FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING
DEGREE PROGRAMME IN ELECTRONICS AND COMMUNICATIONS ENGINEERING

MASTER'S THESIS

WIRELESS CHANNEL LOAD STRESS ANALYSIS USING FPGAS AT THE EDGE

Author	Chanaka Ganewattha
Supervisor	Adj. Prof. Zaheer Khan
Second Examiner	Dr. Vijitha Herath

June 2019

Ganewattha C. (2019) Wireless Channel Load Stress Analysis Using FPGAs at the Edge.
University of Oulu, Degree Programme in Electrical Engineering, 42 p.

ABSTRACT

One of the key usage scenarios of fifth generation (5G) and beyond networks is to provide mission critical, ultra-reliable and low latency communications (URLLC) targeting specific set of applications where low latency and highly reliable wireless links are of utmost importance. 5G and beyond applications that require URLLC links include industry automation, artificial intelligence based technological solutions, vehicle to vehicle communication and robotics enabled medical solutions. URLLC applications using wireless connectivity require that resource utilization, such as wireless channel utilization, does not exceed the levels above which performance can degrade. Real-time radio frequency (RF) data analytics at the wireless network edge can help to design proactive resource allocation solutions that can allocate more radio resources when a particular resource is forecasted to be under stress. Typically, real-time RF data analytics can require processing of hundreds of millions of streaming samples per second and hardware accelerated modules (such as FPGAs) are very well-suited for such processing tasks. We propose FPGA-accelerated real-time data analytics based resource stress forecasting method in this thesis. The proposed method is low in complexity and performs forecasting in real-time. We show its implementation on an FPGA of *Xilinx Zynq-7000* series System on Chip (SoC) board using *Vivado*, *Vivado HLS*, *SDK* and *MATLAB* tools. The proposed method uses quantile estimation and can be used for forecasting a variety of resource utilization scenarios. As an example, in our thesis, we focus on forecasting stress in wireless channel utilization. We test the implemented algorithm with real wireless channel utilization data representing block maxima series. We compare the results from the implemented method against the results from a theoretical method where the generalized extreme value (GEV) theory is used to make forecasts on the considered block maxima data. We show that with high accuracy and low latency, the proposed algorithm can perform the forecasting of channel utilization stress.

Keywords: URLLC, 5G, channel resource allocation, forecasting, generalized extreme value theory (GEV), channel utilization stress, Xilinx, ZedBoard, high level synthesis, FPGA.

TABLE OF CONTENTS

ABSTRACT
TABLE OF CONTENTS
FOREWORD
LIST OF ABBREVIATIONS AND SYMBOLS
1 INTRODUCTION 7
1.1 Real time streaming analytics for 5G and beyond networks 7
1.2 Use of FPGAs as RF analytic modules 8
1.3 Challenges in using FPGAs..... 9
1.4 Overview and contributions 10
2 RELATED WORK 12
3 THEORY 14
3.1 Physical layer channel utilization: background..... 14
3.2 Why only mean is not enough for the characterization of channel utilization data? 15
3.3 Extreme value theory 15
3.3.1 Quantile estimates based on extreme value theory..... 16
3.4 Derivation of a simpler version of the generalized extreme value theory 17
4 IMPLEMENTATION 20
4.1 System overview 20
4.2 Steps involved in implementation 21
4.3 Algorithm implementation..... 22
4.3.1 Histogram and cumulative sum IP modules..... 22
4.3.2 Percentile estimation IP 23
4.3.3 Usage of HLS for IP generation 25
4.4 IP integration and system generation..... 25
4.4.1 *AXI* protocol 25
4.4.2 Overall design flow of the implemented system 28
5 RESULTS AND DISCUSSION 29
5.1 Behavior of channel utilization data..... 29
5.2 Evaluation of the proposed algorithm 31
6 CONCLUSION 37
7 ACKNOWLEDGMENT 38
8 REFERENCES 39
9 APPENDICES 42

FOREWORD

This thesis is focused on studying wireless channel utilization load stress analysis in communication networks using FPGAs at the edge. This thesis was done as a part of the MOSSAF project at the Center for Wireless Communications (CWC) of University of Oulu, Finland. I express my sincere gratitude to my supervisor and mentor Adj. Prof. Zaheer Khan for his assistance and guidance throughout the period of the research. I would like to extend my gratitude to Prof. Matti Latva-aho, the head of the RAT group of CWC for his guidance. I would like to thank Dr. Pekka Pirinen, the project manager of CWC and Marja Matimikko-Blue of MOSSAF project for their support in my research. I would also like to thank Dr. Vijitha Herath from University of Peradeniya for his assistance in my research.

I thank my mother, father and my siblings for their immense support in my life. Finally, I thank my fiancee, Ruwandi for her endless support.

Oulu, 24th June, 2019

Chanaka Ganewattha

LIST OF ABBREVIATIONS AND SYMBOLS

URLLC	Ultra Reliable Low Latency Communication
GEV	Generalized Extreme Value
5G NR	5G New Radio
FPGA	Field Programmable Gate Array
BLER	BLock Error Rate
RF	Radio Frequency
CPU	Central Processing Unit
GPU	Graphic Processing Unit
ASIC	Application Specific Integrated Circuits
DSP	Digital Signal Processor
SoC	System On Chip
RTL	Register Transfer Level
HLS	High Level Synthesis
HDL	Hardware Description Language
GEV	Generalized Extreme Value
SDR	Software Defined Radio
KPI	Key Performance Indicator
NWDAF	Network Data Analytics Function
PCF	Policy Control Function
3GPP	3rd Generation Partnership Project
SELFNET	Self Organized Network
TST	Time Series Theory
EVT	Extreme Value Theory
POT	Peaks Over Threshold
GPD	Generalized Pareto Distribution
SP-POT	Sample Pruning Peaks Over Threshold
CU	Channel Utilization
PDF	Probability Density Function
IP	Intellectual Property
SDK	Software Development Kit
PS	Processing System
PL	Programmable Logic
I/O	Input Output
UART	Universal Asynchronous Receiver Transmitter
AXI	Advanced eXtensible Interface
DMA	Direct Memory Access
ACP	Accelerator Coherancy Port
MM2S	Memory Mapped To Streaming
S2MM	Streaming To Memory Mapped
ILA	Integrated Logic Analyzer

1 INTRODUCTION

1.1 Real time streaming analytics for 5G and beyond networks

One of the key usage scenarios of 5G networks is to provide mission critical, ultra-reliable and low latency communications (URLLC) which are targeted for specific set of applications where the latency is critical. For example, 5G targets to open new applications in industry automation, robotics enabled medical solutions and intelligent autonomous transportation solutions where ultra-reliability and low latency are of utmost importance [1]. New radio technologies like 5G new radio (5G NR) promises to provide an end to end latency of 1ms and a block error rate (BLER) of 10^{-5} [2] to allow for URLLC applications where extreme reliability is required. This kind of reliability and latency requirement in communication will enable new emerging technologies like autonomous driving, tele surgery and industrial automation.

The challenge of providing ultra-reliability and low latency is complicated when wireless communication network resource usage goes high. Imagine a situation where wireless communication channels experience utilization stress due to many users accessing applications at the same time that require high data rates and low latency. This channel utilization stress can result in low reliability and increased latency in the network. Typically, until now, the channel resource allocation has been done based on prediction and forecasting models that have been developed on data collected over 3 to 6 months or no less than 24 [3] hours. Also, such standard forecasting and prediction methods mostly rely on mean network usage statistics. However, URLLC applications need real time resource utilization prediction methods. Channel utilization under stressed scenarios may be modeled accurately by studying high quantiles and extreme channel utilization values, these cannot typically very well be described by using mean or average network usage statistics as these variations are not captured by them [4]. Therefore, such forecasting methodologies may render almost useless in tackling with real-time channel utilization under stress conditions. Some network controllers provide auto-scaling and load balancing mechanisms on network resources which are made dynamically based on the past data sets collected over time [5]. But these controllers in general reside in a centralized server and processing of large data sets in general can lead to slower response to the sudden needs in the network resources. They may incur significant overhead due to collection of large data sets required to perform reliable predictions. For example, a 20 MHz unlicensed wireless channel data can generate 2×40 Million or more IQ samples per second. Thus, there may be additional latency added which would result in not meeting the latency requirement of the URLLC applications. Therefore, there should be some RF predictive analytics at the network edge to characterize which can be used in effective resource allocation prediction and network scaling.

By incorporating real time streaming RF analytics modules at the network edge with workload characterization and prediction, more effective resource allocation can be made [6]. This would facilitate the network to take smart channel utilization decisions proactively in real-time which would enable it to deliver the guaranteed quality of service and relevant end to end performance metrics. Furthermore, this would help to take the decision in scaling the network to fulfill service level agreements with respect to performance metrics. This way, it would enhance network resource utilization, ensure the availability of the network, reduce overhead and reduce the overall energy consumption in terms of processing huge IQ data sets at the controller server. Therefore, real time streaming analytics can be considered very important

for 5G and beyond networks to ensure meeting the tight network specification in latency and BLER.

1.2 Use of FPGAs as RF analytic modules

5G and beyond wireless communication networks support huge data rates in excess of 100 Mbps. To make the network proactive in resource utilization and scaling, the channel utilization data should be collected and processed, and the predictions should be made on the channel utilization loads. To analyze this channel utilization data, it should be sent to a remote server where some prediction and forecast algorithms are run to take respective decisions in optimizing the network. Transmission of collected data to the server itself would impose a huge overhead on the network. Assume, anyhow the data is sent to the server for processing. Then again, it would constantly keep the server CPU busy consuming lots of CPU time to read the data and to do the analysis to make the required decisions on the network. In reality, a system like this is not practical as this applies a huge overhead on the network and the server computing resources itself. Therefore, the IQ usage data should be pre-processed somehow at the edge to extract the necessary statistical parameters which would help in analyzing the network load stress.

Field Programmable Gate Arrays (FPGAs) contain reconfigurable matrix of logic circuitry and interconnects. In the past, FPGAs were mainly used for application specific integrated circuits (ASIC) verification. But present FPGAs offer digital signal processing blocks (DSPs), integrated CPUs and huge amount of logic circuitry which are well suited for statistical signal processing applications and other computing intensive design implementations. System on Chip (SoC) devices containing FPGAs with processing systems can use the programmable logic hardware for accelerating the CPU in tasks where parallelism can be exploited to enhance the performance. Massive amount of parallel resources available in FPGAs can be used to implement complex operations, parallelized functions and pipelined designs. Very complex operations can be implemented to work within few clock cycles using the parallel resources available in the FPGA. Comparative to a CPU which does the execution of instructions sequentially, the performance of FPGA design is massive, though they tend to work with a lower clock frequency than a CPU. Thus, FPGA implementations can be high performance and energy efficient compared to a CPU when real-time RF data analytics are to be performed. This nature of FPGAs is very well suited for pipelining and real time streaming kind of data processing applications. Also, reconfigurability of FPGAs to taylor its processing for a specific application makes them well suited to be applied across a wider range of applications.

In this thesis, an FPGA based RF data analytics solution is proposed for channel utilization data processing and utilization load stress prediction. By using the parallel processing nature and pipelining property of FPGAs, the analytics processing system is implemented to process the channel utilization data which is required for real time analytics for load stress prediction. The processing in FPGA allows very small amount of processed data to be sent to the server for further processing. This way, the overhead on the network usage and the controller server CPU is greatly reduced.

1.3 Challenges in using FPGAs

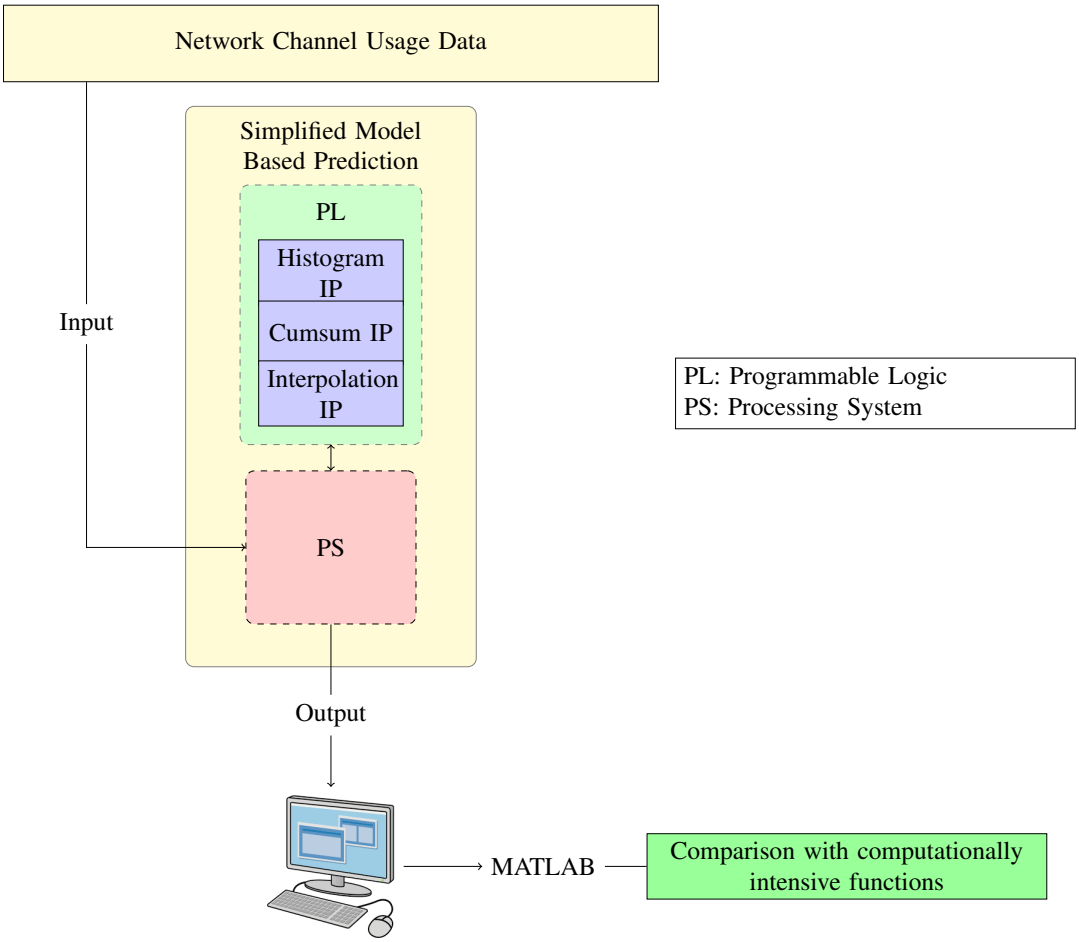

Figure 1. System Model

While GPUs (Graphic Processing Unit) and CPUs are suitable for computation of complex functions, one of the challenges in FPGAs is that complex and computationally intensive functions are not suitable for hardware and hence one need to come up with a computation method/model which is almost efficient at the same time involves simpler operations. Therefore, to take the performance advantage of FPGAs, the complexity of algorithms to be implemented should be reduced, and some algorithmic optimization should be done to take the benefit from the parallelism available in the FPGA and such that it suits the underlying architecture in the FPGA.

Although, FPGAs have huge performance advantage over CPUs and GPUs, they are not well suited for some specific tasks. For example, due to the slow clock frequencies used in FPGAs than in CPUs and GPUs, they are not well suited for sequential operations. Also, for complex mathematical expressions where the evaluations are done sequentially with data dependencies with previous steps, FPGA performs worse than a CPU or a GPU, in this kind of scenarios. Therefore, one should be careful enough to avoid or limit such operations which degrade the performance of FPGA implementations.

Basically, there are two approaches in implementing a design in an FPGA: Register Transfer Level (RTL) design and using High Level Synthesis (HLS) tools. In register transfer level

design methodology, the circuit level description is given by using a High-Level Design Language (HDL) like Verilog or VHDL. Today's FPGAs are so complex with tremendous amount of resources that, to implement an algorithm with moderate complexity would require the hardware designer to have thorough understanding in the underlying hardware platform and hardware design concepts. And, the designer should also have very strong HDL skills (in Verilog or VHDL) to make the implementation to take the maximum performance advantage out from the platform's offerings. Usually, the development time for RTL design are considerably higher due to requiring considerable time for architecture development and debugging and verification.

On the other hand, implementation of designs using HLS is much faster than in RTL design, that it takes less time to get from algorithm design phase to implementation phase in the design flow. In HLS, the high-level algorithm is coded in a programming language like C or C++ and then converted to HDL like Verilog or VHDL using HLS tools. To optimize the design to meet the required specification, tool dependent macros are often needed to guide the tool to achieve the required micro architecture. Some of the tools used for this purpose are Stratus, Catapult and Vivado HLS. Recent advances in the availability of optimized HLS tools such as Xilinx's Vivado HLS have made them very attactive for researchers to use FPGAs with little knowledge of Verilog and VHDL.

The designs resulting from RTL and HLS design methodologies differ very much in performance, area, resource utilization and design time. RTL designs tend to have better performance and lesser resource and area utilization than HLS designs. But designing in HLS with hardware architecture in mind and using a programming style oriented for the hardware, HLS designs may come closer to the properties of an RTL design. And with lesser development and verification time in HLS design flow than in RTL design flow, the difference between the area and resource usage could be justified.

Figure 1 gives an overview of the system model developed for the channel utilization load stress forecasting using an FPGA. The system has been developed keeping in mind the streaming nature of the network traffic usage data. Therefore, the system can be used for real time load stress forecasting and can be readily deployed in the field for this reason.

1.4 Overview and contributions

In this thesis, a solution is presented for channel utilization load stress analysis of communication networks using a Xilinx Zynq FPGA. As a proof of concept of this solution, real channel utilization information from a *Wi-Fi* network is used. The system extracts the real time streaming analytics from a *Wi-Fi* network's channel utilization information and calculates the load stress estimation using generalized extreme value (GEV) theory. As calculations related to generalized extreme value theory requires complex mathematical expressions to be solved, simpler version of the theory is derived which performs almost equal to the GEV, which greatly eases the FPGA implementation. A block maxima time series data is obtained from channel utilization time series which is collected by using a Xilinx Zynq system based ZedBoard. The board is connected to Analog devices AD-FMCOMMS3 software defined radio (SDR). A data window of 1 hour is selected, and empirical probability and cumulative distribution functions are obtained by using histogram and cumulative sum. Based on this information, the system computes the percentile estimation for a given probability of channel utilization.

The presented system is well suited for extracting real time streaming analytics with reduced complexity. The derived version of the GEV theory performs almost comparative to the original GEV. This system is designed to be flexible such that it can be used in different networks and applications where percentile estimates are utilized for different management purposes. Due to the usage of FPGA, the system has very high performance which is analyzed and discussed in a later chapter. Using this system, an accurate insight into the network resource utilization and stress on the network can be obtained which could be used in proactive management of network resources and scaling purposes. Thus, a system like this can be of utmost importance in 5G and beyond networks in achieving the promised performance metrics for URLLC applications.

2 RELATED WORK

Various types of non real-time data for network planning and management has been in usage [7, 8, 9]. However, for 5G and beyond networks usage of real time data analytics and mining techniques have been given tremendous attention at present [10, 11]. It has been identified that utilizing real time data analytics and data mining in wireless communication networks would enable dynamic network management, traffic engineering, radio access selection and network traffic steering [12] and improve overall network performance enabling a 5G and beyond network to meet the required key performance indicators (KPIs). Towards this end, the latest 3gpp specification [13] has introduced a dedicated function called Network Data Analytics Function (NWDAF) which is responsible to provide network analysis information upon request from other network functions. As an example, information on the load level of a certain network instance could be provided upon a request by a certain other network function. Policy Control Function (PCF) takes the input from NWDAF into consideration in deciding policies on network resources and network traffic steering suggest the importance of having data analytics in future communication networks.

Research community has given an immense focus on the application of data mining and analytics in 5G & beyond networks. The work in [14] presents a brief survey of several models proposed by the research community in this regard. It discusses the various types of data which should be collected for data mining purposes. The author in [14] discusses the NWDAF of 3GPP and its drawbacks. Context Extraction and Profiling Engine (CEPE) which can be used to build a user profile to describe the future behavior of a subscriber are also described in [14] and its integration to 5G architecture is also discussed. The author in [14] suggests the use of behavioral profiles in network functions like access and mobility management, session management and traffic steering to optimize the network for better performance. It does not provide a detailed discussion on how the data is extracted and processed to obtain the necessary information. In [15], a design of self organized network management in virtualized and software defined networks (SELFNET) analyzer module is presented. According to the paper, the objective of SELFNET is to facilitate proactive responses from network state diagnosis and data inferred from various aggregated metrics related to 5G mobile network infrastructure. The paper discusses in depth the design of the SELFNET analyzer module and its architecture. The authors claim that the SELFNET module is highly scalable that it can be adapted to new use cases with minimal effort.

Extreme value theory is a statistical data analysis tool which has been used by several research works to successfully forecast and predict time series data [16, 17, 18, 19]. The work presented in [16] introduces an application of extreme value theory in predicting cyber attacks using network data. It proposes a methodology for the prediction of attack rates in the presence of extreme values in observed data. According to the authors, the cyber attacks usually are assumed to exhibit extreme value phenomenon. Using an integration of time series theory (TST) for short term predictions and extreme value theory (EVT) for long term predictions, the authors have developed a model which can predict cyber attack rates 1 hour ahead of time with practical prediction accuracy. In [17], a novel approach is presented to detect outliers in high throughput numerical time series without assuming any distribution for the input data and without manual thresholds setting. The authors introduce two methods, SPOT for streaming data with stationary distribution and DSPOT for streaming data with

some drift. Peaks over threshold (POT) approach and maximum likelihood estimation have been used by the authors in [17] to estimate the required parameters for generalized pareto distribution (GPD). The authors show few example applications of their anomaly detection algorithm in intrusion detection, magnetic field measurements and stock prices. In [20], a method of thread assignment in multi-core/multi-threaded processors using extreme value is presented. The authors use peak over threshold (POT) method to obtain the GPD for the thread assignment problem. The parameter estimation for the GPD is done using maximum likelihood estimation. The authors use a method called sample pruning, which reduces the time for statistical analysis and introduces a methodology called sample pruning POT (SP-POT) which is claimed to reduce the analysis time by 8 folds. The authors declare that the introduced thread assignment performs close to optimal.

In applications where real time data analytics is required, hardware acceleration using FPGAs has been the better choice [21]. FPGAs provide massive performance with its parallel resources than CPUs and GPUs and consume a fraction of power of a CPU or a GPU [22]. Due to this attractive property of FPGAs, they are often favored to be utilized for implementing computationally intensive statistical signal processing algorithms. The author of [23] presents a histogram based probability density function estimation using FPGAs. Cumulative distribution function is computed in real time for the input data and important information like centiles which are used in quality of service oriented decisions in communication systems, are calculated from the probability density function. The estimator is built using access pattern memory and priority encoders to reduce latency and increase the performance. The author claims that the proposed architecture uses minimal amount of hardware resources and area in the FPGA chip. In [24], FPGA based bandwidth selection for kernel density estimation is presented. An algorithm called plug-in is used for the estimation of univariate kernel density estimation. The authors have used different architectural optimizations using the parallelism in FPGA to speed up the calculation. They have implemented a faster version of division operation using multiplications and divisions. For exponent and logarithmic calculations, CORDIC algorithm has been used. The authors claim an average speed up by about 32 than a CPU can be achieved with huge power efficiency. The authors have used HLS for the FPGA implementation and suggest more performance for their algorithm can be achieved if it is implemented in HDL.

To the best of my knowledge, the problem of wireless communication channel load stress analysis and forecasting using FPGAs hasn't been investigated in the literature. The methodology proposed in this thesis uses FPGAs at the edge of the wireless communication network for this purpose. Necessary data processing and forecasting is done using extreme value theory and centile estimates. The proposed architecture is highly suitable for streaming type of data. To the best of my knowledge, this is the first time a study on network channel load stress forecasting using FPGAs and extreme value theory has been done.

3 THEORY

3.1 Physical layer channel utilization: background

Wireless channel utilization is a measure of how much of the available air time is utilized by a wireless network and is typically expressed as a percentage. Channel utilization is a key metric as it can be used to assess the health of a wireless network at physical layer. For example, in *Wi-Fi* networks, channel utilization can be used to assess the impact on the user experience relating to various applications running on various mobile devices and laptop computers. If the channel utilization measured is at a higher value ranging from 80% to 90%, the user experience can be affected. While typically, mean values are used, in this thesis, we consider block maxima approach based on extreme value theory. We utilize real channel utilization data collected in [25] to obtain the block maxima series to forecast the channel load stress using extreme value theory.

Collected channel utilization data was measured as per the Algorithm 3.1 which can be used to calculate both the mean and maximum values of channel utilization data. In the algorithm, for a moving average window size of W, the authors in [25] compute the squared IQ values. Let the squared IQ values, $I^2 + Q^2$ at time instance t be $S_{m,t}$. They compare the squared IQ values, $S_{m,t}$ with a threshold which is computed as a scaled value of the noise floor given by $N_F \times C_s$ where N_F denotes the noise floor estimation and C_s denotes the scaling factor. An in-depth discussion on noise floor estimation is given in [25]. They count the number of times the threshold is surpassed by $S_{m,t}$ for all the samples in the moving average window. Let the average value be $S_{a,t}$ which is the fraction of times, the threshold is surpassed by $S_{m,t}$ for the window, W. Then, for $R_a > 0$ samples, the authors in [25] compute the average value and the maximum value of $S_{a,t}$ which are given out as the mean and max values of the channel utilization data. The complete procedure for the computation of channel utilization is given in Algorithm 3.1.

Algorithm 3.1 Channel Utilization Calculation [25]

Input: Moving average window size, $W > 0$, noise floor N_F, scaling factor C_s, IQ samples, comparison flag C_a
For W, compute N_F and then $I_{th} \leftarrow N_F \times C_s$
for each IQ sample in W **do**
    Compute $S_{m,t} \leftarrow I^2 + Q^2$
    if $S_{m,t} \geq I_{th}$ **then**
        $C_{a,i} \leftarrow 1$
    else
        $C_{a,i} \leftarrow 0$
    end if
end for
Obtain moving average at every t: $S_{a,t} = \frac{\sum_{i=t-W}^{t} C_{a,i}}{W}$
For $R_a > 0$ samples, compute $CU_{mean} = \frac{\sum_{1}^{R_a} S_{a,t}}{R_a}$ and
$CU_{max} = max\{C_{a,t}, C_{a,t+1}, \ldots, C_{a,t+R_a}\}$
Output: CU_{max} and CU_{mean}

3.2 Why only mean is not enough for the characterization of channel utilization data?

Examining collected data from [25] tells us about the behavior of wireless communication channel utilization data. We identify that they have a noise like appearance with spikes which occur occasionally. These spikes or bursts occur due to the aggregation of the utilization of the wireless channel by users in a short time interval. Due to this nature of the channel utilization, the distribution of CU data is no longer normal. They can result in heavily tailed distributions with large deviations from the mean or the median of the CU data. Therefore, statistics like mean and standard deviation can no longer be used to predict the probabilities of the channel utilization. Due to this reason, a forecast method based on mean and standard deviation is not a viable solution. As CU data behaves with extreme values due to the bursty nature, the theory of extreme value can effectively be applied in the prediction and forecasting of wireless channel utilization.

3.3 Extreme value theory

In this section, a review of the classical extreme value theory is given leading to a model that can be used to study the behavior of the series of the maximum values of wireless channel utilization data. Furthermore, application of the extreme value theory on the forecasting of channel load stress and its implications are discussed.

Let $X_1, X_2, \ldots, X_n$ be a sequence of random variables which have a common distribution function F. In practice, measurements of a certain process on a regular time-scale are usually represented by X_i. Then, M_n given in Equation (1) represents the maximum value of n observations of the process over n time units. In extreme value theory terms, this is also known as block maximum. For example, if n is the number of observations of channel utilization data over 1 second, then M_n represents the maximum.

$$M_n = max\{X_1, X_2, \ldots, X_n\} \tag{1}$$

The extremal types theorem in classical extreme value theory states that there exist sequences of constants $a_n > 0$ and b_n such that [26]

$$Pr\left\{\frac{M_n - b_n}{a_n} \leq z\right\} \rightarrow G(z) \quad \text{as } n \rightarrow \infty \tag{2}$$

where G is a non-degenerate distribution function. Then extreme value theory states that G should belong to one of the families, **Gumbel**, **Frechét** or **Weibull** distributions which are also called **type I**, **type II** and **type III** families of distributions respectively [25]. These distributions model different forms of tail behavior for the distribution function F of X_i. For example, in Gumbel distribution, the density of G decays exponentially while for Fréchet distribution, it decays polynomially which correspond to different rates of decay in the tail of F relative to each other. These three families are reformulated to a single family of models having the distribution functions of the form given by [26],

$$G(z) = exp\left\{-\left[1 + \xi\left(\frac{z-\mu}{\sigma}\right)\right]^{-1/\xi}\right\} \tag{3}$$

which is defined on the set $\{z : 1 + \xi(z-\mu)/\sigma > 0\}$. Parameters μ, σ and ξ satisfy $-\infty < \mu < \infty$, $\sigma > 0$ and $-\infty < \xi < \infty$ respectively. This is called **generalized extreme**

value family of distributions. μ, σ and ξ are called location parameter, scale parameter and shape parameter respectively. Fréchet and Weibull families correspond to the cases $\xi > 0$ and $\xi < 0$ and $\xi = 0$ corresponds to Gumbel family.

Generalized extreme value distribution which is formed by the unification of the original three distribution families greatly simplifies the statistical implementation. The tail behavior can easily be determined by the inference on ξ and so that there is no need to assume individual extreme value family for a given set of data.

Estimation of the parameters μ, σ and ξ can be done using different techniques. Graphical techniques based on versions of probability plots, moment-based techniques, procedures based on order statistics and likelihood-based techniques are generally used for this purpose [26]. Due to the all-round utility and adaptability to complex model-building, log likelihood method is widely accepted in robust parameter estimation. Assuming independence in $Z_1, Z_2, \ldots, Z_m$ having GEV distribution, the log likelihood function for the estimation of GEV parameters for the case $\xi \neq 0$ is given by [26],

$$l(\mu, \sigma, \xi) = -m\ log\ \sigma - (1 + 1/\xi) \sum_{i=1}^{m} log \left[1 + \xi \left(\frac{z_i - \mu}{\sigma}\right)\right] - \sum_{i=1}^{m} \left[1 + \xi \left(\frac{z_i - \mu}{\sigma}\right)\right]^{-1/\xi} \quad (4)$$

provided that $1 + \xi \left(\frac{z_i - \mu}{\sigma}\right) > 0,$ for $i = 1, \ldots, m$ hold.

For the case $\xi = 0$, the likelihood function in Equation (5) can be obtained using the Gumbel family of distribution [26].

$$l(\mu, \sigma) = -m\ log\ \sigma - \sum_{i=1}^{m} \left(\frac{z_i - \mu}{\sigma}\right) - \sum_{i=1}^{m} exp\left\{-\left(\frac{z_i - \mu}{\sigma}\right)\right\} \quad (5)$$

Maximization of the Equations (4) & (5) with respect to the parameters μ, σ and ξ should be done using standard numerical optimization algorithms as there are no analytical solutions for the equations.

3.3.1 Quantile estimates based on extreme value theory

In general, the following approach is adopted for modeling a series of independent observations $X_1, X_2 \ldots, X_n$ of a particular process. First, the data are broken in to blocks of n observations for some large values of n which generates a series of block maxima, $M_{n,1}, \ldots, M_{n,m}$. Then the GEV is fitted to this block maxima series by estimating the parameters μ, σ and ξ using some aforementioned technique. The estimates of the extreme quantiles for the maxima series are obtained by the inversion of the Equation (3) which gives [26],

$$z_p = \begin{cases} \mu - \frac{\sigma}{\xi}[1 - \{-log(1-p)\}^{-\xi}], & \text{for } \xi \neq 0, \\ \mu - \sigma\ log\{-log(1-p)\}, & \text{for } \xi = 0, \end{cases} \quad (6)$$

where $G(z_p) = 1 - p$.

For our problem of wireless channel load stress forecasting using GEV for fast processing in real time, we need to implement the GEV and quantile estimates in the FPGA. Steps of the implementation of GEV require that we estimate the GEV parameters, μ, σ and ξ using log likelihood function for the given set of data using numerical optimization methods and then obtaining the quantile estimates for the task of forecasting. Direct implementation of these functions requires lots of arithmetic operations to be executed and would take a considerable amount of CPU processing and time which is not suitable for the implementation of real time forecasting. For fast processing in real time, direct implementation of these equations are not appropriate and would incur a considerable latency in execution. Hence, it would result in a performance degradation and the actual benefit of the implementation in FPGA would be lost completely. Therefore, we need to find a simpler method for the implementation of GEV with some compromise between the accuracy and performance. Next section discusses the approach we adopted for the implementation of GEV with improved performance which suits the underlying hardware architecture in the FPGA.

3.4 Derivation of a simpler version of the generalized extreme value theory

Due to the complexity in implementation in parameter estimation of GEV for wireless channel utilization data, we utilize the histogram which can readily be computed easily. Histogram is considered to be an accurate representation of the numerical data distribution. It is an approximation to the probability density function (pdf), $p_X(x)$ of a random variable X. Probability density function is essentially a normalized version of the histogram with the assumption of infinite number of data samples and the bin width chosen to be infinitesimally small. Hence, histogram can be used for the estimation of the probability density function for a given set of input data.

Let H denotes the histogram, I_i denotes the i^{th} bin interval of the histogram, π_i denotes the frequency of the data values which lie within the i^{th} interval I_i and M be the number of histogram bins. Then, the interval I_i can be written as,

$$I_i = [bin\_edge_i, bin\_edge_{i+1}) \tag{7}$$

and the histogram can be denoted as,

$$H = \{(I_1, \pi_1), (I_2, \pi_2), \ldots, (I_M, \pi_M)\} \tag{8}$$

Figure 2 shows the normalized histogram generated for the maximum values of the channel utilization data and the overlay of the GEV distribution function computed from *MATLAB* on the same graph. We can clearly see that the normalized histogram closely approximates the GEV returned from *MATLAB*. Therefore, using histogram for the estimation of GEV can be justified.

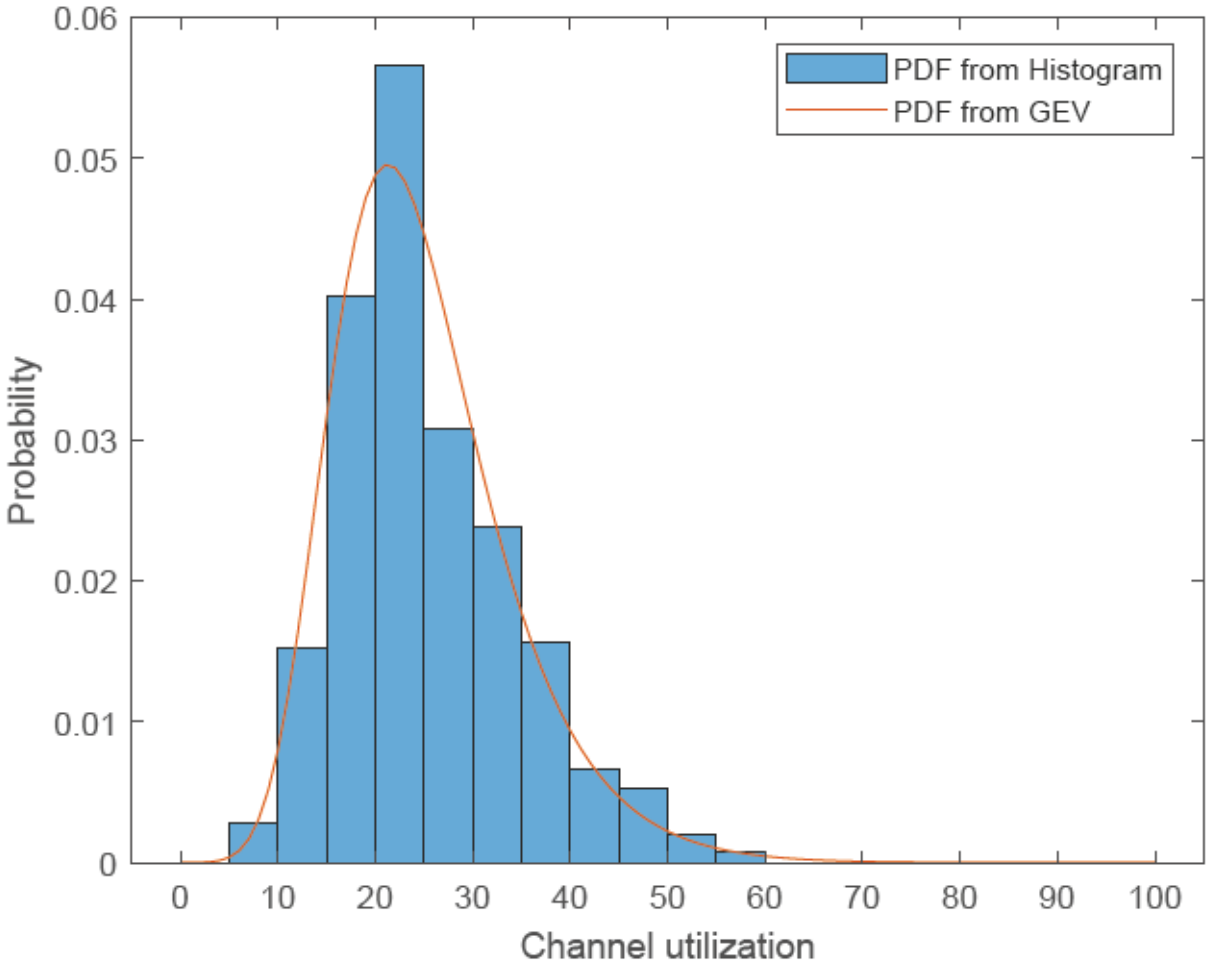

Figure 2. Histogram vs the GEV for series maxima of channel utilization data

From histogram, we obtain the distribution of cumulative sum which is used for the quantile estimation. Let C denotes the distribution of the cumulative sum for the input data samples. For i^{th} bin interval, we calculate,

$$\psi_i = \sum_{k=1}^{i} \pi_k \tag{9}$$

and obtain the distribution of cumulative sum as,

$$C = \{(I_1, \psi_1), (I_2, \psi_2), \ldots, (I_M, \psi_M)\} \tag{10}$$

For the quantile estimation, we need to find the the estimate, $p = P(X > x)$. As we are dealing with cumulative sum values, we compute $P = p \times N$ where N denotes the total number of data samples in the histogram. We find the interval in the distribution of cumulative sum where P resides. Let this interval be $[I_i, I_{i+1})$, then we can write,

$$P \, \epsilon \, [\psi_i, \psi_{i+1}) \tag{11}$$

Then, using the linear interpolation as illustrated in Figure 3, we find the quantile estimate value, $Q \, \epsilon \, [I_i, I_{i+1})$ which is given by,

$$Q = I_i + (P - \psi_i)\,\frac{(I_{i+1} - I_i)}{(\psi_{i+1} - \psi_i)} \tag{12}$$

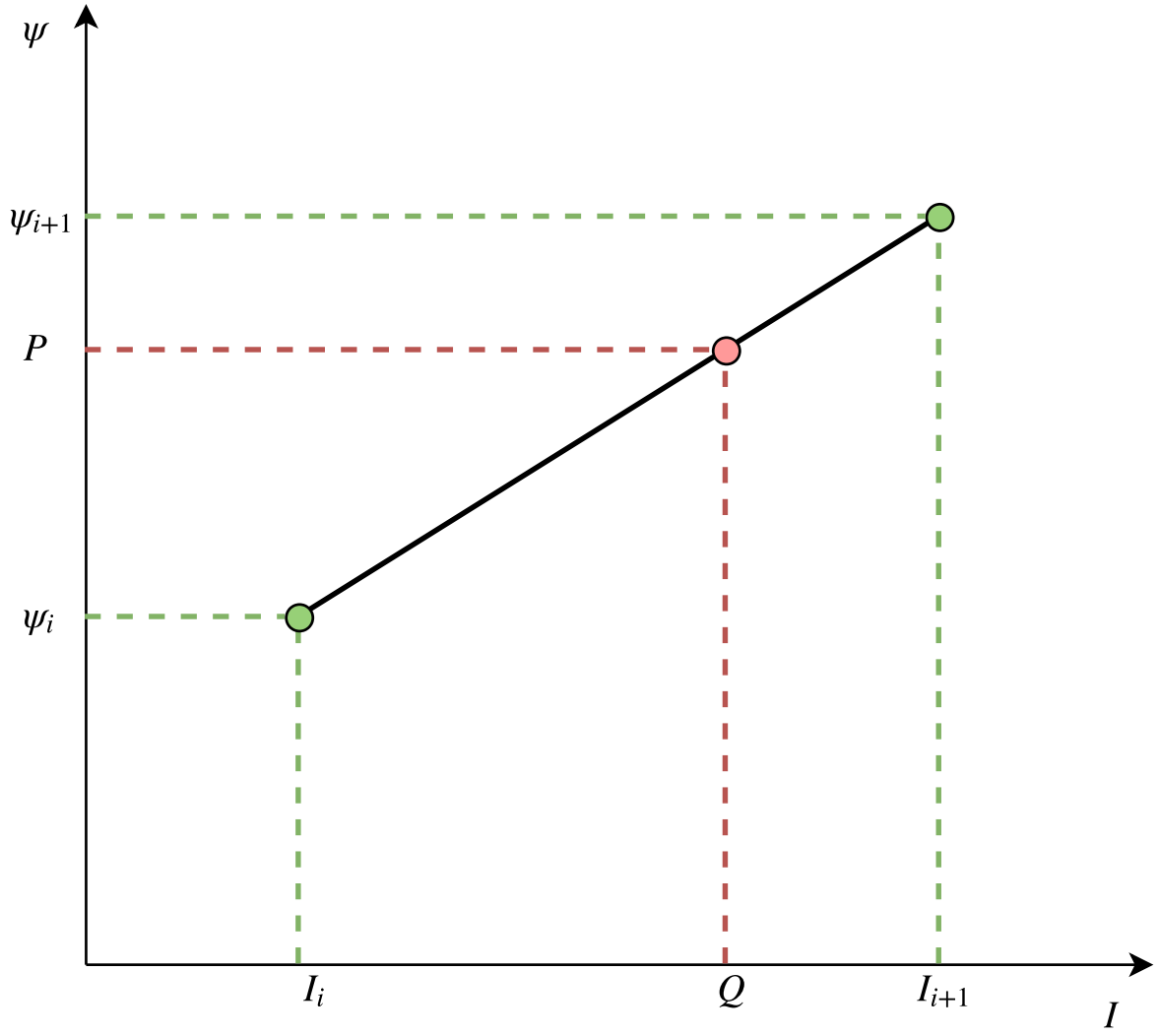

Figure 3. Linear interpolation for quantile estimation

Due to the reduced complexity of the proposed method, it is simpler and easy to implement. Opposed to the direct implementation of GEV theory, there is less arithmetic operations required and the method can readily be utilized in streaming type of data. Thus, the proposed method is very well suited for the hardware architecture in the FPGA. Real-time histogram generation and cumulative sum calculation for the streaming data can easily be implemented in FPGA. The quantile estimation method which is used for the wireless channel load stress forecasting can also be implemented with less latency. The downside of the proposed method can be the loss of accuracy for implementation simplicity. However, we could see that the degradation in the performance of the proposed method is not much compared to the direct GEV implementation. In a later chapter, a thorough analysis of the performance of the both of proposed method and the direct GEV implementation are given.

4 IMPLEMENTATION

This chapter describes the implementation of the proposed channel utilization load stress analysis method. The chapter thoroughly explains the designed algorithms relating to various processing modules. It also explains the steps involved in the implementation of each processing module on a *ZedBoard*. It is important to note that the implementation is challenging due to the streaming nature of the channel utilization data coming to the *ZedBoard*. The final intention of this implementation is to deploy the developed modules at the edge of wireless communication networks for channel load stress forecasting purposes. Due to this reason, the design is implemented in a way that the implemented modules are divided between the FPGA and the processor part of the *ZedBoard*. For example, modules which require low latency are implemented on the FPGA part of the *ZedBoard*. Hence, our implementation suits the streaming nature of the channel utilization data samples. Several key design techniques are adopted in the hardware implementation to meet the required latency requirement.

4.1 System overview

Figure 4 depicts a high-level overview of the complete system architecture of our implementation. The implementation is done using a *Xilinx Zynq-7000 all programmable SoC* in *ZedBoard* [27]. The board contains FPGA fabric, embedded arm processor, and most of the necessary interfaces/supporting functions for our requirement. Due to the simplicity and flexibility of HLS, necessary functional modules are designed and developed and corresponding IP cores are generated using the tool, *Xilinx Vivado HLS* [28]. *Xilinx Vivado* is used for the complete system integration and bit stream generation for hardware programming. *Xilinx Vivado SDK* is used for the bare-metal (standalone) [29] application development.

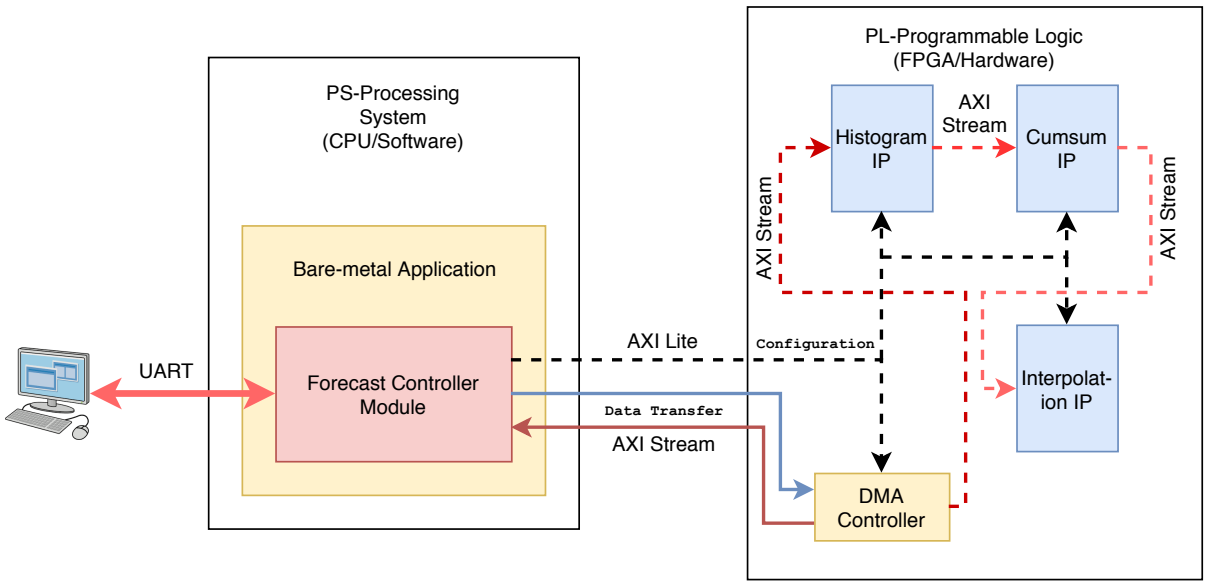

Figure 4. System block diagram

ZedBoard with *Xilinx Zynq-7000* SoC contains 1) processing system (PS) which features a dual core *ARM Cortex A9* processor and 2) programmable logic (PL). All the other necessary peripherals like on-chip memory, external memory interfaces and I/O (Input/Output)

peripherals and interfaces are included in the SoC [30]. *Zynq-7000* offers flexibility and scalability of an FPGA while delivering performance, low power consumption and ease of use typically associated with ASICs (application specific integrated circuit). Due to it's flexibility in customization, PL can be configured according to the design requirement. PS of the SoC manages the PL part and the communication among them. The resulted outputs are obtained on a laptop computer connected via UART interface to the *ZedBoard*.

Figure 4 shows all the important functional blocks in the implemented system. The forecast system controller module resides in the PS and manages the data exchange between 1) the computer and PS 2) the PS and the PL. Data exchange between the computer and PS is done through *universal asynchronous receiver-transmitter (UART)*. All the other functional blocks related to the forecast system reside in the PL. The system uses *Advanced eXtensible Interface* (*AXI*) bus specification [31] to exchange data among functional modules, and between the PS and the PL. More detailed description of the whole system and inter communication is given in a later section.

4.2 Steps involved in implementation

Key steps involved in the implementation of the proposed system are shown in Figure 5. Design flow for the implementation task starts with the algorithm design. In this phase, several approaches for the problem are discovered and each are modeled and simulated for the required functionality using *MATLAB* mathematical computing software. *MATLAB* is used in this phase due to its flexibility and easiness in programming and the availability of a rich set of libraries which speeds up the initial prototyping of the algorithms. The selected algorithms are evaluated based on their performance, accuracy and complexity. The algorithm which gives a compromise between the performance and the complexity are selected for the implementation. Next, the development of the C models starts. Initially, floating point models are developed and simulated. Then, the associated fixed point models are developed and simulated and the impact on the algorithms' performance due to finite fixed point word lengths are evaluated. Necessary optimizations are made for word lengths without degrading the algorithm performance drastically. Once completed the development of fixed point models, design of HLS modules take place and the corresponding IP cores are generated. One of the key decision to be taken is on how the communication between the IPs should take place. Due to the low latency requirement of our application, the communication method should have a minimal effect on latency and it should be robust and accurate. HLS allows for different kinds of interfacing on the developed HLS modules. Depending on the requirement of the application, necessary interfacing methods are selected and added to the HLS modules in this phase. Final integration of the modules and bit stream generation is done using *Xilinx Vivado*. At last, the bare metal application development, which runs on the *ARM* processor in PS is developed using *Xilinx SDK*. Then the bit stream is downloaded to the *ZedBoard* development board and the bare-metal application is loaded in to the memory which marks the final step in the design procedure. For the proposed system, three IPs, histogram IP, cumulative sum IP and interpolation IP are implemented in PL and the bare-metal application which run on the *ARM* processor and manages the data exchange among the PS, PL and the computer is developed accordingly.

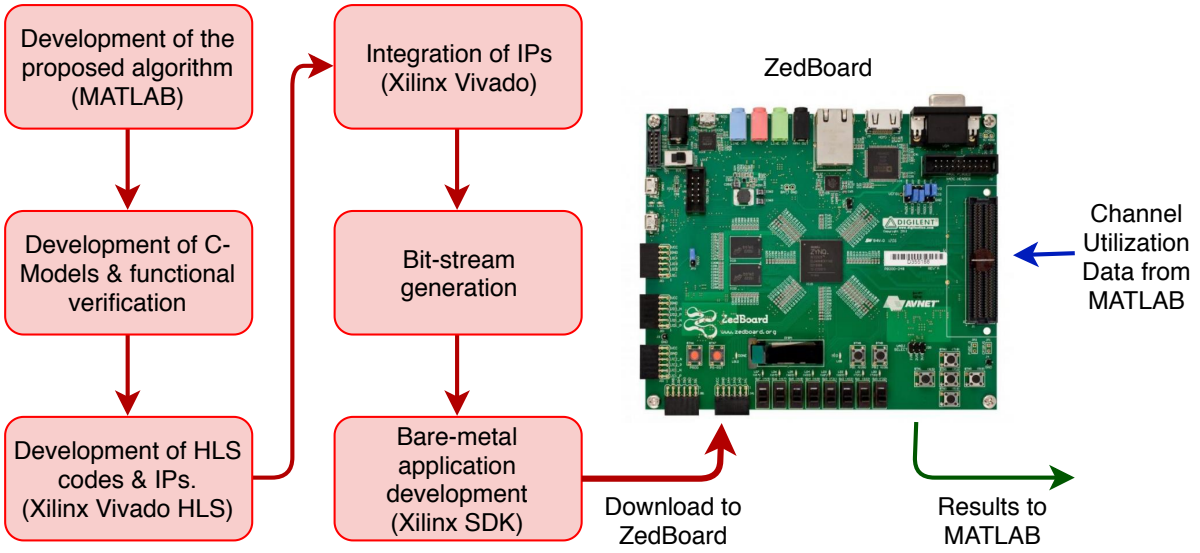

Figure 5. Steps involved in the implementation of the proposed system on a *ZedBoard*

4.3 Algorithm implementation

The PL contains three functional modules (IPs), namely, *makehist* IP, *cumsum* IP and *invcum* IP (Interpolation IP). These IPs work in conjunction with the PS to evaluate the percentile estimate for a given channel utilization probability. The IPs are arranged in a pipeline such that the processed data from one IP travels to the next IP in succession. The algorithms implemented in each IP and their detailed functionality are discussed in next sub sections.

4.3.1 Histogram and cumulative sum IP modules

Algorithm (4.1) is used for the histogram calculation for the input channel utilization (CU) data. As the CU data is streaming type, there is only single CU data value available at a certain time instance, i. Let D_i denote the CU data at time instance i, BIN_EDGE_j & BIN_EDGE_{j+1} be the left and right bin edges at j^{th} bin interval, $HISTOGRAM\_BIN_j$ be the histogram bin counter value at j and W be the CU data window for which the histogram is computed. The CU data value D_i is compared against the histogram bin interval, $[BIN\_EDGE_j\ ,\ BIN\_EDGE_{j+1})$. If D_i is inside the considered bin interval, $HISTOGRAM\_BIN_j$ counter at location j is incremented by one and next CU data sample is taken for comparison. If D_i is not inside the j^{th} bin interval, it is compared with the next bin interval $[BIN\_EDGE_{j+1}, BIN\_EDGE_{j+2})$. This comparison is carried out until the CU data sample matches the corresponding bin interval in the histogram. This process is continued for every CU data sample in the considered data window. Once all the CU data in the data window is processed, the computed histogram is given at the output. The IP in which this algorithm has been implemented is *Makehist* IP and the timing and latency information for the IP are given in Table 1 and Table 2 respectively for a sample data window size of 64.

Algorithm (4.2) is used for the generation of cumulative sum. Let $CUMSUM_i$ represents the cumulative value at i^{th} location of the corresponding bin in the histogram. Cumulative value at i^{th} location is given by the sum of all the past histogram values up to the i^{th} bin. The

Algorithm 4.1 Histogram Algorithm

Input: Data samples size of window W
for each data sample D_i in W **do**
  for each j^{th} bin in HISTOGRAM **do**
    if $BIN\_EDGE_j \leq D_i < BIN\_EDGE_{j+1}$ **then**
      $HISTOGRAM\_BIN_j \leftarrow HISTOGRAM\_BIN_j + 1$;
      break;
    end if
  end for
end for
Output: $HISTOGRAM$

algorithm calculates the sum over all the past histogram bin values for the i^{th} value and this process is repeated for all the histogram bins. Cumulative sum is given as output at the end of this operation. This algorithm has been implemented in *Cumsum* IP and its timing and latency information are given in Table 1 and Table 2 respectively.

Algorithm 4.2 Cumulative Sum Algorithm

Input: HISTOGRAM
$CUMSUM \leftarrow HISTOGRAM$
$i \leftarrow 0$;
for each i^{th} histogram bin **do**
  $CUMSUM_{i+1} \leftarrow CUMSUM_{i+1} + CUMSUM_i$
end for
Output: $CUMSUM$

4.3.2 Percentile estimation IP

Algorithm (4.3) shows the steps in estimating the percentile for a given probability. As the cumulative sum is derived for the channel utilization data instead of the probability density, it is required to compute corresponding value P which is in the range of cumulative sum values. P is calculated by multiplying p with the data window size, $DATA\_WINDOW\_SIZE$ for which the histogram was calculated earlier. Then, P is compared against each cumulative value to find the interval i where P is residing. A line is fitted to the two cumulative sum data points, $CUMSUM_i$ & $CUMSUM_{i+1}$ which are located to the left and right of P. The corresponding value, X to P in the domain of the line is calculated by evaluating the inverse equation of the fitted line. X value thus calculated is given out as the percentile estimate for the corresponding probability value. The algorithm (4.3) is implemented in *Invcum* IP and its timing and latency information are summarized in Table 1 and Table 2 respectively.

Algorithm 4.3 Percentile Estimation Algorithm

1: **Input:** $CUMSUM, Probability\ p$
2: $P \leftarrow DATA\_WINDOW\_SIZE * p$
3: Find the interval, $(CUMSUM_i \leq P < CUMSUM_{i+1})$ where P resides in
4: Fit a line which goes through $CUMSUM_i$ & $CUMSUM_{i+1}$
5: Find the value X corresponding to the value P in the domain of the fitted line
6: **Output:** Percentile estimate, X

Table 1. Timing summary of generated IPs

IP	**Estimated Clock (ns)**
Makehist	4.806
Cumsum	4.321
Invcum	8.510

Table 2. Latency information of generated IPs

IP	**Latency (clock cycles)**	
	Min	***Max***
Makehist	36	288
Cumsum	35	35
Invcum	102	102

4.3.3 Usage of HLS for IP generation

Xilinx Vivado HLS tool was primarily used for the design and implementation. In HLS, *C* language description of functionality is converted to hardware description language which is used by the tool for hardware synthesis. Due to the simplicity of HLS, a designer doesn't need to have thorough knowledge neither in hardware description languages such as verilog, vhdl nor hardware architecture design. But still the designer should pay some attention on how the functionality is matched to the hardware and not to use un-synthesizeable C functions and constructs when designing the IP cores. Appendix I shows an example HLS code developed for an implemented IP using *Vivado HLS*.

Generally, *Xilinx Vivado HLS* tool's default behavior is to synthesize the C codes to use minimum amount of resources and area with increased latency. There are tool specific macros and directives available in HLS which are applied in the form of #pragma macros or #directives. These directives or pragmas are very important for guiding the tool in optimizing the micro architecture of the design which is resulted by the tool synthesis process. For example, a *for* loop with *N* iterations would be implemented with a latency of *N* cycles or more by default by the tool. If we need to reduce the latency of the *for* loop, we need to use tool directive, *UNROLL* [28] to implement the *for* loop with a single clock cycle by completely unrolling the loop or reducing the latency with some factor by partially unrolling the loop. The variables used in loops should also be partitioned completely or partially depending on the unrolling factor of the loop. In this way, the designer should use the tool directives to optimize the micro architecture to meet the required design specification.

4.4 IP integration and system generation

Figure 6 illustrates the complete system block diagram once the integration is done. It also shows the communication protocols which are used between different subsystems. Mainly, *Advanced eXtensible Interface (AXI)* protocol [32] is used for the control and configuration of the IPs and other modules which are residing in the PL. *Xilinx* adopted *AXI* protocol is called *AXI Lite*. *AXI* protocol is targeted for high performance, high frequency system designs and provides a high bandwidth and a low latency.

4.4.1 AXI protocol

AXI is a burst based protocol. Nature of the data to be transferred are described by address and control information on the address channel. Data transfer between the master and slave happens through the write data channel to the slave or read data channel to the master [32]. Figure 7 shows the data read transaction which happens between a master and a slave. Figure 8 shows the data write transaction which happens between a master and a slave. In write transactions, the completion of the write transaction is signaled by the slave to the master through an additional write response channel which can be identified in the same figure.

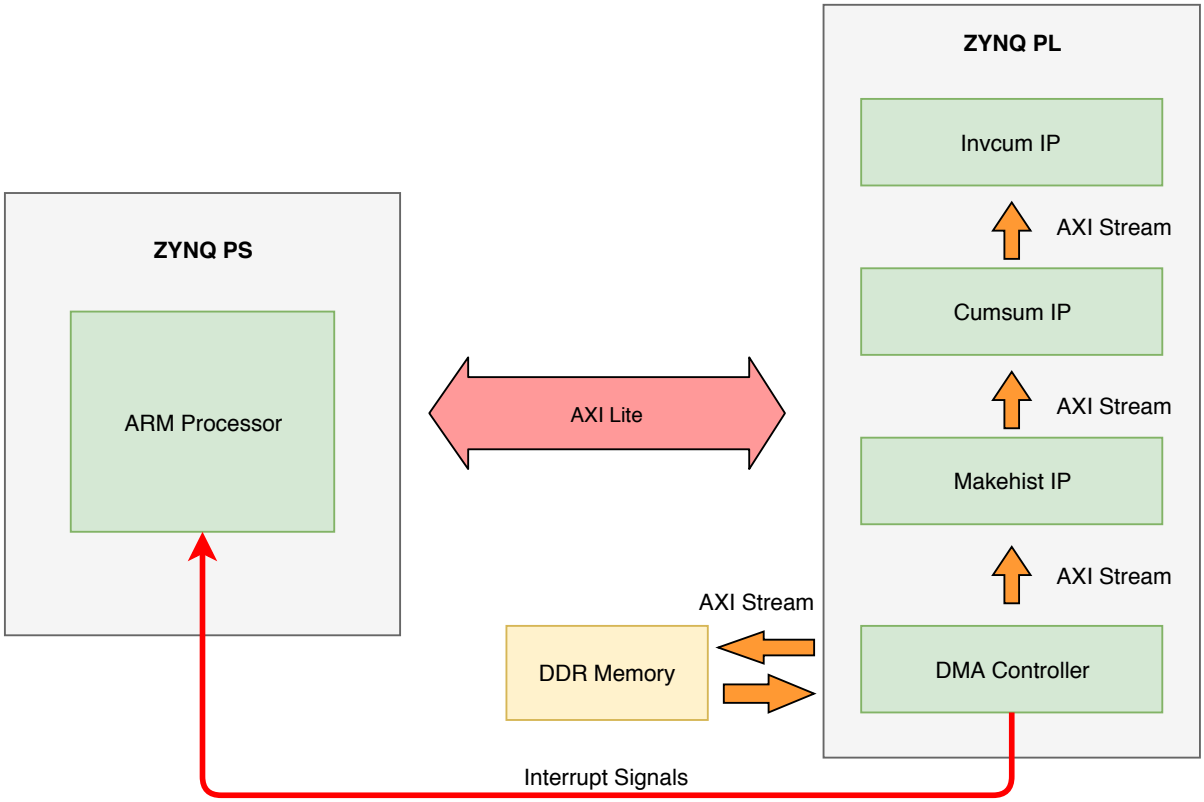

Figure 6. Complete System Block Diagram

AXI Lite protocol

AXI Lite protocol which is a derivation of *AXI* protocol adopted by *Xilinx* is used to control the IPs in the hardware design. Basically, *AXI* Lite interface in an IP device is used for device initialization, device status querying and controlling, reading and writing to the registers available in the device and setting up, monitoring and controlling interrupts in the device [28]. ARM processor in the PS uses *AXI Lite* protocol to control the IPs in the proposed design. Basically, IPs are initialized through *AXI Lite* protocol. Data window size which is required for the histogram generation are written to the respective register in the *Makehist* IP through the *AXI Lite* interface. Probability for the percentile estimation is also set in *Invcum* IP and the result is read back through the *AXI Lite* interface available in the IP.

AXI Stream protocol

We use AXI Stream protocol for high speed streaming data transfers. *AXI Stream* protocol is used in applications where the focus is on a data-centric and data-flow paradigm where the concept of address is not important [31]. *AXI Stream* behaves as a single unidirectional channel with a handshaking data flow. Due to this property, the mechanism to move data between IPs are efficient and fast. *AXI Stream* protocol is optimized for high performance data flow applications. Therefore, to fulfill low latency of our implementation design, the data transfer from the memory (DDR memory) to the respective IP (*makehist* IP) is implemented using the *AXI Stream* protocol. Direct memory access controller (DMA controller) takes care of the data movement from the system memory (DDR memory) to the *makehist* IP independently

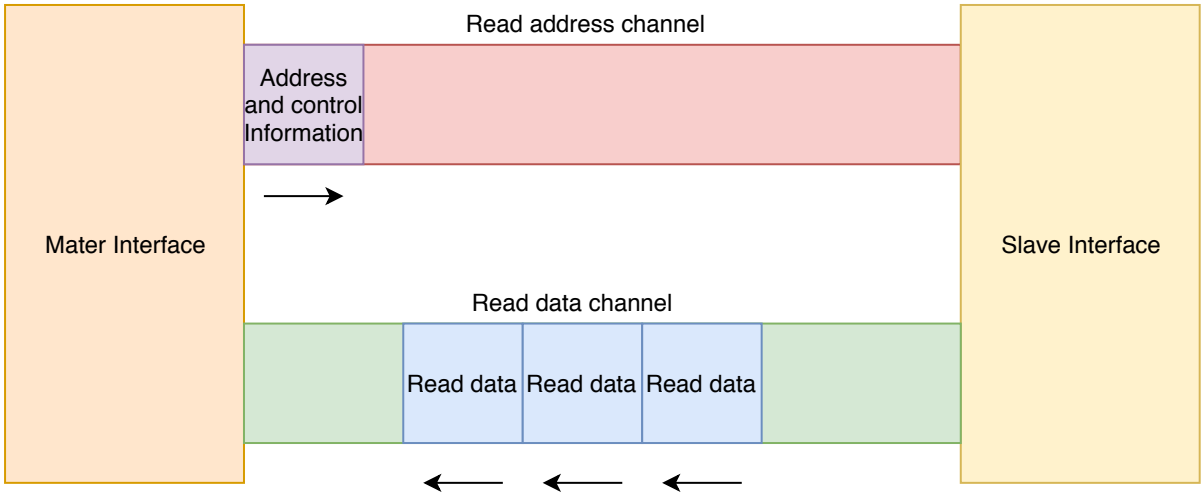

Figure 7. *AXI* protocol read architecture

with negligible amount of intervention from the *ARM* processor [33]. Due to this reason, the overhead on the processor on data transfer process is kept at minimal. Same protocol is used for the data transfers between the IPs also. Figure 6 depicts the locations where *AXI Stream* protocol is used for data transfer operations. Once the data is streamed to the makehist IP, the data flows through all the three IPs sequentially as a stream. Each IP processes the data as per the algorithms described in section 4.3 and the processed data is forwarded to the next IP for further processing.

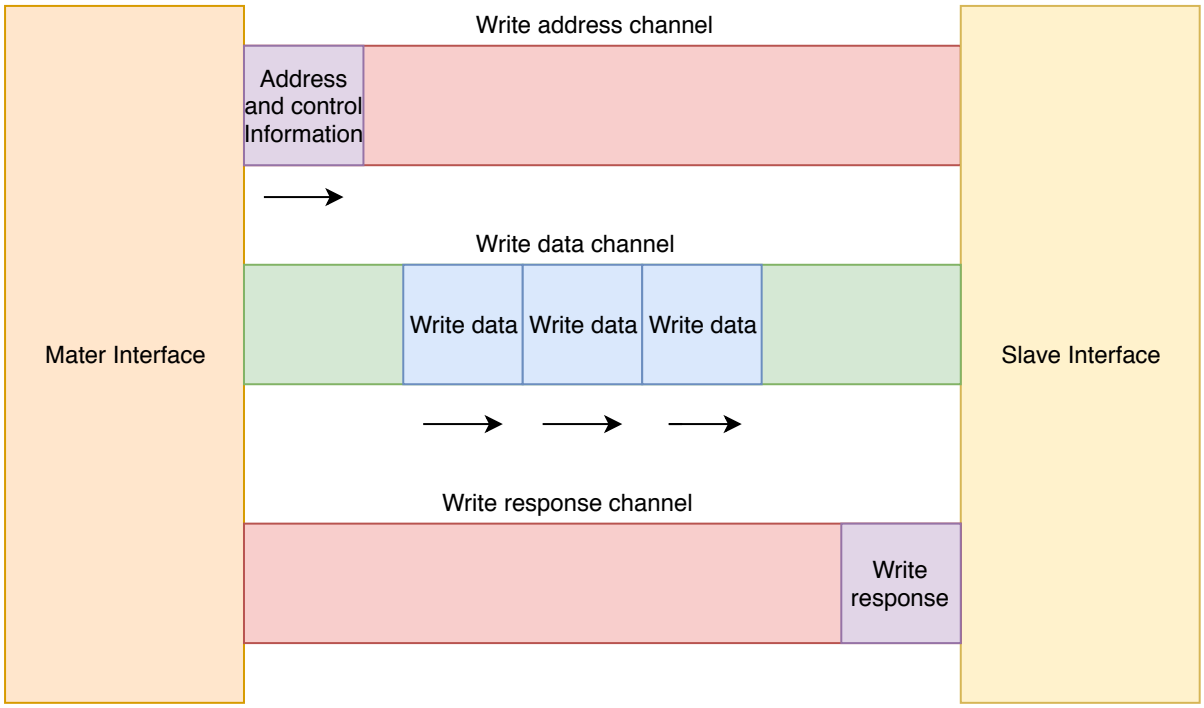

Figure 8. *AXI* protocol write architecture

4.4.2 Overall design flow of the implemented system

Figure 9 shows block design implemented in *Xilinx Vivado* by integrating IPs. It shows all the IPs with system interconnects and other peripheral IPs which are needed for the integration purposes. Interrupts of DMA controller are used to detect the completion of write data transaction between the system memory and the *makehist* IP. DMA controller is connected to accelerator coherency port (ACP port) of the *ARM* processor. Data streaming from system memory to the streaming device (*makehist* IP) takes place through the memory mapped to streaming (MM2S) channel. Data streaming from streaming device to the system memory takes place through the streaming to memory mapped (S2MM) channel. At the end of each transaction, DMA asserts an interrupt to notify the *ARM* processor, the completion of the data transaction. Therefore, the DMA interrupts are connected to the interrupt controller in the *ARM* processor. This interrupt information facilitates the *ARM* processor in reading the results and scheduling the next data transaction.

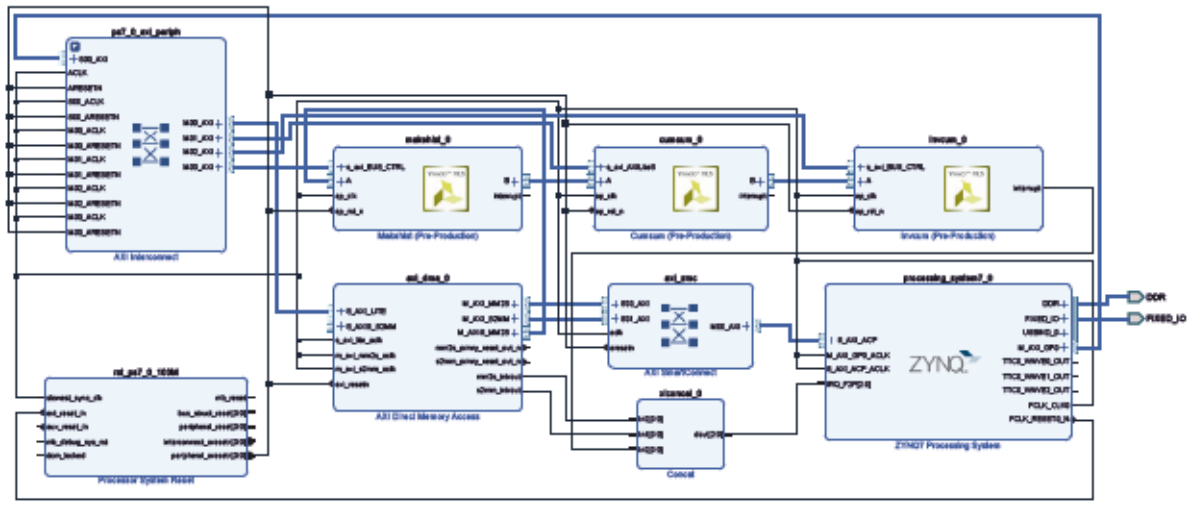

Figure 9. Integrated system in *Xilinx Vivado*

The system is synthesized and implemented in *Xilinx Vivado* after the block design. Then programming bitstream is generated for FPGA configuration. For the firmware development for *ARM* processor, we need the implemented hardware in *Xilinx Vivado*. It is exported to *Xilinx SDK* where bare-metal (standalone) application development takes place. ZYNQ SoC is configured through *Xilinx SDK* and the developed bare-metal application is executed in the *ARM* processor. *Xilinx SDK* can be used in conjunction with *Xilinx Vivado* for signal tracing and debugging, through integrated logic analyzer (ILA) if needed to ease the debugging process. This concludes the hardware implementation phase of the proposed system.

5 RESULTS AND DISCUSSION

In this chapter, we present the results relating to performance evaluation of the proposed algorithm which is based on extreme value theory and is implemented on a *ZedBoard*. We evaluate the performance of the implemented algorithm by comparing the results with *MATLAB* based simulations. Evaluation was done based on the algorithmic design. In our testing, ZedBoard was connected to a computer with *MATLAB* using serial port and the results were obtained by sending channel utilization data to the *ZedBoard* and reading back the computed output data from *ZedBoard*. Channel utilization data used for the evaluation purpose was actual data collected from the university's *Wi-Fi* network. The details regarding how the data was collected can be found in [].

5.1 Behavior of channel utilization data

In statistical analysis of data, one of the preliminary steps is to get a sense of how the data is distributed. Generally in statistics based forecasting methods, a certain probability density function is assumed for the input data set and then respective parameters in the probability density function are estimated using the input data. This estimated probability density function is used in statistical methods like inferencing and forecasting. Without having any information on the data distribution, prior assumption of a probability distribution for the data often leads to strange and erroneous results. To avoid this problem, techniques like data visualization should be adopted. One of the best tools in data visualization is histogram which gives a coarse estimation for the probability distribution function of underlying data. Therefore, for our data set of maximum values (block maxima series) of wireless channel utilization, we constructed the histogram to visualize the behavior of the data set which is shown in Figure 10. From Figure 10, it is clear that the data doesn't behave according to a normal distribution. The data distribution is skewed to the left (negative skewness) closely mimicking a GEV distribution.

When approximating a probability distribution function for a set of data, the other important aspect is the granularity of the data. If the histogram is built using a lesser number of data, the approximation to the pdf also becomes less accurate and some of the important features in the data distribution which we are interested in might get averaged out. If the histogram is built using a greater number of data, the approximation to the pdf becomes more accurate, but then the resulting distribution might not be smooth due to the outliers in the data. Therefore, selection of the granularity of the data is a key aspect for better performance of our proposed algorithm. To select the best granularity, we visually inspect the histograms of random contiguous blocks of samples from the data set to arrive on the decision on the correct block size. Figures 11 and 12 show the distribution of the max values of CU data for a period of 20min and 1 hour respectively. Corresponding GEV pdf are also fitted for the same data set using *MATLAB*. It is clear from the figures that the best approximation to the GEV pdf is given by the data set collected for 1 hour. Therefore, we selected the time period of 1 hour for the proposed percentile estimation algorithm.

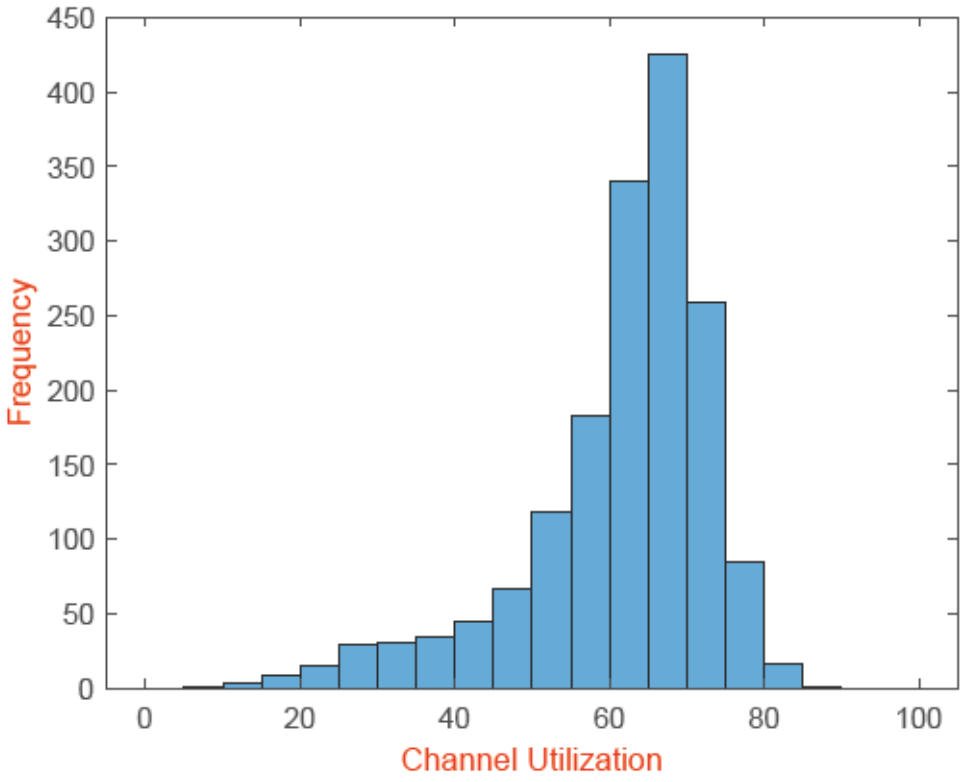

Figure 10. Generated histogram for the block maxima series of channel utilization data

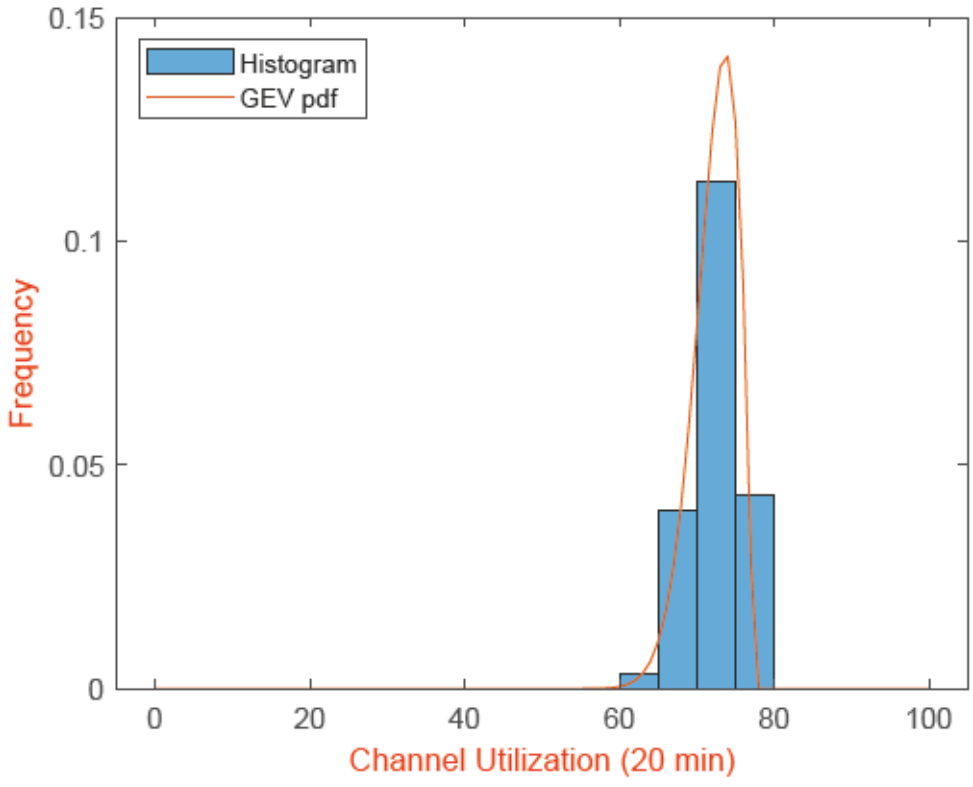

Figure 11. Generated histogram for 20min of max values of CU data

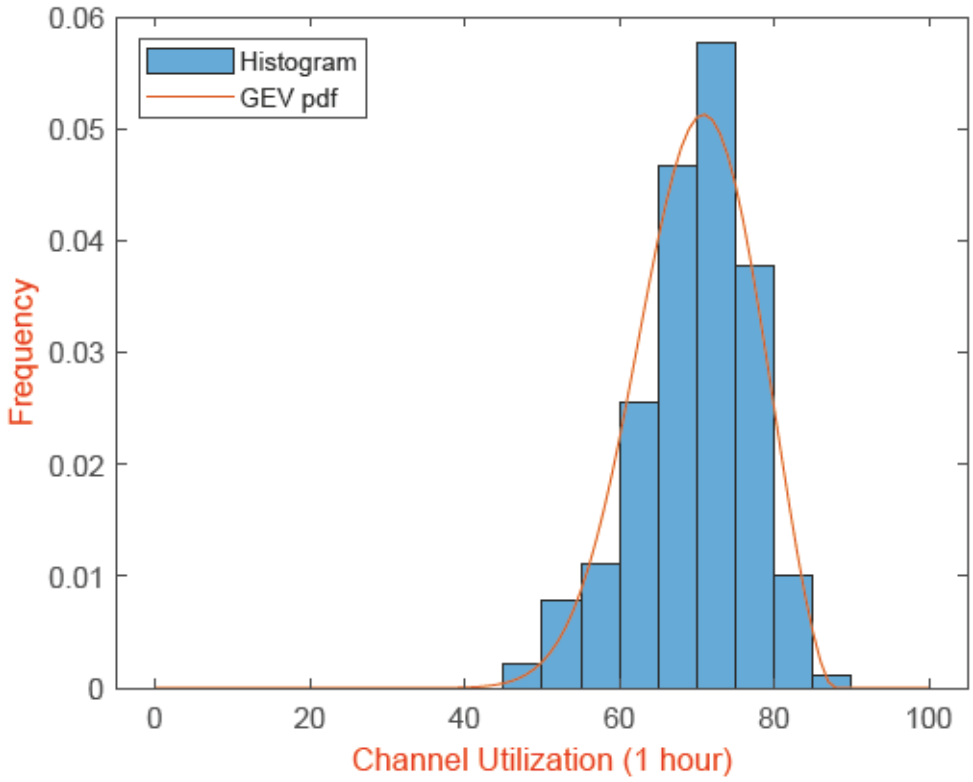

Figure 12. Generated histogram for 1hour of max values of CU data

5.2 Evaluation of the proposed algorithm

In this section, the performance of the proposed algorithm is evaluated against the GEV of *MATLAB* implementation. The block maxima series of channel utilization data used for the testing has been collected with a rate of 3 samples per minute for a period of 9 hours from university's *Wi-Fi* network. As described in Section 5.1, max values of CU data for the evaluation were partitioned to 1 hour data blocks on which the percentile estimates are made using the proposed algorithm and compared against the results from the *MATLAB* implementation of GEV.

Table 3. Error Statistics for 1 hour of CU data

Mean	Standard Deviation
-0.469	0.891

For a data set of 1 hour, Figure 13 shows the percentile estimates given by the proposed method and the GEV implementation of *MATLAB* for probabilities from 0.01 to 0.99. It is

clear from the figure that through out the probability range from $0.01 - 0.99$, the estimated percentiles from the proposed method closely follow the percentile estimates calculated using GEV implementation of *MATLAB*. There are some deviations visible which are due to the approximation errors of histogram for the exact pdf. Generally, errors are significant around the probability values in the region close to 0 and close to 1. This is mostly due to the effect of finite bin width of the histogram. Figure 14 shows the error plot between the *MATLAB* GEV and the proposed method. We can observe that through out the probability range, the variation of the error approximately resides in the interval $[-2, 2.5]$. Table 3 shows some of the important statistics related to the error between the two methods.

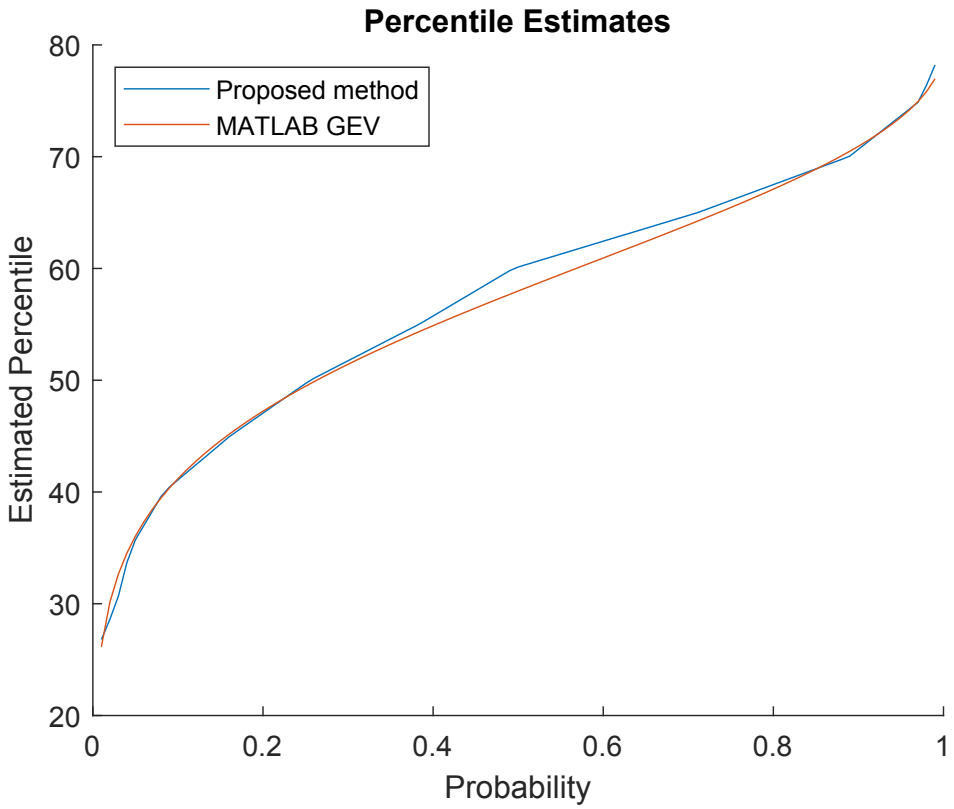

Figure 13. Estimated percentiles from the proposed method and *MATLAB* GEV method

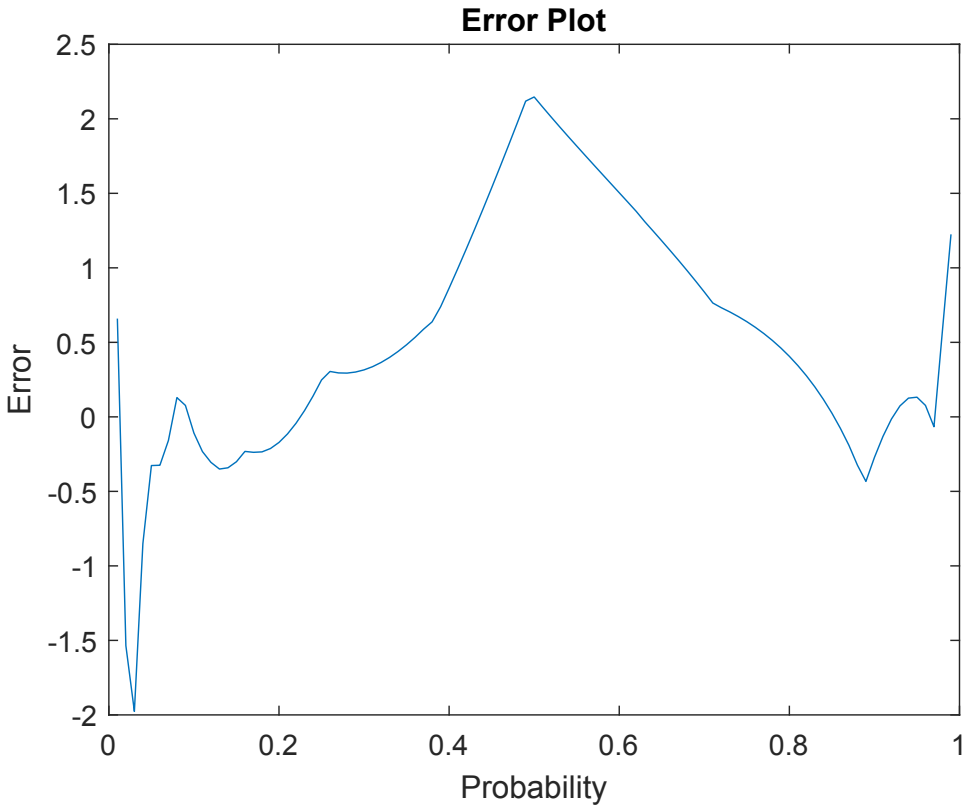

Figure 14. Error in the percentile estimates between the proposed method and *MATLAB* GEV method

To evaluate the accuracy of percentile estimation of the proposed method, we plotted the estimated percentiles of the proposed method against the percentile estimation of *MATLAB* GEV. Figure 15 shows resulted correlation plot between the two methods. We also fitted a regression model to the data set and the respective 95% confidence interval on the same figure. It is visible from the figure that most of the data points fall inside the 95% confidence interval. Therefore, we can be 95% confident that the mean of the future observations would fall inside the confidence interval. This implies that the estimation performance of the proposed method would stay almost invariant through out random samples of the data set. Table 4 shows the correlation coefficient between the estimated percentiles from the proposed method and *MATLAB* GEV. Assuming a significance level (denoted as α) of 0.005, we receive a $P - value$ equal to 0.000 which tells us whether the correlation coefficient is significantly different from 0. With $P - value \leq \alpha$, we can presume that the calculated correlation coefficient is significant. Therefore, we can conclude that the performance of the proposed method is closely related to the original GEV method.

Table 4. Correlation coefficient for 1 hour of CU data

Correlation Coefficient	P-value
0.995	0.000

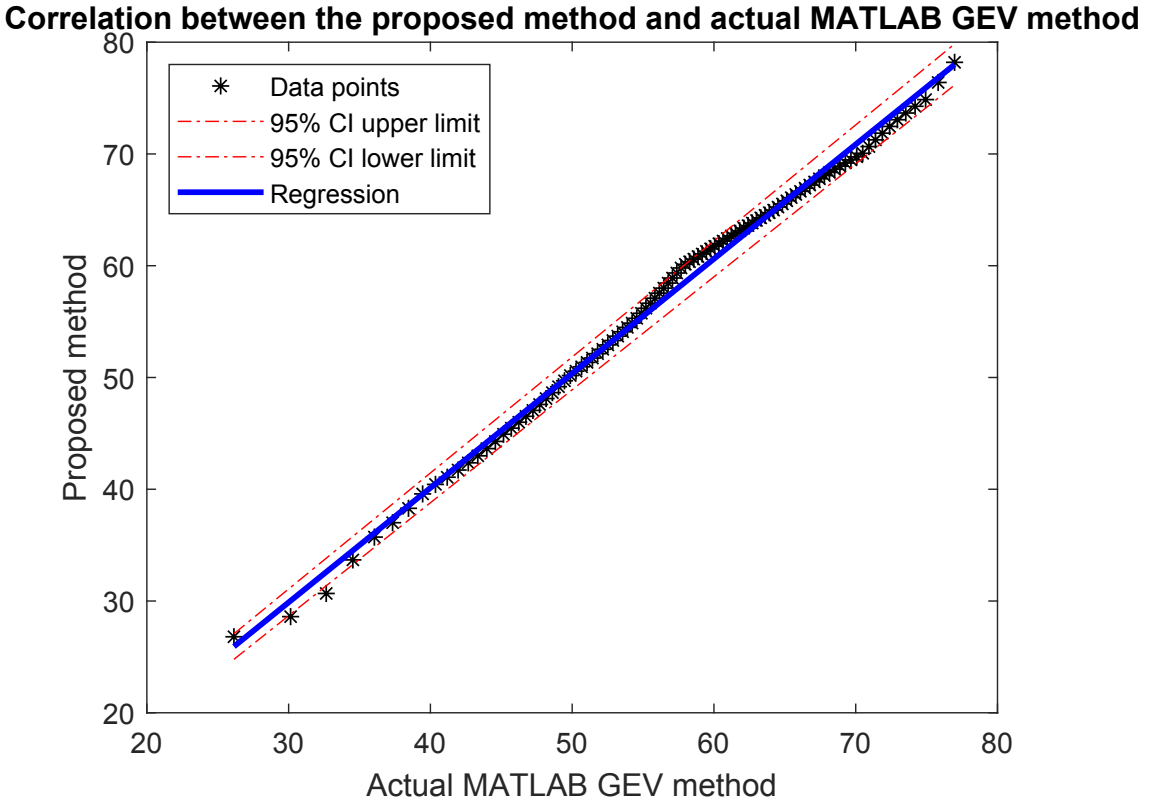

Figure 15. Correlation between the proposed method and *MATLAB* GEV method

To study the behavior of error in percentile estimation, we calculated the difference between the results from the proposed method and *MATLAB* GEV method. Using the statistical software *Minitab*, we obtained the tolerance interval plot. Tolerance interval is an important measure which gives the range that is likely to accommodate a specified proportion of the population. Confidence level for the tolerance interval gives the likelihood that the interval would cover the specified proportion. Therefore, we can use the tolerance interval for the error to predict the future values of error with a specified confidence level.

Figure 16 depicts the tolerance interval plot for the calculated error between the proposed method and *MATLAB* GEV method. It also shows the normality test for the error. The calculated p_value for the normality test is lower than 0.005. Therefore, we should reject the null hypothesis and come to the conclusion that the error doesn't follow a normal distribution. Consequently, we can use the non parametric tolerance interval for the error. The lower and

upper bounds of the tolerance interval for the error are given as -2.425 and 1.852 respectively. Therefore, with a confidence level of 95%, we can expect that future errors which would be generated from the proposed method would fall inside this tolerance interval.

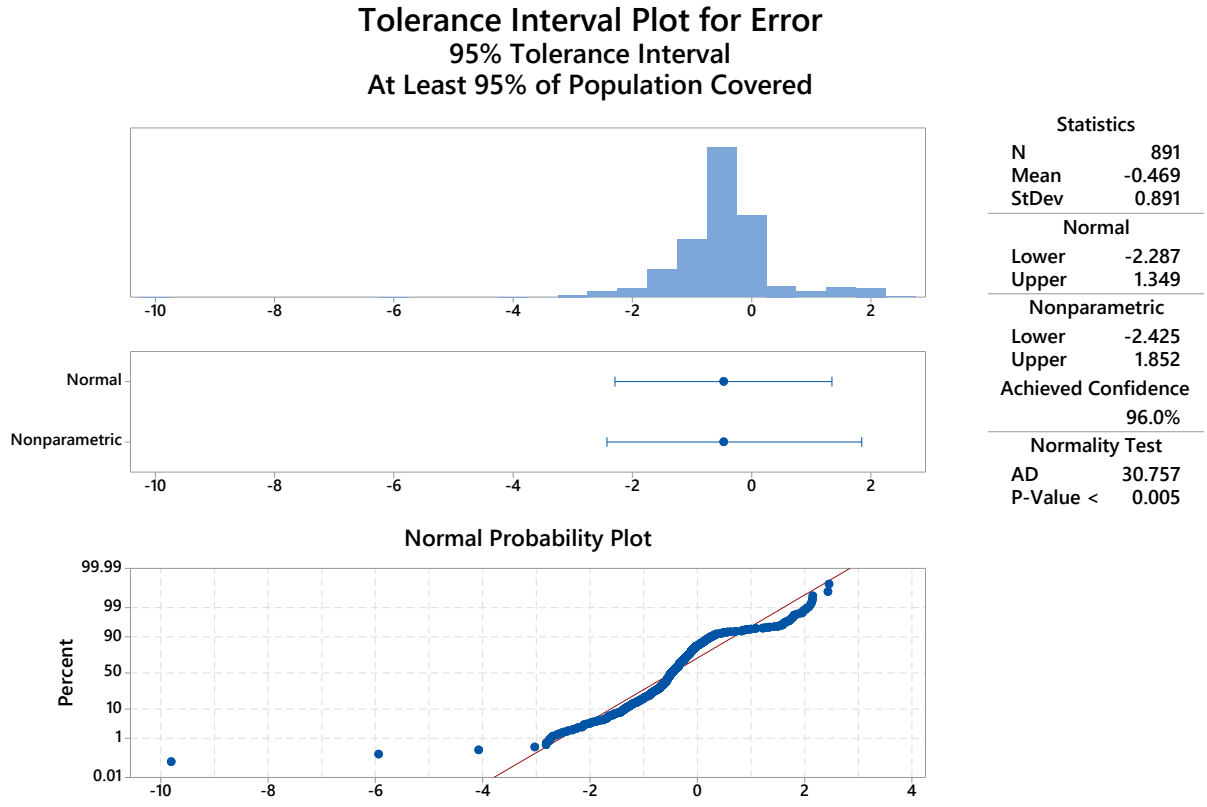

Figure 16. Tolerance interval plot for the error between the *MATLAB* GEV and the proposed method

Figure 17 shows the estimated percentiles of the proposed method and *MATLAB* GEV over time for a probability of 0.8. It is clear from the figure that the proposed method closely follows the behavior of *MATLAB* GEV in different samples. There are some deviations in the estimated values of the proposed method from *MATLAB* GEV. But, according to earlier observations in the behavior of the error, we know with a confidence level of 95% that the error would reside in the tolerance interval, $[-2.425, 1.852]$. Therefore, it can be concluded that the accuracy of the results of the proposed method is higher at an acceptable level.

Despite the simplicity, our evaluation of the performance of the proposed algorithm gives positive results. The percentile estimates given by the proposed algorithm have very high accuracy and have very little deviations from the estimates of actual GEV. With the

performance gain and reduced implementation complexity, small deviations from the expected results are tolerable. Therefore, with a considerable confidence, we can rely on the percentile estimates generated by the proposed method to apply for the load stress analysis of wireless communication channels.

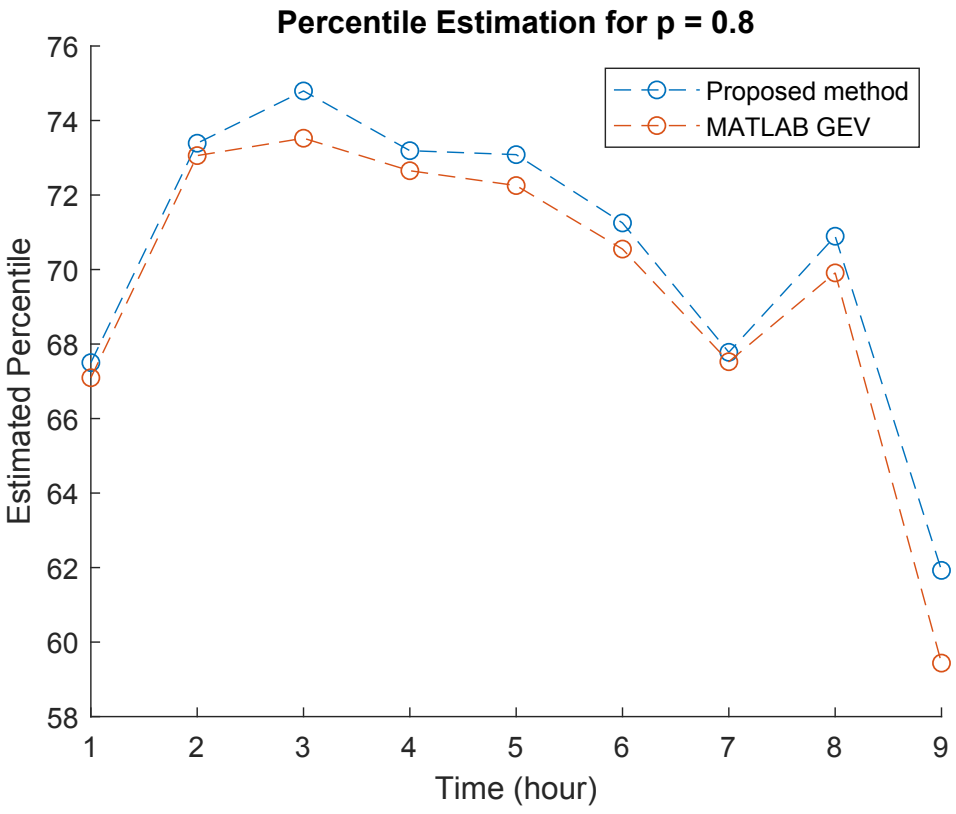

Figure 17. Percentile estimation for $p = 0.8$ across time for proposed method and *MATLAB* GEV method

6 CONCLUSION

One of the key usage scenarios of fifth generation (5G) and beyond networks is to provide mission critical, ultra-reliable and low latency communications (URLLC) targeting applications where latency and reliability of wireless links are extremely important. Some of the applications where URLLC links with extreme reliability and low latency required are industry automation, robotics enabled medical solutions and intelligent autonomous transportation. With incredible growth of connectivity and traffic volume, provisioning of the latency and reliability requirements for URLLC services is becoming challenging. Due to simultaneous access of applications by users requiring low latency and high data rates, wireless communication channels can experience increased resource utilization, such as increased wireless channel utilization when increased beyond certain point can degrade the performance of URLLC applications. These kind of problems can be avoided by carrying out real-time radio frequency data analytics at the edge of the wireless network to allow for proactive resource allocation combined with some forecasting method which can assign more radio resources when a particular resource is forecasted to be under stress. There are some network controllers providing auto-scaling and load balancing mechanisms on network resources based on some forecasting and prediction methods. But, until now, most of the forecasting and prediction methods have relied on the data collected over 3 to 6 months or no less than 24 hours.

Real-time RF data analytics require that hundreds of millions of streaming samples can be processed within a second and therefore, hardware acceleration using FPGAs can be considered more appropriate. In this thesis, we proposed a real-time data analytics based resource stress forecasting method using an FPGA. The proposed method has less complexity and can perform the data analytics in real time. The proposed method is based on quantile estimation and can be used on different probability distributions related to a variety of resource utilization scenarios. As the interest of this research is on optimizing the resource allocation for URLLC applications, we focused on forecasting the stress in wireless channel utilization. Due to the bursty nature of channel utilization data in stressed conditions, we used the block maxima series of channel utilization data which can be modeled using the generalized extreme value theory. The proposed method is implemented using *Xilinx Zynq-7000* series *SoC* board using *Vivado*, *Vivado HLS* and *Xilinx SDK* along with *MATLAB*. We thoroughly evaluated the performance of the proposed method against the results obtained from the theoretical method using generalized extreme value theory tool in *MATLAB*. The comparison of the results against each other reveals that the proposed algorithm performs almost equal to the theoretical implementation of GEV and the results are within a very small margin of error. The proposed method can be used in streaming channel utilization data and can be used with high throughput applications requiring very low latency. Therefore, the implemented device can easily be utilized at the edge of wireless communication networks to perform forecasting of wireless channel load stress in real-time with high accuracy.

One of the extensions of this work we envision for, is to implement forecasting for wireless channel utilization based on the direct implementation of the generalized extreme value theory in the *ARM* processor of the *Xilinx Zynq-7000 SoC*. The possibility of implementing the direct GEV should be carefully analyzed and the respective mathematical operations need to be optimized to perform the computation with minimal latency. The performance should be evaluated to comprehend whether the direct implementation of GEV can be used for real-time applications. If so, further gain in the accuracy should be possible in the results.

7 ACKNOWLEDGMENT

We gratefully acknowledge *Xilinx* University Program (XUP) for providing us a donation of *Zedboard* product which was used in this work for implementing the wireless channel load stress analytic module.

8 REFERENCES

[1] 5G Americas (2018), 5G Americas spectrum recommendations for the U.S. URL: http://www.5gamericas.org/files/5815/2364/7029/5G_Americas_Spectrum_Recommendations_for_the_U.S_Final.pdf, [Online] Accessed : 06/07/2019.

[2] *3GPP* (2017), 5G: Study on scenarios and requirements for next generation access technologies (3GPP TR 38.913 version 14.2.0 Release 14). URL: https://www.etsi.org/deliver/etsi_tr/138900_138999/138913/14.02.00_60/tr_138913v140200p.pdf, [Online] Accessed : 06/08/2019.

[3] Francesco P.D., Malandrino F. & DaSilva L.A. (2018) Assembling and using a cellular dataset for mobile network analysis and planning. IEEE Transactions on Big Data vol. 4, pp. 614–620.

[4] Rodrigues F. & Pereira F.C. (2018), Beyond expectation: Deep joint mean and quantile regression for spatio-temporal problems. URL: https://arxiv.org/pdf/1808.08798.pdf, [Online] Accessed : 06/07/2019.

[5] Oracle (2015) Improving Communications Service Provider Performance with Big Data: Architect's Guide and Reference Architecture Introduction. Oracle Corporation. URL: http://www.oracle.com/us/technologies/big-data/big-data-communications-2528130.pdf, [Online] Accessed : 06/07/2019.

[6] Restuccia F. & Melodia T. (2019), Big data goes small: Real-time spectrum-driven embedded wireless networking through deep learning in the RF loop. URL: https://arxiv.org/pdf/1903.05460.pdf, [Online] Accessed : 06/07/2019.

[7] Riverbed (2019), Riverbed® OPNET NetOne™. URL: https://www.riverbed.com/document/fpo/media-cms/Riverbed_OPNET_NetOne.pdf, Datasheet, Accesed : 06/04/2019.

[8] Cisco (2015), MATE Design User Guide. URL: https://www.cisco.com/c/dam/en/us/td/docs/net_mgmt/wae/6-1/design/user/guide/MATE_Design_User_Guide.pdf, Chapter 7 : Forecasting Traffic, Accessed : 06/04/2019.

[9] Mariño P.P. (2018), Net2Plan 0.6.0 User's Manual. URL: http://www.net2plan.com/documentation/current/help/usersGuide.pdf, Chapter 2, Accessed: 06/05/2019.

[10] Baldo N., Giupponi L. & Mangues-Bafalluy J. (2014) Big data empowered self organized networks. In: European Wireless 2014; 20th European Wireless Conference, pp. 1–8.

[11] Hadi M.S., Lawey A.Q., El-Gorashi T.E. & Elmirghani J.M. (2018) Big data analytics for wireless and wired network design: A survey. Computer Networks vol. 132, pp. 180 – 199. URL: http://www.sciencedirect.com/science/article/pii/S1389128618300239.

[12] Barmpounakis S., Kaloxylos A., Spapis P., Magdalinos P., Alonistioti N. & Zhou C. (2018) Data analytics for 5G networks: A complete framework for network access selection and traffic steering URL: http://www.iariajournals.org/telecommunications/tele_v11_n34_2018_paged.pdf, international Journal on Advances in Telecommunications, issn 1942-2601 vol. 11, no. 3 & 4, year 2018.

[13] *3GPP* (2018), 5G; system architecture for the 5G system (3GPP TS 23.501 version 15.2.0 Release 15). URL: https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/15.02.00_60/ts_123501v150200p.pdf, [Online] Accessed : 06/08/2019.

[14] Kaloxylos A. (2018) Application of data mining in the 5G network architecture. In: ICDT 2018: The Thirteenth International Conference on Digital Telecommunications, ISBN: 978-1-61208-623-1.

[15] Lòpez L.I.B., Vidal J.M. & Villalba L.J.G. (2017) An approach to data analysis in 5G networks. Entropy 2017 vol. 19(2):74. URL: https://doi.org/10.3390/e19020074, [Online] Accessed : 06/08/2019.

[16] Zhan Z., Xu M. & Xu S. (2015) Predicting cyber attack rates with extreme values. IEEE Transactions on Information Forensics and Security vol. 10, pp. 1666–1677.

[17] Siffer A., Fouque P.A., Termier A. & Largouet C. (2017) Anomaly detection in streams with extreme value theory. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '17, ACM, New York, NY, USA, pp. 1067–1075. URL: http://doi.acm.org/10.1145/3097983.3098144.

[18] Deane J.H.B., Johnstone G.G., Ledford A.W. & Underhill M.J. (1997) Extreme value theory applied to multichannel communication systems. Electronics Letters vol. 33, pp. 832–833.

[19] Shuangqing Wei, Goeckel D.L. & Kelly P.E. (2002) A modern extreme value theory approach to calculating the distribution of the peak-to-average power ratio in ofdm systems. In: 2002 IEEE International Conference on Communications. Conference Proceedings. ICC 2002 (Cat. No.02CH37333), vol. 3, vol. 3, pp. 1686–1690.

[20] Radojković P., Carpenter P.M., Moretó M., Čakarević V., Verdú J., Pajuelo A., Cazorla F.J., Nemirovsky M. & Valero M. (2016) Thread assignment in multicore/multithreaded processors: A statistical approach. IEEE Transactions on Computers vol. 65, pp. 256–269.

[21] Intel (2019), FPGA inline acceleration for streaming analytics. URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01278-fpga-inline-acceleration-for-streaming-analytics.pdf, Whitepaper [Online] Accessed : 06/11/2019.

[22] Berten (2016), GPU vs FPGA performance comparison. URL: http://www.bertendsp.com/pdf/whitepaper/BWP001_GPU_vs_FPGA_

Performance_Comparison_v1.0.pdf, BERTEN Digital Signal Processing, Whitepaper [Online] Accessed : 06/15/2019.

[23] Fahmy S.A. (2010) Histogram-based probability density function estimation on FPGAs. In: 2010 International Conference on Field-Programmable Technology, pp. 449–453.

[24] Gramacki A., Sawerwain M. & Gramacki J. (2016) FPGA-based bandwidth selection for kernel density estimation using high level synthesis approach. Bulletin of the Polish Academy of Sciences Technical Sciences vol. 64, pp. 821–829.

[25] Khan Z., Lehtomäki J.J., Hossain E., Latva-Aho M. & Marshall A. (2018) An FPGA-based implementation of a multifunction environment sensing device for shared access with rotating radars. IEEE Transactions on Instrumentation and Measurement vol. 67, pp. 2561–2578.

[26] Coles S. An Introduction to Statistical Modeling of Extreme Values. Springer Series in Statistics, Springer-Verlag London, 1st ed., 45-73 p. URL: https://www.springer.com/gp/book/9781852334598.

[27] *AVNET* (2014), ZedBoard: (Zynq evaluation and development) hardware user's guide. URL: http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf, [Online] Accessed : 06/08/2019.

[28] *Xilinx* (2018), UG902 (v2018.3): Vivado design suite user guide: High-level synthesis. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug902-vivado-high-level-synthesis.pdf, [Online] Accessed : 06/08/2019.

[29] *Xilinx* (2015), UG821 (v12.0): Zynq-7000 all programmable SoC software developers guide. URL: https://www.xilinx.com/support/documentation/user_guides/ug821-zynq-7000-swdev.pdf, [Online] Accessed : 06/08/2019.

[30] *Xilinx* (2018), DS190 (v1.11.1): Zynq-7000 SoC data sheet: Overview. URL: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf, [Online] Accessed : 06/08/2019.

[31] *Xilinx* (2011), UG761 (v13.1): AXI reference guide. URL: https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf, [Online] Accessed : 06/08/2019.

[32] *ARM* (2018), AMBA AXI protocol v1.0. URL: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0022b/index.html, [Online] Accessed : 06/08/2019.

[33] *Xilinx* (2018), PG021: LogiCORE IP product guide. URL: https://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf, [Online] Accessed : 06/08/2019.

9 APPENDICES

Appendix I Example HLS code for an implemented IP

```
#include "makehist.h"

void makehist(Taxistream &A, Taxistream &B, Tint *DataArraySize)
{
  Tint i=0;

  static Tint hist_local[HISTOGRAM_ARRAY_SIZE] = {0};

  assert(*DataArraySize < 64);
  INDEX_LOOP: for(i = 0; i < *DataArraySize; i = i + 1)
  {
    Taxi dataIn = A.read();
    compare_out(dataIn.data.to_uint(), hist_local);
  }

  COPY_OUTPUT: for(i = 0; i < HISTOGRAM_ARRAY_SIZE -1 ; i++)
  {
    Taxi dataOut;
    dataOut.data = hist_local[i];
    dataOut.keep = 0b1111;       // all high
    dataOut.strb = 0b1111;
    dataOut.last = 0;

    B.write(dataOut); // Write to axi stream output
    hist_local[i] = 0;  // Reset to zero
  }

  SET_TLAST: {
    Taxi dataOut;
    dataOut.data = 0;
    dataOut.keep = 0b1111;  // all high
    dataOut.strb = 0b1111;
    dataOut.last = 1;
    hist_local[i] = 0;
    B.write(dataOut);   // Write to axi stream output
  }
}

void compare_out(Tint value, Tint hist[HISTOGRAM_SIZE])
{
  Tint i = 0;

  COMPARE_LOOP: for(i = 0; i < HISTOGRAM_SIZE; i++)
  {
    if(value >= i * BIN_SIZE && value < (i+1) * BIN_SIZE)
    {
      hist[i] = hist[i] + 1;
      break;
    }
  }
}
```

Listing 9.1. C++ file for Makehist IP