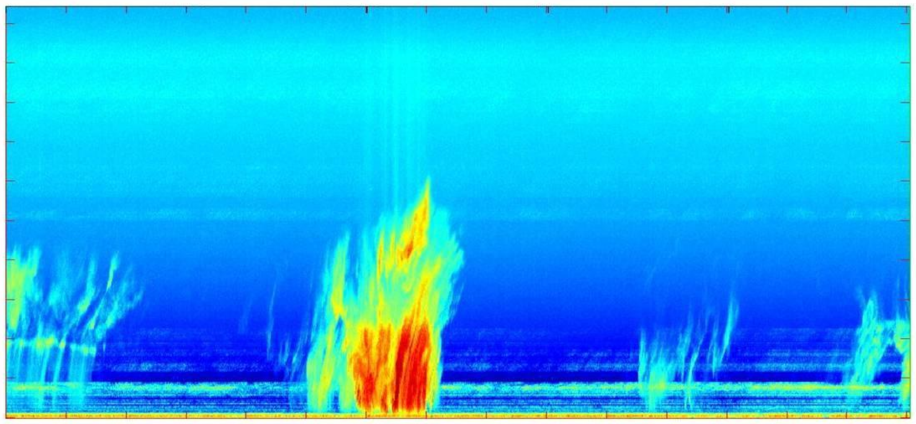
Delft University of Technology

Bachelor Thesis

# Designing for TARA

**Data processing, visualization and storage**



*Supervisors:*
Yann Dufournet
Tobias Otto
Simone Placidi
Christine Unal

*Authors:*
Enzo den Engelsen
Ester Stienstra

June 14, 2011

# Preface

This thesis is part of the Bachelor end project of electrical engineering at Delft University of Technology. With a group of five people we are responsible for the programming of a new system for TARA, which is owned by the ATMOS group. The ATMOS group is part of the *remote sensing of the environment* department of the *electrical engineering, mathematics and computer science* faculty of delft university of technology. Here research is done in order to gain better knowledge about the influence of atmospheric phenomena on climate change.

As subgroup of two (also referred to as group 2) we designed the part that will process the incoming data, visualize the processed data and store collected data on a hard disk. This thesis describes how we have designed this part, which design choices were made and how the system is finally implemented. Group 1 (a group of three people) is responsible for the control of the radar. Their work can be read in the thesis [1]. The subgroups combined are responsible for the main structure of the system and the user interface.

We would like to thank all of the people of the ATMOS group. For giving us the opportunity and helping us with this challenging project.

# Contents

# List of acronyms

| | |
|---|---|
| ADC | Analog-to-Digital Convertor |
| ATMOS | Remote sensing of the atmosphere group, part of the remote sensing of the environment department at Delft U |
| CESAR | Cabauw Experimental Site for Atmospheric Research |
| DDS | Direct Digital Synthesizer |
| FFT | Fast Fourier Transform |
| FM-CW | Frequency Modulated Continuous Wave |
| GUI | Graphical User Interface |
| IRCTR | International Research Centre for Telecommunications and Radar |
| ISA | Industry Standard Architecture computer bus |
| NI | National Instruments |
| PXI | PCI eXtensions for Instrumentation |
| STW | Dutch Scientific Society |
| RADAR | Radio Detection And Ranging |
| TARA | Transportable Atmospheric RAdar |
| VI | Virtual Instrument |
| UTC | Universal Time, Coordinated |

# Summary

The TARA scans through a column of atmosphere in order to observe and study cloud system behavior. After 10 years of operation new knowledge of processing has been acquired which allows for an update that will improve measurement quality and processing speed. The main question of this thesis is: How to implement data processing, visualization and storage for TARA?

The system, from the radar control to the visualization and storage, needs to be built from scratch. In order to most effectively achieve this, the entire system has been split up into two parts that will be designed separately. To be able to combine the parts, the groups will work together to build an overarching structure in which the separate blocks can be easily implemented(chapter 2). In chapter 3 the program of requirements is given, in which all the requirements for the system are mentioned.

Different topics have been researched, in order to be able to do this project, this can be read in chapter 4. Over the years a variety of radar types have been developed, each having its own advantages and disadvantages. Four of the different properties are important in TARA. That are: monostatic vs. bistatic, FM-CW radar vs. pulsed radar, Doppler radar and polarimetric radar. Frequency modulation is used to be able to determine the range of the measured objects. By comparing the received frequency with the current transmitted frequency the time that was needed for the signal to travel can be determined, also the Doppler speed of the particles can be calculated. For the implementation of this project we will work with LabVIEW and Matlab.

An overall block diagram has been made, after which the more detailed block diagrams and state diagrams of the three blocks that group 2 has to design, could be made. The overall block diagram shows the signals that every block in the systems sends out or receives (chapter 5). The final block diagrams are shown in Appendix A and the state diagrams in Appendix B. The state diagrams are used to describe how the sub-systems accomplish their tasks. Incoming data is processed by Matlab code which is provided. This is one of the main reasons for the choice of LabVIEW, because Matlab code can be implemented in the program via Matlab nodes. Variables that should not be changed during measurement and should be used by multiple different systems are put into global functional variables. Variables that can be changed are sent via notifiers.

After everything was designed and some choices for implementation were made, the implementation could begin (chapter 6). First an overarching system was made, but during the development it became apparent that the actions of this system could be interpreted as those of the Radar Control block, a block that was originally designed by group 1. After the choices were made on how the Radar Control operates, the design process of the GUI started and the data processing and visualization block were implemented. Implementation of the storage block has however not been realized, because of time constraints and task re-allocation between the two sub-groups.

Finally the system has been implemented in the actual TARA.First real-time measurements have been succesfully performed. An evaluation about what has been achieved compared to the requirements discussed in chapter 3, can be read in chapter 7. As already stated before, the storage has not been implemented, so requirements considering storage have not been met. Besides that, most requirements are met, although improvements can still be made. In chapter 8 a discussion is given about the points on which the project could be improved. In chapter 9 the conclusion is drawn that most of the requirements are met and recommendations are given on how to improve the system further.

Chapter 1

# Introduction

The TARA scans trough a column of atmosphere in order to observe and study cloud system behavior. TARA is owned by the ATMOS group, which is tasked to do research in the field of cloud systems and how to observe them using radar and satellite observations. They do this in cooperation with other institutes in the CESAR project. CESARs goal is to find out what the effects are that different types of weather have on the climate. The ATMOS group owns two different radars to conduct their research.

TARA has been developed 10 years ago, after 10 years of operation new knowledge of processing has been acquired which allows for an update that will improve measurement and processing speed. The current system will be replaced with a data-acquisition system which primarily uses signals in the digital domain rather than the analogue which is currently the case. Also, the two computers used at the moment will be replaced with one single computer, which makes synchronization a lot easier.

The main question of this thesis is: How to implement data processing, visualization and storage for TARA. The project started framing up the programe of requirements, after which a small study on the current state of the art technology was done. Followed by the designing of the system, which took most of the time. After the design was finished, an implementation was made and tested on the actual radarsystem.The visualization and storage should happen in the way that is usual, so everyone can easily understand it. The dataprocessing code will be provided by the ATMOS group, and is writen in Matlab.

This thesis is constructed in the following way. The problem with the current state of the TARA radar is described in chapter 2. The program of requirements follows in chapter 3, after which an overview of state of the art systems is given in chapter 4. The process of designing the system and relevant subblocks will then be explained in chapter 5. After that the final implementation is described in chapter 6 and the evaluation of the system in chapter 7. Afterwards there will be a discussion in chapter 8, followed by the conclusion in chapter 9.

# Chapter 2

# Problem description

Knowledge on the field of data-acquisition and processing is constantly improving, so it's no surprise that the ten year old systems in TARA are starting to show their age. The radar will therefore be upgraded with new hardware, which will then need to be programmed in order to work as intended.

This chapter describes what needs to be implemented so that the software will accurately process the data the radar receives, visualize the processed data in a useful manner and store the processed data, the visualizations and the raw data. In the first section the problem is described, after that in the seccond section the objectives of the project are named. And finally in the third section the framework of the project is given.

## 2.1 Problem definition

The system, from the radar control to the visualization and storage, needs to be built from scratch. In order to most effectively achieve this, the entire system has been split up into two parts that will be designed separately. To be able to combine the parts, the groups will work together to build an overarching structure in which the separate blocks can be easily implemented. At the end of the project the system should be able to execute and control all processing activities necessary for taking measurements and studying the results, without using any part of the old system.

## 2.2 Objectives

A main block-diagram, where the separate blocks can be easily implemented, needs to be designed by the two groups together and will serve as the back-bone of the project. In this diagram all the necessary signals and communication between the sub-blocks will be clearly defined so that there will be no miscommunication between the two groups. When this has all been done, the sub-group will start to work on its sub-blocks where the data-processing, visualization and storage aspects will be developed into individual blocks.

The processing block needs to process the raw data it receives from the radar control unit which is built by the other group[1]. The block will also receive information on the measurement that is currently running which it will receive from the control block and the header. A noise measurement needs to be done first, which then needs to be processed into noise data that can be used to filter out the noise that will be present in the measurement data. This information will also be provided to the user. After the noise measurement is done the real measurements start and the data processor will process the data it receives so that conclusions can be drawn and visualizations can be made. Because of the large amount of data coming from the radar control unit, the processing block may end up struggling with processing all the data if the user has opted to record raw data as well which will consequentially slow down the entire system. Therefore it needs to be able to skip data packets so that none of the raw data that is recorded will be lost, the processing can then also be done afterwards.

The visualization will be done by the visualization block which will produce a real-time visualization of fifteen different variables, clustered in five groups, two for the polarimetric measurement types and three for the combined polarimetric wind measurement types. In order to do this the visualization block receives the processed data from the processing block, which also provides information about the scaling of the graphs for the visualization. The data will be displayed on 2D graphs in which different colors will provide information about the atmospheric particles, as a function of time and height. The visualization block will also generate a quicklook file, which will be stored so that the user can see what was measured at a given moment in the past.

Finally the processed data and the quicklooks need to be stored on a hard disk which will be done by the storage block. This block should put all the data it receives in a buffer during the measurement and when the measurement is stopped or the day is over, the information needs to be saved to files on the hard disk. The storage block should make sure that the files are saved in the right directory and that the files and folders are named correctly. There are four types of files that need to be saved which are the noise data files, the processed data files, the raw data files and the quicklooks. The user can indicate which type of measurement needs to be done and saved on the user interface. For each file type, except for the quicklooks, different packages of data have to be created and stored into files. The storage block needs to make sure that the different types of packages are put in the correct buffers. After the measurement has been stopped or the day has ended, the block should take care of saving the different files to the right location, one at the time.

## 2.3 Framework of the project

As with every design there are limitations present, with the biggest limit being the short period of time that is given until the project must be completed. There are only seven weeks for both the design and implementation of the software. Because of this there is a need for rapid prototyping, because at the end of the 7 weeks there should at least be a proof of concept. Furthermore, the ATMOS group will provide Matlab code that is to be used for the processing block within the design, which means that the software for the radar needs to be created in a programming language that can understand and execute Matlab code.

# Chapter 3

# Program of requirements

Modern computers are capable of doing complex calculations even in parallel at a much higher rate than the current dedicated hardware in the radar. This allows for an update to improve measurement speeds as well as processing speeds. To implement this update, a computer that is connected to a data aquisition card is used together with a DDS. This computer needs to be programmed in such a way, that it can replace the function of the existing dedicated hardware and has a design to allow for new functions to be implemented easily.

In this program of requirements all the requirements for the data processing, visualisation and storage part of the system are mentioned. This program has been realized after close consultation with the future users of the system. The system will be evaluated following all the requirements mentioned afterwards.

First in section 3.1 the requirements for the use of the product will be given, after that in section 3.2 requirements regarding environment and ecology are given. In section 3.3 requirements for the design will be given and finaly in section 3.4 requirements considering production will be named. In a program of requirements martketing and company strategies are also mentioned, but because this product is not a commercial one, these requirements are not given here.

## 3.1 Product use

1. The system has to process the acquired radar data real-time.
2. The system has to visualize the processed data real-time.
3. Graphs should automatically be scaled correctly.
4. The user should be able to select the radar measurements he/she wants to monitor.
5. The radar parameters should be plotted as function of UTC time and height.
6. The unit of the radar parameters has to be mentioned
7. The important measurement specifications have to be mentioned for the user to check whether his/her measurement is the right one.
8. The raw data has to be stored on a hard disk.
9. The processed data has to be stored on a hard disk.
10. The quicklooks have to be stored on a hard disk, in order to allow the user to see quickly what was measured at a given moment in the past.
11. The system has to add a header, consisting of measurement specifications, to all the saved data.

12. Files have to be saved in the atmospheric data standard format (Netcdf).
13. The quicklooks have to be saved in a standard format.

## 3.2 Environment and ecology

1. The software has to run on a normal PC under the Windows operating system.
2. The software should run stable without causing any instability to the PC running it.
3. The system should be user friendly.

## 3.3 System design

1. Usage features
   (a) The user should be able to update the code easily.
   (b) The code written by the user containing the processing algorithms, is in Matlab and should be easy to implement.
   (c) The noise processing should be done in 1 minute in order to make parallel computing easier.
   (d) The processing block should provide the correct scales for the visualisation.
   (e) The visualisation block should temporarely save all the visualization data in a buffer and make a quicklook file at the end of the measurement or day.
   (f) The storage block should automaticaly store files in the correct directory and create this directory if it does not exist already.
   (g) The storage block should put all the nessesary data in a file automatically.
   (h) The user should be able to choose which data should be stored.
   (i) The system should be designed using an open approach, allowing for easy implementation of new functions and simple debugging.
2. Production and commissioning
   (a) The software should be modular so that it is easily adjustable to other systems.
   (b) The software should be well-documented.
   (c) After installation the software shoul immediately be ready for use.
   (d) The system should be operational before the 7th of June 2011.
   (e) The install process should be done within 60 minutes.
3. Liquidation
   (a) The un-install process should be done within 60 minutes.

## 3.4 Production

1. The user interface as well as the processing software will be built in a standard software package.
2. Pieces of Matlab code should be inserted in the software.
3. The PXI system of National Instruments should be used for data collection.

# Chapter 4

# Review of current systems

In order to be able to design the system, knowledge had to be gained about the topic first. That is why the different topics have been researched. This chapter describes the different atmospheric radars and specifically the TARA. Furthermore, the tools that will be used are shortly discussed.

First in section 4.1 an overview of atmospheric radars is given, then in section 4.2 the TARA is described and a short description of calculations is given. In section 4.3 the data-acquisition is discussed, followed by a description of the programming language LabVIEW in section 4.4. Finally in section 4.5 a short description of Matlab is given.

## 4.1 Atmospheric radars

Over the past few decades climate change has become a hot topic. The earth and its climate is a complex system. With a lot of variables of which for some of them the knowledge is lacking. One of these variables are clouds and their behavior. When the earth gets warmer it is likely that more water out of the oceans evaporates and more clouds will develop. At the moment there is not much known about the influence of clouds. Climate models are more or less just guessing on the effects, but most of them are probably underestimating this effect[2]. In order to gain knowledge on cloud systems, it is necessarry to be able to measure them. The most common way to measure clouds is with radar (radio detection and ranging) techniques.

A radar sends out a radio signal, this signal is reflected by any object in the sky (including cloud particles, raindrops, insects, birds and aircrafts). From the signal that is reflected the power is detected by the radar. Through signal processing there can be determined what the range of the object is, what size it is and how large its speed is, from the Doppler velocity. Radars have been developed for a range of applications, but in this thesis only atmospheric radars will be discussed.

Over the years a variety of radar types have been developed, each having its own advantages and disadvantages. Four of the different properties are important in TARA, and therefore discussed below. These properties are: monostatic vs. bistatic, FM-CW radar vs. pulsed radar, Doppler radar and polarimetric radar.

The difference between a monostatic and a bistatic radar is the number of antennas. The most common monostatic radar uses one antenna for both transmitting and receiving, a bistatic radar on the other hand uses two antennas, one only for transmitting and one only for receiving. The antennas for a bistatic radar can be standing side by side, but they also can be placed at different locations. Of course, in the last case it is necessary to aim both of the antennas at the same volume of air[3].

Most of the radars used nowadays are pulse radars. This means that the radar transmits a pulsed signal and then waits for the signal to return. With the time that that takes the distance of the object can be computed. Another type of radar is the continuous wave radar, this radar continuously transmits and at the same time receives a signal. As expected this is only possible with a bistatic radar. Also a little bit more effort is needed in order to be able to tell the distance of the target. Most of the time this is done by frequency modulation. If this is the case, the radar is called a FM-CW radar. By changing the frequency of the transmitted signal, the time between transmission and reception can be derived from the signal. How this is done will be discussed at the end of the next section.

A very useful radar is the Doppler radar. This radar uses the effect, discovered by Doppler, that a moving source of sound causes a frequency shift. This shift is directly proportional to the speed of the source. For radio waves the same effect is present. And the effect works exactly the same if the source is not moving, but the object is. So the velocity of the object can be calculated when the transmitted and received frequency is compared. This property can be very useful when the behavior of particles inside a cloud system are studied.

The final type is polarimetric radar. In this radar the transmitted electric field can oscillate horizontally and vertically, the user can choose which one to use at any point. Also, the receiving antenna can receive horizontally and vertically, all combinations of sending and receiving can be made if wanted. If the reflected electric field is measured in a different direction than it has been sent the shape of an object can be determined.

## 4.2 TARA

TARA is custom built by the ATMOS department, and became operational in April 2000 [4]. TARA is a FM-CW Doppler polarimetric radar, which scans trough a column of atmosphere. The radar has two antennas of 3 meters in diameter, each antenna is equiped with three beams, a main beam and two offset beams that each are 15 degrees off axis, see figure 4.1. The main beam is fully polarimetric, in other words it can transmit/receive both horizontally and vertically. The offset beams are used to measure wind and are single polarized[5]. Combining the different polarizations there are 6 types of measurement possible, 4 using the main beams, and two using offset beam 1 and 2. The 4 main beam measurement types are: HH, HV, VH and VV using the mean beam in different polarizations, where the fist letter expresses the receiving polarization and the second expresses the transmitting polarization. A measurement consists of a cycle of differtend types after each other.

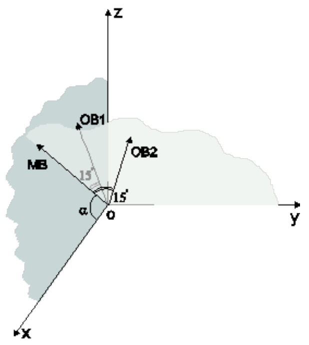


Figure 4.1: Offsetbeams of TARA

### History

The Project for designing and building the TARA system started in the beginning of 1995 at the International Research Center for Telecommunications-transmission and Radar, IRCTR. The radar became operational in April 2000. This five year project was financed by the STW[6] and the final system was to be a transportable atmospheric profiler that could be used for the study of the static and dynamic behavior of atmospheric phenomena at different geological locations.[7] Real-time processing was originally done on a dedicated DSP-based computer system [8]. This processing included filtering, noise correction, clutter suppression and statistical quantification of the Doppler spectrum. Displaying of the measurement and the storage of the data are both done in real-time.

In 2008 the performance of the Plessey ICs responsible for the data processing had deteriorated to such an extent that this unit needed to be replaced. Also there was a need for some maintenance to keep the system running. Because of the fact that FFT algorithms were now commonly available in C and because of the increased processing power of the common PC, the need for custom made data processor was no longer there. So the choice was made to implement the data processing on a PC.

Because the computer used for the data processing did not have a slot for a ISA card, another computer was needed for the user input and radar control. That is the reason why the TARA is currently controlled by two PCs. In order to provide the connectivity from the original ADC to the PC a new I/O device was needed. For this functionality a PCI6534 was ordered from National instruments. This change in configuration was more of a patch-up to keep the system operational until a proper upgrade could be realized.

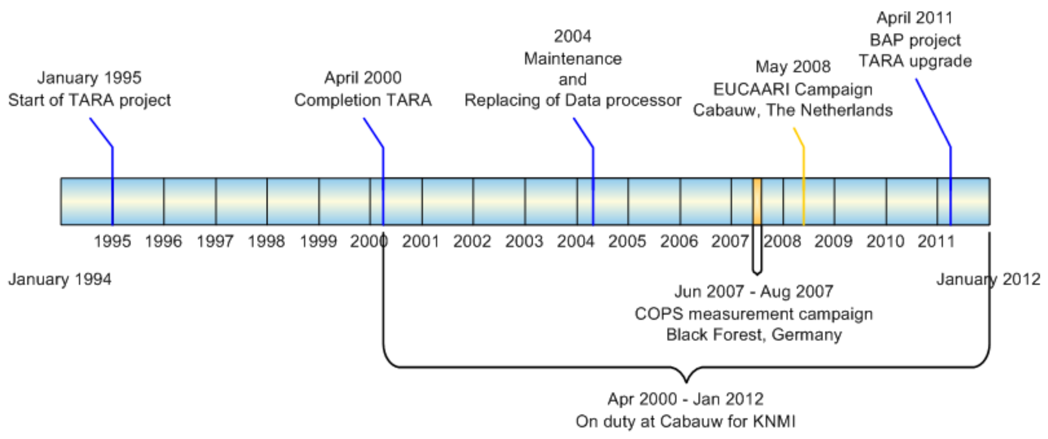


Figure 4.2: Timeline of TARA

## Data processing

Frequency modulation is used to be able to determine the range of the measured objects. Each time a sweep is transmitted, which consists of a sine with a frequency that is either a ramp up or a ramp down. By comparing the received frequency with the current transmitted frequency the time that was needed for the signal to go away and come back can be determined. To be able to compare the two signals, the signals are mixed with as result the beat signal. From that beat signal the frequency is determined by using the Fourier transformation, the frequency of the beat signal is exactly the difference of the frequency of the two signals. Since it is known that the speed of radio waves equals the speed of light, calculations can be made to determine the range. The basic formula used for that is:

$$R = \frac{cT_s f_b}{2B_{sweep}} \tag{4.1}$$

where $R$ represents the range in meters, $c$ is the speed of light, $T_s$ is the time that one sweep takes, also known as sweep time, $f_b$ is the beat frequency and $B_{sweep}$ is the bandwidth of the sweep.

Also the TARA is able to perform Doppler measurements. By this the velocity of particles can be determined. The shift of the Doppler frequency is assumed to be a lot smaller than the effect of the fm modulation so that formula (4.1) still holds. Since the frequency shift is too small to use, the change of the phase of the received signal in time, at a fixed range $R_o$ is used:

$$\psi(t) = \frac{2\pi}{\lambda}(2R(t)) = \frac{4\pi}{\lambda}(R_o + v_r t) \tag{4.2}$$

with $v_r$ being the radial speed of the particle, $t$ being the time and $\lambda$ the radar wavelength. Because $R_o$ and $t$ are known, $v_r$ can be calculated. Of course the atmosphere contains a lot of particles which makes the calculations more complicated than described above, but the basic formulas are still used.

As stated before, in order to know the frequency a Fourier transform has to be done. Since the signal is a digital signal, this is a discrete Fourier transform. With this transform a transformation from a digital sequence into a digital sequence is done. The transformation is almost the same, with the biggest difference that the integral is replaced by a summation, in order to be able to input a discrete signal. The formula of the discrete Fourier transform is given by[9]:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N}, \qquad k = 0, 1, 2, \ldots, N-1 \tag{4.3}$$

The outcome of this formula is a sequence representing the frequency domain of the signal. With this information the calculations in formulas (4.1) and (4.2) can be done. In order to compute the discrete Fourier transformation on a computer, a variety of algorithms have been developed known as FFT algorithms. Matlab has developed its own FFT algorithm[10], which is easy to use by using the command x = fft(y).

A measurement is always started with a noise measurement. From this noise measurement two important variables are calculated, the mean of the noise and a calibration factor. The mean of the noise is later used to filter out noise from the measurement. Roughly speaking all the signals that are lower than the mean of the noise are removed. The calibration factor is used to convert the processed data from power to reflectivity values. This factor is calculated from the ratio between the average noise measurement and the computed theoretical noise reflectivity[11].

## 4.3 PXI System

Technology is going fast, so a new data-acquisition system will be a big improvement of the capabilities of TARA. The system should be able to acquire the data and perform some preprocessing tasks. The choice has been made to use the PXI system from NI. This choice was mainly based on the fact that the department already had a good relationship with NI and that NI offers good quality of service. Also this PXI system is modular, which means that new modules can be added later on.

## 4.4 Labview

When developing a software program, first of all a programming language has to be chosen. This language should be easy to learn, because of the time linitation, and should be able to communicate with the PXI system. NI has developed its own programming language to accompany their products, which means that it is relatively easy to use that language together with the PXI system. This language is LabVIEW which is a graphical programming language.

In this language programming is done by selecting blocks with the correct functionality and connecting those blocks with wires. For most engineers this is more intuitive than a text based programming language. One of the problems faced with LabVIEW is that there was almost no experience with it at the beginning of the project, by both the project group and the department. This implies that a lot of time was spent on learning how to deal with the language instead of implementing the desingns. An advantage of LabVIEW is that Matlab code can be included in the project, via a Matlab script node, this was very useful because Matlab code for the calculations was provided.

## 4.5 Matlab

To process the data coming from the PXI system, a lot of calculations have to be done. LabVIEW mostly deals with the dataflow from one block to another. Small and simple calculations like adding or multiplying two numbers can easily be done in LabVIEW, but bigger calculations become a problem very quickly. This is why LabVIEW is offering the opportunity to implement Matlab code in a special designed block.

For TARA the signal processing is already done in Matlab and this code will also be used in the new design. Matlab is a high-level programming language specialized in making calculations with matrices. Matlab does not have a big influence on the design process, because the code is already written. The only thing it has influenced is the choice for LabVIEW because of the fact that it can be implemented in LabVIEW.

# Chapter 5

# Design process

This chapter describes the design process of the project. First, the main block diagram of the system is presented in Section 5.1 in order to put the sub system designs handled by group 2 into context. In section 5.2, the state diagrams of the Data Processor, Visualization and Storage blocks are exposed in details, (those diagrams being the main design task of group 2). In Section 5.3 the choice for LabVIEW is explained and a small explanation of the provided Matlab codes is given. Finally, each of the design choices made during the implementation in LabVIEW are motivated and commented in Section 5.4.

## 5.1 Blockdiagram and signals

In order to obtain an overview of all the sub systems, a top-down design approach was chosen for the system, started by discussing an overall block diagram. This was necessary because the work was divided between two groups and the overall block diagram with carefully made signal definitions ensured that the sub-blocks, although designed and implemented independently from each other, would work together.

The overall block diagram has gone through a number of iterations involving different choices motivated along with the state of progress of the system requirements. The final version can be found in Appendix A. It consists of the following seven blocks:

### GUI

The Graphical User Interface is made to allow the user to define the desired type of measurement via an intuitive and easy to use interface (the front panel of the user interface is displayed in Appendix C). The GUI contains:

- A switch to start or stop the measurement
- Input boxes for all the variables the user needs to set for the measurement
- Tabs to switch between normal and advanced measurement options
- Check boxes to choose which data the user wishes to save
- Input boxes to set a measurement schedule
- A text box to input user comments
- 15 graphs displaying the processed data parameters as a function of height and time
- 5 tabs grouping the 15 graphs by parameters for the main beam, the 2 offset beams, the polarimetric parameters and the wind parameters

- A text box displaying the system status/errors
- The GPS location and time

All of the measurement settings are sent to the Header Builder and the signal from the on/off switch and the measurement schedule are sent to the Radar Control block.

### Header Builder

The Header Builder takes all of the measurement settings that are set in the GUI and bundles them together so they can be easily transmitted. All variables present in the Header Builder block are fixed and set only once before the measurement. Therefore, they can easily be transferred to the other blocks.

The Radar Control needs the measurement settings to tell the PXI and consequentially the radar what type of measurement must be done. The Data Processor needs them in order to process the beat signal correctly, the visualization needs to know what exactly to display (the processed 3 or 5 cycle measurement data) and the Storage block needs to store the measurement settings together with the data that is to be saved.

### Radar Control

The Radar Control block is designed to control and receive signals from the radar itself and start each of the other blocks when they are needed. Also, through the connection with the radar connector panel, it is able to send signals to the PXI which then sends the signals to the radar. By doing this, the Radar Control block can instruct the PXI to create the type of beat signal that is required for the noise measurement or the 3, 5 or customized cycle measurement.

### Data Processor

The task of the Data Processor is to process the beat signal in order to extract atmospheric echoes that are received by the radar, so they can be stored and displayed. The processing is only performed in case of specific 3 cycle and 5 cycle measurement configurations. If the beat signal is of a custom measurement configuration, it will not be processed and only raw data will be stored. The detailed block diagram of the Data Processor can be found in Appendix A.

The signals that enter the Data Processor are the header variables and the beat signal. The header variables are used to process the raw data using the correct parameters. The beat signal contains either 'noise data' or '3 or 5 cycle measurement data' and is sent to either *noise processing* or *3 or 5 cycle measurement processing* depending on which type of data it contains.

The noise processing will provide 2 sets of variables of which the first is the mean noise and the second is a calibration constant. These 2 variables will be used in the upcoming 3 or 5 cycle polarimetric measurement processing and also sent to the Storage block as Noise Data. The '3 or 5 cycle block' as the name implies checks whether the beat signal is of a 3 cycle polarimetric measurement or a 5 cycle polarimetric measurement. The beat signal is then sent to either the 'Pola rt block' if the measurement is of the 3 cycle variant or to the 'Polawind_rt block' if the measurement is of the 5 cycle variant. These blocks will process the beat signal and provide up to 15 processed radar spectral polarimetric parameters that will be output as *processed data.* This processed data that will be sent to the output of the block so it can be sent to Storage to save the data and to Visualization to display it in graphs.

The sub-blocks of the Data Processor will also send a string with the code version along with the data. This allows the user to easily check which Matlab code was used the processing of the data that is being viewed.

## Visualization

The Visualization block has to create packets of visualized data which the GUI can interpret and display in graphs. It also needs to create the quicklooks that need to be saved. The detailed block diagram of the Visualization can be found in Appendix A.

The block receives the header variables and the processed data as inputs. The processed data is sent to the 'scale adapting block' and the '3 or 5 cycle block'. The 'scale adapting block' checks the minimum and maximum height that is contained within the processed data packet and determines how to scale the y-axis of the graphs according to those values and sends them to the 'real time visualization blocks'. The '3 or 5 cycle block' checks of which measurement cycle the processed data is and sends it to the correct real time visualization. Both the real time visualization blocks send their signal to the next '3 or 5 cycle block' that checks again of which measurement cycle the data is and then outputs it.

The processed radar spectral polarimetric parameters coming out of Data Processor block combined with the scale adaptation for the graphs are then sent out as a visualization packet to the GUI. Each of the processed parameters is range and time dependent (represented as 2D matrices). They are therefore visualized on a 2D graph (time and range) in the GUI.

## Storage

The Storage block needs to store the data so it can easily be found back and extracted. The detailed block diagram of the Storage can be found in Appendix A. The noise data, processed data, beat signal and the quicklooks can all be sent to this block along with the header information depending on what the user desires. For the sake of clarity and in order to easily trace back the measurement specification, the header data are always combined together with the different data stored.

There should be 4 types of data packages that can be stored:

- A noise package, which consists of the header, processed noise data from a measurement of 2 minutes and 1 minute of raw data.
- A transmission package, which consists of the header and the processed data.
- A raw data package, which consists of the header and raw data.
- A quicklook package

To be able to different packages at the same time, several buffers are considered to be used in parallel after the merging of the required data has been done. When the buffers are full, the data should be written in NetCDF format. Finally, the data should be stored into several package dedicated folders present on the hard disk. These folder should also be placed within daily classified directories. This folder classification tree will be handled and created by the Storage block itself.

The Storage block also has its own control block. This block will be notified if the day is at an end, so that new file for a new day of measurements can be created. It will also receive a signal which indicates what packages the user wants to save as well as a signal that tells the Storage block that the measurement has been stopped and it needs to save the data.

## System Control

The System Control block is there to receive and process errors. When one of the sub-blocks of the system runs into problems it will output an error to the System Control block. This block will then output the error to the user through the GUI and in the worst case it will send a signal to the Radar Connector Panel to shut down the radar.

## 5.2 Statediagrams

State diagrams are used to describe how the sub-systems would accomplish their tasks. This gives a good indication of what specific events are required for the sub-system to transition to its next task. The state diagrams of the 3 different subsystems detailed in this section can be found in Appendix B.

### Data Processor State Diagram

When the Data Processor is started it needs to do nothing as long as it has not received a beat signal. Therefore an *idle state* is created in which it will wait for a beat signal and only after it is received, it will advance to the next state.

The next thing the Data Processor needs to do after a measurement cycle is started, is to take a beat signal containing noise data and process it be obtain noise data that can be used to filter out the noise in the measurement itself. This will be done in the *noise processing state*.

After the processed noise data is obtained, the Data Processor will need to either start processing a 3 cycle measurement, a 5 cycle measurement or return to doing nothing depending on whether the user has selected to do a 3 or 5 cycle measurement or just store raw data. Two states are added in order to do this. Both a *3 cycle signal processing state* and a *5 cycle signal processing state* to do the respective processing. The Data Processor needs to keep processing data whenever a new beat signal is received, so it needs to remain in its current state (be it the *idle, 3 cycle signal processing* or *5 cycle signal processing state*) until the measurement is stopped.

### Visualization State Diagram

The Visualization part is started alongside the Data Processor and needs to wait until it receives processed data it can display. So it remains in an *idle state* until it receives the required data.

After it receives the data it needs to scale all the graphs using information from the data packet so the processed data will be displayed correctly. When this is done the Visualization part needs to check whether a 3 or 5 cycle measurement needs to be on the GUI. There is a *scale graphs and check cycle state* created for that, because it needs to be done before anything is visualized.

When the graphs are set up to visualize the processed data it will move to the *3 cycle visualization and create quicklook state* or the *5 cycle visualization and create quicklook state* depending on which measurement cycle was selected by the user. In these states the Visualization will send data packages containing the data to be visualized to the GUI and because all the visualization data is already in the Visualization block these states will also create the quicklooks that need to be stored. As with the Data Processor it needs to keep visualizing as long as processed data is being provided, so it will stay in the same state until the measurement stops after which it will return to the idle state.

### Storage State Diagram

As soon as the Data Processor and Visualization part are started, the Storage part will start waiting for data it needs to store. When it receives data it needs to put it into a buffer and keep adding to that buffer until it is full. The Storage part will stay in the *wait for data state* until it receives the data, it will then move to the *put data in buffer state* where it will then send the data to a buffer and return to the wait for data state if buffer is not full.

When the buffer is full the data it has gathered needs to be stored and the buffer needs to be emptied for the next sets of data. The storage of the files needs to be in the netCDF format, which will be created in the *create netCDF / clear buffer state.* The buffer will also be emptied in this state.

The Storage part will then need to check if the directory and file it wishes to save to have already been created and then either add the data to the file and directory if they exist or create them and then add the data. This is done in the *check directory, create directory, check file, create file and add data to file state.* After the data has been added, the Storage part will return to wait for new data.

## 5.3 Matlab and LabVIEW

Important factors to consider when choosing a programming language are programmer productivity, maintainability, efficiency, portability, tool support and hardware and software interfacing. [12] The different programming languages that were considered are: C, Matlab and LabVIEW. These languages all fulfill the requirement for software interfacing, in that they can communicate with Matlab.

The choice for LabVIEW was made, because the programming language allows for rapid prototyping, where everyone can design their own independent parts and easily combine them, allowing full design flexibility. It is already capable of interfacing with the PXI, so the only thing that needed to be learnt was how to use the elements which interface with the PXI in LabVIEW. Another benefit of LabVIEW is that it is a graphical programming language, which makes it easier to understand when beginning to work with the language without prior experience. LabVIEW also makes it possible to run different sub systems in parallel, which is required to make real-time visualization possible. These sub systems are defined as Virtual Instruments (VI) in LabVIEW. These VIs are similar to user made functions in other programming languages, in that they can be called to execute specific parts of the code.

One of the main benefits is that LabVIEW allows for easy implementation of Matlab codes in the program. These can just be run via so called Matlab nodes, where data input from LabVIEW, are used as variables for the function you call within Matlab. The outputs from Matlab are then converted to LabVIEW so it can be used in the LabVIEW environment. In order to process the raw data coming from the radar, Matlab code is provided that needs to be implemented in the program. This unique design (including Matlab code in LabVIEW environment) provides great flexibilities that are required for continuous improvement and keeping the current system as a state of the art system.

The Matlab functions that need to be called are: 'Noiseprocessor_rt', 'Pola_rt' and 'Polawind_rt'. These functions will call other functions for their calculations, but only these 3 need to be called in LabVIEW. Examples of the Matlab code provided can be found in Appendix E.

The Noiseprocessor_rt function takes a (raw) beat signal containing 2 minutes of noise measurement data arranged as a 2D array and specific header variables needed for the calculations the function needs to perform. It then performs the processing on the beat signal and outputs 2 variables (both 2D arrays) containing the mean noise and a calibration factor for use in the 3 and 5 cycle polarimetric measurement processors and a string that indicates which version of the Matlab code was used (when reviewing the processed data it is desired to know which version of the code was used for the calculations).

The Pola_rt function takes a beat signal containing 2 profiles of 3 cycle polarimetric measurement data arranged as a 2D array so it can be processed. In order to process the signal, it makes use of the 2 noise variables output by the Noiseprocessor_rt function and specific header variables. It outputs 6 radar spectral polarimetric parameters, the height that corresponds to the data and the version of the Matlab code.

The Polawind_rt function works similar to the Pola_rt function, but is used for 5 cycle polarimetric measurement data. It uses an additional variable from the header and outputs an additional 9 radar spectral polarimetric parameters, making a total 15 parameters plus the height that corresponds to the data and the version of the Matlab code.

## 5.4 Choice motivations

Because the Data Processor, Visualization and Storage all need to run in parallel, a way to deliver signals to constantly running subVIs needed to be found. The main problem with parallel tasking is a timing issue which arise from the non ability of the subVIs to handle new sets of data while they are still busy running with previous ones. Solutions for this problem are to use *global variables, functional global variables* or *notifiers.*

Global variables are elements within LabVIEW that can be used to access and pass data between several VIs that run simultaneously. They are however subject to race and performance issues. The race issue occurs when two different VIs try to write to the same global variable, only one of the messages will be added.

Meaning that one of the messages will be lost. The performance issues occur when a large amount of data is present on the global variable and it is called by a lot of VIs. Every time a VI calls the global variable, a new copy of the data will be put on the wire and sent to the VI. So when the global variable is put in a loop, every iteration it will retrieve a new copy of the data, which will slow down the program. [13]

Functional global variables are similar to global variables with respect to their ability to access and pass data between simultaneously running VIs. The main difference between the two is that functional global variables need to be called with either an 'initialization' or a 'get' signal. When the functional global variable is called with the initialization signal, it takes the variables that the user wants to pass between the VIs and stores it in the functional global variable. When it is called with the get signal, it takes the data that has been stored before with the initialization signal and outputs it, so it can be used in the VI that calls it. This prevents race issues with global variables, where two VIs would write to it at the same time and only one of the two would have their data added to the global variable. Functional global variables also prevent the slowdown caused by global variables when large amounts of data are retrieved every time the global variable is called.

Notifiers work quite differently. Similar to global variables and functional global variables, they can be used to access and pass data between simultaneously running VIs. However, instead of simply calling the function, the notifier is obtained before a while loop and released after the while loop has stopped. Within the while loop the notifier will be constantly available and updated with the latest variables that have been put on there by another VI that is running at the same time.

The main advantage of using notifiers is that a loop within a VI can be paused and made to wait for the data on the notifier to update. This makes it easy to run a loop within a VI only when new data is available. The drawback of using notifiers is, that in order to not lose any data, the rate at which the notifications are sent and the rate at which they are received needs to be synchronized. The notifiers can also not be buffered and the data is therefore lost if it is overwritten before the VI has been able to take it out. [14]

### The choice for global functional variables in the design

The variables in the Header Builder need to be accessed by a lot of the VIs, so the choice to put the variables on a functional global variable was made. The race issue is not present here, because only one VI sets the data for the Header Builder, but the performance of global functional variables is better than the performance of global variables.

A global functional variable called "GUI refnum select" is also used in the VI for the GUI. All the references to the properties of the graphs that display the measurement data are bundled and the references to the header are bundled and put on the global functional variable. The references can then be used in the two different while loops in the GUI. This allows the GUI to clear the graphs in the while loop that waits for the start of the transmission and at the same time to constantly update the graphs in the while loop that waits for a visualization packet for the GUI.

### The choice for notifiers in the design

All the variables that can be changed during system operation are sent and received via notifiers. This gives the possibility to pause VIs pending on the data they need to continue, which also means that the VIs within the system will be automatically synchronized. For example, when the Data Processor needs a beat signal to process, it will wait for the PXI to send it and before the Visualization can start creating visualization packets for the GUI, it will wait for the processed data to be received from the Data Processor. Every VI will operate as soon as the data is available and wait for the next set of data after it's done.

All of the notifiers are obtained in the Radar Control VI and then wired to the inputs of the other VIs that use them. This means they are only obtained once and can all be released in the same VI. Now if a "wait for notification" element in one of the subVIs is waiting when the notifier it belongs to is released, it will generate an error within that element. This error signal can then be used in order to shut down a subVI.

This decision is a response to the problem that arises when a subVI that creates and releases a notifier within itself, it can end up getting stuck on the "wait for notification" element. This happens when this element is waiting for a notifier that is supposed to be sent by another VI, but the VI sending the notifier has already been shut down. The loop in which the wait for notification element is present will then continue running causing the VI to never shut down, which will result in system instability or data corruption.

A LabVIEW control file was created to store the names of the different notifiers that are called in the entire program. This prevents spelling errors when obtaining a notifier and makes it easy to look up if a VI is receiving all the required notifiers.

## Timing Choices

To ensure the correct working of the Matlab code in the LabVIEW environment, a test VI was created to understand how the Matlab node needs to have the data delivered in order for it to work correctly and how it outputs its data. The noise and data processor Matlab nodes were set up in such a way that it simulated how it needed to work in the Data Processor VI. To get a sense of how long the processing of the noise and data would take, a timing test was setup in the same test VI.

Tests with this VI showed that there was a significant overhead when communicating between LabVIEW and Matlab. When a the processing of a measurement beat signal takes 1.5 seconds in just Matlab, it takes a total of 2.5 seconds in Labview, which is quite significant.

The 2.5 seconds it takes for the processing was however still within the time it takes the PXI to receive and send a beat signal. So with those numbers the data processor should be able to process all the data received from the radar without dropping a packet. When the Matlab nodes were implemented in the Data Processor and the data was sent to the nodes via notifier (simulating the actual working of the system), it turned out that there was additional slowdown. It takes an average of 4 seconds to process the data, which is more than the 3 seconds it takes for the PXI to receive and send a beat signal. One packet is therefore dropped every two packets that are sent to the Data Processor.

While implementing the Matlab code in the Data Processor, there was a choice between adding the Matlab node directly into the LabVIEW environment or to create a subVI containing the Matlab node. The latter choice makes the code easier to interpret and change later on, by being able to just find the subVI containing the Matlab node in the project folder and changing the function it needs to call. However, after some tests it seems that when processing 10 profiles, the processing takes 100-300ms longer when the Matlab node is placed inside a subVI and then called. Therefore in order to speed up the processing as much as possible the choice for just using Matlab nodes was made.

## Choice for the use of intensity charts

In LabVIEW there are several types of graphs, each for a different type of data. For the graphs in this design a display is wanted that displays 3D data on a 2D plot using colors. Because of the fact that the data should be displayed as function of time and height a normal 2D graph cannot be used. In LabVIEW there are 2 types of graphs that can be used for plotting 3D data with the use of colors. These are an intensity graph and an intensity chart[15]. The main difference between these two is that in a graph new data results in a completely new figure every time, were as in a chart every time new data comes in a new vertical line is drawn. When a measurement is taken with TARA a new array should always result in a new line on the graph, not in a new picture. This is why the intensity chart is chosen. Also this chart has a history which saves data that is not visible anymore, making scrolling back possible. This is an advantage, but the memory size should not be set to high, because then the system will be slowed down.

# Chapter 6

# Implementation

In this chapter it is explained how the different subroutines have been implemented in LabVIEW. Section 6.1 details the implementation of the Radar Control as an overarching system, followed in section 6.2 with how the GUI was implemented. Section 6.3 briefly explains the how the Header Builder works. After which section 6.4 deals with the implementation of the Radar Processor and this chapter will end with the implementation of the Visualization in section 6.5. Finally in section 6.6 some future work related to the Storage implementation is briefly discussed. The LabVIEW code for these Vis can be found in Appendix D.

## 6.1 Radar control

To make sure that all the different subroutines run at the appropriate time and get the right signals for their actions, a design of an overarching VI was created that would run and link these subroutines together. During development it became apparent that the actions of this overarching VI could be interpreted as those of the Radar Control block that was created during the design of the Main Block Diagram. This overarching VI was then rebranded as the Radar Control VI.

The VIfor the Radar Control starts with obtaining and naming all the notifiers that are used in the systemand then wiring them to the input connectors of the subVIs that also need them. After the notifiers are obtained the System Control VI is started, which will keep running as long as the program is running. Afterwards the GUI and the subroutines for the Data Processor, Visualization and Storage are disabled by setting their 'case' to false when Radar Control first runs. The GUI should not be displayed until the radar is set up correctly and no errors are found and the three subroutines should not run without the header-data that is provided by the GUI.

### The Main State Machine

*The Main State Machine* is the part of the Radar Control that controls when certain VIs should run and when notifiers are sent to the different subVIs of the system. The first state it enters after it starts up is the *Radar Initialization state*. In this state the radar system checks for errors and sets the initial values for the PXI. When this is done and no errors are found, a signal is sent via notifier to the 'case' containing the GUI. The GUI will then run and pop up on the screen, allowing the user to input the desired measurement options. The State Machine will then enter into an *Idle state*.

The State Machine will wait in this *idle state* until it receives notification from the GUI on the *GUI control of Radar Control notifier*. This notification indicates, by sending along the next state, that either the measurement options have been set and the switch has been turned on to start the measurement or that the exit program button has been pressed. Regarding this notification, the next state will either be the *Noise Measurement state* or the *Shutdown state*. In the *Idle state*, the 'case' containing the Data Processor, Visualization and Storage VIs will be set to false via the *Start data processor /storage and visualization*

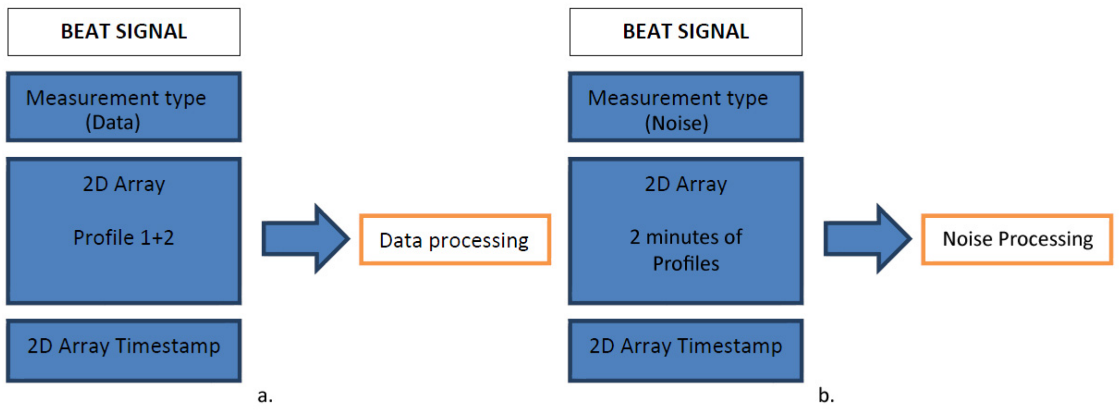


Figure 6.1: Beat Signal notifier

*notifier*. This is done so that whenever the user stops the measurement and the State Machine returns to its *Idle state*, the three VIs in the 'case' can be started again after new measurement options have been input.

When the State Machine enters the *Noise Measurement State*, the VIs for the Data Processor, Visualization and Storage will start running by setting the 'case' that contains them to 'true' by making use of the *Start data processor /storage and visualization notifier*. The Noise Measurement VI is then run in this state, which is used to interact with the PXI in order to put a beat signal containing 2 minutes of noise (bundled with a time stamp and the measurement type, see figure 6.1.a) on the *Beat Signal notifier*. This notifier will be obtained by the Data Processor where it is meant to be processed. Afterwards the Noise Measurement VI will put another beat signal containing 1 minute of raw data on the notifier to be stored, which should be done in parallel to the processing of the 2 minutes of noise collected before. Should this VI encounter an error during execution, this error will be output in State Machine and it will return to the *Idle state*. If one of the notifier elements outputs an error, this means that the notifiers have been released, because the user has pressed the exit program button. The State Machine will go the *Shutdown state* afterwards. In case there are no errors, the next state will be the *Data Measurement state*. The *Data Measurement state* will start the Data Measurement VI. This VI will keep running as long as the switch for the transmission in the GUI is switched on. It will continuously collect beat signals and put them on the *Beat Signal notifier* every time it has collected 2 profiles (bundled with a time stamp and measurement type, see figure 6.1.b). This beat signal can then be used in the Data Processor and Storage VIs. The Data Measurement VI should only exit when the user has stopped the transmission. So after it stopped running, the *Data Measurement state* will check for errors and then return to the *Idle state*, ready for the user to start the next measurement.

Currently the only purpose of the *Shutdown state* is to exit the loop containing the Main State Machine when the user has pressed the exit program button. When storage is implemented, this state will be used to allow the Storage VI to finish storing data that is still in the buffers before shutting it down.

## 6.2 GUI

The GUI is the only part that the user can actively interact with. In the GUI the user can input different settings. The output of the system is presented in graphs and a comment box. The measurement is started with a switch. At this point all the underlying systems are started in order to complete the measurement with the settings of the user. A picture of the GUI is given in appendix C. The software behind the GUI is running in three separate while loops, in order to keep the different processes running in parallel. There is a loop for the inputs, one for the graphs and one for the system comment box.

## User inputs

The user has a variety of inputs that he/she can give to the system. Most important are the setting for the measurement. For these settings, the user can choose between normal settings and advanced settings. These two categories are separated by a tab. On the tab with normal settings, settings can be made only in a way that the data processor can process the data. On the tab with advanced settings the user can also make settings that are not supported by the data processor. In that case only raw data will be saved. In both cases the user can input the polarimetric measurement cycle, the frequency excursion, sweep time, range FFT length, Doppler FFT length and attenuation. At the advanced settings the user can choose if the delay line is on or off and input a fully customized polarimetric measurement cycle.

Besides these settings the user should input the GPS coordinates, the elevation of the antennas and the azimuth of the radar. For storage, the user is able to choose which types of files he wants to save and the user can add a comment that will be saved, for instance what the weather is like. Also the user can input a schedule for measurements, but at this moment, it is not functional. Finally, there are three buttons, one to start or stop the measurement, one to reset all the settings to their default values and one to exit the program.

When the measurement starts the input fields are all disabled. This is done via a property node in a for loop. A property node is a block in labVIEW in which you can input a large amount of properties for the fields on a GUI. One of these properties is the disable property. As soon as a '1' is sent to the node the corresponding field is disabled. A property node can either belong strictly to one object or different objects can be linked to it. This is done in a for loop, by putting different references in it one after each other. Also when the measurement starts the header should be built. For this purpose, the inputs are bundled and then sent from the GUI to the header builder. Also, some calculations are done in the calc matlab block to check whether the settings for the measurement are valid. If they are not the measurement will not start. When the measurement is stopped by the user, all the other blocks are informed, and all the inputs are enabled again, using the same method as when disabling them. When the exit program button is pressed the measurement should be stopped and all the other VIs in the system should stop, in order to be able to close the program. To do this a signal is send, so that the other blocks can finish their work, to prevent loss of information. After all the blocks have stopped working the program is shut down.

In advanced mode the user can make his own measurement cycle. To do this there are 7 buttons, 6 for the different polarimetric measurement types (HH HV VH VV O1 O2) and one button to remove the last added type in the sequence. When a button is pressed the corresponding type is added to a string. In order to do this 7 cases were made, one case for each button. The string is a local variable, which is necessary to keep it while going trough different cases. Each time a button is pressed the corresponding case is executed. In this case the correct string is added to the final string. When the measurement is started the string is put in the header builder and thereby passed on to the radar control.

## Graphs

There are 15 graphs all of the same type. These graphs show the processed data in a 2D graph, the x and y-axes always represent relative time and height respectively. The z-axis is a color representation displaying the value of the variable shown in the graph. The 15 graphs are grouped in 5 groups of 3 displayed on different tabs. The 5 groups are: main beam, offsetbeam 1, offsetbeam 2, polarimetric and wind. This way the GUI stays orderly without throwing away measurement information.

For the graphs, intensity charts are chosen. An intensity chart is a chart that shows values by color, the chart automatically draws a new vertical line when new information comes in. When the chart is full the oldest line (the one on the left) slides out of the box and a new one is drawn on the right (untill the real time buffer of the chart is full). The lenght of this scrolling option can be set automatically, but should be limited in order to not saturate the physical memory. All the charts receive different information. The information is represented in an array, which is sent by the visualization block in the *Visualization for GUI notifier*.

The axes are all scaled programmatically in the GUI block. The information needed for this comes from the visualization block in the *Axes notifier*. The scales are set with property nodes. The y-axes are set in a for loop, because of the fact that they should all be the same. For the y-axes the property node gets 3 values, a minimum, a maximum and a multiplier. The z-axes are all set individually, each node receives an array of colors and corresponding values.

### System comments

The system comments box displays all the information about the system. This includes warnings, errors and the version of the matlab code. The information for the box comes from the system control, where all the comments are collected and put in a queue. In the GUI, the strings should be collected and displayed. This is done by concatenating the new string with the old one, with in between an enter, in order to display them one under the other. In order to do this the value of the string is read out by a property node and then put into a concatenate block.

## 6.3 Header Builder

The Header Builder VI is a basic Functional Global Variable. When it is called in the GUI with the *initialize state* it puts the bundled variables it receives from the GUI on a shift register. Other VIs can then call it with the *get state* and it will then take the bundled variables from the shift register and output them to the VI that calls the Header Builder.

## 6.4 Data Processor

The Data Processor has to receive the measurement options set in the GUI to process the beat signal provided by theradar. The processing itself is done in Matlab, by making use of the Matlab nodes in the LabVIEW environment. This allows a great flexibility in updating processing codes which are all written in Matlab.

In the Data Processor VI, the notifiers are wired to the notifiers outside of the while loop throught the references. This is also where the measurement settings are extracted from the Header Builder and then unbundled within the while loop in order to use them for processing of the beat signals. The notifier elements generate errors when the notifiers are released while they are waiting. The Matlab Script Nodes also generate an error when there is a problem with the input variables. In both cases where an error is produced the VI should shut down. When no errors occur, the VI is then shut down when the user stops the measurement via the GUI. Therefore the signal from the *Transmit ON/OFF notifier* also generates an error if the value is false in order synchronize the shut down operation with the other two errors that can cause the VI to shut down.

### State Machine

The State Machine in the Data Processor is used to run the Matlab nodes which contain the code for the processing of the noise and measurement beat signals, in the right order and choose the correct Matlab node for the type of measurement (3 or 5 cycles). The State Machine starts in the Idle state where it waits for a beat signal to be put on the Beat Signal notifier. When the beat signal is retrieved from the notifier, the 'Polarimetric Cycle' is read from the Header Builder and compared to the standard polarimetric cycles for 3 and 5 cycle measurements. If they match, the next state will be the *Noise Processing state*. If not, it will remain in the *Idle state*.

When entering the *Noise Processing state*, the beat signal cluster coming from the *Beat Signal notifier* is unbundled in order to separate within the cluster the beat signal from the time stamp corresponding to when the beat signal was taken and the measurement type, which is of either the noise or transmission type. Because noise measurement is required to get processed data, the beat signal is checked, if it first supplied the noise measurement type. If not, the state machine will return to the *Idle state*. If the measurement type is correct, the header data required for the processing along with the noise beat signal that needs to be processed will be used as input for the Matlab node that contains the Matlab code to process the noise.

Two variables will be computed that will be bundled together as 'processed noise data'. The code version of the Matlab code that was used in the calculations will also be provided as an output of the Matlab node in order to keep track of the type of processing performed. When the noise is processed, both the code version and the processed noise data will be put on notifiers for use in other VIs. The Processed Noise Data will also be sent to shift registers to be used for processing of the raw data in the next state. When the next state is activated, a first selection is done by comparing the input polarimetric cycle to the polarimetric cycle for 3 cycle measurements. If these are equal, the next state will be the *3 cycle state*. If not, the next state will be the *5 cycle state*.

In the *3 cycle state*, there will be another check to see if the measurement type of the provided beat signal is of the correct type, i.e. Data measurement type. If not it will not use the matlab code to process the data, but instead, skip it and remain in the state and check the measurement type of the next beat signal. If the measurement type is of the Data Measurement type it will take the variables extracted from the header, the beat signal and the processed noise data that was calculated in the *Noise Processing state* and use them as input variables for the Matlab node containing the Matlab code for 3 cycles polarimetric data processing. 7 variables are computed, which will be bundled together as a '3 cycle processed data' cluster. The cluster consists of the following variables:

- $Z_{DR}$: The differential reflectivity in dB.
- $L_{DR}$: The linear depolarization ratio in dB.
- $Z_{HH}$ : The reflectivity value of the main beam for HH polarization in dBZ
- $V_{HH}$ : The mean Doppler velocity of the main beam for HH polarization in m/s
- $W_{HH}$ : Doppler width of the main beam for HH polarization in m/s
- $\rho$: The complex cross correlation coefficient
- $Height$ : The height vector in meters above ground level

This cluster will be put on the *Processed Data (3 cycle) notifier* to be used by the Visualization and Storage VIs. The code version of the Matlab code that has been used will be put on the *code version notifier* also for keeping track of the processing type performed on the data. The 5 cycles state acts similar to the 3 cycle states, the differences being that the Matlab node with the Matlab code for 5 cycle polarimetric data processing require an extra input and gives additional outputs. The outputs are 15 variables that are bundles as '5 cycle processed data' cluster which will be put on the *Processed Data (5 cycle) notifier.* This cluster has the following additional variables compared to the '3 cycle processed data' cluster:

- $Z_{OB1}$: The reflectivity value of the offset beam 1 in dBZ
- $Z_{OB2}$: The reflectivity value of the offset beam 2 in dBZ
- $V_{OB1}$: The mean Doppler velocity of the offset beam 1 in m/s
- $V_{OB2}$: The mean Doppler velocity of the offset beam 2 in m/s
- $W_{OB1}$: The Doppler width of the main beam for HH polarization in m/s
- $W_{OB2}$: The Doppler width of the main beam for HH polarization in m/s
- $V_z$ : The vertical mean Doppler velocity in m/s
- $V_h$ : The horizontal wind speed in m/s
- $D$ : The horizontal wind direction in degrees

As with the previous two Matlab nodes, this Matlab node also provides the code version which will be put on the *code version notifier.*

## 6.5 Visualisation

The visualization block takes care of visualizing the measured data. First the scales of the graphs are set correctly. Then the data is sent to the graphs. At the end of a measurement or a day, a quicklook should be made, but this has not been implemented yet, because of the lack of time.

### State Machine

At startup the visualization block starts in the *initialization state.* In this state the polarimetric cycle that is used for the measurement is extracted from the header builder, for later use. Furthermore, this is the only state in which the visualization sub VI can be shut down by the system, in order to prevent the loss of quicklooks. After the initialization state the block goes to the *Adapt scales state.*

In the *Adapt scales state* the y scales and the z scales are set. The y scales are fixed for all graphs, while the z scales are set individually. The y scale are set with information from the processing block. The processing block sends a height vector in the *processed data notifier.* From this vector the minimum and maximum are derived. For the z scale, 2D arrays are needed with the combination of color and the value for that color. At the moment the maxima and minima for the z scales are fixed for each different parameter, but later they can be made variable if necessary. From the minima and maxima, other values are calculated, in order to make a full color scale. These arrays of collors and values for each graph are then put in a larger array, and this array is bundled with the values for the minimum and maximum of y. This bundle is then sent to the GUI via the *Axes notifier.* After this state the visualization block goes either to the *3 cycle visualization state* or the *5 cycle visualization state.*

The *3 cycle visualization state* and the *5 cycle visualization state* are very similar. The data for the graphs comes from the data processor via the *processed data notifier.* In the visualization block the data is given trough to the GUI via the *Visualization for GUI notifier.* The arrays are first transposed, in order to match the vizualisation format. After that, all the information is put into a bundle, which goes to the GUI via a notifier. When a 3 cycle measurement is taken not all the graphs are used. To be sure that the unused graphs stay black, an array filled with zeros is sent to the unused graphs. After this the block goes back to the *initialization state.*

## 6.6 Storage

The design for the storage has been done, a block and state diagram have been made (see chapter 5). But implementation has not been realized, because of time constraints and tasks re-allocation between the two sub-groups (in order to compensate unexpected extra working load which would affect the design implementation). Therefore, in agreement with the user there has been decided to skip the storage part. The implementation of the storage should be done based on the current block and state diagram, in order to complete the program.

# Chapter 7

# System evaluation

In this chapter the result of the project is evaluated by looking at the system and the program of requirements. All the requirements made in chapter 3 have been checked. During the excursion to Cabauw, at the last day of the project, where the system has been implemented in the actual TARA.

This chapter follows the same outline as chapter 3. In section 7.1 the product use is discussed, after that in 7.2 the environment and ecology of the system is evaluated. Then in 7.3 the requirements for the system design are discussed and in 7.4 the production of the system is evaluated. Finally in 7.5 conclusions are drawn and recommendations are made.

## 7.1 Product use

The system processes the data and visualizes it in real time, in other words the user has the feeling that everything is happening immediately after measuring. The user can select which measurement needs to be taken and all the settings stay visible when the measurement is started, so the user can see if everything is set up correctly. The graphs are scaled automatically and the units of the radar parameters are shown next to the graphs.

Due to a lack of time, because of a change in task division and the complexity of the system to be implemented, the storage part has not been implemented. All these requirements are therefore not met. However, a block diagram and a state diagram have already been made for the storage block, and is therefor ready to implement. In these diagrams it has been taken into account that different types of files have to be stored on the same hard disk. Also knowledge of the standard file formats is present at the department. A header has been made in order to hand all the necessary information to the different blocks, so this element can be added to the files when data is saved.

The radar parameters should be plotted as function of UTC time and height. The height is used to scale the graphs, but the x-axis are not showing the UTC time. The x-axis are now showing relative time compared to the starting time. The demand of showing UTC time is most important when making a quicklook, because when a measurement is taken it is known which time it is at that moment, but when data is taken out from a database this is harder to know. At this moment no quicklooks are made, when this is implemented it also might be possible to make quicklooks with UTC time.

## 7.2 Environment and ecology

The software is running on a normal PC under Windows. The PC that is used is a high end consumer PC with Microsoft Window 7 as the operating system. The system is user friendly, and has been created in close interaction with the people working in the ATMOS-RSE group, who are the future users. The software should run stable, but this is not entirely the case yet. Every couple of runs the computer has to be restarted, the likely problem is that parts of the system (mainly graphs and processor) uses too much memory or does not release all of it.

## 7.3 System design

### Usage features

The system is designed using an open and flexible approach, so it allows for easy implementation of new functions and simple debugging. The user is able to update the code easily and the matlab code written is easily implementable in the program.

The noise processing should be done in 1 minute, to make parallel computing easier. This hasn't been tested in the final design, because the code to collect 2 minutes of noise data which is necessary had not been implemented in the design during the final test. This code wasn't there, because it takes a lot of time during the test, but also because it is uncertain if it is possible to transfer such amounts of data in LabVIEW. In the design during the test there has been a noise data collection of 15 seconds, and the processing of this was done in a few seconds.

There have been tests with the test VI, where the results looked promising. During those tests it took around 40 seconds to process the noise data, which is well within the limit of 60 seconds. However during implementation of the data processor in the system, it became apparent that the communication with and calculations in Matlab slowed down. The processing of 2 profiles in the test VI takes 2.5 seconds in the Test VI, but takes 4 seconds in the implemented data processor. This could possibly mean that the processing of the noise takes longer than the 60 seconds it should take, but further tests will need to be done to be sure.

The processing block should provide the correct scales for the axes. At this moment only the y scale is provided. The z scale is fixed, although the visualization block is made in such a way that flexible z scales would be possible. The x scale is relative time instead of UTC time. At this moment there is not enough LabVIEW knowledge to build a graph with a scale depending on a timestamp.

The storage block has not been build. However, checkboxes are present in the GUI in which the user can choose which kind of file he/she wants to store. With this information nothing is done at the moment. Because the storage block has not been build, in the visualization block there has been no attention for the quicklooks, so a buffer and quicklook generator are not present in the visualization block.

### Production and commissioning

The software is made modular, it consists of a number of Vis, so it is adjustable to other systems. It is easy to put in new VIs or remove unnecessary ones, also changes to the VIs can be made easily because of the modular structure. After the software is installed it is immediately ready for use, this installation process can be done in 60 minutes for the whole system (including the parts of the other subgroup). The system was operational at the 7th of June 2011. The software is documented at some points but at other points there is a lack of documentation. This is because of the fact that time was limited and the emphasis of the project was on making a product of good qaulity.

### Liquidation

The un-install process can be done within 60 minutes, again for the whole system.

## 7.4 Production

All the requirements for production have been met. The software is build in LabVIEW which is a standard software packet for building virtual instrument in order to take measurements. The pieces of matlab code are inserted in LabVIEW, and can even be changed easily. The PXI system is used for the data collection.

## 7.5 Conclusions and recommendations

Most of the requirements listed in chapter 3 are met. And for the storage most of the requirements are met by the block and state diagram that have been made. Also the storage bock has been taken into account in every design and implementation step of the other blocks. When the block is implemented it is thus easy to connect it with the rest of the product. Some requirements have not been met due to a lack of LabVIEW knowledge. At the start of the project there was little knowledge about this programming language, so everything had to be figured out. This caused for some not implemented functions. However, the software is made flexible, so it is easy to put in new elements.

After evaluation with the staff and the students, it was considered that more time and manpower would have been required to fully fulfill the project tasks. The product that has been designed can be used, although no storage is done. Someone of the ATMOS group will build the storage block, to make the system complete, this can be done according to the block and state diagrams described in chapter 5. The VI can then be put into the rest of the project and when connected correctly the data can be saved. Also, in order to make it easier to make future changes, the documentation should be expanded.

# Chapter 8

# Discussion

In this chapter three points are discussed on which the project or the product could be improved. Things that went wrong are pointed out and possible solutions are given.

## Time

As mentioned earlier, the Bachelor end project has a tight time frame and deadlines have to be met. Because of this, decision are made that would perhaps be different if there would be more time. Also some choises had to be made, in order to keep the qaulity of the delivered product. These choises where the cause of a smaller product.

## LabVIEW

At the start of this project the language LabVIEW has been chosen to program in. This was mainly done because of the fact that, with this language it was possible to have a prototype within a short time which is also very flexible for later adjustments. In the beginning of the project knowledge of this language was limited, both in the project group and in the department. This meant that a lot of time was spend on getting to know LabVIEW. The languages well known by the project group are Matlab and java, both not considered suitable to use for the main frame of the radar system. Although Matlab has been used for most calculations, it is not suitable for building a main structure .

Another language that could have been used is C. This language has not been chosen because of the fact that it is harder to learn and harder to keep a good overview of what has been programmed. With C it would be possible to use Matlab code. There are even two options to do that, Matlab can be called in C code, or Matlab code can be translated into C which can easilly be done in the Matlab compiler. If Matlab is translated in C and then put in the project, the program will probably run faster, because of the fact that only one platform is running. In LabVIEW two platforms are running, that should constantly exchange information which can slow down the process. With the main limitation being the time to finish the project, LabVIEWs advantage of being relatively easy to learn was bigger than the disadvantage of slowing down the system. Also, if LabVIEW is well documented it is easier for someone else to understand than C code. Making it easy for someone to change parts of the system.n

## Task division

During the project, some tasks have been shifted from one group to another. The main task of designing a main block diagram and state diagram have been executed by both of the groups combined. But building the overarching system and the GUI have mainly be done by this group, because the task of the other group turned out to be more difficult than thought at first. Because of this there was not enough time to complete the task of building a storage block. The designing has been done, but there has been no implementation.

It is likely that the storage block could have been implemented successfully if more time was allocated for this task. Because, design of the blocks was already achieved and LabVIEW knowledge was already gained. However, in order to keep the quality of the delivered work, it has been chosen to deliver two properly finished blocks, rather than three unfinished. The two block that now were to be done, are indeed of good quality.

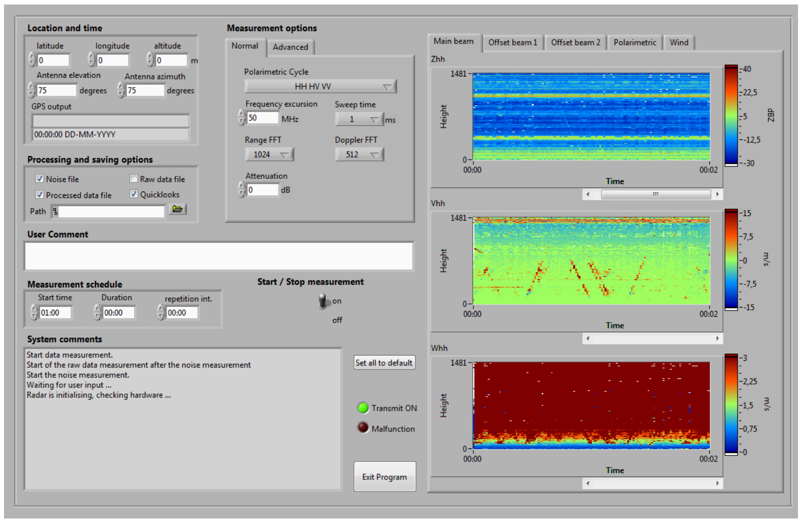
# Chapter 9

# Conclusions and Recommendations

The question of this thesis was: How to implement data processing, visualization and storage for TARA. In this chapter the conclusion will be drawn in section 9.1. After that in section 9.2 recommendations are done.

## 9.1 Conclusions

In order to design the full system, block and state diagrams were made. First, the main structure was designed in the main block diagram, and after that the blocks were worked out in detail and state diagrams were made. This task required almost three weeks, but proved to be extremely useful during implementation in LabVIEW. The Radar Control was implemented. Data Processor and Visualization were implemented and worked correctly during the tests with the PXI system at the TU Delft. The Storage block however has not been implemented in the system in LabVIEW. This was decided in agreement with the supervisors, because after a while it became clear that it would not be possible to implement all the designed blocks within the given timespan. Because of the fact that storage is at the end of the chain, other block can be very well implemented without it.
On the 7th of June 2011 there has been an excursion to TARA, where the new system has been installed and



Location and time
latitude 0, longitude 0, altitude 0 m
Antenna elevation 75 degrees, Antenna azimuth 75 degrees
GPS output
00:00:00 DD-MM-YYYY

Processing and saving options
[✓] Noise file, [ ] Raw data file, [✓] Processed data file, [✓] Quicklooks
Path

User Comment

Measurement schedule
Start time 01:00, Duration 00:00, repetition int. 00:00

Measurement options (Normal)
Polarimetric Cycle: HH HV VV
Frequency excursion 50 MHz, Sweep time 1 ms
Range FFT 1024, Doppler FFT 512
Attenuation 0 dB

System comments
Start data measurement.
Start of the raw data measurement after the noise measurement
Start the noise measurement.
Waiting for user input ...
Radar is initialising, checking hardware ...

Figure 9.1: Measurements taken on the 7th of June, at Cabauw

tested. At that day the system worked as was expected, see figure 9.1. In the graphs, noise, radar artefacts (horizontal lines in the top picture) and cars driving by at the nearby road(oblique lines in the mean Doppler velocities in the middle picture) can be measured in real-time. Normally the program would filter this out, but that function was put out on purpose, since no clouds were present to be measured. In this way it is shown that the system functions correctly. As shown in chapter 7, except from the non implementation of the storage block, most of the other requirements have been met. Summarizing, it has been demonstrated that real-time measurement can be performed with this system.

## 9.2 Recommendations

The system worked well during tests at the TU and at the site in Cabauw. This doesn't mean there is no room for improvements. Following are 5 recommandations for improving the system.

As mentioned, the storage block has not been implemented, but a block diagram and a state diagram have been designed. These diagrams can be used to implement a storage block, in order to make storage of the measured data possible. It is recommended to do this as soon as possible, since putting the data in a database is important for doing the research for which TARA is build.

One of the requirements was that the graphs are scaled according to the UTC time on the x-axis. This has not been achieved, due to a lack of knowledge of LabVIEW at this point. Research has been done but a solution has not been found yet. Some further investigation could be driven at this point so that this feature can be implemented in the product.

At this moment, inputs have been created on the GUI to set up a measurement schedule. This schedule is not implemented, nor is there a design to implement them within the system. A full investigation of the possibilities in LabVIEW for creating such a schedule is required. After which implementation can be done.

The software should run stable on a PC. This is not yet the case, since for an unknown reason, the PC has to be restarted every now and then (approximately every 15 min.), because it becomes very slow. It could be that the memory becomes full and old data is not removed. A possible solution would be to limit the amount of data stored in the graphs, but further investigation on the way the overall system is programmed should be done in order to accurately detect and resolve this issue.

Finally the LabVIEW code, Matlab code and the communication between them should be investigated in order to see if improvements can be made considering the timing. At this moment the processing block is the slowest of the system, so improvements should start from there. It is not known if the noise processing is done quickly enough when integrated into the complete design, so this should also be investigated and afterwards perhaps improved.

# Bibliography

[1] Jeroen van Gemert, Martijn Janssen and Satoshi Malotaux, Designing for TARA, The radar control unit, Bsc. Thesis, Electrical Engineering, TU Delft, The Netherlands, 2011.

[2] Muller, M., "Climate change - clouds remain the misty factor", *Delft Outlook*, pp. 22-27, 2007, no. 3.

[3] Ronald E. Rinehart, *Radar for meteorologists*, fifth edition. Nevada, Missouri: Rinehart Publications, 2010, pp. 1-14.

[4] Herman Russchenberg. (2011, May) TARA, the S-band Transportable Atmospheric Radar.
Available:
http://home.tudelft.nl/en/research/knowledge-centres/irctr/research/radar/research-facilities/tara/

[5] *Transportable Atmospheric RAdar TARA: User Manual*, TU Delft, The Netherlands, 2005, pp 1-7.

[6] S.H.Heijnen, L.P.Ligthart,"TARA: Transportable Atmospheric Radar", 28th Europian Microwave Conference,1998, pp. 61-66.

[7] S.H.Heijnen, L.P.Lighthart and H.W.J.Russchenberg,"First Measurements with TARA; An S-Band Transportable Atmospheric Radar", *Phys.Chem. Earth (B)*, vol. 25, no 10-12, pp.995-998, 2000

[8] Silvester H. Heijenen et al., "A dedicated computer system for FM-CW radar applications", *Journal of telecommunications and information technology*, pp. 21-25, 2001, no. 4.

[9] John G. Proakis, Dimitris G. Manolakis, *Digital Signal Processing*, fourth international edition. Upper Saddle River, New Jersey: Pearson Prentice Hall, 2007, pp. 416-425.

[10] *MATLAB Function Reference*, The MathWorks,Inc., 2010, fft.

[11] Y. Dufournet, Ice Crystal Properties Retrieval, doctoral dissertation, Dept. Telecommunications, TU Delft, The Netherlands, 2010, pp. 47-48.

[12] D. Spinellis, "Choosing a Programming Language", *IEEE Software*, pp. 62-63, 2006, July/August.

[13] NI Developer Zone. (2011, June) Are LabVIEW global variables good or bad, and when is it OK to use them?
Available:
http://zone.ni.com/devzone/cda/tut/p/id/5317

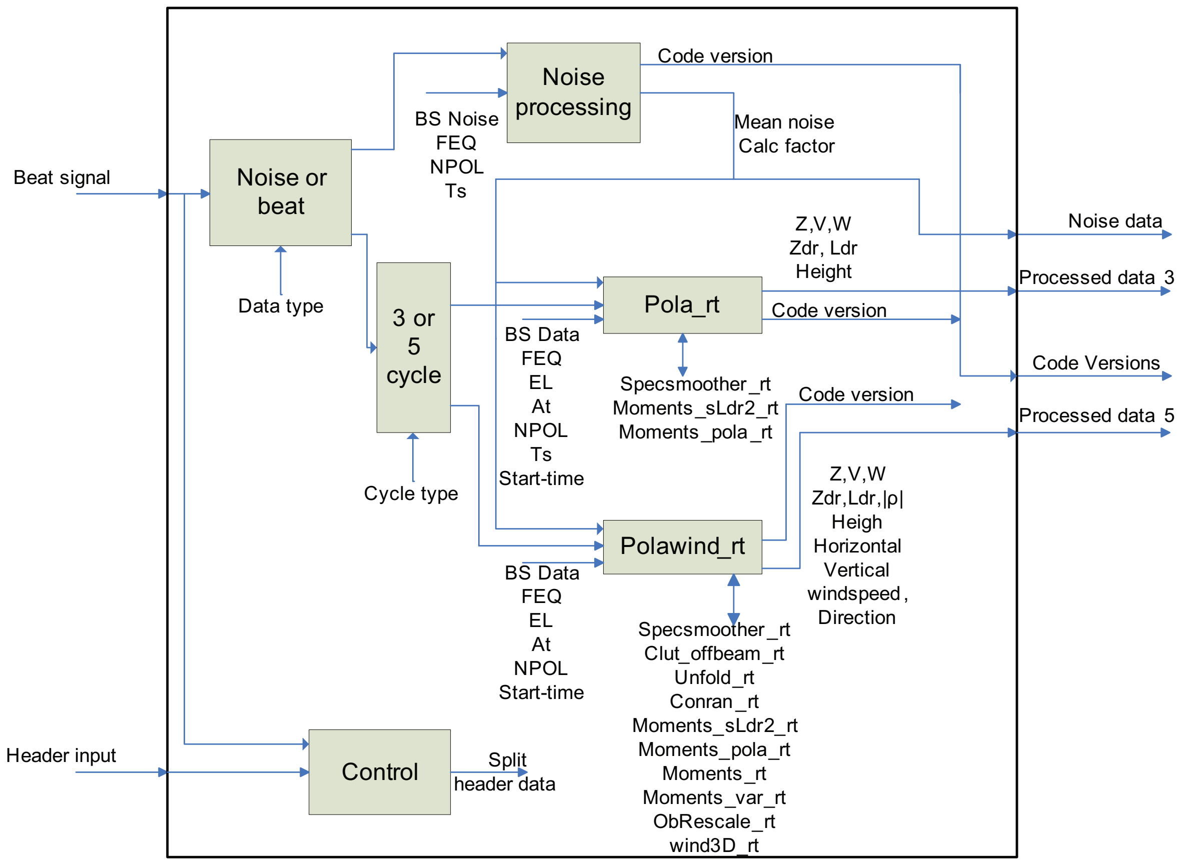
[14] Felix Annan. (2011, June) Notifiers in LabVIEW.
Available:
http://cnx.org/content/m13762/latest/

[15] LabVIEW manual. (2011, June) Types of Graphs and Charts.
Available:
http://zone.ni.com/reference/en-XX/help/371361B-01/lvconcepts/types_of_graphs_and_charts/

# Appendix A

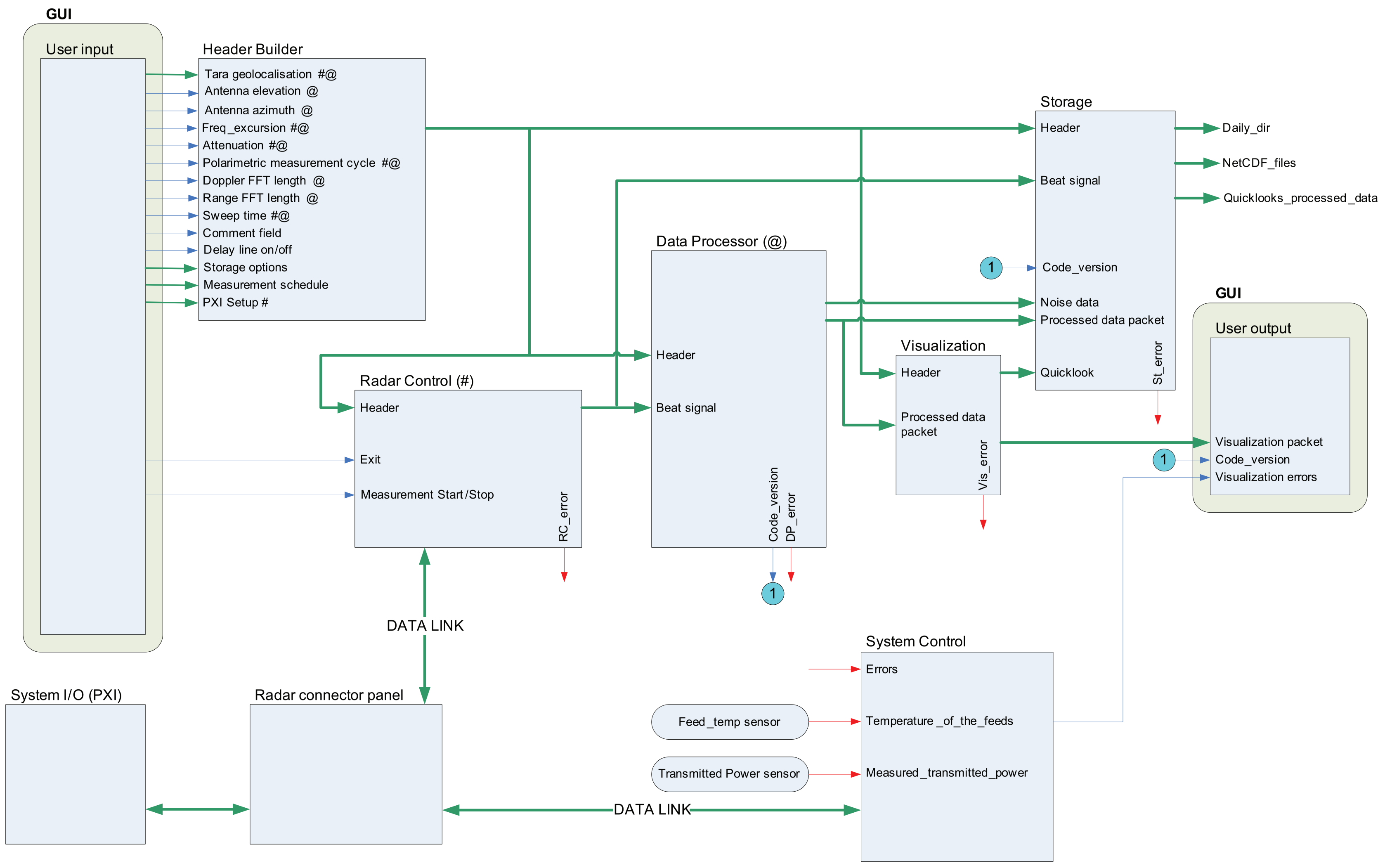
# Block diagrams

## Processing block

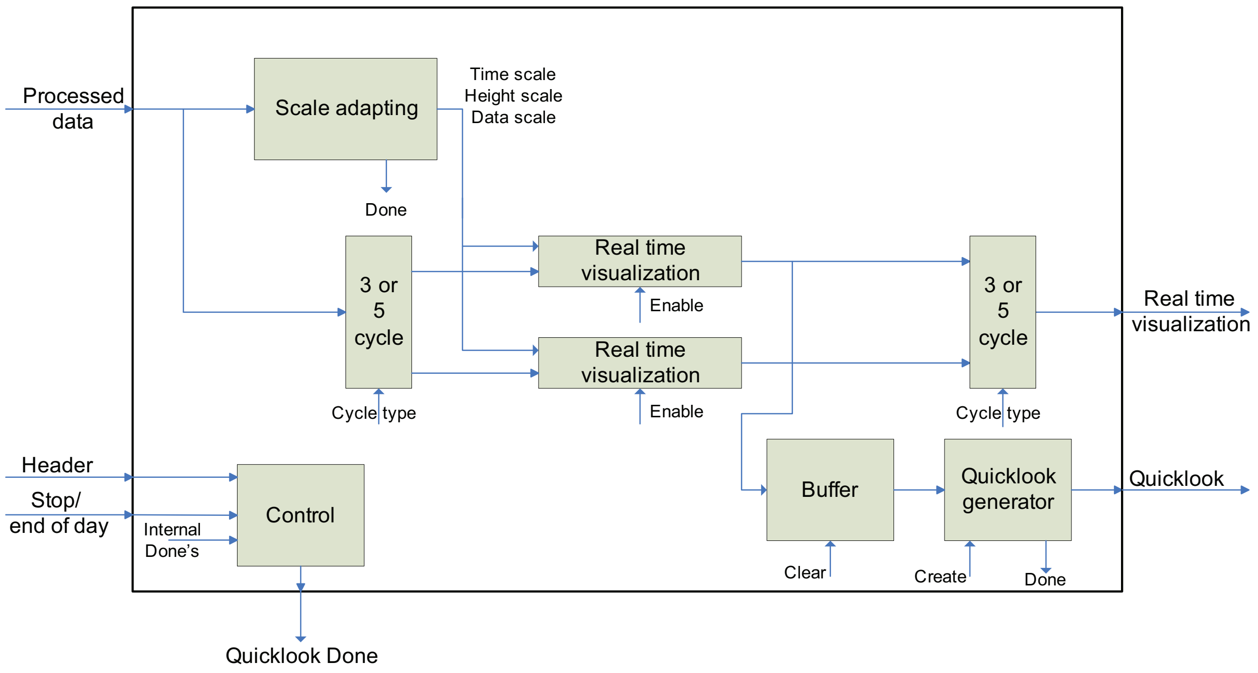


# TARA overall block diagram

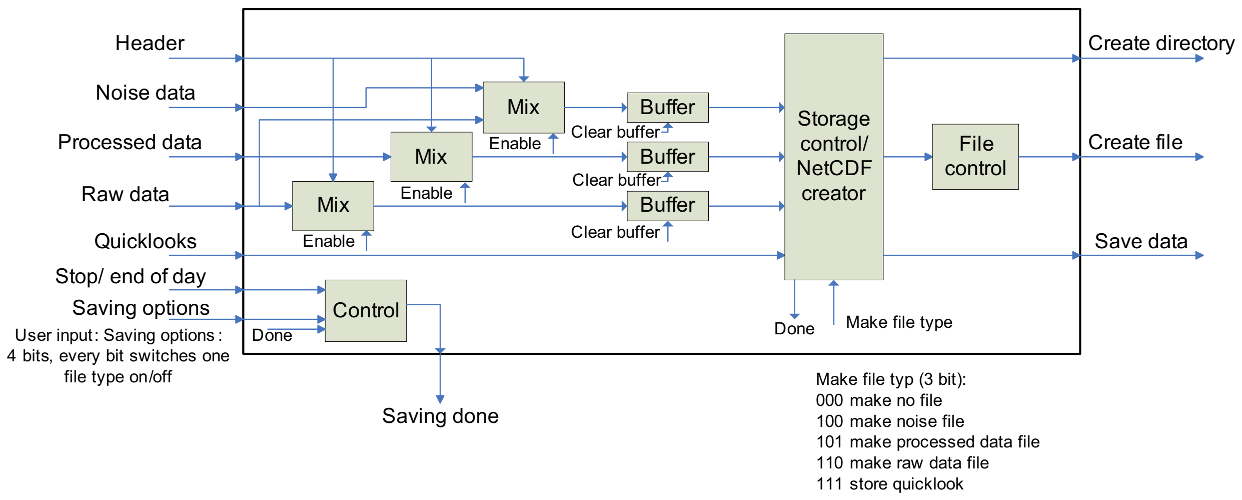
27-05-2011



## Visualization block



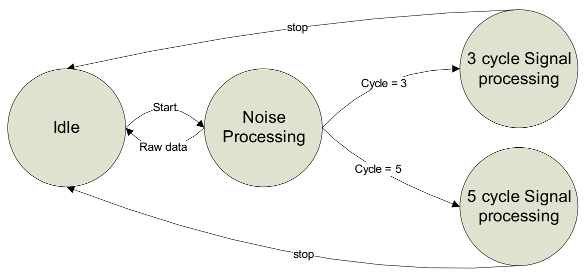
## Storage/NetCDF writer



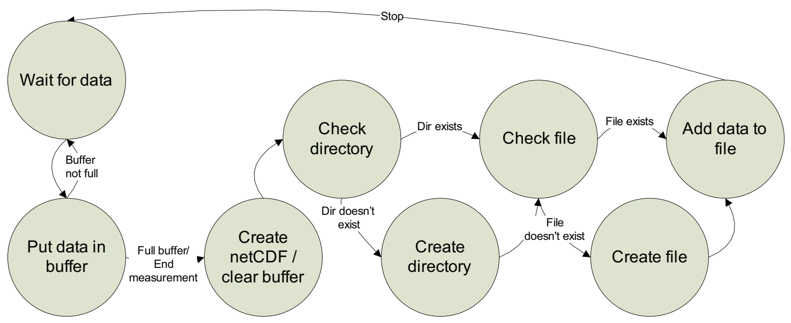
# Appendix B

# State diagrams

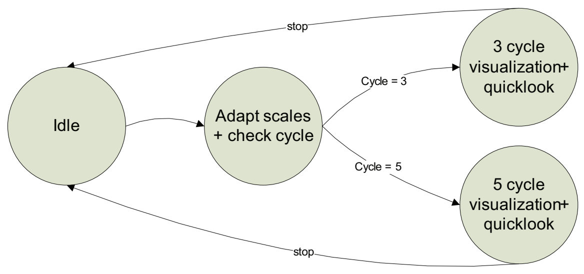
## Processing state diagram



## Storage Dataflow

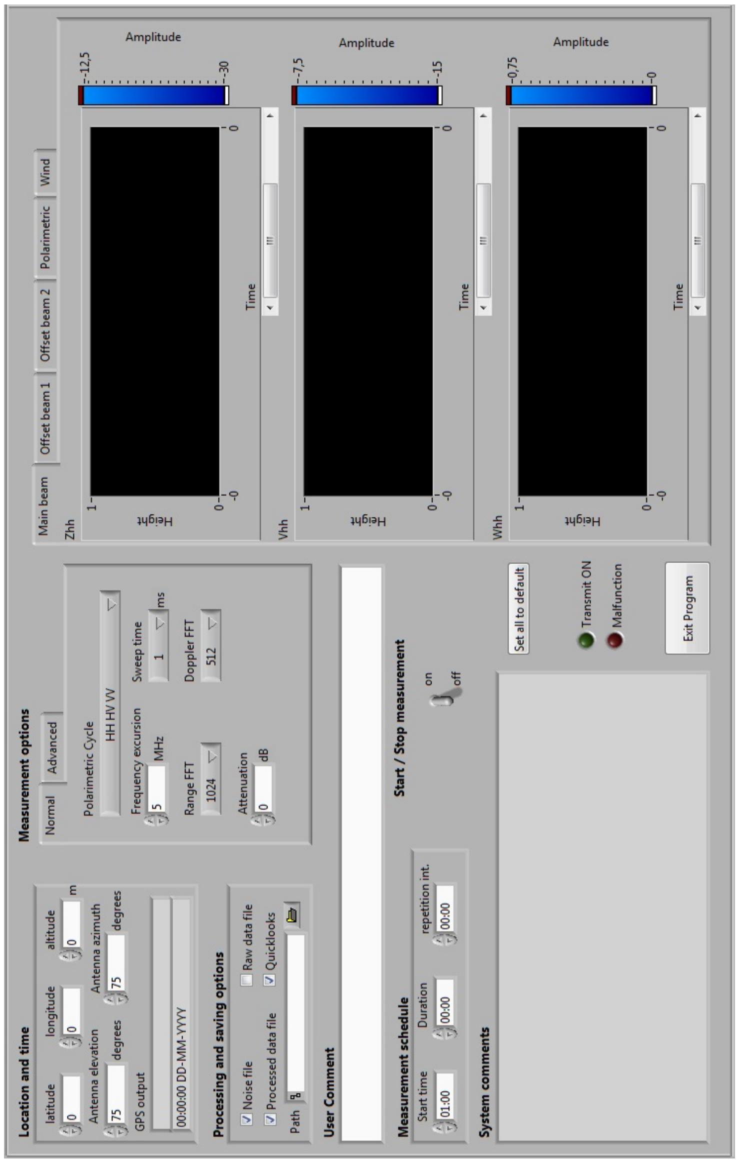


## Visualization Dataflow


stop
Idle
Adapt scales + check cycle
Cycle = 3
Cycle = 5
3 cycle visualization+ quicklook
5 cycle visualization+ quicklook
stop

# Appendix C

# User interface



**Location and time**

latitude 0, longitude 0, altitude 0 m

Antenna elevation 75 degrees

Antenna azimuth 75 degrees

GPS output 00:00:00 DD-MM-YYYY

**Processing and saving options**

✓ Noise file, ☐ Raw data file, ✓ Processed data file, ✓ Quicklooks

Path

**Measurement schedule**

Start time 01:00, Duration 00:00, repetition int. 00:00

**Measurement options** (Normal / Advanced)

Polarimetric Cycle HH HV VV

Frequency excursion 5 MHz

Sweep time 1 ms

Range FFT 1024

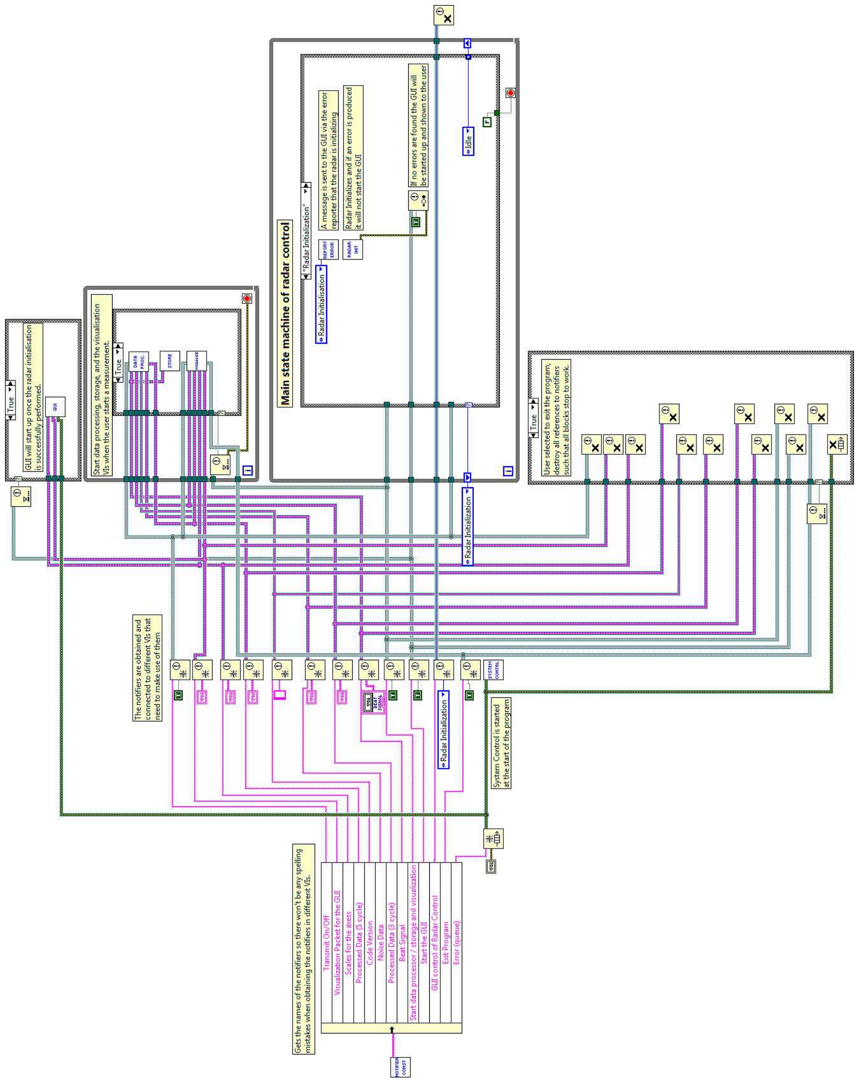
Doppler FFT 512

Attenuation 0 dB

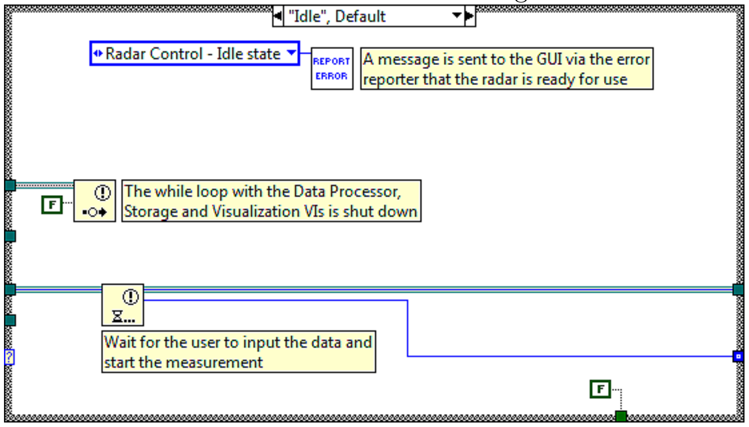
# Appendix D

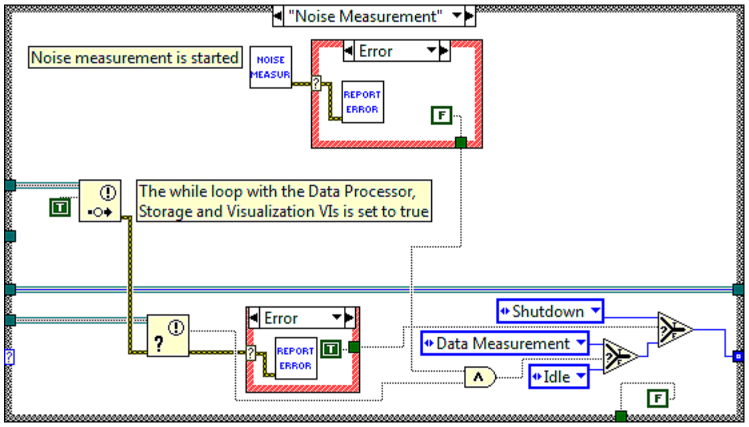
# Labview figures

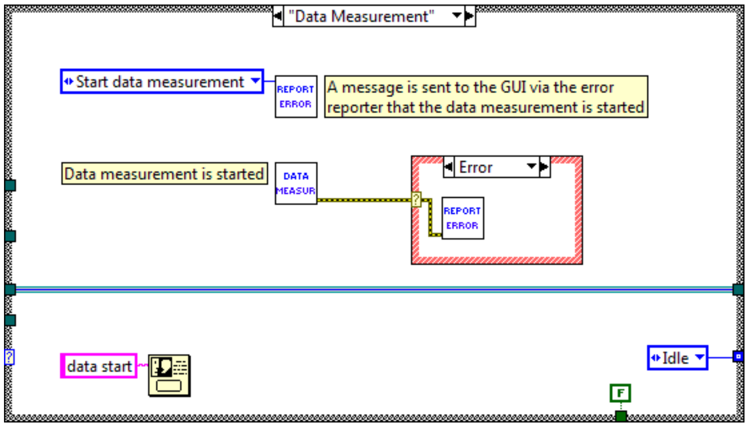
## Radar control


GUI will start up once the radar initialization is successfully performed.
Start data processing, storage, and the visualisation VIs when the user starts a measurement.
Main state machine of radar control
"Radar Initialization"
A message is sent to the GUI via the error reporter that the radar is initializing
Radar Initializes and if an error is produced it will not start the GUI
If no errors are found the GUI will be started up and shown to the user
Idle
Radar Initialization
True
User selected to exit the program, destroy all references to notifiers such that all blocks stop to work.
The notifiers are obtained and connected to different VIs that need to make use of them
System Control is started at the start of the program
Gets the names of the notifiers so there won't be any spelling mistakes when obtaining the notifiers in different VIs.
Transmit On/Off
Visualization Packet for the GUI
Scales for the axess
Processed Data (5 cycle)
Code Version
Noise Data
Processed Data (3 cycle)
Beat Signal
Start data processor / storage and visualization
Start the GUI
GUI control of Radar Control
Exit Program
Error (queue)

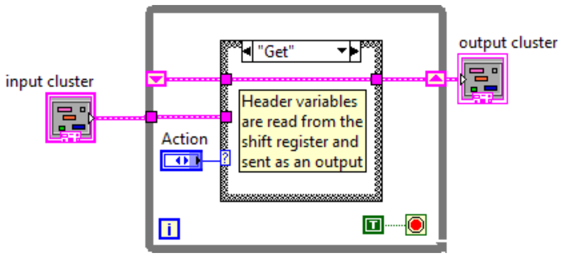
The additional states of the Main State Diagram.

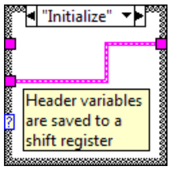

"Idle", Default
Radar Control - Idle state
REPORT ERROR
A message is sent to the GUI via the error reporter that the radar is ready for use
F
The while loop with the Data Processor, Storage and Visualization VIs is shut down
Wait for the user to input the data and start the measurement
F


"Noise Measurement"
Noise measurement is started
NOISE MEASUR
Error
REPORT ERROR
F
T
The while loop with the Data Processor, Storage and Visualization VIs is set to true
?
Error
REPORT ERROR
T
Shutdown
Data Measurement
Idle
F


"Data Measurement"
Start data measurement
REPORT ERROR
A message is sent to the GUI via the error reporter that the data measurement is started
Data measurement is started
DATA MEASUR
Error
REPORT ERROR
data start
Idle
F

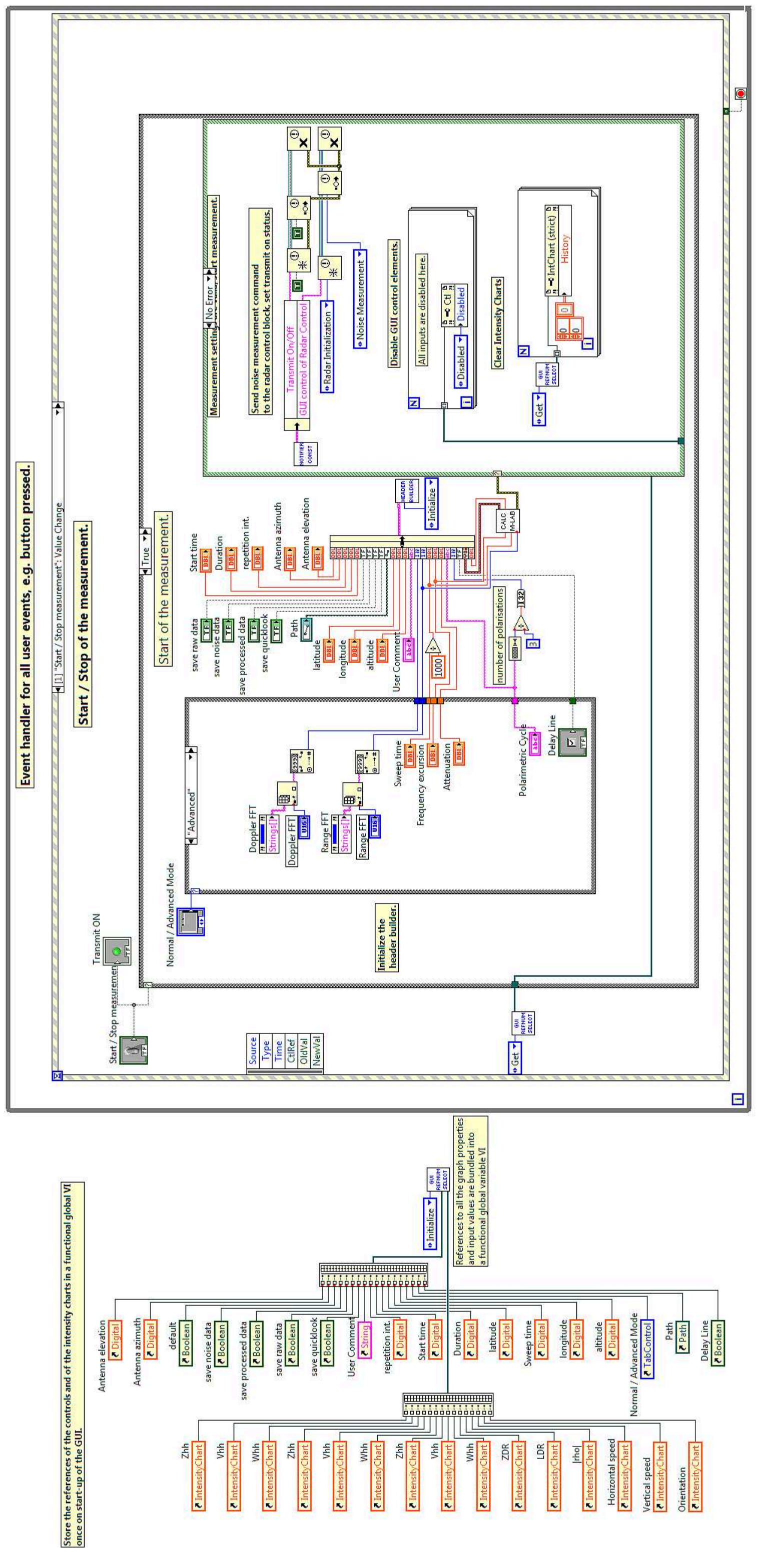
## Headerbuilder


input cluster
"Get"
output cluster
Action
Header variables are read from the shift register and sent as an output

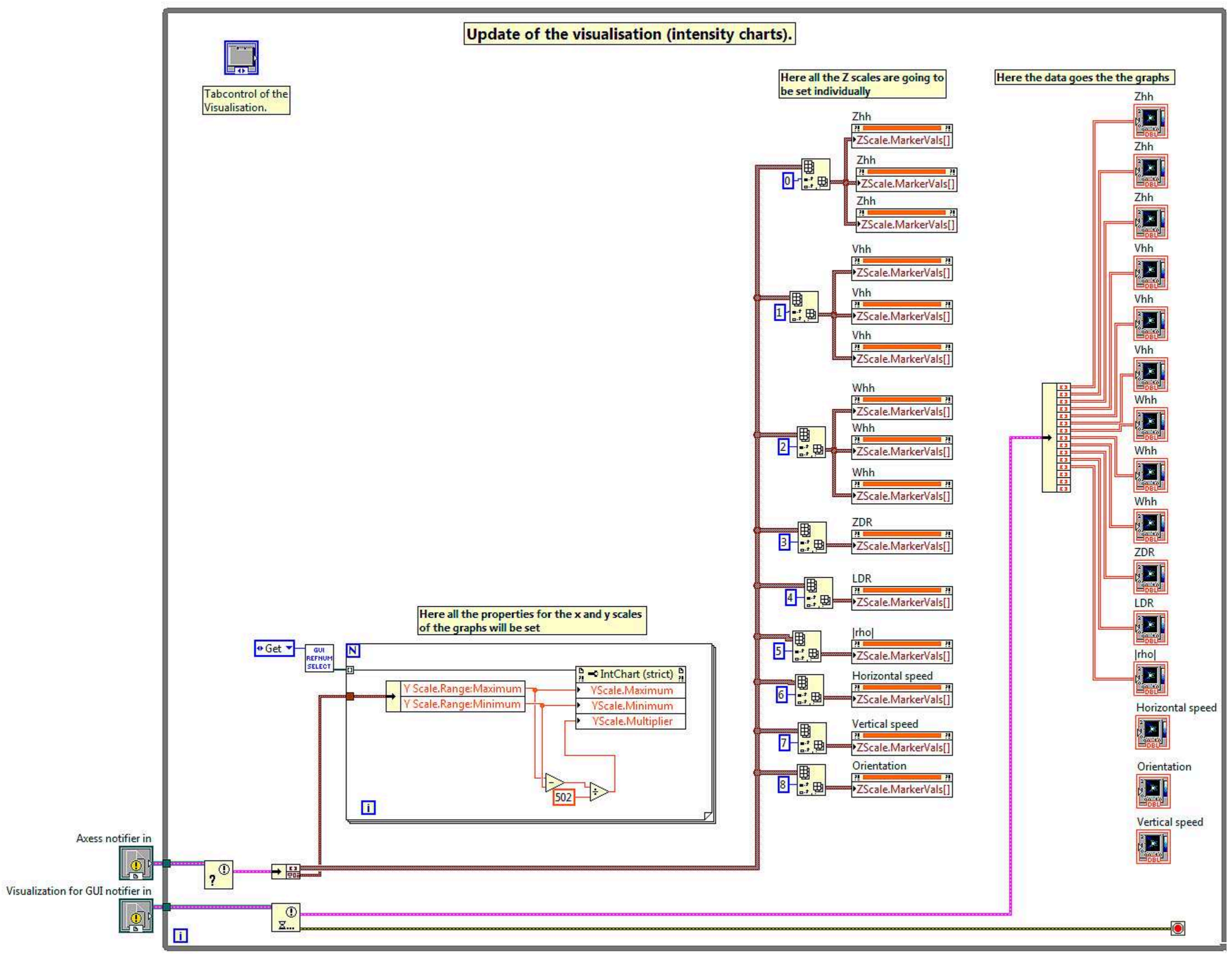


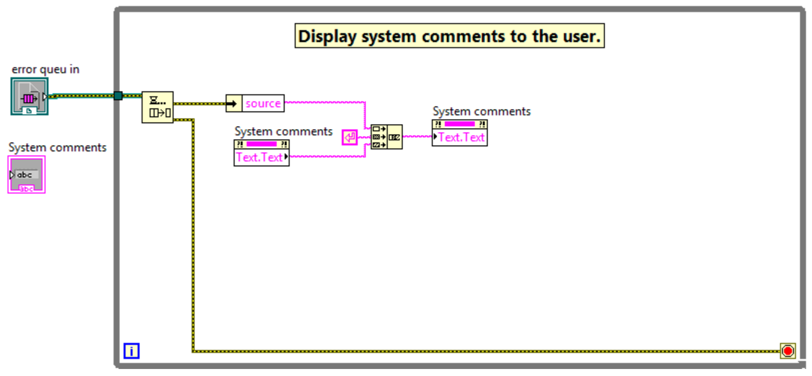
## GUI

The GUI VI, because of its size, is split up in 2 pages.

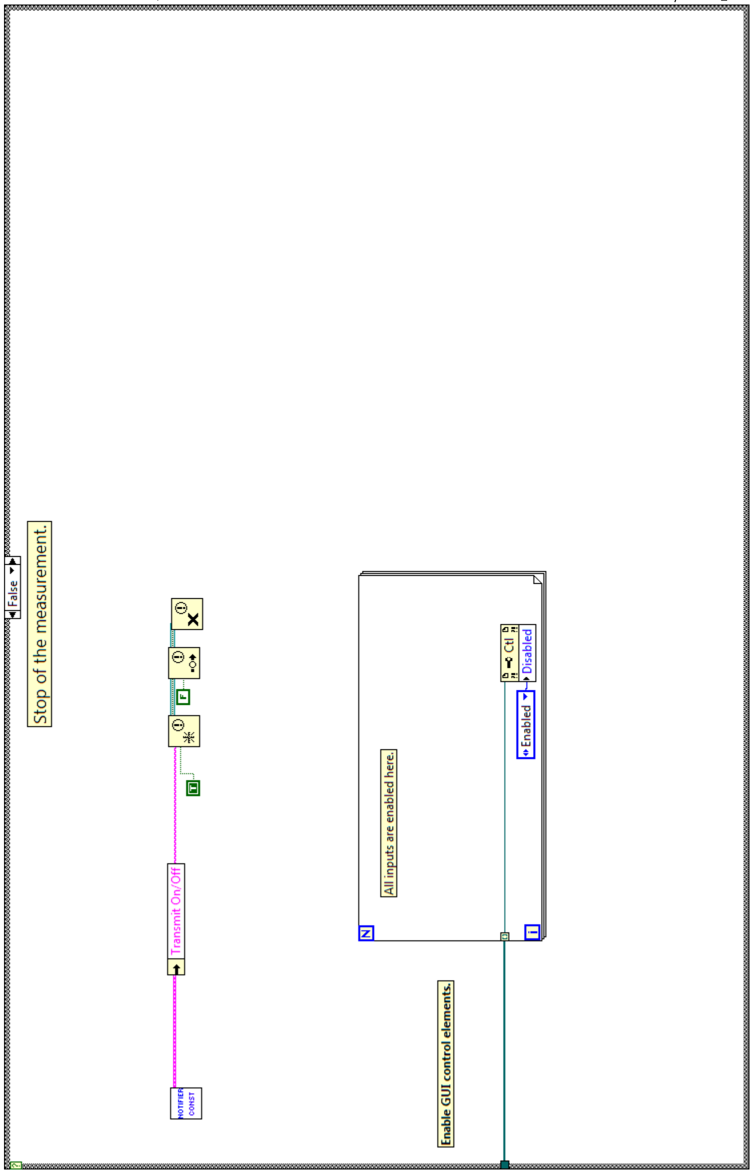


Event handler for all user events, e.g. button pressed.
[1] "Start / Stop measurement": Value Change
Start / Stop of the measurement.
Start of the measurement.
Measurement settings OK, start measurement.
Send noise measurement command to the radar control block, set transmit on status.
Disable GUI control elements.
All inputs are disabled here.
Clear Intensity Charts
Initialize the header builder.
Store the references of the controls and of the intensity charts in a functional global VI once on start-up of the GUI.
References to all the graph properties and input values are bundled into a functional global variable VI

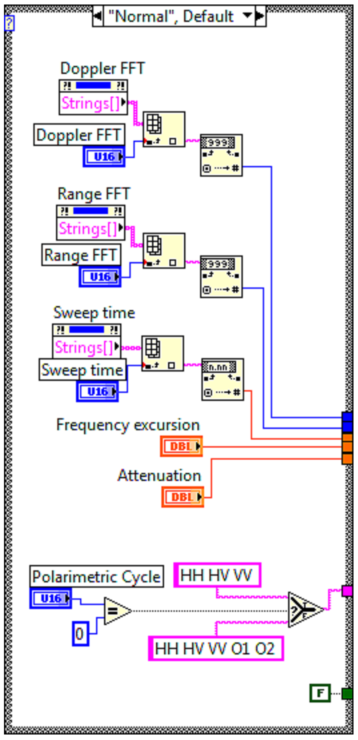

Update of the visualisation (intensity charts).
Tabcontrol of the Visualisation.
Here all the Z scales are going to be set individually
Here the data goes the the graphs
Here all the properties for the x and y scales of the graphs will be set
Get
GUI REFNUM SELECT
IntChart (strict)
Y Scale.Range:Maximum
Y Scale.Range:Minimum
YScale.Maximum
YScale.Minimum
YScale.Multiplier
502
Zhh
Vhh
Whh
ZDR
LDR
|rho|
Horizontal speed
Vertical speed
Orientation
ZScale.MarkerVals[]
Axess notifier in
Visualization for GUI notifier in


Display system comments to the user.
error queu in
System comments
source
Text.Text

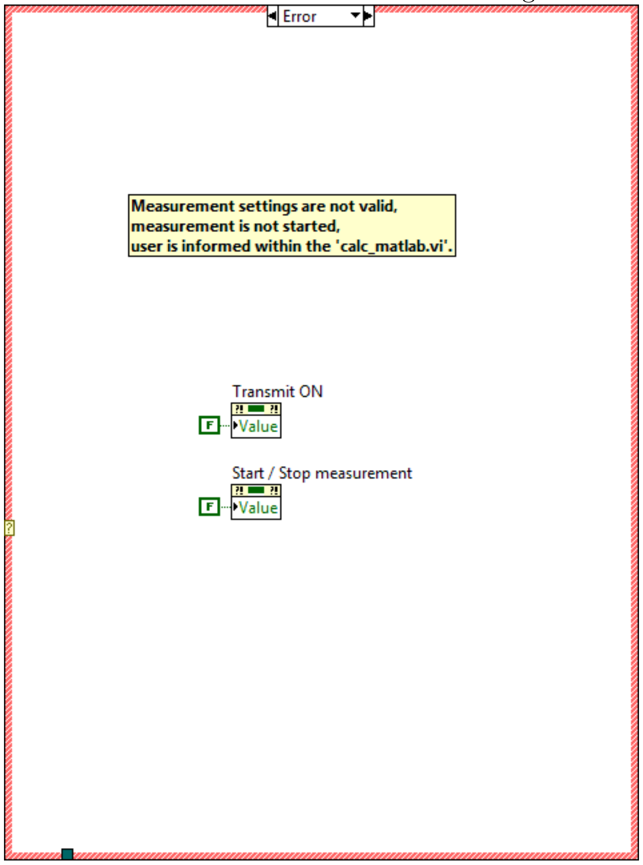
The state, the state machine will enter when the start/stop measurement switch has been switched off.



The state for the normal measurement settings for the Polarization Cycle.

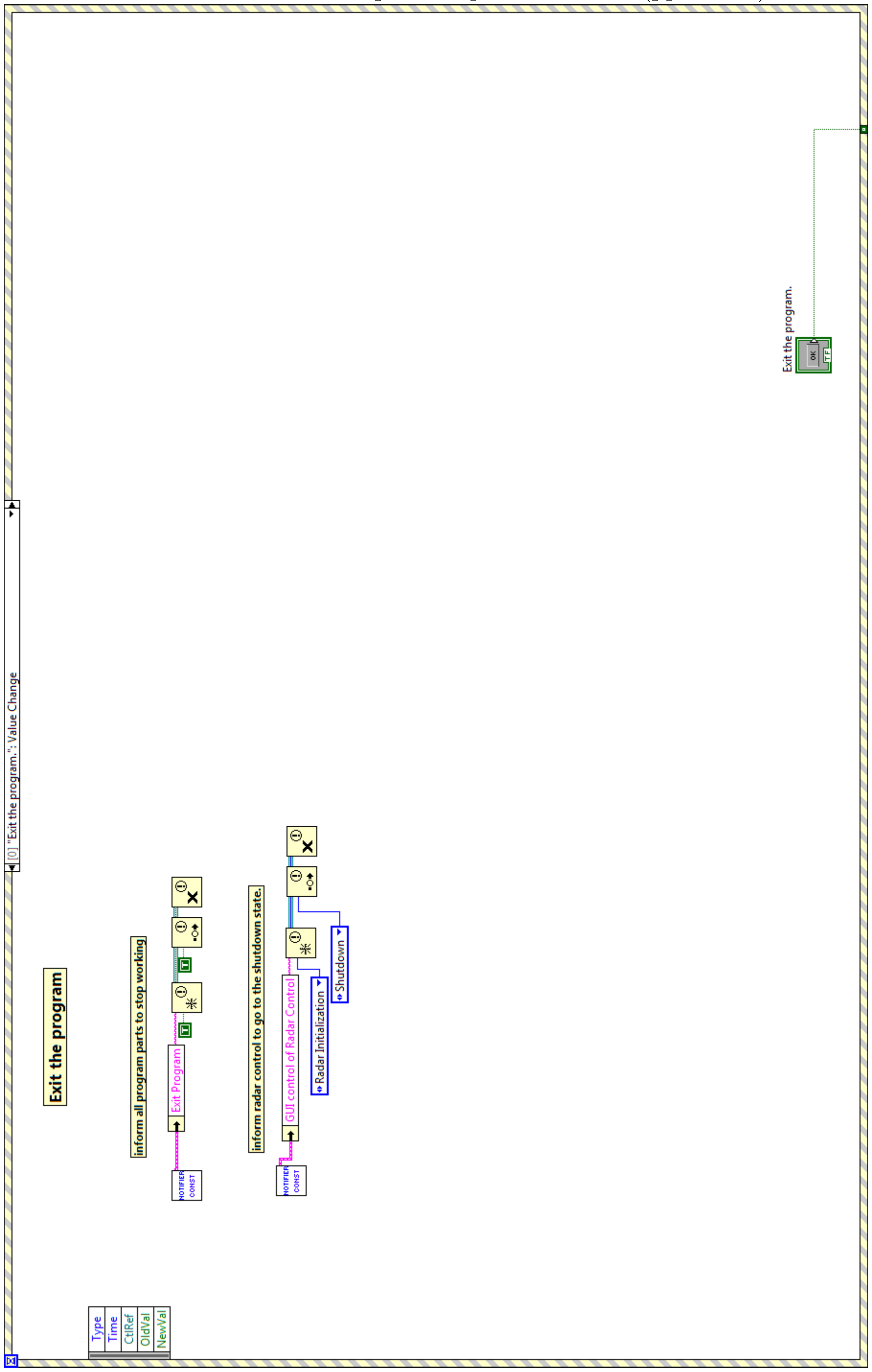


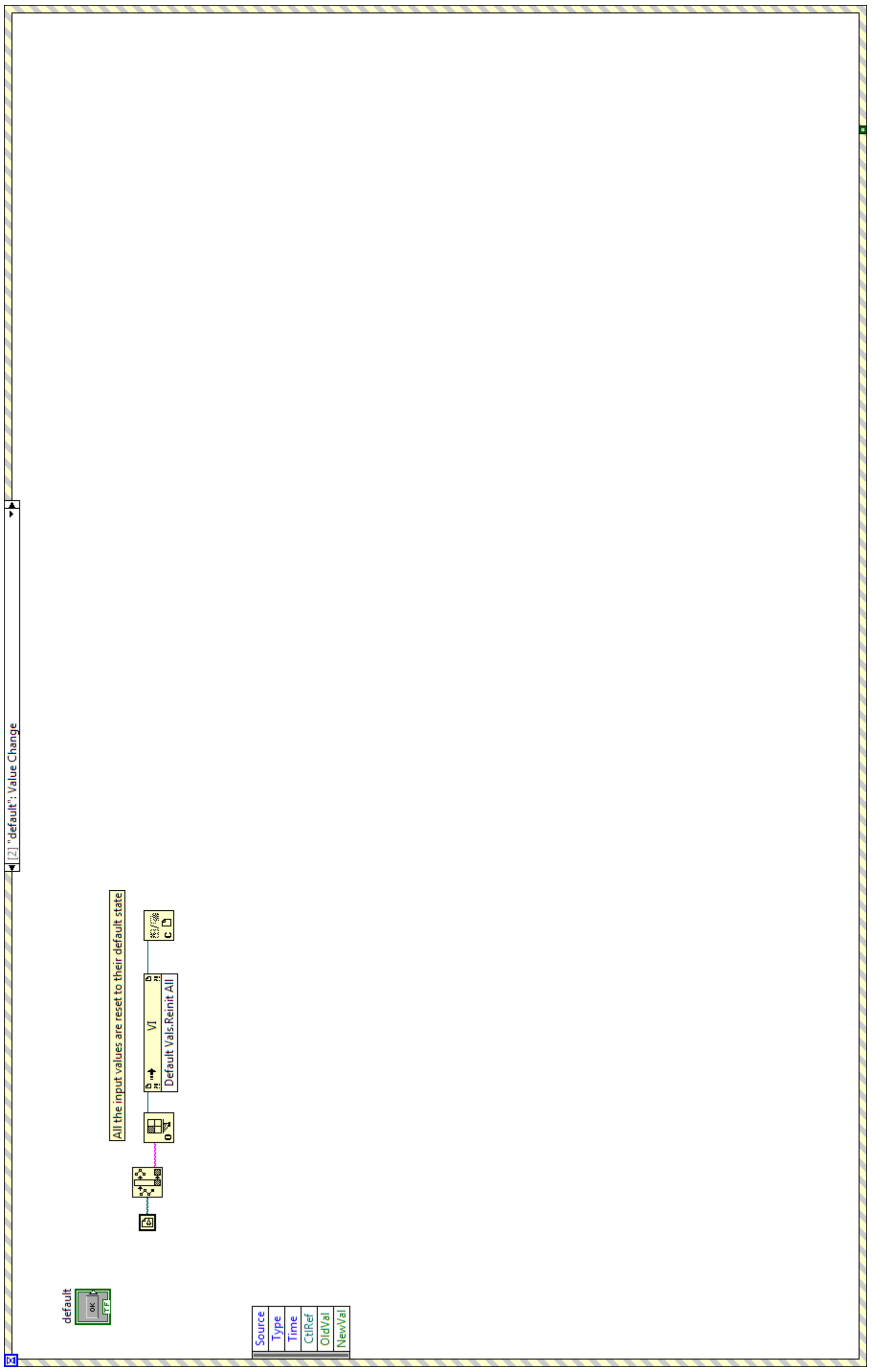
Error state when the measurement settings are incorrect

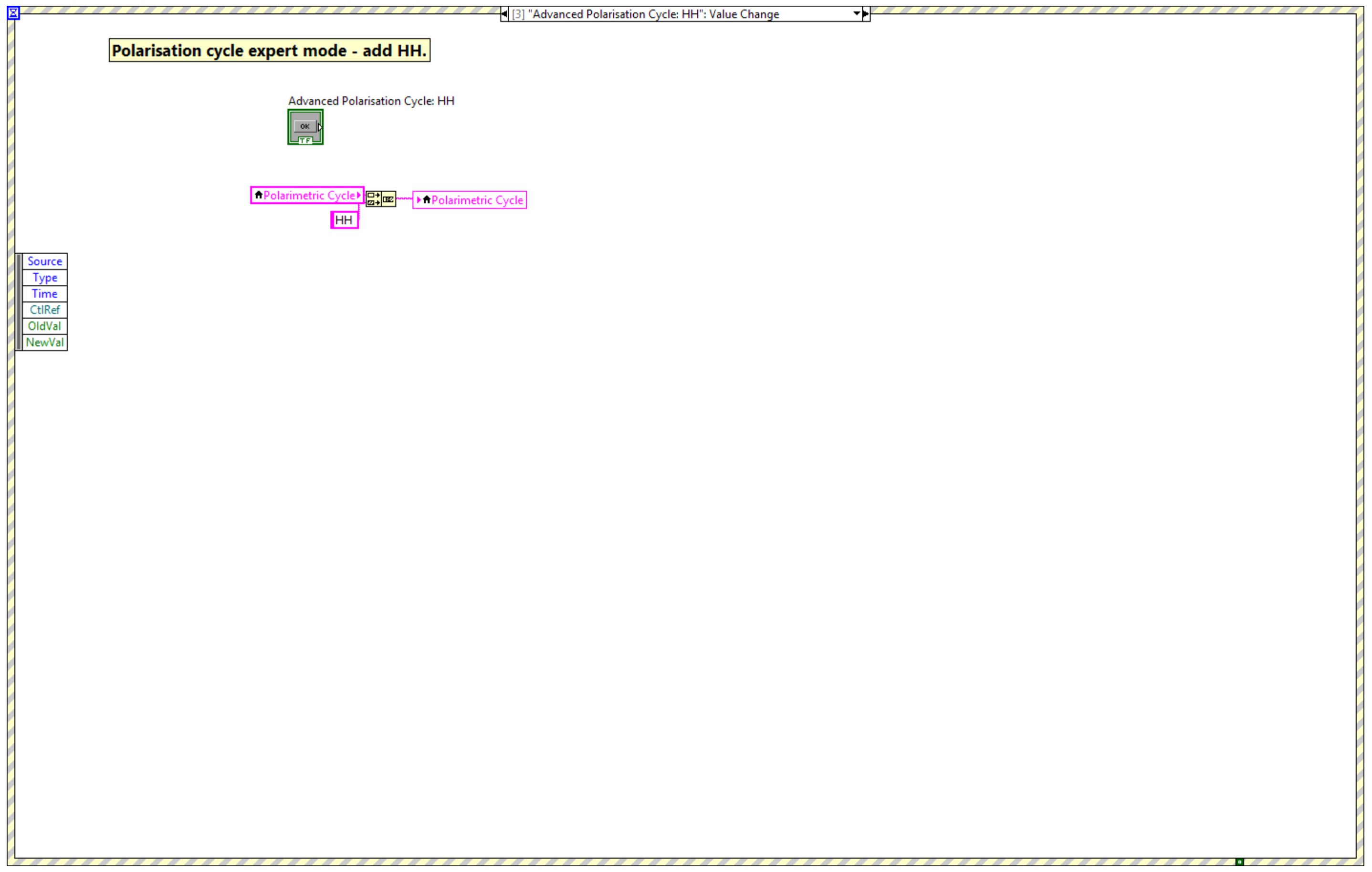


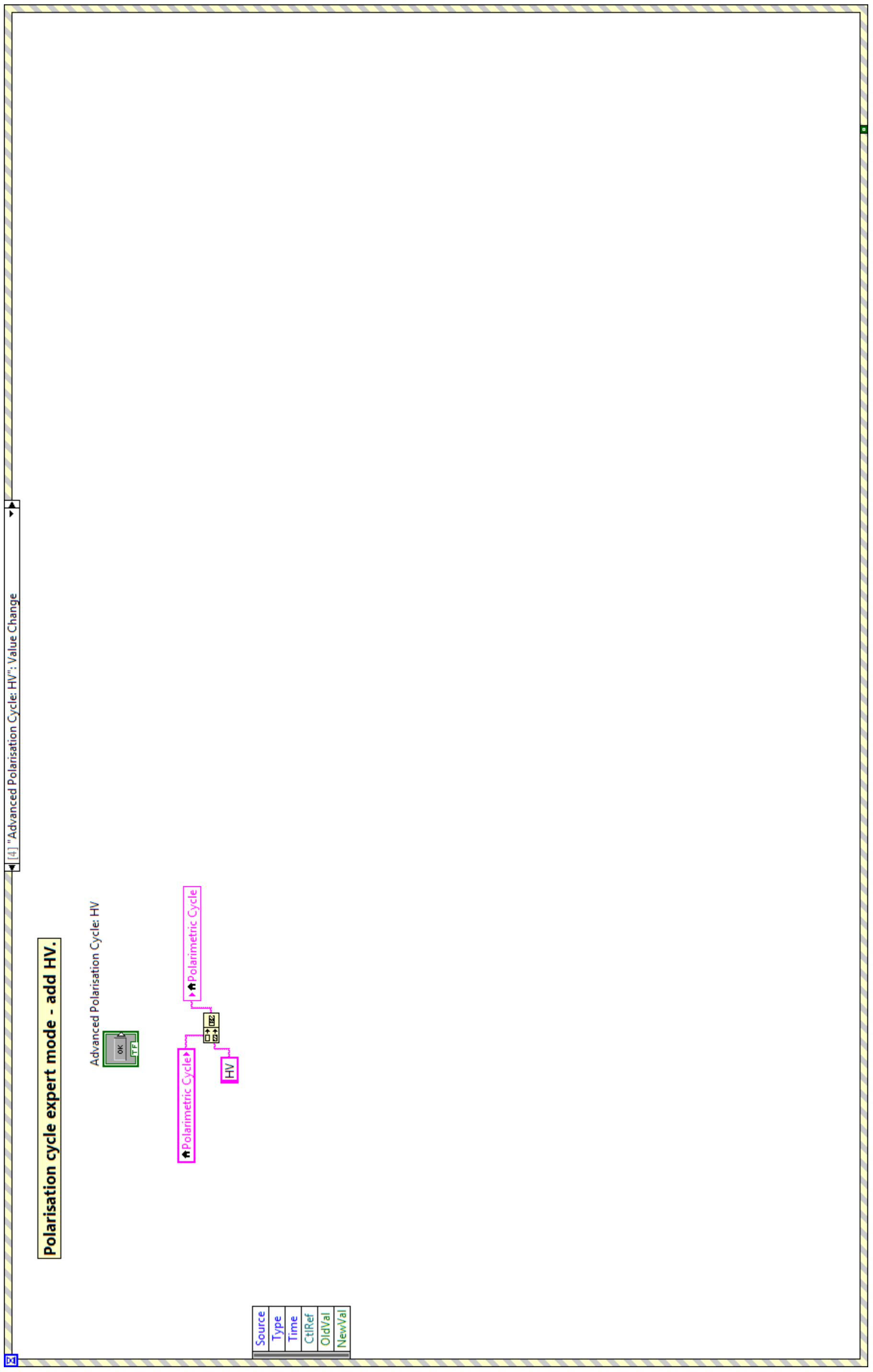
Measurement settings are not valid,
measurement is not started,
user is informed within the 'calc_matlab.vi'.

The events for when the user presses specific buttons (pp. 45-52).

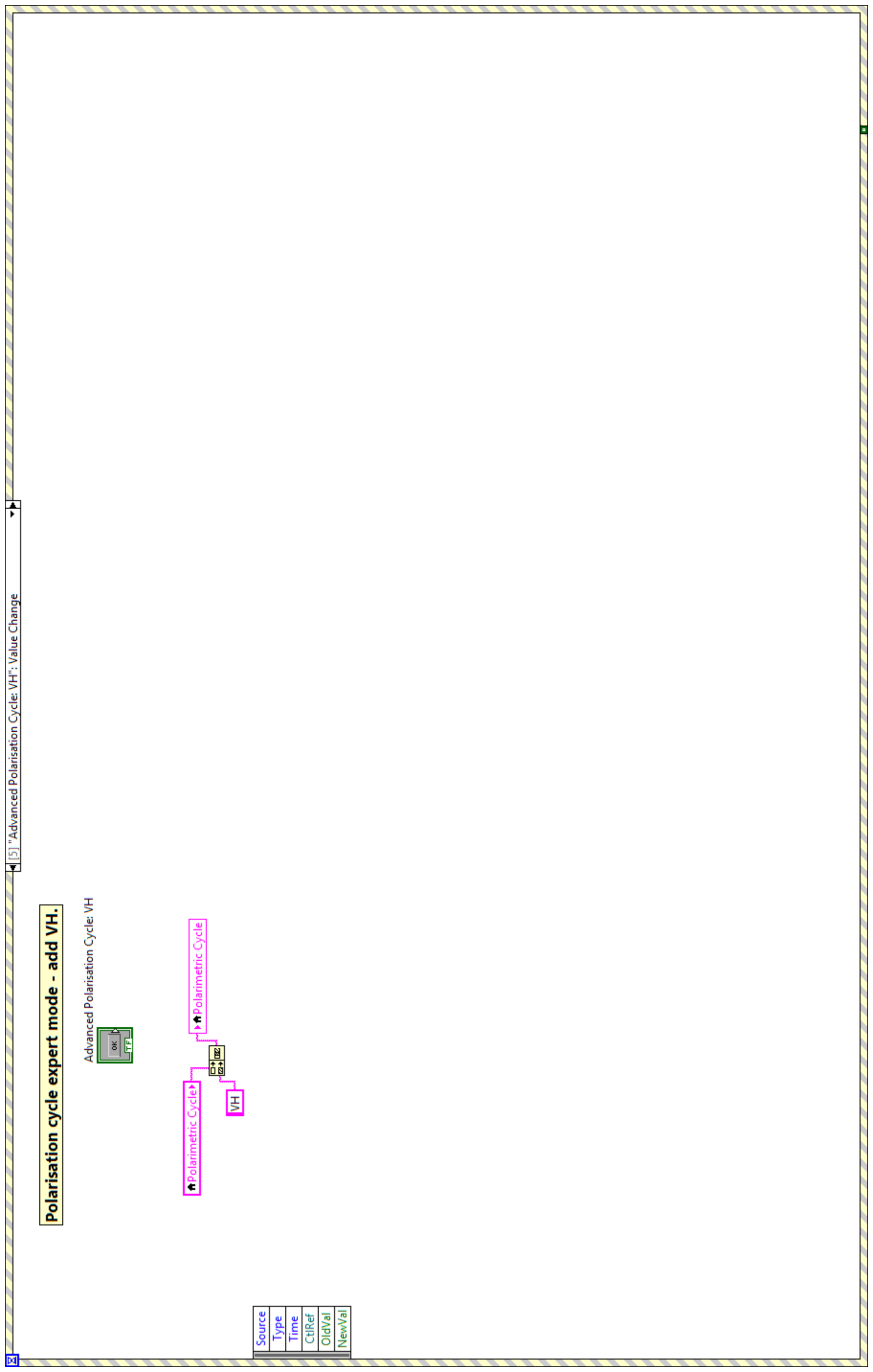

[0] "Exit the program.": Value Change
Exit the program
inform all program parts to stop working
Exit Program
inform radar control to go to the shutdown state.
GUI control of Radar Control
Radar Initialization
Shutdown
NOTIFIER CONST
Type
Time
CtlRef
OldVal
NewVal
Exit the program.
OK
TF

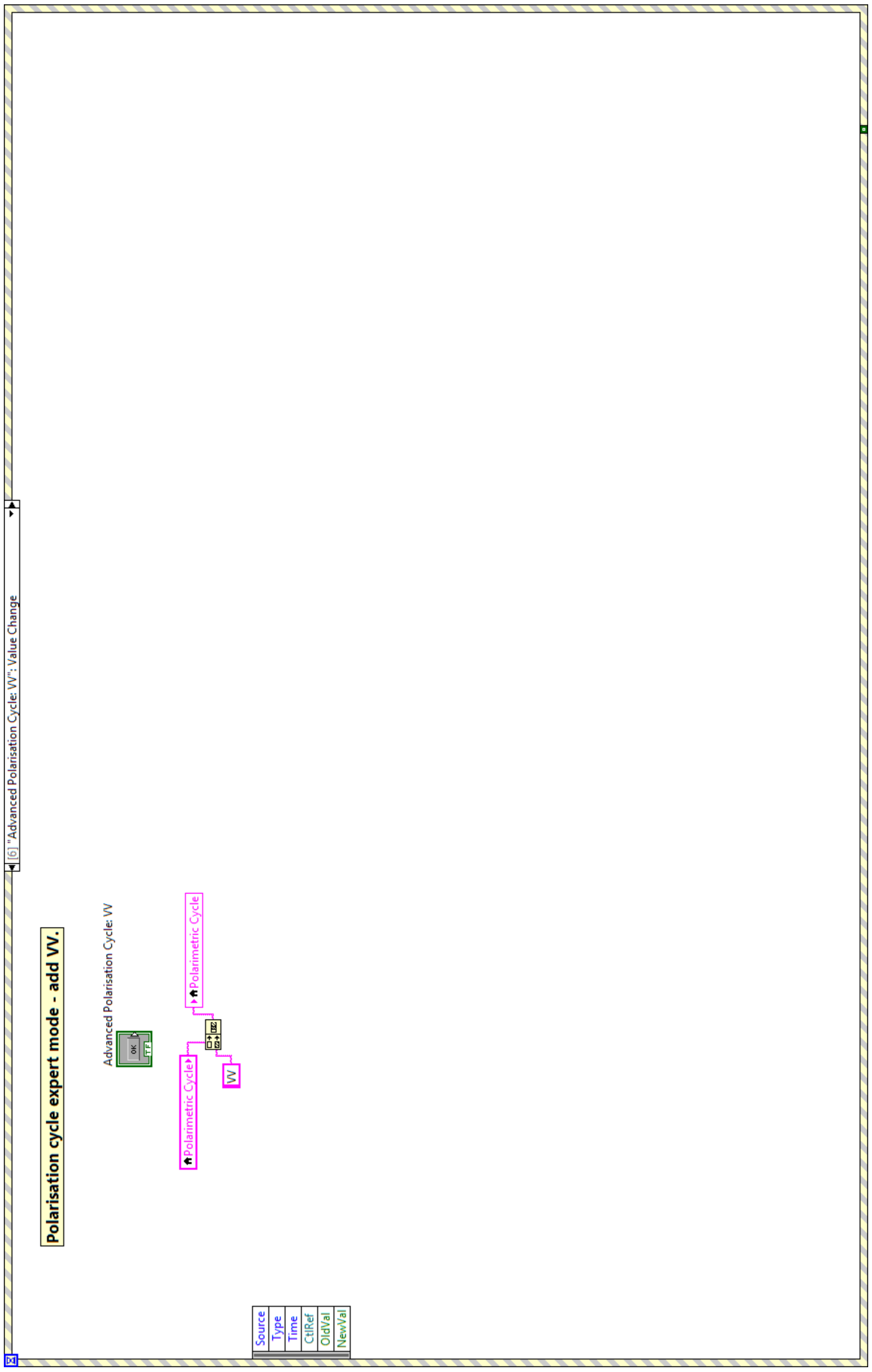

[2] "default": Value Change
default
OK
TF
All the input values are reset to their default state
VI
Default Vals.Reinit All
Source
Type
Time
CtlRef
OldVal
NewVal

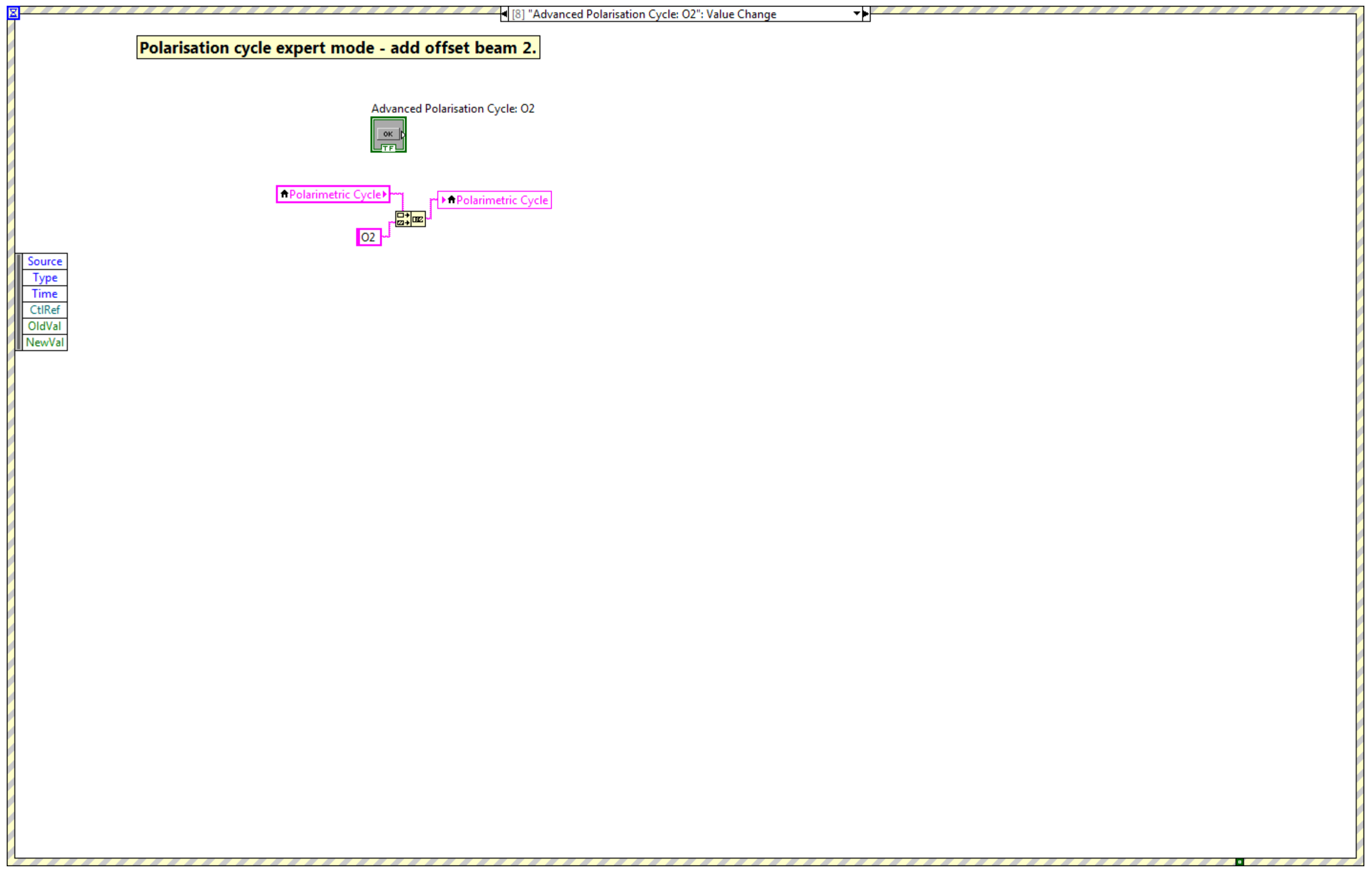

[3] "Advanced Polarisation Cycle: HH": Value Change
Polarisation cycle expert mode - add HH.
Advanced Polarisation Cycle: HH
OK
Polarimetric Cycle
HH
Polarimetric Cycle
Source
Type
Time
CtlRef
OldVal
NewVal

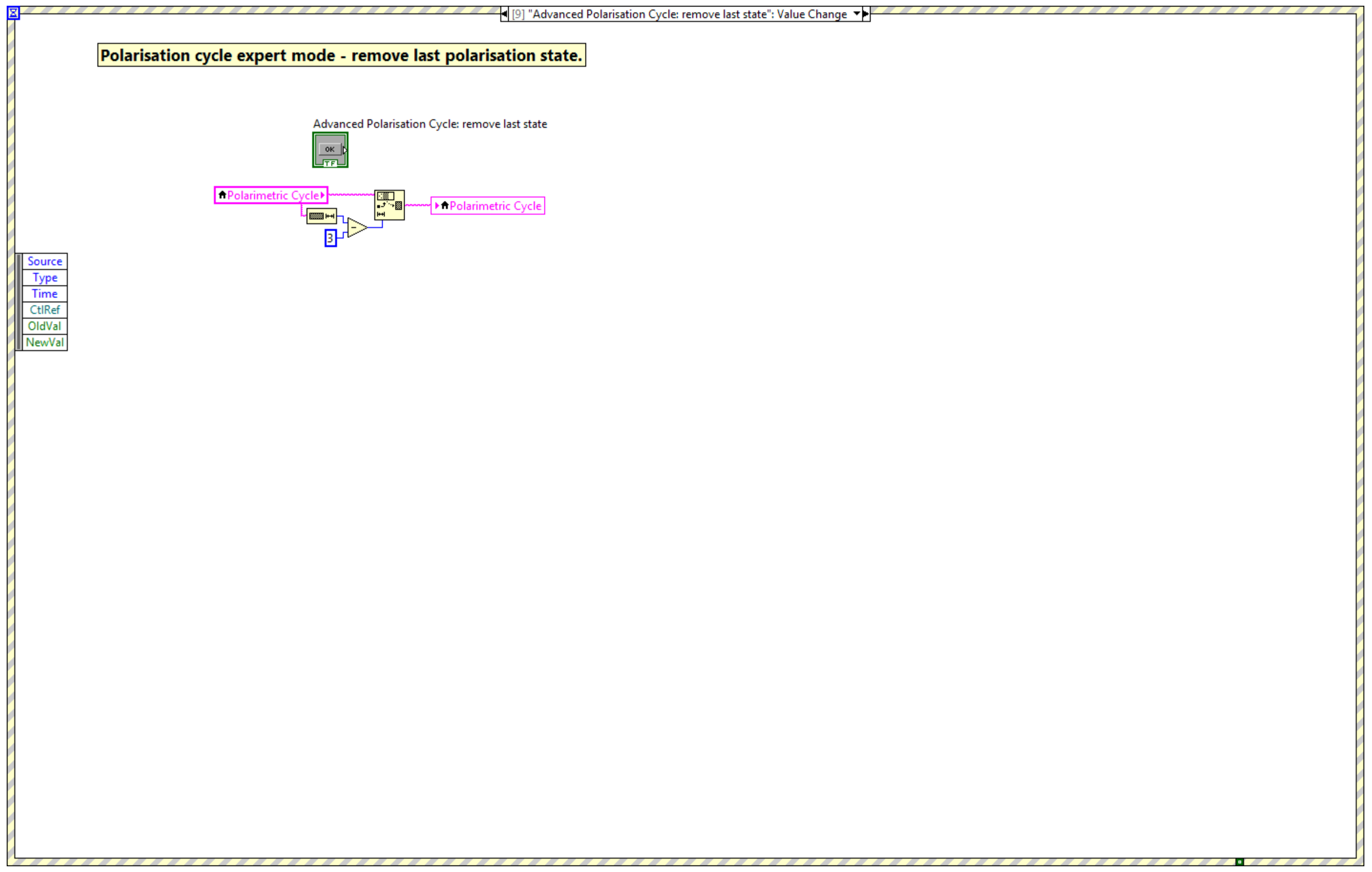


[4] "Advanced Polarisation Cycle: HV": Value Change
Polarisation cycle expert mode - add HV.
Advanced Polarisation Cycle: HV
Polarimetric Cycle
Polarimetric Cycle
HV
Source
Type
Time
CtlRef
OldVal
NewVal

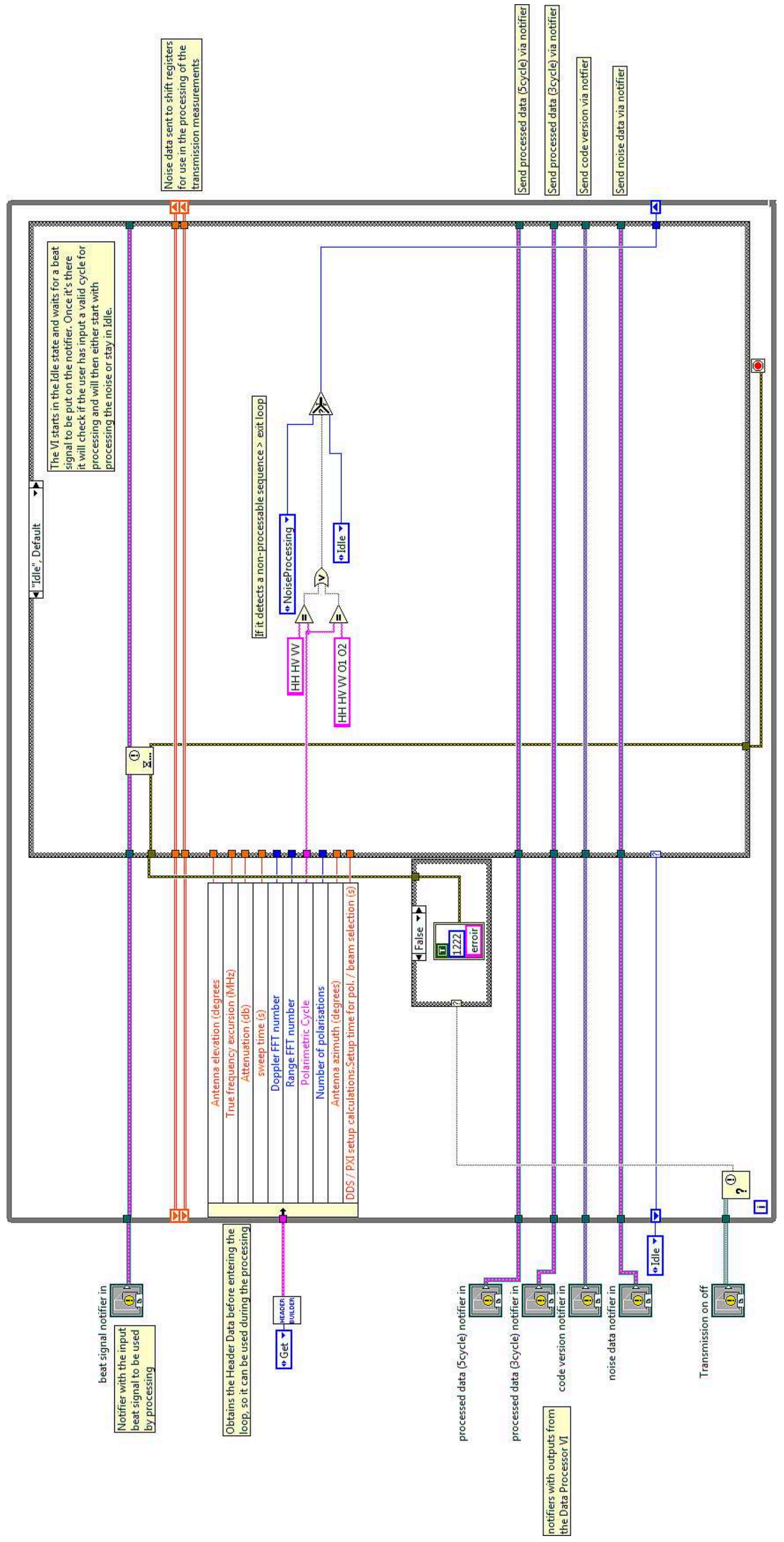

[5] "Advanced Polarisation Cycle: VH": Value Change
Polarisation cycle expert mode - add VH.
Advanced Polarisation Cycle: VH
OK
TF
Polarimetric Cycle
Polarimetric Cycle
VH
Source
Type
Time
CtlRef
OldVal
NewVal

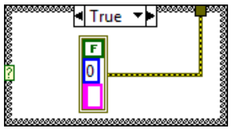

[6] "Advanced Polarisation Cycle: VV": Value Change
Polarisation cycle expert mode - add VV.
Advanced Polarisation Cycle: VV
Polarimetric Cycle
Polarimetric Cycle
VV
Source
Type
Time
CtlRef
OldVal
NewVal


[8] "Advanced Polarisation Cycle: O2": Value Change
Polarisation cycle expert mode - add offset beam 2.
Advanced Polarisation Cycle: O2
Polarimetric Cycle
O2
Polarimetric Cycle
Source
Type
Time
CtlRef
OldVal
NewVal

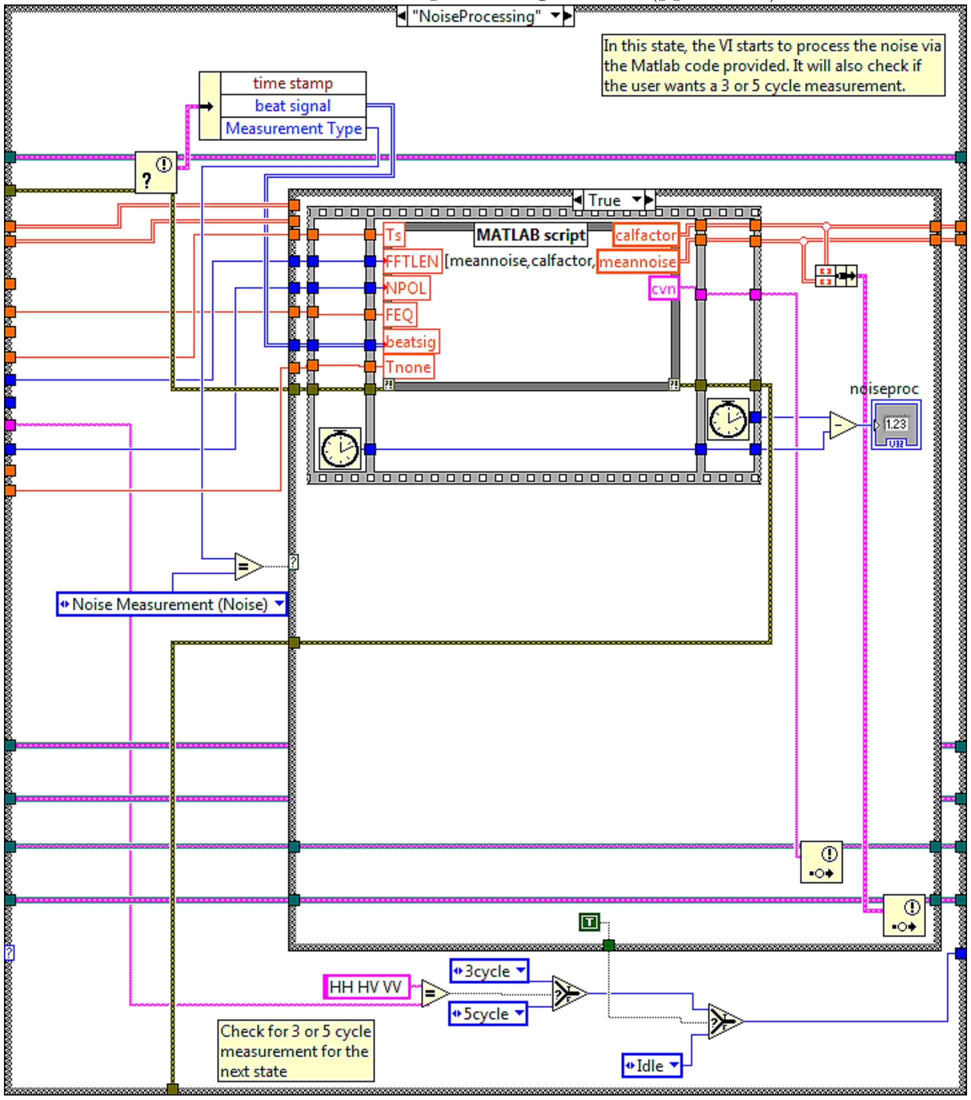

[9] "Advanced Polarisation Cycle: remove last state": Value Change
Polarisation cycle expert mode - remove last polarisation state.
Advanced Polarisation Cycle: remove last state
Polarimetric Cycle
Polarimetric Cycle
3
Source
Type
Time
CtlRef
OldVal
NewVal

## Data processor


Noise data sent to shift registers for use in the processing of the transmission measurements
Send processed data (5cycle) via notifier
Send processed data (3cycle) via notifier
Send code version via notifier
Send noise data via notifier
"Idle", Default
The VI starts in the Idle state and waits for a beat signal to be put on the notifier. Once it's there it will check if the user has input a valid cycle for processing and will then either start with processing the noise or stay in Idle.
If it detects a non-processable sequence > exit loop
NoiseProcessing
Idle
HH HV VV
HH HV VV O1 O2
False
1222
erroir
Antenna elevation (degrees
True frequency excursion (MHz)
Attenuation (db)
sweep time (s)
Doppler FFT number
Range FFT number
Polarimetric Cycle
Number of polarisations
Antenna azimuth (degrees)
DDS / PXI setup calculations.Setup time for pol. / beam selection (s)
beat signal notifier in
Notifier with the input beat signal to be used by processing
Get
HEADER BUILDER
Obtains the Header Data before entering the loop, so it can be used during the processing
processed data (5cycle) notifier in
processed data (3cycle) notifier in
code version notifier in
noise data notifier in
Idle
Transmission on off
notifiers with outputs from the Data Processor VI

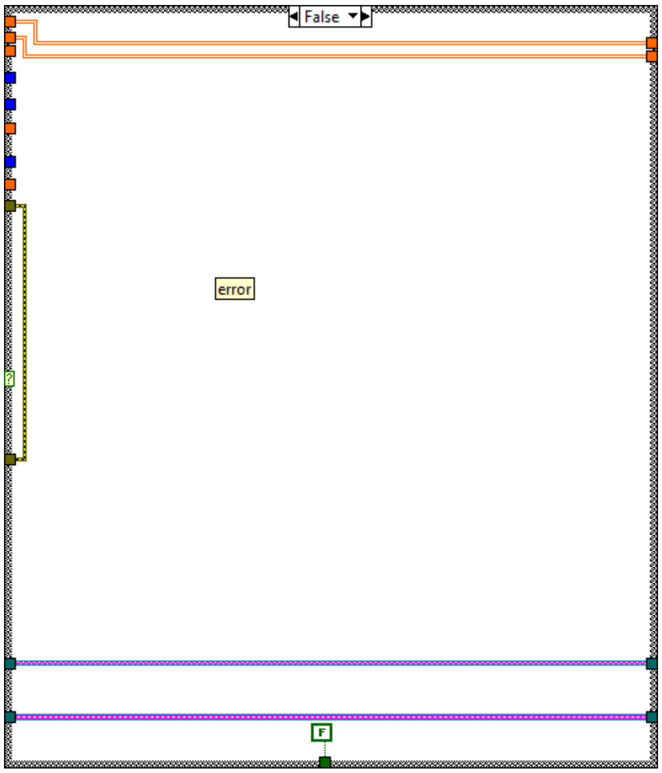


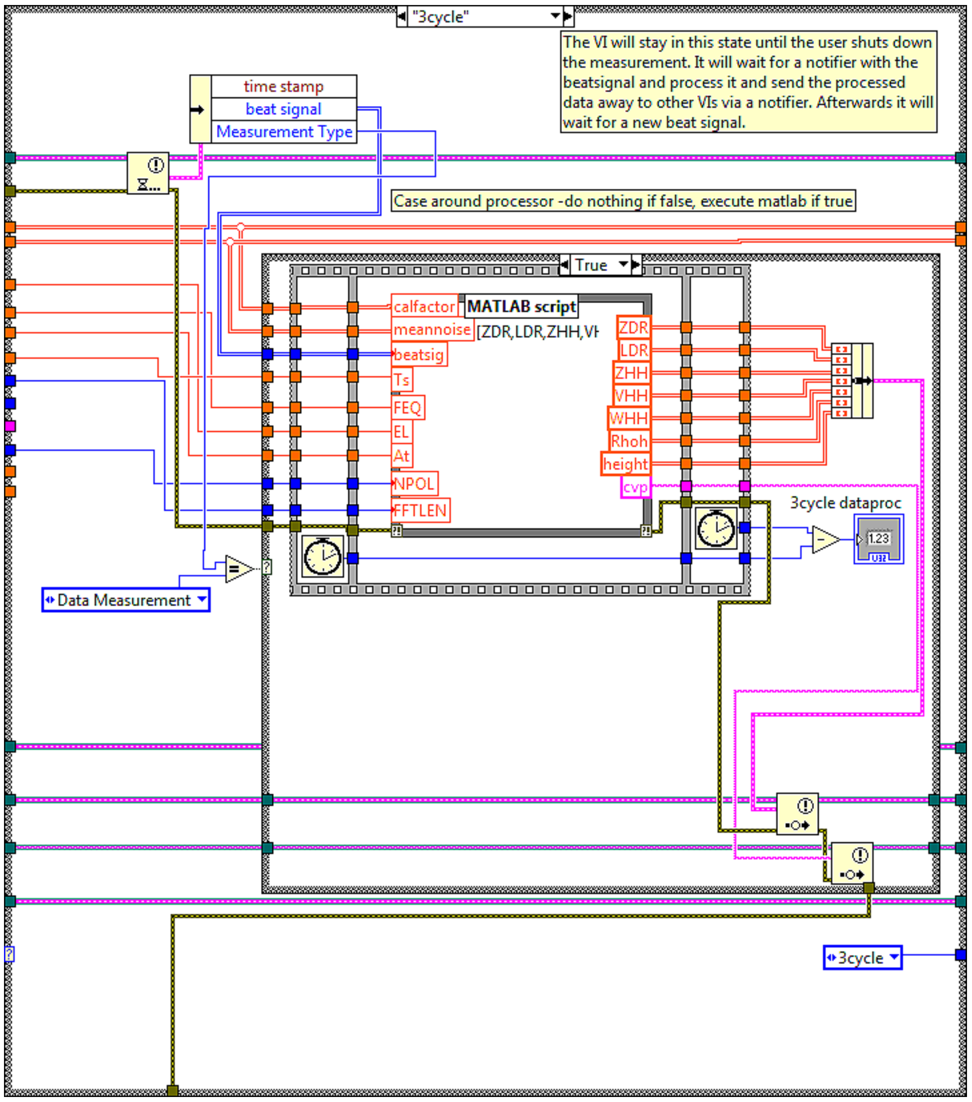
The different states for the different processing actions (pp. 54-58).



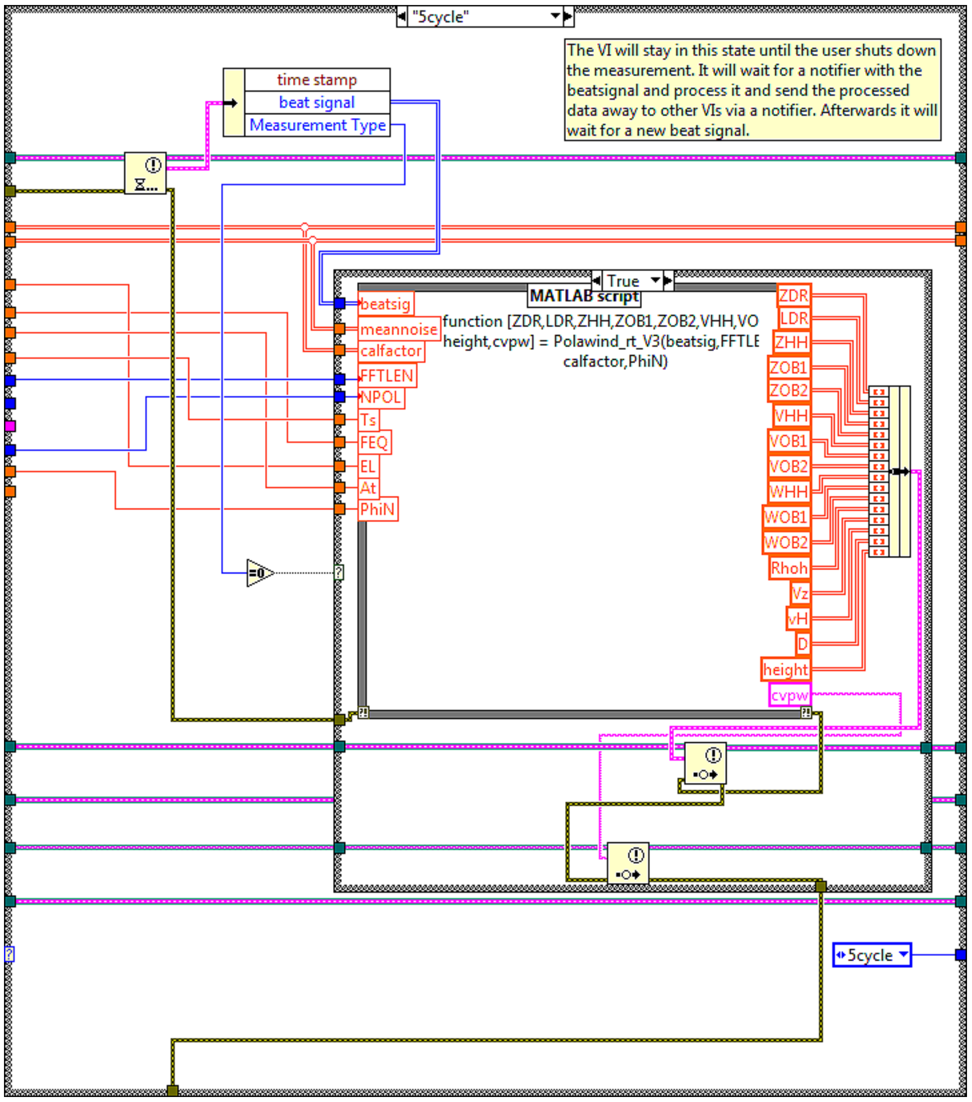
In this state, the VI starts to process the noise via the Matlab code provided. It will also check if the user wants a 3 or 5 cycle measurement.

Check for 3 or 5 cycle measurement for the next state


False
error
?
F

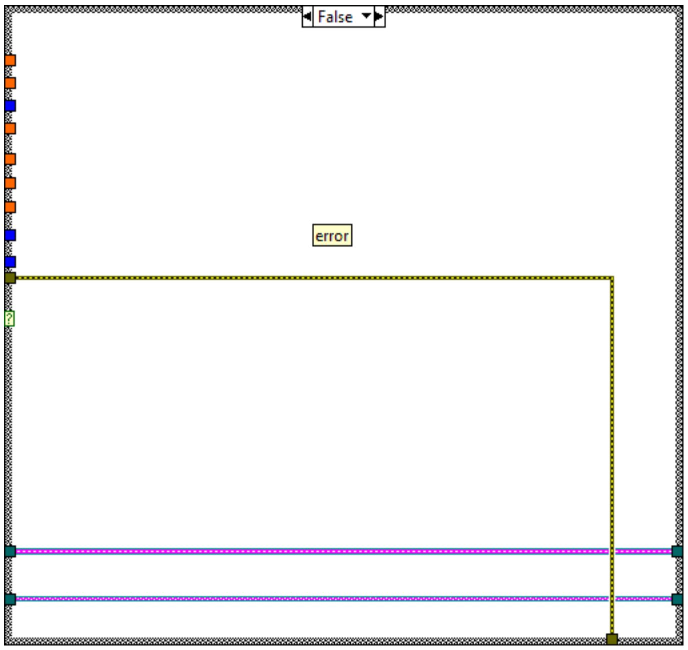


The VI will stay in this state until the user shuts down the measurement. It will wait for a notifier with the beatsignal and process it and send the processed data away to other VIs via a notifier. Afterwards it will wait for a new beat signal.

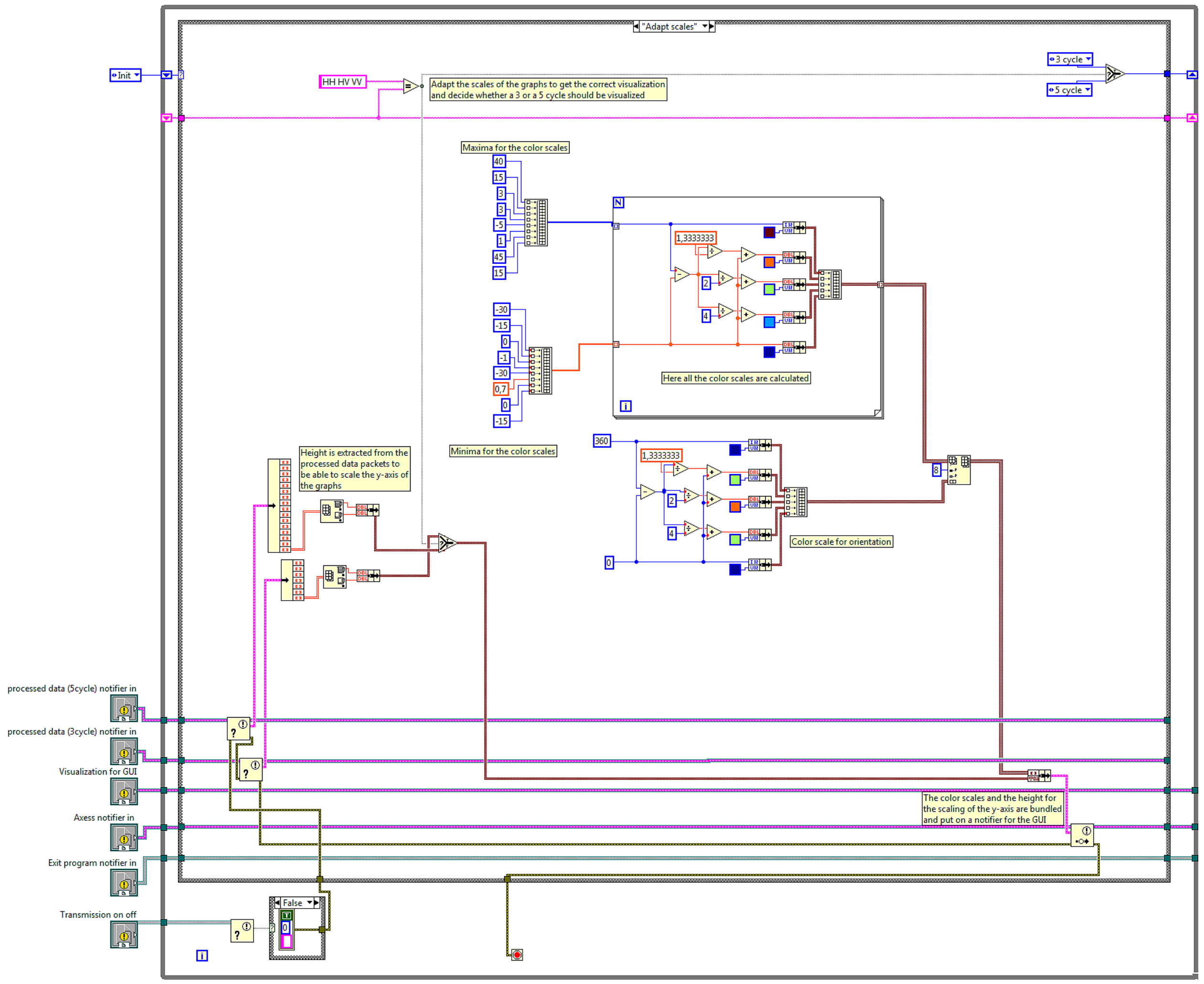


The VI will stay in this state until the user shuts down the measurement. It will wait for a notifier with the beatsignal and process it and send the processed data away to other VIs via a notifier. Afterwards it will wait for a new beat signal.

```
function [ZDR,LDR,ZHH,ZOB1,ZOB2,VHH,VO
height,cvpw] = Polawind_rt_V3(beatsig,FFTLE
calfactor,PhiN)
```


False
error

## Visualisation


Adapt the scales of the graphs to get the correct visualization and decide whether a 3 or a 5 cycle should be visualized

Maxima for the color scales

Minima for the color scales

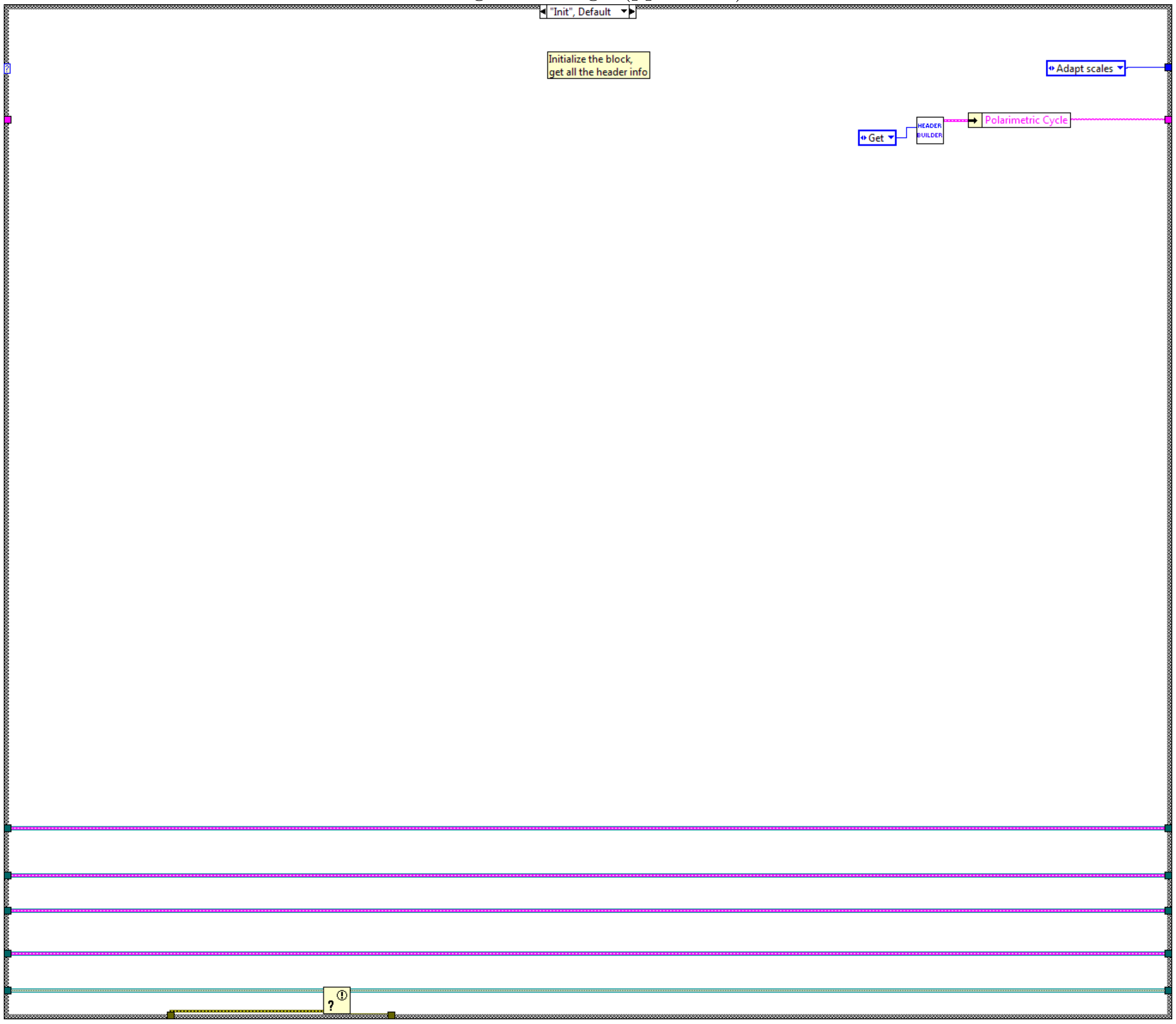
Here all the color scales are calculated

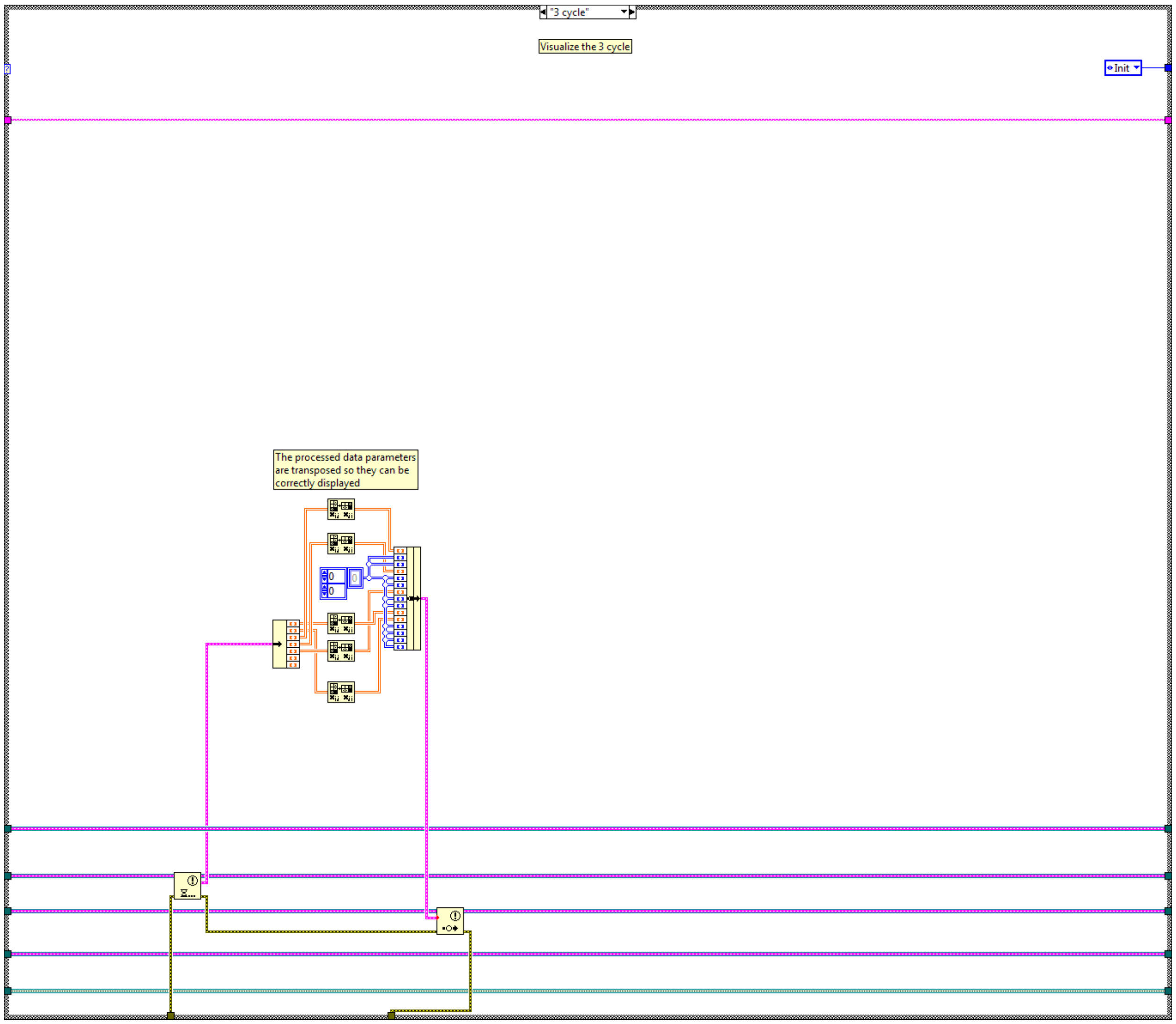
Height is extracted from the processed data packets to be able to scale the y-axis of the graphs

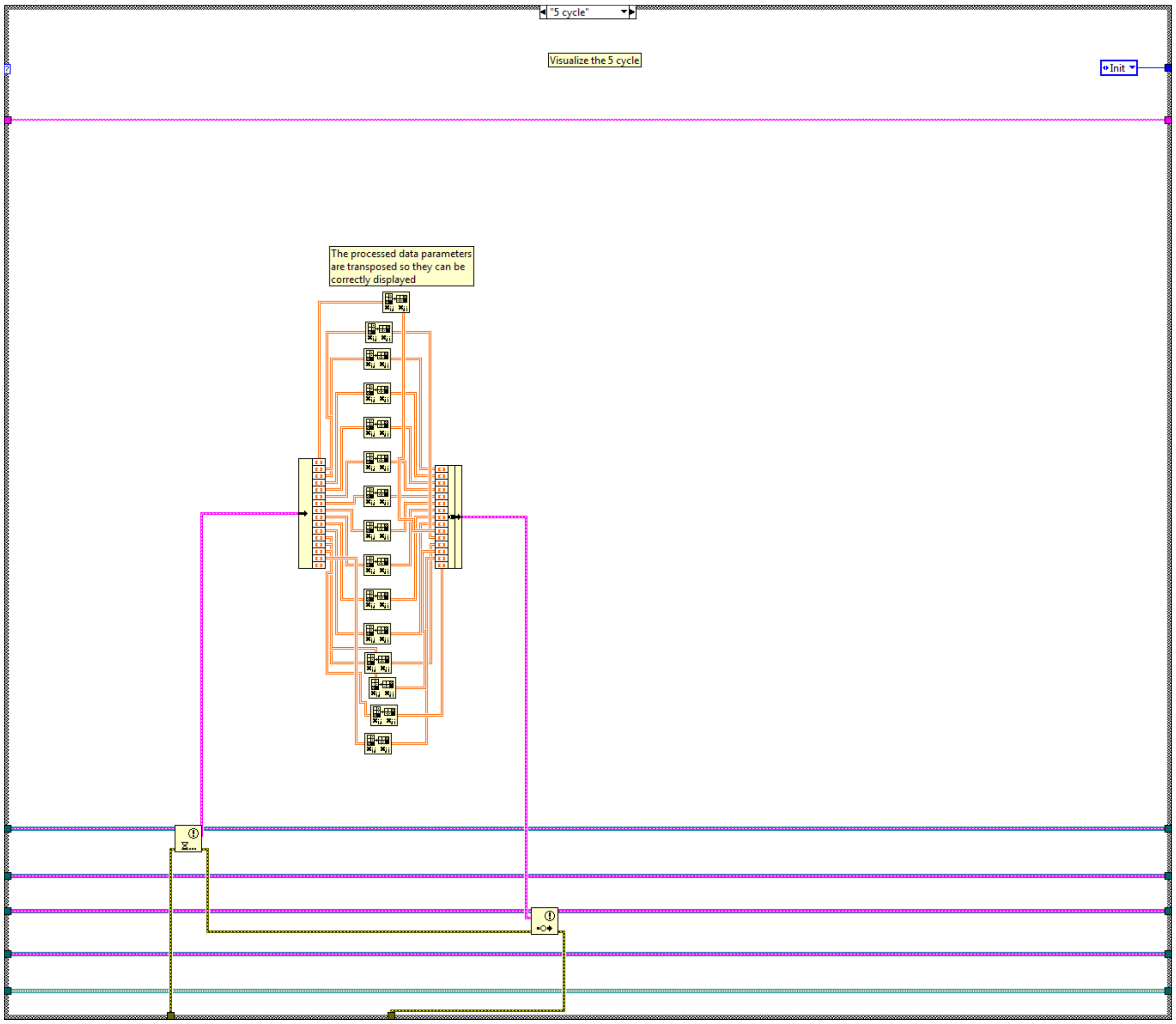
Color scale for orientation

The color scales and the height for the scaling of the y-axis are bundled and put on a notifier for the GUI

The additional states the Visualization goes through (pp. 60-62).

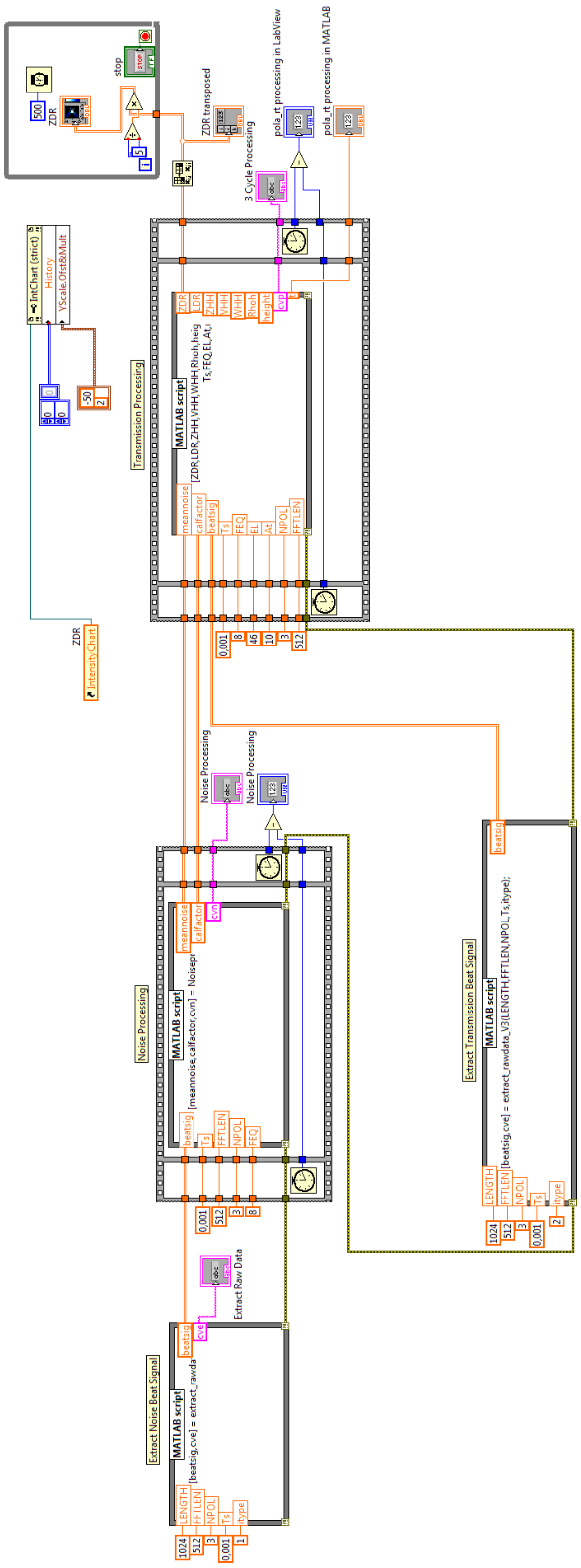



"3 cycle"
Visualize the 3 cycle
Init
The processed data parameters are transposed so they can be correctly displayed
0
0


"5 cycle"
Visualize the 5 cycle
Init
The processed data parameters are transposed so they can be correctly displayed

## Matlab prosessing test

The test VI used to test the Matlab nodes containing the Processing codes with some test data.


stop
ZDR
ZDR transposed
3 Cycle Processing
pola_rt processing in LabView
pola_rt processing in MATLAB
Transmission Processing
MATLAB script
[ZDR,LDR,ZHH,VHH,WHH,Rhoh,heig
Ts,FEQ,EL,At,
IntChart (strict)
History
YScale.Ofst&Mult
IntensityChart
Noise Processing
[meannoise,calfactor,cvn] = Noisep
Extract Transmission Beat Signal
[beatsig,cve] = extract_rawdata_V3(LENGTH,FFTLEN,NPOL,Ts,itype);
Extract Noise Beat Signal
[beatsig,cve] = extract_rawda
Extract Raw Data
LENGTH
FFTLEN
NPOL
Ts
itype
beatsig
cve
meannoise
calfactor
cvn
cvp
FEQ
EL
At
height

# Appendix E

# Examples of the implemeted Matlab code

## Noiseprocesor real time: V4

```
function [meannoise,calfactor,cvn] = Noiseprocessor_rt_V4(beatsig,FFTLEN,...
                                                       NPOL,Ts,FEQ,Tnone)

% Noiseprocessor real time : V4
% C. Unal. March-April 2011 (Noiseprocessor)
% Reads raw data of TARA noise measurements (with the transmitter switched
% off)
% Uses the standard processing scheme (windowing and FFT's) to obtain
% (a) the calibration factor, (b) the mean noise
% The calibration factor is polarimetrically corrected as follows:
% (1) The transmitter imbalance is post corrected (H > V, 0.35 dB)
% (2) The receiver imbalance is obtained from the measured data (H > V,
% 0.19 dB)
% Modifications: Y. Dufournet en C. Unal. May 2011 (real-time interface)

%INPUT
%beatsig  : 2D array raw data [no units] (LENGTH, number of sweeps)
%FFTLEN   : Doppler FFT-length
%NPOL     : Number of polarisation/beam states
%Ts       : Sweep time [s]
%FEQ      : frequency excursion [MHz]
%Tnone    : time [s] within one sweep [s] where no data are collected
%            (fixed for the moment)

%OUTPUT
%meannoise: measured mean noise level not calibrated
%calfactor: calibration factor
%cvn      : version of the code used

warning off all
%tic
%**********************************************************************
```

```
%number of profiles determination
[LENGTH,sizey] = size(beatsig);
NPROFS = fix(sizey/(FFTLEN*NPOL)); %determined from the radar control block
%(how much data is sent by the radar control?)
cvn = 'noise_V4';
%***********************************************************************

% Calculation of the Radar constant
K2      = 0.93;                 % |K|^2 related to water permittivity
beamw   = 2.1*pi/180;           % beamwidth [deg]
GdB     = 38.8;                 % antenna gain [dB]
G       = 10^(GdB/10);          % antenna gain (linear)
PJ      = 512 * log(2);         % Probert-Jones correction factor
FF      = 3298e6;               % TARA carrier frequency [Hz]
CC      = 3.e8;                 % light speed [m/s]
L       = CC/FF;                % radar wavelength [m]
Zc      = 1e18;                 % dbZ conversion factor
Dr      = 3e8/(2*FEQ*1e6);      % range resolution [m]
% radar constant C with dbz conversion factor included
Const   = (PJ*L^2*Zc)/(pi^3*G^2*beamw^2*K2*Dr);
% the radar resolution volume must be multiplied by exp(-C2/r^2) to account
% for the non-overlapping of the antennas in the near range.This correction
% takes in account the spacing between the two antennas of Delta=5.52 m,
% the parallax error (<0.5 deg) is assumed neglectable (transmit and
% receive antenna are looking in the same direction and have the same
% beamwidths), Delta << range (minimum value 100 m).
Delta = 5.52;
C2 = 2*log(2)*Delta^2/beamw^2;

% Transmit power outside the antennas
Pt = 41.3;                     % in watts
% Tnone= 0.0001;
 Tsc=Ts-Tnone;                % no data collected at the begin of a sweep
%Tsc=(7/8)*Ts;                  % old version

% Calculation of the system noise
bolt    = 1.38e-23;            % Boltzman's constant
NfdB    = 1;                   % noise figure [dB]
Nf      = 10^(NfdB/10);        % noise figure (linear)
Ta      = 50;                  % antenna temperature [K]
Tr      = 290;                 % room temperature [K]
Tsys    = Ta+Tr*(Nf-1);        % system temperature [K]
Bandw   = 1./Tsc;              % system bandwidth
Pcalc   = bolt*Tsys*Bandw;     % Calculated noise power

% Calculation of the measured noise reflectivity (Zcalc)
x = 1:LENGTH/2;
range = x*Dr-Dr;               % range starts at 0 m
% range bin 1 artificially put to 1 m to avoid 0 value for Zcalc
range(1) = 1;
range2= range.*range;
```

```
Zcalc = (Pcalc/Pt) * Const * range2;
% First 100 m of range: backscattered signal not reliable
zabo = find(range>100);
br = zabo(1);
% near range correction for the physical separation of the antennas
Zcalc(br:LENGTH/2) = Zcalc(br:LENGTH/2).*exp(C2./range2(br:LENGTH/2));


% ***********************************************************************
% ************************ NOISE PRE-PROCESSING *************************
% ***********************************************************************
% The noise pre-processing should be identical to the atmospheric data pre-
% processing in order to absolutely calibrate.

% create the window for range FFT(Hamming)
y = hamming(LENGTH);
z = ones(1,FFTLEN);
% LENGTH by FFTLEN - FFTLEN sweeps of LENGTH samples, each sweep in column
winr = kron(z,y);

% create window for Doppler FFT (Hamming)
y = hamming(FFTLEN);
z = ones(1,LENGTH/2);
% FFTLEN by LENGTH/2 - LENGTH/2 range gates with FFTLEN sweeps
% in each to do FFTLEN point FFT for each gate
wind = kron(z,y);

calnoise = zeros(NPROFS,LENGTH/2,NPOL);


%**********************    PROFILE LOOP    ******************************
 % *********************************************************************
for ns = 1:NPROFS;

    %read in data block for one profile
    allbeat=beatsig(1:LENGTH,((ns-1)*FFTLEN*NPOL)+1:ns*FFTLEN*NPOL);

    %allbeat=fread(fid,[LENGTH,FFTLEN*NPOL],'short');   %older version

    % Run through all polarization/beam states
    for np = 1:NPOL

        % extract the beat signal for one polarization state/beam
        % polarization/beam changes from sweep to sweep
        beat = allbeat(:,np:NPOL:FFTLEN*NPOL);

        % remove the DC component from the beat signal
        z = ones(LENGTH,1);
        bm = mean(beat);
        beat = beat - kron(z,bm);
        % do range FFT with windowing
        beat = beat.*winr;
        chwin = fft(beat);

        % discard image half of matrix, and take transpose, since we will
        % now do FFT over other dimension. Make both components zero mean,
        % and then apply the Hamming window over each column.
        z = ones(FFTLEN,1);
        iqw  = chwin(1:LENGTH/2,:).';
        iqw  =iqw-complex(kron(z,mean(real(iqw))),kron(z,mean(imag(iqw))));
        iqw  = iqw.*wind;

        % now do FFT over all columns to get spectrum
        specch = fft(iqw);
        specch = specch.*conj(specch)/FFTLEN;
        finch = fftshift(specch,1);    % flip fft (center 0-frequency)

        % determine the measured noise floor, it is range dependent for
        % TARA because of a hardware filter.
        calnoise(ns,:,np) = sum(finch);

    end;                    % end polarization/beam loop

end;                        % end profile loop (end PRE-PROCESSING)


% ***************************************************************************
% ************************* ABSOLUTE CALIBRATION ************************
% ***************************************************************************
% Absolute calibration is obtained by calculating the ratio between the
% theoretical noise reflectivity and the measured noise (calfactor).
% Calfactor depends on range and polarization/beam. Calfactor should not
% contain the statistical fluctuations of the measured noise (need for
% averaging and smoothing)

% Average noise measurement
meannoise = squeeze(mean(calnoise));
% Smooth noise measurement
beta=0.1;       %smoothing parameter
it=50;          %smoothing paramter
for np=1:NPOL
    meannoise(:,np)=Specsmoother_rt(meannoise(:,np),LENGTH/2,beta,it,0);
end

tranZcalc=Zcalc.';                 % Measured noise reflectivity (Zcalc)
calfactor=zeros(LENGTH/2,NPOL);
for np=1:NPOL
    calfactor(:,np) = tranZcalc./meannoise(:,np);
end;

% correction for transmitter imbalance
```

```
calfactor(:,1) = 10^(0.35/10)*calfactor(:,1);   % VV
calfactor(:,2) = 10^(0.35/10)*calfactor(:,2);   % for HV only (not VH)

% calibration range independent for Ldr_v (ZHV/ZVV)
off2 = mean(calfactor(:,2))/mean(calfactor(:,1));
calfactor(:,2) = calfactor(:,1)*off2;
% calibration range independent for Zdr (ZHH/ZVV)
off3 = mean(calfactor(:,3))/mean(calfactor(:,1));
calfactor(:,3) = calfactor(:,1)*off3;

%################## end of noiseprocessor
%t=toc;
%disp(['temps de calcul :',num2str(t),' s'])

figure
plot(10*log10(meannoise(:,3)))
```

## Polarimetric processor: V4

```
function [ZDR,LDR,ZHH,VHH,WHH,Rhoh,height,cvp] = ...
           Pola_rt_V4(beatsig,FFTLEN,NPOL,Ts,FEQ,EL,At,meannoise,calfactor)

% Polarimetric processor: V4
% C. Unal. March-April 2011 (Polarimetric processor)
% This script processes one datafile. This data file is part of a time
% series of data files measured with the same radar specifications.
% From the radar signal, spectrograms (Doppler spectra per range) and
% profiles of radar observables are obtained. A profile shows the variation
% of one radar observable versus range or height at a specific time. The
% profiles are output of this script.
% Modifications: C. Unal and Y. Dufournet, May 2011, real-time interface
% C. Unal, May 2011, two spectrograms are averaged at the beginning to
% decrease time constraint.


    %INPUT
%beatsig   : 2D array raw data (with transmit on!)
%            [LENGTH, number of sweeps]
%FFTLEN    : Doppler FFT-length
%NPOL      : Number of polarisation/beam states
%Ts        : Sweep time [s]
%FEQ       : frequency excursion [MHz]
%EL        : elevation angle (deg.)
%FEQ       : frequency excursion (MHz)
%At        : attenuation (dB) 0 dB (100 W), 10 dB (10 W), 20 dB (1W), etc..
%meannoise : related mean noise level measured (ouput of noise processor)
%calfactor : calibration factor (ouput of noise processor)
%FFTLEN    : length Doppler FFT (standard at 512)
%LENGTH    : length range FFT (standard at 1024)
%NPOL      : Number of polarisation states
%Ts        : Sweep time [s]
```

```
    %OUTPUT
% ZDR, LDR, ZHH, VHH, WHH and Rhoh are 2D array (Nran-near+1,NPROFS/2)
%ZDR    : differential reflectivity [dB]
%LDR    : linear depolarization ratio[dB]
%ZHH    : reflectivity value of the main beam for HH pol [dBZ]
%VHH    : Mean Doppler velocity of the main beam for HH pol [m.s-1]
%WHH    : Doppler width of the main beam for HH pol [m.s-1]
%Rhoh   : complex cross correlation coefficient
%height : height vector [m above ground level]
%cvp    : code version

% ************************ called functions  *****************************
% Specsmoother_rt : signal smoothing
% Moments_sLdr2_rt: first profile calculation (main beam polarimetric)
% Moments_pola_rt : definitive profile calculation (main beam polarimetric)
% ***********************************************************************

% ************************ publications **********************************
% Spectral polarimetric dealiasing:
% Unal, C. M. H., and D. N. Moisseev, 2004: Combined Doppler and
% polarimetric radar measurements: correction for spectrum aliasing and non
% simultaneous polarimetric measurements. J. Atmos. Oceanic Technol., 21,
% 443-456.
% Spectral polarimetric clutter suppression:
% Unal, C., 2009: Spectral polarimetric radar clutter suppression to
% enhance atmospheric echoes. J. Atmos. Oceanic Technol., 26, 1781-1797.
%***********************************************************************
tic
%***********************************************************************
% Measurement specifications
% FFTLEN = 512;  %number of samples (or sweeps) for Doppler processing
% LENGTH = 1024; %number of samples for range processing (within one sweep)
% NPOL   = 3;                     %number of polarizations
% Ts     = 0.001;                 % sweep time (s)

%number of profiles determination
%NPROFS = 2;      % !! TO be determined
[LENGTH,sizey] = size(beatsig);
NPROFS = fix(sizey/(FFTLEN*NPOL)); %determined from the radar control block
%(how much data is sent by the radar control?)
Z1   = 1;                         %lowest range bin (0 m)
Z2   = LENGTH/2;                  %highest range bin
SNR  = 5;        %signal-to-noise ratio for the Doppler spectrum clipping
Nran = Z2-Z1+1;                   % number of range bins
cvp  = 'pola_V4';
%***********************************************************************

% ********************** PRE-PROCESSING PREPARATION *********************
% ***********************************************************************
% % open the data file
% fid = fopen(beatsig);
% create the window for range FFT
y = hamming(LENGTH);
z = ones(1,FFTLEN);
% LENGTH by FFTLEN - FFTLEN sweeps of LENGTH samples, each sweep in column
winr = kron(z,y);

% create window for Doppler FFT
y = hamming(FFTLEN);
z = ones(1,LENGTH/2);
% FFTLEN by LENGTH/2 - LENGTH/2 range gates with FFTLEN sweeps
% in each to do FFTLEN point FFT for each gate
wind = kron(z,y);

% nhea = 784/2;                        % header consists of 784 bytes
% skip the header (not yet useful information)
% header = fread(fid,nhea,'short');
%***********************************************************************

% ********************** POST-PROCESSING PARAMETERS *********************
% *********************************************************************

if rem(NPROFS,2)==1
   NPROFS = NPROFS-1;             % to ensure even number of profiles
end

% calculation of useful vectors
% range
rangeres = 3e8/(2*FEQ*1e6);       % range resolution [m]
range    = (Z1-1:Z2-1)*rangeres;  % range [m]

% Doppler velocity or frequency
lambda   = 3e8/3.298e9;           % TARA wavelength [m]
nyquist  = lambda/(4.*Ts*NPOL);   % max. unambiguous Doppler velocity [m/s]
velres   = 2*nyquist/FFTLEN;      % Doppler velocity resolution [m/s]
vel      = (velres-nyquist):velres:nyquist;  % Doppler velocities [m/s]
% Extension of the Doppler velocity interval after dealiasing
lvel     = (velres-NPOL*nyquist):velres:NPOL*nyquist;  % [m/s]
nyfeq    = 1/(2.*Ts*NPOL);        % max. unambiguous Doppler frequency [Hz]
feqres   = 2*nyfeq/FFTLEN;        % Doppler frequency resolution [Hz]
feq      = (feqres-nyfeq):feqres:nyfeq;  % Doppler frequencies [Hz]

% parameters for spectral smoother
beta=0.1;                        % moderate smoothing
it=50;                           % max number of iterations

% NOISE FILE: radar noise level + calibration factor
% scaling of the calibration factor
Atlin=10^(At/10);
calfactor=Atlin*calfactor;
% determine the clip level for all polarizations
```

```
DBaboveNoise = SNR;           % threshold acceptable SNR in db for clipping
factnoise = 10^(DBaboveNoise/10);   % linear
% in the Doppler domain
noisefloor = meannoise.*calfactor/FFTLEN*factnoise;

% DEALIASING MAIN BEAM (use phase diff between signal vv and hh)
% Doppler phase compensation for the cross-spectrum because of non
% simultaneity of the measurements vv and hh
Compvvhh = (exp(-i*2*pi*feq*2*Ts))';
Tm = NPOL*Ts;          % sampling time [s]
tt = 2*Ts;             % offset time between VV and HH measurements [s]
Phimax = 2*pi*tt/Tm;   % absolute phase increment due to Doppler aliasing
imax    = fix(NPOL/2);
% mean values of the cross spectrum phase for meteorological targets in
% case of Doppler aliasing after Doppler phase compensation.
Phi_dea = -imax*Phimax:Phimax:imax*Phimax;
% conversion between -pi,+pi  minus sign due to measurement specifications
% of TARA (has to be checked for the first measurements: which sign has the
% measured phase for [-3VDmax,-Vdmax[)
Phi = angle(exp(-i*Phi_dea));
% NPOL=3 sig=60 deg. rho_co > 0.72
sig = mean(diff(sort(Phi)))/2;    % max phase std permitted
% All the targets with ro_co < 0.72 are suppressed with this dealiasing

% Phase diff between VV and HH (due to radar)
Phi_cal = -pi;


 %*********************************************************************
 %*********************    PROCESSING    *******************************
 %*********************************************************************

 % OUTPUTS: Moments (interpolated)
 Zh = zeros(Nran,NPROFS/2,NPOL);      % reflectivity [mm6/m3]
 Vh = zeros(Nran,NPROFS/2,NPOL);      % mean Doppler velocity [m/s]
 Wh = zeros(Nran,NPROFS/2,NPOL);      % Doppler width [m/s]
 Rhoh = zeros(Nran,NPROFS/2);         % cross correlation coefficient vv hh

 %*********************    PROFILE LOOP    *****************************
 % ******************************************************************
 for is = 1:2:NPROFS;
     ns = (is-1)/2 + 1;                 % profile number
     % ******************** Initialization ***************************
     spec_1 = zeros(FFTLEN,Nran,NPOL);  % raw Doppler spectra
     spec_2 = zeros(FFTLEN,Nran,NPOL);
     % Doppler spectra sZ_dea and cross spectrum Fvh_dea dealiased for all
     % the ranges (Nran) and polarizations (NPOL)
     sZ_dea=zeros(NPOL*FFTLEN,Nran,NPOL);
     Fvh_dea=zeros(NPOL*FFTLEN,Nran);
     % Moments (not yet interpolated)
     Z = zeros(Nran,NPOL);
```

```
V = zeros(Nran,NPOL);
W = zeros(Nran,NPOL);
% Doppler spectra (spol)and cross spectrum (Fvh) dealiased,
% with clutter suppression and clipped (they are not yet used)
%spol=zeros(NPOL*FFTLEN,Nran,NPOL);
%Fvh=zeros(NPOL*FFTLEN,Nran);
% ********************* end Initialization ***************************

% ********************* PRE-PROCESSING *******************************
% from radar signal (3) to raw spectrograms (Doppler spec. per range)

% read block of data
%allbeat=fread(fid,[LENGTH,FFTLEN*NPOL],'short');  % 1 spectrogram
allbeat=beatsig(1:LENGTH,((ns-1)*FFTLEN*NPOL*2)+1:ns*FFTLEN*NPOL*2);

 for np = 1:NPOL     % NPOL = number of polarizations and radar beams

    % extract the beat signal for one polarization state or beam
    % polarization or beam changes from sweep to sweep
    beat_1 = allbeat(:,np:NPOL:FFTLEN*NPOL);
    beat_2 = allbeat(:,FFTLEN*NPOL+np:NPOL:FFTLEN*NPOL*2);

    % remove the radar DC component from the beat signal
    z = ones(LENGTH,1);
    bm = mean(beat_1);
    beat_1 = beat_1 - kron(z,bm);
    bm = mean(beat_2);
    beat_2 = beat_2 - kron(z,bm);

    % do range FFT with windowing
    beat1 = beat_1.*winr;
    chwin_1 = fft(beat1);
    beat2 = beat_2.*winr;
    chwin_2 = fft(beat2);

   % discard image half of matrix, and take transpose, since we will
   % now do FFT over other dimension. Make both components zero mean,
   % and then apply the Hamming window over each column.
   z = ones(FFTLEN,1);
   iqw  = chwin_1(1:LENGTH/2,:).';
   iqw  =iqw-complex(kron(z,mean(real(iqw))),kron(z,mean(imag(iqw))));
   iqw  = iqw.*wind;
   % now do FFT over all column to get spectrum
   specch = fft(iqw);
   specch = specch/sqrt(FFTLEN);
   finch = fftshift(specch,1);  % flip fft (center 0-frequency)
   spec_1(:,:,np) = finch;      % raw Doppler-range spectrogram

   iqw  = chwin_2(1:LENGTH/2,:).';
   iqw  =iqw-complex(kron(z,mean(real(iqw))),kron(z,mean(imag(iqw))));
   iqw  = iqw.*wind;
   specch = fft(iqw);
   specch = specch/sqrt(FFTLEN);
   finch = fftshift(specch,1);
   spec_2(:,:,np) = finch;      % second raw Doppler range spectrogram

end                            % end loop (end PRE-PROCESSING)


% ************************* POST-PROCESSING 1 ***********************
% from raw Doppler-range spectrograms to first-step processed Doppler
% -range spectrograms. First-step processed means smoothing,
% calibration, correct Doppler velocities (dealiasing).
% *****************************************************************

%**********************    RANGE LOOP    ****************************

for nr=Nran:-1:1

    % Doppler spectrum for each pol.
    sZ = zeros(FFTLEN,NPOL);
    % CALIBRATION AND SMOOTHING
    % sZ    absolute calibration with calfactor
    for np=1:NPOL
        sZ(:,np) = (  spec_1(:,nr,np).*conj(spec_1(:,nr,np))+...
         spec_2(:,nr,np).*conj(spec_2(:,nr,np))  )*calfactor(nr,np)/2;
        sZ(:,np) = Specsmoother_rt(sZ(:,np),FFTLEN,beta,it,0);
    end

    % MAIN BEAM: CROSS SPECTRUM, CALIBRATION (calfactor)
    % PHASE COMPENSATION FOR THE NON SIMULTANEITY OF THE MEASUREMENTS
    % VV AND HH (Compvvhh) and phase calibration (Phi_cal)
    Fvvhh = (spec_1(:,nr,1).*conj(spec_1(:,nr,3)) + ...
      spec_2(:,nr,1).*conj(spec_2(:,nr,3))).*Compvvhh*exp(-i*Phi_cal);
    Fvvhh = Fvvhh*sqrt(calfactor(nr,3))*sqrt(calfactor(nr,1))/2;
    Phase = angle(Fvvhh);    % should be not smoothed for dealiasing
    Fvvhh_amp = Specsmoother_rt(abs(Fvvhh),FFTLEN,beta,it,0);
    Fvvhh_pha = Specsmoother_rt(Phase,FFTLEN,beta,it,0);
    Fvvhh = Fvvhh_amp.*exp(i*Fvvhh_pha);

    % MAIN BEAM (POLARIMETRIC): DEALIASING
    % The Doppler spectrum is placed at the right Doppler velocities
    for np=1:NPOL;
        abo = find(Phase>Phi(np)-sig & Phase<=Phi(np)+sig);
        itemp = (np-1)*FFTLEN+abo;
        sZ_dea(itemp,nr,1) = sZ(abo,1);
        sZ_dea(itemp,nr,2) = sZ(abo,2);
        sZ_dea(itemp,nr,3) = sZ(abo,3);
        % phase correction when aliasing
        Fvh_dea(itemp,nr)  = Fvvhh(abo)*exp(-i*Phi(np));
    end;
    end         % END RANGE LOOP ****************************************


    % ************************* POST-PROCESSING 2 ***********************
    % from first-step Doppler-range spectrograms to processed Doppler
    % -range spectrograms. Processed means removal of  non-atmospheric
    % echoes (clutter) and noise. From the processed Doppler-range
    % spectrograms, the profiles of several parameters (Z, V, W) are
    % estimated.
    % *****************************************************************

    %********************     MAIN BEAM (POLARIMETRIC)   ****************
    % Noise and clutter suppression in the spectrograms and first
    % calculation of the mean Doppler velocity.
    [Z,V]=Moments_sLdr2_rt_V2(sZ_dea,noisefloor,NPOL,lvel);
    Vint = vel(FFTLEN);     % initial maximum unambiguous Doppler velocity
    % Doppler velocities interval reduction around mean Doppler velocity V
    % noise, clutter suppression and single of double missing data
    % replaced in the spectrograms, definitive calculation of the profiles
    [Z,V,W,Rho]=Moments_pola_rt_V3(sZ_dea,Fvh_dea,noisefloor,NPOL,lvel,...
                                                               V,Vint);

    % Calculation of the final profiles (OUTPUT)
    Zh(:,ns,:)=Z;           % reflectivity [mm6/m3]
    Vh(:,ns,:)=V;           % mean Doppler velocity [m/s]
    Wh(:,ns,:)=W;           % Doppler width [m/s]
    Rhoh(:,ns)=Rho;         % cross correlation coefficient
    % *****************************************************************


end                     % END PROFILE LOOP ******************************
                    % ************************************************

% *********************************************************************
% ********************   END PROCESSING   *****************************
% *********************************************************************


% ****************************** MASK *********************************
% *********************************************************************
% Suspicious data still remaining after the processing (Doppler width
% excessively small or large) are discarded.
% *********************************************************************
WMB = Wh;
% mask using the Doppler width
WMB(WMB(:,:,1)<velres)=NaN;             % main beam
WMB(WMB(:,:,1)>3)=NaN;
for ns=1:NPROFS/2
    wabo1=find(isnan(WMB(:,ns,1)));     % main beam mask
    Zh(wabo1,ns,:)=NaN;
    Vh(wabo1,ns,:)=NaN;
```

```
     Wh(wabo1,ns,:)=NaN;
     Rhoh(wabo1,ns)=NaN;
 end

 %**************************    STORAGE  *********************************
 height = range*sin(EL*pi/180);
 hnear = 200*sin(EL*pi/180);
 habo = find(height<hnear);
 near = size(habo,2);
 height = height(1,near:Nran);
%  Nhc = size(height,2);
 % Data in the radar near-field are discarded
 Zh = 10*log10(Zh(near:Nran,:,:));                     % unit: dBZ
 ZHH = squeeze(Zh(:,:,3));
 ZDR = ZHH-squeeze(Zh(:,:,1));
 LDR = squeeze(Zh(:,:,2))-ZHH;
 VHH = squeeze(Vh(near:Nran,:,3));
 WHH = squeeze(Wh(near:Nran,:,3));
 Rhoh = Rhoh(near:Nran,:);

 t=toc;
 disp(['temps de calcul :',num2str(t),' s'])

figure
plot(ZDR,height)
```