Implementation of FPGA based star tracker pre processing pipeline

Oscar Björkgren

Supervisors, Åbo Akademi: Annamari Soini, Johan Lilius
Supervisors, Aboa Space Research Oy: Tero Säntti, Tuomo Komulainen
Master's thesis in Computer Engineering
Åbo Akademi
2021

Abstract

Keywords

Abbreviations

Contents

List of Figures

List of Tables

1. Introduction

A hardware based solution for a software algorithm is sometimes needed within application areas where performance matters the most. One example of this is the aerospace industry where the most optimized solution is sometimes the only choice. In this thesis I work together with Aboa Space Research Oy to create a hardware implementation of a pre-processor module for a star tracker system using a software component as a basis. A star tracker is commonly used as part of a larger system where the role of the tracker is to provide orientational data to other components. This information might be used for navigation and control or in combination with scientific instruments to help with further analysis. Hardware design can be inspired by software algorithms when designed for a specific purpose as in this case. The software defines the functional part of the system, which leaves a big part of the supporting architecture to be designed. In this thesis I will explain the process of implementing functionality to a digital circuit, by example.

To complete a set of tasks in any environment, a scientific project needs well defined tools to reach its goals. In technologically restricted environments such as space, tools often need to be multipurpose for optimal use. An image sensor is in this case used for providing reference data for a star tracker. This image sensor can provide data to multiple systems by reuse of data or capturing new images with other settings. The purpose of the system in this project is to refine data from predefined sensors to provide additional and reinforcing information about the environment for the use of other scientific instruments. The goal of this thesis is to create a roadmap of the system design process and include theory of relevant areas.

2. Custom hardware design in general

2.1 Single and general purpose processors

A general purpose processor refers to a hardware computing platform which is designed for universal use with broad benefits across different problem solving domains. Computing platforms such as microprocessors and CPUs are examples of this type of processor, their hardware logic is implemented in such a way that it enables a large variety of computations to be performed. The characteristics of a processor for general problem solving are well suited to common tasks where there might be many hard- and software abstractions between the application interface and hardware logic. This is a necessity in for example PCs.

When hardware is required for a single purpose with a finite set of tasks to be performed there will be drawbacks with using hardware designed with flexibility in mind. This could be compared to the use of a multitool for driving a screw into a piece of wood when in fact only a screwdriver is needed. It is not a wrong way to do it but there is a more optimized way of achieving the result. Any CPU could be used for computer graphics calculations but since GPUs are designed for the single purpose of this type of processing they are a better tool for the job.

Single purpose processors have a rich history with roots in the early ages of computers. Vector processors were for example used in supercomputers. This was before computers were seen as general purpose technology and high performance computing was about as common as ordinary PCs are now. Since recent years the utilization of specialized hardware has been rising, with new application areas evolving. Single purpose processors such as GPUs and field programmable gate arrays (FPGAs) have found themselves into areas such as machine learning, cryptocurrency mining, and other high performance computing applications. As
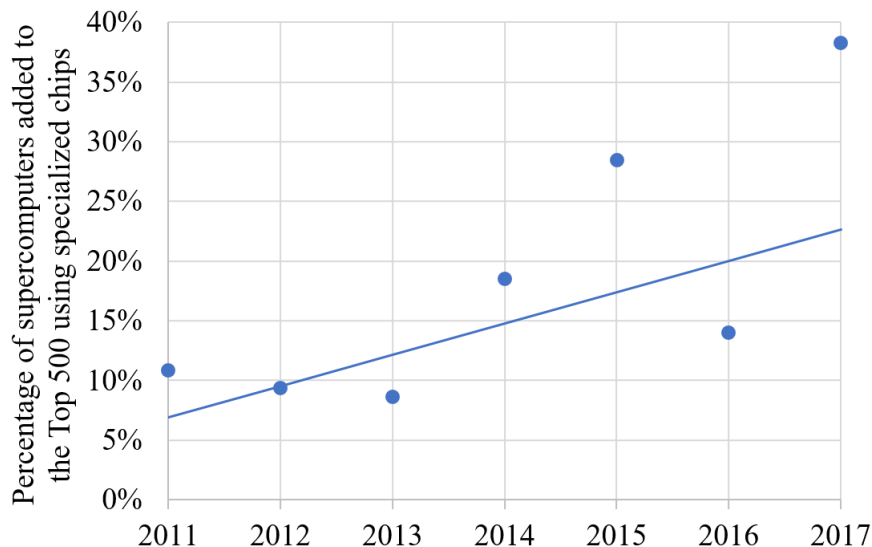
Figure 2.1: Graph showing increasing popularity of specialized hardware in high performance computing [1]

the hardware used in aerospace applications generally is specialized due to strict requirements, single purpose computing platforms are heavily used. For example, the NASA perseverance Mars rover relies heavily on Xilinx manufactured FPGAs for different tasks, such as image processing pipelines [8].

2.2 FPGA Overview

A field programmable gate array, or FPGA, is a digital integrated circuit that consists of millions of logic blocks that can be configured to perform different operations. The process of configuring the logic blocks is called FPGA design and can be compared to the design of integrated circuits. Reconfigurability and ease of design are the main advantages when using an FPGA as a computing platform. When comparing an FPGA design to a solution that is developed using software, the FPGA stands out with process pipelining and high a level of parallel processing. The difference between executing a software program on a processor and executing the equivalent operation on an FPGA is that the compiled software instructs the processor to do operations in a generic way, as in loading a register and shifting a bit. An FPGA does not need to explicitly load a register because the registers are by default connected to the input of the operation, which is defined by the FPGA design. A bit shift operation is typically an inexpensive
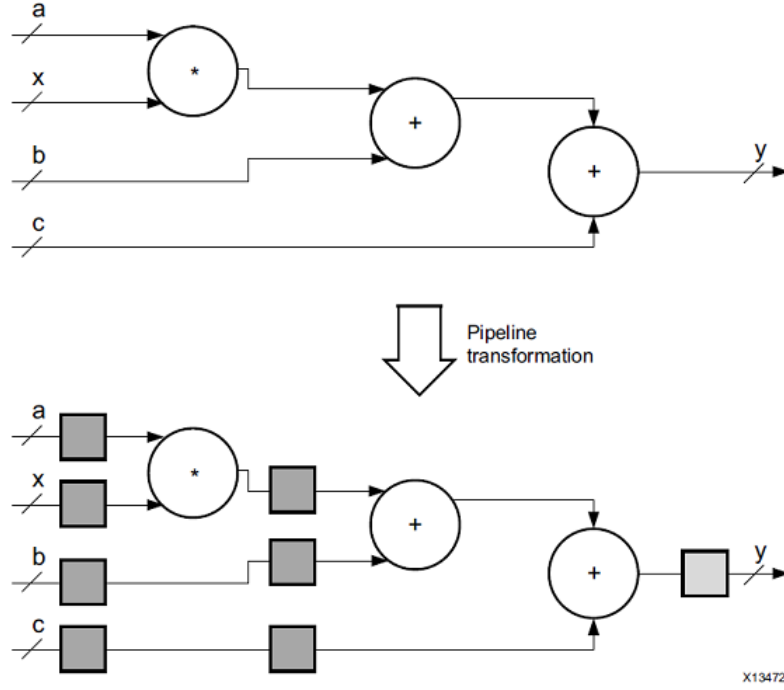
Figure 2.2: Figure showing two execution paths for the expression $y = (a \times x) + b + c$. The operations in the first path depends on the output from previous operations whereas the operations in the second path depends on a separate input registers. The second path is a pipeline transformation of the first path [2]

.

operation regardless of the computing platform, but the FPGA can pipeline the result directly into another register or operation, which saves clock cycles [2]. The pipeline principle is demonstrated in figure 2.2 by comparing it to a non pipelined solution. The pipeline transformation enables the inactive parts of the calculation to be utilized when they normally would idle, waiting for all parts to finish.

Xilinx is one of the key companies involved in FPGA production both currently and historically. They introduced the world to FPGAs in 1985 and have since then led the programmable logic technology industry with a 51% market share of programmable logic device suppliers in 2017. Major end market categories for Xilinx are aerospace, defense, and communications. The main competitor of Xilinx is Intel with a market share of 37% [9][10].

Due to the significance of Xilinx, and relevancy in this project, FPGA technology discussed in this work is Xilinx based. Differences between manufacturers includes technological and architectural nuances. For example, Xilinx exclusively

manufactures static random access memory (SRAM) based FGPAs. This refers to the technology of the programmable parts of the FPGA. Other available tehcnologies are flash and anti-fuse, which are less common than SRAM [11].

The basic internal elements of an FPGA consist of circuits such as look-up tables, flip-flops, wires, and input/output pads. Using these elements the FPGA design is implemented in the hardware. The architecture is explained by Figure 2.3 which shows a matrix arrangement of configurable logic blocks which are connected to each other with the interconnect, consisting of wires and switch matrices. The wires also run to the input/output pads, which enables interfacing with off-chip components such as synchronous dynamic random access memory (SDRAM) or sensors [2]. Configurable logic blocks (CLBs) contain the look-up tables and flip flops. An overview of a CLB is shown in Figure 2.4.
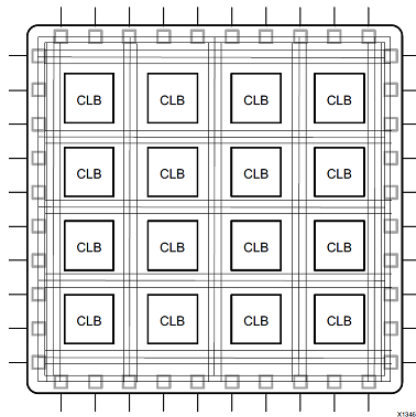
Figure 2.3: FPGA architecture. An FPGA consist of configurable logic blocks, interconnect, and input/output pads [2].

The slices in Figure 2.4 represent core elements of an FPGA. Slices consist of look-up tables, storage elements, multiplexers, and carry logic. Depending on the FPGA there are different numbers of slices per CLB. The slices are collectively responsible for implementing a significant portion of the FPGA design [3].

Look-up tables, or LUTs, are mainly used as logic elements in FPGAs. They can implement any boolean logic operation by combining memory cells with multiplexers. The look-up table is essentially a truth table that can be used for reading the result of boolean operations. Memory cells and multiplexer routings are initialized by the FPGA design, which is what makes the look-up table element configurable [2]. An example of a 4 memory cell wide LUT is shown in Figure 2.5.
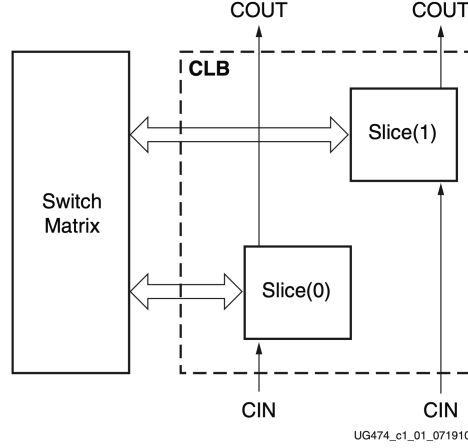
Figure 2.4: Diagram of a configurable logic block. A CLB consists of logic element containers, called slices, inputs, and outputs. Both slices are attached to the switch matrix, connecting the CLB to other CLBs [3].
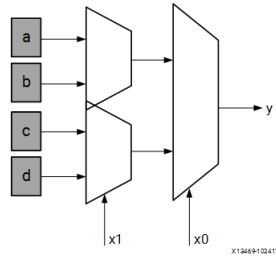
Figure 2.5: Look-up table with 4 memory cells and two multiplexers. The FPGA design initializes memory cells **a-d** and uses inputs **x1** and **x0** to select value **y** [2]

The circuit elements discussed in this chapter form the basic blocks of FPGA technology. A larger set of elements are often included in an FPGA to enable some optimizations and further functionality in the design. For example, there are different types of slices a CLB can contain, which makes it more suitable for storage purposes rather than logic. Signal processing pipelines are a popular use case for FPGAs and they often contain slices specialized for operations in this problem domain. Digital signal processing can be accelerated by parallelizing operations in a single instruction multiple data (SIMD) fashion, thus to maximize the advantage there can often be found a large number of these slices. Figure 2.6 shows an example of an FPGA configuration with some of these specialized blocks mapped out. The green CLBs are designated to memory usage and are closely coupled to the red digital signal processor (DSP) blocks to enable fast processing

of the stored data. The purple external memory controllers use the input/output pads to interface with external memory such as SDRAM [2] [3].
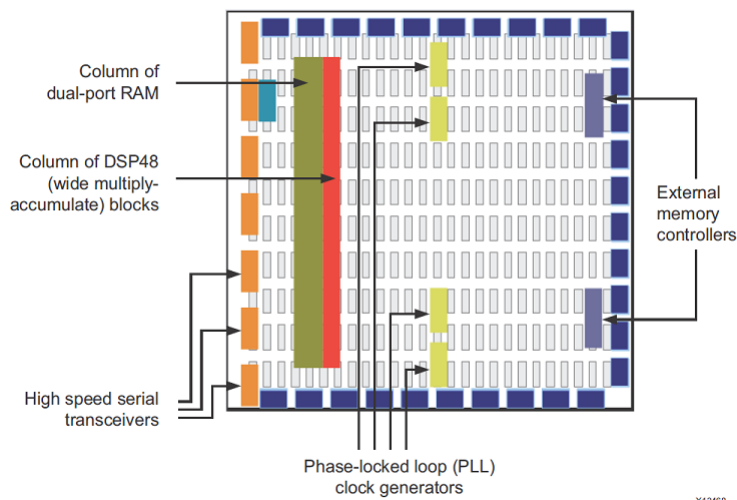
Figure 2.6: DSP system design on FPGA [2]

When comparing FPGAs with each other the main performance metrics are power consumption and number of logic cells. The number of logic cells determines the size and complexity of the design that the FPGA can fit. Additionally, the FPGAs can often be compared by specifications such as specialized CLB slices, IO interfaces, and communication rates. Clock speed is seldom compared because although there is an upper limit, it is often design specific [3] [2].

2.3   System design with HDL

The main tool for creating an FPGA design is a hardware description language (HDL). VHDL and Verilog are examples of commonly used HDLs. Compared to conventional computer programming languages, HDLs are used in a data flow fashion to describe the timing and logic of an integrated circuit. The data flow description of a circuit in an HDL is such that all code blocks are executed simultaneously by default. This is one of the main advantages of FPGAs as the system is inherently parallelized. It creates also one of the biggest challenges in HDL based design as it complicates the timing of events and resource concurrency.

The output of an HDL design is produced by a build process that computes the high level data flow description into low level circuit elements and routing.

This can be compared to the compilation phase of code written in a statically typed software programming language.

The synthesis process is where the HDL design known as register transfer level code is transformed to a gate level description called netlist. The netlist is a mapping of circuit components that are found inside the FPGA. The convenience of HDL based design is being able to describe the circuit at a higher level. In practice this means that registers and buses are created and connected to be inferred into low level circuitry at the synthesis phase. When developing a design against a specific FPGA, a bitstream file is created to configure the device with. The end product of an HDL based design is the bitstream file or, alternatively, the synthesis result as it can be used for simulation purposes by a test bench [12].

2.4   System design with IP cores

Designing a hardware system often involves components of a reusable nature, such as dividers, communication protocols, and memory controllers. Intellectual property (IP) cores are distributions of components such as these. An IP core can be of varying complexity as it can consist of a solution for a specific task or describe a complete system on a chip (SoC). IP cores are licensed through a patented design which is what the name refers to. They can often be configured to suit different properties of the implementation such as data width and clock rate. The core is then connected to the system using an HDL. The core itself may consist of software or hardware macros. The software macro, or soft IP core, is an HDL implementation of the IP core logic that is built to the specific FPGA with the rest of the system design. A hardware macro, or hard IP core is prebuilt and FPGA specific [13] [12].

The hardware sector and consumer electronics industry rely heavily on reusable IP cores. For example, the SoC or motherboard found in mobile phones often include advanced RISC machine (ARM) based processor cores licensed from ARM holdings. The producer of the mobile phone benefits from avoiding the development of the processing unit but still has the ability to customize the SoC to a high degree by combining different processing units, memories, and other IP cores [4] [14]. Figure 2.7 shows the architecture of a SoC which has an ARM based CPU, and other modules commonly found in mobile phones. In addition to the CPU, any of the remaining modules might also be licensed IP cores.

In aerospace applications, the advantages of IP cores are utilized in the same
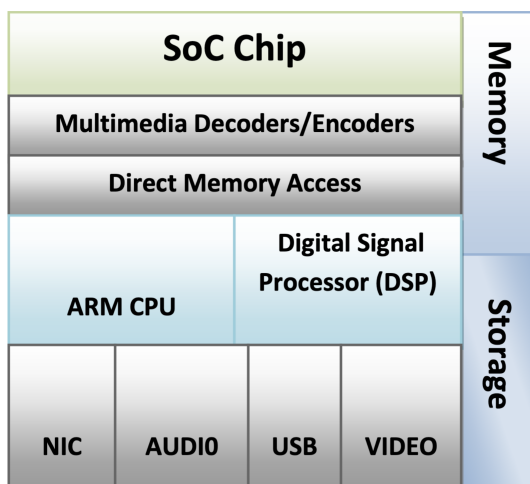
Figure 2.7: Common SoC found in smartphones includes ARM based CPU and a DSP. [4]

manner as within the mobile phone industry. Additional requirements such as fault tolerance limits, redundancy, and formal verification of the IP cores increase the advantages of their usage when comparing to in-house developed solutions. This, however, comes with pricier licenses compared to non-aerospace grade IP core designs. Also, due to the specialized requirements, lower competition in the market sector can be expected, increasing the prices even more [15].

In addition to FPGA products, Xilinx also provides design tools and IP cores to help with development on their FPGA platforms. Their IP core library ranges from common IC to complete microcontrollers. Some of the cores are available free to use through the Vivado design suite, and a majority of them are soft cores to allow compatibility across their FPGAs. Integration of the IP cores and other development is also done through Vivado [12].

3. System requirements

A hardware implementation of an image analysis software component is to be developed to be used on an FPGA. The software in question is a product of a European space agency funded research project regarding detection and analysis of space debris in low earth orbit. Using computer vision among other analysis, space debris is identified from image data. With the help of a star catalog the image is mapped to a position in space to provide positional data of the detected debris. A diagram of the system is shown in Figure 3.1. The software component relevant to this project is the Pre-processing & Segmentation part of the diagram, which is in the input phase of the complete system [5] [16].
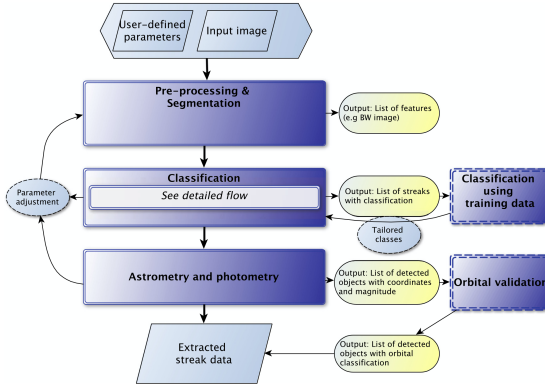
Figure 3.1: Flowchart of StreakDet software [5]

The star extraction process in StreakDet passes the input image through a series of algorithms to identify stars. The algorithms are well known image filters and analysis methods that in combination with each other produce a way to pick out objects from an image. The main task in this project is to identify these algorithms and fit them into a hardware design using a hardware description language.

Star extraction is a key stage of a star tracker system which uses a star catalog to match the extracted stars against. Using the star matches and catalog infor-

mation the position and orientation of the captured image can be determined. The goal of this project is to create a pre-processing pipeline to be used in a star tracker.

3.1 System specification

The specification consists of development and system requirements as well as a description of tools and components. System requirements are created based on meetings and discussions with Aboa Space Research Oy. Some requirements are derived from higher level requirements to suit the development process. Table 3.1 presents an overview of the system in a functional and non-functional requirements fashion.

Req. Id	Description
1	Client endpoint for FPGA module
1.1	Bitstream can be written to FPGA
1.2	Image processing pipeline parameters can be written to FPGA
1.3	Allows for debugging of system
1.4	Provides visual results with metadata
2	Validation and verification of star extraction
2.1	First stage validation and verification with software prototype
2.2	Second stage validation and verification on FPGA platform
3	Star extraction result and operation
3.1	Image coordinates of extracted stars are stored
3.2	Centroid of detected stars are calculated
3.3	Error handling to avoid false detection

Table 3.1: System requirements specification

3.1.1   System components

Processing nodes

The system consists of two nodes, the host machine and the target FPGA. Because the application of this system will be different in a production environment, the FPGA part should not be too dependent on the host machine. When integrated into a larger system with shared memories and communication, the FPGA will have to be able to read an input image directly from memory. The same applies to the output, the register where the output is written needs to be configurable and not sent to a host machine by default.

FPGA module

The FPGA module Opal Kelly XEM7305 chip with a Xilinx Spartan-7 FPGA will function as a testbed for development. The module contains a part from the FPGA external SDRAM memory and communication interfaces such as USB 3. These features allow for host machine communication and data storage of features such as images, star catalog, and output. These data could be stored on the FPGA itself but using the SDRAM is a sustainable solution because it allows for shared memory usage in production and more flexible image and star catalog sizes.

4. FPGA based computer hardware

Common computer hardware is the corner stone of any digital circuit that needs to perform common tasks such as storing and transmitting data. With the help of the basic elements of an FPGA discussed in Chapter 2, circuits of this nature can be created. Although an 8-bit adder circuit can be regarded as common computer hardware, the focus of this chapter is on larger circuits used as hardware modules such as memory and communication interfaces. The components discussed are essential building blocks for the image processing pipeline. Figure 4.1 shows an overview of these components on the XEM7305 FPGA module.

4.1   Communication

The system requirements state a communication channel to be available for shared memory usage and controlling the FPGA. Machine to machine communication is enabled through the interfaces available on the Opal Kelly FPGA module. The default interface on XEM7305 is USB C [6].

The advantage of the USB is flexibility and ease of use. Transfer speed is not a high priority in this case because of the relatively small size of data, which is less than a megabyte. The USB module on the XEM7305 contains a USB C port and a Cypress FX3 peripheral controller, shown in figure 4.1. The peripheral controller operates the data bus by implementing a communication protocol stack consisting of a physical, link, and protocol layer. It allows the usage of the USB communication link on the FPGA module by exposing a programmable interface. The programmable interface can be used by various systems on the FPGA module such as a image sensor or the FPGA itself. To allow the usage of the peripheral controllers programmable interface in the FPGA design, a hard IP
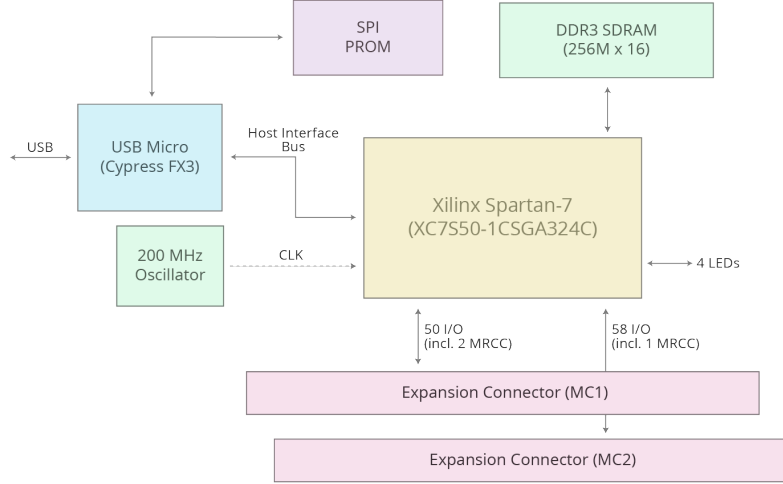
Figure 4.1: Overview of FPGA module showing the FPGA itself in the center, SDRAM off-chip memory on the top right, and USB interface on the left. [6]

core is provided. [6] [17].

Both the USB 3 standard and the FX3 peripheral controller supports serial and parallel communication but the hard IP core connecting to the FX3 interface supports only serial communication. This simplifies the use of the IP core, requiring less FPGA fabric, but also reduces the throughput [6] [17].

The internal communication mechanism of an FPGA consists the FPGA interconnect. The interconnect is the physical layer of the communication protocol stack leaving the rest of the stack to be implemented in the FPGA design with HDL or IP cores.

The limiting factor and bottleneck of the system is often some off-chip hardware module that places certain requirements on the communication, for example the USB or memory controller. When working outside these types of requirements, the throughput of the FPGA based system can be optimized by increasing the bit width between modules. When applying this to the image processing pipeline context, this might be practical within communication of image data. The implementation of this requires more memory cells and wires, which in turn increases the footprint of the FPGA design.

When comparing to a software program, the same optimization method of maximizing bit width might be considered. This, however, does not inherently result in a gain in throughput because conventional CPUs have a fixed bit width for their operations. When this limit is exceeded the operation needs to be per-

formed across multiple clock cycles which counteracts the wanted result. The bit width of an operation on an FPGA during a clock cycle is consistent with the HDL design.

4.2 Memory

To enable processing of image data, a set of storage mechanisms is necessary. The use of a certain memory type is mostly dependent on the size of the data and how it will be used. As in conventional CPU architectures, different levels of memory are used to satisfy these dependencies.

FPGAs are generally not well suited for storing a large quantity of data as it is considered inefficient use of available CLBs. Since customization is a key feature, the most standardized parts, such as memory, are often placed outside the FPGA. Off-chip memory allows storage of large data, such as images, freeing up CLBs to other usage such as computation and logic. The downside with off chip components is them often being a bottleneck in the system. This is due to a number of factors such as restrictions in clock speed, concurrency, and bus width. A larger FPGA with an increased number of CLBs could offset the need for off chip memory avoiding the bottleneck, with consequences being greater power consumption and physical size, potentially invalidating the solution.

An overview of the XEM7305 module and the memory types can be seen in figure 4.1 where the off-chip SDRAM memory is on the top right and the lower level memories inside the Xilinx Spartan 7 FPGA in the center.

4.2.1 SDRAM

One of the most common storage mechanisms is SDRAM which is useful to store data that is frequently used and ranges from kilobytes to megabytes in size. Key features include random access, meaning same access time for any part of the memory, and high bandwidth. It is often paired with FPGAs because it is next in the memory hierarchy after on chip storage mechanisms.

At the core of SDRAM lie capacitors trapping the charge representing a bit. An overview of a typical SDRAM circuit is shown in Figure 4.2. The capacitors are refreshed periodically to retain the stored data. This makes SDRAM a volatile memory, requiring power to keep its state. The advantage over closely related memory types such as flash memory, is that it does not degrade at the same

rate of read/write cycles and it is truly random access, giving the advantage to SDRAM as a volatile working memory in most FPGA applications [7] [18].
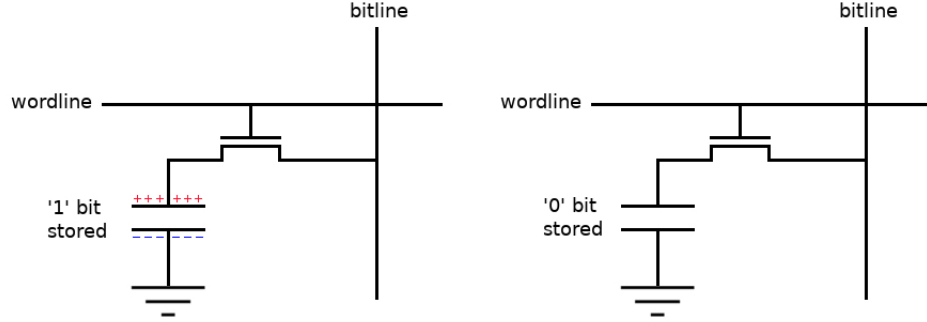
Figure 4.2: SDRAM memory cell. The capacitor traps the charge which is controlled by the transistor at the top [7].

The memory cells in SDRAM are ordered in rows and columns, forming an array. A collection of these arrays are referred to as memory banks. The number of rows, columns, arrays, and banks varies depending on the type of SDRAM, but is usually a power of 2. Figure 4.3 shows the geometry of an SDRAM with 4 banks with 4 arrays each. The data width of a memory is the number of arrays that are stacked inside each memory bank. When accessing a specific row and column, the number of affected bits is the same as the data width of the memory.

To increase SDRAM performance, double data rate (DDR) is used to access data at double the clock rate. DDR enables accessing a memory bank on both the rising and falling edge of the clock cycle. A prefetch mechanism enables the transfer of multiple bits concurrently per bank access, increasing the throughput even more. This mechanism enables efficient use of the internal SDRAM bus width by also transferring data from neighboring addresses. The gain comes from the fact that the time to access a single address is the same as accessing the neighboring ones as well. The prefetch mechanism gives an advantage to data which is structured in a way where each access is in adjacent addresses. The width of the prefetch data depends on the DDR version. Other variations between DDR versions are the supported clock frequency and burst length, which is a feature related to the prefetch mechanism. The burst length specifies the number of SDRAM words contained in each read or write operation [19].

The XEM7305 features a Micron DDR3 SDRAM chip with a 32-bit wide interface. The memory part has 8 banks, 32 000 row addresses, 1000 column

16
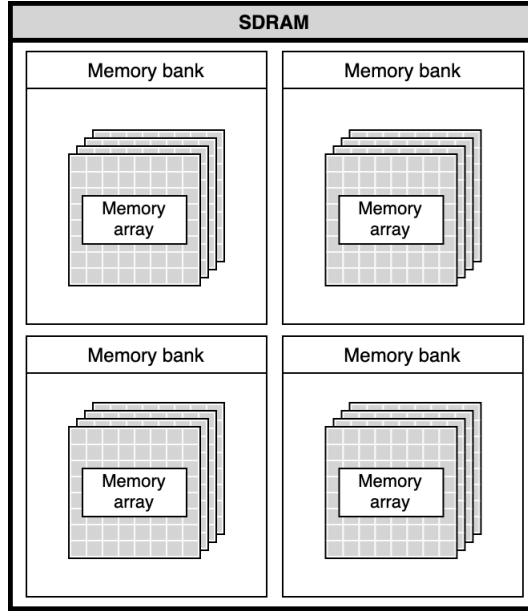
Figure 4.3: SDRAM geometry.

addresses, and 16 bits to each address, giving the a capacity of 4Gb. As previously discussed, the DDR version dictates the prefetch data width, which in DDR3 is 8 words, as is the burst length. The word length of the memory is 16 bit, which results in 128-bit burst transfers. The SDRAM is rated with a clock frequency of 1600 MHz, giving a theoretical peak bandwidth of 3.2 GB/s. In practice, the bandwidth is lower and depending on system state and access location [6].

The oscillator on the FPGA board provides a 200 MHz signal, which is also the limit of the Spartan 7 FPGA. This allows a maximum clock frequency of 325 MHz to be generated through the SDRAM controller interface and then scaled up to 650 MHz with DDR at the SDRAM. [6] [20].

The memory controller interface connects the FPGA to the SDRAM controller, which in turn sends commands to the SDRAM core. The interface is unique to each memory controller and depends on all parameters discussed in this chapter such as memory geometry, DDR version, and clock frequency [6].

4.2.2   Block RAM

When data size is within a few megabytes or less, lower level storage mechanisms become suitable to use as memory. The block random access memory (BRAM) is a type of on-chip memory, which uses memory cells available on the FPGA. The specification of an FPGA chip often include the available BRAM capacity.

Although BRAM is a type of on-chip memory, it is separate from the CLBs on the FPGA. In the case of SRAM based FPGAs, the memory cells of the BRAM are essentially SRAM cells, arranged in arrays. The BRAM is made available to the other CLBs by being attached to the FPGA interconnect.

BRAM can be configured in size and functionality depending on the needs. Common BRAM arrays are 36Kb, that can either be divided into smaller arrays, or combined with other arrays to form a larger memory resource. A useful feature often found regarding BRAM is multi port support. This enables concurrent usage of the memory resource. In case of concurrency issues, the configurable operation mode of the BRAM dictates how to resolve them. For example, in a dual port configuration, the read first operation mode will prioritize a read operation when the other port requests a write operation on the same address [21].

Similarly as SDRAM, BRAM also is synchronous and requires a clock signal. The latency is depending on the configuration and state of the memory, but usually within a few clock cycles. The low latency, together with the high configurability makes BRAM a useful resource in most FPGA designs [21].

4.2.3   Distributed RAM

The lowest level of memory found in FPGAs is distributed RAM (DRAM). It is a type of on-chip memory which utilizes the same CLBs that is common with other logic of the FPGA design. This makes the use of DRAM a trade-off between fast memory and CLB resources. If other parts of the FPGA design is large, there might not be room for DRAM, leading to slightly less performant BRAM to be used instead.

The slices in Xilinx CLBs are either of type SLICEL or SLICEM. The slices in a Xilinx 7 series FPGA CLB consists either two SLICEL or one SLICEL and one SLICEM. DRAM is available only on SLICEM type of slices. The SLICEM contains a superset of SLICEL elements, allowing SLICEM to be used in non storage purposes as well. The geometry of DRAM memory is defined by the configuration of SLICEM slices. The configuration and decision to use DRAM is inferred from the FPGA design. A single SLICEM CLB in the Xilinx 7 series FPGA allows for 256 bits of DRAM storage. The DRAM capacity is increased beyond this by connecting DRAM enabled CLBs together. The DRAM capacity and number of SLICEM enabled CLBs is often mentioned in the FPGA product specification [21].

Since DRAM resides in the FPGA fabric, it communicates through the FPGA interconnect making read and write operations fast. Read and write operations require a single clock cycle to complete, making it also very simple to use from a HDL design perspective [21].

4.3   Clock

As most digital hardware systems, FPGAs use clocks to synchronize operations. The clock is an off-chip component, providing the signal through input/output pads to the FPGA. Figure 4.1 shows the off-chip clock circuit on the left side of the FPGA on the XEM7305 module. The clock signal is generated by an oscillator circuit that produces a specific frequency. This frequency may then be used as is or transformed into additional clock signals within a HDL design or IP core. For example, the DDR SDRAM utilize the clock signal on both the rising and falling edge to increase the throughput of data. The clock signal on the XEM7305 is passed from the external clock module to the SDRAM module within the HDL design. This ensures that the SDRAM and FPGA design is synchronized [6].

The clock module on the XEM7305 provides a 200MHz frequency signal. An additional 100.8MHz clock signal is provided by the USB interface, which is required interacting with the USB peripheral controller. A third clock is used by the SDRAM, which is derived from the 200MHz clock module [6].

In addition to using the clock signal to drive tasks, a common practice within system modeling with a HDL is to drive tasks using signals that occur more spontaneously. This could for example be a signal originating from a sensor on the FPGA module that has an interface to the FPGA design without a dependence to a clock. This enhances the suitability of using FPGAs for real-time requirements since the reaction to an input can be within clock cycles instead of milliseconds. Another reason to use FPGAs for real-time applications is predictability. For example, when using a clock divider to run a task at every 5ms, the task will run precisely at 5ms intervals as long as the clock oscillator itself is accurate. Software based real-time applications cannot guarantee the same level of confidence for the timing requirements due to possible scheduling and concurrency related overhead.

5. Star tracker systems

A star tracker is an instrument that provides navigational data such as position and attitude by recognizing stars from optical images. The system consists of a camera and a computer. The system operates by capturing an image, extracting and comparing data to a star catalog. The star catalog contains columns of information about known stars [22].

Star tracker systems depends on image processing and analysis that run on varying computing platforms. The platform and purpose of a star tracker in focus is a FPGA based star tracker for spacecraft attitude determination. The general idea of star tracker based attitude determination is explored in this chapter to explain the context of a pre processing pipeline for such a system.

5.1 Argumentation

There are multiple ways to determine the position and attitude of a spacecraft, but when the accuracy needs to be better than $0.1°$ a star tracker is to be used. When the high accuracy of a star tracker is not needed, other methods such as navigation satellite systems, such as GPS, or momentum wheels can be used for determining the attitude and position. Other than accuracy advantages, the star tracker is also more independent than navigation satellite systems and mechanically simpler than momentum wheels. On the other hand, star trackers require more complexity in a computational sense [22].

A unique feature of star trackers is the ability to find the attitude and position of a spacecraft without any previous knowledge, also known as Lost-In-Space attitude determination. Momentum wheel-based navigation for example, requires a reference to previous position to determine its current position. The Lost-In-Space feature is also available using navigation satellite systems, but with limited range [22].

5.2 Image acquisition

Images are usually captured by a charged-coupled device, or CCD, sensor. In contrast to another common image sensor type, the CMOS sensor, CCD is more resistant to noise resulting in better quality images.

The camera uses a short focal length lens with a fixed field of view (FOV) to maximize the number of stars captured in each image. To maximize visibility of the stars, a wide aperture in the lens is used to allow more light to enter the image sensor.

5.3 Image processing

The image processor of a star tracker extracts data from the captured image to use for star detection. The extracted data and processing algorithms varies between implementations, but the general idea is that the extraction process enhances the image features, scans the image for targets, and stores information about the targets for the star detection.

5.4 Star detection

5.5 Attitude determination

5.6 Pre processing pipeline

6. Documentation

The documentation of the pre processing pipeline explains the features and technical details of the implemented system. Each feature contributes to filling requirements stated in the system specification. Figure 6.1 shows an overview of the final product which includes the required components.
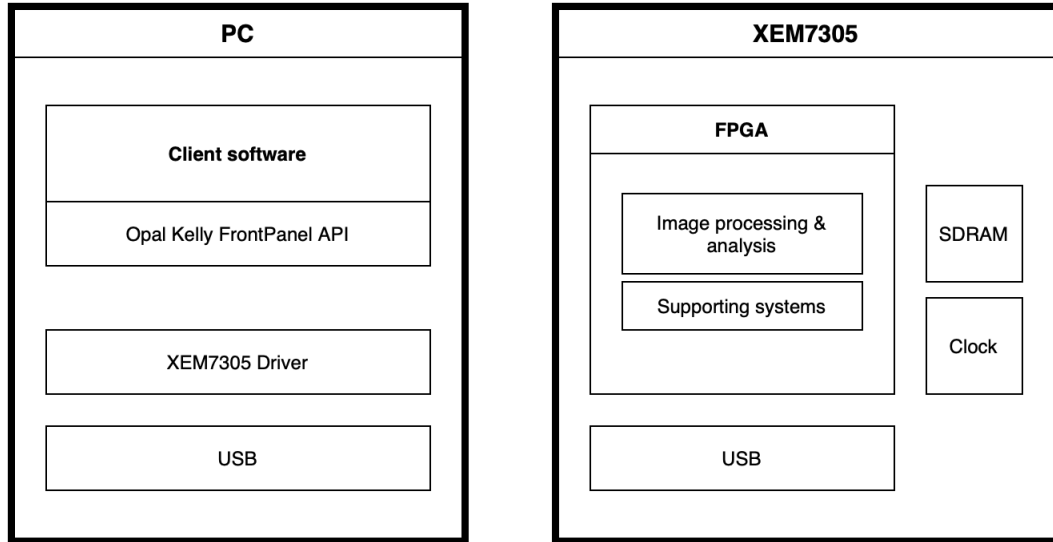
Figure 6.1: Diagram of star tracker pre processing pipeline. The FPGA, performing the image processing and analysis, on the right and the PC, being the user interface on the left.

6.1   Functional overview

The star tracker pre processing pipeline is a system that uses an input image to extract features from. The features, in this case stars, can be used for further implementation of a star tracker.

The system can be integrated to a completely embedded environment where it does not depend on a user interfacing client. An integration to a star tracker can

be independent and highly isolated, with minor adjustments made to the off-chip module interfaces. Only a few essential parameters, triggers and data is required.

From a user experience point of view, the pre processing pipeline works by powering on the FPGA, connecting it to the pre processing pipeline client. The client then programs the FPGA with the pre processing pipeline FPGA design. After successfully programming the FPGA, the pre processing pipeline is configured with a valid image and parameters. The FPGA executes the image processing and analysis after which the client reads the result, consisting of an image and metadata, for the user to evaluate.

From an integrated system point of view, the pre processing pipeline is powered on and configured by an external system, a microprocessor with a program or embedded operating system supplying parameters, triggers and data.

6.2 FPGA design

The star tracker pre processing pipeline FPGA design contributes to all system requirements in table 3.1.

Figure 6.2: Diagram showing FPGA design as functional blocks where top section is furthest from off-chip modules while bottom section is closest to off-chip modules. Boxes in the diagram are not in scale to the actual footprint of the components on the FPGA.
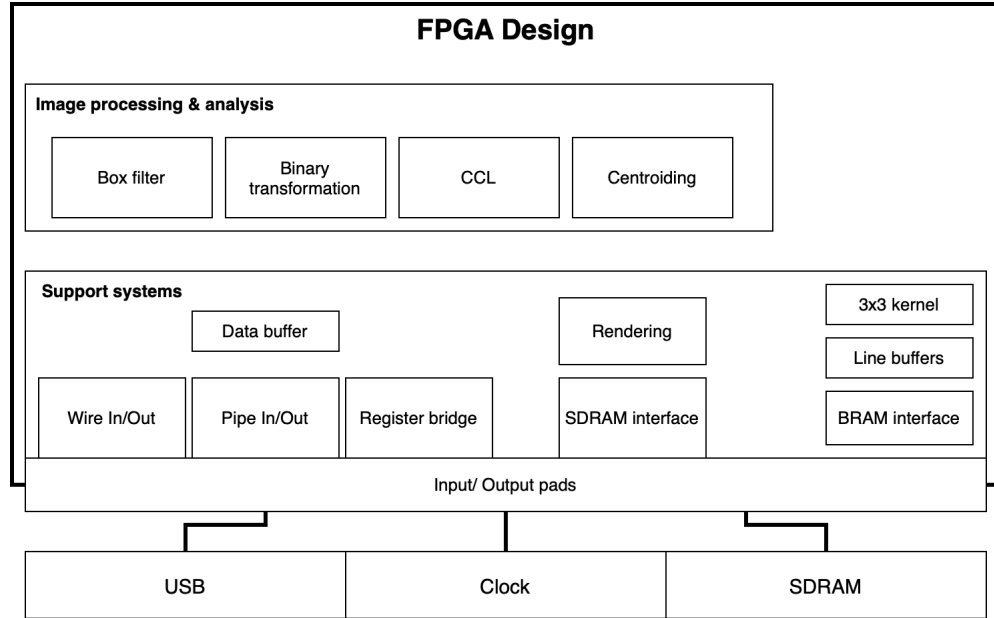
6.2.1 Support systems

The supporting systems included in the FPGA design consists of components that does not contribute to image processing and analysis directly but provides other necessary features such as storage and communication.

The core supporting systems of the FPGA design consists of modules provided by Opal Kelly. These include communication, buffering and storage solutions that are specific to the FPGA and off-chip components.

The bottom section of figure 6.2 shows main functional components of the support systems in the FPGA design. The isolation of this section can be made in a functional and in some cases also in a technical sense.

Communication and host interface

The Opal Kelly host interface, called okHost, is a gateway for controlling and monitoring the FPGA. It contains the logic for communication with the FX3 USB microcontroller, exposing signal endpoints used for controlling USB communication. The host interface also contains a clock signal derived from the external clock module on the XEM7305 board. The 100.8MHz clock signal synchronizes the logic in the FPGA design with the USB microcontroller operations.

Opal Kelly provides a set of IP core modules supporting data transfer from single bits to megabytes. Different synchronization and control methods are used for each module resulting in smaller scale data transfers being simpler than large scale transfers. Three Opal Kelly communication modules are used within the FPGA design which each fill their own purpose.

The Opal Kelly register bridge connects the host interface directly to a dedicated address space on the FPGA. The register bridge is connected to 32 addresses, with 32 bits of storage each to use synchronously. The register bridge interface contains separate 32-bit data buses for input and output, and a register address bus. This adds complexity to the FPGA design because a mapping mechanism, for example a multiplexer, is needed to make all 32 addresses usable. The register bridge is used through the Opal Kelly API in the client software. Figure 6.2 shows the register bridge connected to the input/output pads, which acts as interface between the register bridge and the off-chip modules. The required off chip modules for the register bridge is clock and USB.

Wire In and Wire Out modules are directional 32-bit buses that transfer data asynchronously. The Wire In/Out modules are by their asynchronous nature and

limited capacity good for state management and triggering events. The asynchronous data is still in many cases synchronized to a clock in the design. Figure 6.2 shows the Wire In/Out connected to the input/output pads, giving it access to the USB off-chip module.

Pipe In and Pipe Out modules are used for multi byte synchronous data transfer. The data bus on both modules are 32-bit wide. The block throttled variant of the modules adds negotiation signals to the communication system, which allows the FPGA to control the flow of data. The PC can initiate the data transfer but unless the `EP_READY` control signal is high, data is not sent. This is useful in cases where incoming data needs to be stored using buffers. Figure 6.2 shows the Pipe In/Out connected to the input/output pads, allowing it to interface with Clock and USB off-chip modules.

FPGA state management and event triggers

The Wire In module is used on the FPGA to enter states that needs to be externally managed. Table 6.1 shows a description and wire values of them. The Wire In uses only the first 4 bits of the 32-bit data and the values are set so that only one state can occur at a given moment which ensures predictability of the system. Each state in table 6.1 is mapped to an index which is used to read the value corresponding to the state on the 32-bit Wire In data bus.

The Wire Out module has the single purpose of providing verification of the setup of the FPGA. The signal is expected to be high after a small delay after the FPGA is powered on and the bitstream is programmed. Table 6.2 shows the description and wire value of the Wire Out state.

The Wire In/Out modules are a good suit for the state triggers because of the reactive and simple user experience. The state management and event trigger system contributes to satisfying the system requirements. The features provided by the Wire In module are fundamental to the requirements 1.3, 1.4, 2.2 and 3.3.

Parameter configuration and metrics

To allow data transfer for accessing register values on the FPGA, the Opal Kelly register bridge is used. The registers are addressed so that the first 15 contains writable data and second 15 contains readable data. This prevents any concurrency issues with register access. Pre assigned values to the writable addresses ensures that the pre processing pipeline parameters have default values to use in

Wire value	Description
0x0001	Read storage to Pipe Out. Used when reading image back from FPGA to client
0x0002	Write to storage from Pipe In. Used when writing image from client to FPGA
0x0004	Resets FIFO buffers. Used between image reads and writes
0x0008	Triggers image processing and analysis. Used when FPGA is configured and ready to perform star extraction

Table 6.1: Description of Wire In state control bits

Wire value	Description
0x0001	Memory calibration signal. Used to verify successful initialization of SDRAM memory.

Table 6.2: Description of Wire Out state bits

case none are assigned through the register bridge. Table 6.3 and 6.4 explains the use of the register bridge.

The features enabled by the register bridge for parameter configuration and metrics contribute to system requirements 1.2, 1.3, 1.4, 2.2 and 3.3.

Address	Data	Description
0x0000	32-bit int	Minimum brightness of an image feature. Used to filter out faint objects in the image.
0x0001	8-bit int	Binary transformation threshold. Used for filtering faint pixels out when preparing image for feature detection.

Table 6.3: Description of writable values used by the FPGA.

Image transfer and SDRAM storage

To transfer and store the image data used as input to the star tracker pre processing pipeline, a few mechanisms needs to be implemented due to the nature of the image data. The image is fixed size and format, namely 270 x 480 pixels,

Address	Data	Description
0x000F	32-bit int	Image processing and analysis timer. Used for evaluating performance. Stored in register as clock cycle count.
0x0010	32-bit int	Box filter timer. Used for evaluating box filter algorithm performance.
0x0011	32-bit int	CCL timer. Used for evaluating CCL algorithm performance.
0x0012	32-bit int	Feature rendering timer. Used for evaluating performance of feature rendering.

Table 6.4: Description of readable registers on the FPGA.

single channel 8-bit per pixel. The image is not compressed to decrease image processing complexity, making the total image size 129,6kb.

The Pipe In module is used to receive the image data on the FPGA. The module is connected to, and partly managed by a FIFO buffer. The buffer is also connected to the SDRAM interface, which pulls data from the buffer to storage. The entire data flow is thus controlled by the SDRAM, which is the bottleneck of the image transfer system.

The 32-bit wide data bus of the Pipe In module is connected to the input of the FIFO buffer. The buffer capacity is 128-bit to suit the input data bus width of the SDRAM interface. Both Pipe In and FIFO modules are synchronized to the host interface clock. The SDRAM interface is synchronized to its own 87MHz clock, also derived from the 200MHz system clock. The modules are synchronized together with control signals to manage data flow and mitigate issue with clock speed differences.

Beyond the input image, the SDRAM is used to store intermediate image processing results. The sequential process writes an image to storage after each step. Since each output is only needed as the input to the next step, the stored images can be overwritten. The process requires a storage capacity of 3 times the image size, one of which being the original. The original image is kept incase the process is re executed. In an integrated environment, the storage capacity could be scaled down to two times the image size, since the original image would be useless.

After image processing and analysis the output image is stored to SDRAM.

The output image is transferred from SDRAM to an external endpoint using the Pipe Out module trough another FIFO buffer. The image output transfer uses a similar FIFO buffer as the input image transfer, except with reversed port widths and other control signals.

The image transfer and storage systems contribute to system requirements 1.4 and 2.2.

Line buffer and BRAM storage

The image data is accessed by the image processing and analysis components through line buffers implemented as BRAM. Each line consists of 270 8-bit pixels and three lines are stored in the buffer. This geometry enables image processing on both the horizontal and vertical axis of the image. Figure 6.3 shows the line buffer in the middle, as the intermediate storage mechanism.

The dual port BRAM has two data buses and two address ports to allow simultaneous read and write operations. The interface also includes a 270-bit data mask to select which pixels to write to BRAM. This adds complexity to the line buffer system but reduces the number of registers needed.

The line buffer system is the intermediate storage between the image storage and image processing. The read and write operations are triggered by counters that are updated during the image processing.
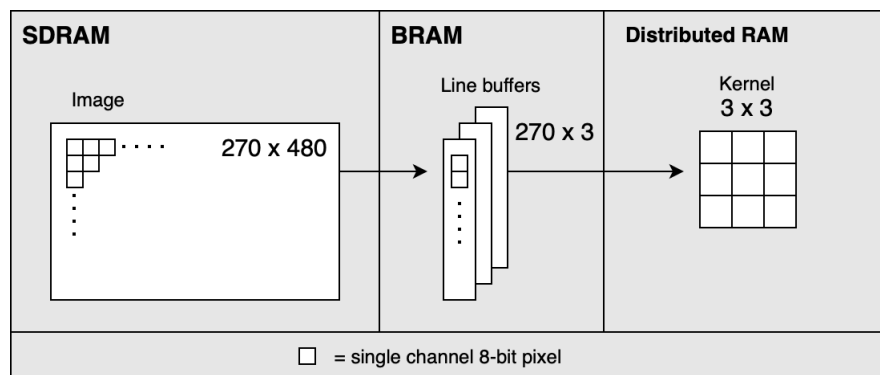

Figure 6.3: Diagram of different levels of memory used for storing image pixels of the star tracker pre processing pipeline.

3x3 Kernel

The lowest level storage mechanism where image pixels are kept is 3 small line buffers, forming a 3 x 3 pixel sliding window, or image processing kernel. It is

implemented in distributed memory, within the FPGA fabric. The kernel consists of three pixels from each column in the line buffer, creating the 3 x 3 pixel geometry shown in figure 6.3. Data from the line buffer is fed into the kernel data structure which is then used for image processing and analysis. The results of these operations are written directly to a higher level storage. Thus the the data flow is directional within this storage mechanism.

Rendering

The image is rendered in the original 270 x 480 single channel 8-bit format to SDRAM in between image processing steps and for final output.

6.2.2   Image processing and analysis

To extract stars from an image, a number of image processing algorithms are implemented. The general idea is that an image containing bright stars in the foreground and faint objects or empty space in the background is used as input. The first algorithm smoothens the image to create larger and more detectable foreground features. The second process separates the foreground from the background with a threshold. This is to enable connected component labeling (CCL) in the next step, which requires a fully separate foreground and background. Connected component labeling algorithm traverse trough the image pixels and assigns them to features. Metrics of the features are stored for calculating the centroid in the final step.

Box Filter

A box filter is applied on the image to prepare the image for CCL. The principle of the box filter is to blur the image by replacing a pixel with the average of the surrounding pixels. The 3 x 3 kernel is used for this filter, where the center pixel is the target. This filter is commonly described with the expression

$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$, where the matrix represents the kernel with pixel values. This

gives a uniformly applied average of the image where high value pixel areas are expanded without too much loss of detail.

A problem with the box filter is the need for integer division. A division operation is problematic when implemented in hardware and is often dedicated a

separate IP core module. This increased complexity can however be avoided in the case of the box filter because it lacks the need for high accuracy output of the operation, as the pixel values are 8-bit integers. The alternative operation to replace the division takes advantage of the constants in the box filter, which is the output bit width and the denominator. The lack of accuracy is compensated by the performance which is reduced from a multi clock cycle operation into a single clock cycle operation. The division is approximated with the expression $(\sum x_{ij} \cdot 28) \gg 8$, where x is the 3 x 3 kernel.

Image edges are handled by ignoring the pixel value and setting them to 0. This assigns the edges of the image as part of the background feature. The possible loss of edge features is also acceptable because of the difficulty to approximate the complete geometry and center of an edge feature.

Thresholding

Thresholding is done to separate the background from the foreground in the image. The foreground needs to contain enough pixels to enable successful star extraction. The box filter allows foreground feature edges to be brighter, which otherwise might be categorized as background in this step. If the threshold is too low, background and noise blends together with foreground features, making the choice of optimal threshold a complex problem. This is out of scope of this project and the threshold is manually set.

An example solution for setting the threshold could be to use a feedback loop to adjust the parameter. The loop would use previous output of the pipeline to use as input, providing the number of stars, brightness and distance to determine a threshold setting and acceptable result.

The thresholding process transforms each pixel value to either 0, being background, or 255 as foreground. As it is a process with no dependency to neighboring pixels, it can be conveniently placed in the output phase of the box filter. The output phase is where the result of the box filter operation is written to SDRAM memory. The register containing the output of the box filter is piped to the SDRAM write bus through the threshold. This makes the thresholding highly efficient because of no additional clock cycles required.

Connected component labeling and analysis

The goal with CCL is to assign foreground pixels to features. The algorithm traverses the image using a kernel, updating the feature tables and pixels in the line buffer. Three feature tables keeps information about pixel values and the feature geometry. Every encountered foreground pixel is updated depending on its neighbors. If any neighboring pixel has a value with an entry in the feature table, the current pixel is updated and mapped to the same feature. The geometry of each feature is stored as minimum and maximum coordinates within the image. Also the brightness of each feature is stored. This is used together with the geometry for feature analysis.

The requirement for a component to be connected to a feature is that at least one side of the pixel touches the feature, making it a 4-connected feature. In contrast, 8-connected feature accepts diagonally connected components. Diagonally connected components are not detected in this CCL implementation because the previously performed box filtering already exaggerates the component connections, filling the gaps between pixels.

The 4-connect comparison is done by comparing the top center and left center to the center pixel in the kernel. The movement of the kernel mitigates the need for comparing all 4 connections surrounding the center pixel in the kernel. This simplifies the design while keeping line buffer mechanism uniform with other parts of image processing.

Conflicts in the feature table is encountered when connected components refer to multiple features. This happens when the shape of the feature is unfavorable in regards to kernel movement and component comparison process. A conflict is resolved by merging the connected features. This is done by merging brightness of the features involved, updating the feature coordinates, and reassigning the feature label. Depending on the number of mergers, a single feature might be mapped to a number of pixel values.

While traversing the image, CCL will enter one of three states for each foreground pixel, while background pixels are ignored. When encountering a foreground pixel in the left center it can be assumed to belong to a feature. The state where left center is foreground is prioritized because the feature it refers to has the earliest entry in the feature table. The feature with the earliest entry needs to be selected for the center pixel to avoid conflicts. The next priority is to compare the top center pixel. This state updates the center pixel with the value of the top

pixel. A new feature is created when neither the left center or top center pixel is in the foreground. This state adds an entry to the feature tables.

The implemented CCL algorithm streamlined and simple, but not without drawbacks. The most significant limitation is the number of features that can be stored. In case all foreground features in the image are square, the maximum number of features is 253. This is more than enough for what a star tracker needs, but depending on the number of conflicts the number of available features is significantly lower. This limitation originates from the 8-bit pixel values stored in the feature tables and the lack of re-renders of the CCL image. With re-rendering implemented, the maximum would be guaranteed to 253 features, instead of 253 as best case. The trade-off for choosing the re-render to allow more features is a performance loss and a larger memory and FPGA design footprint.

Pixel Value	Feature Label
0x01	0x01
0x03	0x02
0x04	0x03
0x05	0x03
0x00	background
0xFF	foreground

Feature Label	Coordinates
0x01	$(x_{min}, y_{min}, x_{max}, y_{max})$
0x02	$(x_{min}, y_{min}, x_{max}, y_{max})$
0x03	$(x_{min}, y_{min}, x_{max}, y_{max})$

Feature Label	Brightness
0x01	[0x00 - 0xFFFFFFFF]
0x02	[0x00 - 0xFFFFFFFF]
0x03	[0x00 - 0xFFFFFFFF]

Figure 6.4: Example of CCL feature tables. Left table is where the mapping of pixel values to feature labels are stored. Labels are also renderable, as they are stored as 8-bit integers. Center table stores minimum and maximum coordinates of each feature in 35-bit integers. Right table stores feature brightness as 32-bit integers.

Feature detection and centroiding

6.3 Client endpoint

6.3.1 Communication

6.3.2 Pipeline configuration

7. Review and reflection

8. Discussion

Bibliography

[1] Neil C. Thompson and Svenja Spanuth. The Decline of Computers as a General Purpose Technology. Visited 20.1.2021, 2018.

[2] Xilinx Inc. *SDSoC Environment Profiling and Optimization Guide*, 2018.

[3] Xilinx Inc. *7 Series FPGAs Configurable Logic Block*, 2016.

[4] Mahendra Pratap Singh and Manoj Kumar Jain. Evolution of Processor Architecture in Mobile Phones. *International Journal of Computer Applications*, 90:35, 2014.

[5] Jenni Virtanen, Jonne Poikonen, Tero Säntti, Tuomo Komulainen, Johanna Torppa, Mikael Granvik, Karri Muinonen, Hanna Pentikäinen, Julia Martikainen, Jyri Näränen, Jussi Lehti, and Tim Flohrer. Streak detection and analysis pipeline for space-debris optical images. *Advances in Space Research*, 57:1607–1623, 2016.

[6] Opal Kelly Inc. *XEM7305*, 2021.

[7] Robin Mitchell. DRAM, SRAM, FLASH, and a New Form of NVRAM: What's the Difference? Visited 5.10.2021, 2020.

[8] Xilinx Employee. Touchdown! NASA's Perseverance Rover Lands on Mars with Xilinx FPGAs On Board. Visited 19.4.2021, 2021.

[9] Wim Roelandts. 15 Years of Innovation. *XCell*, 32:2, 1999.

[10] Investor Overview. https://investor.xilinx.com/static-files/ 2958145c-e3a2-456f-a461-bc9cba375af3. Visited 19.4.2021.

[11] Clive Maxfield. FPGAs from A to Z part 1. Visited 19.10.2021, 2006.

[12] Xilinx Inc. *Vivado Design Suite User Guide*, 2021.

[13] DJ Holding. *Electrical Engineer's Reference Book (Sixteenth Edition)*. Aston University, 2003.

[14] Eli Greenbaum. Open source seminconductor core licensing. *Harvard Journal of Law & Technology*, 25:132, 2011.

[15] Do-254 explained. Technical report, Cadence, 2019.

[16] Aboa Space Research Oy. Starmatch, 2020.

[17] Cypress Semiconductor. *EZ-USB FX3 Technical Reference Manual*, 2019.

[18] Avinash Aravindan. Flash 101: NAND Flash vs NOR Flash. Visited 5.10.2021, 2018.

[19] Micron Technology, Inc. *General DDR SDRAM Functionality*, 2001.

[20] Micron Technology, Inc. *DDR3L SDRAM*, 2011.

[21] Xilinx Inc. *7 Series FPGAs Memory Resources*, 2019.

[22] George Sebestyen, Steve Fujikawa, Nicholas Galassi, Alex Chuchra. *Low Earth Orbit Satellite Design*. Springer, 2018.