Embedded Systems Lab

Victor van Wijhe (s2426145) (EmSys),
Joppe Blondel (s1681850) (EmSys)

University of Twente, Group 4, June 25, 2021

Contents

1 Introduction

This report describes the development and design space exploration of a cyber physical system which is designed for the Embedded systems lab course. The hardware was provided by the course but choices needed to be made which parts where used in the final system.

2 Demonstrator specifications

For this project the following material is available:

- RamStix board with Cyclone III
- Terasic DE0 with Cyclone IV
- Gumstix Overo fire
- Motor controller board (H-Bridge)
- Camera mount with pan and tilt motor, rotary encoders and a USB webcam

With these materials the aforementioned goal must be achieved. The system must steer a rotating camera head and use the captured video stream to control the direction it is moving. The chosen application will let the camera head follow an aruco marker in such a way the marker is in the center of the video (see section Image processing). This means that the setup should collect video frames from the USB webcam, find the aruco marker, calculates the amount the camera head should move, let the controller move the camera head and repeat. The system should collect frames at 25/30 FPS at a resolution big enough to recognize aruco markers.

The system can be divided in the following subsystems:

- **Video acquisition**: This subsystem will collect video frames from the USB webcam.
- **Image processing**: This susbsystem will try to recognize aruco markers and calculate the movements which need to be made to center the aruco marker.
- **Communication between Gumstix Overo fire and FPGA**: can be done with GPMC or UART.
- **Motor controllers**: PID controllers controlling the pan and tilt motors.
- **Encoder**: This subsystem reads signals in order to determine the rotation of the motors.
- **PWM generator**: The subsystem that generates a PWM to controll the motors.

These subsystems will be described in detail in the subsections below.

3 Video acquisition

The project setup contained a Logitech webcam. Gstreamer was used to capture the frames of the webcam which had a YUY2 format. The code was first tested on a separate laptop with Linux and was able to capture and transmit the frames using Appsink. When the same software was implemented on the Gumstix the code resulted in two empty frames. After three lab sessions debugging the problem, the conclusion was that the Gstreamer version (last updated in 2014) was outdated. Updating the software was not possible due to the Gumstix not containing any package manager or maven. Switching to other another camera which could use MJPEG as output solved the 'two emtpy frames' problem but resulted in seemingly random segmentation faults within the library code. The final solution to this problem was to use OpenCV to capture the frames of the webcam.

4 Image processing

The image processing is done using the OpenCV library, it contains a function to get the coordinates from a Aruco marker (such a marker can be seen in Figure 1). During development it was discovered that the OpenCv library on the Gumstix was missing the Aruco module. The module could have been imported and compiled on the Gumstix, but due to time constrains the choice was made to detect a single color instead. The red pixels of the captured image are changed to white and the rest to black. The center coordinates of the white pixels could be calculated and sent to the PID controller. Calculating the coordinates of the white pixels is not finished due to time constrains.

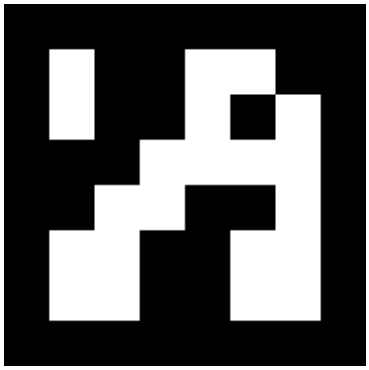

Figure 1: an example of an Aruco marker

5 Communication between FPGA and ARM core (GPMC vs. UART)

Communication between the used ARM core and the FPGA (the programmable logic itself on the RamStix board, the NIOS II core on the DE0) can be done over a dedicated high speed memory access bus (GPMC) directly connected to the ARM core for the RamStix or over UART connected to the NIOS II softcore on the DE0. For UART the maximum transfer speed is directly connected to the used baudrate: when using a baudrate of 115200 symbols per second the maximum transfer rate would be 14.06KiB. A practical maximum transfer speed for the GPMC is measured by writing from the ARM core repeatedly to the GPMC bus for a large number of times and measuring the time it takes. It is found that in this fashion the transfer speed is 512KiB.

When testing the UART the level-shifters between the Overo and the DE0 had to replaced repeatedly taking up more time than necessary.

6 Encoder

The project uses encoders to detect the rotation of the motors from which the motors position can be determined. The output of the rotary encoder (while being rotated) can be seen in Figure 2 it consist of two identical signals with an offset. This offset allows the interpreter to determine if the motor changes direction. The encoder could be designed to check a single edge, and up to four edges. Checking four edges gives the highest resolution and thus accuracy, therefore it was decided to take this approach.

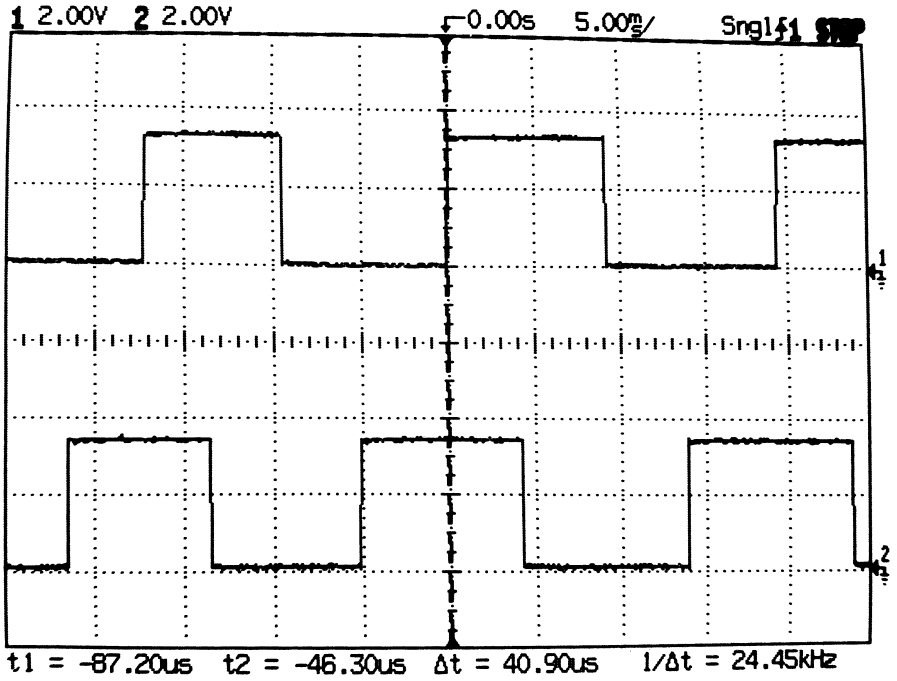

Figure 2: Encoder output

After developing the code on the FPGA it was simulated in Modelsim, a two edge variant can be seen in Figure 3. Everything worked in simulation but when it was implemented on hardware the results where not consistent. Later on in the project it was determined that the level shifters where defective which could explain the undefined behaviour. Implementing an interpreter that checks a single edge of the encoder did work on hardware, this setup was used in the project. The interpreter increases or decreases a counter based on the direction of the motor.

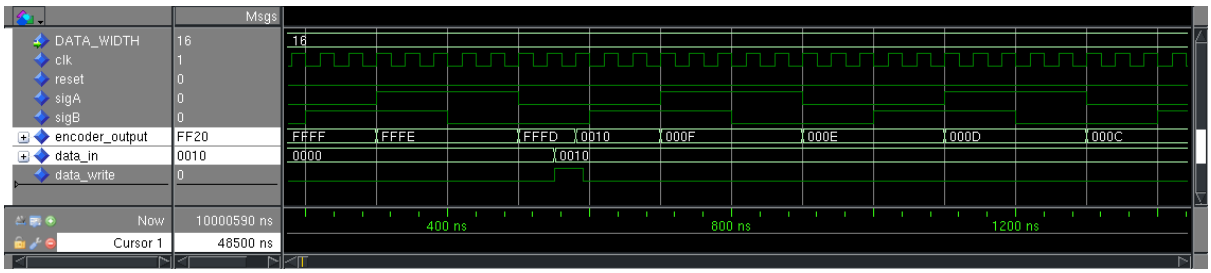

Figure 3: Encoder output in simulation

It was determined that with a one edge interpreter, the encoder generates roughly 1000 edges per rotation or one edge per 0.36°(when assuming 1000 ticks per rotation). The counter of the interpreter is the output which can be transferred using the Avalon bus(32 bit) or GPMC(16 bit) both options have a higher resolution than the possible counter size.

7 PWM generator

The PWM generator, which is used to drive the motors, is written in VHDL and generates a PWM signal of around 25 KHz, which is above the hearable audio range of 20 KHz.

Figure 5 and Figure 6 show the generated PWM signals for a duty cycles of 50% and 25% respectively. As can be seen the generated signal has a frequency of 24.414 kHz. The generator itself counts up and wraps back to zero when reaching its maximum. The output is set to high when the counter value is below the set duty cycle (which is expressed as a value between zero and the maximum counter value). The frequency at which the counter increments should be $2^n \cdot f_{PWM}$ where n is the amount of bits used for the duty cycle and counter. The final implementation uses an n of 8 since a duty cycle resolution of 256 seems wide enough for the use case. This results in a frequency for the generator of $2^8 \cdot 25kHz = 6.4MHz$. To generate this frequency, a simple frequency divider is used to convert 50MHz to 6.4MHz. Due to the working of this frequency divider (it counts up to the ratio between the incoming clock signal and the desired clock signal) this frequency will be rounded up to the first integer division: At a clock frequency of 50MHz the ratio will be $50MHz/6.4MHz = 7.8125$, which means the counter will count to 8 for a full period, resulting in a frequency of 24.414kHz (which is the same frequency found in the simulation results). The used clock frequency can be viewed in Figure 4.

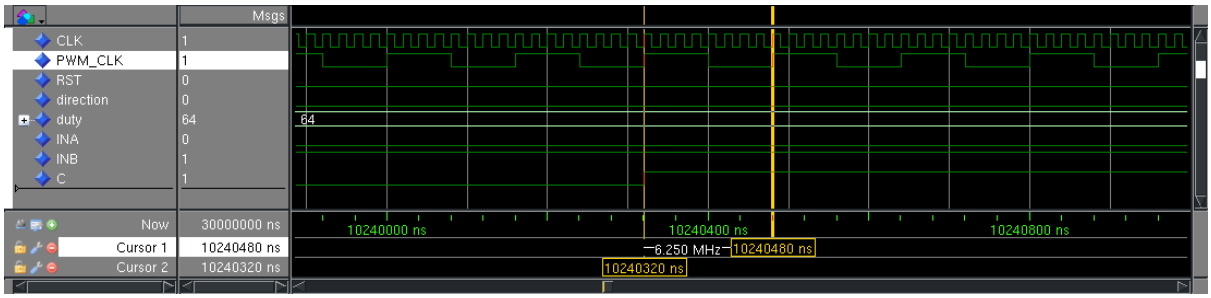

Figure 4: Clock used by PWM generator

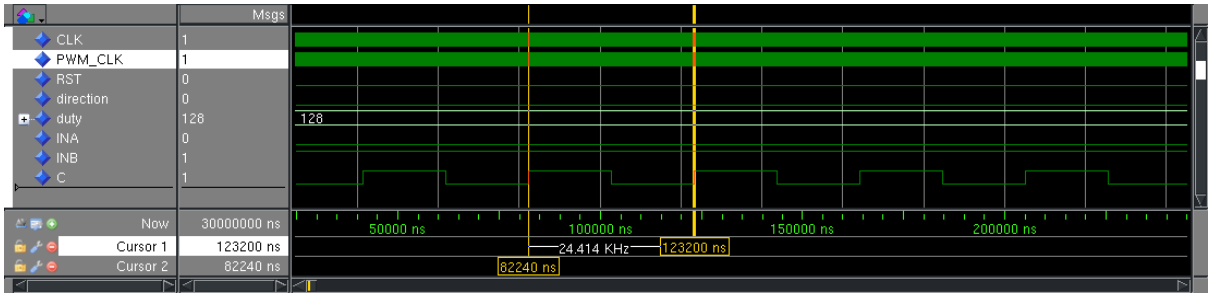

Figure 5: Generated PWM signal with duty cycle of 50%

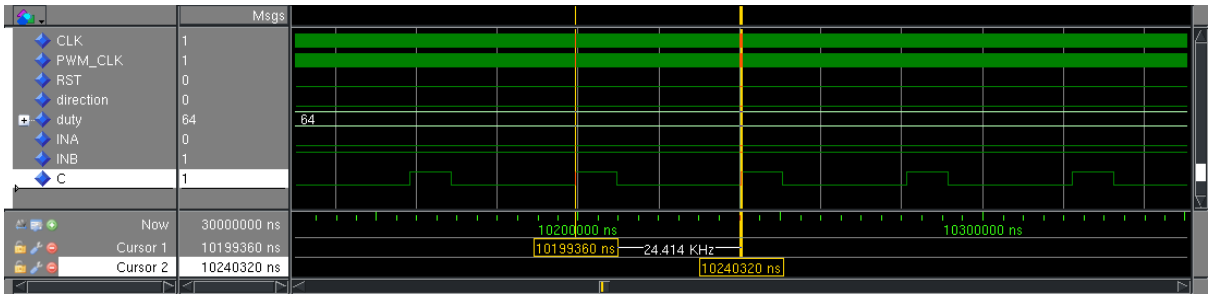

Figure 6: Generated PWM signal with duty cycle of 25%

Figure 7 shows the PWM signal generated on a FPGA. The signal is generated with a duty cycle of 12.5%, a value which the measurement tools on the oscilloscope validates. Furthermore, the frequency measured by the oscilloscope is 24.45kHz (time difference between the two visible cursors is 40.90us). Here

a small discrepancy can be seen of 1.4%. A possible explanation for this discrepancy is the fact that the frequency is measured with the oscilloscope (which measures the cursor positions in 3 significant figures, meaning that on a scale of tens of microseconds, the precision is less than a tenth of a microsecond) and it is done for one period only.

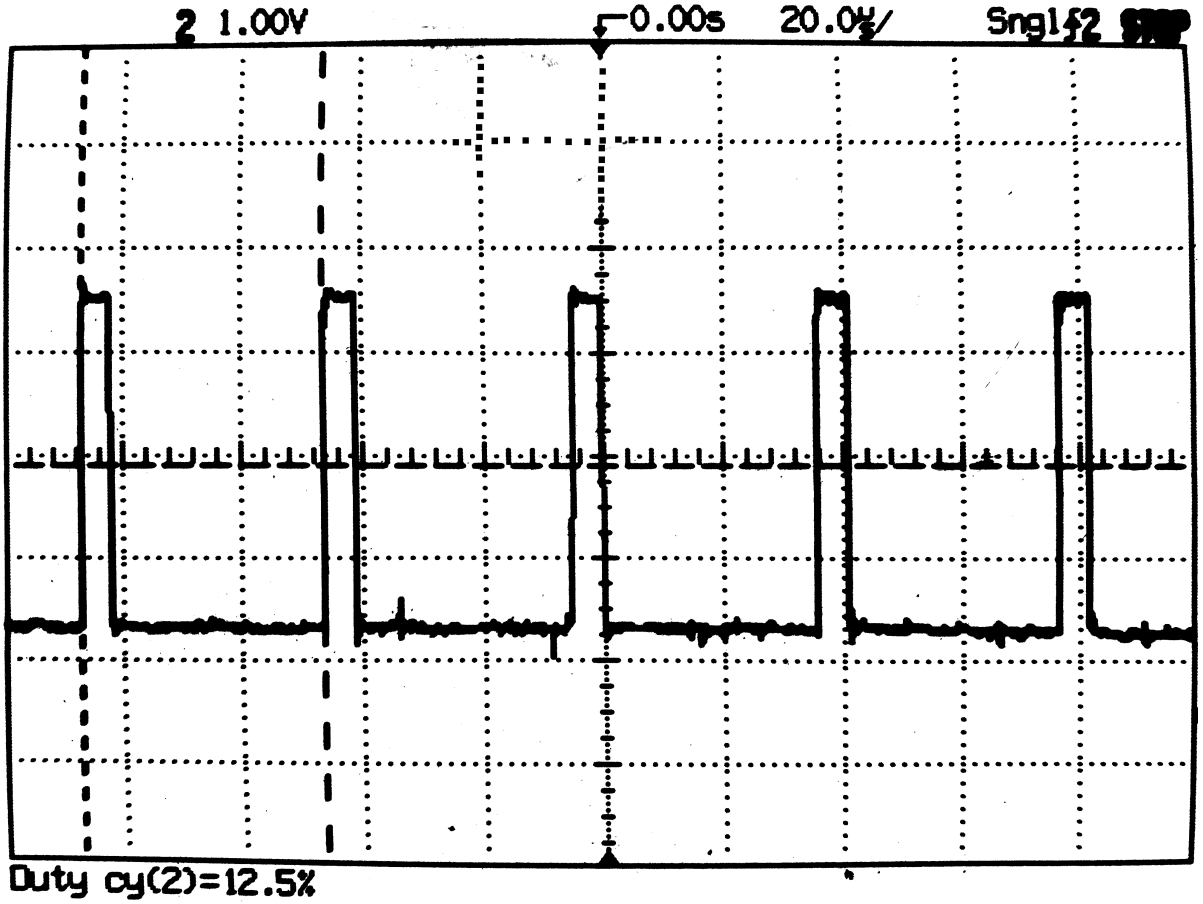

Figure 7: Output of PWM generator

The PWM generator can be interfaced with the controller in different ways. When implemented on the RamStix, the PWM generator can be connected directly to the GPMC interface, using the lower 8 bits for the duty cycle and a higher bit for the direction. The generator can be wrapped up in an Avalon slave an be connected to the Avalon bus of a NIOS II core.
While working on the PWM generator it was found that multiple level-shifters, which were used to interface the FPGA boards and the motor controller boards, failed.

8 Controller

The control of the motors can be done by using a PID controller (one alike Figure 8). The controller can be implemented in different locations: in hardware on an FPGA, in software on a softcore CPU on an FPGA or on the ARM core. Furthermore, there is a choice to implement an exported control loop from 20-sim or implement an own PID control loop. The location of the controllers (one for pan, and one for tilt) directly influences the needed data rate on the communication channel between the FPGA and the ARM core: when the control loop is implemented on the ARM core, the encoder values must be read and the PWM values must written each time step, meaning that at least four values must be transferred over the channel each timestep. When the control loop is placed on the NIOS II core or even on hardware, the only data which should be transferred are the setpoints for the controllers (meaning two values every time the camera should move). Due to time constrains the exported controller from 20-sim is only tested visually and seemed stable. Moving the camera head back and forth had shown an acceptable repeatablility by visually inspecting the setup while running.

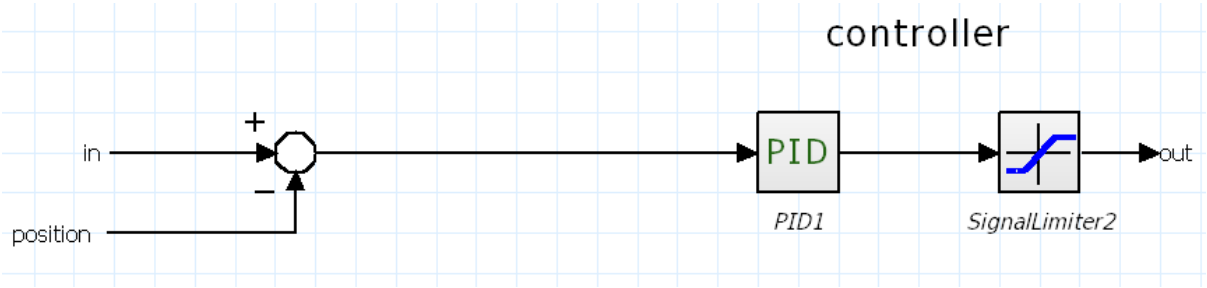

Figure 8: Controller

9 Design space exploration

This chapter describes the design space exploration which has been done during this project. Each module is assessed by itself so at the the end, the best setup can be determined. Due to time constrains the development time became the most critical factor followed by the complexity of the design. To calculate the score, each individual factor is multiplied with its weighting factor. The factor is divided as followed:

- ++ = 2
- + = 1
- +/- = 0
- - = -1
- - - = -2

The webcam is connected with USB, the Gumstix already contains a dedicated USB host controller which result in an evident preference compared with the NIOS and FPGA this can be seen in Table 1.

Image capture	Dev. time	Complexity	Performance	Flexibility	Score
Weighting factor	3	1	2	1	
Gumstix	++	++	+/-	+	9
NIOS	-	-	+/-	- -	-6
FPGA	- -	- -	+/-	- -	-10

Table 1: DSE table for the Image capture

Table 2 shows the possible choices and scores for the image processing. This can be done on the ARM core on the Gumstix, the NIOS softcore and on hardware on the FPGA. Again, development time is not taken into account lightly, as well as the performance. Looking at development time, the Gumstix would get the highest score since it is able to use OpenCV directly. Implementing it on the NIOS softcore would mean that OpenCV should be ported or the usage of libraries must be completely disregarded. The same holds for implementations directly on programmable logic. The image processing algorithms must be completely implemented in VHDL without the use of libraries which massively increases the development time. Looking at the complexity, due to not having the possibility of using image processing libraries on the FPGA, the score for being implemented on the FPGA is very low. The possibility of porting OpenCV funcions (or even the the whole library) to the NIOS softcore gives it a better score than the FPGA, but it still is not as convenient as using the OpenCV library directly as intended. Even tough the implementation on the Gumstix would be the least complex, the application would be scheduled with other applications running on the OS (such as important daemons), so in comparison with lower level implementations, the performance would be lower. At last, the possibility to directly use OpenCV makes it very flexible since changing, adding or removing functionality would be just changing some C code, while changing or adding functionality on the NIOS could result in writing new image processing functionality.

Image processing	Dev. time	Complexity	Performance	Flexibility	Score
Weighting factor	3	1	2	1	
Gumstix	++	++	-	++	8
Nios	-	+	+	+	2
FPGA	- -	- -	++	- -	-8

Table 2: DSE table for the image processing

Table 3 shows the options where the software of the PWM and encoder can be run on. The Gumstix and NIOS do not have preconfigured GPIO configurations while the FGPA is provided with this. Implementing an GPIO interface for the NIOS and Gumstix can be done but it would result in a significant increase in development time. Running the code on the Gumstix or NIOS would cost a lot of processing power because the processing cores would continuously check if something needs to be done while the FPGA only executes its code on a clock edge. An additional feature that the FPGA comes with is the simulation tool Modelsim, this provides a better insight in program stability and edge case. Because the Gumstix runs linux this could introduce a jitter which in turn makes the PWM and Encoder inconsistent. The NIOS could also contain a jitter, but because it runs a single program this is significantly less. Because the logic in the FPGA is controlled by a consistent 50 MHz clock it does hardly contain any jitter.

Encoder and PWM	Dev. time	Complexity	Performance	Flexibility	Simulation	Jitter	Score
Weighting factor	3	1	2	1	2	1	
Gumstix	-	+	-	+	-	-	-6
NIOS	+	+	-	+	-	+	2
FPGA	++	++	++	++	++	++	20

Table 3: DSE table for the encoder and pwm

In Table 4 one can see the possible options for the PID controller implementation. Once more the development time has the highest priority. Since for both the Gumstix and the NIOS softcore applications are written in C, no difference between development time is expected. In comparison, an implementation on the FPGA will probably encompass much debugging. Next to that, an implementation on the FPGA would mean that hardware multiplication (integer or floating point, whichever is required) must be included. The same can be said for complexity. An implementation on an FPGA in contrary could be very beneficial in terms of performance since, with a good hardware design, a whole timestep could take just a few clock cycles, together with the fact that the task of the programmed logic will only be used for the control loop, no jitter due to scheduling or possible CPU related timing inconsistencies (such as cache or pipeline behaviour) would be introduced. At last, since laying out everything in hardware could mean a complete restructuring of the module when chanced have to be made, implementations in C, thus being on the Gumstix or NIOS softcore, would be more flexible.

PID	Dev. time	Complexity	Performance	Flexibility	Jitter	Score
Weighting factor	3	1	2	1	1	
Gumstix	+	+	+	++	-	7
Nios	+	+	+	++	+	9
FPGA	- -	-	++	-	++	-2

Table 4: DSE table for the PID

Looking at Table 5 one could see the different advantages and disadvantages of using different data types within the control loop. One could use only floating points, only integer or fixed point values and a mix of both. It must be said that when using a controller exported from 20-sim it is necessary to use at least floating point values. When the controller can be implemented in C, the usage of floating point or fixed point arithmetic does not result in any change in development time (but when being implemented on the FPGA floating point multiplication would have to be implemented in VHDL, something which can take up much time and could be prone to errors). When using a mix conversions have to be made between both domains and can make the subsystem slightly more complex. Specifically looking at precision, the usage of floating point values does score higher than the others. This is due to the fact that the error values calculated in a control loop would be very small. Fixed point numbers will require a great number of bits to achieve the same precision as floating point numbers.

Floating p. vs integer	Dev. time	Complexity	Performance	Flexibility	Precision	Score
Weighting factor	3	1	2	1	1	
all floating p.	- -	- -	+	+/-	++	-4
all integer	-	+/-	+	+/-	+	0
mix	+	+	-	+/-	-	1

Table 5: DSE table for floating point vs integer

Table 6 shows the design space exploration of the GPMC against an UART. The GPMC implementation in this project is dependent on given GPMC framework both in hardware and software. The UART however works with writing and reading a file on both the Gumstix and NIOS. When the UART and GPMC are compared, the UART has a slightly better development time rating. The complexity of the UART and GPMC are comparable. Performance wise the GPMC has an edge on the UART but both are more than viable enough to transmit the messages for this assignment. Because the UART works with simply writing to a file while the GPMC uses a given framework, the UART has a better flexibility. The GPMC and UART have a direct consequence which setup is used (Gumstix + Ramstix or Gumstix + DE0), but because thet scores are quite similar this has not a significant impact on the end result.

GPMC vs UART	Dev. time	Complexity	Performance	Flexibility	Score
Weighting factor	3	1	2	1	
GPMC	+	+	++	-	7
UART	++	+	+	+	10

Table 6: DSE table for GPMC vs UART

When all the individual scores are combined the most optimal setup can be determined. In Table 7 four example setups are given, setup 1 has the highest score and for this reason is chosen to be developed. Setup 2 has a score similar to setup 1, but implementing the PID on the Gumstix is slightly worse. Setup 3 and setup 4 show the result if the entire project would be developed on the Gumstix or FPGA, this shows that the combination of the Gumstix and FPGA is much more viable.

Design space exploration

	Setup 1		Setup 2		Setup 3		Setup 4	
	Choice	Score	Choice	Score	Choice	Score	Choice	Score
Image capture	Gumstix	9	Gumstix	9	Gumstix	8	FPGA	-10
Image processing	Gumstix	8	Gumstix	8	Gumstix	-6	FPGA	-8
Encoder and PWM	FPGA	20	FPGA	20	Gumstix	9	FPGA	20
PID	Nios	9	Gumstix	7	Gumstix	7	FPGA	-2
Total score	(DE0)	46	(RamStix)	44	(Gumstix)	18	(DE0)	4

Table 7: DSE summation table

10 Conclusion and discussion

All together it is shown that the individual components behave correctly, but due to time constrains it was not possible to combine the video capture/processing and the motor controller.
An apparent disadvantage in a project such as this would be that, when comparing different possibilities and different combinations, the necessity could arise to have multiple measurements on the combinations to support a choice. This could mean, at least in the case this project, precious time is spent on trying out the different hardware choices while this could have been spent on working on the end result itself (assuming that the developers can make sane choices without needing to do in-depth research). But looking a little bit further, when no logical and sane choices could be made due to the lack of (pre)knowledge or it is necessary to develop something which is very cheap/small/power efficient, in-depth assessment of these variables in different configurations could be very helpful. Looking back at the available hardware some remarks could be made: Precious time was unfortunately wasted finding out why several combined modules did not work correctly. It is found that the level shifters between the different boards broke down time after time thus bringing in the necessity to check them every time two boards were connected together. This led to setups which did not completely work and items had to

be switched between different setups, which in its turn resulted in idle time while waiting for a specific thing to be available. Furthermore it should be noted that, as written before, the provided GStreamer library was severely outdated which could have been the reason for seemingly random failures. Since one of the assignments specifically stated that GStreamer's appsink had to be used, switching to the frame capture functionality within OpenCV was not yet possible.