ADVANCED EMBEDDED SYSTEMS

AES7 group 8 report

The development of a motion controller with the ZYBO Z7010

Fontys University of Applied Sciences
Department of Mechatronics

January 2020
Eindhoven

Authors

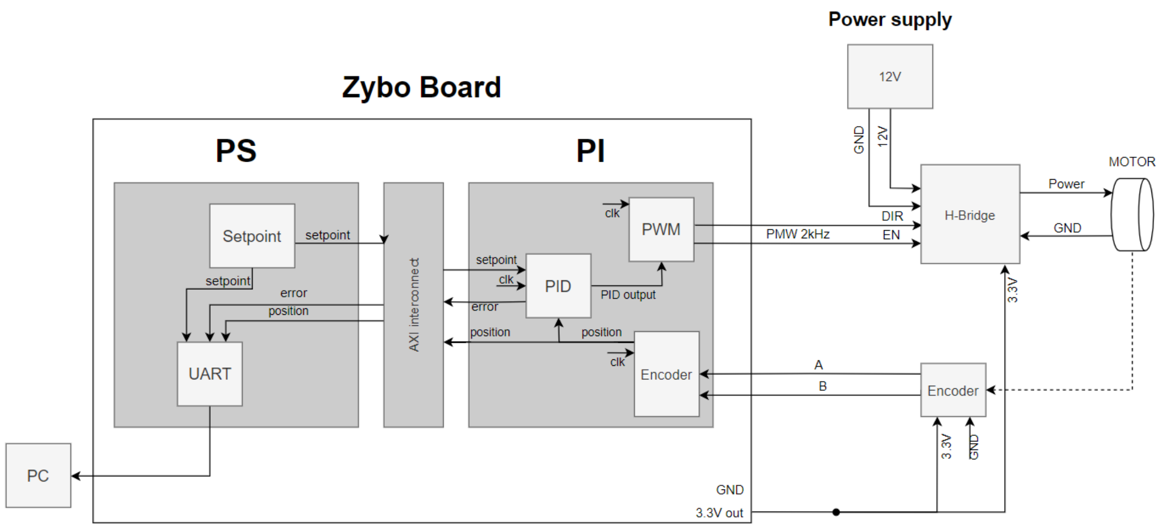
Roy Maas	2976382	r.maas@student.fontys.nl
Bart Hulsen	2949687	bart.hulsen@student.fontys.nl
Remco Kuijpers	3032663	remco.kuijpers@student.fontys.nl
Menno van der Steen	2991977	menno.vandersteen@student.fontys.nl

Summary

The Advanced Embedded Systems course provides an assignment that needs to be tackled in project form, with a duration of 7 weeks in total. The goal of the assignment is to implement a motion controller on the ZYBO Z7010 development board. The main research question is as follows.

"How do we design a motion controller on the ZYBO Z7010 to control the Maxon RE25 DC motor, powered through the PhodHB3 H-bridge and with readings from the HEDL 5540 encoder, given the requirements that it shall make one full rotation, settled within 15 ms, with a maximum position error of ± 0.05 revolutions, and a maximum steady state error of ± 0.0005 revolutions."

The hardware architecture is as shown below.



The ZYBO development board includes the Zynq-7010 System on Chip. It features a Processing System with a dual ARM Cortex-A9 processor and Programmable Logic, the FPGA. All motion controller components were programmed onto the FPGA, meaning that the Processing System runs bare metal and is only used to transfer position setpoints to the controller, and to read out values for verification and validation.

The motion controller incorporates the following Intellectual Property blocks.

Decoder – The decoder uses quadrature decoding to measure 2000 counts per turn in total, by using channel A and B with rising and falling edge detection. It produces a position signal of 16 bits, with a range of -4000 to 4000, which is sent to the PID controller.

PID controller – The PID controller calculates the error signal, by subtracting the position from the reference value (coming through AXI4 Lite from the Processing System) and performs the PID actions (32-bit signals). The outcome represents a power signal, that is sent to the PWM generator. The block also has a direction output to control the direction of the motor (with the H-bridge).

PWM generator – The PWM generator uses the outcome of the PID controller as the duty cycle for the 16-bit PWM signal, which powers the motor.

Matlab was used to create a model of the DC motor and the motion controller. Both the bode plot and transfer function were found, and attempts were made to let Matlab automatically tune the PID constants (based on the requirements as stated above). Unfortunately, these values did not result in stable responses when applied on the system, and therefore the constants were determined one by one, by trial and error.

For verification and validation, a connection to a laptop with UART has been used. As a result, the position values during a full rotation have been plotted with their timestamps. See below. Based on the position and time response plot, the following conclusions can be drawn. The performance requirements as stated above have not been entirely reached. The settling time is too large. The maximum position error was not calculated, because the time required to send a single setpoint over AXI was not measured (meaning that the reference profile could not be plotted). However, the system can perform a full and controlled rotation and it converges to exactly 2000 counts, meaning that the steady state error equals zero. In order to improve the performance of the system, the PID constants need to be finetuned. It is recommended to use a structured approach such as Model Based Design with PID autotune in Matlab.

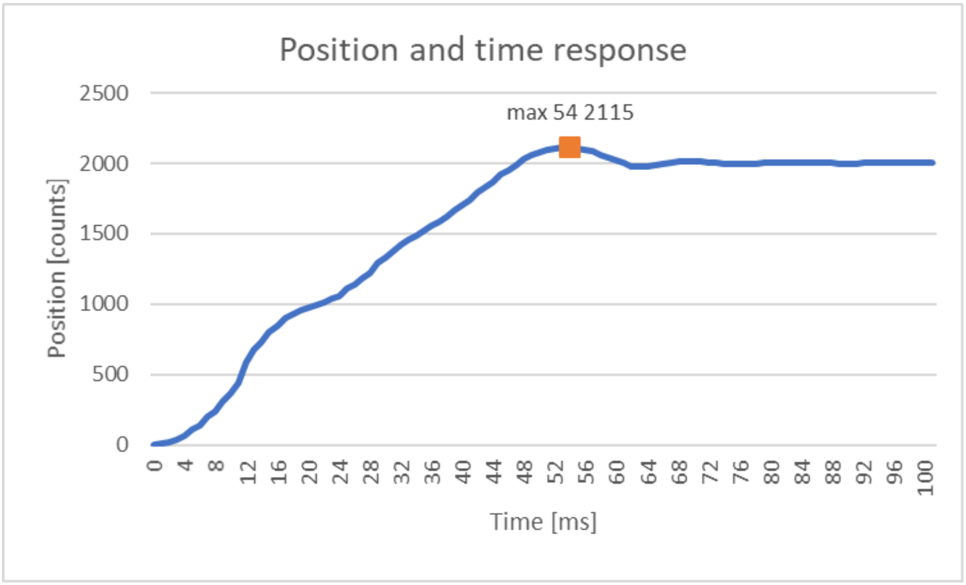


Table of Contents

Table of figures

Table of tables

1. Introduction

The Advanced Embedded Systems course is thought at the Fontys University of Applied Sciences in the 7th semester of the Mechatronics department. The objective of the course is to apply theoretical knowledge about FPGA and C programming in practice. Furthermore, the AES7 course gives additional insight into these topics to gain a better understanding of working with a System on Chip (SoC), such as the Xilinx Zynq chip.

The AES7 course provides an assignment that needs to be tackled in project form. The goal of the assignment is to develop a motion controller for the Maxon RE25 DC motor with the HEDL 5540 encoder using the ZYBO Z7010 board, which includes the Xilinx Zynq SoC. To power the actuator, the PhodHB3 H-bridge is used. The project has a duration of 6 weeks, with one additional week to finalize and present the results.

To tackle the assignment, research questions and requirements have been defined. The requirements below were provided by the managing teachers of the course.

The main research question is as follows. *"How do we design a motion controller on the ZYBO Z7010 to control the Maxon RE25 DC motor, powered through the PhodHB3 H-bridge and with readings from the HEDL 5540 encoder, given the requirements that it shall make one full rotation, settled within 15 ms, with a maximum position error of ± 0.05 revolutions, and a maximum steady state error of ± 0.0005 revolutions."*

Sub research questions were defined to breakdown the main question, these are: (1) *"What are the constraints of the hardware?"*, (2) *"How do we program, test and implement Intellectual Property (IP) blocks on the ZYBO Z7010 board?"*, and (3) *"What are the motion controller components and how are they integrated into ZYBO?"*.

The research questions are answered one by one in the upcoming chapters of the report. Chapter 2 covers the research part of the project, in which topics such as the hardware and the theory behind motion control are discussed. It also provides a section dedicated to the general solution idea (a final concept), of which the implementation is covered by Chapter 3. Chapter 3 includes the integration of hardware and software, and the test results. Finally, chapter 4 covers the general conclusion of the report.

2. Research

2.1. Hardware constraints

This section includes the results of research into the hardware constraints. Each of the components has its own subsection, in which the most important specifications, background information and characteristics are described. The aim is to have a thorough understanding of the hardware, in such a way that it can be used correctly and efficiently during development in order to satisfy the requirements. Subsection 2.1.5 covers the integration of hardware and considerations about additional apparatus that is required. The sub research question related to this paragraph is (1) *"What are the constraints of the hardware?"*. Section 2.1.1.1 answers the second sub research question: (2) *"How do we program, test and implement Intellectual Property (IP) blocks on the ZYBO Z7010 board?".*

2.1.1. ZYBO Z7010 with the Zynq SoC

The ZYBO Z7010 (Figure 1) is an embedded systems development board equipped with a Xilinx Zynq-7010 SoC, which incorporates a dual ARM Cortex-A9 processor (the Processing System (PS) side) and an FPGA (the Programmable Logic (PL) side).

Figure 1 The ZYBO Z7010 development board

Important features are that it has 512 MB DDR3 memory, 4400 logic slices, 240 KB of block RAM and an internal clock speed of 450 MHz Furthermore, there are 6 Pmod ports to interface with other components and it has on-board JTAG programming. UART to USB conversion is also available. Logically, the board operates at 3.3 V. It needs to be powered through USB at 5 V.

To share data between the PS and PL, the AXI bus can be used. The AXI bus has 3 types of operation, which are the following.

1. AXI4	Full performance bursting interconnection
2. AXI4 Lite	Non bursting interconnection (lower performance)
3. AXI4 Stream	Non-addressed packet based or raw interface

On a final note, the PS can run with or without (bare metal) operating systems, and needs to be programmed in C. The PL needs to be programmed with HDL code.

2.1.1.1. The development of IP blocks with ZYBO

Developing an IP block starts with configuring project sources. These source files contain HDL source files, constrains files, simulation files and block designs. When the project is configured correctly the block design can begin, first create a block design. After finishing the block design, the block must be simulated. A few simulations must be run to ensure the block functions as intended. These simulations consist of a behavioural simulation, which is intended to verify RTL (behavioural) code and to confirm that the block is functioning as intended. These simulations are typically done using test benches. The second simulation is the post synthesis simulation. Post synthesis simulation is used to verify that the functionality is correct after synthesis. Before the bitstream can be generated one last simulation is ran. To ensure everything works correctly a post implementation simulation is performed. If all functions a intended the ZYBO board can be programmed. This is done by generating a bitstream, this process translates the embedded system design to a bitstream file. This file is to be downloaded to the ZYBO board.

2.1.2. Maxon RE25 DC motor

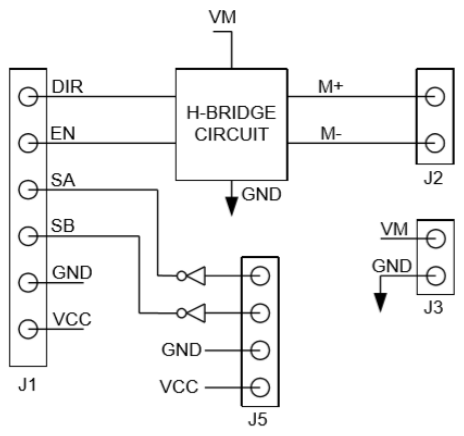
To make sure that other components are not damaged by the electrical specifications of the actuator (such as the stall current), it is important to take the following into consideration. The motor shall operate at no load characteristics.

Table 1 Important specifications of the actuator

Specification	Value
Nominal voltage	24 V
No load speed	5190 rpm
No load current	14.4 mA
Stall current	3.1 A

2.1.3. PhobHB3 H-bridge

To supply the DC motor with power, a H-bridge needs to be used. The PhobHB3 H-bridge can power actuators up to 12 V. It has a 6-pin Pmod port, with the following inputs and outputs.



Pin	Signal	Description
1	DIR	Direction pin
2	EN	Enable pin
3	SA	Sensor A feedback pin
4	SB	Sensor B feedback pin
5	GND	Power supply ground
6	VCC	Positive power supply (3.3/5V)

Figure 2 H-Bridge block diagram and pinout description table

Note that the DC motor is forced to operate at 12 Volts (instead of 24) due to the maximum specifications of the H-bridge.

2.1.4. HEDL 5540 Encoder

The HEDL 5540 encoder shall be used to measure the position of the actuator. It has three channels, A, B and Index, all of which have an inverse channel as well. The encoder reads 500 counts per turn (on both A and B) and has a nominal operating voltage of 5V.

2.1.5. Integration with additional components

At this point, the most important thing to consider is how to power the system in total. As this is a testing setup to prove that the requirements can be met (thus the size and type of apparatus is of low importance), the following decisions have been made based on ease of use and flexibility.

- The ZYBO board shall be powered through a USB connection to a laptop.
- A lab DC power supply shall be used to supply 12 V to the H-bridge.
- The encoder shall be powered with 3.3 V, instead of the nominal 5 V, to make sure that its output does not damage the Pmod interface of the ZYBO (which is rated for 3.3 V). This eliminates the need for a level shifter.

The hardware schematic in section 0 shows a clear image of the connections, both for power and for data.

2.2. Motion control

This part of the research chapter is dedicated to motion control. It starts with a subsection on the control theory, followed by a subsection on ways to integrate these theories in ZYBO. Finally, there is a separate subsection for Model Based Design. The aim of this paragraph is to understand how to use the hardware as described previously to develop a motion controller. It answers sub research question (3) "*What are the motion controller components and how are they integrated into ZYBO?*".

2.2.1. Control theory

A commonly used technique to design a motion controller is through PID control in a feedback loop. PID stands for Proportional-Integral-Derivative. The process is as follows. The output of the system needs to be measured and subtracted from the reference value, which yields an error signal. The error signal is then multiplied with a proportional constant Kp for the proportional control action. In parallel, the integral of the error signal is calculated and then multiplied with another constant, called Ki. To perform the derivative action, the derivative of the error signal is calculated and multiplied with a constant, in this case Kd. All the results are added up together and act as the input for the system. Figure 3 depicts a block diagram of the general PID controller in a feedback loop.

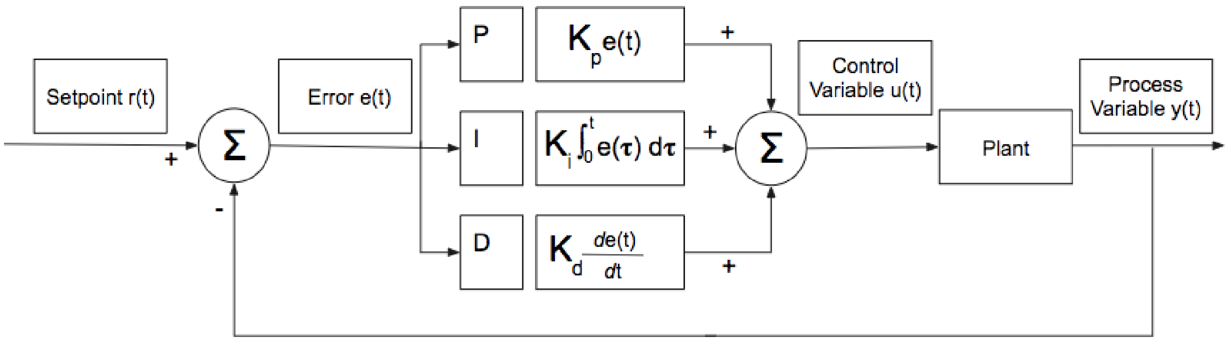


Figure 3 Block diagram of a PID controller in a feedback loop

The engineer can choose to use all control actions, or just the proportional action for instance, which would be called a P controller. Important to note is that a PID controller can make the system unstable if the constants are chosen incorrectly, meaning that the output does not converge to the setpoint value.

The main advantage of PID control is that the project team has used it before within HDL programming. An alternative approach would be to use Full State Feedback control. Based on a discussion, the conclusion has been drawn that using this method would impose unnecessary risks (with regards to achieving the requirements, because of complexity) and that these do not outweigh the potential rise in performance. The challenge with PID control is to find the optimal Kp, Ki and Kd values. Model Based Design might be the answer, which is discussed in section 2.2.3.

2.2.2. Ways to integrate in ZYBO

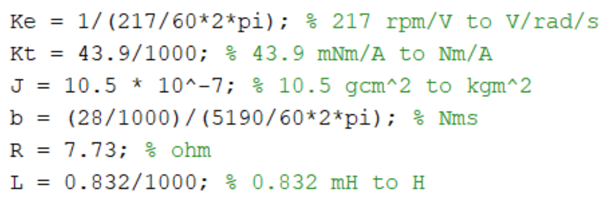
There are several ways to integrate a PID controller in ZYBO. As was discussed in section 2.1.1.1, Vivado is the main tool used to program the board. An alternative approach to directly programming IP blocks in Vivado is to use Matlab for Model Based Design, and afterwards let Matlab automatically generate the HDL code (which utilizes Vivado). If the model of the system is accurate, it leads to an optimized motion controller and reduces programming time. The main disadvantage of this approach is that there is no guarantee that a sufficiently accurate model can be found, and furthermore, generating code *behind-the-scene* would mean that error checking becomes less intuitive. Based on these advantages and disadvantages, the following decision has been made. The motion controller shall be integrated in ZYBO using Vivado, by manually programming IP blocks and writing C code. Matlab shall be used to design a model of the system and to tune the PID constants. If the model turns out to be inaccurate, the backup-plan is to iteratively find the constants one by one.

2.2.3. Model Based Design

To prepare for development and testing, a model of the system has been made to find optimal PID constants. In addition to section 2.1.2, the following parameters are relevant. See Table 2. An image of the values in Matlab with the conversion to standard units has been added as well.

Table 2 Additional DC motor specifications for Model Based Design

Specification		Value
Torque constant	Ke	43.9 mNm/A
Speed constant	Kt	217 rpm/V
Rotor inertia	J	10.5 gcm^2
Terminal resistance	R	7.73 Ω
Terminal inductance	L	0.832 mH



```
Ke = 1/(217/60*2*pi); % 217 rpm/V to V/rad/s
Kt = 43.9/1000; % 43.9 mNm/A to Nm/A
J = 10.5 * 10^-7; % 10.5 gcm^2 to kgm^2
b = (28/1000)/(5190/60*2*pi); % Nms
R = 7.73; % ohm
L = 0.832/1000; % 0.832 mH to H
```

The damping coefficient (see also above) had to be calculated manually with the formula below, for which the nominal values at no load as provided by the datasheet have been used. Note that the viscous friction is not the same at each speed and that the motor will run at 12 V, instead of the nominal 24 V, meaning that the value for *b* cannot be assumed to be as accurate as the rest of the parameters.

$$b = \frac{T}{\omega}$$

The model of the DC motor, based on the electrical and mechanical differential equations of a standard DC motor model, was built in Simulink, see Figure 4.

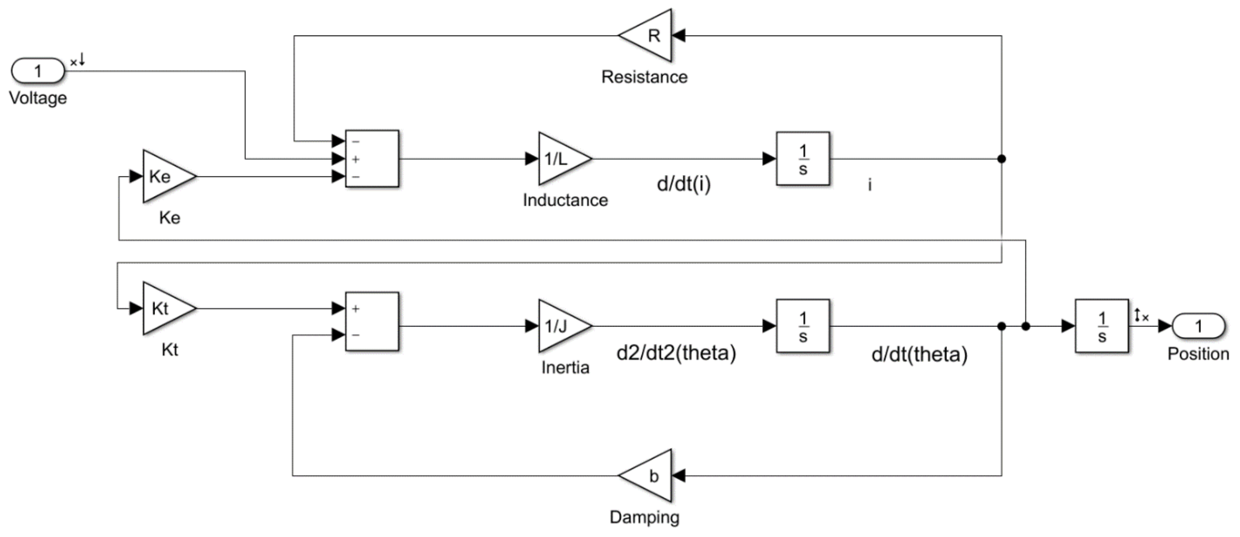


Figure 4 DC motor model in Simulink

The DC motor model is integrated into a feedback loop, as introduced in section 2.2.1. See Figure 5. The PID controller has its own block. Saturation has been added to stay within the limits of the H-bridge, and a step block acts as the reference signal. Both the error and the position signal are plotted with scope blocks.

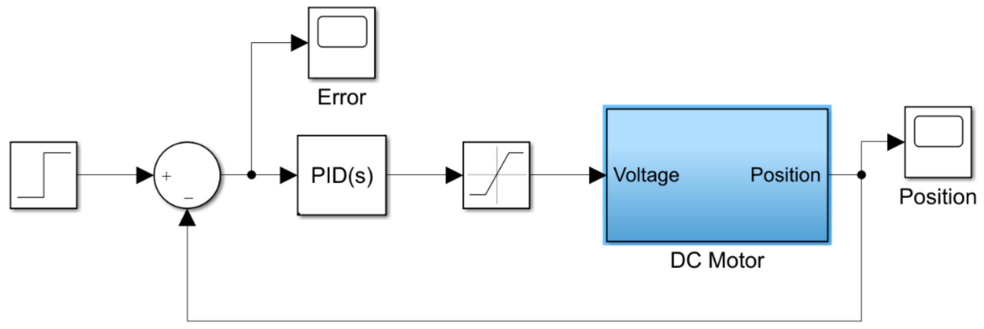


Figure 5 Model of the system with PID control

Linear analysis has been performed to find both the transfer function and the bode plot of the plant, which is the DC motor model. See below.

```
DC_motor_transferfunction =

           5.025e07
  ---------------------------
  s^3 + 9340 s^2 + 2.667e06 s

Continuous-time transfer function.
```

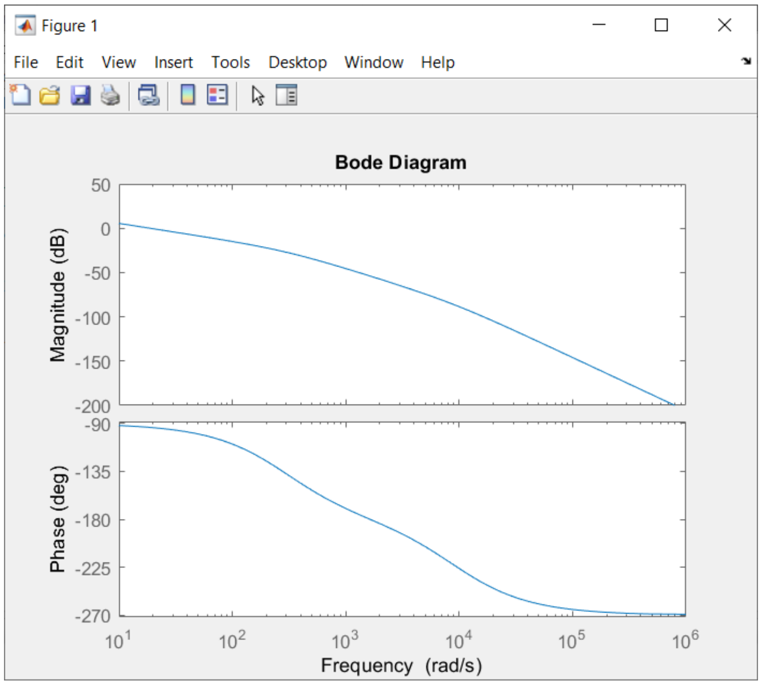


Figure 6 Bode plot of the DC motor model

At this point, all preparations to find the values for PID control have been made. The Simulink PID block has an autotune feature, which takes requirements for performance criteria such as dynamic overshoot and settling time, and outputs the PID constants to achieve them. As this is part of the development process, chapter 3 will further discuss the results of Model Based Design in detail.

2.3. Solution idea

Based on the results of research into hardware constraints and motion control, as discussed in sections 2.1 and 2.2, a solution idea to tackle the requirements has been defined. This paragraph covers the final hardware configuration and shows a hardware schematic in section 0. The IP blocks to build the motion controller are explained in section 2.3.2. Finally, section 2.3.3 includes the strategy to validate and verify the requirements. Together with chapter 3 on development, this section provides answers to the main research question:

2.3.1. Hardware schematic

The final concept for the hardware architecture has been defined according to the hardware constraints as shown in section 2.1.

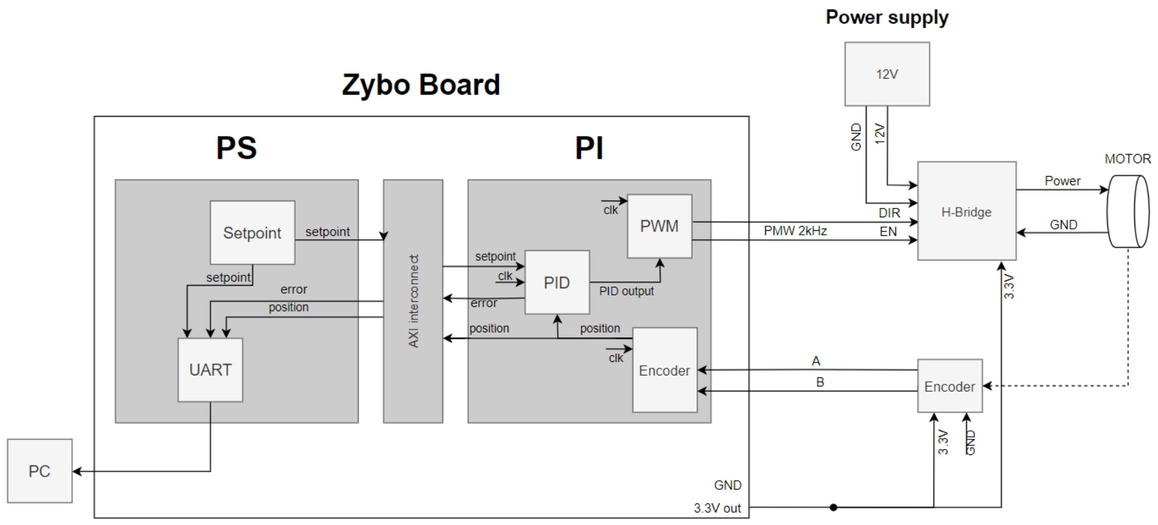


Figure 7 Hardware schematic

2.3.2. IP blocks

To develop the motion controller in Vivado, the following IP blocks shall be programmed. All of these are situated in the Programmable Logic. The Processing System shall run bare metal and will only be used to pass through the reference trajectory and to read out data for validation and verification (see section 2.3.3), over AXI4 Lite. The most important reason to have all IP blocks in PL, is due to the experience that the project team has with regards to programming HDL code.

Decoder

The Decoder block takes the encoder signals and translates them to a position signal. It shall use the A and B channel, and measures both rising and falling edges, meaning that a total of 2000 counts per turn can be measured. With 2000 counts per turn, a resolution of $\frac{360}{2000} = 0.18\, degrees$ can be measured, which is sufficient to achieve the requirements. To achieve this resolution a quadrature decoder must be used. This decoder changes from states when the A or B signal changes. With this transition the position is added by one count ore subtracted by one count, depending on the kind of transition.

PID controller

The PID controller calculates the power needed to control the output that it follows the input. The PID constants will be calculated by model-based design.

PID controller – The PID controller calculates the error signal, by subtracting the position from the reference value (coming through AXI4 Lite from the Processing System) and performs the PID actions. The outcome represents a power signal, that is sent to the PWM generator. The block also has a direction output to control the direction of the motor (with the H-bridge).

PWM generator – The PWM generator uses the outcome of the PID controller as the duty cycle for the 16-bit PWM signal, which powers the motor.

2.3.3. Validation and verification

To validate and verify the requirements, ZYBO's built-in UART functionality shall be used. The most important values that need to be sent to a laptop are the actual position, and the reference value at that point. Both can be plotted with timestamps (in Microsoft Excel for instance), to show if the system meets the error and speed criteria. With the AXI communication the demanded values can be presented trough UART.

3. Development

3.1. Decoder

Resource: https://www.digikey.com/eewiki/pages/viewpage.action?pageId=62259228

To get useful information out of the quadrature encoder output signals, the output signals need to be decoded. The quadrature encoder has two digital output signals, so called "A" and "B". When the motor is spinning in one direction the output signals of the quadrature encoder give pulses. The phase difference between the "A" channel and "B" channel is 90 degrees. Depending on what direction the motor is rotating "A" or "B" will lead, see Figure 8.

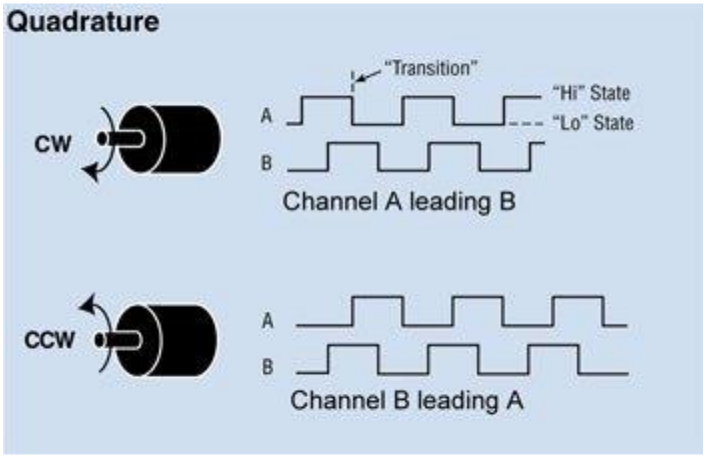


Figure 8 Quadrature encoder

To decode these signals to a position value of the motor shaft, the transitions between the two channels can be counted. In this case the encoder has 500 pulses per revolution, by counting the rising and falling edges of the signals it's possible to divide one rotation in 2000 counts.

Port description	**Port name**	**I/O**
System clock	Clk	Input
Encoder A signal	A	Input
Encode B signal	B	Input
Reset to origin	Set_origin_n	Input
Direction of movement	Direction	Output
Position	Pos{15:0}	Output

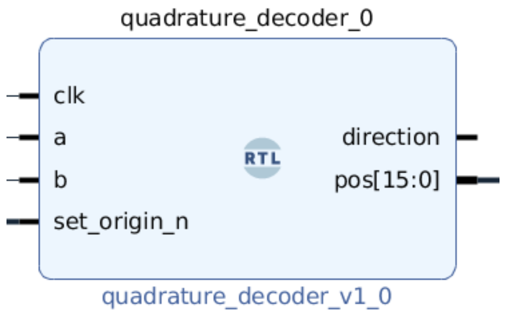


Figure 9 Decoder entity and decoder block

If the set_origin_n input gets pulled low the position gets reset to 0. Pos{15:0} represents the 16 bits position counter. In Figure 9 the decoder entity is described.

The flow of the quadrature decoder can be described as in Figure 10 and the behaviour in Table 4.

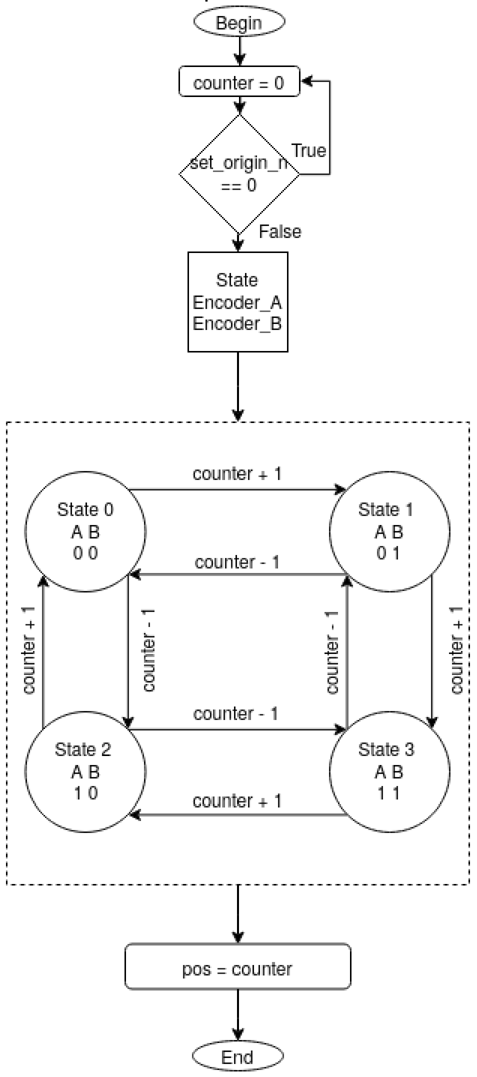


Figure 10 Flowchart quadrature decoder

	Previous inputs		New inputs		Results	
Set_origin_n	**A_prev**	**B_Prev**	**A_new**	**B_new**	**Direction**	**Position**
1	0	0	1	0	1	Decrement
1	1	0	1	1	1	Decrement
1	1	1	0	1	1	Decrement
1	0	1	0	0	1	Decrement
1	0	0	0	1	0	Increment
1	0	1	1	1	0	Increment
1	1	1	1	0	0	Increment
1	1	0	0	0	0	Increment
0	X	X	X	X	No change	0

Table 3 Behaviour quadrature decoder

With the demanded behaviour of the decoder, the function is validated with a simulation (see Figure 11). In the top of the simulation the A and B signal are produced by a forced clock. Between A and B is a phase shift of half a period. In this situation the encoder should count up. As shown in the figure the position counts up, so the decoder works.

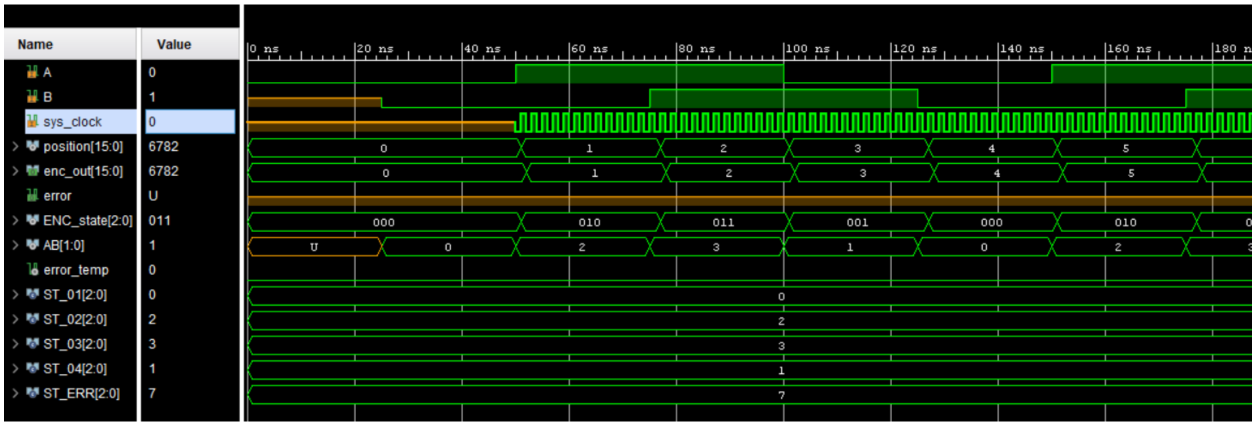


Figure 11 Simulation decoder

3.2. PID controller

Resource: https://www.instructables.com/id/PID-Controller-VHDL/ & https://github.com/deepc94/pid-fpga-vhdl

In order to perform according the control theory, an PID controller is implemented in the VHDL code. This controller acts like the controller described in the chapter 2.2.1. Based on the resource and the flowchart (see Figure 12) an PID controller is designed in VHDL code. To implement decimal number with accuracy, the constants are created by a numerator and denominator. In this way the constants are more accurate and floating-point values are used. In order to get the derivative and integral action, the old error is stored to create the difference in the error signal.

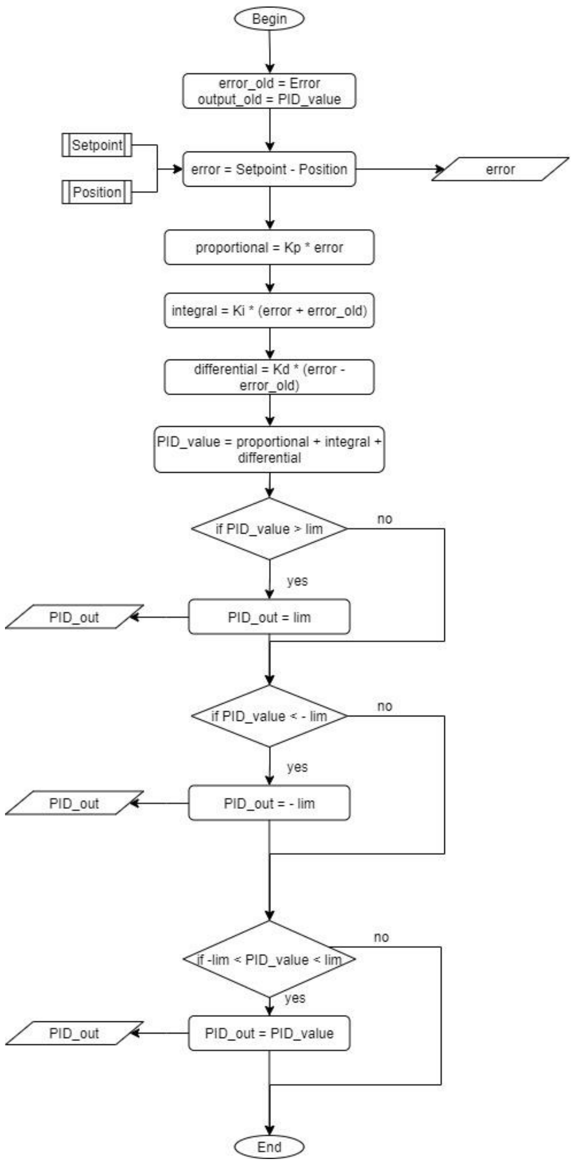

Begin
error_old = Error
output_old = PID_value
Setpoint
Position
error = Setpoint - Position
error
proportional = Kp * error
integral = Ki * (error + error_old)
differential = Kd * (error - error_old)
PID_value = proportional + integral + differential
if PID_value > lim
no
yes
PID_out
PID_out = lim
if PID_value < - lim
no
yes
PID_out
PID_out = - lim
if -lim < PID_value < lim
no
yes
PID_out
PID_out = PID_value
End

Figure 12 Flowchart PID controller

In **Error! Reference source not found.** the entity of the PID is described and the block design is shown.

Port description	Port name	I/O
Position	Adc_data	Input
Setpoint	Setval	Input
Output power	output	Output
Direction	Dir	Output
System clock	CLK	Input
Enable Kp	Kp_sw	Input
Enable Ki	Ki_sw	Input
Enable Kd	Kd_sw	Input
Kp numerator	Kp_num	Input
Kp denominator	Kp_den	Input
Ki numerator	Ki_num	Input
Ki denominator	Ki_den	Input
Kd numerator	Kd_num	Input
Kd denominator	Kd_den	Input
Enable or disable PID	On_off_swith	Input

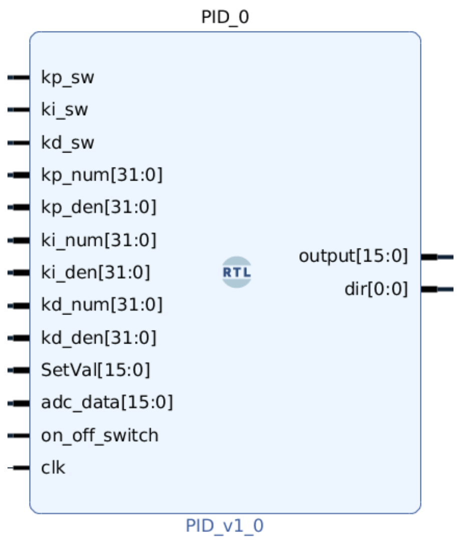
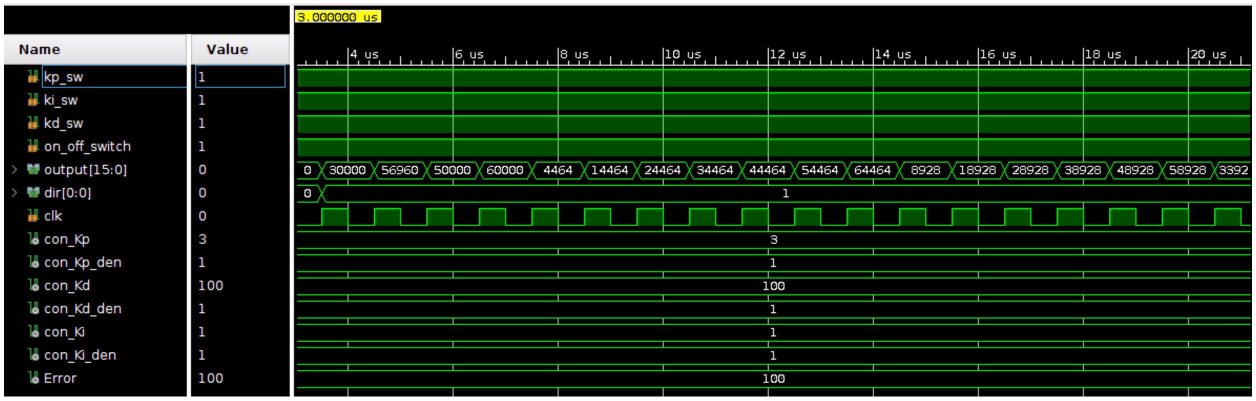


Figure 13 PID entity with PID block design

With the simulation of the PID controller, the performance of the controller is validated, see Figure 15. In this simulation a constant error value of 100 is implemented in the simulation. Because the integrator will add more power to the output when there is a constant error, the output increases. Hereby the PID controller functionates as demanded.



Name	Value
kp_sw	1
ki_sw	1
kd_sw	1
on_off_switch	1
output[15:0]	0
dir[0:0]	0
clk	0
con_Kp	3
con_Kp_den	1
con_Kd	100
con_Kd_den	1
con_Ki	1
con_Ki_den	1
Error	100

Figure 14 Simulation PID controller

3.3. Clock divider

Resource: https://www.codeproject.com/Tips/444385/Frequency-Divider-with-VHDL-2

For the PID block a different clock frequency is used, because the PID block can't run on the same clock frequency as the reset of the blocks. The PID block needs 10 steps to do one cycle. So, the maximum frequency of the PID block would be 5 MHz for a 50 MHz system clock frequency. In this case a frequency of 1 KHz is chosen for the PID block. To create this 1 KHz clock signal a clock divider block is used. The clock divider divides the original clock frequency by 5000, which results in a clock frequency of 1 KHz

Port description	**Port name**	**I/O**
System clock	Clk_in	Input
Reset	reset	Input
Divided clock signal	Clk_out	Output

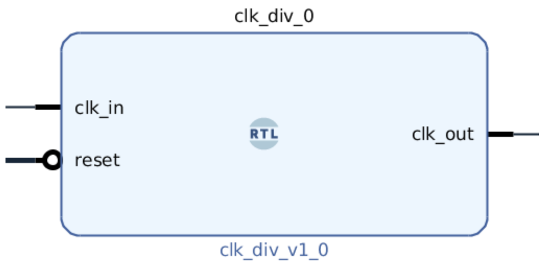


Figure 15 Clock divider entity and clock divider block

The clk_in is connected to the 50 MHz input clock, and the reset is connected to a constant LOW. Because the clock divider never has to be reset in this case. The output of the clock divider block is clk_out. This outputs the clock signal with a frequency of 1 KHz This is 1 KHz clock is connected to the clock input on the PID block.

In Figure 16 the simulation of the clock divider is shown. With an input 50 MHz the output should be 1 KHz. This means the output should have a period of 1000 ns.

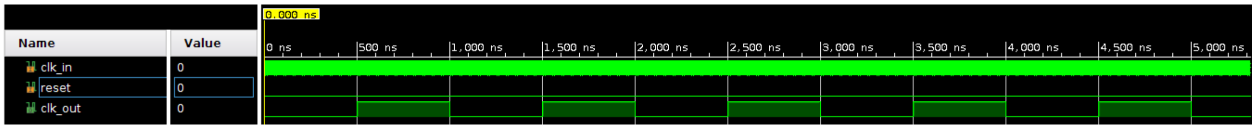


Figure 16 Simulation clock divider

3.4. PWM generation

Resource: https://www.digikey.com/eewiki/pages/viewpage.action?pageId=20939345

In the PWM generation block takes care of the pulse width modulation generation in order to control the H-bridge. With a variable input for the duty cycle the amount of current is variated on the H-bridge. The duty cycle is determined by the amount high signal in one period of the PWM signal. The H-bridge needs a 2 kHz PWM signal. The PWM-block calculates the amount of system clock pulses in the requested duty cycle of one PWM period. With the duty cycle ratio, the counts of high signal in one PWM is calculated. This is developed in VHDL code.

Port description	**Port name**	**I/O**
System clock	clk	Input
Asynchronous reset	reset_n	Input
Latches in new duty cycle	ena	Input
Duty cycle	duty	Input
PWM output	pwm_out	Output
PWM inverse output	pwm_n_out	Output

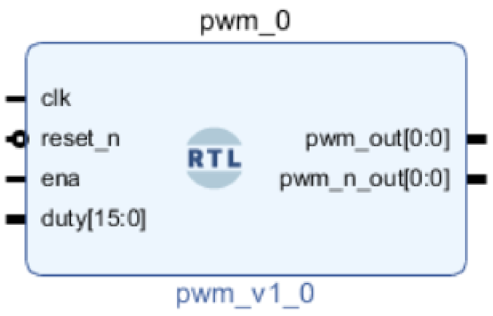


Table 4 PWM entity description with PWM block design

To validate the function of the PWM generation, a simulation is created. In Figure 17 and Figure 18 there are two circumstances visualised. In (1) the input duty cycle is higher than in (2). As a result, the output PWM in (1) has a more ratio high signal in one period.

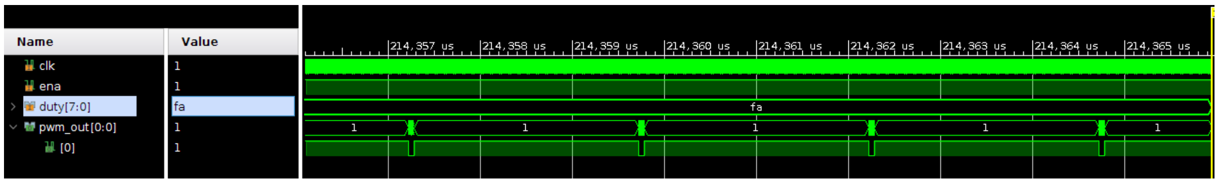


Figure 17 Testing PWM-block (1)

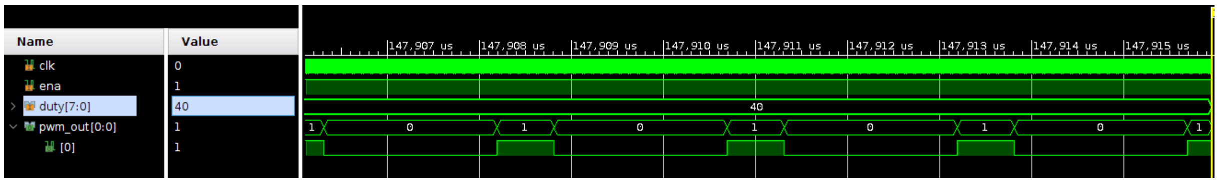


Figure 18 Testing PWM-block (2)

The PWM generation works in the simulation. In Figure 19 the plot of the oscilloscope from the ports of the PWM signal are visible. The oscilloscope indicates that the frequency is the required 2000 kHz. This means the PWM generation works.

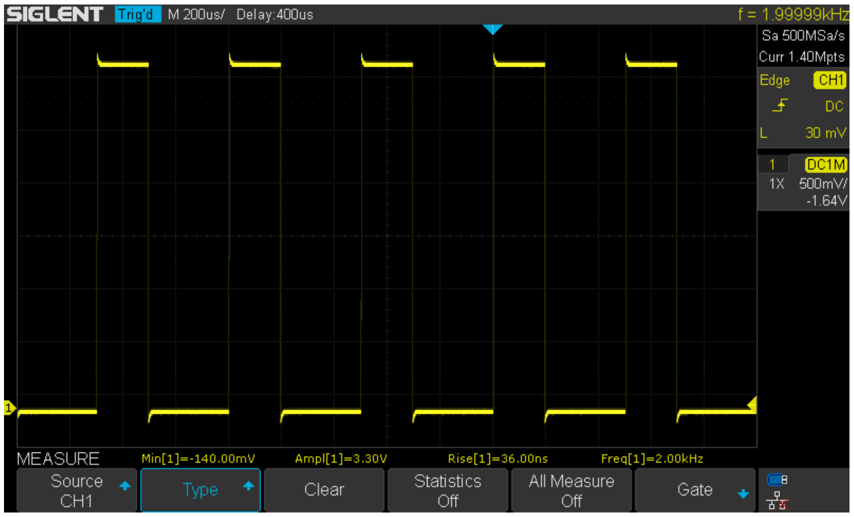


Figure 19 Test PWM generation

3.5. Processing system

For the processing system it's chosen to use both ARM processors of the Zynq chip. This is done because validation is done by printing values over to a laptop via UART. UART slows the program down massively, so the printing over UART is done by a separate ARM processor. In Figure 20 the architecture of the processing system is visible.

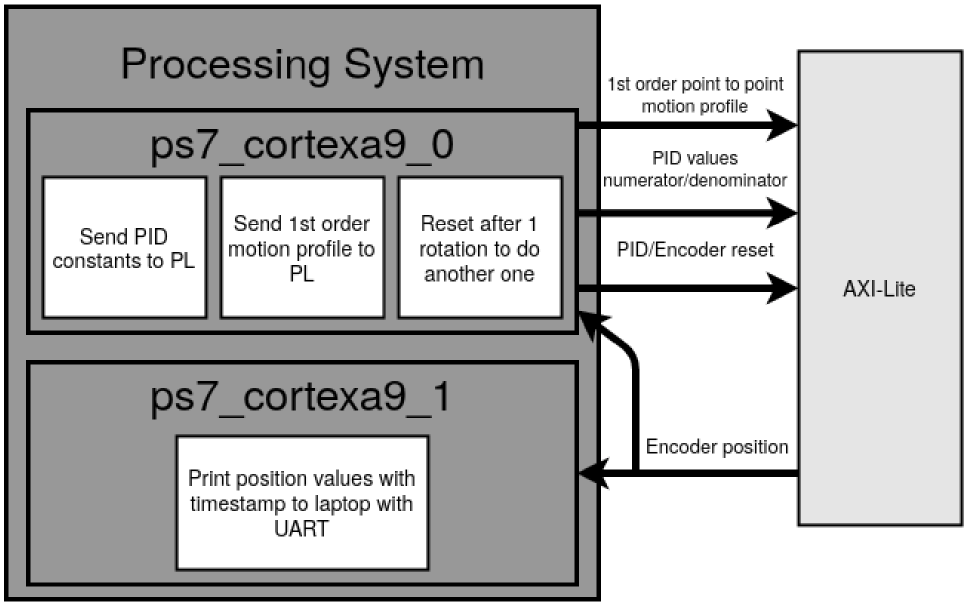

Processing System
ps7_cortexa9_0
Send PID constants to PL
Send 1st order motion profile to PL
Reset after 1 rotation to do another one
ps7_cortexa9_1
Print position values with timestamp to laptop with UART
1st order point to point motion profile
PID values numerator/denominator
PID/Encoder reset
Encoder position
AXI-Lite

Figure 20 Architecture processing system

3.5.1. Processor 1

The first ARM processor of the Zynq chip is used to send the control signals to the PL. This includes the 1st order motion profile, PID constants and the reset signals for the PID block and encoder block. The 1st order motion profile is generated by sending an increasing value, that counts to 2000 counts, to the PL as setpoint for the PID controller.

The PID constants are sent to the PL as a numerator and denominator. In this way it's possible to use floating point values. The numerator and denominator will be converted in a floating-point value inside the PL.

In Figure 21 the flowchart of the program is shown.

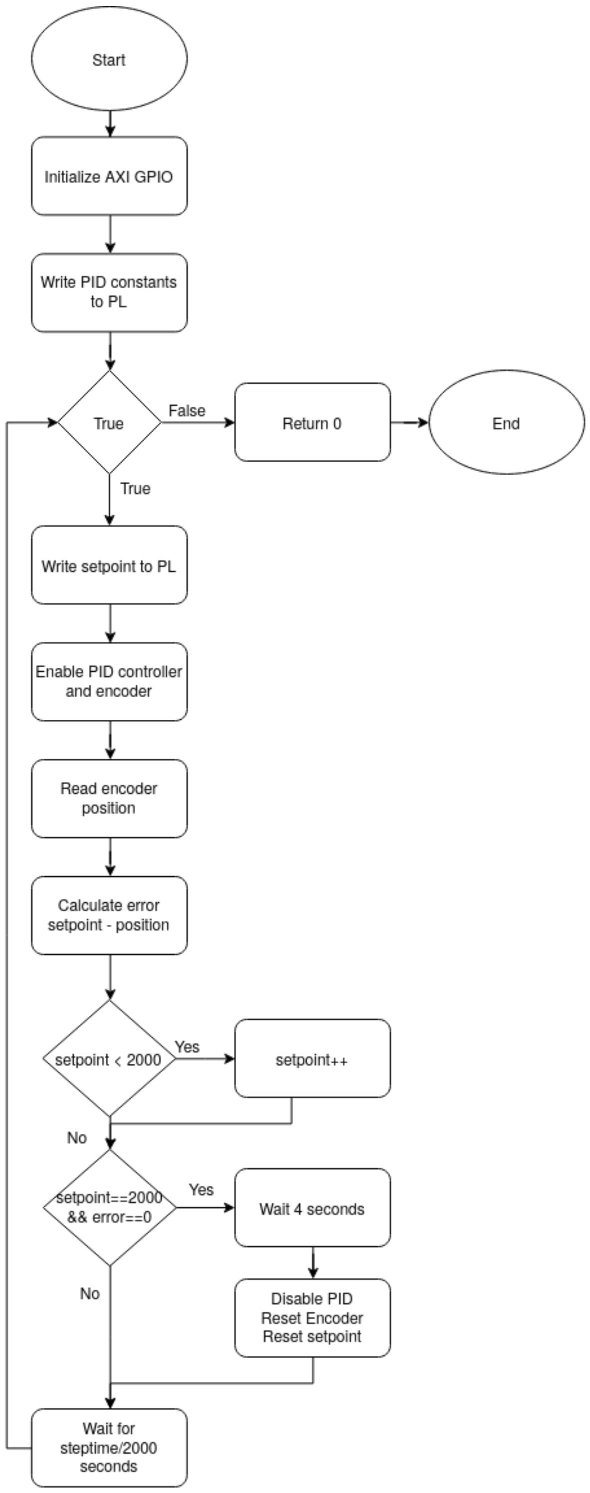

Start
Initialize AXI GPIO
Write PID constants to PL
True
False
Return 0
End
True
Write setpoint to PL
Enable PID controller and encoder
Read encoder position
Calculate error setpoint - position
setpoint < 2000
Yes
setpoint++
No
setpoint==2000 && error==0
Yes
Wait 4 seconds
Disable PID Reset Encoder Reset setpoint
No
Wait for steptime/2000 seconds

Figure 21 Flowchart processor 1

3.5.2. Processor 2

The second processor takes care of the printing values for the verification. This decision is made because of the delay causes by the printing function trough UART. This influences controller in a way the demanded accuracy is not achieved. Based on the flowchart in Figure 22 a code is created in the software development kit.

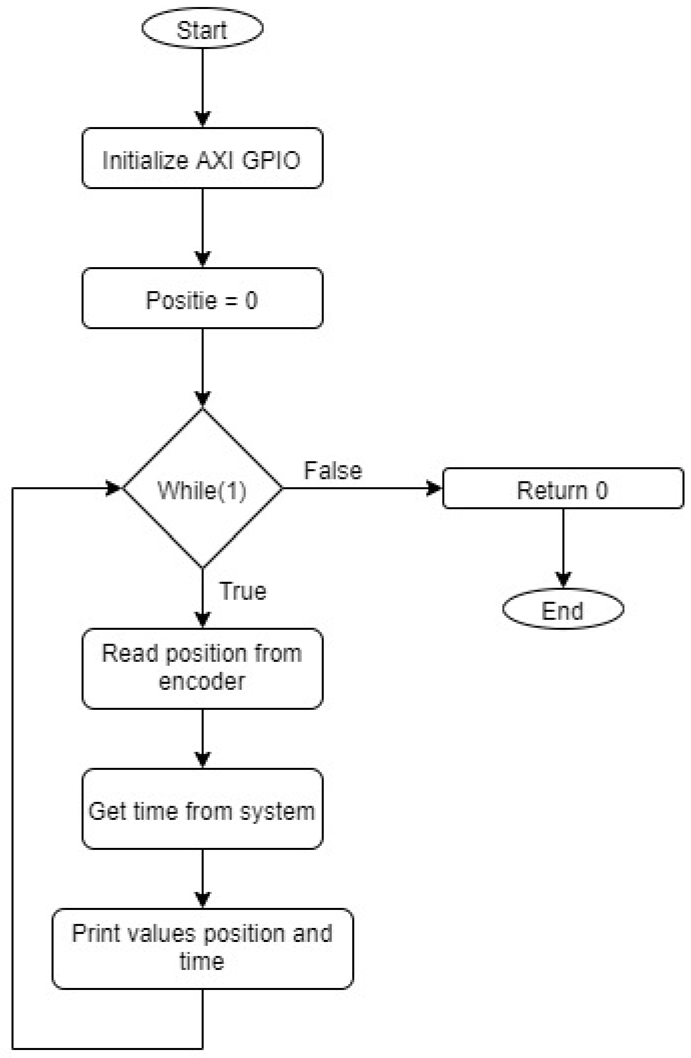


Figure 22 Flowchart processor 2

When programming the FPGA, the UART communication is tested. With opening a monitor in the software development kit, the values will be printed through the UART monitor.

3.6. Integration of the system

The integration of all parts is done with a block diagram. All the parts come together in the block diagram. In this block diagram all the connections between each component are made. Also, the connections to the physical ports of the ZYBO are done in the block diagram in combination with a constraints file. The constraint file connects the input and output ports in the block diagram to the physical output ports of the ZYBO board. In Figure 23 the block diagram of the integrated system is shown.

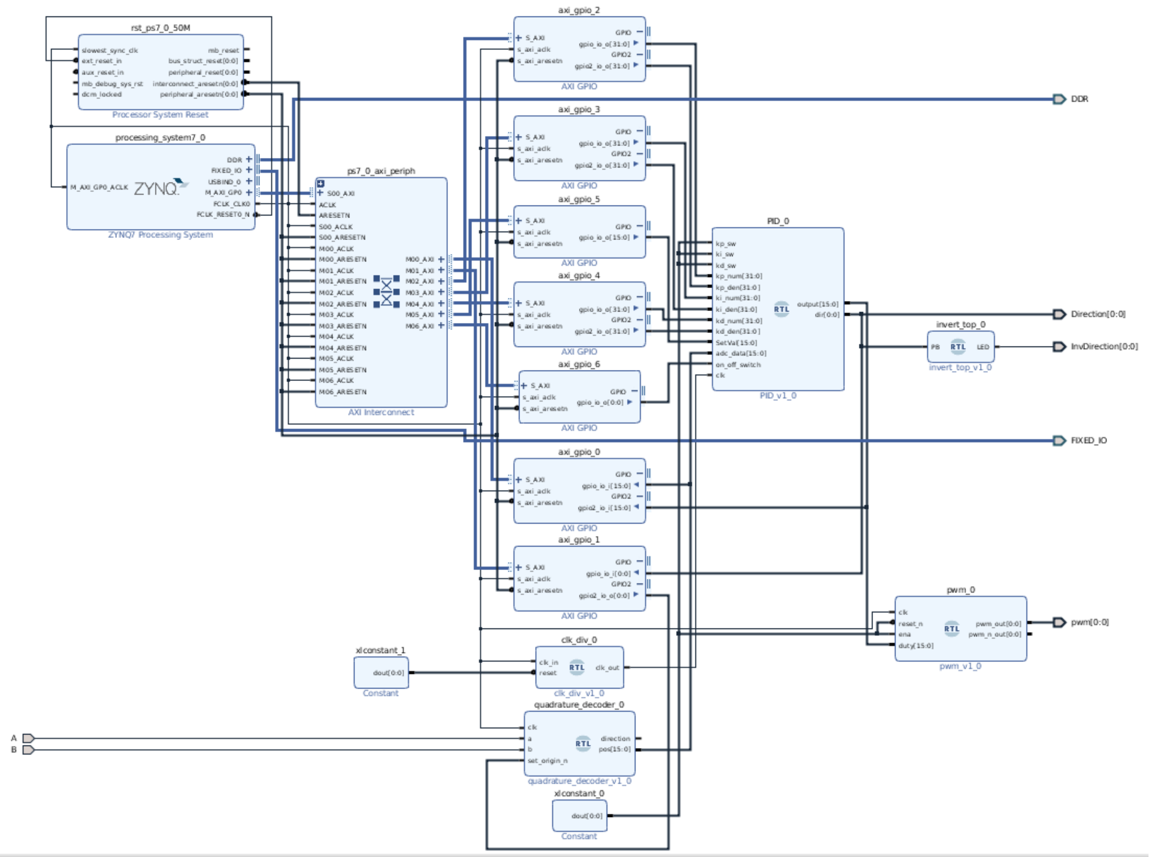


Figure 23 Block design integrated system

With the processing system linked to the programmable logic, the values of the encoder are printed trough UART. Furthermore, the total integration is tested with printing values trough UART. Therefore, no simulation is needed. The entity is described in Table 5. The only in and outputs are for the encoder (input) and H-bridge (output).

Port description	**Port name**	**I/O**
Encoder signal A	A	Input
Encoder signal B	B	Input
Pwm signal for H-bridge	Pwm	Output
Direction for H-bridge	direction	Output
Direction for H-bridge	invdirection	Output

Table 5 Integrated system entity

The values of the PID controller can be changed in the processing system. The used constants are shown Table 6.

Kp numerator	Kp_num	Input	15
Kp denominator	Kp_den	Input	1
Ki numerator	Ki_num	Input	5
Ki denominator	Ki_den	Input	1000
Kd numerator	Kd_num	Input	200
Kd denominator	Kd_den	Input	1

Table 6 PID constants

4. Conclusion and recommendations

The goal of the project was defined with the following research question:

"How do we design a motion controller on the ZYBO Z7010 to control the Maxon RE25 DC motor, powered through the PhodHB3 H-bridge and with readings from the HEDL 5540 encoder, given the requirements that it shall make one full rotation, settled within 15 ms, with a maximum position error of ± 0.05 revolutions, and a maximum steady state error of ± 0.0005 revolutions."

Six weeks of development have yielded the following results. To control the motor the following IP blocks were developed.

Decoder - The decoder uses quadrature decoding to measure 2000 counts per turn in total, by using channel A and B with rising and falling edge detection. It produces a position signal of 16 bits, with a range of -4000 to 4000, which is sent to the PID controller.

PID controller – The PID controller calculates the error signal, by subtracting the position from the reference value (coming through AXI4 Lite from the Processing System) and performs the PID actions. The outcome represents a power signal, that is sent to the PWM generator. The block also has a direction output to control the direction of the motor (with the H-bridge).

PWM generator – The PWM generator uses the outcome of the PID controller as the duty cycle for the 16-bit PWM signal, which powers the motor.

Matlab was used to create a model of the DC motor and the motion controller. Both the bode plot and transfer function were found, and attempts were made to let Matlab automatically tune the PID constants (based on the requirements as stated above). Unfortunately, these values did not result in stable responses when applied on the system, and therefore the constants were determined one by one, by trial and error. This resulted in the following PID constants:

Table 7 PID constants

Kp numerator	Kp_num	Input	15
Kp denominator	Kp_den	Input	1
Ki numerator	Ki_num	Input	5
Ki denominator	Ki_den	Input	1000
Kd numerator	Kd_num	Input	200
Kd denominator	Kd_den	Input	1

With these constants the following results were achieved:

Table 8 Performance requirements results

Requirement	***Required***	***Achieved***
Settling time	15 ms	~ 62 ms
Maximum position error	*± 18 degrees*	20.7 degrees
Steady state error	*± 0.18 degrees*	0.0 degrees

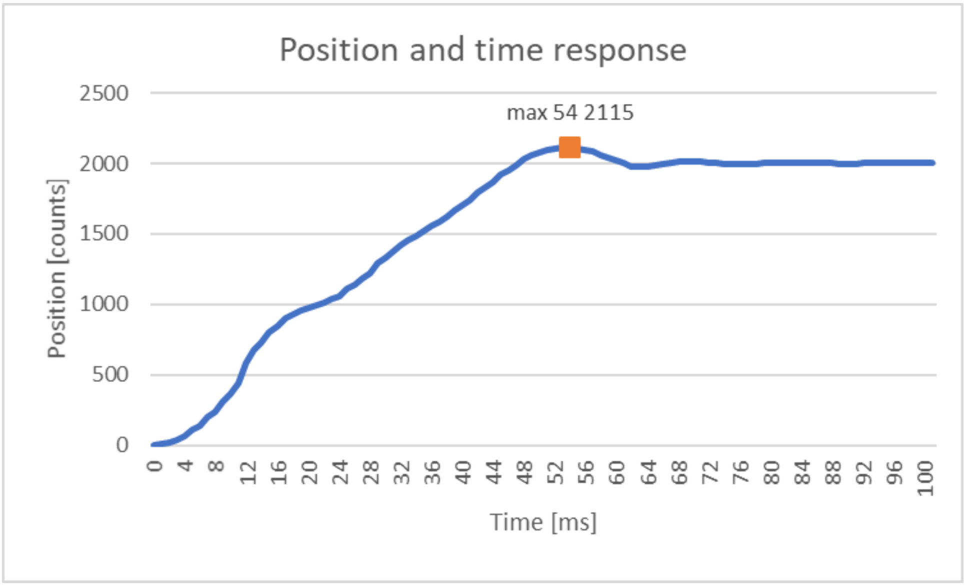


Figure 24 Position and time response

As seen in the tables and graph above the requirements were not entirely met. This is because the PID values were not tuned to optimal values. If the model-based design would have resulted in a model that represented the system better. The PID constant could have been tuned more precisely. Furthermore, the overshoot requirement could have been achieved if the system was set to control less aggressively. However, the system can perform a full and controlled rotation.