Abstract

Vibrations in machines increase friction on moving parts which cause chafing that will tear down the fabric of the machine components when given time, thus monitoring and analysis of machine vibrations are important for preventive maintenance. Vibration analysis utilizes time domain as well as frequency domain analysis for which there have been analog solutions for quite some time. This work has been about moving a predominantly analog mixed signal system onto an FPGA and making it mostly digital. Vibration analysis on an FPGA have its own challenges and benefits compared to other methods. The inherent parallelism of the FPGA makes it suitable for high performance signal analysis. This report shows through two proof-of-concept solutions that the translation of a predominantly analog system is viable, economic and can deliver improved performance. The two solutions have utilized two different units from Xilinx, the Spartan-6 FPGA and the Zynq-7000 system on chip FPGA. The solution implemented on Spartan-6 produces a result in 9.32 ms and the other implementation based on Zynq-7000 produces a result in 9.39 ms, which is more than a 10-fold increase in performance of the current system. The results obtained show that both solutions can perform the calculations for the proof of concept within 20% of the allotted time. Costs of both solutions as well as other qualities of each solution are presented in this paper.
Table of Contents

1 Background .......................................................................................................................... 1
  1.1 Abbreviations .................................................................................................................. 2

2 Relevant theory ..................................................................................................................... 3
  2.1 Vibration Measurement and Analysis ............................................................................. 3
  2.2 Sampling and aliasing ..................................................................................................... 3
  2.3 ADC .................................................................................................................................. 4
  2.4 Fast Fourier Transform ................................................................................................. 4
  2.5 Filters ............................................................................................................................ 5
  2.6 FPGA ............................................................................................................................ 5
  2.7 SoC .................................................................................................................................. 5

3 Related Work ....................................................................................................................... 6

4 Problem Formulation .......................................................................................................... 6

5 Analysis of Problem ........................................................................................................... 7
  5.1 Measurement module ...................................................................................................... 7
    5.1.1 Overview ................................................................................................................... 7
    5.1.2 Filter Block ............................................................................................................... 7
    5.1.3 ADC Block ............................................................................................................... 8
    5.1.4 DSP Block ................................................................................................................. 9
    5.1.5 I/O configuration and control (IOCC) block ............................................................. 9
    5.1.6 EtherCAT block ........................................................................................................ 10
  5.2 System requirements (Measurement module) .............................................................. 10
  5.3 Component cost (Measurement module) ..................................................................... 10
  5.4 CPU-module ................................................................................................................... 11

6 Method .................................................................................................................................... 12

7 Research ................................................................................................................................ 13
  7.1 Filter ............................................................................................................................... 13
  7.2 ADC ................................................................................................................................ 13
  7.3 DSP ................................................................................................................................ 13
    7.3.1 FFT .......................................................................................................................... 13
    7.4 EtherCAT ...................................................................................................................... 13
    7.5 FPGA / Soc FPGA ........................................................................................................ 13

8 Research Results .................................................................................................................. 14
  8.1 Analog Filters .................................................................................................................. 14
  8.2 Digital Filters .................................................................................................................. 14
    8.2.1 FIR description .......................................................................................................... 14
    8.2.2 FIR Compilers .......................................................................................................... 15
  8.3 ADCs .............................................................................................................................. 16
    8.3.1 Conventional ADC ................................................................................................. 16
    8.3.2 FPGA based ADC .................................................................................................... 17
  8.4 DSP Replacements .......................................................................................................... 19
    8.4.1 ARM SoC ................................................................................................................ 19
    8.4.2 FFT IP ..................................................................................................................... 20
  8.5 EtherCAT ......................................................................................................................... 21
  8.6 Estimation of total FPGA resource usage ...................................................................... 23
  8.7 FPGA/SoC FPGA .......................................................................................................... 23
    8.7.1 Altera Cyclone III ..................................................................................................... 23
    8.7.2 Xilinx Spartan-6 .......................................................... 24
    8.7.3 Xilinx Zynq-7000 ................................................................................................. 24
    8.7.4 Microsemi SmartFusion ......................................................................................... 24
    8.7.5 Summary ................................................................................................................ 24
    8.7.6 FPGA prices ........................................................................................................... 25
    8.7.7 Peripherals .............................................................................................................. 26
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.5</td>
<td>Testing</td>
<td>50</td>
</tr>
<tr>
<td>13.6</td>
<td>Cost</td>
<td>50</td>
</tr>
<tr>
<td>14</td>
<td>Future work</td>
<td>52</td>
</tr>
<tr>
<td>15</td>
<td>Summary and conclusions</td>
<td>53</td>
</tr>
<tr>
<td>16</td>
<td>References</td>
<td>54</td>
</tr>
<tr>
<td>17</td>
<td>Appendix A</td>
<td>55</td>
</tr>
<tr>
<td>17.1</td>
<td>IIR description</td>
<td>55</td>
</tr>
<tr>
<td>17.2</td>
<td>FFT configurations</td>
<td>56</td>
</tr>
<tr>
<td>18</td>
<td>Appendix B</td>
<td>58</td>
</tr>
<tr>
<td>18.1</td>
<td>Code for Design 3</td>
<td>58</td>
</tr>
<tr>
<td>18.1.1</td>
<td>Analysis result</td>
<td>58</td>
</tr>
<tr>
<td>18.1.2</td>
<td>Time measurement</td>
<td>60</td>
</tr>
<tr>
<td>18.2</td>
<td>Code for Design 4</td>
<td>62</td>
</tr>
</tbody>
</table>

IV
1 Background

Monitoring and analysis of machine vibrations are important for preventive maintenance since the vibrations increase friction which cause chafing that will tear down the fabric of the machine components when given time. While humans have the ability to sense vibrations, they have difficulties to assess the vibrations in terms of frequency and amplitude. This require instruments with the ability to identify different frequencies, as a vibration consist of one or multiple frequencies. Identifying the individual frequencies of a machine is akin to looking at a fingerprint as an individual frequency is related to a certain machine part and all frequencies combined captures the entire machine. This makes it possible for identification of the individual problem parts by frequency analysis of the vibrations measured in the system, which in turn reduces unplanned downtime by allowing for planned maintenance and replacement of faulty parts before they break down and when the production is least affected.[1]

Digital signal processing is of great significance when monitoring and analyzing vibrations because numerous computation-heavy calculations need to be done on the signal [1]. Although signal processing systems usually are a mixture of both analog and digital components, some analog components are inherently required in order to manage analog signals. Processing signals in a digital system has several advantages over processing in an analog system ranging from signal purity to cost.

Field-Programmable Gate Arrays (FPGA) are becoming an affordable option in digital signal processing applications where the Digital Signal Processor (DSP) previously was the natural choice, even for low-volume applications. Utilization of the massive parallelism inherent in the FPGA makes it a possible replacement of the less parallel DSP, especially in high-performance signal processing applications [2]. The use of an FPGA could eliminate the need for specialized external hardware performing one specific task, by incorporating that functionality into the FPGA.

In this paper we will present a design optimization of an existing vibration monitoring instrument with the use of an FPGA. The first section will give a short introduction to vibration analysis and important theory belonging to it, such as sampling, FFT and filters.
## 1.1 Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC</td>
<td>Alternating Current</td>
</tr>
<tr>
<td>ADC</td>
<td>Analog to Digital Converter</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DC</td>
<td>Direct Current</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital Signal Processor</td>
</tr>
<tr>
<td>EtherCAT</td>
<td>Ethernet for Control Automation Technology</td>
</tr>
<tr>
<td>FFT</td>
<td>Fast Fourier Transform</td>
</tr>
<tr>
<td>FIR</td>
<td>Finite Impulse Response</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field-Programmable Gate Array</td>
</tr>
<tr>
<td>HDL</td>
<td>Hardware description language</td>
</tr>
<tr>
<td>HP</td>
<td>High-pass</td>
</tr>
<tr>
<td>IC</td>
<td>Integrated Circuit</td>
</tr>
<tr>
<td>IIR</td>
<td>Infinite Impulse Response</td>
</tr>
<tr>
<td>IP</td>
<td>Intellectual Property (Something protected by patent, copyright or trade mark etc)</td>
</tr>
<tr>
<td>IP core</td>
<td>A protected component that can be used in HDL designs.</td>
</tr>
<tr>
<td>LP</td>
<td>Low-pass</td>
</tr>
<tr>
<td>P-P</td>
<td>Peak-to-Peak</td>
</tr>
<tr>
<td>RMS</td>
<td>Root Mean Square</td>
</tr>
<tr>
<td>SoC</td>
<td>System on Chip</td>
</tr>
<tr>
<td>SPI</td>
<td>Serial Peripheral Interface</td>
</tr>
<tr>
<td>VHDL</td>
<td>VHSIC (Very High Speed Integrated Circuit) Hardware Description Language</td>
</tr>
<tr>
<td>VMI</td>
<td>Vibration Monitor Instrument</td>
</tr>
</tbody>
</table>
2 Relevant theory

2.1 Vibration Measurement and Analysis

The problem the existing system addresses is vibration analysis,[3] that is to identify and analyze frequencies caused by moving parts in a running machine. All moving parts give rise to vibrations of certain frequencies [1]. Every measured signal is a sum of all vibrations sensed at that point and must be decomposed into a frequency spectrum, by applying FFT, in order to identify individual vibration frequencies embedded in that signal. That is, transforming the signal from time domain to frequency domain, as shown in figure 2.1. Analyzing and monitoring the frequencies will give an indication of the current condition of the parts in the monitored machine, which allows for replacement of malfunctioning parts before greater or critical damage is inflicted to the machine. Calculations in the time domain are also important when determining the condition of a part in a running machine. Values calculated in the time domain are for example root mean square (RMS), peak and peak-to-peak (P-P).

![Figure 2.1: Illustration of two different sine waves combined in the time domain and their correspondent in the frequency domain.](image)

2.2 Sampling and aliasing

Sampling is of great importance for a successful vibration analysis. According to Nyquist sampling theorem [3], in order to accurately measure the frequency of a signal, it needs to be sampled with at least double the frequency.

Given a sample frequency $f_s$ and the frequency of the measured signal being higher than $f_s/2$, which is the Nyquist-frequency, aliasing of the sampled signal will occur, shown in figure 2.2. This means that the signal will be mirrored across $f_s/2$ and appear as a lower frequency in the interval 0 to $f_s/2$. In most applications aliasing is not desired so an anti-aliasing filter, a low-pass (LP) filter with cutoff frequency at or below the Nyquist frequency, can be used to prevent this behavior.
2.3 **ADC**

Analog-to-Digital converters are characterized by their resolution and frequency to identify their respective working capabilities. The resolution of the ADC shows how many individual voltage levels the ADC can differentiate between and is represented by the number of bits in the output. This gives that an 8-bit ADC will differentiate between $2^8 = 256$ different voltage levels and a 16-bit ADC will differentiate between $2^{16} = 65536$ voltage levels with each level being represented by a value. For an ideal ADC the output is linear, where each level in the output are of equal width, that is, uniform step width. Due to anomalies in the architecture of conventional ADCs there is a deviation from the ideal step width. This deviation is the Differential Non-Linearity (DNL), stated in the data sheet of the ADC. The DNL error accumulates over the range of output steps, increasing the deviation from from the ideal ADC output. The greatest accumulated deviation is referred to as Integral Non-Linearity (INL), which is the maximum deviation from the ideal ADC output. DNL and INL reduces the actual resolution of the ADC.\[^4\,\,^5\]

Most implementations of ADCs are traditional integrated circuits (IC) but in recent years ADCs have also been implemented on FPGAs. Implementations of ADC comes in many different kinds with their own benefits and limitations.

2.4 **Fast Fourier Transform**

FFT [6] is an algorithm for calculating the Discrete Fourier Transform (DFT) efficiently. The DFT is used to transform a signal in the time-domain into the frequency-domain, making the basic sinusoids the signal is composed of visible. Signals in the frequency-domain can be reversed to the time-domain, an operation called Inverse FFT (IFFT).

The DFT is defined by the following formula.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi k \frac{n}{N}}$$ \hspace{1cm} (1)

In the formula 1, $x_0 - x_{N-1}$ are the samples to perform the transformation on i.e. the input signal and $X_0 - X_{N-1}$ is the complex result in the frequency domain. $N$ is the transform size.

The FFT algorithm reduces the complexity of the DFT algorithm from $O(N^2)$ to $O(N\log(N))$. This is achieved by dividing one large DFT into two smaller DFTs. The same procedure is applied to both sub DFTs until the basic DFT is reached. There are different types of algorithms and the size of the basic DFT differs between them, for Radix-2 the size is 2.
2.5 Filters

In signal analysis it is desirable to remove unwanted frequencies and only keep the frequency spectrum of interest, that is, to filter the signal. There are various filters for different purposes. All filters will allow some frequencies to pass, the pass-band, and will suppress some frequencies, the stop-band, with the transition between these two being called transition-band, as seen in figure 2.3. A low-pass (LP) filter will attenuate signals with frequencies higher than the cutoff frequency and the lower frequencies will pass unchanged. The attenuation depends on the order of the filter, thus a high order filter will attenuate the higher frequencies more than a low order filter. Generally the signal is attenuated $20^n \text{dB} \text{ per decade or } 6^n \text{dB} \text{ per octave}$, i.e. how steep the slope in the transition-band is, where $n$ is the order of the filter. The high-pass (HP) filter is the opposite of the LP filter, it allows high frequencies to pass and attenuates frequencies lower than the cutoff frequency. Other types of filters are available but not relevant for this work.

![Figure 2.3: The different frequency response bands for a low-pass filter. Source: http://cnx.org/content/m16895/latest/figC0.png (last visit 2012-11-20)](http://cnx.org/content/m16895/latest/figC0.png)

2.6 FPGA

A Field-Programmable Gate Array (FPGA) typically contains fixed function logic blocks, consisting of multipliers and embedded block ram, and programmable logic blocks, typically consists of flip-flops and Look-Up-Tables (LUTs), together with a programmable interconnect connecting the logic blocks together. The programmable logic block is the basic unit of the FPGA, also referred to as slice, logic element or logic cell, among others. The FPGA is configured using a hardware description language (HDL). A design tool is used to convert the HDL into a bitstream for downloading to the FPGA.

2.7 SoC

System on Chip (SoC) is an electronic system embedded on a single chip. An SoC usually consists of a combination of microcontroller/microprocessor/DSP, memory block, peripherals, external interfaces, timing sources, analog interfaces and others. An FPGA solution made from several IP cores often with a soft CPU core, that controls the system, is an FPGA based SoC. There are solutions with a hard-wired SoC, which often includes a CPU, combined with an FPGA on a single chip, that are called SoC FPGA.
Related Work

Contreras-Medina et al., 2008,[7] used a low-cost FPGA when developing a special purpose vibration analyzer with multiple input channels. It was developed because several applications required simultaneous vibration monitoring of multiple-channels and most of the available equipment were not suited for that. An FPGA was used because of its parallel architecture, reconfigurability and the ability to become an SoC solution.

The solution consists of two parts; an instrumentation system and the FPGA. The instrumentation system consists of a three-axis accelerometer that senses vibration and its output, acceleration in X-, Y- and Z-direction, are passed to a 4-channel 12-bit ADC, although only three channels are used. The instrumentation system then transmits data on 3-channels in parallel to the FPGA. When the data is received three 1024-point FFT computations are done simultaneously, with a total calculation time of 1.33 ms.

A vibration measurement and analysis instrument was developed by da Costa et al., 2010,[8] that implemented the digital signal processing algorithms e.g. FIR filter, FFT etc onto a low-cost FPGA with the use of a MATLAB/Simulink model. The DSP Builder from Altera was used to automatically create HDL from the MATLAB/Simulink model. The system was developed with the purpose to diagnose the condition of an induction motor so that no trained expert would be required for that task.

The system is built up of five functional blocks; Data acquisition and filter, time domain analysis, vibration severity measurement, critical alarms and frequency domain analysis.

The data acquisition and filter block samples one analog input with a sample rate of 25 kHz, then performs linear scaling of the signal and filters it through a digital 6th order low-pass Butterworth filter with a cut-off frequency at 12 kHz. The signal can then be filtered in a high-pass filter by using one of four predetermined configurations, after which the signal is sent to the time domain analysis block and the frequency domain analysis block. In the time domain analysis block, RMS value, peak value, crest factor and kurtosis are calculated. The Vibration severity measurement block uses the overall RMS level to extract the vibration severity specified by ISO standard 10816-1 and has four outputs of machine status: good, satisfactory, unsatisfactory and unacceptable. In the critical alarm block three outputs of alarm status are displayed and show if the peak value, crest factor or kurtosis is larger than specified alarm levels. In the frequency domain analysis block the signal can be filtered again, but this time in a band-pass filter, using one out of four predetermined configurations before the FFT is calculated and displayed.

Problem Formulation

A vibration monitor instrument has been developed for the maintenance market to ensure a higher availability for production equipment. The instrument monitors vibrations and analyzes their frequency patterns by applying numerous algorithms. Due to the high production cost and the need for increased performance of the current design, a new version is planned. The aim of this work is to find a method to translate the existing processing system composed of both analog and digital components into a mostly digital system using an FPGA and show the viability of that method. The main goal of the new version is to increase performance and decrease the production cost.

The solution needs only be a proof of concept on an FPGA of one functional vibration input channel. In this work focus should be on the measurement module.
5 Analysis of Problem

The new system has to fulfill the requirements of the existing one and in addition be more responsive and cost less. First the structure of the existing system is given and then some characteristics are presented. The vibration monitor instrument consists of a CPU-module and up to 10 measurement modules.

5.1 Measurement module

5.1.1 Overview

The existing measurement module is made up of two measurement boards, one backplane, one DSP-board and has eight input channels for sensors of vibration or temperature type. The different boards are connected through the backplane. In the system the signal passes through a setup of analog filters, one setup per sensor channel. The signal then passes through an Analog to Digital Converter (ADC) before entering a DSP for FFT and other computations. The result is then sent to the CPU module over EtherCAT for further computation and distribution. Figure 5.1 shows a simplified overview of the system abstracted to one vibration input channel. The separate functional blocks of the system will be explained more thoroughly in the following sections.

![Figure 5.1: Abstract overview of the measurement module.](image)

5.1.2 Filter Block

The filter block, in figure 5.2, filters the input signal by removing unwanted frequencies above 12.8 kHz. Before the analog input signal enters the filter block a voltage divider scales the input signal to a suitable level. The analog signal is then divided where one part is mapped to a 1st order HP-filter, to remove the DC part of the signal, and the other to a differential amplifier. The HP-filter outputs a true AC-signal which is again split, with one part being passed to a differential amplifier and the other part to an instrumentation amplifier. The DC-part of the input signal is obtained by differentiating the true AC-signal and the analog input signal, which is used to detect anomalies in the sensor attached to the module. The instrumentation amplifier scales the AC-signal to an appropriate voltage level for the ADC’s input channel. Between the signal leaving the instrumentation amplifier and reaching the ADC it passes through a series of three 2nd order Butterworth LP-filters of Sallen-Key topology making them together act as one 6th order filter. This filter also works as an anti-aliasing filter for the signal.
5.1.3 ADC Block

The ADC has 8-channels with 16-bit resolution where each channel has a designated sample and hold circuit. In addition each channel contains a 2nd order Butterworth LP-filter with a cut-off frequency at 15 kHz that prevents aliasing since they are sampled at 32 kHz. The ADC block, shown in figure 5.3, is composed of the ADC and two analog switches. In the figure below it is shown how the ADC block is connected to the input of the system. Four of the ADC channels are connected to the AC-parts of the filter blocks while the other four are either connected to the DC-parts of the filter blocks or temperature-signals. Each temperature signal is filtered with a 2nd order LP Butterworth filter with a cut-off frequency at 30 Hz in order to remove noise from the supply voltage. The choice between the DC-signals and the temperature signals are controlled with the analog switches.

Figure 5.2: The Filter Block that prepares the input signal for the ADC and separates the DC component from the AC component in the input signal.

Figure 5.3: Abstract overview of the measurement board and its different blocks, specifically the ADC block. The ADC has eight input channels of which four of them are AC signals and the other are DC or temperature, which are controlled by analog switches.
5.1.4 DSP Block

The DSP block, in figure 5.4, making up the DSP-board, consists of a DSP, EEPROM, SDRAM with input from two measurement boards through the backplane, an inter-board communications board. A measurement board consists of the components shown in figure 5.3 above, with one addition, the Sense/CS signals. Analysis of the samples from the ADC is performed by the DSP. The analysis at this stage is FFT, peak-to-peak and root mean square etc. The FFT calculation is done on a sample window of 1 second, thus a transform size of 32 768 samples. For the existing system the calculations have taken a couple of seconds.

The Sense/CS is a bus that is switched to be either input, sense signals (Sense), or output, control signals (CS). The Sense signals are I/O configurations which are configured with jumpers. The control signals consists of chip-select signals and a control signal for the DC/Temp switch. Four additional signals are taken as input by the DSP, two RPM signals and two steering signals, used for calculations. Results from computations done by the DSP are transmitted through EtherCAT.

![Figure 5.4: Shows the I/O connections of the DSP.](image)

5.1.5 I/O configuration and control (IOCC) block

Routing of the I/O configuration signals to the DSP and control signals from the DSP is done over a switched bus system controlled by latches. This allows the DSP to receive input or transmit output depending on the state of the latches, controlled by the DSP. One IOCC block is present in each measurement board. Figure 5.5 below shows how the routing is done on an abstract level where signals are grouped into buses. The I/O configuration signals are labeled Sense and the control- and chip select signals are labeled Control.

---

1 The name of the signal is Sense and is part of the original systems naming scheme.
5.1.6 EtherCAT block

EtherCAT is a high performance Ethernet based bus-system and is in this system used to send data between the measurement module and CPU-module. The EtherCAT component used in the measurement module is ET1200 Slave controller.

5.2 System requirements (Measurement module)

An overview of the system requirements are listed, in table 1, for ease of access.

Table 1: The system requirements.

<table>
<thead>
<tr>
<th>Functional Block</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter</td>
<td></td>
</tr>
<tr>
<td>Cut-off frequency</td>
<td>12.8 kHz</td>
</tr>
<tr>
<td>ADC for vibration measurement</td>
<td></td>
</tr>
<tr>
<td>Sampling frequency</td>
<td>32 kHz</td>
</tr>
<tr>
<td>Resolution</td>
<td>16-bit</td>
</tr>
<tr>
<td>Simultaneous inputs</td>
<td>8 channels</td>
</tr>
<tr>
<td>ADC for DC measurement</td>
<td></td>
</tr>
<tr>
<td>Sampling frequency</td>
<td>&gt; 1 Hz</td>
</tr>
<tr>
<td>Resolution</td>
<td>12-bit</td>
</tr>
<tr>
<td>ADC for temperature measurement</td>
<td></td>
</tr>
<tr>
<td>Sampling frequency</td>
<td>10 Hz</td>
</tr>
<tr>
<td>Resolution</td>
<td>12-bit</td>
</tr>
<tr>
<td>FFT</td>
<td></td>
</tr>
<tr>
<td>Sample size</td>
<td>32 768 samples</td>
</tr>
<tr>
<td>Transform time</td>
<td>&lt;50 ms</td>
</tr>
</tbody>
</table>

5.3 Component cost (Measurement module)

The measurement module consists of a DSP-board, two Measurement-boards and a Backplane. The amount and cost of components on each board need to be known before changing the system, because the new design decisions depends on it. The cost for the possible new design
need to be compared with the existing design.

The cost and the amount of components for the different boards are summarized in tables 2, 3 and 4. Number of unique components are listed since many different types of components will have a negative effect on the price due to mounting costs. For the tables below prices were updated 2012-04-23 unless otherwise specified.

**Table 2: Amount and cost for the components on the DSP-board.**

<table>
<thead>
<tr>
<th>Components</th>
<th>Cost (SEK)</th>
<th>Amount</th>
<th>Unique</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICs</td>
<td>443.04</td>
<td>33</td>
<td>22</td>
</tr>
<tr>
<td>of which DSP</td>
<td>102.28(^1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capacitors</td>
<td>84.653</td>
<td>66</td>
<td>15</td>
</tr>
<tr>
<td>Resistors</td>
<td>22.113</td>
<td>63</td>
<td>18</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>549.805</strong></td>
<td><strong>162</strong></td>
<td><strong>55</strong></td>
</tr>
</tbody>
</table>

\(^1\)Price updated 2012-05-25

**Table 3: Amount and cost for the components on the Measurement-board**

<table>
<thead>
<tr>
<th>Components</th>
<th>Cost (SEK)</th>
<th>Amount</th>
<th>Unique</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICs</td>
<td>912.24</td>
<td>64</td>
<td>15</td>
</tr>
<tr>
<td>of which ADC</td>
<td>433.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capacitors</td>
<td>24.868</td>
<td>100</td>
<td>13</td>
</tr>
<tr>
<td>Resistors</td>
<td>43.524</td>
<td>124</td>
<td>20</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>980.632</strong></td>
<td><strong>288</strong></td>
<td><strong>48</strong></td>
</tr>
</tbody>
</table>

**Table 4: Amount and cost for the components on the Backplane**

<table>
<thead>
<tr>
<th>Components</th>
<th>Cost (SEK)</th>
<th>Amount</th>
<th>Unique</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICs</td>
<td>216.74</td>
<td>17</td>
<td>8</td>
</tr>
<tr>
<td>Capacitors</td>
<td>16.846</td>
<td>17</td>
<td>3</td>
</tr>
<tr>
<td>Resistors</td>
<td>3.861</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>237.447</strong></td>
<td><strong>45</strong></td>
<td><strong>17</strong></td>
</tr>
</tbody>
</table>

The total component cost, for all the boards (DSP-board + 2*Measurement-board + Backplane), is 2748.516 SEK. The total production cost is however unknown for the existing system, but by reducing the amount of components in the new design, especially unique ones, the production cost will be reduced.

### 5.4 CPU-module

This module sends requests via EtherCAT to the measurement modules about what to measure e.g. to perform an FFT- or RMS-calculation on the sensory input to measurement module 1. The processed data is then sent from the measurement module to the CPU over EtherCAT. This module forwards the analyzed data to the PC where it is presented in a program called SpectraLive or in a web interface. If the data is viewed in the web interface communication is done over the Ethernet protocol, otherwise RS-232 is utilized.

For some of the more computation-heavy calculations such as envelope and vector calculations the above flow differs in that the actual calculation is performed by the CPU-module itself. In “turbo mode” the FFT calculation will be done in the CPU-module instead of the DSP which reduces the transformation time to 50 ms.
6 Method

There are many existing project methodologies available, with their pros and cons, but the most obvious distinction between them are agile and non-agile methods (waterfall). Since the burden of documentation and the rigid work flow enforced by the waterfall methods could slow down progress, the chosen methodology for this project was according to agile models, with meetings more in the form of discussions during the day. However, to not get lost in the freedom given by the agile method the waterfall model served as an underlying structure of the big picture, giving information of what needed to be done, although the order of the tasks were not strictly followed. The method allowed work to progress on many tasks simultaneously and also jump back and forth between task.

The work consisted of these main parts:

- Study circuit diagrams of the existing system and identify their functional blocks.
- Search for replacements to the identified blocks
- Design of the new system
- Find the best suited FPGA/SoC FPGA
- Implementation and testing
- Write the report

During the study and system analysis the method of choice has been the break-down approach to identify groups of components making up functional blocks of which some can be exchanged for either digital solutions or altered in other ways to achieve the goals of this work. Each and every block performs some important function within the system.

Possible replacements for identified blocks or groups of blocks of components must maintain the same functionality as the originals, but preferably at a lower cost. A digital replacement could be a component written in HDL or an IP Core implemented on an FPGA but even physical components are possible although a secondary option. Analog replacements are only of interest when they can replace a larger set of components for a smaller set or when the same functionality can be obtained at a lower cost. Due to the above reasons most replacements will be aimed at becoming components in the FPGA.

New system designs must maintain the functionality of the existing system as a whole; meaning the system is treated like a black box, where for a given input the output must be consistent with the existing system. The internal design of the system can be varied.

Searching for the best suited FPGA or SoC FPGA requires estimations of how much resources the components in the design will demand, which is obtained through reading datasheets and by implementing test versions of components. Comparing prices between vendors are also necessary.

Designs found suitable for the system will be in part implemented on an FPGA in order to achieve a proof of concept. Testing has been done on every individual component in the system before they were integrated into the final system. This ensures that the parts of the system are correct in the case of the tested scenarios which increases the possibility that they function correctly after integration into the system.

Documentation of important information and writing on the report has been done continuously during the projects lifetime.
7 Research

Potential replacements for the functional blocks identified during the “Analysis of problem” were researched as a foundation for the design phase. The replacements researched are presented in this section. Each of the replacements has been evaluated in terms of performance, resource usage and cost where possible. Although different implementation options were researched, FPGA based implementations were focused upon, but also the possibility of DSP implementations were considered.

7.1 Filter

Research has been done to better understand analog filters and to see if any analog replacements exists within the performance and cost frames. Digital filters such as FIR and IIR have been researched to obtain knowledge of existing solutions and an understanding of the workings of and how to implement these.

7.2 ADC

Possible replacement alternatives to the ADC in the existing system has to support the desired sampling frequency, resolution and have enough input channels. Research after alternatives have focused on physical hardware components, but also covered the possibility of having ADC IP-cores on an FPGA.

7.3 DSP

Possible options to the current DSP have been researched with focus on other DSP ICs, ARM SoC or having functionality of the DSP done as IP-components on an FPGA. The functionality looked into in more detail has been FFT. In the search for suitable ARM SoCs there are a few requirements that has to be satisfied: enough RAM memory to store data sampled at 32 768 Hz for 1 second, enough on-chip ADCs, sufficient number of I/O pins and low cost.

7.3.1 FFT

The theory of FFT has been studied in an attempt to understand how the transform is calculated and to determine if an implementation from scratch is a viable option. The research also included implementations of FFT on FPGAs as IP Cores.

7.4 EtherCAT

An EtherCAT-controller IP core has been searched for in order to determine the possibility of replacing the current external component with an FPGA implementation.

7.5 FPGA / Soc FPGA

The use of an FPGA or SoC FPGA in order to replace the functionality of physical components, both analog and digital, is a requirement for this project. Due to this a study of available and planned FPGAs and SoC FPGAs have been conducted in order to evaluate possible candidates for this work. Important aspects to evaluate are the amount of resources, such as I/O pins, programmable logic blocks etc., and the cost.
8 Research Results

The results from the research are presented in this section. Research and development of designs were worked on concurrently; a workflow supported by the agile project methodology. There are parts of the research that could not be done until a certain level of knowledge had been achieved. Knowing the number of required input channels are needed to select e.g. certain ADC ICs or FPGA-based ADC implementations.

8.1 Analog Filters

Analog filters come in several types where the first distinction is the division in passive versus active filters. All filters are categorized according to how steep their transition-band is, which is referred to as the order of the filter. The simplest 1st order passive filters are made from any combination of a resistor (R), a capacitor (C) or an inductor (L). Filters can be type categorized as low-pass, high-pass, band-pass or band-stop.

Active filters can be made of any RLC combination together with an active component such as an amplifier. In the existing system the analog filters are active filters, where the LP-filters are 2nd order of Sallen-Key topology and the HP-filter is a 1st order active filter. Analog LP- and HP-filters can be combined in series of the same type, to form higher order filters, or together to form band-pass or band-stop filters. These analog filters give continuous time-domain filtering of signals which is of great value in signal analysis.

8.2 Digital Filters

Digital filters have many advantages over analog filters such as the ability to be reconfigured during runtime and to be of higher order, which allows for a steeper transition between the passband and stopband frequency (roll-off). No external components are required for a digital filter. The properties of a digital filter are determined by values stored in the digital system and will therefore stay unchanged over time, as compared to analog filters were the resistor, inductor and capacitor values can change.

The characteristics of a filter is the filter’s response given an impulse as input. In digital electronics this is valuable since a sampled input signal can be seen as a sequence of consecutive impulses. The output of a filter may be calculated by convolving (briefly described in the FIR description) the input signal with the filter’s impulse response. The response may be of finite length or very long (infinite), which connects to the terminology used for digital filters, Finite Impulse Response (FIR) filter and Infinite Impulse Response (IIR). IIR filters are derived from analog filters and do not give a linear phase response and can also be unstable due to the feedback-loop. FIR on the other hand gives a linear phase but do not originate from analog filters [9]. For digital solutions IIR-filters can be harder to implement [10].

Due to the non-linear phase of the IIR filter they are not of interest for this work since analysis of the phase is part of the system. Therefore only the FIR filter has been studied further; a short description of IIR can be found in Appendix A.

8.2.1 FIR description

A FIR filter [6, 9] is built up of multiply-accumulate (MAC) units and delay elements. The input sample data is multiplied with a coefficient and added together with delayed input samples multiplied by other coefficients. The multipliers that taps the signal from the delay line are called taps. The length of the delay line, number of delay units, determines the order of the filter. A delay line of length N yields an \( N^{th} \) order filter and N+1 taps. Figure 8.1 shows an example of a FIR filter structure where \( x \) is input, \( Z^{-1} \) is a delay unit, \( f \) is a coefficient and \( y \) is the totaled output.
The output of a FIR filter is calculated with equation 2 where \( f \) are the coefficients, \( x \) is the input samples and \( L \) is the number of filter coefficients.

\[
y[n] = x[n] \ast f[n] = \sum_{k=0}^{L-1} f[n] \ast x[n-k] \quad n=0,1,...
\]  

The output \( y \), in equation 2, is said to be obtained by convolving the two functions \( x \) and \( f \). Convolving is the act of doing a convolution which is calculating the area overlap in time between two functions.

Coefficients can be calculated using the filter functions in MATLAB or GNU Octave.

### 8.2.2 FIR Compilers


Altera also provides a tool for generating FIR filters, the FIR Compiler II MegaCore Function. The full production license is included in an active Quartus II Subscription Edition software which cost 3 995 USD for one year.

The FIR IP cores provided by Xilinx and Altera are optimized for their own FPGA devices. They also provide graphical user interfaces to simplify creation and configuration of the filter’s parameters. Table 5 shows the features of the two different FIR IP cores.

#### Table 5: Features of Altera’s FIR Compiler II and Xilinx’s FIR Compiler.

<table>
<thead>
<tr>
<th>Features</th>
<th>Altera FIR Compiler II</th>
<th>Xilinx FIR Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus interface</td>
<td>Avalon Streaming</td>
<td>AXI4-Stream</td>
</tr>
<tr>
<td>Filter Type</td>
<td>Single rate, Decimation, Interpolation, Fractional rate</td>
<td>Single rate, Decimation, Interpolation, Hilbert, Interpolated</td>
</tr>
<tr>
<td>Channels</td>
<td>1 – 128</td>
<td>1 – 64</td>
</tr>
<tr>
<td>Run-time Coefficient Reloading</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Coefficients per set</td>
<td>N/A</td>
<td>2 – 2048</td>
</tr>
<tr>
<td>Coefficient Sets</td>
<td>1 – Infinite</td>
<td>1 – 256</td>
</tr>
</tbody>
</table>

Resource estimations for Altera FIR Compiler II and Xilinx FIR Compiler filter cores are presented in tables 6 and 7. The estimations have been acquired in each design tool by synthesizing the cores. The configuration of the filters are single rate, one channel and 21 coefficients.
Table 6: Resource estimations for Xilinx FIR Compiler on Zynq-7020.

<table>
<thead>
<tr>
<th>Logic</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flip-flops</td>
<td>152</td>
</tr>
<tr>
<td>LUTs</td>
<td>128</td>
</tr>
<tr>
<td>BRAM</td>
<td>0</td>
</tr>
<tr>
<td>DSP slices</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 7: Resource estimations for Altera FIR Compiler II on Cyclone III EP3C55F484C8.

<table>
<thead>
<tr>
<th>Logic</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic elements</td>
<td>261</td>
</tr>
<tr>
<td>Flip-flops</td>
<td>206</td>
</tr>
<tr>
<td>LUTs</td>
<td>164</td>
</tr>
<tr>
<td>Memory bits</td>
<td>512</td>
</tr>
<tr>
<td>Embedded multiplier 9-bit</td>
<td>2</td>
</tr>
</tbody>
</table>

WinFilter is a free software tool used to design digital filters. It can generate C code for both FIR and IIR filters and VHDL code for FIR filters. The VHDL code can be optimized in regards to either size or speed and the software tool will show an estimation of FPGA resource usage. The filters supported are low-pass, high-pass, band-pass or band-stop. The filter models to choose from are Butterworth, Chebyshev, Bessel, Raised Cosine and Rectangular.

8.3 ADCs

8.3.1 Conventional ADC

ADC units were investigated after new designs were made. The new designs call for the use of one 8-channel 16-bit ADC. During the research of ADC ICs two possible alternatives have been found, both from Maxim. Both have 8-channel track and hold (T/H) with a dedicated ADC for each channel followed by 8 registers for holding the conversion values and with parallel output of the result. The ADCs found were the Maxim MAX11046ECB+ and the MAX11049ETN+, shown in table 8.

Table 8: Lists the properties of MAX11046ECB+ and MAX11049ETN+.

<table>
<thead>
<tr>
<th></th>
<th>MAX11046ECB+</th>
<th>MAX11049ETN+</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td>16-bit</td>
<td>16-bit</td>
</tr>
<tr>
<td>Input voltage</td>
<td>-5V to +5V</td>
<td>0 to +5V</td>
</tr>
<tr>
<td>Input bandwidth (MHz)</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Channels</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>On-Chip T/H Circuit for Each Channel</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Output interface</td>
<td>16-bit parallel</td>
<td>16-bit parallel</td>
</tr>
<tr>
<td>INL (LSB)</td>
<td>&gt; -2</td>
<td>± 0.4</td>
</tr>
<tr>
<td>DNL (LSB)</td>
<td>&gt; -1</td>
<td>± 0.4</td>
</tr>
<tr>
<td>Signal-to-noise ratio (SNR) dB</td>
<td>91</td>
<td>92.3</td>
</tr>
<tr>
<td>Total Harmonic Distortion (THD)</td>
<td>-105</td>
<td>-98</td>
</tr>
<tr>
<td>Throughput rate per channel (ksps)</td>
<td>1</td>
<td>250</td>
</tr>
<tr>
<td>Price¹</td>
<td>130.29 SEK²</td>
<td>117.99 SEK²</td>
</tr>
</tbody>
</table>

¹ Prices updated 2012-10-31
8.3.2 FPGA based ADC

By implementing the ADC on an FPGA some space can be freed up on the circuit board. Research on ADCs implemented on FPGA has been done by several researchers [11, 12, 13]. The common way of implementing ADCs on FPGA is to have an output from the FPGA connected to an RC circuit. The feedback voltage obtained from the RC circuit are then compared with a sampled voltage either using an external analog comparator or a LVDS input buffer on the FPGA. The ADC logic implemented on FPGA differs in all three papers and the solution by Uchagaonkar et. al., 2012, [12] will be described more in detail below.

Figure 8.2 shows the ADC structure which is based on sigma-delta modulation. The components implemented on the FPGA are D Flip Flop, CIC Filter (cascaded integrator-comb), and a Digital Filter. CIC is a special type of FIR filter combined with either an interpolator or decimator; for this design a decimator. The external components used are a comparator, resistor and capacitor. The resistor and capacitor together creates an RC circuit which voltage value is compared to the sampled value using the comparator. If the sampled voltage is larger than the RC voltage the comparator will output value ‘1’, else it will output ‘0’. The flip flop receives the value from the comparator and generates a feedback value to the RC circuit. It also sends the value to the CIC filter. The CIC filter is used for decimation to reduce the sample rate and averages a number of samples. A digital filter is then used to eliminate high frequency noise.

![Diagram of ADC implemented on FPGA](image1)

Figure 8.2: An ADC implemented on an FPGA. [12]

Stellamar offers a configurable ADC IP core, Digital ADC, that can be implemented on an FPGA. The only external components needed are resistors and capacitors for a simple reconstruction filter. The architecture of the ADC is shown in figure 8.3 below. A reconstruction filter limits frequencies that can be reconstructed and has a similar task as the anti-aliasing filter; Anti-aliasing filters are used before converting an analog signal into a digital and a reconstruction filter is used to produce a smooth analog signal from a digital one. The Digital ADC supports 10-bits, 12-bits and 14-bits of resolution. For 10-bits of resolution the supported bandwidth is up to 100 kHz, 12-bits supports up to 20 kHz and 14-bits up to 10 kHz. INL and DNL issues are not a problem as this is compensated for with oversampling and removal of unneeded bits. This IP core has a license fee in addition to a royalty for each unit.

![Diagram of Digital ADC from Stellamar](image2)

Figure 8.3: The architecture of the Digital ADC from Stellamar. Source: [http://www.stellamar.com/products.shtml](http://www.stellamar.com/products.shtml) (last visit 2012-11-20)
Xilinx also provides an ADC IP core, bundled with Xilinx EDK, that can be implemented on an FPGA, the XPS Delta-Sigma ADC. This ADC IP core requires a pair of resistors, a capacitor and a comparator as analog external components, as shown in figure 8.4. The supported resolutions are 10-bit and 12-bit. The supported sample rate for 10-bit resolution is up to 4340 Hz and for 12-bit resolution up to 887 Hz. The IP core connects as a 32-bit slave on a PLB v4.6 bus. A full production license is included with Xilinx ISE Design Suite software tools at no additional charge.

![Figure 8.4: Xilinx XPS ADC FPGA based design. Source: http://www.xilinx.com/support/documentation/ip_documentation/xps_deltasigma_adc.pdf (last visit 2012-11-20)](image)

A Simple Sigma-Delta ADC reference design, shown in figure 8.5, that can be implemented on an FPGA is provided by Lattice Semiconductor. The external components required are resistors and capacitors for a RC circuit. An external analog comparator may be required if the FPGA does not support LVDS input. The ADC supports up to 10-bits of resolution with a bandwidth up to 3.8 kHz.

![Figure 8.5: The architecture of Simple Sigma-Delta ADC. Source: http://www.latticesemi.com/documents/rd1066.pdf (last visit 2012-11-20)](image)

A summary of the different digital ADC IP cores are shown in Table 9.

Table 9: Summary of the digital ADC IP cores.

<table>
<thead>
<tr>
<th>Provider</th>
<th>Resolution</th>
<th>Bandwidth</th>
<th>FPGA Resource Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stellamar</td>
<td>10-bit</td>
<td>DC – 100 kHz</td>
<td>1195 LUTs, 9 DSP48A1s (Spartan-6 LX75)</td>
</tr>
<tr>
<td></td>
<td>12-bit</td>
<td>DC – 20 kHz</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>14-bit</td>
<td>DC – 10 kHz</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>10-bit</td>
<td>DC – 1.7 kHz</td>
<td>111 Slices, 190 LUTs (Spartan-6 LX45)</td>
</tr>
<tr>
<td></td>
<td>12-bit</td>
<td>DC – 346 Hz</td>
<td>96 Slices, 204 LUTs (Spartan-6 LX45)</td>
</tr>
<tr>
<td>Lattice Semiconductor</td>
<td>8-bit</td>
<td>DC – 3.8 kHz</td>
<td>62 LUTs (MachXO2)</td>
</tr>
<tr>
<td></td>
<td>10-bit</td>
<td>DC – 3.8 kHz</td>
<td></td>
</tr>
</tbody>
</table>
8.4 DSP Replacements

Replacements for the DSP found during research will be presented in this section. This comprises details about various ARM SoCs, tables 10 and 11, and the configurability of FFT IP cores from Altera and Xilinx. Note that the replacements only consider the DSP functionality of the existing system, therefore the FIR filter is not included.

8.4.1 ARM SoC

Table 10: ARM SoC units replacing the DSP. One of them does not have a floating point unit, but the other three does; denoted by the ‘F’ in the name of the core, Cortex M4F. All of them have DSP extensions.

<table>
<thead>
<tr>
<th>ARM SoC</th>
<th>Flash (KB)</th>
<th>SRAM (KB)</th>
<th>Freq. (MHz)</th>
<th>PIOs</th>
<th>ADC</th>
<th>Price/ unit¹</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel SAM4S16C</td>
<td>1024</td>
<td>128</td>
<td>120</td>
<td>79</td>
<td>12-bit</td>
<td>47.30 SEK</td>
<td>100</td>
</tr>
<tr>
<td>ATSAM4S16CA-AU ARM Cortex M4 SoC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Freescale Kinetis</td>
<td>512</td>
<td>128</td>
<td>120</td>
<td>104</td>
<td>12-bit</td>
<td>78.35 SEK</td>
<td>1000</td>
</tr>
<tr>
<td>PK10FX512VLQ12² ARM Cortex M4F SoC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Infineon XMC4500</td>
<td>1024</td>
<td>160</td>
<td>120</td>
<td>144³</td>
<td>12-bit x 4</td>
<td>73.62 SEK</td>
<td>1000</td>
</tr>
<tr>
<td>XMC4500E144F1024 ARM Cortex M4F SoC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STMicroelectronics</td>
<td>1024</td>
<td>192+4</td>
<td>168</td>
<td>140</td>
<td>12-bit x 3</td>
<td>65.27 SEK</td>
<td>250</td>
</tr>
<tr>
<td>STM32F407VGT6 ARM Cortex M4F SoC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

¹ Prices updated 2012-11-01
² One IC in the K10P144M120SF3-family of ICs
³ Total number of pins since the Programmable I/O pins are not available.
Table 11: Available on-chip peripheral control interfaces for the ARM SoC units.

<table>
<thead>
<tr>
<th>ARM SoC</th>
<th>I2C</th>
<th>SPI</th>
<th>Ethernet</th>
<th>UART</th>
<th>USB</th>
<th>CAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel SAM4S16C ARM Cortex M4F SoC</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Freescale Kinetis PK10FX512VLQ12, ARM Cortex M4F SoC</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Infineon XMC4500 XMC4500E144F1024 ARM Cortex M4F SoC</td>
<td>6¹</td>
<td>6¹</td>
<td>1</td>
<td>6¹</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>STMicroelectronics STM32407407VG ARM Cortex M4F SoC</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

¹ There are six universal serial interface channels usable as UART, double-SPI, quad-SPI, I2C etc.
² Also has two I²S interfaces

8.4.2 FFT IP

The implementation of the DFT algorithm can be done in a DSP or CPU, but also on an FPGA. During the study of the FFT algorithm different approaches to the computation of the FFT, along with optimizations to speed up the process were found [14, 15, 16]. Implementing the algorithm from scratch turned out to be beyond the scope of this project. Altera and Xilinx both provide core generators for creating FFT IP cores, shown in table 12. These tools allows for customization of the core regarding transform size, data format, precision of the data and architecture etc. Production licenses are included with a license for Xilinx ISE Design Suite software tools and with an active Quartus II Subscription Edition software respectively, at no additional charge.

Table 12: Configuration possibilities for Altera’s FFT and Xilinx’s FFT.

<table>
<thead>
<tr>
<th>Features</th>
<th>Altera FFT</th>
<th>Xilinx FFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus interface</td>
<td>Avalon Streaming</td>
<td>AXI4-Stream</td>
</tr>
<tr>
<td>Transform size</td>
<td>64 – 65 536</td>
<td>8 – 65 536</td>
</tr>
<tr>
<td>Channels</td>
<td>1</td>
<td>1 – 12</td>
</tr>
<tr>
<td>Run-time configurable transform length</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Input data width</td>
<td>8 – 32</td>
<td>8 – 34</td>
</tr>
<tr>
<td>Output order</td>
<td>Natural order, Bit reverse order</td>
<td>Digit reversed order, Bit reversed order, Natural order</td>
</tr>
<tr>
<td>Rounding output</td>
<td>Truncation, Convergent rounding ¹</td>
<td>Truncation, Convergent rounding</td>
</tr>
<tr>
<td>Architectures</td>
<td>Streaming, Variable Streaming, Buffered Burst and Burst</td>
<td>Pipelined Streaming, Radix-4 Burst, Radix-2 Burst and Radix-2 Lite</td>
</tr>
</tbody>
</table>

¹ The architecture determines which rounding method will be used; Convergent rounding is used for variable streaming and truncation is used otherwise.

Resource estimations for Xilinx FFT IP core are presented in table 13 and for Altera FFT IP core in table 14. The estimations have been acquired in each design tool by synthesizing the cores. The configurations are shown in Appendix A.
Table 13: Resource estimations using Xilinx Fast Fourier Transform core on Zynq-7020 for some of the different architectures with a transform size of 32 768. For more detailed information on the configuration options used see Appendix A.

<table>
<thead>
<tr>
<th>Architecture Used</th>
<th>Resource</th>
<th>Pipelined Streaming</th>
<th>Radix-4 Burst</th>
<th>Radix-2 Burst</th>
<th>Radix-2 Lite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flip-flops</td>
<td></td>
<td>6722</td>
<td>2756</td>
<td>1379</td>
<td>1101</td>
</tr>
<tr>
<td>LUTs</td>
<td></td>
<td>6415</td>
<td>2731</td>
<td>1438</td>
<td>981</td>
</tr>
<tr>
<td>BRAM (36 Kb)</td>
<td></td>
<td>38</td>
<td>42</td>
<td>33</td>
<td>36</td>
</tr>
<tr>
<td>DSP slices</td>
<td></td>
<td>21</td>
<td>9</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Transform Cycles</td>
<td></td>
<td>98 467</td>
<td>131 239</td>
<td>311 549</td>
<td>557 146</td>
</tr>
</tbody>
</table>

Table 14: Resource estimations using Altera FFT MegaCore function on Cyclone III EP3C55F484C8 for the four different FFT architectures, all with a transform size of 32 768. For more information on the configuration used see Appendix A.

<table>
<thead>
<tr>
<th>Architecture Used</th>
<th>Resource</th>
<th>Streaming</th>
<th>Variable Streaming</th>
<th>Buffered Burst</th>
<th>Burst</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic elements</td>
<td></td>
<td>7480</td>
<td>14 547</td>
<td>7 492</td>
<td>7 388</td>
</tr>
<tr>
<td>Flip-flops</td>
<td></td>
<td>6 427</td>
<td>11 563</td>
<td>6 472</td>
<td>6 367</td>
</tr>
<tr>
<td>LUTs</td>
<td></td>
<td>6 074</td>
<td>9 482</td>
<td>5 378</td>
<td>5 351</td>
</tr>
<tr>
<td>M9K</td>
<td></td>
<td>541</td>
<td>149</td>
<td>427</td>
<td>200</td>
</tr>
<tr>
<td>Memory bits</td>
<td></td>
<td>4 980 992</td>
<td>1 369 372</td>
<td>3 932 416</td>
<td>1 835 264</td>
</tr>
<tr>
<td>Embedded multiplier 9-bit</td>
<td></td>
<td>48</td>
<td>104</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Block Throughput Cycles</td>
<td></td>
<td>32 768</td>
<td>32 768</td>
<td>36 864</td>
<td>94 355</td>
</tr>
</tbody>
</table>

8.5 EtherCAT

Beckhoff Automation provides an EtherCAT Slave Controller IP core that can replace the existing EtherCAT ET1200 Slave Controller. The EtherCAT IP core is configurable, making it possible to use the same configuration as for the ET1200 or another better suited configuration. Table 15 list the features of ET1200 and the IP Core solution.
Table 15: Characteristics of ET1200 and IP Core EtherCAT controllers.

<table>
<thead>
<tr>
<th>Features</th>
<th>ET1200</th>
<th>IP Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ports</td>
<td>2 – 3 (each EBUS/MII, max-1xMII)</td>
<td>1 – 3 MII or 1 – 2 RMII</td>
</tr>
<tr>
<td>FMMUs</td>
<td>3</td>
<td>0 – 8</td>
</tr>
<tr>
<td>SyncManagers</td>
<td>4</td>
<td>0 – 8</td>
</tr>
<tr>
<td>RAM (KB)</td>
<td>1</td>
<td>1 – 60</td>
</tr>
<tr>
<td>Distributed Clocks</td>
<td>64 bit</td>
<td>32/64 bit</td>
</tr>
<tr>
<td>Process Data Interfaces</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Digital I/O</td>
<td>16 bit</td>
<td>8 – 32 bit</td>
</tr>
<tr>
<td>SPI Slave</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>8/16 bit µController</td>
<td>-</td>
<td>Asynchronous</td>
</tr>
<tr>
<td>On-chip bus</td>
<td>-</td>
<td>Avalon or PLB/OPB</td>
</tr>
</tbody>
</table>

The EtherCAT IP core is available for both Altera FPGAs and Xilinx FPGAs. Estimated resource usage, in table 16, of the IP core, with the same configuration as for ET1200, on an Altera FPGA and a Xilinx FPGA has been calculated using values given in the EtherCAT IP core Altera Datasheet and EtherCAT IP core Xilinx Datasheet.

Table 16: Estimated resources required of the EtherCAT Slave Controller IP core.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Altera Cyclone III</th>
<th>Xilinx Spartan-6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Logic elements</td>
<td>Slices</td>
</tr>
<tr>
<td></td>
<td>M9K</td>
<td></td>
</tr>
<tr>
<td>1xMII</td>
<td>900</td>
<td>150</td>
</tr>
<tr>
<td>3x FMMUs</td>
<td>1950</td>
<td>600</td>
</tr>
<tr>
<td>4x SyncManagers</td>
<td>2200</td>
<td>600</td>
</tr>
<tr>
<td>DPRAM (1 KB)</td>
<td>200</td>
<td>1</td>
</tr>
<tr>
<td>Distributed Clocks (64 bit)</td>
<td>6200</td>
<td>1600</td>
</tr>
<tr>
<td>Digital I/O (32 bit)</td>
<td>250</td>
<td>100</td>
</tr>
<tr>
<td>SPI</td>
<td>350</td>
<td>100</td>
</tr>
<tr>
<td>Avalon</td>
<td>200</td>
<td>-</td>
</tr>
<tr>
<td>PLB</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Total:</td>
<td>12250</td>
<td>3350</td>
</tr>
</tbody>
</table>

There are different licenses for the IP core depending on how and what it should be used for. The price is not available for the EtherCAT IP core and the IP core can only be purchased by members of the EtherCAT Technology Group who have signed an EtherCAT Technology Family License Agreement. For this reason it is assumed that the price for the EtherCAT IP core is the same as for ET1200.

An evaluation license for the IP core, which is full-featured but time-limited, is available for members of the EtherCAT Technology Group.
8.6 Estimation of total FPGA resource usage

This section contains the estimated resource usage of IP cores on Altera and Xilinx FPGAs, shown in Table 17.

Table 17: FPGA resource usage for different IP cores. The FFT configurations are found in Table 14, under Streaming and Burst architecture and Table 13, under Pipelined Streaming and Radix-2 Lite architecture. The estimated total resource usage is the accumulated resource estimation for the IP cores, FIR, FFT, ADC and EtherCAT.

<table>
<thead>
<tr>
<th>Core</th>
<th>Altera</th>
<th>Xilinx</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min.</td>
<td>max.</td>
</tr>
<tr>
<td>FIR</td>
<td>261 LE</td>
<td>152 flip-flops</td>
</tr>
<tr>
<td></td>
<td>164 LUTs</td>
<td>1 DSP slice</td>
</tr>
<tr>
<td></td>
<td>2 Multipliers (9-bit)</td>
<td></td>
</tr>
<tr>
<td>FFT</td>
<td>7 388 LE</td>
<td>1 101 flip-flops</td>
</tr>
<tr>
<td></td>
<td>6 351 LUTs</td>
<td>981 LUTs</td>
</tr>
<tr>
<td></td>
<td>1 835 264 Memory bits</td>
<td>36 BRAM</td>
</tr>
<tr>
<td></td>
<td>(200 M9K)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>48 Multipliers (9-bit)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>48 Multipliers (9-bit)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FAC based ADC</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min.</td>
<td>max.</td>
</tr>
<tr>
<td></td>
<td>19 899 LE</td>
<td>13 724 flip-flops</td>
</tr>
<tr>
<td></td>
<td>18 765 LUTs</td>
<td>9 863 LUTs</td>
</tr>
<tr>
<td></td>
<td>50 Multipliers (9-bit)</td>
<td>37 BRAM</td>
</tr>
<tr>
<td></td>
<td>50 Multipliers (9-bit)</td>
<td>1 BRAM (18 Kb size)</td>
</tr>
</tbody>
</table>

An assumption based on the fact that one logic element in Cyclone III consists of one flip-flop and one LUT.

8.7 FPGA/SoC FPGA

FPGA/SoC FPGA devices were researched and some selected based on the estimated resource usage for all the IP cores. The devices found, with enough resources, were Altera Cyclone III, Xilinx Spartan-6 and Xilinx Zynq.

8.7.1 Altera Cyclone III

Cyclone III is the third generation in the Altera Cyclone FPGA series and offers high performance at low power and low cost. The Cyclone III consists of up to 120k logic elements, 432 embedded memory blocks at 9 Kb each making it a total of 3 888 Kb, 531 I/O pins and 288 embedded 18-bit x 18-bit multipliers that can be used for an efficient implementation of DSP algorithms. Each logic element contains one flip-flop and one LUT.
8.7.2 Xilinx Spartan-6
Spartan-6 gives a balanced trade-off between high performance and low cost and is widely used in the industry. The FPGA has up to 23,038 slices, where each slice consists of four LUTs and eight flip-flops, 268 block RAMs at 18 Kb each corresponding to 4,824 Kb, 576 I/O pins and 180 DSP slices. A DSP slice is a piece of dedicated hardware consisting of an 18×18 multiplier and a 48-bit accumulator. DSP operations are costly to implement in the logic of the FPGA, which is the reason for dedicated DSP slices.

8.7.3 Xilinx Zynq-7000
The Zynq-7000 is a new SoC FPGA containing a dual core ARM Cortex-A9 including many communication controllers. The ARM cores have 64 KB L1 cache, 512 KB shared L2 cache, a 256 KB scratch memory and can work at 1 GHz. Among the controllers are two Gigabit Ethernet controllers and two USB 2.0 controllers. Within the ARM cores there are the Jazelle engine for Java bytecode, the NEON media-processing engine for advanced DSP calculation, doing up to 16 parallel executions, and a single precision and double precision vector floating point unit. The bus system used both in the ARM system as well on the FPGA is the AMBA AXI-bus. The FPGA is part of Xilinx 7th generation architecture and contains two 12-bit ADCs in hardware with up to 17 differential inputs. It can contain up to 54,650 slices, where each slice consists of four LUTs and eight flip-flops, 2,180 KB block RAM (545 blocks at 36 Kb) and 900 DSP slices. The DSP slices on the Zynq-7000 are made up of one 18×25 two’s complement multiplier and an accumulator 48-bit, both able to operate at up to 741 MHz.

8.7.4 Microsemi SmartFusion
The SmartFusion was suggested as a candidate at the beginning of the project due to its special blend of SoC, FPGA and programmable analog logic. The SoC has at its core an ARM Cortex M3 which does not have any floating point unit nor DSP extensions. The amount of available SRAM is 64 KB, 512 KB flash memory and the SoC ADC units (up to 3 units) have 12-bit resolution. Its maximum working speed is 100 MHz. The most powerful IC has 24 blocks of of 4608 bits RAM totaling 110,592 bits available RAM, 11,520 flip-flops, 128 I/O and 500k system gates on the FPGA part.

8.7.5 Summary
A summary of the different FPGA families mentioned, where the maximum resources available for the shown devices and additional CPU related information in case of a SoC FPGA is presented in tables 8.14 and 8.15.
Table 18: Maximum available resources for two FPGAs and two SoC FPGAs. The numbers are however not always comparable because the resources contained in a slice, logic gate, or system gate are not the same, in fact they can differ between FPGAs from the same vendor.

<table>
<thead>
<tr>
<th>Name</th>
<th>FPGA Type</th>
<th>Slices / logic elements</th>
<th>flip-flops</th>
<th>LUTs</th>
<th>Block RAM (Kb)</th>
<th>DSP slices / dedicated multipliers</th>
<th>I/O pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altera Cyclone III</td>
<td>FPGA</td>
<td>120k</td>
<td>120k</td>
<td>120k</td>
<td>3 888</td>
<td>288</td>
<td>531</td>
</tr>
<tr>
<td>Xilinx Spartan-6</td>
<td>FPGA</td>
<td>23k(^1)</td>
<td>184k</td>
<td>92k</td>
<td>4 824</td>
<td>180</td>
<td>576</td>
</tr>
<tr>
<td>Xilinx Zynq-7000</td>
<td>SoC FPGA</td>
<td>54.6k(^1)</td>
<td>437.2k</td>
<td>218k</td>
<td>19 620</td>
<td>900</td>
<td>362</td>
</tr>
<tr>
<td>Microsemi SmartFusion</td>
<td>SoC FPGA</td>
<td>500k(^2)</td>
<td>11.5k</td>
<td>N/A</td>
<td>110 592</td>
<td>N/A</td>
<td>128</td>
</tr>
</tbody>
</table>

\(^1\) One slice contains four LUTs and eight flip-flops.

\(^2\) This is not equivalent to slices/logic elements; in the datasheet they call it system gates.

Table 19: A summary of the CPU parts of the two SoC FPGAs.

<table>
<thead>
<tr>
<th>Name</th>
<th>CPU</th>
<th>RAM</th>
<th>Cache</th>
<th>CPU freq. (MHz)</th>
<th>DSP ext.</th>
<th>FPU(^1)</th>
<th>ADC</th>
<th>I/O pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xilinx Zynq-7000</td>
<td>ARM Cortex-A9</td>
<td>256 KB</td>
<td>2(^*)64 KB L1, 512 KB L2</td>
<td>1000</td>
<td>Yes</td>
<td>Yes</td>
<td>2</td>
<td>130</td>
</tr>
<tr>
<td>Microsemi SmartFusion</td>
<td>ARM Cortex-M3</td>
<td>64 KB</td>
<td>N/A</td>
<td>100</td>
<td>No</td>
<td>No</td>
<td>3</td>
<td>41</td>
</tr>
</tbody>
</table>

\(^1\) FPU = Floating Point Unit

8.7.6 FPGA prices

Prices for some specific units from the FPGA and SoC FPGA families above provided by different vendors, are shown in table 20.

Table 20: Comparison of FPGA and SoC FPGA programmable logic capacity and prices. Prices were updated 2012-11-08.

<table>
<thead>
<tr>
<th>Name</th>
<th>slices/logic elements</th>
<th>DSP Slices</th>
<th>Block RAM (Kb)</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altera Cyclone III – EP3C55F484C8N</td>
<td>55 856</td>
<td>156</td>
<td>2 340</td>
<td>$ 141.50</td>
</tr>
<tr>
<td>Altera Cyclone III – EP3C80F484C8N</td>
<td>81 264</td>
<td>244</td>
<td>2 745</td>
<td>$ 223.00</td>
</tr>
<tr>
<td>SPARTAN 6 XC6SLX45-3CSG324C</td>
<td>6 822</td>
<td>58</td>
<td>2 088</td>
<td>$ 57.19</td>
</tr>
<tr>
<td>SPARTAN 6 XC6SLX75-2CSG484C</td>
<td>11 662</td>
<td>132</td>
<td>3 096</td>
<td>$ 94.56</td>
</tr>
<tr>
<td>Zynq7000 XC7Z020CLG400</td>
<td>13 300</td>
<td>220</td>
<td>5 040</td>
<td>$ 154(^1)</td>
</tr>
<tr>
<td>SmartFusion A2F500M3G-FGG256</td>
<td>500k (^2)</td>
<td>N/A</td>
<td>108</td>
<td>$ 47.40</td>
</tr>
</tbody>
</table>

\(^1\) The price is only an estimate and the product is still new on the market.

\(^2\) This is not equivalent to slices/logic elements; in the datasheet they are called system gates.
8.7.7 Peripherals

The system used several different means to communicate between components and modules. A comparison of the relevant peripheral controllers available for each unit is shown in table 21.

Table 21: Comparison of device peripheral abilities.

<table>
<thead>
<tr>
<th>Device</th>
<th>Package</th>
<th>I²C</th>
<th>SPI</th>
<th>Ethernet</th>
<th>UART</th>
<th>USB</th>
<th>CAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyclone III</td>
<td>EP3C80F484C8N</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SPARTAN 6</td>
<td>XC6SLX75-2CSG484C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Zynq7000</td>
<td>XC7Z020CLG400</td>
<td>2</td>
<td>2</td>
<td>2¹</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>SmartFusion</td>
<td>A2F500M3G-FGG256</td>
<td>2</td>
<td>2</td>
<td>1²</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

¹ GigaEther, 10/100/1000 Mb/s controllers
² 10/100 Mb/s controller
9 System designs

The different design variations originates from the same abstract design, figure 9.1 i.e. the vibration signal enters the ADC preparation block and is converted into digital form by the subsequent ADC. Analysis of the signal can then be performed in the FPGA, which is a conceptual design unit representing different FPGA based solutions presented in the designs, Design 1, 2, 3 and 4.

9.1 ADC Preparation block

Each design proposal uses the same design for the signal preparation block, shown in figure 9.2. The change in the new design from the existing design is that the analog 6th order LP Butterworth filter is exchanged for a digital filter in the FPGA and an analog 2nd order anti-aliasing filter has been inserted.

9.2 Temperature block

The temperature signal in the existing system is filtered with a 2nd order LP filter with a cut-off frequency of about 30 Hz that removes 50 Hz supply voltage noise. A simplification of this 2nd order filter to a 1st order filter with a cut-off frequency of 15 Hz is proposed in the new design. This filters out the unwanted supply voltage noise and reduces the amount of unique components within this filter.

9.3 Shift register

Shift registers on an FPGA can replace the latches used to control the Sense and CS data flow if needed. This need is dependent on the I/O resources available on the processing unit. One of the shift registers receives a parallel bit stream of a certain width and outputs it serially, that is a parallel to serial conversion. The other shift register works the other way around, serial to parallel conversion.
9.4 **Design 1 (SmartFusion)**

In this conceptual design, figure 9.3, the high-order LP-filter and switching of chip-select signals are conceived to be implemented on the FPGA. The ARM would perform less demanding calculations. An external 16-bit ADC is used to digitize the AC signal and two on-board 12-bit ADCs are used to digitize the DC- and temperature signals. Shift registers are implemented in the FPGA together with a FIR filter. The ARM Cortex M3 performs peak-to-peak and RMS calculations and the result is passed on to the EtherCAT controller. Note that the design does not include FFT, hence the task is passed on to the CPU.

![Figure 9.3: SmartFusion based design where two 12-bit ADCs are on-chip. Only basic calculations can be performed by this system.](image)

9.5 **Design 2 (FPGA+DSP)**

This design is based on an FPGA in tandem with a DSP unit where the filtering and possibly demanding calculations like FFT are performed in the FPGA, shown in figure 9.4. Less resource demanding calculations such as peak-to-peak and RMS are done in the DSP. Note that the DSP is only a conceptual unit performing digital signal processing. Neither the DSP nor FPGA contains ADCs, therefore required ADCs are shown as external components. A 16-bit ADC is needed for the vibration signal and a 12-bit ADC is needed for the temperature signal in order to meet the system requirements. For the conversion of the DC signal a 12-bit ADC provides enough resolution. The DSP acts as a master and transmits analyzed data over EtherCAT.

![Figure 9.4: DSP and FPGA combined; The DSP does basic computation whereas the FPGA does more advanced computation.](image)
9.6 Design 3 (FPGA)

This design, figure 9.5, based on having as much functionality as possible in an FPGA requires more logic blocks, memory blocks, LUTs etc. Resources has to be shared between a soft IP core DSP and all other components. For this design a 12-bit ADC is intended to be implemented on the FPGA and also an EtherCAT controller, while a 16-bit ADC will still be in use as an external component. The 12-bit ADC converts the DC and temperature signals whereas the 16-bit ADC converts the vibration signal.

![Figure 9.5: An all FPGA based design with as much as possible performed by the FPGA.](image)

9.7 Design 4 (Zynq-7000)

The Zynq design requires an external 16-bit ADC due to the resolution requirement for the vibration signal and 12-bit ADCs are used for both the DC- and temperature signals. FFT, FIR, EtherCAT-controller and other digital components could be implemented in the programmable logic. The Dual Core ARM Cortex-A9 controls the measurement module while also performing several computations for signal processing, as shown in figure 9.6.

![Figure 9.6: The 16-bit ADC for the AC signal remains external to the Zynq-7000 unit while the unit can take care of the other conversions, system management and all calculations in its programmable logic and ARM Cortex-A9 dual cores.](image)


9.8 CPU-Module

This module has no new designs but a couple of concept ideas. In order to increase computational power the current CPU-card, which is ARM9 based, could be exchanged for an SoC FPGA with dual ARM Cortex-A9 on-board. For this to be viable the SoC FPGA needs to have the same type of communication controllers on-chip that the present card holds. Another way is to exchange the card for an ARM SoC that does what the current ARM9 based card does.
10 Implementation

The hardware and software implementation will be presented in this section. First the development boards and an overview of the design tools used will be presented.

10.1 Development Boards

Two development boards, table 22, were used during the thesis: the Atlys Spartan-6, figure 10.1, and ZedBoard Zynq-7000, figure 10.2.

Table 22: Lists the features of Atlys and Zedboard.

<table>
<thead>
<tr>
<th>Features</th>
<th>Atlys</th>
<th>ZedBoard</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPGA/SoC FPGA</td>
<td>Spartan-6 XC6SLX45-CSG324-3</td>
<td>Zynq-7000 XC7Z020-CLG484-1</td>
</tr>
<tr>
<td>Memory</td>
<td>128 MB DDR2 16 MB Quad-SPI Flash</td>
<td>512 MB DDR3 256 Mb Quad-SPI Flash 4 GB SD card</td>
</tr>
<tr>
<td>Display</td>
<td>Two HDMI video input ports Two HDMI output ports</td>
<td>HDMI output VGA output 128×32 OLED display</td>
</tr>
<tr>
<td>Communication</td>
<td>USB-JTAG 10/100/1000 Ethernet USB-UART USB-HID</td>
<td>USB-JTAG Programming 10/100/1000 Ethernet USB OTG 2.0 USB-UART</td>
</tr>
<tr>
<td>GPIO</td>
<td>8 user LEDs 6 push buttons 8 slide switches</td>
<td>8 user LEDs 7 push buttons 8 slide switches</td>
</tr>
</tbody>
</table>

Figure 10.1: The Atlys development board.  
Figure 10.2: The ZedBoard development board.

10.2 Design Tools

The design tool used for implementation was Xilinx ISE Design Suite: System Edition, which includes different software tools shown in table 23.
Table 23: Software tools included in the ISE Design Suite that were used during implementation.

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Navigator / PlanAhead</td>
<td>The starting point for a project from which HDL components, IP cores and an embedded system can be added or created.</td>
</tr>
<tr>
<td>Embedded Design Kit (EDK)</td>
<td>A development kit that contains tools to configure the hardware of an embedded system and software development.</td>
</tr>
<tr>
<td>Xilinx Platform Studio (XPS)</td>
<td>The hardware section of the development kit.</td>
</tr>
<tr>
<td>Xilinx Software Development Kit (SDK)</td>
<td>The software section of the development kit.</td>
</tr>
</tbody>
</table>

### 10.3 Bus protocol

AXI is a memory-mapped burst protocol, that means a target address has to be provided for packets sent or received and a response indicates the status of the transaction. The protocol permits a burst of packets per addressing to be done, hence a number of consecutive data packets can accompany one address. AXI-Lite is a sub-protocol of the AXI protocol that only allows one packet to be sent or received per addressing; similar to register reads or writes. AXI-Stream is a point-to-point protocol and therefore omits the address management, increasing the throughput of the transfer.

The AXI, memory-mapped, protocol has a theoretical best worst case overhead for transfers that is 50%, one address and one data, equivalent to AXI-Lite. In the best case, using a burst length of 16 when excluding AXI4 incremental mode, the overhead is 5.9%, one address and 16 data.

The burst length is limited by the version of the AXI protocol where 16 data packets is the limit for version 3 and 256 packets for version 4. AXI uses the same transfer rules as AXI-lite but with the option of burst size being larger than 1 and by this having less overhead in the communication.

### 10.4 Spartan-6 based

This section presents the HDL implementation and software implementation for the solution based upon the Xilinx Spartan-6 FPGA.

#### 10.4.1 HDL implementation

An overview of the Spartan-6 based implementation is illustrated in figure 10.3. The IP cores used will be described in the sections following. All cores, except the MicroBlaze processor, are Xilinx LogiCORE IP cores. All cores runs at 100 MHz.

*Figure 10.3: The Spartan-6 based proof of concept solution with arrows showing the components internal relations.*
10.4.1.1 FFT core

The FFT IP core was generated using the Fast Fourier Transform v8.0 with the Xilinx CORE Generator system. The FFT architecture is Radix-2 Lite Burst with a transform size of 32,768. The data format is fixed-point and the scaling option is set to “Block floating-point” that automatically scales the signal during run-time to prevent overflow. The scaling factor for a transform is output alongside the data from the component and ordering of the output data is in natural order. The scaling factor needs later be used to restore the output to correct levels. Radix-2 Lite Burst uses the least resources of all the architectures provided, but with the cost of a longer transform time.

The interface of the FFT IP core is AXI4-Stream. The core takes a real 16-bit data value as input and outputs a complex 32-bit value, where the least significant 16-bits of those are the real part and the other 16-bits are the imaginary part.

10.4.1.2 FIR filter

The FIR filter IP core was generated using the FIR Compiler v6.3 with the Xilinx CORE Generator system. The number of filter coefficients is 21, making it a 20th order filter, and the values of the coefficients were calculated with the GNU Octave code below, giving a cut-off frequency at 12.8 kHz:

```octave
coefficients = fir1(20, 12800/16384);
rounded_coefficients = round(coefficients*256);
```

Function fir1() takes two parameters as input. The first one is the order of the filter, 20, and the second one is the normalized cut-off frequency. The normalized cut-off frequency is given by dividing the cut-off frequency with the Nyquist-frequency. The coefficients was rounded, with the round() function, from floating points to integers to suit the FPGA. The coefficients are multiplied by 256 before rounded, otherwise data would be lost from truncation.

The final values of the coefficients are:

\[-0, -0, 1, -3, 5, -3, -5, 19, -37, 51, 200, 51, -37, 19, -5, -3, 5, -3, 1, -0, -0 \]

The FIR filter gives the frequency response, shown in figure 10.4, plotted with GNU Octave function freqz().

![Frequency Response of FIR Filter](image)

Figure 10.4: The frequency response of the FIR filter.

The FIR IP core uses a AXI4-Stream interface. The core takes a 16-bit data value as input and outputs a 16-bit value.
10.4.1.3 MicroBlaze CPU

The 32-bit MicroBlaze processor, a soft IP core, is part of EDK and was generated using Base System Builder in XPS. The processor runs at 100 MHz and has a local memory of 32 KB. Floating point operations are supported. All IP cores in the system are connected to the MicroBlaze, either with AXI4-Lite or AXI4-Stream. The software running on the MicroBlaze will be explained in section 10.4.2 Software implementation found below.

10.4.1.4 Timer core

The Timer core was generated in XPS using AXI Timer v1.03.a. The Timer was added for benchmarking purposes for verification of timing requirements and is not essential for the implementation to function properly. This core has an AXI4-Lite interface.

10.4.1.5 UART Lite core

The UART Lite IP core was generated in XPS using AXI UART (Lite) v1.02.a. The configuration of the core is; 460 800 baud rate, 8 bit characters, 1 stop bit and no parity bit. It has two 16-characters deep FIFOs, one for receive and one for transmit data. The IP core has an AXI4-Lite interface and a UART interface. The use of the core is to communicate with a PC, through the USB-UART peripheral on the Atlys board.

10.4.1.6 AXI4-Lite interconnect

The AXI4-Lite interconnect was generated in XPS using AXI Interconnect v1.06.a. The AXI4-Lite interconnect is used to connect one or more AXI4-Lite master devices to one or more AXI4-Lite slave devices.

10.4.2 Software implementation

The software runs on the MicroBlaze as a standalone program. That means that no operating system is used. For all the IP cores generated in XPS there exists MicroBlaze drivers which have been used for implementation. The flowchart of the software implemented, shown in figure 10.5, is explained below.

After initialization samples read from UART are sent to the FIR Filter. When all samples, in this case 32 768, have been received and sent to FIR the program moves on to the next step.

Two different solutions have been implemented. In order to distinguish the solutions from each other different line patterns have been used in figure 10.5. Both solutions reads the result from the FFT but only one writes it to UART and in this one the dotted blocks are excluded. One value is read at a time from the FFT until all 32 768 values in the result have been received and written to UART. The other solution is for measuring the time of the FFT calculation and therefore does not write the result to UART, showed as a dashed block in the figure. In this solution the first step is to start the timer followed by reading values from the FFT. Note that the result is read but never sent to UART. The read operation progresses until all 32 768 values have been received at which point the timer is stopped and the measured time is sent to UART.

For the C code of the software implementation see Appendix B.
10.5 Zynq-7020 based hardware implementation

The main cores, FIR and FFT, in the Zynq-7020 implementation, shown in figure 10.6, are identical to the Spartan-6 implementation. The difference is in the configuration of the FFT architecture. The Zynq-7020, which is an SoC FPGA, has an on-chip ARM Cortex A9 in contrast to the MicroBlaze, a
soft IP core. Components used in the hard processing system are part of the section Processing system whereas components implemented as logic blocks are part of the Programmable logic. Since the IP cores and the CPU have different types of AXI (bus) interfaces, intermediate components have to tie them together; this is the purpose of the FIFO.

10.5.1 Processing System

10.5.1.1 Cortex A9
The ARM Cortex A9 can operate at a clock frequency of 667 MHz and has 32 KB level 1 cache, 512 KB level 2 cache (shared between the two cores) and a 256 KB on-chip RAM. Its interface is AXI3 Burst.

10.5.1.2 UART controller
Through the UART input and output data is communicated to the system and corresponds to the output from an ADC and the result from the FIR and FFT calculations. Accompanying software drivers enables configuration of baud rate (bits per second) etc. from the CPU. The UART runs at a clock rate of 50 MHz.

10.5.2 Programmable Logic
All IP cores in the programmable logic are connected to the same clock that runs at 100 MHz.

10.5.2.1 FIR
Identical to the Spartan-6 implementation.

10.5.2.2 FFT
The FFT IP core was generated using the Fast Fourier Transform v8.0 with the Xilinx CORE Generator system, same as for the Spartan-6 implementation. The FFT architecture is Pipelined, Streaming I/O with a transform size of 32 768. These 32 768 samples make up one frame of data. This FFT architecture allows data to be received simultaneously as one frame of data is transformed (processed) and a previously transformed result is transmitted. The data format is fixed-point and the scaling option is set to “Scaled”, indicating that a scaling schedule for each data frame must be provided to the FFT before transformation of the data frame with the output order set to bit reversed. The scaling set during configuration must be multiplied with the result from the FFT to restore the output to correct levels.

The interface of the FFT IP core is AXI4-Stream. The core takes a real 16-bit data value as input and outputs a complex 32-bit value, where the least significant 16-bits of those are the real part and the other 16-bits are the imaginary part.

Output data from the FFT was split into transactions of 256 data values, instead of one large transaction of 32 768 values.

10.5.2.3 AXI4-Lite/AXI4-Stream FIFO Bridge
The bridge component was generated with the core AXI-Stream FIFO v.2.01.a. It translates from AXI4-Lite protocol to AXI4-Stream protocol, thus the name bridge. Apart from bridging between the two bus-protocols it also queues up the data received and sent in internal buffers with First-In-First-Out policy. It has two separate FIFOs, one for the transmit channel and another for the receive channel. The CPU, which is the master, reads from the receive channel and sends data to the transmit channel. It controls the FIFO through the AXI-Lite interface by sending instructions to certain registers and determines what data to send and when to receive data. A generated
software driver abstracts the low-level register reads and writes from the programmer. One FIFO bridge is required for the FIR for transmission and retrieval of data. The FFT needs two FIFO bridges, one for transmission and retrieval of data and another for configuration, that is, setting the scale factor to avoid overflow. The data width of the AXI-Lite and AXI-Stream interfaces are 32-bit.

10.5.2.4 AXI interconnect

The AXI interconnect was automatically generated in XPS using AXI Interconnect v1.06.a. The AXI interconnect is used to connect a AXI master device to several AXI slave devices. The ARM Cortex-A9, which is the master, uses AXI3, which is compatible with AXI4-Lite although some restrictions are put on the communication between the IP. Many restrictions are handled by the interconnect and are not an issue for the connected slave IP components. However, it is not allowed to send burst transactions of more than one word to a Lite slave.

10.5.2.5 Timer core

Identical to the Spartan-6 implementation.

10.6 Zynq-7020 based software implementation

Software was implemented in Xilinx SDK and created as a standalone application. Zynq7020 has two ARM Cortex-A9 CPUs but the software implementation utilizes only one of them.

10.6.1 UART

The UART is the input and output for the proof of concept system as mentioned in the implementation section. Prior to data is sent or received the UART driver has to be initialized and the desired baud rate must be set. The automatically generated driver can receive and send an arbitrary amount of bytes. The CPU only uses the receive functionality of the driver to receive samples. Data is sent from the CPU by calls to Xilinx printf function.

10.6.2 FIR and FFT

Communication with the FIR and FFT is similar since none of the IP cores are directly connected to the CPU. Both components have an intermediate AXI4-Lite/AXI4-Stream FIFO bridge for input and output of data, so the interface presented to the CPU will be the same. What differentiates them is that the FIR filter produces output a short delay after input has been received, determined by the length of the filter, while the FFT requires all samples belonging to a frame to be received before transformation of the input data can occur and consequently output to be produced. There is a specific order in which calls to the FIFO bridge has to be done using the API.

For reading the following order is required:

1. Check the number of samples in the receive channel of the FIFO (internal buffer). If not empty go to the next step.
2. Get the size in bytes of the data in the receive channel.
3. Receive number of bytes of data equal to the size.

Writing to the FIFO consists of the following steps:

1. Write the size in bytes of the data to the transmit channel of the FIFO.
2. Transfer the data to the FIFO.
As stated earlier, the FFT requires a second FIFO bridge through which the scaling of the FFT is set. Scaling has been set for each stage of the FFT with the total scaling at 65 536, this completely avoids overflow.

10.6.3 Timer

The software driver for the timer provide basic functionality such as initialize, start and stop timer. The resolution of the timer is determined by the clock connected to it. Since the timer runs at 100 MHz, each clock tick is equal to 10 ns.

10.6.4 Flowchart of the system

Interaction between the system, or more precisely the CPU, and the outside world is done through the UART interface and communication with the FIR and FFT is performed indirectly via the FIFO bridge, referred to as FIFO in the flowchart, shown in figure 10.7. The flow-chart displays how data is passed throughout the system. First input samples are received from the UART and sent to the FIFO connected to the FIR filter. Subsequently the receive channel of the FIR FIFO is checked for data; If a sample is present then it is read and passed on to the FIFO connected to the FFT. Samples are read from the UART until all, that is 32 768 samples, have been received. A delay exists from the point in time data is put into the FIR FIFO to the point that a result is produced. This implies that when the last sample is received from UART and passed to the FIR FIFO the result for this sample will not be ready in the receive channel of the FIFO. This is the reason for the next part of the flowchart which waits for all 32 768 samples has been received and passed to the FFT FIFO. By the time all frames have been sent to the FFT FIFO a timer is started and the program proceeds by waiting for the result of the FFT, which will become available in the FFT FIFO. After reading all samples the timer is stopped and the result is transmitted out of the system over UART.

Some extra considerations must be taken when it comes to the FFT because it produces result in bit reversed order. Samples read from the FFT FIFO are counted and stored in natural order in an array. This is achieved by bit reversing the value of the counter when a sample is received and using that bit reversed value as the index of the array.

For the C code of the software implementation see Appendix B.
Figure 10.7: Flowchart of the implemented software for the Zynq-7020 proof of concept.
11 Testing

During testing of the system the tools used were iSim v14.2, HyperTerminal v6.1, Tera Term v4.74 and GNU Octave v3.6.1. Behavioral simulation of the system was done using iSim. HyperTerminal or Tera Term was used to transmit data between the PC and processor on the FPGA. GNU Octave was used to create samples representing a vibration signal and to plot the processed data from the processor on the FPGA.

During the implementation phase, tests were done on every individual IP core in the system before they were integrated into the final system. The tests were aimed to verify the behavior but also to gain an understanding of the inputs and outputs.

Initial testing was executed using the iSim simulation tool which is included in the ISE Design Suite. iSim can be used with test benches to provide the simulated IP core with input, instead of manually setting the required signals. Test benches were created automatically when generating the FIR IP core and FFT IP core. Those test benches were modified to read samples from a file and write the results to another file. GNU Octave was used to generate the text file containing samples representing a vibration signal built up of different frequencies. It was also used to plot the results of the system and compare them with the results generated by Octave’s FIR and FFT function.

11.1 Spartan-6 based testing

The MicroBlaze was simulated in iSim with a test bench that was generated using XPS and with software implemented in SDK. In SDK an elf file was created, an executable file of the software, and by adding and associating it with the test bench the software could be simulated on the MicroBlaze. The initial test of the MicroBlaze was to read and write over the AXI4-Stream interface since both the FIR and FFT IP core use that interface. After this had been tested and verified the whole system was put together.

The system behavior was tested by running it on the Atlys board and using Tera Term on the PC to send and receive data to/from the MicroBlaze over a serial port. Data, 32 768 samples of a vibration signal, created in GNU Octave, was sent to the MicroBlaze. When the data had been processed by FIR and FFT the result was sent back to the PC where GNU Octave was used to plot it. Correctness of the system was verified by calculating FIR and FFT in GNU Octave on the same vibration signal sent to the MicroBlaze. The result from GNU Octave was plotted and compared with the result from the FPGA.

The timing requirement of the system was verified using the timer IP core, which measured the number of clock cycles taken for the 32768 point FFT calculation. The timer was started when the FFT had obtained the last input sample and stopped when it had sent out the last data value.

11.2 Zynq based testing

Testing of the Zynq based implementation in iSim was done to some extent to understand the individual IP cores, but the main reason was to verify individual IP cores and the system as a whole. The simulation tool cannot simulate the ARM Cortex A9 since it is hardware of the SoC and not an IP core. This requires a different approach in terms of verification and validation.

Xilinx has IP cores available dedicated to testing and debugging. The key component used for this purpose was the ChipScope AXI Monitor which aids in debugging of an AXI interface. Connecting the IP core to an AXI bus in XPS allows the signals to be viewed similarly to the simulation in iSim. Both AXI, the memory-mapped interface, and AXI-Stream can be debugged by configuring the IP core appropriately. There are numerous settings of interest, the number of samples to store being one. In order to control the AXI monitor and view the waveforms from the system on the PC an additional component is required, namely the ChipScope Integrated Controller, shown in figure 11.1, that communicates with the AXI Monitor through the JTAG port. After the bitstream is
downloaded to the FPGA, a software tool, ChipScope Analyzer Software, is used to set triggers and view the waveforms from the system. A trigger is a user defined state of a signal or a boolean combination of multiple signals.

The FIFO bridge were tested using the AXI Monitor. To monitor the bridge’s AXI stream interface and AXI interface simultaneously two AXI Monitor IP cores were utilized.

Input to the system was generated in GNU Octave and contained 32 768 samples of a vibration signal. The input was sent to the system through UART by using the software tool HyperTerminal. The same tool was also used to obtain the result of the system, that is, a filtered signal and the FFT of that signal. GNU Octave was used to plot the result of the system. The system’s validity and correctness was tested by applying a FIR filter with the same coefficients as used in the system and compute the FFT of that signal within GNU Octave and then compare it with the result obtained from the system.

The FIR filter was also tested individually by transmitting the result from the FIR back to the PC and plotted and compared with the result computed by GNU Octave. For this test the input data was 100 signals with constant amplitude and linearly spaced frequencies.

The time was measured using the Timer IP Core. The time it takes for data to pass through the FFT and sent to the CPU is where the time restriction on the system is, therefore the timer is started when the last sample is transmitted to the FFT and stopped as the last data value is received by the CPU. This yields time required by the system to complete.
12 Results

Different FFT architectures and their Estimated Time, Estimated Calculation Time and Measured Time together with transfer time are presented in table 24.

- Estimated Time is obtained from the Core Generator and reflects the time for the transformation from that the first input is received till the last value is sent from the IP core, where output is transmitted over AXI-stream without stalls.
- Estimated Calculation time is an estimate of the true calculation time obtained from the Estimated Time. At 100 MHz a transfer of 32 768 values would take at least take 1 clock cycle per value in one direction, resulting in 0.65 ms for both directions. The estimated calculation time is then 0.65ms subtracted from the Estimated Time.
- Measured Time is the time from the last value is sent from the CPU until the last value is received by the CPU.

For Pipelined Streaming two times for Measured Time are given, the first includes an order conversion required to get a bit reversed result in natural order whereas the latter within parenthesis is without this conversion.

Table 24: Measured and estimated times for the different FFT implementations on Zynq-7000 and on Spartan-6.

<table>
<thead>
<tr>
<th>FFT Module</th>
<th>Frequency (kHz)</th>
<th>Estimated Time (ms)</th>
<th>Estimated Calculation Time (ms)</th>
<th>Measured Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipelined Streaming (Zynq)</td>
<td>100</td>
<td>0.98</td>
<td>0.33</td>
<td>9.39</td>
</tr>
<tr>
<td>Pipelined Streaming (Zynq)</td>
<td>100</td>
<td>0.98</td>
<td>0.33</td>
<td>9.81 (9.07)³</td>
</tr>
<tr>
<td>Radix-4 Burst (Zynq)</td>
<td>100</td>
<td>1.31</td>
<td>0.66</td>
<td>9.89</td>
</tr>
<tr>
<td>Radix-2 Lite Burst (Zynq)</td>
<td>100</td>
<td>5.57</td>
<td>4.92</td>
<td>14.15</td>
</tr>
<tr>
<td>Radix-2 Lite Burst (Spartan-6)</td>
<td>100</td>
<td>5.57</td>
<td>4.92</td>
<td>9.32</td>
</tr>
</tbody>
</table>

³Additional resources are required for output in natural order and the scaling option, Block Floating Point.
²Scaling by fixed schedule and output in bit-reversed order.
³Time without software reordering of results to natural order.

12.1 Estimated cost for Design 3 and Design 4

The estimated cost, table 25, for the partially implemented designs, Design 3 and Design 4, are based on the cost for the main components, 16-bit ADC, FPGA and Design Tool. The Xilinx Design Suite includes the production licenses for IP cores such as FIR, FFT, ADC and others. Prices in table 25 were updated 2012-11-08.
Table 25: Estimated partial cost for Design 3 and Design 4.

<table>
<thead>
<tr>
<th>Component</th>
<th>Cost for Design 3 (Spartan-6)</th>
<th>Cost for Design 4 (Zynq-7020)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-bit ADC (MAX11046ECB+)</td>
<td>130.00 SEK</td>
<td>130.00 SEK</td>
</tr>
<tr>
<td>FPGA</td>
<td>381.30 SEK^2</td>
<td>1029.00 SEK^3</td>
</tr>
<tr>
<td>Design Tool (Xilinx ISE Design Suite: System Edition)</td>
<td>34.60 SEK^1</td>
<td>34.60 SEK^1</td>
</tr>
<tr>
<td><strong>Total price:</strong></td>
<td><strong>545.90 SEK</strong></td>
<td><strong>1193.60 SEK</strong></td>
</tr>
</tbody>
</table>

^1 Based on the fact that Xilinx ISE Design Suite: System Edition cost 5 195 USD per year and 1000 units will be produced per year.

^2 SPARTAN 6 XC6SLX45-3CSG324C

^3 This price is for an early version of the unit, XC7Z020-CLG484-1.

In the existing system the total cost for the DSP, 102.28 SEK, and two 16-bit ADCs, 2×433.40 SEK, latches, analog switches and operational amplifiers was 1066.12 SEK. In design 3 using Spartan-6 the corresponding components are 51.2% less expensive and in design 4 using Zynq-7020 they are 12.0% more expensive.

### 12.2 Spartan-6 based results

#### 12.2.1 Behavioral results

The frequency spectrum plotted in the left half of figure 12.1 is the result from the GNU Octave calculation and the right half of figure 12.1 shows the result from the Spartan-6 FPGA based system. Both results were produced from the same input vibration signal, created in GNU Octave, composed of four sine waves with frequencies at 5 kHz, 13.5 kHz, 15.063 kHz and 15.967 kHz and all with an amplitude of one. Frequencies above the cut-off frequency, 12.8 kHz, are attenuated whereas the frequency at 5 kHz below remains unchanged. From the figure it can be observed that the frequency at 15.967 kHz is completely filtered away. The amplitude of the FPGA based result is scaled down with 4, before plotted, to be in the same scale as the GNU Octave based since the FIR and FFT core in the Spartan-6 system scales the signal to keep the values in range.

![GNU Octave based FIR and FFT Frequency Spectrum](image1)

![FPGA based FIR and FFT Frequency Spectrum](image2)

*Figure 12.1: The left figure shows the result calculated with GNU Octave and the right the result from Spartan-6.*

In figure 12.2 is a result from a FIR and FFT calculation in GNU Octave and from a FPGA based FIR calculation with FFT done in GNU Octave on a vibration signal, created in GNU Octave, composed of 100 sine waves with an amplitude of one and linearly spaced frequencies in the range zero to the Nyquist frequency.
In figure 12.3 is a result from a FPGA based and GNU Octave FIR and FFT calculation on a vibration signal, created in GNU Octave, composed of 100 sine waves with an amplitude of one and linearly spaced frequencies in the range zero to the Nyquist frequency.

12.2.2 Timing performance

The time it takes for the MicroBlaze to receive the result from the FFT after the last data value of the frame is sent is approximately 9.32 ms. One frame contains 32 768 data values.

12.2.3 Resource usage

The FPGA resources used in the Spartan-6 based implementation is given by table 26. The values are calculated in Xilinx ISE Project Navigator v14.2. The majority of the used BRAM blocks is used by the FFT IP core, 72 out of 88, while the rest is used by the MicroBlaze.
Table 26: Summary of required resources on the Spartan-6 FPGA for the proof of concept solution.

<table>
<thead>
<tr>
<th>Resources</th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slices</td>
<td>1347</td>
<td>6822</td>
<td>19%</td>
</tr>
<tr>
<td>Slice Registers</td>
<td>3516</td>
<td>54576</td>
<td>6%</td>
</tr>
<tr>
<td>Slice LUTs</td>
<td>3978</td>
<td>27288</td>
<td>14%</td>
</tr>
<tr>
<td>DSP Slices</td>
<td>12</td>
<td>58</td>
<td>20%</td>
</tr>
<tr>
<td>BRAM (18 Kb)</td>
<td>88</td>
<td>116</td>
<td>75%</td>
</tr>
<tr>
<td>IOBs</td>
<td>4</td>
<td>218</td>
<td>1%</td>
</tr>
</tbody>
</table>

12.3 Zynq based results

12.3.1 Behavioral results

The first IP component in the FIR-FFT chain is the FIR, hence the result of that test is presented prior to the result of the complete system. Presented, in figure 12.4, is the FFT of the vibration signal consisting of 100 sinusoids with an amplitude of one and linearly spaced frequencies in the range zero to half the sampling frequency (the Nyquist frequency).

The result of the whole system is seen in figure 12.5, which used the same sample data as input for the FIR test.
12.3.2 Timing performance

The time it takes to perform the FFT and until the result is available at the CPU is 9.81 ms for the Pipelined, Streaming I/O architecture. Timing information about the system obtained from PlanAhead 14.2 after implementation states that the minimum period of the system is 9.4 ns and a maximum frequency of 106.2 MHz.

12.3.3 Resource usage

The FPGA resources required by the Zynq-7000 implementation are shown in the table 27 below. The resource usage were retrieved from Xilinx tool PlanAhead 14.2. Since the resource usage is supposed to represent the resources required by the proof of concept implementation the IP cores utilized during debugging are excluded from this implementation.

<table>
<thead>
<tr>
<th>Resources</th>
<th>Used</th>
<th>Available</th>
<th>Utilization1 %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slices</td>
<td>2413</td>
<td>13300</td>
<td>18</td>
</tr>
<tr>
<td>Slice Registers</td>
<td>8174</td>
<td>106400</td>
<td>7</td>
</tr>
<tr>
<td>Slice LUTs</td>
<td>7452</td>
<td>53200</td>
<td>14</td>
</tr>
<tr>
<td>Slice LUT – Flip Flop pairs</td>
<td>8100</td>
<td>53200</td>
<td>15</td>
</tr>
<tr>
<td>DSP48</td>
<td>22</td>
<td>220</td>
<td>10</td>
</tr>
<tr>
<td>RAMB18s</td>
<td>66</td>
<td>280(^2)</td>
<td>23</td>
</tr>
<tr>
<td>RAMB36s</td>
<td>11</td>
<td>140(^2)</td>
<td>7</td>
</tr>
</tbody>
</table>

1 Values are from Xilinx tool and are truncated, not rounded.
2 The amounts of RAM available if all BRAM is configured to be either RAMB18 or RAMB36. The system does not contain these amounts put together.
13 Discussion (Analysis of results, Recommendations)

The ideal implementation of the system would be to fit all functionality of the existing system onto the FPGA, including ADCs and anti-aliasing filters residing in the Filter Block and ADC Block. Each vibration measurement channel in the existing system has a block consisting of a 6th order LP Butterworth filter. It allows frequencies within the range 0-12.8 kHz to pass. The filter is a block that can be replaced with an FIR-filter in the FPGA. Removing the LP filter prior to the ADC also removes the anti-aliasing functionality. Moving the functionality of those blocks into the FPGA would require a higher sampling frequency than 32 kHz in order to prevent aliasing. Due to this, passing the input vibration signal directly to the FPGA leads to a more complicated sampling implementation since it has to manage all possible input frequencies without aliasing them. A simple anti-aliasing filter makes sure that only the frequency band of interest is passed to the FPGA. This advantage is the reason why a filter is preserved as an external component, prior to the ADC.

An FPGA-based implementation of the ADC with up to 14-bit resolution is provided by Stellamar and implementations with lower resolution are available as Xilinx IP cores. ADCs digitized in the FPGA are relatively new as compared to conventional ADCs and can therefore not yet achieve the same performance. For the time being a 16-bit ADC with 32 ksp.s capacity is still within the realm of research and development. Because of that the 16-bit ADC can not be replaced with an FPGA based ADC solution and an external 16-bit ADC is used in all of the proposed designs. Although there exist methods to perform an ADC using FPGA there were not enough time left for this to be included in the proof of concept system.

The aim of the project was to create a proof of concept vibration analysis tool and not to create a somewhat efficient FPGA implementation of the FFT, which was more or less what it boiled down to considering the complexity of the algorithm. Something more sophisticated than the simplest case had to be implemented to meet the timing requirement and this task would probably exceed the time frame of the project. Due to the mentioned reasons the idea of creating an FFT component from scratch was abandoned and available FFT IP cores were favored instead. A difficulty regarding the FFT was the transform size of 32 768 points. There exist several FFT IP cores but only a few support a transform size of 32 768. Both Altera and Xilinx provides cores that meets this requirement.

13.1 Design 1

The SmartFusion based design was initially a strong candidate for the new system since it has the capability to implement digital solutions in the FPGA, contains an ARM Cortex M3 core and configurable analog components. The major downside is the FPGA, which has a low amount of resources and thus only basic IP cores, FIR and Shift registers, could be possible to place onto the FPGA. A closer look at the ARM Cortex M3 reveals that its computational capacity is not sufficient to perform the more demanding tasks e.g. FFT, and therefore limited to less demanding calculations such as peak-to-peak and RMS. Moreover, the core does not have an FPU nor DSP extensions, required for many of the DSP algorithms. The unit has 64 KB SRAM and theoretically that amount of RAM would be enough to hold one frame but then there would be no memory left for instructions. To summarize, this option lacks the required resources for this project.

13.2 Design 2

In this design a DSP was intended as co-processor so that only a low-cost FPGA would be required. The advantage with this design is that the workload can be split between two possibly cheaper components without the need for one powerful FPGA solution but with a downside of increased use of board space. In the case of using one of the researched ARM SoC as the DSP unit the 12-bit ADCs would be on-chip and not required as external components. Due to the resource demanding 32 768-point FFT nearly the same FPGA as for Design 3 is required and
therefore a DSP unit as co-processor does not seem like the best solution.

13.3 Design 3

This is an all on FPGA design where even the DSP unit is implemented as a component along with the FIR, FTT, shift registers and EtherCAT controller. This solution would put the most demanding requirements on the FPGA as having a specialized processing unit alongside the other components would require the FPGA to have a fair amount of resources and the conversion data from the ADCs adds to the memory demands upon the FPGA as this data needs buffering at different stages for calculations to progress efficiently. For this design a proof-of-concept solution has been made which works within the allotted time for performing FIR and FFT. The FPGA chosen for this solution was Xilinx Spartan-6 because of its lower price, as compared to the Altera Cyclone III. Spartan-6 has enough resources to fit the whole design and is supported by many IP cores. This solution utilizes a MicroBlaze CPU core which has potential top speed of 200 MHz but in the implementation is limited to 100 MHz. This processor takes the place of the DSP in the design and due to its rather low working speed any overhead in communication will have less impact on the overall time the calculations takes. Additionally as there exists 12-bit ADC solutions as IP components these can be implemented as well, although the 16-bit must remain as an external IC unit.

It can be seen that the results, figures 12.1, 12.2 and 12.3, of the Spartan-6 based system are almost identical to the results obtained from GNU Octave. Minor differences between the two occur as actual implementations will always have some differences from estimations. This means that the Spartan-6 based system seems to function properly. The FFT scales the result and sends out the scale factor applied, which is ignored for this proof of concept implementation. Instead the scale factor is determined afterward by comparing it with GNU Octave’s result. This is sufficient for the proof of concept system.

This design demonstrates a proof-of-concept solution where the measured time of the 32 768-point FFT is approximately 9.32 ms, which is well below the requirement of 50 ms. It is however not as good as the estimated time, 5.57 ms, but that is because the MicroBlaze can not read a data value every single clock cycle, due to the fact that executing the read function requires some clock cycles before the data can be read. The Estimated Calculation Time is 4.92 ms, thus the transfer time out from the FFT is 4.4 ms and the transmission overhead becomes 47%.

The overall resource utilization of Spartan-6 is less than 20%, except for the BRAM blocks which are 75% utilized. This still leaves enough resources for the components not yet implemented in the system such as the EtherCAT IP core, 12-bit ADC IP core, 16-bit ADC interface etc. The estimated price for this design would be lower than the existing system and because components are moved into the FPGA less components need to be mounted on the PCB, thus the production cost is reduced.

13.4 Design 4

The Zynq-7000 based design is conceptually and on paper an interesting design, utilizing a cutting edge unit with dual ARM Cortex A9 cores along plenty of peripheral controllers and a medium sized FPGA. More specifically the Zynq-7020 unit was used for the proof of concept implementation of this design. It contains two 12-bit ADCs which are more than enough to manage the conversion of both temperature and vibration DC signals. The vibration AC signal is as mentioned passed through an external ADC before it enters the Zynq-7000 unit. Everything seems as if setup to have an optimal solution with no lack of computational power nor a system that would need much time to perform its tasks, but with this unit being so cutting edge there are some problems that was not foreseen at the discovery of the Zynq-7000.

The implementation of this design needs an IP-component acting as a bus bridge between the AXI and the AXI-stream buses as these are not directly compatible. No such dedicated component exists yet. There are however the AXI-lite to AXI-stream FIFO which can be used as a bridge, but
sadly has a low working frequency at 150 MHz as maximum speed. Due to the AXI-Lite interfaced FIFO there is an added overhead to transactions. Each transaction results in at least two packets being sent over AXI-Lite, one for the address and another for the data and response. The transmit operation for one data value consists of two transactions: set length of the data to send and send data. This results in an overhead of at least 75% (four packets required for one data packet). Retrieval of a single transfer results in a slightly greater overhead because an additional transaction is required; The transactions are: check for data in the receive channel, get length of received data and receive the data, consequently the overhead becomes 83.3%. This is under the assumption that data of length 4 bytes (32-bits), one value, is received. In the case of multiple values and therefore a greater data length, for example eight, a common length seen during the testing, the overhead is decreased to 75%. Note that the overhead from the CPU is unaccounted for and the true overhead is even worse/greater than presented. Control of the FIFO is handled by the CPU through the software driver, provided by Xilinx, which further adds to the communication overhead since the FIFO operations take additional clock cycles.

During implementation the FIR filter was the first component implemented with the need of a FIFO bridge. It passed the testing phase and gave correct results. However in the case of the FFT, writing data over AXI-Stream to the FIFO turned out to be a big problem due to the fact that it did not follow the protocol at high throughput. Much data were lost between the two components, one third to be more precise, and the amount of dropped data was consistent between multiple runs. Testing with ChipScope made it possible to see that every third data was lost, by comparing the output from the FFT AXI-Stream interface and the actual output retrieved from the FIFO. Since it is not that difficult to verify the AXI-Stream protocol manually, by inspection of the waveforms presented in ChipScope, the conclusion that the data stream from the FFT was correct and followed the protocol could be made. This means the FIFO could not handle the high throughput of the FFT. A delay of one clock cycle was added at the end of each transaction from the FFT to slow down the communication, something that proved to be the solution. This addition in turn also adds to the already large overhead for the FIFO as a bus bridge between the AXI-stream and AXI-lite making this even more of a communications bottleneck, but it works. The size of the transaction was chosen to be 256 data values, which reduces the overhead from an unnecessary amount of delay cycles; Greater transaction size means less transactions and reduced delay. The size of the FIFO limits the transaction size since it can hold at most 512 values, so by setting the transaction size to half of the FIFOs cache size potential data can be written to and read from the FIFO simultaneously.

Estimated time for the Pipelined Streaming FFT is approximately 0.98 ms. This includes the time to transmit data to the IP core, the FFT calculation and to transmit the result from the FFT. The estimated calculation time is 0.33 ms. Subtracting this time from the time not including the order conversion yields an estimated transfer time for data out of the FFT of 8.74 ms. It becomes apparent that the transmission of data is a bottleneck and responsible for 89% of the total Measured Time, being 9.81 ms which includes the order conversion. The time required to convert the FFT result into natural order is the difference between the Measured Time including the order conversion and the time excluding it, 0.74 ms, which is more than twice the estimated calculation time for the FFT. It is unfortunate that bit reversal has to be performed in the CPU since the FFT can be configured to send the index of the data alongside the data. However, this requires an additional bus and the FIFO bridge is limited to one 32-bit wide data bus. The use of two FIFO bridges would remove the inherent synchronization between data and index due to the control signals of the AXI-bus. Another option would be to configure the FFT to output data directly in natural order. The downside of this is a significant increase in required resources.

The Measured Time for the Radix-4 Burst architecture is almost identical to the Pipelined Streaming. When compared to the Pipelined architecture this one requires less resources in general, even with output in natural order enabled. Converting to natural order directly in hardware is superior to the approach of performing it in software since the Measured Time for Pipelined Streaming and Radix-4 Burst is almost the same despite that Radix-4 Burst requires approximately twice the time of the Pipelined Streaming, having an Estimated Calculation Time of 0.66 compared to 0.33. Estimated overhead for Radix-4 Burst due to the transmission is 93%. Radix-2 Lite Burst
demands substantially less resources than the other architectures but the time it takes to calculate
the FFT is more than 4 ms longer.

Although having the calculations done within 10 ms, or even 15 ms, is well within the requirement
of 50 ms, exchanging the FIFO for a faster and more optimal bridging component would do much
for this solution. Before the solution of adding delay to the FIFO bridge were discovered, effort
were put into creating a bridge component from scratch tailored for the needs of the
implementation. The component were tested by simulation but tools needed to efficiently debug the
component attached to the ARM Cortex were lacking, making this task hard. The issue of bridging
between AXI-Stream and AXI were the main difficulty for this implementation and caused the
project to be delayed. For this reason the in-house bridge was not completed when the FIFO
bridge provided by Xilinx was altered to function.

Plotted results, figures 12.4 and 12.5, are similar to the results calculated with GNU Octave. The
shape of the Zynq-7020 based FIR and FFT results seem identical to the results obtained from
GNU Octave but the scaling differs in the case of the FIR filter due to truncation. The amplitude of
the result from the FPGA based FIR is lower than the calculated result from GNU Octave. From
result of the complete system, figure 12.5, it can be seen that the amplitude of the FPGA based
result has an amplitude four times lower than the result from GNU Octave. Note that scaling is
performed by the FFT as well but since a scaling schedule with a fixed scale factor is provided, the
unscaled result is obtained by multiplying the output with the known scale factor.

For the proof of concept implementation less than 20% of the total amount of slices, slice LUTs etc.
are utilized. The highest utilization is of RAM blocks; 23% RAMB18s and 7% RAMB36s. The result
regarding the memory, obtained from Xilinx resource usage summary, could be misleading due to
the fact that the total memory utilization is shown both in blocks of 18 bits and 36 bits. This could
give the impression that twice the amount of memory is available, which is false, making the
memory utilization in total 31.4%.

Despite these drawbacks this design presents a system for a proof of concept fulfilling the timing
requirements.

13.5 Testing

Measurement of the performance, i.e. the time it takes to compute the result, was done with the
Timer IP core. This is not free in the terms of resources but its footprint is small and does not
increase the resource requirement of the system substantially. The timer was started when the
CPU sent the last sample to the FFT and stopped when the last value was received by the CPU.
The FIR filter does not have the same requirement as the FFT on time efficiency because of the
sample rate at 32 kHz, as it will take one second to receive the complete frame of 32 k samples
independently of the FIR's efficiency.

Testing of the implemented systems, Spartan-6 and Zynq-7000 based, have utilized the UART for
input and output. This is enough to show the validity of the system. In the early stages of the
implementation testing were thought to be done on the existing system, something that would
require it to be slightly modified in order for the ADC's output to be tapped and forwarded to the
new system. Additionally, the existing system had the functionality of the FIR filter preceding the
ADC, hence that filter would need to be bypassed or the signal would be filtered twice and not give
a correct response of the system.

13.6 Cost

Analog components, such as filters, occupied a large part of the board space for the measurement
module. Removal of the filters and other analog components reduced component cost and would
also reduce the mounting cost. All unique components incur a cost for mounting the component to
the circuit board. This cost is unknown to us and can therefore not be taken into consideration.
Likewise both the Spartan-6 and the Zynq-7000 would need capacitors, voltage supply, ROM and
other specific components for which cost has not been calculated as these needs are partly
unknown. Comparing the price of known components for the Spartan-6 implementation, based on design 3, with the Zynq-7000 implementation, based on design 4, a large price difference can be observed. The Zynq-7000 implementation is about twice as expensive. Moreover, the Spartan-6 proof of concept implementation fulfills the price requirement by reducing the price to the existing implementation by 51%, whereas the Zynq-7000 proof of concept implementation exceeds that amount by 12%. All in all with efficiency and cost perspectives the Spartan-6 solution of design 3 is the recommended solution at this time with the Zynq-7000 being an exciting component for future work and next generation solutions.
14 Future work

The thesis work was only to make a proof of concept on an FPGA. Therefore a lot of important functionalities of the vibration monitor instrument which are not yet implemented. These functionalities will be presented in this section as future work together with other areas of improvement.

The Zynq-7000 based implementation is a very interesting option for future versions. If the communication bottleneck and overhead from AXI-Lite is removed this design could be developed into a very potent system with capabilities far exceeding that required for this project. To remove the communications bottleneck a better bridging component needs to be found or developed. Developing an efficient AXI4-stream to AXI4 bridge component is a future work that could release much of the potential in the Zynq-7000. A different approach than the one presented in the Zynq-7000 based design would be to let the two on-chip cores do all calculations. In the current system the CPU module has one ARM9 CPU which has been used to perform up to 10 FFT calculations within 1 second. The ARM9 belongs to the 5th version of the ARM architecture and the ARM Cortex-A9 core is of the 7th version. The Cortex-A9 has more cache, more on-chip SRAM, better FPU and DSP capabilities as well as higher working frequency and the logical conclusion is that such a dual core should have no problem to perform 8 FFT calculations and other tasks in the time required before new frames of sampled data arrive.

Focus of the implementation was the FFT because it was time consuming in the existing solution and the FIR since the analog filter could be removed as an external component. However, the vibration measurement system must perform time domain analysis as well. This involves functions such as RMS and P-P. These functions can be implemented in either software and run on the MicroBlaze/ARM Cortex-A9 or in programmable logic as coprocessors.

Neither input nor output of the system is implemented the way it is meant to be in a complete system, since UART is used at the time. There are three different types of input data: vibration signal, DC signal, temperature signal in addition to sensor configuration from jumper settings. For the vibration signal the system misses an interface to the 16-bit ADC, presented in the designs, in order to be able to receive sampled data. A low resolution digital ADC IP core will be added for the DC and temperature measurement for the Spartan-6 based system. The Zynq-7020 based system could utilize the same type of low resolution ADC but it has two 12-bit ADC on-chip, which is not used in the proof of concept implementation. These 12-bit ADC could be used for both DC and temperature measurement. The passing of the jumper settings for the sensor configuration is in the current system managed with a switched bus system, but can be managed in the FPGA by shift-registers. Shift-registers can also be used to manage outgoing configuration signals to the components of the module as the existing system has outgoing chip select signals from the DSP to external components such as ADCs and instrumentation amplifier. Depending on the future implementation the amount of chip select signals could vary but some control signals will be required. Communication with the CPU module must take place to retrieve requests of what to compute and the result of the computation must be transmitted back to the CPU-module. EtherCAT is used for this purpose so an EtherCAT Controller IP core needs to be added, which in the current system is an external IC on the board.
15 Summary and conclusions

“The aim of this work is to find a method to translate the existing processing system composed of both analog and digital components into a mostly digital system using an FPGA and show the viability of that method. The main goal of the new version is to increase performance and decrease the production cost.”

The work started with identification of functional blocks in the current vibration monitor instrument and a tally of the cost of the components in this system. Replacements for the identified blocks were searched for, both digital replacements possible to implement on an FPGA as well as IC components. Although the latter one as a secondary option since the optimal solution would be to implement all components of the vibration monitor instrument on the FPGA. Four different designs were presented and two of those were partially implemented as proof of concepts, wherein one, a Spartan-6 FPGA was used and in the other a Zynq SoC FPGA. Both implementations shows a proof of concept of a vibration monitor instrument by passing input data through a 20th order FIR filter and a 32 768-point FFT core within the required time. The measured time for the Spartan-6 based implementation is 9.32 ms and 9.81 ms for the Zynq based.

Currently the available bridge components limit the potential of the Zynq implementation. The Spartan implementation’s Measured Time is lower than that of the Zynq-7000’s even when the former uses the most resource-economic architecture and the latter uses the best architecture alternative for performance. Since the Zynq is equipped with two ARM Cortex A9 cores it has potential to perform much of the CPU-module’s functionality in the existing system, although it is not included in this proof of concept implementation. Finding or developing a bridge component that is 50% more efficient than the currently used component would make the Zynq-7000 solution about 4 ms faster than it currently is. It is perfectly reasonable to assume that the ARM Cortex A9 dual cores can perform all the work within the required time, although this would severely under utilize the FPGA part of that unit. The Zynq-7000 is not currently a good choice, but clearly an option for the future.

The Spartan-6 implementation gets the job done and is as of today the better solution in terms of performance as well as cost, being 51.2% less expensive than the current solution and 46% of the Zynq-7000 implementation’s price. The MicroBlaze can perform simpler calculations required to replace the DSP of the current system.
16 References

17  Appendix A

17.1  IIR description

The IIR filter [UMB] compared to the FIR filter has a recursive part in addition to the non-recursive part. Figure X, copied from [UMB], shows an example of a canonical IIR filter where the non-recursive and recursive parts are merged together.

Equation 3 is used to calculate the output of the IIR filter where $b$ is the coefficients of the non-recursive part and $a$ is the coefficients for the recursive part. $x$ is the input samples for the non-recursive part and $y$ for the recursive part. $L$ is the number of filter coefficients.

\[
y[n] = \sum_{l=0}^{L-1} b[l] x[n-l] + \sum_{l=1}^{L-1} a[l] y[n-l] \quad n = 0, 1, \ldots
\]  

\(3\)
17.2 FFT configurations

The following FFT configurations, shown in table 17.1, were used for the implementations using Xilinx CORE Generator in ISE Project Navigator 14.2.

Table 28: Xilinx's FFT configuration options. Note that options not used are discarded from the configuration.

<table>
<thead>
<tr>
<th></th>
<th>Pipelined Streaming</th>
<th>Radix-4 Burst</th>
<th>Radix-2 Burst</th>
<th>Radix-2 Lite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channels:</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Transform size:</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
</tr>
<tr>
<td>Data format:</td>
<td>Fixed Point</td>
<td>Fixed Point</td>
<td>Fixed Point</td>
<td>Fixed Point</td>
</tr>
<tr>
<td>Precision options</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input data width:</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Phase factor width:</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Scaling option:</td>
<td>Scaled</td>
<td>Block Floating Point</td>
<td>Block Floating Point</td>
<td>Block Floating Point</td>
</tr>
<tr>
<td>Rounding modes:</td>
<td>Truncation</td>
<td>Truncation</td>
<td>Truncation</td>
<td>Truncation</td>
</tr>
<tr>
<td>Output ordering:</td>
<td>Bit/Digit reversed</td>
<td>Bit/Digit reversed or Natural order</td>
<td>Bit/Digit reversed or Natural order</td>
<td>Bit/Digit reversed or Natural order</td>
</tr>
<tr>
<td>Throttle schemes:</td>
<td>Non Real Time</td>
<td>Non Real Time</td>
<td>Non Real Time</td>
<td>Non Real Time</td>
</tr>
<tr>
<td>Number of stages</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>using block ram:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Butterfly arithmetic:</td>
<td>Use CLB logic</td>
<td>Use CLB logic</td>
<td>Use CLB logic</td>
<td>Use CLB logic</td>
</tr>
</tbody>
</table>

The configurations used for the Altera FFT cores are shown in table 17.2. The FFT cores were generated with Altera FFT MegaCore Function v12.0 in Quartus II v12.0.

The ordering option marginally affects the resources used.
Table 29: Altera’s FFT configuration options.

<table>
<thead>
<tr>
<th>Parameters</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Channels</td>
<td>1</td>
</tr>
<tr>
<td>Transform size</td>
<td>32 768</td>
</tr>
<tr>
<td>Data format</td>
<td>Fixed Point</td>
</tr>
<tr>
<td>Data Input Precision</td>
<td>16</td>
</tr>
<tr>
<td>Twiddle Precision</td>
<td>16</td>
</tr>
<tr>
<td>Scaling option</td>
<td>Fixed-point, natural word growth for Variable Streaming and Block Floating Point for the other architectures.</td>
</tr>
<tr>
<td>Output ordering</td>
<td>Bit Reverse Order</td>
</tr>
<tr>
<td>FFT Engine Architecture(^1)</td>
<td>Quad Output</td>
</tr>
<tr>
<td>Number of Parallel FFT Engines(^1)</td>
<td>2</td>
</tr>
<tr>
<td>Structure(^2)</td>
<td>4 Multipliers / 2 Adders</td>
</tr>
<tr>
<td>Implement Multipliers in(^2)</td>
<td>DSP Blocks / Logic Cells</td>
</tr>
</tbody>
</table>

\(^1\) This parameter can only be modified in the two Burst architectures.

\(^2\) This parameter can’t be modified for the Variable Streaming architecture.
Appendix B
Code for the software implementations are presented in this section.

18.1 Code for Design 3

18.1.1 Analysis result

```c
#include <stdio.h>
#include "xparameters.h"
#include "xil_cache.h"
#include "uartlite_header.h"
#include "xuartlite.h"
#include "xbasic_types.h"
#include "fsl.h"

/* Receives and returns a sample from UART */
int UART_RecvSample(u32 BaseAddress)
{
    int sample = 0;
    int negate = FALSE;
    u8 character = 0;

    while( character != '\r')
    {
        character = XUartLite_RecvByte(BaseAddress);
        if(character == '-')
            negate = TRUE;
        else if( character != '\r')
            sample = sample*10 + (character-'0');
    }
    if(negate)
        sample = sample*(-1);

    return sample;
}

#define N 32768

int main()
{
    int counter = 0;
    int value = 0;
```
short im;
short re;

Xil_ICacheEnable();
Xil_DCacheEnable();

print("---Entering main---\n\r");

/* Read N samples from UART */
while(counter < N)
{
    /* Read one sample from UART */
    value = UART_RecvSample(XPAR_RS232_UART_1_BASEADDR);

    /* Write value to FIR filter */
    putfsl(value,0);

    /* Increase counter */
    counter++;
}

print("All values sent to FFT\r\n");

counter = 0;

/* Read N values from FFT */
while( counter < N)
{
    /* Read value from FFT */
    getfsl(value,1);

    /* Convert 32-bit value to 16-bit real and 16-bit imaginary */
    re = value;
    value = value >> 16;
    im = value;

    /* Write value to STDOUT, in this case UART */
    xil_printf("%d %d\r\n", re, im);

    /* Increase counter */
    counter++;
}
18.1.2 Time measurement

#include <stdio.h>
#include "xparameters.h"
#include "xil_cache.h"
#include "xtmrctr.h"
#include "tmrctr_header.h"
#include "uartlite_header.h"
#include "xuartlite.h"
#include "xbasic_types.h"
#include "fsl.h"

/* Receives and returns a sample from UART */
int UART_RecvSample(u32 BaseAddress)
{
    int sample = 0;
    int negate = FALSE;
    u8 character = 0;

    while( character != '\r')
    {
        character = XUartLite_RecvByte(BaseAddress);
        if(character == '-')
            negate = TRUE;
        else if( character != '\r')
            sample = sample*10 + (character-'0');
    }
    if(negate)
        sample = sample*(-1);

    return sample;
}
#define N 32768

int main()
{
    XTimer counter_time;
    int n_count = 0;
    int counter = 0;
    int value = 0;

    Xil_ICacheEnable();
    Xil_DCacheEnable();

    print("---Entering main---\n\r");

    /* Read N samples from UART */
    while(counter < N)
    {
        /* Read one sample from UART */
        value = UART_RecvSample(XPAR_RS232_UART_1_BASEADDR);
        /* Write value to FIR filter */
        putfsl(value,0);

        /* Increase counter */
        counter++;
    }

    counter = 0;

    print("All values sent to FFT\r\n");

    /* Initialize Timer */
    XTimer_Initialize(&counter_time, XPAR_AXI_TIMER_0_DEVICE_ID);

    /*Start Timer */
    XTimer_Start(&counter_time, 0x0);

    /* Read N values from FFT */
    while(counter < N)
    {
        /* Read value from FFT */
        getfsl(value,0);
/* Increase counter */
counter++;
}

/* Stop Timer */
XTmrCtr_Stop(&counter_time, 0x0);

/* Get number of clock ticks from timer */
n_count = XTmrCtr_GetValue(&counter_time, 0x0);

/* Write to UART the number of clock ticks measured */
xil_printf("Clock ticks = %d\r\n", n_count);

print("FFT Done!\r\n");
print("---Exiting main---\n\r");
Xil_DCacheDisable();
Xil_ICacheDisable();

return 0;
}

18.2 Code for Design 4
#include <stdio.h>
#include <math.h>
#include "xparameters.h"
#include "xil_cache.h"
#include "xil_cache.h"
#include "xtmrctr.h"
#include "xllfifo.h"
#include "xllfifo_hw.h"
#include "xil_printf.h"
#include "xuartps.h"

#define FFT_SIZE 32768
#define CHAR_BIT 8

/* Read bytes (MSB first) from UART and convert to u16. */
* Each byte is interpreted as a number in ASCII representation.
* Bytes are read until '\r' is received */
```c
int UART_RecvValue(XUartPs* InstancePtr)
{
    int value = 0;
    u8 byte = 0;
    u8 negate = 0;
    u8 bytes_returned;

    while(byte != '\r')
    {
        bytes_returned = XUartPs_Recv(InstancePtr, &byte, 1);
        if(byte == '-')
            negate=1;
        else if(byte != '\r' && bytes_returned)
            value = value*10 + (byte-'0');
    }
    if(negate)
        value = value*(-1);

    return value;
}

// bitreverses the integer 'n'
unsigned int bitreverse(unsigned int n, unsigned int bits)
{
    unsigned int nrev, N;
    unsigned int count;
    N = 1<<bits;

    count = bits-1;  // initialize the count variable
    nrev = n;
    for(n>>=1; n; n>>=1)
    {
        nrev <<= 1;
        nrev |= n & 1;
        count--;
    }

    nrev <<= count;
    nrev &= N - 1;

    return nrev;
}
```
```c
// generates a look-up table of bitreversed values
void generate_bitreverse_LUT(unsigned int *bitreverseLUT)
{
    int i;
    int bits;
    bits = 15; //log(FFT_SIZE)/log(2); // log2(FFT_SIZE)
    for(i=0; i<FFT_SIZE; i++)
    {
        bitreverseLUT[i]=bitreverse(i, bits);
    }
}

int main()
{
    int Status, value, result;
    unsigned int results_array[FFT_SIZE];
    unsigned int bitreverseLUT[FFT_SIZE];
    u32 length, i, recvd_values, received_bytes;

    XUartPs Uart_Ps;       /* The instance of the UART Driver */
    XUartPs_Config *Config;
    XLlFifo fifoFIR, fifoFFTdata, fifoFFTconfig;
    XTimerCtr timer_counter;
    int time;
    u32 FFT_SCALING = 0b01101010101011;

    //*************************************************************************
    // Initialize system
    //*************************************************************************/
    Xil_DCACHEEnable();
    Xil_ICACHEEnable();

    // generate a look-up table of bitreversed values
    generate_bitreverse_LUT(bitreverseLUT);

    //*************************************************************************/
    // Initialize Timer */
    XTimerCtr_Inititalize(&timer_counter, XPAR_AXI_TIMER_0_DEVICE_ID);

    // initialize read and write FIFO
    XLlFifo_Inititalize(&fifoFIR, XPAR_AXI_FIFO_MM_S_FIR_BASEADDR);
}
XllFifo_Initialize(&fifoFFTdata, XPAR_AXI_FIFO_MM_S_FFT_DATA_BASEADDR);
XllFifo_Initialize(&fifoFFTconfig,
    XPAR_AXI_FIFO_MM_S_FFT_CONFIG_BASEADDR);

/* configure scaling of the FFT */
XllFifo_Write(&fifoFFTconfig, &FFT_SCALING, sizeof(int));
XllFifo_Write(&fifoFFTconfig, &FFT_SCALING, sizeof(int));
XllFifo_Write(&fifoFFTconfig, &FFT_SCALING, sizeof(int));
XllFifo_TxSetLen(&fifoFFTconfig, sizeof(int)*3);

/*
 * Initialize the UART driver so that it's ready to use
 * Look up the configuration in the config table and then initialize it.
 */
Config = XUartPs_LookupConfig(XPAR_PS7_UART_1_DEVICE_ID);
if (NULL == Config) {
    return XST_FAILURE;
}

Status = XUartPs_CfgInitialize(&Uart_Ps, Config, Config->BaseAddress);
if (Status != XST_SUCCESS) {
    return XST_FAILURE;
}

XUartPs_SetBaudRate(&Uart_Ps, 115200);

// garbage collection (receive anything left in the UART receive buffer)
received_bytes = 1;
while(received_bytes)
{
    received_bytes = XUartPs_Recv(&Uart_Ps, &value, 1);
}

xil_printf("---Entering main---\r\n");
recvd_values = 0;

/*=============================================================================*/
*   Receive input from UART and send N samples to FIR.
* The result from the FIR is passed to the FFT.
* Input -> FIR -> FFT
/*=============================================================================*/
for(i=0; i<FFT_SIZE; i++)
{
    /* send values */
    value = UART_RecvValue(&Uart_Ps);  // get value from UART
    XLlFifo_Write(&fifoFIR, &value, sizeof(int));
    XLlFifo_TxSetLen(&fifoFIR, sizeof(int));

    /* check if there are values in RX fifo and receive them */
    if( XLlFifo_RxOccupancy(&fifoFIR) ) // if there are frames in recv FIFO
    {
        // get length (bytes) of the frame
        length = XLlFifo_RxGetLen(&fifoFIR);
        while(length>=4)
        {
            XLlFifo_Read(&fifoFIR, &result, 4);  // read frame
            XLlFifo_Write(&fifoFFTdata, &result, sizeof(int));
            XLlFifo_TxSetLen(&fifoFFTdata, sizeof(int));
            recvd_values++;
            length-=4;
        }
        if(length)
        {
            xil_printf("Error: length mismatch\r\n");
        }
    }
}

/* Handle rest. Make sure that all N samples are received from the FIR and */
/* passed to FFT */
while(recvd_values < FFT_SIZE)
{
    while(!XLlFifo_RxOccupancy(&fifoFIR));  // wait for values in fifo
    // get length (bytes) of the frame
    length = XLlFifo_RxGetLen(&fifoFIR);
    while(length>=4)
    {
        XLlFifo_Read(&fifoFIR, &result, 4);  // read frame
        XLlFifo_Write(&fifoFFTdata, &result, sizeof(int));
        XLlFifo_TxSetLen(&fifoFFTdata, sizeof(int));
        recvd_values++;
        length-=4;
    }
}
if(length)
    xil_printf("Error: length mismatch\n");
}
xil_printf("Fir Phase done (read %d values)\n", recvd_values);

/*Start Timer */
XTmrCtr_Start(&timer_counter, 0x0);

/*************************************************************************/
* Receive the result from the FFT
*************************************************************************/
recvd_values = 0;
while(recvd_values<FFT_SIZE)
{
    // wait for frames in recv FIFO
    while( !XLlFifo_RxOccupancy(&fifoFFTdata) );
    // get length (bytes) of the frame
    length = XLlFifo_RxGetLen(&fifoFFTdata);

    while(length>=4)
    {
        XLlFifo_Read(&fifoFFTdata, &result, 4); // read frame

        results_array[bitreverseLUT[recvd_values++]] = result;
        //results_array[recvd_values++] = result; // without bitreverse
        length-=4;
    }
    if(length)
        xil_printf("Error: length mismatch\n");
}
/* Stop Timer */
XTmrCtr_Stop(&timer_counter, 0x0);
time = XTmrCtr_GetValue(&timer_counter, 0x0);
xil_printf("FFT Phase done (read %d values)\n", recvd_values);

// print result
xil_printf("real\timag\n");
for(i=0; i<FFT_SIZE; i++)
{
    xil_printf("%d\t%d\n", (short)(results_array[i] & 0x0000ffff),
               (short)(results_array[i]>>16));
}
xil_printf("%d ticks @ 100MHz\n\r", time);
xil_printf("---Exiting main---\n\r");
Xil_DCacheDisable();
Xil_ICacheDisable();

    return 0;
}