(2)
      port map(ko_4b8_10( (1+ i*2) downto (0 + i*2) ), ki17b8_10(i) );
    end generate COMP4_b8_10;

   COMP4_b9: for i in 0 to 8 generate
     COMP4b9: comp
      generic map(2)
      port map(ko_4b9( (1+ i*2) downto (0 + i*2) ), ki17b9(i) );
    end generate COMP4_b9;

   REG4: ncl_register_D
      generic map(730, -4)
      port map(temp4_in, ki_4, reset, temp4_out, ko_4);
--------------------------------------------------------------------------------
--0
--------------------------------------------------------------------------------
            ki_4(0) <= ki4b(0);--ko_10(5);
            ki_4(2 downto 1) <= ko_10(7 downto 6);
            ki_4(3) <= ko_10(8);
            ki_4(4) <= ki4b(0);
            ki_4(6 downto 5) <= ko_10(11 downto 10);

 ACK4_1: for i in 0 to 28 generate
            ki_4(i+7) <= ki4b(i+1);
 end generate ACK4_1;

            ki_4(36) <= ki4b(30);
            ki_4(37) <= ki4b(31);
            ki_4(38) <= ko_10(518);
            ki_4(39) <= ki4b(32);
            ki_4(40) <= ko_10(490);
            ki_4(41) <= ko_10(491);
            ki_4(42) <= ko_10(519);
            ki_4(43) <= ko_10(520);
            ki_4(44) <= ki4b(33);
            ki_4(45) <= ki4b(34);
            ki_4(46) <= ko_10(496);
            ki_4(47) <= ko_10(497);
            ki_4(48) <= ko_10(521);
            ki_4(49) <= ko_10(522);
            ki_4(50) <= ki4b(35);
            ki_4(51) <= ki4b(36);
            ki_4(52) <= ko_10(502);
            ki_4(53) <= ko_10(503);
            ki_4(54) <= ko_10(523);
            ki_4(55) <= ko_10(524);
            ki_4(56) <= ki4b(37);
            ki_4(57) <= ko_10(452);
            ki_4(61 downto 58) <= ko_10(489 downto 486);
            ki_4(62) <= ki4b(38);
```

```
        ki_4(63) <= ki4b(39);
        ki_4(64) <= ki4b(40);
        ki_4(65) <= ki4b(41);
        ki_4(68 downto 66) <= ko_10(494 downto 492);
        ki_4(69) <= ki4b(42);
        ki_4(70) <= ko_10(495);
        ki_4(71) <= ki4b(43);
        ki_4(72) <= ki4b(44);
        ki_4(73) <= ki4b(45);
        ki_4(74) <= ki4b(46);
        ki_4(77 downto 75) <= ko_10(500 downto 498);
        ki_4(78) <= ki4b(47);
        ki_4(79) <= ko_10(501);
        ki_4(80) <= ki4b(48);
        ki_4(81) <= ki4b(49);
        ki_4(82) <= ki4b(50);
        ki_4(83) <= ki4b(51);
        ki_4(86 downto 84) <= ko_10(506 downto 504);
        ki_4(87) <= ki4b(52);
        ki_4(88) <= ko_10(507);
        ki_4(89) <= ko_10(453);
        ki_4(90) <= ko_10(454);
        ki_4(401) <= ko_10(387);
        ki_4(403 downto 402) <= ko_10(420 downto 419);

ACK4_2: for i in 0 to 29 generate
    ki_4(i+404) <= ki4b(i+53);
                    ki_4(i+91) <= ki4b(i+53);
end generate ACK4_2;

        ki_4(152) <= ko_10(351);
        ki_4(153) <= ko_10(352);
        ki_4(155) <= ko_10(354);
        ki_4(434) <= ko_10(385);
        ki_4(436) <= ko_10(418);
        ki_4(154) <= ki4b(83);
        ki_4(435) <= ki4b(83);
        ki_4(121) <= ki4b(83);

ACK4_3: for i in 0 to 26 generate
  ki_4(i+156) <= ki4b(i+84);
                    ki_4(i+437) <= ki4b(i+84);
                    ki_4(i+122) <= ki4b(i+84);
end generate ACK4_3;

        ki_4(464) <= ki4b(111);
        ki_4(149) <= ki4b(111);
        ki_4(465) <= ki4b(112);
        ki_4(150) <= ki4b(112);
        ki_4(466) <= ki4b(113);
        ki_4(151) <= ki4b(113);

        ki_4(502 downto 501) <= ko_10(284 downto 283);
```

```
        ki_4(500) <= ko_10(251);
        ki_4(183) <= ko_10(318);

ACK4_4: for i in 0 to 29 generate
        ki_4(i+503) <= ki4b(i+114);
                ki_4(i+184) <= ki4b(i+114);
                ki_4(i+467) <= ki4b(i+114);
end generate ACK4_4;
        ki_4(497) <= ki4b(144);
        ki_4(499) <= ki4b(145);
        ki_4(498) <= ko_10(316);
        ki_4(245) <= ko_10(215);
        ki_4(246) <= ko_10(216);
        ki_4(533) <= ko_10(249);
        ki_4(535) <= ko_10(282);
        ki_4(248) <= ko_10(218);
        ki_4(247) <= ki4b(146);
        ki_4(534) <= ki4b(146);
        ki_4(214) <= ki4b(146);
ACK4_5: for i in 0 to 26 generate
        ki_4(i+249) <= ki4b(i+147);
                ki_4(i+536) <= ki4b(i+147);
                ki_4(i+215) <= ki4b(i+147);
end generate ACK4_5;

        ki_4(563) <= ki4b(174);
        ki_4(242) <= ki4b(174);
        ki_4(564) <= ki4b(175);
        ki_4(243) <= ki4b(175);
        ki_4(565) <= ki4b(176);
        ki_4(244) <= ki4b(176);
        ki_4(601 downto 600) <= ko_10(148 downto 147);
        ki_4(599) <= ko_10(115);
        ki_4(276) <= ko_10(182);

ACK4_6: for i in 0 to 29 generate
        ki_4(i+602) <= ki4b(i+177);
        ki_4(i+277) <= ki4b(i+177);
        ki_4(i+566) <= ki4b(i+177);
end generate ACK4_6;
        ki_4(596) <= ki4b(207);
        ki_4(598) <= ki4b(208);
        ki_4(597) <= ko_10(180);

        ki_4(339) <= ko_10(80);
        ki_4(338) <= ko_10(79);
        ki_4(341) <= ko_10(82);
        ki_4(632) <= ko_10(113);
        ki_4(634) <= ko_10(146);

        ki_4(340) <= ki4b(209);
        ki_4(633) <= ki4b(209);
        ki_4(307) <= ki4b(209);
```

```
ACK4_7: for i in 0 to 26 generate
        ki_4(i+342) <= ki4b(i+210);
        ki_4(i+635) <= ki4b(i+210);
        ki_4(i+308) <= ki4b(i+210);
end generate ACK4_7;

        ki_4(662) <= ki4b(237);
        ki_4(335) <= ki4b(237);
        ki_4(663) <= ki4b(238);
        ki_4(336) <= ki4b(238);
        ki_4(664) <= ki4b(239);
        ki_4(337) <= ki4b(239);

        ki_4(698) <= ki4b(240);
        ki_4(369) <= ki4b(240);
ACK4_8: for i in 0 to 30 generate
        ki_4(i+699) <= ki4b(i+241);
                ki_4(i+370) <= ki4b(i+241);
                ki_4(i+665) <= ki4b(i+241);
end generate ACK4_8;

        ki_4(696) <= ko_10(44);
        ki_4(697) <= ki4b(272);

 s0_1r <= temp4_out(729 downto 698);
 s0_2r <= temp4_out(697 downto 665);
 s0_3r <= temp4_out(664 downto 632);
 s0_4r <= temp4_out(631 downto 599);
 s0_5r <= temp4_out(598 downto 566);
 s0_6r <= temp4_out(565 downto 533);
 s0_7r <= temp4_out(532 downto 500);
 s0_8r <= temp4_out(499 downto 467);
 s0_9r <= temp4_out(466 downto 434);
 s0_10r <= temp4_out(433 downto 401);
 c0_1r <= temp4_out(400 downto 369);
 c0_2r(35 downto 6) <= temp4_out(368 downto 339);
 c0_2r(4) <= temp4_out(338);
 c0_3r(38 downto 9) <= temp4_out(337 downto 308);
 c0_3r(7) <= temp4_out(307);
 c0_4r(41 downto 12) <= temp4_out(306 downto 277);
 c0_4r(10) <= temp4_out(276);
 c0_5r(44 downto 15) <= temp4_out(275 downto 246);
 c0_5r(13) <= temp4_out(245);
 c0_6r(47 downto 18) <= temp4_out(244 downto 215);
 c0_6r(16) <= temp4_out(214);
 c0_7r(50 downto 21) <= temp4_out(213 downto 184);
 c0_7r(19) <= temp4_out(183);
 c0_8r(53 downto 24) <= temp4_out(182 downto 153);
 c0_8r(22) <= temp4_out(152);
 c0_9r(56 downto 27) <= temp4_out(151 downto 122);
 c0_9r(25) <= temp4_out(121);
 c0_10r(59 downto 30) <= temp4_out(120 downto 91);
 c0_10r(28) <= temp4_out(90);
```

```
    pp4(1023 downto 989) <= temp4_out(89 downto 55);
    pp4(987 downto 986) <= temp4_out(54 downto 53);
    pp4(984 downto 983) <= temp4_out(52 downto 51);
    pp4(981 downto 980) <= temp4_out(50 downto 49);
    pp4(978 downto 977) <= temp4_out(48 downto 47);
    pp4(975 downto 974) <= temp4_out(46 downto 45);
    pp4(972 downto 971) <= temp4_out(44 downto 43);
    pp4(969 downto 968) <= temp4_out(42 downto 41);
    pp4(966 downto 965) <= temp4_out(40 downto 39);
    pp4(961 downto 959) <= temp4_out(38 downto 36);

    RENAME: for i in 31 to 59 generate
            pp4(((i-31)*32)+31) <= temp4_out(i-24);
    end generate RENAME;

    pos2(1 downto 0) <= temp4_out(6 downto 5);
    sign2o4 <= temp4_out(4 downto 3);
    add_sub2o4 <= temp4_out(2);
    mac_mpy2o4 <= temp4_out(1);
    MS1o4 <= temp4_out(0);

-- SECOND LEVEL (add register)

    INVERT: inv1
       port map(sign2o4(1), nSh);

    e_72_63: for i in 63 to 72 generate
       e(i) <= nSh;
    end generate e_72_63;

    HA1_1_2: half_add
       port map(s0_1r(2), c0_1r(2), c1_1(3), pos2(2));

    WALLACE_TREE_F1_1: for i in 3 to 33 generate
          FULL_ADDERS: full_add
             port map(s0_1r(i), c0_1r(i), s0_2r(i), c1_1(i+1), s1_1(i));
    end generate WALLACE_TREE_F1_1;

    s1_1(34) <= s0_2r(34);

    FA1_1_35: full_add
       port map(s0_2r(35), pp4(996), pp4(965), c1_1(36), s1_1(35));

    s1_2(6) <= c0_2r(6);
    s1_2(4) <= c0_2r(4);
    c1_2(6) <= s0_3r(6);
    s1_3(8) <= s0_3r(8);
    s1_2(8) <= c0_2r(8);

    FA1_2_7: full_add
       port map(c0_2r(7), s0_3r(7), c0_3r(7), c1_2(8), s1_2(7));

    WALLACE_TREE_F1_2: for i in 9 to 35 generate
```

```
      FULL_ADDERS: full_add
         port map(c0_2r(i), s0_3r(i), c0_3r(i), c1_2(i+1), s1_2(i));
end generate WALLACE_TREE_F1_2;

WALLACE_TREE_F1_2L: for i in 36 to 38 generate
      FULL_ADDERS: full_add
         port map(pp4(i+961), s0_3r(i), c0_3r(i), c1_2(i+1), s1_2(i));
end generate WALLACE_TREE_F1_2L;

s1_3(11 downto 10) <= s0_4r(11 downto 10);
c1_2(9) <= s0_4r(9);
c1_3(10) <= c0_4r(10);

WALLACE_TREE_F1_3: for i in 12 to 41 generate
      FULL_ADDERS: full_add
         port map(s0_4r(i), c0_4r(i), s0_5r(i), c1_3(i+1), s1_3(i));
end generate WALLACE_TREE_F1_3;

FA1_3_42: full_add
   port map(s0_5r(42), pp4(1003), pp4(972), c1_3(43), s1_3(42));

FA1_3_44: full_add
   port map(s0_5r(44), pp4(1005), pp4(974), c1_3(45), s1_3(44));

s1_3(43) <= s0_5r(43);

--s1_4(15 downto 13) <= c0_5r(15 downto 13);
s1_4(13) <= c0_5r(13);
s1_4(15) <= c0_5r(15);
c1_4(15) <= s0_6r(15);
s1_5(17) <= s0_6r(17);
s1_4(17) <= c0_5r(17);
FA1_4_16: full_add
   port map(c0_5r(16), s0_6r(16), c0_6r(16), c1_4(17), s1_4(16));

WALLACE_TREE_F1_4: for i in 18 to 44 generate
      FULL_ADDERS: full_add
         port map(c0_5r(i), s0_6r(i), c0_6r(i), c1_4(i+1), s1_4(i));
end generate WALLACE_TREE_F1_4;

WALLACE_TREE_F1_4L: for i in 45 to 47 generate
      FULL_ADDERS: full_add
         port map(pp4(i+961), s0_6r(i), c0_6r(i), c1_4(i+1), s1_4(i));
end generate WALLACE_TREE_F1_4L;


s1_5(20 downto 19) <= s0_7r(20 downto 19);
c1_4(18) <= s0_7r(18);
c1_5(19) <= c0_7r(19);

WALLACE_TREE_F1_5: for i in 21 to 50 generate
      FULL_ADDERS: full_add
         port map(s0_7r(i), c0_7r(i), s0_8r(i), c1_5(i+1), s1_5(i));
```

```
end generate WALLACE_TREE_F1_5;

FA1_5_51: full_add
   port map(s0_8r(51), pp4(1012), pp4(981), c1_5(52), s1_5(51));

FA1_5_53: full_add
   port map(s0_8r(53), pp4(1014), pp4(983), c1_5(54), s1_5(53));

s1_5(52) <= s0_8r(52);

--s1_6(24 downto 22) <= c0_8r(24 downto 22);
s1_6(22) <= c0_8r(22);
s1_6(24) <= c0_8r(24);
c1_6(24) <= s0_9r(24);
s1_7(26) <= s0_9r(26);
s1_6(26) <= c0_8r(26);

FA1_6_25: full_add
   port map(c0_8r(25), s0_9r(25), c0_9r(25), c1_6(26), s1_6(25));

WALLACE_TREE_F1_6: for i in 27 to 53 generate
      FULL_ADDERS: full_add
         port map(c0_8r(i), s0_9r(i), c0_9r(i), c1_6(i+1), s1_6(i));
end generate WALLACE_TREE_F1_6;

WALLACE_TREE_F1_6L: for i in 54 to 56 generate
      FULL_ADDERS: full_add
         port map(pp4(i+961), s0_9r(i), c0_9r(i), c1_6(i+1), s1_6(i));
end generate WALLACE_TREE_F1_6L;


s1_7(29 downto 28) <= s0_10r(29 downto 28);
c1_6(27) <= s0_10r(27);
c1_7(28) <= c0_10r(28);

FA1_7_30: full_add
   port map(s0_10r(30), c0_10r(30), pp4(960), c1_7(31), s1_7(30));

WALLACE_TREE_F1_7: for i in 31 to 59 generate
      FULL_ADDERS: full_add
         port map(s0_10r(i), c0_10r(i), pp4(((i-31)*32)+31), c1_7(i+1), s1_7(i));
end generate WALLACE_TREE_F1_7;

FA1_7_60: full_add
   port map(pp4(959), pp4(1021), pp4(990), c1_7(61), s1_7(60));

s1_7(62 downto 61) <= pp4(1023) & pp4(991);


s1_8(34 downto 31) <= pp4(995 downto 992);
c1_8(31) <= pp4(961);
s1_8(41 downto 39) <= pp4(1002 downto 1000);
c1_8(41) <= pp4(971);
```

```
   c1_8(39) <= pp4(969);
   s1_8(38) <= pp4(968);
   s1_8(36) <= pp4(966);
   s1_8(43) <= pp4(1004);
   s1_8(50 downto 48) <= pp4(1011 downto 1009);
   c1_8(50) <= pp4(980);
   c1_8(48) <= pp4(978);
   s1_8(47) <= pp4(977);
   s1_8(45) <= pp4(975);

   s1_8(52) <= pp4(1013);
   s1_8(59 downto 57) <= pp4(1020 downto 1018);
   c1_8(59) <= pp4(989);
   c1_8(57) <= pp4(987);
   s1_8(56) <= pp4(986);
   s1_8(54) <= pp4(984);
   s1_8(72 downto 63) <= e;
   s1_8(61) <= pp4(1022);

-- ADD REGISTER

   temp10 <= c1_8(59) & c1_8(57) & c1_8(50) & c1_8(48) & c1_8(41) & c1_8(39) & c1_8(31) & s1_8(72 downto 63)
         &
               s1_8(61) & s1_8(59 downto 56) & s1_8(54) & s1_8(52) & s1_8(50 downto 47) & s1_8(45) &
         s1_8(43) &
               s1_8(41 downto 38) & s1_8(36) & s1_8(34 downto 31) & c1_7(61 downto 31) & c1_7(28) &
         s1_7(62 downto 28) &
               s1_7(26) & c1_6(57 downto 26) & c1_6(24) & s1_6(56 downto 24) & s1_6(22) & c1_5(54) &
         c1_5(52 downto 22) &
               c1_5(19) & s1_5(53 downto 19) & s1_5(17) & c1_4(48 downto 17) & c1_4(15) & s1_4(47 downto
         15) & s1_4(13) &
               c1_3(45) & c1_3(43 downto 13) & c1_3(10) & s1_3(44 downto 10) & s1_3(8) & c1_2(39 downto
         8) & c1_2(6) &
               s1_2(38 downto 6) & s1_2(4) & c1_1(36) & c1_1(34 downto 3) & s1_1 &
         pos2 & sign2o4 & add_sub2o4 & mac_mpy2o4 & nSh; -- MS1o4;
-- nSh bit extended in the MS bit to calculate overflow (see next register below)

      temp10_in <= temp10;
-------------------------------------------------------------------------------------------------------------
--0
-------------------------------------------------------------------------
   ko_10b1a(11 downto 0) <= ko_10(517 downto 508) & ko_10(9) & ko_10(5);
   COMP10: comp
      generic map(12)
      port map(ko_10b1a, ki4b(0));

   ko_10_b1: for i in 0 to 28 generate
       ko_10b1( (1+ i*2) downto (0 + i*2) ) <= ko_10(456+i) & ko_10(422+i);
    end generate ko_10_b1;
   COMP10_2b: for i in 0 to 28 generate
     COMP10_2: comp
      generic map(2)
      port map(ko_10b1( (1+ i*2) downto (0 + i*2) ), ki4b(i+1) );
```

```
  end generate COMP10_2b;

ko_10b2(1 downto 0) <= ko_10(485) & ko_10(451);
COMP10_3: comp
   generic map(2)
   port map(ko_10b2, ki4b(30));

ko_10b3(1 downto 0) <= ko_10(455) & ko_10(421);
COMP10_4: comp
   generic map(2)
   port map(ko_10b3, ki4b(31));

ko_10b4(1 downto 0) <= ko_10(78) & ko_10(45);
COMP10_5: comp
   generic map(2)
   port map(ko_10b4, ki4b(32));

ko_10b5(1 downto 0) <= ko_10(213) & ko_10(179);
COMP10_6: comp
   generic map(2)
   port map(ko_10b5, ki4b(33));

ko_10b6(1 downto 0) <= ko_10(214) & ko_10(181);
COMP10_7: comp
   generic map(2)
   port map(ko_10b6, ki4b(34));

ko_10b7(1 downto 0) <= ko_10(349) & ko_10(315);
COMP10_8: comp
   generic map(2)
   port map(ko_10b7, ki4b(35));

ko_10b8(1 downto 0) <= ko_10(350) & ko_10(317);
COMP10_9: comp
   generic map(2)
   port map(ko_10b8, ki4b(36));

ko_10b9(1 downto 0) <= ko_10(485) & ko_10(451);
COMP10_10: comp
   generic map(2)
   port map(ko_10b9, ki4b(37));

ko_10b10(1 downto 0) <= ko_10(78) & ko_10(45);
COMP10_11: comp
   generic map(2)
   port map(ko_10b10, ki4b(38));

ko_10b11(1 downto 0) <= ko_10(143) & ko_10(110);
COMP10_12: comp
   generic map(2)
   port map(ko_10b11, ki4b(39));

ko_10b12(1 downto 0) <= ko_10(144) & ko_10(111);
```

```
COMP10_13: comp
   generic map(2)
   port map(ko_10b12, ki4b(40));

ko_10b13(1 downto 0) <= ko_10(145) & ko_10(112);
COMP10_14: comp
   generic map(2)
   port map(ko_10b13, ki4b(41));

ko_10b14(1 downto 0) <= ko_10(213) & ko_10(179);
COMP10_15: comp
   generic map(2)
   port map(ko_10b14, ki4b(42));

ko_10b15(1 downto 0) <= ko_10(214) & ko_10(181);
COMP10_16: comp
   generic map(2)
   port map(ko_10b15, ki4b(43));

ko_10b16(1 downto 0) <= ko_10(279) & ko_10(246);
COMP10_17: comp
   generic map(2)
   port map(ko_10b16, ki4b(44));

ko_10b17(1 downto 0) <= ko_10(280) & ko_10(247);
COMP10_18: comp
   generic map(2)
   port map(ko_10b17, ki4b(45));

ko_10b18(1 downto 0) <= ko_10(281) & ko_10(248);
COMP10_19: comp
   generic map(2)
   port map(ko_10b18, ki4b(46));

ko_10b19(1 downto 0) <= ko_10(349) & ko_10(315);
COMP10_20: comp
   generic map(2)
   port map(ko_10b19, ki4b(47));

ko_10b20(1 downto 0) <= ko_10(350) & ko_10(317);
COMP10_21: comp
   generic map(2)
   port map(ko_10b20, ki4b(48));

ko_10b21(1 downto 0) <= ko_10(415) & ko_10(382);
COMP10_22: comp
   generic map(2)
   port map(ko_10b21, ki4b(49));

ko_10b22(1 downto 0) <= ko_10(416) & ko_10(383);
COMP10_23: comp
   generic map(2)
   port map(ko_10b22, ki4b(50));
```

```
ko_10b23(1 downto 0) <= ko_10(417) & ko_10(384);
COMP10_24: comp
   generic map(2)
   port map(ko_10b23, ki4b(51));

ko_10b24(1 downto 0) <= ko_10(485) & ko_10(451);
COMP10_25: comp
   generic map(2)
   port map(ko_10b24, ki4b(52));

ko_10_b25: for i in 0 to 29 generate
     ko_10b25( (1+ i*2) downto (0 + i*2) ) <= ko_10(455+i) & ko_10(421+i);
  end generate ko_10_b25;
COMP10_26b: for i in 0 to 29 generate
  COMP10_26: comp
   generic map(2)
   port map(ko_10b25( (1+ i*2) downto (0 + i*2) ), ki4b(i+53) );
  end generate COMP10_26b;

ko_10b26(1 downto 0) <= ko_10(386) & ko_10(353);
COMP10_27: comp
   generic map(2)
   port map(ko_10b26, ki4b(83));

ko_10_b27: for i in 0 to 26 generate
     ko_10b27( (1+ i*2) downto (0 + i*2) ) <= ko_10(355+i) & ko_10(388+i);
  end generate ko_10_b27;
COMP10_28b: for i in 0 to 26 generate
  COMP10_28: comp
   generic map(2)
   port map(ko_10b27( (1+ i*2) downto (0 + i*2) ), ki4b(i+84) );
  end generate COMP10_28b;

ko_10b28(1 downto 0) <= ko_10(415) & ko_10(382);
COMP10_29: comp
   generic map(2)
   port map(ko_10b28, ki4b(111));

ko_10b29(1 downto 0) <= ko_10(416) & ko_10(383);
COMP10_30: comp
   generic map(2)
   port map(ko_10b29, ki4b(112));

ko_10b30(1 downto 0) <= ko_10(417) & ko_10(384);
COMP10_31: comp
   generic map(2)
   port map(ko_10b30, ki4b(113));

ko_10_b31: for i in 0 to 29 generate
     ko_10b31( (1+ i*2) downto (0 + i*2) ) <= ko_10(285+i) & ko_10(319+i);
  end generate ko_10_b31;
COMP10_32b: for i in 0 to 29 generate
```

```
   COMP10_32: comp
    generic map(2)
    port map(ko_10b31( (1+ i*2) downto (0 + i*2) ), ki4b(i+114) );
  end generate COMP10_32b;

 ko_10b32(1 downto 0) <= ko_10(349) & ko_10(315);
 COMP10_33: comp
    generic map(2)
    port map(ko_10b32, ki4b(144));

 ko_10b33(1 downto 0) <= ko_10(350) & ko_10(317);
 COMP10_34: comp
    generic map(2)
    port map(ko_10b33, ki4b(145));

 ko_10b34(1 downto 0) <= ko_10(250) & ko_10(217);
 COMP10_35: comp
    generic map(2)
    port map(ko_10b34, ki4b(146));

 ko_10_b35: for i in 0 to 26 generate
      ko_10b35( (1+ i*2) downto (0 + i*2) ) <= ko_10(252+i) & ko_10(219+i);
  end generate ko_10_b35;
 COMP10_36b: for i in 0 to 26 generate
   COMP10_36: comp
    generic map(2)
    port map(ko_10b35( (1+ i*2) downto (0 + i*2) ), ki4b(i+147) );
  end generate COMP10_36b;

ko_10b36(1 downto 0) <= ko_10(279) & ko_10(246);
 COMP10_37: comp
    generic map(2)
    port map(ko_10b36, ki4b(174));

ko_10b37(1 downto 0) <= ko_10(280) & ko_10(247);
 COMP10_38: comp
    generic map(2)
    port map(ko_10b37, ki4b(175));

ko_10b38(1 downto 0) <= ko_10(281) & ko_10(248);
 COMP10_39: comp
    generic map(2)
    port map(ko_10b38, ki4b(176));

 ko_10_b39: for i in 0 to 29 generate
      ko_10b39( (1+ i*2) downto (0 + i*2) ) <= ko_10(183+i) & ko_10(149+i);
  end generate ko_10_b39;
 COMP10_40b: for i in 0 to 29 generate
   COMP10_40: comp
    generic map(2)
    port map(ko_10b39( (1+ i*2) downto (0 + i*2) ), ki4b(i+177) );
  end generate COMP10_40b;
```

```
ko_10b40(1 downto 0) <= ko_10(213) & ko_10(179);
 COMP10_41: comp
    generic map(2)
    port map(ko_10b40, ki4b(207));

ko_10b41(1 downto 0) <= ko_10(214) & ko_10(181);
 COMP10_42: comp
    generic map(2)
    port map(ko_10b41, ki4b(208));

ko_10b42(1 downto 0) <= ko_10(114) & ko_10(81);
 COMP10_43: comp
    generic map(2)
    port map(ko_10b42, ki4b(209));

 ko_10_b43: for i in 0 to 26 generate
     ko_10b43( (1+ i*2) downto (0 + i*2) ) <= ko_10(116+i) & ko_10(83+i);
  end generate ko_10_b43;
 COMP10_44b: for i in 0 to 26 generate
   COMP10_44: comp
    generic map(2)
    port map(ko_10b43( (1+ i*2) downto (0 + i*2) ), ki4b(i+210) );
  end generate COMP10_44b;

ko_10b44(1 downto 0) <= ko_10(143) & ko_10(110);
 COMP10_45: comp
    generic map(2)
    port map(ko_10b44, ki4b(237));

ko_10b45(1 downto 0) <= ko_10(144) & ko_10(111);
 COMP10_46: comp
    generic map(2)
    port map(ko_10b45, ki4b(238));

ko_10b46(1 downto 0) <= ko_10(145) & ko_10(112);
 COMP10_47: comp
    generic map(2)
    port map(ko_10b46, ki4b(239));

ko_10b47(1 downto 0) <= ko_10(46) & ko_10(12);
 COMP10_48: comp
    generic map(2)
    port map(ko_10b47, ki4b(240));

 ko_10_b48: for i in 0 to 30 generate
     ko_10b48( (1+ i*2) downto (0 + i*2) ) <= ko_10(47+i) & ko_10(13+i);
  end generate ko_10_b48;
 COMP10_49b: for i in 0 to 30 generate
   COMP10_49: comp
    generic map(2)
    port map(ko_10b48( (1+ i*2) downto (0 + i*2) ), ki4b(i+241) );
 end generate COMP10_49b;
```

```
  ko_10b49(1 downto 0) <= ko_10(78) & ko_10(45);
   COMP10_50: comp
      generic map(2)
      port map(ko_10b49, ki4b(272));

   REG10: ncl_register_D
      generic map(520, -4)
      port map(temp10_in, ki_10, reset, temp10_out, ko_10);

     ki_10 <= ki;
     PPC_out <= temp10_out;

end TEST;

library ieee;
use ieee.std_logic_1164.all;
use work.ncl_signals.all;

entity feedforward_stage1_2_3_4_ncr is
 port(FF_in: in DUAL_RAIL_LOGIC_VECTOR(67 downto 0);
                reset : in STD_LOGIC;
                     ki: in STD_LOGIC_VECTOR(519 downto 0);
                ko: out STD_LOGIC_VECTOR(67 downto 0);
                PPC_out: out dual_rail_logic_VECTOR(519 downto 0));
end;

architecture BEHAVIOR of feedforward_stage1_2_3_4_ncr is

        signal di1, di2: dual_rail_logic_vector(67 downto 0);

        signal ko1, ko2: std_logic_vector(67 downto 0);
        signal kod: std_logic_vector(67 downto 0);

        signal do1, do2: dual_rail_logic_vector(519 downto 0);
        signal ki1, ki2: std_logic_vector(519 downto 0);

        signal ds1, ds2: std_logic_vector(519 downto 0);
        signal s1, s2: std_logic_vector(67 downto 0);

        signal temp4, temp4_in, temp4_out: dual_rail_logic_VECTOR(519 downto 0);
        signal ki_4, ko_4: std_logic_vector(519 downto 0);

        component feedforward_stage1_2_3_4
 port(FF_in: in DUAL_RAIL_LOGIC_VECTOR(67 downto 0);
                reset : in STD_LOGIC;
                     ki: in STD_LOGIC_VECTOR(519 downto 0);
                ko: out STD_LOGIC_VECTOR(67 downto 0);
                PPC_out: out dual_rail_logic_VECTOR(519 downto 0));
        end component;
  component demux
        port (a: IN dual_rail_logic;
        rst: IN std_logic;
                     ki1, ki2: IN std_logic;
```

```
        s1, s2: IN std_logic;
        z1, z2: OUT dual_rail_logic;
                ko: OUT std_logic);
  end component;

  component demux_rD0
        port (a: IN dual_rail_logic;
        rst: IN std_logic;
                ki1, ki2: IN std_logic;
        s1, s2: IN std_logic;
        z1, z2: OUT dual_rail_logic;
                ko: OUT std_logic);
  end component;

  component mux
        generic(width: in integer := 1);
        port(a1, a2: IN dual_rail_logic_vector(width-1 downto 0);
        z: OUT dual_rail_logic_vector(width-1 downto 0));
  end component;

        component ncl_register_D
          generic(width: positive ;
      initial_value: integer);
          port(D: in dual_rail_logic_vector(width-1 downto 0);
            ki: in std_logic_vector(width-1 downto 0);
            rst: in std_logic;
            Q: out dual_rail_logic_vector(width-1 downto 0);
            ko: out std_logic_vector(width-1 downto 0));
  end component;

  component comp
                generic(width: in integer := 4);
                port(a: IN std_logic_vector(width-1 downto 0);
        ko: OUT std_logic);
  end component;


  component selct
        port (ki, rst: IN std_logic;
        s1, s2: OUT std_logic);
  end component;

  component selct_rD0
    port (ki, rst: IN std_logic;
        s1, s2: OUT std_logic);
  end component;

begin
        DEMUX_INPUT_FF: for i in 0 to 67 generate
                comp12: demux
                        port map(FF_in(i), reset, ko1(i), ko2(i), s1(i), s2(i), di1(i), di2(i), kod(i));
        end generate DEMUX_INPUT_FF;
```

```
        ko <= kod;

        SELECT_INPUT_FF: for i in 0 to 67 generate
          SELECT_INPUT_1: selct
                    port map(kod(i), reset, s1(i), s2(i));
 end generate SELECT_INPUT_FF;

        COMB1: feedforward_stage1_2_3_4
                  port map(di1, reset, ki1, ko1, do1);

    COMB2: feedforward_stage1_2_3_4
      port map(di2, reset, ki2, ko2, do2);

        MUX_OUTPUT: mux
                generic map(520)
                port map(do1, do2, temp4);

        SELECT_OUTPUT: for i in 0 to 519 generate
          SELECT_OUTPUT: selct
                    port map(ko_4(i), reset, ki1(i), ki2(i));
 end generate SELECT_OUTPUT;

   temp4_in <= temp4;

   REG4: ncl_register_D
     generic map(520, -4)
     port map(temp4_in, ki, reset, PPC_out, ko_4);

end BEHAVIOR;

library ieee;
use work.dual_rail.all;
use work.ncl_signals.all;
use ieee.std_logic_1164.all;

entity feedforward_SHIFT is
        port(FF_in: in dual_rail_logic_VECTOR(137 downto 0);
                reset : in std_logic;
                ki: in std_logic_vector(143 downto 0);
                ko: out std_logic_vector(137 downto 0);
                PPC_out: out dual_rail_logic_VECTOR(143 downto 0));
end;

architecture SYN of feedforward_SHIFT is

        signal temp14, temp14_in, temp14_out, temp14_outb: dual_rail_logic_VECTOR(143 downto 0);
        signal ki_14, ki_14b, ko_14: std_logic_vector(143 downto 0);

 signal temp8, temp8_in, temp8_out: dual_rail_logic_VECTOR(137 downto 0);
        signal ki_8, ko_8: std_logic_vector(137 downto 0);
        signal ki8: std_logic;

 signal s7_1: dual_rail_logic_VECTOR(72 downto 0);
```

```
signal s7_1r: dual_rail_logic_VECTOR(72 downto 0);
signal c7_1, c7_1r: dual_rail_logic_VECTOR(62 downto 9);
signal c8_1, s8_1, c8_1r, s8_1r: dual_rail_logic_VECTOR(71 downto 0);
signal s8_2, s8_2r: dual_rail_logic_VECTOR(10 downto 8);


signal sign2o8: dual_rail_logic_VECTOR(1 downto 0);
signal add_sub2o8, mac_mpy2o8, MSo8: dual_rail_logic;
signal ShSlAS, ShAS, SlAS: dual_rail_logic;
-----------------------------------------------------------------------------------------------
 signal ki8a: std_logic;
signal sign2o8a: dual_rail_logic_VECTOR(1 downto 0);
signal add_sub2o8a: dual_rail_logic;
signal ko_8aa: std_logic_vector(63 downto 0);
signal ki8b: std_logic;
signal sign2o8b: dual_rail_logic_VECTOR(1 downto 0);
signal add_sub2o8b: dual_rail_logic;
signal ko_8bb: std_logic_vector(63 downto 0);
signal ko_8cc: std_logic_vector(19 downto 0);
-----------------------------------------------------------------------------------------------
  component and2     -- complete AND function
    port(a, b: in dual_rail_logic;
      z: out dual_rail_logic);
  end component;

  component sign_ss
    port (AS, Sh, Sl: in dual_rail_logic;
            ShSlAS: out dual_rail_logic);
  end component;

  component shift_comp
    port(Pu, Pp, Pl, AS, Sh, Sl: in dual_rail_logic;
      P: out dual_rail_logic);
  end component;

  component shift_comp_low
    port(Pu, AS, Sh, Sl: in dual_rail_logic;
      P: out dual_rail_logic);
  end component;

  component shift_comp_m_low
    port(Pu, Pp, AS, Sh, Sl: in dual_rail_logic;
      P: out dual_rail_logic);
  end component;

  component shift_comp_high
    port(Pl, AS, Sh, Sl: in dual_rail_logic;
      P: out dual_rail_logic);
  end component;

  component shift_comp_m_high
    port(Pp, Pl, AS, Sh, Sl: in dual_rail_logic;
      P: out dual_rail_logic);
```

```
   end component;

            component ncl_register_D
               generic(width: positive ;
          initial_value: integer);
               port(D: in dual_rail_logic_vector(width-1 downto 0);
                  ki: in std_logic_vector(width-1 downto 0);
                  rst: in std_logic;
                  Q: out dual_rail_logic_vector(width-1 downto 0);
                  ko: out std_logic_vector(width-1 downto 0));
   end component;

   component comp
                        generic(width: in integer := 4);
                        port(a: IN std_logic_vector(width-1 downto 0);
            ko: OUT std_logic);
   end component;

begin

   ko <= ko_8;

   REG8: ncl_register_D
      generic map(138, -4)
      port map(FF_in, ki_8, reset, temp8_out, ko_8);

   sign2o8b(1 downto 0) <= temp8_out(137 downto 136);
   add_sub2o8b <= temp8_out(135);

   sign2o8a(1 downto 0) <= temp8_out(134 downto 133);
   add_sub2o8a <= temp8_out(132);

   s7_1r(72 downto 0) <= temp8_out(131 downto 59);
   c7_1r(62 downto 9) <= temp8_out(58 downto 5);
   sign2o8(1 downto 0) <= temp8_out(4 downto 3);
   add_sub2o8 <= temp8_out(2);
   mac_mpy2o8 <= temp8_out(1);
   --MSo8 <= temp8_out(0);

-------------------------------------------------------------
            ki_8(0) <= ki8a;--ko_14(0);
            ki_8(1) <= ko_14(1);
            ki_8(2) <= ki8;
            ki_8(3) <= ki8;
            ki_8(4) <= ki8;

  ACK8: for i in 59 to 120 generate
         ki_8(i) <= ki8;
  end generate ACK8;
-------------------------------------------------------------
            ki_8(132) <= ki8a;
            ki_8(133) <= ki8a;
            ki_8(134) <= ki8a;
```

```
            ACK8a2: for i in 121 to 131 generate
          ki_8(i) <= ki8a;
  end generate ACK8a2;
            ACK8a: for i in 5 to 49 generate
          ki_8(i) <= ki8a;
  end generate ACK8a;
-------------------------------------------------------------
-------------------------------------------------------------
            ki_8(135) <= ki8b;
            ki_8(136) <= ki8b;
            ki_8(137) <= ki8b;

            ACK8b: for i in 50 to 58 generate
          ki_8(i) <= ki8b;
  end generate ACK8b;
  -------------------------------------------------------------
-------------------------------------------------------------
-- NINTH LEVEL (add register)
-- SHIFT & COMPLEMENT

   SHIFT_S0: shift_comp_m_low
            port map(s7_1r(1), s7_1r(0), add_sub2o8, sign2o8(1), sign2o8(0), s8_1(0));

   SHIFT_S: for i in 1 to 60 generate
      SHIFTERS: shift_comp
            port map(s7_1r(i+1), s7_1r(i), s7_1r(i-1), add_sub2o8, sign2o8(1), sign2o8(0), s8_1(i));
   end generate SHIFT_S;

-------------------------------------------------------------------------
   ko_8aa(61 downto 0) <= ko_14(132 downto 72) & ko_14(2);
   COMP8a: comp
      generic map(62)--62
      port map(ko_8aa(61 downto 0), ki8); --(132 downto 69)
-------------------------------------------------------------------------
   SHIFT_S_a: for i in 61 to 71 generate
      SHIFTERS: shift_comp
            port map(s7_1r(i+1), s7_1r(i), s7_1r(i-1), add_sub2o8a, sign2o8a(1), sign2o8a(0), s8_1(i));
   end generate SHIFT_S_a;
-----------------------------------------------------------------------------------
--overflow computation for s8_1(72)
-----------------------------------------------------------------------------------
      SHIFTERS_2: shift_comp
            port map(temp8_out(0), s7_1r(72), s7_1r(71), add_sub2o8a, sign2o8a(1), sign2o8a(0), MSo8);
-----------------------------------------------------------------------------------
   AND_SIGN_L: and2
            port map(add_sub2o8a, sign2o8a(0), SlAS);

   AND_SIGN_H: and2
      port map(add_sub2o8a, sign2o8a(1), ShAS);

   AND_SIGN_SS: sign_ss
      port map(add_sub2o8a, sign2o8a(1), sign2o8a(0), ShSlAS);
```

```
   c8_1(0) <= SlAS;
   c8_1(1) <= ShSlAS;
   s8_2(10) <= ShSlAS;
   s8_2(9) <= SlAS;
   s8_2(8) <= ShAS;

   SHIFT_C8: shift_comp_low
      port map(c7_1r(9), add_sub2o8a, sign2o8a(1), sign2o8a(0), c8_1(8));

   SHIFT_C9: shift_comp_m_low
      port map(c7_1r(10), c7_1r(9), add_sub2o8a, sign2o8a(1), sign2o8a(0), c8_1(9));

   SHIFT_C: for i in 10 to 52 generate
      SHIFTERS: shift_comp
            port map(c7_1r(i+1), c7_1r(i), c7_1r(i-1), add_sub2o8a, sign2o8a(1), sign2o8a(0), c8_1(i));
   end generate SHIFT_C;

-------------------------------------------------------------------------------------------------------------------
   ko_8bb(61 downto 0) <= ko_14(143 downto 133) & ko_14(52 downto 8) & ko_14(7 downto 6) & ko_14(5
         downto 3) & ko_14(0);
   COMP8b: comp
      generic map(62)--61
      port map(ko_8bb(61 downto 0), ki8a); --(68 downto 5)
-------------------------------------------------------------------------------------------------------------------

   SHIFT_Ca: for i in 53 to 61 generate
      SHIFTERS: shift_comp
            port map(c7_1r(i+1), c7_1r(i), c7_1r(i-1), add_sub2o8b, sign2o8b(1), sign2o8b(0), c8_1(i));
   end generate SHIFT_Ca;

   SHIFT_C62: shift_comp_m_high
      port map(c7_1r(62), c7_1r(61), add_sub2o8b, sign2o8b(1), sign2o8b(0), c8_1(62));

   SHIFT_C63: shift_comp_high
      port map(c7_1r(62), add_sub2o8b, sign2o8b(1), sign2o8b(0), c8_1(63));

   c8_1(64) <= add_sub2o8b;
   c8_1(65) <= add_sub2o8b;
   c8_1(66) <= add_sub2o8b;
   c8_1(67) <= add_sub2o8b;
   c8_1(68) <= add_sub2o8b;
   c8_1(69) <= add_sub2o8b;
   c8_1(70) <= add_sub2o8b;
   c8_1(71) <= add_sub2o8b;

-- CSA
-- TENTH LEVEL (add register)

   temp14 <= s8_1(71 downto 0) & c8_1(71 downto 8) & c8_1(1 downto 0) & s8_2(10 downto 8) & add_sub2o8 &
         mac_mpy2o8 & MSo8; --(    (143-72)  (71-8)  (7-6)  (5-3)  (2-1-0))
   temp14_in <= temp14;
```

```
-------------------------------------------------------------------------------
   ko_8cc(18 downto 0) <= ko_14(71 downto 53);
   COMP14: comp
      generic map(19)  --
      port map(ko_8cc(18 downto 0), ki8b); --(63 downto 0) --(143 downto 0)
-------------------------------------------------------------------------------

   REG14: ncl_register_D
      generic map(144, -4)
      port map(temp14_in, ki_14, reset, temp14_out, ko_14);

    REG14b: ncl_register_D
      generic map(144, -4)
      port map(temp14_out, ki_14b, reset, temp14_outb, ki_14);

  PPC_out <= temp14_outb;
  ki_14b <= ki;

end syn;
library ieee;
use ieee.std_logic_1164.all;
use work.ncl_signals.all;

entity feedforward_SHIFT_ncr is
 port(FF_in: in DUAL_RAIL_LOGIC_VECTOR(137 downto 0);
                  reset : in STD_LOGIC;
                        ki: in STD_LOGIC_VECTOR(143 downto 0);
                  ko: out STD_LOGIC_VECTOR(137 downto 0);
                  PPC_out: out dual_rail_logic_VECTOR(143 downto 0));
end;

architecture BEHAVIOR of feedforward_SHIFT_ncr is

          signal di1, di2: dual_rail_logic_vector(137 downto 0);
          signal ko1, ko2: std_logic_vector(137 downto 0);
          signal kod: std_logic_vector(137 downto 0);
          signal do1, do2: dual_rail_logic_vector(143 downto 0);
          signal ki1, ki2: std_logic_vector(143 downto 0);
          signal ds1, ds2: std_logic_vector(143 downto 0);
          signal s1, s2: std_logic_vector(137 downto 0);
          signal temp4, temp4_in, temp4_out: dual_rail_logic_VECTOR(143 downto 0);
          signal ki_4, ko_4: std_logic_vector(143 downto 0);

component feedforward_SHIFT
          port(FF_in: in dual_rail_logic_VECTOR(137 downto 0);
                    reset : in std_logic;
                    ki: in std_logic_vector(143 downto 0);
                    ko: out std_logic_vector(137 downto 0);
                    PPC_out: out dual_rail_logic_VECTOR(143 downto 0));
end component;

   component demux
          port (a: IN dual_rail_logic;
```

```
        rst: IN std_logic;
                ki1, ki2: IN std_logic;
        s1, s2: IN std_logic;
        z1, z2: OUT dual_rail_logic;
                ko: OUT std_logic);
  end component;

  component demux_rD0
        port (a: IN dual_rail_logic;
        rst: IN std_logic;
        ki1, ki2: IN std_logic;
        s1, s2: IN std_logic;
        z1, z2: OUT dual_rail_logic;
        ko: OUT std_logic);
  end component;

  component mux
        generic(width: in integer := 1);
        port(a1, a2: IN dual_rail_logic_vector(width-1 downto 0);
        z: OUT dual_rail_logic_vector(width-1 downto 0));
  end component;

        component ncl_register_D
          generic(width: positive ;
       initial_value: integer);
          port(D: in dual_rail_logic_vector(width-1 downto 0);
            ki: in std_logic_vector(width-1 downto 0);
            rst: in std_logic;
            Q: out dual_rail_logic_vector(width-1 downto 0);
            ko: out std_logic_vector(width-1 downto 0));
  end component;

  component comp
                generic(width: in integer := 4);
                port(a: IN std_logic_vector(width-1 downto 0);
        ko: OUT std_logic);
  end component;

  component selct
        port (ki, rst: IN std_logic;
        s1, s2: OUT std_logic);
  end component;

  component selct_rD0
    port (ki, rst: IN std_logic;
        s1, s2: OUT std_logic);
  end component;

begin
        DEMUX_INPUT_FF: for i in 0 to 137 generate
                comp12: demux
                        port map(FF_in(i), reset, ko1(i), ko2(i), s1(i), s2(i), di1(i), di2(i), kod(i));
        end generate DEMUX_INPUT_FF;
```

```
            ko <= kod;
            SELECT_INPUT_FF: for i in 0 to 137 generate
              SELECT_INPUT_1: selct
                        port map(kod(i), reset, s1(i), s2(i));
 end generate SELECT_INPUT_FF;

   COMB1: feedforward_SHIFT
              port map(di1, reset, ki1, ko1, do1);

    COMB2: feedforward_SHIFT
          port map(di2, reset, ki2, ko2, do2);

            MUX_OUTPUT: mux
                      generic map(144)
                      port map(do1, do2, temp4);



            SELECT_OUTPUT: for i in 0 to 143 generate
              SELECT_OUTPUT: selct
                        port map(ko_4(i), reset, ki1(i), ki2(i));
 end generate SELECT_OUTPUT;

   temp4_in <= temp4;

   REG4: ncl_register_D
      generic map(144, -4)
      port map(temp4_in, ki, reset, PPC_out, ko_4);

end BEHAVIOR;
use work.ncl_signals.all;
use work.dual_rail.all;
library ieee;
use ieee.std_logic_1164.all;

entity feedforward_comp is
  port(FF_in: in DUAL_RAIL_LOGIC_VECTOR(67 downto 0);
                  reset : in STD_LOGIC;
                        ki: in STD_LOGIC_VECTOR(156 downto 0);
                  ko: out STD_LOGIC_VECTOR(67 downto 0);
                  PPC_out: out dual_rail_logic_VECTOR(156 downto 0)); -- width = 139
end;

architecture TEST of feedforward_comp is
            signal x1, y1: dual_rail_logic_VECTOR(31 downto 0);
            signal add_sub2o4, mac_mpy2o4, MS1o4, add_sub1, mac_mpy1, OV1: dual_rail_logic;

       signal add_sub1t, mac_mpy1t: dual_rail_logic;
       signal kif : std_logic;
       signal add_sub2, mac_mpy2: dual_rail_logic;
       signal add_sub2i, mac_mpy2i: dual_rail_logic;
```

```
    signal add_sub2o, mac_mpy2o, add_sub2o8, mac_mpy2o8, MSo8, add_sub2o14, mac_mpy2o14, MSo14:
        dual_rail_logic;
    signal add_sub2oo, mac_mpy2oo: dual_rail_logic;
    signal add_sub3, mac_mpy3: dual_rail_logic;
    signal add_sub10, mac_mpy10, MS10: dual_rail_logic;
    signal add_sub10i, mac_mpy10i, MS10i, add_sub15i, mac_mpy15i, MS15i: dual_rail_logic;
    signal sign1t, sign2, sign2i, sign2o, sign2o4, sign2o8, sign15i: dual_rail_logic_VECTOR(1 downto 0);
    signal sign10, sign10i, sign3, sign3o, sign2oo: dual_rail_logic_VECTOR(1 downto 0);
    signal AS3o, add_sub3o, mac_mpy3o, round3o, rnd_type3o, saturate3o, MS3o: dual_rail_logic;

     signal Aout1: dual_rail_logic_VECTOR(71 downto 0);
     signal Ain: dual_rail_logic_VECTOR(141 downto 0);
     signal temp1, temp1_in, temp1_out : dual_rail_logic_VECTOR(67 downto 0);
     signal ki_1, ko_1: std_logic_vector(67 downto 0);

         signal temp2, temp2_in, temp2_out: dual_rail_logic_VECTOR(288 downto 7);
         signal kof, ki_2, ko_2, ko_2B: std_logic_vector(288 downto 7);
         signal temp3, temp3_in, temp3_out: dual_rail_logic_VECTOR(72 downto 0);
         signal ki_3, ko_3, ki_f, ko_3A: std_logic_vector(72 downto 0);
         signal temp4, temp4_in, temp4_out: dual_rail_logic_VECTOR(729 downto 0);
         signal ki_4, ko_4: std_logic_vector(729 downto 0);
         signal temp5, temp5_in, temp5_out: dual_rail_logic_VECTOR(110 downto 0);
         signal ki_5, ko_5: std_logic_vector(110 downto 0);
         signal temp6, temp6_in, temp6_out: dual_rail_logic_VECTOR(172 downto 5);
         signal ki_6, ko_6: std_logic_vector(172 downto 5);
         signal temp7, temp7_in, temp7_out: dual_rail_logic_VECTOR(212 downto 5);
         signal ki_7, ko_7: std_logic_vector(212 downto 5);
         signal temp8, temp8_in, temp8_out: dual_rail_logic_VECTOR(137 downto 0);
         signal ki_8, ko_8: std_logic_vector(137 downto 0);
         signal temp12, temp12_in, temp12_out: dual_rail_logic_VECTOR(146 downto 5);
         signal ki_12, ko_12: std_logic_vector(146 downto 5);
         signal temp13, temp13_in, temp13_out: dual_rail_logic_VECTOR(377 downto 5);
         signal ki_13, ko_13: std_logic_vector(377 downto 5);
         signal temp14, temp14_in, temp14_out: dual_rail_logic_VECTOR(143 downto 0);
         signal ki_14, ko_14: std_logic_vector(143 downto 0);
         signal temp15, temp15_in, temp15_out: dual_rail_logic_VECTOR(274 downto 0);
         signal ki_15, ko_15: std_logic_vector(274 downto 0);
         signal ko_2a: std_logic_vector(142 downto 0);
         signal pp, pp4: dual_rail_logic_VECTOR(1023 downto 0);
         signal ki1, ki2, ki3, ki4, ki5, ki6, ki7, ki8, ki9, ki10, ki11, ki12, ki13, ki14, ki15, ki16: std_logic;
-------------------------------------------------------------------------------------------------------------
         signal pos2: dual_rail_logic_VECTOR(2 downto 0);
         signal pos2o4: dual_rail_logic_VECTOR(1 downto 0);
         signal pos10: dual_rail_logic_VECTOR(3 downto 0);
         signal pos15: dual_rail_logic_VECTOR(4 downto 0);
         signal pos13: dual_rail_logic_VECTOR(5 downto 0);
         signal pos2i: dual_rail_logic_VECTOR(6 downto 0);
         signal pos3: dual_rail_logic_VECTOR(7 downto 0);
         signal pos: dual_rail_logic_VECTOR(71 downto 0);
         signal pos1, R, Ro: dual_rail_logic_VECTOR(71 downto 0);
-----------------------------------------------------------------------
         signal s3_1, s3_1r: dual_rail_logic_VECTOR(45 downto 5);
         signal c3_1, c3_1r: dual_rail_logic_VECTOR(44 downto 5);
```

```
signal s3_2, s3_2r: dual_rail_logic_VECTOR(54 downto 6);
signal c3_2, c3_2r: dual_rail_logic_VECTOR(54 downto 11);
signal s3_3, s3_3r: dual_rail_logic_VECTOR(72 downto 16);
signal c3_3, c3_3r: dual_rail_logic_VECTOR(62 downto 22);
signal s3_4, s3_4r: dual_rail_logic_VECTOR(58 downto 26);
signal s4_1, s4_1r: dual_rail_logic_VECTOR(72 downto 6);
signal c4_1, c4_1r: dual_rail_logic_VECTOR(62 downto 6);
signal s4_2, s4_2r: dual_rail_logic_VECTOR(55 downto 7);
signal c4_2, c4_2r: dual_rail_logic_VECTOR(47 downto 16);
signal s4_3, s4_3r: dual_rail_logic_VECTOR(38 downto 23);
signal s5_1, s5_1r: dual_rail_logic_VECTOR(72 downto 7);
signal c5_1, c5_1r: dual_rail_logic_VECTOR(62 downto 7);
signal s5_2, s5_2r: dual_rail_logic_VECTOR(48 downto 8);
signal c5_2, c5_2r: dual_rail_logic_VECTOR(38 downto 23);
signal s6_1, s6_1r: dual_rail_logic_VECTOR(72 downto 8);
signal c6_1, c6_1r: dual_rail_logic_VECTOR(62 downto 8);
signal s6_2, s6_2r: dual_rail_logic_VECTOR(39 downto 9);

signal s7_1: dual_rail_logic_VECTOR(72 downto 0);
signal s7_1r: dual_rail_logic_VECTOR(72 downto 0);
signal c7_1, c7_1r: dual_rail_logic_VECTOR(62 downto 9);
signal c8_1, s8_1, c8_1r, s8_1r: dual_rail_logic_VECTOR(71 downto 0);
signal s8_2, s8_2r: dual_rail_logic_VECTOR(10 downto 8);
signal s9_1, c9_1, s9_1r, c9_1r: dual_rail_logic_VECTOR(71 downto 0);
signal s10_1r, c10_1r: dual_rail_logic_VECTOR(71 downto 2);
signal Aout_old: dual_rail_logic;
signal ko_11a: std_logic_vector(1 downto 0);
signal kif2: std_logic;
signal kof2: std_logic_vector(75 downto 0);
signal e_31, e_32, nSh, AS, AS_t, AS1, AS3, MS, MS1, MS1i, MSo, MSoo, MS_t, pos_0, MS3, complete, ShSlAS,
      ShAS, SlAS: dual_rail_logic;
signal e: dual_rail_logic_VECTOR(72 downto 63);
signal Ac: dual_rail_logic_VECTOR(71 downto 0);
signal c11_1: dual_rail_logic_vector(4 downto 3);
signal c11_1r2: dual_rail_logic_vector(6 downto 4);
signal c11_1r3: dual_rail_logic_vector(8 downto 6);
signal c11_1r4: dual_rail_logic_vector(10 downto 8);
signal c11_1r5: dual_rail_logic_vector(12 downto 10);
signal c11_1r6: dual_rail_logic_vector(14 downto 12);
signal c11_1r7: dual_rail_logic_vector(16 downto 14);
signal c11_1r8: dual_rail_logic_vector(18 downto 16);
signal c11_1r9: dual_rail_logic_vector(20 downto 18);
signal c11_1r10: dual_rail_logic_vector(22 downto 20);
signal c11_1r11: dual_rail_logic_vector(24 downto 22);
signal c11_1r12: dual_rail_logic_vector(26 downto 24);
signal c11_1r13: dual_rail_logic_vector(28 downto 26);
signal c11_1r14: dual_rail_logic_vector(30 downto 28);
signal c11_1r15: dual_rail_logic_vector(32 downto 30);
signal c11_1r16: dual_rail_logic_vector(34 downto 32);
signal c11_1r17: dual_rail_logic_vector(36 downto 34);
signal c11_1r18: dual_rail_logic_vector(38 downto 36);
signal c11_1r19: dual_rail_logic_vector(40 downto 38);
signal c11_1r20: dual_rail_logic_vector(42 downto 40);
```

```
signal c11_1r21: dual_rail_logic_vector(44 downto 42);
signal c11_1r22: dual_rail_logic_vector(46 downto 44);
signal c11_1r23: dual_rail_logic_vector(48 downto 46);
signal c11_1r24: dual_rail_logic_vector(50 downto 48);
signal c11_1r25: dual_rail_logic_vector(52 downto 50);
signal c11_1r26: dual_rail_logic_vector(54 downto 52);
signal c11_1r27: dual_rail_logic_vector(56 downto 54);
signal c11_1r28: dual_rail_logic_vector(58 downto 56);
signal c11_1r29: dual_rail_logic_vector(60 downto 58);
signal c11_1r30: dual_rail_logic_vector(62 downto 60);
signal c11_1r31: dual_rail_logic_vector(64 downto 62);
signal c11_1r32: dual_rail_logic_vector(66 downto 64);
signal c11_1r33: dual_rail_logic_vector(68 downto 66);
signal c11_1r34: dual_rail_logic_vector(70 downto 68);
signal c11_1r35: dual_rail_logic_vector(71 downto 70);
signal s10_1r2, c10_1r2: dual_rail_logic_vector(71 downto 4);
signal s10_1r3, c10_1r3: dual_rail_logic_vector(71 downto 6);
signal s10_1r4, c10_1r4: dual_rail_logic_vector(71 downto 8);
signal s10_1r5, c10_1r5: dual_rail_logic_vector(71 downto 10);
signal s10_1r6, c10_1r6: dual_rail_logic_vector(71 downto 12);
signal s10_1r7, c10_1r7: dual_rail_logic_vector(71 downto 14);
signal s10_1r8, c10_1r8: dual_rail_logic_vector(71 downto 16);
signal s10_1r9, c10_1r9: dual_rail_logic_vector(71 downto 18);
signal s10_1r10, c10_1r10: dual_rail_logic_vector(71 downto 20);
signal s10_1r11, c10_1r11: dual_rail_logic_vector(71 downto 22);
signal s10_1r12, c10_1r12: dual_rail_logic_vector(71 downto 24);
signal s10_1r13, c10_1r13: dual_rail_logic_vector(71 downto 26);
signal s10_1r14, c10_1r14: dual_rail_logic_vector(71 downto 28);
signal s10_1r15, c10_1r15: dual_rail_logic_vector(71 downto 30);
signal s10_1r16, c10_1r16: dual_rail_logic_vector(71 downto 32);
signal s10_1r17, c10_1r17: dual_rail_logic_vector(71 downto 34);
signal s10_1r18, c10_1r18: dual_rail_logic_vector(71 downto 36);
signal s10_1r19, c10_1r19: dual_rail_logic_vector(71 downto 38);
signal s10_1r20, c10_1r20: dual_rail_logic_vector(71 downto 40);
signal s10_1r21, c10_1r21: dual_rail_logic_vector(71 downto 42);
signal s10_1r22, c10_1r22: dual_rail_logic_vector(71 downto 44);
signal s10_1r23, c10_1r23: dual_rail_logic_vector(71 downto 46);
signal s10_1r24, c10_1r24: dual_rail_logic_vector(71 downto 48);
signal s10_1r25, c10_1r25: dual_rail_logic_vector(71 downto 50);
signal s10_1r26, c10_1r26: dual_rail_logic_vector(71 downto 52);
signal s10_1r27, c10_1r27: dual_rail_logic_vector(71 downto 54);
signal s10_1r28, c10_1r28: dual_rail_logic_vector(71 downto 56);
signal s10_1r29, c10_1r29: dual_rail_logic_vector(71 downto 58);
signal s10_1r30, c10_1r30: dual_rail_logic_vector(71 downto 60);
signal s10_1r31, c10_1r31: dual_rail_logic_vector(71 downto 62);
signal s10_1r32, c10_1r32: dual_rail_logic_vector(71 downto 64);
signal s10_1r33, c10_1r33: dual_rail_logic_vector(71 downto 66);
signal s10_1r34, c10_1r34: dual_rail_logic_vector(71 downto 68);
signal s10_1r35, c10_1r35: dual_rail_logic_vector(71 downto 70);
signal ki7b: std_logic_VECTOR(274 downto 0);
signal ko_7b1,ko_7b2: std_logic_VECTOR(1 downto 0);
signal ko_7b3: std_logic_VECTOR(99 downto 32);
signal ko_7b4,ko_7b5,ko_7b6: std_logic_VECTOR(1 downto 0);
```

```
signal ko_7b7: std_logic_VECTOR(181 downto 114);
signal ko_7b8: std_logic_VECTOR(1 downto 0);
signal ko_7b9: std_logic_VECTOR(251 downto 184);
signal ko_7b10,ko_7b11,ko_7b12,ko_7b13: std_logic_VECTOR(1 downto 0);
signal ko_7b14: std_logic_VECTOR(271 downto 260);
signal ko_7b15: std_logic_VECTOR(1 downto 0);
signal ko_7b16: std_logic_VECTOR(317 downto 294);
signal ko_7b17: std_logic_VECTOR(331 downto 320);
signal ko_7b18,ko_7b19,ko_7b20: std_logic_VECTOR(1 downto 0);
signal ko_7b21: std_logic_VECTOR(349 downto 338);
signal ko_7b22: std_logic_VECTOR(1 downto 0);
signal ko_7b23: std_logic_VECTOR(383 downto 360);
signal ko_7b24: std_logic_VECTOR(397 downto 386);
signal ko_7b25,ko_7b26,ko_7b27: std_logic_VECTOR(1 downto 0);
signal ko_7b28: std_logic_VECTOR(417 downto 406);
signal ko_7b29: std_logic_VECTOR(425 downto 420);
signal ko_7b30: std_logic_VECTOR(1 downto 0);
signal ko_7b31: std_logic_VECTOR(479 downto 456);
signal ko_7b32: std_logic_VECTOR(491 downto 480);
signal ko_7b33: std_logic_VECTOR(1 downto 0);
signal ko_7b34: std_logic_VECTOR(517 downto 512);
signal ko_7b35: std_logic_VECTOR(549 downto 544);
signal ki6b: std_logic_VECTOR(212 downto 5);
signal ko_6b1,ko_6b2: std_logic_VECTOR(1 downto 0);
signal ko_6b3: std_logic_VECTOR(115 downto 52);
signal ko_6b4: std_logic_VECTOR(131 downto 118);
signal ko_6b5,ko_6b6: std_logic_VECTOR(1 downto 0);
signal ko_6b7: std_logic_VECTOR(249 downto 186);
signal ko_6b8: std_logic_VECTOR(265 downto 252);
signal ko_6b9: std_logic_VECTOR(1 downto 0);
signal ko_6b10: std_logic_VECTOR(343 downto 280);
signal ko_6b11: std_logic_VECTOR(357 downto 344);
signal ki12b: std_logic_VECTOR(172 downto 5);
signal ko_12b1,ko_12b2: std_logic_VECTOR(1 downto 0);
signal ko_12b3: std_logic_VECTOR(97 downto 66);
signal ko_12b4: std_logic_VECTOR(117 downto 100);
signal ko_12b5,ko_12b6: std_logic_VECTOR(1 downto 0);
signal ko_12b7: std_logic_VECTOR(229 downto 198);
signal ko_12b8: std_logic_VECTOR(249 downto 232);
signal ko_12b9: std_logic_VECTOR(1 downto 0);
signal ko_12b10: std_logic_VECTOR(309 downto 278);
signal ko_12b11: std_logic_VECTOR(327 downto 310);
signal ki8b: std_logic_VECTOR(146 downto 5);
signal ko_8b1,ko_8b2: std_logic_VECTOR(1 downto 0);
signal ko_8b3: std_logic_VECTOR(99 downto 40);
signal ko_8b4,ko_8b5: std_logic_VECTOR(1 downto 0);
signal ko_8b6: std_logic_VECTOR(229 downto 170);
signal ko_8b7: std_logic_VECTOR(1 downto 0);
signal ko_8b8: std_logic_VECTOR(293 downto 276);
signal ki8bb: std_logic_VECTOR(2 downto 0);
signal ko_8b9 : std_logic_VECTOR(2 downto 0);
signal ko_8b10: std_logic_VECTOR(2 downto 0);
signal ko_8b11: std_logic_VECTOR(2 downto 0);
```

```
        signal ki14b: std_logic_VECTOR(143 downto 0);
        signal ko_2b1: std_logic_VECTOR(11 downto 6);
        signal ko_2b2: std_logic_VECTOR(141 downto 16);
        signal ko_2b3: std_logic_VECTOR(285 downto 160);
        signal ko_2b4: std_logic_VECTOR(17 downto 0);
        signal ki144: std_logic;
        signal temp2a, temp2_outa: dual_rail_logic_VECTOR(161 downto 7);
        signal kofa, ko_2aa : std_logic_vector(161 downto 7);
        signal ki14a1,ki14a2,ki14a3,ki14a4: std_logic;
        signal ko_2ba1: std_logic_VECTOR(3 downto 0);
        signal ko_2ba2: std_logic_VECTOR(3 downto 0);
        signal ko_2ba3: std_logic_VECTOR(3 downto 0);
        signal ko_2ba4: std_logic_VECTOR(3 downto 0);
        signal temp2bb, temp2_outbb: dual_rail_logic_VECTOR(163 downto 7);
        signal kofbb, ko_2bb : std_logic_vector(163 downto 7);
        signal ki14b1: std_logic;

-------------------------------------------------------------------------------------------
-- overflow stuff signals
        signal ko_2b5: std_logic_VECTOR(3 downto 0);
        signal ki14_2: std_logic;

        signal c9_1_72, s9_1_72: dual_rail_logic;
-------------------------------------------------------------------------------------------
  component ncl_register_D
          generic(width: positive ;
      initial_value: integer);
          port(D: in dual_rail_logic_vector(width-1 downto 0);
            ki: in std_logic_vector(width-1 downto 0);
            rst: in std_logic;
            Q: out dual_rail_logic_vector(width-1 downto 0);
            ko: out std_logic_vector(width-1 downto 0));
  end component;

  component and2    -- complete AND function
    port(a, b: in dual_rail_logic;
      z: out dual_rail_logic);
  end component;

  component and2i        -- incomplete AND function
    port(a, b: in dual_rail_logic;
      z: out dual_rail_logic);
  end component;

  component sign_ss
    port (AS, Sh, Sl: in dual_rail_logic;
          ShSlAS: out dual_rail_logic);
  end component;

  component full_add
    port(c_in, x, y: in dual_rail_logic;
      c_out, s: out dual_rail_logic);
  end component;
```

```
  component half_add
     port(x, y: in dual_rail_logic;
        c_out, s: out dual_rail_logic);
  end component;

  component comp
          generic(width: in integer := 4);
          port(a: IN std_logic_vector(width-1 downto 0);
          ko: OUT std_logic);
  end component;

 component feedforward_stage1_2_3_4_ncr
 port(FF_in: in DUAL_RAIL_LOGIC_VECTOR(67 downto 0);
                reset : in STD_LOGIC;
                     ki: in STD_LOGIC_VECTOR(519 downto 0);
                ko: out STD_LOGIC_VECTOR(67 downto 0);
                PPC_out: out dual_rail_logic_VECTOR(519 downto 0));
 end component;

component feedforward_SHIFT_ncr
 port(FF_in: in DUAL_RAIL_LOGIC_VECTOR(137 downto 0);
                reset : in STD_LOGIC;
                     ki: in STD_LOGIC_VECTOR(143 downto 0);
                ko: out STD_LOGIC_VECTOR(137 downto 0);
                PPC_out: out dual_rail_logic_VECTOR(143 downto 0));
end component;

component feedforward_SHIFT
 port(FF_in: in DUAL_RAIL_LOGIC_VECTOR(137 downto 0);
                reset : in STD_LOGIC;
                     ki: in STD_LOGIC_VECTOR(143 downto 0);
                ko: out STD_LOGIC_VECTOR(137 downto 0);
                PPC_out: out dual_rail_logic_VECTOR(143 downto 0));
end component;

begin

     FF_stage1_2_3_4_ncr: feedforward_stage1_2_3_4_ncr
          port map(FF_in, reset, ki_10, ko, temp10_out
--------------------------------------------------------------------------
--                                                  1
--------------------------------------------------------------------------
          ki_10(5) <= ko_13(5);
          ki_10(6) <= ko_13(6);
          ki_10(7) <= ko_13(7);
          ki_10(9 downto 8) <= ko_13(9 downto 8);
          ki_10(12 downto 10) <= ko_13(12 downto 10);

          ki_10(13) <= ki10b(13);
          ki_10(14) <= ki10b(14);
          ki_10(15) <= ko_13(15);
```

```
ACK10_1: for i in 16 to 44 generate
        ki_10(i) <= ki10b(i);
end generate ACK10_1;

ki_10(45) <= ki10b(45);

ki_10(46) <= ki10b(46);
ki_10(47) <= ki10b(47);
ki_10(48) <= ko_13(86);

ACK10_2: for i in 49 to 77 generate
        ki_10(i) <= ki10b(i);
end generate ACK10_2;
ki_10(78) <= ki10b(78);

ki_10(79) <= ki10b(79);
ACK10_3: for i in 80 to 108 generate
        ki_10(i) <= ki10b(i);
end generate ACK10_3;

ki_10(109) <= ki10b(109);
ki_10(110) <= ki10b(110);
ki_10(111) <= ki10b(111);
ki_10(112) <= ki10b(112);

ki_10(113) <= ko_13(52);
ki_10(115 downto 114) <= ko_13(88 downto 87);
ki_10(116) <= ki10b(116);
ki_10(118 downto 117) <= ko_13(91 downto 90);
ACK10_4: for i in 119 to 145 generate
        ki_10(i) <= ki10b(i);
end generate ACK10_4;

ki_10(146) <= ko_13(125);
ki_10(147) <= ki10b(147);
ki_10(148) <= ko_13(162);
ki_10(149) <= ko_13(127);
ACK10_5: for i in 150 to 176 generate
        ki_10(i) <= ki10b(i);
end generate ACK10_5;
ACK10_6: for i in 177 to 180 generate
        ki_10(i) <= ki10b(i);
end generate ACK10_6;
ki_10(181) <= ki10b(181);

ki_10(182) <= ki10b(182);
ACK10_7: for i in 183 to 209 generate
        ki_10(i) <= ki10b(i);
end generate ACK10_7;
ACK10_8: for i in 210 to 213 generate
        ki_10(i) <= ki10b(i);
end generate ACK10_8;
ki_10(214) <= ki10b(214);
```

```
ki_10(215) <= ko_13(128);
ki_10(216) <= ko_13(163);
ki_10(217) <= ko_13(164);
ki_10(218) <= ki10b(218);
ki_10(219) <= ko_13(166);
ACK10_9: for i in 220 to 248 generate
        ki_10(i) <= ki10b(i);
end generate ACK10_9;

ki_10(249) <= ko_13(202);
ki_10(250) <= ki10b(250);
ki_10(251) <= ko_13(240);
ACK10_10: for i in 252 to 280 generate
        ki_10(i) <= ki10b(i);
end generate ACK10_10;
ki_10(281) <= ki10b(281);

ki_10(282) <= ki10b(282);
ACK10_11: for i in 283 to 311 generate
        ki_10(i) <= ki10b(i);
end generate ACK10_11;
ki_10(312) <= ki10b(312);
ACK10_12: for i in 313 to 317 generate
        ki_10(i) <= ki10b(i);
end generate ACK10_12;

ki_10(318) <= ko_13(204);
ki_10(319) <= ko_13(241);
ki_10(320) <= ko_13(242);
ki_10(321) <= ki10b(321);
ki_10(322) <= ko_13(244);
ACK10_13: for i in 323 to 349 generate
        ki_10(i) <= ki10b(i);
end generate ACK10_13;
ki_10(350) <= ki10b(350);

ki_10(351) <= ko_13(280);
ki_10(352) <= ki10b(352);
ki_10(353) <= ko_13(317);
ACK10_14: for i in 354 to 380 generate
        ki_10(i) <= ki10b(i);
end generate ACK10_14;
ki_10(381) <= ko_13(272);
ki_10(382) <= ki10b(382);
ki_10(383) <= ki10b(383);
ki_10(384) <= ki10b(384);

ki_10(385) <= ki10b(385);
ACK10_15: for i in 386 to 412 generate
        ki_10(i) <= ki10b(i);
end generate ACK10_15;
ki_10(413) <= ko_13(340);
```

```
ki_10(414) <= ki10b(414);
ki_10(415) <= ki10b(415);
ki_10(416) <= ki10b(416);
ki_10(417) <= ki10b(417);

ki_10(418) <= ko_13(282);
ki_10(421 downto 419) <= ko_13(320 downto 318);
ACK10_16: for i in 422 to 425 generate
        ki_10(i) <= ki10b(i);
end generate ACK10_16;
ki_10(426) <= ki10b(426);
ki_10(427) <= ki10b(427);
ki_10(428) <= ki10b(428);
ki_10(429) <= ki10b(429);
ki_10(430) <= ki10b(430);
ACK10_17: for i in 431 to 434 generate
        ki_10(i) <= ki10b(i);
end generate ACK10_17;
ki_10(435) <= ki10b(435);
ki_10(436) <= ki10b(436);
ki_10(437) <= ko_13(334);
ki_10(438) <= ki10b(438);
ki_10(439) <= ki10b(439);
ACK10_18: for i in 440 to 444 generate
        ki_10(i) <= ki10b(i);
end generate ACK10_18;
ki_10(445) <= ko_13(310);
ki_10(446) <= ki10b(446);
ki_10(447) <= ki10b(447);
ki_10(448) <= ki10b(448);
ki_10(449) <= ki10b(449);
ki_10(450) <= ki10b(450);
ki_10(451) <= ko_13(279);
ki_10(452) <= ki10b(452);
ki_10(453) <= ko_13(348);

ki_10(454) <= ko_13(359);
ACK10_19: for i in 455 to 458 generate
        ki_10(i) <= ki10b(i);
end generate ACK10_19;
ki_10(460 downto 459) <= ko_13(326 downto 325);
ki_10(461) <= ki10b(461);
ki_10(462) <= ki10b(462);
ki_10(463) <= ki10b(463);
ki_10(464) <= ko_13(329);
ki_10(465) <= ki10b(465);
ki_10(467 downto 466) <= ko_13(332 downto 331);
ki_10(468) <= ki10b(468);
ki_10(469) <= ki10b(469);
ki_10(470) <= ko_13(370);
ki_10(471) <= ki10b(471);
ki_10(472) <= ki10b(472);
ACK10_20: for i in 473 to 477 generate
```

```
                ki_10(i) <= ki10b(i);
        end generate ACK10_20;
        ki_10(480 downto 478) <= ko_13(343 downto 341);
        ki_10(481) <= ki10b(481);
        ki_10(482) <= ki10b(482);
        ki_10(483) <= ki10b(483);
        ki_10(484) <= ko_13(346);
        ki_10(485) <= ki10b(485);

        ACK10_21: for i in 486 to 489 generate
                ki_10(i) <= ki10b(i);
        end generate ACK10_21;
        ki_10(490) <= ko_13(365);
        ki_10(491) <= ko_13(327);
        ki_10(492) <= ki10b(492);
        ki_10(493) <= ko_13(49);
        ki_10(494) <= ki10b(494);
        ki_10(495) <= ko_13(369);
        ki_10(496) <= ko_13(333);
        ki_10(497) <= ki10b(497);
        ki_10(498) <= ki10b(498);
        ki_10(500 downto 499) <= ko_13(338 downto 337);
        ki_10(501) <= ko_13(339);
        ki_10(502) <= ko_13(374);
        ki_10(503) <= ko_13(375);
        ki_10(504) <= ko_13(344);
        ki_10(505) <= ki10b(505);
        ki_10(506) <= ki10b(506);
        ki_10(507) <= ki10b(507);
        ki_10(517 downto 508) <= ko_13(358 downto 349);
        ki_10(518) <= ko_13(360);
        ki_10(519) <= ko_13(366);
        ki_10(520) <= ki10b(520);
        ki_10(521) <= ki10b(521);
        ki_10(522) <= ko_13(373);
        ki_10(523) <= ko_13(376);
        ki_10(524) <= ko_13(345);
-------------------------------------------------------------------------------

   c1_8r(59) <= temp10_out(524);
   c1_8r(57) <= temp10_out(523);
   c1_8r(50) <= temp10_out(522);
   c1_8r(48) <= temp10_out(521);
   c1_8r(41) <= temp10_out(520);
   c1_8r(39) <= temp10_out(519);
   c1_8r(31) <= temp10_out(518);
   s1_8r(72 downto 63) <= temp10_out(517 downto 508);
   s1_8r(61) <= temp10_out(507);
   s1_8r(59 downto 56) <= temp10_out(506 downto 503);
   s1_8r(54) <= temp10_out(502);
   s1_8r(52) <= temp10_out(501);
   s1_8r(50 downto 47) <= temp10_out(500 downto 497);
   s1_8r(45) <= temp10_out(496);
```

```
    s1_8r(43) <= temp10_out(495);
    s1_8r(41 downto 38) <= temp10_out(494 downto 491);
    s1_8r(36) <= temp10_out(490);
    s1_8r(34 downto 31) <= temp10_out(489 downto 486);
    c1_7r(61 downto 31) <= temp10_out(485 downto 455);
    c1_7r(28) <= temp10_out(454);
    s1_7r(62 downto 28) <= temp10_out(453 downto 419);
    s1_7r(26) <= temp10_out(418);
    c1_6r(57 downto 26) <= temp10_out(417 downto 386);
    c1_6r(24) <= temp10_out(385);
    s1_6r(56 downto 24) <= temp10_out(384 downto 352);
    s1_6r(22) <= temp10_out(351);
    c1_5r(54) <= temp10_out(350);
    c1_5r(52 downto 22) <= temp10_out(349 downto 319);
    c1_5r(19) <= temp10_out(318);
    s1_5r(53 downto 19) <= temp10_out(317 downto 283);
    s1_5r(17) <= temp10_out(282);
    c1_4r(48 downto 17) <= temp10_out(281 downto 250);
    c1_4r(15) <= temp10_out(249);
    s1_4r(47 downto 15) <= temp10_out(248 downto 216);
    s1_4r(13) <= temp10_out(215);
    c1_3r(45) <= temp10_out(214);
    c1_3r(43 downto 13) <= temp10_out(213 downto 183);
    c1_3r(10) <= temp10_out(182);
    s1_3r(44 downto 10) <= temp10_out(181 downto 147);
    s1_3r(8) <= temp10_out(146);
    c1_2r(39 downto 8) <= temp10_out(145 downto 114);
    c1_2r(6) <= temp10_out(113);
    s1_2r(38 downto 6) <= temp10_out(112 downto 80);
    s1_2r(4) <= temp10_out(79);
    c1_1r(36) <= temp10_out(78);
    c1_1r(34 downto 3) <= temp10_out(77 downto 46);
    s1_1r <= temp10_out(45 downto 13);
    pos10(2 downto 0) <= temp10_out(12 downto 10);
    sign10 <= temp10_out(9 downto 8);
    add_sub10 <= temp10_out(7);
    mac_mpy10 <= temp10_out(6);
    MS10 <= temp10_out(5);

-- THIRD LEVEL (add register)

    HA_POS3_2: half_add
       port map(s1_1r(3), c1_1r(3), c2_1(4), pos10(3));

    FA2_1_4: full_add
       port map(s1_1r(4), c1_1r(4), s1_2r(4), c2_1(5), s2_1(4));

    s2_1(5) <= s1_1r(5);
    s2_2(5) <= c1_1r(5);

    WALLACE_TREE_F2_1: for i in 6 to 34 generate
          FULL_ADDERS: full_add
             port map(s1_1r(i), c1_1r(i), s1_2r(i), c2_1(i+1), s2_1(i));
```

```
end generate WALLACE_TREE_F2_1;

FA2_1_35: full_add
   port map(s1_1r(35), s1_2r(35), s1_7r(35), c2_1(36), s2_1(35));

FA2_1_36: full_add
   port map(c1_1r(36), s1_2r(36), s1_7r(36), c2_1(37), s2_1(36));

FA2_1_37: full_add
   port map(s1_2r(37), s1_7r(37), c1_7r(37), c2_1(38), s2_1(37));

FA2_1_38: full_add
   port map(s1_2r(38), s1_7r(38), c1_7r(38), c2_1(39), s2_1(38));


FA2_2_10: full_add
   port map(c1_2r(10), s1_3r(10), c1_3r(10), c2_2(11), s2_2(10));

s2_2(9 downto 8) <= c1_2r(9 downto 8);
c2_2(8) <= s1_3r(8);
c2_1(6) <= c1_2r(6);
s2_2(12 downto 11) <= c1_2r(12 downto 11);
c2_2(12) <= s1_3r(12);
s2_3(11) <= s1_3r(11);

WALLACE_TREE_F2_2: for i in 13 to 39 generate
      FULL_ADDERS: full_add
         port map(c1_2r(i), s1_3r(i), c1_3r(i), c2_2(i+1), s2_2(i));
end generate WALLACE_TREE_F2_2;

WALLACE_TREE_F2_2L: for i in 40 to 43 generate
      FULL_ADDERS: full_add
         port map(s1_3r(i), c1_3r(i), s1_7r(i), c2_2(i+1), s2_2(i));
end generate WALLACE_TREE_F2_2L;

FA2_2_44: full_add
   port map(s1_3r(44), s1_7r(44), c1_7r(44), c2_2(45), s2_2(44));

FA2_2_45: full_add
   port map(c1_3r(45), s1_7r(45), c1_7r(45), c2_2(46), s2_2(45));


FA2_3_17: full_add
   port map(s1_4r(17), c1_4r(17), s1_5r(17), c2_3(18), s2_3(17));

s2_3(16 downto 15) <= s1_4r(16 downto 15);
c2_2(13) <= s1_4r(13);
c2_3(15) <= c1_4r(15);
s2_4(18) <= c1_4r(18);
s2_3(18) <= s1_4r(18);

WALLACE_TREE_F2_3: for i in 19 to 47 generate
      FULL_ADDERS: full_add
```

```
        port map(s1_4r(i), c1_4r(i), s1_5r(i), c2_3(i+1), s2_3(i));
end generate WALLACE_TREE_F2_3;

WALLACE_TREE_F2_3L: for i in 49 to 53 generate
      FULL_ADDERS: full_add
        port map(s1_5r(i), s1_7r(i), c1_7r(i), c2_3(i+1), s2_3(i));
end generate WALLACE_TREE_F2_3L;

FA2_3_48: full_add
   port map(c1_4r(48), s1_5r(48), s1_7r(48), c2_3(49), s2_3(48));

FA2_4_24: full_add
   port map(c1_5r(24), s1_6r(24), c1_6r(24), c2_4(25), s2_4(24));

s2_4(23 downto 22) <= c1_5r(23 downto 22);
c2_4(22) <= s1_6r(22);
c2_3(19) <= c1_5r(19);
s2_5(25) <= s1_6r(25);
s2_4(25) <= c1_5r(25);

WALLACE_TREE_F2_4: for i in 26 to 52 generate
      FULL_ADDERS: full_add
        port map(c1_5r(i), s1_6r(i), c1_6r(i), c2_4(i+1), s2_4(i));
end generate WALLACE_TREE_F2_4;

s2_4(53) <= s1_6r(53);
s2_5(53) <= c1_6r(53);

FA2_4_54: full_add
   port map(c1_5r(54), s1_6r(54), c1_6r(54), c2_4(55), s2_4(54));

FA2_4_55: full_add
   port map(s1_6r(55), c1_6r(55), s1_7r(55), c2_4(56), s2_4(55));

FA2_4_56: full_add
   port map(s1_6r(56), c1_6r(56), s1_7r(56), c2_4(57), s2_4(56));

FA2_4_57: full_add
   port map(c1_6r(57), s1_7r(57), c1_7r(57), c2_4(58), s2_4(57));

FA2_4_58: full_add
   port map(s1_7r(58), c1_7r(58), s1_8r(58), c2_4(59), s2_4(58));

FA2_4_59: full_add
   port map(s1_7r(59), c1_7r(59), s1_8r(59), c2_4(60), s2_4(59));


c2_4(26) <= s1_7r(26);
s2_5(30 downto 28) <= s1_7r(30 downto 28);
c2_5(28) <= c1_7r(28);
c2_5(31) <= c1_8r(31);

WALLACE_TREE_F2_5: for i in 31 to 34 generate
```

```
        FULL_ADDERS: full_add
          port map(s1_7r(i), c1_7r(i), s1_8r(i), c2_5(i+1), s2_5(i));
    end generate WALLACE_TREE_F2_5;

    s2_5(36 downto 35) <= c1_7r(36 downto 35);
    c2_5(36) <= s1_8r(36);
    s2_5(38) <= s1_8r(38);
    c2_5(39) <= c1_8r(39);
    s2_5(40) <= c1_7r(40);
    s2_1(40) <= s1_8r(40);

    FA2_5_39: full_add
      port map(s1_7r(39), c1_7r(39), s1_8r(39), c2_5(40), s2_5(39));

    FA2_5_41: full_add
      port map(c1_7r(41), s1_8r(41), c1_8r(41), c2_5(42), s2_5(41));

    s2_5(43 downto 42) <= c1_7r(43 downto 42);
    c2_5(43) <= s1_8r(43);
    s2_5(45) <= s1_8r(45);
    s2_5(46) <= s1_7r(46);
    c2_5(46) <= c1_7r(46);

    FA2_5_47: full_add
      port map(s1_7r(47), c1_7r(47), s1_8r(47), c2_5(48), s2_5(47));

    FA2_5_48: full_add
      port map(c1_7r(48), s1_8r(48), c1_8r(48), c2_5(49), s2_5(48));

    --s2_5(52 downto 49) <= s1_8r(52 downto 49);
    s2_5(52) <= s1_8r(52);
    s2_5(50 downto 49) <= s1_8r(50 downto 49);
    c2_5(50) <= c1_8r(50);
    s2_5(56 downto 54) <= c1_7r(56 downto 54);
    c2_5(54) <= s1_8r(54);
    c2_4(54) <= s1_7r(54);
    c2_5(56) <= s1_8r(56);
    s2_5(57) <= s1_8r(57);
    c2_5(57) <= c1_8r(57);
    s2_5(59) <= c1_8r(59);
    s2_4(60) <= s1_7r(60);
    s2_5(60) <= c1_7r(60);

    FA2_5_61: full_add
      port map(s1_7r(61), c1_7r(61), s1_8r(61), c2_5(62), s2_5(61));

    s2_5(62) <= s1_7r(62);
    s2_5(72 downto 63) <= s1_8r(72 downto 63);


-- ADD REG HERE
```

```
   temp13 <= c2_5(62) & c2_5(57 downto 56) & c2_5(54) & c2_5(50 downto 48) & c2_5(46) & c2_5(43 downto 42)
          &
                c2_5(40 downto 39) & c2_5(36 downto 31) & c2_5(28) & s2_5(72 downto 59) & s2_5(57 downto
          52) &
                s2_5(50 downto 45) & s2_5(43 downto 38) & s2_5(36 downto 28) & s2_5(25) & c2_4(60 downto
          25) & c2_4(22) &
                s2_4(60 downto 22) & s2_4(18) & c2_3(54 downto 18) & c2_3(15) & s2_3(53 downto 15) &
          s2_3(11) &
          c2_2(46 downto 11) & c2_2(8) & s2_2(45 downto 8) & s2_2(5) & c2_1(39 downto 4) & s2_1(40)
          & s2_1(38 downto 4) & pos10 & sign10 & add_sub10 & mac_mpy10 & MS10;

      temp13_in <= temp13;
-------------------------------------------------------------------------
-- 1
-------------------------------------------------------------------------
--       COMP13_89: comp
--         generic map(373)
--         port map(ko_13, ki10);

      ko_13b1(1 downto 0) <= ko_13(50) & ko_13(13);
      COMP13_1: comp
        generic map(2)
        port map(ko_13b1, ki10b(13) );

      ko_13b2(1 downto 0) <= ko_13(51) & ko_13(14);
      COMP13_2: comp
        generic map(2)
        port map(ko_13b2, ki10b(14) );

   ko_13_b3: for i in 16 to 44 generate
      ko_13b3( (1+ i*2) downto (0 + i*2) ) <= ko_13(i+37) & ko_13(i); -- (53-81) , (16-44)
      COMP13_3: comp
        generic map(2)
        port map(ko_13b3( (1+ i*2) downto (0 + i*2) ), ki10b(i) );
     end generate ko_13_b3;

      ko_13b4(1 downto 0) <= ko_13(82) & ko_13(45);
      COMP13_4: comp
        generic map(2)
        port map(ko_13b4, ki10b(45) );

      ko_13b5(1 downto 0) <= ko_13(50) & ko_13(13);
      COMP13_5: comp
        generic map(2)
        port map(ko_13b5, ki10b(46) );

      ko_13b6(1 downto 0) <= ko_13(51) & ko_13(14);
      COMP13_6: comp
        generic map(2)
        port map(ko_13b6, ki10b(47) );

   ko_13_b7: for i in 49 to 77 generate
      ko_13b7( (1+ i*2) downto (0 + i*2) ) <= ko_13(i+4) & ko_13(i-33); -- (53-81) , (16-44)
```

```
   COMP13_7: comp
     generic map(2)
     port map(ko_13b7( (1+ i*2) downto (0 + i*2) ), ki10b(i) );
  end generate ko_13_b7;

  ko_13b8(1 downto 0) <= ko_13(83) & ko_13(46);
  COMP13_8: comp
    generic map(2)
    port map(ko_13b8, ki10b(78) );

  ko_13b9(1 downto 0) <= ko_13(51) & ko_13(14);
  COMP13_9: comp
    generic map(2)
    port map(ko_13b9, ki10b(79) );

ko_13_b10: for i in 80 to 108 generate
   ko_13b10( (1+ i*2) downto (0 + i*2) ) <= ko_13(i-27) & ko_13(i-64); -- (53-81) , (16-44)
   COMP13_10: comp
     generic map(2)
     port map(ko_13b10( (1+ i*2) downto (0 + i*2) ), ki10b(i) );
  end generate ko_13_b10;

  ko_13b11(1 downto 0) <= ko_13(82) & ko_13(45);
  COMP13_11: comp
    generic map(2)
    port map(ko_13b11, ki10b(109) );

  ko_13b12(1 downto 0) <= ko_13(83) & ko_13(46);
  COMP13_12: comp
    generic map(2)
    port map(ko_13b12, ki10b(110) );

  ko_13b13(1 downto 0) <= ko_13(84) & ko_13(47);
  COMP13_13: comp
    generic map(2)
    port map(ko_13b13, ki10b(111) );

  ko_13b14(1 downto 0) <= ko_13(85) & ko_13(48);
  COMP13_14: comp
    generic map(2)
    port map(ko_13b14, ki10b(112) );

  ko_13b15(1 downto 0) <= ko_13(126) & ko_13(89);
  COMP13_15: comp
    generic map(2)
    port map(ko_13b15, ki10b(116) );

ko_13_b16: for i in 119 to 145 generate
   ko_13b16( (1+ i*2) downto (0 + i*2) ) <= ko_13(i+10) & ko_13(i-27); -- (129-155) , (92-118)
   COMP13_16: comp
     generic map(2)
     port map(ko_13b16( (1+ i*2) downto (0 + i*2) ), ki10b(i) );
  end generate ko_13_b16;
```

```
  ko_13b17(1 downto 0) <= ko_13(126) & ko_13(89);
  COMP13_17: comp
    generic map(2)
    port map(ko_13b17, ki10b(147) );

ko_13_b18: for i in 150 to 176 generate
   ko_13b18( (1+ i*2) downto (0 + i*2) ) <= ko_13(i-21) & ko_13(i-58) ;-- (129-155) , (92-118)
   COMP13_18: comp
     generic map(2)
     port map(ko_13b18( (1+ i*2) downto (0 + i*2) ), ki10b(i) );
  end generate ko_13_b18;

ko_13_b19: for i in 177 to 180 generate
   ko_13b19( (1+ i*2) downto (0 + i*2) ) <= ko_13(i-21) & ko_13(i-58) ;-- (156-159) , (119-122)
   COMP13_19: comp
     generic map(2)
     port map(ko_13b19( (1+ i*2) downto (0 + i*2) ), ki10b(i) );
  end generate ko_13_b19;

  ko_13b20(1 downto 0) <= ko_13(160) & ko_13(123);
  COMP13_20: comp
    generic map(2)
    port map(ko_13b20, ki10b(181) );

  ko_13b21(1 downto 0) <= ko_13(126) & ko_13(89);
  COMP13_21: comp
    generic map(2)
    port map(ko_13b21, ki10b(182) );

ko_13_b22: for i in 183 to 209 generate
   ko_13b22( (1+ i*2) downto (0 + i*2) ) <= ko_13(i-54) & ko_13(i-91) ;-- (129-155) , (92-118)
   COMP13_22: comp
     generic map(2)
     port map(ko_13b22( (1+ i*2) downto (0 + i*2) ), ki10b(i) );
  end generate ko_13_b22;

ko_13_b23: for i in 210 to 213 generate
   ko_13b23( (1+ i*2) downto (0 + i*2) ) <= ko_13(i-54) & ko_13(i-91) ;-- (156-159) , (119-122)
   COMP13_23: comp
     generic map(2)
     port map(ko_13b23( (1+ i*2) downto (0 + i*2) ), ki10b(i) );
  end generate ko_13_b23;

  ko_13b24(1 downto 0) <= ko_13(161) & ko_13(124);
  COMP13_24: comp
    generic map(2)
    port map(ko_13b24, ki10b(214) );

  ko_13b25(1 downto 0) <= ko_13(203) & ko_13(165);
  COMP13_25: comp
    generic map(2)
    port map(ko_13b25, ki10b(218) );
```

```
ko_13_b26: for i in 220 to 248 generate
   ko_13b26( (1+ i*2) downto (0 + i*2) ) <= ko_13(i-15) & ko_13(i-53) ;-- (205-233) , (167-195)
   COMP13_26: comp
    generic map(2)
    port map(ko_13b26( (1+ i*2) downto (0 + i*2) ), ki10b(i) );
 end generate ko_13_b26;

  ko_13b27(1 downto 0) <= ko_13(203) & ko_13(165);
  COMP13_27: comp
   generic map(2)
   port map(ko_13b27, ki10b(250) );

ko_13_b28: for i in 252 to 280 generate
   ko_13b28( (1+ i*2) downto (0 + i*2) ) <= ko_13(i-47) & ko_13(i-85) ;-- (205-233) , (167-195)
   COMP13_28: comp
    generic map(2)
    port map(ko_13b28( (1+ i*2) downto (0 + i*2) ), ki10b(i) );
 end generate ko_13_b28;

  ko_13b29(1 downto 0) <= ko_13(234) & ko_13(196);
  COMP13_29: comp
   generic map(2)
   port map(ko_13b29, ki10b(281) );

  ko_13b30(1 downto 0) <= ko_13(203) & ko_13(165);
  COMP13_30: comp
   generic map(2)
   port map(ko_13b30, ki10b(282) );

ko_13_b31: for i in 283 to 311 generate
   ko_13b31( (1+ i*2) downto (0 + i*2) ) <= ko_13(i-78) & ko_13(i-116) ;-- (205-233) , (167-195)
   COMP13_31: comp
    generic map(2)
    port map(ko_13b31( (1+ i*2) downto (0 + i*2) ), ki10b(i) );
 end generate ko_13_b31;

  ko_13b32(1 downto 0) <= ko_13(234) & ko_13(196);
  COMP13_32: comp
   generic map(2)
   port map(ko_13b32, ki10b(312) );

ko_13_b33: for i in 313 to 317 generate
   ko_13b33( (1+ i*2) downto (0 + i*2) ) <= ko_13(i-78) & ko_13(i-116) ;-- (235-239) , (197-201)
   COMP13_33: comp
    generic map(2)
    port map(ko_13b33( (1+ i*2) downto (0 + i*2) ), ki10b(i) );
 end generate ko_13_b33;

  ko_13b34(1 downto 0) <= ko_13(281) & ko_13(243);
  COMP13_34: comp
   generic map(2)
   port map(ko_13b34, ki10b(321) );
```

```
ko_13_b35: for i in 323 to 349 generate
   ko_13b35( (1+ i*2) downto (0 + i*2) ) <= ko_13(i-40) & ko_13(i-78) ;-- (283-309) , (245-271)
   COMP13_35: comp
     generic map(2)
     port map(ko_13b35( (1+ i*2) downto (0 + i*2) ), ki10b(i) );
  end generate ko_13_b35;

  ko_13b36(1 downto 0) <= ko_13(311) & ko_13(273);
  COMP13_36: comp
    generic map(2)
    port map(ko_13b36, ki10b(350) );

  ko_13b37(1 downto 0) <= ko_13(281) & ko_13(243);
  COMP13_37: comp
    generic map(2)
    port map(ko_13b37, ki10b(352) );

ko_13_b38: for i in 354 to 380 generate
   ko_13b38( (1+ i*2) downto (0 + i*2) ) <= ko_13(i-71) & ko_13(i-109) ;-- (283-309) , (245-271)
   COMP13_38: comp
     generic map(2)
     port map(ko_13b38( (1+ i*2) downto (0 + i*2) ), ki10b(i) );
  end generate ko_13_b38;

  ko_13b39(1 downto 0) <= ko_13(311) & ko_13(273);
  COMP13_39: comp
    generic map(2)
    port map(ko_13b39, ki10b(382) );

  ko_13b40(1 downto 0) <= ko_13(312) & ko_13(274);
  COMP13_40: comp
    generic map(2)
    port map(ko_13b40, ki10b(383) );

  ko_13b41(1 downto 0) <= ko_13(313) & ko_13(275);
  COMP13_41: comp
    generic map(2)
    port map(ko_13b41, ki10b(384) );

  ko_13b42(1 downto 0) <= ko_13(281) & ko_13(243);
  COMP13_42: comp
    generic map(2)
    port map(ko_13b42, ki10b(385) );

ko_13_b43: for i in 386 to 412 generate
   ko_13b43( (1+ i*2) downto (0 + i*2) ) <= ko_13(i-103) & ko_13(i-141) ;-- (283-309) , (245-271)
   COMP13_43: comp
     generic map(2)
     port map(ko_13b43( (1+ i*2) downto (0 + i*2) ), ki10b(i) );
  end generate ko_13_b43;

  ko_13b44(1 downto 0) <= ko_13(311) & ko_13(273);
```

```
  COMP13_44: comp
    generic map(2)
    port map(ko_13b44, ki10b(414) );

  ko_13b45(1 downto 0) <= ko_13(312) & ko_13(274);
  COMP13_45: comp
    generic map(2)
    port map(ko_13b45, ki10b(415) );

  ko_13b46(1 downto 0) <= ko_13(313) & ko_13(275);
  COMP13_46: comp
    generic map(2)
    port map(ko_13b46, ki10b(416) );

  ko_13b47(1 downto 0) <= ko_13(314) & ko_13(276);
  COMP13_47: comp
    generic map(2)
    port map(ko_13b47, ki10b(417) );

ko_13_b48: for i in 422 to 425 generate
   ko_13b48( (1+ i*2) downto (0 + i*2) ) <= ko_13(i-61) & ko_13(i-101) ;-- (361-364) , (321-324)
   COMP13_48: comp
     generic map(2)
     port map(ko_13b48( (1+ i*2) downto (0 + i*2) ), ki10b(i) );
  end generate ko_13_b48;

  ko_13b49(1 downto 0) <= ko_13(82) & ko_13(45);
  COMP13_49: comp
    generic map(2)
    port map(ko_13b49, ki10b(426) );

  ko_13b50(1 downto 0) <= ko_13(83) & ko_13(46);
  COMP13_50: comp
    generic map(2)
    port map(ko_13b50, ki10b(427) );

  ko_13b51(1 downto 0) <= ko_13(84) & ko_13(47);
  COMP13_51: comp
    generic map(2)
    port map(ko_13b51, ki10b(428) );

  ko_13b52(1 downto 0) <= ko_13(85) & ko_13(48);
  COMP13_52: comp
    generic map(2)
    port map(ko_13b52, ki10b(429) );

  ko_13b53(1 downto 0) <= ko_13(367) & ko_13(328);
  COMP13_53: comp
    generic map(2)
    port map(ko_13b53, ki10b(430) );

ko_13_b54: for i in 431 to 434 generate
   ko_13b54( (1+ i*2) downto (0 + i*2) ) <= ko_13(i-275) & ko_13(i-312) ;-- (156-159) , (119-122)
```

```
   COMP13_54: comp
     generic map(2)
     port map(ko_13b54( (1+ i*2) downto (0 + i*2) ), ki10b(i) );
  end generate ko_13_b54;

  ko_13b55(1 downto 0) <= ko_13(160) & ko_13(123);
  COMP13_55: comp
    generic map(2)
    port map(ko_13b55, ki10b(435) );

  ko_13b56(1 downto 0) <= ko_13(161) & ko_13(124);
  COMP13_56: comp
    generic map(2)
    port map(ko_13b56, ki10b(436) );

  ko_13b57(1 downto 0) <= ko_13(371) & ko_13(335);
  COMP13_57: comp
    generic map(2)
    port map(ko_13b57, ki10b(438) );

  ko_13b58(1 downto 0) <= ko_13(234) & ko_13(196);
  COMP13_58: comp
    generic map(2)
    port map(ko_13b58, ki10b(439) );

ko_13_b59: for i in 440 to 444 generate
   ko_13b59( (1+ i*2) downto (0 + i*2) ) <= ko_13(i-205) & ko_13(i-243) ;-- (235-239) , (197-201)
   COMP13_59: comp
     generic map(2)
     port map(ko_13b59( (1+ i*2) downto (0 + i*2) ), ki10b(i) );
  end generate ko_13_b59;

  ko_13b60(1 downto 0) <= ko_13(312) & ko_13(274);
  COMP13_60: comp
    generic map(2)
    port map(ko_13b60, ki10b(446) );

  ko_13b61(1 downto 0) <= ko_13(313) & ko_13(275);
  COMP13_61: comp
    generic map(2)
    port map(ko_13b61, ki10b(447) );

  ko_13b62(1 downto 0) <= ko_13(314) & ko_13(276);
  COMP13_62: comp
    generic map(2)
    port map(ko_13b62, ki10b(448) );

  ko_13b63(1 downto 0) <= ko_13(315) & ko_13(277);
  COMP13_63: comp
    generic map(2)
    port map(ko_13b63, ki10b(449) );

  ko_13b64(1 downto 0) <= ko_13(316) & ko_13(278);
```

```
  COMP13_64: comp
    generic map(2)
    port map(ko_13b64, ki10b(450) );

  ko_13b65(1 downto 0) <= ko_13(377) & ko_13(347);
  COMP13_65: comp
    generic map(2)
    port map(ko_13b65, ki10b(452) );

ko_13_b66: for i in 455 to 458 generate
    ko_13b66( (1+ i*2) downto (0 + i*2) ) <= ko_13(i-94) & ko_13(i-134) ;-- (361-364) , (321-324)
    COMP13_66: comp
      generic map(2)
      port map(ko_13b66( (1+ i*2) downto (0 + i*2) ), ki10b(i) );
  end generate ko_13_b66;

  ko_13b67(1 downto 0) <= ko_13(84) & ko_13(47);
  COMP13_67: comp
    generic map(2)
    port map(ko_13b67, ki10b(461) );

  ko_13b68(1 downto 0) <= ko_13(85) & ko_13(48);
  COMP13_68: comp
    generic map(2)
    port map(ko_13b68, ki10b(462) );

  ko_13b69(1 downto 0) <= ko_13(367) & ko_13(328);
  COMP13_69: comp
    generic map(2)
    port map(ko_13b69, ki10b(463) );

  ko_13b70(1 downto 0) <= ko_13(368) & ko_13(330);
  COMP13_70: comp
    generic map(2)
    port map(ko_13b70, ki10b(465) );

  ko_13b71(1 downto 0) <= ko_13(160) & ko_13(123);
  COMP13_71: comp
    generic map(2)
    port map(ko_13b71, ki10b(468) );

  ko_13b72(1 downto 0) <= ko_13(161) & ko_13(124);
  COMP13_72: comp
    generic map(2)
    port map(ko_13b72, ki10b(469) );

  ko_13b73(1 downto 0) <= ko_13(371) & ko_13(335);
  COMP13_73: comp
    generic map(2)
    port map(ko_13b73, ki10b(471) );

  ko_13b74(1 downto 0) <= ko_13(372) & ko_13(336);
  COMP13_74: comp
```

```
    generic map(2)
    port map(ko_13b74, ki10b(472) );

ko_13_b75: for i in 473 to 477 generate
   ko_13b75( (1+ i*2) downto (0 + i*2) ) <= ko_13(i-238) & ko_13(i-276) ;-- (235-239) , (197-201)
   COMP13_75: comp
     generic map(2)
     port map(ko_13b75( (1+ i*2) downto (0 + i*2) ), ki10b(i) );
  end generate ko_13_b75;

  ko_13b76(1 downto 0) <= ko_13(314) & ko_13(276);
  COMP13_76: comp
    generic map(2)
    port map(ko_13b76, ki10b(481) );

  ko_13b77(1 downto 0) <= ko_13(315) & ko_13(277);
  COMP13_77: comp
    generic map(2)
    port map(ko_13b77, ki10b(482) );

  ko_13b78(1 downto 0) <= ko_13(316) & ko_13(278);
  COMP13_78: comp
    generic map(2)
    port map(ko_13b78, ki10b(483) );

  ko_13b79(1 downto 0) <= ko_13(377) & ko_13(347);
  COMP13_79: comp
    generic map(2)
    port map(ko_13b79, ki10b(485) );

ko_13_b80: for i in 486 to 489 generate
   ko_13b80( (1+ i*2) downto (0 + i*2) ) <= ko_13(i-125) & ko_13(i-165) ;-- (361-364) , (321-324)
   COMP13_80: comp
     generic map(2)
     port map(ko_13b80( (1+ i*2) downto (0 + i*2) ), ki10b(i) );
  end generate ko_13_b80;

  ko_13b81(1 downto 0) <= ko_13(367) & ko_13(328);
  COMP13_81: comp
    generic map(2)
    port map(ko_13b81, ki10b(492) );

  ko_13b82(1 downto 0) <= ko_13(368) & ko_13(330);
  COMP13_82: comp
    generic map(2)
    port map(ko_13b82, ki10b(494) );

  ko_13b83(1 downto 0) <= ko_13(371) & ko_13(335);
  COMP13_83: comp
    generic map(2)
    port map(ko_13b83, ki10b(497) );

  ko_13b84(1 downto 0) <= ko_13(372) & ko_13(336);
```

```
      COMP13_84: comp
        generic map(2)
        port map(ko_13b84, ki10b(498) );

      ko_13b85(1 downto 0) <= ko_13(315) & ko_13(277);
      COMP13_85: comp
        generic map(2)
        port map(ko_13b85, ki10b(505) );

      ko_13b86(1 downto 0) <= ko_13(316) & ko_13(278);
      COMP13_86: comp
        generic map(2)
        port map(ko_13b86, ki10b(506) );

      ko_13b87(1 downto 0) <= ko_13(377) & ko_13(347);
      COMP13_87: comp
        generic map(2)
        port map(ko_13b87, ki10b(507) );

      ko_13b88(1 downto 0) <= ko_13(368) & ko_13(330);
      COMP13_88: comp
        generic map(2)
        port map(ko_13b88, ki10b(520) );

      ko_13b89(1 downto 0) <= ko_13(372) & ko_13(336);
      COMP13_89: comp
        generic map(2)
        port map(ko_13b89, ki10b(521) );
-------------------------------------------------------------------------
-------------------------------------------------------------------------
    REG13: ncl_register_D
       generic map(373, -4)
       port map(temp13_in, ki_13, reset, temp13_out, ko_13);
-------------------------------------------------------------------------------
--                                                    2
-------------------------------------------------------------------
              ki_13(5) <= ko_15(0);
              ki_13(6) <= ko_15(1);
              ki_13(7) <= ko_15(2);
              ki_13(9 downto 8) <= ko_15(4 downto 3);
              ki_13(13 downto 10) <= ko_15(8 downto 5);

              ki_13(14) <= ki15b(14);
              ki_13(15) <= ki15b(15);
              ki_13(17 downto 16) <= ko_15(12 downto 11);
              ACK13_1: for i in 18 to 48 generate
                        ki_13(i) <= ki15b(i);
              end generate ACK13_1;
              ki_13(49) <= ki15b(49);

              ki_13(50) <= ki15b(50);
              ki_13(51) <= ki15b(51);
              ki_13(52) <= ko_15(91);
```

```
ki_13(53) <= ko_15(53);
ACK13_2: for i in 54 to 84 generate
        ki_13(i) <= ki15b(i);
end generate ACK13_2;
ki_13(85) <= ki15b(85);

ki_13(86) <= ki15b(86);
ACK13_3: for i in 87 to 117 generate
        ki_13(i) <= ki15b(i);
end generate ACK13_3;
ki_13(118) <= ki15b(118);
ki_13(119) <= ki15b(119);
ki_13(124 downto 120) <= ko_15(50 downto 46);

ki_13(125) <= ko_15(54);
ki_13(129 downto 126) <= ko_15(95 downto 92);
ki_13(130) <= ki15b(130);
ki_13(132 downto 131) <= ko_15(98 downto 97);
ACK13_4: for i in 133 to 161 generate
        ki_13(i) <= ki15b(i);
end generate ACK13_4;

ki_13(162) <= ko_15(136);
ki_13(163) <= ki15b(163);
ki_13(164) <= ko_15(175);
ki_13(165) <= ko_15(138);
ACK13_5: for i in 166 to 194 generate
        ki_13(i) <= ki15b(i);
end generate ACK13_5;
ki_13(195) <= ko_15(128);
ACK13_6: for i in 196 to 198 generate
        ki_13(i) <= ki15b(i);
end generate ACK13_6;
ki_13(199) <= ki15b(199);
ki_13(201 downto 200) <= ko_15(134 downto 133);

ki_13(202) <= ki15b(202);
ACK13_7: for i in 203 to 231 generate
        ki_13(i) <= ki15b(i);
end generate ACK13_7;
ki_13(232) <= ko_15(271);
ACK13_8: for i in 233 to 235 generate
        ki_13(i) <= ki15b(i);
end generate ACK13_8;
ki_13(236) <= ki15b(236);
ki_13(237) <= ko_15(252);
ki_13(238) <= ko_15(173);
ki_13(239) <= ko_15(135);

ki_13(240) <= ko_15(139);
ki_13(243 downto 241) <= ko_15(178 downto 176);
ki_13(244) <= ki15b(244);
ki_13(246 downto 245) <= ko_15(181 downto 180);
```

```
ACK13_9: for i in 247 to 255 generate
        ki_13(i) <= ki15b(i);
end generate ACK13_9;
ki_13(256) <= ko_15(191);
ACK13_10: for i in 257 to 262 generate
        ki_13(i) <= ki15b(i);
end generate ACK13_10;
ki_13(263) <= ko_15(198);
ACK13_11: for i in 264 to 269 generate
        ki_13(i) <= ki15b(i);
end generate ACK13_11;
ki_13(270) <= ki15b(270);
ACK13_12: for i in 271 to 276 generate
        ki_13(i) <= ki15b(i);
end generate ACK13_12;
ki_13(277) <= ko_15(212);
ki_13(278) <= ki15b(278);
ki_13(279) <= ki15b(279);

ki_13(280) <= ko_15(227);
ki_13(281) <= ki15b(281);
ki_13(282) <= ko_15(262);
ki_13(283) <= ko_15(229);
ACK13_13: for i in 284 to 292 generate
        ki_13(i) <= ki15b(i);
end generate ACK13_13;
ki_13(293) <= ko_15(269);
ACK13_14: for i in 294 to 299 generate
        ki_13(i) <= ki15b(i);
end generate ACK13_14;
ki_13(300) <= ko_15(90);
ACK13_15: for i in 301 to 306 generate
        ki_13(i) <= ki15b(i);
end generate ACK13_15;
ki_13(307) <= ko_15(205);
ACK13_16: for i in 308 to 313 generate
        ki_13(i) <= ki15b(i);
end generate ACK13_16;
ki_13(314) <= ko_15(274);
ki_13(315) <= ki15b(315);
ki_13(316) <= ki15b(316);

ki_13(317) <= ki15b(317);
ACK13_17: for i in 318 to 326 generate
        ki_13(i) <= ki15b(i);
end generate ACK13_17;
ACK13_18: for i in 327 to 332 generate
        ki_13(i) <= ki15b(i);
end generate ACK13_18;
ACK13_19: for i in 333 to 338 generate
        ki_13(i) <=  ki15b(i);
end generate ACK13_19;
ACK13_20: for i in 339 to 344 generate
```

```
                ki_13(i) <= ki15b(i);
        end generate ACK13_20;
        ki_13(345) <= ki15b(345);
        ki_13(346) <= ki15b(346);
        ki_13(358 downto 347) <= ko_15(226 downto 215);

        ki_13(359) <= ko_15(230);
        ki_13(365 downto 360) <= ko_15(268 downto 263);
        ki_13(366) <= ki15b(366);
        ki_13(367) <= ki15b(367);
        ki_13(369 downto 368) <= ko_15(89 downto 88);
        ki_13(370) <= ko_15(270);
        ACK13_21: for i in 371 to 373 generate
                ki_13(i) <=  ki15b(i);
        end generate ACK13_21;
        ki_13(374) <= ko_15(174);
        ki_13(376 downto 375) <= ko_15(273 downto 272);
        ki_13(377) <= ko_15(261);

c2_5r(62) <= temp13_out(377);
c2_5r(57 downto 56) <= temp13_out(376 downto 375);
c2_5r(54) <= temp13_out(374);
c2_5r(50 downto 48) <= temp13_out(373 downto 371);
c2_5r(46) <= temp13_out(370);
c2_5r(43 downto 42) <= temp13_out(369 downto 368);
c2_5r(40 downto 39) <= temp13_out(367 downto 366);
c2_5r(36 downto 31) <= temp13_out(365 downto 360);
c2_5r(28) <= temp13_out(359);
s2_5r(72 downto 59) <= temp13_out(358 downto 345);
s2_5r(57 downto 52) <= temp13_out(344 downto 339);
s2_5r(50 downto 45) <= temp13_out(338 downto 333);
s2_5r(43 downto 38) <= temp13_out(332 downto 327);
s2_5r(36 downto 28) <= temp13_out(326 downto 318);
s2_5r(25) <= temp13_out(317);
c2_4r(60 downto 25) <= temp13_out(316 downto 281);
c2_4r(22) <= temp13_out(280);
s2_4r(60 downto 22) <= temp13_out(279 downto 241);
s2_4r(18) <= temp13_out(240);
c2_3r(54 downto 18) <= temp13_out(239 downto 203);
c2_3r(15) <= temp13_out(202);
s2_3r(53 downto 15) <= temp13_out(201 downto 163);
s2_3r(11) <= temp13_out(162);
c2_2r(46 downto 11) <= temp13_out(161 downto 126);
c2_2r(8) <= temp13_out(125);
s2_2r(45 downto 8) <= temp13_out(124 downto 87);
s2_2r(5) <= temp13_out(86);
c2_1r(39 downto 4) <= temp13_out(85 downto 50);
s2_1r(40) <= temp13_out(49);
s2_1r(38 downto 4) <= temp13_out(48 downto 14);
pos15(3 downto 0) <= temp13_out(13 downto 10);
sign15i <= temp13_out(9 downto 8);
add_sub15i <= temp13_out(7);
mac_mpy15i <= temp13_out(6);
```

```
   MS15i <= temp13_out(5);

-- FORTH LEVEL (add register)

   HA_POS4_2: half_add
      port map(s2_1r(4), c2_1r(4), c3_1(5), pos15(4));

   FA3_1_5: full_add
      port map(s2_1r(5), c2_1r(5), s2_2r(5), c3_1(6), s3_1(5));

   s3_1(7 downto 6) <= s2_1r(7 downto 6);
   s3_2(6) <= c2_1r(6);
   c3_1(7) <= c2_1r(7);

   WALLACE_TREE_1_F3: for i in 8 to 38 generate
         FULL_ADDERS: full_add
            port map(s2_1r(i), c2_1r(i), s2_2r(i), c3_1(i+1), s3_1(i));
   end generate WALLACE_TREE_1_F3;

   FA3_1_39: full_add
      port map(c2_1r(39), s2_2r(39), c2_5r(39), c3_1(40), s3_1(39));

   FA3_1_40: full_add
      port map(s2_1r(40), s2_2r(40), c2_5r(40), c3_1(41), s3_1(40));

   s3_1(45 downto 41) <= s2_2r(45 downto 41);


   c3_1(8) <= c2_2r(8);
   s3_2(14 downto 11) <= c2_2r(14 downto 11);
   c3_2(11) <= s2_3r(11);

   FA3_2_15: full_add
      port map(c2_2r(15), s2_3r(15), c2_3r(15), c3_2(16), s3_2(15));

   s3_2(17 downto 16) <= c2_2r(17 downto 16);
   s3_3(16) <= s2_3r(16);
   c3_2(17) <= s2_3r(17);

   WALLACE_TREE_2_F3: for i in 18 to 46 generate
         FULL_ADDERS: full_add
            port map(c2_2r(i), s2_3r(i), c2_3r(i), c3_2(i+1), s3_2(i));
   end generate WALLACE_TREE_2_F3;

   s3_2(47) <= s2_3r(47);
   s3_4(47) <= c2_3r(47);

   WALLACE_TREE_2_F3L: for i in 48 to 50 generate
         FULL_ADDERS: full_add
            port map(s2_3r(i), c2_3r(i), c2_5r(i), c3_2(i+1), s3_2(i));
   end generate WALLACE_TREE_2_F3L;

   FA3_2_51: full_add
```

```
   port map(s2_3r(51), c2_3r(51), s2_4r(51), c3_2(52), s3_2(51));

c3_3(52) <= c2_3r(52);
s3_2(53 downto 52) <= s2_3r(53 downto 52);
s3_2(54) <= c2_3r(54);
c3_2(53) <= c2_3r(53);


c3_2(18) <= s2_4r(18);
s3_3(24 downto 22) <= s2_4r(24 downto 22);
c3_3(22) <= c2_4r(22);

FA3_3_25: full_add
   port map(s2_4r(25), c2_4r(25), s2_5r(25), c3_3(26), s3_3(25));

s3_3(27 downto 26) <= s2_4r(27 downto 26);
s3_4(26) <= c2_4r(26);
c3_3(27) <= c2_4r(27);

WALLACE_TREE_3_F3R: for i in 28 to 36 generate
      FULL_ADDERS: full_add
         port map(s2_4r(i), c2_4r(i), s2_5r(i), c3_3(i+1), s3_3(i));
end generate WALLACE_TREE_3_F3R;

s3_3(37) <= s2_4r(37);
s3_4(37) <= c2_4r(37);

WALLACE_TREE_3_F3MR: for i in 38 to 43 generate
      FULL_ADDERS: full_add
         port map(s2_4r(i), c2_4r(i), s2_5r(i), c3_3(i+1), s3_3(i));
end generate WALLACE_TREE_3_F3MR;

s3_3(44) <= s2_4r(44);
c3_1(44) <= c2_4r(44);

WALLACE_TREE_3_F3ML: for i in 45 to 50 generate
      FULL_ADDERS: full_add
         port map(s2_4r(i), c2_4r(i), s2_5r(i), c3_3(i+1), s3_3(i));
end generate WALLACE_TREE_3_F3ML;

s3_3(51) <= c2_4r(51);

WALLACE_TREE_3_F3L: for i in 52 to 57 generate
      FULL_ADDERS: full_add
         port map(s2_4r(i), c2_4r(i), s2_5r(i), c3_3(i+1), s3_3(i));
end generate WALLACE_TREE_3_F3L;

s3_4(58) <= c2_4r(58);
s3_3(58) <= s2_4r(58);

FA3_3_59: full_add
   port map(s2_4r(59), c2_4r(59), s2_5r(59), c3_3(60), s3_3(59));
```

```
   FA3_3_60: full_add
      port map(s2_4r(60), c2_4r(60), s2_5r(60), c3_3(61), s3_3(60));

   s3_3(72 downto 61) <= s2_5r(72 downto 61);
   c3_3(28) <= c2_5r(28);
   s3_4(36 downto 31) <= c2_5r(36 downto 31);
   c3_1(43 downto 42) <= c2_5r(43 downto 42);
   s3_4(46) <= c2_5r(46);
   c3_2(54) <= c2_5r(54);
   s3_4(57 downto 56) <= c2_5r(57 downto 56);
   c3_3(62) <= c2_5r(62);

-- ADD REGISTER

   temp15 <= s3_4(58 downto 56) & s3_4(47 downto 46) & s3_4(37 downto 31) & s3_4(26) & c3_3(62 downto 60)
          & c3_3(58 downto 52) &
                  c3_3(51 downto 46) & c3_3(44 downto 39) &
                  c3_3(37 downto 26) & c3_3(22) & s3_3(72 downto 22) & s3_3(16) & c3_2(54 downto 49) &
          c3_2(47 downto 16) &
          c3_2(11) & s3_2(54 downto 11) & s3_2(6) & c3_1(44 downto 5) &
          s3_1(45 downto 5) & pos15 & sign15i & add_sub15i & mac_mpy15i & MS15i;

      temp15_in <= temp15;

     ko_15b1(1 downto 0) <= ko_15(51) & ko_15(9);
     COMP15_1: comp
      generic map(2)
      port map(ko_15b1, ki15b(14));

     ko_15b2(1 downto 0) <= ko_15(52) & ko_15(10);
     COMP15_2: comp
      generic map(2)
      port map(ko_15b2, ki15b(15));

   ko_15_b3: for i in 18 to 48 generate
      ko_15b3( (1+ i*2) downto (0 + i*2) ) <=  ko_15(i+37) & ko_15(i-5);-- (55-85) , (13-43)
      COMP15_3: comp
       generic map(2)
       port map(ko_15b3( (1+ i*2) downto (0 + i*2) ), ki15b(i) );
    end generate ko_15_b3;

     ko_15b4(1 downto 0) <= ko_15(87) & ko_15(45);
     COMP15_4: comp
      generic map(2)
      port map(ko_15b4, ki15b(49));

     ko_15b5(1 downto 0) <= ko_15(51) & ko_15(9);
     COMP15_5: comp
      generic map(2)
      port map(ko_15b5, ki15b(50));

     ko_15b6(1 downto 0) <= ko_15(52) & ko_15(10);
     COMP15_6: comp
```

```
    generic map(2)
    port map(ko_15b6, ki15b(51));

ko_15_b7: for i in 54 to 84 generate
   ko_15b7( (1+ i*2) downto (0 + i*2) ) <=  ko_15(i+1) & ko_15(i-41);-- (55-85) , (13-43)
   COMP15_7: comp
    generic map(2)
    port map(ko_15b7( (1+ i*2) downto (0 + i*2) ), ki15b(i) );
 end generate ko_15_b7;

  ko_15b8(1 downto 0) <= ko_15(86) & ko_15(44);
  COMP15_8: comp
   generic map(2)
   port map(ko_15b8, ki15b(85));

  ko_15b9(1 downto 0) <= ko_15(52) & ko_15(10);
  COMP15_9: comp
   generic map(2)
   port map(ko_15b9, ki15b(86));

ko_15_b10: for i in 87 to 117 generate
   ko_15b10( (1+ i*2) downto (0 + i*2) ) <= ko_15(i-32) & ko_15(i-74);-- (55-85) , (13-43)
   COMP15_10: comp
    generic map(2)
    port map(ko_15b10( (1+ i*2) downto (0 + i*2) ), ki15b(i) );
 end generate ko_15_b10;

  ko_15b11(1 downto 0) <= ko_15(86) & ko_15(44);
  COMP15_11: comp
   generic map(2)
   port map(ko_15b11, ki15b(118));

  ko_15b12(1 downto 0) <= ko_15(87) & ko_15(45);
  COMP15_12: comp
   generic map(2)
   port map(ko_15b12, ki15b(119));

  ko_15b13(1 downto 0) <= ko_15(137) & ko_15(96);
  COMP15_13: comp
   generic map(2)
   port map(ko_15b13, ki15b(130));

ko_15_b14: for i in 133 to 161 generate
   ko_15b14( (1+ i*2) downto (0 + i*2) ) <= ko_15(i+7) & ko_15(i-34);-- (140-168) , (99-127)
   COMP15_14: comp
    generic map(2)
    port map(ko_15b14( (1+ i*2) downto (0 + i*2) ), ki15b(i) );
 end generate ko_15_b14;

  ko_15b15(1 downto 0) <= ko_15(137) & ko_15(96);
  COMP15_15: comp
   generic map(2)
   port map(ko_15b15, ki15b(163));
```

```
ko_15_b16: for i in 166 to 194 generate
   ko_15b16( (1+ i*2) downto (0 + i*2) ) <= ko_15(i-26) & ko_15(i-67);-- (140-168) , (99-127)
   COMP15_16: comp
     generic map(2)
     port map(ko_15b16( (1+ i*2) downto (0 + i*2) ), ki15b(i) );
  end generate ko_15_b16;

ko_15_b17: for i in 196 to 198 generate
   ko_15b17( (1+ i*2) downto (0 + i*2) ) <= ko_15(i-27) & ko_15(i-67);-- (169-171) , (129-131)
   COMP15_17: comp
     generic map(2)
     port map(ko_15b17( (1+ i*2) downto (0 + i*2) ), ki15b(i) );
  end generate ko_15_b17;

  ko_15b18(1 downto 0) <= ko_15(172) & ko_15(132);
  COMP15_18: comp
    generic map(2)
    port map(ko_15b18, ki15b(199));

  ko_15b19(1 downto 0) <= ko_15(137) & ko_15(96);
  COMP15_19: comp
    generic map(2)
    port map(ko_15b19, ki15b(202));

ko_15_b20: for i in 203 to 231 generate
   ko_15b20( (1+ i*2) downto (0 + i*2) ) <= ko_15(i-63) & ko_15(i-104);-- (140-168) , (99-127)
   COMP15_20: comp
     generic map(2)
     port map(ko_15b20( (1+ i*2) downto (0 + i*2) ), ki15b(i) );
  end generate ko_15_b20;

ko_15_b21: for i in 233 to 235 generate
   ko_15b21( (1+ i*2) downto (0 + i*2) ) <= ko_15(i-64) & ko_15(i-104);-- (169-171) , (129-131)
   COMP15_21: comp
     generic map(2)
     port map(ko_15b21( (1+ i*2) downto (0 + i*2) ), ki15b(i) );
  end generate ko_15_b21;

  ko_15b22(1 downto 0) <= ko_15(172) & ko_15(132);
  COMP15_22: comp
    generic map(2)
    port map(ko_15b22, ki15b(236));

  ko_15b23(1 downto 0) <= ko_15(228) & ko_15(179);
  COMP15_23: comp
    generic map(2)
    port map(ko_15b23, ki15b(244));

ko_15_b24: for i in 247 to 255 generate
   ko_15b24( (1+ i*2) downto (0 + i*2) ) <= ko_15(i-16) & ko_15(i-65);-- (231-239) , (182-190)
   COMP15_24: comp
     generic map(2)
```

```
    port map(ko_15b24( (1+ i*2) downto (0 + i*2) ), ki15b(i) );
 end generate ko_15_b24;

ko_15_b25: for i in 257 to 262 generate
   ko_15b25( (1+ i*2) downto (0 + i*2) ) <= ko_15(i-17) & ko_15(i-65);-- (240-245) , (192-197)
   COMP15_25: comp
    generic map(2)
    port map(ko_15b25( (1+ i*2) downto (0 + i*2) ), ki15b(i) );
 end generate ko_15_b25;

ko_15_b26: for i in 264 to 269 generate
   ko_15b26( (1+ i*2) downto (0 + i*2) ) <= ko_15(i-18) & ko_15(i-65);-- (246-251) , (199-204)
   COMP15_26: comp
    generic map(2)
    port map(ko_15b26( (1+ i*2) downto (0 + i*2) ), ki15b(i) );
 end generate ko_15_b26;

  ko_15b27(1 downto 0) <= ko_15(172) & ko_15(132);
  COMP15_27: comp
   generic map(2)
   port map(ko_15b27, ki15b(270));

ko_15_b28: for i in 271 to 276 generate
   ko_15b28( (1+ i*2) downto (0 + i*2) ) <= ko_15(i-18) & ko_15(i-65);-- (253-258) , (206-211)
   COMP15_28: comp
    generic map(2)
    port map(ko_15b28( (1+ i*2) downto (0 + i*2) ), ki15b(i) );
 end generate ko_15_b28;

  ko_15b29(1 downto 0) <= ko_15(259) & ko_15(213);
  COMP15_29: comp
   generic map(2)
   port map(ko_15b29, ki15b(278));

  ko_15b30(1 downto 0) <= ko_15(260) & ko_15(214);
  COMP15_30: comp
   generic map(2)
   port map(ko_15b30, ki15b(279));

  ko_15b31(1 downto 0) <= ko_15(228) & ko_15(179);
  COMP15_31: comp
   generic map(2)
   port map(ko_15b31, ki15b(281));

ko_15_b32: for i in 284 to 292 generate
   ko_15b32( (1+ i*2) downto (0 + i*2) ) <= ko_15(i-53) & ko_15(i-102);-- (231-239) , (182-190)
   COMP15_32: comp
    generic map(2)
    port map(ko_15b32( (1+ i*2) downto (0 + i*2) ), ki15b(i) );
 end generate ko_15_b32;

ko_15_b33: for i in 294 to 299 generate
   ko_15b33( (1+ i*2) downto (0 + i*2) ) <= ko_15(i-54) & ko_15(i-102);-- (240-245) , (192-197)
```

```
   COMP15_33: comp
    generic map(2)
    port map(ko_15b33( (1+ i*2) downto (0 + i*2) ), ki15b(i) );
  end generate ko_15_b33;

ko_15_b34: for i in 301 to 306 generate
   ko_15b34( (1+ i*2) downto (0 + i*2) ) <= ko_15(i-55) & ko_15(i-102);-- (246-251) , (199-204)
   COMP15_34: comp
    generic map(2)
    port map(ko_15b34( (1+ i*2) downto (0 + i*2) ), ki15b(i) );
  end generate ko_15_b34;

ko_15_b35: for i in 308 to 313 generate
   ko_15b35( (1+ i*2) downto (0 + i*2) ) <= ko_15(i-55) & ko_15(i-102);-- (253-258) , (206-211)
   COMP15_35: comp
    generic map(2)
    port map(ko_15b35( (1+ i*2) downto (0 + i*2) ), ki15b(i) );
  end generate ko_15_b35;

  ko_15b36(1 downto 0) <= ko_15(259) & ko_15(213);
  COMP15_36: comp
   generic map(2)
   port map(ko_15b36, ki15b(315));

  ko_15b37(1 downto 0) <= ko_15(260) & ko_15(214);
  COMP15_37: comp
   generic map(2)
   port map(ko_15b37, ki15b(316));

  ko_15b38(1 downto 0) <= ko_15(228) & ko_15(179);
  COMP15_38: comp
   generic map(2)
   port map(ko_15b38, ki15b(317));

ko_15_b39: for i in 318 to 326 generate
   ko_15b39( (1+ i*2) downto (0 + i*2) ) <= ko_15(i-87) & ko_15(i-136);-- (231-239) , (182-190)
   COMP15_39: comp
    generic map(2)
    port map(ko_15b39( (1+ i*2) downto (0 + i*2) ), ki15b(i) );
  end generate ko_15_b39;

ko_15_b40: for i in 327 to 332 generate
   ko_15b40( (1+ i*2) downto (0 + i*2) ) <= ko_15(i-87) & ko_15(i-135);-- (240-245) , (192-197)
   COMP15_40: comp
    generic map(2)
    port map(ko_15b40( (1+ i*2) downto (0 + i*2) ), ki15b(i) );
  end generate ko_15_b40;

ko_15_b41: for i in 333 to 338 generate
   ko_15b41( (1+ i*2) downto (0 + i*2) ) <= ko_15(i-87) & ko_15(i-134);-- (246-251) , (199-204)
   COMP15_41: comp
    generic map(2)
    port map(ko_15b41( (1+ i*2) downto (0 + i*2) ), ki15b(i) );
```

```
    end generate ko_15_b41;

   ko_15_b42: for i in 339 to 344 generate
      ko_15b42( (1+ i*2) downto (0 + i*2) ) <= ko_15(i-86) & ko_15(i-133);-- (253-258) , (206-211)
      COMP15_42: comp
        generic map(2)
        port map(ko_15b42( (1+ i*2) downto (0 + i*2) ), ki15b(i) );
    end generate ko_15_b42;

     ko_15b43(1 downto 0) <= ko_15(259) & ko_15(213);
     COMP15_43: comp
       generic map(2)
       port map(ko_15b43, ki15b(345));

     ko_15b44(1 downto 0) <= ko_15(260) & ko_15(214);
     COMP15_44: comp
       generic map(2)
       port map(ko_15b44, ki15b(346));

     ko_15b45(1 downto 0) <= ko_15(86) & ko_15(44);
     COMP15_45: comp
       generic map(2)
       port map(ko_15b45, ki15b(366));

     ko_15b46(1 downto 0) <= ko_15(87) & ko_15(45);
     COMP15_46: comp
       generic map(2)
       port map(ko_15b46, ki15b(367));

   ko_15_b47: for i in 371 to 373 generate
      ko_15b47( (1+ i*2) downto (0 + i*2) ) <= ko_15(i-202) & ko_15(i-242);-- (169-171) , (129-131)
      COMP15_47: comp
        generic map(2)
        port map(ko_15b47( (1+ i*2) downto (0 + i*2) ), ki15b(i) );
    end generate ko_15_b47;

   REG15: ncl_register_D
      generic map(275, -4)
      port map(temp15_in, ki_15, reset, temp15_out, ko_15);
--------------------------------------------------------------------------------
--                                                3
--------------------------------------------------------------------------------
  ki_15(0) <= ko_7(5);
            ki_15(1) <= ko_7(6);
            ki_15(2) <= ko_7(7);
            ki_15(4 downto 3) <= ko_7(9 downto 8);
            ki_15(9 downto 5) <= ko_7(14 downto 10);

            ki_15(10) <= ki7b(10);
            ki_15(11) <= ki7b(11);
            ki_15(15 downto 12) <= ko_7(20 downto 17);
            ACK15_1: for i in 16 to 49 generate
                        ki_15(i) <= ki7b(i);
```

```
end generate ACK15_1;
ki_15(50) <= ki7b(50);

ki_15(51) <= ki7b(51);
ki_15(52) <= ki7b(52);
ki_15(53) <= ko_7(139);
ki_15(56 downto 54) <= ko_7(87 downto 85);
ACK15_2: for i in 57 to 90 generate
        ki_15(i) <= ki7b(i);
end generate ACK15_2;

ki_15(91) <= ki7b(91);
ACK15_3: for i in 92 to 125 generate
        ki_15(i) <= ki7b(i);
end generate ACK15_3;
ki_15(126) <= ki7b(126);
ki_15(127) <= ki7b(127);
ki_15(128) <= ki7b(128);
ki_15(129) <= ki7b(129);
ACK15_4: for i in 130 to 135 generate
        ki_15(i) <= ki7b(i);
end generate ACK15_4;

ki_15(136) <= ko_7(88);
ki_15(142 downto 137) <= ko_7(145 downto 140);
ki_15(143) <= ki7b(143);
ki_15(146 downto 144) <= ko_7(149 downto 147);
ACK15_5: for i in 147 to 158 generate
        ki_15(i) <= ki7b(i);
end generate ACK15_5;
ki_15(159) <= ko_7(162);
ACK15_6: for i in 160 to 165 generate
        ki_15(i) <= ki7b(i);
end generate ACK15_6;
ki_15(166) <= ki7b(166);
ki_15(167) <= ki7b(167);
ki_15(168) <= ki7b(168);
ACK15_7: for i in 169 to 174 generate
        ki_15(i) <= ki7b(i);
end generate ACK15_7;

ki_15(175) <= ko_7(179);
ki_15(176) <= ki7b(176);
ki_15(177) <= ko_7(204);
ki_15(179 downto 178) <= ko_7(182 downto 181);
ACK15_8: for i in 180 to 191 generate
        ki_15(i) <= ki7b(i);
end generate ACK15_8;
ki_15(192) <= ko_7(212);
ACK15_9: for i in 193 to 198 generate
        ki_15(i) <= ki7b(i);
end generate ACK15_9;
ki_15(199) <= ko_7(169);
```

```
		ki_15(200) <= ki7b(200);
		ki_15(201) <= ki7b(201);
		ki_15(202) <= ki7b(202);
		ACK15_10: for i in 203 to 208 generate
			ki_15(i) <= ki7b(i);
		end generate ACK15_10;
		ki_15(209) <= ko_7(65);
		ACK15_11: for i in 210 to 212 generate
			ki_15(i) <= ki7b(i);
		end generate ACK15_11;
		ki_15(226 downto 213) <= ko_7(82 downto 69);

		ki_15(227) <= ki7b(227);
		ACK15_12: for i in 228 to 239 generate
			ki_15(i) <= ki7b(i);
		end generate ACK15_12;
		ACK15_13: for i in 240 to 245 generate
			ki_15(i) <= ki7b(i);
		end generate ACK15_13;
		ki_15(247 downto 246) <= ko_7(171 downto 170);
		ki_15(248) <= ki7b(248);
		ki_15(255 downto 249) <= ko_7(178 downto 172);
		ACK15_14: for i in 256 to 258 generate
			ki_15(i) <= ki7b(i);
		end generate ACK15_14;
		ki_15(261 downto 259) <= ko_7(138 downto 136);

		ki_15(262) <= ko_7(183);
		ki_15(269 downto 263) <= ko_7(211 downto 205);
		ki_15(271 downto 270) <= ko_7(203 downto 202);
		ACK15_15: for i in 272 to 274 generate
			ki_15(i) <= ki7b(i);
		end generate ACK15_15;
-------------------------------------------------------------------------------------------
   s3_4r(58 downto 56) <= temp15_out(274 downto 272);
   s3_4r(47 downto 46) <= temp15_out(271 downto 270);
   s3_4r(37 downto 31) <= temp15_out(269 downto 263);
   s3_4r(26) <= temp15_out(262);
   c3_3r(62 downto 60) <= temp15_out(261 downto 259);
   c3_3r(58 downto 52) <= temp15_out(258 downto 252);
   c3_3r(51 downto 46) <= temp15_out(251 downto 246);
   c3_3r(44 downto 39) <= temp15_out(245 downto 240);
   c3_3r(37 downto 26) <= temp15_out(239 downto 228);
   c3_3r(22) <= temp15_out(227);
   s3_3r(72 downto 22) <= temp15_out(226 downto 176);
   s3_3r(16) <= temp15_out(175);
   c3_2r(54 downto 49) <= temp15_out(174 downto 169);
   c3_2r(47 downto 16) <= temp15_out(168 downto 137);
   c3_2r(11) <= temp15_out(136);
   s3_2r(54 downto 11) <= temp15_out(135 downto 92);
   s3_2r(6) <= temp15_out(91);
   c3_1r(44 downto 5) <= temp15_out(90 downto 51);
   s3_1r(45 downto 5) <= temp15_out(50 downto 10);
```

```
    pos13(4 downto 0) <= temp15_out(9 downto 5);
    sign10i <= temp15_out(4 downto 3);
    add_sub10i <= temp15_out(2);
    mac_mpy10i <= temp15_out(1);
    MS10i <= temp15_out(0);

-- FIFTH LEVEL (add register)

    HA_POS5_2: half_add
       port map(s3_1r(5), c3_1r(5), c4_1(6), pos13(5));

    FA4_1_6: full_add
       port map(s3_1r(6), c3_1r(6), s3_2r(6), c4_1(7), s4_1(6));

    s4_2(7) <= c3_1r(7);
    s4_1(10 downto 7) <= s3_1r(10 downto 7);
    c4_1(10 downto 8) <= c3_1r(10 downto 8);

    WALLACE_TREE_1_F4: for i in 11 to 44 generate
          FULL_ADDERS: full_add
             port map(s3_1r(i), c3_1r(i), s3_2r(i), c4_1(i+1), s4_1(i));
    end generate WALLACE_TREE_1_F4;

    FA4_1_45: full_add
       port map(s3_1r(45), s3_2r(45), c3_2r(45), c4_1(46), s4_1(45));

    FA4_1_46: full_add
       port map(s3_2r(46), c3_2r(46), s3_3r(46), c4_1(47), s4_1(46));

    FA4_1_47: full_add
       port map(s3_2r(47), c3_2r(47), s3_3r(47), c4_1(48), s4_1(47));

    FA4_1_48: full_add
       port map(s3_2r(48), s3_3r(48), c3_3r(48), c4_1(49), s4_1(48));

    WALLACE_TREE_1_F4M: for i in 49 to 54 generate
          FULL_ADDERS: full_add
             port map(s3_2r(i), c3_2r(i), s3_3r(i), c4_1(i+1), s4_1(i));
    end generate WALLACE_TREE_1_F4M;

    s4_1(55) <= s3_3r(55);

    WALLACE_TREE_1_F4L: for i in 56 to 58 generate
          FULL_ADDERS: full_add
             port map(s3_3r(i), c3_3r(i), s3_4r(i), c4_1(i+1), s4_1(i));
    end generate WALLACE_TREE_1_F4L;

    s4_1(72 downto 59) <= s3_3r(72 downto 59);
    c4_1(62 downto 60) <= c3_3r(62 downto 60);


    c4_1(11) <= c3_2r(11);
    s4_2(21 downto 16) <= c3_2r(21 downto 16);
```

```
   c4_2(16) <= s3_3r(16);
   s4_3(23) <= s3_3r(23);
   s4_2(25 downto 23) <= c3_2r(25 downto 23);
   c4_2(25 downto 24) <= s3_3r(25 downto 24);

   FA4_2_22: full_add
      port map(c3_2r(22), s3_3r(22), c3_3r(22), c4_2(23), s4_2(22));

   WALLACE_TREE_2_F4: for i in 26 to 37 generate
         FULL_ADDERS: full_add
            port map(c3_2r(i), s3_3r(i), c3_3r(i), c4_2(i+1), s4_2(i));
   end generate WALLACE_TREE_2_F4;

   s4_3(38) <= s3_3r(38);
   s4_2(38) <= c3_2r(38);

   WALLACE_TREE_2_F4L: for i in 39 to 44 generate
         FULL_ADDERS: full_add
            port map(c3_2r(i), s3_3r(i), c3_3r(i), c4_2(i+1), s4_2(i));
   end generate WALLACE_TREE_2_F4L;

   s4_2(45) <= s3_3r(45);
   s4_2(47 downto 46) <= c3_3r(47 downto 46);
   s4_2(55 downto 49) <= c3_3r(55 downto 49);
   c4_2(26) <= s3_4r(26);
   s4_3(37 downto 31) <= s3_4r(37 downto 31);
   c4_2(47 downto 46) <= s3_4r(47 downto 46);

-- ADD REGISTER

   temp7 <= s4_3(38 downto 31) & s4_3(23) & c4_2(47 downto 40) & c4_2(38 downto 23) & c4_2(16) & s4_2(55
            downto 49) &
                     s4_2(47 downto 16) & s4_2(7) & c4_1(62 downto 57) & c4_1(55 downto 6) &
            s4_1(72 downto 6) & pos13 & sign10i & add_sub10i & mac_mpy10i & MS10i;

      temp7_in <= temp7;
--------------------------------------------------------------------------------
--                                                    3
--------------------------------------------------------------------------------
         ko_7b1(1 downto 0) <= ko_7(83) & ko_7(15);
      COMP7_1: comp
        generic map(2)
        port map(ko_7b1, ki7b(10));

      ko_7b2(1 downto 0) <= ko_7(84) & ko_7(16);
      COMP7_2: comp
        generic map(2)
        port map(ko_7b2, ki7b(11));

   ko_7_b3: for i in 16 to 49 generate
      ko_7b3( (1+ i*2) downto (0 + i*2) ) <= ko_7(i+73) & ko_7(i+5);-- (89-122) , (21-54)
      COMP7_3: comp
        generic map(2)
```

```
    port map(ko_7b3( (1+ i*2) downto (0 + i*2) ), ki7b(i) );
  end generate ko_7_b3;

  ko_7b4(1 downto 0) <= ko_7(123) & ko_7(55);
  COMP7_4: comp
    generic map(2)
    port map(ko_7b4, ki7b(50));

  ko_7b5(1 downto 0) <= ko_7(83) & ko_7(15);
  COMP7_5: comp
    generic map(2)
    port map(ko_7b5, ki7b(51));

  ko_7b6(1 downto 0) <= ko_7(84) & ko_7(16);
  COMP7_6: comp
    generic map(2)
    port map(ko_7b6, ki7b(52));

ko_7_b7: for i in 57 to 90 generate
   ko_7b7( (1+ i*2) downto (0 + i*2) ) <= ko_7(i+32) & ko_7(i-36);-- (89-122) , (21-54)
   COMP7_7: comp
     generic map(2)
     port map(ko_7b7( (1+ i*2) downto (0 + i*2) ), ki7b(i) );
  end generate ko_7_b7;

  ko_7b8(1 downto 0) <= ko_7(84) & ko_7(16);
  COMP7_8: comp
    generic map(2)
    port map(ko_7b8, ki7b(91));

ko_7_b9: for i in 92 to 125 generate
   ko_7b9( (1+ i*2) downto (0 + i*2) ) <= ko_7(i-3) & ko_7(i-71);-- (89-122) , (21-54)
   COMP7_9: comp
     generic map(2)
     port map(ko_7b9( (1+ i*2) downto (0 + i*2) ), ki7b(i) );
  end generate ko_7_b9;

  ko_7b10(1 downto 0) <= ko_7(123) & ko_7(55);
  COMP7_10: comp
    generic map(2)
    port map(ko_7b10, ki7b(126));

  ko_7b11(1 downto 0) <= ko_7(124) & ko_7(56);
  COMP7_11: comp
    generic map(2)
    port map(ko_7b11, ki7b(127));

  ko_7b12(1 downto 0) <= ko_7(125) & ko_7(57);
  COMP7_12: comp
    generic map(2)
    port map(ko_7b12, ki7b(128));

  ko_7b13(1 downto 0) <= ko_7(126) & ko_7(58);
```

```
  COMP7_13: comp
    generic map(2)
    port map(ko_7b13, ki7b(129));

ko_7_b14: for i in 130 to 135 generate
   ko_7b14( (1+ i*2) downto (0 + i*2) ) <= ko_7(i-3) & ko_7(i-71);-- (127-132) , (59-64)
   COMP7_14: comp
     generic map(2)
     port map(ko_7b14( (1+ i*2) downto (0 + i*2) ), ki7b(i) );
 end generate ko_7_b14;

  ko_7b15(1 downto 0) <= ko_7(180) & ko_7(146);
  COMP7_15: comp
    generic map(2)
    port map(ko_7b15, ki7b(143));

ko_7_b16: for i in 147 to 158 generate
   ko_7b16( (1+ i*2) downto (0 + i*2) ) <= ko_7(i+37) & ko_7(i+3);-- (184-195) , (150-161)
   COMP7_16: comp
     generic map(2)
     port map(ko_7b16( (1+ i*2) downto (0 + i*2) ), ki7b(i) );
 end generate ko_7_b16;

ko_7_b17: for i in 160 to 165 generate
   ko_7b17( (1+ i*2) downto (0 + i*2) ) <= ko_7(i+36) & ko_7(i+3);-- (196-201) , (163-168)
   COMP7_17: comp
     generic map(2)
     port map(ko_7b17( (1+ i*2) downto (0 + i*2) ), ki7b(i) );
 end generate ko_7_b17;

  ko_7b18(1 downto 0) <= ko_7(123) & ko_7(55);
  COMP7_18: comp
    generic map(2)
    port map(ko_7b18, ki7b(166));

  ko_7b19(1 downto 0) <= ko_7(124) & ko_7(56);
  COMP7_19: comp
    generic map(2)
    port map(ko_7b19, ki7b(167));

  ko_7b20(1 downto 0) <= ko_7(125) & ko_7(57);
  COMP7_20: comp
    generic map(2)
    port map(ko_7b20, ki7b(168));

ko_7_b21: for i in 169 to 174 generate
   ko_7b21( (1+ i*2) downto (0 + i*2) ) <= ko_7(i-42) & ko_7(i-110);-- (127-132) , (59-64)
   COMP7_21: comp
     generic map(2)
     port map(ko_7b21( (1+ i*2) downto (0 + i*2) ), ki7b(i) );
 end generate ko_7_b21;

  ko_7b22(1 downto 0) <= ko_7(180) & ko_7(146);
```

```
  COMP7_22: comp
    generic map(2)
    port map(ko_7b22, ki7b(176));

ko_7_b23: for i in 180 to 191 generate
   ko_7b23( (1+ i*2) downto (0 + i*2) ) <= ko_7(i+4) & ko_7(i-30);-- (184-195) , (150-161)
   COMP7_23: comp
     generic map(2)
     port map(ko_7b23( (1+ i*2) downto (0 + i*2) ), ki7b(i) );
 end generate ko_7_b23;

ko_7_b24: for i in 193 to 198 generate
   ko_7b24( (1+ i*2) downto (0 + i*2) ) <= ko_7(i+3) & ko_7(i-30);-- (196-201) , (163-168)
   COMP7_24: comp
     generic map(2)
     port map(ko_7b24( (1+ i*2) downto (0 + i*2) ), ki7b(i) );
 end generate ko_7_b24;

  ko_7b25(1 downto 0) <= ko_7(124) & ko_7(56);
  COMP7_25: comp
    generic map(2)
    port map(ko_7b25, ki7b(200));

  ko_7b26(1 downto 0) <= ko_7(125) & ko_7(57);
  COMP7_26: comp
    generic map(2)
    port map(ko_7b26, ki7b(201));

  ko_7b27(1 downto 0) <= ko_7(126) & ko_7(58);
  COMP7_27: comp
    generic map(2)
    port map(ko_7b27, ki7b(202));

ko_7_b28: for i in 203 to 208 generate
   ko_7b28( (1+ i*2) downto (0 + i*2) ) <= ko_7(i-76) & ko_7(i-144);-- (127-132) , (59-64)
   COMP7_28: comp
     generic map(2)
     port map(ko_7b28( (1+ i*2) downto (0 + i*2) ), ki7b(i) );
 end generate ko_7_b28;

ko_7_b29: for i in 210 to 212 generate
   ko_7b29( (1+ i*2) downto (0 + i*2) ) <= ko_7(i-77) & ko_7(i-144);-- (133-135) , (66-68)
   COMP7_29: comp
     generic map(2)
     port map(ko_7b29( (1+ i*2) downto (0 + i*2) ), ki7b(i) );
 end generate ko_7_b29;

  ko_7b30(1 downto 0) <= ko_7(180) & ko_7(146);
  COMP7_30: comp
    generic map(2)
    port map(ko_7b30, ki7b(227));

ko_7_b31: for i in 228 to 239 generate
```

```
      ko_7b31( (1+ i*2) downto (0 + i*2) ) <= ko_7(i-44) & ko_7(i-78);-- (184-195) , (150-161)
      COMP7_31: comp
        generic map(2)
        port map(ko_7b31( (1+ i*2) downto (0 + i*2) ), ki7b(i) );
    end generate ko_7_b31;

  ko_7_b32: for i in 240 to 245 generate
      ko_7b32( (1+ i*2) downto (0 + i*2) ) <= ko_7(i-44) & ko_7(i-77);-- (196-201) , (163-168)
      COMP7_32: comp
        generic map(2)
        port map(ko_7b32( (1+ i*2) downto (0 + i*2) ), ki7b(i) );
    end generate ko_7_b32;

     ko_7b33(1 downto 0) <= ko_7(126) & ko_7(58);
     COMP7_33: comp
       generic map(2)
       port map(ko_7b33, ki7b(248));

  ko_7_b34: for i in 256 to 258 generate
      ko_7b34( (1+ i*2) downto (0 + i*2) ) <= ko_7(i-123) & ko_7(i-190);-- (133-135) , (66-68)
      COMP7_34: comp
        generic map(2)
        port map(ko_7b34( (1+ i*2) downto (0 + i*2) ), ki7b(i) );
    end generate ko_7_b34;

  ko_7_b35: for i in 272 to 274 generate
      ko_7b35( (1+ i*2) downto (0 + i*2) ) <= ko_7(i-139) & ko_7(i-206);-- (133-135) , (66-68)
      COMP7_35: comp
        generic map(2)
        port map(ko_7b35( (1+ i*2) downto (0 + i*2) ), ki7b(i) );
    end generate ko_7_b35;
--------------------------------------------------------------------------------

  REG7: ncl_register_D
     generic map(208, -4)
     port map(temp7_in, ki_7, reset, temp7_out, ko_7);
--------------------------------------------------------------------------------
--                                                    4
--------------------------------------------------------------------------------
          ki_7(5) <= ko_6(5);
          ki_7(6) <= ko_6(6);
          ki_7(7) <= ko_6(7);
          ki_7(9 downto 8) <= ko_6(9 downto 8);
          ki_7(15 downto 10) <= ko_6(15 downto 10);

          ki_7(16) <= ki6b(16);
          ki_7(17) <= ki6b(17);
          ki_7(25 downto 18) <= ko_6(25 downto 18);
          ACK7_1: for i in 26 to 57 generate
                    ki_7(i) <= ki6b(i);
          end generate ACK7_1;
          ki_7(58) <= ko_6(58);
          ACK7_2: for i in 59 to 65 generate
```

```
                ki_7(i) <= ki6b(i);
        end generate ACK7_2;
        ki_7(82 downto 66) <= ko_6(82 downto 66);

        ki_7(83) <= ki6b(83);
        ki_7(84) <= ki6b(84);
        ki_7(85) <= ko_6(138);
        ki_7(92 downto 86) <= ko_6(91 downto 85);
        ACK7_3: for i in 93 to 124 generate
                ki_7(i) <= ki6b(i);
        end generate ACK7_3;
        ki_7(125) <= ko_6(163);
        ACK7_4: for i in 126 to 132 generate
                ki_7(i) <= ki6b(i);
        end generate ACK7_4;
        ki_7(138 downto 133) <= ko_6(137 downto 132);

        ki_7(139) <= ki6b(139);
        ACK7_5: for i in 140 to 171 generate
                ki_7(i) <= ki6b(i);
        end generate ACK7_5;
        ACK7_6: for i in 172 to 178 generate
                ki_7(i) <= ki6b(i);
        end generate ACK7_6;

        ki_7(179) <= ko_6(92);
        ki_7(195 downto 180) <= ko_6(154 downto 139);
        ki_7(203 downto 196) <= ko_6(162 downto 155);

        ki_7(204) <= ko_6(164);
        ki_7(212 downto 205) <= ko_6(172 downto 165);
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
   s4_3r(38 downto 31) <= temp7_out(212 downto 205);
   s4_3r(23) <= temp7_out(204);
   c4_2r(47 downto 40) <= temp7_out(203 downto 196);
   c4_2r(38 downto 23) <= temp7_out(195 downto 180);
   c4_2r(16) <= temp7_out(179);
   s4_2r(55 downto 49) <= temp7_out(178 downto 172);
   s4_2r(47 downto 16) <= temp7_out(171 downto 140);
   s4_2r(7) <= temp7_out(139);
   c4_1r(62 downto 57) <= temp7_out(138 downto 133);
   c4_1r(55 downto 6) <= temp7_out(132 downto 83);
   s4_1r <= temp7_out(82 downto 16);
   pos2i(5 downto 0) <= temp7_out(15 downto 10);
   sign2i <= temp7_out(9 downto 8);
   add_sub2i <= temp7_out(7);
   mac_mpy2i <= temp7_out(6);
   MS1i <= temp7_out(5);


-- SIXTH LEVEL (add register)
```

```
   HA_POS6_2: half_add
      port map(s4_1r(6), c4_1r(6), c5_1(7), pos2i(6));

   FA5_1_7: full_add
      port map(s4_1r(7), c4_1r(7), s4_2r(7), c5_1(8), s5_1(7));

   s5_1(15 downto 8) <= s4_1r(15 downto 8);
   c5_1(15 downto 9) <= c4_1r(15 downto 9);
   c5_1(16) <= c4_2r(16);
   s5_2(8) <= c4_1r(8);

   WALLACE_TREE_1_F5: for i in 16 to 47 generate
         FULL_ADDERS: full_add
            port map(s4_1r(i), c4_1r(i), s4_2r(i), c5_1(i+1), s5_1(i));
   end generate WALLACE_TREE_1_F5;

   WALLACE_TREE_1_F5L: for i in 49 to 55 generate
         FULL_ADDERS: full_add
            port map(s4_1r(i), c4_1r(i), s4_2r(i), c5_1(i+1), s5_1(i));
   end generate WALLACE_TREE_1_F5L;

   s5_1(48) <= s4_1r(48);
   s5_2(48) <= c4_1r(48);
   s5_1(72 downto 56) <= s4_1r(72 downto 56);
   c5_1(62 downto 57) <= c4_1r(62 downto 57);
   s5_2(47 downto 23) <= c4_2r(47 downto 23);
   c5_2(23) <= s4_3r(23);
   c5_2(38 downto 31) <= s4_3r(38 downto 31);

-- ADD REGISTER

   temp6 <= c5_2(38 downto 31) & c5_2(23) & s5_2(48 downto 40) & s5_2(38 downto 23) & s5_2(8) & c5_1(62
         downto 50) & c5_1(48 downto 7)
                  & s5_1(72 downto 7) & pos2i & sign2i & add_sub2i & mac_mpy2i & MS1i;

    temp6_in <= temp6;

   ko_6b1(1 downto 0) <= ko_6(83) & ko_6(16);
   COMP6_1: comp
    generic map(2)
    port map(ko_6b1, ki6b(16));

   ko_6b2(1 downto 0) <= ko_6(84) & ko_6(17);
   COMP6_2: comp
    generic map(2)
    port map(ko_6b2, ki6b(17));

   ko_6_b3: for i in 26 to 57 generate
      ko_6b3( (1+ i*2) downto (0 + i*2) ) <= ko_6(i+67) & ko_6(i);-- (93-124) , (26-57)
      COMP6_3: comp
       generic map(2)
       port map(ko_6b3( (1+ i*2) downto (0 + i*2) ), ki6b(i) );
     end generate ko_6_b3;
```

```
ko_6_b4: for i in 59 to 65 generate
   ko_6b4( (1+ i*2) downto (0 + i*2) ) <= ko_6(i+66) & ko_6(i);-- (125-131) , (59-65)
   COMP6_4: comp
    generic map(2)
    port map(ko_6b4( (1+ i*2) downto (0 + i*2) ), ki6b(i) );
 end generate ko_6_b4;


  ko_6b5(1 downto 0) <= ko_6(83) & ko_6(16);
  COMP6_5: comp
   generic map(2)
   port map(ko_6b5, ki6b(83));

  ko_6b6(1 downto 0) <= ko_6(84) & ko_6(17);
  COMP6_6: comp
   generic map(2)
   port map(ko_6b6, ki6b(84));

ko_6_b7: for i in 93 to 124 generate
   ko_6b7( (1+ i*2) downto (0 + i*2) ) <= ko_6(i) & ko_6(i-67);-- (93-124) , (26-57)
   COMP6_7: comp
    generic map(2)
    port map(ko_6b7( (1+ i*2) downto (0 + i*2) ), ki6b(i) );
 end generate ko_6_b7;

ko_6_b8: for i in 126 to 132 generate
   ko_6b8( (1+ i*2) downto (0 + i*2) ) <= ko_6(i-1) & ko_6(i-67);-- (125-131) , (59-65)
   COMP6_8: comp
    generic map(2)
    port map(ko_6b8( (1+ i*2) downto (0 + i*2) ), ki6b(i) );
 end generate ko_6_b8;

  ko_6b9(1 downto 0) <= ko_6(84) & ko_6(17);
  COMP6_9: comp
   generic map(2)
   port map(ko_6b9, ki6b(139));

ko_6_b10: for i in 140 to 171 generate
   ko_6b10( (1+ i*2) downto (0 + i*2) ) <= ko_6(i-47) & ko_6(i-114);-- (93-124) , (26-57)
   COMP6_10: comp
    generic map(2)
    port map(ko_6b10( (1+ i*2) downto (0 + i*2) ), ki6b(i) );
 end generate ko_6_b10;

ko_6_b11: for i in 172 to 178 generate
   ko_6b11( (1+ i*2) downto (0 + i*2) ) <= ko_6(i-47) & ko_6(i-113);-- (125-131) , (59-65)
   COMP6_11: comp
    generic map(2)
    port map(ko_6b11( (1+ i*2) downto (0 + i*2) ), ki6b(i) );
 end generate ko_6_b11;

REG6: ncl_register_D
```

```
    generic map(168, -4)
    port map(temp6_in, ki_6, reset, temp6_out, ko_6);


        ki_6(5) <= ko_12(5);
        ki_6(6) <= ko_12(6);
        ki_6(7) <= ko_12(7);
        ki_6(9 downto 8) <= ko_12(9 downto 8);
        ki_6(16 downto 10) <= ko_12(16 downto 10);

        ki_6(17) <= ki12b(17);
        ki_6(18) <= ki12b(18);
        ki_6(32 downto 19) <= ko_12(32 downto 19);
        ACK6_1: for i in 33 to 48 generate
                ki_6(i) <= ki12b(i);
        end generate ACK6_1;
        ki_6(49) <= ko_12(49);
        ACK6_2: for i in 50 to 58 generate
                ki_6(i) <= ki12b(i);
        end generate ACK6_2;
        ki_6(82 downto 59) <= ko_12(82 downto 59);

        ki_6(83) <= ki12b(83);
        ki_6(84) <= ki12b(84);
        ki_6(85) <= ko_12(137);
        ki_6(98 downto 86) <= ko_12(97 downto 85);
        ACK6_3: for i in 99 to 114 generate
                ki_6(i) <= ki12b(i);
        end generate ACK6_3;
        ki_6(115) <= ko_12(146);
        ACK6_4: for i in 116 to 124 generate
                ki_6(i) <= ki12b(i);
        end generate ACK6_4;
        ki_6(137 downto 125) <= ko_12(136 downto 124);

        ki_6(138) <= ki12b(138);
        ACK6_5: for i in 139 to 154 generate
                ki_6(i) <= ki12b(i);
        end generate ACK6_5;
        ACK6_6: for i in 155 to 163 generate
                ki_6(i) <= ki12b(i);
        end generate ACK6_6;

        ki_6(164) <= ko_12(98);
        ki_6(172 downto 165) <= ko_12(145 downto 138);
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

   c5_2r(38 downto 31) <= temp6_out(172 downto 165);
   c5_2r(23) <= temp6_out(164);
   s5_2r(48 downto 40) <= temp6_out(163 downto 155);
   s5_2r(38 downto 23) <= temp6_out(154 downto 139);
   s5_2r(8) <= temp6_out(138);
```

```
   c5_1r(62 downto 50) <= temp6_out(137 downto 125);
   c5_1r(48 downto 7) <= temp6_out(124 downto 83);
   s5_1r <= temp6_out(82 downto 17);
   pos3(6 downto 0) <= temp6_out(16 downto 10);
   sign2oo <= temp6_out(9 downto 8);
   add_sub2oo <= temp6_out(7);
   mac_mpy2oo <= temp6_out(6);
   MSoo <= temp6_out(5);

-- SEVENTH LEVEL (add register)

   HA_POS7_2: half_add
      port map(s5_1r(7), c5_1r(7), c6_1(8), pos3(7));

   FA6_1_8: full_add
      port map(s5_1r(8), c5_1r(8), s5_2r(8), c6_1(9), s6_1(8));

   s6_1(22 downto 9) <= s5_1r(22 downto 9);
   c6_1(22 downto 10) <= c5_1r(22 downto 10);
   s6_2(9) <= c5_1r(9);
   c6_1(23) <= c5_2r(23);

   WALLACE_TREE_1_F6: for i in 23 to 38 generate
         FULL_ADDERS: full_add
            port map(s5_1r(i), c5_1r(i), s5_2r(i), c6_1(i+1), s6_1(i));
   end generate WALLACE_TREE_1_F6;

   WALLACE_TREE_1_F6L: for i in 40 to 48 generate
         FULL_ADDERS: full_add
            port map(s5_1r(i), c5_1r(i), s5_2r(i), c6_1(i+1), s6_1(i));
   end generate WALLACE_TREE_1_F6L;

   s6_1(39) <= s5_1r(39);
   s6_2(39) <= c5_1r(39);
   s6_1(72 downto 49) <= s5_1r(72 downto 49);
   c6_1(62 downto 50) <= c5_1r(62 downto 50);
   s6_2(38 downto 31) <= c5_2r(38 downto 31);

--ADD REG HERE

   temp12 <= s6_2(39 downto 31) & s6_2(9) & c6_1(62 downto 41) & c6_1(39 downto 8) & s6_1 & pos3 &
                     sign2oo & add_sub2oo & mac_mpy2oo & MSoo;

      temp12_in <= temp12;

     ko_12b1(1 downto 0) <= ko_12(83) & ko_12(17);
     COMP12_1: comp
      generic map(2)
      port map(ko_12b1, ki12b(17));

     ko_12b2(1 downto 0) <= ko_12(84) & ko_12(18);
     COMP12_2: comp
      generic map(2)
```

```
    port map(ko_12b2, ki12b(18));

ko_12_b3: for i in 33 to 48 generate
   ko_12b3( (1+ i*2) downto (0 + i*2) ) <= ko_12(i+66) & ko_12(i);-- (99-114) , (33-48)
   COMP12_3: comp
    generic map(2)
    port map(ko_12b3( (1+ i*2) downto (0 + i*2) ), ki12b(i) );
 end generate ko_12_b3;

ko_12_b4: for i in 50 to 58 generate
   ko_12b4( (1+ i*2) downto (0 + i*2) ) <= ko_12(i+65) & ko_12(i);-- (115-123) , (50-58)
   COMP12_4: comp
    generic map(2)
    port map(ko_12b4( (1+ i*2) downto (0 + i*2) ), ki12b(i) );
 end generate ko_12_b4;

  ko_12b5(1 downto 0) <= ko_12(83) & ko_12(17);
  COMP12_5: comp
   generic map(2)
   port map(ko_12b5, ki12b(83));

  ko_12b6(1 downto 0) <= ko_12(84) & ko_12(18);
  COMP12_6: comp
   generic map(2)
   port map(ko_12b6, ki12b(84));

ko_12_b7: for i in 99 to 114 generate
   ko_12b7( (1+ i*2) downto (0 + i*2) ) <= ko_12(i) & ko_12(i-66);-- (99-114) , (33-48)
   COMP12_7: comp
    generic map(2)
    port map(ko_12b7( (1+ i*2) downto (0 + i*2) ), ki12b(i) );
 end generate ko_12_b7;

ko_12_b8: for i in 116 to 124 generate
   ko_12b8( (1+ i*2) downto (0 + i*2) ) <= ko_12(i-1) & ko_12(i-66);-- (115-123) , (50-58)
   COMP12_8: comp
    generic map(2)
    port map(ko_12b8( (1+ i*2) downto (0 + i*2) ), ki12b(i) );
 end generate ko_12_b8;

  ko_12b9(1 downto 0) <= ko_12(84) & ko_12(18);
  COMP12_9: comp
   generic map(2)
   port map(ko_12b9, ki12b(138));

ko_12_b10: for i in 139 to 154 generate
   ko_12b10( (1+ i*2) downto (0 + i*2) ) <= ko_12(i-40) & ko_12(i-106);-- (99-114) , (33-48)
   COMP12_10: comp
    generic map(2)
    port map(ko_12b10( (1+ i*2) downto (0 + i*2) ), ki12b(i) );
 end generate ko_12_b10;

ko_12_b11: for i in 155 to 163 generate
```

```
      ko_12b11( (1+ i*2) downto (0 + i*2) ) <= ko_12(i-40) & ko_12(i-105);-- (115-123) , (50-58)
      COMP12_11: comp
       generic map(2)
       port map(ko_12b11( (1+ i*2) downto (0 + i*2) ), ki12b(i) );
    end generate ko_12_b11;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
  REG12: ncl_register_D
    generic map(142, -4)
    port map(temp12_in, ki_12, reset, temp12_out, ko_12);
-----------------------------------------------------------------------------------------
--                                                  6
-----------------------------------------------------------------------------------------
            ki_12(5) <= ko_8(0);
            ki_12(6) <= ko_8(1);
            ki_12(7) <= ki8bb(0);--ko_8(2);
            --ki_12(9 downto 8) <= ko_8(4 downto 3);
            ki_12(8) <= ki8bb(1);
            ki_12(9) <= ki8bb(2);

            ki_12(17 downto 10) <= ko_8(66 downto 59);

            ki_12(18) <= ki8b(18);
            ki_12(19) <= ki8b(19);
            ACK12_1: for i in 20 to 49 generate
                        ki_12(i) <= ki8b(i);
            end generate ACK12_1;
            ki_12(82 downto 50) <= ko_8(131 downto 99);

            ki_12(83) <= ki8b(83);
            ki_12(84) <= ki8b(84);
            ACK12_2: for i in 85 to 114 generate
                        ki_12(i) <= ki8b(i);
            end generate ACK12_2;
            ki_12(136 downto 115) <= ko_8(58 downto 37);

            ki_12(137) <= ki8b(137);
            ACK12_3: for i in 138 to 146 generate
                        ki_12(i) <= ki8b(i);
            end generate ACK12_3;
-----------------------------------------------------------------------------------------
  s6_2r(39 downto 31) <= temp12_out(146 downto 138);
  s6_2r(9) <= temp12_out(137);
  c6_1r(62 downto 41) <= temp12_out(136 downto 115);
  c6_1r(39 downto 8) <= temp12_out(114 downto 83);
  s6_1r <= temp12_out(82 downto 18);
  s7_1(7 downto 0) <= temp12_out(17 downto 10);
  sign2o <= temp12_out(9 downto 8);
  add_sub2o <= temp12_out(7);
  mac_mpy2o <= temp12_out(6);
  MSo <= temp12_out(5);
```

```
-- EIGHTH LEVEL (add register)

   HA_POS8_2: half_add
      port map(s6_1r(8), c6_1r(8), c7_1(9), s7_1(8));

   FA7_1_9: full_add
      port map(s6_1r(9), c6_1r(9), s6_2r(9), c7_1(10), s7_1(9));

   WALLACE_TREE_1_H7: for i in 10 to 30 generate
         HALF_ADDERS: half_add
            port map(s6_1r(i), c6_1r(i), c7_1(i+1), s7_1(i));
   end generate WALLACE_TREE_1_H7;

   WALLACE_TREE_2_F7: for i in 31 to 39 generate
         FULL_ADDERS: full_add
            port map(s6_1r(i), c6_1r(i), s6_2r(i), c7_1(i+1), s7_1(i));
   end generate WALLACE_TREE_2_F7;

   s7_1(72 downto 40) <= s6_1r(72 downto 40);
   c7_1(62 downto 41) <= c6_1r(62 downto 41);


   temp8 <= sign2o & add_sub2o & sign2o & add_sub2o & s7_1 & c7_1 & sign2o & add_sub2o & mac_mpy2o &
            MSo;

   temp8_in <= temp8;
-------------------------------------------------------------------------
--                                                      6
-------------------------------------------------------------------------
      ko_8b1(1 downto 0) <= ko_8(5) & ko_8(67);
      COMP8_1: comp
        generic map(2)
        port map(ko_8b1, ki8b(18));

      ko_8b2(1 downto 0) <= ko_8(6) & ko_8(68);
      COMP8_2: comp
        generic map(2)
        port map(ko_8b2, ki8b(19));

   ko_8_b3: for i in 20 to 49 generate
      ko_8b3( (1+ i*2) downto (0 + i*2) ) <= ko_8(i-13) & ko_8(i+49);-- (7-36) , (69-98)
      COMP8_3: comp
        generic map(2)
        port map(ko_8b3( (1+ i*2) downto (0 + i*2) ), ki8b(i) );
    end generate ko_8_b3;

      ko_8b4(1 downto 0) <= ko_8(5) & ko_8(67);
      COMP8_4: comp
        generic map(2)
        port map(ko_8b4, ki8b(83));

      ko_8b5(1 downto 0) <= ko_8(6) & ko_8(68);
      COMP8_5: comp
```

```
      generic map(2)
      port map(ko_8b5, ki8b(84));

   ko_8_b6: for i in 85 to 114 generate
      ko_8b6( (1+ i*2) downto (0 + i*2) ) <= ko_8(i-78) & ko_8(i-16);-- (7-36) , (69-98)
      COMP8_6: comp
       generic map(2)
       port map(ko_8b6( (1+ i*2) downto (0 + i*2) ), ki8b(i) );
    end generate ko_8_b6;

    ko_8b7(1 downto 0) <= ko_8(6) & ko_8(68);
    COMP8_7: comp
     generic map(2)
     port map(ko_8b7, ki8b(137));

   ko_8_b8: for i in 138 to 146 generate
      ko_8b8( (1+ i*2) downto (0 + i*2) ) <= ko_8(i-110) & ko_8(i-48);-- (28-36) , (90-98)
      COMP8_8: comp
       generic map(2)
       port map(ko_8b8( (1+ i*2) downto (0 + i*2) ), ki8b(i) );
    end generate ko_8_b8;

    ko_8b9(2 downto 0) <= ko_8(135) & ko_8(132) & ko_8(2);
    COMP8_9: comp
     generic map(3)
     port map(ko_8b9, ki8bb(0));

    ko_8b10(2 downto 0) <= ko_8(136) & ko_8(133) & ko_8(3);
    COMP8_10: comp
     generic map(3)
     port map(ko_8b10, ki8bb(1));

    ko_8b11(2 downto 0) <= ko_8(137) & ko_8(134) & ko_8(4);
    COMP8_11: comp
     generic map(3)
     port map(ko_8b11, ki8bb(2));

   REG8: ncl_register_D
      generic map(138, -4)
      port map(temp8_in, ki_8, reset, temp8_out, ko_8);--ki_8=ko_8c
 ------------------------------------------------------------------------
    ki_8 <= ko_8a;
    temp8a_in <= temp8_out;

   FF_feedforward_SHIFT_ncr: feedforward_SHIFT_ncr
        port map(temp8a_in, reset, ki_14, ko_8a, temp14_out);
---------------------------------------------------------------------------
   ki_14(0) <= ki14_2;--ko_2(7);
   ki_14(1) <= ki14;--ko_2(8);
   ki_14(2) <= ki14_2;--ko_2(9);

            ACK14_1: for i in 3 to 5 generate
                     ki_14(i) <= ki14b(i);
```

```
            end generate ACK14_1;
   ki_14(7 downto 6) <= ko_2(11 downto 10);

            ACK14_2: for i in 8 to 70 generate
                  ki_14(i) <= ki14b(i);
            end generate ACK14_2;

   ki_14(71) <= ki14_2;--ko_2(146);
   ki_14(79 downto 72) <= ko_2(82 downto 75);

            ACK14_3: for i in 80 to 142 generate
                  ki_14(i) <= ki14b(i);
            end generate ACK14_3;

   ki_14(143) <= ki14_2;--ko_2(146);
-------------------------------------------------------------------------
   s8_1r(71 downto 0) <= temp14_out(143 downto 72);
   c8_1r(71 downto 8) <= temp14_out(71 downto 8);
   c8_1r(1 downto 0) <= temp14_out(7 downto 6);
   s8_2r(10 downto 8) <= temp14_out(5 downto 3);
   mac_mpy2o14 <= temp14_out(1);
-------------------------------------------------------------------------
   c9_1(1 downto 0) <= c8_1r(1 downto 0);
   s9_1(7 downto 0) <= s8_1r(7 downto 0);

   WALLACE_TREE_F8_8_10: for i in 8 to 10 generate
         FULL_ADDERS: full_add
            port map(s8_1r(i), c8_1r(i), s8_2r(i), c9_1(i+1), s9_1(i));
   end generate WALLACE_TREE_F8_8_10;

   WALLACE_TREE_H8_11_70: for i in 11 to 70 generate
         HALF_ADDERS: half_add
            port map(s8_1r(i), c8_1r(i), c9_1(i+1), s9_1(i));
   end generate WALLACE_TREE_H8_11_70;

   HA8_71a: half_add
      port map(s8_1r(71), c8_1r(71), c9_1_72 , s9_1(71));

   HA8_71b: half_add
      port map(s8_1r(71) , c8_1r(71), open, s9_1_72);

   temp2(163 downto 7) <= mac_mpy2o14 & mac_mpy2o14 & mac_mpy2o14 & mac_mpy2o14 & mac_mpy2o14
            & mac_mpy2o14 & mac_mpy2o14 & mac_mpy2o14 & mac_mpy2o14 & mac_mpy2o14 & mac_mpy2o14
            & mac_mpy2o14 & mac_mpy2o14 & mac_mpy2o14 & mac_mpy2o14 & mac_mpy2o14 & mac_mpy2o14
            & s9_1 & c9_1(71 downto 9) & c9_1(1 downto 0) & c9_1_72 & mac_mpy2o14 & s9_1_72;
-------------------------------------------------------------------------
--                                      7
-------------------------------------------------------------------------
   ko_2_b1: for i in 3 to 5 generate
      ko_2b1( (1+ i*2) downto (0 + i*2) ) <= ko_2(i+9) & ko_2(i+80);-- (12-14) , (83-85)
      COMP14_1: comp
       generic map(2)
       port map(ko_2b1( (1+ i*2) downto (0 + i*2) ), ki14b(i) );
```

```
    end generate ko_2_b1;

  ko_2_b2: for i in 8 to 70 generate
     ko_2b2( (1+ i*2) downto (0 + i*2) ) <= ko_2(i+4) & ko_2(i+75);-- (12-74) , (83-145)
     COMP14_2: comp
      generic map(2)
      port map(ko_2b2( (1+ i*2) downto (0 + i*2) ), ki14b(i) );
    end generate ko_2_b2;

  ko_2_b3: for i in 80 to 142 generate
     ko_2b3( (1+ i*2) downto (0 + i*2) ) <= ko_2(i-68) & ko_2(i+3);-- (12-74) , (83-145)
     COMP14_3: comp
      generic map(2)
      port map(ko_2b3( (1+ i*2) downto (0 + i*2) ), ki14b(i) );
    end generate ko_2_b3;
    ko_2b4(17 downto 0) <= ko_2(163 downto 147) & ko_2(8); -- 3 downto 0 -- 149 downto 147
     COMP14_4: comp
      generic map(18)
      port map(ko_2b4(17 downto 0), ki14 ); -- 3 downto 0
    ko_2b5(2 downto 0)   <= ko_2(146) & ko_2(9) & ko_2(7);
     COMP14_5: comp
      generic map(3)
      port map(ko_2b5(2 downto 0), ki14_2 );

   REG2_0: ncl_register_D
     generic map(157, -4)
     port map(temp2(163 downto 7), kof(163 downto 7), reset, temp2_out(163 downto 7), ko_2(163 downto 7));

     kof(163 downto 7) <= ki;
     PPC_out(156 downto 0) <= temp2_out(163 downto 7);
end TEST;
use work.dual_rail.all;
use work.ncl_signals.all;
library ieee;
use ieee.std_logic_1164.all;

entity feed_back_RNULL is
          port(temp2: in dual_rail_logic_VECTOR(288 downto 7);
                    reset : in std_logic;
                    ki: in std_logic_vector(152 downto 8);
                    ko: out std_logic_vector(288 downto 7);
                    temp9_out: out dual_rail_logic_VECTOR(152 downto 8));
end;

architecture SYN of feed_back_RNULL is

          signal x1, y1: dual_rail_logic_VECTOR(31 downto 0);
          signal add_sub2o4, mac_mpy2o4, MS1o4, add_sub1, mac_mpy1, OV1: dual_rail_logic;
     signal ki9_1, ki9_2, add_sub1t, mac_mpy1t: dual_rail_logic;
     signal add_sub2, mac_mpy2: dual_rail_logic;
     signal add_sub2i, mac_mpy2i: dual_rail_logic;
     signal add_sub2o, mac_mpy2o, add_sub2o8, mac_mpy2o8, MSo8, add_sub2o14, mac_mpy2o14, MSo14:
          dual_rail_logic;
```

```
signal add_sub2oo, mac_mpy2oo: dual_rail_logic;
signal add_sub3, mac_mpy3: dual_rail_logic;
signal add_sub10, mac_mpy10, MS10: dual_rail_logic;
    signal add_sub10i, mac_mpy10i, MS10i: dual_rail_logic;
    signal ko_9_2, sign1t, sign2, sign2i, sign2o, sign2o4, sign2o8: dual_rail_logic_VECTOR(1 downto 0);
signal sign10, sign10i, sign3, sign3o, sign2oo: dual_rail_logic_VECTOR(1 downto 0);
    signal AS3o, add_sub3o, mac_mpy3o, round3o, rnd_type3o, saturate3o, MS3o: dual_rail_logic;

    signal temp1, temp1_in, temp1_out : dual_rail_logic_VECTOR(67 downto 0);
    signal ki_1, ko_1: std_logic_vector(67 downto 0);

    signal temp2_in, temp2_out: dual_rail_logic_VECTOR(288 downto 7);
    signal kof, ki_2, ko_2, ko_2B: std_logic_vector(288 downto 7);
    signal ki15: std_logic;

    signal A1_i1 : dual_rail_logic_VECTOR(71 downto 0);
    signal A2_i1 : dual_rail_logic_VECTOR(71 downto 2);
signal A1_o1: dual_rail_logic_VECTOR(71 downto 0);
signal A2_o1: dual_rail_logic_VECTOR(71 downto 2);
    signal ds9_1r, dc9_1r: dual_rail_logic_VECTOR(71 downto 0);
    signal dadd_sub1, dmac_mpy1, dMS: dual_rail_logic;

    signal temp15, temp15_in, temp15_out: dual_rail_logic_VECTOR(288 downto 7);
    signal ki_15, ko_15: std_logic_vector(288 downto 7);
    signal ki16: std_logic;
signal A1_o: dual_rail_logic_VECTOR(71 downto 0);
signal A2_o: dual_rail_logic_VECTOR(71 downto 2);
    signal d2add_sub1, d2mac_mpy1, d2MS: dual_rail_logic;

    signal temp16, temp16_in, temp16_out: dual_rail_logic_VECTOR(222 downto 7);

    signal ki_16, ko_16: std_logic_vector(222 downto 7);
signal A2_o_a: dual_rail_logic_VECTOR(71 downto 2);
    signal s10_1r2, c10_1r2: dual_rail_logic_VECTOR(71 downto 2);

    signal ko_16b1: std_logic_vector(1 downto 0);
    signal ko_16b2: std_logic_vector(1 downto 0);
    signal ko_16b3: std_logic_vector(13 downto 0);
    signal ko_16b4: std_logic_vector(123 downto 0);
    signal ki16b2: std_logic;
    signal ki16b3: std_logic_vector(6 downto 0);
    signal ki16b4: std_logic_vector(61 downto 0);
signal temp3, temp3_in, temp3_out: dual_rail_logic_VECTOR(72 downto 0);
    signal ki_3, ko_3, ki_f, ko_3A: std_logic_vector(72 downto 0);
    signal temp4, temp4_in, temp4_out: dual_rail_logic_VECTOR(729 downto 0);
    signal ki_4, ko_4: std_logic_vector(729 downto 0);
signal temp5, temp5_in, temp5_out: dual_rail_logic_VECTOR(110 downto 0);
    signal ki_5, ko_5: std_logic_vector(110 downto 0);
signal temp6, temp6_in, temp6_out: dual_rail_logic_VECTOR(172 downto 5);
    signal ki_6, ko_6: std_logic_vector(172 downto 5);
signal temp7, temp7_in, temp7_out: dual_rail_logic_VECTOR(212 downto 5);
    signal ki_7, ko_7: std_logic_vector(212 downto 5);
signal temp8, temp8_in, temp8_out: dual_rail_logic_VECTOR(131 downto 0);
```

```
	signal ki_8, ko_8: std_logic_vector(131 downto 0);
signal temp9, temp9_in: dual_rail_logic_VECTOR(152 downto 8);
	signal ki_9, ko_9: std_logic_vector(152 downto 8);

	signal ko_9b1: std_logic_vector(1 downto 0);
	signal ko_9b2: std_logic_vector(137 downto 0);
	signal ki9b2: std_logic_vector(68 downto 0);
	signal ko_2a: std_logic_vector(142 downto 0);

	signal pp, pp4: dual_rail_logic_VECTOR(1023 downto 0);
	signal ki1, ki2, ki3, ki4, ki5, ki6, ki7, ki8, ki9, ki10, ki11, ki12, ki13, ki14: std_logic;
signal pos2: dual_rail_logic_VECTOR(2 downto 0);
	signal pos2o4: dual_rail_logic_VECTOR(1 downto 0);
signal pos10: dual_rail_logic_VECTOR(3 downto 0);
signal pos13: dual_rail_logic_VECTOR(5 downto 0);
signal pos2i: dual_rail_logic_VECTOR(6 downto 0);
	signal pos3: dual_rail_logic_VECTOR(7 downto 0);
	signal pos: dual_rail_logic_VECTOR(71 downto 0);
	signal pos1, R, Ro: dual_rail_logic_VECTOR(71 downto 0);
signal s0_1, c0_1, s0_1r, c0_1r: dual_rail_logic_VECTOR(33 downto 2);
	signal s0_2, s0_2r: dual_rail_logic_VECTOR(35 downto 3);
signal c0_2, c0_2r: dual_rail_logic_VECTOR(35 downto 4);
signal s0_3, s0_3r: dual_rail_logic_VECTOR(38 downto 6);
signal c0_3, c0_3r: dual_rail_logic_VECTOR(38 downto 7);
signal s0_4, s0_4r: dual_rail_logic_VECTOR(41 downto 9);
signal c0_4, c0_4r: dual_rail_logic_VECTOR(41 downto 10);
signal s0_5, s0_5r: dual_rail_logic_VECTOR(44 downto 12);
signal c0_5, c0_5r: dual_rail_logic_VECTOR(44 downto 13);
signal s0_6, s0_6r: dual_rail_logic_VECTOR(47 downto 15);
signal c0_6, c0_6r: dual_rail_logic_VECTOR(47 downto 16);
signal s0_7, s0_7r: dual_rail_logic_VECTOR(50 downto 18);
signal c0_7, c0_7r: dual_rail_logic_VECTOR(50 downto 19);
signal s0_8, s0_8r: dual_rail_logic_VECTOR(53 downto 21);
signal c0_8, c0_8r: dual_rail_logic_VECTOR(53 downto 22);
signal s0_9, s0_9r: dual_rail_logic_VECTOR(56 downto 24);
signal c0_9, c0_9r: dual_rail_logic_VECTOR(56 downto 25);
signal s0_10, s0_10r: dual_rail_logic_VECTOR(59 downto 27);
signal c0_10, c0_10r: dual_rail_logic_VECTOR(59 downto 28);
	signal s1_1, s1_1r: dual_rail_logic_VECTOR(35 downto 3);
signal c1_1, c1_1r: dual_rail_logic_VECTOR(36 downto 3);
signal s1_2, s1_2r: dual_rail_logic_VECTOR(38 downto 4);
signal c1_2, c1_2r: dual_rail_logic_VECTOR(39 downto 6);
signal s1_3, s1_3r: dual_rail_logic_VECTOR(44 downto 8);
signal c1_3, c1_3r: dual_rail_logic_VECTOR(45 downto 10);
signal s1_4, s1_4r: dual_rail_logic_VECTOR(47 downto 13);
signal c1_4, c1_4r: dual_rail_logic_VECTOR(48 downto 15);
signal s1_5, s1_5r: dual_rail_logic_VECTOR(53 downto 17);
signal c1_5, c1_5r: dual_rail_logic_VECTOR(54 downto 19);
signal s1_6, s1_6r: dual_rail_logic_VECTOR(56 downto 22);
signal c1_6, c1_6r: dual_rail_logic_VECTOR(57 downto 24);
signal s1_7, s1_7r: dual_rail_logic_VECTOR(62 downto 26);
signal c1_7, c1_7r: dual_rail_logic_VECTOR(61 downto 28);
signal s1_8, s1_8r: dual_rail_logic_VECTOR(72 downto 31);
```

```
    signal c1_8, c1_8r: dual_rail_logic_VECTOR(59 downto 31);
    signal s2_1, s2_1r: dual_rail_logic_VECTOR(40 downto 4);
    signal c2_1, c2_1r: dual_rail_logic_VECTOR(39 downto 4);
    signal s2_2, s2_2r: dual_rail_logic_VECTOR(45 downto 5);
    signal c2_2, c2_2r: dual_rail_logic_VECTOR(46 downto 8);
    signal s2_3, s2_3r: dual_rail_logic_VECTOR(53 downto 11);
    signal c2_3, c2_3r: dual_rail_logic_VECTOR(54 downto 15);
    signal s2_4, s2_4r: dual_rail_logic_VECTOR(60 downto 18);
    signal c2_4, c2_4r: dual_rail_logic_VECTOR(60 downto 22);
    signal s2_5, s2_5r: dual_rail_logic_VECTOR(72 downto 25);
    signal c2_5, c2_5r: dual_rail_logic_VECTOR(62 downto 28);
    signal s3_1: dual_rail_logic_VECTOR(45 downto 5);
    signal c3_1: dual_rail_logic_VECTOR(44 downto 5);
    signal s3_2: dual_rail_logic_VECTOR(54 downto 6);
    signal c3_2: dual_rail_logic_VECTOR(54 downto 11);
    signal s3_3: dual_rail_logic_VECTOR(72 downto 16);
    signal c3_3: dual_rail_logic_VECTOR(62 downto 22);
    signal s3_4: dual_rail_logic_VECTOR(58 downto 26);
    signal s4_1, s4_1r: dual_rail_logic_VECTOR(72 downto 6);
   signal c4_1, c4_1r: dual_rail_logic_VECTOR(62 downto 6);
    signal s4_2, s4_2r: dual_rail_logic_VECTOR(55 downto 7);
    signal c4_2, c4_2r: dual_rail_logic_VECTOR(47 downto 16);
    signal s4_3, s4_3r: dual_rail_logic_VECTOR(38 downto 23);
    signal s5_1, s5_1r: dual_rail_logic_VECTOR(72 downto 7);
    signal c5_1, c5_1r: dual_rail_logic_VECTOR(62 downto 7);
    signal s5_2, s5_2r: dual_rail_logic_VECTOR(48 downto 8);
    signal c5_2, c5_2r: dual_rail_logic_VECTOR(38 downto 23);
    signal s6_1, s6_1r: dual_rail_logic_VECTOR(72 downto 8);
    signal c6_1, c6_1r: dual_rail_logic_VECTOR(62 downto 8);
    signal s6_2, s6_2r: dual_rail_logic_VECTOR(39 downto 9);

    signal s7_1: dual_rail_logic_VECTOR(72 downto 0);
    signal s7_1r: dual_rail_logic_VECTOR(72 downto 0);
    signal c7_1, c7_1r: dual_rail_logic_VECTOR(62 downto 9);
    signal c8_1, s8_1, c8_1r, s8_1r: dual_rail_logic_VECTOR(71 downto 0);
        signal s8_2, s8_2r: dual_rail_logic_VECTOR(10 downto 8);
        signal s9_1, c9_1, s9_1r, c9_1r: dual_rail_logic_VECTOR(71 downto 0);

    signal s10_1, c10_1, s10_1r, c10_1r: dual_rail_logic_VECTOR(71 downto 1);                    --

    signal s10_1r5, c10_1r5: dual_rail_logic_VECTOR(71 downto 36);
    signal c11_1: dual_rail_logic_VECTOR(36 downto 2);
    signal c11_1r: dual_rail_logic_VECTOR(71 downto 36);
        signal e_31, e_32, nSh, AS, AS_t, AS1, AS3, MS, MS1, MS1i, MSo, MSoo, MS_t, pos_0, pos_0_a, pos_1,
        MS3, complete, ShSlAS, ShAS, SlAS: dual_rail_logic;
        signal e: dual_rail_logic_VECTOR(72 downto 63);
    signal Ac: dual_rail_logic_VECTOR(71 downto 0);

-- overflow signals--------------------------
 signal ko_16b5: std_logic_vector(2 downto 0);
 signal ki16b5: std_logic;
-------------------------------------------
 signal ko_9b5: std_logic_vector(2 downto 0);
```

```
 signal ki9b5: std_logic;

 signal s10_1_72, c10_1_72, s10_1_72r2, c10_1_72r2: dual_rail_logic;
-----------------------------------------------
   component ncl_register_D
              generic(width: positive ;
         initial_value: integer);
              port(D: in dual_rail_logic_vector(width-1 downto 0);
                 ki: in std_logic_vector(width-1 downto 0);
                 rst: in std_logic;
                 Q: out dual_rail_logic_vector(width-1 downto 0);
                 ko: out std_logic_vector(width-1 downto 0));
   end component;

   component and2     -- complete AND function
      port(a, b: in dual_rail_logic;
         z: out dual_rail_logic);
   end component;

   component and2i          -- incomplete AND function
      port(a, b: in dual_rail_logic;
         z: out dual_rail_logic);
   end component;


   component full_add
      port(c_in, x, y: in dual_rail_logic;
         c_out, s: out dual_rail_logic);
   end component;

   component half_add
      port(x, y: in dual_rail_logic;
         c_out, s: out dual_rail_logic);
   end component;

   component comp
                        generic(width: in integer := 4);
                        port(a: IN std_logic_vector(width-1 downto 0);
            ko: OUT std_logic);
   end component;

begin
            ko(288 downto 7) <= ko_15(288 downto 7);
  temp15(288 downto 7) <= temp2(288 downto 7);

   REG15_0: ncl_register_D
      generic map(140, -4)
      port map(temp15(146 downto 7), ki_15(146 downto 7), reset, temp15_out(146 downto 7), ko_15(146 downto
            7));         -- ko = ko_15        temp2 = temp15

   REG15_1: ncl_register_D
      generic map(142, -4)  --generic map(142, ??) for feedback macro value is 0.
```

```
   port map(temp15(288 downto 147), ki_15(288 downto 147), reset, temp15_out(288 downto 147), ko_15(288
        downto 147));

        ki_15(9) <= ki16b5;
        ki_15(8) <= ko_16(8);-- ki16b5;
        ki_15(7) <= ki16b5;
        ki_15(216 downto 147) <= ko_16(222 downto 153);

   ki_15(10) <= ki16;
   ki_15(75) <= ki16;
   ki_15(217) <= ki16;
   ki_15(11) <= ki16b2;
   ki_15(76) <= ki16b2;
   ki_15(218) <= ki16b2;

   ACK15: for i in 0 to 6 generate
        ki_15(i+77) <= ki16b3(i);
        ki_15(i+219) <= ki16b3(i);
   end generate ACK15;

   ACK15b:for i in 0 to 61 generate
        ki_15(i+12) <= ki16b4(i);
        ki_15(i+84) <= ki16b4(i);
        ki_15(i+226) <= ki16b4(i);
   end generate ACK15b;

   ki_15(74) <= ki16b5;
   ki_15(146) <= ki16b5;
   ki_15(288) <= ki16b5;

A1_o <= temp15_out(288 downto 217);                    -- A1 = 72 bits
A2_o <= temp15_out(216 downto 147);                    -- A2 = 70 bits
s9_1r <= temp15_out(146 downto 75);                    -- s = 72 bits
c9_1r(71 downto 9) <= temp15_out(74 downto 12);        -- c1 = 63 nits
c9_1r(1 downto 0) <= temp15_out(11 downto 10);         -- c2 = 2 bits

d2mac_mpy1 <= temp15_out(8);

FA_POS0: full_add
   port map(s9_1r(0), c9_1r(0), A1_o(0), c10_1(1), pos_0);

FA10_1: full_add
   port map(s9_1r(1), c9_1r(1), A1_o(1), c10_1(2), s10_1(1));

WALLACE_TREE_H10_2_8: for i in 2 to 8 generate
      HALF_ADDERS: half_add
        port map(s9_1r(i), A1_o(i), c10_1(i+1), s10_1(i));
end generate WALLACE_TREE_H10_2_8;

WALLACE_TREE_F10_9_70: for i in 9 to 70 generate
      FULL_ADDERS: full_add
        port map(s9_1r(i), c9_1r(i), A1_o(i), c10_1(i+1), s10_1(i));
end generate WALLACE_TREE_F10_9_70;
```

```
  c10_1_72 <= c10_1(71);

  FA10_71: full_add
    port map(s9_1r(71), c9_1r(71), A1_o(71), open, s10_1(71)); -- d2add_sub1

  HA10_71b: full_add
    port map(temp15_out(7) , temp15_out(9), A1_o(71), open, s10_1_72);   -- temp15_out(9) contains c9_1(72) ;
        temp15_out(7)  contains s9_1(72)

   temp16(222 downto 7) <= A2_o & s10_1 & pos_0 & c10_1 & c10_1_72 & d2mac_mpy1 & s10_1_72;

    COMP16_0: comp
        generic map(2)
    port map(ko_16b1, ki16);

  COMP16_0_2: comp
        generic map(2)
    port map(ko_16b2, ki16b2);

  COMP16b3: for i in 0 to 6 generate
        COMP_16_0_3: comp
                generic map(2)
        port map(ko_16b3(1+i*2 downto 0+i*2), ki16b3(i));
  end generate COMP16b3;

  COMP16b4: for i in 0 to 61 generate
        COMP_16_0_4: comp
                generic map(2)
        port map(ko_16b4(1+i*2 downto 0+i*2), ki16b4(i));
  end generate COMP16b4;

 ko_16b1 <= ko_16(81 downto 81) & ko_16(10 downto 10);
 ko_16b2 <= ko_16(82 downto 82) & ko_16(11 downto 11);
  COMP_16b3: for i in 0 to 6 generate
                ko_16b3(1+i*2 downto 0+i*2) <= ko_16(12+i downto 12+i) & ko_16(83+i downto 83+i);
  end generate COMP_16b3;
  COMP_16b4: for i in 0 to 61 generate
                ko_16b4(1+i*2 downto 0+i*2) <= ko_16(19+i downto 19+i) & ko_16(90+i downto 90+i);
  end generate COMP_16b4;

-- overflow addition

                ko_16b5(1 downto 0) <= ko_16(7) & ko_16(152);--ko_16b5(2 downto 0) <= ko_16(9) & ko_16(7)
        & ko_16(152);
  COMP_16_0_5: comp
                generic map(2)
        port map(ko_16b5(1 downto 0), ki16b5);
-------------------------------------------------------------------------------------------------------------------------------

  REG16_0: ncl_register_D
    generic map(216, -4)
    port map(temp16(222 downto 7), ki_16(222 downto 7), reset, temp16_out(222 downto 7), ko_16(222 downto
        7));
```

```
        ki_16(9) <= ki9b5;
        ki_16(8) <= ko_9(9);
        ki_16(7) <= ki9b5;


        ki_16(10) <= ki9;
        ki_16(82) <= ki9;
        ki_16(81) <= ko_9(81);

        ki_16(80) <= ki9b5;
        ki_16(152) <= ki9b5;
        ki_16(222) <= ki9b5;

   ACK16: for i in 0 to 68 generate
        ki_16(i+11) <= ki9b2(i);
        ki_16(i+83) <= ki9b2(i);
        ki_16(i+153) <= ki9b2(i);
   end generate ACK16;

A2_o_a <= temp16_out(222 downto 153);
s10_1r <= temp16_out(152 downto 82);
pos_0_a <= temp16_out(81);
c10_1r <= temp16_out(80 downto 10);

mac_mpy1 <= temp16_out(8);

aFA_POS0: half_add
   port map(s10_1r(1), c10_1r(1), c10_1r2(2), pos_1);

aWALLACE_TREE_F10_9_70: for i in 2 to 70 generate
      aFULL_ADDERS: full_add
         port map(s10_1r(i), c10_1r(i), A2_o_a(i), c10_1r2(i+1), s10_1r2(i));
end generate aWALLACE_TREE_F10_9_70;
c10_1_72r2 <= c10_1r2(71);

aFA10_71: full_add
   port map(s10_1r(71), c10_1r(71), A2_o_a(71), open, s10_1r2(71));--add_sub1

aHA10_71b: full_add
   port map(temp16_out(9) , temp16_out(7), A2_o_a(71), open, s10_1_72r2);   -- temp16_out(9) contains
        c10_1(72) ; temp16_out(7)  contains s10_1(72)

temp9(152 downto 8) <= s10_1r2 & pos_1 & pos_0_a & c10_1r2 & c10_1_72r2 & mac_mpy1 & s10_1_72r2;--
        add_sub1

COMP9: comp
        generic map(2)
   port map(ko_9b1, ki9);

COMP9b2: for i in 0 to 68 generate
        COMP_9: comp
                generic map(2)
```

```
        port map(ko_9b2(1+i*2 downto 0+i*2), ki9b2(i));
  end generate COMP9b2;

        ko_9b1 <= ko_9(11 downto 11) & ko_9(82 downto 82);

  COMP_9b2: for i in 0 to 68 generate
                ko_9b2(1+i*2 downto 0+i*2) <= ko_9(12+i downto 12+i) & ko_9(83+i downto 83+i);
  end generate COMP_9b2;

                ko_9b5(1 downto 0) <= ko_9(8) & ko_9(152);--ko_9(10) & ko_9(8) & ko_9(152);
  COMP_9_0_5: comp
                generic map(2)
        port map(ko_9b5(1 downto 0), ki9b5);

  REG9: ncl_register_D
    generic map(145, 0)
    port map(temp9, ki_9, reset, temp9_out, ko_9);

  ki_9 <= ki;

end syn;
library ieee;
use work.dual_rail.all;
use work.ncl_signals.all;
use ieee.std_logic_1164.all;

entity feed_back_RNULL_bypass is
        port(di2a: in dual_rail_logic_VECTOR(281 downto 0);
                reset : in std_logic;
                ki: in std_logic_vector(144 downto 0);
                ko: out std_logic_vector(281 downto 0);
                do2: out dual_rail_logic_VECTOR(144 downto 0));
end;

architecture SYN of feed_back_RNULL_bypass is

signal di2: dual_rail_logic_vector(281 downto 0);
signal do2a: dual_rail_logic_vector(146 downto 0);
signal ki_2a: std_logic_vector(281 downto 0);
signal ko_2a: std_logic_vector(146 downto 0);
signal ki2a: std_logic;
 signal kib: std_logic_vector(146 downto 0);
 signal do2t: dual_rail_logic_vector(146 downto 0);
 signal ko_b: std_logic_vector(144 downto 0);
 signal do2tt: dual_rail_logic_vector(144 downto 0);
 signal ki16: std_logic;
 signal cc: dual_rail_logic;
 signal kkk: std_logic_vector(2 downto 0);

            component ncl_register_D
            generic(width: positive ;
        initial_value: integer);
            port(D: in dual_rail_logic_vector(width-1 downto 0);
```

```
                ki: in std_logic_vector(width-1 downto 0);
                rst: in std_logic;
                Q: out dual_rail_logic_vector(width-1 downto 0);
                ko: out std_logic_vector(width-1 downto 0));
  end component;

  component comp
                    generic(width: in integer := 4);
                    port(a: IN std_logic_vector(width-1 downto 0);
          ko: OUT std_logic);
  end component;

      component th33w2x0
       port(a: in std_logic;
           b: in std_logic;
           c: in std_logic;
           z: out std_logic);
     end component;

  component th12x0
    port(a: in std_logic;
        b: in std_logic;
        z: out std_logic);
 end component;

  component full_add
     port(c_in, x, y: in dual_rail_logic;
         c_out, s: out dual_rail_logic);
  end component;

  component half_add
     port(x, y: in dual_rail_logic;
         c_out, s: out dual_rail_logic);
  end component;

begin

 REG4: ncl_register_D
   generic map(140, -4)
 port map(di2a(139 downto 0), ki_2a(139 downto 0), reset, di2(139 downto 0), ko(139 downto 0));
 REG4b: ncl_register_D
   generic map(142, -4)
 port map(di2a(281 downto 140), ki_2a(281 downto 140), reset, di2(281 downto 140), ko(281 downto 140));

 ACK4_1: for i in 0 to 2 generate
           ki_2a(i) <= ko_2a(i);
 end generate ACK4_1;

 ACK4_2: for i in 3 to 4 generate
           ki_2a(i) <= ko_2a(i);
           ki_2a(i+137) <= ko_2a(i);
 end generate ACK4_2;
```

```
ACK4_3: for i in 5 to 11 generate
        ki_2a(i+137) <= ko_2a(i);
end generate ACK4_3;

ACK4_4: for i in 12 to 72 generate
        ki_2a(i-7) <= ko_2a(i);
        ki_2a(i+137) <= ko_2a(i);
end generate ACK4_4;

ACK4_5: for i in 73 to 74 generate
        ki_2a(i-7) <= ko_2a(i);
end generate ACK4_5;

ACK4_6: for i in 75 to 146 generate
        ki_2a(i-7) <= ko_2a(i);
        ki_2a(i+137-2) <= ko_2a(i);
end generate ACK4_6;

        COMB1: for i in 0 to 2 generate       -->> control signals
          --do2a(i) <=  di2(i);
                OV_rail0: th33w2x0
                        port map(di2(i).rail0, di2(i).rail0, di2(i).rail1,   do2a(i).rail0); -- OV bits
                OV_rail1: th33w2x0
                        port map(di2(i).rail1, di2(i).rail0, di2(i).rail1,   do2a(i).rail1);

        end generate COMB1;
-------------------------------------------------------------------------------------------------------------
        COMB2: for i in 3 to 4 generate       -->> A2  2bits  PP-c-(1-0)
                A2_gate_rail0: th33w2x0
                        port map(di2(i).rail0, di2(i+137).rail0, di2(i+137).rail1,   do2a(i).rail0); --A2-- 65 bits
                A2_gate_rail1: th33w2x0
                        port map(di2(i).rail1, di2(i+137).rail0, di2(i+137).rail1,   do2a(i).rail1);
        end generate COMB2;


        COMB3: for i in 5 to 11 generate       -->> A2 append 7 zeros  pp-c-(8-2)
                 A2b_gate_rail0: th12x0
                        port map(di2(i+137).rail0, di2(i+137).rail1,   do2a(i).rail0);
                 A2b_gate_rail1:  do2a(i).rail1 <= '0';
        end generate COMB3;


        COMB2b: for i in 12 to 72 generate       -->> A2  63bits     pp-c-(71-9)
                A2_gate_rail0: th33w2x0
                        port map(di2(i-7).rail0, di2(i+137).rail0, di2(i+137).rail1,   do2a(i).rail0); --A2-- 65 bits
                A2_gate_rail1: th33w2x0
                        port map(di2(i-7).rail1, di2(i+137).rail0, di2(i+137).rail1,   do2a(i).rail1);
        end generate COMB2b;

        COMB2c: for i in 73 to 74 generate       -->> A2  63bits     pp-c-(71-9)
                A2_gate_rail0: th33w2x0
                        port map(di2(i-7).rail0, di2(i+137-2).rail0, di2(i+137-2).rail1,   do2a(i).rail0); --A2-- 65
        bits
```

```
                A2_gate_rail1: th33w2x0
                        port map(di2(i-7).rail1, di2(i+137-2).rail0, di2(i+137-2).rail1,   do2a(i).rail1);
        end generate COMB2c;


        COMB4: for i in 75 to 146 generate
                A1_gate_rail0: th33w2x0
                        port map(di2(i-5-2).rail0, di2(i+137-2).rail0, di2(i+137-2).rail1,   do2a(i).rail0);--
                A1_gate_rail1: th33w2x0
                        port map(di2(i-5-2).rail1, di2(i+137-2).rail0, di2(i+137-2).rail1,   do2a(i).rail1);
        end generate COMB4;

 REG4c: ncl_register_D
   generic map(147, -4)
  port map(do2a(146 downto 0), kib(146 downto 0), reset, do2t(146 downto 0), ko_2a(146 downto 0));

          kib(2 downto 0) <= ko_b(2 downto 0);
          kib(74 downto 6) <= ko_b(72 downto 4);
          kib(146 downto 77) <= ko_b(144 downto 75);

          kib(3) <= ki16;
          kib(4) <= ki16;
          kib(5) <= ki16;
          kib(75) <= ki16;
          kib(76) <= ki16;
         kkk(2 downto 0) <= ko_b(74 downto 73) & ko_b(3);
         COMPrc1: comp
           generic map(3)
   port map(kkk(2 downto 0), ki16);

   HA_POS1: half_add
      port map(do2t(3), do2t(75), cc, do2tt(73));

   FULL_ADDERS: full_add
      port map(do2t(4), do2t(76), cc, do2tt(3), do2tt(74));

 do2tt(2 downto 0) <= do2t(2 downto 0);
 do2tt(72 downto 4) <= do2t(74 downto 6);
 do2tt(144 downto 75) <= do2t(146 downto 77);

 REG4d: ncl_register_D
   generic map(145, 0)
  port map(do2tt(144 downto 0), ki(144 downto 0), reset, do2(144 downto 0), ko_b(144 downto 0));

end syn;

library ieee;
use ieee.std_logic_1164.all;
use work.ncl_signals.all;

entity feed_back_bypass is
   port(temp2: in dual_rail_logic_VECTOR(305 downto 7);
       reset : in std_logic;
```

```
            ki: in std_logic_vector(152 downto 8);
        ko: out std_logic_vector(305 downto 7);
        temp9_out: out dual_rail_logic_VECTOR(152 downto 8));
end;

architecture BEHAVIOR of feed_back_bypass is

  signal temp2b: dual_rail_logic_VECTOR(305 downto 7);
            signal kob: std_logic_vector(305 downto 7);
            signal di1, di2: dual_rail_logic_vector(281 downto 0);
            signal ko1, ko2: std_logic_vector(281 downto 0); -
            signal kod: std_logic_vector(281 downto 0);
            signal kot: std_logic_vector(17 downto 0);

   signal tempR1_out: dual_rail_logic_VECTOR(2 downto 0);
             signal ki_R1: std_logic_vector(2 downto 0);
   signal tempR2,tempR2_out: dual_rail_logic_VECTOR(141 downto 0);
            signal do1, do2: dual_rail_logic_vector(144 downto 0); -- 144>141
            signal ki1: std_logic_vector(144 downto 0);-- 144>141

   signal tempR3,tempR3_out: dual_rail_logic_VECTOR(2 downto 0);

  signal temp9, temp9_in: dual_rail_logic_VECTOR(152 downto 8);
            signal ki_9, ko_9,ko_9b: std_logic_vector(152 downto 8);


            component feed_back_RNULL
            port(temp2: in dual_rail_logic_VECTOR(288 downto 7);
                        reset : in std_logic;
                        ki: in std_logic_vector(152 downto 8);
                        ko: out std_logic_vector(288 downto 7);
                        temp9_out: out dual_rail_logic_VECTOR(152 downto 8));
            end component;

            component feed_back_RNULL_bypass
            port(di2a: in dual_rail_logic_VECTOR(281 downto 0);
                        reset : in std_logic;
                        ki: in std_logic_vector(144 downto 0);
                        ko: out std_logic_vector(281 downto 0);
                        do2: out dual_rail_logic_VECTOR(144 downto 0));
            end component;

    component demux
            port (a: IN dual_rail_logic;
            rst: IN std_logic;
                        ki1, ki2: IN std_logic;
            s1, s2: IN std_logic;
            z1, z2: OUT dual_rail_logic;
                        ko: OUT std_logic);
    end component;

   component demux_rD0
            port (a: IN dual_rail_logic;
```

```
            rst: IN std_logic;
                        ki1, ki2: IN std_logic;
            s1, s2: IN std_logic;
            z1, z2: OUT dual_rail_logic;
                        ko: OUT std_logic);
    end component;

    component mux
            generic(width: in integer := 1);
            port(a1, a2: IN dual_rail_logic_vector(width-1 downto 0);
            z: OUT dual_rail_logic_vector(width-1 downto 0));
    end component;

    component comp
                        generic(width: in integer := 4);
                        port(a: IN std_logic_vector(width-1 downto 0);
            ko: OUT std_logic);
    end component;

    component selct
            port (ki, rst: IN std_logic;
            s1, s2: OUT std_logic);
    end component;

    component selct_rD0
       port (ki, rst: IN std_logic;
            s1, s2: OUT std_logic);
    end component;

    component ncl_register_D
                generic(width: positive ;
          initial_value: integer);
                port(D: in dual_rail_logic_vector(width-1 downto 0);
                   ki: in std_logic_vector(width-1 downto 0);
                   rst: in std_logic;
                   Q: out dual_rail_logic_vector(width-1 downto 0);
                   ko: out std_logic_vector(width-1 downto 0));
    end component;

         component th33w2x0
          port(a: in std_logic;
              b: in std_logic;
              c: in std_logic;
              z: out std_logic);
       end component;

    component th12x0
      port(a: in std_logic;
          b: in std_logic;
          z: out std_logic);
   end component;

begin
```

```
        ko(305 downto 164) <= kod(281 downto 140);

        ko(163) <= kot(17);
        ko(162) <= kot(16);
        ko(161) <= kot(15);
        ko(160) <= kot(14);
        ko(159) <= kot(13);
        ko(158) <= kot(12);
        ko(157) <= kot(11);
        ko(156) <= kot(10);
        ko(155) <= kot(9);
        --
        ko(154) <= kot(8);
        ko(153) <= kot(7);
        ko(152) <= kot(6);
        ko(151) <= kot(5);

        ko(150) <= kot(4);
        ko(149) <= kot(3);
        ko(148) <= kot(2);
        ko(147) <= kot(1);

        ko(146 downto 9) <= kod(139 downto 2);

 ko(8) <= kot(0);

        ko(7) <= kod(0);

-------------------------------------------------------------------------------------------------------------------
        COMP_in18: comp
                generic map(14)--10
                port map(kod(281 downto 268), kot(17));--281 downto 272

        COMP_in17: comp
                generic map(16)
                port map(kod(267 downto 252), kot(16));--271 downto 256

        COMP_in16: comp
                generic map(16)
                port map(kod(251 downto 236), kot(15));--255 downto 240

        COMP_in15: comp
                generic map(16)
                port map(kod(235 downto 220), kot(14));--239 downto 224

        COMP_in14: comp
                generic map(16)
                port map(kod(219 downto 204), kot(13));--223 downto 208

        COMP_in13: comp
                generic map(16)
                port map(kod(203 downto 188), kot(12));--207 downto 192
```

```
COMP_in12: comp
        generic map(16)
        port map(kod(187 downto 172), kot(11));--191 downto 176

COMP_in11: comp
        generic map(16)
        port map(kod(171 downto 156), kot(10));--175 downto 160

COMP_in10: comp
        generic map(16)
        port map(kod(155 downto 140), kot(9));--159 downto 144

COMP_in9: comp
        generic map(12)
        port map(kod(139 downto 128), kot(8));--143 downto 128

COMP_in8: comp
        generic map(16)
        port map(kod(127 downto 112), kot(7));

COMP_in7: comp
        generic map(16)
        port map(kod(111 downto 96), kot(6));

COMP_in6: comp
        generic map(16)
        port map(kod(95 downto 80), kot(5));

COMP_in5: comp
        generic map(16)
        port map(kod(79 downto 64), kot(4));

COMP_in4: comp
        generic map(16)
        port map(kod(63 downto 48), kot(3));

COMP_in3: comp
        generic map(16)
        port map(kod(47 downto 32), kot(2));

COMP_in2: comp
        generic map(16)
        port map(kod(31 downto 16), kot(1));

COMP_in: comp
        generic map(16)
        port map(kod(15 downto 0), kot(0));

DEMUX_INPUT_FBh: for i in 275 to 288 generate--263 to 278
        comp11h: demux--demux_rD0
                port map(temp2(i+17), reset, ko1(i-7), ko2(i-7), temp2(163).rail0, temp2(163).rail1,
di1(i-7), di2(i-7), kod(i-7));
```

```
end generate DEMUX_INPUT_FBh;

DEMUX_INPUT_FBg: for i in 259 to 274 generate--247 to 262
	comp11g: demux--demux_rD0
		port map(temp2(i+17), reset, ko1(i-7), ko2(i-7), temp2(162).rail0, temp2(162).rail1,
di1(i-7), di2(i-7), kod(i-7));
end generate DEMUX_INPUT_FBg;

DEMUX_INPUT_FBf: for i in 243 to 258 generate--231 to 246
	comp11f: demux--demux_rD0
		port map(temp2(i+17), reset, ko1(i-7), ko2(i-7), temp2(161).rail0, temp2(161).rail1,
di1(i-7), di2(i-7), kod(i-7));
end generate DEMUX_INPUT_FBf;

DEMUX_INPUT_FBe: for i in 227 to 242 generate--215 to 230
	comp11e: demux--demux_rD0
		port map(temp2(i+17), reset, ko1(i-7), ko2(i-7), temp2(160).rail0, temp2(160).rail1,
di1(i-7), di2(i-7), kod(i-7));
end generate DEMUX_INPUT_FBe;

DEMUX_INPUT_FBd: for i in 211 to 226 generate--199 to 214
	comp11d: demux--demux_rD0
		port map(temp2(i+17), reset, ko1(i-7), ko2(i-7), temp2(159).rail0, temp2(159).rail1,
di1(i-7), di2(i-7), kod(i-7));
end generate DEMUX_INPUT_FBd;

DEMUX_INPUT_FBc: for i in 195 to 210 generate--183 to 198
	comp11c: demux--demux_rD0
		port map(temp2(i+17), reset, ko1(i-7), ko2(i-7), temp2(158).rail0, temp2(158).rail1,
di1(i-7), di2(i-7), kod(i-7));
end generate DEMUX_INPUT_FBc;

DEMUX_INPUT_FBb: for i in 179 to 194 generate--167 to 182
	comp11b: demux--demux_rD0
		port map(temp2(i+17), reset, ko1(i-7), ko2(i-7), temp2(157).rail0, temp2(157).rail1,
di1(i-7), di2(i-7), kod(i-7));
end generate DEMUX_INPUT_FBb;

DEMUX_INPUT_FBa: for i in 163 to 178 generate--151 to 166
	comp11a: demux--demux_rD0
		port map(temp2(i+17), reset, ko1(i-7), ko2(i-7), temp2(156).rail0, temp2(156).rail1,
di1(i-7), di2(i-7), kod(i-7));
end generate DEMUX_INPUT_FBa;

DEMUX_INPUT_FB: for i in 147 to 162 generate--147 to 150
	comp11: demux--demux_rD0
		port map(temp2(i+17), reset, ko1(i-7), ko2(i-7), temp2(155).rail0, temp2(155).rail1,
di1(i-7), di2(i-7), kod(i-7));
end generate DEMUX_INPUT_FB;

DEMUX_INPUT_FFh: for i in 135 to 146 generate--135 to 146
	comp12h: demux
```

```
		port map(temp2(i), reset, ko1(i-7), ko2(i-7), temp2(154).rail0, temp2(154).rail1, di1(i-7),
di2(i-7), kod(i-7));
end generate DEMUX_INPUT_FFh;

DEMUX_INPUT_FFg: for i in 119 to 134 generate
	comp12g: demux
		port map(temp2(i), reset, ko1(i-7), ko2(i-7), temp2(153).rail0, temp2(153).rail1, di1(i-7),
di2(i-7), kod(i-7));
end generate DEMUX_INPUT_FFg;

DEMUX_INPUT_FFf: for i in 103 to 118 generate
	comp12f: demux
		port map(temp2(i), reset, ko1(i-7), ko2(i-7), temp2(152).rail0, temp2(152).rail1, di1(i-7),
di2(i-7), kod(i-7));
end generate DEMUX_INPUT_FFf;

DEMUX_INPUT_FFe: for i in 87 to 102 generate
	comp12e: demux
		port map(temp2(i), reset, ko1(i-7), ko2(i-7), temp2(151).rail0, temp2(151).rail1, di1(i-7),
di2(i-7), kod(i-7));
end generate DEMUX_INPUT_FFe;

DEMUX_INPUT_FFd: for i in 71 to 86 generate
	comp12d: demux
		port map(temp2(i), reset, ko1(i-7), ko2(i-7), temp2(150).rail0, temp2(150).rail1, di1(i-7),
di2(i-7), kod(i-7));
end generate DEMUX_INPUT_FFd;

DEMUX_INPUT_FFc: for i in 55 to 70 generate
	comp12c: demux
		port map(temp2(i), reset, ko1(i-7), ko2(i-7), temp2(149).rail0, temp2(149).rail1, di1(i-7),
di2(i-7), kod(i-7));
end generate DEMUX_INPUT_FFc;

DEMUX_INPUT_FFb: for i in 39 to 54 generate
	comp12b: demux
		port map(temp2(i), reset, ko1(i-7), ko2(i-7), temp2(148).rail0, temp2(148).rail1, di1(i-7),
di2(i-7), kod(i-7));
end generate DEMUX_INPUT_FFb;

DEMUX_INPUT_FFa: for i in 23 to 38 generate
	comp12a: demux
		port map(temp2(i), reset, ko1(i-7), ko2(i-7), temp2(147).rail0, temp2(147).rail1, di1(i-7),
di2(i-7), kod(i-7));
end generate DEMUX_INPUT_FFa;

DEMUX_INPUT_FF: for i in 7 to 22 generate
	comp12: demux
		port map(temp2(i), reset, ko1(i-7), ko2(i-7), temp2(8).rail0, temp2(8).rail1, di1(i-7), di2(i-
7), kod(i-7));
end generate DEMUX_INPUT_FF;

COMB1: feed_back_RNULL
```

```
                port map(di1, reset, ki1, ko1, do1);

    COMB2: feed_back_RNULL_bypass
        port map(di2, reset, ki1, ko2, do2);

        MUX_OUTPUT: mux
                generic map(145)
                port map(do1, do2, temp9_out); -- tempR2:temp9_in
 ki1 <= ki(152 downto 8);

end BEHAVIOR;

Library IEEE;
use IEEE.std_logic_1164.all;
use work.ncl_signals.all;

entity FEED_BACK_comp is
 port(FF_in: in DUAL_RAIL_LOGIC_VECTOR(156 downto 0);
            reset : in STD_LOGIC;
                ki: in STD_LOGIC_VECTOR(144 downto 0);
            ko: out STD_LOGIC_VECTOR(156 downto 0);
            PPC_out: out dual_rail_logic_VECTOR(144 downto 0));
end;

architecture TEST of FEED_BACK_comp is
    signal temp2: DUAL_RAIL_LOGIC_VECTOR(298 downto 0);
    signal kif,kif3: STD_LOGIC_VECTOR(144 downto 0);
    signal kif2,kif2_1,kif2_2,kif2_3,kif2b,kif2b_1,kif2b_2,kif2b_3,kif2c: STD_LOGIC;
    signal ko_reg: STD_LOGIC_VECTOR(298 downto 0);
    signal ko_2a: STD_LOGIC_VECTOR(298 downto 0); --
    signal temp_out,temp_out2: DUAL_RAIL_LOGIC_VECTOR(144 downto 0);

        component feed_back_bypass
          port(temp2: in DUAL_RAIL_LOGIC_VECTOR(305 downto 7);
            reset : in STD_LOGIC;
                ki: in STD_LOGIC_VECTOR(152 downto 8);
            ko: out STD_LOGIC_VECTOR(305 downto 7);
            temp9_out: out dual_rail_logic_VECTOR(152 downto 8));
        end component;

        component comp
                generic(width: in integer := 4);
                port(a: IN std_logic_vector(width-1 downto 0);
                ko: OUT std_logic);
        end component;

        component th22x0
   port(a: in std_logic;
      b: in std_logic;
      z: out std_logic );
 end component;

        component ncl_register_D
```

```
            generic(width: positive ;
        initial_value: integer);
            port(D: in dual_rail_logic_vector(width-1 downto 0);
               ki: in std_logic_vector(width-1 downto 0);
               rst: in std_logic;
               Q: out dual_rail_logic_vector(width-1 downto 0);
               ko: out std_logic_vector(width-1 downto 0));
 end component;

begin
 temp2(156 downto 0) <= FF_in(156 downto 0);
 temp2(298 downto 157) <= temp_out(144 downto 3);

 ko <= ko_reg(156 downto 0);

     FUUT: feed_back_bypass
          port map(temp2, reset, kif, ko_reg, temp_out);

 PPC_out <= temp_out;

  COMPFB: for i in 3 to 144 generate
            GATE: th22x0
                     port map(ki(i) ,ko_reg(i+137+17), kif(i));
   end generate COMPFB;

 kif(2 downto 0) <= ki(2 downto 0);

end;

library ieee;
use ieee.std_logic_1164.all;
use work.ncl_signals.all;

entity RCA_comp_ncr is
 port(FF_in: in DUAL_RAIL_LOGIC_VECTOR(152 downto 8);
               reset : in STD_LOGIC;
                    ki: in STD_LOGIC_VECTOR(74 downto 2);
               ko: out STD_LOGIC_VECTOR(152 downto 8);
               PPC_out: out dual_rail_logic_VECTOR(74 downto 2));
end;

architecture BEHAVIOR of RCA_comp_ncr is

          signal di1, di2: dual_rail_logic_vector(152 downto 8);
          signal ko1, ko2: std_logic_vector(152 downto 8);
          signal kod: std_logic_vector(152 downto 8);
          signal do1, do2: dual_rail_logic_vector(74 downto 2);
          signal ki1, ki2: std_logic_vector(74 downto 2);
          signal ds1, ds2: std_logic_vector(74 downto 2);
          signal s1, s2: std_logic_vector(152 downto 8);
          signal temp4, temp4_in, temp4_out: dual_rail_logic_VECTOR(74 downto 2);
          signal ki_4, ko_4: std_logic_vector(74 downto 2);
```

```
component RCA_comp
  port(FF_in: in DUAL_RAIL_LOGIC_VECTOR(152 downto 8);
                  reset : in STD_LOGIC;
                        ki: in STD_LOGIC_VECTOR(74 downto 2);
                  ko: out STD_LOGIC_VECTOR(152 downto 8);
                  PPC_out: out dual_rail_logic_VECTOR(74 downto 2));
end component;

   component demux
            port (a: IN dual_rail_logic;
            rst: IN std_logic;
                        ki1, ki2: IN std_logic;
            s1, s2: IN std_logic;
            z1, z2: OUT dual_rail_logic;
                        ko: OUT std_logic);
   end component;

  component demux_rD0
            port (a: IN dual_rail_logic;
            rst: IN std_logic;
                        ki1, ki2: IN std_logic;
            s1, s2: IN std_logic;
            z1, z2: OUT dual_rail_logic;
                        ko: OUT std_logic);
  end component;

   component mux
            generic(width: in integer := 1);
            port(a1, a2: IN dual_rail_logic_vector(width-1 downto 0);
            z: OUT dual_rail_logic_vector(width-1 downto 0));
   end component;

            component ncl_register_D
               generic(width: positive ;
          initial_value: integer);
               port(D: in dual_rail_logic_vector(width-1 downto 0);
                  ki: in std_logic_vector(width-1 downto 0);
                  rst: in std_logic;
                  Q: out dual_rail_logic_vector(width-1 downto 0);
                  ko: out std_logic_vector(width-1 downto 0));
   end component;

   component comp
                        generic(width: in integer := 4);
                        port(a: IN std_logic_vector(width-1 downto 0);
            ko: OUT std_logic);
   end component;

   component selct
            port (ki, rst: IN std_logic;
            s1, s2: OUT std_logic);
   end component;
```

```
   component selct_rD0
      port (ki, rst: IN std_logic;
            s1, s2: OUT std_logic);
   end component;

begin
            DEMUX_INPUT_FF: for i in 8 to 152 generate
                        comp12: demux
                                    port map(FF_in(i), reset, ko1(i), ko2(i), s1(i), s2(i), di1(i), di2(i), kod(i));
            end generate DEMUX_INPUT_FF;

            ko <= kod;

            SELECT_INPUT_FF: for i in 8 to 152 generate
              SELECT_INPUT_1: selct
                          port map(kod(i), reset, s1(i), s2(i));
 end generate SELECT_INPUT_FF;

            COMB1: RCA_comp
                         port map(di1, reset, ki1, ko1, do1);

     COMB2: RCA_comp
          port map(di2, reset, ki2, ko2, do2);

            MUX_OUTPUT: mux
                        generic map(73)
                        port map(do1, do2, temp4);

            SELECT_OUTPUT: for i in 2 to 74 generate
              SELECT_OUTPUT: selct
                          port map(ko_4(i), reset, ki1(i), ki2(i));
 end generate SELECT_OUTPUT;

   temp4_in <= temp4;

   REG4: ncl_register_D
      generic map(73, -4)
      port map(temp4_in, ki, reset, PPC_out, ko_4);
end BEHAVIOR;

use work.ncl_signals.all;
use work.dual_rail.all;
library ieee;
use ieee.std_logic_1164.all;

entity mac32x32 is
   port(x, y: in dual_rail_logic_VECTOR(31 downto 0);
        sign: in dual_rail_logic_VECTOR(1 downto 0);
        add_sub, mac_mpy: in dual_rail_logic;
        reset, ki: in std_logic;
        ko: out  std_logic_VECTOR(67 downto 0);
        Aout: out dual_rail_logic_VECTOR(71 downto 0);
        OV: out dual_rail_logic);
```

```
end;

architecture SYN of mac32x32 is

 signal OV2: dual_rail_logic;

        signal temp1, temp1_in, temp1_out : dual_rail_logic_VECTOR(67 downto 0);
        signal ki_1, ko_1: std_logic_vector(67 downto 0);
        signal temp2, temp2_in, temp2_out,
        temp2_outb,temp2_outc,temp2_outd,temp2_oute,temp2_outf,temp2_outg,temp2_outh:
        dual_rail_logic_VECTOR(288 downto 7);
        signal kof, ki_2, ko_2,ki_2b,ki_2c,ki_2d,ki_2e,ki_2f,ki_2g: std_logic_vector(288 downto 7);

    signal temp9, temp9_in, temp9_out, temp9_outb: dual_rail_logic_VECTOR(152 downto 8);
        signal ki_9, ko_9, ko_9b: std_logic_vector(152 downto 8);

        signal Ain: dual_rail_logic_VECTOR(141 downto 0);

        signal kif: std_logic_vector(152 downto 8);
        signal kif3: std_logic;

        signal temp11, temp11_in, temp11_out: dual_rail_logic_VECTOR(75 downto 0);
        signal ki_11, ko_11: std_logic_vector(75 downto 0);


    signal temp3, temp3_in, temp3_out: dual_rail_logic_VECTOR(72 downto 0);
        signal ki_3, ko_3, ki_f, ko_3A: std_logic_vector(72 downto 0);
        signal Aout_old: dual_rail_logic;

        signal ko_11a: std_logic_vector(1 downto 0);
        signal kif2: std_logic;
        signal kof2: std_logic_vector(75 downto 0);

        signal kif4: std_logic_vector(72 downto 0);

 component feedforward_comp
  port(FF_in: in DUAL_RAIL_LOGIC_VECTOR(67 downto 0);
                reset : in STD_LOGIC;
                        ki: in STD_LOGIC_VECTOR(156 downto 0);
                ko: out STD_LOGIC_VECTOR(67 downto 0);
                PPC_out: out dual_rail_logic_VECTOR(156 downto 0)); -- width = 139
 end component;

component FEED_BACK_comp
 port(FF_in: in DUAL_RAIL_LOGIC_VECTOR(156 downto 0);
                reset : in STD_LOGIC;
                     ki: in STD_LOGIC_VECTOR(144 downto 0);
                ko: out STD_LOGIC_VECTOR(156 downto 0);
                PPC_out: out dual_rail_logic_VECTOR(144 downto 0));
end component;

   component ncl_register_D
            generic(width: positive ;
```

```
        initial_value: integer);
             port(D: in dual_rail_logic_vector(width-1 downto 0);
                ki: in std_logic_vector(width-1 downto 0);
                rst: in std_logic;
                Q: out dual_rail_logic_vector(width-1 downto 0);
                ko: out std_logic_vector(width-1 downto 0));
   end component;

   component comp
          generic(width: in integer := 4);
          port(a: IN std_logic_vector(width-1 downto 0);
          ko: OUT std_logic);
   end component;

component RCA_comp_ncr
  port(FF_in: in DUAL_RAIL_LOGIC_VECTOR(152 downto 8);
                reset : in STD_LOGIC;
                     ki: in STD_LOGIC_VECTOR(74 downto 2);
                ko: out STD_LOGIC_VECTOR(152 downto 8);
                PPC_out: out dual_rail_logic_VECTOR(74 downto 2));
end component;

component RCA_comp
  port(FF_in: in DUAL_RAIL_LOGIC_VECTOR(152 downto 8);
                reset : in STD_LOGIC;
                     ki: in STD_LOGIC_VECTOR(74 downto 2);
                ko: out STD_LOGIC_VECTOR(152 downto 8);
                PPC_out: out dual_rail_logic_VECTOR(74 downto 2));
end component;

component feed_back_RNULL_2
   port(temp1: in dual_rail_logic_VECTOR(75 downto 0);
      reset : in std_logic;
           ki: in std_logic_VECTOR(72 downto 0);
      ko: out std_logic_vector(75 downto 0);
      temp3_out: out dual_rail_logic_VECTOR(72 downto 0));
end component;

component feed_back_ncr_2
   port(temp1: in dual_rail_logic_VECTOR(75 downto 0);
      reset : in std_logic;
           ki: in std_logic_VECTOR(72 downto 0);
      ko: out std_logic_vector(75 downto 0);
      temp3_out: out dual_rail_logic_VECTOR(72 downto 0));
end component;

component th22x0
     port(a: in std_logic;
        b: in std_logic;
        z: out std_logic );
  end component;

component xor2
```

```
  port (a: IN dual_rail_logic ;
        b: IN dual_rail_logic ;
        z: OUT dual_rail_logic);
end component;

component xor2_b
  port (a: IN dual_rail_logic ;
        b: IN dual_rail_logic ;
        z: OUT dual_rail_logic);
end component;

begin
 temp1_in <= x & y & sign & add_sub & mac_mpy;
 ko <= ko_1;

  FF_feedforward_comp: feedforward_comp
   port map(temp1_in, reset, ki_2(163 downto 7), ko_1, temp2_out(163 downto 7));

  REG1a5: ncl_register_D
     generic map(157, -4)
     port map(temp2_out(163 downto 7), ki_2e(163 downto 7), reset, temp2_outf(163 downto 7), ki_2(163
          downto 7));

  REG1a4: ncl_register_D
     generic map(157, -4)
     port map(temp2_outf(163 downto 7), ki_2d(163 downto 7), reset, temp2_oute(163 downto 7), ki_2e(163
          downto 7));

  REG1a3: ncl_register_D
     generic map(157, -4)
     port map(temp2_oute(163 downto 7), ki_2c(163 downto 7), reset, temp2_outd(163 downto 7), ki_2d(163
          downto 7));

  REG1a2: ncl_register_D
     generic map(157, -4)
     port map(temp2_outd(163 downto 7), ki_2b(163 downto 7), reset, temp2_outc(163 downto 7), ki_2c(163
          downto 7));

  REG1a: ncl_register_D
     generic map(157, -4)
     port map(temp2_outc(163 downto 7), kof(163 downto 7), reset, temp2_outb(163 downto 7), ki_2b(163
          downto 7));

  FEEDBACK_bypass: FEED_BACK_comp
     port map(temp2_outb(163 downto 7), reset, ko_9(152 downto 8), kof(163 downto 7), temp9_out(152 downto
          8));

  RCA_ncr: RCA_comp_ncr
    port map(temp9_out(152 downto 8), reset, ki_11(74 downto 2), ko_9(152 downto 8), temp11(74 downto 2));

  ki_11(74 downto 2) <= ko_11(74 downto 2);

  REG11: ncl_register_D
```

```
      generic map(73, -4)
      port map(temp11(74 downto 2), kof2(74 downto 2), reset, temp11_out(74 downto 2), ko_11(74 downto 2));
-------------------------------------------------------------------------------------------------------------------
   Aout(71 downto 0) <= temp11_out(74 downto 3);
   OV <= temp11_out(2);

   ACK33b: process(ki)
   begin
      for i in 2 to 74 loop
            kof2(i) <= ki;
      end loop;
   end process;

end SYN;
```

# APPENDIX C. FEATURES EXTRACTION CODE FOR SVM

```
function [Fcoarseness] = Coarseness(InputImage)

I = rgb2gray(InputImage);%Converts RGB image to grayscale
[row,column] = size(I);%size of array
G=double(I);

%%-------------------Coarseness--------------------
%initialization

A1=zeros(row,column);
A2=zeros(row,column);
A3=zeros(row,column);
A4=zeros(row,column);
A5=zeros(row,column);
A6=zeros(row,column);
%Sbest for coarseness
Sbest=zeros(row,column);

%Subtracting for Horizontal and Vertical case
E1h=zeros(row,column);E1v=zeros(row,column);
E2h=zeros(row,column);E2v=zeros(row,column);
E3h=zeros(row,column);E3v=zeros(row,column);
E4h=zeros(row,column);E4v=zeros(row,column);
E5h=zeros(row,column);E5v=zeros(row,column);
E6h=zeros(row,column);E6v=zeros(row,column);
flag=0;%To avoid errors

%2x2  E1h and E1v
%subtracting average of neighbouring 2x2 pixels
for x=2:row
   for y=2:column
      A1(x,y)=(sum(sum(G(x-1:x,y-1:y))));
   end
end
for x=2:row-1
   for y=2:column-1
      E1h(x,y) = A1(x+1,y)-A1(x-1,y);
      E1v(x,y) = A1(x,y+1)-A1(x,y-1);
   end
end
E1h=E1h/2^(2*1);
E1v=E1v/2^(2*1);

%4x4  E2h and E2v
if (row<4||column<4)
   flag=1;
end
%subtracting average of neighbouring 4x4 pixels
if(flag==0)
   for x=3:row-1
```

```
      for y=3:column-1
         A2(x,y)=(sum(sum(G(x-2:x+1,y-2:y+1))));
      end
   end
   for x=3:row-2
      for y=3:column-2
         E2h(x,y) = A2(x+2,y)-A2(x-2,y);
         E2v(x,y) = A2(x,y+2)-A2(x,y-2);
      end
   end
end
E2h=E2h/2^(2*2);
E2v=E2v/2^(2*2);

%8x8 E3h and E3v
if (row<8||column<8)
   flag=1;
end
%subtracting average of neighbouring 8x8 pixels
if(flag==0)
   for x=5:row-3
      for y=5:column-3
         A3(x,y)=(sum(sum(G(x-4:x+3,y-4:y+3))));
      end
   end
   for x=5:row-4
      for y=5:column-4
         E3h(x,y) = A3(x+4,y)-A3(x-4,y);
         E3v(x,y) = A3(x,y+4)-A3(x,y-4);
      end
   end
end
E3h=E3h/2^(2*3);
E3v=E3v/2^(2*3);

%16x16 E4h and E4v
if (row<16||column<16)
   flag=1;
end
%subtracting average of neighbouring 16x16 pixels
if(flag==0)
   for x=9:row-7
      for y=9:column-7
         A4(x,y)=(sum(sum(G(x-8:x+7,y-8:y+7))));
      end
   end
   for x=9:row-8
      for y=9:column-8
         E4h(x,y) = A4(x+8,y)-A4(x-8,y);
         E4v(x,y) = A4(x,y+8)-A4(x,y-8);
      end
   end
end
```

```
E4h=E4h/2^(2*4);
E4v=E4v/2^(2*4);

%32x32 E5h and E5v
if (row<32||column<32)
   flag=1;
end
%subtracting average of neighbouring 32x32 pixels
if(flag==0)
   for x=17:row-15
      for y=17:column-15
         A5(x,y)=(sum(sum(G(x-16:x+15,y-16:y+15))));
      end
   end
   for x=17:row-16
      for y=17:column-16
         E5h(x,y) = A5(x+16,y)-A5(x-16,y);
         E5v(x,y) = A5(x,y+16)-A5(x,y-16);
      end
   end
end
E5h=E5h/2^(2*5);
E5v=E5v/2^(2*5);

%64x64 E6h and E6v
if (row<64||column<64)
   flag=1;
end
%subtracting average of neighbouring 64x64 pixels
if(flag==0)
   for x=33:row-31
      for y=33:column-31
         A6(x,y)=(sum(sum(G(x-32:x+31,y-32:y+31))));
      end
   end
   for x=33:row-32
      for y=33:column-32
         E6h(x,y) = A6(x+32,y)-A6(x-32,y);
         E6v(x,y) = A6(x,y+32)-A6(x,y-32);
      end
   end
end
E6h=E6h/2^(2*6);
E6v=E6v/2^(2*6);

%at each point pick best size "Sbest", which gives highest output value
for i=1:row
   for j=1:column
      [maxv,index]=max([abs(E1h(i,j)),abs(E1v(i,j)),abs(E2h(i,j)),abs(E2v(i,j)),...
         abs(E3h(i,j)),abs(E3v(i,j)),abs(E4h(i,j)),abs(E4v(i,j)),abs(E5h(i,j)),...
         abs(E5v(i,j)),abs(E6h(i,j)),abs(E6v(i,j))]);
      k=floor((index+1)/2);
      Sbest(i,j)=2.^k;
```

```
   end
end
%Coarseness Value
Fcoarseness=sum(sum(Sbest))/(row*column);

function [Fcontrast] = Contrast(I)
[row,column] = size(I);%size of array
[counts,graylevels]=imhist(I);%histogram of image
PI=counts/(row*column);
averagevalue=sum(graylevels.*PI);%mean value
u4=sum((graylevels-repmat(averagevalue,[256,1])).^4.*PI);%4th moment about mean
variance=sum((graylevels-repmat(averagevalue,[256,1])).^2.*PI);%variance(2nd moment about mean)
alpha4=u4/variance^2;%kurtosis
%Contrast Value
Fcontrast=sqrt(variance)/alpha4.^(1/4);

function [Flinelikeness] = LineLikeness(InputImage)
I = InputImage(:,:,1);
%imshow(I)
glcm = graycomatrix(I,'Offset',[2 0]);
[row,column] = size(glcm);%size of array

Fn=0;
for i=1:row
   for j=1:column
      Fn= Fn+glcm(i,j)*cos((i-j)*(2*pi/row));
   end
end

Fd=0;
for i=1:row
   for j=1:column
      Fd= Fd+glcm(i,j);
   end
end
Flinelikeness = Fn/Fd;

function [Fdirection] = direction(I)

[r,c] = size(I);%size of array
G=double(I);

PrewittH = [-1 0 1;-1 0 1;-1 0 1];%for measuring horizontal differences
PrewittV = [1 1 1;0 0 0;-1 -1 -1];%for measuring vertical differences

%Applying PerwittH operator
deltaH=zeros(r,c);
for i=2:r-1
   for j=2:c-1
      deltaH(i,j)=sum(sum(G(i-1:i+1,j-1:j+1).*PrewittH));
   end
end
%Modifying borders
```

```
for j=2:c-1
   deltaH(1,j)=G(1,j+1)-G(1,j);
   deltaH(r,j)=G(r,j+1)-G(r,j);
end
for i=1:r
   deltaH(i,1)=G(i,2)-G(i,1);
   deltaH(i,c)=G(i,c)-G(i,c-1);
end

%Applying PerwittV operator
deltaV=zeros(r,c);
for i=2:r-1
   for j=2:c-1
      deltaV(i,j)=sum(sum(G(i-1:i+1,j-1:j+1).*PrewittV));
   end
end
%Modifying borders
for j=1:c
   deltaV(1,j)=G(2,j)-G(1,j);
   deltaV(r,j)=G(r,j)-G(r-1,j);
end
for i=2:r-1
   deltaV(i,1)=G(i+1,1)-G(i,1);
   deltaV(i,c)=G(i+1,c)-G(i,c);
end

%Magnitude
deltaG=(abs(deltaH)+abs(deltaV))/2;

%Local edge direction (0<=theta<pi)
theta=zeros(r,c);
for i=1:r
   for j=1:c
      if (deltaH(i,j)==0)&&(deltaV(i,j)==0)
         theta(i,j)=0;
      elseif deltaH(i,j)==0
         theta(i,j)=pi;
      else
         theta(i,j)=atan(deltaV(i,j)/deltaH(i,j))+pi/2;
      end
   end
end

deltaGt = deltaG(:);
theta1=theta(:);

%Set a Threshold value for delta G
n = 16;
HD = zeros(1,n);
Threshold=12;
counti=0;
for m=0:(n-1)
   countk=0;
```

```
   for k = 1:length(deltaGt)
      if ((deltaGt(k)>=Threshold) && (theta1(k)>=(2*m-1)*pi/(2*n)) && (theta1(k)<(2*m+1)*pi/(2*n)))
         countk=countk+1;
         counti=counti+1;
      end
   end
   HD(m+1) = countk;
end
HDf = HD/counti;

%peakdet function to find peak values
[m p]=peakdet(HDf,0.000005);

Fd=0;
for np = 1:length(m)
   phaiP=m(np)*(pi/n);
   for phi=1:length(HDf)
         Fd=Fd+(phi*(pi/n)-phaiP)^2*HDf(phi);
   end
end
r = (1/n);
Fdirection = 1 - r*np*Fd;

function [r,g,b,Ch] = RGB_exg(rgbImage)

[row, column, ~] = size(rgbImage);
redChannel = im2double(rgbImage(:, :, 1));
greenChannel = im2double(rgbImage(:, :, 2));
blueChannel = im2double(rgbImage(:, :, 3));

for m = 1:1:row
   for n = 1:1:column
      r(m, n) = redChannel(m, n)/(redChannel(m, n) + greenChannel(m, n) + blueChannel(m, n));
   end
end

for m = 1:1:row
   for n = 1:1:column
      g(m, n) = greenChannel(m, n)/(redChannel(m, n) + greenChannel(m, n) + blueChannel(m, n));
   end
end

for m = 1:1:row
   for n = 1:1:column
      b(m, n) = blueChannel(m, n)/(redChannel(m, n) + greenChannel(m, n) + blueChannel(m, n));
   end
end

% allBlack = zeros(size(rgbImage, 1), size(rgbImage, 2), 'uint8');

Ch = 2 * g - r - b;
```