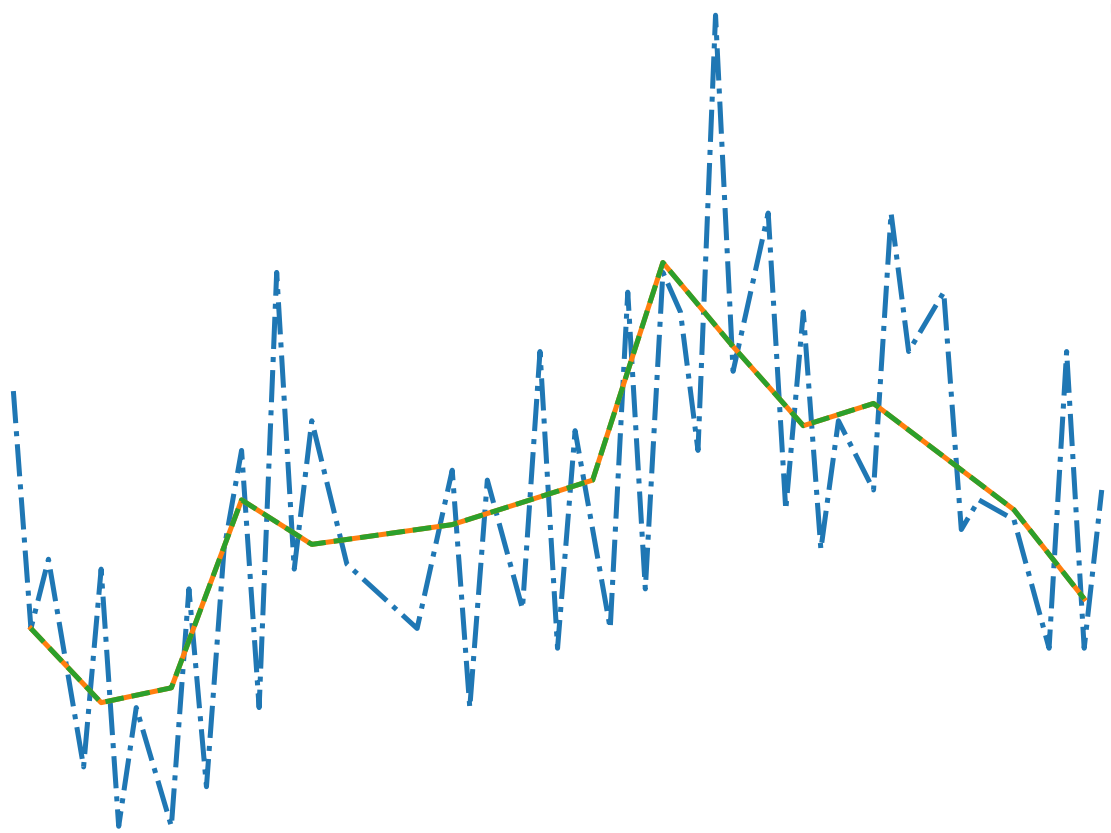

FPGA-Based Data Compression for Large-Scale Nuclear Physics Experiments

Master thesis in Embedded Electronic System Design

Hezhe Xiao

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

Master's thesis 2021

FPGA-Based Data Compression for Large-Scale Nuclear Physics Experiments

Hezhe Xiao

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2021

FPGA-Based Data Compression for Large-Scale Nuclear Physics Experiments

Hezhe Xiao

Supervisor: Håkan T. Johansson, Department of Physics
Examiner: Per Larsson-Edefors, Department of Computer Science and Engineering

Master's Thesis 2021
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The illustration of the effect of the trigger level K on a signal trace, see section 4.8.

Gothenburg, Sweden 2021

FPGA-Based Data Compression for Large-Scale Nuclear Physics Experiments

Hezhe Xiao
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

In nuclear physics experiments, FPGAs control a large number of front-end electronics boards for signal processing. This project provides a generic code for adaptive downsampling of signal traces. The performance of the code is characterized quantitatively in sampling frequency and resource usage for different FPGAs. The process of the data compression is based on the noise estimate and a trigger level to decide the compression ratio so that smaller sample groups are used to store the signal data in regions dominated by critical signal information and larger lengths of sample groups are averaged in areas containing noise. The method and implementation of the compression are described in detail with both VHDL and Python.

Keywords: VHDL, FPGAs, Data compression, Adaptive downsampling, resource utilization.

Acknowledgements

I'm deeply indebted to my supervisors Håkan Johansson and Andreas Heinz for their inexhaustible instructions and support. I would also like to extend my deepest gratitude to my examiner Per Larsson-Edefors for his helpful suggestions.

Hezhe Xiao, Gothenburg, August 2021

Contents

List of Figures

List of Tables

1
Introduction

Adaptive downsampling provides data compression to store signal traces using smaller sample groups in regions containing predominantly signal information, but larger sample groups in the areas dominated by noise. The downsampling process is based on taking groups of samples and to replace them with their average value. The local group length (the number of samples per group) decision is based on the acceptable downsampling ratio which is determined by the estimated standard deviation of the noise [1]. As a result, the adaptive downsampling method reduces the storage of noise and thus increases the compression ratio by keeping fewer values of averaged samples in regions dominated by noise.

In nuclear physics experiments, a large number of different field programmable gate arrays (FPGAs) are used to control modern front-end electronics as well as for signal processing. One limitation is that the amount of data produced by the experiments can be large, which requires a vast amount of transfer bandwidth and storage, and imposes timing constraints for the data acquisition. On the other hand, implementing the algorithm of adaptive downsampling in define VHDL is more complicated and difficult than in higher-level languages (e.g. Python and Matlab) due to the need for clock synchronization and hardware testing.

1.1 Thesis goal

The goal of the project is to design a VHDL code for the compression of signal traces and find the best implementation for FPGAs. The code shall be characterized quantitatively in terms of sampling frequency and resource usage for different FPGA families. It will be described how these scale with the number of bits for each sample and the maximum downsampling level. The code will be tested by synthesis for multiple FPGAs from Xilinx: Spartan 6 and Virtex 7 [2].

In order to have a baseline for comparison, an estimate of the resource usage by considering the compression algorithm as such is needed. From this and the knowledge of how different features are realized on FPGAs, the LUT and FF usage can be estimated. To reach higher frequencies, more pipelining is needed, which also drives resource usage and thus also needs to be taken into account. This estimate should be compared with the actual values for the specific FPGAs to identify how well they match.

The behavior of the compression algorithm critically depends on the trigger level. Therefore, it will also be quantitatively investigated how much the noise estimation

is affected as a function of the frequency of occurrence and size of actual pulses in continuous ADC data streams.

1.2 Challenges

There is an existing VHDL code for adaptive downsampling on FPGAs consisting of a compressor and a standard deviation estimator block [1], however the logic of the design is hard to follow. The challenge of this project is to develop an efficient and easier to follow implementation of adaptive downsampling of signal traces on FPGAs for realistic applications.

Due to the increasing number of data produced by nuclear physics experiments, one strong demand is to reduce the amount of the data to be handled by largely removing useless noise data. This work should result in implementation of a compressor suitable for FPGAs and in improving the algorithm for compression and decompression to be written in higher-level software combined with previously-proposed methods. The result after the development of the implementation on FPGAs should be the same as the simulation result using higher-level software. Thus, it is necessary to implement the algorithm in Python or Matlab. In addition, it is also significant to find and optimize critical paths of the design to maximize the achievable clock frequency. The frequency capability of any design is limited where the signal propagation exceeds the clock period, which happens when the logic is too convoluted or the routing distance is too long.

1.3 Thesis outline

This thesis is organized in the following way: Chapter 2 gives an introduction to the technical background. Chapter 3 introduces the methods, including the algorithm description of mean absolute deviation estimation for noise and adaptive downsampling. Chapter 4 describes VHDL and Python simulation results of the signal traces, the performance of the implementation, and its resource usage on actual FPGAs. Chapter 5 provides a conclusion.

2

Technical Background

One of the detectors that motivated this project is the CALIFA detector. CALIFA (CALorimeter for the In Flight detection of γ rays and light charged pArticles) detector [3], which is a complex detector based on scintillation crystals, the purpose of which is to detect protons and γ-rays from nuclear physics experiments using beams at relativistic energies. It is part of the R^3B experimental setup [3].

Signals of the detector arrive at the Febex Add-on Board (FAB) [4] and are fed into the Febex for sampling and processing. The Febex board has a 50 MHz sampling frequency and 14-bit resolution given by the ADC AD9259 [5]. The ADCs and FPGAs are both located on the Febex board. All Febex boards are inserted in a Febex crate, which provides connections for readout on its backplane. Following digitization, the data can be compressed, which is the motivation for this project. The diagram of the signal processing is shown in Fig. 2.1.

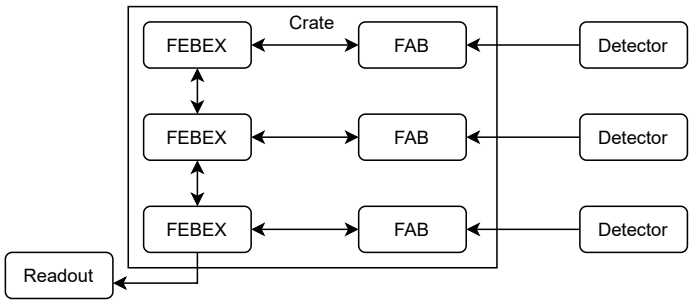

Figure 2.1: Diagram of the signal processing and sampling from detectors using the Febex system. The input for the detector signals are on the Febex Add-on Boards (FAB), while the ADCs and controlling FPGA are on the Febex boards

2.1 ADC

Generally, in choosing a suitable ADC for a detector, there is a tradeoff between sampling frequency and resolution. The sampling rate for an ADC is the number of output samples available per unit time and is specified as samples per second (SPS) [6]. The resolution refers to the number of bits used to encode the voltage level of an analog signal.

A higher-resolution ADC uses a larger number of discrete digital levels to obtain a better representation of the behavior of an analog signal as function of time. For

Digitizer	Sampling frequency (MS/s)	Resolution (bit)
CAEN 740 Digitizer Family	62.5	12
CAEN 724 Digitizer Family	100	14
Struck SIS3302	100	16
Struck SIS3316-125-16	125	16
CAEN 720 Digitizer Family	250	12
CAEN 725S Digitizer family	250	14
CAEN 730S Digitizer Family	500	14
Struck SIS3305	1250	10
CAEN 751 Digitizer Family	2000	10
Struck SIS3305	2500	10
CAEN 743 Digitizer Family	3200	12
CAEN 761 Digitizer Family	4000	10
Struck SIS3305	5000	10
CAEN 742 Digitizer Family	5000	12

Table 2.1: 14 different types of digitizers with various sampling frequencies and resolutions. The trend with faster sampling is associated with lower bit resolution.

an analog signal with a higher frequency, an ADC with a higher sampling rate is required due to the Nyquist theorem [7]. Table 2.1 provides an overview of commercially available digitizer modules. Listed are 14 types of digitizers for radiation detectors with various sampling frequencies and resolutions. Fig. 2.2 illustrates the relationship between the sampling frequency and the resolution. The red dot at 50 MHz is the ADC mounted on the Febex board. Compared to ADCs with a higher sampling rate and less than 12-bit resolution, ADCs with a lower sampling rate have more resolution bits [8][9].

However, it is more expensive to select a high-resolution and high-frequency ADC for a system. Using two or more ADC chips in one module can overcome performance trade-offs to have both high sampling rate and higher resolution.

2.2 Data compression

Data compression can save resources as it limits the storage needs and puts less demand on the transfer rate instead of buying larger servers [10].

A lossless compression algorithm named Difference Predicted Trace Compression (DPTC) [11] provides a method to record the signal trace efficiently, which can be used in front-end electronics directly. It can handle data streams controlled by FPGAs and operates by calculating the difference between trace samples. Such differences are then gathered around zero which is then the most frequently occurring value. But even larger savings can be provided by storing fewer samples although with better resolution via summing of adjacent samples before storage. Whereas DPTC on its own can yield around 4-5 bits per sample for real traces, using an adaptive method for downsampling before it, the trace results in an outcome where about 1-2 bits are needed per sample (see table 4.1 in [1]).

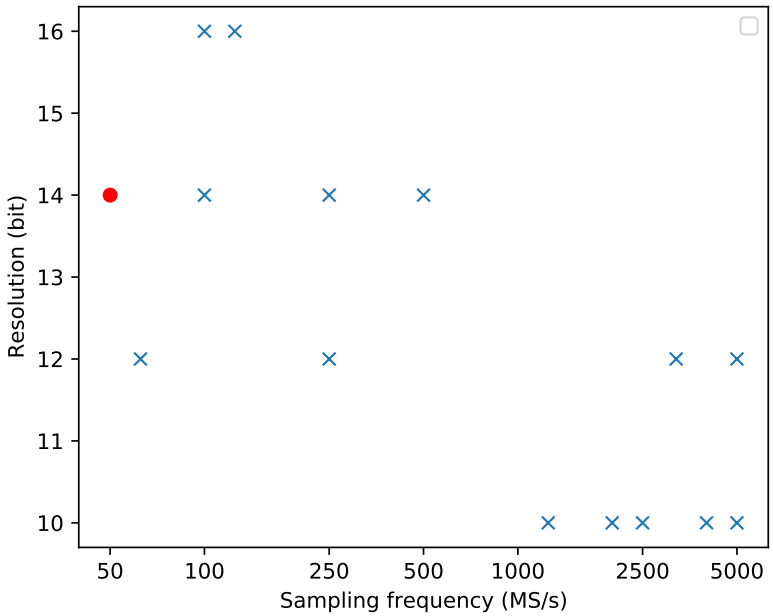

Figure 2.2: Sampling frequency and resolution for different ADCs, the types of digitizer is shown in Table 2.1. The red dot at 50 MHz is the ADC mounted on the Febex board.

Discrete wavelet transform (DWT) [12] is widely used as effective solutions in different fields of signal processing. This method is already implemented for trace compression but needs a large number of computations and large storage resources. Thus, an real-time architecture [12] for forward/inverse wavelet transforms is proposed to shorten the critical path and meet the timing requirement without adding adders or multipliers by using folded and pipelined schemes to implement DWT.

3

Methods

This chapter describes the algorithm and the VHDL design. An easier-to-follow implementation of the noise estimator with a new approach to background subtraction and a modificated IIR filter is provided based on the mean absolute deviation algorithm proposed by Fredriksson and Rahmn [1]. An efficient implementation of adaptive downsampling, the algorithm of which has also been modified, is provided to downsample the signal by the compression ratio and replace the samples with the group mean value. For verification purposes, a Python code which simulates the implemented algorithm has also been written.

3.1 Algorithm description

3.1.1 Mean absolute deviation estimate of noise amplitude

The compression algorithm requires the calculation of a statistic that estimates the inherent noise of the signal trace to be downsampled. Generally, the standard deviation, a measure of the spread of a distribution, is used to determine the amount of noise in a signal. The standard deviation is the square root of the variance, viz.,

$$S = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2},$$

where x_i is the i:th sample of a dataset with mean $\overline{x}$, out of a total of N samples. However, the use of powers and square roots is very costly in an FPGA. For this reason, the **Mean Absolute Deviation** (MAD) is instead used. The MAD is an alternative measure of spread, and is defined as

$$S_{\mathrm{MAD}} = \frac{\sum |x_i - \overline{x}|}{N},$$

where, again, x_i is the i:th sample of a dataset with mean $\overline{x}$, out of a total of N samples.

3.1.2 Background subtraction calculation

To eliminate the effect of a non-zero mean (trace baseline) and a locally linear slope of an increasing or decreasing signal (that is, a change in $\overline{x}$ across any sample of

the data used in determining the MAD), the sums of the values of two groups of previous and following samples are subtracted from the central data. The central data value is multiplied by the total number of group members. This is done rather than calculating the mean of each group in order to preserve as many significant digits as possible. The calculation is shown in Fig. 3.1.

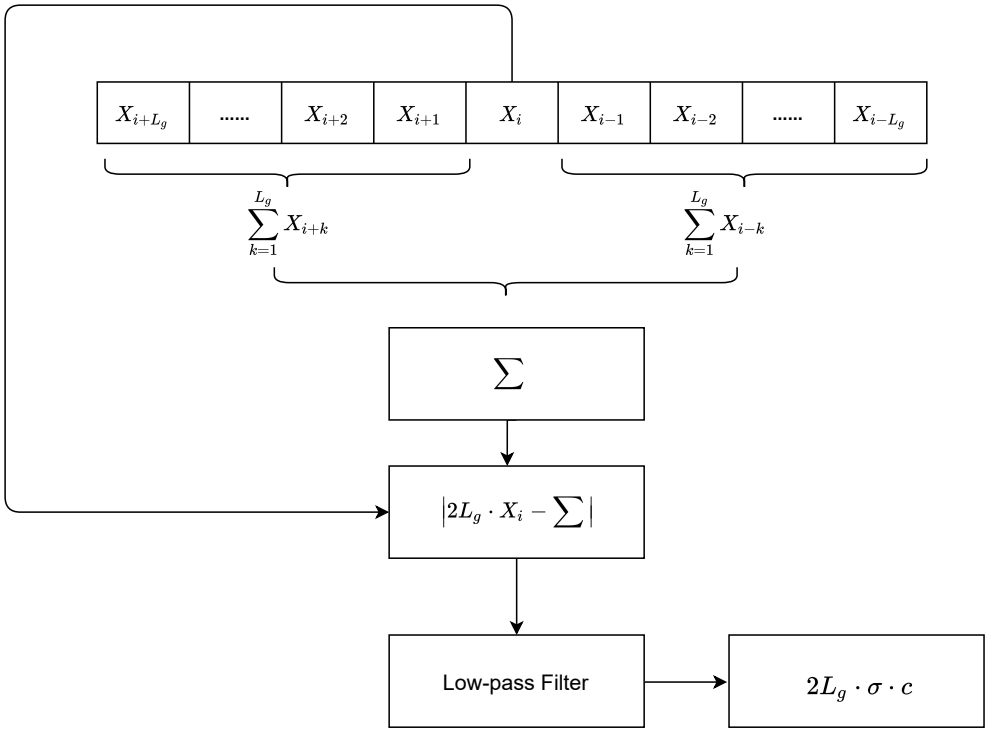

Figure 3.1: Block diagram of the background subtraction calculation to remove the slope of the signal and feed it into the low-pass filter to get a smoothed absolute deviation. g is the group size, L_g is the number of data for each neighboring group ($L_g = 2^g$), σ is the standard deviation of noise and c is the extended correction argument.

3.1.3 Low-pass filter—IIR filter

An infinite impulse response (IIR) filter [13] is used to estimate the mean absolute deviation [14]. The filter reduces the influence of individual large random fluctuations in the signal, causing the output to converge toward the mean absolute deviation over time. The filter performance is determined by a coefficient, referred to as the time constant. The equation for each cycle of the filter is

$$y(i+1) = \frac{L_n - 1}{L_n} y(i) + \frac{1}{L_n} x(i+1). \tag{3.1}$$

where $y(i)$ is the i:th output, $x(i)$ is the i:th input, and L_n is the filter coefficient given by 2^n where n is an integer. To further limit the influence of large outliers in

the input data, the filter is also guarded from any input that is more than T times as large as the current estimate. Only if such a large input occurs three times in a row does the filter update the estimate according to the formula

$$y(i+1) = \frac{L_n - 1 + T}{L_n} y(i). \tag{3.2}$$

This ensures that the updated value is approximately equal to what it would have been if an input had been slightly smaller than $y(i) \cdot T$, placing a bound on how much the estimate can increase in one cycle.

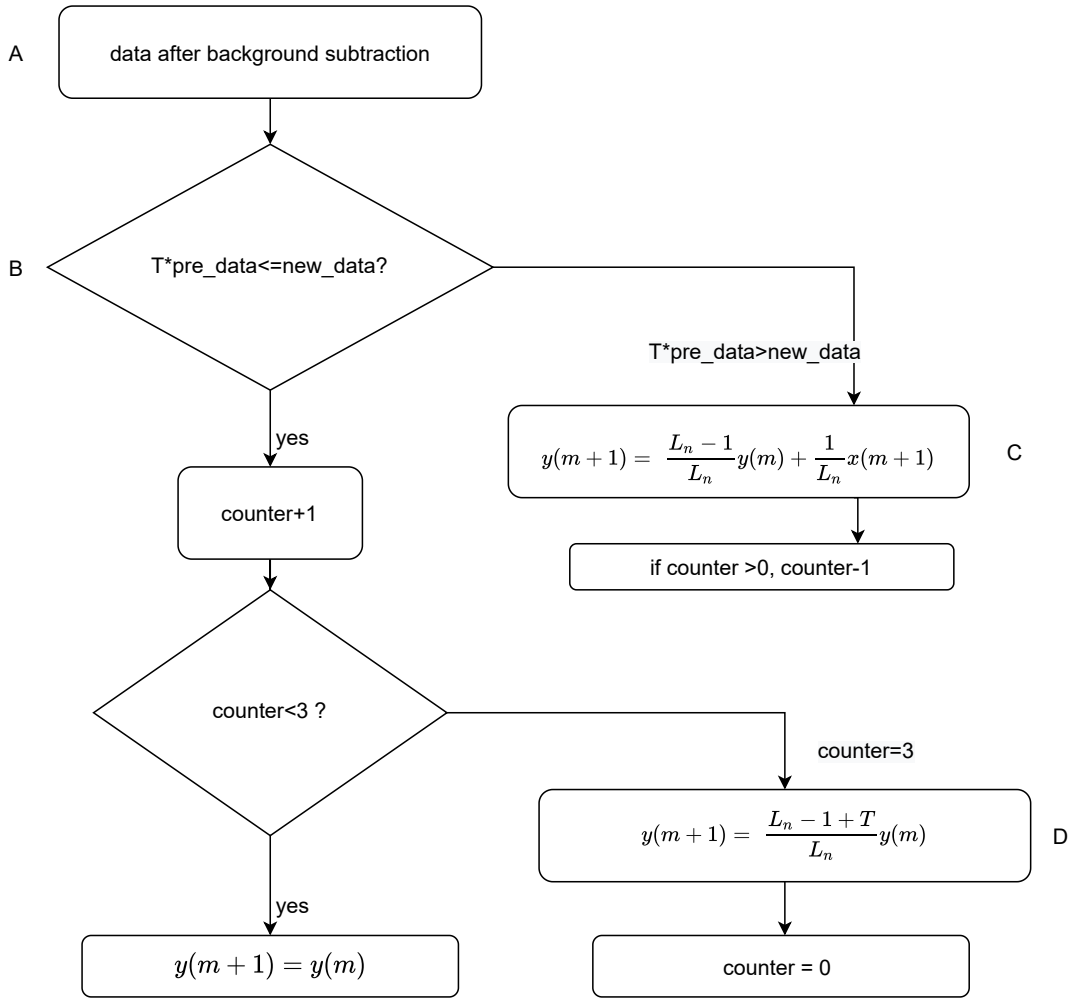

Figure 3.2: The workflow of the IIR filter, including the regular filter (filter equation C), guard against excessively large input (condition in block B), the counter that tracks the occurrences of such large inputs, and the special rule for increasing the estimate by a constant fraction when large inputs occur repeatedly (filter equation D). For more details, see text.

All multiplications and divisions are implemented with either left- and right-shifts or zero-padding in binary arithmetic. In particular, to prevent loss of precision inside

the filter, the inputs and current estimate are padded with n zeroes, which are only truncated when used in another component. The workflow of the implemented IIR filter is shown in Fig. 3.2, illustrating the process:

1. The data after background subtraction is fed into the filter.
2. Comparing each input to the previous output, if the input is not in excess of T times the input, it will be calculated according to the filter equation (3.1). If the variable `counter` is in excess of zero, it is decremented by one.
3. Each time an input in excess of T times the previous output occurs, the `counter` variable is incremented by 1.
4. If `counter` is smaller than 3, the estimate is not changed, that is to say, $y(m+1) = y(m)$.
5. If `counter` reaches 3, the estimate is updated according to equation (3.2). `counter` is then reset to zero.

If the first input is enumerated as $x(1)$, then the filter must be initialized by choosing a value for $y(0)$. Because the filter will be stuck at $y = 0$ and can not start to estimate the noise owing to the guard against excessively large inputs, an initial value of $y(0) = 1$ is chosen instead. Even with n as large as 12, with an input noise level of $\sigma = 0.5$, a 100 MHz FPGA needs only around 0.01 ms to accurately estimate the noise.

3.1.4 Adaptive Downsampling

The Adaptive Downsampling uses the smaller data group lengths to store the significant signal trace parts and larger group lengths to increase the compression effect and reduce the impact of noise. The choice of each local group is based on a mean absolute deviation estimate of the noise and the trigger level, which is a tunable parameter that together with the noise estimate determines a threshold, which in turn determines the local group length for the signal trace. If the trigger level is small, even small changes of the signal will result in a small group length. In particular, if the trigger level is set to 0, no downsampling will occur. Similarly, when the noise estimate is small, the adaptive downsampling will be more reactive in decreasing the group length in response to small changes in the signal trace.

3.1.4.1 The compression ratio

The compression ratio is $R_j = 2^j$, where j is a non-negative integer. $R_{\max}$ is the maximum downsampling ratio, which is the largest group length the data can be grouped in. It is a tunable parameter decided by the user.

The selection of ratio requires the calculation for the following data a sequence of difference values between neighboring groups with each given group length. If the differences of the group length conform to equation (3.3)[1], the ratio can be maintained or increased, otherwise it is decreased to a smaller group length.

$$f_{R_j}\left(x_i, x_{i+1}, ..., x_{i-1+R_j}\right) = |\bar{x}_i - \bar{g}_i| \leqslant \frac{K \cdot \sigma}{\sqrt{R_j}}, \tag{3.3}$$

where x_i is the average value of the group length of data, $x_i = \frac{1}{R_j}\sum_{k=0}^{R_j-1} x_{i+k}$. g_i is the average value of the half group length of data, $g_i = \frac{2}{R_j}\sum_{k=0}^{\frac{R_j}{2}-1} x_{i+k}$. K is the trigger level and σ is the noise amplitude estimate. The reason for using one group average and one half-group average, rather than two half-group averages, is because the group average for each $j < j_{\max}$ can be efficiently re-used as the half-group average for $j + 1$.

3.2 The VHDL design

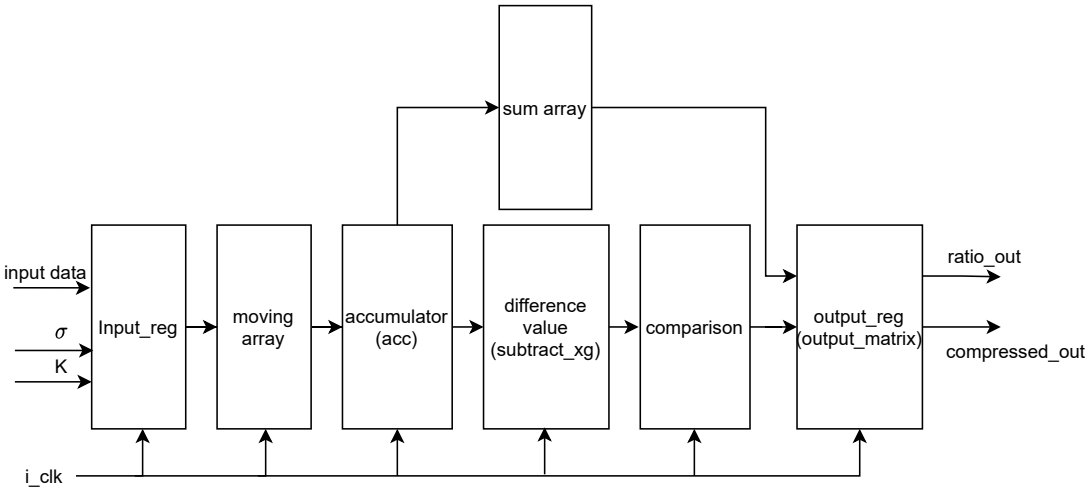

Figure 3.3: The block diagram of the VHDL design, it includes different modules of moving array, accumulator, difference value, sum array and comparison.

A block diagram for the VHDL design of adaptive downsampling is shown in Fig. 3.3. The data from the signal trace is stored in the moving array, the length of the array is R_{max}. The sum of R_j data is stored in the accumulator array and acc[j] means the sum of R_j ($R_j = 2^j$) data. Thus, every rising edge, acc[0] stores the latest signal data, acc[1] stores the sum of two latest signal data, acc[2] stores the latest four data until acc[j] stores the latest R_j data. After that, these sum data goes into the 2-D sum array output_matrix and output_matrix[R_j][t] stores the sequence number of sum data for the group length is R_j, the number of sum data stored can be changed by the user, t means the delay clocks to synchronize the sum data to calculate the difference values for the neighboring data groups.

According to the equation (3.9), the difference value of R_j data as a group is calculated and stored in the subtract_xg array. The length of the array is decided by R_{max}. Subtract_xg[0] is the difference of two latest consecutive data, Subtract_xg[1] is the difference of the continuous two local data groups of two data, subtract_xg[2] is the difference of the neighboring two local data groups of four data until subtract_xg[$j_{max} - 1$] stores the difference of the data groups of $R_{max}/2$ data.

Then, these difference values are compared with the parameter of $K \cdot \sigma \cdot \sqrt{R_j}$. If

the difference value for each R_j is smaller than the parameter, it means the two neighboring R_j group data can be grouped together, otherwise these two groups data can not go together to output because their values have too big a difference. These particular signal data need to be stored with a smaller compression ratio to track.

As Fig. 3.4 shows, a moving array is used to store the number of $R_{\max}$ data (here $j = 2$, $R_{\max} = 4$), the half-length array stores two data ($j = 0$) and a quarter of the array stores one data. Every rising clock edge, the last data from the signal is stored in the front of the array and other data shift right one position, thus, the data the most earliest added in the rear is removed from the moving array.

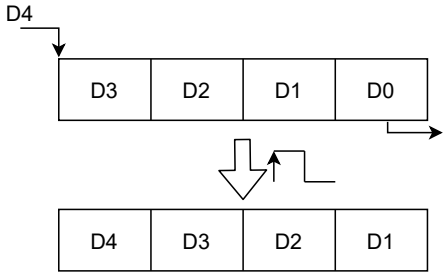

Figure 3.4: The diagram shows how the moving array is used on every rising clock edge to store the data and provide the delayed data needed to get the difference value of neighboring data groups. In this example, $j = 2$, $R_{\max} = 4$.

Therefore, every clock cycle, the sum of latest four data, two data and one data is stored in the accumulate array respectively. In the accumulate array, in the same clock cycle,

$$acc[2] = D1 + D2 + D3 + D4, \tag{3.4}$$

$$acc[1] = D3 + D4, \tag{3.5}$$

$$acc[0] = D4. \tag{3.6}$$

In this case, the difference value of the two neighboring data is

$$|D3 - D4| = |acc[1] - 2acc[0]| = |(D3 + D4) - 2D4| = |D3 - D4|, \tag{3.7}$$

and the difference value of two neighboring groups of combining two data-words as a group is

$$|(D3+D4)-(D1+D2)| = |acc[2]-2acc[1]| = |(D1+D2+D3+D4)-2(D3+D4)|. \tag{3.8}$$

From the equations (3.7) and (3.8), using one array ([D4,D3,D2,D1]) and one half of this array data ([D4,D3]) has the same effect to get the difference value by using

two half-array data ([D2,D1] and [D4, D3]). However, comparing with two half-group data, using one array data and half array data is more efficiency because it synchronizes the signals at the same clock to calculate the difference value of the neighboring data. On the contrary, two half-array data needs another two more registers to store the half-array data [D2,D1] and D3 since they all happen on the last rising clock edge.

To keep the results accurate, the division operation is replaced by the sum operation in VHDL since the implementation of right-shift operation in VHDL for the division would lose precision by giving up the last bits. Thus, an alternative equation (3.9) is used in the adaptive downsampling block,

$$f_{R_j}\left(x_i, x_{i+1}, ..., x_{i-1+R_j}\right) = \left| \sum_{k=0}^{R_j-1} x_{i+k} - 2 \sum_{k=0}^{\frac{R_j}{2}-1} x_{i+k} \right| \leqslant K \cdot \sigma \cdot \sqrt{R_j}. \tag{3.9}$$

The general rule to select the compression ratio is to make sure that equation (3.9) is fulfilled for all k smaller than the selected j. The equations to be satisfied for a given R_j are (3.10) for j>1 and (3.11) for j=1.

$$D_j\left(x_0, ..., x_{R_j-1}\right) = \prod_{k=1}^{j} \left[\prod_{i=-R_j+1}^{R_j-1+\sum_{n=k}^{j-1} 2^n} f\left(x_i, ..., x_{i+2^k-1}\right) \right] \tag{3.10}$$

$$D_j\left(x_0, ..., x_{R_j-1}\right) = \prod_{i=-R_j+1}^{R_j-1} f\left(x_i, ..., x_{i+2^j-1}\right) \tag{3.11}$$

Here D_j is either 1 or 0 - if D_j is 1 then R_j is an acceptable downsampling ratio; if it is 0 R_j is not an acceptable downsampling ratio. Thus based on the value of each D_j the ratio is either maintained, increased, or decreased.

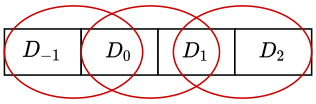

Figure 3.5: The diagram shows how to decide the compression value when R_{max} is 2. If the absolute difference between each value in the red-circled groups is smaller than $K \cdot \sigma \cdot \sqrt{2}$, the middle two data $\{D_0,\ D_1\}$ can be grouped together.

3.2.1 The case analysis for small compression ratios

For the simplest case, if R_{max} is 2, it means the signal trace can either be reduced by combining two data-words into a group, or maintained by storing two individual data-words. In this case, if the data difference values of any of the three adjacent

data-word groups can not satisfy the equation (3.3), the ratio decreases to 0. Fig. 3.5 shows how the algorithm of adaptive downsampling works when R_{max} is 2. The choice of $R = 2$ for the group $\{D_0,\ D_1\}$ requires that its and the two neighboring data group difference values all be smaller than the threshold $K \cdot \sigma \cdot \sqrt{2}$. In other words, the differences of $\{D_{-1},\ D_0\}$, $\{D_0,\ D_1\}$, and $\{D_1,\ D_2\}$ must all satisfy the condition for D_0 and D_1 to be grouped together. If any difference value of these three groups exceed the threshold, D_0 and D_1 need to be stored separately. This is why this compression algorithm is called adaptive downsampling - it uses neighbouring data in choosing the local compression ratio. Using this approach, the data can be grouped in smaller groups in regions containing mostly signal information, and in larger groups in areas dominated by noise.

The sequence diagram in Fig. 3.6 shows the VHDL simulation output of the calculation process. The difference of Data(50) and Data(51) is larger than the parameter of $K \cdot \sigma \cdot \sqrt{2}$ (in this case,$K \cdot \sigma \cdot \sqrt{2} = 16$). Due to this large difference, the previous data Data(49) and Data(50), Data(50) and Data(51), Data(51) and Data(52) can not be grouped together. Thus, for the sum result, Data(48) is grouped with Data(49), and Data(52) is grouped with Data(53), but Data(50) and Data(51) are stored separately.

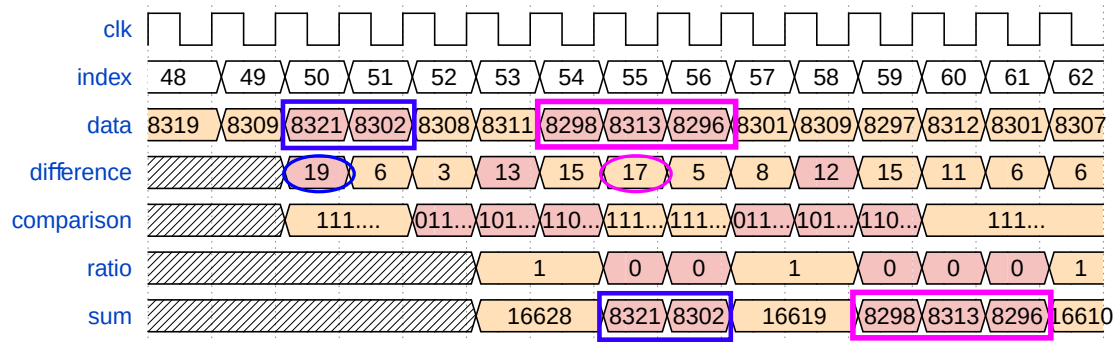

Figure 3.6: The sequence diagram when R_{max} is 2. It shows the simulation result and the calculation process of comparing the difference of two data with the parameter of $K \cdot \sigma \cdot \sqrt{2}$ (here $K \cdot \sigma \cdot \sqrt{2} = 16$ in VHDL code). Due to the large differences 19 and 17 marked in circle, Data(50) and Data(51) are stored separately, and Data(54), Data(55) and Data(56) are also stored separately.

When R_{max} is 4, according to equation (3.10), for 4 to be a valid downsampling ratio, both groupings of 2 and groupings of 4 need to meet the condition of equation (3.9). This is shown in Fig. 3.7, to test the downsampling ratio 4, we need to check if the difference values of four data are smaller than the parameter of $K \cdot \sigma \cdot \sqrt{4}$, from the group of $\{D_{-3},\ D_{-2},\ D_{-1},\ D_0\}$ to $\{D_3,\ D_4,\ D_5,\ D_6\}$. If they are all smaller than the threshold for $R_j = 4$, we need to check if the difference values of two data are smaller than the parameter of $K \cdot \sigma \cdot \sqrt{2}$, from the group of $\{D_{-3},\ D_{-2}\}$ to $\{D_5,\ D_6\}$. If they are all smaller of the parameter for ratio=4, the data D_0, D_1, D_2, D_3 can be grouped together. Otherwise, we need to check it the ratio can be 2. To guarantee the ratio=1, we need to check the different values of two data are smaller

than the parameter of $K \cdot \sigma \cdot \sqrt{2}$ from the data group of $\{D_{-1}, D_0\}$ to $\{D_1, D_2\}$. If any of them are larger than the parameter for $R_j = 2$, the ratio needs to be 1.

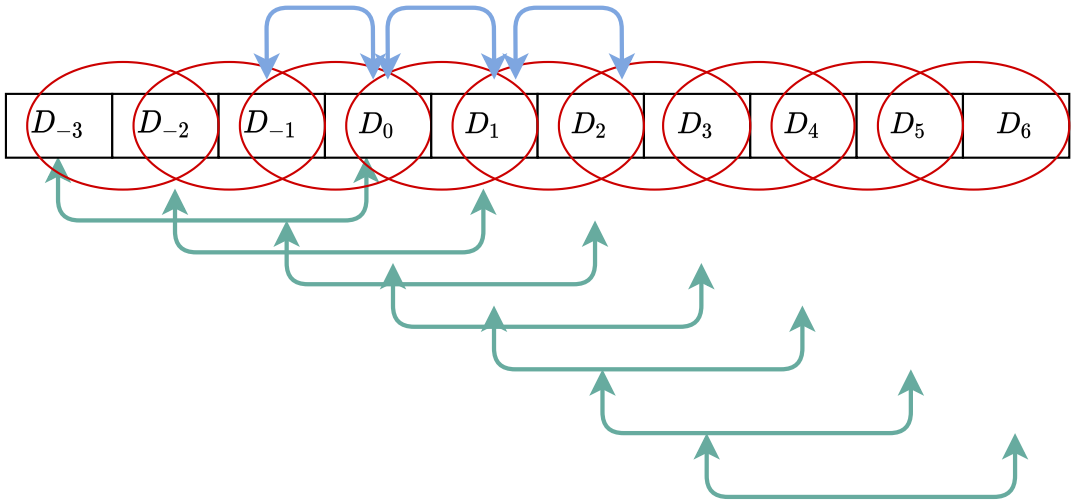

Figure 3.7: The diagram shows how to decide the compression value when R_{max} is 4. The green groups data and red circle groups data are used to check if the ratio can be 4. They are calculated by using the difference value and compared with the parameter of $K \cdot \sigma \cdot \sqrt{4}$ and $K \cdot \sigma \cdot \sqrt{2}$ respectively. The blue groups data are used to check if the ratio can be 2.

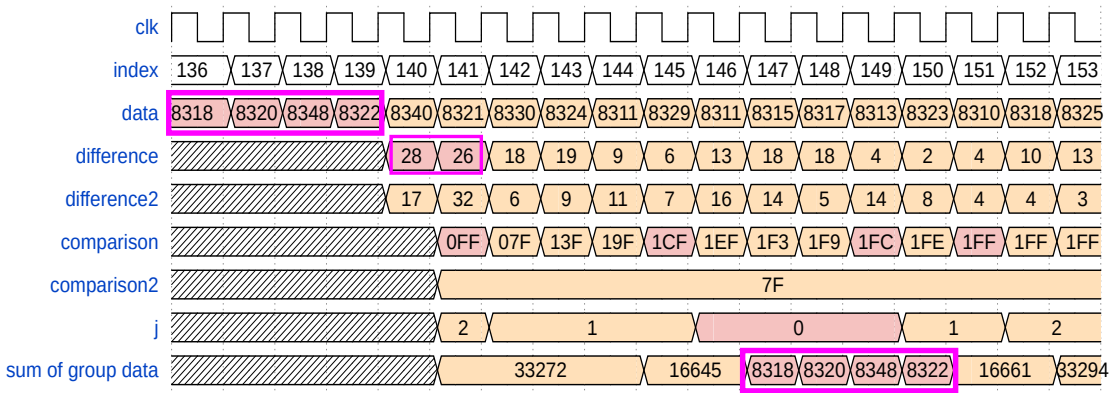

Figure 3.8: The sequence diagram when $R_{\max}$ is 4. It shows the simulation result and the calculation process of comparing the difference of continuous two data with the parameter of $K \cdot \sigma \cdot \sqrt{2}$, and comparing the difference value of two neighboring groups of two data with the parameter of $K \cdot \sigma \cdot \sqrt{4}$ to check if the local four data can be grouped. Due to the marked large differences 28 and 26, Data(136), Data(137), Data(138) and Data(139) are stored separately.

The sequence diagram in Fig. 3.8 shows the VHDL simulation output of the calculation process to check if the data can be grouped as the four data-words sample group. The difference values of Data(137) and Data(138), Data(138) and Data(139) are larger than the parameter of $K \cdot \sigma \cdot \sqrt{2}$. Due to the large differences, these four

data can not be grouped together, they need to be stored separately. Not only the items with big difference need to be stored separately, the surrounding data also needs to be stored in a smaller group size. As the Fig. 3.7 shows, once the eight neighboring groups of two data-words around the local data D0 and D1, D0 and D1 can not be grouped together with other two data-words group. Thus, Data(135) and Data(136) should be grouped together as a two data-words group instead four data-words group. Similarly, because the influence of the difference values of Data(137) and Data(138), Data(138) and Data(139), Data(140) and Data(141) are grouped together as a two data-words group.

4

Results and Discussion

This chapter describes the result of the noise estimator from the VHDL design and discusses the influence of the local sample group length on the performance of the noise estimator, and compares its output with the simulated result in Python. It also presents the result of the adaptive downsampling in VHDL and Python for different maximum compression ratios with the experimental input data. Since the maximum compression ratio implemented as a VHDL design is 16, section 4.7 uses the Python code, which can use any power of 2 as the ratio, to investigate the effect of using larger ratios in the compression of experimental data.

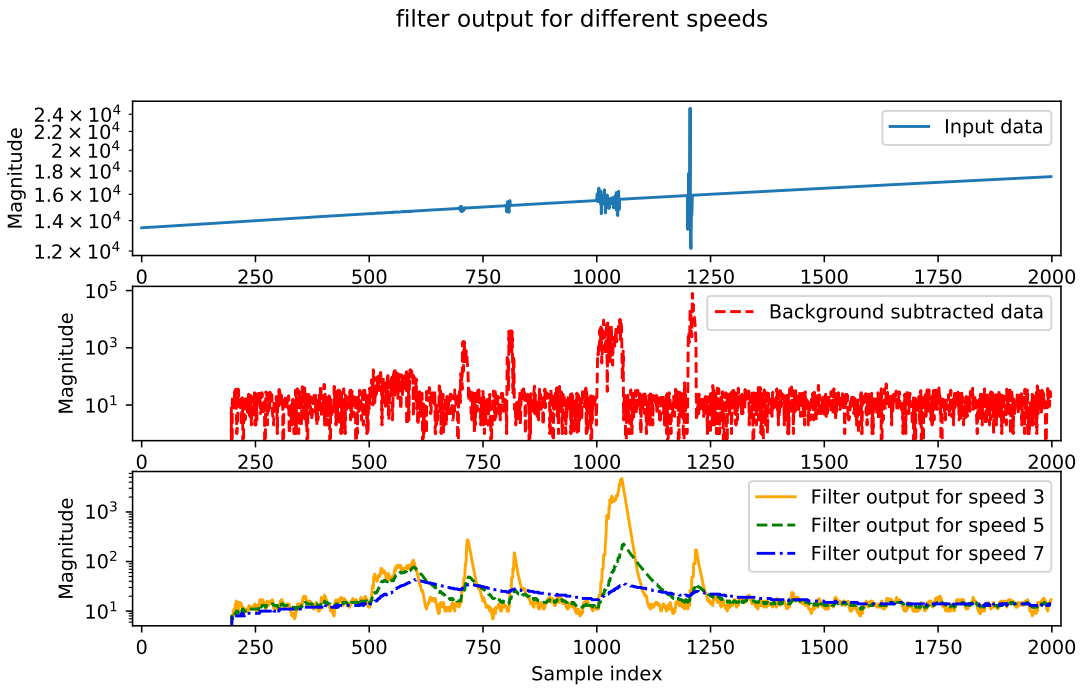

Figure 4.1: The estimate result of filter time constant for the data with different noise. The middle panel shows that the subtraction removes the effect of a baseline or linear slope. The lower panel shows that with larger time constant (labelled speed, refers to n in equation 3.1), the filter responds to deviations in the input more slowly.

4.1 Simulation results for different signals

To check the adaptability of the noise estimate to different situations, test data is generated using Python script with various standard deviations σ of the noise. For example, in Fig. 4.1, the input data consists of 2000 signal samples with a linear baseline and baseline noise $\sigma = 2$. Between samples 500 and 600, a noise of $\sigma = 10$ is added; between samples 700 and 710, a noise of $\sigma = 100$, between samples 800 and 810, a noise of $\sigma = 200$, between samples 1000 and 1050, a noise of $\sigma = 500$. Finally, there are ten huge noise values between samples 1200 and 1210 with an amplitude of $\sigma = 3000$.

For the background subtracted data, the middle panel shows how the deviation changes over 2000 samples of data on a $\log_{10}$ scale. The subtraction is used to remove the slope effect of the signal data, the central data value (see section 3.1.2) is eight times larger than the original input test data since the neighboring group size of each data is four in this case.

The filter output in the lower panel is calculated using (3.1). As expected, the filter reduces the influence of individual random fluctuations. When the noise level changes, the time constant determines how quickly they influence the estimate. With a large time constant, it is clear that the estimate does respond more slowly to the large noise regions, and also returns to the smaller baseline estimate more slowly after.

4.2 Edge effects

4.2.1 Time delay

The VHDL code is driven by the clock when calculating the background subtraction and filtered absolute deviation. Each data point requires a group of $1+2L_g$ data for a single MAD estimate based on the central data sample in this group. Effectively, this means a delay of L_g cycles. Additionally, the calculations of the filter require an extra cycle, adding up to a total delay of $1 + L_g$ cycles.

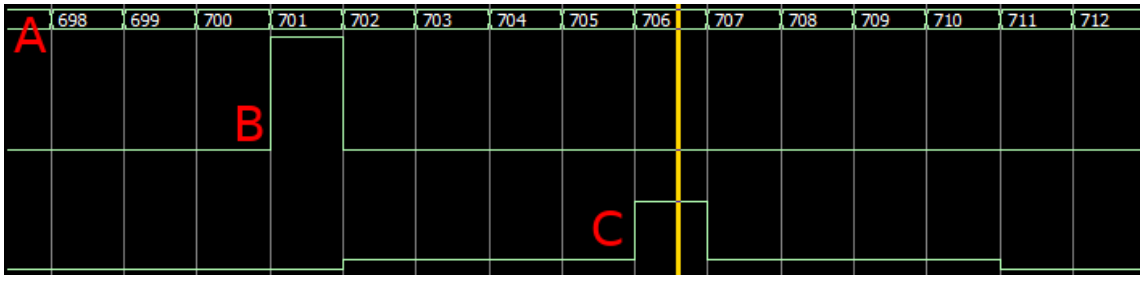

Figure 4.2: Output from VHDL simulation of the noise estimator to show the potential effect of a single large outlier. The data index is at the top (A), the input below it with a spike at B, and the result from the background subtraction at the bottom, with a spike and an increase in the estimated noise around it at C.

4.2.2 Large noise deviation

The use of relatively small groups of values from which to subtract the background and estimate the noise level can cause problems when one of the values in the group is a very large outlier. Moreover, if members are reused between groups, such an outlier will influence several estimates that are then fed into the filter. In particular, a single large outlier will lead to an incorrect slope calculation, which will make the noise appear much larger than it is. The effect of this can be seen in Fig. 4.2, where the noise spike at index 701 (top signal) causes not only a single large noise estimate at index 706, but also substantially larger outputs at $702 - 705$ and $707 - 710$. A remedy for this is to modify the filter to reduce the reuse of group members, so that each group is used only once or twice. In practical terms, this means waiting a number of cycles before a new background subtracted data point is fed into the filter.

4.3 Extending the correction argument for the MAD estimation

Each of the data values in a neighbouring group can be described as a random variable from a normal distribution with mean μ and standard deviation σ. When the group size specifier is g and the number of data for each neighboring group is L_g ($L_g = 2^g$), their variances σ^2 sum linearly to $(2L_g)\sigma^2$, and consequently the standard deviation of the sum will be $\sigma\sqrt{2L_g}$ [15, pg. 57-58]. Meanwhile, the middle data point is simply multiplied by the total number of neighboring group values $2L_g$, giving it a standard deviation of $2L_g\sigma$ and thus variance of $(2L_g)^2\sigma^2$. Hence the difference between the middle data point multiplied by $2L_g$ and the sum of the $2L_g$ data in the group has variance $(2L_g + (2L_g)^2)\sigma^2$ or standard deviation $\sqrt{(2L_g + (2L_g)^2)}\sigma$, with a mean of 0. Once it is normalized by being divided by $2L_g$, it will have standard deviation $\sigma\sqrt{1 + \frac{1}{2L_g}}$.

The standard deviation from the test input data relates to the MAD estimate by $\sqrt{\frac{2}{\pi}}\sigma$ [1]. Thus, when taking the group length (L_g) into account in the correction argument for the MAD estimation, σ from the filter output should be multiplied according to

$$\sigma \cdot c = \sigma \cdot \sqrt{\frac{2}{\pi}} \cdot \sqrt{1 + \frac{1}{2L_g}}, \tag{4.1}$$

where c is simply the chosen designation for the correction factor on the right-hand side.

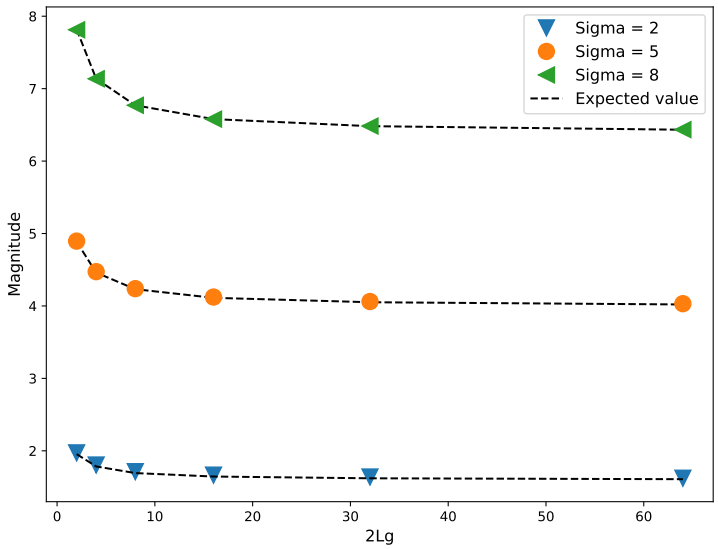

Figure 4.3: Filters with different group length L_g (1, 2, 4, 8, 16, 32) are used to estimate the noise of traces with different amount of noise (σ=2, 5, 8). The results agree with the expected values (dashed lines). In particular, the estimates decrease as the group lengths increase.

In Fig. 4.3, the mean output values of filters with different group lengths, all using time constant 5 are shown. They agree with the expected values of $\sigma \cdot c$ of the noise from a normal distribution, and show the expected decrease in the estimate with an increase in group length. This verifies Eq (4.1) above. Fig. 4.4 shows the associated standard deviations for the points $2L_g = 2, 4, 8, 16$ in absolute and normalized units. Viewing the standard deviations in normalized format suggests that group length 4 ($2L_g = 8$) is overall the best choice, at least for time constant 5 – only for $\sigma = 5$ does group length 8 show slightly smaller standard deviation. Further investigation of the dependency of the standard deviation on the group length is necessary.

4.4 Modified IIR filter

To avoid the multiple impact of large signal deviations caused by the moving average, the filter is modified by adding a flag signal to count each input value until the number of data of moving group size has passed. Thus, for the modified IIR filter, every amount of data (the number of amounts data is decided by the filter group length), only one value feeds into the filter. The modified filter uses every $L_g + 1$ data points and its two neighbouring groups, each of size L_g, to estimate the noise. The filter could be further improved by introducing an additional delay of random length, making it resilient against spurious influences with particular periodicity in the noise. The maximum length of this delay must be chosen based on how much of the input data one is willing to discard to obtain this resilience.

Fig. 4.5 shows the noise estimate results of the current modified version of the filter

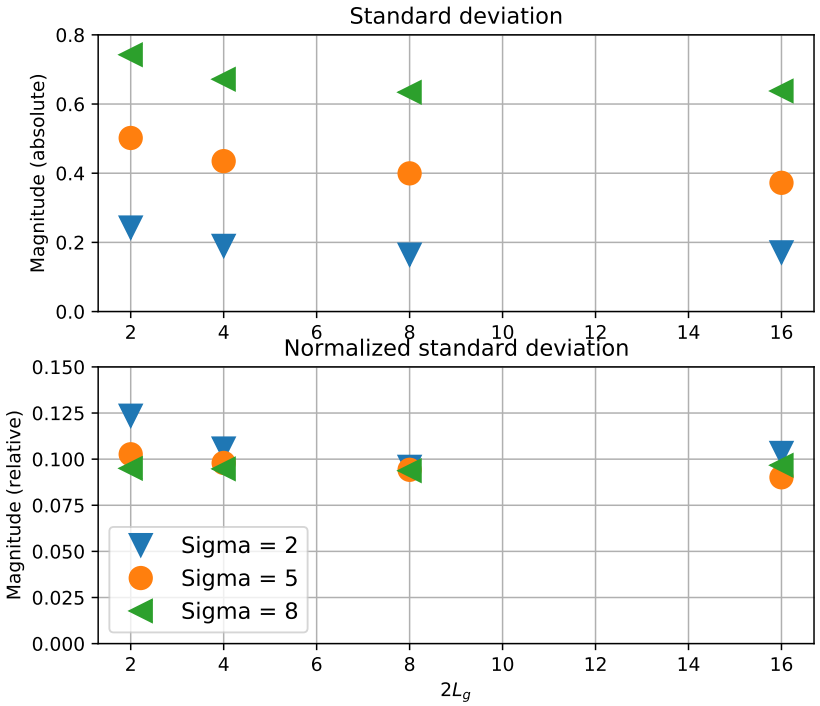

Figure 4.4: The standard deviation estimates associated with the means for $2L_g = 2, 4, 8, 16$ in Fig 4.3, in absolute units (top) and normalized units (bottom), respectively. It shows the variation of the noise estimate. The normalized standard deviation is obtained by dividing the absolute standard deviation with the mean values.

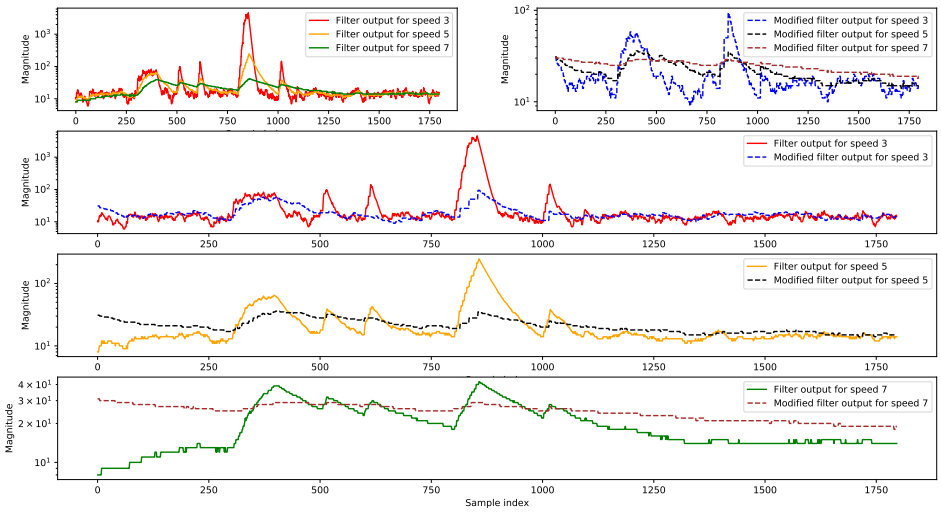

Figure 4.5: Comparison between filters before and after modification for different time constants of the same input noise. The current modification version of the filter is compared for the time constants $n = 3$, 5, and 7 (labelled speed, refers to n in equation 3.1).

for the time constants 3, 5, and 7. Comparing to Fig. 4.1, the modified filter uses every five data to estimate the noise (g=2, $L_g = 4$ in this case) instead of using each data value. It is not as easily affected by the noise, thus it responds to the huge noise more slowly than the previous filter.

4.4.1 Group length

In order to assess the performance of the filter before and after modification, the noise from an ADC is simulated by sampling artificial noise from a normal distribution with mean $\mu = 0$ and constant standard deviation $\sigma = 8$ added to a constant baseline of 13500. For each trial, 10^6 input data are used, with the output corresponding to the final 10^5 data being analyzed. The precise value of the baseline is arbitrary since it will be subtracted as background, but for the test to be well-behaved, the signal must be non-negative. It should therefore be larger than any negative noise value that can be reasonably expected. Moreover, to avoid overflow in the 16-bit input, the signal must not exceed $2^{16} - 1$. A baseline value of 13500 fulfills these conditions for all test parameters.

With a fixed noise level in the input, two degrees of freedom remain which influence the performance of the filter. These two parameters are the **group length** $L_g = 2^g$ and **filter coefficient** $L_n = 2^n$, where the time constant n and **group size** g are integers. For the purpose of these tests, they are restricted to the intervals $[3 \leq n \leq 12]$, and $[1 \leq L_g \leq 8]$. n dictates the decay rate of the infinite impulse response, and a value smaller than 3 causes it to decay too quickly to converge. Similarly, values above 12 result in convergence that is too slow to be useful for these tests, in addition to requiring slightly more resources for accurate arithmetic. The group size g simply determines the number of neighbouring data points used to subtract the background, and values of g above 3 do not offer substantial benefits in terms of estimating the local slope, while with each increment doubling the number of 16-bit pipeline registers required to perform the moving average background subtraction.

To characterize the performance of the filter, the mean relative error is calculated as

$$\varepsilon = \left| \frac{1}{N} \sum_{i=1}^{N} \frac{\frac{y(i)}{2 \cdot L_g} - \sigma \cdot c}{\sigma \cdot c} \right|, \tag{4.2}$$

where N is the number of outputs summed over, $L_g = 2^g$ is the group length, σ is the noise level of the input, c is the correction factor for the mean absolute deviation and increased noise from the background subtraction, and where the sum over i from 1 to N should be understood to encompass the outputs obtained from the final 10^5 inputs out of a total of 10^6.

In Fig. 4.6, which shows the results for a selection of time constants, the mean relative error [16] of the filter before and after modification is for all time constants smaller than 0.57%, meaning the estimated magnitude of the noise is quite close to $\sigma \cdot c$. The two filters follow different trends, with errors being particularly large for the modified filter at group lengths 1 and 8 for time constants 7, 9 and 11. While the old filter has better worst-case behaviour, the overall best performance is for

group length 4 for the modified filter. The underlying cause of these trends, and why they change behaviour, should be further investigated.

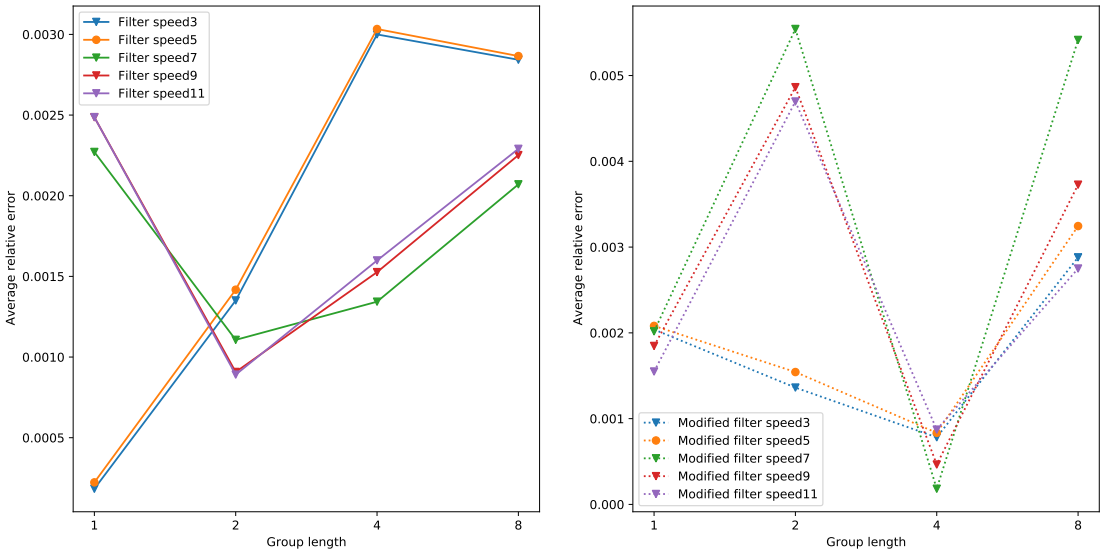

Figure 4.6: The mean relative error of the previous and modified filter with different group lengths from 1 to 8, of signals with the same noise fluctuation ($\sigma = 8$) and filter time constants from 3 to 11. The group length 4 has the best performance for the modified filter. The reason why the performance of filter time constant 7, 9, 11 differ from the time constant of 3 and 5 should be further investigated.

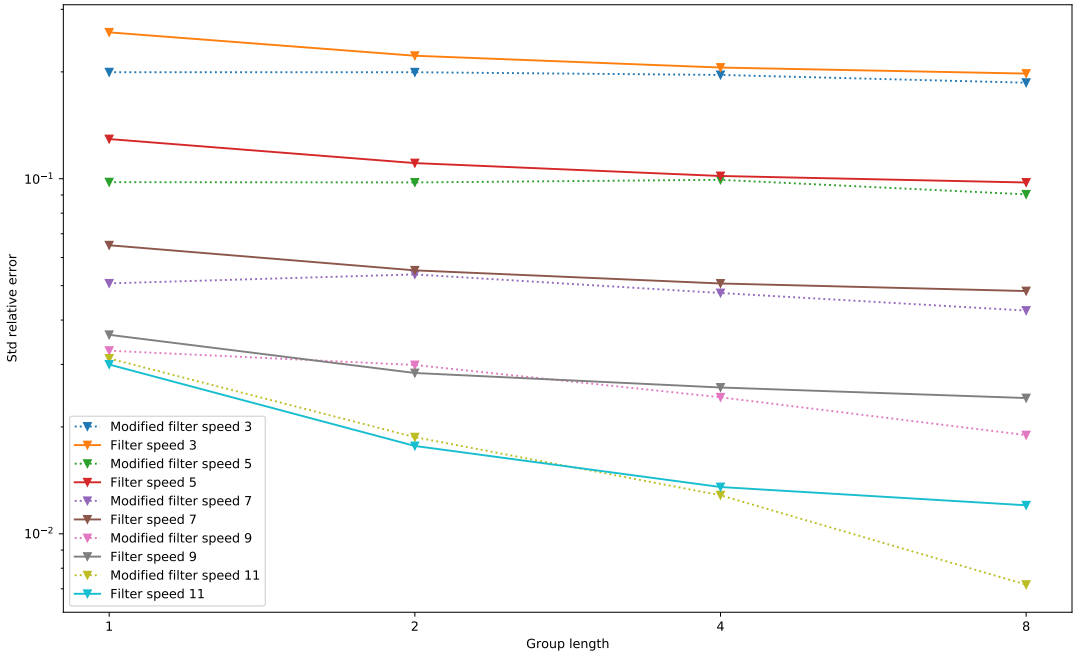

Figure 4.7: The standard deviation of the relative error of the noise estimate with different group lengths from 1 to 8 of same noise level ($\sigma = 8$) and filter time constants ranging from 3 to 11.

In Fig. 4.7, it can be seen that the standard deviation of the relative error decreases when the group length increases. The decrease is proportionally greater for larger time constants. This decrease occurs because the subtracted group noise is smaller (per equation (4.1)) and therefore the estimate is less affected by fluctuations in it. The standard deviation of the error is also seen to be smaller for larger time constants, as expected, because the IIR filter is less responsive to any one data.

4.5 Simulated MAD estimation in Python

Since VHDL code easily deals with multiplication and division by powers of two using left or right shifts, it excels at integer, rather than floating-point, arithmetic. It is therefore interesting to investigate to what extent the usage of integer arithmetic affects the behavior of the filter. This was done by implementing the filter in Python. It was given the same integer input as the VHDL implementation(16-bit input), but with floating-point arithmetic used in the calculations. The comparison between the results of the two implementations is shown in Fig. 4.8. The output from the VHDL filter using integer arithmetic is very similar, and follows the same trend as the Python implementation using floating-point arithmetic. Therefore, the use of integer arithmetic in estimating the MAD does not appear to influence the accuracy to any significant degree.

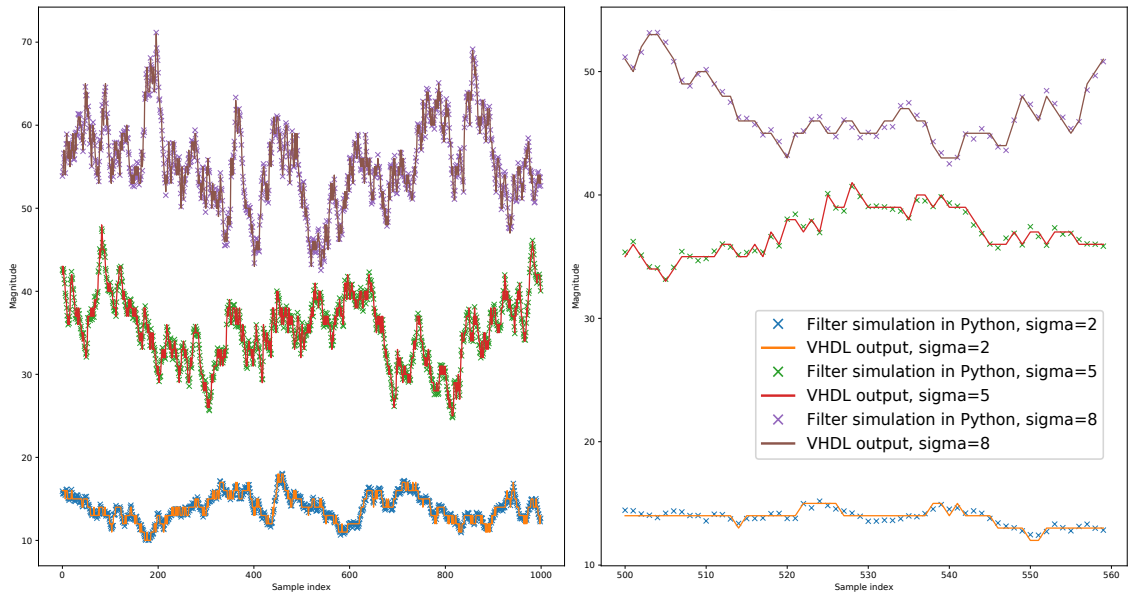

Figure 4.8: VHDL result with integer calculation and Python simulation result with floating point calculations for three different noise cases. The right panel is a zoom of a part of the left figure. The results are very similar, although not identical (as is expected when comparing calculations using floating-point and integer arithmetic).

4.6 Adaptive downsampling performance

Before implementing the algorithm of adaptive downsampling in VHDL, it is first implemented in Python to investigate the compression results for different $R_{\max}$ values and to improve the algorithm. The Python simulation results provide a reference for the VHDL implementation since the Python language supports structured and functional programming. Unlike VHDL, one does not need to account for clock-based synchronization in a Python implementation. Thus, the results are given by the Python and VHDL code. This part of the thesis describes and discusses the results of the codes for different maximum compression ratios, the effect of the trigger level K and the resource utilization and timing analysis of the VHDL design on FPGAs.

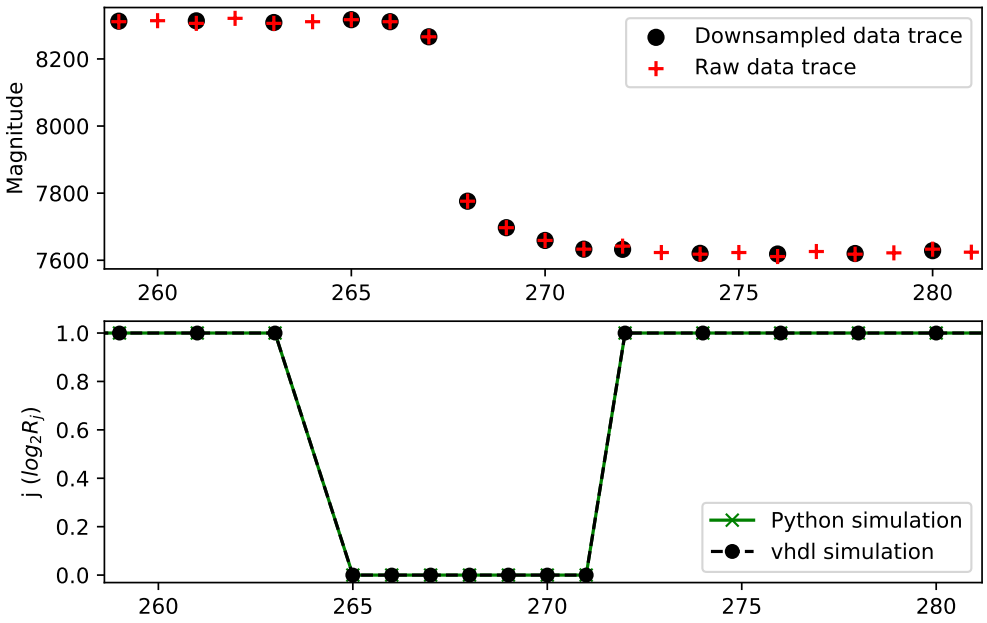

Figure 4.9: The upper subplot illustrates the compressed data trace after adaptive downsampling compared to the raw signal data trace. The lower subplot shows the two-logarithm of the downsampling ratio, going from 1 to 0 – that is, the ratio itself going from 2 to 1.

4.6.1 Maximum compression ratio 2

In Fig. 4.9, the trigger level K is set to 2, the standard deviation of the signal data trace is 8 and the maximum compression ratio is 2. The result of compression j ($j = \log_2 R_j$) in VHDL is identical to the Python simulation result. There are two possibilities for the compression ratio of the local sample data. j=0 stores the larger signal data separately, while j=1 stores two local signal data values as a group. If the local data differ significantly from the neighboring data, the compression ratio decreases to 1 and the large data is stored itself. Otherwise the compression ratio stays at 2, which means two signal data can be grouped together. Fig. 4.9 shows the compression output of the VHDL and Python simulation results. When the

local data is grouped together as a two data-word group, the mean value of the data group is the compression output replacing the signal data trace.

4.6.2 Maximum compression ratio 4

Fig. 4.10 shows the compression ratio for the local sample groups when the trigger level K is set to 2, the standard deviation of the signal data trace is 8 and the maximum compression ratio is 4. If the local signal data is too different from the surrounding data, the compression ratio decreases to a smaller value. Therefore, there are three possibilities for the sample group lengths based on the compression ratio, they are one data stored for $j = 0$, two data-word as a group for $j = 1$ and four data-word as a group for $j = 2$ respectively. Fig. 4.10 presents the group mean values replacing the sample groups data to track the signal trace. The smaller group lengths are used to store the important signal in the area containing the significant signal information.

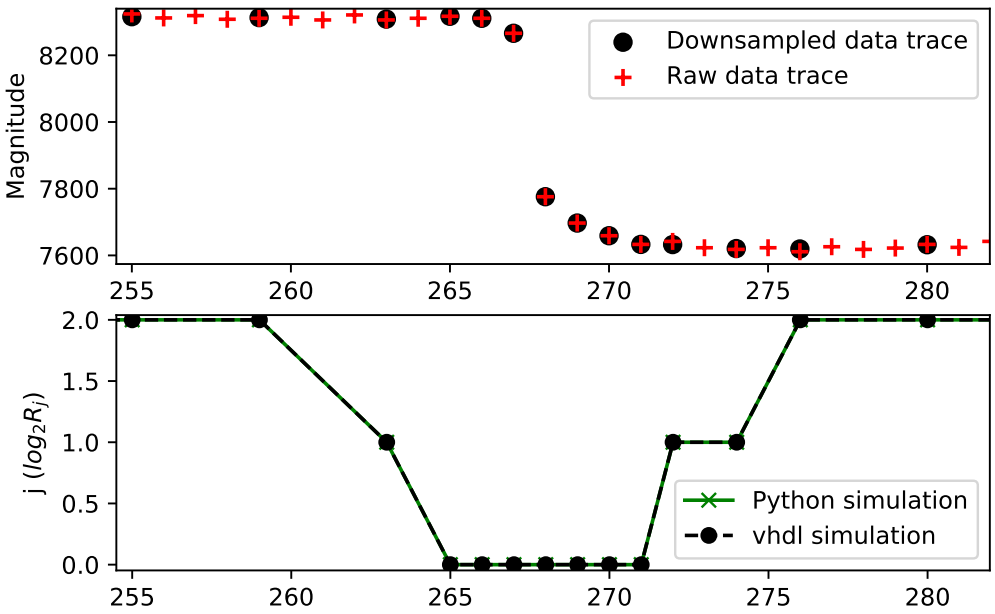

Figure 4.10: The upper subplot shows group mean values after downsampling. When $j = 0$, the compression ratio for the local signal sample is 1, it means the local data differ from the neighboring data, it needs to be stored separately. When $j = 1$, two signal data are grouped together. When $j = 2$, the signal data is stored in four data-word sample groups. The lower subplot is zoom in the figure of the selection of the compression ratio for the local sample groups. For Data(255) to Data(258), four local signal data are collected into one sample group for $j = 2$. Data(263) and Data(264) are stored together for $j = 1$. Since the values of Data(265) to Data(271) change a lot compared to the surrounding signal data, they are kept separately.

4.6.3 Maximum compression ratio 8

In the lower subplot of Fig. 4.11, it shows the group length based on the compression ratio of VHDL and Python simulation result when the trigger level K is 2, the standard deviation of the signal data trace is 8 and the maximum compression ratio is 8. Similar as the maximal compression ratio of 2 and 4, there are four choices for the compression ratio to group the local signal data. They are one data stored for $j = 0$, two data-words as a group for $j = 1$, four data-words as a group for $j = 1$ and eight data-words as a group for $j = 3$ respectively. The figure of the signal trace after downsampling compression by replacing the sample group data with the group mean value is presented in Fig. 4.11 upper subplot.

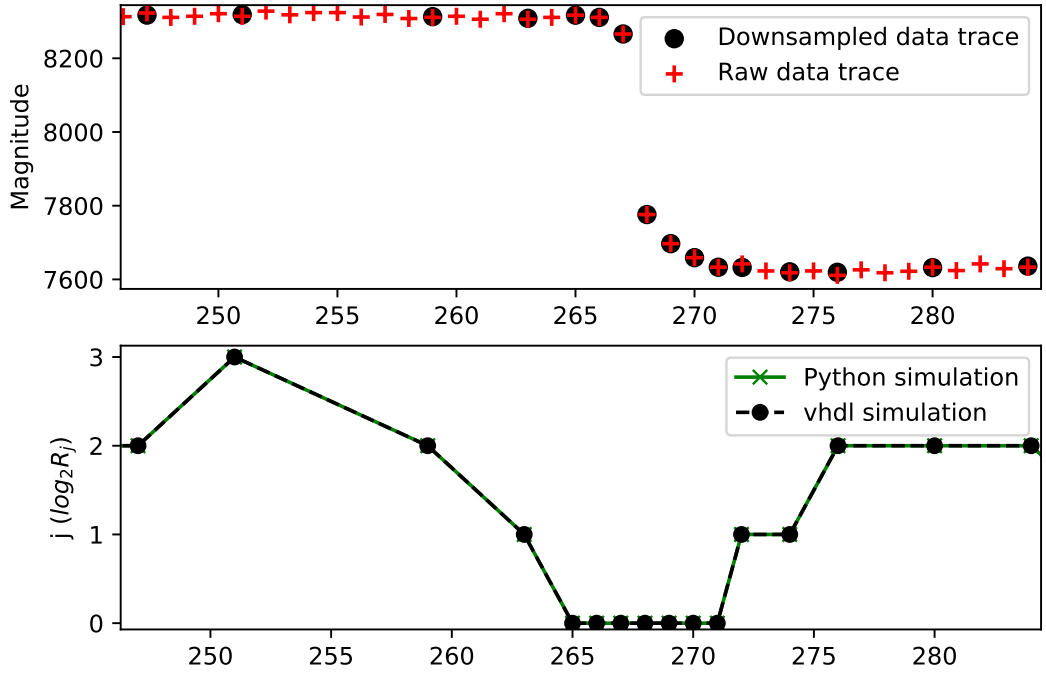

Figure 4.11: The lower subplot illustrates the selection of the compression ratio for the local sample groups. The compression ratio is $R_j = 2^j$. The upper subplot is a zoom of an interesting area from Data(265) to Data(271). To store the significant signal data, the compression ratio decreases gradually to track not only the large local signal but also keep the surrounding data in smaller sample groups. Thus, j decreases one by one from 3 to 0.

4.6.4 Maximum compression ratio 16

Fig. 4.12 presents the downsampling compression result with the trigger level K =2, where the standard deviation of the signal data trace is 8 and the maximum compression ratio is 16. Comparing to the $R_{\max} = 8$, the compression ratio at most can achieve 16 data-word as a group.

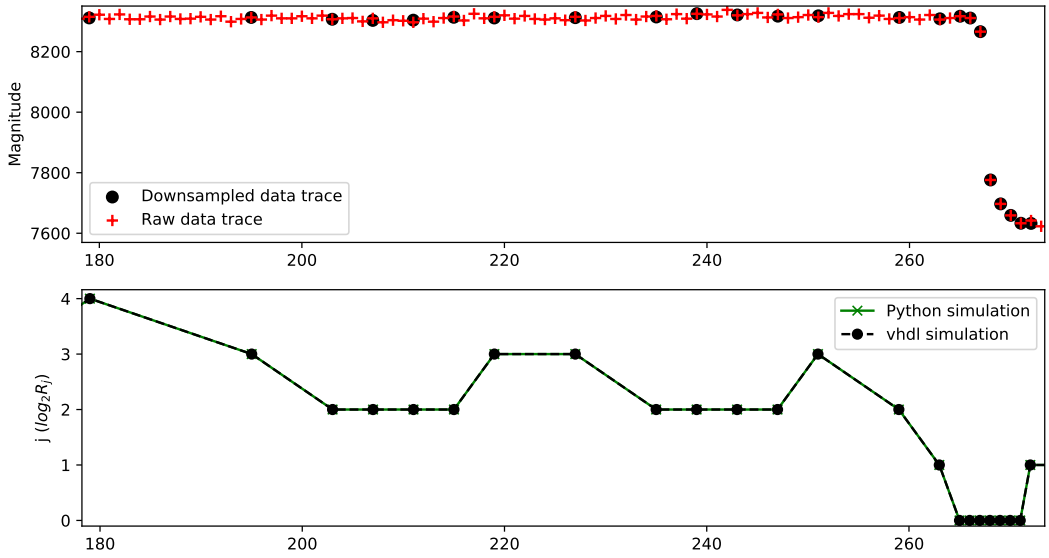

Figure 4.12: The lower subplot illustrates the downsampling compression ratio selected for a signal trace. The range of j is from 0 to 4, so the compression ratio for the local signal sample corresponds to 1 to 16. The upper subplot focuses on the data from Data(180) to Data(260), the compression ratio decreases to one smaller value or remains unchanged until the large different Data(265) is stored separately

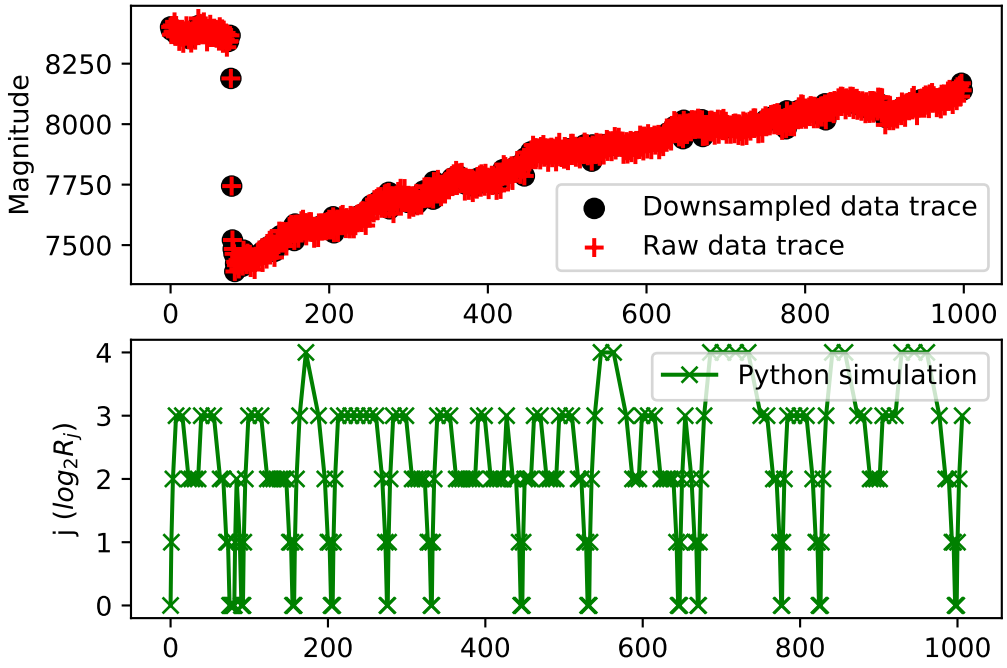

Figure 4.13: The downsampled result for the experimental signal trace with 14 bits, standard deviation is 23, trigger level K = 2 and the maximal compression ratio 16.

4.7 The maximum compression ratio for experimental data

To investigate the greatest attainable compression ratio of the adaptive downsampling, a real experimental signal trace is used. Since the signal data has a value around 8000, the number bits of the signal is 14. By calculating the standard deviation of the flat part of the signal data, the standard deviation for the experimental signal trace is rounded to the nearest whole number. If we set the trigger level to 2, the maximum used compression ratio is 16, as is presented in Fig 4.13. The rapidly decreasing data from Data(75) to Data(81) is stored as individual data points and other signal data can be grouped by 16 at the most. However if we continue to increase the maximal compression ratio to 32, the result of the downsampling is the same as the case of maximal compression ratio of 16. The signal data is not compressed as a large data group of 32. Fig. 4.14 shows the maximal sample group length is 4 even though the maximal compression ratio changes to 32. Therefore, the maximal compression ratio the adaptive downsampling can achieve for this signal trace is 16. Fig. 4.15 shows the average downsampling ratio as a function of the maximum compression ratio for four experimental traces.

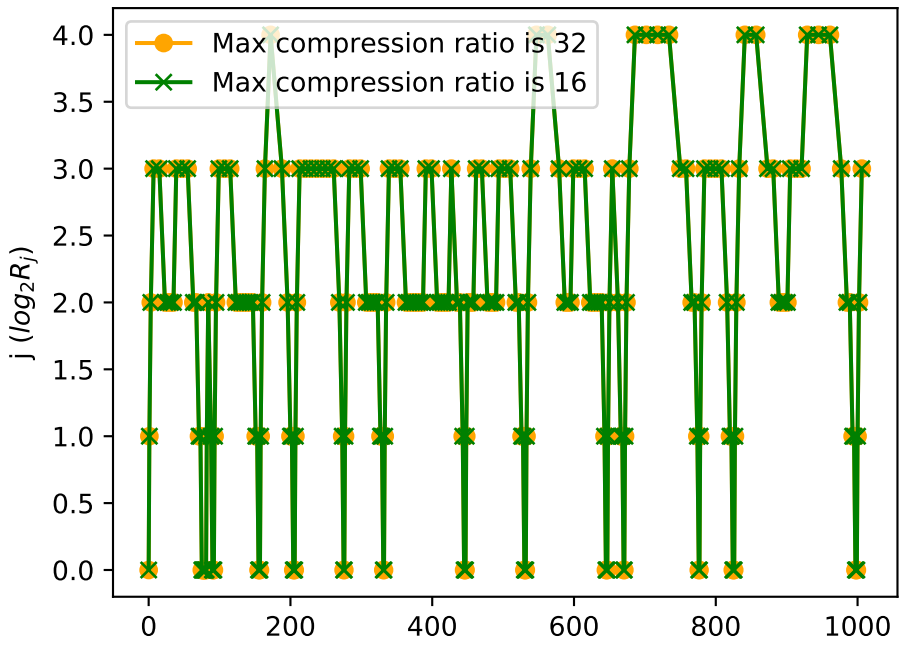

Figure 4.14: Illustration of the compression ratio for the signal trace. If trigger level to 2 and the maximal compression ratio is 32, in this case, the selection of ratio is the same as the maximal compression ratio of 16.

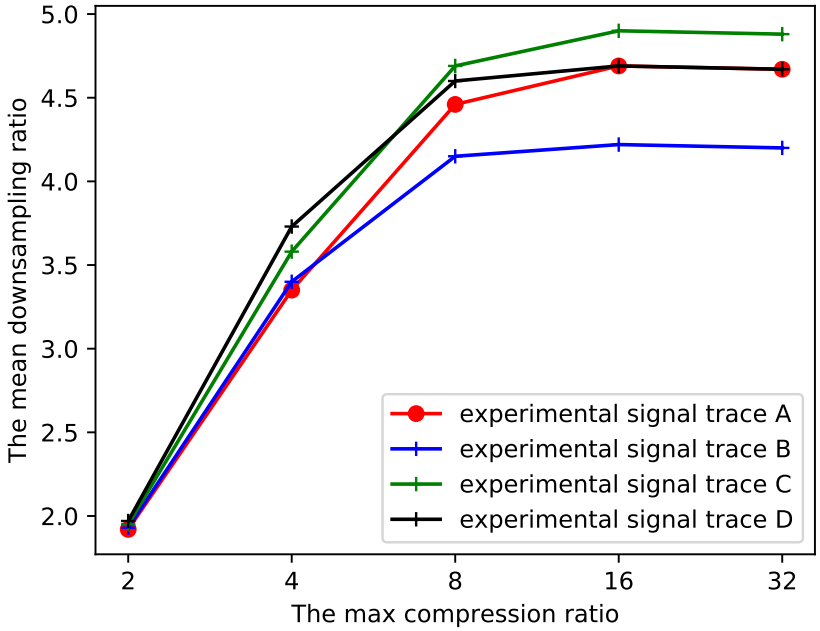

Figure 4.15: Illustration of the mean downsampling ratio for different experimental signal traces with the trigger level 2. The mean downsampling compression ratio does not improve when the maximal compression ratio is increased beyond 16.

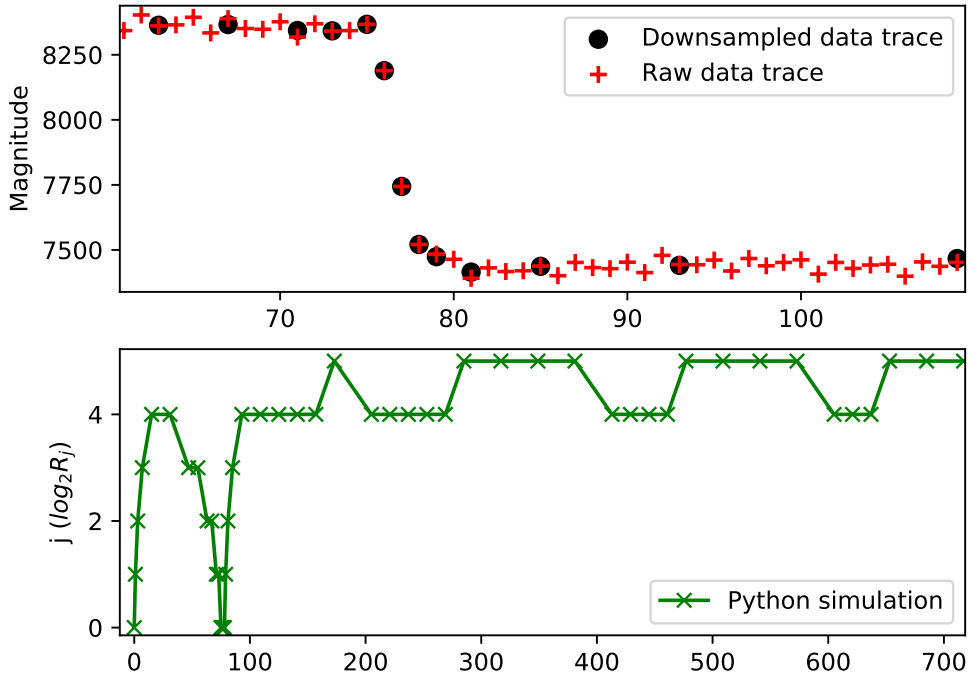

Figure 4.16: The result of downsampled output when the maximal compression ration is 32 and the trigger level is 5. Comparing to Fig. 4.13, the larger compression ratio of 32 is in this case used when the value of the trigger level is increased.

To continue to investigate the influence of larger trigger level in the maximal achievable compression ratio, the trigger level is increased to 5 instead. In Fig. 4.16, the

result of this can be seen; the downsampling is more aggressive than with a lower trigger level and the downsampling ratio is able to reach 5. In this case, the noise level is small compared to the abrupt change in the signal between Data(70) and Data(80), so the fidelity of the downsampled signal is not affected much. Generally, making use of a large downsampling ratio requires a larger trigger level to be chosen, as there is a larger probability of any one data point exceeding the threshold for downsampling by chance even in a noise-dominated region in the larger groupings.

4.8 The effect of the trigger level K

The trigger level K is a tunable parameter for the compression, which determines the threshold for downsampling relative to the noise level, as per equation (3.9). If K is small, the compressed signal trace will be more noisy and larger in terms of storage space, but reproduce smaller fluctuations in the original data. If K is large, the compressed signal trace will be less noisy and small in terms of storage space, but only reproduce large fluctuations relative to the noise. This is illustrated for three values of K in figure 4.17.

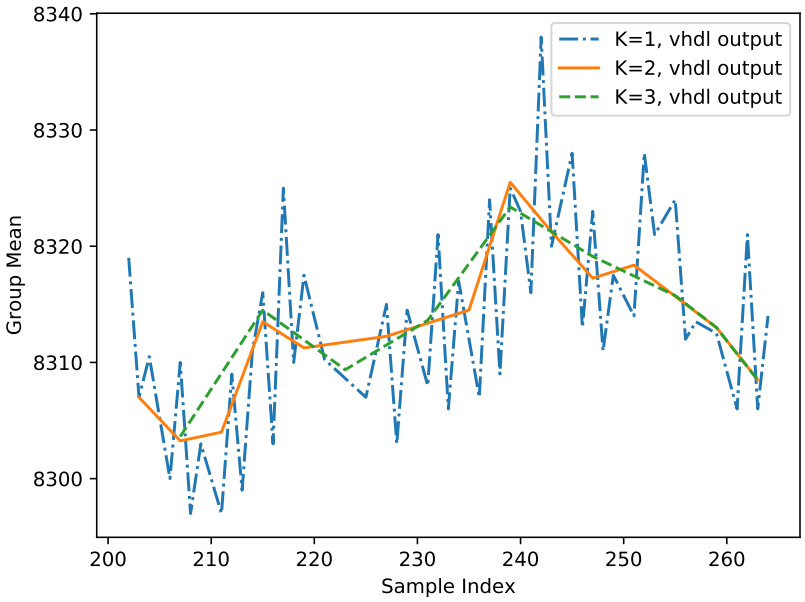

Figure 4.17: Comparing the effect of the compression threshold on a signal with $\sigma = 8$ and $R_{\max} = 8$. With a threshold $K = 1$, the signal trace is very noisy. When $K = 2$ and $K = 3$, the trace is much less noisy, but also reproduces changes in the original signal trace less accurately.

4.9 Performance and resource utilization on FPGAs

Table 4.1 shows the final amount of LUTs and FFs after the design of adaptive downsampling is physical optimized and full implemented on two different FPGAs, which are Artix-7 xc7a50tcsg324-1 and Spartan-7 xc7s75fgga484-1. An LUT is a configurable truth table, and a FF is a register for saving and synchronizing the logic state of a signal between clock cycles on FPGA. Table 4.2 and Table 4.3 present the resource utilization and available maximum frequency that can be used in an implementation on hardware when the maximal compression ratio is 8 for Artix-7 FPGA and Spartan-7 FPGA.

The Artix-7 and Spartan-7 have the same number of LUTs and FFs since they all implement the same design code for the compressor and have the same underlying FPGA building blocks. As the maximal compression ratio increases, more registers are used to store the different values and sum values for groups of data as well as to synchronize the output so that larger number of LUTs and FFs are used after the design route optimizations. The Artix-7 FPGA includes 20800 LUT elements and 41600 FFs elements, and the Spartan-7 FPGA has 48000 LUT elements and 96000 FFs. In this case, the maximal resource usages of the compressor on the FPGA of Artix-7 and Spartan-7 ($R_{\max} = 16$) are 3.17% and 1.37% for LUTs, 1.56% and 1.35% for FFs respectively. When $R_{\max} = 16$, the design utilizes more area on the chip and it also has slower clock frequency (200 MHz) and larger critical path lengths.

When the maximal compression ratio changes from 2 to 4, the usage of LUTs increases by 52% (89/171). Similarly, when the maximal compression ratio changes from 4 to 8, the usage of LUTs increases by 50% (130/260). However, when the maximal compression ratio is 16, the usage of LUTs increases more than 50%, using 69% (269/390) more than when the ratio is 8. One possible reason for the large number of LUTs increasing is from the comparison of different values for larger number of local groups.

For details on the calculation of the registers used and the longest critical path for the compression, see section 3.1.4.1−the compression ratio. When the maximal compression ratio is 16, to decide if the local signal can be grouped as the 16 data-words sample group or less group length, 31 number of 8 data-words groups, 39 number of 4 data-words groups, 43 number of 2 data-words and one data around the data need to be compared to the neighboring data groups. These difference values are stored in the register bits, thus there are at least 158 times of comparisons to decide the local group length and each comparison produces one more LUTs.

To align the output of sum values for each local sample groups, there are at most 32 clock delays for 20-bits output, so there are at most 40 LUTs produced for one compression ratio. There are about 160 LUTs produced for different ratios due to the signal synchronization. It explains the reason why the number of LUTs increases more when the case of the maximal comperssion ratio is 16. Fig. 4.18 illustrates the frequency trend as the number of input signal bits increase, and how the maximal frequency is related to the maximal compression ratio, for different FPGAs. Notably, the performance of the implementation is better when the input uses 16 bit data

words than for 14 bit data words. In this case, it may be that more efficiency is gained in the optimization by using a multiple of 8 bits than is lost by making the data words larger. Regardless of the exact cause, the difference is small and the overall trend of decreasing maximum frequency is right, even if the decrease is not totally smooth.

Fig. 4.19 presents the trend of the resource usage for the number of signal bits in the range from 8 to 16 and the maximal compression ratio in the range from 2 to 16 on FPGAs.

$R_{\max}$	FPGA	LUTs	FF	Clock period	Max Frequency
2	Artix-7 xc7a50tcsg324-1	171	208	4.40 ns	227 MHz
	Spartan-7 xc7s75fgga484-1	171	208	4.45 ns	225 MHz
4	Artix-7 xc7a50tcsg324-1	260	264	4.50 ns	222 MHz
	Spartan-7 xc7s75fgga484-1	260	264	4.46 ns	224 MHz
8	Artix-7 xc7a50tcsg324-1	390	393	4.56 ns	219 MHz
	Spartan-7 xc7s75fgga484-1	390	393	4.57 ns	219 MHz
16	Artix-7 xc7a50tcsg324-1	659	648	4.90 ns	204 MHz
	Spartan-7 xc7s75fgga484-1	659	648	5.00 ns	200 MHz

Table 4.1: Resource utilization and maximum frequency for different FPGAs when the maximal compression ratio is 2, 4, 8 and 16. The compression is synthesized with $A_{bits} = 16$ signal data bits from the ADC.

A_{bits}	LUTs	FFs	Clock period (ns)	Max Frequency (MHz)
8	250	257	4.36	229
10	285	291	4.43	226
12	320	325	4.48	223
14	355	359	4.7	213
16	390	393	4.56	219

Table 4.2: Resource utilization and maximum frequency for different number of signal bits on Artix-7 FPGA when the maximal compression ratio is 8.

A_{bits}	LUTs	FFs	Clock period (ns)	Max Frequency (MHz)
8	250	257	4.35	230
10	285	291	4.38	228
12	320	325	4.4	227
14	355	359	4.6	217
16	390	393	4.57	219

Table 4.3: Resource utilization and maximum frequency for different number of signal bits on Spartan-7 FPGA when the maximal compression ratio is 8.

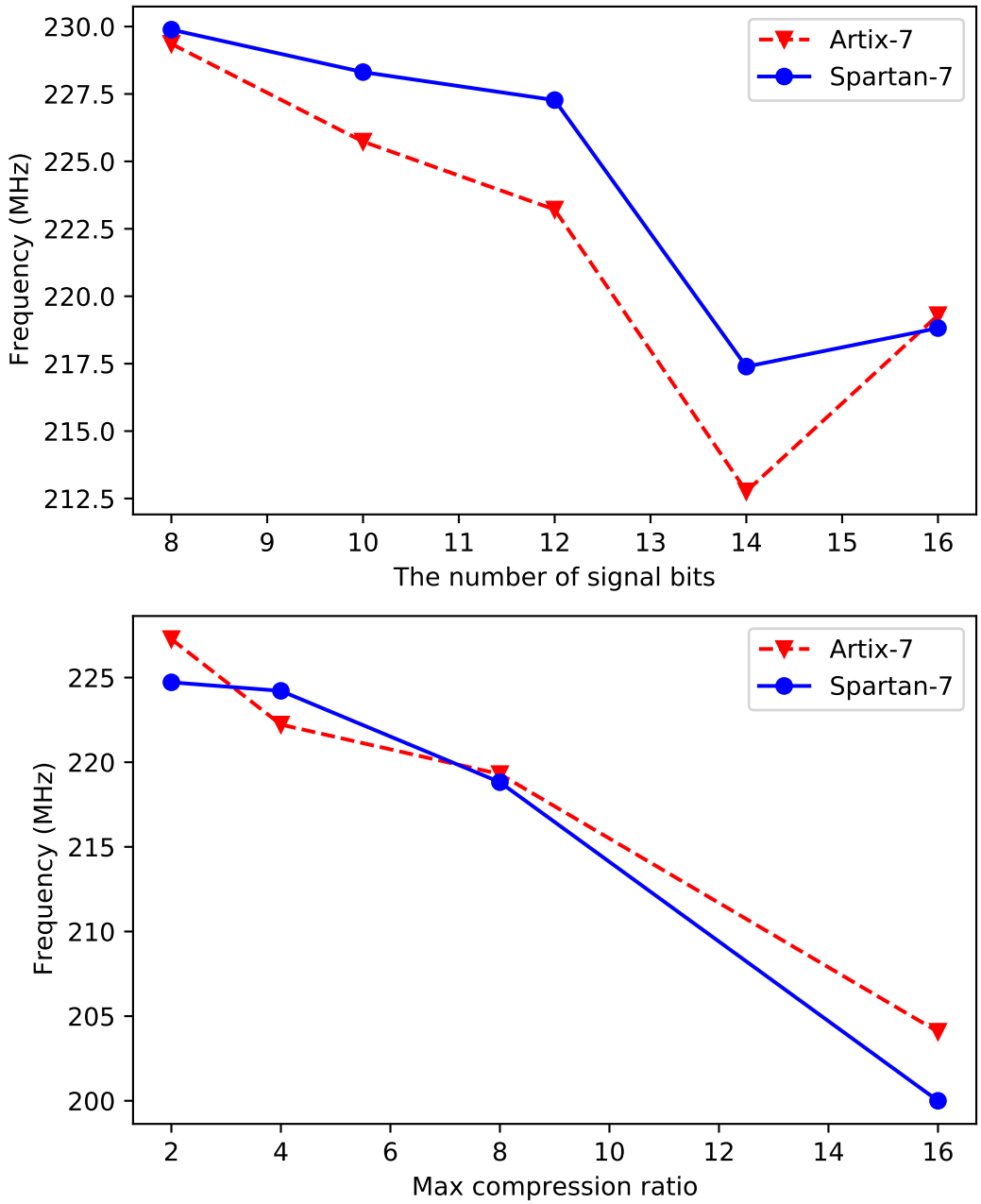

Figure 4.18: Comparison between the number bits of input signal for the different FPGAs and how the maximal frequency is related to the maximal compression from 2 to 16.

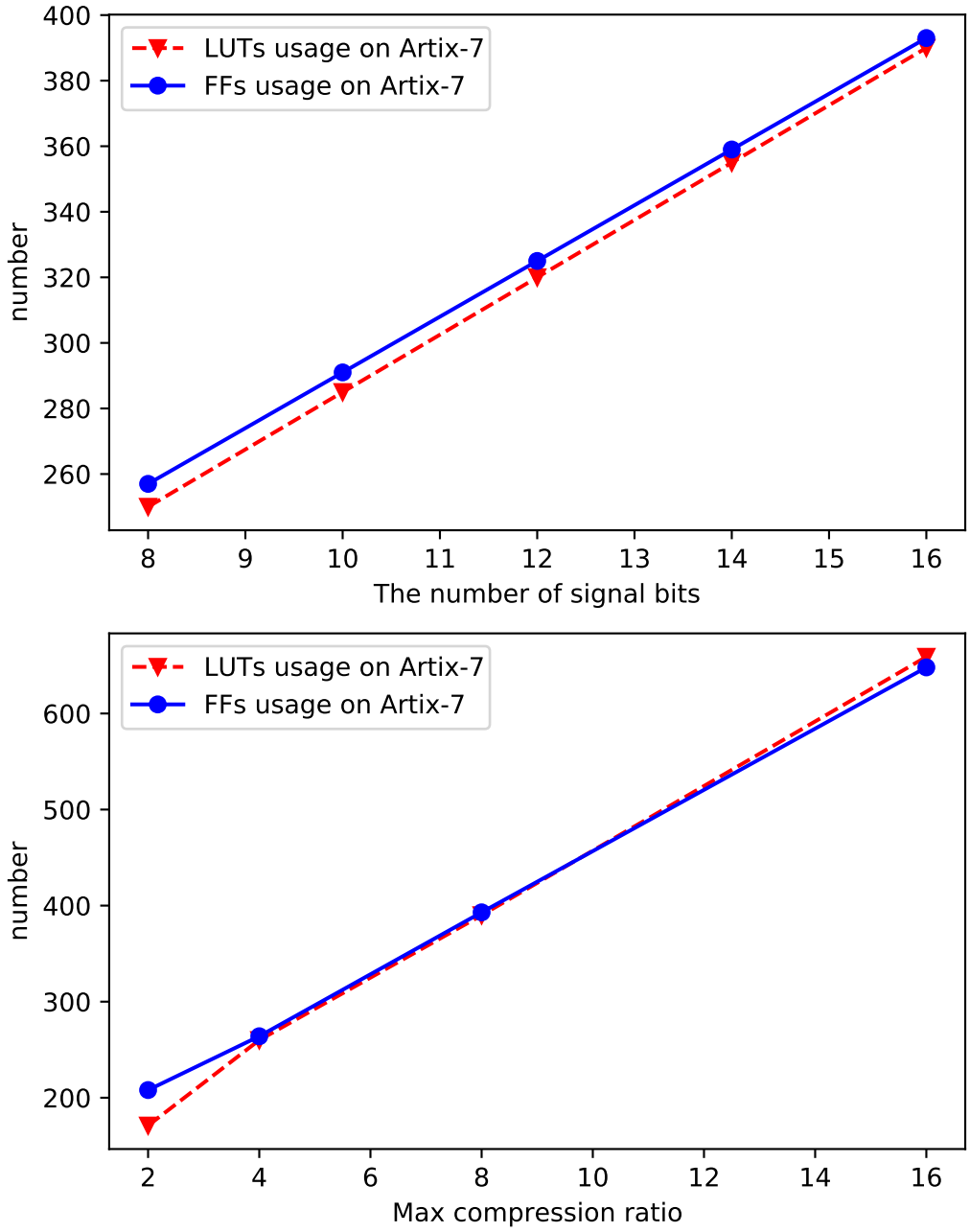

Figure 4.19: The upper panel shows the trend of the resource usage for various number of signal bits and it presents how much more resources are needed for each additional signal bit. The lower panel presents the trend of resource utilization for the increasing maximal compression ratio on Artix-7. The number of LUTs and FFs used on Spartan-7 is is identical with Artix-7 and they have the same trend. The lower panel shows an almost exponential behavior in j.

5

Conclusion

This project provides a generic VHDL design to describe how large group lengths and the maximal compression ratio can be used to store signal traces efficiently, focus on regions of predominantly high signal information content. To determine the choice of the compression ratio of the local signal samples, it also provides an algorithm of mean absolute deviation to estimate the noise. Thus, based on the noise estimation and the trigger level, the downsampling ratio is selected under the constraint that signal traces use smaller group lengths to store the significant signal in regions dominated by signal information but larger group length in areas of noise. This project also provides the Python code for the design, which is compared with the VHDL code to improve the algorithms since the VHDL code is good at dealing with an integer rather than floating-point arithmetic. Thus, Python code to implement the algorithm of noise estimate and downsampling is a good starting point and reference to evaluate the algorithm design for VHDL code. It is also interesting to investigate to what extent the integer arithmetic operation affects the performance and result of the design.

To estimate how much resources are used by the design and how efficient this VHDL code works, synthesis tests on different FPGAs have been performed. The influence of the configuration parameters, including the group length, trigger level, the maximal downsampling ratio, and the number of bits of signal trace in the performance of the design, have been shown in this project.

The future work of the adaptive downsampling design could focus on testing the whole system design combining the noise estimation with downsampling compression, and comparing the result of the whole system to the current result to investigate if they have the same performance of downsampling. In addition, it is also valuable to investigate how to use more pipelines or RAM to optimize the VHDL code to decrease resource usage, especially when the maximal compression ratio is larger than 8.

Bibliography

[1] Anton Fredriksson and Lukas Rahmn. "Adaptive downsampling of traces with FPGAs". MA thesis. 2020. Chalmers, Gothenburg.

[2] Peter Alfke. "Xilinx virtex-6 and spartan-6 FPGA families". In: *2009 IEEE Hot Chips 21 Symposium (HCS)*. IEEE. 2009, pp. 1–20.

[3] D Cortina-Gil et al. "CALIFA, a Dedicated Calorimeter for the R3B/FAIR". In: *Nuclear Data Sheets* 120 (2014), pp. 99–101.

[4] J. Hoffman. *FEBEX3/16, preliminary specification*. `https://www.gsi.de/fileadmin/EE/Module/FEBEX/febex3.pdf`.

[5] *AD9252 Data Sheet. Version REV. E. Analog Devices, Inc. 2017.* `https://www.analog.com/media/en/technical-documentation/data-sheets/AD9252.pdf`.

[6] Marcel JM Pelgrom. "Analog-to-digital conversion". In: *Analog-to-Digital Conversion*. Springer, 2013, pp. 325–418.

[7] Bernard Mulgrew, Peter Grant, and John Thompson. *Digital signal processing: concepts and applications*. Macmillan International Higher Education, 1999.

[8] *Digitizer Families*. `https://www.caen.it/sections/digitizer-families/`.

[9] *Digitizer overview table*. `https://www.struck.de/vme.htm`.

[10] Giovanni Bruni. "How to Lighten Experimentalists' Life with Electronics Development of Front-End Readout for the CALIFA Detector". MA thesis. 2017.

[11] Giovanni Bruni and Håkan T Johansson. "DPTC—An FPGA-Based Trace Compression". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 67.1 (2019), pp. 189–197.

[12] Chin-Fa Hsieh et al. "Implementation of an Efficient DWT Using a FPGA on a Real-time Platform". In: *Second International Conference on Innovative Computing, Informatio and Control (ICICIC 2007)*. IEEE. 2007, pp. 235–235.

[13] Li Tan and Jean Jiang. "Novel adaptive IIR filter for frequency estimation and tracking [DSP Tips&Tricks]". In: *IEEE signal processing magazine* 26.6 (2009), pp. 186–189.

[14] Hiroshi Konno and Tomoyuki Koshizuka. "Mean-absolute deviation model". In: *Iie Transactions* 37.10 (2005), pp. 893–900.

[15] J Susan Milton and Jesse C Arnold. *Introduction to probability and statistics*. Vol. 4. McGraw-Hill New York, 1990.

[16] X Rong Li and Zhanlue Zhao. "Relative error measures for evaluation of estimation algorithms". In: *2005 7th International Conference on Information Fusion*. Vol. 1. IEEE. 2005, 8–pp.