ROBOTICS AND MECHATRONICS

# EFFICIENT VIDEO PIPELINES FOR DRONE APPLICATIONS

W.A. (Wolfgang) Baumgartner

MSC ASSIGNMENT

July 2021

050RaM2021
Robotics and Mechatronics
EEMathCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE. | TECHMED CENTRE UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

# Summary

A video pipeline design has been proposed in this project to look at a basic part of numerous machine vision applications involving drones. Most research in that field focuses on a complete application and does not look at the influence of implementation choices on video streaming. This project tries to fill in that gap.

Several options are explored to design a video pipeline that fits on a drone. A developer board is used that combines hardware and software to have enough performance and is suitable for use on a drone. The communication block of the design is tested and reached an average bandwidth of 461 MB/s with a latency of 76.6e $\mu$s.

The results indicate that the proposed design is feasible. Additionally, it can be used as a starting point for visual odometry or machine vision applications. Unfortunately, as yet nothing can be said about the influence of combining hardware and software on performance.

# Contents

# 1 Introduction

## 1.1 Context

Machine vision is a well established discipline that investigates how to extract information from digital images and how to interpret these images. Examples are an algorithm processing a camera feed to find a red ball or a system that can count traffic on a crossroads. These tasks seem easy to perform for a human. Intuitively, one can recognize known objects and give meaning to visual information. However, it is hard to express these tasks in a way a machine can understand. Therefore, it is not surprising that there is a lot of existing research on automatically processing images and related tasks.

An interesting branch of machine vision deals with smaller systems like drones. Drones are fast and can provide sight on hard-to-reach places. That makes them ideal for automated inspection tasks. However, drones also provide an additional challenge. Often, machine vision computations are performed on computer platforms that offer a lot of computational resources. On a drone, that is not possible because these platforms are usually too big and heavy. Therefore, machine vision implementations need to deal with limited resources that smaller computer platforms offer.

One example of such an implementation was done by Schmid et al. (2014). A commercial quadrotor drone was used as a basis for their design which was adapted to make autonomous navigation possible. For that, a combination of hardware and software processed the drone's camera feeds to track it's own motion. In experiments, the drone could safely navigate through a building and a coal mine. As this study focussed on designing a drone capable of autonomous navigation, it is beyond its scope to optimize the amount of image data processed or compare different ways of video streaming. A similar quadrotor drone was used by Meier et al. (2012) that uses image data to avoid obstacles and for flight control. Measurements showed that fusing inertial and visual data improved accuracy of the planned flight path. Additionally, the project resulted in a basic platform for autonomous flight for future research. As these objectives were reached, it was neglected to explain the choice of hardware for the image processing unit or its influence on measurements. In summary, two studies about autonomous drones have been shown that do not investigate the influence of choices concerning video streaming hardware.

## 1.2 Problem statement

Despite the fact that research on the just mentioned applications always needs video streaming, it does not get the necessary attention. It is essential for any machine vision application and influences its performance. This is also true for autonomous drone applications. Therefore, the focus of this thesis is video streaming in a drone context.

One of the challenges of video streaming is the amount of data that needs to be processed on time. Cameras have high data output, especially when resolution and frame rate of the video are high. In the chosen context, this is even more difficult because of limited resources on a system like a drone. Nonetheless, it is important for this thesis to keep bandwidth high.

## 1.3 Project goals

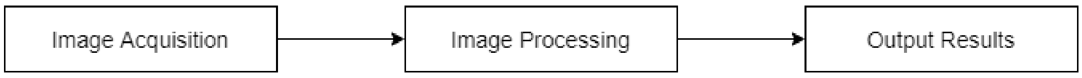

**Figure 1.1:** block diagram of top-level design

The goal of this project is to investigate the problem mentioned before, namely video streaming for drone applications. For this purpose, a video pipeline is designed. This design must observe relevant limitations for use on a drone and achieve sufficient performance for an actual application. Figure 1.1 shows what the most important design blocks are. The result of this design process fills the gap of earlier mentioned research and serves as groundwork for autonomous drones.

As already mentioned, resources for this design are limited. A drone can only carry a light, small platform that does not use too much power. This means that there is much less processing power available than on for example a big desktop PC. To compensate for this decrease in processing power, hardware and software are combined to make high performance video streaming possible. This project investigates how to integrate hardware and software in a beneficial way.

## 1.4 Plan of approach

As this project has the objective to design a video pipeline, different options for the design have been explored. Requirements have been defined to have a measure of what a good solution is. The design has been split in several parts and for each part, various solutions have been compared. The best option from each part has been picked to get a feasible design for a video pipeline.

In order to show the feasibility in practice, as much as possible of this design has been implemented and tested. For this, a significant part has been chosen and expanded with a test bed. Performance of this implementation has been measured to evaluate the proposed video pipeline and check if the earlier defined requirements have been fulfilled.

## 1.5 Outline report

In Chapter 2, two different hardware accelerators are explained and compared. The following chapter illustrates the video pipeline and what led to the proposed design. Subsequently, parts of the pipeline are implemented and tested which is shown in Chapter 4. The test results are interpreted and discussed in Chapter 5. Finally, Chapter 6 describes what can be concluded and what is recommended for future projects.

# 2 Background

Typically, machine vision algorithms like visual odometry are implemented in software and run on high-performance desktop computers (Warren et al., 2016; Song et al., 2013; Nistér et al., 2006). These systems are often rather big and consume a lot of power to deliver the necessary performance. On a drone however, these resources are quite limited. The embedded hardware platform needs to offer sufficient performance within these limits (Jeon et al., 2016). Therefore, GPUs and FPGAs are a viable option as hardware accelerators for this project that can outperform a pure CPU solution. A solution based on an ASIC can also be powerful enough. However, development time is much bigger than the available time for this project.

## 2.1 GPU

A modern GPU consists of several computation units. Each of these units contains a number of simple processing cores, control logic, a small memory cache and some dedicated cores for special functions. The Fermi architecture described in Gao (2017) for example has 16 streaming multiprocessors with 32 CUDA cores each. All of these computation units have their own registers available as well as some shared memory and L1 cache. Additionally, there is a shared L2 cache and a large RAM which is similar to main memory for a CPU. Another example is visible in Figure 2.1 which shows two of the 16 streaming multiprocessors inside a NVIDIA Geforce 8800GTX. This graphics card is not actually a candidate for this project as it is too heavy and needs too much power. However, the architecture is similar to GPUs on modern embedded platforms like the Nvidia Jetson series.

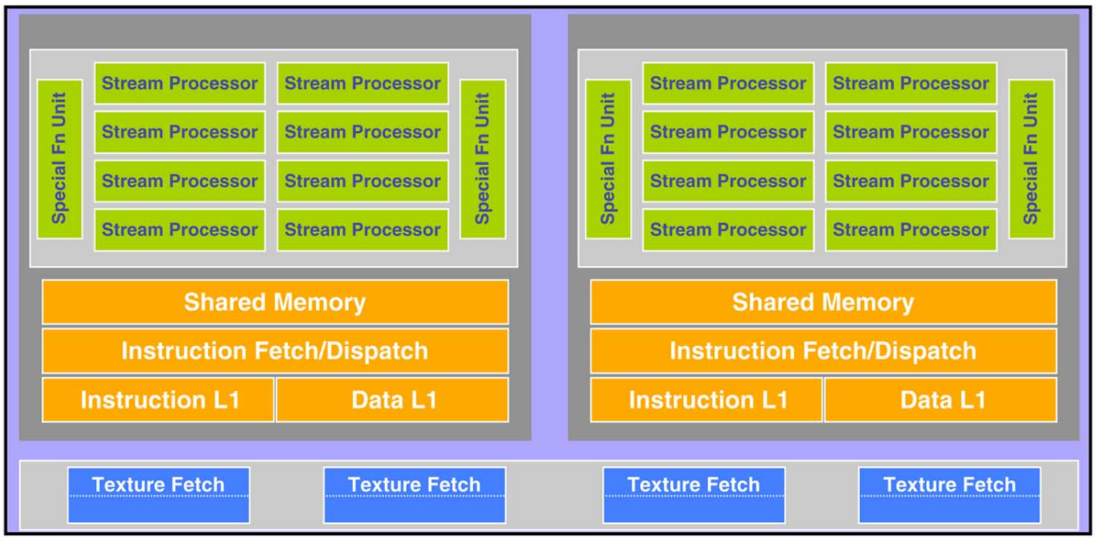

**Figure 2.1:** A pair of streaming multiprocessors of a NVIDIA Geforce 8800GTX; each multiprocessor contains eight stream processors, two special function units, shared cache, control logic and shared memory (Owens et al., 2008)

Older GPUs were mainly built to compute 3D graphics and for that they contained a graphics pipeline with more dedicated stages as opposed to general purpose computation units nowadays. However, the main idea holds that high performance is accomplished by processing the whole data set in parallel. This means a single processing step is computed in parallel for a lot of data in contrast to a CPU pipeline that tries to compute several steps in parallel on a single data point.

Traditionally, using a GPU for anything else but graphics was a difficult task because for a long time there were no high-level languages for GPUs available. This meant that any task to be run on a GPU had to be mapped on the graphics pipeline and expressed in terms of vertices or fragments. Nowadays though there are several languages that are relatively easy to use for a software programmer. Nvidia's CUDA for example is an API that can be used together with C code (Owens et al., 2008; Gao, 2017).

## 2.2 FPGA

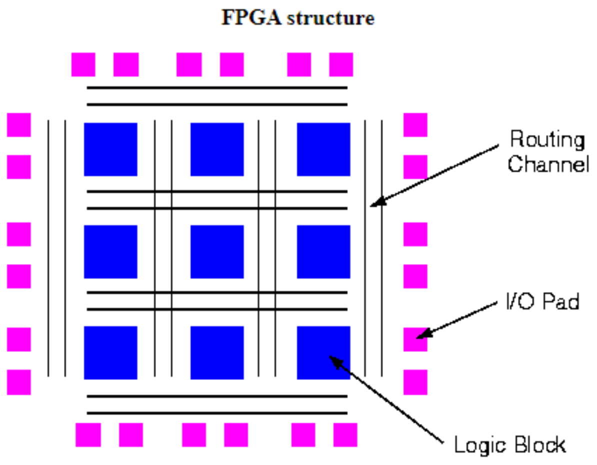

**Figure 2.2:** Structure of an island-style FPGA (Betz, 2005)

Most Field Programmable Gate Arrays (FPGA) consist of three building blocks. One of these blocks is called a logic element. It is made of a lookup table with four inputs, a full adder and a flip-flop and can be configured to behave like any logic gate. Therefore, it is a suitable building block for a digital circuit. A logic element can be connected directly to another logic element or via a routing channel when it is on another part of the chip. The last building block is the I/O pad which allows an FPGA to communicate with the environment. These three elements are enough to form a digital circuit with input and output (Altera, 2011). An illustration of a typical FPGA structure appears in Figure 2.2.

Modern devices additionally feature dedicated hardware like memory blocks or multipliers. These can reduce the area of a circuit because such functionality requires a lot of logic elements to implement. More complex examples of such dedicated blocks are DSP blocks or embedded processors. They can also feature logic elements that are structured a bit differently but with the same function (Intel, 2019b; Cullinan et al., 2013). Although FPGAs still fulfil the same role, they have evolved a lot over the years.

There are several ways to describe the desired behaviour of an FPGA. First, a hardware description language (HDL) can be used which is comparable to a programming language. The most common HDLs are Verilog and VHDL. More recently, some vendors like Xilinx and Intel try to raise the abstraction level by releasing compilers that can synthesize a circuit from a behaviour description in C/C++. At last, not all behaviour descriptions have to be written by hand as design suites like Intel's Quartus Prime comes with prebuilt blocks that can be combined in the Platform Designer.

After a digital circuit is described in the mentioned ways, the circuit needs to be synthesized. Nevertheless, it is considered good practice to simulate a circuit first. This has the advantage that all signals are visible for testing. On a synthesized circuit, this depends on the design but it is usually not the case which makes debugging more difficult. Software like Quartus Prime places all necessary gates including routing and I/O pins on the target device. The behaviour description is mapped onto the hardware (Chu, 2006; Farooq et al., 2012).

# 3 Design

Image Sequence
Communication
Output

**Figure 3.1:** block diagram of top-level design

## 3.1 Introduction

As context for the design objective, a use case has been picked that describes the application that was envisioned as long term goal. This use case is a drone that can inspect modern wind turbines. It has to fly up and down the tower on which the generator is mounted and look for signs of damage. This means it needs to autonomously navigate the wind turbine in its environment. Additionally, some kind of sensor is necessary to inspect the turbine structure.

The objective for this design space exploration is a video pipeline that serves as a basis for visual odometry on for example a drone. This means there needs to be a video source that streams image data. The data needs to be available for hardware and software to enable advanced image processing in the future. Additionally, communication between hardware and software on the targeted platform is necessary. Finally, a way of outputting information is required. Figure 3.1 is a representation of this video pipeline without implementation details.

In order to reach the design objective, requirements have been set up in the following section. These form a basis for the design criteria in the subsequent section. In Section 3.4, the platform choice is explained. The section after that describes how the design was divided in parts which were separately explored and evaluated according to the established design criteria. Each chosen part solution was combined for the design which is subject of the last section in this chapter.

## 3.2 Requirements

As a first step towards a design, requirements need to be deduced from the use case mentioned earlier. As this use case is about an inspection drone, the design has to be implemented on a small, lightweight platform that fits on a drone. As this project is about a video pipeline, the sensor is a camera. Consequently, the design must be able to process the data coming from the camera. That means a certain bandwidth must be available while the latency must be low as well. Given that this project aims for video streaming as basis for more complex designs, there should still be resources available after implementation.

One of the most important requirements for this project is the bandwidth, i.e. the amount of data that can be processed in a certain time span. It has a significant influence on performance of machine vision algorithms. Looking at our drone, higher bandwidth means a higher resolution and more pictures per second can be processed. As there is more information available for an algorithm to work on, accuracy improves. An example with a working application is Schmid et al. (2014) that successfully tests an autonomous drone in a coal mine while making use of a camera stream with 18.75 MB/s. In the mentioned case, it means two cameras taking 15 pictures per second with a resolution of 750x480 pixels. This design aims at 18.75 MB/s which is equivalent to 30 pictures per second with a resolution of 750*480 pixels from a single camera. There are camera modules available with this amount of data output. Therefore, these numbers were chosen as a requirement for this project.

Another important characteristic for video streaming is latency which is the time from a picture being taken to the processed result. This latency is relevant for an autonomous drone because it influences the time between recognizing for example an obstacle and appropriate course correction to avoid the obstacle. This makes the latency also a critical characteristic because a low latency can help avoid accidents. Logically, low latency is desirable. The precise relation between latency and performance of an autonomously navigating drone is hard to derive analytically. Therefore, a latency of 0.3s was considered sufficiently low for this design.

This design is meant as a basic starting point for machine vision applications. Therefore, there still has to be the possibility to add algorithms to the implemented design. This means that the chosen platform has to offer resources which can be used for later additions to the design.

As the end product is for drones and other robotics projects, this design has to be implemented on a platform that is small and light enough to fit on a drone. Nikolic et al. (2014) presents a module performing a SLAM algorithm fused with inertial data. It was tested in Omari et al. (2014) and is light and small enough for a drone. The mentioned module weighs 130 g and its dimensions are 144 x 40 x 57 mm. This size is used as a requirement for this project to make sure that the result fits on a drone. An overview of all requirements can be found in Table 3.1.

| **requirement** | **number** |
|---|---|
| bandwidth | 18.75 MB/s |
| latency | 0.3 s |
| weight | 130 g |
| size | 144 x 40 x 57 mm |

**Table 3.1:** requirements

## 3.3 Design criteria

In order to evaluate the considered solutions in the following sections, criteria are chosen that are relevant for the design. Each possible solution gets a certain number of points for each criterion. Additionally, each criterion gets a weighting factor corresponding to the importance for the design. Points get multiplied with the related weighting factor, the sum of all points for a solution gives its score. All design criteria are in Table 3.2.

For the design process, the time it takes to build and implement the design is quite important because time is limited and it is hard to accurately plan a schedule.

Bandwidth counts just as much as this is the criterion where the hardware acceleration should be noted the most. Therefore, the objective is as well to make bandwidth a strong point for this design.

Latency has been chosen as it is also part of the requirements. However, it is less important for the video pipeline because the result is not yet used for a critical process like in an autonomous drone.

The amount of resources available for this design are determined by the platform choice described in the following chapter, which is in itself a limiting factor. However, it is not supposed to be optimized for efficiency which is why the resources criterion has a low weighting factor. It is much more important for possible applications that might be designed in future research.

## 3.4 Platform choice

From the requirements described earlier, there are some that are especially relevant for the choice of a suitable platform. This platform needs to offer enough performance for this project as well as some extra resources for future algorithms. Additionally, the platform has to meet the weight and size limit in order to fit on a drone. And lastly, the platform must allow the combination of hardware and software as this is crucial for the approach mentioned in the

| design criterion | weighting factor |
|---|---|
| build time | 3 |
| bandwidth | 3 |
| latency | 2 |
| resource use | 1 |

**Table 3.2:** weighting factors of the design criteria

Introduction. With the relevant requirements in mind, we can now discuss which hardware accelerator is suitable.

One option introduced in Chapter 2 is to use an FPGA. It works like a reconfigurable digital circuit which has several implications. First, it allows to perform different tasks at the same time or process several data points simultaneously which is important for the established bandwidth requirement. Second, FPGA implementations can be optimized to keep latency low. Therefore, choosing an FPGA for this design would help to satisfy the latency criterium. Additionally, latency in an FPGA is deterministic which makes real-time applications possible.

The second option mentioned earlier was a GPU. GPU architecture makes it possible to process a lot of data in parallel because it is optimized for bandwidth. This has the downside that latency can be quite big and variable. Also, GPUs are rather easy to program. NVIDIA for example offers an API called CUDA which allows to use C-like code for programming (NVIDIA, 2019).

| advantage | FPGA | GPU | |
|---|---|---|---|
| parallelism | digital circuit | streaming cores | |
| latency | deterministic | - | |
| configuration | HDL | CUDA | |

**Table 3.3:** Advantages of using an FPGA or a GPU

The just mentioned advantages of FPGA and GPU were weighed to see which one is more suitable for this project. It was decided to go for a platform which incorporates an FPGA because that makes latency more manageable. Furthermore, potential GPUs usually are quite big and the few suitable platforms with a GPU are expensive. Therefore, a platform that uses an FPGA as a hardware accelerator seems like the best option. A summary of the advantages is shown in Table 3.3 and Table 3.4 shows the score of both options according to the established design criteria.

| solution/criterion | build time | bandwidth | latency | resources | score |
|---|---|---|---|---|---|
| FPGA | 1 | 3 | 3 | 3 | 21 |
| GPU | 2 | 2 | 1 | 2 | 16 |

**Table 3.4:** Possible platforms and their design criteria score

This said, a well-suited option turns out to be the DE10-nano SOC kit. It features an Intel Cyclone V SE SoC combining an FPGA with 110k logic elements and a dual ARM core. The amount of logic elements is sufficient because Nikolic et al. (2014) used a Xilinx Zedboard with 85k logic elements to implement SLAM which is more demanding than visual odometry. However, an Intel-based device was chosen over other vendors because of previous experiences with the software tools that Intel provides for development. Also, the platform falls within the size and weight requirements established in Section 3.2. Communication between hardware and software is expected to be fast enough because FPGA and CPU reside on a single chip.

## 3.5 Video Pipeline Design

Having discussed the requirements and platform choice for the design, the following section covers the video pipeline design itself. In this section, the design is split up in the blocks input, communication and output (see Figure 3.1). For each block, possible solutions are compared and evaluated according to the criteria in Section 3.3. In the last Section of this Chapter, all part solutions will be put together for the complete, chosen solution.

### 3.5.1 Data input

The first block is about acquiring data. A camera records a video and it needs to connect to an interface. This can happen either by connecting the camera to several hardware pins or by using the USB interface.

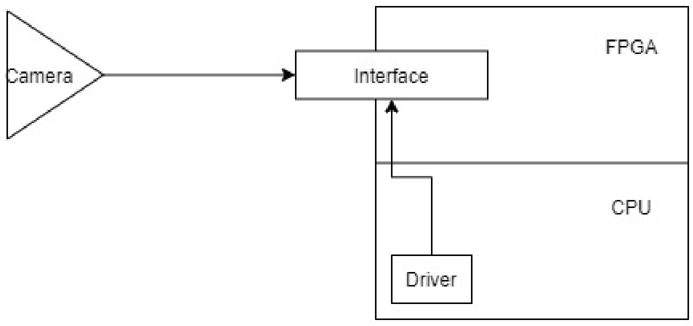

**Figure 3.2:** block diagram of data input with hardware interface

In order for the data to enter the system via hardware pins, a hardware interface has to be written in a hardware description language. It also requires a driver for software control of the data input (see Figure 3.2). Consequently, connecting the camera with hardware pins requires a lot of development work and build time. The upside is that performance is expected to be high. Taking the MT9V034 CMOS image sensor in a camera module as an example, our platform offers enough power for a hardware interface. The 50 MHz FPGA clock is sufficient to switch the input pins fast enough as the image sensor has a clock rate of 27 MHz. This ensures a high bandwidth while latency in this block is kept low because it is a digital circuit. As it is only necessary to switch pins and route the data to the next block, it does not require a lot of resources either.

Moving on to the second option which is a USB camera with a software interface. A Logitech C920 for example works out of the box with Linux and offers 62.3 MB/s of data. Using USB adds latency compared to the hardware interface because the operating system is responsible for that. It is not possible to use a real-time operating system within the available time for this project which means that latency is not deterministic and hard to control. However, it is impossible to measure the latency in this specific block only. Therefore, the latency gets a slightly lower score. Bandwidth is more than sufficient and gives a high score. This solution does not require a lot of resources as the driver is part of the existing kernel and expected to be quite efficient.

In the end, both solutions are quite similar. The USB camera is easier to implement. Manually writing a hardware interface that matches the timing of the camera module can be challenging. Nonetheless, if done correctly, latency is expected to be lower than with a USB camera. The score with applied weight factors is shown in Table 3.5.

| solution/criterion | build time | bandwidth | latency | resources | score |
|---|---|---|---|---|---|
| camera + HW interface | 1 | 3 | 3 | 3 | 21 |
| USB camera | 2 | 2 | 2 | 3 | 19 |

**Table 3.5:** Possible solutions for data input, weight factors applied accordingly

### 3.5.2 Communication HW/SW

Moving on now to consider communication between FPGA and CPU. There is a complex bus architecture on the DE10-nano SoC connecting all the different parts on the chip. Several bridges allow devices on the FPGA or the ARM core to function as master and initiate data transfers on the chip. They mainly differ in width and in which side is master and slave. For a simplified block diagram of the connections between FPGA and CPU see Figure B.2. For more information see Intel (2019b). Additionally, different parts on the chip can move data around. The CPU or the dedicated ARM DMA are options on the ARM core while it is also possible to implement DMA blocks on the FPGA fabric. In more complex designs the placement of this block is also important. However, in this design communication is by default after data input. The possible options are:

- Hardware DMA with the FPGA-to-HPS (Hard Processor System) bridge
- Hardware DMA with SDRAM bridge
- ARM DMA with HPS-to-FPGA bridge

For this part of the design, the bandwidth is very important. It is a part that does add overhead to the design but it is necessary to make use of the hardware accelerator. Therefore, it is a potential bottleneck when the implementation is not performing well. This is also restricted by platform choice.

There are several possible ways to move data from one place to another on this platform. The simplest method is to use the CPU for that. However, the CPU usually has a lot of tasks to do and using a DMA controller improves overall performance. Therefore, only DMA options were considered for this design. The ARM core has an integrated DMA controller which is "primarily used to move data to and from other slow-speed HPS modules" (Intel, 2019a). Another chip from the same device family was tested here (Land et al., 2019) where a bandwidth of 28 MB/s was mentioned. That is much lower than the 100 Gb/s peak bandwidth advertised on the Intel website (Intel, 2018). A DMA controller implemented on the FPGA can be a way to improve communication bandwidth between FPGA and CPU. Quartus Prime comes with a normal DMA controller and a scatter/gather controller as IP cores. The Intel Cyclone V design guidelines (Intel, 2019a) recommend to use the scatter/gather controller.

There are not only several relevant DMA options but more data bridges as well to choose from. The first option is the FPGA-to-HPS bridge which allows communication between FPGA and CPU. In this case, it can enable a DMA controller in the FPGA fabric to move data to and from memory connected to the ARM core. It has a width of up 128 bit and a special port for cache-coherent memory access. It is expected to be fast enough for this design because the design guidelines recommend this bridge for data transfers (Intel, 2019a). However, documentation does not mention expected bandwidth because it always depends on the particular design as well. Another interesting bridge is the lightweight HPS-to-FPGA bridge which is suitable for control signals. Most devices implemented on an FPGA have control registers which can be accessed by software. Using the lightweight bridge only for control signals helps keep latency down because data traffic is routed through a different bridge. There is also a counterpart to that bridge that allows the ARM core to initiate data transfers. The HPS-to-FPGA bridge is similar to the just mentioned bridge except that master and slave are different. The last option is the FPGA-to-SDRAM bridge which allows an FPGA master direct access to the memory controller without involving the L3 interconnect. According to the design guidelines, this bridge offers the most bandwidth while keeping latency low. It does not offer cache-coherent access and it is harder to set up.

After having discussed the available bridges and DMAs that are relevant for this design, the solution for this design block is discussed. As data moving device, the scatter/gather DMA was selected because it is recommended by (Intel, 2019a) and it is expected to be much faster than the ARM core. Additionally, the ARM DMA might be more useful when there are peripherals used that are directly connected to the ARM core. Together with this DMA, the FPGA-to-HPS bridge is most suitable as it is not hard to implement and should offer enough bandwidth. In this case, the solution with the SDRAM bridge has the same amount of points but still the solution with a higher build time score was chosen because of lack of time.

| solution/criterion | build time | bandwidth | latency | resources | score |
|---|---|---|---|---|---|
| HW DMA + FPGA-to-HPS bridge | 2 | 2 | 2 | 3 | 19 |
| HW DMA + SDRAM bridge | 1 | 3 | 2 | 3 | 19 |
| ARM DMA + HPS-to-FPGA bridge | 2 | 1 | 1 | 3 | 14 |

**Table 3.6:** Possible solutions for HW/SW communication, weight factors applied accordingly

### 3.5.3 Output

This part of the design is about showing the results of earlier executed image processing. There are two options on the DE10-nano. The board comes with an HDMI output that can be used to show the current image. Alternatively, relevant data like measured bandwidth or latency can be displayed in a text interface. The possible options are:

- images and diagrams via HDMI
- text interface

The HDMI output is more versatile as it can present information in different ways. Written text and numbers can be displayed as well as processed images or diagrams. However, it is harder to set up on the DE10-nano because one has to manually connect all pins on the board and write a hardware interface for it. There is an HDMI controller on the board but there is no ready-made interface available that allows the use of this controller. On the other hand, a text interface is really simple to make and can display all the relevant data. See Table 3.7 for the score.

| solution/criterion | build time | bandwidth | latency | resources | score |
|---|---|---|---|---|---|
| SW console text | 3 | 2 | 2 | 3 | 22 |
| HW HDMI | 1 | 3 | 3 | 2 | 20 |

**Table 3.7:** Possible solutions for output, weight factors applied accordingly

## 3.6 Comparing solutions

Thus far, each part of the design has been discussed and they can be put together. For the data input, the HW interface scores more points than the USB camera because it offers more performance. The data is then sent to RAM by a scatter/gather DMA via the FPGA-to-HPS bridge. The results can be seen on a text interface. Figure 3.3 displays the solution in a block diagram.

According to the established design criteria, this is the best design among the considered options.

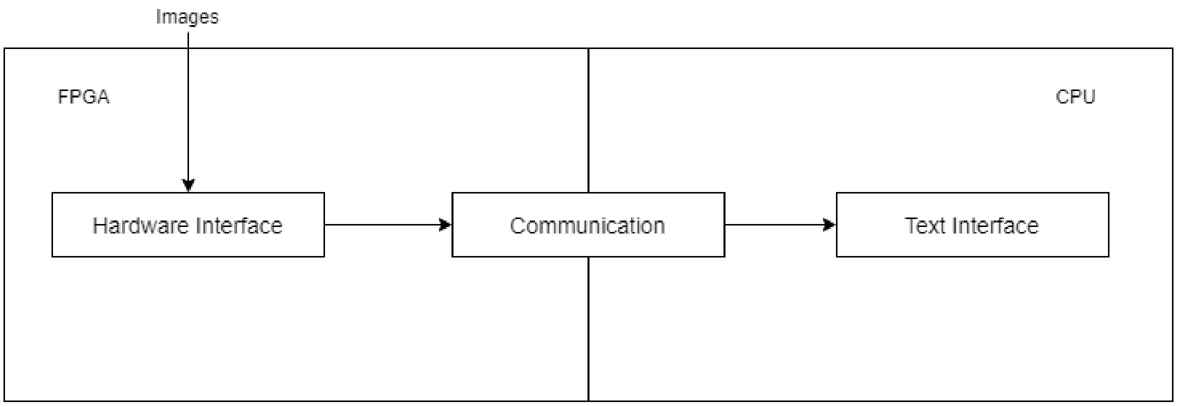

**Figure 3.3:** block diagram of the solution

As can be seen in this Chapter, the design is complex and its implementation time-consuming. This makes implementation challenging because available time is limited. Therefore, it was decided to only implement the communication block. It is a vital part of the video pipeline and its performance is a good indicator for the overall pipeline performance.

# 4 Testing

## 4.1 Introduction

This chapter describes the tests. As mentioned in the previous Chapter, the communication block was implemented and its performance measured. As it is a big part of the proposed design, this gives an indication of the overall design performance. Additionally, the results show if the platform is a suitable choice. In this case, bandwidth and latency are measured as performance indicators while area on the FPGA and CPU usage show the resources used. These results show if the proposed design is relevant for future research.

## 4.2 Setup

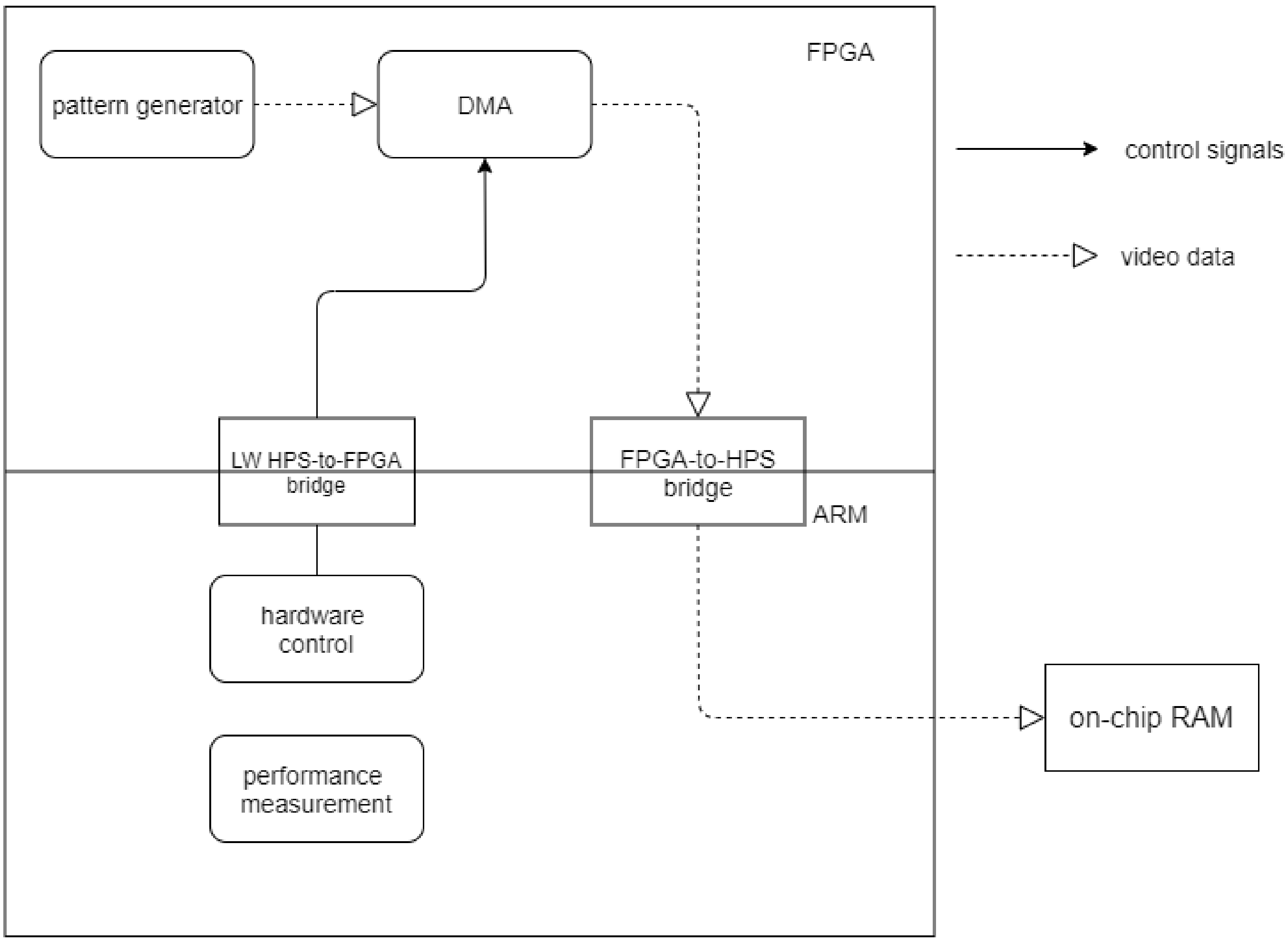

**Figure 4.1:** block diagram of the test setup

The design block described in Section 3.5.2, which was implemented for testing, moves data between the FPGA and the CPU part of the board. A scatter/gather DMA was selected to do the actual copying of data. It is controlled by software which sends commands via the lightweight HPS-to-FPGA bridge. The FPGA-to-HPS bridge is used for transferring data from the FPGA to the on-chip RAM on the CPU.

This design block was expanded with a data source to simulate a camera taking pictures. This data source realized on the FPGA side is an IP core included in the Quartus software which generates certain data patterns and streams them to the DMA. Data can then be sent from the

FPGA to the CPU's on-chip RAM. Then, all necessary measurements are done in software as well as a text interface which shows the results. An overview of the setup is shown in Figure 4.1.

## 4.3 Execution

Software has been written that controls all the mentioned peripherals and gets all necessary measurements. First, the data generator and the DMA are prepared. Then, a descriptor is constructed which contains information about the following data transfer. A clock is started and right after that, the data transfer starts. As soon as the DMA is not busy any more, the clock stops. The measured time is used to calculate bandwidth. After that, several small data transfers are executed and measured in the same way. The average of the measured times is the latency for a data transfer.

For the bandwidth measurement, 64 kB of data are sent to the on-chip RAM. In the time measured, several things happen to make the data transfer possible. The software checks if the DMA can accept another descriptor. If so, the descriptor is sent to the DMA. Subsequently, the DMA dispatcher is activated and starts the transfer. After that, the software waits until the DMA stops sending a busy signal (see Figure 4.2). The bandwidth is the amount of transferred data divided by time. In comparison, the latency measurement works similar as the same things for a data transfer have to happen. However, only 2 kB of data are sent.

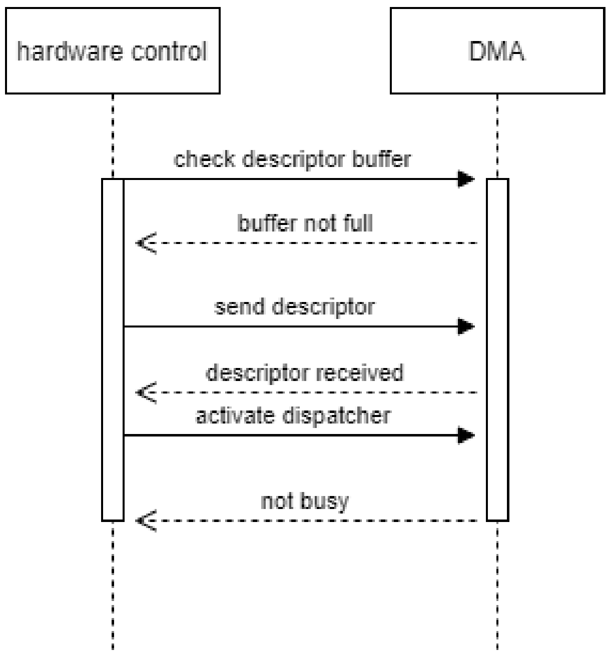

**Figure 4.2:** sequence diagram of the bandwidth and latency measurements

The last measurements that are discussed here, are measurements concerning the amount of used resources in this implementation. For the FPGA, the amount of used LEs and other blocks are read from the Quartus Prime synthesis report. For the CPU resources, the Linux command time is used. It measures the execution time of a command, the CPU time spent on it and the CPU usage. These measurements give an indication about the possibility of extending the proposed design.

## 4.4 Results

The bandwidth has been measured with different burst sizes with each series being measured 20 times. The results are shown in Table 4.1 and Figure 4.3 shows the measured bandwidth with a burst size of eight. The latency was measured 200 times in total with a burst size of one. Table 4.2 shows the results. In both tables, averages and standard deviation were calculated with all values and adapted average and adapted standard deviation excluding outlying measurements. Quartus reports that 6072 adaptive logic modules (modern logic elements) were used which is 14% of the available ALMs. According to the time command, it takes 0.01 s to execute the code which takes 52% of the CPU. Table 4.3 shows the used resources for the implementation.

| **bandwidth in MB/s** | | | | | |
|---|---|---|---|---|---|
| burst size | 1 | 2 | 4 | 8 | 16 |
| minimal | 328 | 325 | 452 | 383 | 452 |
| maximal | 443 | 452 | 461 | 464 | 463 |
| average | 428 | 441 | 458 | 457 | 459 |
| standard deviation | 24 | 28 | 3 | 17 | 3 |
| adapted average | 433 | 448 | 458 | 461 | 459 |
| adapted standard deviation | 7 | 3 | 3 | 1 | 3 |

**Table 4.1:** bandwidth measurements

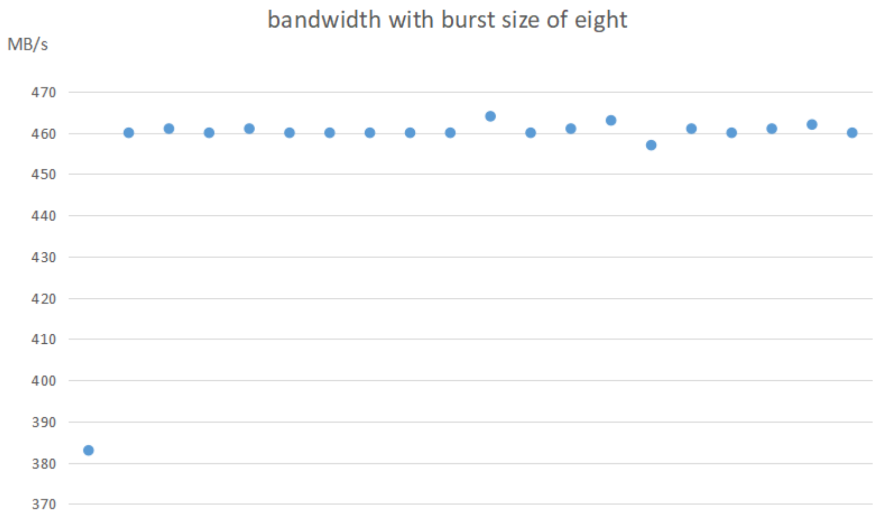

**Figure 4.3:** bandwidth measurements with a burst size of eight

| | **minimal** | **maximal** | **average** | **standard deviation** |
|---|---|---|---|---|
| latency in ns | 74,110 | 111,060 | $76.6 \times 10^3$ | $2.5 \times 10^3$ |
| | **adapted average** | **adapted standard deviation** | | |
| latency in ns | $76.5 \times 10^3$ | $0.5 \times 10^3$ | | |

**Table 4.2:** latency measurements

| | |
|---|---|
| area | 6072 LEs |
| | 14 % |
| execution time | 0.01s |
| CPU usage | 52% |

**Table 4.3:** resources used

# 5 Discussion

The objective of this project is to design a video pipeline suitable for drone applications. A combination of hardware and software has been used to achieve high performance that fits on a drone. In this chapter, the measurements from the previous chapter are discussed to evaluate if the objective has been reached.

First of all, all measurements satisfy the previously established requirements. It is noteworthy that the measured bandwidth is about 25 times the required bandwidth (see Section 3.2). Also, the measured latency is much lower than the latency stated as requirement. Additionally, there are resources left to complete the video pipeline. As all requirements are met, the measurements suggest that the proposed design is feasible.

Aside from how the measurements relate to the requirements, the bandwidth measurements show some peculiarities. When looking at the adapted average, choosing a burst size of eight is the best choice for the proposed design. When the average including all values is the deciding characteristic, a burst size of 16 should be chosen. However, that is apparently because the series with a burst size of 16 does not have an outlier. Figure 4.3 shows a series of measurements with the outlier right at the beginning. Outliers are not exclusively occurring at the beginning of the measurement. The outliers do increase standard deviation by several factors but there are so few that the average does not change a lot. It is not clear what the cause of the outliers is. A possible reason is that the operating system interrupted the user code during the bandwidth measurement.

The latency measurement has only one outlier and therefore it has almost no influence on the average while the standard deviation changes by a factor of five. Here, all values were measured with a burst size of one as it simplified the measurement. A higher burst size might lower the latency because the DMA can transfer more data without interruptions.

While the measurements satisfy the requirements, it is important to look at how meaningful they are. Several facts speak against these measurements being meaningful:

- only one part of the video pipeline has been implemented and tested
- bandwidth and delay measurements include overhead like the control sequence for the DMA
- the bandwidth has been measured by transferring 64 kB at a time to the on-chip RAM because of technical issues; results might be different transferring more data to the SDRAM
- reading data from the on-chip RAM and transferring it to memory on the FPGA might lead to a different bandwidth

There are also some reasons that speak for these measurements being meaningful:

- the implemented design part is the biggest part in the design
- even though reading from the on-chip RAM was not tested, it is very similar to writing to it and bandwidth is expected to be similar
- in case of bandwidth and latency, requirements are exceeded a lot
- overhead from measuring time expected to be small compared to transfer time

After considering these facts, it is still reasonable to believe that the proposed design is feasible.

In the Introduction, it was stated that autonomous drones are a valuable research topic and this project is a first step towards that application. Therefore, it is interesting to discuss if this design might also be extended to a visual odometry module. On one hand, the measurements suggest that the proposed design is feasible and there are resources left to implement a bigger design. On the other hand, the difference between the implemented part of the design and a module performing visual odometry is quite big. This means that it is impossible to conclude anything about a visual odometry module with the information currently available.

The approach to combine hardware and software was chosen to increase performance. There is no conclusive evidence that it did or did not work. The measured bandwidth exceeds the requirements but there is no pure software solution to compare it to. Also, the implemented block (see Subsection 3.5.2) would probably not be necessary when all calculations are done by a single CPU because all the acquired data would stay in main memory. However, combining hardware and software did increase the complexity of the project. There are more options on how to solve a problem but also more information and experience is needed to make an informed decision. A lot of practical experience was acquired this way. However, implementation for testing took longer than expected.

As the complexity increases because of the chosen approach, so increases the necessary knowledge to develop a good design. Adding hardware to it required hardware design knowledge. Additionally, drivers were necessary to make software and hardware work together. This also meant that development and implementation required more time. Furthermore, debugging was much more complicated as low-level details in an FPGA design are hard to observe but can be crucial for a design. In conclusion, the original design objective was very ambitious and had to be limited in order to finish within the available time.

# 6 Conclusions and Recommendations

## 6.1 Conclusions

This project set out to propose a video pipeline design that might be used as a starting point for machine vision applications. As discussed in the previous chapter, a design has been proposed and tested. The results suggest that the design is feasible, but only a part of it has been implemented. Therefore, the performance of the implemented video pipeline might be different from the part in the conducted experiment. Additionally, the project provides insight into video pipeline design with limited available resources. It is suitable for further studies and, eventually, applications.

Another project goal was "to integrate hardware and software in a beneficial way" (see Chapter 1). Both hardware and software are used in the design. Therefore, this goal is also achieved. However, it is unclear how combining hardware and software influences the performance of the video pipeline. Nonetheless, the proposed video pipeline is a good starting point for machine vision applications with a similar design approach.

## 6.2 Recommendations

A natural progression of this work is to implement the complete video pipeline design and test the performance. For the experiment, a camera interface can replace the pattern generator and additional software is needed for data transfers to main memory. Then, the performance can be measured again to see if the new results confirm or disprove the conclusions in this project.

A further study could extend the proposed video pipeline to assess if it is suitable for visual odometry. For that, several image processing algorithms like feature detection and matching can be added to the design and measure the resulting performance. Implementing such a design would show if the currently chosen hardware and approach is suitable for an application including visual odometry.

# A How to use the DE10 NANO SOC kit without an OS

## A.1 Requirements

- Quartus Prime Software 18.1
- Intel SoC FPGA Embedded Development Suite 18.1

## A.2 Process

- Compile your hardware project with Quartus Prime
- Generate the header file with all memory addresses derived from Platform Designer file
- Convert .sof output file to .rbf file with the following command:

```
$ quartus_cpf -c *.sof *.rbf
```

- Download the software example Altera-SoCFPGA-HardwareLib-Unhosted-CV-GNU from the Intel website
- Compile with Eclipse DS-5
- Start the preloader generator with

```
$ bsp-editor
```

  in the embedded command shell
- Disable watchdog, enable boot from SD, enable FAT support, disable semihosting
- Use make command to build preloader
- Use make uboot to build bootloader image
- Generate bootloader script file with

```
$ mkimage -T script -C none -n 'Script␣File' -d
  u-boot.script u-boot.scr
```

- Prepare SD card with an "a2" partition for the preloader and a FAT32 partition for your hardware project, bootloader and software
- Copy preloader in "a2" partition and all other files to the FAT partition
- Put SD card in board, turn on and connect to serial console

## B De10-nano SoC-board

The chosen platform for this project is the DE10-nano development kit. It is based on the Cyclone 5 SE 5CSEBA6U23I7 chip which combines an FPGA and an ARM core. As shown in Figure B.1, there are a lot of connectors and peripherals connected to the chip which makes this board versatile and powerful.

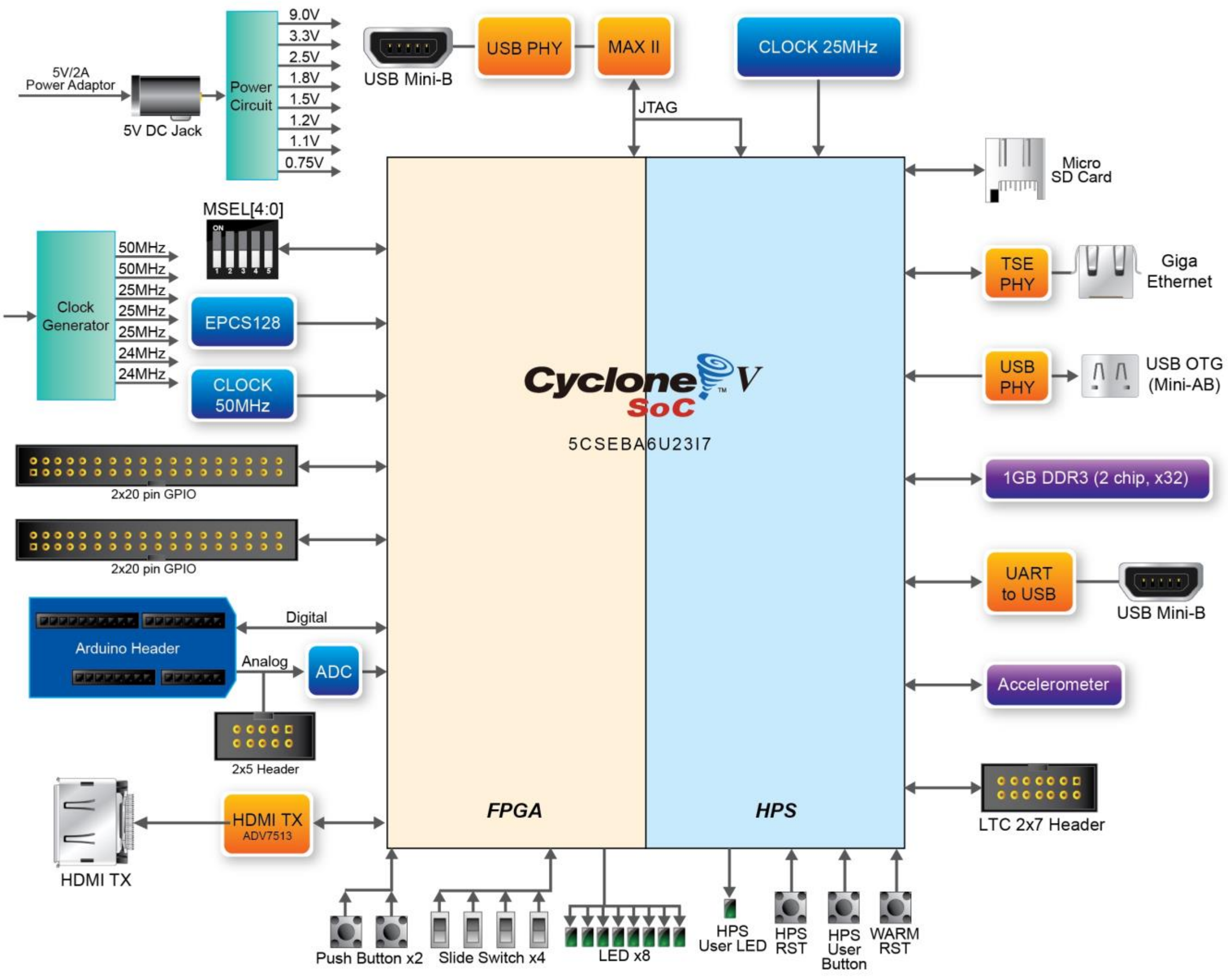

**Figure B.1:** block diagram of DE10-nano (Terasic, 2017)

The FPGA features 110k logic elements and about 6 kB of dedicated RAM. There is a USB Blaster port connected to it for programming. 40 GPIO pins are available as well as extra pins similar to the Arduino header. There are several 50 MHz clock sources that can be combined with PLLs to increase clock frequency. The HDMI can be used for output directly to a screen.

The processor on the chip is a 800 MHz dual-core ARM Cortex-A9. It has access to 1 GB of DDR3 RAM. There is an ethernet port, a USB interface and a micro SD card slot for an operating system.

Figure B.2 shows the interconnect between the microprocessor subsystem(MPU), FPGA and peripherals on the chip. Of special interest are the bridges connecting the L3-interconnect with the FPGA portion. The lightweight HPS-to-FPGA bridge offers little bandwidth and low

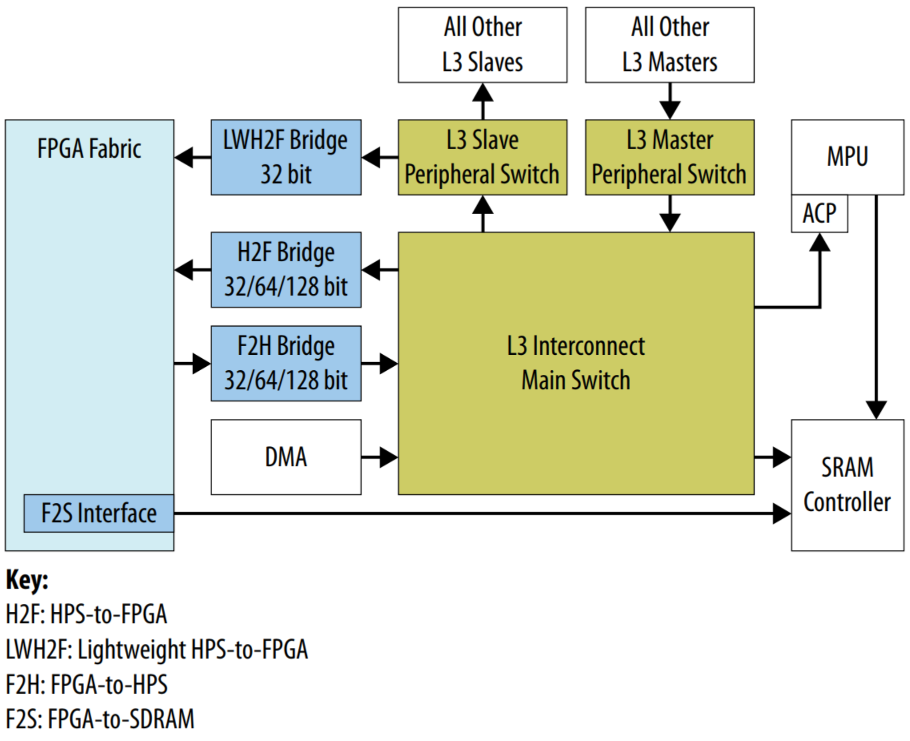

**Figure B.2:** simplified block diagram of connection system between HPS and FPGA (Intel, 2019a)

latency. This bridge is suitable for control signals from software to synthesized hardware on the FPGA portion. The other two bridges offer a wider interface and more bandwidth, i.e. are more suitable for sending data.

There is one last connection between the FPGA portion and the SDRAM controller subsystem. It allows any synthesized hardware access to main memory and is even wider than the other bridges. The FPGA-to-SDRAM interface therefore offers the most throughput and lower latency than the other data bridges. The downside is that it only offers non-cacheable memory access.

## C Software

The ARM core is powerful enough to run an operating system. In this case the embedded Linux distribution Angström was used. Intel provides a meta layer for the DE10-nano together with a build recipe. Therefore, the yocto project was used to build an image for the board. The recipe includes a preloader, the bootloader uboot and it generates a device tree for the kernel. The kernel version 4.9 is built as well as a root filesystem.

For the hardware design, Intel Quartus Prime 18.1 was used. Already implemented blocks in the platform designer - previously known as Qsys - were used as much as possible. All the hardware projects were based on the example design provided by the board designer Terasic.

Software development was done on the ARM DS-5 IDE as part of the Intel SoC Embedded Development Suite. This includes some libraries and drivers for Cyclone 5 chips as well as the cross compiler arm-linux-gnueabihf-gcc 5.4 provided by Linaro.

# D Visual Odometry

## D.1 Introduction

Visual Odometry is a way to track the motion of a drone or a robot. A camera on a drone sees the environment move when the drone itself moves. These changes can be used to estimate the drone's motion. There are different ways to do this. The feature-based method which looks at certain points in consecutive images is relevant for this project. By tracking these points and observing their movement, it can be deduced how the drone moved.

What makes Visual Odometry relevant for this project is that it can be used for path planning. A camera is the only sensor necessary to track a drone's movement. As cameras can be very small, they fit easily on a drone. Enabling a drone to track its own position would also allow it to plan its own path; therefore, developing a video pipeline on a lightweight processing platform is a step towards autonomous drones (Yousif et al., 2015).

There is a range of research for VO. Weiss et al. (2011) extended VO and used a SLAM algorithm to let a drone take off, fly to waypoints and land autonomously. Dunbabin et al. (2005) built an underwater vehicle to inspect for coral reef inspection. To circumvent the lack of GPS signal underwater, VO was used for navigation. Nistér et al. (2006) proposed a VO system to let ground vehicles successfully navigate on their own. The most famous example of VO is probably the Mars Rovers Spirit and Opportunity. In Maimone et al. (2007) and Cheng et al. (2006) the VO algorithm was used to make navigation more accurate, especially when the terrain made wheel odometry unreliable.

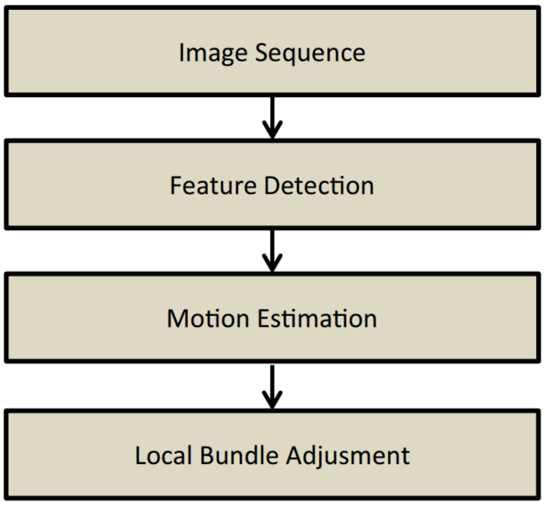

**Figure D.1:** Essential components of Visual Odometry

## D.2 Process

As shown in Figure D.1 this process starts with a sequence of images. The first step is the detection of points of interest. These features can be simple like corners or more elaborate like windows. As an example of a feature detector the FAST algorithm shall be explained soon. Once, all features in two or more consecutive frames are detected, they need to be matched. A criterion like the Sum of Square Differences or the Sum of Absolute Differences can be used as a measure of similarity.

When a sufficient number of corresponding features in consecutive frames is found, a transformation between frames can be calculated. In the ideal case that 3D coordinates of said features are known, the following formula can be applied:

$$\mathbf{T} = argmin_T \sum |\mathbf{X_i} - \mathbf{T}\acute{\mathbf{X}}_\mathbf{i}|^2 \tag{D.1}$$

**T** is the transformation between one frame and the next, **X** is one 3D point observed in the current frame and $\acute{\mathbf{X}}$ is the corresponding 3D point in the previous frame. The necessary amount of feature pairs depends on the system's degrees of freedom and the type of camera model applied. More feature pairs increase the accuracy of the transformation as well as the computational effort.

In cases where image points are compared with triangulated 3D points from the previous frame, a re-projection function can be used. When there is no 3D information available yet, like in the first two frames, epipolar geometry can be exploited to get a transformation from one frame to the next. These methods are described in more detail in Yousif et al. (2015).

## D.3 Advantages

As mentioned earlier, VO is a way to measure egomotion. However, there are also other methods with each their set of advantages. Compared to methods like LIDAR (light detection and ranging sensor) or INS (inertial navigation system), a camera is quite cheap and delivers a lot of information which allows accurate trajectory estimates, with relative position error ranging from 0.1% to 2% (Scaramuzza and Fraundorfer, 2011).

Unlike GPS, it does not need clear sight of the sky. Therefore, it works inside and outside. Nonetheless, not all environments are particularly suited for VO as lighting conditions or movements in the environment have a great impact on accuracy.

Furthermore, VO is not affected by slippery or loose terrain like wheel odometry is. It does not require additional signals or satellites which makes on-board solutions easier. Additionally, it can be combined with other vision-based algorithms as a camera is already present.

The biggest challenge for VO is the computational effort due to the amount of data that needs to be processed. Especially when high accuracy is crucial, extra optimisation steps are necessary that can be computationally expensive. While on a ground station, this might not be a big problem, small onboard platforms that fit on a drone, might struggle with this (Aqel et al., 2016).

## D.4 FAST

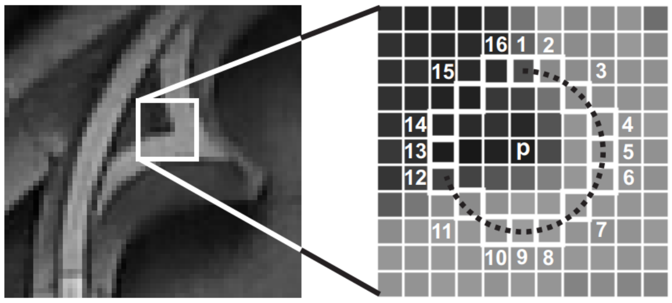

**Figure D.2:** Illustration of segment test

The Features From Accelerated Segment Test (FAST) algorithm is a feature detector, more specifically a corner detector. It places an approximated circle around a pixel in question such that 16 pixels lie on this circle as shown in Figure D.2. Then pixel intensities are compared. If there is a segment on the circle with enough pixels that are either lighter or darker than the centre pixel plus a threshold, then the centre pixel lies on a corner.

According to Yousif et al. (2015), FAST is computationally efficient which makes it suitable for embedded applications. Additionally, Kraft et al. (2008) successfully implemented it on an FPGA and found that it its implementation does not require a lot of resources and allows for a high frame rate. The main downside of this algorithm is that it is sensitive to noise. An alternative algorithm to solve this problem would be the SIFT (Scale Invariant Feature Transform) algorithm but it is computationally much more expensive (Yousif et al., 2015).

# Bibliography

Altera, C. (2011), *Cyclone 3 Device handbook*, volume 1, Altera Corporation.

Aqel, M. O. A., M. H. Marhaban, M. I. Saripan and N. B. Ismail (2016), Review of visual odometry: types, approaches, challenges, and applications, **vol. 5**, no.1, pp. 1897–1897, ISSN 2193-1801, doi:10.1186/s40064-016-3573-7. https://www.ncbi.nlm.nih.gov/pubmed/27843754

Betz, V. (2005), FPGA Architecture for the Challenge. http://www.eecg.toronto.edu/~vaughn/challenge/fpga_arch.html

Cheng, Y., M. W. Maimone and L. Matthies (2006), Visual odometry on the Mars Exploration Rovers - A tool to ensure accurate driving and science imaging, **vol. 13**, no.2, pp. 54–62, ISSN 1070-9932, doi:Doi10.1109/Mra.2006.1638016. <GotoISI>://WOS:000238068600011

Chu, P. P. (2006), *RTL hardware design using VHDL : coding for efficiency, portability, and scalability*, Wiley-Interscience, Hoboken, N.J., ISBN 047178639X 9780471786399 0471786411 9780471786412 1280448105 9781280448102, doi:10.1002/0471786411. http://ieeexplore.ieee.org/xpl/bkabstractplus.jsp?bkn=5237648

Cullinan, C., C. Wyant, T. Frattesi and X. Huang (2013), Computing Performance Benchmarks among CPU, GPU, and FPGA.

Dunbabin, M., J. Roberts, K. Usher, G. Winstanley, P. Corke, I. I. C. o. R. Proceedings of the and S. A. A. Automation Barcelona (2005), *A Hybrid AUV Design for Shallow Water Reef Navigation*, IEEE, pp. 2105–2110, ISBN 0-7803-8914-X, doi:10.1109/ROBOT.2005.1570424.

Farooq, U., Z. Marrakchi and H. Mehrez (2012), *Tree-based heterogeneous FPGA architectures : application specific exploration and optimization*, Springer, New York, NY, ISBN 9781461435945 1461435943 1461435935 9781461435938. http://books.scholarsportal.info/viewdoc.html?id=/ebooks/ebooks2/springer/2012-06-14/1/9781461435945http://link.springer.com/10.1007/978-1-4614-3594-5

Gao, H. (2017), Basic Concepts in GPU Computing. https://medium.com/@smallfishbigsea/basic-concepts-in-gpu-computing-3388710e9239

Intel, C. (2018), *Cyclone V Device Overview*. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-v/cv_51001.pdf

Intel, C. (2019a), *Cyclone V and Arria V SoC device design guidelines*, Intel.

Intel, C. (2019b), *Cyclone V Device Handbook*, volume 1, ALTERA CORPORATION.

Jeon, D., D.-H. Kim, Y.-G. Ha and V. Tyan (2016), Image processing acceleration for intelligent unmanned aerial vehicle on mobile GPU, **vol. 20**, no.5, pp. 1713–1720, ISSN 1433-7479, doi:10.1007/s00500-015-1656-y. https://doi.org/10.1007/s00500-015-1656-y

Kraft, M., A. Schmidt and A. Kasiński (2008), High-Speed Image Feature Detection Using FPGA Implementation of Fast Algorithm, in *Proceedings of the Third International Conference on Computer Vision Theory and Applications*, volume 1, pp. 174–179.

Land, B., J. Diaz, H. Ryan and A. Weld (2019), DE1-SoC: ARM HPS and FPGA Addresses and Communication Cornell ece5760. https://people.ece.cornell.edu/land/courses/ece5760/DE1_SOC/HPS_peripherials/FPGA_addr_index.html

Maimone, M., Y. Cheng and L. Matthies (2007), Two years of Visual Odometry on the Mars Exploration Rovers, **vol. 24**, no.3, pp. 169–186, ISSN 1556-4959.

Meier, L., P. Tanskanen, L. Heng, G. H. Lee, F. Fraundorfer and M. Pollefeys (2012), PIXHAWK: A micro aerial vehicle design for autonomous flight using onboard computer vision, **vol. 33**, no.1-2, pp. 21–39, ISSN 0929-5593, doi:10.1007/s10514-012-9281-4. <GotoISI>://WOS:000305227600003

Nikolic, J., J. Rehder, M. Burri, P. Gohl, S. Leutenegger, P. T. Furgale and R. Y. M. Siegwart (2014), A Synchronized Visual-Inertial Sensor System with FPGA Pre-Processing for Accurate Real-Time SLAM, Eidgenössische Technische Hochschule Zürich, doi:10.3929/ethz-a-010061790. http://e-collection.ethbib.ethz.ch/show?type=inkonf&nr=907

Nistér, D., O. Naroditsky and J. Bergen (2006), Visual odometry for ground vehicle applications, **vol. 23**, no.1, pp. 3–20, ISSN 1556-4959, doi:10.1002/rob.20103. https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20103

NVIDIA, C. (2019), *CUDA C Programming Guide*, NVIDIA Corporation, https://docs.nvidia.com/cuda/index.html.

Omari, S., P. Gohl, M. Burri, M. Achtelik and R. Siegwart (2014), Visual industrial inspection using aerial robots, in *Proceedings of the 2014 3rd International Conference on Applied Robotics for the Power Industry*, pp. 1–5, doi:10.1109/CARPI.2014.7030056.

Owens, J. D., M. Houston, D. Luebke, S. Green, J. E. Stone and J. C. Phillips (2008), GPU Computing, **vol. 96**, no.5, pp. 879–899, ISSN 0018-9219, doi:10.1109/JPROC.2008.917757.

Scaramuzza, D. and F. Fraundorfer (2011), Visual Odometry [Tutorial], **vol. 18**, no.4, ISSN 1070-9932, doi:10.1109/MRA.2011.943233.

Schmid, K., P. Lutz, T. Tomić, E. Mair and H. Hirschmüller (2014), Autonomous Vision-based Micro Air Vehicle for Indoor and Outdoor Navigation, **vol. 31**, no.4, pp. 537–570, ISSN 1556-4959, doi:10.1002/rob.21506.

Song, S., C. C. Guest and M. Chandraker (2013), Parallel, real-time monocular visual odometry, *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 4698–4705, ISSN 1050-4729, doi:10.1109/ICRA.2013.6631246.

Terasic, I. (2017), DE10-NANO Board Schematic. https://software.intel.com/content/www/us/en/develop/articles/de10-nano-board-schematic.html

Warren, M., P. Corke and B. Upcroft (2016), Long-range stereo visual odometry for extended altitude flight of unmanned aerial vehicles, **vol. 35**, no.4, pp. 381–403, ISSN 0278-3649, doi:10.1177/0278364915581194.

Weiss, S., D. Scaramuzza and R. Siegwart (2011), Monocular-SLAM–based navigation for autonomous micro helicopters in GPS-denied environments, **vol. 28**, no.6, pp. 854–874, ISSN 1556-4959, doi:10.1002/rob.20412. https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20412

Yousif, K., A. Bab-Hadiashar and R. Hoseinnezhad (2015), An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics, **vol. 1**, no.4, pp. 289–311, ISSN 2363-6912, doi:10.1007/s40903-015-0032-7.