Christoph Ehrenhöfer BSc

Design and Implementation of an FPGA-based Time-of-Flight Processing System

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Telematics

submitted to

Graz University of Technology

Supervisor

Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger

Institute for Technical Informatics

Advisor: Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger
Dipl.-Ing. Dr.techn. Norbert Druml (Infineon Technologies Austria AG)

Graz, May 2016

Affidavit

Kurzfassung

Der Einsatz von dreidimensionalen Bildverarbeitungstechnologien in Unterhaltungs- und Industrieelektronik wurde in den letzten Jahren sehr populär, wobei die Time-of-Flight-Technologie insbesondere das Interesse auf sich gezogen hat. Aufgrund der kompakten Baugröße von Time-of-Flight-Kameras, können solche Systeme leicht in verschiedene eingebettete Systeme, z.B. Smartphones, integriert werden. Daher können die Anforderungen an implementierte Anwendungsfälle sehr unterschiedlich sein. Deshalb ist es nicht so einfach ein Time-of-Flight-System zu entwerfen, welches einen guten Kompromiss zwischen Hardwarebeschleunigung und Flexibilität erzielt.

Diese Masterarbeit stellt eine neuartige Plattform zur Verarbeitung von Time-of-Flight-Daten auf einem flexiblen und schnellen Hardware-/Software-System dar. Das wird unter Verwendung der Xilinx Zynq-Plattform erreicht, die bereits erfolgreich für die zweidimensionale Bildverarbeitung in verschiedenen Anwendungsfällen, z.B. Falldetektion, eingesetzt wurde. Der Zynq-System-on-Chip ermöglicht die hardwarebeschleunigte Berechnung von Time-of-Flight-Daten auf einem FPGA. Zusätzlich können die hardwareintegrierten Komponenten in Software, auf einem ARM-Prozessor, gesteuert werden. Dieses leistungsstarke Hardware-/Software-System bietet hohe Flexibilität und eine wesentliche Beschleunigung in Hardware.

Darüber hinaus wird die Durchführbarkeit des vorgestellten Systems demonstriert. Das komplette System besteht aus der Automobilplattform AURIX von Infineon Technologies, einer Time-of-Flight-Kamera, entwickelt von Infineon Technologies in Kooperation mit PMDTechnologies, und einem Zynq-Entwicklungsboard. Zwei Time-of-Flight-Algorithmen für die Vorverarbeitung der Tiefeninformation für typische Anwendungsfälle (z.B. Gestenerkennung, Indoor-Navigation) werden implementiert. Diese Tiefen- und 3D-Daten werden dann dem AURIX zur Verfügung gestellt. Die erhaltenen Ergebnisse zeigen, dass nahezu 100 FPS bei einem durchschnittlichen Rechenfehler von 0.08 mm erreicht werden können.

Außerdem wird ein praktischer Ansatz von Rapid-Prototyping für Algorithmen vorgestellt. High-Level-Synthesis wird für die Erstellung von Hardwarekomponenten für zwei Testfälle verwendet, welche dann mit dem FPGA-basierten Time-of-Flight-Koprozessor verglichen werden. Die Ergebnisse zeigen, dass die implementierten Hardwaremodule die gleiche Größenordnung in Bezug auf Leistung und Größe haben. Dies zeigt, dass die entwickelte Time-of-Flight-Plattform hinsichtlich verschiedener Anwendungsfälle und deren Anforderungen sehr flexibel ist.

Abstract

Three-dimensional imaging technologies have become very popular during the last years in consumer and industrial electronics. The Time-of-Flight (ToF) technology has especially attracted lots of interest. Due to the small form factor of Time-of-Flight cameras, such systems can be easily integrated in a variety of different embedded devices, for example, in smart phones. The requirements of implemented use-case applications can therefore differ widely. Thus, it is not easy to create a Time-of-Flight framework that can achieve a good trade-off between hardware-acceleration and flexibility.

This thesis presents a novel platform to process Time-of-Flight data on a flexible and fast hardware/software system. This is accomplished by using the Xilinx Zynq platform, which is already successfully used in evaluating two-dimensional image processing in various kinds of applications, for example, fall detection. The Zynq System-on-Chip (SoC) allows the hardware-accelerated computation of Time-of-Flight data on an FPGA. In addition, the hardware-integrated components are controlled in software on an ARM CPU. This powerful hardware/software system provides high flexibility while achieving an essential speed-up in hardware.

The work also demonstrates the feasibility of the proposed system. The complete system consists of the automotive platform AURIX from Infineon Technologies and a Time-Flight camera system, developed from Infineon Technologies in cooperation with PMDTechnologies, and the Zynq development board. Two Time-of-Flight pre-processing algorithms for typical use-case applications (e.g., gesture recognition, indoor navigation) are implemented to provide distance and 3D data to the AURIX. The results show that almost 100 FPS are possible with an average calculation error of 0.08 mm.

Furthermore, a practical approach of rapid algorithm prototyping is introduced. High Level Synthesis is used to create hardware components for two test cases, which are compared to the FPGA-based Time-of-Flight co-processor. The results show that the implemented hardware modules have the same dimension regarding performance and utilization. This shows that the developed Time-of-Flight processing platform is highly flexible regarding different use-case applications and requirements.

Acknowledgments

This master's thesis was carried out at the Institute for Technical Informatics at the Technical University Graz. The practical part of this work was developed at Infineon Technologies in Graz. First of all, I want to express my thanks to all people who were involved in the creation process of this thesis.

I would like to appreciate the efforts and assistance of my thesis advisor Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger. With his participation and valuable comments, the quality of this work was considerably improved. In addition, it gives me great pleasure in acknowledging the support of my advisor Dipl.-Ing. Dr. techn. Norbert Druml at Infineon Technologies Austria AG. He attracted me to the interesting topic of this thesis and steered me in the right direction to successfully complete my work.

I also would like to acknowledge my fellow students throughout my years of study in Graz. In particular I consider it an honor to work with Josef and Lukas on a vast number of projects during my bachelor and master studies. Furthermore, I would like to underscore the harmonic and effective teamwork before deadlines and exams with them.

Finally, I would like to extend my thanks to my whole family, especially my parents, for their emotional and financial support during my academic studies. Without their unfailing help, the accomplishment of my master's degree would have been significantly more difficult. Thank you.

Graz, May 2016 Christoph Ehrenhöfer

Contents

List of Abbreviations

ARM	Advanced RISC Machine
AXI	Advanced eXtensible Interface
BRAM	Block Random Access Memory
CPU	Central Processing Unit
DDR	Double Data Rate
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processor
FF	Flip-Flop
FPGA	Field Programmable Gate Array
FPN	Fixed Pattern Noise
FPPN	Fixed Pattern Phase Noise
FPS	Frames per second
HLS	High Level Synthesis
I2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IO	Input/Output
IP	Intellectual Property
LUT	Look-up Table
PCB	Printed Circuit Board
PIF	Parallel Interface
PMD	Photonic Mixer Device
PL	Programmable Logic
PS	Processing System
SDK	Software Development Kit
SDRAM	Synchronous Dynamic Random Access Memory
SoC	System-on-Chip
ToF	Time-of-Flight
TPG	Test Pattern Generator
UDP	User Datagram Protocol
URE	Unambiguous Range Extension
VDMA	Video Direct Memory Access
VTC	Video Timing Controller

List of Figures

List of Tables

Chapter 1

Introduction

In application domains such as consumer electronics, machine vision and automotive technology, three-dimensional imaging has grown in popularity in the last years due to improvements in such systems. In a 3D image, each pixel corresponds to the range of scenery. Thus, scanning the environment with an additional dimension in comparison to 2D is possible.

The Time-of-Flight range imaging technology has especially raised the interest. The principle is 3D imaging technology that measures the travel time of emitted light and provides distance information, by sending a laser pulse from an emitter. The reflection of the signal at the object is then detected and processed to a range image. The time consumed in-between corresponds to the distance. One implemented approach of this concept is shown in Figure 1.1. With photonic mixer devices (PMD), the phase difference of the transmitted and received modulated optical signal is measured. This phase shift is converted into a voltage that is proportional to the distance. After post-processing, typically done in software, the results are a depth image and an amplitude image.

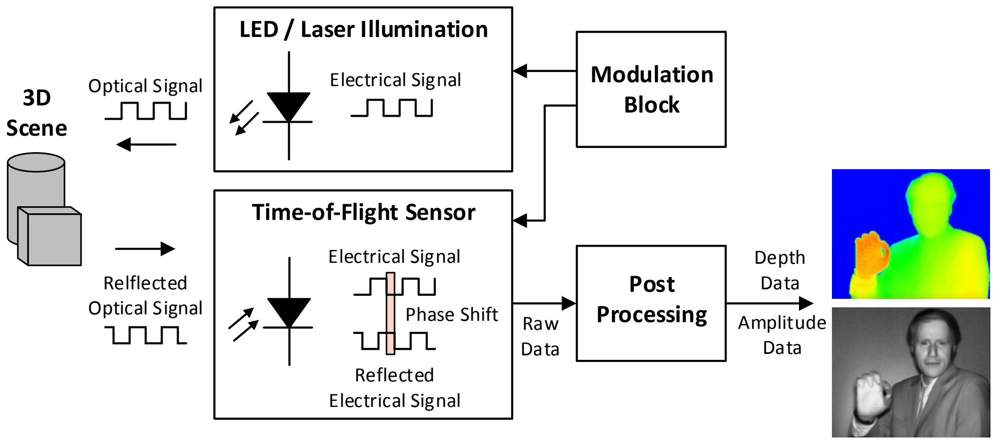


Figure 1.1: Basic Time-of-Flight working principle [DFH+15] (with changes).

ToF-based cameras can gather a whole scene with one shot. Thus, such systems can achieve very high frame rates up to 100 FPS, which is perfect for real-time applications. Compared to other 3D imaging technologies, no moving parts are necessary, allowing cost-efficiency cameras with very small form factors to be built. The resulting distance images can be efficiently computed by a post-processing algorithm. Therefore, such robust systems are perfectly suitable to be integrated into consumer electronic or safety-critical applications.

With the growing distribution of this technology in lots of domains, various kinds of different use-cases occur. The requirements of applications vary a lot regarding the use-case. For instance, for gesture recognition high frame rates and a short measurement range is required, whereas for indoor navigation a long measurement range is needed and lower frame rates are sufficient. By keeping these characteristics in mind, it is difficult to design a cost-efficient ToF-based framework for completely different and yet, unknown use-case applications. In addition, the processing algorithm has to be fast, accurate and flexible.

1.1 Motivation

Today Time-of-Flight cameras can be embedded in all sorts of smart devices such as, smart phones and tablets, based on the small form factor of indirect ToF cameras, and the compact design of integrated circuits in the semiconductor industry. Due to the large range of devices, a variety of use-case applications with particular requirements, such as gesture recognition and indoor navigation, are needed. Parameters include the image resolution, frame rate, range and computation specific details. Therefore, commercial ToF solutions are normally focused on one specific use-case only.

Furthermore, some use-cases, such as gesture recognition, require high frame rates of around 40 frames per second (FPS). For a depth image, several raw images provided by the ToF camera are required. It is challenging to deliver a high number of images in a short time if the processing system performs time-consuming computation in software. In addition, the interface needs to transmit large loads of data, which also takes up time and resources.

This raises interest in showing the feasibility of a hardware/software platform that tackles these issues. This work will focus on a hardware-accelerated and flexible ToF co-processing framework. This system should efficiently implement a basic ToF post-processing flow on a Field-Programmable Gate Array (FPGA). Additionally, it will feature a flexible realization for different use-case applications. A demonstrator should be developed using the FPGA-based system as an interface between a ToF camera and an automotive processing platform.

1.2 Objectives

The goal of this thesis is to design and implement a hardware-accelerated ToF imaging processing system, which is highly flexible in terms of different use-case applications. The hardware/software platform enables the possibility to have a framework which is highly customizable regarding unknown applications in future. Two different use-cases should be efficiently implemented on an FPGA and evaluated regarding their performance. To reach this goal, the structure of the work represents the process of the performed activities:

- Research of basic principles and related state-of-the-art publications
- Design of a flexible hardware-accelerated ToF imaging framework
- Implementation of the FPGA-based ToF processing system
- Realization of the concept in terms of a demonstrator
- Evaluation of the implementation results regarding different performance parameters

1.3 Outline

The work is organized as follows: in Chapter 2, an overview of the literature research in the areas of Time-of-Flight and hardware-accelerated 2D and 3D image processing systems is given. The important topics of this work are discussed and the state-of-the-art work regarding related solutions is presented. In addition, the current system and its limitations are described. Then in Chapter 3, the requirements of the new system are summarized. Furthermore, the design of the FPGA-based Time-of-Flight processing system is shown in detail. This is followed by Chapter 4, which presents the implementation of the hardware/software system. Details about the development environment and the workflow are explained. Chapter 5 contains the results of the implementation regarding area, performance and other parameters of the processing system. Further on, details about the final demonstrator of the implemented system are shown. Finally, in Chapter 6, the work is concluded and an outlook of possible future prospects is given.

Chapter 2

Related Work

The goal of this chapter is to provide a good overview of the literature research. At first, the basic Time-of-Flight principle and its applications, as well as the common systematic errors are outlined. Next, state-of-the-art projects regarding related solutions are represented. An outline therefore about current hardware-accelerated 2D and 3D image processing systems is provided. Finally, the publication of the existing framework is described in detail.

2.1 Time-of-Flight Principle

By knowing the precise value of the speed of light, it is possible to determine distances. This is calculated by measuring the time that light travels from a transmitter to a target and back to a receiver [Lan00].

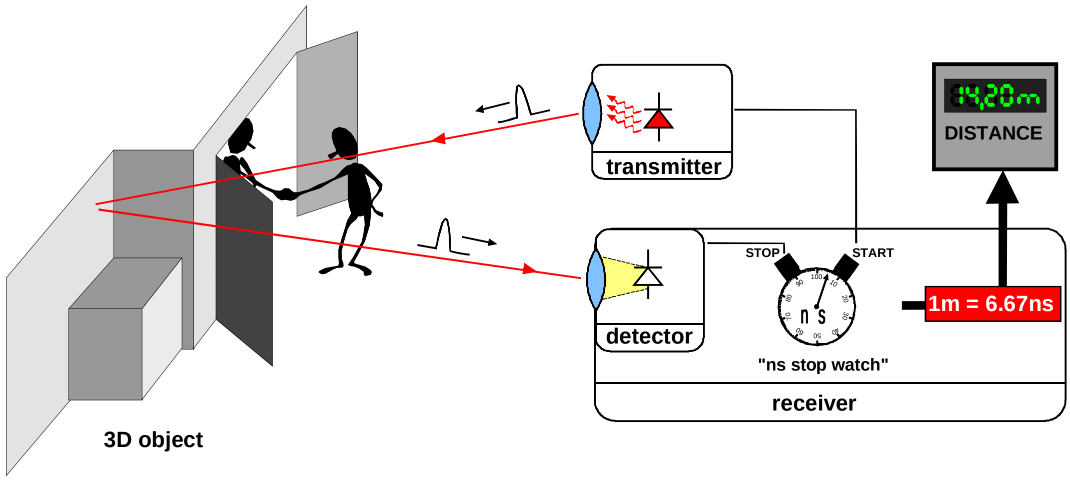


Figure 2.1: Basic principle of a Time-of-Flight ranging system [Lan00].

The basic principle of a Time-of-Flight ranging system, as outlined in [Lan00], is depicted in Figure 2.1. A light pulse is emitted by an active light source and the measurement system is simultaneously initiated. The light pulse is then reflected back by the target and received by a detector, which stops the measurement system. By considering that the light travels to and from the reference point, a distance of 1 m corresponds to a measured time of 6.67 ns. This relation is illustrated in Equation 2.1 where τ is the echo time and c represents the speed of light. In order to avoid shadowing effects due to different camera angles, the emitter and detector are closely located. Another important property of the system is the synchronous operation of the active light source and the detector.

$$d = \frac{c \cdot \tau}{2} \tag{2.1}$$

2.1.1 Modulation Signals

In order to implement this approach, different modulation types of the emitted signal exist, as described in [Lan00]. In general, pulsed modulation or continuous wave (CW) modulation is used. The most obvious technique is the pulsed light operation. By directly measuring the turn-round time of an emitted light pulse, the distance can be determined. The basic problem of such a system is the high accuracy that is required for the measurement mechanism.

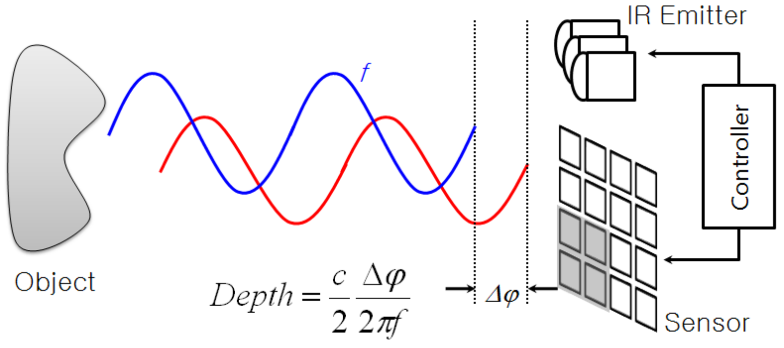


Figure 2.2: Indirect Time-of-Flight principle [HLCH12].

With the use of continuous wave modulation [Lan00], it is possible to indirectly measure the echo time by determining the phase difference between the modulated emitted and detected signal, as depicted in Figure 2.2. For such systems, lots of different light sources are available and the shape of the signal can be chosen, for example, sine wave, square wave. With the knowledge of the modulation frequency f_M and the measured phase φ, the echo time τ can be calculated, as shown in Equation 2.2. With the help of this formula, the final distance d can then be computed, as represented in Equation 2.3. Because of the periodicity of the measured phase, the unambiguous range is restricted to $c/(2f_M)$.

$$\tau = \frac{\varphi}{2\pi f_M} \tag{2.2}$$

$$d = \frac{c \cdot \varphi}{4\pi f_M} \tag{2.3}$$

2.1.2 Other 3D Measurement Systems

A hierarchical characterization of contactless 3D measurement techniques is given in Figure 2.3, as summarized from the author in [Lan00]. Such systems use microwave, light wave or ultrasonic wave techniques to measure the range to a target. Microwave- and ultrasonic wave-based systems are typically used in GPS or radar use-case applications. However, these two methods are not further discussed because they are not suitable for 3D measurements with high angular resolution, due to diffraction limitations. In this work, the focus is to provide an overview of optical range measurements techniques with a wave length between 0.5 - 1 μm. The most important optical 3D measurements techniques are triangulation, interferometry and Time-of-Flight.

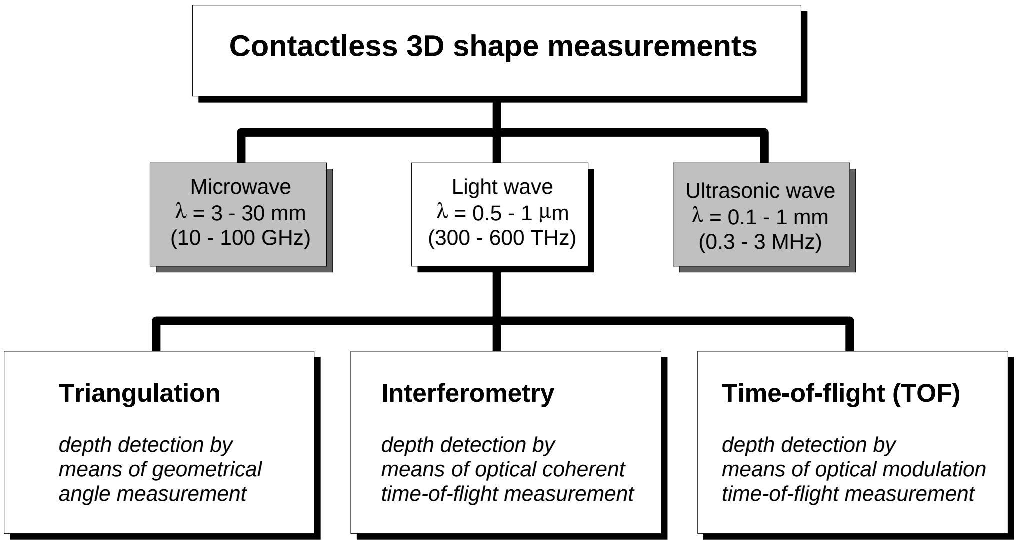

Contactless 3D shape measurements
Microwave
λ = 3 - 30 mm
(10 - 100 GHz)
Light wave
λ = 0.5 - 1 µm
(300 - 600 THz)
Ultrasonic wave
λ = 0.1 - 1 mm
(0.3 - 3 MHz)
Triangulation
depth detection by means of geometrical angle measurement
Interferometry
depth detection by means of optical coherent time-of-flight measurement
Time-of-flight (TOF)
depth detection by means of optical modulation time-of-flight measurement

Figure 2.3: Contactless 3D measurement systems [SHB+99].

Triangulation

The triangulation measurement method, as outlined in [Lan00], is based on geometrical analysis of a triangle. The distance of a reference point is one unknown point of the triangle, whereas the other two known points belong to the system. By measuring the angles or the base line of the triangle, the unknown distance can be determined. Two different techniques exist: passive and active triangulation.

For passive triangulation [Lan00], two cameras are placed apart with a known distance and observe the same point. By measuring the viewing angles, the distance can then be determined. One common realization of this principle is **stereo vision**. 2D-correlation is used to find typical object features in both images. Due to this identification of the same point, high contrast images are necessary. This method works quite well for rich contrast scenes that need no active light source. The disadvantages however are the shadowing effect, the high computation effort and the system's size.

In the active triangulation approach [Lan00], a positive sensitive detector observers a point on the scene projected from a light source. There is no need of measuring the viewing angles. Instead, the distance can be determined by knowing the focal length of the system and the point projection on the detector. In order to measure the whole scene, the laser point needs to scan the whole object surface. Using **laser scanning** on a complete line provides a faster measurement. Drawbacks are the time-consuming scanning and the need of mechanical parts in the system. An advanced technique that does not require moving parts, is based on two-dimensional **structured light**. This system projects a 2D striped pattern on the whole surface, a 2D camera then captures the scene. The range can then be determined by analyzing the displacement of the stripes.

Interferometry

This technique [Lua01] is based on the superposition of two monochromatic waves with the same frequency. A beam splitter is used to split a laser beam. One beam is projected to a mirror and the other to the target. An integrated detector captures both rays as they are reflected back. After integration, an interferogram provides phase information, related to the distance. Interferometry is suited for applications where high accuracy over small distances is required. The main disadvantages are that only relative distances can be measured and that the unambiguous range is very low.

Time-of-Flight

The basic principle of Time-of-Flight has already been described in the beginning of this chapter. An advantage of indirect ToF cameras is the small form factor because no moving parts are used and the sensor can be built in CMOS technology [Lan00]. Therefore, the camera system is robust and inexpensive. High frame rates are possible because information for all pixels is gathered in parallel whereby the phase information can be determined without much computation effort. The main drawbacks are the low resolution, the maximum unambiguous range and measurement errors, which need to be compensated by calibration.

2.1.3 Photonic Mixer Device

This work will focus on Time-of-Flight imaging based on continuous wave modulation. The main feature of this method is the realization of the sensor with the photonic mixer device (PMD) technology, which implements the process of mixing and correlation of the detected optical signal and the reference signal in one semiconductor circuit, as shown in [Lua01]. The correlation between the received signal and the original signal is known as cross-correlation. Such an independent PMD pixel can be easily integrated into a matrix in 3D, which provides parallel determination of the phase.

The schematic structure of a PMD pixel, as depicted in Figure 2.4, is described in detail in [Alb07]. Because of the inner photoelectric effect, incoming photons in the poly-silicon-layer are split-up into electron-hole pairs. The electrons are then separated into bucket A or B. This depends on the modulation of the reference signal, which generates an electrical field. After a certain integration time, the buckets are read out. The difference between the voltages of both buckets corresponds to the overlapping of the emitted and detected signal. With a longer integration time, the signal-to-noise ratio can be improved. An issue of this approach is that one measurement cannot unambiguously determine the phase delay. Hence, a further measurement is performed with the phase-shifted emitted signal. With the results of these two measurements, the phase delay can then be unambiguously calculated. Therefore, at least two illuminations are necessary to determine the phase delay.

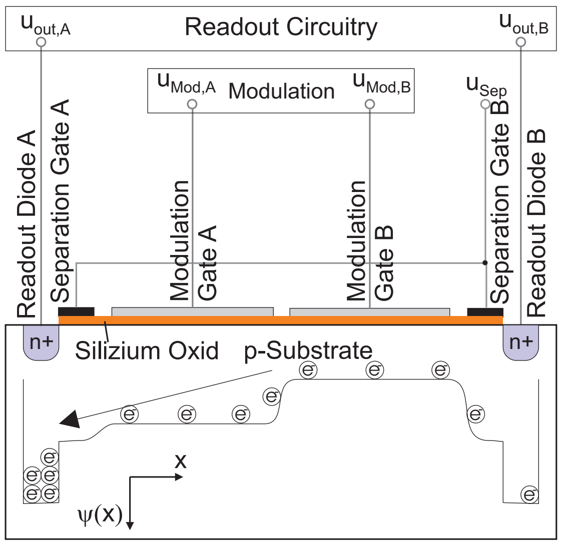


Figure 2.4: Schematic structure of a PMD pixel [Alb07].

2.1.4 4-Phases Measurement

The indirect TOF measurement method using the PMD technology is within the focus of this thesis. Therefore, at least two measurements with different phase-shifts are necessary, as mentioned in the previous paragraph. Most of the state-of-the-art solutions, which use the same method and technology, perform four measurements with different shifted phases, as presented in [LS01]. Advantages of this approach include a simplified formula in calculating the final phase delay and the reduced impact of some systematic errors. In general, an equivalent phase-shift of 90° is used, which results in measurements with the following phase-shifts: 0°, 90°, 180° and 270°.

An example measurement [HLCH12] is depicted in Figure 2.5. The amount of electric charge for the measurements C_1 to C_4 is respectively represented with the quantities Q_1 to Q_4. The PMD pixel efficiency determines the overlapping in one semiconductor based circuit.

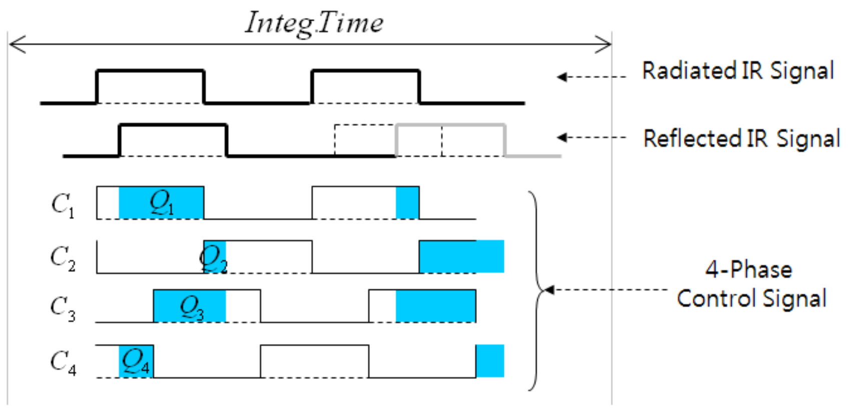


Figure 2.5: Example of a 4-Phases measurement [HLCH12].

The phase and the amplitude can be unambiguously computed [HLCH12]. The phase φ can be calculated with the arcus tangent function, as illustrated in Equation 2.4. The values from the four measurements (Q_{0°, Q_{90°, Q_{180°, Q_{270°) provided from the PMD pixel array, are used to calculate two intermediate differences, which are divided. With the same differences the amplitude A can be computed, as shown in Equation 2.5. The amplitude, which represents the strength of the received signal [FAT11], is used for prediction of the measurement quality.

$$\varphi = \arctan\left(\frac{Q_{270^\circ} - Q_{90^\circ}}{Q_{0^\circ} - Q_{180^\circ}}\right) \tag{2.4}$$

$$A = \frac{\sqrt{(Q_{270^\circ} - Q_{90^\circ})^2 + (Q_{0^\circ} - Q_{180^\circ})^2}}{2} \tag{2.5}$$

2.1.5 Unambiguous Range Extension

As already mentioned, the measured phase exhibits a $2pi$-periodicity. Hence, the distance is restricted to an unambiguous range of $c/(2f_M)$. In this section, a method is presented to extend the unambiguous range. The most obvious approach is to decrease the modulation frequency [JCPD10]. The drawback of this technique is that the measurement precision is also decreased. Another method is presented in [DCP+07]. By performing two measurements with different modulation frequencies of the scene, it is possible to extend the unambiguous range, as shown in Equation 2.6. The precision of the measurement does not significantly worsened.

$$d_u = \frac{c}{2\left|f_{M1} - f_{M2}\right|} \tag{2.6}$$

The basic idea behind dual frequency modulation is depicted in Figure 2.6. The result of each measurement is a possible set of object locations. The true location of the object is where both measurements are most in agreement. Algorithms that perform this selection step using two modulation frequencies are presented in [JCPD10] and [JBP+10].

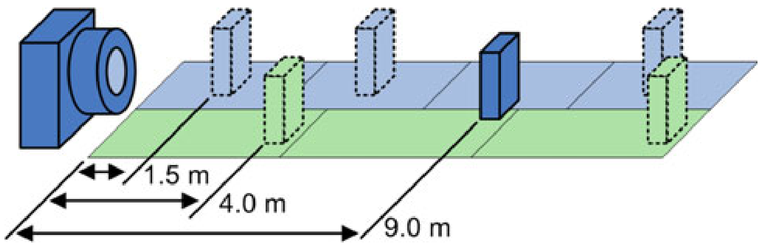


Figure 2.6: Example of a measurement combining two modulation frequencies [JBP+10]. The correct location of the object is at 9 m.

2.1.6 Systematic Measurement Errors and Compensation

Like almost all sensing devices, Time-of-Flight cameras also show different measurement error sources. Lots of research is done in this field to identify and minimize the common errors. A good overview of literature regarding systematic errors is given in [FAT11]. In general, the ToF measurement exhibits systematic, which are compensated using calibration, and non-systematic errors, which are reduced using filters. This section will focus on the main system-errors and their countermeasures.

Wiggling Error

The wiggling error, also referred to as circular distance error, is based on imperfect sinusoidal waves used as emitted and reference signal, as shown in [FAT11]. It is difficult to generate in practice a plain sinusoidal signal. Therefore, an offset, which is distance-depended, is added to every pixel. The shape of the error, which can look like a sine wave, is depicted in Figure 2.7.

The error can be compensated, by measuring the distance with the camera to a known reference point [FAT11]. This is repeated for every distance value within the unambiguous range. Measured and reference distances are then compared. The result is an offset for every distance value over the complete unambiguous range. This method comes with the drawback that a high accuracy sensor is needed to determine the reference distance, e.g., track line, color camera. Another approach is to generate a model of the error by evaluating multiple relative measurements, but has the disadvantage that the processed compensation values are only suited for a limited range.

With the processed offsets, calibration data is generated to compensate the wiggling error [FAT11]. Several solutions exist to encode the data. It can be stored in a look-up table (LUT), which represents the offset, depending on the distance. It is also possible, to express the depth error by a mathematical function. A B-spline or a polynomial function higher degree can be used to store the values in a compact form.

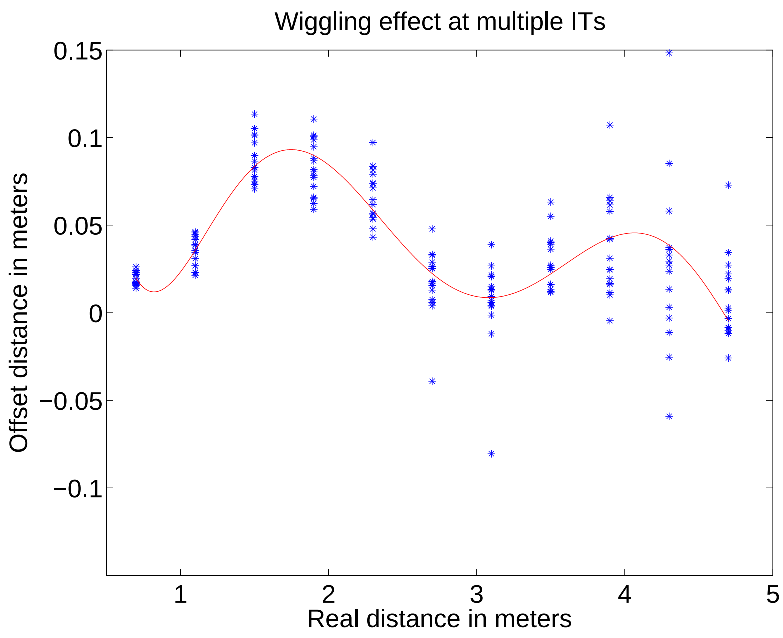


Figure 2.7: Wiggling effect, approximated with a 6-degree polynomial, at multiple integration times [FAT11].

Pixel-Related Error

Pixel-related errors vary for every pixel and are independent of the modulation frequency, as outlined in [FAT11]. One error source is diversity of produced pixels regarding manufacturing tolerances and different material characteristics. Neighboring pixels determine different results for the same distance. Another reason for occurrence of this error is the location of the pixel on the sensor. Because of different signal path lengths, a latency-related offset is generated.

Literature distinguishes between following pixel-related errors: **fixed-pattern noise** (FPN) and **fixed-pattern phase noise** (FPPN). In [Alb07], pixel-characteristic-related errors are referred to as FPN, whereas FPPN is known as pixel-location-related errors. In general, the calibration is performed by measuring the distance to a reference plain, and determining the offsets between both results, as depicted in Figure 2.8. The compensation results are then saved in a LUT. In [LSKK10], an approach to compensate FPN is described by averaging a high amount of images that are taken with shut optics (black images).

Temperature-Related Error

Like almost all electronic devises, the temperature of the ToF camera has an impact on the measurement, as illustrated in [FAT11]. The measured distances of the whole image drift with the temperature. Internal and external temperature changes can cause this error because the sensor shows a temperature-dependent behavior. Until the working temperature is reached, the internal temperature rises and causes an offset error, as depicted in Figure 2.9.

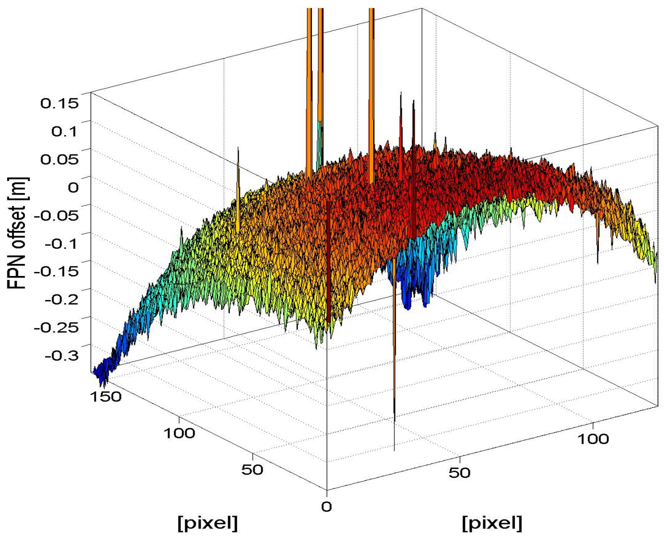


Figure 2.8: Fixed pattern noise offset per pixel [KIR06].

The calibration, as described in [PMD13b], can be performed by measuring the distance to a known reference point, as well as the temperature of the camera while changing the internal/external temperature over a certain range. By comparing the offset error and the temperature, a thermal correction coefficient can be determined.

If a temperature sensor is directly positioned in the camera, it is possible to compensate the error with the processed calibration data [PMD13b]. If this not the case, the typically strategy is to wait until the camera reaches working temperature. After that, the compensation coefficient for the typical temperature occurring in the intended use-case is used.

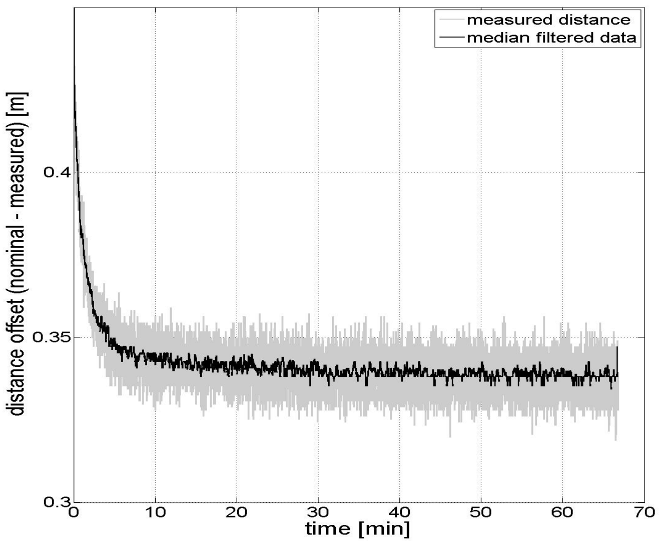


Figure 2.9: Temperature-related error over an observed time [KIR06].

2.2 State-of-the-Art

The goal of this section is to provide an overview of similar solutions. First of all, FPGA-based Time-of-Flight imaging systems are presented. Hence the architecture and the processing algorithm of the implementations are described. Finally, the image processing projects for different use-case applications, which use the Xilinx Zynq platform, are shown.

2.2.1 FPGA-based Time-of-Flight Processing

In the last years 3D imaging raised the interest in several domains, some examples include automotive technology and consumer electronics. Real-time processing is required for lots of use-case applications. For instance, a typical gesture recognition use-case requires around 40 FPS to work properly. Hence, the following paragraphs list the projects that investigate this challenge with the help of an FPGA.

FPGA-based Time-of-Flight Characterization System

The authors of [SHDZ13] presented an FPGA-based characterization system for a ToF sensor. The proposed framework allows the flexible generation of all control signals that are needed for a ToF distance measurement, for example, the shape of the control signal. After a measurement the captured data is pre-processed on an FPGA and sent to the host PC, which decreases the amount of data that needs to be transferred over the USB interface.

A simplified block diagram is depicted in Figure 2.10. A new measurement starts by activating the FPGA (i.e., Altera Stratix IV) over the USB interface of the host PC. The control signals are generated from the FPGA and sent to the illumination unit, as well as the 3D sensor. A laser is used as an illumination source because of the large signal bandwidth. Therefore, the illuminated light is coupled into the ToF chip through an optical fiber to remove possible error sources. The used pixels of the ToF chip, which are able to suppress very high intensities of background light, are described in [DSH$^+$13]. The camera supports an image resolution of up to 128x128 pixels. After successful measurement the output voltages generated from the ToF chip are read from the FPGA via an ADC. The device performs the storage of a full correlation triangle into the SRAM. With a memory controller, the data can be read out to the USB interface and forwarded to the host PC. In addition, all control signals are generated in a signal generation block on the FPGA.

The author is highlighting the high flexibility of the system as compared to other publications, no commercial products are used in this work. The results show that the standard deviation of the measured distance is 0.15 mm. The FPGA can average a maximum of 256 correlation triangles to improve measurement precision. By performing the pre-processing on the FPGA, the integration time can be virtually increased. In addition, the load of data, which is transferred over the USB interface, is reduced. Hence, a highly flexible platform to characterize error sources was introduced.

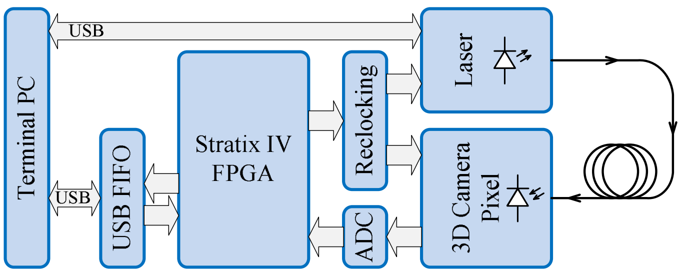


Figure 2.10: Block diagram of the measurement setup [SHDZ13].

Easily Configurable Range Imaging System

Jongenelen et al. [JCP+09] developed a range imaging system that is easily configurable by means of modulation patterns. The platform is suited for further experiments regarding ToF-based applications. Commercial products are limited to a choice of fixed configurable parameters, whereas the proposed system can be more flexibly configured from the FPGA.

The system architecture of the hardware design, as shown in Figure 2.11, comprises several components. A ToF sensor from PMDTechnologies is connected via a mainboard to the Altera Stratix III FPGA. An illumination board provides the modulated light. The connection between the FPGA and the host PC is established with a VGA/Ethernet board. The FPGA drives the modulation signals and generates control signals for the PMD sensor. In addition, the device temporally stores the gathered raw data and computes distance images, which are then forward to a VGA monitor and a host PC. Over a JTAG interface, the PC can configure the operating parameters. The computation of the range images as well as the PMD and VGA interface are implemented in VHDL, whereas the Ethernet and JTAG interface are programmed in C on the Nios II processor.

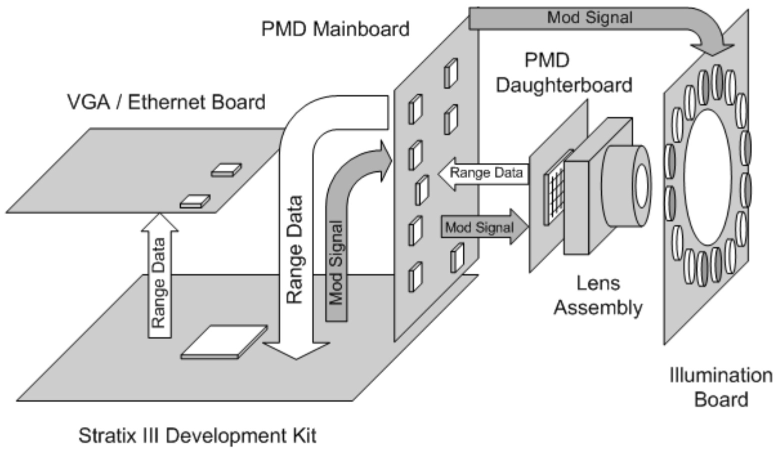


Figure 2.11: Interaction between the different sub-components. Obtained from [CCJD11].

The experimental results showed the relation between modulation frequency and standard deviation of the phase and distance. Therefore, several measurements were obtained using different configured modulation frequencies. It was demonstrated that at a modulation frequency of 36 MHz, the best precision of the distance measurements, which is approximately 3.5 mm, can be achieved. The distance precision is related to the phase precision, which becomes worse at higher modulation frequencies due to the bandwidth limitation of electronics. The conclusion highlights the aspect of the ease of configuration of several operating parameters.

Image Ranger System for Mobile Robotic Platforms

In [CCJD11], the authors demonstrated an FPGA-based range imaging system that can be mounted on mobile robotic platforms. An unambiguous region of up to 15 m is required. The proposed system in [JCP+09], as mentioned in the previous paragraph, was redesigned to achieve a small form factor. The Stratic III FPGA from Altera was therefore replaced by the Altera's Cyclone III FPGA.

The basic architecture of the system is illustrated in Figure 2.12. The sub-components are designed to allow stacking on top of each other. This stacked setup provides a small form factor and upgradability of each unit. The image sensor from PMDTechnolgies allows a resolution of up to 160 x 120 pixels. The FPGA implements a typical 4-Phases measurement algorithm.

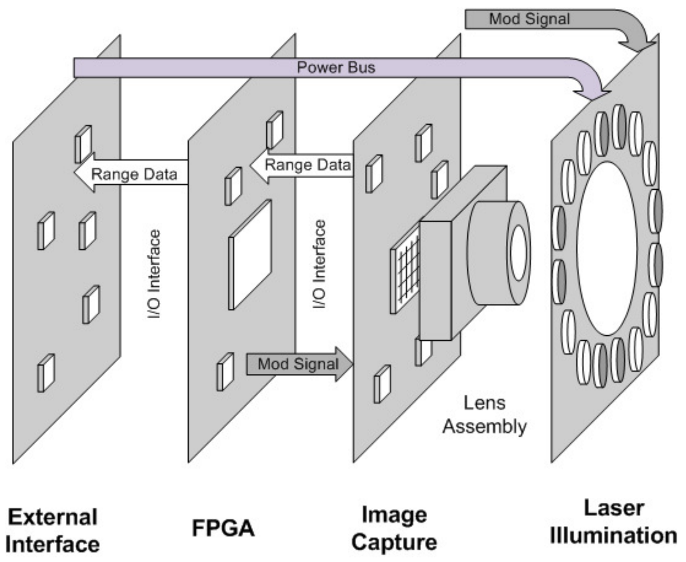


Figure 2.12: System architecture of mobile image ranger system [CCJD11].

The results showed that the imaging system has the compact size of 120 x 200 x 120 mm, which corresponds to approximately 30% of the initial setup. In order to evaluate the imaging quality, a video was captured with the following configuration: 26 MHz modulation frequency, 16.7 FPS and 20 ms integration time per frame. The reference points were evaluated in the captured video sequence. Objects located in the mid-range of the image have a standard deviation of 3.8 mm, whereas objects placed further away have a standard deviation of 32.6 mm, which corresponds to 4.8% of the measured distance. By using calibration, the errors can be significantly decreased. In comparison to commercial systems for instance, products from Canesta and Mesa Imaging, the result were reasonably good. The conclusion of this work highlighted the good performance for the cost, in comparison to equivalent commercial products.

Range Imaging in Real-time

In [JCDP08], the authors developed a ToF imaging system with the help of an FPGA. Laser diodes are used for illumination and the Dalsa Pantera 1M60 digital video camera captures the images. A constructed circuit board with Direct Digital Synthesizer ICs is used to control the emitted and reference signal. The architecture on the FPGA is depicted in Figure 2.13. Via the cameralink interface, the frames are transmitted to the FPGA. For computation of the phase, five raw frames are used. After one received frame, a new depth image is calculated. With a configured image resolution of 128 x 128 pixels a frame rate of up to 30 FPS can be achieved. Furthermore, the system is evaluated by capturing moving targets with modulation frequency of 40 MHz, which results in an error of around 4 cm.

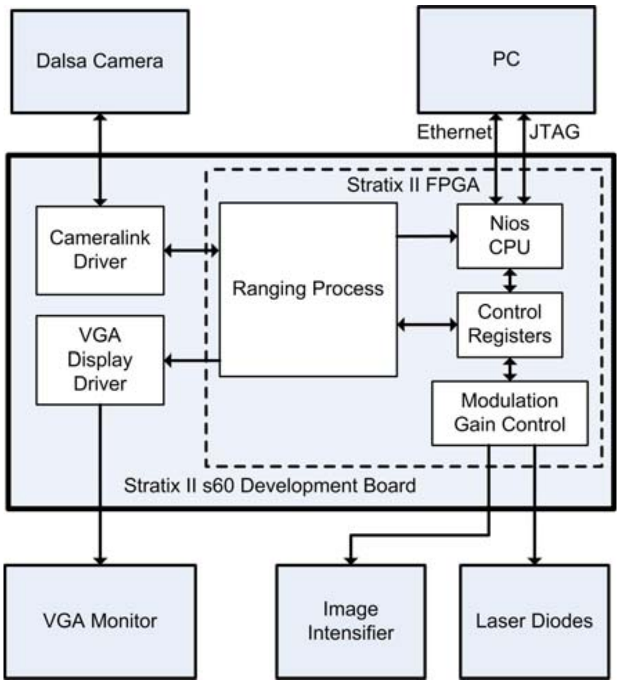


Figure 2.13: Development board and interconnections [JCDP08].

Time-of-Flight Range Imaging

The authors of [JBP+10] implemented an FPGA-based system to efficiently compute the phase of captured raw data. The same hardware platform, as described in [JCP+09], was used. In particular, the trade-off between the accumulator's bit widths and the processing accuracy was evaluated. A 4-Phases algorithm was implemented using accumulators to calculate the pixel intensities and the arcus tangent function. The results showed that for the proposed system, an accumulator width of 14 bits is necessary to provide a phase error below the system standard deviation. Furthermore, a novel algorithm was presented for extending the unambiguous range by using two different modulation frequencies.

2.2.2 Image Processing on the Xilinx Zynq Platform

The Xilinx Zynq platform has attracted the interest in recent years for the realization of FPGA-based imaging systems. In this section, two-dimensional imaging projects are presented which implement this framework. The publications focus on a specific use-case application and outline the advantages of the Zynq platform.

The Zynq platform consists of several ARM CPUs and an FPGA, which are integrated on one chip to provide high data bandwidth between both components, as described in [Xil15j]. The system is highly flexible as software can be executed on an ARM CPU while simultaneously using hardware-acceleration on the FPGA. In addition, Xilinx provides a practical approach of rapid prototyping for the platform, namely High Level Synthesis (HLS).

Road Sign Recognition

Russell et al. [RF13] has proposed a road sign recognition system using the Zynq platform. The detection of road signs is an essential component in autonomous driving, as well as automotive assistance systems and road sign maintenance. The system was implemented on the Zedboard, which is a development board with an integrated Xilinx Zynq chip.

In Figure 2.14 the basic architecture of the system is illustrated. The VITA-2000 image sensor from ON Semiconductor provides a stream with an image resolution of 1920 x 1080 at 72 frames per second. Image pre-processing is performed with hardware-integrated components on the FPGA. Predefined video hardware modules provided by Xilinx are used to correct the image, such as defective pixel correction. For the color based filtering, a manually developed hardware unit performs the color segmentation algorithm. The classification of the shapes and the identification of the signs from a database are implemented on the ARM CPU with the help of OpenCV.

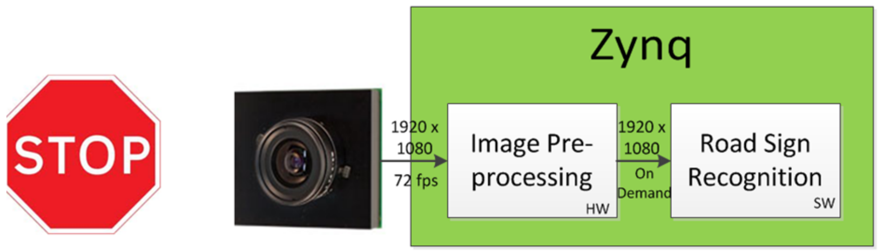


Figure 2.14: Basic system architecture for road sign recognition [RF13].

The results showed that the system is able to detect all red and blue signs in one frame, in approximately 5 seconds. Compared to other designs that also use hardware-acceleration, the proposed system is slower. The authors considered that other implementations only process frames with an image resolution of VGA quality or below, while the proposed design processes Full HD images. Furthermore, more software computations could be hardware-integrated on the FPGA to achieve an essential speed-up. The work was concluded with an emphasis of the potential of the Zynq. This platform is well suited for embedded road sign recognition or other imaging systems. The design of the entire system took only six weeks due to the use of Xilinx tools and the standard memory interface.

Fall Detection

Several publications exist that implement a fall detection use-case application using the Xilinx Zynq platform, as shown in [SCH$^+$14], [ASA$^+$14] and [NBVT14]. All of these systems similarly investigated fast prototyping of fall detection in daily life. In order to achieve the real-time requirement, parts of the algorithms were hardware-accelerated using High Level Synthesis tools, which directly translate code in high-level language, such as C++, into a hardware description.

In [SCH$^+$14], the authors developed a smart-camera based fall detection processing system. Based on the Zynq platform, a fast prototyping methodology was proposed which allows comparison between different real-time fall detection implementations. By using high-level algorithmic description, such as C++, and High Level Synthesis tools, this hardware/software system was used in the design space exploration phase for testing different architectures.

The implemented use-case has applications in the detection of falls of the elderly in daily life. Therefore, the automatic detection of falls in real-time is required. A fast response of the system is very important. The algorithm consists of image acquisition, low-level processing and fall detection. Most of the processing steps are performed on the ARM using OpenCV. The classification algorithm, which is described in C++, is used to generate a hardware component with the help of High Level Synthesis.

The basic hardware architecture is illustrated in Figure 2.15. The Intellectual Property (IP) description, which performs the hardware-accelerated computation, communicates with the two ARM CPUs and the extended memory through an AXI memory interface. This hardware module can be manually developed or generated using HLS. A pre-built Linux is used to provide a software adaption layer between hardware and use-case application.

The authors especially outlined the benefits of the hardware/software integration. Therefore, the used prototyping flow of the system is depicted in Figure 2.16. The flow is proposed for the Zynq platform, but can be used also for similar hardware/software systems. It allows a fast validation and prototyping of the fall detection application. Firstly, part 1 illustrates the standard flow on a host computer. After the initial algorithm development, the computation steps are validated and optimized.

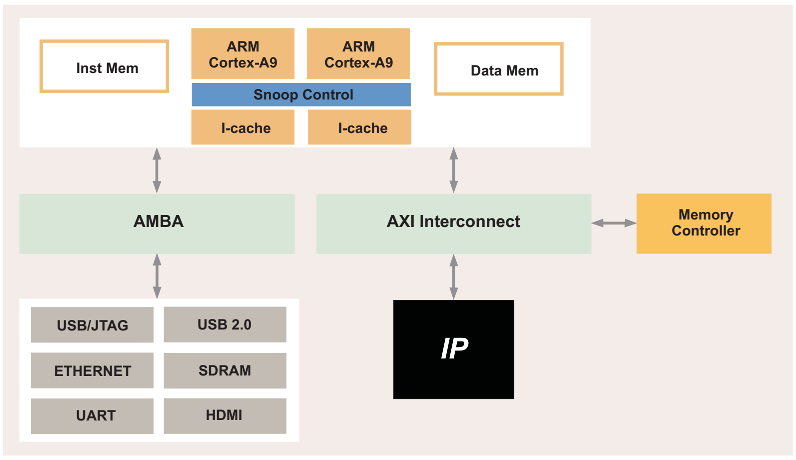


Figure 2.15: Hardware architecture including HLS-generated IP [SCH+14].

In part 2, the basic flow for creation of the hardware/software system is presented. Therefore, the initial algorithm is partitioned into hardware/software. The least regular part should be performed on the CPU to provide high flexibility. The most regular part is implemented on the FPGA. Compared to the algorithm on the host PC, additional computation performance is available because the regular parts are then hardware-accelerated. The hardware components can be manually developed or with High Level Synthesis.

Finally, in part 3 all sub-components are compiled and executed on the target platform. Implementation results can be used to start a further iteration of part 2 to improve the overall system.

The synthesis and timing results of the HLS-based hardware component were evaluated. Different versions were implemented using improved synthesis directives. The execution time of the full ARM-based software implementation (30 μs) was almost the same as the generated hardware IP without any directives (29 μs). By using synthesis directives the final execution time could be decreased to 2.5 μs by using a pipeline architecture and parallelization of computation steps. The work was concluded with a description of the benefits that the Zynq platform offers for fast prototyping.

Border Detection

Sabouri et al. [SGC14] developed a border detection processing system for early diagnosis of melanoma using the Xilinx Zynq platform. By analyzing melanoma skin lesions images, skin cancer can be discovered. The goal of the study was the investigation of novel methods on an embedded system. High resolution and performance are important for portable imaging systems used in the medical domain.

Different edge detection methods, such as Sobel, Kirsch, Canny, LoG, were therefore implemented and analyzed regarding accuracy and performance. The hardware component used for border detection was generated with HLS software, which converted the C++ code to synthesizable FPGA code.

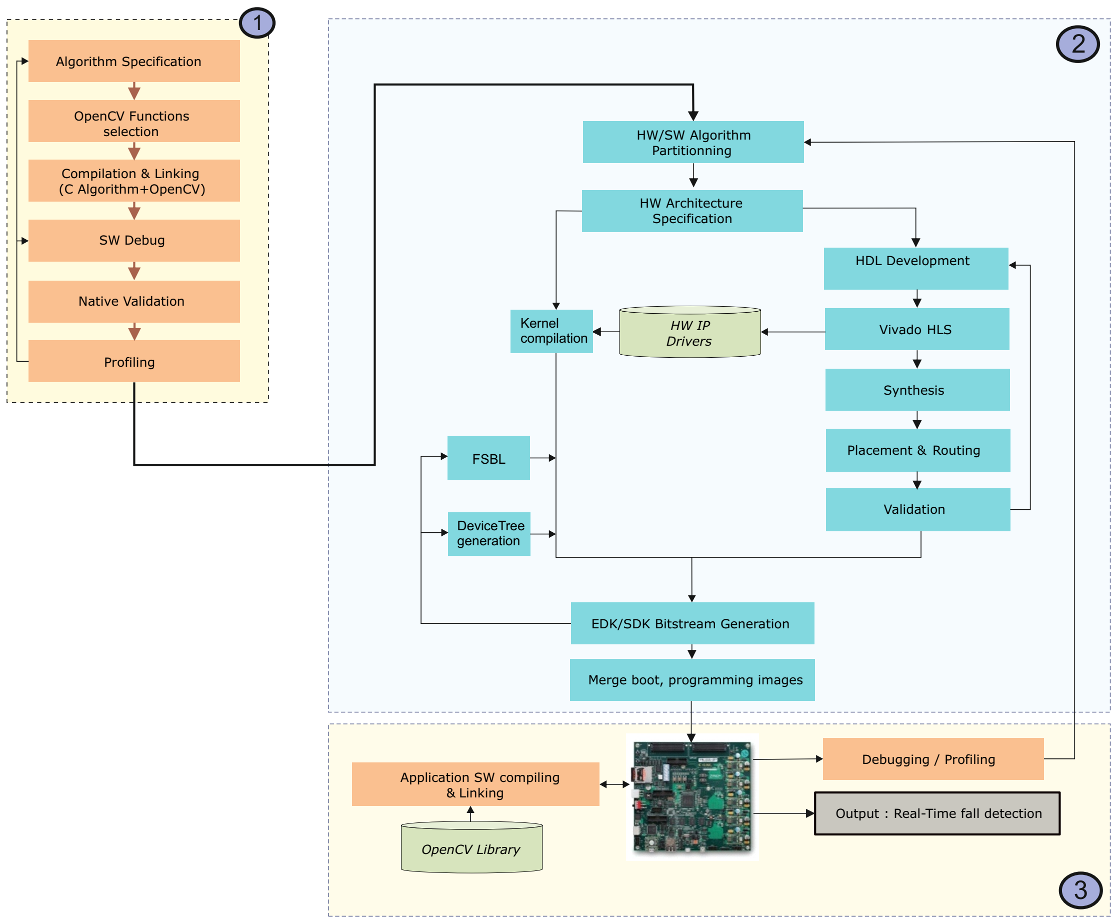

1
Algorithm Specification
OpenCV Functions selection
Compilation & Linking (C Algorithm+OpenCV)
SW Debug
Native Validation
Profiling
2
HW/SW Algorithm Partitionning
HW Architecture Specification
HDL Development
Kernel compilation
HW IP Drivers
Vivado HLS
Synthesis
Placement & Routing
Validation
FSBL
DeviceTree generation
EDK/SDK Bitstream Generation
Merge boot, programming images
Application SW compiling & Linking
OpenCV Library
Debugging / Profiling
Output : Real-Time fall detection
3

Figure 2.16: System prototyping flow of the fall detection system [SCH+14].

The results indicated that the extended 5x5 canny edge detection algorithm had the best performance compared to other presented methods. The authors showed that it is possible to achieve a frame rate of up to 60 FPS, which is well suited for real time applications. The work also outlined the rapid processing power, which is demonstrated by the Zynq platform.

Grayscale Conversion and Convolution

The authors of [AGBS15] implemented a real-time image processing system on the Zynq platform using hardware/software co-design. The Zedboard, a Zynq-based development kit from Xilinx, was used to explore different realizations of image algorithms. The following hardware-accelerated algorithms were investigated: grayscale conversion and convolution operations, such as edge detection, sharpening, blurring).

The overall system consists of a HDMI monitor, a standard USB camera and the Zedboard. A Linux operating system on the ARM processor is used to capture images from a video source. A manually developed image co-processor in Verilog HDL performs the algorithms on the gathered image data.

The experimental results showed that images with a resolution of 256 x 256 could be processed with around 40 FPS. The hardware design utilized around 5% of the available flip-flops and look-up tables. For future work, a direct memory access (DMA) implementation was proposed, which would result in an essential speed-up of the system.

2.3 Existing Framework

The ToF 3D imaging system presented in [DFH+15] outlines the current system of this thesis. Therefore, this paper is described in detail in this section.

A ToF camera sensor provides raw data to an automotive computation platform. The distance is computed by performing an efficient implementation of a ToF processing dataflow. The works shows a robust solution for mixed-critical applications, in the automotive domain, on a safety-critical platform. This existing framework is already implemented and used as a starting point for the new proposed system.

2.3.1 System

The architecture, as illustrated in Figure 2.17, consists of a ToF camera and an automotive microcontroller. Through a parallel interface which connects the two systems, raw image data provided by the camera is transmitted. The processing of the raw data is then implemented on the automotive microcontroller, which calculates depth data on one core and computes use-case specific data and events on the other, in a mixed-critical environment.

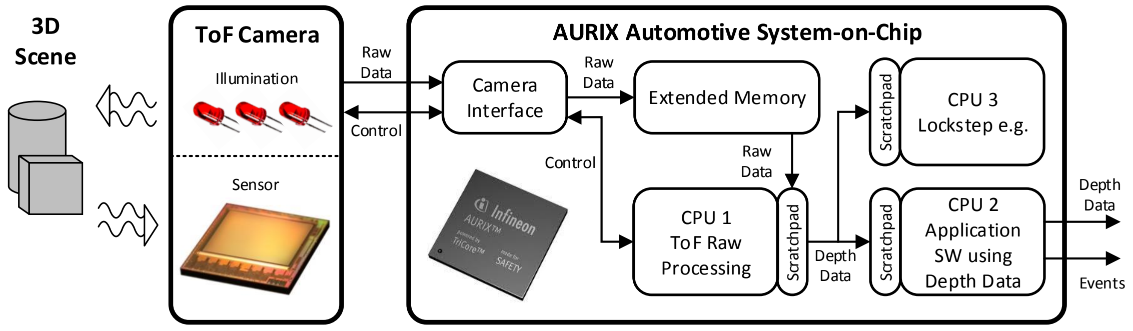


Figure 2.17: Architecture of the ToF 3D imaging system [DFH+15].

Automotive System-on-Chip

The computation platform AURIX TC299 from Infineon Technologies is used, which targets performance and safety critical applications in the automotive industry, as outlined in [Inf14]. It is compliant to several standards including IEC 61508, ISO 26262 and ISO 25119. The multicore approach is based on three independent Tri-Core CPUs, where each core operates with a maximum clock frequency of 300 MHz. The Tri-Core architecture, implemented on one chip package, unifies the instruction sets of a microcontroller, a RISC processor and a DSP. Several concepts for safety critical systems including lockstep architecture and safe internal communication buses are well-integrated in the platform. These features express the perfect environment for secure, mixed-critical and real-time applications.

One issue concerning memory-intensive computations for use-cases, such as gesture recognition, is the limited memory capacity of the AURIX. There is in total 2728 kBytes static RAM (SRAM) available, which is divided into 2048 kBytes global extended memory and up to 240 kBytes scratchpad memory attached to each Tri-Core. Therefore, difficult image processing must be implemented in a highly optimized way. Due to this limitation, implementations of complex data processing algorithms are hardly feasible.

Time-of-Flight Camera

The camera evaluation kit used in the framework is an ongoing joint project of Infineon Technologies and PMDTechnologies. The camera system is Infineon's IRS10x0C Evaluation Kit that features the 3D image sensor IRS10x0C, as shown in [Inf13]. The ToF-based camera can be used in consumer electronics as well as in critical applications, such as automotive domain and robotics. It features a flexible setup which consists of a LED-based illumination unit and a sensor System-on-Chip (SoC). The evaluation board features different external interface, for example parallel interface and CSI-2, and allows access to important signals of the image sensor. Some characteristics are the resolution of up to 100k pixels, a maximum frame rate of up to 100 FPS and a modulation frequency of maximum 100 MHz. Furthermore, robustness during darkness and full sunlight is given by implementing suppression of background illumination on the sensor.

The camera is configured as followed: image resolution is set to 160 x 120 pixels, 12 Bits are transmitted per pixel and 4 phases are measured with an exposure time of 1 ms. The output, processed on the AURIX, consists of a 16-Bit depth and 8-Bit amplitude image.

2.3.2 Image Processing

The framework computes the depth data from the raw data, transmitted from the camera sensor, by implementing several steps of a generic ToF processing pipeline. A use-case in the automotive domain can be implemented, for example, interior monitoring. Due to the memory and performance restrictions of the AURIX, only the necessary steps of the ToF processing algorithm are implemented. In detail, only one modulation frequency is used, post-processing, such as filter, analysis, Cartesian coordinate conversions, are omitted and only a minimum of systematic error removal is carried out.

In order to clearly illustrate the interaction of the different components, a simplified sequence diagram is depicted in Figure 2.18. The data processing steps of one full calculation of a depth/amplitude image, including the start of the system, are shown. An optional host PC is connected over Ethernet with the AURIX evaluation board. The calculated data is transferred via User Datagram Protocol (UDP) to the PC to display a live-stream in a simple application or to debug the computed images.

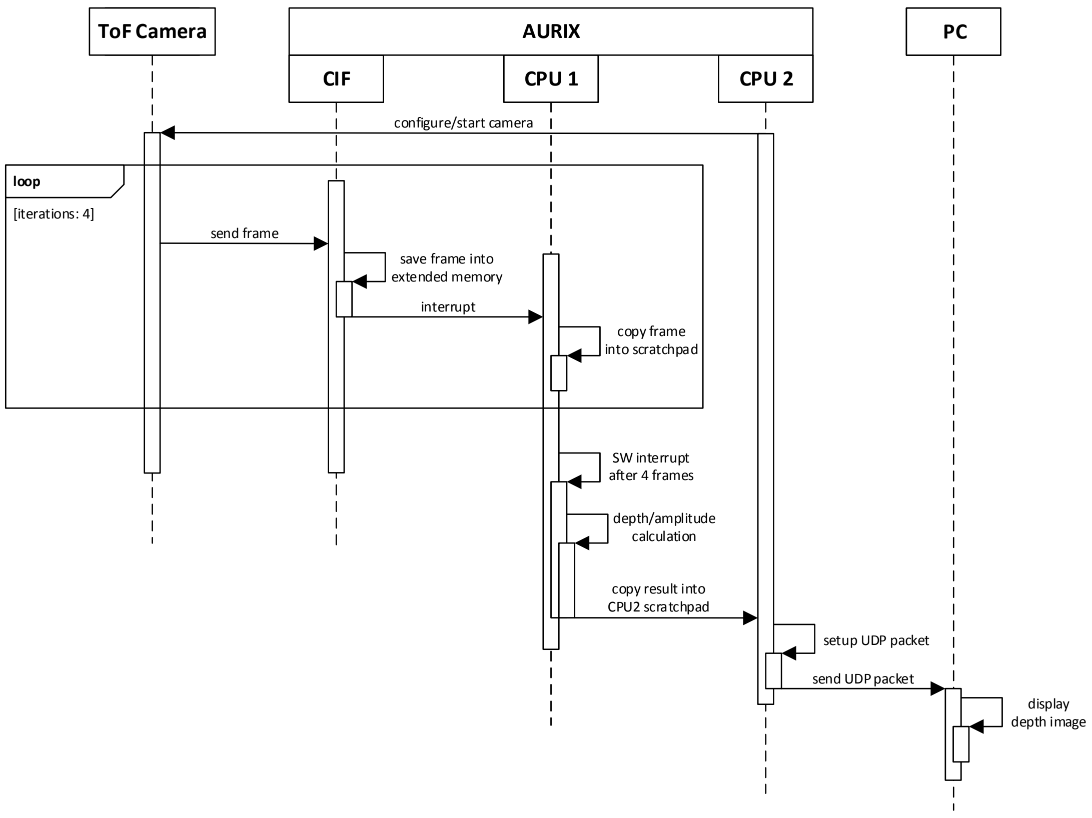

ToF Camera
AURIX
PC
CIF
CPU 1
CPU 2
configure/start camera
loop
[iterations: 4]
send frame
save frame into extended memory
interrupt
copy frame into scratchpad
SW interrupt after 4 frames
depth/amplitude calculation
copy result into CPU2 scratchpad
setup UDP packet
send UDP packet
display depth image

Figure 2.18: Sequence diagram of the current system [DFH+15] (with changes).

On a reset of the AURIX, the processing platform is started. At first CPU 2 configures the ToF camera over the Inter Integrate Circuit (I2C) interface, and the camera interface (CIF) is activated. Therefore, a system reset of the camera evaluation board is done and the configuration stored on the AURIX is sent. After sending the start signal, the image transmission is started. The image stream is received on the AURIX via the parallel interface and pushed from the CIF module into the extended memory.

After the transmission of one raw frame, an interrupt is triggered. Upon this event, CPU 1 copies the frame from the extended memory in its scratchpad memory. These processing steps are repeated four times resulting in a software interrupt thrown by CPU 1 to start data computation. The calculation is efficiently implemented and results in a phase and amplitude image. The images are stored in its own scratchpad memory. After the execution of the ToF algorithm, the phase and amplitude image are copied to the scratchpad memory of CPU 2 by using DMA.

After finishing computation and copy operation, an interrupt is thrown by CPU 2, signaling that the results can be further processed. On this core, the transmission to the host PC for display purposes is carried out. A UDP stack is implemented in order to send the data via the network interface. Hence, multiple UDP packets of the phase image and amplitude image are created and transmitted. Because one single UDP packet is too small to transmit the complete result, the data has to be divided and provided with some metadata. With a simple application, the calculated data is viewed as a live-stream and analyzed.

The PC is used to simulate a real system that can react on the use-case events generated on the AURIX. For instance, a human detection system can be implemented on CPU 3. If a person sits on a seat without a fastened seat belt, an event will be thrown and the car will be notified to alert the passengers with an acoustic signal.

2.3.3 Results

The timing results of the work are presented in Table 2.1. Due to an exposure time of 1 ms, it requires 4 ms to gather the raw data of four measurements. It takes a further 3.4 ms to transfer the four images from the extended memory into the scratchpad. In addition, it takes 3.25 ms to perform the ToF processing and saving the data into the scratchpad of CPU 2. These results take up a total time of 12.5 ms, which includes some overhead timing to compute a depth and amplitude image. Therefore, the frame rate of the camera can be set to a maximum of 80 FPS. The depth (16 Bits/pixel) and amplitude (8 Bits/pixel) images take up 57,600 Bytes in total. The memory results are shown in detail in Table 2.2. For the computation of the depth and amplitude images the equations 2.7 and 2.8 are efficiently implemented. Thus, the results meet the predefined accuracy requirements for the use-case.

$$\varphi = \arctan\left(\frac{A_{270^\circ} - A_{90^\circ}}{A_{0^\circ} - A_{180^\circ}}\right) \tag{2.7}$$

$$A = \frac{\sqrt{(A_{270^\circ} - A_{90^\circ})^2 + (A_{0^\circ} + A_{180^\circ})^2}}{2} \tag{2.8}$$

The conclusion of this work is a robust multi-core ToF framework for mixed-critical applications in the automotive domain. The processing is efficiently performed to provide a frame rate of 80 FPS. Consideration of functional safety, such as the ISO 26262 standard, is also fulfilled. As future work, a hardware accelerator can be integrated into the framework to outsource performance-intensive calculations and to have more free resources available for use-case applications on the AURIX.

Table 2.1: Timing results of the ToF framework [DFH+15].

Operation	Time[ms]
Exposure Time and Raw Data Acquisition	4
Raw Data DMA Transfers	3.4
Phase and Amplitude Calculations	3.25
Total (including Overhead Timing)	12.5

Table 2.2: Memory results of the ToF framework [DFH+15].

Memory Description	# [Bytes]
Raw Data in Extended Memory	153600
CPU 1 Scratchpad Raw Data Memory	115200
CPU 1 Scratchpad Phase and Amplitude Data	0
CPU 2 Scratchpad Phase and Amplitude Data	57600

2.3.4 Limitations

One of the major challenges is the memory limitation of the AURIX. The complete amount of internal memory is 2728 kBytes of SRAM, which is divided into 2048 kBytes of extended memory and three times up to 240 kBytes of scratchpad memory allocated to each CPU. In order to compare this given condition with the memory used in practice, the configuration of the camera has to be considered.

The image resolution of the chosen configuration is 160 x 120 pixels (16 Bits/pixel), and for a computed distance image four raw measurements are necessary. Due to this setup, the memory needed for one image calculation is shown in Equation 2.9. For four raw frames around 115 kBytes of memory is needed. As a result, the scratchpad memory will be almost half full if the complete raw data is saved for further computation. It is not possible to save four frames with the maximum image size of the camera system, which is approximately 100k pixels. Furthermore, pre-processing (e.g., averaging) that uses higher amount of images, is not feasible because only a small number of frames can be stored.

$$\begin{aligned} memory_{frames} &= image_{rows} \cdot image_{columns} \cdot memory_{pixel} \cdot \#frames \\ &= 120 \cdot 160 \cdot 12\ bit \cdot 4 \\ &= 115200\ Byte \end{aligned} \tag{2.9}$$

Another limitation is the computation power of the automotive SoC. The time needed for calculation of the phase and amplitude calculation is 3.25 ms. As the computation is in software, and not performed in parallel to the receive logic, it is completely added to the total time (raw data acquisition, DMA transfers and calculation). By keeping in mind that the implemented algorithm only calculates the phase and amplitude without any systematic error correction and further processing, like filters, the total time would be largely increased by adding an error correction consisting of many operations. It is possible to involve additional CPUs, but for the use of another core, the data has to be copied into its scratchpad memory. Thus, the performance in terms of latency will be decreased.

Finally, due to the mentioned memory and performance limitations, it is not possible to implement challenging use-cases, such as gesture recognition, on the AURIX. Thus, the interest in a new hardware-accelerated ToF processing platform, with a sufficient high-speed memory, is raised.

Chapter 3

Design

The focus of this chapter is to provide the design of the new system. The architectures and interactions are presented in a top-down approach. Therefore, the requirements are derived from the already described existing framework and its limitations. In addition, the ToF processing algorithm is introduced and the proposed High Level Synthesis design decisions are explained.

3.1 Requirements

The main aim of this master's thesis is the creation of a flexible and fast Time-of-Flight processing platform. Following requirements for the platform are as specified:

- **HW-accelerated Computation**

  A ToF co-processor running on an FPGA shall be used to enhance the performance of imaging algorithms. Operations that are computational intense shall be shifted from software, such as a microcontroller, to the FPGA.

- **High Flexibility**

  The hardware/software framework shall be highly customizable in terms of different ToF applications. The system shall be designed for a high degree of reuse for projects in future by allowing rapid algorithm prototyping, such as High Level Synthesis. Camera configuration details, for example the image resolution, shall be configured during runtime without a reprogramming of the FPGA.

- **All-in-one System**

  The FPGA system shall act as interface between the camera system and an automotive microcontroller connected through a parallel interface. Hence, the target platform's computation and memory resources shall be free to implement more challenging use-cases. In future, the calculated data shall also be transmitted via other interfaces (e.g., Ethernet, USB) to other processing devices, for instance, a host PC.

- **System Performance**

  A hardware-integrated co-processor shall be used to perform the calculation of depth data on the FPGA (e.g., distance image, amplitude image, 3D coordinates). Furthermore, the transmitted data size shall be decreased. Thus, the overall performance regarding throughput shall be increased.

For a camera configuration with an image resolution of 160 x 120 and an exposure time of 1 ms, two different ToF pre-processing algorithms shall be efficiently implemented and analyzed to show the feasibility of the system:

- **4-Phases Time-of-Flight Algorithm**

  The 4-Phases algorithm, as mentioned in Section 2.1.4, calculates the distance image, the amplitude image and the 3D coordinates of a scene. Therefore, four raw frames with a phase shift of 90 degrees are captured from the camera system. In addition, a compensation of common system errors, as introduced in Section 2.1.6, is performed. Such an algorithm is used in a typical gesture recognition use-case where high frames rates and high relative distance accuracy are required.

- **8-Phases Time-of-Flight Algorithm**

  As a second ToF algorithm, the 8-Phases calculation is performed. According to Section 2.1.5, two depth images recorded with two different modulation frequencies (e.g., 60 MHz, 80 MHz), are necessary, and are combined by an unambiguous range extension, after systematic error compensation. The result is a distance image, an amplitude image and a 3D point cloud. This approach can be used for a Google Tango indoor navigation algorithm that can cope with lower frame rates, but requires high absolute distance accuracy.

3.2 System Architecture

3.2.1 Concept

In order to create a hardware-accelerated imaging platform, an FPGA is used to shift the operations of ToF pre-processing and use-case processing onto hardware. The basic architecture of the new proposed imaging system is depicted in Figure 3.1.

Starting with the existing framework, as already described in Section 2.3, the FPGA works as an interface between the AURIX evaluation board and Infineon's ToF evaluation kit. Due to the hardware-integrated pre-processing, parts of the work load of the memory and computation resources are free on the AURIX, and allow more challenging use-cases.

In addition to the distance calculation, the FPGA handles the camera control and configuration. For the configuration, the camera's I2C interface is used whereas for transmission of the raw data, the parallel interface of the camera is utilized. The parallel interface from the FPGA to the AURIX is equally specified (e.g., data signals, synchronization signals), which has a major advantage in that the camera can also be directly

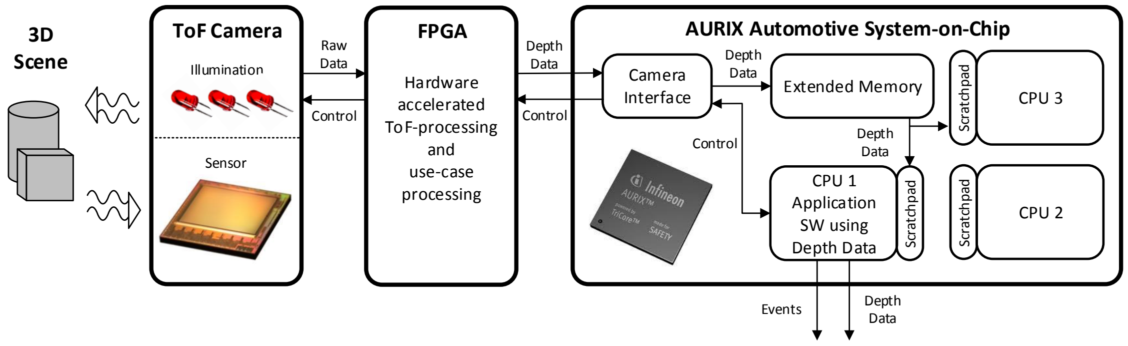


Figure 3.1: Basic system architecture of the proposed platform.

connected with the AURIX without any big changes being made. So, the data is also transferred via PIF to the AURIX and the AURIX sends control commands to the FPGA via I2C.

One big advantage of this architecture is that the system can also be used without AURIX. For instance, the calculated data can be transmitted for further processing over a network interface on the FPGA board to a host PC.

3.2.2 Selection of the FPGA Platform

In order to fulfill the requirements regarding performance and flexibility, a decision for the best suitable FPGA platform has to be made. Based on the analysis of the requirements and the literature research, the hardware/software platform Xilinx Zynq has been chosen. This platform, as outlined in [Xil15j], provides a heterogeneous ARM-based FPGA system that allows the development of software on an ARM CPU with the ability to use hardware components on an FPGA. These two components are integrated on a single chip, which allows high data bandwidth between ARM and FPGA. Furthermore, High Level Synthesis, a practical approach to rapid prototyping, is available.

The architecture of the Zynq SoC is presented in Figure 3.2. The main parts are the Processing System (PS) and the Programmable Logic (PL). The PS contains the ARM cores, NEON/DSP processors and several interface controllers, for instance, Ethernet, USB, I2C as well as the memory controller to the external memory. The PL is a typical FPGA. The communication between the different sub-components (e.g., CPUs, interface controllers, FPGA) is performed over an AMBA/AXI bus. The configuration of the hardware components from the CPU is done via General Purpose (GP) AXI ports, whilst the communication between the FPGA and the memory is done via High Performance (HP) AXI Ports.

3.2.3 Interaction

The interaction of the different components of the overall system is depicted as sequence diagram in Figure 3.3. The shown hardware components are the ToF camera, Infineon's

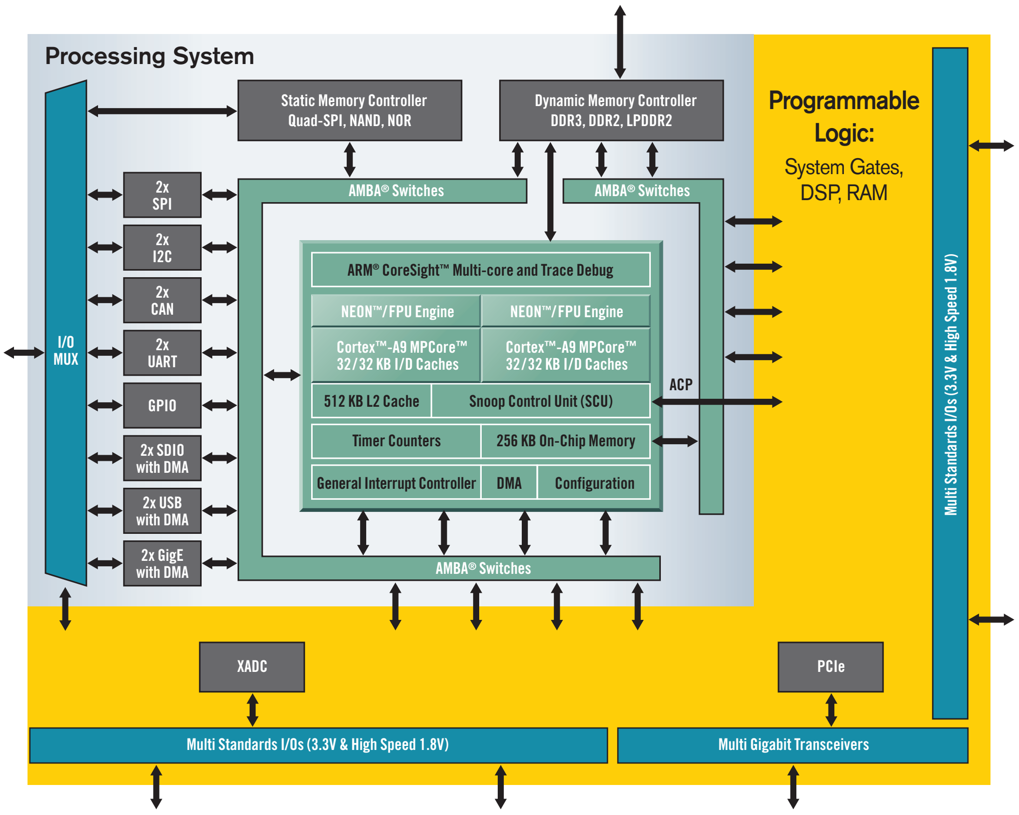

Processing System
Static Memory Controller Quad-SPI, NAND, NOR
Dynamic Memory Controller DDR3, DDR2, LPDDR2
Programmable Logic:
System Gates, DSP, RAM
2x SPI
2x I2C
2x CAN
2x UART
GPIO
2x SDIO with DMA
2x USB with DMA
2x GigE with DMA
I/O MUX
AMBA® Switches
AMBA® Switches
ARM® CoreSight™ Multi-core and Trace Debug
NEON™/FPU Engine
NEON™/FPU Engine
Cortex™-A9 MPCore™ 32/32 KB I/D Caches
Cortex™-A9 MPCore™ 32/32 KB I/D Caches
512 KB L2 Cache
Snoop Control Unit (SCU)
Timer Counters
256 KB On-Chip Memory
General Interrupt Controller
DMA
Configuration
ACP
AMBA® Switches
XADC
PCIe
Multi Standards I/Os (3.3V & High Speed 1.8V)
Multi Gigabit Transceivers
Multi Standards I/Os (3.3V & High Speed 1.8V)

Figure 3.2: Basic architecture of the Xilinx Zynq platform [Xil11a].

AURIX and the Zynq platform. The host PC used for display purposes, as mentioned in the current system, is not illustrated.

The AURIX platform controls the system by triggering the Zynq to start a use-case through I2C commands. Next, the hardware-accelerated computation platform resets, configures and starts the ToF camera system via the I2C interface. The raw data captured from the camera is transferred through the parallel interface and fetched from the Zynq, which saves the data into the global extended memory. For the required 4-Phases algorithm, four raw images with the same modulation frequency and a phase shift of 90 degrees are sent.

In addition to the implemented data processing steps, memory or performance intense operations of a use-case can be performed. The resulting data, for example distance image, amplitude image, 3D point cloud, is then forwarded through a parallel interface to the AURIX, where it is received from the camera interface and saved into the global extended memory. A software interrupt thrown from the CIF notifies CPU 1 to copy the received results into its own scratchpad. After that, use-case specific events are computed and transferred via available interfaces, for example Ethernet, I2C, CAN bus, to other processing targets. Optionally, the results can be copied into the scratchpad memory of CPU 2 for further processing. For display purposes, the calculated data is sent over the network interface using UDP to a host PC.

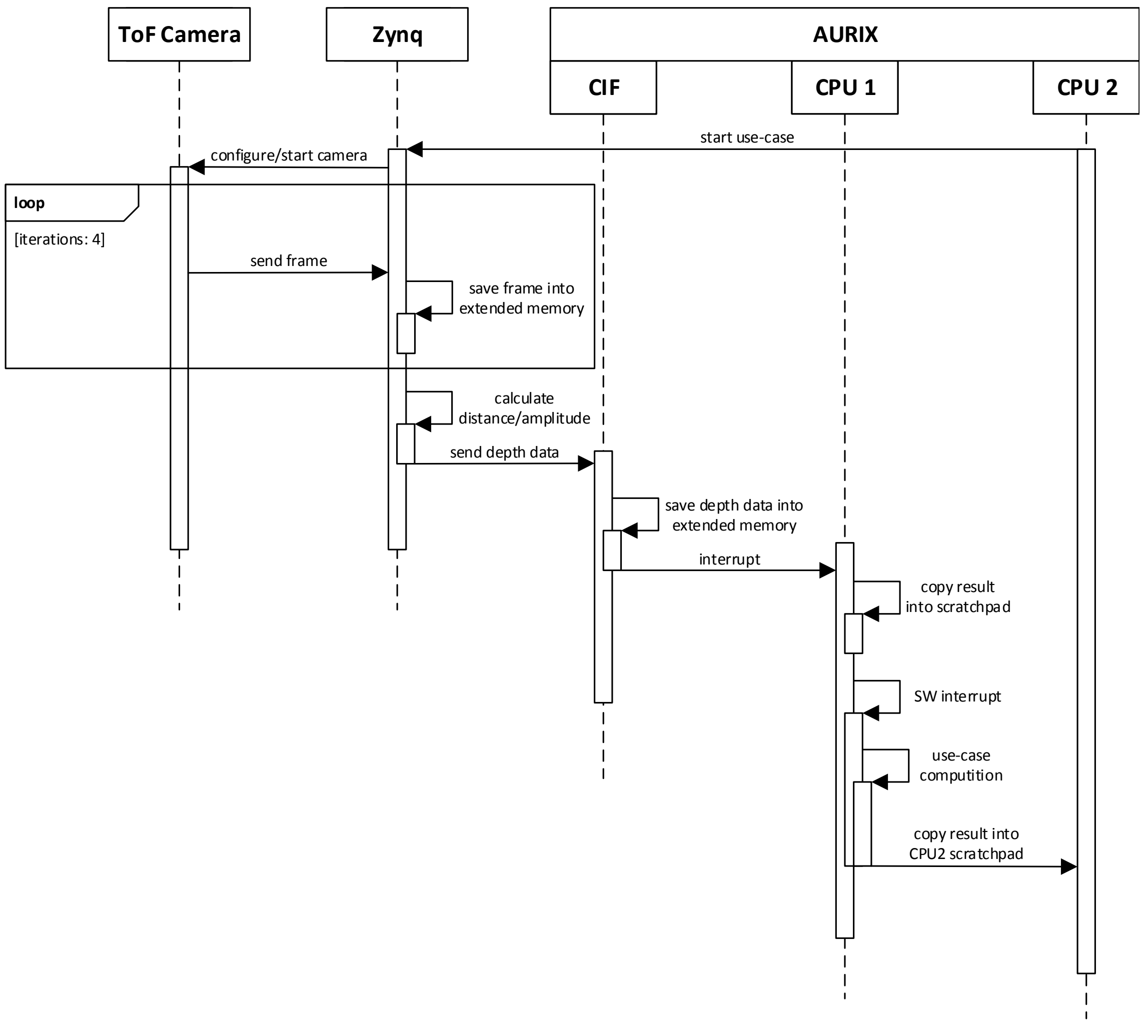

ToF Camera
Zynq
AURIX
CIF
CPU 1
CPU 2
start use-case
configure/start camera
loop
[iterations: 4]
send frame
save frame into extended memory
calculate distance/amplitude
send depth data
save depth data into extended memory
interrupt
copy result into scratchpad
SW interrupt
use-case computition
copy result into CPU2 scratchpad

Figure 3.3: Sequence diagram of the new system.

3.3 Architecture on the Zynq Platform

3.3.1 Concept

The architecture of the hardware-integrated system on the Zynq is depicted in Figure 3.4. Several sub-components realize the functionality of the fast and flexible image processing system. The benefit of having clearly separated units connected through interfaces allows for a high level of flexibility. The system consists of a control block in software, several FPGA-based hardware components, such as the video receiver/transmitter and an image processing unit.

The receiving and transmitting logic are not connected to the image processing block, so as to allow the use of interfaces other than the PIF. For instance, the USB interface can be used to receive images from the camera or calculated depth data is transmitted over the network interface to the AURIX. All components are connected through the AXI interface to the control logic, which manages the correct execution of the sub-components.

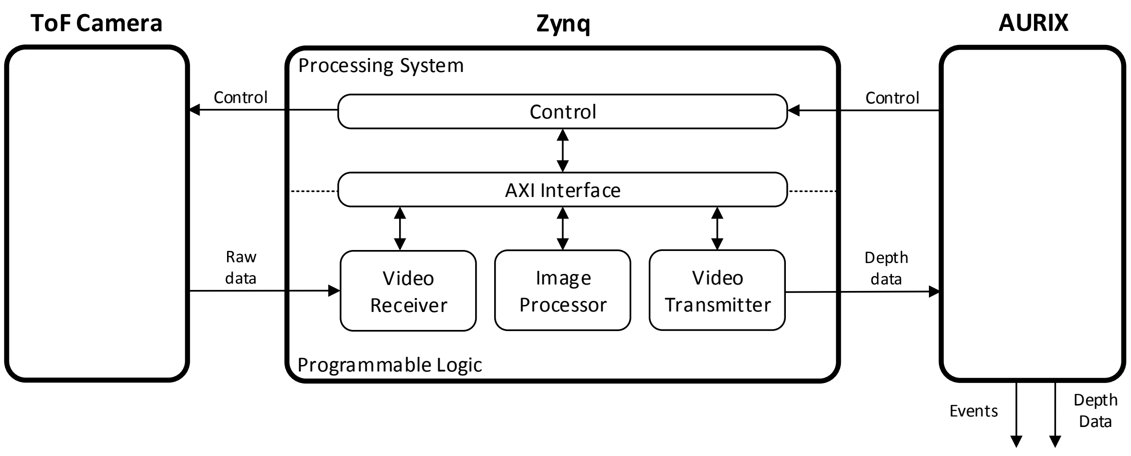


Figure 3.4: Architecture on the FPGA of the new system.

The **Control** unit is responsible for the correct functionality of the system by configuring all sub-components and controlling the dataflow. This component is executed on the ARM core to provide high flexibility. Furthermore, it responds to control commands via I2C from the AURIX, such as starting a special implemented use-case on the Zynq. The camera system is also controlled from this component via I2C. In addition to the I2C interface, a reset line is also used in the control interface to reset the camera from the FPGA.

The parallel data interface of the ToF camera is connected to the **Video Receiver** component. It stores the camera's gathered images in the global extended memory via the AXI interface. The receive logic has to interpret the 12-Bit data signals and the synchronization details correctly to push an image into a frame buffer. The control logic is notified when the use-case specific number of frames is received.

The **Image Processor** unit performs the ToF imaging processing. The hardware-integrated operations are independent from the algorithm that is chosen to provide the flexibility to implement further ToF pre-processing or use-cases applications. Through the AXI interface, one or several images are read and saved after successful computation. This design allows flexible replacement of the image processor.

The AURIX's parallel interface is connected through the **Video Transmitter** to the Zynq. The computed depth data is read with the AXI interface from the memory and used to provide 12-Bit data signals and the synchronization signals.

The **AXI Interface** connects the external global memory to the previous mentioned sub-components. It provides a protocol for communication to the external memory for a simple use of the memory, for example, read and write commands. The external memory is needed to save the images because the internal memory of an FPGA does not have enough space.

3.3.2 Interaction

In Figure 3.5 the interaction of the different sub-components is depicted. Due to simplicity, the AURIX object is summarized to only one receive and process operation. In detail, the AURIX operates similarly as illustrated in Figure 3.3. The CIF interface receives the data and CPU 1/CPU 2 are used for further processing.

The control logic is not presented as sub-component of the Zynq to further reduce the complexity of the sequence diagram, this component is connected to every sub-component. The interaction between the video receiver, the image processing unit and the video transmitter at the beginning of the raw data transmission is shown. The starting sequence for the system is not illustrated because no control logic is displayed.

The system is started through an external reset, for example, the main reset of the AURIX. After start-up, the AURIX sends a command to the control logic of the Zynq to initiate the pre-processing or use-case application via the I2C interface. Next, the Zynq sends the corresponding configuration and a start command via I2C to the camera system. The ToF camera gathers typically four images and sends one image via another through the parallel interface.

The already activated video receiver recognizes the synchronization signals and saves the input data into a frame buffer in the external memory. It is important that the input is immediately saved otherwise the next frame overrides the current one. After enough raw data is received, for example four raw frames for the 4-Phases algorithm, the video receiver notifies the image processor via the control logic.

The image processor computes the ToF depth data in an optimized way to minimize time and optimize accuracy. During computation, the next frames are sent and saved by the receive logic. The processing time must be short enough to be finished before the next raw frames are transmitted for calculation. Otherwise, it is possible to override images in the frame buffer. The results of the image processor are in the external memory and copied into another memory part via the memory interface, in particular the output frame buffer.

After depth calculation of the image processing unit, the data is located in the output frame buffer. The results are read from the external memory, and the synchronization signals for the correct transfer are generated. From there, the data is transmitted via the parallel interface to the AURIX. The data is received via the CIF on the AURIX for further use-case specific processing. The transmission of the data is done in parallel to the next receiving and computation operation.

Finally, after successful completion of the use-case, the AURIX can stop the data processing of the Zynq via an I2C command. If only pre-processing is used on the Zynq, a live-stream of the system can continue without a stop signal. From the AURIX, use-case specific events are sent to other processing devices, such as a host PC for display purposes.

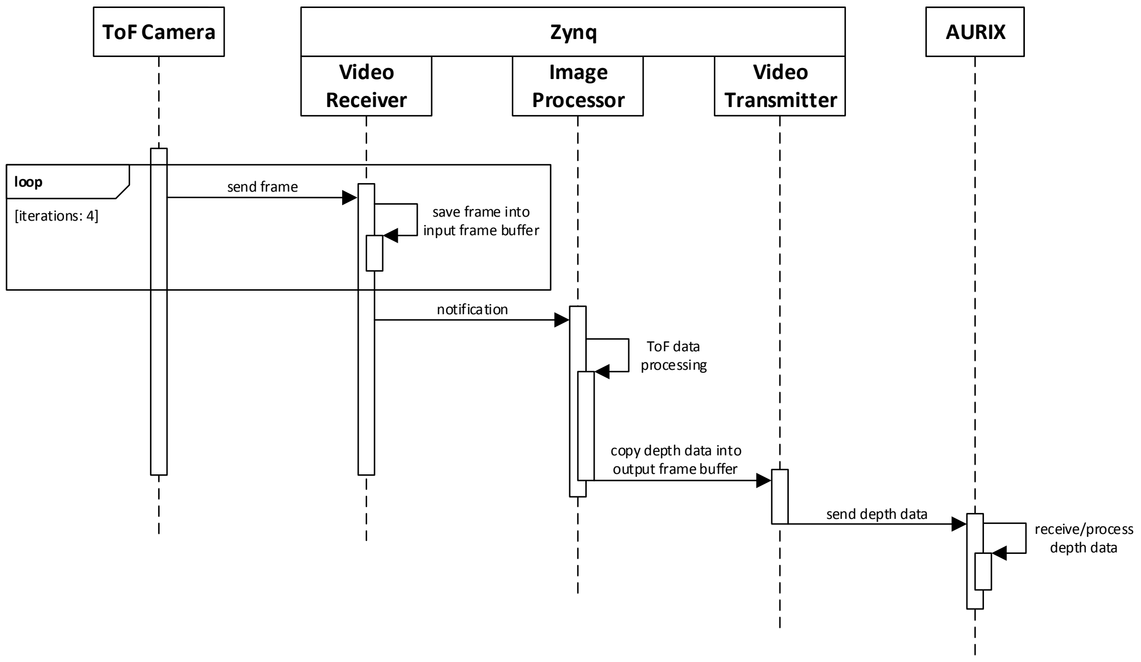


Figure 3.5: Sequence diagram of the architecture on the Zynq.

3.4 Interfaces

This section is focused on details of the camera system that affect the design. The parallel and the control interface of the system are presented.

3.4.1 Parallel Interface

The parallel interface (PIF) directly connects the camera evaluation board with the Zynq and the automotive platform with the Zynq via a multiline channel. From the sensor and the processing platform, twelve bits of data are transmitted in parallel. In addition to the data stream, three more signals are necessary to correctly send the rows of one raw frame pixel-wise. The start and end of a frame are marked by the vertical synchronization signal $Vsync$. The horizontal synchronization signal $Hsync$ defines the start and end of one row. The third signal is the pixel clock $ClkPix$. In one $ClkPix$ cycle, one pixel is transmitted.

The general timing of the PIF is shown in Figure 3.6. The frame blank is the elapsed time between the transmission of two frames, whereas the line blank is the elapsed time between two rows. It is possible to configure the PIF to delay the transmission of the first row by the vertical synchronization delay $thsd$. The horizontal synchronization delay tdd is the delay between start of a row and data transmission. The transmitter (camera system and Zynq) generates the pixel clock whereby the clock frequency can be configured. In addition, the polarity of the synchronization signals are configurable (active low or active high) and if the pixels are valid on rising or falling pixel clock edge.

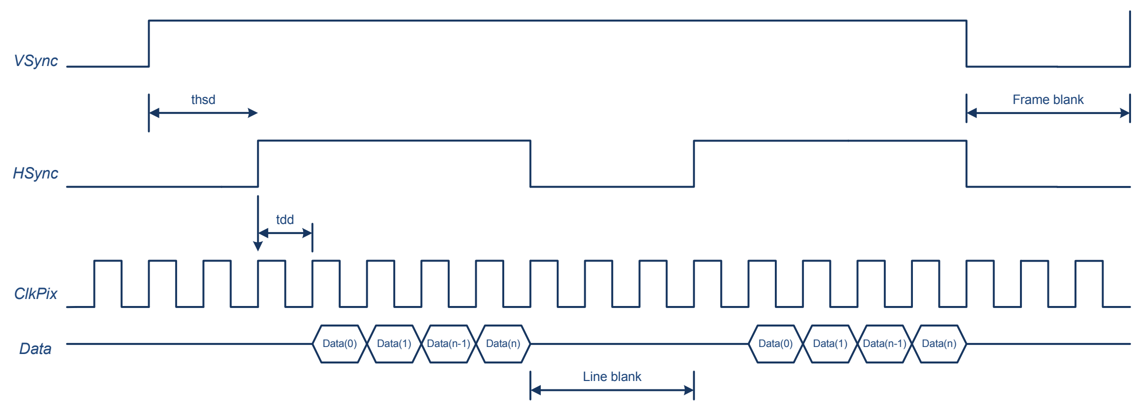


Figure 3.6: Parallel interface timing [Inf15b].

Image structure

The structure of the frames that are transmitted via the parallel interface is depicted in Figure 3.7. For the pre-processing use-case of the proposed image processing platform, an image resolution of 160 x 120 pixels is used. One row, also referenced as line, consists of 160 pixels. An image is built of 120 rows. Therefore, the complete number of pixels is 19,200 pixels. The amount of memory is equal to the determined one in the current system and is around 115 kBytes, as already shown in Equation 2.9. From the camera a pseudo row is transmitted with metadata, for example, the frame number. The placement of the pseudo row can be configured on the camera's parallel interface. It is possible to place the pseudo data as first row, last row or four LSB bits in the first row.

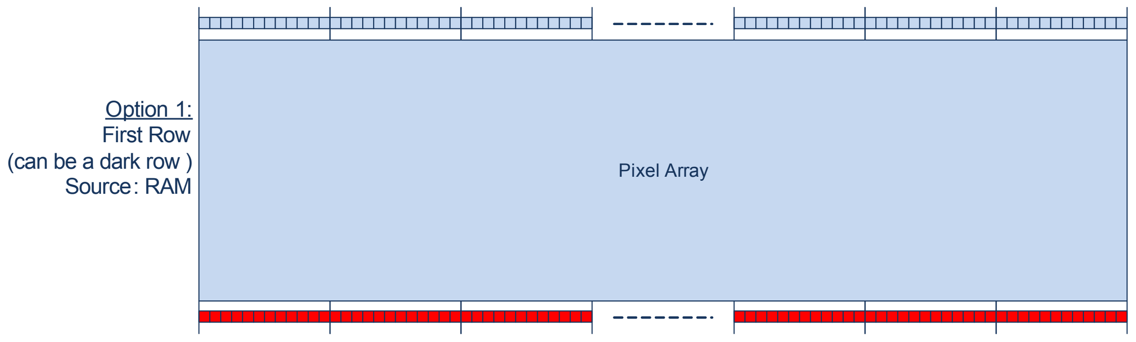


Figure 3.7: Structure of one frame [Inf15b].

3.4.2 Control Interface

For the control interface between AURIX, Zynq and ToF camera the standardized I2C interface is used. The camera works as a slave and provides an address to the Zynq. One I2C data packet consists of an address and a value. The address specifies the register where the value is written. On AURIX's side, the Zynq is the slave and the AURIX the master. The communication works the same.

In addition, the camera evaluation kit provides a reset line that is also connected to the Zynq. Hence, a camera system reset can be triggered.

3.5 Time-of-Flight Processing Pipeline

The goal of this section is to provide a clear understanding of the data processing used in the ToF algorithms. The pre-processing 4-Phases and 8-Phases algorithm, which are defined in the use-case requirements, are presented. The results of the algorithms are a depth image, an amplitude image and a 3D point cloud. Furthermore, the correction of the systematic errors with the help of calibration data is shown. For better understanding, the processing steps are clearly separated in this section, but for the implementation, the order can be changed or operations can be summarized to optimize the runtime. Further post-processing, such as filters, are not discussed.

3.5.1 4-Phases Algorithm

The typically approach to generate a depth image is a 4-Phases measurement, as outlined in [HLCH12]. Four raw frames are gathered from the camera, with a user configured phase-shift, between every frame, as illustrated in Figure 3.8. Typically an equivalent phase-shift of 90° is used, which results in measurements with the following phase-shifts: 0°, 90°, 180° and 270°. The modulation frequency and the illumination time are the same for all measurements.

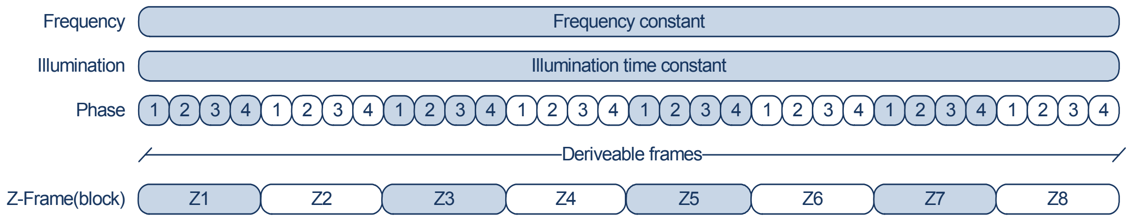


Figure 3.8: Frame sequence of the 4-Phases algorithm [Inf15b].

The basic data flow and calculation steps of the 4-Phases algorithm are illustrated in Figure 3.9. The order of the operations can be interchanged or combined to improve runtime or accuracy of the results.

Raw to Phase

The four raw frames $F_{0°}$, $F_{90°}$, $F_{180°}$, $F_{270°}$ provided from the configured camera system are used to generate the phase image. Thus, an equivalent phase-shift of 90° is used. In order to calculate the phase, an arc-tangent operation is performed with an imaginary and a real part as shown in Equation 3.1. For this step, two intermediate results that are the differences of two frames are computed. These intermediate results are also used in a subsequent processing step. The result is in the range $(-\pi, \pi]$. An addition of 2π is

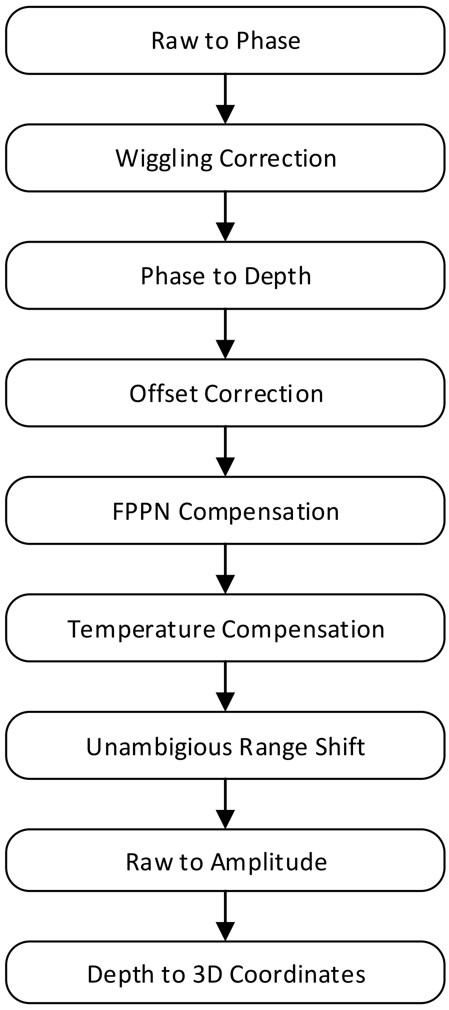


Figure 3.9: Calculation steps of the 4-Phases algorithm [PMD13a].

performed on negative results to map the range to $(0, 2\pi]$. It is not correct to map the phase values by adding only π to all values because positive values are then mapped to the range $(\pi, 2\pi]$ and negative values to $(0, \pi]$.

$$\varphi = \arctan\left(\frac{F_{270^\circ} - F_{90^\circ}}{F_{0^\circ} - F_{180^\circ}}\right) \tag{3.1}$$

Wiggling Correction

The origin of the wiggling error results in the signal modulation with imperfect sine waves. In the raw frames, non-linearity effects appear. By providing calibration data, this systematic error can be compensated as shown in Equation 3.2. Because the error is phase-dependent, the compensation must be calculated for every depth value. This sophisticated operation, which consists of sine wave calculations, can be simplified by using the pre-processed linearized values of a look-up table (LUT) saved in the memory. This strategy can decrease the processing requirements with a decrease of memory resources.

$$\varphi = \varphi + \varphi_{wiggling} \tag{3.2}$$

Phase to Depth

In order to obtain the distance values from the phase values, a multiplication with a constant factor is performed, as presented in Equation 3.3. The constant is based on the speed of light and the modulation frequency.

$$d = \varphi \cdot \frac{c}{4\pi f_{Mod}} \tag{3.3}$$

FPPN Correction

The FPPN error is pixel-dependent and results by manufacturing tolerances. It is provided by the calibration data. Therefore, a varying constant has to be added to every pixel to get a correct distance image, as illustrated in Equation 3.4.

$$d = d + d_{FPPN} \tag{3.4}$$

Offset Correction

The global offset, known from the calibration data, is compensated by adding the same value to all pixels of the distance image, as shown in Equation 3.5. It represents the mean error of every pixel's individual offset, which is referred to as global offset.

$$d = d + d_{offset} \tag{3.5}$$

Temperature Compensation

Adding the same constant error to all pixels, as shown in Equation 3.6, compensates for the temperature. The operation corrects different long-term and short-term effects and is calculated using the provided calibration data.

$$d = d + d_{temperature} \tag{3.6}$$

Unambiguous Range Shift

It is possible that after systematic error compensation, some distance values are outside of the unambiguous range. For instance, distance values can be below zero or can become higher than the unambiguous range. Therefore, the values are shifted into the unambiguous range by performing a modulo operation, as represented in Equation 3.7.

$$d = d \bmod \frac{c}{2f_{Mod}} \tag{3.7}$$

Raw to Amplitude

If only the illumination is active, an amplitude image is generated, which can be used to check the correctness of individual pixels of the distance image. A magnitude operation is performed on the imaginary and real part, already calculated for the raw-to-phase processing step, as presented in Equation 3.8.

$$A = \frac{\sqrt{(F_{270^\circ} - F_{90^\circ})^2 - (F_{0^\circ} + F_{180^\circ})^2}}{2} \tag{3.8}$$

Depth to 3D Coordinates

In order to get a 3D point cloud, the distance values that are polar coordinates are translated into Cartesian coordinates. Thus, the distance values are multiplied by so called direction parameters that are different for every pixel, as depicted in Equation 3.9. Those parameters are based on the direction vectors, which depend on the lens' distortion properties and the field of view, and are provided by the calibration data.

$$\begin{aligned} d_x &= d \cdot directions_x \\ d_y &= d \cdot directions_y \\ d_z &= d \cdot directions_z \end{aligned} \tag{3.9}$$

3.5.2 8-Phases Algorithm

The maximum unambiguous range is determined by the modulation frequency. In order to increase the unambiguity, it is possible to combine two distance images captured with two different modulation frequencies, as outlined in [JBP$^+$10]. Therefore, two sequent 4-Phases measurements with alternating frequencies are gathered, as shown in Figure 3.10. The illumination is constant during the whole measurement process. Due to the unambiguous range extension processing step, which computes the final distance image, the latency is increased.

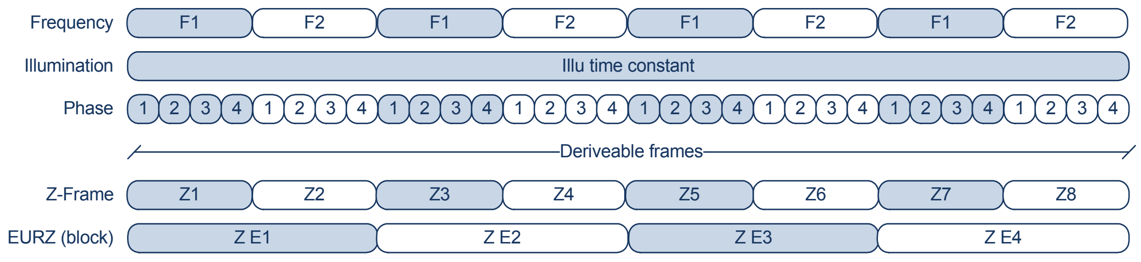


Figure 3.10: Frame sequence of the 8-Phases algorithm [Inf15b].

The data flow of the 8-Phases algorithm is depicted in Figure 3.11. Two already described 4-Phases data flows are performed and the two resulting distance images are combined in one computation step. The 3D point cloud is only calculated once after the unambiguous range extension and not for both images.

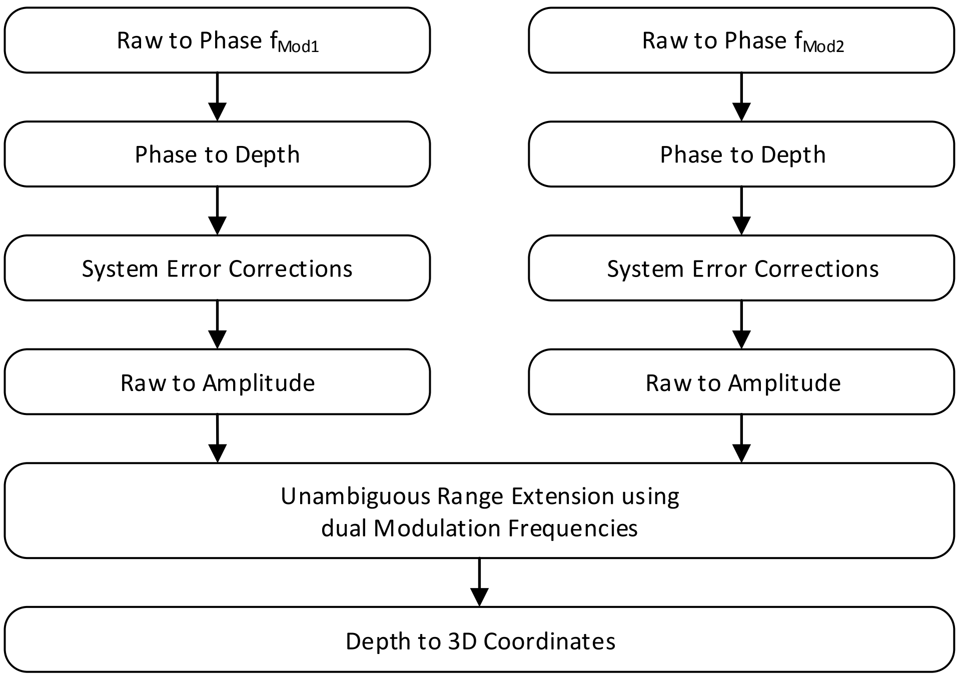


Figure 3.11: Calculation steps of the 8-Phases algorithm [PMD13a].

Unambiguous Range Extension

For this data processing step the equation for the unambiguous range extension, as shown in [JBP^{+}10], is similarly implemented. The basic equation uses basic arithmetic operations, modulo operations and round operations. Some required constants are calculated from the two modulation frequencies. Due to simplification, the formula is presented as function of the two distance images and the two modulation frequencies, as shown in Equation 3.10.

$$d = f(d_1, d_2, f_{Mod1}, f_{Mod2}) \tag{3.10}$$

3.5.3 Calibration Data

For correction of systematic errors, it is important to have a calibrated camera. The process of determining the camera's parameters is performed during calibration.

In the proposed ToF processing system the calibration data of the ToF camera system is already determined for Infineon's ToF evaluation kit. The data is saved in the board's memory and can be read with a software tool through the USB interface and used for further processing. In order to use the calibration data in the algorithm, the files have to be correctly interpreted which is described in the application note [PMD14].

For the following systematic errors the calibration is specified: wiggling, offset, FPPN and temperature. In addition, the direction parameters for the lens calibration used in the Cartesian coordinate conversion are provided.

3.6 High Level Synthesis

In order to fully achieve the requirement of a highly flexible system, the concept of High Level Synthesis, which is used for rapid prototyping in practice, is evaluated in the proposed hardware/software system. As mentioned in the related work (Section 2.2.2), HLS is already used in different use-cases for two-dimensional images. In this work, this approach is applied on hardware-accelerated ToF processing algorithms.

The main idea behind HLS is the raising of the design abstraction level especially in digital hardware development. Challenging use-cases make the process of creation of hardware components more time-intensive, resource-intensive and error-prone. Thus, HLS provides a technique to generate register-transfer level (RTL) hardware that is optimized for performance, power and area.

The basic creation flow is shown in Figure 3.12. An algorithm written in a high level language (e.g., C, C++, Matlab) is interpreted and automatically transformed into the register-transfer level abstraction level by providing a design in a hardware description language, for example, VHDL, Verilog. Such a generated hardware block can be integrated into an FPGA and allows the rapid generation of hardware-integrated blocks.

Test Cases

Two different test cases are designed for implementation to show the feasibility of High Level Synthesis. The results are compared with a corresponding implementation in software and the manually developed imaging processing unit. One simple test case with only one operation and a more complex one with several instructions are performed.

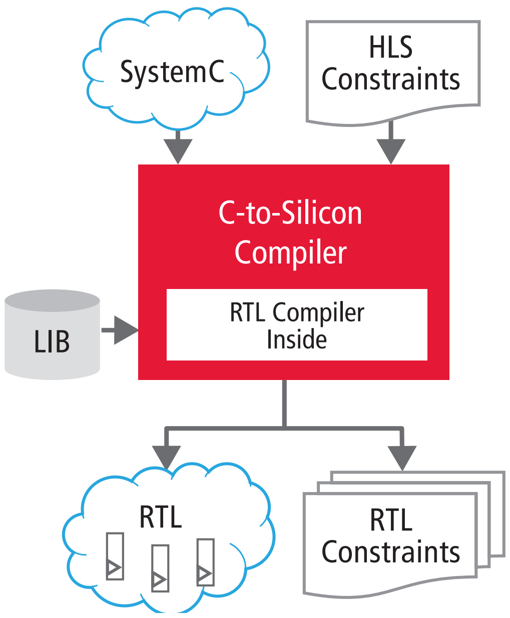


Figure 3.12: Basic concept of High Level Synthesis [Cad13].

This first test case represents a **pixel-by-pixel addition** by calculating the sum of two frames in one operation. The 16-Bit pixels are read from the extended memory, computed and pushed into the memory.

In order to provide a more complex test case than the previous one, the **amplitude calculation**, as shown in Equation 3.8, is calculated. Hence, four frames have to be used as input. Furthermore, the more complex root operation is used.

Chapter 4

Implementation

This chapter deals with the implementation of the whole system including hardware and software specifics, the used development workflow and environment, as well as the realization of the hardware-accelerated ToF processing algorithm.

4.1 Development

The goal of this section is to provide a detailed overview of the used development workflow and environment. Due to the chosen FPGA platform, a hardware/software co-design workflow developed the system. In addition, the software tools, which were used during development for the proposed platform, are shortly described.

4.1.1 Workflow

The life cycle, as depicted in Figure 4.1, that is used in particular for Zynq SoC systems is applied in the proposed framework. Beginning with the literature research of the basic principles and the state-of-the-art projects, the requirements of the system and the use-cases are defined. As a next step, the design of the system and the used algorithms regarding architecture, interaction and interfaces of the different components are determined. Based on the Zynq architecture, the system is partitioned into software and hardware. During implementation, the hardware system on Zynq's PL is developed using Xilinx's IP cores, manually developed IP Cores in HDL and generated IP cores with HLS. Simultaneously the software system on the ARM CPUs is implemented by configuring the hardware-integrated IP cores, and constructing SW algorithms. The hardware/software system is finally integrated by combining the compiled executable for the ARM and the generated bitstream for the FPGA on the Zynq SoC board.

Over the whole workflow, verification is performed between the different steps and measurements are interpreted to accomplish the system's requirements. Furthermore, the hardware/software components are concurrently developed on a demonstrator to show the feasibility.

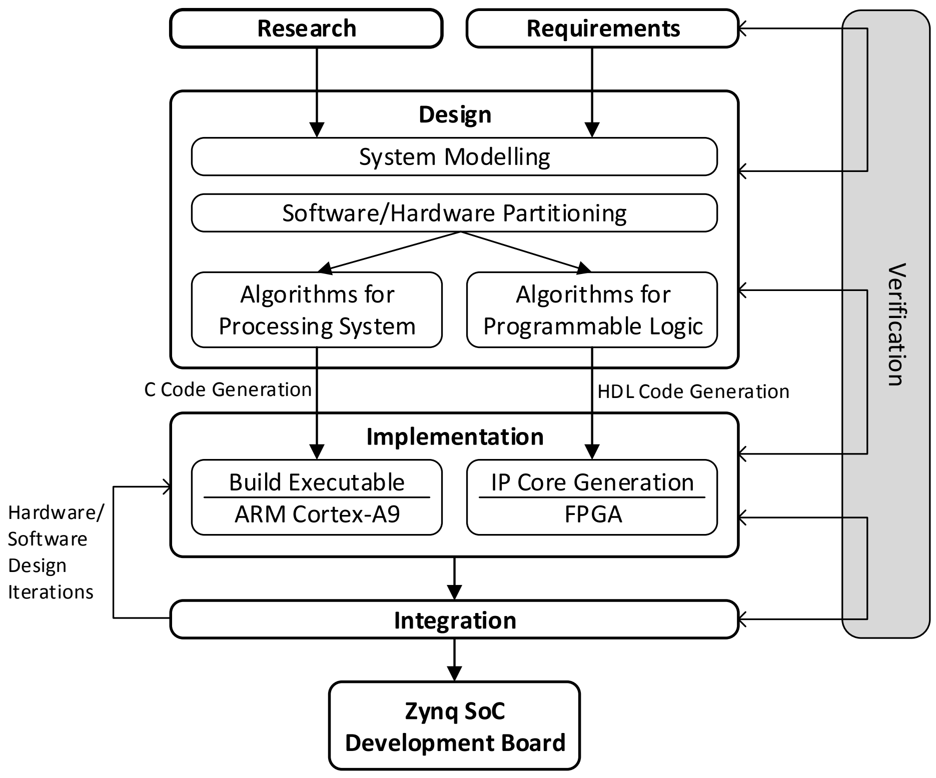


Figure 4.1: Development flow based on the Xilinx Zynq platform [Cig15] (with changes).

4.1.2 Environment

In order to develop the processing system, the following software programs are used. Based on the Zynq SoC platform, Xilinx's Vivado Design Suite is used, which provides lots of different tools for creation of a hardware/software framework, as outlined in [Xil15f]. The Vivado IP Integrator, the Xilinx SDK and the Vivado HLS tools are described in detail. The WebPACK edition for this design environment is freely available. Furthermore, tools for development of the demonstrator are presented.

Vivado IP Integrator

Vivado Intellectual Property (IP) Integrator, as shown in [Xil15h], is a tool to develop hardware designs for Xilinx's FPGAs. It provides a graphical design development flow to integrate and configure IPs that are manually developed with a hardware description language, generated with HLS or already created by Xilinx. Furthermore, the system is synthesized, analyzed and a bitstream for the Xilinx FPGAs is created.

This tool is used during implementation to create the ToF processing hardware system. Several IP cores from Xilinx are used and configured to meet the system's requirements. Such provided IPs are already verified and provide high flexibility. Thus, the development time can be significantly reduced. For debugging, logic cores can be used to monitor internal signals. The system is automatically synthesized and implemented. Then the placement and routing is performed with a final generation of the bitstream that is used for configuration of the FPGA.

Xilinx SDK

The Xilinx Software Development Kit (SDK) is an Eclipse-based IDE to develop C-based embedded applications for the ARM core, as represented in [Xil15i]. A project works with the hardware design that is created from Vivado IP Integrator. It provides library and drivers for Xilinx IP cores used in the hardware system. Furthermore, it allows debugging and profiling on the Zynq platform. It is used to program the ARM with the executable and the FPGA with the bitfile.

In this work the tool is used to create the ToF processing library that is a bare-metal C++ application that configures all IP cores in the hardware design and implements the use-case application. Furthermore, the Zynq is programmed with the software and hardware description, and via a terminal the output of the application is displayed during runtime.

Vivado HLS

Vivado High Level Synthesis, as described in [Xil15g], is a program based on the Eclipse IDE that directly transforms C, C++ and SystemC applications into IP cores without the manually creation of RTL code. With directives and constraints in the high-level language code, the produced hardware component can be configured regarding architecture, performance and area.

In order to evaluate this practical approach of rapid prototyping, Vivado HLS is used to develop an image processing unit that can be compared to the image processor. At first, the code is written in C++ and verified against a test bench. The code is then synthesized, and the synthesis results are evaluated. Different directives are used to optimize the generated IP core. Finally, the hardware component is exported and used in the Vivado IP Integrator.

Matlab

Matlab developed by MathWorks is a computing environment for numerical analysis, as shown in [Mat14]. It allows the processing of big amounts of data with vectorized operations. Scripts or functions can be used to develop algorithms. Data can be easily plotted and analyzed. Furthermore, it provides easy access to external interfaces, for example, Ethernet.

One application for Matlab in this work is the interpretation of memory dumps, transmitted with Xilinx SDK from the external memory of the Zynq. These frames are saved in a hexadecimal format and therefore, transformed into raw data provided by the camera system. The data is plotted and used for further processing. Furthermore, the network interface is accessed for receiving the data via UDP of the AURIX. The data is plotted, saved and processed to check the correct functionality of the demonstrator.

Visual Studio

For C++ development on a Windows PC, the integrated development environment Visual Studio, as presented in [Mic13], from Microsoft is used. It is used for software development for Windows programs, web applications and mobile apps. It supports different programming languages, such as C++, C#. Different Windows APIs are provided which allow an easy development of programs with graphical user interfaces.

The reference implementation is developed in C++, in Visual Studio. The test data is read from files, calculated and verified against the reference data. The advantage over software development on the ARM is the fast compilation and execution time. Furthermore, a Visual Studio application with a user interface is used to display a live-stream of the received image via UDP from the demonstrator. This functionality is also developed in Matlab but not used for a live-stream as at higher FPS, Matlab performs poorly.

Free TriCore Entry Tool Chain

Free TriCore Entry Tool Chain, as illustrated in [Tec15], is a tool chain for Infineon's automotive platform AURIX. It is an Eclipse-based IDE that allows the programming of the TriCore CPUs and the evaluation board of the automotive microcontroller.

In this project, the tool is used to develop the application for the AURIX. The interaction between the three CPUs, the Ethernet interface, the I2C interface and the CIF are configured in the IDE. Furthermore, the use-case application that is executed on the main core is deployed.

4.2 Time-of-Flight Processing Platform

The FPGA-based ToF processing platform consists of a ToF camera, an automotive computing platform and a Zynq development board. The components used in the overall system are described in the following paragraphs.

Time-of-Flight 3D Camera

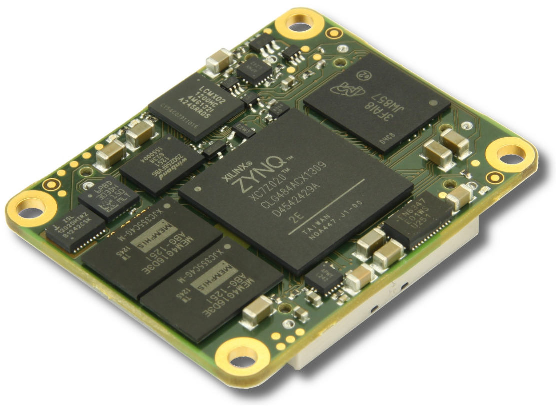
Infineon's ToF camera evaluation kit, as shown in [Inf13], is used in the implemented system. The sensor, which is developed from Infineon in cooperation with PMDTechnologies, comes with the highest on-chip integration on the market (2013). The evaluation board features different external interfaces, such as parallel interface and CSI-2, and allows access to important signals of the image sensor. Furthermore, the sensor unit and the illumination unit are exchangeable.

Automotive System-on-Chip

The automotive microcontroller AURIX, as shown in [Inf14], developed from Infineon is used as a starting point for a pre-processing or use-case application. The AURIX evaluation board is connected with the hardware-accelerated system, and provides three independent TriCore CPUs. It is compliant to various safety standards used in the automotive domain.

Zynq Development Board

The Zynq development board from Trenz Electronic is used in the system, as represented in [Tre15]. It consists of a Zynq-based SoC module, as shown in Figure 4.2a, and a connected carrier board with peripherals, as depicted in Figure 4.2b. On the module a Zynq 7020, as explained in [Xil15j], is placed with a Dual ARM Cortex-A9 MPCore with a clock frequency of up to 1 GHz as well as an FPGA with 85k logic cells and 560 KByte Block RAM. In addition a 256 MByte DDR3 SDRAM is available. The carrier board provides lots of interfaces (e.g., Ethernet, USB) and allows a direct connection to the FPGA through board connector pins.


(a) SoC module TE0720 [Tre15].

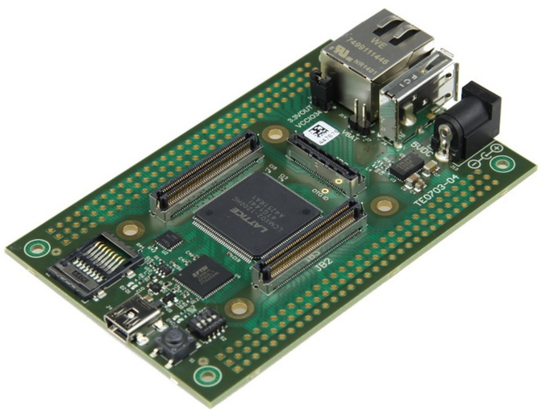

(b) Carrier board TE0703 [Tre14].

Figure 4.2: The used hardware from Trenz Electronics.

4.2.1 System

The implemented system and its sub-components are illustrated in Figure 4.3. The interfaces of the camera and the AURIX are connected with the board connector pins that are directly controlled by the FPGA. The pin allocation can be manually defined in the hardware design.

The control of the sub-components and the dataflow through the system is implemented on the ARM core. The programmable logic configures the hardware components. The video input stage receives the frames from the camera and pushes it to the memory. The image processor provides hardware-accelerated operations for the ToF processing pipeline. The video output stage reads the results from the memory and sends the data to the AURIX. The memory interface is an AXI interface, and is provided by the platform that connects the extended memory of the carrier board to the Zynq.

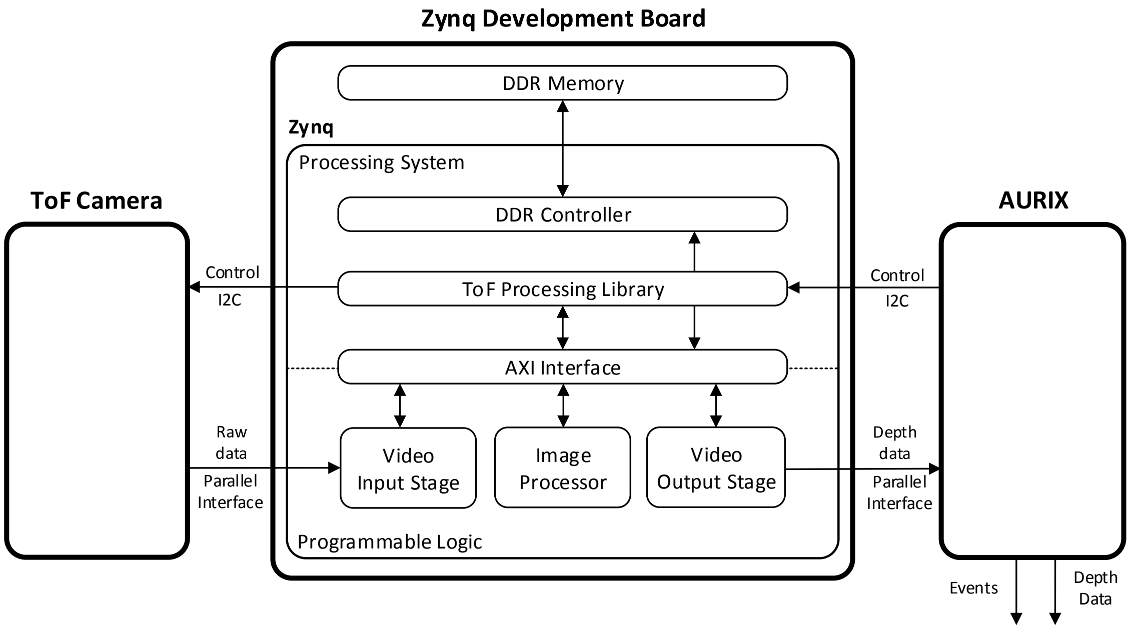


Figure 4.3: Implementation of the new system.

Interfaces

The control interface is connected with the I2C controller of the processing system. The reset line of the camera is connected through a GPIO port. Following configuration is used in this work for the data streams via the parallel sensor interface:

- Vertical Synchronization: The polarity of $Vsync$ is active high. The signal is high during the whole frame. The delay of $Vsync$ to $Hsync$ is zero pixels.
- Horizontal Synchronization: $Hsync$ is active high when data is transmitted. The signal is active during the transmission of one complete row. The delay between an active $Vsync$ and the first active $Hsync$ is one pixel clock cycle. If there is no delay, following problem will occur: the last row of a transmitted frame is the first row of the next frame. The receive logic requires time for a correct setup.
- Pixel clock: The pixel clock runs only during transmission of one frame and is set to the maximum clock frequency of 66.66 MHz. On the falling edge of a clock cycle, the data is valid.

4.2.2 Hardware Specifics

The focus of this section is to provide a high-level description of the hardware design. The main configuration details of the hardware components are presented. In order to develop a flexible framework, predefined IP cores are used. Xilinx provides several high-performance cores for video processing that are highly configurable in the processing system, as illustrated in [Xil14a]. Furthermore, the development time is shortened because the components are already verified. In the following paragraphs the connection to other cores and the configuration of the main IP cores are described.

Processing System

The hardware characteristics of the processing system are configured by the corresponding IP core in the hardware design, such as clock frequencies and activated interface controllers, as outlined in [Xil15e]. It is configured to enable an AXI HP port and an AXI GP port. All configurable IP cores are connected via an AXI Interconnect to the AXI GP port for configuration, as represented in [Xil15a]. For a high speed connection to the extended memory, the input and output stage, as well as the image processor, are connected to the AXI HP port via another AXI Interconnect. The memory controller and two I2C interfaces for the control interfaces are enabled. Interrupt signals provided by IP cores are directly connected to the PS. PL fabric clocks are used to generate a clock for the AXI GP slaves and to generate the pixel clock of the output stage. The used clock frequencies are listed in Table 4.1.

Table 4.1: Configured clock frequencies of the system.

Component	Clock Frequency [MHz]
CPU	666
DDR	533
AXI HP Ports	100
AXI GP Ports	100
PL Fabric (Pixel Clock)	66

Video Input Stage

The video input stage, as depicted in Figure 4.4, is responsible in receiving the raw data through the parallel interface from the camera system and in writing the corresponding data into the DDR memory. At first the output signals of the PIF are interpreted from the *Video In to AXI4-Stream* block and transformed into an AXI4-Stream. The stream is then forwarded to the *Test Pattern Generator (TPG)*. During a live data stream, the data is passed through this block without modifications. In addition, this block can also be configured to transmit a predefined pattern for tests. The *AXI4-Stream Subset Converter* adds a signal to the data stream, where it is necessary to have a fully specified AXI4-Stream. Finally, the write channel of the *AXI VDMA* unit generates an AXI4-Memory-Mapped signal set that pushes the data into the RAM.

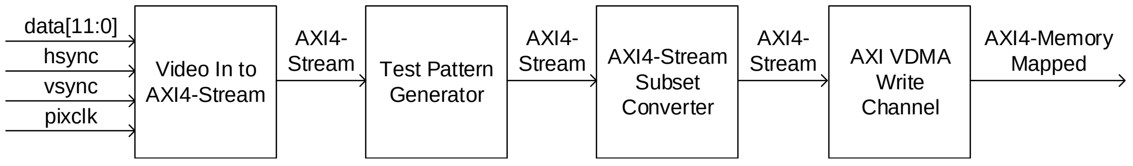


Figure 4.4: Basic architecture of the video input stage.

The **Video In to AXI4-Stream** core, as described in [Xil14d], is an interface between the parallel video input and the AXI4-Stream Video Protocol. The video source has to provide a pixel clock and synchronization signals (e.g., syncing and/or blanking signals). This allows easy connection of different video sources (e.g., DVI, HDMI, monochrome data) to a video processing pipeline that uses the AXI4-Stream interface. The following main configuration parameters are specified: one pixel per clock is transmitted from the video source. The component data width of the video source is 12-Bit. The video format Mono/Sensor is used. The core is connected to the PIF input signals (*data[11:0]*, *hsync*, *vsync*, *pixclk*). Additionally, it is important to tie the blanking signals to logical low and connect the *hsync* signal to the active video flag.

In order to test the subsequent hardware blocks without a transmitter, the optional **Test Pattern Generator** core, as shown in [Xil14e], is used to generate horizontal/vertical ramps, color or checkerboard patterns. In order to bring up and debug the complex *AXI VMDA* block, it is useful to not remove this component in the final design. In pass-through mode, the input signal is passed straight forward to the output. The main parameters result from the characteristics of the previous described core: the video component width is 12-Bit and the video input format is monochrome. Test pattern and image size can be configured in software.

The **AXI4-Stream Subset Converter** unit, as presented in [Xil15c], connects incompatible AXI4-Streams. All signals of the AXI4-Stream interface can be added or removed. The output signal set of the *Video In to AXI4-Stream* core does not include the *tkeep* signal (marked null byte), which is required for the *AXI VDMA* block. Therefore, this unit is configured to generate the *tkeep* signal that is tied to logical high.

In order to write the input AXI4-Stream into the memory, the **AXI VDMA** IP core, as shown in [Xil15b], is used. It provides high-speed direct memory access between the AXI4-Stream and the memory. Two-dimensional DMA operations are efficiently implemented, and an independent read and write channel is supported. The block is highly configurable through software, for example, after how many frames the CPU is notified through an interrupt. In this design, the write channel is used for the input stage. The following main configuration parameters are specified: the number of frame buffer locations is 20 and the address width is 32 Bits. The frame synchronization mode is set to *s2mm tuser*, which defines the start-of-frame (SOF) through the *tuser* signal of the AXI4-Stream.

Image Processor

The image processor is a hardware component that performs parts of the hardware-accelerated ToF processing algorithm and is provided by Infineon, as outlined in [Enc14].

Thus, it is an important component of the hardware/software platform that allows flexible exploration of different computation methods of a distance images. It provides ToF specific operations for image processing on the FPGA. In addition to the basic arithmetic operations, the co-processor also supports more complex ones, for instance, the square root or arcus tangent. The following operations are supported by the image processing unit:

- Pixel by pixel multiplication/addition/subtraction/division of two images
- Constant multiplication/addition/subtraction/division of an image
- Pixel by pixel limit check
- Pixel by pixel square root/arcus tangent (angle and magnitude)
- Variance/average of an image
- Variance/average per pixel over multiple images

The input data format of all images is represented in a fixed-point number format. The fixed-point number format is specified by the word length w and fraction length f. One pixel is stored in memory in the 16-Bit format, whereby one sign bit, zero integer bits and 15 fraction bits give the presentation. Two's complement is used to represent signed numbers. The image size can be more than 100k pixels.

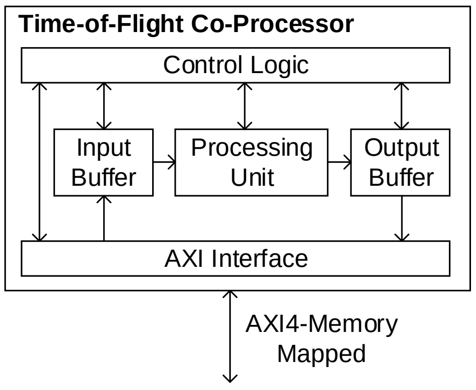


Figure 4.5: Baisc architecture of the image processor [Enc14].

The basic architecture of the co-processor is illustrated in Figure 4.5. At first the CPU sets up an instruction, which is then executed by the image processor. The component reads the images from the memory, processes them and writes the results back. Finally, the CPU is notified about the completion through an interrupt. The AXI interface interconnects the memory and the hardware-integrated block through which configuration and DDR memory access is possible. The input buffer is responsible for reading the images into a buffer using a receive logic. In addition, it provides the image streams for the processing unit that executes the instruction on one or two images. The results are loaded into the output buffer block that writes the resulting image to the memory. The control logic configures the state machines for the data flow of the different blocks based on the incoming operation.

Video Output Stage

After the image processing unit performs a successful computation, the results are transmitted via the FPGA's parallel interface to the AURIX. The video output stage, as shown in Figure 4.6, reads the results from the memory and provides the data, the synchronization signals and a pixel clock. The read channel of the *AXI VDMA* block, as illustrated in [Xil15b], gathers the resulting data of the RAM and generates an AXI4-Stream signal. The vertical and horizontal synchronization timing signals are provided by the *Video Timing Controller (VTC)* unit. A PL fabric clock, generated from the processing system, is used as pixel clock. With the *Clocking Wizard* core it is possible to change the pixel clock through software. Finally, the *AXI4-Stream to Video Out* block combines the AXI4-Stream and the timing signals to provide the PIF output signals.

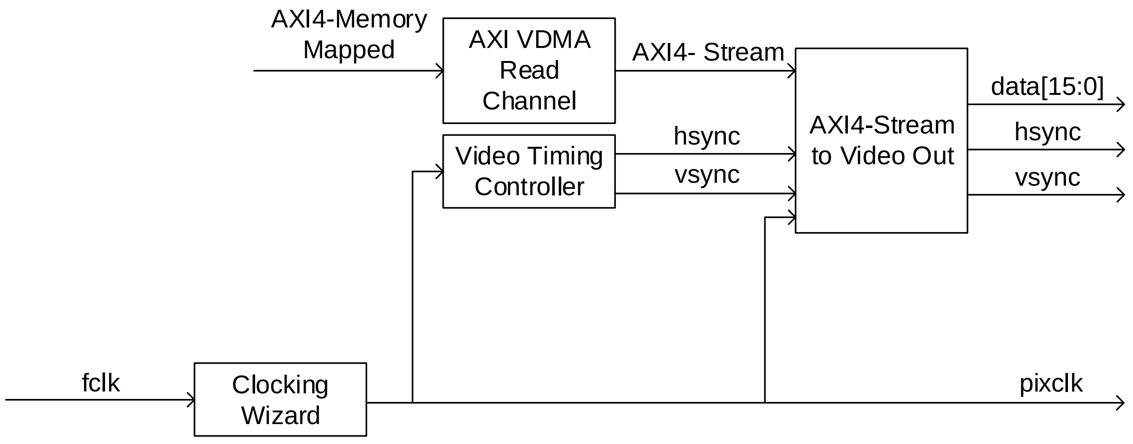


Figure 4.6: Basic architecture of the video output stage.

In order to read the data from the memory, the read channel of the **AXI VDMA** core, as shown in [Xil15b], is used. Because the write channel and the read channel of this core are independent, the same block is used for the input stage and the output stage. The main parameters are specified as following: the AXI4-Stream data widths and the size of the frame buffer are the same as for the write channel. For frame synchronization the free run mode is used, which waits for no external triggers and sends the data as fast as possible. Further behavior is configured through software.

With the **Video Timing Controller** core, as described in [Xil13], it is possible to detect or generate video timing signals. It is commonly used with the *Video In to AXI4-Stream* or *AXI4-Stream to Video Out* IP core. Lots of different combinations of synchronization and blanking signals are possible and the polarity of the signals is configurable. The maximum supported frame size is 8192 x 8192 pixels. In this design, the core is used as signal generator and is configured through software. The pixel clock is connected as input to this bock.

The **Clocking Wizard** unit, as represented in [Xil15d], provides methods to flexibly change the parameters of an active input clock, such as clock frequency, phase shift. In this work, the core is used to generate a dynamic reconfigurable pixel clock through software. The input clock is a PL fabric clock with 66.66 MHz from the processing system.

The output clock is used in the FPGA's parallel interface and as input for the *VTC* and *AXI4-Stream to Video Out* core. The division factor of the clock frequency can be changed through software.

After creation of all necessary signals for the output parallel interface, the data AXI4-Stream and synchronization signals are brought together by the **AXI4-Stream to Video Out** IP core, as can be seen in [Xil14c]. This core has the analog behavior compared to the *Video In to AXI4-Stream* block by providing an interface between the AXI4-Stream interface and a parallel video output. The following main configuration parameters are specified: one pixel per clock cycle is transmitted and the video format is Mono/Sensor. The timing mode is specified as master because a *VDMA* is used in the video processing pipeline. Thus, it automatically synchronizes the read channel of the *VDMA* to the *VTC* timing signals by applying back-pressure to the AXI4-Stream. The pixel clock is needed as input for this core. Due the fact that the image processor provides 16-Bit results, the output data width is also 16-Bit.

4.2.3 Time-of-Flight Processing Library

The Time-of-Flight processing library is the main part of the hardware/software system. It connects the hardware components of the programmable logic to the processing system. The library is executed on the ARM CPU as bare-metal application. Furthermore, it implements the link between use-case application and the camera as well as the AURIX. It is designed to be flexible for further use-case applications to fulfill the requirements. Hence, the hardware-integrated components for the input and output stage are configured in software for the required use-case. The image processor provides hardware-accelerated operations, which are used to implement the required two ToF algorithms. In addition, an interrupt system is activated to react on triggered events from the FPGA.

The class diagram of the ToF processing library is depicted in Figure 4.7. The main class *ToFProcessingLibrary* implements the link between communication on the interfaces and the ToF algorithm. The class *ToFAlgorithm* performs the hardware-accelerated use-case application. In addition, it provides methods to verify the hardware-integrated algorithm to the reference implementation. Class *I2CCommuncation* provides methods for the control interface to configure and control the camera, as well as receive commands from the AURIX. The class *PIFCommunication* is responsible to set up the input and output stage. For every IP core that is configurable via software, an own class exists which adjusts the behavior regarding the use-case with the low-level drivers provided by Xilinx or Infineon. No methods are displayed because each core has almost the same: *reset()*, *start()*, *stop()*, *configure()*, *setupInterruptSystem()*. In the following paragraphs all classes are shortly presented in the bottom-up approach.

The class **VDMA** configures the central part of the receive and transmit logic. For the input stage the write channel, and for the output stage the read channel are used. It is important to set the correct image resolution. The write channel is configured to run in circular buffer mode. Frames received through the parallel interface are written into the input frame buffer on the extended memory. The start address of the frame buffer can be set in software. An interrupt is thrown every four raw frames to notify the ARM core by

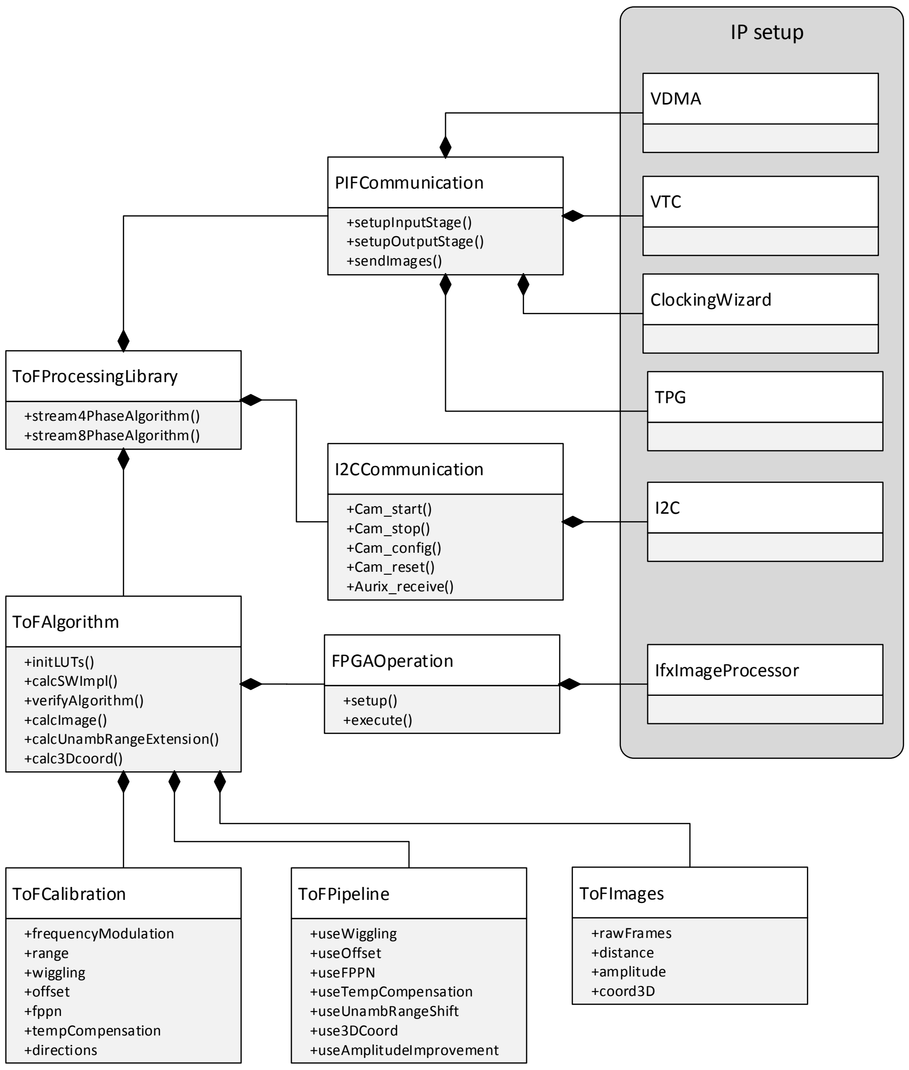


Figure 4.7: Class diagram of the ToF processing library.

setting a flag. The circular mode for the read channel is disabled. After the given number of frames are sent from the output frame buffer, the VDMA stops the transmission. This allows more flexibility than the circular mode because otherwise, frame synchronization between write and read channel must be considered.

The Video Timing Controller is configured from the class **VTC**. The synchronization signals are set for the output stage depending on the frame resolution. The polarity and delays are described in detail in Section 4.2.1.

The class **ClockingWizard** configures the Clocking Wizard IP core that provides the pixel clock for the output stage. A divisor can be configured during runtime to change the frequency.

The Test Patter Generator is configured using the class **TPG**. The image resolution and mode can be set. For testing, a predefined pattern is used, for example a check board pattern, and for the running system the pass-through mode is activated.

The class **I2C** sets the frequency and slave address of the I2C interfaces. Read and write methods are provided from the Xilinx driver.

The image processor hardware-component is configured in software by the class **Ifx-ImageProcessor**. The configuration registers and the interrupt system are set up.

The class **PIFCommunication** implements the link between the cores of the input and output stage. The methods from the corresponding core classes are used to configure and start the independent read and write channels. The method *sendImages()* starts the read channel to transmit frames from the output frame buffer.

I2C commands are sent with predefined values with the class **I2CCommunication** to the camera. In addition, it is used to read I2C messages from the AURIX.

The class **FPGAOperation** provides an easy setup of the image processor operations. With one function call the instruction can be set up and executed.

The use-case applications are implemented in the class **TofAlgorithm**. It provides methods to initialize the look-up tables and to run the software implementation of the reference implementation. The methods *calcImage()*, *calcUnambRangeExtension()* and *calc3Dcoord()* perform the algorithm with hardware-integrated operations on the FPGA. These three methods can be verified with the method *verifiyAlgorithm()*, which checks the results against the reference implementation results. The data is saved in objects of separate classes. The class **TofImages** represents the captured raw frames as well as the results (e.g., distance, amplitude, 3D point cloud). An object of the class **ToFCalibration** is used to save the frequency and the range as well as the calibration values to compensate systematic errors. Which operations are performed from the ToF Pipeline can be configured with the class **ToFPipeline**. A boolean flag can be set for every processing step.

The combination of all parts is implemented in the class **ToFProcessingLibrary**. The two use-cases can be run in a video stream by gathering the data from the camera via the input stage, performing the algorithm and sending the images to the AURIX via the output stage. This is achieved by executing an endless loop which polls the interrupt flag of the VDMA write channel. If the flag is true, the addresses of the current four frames in the frame buffer are calculated and the methods from the class *TofAlorithm* depending on the use-case are performed. Afterwards, the images are sent via the class *PIFCommunication.* Finally, the flag is set to false. The loop stops if the AURIX sends the corresponding I2C command.

The ToF processing library allows a high degree of flexibility and re-usability. Further use-case algorithms can be easily added by using hardware-integrated operations. The interfaces can be changed, e.g., using the network interface as output stage. Furthermore, the frame resolution is completely configured through software.

4.3 Time-of-Flight Processing Algorithm

In this section the implementation of the ToF processing pipeline, as described in Section 3.5, and the used workflow are presented.

4.3.1 Workflow

The workflow used to develop the ToF algorithm is illustrated in Figure 4.8. Several different implementations of the algorithm are generated to allow a fast and easy verification of the final results.

Starting with the deployment of a reference implementation in C++ on a host PC, the ground-truth prototype is created. The test data and the reference implementation are then ported on the Zynq's ARM CPU to verify the results of the hardware-accelerated algorithm. This step can be easily done and shows the advantage of a hardware/software system. The development of the reference implementation is done on PC because it allows faster compilation and execution of the application.

Next, a hardware-related software implementation is developed to test the intermediate results of the hardware algorithm. Specific characteristics of the image processor's operations have to be kept in mind. The order of operations can be changed and instructions can be summarized to improve the algorithm in terms of performance and accuracy. This implementation is then developed with image processor operations on the FPGA. These two steps are performed until the timing and accuracy requirements for the application are met. During the whole process, the results are verified with the previous implementations.

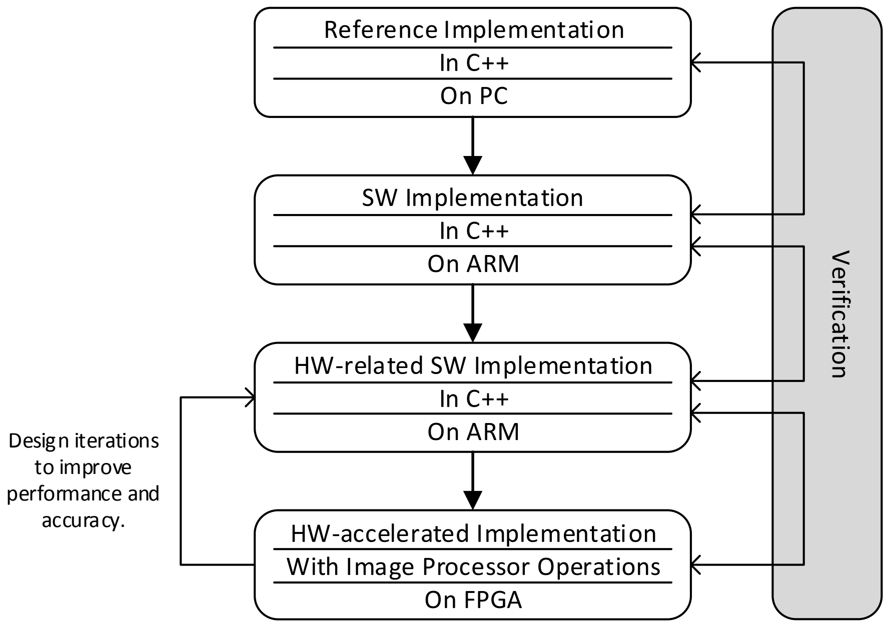

Reference Implementation
In C++
On PC
SW Implementation
In C++
On ARM
HW-related SW Implementation
In C++
On ARM
HW-accelerated Implementation
With Image Processor Operations
On FPGA
Design iterations to improve performance and accuracy.
Verification

Figure 4.8: Development flow for the implemented ToF algorithm.

4.3.2 Implementation

The used arithmetic operations, which perform the calculation of the depth data and the compensation of the systematic errors, are shown in detail in the following paragraphs.

Reference Algorithm

The reference algorithm, as presented in Algorithm 1, implements the 4-Phases use-case in software. This implementation is developed with knowledge of application notes ([PMD13a], [PMD13b], [PMD14]) and literature ([Lan00], [HLCH12], [FAT11], [JBP$^+$10]).

As a starting point, the real and imaginary part of the raw data are computed, as shown in Lines 2 and 3. Next, the arcus tangent is calculated, as can be seen in Line 4. The resulting phase values are in the value range of $(-\pi, \pi]$ and mapped to $(0, 2\pi]$ by adding 2π to negative results, as represented in Line 6. If π is added to the initial phase values, this will map the positive values to a range of $(\pi, 2\pi]$ and the negative values to $(0, \pi]$, which is a completely different result.

The calibration data to compensate the wiggling error, which depends on the distance, is saved in a look-up table. By multiplying the phase with a factor, the index of the LUT is calculated, as can be seen in Line 8. For the distance, the phase values are multiplied with the range factor, as illustrated in Line 9. As represented in Line 10, the remaining error corrections are only additions of positive or negative values. For the modulo operation of the unambiguous range shift, the fraction part is calculated with the floor function and subtracted from the distance, as shown in Line 11. The amplitude and 3D point cloud are straight forward computed, which is evident in Lines 12 to Line 15.

Algorithm 1 Reference implementation.

```
 1: $range = \frac{c}{2f_{Mod}}$
 2: $imag = F_{270°} - F_{90°}$
 3: $real = F_{0°} - F_{180°}$
 4: $\varphi = \arctan(imag, real)\ (-\pi, \pi]$
 5: **if** $\varphi < 0$ **then**
 6:     $\varphi = \varphi + 2\pi$
 7: **end if**
 8: $\varphi = \varphi + WigglingLUT\left[\left\lfloor \varphi \cdot \frac{N-1}{2\pi} \right\rfloor\right]$
 9: $d = \varphi \cdot \frac{range}{2\pi}$
10: $d = d + d_{offset} + d_{FPPN} + d_{temperature}$
11: $d = d - \left\lfloor \frac{d}{range} \right\rfloor \cdot range$
12: $A = \frac{\sqrt{real^2 + imag^2}}{2}$
13: $d_x = d \cdot directions_x$
14: $d_y = d \cdot directions_y$
15: $d_z = d \cdot directions_z$
```

Hardware-accelerated Algorithm

The hardware-accelerated algorithm, as shown in Algorithm 2, efficiently implements the reference algorithm. The focus of this algorithm is the improvement of the software algorithm regarding performance and accuracy, done by changing the order of operations and summarizing instructions. Furthermore, specific features of the image processor operations have to be considered. The hardware-related software algorithm is the same and used to provide correct intermediate results to verify the hardware implementation. The reference algorithm is used as a starting point. Over several design iterations following improvements can be made:

- **Arctan operation**

  The value range of the image processor's arctangent operation is $(-1, 1)$ instead of $(-\pi, \pi)$, as illustrated in Line 4. Instead of a multiplication with π, the advantage of a smaller value range for subsequent operations is used. For correct application, all given values for systematic error compensation must be divided by π.

- **Range multiplication**

  The distance is the multiplication of the phase with the range constant. By performing this operation at the end of the distance calculation and not after the wiggling compensation, as can be seen in Line 11, the value range of the intermediate results is smaller: $(0, 1]$ instead of $(0, range]$. This can be applied because the calibration data for the systematic errors are divided by the range constant during initialization of the algorithm. Hence, the results are more precise because of the smaller value range. Due to the fixed-point number representation, a smaller value range decreases the number of bits used to specify the integer part. Thus, the fraction length is increased. In addition, fewer operations are used because for the unambitious range shift, no division and multiplication operations with the range constant are necessary.

- **No Wiggling Compensation**

  The calibration values to compensate the wiggling error are determined with the help of a LUT, as represented in Line 8. Due to the fact that the image processor provides no index operation, the index look-up is made in software on the ARM core, as shown in Listing 4.1. The index is calculated with hardware-accelerated operations. In some use-cases, such as gesture recognition, and in newer camera generations, the wiggling compensation is not necessarily needed. Therefore, an implementation without wiggling compensation exists to speed up the performance.

- **Arctan Mapping**

  The resulting phase values of the image processor's arctangent operation are in the value range of $(-1, 1]$. Adding 2 to the negative values, map the result to the value range of $(0, 2]$. The improvement is to compute the if statement for negative phases only if wiggling compensation is used, which is evident in Line 5 to Line 7. This can be applied because the unambiguous range shift, as shown in Line 10, is also mapping the negative results into the correct value region, whereas for the wiggling index calculation the correct phase is required.

- **No Amplitude Shifts**

  The 12-Bit raw data defines a maximum possible real and imaginary part of 4,096. If these values are inserted into the amplitude calculation, as shown in Line 12, the maximum possible amplitude value will be 2,895.6. This value can be represented by a 16-Bit fixed-point number with the format (Q12.3). The fixed-point format of the result of the image processor's magnitude operation is the same as the fixed-point format of the input values. Therefore, the real and imaginary parts have to be shifted from (Q15.0) to (Q12.3). For most of the use-cases, it is not necessary to use precise amplitude values. Hence, an implementation exists that skips these two shift operations to improve performance.

Algorithm 2 Hardware-accelerated implementation.

```
1: $range = \frac{c}{2f_{Mod}}$
2: $imag = F_{270^\circ} - F_{90^\circ}$
3: $real = F_{0^\circ} - F_{180^\circ}$
4: $\varphi = \arctan(imag, real)\ (-1, 1]$
5: if $\varphi < 0$ then
6:     $\varphi_{Wigg} = \varphi + 2$
7: end if
8: $\varphi = \frac{\varphi}{2} + \frac{WigglingLUT[\lfloor \varphi_{Wigg} \cdot \frac{N-1}{2} \rfloor]}{2\pi}$
9: $\varphi = \varphi + \frac{d_{offset} + d_{FPPN} + d_{temperature}}{range}$
10: $\varphi = \varphi - \lfloor \varphi \rfloor$
11: $d = \varphi \cdot range$
12: $A = \frac{\sqrt{real^2 + imag^2}}{2}$
13: $d_x = d \cdot directions_x$
14: $d_y = d \cdot directions_y$
15: $d_z = d \cdot directions_z$
```

Listing 4.1: Wiggling compensation in software.

```
for( i=0; i<numPixels; i++ )
{
  phase[i] = phase[i] + wiggLUT[wiggIdx[i]];
}
```

4.4 High Level Synthesis

Xilinx provides a complete framework for High Level Synthesis with an easy integration of generated IP Cores into existing Vivado designs. Vivado HLS allows the addition of directives and constraints to high-level language code to specify how the program is synthesized, for instance, memory interfaces, loop unrolling or pipelining.

4.4.1 Workflow

The workflow for Xilinx's HLS framework is shown in Figure 4.9. Xilinx provides the IDE Vivado HLS for developing HLS-based hardware components.

At first a project is created that consists of a test bench and a top function that should be hardware-integrated. Following high-level languages (HLL) are supported: C, C++ or System C. Next, the desired functionality is developed in software by keeping in mind that not all HLL constructs can be used, e.g., dynamic memory allocation. The program is compiled and verified against the test bench. As a next step, the synthesis is started that results in a synthesis report including timing and area estimations. By including directives and constraints in the top function (e.g., loop unrolling, memory interface), different solutions can be synthesized and compared. After that, the generated RTL code is verified against the test bench by running an RTL simulation. If everything is correct, the code is exported in HDL language and can be used in Vivado design as IP core.

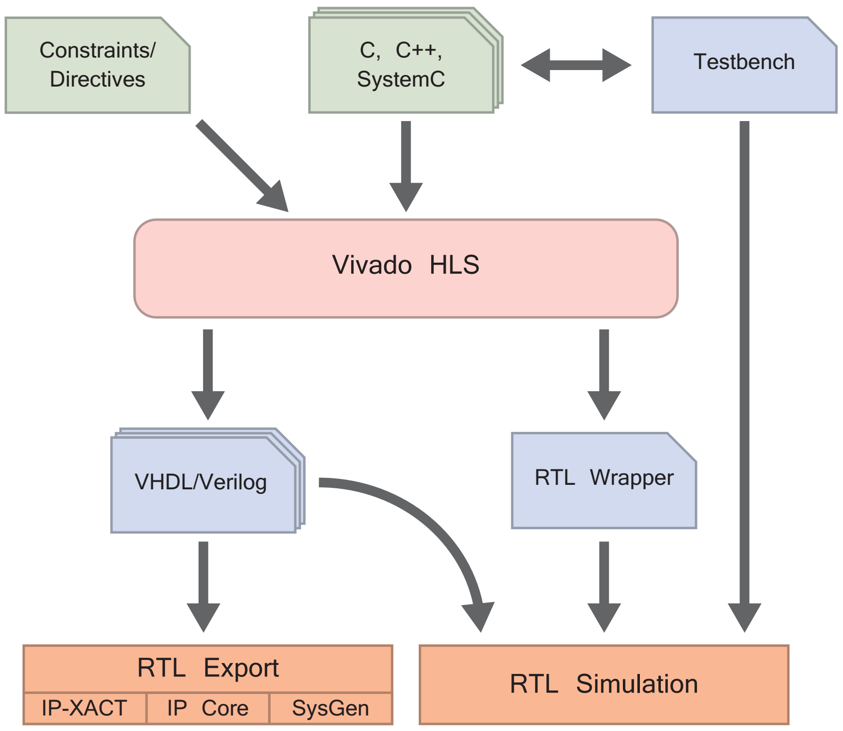


Figure 4.9: Basic concept of High Level Synthesis [CEES14] (based on [Xil11b]).

4.4.2 Implementation

In order to rebuild the image processing unit, the memory interface is very important because high amounts of data must be processed. The following directives, as described in [Xil14b], are used: in Vivado HLS framework it is possible to use an AXI High Performance port as the memory interface. The frame is sequentially calculated by reading a data burst from the memory, computing it and saving it to the memory. The processing unit, which implements the functionality of the test case, is pipelined.

Therefore, two different dataflow approaches for two test-cases are implemented. The first test-case that is implemented is shown in the Algorithm 3. It is a simple pixel-by-pixel addition of two 16-Bit per pixel frames. In Algorithm 4 the second test-case is shown. It calculates the amplitude of four input frames. The magnitude is calculated by using two multiplications for the square, an addition and a root operation.

Algorithm 3 Pixelwise addition.

```
1: Fout = ImgProc_Add( F1, F2 );
```

Algorithm 4 Amplitude calculation.

```
1: imag = ImgProc_Sub( F3, F1 );
2: real = ImgProc_Sub( F0, F2 );
3: mag = ImgProc_Magnitude( imag, real );
4: amplitude = ImgProc_Div( mag, 2 );
```

Dataflow solutions

The frames of both algorithms are saved in the extended memory. The HLS hardware component reads the data in bursts from the memory, performs the computation on one data burst at the time, and saves the results in bursts into the memory. During implementation, two solutions for efficient memory operations were encountered.

The first approach is the straight-forward implementation by running a loop with following three operations: reading, computing and saving the data. It sequentially works on the data burst by reading the next burst after the previous one is completely saved in memory, as shown in Figure 4.10. It can be observed that no pipeline architecture is used.

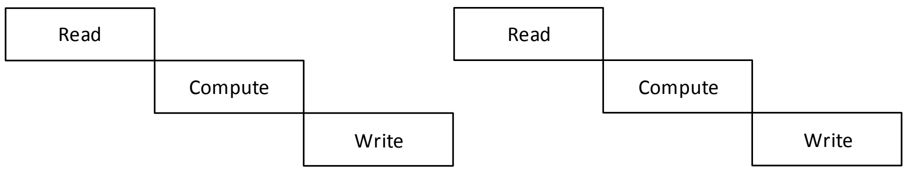


Figure 4.10: Architecture of the multi-burst approach.

The second strategy is based on a pipeline, as depicted in Figure 4.11. A directive is used that streams the data into a buffer. During computation the next data burst is read from the memory and the already previous calculated burst is saved into memory. As a result, the throughput is higher. Therefore, independent read and write access of the memory is necessary.

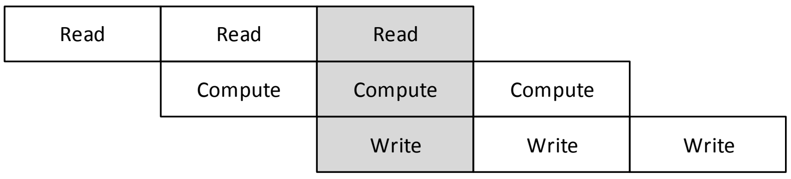


Figure 4.11: Architecture of the pipeline approach.

4.5 Verification

Verification is an important part to check if the system's functionality fulfills the initial requirements. The different sub-components of the implemented hardware/software platform and the transmission of data through the interfaces have to be tested to accomplish a correct result. The components, listed below, are verified in the following way:

- **Video Input Stage**

  From the video input stage module the VDMA core and the memory interface are tested without the ToF camera by using the Test Pattern Generator. A predefined pattern, such as a horizontal pattern, is created and transferred to memory. By comparing the memory and the predefined pattern, the correct functionality of the VDMA is verified. Using the pass-through mode of the TPG with the connected ToF camera, the receive logic is tested. Because it is not possible to know the correct values of the transmitted pixels, only the pseudo row is checked. This row is predefined and always the same. By running a live- stream and checking this row in real-time, the correct transmission functionality is verified.

- **ToF Processing Algorithm**

  The verification of this component is also explained in detail in Section 4.3.1. The hardware/software platform Zynq allows an easy comparison of software and hardware results. A hardware-related software implementation is used to check all intermediate results of the hardware implementation. These results are then verified with the reference algorithm implemented on the ARM core.

- **Video Output Stage**

  The video output stage component is verified by sending a predefined test pattern, which is saved in RAM. On the AURIX, the received test pattern can be compared to the specified one. Hence, successful verification can be performed. Furthermore, predefined images are saved into extended memory, sent to AURIX and forwarded over Ethernet to a PC. The predefined image is compared with the received image in Matlab.

In order to check the whole integrated system, the calculated results, as well as the raw data, are all sent to AURIX and to the PC. On the PC, the raw data is computed with the reference implementation and compared to the results from the processing platform. Hence, the successful verification of the framework can be carried out.

Chapter 5

Results

This chapter deals with the evaluation of the implemented ToF processing platform. First of all, the implementation results, including utilization and throughput, are analyzed. Furthermore, the two implemented pre-processing applications regarding performance are discussed. The accuracy of the ToF computation using hardware-accelerated fixed-point operations is compared in detail with the software implementation. In addition, the demonstrator and its interfaces are evaluated. Finally, the results of the two developed hardware-integrated components, using Vivado's High Level Synthesis toolchain, are presented.

5.1 Implementation

In this section the implementation results are discussed. For synthesis and implementation of the hardware design, the software tool Xilinx Vivado 2015.1 is used. The target system for creation of the FPGA hardware bitstream is the Xilinx Zynq 7020 SoC platform.

5.1.1 Utilization

In Table 5.1 the different used and available hardware resources are shown in detail. Xilinx Vivado creates utilization reports after every step generation of the hardware design. The post-implementation results are presented. These results are optimized and represent the real numbers contrary to the post-synthesize ones.

The actual number of used flip-flops (FF) is 19,493 whereby 106,400 FFs are available, this is equivalent to 18.2%. 28.1% of the look-up tables (LUT) are used which results in 14,991 LUTs out of 53,200 available LUTs. 13 block RAM (BRAM) units out of the available 140 are consumed. Digital signal processor (DSP) slices utilize around 6% of the complete available slices, which corresponds to 13 DSP slices. In order to fully complete the utilization listing, the numbers of following hardware resources are shown: memory look-up tables, input/output (I/O) interfaces, global clock buffers (BUFCG) and mixed-mode clock manager (MMCM) modules.

Table 5.1: FPFA utilization of used resources.

Resource	Utilization	Available	Utilization %
FF	19,493	106,400	18.2
LUT	14,991	53,200	28.12
Memory LUT	647	17,400	3.72
I/O	43	200	21.50
BRAM	13	140	9.29
DSP	13	220	5.91
BUFG	5	32	15.62
MMCM	1	4	25

The percentage distribution of used and available resources is graphical presented in Figure 5.1. It is clearly obvious that less than one third of the disposable resources from the FPGA are used for the implemented hardware design. Thus, lots of resources are available for further use-case applications or post-processing, such as filters. Performance and memory intensive programs on the AURIX, such as gesture recognition, can be transferred to the Zynq platform for a speed-up of the overall system .

The performance can also be increased by using more hardware components at the same time. It is possible to initiate the image processing unit several times to speed up the computation by using hardware-accelerated operations in parallel. Due to no data dependency between pixels, different parts of one image can be simultaneously computed. On top of that, several video input and output stages can be instantiated for implementing further ToF cameras in the system by using additional connector pins of the Zynq development board.

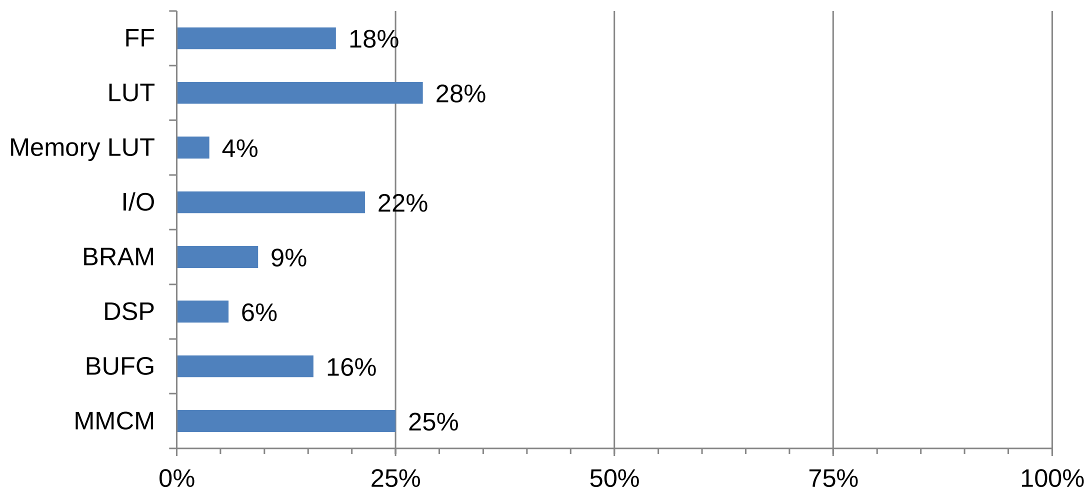


Figure 5.1: Percentage utilization of the resources.

The utilized area of each hardware component is illustrated in detail in Table 5.2. Shown are the four important hardware resource types BRAM, DSP, FF and LUT. The percentage distribution of the hardware components regarding used look-up tables is graphically depicted in Figure 5.2.

The biggest components are the already described AXI VMDA IP core and the image processor provided by Infineon as well as the AXI Interconnect hardware module that connects the AXI memory-mapped master or slaves to the processing system. Each of the three units needs about one quarter of the complete system. The Test Pattern Generator IP core, which utilizes around 17% of the area, is optional and can be removed because it is only used to test the AXI VDMA of the input stage. In the running system the TPG is configured in pass-through mode.

Components of the video output stage including the Video Timing Controller and Clocking Wizard allocate around 8% of available LUTs. Further IP cores of the receive and transmit logic utilize very little resources, such as the Video In to AXI4-Stream IP, the AXI4-Stream to Video Out core and the AXI4-Stream Subset Converter unit. System specific components (e.g., processing system, processing system reset) need hardly any resources. Almost all of the used BRAM memory is used by the image processor due to the input and output buffering of the pixels. The DSP slices are nearly equally separated into the image processor and the Test Pattern Generator.

Due to the use of already developed IP cores, no further improving of area consumption is possible. The IP cores are synthesized in an optimized way for the given configuration and wiring. This drawback comes with the advantage of having highly configurable components through software, such as AXI VDMA, which are already verified.

Table 5.2: FPFA utilization of all components.

Hardware Unit	BRAMs	DSPs	FFs	LUTs
AXI Interconnect	0	0	3,751	3,426
AXI VDMA	2	0	4,807	3,206
AXI4-Stream Subset Converter	0	0	198	105
AXI4-Stream to Video Out	1	0	185	90
Clocking Wizard	0	0	1,439	1,097
IFX Image Processor	9.5	7	4,732	3,343
Processor System Reset	0	0	59	32
Processing System	0	0	0	4
Test Pattern Generator	0.5	6	2,076	2,491
Video In to AXI4-Stream	0	0	151	74
Video Timing Controller	0	0	2,095	1,123
Total	13	13	19,493	14,991

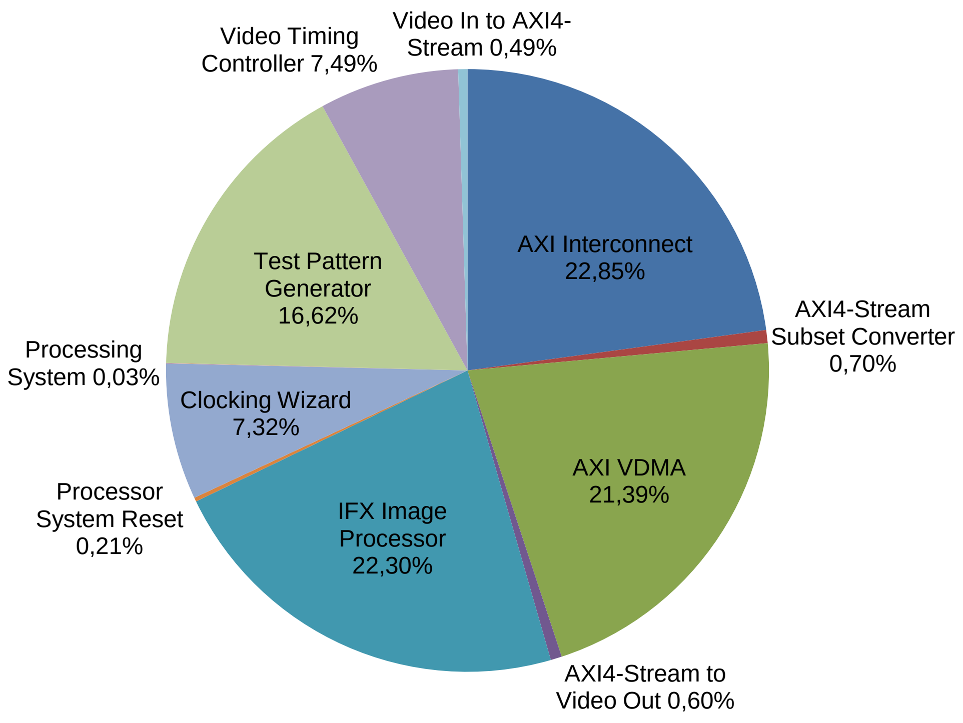


Figure 5.2: Percentage distribution of the used look-up tables.

5.1.2 Throughput

In Table 5.3 the maximum theoretical bandwidths of the memory interfaces are shown, which are calculated from the configured hardware parameters. The parallel interface of the ToF camera sends 12 Bit in parallel and works at a clock frequency of 66.66 MHz. As a result, the bandwidth of the camera is 100 MB/s without the use of an illumination time between captured frames. The AXI interfaces run with a clock frequency of 100 MHz. Hence, the bandwidth of Video In to AXI4-Stream IP core is 200 MB/s due to an increase of the data width to 16 Bit. The data width of the AXI VDMA module is 32 Bit, which corresponds to a bandwidth of 800 MB/s for the read and write channel. 128 Bit data width is delivered from the image processor, which is equivalent to a bandwidth of 3.2 GB/s. The high performance port provides a bandwidth of 1.6 GB/s and the DDR memory interface performs read and write commands with a bandwidth of around 4.2 GB/s. The AXI interconnect IP cores generally provides enough bandwidth for the access to the DDR memory, as outlined in [Enc14] and [Xil15j].

The components are not running with the maximum possible clock frequency for the AXI buses. Therefore, the interfaces between processing system, programmable logic and extended memory are not the bottlenecks. The video input stage and video output stage can gather and provide the calculated data in real-time. The image processor can be further accelerated to minimize the time for an execution of one operation.

Table 5.3: Theoretical bandwidths of the memory interfaces, which are calculated from the configured hardware parameters.

Interface	Data width [bits]	Clock frequency [MHz]	Bandwidth [MB/s]
PIF Camera	12	66.66	100
Video In to AXI4-Stream	16	100	200
VDMA (R+W)	32	200	800
Image Processor (R+W)	128	200	3,200
AXI4-Stream to Video Out	16	66.66	133.3
AXI HP (R+W)	64	200	1,600
DDR (R+W)	32	1,066	4,264

Frames per Second

In order to calculate the maximum possible FPS, the actual used camera configuration has to be considered: an illumination time of 1 ms between frames and a frame resolution of 160 x 120 is used. For one distance image, four raw frames with an equivalent phase-shift of 90° are required. The time for the transmission of one raw frame is 0.65 ms using the camera's parallel interface with a pixel clock of 66 MHz, as presented in Equation 5.1. The formula is outlined in [Inf15b]. As a result, the maximum FPS with the used configuration is 151, as illustrated in Equation 5.2.

$$\begin{aligned} t_{frame} &= \frac{image_{columns} \cdot image_{rows}}{f_{PIF}} + (3\mu s \cdot image_{rows}) \\ &= \frac{160 \cdot 120}{66.66\ MHz} + (3\mu s \cdot 120) = 0.65\ ms \end{aligned} \tag{5.1}$$

$$FPS_{max} = \frac{1\ s}{\#_{frames} \cdot (t_{frame} + t_{expo})} = \frac{1\ s}{4 \cdot (0.65\ ms + 1\ ms)} = 151 \tag{5.2}$$

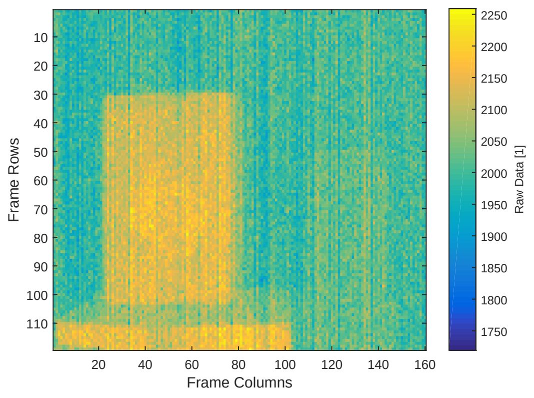
5.2 Time-of-Flight Processing Measurements

The focus of this section is to provide the evaluation of the implemented hardware-accelerated ToF algorithms. The results regarding performance and accuracy of the two pre-processing applications are presented. The number of hardware-integrated image processor operations is analyzed. Furthermore, the fixed-point calculation results of the hardware implementation are compared with the floating-point values of the hardware-related software implementation. The accuracy results, which represent only the calculation errors of the algorithm due to fixed-point arithmetic, are achieved by measuring a typical scene. Therefore, the shown mean and maximum errors are not absolute values. It is important to note that no system errors, which result from the measurement setup, are analyzed.

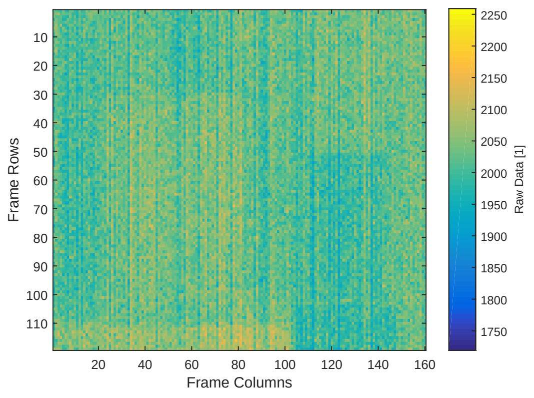
5.2.1 4-Phases Measurement

The 4-Phases algorithm is a typical pre-processing approach for a gesture recognition use-case. Such a use-case application requires high frame rates (40+ FPS) and high relative distance accuracy. The distance and amplitude image as well as the 3D coordinates are calculated. A compensation of the systematic errors is performed. Furthermore, the algorithm is part of the 8-Phases measurement and directly affects its computation performance and accuracy.

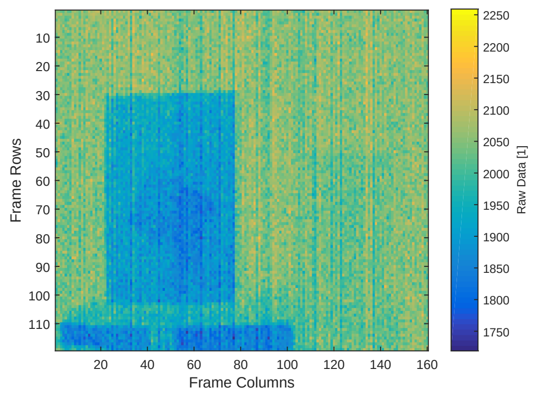
The results of one 4-Phases measurement are depicted in Figure 5.3 and Figure 5.4. The used ToF camera evaluation board only provides the calibration data for the FPPN, offset error and direction parameters. Therefore, no wiggling and temperature compensation is possible. The gathered four raw frames with an equivalent phase-shift of 90° are shown in Figures 5.3a to 5.3d. The calculated distance image, as illustrated in Figure 5.4a, shows the test objects. At the sharp edges of the objects, flying pixels are visible. These defective pixels can be improved by further post-processing methods, which are not further discussed in this work. The amplitude image is represented in Figure 5.4b. It shows an infrared image of the scenery.



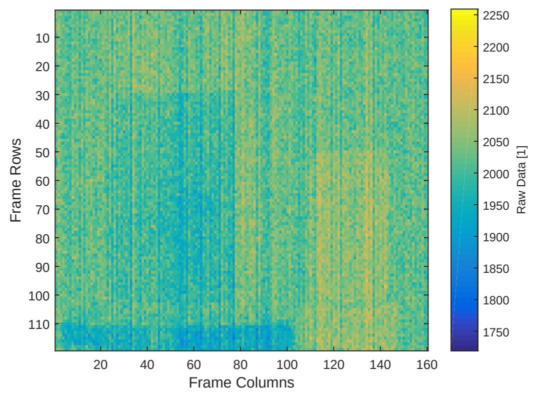
(a) Phase image 0°.



(b) Phase image 90°.

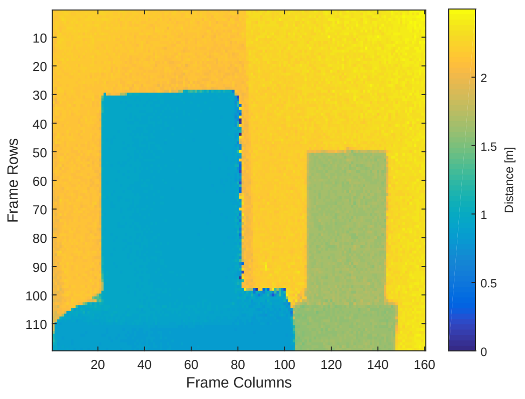


(c) Phase image 180°.

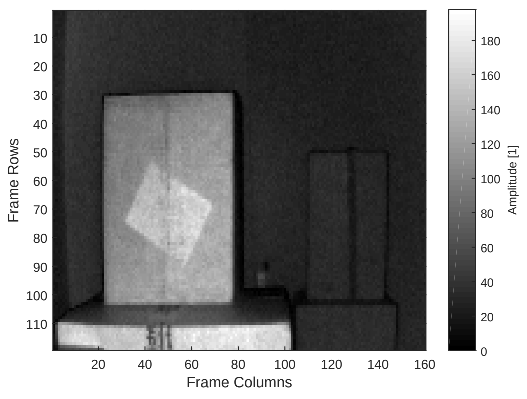


(d) Phase image 270°.

Figure 5.3: Raw frames of 4-Phases measurement (f_{Mod}: 60 MHz).



(a) Calculated distance image.



(b) Calculated amplitude image.

Figure 5.4: Calculated depth data of 4-Phases measurement (f_{Mod}: 60 MHz).

Performance

The timing results for the 4-Phases hardware implementation are outlined in Table 5.4. The floating-point software implementation of the reference algorithm executed on the ARM core needs 248.15 ms, which corresponds to 4 FPS. This computation is performed in floating-point arithmetic, and predefined math functions (e.g., arcus tangent, square root) are used. The time is not feasible for the required use-case.

The hardware algorithm implemented with image processor operations calculates the results in 30.5 ms. 32.8 FPS are possible with the use of hardware-integrated operations. This time results from the use of 23 hardware-accelerated operations, whereby one operation needs around 0.7 ms as well as the calculation time on the ARM core for the index operation of the wiggling compensation, as already shown in Listing 4.1. The execution time for this software part is 14.12 ms. There are more hardware operations than executed operations in the hardware-related software implementation needed due to specifics of the image processor. For instance, shift operations are necessary to bring values to the same fixed-point format. In addition, for the floor function one subtraction and two shift operations are necessary. In order to meet the timing requirements two improvements with trade-offs are introduced:

- **No Wiggling Compensation**

  The first improvement is the omission of the wiggling compensation. Wiggling is a distance-dependent error that needs to be compensated. Hence, the sophisticated calculation is typically calculated with the help of a look-up table. Because this computation is mostly done in software, it represents around half of the total time and significantly decreases the performance of this pre-processing approach. Wiggling correction is not necessarily needed in some use-case applications, such as gesture recognition. Hence, those calculation steps can be skipped to speed up the performance. In addition, there is no need of performing several hardware operations, such as the wiggling index calculation or the mapping of negative arcus tangent values. The time needed to calculate the results without wiggling compensation is 11.46 ms, which corresponds to 87.2 FPS. These results are compatible with the gesture recognition use-case, which requires high frame rates around 40 FPS.

- **No Amplitude Shifts**

  Another improvement concerns the amplitude calculation. Two hardware operations can be saved by not shifting the real and imaginary part. The fixed-point format (Q15.0) is used instead of (Q12.3). Thus, the amplitude values are not precise, but in most case this is not necessary. The execution time is 10.11 ms. The corresponding FPS, without wiggling compensation and amplitude shifts, are 98.9.

Table 5.4: Timing results of the 4-Phases algorithm.

Time-of-Flight Algorithm	t [ms]	FPS	#HW Instr.
Reference in Software	248.15	4.0	-
Reference in Zynq HW/SW	30.48	32.8	23
+ Improvement: No Wiggling Comp.	11.46	87.2	16
+ Improvement: No Amplitude Shifts	10.11	98.9	14

Accuracy

The accuracy of the 4-Phases algorithm is shown in Table 5.5. In addition to the final results (e.g. distance, amplitude, 3D point loud), the intermediate results of every hardware operation are calculated to view the error source and propagation, as illustrated in Table 5.6. The average error and the maximum error are represented. An error is defined as the absolute error between the fixed-point and floating-point value. For the mean error, the sum of all errors is divided by the number of pixels, which is 19,200, with the chosen camera configuration. The algorithm is performed with the provided test data and calibration data for all systematic errors. The modulation frequency is 80 MHz, which corresponds to an unambiguous range of 1.87 m.

The mean error of the distance is 0.032 mm and the maximum error is 1.33 mm. The maximum error is significant higher than the mean error. Hence, the maximum error value corresponds to only 0.07% of the unambiguous range. The mean amplitude error

Table 5.5: Mean/max errors of the 4-Phases algorithm.

Result	E_{mean} [mm]	E_{max} [mm]	E_{mean}[1]	E_{max}[1]
Distance	0.092	1.330	-	-
Amplitude	-	-	0.030	0.075
3D point cloud	0.080	1.343	-	-
+ Improvement: No Wigg. Comp.	0.076	0.268	-	-
+ Improvement: No Amplit. Shifts	-	-	0.117	0.253

Table 5.6: Mean/max errors after each image processor operation of the 4-Phases algorithm. In addition, the line numbers of the corresponding hardware-acceleration implementation, as shown in Algorithm 2, are represented.

Intermediate Result	E_{mean} [mm]	E_{max} [mm]
Imaginary part (3)	0.000	0.000
Real part (2)	0.000	0.000
Phase (4)	0.032	0.085
Phase Below Zero (5)	0.000	0.000
Phase Mapped (6)	0.032	0.085
Wiggling Index (8)	0.020	1.000
Wiggling Error (LUT) (8)	0.042	1.309
Phase Add. Wiggling (8)	0.056	1.320
Phase Div. 2 (8)	0.028	0.660
Phase Add. Errors (9)	0.032	0.672
Phase Floor (10)	0.000	0.061
Phase Subtr. (10)	0.032	0.672
Distance Mult. (11)	0.092	1.330

is 0.03 and the maximum amplitude error is 0.075. By reconsidering the value range of the amplitude, which is 0 to 2895.6, the percentage maximum error is 0.003%. This result is feasible for most use-cases. The errors of the 3D point cloud amounts to 0.08 mm for the mean error and 1.343 mm for the maximum error. Because the 3D coordinates are computed only with a multiplication of the distance with the direction parameters, the errors are almost the same.

In order to analyze the origin of the error, the intermediate results must be sequentially evaluated. The errors propagate linearly that means that for an addition or subtraction, the errors of the two operands are added or subtracted. If the two operands have the same number of integer and fraction bits, no error will occur. For multiplications and divisions, the errors of the two operands are multiplied or divided. Furthermore, multiplications and divisions cause an error if the fraction part is decreased. It is not possible to represent every exact floating-point value in the fixed-point format. Hence, the multiplication or division with a constant can also cause an error.

By considering the first few intermediate results, the first big maximum error occurs at the wiggling index calculation. A floor function is performed that shifts the values to the right and generates an integer value for the index look-up implemented on the ARM core. It was investigated that the error occurs when the correct floating-point value is slightly below an integer value, for example 0.9999, and the calculated fixed-point value is above, for example 1.0001, or vice versa. The absolute error is minimal but the floor function calculates 0 in the first case and 1 in the second case. That means for the wiggling compensation, the neighboring wiggling compensation value in the look-up table is chosen. This results in a maximum error of 1.32 mm for the phase value after wiggling compensation. The sequent intermediate errors are then linearly propagated. In order to solve this error source, the image processor architecture can be changed from 16-Bit to 32-Bit. Thus, the accuracy is increased and the appearance of such errors is decreased but the main issue still remains. This error can also arise with the floor function at the unambiguous range shift calculation step.

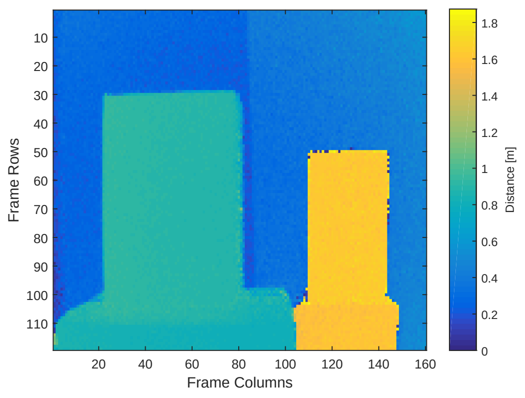
The errors of the improvements are also given. The omission of the wiggling compensation greatly reduces the maximum error of the distance. Due to this improvement, the wiggling index error does not occur, and no neighboring value in the look-up table is added. Thus, the mean distance error is 0.076 mm and the maximum distance error is 0.268 mm. This result is significantly better than with wiggling compensation. For the required use-case application, such as in gesture recognition, high relative accuracy is necessary. This can be achieved with this improvement by only causing a maximum calculation error of 0.268 mm.

The other improvement skips the two amplitude shifts. The inputs of the arcus tangent magnitude are the real and imaginary part, which are represented in the fixed-point format (Q15.0). Without the two shift operations the output of the magnitude operation is also in the format (Q15.0). Due the division with the constant two, the fixed-point format of the result is (Q14.1), which corresponds to a maximum error of 0.5. This corresponds to 0.017% of the value range, which is feasible for most use-case applications. The actual mean error is 0.117 and the maximum error is 0.253. The value range of the amplitude is 0 to 2895.6.

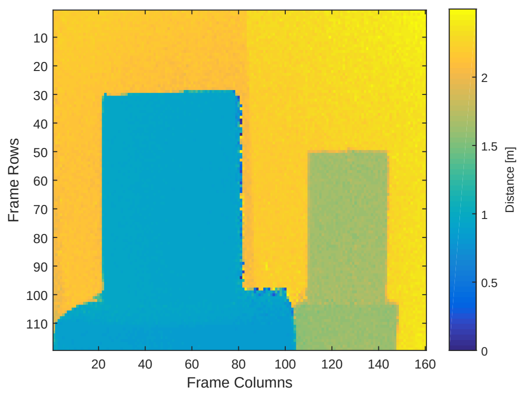
The results show that skipping the wiggling compensation can only fulfill the use-case requirements. The processing steps for the calculation of wiggling compensation are the main bottleneck of the 4-Phases algorithm. At first the calculation of the wiggling compensation value is not precise enough due the floor function of the wiggling index. Furthermore, the execution time is almost doubled because of the software part for the necessary index look-up. Thus, the interest arises to create in future a hardware component for the wiggling compensation or furthermore an IP core for calculation steps with look-up tables. The expansion of the used image processor for that is not easily realized as the LUT can be read in any order from the extended memory. Hence, the best-case would be to load the LUT into the BRAM of the FPGA. Another issue is the floor function, which is not precise enough. Therefore, the image processor could be extended to provide a floor or modulo operation with more precise intermediate results. Those ideas are not further discussed in this work.

5.2.2 8-Phases Measurement

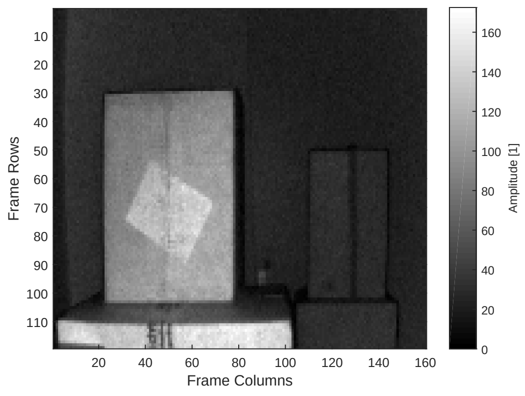
The 8-Phases algorithm represents a typical pre-processing approach for an indoor navigation use-case, for example, Google Tango. For this use-case lower, frame rates are sufficient and a high absolute distance accuracy is required. Two distance images of 4-Phases measurements with different modulation frequencies are combined. The final step is an unambiguous range extension. The result is a distance image and a 3D point cloud. Because the 4-Phases algorithm is part of this algorithm, the distance error of this approach directly propagates to this computation.



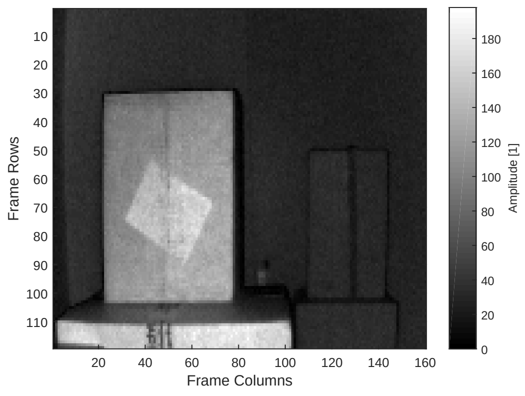
(a) Calculated distance image (f_{Mod}: 80 MHz).



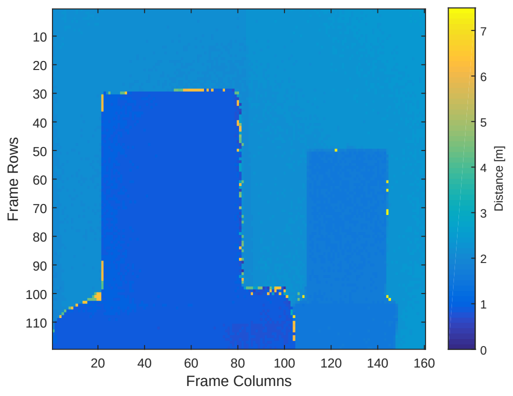
(b) Calculated distance image (f_{Mod}: 60 MHz).



(c) Calculated amplitude image (f_{Mod}: 80 MHz).



(d) Calculated amplitude image (f_{Mod}: 60 MHz).



(e) Calculated distance image.

Figure 5.5: Unambiguous range extension using dual modulation frequencies.

In Figure 5.5 the results for one measurement are depicted. The ToF camera is placed on the table to capture a scene of a room to observe the extension of the unambiguous range. The first distance image, as illustrated in Figure 5.5a, is the result of a 4-Phases measurement with a modulation frequency of 80 MHz, which corresponds to a unambiguous range of 1.87 m. The second measurement is done with a modulation frequency of 60 MHz, which provides valid results within the range of 2.49 m. Because of unambiguous range extension computation, the valid range can be increased to 7.46 m. This value can be calculated by using Equation 2.6.

It can be clearly seen that in the first picture the background beyond the unambiguous range, is mapped into the valid range by the unambiguous range shift calculation step of the 4-Phases algorithm, due to the 2π-periodicity of the phase. Thus, the distance information of such objects cannot not be correctly recognized. In Figure 5.5e the final distance image of the scenery is shown. By performing the extension processing step, the objects that are not correctly displayed can be mapped into the bigger range. Therefore, a decrease of the modulation frequency, which would also increase the unambiguous range, is not necessary. Flying pixels are visible at the sharp edges of the objects.

The calculated amplitude images of the scenery are represented in Figure 5.5c and Figure 5.5d. The amplitude can be used to provide further pre-processing steps on the distance before unambiguous range extension, such as the invalid pixel detection. Finally, the distance values are then translated into 3D coordinates. Therefore, the 3D point cloud calculation can be skipped in the 4-Phases algorithm for both images.

Performance

In Table 5.7 the timing results of the 8-Phases hardware implementation are shown. One successful computation needs the time of two 4-Phases executions, the unambiguous range extension and the 3D point cloud calculation. The software implementation on the ARM core needs 559.54 ms, which corresponds to 1.9 FPS. This result is not feasible for the most use-cases.

Table 5.7: Timing results of the 8-Phases algorithm.

Time-of-Flight Algorithm	t [ms]	FPS	#HW Instr.
Reference in Software	559.54	1.9	-
Reference in Zynq HW/SW	66.45	15.05	54
+ Improvement: No Wiggling Comp.	28.41	35.2	40
+ Improvement: No Amplitude Shifts	27.11	38.9	36

The hardware/software implementation using the ARM core and image processor operations needs 66.45 ms. Hence, 15.05 FPS are possible. The 4-Phases wiggling compensation on the ARM core significantly increases the execution time. For the unambiguous range extension 13 hardware-accelerated operations are necessary. This number results in the use of several operations for basic arithmetic, as well as a round and floor function. Thus, the complete algorithm is implemented with 54 image processor operations.

Because high frame rates are not required for this use-case, the obtained execution time without improvements is suitable.

Furthermore, the timings are also simulated using the two improvements (i.e., no wiggling compensation, no amplitude shifts) in the execution of the 4-Phases algorithm. A speed up can be achieved by skipping the wiggling correction. The timing result for that improvement amounts to 28.41 ms, which corresponds to 35.2 FPS. This is achieved by skipping the software implementation and by using 40 operations of the image processor. The skipping of the amplitude shifts decreases the time only minimally. Because only two operations can be saved, the execution time is 27.11 ms and the FPS are 38.9.

Accuracy

In Table 5.8 the accuracy results of the 8-Phases hardware implementation is shown. The fixed-point values are compared to the floating-point results of the hardware-related software implementation. The mean and the maximum error are equally defined as for the evaluation of the 4-Phases algorithm. The 4-Phases distance images are used. The modulation frequencies are 80 MHz for the first image and 60 MHz for the second image. The second distance image is the same as described in the accuracy section of the 4-Phases algorithm.

Table 5.8: Mean/max errors of the 8-Phases algorithm.

Result	$E_{mean}[mm]$	$E_{max}[mm]$
Distance	0.534	2.456
3D point cloud	0.272	2.043
+ Improvement: No Wigg. Comp.	0.523	2.109

The distance mean error is 0.534 mm and the maximum error is 2.456 mm. As for the 4-Phases algorithm the maximum error is also significant higher than the mean error. Due to the bigger unambiguous range, the percentage maximum error is only 0.003%. This result can be accepted for the required use-case, which requires high absolute accuracy due to the compensation of all systematic errors. 0.272 mm is the mean error and 2.042 mm is the maximum error of the 3D point cloud. The improvement of skipping the wiggling compensation results in a mean distance error of 0.523 mm and a maximum distance error of 2.109 mm. Compared to the 4-Phases algorithm, it is obvious that this improvement only slightly reduces the maximum error. This behavior is further inspected in the next paragraph. The amplitude errors are not represented because the amplitude images of the 4-Phases measurements are used and no special calculation in the unambiguous range extension is performed.

The intermediate results of every used operation are outlined in Table 5.9 to analyze the error source. At the first few operations the errors are high due to the error propagation of the 4-Phases distance values. The error of the round intermediate result is 0 mm. After the round operation, the intermediate result is multiplied with a constant. Therefore, the error of the second multiplication should also equal 0 mm. This is not the case because the multiplication constant cannot be represented exactly in the fixed-point format. Thus, a maximum error of 0.163 mm occurs which propagates to the final value. Due to the final multiplication with the large unambiguous range constant, the maximum distance error is finally 2.456 mm. Because of this error source, the error after the first improvement (no wiggling compensation) is not that strongly decreased as in the 4-Phases algorithm.

The results show that the high absolute accuracy, which is required for this use-case application, is achieved by providing an implementation with a compensation of all systematic errors. Because lower frame rates are sufficient, the wiggling compensation in software can be performed. Furthermore, it is shown that the limited accuracy of the 16-Bit image processor significantly limits the system's accuracy. The errors of the unambiguous range extension computation are almost eliminated because of the multiplications with very small constants. Thus, the errors of the 4-Phases algorithm do not heavily influence the results. Because of inaccuracies of the representation of values in the 16-Bit fixed-point format, another error source is raised. One solution would be to upgrade the image processor to a 32-Bit architecture in order to minimize such errors. This work does not further discuss this issue.

Table 5.9: Mean/max error after each image processor operation of the 8-Phases algorithm.

Intermediate Result	$E_{mean}[mm]$	$E_{max}[mm]$
Addition (1st)	0.141	2.403
Subtraction	0.144	2.165
Multiplication (1st)	0.263	3.564
Round	0.000	0.000
Multiplication (2nd)	0.016	0.163
Multiplication (3rd)	0.067	0.253
Addition (2nd)	0.071	0.329
Floor	0.000	0.000
Subtraction	0.071	0.329
Distance Mult.	0.534	2.456

5.3 Demonstrator

A demonstrator is implemented to show the feasibility of the system. Figure 5.6 shows the complete developed ToF processing platform. Infineon's ToF evaluation kit is connected through the parallel interface with the Xilinx Zynq platform and provides the gathered raw data. The Zynq development board from Trenz Electronic performs the ToF processing steps and sends the results through the FPGA's parallel interface via an adapter PCB to the AURIX platform. The pre-processed data is then evaluated and transmitted over the Ethernet interface to the PC. There it can be further used for other purposes, for example, displaying a live-stream.

During development some issues regarding the transmission over the parallel interfaces were recognized. In the following paragraphs, those results are described in detail.

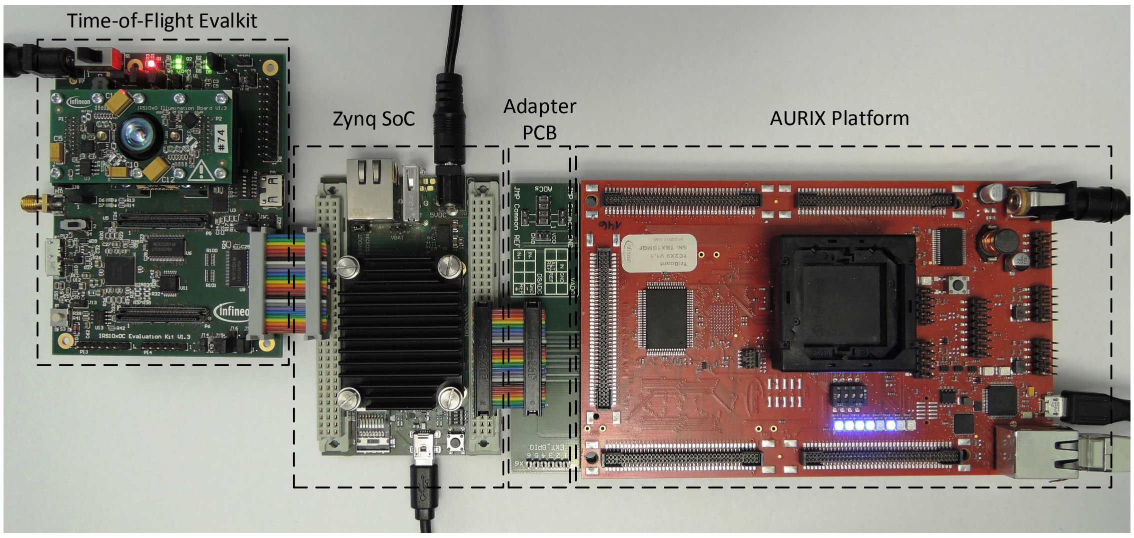


Figure 5.6: Developed demonstrator of the overall system.

Video Input Stage

At first, the initial setup of the connection between the ToF camera and the Zynq platform is described. Instead with a short flat cable, the parallel interface was firstly connected with separate cables. Each signal was connected with a 10 cm long cable. This floating wiring caused transmission errors. Therefore, the separate cables were exchanged with a flat cable of 10 cm length. With this replacement the setup was more stable but also transmission errors were occurring.

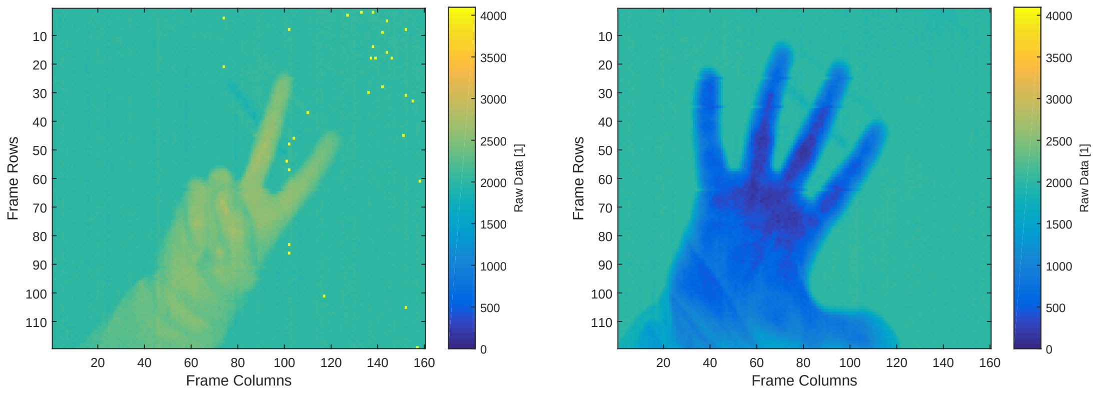
In Figure 5.7 some error-prone transmitted frames are depicted. The raw frames are gathered from the ToF camera and then received through the video input stage on the Zynq. The data is directly copied from the Zynq's extended memory to the PC to exclude errors in the video output stage and plotted with the software tool Matlab. One effect of the unstable transmission is the occurrence of pixels with a wrong value value, e.g., the maximum value. Hence, these pixels are easily recognizable in Figure 5.7a. The frame is captured with a pixel cock of 66.66 MHz.

In order to see if the error is reasoned on the high clock frequency, the pixel clock is changed. The ToF camera can be configured to use another pixel clock. A divider can be configured to decrease the maximum pixel clock frequency of 66.66 MHz. Following dividers are possible: 2, 4 and 8. Hence, following clock frequencies are adjustable: 33.33 MHz, 16.66 MHz and 8.33 MHz. Therefore, the pixel clock frequency is changed to 16.66 MHz. As illustrated in Figure 5.7b, the false pixels do not appear anymore but there are blurred rows visible.

In order to be sure about the occurrence of these errors, the visible verification is not enough. Because the exact pixel values are not known for verification, only the known pseudo row can be verified. This was done during the verification step of the video input stage. These pixel values of captured frames are compared to the predefined values in real-time. This test shows that at almost every received frame, some transmission errors occur.

The error source might be an electromagnetic compatibility (EMC) issue due to the long cable length. In fact, each row is buffered on the ToF evaluation kit to provide better transmission results. The 10 cm flat cable with nearly no isolation is not suitable for this framework. For instance, if a line is logical low between two logical high lines, the signal can also be pulled to high. Therefore, it is not possible to use the framework in use-case applications where the ToF camera needs to be placed far away from the Zynq platform.

The solution is a shorter flat cable with a length of around 2 cm, which replaces the long cable. In this setup no such transmission errors occur. Such a short cable is also used to connect the Zynq development board with the AURIX platform where such errors also occurred. This setup is verified with a test of the pseudo row of transmitted frames. This final testing step is performed with a high amount of frames in real-time, so that the error can be excluded.



(a) Pixel clock: 66.66 MHz. (b) Pixel clock: 16.66 MHz.

Figure 5.7: Transmission errors at the video input stage using a 10 cm long flat cable, e.g., pixel errors, blurred rows.

Video Output Stage

The connection between the Zynq development board and the AURIX platform is established with an adapter printed circuit board (PCB) provided from Infineon, as illustrated in Figure 5.8. It connects the camera interface of the AURIX with the parallel interface of the ToF camera. Therefore, the parallel interface is equally specified (e.g., data signals and synchronization signals) on the FPGA. As shown in the top view of the adapter PCB the 12-Bit data, synchronization and other signals are connected to a parallel interface plug, as outlined in [Inf15a].

The connection between the Zynq development board and the adapter PCB is realized with a 2 cm long flat cable. Because of the transmission errors that occur with a 10 cm long cable at the video input stage, also a shorter flat cable is used. Due to the previous

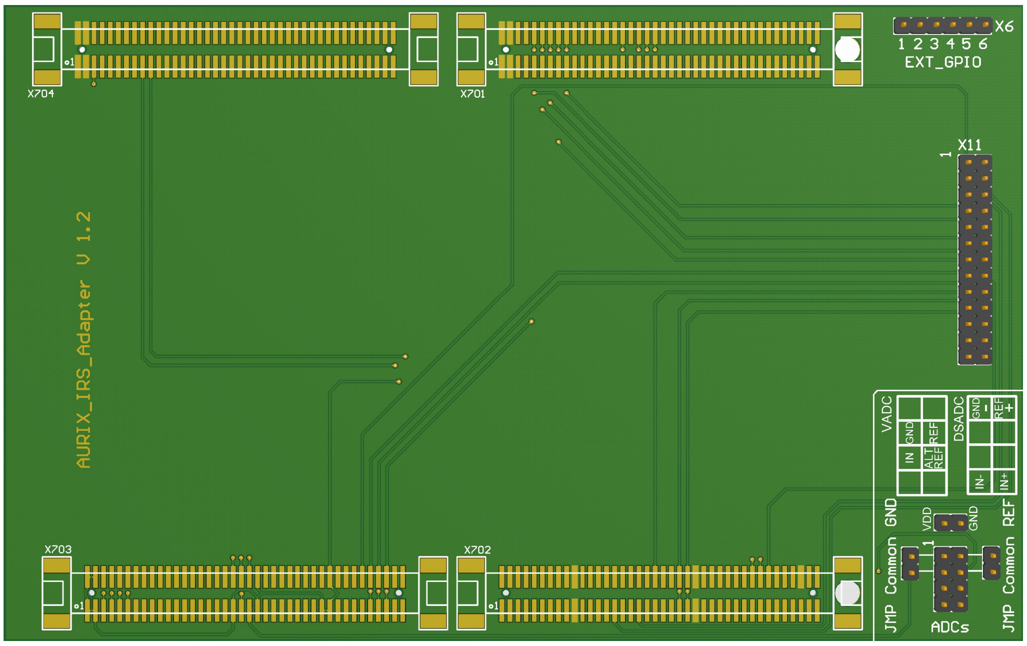
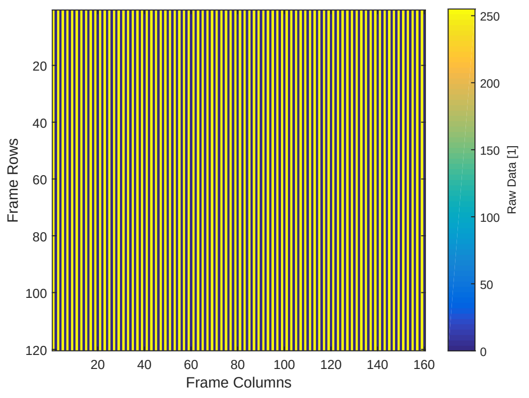


Figure 5.8: Adapter PCB [Inf15a].

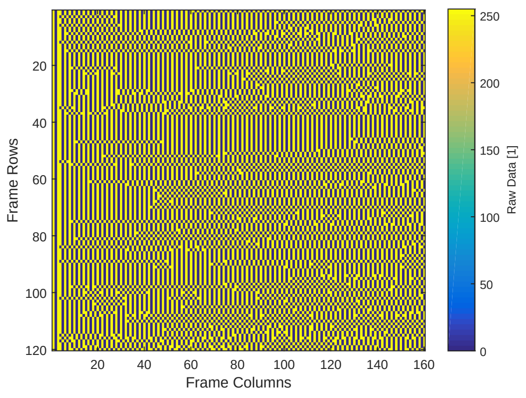
insights, one would assume that no error-prone transmission would appear. But during the verification step of the video output stage, similar errors could be recognized. Therefore, a predefined pattern was loaded into the extended memory and then sent via the transmit logic through the parallel interface to the AURIX.

In Figure 5.9 an example of the transmission errors is shown. The Zynq platform is connected with a 2 cm long flat cable with the adapter PCB. The test pattern that is sent is depicted in Figure 5.9a. Each alternating column has the same value. The pixel values of the complete first column are 0xFFF. The second column is 0x0 and so on. The frequent change of a pixel value from logical high to logical low, provokes lots of transmission errors. The received test pattern on the AURIX is shown in Figure 5.9b. It is clearly obvious that blurred rows occur like at the video input stage. In addition, false pixel values are also transmitted.

It is not possible to further decrease the length of the flat cable. Therefore, the adapter PCB might be the error source regarding electromagnetic compatibility issues. Looking at the top view of the adapter board, it is recognizable that long signal paths are used to connect the parallel interface plug with the camera interface. Furthermore, the signals of the camera interface are not placed in the same area. The pins that connect the camera interface of the AURIX are far away distributed. Thus, different signal path lengths are used for the connection, which can cause timing problems. Another issue might be the long distance between the two interfaces. The signal paths on the PCB are around 10 cm long. This length is already a problem at the video input stage.



(a) Test data send from Zynq SoC.



(b) Received data on PC.

Figure 5.9: Transmission errors at the adapter PCB using a 2 cm long flat cable, e.g., blurred Rows (pixel clock: 66.66 MHz, test pattern: 0xFFF/0x0)

The solution is to change the pixel clock frequency of the output stage to 8.33 MHz. This decreases the electromagnetic compatibility issues. As already recognized at the video input stage, lower pixel clocks provide better results. This solution works also with the output stage. Due to this decrease, latency is created which can limit the FPS. Another solution might be the integration of a buffer on the adapter PCB. Regarding the short flat cable, which can reduce the number of possible use-case applications, it is possible to connect the AURIX over another interface with the Zynq development board. One good alternative is the Ethernet interface. For an easy use of this interface an operating system, e.g., FreeRTOS ([Ltd15]), has to be used. The proposed solutions are not further discussed in this thesis.

5.4 High Level Synthesis

High Level Synthesis is used to fulfill the requirement of high flexibility in the system. Therefore, two algorithms are implemented and compared to the software implementation and Infineon's image processor regarding performance and area. In this section the estimation results of the HLS tool chain and the final implemented results are presented and evaluated.

Two test-cases were carried out with an image resolution of 160 x 128, which corresponds to 20,480 pixels. The generated IP cores run with a clock frequency of 50 MHz. In order to determine the execution time of a competition, a timer is started when the start register is set and immediately stopped after a thrown hardware interrupt.

Pixel-by-Pixel Addition

In the first test-case, a pixel-wise addition of two 16-Bit images is performed. In Table 5.10 the timing and utilization results are outlined. The execution time of the HLS implementations clearly outperforms the software implementation on the ARM core. The ToF image processor needs 0.44 ms for one operation. The execution time of the implemented solution with the multi-burst approach is 1.7 ms and with the dataflow directive is 0.82 ms.

The specified clock frequency is 50 MHz, which corresponds to a clock period of 20 ns. If one pixel calculation is performed in one clock cycle, the computation time of 20,480 pixels will be 0.41 ms. By keeping the execution times in mind, the image processor needs one clock cycle per pixel including some overhead timing.

The multi-burst approach sequentially performs following processing steps: read first image from memory, read second image from memory, perform calculation on two images and write result into memory. Therefore, four clock cycles are needed to compute one result pixel. Hence, the estimated time and the real time are almost the same.

The dataflow approach should need one clock cycle per pixel. Thus, the estimated time is 0.41 ms. The real timing result is however 0.82 ms, which is twice the estimated time. Hence, there must be a bottleneck in the reading of the memory because two clock cycles are necessary. In the first pipeline stage the first image is read from memory, whereas in the second pipeline stage the second image is read, the calculation is performed and the result image is written into memory. It is not possible to read both images in one pipeline stage. The image processor efficiently implements its input buffer for two input image streams, as shown in [Enc14].

The area is smaller for both HLS implementations because the image processor implements lots of more different operations. The dataflow approach needs more utilization resources because every input or output parameter is synthesized with an own AXI High Performance (HP) port, whereas the multi-burst approach is generated with only one AXI HP interface.

Table 5.10: HLS results: addition of two images.

Implementation	Est. Clock Cycles	Est. t [ms]	t[ms]	BRAMs	DSPs	FFs	LUTs
Reference in Software	-	-	15.60	-	-	-	-
HLS Multi-Burst	82,056	1.64	1.70	6	0	1980	1589
HLS Dataflow	20,502	0.41	0.82	6	0	3900	3836
Image Processor	-	-	0.44	9.5	7	4732	3343

Amplitude calculation

The second test-case calculates the amplitude. The timing and utilization results are shown in Table 5.11. The software implementation is much slower than the hardware-accelerated implementations. With the image processor four separate operations have to be performed which needs 1.76 ms. The execution time of the multi-burst approach is 2.48 ms, and of the dataflow approach is 1.62 ms.

Table 5.11: HLS results: ToF amplitude calculation.

Implementation	Est. Clock Cycles	Est. t [ms]	t[ms]	BRAMs	DSPs	FFs	LUTs
Reference in Software	-	-	95.60	-	-	-	-
HLS Multi-Burst	123,316	2.47	2.48	10	2	4491	4014
HLS Dataflow	20,543	0.41	1.62	10	2	8902	8438
Image Processor	-	-	1.76	9.5	7	4732	3343

In the multi-burst implementation six clock cycles are used to compute on result pixel value. At first, the four images are read from memory. Next, the amplitude is calculated and saved into memory. The estimated results are quite similar to the real execution time.

The dataflow approach implements a pipeline architecture and should use one clock cycle per pixel but it needs around four. In the first three pipeline stages the first three images are read from memory, whereas in the fourth pipeline stage the fourth image is read, the calculation is performed and the result image is written into memory. It is not possible to read all four images in one pipeline stage. However, the execution time is still lower than the one of the image processor because the calculation steps are successfully implemented in a pipeline. Therefore, the bottleneck is also the memory interface.

The utilization of the multi-burst approach is quite similar to the image processor because more operations are used in the HLS implementation. The area of the dataflow approach is twice as big because four input images are used and therefore four AXI HP ports are necessary.

The results show the trade-off between flexibility and achievable hardware-acceleration. It is outlined that High Level Synthesis is a quite useful tool for rapid prototyping. The estimated results of the HLS toolchain should be carefully noted because they can differ widely. Furthermore, the limits of HLS are quickly reached with hardware-accelerated components, which need to process high amounts of data from the extended memory.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The Time-of-Flight range imaging technology has attracted the interest of developers for use-case applications in recent years. Due to the small form factor that can be achieved with indirect ToF cameras, such systems can be integrated into embedded systems, for instance, smart phones. Thus, commercial solutions will come up with a range of applications with specialized requirements.

In this work an FPGA-based ToF processing platform is introduced that is based on a hardware/software system and allows the flexible realization of lots of different use-case applications. Furthermore, critical calculations are performed in hardware that increases the performance of the overall system.

To fulfill the requirements, the Xilinx Zynq platform is used, which allows the hardware/software co-development of complex ToF-based applications. Two pre-processing applications are developed to show the features of the framework. The state-of-the-art 4-Phases and 8-Phases algorithms are implemented. Such algorithms have applications in gesture recognition or indoor navigation use-cases. The distance, amplitude and 3D point cloud can be computed with almost 100 FPS while only producing an average calculation error of 0.08 mm.

In order to present the feasibly of the proposed framework, a demonstrator with the automotive platform AURIX developed by Infineon is implemented. It is shown that the system can also fulfill the high performance and memory requirements in safety-critical applications in the automotive domain. Thanks to the pre-processing on the Zynq platform, the amount of transferred data can be decreased and allows the AURIX to perform more complex use-case applications with the additional free resources.

Finally, rapid algorithm prototyping is possible through the presented high level synthesis approach. It is shown that the performance and utilization is in the same dimension as an application-specific developed hardware-component. A computation implemented with several image processor operations is even outperformed by the generated high level synthesis module. Therefore, performance-intensive calculations in software can be successively integrated into hardware. The trade-off between acceleration and flexibility is illustrated by the obtained results.

6.2 Future Work

The implementation of a demonstrator proves the feasibility of the presented FPGA-based ToF processing systems in this work. As a next step, the system can be improved regarding performance and accuracy. Furthermore, the framework can be extended for further yet, unknown use-case applications. Future work might involve following topics:

- **32-Bit Image Processor**

  In order to improve the processing accuracy of the two implemented ToF algorithms, the 16-Bit architecture of the image processor can be increased to, for example, 32-Bit. The maximum error of the 8-Phases algorithm is caused by the limited representation of a constant in the 16-Bit fixed-point format.

- **Additional Hardware-Integrated Operations**

  The image processor can be extended with additional hardware-integrated operations. For instance, a modulo operation with a precise intermediate result in hardware can increase the accuracy. A shift operation can minimize error sources because in the current system a shift is performed with a multiplication or division that automatically rounds the result.

- **Parallelization of Hardware Components**

  Due to enough free resources on the FPGA and no data dependency between pixels, the performance can be quickly improved by adding further image processors to the hardware design. For instance, adding a second image processor can double the maximum achievable FPS.

- **Wiggling Correction in Hardware**

  The wiggling compensation can currently only be implemented in software on the ARM. Hence, a considerable speed-up can be achieved with a hardware-integrated calculation. Because the wiggling look-up table is non-sequentially read from memory, an extension of the image processor is not easily achieved. For instance, the proposed High Level Synthesis approach can be used to create such a hardware component.

- **Hardware-accelerated Post-processing**

  In order to improve the quality of the computed distance data, hardware-accelerated calculation for further post-processing can be integrated in the proposed system, for example, noise filtering.

- **Ethernet Interface**

  The Ethernet interface of the Zynq development board can be activated and used for transmission of the calculated data to the AURIX. This connection is less error-prone than the parallel interface in combination with the adapter PCB and can also be used to directly connect a host PC to the Zynq platform.

- **Real-Time Operating System**

  For an easy use of the already provided Ethernet protocol implementation, a real-time operating system, such as FreeRTOS, can be integrated in the current software system. Furthermore, such an operating system would extend the proposed platform for use-case applications with real-time requirements.

- **Further Use-Case Applications**

  Additional use-case applications, such as face recognition, can be directly implemented on the Zynq platform. Memory- and performance-intensive calculations can be moved from the AURIX to the hardware/software platform to free the valuable resources.

Future projects can use the developed Time-of-Flight processing system for design exploration in lots of different fields of applications. Furthermore, the platform can be extended with real-time, multi-core and mixed-criticality concepts, which allow the implementation of critical applications in the industrial and automotive domain.

Appendix A

Hardware System

Listing A.1: Detailed hardware implementation of the 4-Phases algorithm. Fixed-point number format (Q$w.f$) is represented for a modulation frequency of 60 MHz.

```
imag = ImgOp_Sub(F270, F90); //(Q15.0)
real = ImgOp_Sub(F0, F180); //(Q15.0)

phase = ImgOp_Atan(imag, real); //(Q0.15)
phase = ImgOp_Mul(phase, 1/pow(2, 2)); //(Q2.13); phase >>= 2;

// Wiggling Compensation
phaseLim = ImgOp_Limit(0); //(Q0.15); phaseLim = (phase < 0) ? 1:0;
phaseLim = ImgOp_Div(phaseLim, 1/pow(2, 14); //(Q1.14); phaseLim <<= 14;
phaseWigg = ImgOp_Add(phase, phaseLim); //(Q2.13)
wiggIdx = ImgOp_Mul(phaseWigg, (N_LUT-1)/2); //(Q13.2)
wiggIdx = ImgOp_Sub(wiggIdx, pow(2, 1)); //no rounding of next shift
wiggIdx = ImgOp_Mul(wiggIdx, 1/pow(2, 2)); //(Q0.15); wiggIdx >>= 2;
for( i=0; i<numPixels; i++ )
{
  wiggError[i] = wiggLUT[wiggIdx[i]];
}
phase = ImgOp_Add(phase, wiggError);

phaseFractionBits += 1; //(Q1.14); equal to division of 2
phase = ImgOp_Add(phase, offsetFppnTempErros); //(Q1.14)
phFloor = ImgOp_Sub(phase, pow(2, 13)); //no rounding of next shift
phFloor = ImgOp_Mul(phFloor, 1/pow(2, 14)); //(Q15.0); phFloor >>= 14;
phFloor = ImgOp_Div(phFloor, 1/pow(2, 14)); //(Q1.14); phFloor <<= 14;
phase = ImgOp_Sub(phase, phFloor); //(Q1.14)
distance = ImgOp_Mul(phase, c/(2*fMod)); //(Q3.12)

// Amplitude
real = ImgOp_Div(real, 1/pow(2, 2)); //(Q13.2); real <<= 2;
imag = ImgOp_Div(imag, 1/pow(2, 2)); //(Q13.2); imag <<= 2;
amplitude = ImgOp_Magnitude(real, imag); //(Q13.2)
amplitudeFractionBits += 1; //(Q12.3); equal to division of 2

// 3D Point Cloud
distanceX = ImgOp_Mul(distance, directionsX); //(Q3.12)
distanceY = ImgOp_Mul(distance, directionsY); //(Q3.12)
distanceZ = ImgOp_Mul(distance, directionsZ); //(Q3.12)
```

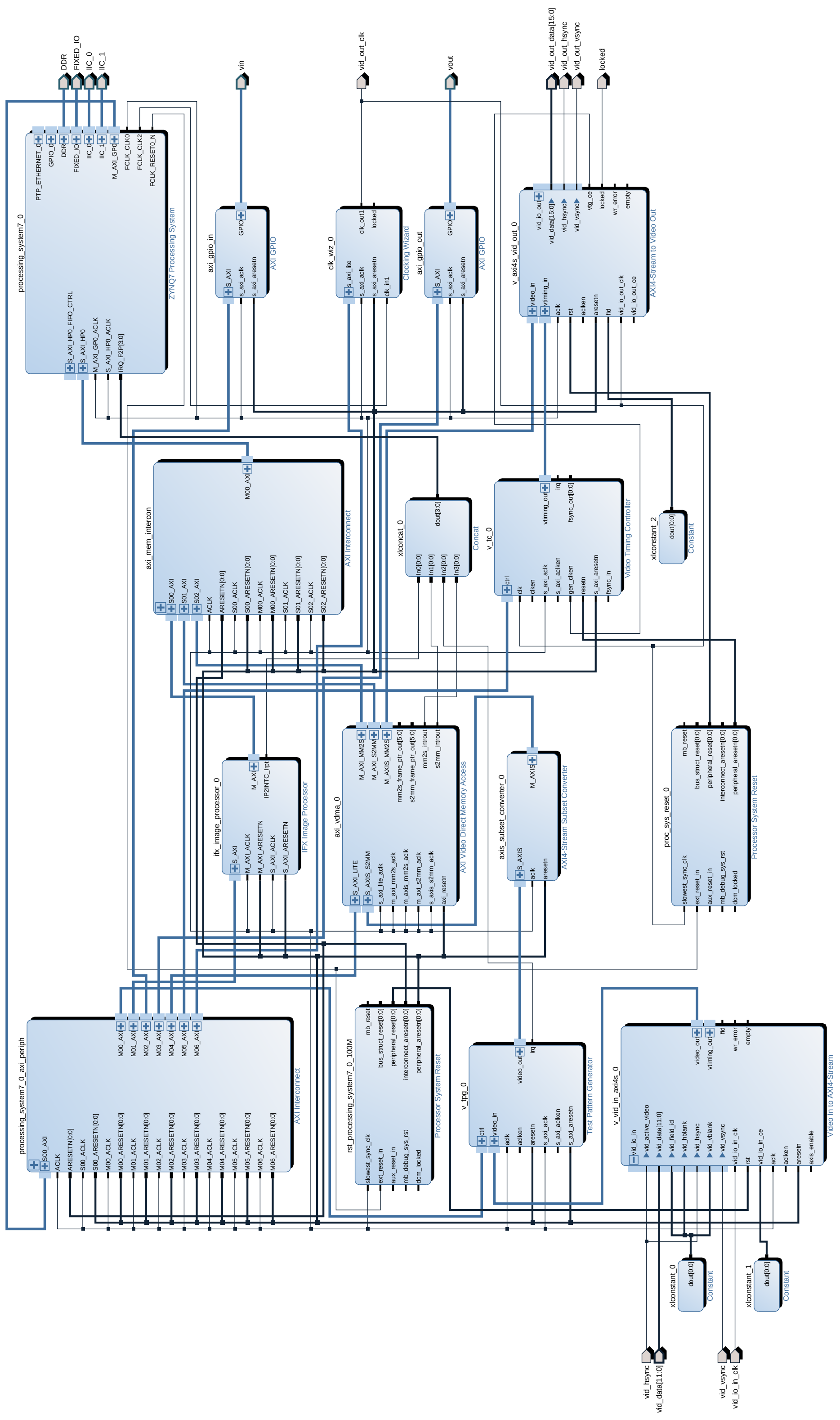


Figure A.1: Block diagram of the hardware system on the FPGA (Vivado IP Integrator).

Bibliography

[AGBS15] M. Ali Altuncu, Taner Guven, Yasar Becerikli, and Suhap Sahin. Real-Time System Implementation for Image Processing with Hardware/Software Codesign on the Xilinx Zynq Platform. *International Journal of Information and Electronics Engineering*, 5(6):473–477, November 2015.

[Alb07] Martin Albrecht. Untersuchung von photogate-pmd-sensoren hinsichtlich qualifizierender charakterisierungsparameter und -methoden. *PhD thesis, Department of Electrical Engineering and Computer Science, University of Siegen*, 2007.

[ASA$^+$14] Amine Ait Si Ali, Marek Siupik, Abbes Amira, Faycal Bensaali, and Pablo Casaseca-de-la Higuera. HLS based hardware acceleration on the zynq SoC: A case study for fall detection system. In *Computer Systems and Applications (AICCSA), 2014 IEEE/ACS 11th International Conference on*, pages 685–690, November 2014.

[Cad13] Cadence. C-to-Silicon Compiler High-Level Synthesis, December 2013.

[CCJD11] Johnny Mc Clymont, Dale A Carnegie, Adrian Jongenelen, and Benjamin Drayton. The Development of a Full-Field Image Ranger System for Mobile Robotic Platforms. In *IEEE International Symposium on Electronic Design, Test and Application (DELTA)*, pages 128–133, January 2011.

[CEES14] Louise H Crockett, Ross A Elliot, Martin A Enderwitz, and Robert W Stewart. *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc.* Strathclyde Academic Media, 2014.

[Cig15] Eric Cigan. MathWorks Targets Hardware/Software. `http://www.eejournal.com/archives/articles/20150127-matlab/`, January 2015. Accessed: 2016-08-04.

[DCP$^+$07] Adrian A Dorrington, Michael J Cree, Andrew D Payne, Richard M Conroy, and Dale A Carnegie. Achieving sub-millimetre precision with a solid-state full-field heterodyning range imaging camera. *Measurement Science and Technology*, 18(9):2809, 2007.

[DFH$^+$15] N. Druml, G. Fleischmann, C. Heidenreich, A. Leitner, H. Martin, T. Herndl, and G. Holweg. Time-of-Flight 3D Imaging for Mixed-Critical Systems. In *13th International Conference on Industrial Informatics (INDIN)*, pages 1432–1437, July 2015.

[DSH$^+$13] Milos Davidovic, Johannes Seiter, Michael Hofbauer, Wolfgang Gaberl, and Horst Zimmermann. A background light resistant tof range finder with integrated pin photodiode in 0.35 μm cmos. In *SPIE Optical Metrology 2013*, pages 87910R–87910R. International Society for Optics and Photonics, 2013.

[Enc14] Enclustra. IFX Image Processor, Design Document, November 2014.

[FAT11] Sergi Foix, Guillem Alenya, and Carme Torras. Lock-in time-of-flight (tof) cameras: a survey. *Sensors Journal, IEEE*, 11(9):1917–1926, 2011.

[HLCH12] Miles Hansard, Seungkyu Lee, Ouk Choi, and Radu Patrice Horaud. *Time-of-flight cameras: principles, methods and applications.* Springer Science & Business Media, 2012.

[Inf13] Infineon Technologies. IRS10x0C, Evaluation Kit User Manual, December 2013.

[Inf14] Infineon Technologies. AURIX, User Manual, May 2014.

[Inf15a] Infineon Technologies. Camera Adapter Board for TriBoard, December 2015.

[Inf15b] Infineon Technologies. MiraCE Development Specification, Januar 2015.

[JBP$^+$10] Adrian Jongenelen, DG Bailey, Andrew D Payne, Dale A Carnegie, and Adrian A Dorrington. Efficient FPGA implementation of homodyne-based time-of-flight range imaging. *Journal of Real-Time Image Processing*, 7(1):21–29, 2010.

[JCDP08] A. Jongenelen, D.A. Carnegie, A.A. Dorrington, and A.D. Payne. Heterodyne range imaging in real-time. In *3rd International Conference on Sensing Technology*, pages 57–62, November 2008.

[JCP$^+$09] Adrian Jongenelen, Dale Carnegie, Andrew D Payne, Adrian Dorrington, et al. Development and Characterisation of an Easily Configurable Range Imaging System. In *24th International Conference Image and Vision Computing New Zealand*, pages 79–84, November 2009.

[JCPD10] Adrian PP Jongenelen, Dale A Carnegie, Andrew D Payne, and Adrian A Dorrington. Maximizing precision over extended unambiguous range for tof range imaging systems. In *Instrumentation and Measurement Technology Conference (I2MTC), 2010 IEEE*, pages 1575–1580. IEEE, 2010.

[KIR06] Timo KHALMANN, Hilmar INGENSAND, and Fabio REMONDINO. Calibration for increased accuracy of the range imaging camera swissrangertm. *ISPRS Archives*, 36(Part 5):136–141, 2006.

[Lan00] Robert Lange. 3D Time-of-Flight Distance Measurement with custom Solid-State Image Sensors in CMOS/CCD-Technology. *PhD thesis, Department of Electrical Engineering and Computer Science, University of Siegen*, 2000.

[LS01] R. Lange and P. Seitz. Solid-state time-of-flight range camera. *IEEE Journal of Quantum Electronics*, 37(3):390–397, March 2001.

[LSKK10] Marvin Lindner, Ingo Schiller, Andreas Kolb, and Reinhard Koch. Time-of-flight sensor calibration for accurate range sensing. *Computer Vision and Image Understanding*, 114(12):1318–1328, 2010.

[Ltd15] Real Time Engineers Ltd. Free RTOS, 2015, 2015. Free RTOS Website.

[Lua01] Xuming Luan. Experimental investigation of photonic mixer device and development of TOF 3D ranging systems based on PMD technology. *PhD thesis, Department of Electrical Engineering and Computer Science, University of Siegen*, 2001.

[Mat14] MathWorks. Matlab, R2014b, 2014. Matlab Website.

[Mic13] Microsoft. Visual Studio, 2013, 2013. Visual Studio Website.

[NBVT14] Hong Thi Khanh Nguyen, Cecile Belleudy, and Pham Van Tuan. Fall detection application on an ARM and FPGA heterogeneous computing platform. *International journal of advanced research in electrical, electronics and instrumentation engineering*, 3(8):11349–11357, 2014.

[PMD13a] PMDTechnologies. pmd Application Note AN_004, Basic ToF Data Processing, December 2013.

[PMD13b] PMDTechnologies. pmd Application Note AN_005, Data Calibration, October 2013.

[PMD14] PMDTechnologies. Calibration Data, File format, April 2014.

[RF13] Matthew Russell and Scott Fischaber. OpenCV based road sign recognition on Zynq. In *IEEE International Conference on Industrial Informatics (INDIN)*, pages 596–601, July 2013.

[SCH$^+$14] Benaoumeur Senouci, Imen Charfi, Barthelemy Heyrman, Julien Dubois, and Johel Miteran. Fast prototyping of a SoC-based smart-camera: a real-time fall detection case study. *Journal of Real-Time Image Processing*, October 2014.

[SGC14] Peyman Sabouri, Hamid GholamHosseini, and John Collins. *Advanced Technologies, Embedded and Multimedia for Human-centric Computing: HumanCom and EMC 2013*, chapter Border Detection of Skin Lesions on a Single System on Chip, pages 465–471. Springer Netherlands, 2014.

[SHB$^+$99] R Schwarte, H Heinol, B Buxbaum, T Ringbeck, Z Xu, and K Hartmann. Principles of 3-d imaging techniques, 1999.

[SHDZ13] Johannes Seiter, Michael Hofbauer, Milos Davidovic, and Horst Zimmermann. FPGA based time-of-flight 3D camera characterization system. In *16th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pages 240–245, April 2013.

[Tec15] Infineon Technologies. Free TriCore Entry Tool Chain, Version 4.6.6.0-infineon-1.1, 2015. Free TriCore Entry Tool Chain Website.

[Tre14] Trenz Electronic. 4x5 Carrier Boards Overview, September 2014.

[Tre15] Trenz Electronic. TE0720 User Manual, March 2015.

[Xil11a] Xilinx, Inc. The First Generation of Extensible Processing Platforms: A New Level of Performance, Flexibility and Fcalability, 2011.

[Xil11b] Xilinx, Inc. Vivado Design Suite User Guide: Logic Simulation, v2014.2, UG900, June 2011.

[Xil13] Xilinx, Inc. LogiCORE IP Video Timing Controller, v6.0, PG016, October 2013.

[Xil14a] Xilinx, Inc. Designing High-Performance Video Systems with the Zynq-7000 All Programmable SoC Using IP Integrator, v1.0, XAPP1205, March 2014.

[Xil14b] Xilinx, Inc. High-Level Synthesis, v2015.2, UG902, April 2014.

[Xil14c] Xilinx, Inc. LogiCORE IP AXI4-Stream to Video Out, v3.0, PG044, April 2014.

[Xil14d] Xilinx, Inc. LogiCORE IP Video In to AXI4-Stream, v3.0, PG043, March 2014.

[Xil14e] Xilinx, Inc. Test Pattern Generator, v6.0, PG103, October 2014.

[Xil15a] Xilinx, Inc. AXI Interconnect, v2.1, PG059, November 2015.

[Xil15b] Xilinx, Inc. AXI Video Direct Memory Access, v6.2, PG020, November 2015.

[Xil15c] Xilinx, Inc. AXI4-Stream Infrastructure IP Suite, PG085, April 2015.

[Xil15d] Xilinx, Inc. Clocking Wizard, v5.1, PG065, April 2015.

[Xil15e] Xilinx, Inc. Processing System 7, v5.5, PG082, September 2015.

[Xil15f] Xilinx, Inc. Vivado Design Suite - HLx Editions, 2015.1, 2015. Vivado Design Suite Website.

[Xil15g] Xilinx, Inc. Vivado High-Level Synthesis, 2015.1, 2015. Vivado High-Level Synthesis Website.

[Xil15h] Xilinx, Inc. Vivado IP Integrator, 2015.1, 2015. Vivado IP Integrator Website.

[Xil15i] Xilinx, Inc. Xilinx Software Development Kit, 2015.1, 2015. Xilinx Software Development Kit Website.

[Xil15j] Xilinx, Inc. Zynq-7000 All Programmable SoC, Technical Reference Manual, v1.10, UG585, February 2015.