Implementation of neurobiological and various signal processing algorithms using FPGA circuits

PhD Thesis

László Schäffer

Supervisors:
Szilveszter Pletl, PhD and Zoltán Kincses, PhD

Doctoral School of Computer Science
Department of Technical Informatics
Faculty of Science and Informatics
University of Szeged

Szeged
2021

Contents

Chapter 1

Introduction

Signal processing is an important part of computer science, which is used in but not limited to automation, pattern recognition, control theory, artificial intelligence, and networking and communication. A signal can be anything, which is measurable and everything that can be measured is also can and inevitably will be processed using analog or digital signal processing methods. It can be a temperature measurement, an image from a camera or a recording of neural brain activity, all of them have to be processed in some way. In signal processing the first step is to measure a physical quantity. After the measurement the quantity can be filtered and amplified. In ideal circumstances a noise-free and proper amplitude quantity can be converted to a digital signal using an analog-digital converter (ADC). This digital signal can be processed by a Central Processing Unit (CPU) / microprocessor, a microcontroller, a Graphics Processing Unit (GPU), a Field Programmable Gate Array (FPGA) or an Application-Specific Integrated Circuit (ASIC).

There are various hardware and also software solutions in signal processing. All hardware types have their role in different circumstances and environments, which depends on performance, reliability, power consumption and form-factor. Nowadays CPUs are widely used in personal computing, although they also widely used for data processing with high performance, but at a cost of high power consumption and better sequential processing than parallel. Microcontrollers as their name suggests are used for controlling automated devices, and can be found in almost every household appliances due to their small form-factor and low power consumption. GPUs are used for gaming in personal computing, but also widely used for data processing, mainly for neural networks and crypto mining, which are complex parallel computing demanding problems. Unfortunately GPUs are big, with high energy consumption and require an operating system with a CPU to function. On the other hand FPGAs and ASICs are small form-factor, low energy consumption devices with medium to high performance due to their parallel computing capabilities, and a CPU can be implemented on their surface too, but they are far behind in sequential computing

performance.

In performance demanding applications the usage of GPUs, FPGAs or ASICs are inevitable, due to their parallel computation capabilities. When low power consumption is required usually FPGAs or ASICs are used, but the prototyping step is always done on FPGAs, because of their true parallel nature and yet flexibility for reprogramming. FPGAs can be applied for all problems, but their real usefulness comes in those where the algorithm can be separated into independent side-by-side running tasks. FPGAs are programmable semiconductor devices, which consists of a matrix of programmable logic elements connected through programmable interconnections. The basic structure of a Xilinx FPGA is built up from Configurable Logic Blocks (CLBs), a Programmable Interconnection (PI) network, Block RAM memories (BRAMs), configurable I/O Blocks (IOBs), Digital Signal Processors (DSP) and Digital Clock Management (DCM) blocks. The CLBs are the main digital processing blocks, which are capable of performing combinational as well as sequential operations. For complex arithmetic operations DSP blocks can be used, while internal data storage can be done with BRAMs. FPGA architectures can contain embedded soft or hard core microprocessors (MicroBlaze, ARM processor cores), support numerous data communication protocols (USB, Ethernet, SPI, I2C, UART, etc.) and additional peripherals. It is no surprise then that FPGAs are already used in off-the-shelf neural recording systems (e.g. Intan), but usually in the measurement part due to the high bandwidth requirements with densely packed electrode arrays. There is a scientific progress for using FPGAs not only in measurement, but for neural signal processing at the same time, which opened a way for online or real-time spike sorting methods.

This work presents FPGA-based acceleration of signal processing algorithms ranging from neurophysiological signal detection, classification, synthesis through hardware-in-the-loop simulation to control theory. The relation between the presented solutions is that instead a solely software-based optimization a new parallel FPGA architecture has been developed, which makes these algorithms operate more efficiently and in real-time. Furthermore FPGA-based solutions can be used in otherwise inaccessible environments.

Changing the hardware in signal processing applications are a complex task and usually requires to modify an optimised software. In many cases only software optimisation offers limited improvements, although the time and energy investment is still significant. In these cases the usage of FPGA makes possible to rapidly change the hardware architecture along with the software.

In Chapter 2., an FPGA-based multi-channel neurophysiological action potential signal detection approach is presented along with an also multi-channel template-based neural signal pattern generator. The main idea is to take advantage of the parallel nature of the FPGA architecture to handle multiple electrode channels at the same time. A problem arises with the detection algorithm selection, which needs

to be simple, but efficient. Another problem is the validation of the proposed solution, which requires a multi-channel real-time neural measurement. To solve this a real-time neural signal generator architecture is proposed, which generates action potentials on multiple channels considering among other important factors the spatial information like the crosstalk and the amplitude decay between adjacent channels.

In Chapter 3., an FPGA-based architecture is presented for multi-channel neural signal classification. The problem is that a simple, but efficient spike classification algorithm is required. Most of the spike sorting algorithms works offline and have unfeasible requirements for real-time operation such as storing the entire measurement before the processing takes place. There are already online spike sorting solutions, which can be used, but only for single-channel processing. Therefore an already available template-based unsupervised online spike sorting algorithm is chosen to modify it to be capable of multi-channel processing. Furthermore using a closely packed electrode array allows to exploit the small distances between electrode channels and use the spatial information to make the classification more precise.

In Chapter 4., different signal processing algorithms are accelerated using FPGAs to make real-time usage available, moreover form-factor and power consumption are also taken into consideration. It is an all-around presentation of FPGA capabilities with rapid prototyping to prove that vastly different algorithms can be efficiently applied with FPGAs for real-time operation. Different fields of computer science are chosen such as control theory, hardware-in-the-loop simulation and pattern recognition.

These chapters present different signal processing FPGA applications, however similarities can be noticed. FPGA-based acceleration links them together, which leads to more efficient and real-time solutions. Also from sequential operation a more parallel approach can be seen for which a change in the current algorithm development perspective is required. Furthermore in each chapter is a comparison between the regular and the proposed solution, which emphasises the benefits of FPGA-based systems.

1.1 Contributions

The ideas, figures, tables and results included in this thesis were published in scientific papers (listed at the end of the thesis). In a nutshell, the author is responsible for the following contributions:

Chapter 2.: The algorithmic descriptions and architectures of multi-channel action potential detection and hybrid template-based neural signal generator.

Chapter 3.: The idea and architecture of multi-channel online sorting using spatial

information.

Chapter 4.: Real-time operation of signal processing methods and simulation using FPGA. Real-time operating FPGA implementation of face recognition, sensor fusion, active suspension control and wind turbine HIL simulation are presented.

Chapter 2

FPGA-based real-time detection and synthesis of neurophysiological signals

Neurophysiological signal detection has been an area of active research in recent decades. Many algorithms and systems were developed to create reliable solutions. In this chapter a real-time multi-channel FPGA-based neurophysiological signal detector is introduced together with an a neurophysiological template-based signal generator for validation purposes.

The main novelties of the work are a multi-channel Non-Linear-Energy Operator (NEO) based multi-channel neural signal detector FPGA architecture capable of detecting on 128 different electrodes in real-time and a multi-channel neural signal generator, which can generate signals on 128 channels in real-time. The randomness of the neural signal generator is provided by a Mersenne-Twister based pseudo random number generator FPGA architecture working in real-time.

The structure of the chapter is as follows. Section 2.1 and Section 2.2 introduce the problem, summarizes the related works and briefly presents previous results. The description of the overall spike sorting system is presented in Section 2.3. The description of the neurophysiological signal generator is presented in Section 2.4. The comparison of various spike detection algorithms are covered in Section 2.5. The algorithmic description and architecture of the proposed multi-channel spike detection are covered in Section 2.6. Result and final thoughts are summarized in Section 2.8.

2.1 Introduction

The brain is one of the most complex biological systems containing quadrillions of synapses and billions of neurons. To study this organ in humans or in animal models,

and to examine elementary neuronal mechanisms or high-order brain functions, such as learning, sleeping, perception and memory, a widely used method can be applied: the extracellular measurement of the electrical activity in the brain. During a typical *in vivo* electrophysiological experiment a single or multiple neural implants comprising dozens of small electrodes are inserted into the brain tissue for recording short, electrical impulses (usually referred to as action potentials or spikes) generated by neurons located close to the implanted devices. In Fig. 2.1 an example can be seen for neural probes.

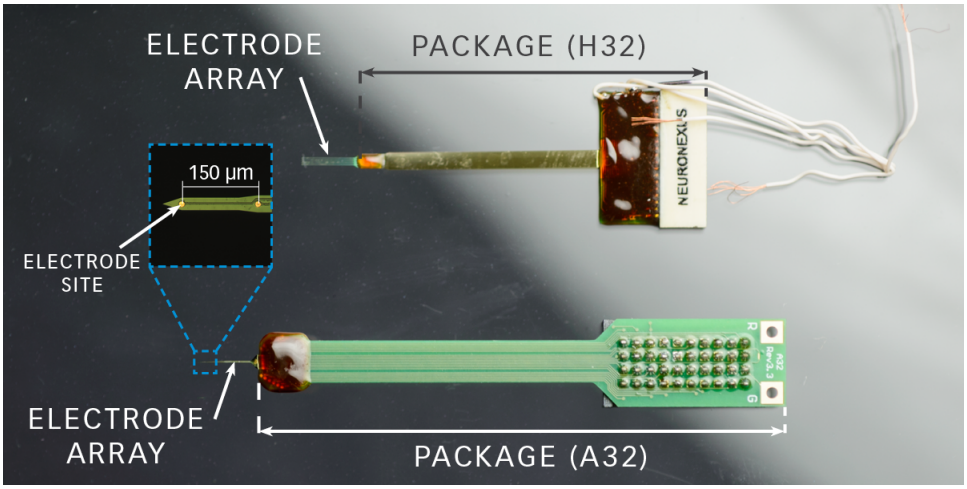

Figure 2.1: *Example of neural probes for measurements. [1]*

The obtained signals contain the trains (sequences) of action potentials fired by neurons located around the electrodes of the neural probe [2, 3].

Using a high electrode density ($<$20 μm electrode-pitch) is advantageous in multiple other ways. For instance, there is a higher chance that recording sites are physically located near individual neurons [4]. Furthermore, several studies showed that a higher single unit yield can be achieved with a higher electrode density [5], and that a considerable number of pyramidal cells fire spikes with a larger spatial spread than interneurons [6]. Thus, the spikes of several small interneurons located in the vicinity of neural probes having a low electrode density might not be recorded, resulting in a biased pyramidal cell – interneuron ratio. High electrode coverage and increased electrode density might also provide other benefits, including, for instance, the compensation for electrode drift or a more accurate separation of overlapping spike waveforms [7]. However, using an electrode pitch below 4 μm will not lead to any further improvement in terms of being close to neuronal signal sources [4].

Usually one of the first analysis methods which is applied on the recorded spiking activity data is spike sorting, which is used in many fields of basic neuroscience research (e.g. to study the dynamics of neural networks [8]). The identification of

spikes of individual neurons is of great importance in some real-time clinical applications as well (e.g. neuroprosthetic devices, brain-machine interfaces [9]). Usually, the first step in a spike sorting algorithms is the detection of spikes, which can be computationally intensive [10].

The performance of current spike detection methods is challenged by multi-channel neural data recorded with high-density, high-channel count silicon probes developed recently [11]. High-density neural probes with closely-packed recording sites can detect the spikes of the same neuron simultaneously on multiple, adjacent sites [12, 13]. In contrast, most spike sorting algorithms are prepared to process data recorded with only a few (usually four) electrodes.

Simple detection methods like amplitude thresholding can offset the detection towards neurons with high amplitude spikes. There are more robust detection methods, like Non-linear Energy Operator (NEO), which has great accuracy, but the number of falsely detected spikes can also impact the performance [14]. Therefore with the help of dedicated equipment e.g. FPGA, GPU or ASIC, the detection of spikes can be made real-time.

Other challenge is the validation of the spike sorting, because not even the highly experienced neuroscientists can annotate the recorded neural data with 100% accuracy. Therefore using a hand-annotated ground truth dataset can produce a deceptive validation result. Neurophysiological signal synthesis can be an important part of spike sorting to produce a reliable ground truth dataset and use it for automatic validation [15].

2.2 Related Works

In the literature of spike detection and neurophysiological signal synthesis many solutions can be seen. In the following various widely used methods are presented.

Spike detection

In [16] an analog template matching based spike detection is proposed, using 8 template points in a spike form discriminator called SPIFODIS. Even for very low Signal-to-Noise Ratio (SNR) values the error rate of the algorithm was 50%-15%. In multi-unit neural signals SPIFODIS can reliably isolate spikes, even when action potentials have equal amplitudes.

The authors of [17] combined wavelet transforms with basic detection theory to develop an unsupervised method for robustly detecting and localizing spikes in noisy neural recordings. The authors presented an extensive Monte Carlo simulation, based on actual extracellular recordings and demonstrated that false positive rate of

the algorithm is better than the false positive rates of the amplitude thresholding method or the power detection method.

An optimized amplitude threshold technique to detect the spikes in signals is proposed in [18] using Haar Wavelet to remove the noise, then a morphological filter to detect the spikes. The morphological filter utilizes hybrid GAPSO (Genetic Algorithm-Particle Swarm Optimization) technique to select the optimal structuring element. The optimized amplitude threshold techniques exploited Improved Particle swarm Optimization (IPSO) which is based on fractional velocity, which was implemented in Matlab and capable of achieving 71% average precision rate.

In [19] a template matching based detection algorithm is described that only requires the user to specify the minimum and maximum firing rates of the neurons. It is capable of achieving an accuracy of 90% with a false positive rate of 5 Hz in recordings with an estimated SNR of 3 dB using real recordings with added background noise taken from other recordings.

The authors of [20] presented two approaches for spike detection. First the ensemble empirical mode decomposition (EEMD) is used, then the Hilbert transform is utilized as a pre-processing step to achieve 100% accuracy and very low (4%) false positive ratio wit varying SNR. The second uses fuzzy and probability theories to combine a number of spike detectors to achieve also 100% accuracy and (5%) false positive ratio.

In [21] a genetic algorithm (GA) based method is proposed using the non-linear energy operator (NEO), which utilizes a GA to adjust the threshold value of the NEO technique. The method can recognize the number and the location of action potentials in a neural signal. The authors showed that the genetic algorithm gives better results than selecting the threshold manually and achieves an average accuracy of 86% with a low false positive rate testing on altogether 15 spikes with varying noise levels.

Neurophysiological signal synthesis

There are several methods to obtain “ground truth” datasets for spike sorting efficiency validation. Simultaneous paired recordings collecting the spikes of the same neuron both extracellularly and intracellularly would be the most optimal solution, however, this method is technically challenging, therefore the availability of such datasets are limited [22]. Simulating the activity of biophysically realistic neural networks is also an option [23], but the generation of synthesized multi-channel datasets needs high computational power provided only by computer clusters. Finally, “hybrid ground truth” datasets can be generated by using the mean spike waveforms of a subset of well-separated units isolated from real extracellular recordings as templates or donors and add these spike templates at random times and positions of simulated or real recordings [24, 25] .

2.3 Overall system architecture

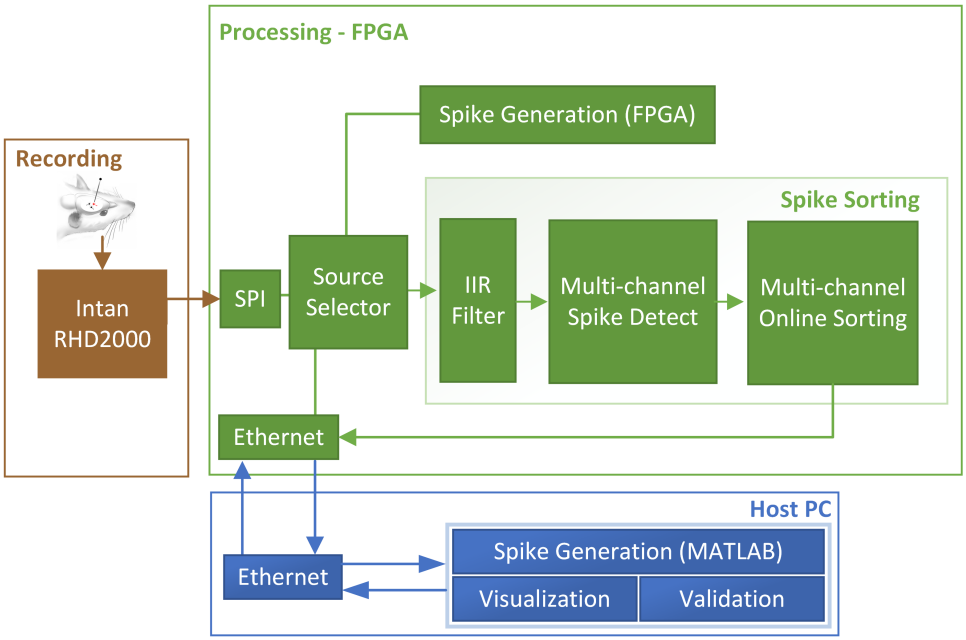

Figure 2.2: *Schematic diagram of the proposed spike sorting system.*

The overall system is presented in the actual Chapter, because it is necessary for the better understanding of the connections between the detection and synthesis parts detailed in this Chapter. The overall system information is also relevant in Chapter 3, where the classification part is detailed.

The schematic diagram of the proposed spike sorting system can be seen in Fig. 2.2. The proposed system can be split into three main blocks, the *Recording*, the *Processing* and the *Host PC*, which can be seen in Fig. 3.1. The *Recording* block contains an *Intan RHD2000* electrophysiological recording system. The *Processing* block can be further divided into the *Ethernet* interface, *SPI* interface, Pre-processing *IIR Filter* module, *Spike Generation (FGPA)* (see Section 2.4) module and the *Spike Sorting* module, which contains the *Multi-channel Spike Detect* (see Section 2.6) and the *Multi-channel Online Sorting* (see Section 3.3) cores. The *Host PC* block includes the *Ethernet* interface, the *Visualization* module, the *Spike Generation* module, and the *Validation* module.

The proposed system is able to process in vivo neural measurement acquired by the Intan RHD2000 (Intan Technologies, Los Angeles, CA, USA) board, as well as simulated neural recordings generated by the *Spike Generation* module on the *Host PC*. The path of the incoming neural data can be selected in case of *in vivo* (online), stored data (offline) or simulated operations. The selection is made using the *Source Selector* block, which handles the incoming data based on the configuration, which

can be altered using the switches on the FPGA board.

2.3.1 Electrophysiological Recording

The proposed system was designed to process neural data recorded with a 128-channel high-density silicon-based probe ($50\mu m \times 100\mu m$) comprising closely-packed electrodes (20 $\mu m \times 20\mu m$) arranged in a 32×4 array with a center-to-center electrode distance of $22.5\mu m$ [26]. Measurements of wideband brain signals (0.1-7500 Hz) with this type of probe were obtained with the Intan RHD2000 electrophysiological recording system at 20 kHz sampling frequency/channel and with 16 bit resolution. These 128-channel recordings usually contain hundreds of thousands of spikes fired by dozens of neurons during a time period of one hour. The Intan RHD2000 uses the SPI protocol for communication purposes.

2.3.2 Pre-processing IIR Filter

The high-density bioelectrical activity recorded from the brain tissue can be separated into local field potentials (below 500 Hz) and spiking activity (500-5000 Hz). Therefore the recorded wideband data should be filtered before spike sorting. In the proposed system a Butterworth Infinite Impulse Response (IIR) third-order zero-phase band-pass filter is used to extract the spiking activity.

The filter can be implemented as two Finite Impulse Response (FIR) filters combined together based on Direct Form I structure with seven 18 bit coefficients in each filter. The FPGA resource requirement of the pre-processing IIR filter on 128-channels is only one 36k BRAM and one DSP slice.

2.3.3 Processing - FPGA

The proposed system was implemented on ZCU106 FPGA board [27], which has a special Xilinx architecture called Zynq, which contains a traditional Programmable Logic (PL) and a Processing System (PS). The latter is an ARM-Cortex processor with various I/O interfaces to connect the system to the outside world. The Gigabit Ethernet interface is used to communicate with the *Host PC* and the SPI interface is used to receive data from the Intan RHD2000 device. The ARM PS communicates with the PL on AXI4 buses (AXI4-Lite, AXI4-Stream, AXI4).

2.3.4 Host PC

The task of the Host PC (Intel Core i7-4770 CPU at 3.4 GHz, 8 GB DDR3 RAM) is to visualize the results of the spike sorting, when in vivo neural recordings are used, and

to validate the sorting algorithm using hybrid ground truth neural data generation, when the signal generator is operating.

2.4 Neural Signal Generator

Developing an FPGA-based solution to accelerate the clustering of spikes detected in high-channel count neural recordings is pointless without knowing the spike sorting efficiency. It is a crucial step of the development to validate the performance of the sorting algorithm. This can be achieved by using "ground truth" datasets where the exact time of spikes fired by different single units are known. The "hybrid ground truth" method is chosen and a multi-channel hybrid ground truth dataset can be synthesized using an FPGA board for real-time usage or MATLAB for offline validation. These datasets are used for the validation of the proposed FPGA-based detecting and clustering algorithms (see Section 2.6 and Section 3.3).

2.4.1 Spike Generation Requirements

In order to generate realistic neural data (spikes) many requirements have to be fulfilled. The absolute refractory period of a neuron is 1-2 milliseconds long, during this period the neuron is not able to generate spikes, and the interspike intervals of a neuron have a log-normal distribution [28]. Therefore, a log-normally distributed random number generator should be applied. Furthermore, it was found that the spike-amplitude decays as $(1/r)^n$ with $1 <= n <= 2$ close to soma and $n >= 2$ far away, where r corresponds to the distance of the electrode from the recorded neuron. Based on this decay and on a 128-channel model probe, where the electrodes are arranged in a 32×4 matrix shape with an electrode pitch of 20 μm, the action potential of an average neuron can be detected at most six electrodes (120 μm) away from the center, where the spike can be recorded with the highest amplitude [29]. Finally, in the first approach, noise with uniform distribution was added to the generated data.

In addition to these requirements, real-time multi-channel spike generation is required to test the multi-channel Online Sorting architecture (Section 3.3) where the main parameters like the number of neurons, the number of channels and the size of the inter-channel action potential spreading should be configurable.

The sampling rate of the real measurement system where the OSort architecture will be used is 20 kHz, therefore in this case real-time means that the spike generator should produce a sample on all channels with at least 20 kHz frequency.

2.4.2 Mersenne Twister Random Number Generator

To generate the neural dataset, uniform and log-normal distributed random number generation is required. In the literature various uniform random number generation methods can be found. The pseudo-random Mersenne Twister (MT) algorithm is chosen, which can provide an astronomical period of $2^{19937} - 1$ and 623-dimensional equidistribution up to 32-bit accuracy. Furthermore, it passes several statistical tests, including diehard. The algorithm is based on polynomial calculations over the two-element field and uses a twisted generalised feedback shift register with the help of a tempering matrix. The pseudo code and the detailed description of the algorithm can be found in [30].

A continuous probability distribution of a randomly generated variable, which logarithm is normally distributed, is called log-normal distribution. The log-normally distributed random number generation is based on the output of the MT algorithm normalized into the $(0, 1]$ interval. The inverse transform sampling method can be used to generate random numbers from any probability distribution, if the cumulative distribution function (CDF) is known and invertible. The CDF of the log-normal distribution can be written as follows:

$$\frac{1}{2} + \frac{1}{2}\,\mathrm{erf}\Big[\frac{\ln x - \mu}{\sqrt{2}\sigma}\Big], \tag{2.1}$$

where x is the probability variable, μ is the mean, σ is the variance, and erf is the error function, which can be written as follows:

$$\frac{1}{\sqrt{\pi}} \int_{-x}^{x} e^{-t^2} dt, \tag{2.2}$$

and $erf(x)$ describes the probability of a random variable X falling in the range $[-x, x]$. Inverting the log-normal CDF, the quantile function can be given by:

$$e^{\mu + \sigma \Phi^{-1}(p)}, \tag{2.3}$$

where p is a uniformly distributed random number in the range of $(0, 1]$, and Φ^{-1} is the inverse error function, which can be approximated using a Maclaurin series as follows:

$$\sqrt{\pi}(\frac{1}{2}z + \frac{\pi}{24}z^3 + \frac{7\pi^2}{960}z^5 + \frac{127\pi^3}{80640}z^7 ...), \tag{2.4}$$

where z is an uniformly distributed random number in the range of $(-1, 1)$.

2.4.3 FPGA architecture

In Fig 2.3 the schematic diagram of the proposed spike sorting system can be seen, where the Spike Generation (FGPA) part is emphasized and detailed.

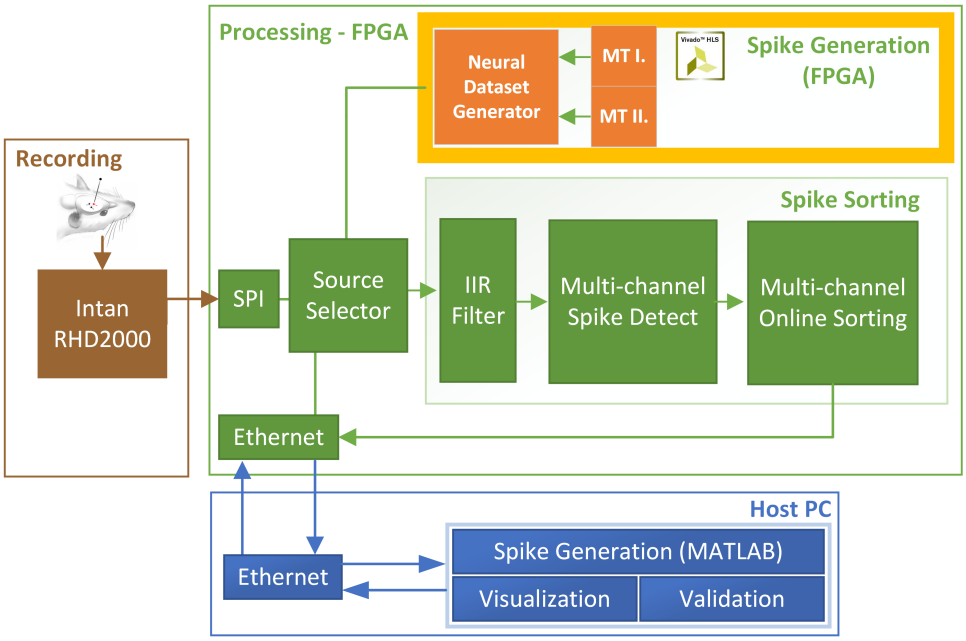

Figure 2.3: *Schematic diagram of the proposed spike sorting system. The Spike Generation (FGPA) part is emphasized and detailed.*

The Spike Generation contains two main parts: the *Mersenne Twister* (MT I. and MT II.) cores, and the *Neural Dataset Generator* core. The *Mersenne Twister* cores are responsible for the generation of random numbers used in the inverse transform sampling to compute the log-normal distribution of interspike intervals (*Mersenne Twister I.* core) and the noise generation (*Mersenne Twister II.* core).

The FPGA implementation of the log-normally distributed random number generator is done, however the *ARM Processing System (PS)*, which is a dedicated CPU on the FPGA board has enough computation performance to complete this task. In this way, the log-normal random number calculation does not occupy any FPGA resources.

The *ARM (PS)* also handles the communication (SPI, Ethernet) and the DDR memory access through DMAs.

Mersenne Twister

The Mersenne Twister I core calculates the random numbers required in inverse transform sampling to generate log-normally distributed firing times. The Mersenne Twister II core is responsible for the generation of the noise.

These cores are based on the pseudo-code and parameters published in [30], but some modifications were required to achieve optimal performance during the FPGA implementation. The initialization phase of the algorithm is not changed, but the twist and the output generation is separated, since only the output generation can be pipelined. Using this modified algorithm the twist step can be performed in 1872 clock cycles, when N is 624. After this initialization period is completed a new random number can be generated in each clock cycle. The twist step is called after generating 624 random numbers.

Neural Dataset Generator core

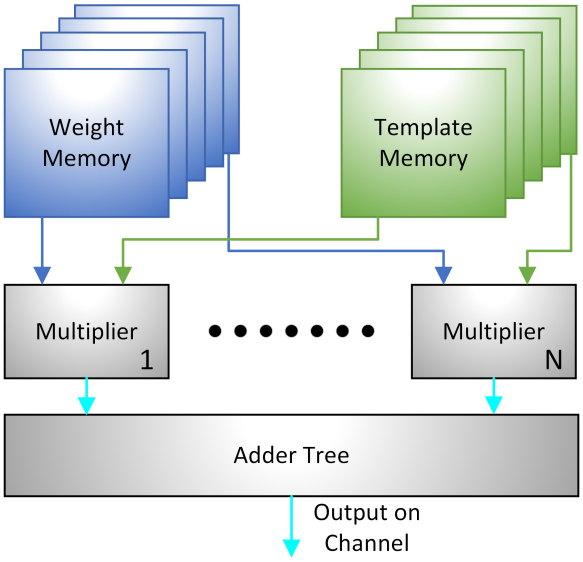

Figure 2.4: *Schematic diagram of the output generation for one channel*

The *Neural Dataset Generator* core generates the multi-channel neural dataset. The first part of this process is the simulation of the action potential spreading through the neighbouring 6 channels. This is based on a pre-defined normally distributed Gaussian-kernel. The shape of the electrode array is stored in a Look Up Table (LUT), which is used for proper indexing of the weight matrix. For each channel and each neuron, the weight matrix contains the corresponding spreading weight value from the Gaussian-kernel. The electrode array LUT and the weight matrix is pre-calculated on the *ARM PS*. Therefore, the generation of one output sample is a simple multiplication between the weight coefficient of the actual channel and the value of the actual spike template. Finally, an adder tree summarizes the weighted template values and the result will be the output of one channel at one time. The architecture of the *Neural Dataset Generator* core can be seen in Fig. 2.4.

Another important part of the dataset generation is to set up the log-normally distributed intra-channel spike firing distances for each individual neuron. On the *ARM PS* the generated log-normally distributed random numbers are sorted in ascending order with the corresponding neuron values and stored in a structure. This firing structure contains the firing time and the identification number of the firing neuron.

To generate the final multi-channel dataset, the *Neural Dataset Generator* core reads the neuron identification value from the firing time structure and enables that neuron to fire at the corresponding time stamp. The next fire time will be read in the next clock cycle. When all neurons are enabled to fire, then the *Neural Dataset Generator core* pauses the reading.

The *Spike Sorting* part processes the data from the electrode by channel, so the core gets one channel sample per clock cycle. Therefore the *Neural Dataset Generator* core should work in the same fashion.

Table 2.1: *Area requirements and device utilization of the synthesized modules*

	Available resources			
	BRAM	DSP	FF	LUT
	102	120	12,150	14,034
	Device Utilization			
	BRAM	DSP	FF	LUT
Mersenne Twister x2	1%	7%	1%	4%
Log-normal (ARM)	4%	20%	3%	11%
Neural Signal Generator	7%	34%	6%	25%
Summarized	8%	41%	7%	29%

2.4.4 Implementation Results

The proposed *Mersenne Twister* and *Neural Spike Generator* cores (Signal Generation subsystem) are developed and synthesized using the Vivado High Level Synthesis (HLS) version 2016.4. The spike generation architecture is synthesized to a ZedBoard equipped with a Zynq-7020 FPGA. The available resources on the Zynq-7020 and the resource requirements of the cores can be seen in Table 2.1.

During the development, the FPGA and the ARM-based implementation of the *Log-normal* core were tested. Since the *ARM PS* is capable to generate log-normally distributed random numbers in 150 FPGA clock cycles, the ARM PS was chosen for this calculation. In this case this core does not require any FPGA Programmable Logic resources.

Table 2.2: *Detailed Latencies of the Synthesized Cores*

		Latency	Iteration Latency	Trip Count
Mersenne Twister	init	5607	9	623
	twist	1872	3	624
	output_gen	625	3	624
Log-normal (ARM)	init maclaurin output_gen	57 140 65	27 14 25	32 128 32
Neural Signal Generator	dataset_gen	2560026	28	256000

If the resource requirement of the multi-channel Spike Detect and Online Sorting (Section 3.3) is added to the resource requirements of the Spike Generation subsystem, the DSP resource utilization is greater than 115%. Therefore, the Log-normal core can not be implemented on the Zynq-7020 FPGA, so it is important to move this computation to the *ARM PS*.

After the Vivado HLS synthesis, the latencies of the cores can be seen in Table 2.2. In the case of the *Mersenne Twister* core, the twist step requires 1872 clock cycles, while the output generation requires 624 of it. So, $1872 + 624 = 2496$ clock cycles are needed to generate 624 random numbers. Therefore, an average of 4 clock cycles are required to generate one random number. The *Log-normal* core running on the *ARM PS* requires 150 clock cycle latency to generate one log-normal value. The spike generation is performed in one second windows using the *Neural Signal Generator* core. This core requires only one clock cycle to generate one sample on all 128 channels. Before the first window, a proper initialization is required. This initialization consists of the electrode array LUT, the weight matrix, the random number and the log-normal number generation. These calculations are performed by the *ARM PS*, except the random number generation, which is done by the *Mersenne Twister II.* core. During the generation of the channel samples of the next windows, only the log-normal number generation is required, which can be performed parallel to the previous window. The clock frequency of the Zynq-7020 on the ZedBoard is 100 MHz, so the generation of the channel samples can reach and even exceed the necessary 20 kHz sampling rates.

In Fig. 2.5 the results of the dataset generation can be seen on 5 adjacent channels with 3 firing neurons in a 0.1 second window, where the inter-channel signal spreading can be observed. The spikes are grouped by the firing neurons, each neuron has a unique color, while multi-unit spikes have black color.

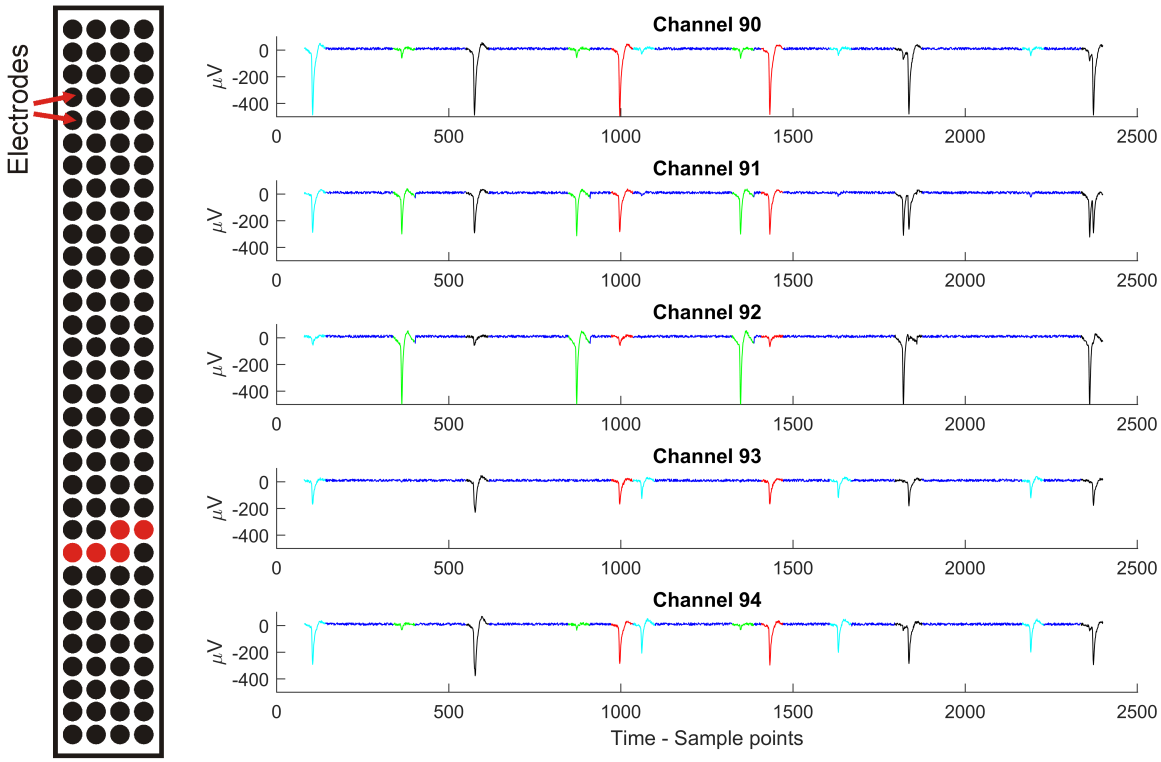

Figure 2.5: *A 0.1 second long segment of the generated neural data on 5 adjacent channels (electrodes corresponding to the channels are indicated on the left side with red circles on the schematic image of the model probe). The spikes of the three different neurons are highlighted in different colors.*

2.4.5 Conclusion

A real-time multi-channel hybrid ground truth neural dataset generator is presented using a Zynq-7020 FPGA. The results show that the presented architecture can be implemented on the Zynq-7020 FPGA with 100 MHz clock frequency and it is capable of generating multi-channel neural samples at every clock cycle. The neural dataset generator are used for the validation of the spike sorting architecture. Another purpose of the proposed neural spike generator could be the pre-calibration or pre-teaching of the spike sorting architecture.

2.5 Comparison of spike detection algorithms

In Fig. 2.6 a conventional spike sorting system can be seen, where the spike detection step is highlighted. In a conventional spike sorting system the electrical signal from the neural probe is converted to a digital signal first, then a bandpass filter is applied to reduce noise, because the action potentials are often in the 300 Hz - 3000 Hz fre-

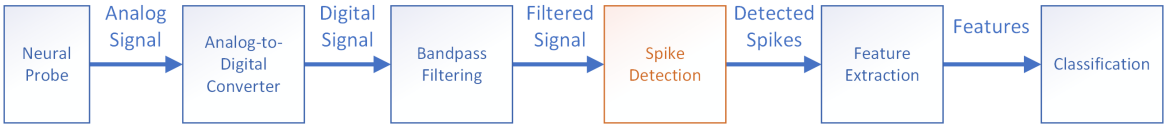

Figure 2.6: *Structure of a conventional spike sorting system*

quency range. The spike detection step is applied after the filtering, then the features are extracted from the detected spikes. Based on the features the classification can be done [31].

There are several simple spike detection method, which offers easy implementation, but reliable accuracy. These are the NEO, the power method, the positive, the negative, and the absolute amplitude thresholding. Spike detection methods, which based on creating an intermediate signal often suffers from creating a lag between the detection and the input signal. A realign step has to be added to mitigate the lag, which finds the peak of the spike around the detection in a pre-determined window.

2.5.1 Non-Linear Energy Operator

The Non-Linear Energy Operator is widely used for spike detection. In discrete time the Ψ operator can be defined as follows:

$$\Psi[x(n)] = x^2(n) - x(n+1)x(n-1) \tag{2.5}$$

The NEO signal is high only when the input signal is high in power (x^2 is high) and also high in frequency ($x[n]$ value is big, while $x[n+1], x[n-1]$ is small). Action potentials are often characterized as instant impulses, therefore this method has an advantage [32].

The acquired NEO signal is thresholded with an automatically calculated threshold value T_{NEO}.

$$T_{\text{NEO}} = C_{\text{NEO}} \frac{1}{N} \sum_{n=1}^{N} \Psi[x(n)] \tag{2.6}$$

where N is the number of samples, and C_{NEO} is the scaling factor, which is adjusted by experiment for every signal type [33].

2.5.2 Power Method

The standard deviation is computed of the signal using a sliding window as the power signal. Then the mean and the standard deviation of this signal is computed, finally the power signal is thresholded with an automatically calculated threshold value

from the standard deviation and the baseline mean. The power signal is calculated as follows.

$$s[n,m] = \sqrt{\frac{\sum_{i=n}^{m}(x_i - \bar{x}[n,m])^2}{i-1}} \tag{2.7}$$

where $s[n,m]$ is the sliding standard deviation and $\bar{x}[n,m]$ is the sliding mean between n and m samples. The used kernel size in this case is 18 samples, which is a typical spike width($0.9ms$ at 20 kHz sampling rate). The sliding mean is defined as follows.

$$\bar{x}[n,m] = \frac{\sum_{i=n}^{m} x_i}{m} \tag{2.8}$$

where $\bar{x}[n,m]$ the average value of the input signal between the n and m samples. The automatic threshold value T_P is calculated as follows.

$$T_P = \bar{s} + C_P\sqrt{\frac{\sum_{i=1}^{N}(s_i - \bar{s})^2}{N-1}} \tag{2.9}$$

where $\bar{s}$ is the mean value of the power signal, N is the number of samples and C_P is the scaling factor [34].

2.5.3 Amplitude Thresholding (Positive,Negative,Absolute)

The simplest spike detection algorithms are the amplitude thresholding methods. There are 3 different approaches, where the signal, which is being thresholded are different:

- Positive - Uses the signal without modification
- Negative - Negates the signal
- Absolute - Uses the absolute value of the signal

The automatically calculated threshold value T_{AMP} is defined as follows.

$$T_{\text{AMP}} = C_{\text{AMP}}\frac{median(|x|)}{0.6745} \tag{2.10}$$

where x is the input signal, and C_{AMP} is the scaling factor. The 0.6745 denominator value is based on the assumption that the noise in the signal has a normal distribution, therefore the denominator is the inverse of the cumulative distribution function (CDF) of the standard normal distribution at 0.75 [35].

2.5.4 Cross-Correlation Based Spike Detection

Using correlation for template-matching is a usual approach, although in neural activity detection is rarely used due to the base template requirement. A cross-correlation based spike detection implementation is proposed with pre-defined average templates, which requires an expert neuroscientist who is capable of selecting spikes in a pre-recorded signal. With this a priori knowledge the average template will be similar to the spikes in the actual recording, therefore the cross-correlation can be effective. The cross-correlation R_{xy} is calculated as follows.

$$R_{xy}[k] = \sum_{-\infty}^{\infty} x[m]y[m-k], -\infty \leq k \leq \infty \tag{2.11}$$

where x is the average template and y is the signal.

A normalization step is required, so the results will be in the $[0, 1]$ range, which calculated as follows.

$$\hat{R}_{xy} = \frac{R_{xy}}{max(|R_{xy}|)} \tag{2.12}$$

where $\hat{R}_{xy}$ is the normalized cross-correlation. After the normalization the spikes are extracted by a threshold T_{XCorr} on the resulting signal.

$$T_{XCorr} = 0.3 \tag{2.13}$$

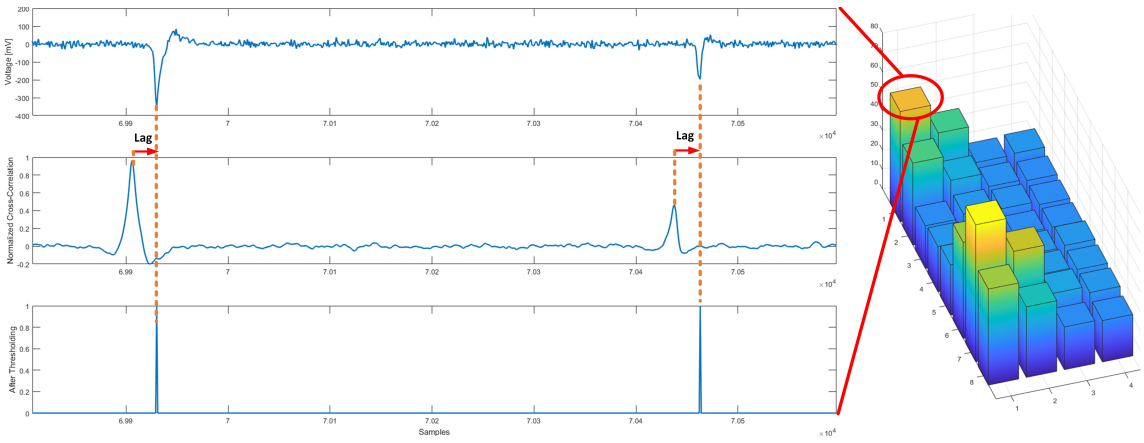

Figure 2.7: *Cross-correlation based spike detection, visualization on channel 1. From top to bottom: generated neural (3 neurons) signal, normalized cross-correlation, results of thresholding and realign. The generated neural signal on the 4x8 electrode array can be seen on the right*

In Fig. 2.7 the cross-correlation based spike detection can be seen, with the generated neural signal on 4×8 electrode array.

Without the right quality templates, the proposed method is not suitable for real-time spike detection, but it can be a good candidate for re-detecting and re-classifying an already partially classified recording.

2.5.5 Evaluation

The comparison between different spike detection methods are evaluated using the known spike times from the template based neural signal generator (see Section 2.4) using MATLAB. The neural signal generator is capable of generating a maximum channel number of 128, but it is reduced to 32 channels (4×8) in this case. Furthermore it simulates the inter-channel action potential flow with a gaussian kernel, uses a random firing frequency for each neuron and the noise level is variable.

The signal generator has been set to iteratively generate 4 dB - 10 dB SNR signals with 3-12 neurons, for each iteration all of the detection methods has been evaluated, altogether on 90 combination.

The SNR is calculated as follows.

$$\text{SNR} = 10 \log_{10} \frac{\text{Energy of Signal}}{\text{Energy of Noise}} \tag{2.14}$$

One evaluation metric is the true positive ratio (TP), which can be defined as:

$$\text{TP} = (\frac{\text{Number of good spikes found}}{\text{Number of spikes}}) \cdot 100 \tag{2.15}$$

Second metric is the false positive ratio (FP), which computed as:

$$\text{FP} = \frac{\text{Number of false spikes detected}}{\text{Number of spikes}} \tag{2.16}$$

Table 2.3: *Comparison of different spike detection methods, with 12 neurons, 10,7,4 SNR*

Method	**SNR**					
	10 dB		7 dB		4 dB	
	TP [%]	FP	TP [%]	FP	TP [%]	FP
NEO	100	50.40	100	29.15	100	17.55
Power	32.87	0.67	32.86	0.38	35.84	3.24
AMPP	0	19.142	0	8.42	0	1.79
AMPN	100	37.97	100	24.39	100	18.09
AMPA	100	58.87	100	33.97	100	21.03
Cross-Corr.	99.41	0.53	96.63	0.53	92.47	0.52

2.5.6 Results and Conclusion

The results are summarized in Table 2.3, which contains the results for 12 neurons on 32 channels with varied 4, 7, 10 dB SNR. It can be seen, that the AMPP method unfortunately cannot hit any good spike, only false. It is an expected outcome, due to the generated signal only contains spikes, which are in the negative region.

The NEO, AMPN and AMPA methods are very accurate even at low SNR value, but have a high false positive ratio. The NEO spike detection found 50 times more spikes than the true spikes in the signal. In the case of the NEO it can be seen, that the false positive ratio is decreasing as the SNR increasing, which can be explained with the threshold computation of the NEO. The Power spike detection method has a low accuracy, but also a low false positive ratio.

However the proposed cross-correlation based method maintains accuracy over 90% and the false positive ratio is at a maximal value of 0.53.

It can be concluded, that the cross-correlation based method has lower accuracy, then the NEO, AMPN or AMPA, but has a far lower false positive ratio, but NEO do not requires stored templates and false positives are decreasing as the noise increases, which makes NEO a good candidate for FPGA implementation.

2.6 Multi-channel Spike Detection

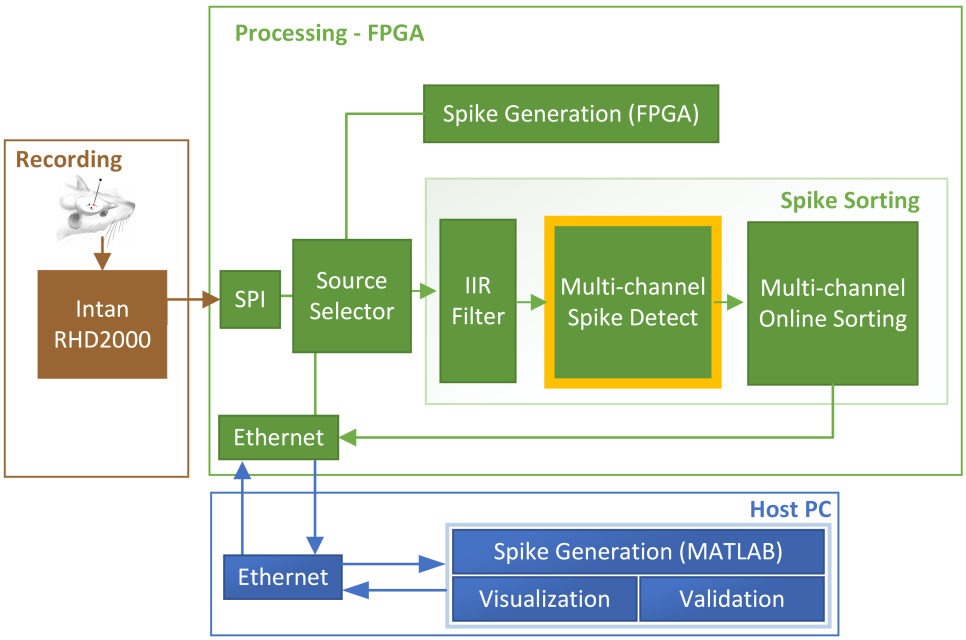

Figure 2.8: *Schematic diagram of the proposed spike sorting system. The Multi-channel Spike Detect part is emphasized.*

Single-channel spike detection is easier, because there are no adjacent electrodes, therefore overlapping (multi-unit) spikes are less likely to appear and the number of active cells are small. On the other hand, using closely packed electrodes the number of cells are proportionally increasing with the number of channels and also the structure of the electrode array is important. A new source of information appears, which is the spatial component. The action potential of a firing neuron propagates through the electrodes with decreasing amplitude in all direction. The spatial information can point towards the source of the action potential to the channel with the highest action potential amplitude.

In Fig. 2.8 the schematic diagram of the schematic diagram of the proposed spike sorting system can be seen. The Multi-channel Spike Detect part is emphasized and it is detailed in this Section.

2.6.1 Spike Detection Algorithm

In the FPGA architecture the spike detection part is called *Multi-channel Spike Detect* core, which is responsible for the real-time multi-channel spike detection. In the proposed system the Non-Linear Energy Operator (NEO) is used for spike detection, because it is the most efficient among the commonly used spike detection methods and it is easily implementable on FPGA devices [35, 36, 37].

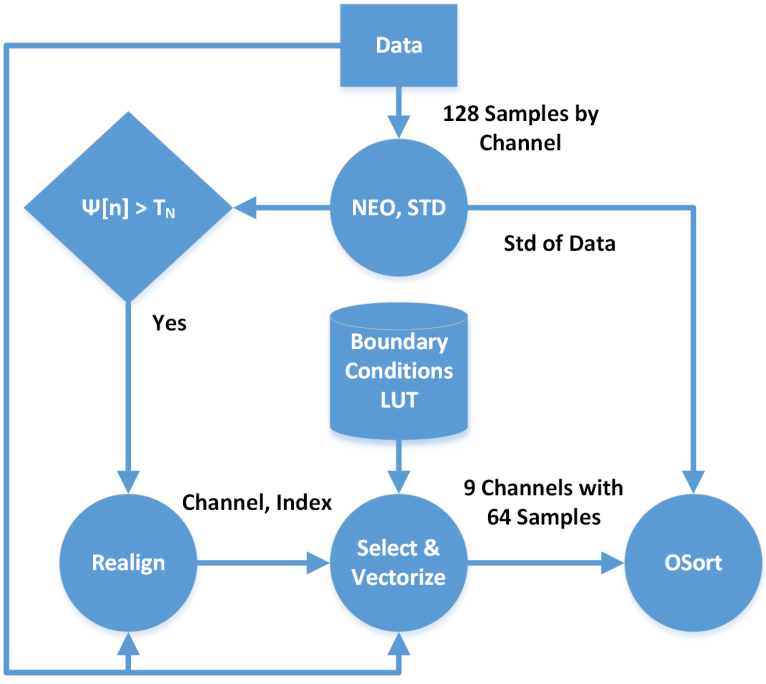

Figure 2.9: *Flow diagram of the NEO based spike detection algorithm*

The flow diagram of the spike detection algorithm can be seen in Fig. 2.9. In this

case one sample from each of the 128 channels is fed into the NEO and the standard deviation (STD) calculation block. The STD value is used later in the spatial window-based OSort module to calculate the automatic clustering and merging thresholds. When the $\Psi[x(n)]$ NEO value is larger than the automatically calculated T_N threshold, then the peak of a spike is detected. Afterwards, a temporal window around the detected peak on the actual channel is checked by the realign method, that the peak of the spike is certainly in the center of this temporal window. The correct alignment is crucial, because the comparison can not be done correctly with unaligned spikes, resulting in compromised classification.

In an electrode array with closely-spaced recording sites the spike waveform of a neuron spreads through adjacent sites, which also has to be considered for detection. After the realignment of the temporal spike window, a spatial window of 3×3 electrodes is examined around the firing channel at the aligned spike position to determine the source of the activity. The channel with a higher absolute amplitude in the neighbourhood will be the new center of origin, assuming that electrode is possibly located the closest to the soma of the neuron. Using this method the source channel of the unit activity can be found, eliminating the effect of the multiple detection of the same spike. In case of two or more neurons firing in the spatial vicinity of each other (8-connected or 8-neighbour) the detected spike matrix will be an overlapping spike waveform and will be removed in the merge phase of the classification process. Recording sites on the edges of the electrode array will also detect spikes. In this

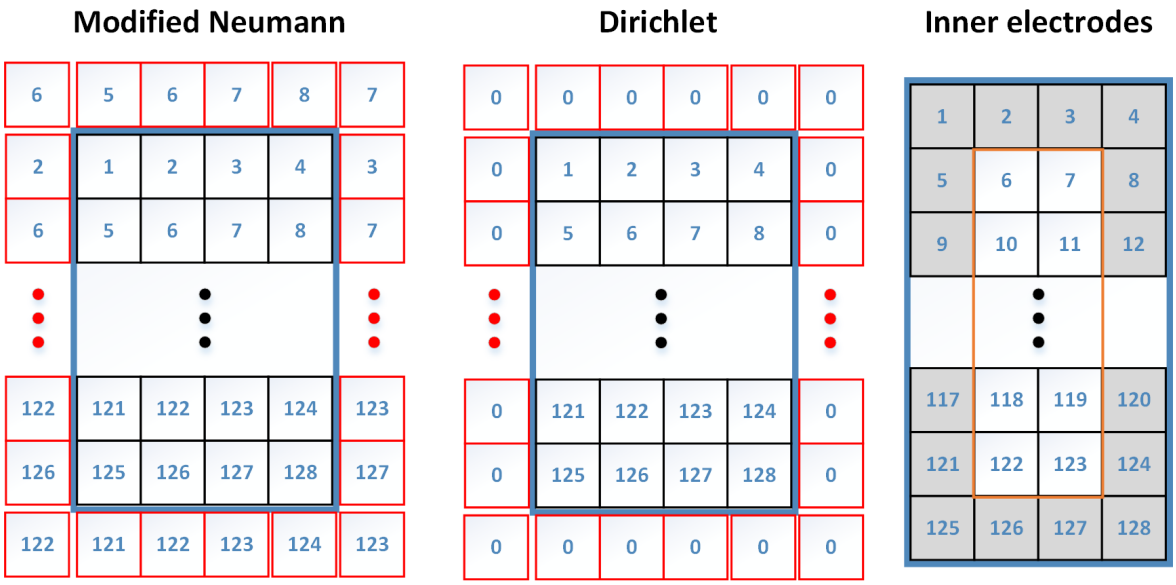

Figure 2.10: *Possible boundary conditions for the 32 x 4 electrode array configuration, which is stored in the LUT. The recording sites of the electrode array are within the blue rectangle.)*

case an incomplete electrode window is selected, because there is no measurement beyond the edges of the electrode array. There are three possible solutions to deal with this issue, which can also be seen in Fig. 2.10:

- *Modified Neumann* - The boundary is filled with data of the neighbouring channels, considering a Gaussian distributional cross-talk between the channels.
- *Dirichlet* - The boundary is filled with fixed constants, in this case zeros.
- *Inner electrodes* - Only the inner electrodes are used for detection, therefore spike sorting is limited in this case.

The realign method specifies the origin of the activity and the precise alignment, so a spike matrix containing the data of 9 channels is selected from the electrode array and sent to classification. In the proposed system the *Modified Neumann* boundary condition is used.

2.6.2 FPGA Architecture of Spike Detection

The *Multi-channel Spike Detect* architecture can be split into the STD, the NEO and the Realign computation parts, which can be seen in Fig. 2.11. The signal is sampled at $20\ kHz$ with 128 channels in the proposed system.

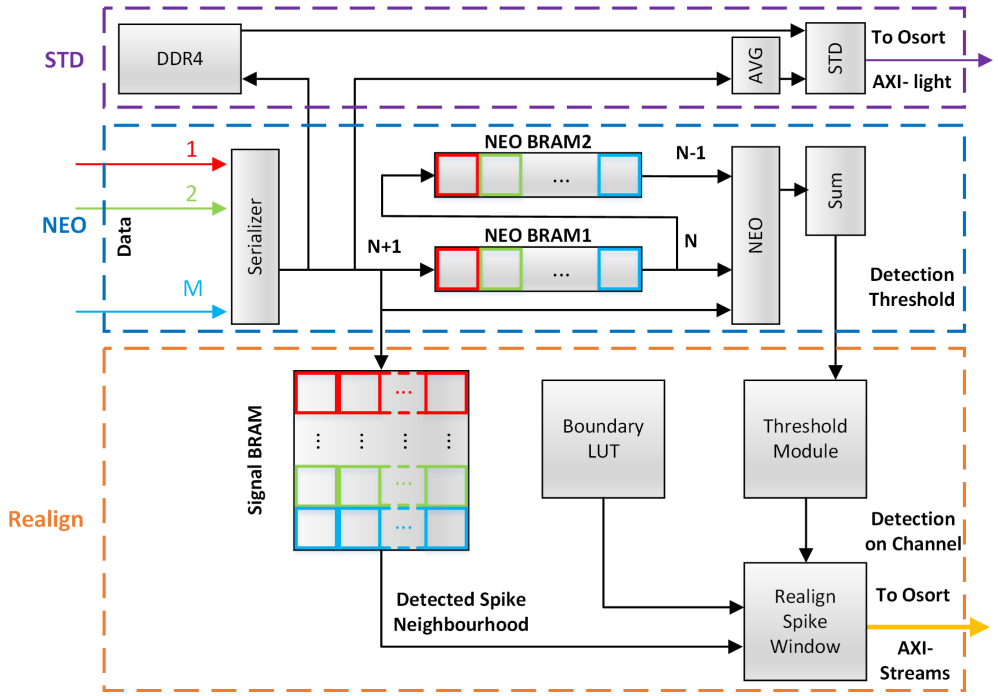

Figure 2.11: *The architecture diagram of the detection*

The STD computational part is responsible for computing the standard deviation value of the incoming signal using a five seconds window, which contains 100000×128 samples. There is not enough memory for this on the FPGA, therefore the off-chip DDR4 memory is used to store the samples. The average computation can be done continuously in the AVG block, while the subtraction and power operations are done in the STD block. In the NEO computation part the spike detection with Eq. 2.5 and the threshold calculation using Eq. 2.6 is done. The NEO signal can be calculated parallel for each channel using the NEO BRAM1, BRAM2 and the Serializer as N-1, N and N+1 inputs.

In this implementation the value of c_N and N are both 2 (determined experimentally), so the T_N NEO threshold value calculation can be reduced to a simple accumulation. At the first 2 samples the system is not operating, this can be considered as an initialization phase.

The Realign part of the *Multi-channel Spike Detect* is responsible for the following:

- Alignment of the detected spikes in the spike window;
- Determination of the channel containing the maximal amplitude spike in the neighbourhood around the detection;
- The appropriate selection of the 3×3 channels on the electrode array.

The Realign part uses the Signal BRAM, which can store 80 samples for each (128) channel. A spike is 64 samples wide, but the detection is not precise, so 16 more samples are stored, altogether 80. The Threshold Module compares the actual NEO value on each channel with this detection threshold value and when it is lower than the actual NEO value, then a spike is detected. Detection events for the channels are stored in a 128 wide vector. If a detection occurs, then the detection event flag for the actual channel is switched, which indicates that 40 samples later the Signal BRAM will contain the full spike for that channel and can be used for realignment. The Realign Spike Window block uses the absolute value of the spike in the temporal window from the Signal BRAM to find the maximal point for the alignment.

Afterwards, the neighbourhood of the detected spike at the alignment point is extracted from the Signal BRAM, and searched for the maximal value in that 3×3 electrode window. Using this method the channel with the maximal amplitude will be in the center of the electrode window, which is crucial for the effectiveness of the comparison performed in the *Window-based Online Sorting* core later. If the channel is at the border of the electrode array, and the spatial window is out of this border, then with the help of the Boundary Look Up Table (BLUT), the 3×3 spatial window can be prepared and sent to classification using AXI-Streams, 9 streams parallel for each channel.

2.6.3 Results and Discussion

Table 2.4: *Area requirements and device utilization of the synthesized multi-channel spike detection architecture*

	Utilization	%
FF	10915	2.36%
LUT	42692	18.52%
DSP	6	0.35%
BRAM	7	1.12%

Table 2.5: *Detailed latency of the multi-channel spike detect architecture*

	Latency(clock cycles)	Trip Count(clock cycles)
Load Channel	129	128
NEO	131	128
Realign	163	154
Total	423	410
STD	12.8 M	12.8 M

In Table 2.4 the area requirements and device utilization of the synthesized multi-channel spike detection architecture can be seen, which shows that the LUT requirement is the highest with 18%. Based on the detailed latency of the multi-channel spike detect architecture in Table 2.5, it can be concluded that spike detection part can be done in 423 clock cycles, 2.12 μs at 200 MHz clock frequency, therefore the proposed multi-channel spike detection architecture can process up to 471 698 spike/s. The STD block is shown in a separated row, because it is required to run independently and does not influence the operation latency of the detection. Table 2.3 shows the efficiencies of several spike detection methods, the NEO is included, which is used in the proposed architecture. The efficiency of the multi-channel spike detection FPGA architecture is very close to the MATLAB simulation, only the fix-point precision and the number representational approximations are different.

For different electrode array configurations different pre-calculated Boundary LUTs could be used, which needs different bit files on the FPGA. The appropriate one must be chosen for the corresponding electrode array and programmed it to the FPGA.

2.7 Discussion and concluding remarks

An FPGA-based multi-channel hybrid template-based multi-channel neural signal generator for the simulation and a multi-channel spike detector using NEO have been developed for the detection of neural signals from closely packed microelectrode arrays. The main challenges with the simulation were the randomness of the activities and the modelling of crosstalk between channels. Also the main importance was to resemble reality as much as possible. The spike detection also posed challenges with the overlapping spikes due to the crosstalk and to find the source channel of a spike ”wave”. The neural signal synthesis were verified with neuroscientists and the evaluation of the detection showed promising results for real-time usage.

2.8 Contributions

The author of this PhD thesis is responsible for all the contributions presented in this chapter, which include the following main novelties:

I / 1. I gave a Non-Linear Energy Operator (NEO) based multi-channel action potential detecting FPGA architecture, which also capable of determining the source of the firing neuron in a multi-channel electrode array. I also showed that it can operate in real time.

I / 2. I proposed a cross-correlation based action potential detection method, showed its efficiency and pointed out its limitations.

I / 3. I designed a neurophysiological signal generator algorithm, which is capable of placing action potential templates on randomly selected channels with randomly determined frequency and firing crosstalk, taking into account the theory of neuron firing known so far in neuroscience.

I / 4. I gave an FPGA architecture, which is capable of generating neurophysiological signals using the Mersenne-Twister pseudo-random number generator algorithm. I showed that the architecture can generate real-time neurophysiological signals for an electrode array with 128 channels.

Chapter 3

FPGA-based real-time classification of neurophysiological signals taking into account spatial information

In this chapter the real-time FPGA-based classification of multi-channel neurophysiological signals using also the spatial information from the closely packed microelectrode array is described. An unsupervised single-channel online classification algorithm called Online Sorting (OSort) is chosen for FPGA implementation and modification to handle multi-channel neural signals. The main challenge is to handle the signals coming from a microelectrode array with 128 electrodes sampled with 20 kHz, which means $2560000\ samples/s$.

The classification step is strictly bound to the results of the previous step, the detection. In some methods also a feature extraction step is present, but the OSort algorithm uses the detected spikes as features, also called templates. Therefore in this chapter the results are presented with the detection step, forming together a spike sorting system.

The structure of the the chapter is as follows. In Section 3.1 and Section 3.2 the introduction and related works are presented. The detailed description of spatial information based OSort for real-time spike sorting using FPGA can be seen in Section 3.3. The discussion and concluding remarks is presented in Section 3.4. In Section 3.5 the contributions can be seen.

3.1 Introduction

High-channel-count neural probes ([7, 38, 39]) comprising over hundred electrodes are able to record the activity of hundreds of neurons from numerous individual brain positions simultaneously. Spike trains of individual neurons (called single-unit

activity) can be separated from the detected multi-unit activity in the classification step of spike sorting [35].

In basic neuroscience research spike classification or clustering is used during the offline analysis of the recorded neural data as well as in real-time clinical applications (e.g. in brain-machine interfaces to control neuroprosthetic devices [40]). However, using a typical spike sorting solution with general-purpose computers the real-time processing of multi-channel neural data is challenging and can greatly reduce the efficiency of clinical applications designed to provide rapid feedback.

Therefore, real-time clinical applications as well as electrophysiological experiments performed using high-density neural probes could get advantage of hardware-accelerated clustering of neural recordings. Dedicated systems based on Field-Programmable Gate Arrays (FPGAs) or Application-Specific Integrated Circuits (ASICs) are especially suited for this task because they are small, wearable and use insignificant amount of energy compared to traditional Central Processing Units (CPUs) or Graphics Processing Unit (GPUs). Using a dedicated system the computation time to process the vast amount of data from a multi-channel (high-dimensional) neural recording can be significantly reduced ([41, 42, 43, 44]). Although ASIC chips might be smaller and consume less power compared to FPGAs, which is advantageous in electrophysiological experiments with freely behaving animals where wireless technology is used to transfer neural data from the brain to the recording system [45], they usually lack flexibility for changes. In contrast, design flexibility provided by FPGAs might be in many cases a more important factor than small chip area or low power consumption. For example, electrophysiological recording systems as well as algorithms used for spike classification are subject to extensive research and development, therefore tend to change rapidly.

The high spatial sampling of spikes might allow a more reliable and accurate identification of neurons in case of template matching-based spike sorters. The reason behind this is that neurons located in different positions relative to the electrode array will have at least a slightly different multi-channel spike waveform (investigated for example in a spatial window incorporating 3×3 electrodes). Thus, the additional spatial information provided by high-density probes might be exploited to increase the accuracy of the spike sorting process [6].

Therefore, new algorithms using novel approaches and/or implementing the clustering step of spike sorting on dedicated hardware (e.g. Field Programmable Gate Array (FPGA), Application-Specific Integrated Circuits (ASIC), Graphics Processing Unit (GPU)) are under intensive development to reduce the computation time required to process the recorded high-dimensional data [36, 46, 47, 48].

In this work, a window-based spike sorting hardware architecture using System-on-Chip (SoC) FPGA is presented for real-time processing of high-channel-count neural data recorded with a dense electrode array. The incoming spikes are observed

on each channel in parallel. The sorting part of the proposed system is window-based, because it takes into consideration the spatial information in a window of the electrode array. To perform spike sorting in this window-based manner, the Online Sorting (OSort) algorithm [49] was modified. OSort is an unsupervised template matching algorithm, originally designed to process single-channel recordings. The functionality of the system was verified using the hybrid ground truth signal generator (Section 2.4) and in vivo measurement data [26].

3.2 Related Works

In the literature several FPGA-based solutions can be found, which were developed for spike sorting not suited to process data recorded with high-channel-count neural probes [15, 36, 41, 43, 46, 50], while [44], [51, 52, 53] capable of sorting action potentials recorded with high spatial resolution probes [54, 55]. A detailed comparison to the proposed system is presented in Section 3.3.5.

3.3 Spatial information based OSort for real-time spike sorting using FPGA

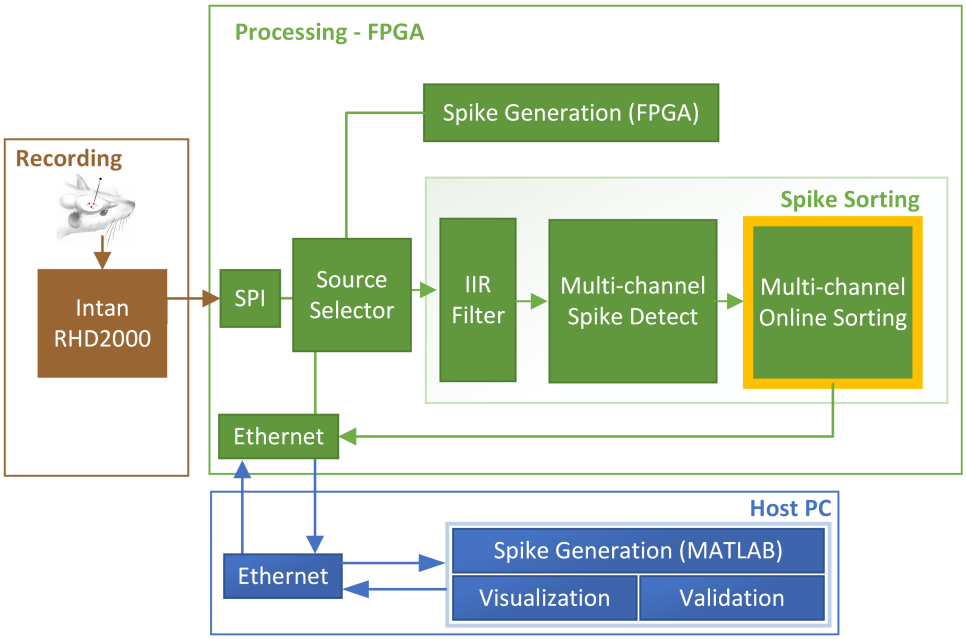

Figure 3.1: *Schematic diagram of the proposed spike sorting system. The Multi-channel Online Sorting part is emphasized.*

In Fig. 3.1 the schematic diagram of the proposed system can be seen, where the Multi-channel Online Sorting part is emphasized and detailed in this section. The Mullti-channel Online Sorting part is based on the proposed spatial window-based Online Sorting algorithm.

Spatial window-based Online Sorting

This algorithm extends the original OSort algorithm with a cluster memory, which is required for the optimal FPGA implementation. The flow diagram of the proposed algorithm can be seen in Fig. 3.2.

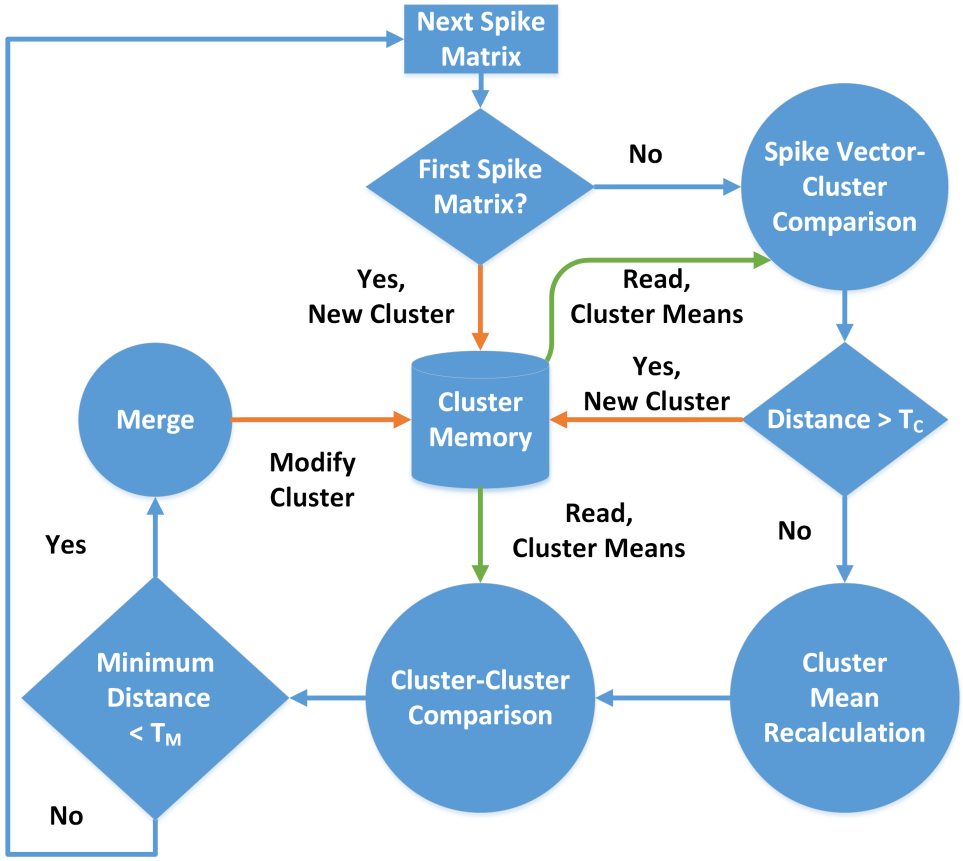

Figure 3.2: *Flow diagram of the spatial window-based Online Sorting algorithm*

The clustering and cluster merging thresholds (T_C and T_M respectively) are automatically determined and adaptively changed during the classification based on the STD value calculated in the spike detection part. The threshold value of one channel is calculated based on the standard deviation of a five seconds long moving window (determined experimentally) from the neural recording ($signal$) as follows.

$$T_C = T_M = std(signal)^2 \cdot c_C \cdot N_S, \tag{3.1}$$

where c_C is the clustering correction factor [49], and N_S is the number of sample points in a spike.

When the *Multi-Channel Spike Detect* core sends the first spike matrix containing 9 spikes from the 3×3 electrode window, it is stored as the first cluster in the cluster memory. The next spike matrix will be compared to the saved cluster mean using squared difference as the distance metric. If this spike matrix is similar to the already saved one, then the calculated distance is below T_C, so it is assigned to this cluster. If it is not similar, and the distance is above T_C, then the creation of a new cluster is required. This process is applied to the subsequent incoming spike matrices. After the assignment the mean of this cluster will be updated, because the composition of the cluster is changed. Furthermore, the cluster mean update changes the distance between cluster means, therefore a distance check between clusters is needed. If a distance is below T_M, then the updated and the closest cluster will be merged together forming a new, larger cluster. In this case the smaller cluster will be removed from the memory, and the spike matrices from the smaller are assigned to the larger cluster.

Spike Generation

The algorithm and the implementation of the Spike Generation module is fully discussed in Section 2.4. Using ground truth data, the time corresponding to spikes of individual neurons is known and thereby it allows the quantitative assessment of the performance of the spatial window-based OSort spike sorting algorithm. Since in vivo neural data (obtained by direct measurement) and synthetic data (background noise, position and firing time of neurons) were combined to generate the dataset for validation, this is referred to as hybrid ground truth. Spike templates were created from *in vivo* electrophysiological data recorded with the previously mentioned, 128-channel silicon probes from the neocortex of anesthetized rats [26]. The spikes of well-separated units (n = 72) were averaged and the average spike waveforms were used for the construction of templates. Since the high-density probe could record the spikes of a particular unit on multiple, adjacent recording sites, only the recording channel on which the spike appeared with the largest peak-to-peak amplitude was used for template construction. A spike template was represented with 64 sample points/channel (3.2 ms). For a higher variability of spike templates, spike waveforms corresponding to both putative neocortical principal cells (wide spikes, n = 58) and putative neocortical interneurons (narrow spikes, n = 13) were selected [56].

High-density neural probes provide high spatial resolution, that is, the spikes of neurons can be recorded on multiple electrodes simultaneously. The extent of spatial spreading of the spike waveform of a neuron depends on several factors (e.g. neuron-electrode distance, type of the neuron) but is usually a few tens of microns. Therefore, the maximal radius of spatial spreading of a template spike waveforms

was six electrodes ($132.5\mu m$) in each direction with the mean spike waveform located in the center.

To generate a more realistic neural dataset the spatial spread of the spike waveforms of 160 in vivo recorded cortical neurons were examined (obtained with the 128-channel silicon probe used in this study) to extract the spreading patterns. The Spike Generation module (Section 2.4.) was modified based on the found spatial patterns, so in the modified version each simulated neuron has an individual and asymmetrical spike waveform spreading.

3.3.1 FPGA architecture

The *Spike Sorting Module* consists of the *Multi-channel Online Sorting* core and the *Multi-channel Spike Detection* core. The original version of the OSort algorithm [49] works with a single-channel. To process data from a window of the electrode array, the algorithm was modified. The data flow and the structure of the original algorithm are completely redesigned. The neural data are processed in 3×3 spatial windows, so the clustering algorithm works on 9 selected channels at a time.

Window-based Online Sorting

In the *Multi-channel Online Sorting* core a cluster mean consists of $3 \times 3 \times 64$ data points, which are represented as 16 bit integers. The clusters are represented as fixed point type numbers, all of them are 18 bit with 2 bit fractional part. The single-channel OSort processes spikes with 64 samples using double precision. The memory required for cluster mean storage are less if integer values are used. The T_C classification threshold values are in the range of 10^5, therefore rounding the values does not change the result of the comparison.

In this implementation for the calculation of the T_C classification threshold the c_C clustering correction factor was determined experimentally and set to 0.4 for all simulated data and 0.1 for the in vivo recording, while the number of sample points N_S were 64 for both cases.

In this case a 18 bit wide and 2048 element deep configuration of Xilinx 7 series BRAMs are used, therefore, 32 clusters can be stored in 9 BRAMs. The number of clusters in the proposed system (32×4 electrode array) are expected to be around 100. Therefore the maximal number of clusters is 128 using 36 BRAMs. The computation of the *Multi-channel Online Sorting* core uses an array of processing elements with 9 (3×3) Arithmetic Units.

The architecture of *Multi-channel Online Sorting* core is shown in Fig. 3.3. Spike data is received from the *Multi-channel Spike Detect* core via AXI-Stream buses and go through a deserializer and a 14 spike matrices deep buffer to handle multiple detections. During a 0.1 ms period (the processing and classification time for one

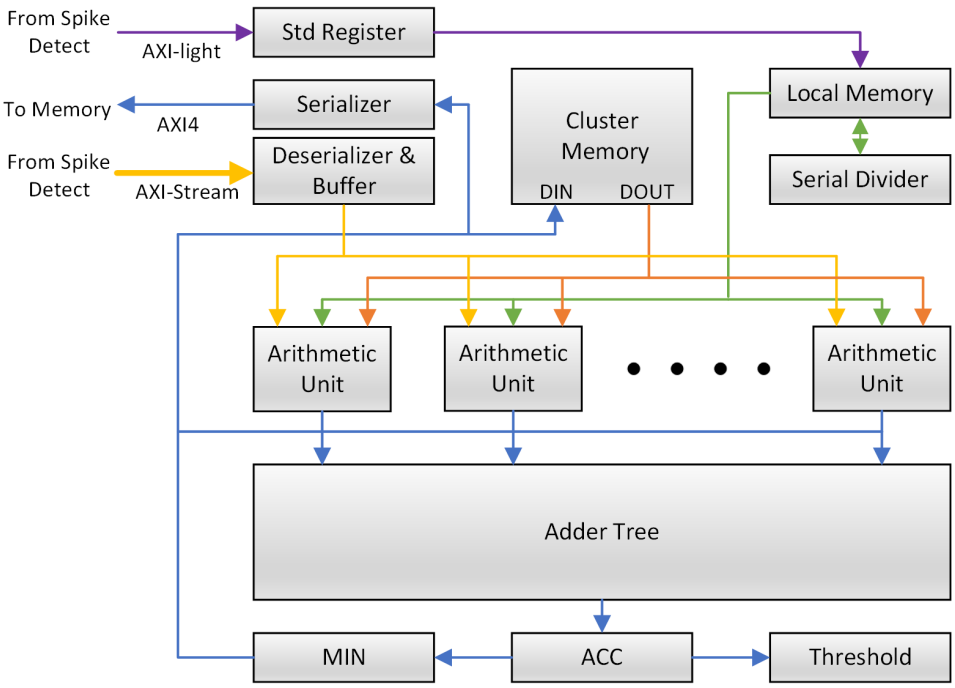

Figure 3.3: *Architecture of the Multi-channel Online Sorting core*

spike matrix is 85.17 μs, only one neuron will fire on average in the vicinity of a recording site (or near a 3 x 3 window of sites). Therefore, on 128 channels 14 neurons can fire in average in a 0.1 ms period, so a system capable of storing 14 spike matrices will cover most of the cases, even when multiple neurons fire in a short time frame [57].

There are 4 stages, which function is to load the ($3 \times 3 \times 64$ samples) spike matrix and compare it to the cluster means in the Cluster Memory (1), update the chosen cluster mean or create a new cluster (2), compare the cluster means to each other (3) and finally merge clusters (4).

In the first stage of the OSort algorithm spike data for each channel is loaded in parallel through the AXI-Stream bus and the corresponding mean values of the first cluster is provided by the Cluster Memory. Partial results of the summed squared difference is computed by the Arithmetic Units. The partial results are summed by the Adder Tree for each sample and an accumulator (ACC) is used to compute the sum over the entire 64 elements sample window. The result and the cluster number is saved into the MIN register if it is smaller than the previous minimum squared distance. These steps should be executed for each living cluster in the Cluster Memory.

If the distance of the current spike between all clusters is computed and the value in the MIN register is larger than the T_C clustering threshold then the spike data is reloaded and saved into the Cluster Memory as a new cluster. If the distance is smaller than T_C the spike should be merged with the closest cluster. The weighted

average of the cluster mean and the new spike is computed by the Arithmetic Units and the results are saved into the Cluster Memory and also to the spike memory. The weights which can be pre-computed by the serial divider and stored in the local memory are $n/(n+1)$ and $1/(n+1)$ for the cluster mean and the new spike respectively, where n is the number of spikes in the cluster. The new number of spikes $(n+1)$ and the new weights $((n+1)/(n+2), 1/(n+2))$ is computed and stored in the local memory.

In the third stage the cluster means, which updated previously, is loaded from the spike memory and its distance is computed for all clusters similarly to stage one. When the computed minimum distance is smaller than the T_M merging threshold the mean values of the two clusters are merged in stage four similarly to stage two. If the recently updated cluster A has n elements and the closest cluster B has m elements then the weights are $n/(n+m)$ and $m/(n+m)$, which can be pre-computed when the distance of the two clusters is computed in stage three. Cluster A marked as unused and merged in a table held in the local memory and the new number of spikes in cluster B is computed $(n+m)$. The new weights required in stage two $((n+m+1)/(n+m+2), 1/(n+m+2))$ are also pre-computed. The system halts until a new spike is detected and computation is restarted from stage one.

3.3.2 Implementation Results

The proposed *Spike Sorting* and *Processing* blocks were developed using Vivado HLS 2018.3. The prototype MATLAB algorithms are translated to a High Level Synthesis (HLS) based C/C++ solution taken into consideration the architecture described in Section 3.3.1. The *Processing* block was implemented on a Xilinx ZCU106 SoC FPGA board, which contains a Zynq UltraScale+ XCZU7EV FPGA as PL and a quad-core ARM Cortex-A53 processor as PS.

During the overall validation process, the system was tested with varying number of neurons and noise levels using the hybrid ground truth generator and in vivo cortical measurements. The results were evaluated and visualized on the host computer in MATLAB.

In case of the *Multi-channel Online Sorting* core, the spike matrix input is loaded trough 9 AXI-Stream buses, the output for the clustering result are mapped to an AXI-Stream bus, while the standard deviation value used in the calculation of the clustering threshold can be updated using an AXI-Light connection.

The available resources on the XCZU7EV FPGA and the resource requirements of the cores can be seen in Table 3.1, which shows that the main resource consumption is the memory requirements (LUT, BRAM). The LUT resource utilization can be reduced if more BRAMs are used instead of LUTRAMs or URAMs can be utilized.

The results of the test measurements on the XCZU7EV FPGA showed that spike

Table 3.1: *Area requirements and device utilization of the synthesized overall system*

	Device Utilization			
	FF	LUT	DSP	BRAM
M. Spike Detect Core	10915	42692	6	7
W. OSort Core	6569	8982	54	91
Summarized	17484	51674	60	98
Utilization(%)	3.79%	22.42%	3.47%	15.74%

matrices can be clustered in $18,005$ clock cycles. The latencies of the synthesized cores, which are capable of operating on $200MHz$ clock frequency can be seen in Table 3.2. The latency is given in clock cycles, while trip count is the minimum number of times a loop executes. The latencies are grouped by the *Multi-channel Spike Detect* and *Multi-channel Online Sorting* cores. The STD block is shown separately, because it can run independently of the detection or the classification.

Due to the relatively low operand bit width, the system can operate on 200 MHz clock frequency, so a spike can be detected, realigned and selected in 423 clock cycles, which is 2.12 μs. One spike matrix ($3 \times 3 \times 64$ data points) can be clustered in 17034 clock cycles in the worst case, resulting in 85.17 μs clustering time. Altogether the detection and classification can be done in 87.29 μs, therefore the proposed system can process up to 11456 $spike/s$.

The spike sorting was also tested offline with the same dataset on the Host PC in MATLAB and the results showed that the average computation time is 7202 μs. Therefore the XCZU7EV FPGA-based system is around 80 times faster.

3.3.3 Validation Results

The verification of the classification performance of the proposed system contains multiple datasets with different average Signal-to-Noise Ratios (SNR with $3-10dB$) and different numbers of neurons ($4-32$), since using the 128-channel neural probe an average of 30 neurons can be separated from a single cortical recording location [57]. The average SNR is calculated using the same method as in [49]. The duration of a dataset was 300 seconds with $20,000$ samples/second/channel on a 32×4 electrode array (128 channels). Spike templates corresponding to different neurons were added to arbitrarily chosen channels at random time points. Spike templates were selected randomly from the template database [58] with 80-20 ratio between pyramidal cells and interneurons.

To measure the accuracy of the clustering, spike times of the template neurons

Table 3.2: *Detailed latency of the Spike Sorting architecture*

	Latency (clock cycles)	Trip Count (clock cycles)
Spike Sorting Module		
Load Spike	193	192
Stages 1-4	16712	16704
Read Merge Table	129	128
Total	17034	17024
Spike Detection Module		
Load Channel	129	128
NEO	131	128
Realign	163	154
Total	423	410
STD	12.8 M	12.8 M

and the spike times of the created clusters are matched. Only the spikes placed in the appropriate (best match) cluster are taken into account.

The average and standard deviation of the classification efficiency can be seen in Table 3.3, in case of 128 channels using 300 seconds neural dataset with varying SNR and neuron numbers. In most cases the accuracy is between 80%-90%, except for low neuron numbers and low SNR, where the proposed method has low accuracy. Increasing the number of neurons the accuracy is also increasing for all SNRs.

In case of low neuron numbers on a high-channel count electrode array (128 channels) only a small part of the channels contain spikes and these channels are also used in the STD calculation making the clustering threshold lower, so in case of multi-channel clustering the spike matrices do not meet the clustering condition and falsely new clusters are created.

To visualize the performance of the proposed system a specific table layout is used in which each row represents a spike template of a neuron, while each column represents a sorted cluster and the fields of the matrix contain the number of spikes. A matching matrix for 8 simulated neurons can be seen in Fig. 3.4, where, for example, the cluster identified with *1* has 442 spikes from spike template *7* . The column named *undetermined* includes those (usually noisy) spikes, which were in clusters discarded due to low number of spikes (< 10).

Table 3.3: *Classification Accuracy (%) of the Proposed System*

Neurons	Average SNR (dB)				
	10	8	6	4	3
4	88 ± 9	87 ± 10	59 ± 26	27 ± 43	16 ± 23
8	94 ± 3	93 ± 3	84 ± 13	57 ± 38	28 ± 28
12	92 ± 2	92 ± 1	87 ± 2	59 ± 17	21 ± 18
16	92 ± 3	91 ± 2	88 ± 2	69 ± 9	21 ± 14
20	89 ± 3	90 ± 3	87 ± 3	81 ± 5	39 ± 27
24	90 ± 1	90 ± 1	88 ± 2	80 ± 3	55 ± 12
28	89 ± 3	89 ± 2	87 ± 2	82 ± 7	62 ± 16
32	86 ± 3	86 ± 3	85 ± 2	81 ± 6	64 ± 8

The accuracy of the proposed system using multi-channel spike detect with only single-channel clustering was evaluated. In Table 3.4 the results can be seen, where gray cells denote lower accuracy compared to the proposed system. The results show that the proposed system achieves better performance in most cases, except some low neuron and SNR configurations. Recordings with the 128-channel probe typically contains neural activity from 30-40 neurons in average [57], therefore using the proposed system with higher neuron numbers (e.g. 32), a better performance can be achieved than single-channel clustering. In cases above 3 dB SNR the difference between the proposed method and the single-channel version is significant (Student's t-test, $p = 0.002$).

A 5-minute-long in vivo cortical dataset obtained with a 128-channel silicon probe from an anesthetized rat was used for validation. After using the kiloSort [24] spike sorting algorithm offline the resulting clusters were manually curated defining 26 well-separated single unit clusters [26]. Running the proposed system on this cortical recording, 32 clusters were created, which showed 80% similarity using cross-correlation between the results obtained with the method described above and with the proposed system.

The results of the spike sorting can be seen in Fig. 3.5, which contains some example clusters from two distinct channels.

Neuron ID \ Cluster ID	1	2	3	4	5	6	7	8	Undet.
7	442	0	0	0	0	0	0	0	43
6	0	2342	0	1	0	1	0	0	152
4	0	0	462	0	0	0	0	0	60
1	0	1	0	1598	0	0	0	0	178
2	0	0	0	0	833	0	0	0	115
3	0	1	0	0	0	1824	0	0	116
5	0	0	0	0	0	0	293	0	28
8	0	0	0	0	0	0	0	284	28

Figure 3.4: *Matching matrix for 8 simulated neurons using the proposed system*

Table 3.4: *Classification Accuracy (%) using Multi-channel Spike Detect and Single-channel Clustering*

Neurons	Average SNR (dB)				
	10	8	6	4	3
4	64 ± 4	64 ± 4	66 ± 5	64 ± 6	62 ± 5
8	79 ± 4	79 ± 3	78 ± 2	75 ± 2	70 ± 4
12	78 ± 5	77 ± 5	77 ± 4	71 ± 4	69 ± 6
16	78 ± 6	78 ± 6	77 ± 6	74 ± 6	67 ± 5
20	76 ± 7	78 ± 4	75 ± 7	69 ± 7	60 ± 9
24	75 ± 4	75 ± 3	71 ± 4	65 ± 7	54 ± 8
28	80 ± 6	80 ± 5	77 ± 6	70 ± 3	61 ± 5
32	75 ± 3	73 ± 3	72 ± 1	65 ± 2	57 ± 4

3.3.4 Discussion

Besides the spatial information based online spike sorting, the proposed FPGA-assisted system can support *in vivo* experiments by determining the position of electrodes containing spiking activity in real-time, even shortly after the implantation or the relocation of high-density neural probes. Furthermore, by obtaining various firing properties (autocorrelogram, firing rate, spike width, spike shape, location on the electrode, etc.) and the waveform of the single units extracted in real-time, neural activity in brain areas under examination could be surveyed in a short time, which

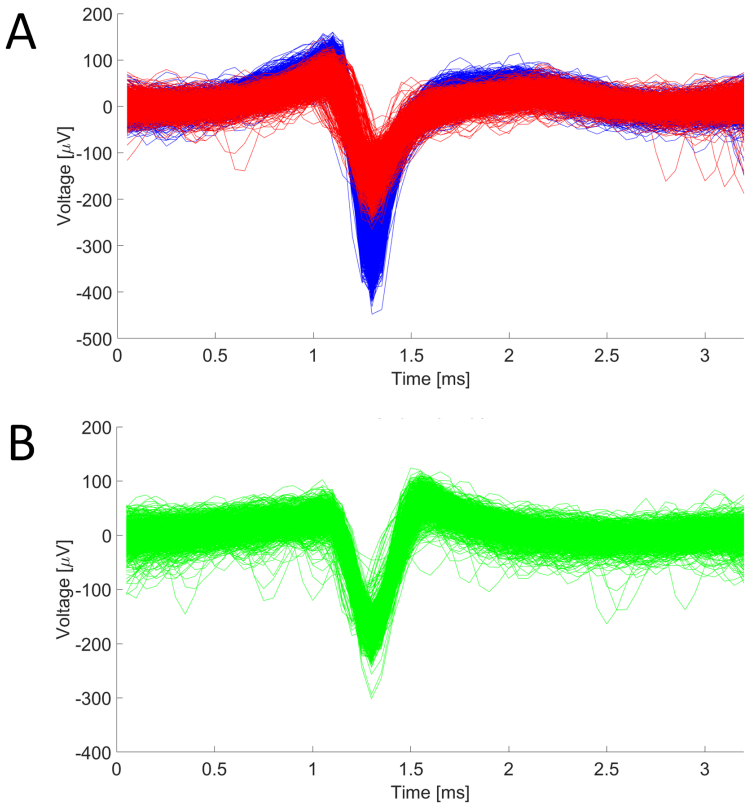

Figure 3.5: *Partial results of the proposed system on the 5-minute-long in vivo cortical dataset from an anesthetized rat [26]. (A) shows 2 clusters on channel 12. (B) shows a cluster on channel 109.*

presumably would be a useful aid for neuroscientists.

The proposed system is capable of clustering more than $11,000$ spikes in a second, therefore real-time spike sorting is possible. Based on the latencies and the resource utilization (Table. 3.1), even 1024 clusters and channels (8 times more) could be used.

3.3.5 Comparison to other systems

The original OSort algorithm [49] was compared to the proposed system using the hybrid ground truth signal generator [58]. Altogether six neurons were simulated on channels (two neurons on each) 33, 42, and 43. The original OSort algorithm was running in parallel on 128 channels to be comparable to the proposed system. The proposed system created only 2 clusters on each appropriate channel (F-Score 100% [59]), while the original OSort (with the default parameters) also found the same 6 clusters on the appropriate channels, but made another 735 false clusters too (F-Score 0.8%) from noisy, overlapping spikes and due to the crosstalk between

channels.

In addition to the original software solution described above many FPGA-based spike sorting algorithms can be found in the literature (see Section 3.2) but only three of them [36, 51, 52] are using a version of the original OSort, and only one OSort-based solution is capable of multi-channel classification [52].

In [36] the original OSort algorithm was implemented on FPGA, but it cannot be used for multi-channel spike sorting, and the system can only process single-channel neural recordings containing only 3 neurons with the accuracy of 89%.

A real-time unsupervised FPGA-based spike sorting system is presented in [51], which applies NEO for spike detection and a parallel architecture based on the OSort algorithm for classification. The presented system works only on single-channel measurements and was tested on the Easy1_noise01, Easy2_noise005, and the Difficult1_noise005 data containing only 3 neurons from the WaveClus dataset [60] achieving F-Score accuracy of 94.93%, 96.94%, and 91.50% respectively.

In [52] a real-time template matching multi-channel spike sorting system is presented. In this system also a NEO-based spike detection is applied, but the OSort algorithm is used only to define the most commonly occurring waveforms offline. These predetermined waveforms (only three different types) are stored in a template memory. During the classification the detected spikes are compared to these templates, so this solution is not fully unsupervised. Furthermore in this spike sorting system the multi-channel operation means only that each channel processed individually and does not take into account the spatial correlation between the channels. The Easy1_Noise and Difficult1_Noise005 data containing only 3 neurons were used from the WaveClus dataset [60] for single-channel simulations to test this system achieving F-Score accuracy of 93.3% and 93.9% respectively. The proposed system is processing only multi-channel recordings, therefore applying single-channel simulations from the WaveClus dataset is pointless.

3.3.6 Scalability and Limitations

The scalability of the system depends on the number of channels in the electrode array (in this case 128) and on the spatial window size used in the classification. In the proposed system the spatial window size is fixed (3×3), therefore only the number of channels can be changed, which impacts the maximum number of clusters. The processable number of clusters can be further increased using more BRAMs or URAMs. Creating more clusters increases the processing time, therefore more window-based OSort Cores can be utilized in parallel to maintain the same classification speed. Based on the resource requirement of the system even 4 window-based OSort Core can be implemented on the XCZU7EV FPGA (Table 3.1).

In the actual state of the proposed system every 3×3 spike matrix is compared

with every cluster mean, even if it comes from a completely different electrode channel. To address this issue in the future the electrode coordinates can be checked during the classification, which requires only an additional 1-byte information (channel number) per cluster.

3.4 Discussion and concluding remarks

In this work an FPGA-based implementation of the OSort algorithm for unsupervised online window-based spike sorting system is presented. The results show that the architecture can be implemented on a mid-range SoC FPGA device running on 200MHz, which is capable of the detection and classification of the incoming spikes on 3×3 spatial windows in 87.29μs, or more than 11000 spikes/s above 80% classification accuracy for 32 simulated neurons with 4-10 dB SNR.

Furthermore it can be concluded that the neural spike sorting on the FPGA is 80 times faster than the identical algorithm running offline with the same dataset on the Host PC in MATLAB.

The proposed architecture is currently capable of sorting 128 channels of neural data in 3×3 spatial windows in real-time using the SoC FPGA. Based on the validation results it can be concluded that the clustering efficiency of the system can reach an average accuracy of 86% for high neuron numbers (16-32) above 3 dB SNR, while the single-channel clustering version achieves 74% average accuracy in the same cases.

The proposed system was tested with generated neural data, and real data recorded with the Intan measurement system.

3.5 Contributions

The author of this PhD thesis is responsible for all the contributions presented in this chapter, which include the following main novelties:

II / 1. I proposed the use of inter-channel spatial information in multi-channel neurophysiological recordings using the Online Sorting algorithm. I compared the efficiency of the original single-channel Online Sorting algorithm to the proposed algorithm on synthetic data. I showed that the proposed algorithm, which uses spatial information has better efficiency.

II / 2. I gave an FPGA architecture based on the Online Sorting algorithm capable of classifying multi-channel neurophysiological recordings in real-time. I showed the effectiveness of the architecture with tests based on synthetic data and real measurements. I showed the FPGA architecture can operate in real time and is suitable for immediate feedback during experiments.

Chapter 4

FPGA implementations in the field of signal processing and simulation

Signal processing is inevitably used in the research fields of computer science. It is important part of artificial intelligence, control theory, simulation, pattern recognition and these fields nowadays are becoming more and more overlapping or inseparable. Some of the current signal processing algorithms can be used in embedded systems, which can also be applied in real-time applications, if the requirements are fulfilled. These requirements on the software part are parallel optimization capability, low memory usage and low processing bandwidth. On the hardware requirement part there is low power consumption and small form-factor.

Algorithms that fulfill these software requirements could be accelerated to be used in real-time applications using FPGAs. Furthermore using FPGAs apparently provide small form-factor and low power consumption and the possibility to further improved into ASICs.

In this chapter some signal processing algorithms from different computer science research fields are presented, which can be modified to be accelerated with FPGAs for real-time applications. These applications are face recognition, sensor fusion, suspension control and hardware-in-the-loop (HIL) simulation.

In Section 4.1 and Section 4.2 summarizes the related works and briefly presents previous results. The FPGA-based low-cost real-time face recognition is covered in Section 4.3. In Section 4.4 a real-time pose estimation algorithm based on FPGA and sensor fusion is presented. Result and final thoughts are summarized in Section 4.8. An implementation of an FPGA-based actual observer for active suspension control can be seen in Section 4.5. In Section 4.6 the implementation of an FPGA-based wind turbine HIL model is presented. In Section 4.7 the discussion and concluding remarks can be seen. The contributions are presented in Section 4.8.

4.1 Introduction

4.1.1 Face recognition

In the recent years automated person identification became a widely researched area, because entry control and security are playing a central role in the industry. Furthermore, public places are exposed to criminal activity, therefore eliminating threat sources from frequently crowded places also requires the recognition of known criminals. Face recognition is a well applicable solution to person identification, and has many applications in surveillance, access management, law enforcement and biometrics. In general the number of persons to be identified are low, but there are places e.g. bus stations, airports where a high number of faces need to be identified in a very short amount of time. In the identification process it is required to compare the face of the person to the faces in the database. Therefore the recognition time depends on the number and size of the stored faces. In order to identify a person in real-time the usage of a specialized hardware is necessary.

4.1.2 Pose estimation and sensor fusion

In mobile vehicle localization the robust, real-time and power efficient pose estimation plays a central role. The estimation of the position and orientation (pose) can be solved with different sensors and methods. The most frequently used sensors are the accelerometer and gyroscope, because they are cheap, small and available in a compact version called Inertial Measurement Unit (IMU), which often contains a magnetometer in addition. Only IMU based pose estimation is possible, but the estimation error accumulates over time, leading to inaccurate results on long-term. Therefore another different source of information is required, which could be a GPS, a LiDAR, a mono or stereo camera (computer vision), etc. Mounting a camera on a vehicle and using images as an information source requires the deployment of computer vision methods, e.g. visual odometry or pattern matching. Therefore, when using multiple information sources the different sampling interval of the sensors, the processing delay of each device and the type of measurements need attention. One solution could be to combine the advantages of different sensors and reducing the disadvantages, which is called sensor fusion. There are several methods for fusing the measurements together. One of these is the widely used Kalman filter, which is capable of combining different measurements using state estimation [61, 62, 63].

4.1.3 Suspension control

The most significant part, which affects the handling performance and comfortability of a vehicle is the suspension system. A traditional suspension system consists of

springs and dampers to decrease road noise, bump effects and vibrations. There are three types of suspension systems, which are the passive, the semi-active and the active ones. Nowadays vehicles use passive suspension system, which consists of springs, that can store energy and dampers for energy dissipation. In a passive suspension system the spring and damper coefficients are constants, and can not adapt to the road surface in all circumstances. In addition to the passive type a semi-active suspension system is capable of controlling the damping coefficient. It can offer an improvement in the dynamic performance of the vehicle, and more effective than the passive suspension, but unfortunately can not handle the high frequency road noise. The Electro Rheological (ER) and Magneto Rheological (MR) fluid dampers are two types of semi-active suspension systems [64]. As the sensor, signal processor and electro-hydraulic component technology upgraded the active suspension system was born. An active suspension system uses actuators to add force, rather than dissipating energy. It can create a restoring force to maintain the steadiness of the vehicle, dampen the vibration and avoid the lifting of the wheels. Based on the sensor values the control algorithm controls the additional force to pull down or to push up the body masses, thus it can compensate the disturbances of the road. The drawback is the actuation energy needed for the compensation, because an external energy source is required, which increases the system cost. It is proved, that an active suspension system can maintain good comfort and handling performance in various conditions [65].

4.1.4 Wind turbine modelling

Renewable energy systems are becoming more and more important as non-renewable energy sources run out. The most widely used renewable energy systems are wind turbine, hydropower, hydroelectric, photovoltaics, concentrated solar power, biomass, biofuel and geothermal power. According to [66] about 337 GW wind power generation capacity has been set up in the worldin June 2014, which indicates a growing trend comparing to the previous years. Nowadays wind energy is used in more than 100 countries as an electrical power source. Due to this, wind energy can play the most significant role to reduce the effect of global warming. Wind energy became a mainstream electric energy resource. Usually, a wind turbine consists of a rotor with blades, a shaft and gearbox, and a generator. Wind turbines convert the kinetic energy of the wind into electrical energy and wind farms generate electricity for public service.

4.2 Related Works

4.2.1 Face recognition

In the literature various software-based approaches can be found for face recognition. In [67] the applications of PCA are discussed in vision based computing. The results showed, that PCA based dimension reduction and image classification has a potential for industrialization, but it requires large amount of computation. A new two-dimensional technique for PCA is shown in [68], where PCA is based on 2D image matrices, rather than 1D. It can be concluded, that 2DPCA is more efficient, than simple PCA, but it requires more memory. A face recognition algorithm based on a modular PCA approach is presented in [69]. The modular approach performs better in various illumination and expression conditions, but the computational cost is increased. Although the software-based PCA face recognition implementations are capable of accurate classification, but their real-time usage is not possible.

Field-Programmable Gate Arrays (FPGA) are applied to improve computational efficiency with hardware parallelism, therefore a high level of acceleration is possible. Reading the literature of FPGA-based face recognition several approaches can be found. In [70] a real-time PCA-based portable emotion detection system is proposed, which is capable of helping autistic children understand facial emotions of other people. The proposed architecture uses an optimized eigen calculation with power-deflation iteration method, and implemented on a Virtex-7 FPGA, and the results showed a 82.3% detection accuracy. An FPGA-based neural network, that capable of PCA computation is presented in [71]. The algorithm was implemented on a Virtex-2 FPGA, and this architecture can recognize 1400 faces in an image frame, which makes it suitable for real-time face recognition. In [72] an FPGA-based face recognition architecture is proposed, which performs PCA on wavelet transformed images. The architecture is running on a Virtex-2 Pro with 46.79 MHz and capable of 100% classification accuracy with the usage of 40 eigenvectors/face.

4.2.2 Pose estimation with sensor fusion

In the literature of FPGA-based sensor fusion several research can be found. In [73] the hardware implementation of a Kalman filter based sensor fusion technique is applied to ultrasound and infrared sensors, for estimating the distance, using a CYCLONE IV FPGA, which is capable of running with 58 MHz and reaching a speedup factor of 514 in comparison with an embedded software implementation. The authors in [74] present the design and implementation of a high performance Cyclone II FPGA hardware with low power consumption that computes optical flow in real-time running on 50 MHz. An FPGA based circuit for angular position estimation using a complementary filter with the measurements of an inertial measurement unit

(IMU) is presented in [75]. The complementary filter combines the data measured by the sensors of the IMU (3DoF accelerometer and the 3DoF gyroscope), and it is capable of running real-time on a XC7Z010 FPGA at 100 MHz. In [76] a Xilinx Zynq 7020 FPGA hardware accelerator for a Kalman filter based sensor fusion focusing on low power consumption is presented. The proposed architecture lowers the energy consumption by a factor of 1.14 compared to the ARM Cortex A9 processor. Based on the results it can be concluded that the Xilinx Zynq 7020 power dissipation with the presented architecture fulfils the power consumption requirements for mobile usability. The authors of [77] present a system for the alignment of sensors using inertial measurement devices as the basis for developing a range of dynamic real-time sensor fusion applications. The proof of concept utilizes a COTS FPGA platform for Kalman-filter based sensor fusion and real-time correction of a misaligned video stream using a custom-designed 32-bit soft processor core. A visual-inertial sensor unit with real-time Simultaneous Localization and Mapping (SLAM) capabilities is presented in [78]. Up to four cameras are interfaced through a Z-7020 FPGA system, along with an Inertial Measurement Unit (IMU) providing gyroscope and accelerometer measurements, which are tightly fused together for robust estimation. In [79] an orientation tracking system based on a double stage Kalman filter for sensor fusion with a 9DoF IMU is presented. The Kalman filter was divided into two stages for complexity reduction. The gyroscope data are used to first estimate the angular position, then the first stage corrects roll and pitch angles using accelerometer data. The second stage processes magnetic compass data to correct the yaw angle. The authors of [80] present an implementation of the Extended Kalman-filter targeting an embedded system based on an Altera Nios II FPGA device at 50 MHz. The results show that the proposed system is applicable for indoor robotic applications.

4.2.3 Active suspension control

Reviewing the literature of active suspension system control various approaches can be found, such as sliding [81], Linear-Quadratic Regulator (LQR) [82], and PID control [83]. Although all of them is capable of an effective active suspension control, in this work the LQR based controller is used, because it has easier implementation and has similar efficiency. In [81] the road profile is estimated via a disturbance observer, and the active suspension system is controlled by a sliding mode control algorithm. A sliding mode controller and a sliding mode observer is used together in [84]. The observer can detect and isolate the failure of sensors, because every fault has its own pattern, and can be recognized. Due to the great flexibility, high computational capacity and boundless reconfiguration possibilities of Field Programmable Gate Arrays (FPGA), FPGAs can be used to easily implement various types of controlling algorithms. In the literature of FPGA-based active suspension control some approaches

can be found. In [85] a Piece-Wise Multilinear Fuzzy Inference System (PWM-FIS) controlled semi-active suspension system is presented. It is proven that the FPGA architecture is capable of real-time advanced control of any suspension system. An Adaptive Neuro Fuzzy Inference System (ANFIS) is used for controlling a nonlinear active suspension system in [86]. Very High speed integrated circuit Hardware Description Language (VHDL) is used for the implementation, and offline training is needed for the controller. It provides an efficient and robust hardware implementation of the control algorithm. Based on the literature review of Field Programmable Gate Array (FPGA) based observers some approaches can be found. In [87] a linear observer is designed and implemented on a Virtex-5 FPGA, while in [88] the design and implementation of a sliding-mode observer is presented. Currently in the literature active suspension system control through FPGA-based observer can not be found, so it is reasonable to implement an actual observer on an FPGA for real-time observer based control.

4.2.4 Wind Turbine modelling

Reviewing the literature of modelling wind turbine systems, many various wind turbine, shaft and gearbox, and generator models have been analyzed. A fixed-pitch angle wind turbine simulator was presented in [89], which was built in MATLAB/Simulink. The goal of the simulator was to verify the effectiveness of the fixed-pitch angle wind turbine model with an induction generator when the applied rotational speed was higher than the nominal. A fix-speed wind turbine model was introduced in [90], including the model of the aerodynamic, mechanical and electrical components of the turbine. Using this model a wind farm was developed in Power System Computer Aided Design/Electromagnetic Transients including Direct Current (PSCAD/EMTDC) to study the operation and power grid integration issues of wind turbines. In [91] a comparative study of various methods for mathematically modelling wind turbines was presented. In this work the wind turbine was modelled based on the fundamental equations of wind power and a presumed shape of the power curve. The first model does not correctly substitute the physical wind turbine, and neither the second model can reach the desired accuracy for lower annual average wind speeds. A review of commonly used Variable Speed Wind Turbine (VSWT) power curve equations were presented in [92] from the data of 200 commercial VSWTs ranging from 225 to 7500 kW. In [93] a model used for representing all types of VSWTs in power system dynamics simulations was presented. The Power System Simulator for Engineering (PSS/E) software package was used to simulate the model and compare the results with real measurements. Results showed that the simulation and the measurements were correlating. In the literature of Field-Programmable Gate Array (FPGA) based Hardware-In-the-Loop (HIL) systems nu-

merous applications have been made, but none of them dealt with wind turbine model implementation. Examples for FPGA based HIL applications implemented in LabVIEW were presented in [94] where HIL simulation and model validation of thermocouples,linear variable differential transformers and resolvers were made. Results showed that high-performance prototypes and test systems can be developed easily, without prior experience in FPGA hardware design. In [95] a dynamic but not HIL system model of Doubly Fed Induction Generator (DFIG) wind turbine system was implemented on an FPGA. The FPGA-based and the MATLAB/Simulink simulation results were compared to each other. The results showed that significant simulation speed-up can be reached by using FPGA circuits.

4.3 FPGA-based low-cost real-time face recognition

A widely used approach for face recognition is the Eigenfaces approach, which means the usage Principal Component Analysis (PCA) on a database containing human faces. Due to the dimension reduction of the features the calculation can be faster and the implementation is easier than other face recognition approaches. It is concluded, that the Eigenfaces is tolerant to small rotation, translation and scale changes [96].

In this work a low-cost FPGA-based architecture for face recognition is presented, which is capable of real-time classification of human faces using Eigenfaces, also known as the PCA algorithm. The proposed face recognition architecture is the part of an automated person identification system, which enables real-time surveillance and threat monitoring.

4.3.1 Eigenfaces method

The main idea behind the Eigenfaces method is to use only the most relevant information possible from the images of human faces and make a comparison based on those features between the faces. Using the variation of face images for the comparison base is a simple, but efficient approach. Mathematically that means the covariance calculation of the human face images, and the computation of the eigenvectors based on the resulting covariance matrix. The eigenvectors are the features, that together describe the face images and can be displayed as a ghostly face image, called the eigenface. The eigenvectors characterize the variation between the face images, therefore each human face can be reconstructed by a linear combination of the corresponding eigenvectors. The eigenvectors with the largest eigenvalues belong to the highest variance between the face images, so a subspace from the set of eigenvectors can be spanned, that called the face space. Therefore each individual human face can

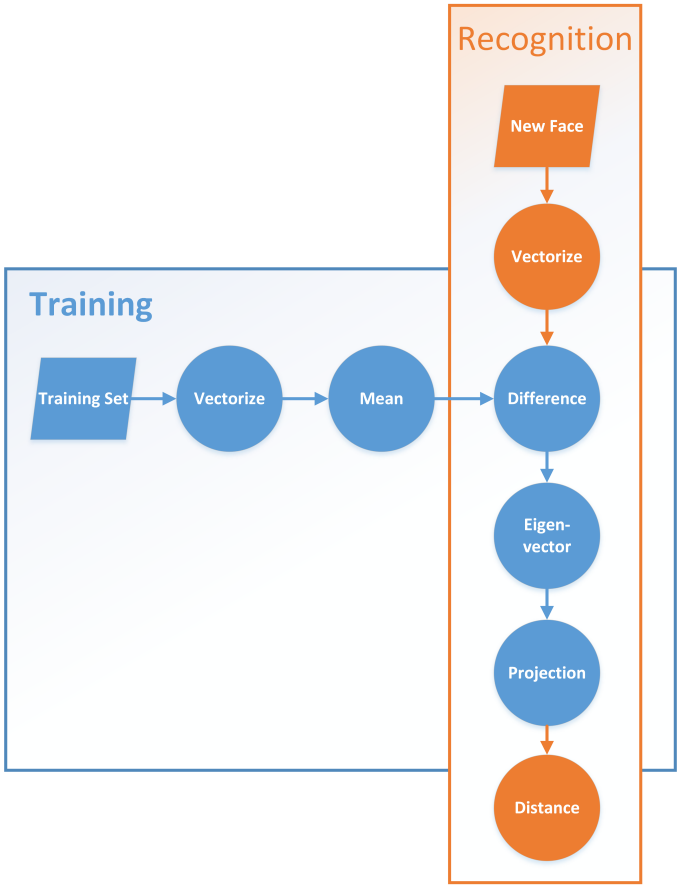

Figure 4.1: *Schematic diagram of the Eigenfaces method*

be characterized by a small set of eigenvectors, which means a compact and efficient representation of the original image.

The Eigenfaces method or PCA can be separated into the learning and the recognizing phase, as in Fig. 4.1 can be seen. In the learning phase firstly a set of human face images are needed, which will be the training set. Then calculate the eigenvectors of the training set and define the face space. Finally the projection of the face images onto the face space is required, by computing the distribution in the face space of each face image. These steps could be performed again whenever a new face has to be learned. After the learning phase ended, the recognizing phase can begin by computing the eigenvectors of the face to be recognized. Finally the projection to each eigenvector is required, which will show the degree of similarity between the input face image and the set of training faces.

Furthermore it can be determined, that the input is a face or another object by the distance between the projection result and the elements of the face space. Also there is the potential to learn the unknown faces if that face is already recognized as unknown face several times.

4.3.2 Eigenface Calculation

A face image I is a two-dimensional $N \times N$ array containing grayscale $0-255$ values. In the calculation of the Eigenfaces firstly the creation of column vectors is required from the original face images. Consequently the length of the vectors will be N^2. The next step is to compute the mean face, which can be done as follows:

$$\Psi = \frac{1}{M}\sum_{n=1}^{M}\Gamma_n, \tag{4.1}$$

where Ψ is the mean face, Γ_n is the n_{th} face image from the training set and M is the number of faces. Then the difference from the average can be calculated:

$$\Phi_i = \Gamma_i - \Psi, \tag{4.2}$$

where Φ_i is the i_{th} difference image. Based on the difference images the distribution of the data has to be defined by finding the orthonormal vectors u, known as the eigenvectors. The k_{th} eigenvector can be chosen as:

$$\lambda_k = \frac{1}{M}\sum_{n=1}^{M}u_k^T{\Phi_n}^2, \tag{4.3}$$

where λ_k is the k_{th} eigenvalue of the C covariance matrix:

$$C = \frac{1}{M}\sum_{n=1}^{M}\Phi_n\Phi_n^T = AA^T, \tag{4.4}$$

where the A matrix consists of the difference images $[\Phi_1\Phi_2...\Phi_M]$. However the covariance matrix has an $N^2 \times N^2$ dimension, and the computation of the eigenvectors could be time and resource consuming even for small images. Therefore a computationally feasible approach is required for the eigenvector computation.

There is a practical solution for the problem. Based on the number of images there will be only $M-1$ usable eigenvectors with the highest eigenvalues, while the other eigenvectors have eigenvalues close to zero. This scenario can be solved easily by calculating the eigenvectors of a $M \times M$ dimension matrix, rather than a 4096×4096 matrix. Taking the eigenvectors v_i of A^TA as:

$$A^TAv_i = \mu_i v_i, \tag{4.5}$$

and multiplying (5) with the A matrix on both sides:

$$AA^TAv_i = \mu_i Av_i, \tag{4.6}$$

where μ_i is the i_{th} corresponding eigenvalue, it can be seen, that Av_i is the eigenvector of the original covariance matrix C. Using this method the eigenvectors of the M training face images can be determined as follows:

$$u_i = \sum_{k=1}^{M} v_{ik}\Phi_k, \quad i = 1 \ldots M \tag{4.7}$$

Using this method the order of computations is only M, which is the number of images in the training set, rather than N^2, which is the number of pixels in one image.

4.3.3 Classification

In the classification process a new face is projected onto the face space as follows:

$$\omega_i = u_i^T(\Gamma_{new} - \Psi), \quad i = 1 \ldots M, \tag{4.8}$$

where ω_i is the i_{th} projection value based on the i_{th} eigenvector and the new face image Γ_{new}. The projection values represent the contribution to the corresponding eigenvector to reconstruct the input face. A vector Ω can be formed of the projection values as follows:

$$\Omega = [\omega_1\ \omega_2 \ldots \omega_M] \tag{4.9}$$

Based on the Ω projection vector the most similar face can easily determined by a distance metric as follows:

$$d_i = ||\Omega_T - \Omega_i||, \quad i = 1 \ldots M, \tag{4.10}$$

where Ω_T is the projection matrix based on the training images. The minimum of the distances will be the most similar face to the new face [97, 98].

4.3.4 Overall System

The automatic person identification system is capable of detecting and recognizing human faces, which schematic diagram can be seen in Fig. 4.2. The automated identification system consists of a video camera, a Personal Computer (PC), an FPGA, and a display. In this work only the face recognition has been implemented and tested on the FPGA hardware, the other software parts of the system are running in MATLAB on the PC. The configuration of the PC consists of a Core i7-4770 running on 3.4 GHz and 8 GB DDR3 RAM.

The automatic identification system works as follows. The video stream provided by the video camera is processed on the host PC to detect faces using the Viola-

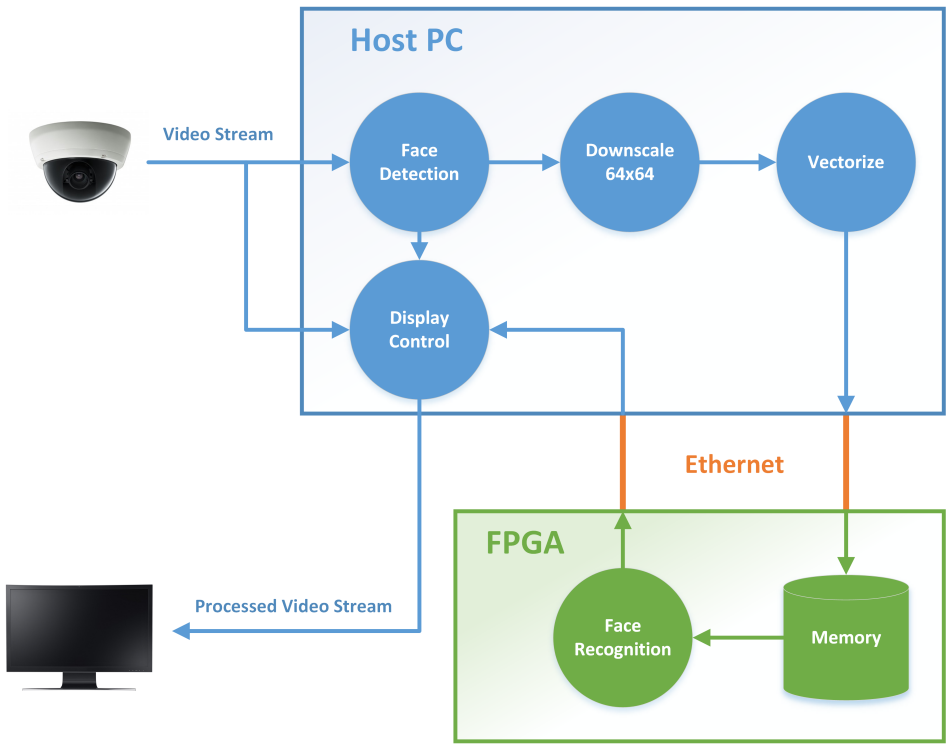

Figure 4.2: *Schematic diagram of the automatic person identification system*

Jones algorithm with Haar-like features [99]. The detected faces are downscaled to 64×64 and vectorized, which is the preparation for the face recognition with the Eigenfaces method. The FPGA can be configured through the Gigabit Ethernet to learn or recognize the transferred face, therefore it can be used to learn faces automatically or to define the training database. The vectorized faces are transferred via Gigabit Ethernet to the FPGA, which classifies the faces in real-time based on the training database. The classes of the identified faces then transferred back to the PC, which visualizes the results on the monitor based on the original video frame, the detected faces and the resulting face classes.

4.3.5 FPGA implementation

The face recognition architecture was implemented on a Zybo development board using Vivado 2016.4 and Vivado High Level Synthesis (HLS). Vivado HLS enables the usage of C/C++ or SystemC with some FPGA specific restriction. The Zybo development board contains a 650 MHz dual-core Cortex-A9 microprocessor (Processing System - PS), a Zynq-Z7010 FPGA (Program Logic - PL), an external 512 MB DDR3 memory with 1050 Mbps bandwidth, a high-bandwith 1 Gigabit Ethernet, and other peripherals.

The architecture of the proposed system consists of six main parts as can be seen

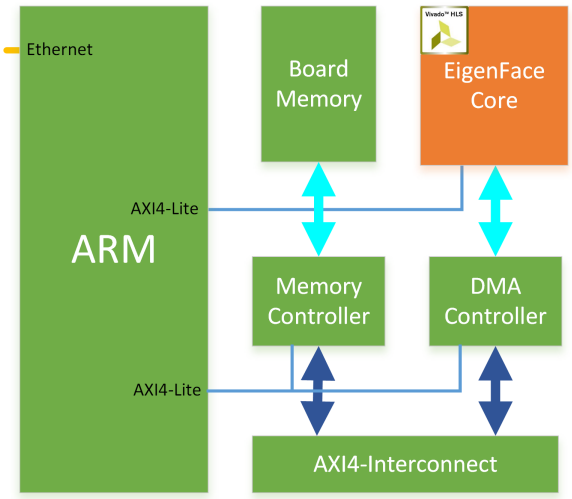

Figure 4.3: *Architecture of the face recognition system*

in Fig. 4.3. These are the *ARM Processor*, the *DMA Controller*, the *AXI-4 Interconnect*, the *Memory Controller*, the *Board Memory* and the *EigenFace Core*.

The *ARM Processor* communicates with the host computer via Gigabit Ethernet and controls the data-flow on the AXI4-Lite and AXI4 buses. Furthermore it pre-calculates the required matrices for the *EigenFace Core*. The *Memory Controller* and the *Board Memory* are responsible to store the human faces extracted from the video stream and also the sub and final results of the classification process. The data transfer between the *EigenFace Core* and the *Board Memory* is handled by the *DMA Controller*. In the memory the data is stored sequentially, but the *EigenFace Core* requires it in a mixed manner, therefore scatter-gather (SG) DMA instructions are used. The *EigenFace Core* computes the algorithmic steps of the EigenFaces method. In the first step it calculates the mean face. In the second step difference from the mean is computed for each face vector. As the third step the eigenvectors are computed. In the fourth step the projection to the face space is computed. If the system is in recognition mode, then finally the distances between the projections of the training set and the projection of the input face is calculated. In the *EigenFaces Core* only the data for the actual step is stored, to minimize the BRAM memory usage. The *AXI-4 Interconnect* provides the connection between the parts of the system.

In mean face calculation only addition and a multiplication with the reciprocal of M is required at the end of each row, when the faces are column vectors. This calculation can be done parallel with a factor of 64, which is one dimension of the faces. The eigenvector computation consists of three matrix multiplication, which unfortunately can not be done parallel due to data dependency. However the multiplication of two matrices can use parallelism in the resulting matrix element calculation, therefore one element of the new matrix can be computed in a clock cycle. After one column

Table 4.1: *Resource Requirements of the EigenFaces Core*

#	BRAM(18K)	DSP48E	FF	LUT
Available	120	80	35200	17600
Required	80	11	2802	4493
Utilization	66%	13%	7%	25%

Table 4.2: *Comparison of System Costs*

Paper	FPGA	Cost (EUR)	Face/Sec	Cost / Face/ Sec (EUR)
[70]	Virtex-7	3613	-	-
[71]	Virtex-2 PRO	1045	-	-
[72]	Virtex-2 PRO	1045	43049	0.02
Proposed	Zynq-Z7010	169	13026	0.013

of the resulting matrix is computed it can be used immediately for the next matrix multiplication. The projection step includes the multiplication of each face vector with the eigenvector matrix, which can be parallel with a factor of 64. Finally the euclidean distance calculation is required, which is done serially due to the high DSP requirements of the square roots.

4.3.6 Results

The resource requirement of the EigenFace Core can be seen in Table 4.1. To maintain a low Block RAM memory usage, the required matrices are loaded from the DDR3 memory in every step and used immediately.

The FPGA implementation of the EigenFaces method has been tested with the Faces94 database, which contains 20 face images of 153 people. The facial expression on the images are slightly varied. From the 153 people 8 were chosen for testing. From the 20 images of the 8 people only 1 was used for training and the other 19 for validation. Using these face images 95% recognition efficiency can be achieved, with a 13026 faces/second throughput on 100 MHz clock frequency.

A comparison with the system costs of other papers can be seen in Table 4.2, which shows that the proposed system is more cost-efficient, than the similar real-time FPGA-based systems.

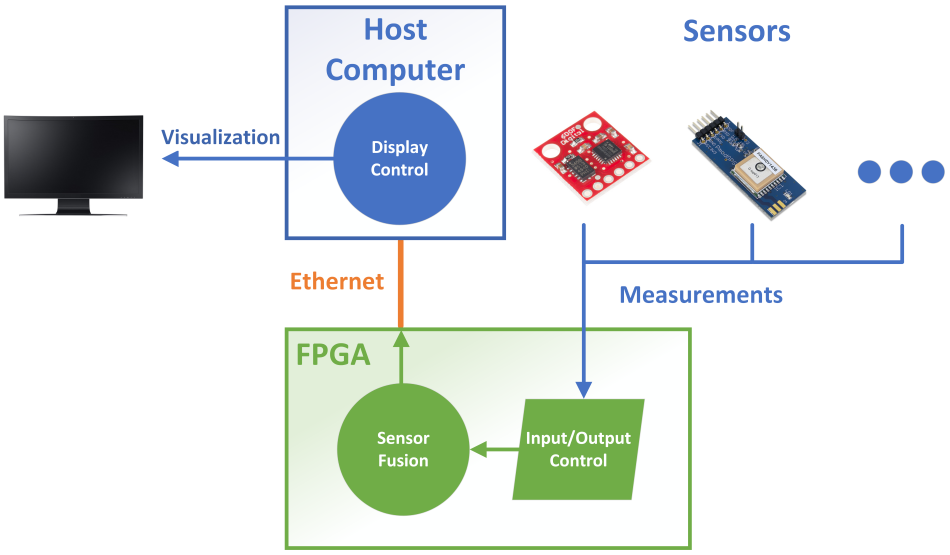

Figure 4.4: *Schematic diagram of the pose estimation system*

4.3.7 Conclusion

In this work a low-cost real-time FPGA-based implementation of the EigenFaces method for human face recognition is proposed.

The test results show that the architecture is capable of 95% recognition accuracy, and can process 13026 faces in a second. Furthermore, the architecture can be implemented on a low-cost Z7010 FPGA.

4.4 A real-time pose estimation algorithm based on FPGA and sensor fusion

On a mobile vehicle the calculation time of the estimations is critical, because the environment can change swiftly, therefore the measurements have to be processed in real-time. Taking advantage of the hardware acceleration capabilities of a Field-Programmable Gate Array (FPGA) the processing time can be significantly reduced.

In this work a real-time pose estimation algorithm is presented using sensor fusion, which is implemented in MATLAB and on a low-cost energy-efficient ZYBO development board containing a Z-7010 FPGA.

4.4.1 Measurement Setup

The measurement setup can be seen in Fig. 4.4, which consists of four different parts:

- An Inertial Measurement Unit (IMU) board

- A Global Positioning System (GPS) module
- An FPGA
- The host computer

In the following, the different parts of the measurement setup are detailed.

Inertial Measurement Unit

A Sparkfun ITG3200/ADXL345 IMU board was used for measuring acceleration and angular velocity. The ITG3200 is a MEMS gyroscope with a $\pm 2000 deg/s$ range, 14 $LSBs/deg/sec$ sensitivity and 8 kHz maximal sampling frequency, while the ADXL345 is an accelerometer with $16g$ range, 31.2 mg/LSB sensitivity and 3.2 kHz maximal sampling frequency. Both sensors communicate with I^2C protocol on separate outputs.

Global Positioning System

A Digilent GPS module was used for measuring the location with a GlobalTop FGPMMOPA6H GPS antenna board, which contains a MediaTek GPS MT3329 chip. The module works with 3 m positioning accuracy, the maximal update rate is 10 Hz and uses the UART protocol to communicate the NMEA GPS measurements.

FPGA

A ZYBO development board was used to real-time estimate the pose using the sensor fusion algorithm. The ZYBO development board contains a 650 MHz dual-core Cortex-A9 microprocessor, called Processing System (PS), a Z-7010 FPGA, called Program Logic (PL), an external 512 MB DDR3 memory with 1050 Mbps bandwidth, a high-bandwidth 1 Gigabit Ethernet port, and peripheral module (PMOD) input/outputs. The PMOD ports can be used to connect different external modules using digital (I^2C, SPI, $UART$, etc.) communication protocols.

Host PC

The host PC was used to prototype the sensor fusion algorithm in MATLAB and to visualize the results of the pose estimation. The configuration of the PC consists of 8 GB DDR3 RAM and a Core i7-4770 CPU running at 3.4 GHz.

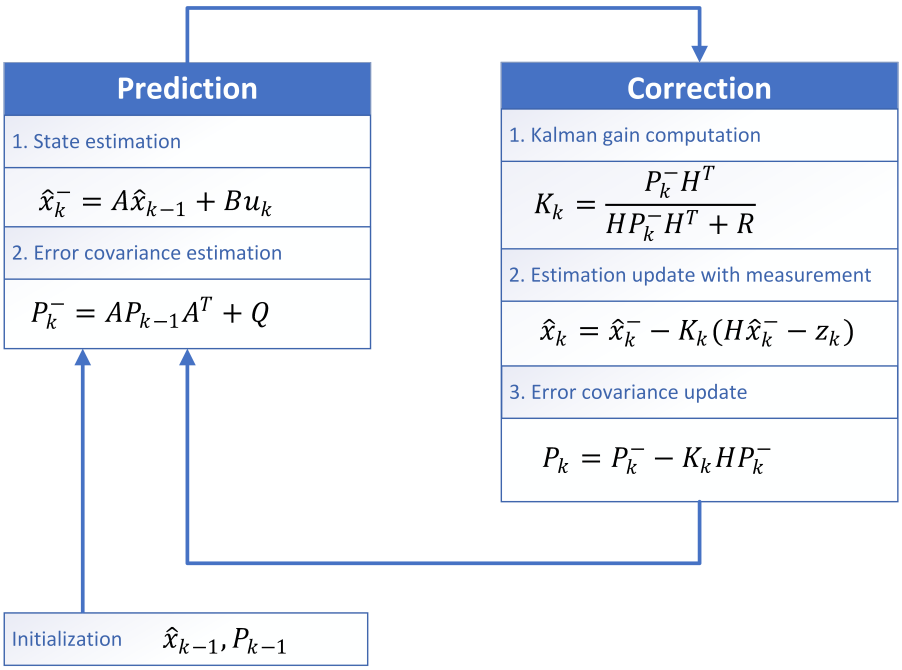

Figure 4.5: *Schematic diagram of the Kalman filter*

4.4.2 Sensor Fusion

The main part of the pose estimation is the sensor fusion algorithm, which is based on the widely known Kalman filter. The Kalman filter estimates the states $\underline{x}$ of a system based on the state transition matrix $\underline{\underline{A}}$, the error covariance $\underline{\underline{P}}$, the observed state (state observation matrix) $\underline{\underline{H}}$, the Kalman gain $\underline{\underline{K}}$ and the measurements $\underline{z}$, considering the process noise covariance $\underline{\underline{Q}}$ and the measurement covariance $\underline{\underline{R}}$. The Kalman gain describes the reliability of the actual measurement based on the previous knowledge. The schematic diagram of the Kalman filter algorithm can be seen in Fig. 4.5. The Kalman filter can be used to control the actuators of the system, in this case the control matrix $\underline{\underline{B}}$ and the control input $\underline{u}$ are used as well. In the presented case the states of the system are acceleration $\underline{a}$, velocity $\underline{v}$, position $\underline{p}$, angular rate $\Delta\phi$ and angle ϕ , so the state vector for one axis can be written as follows:

$$\underline{x} = \begin{bmatrix} a_x & v_x & p_x & \Delta\phi & \phi \end{bmatrix}^T \tag{4.11}$$

The state transitions of a simple linear mathematical model are used to describe the system, which can be written as follows:

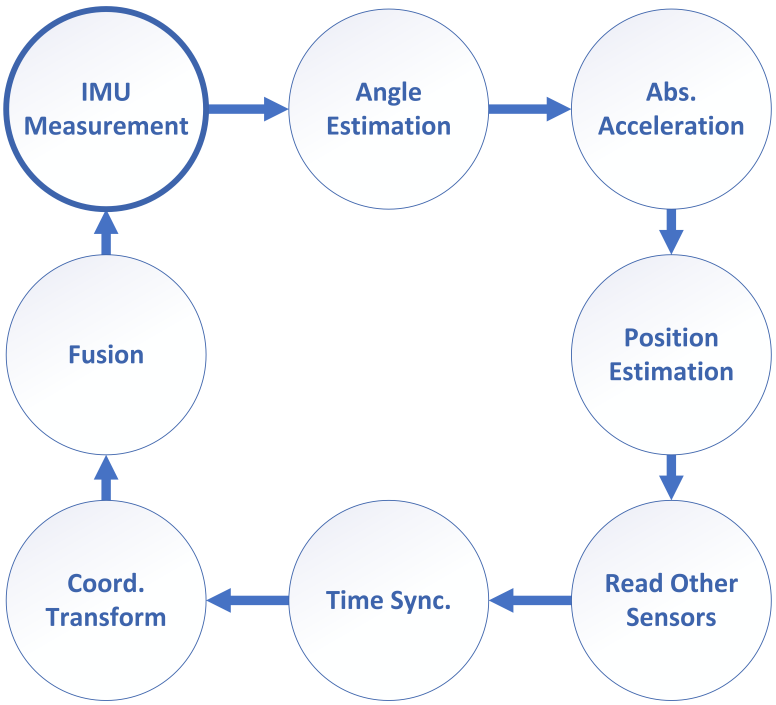

Figure 4.6: *Algorithmic steps of the pose estimation with sensor fusion*

$$\underline{\underline{A}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ dt & 1 & 0 & 0 & 0 \\ 0 & dt & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & dt & 1 \end{bmatrix}, \underline{\underline{B}} = 0, \tag{4.12}$$

where dt is the sampling interval.

The main steps of the sensor fusion can be seen in Fig. 4.6, which are the following:

1. Acceleration and angular rate (IMU) measurement
2. Angle estimation
3. Absolute acceleration calculation
4. Position estimation
5. Read other sensors (GPS in this case)
6. Time synchronization
7. Coordinate transformation

8. Fusion (Position in this case)

In the presented algorithm the Kalman filter is divided into two parts. The first part estimates the angles based on the angular rate measurements, while the second part estimates the position using absolute acceleration, which is computed based on the rotational compensation by the estimated angle on each acceleration axis. The sampling rate of the GPS is less frequent than the IMU, therefore if a measurement is available time synchronization required, which is based on the attached timestamps. Each sensor module works in a different coordinate system, so coordinate transformation is required to be able to use the measurements for computations. Finally, the position and the velocity fusion can be done with the following steps:

1. Selection of the observed state ($\underline{\underline{H}}$)
2. Calculation of the innovation ($\underline{\tilde{x}}$)

$$\underline{\tilde{x}}_k = \underline{x}_k - \underline{z} \tag{4.13}$$

3. Kalman gain computation ($\underline{\underline{K}}$)
4. State correction

$$\underline{x}_{k+1} = \underline{x}_k - \underline{\underline{K}} \cdot \underline{\tilde{x}}_k \tag{4.14}$$

5. Update state error covariance ($\underline{\underline{P}}$)
6. Ensure symmetry of $\underline{\underline{P}}$

$$\underline{\underline{P}}_{k+1} = \frac{\underline{\underline{P}}_{k+1} + \underline{\underline{P}}^T_{k+1}}{2} \tag{4.15}$$

When fusing a measurement with a state variable, the reliability of the actual measurement is considered using the Kalman gain $\underline{\underline{K}}$. If a measurement is too noisy, the corresponding $\underline{\underline{R}}$ value is high, so the actual state will be taken into account in greater part. On the other hand, an accurate measurement has a greater impact on the observed state.

4.4.3 FPGA system architecture

The architecture of the FPGA-based sensor fusion system, which can be seen in Fig. 4.7. was implemented on a low-cost ZYBO development board using Vivado 2016.4 and Vivado High Level Synthesis (HLS). Vivado HLS enables the usage of C/C++ or SystemC languages with some FPGA specific restrictions.

The architecture of the proposed system is built-up from seven main parts as can be seen in Fig. 4.7. These are the *ARM Processor*, the *DMA Controller*, the *AXI-4*

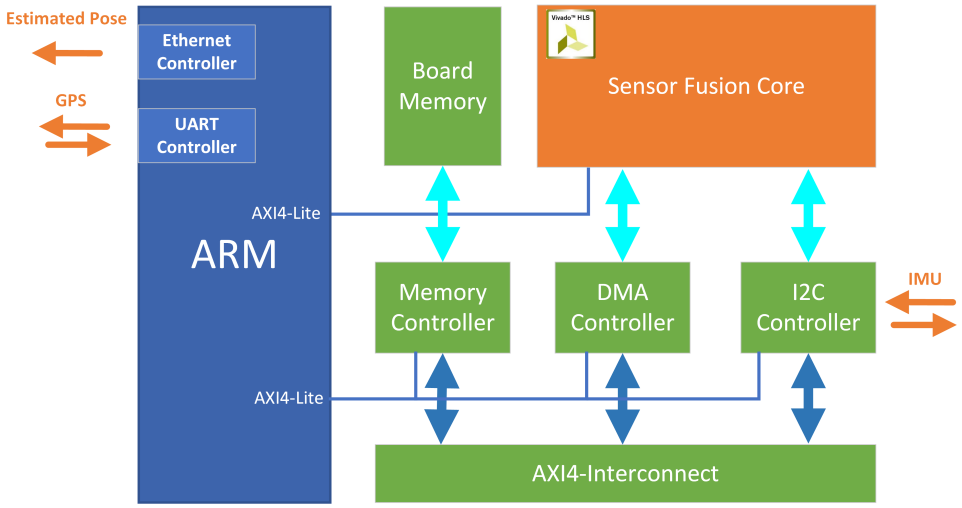

Figure 4.7: *Architecture of the sensor fusion system*

Interconnect, the *Memory Controller*, the *Board Memory*, the *I2C Controller* and the *Sensor Fusion Core*.

The *ARM Processor* communicates with the host computer via Gigabit Ethernet and controls the data-flow on the AXI4-Lite and AXI4 buses. The system is capable of handling measurements from various sensor modules in this case an IMU and a GPS. The *ARM Processor* receives GPS data via the built-in *UART Controller* from the GPS and transfer to the *Sensor Fusion Core*. The *I2C Controller* communicates with the IMU module. Since the system can be extended with other sensor boards (e.g. the visual odometry module), an additional *SPI Controller* can be added to the system during the implementation if it is required. The implementation of the visual odometry external module is not detailed in this paper.

The *Memory Controller* and the *Board Memory* are responsible for storing the results of the pose estimation until the *ARM Processor* transmits it to the *Host Computer*. The data transfer between the *Sensor Fusion Core* and the *Board Memory* is handled by the *DMA Controller*. The *Sensor Fusion Core* computes the algorithmic steps of the sensor fusion using a Kalman filter. The *AXI-4 Interconnect* provides the connection between the parts of the system.

4.4.4 Implementation

The sensor fusion algorithm is implemented on the FPGA in four modes. In the first mode the states and the matrices are initialized during the start-up. In the second mode only IMU based position and angle estimation is calculated, which is necessary because the sampling rate of the IMU is the highest. In the third mode the fusion of the previously estimated position (based on the IMU) and the new measurement of a

less frequent sensor is computed. In the fourth mode the fusion of the previously estimated angle (also based on the IMU) and the new measurement of another sensor is calculated. The different modes are started by the ARM processor, which determines the required mode for the actual measurement through the communication protocol or the ID of the sensor.

The ARM Processor is capable of computing the coordinate transformations and time synchronizations in real-time, so it does not require any FPGA resources. The different measurements of the different sensors are timestamped on the ARM processor using the FPGA timer and subtracted from the actual timestamp of the IMU measurement to calculate the difference between the time of the measurements. Using the time differences the reliability of the measurements can be computed.

4.4.5 Results

Table 4.3: *Area requirements and device utilization of the synthesized sensor fusion module*

#	BRAM (max. 120)	DSP (max. 80)	FF (max 35.2K)	LUT (max 17.6K)
Unrolled	3%	58%	27%	80%
Pipelined	3%	77%	34%	92%

The utilization results of the pose estimation algorithm using sensor fusion can be seen in Table 4.3. Two versions of the algorithm were synthesized to the FPGA. The first version of the optimized matrix multiplication were unrolled, so the parallelism of the FPGA was maximally utilized. In the second version the optimized matrix multiplications were pipelined. The vectors were stored in Look Up Tables (LUTs), therefore the most utilized resource is the LUT. Only the covariance matrices were stored in 18Kbit Block RAMs (BRAMs).

The detailed latency results can be seen in Table 4.4. Both versions can run on $9.26ns$ clock period, so the main difference is in the utilization, therefore the unrolled version is better. The unrolled version has a latency of 105 clock cycles, and an interval (wait for rerun) is 106 clock cycles, therefore on $9.26ns$ the processing time of one measurement is $1.95\mu s$, which enables the real-time usage of the system. The power consumption of the ZYBO running the proposed pose estimation architecture is only 2 $Watts$.

The pose estimation was tested on a recording where the vehicle travels $50meters$ along a curved path. The average error of the sensor fusion was $0.4m$ with $0.18m$

Table 4.4: *Detailed latencies of the sensor fusion module*

#	Latency (clock cycle)	Interval (clock cycle)	Clock
Unrolled	105	106	9.26 ns
Pipelined	122	123	9.26 ns

standard deviation both in MATLAB and on the ZYBO. The MATLAB running on the *Host Computer* processed one measurement at $0.7ms$ and the power consumption of the CPU was $19.3Watts$, the FPGA-based architecture operate at $1.95\mu s$ per measurement and consume $2Watts$, therefore the FPGA-based implementation is 358 times faster and 9.65 times more energy efficient.

In safety-critical applications the reliability of the system is an important factor, which is based on the failure rate of the FPGA, the reliability of the power supply and the measurements of the sensors. The failure rate of a Xilinx FPGA manufactured with 28 nm technology is 11 FIT (Failure In Time), which means 11 failures in 10^9 device hours [100]. If the measurements of the sensors are not in the range of standard operation the pose estimation is based on the mathematical model, which is reliable only in short-terms.

4.4.6 Extension with UWB-based module for indoor localization

Indoor position estimation is an important part of any indoor application which contains object tracking or environment mapping. Many indoor localization techniques (Angle of Arrival - AoA, Time of Flight - ToF, Return Time of Flight - RToF, Received Signal Strength Indicator - RSSI) and technologies (WiFi, Ultra Wideband - UWB, Bluetooth, Radio Frequency Identification Device - RFID) exist which can be applied to the indoor localization problem. Based on the measured distances (with a chosen technique), the position of the object can be estimated using several mathematical methods. The precision of the estimated position crucially depends on the placement of the anchors (fixed positioned UWB transceiver), which makes the position estimate less reliable. The error characteristics of the DWM1001 UWB ranging module (a commercially available UWB localization system using RToF) were measured and used to find an optimal anchor placement, which improves position estimation accuracy [101]. Using the proposed UWB extension with the fusion of IMU measuerements indoor localization is possible.

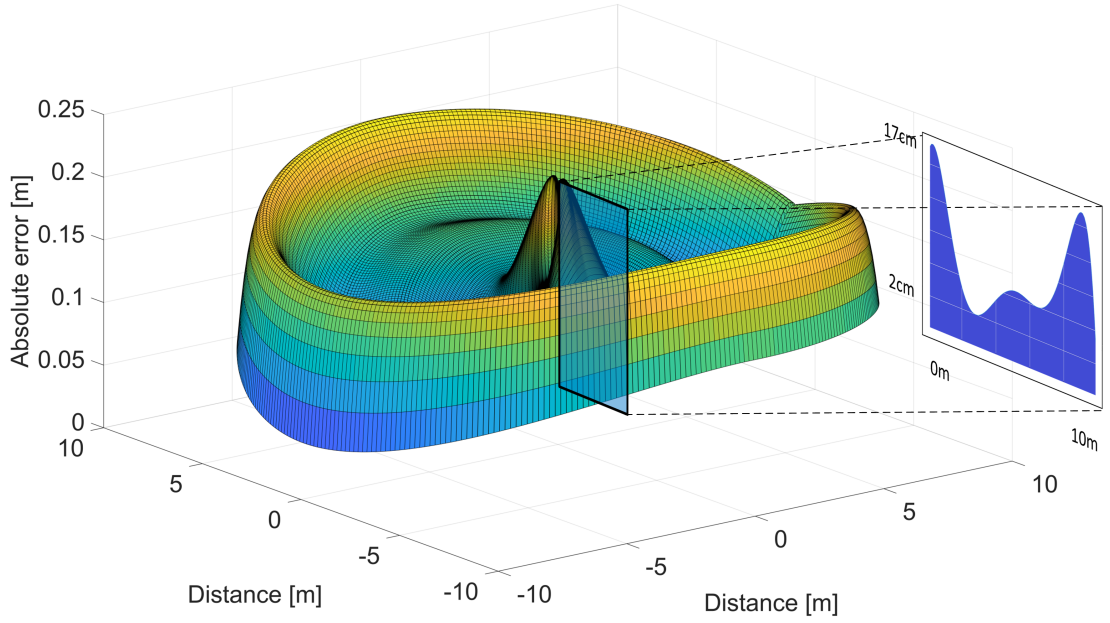

Figure 4.8: *Realistic 2D error characteristic of DWM1001*

Error characteristic

Based on the measurements using a laser rangefinder ($\pm 1mm$ accuracy), the 2D error characteristics of the DWM1001 module with the built-in on-board antenna was determined. The measurement consisted of placing two modules in a known distance from each other, and performing a measurement with the UWB technology using RToF technique.

The distances was determined in discrete points from 0.3 to 10 metres in steps of 0.3 meters. At each discrete point at least 100 measurements were conducted with varying antenna orientation ($0°$, $90°$, $180°$) and the mean value was calculated.

After the measurements, at each measured distance, an error value was interpolated for every degree between $0° - 180°$. Since the error characteristic is symmetrical in this plan the corresponding error values for degrees between $180°$ and $360°$ can be used from the interpolated ones. Each point of the 2D error characteristic is loaded into a Look-Up-Table (LUT), since in the optimization process the genetic algorithm reads the corresponding error value addressing the LUT by the distance and orientation parameters.

The visualization of the realistic 2D error characteristic can be seen in Fig. 4.8, where one slice of the surface represents the distance error of one degree of rotation of the DWM1001 module.The DWM1001 has a ranging accuracy within $10cm$. But the results in Fig. 4.8 showed that the accuracy is a nonlinear function of distance

and orientation. Furthermore, between 0-1m and 9-10m, the error is significantly higher than $10cm$ ($17cm$ $\pm 2cm$). The highest accuracy can be reached around 2 meters ($7cm$ $\pm 2cm$).

Multilateral algorithm

The multilateral algorithm [102] can be used to determine the unknown position in a 3D space - assuming an adequate number of reference points in a common Descartes coordinate system spanning the space - by solving the following linear equation system:

$$(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 = d_i^2; \qquad i = 1...N, \tag{4.16}$$

where N is the number of anchor nodes, (x_i, y_i, z_i) is the coordinate of the i^{th} anchor node, (x, y, z) is the coordinate of the mobile node and d_i is the point-to-point distance between the i^{th} anchor node and the mobile node respectively. Rearranging (4.16) in 2D case, it can be written in the following matrix representation:

$$\begin{pmatrix} -1 & -2x_1 & -2y_1 \\ -1 & -2x_2 & -2y_2 \\ \vdots & \vdots & \vdots \\ -1 & -2x_N & -2y_N \end{pmatrix} \cdot \begin{pmatrix} x^2 + y^2 \\ x \\ y \end{pmatrix} = \begin{pmatrix} d_1^2 - x_1^2 - y_1^2 \\ d_2^2 - x_2^2 - y_2^2 \\ \vdots \\ d_N^2 - x_N^2 - y_N^2 \end{pmatrix}. \tag{4.17}$$

Rewriting (4.17) in a short form:

$$\mathbf{A} \cdot \boldsymbol{\eta} = \mathbf{b}; \quad \mathbf{A} \in \mathbb{R}^{N \times 3}, \quad \boldsymbol{\eta} \in \mathbb{R}^3, \quad \mathbf{b} \in \mathbb{R}^N. \tag{4.18}$$

The solution of (4.18) for $\boldsymbol{\eta}$ is given by:

$$\boldsymbol{\eta} = \mathbf{A}^+ \cdot \mathbf{b}, \tag{4.19}$$

where A^+ denotes the Moore-Penrose pseudo-inverse of matrix A. Assuming that the positions of anchor nodes are fixed during the measurement, it is enough to calculate A^+ once offline thus speeding up the position estimation process.

FPGA architecture

As can be seen in Fig. 4.7 various sensors can be attached to the the FPGA. Therefore the GPS sensor can be replaced with an interface board using the DWM1001 module, which also uses the UART communication protocol. In this case the interpretation of the incoming data packets and the corresponding parameters in the sensor fusion core have to be changed. The sensor fusion parameters can be sampled based on

the measured distance from a LUT containing the error characteristics. The Moore-Penrose pseude-inverse of matrix A can be computed only once beforehand, so it can be stored also in a LUT. It depends only on the reference anchor points, which are fixed and have to be known beforehand. Therefore, using 4.19 only b has to be calculated and multiplied by A^+. A schematic architecture for this matrix-vector multiplication can be seen in Fig. 4.9.

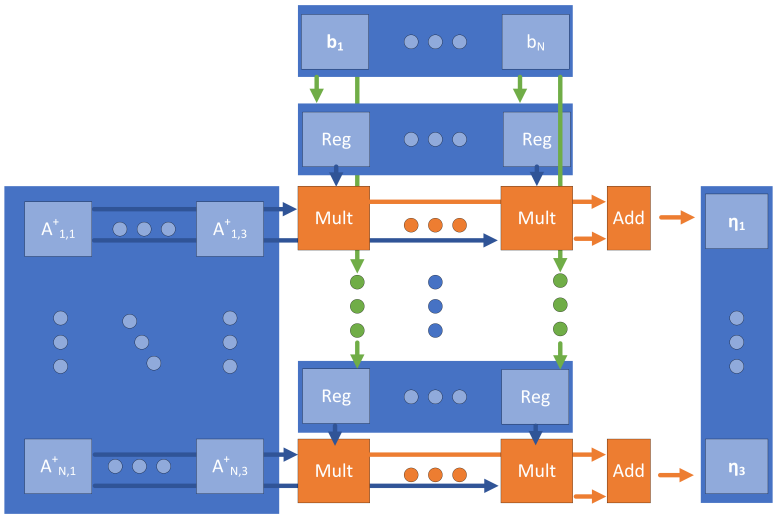

Figure 4.9: *Schematics of multilateral matrix-vector multiplication architecture*

The multilateral calculation with the matrix-vector multiplication when each row computed in parallel takes 59 clock cycles. Together with the sensor fusion computation it takes 165 clock cycles, which also makes real-time operation possible.

The utilization of the multilateral computation can be seen in Table 4.5. Unfortunately, together with the sensor fusion architecture the resource consumption exceeds the available DSP resources of the chosen Zybo (Z-7010) board. Therefore the Z-7020 Zybo could be used, which has more than double resources.

Table 4.5: *Utilization of the Multilateral Computation Architecture*

	Utilization	%
FF	4111	11.7%
LUT	5905	33.5%
DSP	42	52.5%
BRAM	0	0%

4.4.7 Conclusion

In this work a real-time pose estimation algorithm is presented using sensor fusion implemented on the ZYBO development board. The proposed low-cost sensor fusion architecture processes one sensor measurement in $1.95\mu s$ and consumes only 2 $Watts$, which enables real-time mobile applications. The presented algorithm was tested in MATLAB and on the FPGA. The results showed that the FPGA-based implementation is 358 times faster.

Also an indoor extension possibility is presented with the usage of a commercially available UWB module. The optimal placement of the reference position devices are determined, also the error characteristics of the UWB module is measured. An FPGA architecture for the multilateral algorithm to estimate the indoor position based on the UWB measurements are presented.

4.5 Implementation of an FPGA-based actual observer for active suspension control

In an active suspension system, some state variables can not be measured directly and the usage of sensors is not necessary, due to the cost, accuracy and reliability. Therefore a full state feedback control can not be used, unless a state observer is applied.

The easy reconfiguration of FPGAs gives opportunity for education purposes, because students can analyze the operation of observer based control, and study the effect of parameter changes, without the risk of damaging or endangering the vehicle or the suspension system.

Table 4.6: *ZYNQ-Z7020 General and Dedicated Resources*

Resource type	Name	Quantity
General-Resources	Slices	13 300
	6-input LUT	53 200
	Flip-Flops	106 400
Dedicated-resources	36Kbit RAM	140
	DSP48E1 Slice	220
Microprocessor	ARM-Cortex A9	2 Cores

The actual observer was implemented on a ZedBoard which has a Zynq-z7020 (XC7Z020) FPGA (Programmable Logic – PL), and an embedded ARM Cortex A9 hard core microprocessor (Processing System – PS). The XC7Z020 FPGA is based on the Artix-7 FPGA architecture. The general and dedicated resources of this device

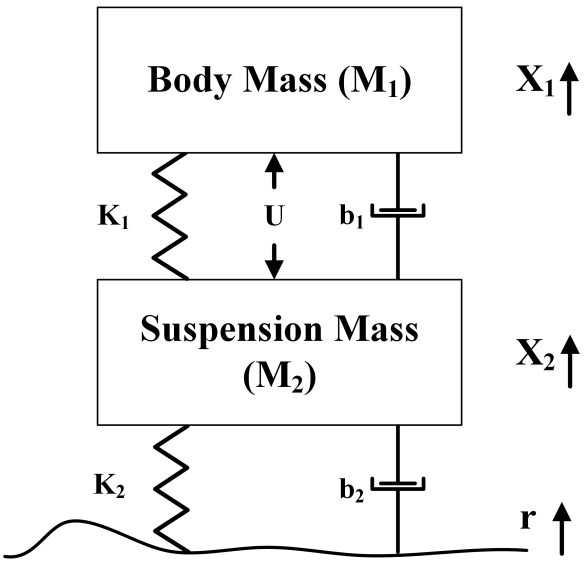

Figure 4.10: *A quarter of the active suspension system model*

are shown in Table 4.6. One CLB of the Artix-7 FPGA contains two Slices. A Slice contains eight registers, four 6-input LUTs, multiplexers and carry logics as general logic resources. A single LUT can be configured as a 32-bit shift register, a 6-input LUT or a 64-bit distributed RAM. Dedicated resources can be found on the Artix-7 architecture like 36kbit dual port BRAM memories, and 25x18bit DSP Slices. The ARM Cortex A9 microprocessor can work with a maximum of 866 MHz clock frequency, has 32 KB instruction and 32 KB data L1 cache in one core, 512 KB L2 cache memory, and 256 KB on chip memory. It supports DDR3 external memory, UART, CAN, I2C, SPI communication and maximum of four GPIO peripherals. The microprocessor communicates with the PL through AXI buses, and 16 interrupt ports.

4.5.1 Active suspension system model

Only the quarter of the active suspension system has been modelled and implemented in MATLAB/Simulink, because of an easier implementation. The model consists of two masses, the body and the suspension mass, which are connected to each other through a spring, a damper and a controllable actuator. The suspension mass is connected to the road through a spring and a damper. A quarter of the active suspension system model can be seen on Fig. 4.10, and the parameters of the system are shown in Table4.7. The mathematical model of the system can be obtained using Newton's law:

$$\ddot{X_1} = \frac{1}{M_1}(-b_1\dot{X_1} + b_1\dot{X_2} - K_1X_1 + K_1X_2 + U) \tag{4.20}$$

Table 4.7: *Parameters of the Active Suspension System Model*

Parameter	Description	Value	Unit
M_1	Quarter mass of the vehicle	250	kg
M_2	Suspension mass	320	kg
K_1	Spring coefficient of the Suspension	80 000	N/m
K_2	Spring coefficient of the Wheel	500 000	N/m
b_1	Damping coefficient of the Suspension	350	Ns/m
b_2	Damping coefficient of the Wheel	15 020	Ns/m
U	Control force	-	Nm
X_1	Position of the vehicle body	-	cm
X_2	Position of the suspension	-	cm
r	Disturbance from the road	-	cm

$$\ddot{X_2} = \frac{1}{M_2}(b_1\dot{X_1} - b_1\dot{X_2} + K_1X_1 - K_1X_2 + b_2\dot{r} - b_2\dot{X_2} + K_2r - K_2X_2 - U) \quad (4.21)$$

It can be seen, that the control force (U) and the disturbance from the road (r) are the two inputs of the system, while the output of the active suspension system is the distance between the vehicle body and the suspension (X1-X2). Using 4.20 and 4.21 the state-space model can be written in the form of:

$$\dot{x} = Ax + Bu, y = Cx + Du \quad (4.22)$$

Where x is the state vector, which contains the position and the velocity of the vehicle body and the suspension. It can be written as follows:

$$x = \begin{bmatrix} X_1 \\ \dot{X_1} \\ X_2 \\ \dot{X_2} \end{bmatrix} \quad (4.23)$$

Using 4.20 and 4.21 in the form of 4.22 the state-space matrices can be expressed:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{-k_1}{M_1} & \frac{-b_1}{M_1} & \frac{k_1}{M_1} & \frac{b_1}{M_1} \\ 0 & 0 & 0 & 1 \\ \frac{k_1}{M_2} & \frac{b_1}{M_2} & \frac{-k_1-k_2}{M_2} & \frac{-b_1-b_2}{M_2} \end{bmatrix} \quad (4.24)$$

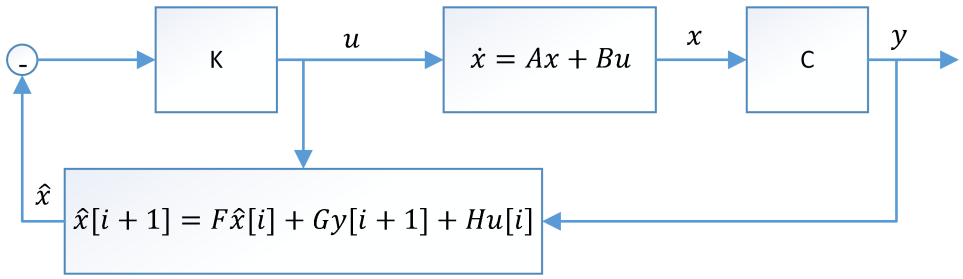

Figure 4.11: *The flow diagram of an actual observer based system*

$$B = \begin{bmatrix} 0 \\ \frac{1}{M_1} \\ 0 \\ \frac{-1}{M_2} \end{bmatrix} \tag{4.25}$$

$$C = \begin{bmatrix} 1 & 0 & -1 & 0 \end{bmatrix} \tag{4.26}$$

It can be seen that the D matrix is not presented, because it contains zeros. The active suspension system modelled in the state-space form is the base of the actual observer, and the observer is not prepared for the road disturbance, therefore the r input has been removed.

Using the *A*, *B* and *C* state-space matrices the observability (4.27) and controllability (4.28) of the system can be determined as follows:

$$O = \begin{bmatrix} C \\ CA \\ CA^2 \\ \dots \\ CA^{n-1} \end{bmatrix} \tag{4.27}$$

$$Co = \begin{bmatrix} B & AB & A^2B & \dots & A^{n-1}B \end{bmatrix} \tag{4.28}$$

If the rank of Co and O matrices equals to the number of states, then the system is controllable and observable. In this case the rank of these matrices are 4, therefore the requirements are fulfilled.

A model based on the equation system of 4.20 and 4.21 and a model in the state-space form have been implemented using the MATLAB/Simulink environment. The equation based model can be seen in Fig. 4.11, which will be the plant, that is controlled by the observer based LQR control.

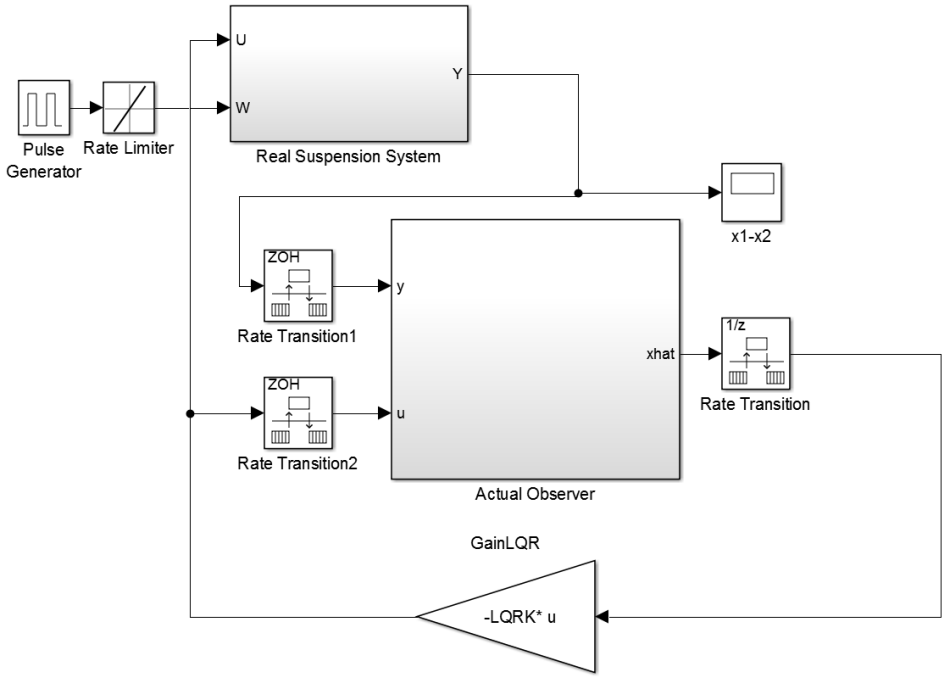

Figure 4.12: *The active suspension control Simulink system*

4.5.2 Actual observer

In a full-state feedback control all state variables should be measurable, if not the usage of an observer is required. Based on the output of the plant the actual observer can estimate the states and use them for a full-state feedback control. Using this method only a minimal number of sensors are required, the other state variables can be estimated. In this case from the position and velocity measurements, only the position is used in the observer. The main equation of the actual observer can be written as follows:

$$\frac{d\hat{x}}{dt} = F\hat{x} + Gy + Hu, F = A - GCA \tag{4.29}$$

where $\hat{x}$ is the estimated state vector, F, G and H are the observer gains. The dimension of F is 4x4, while G and H are only 4x1 vectors.

The actual observer works in discrete time, so the eigenvalues of the G matrix should be placed inside the unit circle with pole placement. The poles chosen for the G matrix are:

$$poles(G) = \begin{bmatrix} 0.63 & 0.62 & 0.61 & 0.6 \end{bmatrix} \tag{4.30}$$

The Simulink model of the active suspension system can be seen on Fig. 4.12., where the Real Suspension System is based on 4.20 and 4.21. The Rate Transition

blocks are converting the continuous time signals to discrete time, while the LQRK variable is the gain of the LQR control.

The Linear Quadratic Regulator is based on the following quadratic cost function:

$$J = \int_0^{\inf} (x^T Q x + u^T R u)\, dt \tag{4.31}$$

where the matrix Q is symmetric positive semi-definite and R is positive symmetric definite. In this case the optimal linear feedback control law can be obtained as:

$$u(t) = -Kx(t) \tag{4.32}$$

The weighting matrixes Q and R are:

$$Q = \begin{bmatrix} 100000 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.33}$$

$$R = 4 \cdot 10^{-6} \tag{4.34}$$

4.5.3 FPGA architecture

The implementation of the FPGA-based actual observer was made in the Vivado High Level Synthesis (HLS) programming environment, which enables fast prototyping with the usage of C/C++ code and enables the export of the resulting Intellectual Property (IP) Core into the Xilinx System Generator, so it can be used in the Simulink.

As in 4.29 can be seen the estimated state of the observer can be calculated with three matrix multiplication, and a summation. The values of the previous state estimation should be stored in registers and used in the multiplication with the F matrix. Using this technique, a delay can be made, so 4.29 will be satisfied. The F, G, H matrices are pre-calculated, and stored as constants. In the Simulink system, the Actual Observer block is replaced with the generated HLS actual observer IP core, and with Co-Simulation through JTAG, it can be evaluated in the FPGA. The block diagram of the FPGA-based actual observer can be seen in Fig. 4.13.

4.5.4 Results

The active suspension system was tested either using Simulink-based simulation and real FPGA hardware. The results show that the actual observer can be implemented on the ZedBoard with an 8.47 ns clock period, maximum of 100 clock cycle latency, which means one calculation takes 0.8 μs The DSP Slice requirement is the function

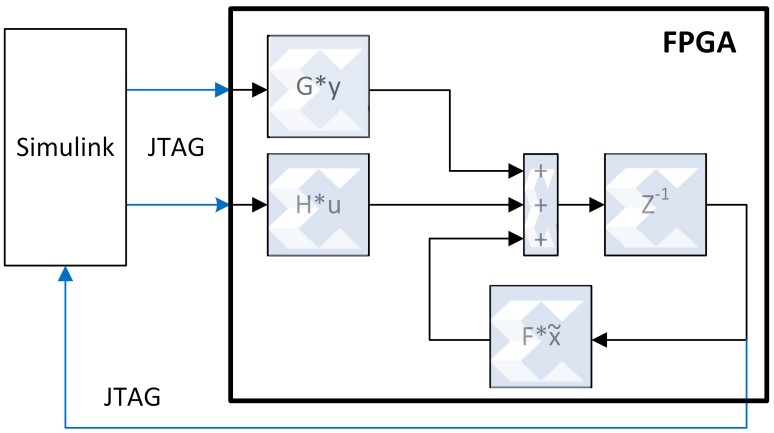

Figure 4.13: *The active suspension control FPGA architecture system*

Table 4.8: *Resource Requirements of the Actual Observer*

Resource type	Name	Quantity	Utilization (%)
General resources	Slices	218	1.6
	6-input LUT	567	1
	Flip-Flops	610	0.6
Dedicated resources	36Kbit RAM	0	0
	DSP48E1 Slice	32	14

of the number of add/sub and multiplication blocks. Because the calculations of (8) includes only three matrix multiplications, and two of them can be done at the same time, the architecture uses 32 DSP slices. The constant matrices are stored in LUTs, so no BRAM is required. The resource requirements can be seen in Table 4.8.

A comparison between the Matlab and the FPGA control force signal in the active suspension system can be seen in Fig. 4.14. It can be concluded that the control force signal emitted by the FPGA is similar to the Matlab/Simulink model.

4.5.5 Conclusion

It can be concluded, that the actual observer can be implemented on an FPGA, and it can calculate the results in real-time, with 8.47 ns clock period, 0.8 μs per calculation. The Simulink has the simulation period set to 0.001, therefore the observer calculation can run 125 times faster, than the Simulink. In the future other observer and control algorithms can be implemented in this framework, so it can be used not only for experimental, but for education purposes as well. Students can analyze the operation of the active suspension, and study the effect of parameter changes, without the risk of damaging or endangering the physical system. Furthermore, the plant and the corresponding model can be changed.

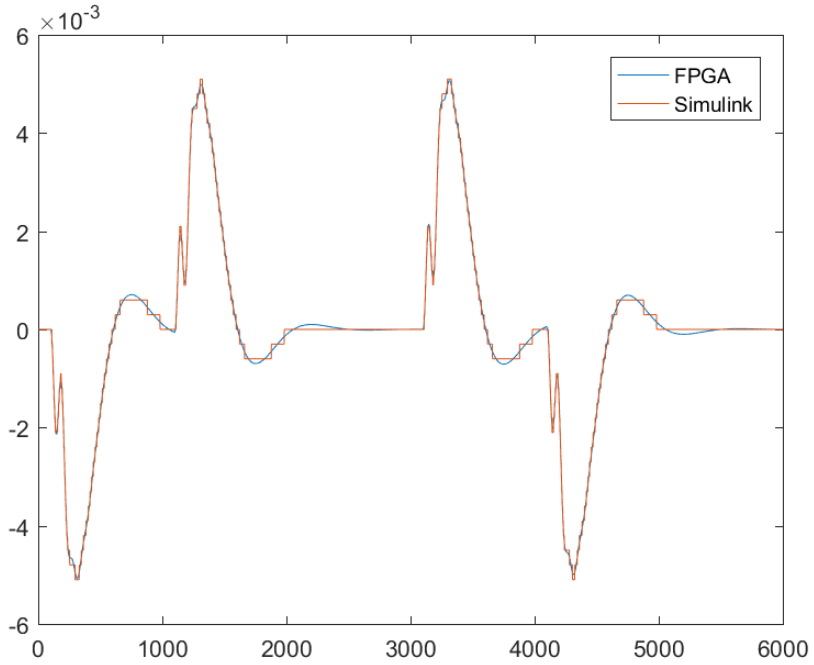

Figure 4.14: *Control force signals in the active suspension system (Blue - FPGA, Orange - Matlab/Simulink)*

4.6 Implementation of an FPGA-based wind turbine HIL model

Based on the literature research (see Section 4.2), it was reasonable to implement a dynamically adjustable FPGA-based HIL device for wind turbine modelling. Due to the great flexibility, high computational capacity and boundless reconfiguration possibilities of FPGAs, FPGA-based HIL devices can be used to easily implement various types of wind turbine systems.

Furthermore, the HIL approach is well applicable in education of wind turbine based systems, because students can analyze the operation of the wind turbine, and study the effect of different parameter changes, without the risk of damaging or endangering the real physical system.

4.6.1 System setup

The wind turbine model was implemented on an Artix-7 XC7A100T FPGA. The general and dedicated resources of this device are shown in Table 4.9. One CLB of the Artix-7 FPGA contains two Slices. A Slice contains four 6-input LUTs, eight registers, multiplexers and carry logics as general logic resources. A single LUT can be config-

ured as a 6-input LUT, a 32-bit shift register or a 64-bit distributed RAM. The Artix-7 architecture also contains dedicated resources like 36kbit dual port BRAM memories, and 25x18bit DSP Slices.

Table 4.9: *Artix-7 XC7A100T General and Dedicated Resources*

Resource type	Name	Quantity
General resources	Slices	15850
	6-input LUT	63400
	Flip-Flops	126800
Dedicated resources	36Kbit RAM	135
	DSP48E1 Slice	240

The BRAMs can be configured as two independent 18kbit SRAM, or a FIFO memory due to the dedicated logic of the BRAM. The DSP can be used for 25x18bit signed multiplication, 48bit addition/subtraction or logic operations. [103]

A Digilent Nexys-4 board was used to set-up the FPGA- based wind turbine HIL system. This board contains all the circuitry for configuring the Artix-7 XC7A100T FPGA, and also the required peripherals for implementing the HIL device. The PMOD interfaces can be used to connect low frequency, low pin-count peripheral modules to the Nexys-4 board. Several types of PMOD compatible modules are available from simple push buttons to more complex modules like accelerometers, gyroscopes, thermometers, ADC and DAC converters [104].

The adjustable parameters of the wind turbine model can be configured by using the USB-UART Bridge of the Nexys-4 board. Since the input of the HIL device should be an analogue voltage proportional to the wind speed and the output should be an analogue voltage proportional to the torque, AD and DA conversion is required. Using this configuration real miniature wind sensor and generator can be connected to the HIL.

The analog to digital conversion was performed by the PMOD AD1 and the digital to analog conversion was performed by the PMOD DA2 module. The PMOD AD1 module uses a 6- pin connector header and it is built up from 2 Analog Devices AD747612bit ADCs [105]. It canconvert signals at a maximum sampling rate of 1MS/s. The PMOD DA2 also uses a 6 pin connector header and the digital-analog conversion was performed by using 2 National Semiconductor DAC121S101 12bit DACs [106]. Each module uses SPI interface for the communication with the FPGA, and converts the 0-3.3V voltage into digital signals and vice-versa [104].

4.6.2 Wind Turbine model

Two different and still similar models have been implemented in MATLAB/Simulink and on the Artix-7 FPGA. The first model is based on the normalized wind turbine model of the SimPowerSystems/Specialized Technology/Renewable Energy Systems library of MATLAB [107], which has a generator connected to the turbine. The output of the system is the torque applied to the generator shaft in per unit of the generator ratings. The second model is based on a standalone wind turbine model [108], without the generator. This is a non-normalized model, so the output is the torque on the shaft of the turbine. The mathematical description of the models is similar, so they can be considered as one model, in different aspects.

The frequently used mathematical models are based on the kinetic energy of the wind, because in that way a dynamically configurable system is achievable. The kinetic energy contained in a mass m (kg) of the moving air E_k can be calculated using 4.35.

$$E_k = \frac{1}{2}mv^2 \tag{4.35}$$

Where v denotes the wind speed (m/s). From 4.35 the total mechanical power P_T (W) in the mass of moving air E_k can be obtained as follows 4.36:

$$P_T = \frac{1}{2}\rho A v^3 \tag{4.36}$$

where ρ represents air density (kg/m 3) and A is the area swept by the blades (m^2). The amount of energy extracted from the wind depends on the aerodynamics of the rotor blade. The ratio between the total mechanical power and the extracted mechanical power is defined by the power coefficient C_P . From this the extracted mechanical power P can be obtained as 4.37.

$$P = \frac{1}{2}\rho A v^3 C_p \tag{4.37}$$

The power coefficient is given by the manufacturers and it is a function of the rotational speed of the turbine, wind speed and pitch angle of the blades. Generally the rotational speed and the wind speed are combined into the Tip Speed Ratio (TSR) represented as λ 4.38.

$$\lambda = \frac{R\omega}{v} \tag{4.38}$$

Where R represents the length of the rotor blades (m) and ω denotes the mechanical angular velocity (rad/s), what can be calculated as 4.39.

$$\omega = \frac{2\pi n}{60} \tag{4.39}$$

Where n is the rotational speed (r/min). A can be substituted by $R^2\pi$. Therefore the equation for the torque T can be obtained as 4.40 by dividing 4.37 with 4.39.

$$T = \frac{0.5\rho\pi R^2 v^3 C_p}{\omega} \tag{4.40}$$

Another approach for the torque is (4.41).

$$T = 0.5\rho\pi R^3 v^2 C_q \tag{4.41}$$

Where C_q represents the torque coefficient defined as 4.42.

$$C_q = \frac{C_P}{\lambda} \tag{4.42}$$

The power coefficient is the function of the blade pitch angle represented as β and the TSR, and can be obtained as follows 4.43:

$$C_P(\lambda, \beta) = c_1(c_2 - c_3\beta - c_4)e^{-c_5} + c_6\lambda \tag{4.43}$$

The general parameters are [109]:

- $c_1 = 0.5176$
- $c_2 = \frac{116}{\lambda_i}$
- $c_3 = 0.4$
- $c_4 = 5$
- $c_5 = \frac{21}{\lambda_i}$
- $c_6 = 0.0068$

Where λ_i can be written as follows 4.44:

$$\frac{1}{\lambda_i} = \frac{1}{\lambda + 0.08\beta} - \frac{0.035}{\beta^3 + 1} \tag{4.44}$$

The theoretical maximum of the power coefficient is 0.59 referred as the Betz limit, but the practical values vary from 0.2 to 0.5.The normalized wind turbine model is built from 4.37 and can be obtained as follows 4.45 [107]:

$$P_{pu} = k_p v_{pu}^3 C_{Ppu} \tag{4.45}$$

where:

- P_{pu} is the power of nominal power for particular values of air density and swept area in per unit

Table 4.10: *Input/Output Variables of the Models*

	Variable	Description	Unit
Normalized Model	v	Wind Speed	m/s
	β	Blade pitch angle	degrees
	$omega_{gen}$	Angular velocity	rad/s
	T_{pu}	Generator shaft	Nm
	v	Wind speed	m/s
Non-Normalized Model	β	Blade pitch angle	degrees
	R	Blade Radius	m
	ρ	Air Density	kg/m^3
	T	Turbine torque	Nm

- C_{Ppu} is the power coefficient of the maximum value of C_P in per unit
- v_{pu} is the wind speed of the base wind speed in per unit. The base wind speed is the mean value of the expected wind speed in m/s.
- k_p is the power gain for $C_{Ppu} = 1$ per unit, and $v_{pu} = 1$ per unit. The k_p is less than or equal to 1.

Equation 4.41 is used for the second, non-normalized model.

4.6.3 FPGA implementation

First the mathematical models were implemented in MATLAB/Simulink to verify their usability. Table 4.10 shows the inputs and outputs of the corresponding models.To avoid damage and unnecessary energy loss, wind turbines have mechanical breaks to stop the rotor when wind speed is lower than the cut-in speed and higher than the cut-out speed. The cut-in and cut-out wind speed thresholds and the angular velocity of the turbine are predefined inner parameters in the non-normalized case.

Xilinx System Generator

Engineers who use model-based design to target Xilinx FPGAs can use the Xilinx System Generator for DSP as a Simulink plug-in. This tool provides an easy graphical way to implement complex models using FPGA specific blocks collected into a Xilinx Library of the Simulink. This library contains general (add/sub, multiplication, divider, MATLAB- code, ROM block ...) and Xilinx specific (AXI-Stream, PicoBlaze, Vivado...) blocks. If the model based on the Xilinx blocks is completed, the System Generator either can be used to simulate the behavior of the FPGA-based model on the PC or co-simulate it on the real FPGA hardware. In addition to this, the

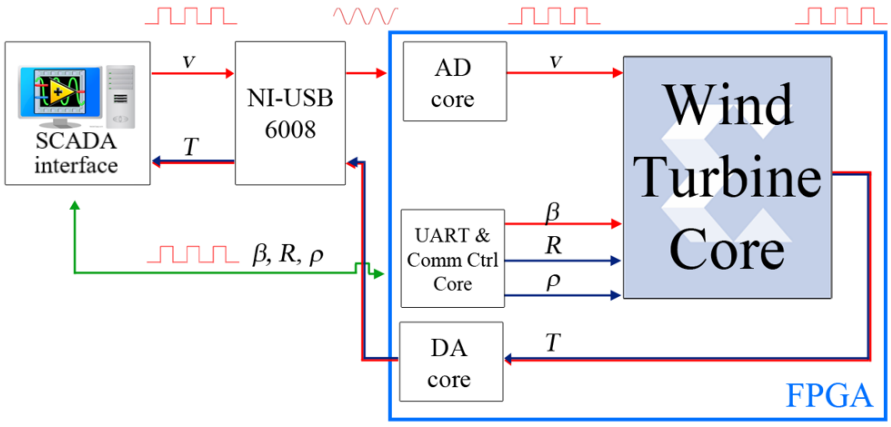

Figure 4.15: *The FPGA-based HIL system architecture*

model can be converted into a (Hardware Description Language) HDL-based native Xilinx ISE Development Kit project for further development or analysis [110]. The Simulink can be used to feed data to and from the System Generator model. Since the Simulink uses double precision fixed-point numbers which are not efficiently implementable on FPGAs, fixed-point data conversion is required. This can be done by using the Gateway In and Gateway Out blocks of the Xilinx Library.The wind turbine models were implemented using the Xilinx library blocks and also the standard Simulink blocks. Using the co-simulation possibilities the outputs of the models were compared to each other to verify the correctness of the FPGA-based hardware model.

Hardware-in-the-Loop architecture

The architecture of the FPGA-based HIL system connected to a PC is shown in Fig. 4.15. The system is built up from a *Wind Turbine Core*, an *AD Core*, a *DA Core*, and an *UART&Comm Ctrl Core*.

The Wind Turbine Core is responsible for the real time computation of the output torque based on the normalized or the non-normalized models according to the input parameters. The input parameters denoted by red are used in both models and parameters denoted by blue are used only in the non- normalized model.

The AD and DA Core communicate with the PMOD AD1 and PMOD DA2 modules via SPI communication, which send and receive the input wind speed and output torque using analogue signals. The UART&Comm Ctrl Core enables an 8- bit serial UART communication to send digital commands and data to, or receive digital data from the FPGA HIL device. This core provides dynamic input parameter changes while the FPGA is operating. An UART state machine was built in the UART&Comm Ctrl Core in VHDL (Very High Speed Integrated Circuit Hardware Description Lan-

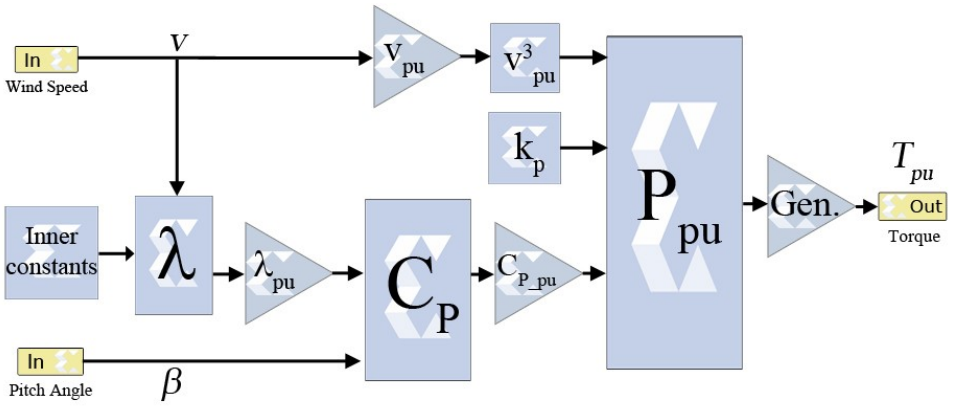

Figure 4.16: *The Normalized Wind Turbine model architecture*

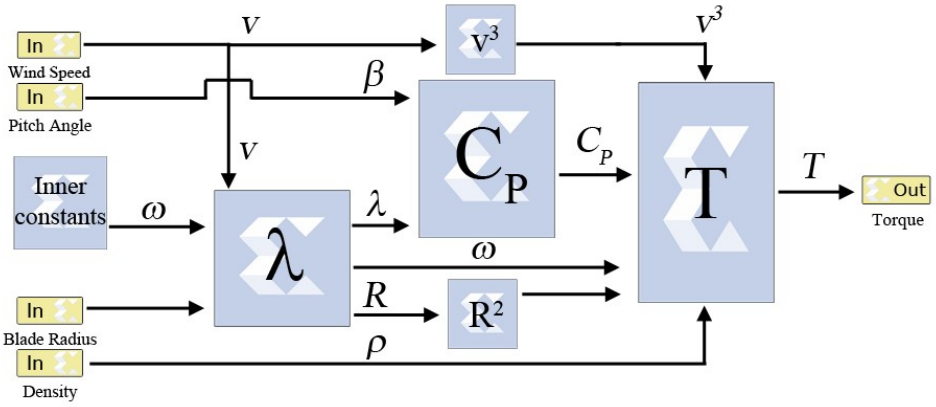

Figure 4.17: *Non-Normalized Wind Turbine model architecture*

guage), to control the simple UART RX/TX VHDL modules. The communication begins with an exclamation mark, and then the type of the request is sent together with the register address and the data from the SCADA interface, which based on LabView. The 8-bit request can be a write or read register, the register address is 1 byte, and the data are 2 byte fix-point type.

The architecture of the normalized and non-normalized wind turbine models are shown in Fig. 4.16 and Fig. 4.17.

The main parts of the models are:

- Computation of λ
- Computation of C_p
- Computation of the torque

The computation of the Tip Speed Ratio (TSR) is made in the λ part. It is important, because the functions in the C_P part require the value of λ.At the end of

Table 4.11: *Input Ranges*

Inputs	Input Name	Unit	Range	Word length (bit)
v	Wind Speed	m/s	0-15	10, (fraction 4)
β	Pitch Angle	$degree$	0-30	10, (fraction 4)
R	Blade Length	$meter$	0-64	12, (fraction 4)
ρ	Air Density	kg/m^3	0-1000	15, (fraction 4)

the architecture the value of the torque is computed in both cases.In the normalized model it is necessary to multiply the output of the λ, C_P, and torque parts with constants in order to get the output torque in per unit of the generator parameters.

Because of the limited resources of the FPGA it is required to exactly define the input parameter ranges as it can be seen in Table 4.11. The input ranges define the computation time, the resource requirement and the latency of the architecture. During the implementation of the Wind Turbine Cores the following three problems have to be solved:

- Synchronization of the divider blocks
- Exponential function
- Timing errors

Either in the normalized or in the non-normalized model division is required. Using this operation in MATLAB or Simulink is simple, but to implement it in an FPGA circuit is more complicated. The divider block in the Xilinx System Generator uses the High Radix algorithm to compute the result of the division, which is an iterative method. Due to this, the output of a divider block is fluctuating till the appropriate result has been computed. In case of one or two dividers this problem can be solved with a well-planned latency value configured in the divider core, but using two or more dividers the system is going to be useless. The best solution for the problem is to use the divider blocks with the enable and the data ready ports. With these ports the data transfer between the blocks can be synchronized. Another problem appears when the input word length of the divider block is too large, since the sum of the length of input word must be less or equal to 64-bit. So, it is required to optimize the inputs data width.In case of the wind turbine model, allof the inputs are fix-point 8-bitintegers with 4 fraction bits, so the size of the inputs is 12 bits.

In case of both models the exponent function of the C P part should be computed, but this function is not implemented for FPGAs in the Xilinx System Generator. This

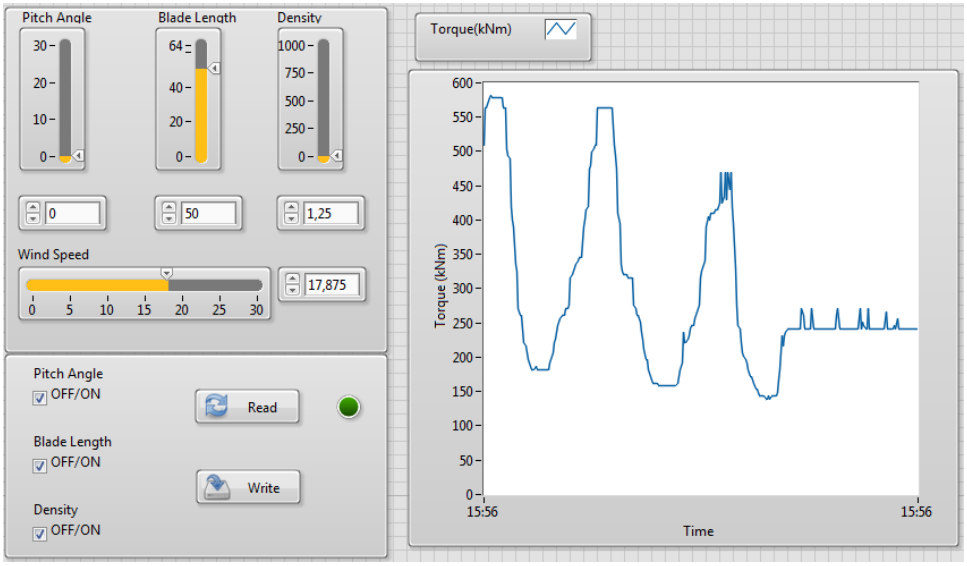

Figure 4.18: *The SCADA interface of the Wind Turbine HIL system, which is based on LabView*

problem can be solved with using a ROM block with pre-calculated exponential function values.The ROM block automatically reserves the required amount of BRAM memory resources of the FPGA, depending on the required precision and interval.

It is not achievable to store all the possible exponential values with any precision, therefore the required precision and interval should be determined and optimized to lower the memory requirement of the design.

During the implementation phase of the designs the timing constraints are not met, because the design contains high latency data paths in case of the divider and multiplier blocks. To eliminate this error, delay registers should be added to the critical paths too.

An interface was built in LabVIEW to realize analog and digital communication between the PC and the FPGA- basedHIL device. The interface enables the use of UART serial communication with the built-in VISA driver, and analog communication via the NI-USB6008 device. Using this interface the pitch angle, the blade length, the air density, and the wind speed can be set using 4 different sliders, and the output torque of the model can be examined on a waveform graph. The interface is shown in Fig. 4.18.

4.6.4 Results

The normalized and the non-normalized model were tested either using Simulink-based simulation or real FPGA hardware. During the test the pitch angle was 0, the blade length was $50\ m$, the air density was $1.25kg\ /m^3$, the angular velocity was

$1.46\ rad/s$ and the wind speed was changed linearly between $3\ m/s$ and $15\ m/s$.

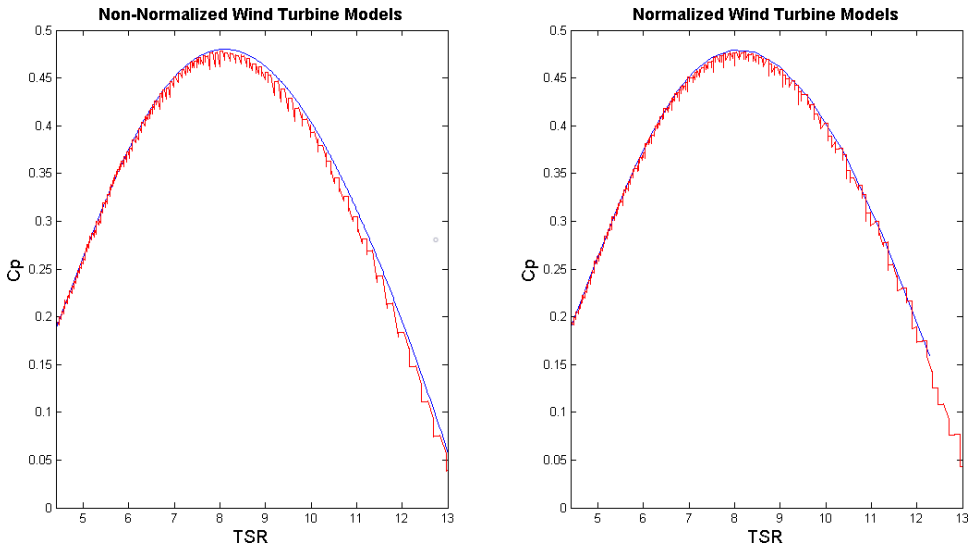

Figure 4.19: *Cp-TSR graphs of the models*

The best way to describe the behavior of a wind turbine is the C p –TSR characteristic, which plots the extracted power from the wind in the function of the TSR. The comparison of the C p – TSR curves in case of the Simulink and FPGA-based models can be seen in Fig. 4.19. The characteristics of the FPGA- based models are denoted by red, while the characteristics of the Simulink simulation are denoted by blue. During the tests the wind speed was increased with 0.01 steps. The FPGA- based curves are stepped, because the divider blocks use the High Radix algorithm.In Fig. 4.19 it can be seen the FPGA and the Simulink-based characteristics are very similar, so the HIL system works correctly for the two models.Below $3\ m/s$ wind speed none of the models were working correctly, but in general the wind turbine systems work with $3\ m/s$ cut-in wind speed.

The resource requirement of the models is shown in Table 4.12 The BRAM and the DSP Slice requirements are the main bottlenecks of the implementation.

The DSP Slice requirement is the function of the number of the add/sub and multiplication/divider blocks, while the BRAM requirement is the function of the required precision and interval of the exponential function. The bottleneck of the implementation is the number of BRAM and DSP Slices.

Improving the output precision or the input parameter range of the model it is required to increase the precision or the interval of the stored e x function which increase the BRAM requirement. So an optimum should be found between the precision and the input parameter range to fit the design into the Artix-7 FPGA-based Nexys-4 board. This can be found if the wind speed is changing between $3\ m/s$ and $15\ m/s$. In this case an acceptable computing precision can be reached.

Table 4.12: *Resource Requirements of the Models*

Resource Type	**Name**	**Normalized**	**Usage**	Non-normalized	Usage
General resources	Slices	2405	3%	2306	3%
	6-input LUT	5959	9%	5574	8%
	Flip-Flops	4484	3%	4380	3%
Dedicated resources	36Kbit BRAM	84	62%	84	62%
	DSP48E1	170	70%	164	68%

4.6.5 Conclusion

In this work a horizontal three-bladed variable pitch wind turbine model was presented and implemented on a Digilent Nexys-4 board containing Xilinx Artix-7 FPGA as a HIL device using Xilinx System Generator. In the normalized model the wind speed is an analog adjustable input parameter converted to digital data with the AD core and interface, while the pitch angle of the blades of the wind turbine is a digital adjustable input parameter. The output is the torque applied to the generator shaft in per unit of the generator ratings converted with the DA core and interface into analog voltage. The non- normalized model has another two changeable digital input parameters: blade length, and air density. These parameters can be setup with serial and analog communication using a SCADA interface. This FPGA-based HIL system with the SCADA interface can be used not only for experimental but for education purposes as well. Students can analyze the operation of the wind turbine, and study the effect of different parameter changes, without the risk of damaging or endangering the real physical system.

The characteristics of the PC and FPGA-based models were compared and it can be concluded that the output of the FPGA- based models is similar compared to the Simulink-based simulations. The results showed that a three bladed variable pitch wind turbine can be implemented on an FPGA and used as a real-time dynamically adjustable HIL system. In the future, this HIL system could be used for power output estimation of various generator models, comparison of different controlling methods or even a real miniature wind sensor and generator could be connected to it.

4.7 Discussion and concluding remarks

In Sections 4.3 - 4.6 signal processing algorithms are successfully applied to FPGAs in order to accelerate their operation making them real-time. Based on the results it can be concluded that FPGA-based signal processing applications in embedded systems are power, performance and size efficient compared to the same algorithms running on CPU. Furthermore the presented applications show that FPGAs are suitable for fast prototyping due to their reprogramming flexibility and can be used in diverse computer science research areas.

4.8 Contributions

The author of this PhD thesis is responsible for all the contributions presented in this chapter, which include the following main novelties:

III / 1. I proposed a new parallel cost-effective FPGA architecture, which capable of operating the Principal Component Analysis (PCA) algorithm in real-time.

III / 2. I gave a new parallel cost-effective FPGA architecture of sensor fusion, which is able to estimate position and orientation in real-time using the measurements of multiple sensors. I proposed an indoor localization solution, which allows the incorporation of information regarding the absolute position into the sensor fusion architecture. Furthermore, I gave an FPGA architecture of the two-dimensional multilateral algorithm.

III / 3. I developed a reliable, cost-effective method, which uses a minimal number of sensors to effectively control the active suspension system of a vehicle. I gave a new parallel FPGA architecture based on an actual observer, which controls the active suspensions force. Furthermore, I showed the real-time functionality of the FPGA architecture.

III / 4. I gave a new parallel FPGA architecture, which is capable of the real-time hardware emulation of a wind turbine in a hardware-in-the-loop (HIL) system. I developed a SCADA interface, which can manipulate the parameters of the system in real-time. Furthermore, I showed the real-time functionality of the FPGA-based hardware emulated wind turbine HIL system.

Bibliography

[1] neuronexus neural probe. Neuronexus neural probes. http://neuronexus.com/products/neural-probes, 2019.

[2] Jozsef Csicsvari et al. Massively parallel recording of unit and local field potentials with silicon-based electrodes. *Journal of Neurophysiology*, 90(2):1314–1323, 2003. doi: 10.1152/jn.00116.2003. URL https://doi.org/10.1152/jn.00116.2003. PMID: 12904510.

[3] Antal Berényi et al. Large-scale, high-density (up to 512 channels) recording of local circuits in behaving animals. *Journal of Neurophysiology*, 111(5):1132–1149, 2014. doi: 10.1152/jn.00785.2013. URL https://doi.org/10.1152/jn.00785.2013. PMID: 24353300.

[4] Vijay Viswam et al. Optimal electrode size for multi-scale extracellular-potential recording from neuronal assemblies. *Frontiers in Neuroscience*, 13, 2019.

[5] George Dimitriadis et al. Why not record from every channel with a cmos scanning probe? *bioRxiv*, 2019. doi: 10.1101/275818. URL https://www.biorxiv.org/content/early/2019/02/09/275818.

[6] Richárd Fiáth et al. Fine-scale mapping of cortical laminar activity during sleep slow oscillations using high-density linear silicon probes. *Journal of Neuroscience Methods*, 316:58–70, 2019.

[7] James Jun et al. Fully integrated silicon probes for high-density recording of neural activity. *Nature*, 551:232–236, 2017.

[8] Shigeyoshi Fujisawa et al. Behavior-dependent short-term assembly dynamics in the medial prefrontal cortex. *Nature Neuroscience*, 11:823–833, 2008.

[9] L. Hochberg et al. Neuronal ensemble control of prosthetic devices by a human with tetraplegia. *Nature*, 442:164–171, 2006.

[10] Michael S. Lewicki. A review of methods for spike sorting: the detection and classification of neural action potentials. *Network: Computation in Neural Systems*, 9(4):R53–R78, jan 1998. doi: 10.1088/0954-898x_9_4_001. URL https://doi.org/10.1088/0954-898x_9_4_001.

[11] Carolina M. Lopez et al. An implantable 455-active-electrode 52-channel cmos neural probe. *IEEE Journal of Solid-State Circuits*, 49(1):248–261, 2014. doi: 10.1109/JSSC.2013.2284347.

[12] T. Blanche et al. Polytrodes: high-density silicon electrode arrays for large-scale multiunit recording. *Journal of neurophysiology*, 93 5:2987–3000, 2005.

[13] C. Gold et al. On the origin of the extracellular action potential waveform: A modeling study. *Journal of neurophysiology*, 95 5:3113–28, 2006.

[14] Scott B. Wilson and Ronald Emerson. Spike detection: a review and comparison of algorithms. *Clinical Neurophysiology*, 113(12):1873–1881, 2002. ISSN 1388-2457. doi: https://doi.org/10.1016/S1388-2457(02)00297-3. URL https://www.sciencedirect.com/science/article/pii/S1388245702002973.

[15] Bo Yu et al. Real-time fpga-based multichannel spike sorting using hebbian eigenfilters. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 1(4):502–515, 2011. doi: 10.1109/JETCAS.2012.2183430.

[16] Florentin Wörgötter et al. An on-line spike form discriminator for extracellular recordings based on an analog correlation technique. *Journal of Neuroscience Methods*, 17(2):141–151, 1986. ISSN 0165-0270. doi: https://doi.org/10.1016/0165-0270(86)90067-1. URL https://www.sciencedirect.com/science/article/pii/0165027086900671.

[17] Z. Nenadic and J.W. Burdick. Spike detection using the continuous wavelet transform. *IEEE Transactions on Biomedical Engineering*, 52(1):74–87, 2005. doi: 10.1109/TBME.2004.839800.

[18] K. G. Parthiban and S. Vijayachitra. Spike detection from electroencephalogram signals with aid of hybrid genetic algorithm-particle swarm optimization. *Journal of Medical Imaging and Health Informatics*, 5 (5):936–944, 2015. ISSN 2156-7018. doi: doi:10.1166/jmihi.2015.1480. URL https://www.ingentaconnect.com/content/asp/jmihi/2015/00000005/00000005/art00007.

[19] Sunghan Kim and James McNames. Automatic spike detection based on adaptive template matching for extracellular neural recordings. *Journal of*

neuroscience methods, 165(2):165—174, September 2007. ISSN 0165-0270. doi: 10.1016/j.jneumeth.2007.05.033. URL https://doi.org/10.1016/j.jneumeth.2007.05.033.

[20] Hamed Azami et al. Extracellular spike detection from multiple electrode array using novel intelligent filter and ensemble fuzzy decision making. *Journal of Neuroscience Methods*, 239:129–138, 2015. ISSN 0165-0270. doi: https://doi.org/10.1016/j.jneumeth.2014.10.006. URL https://www.sciencedirect.com/science/article/pii/S0165027014003665.

[21] Mohammad Hossein Zarifia et al. A new evolutionary approach for neural spike detection based on genetic algorithm. *Expert Systems with Applications*, 42(1):462–467, 2015. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2014.07.038. URL https://www.sciencedirect.com/science/article/pii/S095741741400445X.

[22] J. Neto et al. Validating silicon polytrodes with paired juxtacellular recordings: method and dataset. *Journal of Neurophysiology*, 116:892 – 903, 2016.

[23] Espen Hagen et al. Visapy: A python tool for biophysics-based generation of virtual spiking activity for evaluation of spike-sorting algorithms. *Journal of Neuroscience Methods*, 245:182–204, 2015.

[24] Marius Pachitariu et al. Kilosort: realtime spike-sorting for extracellular electrophysiology with hundreds of channels. *bioRxiv*, 2016.

[25] James Jun et al. Real-time spike sorting platform for high-density extracellular probes with ground-truth validation and drift correction. *bioRxiv*, 2017.

[26] Richárd Fiáth et al. A silicon-based neural probe with densely-packed low-impedance titanium nitride microelectrodes for ultrahigh-resolution in vivo recordings. *Biosensors & bioelectronics*, 106:86–92, 2018.

[27] XZCU7EV. Xilinx zcu106. https://www.xilinx.com/products/boards-and-kits/zcu106.html, 2021.

[28] G. Buzsáki and K. Mizuseki. The log-dynamic brain: how skewed distributions affect network operations. *Nature Reviews Neuroscience*, 15:264–278, 2014.

[29] Klas H. Pettersen and G. Einevoll. Amplitude variability and extracellular low-pass filtering of neuronal spikes. *Biophysical journal*, 94 3:784–802, 2008.

[30] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8:3–30, 1998.

[31] Sarah Gibson et al. Technology-aware algorithm design for neural spike detection, feature extraction, and dimensionality reduction. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 18(5):469–478, 2010. doi: 10.1109/TNSRE.2010.2051683.

[32] J.F. Kaiser. Some useful properties of teager's energy operators. In *1993 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages 149–152 vol.3, 1993. doi: 10.1109/ICASSP.1993.319457.

[33] S. Mukhopadhyay and G.C. Ray. A new interpretation of nonlinear energy operator and its efficacy in spike detection. *IEEE Transactions on Biomedical Engineering*, 45(2):180–187, 1998. doi: 10.1109/10.661266.

[34] Kenneth D. Harris et al. Accuracy of tetrode spike separation as determined by simultaneous intracellular and extracellular measurements. *Journal of Neurophysiology*, 84(1):401–414, 2000. doi: 10.1152/jn.2000.84.1.401. URL https://doi.org/10.1152/jn.2000.84.1.401. PMID: 10899214.

[35] Hernan Gonzalo Rey et al. Past, present and future of spike sorting techniques. *Brain Research Bulletin*, 119:106–117, 2015. ISSN 0361-9230. doi: https://doi.org/10.1016/j.brainresbull.2015.04.007. URL https://www.sciencedirect.com/science/article/pii/S0361923015000684. Advances in electrophysiological data analysis.

[36] Sarah Gibson et al. An fpga-based platform for accelerated offline spike sorting. *Journal of Neuroscience Methods*, 215:1–11, 2013.

[37] Sarah Gibson et al. Comparison of spike-sorting algorithms for future hardware implementation. In *2008 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 5015–5020, 2008. doi: 10.1109/IEMBS.2008.4650340.

[38] Bogdan C. Raducanu et al. Time multiplexed active neural probe with 1356 parallel recording sites. *Sensors*, 17(10), 2017. ISSN 1424-8220. doi: 10.3390/s17102388. URL https://www.mdpi.com/1424-8220/17/10/2388.

[39] Richárd Fiáth et al. Large-scale recording of thalamocortical circuits: in vivo electrophysiology with the two-dimensional electronic depth control silicon probe. *Journal of neurophysiology*, 116 5:2312–2330, 2016.

[40] Dawn M. Taylor et al. Direct cortical control of 3d neuroprosthetic devices. *Science*, 296(5574):1829–1832, 2002. ISSN 0036-8075. doi: 10.1126/science.1070291. URL https://science.sciencemag.org/content/296/5574/1829.

[41] Ying-Lun Chen et al. An efficient vlsi architecture for multi-channel spike sorting using a generalized hebbian algorithm. *Sensors (Basel, Switzerland)*, 15:19830 – 19851, 2015.

[42] H. Chen et al. An efficient hardware circuit for spike sorting based on competitive learning networks. *Sensors (Basel, Switzerland)*, 17, 2017.

[43] S. Luan et al. Compact standalone platform for neural recording with real-time spike sorting and data logging. *Journal of neural engineering*, 15 4:046014, 2018.

[44] Jongkil Park et al. A 128-channel fpga-based real-time spike-sorting bidirectional closed-loop neural interface system. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 25(12):2227–2238, 2017. doi: 10.1109/TNSRE.2017.2697415.

[45] R. Hampson et al. A wireless recording system that utilizes bluetooth technology to transmit neural activity in freely moving animals. *Journal of Neuroscience Methods*, 182:195–204, 2009.

[46] Wen-Jyi Hwang et al. Efficient architecture for spike sorting in reconfigurable hardware. *Sensors (Basel, Switzerland)*, 13:14860 – 14887, 2013.

[47] Cyrille Rossant et al. Spike sorting for large, dense electrode arrays. *Nature neuroscience*, 19:634 – 641, 2016.

[48] P. Yger et al. Fast and accurate spike sorting in vitro and in vivo for up to thousands of electrodes. *bioRxiv*, 2016.

[49] U. Rutishauser et al. Online detection and sorting of extracellularly recorded action potentials in human medial temporal lobe recordings, in vivo. *Journal of Neuroscience Methods*, 154:204–224, 2006.

[50] V. Karkare et al. A 75μw, 16-channel neural spike-sorting processor with unsupervised clustering. *2011 Symposium on VLSI Circuits - Digest of Technical Papers*, pages 252–253, 2011.

[51] Daniel Valencia and Amirhossein Alimohammad. A real-time spike sorting system using parallel osort clustering. *IEEE Transactions on Biomedical Circuits and Systems*, 13(6):1700–1713, 2019. doi: 10.1109/TBCAS.2019.2947618.

[52] Daniel Valencia and Amirhossein Alimohammad. An efficient hardware architecture for template matching-based spike sorting. *IEEE Transactions on Biomedical Circuits and Systems*, 13(3):481–492, 2019. doi: 10.1109/TBCAS.2019.2907882.

[53] Jelena Dragas et al. Complexity optimization and high-throughput low-latency hardware implementation of a multi-electrode spike-sorting algorithm. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 23(2):149–158, 2015. doi: 10.1109/TNSRE.2014.2370510.

[54] Mora Lopez et al. A neural probe with up to 966 electrodes and up to 384 configurable channels in 0.13 μm soi cmos. *IEEE Transactions on Biomedical Circuits and Systems*, 11(3):510–522, 2017. doi: 10.1109/TBCAS.2016.2646901.

[55] Jörg Scholvin et al. Close-packed silicon microelectrodes for scalable spatially oversampled neural recording. *IEEE Transactions on Biomedical Engineering*, 63(1):120–130, 2016. doi: 10.1109/TBME.2015.2406113.

[56] P. Barthó et al. Characterization of neocortical principal cells and interneurons by network interactions and extracellular features. *Journal of neurophysiology*, 92 1:600–8, 2004.

[57] Richárd Fiáth et al. Slow insertion of silicon probes improves the quality of acute neuronal recordings. *Scientific Reports*, 9, 2019.

[58] László Schäffer et al. Fpga-based real-time multichannel neural dataset generation. In *2017 European Conference on Circuit Theory and Design (ECCTD)*, pages 1–4, 2017. doi: 10.1109/ECCTD.2017.8093235.

[59] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 2nd edition, 1979.

[60] R. Quiroga et al. Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering. *Neural Computation*, 16:1661–1687, 2004.

[61] Raj Talluri and J. K. Aggarwal. *Position Estimation Techniques for an Autonomous Mobile Robot: A Review*, page 769–801. World Scientific Publishing Co., Inc., USA, 1993. ISBN 9810211368.

[62] G.N Desouza. and A.C. Kak. Vision for mobile robot navigation: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(2):237–267, 2002. doi: 10.1109/34.982903.

[63] S. Y. Chen. Kalman filter for robot vision: A survey. *IEEE Transactions on Industrial Electronics*, 59(11):4409–4420, 2012. doi: 10.1109/TIE.2011.2162714.

[64] Bart L. J. Gysen et al. Design aspects of an active electromagnetic suspension system for automotive applications. In *2008 IEEE Industry Applications Society Annual Meeting*, pages 1–8, 2008. doi: 10.1109/08IAS.2008.181.

[65] Mohammad Ali Nekoui and Parisa Hadavi. Optimal control of an active suspension system. In *Proceedings of 14th International Power Electronics and Motion Control Conference EPE-PEMC 2010*, pages T5–60–T5–63, 2010. doi: 10.1109/EPEPEMC.2010.5606776.

[66] WWEA. Wwea 2014 half-year report. `http://www.wwindea.org`, 2014.

[67] Charles Z. Liu and Manolya Kavakli. Extensions of principle component analysis with applications on vision based computing. *Multimedia Tools and Applications*, 75(17):10113–10151, Sep 2016. ISSN 1573-7721. doi: 10.1007/s11042-015-3025-3. URL `https://doi.org/10.1007/s11042-015-3025-3`.

[68] Jian Yang and D. Zhang et al. Two-dimensional pca: a new approach to appearance-based face representation and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(1):131–137, 2004. doi: 10.1109/TPAMI.2004.1261097.

[69] Rajkiran Gottumukkal and Vijayan K. Asari. An improved face recognition technique based on modular pca approach. *Pattern Recognition Letters*, 25 (4):429–436, 2004. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2003.11.005. URL `https://www.sciencedirect.com/science/article/pii/S0167865503002654`.

[70] K. G. Smitha and A. P. Vinod. Low complexity fpga implementation of emotion detection for autistic children. In *2013 7th International Symposium on Medical Information and Communication Technology (ISMICT)*, pages 103–107, 2013. doi: 10.1109/ISMICT.2013.6521709.

[71] A.R. Mohan et al. An embedded face recognition system on a vlsi array architecture and its fpga implementation. In *2008 34th Annual Conference of IEEE Industrial Electronics*, pages 2432–2437, 2008. doi: 10.1109/IECON.2008.4758338.

[72] Ameni Y. Jammoussiet al. Implementation of face recognition system in virtex ii pro platform. In *2009 3rd International Conference on Signals, Circuits and Systems (SCS)*, pages 1–6, 2009. doi: 10.1109/ICSCS.2009.5412313.

[73] Milton E. Conde et al. An efficient data fusion architecture for infrared and ultrasonic sensors, using fpga. In *2013 IEEE 4th Latin American Symposium on Circuits and Systems (LASCAS)*, pages 1–4, 2013. doi: 10.1109/LASCAS.2013.6519059.

[74] Gokhan K. Gultekin and Afsar Saranli. An fpga based high performance optical flow hardware design for computer vision applications. *Microprocessors*

and Microsystems, 37(3):270–286, 2013. ISSN 0141-9331. doi: https://doi.org/10.1016/j.micpro.2013.01.001. URL https://www.sciencedirect.com/science/article/pii/S0141933113000033.

[75] Szabolcs Hajdu et al. Complementary filter based sensor fusion on fpga platforms. In *2017 International Conference on Optimization of Electrical and Electronic Equipment (OPTIM) 2017 Intl Aegean Conference on Electrical Machines and Power Electronics (ACEMP)*, pages 851–856, 2017. doi: 10.1109/OPTIM.2017.7975076.

[76] H.-P. Brückner et al. Energy-efficient inertial sensor fusion on heterogeneous fpga-fabric/risc system on chip. In *2013 Seventh International Conference on Sensing Technology (ICST)*, pages 506–511, 2013. doi: 10.1109/ICSensT.2013.6727704.

[77] S. Chappell et al. Exploiting real-time fpga based adaptive systems technology for real-time sensor fusion in next generation automotive safety systems. In *Design, Automation and Test in Europe*, pages 180–185 Vol. 3, 2005. doi: 10.1109/DATE.2005.147.

[78] Janosch Nikolic et al. A synchronized visual-inertial sensor system with fpga pre-processing for accurate real-time slam. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 431–437, 2014. doi: 10.1109/ICRA.2014.6906892.

[79] Simone Sabatelli et al. A double-stage kalman filter for orientation tracking with an integrated processor in 9-d imu. *IEEE Transactions on Instrumentation and Measurement*, 62(3):590–598, 2013. doi: 10.1109/TIM.2012.2218692.

[80] Vanderlei Bonato et al. An fpga implementation for a kalman filter with application to mobile robotics. In *2007 International Symposium on Industrial Embedded Systems*, pages 148–155, 2007. doi: 10.1109/SIES.2007.4297329.

[81] Elnaz Akbari et al. Observer design for active suspension system using sliding mode control. In *2010 IEEE Student Conference on Research and Development (SCOReD)*, pages 207–212, 2010. doi: 10.1109/SCORED.2010.5704003.

[82] Abdolvahab Agharkakli et al. Simulation and analysis of passive and active suspension system using quarter car model for different road profile. *International Journal of Engineering Trends and Technology*, 3(5):636–644, 2012.

[83] Mohd Avesh and Rajeev Srivastava. Modeling simulation and control of active suspension system in matlab simulink environment. In *2012 Students Conference on Engineering and Systems*, pages 1–6, 2012. doi: 10.1109/SCES.2012.6199124.

[84] Abbas Chamseddine and Hassan Noura. Control and sensor fault tolerance of vehicle active suspension. *IEEE Transactions on Control Systems Technology*, 16 (3):416–433, 2008. doi: 10.1109/TCST.2007.908191.

[85] Koldo Basterretxea et al. A semi-active suspension embedded controller in a fpga. In *International Symposium on Industrial Embedded System (SIES)*, pages 69–78, 2010. doi: 10.1109/SIES.2010.5551387.

[86] Ammar A. Aldair and Weiji Wang. Fpga based adaptive neuro fuzzy inference controller for full vehicle nonlinear active suspension systems. *International Journal of Artificial Intelligence and Applications*, 1(4):1–15, October 2010. URL `http://sro.sussex.ac.uk/id/eprint/72594/`.

[87] R. Rana and B. Kumari. Fpga implementation of linear observer. *International Journal of Future Computer and Communication*, pages 14–17, 2014.

[88] Anne-Marie Lienhardt et al. Digital sliding-mode observer implementation using fpga. *IEEE Transactions on Industrial Electronics*, 54(4):1865–1875, 2007. doi: 10.1109/TIE.2007.898305.

[89] Phlearn Jansuya and Yuttana Kumsuwan. Design of matlab/simulink modeling of fixed-pitch angle wind turbine simulator. *Energy Procedia*, 34: 362–370, 2013. ISSN 1876-6102. doi: https://doi.org/10.1016/j.egypro.2013.06.764. URL `https://www.sciencedirect.com/science/article/pii/S1876610213010072`. 10th Eco-Energy and Materials Science and Engineering Symposium.

[90] Surya Santoso and Ha Thu Le. Fundamental time–domain wind turbine models for wind power studies. *Renewable Energy*, 32(14):2436–2452, 2007. ISSN 0960-1481. doi: https://doi.org/10.1016/j.renene.2006.12.008. URL `https://www.sciencedirect.com/science/article/pii/S0960148106003466`.

[91] Vinay Thapar et al. Critical analysis of methods for mathematical modelling of wind turbines. *Renewable Energy*, 36(11):3166–3177, 2011. ISSN 0960-1481. doi: https://doi.org/10.1016/j.renene.2011.03.016. URL `https://www.sciencedirect.com/science/article/pii/S0960148111001303`.

[92] C. Carrillo et al. Review of power curve modelling for wind turbines. *Renewable and Sustainable Energy Reviews*, 21:572–581, 2013. ISSN 1364-0321. doi: https://doi.org/10.1016/j.rser.2013.01.012. URL `https://www.sciencedirect.com/science/article/pii/S1364032113000439`.

[93] J.G. Slootweg et al. General model for representing variable speed wind turbines in power system dynamics simulations. *IEEE Transactions on Power Systems*, 18(1):144–151, 2003. doi: 10.1109/TPWRS.2002.807113.

[94] Chris Washington and Jordan Dolman. Creating next generation hil simulators with fpga technology. In *2010 IEEE AUTOTESTCON*, pages 1–6, 2010. doi: 10.1109/AUTEST.2010.5613618.

[95] Hao Chen et al. Dynamic simulation of dfig wind turbines on fpga boards. In *2010 Power and Energy Conference At Illinois (PECI)*, pages 39–44, 2010. doi: 10.1109/PECI.2010.5437160.

[96] A. Lemieux and M. Parizeau. Experiments on eigenfaces robustness. In *2002 International Conference on Pattern Recognition*, volume 1, pages 421–424 vol.1, 2002. doi: 10.1109/ICPR.2002.1044743.

[97] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 01 1991. ISSN 0898-929X. doi: 10.1162/jocn.1991.3.1.71.

[98] Hervé Abdi and Lynne J. Williams. Principal component analysis. *WIREs Computational Statistics*, 2(4):433–459, 2010. doi: https://doi.org/10.1002/wics.101. URL `https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wics.101`.

[99] Paul Viola and Michael J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, May 2004. ISSN 1573-1405. doi: 10.1023/B:VISI.0000013087.49260.fb. URL `https://doi.org/10.1023/B:VISI.0000013087.49260.fb`.

[100] Xilinx. Device reliability report, second half 2017. Technical report, Xilinx, 2018.

[101] Ádám Kaló et al. Indoor localization simulation framework for optimized sensor placement to increase the position estimation accuracy. *Annales Mathematicae et Informaticae*, 51:29–39, 2020.

[102] Abdelmoumen Norrdine. An algebraic solution to the multilateration problem. In *International Conference on Indoor Positioning and Indoor Navigation*, pages 1–4, 2012. doi: 10.13140/RG.2.1.1681.3602.

[103] Xilinx. Xilinx website. `http://www.xilinx.com`, 2019.

[104] Digilent. Digilent website. `http://www.digilentinc.com`, 2019.

[105] AD7476. Ad7476 datasheet. `http://www.analog.com`, 2019.

[106] DAC121S101. Dac121s101 datasheet. `http://www.ti.com`, 2019.

[107] windturbine. Implement model of variable pitch wind turbine. https://www.mathworks.com/help/physmod/sps/powersys/ref/windturbine.html, 2021.

[108] Florin Iov et al. *Wind Turbine Blockset in Matlab/Simulink. General Overview and Description of the Model*. Institut for Energiteknik, Aalborg Universitet, 2004. ISBN 8789179463.

[109] Siegfried Heier. *Wind Energy Conversion Systems*, chapter 2, pages 31–117. John Wiley & Sons, Ltd, 2014. ISBN 9781118703274. doi: https://doi.org/10.1002/9781118703274.ch2. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118703274.ch2.

[110] Kiran Kintali and Yongfeng Gu. Model-based design with simulink, hdl coder, and xilinx system generator for dsp. Technical report, Mathworks, 2012.

Summary

The PhD thesis presents FPGA-based real-time signal processing, including the detection and classification of activation potential in neurophysiological measurements, as well as other signal processing applications that can be performed in real time using FPGA. A common feature of the presented approaches is real-time implementation using FPGA.

The dissertation consists of three major parts. The first chapter presents real-time detection and synthesis of neurophysiological signals based on FPGA, the second chapter presents real-time classification of neurophysiological signals based on spatial information, while the third chapter presents the application of FPGA-based systems in signal processing and simulation.

FPGA-based real-time detection and synthesis of neurophysiological signals

Detection of activity (spike) in extracellular neural recordings is a critical step in classifying neural signals. In a high-channel microelectrode array, the recorded action potentials come from different neurons from different channels, resulting in a plethora of spikes (more than 10,000) in a matter of minutes. A large number of spikes has a huge impact on the resources used for typical subtraction and classification if spike detection works with a high false-positive ratio.

In Chapter 2, I implemented, tested, and compared several spike detection methods using MATLAB (NEO, square, AMPP, AMPN, AMPA, cross-correlation), and then selected and implemented an ideal (NEO) in terms of real-time operation and FPGA feasibility. On FPGA. To test the detection and classification, I developed a hybrid template-based neurophysiological signal generator in MATLAB and implemented it on FPGA, which randomly places spike templates on the electrode array (number of rows and columns can be varied), selects firing frequency and spatial firing crosstalk on electrodes. Thus, multi-channel detection and classification algorithms can be tested in software and FPGA-based multi-channel detection and classification circuits can be hardware validated. The chosen detection method achieves a present 100%

TP value with a 10-4 dB SNR in both MATLAB and real-time FPGA.

FPGA-based real-time classification of neurophysiological signals taking into account spatial information

The firing activity of individual neuronal cells can be separated from the recorded multi-activity signal using the detection of activation potentials by different spike classification methods. Processing a high-dimensional image can take a lot of time (a few minutes of data for several hours) if a general-purpose computer is used. However, using FPGA circuits, the processing time is equivalent to the time length of the recording. So you can give instant feedback to researchers during recording, which carries critical information as there is no retrospective error in inserting the electrode array.

In Chapter 3, I created an FPGA-based real-time spike classification system that takes into account spatial information about nearby electrodes and classifies multi-channel neurophysiological data. I modified the original Online Sorting algorithm (based on unattended template matching) in MATLAB and then implemented it on FPGA to be able to process several channels (in this case 128) at the same time, as well as on the close electrodes in the 3x3 window of the current firing. also uses crosstalk (spatial) information for classification. The developed FPGA-based architecture achieves an average accuracy of 86% on synthetic data (10-4 dB SNR), while the original single-channel OSort (separately per channel) averages 74%. The developed FPGA-based architecture was also tested on real data recorded on anesthetized rats, the result of which was compared with the result of the kiloSort (offline) classification algorithm, which was subsequently manually corrected by an expert. Based on cross-correlation, the developed FPGA-based architecture created similar classes in 80%.

Application of FPGA-based systems in the field of signal processing and simulation

FPGA-based systems are widely used for many signal processing and simulation tasks due to their real-time operation, easy reprogramming, and low cost. Thanks to rapid prototyping, algorithms used in different fields can be accelerated and real-time using FPGAs as long as the critical points of the algorithms can be parallelized. I show 4 examples of such prototyping, proving the flexibility and widespread use of FPGAs. Such areas include face recognition, sensor fusion, control technology or hardware-in-the-loop simulation, for which I have developed an FPGA-based circuit.

In the field of face recognition, I have created a cost-effective face recognition system based on PCA. In the field of sensor fusion, I developed a system using sensor fusion that can estimate position and orientation in real time. In the field of control technology, I created a real-time system for an active suspension, which regulates the amount of force to be used. In the field of HIL simulation, I created an FPGA architecture emulating the operation of a wind turbine, which was part of a LabView-based hardware-in-the-loop system.

Contributions of the thesis

In **first thesis group**, my contributions are related to FPGA-based real-time detection and synthesis of neurophysiological signals. Detailed discussion can be found in Chapter 2.

I / 1. I gave a Non-Linear Energy Operator (NEO) based multi-channel activation potential detecting FPGA architecture, which is also capable of determining the source of the firing neuron in a multi-channel electrode array. I showed that it can operate in real time.

I / 2. I proposed a cross-correlation based action potential detection method, showed its efficiency and pointed out its limitations.

I / 3. I designed a neurophysiological signal generator algorithm, which is capable of placing action potential templates on randomly selected channels with randomly determined frequency and firing crosstalk, taking into account the theory of neuron firing known in neuroscience.

I / 4. I gave an FPGA architecture, which is capable of generating neurophysiological signals using the Mersenne-Twister pseudo-random number generator algorithm. I showed that the architecture can generate real-time neurophysiological signals for an electrode array with 128 channels.

In the **second thesis group**, the contributions are related to the FPGA-based real-time classification of neurophysiological signals taking into account spatial information. Detailed discussion can be found in Chapter 3.

II / 1. I proposed the use of inter-channel spatial information in multi-channel neurophysiological recordings using the Online Sorting algorithm. I compared the efficiency of the original single-channel Online Sorting algorithm to the proposed algorithm on synthetic data. I showed that the proposed algorithm, which uses spatial information has better efficiency.

II / 2. I gave an FPGA architecture based on the Online Sorting algorithm capable of classifying multi-channel neurophysiological recordings in real-time. I showed the effectiveness of the architecture with tests based on synthetic data and real measurements. I showed the FPGA architecture can operate in real time and is suitable for immediate feedback during experiments.

In the **third thesis group**, the contributions are related to the application of FPGA-based systems in the field of signal processing and simulation. Detailed discussion can be found in Chapter 4.

III / 1. I proposed a new parallel cost-effective FPGA architecture, which capable of operating the Principal Component Analysis (PCA) algorithm in real-time.

III / 2. I gave a new parallel cost-effective FPGA architecture of sensor fusion, which is able to estimate position and orientation in real-time using the measurements of multiple sensors. I proposed an indoor localization solution, which allows the incorporation of information regarding the absolute position into the sensor fusion architecture. Furthermore, I gave an FPGA architecture of the two-dimensional multilateral algorithm.

III / 3. I developed a reliable, cost-effective method, which uses a minimal number of sensors to effectively control the active suspension system of a vehicle. I gave a new parallel FPGA architecture based on an actual observer, which controls the active suspensions force. Furthermore, I showed the real-time functionality of the FPGA architecture.

III / 4. I gave a new parallel FPGA architecture, which is capable of the real-time hardware emulation of a wind turbine in a hardware-in-the-loop (HIL) system. I developed a SCADA interface, which can manipulate the parameters of the system in real-time. Furthermore, I showed the real-time functionality of the FPGA-based hardware emulated wind turbine HIL system.

Összefoglalás

Az értekezés FPGA alapú valós idejű jelfeldolgozást ismertet, magába foglalva neurofiziológiai méréseken történő akciós potenciál detektálását és osztályozását, valamint FPGA segítségével valós időben elvégezhető egyéb jelfeldolgozási alkalmazásokat. A bemutatott megközelítések közös vonása az FPGA alkalmazásával történő valós idejű kivitelezés.

A munka három fő témakörből áll. Az első fejezetben a neurofiziológiai jelek FPGA alapú valós idejű detektálása és szintézise, a második fejezetben a neurofiziológiai jelek FPGA alapú valós idejű osztályozása térbeli információk figyelembevételével, míg a harmadik fejezetben FPGA alapú rendszerek jelfeldolgozási és szimulációs témakörben való alkalmazása kerül bemutatásra.

Neurofiziológiai jelek FPGA alapú valós idejű detektálása és szintézise

Az extracelluláris neurális felvételekben való aktivitás (tüske) detektálása egy kritikus lépés a neurális jelek osztályozásához. A magas csatornaszámú mikroelektróda tömbben a felvett akciós potenciálok különböző neuronokból származnak különböző csatornákról, emiatt rengeteg tüske keletkezik (több mint 10000) pár perc leforgása alatt. A nagy tüskeszám óriási kihatással van a jellemző kivonáshoz és osztályozáshoz használt erőforrásokra, amennyiben a tüskék detektálása nagy hamis-pozitív aránnyal működik.

A 2. fejezetben többféle tüske detektálási módszert implementáltam, teszteltem és összehasonlítottam MATLAB segítségével (NEO, négyzetes, AMPP, AMPN, AMPA, kereszt-korreláció), majd a valós idejű működés és az FPGA megvalósíthatóság szempontból ideálist (NEO) választottam ki és valósítottam meg FPGA-n. A detektálás és osztályozás teszteléséhez egy hibrid sablon alapú neurofiziológiás jelgenerátort fejlesztettem ki MATLAB-ban és implementáltam FPGA-ra, amely a tüske sablonokat véletlenszerűen helyezi el az elektródatömbön (sorok és oszlopok száma változtatható), választ tüzelési frekvenciát és térbeli tüzelési áthallást az egymás mellett lévő elektródákon. Így szoftveresen tesztelhetőek a több csatornán működni képes de-

tektáló és osztályozó algoritmusok és hardveresen validálhatóak az FPGA alapú szintén több csatornát feldolgozni képes detektáló és osztályozó áramkörök. A kiválasztott detektálási módszer 10-4 dB SNR-el rendelkező jelen 100% TP értéket ér el MATLAB-ban és valós időben FPGA-val egyaránt. Továbbá létrehoztam egy valós időben működő Nem-lináris Energia Operátor (NEO) alapú többcsatornás akciós potenciál detektáló FPGA architektúrát, amely képes az elektródák közötti áthallás felhasználásával a központi tüzelést tartalmazó csatornát meghatározni és validáltam a hatékonyságát a jelgenerátor felhasználásával.

Neurofiziológiai jelek FPGA alapú valós idejű osztályozása térbeli információk figyelembevételével

Az individuális neuron sejtek tüzelési aktivitását szeparálni lehet a felvett több aktivitást tartalmazó jelből az akciós potenciálok detektálását felhasználva különböző tüske osztályozási módszerekkel. A magas dimenziószámú felvétel feldolgozása rengeteg időt igényelhet (pár percnyi adat több óra), amennyiben általános célú számítógépet használnak hozzá. Azonban FPGA áramkörök felhasználásával a feldolgozási idő egyenértékű a felvétel időbeni hosszával. Tehát azonnali visszajelzést adhat a kutatók számára a felvétel közben, amely kritikus információt hordoz magában, hiszen nem utólag derül ki az elektródatömb behelyezésének hibája.

A 3. fejezetben létrehoztam egy FPGA alapú valós idejű tüske osztályozó rendszet, amely figyelembe veszi a térbeli információkat az egymáshoz közeli elektródákról és osztályozza a többcsatornás neurofiziológiás adatokat. Az eredeti Online Sorting algoritmust (felügyelet nélküli sablonillesztés alapú) módosítottam MATLAB-ban, majd implementáltam FPGA-ra, hogy több csatornát is (jelen esetben 128) képes legyen egyszerre feldolgozni, valamint az aktuális tüzeléshez tartozó 3x3-as ablakban lévő az egymáshoz közeli elektródákon való áthallási (térbeli) információt is felhasználja az osztályozás során. Az elkészített FPGA alapú architektúra 86%-os átlagos pontosságot ér el szintetikus adatokon (10-4 dB SNR), míg ezzel szemben az eredeti egycsatornás OSort (csatornánként külön-külön) átlagosan 74%-ot. Az elkészített FPGA alapú architektúra altatott patkányon felvett valós adatokon is tesztelve lett, amelynek eredménye a kiloSort (offline) osztályozó algoritmus utólag szakértő által manuálisan javított eredményéhez lett hasonlítva. Kereszt-korreláció alapján az elkészített FPGA alapú architektúra 80%-ban hasonló osztályokat hozott létre.

FPGA alapú rendszerek jelfeldolgozási és szimulációs témakörben való alkalmazása

Az FPGA alapú rendszerek széleskörben használhatóak valós idejű működésük, könnyű átprogramozhatóságuk és alacsony költségük miatt számos jelfeldolgozási és szimulációs feladatra. A gyors prototipizálásnak köszönhetően a különböző területeken használt algoritmusok FPGA segítségével felgyorsíthatóak és valós idejűvé tehetők, amennyiben az algoritmusok kritikus pontjai párhuzamosíthatóak. Az efféle prototipizálásra mutatok 4 példát, amely az FPGA-ak rugalmasságát és széleskörű felhasználását bizonyítja. Ilyen területek többek között az arcfelismerés, szenzorfúzió, szabályozástechnika vagy hardware-in-the-loop szimuláció, mely témakörökben egy-egy FPGA alapú áramkört fejlesztettem.

Arcfelismerés témakörben egy PCA alapú valós időben működni képes költséghatékony arcfelismerő rendszert hoztam létre. Szenzorfúzió témakörben egy valós időben pozíciót és orientációt becsülni képes szenzorfúziót alkalmazó rendszert fejlesztettem ki. Szabályzástechnika témakörben egy aktív felfüggesztéshez a felhasználni kívánt erő nagyságát szabályzó valós időben működő rendszert hoztam létre. HIL szimuláció témakörben szélturbina működését emuláló FPGA architektúrát hoztam létre, amely egy LabView alapú hardware-in-the-loop rendszer részét képezte.

A disszertáció tézisei

Az **első téziscsoportban** a hozzájárulásaim a neurofiziológiai jelek FPGA alapú valós idejű detektálásához és szintéziséhez kapcsolódnak. A részletes bemutatás a 2. fejezetben található.

I / 1. Megadtam egy Non-Linear Energy Operator (NEO) alapú többcsatornás akciós potenciál detektáló FPGA architektúrát, amely képes az akciós potenciál forrásának meghatározására is egy többcsatornás elektródán. Továbbá megmutattam, hogy a megvalósított rendszer valós időben működni képes.

I / 2. Javasoltam egy keresztkorreláció alapú akciós potentáciál detektáló módszert, megmutattam a hatékonyságát és rámutattam a hátrányaira.

I / 3. Kidolgoztam egy neurofiziológiai jelgenerátor algoritmust, amely képes véletlenszerűen kiválasztott csatornákra, véletlenszerűen meghatározott frekvenciával és tüzelési áthallással az akciós potenciál sablonok elhelyezésére, figyelembe véve a tudományban eddig ismert elméletet a neuronok tüzeléséről.

I / 4. Megadtam egy FPGA architektúrát, amely képes neurofiziológiai jelek generálására a Mersenne Twister pszeudo-véletlenszám generáló algoritmus használatával. Megmutattam, hogy az architektúra alkalmas valós idejű, 128 csatornás elektróda struktúrára neurofiziológiai jeleket generálni.

A **második téziscsoport** a neurofiziológiai jelek FPGA alapú valós idejű térbeli információk figyelembevételével való osztályozásához kapcsolódik. A részletes bemutatás a 3. fejezetben található.

II / 1. Javasoltam többcsatornás neurofiziológiás felvételeken a csatornák közötti térbeli információk felhasználását az Online Sorting algoritmus használatával. Szintetikus adatokon összehasonlítottam az eredeti egycsatornás és a javasolt algoritmus hatékonyságát. Megmutattam, hogy a javasolt, térbeli információkat is felhasználó algoritmus jobb hatékonyságot eredményez.

II / 2. Megadtam az Online Sorting algoritmuson alapuló többcsatornás neurofiziológiai felvételeket valós időben osztályozni képes FPGA architektúrát. Szintetikus és valós adatokon alapuló tesztekkel megmutattam a hatékonyságát. Továbbá megmutattam, hogy az elkészített FPGA architektúra képes valós időben működni és alkalmas a kísérletek során történő azonnali visszajelzésre.

A **harmadik téziscsoport** olyan újszerű FPGA alapú rendszereket tartalmaz, melyek egyes területeken hatékonyabbá teszik a jelfeldolgozási és gyors prototípusfejlesztési feladatokat. Részletes bemutatás a 4. fejezetben található.

III / 1. Javasoltam egy új párhuzamos költséghatékony architektúrát, amely a Principal Component Analysis (PCA) algoritmust FPGA-n képes valós időben elvégezni.

III / 2. Megadtam egy új párhuzamos kölcséghatékony FPGA architektúrát, amely több szenzort és azok méréseit felhasználva képes valós időben szenzorfúzióval pozíciót és orientációt becsülni. Javasoltam egy beltéri lokalizációs megoldást, amely lehetővé teszi az abszolút pozícióra vonatkozó információk beépítését a kidolgozott szenzorfúziós architektúrába. Továbbá megadtam a kétdimenziós multilaterációs algoritmushoz egy FPGA architektúrát.

III / 3. Kidolgoztam egy megbízható, költséghatékony módszert, amely minimális számú szenzor felhasználásával hatékonyan szabályozza egy gépjármű aktív felfüggesztő rendszerét. Megadtam egy új párhuzamos FPGA architektúrát, amely az aktuális megfigyelő alapján szabályozza az aktív felfüggesztés erő nagyságát. Továbbá megmutattam az FPGA architektúra valós idejű működőképességét.

III / 4. Megadtam egy új párhuzamos FPGA architektúrát, amely képes szélturbina valós idejű hardveres emulációjára egy hardware-in-the-loop (HIL) rendszerben. Megvalósítottam egy SCADA interfészt, amelyen keresztül lehet a szélturbina paramétereit valós időben állítani. Továbbá megmutattam a szélturbinát hardveresen emuláló FPGA alapú HIL rendszer valós idejű működőképességét.

Publications

Journal publications

[1] Á. Kaló, Z. Kincses, **L. Schäffer** and Sz. Pletl Indoor localization simulation framework for optimized sensor placement to increase the position estimation accuracy. *Annales Mathematicae et Informaticae*, 51, 29-39, 2020.

[2] **L. Schäffer**, Z. Nagy, Z. Kincses, R. Fiáth and I. Ulbert Spatial Information Based OSort for Real-Time Spike Sorting Using FPGA. *IEEE Transactions on Biomedical Engineering*, 68(1), 99-108, 2021.

Full papers in conference proceedings

[3] **L. Schäffer** and Z. Kincses. Implementation of an FPGA-based wind turbine HIL model. In *Proceedings of the 3rd Mechedu Conference*, Subotica Tech, 159-163, 2015.

[4] **L. Schäffer**, Z. Nagy, Z. Kincses and R. Fiáth. FPGA-based neural probe positioning to improve spike sorting with OSort algorithm. In *Proceedings of the 2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 1375-1378, 2017.

[5] **L. Schäffer**, Z. Kincses and Sz. Pletl. FPGA-based low-cost real-time face recognition. In *Proceedings of the 15th International Symposium on Intelligent Systems and Informatics (SISY)*, IEEE, 35-38, 2017.

[6] **L. Schäffer**, Z. Nagy, Z. Kincses and R. Fiáth. FPGA-based real-time multichannel neural dataset generation. In *Proceedings of the 2017 European Conference on Circuit Theory and Design (ECCTD)*, IEEE, 1-4, 2017.

[7] **L. Schäffer**, Sz. Pletl and Z. Kincses. Implementation of an FPGA-based actual observer for active suspension control. In *Proceedings of the 4th International*

Conference and Workshop on Mechatronics in Practice and Education, Subotica Tech, 28-32, 2017.

[8] **L. Schäffer**, Z. Kincses, Sz. Pletl. A Real-Time Pose Estimation Algorithm Based on FPGA and Sensor Fusion. In *Proceedings of the 16th International Symposium on Intelligent Systems and Informatics (SISY)*, IEEE, 149-154, 2018.

[9] **L. Schäffer**, Sz. Pletl and Z. Kincses. Spike Detection Using Cross-Correlation Based Method. In *Proceedings of the 23rd International Conference on Intelligent Engineering Systems (INES)*, IEEE, 175-178, 2019.

Further related publications

[10] **L. Schäffer**, Z. Nagy, Z. Kincses, Z. Vörösházi, R. Fiáth, I. Ulbert, P. Szolgay. FPGA-based clustering of multi-channel neural spike trains. In *The 15th International Workshop on Cellular Nanoscale Networks and their Applications*, VDE, 115-116, 2016.

Acknowledgments

First of all, I would like to thank my supervisors, Szilveszter Pletl and Zoltán Kincses for directing my PhD studies. I would also like to thank Zoltán Nagy for the support and advices, Richárd Fiáth and István Ulbert for their help in neurobiology and their spike sorting expertise. Also great thanks to my colleagues and friends who helped me to realize the results presented here and to enjoy the period of my studies. Last, but not least, I wish to thank my fiancee and family for their constant love and support.