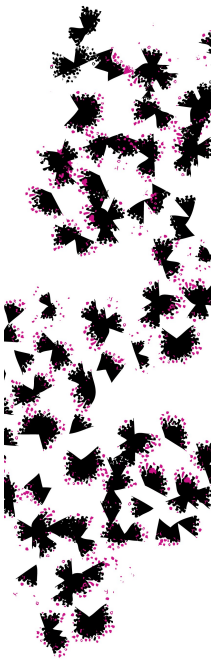


DM3730 Camera Interfaces on Gumstix

M. (Mattanja) Venema

MSc Report

Committee:
Dr.ir. J.F. Broenink
Ir. E. Molenkamp
Ing. M.H. Schwirtz

August 2016

033RAM2016
Robotics and Mechatronics
EE-Math-CS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Summary

The goal of this design project is to explore the paths through which a camera can be connected to the Gumstix board used at the ESL course with the purpose of obtaining a formatted image that can be used for further processing. This is done to expand the design-space-exploration phase of the ESL course and to enable other setups in the RaM lab to make use of different vision systems.

A design space exploration is done to investigate the possible paths. Several of the paths are then designed and implemented using the hardware restrictions of the ESL course. The implemented designs are tested with profiling tools to obtain a performance indication of the impact each path has on the processor load and the memory usage of the Gumstix.

The results show that cameras can be interfaced through either the USB connection, the dedicated Image Signal Processor of the Gumstix, or through the FPGA located on the RaMstix. Of these paths the USB connected camera uses the least resources of the Gumstix, but the camera connected to the ISP is a space effective alternative. The FPGA connected camera can only operate at lower resolutions, but is fast and flexible. It also allows for a stereo vision setup.

Contents

1 Introduction

In this report the design project to exploit the different possibilities of handling camera input by the Gumstix Overo in combination with the RaMstix board is described. This will be done with the purpose of image processing and computer vision in mind.

1.1 Context

The current state of technology makes it possible to implement a broad scala of complex robotic systems. With these advanced systems and hardware the input options are not limited to simple sensors. Complex sensors like vision systems using cameras that generate a lot of data are becoming more common.

When these cameras are combined with embedded systems the resources to operate such a sensor system have to be taken into account, because they are limited. Another point of attention is the interface between the camera and the embedded system. Many practical applications make use of a USB connected camera, but when space constraints are present in a design a dedicated, smaller, camera interface is available on certain platforms.

A practical example is the course of 'Embedded Systems Laboratory'. The final assignment is the implementation of a vision-in-the-loop controller for the JIWY setup. The JIWY setup is a mechatronic device with two rotational degrees of freedom and a camera as its end effector (Lammertink, 2003).

Part of the ESL course is a design-space-exploration phase in which subsystems of the JIWY setup, that can be implemented in various ways, are evaluated. The vision subsystem is at the moment restricted to the use of a USB webcam.

The aim of this project is to expand the vision subsystem with more possible solutions and to determine the performance of those solutions given the hardware used at the ESL course.

This will expand the design-space-exploration phase of the course and might also enable other setups in the lab to benefit from different vision system solutions.

1.2 Goals

The goal of this design project is to explore the paths through which a camera can be connected to the hardware used at the ESL course with the purpose of obtaining a formatted image that can be used for further processing.

To accomplish this, different types of cameras, interfaces and places to do the formatting are researched.

The hardware used for this project is restricted to the hardware used in the ESL course except for the cameras. The hardware consists of the JIWY setup attached to a RaMstix board (Schwirtz, 2014) or attached to the combination of a Gumstix (Gumstix Inc., 2015) and the Altera DE0 Nano board (terasIC, 2012). This limits the number of options to interface a camera to either use USB, the Image Signal Processor (ISP) or the FPGA located on the RaMstix or the DE0 Nano.

All of the solutions have a different impact on the Gumstix regarding processor load and memory usage. This impact has to be measured.

1.3 Approach

To achieve these goals a design space exploration is done to establish the possible paths of interfacing a camera to the Gumstix and obtaining a formatted image. Several of these paths

are designed and implemented. For example, using a camera with the dedicated ISP interface and a solution that interfaces a camera through the FPGA.

The performance of the Gumstix, while retrieving and formatting images, is measured using profiling tools to determine the impact a certain solution has on the processor load and memory usage.

The steps taken can be summarized into the following:

1. Investigate the possible ways to interface a camera with the Gumstix while adhering to the restriction of using only the hardware available in the ESL course, except for cameras.
2. Design and implement several of the solutions in hardware and software.
3. Obtain a performance indication of each implemented solution by profiling the impact on processor load and memory usage.

1.4 Outline

The report will be continued with a chapter containing the background information, Chapter 2. An analysis of the problem including the design space exploration is documented in Chapter 3. The design of the paths to interface a camera to the Gumstix that will be realized are discussed in Chapter 4. Measuring the performance impact and the accompanying results are given in Chapter 5. Finally the conclusions and further recommendations are given in Chapter 6.

2 Background

In this chapter the hardware and software that are used in this project are described. This is the same hardware used in the ESL course except for the cameras.

2.1 Gumstix

The Gumstix Overo is a series of computer-on-module featuring Texas Instruments processors specialized in audio and video applications.

The one used for this project is an Overo Firestorm-P model featuring a DM3730 processor (Gumstix Inc., 2015; Texas Instruments, 2010). This processor has an ARM Cortex-A8 and several subsystems dedicated to video or audio processing. In figure 2.1 a functional block-diagram overview of the subsystems and peripherals supported by the DM3730 processor is shown. The ARM-core is denoted by the blue box.

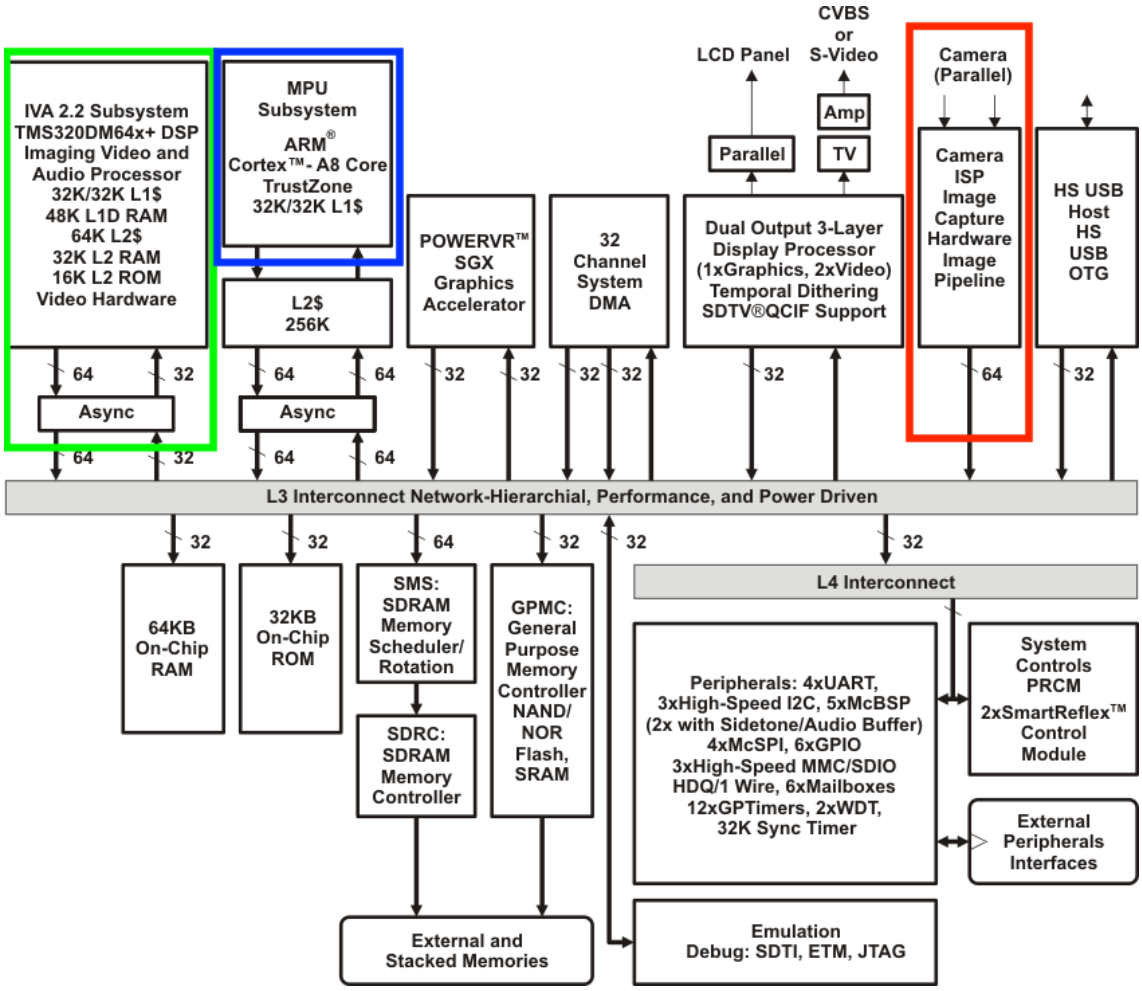


Figure 2.1: Functional block diagram for DM3730 (Texas Instruments, 2011) annotated to show the components essential for this project.

2.1.1 ISP

The subsystem that is especially important for this project is the camera Image Signal Processor (ISP) which is denoted in figure 2.1 by the red box. This system provides an interface and pro-

cessing capabilities for RAW imaging sensors. It can be used to directly interface a camera with the processor through a parallel 12-bit input channel or a serial input channel. An overview diagram of the ISP hardware is shown in figure 2.2.

The ISP has a 27-pin connector on the outside that is used to interface with the camera hardware. There are two physical layers in the ISP connected to this connector these are marked by the red box in figure 2.2.

Each physical layer can interact with one camera. Therefore, it is possible to connect two cameras to the Gumstix ISP simultaneously. This is restricted to two serial cameras or one serial and one parallel camera only, because a parallel camera needs 17 out of the 27 connections to interface with the Gumstix.

Internally the ISP has three receivers that are MIPI compliant (MIPI alliance, 2014). Two of the receivers are compatible with the MIPI D-PHY CSI2 standard and a third CCP2B (compact camera port) is MIPI CSI1 compliant.

The MIPI standard defines hardware and software protocols that dictate how, for example, a camera can be interfaced with a processor.

The ISP also features various internal processing functions. The most important are the previewer and the resizer. The previewer is capable of formatting an incoming RAW image to a specified image format and the resizer can, as the name implies, resize the images to a specified resolution.

The use of these functions can be defined in software on the Linux platform.

2.1.2 RaMstix

At the Robotics and Mechatronics chair the Gumstix is used in combination with an FPGA for expanded I/O capabilities.

To this end a custom baseboard was created in-house consisting of a PCB with an Altera FPGA, buffered I/O channels, dedicated ADC/DAC, CAN, etc. The PCB is also host to a Gumstix with networking options and USB on dedicated connectors. This baseboard is called the RaMstix (Schwirtz, 2014). In figure 2.3 the RaMstix board is shown with colored boxes indicating the Gumstix (red), the ISP connection (yellow), the USB connection (green) and the FPGA connections (purple).

GPMC

Communication between the Gumstix and the FPGA on the RaMstix baseboard is made possible through a General Purpose Memory Controller bus (GPMC). This bus has a 16-bit wide data channel and a 10-bit address channel. Along with a clock and several status lines it can transfer data between the Gumstix and the FPGA.

2.1.3 Accelerators

Another feature of the Gumstix is its on-chip Digital Signal Processor (DSP). This core is specialized in parallel processing of video and audio. It can be used as a hardware accelerator to encode or decode images retrieved from a camera source. It is connected to the ARM-core through an interconnect and can be accessed from the ARM-core by use of an Inter Processor Communication protocol (IPC). The DSP is denoted in figure 2.1 by the green box.

The ARM-core on the Gumstix also embeds the NEON architecture. This is a general-purpose Single Instruction Multiple Data (SIMD) engine that can efficiently process multimedia formats (ARM Ltd., 2011). It can also be used as an accelerator.

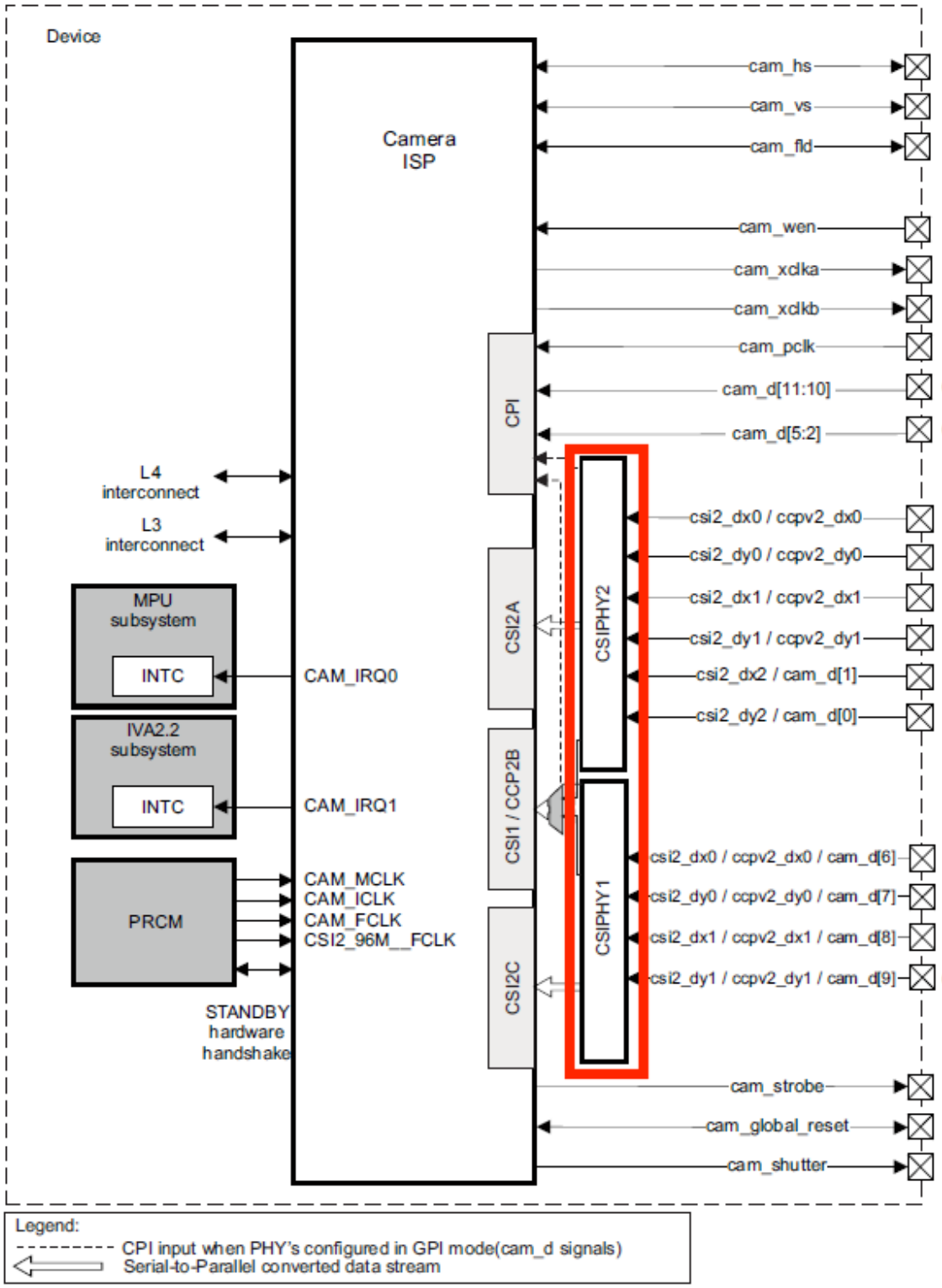


Figure 2.2: Image Signal Processor overview (Texas Instruments, 2012) annotated to show the physical layers.

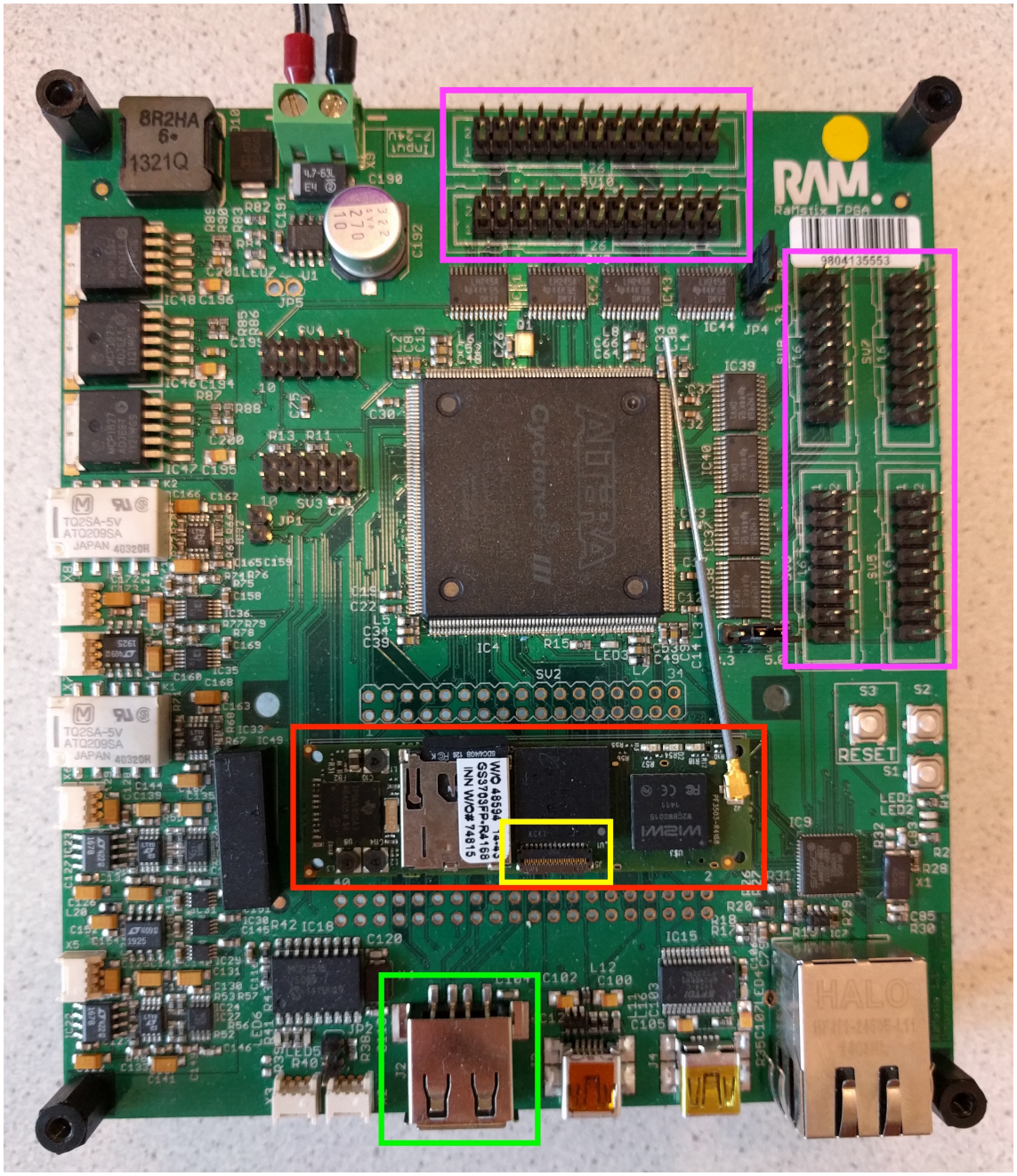


Figure 2.3: RaMstix board with colored boxes indicating the Gumstix (red) and the possible camera connections

2.2 Cameras

In this design project three different cameras are used: A Logitech C250 USB webcam (Logitech, 2009), the Caspa camera (Gumstix Inc., 2016) and the Omnivision OV7670 (OmniVision, 2006). An overview of these cameras and their specifications is shown in table 2.1.

2.2.1 Logitech C250

The Logitech C250 USB webcam is chosen because it is the default camera used in the ESL course. It is embedded in the JIWY system. The images are formatted on the camera to either

Name	Connection	Resolution	max frame rate	Output format
Logitech C250	USB 2.0	640x480	30 fps	MJPG or YUYV
Caspa camera	27-pin ISP	752x480	60 fps	RAW 10-bit Bayer
Omnivision OV7670	18-pin ISP compatible	640x480	30 fps	RGB or YUYV

Table 2.1: Specification of the chosen cameras

the MJPG or YUYV format before being sent to the Gumstix. There is also no possibility to obtain unformatted images from this camera.

2.2.2 Caspa camera

The Caspa camera is chosen because it is sold by Gumstix Inc. and is directly compatible with the Gumstix being used in the ESL course. The actual camera sensor on the board is an Aptina (ON Semiconductor) MT9V032 CMOS sensor (ON Semiconductor, 2015).

Images generated by the image sensor are unformatted. This means RAW 10-bit Bayer values are sent to the Gumstix. Bayer values represent light intensity and are sorted into colors. The Gumstix has to evaluate these values to generate a formatted image that can be used for further processing.

The connection to the ISP contains an I^2C bus to control the camera setup and a 10-bit wide data bus to transfer the image data.

2.2.3 OV7670 camera

The Omnivision OV7670 camera is chosen because it has an internal DSP. Just like the USB webcam it can produce formatted images in either RGB or YUYV format. This means that no formatting has to be done on the Gumstix itself.

The camera is readily available, cheap and a popular choice among electronics-enthusiasts. This also means the camera is well documented and examples of how to use it are available.

The camera module has an 18-pin connector that is used for interfacing with other hardware. These connections include an I^2C bus to control the camera setup and the image data can be retrieved over an 8-bit wide data bus. It is compatible with the ISP, but an adapter board is needed to convert the 18-pin header to a 27-pin flex cable.

2.3 Software

The software used for this project consists of a Linux distribution running on the Gumstix with various programs needed to sample the camera data and perform operations on the retrieved data. Also several profiling tools are needed to ascertain the performance of different camera setups.

2.3.1 Yocto

The Yocto project is an open source tool used to build custom Linux distributions for embedded systems (Yocto Project, 2010).

Gumstix Inc. has made available a set of meta-layers for their Gumstix Computer-On-Modules. These layers provide hardware specific drivers and image templates that can be used in the Yocto Project to build a Linux distribution tailored to the Gumstix Overo.

The Yocto Project uses a reference system called Poky which is a collection of Yocto Project tools and metadata that serves as a set of working examples. It can be used as a basis for a custom Linux distribution and is also used to create a distribution for the Gumstix Overo.

Poky is currently on build number 1.8 also called Fido. This distribution utilizes the Linux 3.18 kernel. This build has been extended with the Gumstix meta-layers that include drivers for the camera ISP subsystem.

A detailed explanation of how to create a bootable Yocto distribution can be found in Appendix A.

2.3.2 Software tools

Besides the Linux distribution a number of other software tools are used. Most important are OpenCV and the profiling tools.

OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library (Bradski, 2000). It is written in C++ and optimized towards real-time operation. The library has a lot of built-in functionality for retrieving image sensor data and especially processing it.

GStreamer

GStreamer is a library used to setup a media pipeline (GStreamer, 2012). It can connect video/audio devices with several filters/codecs and outputs in a flexible way using a command-line interface or through C-programs. It is used in this project to create a pipeline from the camera source to for example a filesink to store images or video data.

Media-ctl

Media-ctl is a command line tool that is used to setup the ISP functionality. This tool allows to enumerate media entities and their pads and generate links between these pads. It is a part of the V4l-utils package from LinuxTV (LinuxTV, 2015). The media entities are the hardware blocks representing the imaging sensor, the receivers, a preview element, resizer and more. The pads are the in- and outputs of the hardware blocks.

These media entities can be linked to form a hardware pipeline for the imaging sensor.

A hardware pipeline can be defined to, for example, grab an image from the camera, format it using the previewer, resize it to a specified resolution and then put it in memory.

Time

Time is a default Linux program that is used to record the time a process takes to execute. The elapsed real time, CPU-seconds spent in user mode and CPU-seconds spent in kernel mode are all recorded separately and can be used to calculate the percentage of the CPU that the process got. A 'CPU second' is a second's worth of CPU cycles running a process or thread. The formula for calculating the CPU load percentage is:

$$\text{CPU load} = \frac{\text{CPU-seconds User mode} + \text{CPU-seconds Kernel mode}}{\text{Elapsed real time}}$$

The peak memory usage is also recorded along with other data that is not relevant for its use in this project.

Perf

Perf is a performance analyzing tool in Linux. It is capable of statistical profiling of the entire system based on performance counters (Perf Wiki Kernel.org community, 2015).

NMON

NMON is a performance monitor for Linux (Griffiths, 2003). It can output important performance information on screen, or save it to a comma separated file for further analysis. To perform this analysis several extra tools are available for creating graphs from the acquired data.

2.3.3 FPGA programming

The development for the FPGA on the RaMstix board is done using Quartus (Altera Corporation, 2013). This is a design tool and with it VHDL code can be written and compiled to be run on the FPGA. It is chosen because an Altera FPGA is used and the course of ESL is also using this tool.

3 Analysis and feasibility

In this chapter the analysis is made of the problem-statement and possible solutions using a design space exploration.

3.1 Introduction

The problem being faced is the lack of vision systems with regard to the setup used at the Embedded System Laboratory course.

This setup currently consists of either a RaMstix with Gumstix or the combination of a Gumstix with the Altera DE0 Nano board. These platforms are connected to the JIWY setup. The USB webcam is connected to the Gumstix and the encoders and motor drivers are connected to the FPGA board.

As is described in Chapter 1 the focus of this project is on the vision subsystem. The acquisition of the images is researched and implemented until the stage where a formatted image is obtained.

A formatted image is defined as an image formatted in an appropriate color-space. The RGB or YUYV color-spaces are used, because they lend themselves well for further processing.

Currently retrieving images through the USB webcam is the only option for the ESL course.

There are multiple possibilities to acquire an image and the formatting of the image can be done in multiple places. Each of these possible solutions of acquiring and formatting an image will be referred to as a 'path' throughout this chapter.

The design space exploration detailed below will give an overview of the possible paths followed by a section listing the prioritized requirements that are used for the design and implementation.

3.2 Design space exploration

Using the available hardware there are multiple possibilities to interface a camera to the Gumstix. An overview of the possibilities, indicated by colored paths, is shown in figure 3.1.

In the figure two cameras are shown and three possible interfaces to the ARM-core. The formatting of the images can be done in either the camera, the interface, or the ARM-core itself. There is also a possibility to speed up the formatting by using the DSP built-in to the Gumstix. Combining these three things seven paths can be formed through which an image can be retrieved and formatted.

These paths are further specified in table 3.1.

Path number	Interface	Formatting done in	Remark
1	USB	camera	No RAW image available
2	ISP	camera	
3	ISP	ARM	Not obvious, because ISP can do it
4	ISP	ISP	
5	FPGA	camera	
6	FPGA	FPGA	
7	FPGA	ARM	

Table 3.1: List of the available paths

Both the cameras, interfaces and formatting options are discussed in separate sections below.

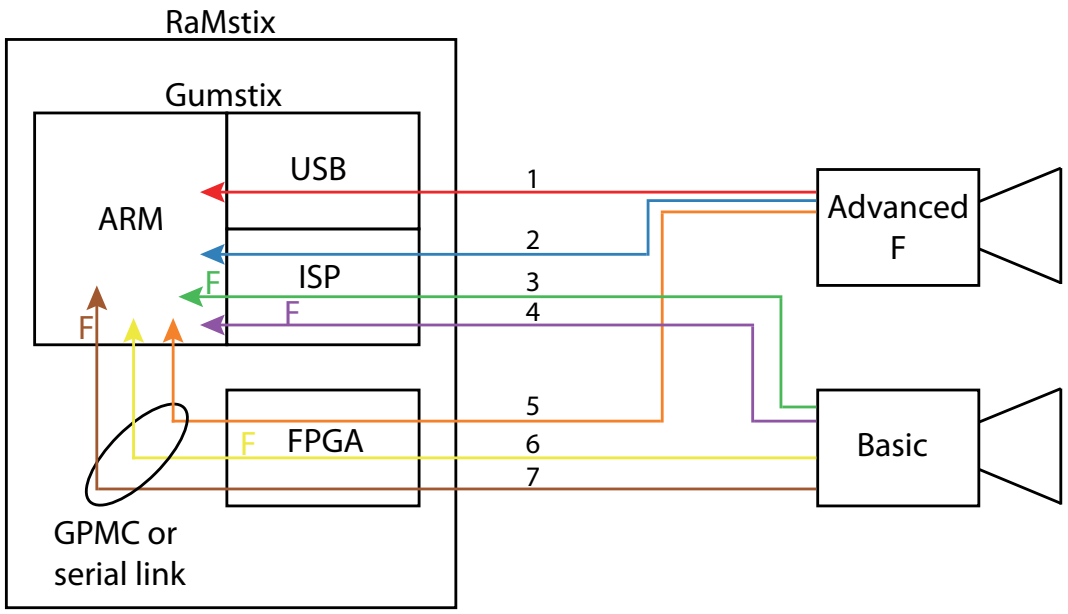


Figure 3.1: Overview of the different paths to interface a camera. 'F' denotes where the formatting is done in case of the 'basic' camera.

3.2.1 Interface possibilities

There are three interface options. Cameras can either be interfaced through the USB, ISP or FPGA.

The USB connection on the Gumstix supports the USB 2.0 protocol and as was mentioned before this is currently used in the ESL course.

The dedicated camera ISP on the Gumstix is a logical next-step to use for image acquisition. It consists of a physical interface and an internal structure that can process the RAW data as is explained in Section 2.1.1.

The third interface possibility is using the FPGA to acquire image data. There are two FPGA's that can be used in this project. The FPGA on the RaMstix and the Altera DE0 Nano board. The difference lies in the connection between the FPGA and the Gumstix. The DE0 Nano board uses a serial connection and the RaMstix a GPMC bus. A comparison between the two is made below.

The RaMstix FPGA has plenty of pins to interface with one of the parallel cameras and with VHDL the functionality of the FPGA can be flexibly defined. It is connected through the GPMC bus that can transfer data 16 bits at a time. Performing a read and write action is clocked at a speed of 280kHz. This amounts to roughly 1 MB/s.

The Altera DE0 Nano FPGA board also has enough pins to interface with a parallel camera. However, the connection between the FPGA and the Gumstix is currently a serial line. The serial line has to transfer data one bit at a time and at a baudrate of 115200 this amounts to roughly 11 KB/s.

To illustrate the data rate generated by the cameras the following calculation is done.

Consider a frame of 160x120 pixels. This is of QQVGA quality and relatively small. Using the RGB565 format for this image results in a data size of 2 bytes per pixel. This equals $160 * 120 * 2 = 38.400$ bytes per frame.

If a frame rate of 15 frames per second is desired this results in $38.400 * 15 = 576.000$ bytes per second or approximately 560 KB/s. The GPMC connection is capable of this data bandwidth,

but the serial connection cannot handle this. The serial connection as explained above can only handle 2% of this data bandwidth.

For this reason the project only deals with the FPGA on the RaMstix board and not the DE0 Nano board regarding the camera interface.

3.2.2 Camera types

Two types of cameras are used in this project. These are cameras that output unformatted images and cameras that can generate formatted images. The first type will be called the 'basic' camera throughout this report and the second will be referred to as the 'advanced' camera.

- Basic camera: generates unformatted images.
- Advanced camera: formatting is done on the camera and generates formatted images.

When using the 'basic' camera formatting has to be done either in the interface or on the ARM-core. This is explained in Section 3.2.3.

There are many different cameras that can be used with the Gumstix ISP connection. However, there are two requirements that have to be met:

- The camera needs to comply with the MIPI standards so it can be connected to the ISP.
- There has to be an appropriate Linux driver for the camera so it shows up as a media device and can be used as input device.

With regard to the ISP requirements set and the need for a 'basic' as well as an 'advanced' camera the following three cameras are chosen for this project: The Logitech C250, the Caspa camera and the Omnivision OV7670.

Gumstix Inc. has made available a camera board called the Caspa as is explained in Section 2.2. It connects to the ISP interface and fulfills both requirements mentioned above. Also because it is developed by Gumstix Inc. itself, it is an excellent candidate.

The Caspa camera is a 'basic' camera and can only generate unformatted frames. Therefore, the formatting still has to be done. This can either be done in the ISP or on the ARM-core as was mentioned.

The OV7670, as explained in Section 2.2, has an internal DSP that is used for formatting images. This 'advanced' camera can be used to reduce the resource usage of the Gumstix.

The Logitech C250 is a USB camera that interfaces with the Gumstix using the USB 2.0 protocol. This camera also generates formatted images and can be classified as an 'advanced' camera.

3.2.3 Formatting

The images generated by the camera still need to be formatted to be usable. This can be done in five places:

- On the camera itself.
- In the ISP.
- In the FPGA.
- Using the DSP.
- On the ARM-core.

When the formatting is done in the ISP this uses resources of the Gumstix processor, because the ISP is a part of it. The ARM-core has to manage the data transfers between the internal ISP elements and the transfer between the ISP and the ARM-core itself. This means the ARM-core is resources are also used. These resources are the processor load and the memory usage. To decrease the load on the processor the built-in DSP is available for hardware acceleration. However, from initial research it was decided not to use the DSP. The drivers that are needed to operate the DSP are provided by Texas Instruments, but they have not updated these drivers for years. The DM3730 has been superseded by a new line of processors and therefore, TI decided to deprecate the support for this processor.

Using the FPGA for formatting is a flexible option, but requires writing the formatting conversion in VHDL which is a cumbersome process.

On the ARM-core the formatting consists of reading in the RAW image data in OpenCV and using a conversion function from the OpenCV library to format it to for example RGB.

3.2.4 Overview

To summarize, there are two types of cameras: 'basic' and 'advanced'. These can interface through three different connections to the Gumstix: USB, ISP, or FPGA. Finally there are five different options for the formatting: on the camera, in the ISP or FPGA, or on the Gumstix in either the ARM-core or the DSP.

Combining the cameras and interfaces results in the paths shown in figure 3.1. As can be seen there are seven paths from either the 'basic' camera or the 'advanced' camera through the different connections to the Gumstix.

As denoted in the remarks of table 3.1 the USB interface only has one path associated with it. This is because the USB camera that is used is an 'advanced' camera and no 'basic' mode is available. When using the ISP or the FPGA either the 'basic' or an 'advanced' camera can be used.

There is some overlap between the paths shown in figure 3.1. Therefore, it is not necessary to implement every single path to test all the possibilities.

The paths numbered 1, 3, and 4 are implemented. Even though path 3 is not an obvious choice when implementing this system in a real-life situation it is the easiest to implement to test the formatting on the ARM-core. The ISP is easy to configure and writing extra VHDL for the FPGA is more cumbersome.

3.2.5 Stereo vision

The paths described above are not limited to single cameras. It is also possible to create a stereo-vision setup. The ISP is equipped to handle two pixel-streams, but as explained in Section 2.1.1 this can only be done with either two serial cameras or a serial and a parallel camera.

There are serial cameras available, but creating a stereo-vision setup using the ISP is not done for the following reasons:

- If two serial cameras are used they both need a different I^2C address, because they share the I^2C bus that is used for setting up the cameras. Using the same cameras is therefore not an option and complicates the stereo-vision setup.
- Any stereo-vision setup using the ISP requires the design of a custom PCB to be able to connect two different cameras to the ISP connector. It is chosen not to do this due to time constraints on the project.

The FPGA on the other hand, is more flexible and can be configured to handle two pixel-streams from two parallel cameras simultaneously. The FPGA on the RaMstix also has enough

I/O pins to interface with two cameras. A separate I^2C can be implemented for either camera. This means two identical cameras can be used.

To test the stereo-vision setup, path 5 is implemented in duplex to interface the cameras. Paths number 6 and 7 can also be used, but these are not implemented because using the advanced camera is easier than implementing the formatting in either the FPGA or on the ARM-core.

3.3 Next steps

Based on the design space exploration and the choices made, several requirements are made that are used in the design and implementation. These requirements are for the work to be done and not for a final product based on this project. The requirements are prioritized using the MoSCoW method.

Requirement 1: *An image must be retrieved through the either of the three interfaces.*

Requirement 2: *The resulting image must be formatted.*

Requirement 3: *A path must be implemented where the formatting is done in the camera.*

Requirement 4: *A path must be implemented where the formatting is done in the ISP.*

Requirement 5: *A path must be implemented where the formatting is done in the ARM-core.*

Requirement 6: *A path should be implemented where the formatting is done in the FPGA.*

Requirement 7: *The resource usage for implemented paths must be measured.*

Requirement 8: *Two sensors could be used in a stereoscopic setup.*

Requirement 9: *Hardware acceleration using the DSP/NEON won't be implemented.*

These requirements cover all the possible interfaces and all the possible places to do the formatting.

3.4 Conclusion

As shown in the design space exploration, Section 3.2, There are three different interfaces to connect a camera to the Gumstix. These are the USB connection, the ISP, or through the FPGA on-board the RaMstix.

There are also five possible places to do the formatting of the image: on the camera, in the ISP, the FPGA, using the DSP, or on the ARM-core itself.

Using a combination of the cameras, interfaces and formatting possibilities results in seven possible paths to obtain a formatted image that can be used for further processing.

Four of these paths are implemented to meet all the requirements set in Section 3.3. These are paths 1, 3, 4 and 5. The last path is used to implement stereo vision.

In the next chapter the design and implementation of these paths are discussed.

4 Design and implementation

The paths that are implemented are path number 1, 3, 4 and 5. As is mentioned in Chapter 3 the last path will be used for the stereo-vision setup. The design of each of these paths is detailed in its own section. Path number 5 is split into a section detailing the implementation using a single camera and a section detailing the design of the stereo-vision setup.

An overview of all the paths that are implemented is shown in figure 4.1. The 'F' again denotes where the formatting is done and the dashed lines separate the different parts of the pipelines.

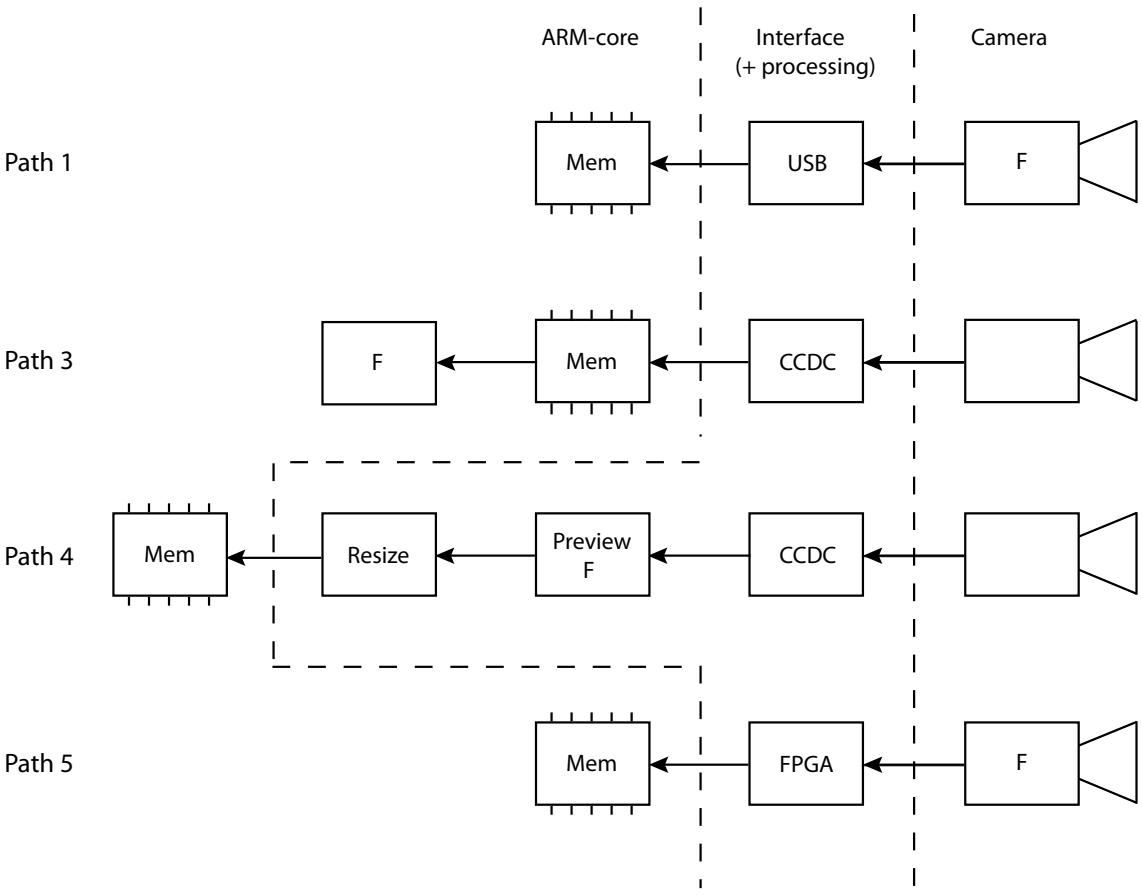


Figure 4.1: An overview of the implemented paths

The basic setup that is used for all the designs consists of a RaMstix board with the software as is defined in Section 2.3. The goal of each design is to implement the path from the camera to a usable video device in the Linux environment running on the Gumstix. Each of the paths is explained into detail in the sections below.

4.1 Path 1: USB advanced camera

This section describes the design of the first path using a USB webcam where the formatting is done in the camera. This is the path that is currently used at the ESL course. An overview of this path is shown in figure 4.2.

Using the Logitech C250 USB webcam is the simplest way to add a vision system to the Gumstix. The physical interface is straightforward and with the correct USB drivers present on the Gumstix the camera shows up as a **/dev/video** device.

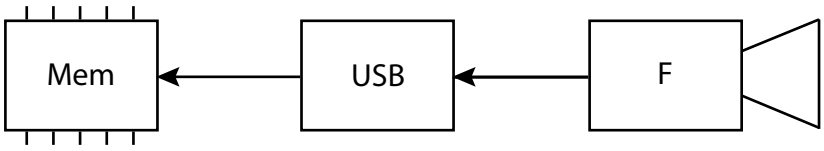


Figure 4.2: Path 1: USB with the advanced camera

With the OpenCV software this video device can be used to retrieve frames and perform further processing when necessary.

4.2 Path 3: ISP basic camera formatting in ARM

This section details the design and implementation of the setup using the Caspa camera connected to the Gumstix through the ISP where the formatting of the image is done in software on the ARM-core. An overview of this path is shown in figure 4.3.

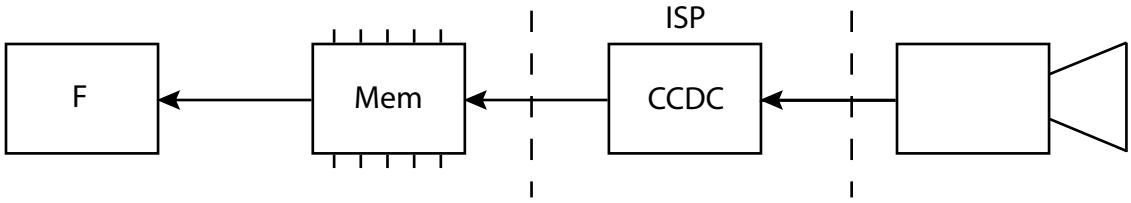


Figure 4.3: Path 3: ISP with the basic camera and formatting in the ARM-core

The steps taken to make the Caspa camera show up as a **/dev/video** device are described below.

4.2.1 Hardware design

To use the Caspa camera with the ISP, an internal pipeline is created to retrieve the data and then store it in memory. The ISP has an internal structure that is shown in figure 4.4.

It can be seen that there are several possible pipelines from 'RAW Bayer' to memory. For this design the pipeline is:

- Camera -> CCDC -> Memory (unformatted)

The CCDC is responsible for accepting unformatted data from the camera as is explained in Section 2.1.1.

Configuring the pipeline of the ISP is done using the ***media-ctl*** program. The command to create the pipeline listed above is shown in listing 4.1.

```
1 media-ctl -r -l '"mt9v032 2-005c":0->"OMAP3 ISP CCDC":0[1],
2 "OMAP3 ISP CCDC":1->"OMAP3 ISP CCDC output":0[1]'
3
4 media-ctl -V '"mt9v032 2-005c":0 [SGRBG10 752x480],
5 "OMAP3 ISP CCDC":1 [SGRBG10 752x480]'
```

Listing 4.1: ISP pipeline: camera straight to memory

The mt9v032 represents the Caspa camera and the other names represent the pipeline entities. Each of the first two lines represents a link between two entities and lines 4 and 5 dictate the data format and resolution that is used in that link. The first link is from the camera to the CCDC, this is the retrieval of data. The second link couples the CCDC to the CCDC output, this writes the acquired data to memory.

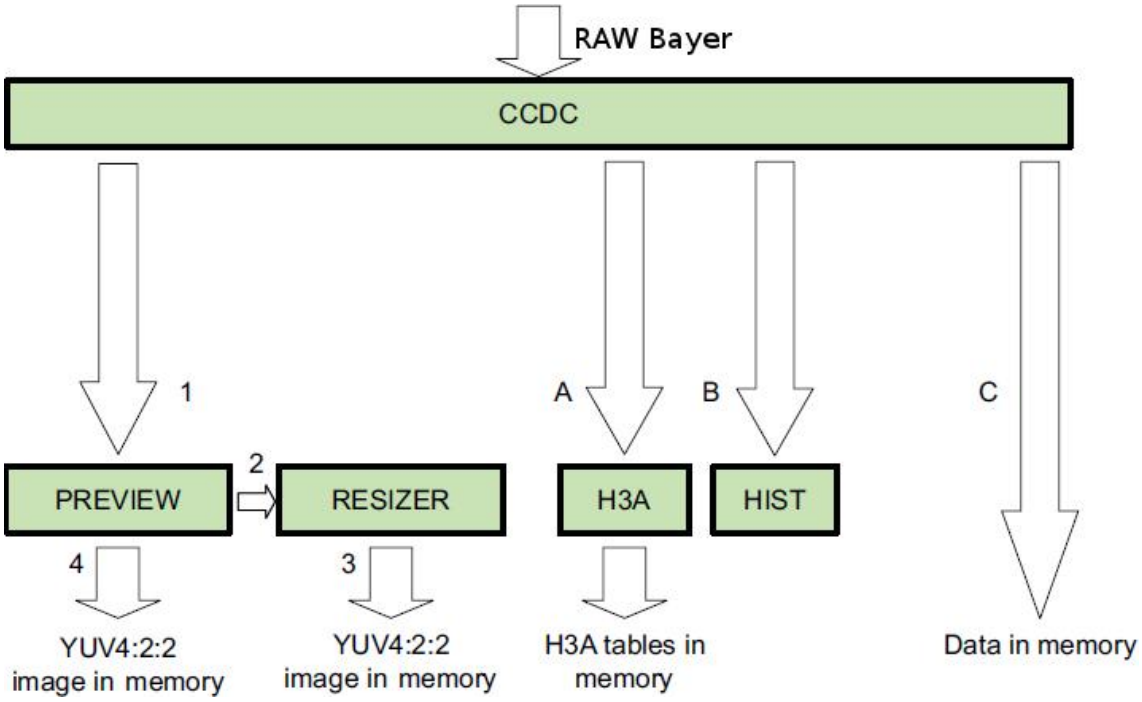


Figure 4.4: Camera ISP internal pipeline (CompuLab, 2015).

The format of the retrieved image is SGRBG10 which represents 10-bit RAW Bayer. The image still needs formatting before it can be used. This is done in software on the ARM-core.

4.2.2 Software design

When the setup of the pipeline is complete the output of the CCDC will show up as **/dev/video2**. This video source generates unformatted images that need further processing by the OpenCV software.

The way this is done is by using a *VideoCapture* object from OpenCV and using the built-in functionality to retrieve a frame from the object. OpenCV uses the *Mat* object as the basic image container. When retrieving a frame from the *VideoCapture* object it has to be stored in a *Mat* object.

The formatting of the RAW image data can be done using the *cvtColor()* function of OpenCV. It can convert the image contained in the *Mat* object from SGRBG10 to RGB.

The code snippet shown in listing 4.2 shows how a frame can be retrieved and then formatted using OpenCV.

```
1  int main(){
2
3    VideoCapture camera(2); // open camera on /dev/video2
4
5    Mat RAW_frame, formatted_frame;  // create empty containers
6    camera >> RAW_frame; // retrieve a frame
7    cvtColor(RAW_frame, formatted_frame, CV_BayerBG2RGB); // convert the frame from
       SGRBG10 to RGB
8
9    imwrite("test.png", formatted_frame); // write frame to image file
10
11   return 0;
12 }
```

Listing 4.2: OpenCV code to retrieve and format frames

The *cvtColor()* function outputs the formatted frame in a *Mat* object and this can be used for further processing.

4.3 Path 4: ISP basic camera formatting in ISP

In this section the design and implementation of the setup using the Caspa camera while formatting the image in the ISP is detailed. An overview of path 4 is shown in figure 4.5.

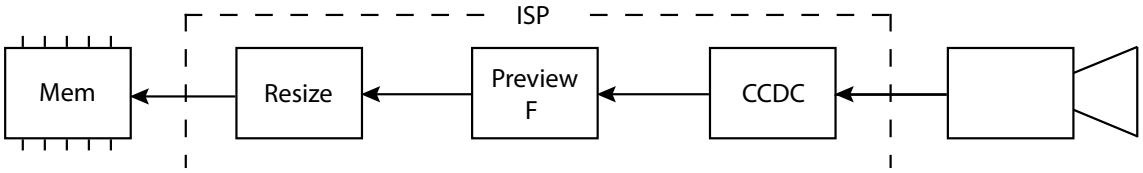


Figure 4.5: Path 4: ISP with the basic camera and formatting in the ISP

4.3.1 Hardware design

As in the previous section this consists of setting up an internal pipeline in the ISP, but this time to retrieve data, format it and then store it in memory. The same structure as shown in figure 4.4 is used, but the pipeline design has changed to:

- Camera -> CCDC -> Preview -> Resizer -> Memory (formatted)

The previewer element does the on-the-fly conversion to a known format and the resizer crops the image to a configured resolution as is explained in Section 2.1.1.

The ***media-ctl*** command shown in listing 4.3 will create the pipeline listed above.

```
1 media-ctl -r -l '"mt9v032 2-005c":0->"OMAP3 ISP CCDC":0[1],
2 "OMAP3 ISP CCDC":2->"OMAP3 ISP preview":0[1],
3 "OMAP3 ISP preview":1->"OMAP3 ISP resizer":0[1],
4 "OMAP3 ISP resizer":1->"OMAP3 ISP resizer output":0[1]'
5
6 media-ctl -V '"mt9v032 2-005c":0 [SGRBG10 752x480],
7 "OMAP3 ISP CCDC":2 [SGRBG10 752x480],
8 "OMAP3 ISP preview":1 [UYVY 752x480],
9 "OMAP3 ISP resizer":1 [UYVY 640x480]'
```

Listing 4.3: ISP pipeline: camera to formatting to memory

The first link, on line 1, is from the camera to the CCDC, this is the retrieval of data. The second link, line 2, is from the CCDC to the previewer, here the image is formatted to the UYVY format. The third link, line 3, couples the previewer to the resizer, this changes the image resolution from 752x480 to 640x480. The final link, line 4, from the resizer to the resizer output writes the data to memory.

When the setup of the pipeline is complete the output of the resizer will show up as **/dev/video6**. This generates formatted images and can be used by the OpenCV software to grab frames for further processing.

4.4 Path 5: FPGA advanced camera

The fifth path for interfacing a camera to the Gumstix is through the FPGA. The Omnivision OV7670 advanced camera is used where the formatting is done in the camera. An overview of this path is shown in figure 4.6.

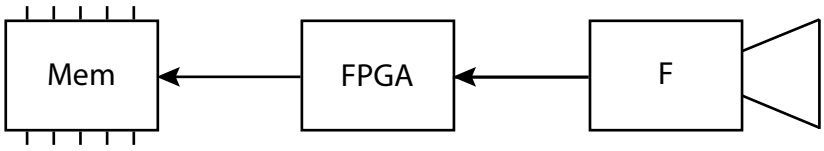


Figure 4.6: Path 5: FPGA with the advanced camera

This path consists of retrieving image data with the FPGA and transferring the formatted image to the Gumstix through the GPMC bus. To accomplish this the design for this path consists of four major components shown in figure 4.7 and listed below:

- An I^2C interface to configure the camera.
- A component to retrieve the image data (frame capture).
- Block RAM to store the image (frame buffer).
- A GPMC interface to transfer data to the Gumstix.

These four component are implemented in VHDL on the FPGA. On the Linux side in the Gumstix the GPMC driver created at RaM is included to interface with the GPMC. To retrieve the image data from the FPGA a custom class is written to communicate with the FPGA through the GPMC bus. It retrieves the image data and puts it in a format that is understood by the OpenCV software.

The image generated by the OV7670 has a resolution of 160x120 pixels with the RGB565 formatting. This small frame size is chosen, because the resources in the FPGA are limited and storing a frame of this size will use 27% of the storage resources. When implementing stereo vision, as will be explained in the next section, more than half of the storage resources will be used and retrieving a bigger frame is not possible.

The design of the VHDL components and the software to retrieve image data through the GPMC will be discussed in Sections 4.4.1 and 4.4.2.

4.4.1 Hardware design

The FPGA has to perform the task of initializing the camera, retrieving the data and making it available to the Gumstix through the GPMC bus. An overview diagram of the design to accomplish this is shown in figure 4.7. This design is the realization of the 'FPGA' block in figure 4.6.

I^2C interface

The Omnivision OV7670 camera needs to be configured with the right settings for it to generate images. The settings that can be configured are the formatting of images by the internal DSP, the resolution of the images and at what speed the camera produces frames. These settings are communicated to the camera through an I^2C interface.

Instead of creating the I^2C interface from scratch a ready-made solution can be found at eewiki (Larson, 2015). It implements an I^2C-master that connects to the camera I^2C bus denoted by the SIOC and SIOD lines in figure 4.7. It is chosen, because it is easy to include in the design and there is proper documentation of how it works. A wrapper is written to utilize the I^2C-master component and it includes the correct register values to setup the camera.

Once the camera is correctly initialized it starts streaming frames one byte at a time.

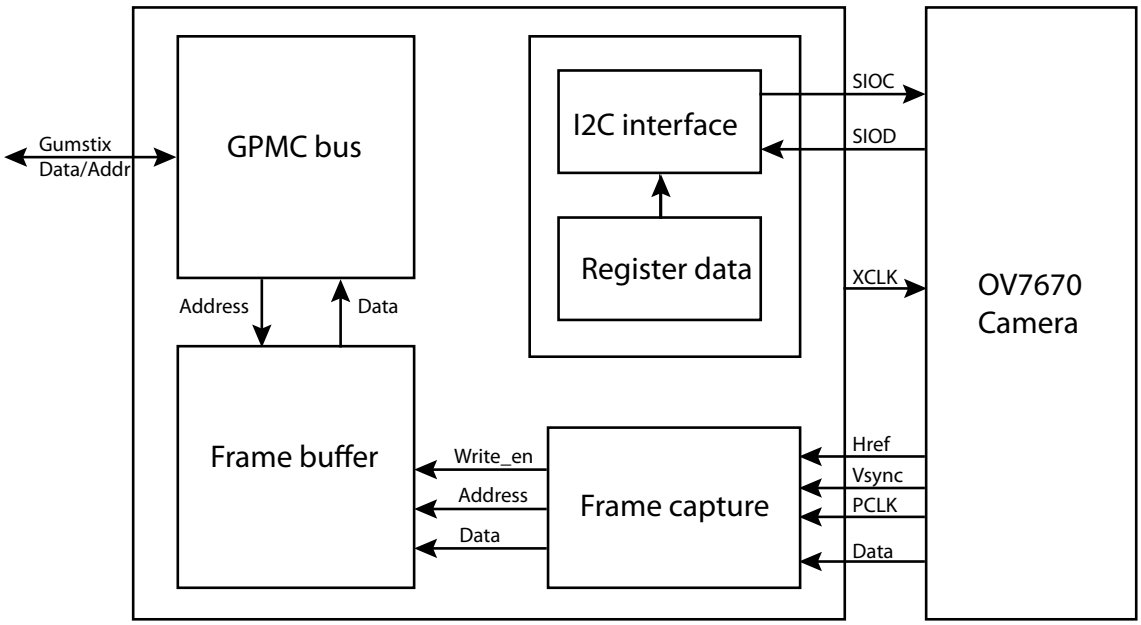


Figure 4.7: Overview diagram of the hardware design in the FPGA.

Frame capture and storage

The frame capture component takes clock and data inputs from the camera to be able to capture frames that are being streamed. Each frame has a specific timing dictated by the vertical synchronization *Vsync*, horizontal reference *Href* and pixel clock *PCLK*. The FPGA uses these lines as triggers to identify when to sample the data bus and read in one byte of the frame.

Because the RGB565 format is used, one pixel consists of two bytes. These two bytes are concatenated and written to the frame buffer. The frame buffer is a piece of Block RAM generated using an Altera-provided megafunction. It uses 16-bit registers and can store an entire frame which adds up to 38.400 bytes of RAM. This uses a 15-bit address space.

GPMC bus interface

The GPMC bus interface is derived from the provided examples created at RaM (Schwirtz, 2014). The example has internal registers that can be accessed by the Gumstix driver. These registers are 16-bit wide as is the actual bus. The address bus is 10-bits wide, as is discussed in Section 2.1.2, and therefore the address space is also limited to 10-bits. These internal registers act as a block of RAM, just like the frame buffer, and the Gumstix can access this RAM by addressing the registers.

Because the address space of the GPMC bus is limited to 10-bits and the address space of the frame buffer is 15-bit wide the frame buffer cannot be used as direct replacement of the internal registers. The Gumstix would not be able to access all of the registers in the frame buffer that way. Therefore, the GPMC component is altered to contain a line buffer that fits within the 10-bit address space. This line buffer can hold one horizontal line of the image stored in the frame buffer. The alteration consists of adding extra logic to the GPMC component that can communicate with the frame buffer and retrieve one line of the image to store in the line buffer. Also some extra steps are taken to make sure the communication between the Gumstix and the FPGA includes a transfer protocol.

The communication between the Gumstix and the FPGA always has to be initialized by the Gumstix. The implementation described above awaits a request from the Gumstix for a specific

line, buffers the line and then signals the Gumstix that the line can be retrieved. This simple protocol allows the Gumstix to retrieve frames one line at a time without missing any data.

4.4.2 Software design

On the Gumstix side a kernel driver is used to interface with the GPMC bus. This driver has been implemented before in the RaMstix project and can be used without modification. Example programs are also available in C and C++ that show how to retrieve 32-bit values from the internal registers of the GPMC implementation on the FPGA (Schwirtz, 2014).

Based on these examples, a Camera class is written in C++. A UML diagram showing the Camera class and the required gpmc_driver is shown in figure 4.8. The Camera class takes care of retrieving frames from the FPGA one line at a time while adhering to the established protocol described in the previous section. The gpmc_driver class provides the low level interface with the GPMC bus.

The *getValue()* function from the gpmc_driver returns 32-bit integers that each represent two pixels from the image. Using the *convert_to_bgr()* function this can be structured to an array containing the red, green and blue channels of the image.

The Camera class contains a single public function *grab_frame()* that wraps up the whole process of retrieving a frame from the FPGA, structuring it into the array that can be put into a *Mat* object and returning a pointer to that array.

Using this class, the camera connected to the FPGA can be used to obtain frames for further processing in OpenCV.

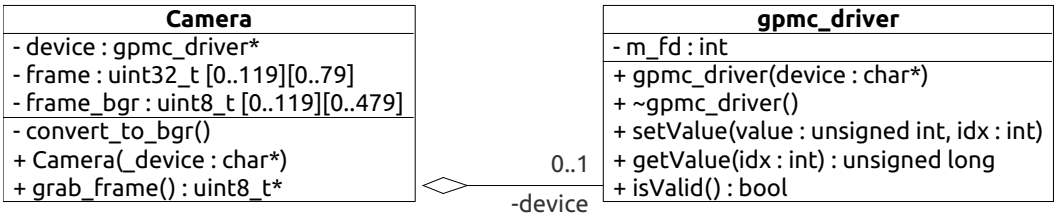


Figure 4.8: UML diagram showing the Camera class and the gpmc_driver

4.5 Path 5: FPGA advanced camera stereo vision

This section describes an extension to the previous path so that it can be used for stereo vision. The stereo-vision setup contains two Omnivision OV7670 advanced cameras. These cameras are connected to the FPGA on the RaMstix and the design of the hard and software is similar to the path detailed in the previous section. An overview of the implementation is shown in figure 4.9.

The cameras are setup to generate an 160x120 pixels image with the RGB565 formatting and both cameras are read simultaneously.

In figure 4.9 it can be seen that there are two external cameras and most components also used in the previous section are now used twice. There are two I^2C interfaces, two frame capture components and two frame buffers.

The I^2C component is still the same as the one used in the previous section only this time the I^2C-master and the register data are shown as one block called I^2C interface.

The GPMC bus is still a single component. The internals of the GPMC bus component are altered to allow reading from both the frame buffers at once. The lines retrieved from the frame buffers are concatenated into a single 320 pixel line. The result of this is that the software on

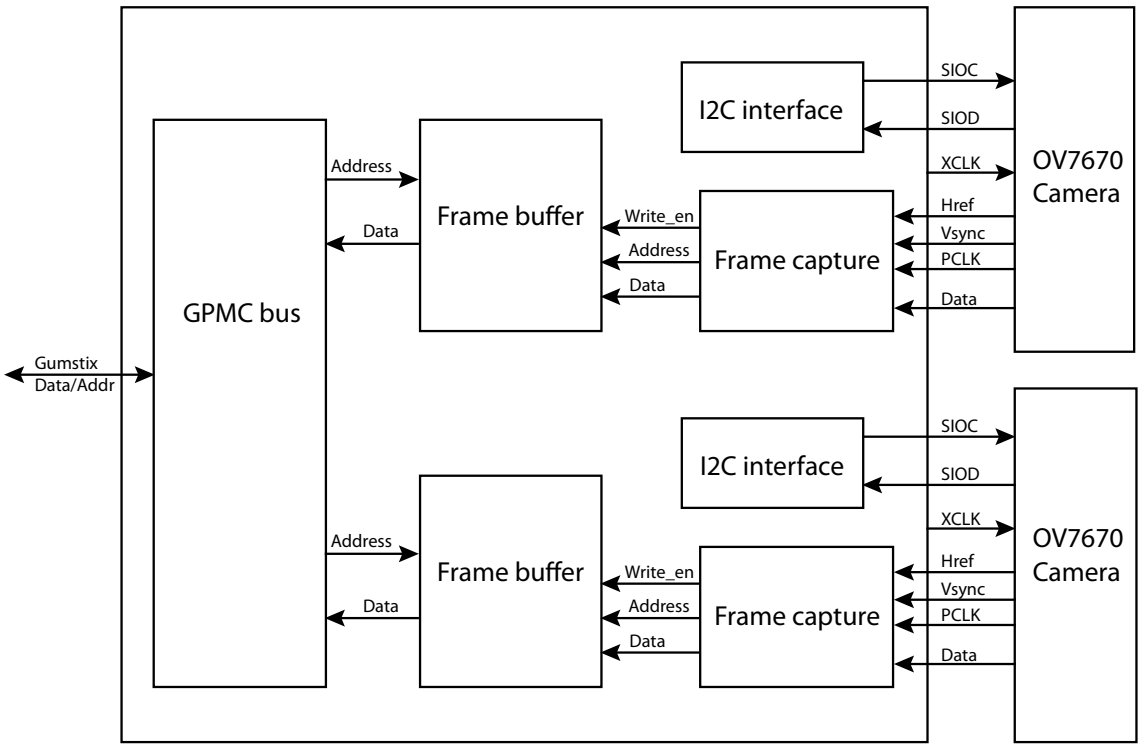


Figure 4.9: Overview diagram of the hardware design in the FPGA for stereo vision.

the Gumstix can grab both frames at the same time. The same Camera class that is discussed in Section 4.4.2 is used. The resulting image in software has a resolution of 320x120 pixels and is made available as an array object that can directly be used by the OpenCV software.

The next chapter details the testing and measuring of the designs presented in this chapter.

5 Measurements and results

The implemented paths as described in Chapter 4 are tested to determine the impact each path has on the performance of the Gumstix.

The performance of the Gumstix is related to the processor load and the memory usage of a certain path. These resources are also used by other programs running on the Gumstix. Therefore, less resource usage leaves more room for other programs like the image processing.

In the sections below the basic setup and the measurements are explained followed by the measurements done for the different paths in separate sections.

5.1 Basic test setup and measurements

The test setup consists of the RaMstix with the Yocto generated Linux-image as is described in Appendix A.

The hardware of the test setup consists of the Caspa camera connected to the ISP connector as is shown in figure 5.1 denoted with the yellow box.

The Logitech C250 camera is connected to the USB port of the RaMstix denoted by the green box in figure 5.1 and two OV7670 cameras are connected to the FPGA denoted by the purple box. Depending on the setup of the FPGA either one camera can be used to test path 5, or both cameras can be used to test the stereo-vision setup with path 5. The pin-layout and instructions for initializing the FPGA setup are described in Appendix B.

To be able to compare the obtained results from paths 1 and 4 with path 5, the resolution of the retrieved images is set to 160x120 pixels. This is because path 5 is limited to using this resolution, because the storage in the FPGA is limited as is explained in Section 4.4. Measurements with the default resolution of 640x480 pixels are also done with path 1 and 4 to determine their performance.

5.1.1 Measurements

Two different measurements are performed to test different aspects of the cameras. These test are the following:

- Compare test
- Load test

The compare test is used to compare the different paths with each other. It uses an OpenCV program to retrieve a set number of frames. The time that this process takes is recorded together with the processor load and peak memory usage.

The load test is done to determine the resource usage of a single path without any overhead. To this end GStreamer is used to eliminate the overhead created by OpenCV.

Path 5 uses a different infrastructure than the other paths, because it uses the FPGA and the GPMC bus. It shows up as a **/dev/gpmc_fpga** device and the C++ class that is created is needed to retrieve frames from it. The other paths infrastructure result in a **/dev/video** device that can be directly used as a video source without the need for a special class to retrieve frames.

Because of this difference it is not possible to setup a GStreamer pipeline using the FPGA connected camera as source. This would require a different driver to be written that combines the GPMC driver with a video driver. This also requires the a more elaborate two-way communication that can get and set the camera configuration through the FPGA. This is a time consuming process and therefore not implemented.

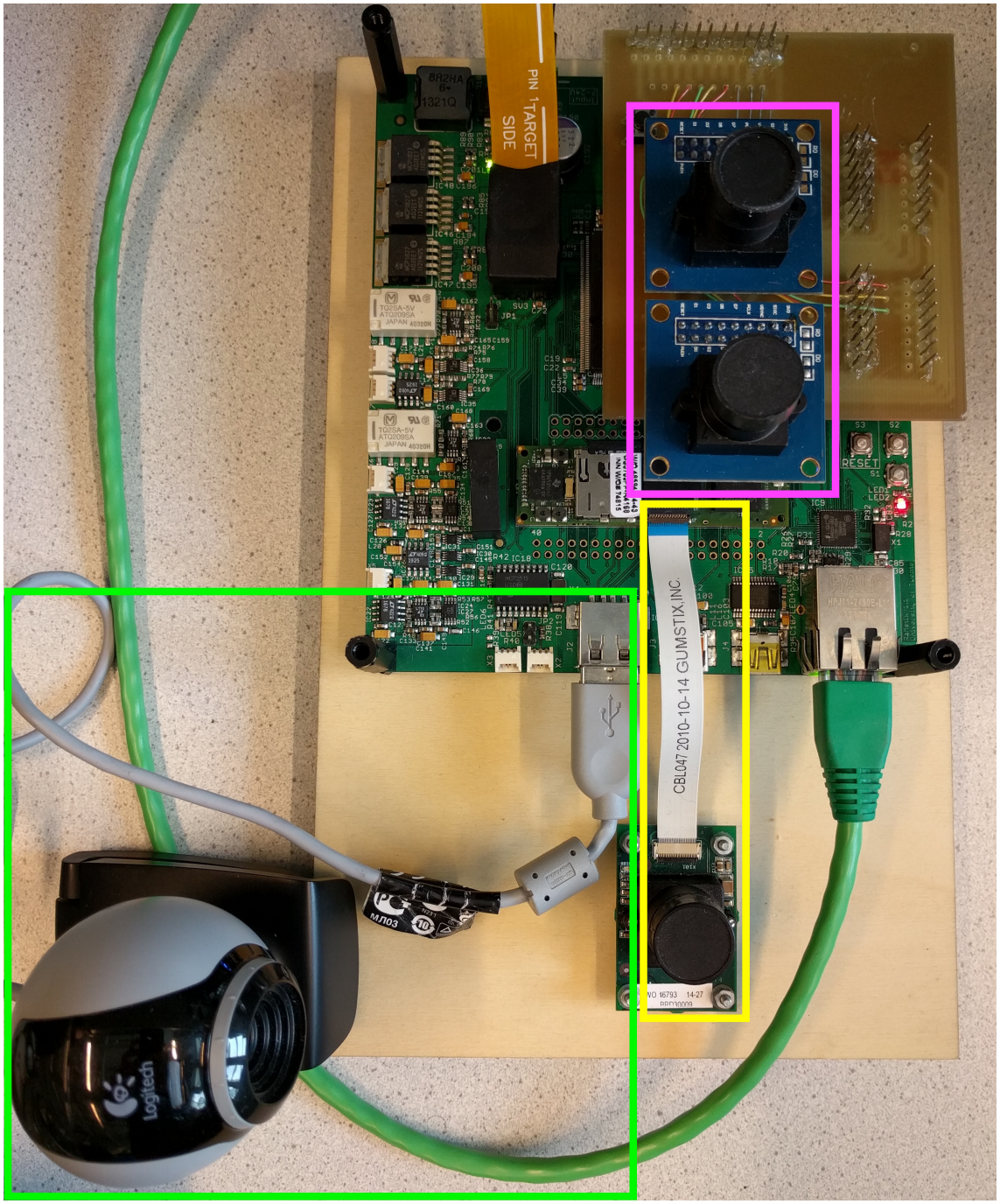


Figure 5.1: RaMstix with connected cameras denoted by the colored boxes.

This means that the load test can only be performed on the paths that result in a **/dev/video** device and not for path 5 using the FPGA and GPMC bus.

Compare test

To perform the compare test measurement an OpenCV program is written that retrieves 100 frames through each of the paths and the time that this process takes is recorded. The results of this test show the difference in speed and processor load each path has.

To measure the duration of retrieving 100 frames as well as the memory usage and processor load during the process the program 'time' is used. The processor load in 'time' is defined as the

number of CPU-seconds the process spent in kernel mode plus the number of CPU-seconds spent in user mode divided by the elapsed real time the process took to execute as is explained in Section 2.3.2.

The essentials of the OpenCV program that is used to retrieve the 100 frames through each path are shown in listing 5.1.

```
#define number_of_frames 100

int main() {

  VideoCapture cam(X);  // open camera on /dev/videoX

  Mat frame;          // create an empty image container

  for(int i=0; i<number_of_frames; i++) { // repeat 100 times
    cam >> frame;    // retrieve a frame from the camera
  }

  return 0;
}
```

Listing 5.1: OpenCV code to retrieve 100 frames

For path number 5 the OpenCV program is slightly altered to communicate with the Camera class instead of the *VideoCapture* object for image retrieval.

Because the tests are run on a Linux system taking a single measurement of a 100 frames is not an accurate depiction of the speed of the path. There are other processes running simultaneously on the system that also require resources and can interfere with the measurement. Therefore, this measurement is automated with a script that performs the test a 100 times to generate more data points. The results of each separate test are stored in a text file. These are then used to calculate an average time that is needed to retrieve a 100 frames, the average processor load and the average memory usage.

Load test

To perform the load test a GStreamer pipeline is created that sets the camera to a fixed frame rate of 30 frames per second. The output of the pipeline is written to a *fakesink* that effectively discards the data. By using a *fakesink*, a better estimation of the resources used by the camera itself can be made. When writing the output of the pipeline to a file some of the measured resource usage is due to accessing and writing to the file system. The results are again obtained by using the program 'time' and as mentioned before this test can only be run on paths that result in a **/dev/video** device.

To setup this pipeline the command shown in listing 5.2 is used.

```
gst-launch-1.0 -e v4l2src device=/dev/videoX num-buffers=100 ! video/x-raw,
    framerate=30/1 ! fakesink
```

Listing 5.2: GStreamer pipeline with fixed frame rate

The tests are automated again to obtain a 100 measurements of every implemented path to account for the variation in used resources that is the result of running the process on a Linux system.

The processor load and memory usage are recorded to text files and again averaged to obtain an approximation of the impact each path has on the Linux system.

The time that the process takes is not recorded for the load test. When the GStreamer pipeline is created it will spent an amount of time in a paused state to allocate buffers and wait for the

cameras to be ready to stream data. The duration of this paused state is random and therefore the otherwise recorded time is not representative.

5.2 Path 1: USB advance camera

Path 1 uses the Logitech C250 camera connected to the USB port of the RaMstix where formatting of the images is done in the camera. Images travel the path shown in figure 4.2. Both the compare test and the load test are run for this setup and the results are discussed below.

All the measurement results of the compare test and load test, which are each repeated 100 times, are shown only in this section to show the results can fluctuate. The following sections detailing the other implemented paths only show the resulting averages instead of all the graphs.

5.2.1 Compare test path 1

The results of the compare test using path 1 with a resolution of 640x480 pixels are shown in figure 5.2.

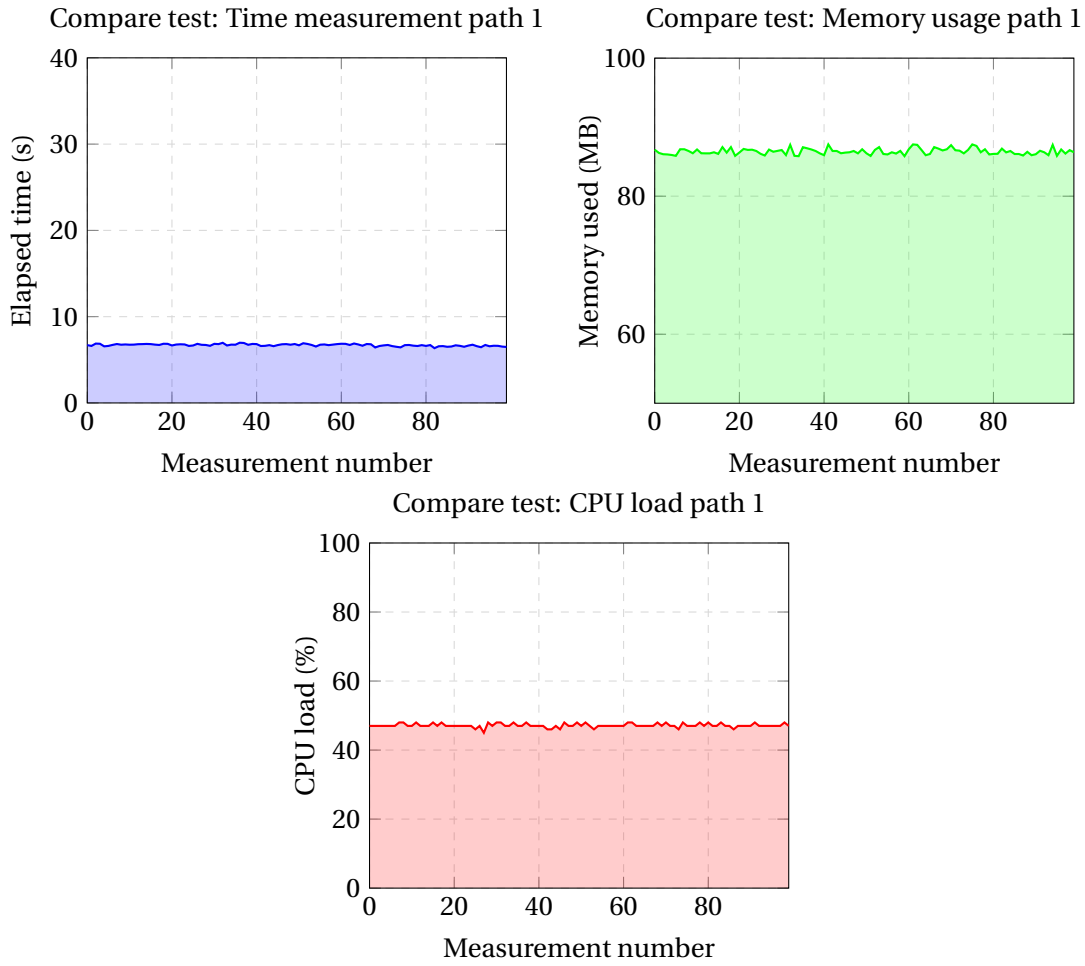


Figure 5.2: Compare test results for path 1

As can be seen the measured elapsed time, memory usage and CPU load are altering slightly between the measurements, but there are no large deviations. The averages and standard deviations of the measurements are:

- Elapsed time: 6,71 s $\sigma = 0.126$

	Compare test		Load test	
	(640x480)	(160x120)	(640x480)	(160x120)
Elapsed time (s)	6,71	4,32		
CPU load (%)	47,14	7,97	3,22	3,01
Memory usage (MB)	86,48	59,26	28,72	24,84

Table 5.1: Measurement results path 1

- Memory usage: 86,48 MB $\sigma = 0.447$
- CPU load: 47,14% $\sigma = 0.569$

5.2.2 Load test path 1

The results of the load test with a resolution of 640x480 are shown in figure 5.3. Again there are

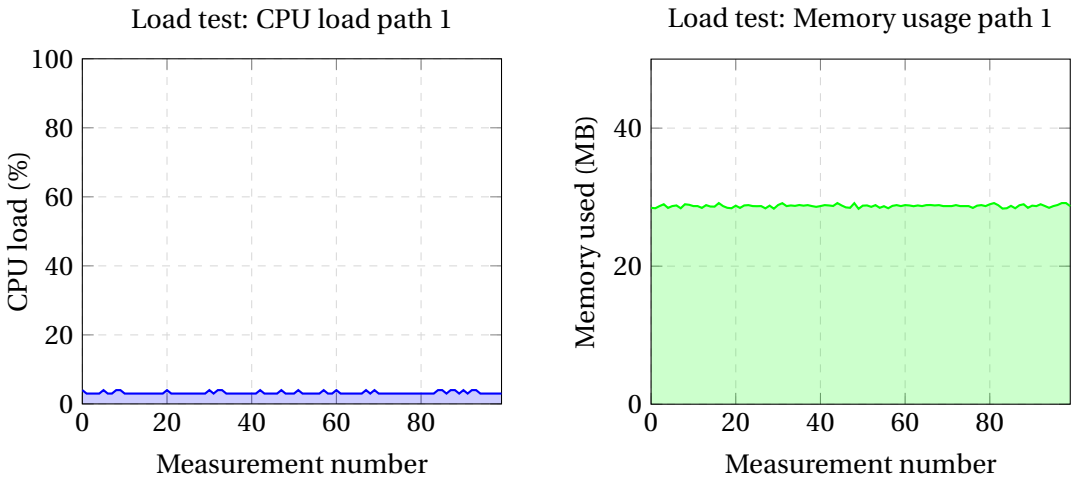


Figure 5.3: Load test results for path 1

no large deviations, but now it can be seen that the load on the CPU is significantly lower when creating a direct pipeline with GStreamer instead of retrieving frames through OpenCV. It can be seen that the memory usage is also lower. This is due to the use of GStreamer and writing to a *fakesink*, which uses no extra memory to buffer the resulting image as opposed to OpenCV. The averages and standard deviation of the measurements are:

- CPU load: 3,22% $\sigma = 0.416$
- Memory usage: 28,72 MB $\sigma = 0.201$

The compare test and load test are also run with the lower resolution of 160x120 pixels to be able to compare the results with path 5. The averages of these measurements are shown in table 5.1 that shows an overview of all the results obtained for path 1.

5.3 Path 4: ISP basic camera formatting in ISP

Path 4 uses the Caspa camera connected to the ISP interface and the formatting of the image is done in the ISP. Images travel the path shown in figure 4.5. Both the compare test and load test are run using this path and the averages of the results for path 4 are listed in table 5.2.

5.4 Path 5: FPGA advanced camera

Path number 5 uses the Omnivision OV7670 camera connected to the FPGA where the formatting is done in the camera. Images travel the path shown in figure 4.6. Only the compare test is

	Compare test		Load test	
	(640x480)	(160x120)	(640x480)	(160x120)
Elapsed time (s)	37,92	3,13		
CPU load (%)	49	48,61	37,42	12,98
Memory usage (MB)	86,43	59,60	33,36	25,16

Table 5.2: Measurement results path 4

run on this path, because the camera is interfaced through the FPGA and does not show up as a **/dev/video** device as is explained in Section 5.1.1. The test is only run on the lower resolution, because the FPGA memory is limited and cannot handle a larger image size as is explained in Section 4.4.

The averages of the compare test for path 5 are shown in table 5.3.

	Compare test (160x120)
Elapsed time (s)	2,91
CPU load (%)	49
Memory usage (MB)	57,70

Table 5.3: Measurement results path 5

5.5 Path 5: FPGA advanced camera stereo vision

This path is similar to the path in the previous section, but now two cameras are used simultaneously.

The averages of the compare test for path 5 with stereo vision are shown in table 5.4.

	Compare test (320x120)
Elapsed time (s)	5,36
CPU load (%)	50,48
Memory usage (MB)	57,72

Table 5.4: Measurement results path 5 stereo vision

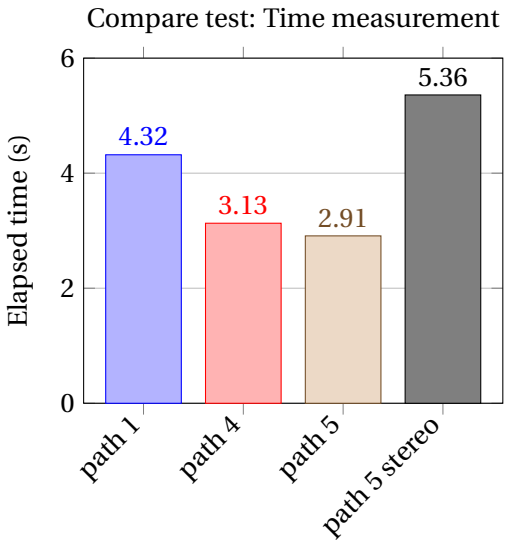
5.6 Comparison

In this section the results of the compare test and load test for different paths are compared and the results explained. Each comparison will be made in its own subsection.

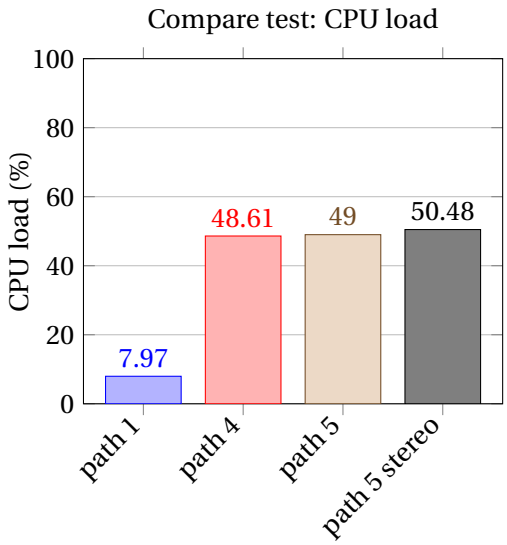
5.6.1 Compare test at low resolution

The results of the compare tests run with the lower resolution of 160x120 pixels are combined in figure 5.4 to clearly show the differences between the paths in terms of speed, CPU load and memory usage.

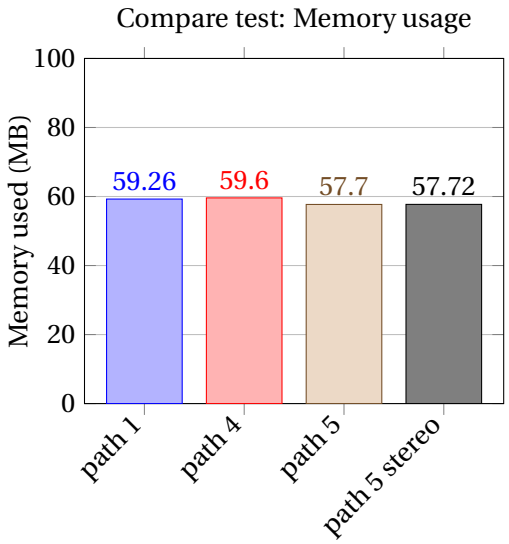
It can be seen in figure 5.4c that the memory usage for the compare test is the same for all of the paths. The CPU load as shown in figure 5.4b is also the same for all paths except path 1, the USB camera. Path 1 requires much less CPU resources than the other paths, this is most likely due to the DMA (Direct Memory Access) that is built into the USB hardware on the Gumstix. The other paths make use of a CPU intensive memory copy function to transfer the images from buffer to buffer. The USB camera in path 1 can use the DMA for this purpose.



(a) Time measurement



(b) CPU load measurement



(c) Memory usage measurement

Figure 5.4: Compare test comparison at low resolution

The time required to retrieve 100 frames through each of the paths is shown in figure 5.4a. It is the lowest for path 5. This is to be expected, because it has the largest bandwidth towards the processor. Images can be retrieved 16 bits at a time. The 4th path comes in second, because it has the second largest bandwidth towards the processor. In path 4 images are retrieved 12 bits at a time. The USB camera in path 1 is slower as is expected, because all the data has to be retrieved serially.

Path 5 using stereo vision is slowest. This is due to the fact that the stereo vision has to retrieve an image twice the size of the images the other paths have to retrieve.

The results of this test are also displayed in a scatter plot that plots the elapsed time against the CPU-load for all the paths. This is shown in figure 5.5. Points that are further to the left are faster and points further to the right are slower solutions. On the y-axis points that are higher impose a bigger load on the Gumstix and points that are lower have less of an impact on the Gumstix resources. Ideally a path would be both fast and impose a low impact on the Gumstix resources, but as can be seen a trade-off has to be made between the two properties.

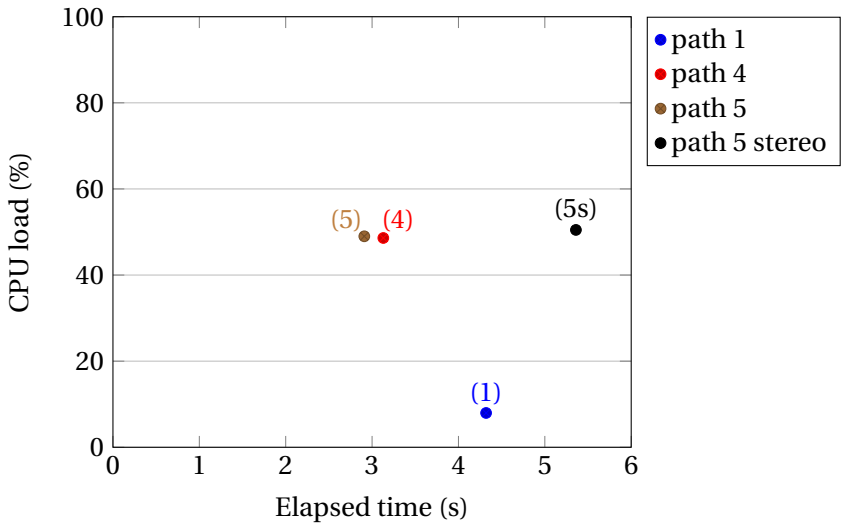


Figure 5.5: Scatter plot showing the elapsed time versus the CPU-load for each path

5.6.2 Compare test high resolution

The results of the compare test that is run on path 1 and 4 with an image resolution of 640x480 pixels are shown in figure 5.6.

The memory usage, which is linked to the image size, of the higher-resolution compare test is shown in figure 5.6c. It can be seen that the memory usage is equal for both path 1 and 4, but both are higher than the memory usage at a lower resolution.

The CPU load as shown in figure 5.6b is also equal for all paths. The higher CPU load for path 1, as compared to the compare test at lower resolution, is most likely due to the higher image resolution used and the memory copy function that is used in OpenCV. The images are still acquired using the DMA of the USB hardware, but also have to be transfered to another piece of memory by the OpenCV software.

The time to retrieve 100 frames through either path 1 or 4 is shown in figure 5.6a and is substantially higher for path 4. Even though the bandwidth from the camera to the processor is higher, the memory copy function most likely slows down the process of retrieving larger frames through path 4.

5.6.3 Load test

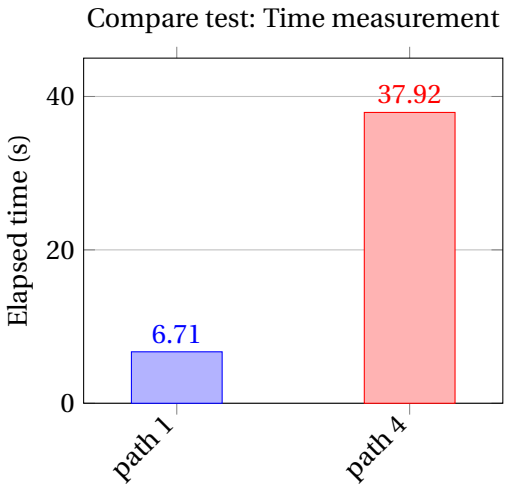
The load test is performed for path 1 and 4 and the results from the measurements are compared in figure 5.7.

With the fixed frame rate and a *fakesink* to deposit the retrieved images, the memory usage as shown in figure 5.7b is lower in comparison with the compare test on the same resolution. The reason for this is that the frame is not copied to a buffer allocated by OpenCV, but is discarded directly after retrieval.

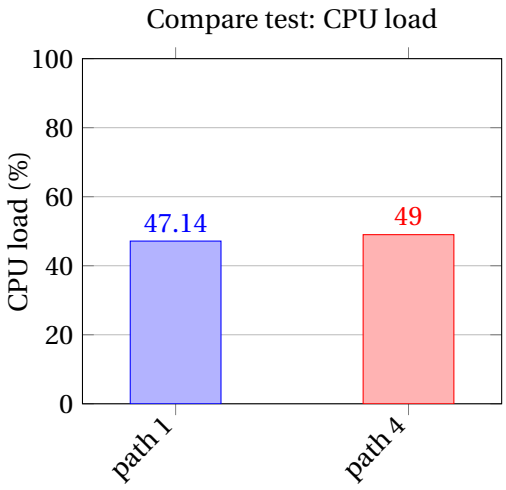
The CPU load needed for path 1 as show in figure 5.7a is very low. This is, as is explained before, due to the DMA present in the USB hardware and when streaming to a *fakesink* the memory copy function is not used at all. The CPU load for path 4 is lower in comparison with the compare test, but still has a considerable impact on the Gumstix processor.

5.6.4 Conclusion

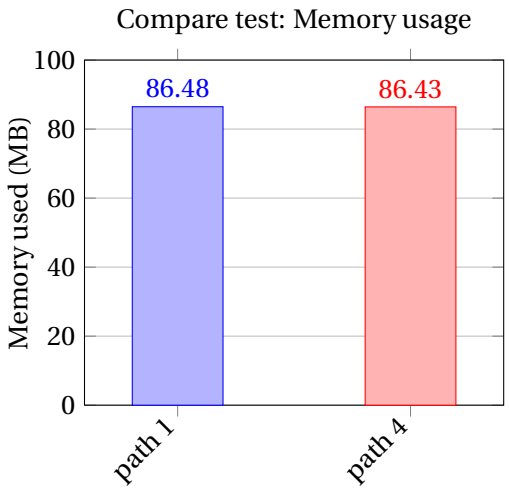
To conclude the comparison, it is shown that a camera interfaced through path 1 has the least impact on the Gumstix either sampling frames through a GStreamer pipeline or an OpenCV



(a) Time measurement

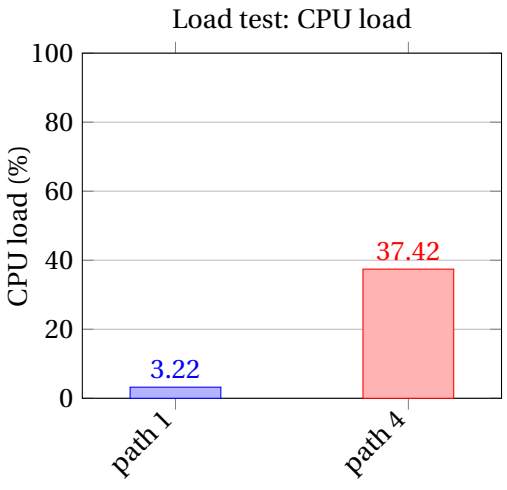


(b) CPU load measurement

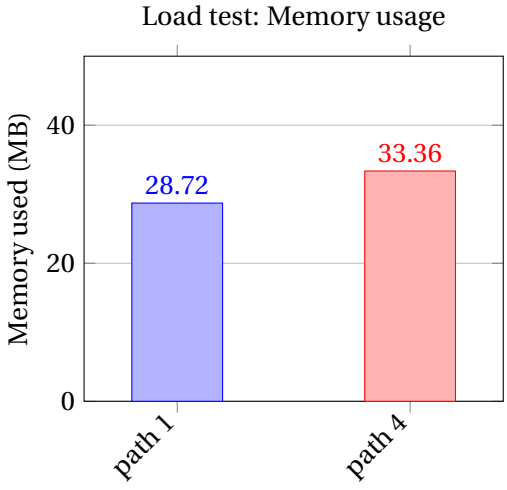


(c) Memory usage measurement

Figure 5.6: Compare test comparison at high resolution



(a) CPU load measurement



(b) Memory usage measurement

Figure 5.7: Load test comparison

program. This is due to the DMA that is present in the USB hardware. Path 4 is a good alternative when the same higher resolution of 640x480 pixels is needed, but requires more resources to operate.

When a lower resolution of 160x120 pixels is sufficient the high bandwidth of the FPGA interfaced cameras using path 5 result in a fast and flexible vision system that also allows for a stereo vision setup.

6 Conclusions and recommendations

The goals and conclusions from previous chapters are summarized in this final chapter. Also several recommendation for future work are given.

6.1 Conclusions

The goal of this project is to explore the paths that can be used to connect a camera to the Gumstix with the purpose of obtaining a formatted image that can be used for further processing.

To this end a design space exploration is done with the available hardware for the ESL course and extra cameras. Based on the design space exploration the RaMstix in combination with the Gumstix and three different cameras are chosen to be used.

Three different interfaces can be used to connect a camera to the Gumstix. These are the USB connection, the ISP, or through the FPGA on-board the RaMstix.

The three cameras that are chosen to be used with the interfaces are the Caspa camera using the ISP connection, the Logitech C250 with a USB connection and the Omnivision OV7670 that can be interfaced through the FPGA.

The formatting of the image can be done in five different places: on the camera itself, in the ISP, using the FPGA, using the DSP on the Gumstix, or on the ARM-core of the Gumstix itself. Of these options the DSP is not usable, because of outdated and deprecated drivers.

When combining the interfaces and formatting options seven possible paths to obtain a formatted image can be made. These paths are also not limited to single cameras, but some of them can be used to create a stereo-vision setup. Four of these paths have been implemented and tested including a stereo-vision setup.

The design and implementation resulted in a working setup where images made by the Caspa camera can be retrieved through the ISP and images made by the Omnivision OV7670 camera can be retrieved through the FPGA in either a mono, or stereo-vision setup.

The measurements performed on the implemented paths show that the USB connected Logitech C250 camera uses the least resources of the Gumstix. This is due to the DMA capabilities of the USB hardware that are not enabled for the other camera options. When an implementation is required that has restraints on the dimensions the Caspa camera connected through the ISP is a good alternative. It requires no extension board like the RaMstix to add an external USB connection, but can be directly interfaced to the ISP that is present on the Gumstix itself. It does require more resources to operate than the USB connected camera.

When a lower resolution vision system is sufficient the FPGA connected OV7670 camera is a fast and flexible option due to the high bandwidth of the interface. The path using the FPGA to interface cameras also allows for a stereo-vision setup. It is limited in resolution, because there are limited storage resources in the FPGA.

6.1.1 Embedded System Laboratory

The Caspa camera is operational and ready to be used in the ESL course. Also the FPGA-connected OV7670 camera is available to be used. The design-space-exploration phase of the course for the vision subsystem has been expanded with two new and different solutions. Each of the solutions uses different parts of the setup that is used at the course. This allows students to choose between implementing more of the vision-in-the-loop controller on the FPGA or the Gumstix based on their preferences.

These vision systems can also be used in other setups in the lab and also a stereo-vision system based on the OV7670 cameras and the RaMstix is now available.

6.2 Recommendations

The different possibilities of interfacing cameras to the Gumstix have been investigated and are partly implemented. It has been made possible to use the ISP-connected camera and the FPGA-connected camera with the Gumstix, but there are still several improvements to be made.

For the ESL course the recommendations are to find a different option to physically connect the Caspa camera to the Gumstix. In this project a flex-cable is used that is short and delicate. A more robust solution would allow for the Caspa camera to be built into the moving part of the JIWY setup.

The ESL manual should be updated with new information about how to connect the different camera options and how to use them from the Linux environment. The appendices of this report provide information about how to accomplish this. Several assignments can be added that deal with the setup of the hardware pipelines using 'media-ctl', or how to extract data from the OV7670 camera using the FPGA.

For research purposes the recommendations are to enable the DMA support for the ISP connection. This will lower the resource usage of the ISP-connected camera and make it even more suitable as a vision system.

Regarding the FPGA-connected camera a better Linux driver should be made that will make the camera show up as a **/dev/video** device. This can be accomplished by expanding the basic GPMC device driver with GStreamer options. When this is implemented the FPGA-connected camera also has a transparent interface instead of needing separate software for frame retrieval.

On the FPGA-side synchronization can be implemented between both cameras in the stereo-vision setup. Now both cameras are streaming data independent of each other, which can result in unsynchronized images being retrieved. Also formatting of images in the FPGA can be implemented. This will further reduce the resource usage on the Gumstix.

A Creating the Yocto image

This appendix explains how to recreate the software setup that is used in the report for interfacing with the Caspa camera.

A.1 Basic Yocto setup

The first step is to follow the instructions on https://github.com/gumstix/yocto-manifest until step 4. This explains the basic setup of obtaining a Yocto build and adding the meta-gumstix layers. As was explained in chapter 2.3.1 these layers provide the necessary software for the Gumstix specific hardware.

There are some extra instructions that need to be followed to include the correct programs used in this report. The following steps ensure that the profiling tools and other software are included.

After step 4 of the instructions from the yocto-manifest do the following: Edit the **yocto/build/-conf/local.conf** file to make sure the MACHINE variable is set to 'overo'. Also in that file locate the line that reads 'EXTRA_IMAGE_FEATURES' and add the dev-pkgs, tools-sdk and debug-tweaks to that line.

The meta-layers include several basic build targets, but a custom one is made to include the OpenCV software, Gstreamer and other tools. To do this create a new file called 'gumstix-camera-image.bb' in the **yocto/poky/meta-gumstix-extras/recipes-images/gumstix/** directory.

The contents of the file are as follows:

```
1 DESCRIPTION = "A Gumstix console image including Gstreamer and OpenCV for computer
      vision"
2
3 require gumstix-console-image.bb
4
5 IMAGE_INSTALL += " \
6   gstreamer1.0 \
7   gstreamer1.0-plugins-base \
8   gstreamer1.0-plugins-good \
9   opencv \
10  perf \
11 "
```

After this is done return to the main yocto folder and run the following commands:

```
1 source ./poky/oe-init-build-env
2 bitbake gumstix-camera-image
```

This process can take a few hours the first time it is run. When the process is finished the Gumstix image is available in the **yocto/build/tmp/deploy/images** folder.

To write the image to a micro-SD card follow steps 6 and 7 from the instructions. In step 6 an output path specifying where the image should be stored can be entered by appending '-o /your/path/' to the wic command. This might be necessary, because wic does not have the right permissions to create the default **/var/tmp/wic** directory.

A.2 Support for driver compilation

The image is now ready to be used with the USB webcam and Caspa camera, but to be able to communicate with the FPGA the GPMC driver has to be compiled and installed. The compilation is best done natively on the Gumstix. For this to work the kernel files need to be present. The following steps need to be done to accomplish this:

1. Boot the Gumstix with the previously created micro-SD card. Login with the root account, this has no password. Then change to the **/usr/src** folder and download the kernel archive to the Gumstix using the following command:

```
cd /usr/src
wget https://github.com/gumstix/linux/archive/yocto-v3.18.y.tar.gz
```

2. Unpack and rename the kernel archive using the following commands:

```
tar xvfz yocto-v3.18.y.tar.gz
mv linux-yocto-v3.18.y linux
```

3. Make a symbolic link from the linux folder to the **/lib/modules/** folder and prepare the kernel:

```
ln -s /usr/src/linux /lib/modules/3.18.18-custom/build
cd /lib/modules/3.18.18-custom/build
make mrproper
```

4. Copy the defconfig file from the yocto build on the host pc to the **/usr/src/linux** folder on the Gumstix. This can be done with for example scp. The file must be renamed to .config after copying. The defconfig file can be found in the **yocto/build/tmp/work/overo-poky-linux-gnueabi/linux-gumstix/3.18-r0/** folder.

   On the host pc run:

```
cd path/to/yocto/build/tmp/work/over-poky-linux-gnueabi/linux-gumstix/3.18-r0
    /
scp defconfig root@gumstixIP-address:/usr/src/linux/
```

   On the Gumstix run:

```
mv /usr/src/linux/defconfig /usr/src/linux/.config
```

5. Then prepare the kernel modules:

```
cd /lib/modules/3.18.18-custom/build
make modules_prepare
```

6. Finally copy the Modules.symvers to the Gumstix and run depmod. On the host pc run:

```
cd path/to/yocto/build/tmp/work/overo-poky-linux-gnueabi/linux-gumstix/3.18-r0/linux-overo-standard-build/
scp Modules.symvers root@gumstixIP-address:/usr/src/linux/
```

On the Gumstix run:

```
cd /lib/modules/3.18.18-custom/build
depmod -a
```

After completing these steps the Gumstix is ready to build kernel modules.

A.3 GPMC kernel driver

To make use of the GPMC connection to the FPGA a kernel driver is needed that handles the low level communication. The source files for this driver can be found in the RaMstix documentation (Schwirtz, 2014). A few changes have to be made to make sure the right includes are used. To simplify the process the altered driver code can be found in the source files accompanying this report.

To compile the kernel module copy the folder **kernel_module** from the source files to the Gumstix by using for example the scp-command again.

On the Gumstix change directories to the **kernel_module** folder and build it:

```
cd kernel_module
make
```

This will result in the file 'gpmc_fpga.ko' being generated in the **kernel_module/src/** directory. This is the compiled kernel driver to communicate with the GPMC.

To use the driver simply add it to the running system with the following command:

```
insmod gpmc_fpga.ko
```

This will insert the kernel module and add the device **/dev/gpmc_fpga** to the running system.

The process for initializing the FPGA with the correct setup for communicating is described in appendix B.

B Setup for FPGA camera

This appendix explains how to connect the Omnivision OV7670 camera to the RaMstix board and load the correct program to retrieve images using this camera.

B.1 Physical setup

The physical setup requires connecting the camera to the I/O pins of the FPGA located on the RaMstix. These pins are highlighted by the purple box in figure 2.3. The pin numbering for the RaMstix can be found in the documentation (Schwirtz, 2014).

The connections that have to be made are listed in table B.1. If only one camera is to be used, connect camera A. For the stereo-vision setup both camera A & B have to be connected.

For the I^2C connection to work properly two pull-up resistors have to be connected to the SIOC and SIOD lines. These resistors have a value of $1.5k\Omega$.

OV7670	RaMstix camera A	RaMstix camera B
SIOC (out)	F_OUT1	F_OUT9
SIOC (in)	ENC3B	ENC4B
SIOD (out)	F_OUT0	F_OUT8
SIOD (in)	ENC3A	ENC4A
VSYNC	ENC1A	ENC2A
HREF	ENC4I	ENC2B
PCLK	ENC1I	ENC2I
XCLK	F_OUT2	F_OUT10
D7	F_IN7	F_IN15
D6	F_IN6	F_IN14
D5	F_IN5	F_IN13
D4	F_IN4	F_IN12
D3	F_IN3	F_IN11
D2	F_IN2	F_IN10
D1	F_IN1	F_IN9
D0	F_IN0	F_IN8

Table B.1: Camera to RaMstix connections

Besides the cameras there is also an external button connected to the RaMstix to start the camera initialization and an LED that shows the initialization status. When using the setup provided in this appendix the button should be connected to ENC3I and the LED is already on the RaMstix board attached to the INIT_DONE pin.

B.2 VHDL

Two designs have been implemented for the FPGA, one using a single camera and one using stereo cameras. The VHDL implementations can be found in the source files accompanying this report.

To use the implementations an installation of Quartus II is required. The projects are created using Quartus II version 13.1, because this is the latest version that still supports the Altera Cyclone III FPGA that is present on the RaMstix.

The source files contain the Quartus project for a mono-vision implementation and a project for the stereo-vision implementation.

To open a project in Quartus go to **File -> Open Project...** and then move to the directory where the project is located and select the *camera_reader.qpf* file.

The project consists of multiple files for the various parts of the design. Open the *camera_reader_top.vhd* file, which is the Top-Level entity.

To compile the VHDL go to **Processing -> Start compilation** and wait for the compilation to finish.

To program the FPGA on the RaMstix connect an Altera USB Blaster to the host computer and to the programming header on the RaMstix. Programming is done by selecting **Tools -> Programmer** this opens a pop-up window for the programmer where the file to be written and the device to which it should be written can be selected. The compiled version of the program should be ready and by clicking the **Start** button the programming will commence.

When the Start button is not highlighted click the **Add File...** button to manually select the compiled file. It is located in the *output_files* folder and is called *camera_reader_top.sof*. When this file is opened click the **Start** button to commence programming.

The red LED located near the FPGA on the RaMstix should turn on to indicate the FPGA is programmed, but the setup of the cameras is not yet triggered.

Press the attached button for at least one second to start the initialization process that sends the correct setup values to the connected OV7670 camera(s). When this is done the FPGA and camera(s) are ready to stream data to the Gumstix over the GPMC bus.

When the kernel module is inserted in the running Linux system as is described in appendix A.3 and the FPGA on the RaMstix is programmed as described above the Camera class can be used to retrieve images for further processing.

Bibliography

Altera Corporation (2013), Quartus II version 13.1.
https://dl.altera.com/13.1/?edition=web

ARM Ltd. (2011), NEON.
http://www.arm.com/products/processors/technologies/neon.php

Bradski, G. (2000), OpenCV, *Dr. Dobb's Journal of Software Tools*.

CompuLab (2015), ISP data path.
http://www.compulab.co.il/workspace/mediawiki/index.php5/Image:Isp_data_path.jpg

Griffiths, N. (2003), Nigel's performance Monitor for Linux.
http://www.ibm.com/developerworks/aix/library/au-analyze_aix/

GStreamer (2012), GStreamer open source multimedia framework.
https://gstreamer.freedesktop.org/

Gumstix Inc. (2015), Overo Firestorm-P COM.
https://store.gumstix.com/coms/overo-coms/overo-firestorm-p-com.html

Gumstix Inc. (2016), Caspa FS.
https://store.gumstix.com/caspa-fs.html

Lammertink, T. (2003), Joystick controller for JIWY.
http://doc.utwente.nl/56883/

Larson, S. (2015), I2C Master (VHDL).
https://eewiki.net/pages/viewpage.action?pageId=10125324

LinuxTV (2015), V4l-utils.
https://linuxtv.org/wiki/index.php/V4l-utils

Logitech (2009), Webcam C250.
http://support.logitech.com/en_ca/product/webcam-c250

MIPI alliance (2014), MIPI Camera Interface Specifications.
http://mipi.org/specifications/camera-interface#CSI2

OmniVision (2006), OV7670/OV7171 CMOS VGA (640x480) CameraChip with OmniPixel Technology.
http://www.eleparts.co.kr/data/design/product_file/Board/OV7670_CMOS.pdf

ON Semiconductor (2015), MT9V032: VGA 1/3" GS CMOS Image Sensor.
http://www.onsemi.com/PowerSolutions/product.do?id=MT9V032

Perf Wiki Kernel.org community (2015), Perf: Linux profiling with performance counters.
https://perf.wiki.kernel.org/index.php/Main_Page

Schwirtz, M. (2014), RaMstix FPGA board.
https://www.ram.ewi.utwente.nl/ECSSoftware/RaMstix/docs/index.html

terasIC (2012), Altera DE0 Nano.
http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=593

Texas Instruments (2010), DM3730 Digital Media Processor.
http://www.ti.com/product/dm3730

Texas Instruments (2011), DM3730/25 DaVinci Block Diagram.
http://www.ti.com/general/docs/datasheetdiagram.tsp?genericPartNumber=DM3730&diagramId=SPRS685D

Texas Instruments (2012), AM/DM37x Multimedia Device, Technical Reference Manual.
http://www.ti.com/lit/pdf/sprugn4

Yocto Project (2010), Yocto Project.
https://www.yoctoproject.org